A report on solution of non-linear - Burgers Equation using upwind, Lax-Wendroff and Second order TVD

Arjun Darbha

Contents

d

1 Introduction

Hyperbolic equations which cannot be solved analytically and that are inherently time-marching can be solved numerically. Moreover, the solution that exists in case of non-linear hyperbolic equations is also non-linear. There are two ways of dealing with such equations. The non-linear analytical solution can be solved numerically using Newton-Raphson, bisection or any such iterative and numerical solving techniques. This task was successfully accomplished in the first programming and a solution obtained without really understanding the underlying physics of the non-linear equation. At this point, with the fundamental understanding of hyperbolic equations, an attempt is made to solve the non-linear Burgers equation using three numerical schemes

1. First order Upwind
2. Lax Wendroff Scheme (Second order)
3. Total Variation Diminishing (TVD) scheme (Second order)

Restating Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 \qquad \text{Eq.1}$$

This equation is hyperbolic, which means that a set of initial conditions are imperative for solving this equation. Also, it can be deduced from Eq.1 that a time marching solution exists for this type of equation. The problem statement provides the initial conditions and the space-time domain as,

$$u(x,0) = f(x) \qquad \text{Eq.2}$$

$$f(x) = u_o + Asinx \qquad \text{Eq.3}$$

for this particular problem, it has been given that constants u_o = A = 1. Also the domain for which Eq.3 is valid is from 0 to 2π above and below which the value of the function is 1.

The objective of this report is to present a solution for Eq.1 using the three schemes mentioned above and compare these solutions with the analytical solution determined in the first programming homework.

2 First Order Upwind

As the name suggests this is a first order scheme which is first order in time and first order in space. Numerically, this is scheme is forward time and backward space. Eq.1 must be simplified algebraically so that it can be discretized using this scheme. On simplification, Eq.1 can be written in the following form

$$\frac{\partial u}{\partial t} + \frac{\partial\left(\frac{u^2}{2}\right)}{\partial x} = 0 \qquad \text{Eq.4}$$

The second term in Eq.4, which is the flux term, is the term that is being advected in this equation. For the rest of the report, this flux term is represented by E which is $u^2/2$. The equation that is dealt with, for discretization in the entire report report is

$$\frac{\partial u}{\partial t} + \frac{\partial(E)}{\partial x} = 0 \qquad \text{Eq.5}$$

Using FTBS on Eq.5

$$\frac{(u_i^{n+1} + u_i^n)}{\Delta t} + \frac{E_i^n - E_{i-1}^n}{\Delta x} = 0 \qquad \text{Eq.6}$$

Eq.6 can be rearranged into

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{\Delta x}(E_i^n - E_{i-1}^n) \qquad \text{Eq.7}$$

This equation can be solved explicitly to obtain the value of *u* as Right Hand Side of Eq.7 is known at each time step.

If linear wave equation is considered, it is known that accurate solution occurs at Courant number(C), 1.0. This is due to the fact that, at C = 1.0, the numerical solution is exactly similar

to analytical solution and it best captures the physics of the problem. The same idea can be used to obtain the solution of Eq.7. Hence, it has to be realized that the physics is best preserved in the higher courant numbers (C). Also, is can be seen that Courant number is not a constant in non-linear case. Hence, solution formulation should not deviate from the basic idea of solving a linear wave equation and necessary modifications should be made to capture the physics effectively.

2.1 Algorithm

- Declare discretized spatial domain (x values with a predetermined dx)
- Declare the initial conditions that is system configuration at t=0
- The same code will be run for three different Courant numbers. Hence set a Courant no. for the specific run
- Now calculate the time difference Δt (dt in the code), using the formula

$$dt = \frac{Cdx}{\max(u)}$$

- Discretize the time domain using dt obtained in the previous step
- Start spatial and temporal loops
- Access each element and calculate velocity u_i^{n+1} using Equation 7
- If maximum of time is reached, stop loop, end program

2.2 Analysis of results for different courant numbers (C)

Plots below represent the difference between analytical and actual solution for different Courant numbers. The deviation of analytical solution from the upwind solution is clearly represented in the figures shown below. Note that in the legend, tu denotes upwind scheme time step and ta denotes analytical time step solution

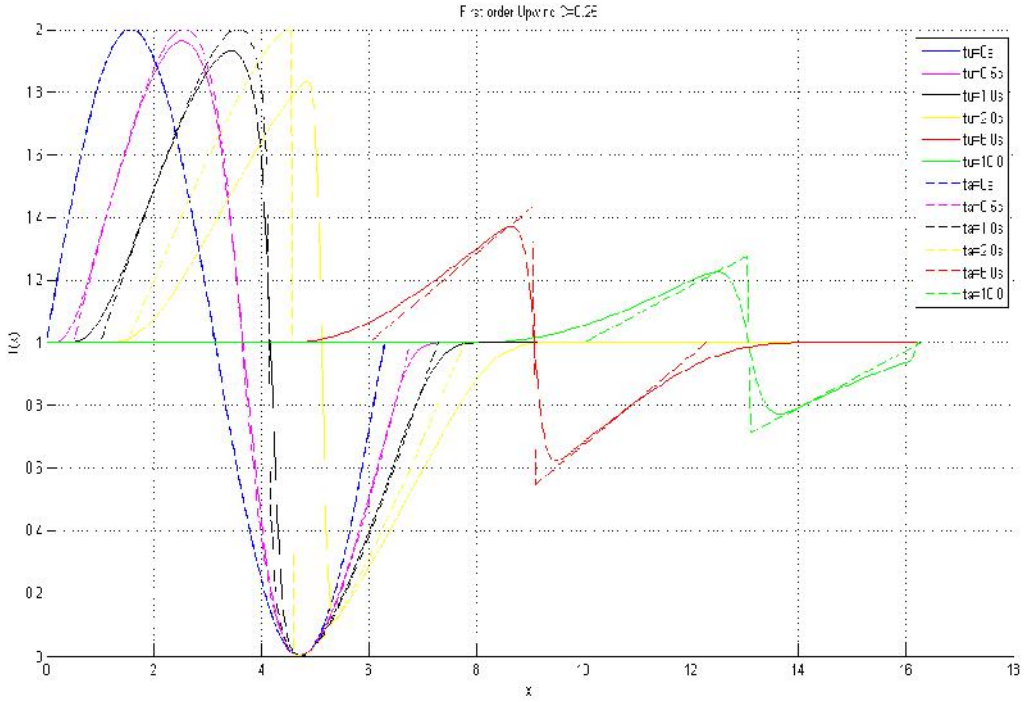

Fig.1 Analytical and upwind comparison C = 0.25

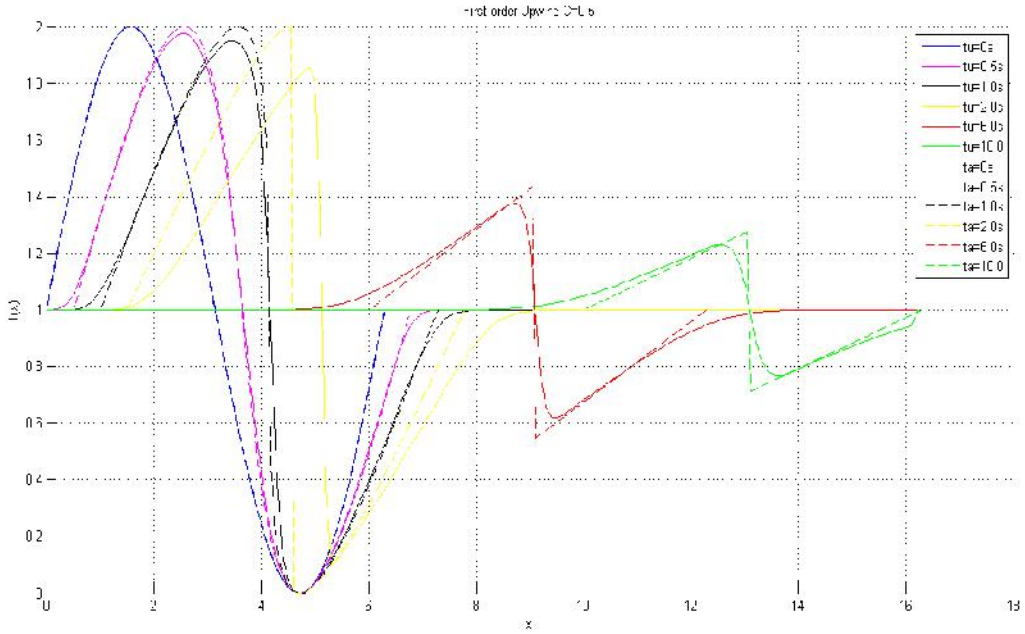

Fig.2 Analytical and upwind comparison C = 0.50

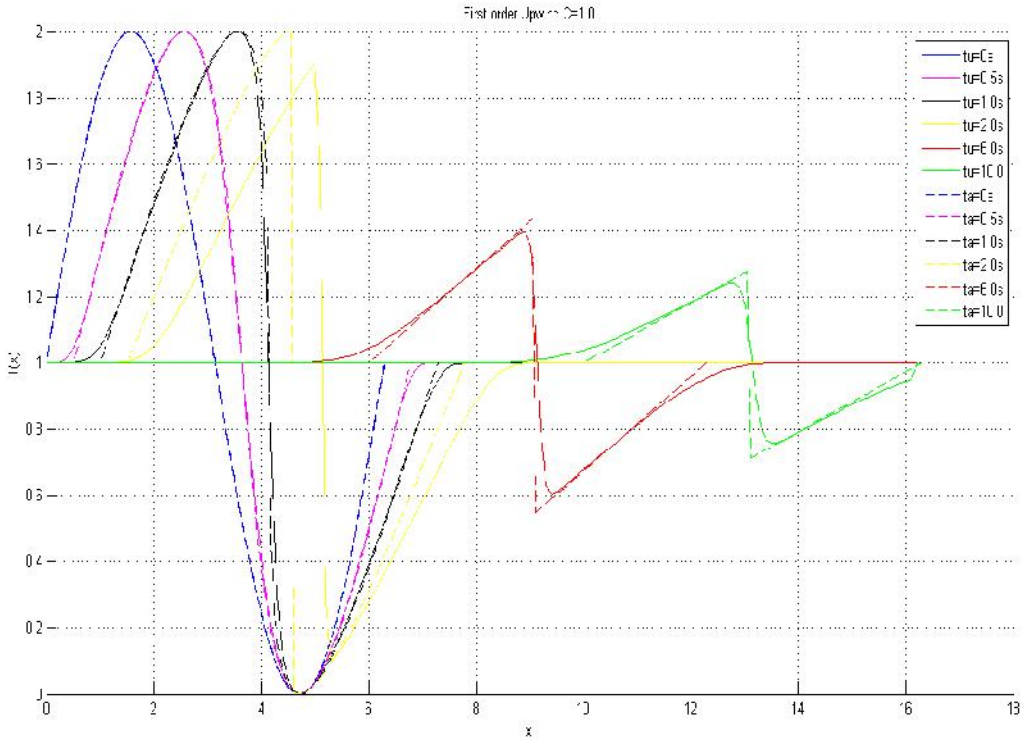

Fig.3 Analytical and upwind comparison C = 1.0

Conclusions

- It can be seen from the plots that the discontinuity is smeared over space as the time step increases. This type of error is called a dissipative error and it is inherent in first order schemes
- As already discussed, it is desirable to have a courant number closer to 1 to capture physics perfectly, this is clearly shown in Fig. 1, 2 and 3
- Hence there is a need for a better solution, which means that the error term has to be reduced. At this point, one of the solutions is to look for a solution in higher order schemes which is the next topic

3 Lax Wendroff Scheme

This is a second order method where the finite difference formulation is obtained directly from the Taylor Series. Consider

$$u_i^{n+1} = u_i^n + \Delta t \frac{\partial u}{\partial t} + \frac{(\Delta t)^2}{2!}\left(\frac{d^2u}{dt^2}\right) + \cdots \quad \text{Eq.8}$$

From Eq.5 which is the conservative form of Burgers equation the following can be deduced

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{2\Delta x}(E_{i+1}^n - E_{i-1}^n) + \frac{(\Delta t)^2}{4\Delta x^2}[(u_{i+1}^n + u_i^n)(E_{i+1}^n - E_i^n) - (u_i^n + u_{i+1}^n)(E^n - E_{i-1}^n)]$$

Eq.9

As it has already been mentioned that Eq.9 is deduced from Eq.8 which is a second order approximation, Lax Wendroff is essentially a second order scheme.

3.1 Algorithm

- Declare discretized spatial domain (x values with a predetermined dx)
- Declare the initial conditions that is system configuration at t=0
- The same code will be run for three different Courant numbers. Hence set a Courant no. for the specific run
- Now calculate the time difference Δt (dt in the code), using the formula

$$dt = \frac{Cdx}{\max(u)}$$

- Discretize the time domain using dt obtained in the previous step
- Start spatial and temporal loops
- Access each element and calculate velocity u_i^{n+1} using Equation 9
- If maximum of time is reached, stop loop, end program

Eq.9 is discretized and the results are shown in the next section

3.2 Analysis of results for different courant numbers (C)

With the basic idea of the Algorithm presented above, Lax Wendroff numerical scheme is programmed and the following results have been obtained. It has to noted that tu denotes the numerical time step and ta denotes the analytical time step

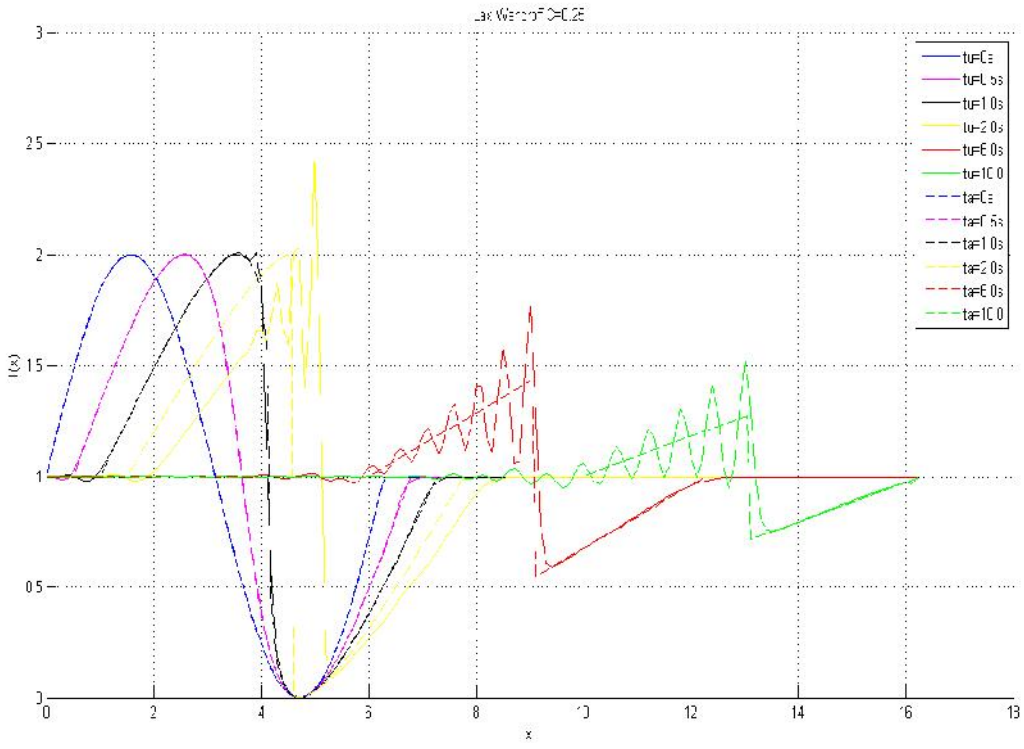

Fig.4 Analytical and Lax-Wendroff comparison C = 0.25

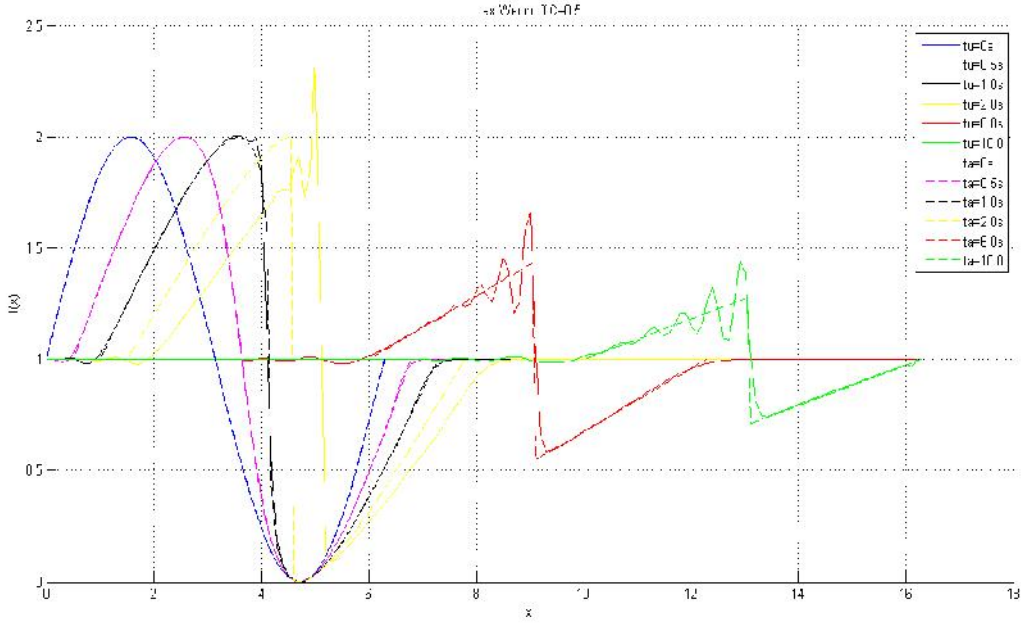

Fig.5 Analytical and Lax-Wendroff comparison C = 0.5

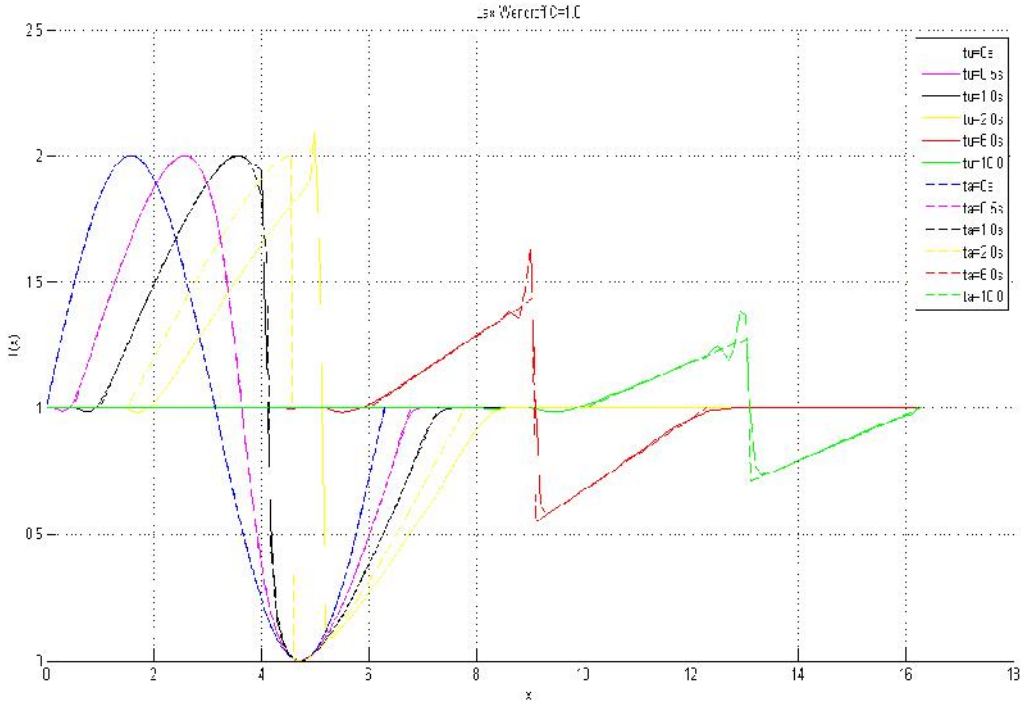

Fig.6 Analytical and Lax-Wendroff comparison C = 1.0

3.3 Conclusions

- From these graphs, it can be concluded that, although there is no dissipation error in case of Lax Wendroff, there is a lot of disturbance around the discontinuity
- These disturbances or oscillations are characteristic of a second order numerical scheme and this error is called a dispersion error
- Also, as it can be expected, the dissipation error reduced as Courant number increases. This can be clearly seen in Fig 4,5 and 6 which possess a decreasing magnitude of dissipation error. Oscillating disturbances are minimum in Fig. 6 where C=1.0

4 Total Variation Diminishing Scheme (Second-order)

TVD scheme is a generic scheme from which all the other schemes have been derived. A scheme is said to be Total Variation Diminishing if

$$TV(u^n) = \sum_{-\infty}^{\infty} |u_{i+1}^n - u_i^n| \qquad \text{Eq.10}$$

$$TV\left(u^{(n+1)}\right) \leq TV(u^n) \qquad \text{Eq.11}$$

TVD's can be classified into First-order and second order schemes. First order upwind and Lax Wendroff can be deduced from first order TVDs. However, here, a Burgers equation is solved using a second order TVD in order to minimize dissipation and dispersion error encountered in the previous attempts.

To start with a second order TVD scheme modifies the original flux term into

$$\bar{E} = E + G \qquad \text{Eq.12}$$

where G is called the flux limiter function. There have been many limiters proposed. For the purpose of solving this equation, Haten-Yee upwind TVD limiter function is used.

The governing equation that is solved is very similar to Eq.7

$$u_i^{n+1} = u_i^n + \frac{\Delta t}{\Delta x}\left(h_{i+\frac{1}{2}}^n - h_{i-\frac{1}{2}}^n\right) \qquad \text{Eq.13}$$

If the above equation is dealt with mathematically,

$$h^n_{i+\frac{1}{2}} = 1/2\left(E^n_i + E^n_{i+1} + \phi^n_{i+\frac{1}{2}}\right) \quad \text{Eq.14}$$

$$h^n_{i-\frac{1}{2}} = 1/2(E^n_i + E^n_{i-1} + \phi^n_{i-\frac{1}{2}}) \quad \text{Eq.15}$$

In Eq.14 and Eq.15

$$\phi_{i+\frac{1}{2}} = \sigma\left(\alpha_{i+\frac{1}{2}}\right)[G_{i+1} + G_i] - \psi\left(\alpha_{i+\frac{1}{2}} + \beta_{i+\frac{1}{2}}\right)\Delta u^n_{i+\frac{1}{2}} \quad \text{Eq.16}$$

$$\phi_{i-\frac{1}{2}} = \sigma(\alpha_{i-\frac{1}{2}})[G_i + G_{i-1}] - \psi\left(\alpha_{i-\frac{1}{2}} + \beta_{i-\frac{1}{2}}\right)\Delta u^n_{i-\frac{1}{2}} \quad \text{Eq.17}$$

In Eq.16 and Eq.17

$$\sigma\left(\alpha_{i+\frac{1}{2}}\right) = \frac{1}{2}\psi\left(\alpha_{i+\frac{1}{2}}\right) + \frac{\Delta t}{\Delta x}\left(\alpha_{i+\frac{1}{2}}\right)^2 \quad \text{Eq.18}$$

$$G = minmod(\Delta u_{i+\frac{1}{2}}, \Delta u_{i-\frac{1}{2}}) \quad \text{Eq.19}$$

$$\alpha_{i+\frac{1}{2}} = (E_{i+1} - E_i)/\Delta u_{i+\frac{1}{2}} \quad \text{Eq.20}$$

$$\beta_{i+\frac{1}{2}} = (G_{i+1} - G_i)/\Delta u_{i+\frac{1}{2}} \quad \text{Eq.21}$$

Equations 14 to 21 can be used to solve Eq. 13 explicitly.

4.1 Algorithm

- Declare discretized spatial domain (x values with a predetermined dx)
- Declare the initial conditions that is system configuration at t=0
- The same code will be run for three different Courant numbers. Hence set a Courant no. for the specific run
- Now calculate the time difference Δt (dt in the code), using the formula

$$dt = \frac{Cdx}{\max(u)}$$

- Discretize the time domain using dt obtained in the previous step
- Start spatial and temporal loops
- Access each element and calculate velocity u_i^{n+1} using Equation 13
- If maximum of time is reached, stop loop, end program

4.2 Analysis of results for different courant numbers (C)

The attempt here is to obtain a solution without dissipation or dispersion. This scheme is as close as we can get to the analytical solution as it can be seen from the figures presented below.

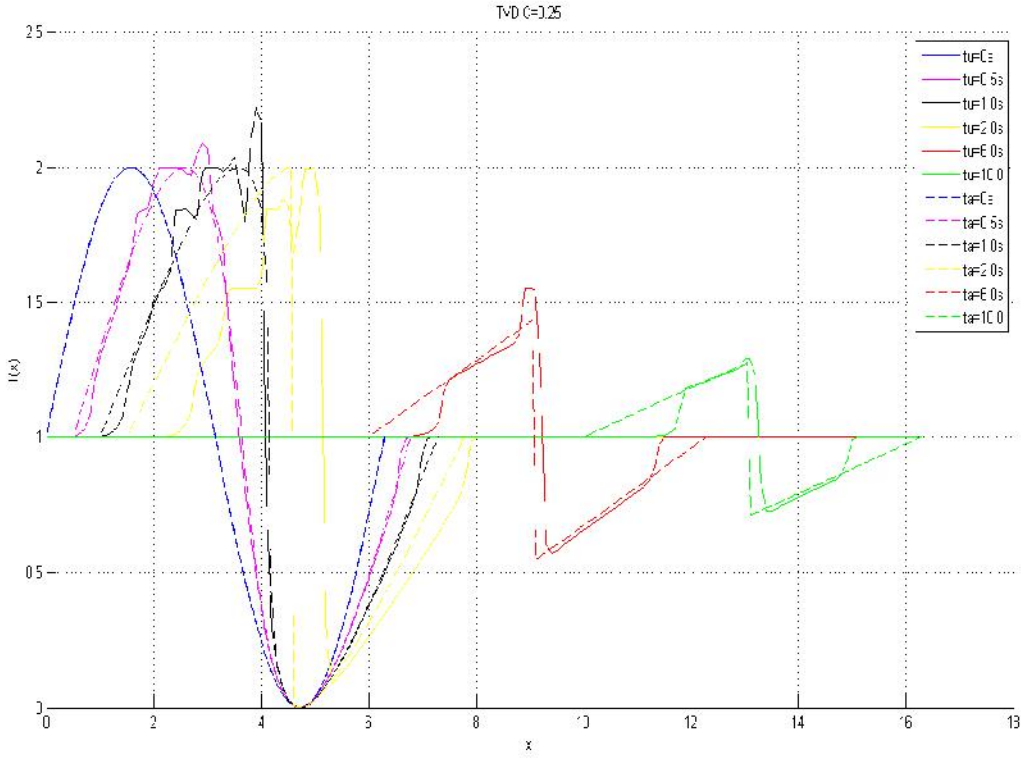

Fig.7 Analytical and TVD comparison C = 0.25

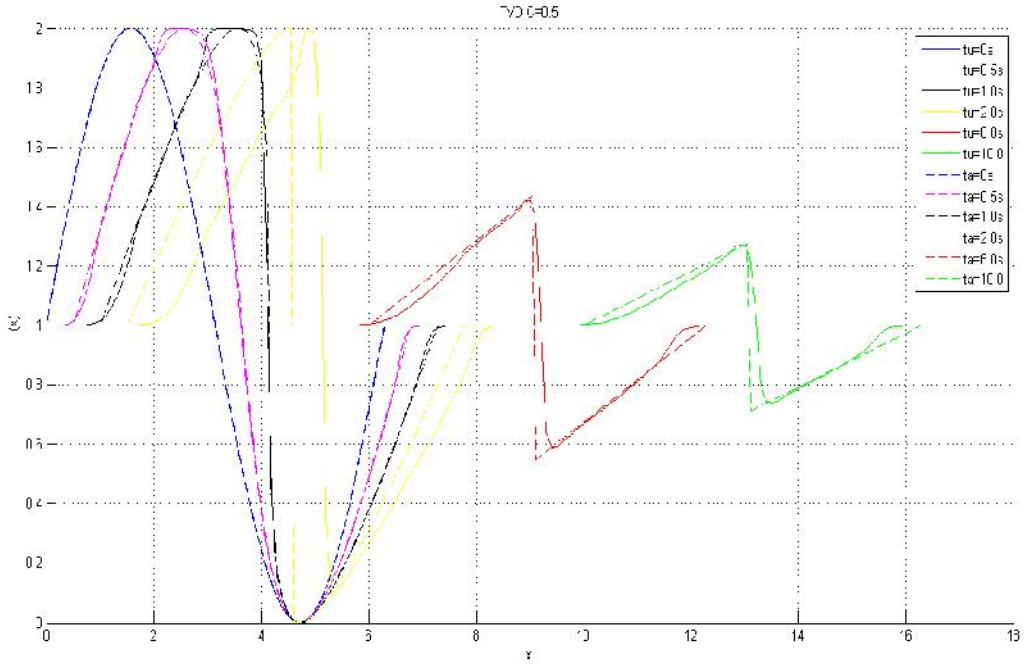

Fig.8 Analytical and TVD comparison C = 0.5

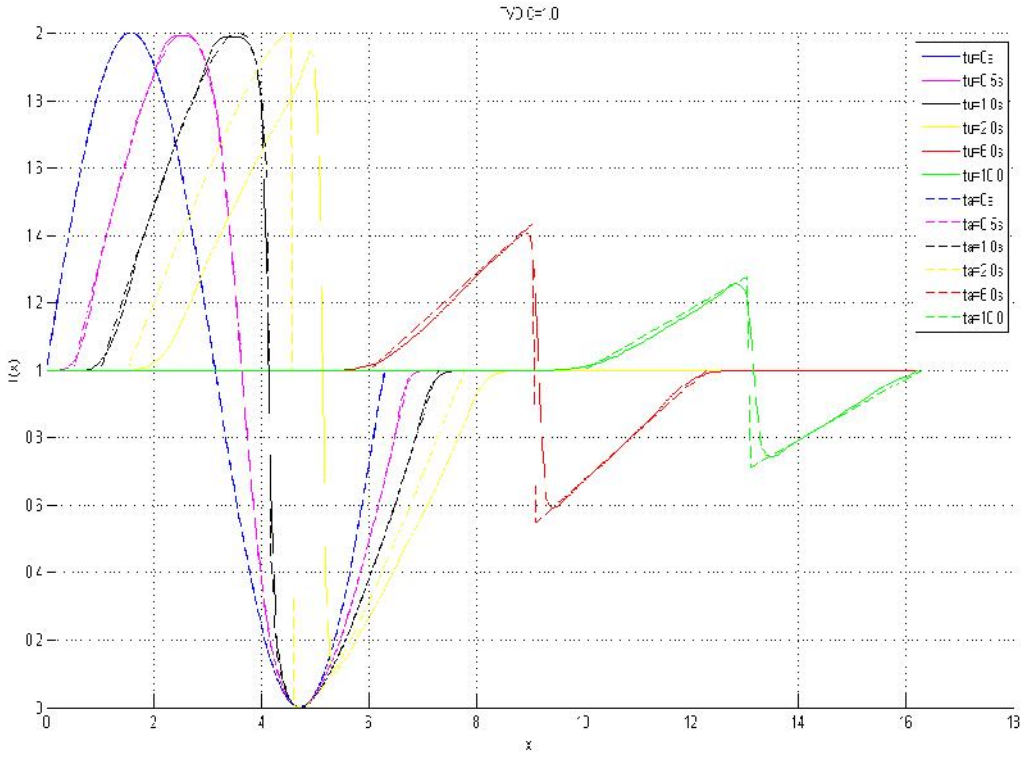

Fig.9 Analytical and TVD comparison C = 1.0

4.3 Conclusions

- It can be clearly seen from the plots that the error is much lower in this case when compared to other numerical schemes
- As with the other schemes, the error is decreasing with increase in Courant number
- This scheme is computationally more intensive when compared to the other two schemes and hence it might be computationally expensive to carry out this scheme on applications where swiftness matters
- This scheme however, is the best scheme, in terms of accuracy when compared to other schemes presented in this report

5 References

- Computational Fluid Dynamics, Vol 1, Fourth Edition, Klaus A. Hoffman, Steve T. Chiang, 2000
- Computational Fluid Dynamics Basics with Applications, John D. Anderson, 1995

6 Appendix

6.1 First order upwind code and Lax Wendroff

Note that the logic behind these schemes is similar except the gamma equation which can be switched to either of these by commenting/uncommenting the appropriate equation.

```
%Declaring constants for initial condition
dx = 0.2;
x = 0:dx:2*pi;
C = 0.5; %max - Courant number

% t = 0:dt:10;
u0 = 1;
A = 1;
len = length(x)+100;
%courant(1) = 0;
%Calculating temporal loop parameters
dt = C*dx/2;
t = 0:dt:10;
```

```
u = ones(length(t),len);

u(1,1:length(x)) = u0 + A.*sin(x);


for n = 2:length(t)
  for i = 2:(len-1)

        %LAX-WENDROFF
        u(n,i)=u(n-1,i)-((dt)/(4*dx))*(u(n-1,i+1)^(2)-u(n-1,i-
1)^(2))+(dt^(2))/(4*dx^(2))*((u(n-1,i+1)+u(n-1,i))*0.5*(u(n-1,i+1)^(2)-u(n-
1,i)^(2))-((u(n-1,i)+u(n-1,i-1))*0.5*(u(n-1,i)^(2)-u(n-1,i-1)^(2))));
        %UPWIND
%         u(n,i) = u(n-1,i) - (dt/dx)*((E(u(n-1,i)))-(E(u(n-1,i-1))));

    end
end
```

6.2 TVD second order

6.2.1 Main code

```
dx = 0.1;
x = 0:dx:2*pi;
C = 0.6;
dt = C*dx/2;
t = 0:dt:10;
u0 = 1;
A = 1;
len = length(x)+100;

u = ones(length(t),len);
u(1,1:length(x)) = u0 + A.*sin(x);

for n = 2:(length(t))
    %for i = 2:(length(x)-1)
    for i = 2:(len-1)

        duplus = u(n-1,i+1) - u(n-1,i);
        duminus = u(n-1,i) - u(n-1,i-1);
        if i == len-1
            duplus3ov2 = 0;
        else
            duplus3ov2 = u(n-1,i+1) - u(n-1,i+2);
        end
        if i == 2;
            duminus3ov2 = 0;
        else
            duminus3ov2 = u(n-1,i-1) - u(n-1,i-2);
        end

        %G's
```

```
        gi1 = minmod(duplus3ov2,duplus);
        gi = minmod(duplus,duminus);
        giminus1 = minmod(duminus,duminus3ov2);

        %alphas
        if duplus ~= 0
            alphaplus = (E(u(n-1,i+1)) - E(u(n-1,i)))/duplus;
        else
            alphaplus = 0.5*(u(n-1,i+1) + u(n-1,i));
        end

        if duminus ~=0
            alphaminus = (E(u(n-1,i)) - E(u(n-1,i-1)))/duminus;
        else
            alphaminus = 0.5*(u(n-1,i) + u(n-1,i-1));
        end

        %betas
        if duplus == 0
            betaplus = 0;
        else
            betaplus = sigma(alphaplus,dt,dx)*(gi1-gi)/duplus;
        end

        if duminus == 0
            betaminus = 0;
        else
            betaminus = sigma(alphaminus,dt,dx)*(gi-giminus1)/duminus;
        end


        %phis
        phiplus = sigma(alphaplus,dt,dx)*(gi1 + gi) - psi((alphaplus +
betaplus),0)*(duplus);
        phiminus = sigma(alphaminus,dt,dx)*(gi + giminus1) - psi((alphaminus
+ betaminus),0)*(duminus);

        %h's
        hhalfplus = 0.5*(E(u(n-1,i+1)) + E(u(n-1,i)) + phiplus);
        hhalfminus = 0.5*(E(u(n-1,i)) + E(u(n-1,i-1)) + phiminus);

        u(n,i) = u(n-1,i) - (dt/dx)*(hhalfplus - hhalfminus);
    end
end
```

6.2.2 minmod function

```
function z = minmod(x,y)

if (abs(x) < abs(y)) && x*y > 0
    z = x;
elseif (abs(x) > abs(y)) && x*y > 0
    z = y;
```

```
else
    z = 0;
end

end
```

6.2.3 psi function

```
function z = psi(y,e)

if  abs(y) >= e
    z = abs(y);
else
    z = (y^2 + e^2)/(2*e);
end

end
```

6.2.4 sigma function

```
function z = sigma(alpha,dt,dx)

z = 0.5*(psi(alpha,0.1)) + (dt/dx)*(alpha^2);

end
```

6.2.5 E function(flux)

```
function [e] = E(u)

e = 0.5*(u^2);

end
```

END