Rössler Oscillator

1. The Rössler System is a three dimensional ordinary differential equation of the form $dy/dt = f(y)$ with a given initial condition $y(t_0) = y_0$ where $y(t) \in \mathbf{R}^3$ and

$$f(y) = \begin{bmatrix} -y_2 - y_3 \\ y_1 + ay_2 \\ b + y_3(y_1 - c) \end{bmatrix} \qquad \text{where} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

with $a = b = 0.2$ and $c = 5.7$. Let Y^h be an approximation of $y(T)$ obtained using a step size of $h = (T - t_0)/n$. Define the error

$$E_h = \|Y^h - y(T)\| = \left\{ \sum_{i=1}^{3} \left(Y_i^h - y_i(T)\right)^2 \right\}^{1/2}.$$

Show that if $E_h \le Kh^k$ then

$$\|Y^h - Y^{h/2}\| \le K\Big\{1 + \frac{1}{2^k}\Big\}h^k.$$

Estimate using the triangle inequality as

$$\begin{aligned} \|Y^h - Y^{h/2}\| &= \|Y^h - Y(T) + y(T) - Y^{h/2}\| \\ &\le \|Y^h - Y(T)\| + \|Y^{h/2} - Y(T)\| \\ &\le Kh^k + K(h/2)^k = K\Big\{1 + \frac{1}{2^k}\Big\}h^k. \end{aligned}$$

2. Write a program to approximate $y(T)$ using Euler's forward difference method and the initial condition

$$y_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \text{when} \qquad t_0 = 0 \quad \text{and} \quad T = 1.$$

Compute Y^h for $h = T/n$ with $n = 64, 128, 256, 512, \ldots, 65536$.

The program

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <string.h>
4
5 #include "force.h"
6
7 double a=0.2,b=0.2,c=5.7;
```

```
8
9  void euler(int N,double T,double Y0[3],double Y[3]){
10     double h=T/N;
11     memcpy(Y,Y0,sizeof(double)*3);
12     for(int n=0;n<N;n++){
13         double F[3];
14         f(Y,F);
15         for(int i=0;i<3;i++) Y[i]=Y[i]+h*F[i];
16     }
17 }
18
19 int main(){
20     double Y0[3]={1,1,1};
21     printf("#%5s %21s %21s %21s\n","N","y1","y2","y3");
22     for(int N=64;N<=65536;N*=2){
23         double Y[3];
24         euler(N,1.0,Y0,Y);
25         printf("%6d %21.14e %21.14e %21.14e\n",N,Y[0],Y[1],Y[2]);
26     }
27     return 0;
28 }
```

produces the output

```
#    N                    y1                    y2                    y3
    64 -5.82142229843936e-01  1.47021847490614e+00  3.62737059405978e-02
   128 -5.80645290074130e-01  1.46432834950022e+00  3.66826634336048e-02
   256 -5.79873627756266e-01  1.46139078338438e+00  3.68968727671971e-02
   512 -5.79482035819852e-01  1.45992393838989e+00  3.70063907275780e-02
  1024 -5.79284804461229e-01  1.45919100833076e+00  3.70617504541436e-02
  2048 -5.79185830543156e-01  1.45882466739671e+00  3.70895801727147e-02
  4096 -5.79136254101054e-01  1.45864152807655e+00  3.71035324538560e-02
  8192 -5.79111443518841e-01  1.45854996621854e+00  3.71105179446109e-02
 16384 -5.79099032638633e-01  1.45850418724196e+00  3.71140130268724e-02
 32768 -5.79092825801433e-01  1.45848129824203e+00  3.71157611521412e-02
 65536 -5.79089722033566e-01  1.45846985386418e+00  3.71166353607999e-02
```

The include file `force.h` contains declarations for the force and its derivatives which appear in `force.c` and will also be used in the subsequent questions. The derivatives were created with Maple's code generator using the script

```
1 restart;
2 with(codegen):
3 K:=5;
4 S1:=[seq(y||i(t)=Y[i],i=1..3)];
5 S2:=[seq(diff(y||i(t),t)=f[i],i=1..3)];
6 f:=[-y2(t)-y3(t),y1(t)+a*y2(t),b+y3(t)*(y1(t)-c)];
7 force[1]:=f;
8 for k from 2 to K
```

```
do
    force[k]:=subs(S2,diff(force[k-1],t));
od;
for k from 1 to K
do
    F:=vector(subs(S1,force[k]));
    fname:=sprintf("jet%d.i",k-1);
    C(F,optimized,filename=fname);
od;
```

This script produces the files `jet0.i, jet1.i, ..., jet4.i` with the generated C code for the force and its derivatives up to order 4. In particular, the files are

`jet0.i`—The force function.

```
      F[0] = -Y[1]-Y[2];
      F[1] = Y[0]+a*Y[1];
      F[2] = b+Y[2]*(Y[0]-c);
```

`jet1.i`—Maple generated code for first derivative.

```
      t1 = a*Y[1];
      t2 = Y[0]-c;
      t3 = Y[2]*t2;
      F[0] = -Y[0]-t1-b-t3;
      F[1] = -Y[1]-Y[2]+a*(Y[0]+t1);
      F[2] = (b+t3)*t2+Y[2]*(-Y[1]-Y[2]);
```

`jet2.i`—Maple generated code for second derivative.

```
      t1 = a*Y[1];
      t3 = a*(Y[0]+t1);
      t4 = Y[0]-c;
      t5 = Y[2]*t4;
      t6 = b+t5;
      t7 = t6*t4;
      t8 = -Y[1]-Y[2];
      t9 = Y[2]*t8;
      F[0] = Y[1]+Y[2]-t3-t7-t9;
      F[1] = -Y[0]-t1-b-t5+a*(-Y[1]-Y[2]+t3);
      F[2] = (t7+t9)*t4+2.0*t6*t8+Y[2]*(-Y[0]-t1-b-t5);
```

`jet3.i`—Maple generated code for third derivative.

```
      t1 = a*Y[1];
      t2 = Y[0]-c;
      t3 = Y[2]*t2;
      t5 = a*(Y[0]+t1);
      t7 = a*(-Y[1]-Y[2]+t5);
```

```
      t8 = b+t3;
      t9 = t8*t2;
      t10 = -Y[1]-Y[2];
      t11 = Y[2]*t10;
      t12 = t9+t11;
      t13 = t12*t2;
      t15 = 2.0*t8*t10;
      t16 = -Y[0]-t1-b-t3;
      t17 = Y[2]*t16;
      F[0] = Y[0]+t1+b+t3-t7-t13-t15-t17;
      F[1] = Y[1]+Y[2]-t5-t9-t11+a*(-Y[0]-t1-b-t3+t7);
      F[2] = (t13+t15+t17)*t2+3.0*t12*t10+3.0*t8*t16
            +Y[2]*(Y[1]+Y[2]-t5-t9-t11)
;
```

`jet4.i`—Maple generated code for fourth derivative.

```
      t1 = a*Y[1];
      t3 = a*(Y[0]+t1);
      t4 = Y[0]-c;
      t5 = Y[2]*t4;
      t6 = b+t5;
      t7 = t6*t4;
      t8 = -Y[1]-Y[2];
      t9 = Y[2]*t8;
      t11 = a*(-Y[1]-Y[2]+t3);
      t13 = a*(-Y[0]-t1-b-t5+t11);
      t14 = t7+t9;
      t15 = t14*t4;
      t17 = 2.0*t6*t8;
      t18 = -Y[0]-t1-b-t5;
      t19 = Y[2]*t18;
      t20 = t15+t17+t19;
      t21 = t20*t4;
      t23 = 3.0*t14*t8;
      t25 = 3.0*t6*t18;
      t26 = Y[1]+Y[2]-t3-t7-t9;
      t27 = Y[2]*t26;
      F[0] = -Y[1]-Y[2]+t3+t7+t9-t13-t21-t23-t25-t27;
      F[1] = Y[0]+t1+b+t5-t11-t15-t17-t19+a*(Y[1]+Y[2]-t3-t7-t9+t13);
      F[2] = (t21+t23+t25+t27)*t4+4.0*t20*t8+6.0*t14*t18
            +4.0*t6*t26+Y[2]*(Y[0]+t1+b+t5-t11-t15-t17-t19);
```

Finally, the file `force.c` defines the functions `f, df, ..., ddddf` and is given by

```
#include "force.h"
```

```

extern double a,b,c;

#define decl_tmpvars double t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,\
t11,t12,t13,t14,t15,t16,t17,t18,t19,t20,\
t21,t22,t23,t24,t25,t26,t27,t28,t29,t30

void f(double Y[3],double F[3]){
decl_tmpvars;
#include "jet0.i"
}

void df(double Y[3],double F[3]){
decl_tmpvars;
#include "jet1.i"
}

void ddf(double Y[3],double F[3]){
decl_tmpvars;
#include "jet2.i"
}

void dddf(double Y[3],double F[3]){
decl_tmpvars;
#include "jet3.i"
}

void ddddf(double Y[3],double F[3]){
decl_tmpvars;
#include "jet4.i"
}
```

3. Graph $\log \|Y^h - Y^{h/2}\|$ versus $\log h$ to verify the order of convergence for Euler's method numerically.

The code to compute $\|Y^h - Y^{h/2}\|$ is

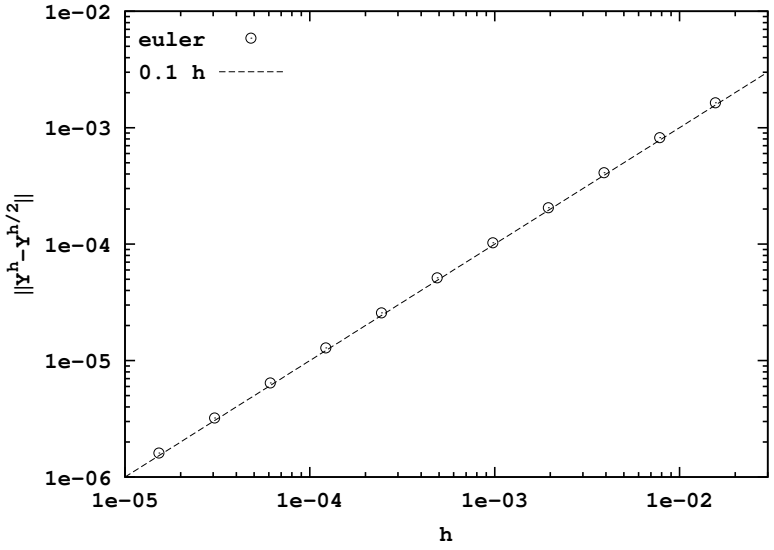
```
#include <stdio.h>
#include <math.h>
#include <string.h>

#include "force.h"

double a=0.2,b=0.2,c=5.7;

```

```
9  void euler(int N,double T,double Y0[3],double Y[3]){
10     double h=T/N;
11     memcpy(Y,Y0,sizeof(double)*3);
12     for(int n=0;n<N;n++){
13         double F[3];
14         f(Y,F);
15         for(int i=0;i<3;i++) Y[i]=Y[i]+h*F[i];
16     }
17 }
18
19 double l2err(double Yold[3],double Y[3]){
20     double r=0.0;
21     for(int i=0;i<3;i++){
22         double t=Yold[i]-Y[i];
23         r+=t*t;
24     }
25     return sqrt(r);
26 }
27
28 int main(){
29     double Y0[3]={1,1,1};
30     printf("#%5s %21s %21s\n","N","h","||Y(h)-Y(h/2)||");
31     double Yold[3];
32     for(int N=64;N<=131072;N*=2){
33         double Y[3];
34         euler(N,1.0,Y0,Y);
35         if(N>64){
36             printf("%6d %21.14e %21.14e\n",N/2,2.0/N,l2err(Yold,Y));
37         }
38         memcpy(Yold,Y,sizeof(double)*3);
39     }
40     return 0;
41 }
```

which produces the output

```
     #    N                     h       ||Y(h)-Y(h/2)||
         64  1.56250000000000e-02  6.09111255871896e-03
        128  7.81250000000000e-03  3.04477307140502e-03
        256  3.90625000000000e-03  1.52216052575486e-03
        512  1.95312500000000e-03  7.61019960140117e-04
       1024  9.76562500000000e-04  3.80494427887568e-04
       2048  4.88281250000000e-04  1.90243266235616e-04
       4096  2.44140625000000e-04  9.51206387344385e-05
       8192  1.22070312500000e-04  4.75600698554388e-05
      16384  6.10351562500000e-05  2.37799724032351e-05
      32768  3.05175781250000e-05  1.18899705746696e-05
      65536  1.52587890625000e-05  5.94498138123117e-06
```

A log-log plot of the output with a comparison to the line Kh where $K = 0.1$ shows that the Euler method is first order.



4. Compute Y^h using the Taylor methods of order 2 and 3 and verify the order of convergence by graphing $\log \|Y^h - Y^{h/2}\|$ versus $\log h$.

The Taylor's method is given by the program

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

#include "force.h"

double a=0.2,b=0.2,c=5.7;

void taylor2(int N,double T,double Y0[3],double Y[3]){
    double h=T/N;
    memcpy(Y,Y0,sizeof(double)*3);
    for(int n=0;n<N;n++){
        double F[3],dF[3];
        f(Y,F); df(Y,dF);
        for(int i=0;i<3;i++)
            Y[i]=Y[i]+h*(F[i]+h/2*dF[i]);
    }
}

void taylor3(int N,double T,double Y0[3],double Y[3]){
    double h=T/N;
    memcpy(Y,Y0,sizeof(double)*3);
```

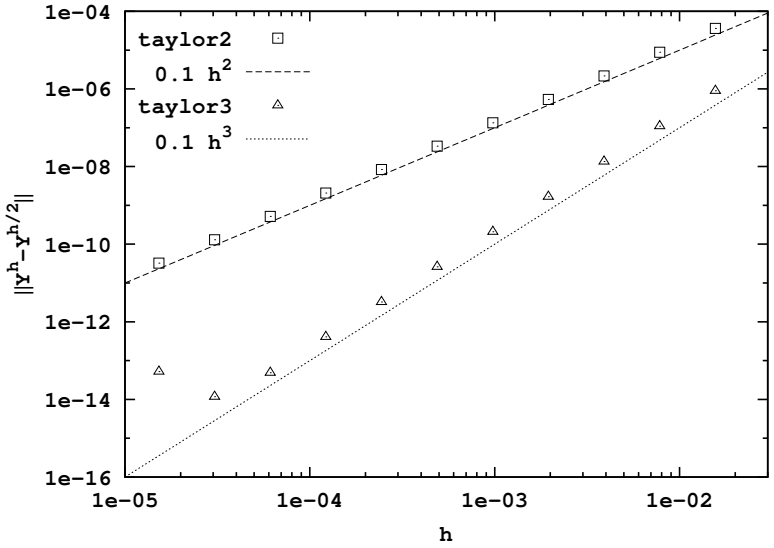
```
23      for(int n=0;n<N;n++){
24          double F[3],dF[3],ddF[3];
25          f(Y,F); df(Y,dF); ddf(Y,ddF);
26          for(int i=0;i<3;i++)
27              Y[i]=Y[i]+h*(F[i]+h/2*(dF[i]+h/3*ddF[i]));
28      }
29  }
30
31  double l2err(double Yold[3],double Y[3]){
32      double r=0.0;
33      for(int i=0;i<3;i++){
34          double t=Yold[i]-Y[i];
35          r+=t*t;
36      }
37      return sqrt(r);
38  }
39
40  int main(){
41      double Y0[3]={1,1,1};
42      double Yold2[3],Yold3[3];
43
44      printf("#%5s %21s %21s %21s\n","N","h","Taylor2-Error","Taylor3-Error");
45      for(int N=64;N<=131072;N*=2){
46          double Y2[3],Y3[3];
47          taylor2(N,1.0,Y0,Y2);
48          taylor3(N,1.0,Y0,Y3);
49          if(N>64){
50              double e2=l2err(Yold2,Y2);
51              double e3=l2err(Yold3,Y3);
52              printf("%6d %21.14e %21.14e %21.14e\n",N/2,2.0/N,e2,e3);
53          }
54          memcpy(Yold2,Y2,sizeof(double)*3);
55          memcpy(Yold3,Y3,sizeof(double)*3);
56      }
57
58
59      return 0;
60  }
```

which produces the output

```
    #     N                     h         Taylor2-Error         Taylor3-Error
       64  1.56250000000000e-02  3.07901086884656e-05  1.10597982882957e-06
      128  7.81250000000000e-03  7.66215921638294e-06  1.35531043360058e-07
      256  3.90625000000000e-03  1.91229886655599e-06  1.67735299067911e-08
      512  1.95312500000000e-03  4.77736738256374e-07  2.08626243461771e-09
```

```
 1024  9.76562500000000e-04  1.19395943336313e-07  2.60134021263161e-10
 2048  4.88281250000000e-04  2.98444406198116e-08  3.24739708178006e-11
 4096  2.44140625000000e-04  7.46057232388297e-09  4.05912985483117e-12
 8192  1.22070312500000e-04  1.86507308388945e-09  5.11341127803845e-13
16384  6.10351562500000e-05  4.66249152108445e-10  6.13114355446638e-14
32768  3.05175781250000e-05  1.16553816970009e-10  1.77773879513407e-14
65536  1.52587890625000e-05  2.91511712180469e-11  2.24290168707128e-14
```

A log-log plot of the data verifies that the second order and third order Taylor's methods are indeed second and third order.



Note the error represented by the data point for the third-order Taylor method when $h = 1/65536$ is greater than when $h = 1/32768$. This is because at this small step size the method becomes so accurate that only rounding contributes to the error.

5. Approximate $y(10)$ to four decimal places. Indicate what method you used and how many steps were needed. Is it possible to achieve this accuracy using Euler's method? Can you find $y(100)$?

The program

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <string.h>
4
5  #include "force.h"
6
7  double a=0.2,b=0.2,c=5.7;
8
9  void euler(int N,double T,double Y0[3],double Y[3]){
10     double h=T/N;
11     memcpy(Y,Y0,sizeof(double)*3);
```

```
12      for(int n=0;n<N;n++){
13          double F[3];
14          f(Y,F);
15          for(int i=0;i<3;i++)
16              Y[i]=Y[i]+h*F[i];
17      }
18  }
19
20  void taylor3(int N,double T,double Y0[3],double Y[3]){
21      double h=T/N;
22      memcpy(Y,Y0,sizeof(double)*3);
23      for(int n=0;n<N;n++){
24          double F[3],dF[3],ddF[3];
25          f(Y,F); df(Y,dF); ddf(Y,ddF);
26          for(int i=0;i<3;i++)
27              Y[i]=Y[i]+h*(F[i]+h/2*(dF[i]+h/3*ddF[i]));
28      }
29  }
30
31  double l2err(double Yold[3],double Y[3]){
32      double r=0.0;
33      for(int i=0;i<3;i++){
34          double t=Yold[i]-Y[i];
35          r+=t*t;
36      }
37      return sqrt(r);
38  }
39
40  int main(){
41      double Y0[3]={1,1,1};
42      double Yold1[3],Yold3[3];
43
44      printf("#%6s %21s %21s %21s\n","N","h","Euler-Error","Taylor3-Error");
45      for(int N=64;N<=2097152;N*=2){
46          double Y1[3],Y3[3];
47          euler(N,10.0,Y0,Y1);
48          taylor3(N,10.0,Y0,Y3);
49          if(N>64){
50              double e1=l2err(Yold1,Y1);
51              double e3=l2err(Yold3,Y3);
52              printf("%7d %21.14e %21.14e %21.14e\n",N/2,2.0/N,e1,e3);
53          }
54          memcpy(Yold1,Y1,sizeof(double)*3);
55          memcpy(Yold3,Y3,sizeof(double)*3);
```

```
56      }
57      printf("#\n# Euler Y(10)=(%.4f,%.4f,%.4f)\n",
58          Yold1[0],Yold1[1],Yold1[2]);
59      printf("# Taylor3 Y(10)=(%.4f,%.4f,%.4f)\n",
60          Yold3[0],Yold3[1],Yold3[2]);
61
62      return 0;
63  }
```

Produces the output

```
#     N                     h           Euler-Error         Taylor3-Error
     64  1.56250000000000e-02  2.33144219162279e+00  6.87363039241346e-03
    128  7.81250000000000e-03  9.09768917649233e-01  8.17946875729940e-04
    256  3.90625000000000e-03  3.99948374163062e-01  9.97005441356068e-05
    512  1.95312500000000e-03  1.87309428184730e-01  1.23086917858950e-05
   1024  9.76562500000000e-04  9.06177035928212e-02  1.52917985292896e-06
   2048  4.88281250000000e-04  4.45654545173638e-02  1.90566890906635e-07
   4096  2.44140625000000e-04  2.20988412770795e-02  2.37848247750358e-08
   8192  1.22070312500000e-04  1.10036933896191e-02  2.97086053326114e-09
  16384  6.10351562500000e-05  5.49044533986234e-03  3.71223986484652e-10
  32768  3.05175781250000e-05  2.74237613385240e-03  4.63404907483632e-11
  65536  1.52587890625000e-05  1.37047690794561e-03  5.76263589056002e-12
 131072  7.62939453125000e-06  6.85060723691014e-04  7.84212630971022e-13
 262144  3.81469726562500e-06  3.42485936314577e-04  1.94718163403492e-13
 524288  1.90734863281250e-06  1.71231862990783e-04  8.50993215964191e-14
1048576  9.53674316406250e-07  8.56131551710563e-05  2.24441841073325e-13
#
# Euler Y(10)=(-0.2950,-3.6966,0.0308)
# Taylor3 Y(10)=(-0.2950,-3.6966,0.0308)
```

The output indicates both the Euler and third order Taylor methods have converged and that

$$y(10) \approx (-0.2950, -3.6966, 0.0308)$$

to four decimal places. In particular, it is possible to achieve accuracy to four decimal places using Euler's method with a very small time step. We now modify of the program to approximate $y(100)$ as

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <string.h>
4
5  #include "force.h"
6
7  double a=0.2,b=0.2,c=5.7;
8
9  void taylor3(int N,double T,double Y0[3],double Y[3]){
10     double h=T/N;
11     memcpy(Y,Y0,sizeof(double)*3);
```

```
12      for(int n=0;n<N;n++){
13          double F[3],dF[3],ddF[3];
14          f(Y,F); df(Y,dF); ddf(Y,ddF);
15          for(int i=0;i<3;i++)
16              Y[i]=Y[i]+h*(F[i]+h/2*(dF[i]+h/3*ddF[i]));
17      }
18  }
19
20  void taylor5(int N,double T,double Y0[3],double Y[3]){
21      double h=T/N;
22      memcpy(Y,Y0,sizeof(double)*3);
23      for(int n=0;n<N;n++){
24          double F[3],dF[3],ddF[3],dddF[3],ddddF[3];
25          f(Y,F); df(Y,dF); ddf(Y,ddF); dddf(Y,dddF); ddddf(Y,ddddF);
26          for(int i=0;i<3;i++)
27              Y[i]=Y[i]+h*(F[i]+h/2*(dF[i]+h/3*(ddF[i]
28                  +h/4*(dddF[i]+h/5*ddddF[i]))));
29      }
30  }
31
32  double l2err(double Yold[3],double Y[3]){
33      double r=0.0;
34      for(int i=0;i<3;i++){
35          double t=Yold[i]-Y[i];
36          r+=t*t;
37      }
38      return sqrt(r);
39  }
40
41  int main(){
42      double Y0[3]={1,1,1};
43      double Yold3[3],Yold5[3];
44
45      printf("#%6s %21s %21s %21s\n","N","h","Taylor3-Error","Taylor5-Error");
46      for(int N=1024;N<=1048576;N*=2){
47          double Y3[3],Y5[3];
48          taylor3(N,100.0,Y0,Y3);
49          taylor5(N,100.0,Y0,Y5);
50          if(N>64){
51              double e3=l2err(Yold3,Y3);
52              double e5=l2err(Yold5,Y5);
53              printf("%7d %21.14e %21.14e %21.14e\n",N/2,2.0/N,e3,e5);
54          }
55          memcpy(Yold3,Y3,sizeof(double)*3);
```

```
56          memcpy(Yold5,Y5,sizeof(double)*3);
57      }
58      printf("#\n# Taylor3 Y(100)=(%.4f,%.4f,%.4f)\n",
59          Yold3[0],Yold3[1],Yold3[2]);
60      printf("# Taylor5 Y(100)=(%.4f,%.4f,%.4f)\n",
61          Yold5[0],Yold5[1],Yold5[2]);
62
63      return 0;
64  }
```

which produces the output

```
#     N                     h         Taylor3-Error         Taylor5-Error
    512  1.95312500000000e-03  9.18300070428134e+00  1.09223189817536e+01
   1024  9.76562500000000e-04  1.34790352426756e+00  7.09890686905542e-01
   2048  4.88281250000000e-04  3.13298051398290e+00  1.96914679957739e-02
   4096  2.44140625000000e-04  1.37643634106468e+00  5.12248706758858e-04
   8192  1.22070312500000e-04  2.22585397265740e-01  1.40326009777372e-05
  16384  6.10351562500000e-05  2.91356173509204e-02  4.04680089972077e-07
  32768  3.05175781250000e-05  3.70044935350887e-03  1.19579834787679e-08
  65536  1.52587890625000e-05  4.65847201769506e-04  3.33476115903369e-10
 131072  7.62939453125000e-06  5.84308312707253e-05  5.88494744531228e-10
 262144  3.81469726562500e-06  7.31733232632964e-06  4.61865108407059e-10
 524288  1.90734863281250e-06  9.14263474835942e-07  3.89429437064632e-10
#
# Taylor3 Y(100)=(9.6582,-3.5334,0.8133)
# Taylor5 Y(100)=(9.6582,-3.5334,0.8133)
```

The convergence of the third and fifth order methods and the fact that they agree implies

$$y(100) \approx (9.6582, -3.5334, 0.8133)$$

to four decimal places.