数理工学実験
テーマ２　常微分方程式の数値解法

2021年10月25日 提出

工学部情報学科数理工学コース２年
1029-32-7314　岡本淳志

課題1:	5/5
課題2:	5/5
課題3:	9/10
課題4:	20/20
課題5:	20/20
課題6:	20/20
課題7:	10/20
Σ	89/100

1　課題１

身の回りにある現象から一つ選び、常微分方程式の初期値問題を定式化する。私は天井からバネで吊るされた質点を選んだ。

天井に対して垂直な x 軸上を運動する、天井からバネ定数 k のバネで吊るされた質量 m の質点を考える。時間変数を t とし、質点の位置と速度をそれぞれ $x(t)$ および $v(t)$ とする。質点には速度に比例する摩擦力 $-cv(t)$ $(c > 0)$ が働く。また、初期位置と初期速度はそれぞれ $x(0) = x_0$ および $v(0) = v_0$ である。重力加速度を g とする。この質点の運動を表す常微分方程式と初期条件は

$$x' = v, \quad mv' = -kx + mg - cv, \tag{1}$$

$$x(0) = x_0, \quad v(0) = v_0 \tag{2}$$

で与えられる。すなわち、

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v, \quad m\frac{\mathrm{d}v}{\mathrm{d}t} = -kx + mg - cv, \tag{3}$$

$$x(0) = x_0, \quad v(0) = v_0 \tag{4}$$

である。

以上より、身の回りにある現象、「天井からバネで吊るされた質点」の常微分方程式の初期値問題を定式化できた。

2　課題２

4 次のアダムス・バッシュフォース法とアダムス・ムルトン法を構成する。また、4 次のアダムス・バッシュフォース法と 4 次のアダムス・ムルトン法を用いて、予測子・修正子法を構成する。

2.1　4 次のアダムス・バッシュフォース法

$t \in [0, T]$ で定義される 1 変数の常微分方程式を考える:

$$u' = f(t, u(t)), \quad u(0) = u_0 \tag{5}$$

この式を $t \in [t_{n-1}, t_n]$ の範囲で積分すると、

$$u_n - u_{n-1} = \int_{t_{n-1}}^{t_n} f(\tau, u(\tau))d\tau \tag{6}$$

となる。

ここで、式（6）の中の f を、τ に関する N 次多項式 $pN(\tau)$ で近似する。（$pN(\tau)$ を Lagrange 補間により構成する。）

$$\int_{t_{n-1}}^{t_n} f(\tau, u(\tau))d\tau = \int_{t_{n-1}}^{t_n} pN(\tau)d\tau \tag{7}$$

いま、4 次のアダムス・バッシュフォース法を構成したいので、これには 4 個の点が必要である。(t_{n-1}, f_{n-1}), (t_{n-2}, f_{n-2}), (t_{n-3}, f_{n-3}), (t_{n-4}, f_{n-4}) を用いて、

$$\begin{aligned} f(\tau, u(\tau)) = pN(\tau) \approx \quad & \frac{\tau - t_{n-3}}{t_{n-4} - t_{n-3}} \frac{\tau - t_{n-2}}{t_{n-4} - t_{n-2}} \frac{\tau - t_{n-1}}{t_{n-4} - t_{n-1}} f_{n-4} \\ & + \frac{\tau - t_{n-4}}{t_{n-3} - t_{n-4}} \frac{\tau - t_{n-2}}{t_{n-3} - t_{n-2}} \frac{\tau - t_{n-1}}{t_{n-3} - t_{n-1}} f_{n-3} \\ & + \frac{\tau - t_{n-4}}{t_{n-2} - t_{n-4}} \frac{\tau - t_{n-3}}{t_{n-2} - t_{n-3}} \frac{\tau - t_{n-1}}{t_{n-2} - t_{n-1}} f_{n-2} \\ & + \frac{\tau - t_{n-4}}{t_{n-1} - t_{n-4}} \frac{\tau - t_{n-3}}{t_{n-1} - t_{n-3}} \frac{\tau - t_{n-2}}{t_{n-1} - t_{n-2}} f_{n-1} \end{aligned} \tag{8}$$

と表すことができる。$t_{n-i} - t_{n-j} = (j-i)\Delta t$ であることを考えて、式（7）に代入すると、

$$\begin{aligned} \int_{t_{n-1}}^{t_n} f(\tau, u(\tau))d\tau &= \int_{t_{n-1}}^{t_n} pN(\tau)d\tau \\ &= -\frac{f_{n-4}}{6(\Delta t)^3} \int_{t_{n-1}}^{t_n} (\tau - t_{n-3})(\tau - t_{n-2})(\tau - t_{n-1})d\tau \\ &\quad + \frac{f_{n-3}}{2(\Delta t)^3} \int_{t_{n-1}}^{t_n} (\tau - t_{n-4})(\tau - t_{n-2})(\tau - t_{n-1})d\tau \\ &\quad - \frac{f_{n-2}}{2(\Delta t)^3} \int_{t_{n-1}}^{t_n} (\tau - t_{n-4})(\tau - t_{n-3})(\tau - t_{n-1})d\tau \\ &\quad + \frac{f_{n-1}}{6(\Delta t)^3} \int_{t_{n-1}}^{t_n} (\tau - t_{n-4})(\tau - t_{n-3})(\tau - t_{n-2})d\tau \end{aligned} \tag{9}$$

それぞれ展開して積分を計算する。式（6）より、

$$u_n \approx u_{n-1} + \frac{\Delta t}{24}(55f_{n-1} - 59f_{n-2} + 37f_{n-3} - 9f_{n-4}) \tag{10}$$

これが、4 次のアダムス・バッシュフォース法である。

2.2　4 次のアダムス・ムルトン法

先述の、4 次のアダムス・バッシュフォース法と同様に考えるが、ここでは (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。

$$\begin{aligned} f(\tau, u(\tau)) = pN(\tau) \approx \quad & \frac{\tau - t_{n-2}}{t_{n-3} - t_{n-2}} \frac{\tau - t_{n-1}}{t_{n-3} - t_{n-1}} \frac{\tau - t_n}{t_{n-3} - t_n} f_{n-3} \\ & + \frac{\tau - t_{n-3}}{t_{n-2} - t_{n-3}} \frac{\tau - t_{n-1}}{t_{n-2} - t_{n-1}} \frac{\tau - t_n}{t_{n-2} - t_n} f_{n-2} \\ & + \frac{\tau - t_{n-3}}{t_{n-1} - t_{n-3}} \frac{\tau - t_{n-2}}{t_{n-1} - t_{n-2}} \frac{\tau - t_n}{t_{n-1} - t_n} f_{n-1} \\ & + \frac{\tau - t_{n-3}}{t_n - t_{n-3}} \frac{\tau - t_{n-2}}{t_n - t_{n-2}} \frac{\tau - t_{n-1}}{t_n - t_{n-1}} f_n \end{aligned} \tag{11}$$

$t_{n-i} - t_{n-j} = (j-i)\Delta t$ であることを考えて、式（7）に代入し、積分を計算する。式（6）より、

$$u_n \approx u_{n-1} + \frac{\Delta t}{24}(9f_n + 19f_{n-1} - 5f_{n-2} + f_{n-3}) \tag{12}$$

これが、4 次のアダムス・ムルトン法である。

2.3　予測子・修正子法

4 次のアダムス・バッシュフォース法により、u_n の予測値 $\widetilde{u_n}$ は、

$$\widetilde{u_n} = u_{n-1} + \frac{\Delta t}{24}(55f_{n-1} - 59f_{n-2} + 37f_{n-3} - 9f_{n-4}) \tag{13}$$

となる。これを用いて $\widetilde{f_n}$ を計算すると、

$$\widetilde{f_n} = \frac{\widetilde{u_n} - u_{n-1}}{\Delta t} = \frac{1}{24}(55f_{n-1} - 59f_{n-2} + 37f_{n-3} - 9f_{n-4}) \tag{14}$$

が得られる。これを式（12）の右辺に代入する。

$$u_n \approx u_{n-1} + \frac{\Delta t}{192}(317f_{n-1} - 217f_{n-2} + 119f_{n-3} - 27f_{n-4}) \tag{15}$$

これが、u_n の修正値である。

3　課題３

常微分方程式の初期値問題

$$u' = u, \quad u(0) = 1 \tag{16}$$

を考える。これを $t \in [0, 1]$ の範囲で数値的に解いて、その精度を確認する。

まず、ステップ幅を $\Delta t = 1/2^i$、ステップ数を $N = 2^i$、離散化した時間変数を $t_n = n\Delta t \quad (n = 0, ..., N)$ と定義する。また、ステップ幅が $\Delta t = 1/2^i$ の条件の下で得られた数値解を $u_n^{(i)}$ と表す。

精度の確認は、i の値を $i = 1, 2, ..., 10$ の範囲で様々に変更した計算を実行し、それらの結果を比較する。比較には、$t = 1$ における数値解 $u_N^{(i)}$ と解析解との差 $E^{(i)} = |u_N^{(i)} - u(1)|$ を用いる。

p を計算手法の次数とし、$E_r^{(i)} = E^{(i)}/E^{(i-1)} \quad (i = 2, ..., 10)$ と定義すると、$E^{(i)}$ は i の増加とともに $1/2^p$ に近づくはずである。

前進オイラー法 (FE), 2 次および 3 次のアダムス・バッシュフォース法 (AB2 および AB3), ホイン法 (H), 4 次のルンゲ・クッタ法 (RK) の 5 種類の方法で、式 (16) を $t \in [0, 1]$ の範囲で数値的に解いた。多段法で必要となる初期条件に関しては、解析解 $u = exp(t)$ から値を求めた。用いたコードを、ソースコード 1 に示す。

ソースコード 1: C 言語を用いて常微分方程式の初期値問題を数値的に解くプログラム

```
1  //数理工学実験テーマ２ //課題３
2  #include <stdio.h>
3  #include <math.h>
4  int main(int argc, char **argv)
5  {
6      int i;
7      int N; //ステップ数
8      double deltaT; //ステップ幅
9
10     double U0 = exp(0); //u0 の値
11
12     double Un, preUn, prepreUn, preprepreUn; //それぞれ、u_n ~
           u_(n-3)
13
14     double UN[5] = {0, 1, 2, 3, 4}; //数値解法ごとのu_N
15
16     double e = exp(1.0); //u(1)の解析解
17
18     double E[5] = {0, 1, 2, 3, 4}; //数値解法ごとの、数値解
           と解析解との差の絶対値
19     double preE[5] = {0, 1, 2, 3, 4};
20
21     double Er[5] = {0, 1, 2, 3, 4}; //=E/preE
22
23     for (int i = 1; i <= 10; i++) //i=1, 2, ... , 10
24     {
25         N = pow(2, i); //ステップ数
26         deltaT = 1.0 / N; //ステップ幅
```

```
27
28          double U1 = exp(deltaT); //u1 の値
29          double U2 = exp(2 * deltaT); //u2 の値
30
31          printf("\n");
32          printf("i=%d のとき (FE, AB2, AB3, H, RK の順)", i);
33          printf("\n");
34
35          //前進オイラー法ここから
36          preUn = U0;
37
38          for (int k = 0; k < N; k++)
39          {
40              Un = preUn + preUn * deltaT; //前進オイラー法
41
42              preUn = Un; //次の試行のための代入
43          }
44          UN[0] = Un; //UN に u_N の値を代入
45          //前進オイラー法ここまで
46
47          //2次のアダムス・バッシュフォース法ここから
48          prepreUn = U0;
49          preUn = U1;
50
51          for (int l = 0; l < N - 1; l++)
52          {
53              Un = preUn + deltaT / 2 * (3 * preUn - prepreUn
                    ); //2次のアダムス・バッシュフォース法
54
55              prepreUn = preUn; //次の試行のための代入。順番に注
                    意。
56              preUn = Un;
57          }
58          UN[1] = Un; //UN に u_N の値を代入
59          //2次のアダムス・バッシュフォース法ここまで
60
61          //3次のアダムス・バッシュフォース法ここから
62          preprepreUn = U0;
63          prepreUn = U1;
64          preUn = U2;
65
66          for (int m = 0; m < N - 2; m++)
67          {
68              Un = preUn + deltaT / 12 * (23 * preUn - 16 *
                    prepreUn + 5 * preprepreUn); //3次のアダムス・
                    バッシュフォース法
```

```
69
70              preprepreUn = prepreUn; //次の試行のための代入。順
                    番に注意。
71              prepreUn = preUn;
72              preUn = Un;
73          }
74          UN[2] = Un; //UN に u_N の値を代入
75          //3次のアダムス・バッシュフォース法ここまで
76
77          //ホイン法(2次のルンゲ・クッタ法)ここから
78          preUn = U0;
79          double temporaryUn; //un*
80
81          for (int p = 0; p < N; p++)
82          {
83              temporaryUn = preUn + preUn * deltaT;
84              Un = preUn + deltaT / 2 * (preUn + temporaryUn);
                    //ホイン法(2次のルンゲ・クッタ法)
85
86              preUn = Un; //次の試行のための代入
87          }
88          UN[3] = Un; //UN に u_N の値を代入
89          //ホイン法(2次のルンゲ・クッタ法)ここまで
90
91          //(4次の)ルンゲ・クッタ法ここから
92          preUn = U0;
93          double F1, F2, F3, F4; //4次のルンゲ・クッタ法で用いら
                れているF1~F4
94
95          for (int q = 0; q < N; q++)
96          {
97              F1 = preUn;
98              F2 = preUn + F1 * deltaT / 2;
99              F3 = preUn + F2 * deltaT / 2;
100             F4 = preUn + F3 * deltaT;
101             Un = preUn + deltaT / 6 * (F1 + 2 * F2 + 2 * F3
                    + F4); //(4次の)ルンゲ・クッタ法
102
103             preUn = Un; //次の試行のための代入
104         }
105         UN[4] = Un; //UN に u_N の値を代入
106         //(4次の)ルンゲ・クッタ法ここまで
107
108         for (int c = 0; c < 5; c++)
109         {
110             E[c] = fabs(UN[c] - e); //数値解と解析解との差の
```

絶対値

```
111
112             printf("uN=%.15lf\n", UN[c]);
113             printf("E%d=%.15lf\n", c, E[c]);
114
115             if (i >= 2)
116             {
117                 Er[c] = E[c] / preE[c];
118                 printf("Er%c=%lf\n", c, Er[c]);
119             }
120             printf("\n");
121             preE[c] = E[c]; //E(i)をE(i-1)として保存
122         }
123     }
124
125     return 0;
126 }
```

プログラムの実行結果に基づき、以下の表１～１０と表１１を作成した。

表 1: i=1 のとき

数値解法	$u_N^{(1)}$	$E^{(1)}$
FE	2.250000000000000	0.468281828459045
AB2	2.635262223725224	0.083019604733821
AB3	2.635262223725224	0.083019604733821
H	2.640625000000000	0.077656828459045
RK	2.717346191406250	0.000935637052795

表 2: i=2 のとき

数値解法	$u_N^{(2)}$	$E^{(2)}$
FE	2.441406250000000	0.276875578459045
AB2	2.675877648973323	0.042404179485722
AB3	2.712456257096107	0.005825571362938
H	2.694855690002441	0.023426138456604
RK	2.718209939201323	0.000071889257722

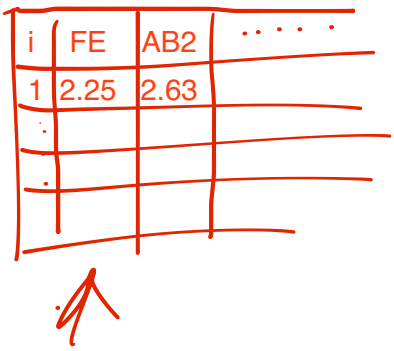

表11のように一つの表で結果をまとめばよかったが。

テキストには探さず、表のキャプションから内容が理解できるように書くこと。

表 3: i=3 のとき

数値解法	$u_N^{(3)}$	$E^{(3)}$
FE	2.565784513950348	0.152497314508697
AB2	2.704215823029824	0.014066005429221
AB3	2.716982169105137	0.001299659353908
H	2.711841238551985	0.006440589907060
RK	2.718276844416734	0.000004984042311

表 4: i=4 のとき

数値解法	$u_N^{(4)}$	$E^{(4)}$
FE	2.637928497366600	0.080353331092446
AB2	2.714309042568377	0.003972785890668
AB3	2.718078299522280	0.000203528936765
H	2.716593522474767	0.001688305984278
RK	2.718281500340586	0.000000328118460

表 5: i=5 のとき

数値解法	$u_N^{(5)}$	$E^{(5)}$
FE	2.676990129378183	0.041291699080862
AB2	2.717231740218152	0.001050088240893
AB3	2.718253626501579	0.000028201957466
H	2.717849673980259	0.000432154478786
RK	2.718281807411193	0.000000021047852

表 6: i=6 のとき

数値解法	$u_N^{(6)}$	$E^{(6)}$
FE	2.697344952565100	0.020936875893945
AB2	2.718012269614910	0.000269558844135
AB3	2.718278123972088	0.000003704486957
H	2.718172511563830	0.000109316895216
RK	2.718281827126323	0.000000001332722

表 7: i=7 のとき

数値解法	$u_N^{(7)}$	$E^{(7)}$
FE	2.707739019688019	0.010542808771026
AB2	2.718213566094684	0.000068262364362
AB3	2.718281353972686	0.000000474486360
H	2.718254338321275	0.000027490137770
RK	2.718281828375204	0.000000000083841

表 8: i=8 のとき

数値解法	$u_N^{(8)}$	$E^{(8)}$
FE	2.712991624253433	0.005290204205612
AB2	2.718264654294481	0.000017174164564
AB3	2.718281768426940	0.000000060032106
H	2.718274935740745	0.000006892718300
RK	2.718281828453784	0.000000000005261

表 9: i=9 のとき

数値解法	$u_N^{(9)}$	$E^{(9)}$
FE	2.715632000168990	0.002649828290055
AB2	2.718277521380348	0.000004307078697
AB3	2.718281820909723	0.000000007549322
H	2.718280102752167	0.000001725706878
RK	2.718281828458716	0.000000000000329

表 10: i=10 のとき

数値解法	$u_N^{(10)}$	$E^{(10)}$
FE	2.716955729466436	0.001326098992609
AB2	2.718280749999386	0.000001078459659
AB3	2.718281827512534	0.000000000946511
H	2.718281396716139	0.000000431742906
RK	2.718281828459026	0.000000000000020

表 11: 数値解法別の $E_r^{(i)}$

	FE	AB2	AB3	H	RK
$E_r^{(2)}$	0.591258	0.510773	0.070171	0.301662	0.076835
$E_r^{(3)}$	0.550779	0.331713	0.223096	0.274932	0.069329
$E_r^{(4)}$	0.526916	0.282439	0.156602	0.262135	0.065834
$E_r^{(5)}$	0.513877	0.264320	0.138565	0.255969	0.064147
$E_r^{(6)}$	0.507048	0.256701	0.131356	0.252958	0.063319
$E_r^{(7)}$	0.503552	0.253237	0.128084	0.251472	0.062910
$E_r^{(8)}$	0.501783	0.251591	0.126520	0.250734	0.062746
$E_r^{(9)}$	0.500893	0.250788	0.125755	0.250367	0.062468
$E_r^{(10)}$	0.500447	0.250392	0.125377	0.250183	0.059459

表１～１０より、i が大きくなるにつれて数値解と解析解の差 $E^{(i)}$ が小さくなっている。つまり、ステップ幅が小さければ小さいほど、数値解析の精度が高まることが分かる。また、同じステップ幅でも、次数精度が向上すれば数値解析の精度が高まる。同じ次数精度のもの、ここでは AB2 と H では、高次精度一段法であるホイン法の方が数値解析の精度が高い。

前進オイラー法 (FE) は 1 次 ($p = 1$) であることと、ホイン法 (H) は 2 次 ($p = 2$) であることを考えると、表１１より、FE, AB2, AB3, H は先述の、「p を計算手法の次数とし、$E_r^{(i)} = E^{(i)}/E^{(i-1)} \quad (i = 2, ..., 10)$ と定義すると、$E^{(i)}$ は i の増加とともに $1/2^p$ に近づくはずである。」を満たしていると言える。

本来、4 次のルンゲ･クッタ法の $E_r^{(i)}$ は理論上 i の増加とともに $1/2^4 = 0.0625$ に近づくはずであるが、今回の数値実験においてその様子が正確には見られなかった。原因としては、倍数制度でも i が大きくなると結果を正しく評価できないからだと考えられる。

4 課題４

計算手法の安定性を確認するためには、$\lim_{t\to\infty} |u(t)| = \infty$ とならなければよい。差分式が以下の形で書けるとし、

$$u_n = a_1 u_{n-1} + a_0 \tag{17}$$

特定方程式を解くことにより u_n の解を求めると、

$$u_n = (u_0 - \frac{a_0}{1-a_1})a_1^n + \frac{a_0}{1-a_1} \tag{18}$$

となる。これより、式（18）が $n \to \infty$ で発散しないためには、$|a_1| \leq 1$ であればよい。

$t \in [0, \infty)$ における常微分方程式の初期値問題

$$u' = -\alpha u + \beta, \quad u(0) = u_0, \quad (\alpha > 0) \tag{19}$$

に対して、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場合に安定性がどのように変わるかを数値的に確認した。

4.1 クランク・ニコルソン法

クランク・ニコルソン法において、

$$u_n \approx u_{n-1} + \frac{\Delta t}{2}(f_{n-1} + f_n) \tag{20}$$

である。式（19）をクランク・ニコルソン法で離散化すると、

$$\begin{aligned} u_n &= u_{n-1} + \frac{\Delta t}{2}((-\alpha u_{n-1} + \beta) + (-\alpha u_n + \beta)) \\ &= \frac{2 - \alpha\Delta t}{2 + \alpha\Delta t} u_{n-1} + \frac{2\beta\Delta t}{2 + \alpha\Delta t} \end{aligned} \tag{21}$$

が得られるので、$a_1 = \frac{2-\alpha\Delta t}{2+\alpha\Delta t}$, $a_0 = \frac{2\beta\Delta t}{2+\alpha\Delta t}$ である。

$$\begin{aligned} |a_1| = |\frac{2 - \alpha\Delta t}{2 + \alpha\Delta t}| &\leq 1 \\ 0 &\leq \alpha\Delta t \end{aligned} \tag{22}$$

より、$\alpha > 0$ に対して、クランク・ニコルソン法では常に $|a_1| < 1$ が満たされているため、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。

このことを以下で数値的に確認した。用いたソースコードを、ソースコード 2 に示す。

ソースコード 2: C 言語を用いてクランク・ニコルソン法の安定性を確認するプログラム

```
1  //数理工学実験テーマ２ //課題４ //クランク･ニコルソン法
2  #include <stdio.h>
3  #include <math.h>
4  int main(int argc, char **argv)
5  {
6      double U0 = 1.0; //u_0 の値
7      double alpha = 10.0; //αの値
8      double beta = 1.0; //βの値
9
10     double deltaT; //ステップ幅
11
12     double Un, preUn; //それぞれ､u_n, u_(n-1)
13
```

```
14      for (int i = 0; i < 3; i++) //i = 0.01, 0.1, 1.0
15      {
16          deltaT = 0.01 * pow(10, i);
17
18          printf("\n");
19          printf("Δt=%lf のとき\n", deltaT);
20
21          preUn = U0;
22          printf("%lf %lf \n", 0.0, preUn);
23
24          for (int j = 1; j <= 10 / deltaT; j++)
25          {
26              Un = (2 - alpha * deltaT) / (2 + alpha * deltaT)
                    * preUn + 2 * beta * deltaT / (2 + alpha *
                   deltaT); //クランク･ニコルソン法
27
28              preUn = Un; //次の試行のための代入
29
30              printf("%lf %lf \n", deltaT * j, Un);
31          }
32      }
33      return 0;
34  }
```

プログラムの実行結果に基づき、以下の図を作成した。

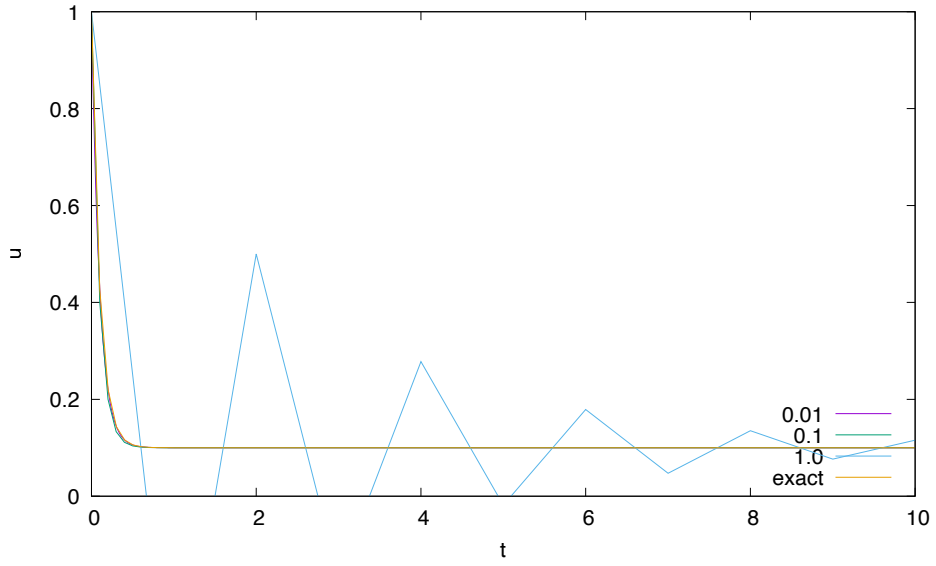

図 1: クランク・ニコルソン法による式（19）の数値解

この図より、解析通りクランク・ニコルソン法は任意の $\Delta t > 0$ に対して安定であると言える。

4.2 予測子・修正子法（AB2 & AM2）

2 次のアダムス・バッシュフォース法 $u_n = u_{n-1} + \frac{\Delta t}{2}(3f_{n-1} - f_{n-2})$ と 2 次のアダムス・ムルトン法 $u_n = u_{n-1} + \frac{\Delta t}{2}(f_n + f_{n-1})$ を用いて、予測子・修正子法を構成すると、

$$u_n = u_{n-1} + \frac{\Delta t}{4}(5f_{n-1} - f_{n-2}) \tag{23}$$

となる。

この式を用いて予測子・修正子法（AB2 & AM2）の安定性を数値的に確認した。用いたソースコードを、ソースコード 3 に示す。

ソースコード 3: C 言語を用いて予測子・修正子法（AB2 & AM2）の安定性を確認するプログラム

```
1  //数理工学実験テーマ２ //課題４ //予測子･修飾子法２
2  #include <stdio.h>
3  #include <math.h>
4  int main(int argc, char **argv)
5  {
6      double U0 = 1.0; //u_0 の値
7      double alpha = 10.0; //αの値
8      double beta = 1.0; //βの値
9
10     double deltaT = 0.1; //ステップ幅 0.36, 0.37あたりから発
           散?? → 0.1, 0.3, 0.4 で確認
11
12     double Un, preUn, pre2Un;
13     double preFn, pre2Fn;
14
15     //printf("\n");
16     printf("Δt=%lf のとき\n", deltaT);
17
18     pre2Un = U0;
19     preUn = (U0 - beta / alpha) * exp(-alpha * deltaT) +
           beta / alpha; //解析解から得たu_1 の値
20
21     printf("%lf %lf \n", 0.0, pre2Un);
22     printf("%lf %lf \n", deltaT, preUn);
23
24     for (int i = 2; i <= 10 / deltaT; i++)
25     {
26         pre2Fn = (-alpha * pre2Un + beta) * deltaT;
```

```
27          preFn = (-alpha * preUn + beta) * deltaT;
28
29          Un = preUn + deltaT / 4 * (5 * preFn - pre2Fn); //予
                測子･修飾子法２
30
31          pre2Un = preUn; //次の試行のための代入
32          preUn = Un;
33
34          printf("%lf %lf \n", deltaT * i, Un);
35      }
36
37      return 0;
38  }
```

プログラムの実行結果に基づき、以下の図を作成した。(ステップ幅 Δt の値を、0.1, 0.3, 0.4 にして実行した。)

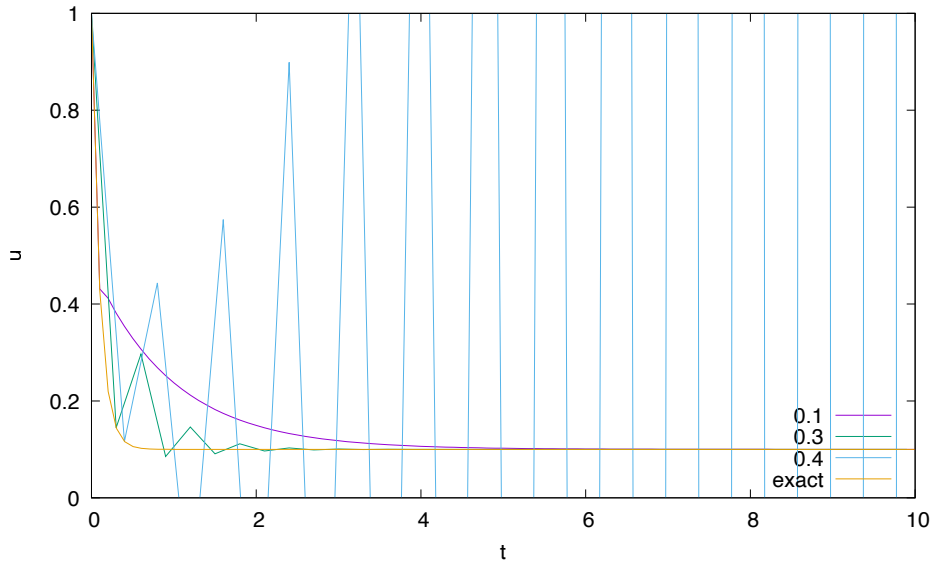

図 2: 予測子・修正子法（AB2 & AM3）による式（19）の数値解

この図より、予測子・修正子法（AB2 & AM3）で、$\Delta t = 0.4$ の場合に、解が振動しながら発散していく様子が分かる。（もう少し詳しく調べると、$0.36 < \Delta t < 0.37$ の間で発散し始めることがわかった。）

4.3　予測子・修正子法（AB3 & AM3）

3 次のアダムス・バッシュフォース法 $u_n = u_{n-1} + \frac{\Delta t}{12}(23f_{n-1} - 16f_{n-2} + 5f_{n-3})$ と 3 次のアダムス・ムルトン法 $u_n = u_{n-1} + \frac{\Delta t}{12}(5f_n + 8f_{n-1} - f_{n-2})$

を用いて、予測子・修正子法を構成すると、

$$u_n = u_{n-1} + \frac{\Delta t}{144}(211 f_{n-1} - 92 f_{n-2} + 25 f_{n-3}) \tag{24}$$

となる。

この式ををを用いて予測子・修正子法（AB3 & AM3）の安定性を数値的に確認した。用いたソースコードを、ソースコード 3 に示す。

ソースコード 4: C 言語を用いて予測子・修正子法（AB3 & AM3）の安定性を確認するプログラム

```
1  //数理工学実験テーマ２ //課題４ //予測子･修飾子法３
2  #include <stdio.h>
3  #include <math.h>
4  int main(int argc, char **argv)
5  {
6      double U0 = 1.0; //u_0 の値
7      double alpha = 10.0; //αの値
8      double beta = 1.0; //βの値
9
10     double deltaT = 0.1; //ステップ幅 0.296, 0.297あたりから
           発散?? → 0.1, 0.2, 0.3 で確認
11
12     double Un, preUn, pre2Un, pre3Un;
13     double preFn, pre2Fn, pre3Fn;
14
15     //printf("\n");
16     printf("Δt=%lf のとき\n", deltaT);
17
18     pre3Un = U0;
19     pre2Un = (U0 - beta / alpha) * exp(-alpha * deltaT) +
           beta / alpha; //解析解から得たu_1 の値
20     preUn = (U0 - beta / alpha) * exp(-alpha * deltaT * 2) +
            beta / alpha; //解析解から得たu_2 の値
21
22     printf("%lf %lf \n", 0.0, pre3Un);
23     printf("%lf %lf \n", deltaT, pre2Un);
24     printf("%lf %lf \n", deltaT * 2, preUn);
25
26     for (int i = 3; i <= 10 / deltaT; i++)
27     {
28         pre3Fn = (-alpha * pre3Un + beta) * deltaT;
29         pre2Fn = (-alpha * pre2Un + beta) * deltaT;
30         preFn = (-alpha * preUn + beta) * deltaT;
31
32         Un = preUn + deltaT / 144 * (211 * preFn - 92 *
               pre2Fn + 25 * pre3Fn); //予測子･修飾子法３
```

```
33
34          pre3Un = pre2Un;
35          pre2Un = preUn; //次の試行のための代入
36          preUn = Un;
37
38          printf("%lf %lf \n", deltaT * i, Un);
39      }
40
41      return 0;
42  }
```

プログラムの実行結果に基づき、以下の図を作成した。(ステップ幅 Δt の値を、0.1, 0.2, 0.3 にして実行した。)

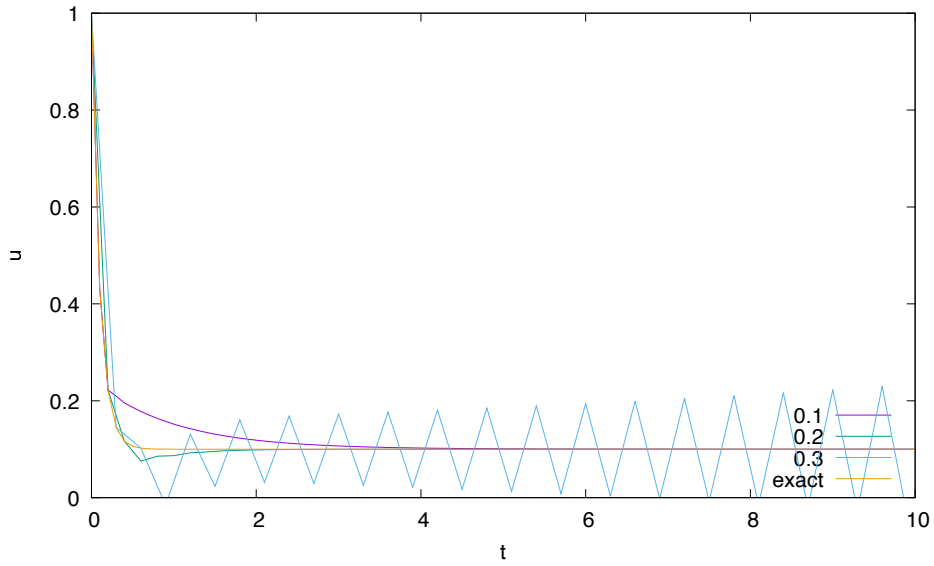

図 3: 予測子・修正子法（AB3 & AM3）による式（19）の数値解

この図より、予測子・修正子法（AB3 & AM3）で、$\Delta t = 0.4$ の場合に、解が振動しながら発散していく様子が分かる。（もう少し詳しく調べると、$0.296 < \Delta t < 0.297$ の間で発散し始めることがわかった。）

また、同じ予測子・修飾子法でも、やはり、より高次のものが精度が高いことが確認できた。

4.4 ホイン法

ホイン法において、

$$\begin{cases} u_n = u_{n-1} + \frac{\Delta t}{2}(f_{n-1} + f(t_n, u_n^*)), \\ u_n^* = u_{n-1} + f_{n-1}\Delta t \end{cases} \tag{25}$$

である。

この式をを用いてホイン法の安定性を数値的に確認した。用いたソースコードを、ソースコード 5 に示す。

ソースコード 5: C 言語を用いてホイン法の安定性を確認するプログラム

```
1  //数理工学実験テーマ２ //課題４ //ホイン法
2  #include <stdio.h>
3  #include <math.h>
4  int main(int argc, char **argv)
5  {
6      double U0 = 1.0; //u_0 の値
7      double alpha = 10.0; //αの値
8      double beta = 1.0; //βの値
9
10     double deltaT = 0.1; //ステップ幅 0.44, 0.45あたりから発
           散?? → 0.1, 0.4, 0.5 で確認
11
12     double Un, preUn;
13     double Fn, preFn;
14
15     //printf("\n");
16     printf("Δt=%lf のとき\n", deltaT);
17
18     preUn = U0;
19
20     printf("%lf %lf \n", 0.0, preUn);
21
22     for (int i = 1; i <= 10 / deltaT; i++)
23     {
24         preFn = (-alpha * preUn + beta) * deltaT;
25
26         Un = preUn + preFn * deltaT; //u_n^*
27         Fn = (-alpha * Un + beta) * deltaT; //f(t_n, u_n^*)
28         Un = preUn + deltaT / 2 * (preFn + Fn); //ホイン法
29
30         preUn = Un; //次の試行のための代入
31
32         printf("%lf %lf \n", deltaT * i, Un);
33     }
```

```
34
35      return 0;
36  }
```

プログラムの実行結果に基づき、以下の図を作成した。(ステップ幅 Δt の値を、0.1, 0.4, 0.5 にして実行した。)

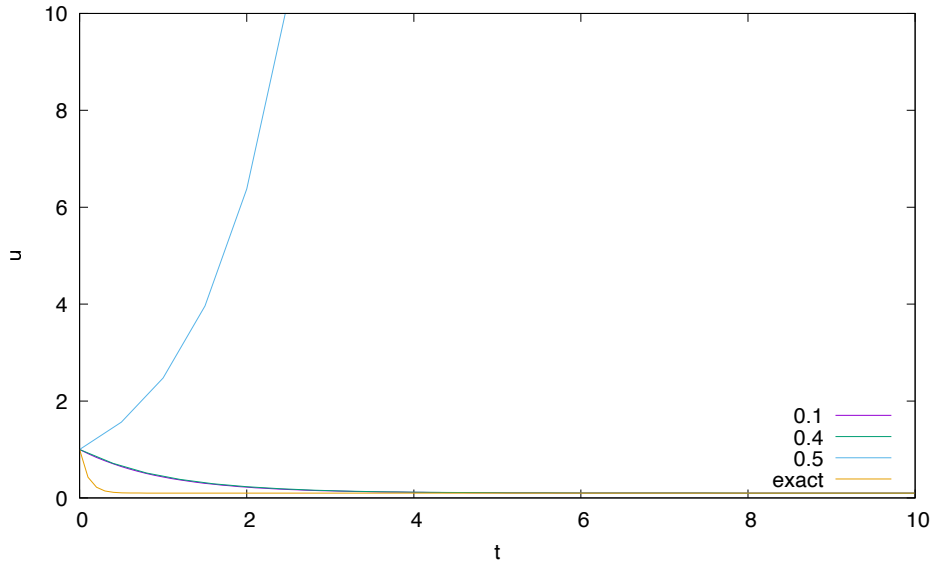

図 4: ホイン法による式（19）の数値解

この図より、ホイン法で、$\Delta t = 0.5$ の場合に、解が発散していく様子が分かる。（もう少し詳しく調べると、$0.44 < \Delta t < 0.45$ の間で発散し始めることがわかった。）

5　課題５

数値計算手法は自由に作ることができる。例えば、式 (6) の代わりに、

$$u_n - u_{n-2} = \int_{t_{n-2}}^{t_n} f(\tau, u(\tau))d\tau \tag{26}$$

として、$t \in [t_{n-2}, t_n]$ の範囲で $f \approx f_{n-1}$ と近似すれば、

$$u_n = u_{n-2} + 2f(t_{n-1}, u_{n-1})\Delta t \tag{27}$$

を得る。しかし、この式は $t \in [0, \infty)$ における常微分方程式の初期値問題

$$u' = -2u + 1, \quad u(0) = 1 \tag{28}$$

を解くことができない。これを以下で示す。

まず、式（27）と式（28）より、u_n は、

$$u_n = u_{n-2} + 2(-2u_(n-1) + 1)\Delta t \tag{29}$$

となる。これを $u_n = \lambda^n$ として特性方程式に直すと、

$$\lambda^n + 4\Delta t\lambda^{n-1} - \lambda^{n-2} - 2\Delta t = 0 \tag{30}$$

と書ける。$\lambda \neq 0$ として、式（30）の同時方程式

$$\lambda^n + 4\Delta t\lambda^{n-1} - \lambda^{n-2} = 0 \tag{31}$$

を解くと、$\lambda = -2\Delta t \pm \sqrt{4(\Delta t)^2 + 1}$ を得る。（$\lambda_1 = -2\Delta t + \sqrt{4(\Delta t)^2 + 1}$, $\lambda_2 = -2\Delta t - \sqrt{4(\Delta t)^2 + 1}$ とする。）

ここで、$-2\Delta t < 0$ であることと、$\sqrt{4(\Delta t)^2 + 1} > 1$ より、$|\lambda_2| > 1$ であることが分かる。u_n は、定数 C_1, C_2, C_3, を用いて、$u_n = C_1\lambda_1^n + C_2\lambda_2^n + C_3$ と表されることを考えると、$|\lambda_2| > 1$ より、$n \to \infty$ で u_n は発散してしまう。

以上のことから、式（27）は式（28）を解くことができないと言える。

このことを、実際に数値計算で確かめる。用いたソースコードを、ソースコード 6 に示す。また、必要となる初期条件は、式（28）の解、$u(t) = \frac{1}{2}e^{-2t} + \frac{1}{2}$ から解析的に求めた。

ソースコード 6: C 言語を用いて式（27）は式（28）を解くことができないことを確認するプログラム

```
1  //数理工学実験テーマ２ //課題５
2  #include <stdio.h>
3  #include <math.h>
4  int main(int argc, char **argv)
5  {
6      double deltaT = 0.1; //ステップ幅 → 0.1, 0.01, 0.001
7      int N = 10 / deltaT; //ステップ数
8      double U0, U1; //u_0, u_1 の値
9      double Un, preUn, pre2Un; //それぞれ、u_n, u_(n-1),u_(n
           -2)
10
11     U0 = 1.0;
12     U1 = 0.5 * exp(-2 * deltaT) + 0.5; //解析解から求めた
           u_1 の値
13
14     printf("t の値 un の値\n");
15     printf("%lf %lf\n", 0.0, U0);
16     printf("%lf %lf\n", deltaT, U1);
17
18     pre2Un = U0;
```

```
19      preUn = U1;
20
21      for (int k = 2; k <= N; k++)
22      {
23          Un = pre2Un + 2 * (-2 * preUn + 1) * deltaT; //
                式(2.33)
24
25          printf("%lf %lf\n", deltaT * k, Un);
26
27          pre2Un = preUn; //次の試行のための代入
28          preUn = Un; //次の試行のための代入
29      }
30
31      return 0;
32  }
```

プログラムの実行結果に基づき、以下の図 5, 6, 7 を作成した。(ステップ幅 Δt の値を、0.001, 0.01, 0.1 にして実行した。)

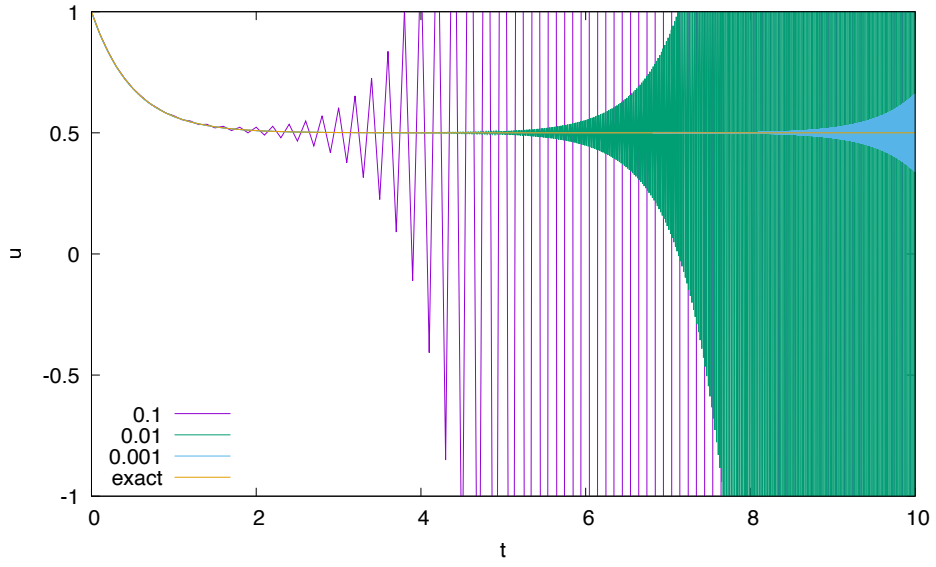

図 5: 式（27）による式（28）の数値解（$u \in [-1, 1]$）

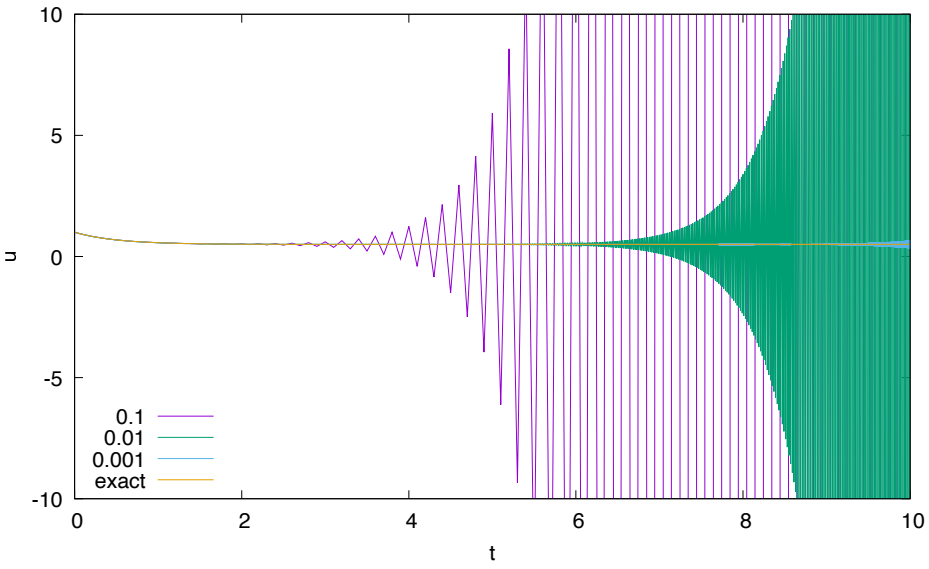

図 6: 式（27）による式（28）の数値解（$u \in [-10, 10]$）

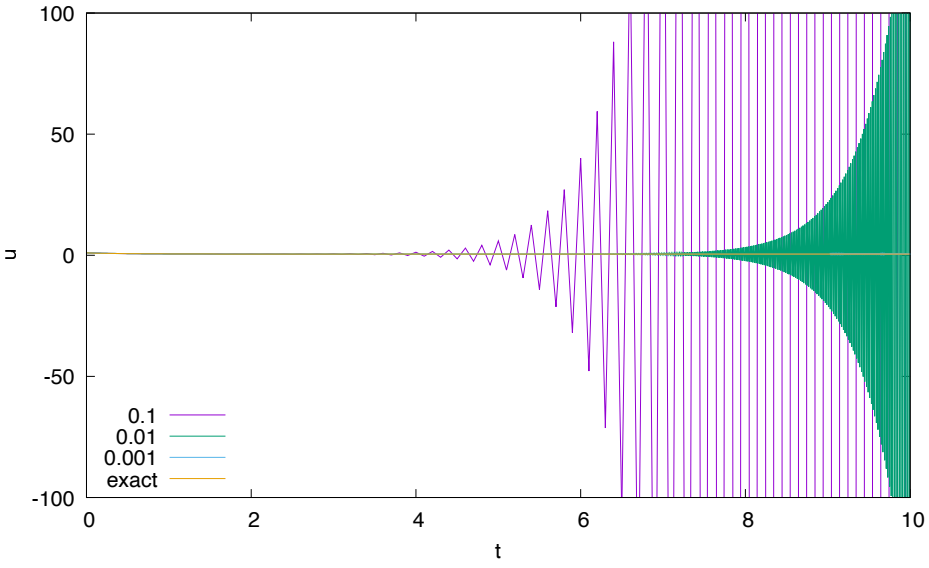

図 7: 式（27）による式（28）の数値解（$u \in [-100, 100]$）

この図より、ステップ幅 Δt に対し決まる、ある $\widetilde{t}$ があるとすると、$t > \widetilde{t}$ の範囲で数値解が発散していく様子が観察できる。また、$\widetilde{t}$ の値は Δt が小さいほど大きくなることが分かる。

6　課題６

生物の個体数の増減に関する数理モデルを考える。$t \geq 0$ を時刻、$u(t)$ を時刻 t における個体数、$u(0) = u0 > 0$ を初期個体数とする。個体数の時間変化 u' は、個体数が多いほど大きいと考えるのが自然である。u が小さいと交配相手が見つかる確率が減り、逆に u が大きいと交配相手が見つかる確率が高まる（アリー効果という）ことを考慮して、今回は個体数の時間変化 u' を、

$$u' = (u-1)u, \quad u(0) = u_0 \tag{32}$$

と表すことにする。

式（32）を数値解析し、解の振舞いを調べたい。ここで、式（32）を見ると有限の T に対して、$t \to T$ で $u = u(t)$ が発散し、解は爆発する。そこで、変数変換をすることにより爆発解の数値計算を簡単にする。新たな独立変数として、$t-u$ 平面において解が描く軌道に沿った長さを s を用いる。つまり、

$$\frac{\mathrm{d}s}{\mathrm{d}t} = \sqrt{1+(u')^2} = \sqrt{1+(u-1)^2u^2} \tag{33}$$

である。このとき、式（32）は $u = u(s)$ および $t = t(s)$ を用いて、

$$\frac{\mathrm{d}u}{\mathrm{d}s} = \frac{(u-1)u}{\sqrt{1+(u-1)^2u^2}}, \quad u(0) = u_0 \tag{34}$$

$$\frac{\mathrm{d}t}{\mathrm{d}s} = \frac{1}{\sqrt{1+(u-1)^2u^2}}, \quad t(0) = 0 \tag{35}$$

と表される。

これらのことを用いて、式（32）を数値解析し、解の振舞いを調べた。用いたソースコードを、ソースコード 7 に示す。計算手法には、前進オイラー法を用いた。

ソースコード 7: C 言語を用いて式（32）を数値解析し、解の振舞いを調べるプログラム

```
1  //数理工学実験テーマ２ //課題６
2  #include <stdio.h>
3  #include <math.h>
4  int main(int argc, char **argv)
5  {
6      double deltaT = 0.01; //ステップ幅
7      int N = 10 / deltaT; //ステップ数
8      double U0, U1; //u_0 の値
9      double Un, preUn; //それぞれ、u_n, u_(n-1)
10     double preFn; //f_(n-1) : u'
11     double duds; //du/ds
12
13     U0 = 0.5; //u_0 の値→ 0.5, 1.0, 1.5
```

```
14
15      printf("t の値 un の値\n");
16      printf("%lf %lf\n", 0.0, U0);
17
18      preUn = U0;
19
20      for (int k = 1; k <= N; k++)
21      {
22          preFn = pow(preUn, 2.0) - preUn;
23          duds = preFn / sqrt(1.0 + pow(preFn, 2.0));
24
25          Un = preUn + duds * deltaT; //オイラー法
26
27          printf("%lf %lf\n", deltaT * k, Un);
28
29          preUn = Un; //次の試行のための代入
30      }
31
32      return 0;
33  }
```

プログラムの実行結果に基づき、以下の図（8）を作成した。(u_0 の値を、0.5, 1.0, 1.5 にして実行した。)

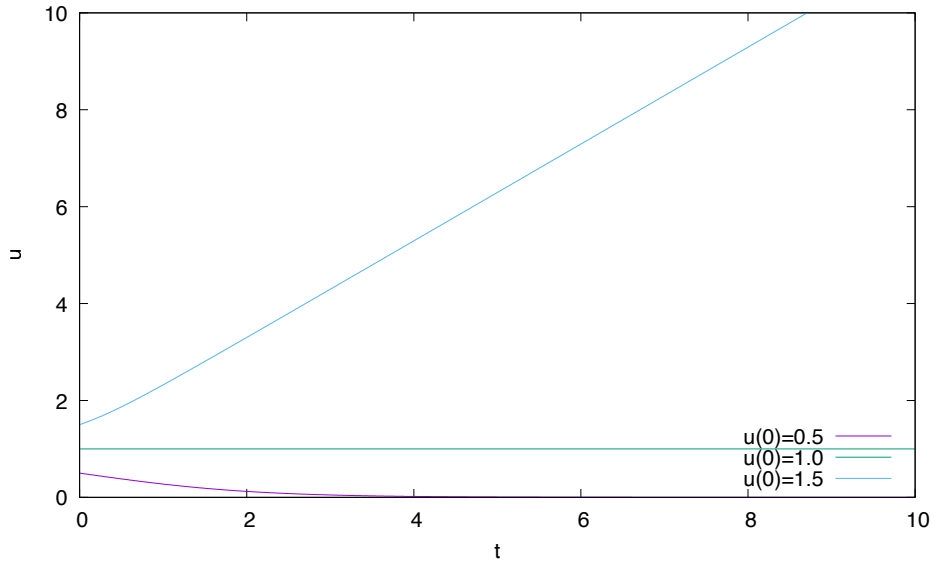

図 8: 式（32）の数値解

この図より、式（32）は、$u_0 = 1$ を境に、解の振舞いが変わると考えられる。

7　課題 7

解が興味深い振舞いをする例として、「カオス」がある。カオス的な性質のうち、「非周期的な長時間挙動」と「初期条件への鋭敏な依存性」について調べる。

ここで、カオス性を持つローレンツ方程式を導入しておく。

$$\begin{aligned} x' &= \sigma(y-x), \quad x(0) = x_0 \\ y' &= rx - y - xz, \quad y(0) = y_0 \\ z' &= xy - bz, \quad z(0) = z_0 \end{aligned} \tag{36}$$

これが、ローレンツ方程式である。

3 つの未知数 $x(t)$, $y(t)$, $z(t)$ が、このローレンツ方程式に従うとする。σ, r, b はパラメータであり、すべて正とする。

今回は、ローレンツ方程式（36）において、$\sigma = 10$, $b = 8/3$, $r = 28$, $x_0 = 1 + \epsilon$, $y_0 = 0$, $z_0 = 0$、つまり、

$$\begin{aligned} x' &= 10(y-x), \quad x(0) = 1 + \epsilon \\ y' &= 28x - y - xz, \quad y(0) = 0 \\ z' &= xy - \frac{8}{3}z, \quad z(0) = 0 \end{aligned} \tag{37}$$

として、数値解析を行う。計算手法は、前進オイラー法とルンゲ・クッタ法を用いる。計算領域は、$t \in [0, 100]$ とする。

7.1　1

まず、$\epsilon = 0$ として数値解析を行なった。用いたソースコードを、ソースコード 8, 9 に示す。

ソースコード 8: C 言語を用いて式（37）を前進オイラー法で数値解析し、解の振舞いを調べるプログラム

```
1  //数理工学実験テーマ２ //課題 7-1 //前進オイラー法
2  #include <stdio.h>
3  #include <math.h>
4  int main(int argc, char **argv)
5  {
6      double deltaT = 0.01; //ステップ幅
7      int N = 100 / deltaT; //ステップ数
8      double X0, Y0, Z0; //x_0, y_0, z_0 の値
9      double Xn, preXn, Yn, preYn, Zn, preZn; //x_n, x_(n-1),
           y_n, y_(n-1), z_n, z_(n-1)
10     double preFnx, preFny, preFnz; //f_(n-1) : x', y', z'
11
```

```
12      X0 = 1.0;
13      Y0 = 0;
14      Z0 = 0;
15
16      printf("t xn yn zn\n");
17      printf("%lf %lf %lf %lf\n", 0.0, X0, Y0, Z0);
18
19      preXn = X0;
20      preYn = Y0;
21      preZn = Z0;
22
23      for (int k = 1; k <= N; k++)
24      {
25          preFnx = 10 * (preYn - preXn);
26          preFny = 28 * preXn - preYn - preXn * preZn;
27          preFnz = preXn * preYn - 8 / 3 * preZn;
28
29          //前進オイラー法
30          Xn = preXn + preFnx * deltaT;
31          Yn = preYn + preFny * deltaT;
32          Zn = preZn + preFnz * deltaT;
33
34          printf("%lf %lf %lf %lf\n", deltaT * k, Xn, Yn, Zn);
35
36          //次の試行のための代入
37          preXn = Xn;
38          preYn = Yn;
39          preZn = Zn;
40      }
41
42      return 0;
43  }
```

ソースコード 9: C 言語を用いて式（37）をルンゲ・クッタ法で数値解析し、解の振舞いを調べるプログラム

```
1  //数理工学実験テーマ２ //課題 7-1 //ルンゲ･クッタ法
2  #include <stdio.h>
3  #include <math.h>
4  int main(int argc, char **argv)
5  {
6      double deltaT = 0.01; //ステップ幅
7      int N = 100 / deltaT; //ステップ数
8      double X0, Y0, Z0; //x_0, y_0, z_0 の値
9      double Xn, preXn, Yn, preYn, Zn, preZn; //x_n, x_(n-1),
           y_n, y_(n-1), z_n, z_(n-1)
```

```
10      double preFnx, preFny, preFnz; //f_(n-1) : x', y', z'
11      double F1x, F2x, F3x, F4x, F1y, F2y, F3y, F4y, F1z, F2z,
            F3z, F4z;
12
13      X0 = 1.0;
14      Y0 = 0;
15      Z0 = 0;
16
17      printf("t xn yn zn\n");
18      printf("%lf %lf %lf %lf\n", 0.0, X0, Y0, Z0);
19
20      preXn = X0;
21      preYn = Y0;
22      preZn = Z0;
23
24      for (int k = 1; k <= N; k++)
25      {
26          F1x = 10 * (preYn - preXn);
27          F2x = 10 * (preYn - preXn) + F1x * deltaT / 2;
28          F3x = 10 * (preYn - preXn) + F2x * deltaT / 2;
29          F4x = 10 * (preYn - preXn) + F3x * deltaT;
30
31          F1y = 28 * preXn - preYn - preXn * preZn;
32          F2y = 28 * preXn - preYn - preXn * preZn + F1y *
                deltaT / 2;
33          F3y = 28 * preXn - preYn - preXn * preZn + F2y *
                deltaT / 2;
34          F4y = 28 * preXn - preYn - preXn * preZn + F3y *
                deltaT;
35
36          F1z = preXn * preYn - 8 / 3 * preZn;
37          F2z = preXn * preYn - 8 / 3 * preZn + F1z * deltaT /
                 2;
38          F3z = preXn * preYn - 8 / 3 * preZn + F2z * deltaT /
                 2;
39          F4z = preXn * preYn - 8 / 3 * preZn + F3z * deltaT;
40
41          //ルンゲ･クッタ法
42          Xn = preXn + deltaT / 6 * (F1x + 2 * F2x + 2 * F3x
                + F4x);
43          Yn = preYn + deltaT / 6 * (F1y + 2 * F2y + 2 * F3y
                + F4y);
44          Zn = preZn + deltaT / 6 * (F1z + 2 * F2z + 2 * F3z
                + F4z);
45
46          printf("%lf %lf %lf %lf\n", deltaT * k, Xn, Yn, Zn);
```

```
47
48          //次の試行のための代入
49          preXn = Xn;
50          preYn = Yn;
51          preZn = Zn;
52      }
53
54      return 0;
55  }
```

プログラムの実行結果に基づき、数値解析した $x(t)$ を t の関数として図示したものが、以下の図 9, 10 である。

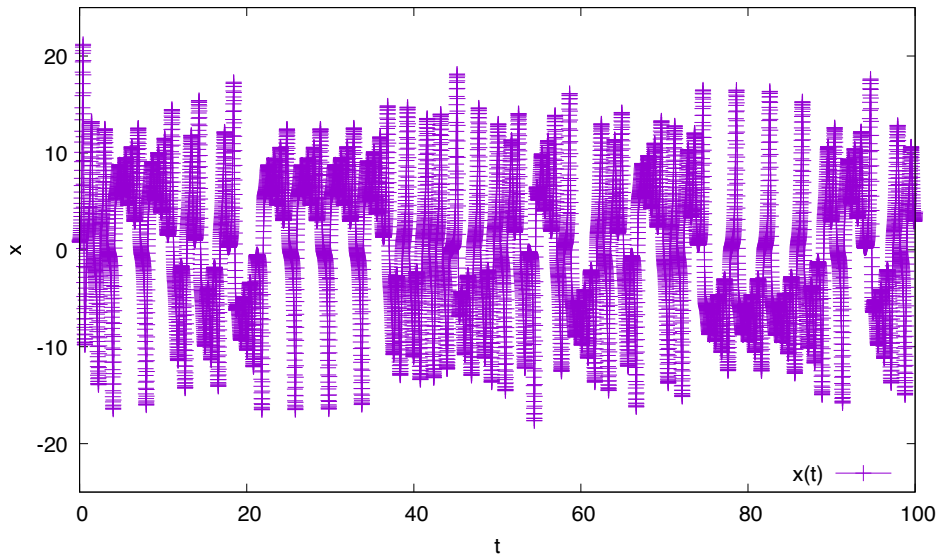

図 9: 前進オイラー法

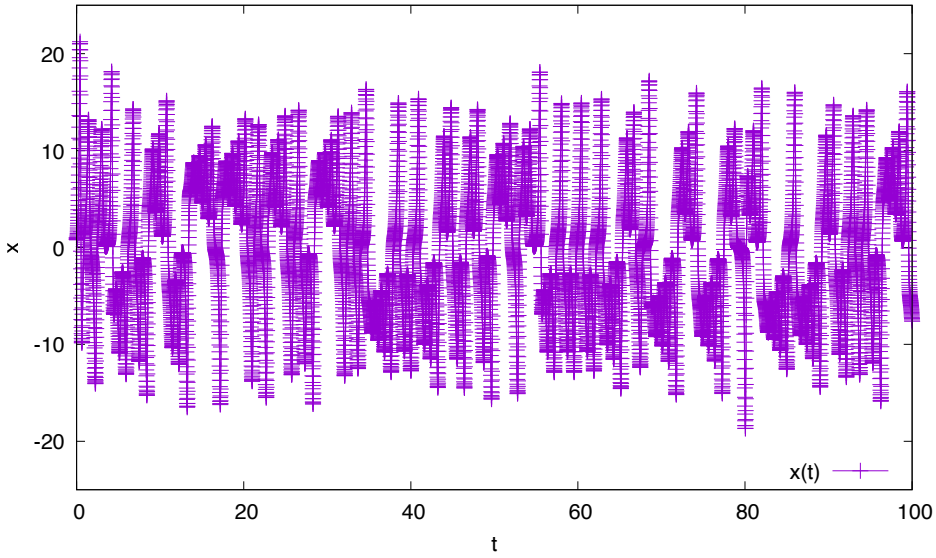

図 10: ルンゲ・クッタ法

また、$(x(t), y(t), z(t))$ の軌道を $x-y-z$ 空間上に図示したものが、以下の図 11, 12 である。

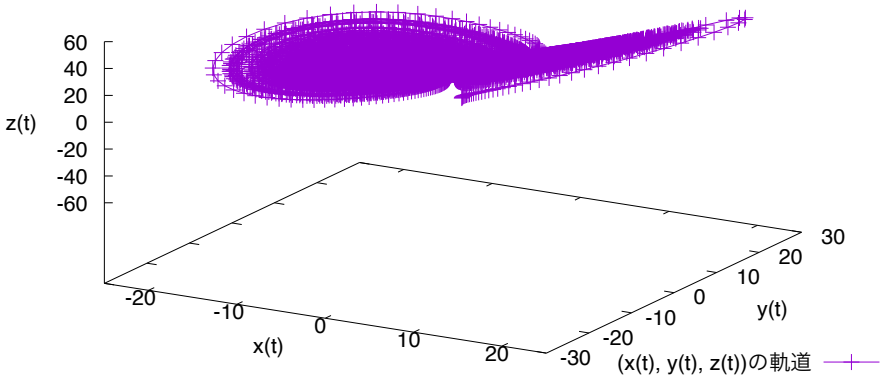

図 11: 前進オイラー法

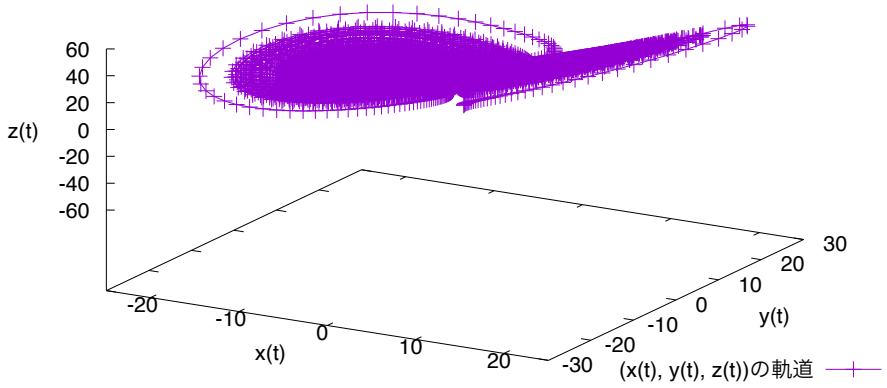

図 12: ルンゲ・クッタ法

これらの図から、カオス的な性質のうち、「非周期的な長時間挙動」が確認できた。

7.2 2

7.3 3

8　参考文献

・数理工学実験テキスト

・Cloud LaTeX
https://cloudlatex.io

・LaTeX コマンド一覧（リスト）
https://medemanabu.net/latex/latex-commands-list/

・LaTeX にソースコードを【美しく】貼る方法
https://ta-b0.hateblo.jp/entry/2020/08/13/001223

・範囲外から範囲外へ動く場合の線の描画
https://qiita.com/bunzaemon/items/c51c1ffb01f011252a27

・３次元データファイルのプロット
http://dsl4.eee.u-ryukyu.ac.jp/DOCS/gnuplot/node170.html