

数値解析レポート No.4

4 年 39 番 湯嶋 皓騎

2024 年 3 月 3 日

1 連立微分方程式

1.1 問題

以下を満たす (x, y) を $t = [0, 2]$ の範囲で求めよ.

$$\begin{cases} \frac{dx}{dt} = x + 6y + t - 10, & x(0) = 1 \\ \frac{dy}{dt} = x + t - 3, & y(0) = 2 \end{cases} \quad (1)$$

$$(2)$$

1.2 解法の検討

式 1 を $f(x, y; t)$, 式 2 を $g(x, y; t)$ とおくと,

$$\begin{cases} x = x(0) + \int_{t_0}^t f(x, y; t) dt \\ y = y(0) + \int_{t_0}^t g(x, y; t) dt \end{cases} \quad (3)$$

$$(4)$$

と表わされる. 第二項の定積分を数値的に求めることで, x, y を求めることができる.

1.2.1 オイラー法を用いた解法

ここで x, y を定数 (x_i, y_i) として $t = t_i$ ($i = 0, 1, 2, \dots$) で一次のテイラー展開をすると,

$$\begin{cases} x_{i+1} = x_i + f(x_i, y_i; t_i) \Delta t \\ y_{i+1} = y_i + g(x_i, y_i; t_i) \Delta t \end{cases} \quad (5)$$

$$(6)$$

とできる. これを用いて $t = 0$ から $t = 2$ までの範囲で x, y を求める. 実装の主要部を List.1 に示す.

List. 1: sod.c

```
1 #define PARAM_FILENAME "sod.csv"
2 #define UNUSED(x) (void)(x)
3
4 typedef double (*F)(double, double, double);
5 typedef struct {
```

```

6     F f;
7     F g;
8     double x0;
9     double y0;
10    double width;
11    double t0;
12    double t_inf;
13 } Condition;
14
15 double f(double x, double y, double t) { return x + 6 * y + t - 10; }
16 double g(double x, double y, double t) {
17     UNUSED(y);
18     return x + t - 3;
19 }
20
21 Matrix *solve_sod(Condition *cond) {
22     size_t len =
23         (size_t)((cond->t_inf - cond->t0) / cond->width) + 1; // len = n + 1
24     Matrix *matrix = allocMatrix(3, len); // data[0][i] = x_i, data[1][i] = y_i, data[2][i]
25         = t_i
26     size_t i;
27     double *x = matrix->data[0];
28     double *y = matrix->data[1];
29     double *t = matrix->data[2];
30
31     if (matrix == NULL) {
32         fprintf(stderr, "Error: allocMatrix() failed.\n");
33         return NULL;
34     }
35
36     x[0] = cond->x0;
37     y[0] = cond->y0;
38     t[0] = cond->t0;
39     for (i = 1; i < len; i++) {
40         x[i] = x[i - 1] + cond->f(x[i - 1], y[i - 1], t[i - 1]) * cond->width;
41         y[i] = y[i - 1] + cond->g(x[i - 1], y[i - 1], t[i - 1]) * cond->width;
42         t[i] = cond->t0 + cond->width * i;
43     }
44
45     return matrix;
46 }

```

ここで、UNUSED マクロは引数を使わないことを明示するためのマクロであり、これによりコンパイラの警告を抑制している。また、連立常微分方程式の解を求める関数 solve_sod は、Condition 構造体を引数に取り、その構造体には $\frac{dx}{dt}, \frac{dy}{dt}$ を表わす関数 f, g, 初期値 x0, y0, 刻み幅 width, 時刻の初期値 t0, 時刻の終端

値 `t_inf` が含まれている.

1.2.2 ルンゲ・クッタ法を用いた解法

ルンゲ・クッタ法を用いて連立微分方程式を解くこともできる. ルンゲ・クッタ法はオイラー法を改良したものであり, 4 次のルンゲ・クッタ法は $O(h^5)$ の精度を持つ. 実装を List.2 に示す.

List. 2: rk.c

```
1 Matrix *solve_sod1(Condition *cond) {
2     size_t len =
3         (size_t)((cond->t_inf - cond->t0) / cond->width) + 1; // len = n + 1
4     Matrix *matrix = allocMatrix(3, len); // data[0][i] = x_i, data[1][i] = y_i
5     size_t i;
6     double *x = matrix->data[0];
7     double *y = matrix->data[1];
8     double *t = matrix->data[2];
9
10    if (matrix == NULL) {
11        fprintf(stderr, "Error: allocMatrix() failed.\n");
12        return NULL;
13    }
14
15    x[0] = cond->x0;
16    y[0] = cond->y0;
17    for (i = 1; i < len; i++) {
18        double kx[4], ky[4];
19        kx[0] = cond->f(x[i - 1], y[i - 1], t[i - 1]);
20        ky[0] = cond->g(x[i - 1], y[i - 1], t[i - 1]);
21        kx[1] = cond->f(x[i - 1] + cond->width / 2 * kx[0],
22                      y[i - 1] + cond->width / 2 * ky[0],
23                      t[i - 1] + cond->width / 2);
24        ky[1] = cond->g(x[i - 1] + cond->width / 2 * kx[0],
25                      y[i - 1] + cond->width / 2 * ky[0],
26                      t[i - 1] + cond->width / 2);
27        kx[2] = cond->f(x[i - 1] + cond->width / 2 * kx[1],
28                      y[i - 1] + cond->width / 2 * ky[1],
29                      t[i - 1] + cond->width / 2);
30        ky[2] = cond->g(x[i - 1] + cond->width / 2 * kx[1],
31                      y[i - 1] + cond->width / 2 * ky[1],
32                      t[i - 1] + cond->width / 2);
33        kx[3] = cond->f(x[i - 1] + cond->width * kx[2],
34                      y[i - 1] + cond->width * ky[2], t[i - 1] + cond->width);
35        ky[3] = cond->g(x[i - 1] + cond->width * kx[2],
36                      y[i - 1] + cond->width * ky[2], t[i - 1] + cond->width);
37
38        x[i] = x[i - 1] +
```

```

39         cond->width * (kx[0] + 2 * kx[1] + 2 * kx[2] + kx[3]) / 6;
40     y[i] = y[i - 1] +
41         cond->width * (ky[0] + 2 * ky[1] + 2 * ky[2] + ky[3]) / 6;
42     t[i] = t[i - 1] + cond->width;
43 }
44
45     return matrix;
46 }

```

1 変数関数の数値積分とは異なり, x と y 両方の仮ステップ k を求める必要があるため, kx , ky を配列として定義している. また, ルンゲ・クッタ法は 4 次のルンゲ・クッタ法を用いている.

1.3 実行結果

式 3, 4 を満たす (x, y) を $t = [0, 2]$ の範囲で求めるために, List. 3 を入力に与えた.

List. 3: sodin.csv

```

1 1,2,0.1,0,2

```

List. 3 は初期値 $x(0) = 1, y(0) = 2$ に対して $t = 0$ から $t = 2$ までステップ幅 0.1 で求めることを指示している. プログラムの出力は x, y, t の順で出力される.

1.3.1 オイラー法

List. 1 を実行した結果を List. 4 に示す.

List. 4: eular.csv

```

1 1.000000,2.000000,0.000000
2 1.300000,1.800000,0.100000
3 1.520000,1.640000,0.200000
4 1.676000,1.512000,0.300000
5 1.780800,1.409600,0.400000
6 1.844640,1.327680,0.500000
7 1.875712,1.262144,0.600000
8 1.880570,1.209715,0.700000
9 1.864456,1.167772,0.800000
10 1.831565,1.134218,0.900000
11 1.785252,1.107374,1.000000
12 1.728201,1.085899,1.100000
13 1.662561,1.068719,1.200000
14 1.590049,1.054976,1.300000
15 1.512039,1.043980,1.400000
16 1.429631,1.035184,1.500000
17 1.343705,1.028147,1.600000
18 1.254964,1.022518,1.700000
19 1.163971,1.018014,1.800000
20 1.071177,1.014412,1.900000

```

21 0.976942,1.011529,2.000000

List. 4 に加えてステップ幅を 0.2, 0.5 に設定したものと解析解を重ねて描画したものを Fig.1 に示す. 解析解の曲線描画には gnuplot の `set parametric` オプションを用いた.

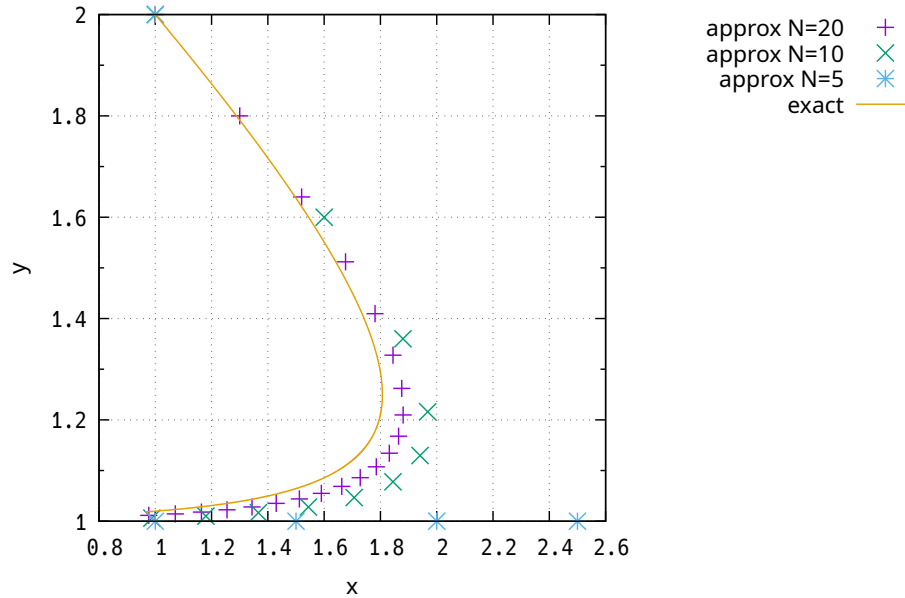


Fig. 1: オイラー法を用いた連立微分方程式の解

まずは Fig.1 の分割数 $N = 20$ の点群と解析解を比較する. この連立微分方程式は初期値 $(x_0, y_0) = (1, 2)$ を与えたとき, $(x, y) = (1.8, 1.25)$ 付近までは x が増加し y が減少しているが, それ以降は x も y も減少していることがわかる. また, ステップ幅を 0.1 としたとき, 解析解との最大誤差は 0.1 程度であった. 数値解と解析解の誤差はグラフの変化が激しい部分で大きくなっている. これは式 5, 6 で一次のテイラー展開を用いているためである.

次に, ステップ幅を変化させたときの数値解と解析解の乖離を確認する. ステップ幅が 0.1, 0.2 のときは解析解の形状に近いが, ステップ幅が 0.5 のときは大きく乖離している. List. 5 にステップ幅 0.5 の実行結果を示す.

List. 5: 1-1-5.csv

```
1 1.000000,2.000000,0.000000
2 2.500000,1.000000,0.500000
3 2.000000,1.000000,1.000000
4 1.500000,1.000000,1.500000
5 1.000000,1.000000,2.000000
```

これはステップ幅を 0.5 に設定したとき, $g(x_i, y_i; t_i)$ が t_0 以外で 0 になってしまうためである. これより, オイラー法ではステップ幅が大きいときには誤差が非常に大きくなる場合があることがわかる.

1.3.2 ルンゲ・クッタ法

ステップ幅を 0.5 としたときの実行結果を List. 6 に示す.

```

1 1.000000,2.000000,0.000000
2 1.750000,1.375000,0.500000
3 1.718750,1.140625,1.000000
4 1.394531,1.052734,1.500000
5 0.960449,1.019775,2.000000

```

List. 6 と解析解を重ねて描画したものを Fig.2 に示す. ステップ幅は 0.1, 0.2, 0.5 の 3 通りで実行し, 図中には分割数 N で示している.

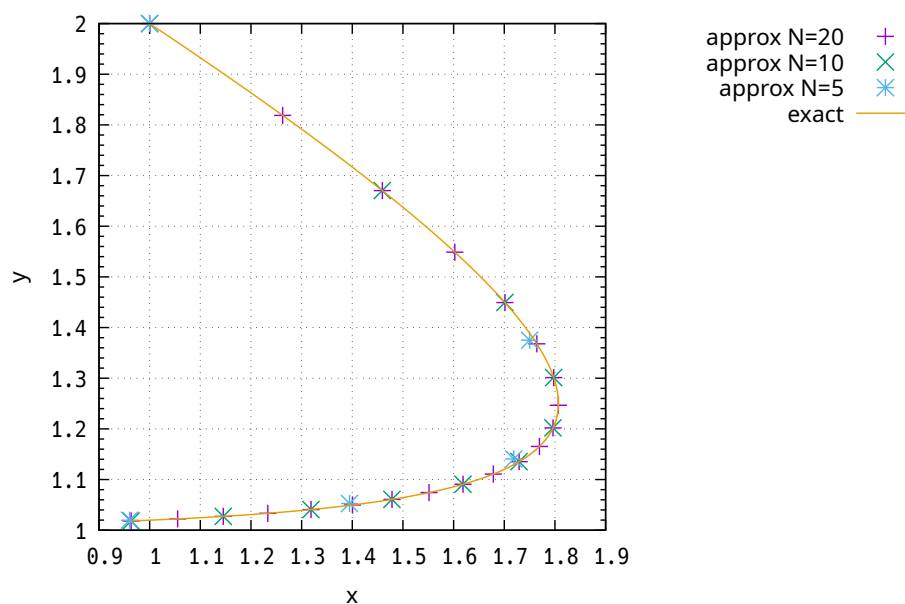


Fig. 2: ルンゲ・クッタ法を用いた連立微分方程式の解

いずれのステップ幅においても, オイラー法より誤差が小さくほとんど解析解と一致していることがわかる.

2 高次微分方程式

2.1 問題

式 7 の $x = 1$ における微分方程式の解 y を求めよ.

$$\frac{d^2 y}{dx^2} + 5 \frac{dy}{dx} - 6y = 0 \quad (7)$$

ここで, 初期条件は $y(0) = 0, \frac{dy}{dx}(0) = 7$ とする.

2.2 解法の検討

$\frac{dy}{dx} = z$ とおくと、式 7 は式 8 のように表わされる.

$$\begin{aligned}\frac{dz}{dx} + 5z - 6y &= 0 \\ \frac{dz}{dx} &= 6y - 5z\end{aligned}\tag{8}$$

これらは独立変数 x に関する連立常微分方程式であるから、先の問題と同様に解くことができる.

$f(y, z; x) = z, g(y, z; x) = 6y - 5z$ とおき、 $x = 0$ での初期条件 $y(0) = 0, z(0) = 7$ に対してルンゲ・クッタ法を用いて解く. List. 2 で示したプログラムは独立変数 t に関する x, y の連立微分方程式を解くものであるため、 $y \rightarrow x, z \rightarrow y, x \rightarrow t$ と変数変換して用いることとする.

2.3 実装

プログラムを List.7 に示す. ここで `solve_sod1()` はルンゲ・クッタ法を用いて連立微分方程式を解く関数である.

List. 7: 2nd.c

```
1 double f(double y, double z, double x) {
2     UNUSED(y);
3     UNUSED(x);
4     return z;
5 }
6 double g(double y, double z, double x) {
7     UNUSED(x);
8     return 6 * y - 5 * z;
9 }
10
11 int main(void) {
12     Matrix *matrix, *res;
13     size_t i;
14     Condition cond;
15
16     // データの読み込みと検証
17
18     cond = (Condition){
19         .f = f,
20         .g = g,
21         .x0 = matrix->data[0][0],
22         .y0 = matrix->data[0][1],
23         .width = matrix->data[0][2],
24         .t0 = matrix->data[0][3],
25         .t_inf = matrix->data[0][4],
26     };
27     printf("#step width: %f\n", cond.width);
```

```

28     res = solve_sod1(&cond);
29     if (res == NULL) {
30         freeMatrix(matrix);
31         return EXIT_FAILURE;
32     }
33
34     for (i = 0; i < res->col; i++) {
35         printf("%.6f,%.6f,%.6f\n", res->data[0][i], res->data[1][i],
36             res->data[2][i]);
37     }
38
39     freeMatrix(res);
40     freeMatrix(matrix);
41
42     return EXIT_SUCCESS;
43 }

```

このプログラムを実行すると、1 列目に y 、2 列目に z 、3 列目に x が出力される。

2.3.1 実行結果

$x = 1$ での y の値を求めるために、List. 8 を入力に与えた。

List. 8: 2ndin.csv

```

1 0,7,0.1,0,1

```

List. 8 は初期値 $y(0) = 0, z(0) = 7$ に対して $x = 0$ から $x = 1$ までステップ幅 0.1 で求めることを指示している。実行した結果を List. 9 に示す。

List. 9: 2nd.csv

```

1 #step width: 0.100000
2 0.000000,7.000000,0.000000
3 0.555771,4.401571,0.100000
4 0.919562,3.032445,0.200000
5 1.184027,2.344845,0.300000
6 1.400717,2.038470,0.400000
7 1.598666,1.949048,0.500000
8 1.794618,1.987118,0.600000
9 1.998643,2.104402,0.700000
10 2.217239,2.275343,0.800000
11 2.455041,2.486963,0.900000
12 2.715774,2.733312,1.000000

```

csv ファイルの 1 列目と 3 列目より、 $x = 1$ における y の値は 2.715774 であることがわかる。解析解 $y = e^x - e^{-6x}$ に $x = 1$ を代入した結果は 2.715803 であるため、相対誤差は

$$\frac{|2.715774 - 2.715803|}{2.715803} \times 100 = 0.0011 [\%] \quad (9)$$

であり、非常に高い精度で数値解が解析解と一致していることがわかる。

次に、List. 9 を解析解と重ねてプロットした結果を Fig. 3 に示す。今回は $x = 1$ での解が正しく求められればよいため、ステップ幅は変化させず 0.1 のみ実行した。

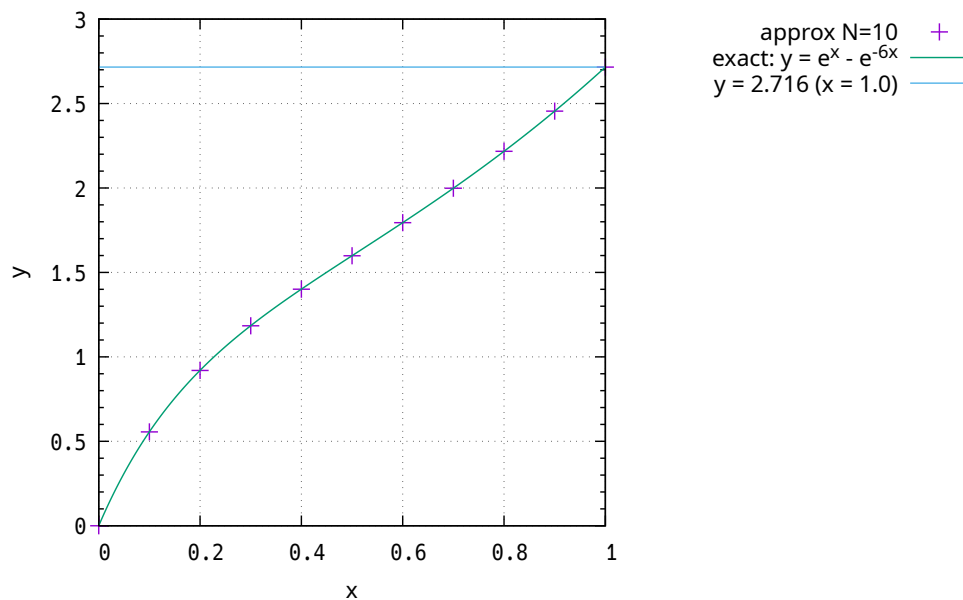


Fig. 3: 高次微分方程式の解

Fig. 3 より、解析解と数値解は広い範囲において高い精度で一致していることがわかる。

3 感想

ルンゲ・クッタ法はオイラー法と比べて高い精度を持つことを改めて確認できた。数値解析で学んだことを卒業研究でも活かしていきたい。

参考文献

- [1] gnuplot でグラフに文字を書く – 米澤進吾 ホームページ <https://sk.kuee.kyoto-u.ac.jp/person/yonezawa/contents/program/gnuplot/label.html>