Math 467/667: Programming Project 2 Answer Key

1. Let $y(x)$ be the exact solution to the differential equation

$$y' = f(x, y), \qquad y(x_0) = y_0.$$

Let $x_n = x_0 + nh$ and define the approximation $y_n \approx y(x_n)$ using the Shu–Osher three-stage Runge–Kutta numerical method

$$\begin{aligned} k_1 &= f(x_n, y_n) \\ k_2 &= f(x_n + h, y_n + hk_1) \\ k_3 &= f(x_n + h/2, y_n + h(k_1 + k_2)/4) \\ y_{n+1} &= y_n + h(k_1 + k_2 + 4k_3)/6. \end{aligned}$$

(i) Determine the truncation error by substituting the exact solution into the numerical method and making a series expansion around $h = 0$. Namely, define

$$\begin{aligned} \kappa_1 &= f(x_n, y(x_n)) \\ \kappa_2 &= f(x_n + h, y(x_n) + h\kappa_1) \\ \kappa_3 &= f(x_n + h/2, y(x_n) + h(\kappa_1 + \kappa_2)/4) \end{aligned}$$

and then find m such that

$$\tau_n = y(x_{n+1}) - y(x_n) - h(\kappa_1 + \kappa_2 + 4\kappa_3)/6 = \mathcal{O}(h^m) \qquad \text{as} \qquad h \to 0.$$

If you use a computer algebra system to do this (recommended), please include both the input and output for the calculation.

I wrote a script in Maple

```
restart;
kernelopts(printbytes=false);
eq:=D(y)=(s->f(s,y(s)));
k1:=f(xn,y(xn));
k2:=f(xn+h,y(xn)+h*k1);
k3:=f(xn+h/2,y(xn)+h*(k1+k2)/4);
ynp1:=y(xn)+h*(k1+k2+4*k3)/6;
r:=y(xn+h)-ynp1;
simplify(subs(h=0,r));
dr[0]:=r;
for j from 1 to 5
do
    dr[j]:=eval(subs(eq,diff(dr[j-1],h)));
    z:=simplify(subs(h=0,dr[j]));
```

```
    print(d^j=z);
    if z<>0 then
        print("Truncation error is ",h^j);
        break;
    end
end:
```

which ran and produced the output

```
License expires in 16 days
    |\^/|     Maple 2020 (X86 64 LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2020
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> restart;
> kernelopts(printbytes=false);
                                      true

> eq:=D(y)=(s->f(s,y(s)));
                          eq := D(y) = (s -> f(s, y(s)))

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

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

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

    y(xn) + 1/4 h (f(xn, y(xn)) + f(xn + h, y(xn) + h f(xn, y(xn)))))

> ynp1:=y(xn)+h*(k1+k2+4*k3)/6;
ynp1 := y(xn) + 1/6 h (f(xn, y(xn)) + f(xn + h, y(xn) + h f(xn, y(xn))) + 4 f(

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

    )

> r:=y(xn+h)-ynp1;
r := y(xn + h) - y(xn) - 1/6 h (f(xn, y(xn))

     + f(xn + h, y(xn) + h f(xn, y(xn))) + 4 f(xn + h/2,
```

```
    y(xn) + 1/4 h (f(xn, y(xn)) + f(xn + h, y(xn) + h f(xn, y(xn))))))

> simplify(subs(h=0,r));
                                        0

> dr[0]:=r;
dr[0] := y(xn + h) - y(xn) - 1/6 h (f(xn, y(xn))

     + f(xn + h, y(xn) + h f(xn, y(xn))) + 4 f(xn + h/2,

    y(xn) + 1/4 h (f(xn, y(xn)) + f(xn + h, y(xn) + h f(xn, y(xn))))))

> for j from 1 to 5
> do
>     dr[j]:=eval(subs(eq,diff(dr[j-1],h)));
>     z:=simplify(subs(h=0,dr[j]));
>     print(d^j=z);
>     if z<>0 then
>         print("Truncation error is ",h^j);
>         break;
>     end
> end:
                                     d = 0

                                      2
                                     d  = 0

                                      3
                                     d  = 0

 4                                  3                                          2
d  = f(xn, y(xn)) D[2](f)(xn, y(xn))  + D[1](f)(xn, y(xn)) D[2](f)(xn, y(xn))

     + (-D[1, 2](f)(xn, y(xn)) f(xn, y(xn)) - D[1, 1](f)(xn, y(xn)))

    D[2](f)(xn, y(xn)) + D[1](f)(xn, y(xn))

    (D[1, 2](f)(xn, y(xn)) + D[2, 2](f)(xn, y(xn)) f(xn, y(xn)))

                                                      4
                          "Truncation error is ", h

> quit
memory used=7.3MB, alloc=40.3MB, time=0.15
```

From this one can can see that the first term that doesn't vanish in the Taylor series is

$$\frac{d^4 r}{dh^4}\Big|_{h=0} = f f_y^3 + f_x f_y^2 - (f_{xy} f + f_{xx}) f_y + f_x (f_{xy} + f_{yy} f).$$

Here for notational convenience the arguments (x_n, y_n) have been dropped from f and all it derivatives. Consequently, the truncation error of the Shu–Osher method is $\mathcal{O}(h^4)$.

(ii) Determine the linear stability domain of the Shu–Osher method. Namely, substitute $f(x, y) = Ay$ into the method, set $h = 1$ and exactly solve the resulting difference equation for a solution of the form $y_n = \rho^n$. Write $A = a + ib$ and plot what values of $a + ib$ in the complex plane lead to values of ρ such that $|\rho| < 1$.

I wrote a script in Maple

```
restart;
kernelopts(printbytes=false);
eq:=D(y)=(s->f(s,y(s)));
k1:=f(xn,y(xn));
k2:=f(xn+h,y(xn)+h*k1);
k3:=f(xn+h/2,y(xn)+h*(k1+k2)/4);
ynp1:=y(xn)+h*(k1+k2+4*k3)/6;
f:=(xi,eta)->A*eta;
method:=y(xn+h)=ynp1;
ceq:=eval(subs(y=(s->rho^s),method));
ceq2:=subs({xn=0,h=1},ceq);
S:=solve(ceq2,rho);
Z1:=subs(A=a+I*b,abs(S));
with(plots):
plotsetup(ps,plotoutput=`plotii.ps`,
    plotoptions=`portrait,color,noborder,width=500,height=500`);
contourplot(Z1,a=-4..2,b=-3..3,contours=[1],grid=[100,100],
    filled=true);
plotsetup(default);
```

which ran and produced the output

```
License expires in 16 days
    |\^/|     Maple 2020 (X86 64 LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2020
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> restart;
> kernelopts(printbytes=false);
                                    true
```

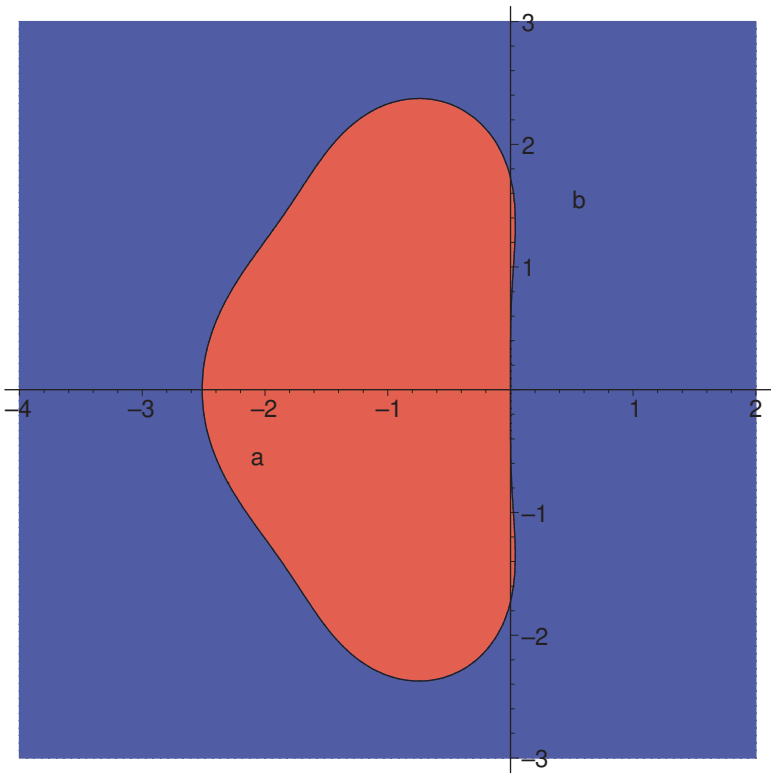
```
> eq:=D(y)=(s->f(s,y(s)));
                        eq := D(y) = (s -> f(s, y(s)))

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

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

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

    y(xn) + 1/4 h (f(xn, y(xn)) + f(xn + h, y(xn) + h f(xn, y(xn)))))

> ynp1:=y(xn)+h*(k1+k2+4*k3)/6;
ynp1 := y(xn) + 1/6 h (f(xn, y(xn)) + f(xn + h, y(xn) + h f(xn, y(xn))) + 4 f(

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

    )

> f:=(xi,eta)->A*eta;
                             f := (xi, eta) -> A eta

> method:=y(xn+h)=ynp1;
method := y(xn + h) = y(xn) + 1/6 h (A y(xn) + A (y(xn) + h A y(xn))

     + 4 A (y(xn) + 1/4 h (A y(xn) + A (y(xn) + h A y(xn)))))

> ceq:=eval(subs(y=(s->rho^s),method));
                                /
          (xn + h)      xn      |     xn          xn           xn
ceq := rho         = rho   + h |A rho   + A (rho   + h A rho  )
                                \

           /                xn          xn           xn  \\
           |   xn   h (A rho   + A (rho   + h A rho  ))||
     + 4 A |rho   + -----------------------------------||/6
           \                          4                 //

> ceq2:=subs({xn=0,h=1},ceq);
                                                  /          A (1 + A)\
                                              2 A |1 + A/4 + ---------|
                                A (1 + A)       \              4    /
```

```
         ceq2 := rho = 1 + A/6 + --------- + -------------------------
                                     6                   3

> S:=solve(ceq2,rho);
                                           2        3
                         S := 1 + A + 1/2 A  + 1/6 A

> Z1:=subs(A=a+I*b,abs(S));
                       |                          2            3 |
                       |                 (a + b I)    (a + b I)  |
                 Z1 := | 1 + a + b I + ---------- + ---------- |
                       |                     2            6      |

> with(plots):
> plotsetup(ps,plotoutput=`plotii.ps`,
>     plotoptions=`portrait,color,noborder,width=500,height=500`);
> contourplot(Z1,a=-4..2,b=-3..3,contours=[1],grid=[100,100],
>     filled=true);
> plotsetup(default);
> quit
memory used=45.0MB, alloc=118.4MB, time=0.89
```

Note from the Maple output that

$$\rho = 1 + A + \frac{1}{2}A^2 + \frac{1}{6}A^3.$$

The script finishes by setting $A = a + ib$ and plotting $|\rho|$ as



where the red region indicates the stable region for which $|\rho| < 1$ and the blue region indicates the unstable region when $|\rho| > 1$.

(iii) Consider the ordinary differential equation

$$y' = y^2 \sin 2x, \qquad y(0) = -1$$

on the interval $[0, 4]$. Find the exact solution.

This is a separable equation, so integrate as

$$\int y^{-2} dy = \int \sin 2x \, dx \qquad \text{to obtain} \qquad -\frac{1}{y} = -\frac{1}{2}\cos 2x + C.$$

Solving for C using the initial condition yields

$$-\frac{1}{-1} = -\frac{1}{2}\cos(2 \cdot 0) + C \qquad \text{or equivalently} \qquad C = 1 + \frac{1}{2} = \frac{3}{2}.$$

It follows that the exact solution is

$$y(x) = \frac{1}{\frac{1}{2}\cos 2x - \frac{3}{2}} = \frac{2}{\cos 2x - 3}.$$

(iv) Use the Shu–Osher method to approximate the solution to the differential equation solved in part (iii) by setting $h = 4/N$ with $N = 32$. Plot both the exact and the approximate solutions on the same graph. Include program source code, numerical output to be plotted as well as the actual plot. Comment on the accuracy of the approximation.

I wrote a program in Julia

```
# Question iv

using Printf

function shu_osher(xn,yn)
    k1 = f(xn,yn)
    k2 = f(xn+h,yn+h*k1)
    k3 = f(xn+h/2,yn+h*(k1+k2)/4)
    return yn + h*(k1+k2+4*k3)/6
end
function f(x,y)
    return y^2*sin(2*x)
end
function yexact(x)
    return 2/(cos(2*x)-3)
end

N = 32
h = 4/N
x0 = 0
y0 = -1

yn=y0
xn=x0
@printf("#%21s %22s %22s\n","xn","yn","yexact(xn)")
for n=1:N
    global xn,yn
    @printf("%22.14e %22.14e %22.14e\n",xn,yn,yexact(xn))
    yn = shu_osher(xn,yn)
    xn = x0 + n*h
end
@printf("%22.14e %22.14e %22.14e\n",xn,yn,yexact(xn))
```

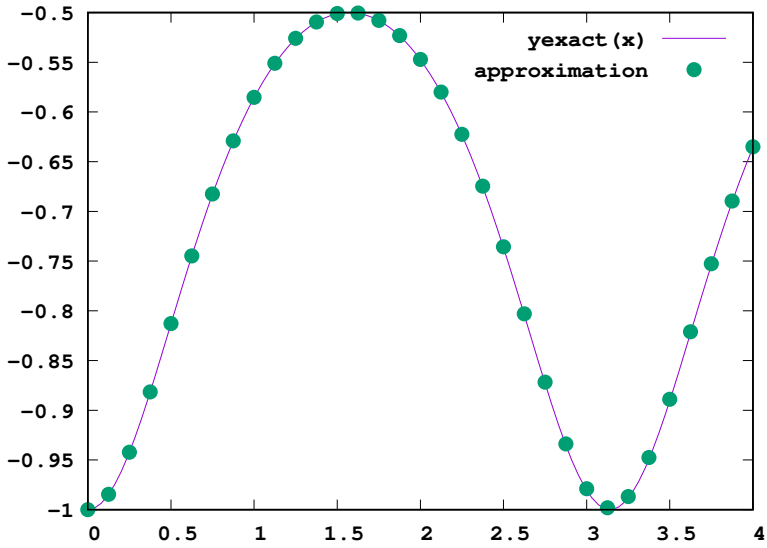
to approximate the solution. The output was

```
#                   xn                     yn              yexact(xn)
  0.00000000000000e+00  -1.00000000000000e+00  -1.00000000000000e+00
  1.25000000000000e-01  -9.84616219866031e-01  -9.84694122192634e-01
  2.50000000000000e-01  -9.42183667707485e-01  -9.42321696047380e-01
  3.75000000000000e-01  -8.81535692972130e-01  -8.81713259065582e-01
  5.00000000000000e-01  -8.12905350940037e-01  -8.13108051762390e-01
  6.25000000000000e-01  -7.44750739618239e-01  -7.44968398434756e-01
  7.50000000000000e-01  -6.82541834540626e-01  -6.82765643676167e-01
  8.75000000000000e-01  -6.29055043947684e-01  -6.29277898872839e-01
  1.00000000000000e+00  -5.85237223249823e-01  -5.85454927933219e-01
  1.12500000000000e+00  -5.51029991968942e-01  -5.51241535816887e-01
  1.25000000000000e+00  -5.25950449849148e-01  -5.26157441623585e-01
  1.37500000000000e+00  -5.09438816871234e-01  -5.09644723324546e-01
  1.50000000000000e+00  -5.01044458865884e-01  -5.01254075466067e-01
  1.62500000000000e+00  -5.00515567745838e-01  -5.00734868969654e-01
  1.75000000000000e+00  -5.07834747063113e-01  -5.08071130683891e-01
  1.87500000000000e+00  -5.23220642316692e-01  -5.23483556447791e-01
  2.00000000000000e+00  -5.47096819785032e-01  -5.47398763409569e-01
  2.12500000000000e+00  -5.80010475734566e-01  -5.80368317941351e-01
  2.25000000000000e+00  -6.22462149095108e-01  -6.22898535109136e-01
  2.37500000000000e+00  -6.74584547449058e-01  -6.75128765013705e-01
  2.50000000000000e+00  -7.35598355651468e-01  -7.36285446271368e-01
  2.62500000000000e+00  -8.03019629131483e-01  -8.03886139055434e-01
  2.75000000000000e+00  -8.71779980469990e-01  -8.72855417154624e-01
  2.87500000000000e+00  -9.33805409104996e-01  -9.35100481243730e-01
  3.00000000000000e+00  -9.78979724229674e-01  -9.80474000800670e-01
  3.12500000000000e+00  -9.98091990751450e-01  -9.99724784876465e-01
  3.25000000000000e+00  -9.86754602476125e-01  -9.88429262087319e-01
  3.37500000000000e+00  -9.47607225955206e-01  -9.49219754392125e-01
  3.50000000000000e+00  -8.88956603469618e-01  -8.90433198585156e-01
  3.62500000000000e+00  -8.21031262165148e-01  -8.22342781435575e-01
  3.75000000000000e+00  -7.52609959938739e-01  -7.53759938633136e-01
  3.87500000000000e+00  -6.89551556652021e-01  -6.90558698753080e-01
  4.00000000000000e+00  -6.34941310537546e-01  -6.35828955175172e-01
```

The `gnuplot` script

```
1 set terminal postscript enhanced color eps font "Courier-Bold"
2 set output 'plotiv.eps'
3 set size 0.7,0.7
4 set key spacing 1.4
5 plot \
6     2.0/(cos(2*x)-3) ti "yexact(x)",\
7     "qiv.dat" pt 7 ps 1.5 ti "approximation"
```

plotted the exact and approximate solutions on the same graph.



Visually the exact and approximate solution are indistinguishable; however, from the output of the program one can see the approximation is accurate to about 2 significant digits.

(v) Define the error

$$E(h) = \max\left\{\, |y_n - y(x_n)| : n = 0, 1, \ldots, N \,\right\}.$$

If $E(h) \approx Kh^p$ for some K and some p we say that the method is of order p. Numerically determine the order of the Shu–Osher method by approximating the differential equation in part (iii) using $N = 2^j$ for $j = 5, 6, \ldots, 16$. Plot the corresponding values of $E(h)$ versus h using log-log coordinates. Include source code, the numerical values of $E(h)$ as well as the resulting plot in your report.

I wrote a program in Julia

```
# Question v

using Printf

function shu_osher(xn,yn)
    k1 = f(xn,yn)
    k2 = f(xn+h,yn+h*k1)
    k3 = f(xn+h/2,yn+h*(k1+k2)/4)
    return yn + h*(k1+k2+4*k3)/6
end
function f(x,y)
    return y^2*sin(2*x)
end
function yexact(x)
```

```
    return 2/(cos(2*x)-3)
end

x0 = 0
y0 = -1
h = 0
@printf("#%21s %22s\n","h","E(h)")
for j=5:16
    global h
    N=2^j
    h = 4/N
    E=0
    yn=y0
    xn=x0
    for n=1:N
        yn = shu_osher(xn,yn)
        xn = x0 + n*h
        en = abs(yn-yexact(xn))
        if E<en
            E=en
        end
    end
    @printf("%22.14e %22.14e\n",h,E)
end
```

to compute error. The output was

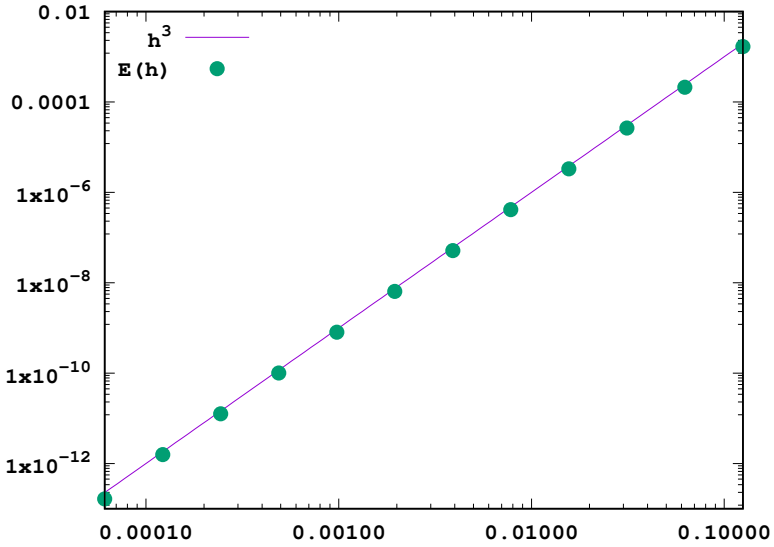
```
#                     h                     E(h)
  1.25000000000000e-01   1.67465961119362e-03
  6.25000000000000e-02   2.10751084668948e-04
  3.12500000000000e-02   2.63816429241226e-05
  1.56250000000000e-02   3.30023661709866e-06
  7.81250000000000e-03   4.12561332097106e-07
  3.90625000000000e-03   5.15711269244434e-08
  1.95312500000000e-03   6.44641984237637e-09
  9.76562500000000e-04   8.05805755454969e-10
  4.88281250000000e-04   1.00725761065235e-10
  2.44140625000000e-04   1.25899290992493e-11
  1.22070312500000e-04   1.57340807049877e-12
  6.10351562500000e-05   1.64868119156836e-13
```

The `gnuplot` script

```
set terminal postscript enhanced color eps font "Courier-Bold"
set output 'plotv.eps'
set size 0.7,0.7
```

```
4 set logscale x
5 set logscale y
6 set key left top
7 set key spacing 1.4
8 plot \
9     x**3 ti "h^3",\
10    "qv.dat" pt 7 ps 1.5 ti "E(h)"
```

plotted the $E(h)$ and h^3 in log-log coordinates on the same graph.



Note that the graphs for $E(h)$ and h^3 agree. Thus, it appears $E(h) = \mathcal{O}(h^3)$ and in particular that the accuracy of the approximation increases as a third-order method.

(vi) Theoretically the value of p found in part (v) should satisfy $p \approx m - 1$ where m is determined by the truncation error in part (i). Use this fact to verify that your computer program is working as expected and discuss how well the theory holds.

As already noted in the previous problem, the convergence for the Shu–Osher method in our calculation behaves as $\mathcal{O}(h^3)$. Since the truncation error found in part (i) was $\mathcal{O}(h^4)$ and $4 - 1 = 3$, this provides evidence that the code is working correctly and as expected.

(vii) [Extra Credit and for Math 667] Note that the exact solution found in part (iii) is bounded for all time $t > 0$. Explore the stability of the numerical approximation by choosing larger and larger values h to find the maximum value h_* such that the resulting numerical approximation y_n remains bounded for all n. Is it possible to relate the linear stability domain determined in part (ii) to the value of h_* by linearizing the non-linear equation in part (iii) about the exact solution?

Since the exact solution is always between -1 and -0.5, then to test for stability I have chosen to perform 100000 time steps at the selected step size h while checking whether the solution remains in absolute magnitude less than 100. If it does, the scheme is considered

stable for that value of h; otherwise, the method is considered unstable. After finding an interval of length 0.1 for which the left endpoint is stable and the right unstable, the program then bisects to find the value of h for the cutoff. In particular, the Julia program

```
# Question vii

function shu_osher(xn,yn,h)
    k1 = f(xn,yn)
    k2 = f(xn+h,yn+h*k1)
    k3 = f(xn+h/2,yn+h*(k1+k2)/4)
    return yn + h*(k1+k2+4*k3)/6
end
function f(x,y)
    return y^2*sin(2*x)
end
function unstable(h)
    x0 = 0
    y0 = -1
    yn=y0
    xn=x0
    for n=1:100000
        yn = shu_osher(xn,yn,h)
        xn = x0 + n*h
        if abs(yn)>100
            return true
        end
    end
    return false
end

a=0.1
b=a
while !unstable(b)
    global a,b
    a=b
    b=b+0.1
end
for i=1:54
    global a,b
    h=(a+b)/2
    if unstable(h)
        b=h
    else
        a=h
    end
```

```
42 end
43 println('[',a,',',b,']')
```

produced the output

```
[2.083612541028182,2.0836125410281823]
```

Note the program was also rerun using 1000000 time steps (not shown) to verify that the same answer was obtained. We conclude that $h^* \approx 2.083612541028182$.

To relate this to the stability domain, note the stability domain extends to about -2.5. Linearizing and plugging in the solution for y we find

$$f_y(x,y)\,dy = 2y\sin 2x\,dy = \frac{4\sin 2x}{\cos 2x - 3}\,dy \approx A\,dy.$$

Therefore

$$A \approx \frac{4\sin 2x}{\cos 2x - 3},$$

which oscillates between $-\sqrt{2}$ and $\sqrt{2}$. For negative values of Ah to remain within the stability domain it follows that

$$-2.5 \le (-\sqrt{2})h \qquad \text{or equivalently} \qquad h < 2.5/\sqrt{2} \approx 1.768.$$

In comparison, the empirically observed value $h_* \approx 2.083$ is slightly larger (possibly due to lucky averaging effects related to the fluctuations) but on the same order of magnitude.