10. 常微分方程式の初期値問題

関数 y(x)に関する一階常微分方程式

$$\frac{dy}{dx} = f(x,y)$$

を初期条件

$$y(x_0) = y_0 \qquad (x = x_0)$$

のもとで数値的に解くときには，x の増分を h，x のステップを n=0, 1, 2…として

$$x_{n+1} = x_n + h$$

に対する y の値 y_{n+1} を順次求めていく。

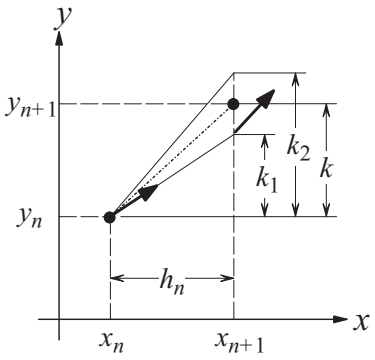
10.1 ルンゲ・クッタ法

(x_n, y_n)まで求められたとき，次のステップにおける y 値 y_{n+1} を求める 2 次および 4 次ルンゲ・クッタ法は以下のように書ける。

（1）2 次のルンゲ・クッタ法

$$\begin{cases} k_1 = h\ f(x_n, y_n) \\ k_2 = h\ f(x_n + h,\ y_n + k_1) \end{cases}$$

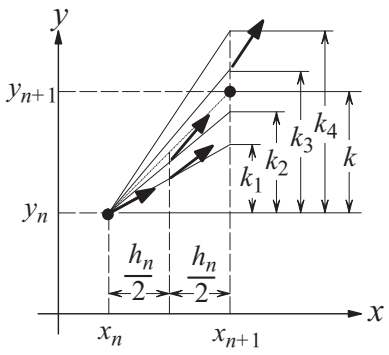
$$y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2)$$



（2）4 次のルンゲ・クッタ法

$$\begin{cases} k_1 = h\ f(x_n, y_n) \\ k_2 = h\ f(x_n + h/2,\ y_n + k_1/2) \\ k_3 = h\ f(x_n + h/2,\ y_n + k_2/2) \\ k_4 = h\ f(x_n + h,\ y_n + k_3) \end{cases}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$



ルンゲ・クッタ法は陽解法であるためプログラミングの手間が少なく済み，実用的によく用いられる数値解法である。

【問題】

常微分方程式

$$\frac{dy}{dx} = -\frac{2y}{x+2}$$

の初期条件

$$x_0 = 0 \text{ のとき } y(x_0) = y_0 = 1$$

のもとでの解は，

$$y(x) = \frac{4}{(x+2)^2}$$

である。

x_0=0 から x_{max}=2 までの数値解を 2 次および 4 次ルンゲ・クッタ法により計算し，x_{max} =2 のときの誤差 E を求めよ。その際 p 次の方法ごとに，x の区間$[x_0, x_{max}]$の分割数を倍々にして増分 $h=(x_{max}-x_0)/n$ を順次 1/2 にしていくとき，両対数グラフの横軸に h，縦軸に|E|をプロットし，p 次の数値解法の誤差は $E=O(h^p)\approx Ch^p$ となっていることを確認せよ。

【プログラム10.1.1】　　2次ルンゲ・クッタ法

```
 1 /* Runge-Kutta method */
 2
 3 #include <stdio.h>
 4
 5 float f(float x, float y)
 6 {
 7       return(-2*y/(x+2));
 8 }
 9
10 void runge(float *x, float *y, float h)  /* 2nd-order */
11 {
12       float xn,yn,k1,k2;
13
14       xn=*x;
15       yn=*y;
16
17       k1=h*f(xn, yn);
18       k2=h*f(xn+h, yn+k1);
19
20       *y +=(k1+k2)/2;
21       *x +=h;
22 }
23
24 void main(void)
25 {
26       float x0,y0,xmax,h,x,y,error;
27       int n,i;
28
29       printf("Runge-Kutta method¥n");
30       printf("initial values (x0, y0) and maximum x: x0, y0, xmax ? ");
31       scanf("%g%g%g", &x0, &y0, &xmax);
32       while(printf("¥nnumber of intervals: n ? "), scanf("%d",&n)!=EOF){
33               h= (xmax-x0)/n;
34               printf("h=%g¥n",h);
35               printf(" i¥tx¥t¥ty¥n 0 %15.6e %15.6e¥n",x0,y0);
36               x=x0; y=y0;
37               for(i=1;i<=n;i++){
38                       runge(&x,&y,h);
39                       printf("%2d %15.6e %15.6e¥n", i, x, y);
40               }
41               error= y - 4/((x+2)*(x+2));
42               printf(" * error =%15.6e¥n",error);
43       }
44 }
```

【プログラム10.1.2】　　4次ルンゲ・クッタ法

2次ルンゲ・クッタ法プログラム10.1.1において，関数 runge を次のものに置き換えればよい。

```
 1 void runge(float *x, float *y, float h)  /* 4th-order */
 2 {
 3       float xn,yn,k1,k2,k3,k4;
 4
 5       xn=*x;
 6       yn=*y;
 7
 8       k1=h*f(xn, yn);
 9       k2=h*f(xn+h/2, yn+k1/2);
10       k3=h*f(xn+h/2, yn+k2/2);
11       k4=h*f(xn+h, yn+k3);
12
13       *y +=(k1+2*k2+2*k3+k4)/6;
14       *x +=h;
15 }
```

【実行例】

まず 4 次ルンゲ・クッタ法の実行例を示す。

```
------------------------実行開始-----------------------
Runge-Kutta method
initial values (x0, y0) and maximum x: x0, y0, xmax ? 0 1 2

number of intervals: n ? 1
h=2
 i       x                y
 0    0.000000e+00    1.000000e+00
 1    2.000000e+00    2.777778e-01
 * error =   2.777779e-02

number of intervals: n ? 2
h=1
 i       x                y
 0    0.000000e+00    1.000000e+00
 1    1.000000e+00    4.466667e-01
 2    2.000000e+00    2.514399e-01
 * error =   1.439899e-03

number of intervals: n ? 4
h=0.5
 i       x                y
 0    0.000000e+00    1.000000e+00
 1    5.000000e-01    6.401235e-01
 2    1.000000e+00    4.445596e-01
 3    1.500000e+00    3.266242e-01
 4    2.000000e+00    2.500749e-01
 * error =   7.486343e-05

number of intervals: n ? 8
h=0.25
 i       x                y
 0    0.000000e+00    1.000000e+00
 1    2.500000e-01    7.901288e-01
```

```
 2    5.000000e-01    6.400067e-01
 3    7.500000e-01    5.289324e-01
 4    1.000000e+00    4.444508e-01
 5    1.250000e+00    3.787040e-01
 6    1.500000e+00    3.265358e-01
 7    1.750000e+00    2.844491e-01
 8    2.000000e+00    2.500041e-01
 * error =   4.112720e-06

number of intervals: n ? 16
h=0.125
 i       x               y
 0    0.000000e+00    1.000000e+00
 1    1.250000e-01    8.858133e-01
 2    2.500000e-01    7.901238e-01
 3    3.750000e-01    7.091416e-01
 4    5.000000e-01    6.400004e-01
 5    6.250000e-01    5.804992e-01
 6    7.500000e-01    5.289260e-01
 7    8.750000e-01    4.839323e-01
 8    1.000000e+00    4.444448e-01
 9    1.125000e+00    4.096003e-01
10    1.250000e+00    3.786985e-01
11    1.375000e+00    3.511663e-01
12    1.500000e+00    3.265309e-01
13    1.625000e+00    3.043998e-01
14    1.750000e+00    2.844447e-01
15    1.875000e+00    2.663894e-01
16    2.000000e+00    2.500002e-01
 * error =   2.086163e-07

number of intervals: n ? 32
h=0.0625
 i       x               y
 0    0.000000e+00    1.000000e+00
 1    6.250000e-02    9.403122e-01
 2    1.250000e-01    8.858131e-01
 3    1.875000e-01    8.359184e-01
 4    2.500000e-01    7.901235e-01
 5    3.125000e-01    7.479913e-01
 6    3.750000e-01    7.091414e-01
 7    4.375000e-01    6.732414e-01
 8    5.000000e-01    6.400001e-01
 9    5.625000e-01    6.091613e-01
10    6.250000e-01    5.804990e-01
               :               :
               :               :
28    1.750000e+00    2.844445e-01
29    1.812500e+00    2.751949e-01
30    1.875000e+00    2.663892e-01
31    1.937500e+00    2.579996e-01
32    2.000000e+00    2.500001e-01
 * error =   5.960464e-08

number of intervals: n ? ^Z (controlとZを同時に押す) ↲(Enter Keyを押す)
                                                        …… 入力終了コードの入力
------------------------おしまい-----------------------
```

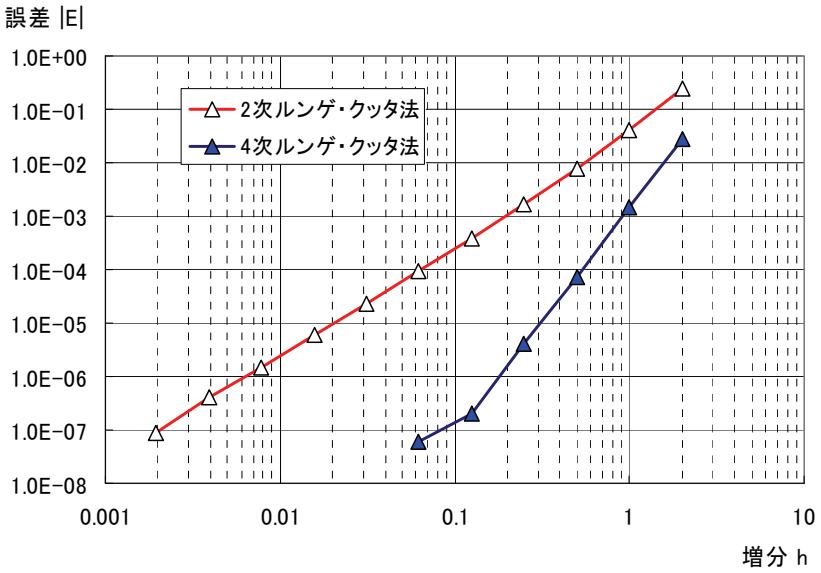
4次ルンゲ・クッタ法を実行させたときの，分割数n,増分h,誤差Eを表にまとめると，次のようになる。

n	h	E
1	2	2.777779E-02
2	1	1.439899E-03
4	0.5	7.486343E-05
8	0.25	4.112720E-06
16	0.125	2.086163E-07
32	0.0625	5.960464E-08

つぎに，2次ルンゲ・クッタ法を実行させたときの，分割数n,増分h,誤差Eを以下の表にまとめる。

n	h	error
1	2	2.500000E-01
2	1	4.166666E-02
4	0.5	7.872015E-03
8	0.25	1.697481E-03
16	0.125	3.940761E-04
32	0.0625	9.500980E-05
64	0.03125	2.333522E-05
128	0.015625	5.900860E-06
256	0.0078125	1.490116E-06
512	0.00390625	4.172325E-07
1024	0.001953125	8.940697E-08

下図は，上記の表における誤差を，両対数グラフ（横軸 h，縦軸|E|）にまとめたものである。p 次の数値解法の主要な誤差項は $E=O(h^p)\approx Ch^p$ であるので，log をとると $\log|E|\approx\log|C|+p(\log h)$ となり，誤差|E|は勾配 p の直線上にほぼ並ぶはずである。実際に数値計算を行ったときの誤差がそうなっていることが下図において確かめられる。



図表から，以下のことがわかる。

- 増分 h が同じ値のときには，4 次のルンゲ・クッタ法のほうが誤差が少ない。
- 誤差を 10^{-7} 以下にするためには，2 次のルンゲ・クッタ法では増分 h を 0.001953125 まで小さくして x を 1024 ステップ進めなくてはならないのに対し，4 次のルンゲ・クッタ法では増分 h を 0.0625 として x を 32 ステップ進めるだけで済む。高精度解法においては，1 ステップ進めるのに要する演算量は増えるが，全計算量を考えると経済的であることが納得されよう。

10.2 オイラー--台形則法

予測子・修正子法にはいろいろな陽解法と陰解法の組み合わせがあるが，予測子として最も簡単な陽解法であるオイラー法（1 次精度）

$$y_{n+1} = y_n + h\, f(x_n, y_n)$$

を用い，修正子として比較的簡単な陰解法である台形則（2 次精度）

$$y_{n+1} = y_n + (h/2)\ \left\{ f(x_n, y_n) + f(x_{n+1}, y_{n+1}) \right\}$$

を用いるものがある。後者を不動点反復法

$$y_{n+1}^{(k+1)} = y_n + (h/2)\ \left\{ f(x_n, y_n) + f(x_{n+1}, y_{n+1}^{(k)}) \right\} \qquad (k = 0,1,2,\cdots)$$

により解く。予測子で得られた値を $y_{n+1}^{(0)}$ にセットして k に関する反復を行い，収束したときの $y_{n+1}^{(k+1)}$ を第 n ステップにおける解 y_{n+1} とする。

【アルゴリズム】

```
x:=x0                                          {初期値の設定}
y:=y0
h:=(xmax-x0)/N                                 {増分 h の計算（N: 全ステップ数）}

for i=1 to N                                   {N ステップ分の数値計算を行う}
    y^old:= y
    f^old:= f(x, y^old)
    y := y + h f^old          {予測子(オイラー法)により y を求め修正子での初期値とする}
    x := x + h                                 {x を増分 h だけ進める}
    for k=1 to KMAX                  {修正子(台形則)により y を反復計算する}
        yk:=y                                  {yk: 直前の反復における y の値}
        y := y^old + (h/2)(f^old+f(x,y))
        if |y-yk|<ε  then break                {収束したら修正子のループから出る}
    end
end
```

【問題】

下記の 2 個の常微分方程式

1) $\dfrac{dy}{dx} = -\dfrac{4y}{x+2}$　　初期条件：$x_0 = 0$ のとき $y(x_0) = 1$　　厳密解：$y(x) = \dfrac{16}{(x+2)^4}$

2) $\dfrac{dy}{dx} = \dfrac{4y}{x+2}$　　初期条件：$x_0 = 0$ のとき $y(x_0) = 1$　　厳密解：$y(x) = \dfrac{1}{16}(x+2)^4$

に関して，x_0=0 から x_{max}=2 までの数値解をオイラー--台形則により計算し，x_{max} =2 のときの誤差 E を求めよ。10.1 節の問題と同様に，各常微分方程式ごとに，x の区間$[x_0, x_{max}]$の分割数を倍々にして増分 $h=(x_{max}-x_0)/n$ を順次 1/2 にしていくとき，両対数グラフの横軸に h，縦軸に|E|をプロットし，本数値解法（2 次）の誤差は $E=O(h^2)\approx Ch^2$ となっていることを確認せよ。

【プログラム 10.2.1】

```
1 /* predictor-corrector method       */
2 /*    predictor  Euler Method       */
3 /*    corrector  Trapezoidal Method */
4
```

```
 5 #include <stdio.h>
 6 #include <math.h>
 7
 8 #define LINE 8
 9 #define KMAX 50
10
11 float f(float x, float y)
12 {
13 /*      return(-4*y/(x+2)); */
14       return(4*y/(x+2));
15 }
16
17 int correct(float *x, float *y, float h, float eps)
18 {
19       float xo,yo,fo,yk;
20       int   k;
21
22       xo=*x;
23       yo=*y;
24
25       fo= f(xo, yo);
26       *y += h*fo;
27       *x += h;
28       for(k=1;k<KMAX;k++){
29               yk= *y;
30               *y=yo+(h/2)*(fo+f(*x,*y));
31               if(fabs(*y-yk)<eps)
32                       return(k);
33       }
34       printf("divergent?¥n");
35       return(k);
36       exit(1);
37 }
38
39 void main(void)
40 {
41       float x0,y0,xmax,eps,h,x,y,error;
42       int m,n,i,j,k;
43
44       printf("Predictor-Corrector method¥n");
45       printf("initial values (x0, y0) and maximum x: x0, y0, xmax ? ");
46       scanf("%g%g%g", &x0, &y0, &xmax);
47       printf("tolerance: eps ? ");
48       scanf("%g", &eps);
49
50       while(printf("¥ndivision number: n ? "), scanf("%d",&n)!=EOF){
51             m=n/LINE;       /* 全時間ステップのうち行数LINEだけ出力する場合，m行おきに出力 */
52             h=(xmax-x0)/n;
53             printf("¥nh=%-10.6g¥tn=%d¥n¥tx¥t¥ty¥n", h, n);
54             printf("%3d%15.6e %15.6e¥n", 0, x0, y0);
55             x=x0; y=y0;
56             for(i=1;i<=n;i++){
57                  k=correct(&x,&y,h,eps);
58                  if( m==0 || i%m==0 )                    /* m行ごとに結果を出力する */
59                     printf("%3d%15.6e %15.6e %5d¥n", i, x, y, k);
60             }
61 /*              error= y - 16/((x+2)*(x+2)*(x+2)*(x+2)); */
```

```
62                error= y -  ((x+2)*(x+2)*(x+2)*(x+2))/16;
63                printf(" * error =%15.6e¥n",error);
64        }
65 }
```

【実行例】

常微分方程式１の実行例を示す。

```
-------------------------実行開始-----------------------
Predictor-Corrector method
initial values (x0, y0) and maximum x: x0, y0, xmax ? 0 1 2
tolerance: eps ? 1.0e-7

division number: n ? 4
h=0.5
         x                y
  0   0.000000e+00    1.000000e+00
  1   5.000000e-01    3.571428e-01    18
  2   1.000000e+00    1.607143e-01    14
  3   1.500000e+00    8.333331e-02    12
  4   2.000000e+00    4.761902e-02    10
 * error =  -1.488098e-02

division number: n ? 8
h=0.25
         x                y
  0   0.000000e+00    1.000000e+00
  1   2.500000e-01    6.136364e-01    11
  2   5.000000e-01    3.977273e-01    10
  3   7.500000e-01    2.692308e-01     9
  4   1.000000e+00    1.888112e-01     8
  5   1.250000e+00    1.363636e-01     8
  6   1.500000e+00    1.009615e-01     7
  7   1.750000e+00    7.635747e-02     7
  8   2.000000e+00    5.882352e-02     6
 * error =  -3.676478e-03

division number: n ? 16
h=0.125
         x                y
  0   0.000000e+00    1.000000e+00
  2   2.500000e-01    6.217105e-01     7
  4   5.000000e-01    4.066986e-01     7
  6   7.500000e-01    2.771739e-01     6
  8   1.000000e+00    1.953785e-01     6
 10   1.250000e+00    1.416667e-01     5
 12   1.500000e+00    1.052166e-01     5
 14   1.750000e+00    7.977615e-02     5
 16   2.000000e+00    6.158359e-02     5
 * error =  -9.164065e-04

division number: n ? 32
h=0.0625
         x                y
  0   0.000000e+00    1.000000e+00
```

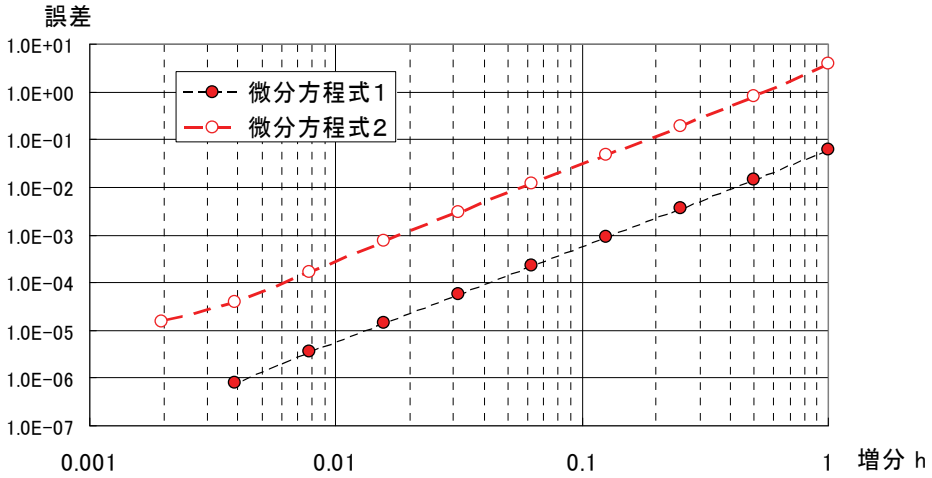
```
  4   2.500000e-01    6.236538e-01     5
  8   5.000000e-01    4.088787e-01     5
 12   7.500000e-01    2.791179e-01     5
 16   1.000000e+00    1.969945e-01     4
 20   1.250000e+00    1.429769e-01     4
 24   1.500000e+00    1.062714e-01     4
 28   1.750000e+00    8.062583e-02     4
 32   2.000000e+00    6.227105e-02     4
 * error =  -2.289489e-04

division number: n ? 64
h=0.03125
        x               y
  0   0.000000e+00    1.000000e+00
  8   2.500000e-01    6.241351e-01     4
 16   5.000000e-01    4.094200e-01     4
 24   7.500000e-01    2.796014e-01     4
 32   1.000000e+00    1.973969e-01     3
 40   1.250000e+00    1.433036e-01     3
 48   1.500000e+00    1.065346e-01     3
 56   1.750000e+00    8.083801e-02     3
 64   2.000000e+00    6.244281e-02     3
 * error =  -5.719066e-05

division number: n ? 128
h=0.015625
        x               y
  0   0.000000e+00    1.000000e+00
 16   2.500000e-01    6.242551e-01     3
 32   5.000000e-01    4.095551e-01     3
 48   7.500000e-01    2.797221e-01     3
 64   1.000000e+00    1.974974e-01     3
 80   1.250000e+00    1.433851e-01     3
 96   1.500000e+00    1.066003e-01     3
112   1.750000e+00    8.089098e-02     3
128   2.000000e+00    6.248569e-02     2
 * error =  -1.431257e-05

division number: n ? 256
h=0.007812
        x               y
  0   0.000000e+00    1.000000e+00
 32   2.500000e-01    6.242852e-01     3
 64   5.000000e-01    4.095890e-01     3
 96   7.500000e-01    2.797524e-01     3
128   1.000000e+00    1.975226e-01     2
160   1.250000e+00    1.434056e-01     2
192   1.500000e+00    1.066168e-01     2
224   1.750000e+00    8.090425e-02     2
256   2.000000e+00    6.249643e-02     2
 * error =  -3.568828e-06
        :
        :
        :
division number: n ? ^Z (controlとZを同時に押す)  ↲(Enter Keyを押す)
                                                        ……  入力終了コードの入力
------------------------おしまい-----------------------
```

下図は常微分方程式１と２の誤差を両対数グラフ（横軸 h，縦軸|E|）にまとめたものである。修正子の台形則は 2 次精度なので，誤差|E|は勾配 2 の直線上にほぼ並んでいることが確かめられる。



参考文献

高倉葉子：数値計算の基礎―解法と誤差―，コロナ社（2007）
森口繁一，伊理正夫，武市正人 編：C による算法痛論，東京大学出版会（2000）