Mathematical Tripos, Part II
Computational project

Soliton Solutions of the KdV Equation

University of Cambridge

October 21, 2021

Contents

1 Question 1

The KdV equation is

$$u_t + uu_x + \delta^2 u_{xxx} = 0. \tag{1}$$

First we consider the general single-soliton solution $f(x-ct)$ into the KdV equation, we obtain the ODE:

$$-cf' + ff' + \delta^2 f''' = 0$$

Then we verify $f(x) = A\text{sech}^2(\frac{x-x_0}{\Delta})$ satisfy the resultant ODE. By direct substitution, writing $X = \frac{x-x_0}{\Delta}$, we have

$$\frac{2c}{\Delta}\text{sech}^2 X \text{tanh} X - \frac{2A}{\Delta}\text{sech}^4 X \text{tanh} X + \frac{8\delta^2}{\Delta^3}(2\tanh X \text{sech}^4 X - \text{tanh}^3 X \text{sech}^2 X)$$

By taking out the common factor$\frac{2}{\Delta}\operatorname{sech}^2 X \tanh X$, we obtain

$$c - A\operatorname{sech}^2 X + \frac{4\delta^2}{\Delta^2}(2\operatorname{sech}^2 X - \tanh^2 X)$$

then by using $\Delta^2 = \frac{12\delta^2}{A}$, $c = \frac{A}{3}$ and $\operatorname{sech}^2 X + \tanh^2 X = 1$,

$$= \frac{A}{3}(1 + 2\operatorname{sech}^2 X - \tanh^2 X - 3\operatorname{sech}^2 X) = 0$$

Therefore, the single soliton is indeed a solution of the KdV equation.
Now suppose we have cyclic boundary conditions

$$u(x+1,t) = u(x,t). \tag{2}$$

We define

$$M \equiv \int_0^1 u(x,t)dx \quad and \quad E \equiv \int_0^1 \frac{1}{2}u^2(x,t)dx.$$

Take the time derivative of M, we have:

$$\frac{dM}{dt} = \frac{d}{dt}\int_0^1 u(x,t)dx = \int_0^1 u_t(x,t)dx.$$

By the KdV equation 1,

$$= -\int_0^1 uu_x + \delta^2 u_{xxx} dx = -\left[\frac{u^2}{2} + \delta^2 u_{xx}\right]_0^1$$

which is 0 by the cyclic boundary conditions 2.
Take the time derivative of E, we have:

$$\frac{dE}{dt} = \frac{d}{dt}\int_0^1 \frac{1}{2}u^2(x,t)dx = \int_0^1 uu_t dx.$$

By the KdV equation 1,

$$= -\int_0^1 u^2 u_x + \delta^2 u u_{xxx} dx = -\left[\frac{u^3}{3} - \delta^2 u u_{xx}\right]_0^1 + \delta^2 \int_0^1 u_x u_{xx} dx$$

$$= \left[\delta^2 \left(u u_{xx} + \frac{u_x^2}{2}\right) - \frac{u^3}{3}\right]_0^1$$

which is 0 by the cyclic boundary conditions 2. Hence both M and E are first integrals of the KdV equation.

2 Question 2

By substituting the exact solution, we have the local truncation error:

$$\begin{aligned} e_{h,k} =& 2(u + 2uu_x + \delta^2 u_{xxx}) + \frac{k^2}{3} u_{ttt} \\ &+ \frac{h^2}{6}(3\delta^2 u_{xxxxx} + 2(u_{xxx} + u_x u_{xx})) + \mathcal{O}(k^4 + h^4) \\ =& \frac{k^2}{3} u_{ttt} + \frac{h^2}{6}(3\delta^2 u_{xxxxx} + 2(u_{xxx} + u_x u_{xx})) + \mathcal{O}(k^4 + h^4) \\ =& \mathcal{O}(k^2 + h^2). \end{aligned}$$

So the numerical scheme has order $\mathcal{O}(k^2 + h^2)$.
Rescale KdV equation with $t \to \tau t$, $x \to Lx$, we obtain:

$$\frac{u_t}{\tau} + \frac{uu_x}{L} + \frac{\delta^2}{L^3} u_{xxx} = 0.$$

Therefore, if we choose the scales $\tau = L = \delta$, we obtain the KdV equation with $\delta = 1$.
To ensure our numerical method converges for the rescaled equation, we should alter our numerical scheme accordingly with $k \to \tau k$ and $h \to Lh$. Since the stability condition for $\delta = 1$, the **rescaled** equation, is given by

$$k \leqslant \frac{h^3}{4 + h^2 |u_{max}|},$$

we conclude that the stability condition for the original equation ($\delta \neq 1$) is

$$(\delta)^{-1} k \leqslant \frac{\delta^{-3} h^3}{4 + \delta^{-2} h^2 |u_{max}|},$$

i.e.

$$k \leqslant \frac{h^3}{4\delta^2 + h^2 |u_{max}|}.$$

The program can be found in A.
I used a explicit Euler scheme for the first step. It is easy to execute and not hard to obtain by adjusting the coefficients of the nonlinear and dispersive parts. [1] The domain is chosen from -10 to 10, which is enough to show the soliton pattern. I chose $h = 0.05$ and $k = \frac{h^2}{4\delta^2 + h^2 A} = 0.0145$. The choice of k is to maintain stability, and the choice for h is such that the size of k is large enough to make sure of a manageable number of iterations to larger times. The amplitude of the exact solution is used in place of $|u_{max}|$. The numerical solution has two peaks. The propagation speed of the numerical solution is greater than that of the exact solution.

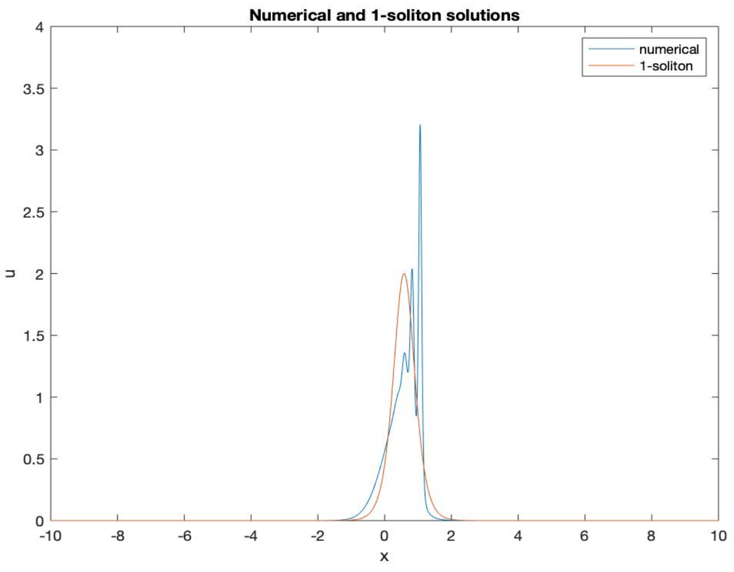
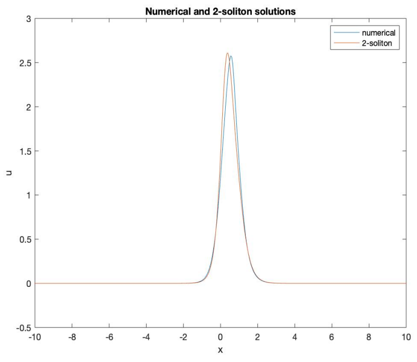


Figure 1.1: Numerical and 1-soliton solutions at t = 0.5

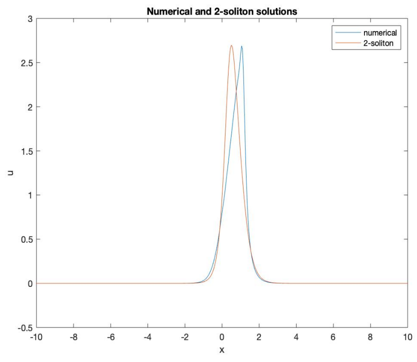
[1] Alternatively, one could try a Crank-Nicolson for the first step. It is implicit, unconditionally stable and is second order in time, the same as the leap-frog method and hence the order of the whole method is preserved. But for such a scheme you will need to solve a nonlinear difference equation, which is hard. So here for simplicity reason I tried the explicit Euler, and it worked well.

3 Question 3

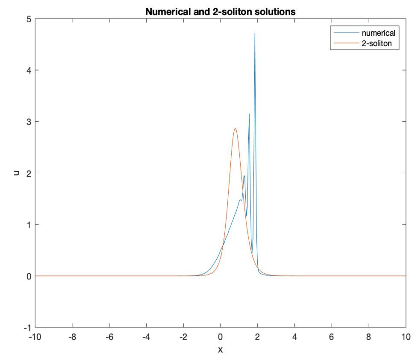
A right-propagating many-peak solution is to be expected. Evolution at representative times:



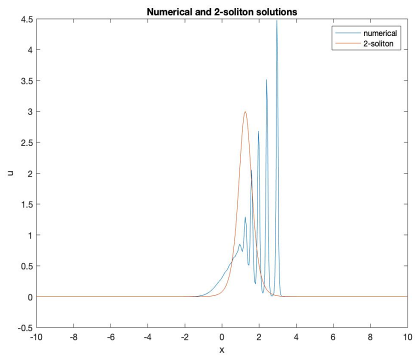
(a) Figure 3.1: Numerical and 2-soliton solutions at t = 0.1



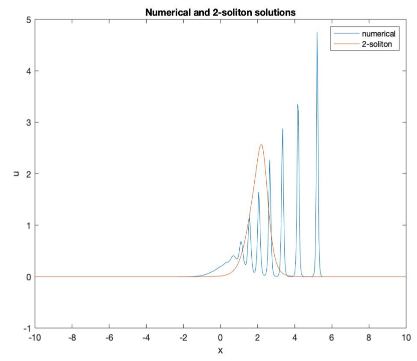
(b) Figure 3.2: Numerical and 2-soliton solutions at t = 0.3



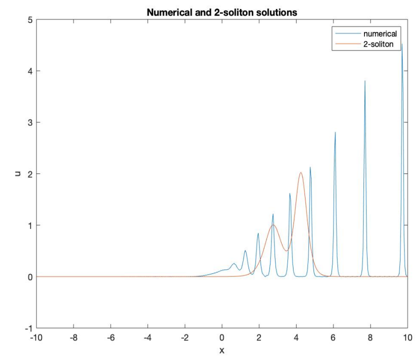
(c) Figure 3.3: Numerical and 2-soliton solutions at t = 0.75



(d) Figure 3.4: Numerical and 2-soliton solutions at t = 1.5



(e) Figure 3.5: Numerical and 2-soliton solutions at t = 3



(f) Figure 3.6: Numerical and 2-soliton solutions at t = 6

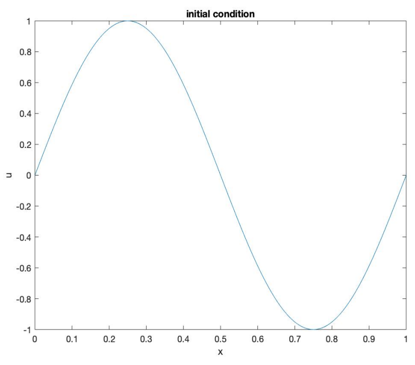
The following table[2] shows the numerical 'mass' and 'energy' at these times, up to 4 significant figures. The variation of mass with time is small. And it can be observed that it is at least of $\mathcal{O}(10^5)$. The variation in energy is much larger, and from the 2-soliton case we can see that the energy is not conserved in time. Which means it is **not a solution** of the KdV equation. It also demonstrates the non-linearity of the Kdv equation as the superposition of two solutions is **not** another solution.

(+k)	Numerical M	Numerical E	2-soliton M	2-soliton E
0.11452	57.9411	48.9154	57.9411	49.7327
0.3	57.9411	48.9145	57.9411	51.3766
0.75	57.9411	50.0413	57.9411	54.5506
1.5	57.9411	50.4534	57.9411	56.8920
3	57.9411	50.4967	57.9411	48.8834

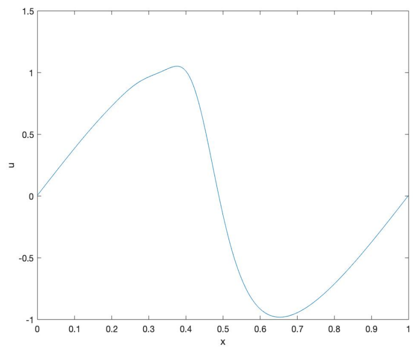
4 Question 4

If $\delta = 0$ we can use the method of characteristics to solve the problem. The characteristic is $x - ut$, hence $u(x,t) = f(x - ut)$. Therefore, the solution satisfies $u = \sin(2\pi x - ut)$. Since the gradient u is not monotonic in x, the characteristics cross and $|\frac{\partial u}{\partial x}|$ becomes unbounded. So we get shocks. By comparing the magnitudes of the nonlinear term uu_x and the dispersive term $\delta^2 u_{xxx}$, we know that when $\frac{U^2}{L} \approx \frac{\delta^2 U}{L^3}$, i.e. $L \approx \frac{\delta}{U}$, the two terms balance. At first nonlinearity dominates and the peaks and troughs deepen. It will tend to forming a shock but before it occurs, the length scale will contract till it is small enough to balance the terms. After that the dispersion dominates and we are expected to get well-defined waves. The second peak emerges at $t \approx 0.09$, after that more peaks emerge and we should expect a train of waves with taller waves goes faster and shorter wave goes slower (which comes from the shallow water wave velocity$c^2 = g(h + a)$ where a is the amplitude of a soliton-like solution). The taller waves goes through the shorter waves and then reappear from the other end.

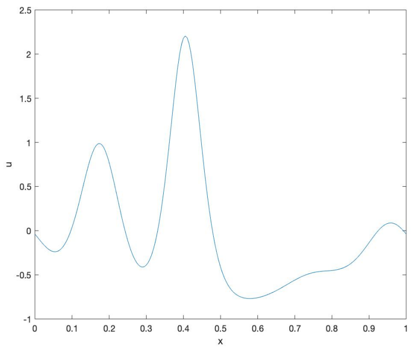
[2]Values in the first column should all +k $\approx 3 * 10^{-5}$ (time-step)



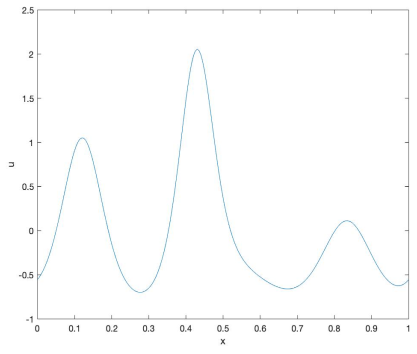
(a) Figure 4.1: Initial condition



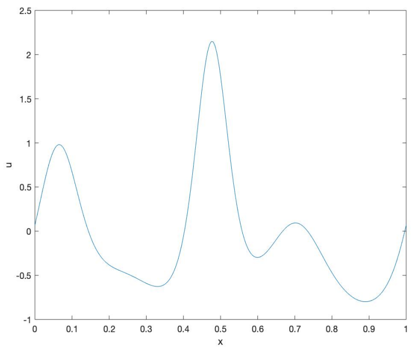
(b) Figure 4.2: Numerical solution at t = 0.1



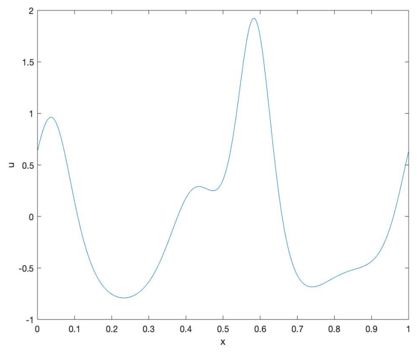
(c) Figure 4.3: Numerical solution at t = 0.33



(d) Figure 4.4: Numerical solution at t = 0.5



(e) Figure 4.5: Numerical solution at t = 0.7



(f) Figure 4.6: Numerical solution at t = 0.95

From the numerical solution, the estimate is $t = 0.3327$ to 4 s.f.s as it is when steepening tops.
The time is earlier for large δ and later for smaller δ, which is clearly because when δ is larger, the dispersion dominates quicker, and the balance between

nonlinearity and dispersion is obtained sooner.

References

[1] Drazin, P.G. & Johnson, R.S. *Solitons: An Introduction* CUP, 1989.

A Codes

'1-soliton'

```
%prep
xmax = 10;%input('Enter a length')
xmin = -10;
tmax = 0.5;%input('Enter a time');
h = 0.005;%input('Enter a spacing')
N = round((xmax-xmin)/h);
a = 2;%input('Enter an amplitude');
delta = 0.03;%input('Enter a scale(delta)')
k = (h^3)/(4*delta^2+a*h^2);%input('Enter a value for time step');
M = round(tmax/k-1);
x0 = 0.25;%;input('Enter an initial location');
c = a/3;
D = sqrt(delta^2*12*a^-1);
xgrid = xmin:h:xmax;
xgrid = xgrid';

%analytic
soliton = @(x,t) a*(sech(sqrt(a/(delta*12))*(x-c*t-x0)).^2);
soli = soliton(xgrid,tmax);

%numerical
uinit = soliton(xgrid,0);

u1 = zeros(N+1,1);
u2 = u1;

alpha = k/(3*h);
beta = (k*delta^2)/(h^3);


%First step
u1(1) = uinit(1)-0.5*alpha*(uinit(2)+uinit(1) ...
     +uinit(N))*(uinit(2)-uinit(N)) ...
      -0.5*beta*(uinit(3)-2*uinit(2)+2*uinit(N)-uinit(N-1));

u1(2) = uinit(2)-0.5*alpha*(uinit(3)+uinit(2) ...
     +uinit(1))*(uinit(3)-uinit(1)) ...
      -0.5*beta*(uinit(4)-2*uinit(3)+2*uinit(1)-uinit(N));
```

```
u1(N) = uinit(N)-0.5*alpha*(uinit(N+1)+uinit(N)...
    +uinit(N-1))*(uinit(N+1)-uinit(N-1)) ...
    -0.5*beta*(uinit(2)-2*uinit(N+1)+2*uinit(N-1)-uinit(N-2));

u1(N+1) = uinit(N+1)-0.5*alpha*(uinit(2)+uinit(N+1) ...
    +uinit(N))*(uinit(2)-uinit(N)) ...
    -0.5*beta*(uinit(3)-2*uinit(2)+2*uinit(N)-uinit(N-1));


for n = 3:N-1
    u1(n) = uinit(n)-0.5*alpha*(uinit(n+1)+uinit(n) ...
        +uinit(n-1))*(uinit(n+1)-uinit(n-1)) ...
        -0.5*beta*(uinit(n+2)-2*uinit(n+1)+2*uinit(n-1)-uinit(n-2));
end


u0 = uinit;
%leapfrog
for m = 1:M

    u2(1) = u0(1)-alpha*(u1(2)+u1(1)+u1(N))*(u1(2)-u1(N)) ...
        -beta*(u1(3)-2*u1(2)+2*u1(N)-u1(N-1));
    u2(2) = u0(2)-alpha*(u1(3)+u1(2)+u1(1))*(u1(3)-u1(1)) ...
        -beta*(u1(4)-2*u1(3)+2*u1(1)-u1(N));

    for n = 3:N-1
        u2(n) = u0(n)-alpha*(u1(n+1)+u1(n)+u1(n-1)) ...
            *(u1(n+1)-u1(n-1)) ...
            -beta*(u1(n+2)-2*u1(n+1)+2*u1(n-1)-u1(n-2));
    end

    u2(N) = u0(N)-alpha*(u1(N+1)+u1(N) ...
        +u1(N-1))*(u1(N+1)-u1(N-1))- ...
        beta*(u1(2)-2*u1(N+1)+2*u1(N-1)-u1(N-2));
    u2(N+1) = u0(N+1)-alpha*(u1(2)+u1(N+1) ...
        +u1(N))*(u1(2)-u1(N))- ...
        beta*(u1(3)-2*u1(2)+2*u1(N)-u1(N-1));

    u0 = u1;
    u1 = u2;


end
```

```
error = abs(u2 - soli);

plot(xgrid,u2)
hold on
plot(xgrid,soli)
axis([xmin xmax 0 2*a]);
xlabel x
ylabel u
title('Numerical and 1-soliton solutions')
legend('numerical', '1-soliton')
```

'2-soliton'

```
%prep
xmax = 10; %input('Enter a length')
xmin = -10;
tmax = 0.1;%input('Enter a time');
h = 0.05; %input('Enter a spacing')
N = round((xmax-xmin)/h);
a1 = 2; a2 = 1;%input('Enter an amplitude');
delta = 0.03;%input('Enter a scale(delta)')
k = (h^3)/(4*delta^2+a1*h^2);%input('Enter a value for time step');
M = round(tmax/k-1);
x01 = 0.25; x02 = 0.75;%;input('Enter an initial location');
c1 = a1/3; c2 = a2/3;
D1 = sqrt(delta^2*12*a1^-1); D2 = sqrt(delta^2*12*a2^-1);
xgrid = xmin:h:xmax;
xgrid = xgrid';

%analytic
twosoliton = @(x,t) a1*(sech(sqrt(a1/(delta*12))*(x-c1*t-x01)).^2) ...
    + a2*(sech(sqrt(a2/(delta*12))*(x-c2*t-x02)).^2);
soli = twosoliton(xgrid,tmax);

%numerical
uinit = twosoliton(xgrid,0);

u1 = zeros(N+1,1);

alpha = k/(3*h);
beta = (k*delta^2)/(h^3);
```

```
%First Step
u1(1) = uinit(1)-0.5*alpha*(uinit(2)+uinit(1) ...
    +uinit(N))*(uinit(2)-uinit(N)) ...
    -0.5*beta*(uinit(3)-2*uinit(2)+2*uinit(N)-uinit(N-1));

u1(2) = uinit(2)-0.5*alpha*(uinit(3)+uinit(2) ...
    +uinit(1))*(uinit(3)-uinit(1)) ...
    -0.5*beta*(uinit(4)-2*uinit(3)+2*uinit(1)-uinit(N));

u1(N) = uinit(N)-0.5*alpha*(uinit(N+1)+uinit(N)...
    +uinit(N-1))*(uinit(N+1)-uinit(N-1)) ...
    -0.5*beta*(uinit(2)-2*uinit(N+1)+2*uinit(N-1)-uinit(N-2));

u1(N+1) = uinit(N+1)-0.5*alpha*(uinit(2)+uinit(N+1) ...
    +uinit(N))*(uinit(2)-uinit(N)) ...
    -0.5*beta*(uinit(3)-2*uinit(2)+2*uinit(N)-uinit(N-1));

for n = 3:N-1
    u1(n) = uinit(n)-0.5*alpha*(uinit(n+1)+uinit(n) ...
        +uinit(n-1))*(uinit(n+1)-uinit(n-1)) ...
        -0.5*beta*(uinit(n+2)-2*uinit(n+1)+2*uinit(n-1)-uinit(n-2));
end


u0 = uinit;

u2 = zeros(N+1,1);
%leapfrog
for m = 1:M

    u2(1) = u0(1)-alpha*(u1(2)+u1(1)+u1(N))*(u1(2)-u1(N)) ...
        -beta*(u1(3)-2*u1(2)+2*u1(N)-u1(N-1));
    u2(2) = u0(2)-alpha*(u1(3)+u1(2)+u1(1))*(u1(3)-u1(1)) ...
        -beta*(u1(4)-2*u1(3)+2*u1(1)-u1(N));


    for n = 3:N-1
        u2(n) = u0(n)-alpha*(u1(n+1)+u1(n)+u1(n-1)) ...
            *(u1(n+1)-u1(n-1)) ...
            -beta*(u1(n+2)-2*u1(n+1)+2*u1(n-1)-u1(n-2));
    end
    u2(N) = u0(N)-alpha*(u1(N+1)+u1(N) ...
        +u1(N-1))*(u1(N+1)-u1(N-1))- ...
        beta*(u1(2)-2*u1(N+1)+2*u1(N-1)-u1(N-2));
    u2(N+1) = u0(N+1)-alpha*(u1(2)+u1(N+1) ...
        +u1(N))*(u1(2)-u1(N))- ...
        beta*(u1(3)-2*u1(2)+2*u1(N)-u1(N-1));


    u0 = u1;
    u1 = u2;
end


mass = trapz(u2);
energy = 1/2*trapz(u2.^2);
disp([mass, energy]);
error = abs(u2 - soli);

plot(xgrid,u2)
hold on
plot(xgrid,soli)
%axis([xmin xmax 0 2*a]);
xlabel x
ylabel u
title('Numerical and 2-soliton solutions')
legend('numerical', '2-soliton')
```

'last question'

```
function umax = Q4(tmax)
%prep
xmax = 1;%input('Enter a length')
xmin = 0;
%tmax = 0.95;%input('Enter a time');
h = 0.005;%input('Enter a spacing')
N = round((xmax-xmin)/h);
delta = 0.03;%input('Enter a scale(delta)')
k = (h^3)/(4*delta^2+h^2);%input('Enter a value for time step');
M = round(tmax/k-1);
xgrid = xmin:h:xmax;
xgrid = xgrid';
```

```
%numerical
uinit = sin(pi*2*xgrid);

u1 = zeros(N+1,1);
u2 = u1;

alpha = k/(3*h);
beta = (k*delta^2)/(h^3);


%First Step
u1(1) = uinit(1)-0.5*alpha*(uinit(2)+uinit(1) ...
    +uinit(N))*(uinit(2)-uinit(N)) ...
    -0.5*beta*(uinit(3)-2*uinit(2)+2*uinit(N)-uinit(N-1));

u1(2) = uinit(2)-0.5*alpha*(uinit(3)+uinit(2) ...
    +uinit(1))*(uinit(3)-uinit(1)) ...
    -0.5*beta*(uinit(4)-2*uinit(3)+2*uinit(1)-uinit(N));

u1(N) = uinit(N)-0.5*alpha*(uinit(N+1)+uinit(N)...
    +uinit(N-1))*(uinit(N+1)-uinit(N-1)) ...
    -0.5*beta*(uinit(2)-2*uinit(N+1)+2*uinit(N-1)-uinit(N-2));

u1(N+1) = uinit(N+1)-0.5*alpha*(uinit(2)+uinit(N+1) ...
    +uinit(N))*(uinit(2)-uinit(N)) ...
    -0.5*beta*(uinit(3)-2*uinit(2)+2*uinit(N)-uinit(N-1));

for n = 3:N-1
    u1(n) = uinit(n)-0.5*alpha*(uinit(n+1)+uinit(n) ...
        +uinit(n-1))*(uinit(n+1)-uinit(n-1)) ...
        -0.5*beta*(uinit(n+2)-2*uinit(n+1)+2*uinit(n-1)-uinit(n-2));
end


u0 = uinit;

%leapfrog
for m = 1:M

    u2(1) = u0(1)-alpha*(u1(2)+u1(1)+u1(N))*(u1(2)-u1(N)) ...
        -beta*(u1(3)-2*u1(2)+2*u1(N)-u1(N-1));
    u2(2) = u0(2)-alpha*(u1(3)+u1(2)+u1(1))*(u1(3)-u1(1)) ...
        -beta*(u1(4)-2*u1(3)+2*u1(1)-u1(N));
```

```
          for n = 3:N-1
               u2(n) = u0(n)-alpha*(u1(n+1)+u1(n)+u1(n-1)) ...
                    *(u1(n+1)-u1(n-1)) ...
                    -beta*(u1(n+2)-2*u1(n+1)+2*u1(n-1)-u1(n-2));
          end

          u2(N) = u0(N)-alpha*(u1(N+1)+u1(N) ...
               +u1(N-1))*(u1(N+1)-u1(N-1))- ...
               beta*(u1(2)-2*u1(N+1)+2*u1(N-1)-u1(N-2));
          u2(N+1) = u0(N+1)-alpha*(u1(2)+u1(N+1) ...
               +u1(N))*(u1(2)-u1(N))- ...
               beta*(u1(3)-2*u1(2)+2*u1(N)-u1(N-1));

          u0 = u1;
          u1 = u2;
     end

     umax = max(u2);
     plot(xgrid,u2)
     xlabel x
     ylabel u
     end
```