数値解析レポート No.4

4 年 39 番 湯嶋 皓騎

2024 年 3 月 2 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.

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

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,

$$\begin{cases} x = x(0) + \displaystyle\int_{t_0}^{t} f(x, y; t)dt & (3) \\ y = y(0) + \displaystyle\int_{t_0}^{t} g(x, y; t)dt & (4) \end{cases}$$

と表わされる. 右辺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 $t = t_0$ で一次のテイラー展開をすると,

$$\begin{cases} x_{i+1} = x_i + f(x_i, y_i; t_i)\Delta t & (5) \\ y_{i+1} = y_i + g(x_i, y_i; t_i)\Delta t & (6) \end{cases}$$

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. ソースコードの主要部を 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]
           = t_i
25     size_t i;
26     double *x = matrix->data[0];
27     double *y = matrix->data[1];
28     double *t = matrix->data[2];
29
30     if (matrix == NULL) {
31         fprintf(stderr, "Error: allocMatrix() failed.\n");
32         return NULL;
33     }
34
35     x[0] = cond->x0;
36     y[0] = cond->y0;
37     t[0] = cond->t0;
38     for (i = 1; i < len; i++) {
39         x[i] = x[i - 1] + cond->f(x[i - 1], y[i - 1], t[i - 1]) * cond->width;
40         y[i] = y[i - 1] + cond->g(x[i - 1], y[i - 1], t[i - 1]) * cond->width;
41         t[i] = cond->t0 + cond->width * i;
42     }
43
44     return matrix;
45 }
```

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．

このプログラムをステップ幅 0.1 で実行した結果を Fig.1 に示す．また，解析解の曲線描画には gnuplot の `set parametric` オプションを用いた．

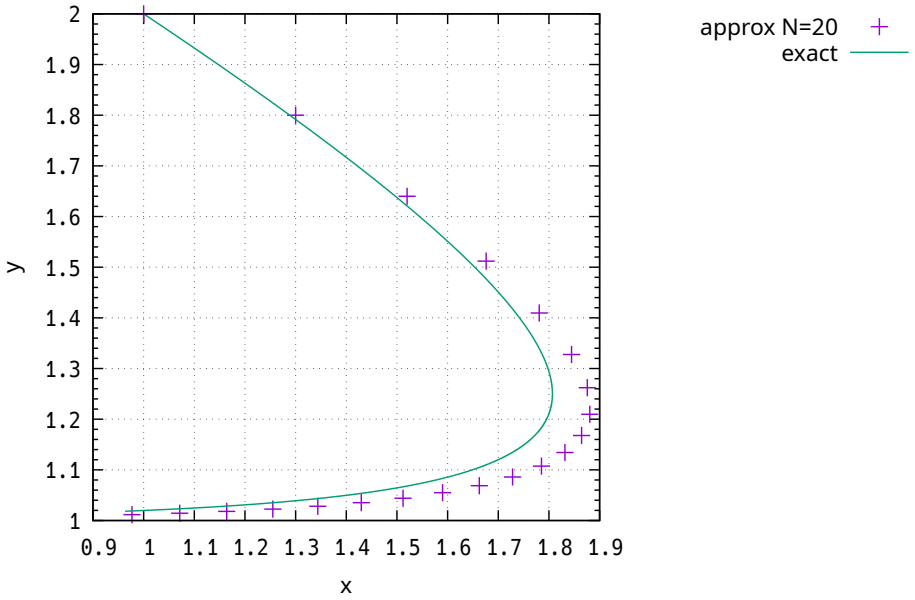

Fig. 1: オイラー法を用いた連立常微分方程式の解

Fig.1 より，この連立常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．

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常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．ルンゲ・クッタ法を用いて連立常微分方程式を解いた結果を 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．

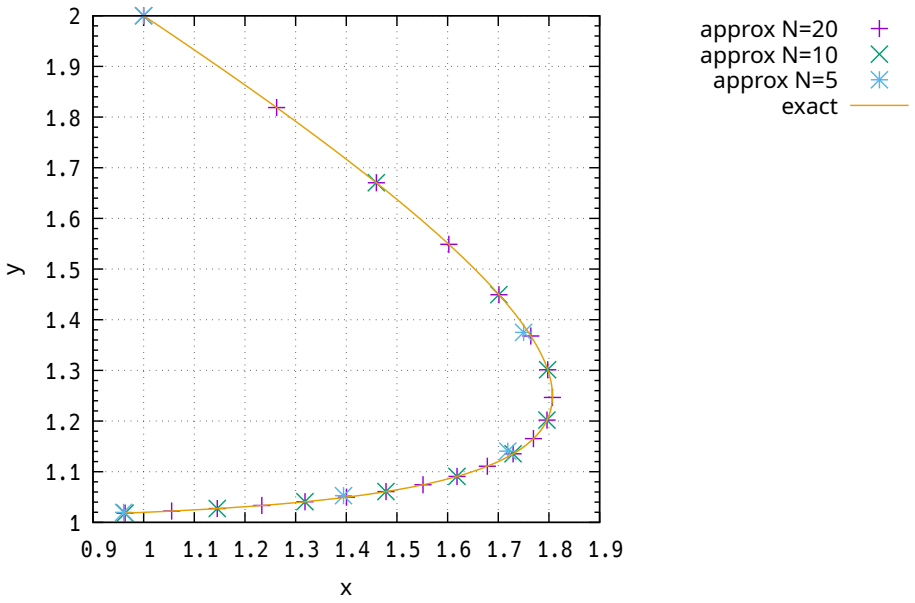

Fig. 2: ルンゲ・クッタ法を用いた連立常微分方程式の解

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．

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

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

2.2　解法の検討

$\dfrac{dy}{dx} = z$ 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 の連立常微分方程式を解くものであるため，$y \to x, z \to y, x \to t$ 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.3 に示す．ここで solve_sod1() はルンゲ・クッタ法を用いて連立常微分方程式を解く関数である．

List. 3: 2nd.c

```
1  typedef double (*F)(double, double, double);
2  typedef struct {
3      F f;
4      F g;
5      double x0;
6      double y0;
7      double width;
8      double t0;
9      double t_inf;
10 } Condition;
11
12 double f(double y, double z, double x) {
13     UNUSED(y);
14     UNUSED(x);
15     return z;
16 }
17 double g(double y, double z, double x) {
18     UNUSED(x);
19     return 6 * y - 5 * z;
20 }
21
22 int main(void) {
23     Matrix *matrix, *res;
24     size_t i;
25     Condition cond;
26
27     // データの読み込みと検証
28
29     cond = (Condition){
30         .f = f,
31         .g = g,
32         .x0 = matrix->data[0][0],
33         .y0 = matrix->data[0][1],
34         .width = matrix->data[0][2],
35         .t0 = matrix->data[0][3],
36         .t_inf = matrix->data[0][4],
37     };
38     printf("#step width: %f\n", cond.width);
39     res = solve_sod1(&cond);
```

```
40      if (res == NULL) {
41          freeMatrix(matrix);
42          return EXIT_FAILURE;
43      }
44
45      for (i = 0; i < res->col; i++) {
46          printf("%.6f,%.6f,%.6f\n", res->data[0][i], res->data[1][i],
47                 res->data[2][i]);
48      }
49
50      freeMatrix(res);
51      freeMatrix(matrix);
52
53      return EXIT_SUCCESS;
54  }
```

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．$x = 1$ での y の値を求めるために，List. 4 を入力に与えた．

List. 4: 2ndin.csv

```
1  0,7,0.1,0,1
```

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 で実行した結果を List. 5 に示す．

List. 5: 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 2.718282 であるため，数値解と解析解は高い精度で一致していることがわかる．次に，List. 5 を解析解と重ねてプロットした結果を Fig. 3 に示す．

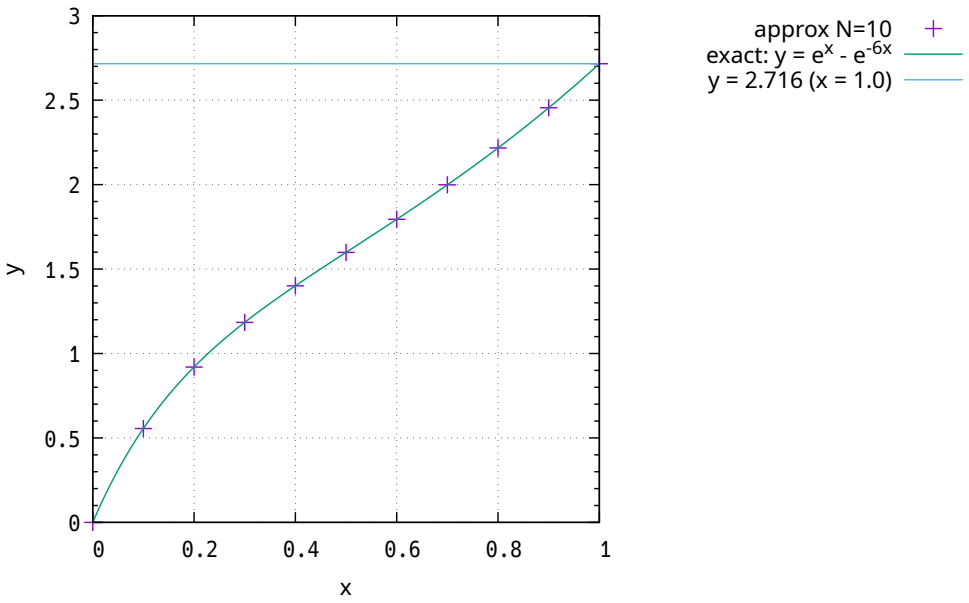

Fig. 3: 高次微分方程式の解

Fig. 3 より，解析解と数値解は高い精度で一致していることがわかる．

3 感想

参考文献

[1]