

Problem 1

Consider the following initial value problem

$$\begin{aligned} u_t &= u_x, & 0 < x < 2\pi, & \quad t > 0 \\ u(x, 0) &= \sin(x) \\ u(0, t) &= u(2\pi, t) \end{aligned}$$

Notation:

$$v_j^n = u(x_j, t_n)$$

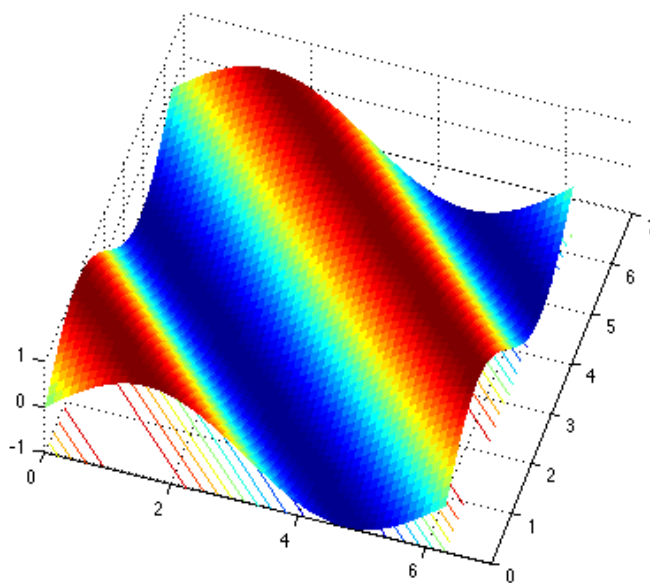
$$k = \Delta t$$

$$h = \Delta x$$

$$\lambda = \frac{k}{h}$$

(a) The exact solution to this problem is $u(x, t) = u_0(x + t) = \sin(x + t)$ (figure 1). Three numerical

Figure 1: Exact solution



methods were implemented (code follows at the end of this report):

(i) **Leap Frog** which is centered difference both in space and time

$$v_j^{n+1} = v_j^{n-1} + \lambda(v_{j+1}^n - v_{j-1}^n)$$

Here I used the Upwind method to compute the first time step from the initial condition, since the Leap Frog method requires two previous time steps.

(ii) **Upwind** which is forward difference both in space and time

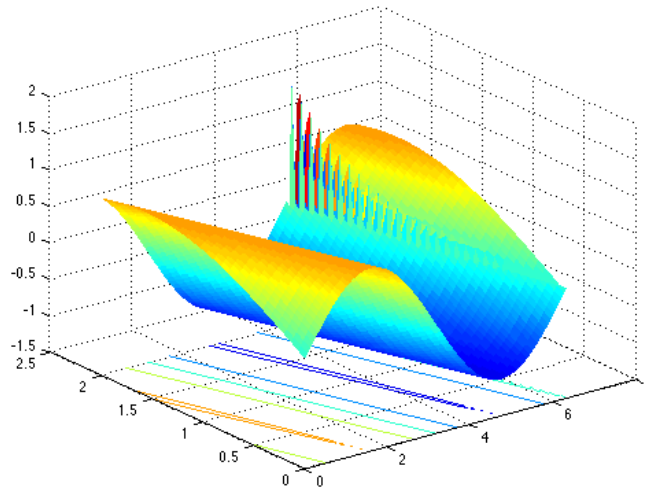
$$v_j^{n+1} = v_j^n + \lambda(v_{j+1}^n - v_j^n)$$

(iii) **Lax-Wendroff** which is forward difference in time and centered in space

$$v_j^{n+1} = v_j^n + \frac{\lambda}{2}(v_{j+1}^n - v_{j-1}^n) + \frac{\lambda^2}{2}(v_{j+1}^n - 2v_j^n + v_{j-1}^n)$$

(b) For all three schemes with $\lambda > 1$ we can observe an instability such as in figure 2, whereas for $\lambda < 1$, all three methods are stable. For $\lambda = 1$ all three also appear stable, even though (according to the text) Leap Frog should not be

Figure 2: Instability of the Upwind scheme for $\lambda = 1.1$



(c) The next four figures show the exact solution (in blue) and three numerical solutions (red, green and black) at four different times (computed with $\lambda = 0.9$ using 101 spatial nodes). Notice how the error propagates from right to the left (which demonstrates a wave traveling from the right to the left). It takes time $= 2 * \pi$ for the error to propagate through the whole interval (which demonstrates the speed of the traveling wave $= 1$). The question remains (and I do not know the answer to this) why is there an error at the right endpoint? Its location suggests that it might be due to the boundary conditions, but I implemented them using the same method as in the interior nodes.

Figure 3: $t=0.1$

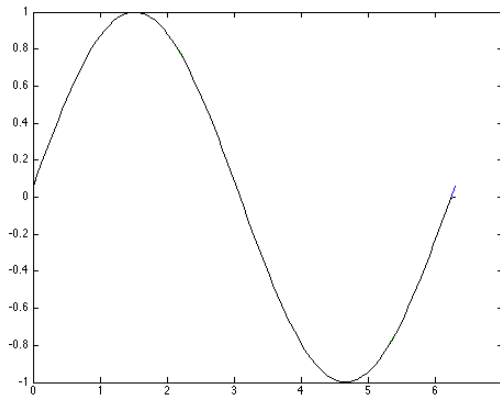


Figure 4: $t=0.5$

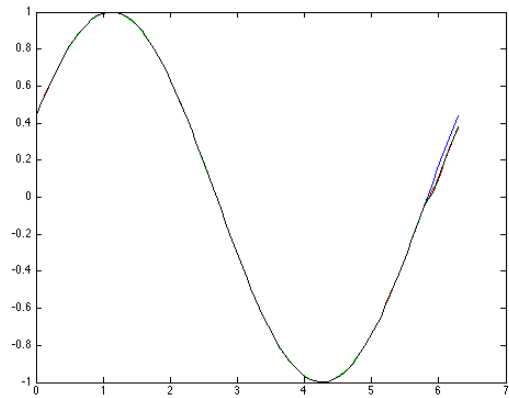
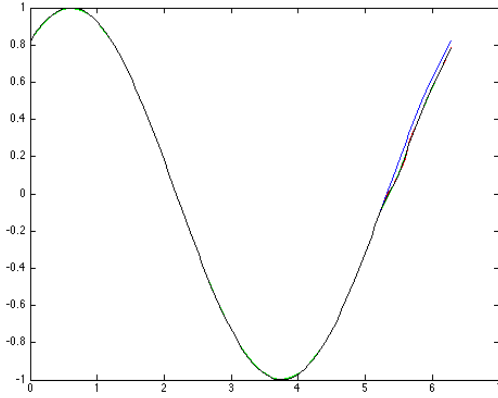
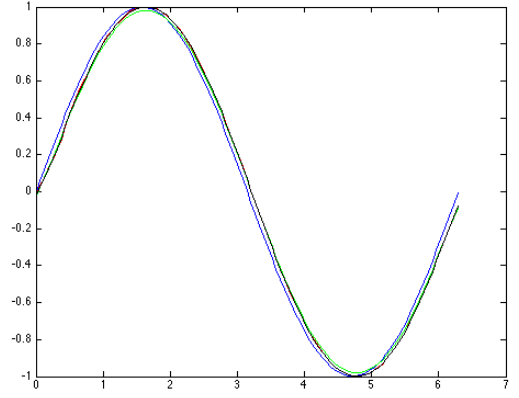


Figure 5: $t=1$ Figure 6: $t=2\pi$ 

(d) In order to compute the order of accuracy of each method, a relative l^2 error was used (computed at $t = 2\pi$ for successively decreasing (halved) values of k , while λ was held constant. In the table we can

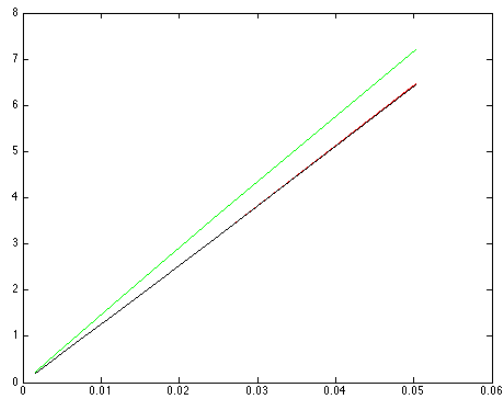
Table 1: Relative error (in %) with respect to the exact solution

k	lf	uw	lw
0.050265482	6.468230633	7.207733750	6.434407102
0.025132741	3.194120693	3.661412223	3.183507851
0.012566371	1.585770992	1.844733852	1.582493879
0.006283185	0.789699788	0.925650435	0.788705573
0.003141593	0.393935880	0.463552334	0.393648608
0.001570796	0.196713247	0.231922525	0.196626404

see that the relative error (approximately) halves as the time step halves. This suggests a first order of accuracy, even though we expect second order for Lax-Wendroff and Leap Frog.

The following figure shows relative l^2 error against grid spacing k . Note that the two lines corresponding

Figure 7: Relative error in %, green=Upwind, red=Leap Frog, black=Lax-Wendroff



to Leap Frog and Upwind methods coincide.

Listing of the Matlab code that I used:

```
%% advection equation on [0,2pi]
% initial condition u(x,0)=sin(x)
% periodic boundary conditions

clear

% final time
T=2*pi;
% number of spatial intervals
N=400;
% spatial step
h=2*pi/N;
% step ratio
lambda=0.8;
% time step
k=lambda*h;
% number of time intervals
M=round(T/k);

%% exact solution
u=zeros(M+1,N+1);

% exact solution u(x,t)=u_0(x+t)
for j=1:N+1 %space
    for n=1:M+1 %time
        u(n,j)=rem(sin((j-1)*h+(n-1)*k),2*pi);
    end
end

% spatial nodes
x=0:h:2*pi;
% time nodes
t=0:k:M*k;
% plot
surf(x,t,u,'EdgeColor','none')
%shading interp

%% leap frog method

% solution
v_lf=zeros(M+1,N+1);

% initial condition
for j=1:N+1 %space
    v_lf(1,j)=sin((j-1)*h);
end

% first time step using upwind
for j=1:N %space
    v_lf(2,j)=v_lf(1,j)+lambda*(v_lf(1,j+1)-v_lf(1,j));
end
% last point uses periodic boundary condition
v_lf(2,N+1)=v_lf(1,N+1)+lambda*(v_lf(1,1)-v_lf(1,N+1));
```

```

for n=3:M+1 %time
    for j=2:N %space
        v_lf(n,j)=v_lf(n-2,j)+lambda*(v_lf(n-1,j+1)-v_lf(n-1,j-1));
    end
    % last point has to use the periodic boundary condition
    v_lf(n,N+1)=v_lf(n-2,N+1)+lambda*(v_lf(n-1,1)-v_lf(n-1,N));
    % first point also uses periodic BCs
    v_lf(n,1)=v_lf(n-2,1)+lambda*(v_lf(n-1,2)-v_lf(n-1,N+1));
end

% plot
figure;
surf(x,t,v_lf,'EdgeColor','none')

%% upwind

% solution
v_uw=zeros(M+1,N+1);

% initial condition
for j=1:N+1 %space
    v_uw(1,j)=sin((j-1)*h);
end

for n=2:M+1 %time
    for j=1:N %space
        v_uw(n,j)=v_uw(n-1,j)+lambda*(v_uw(n-1,j+1)-v_uw(n-1,j));
    end
    % last point has to use the periodic boundary condition
    v_uw(n,N+1)=v_uw(n-1,N+1)+lambda*(v_uw(n-1,1)-v_uw(n-1,N+1));
end

% plot
figure;
surf(x,t,v_uw,'EdgeColor','none')

%% Lax-Wendroff

% solution
v_lw=zeros(M+1,N+1);

% initial condition
for j=1:N+1 %space
    v_lw(1,j)=sin((j-1)*h);
end

for n=2:M+1 %time
    for j=2:N %space
        v_lw(n,j)=v_lw(n-1,j)+lambda*(v_lw(n-1,j+1)-v_lw(n-1,j-1))/2+...
            +(lambda^2)*(v_lw(n-1,j+1)-2*v_lw(n-1,j)+v_lw(n-1,j-1))/2;
    end
    % last point has to use the periodic boundary condition
    v_lw(n,N+1)=v_lw(n-1,N+1)+lambda*(v_lw(n-1,1)-v_lw(n-1,N))/2+...

```

```

        +(lambda^2)*(v_lw(n-1,1)-2*v_lw(n-1,N+1)+v_lw(n-1,N))/2;
    % first point also uses periodic BCs
    v_lw(n,1)=v_lw(n-1,1)+lambda*(v_lw(n-1,2)-v_lw(n-1,N+1))/2+...
        +(lambda^2)*(v_lw(n-1,2)-2*v_lw(n-1,1)+v_lw(n-1,N+1))/2;
end

% plot
figure;
surf(x,t,v_lw,'EdgeColor','none')

%% time plots

% t=0.1
tau=round(0.1/k);
plot(x,u(tau,:))
hold on
plot(x,v_lf(tau,:), 'r')
plot(x,v_uw(tau,:), 'g')
plot(x,v_lw(tau,:), 'k')
hold off

% t=0.5
figure
tau=round(0.5/k);
plot(x,u(tau,:))
hold on
plot(x,v_lf(tau,:), 'r')
plot(x,v_uw(tau,:), 'g')
plot(x,v_lw(tau,:), 'k')
hold off

% t=1
figure
tau=round(1/k);
plot(x,u(tau,:))
hold on
plot(x,v_lf(tau,:), 'r')
plot(x,v_uw(tau,:), 'g')
plot(x,v_lw(tau,:), 'k')
hold off

% t=2*pi
figure
tau=round(2*pi/k)
plot(x,u(M+1,:))
hold on
plot(x,v_lf(M+1,:), 'r')
plot(x,v_uw(M+1,:), 'g')
plot(x,v_lw(M+1,:), 'k')
hold off

%% errors
e_lf=100*norm(v_lf(M+1,:)-u(M+1,:))/norm(u(M+1,:));
e_uw=100*norm(v_uw(M+1,:)-u(M+1,:))/norm(u(M+1,:));

```

```

e_lw=100*norm(v_lw(M+1,:)-u(M+1,:))/norm(u(M+1,:));
e=[k,e_lf,e_uw,e_lw]

% relative errors computed by running the program several times
% with decreasing step size
k_p=[0.050265482,0.025132741,0.012566371,0.006283185,0.003141593,0.001570796]
lf_p=[6.468230633,3.194120693,1.585770992,0.789699788,0.39393588,0.196713247]
uw_p=[7.20773375,3.661412223,1.844733852,0.925650435,0.463552334,0.231922525]
lw_p=[6.434407102,3.183507851,1.582493879,0.788705573,0.393648608,0.196626404]

% plot of the relative error
plot(k_p,lf_p,'r')
hold on
plot(k_p,uw_p,'g')
plot(k_p,lw_p,'k')
hold off

```