The Shu–Oscher Method

Your work should be presented in the form of a typed report using clear and properly punctuated English. Where appropriate include full program listing and output. If you choose to work in a group of two, please turn in independently prepared reports.

1. Consider the ordinary differential equation

$$y' = y^2 \cos(t), \qquad y(0) = 0.8$$

on the interval $[0, 8]$. Find the exact solution.

This ordinary differential equation is separable. Therefore we procede as

$$\int y^{-2} dy = \int \cos t \, dt, \qquad -y^{-1} = \sin t + C.$$

Solving for C using the initial condition implies

$$-1/0.8 = \sin 0 + C, \qquad C = -1.25.$$

Therefore, the exact solution is

$$y(t) = \frac{-1}{\sin t - 1.25}.$$

2. Consider the three-stage Runge–Kutta method with Butcher tableau given by

$$\begin{array}{c|ccc} 0 & & & \\ 1 & 1 & & \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & \\ \hline & \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \end{array}$$

This is called the Shu–Osher method. Use this method to approximate the solution of the differential equation given in part 1. Plot the exact solution and two different approximations, one for $h = 1/2$ and another for $h = 1/4$, on the same graph. Comment on the accuracy of the approximations.

The source code

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

double a=0,b=8,Y0=0.8;
double f(double t,double y){
    return y*y*cos(t);
}

/*  The Shu-Osher RK3 method is given by the tableau

        0  |
        1  |  1
       1/2 | 1/4  1/4
       --------------
           | 1/6  1/6  2/3
*/

double h;
double ShuOsher(double t,double y){
    double k1=f(t,y);
    double k2=f(t+h,y+h*k1);
    double k3=f(t+h/2,y+h*(k1+k2)/4);
    return y+h*(k1+k2+4*k3)/6;
}

double exact(double t){
    return -1/(sin(t)-1.25);
}

int main(){
```

```
32      printf("# Programing Project 2 Question 2.");
33      int N=16;
34      for(int k=4;k<=5;k++){
35          h=(b-a)/N;
36          double y=Y0;
37          printf("\n\n#N=%d h=%g\n",N,h);
38          printf("#%5s %21s %21s %21s\n","n","t","yn","error");
39          for(int n=0;;n++){
40              double t=a+n*h;
41              printf("%6d %21.14e %21.14e %21.14e\n",n,t,y,fabs(y-exact(t)));
42              if(n>=N) break;
43              y=ShuOsher(t,y);
44          }
45          N*=2;
46      }
47      return 0;
48  }
```

produces the output

```
# Programing Project 2 Question 2.

#N=16 h=0.5
#    n                     t                    yn                 error
     0  0.00000000000000e+00  8.00000000000000e-01  4.43980789632814e-17
     1  5.00000000000000e-01  1.27951263776160e+00  1.82204822650200e-02
     2  1.00000000000000e+00  2.30780682841958e+00  1.39999723410693e-01
     3  1.50000000000000e+00  3.44947907130311e+00  5.10838336066266e-01
     4  2.00000000000000e+00  2.54245464976471e+00  3.92656731059394e-01
     5  2.50000000000000e+00  1.33173265907593e+00  2.03121132593042e-01
     6  3.00000000000000e+00  8.04358896828691e-01  9.74519458851063e-02
     7  3.50000000000000e+00  5.71772217231438e-01  5.29219841460222e-02
     8  4.00000000000000e+00  4.62984036045151e-01  3.53211047637147e-02
     9  4.50000000000000e+00  4.19624730843091e-01  2.93029815454636e-02
    10  5.00000000000000e+00  4.22552172497710e-01  3.01568730183482e-02
    11  5.50000000000000e+00  4.72672436707051e-01  3.86951822185884e-02
    12  6.00000000000000e+00  5.90569273238894e-01  6.32752844876506e-02
    13  6.50000000000000e+00  8.29680845858125e-01  1.36614752171896e-01
    14  7.00000000000000e+00  1.29415717503060e+00  3.92145356816892e-01
    15  7.50000000000000e+00  1.99878884971222e+00  1.20633911682646e+00
    16  8.00000000000000e+00  2.17038433751895e+00  1.66629949016784e+00


#N=32 h=0.25
#    n                     t                    yn                 error
```

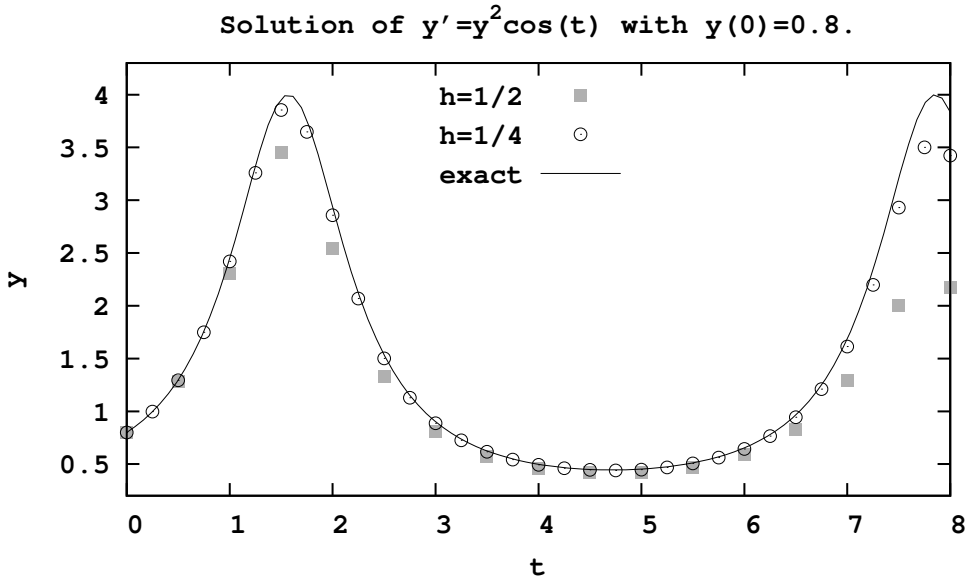
```
 0  0.00000000000000e+00  8.00000000000000e-01  4.43980789632814e-17
 1  2.50000000000000e-01  9.96575108504492e-01  8.35572727056929e-04
 2  5.00000000000000e-01  1.29441928905159e+00  3.31383097502840e-03
 3  7.50000000000000e-01  1.74915996724851e+00  1.02844312489250e-02
 4  1.00000000000000e+00  2.41980747062057e+00  2.79990812097009e-02
 5  1.25000000000000e+00  3.25883762782432e+00  6.32517546490945e-02
 6  1.50000000000000e+00  3.85467965661405e+00  1.05637750755321e-01
 7  1.75000000000000e+00  3.64683430255765e+00  1.12365590261935e-01
 8  2.00000000000000e+00  2.85704533384696e+00  7.80660469771442e-02
 9  2.25000000000000e+00  2.06771756926274e+00  5.12551049011183e-02
10  2.50000000000000e+00  1.50081338222946e+00  3.40404094395110e-02
11  2.75000000000000e+00  1.12900740817923e+00  2.26165438110361e-02
12  3.00000000000000e+00  8.86439391167150e-01  1.53714515466472e-02
13  3.25000000000000e+00  7.25378566212914e-01  1.08926621080204e-02
14  3.50000000000000e+00  6.16566359672663e-01  8.12784170479709e-03
15  3.75000000000000e+00  5.42567509282966e-01  6.41208844500231e-03
16  4.00000000000000e+00  4.92951298442536e-01  5.35384236632927e-03
17  4.25000000000000e+00  4.61472991931834e-01  4.72978719329816e-03
18  4.50000000000000e+00  4.44509790568562e-01  4.41792181999262e-03
19  4.75000000000000e+00  4.40223256773829e-01  4.36092759797569e-03
20  5.00000000000000e+00  4.48159076671912e-01  4.54996884414575e-03
21  5.25000000000000e+00  4.69148911893536e-01  5.02418505041009e-03
22  5.50000000000000e+00  5.05480059019902e-01  5.88755990573645e-03
23  5.75000000000000e+00  5.61383537384660e-01  7.35439096643344e-03
24  6.00000000000000e+00  6.43990390239332e-01  9.85416748721287e-03
25  6.25000000000000e+00  7.65040475515254e-01  1.42738923487596e-02
26  6.50000000000000e+00  9.43761135365142e-01  2.25344626648797e-02
27  6.75000000000000e+00  1.21109426109213e+00  3.89746079842333e-02
28  7.00000000000000e+00  1.61288150798586e+00  7.34210238616369e-02
29  7.25000000000000e+00  2.19687380821346e+00  1.45490238832707e-01
30  7.50000000000000e+00  2.92979918249501e+00  2.75328784043661e-01
31  7.75000000000000e+00  3.50033245406649e+00  4.15075610624395e-01
32  8.00000000000000e+00  3.42330953713037e+00  4.13374290556420e-01
```

which may be graphed with the `gnuplot` script

```
set terminal postscript enhanced eps font "Courier-Bold"
set output 'p2q2.eps'
set size 0.7,0.6
set xlabel "t"
set ylabel "y"
set key spacing 1.4
set key top center
set title "Solution of y'=y^2cos(t) with y(0)=0.8."
plot [] [0.2:4.3] \
```

```
10  "p2q2.dat" using 2:3 index 0 with points pt 5 lc rgb "#AFAFAF" ti "h=1/2",\
11  "p2q2.dat" using 2:3 index 1 with points pt 6 lc rgb "#000000" ti "h=1/4",\
12  -1/(sin(x)-1.25) lt 1 ti "exact"
```

to obtain the plot



The accuracy of the approximations are the worst around $t \approx 2$ and $t \approx 8$ between these values the error miraculously decreases again. This fortuitous. We note that the maximum error when $h = 1/2$ is about 1.666 while the maximum error when $h = 1/4$ is 0.4151. Thus, decreasing the step size results in a more accurate solutions, just as expected.

3. Given $n \in \mathbf{N}$ define $h = 8/n$ and $t_j = hj$. Let y be the exact solution to the differential equation and y_j be the approximation of $y(t_j)$ obtained using the Shu–Osher method. Define the error

$$E(h) = \max\big\{\, |y_j - y(t_j)| : j = 0, 1, \ldots, n \,\big\}.$$

If $E(h) \approx Kh^p$ for some K and some p we say the method is of order p. Numerically determine the order of the Shu–Osher method by taking $n = 2^k$ for $k = 4, 5, \ldots, 16$ for the differential equation in part 1. Plot the corresponding values of $E(h)$ versus h using log-log coordinates, note that $\log E(h) \approx p \log h + \log K$ and solve for p and K.

Modifying the program in the previous question to compute $E(h)$ yields

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

double a=0,b=8,Y0=0.8;
double f(double t,double y){
    return y*y*cos(t);
}

/*  The Shu-Osher RK3 method is given by the tableau

        0  |
        1  |  1
       1/2 | 1/4  1/4
       --------------
           | 1/6  1/6  2/3
*/

double h;
double ShuOsher(double t,double y){
    double k1=f(t,y);
    double k2=f(t+h,y+h*k1);
    double k3=f(t+h/2,y+h*(k1+k2)/4);
    return y+h*(k1+k2+4*k3)/6;
}

double exact(double t){
    return -1/(sin(t)-1.25);
}

double E(int N){
    double y=Y0;
    double r=0;
    for(int n=0;;n++){
```

```
        double t=a+n*h;
        double e=fabs(y-exact(t));
        if(r<e) r=e;
        if(n>=N) break;
        y=ShuOsher(t,y);
    }
    return r;
}

int main(){
    printf("# Programing Project 2 Question 3.\n\n");
    int N=16;
    printf("#%7s %21s %21s\n","N","h","E(h)");
    for(int k=4;k<=16;k++){
        h=(b-a)/N;
        printf("%8d %21.14e %21.14e\n",N,h,E(N));
        N*=2;
    }
    return 0;
}
```

with output

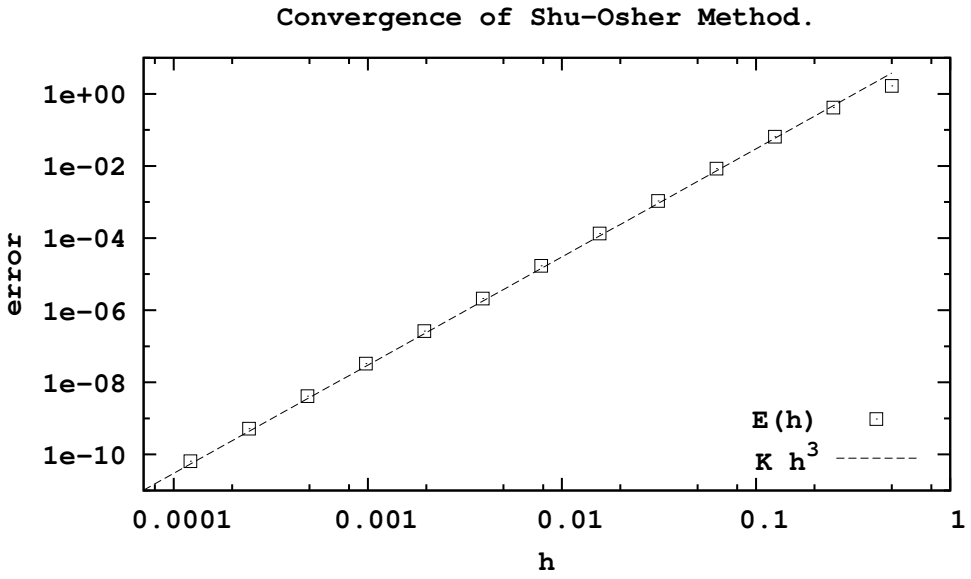
```
# Programing Project 2 Question 3.

#      N                     h                  E(h)
      16  5.00000000000000e-01  1.66629949016784e+00
      32  2.50000000000000e-01  4.15075610624395e-01
      64  1.25000000000000e-01  6.52150342066623e-02
     128  6.25000000000000e-02  8.48086610634722e-03
     256  3.12500000000000e-02  1.07399257382710e-03
     512  1.56250000000000e-02  1.35015118946845e-04
    1024  7.81250000000000e-03  1.69235555161672e-05
    2048  3.90625000000000e-03  2.11837955901243e-06
    4096  1.95312500000000e-03  2.64982487493525e-07
    8192  9.76562500000000e-04  3.31342513283477e-08
   16384  4.88281250000000e-04  4.14253449056677e-09
   32768  2.44140625000000e-04  5.17877228080332e-10
   65536  1.22070312500000e-04  6.46317869015811e-11
```

which may be plotted with the `gnuplot` script

```
set terminal postscript enhanced eps font "Courier-Bold"
set output 'p2q3.eps'
set size 0.7,0.6
set logscale x
```

```
set logscale y
set xlabel "h"
set ylabel "error"
set key spacing 1.4
set key bottom right
set format y "%.0e"
set title "Convergence of Shu-Osher Method."
plot [0.00007:] [1e-11:10] "p2q3.dat" using 2:3 \
    with points pt 4 ti "E(h)", \
    30*x**3 ti "K h^3"
```

to obtain the plot



To find p and K I tried a few choices. Since p should be an integer, it was easy to see that $p = 3$ was fine. After that, tuning K so the line given by Kh^3 overlaid the data points gave a value $K = 30$. While a better fit could be obtained using a least squares approximation, these values are good enough to numerically confirm the Shu-Osher method is third order.

4. To obtain the most accurate approximate what size should h be taken? What happens if you take h even smaller? Can you find an h so small that $E(h)$ is 10 times larger than the minimal value? Why or why not?

To answer this question, line 48 in previous program was modified to include larger values of k and smaller values of $h = (b-a)/2^k$ for $k = 16$ through 30. Values with $k > 30$ were not computed due to time contraints and the overflow of the integer variables used to count the time steps. The output obtained was

```
# Programing Project 2 Question 4.

#        N                     h                  E(h)
     65536  1.22070312500000e-04  6.46317869015811e-11
    131072  6.10351562500000e-05  8.48965667731016e-12
    262144  3.05175781250000e-05  1.92784828899339e-12
    524288  1.52587890625000e-05  5.84372177384052e-13
   1048576  7.62939453125000e-06  1.24362109152343e-13
   2097152  3.81469726562500e-06  1.62474786229161e-12
   4194304  1.90734863281250e-06  4.06653877238483e-13
   8388608  9.53674316406250e-07  5.43940456912417e-12
  16777216  4.76837158203125e-07  1.72264849954140e-12
  33554432  2.38418579101562e-07  7.59448818936359e-12
  67108864  1.19209289550781e-07  8.98830467520317e-12
 134217728  5.96046447753906e-08  3.55158047068982e-12
 268435456  2.98023223876953e-08  1.82164144879593e-11
 536870912  1.49011611938477e-08  7.92447682994302e-12
1073741824  7.45058059692383e-09  2.96292892564692e-11
```

The smallest value of $E(h)$ occured when $k = 20$ with $E(8/2^{20}) \approx 1.24 \times 10^{-13}$. After $k = 20$ there was no consistent decrease in the size of the error. For values $k \geq 23$ the error satisfied $E(h) \geq 10^{-12}$ and, in fact, also greater than 10 times the minimal value. For example, $E(8/2^{28}) \approx 1.82 \times 10^{-11}$ is 147 times greater.

5. [Extra Credit] Let `su3` represent one step of the Shu–Osher method such that

$$y_{j+1} = \texttt{su3}(y_j, t_j, t_{j+1}).$$

Use Taylor's theorem to show that the truncation error

$$\tau_h = y(t+h) - \texttt{su3}(y(t), t, t+h) = \mathcal{O}(h^4).$$

It is fine and recommended to use a computer algebra system such as Maple to assist in your calculations.

The Maple script

```
1  restart;
2  kernelopts(printbytes=false);
3  #compute the truncation error
4  ynp1:=y+h*(k1+k2+4*k3)/6;
5  k3:=f(t+h/2,y+h*(k1+k2)/4);
6  k2:=f(t+h,y+h*k1);
7  k1:=f(t,y);
8  tau:=subs(y=y(t),ynp1)-y(t+h);
9  `diff/y`:=proc(g,x) f(g,y(g))*diff(g,x) end proc;
10 series(tau,h=0,5);
```

produces the output

```
    |\^/|     Maple 9.5 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2004
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> restart;
> kernelopts(printbytes=false);
                                    true

#compute the truncation error
> ynp1:=y+h*(k1+k2+4*k3)/6;
                                  h (k1 + k2 + 4 k3)
                        ynp1 := y + ------------------
                                            6

> k3:=f(t+h/2,y+h*(k1+k2)/4);
                                         h (k1 + k2)
                      k3 := f(t + h/2, y + -----------)
                                              4

> k2:=f(t+h,y+h*k1);
```

```
                          k2 := f(t + h, y + h k1)

> k1:=f(t,y);
                               k1 := f(t, y)

> tau:=subs(y=y(t),ynp1)-y(t+h);
tau := y(t) + 1/6 h (f(t, y(t)) + f(t + h, y(t) + h f(t, y(t)))

     + 4 f(t + h/2, y(t) + 1/4 h (f(t, y(t)) + f(t + h, y(t) + h f(t, y(t))))))

     - y(t + h)

> `diff/y`:=proc(g,x) f(g,y(g))*diff(g,x) end proc;
              diff/y := proc(g, x) f(g, y(g))*diff(g, x) end proc

> series(tau,h=0,5);
(1/24 D[1, 2](f)(t, y(t)) D[2](f)(t, y(t)) f(t, y(t))

     - 1/24 f(t, y(t)) D[2, 2](f)(t, y(t)) D[1](f)(t, y(t))

     - 1/24 D[1, 2](f)(t, y(t)) D[1](f)(t, y(t))

     + 1/24 D[2](f)(t, y(t)) D[1, 1](f)(t, y(t))

                           2
     - 1/24 D[2](f)(t, y(t))  D[1](f)(t, y(t))

                           3              4      5
     - 1/24 D[2](f)(t, y(t))  f(t, y(t))) h  + O(h )

> quit
bytes used=1732168, alloc=1441528, time=0.06
```

which shows that the local truncation error τ of the Shu–Osher method is $\mathcal{O}(h^4)$. It follows that Shu–Osher is a 3rd order method.

6. The Rössler System is a three dimensional ordinary differential equation of the form $du/dt = f(u)$ with a given initial condition $u(t_0) = u_0$ where $u(t) \in \mathbf{R}^3$ and

$$f(u) = \begin{bmatrix} -u_2 - u_3 \\ u_1 + au_2 \\ b + u_3(u_1 - c) \end{bmatrix} \qquad \text{where} \qquad u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

with $a = b = 0.2$ and $c = 5.7$. Write a program to approximate $u(1)$ using the Shu–Osher method and the initial condition

$$u(0) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Approximate $u(1)$ by performing a convergence study taking h smaller and smaller to obtain an approximation good to at least 4 significant digits.

Modifications to the program from question 2 so that it handles vector-valued ordinary differential equations yields

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

double A=0,B=1,U0[3]={1,1,1};
double a=0.2,b=0.2,c=5.7;

void f(double z[3],double t,double u[3]){
    z[0]=-u[1]-u[2];
    z[1]=u[0]+a*u[1];
    z[2]=b+u[2]*(u[0]-c);
}

/*  The Shu-Osher RK3 method is given by the tableau

        0  |
        1  |  1
       1/2 | 1/4  1/4
       --------------
           | 1/6  1/6  2/3
*/

double h;
#define LOOPi for(int i=0;i<3;i++)
void ShuOsher(double z[3],double t,double y[3]){
    double tmp[3],k1[3],k2[3],k3[3];
```

```
27      // k1=f(t,y)
28      f(k1,t,y);
29      // k2=f(t+h,y+h*k1);
30      LOOPi tmp[i]=y[i]+h*k1[i]; f(k2,t+h,tmp);
31      // k3=f(t+h/2,y+h*(k1+k2)/4);
32      LOOPi tmp[i]=y[i]+h*(k1[i]+k2[i])/4; f(k3,t+h/2,tmp);
33      // z=y+h*(k1+k2+4*k3)/6;
34      LOOPi z[i]=y[i]+h*(k1[i]+k2[i]+4*k3[i])/6;
35  }
36
37  int main(){
38      printf("# Programing Project 2 Question 6.\n\n");
39      int N=16;
40      printf("#%5s %21s %21s %21s\n","n","u1","u2","u3");
41      for(int k=4;k<=16;k++){
42          h=(B-A)/N;
43          double u[3]; LOOPi u[i]=U0[i];
44          for(int n=0;;n++){
45              double t=A+n*h;
46              if(n>=N) break;
47              ShuOsher(u,t,u);
48          }
49          printf("%6d %21.14e %21.14e %21.14e\n",N,u[0],u[1],u[2]);
50          N*=2;
51      }
52      return 0;
53  }
```

with output

```
# Programing Project 2 Question 6.

#    n                    u1                    u2                    u3
    16 -5.79121501107407e-01  1.45842033698500e+00  3.70667611849356e-02
    32 -5.79090520966628e-01  1.45845379309230e+00  3.71119704616039e-02
    64 -5.79087078592271e-01  1.45845784165318e+00  3.71168637586746e-02
   128 -5.79086673941648e-01  1.45845833915507e+00  3.71174316971960e-02
   256 -5.79086624919109e-01  1.45845840080233e+00  3.71175000892390e-02
   512 -5.79086618887286e-01  1.45845840847433e+00  3.71175084800086e-02
  1024 -5.79086618139263e-01  1.45845840943121e+00  3.71175095191002e-02
  2048 -5.79086618046128e-01  1.45845840955069e+00  3.71175096483811e-02
  4096 -5.79086618034510e-01  1.45845840956562e+00  3.71175096645035e-02
  8192 -5.79086618033049e-01  1.45845840956748e+00  3.71175096665166e-02
 16384 -5.79086618032881e-01  1.45845840956773e+00  3.71175096667679e-02
 32768 -5.79086618032863e-01  1.45845840956775e+00  3.71175096667995e-02
```

```
65536 -5.79086618032854e-01  1.45845840956777e+00  3.71175096668036e-02
```

This shows to four significant digits that

$$u(1) \approx \begin{bmatrix} -0.5791 \\ 1.458 \\ 0.03712 \end{bmatrix}.$$

7. [Extra Credit] Is it possible to compute $u(10)$, $u(100)$ and $u(1000)$ to 4 significant digits. Explain, why or why not. Does using a different numerical scheme help?

The main routine starting at line 37 from the code in question 6 was modified to compute convergence results for $u(10)$, $u(100)$ and $u(1000)$ as

```
37 int main(){
38     printf("# Programing Project 2 Question 6.");
39     double B0[]={10,100,1000};
40     for(int l=0;l<3;l++){
41         B=B0[l];
42         printf("\n\n# Computing u(%g)...\n",B);
43         printf("#%9s %21s %21s %21s\n","n","u1","u2","u3");
44         int N=1024;
45         for(int k=10;k<=28;k++){
46             h=(B-A)/N;
47             double u[3]; LOOPi u[i]=U0[i];
48             for(int n=0;;n++){
49                 double t=A+n*h;
50                 if(n>=N) break;
51                 ShuOsher(u,t,u);
52             }
53             printf("%10d %21.14e %21.14e %21.14e\n",N,u[0],u[1],u[2]);
54             N*=2;
55         }
56     }
57     return 0;
58 }
```

The resulting output is

```
# Programing Project 2 Question 6.

# Computing u(10)...
#        n                    u1                    u2                    u3
      1024 -2.95003849808899e-01 -3.69655184356218e+00  3.07870232937301e-02
      2048 -2.95004677367375e-01 -3.69655295858790e+00  3.07870245181947e-02
      4096 -2.95004779813878e-01 -3.69655309834745e+00  3.07870246674848e-02
      8192 -2.95004792557400e-01 -3.69655311584106e+00  3.07870246859092e-02
     16384 -2.95004794146460e-01 -3.69655311802922e+00  3.07870246881973e-02
     32768 -2.95004794344827e-01 -3.69655311830283e+00  3.07870246884825e-02
     65536 -2.95004794369644e-01 -3.69655311833701e+00  3.07870246885180e-02
    131072 -2.95004794372666e-01 -3.69655311834125e+00  3.07870246885227e-02
    262144 -2.95004794373066e-01 -3.69655311834186e+00  3.07870246885232e-02
    524288 -2.95004794373240e-01 -3.69655311834196e+00  3.07870246885228e-02
   1048576 -2.95004794373173e-01 -3.69655311834188e+00  3.07870246885231e-02
```

```
   2097152 -2.95004794373411e-01 -3.69655311834186e+00  3.07870246885223e-02
   4194304 -2.95004794373554e-01 -3.69655311834149e+00  3.07870246885219e-02
   8388608 -2.95004794372813e-01 -3.69655311834135e+00  3.07870246885245e-02
  16777216 -2.95004794373108e-01 -3.69655311834207e+00  3.07870246885227e-02
  33554432 -2.95004794373263e-01 -3.69655311834196e+00  3.07870246885230e-02
  67108864 -2.95004794372150e-01 -3.69655311833999e+00  3.07870246885290e-02
 134217728 -2.95004794372105e-01 -3.69655311834390e+00  3.07870246885264e-02
 268435456 -2.95004794370686e-01 -3.69655311834732e+00  3.07870246885289e-02


# Computing u(100)...
#        n                    u1                    u2                    u3
      1024  8.85580462644710e+00 -7.33065475049312e-01  1.17026856648204e+00
      2048  9.14712461061900e+00 -2.33584223306671e+00  8.18564355661438e-01
      4096  1.05355080645244e+01 -4.87147559957140e+00  1.00156984195977e+00
      8192  9.80443722026393e+00 -3.73843796697093e+00  8.45458946062112e-01
     16384  9.67755409618304e+00 -3.56014501158554e+00  8.17537866473780e-01
     32768  9.66069350959131e+00 -3.53681370566271e+00  8.13831221375155e-01
     65536  9.65854342755116e+00 -3.53384447034662e+00  8.13358650865590e-01
    131072  9.65827233081817e+00 -3.53347018774449e+00  8.13299067269881e-01
    262144  9.65823830265327e+00 -3.53342320935407e+00  8.13291588300361e-01
    524288  9.65823404009403e+00 -3.53341732461219e+00  8.13290651440364e-01
   1048576  9.65823350735858e+00 -3.53341658913761e+00  8.13290534351158e-01
   2097152  9.65823343961626e+00 -3.53341649561794e+00  8.13290519461545e-01
   4194304  9.65823343263439e+00 -3.53341648597713e+00  8.13290517927430e-01
   8388608  9.65823343368691e+00 -3.53341648743107e+00  8.13290518158685e-01
  16777216  9.65823343404115e+00 -3.53341648791852e+00  8.13290518236846e-01
  33554432  9.65823343008155e+00 -3.53341648245276e+00  8.13290517366204e-01
  67108864  9.65823343328949e+00 -3.53341648687624e+00  8.13290518072415e-01
 134217728  9.65823343044070e+00 -3.53341648295157e+00  8.13290517445012e-01
 268435456  9.65823342947192e+00 -3.53341648158942e+00  8.13290517237369e-01


# Computing u(1000)...
#        n                    u1                    u2                    u3
      1024                   nan                   nan                   nan
      2048                   nan                   nan                   nan
      4096  4.57792726316119e+00  1.47502462514753e-01  1.19845964041384e-01
      8192  9.37960941527698e-01 -4.54417682304627e+00  3.64071223399243e-02
     16384 -2.25351106542763e+00 -9.34150175124568e+00  2.25363832334806e-02
     32768 -2.70331457112544e+00 -2.16531359128769e+00  2.32402369684749e-02
     65536  3.88874636058173e+00 -6.01321106120780e+00  6.11039773885257e-02
    131072  2.50465901808905e+00 -2.52245299149159e+00  5.22289663030917e-02
    262144 -2.21241110584907e+00 -5.61516679080841e+00  2.35557087189117e-02
```

```
   524288  3.77218910074585e+00 -3.28693254891398e-01  8.46282043036219e-02
  1048576 -1.80907516174246e+00 -8.87009828606047e+00  2.37383879847695e-02
  2097152 -3.91667540441127e+00 -8.69095478726910e+00  1.93105216416161e-02
  4194304 -6.31900834603889e+00 -6.31270507462329e+00  1.60478467571861e-02
  8388608 -7.12734137326140e-01 -9.73603597768513e+00  2.65900167487680e-02
 16777216 -2.09222347322555e+00 -2.55211804559404e+00  2.48209287028857e-02
 33554432 -8.60976314507966e-01 -4.43890521348820e+00  2.81385024367594e-02
 67108864 -3.15107023171801e+00 -8.87257648155197e+00  2.07074577141197e-02
134217728 -7.09681411405404e+00 -5.94979304825445e+00  1.51631044732936e-02
268435456 -1.59935674326445e+00 -6.32487931843131e+00  2.50286140584541e-02
```

Which seems to converge for $u(10)$ and $u(100)$ to at least four significant digits. While there was no sign of convergence for $u(1000)$. In summary

$$u(10) \approx \begin{bmatrix} -0.2950 \\ -3.697 \\ 0.03079 \end{bmatrix} \qquad \text{and} \qquad u(100) \approx \begin{bmatrix} 9.658 \\ -3.533 \\ 0.8133 \end{bmatrix}.$$

It is possible that a higher-order method could help approximate $u(1000)$. It could also happen happen that we need to increase the precision of the floating point arithemetic as well as the order of the method. Further work using extended precision arithmetic and a higher order method could lead to additional extra credit.

8. Find the linear stability domain for the Shu–Osher method by considering the differential equation

$$y' = (a + ib)y$$

and determining what values of $z = h(a + ib)$ guarantee that the approximations $y_j \to 0$ as $j \to \infty$. Plot the domain and include the graph in your report.

The Maple script

```
restart;
kernelopts(printbytes=false);
ynp1:=y+h*(k1+k2+4*k3)/6;
k3:=f(t+h/2,y+h*(k1+k2)/4);
k2:=f(t+h,y+h*k1);
k1:=f(t,y);
f:=(t,y)->zeta*y;
p:=simplify(subs(zeta=z/h,ynp1)/y);
with(plots):
pabs:=abs(subs(z=x+I*y,p));
plotsetup(ps,plotoutput="stability.ps",
    plotoptions=`portrait,width=4in,height=5in,noborder`);
contourplot(pabs,x=-3..3,y=-3..3,contours=[1],numpoints=10000,
    scaling=constrained);
```

produces the output

```
    |\^/|     Maple 9.5 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2004
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> restart;
> kernelopts(printbytes=false);
                                    true

> ynp1:=y+h*(k1+k2+4*k3)/6;
                                  h (k1 + k2 + 4 k3)
                      ynp1 := y + ------------------
                                          6

> k3:=f(t+h/2,y+h*(k1+k2)/4);
                                        h (k1 + k2)
                    k3 := f(t + h/2, y + -----------)
                                            4

> k2:=f(t+h,y+h*k1);
                           k2 := f(t + h, y + h k1)
```

```
> k1:=f(t,y);
                                  k1 := f(t, y)

> f:=(t,y)->zeta*y;
                              f := (t, y) -> zeta y

> p:=simplify(subs(zeta=z/h,ynp1)/y);
                                          2        3
                         p := 1 + z + 1/2 z  + 1/6 z

> with(plots):
Warning, the name changecoords has been redefined
> pabs:=abs(subs(z=x+I*y,p));
                        |                         2            3 |
                        |                (x + y I)    (x + y I)  |
                pabs := | 1 + x + y I + ---------- + ---------- |
                        |                    2            6     |
```

```
> plotsetup(ps,plotoutput="stability.ps",
>   plotoptions=`portrait,width=4in,height=5in,noborder`);
> contourplot(pabs,x=-3..3,y=-3..3,contours=[1],numpoints=10000,
>   scaling=constrained);
> quit
bytes used=55714816, alloc=7600784, time=1.30
```

and the graph `stability.ps` which depicts the linear stability domain as