ME2055 Homework No. 4

Shervin Sammak, University of Pittsburgh

This homework provides a overview of numerical solutions to the wave equation using the finite difference method. The first upwinding, Lax-Wendroff and Euler's BTCS implicit schemes are developed, and applied to a simple problem involving the one-dimensional wave equation. The results of running the codes on different CFL number is demonstrated. These simple calculations show that the how explicit and implicit schemes convereged to results and how dissipation and disspersion error propagetes in the domain.

Categories and Subject Descriptors: [**ME2055- Spring 2014**]

1. INTRODUCTION

The one dimensional wave equation is

$$\frac{\partial U(x,t)}{\partial t} + a\frac{\partial U(x,t)}{\partial x} = 0 \tag{1}$$

wherein, $t \in [0, 0.15]$ is time; $x \in [0, 70]$ is the spatial variable; $U = U(x,t)$ is the velocity; a is the speed of sound which assumed to be 200 m/s . To make the differential equation well settled, I need to give initial condition.

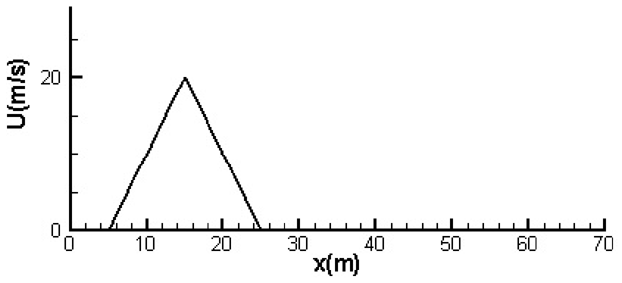

Fig. 1. initial wave distribuation

2. SCHEMES FOR THE HEAT EQUATION

To solve this differential equation via numerical methods, I need to do the following discretization first. I discretize the spatial domain $[0, 70]$ by a uniform mesh with M elements. Then the spatial mesh size is $\Delta x = h = 70/M$. The time domain is discretized by N elements, *i.e.* the mesh size in time direction is $\Delta t = k = 0.15/N$. Then the gird for the discretization is:

$$\Delta t = 0.005,\ 0.0025,\ 0.00125 \tag{2}$$

$$\Delta x = 1.0 \tag{3}$$

For convenience, I denote $U(x_m, t_n)$ as U^n_m. Then I state the numerical method.

2.1 First upwind differncing

I take forward difference for the time derivative and backward difference for the spacial derivative. If I substitute the differential operator by the above finite difference, I may get the first upwind Scheme:

$$\frac{T_m^{n+1} - T_m^n}{k} + a\frac{T_m^n - T_{m-1}^n}{h} \;=\; 0 \tag{4}$$

This method has the accuracy with $[(\Delta t), (\Delta x)]$.

2.2 Lax - Wendroff

When central differencing of the second order for the spatial derivative is used, it follows that

$$U_m^{n+1} \;=\; U_m^n - c\;\frac{U_{m+1}^n - U_{m-1}^n}{2} + c^2\;\frac{U_{m+1}^n - 2U_m^n + U_{m-1}^n}{2} \tag{5}$$

This formulation is known as the Lax - Wendroff method and is of order $[(\Delta t)^2, (\Delta x)^2]$.

2.3 Euler's BTCS implicit

Implicit formulattion of Euler's backward time and central space approximation is unconditionally stable and is of order $[(\Delta t), (\Delta x)^2]$.
This approximation applied to model equation (Eq. (1)) yields:

$$c\;\frac{U_{m-1}^{n+1}}{2} - U_m^{n+1} - c\;\frac{U_{m+1}^{n+1}}{2} \;=\; -U_m^n \tag{6}$$

Once this equation is applied to all grid points at the unknown time level, a set of algebraic equations will result. These equations can be represented in a matrix form, where the coefficient matrix is tridiogonal. The implicit method is just

$$B\mathbf{U}^{n+1} = \mathbf{U}^n \tag{7}$$

This is a process of solving linear system of equations at each time step.

3. STABILITY

For stability condition in two first schemes (first upwind and Lax-Wendroff) , CFL number ($\frac{ak}{h}$) should be lower than 1. Fore three different time steps the CFL number is:

$$c = \frac{ak}{h} \;=\; 1.00, \qquad \text{for } \Delta t = 0.005 \tag{8}$$

$$c = \frac{ak}{h} \;=\; 0.50, \qquad \text{for } \Delta t = 0.0025 \tag{9}$$

$$c = \frac{ak}{h} \;=\; 0.25, \qquad \text{for } \Delta t = 0.00125 \tag{10}$$

4. RESULTS

From the finite difference formulation, it can be seen that if $c = 1$, the solution is exact. The resulting solution for $c = 1$ and Figs. (2-3) a The solution in first upwind and Lax-Wendroff is propagated with minumum difference from initial wave distribuation (exact solution). However it is seen that the Euler's BTCS method have dissipation error which reduce the amplitude of solution.

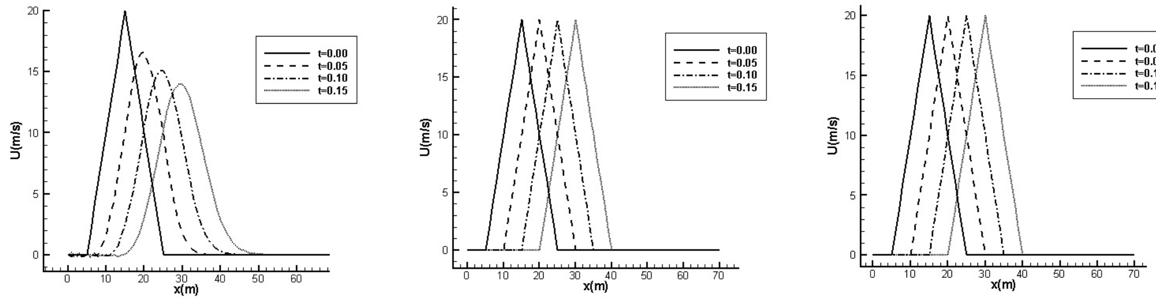

Fig. 2. Wave propagation with c=1, from left to right: Euler's BTCS, Lax-Wendroff and first upwinding

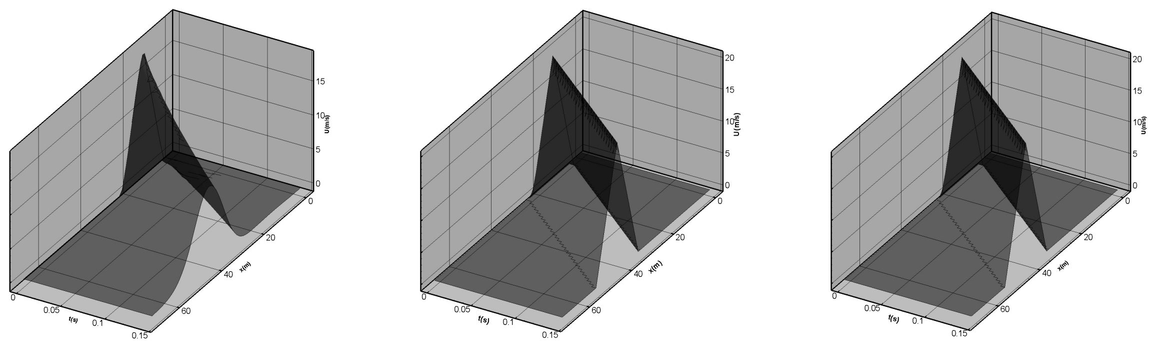

Fig. 3. Solution of first-order wave equation with c=1, from left to right: Euler's BTCS, Lax-Wendroff and first upwinding

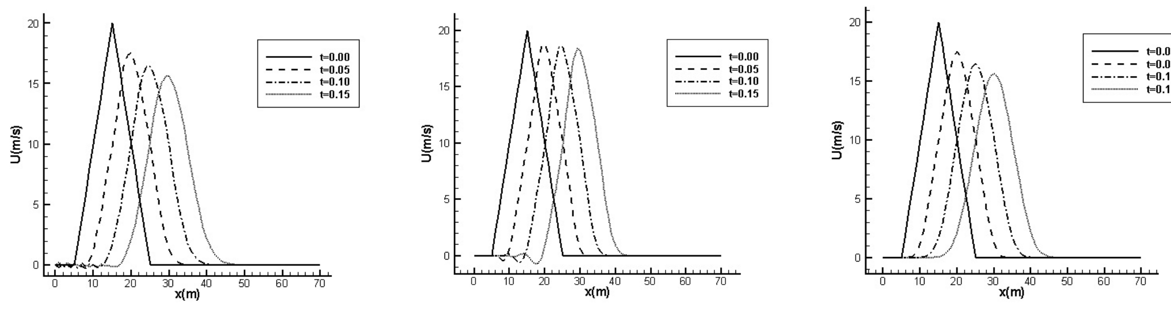

Fig. 4. Wave propagation with c=0.5, from left to right: Euler's BTCS, Lax-Wendroff and first upwinding

As the time step is reduced so that the CFL number is smaller than 1, error appears in the solution. Due to the dissipation error of the mothod, the amplitude of the original function is decreased and the solution dissipated in next time step. The phenomenon is clearly evident in Figs. (4-5). It is also shown that the Lax-Wendroff method has lower dissipation in comparison with two other method.

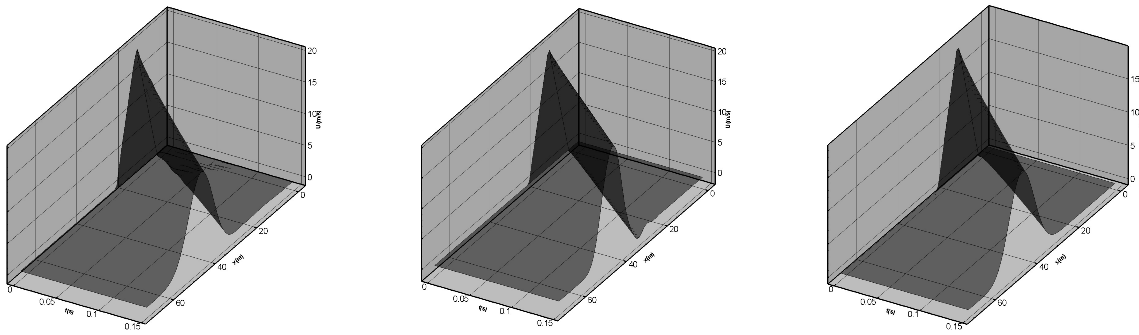
Fig. 5. Solution of first-order wave equation with c=0.5, from left to right: Euler's BTCS, Lax-Wendroff and first upwinding

The solution behaves differently when Lax - Wendroff method is used. Since the algorithm is second-order accurate, some dispersion error is expected. Indeed, the oscillatory behavior of the solition for the smaller CFL number clearly indicates the error developed in the solution. Note that the amplitude of the solution remains the same.

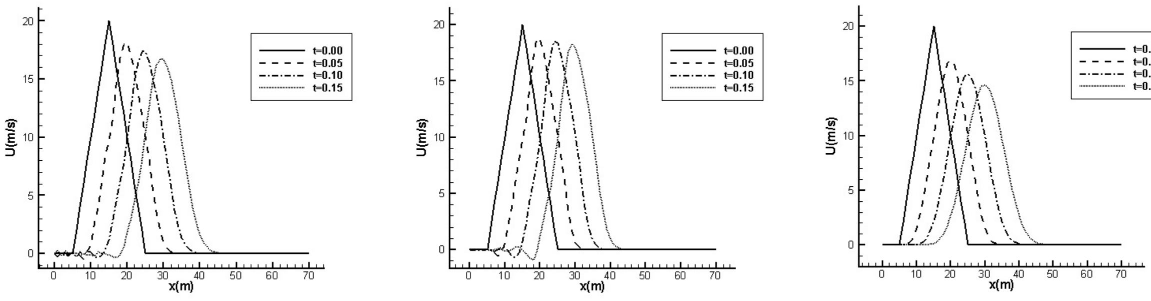

Fig. 6. Wave propagation with c=0.25, from left to right: Euler's BTCS, Lax-Wendroff and first upwinding

5. CONCLUSION

Again, the error is the smallest at the upper limit value of the CFL number, i.e., when the CFL number approaches one. Therefore, the best solution is obtained by selecting step size which yield a CFL number of one, or close to it. These simple applications clearly illustrated the dissipation and dispersion errors. the implict and first upwind method has more dissipation error however dispersion error is higher in Lax-Wendroff method. It should be noted that the as the CFL number gets lower, the amplitude of dissipation error becomes higher.

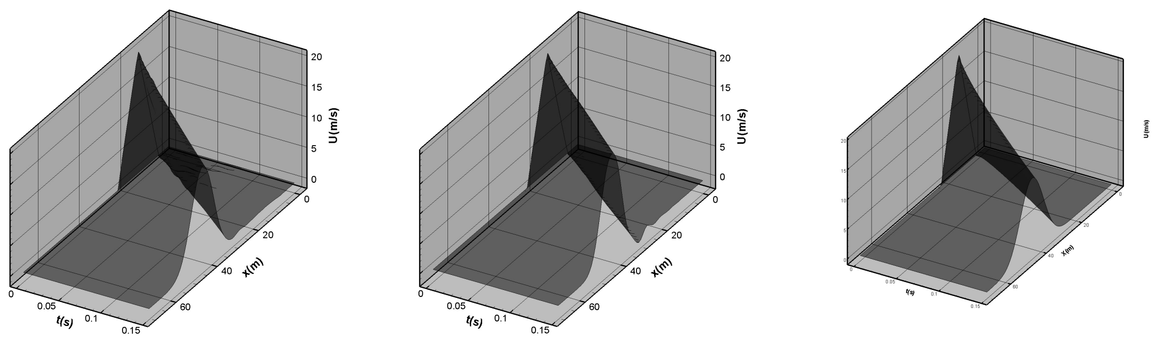

Fig. 7. Solution of first-order wave equation with c=0.25, from left to right: Euler's BTCS, Lax-Wendroff and first upwinding

6. APPENDIX

```
Module prop1
implicit none
save
integer, parameter :: imax=150
integer :: im
real(8) :: c
end module prop1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
program wave

!variables

use prop1
implicit none

integer, parameter :: nmax=150
integer :: opt,nm,i1,i2,i3,count,i, order, n, icase
real(8), dimension(imax) :: x,u,u1,u2,u3,ui
real(8), dimension(imax,nmax) :: ut
real(8) :: pi,int,dlength,delx,delt,a,tottime,w1,w2
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!input data
tottime=0.15d0
pi=dacos(-1.d0)
dlength=70.d0
delx=1.d0
delt=0.00125d0
a=200.d0
int=0.025d0

        print*, 'please␣inter␣a␣number'
        print*, 'First upwind differencing: 1'
        print*, 'Lax - Wendroff : 2'
        print*, 'Eulers BTCS : 3'
        read*, opt

if ( opt<1 .or. opt>3) then
                print*, "wrong number entered!"
                print*, " re-run the program."
                stop
end if


im=idint(dlength/delx)+1
nm=idint(tottime/delt)+1
c= A*delt/delx

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!initial and boundary condition
w1= 40.d0
w2=120.d0
i1=5
i2=15
i3=25

do i=1,im

x(i)=dble(i-1)*delx

        if (i>i1 .and. i<i2+1) then

                u(i)=2*(x(i)-5)

        else if (i>i2 .and. i<i3+1) then

                u(i)= -2*(x(i)-15) +20

        else
                u(i)=0.d0

        end if

ui(i)=u(i)

end do

count=0
```

```
do n=2,nm

        if(dabs (delt*dble(n-1)-int) <=1.d-7) then
                order=1
        elseif(dabs (delt*dble(n-1)-int*2.d0) <=1.d-7) then
                order=1
        elseif(dabs (delt*dble(n-1)-int*3.d0) <=1.d-7) then
                order=1
        elseif(dabs (delt*dble(n-1)-int*4.d0) <=1.d-7) then
                order=1
        elseif(dabs (delt*dble(n-1)-int*5.d0) <=1.d-7) then
                order=1
        elseif(dabs (delt*dble(n-1)-int*6.d0) <=1.d-7) then
                order=1
        else
                order=0
        end if


        select case (opt)
                case(1)
        call fud(u)
                case(2)
        call lwm(u)
                case(3)
        call eubtcs(u)
        end select


        if (order==1) then
                count=count+1

                do i=1,im

                if (count==1) then
                u1(i)=u(i)
                elseif (count==2) then
                u2(i)=u(i)
                elseif (count==3) then
                u3(i)=u(i)
                end if


                end do

        end if
        do i=1,im
        ut(i,n)= u(i)
        end do

end do

select case (opt)
case(1)
open( unit=9, file='fupwind.dat',status='replace', action='write')
open( unit=19, file='tpfupwind.dat',status='replace', action='write')
case(2)
open( unit=9, file='l_wm.dat',status='replace', action='write')
open( unit=19, file='tpl_wm.dat',status='replace', action='write')
case(3)
open( unit=9, file='eu_btcs.dat',status='replace', action='write')
open( unit=19, file='tpeu_btcs.dat',status='replace', action='write')

end select

write(9,*)
write(9,10)
write(9,*)
write(19,*) 'title="time marching" '
write(19,*) 'variable= "x", "t", "u" '
write(19,*) 'zone f=point, I=', im, 'j=', nm

do i=1,im
        write(9,20) x(i), ui(i), u1(i), u2(i), u3(i)

end do

do n=1, nm
                do i=1,im

                        write(19,30) x(i), delt*dble(n-1), ut(i,n)
                end do
end do

10 format (1x,'x', 3x, 't=0.0', 3x, 't=0.025', 3x, 't=0.05', 3x, &
                't=0.075', 3x, 't=0.1', 3x, 't=0.125', 3x, 't=0.15')

20 format (1x, f7.3, 3x, f9.4, 6(3x, f9.4))

30 format (1x, d15.8, 3x, d15.8, 3x, d15.8)

close(9)
close(19)

end program wave
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine fud(u)
use prop1
implicit none
integer :: i
real(8), dimension(imax) :: u,uold

do i=1,im
        uold(i)=u(i)
end do

do i=2,im-1
        u(i)= uold(i) - c*(uold(i)-uold(i-1))
end do

end subroutine fud
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine lwm(u)

use prop1
implicit none
integer :: i
real(8), dimension(imax) :: u, uold

do  i=1,im
        uold(i)=u(i)
end do

do i=2,im-1
        u(i)=uold(i)- c/2.d0*(uold(i+1)- uold(i-1)) &
                +c*c/2.d0*(uold(i+1)-2.d0*uold(i)+ uold(i-1))
end do
return
end subroutine lwm
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine eubtcs(u)

use prop1
implicit none
integer :: i
real(8), dimension(imax) :: u, aa, bb, cc, dd

do i=2,im-1
```

```
          aa(i)= 0.5d0*c
          bb(i)= -1.d0
          cc(i)= -0.5d0*c
          dd(i)= -u(i)
end do

call trid(aa,bb,cc,dd,u)
return
end subroutine eubtcs
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine trid(aa,bb,cc,dd,u)
use prop1
implicit none
integer :: i
real(8), dimension(imax) :: h,g,u,aa,bb,cc,dd

h(1)=0.d0
g(1)=u(1)

do i=2,im-1
          h(i)= cc(i)/ (bb(i)-aa(i)*h(i-1))
          g(i)= (dd(i)-aa(i)*g(i-1))/ (bb(i)-aa(i)*h(i-1))
end do

do i=im-1,2,-1
          u(i)=-h(i)*u(i+1)+g(i)
end do
return
end subroutine trid
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```