Linear Wave Equation and 1D Shock Tube Problem solved using many different schemes

Research · November 2015

DOI: 10.13140/RG.2.1.3118.0248

1 author:

Rajeev KUMAR
Texas A&M University - Corpus Christi
37 PUBLICATIONS **75** CITATIONS

SEE PROFILE

Solution of Linear Wave Equation using four diff schemes and 1D Shock Tube Problem using two different schemes

by

RAJEEV KUMAR

PROBLEM DEFINITION

Following are the given 1D problems

(i) Linear wave equation $u_t + \frac{1}{2}u_x = 0$ with the following definitions

(a) $u(x,0) = \sin \pi x, \qquad t \le 2, -2 \le x \le 2$

(b) $u(x,0) = \begin{cases} 1.0 & if \quad -2 \le x \le 0 \\ 0.0 & otherwise \end{cases} \qquad t \le 2, -2 \le x \le 2$

(ii) One dimensional Shock-tube problem $\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = 0, \quad t \le 2, -5 \le x \le 5$

where $U = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix}$ and $F = \begin{pmatrix} \rho \\ \rho u + p \\ (E+p)u \end{pmatrix}$ with the initial condition

$$\begin{pmatrix} \rho \\ u \\ p \end{pmatrix} = \begin{cases} 1.0 \\ 0 \\ \frac{1}{\gamma - 1} \end{cases} \quad x < 0;$$

$$\begin{pmatrix} \rho \\ u \\ p \end{pmatrix} = \begin{cases} 0.125 \\ 0 \\ \frac{0.125}{\gamma - 1} \end{cases} \quad x \ge 0.$$

SCHEMES EMPLOYED

(i) One dimensional wave equation:

One dimensional wave equation was solved using following schemes

1). Lax Method
2). Lax-Wendroff Method
3). MacCormack Method
4). Fourth order Runge-Kutta Method

(ii) One dimensional shock-tube problem

One dimensional shock-tube problem was solved using flux splitting schemes

1). Steger-Warming Splitting Method
2). Van Leer Flux Splitting Method

One Dimensional Wave Equation

1). Lax Method:

In the Euler explicit method for discretizing linear wave equation $\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x}=0 \quad c>0$ if we replace u_j^n with the average term $\frac{u_{j+1}^n+u_{j-1}^n}{2}$ we obtain the popular Lax method as:

$$\frac{u_j^{n+1}-(u_{j+1}^n+u_{j-1}^n)/2}{\Delta t}+c\frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x}=0$$

This explicit one step scheme is first-order accurate with truncation error of $o[\Delta t,(\Delta x)^2/\Delta t]$ and is stable if Courant number $|\nu|\le 1$. Its modified equation is given as

$$u_t+cu_x=\frac{c\Delta x}{2}\left(\frac{1}{\nu}-\nu\right)u_{xx}+\frac{c(\Delta x)^2}{3}(1-\nu^2)u_{xxx}+.....$$

As the leading term of the RHS is even order derivative of u, Lax method is known for its large dissipative error. It also has leading phase error.

2). Lax-Wendroff Method:

Lax-Wendroff finite difference scheme can be derived from Taylor series expansion as:

$$u_j^{n+1}=u_j^n+\Delta t u_t+\tfrac{1}{2}(\Delta t)^2u_{tt}+o[(\Delta t)^3]$$

and by using the wave equation $u_t=-cu_x \quad and \quad u_{tt}=c^2u_{xx}$

and by substituting $u_x \; and \; u_{xx}$ are replaced by central difference expressions we get the well-known Lax-Wendroff scheme with stability condition $|\nu|\le 1$ as

$$u_j^{n+1}=u_j^n-\frac{c\Delta t}{2\Delta x}(u_{j+1}^n-u_{j-1}^n)+\frac{c^2(\Delta t)^2}{2(\Delta x)^2}(u_{j+1}^n-2u_j^n+u_{j-1}^n)$$

Lax-Wendroff scheme is an explicit one step scheme that is second order accurate with a modified equation as

$$u_t+cu_x=-c\frac{(\Delta x)^2}{6}(1-\nu^2)u_{xxx}-\frac{c(\Delta x)^3}{8}\nu(1-\nu^2)u_{xxxx}+.....$$

As can be seen it has an odd order leading term there for it is very dispersive in nature with pretty large lagging phase error.

3). MacCormack Method:

The MacCormack method is a widely use method in fluid dynamics. It is particularly useful in solving non-linear PDEs. When applied to a linear wave equation, this explicit predictor-corrector method becomes

$$\overline{u_j^{n+1}} = u_j^n - c\frac{\Delta t}{\Delta x}\left(u_{j+1}^n - u_j^n\right) \qquad \text{Predictor}$$

$$u_j^{n+1} = \frac{1}{2}\left[u_j^n + \overline{u_j^{n+1}} - c\frac{\Delta t}{\Delta x}\left(\overline{u_j^{n+1}} - \overline{u_{j-1}^{n+1}}\right)\right] \qquad \text{Corrector}$$

Where $\overline{u_j^{n+1}}$ is the temporary predicted value of u at time level n+1. Since for linear equations MacCormack method tends to be similar to Lax-Wendroff method, its truncation error, stability limit, modified equation, amplification factor are identical to Lax-Wendroff scheme.

4). Fourth Order Runge-Kutta Method:

In Runge-Kutta method first step is to form a "pseudo-ODE" by separating the partial derivative. For example for the linear case

$$u_t = R(u)$$

Where $R(u) = -cu_x$. This "pseudo-ODE" is now a time-continuous equation and any integration scheme is applicable. Fourth order Runge-Kutta method is as following

$$u^{(1)} = u^n + \frac{\Delta t}{2}R^n \qquad \text{step1}$$

$$u^{(2)} = u^n + \frac{\Delta t}{2}R^{(1)} \qquad \text{step2}$$

$$u^{(3)} = u^n + \Delta t R^{(2)} \qquad \text{step3}$$

$$u^{n+1} = u^n + \frac{\Delta t}{6}\left(R^n + 2R^{(1)} + 2R^{(2)} + R^{(3)}\right) \qquad \text{step4}$$

If second order accurate spatial differences are inserted into this, the resulting scheme will have a T. E of $o\left[(\Delta t)^4, (\Delta x)^2\right]$

The given linear wave equation with two definitions of sine wave and discontinuity was solved using the above four schemes for Courant numbers ν = 0.25, 0.5, 0.75 and 1.0 and the comparative results are displayed in Figs 1 and 2 respectively for sine wave and discontinuity. The C-programs are attached in the Appendix. The provision for both discontinuity and sine wave are there in single program.

One Dimensional Shock-Tube Problem

Two flux splitting schemes were used here as following

1). Steger-Warming Splitting Method:

This method was developed by Steger and Warming in 1979. In splitting the flux term the flux is assumed to be composed of a positive and a negative component. Consider our 1D shock-tube problem

$$\frac{\partial U}{\partial t}+\frac{\partial F}{\partial x}=0$$

The system can be written in the form

$$\frac{\partial U}{\partial t}+[A]\frac{\partial U}{\partial x}=0$$

Where [A] is the Jacobian $\partial F/\partial U$.

Flux F can be written as

$$F=[A]U=[T][\lambda][T]^{-1}U$$

The matrix of Eigen values is divided into two matrices, one with positive elements and other with negative elements. We write the [A] matrix as

$$[A]=[A^{+}]+[A^{-}]=[T][\lambda^{+}][T]^{-1}+[T][\lambda^{-}][T]^{-1}$$

The associated split flux component $F^{\pm}=[A]^{\pm}U$ is

$$F^{\pm}=\frac{\rho}{2\gamma}\begin{bmatrix}\lambda_1^{\pm}+2(\gamma-1)\lambda_2^{\pm}+\lambda_3^{\pm}\\(u-a)\lambda_1^{\pm}+2(\gamma-1)u\lambda_2^{\pm}+(u+a)\lambda_3^{\pm}\\(H-ua)\lambda_1^{\pm}+(\gamma-1)u^2\lambda_2^{\pm}+(H+ua)\lambda_3^{\pm}\end{bmatrix}$$

Where $\lambda_1=u$, $\lambda_2=u+a$ $\lambda_2=u-a$

And $\lambda_j^{+}=\frac{\lambda_j+|\lambda_j|}{2}$ and $\lambda_j^{-}=\frac{\lambda_j-|\lambda_j|}{2}$

And final algorithm is written as

$$U_i^{n+1}=U_i^{n}-\frac{\Delta t}{\Delta x}\left[(F^{+}+F^{-})_{i+\frac{1}{2}}-(F^{+}+F^{-})_{i-\frac{1}{2}}\right]$$

Where $F_{i+\frac{1}{2}}^{+}$ is set equal to F_i^{+}, and $F_{i+\frac{1}{2}}^{-}$ is set equal to F_{i+1}^{-}

The C-program for shock tube problem is attached in the Appendix.

2). Van Leer Flux Splitting Method:

Van Leer constructed a splitting for the Euler equation such that split Jacobian matrices are given as

$$\hat{A}^{+}=\frac{\partial F^{+}}{\partial U},\quad \hat{A}^{-}=\frac{\partial F^{-}}{\partial U}$$ and they are continuous

The split fluxes are given as

$$F^{+}=\frac{1}{4}\rho a(1+M^{2})\begin{bmatrix} 1 \\ \frac{2a}{\gamma}\left(\frac{\gamma-1}{2}M+1\right) \\ \frac{2a^{2}}{\gamma^{2}-1}\left(\frac{\gamma-1}{2}M+1\right)^{2} \end{bmatrix},$$

$$F^{-}=-\frac{1}{4}\rho a(1-M^{2})\begin{bmatrix} 1 \\ \frac{2a}{\gamma}\left(\frac{\gamma-1}{2}M-1\right) \\ \frac{2a^{2}}{\gamma^{2}-1}\left(\frac{\gamma-1}{2}M-1\right)^{2} \end{bmatrix}$$

The C-program for shock tube problem is attached in the Appendix. It contains both Steger Warming subroutine as well as Van Leer subroutine. In the program the time step is calculated using relation $\Delta t=\frac{C_{cfl}\Delta x}{s^{n}_{\max}}$ where $S^{n}_{\max}$ is the largest wave speed present in the domain. CFL coefficient $C_{cfl}\approx 0.636$ given by Van Leer fro practical stability was used.

Results and Discussion

Results for 1D wave equations are displayed in Figs. 1 and 2. and those for Shock-tube problem are displayed in Fig. 3 and 4. In Fig. 1 we have results for Sine wave propagated for t = 2. Four different Courant numbers i.e. ν = 0.25,0.5,0.75 and 1.0 were tried. Results in Fig.1 show that at smaller $\nu \approx 0.25$ and 0.5 the Lax scheme is too dissipative and Lax-Wendroff is too dispersive as expected. Whereas at higher Courant numbers these are less so and compare well with Runge-Kutta and MacCormacK schemes.

Fig 2. shows the results for the propagation of the step discontinuity. Here as expected Lax scheme is very dissipative and Lax-Wendroff, MacCormack and RK4 are very dispersive at ν = 0.25. As the Courant number increases they become less so with the exception of Runge-Kutta which is excessively dispersive for all the ν tried. At ν = 1.0, Lax, Lax-Wendroff and MacCormack are as good as exact solution. **(I could not double check the highly oscillatory results from Runge-Kutta Scheme applied to discontinuity)**

In Figs 3 and 4 are displayed the results from shock-tube problem. Fig 3 has the density, and the velocity plots for Steger scheme and Van Leer scheme put together with the exact solution which was obtained by solving the Riemann problem using the Algorithm listed in a book by Laney. The result match the exact solution well. Van Leer results seem to be better than that of Steger.

Fig. 4 has the plots for pressure and energy comparison. Here too they match with the exact solution very well and again Van Leer solution seems to be much sharper comparatively.

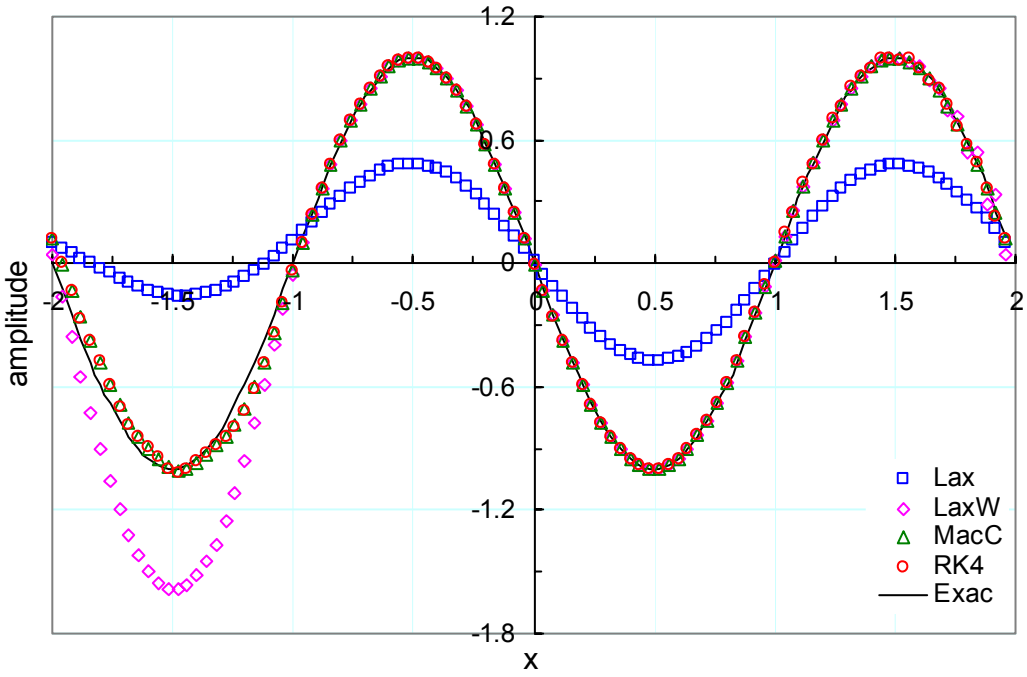

Fig.1a Comparative Plots for Courant No., ν = 0.25

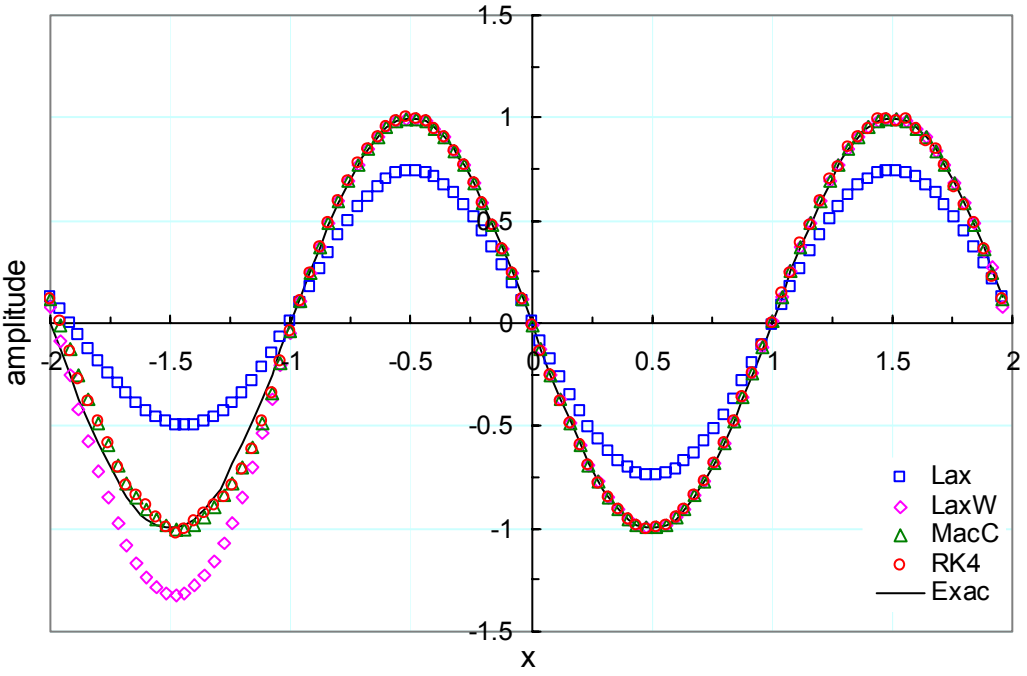

Fig.1b Comparative Plots for Courant No., ν = 0.5

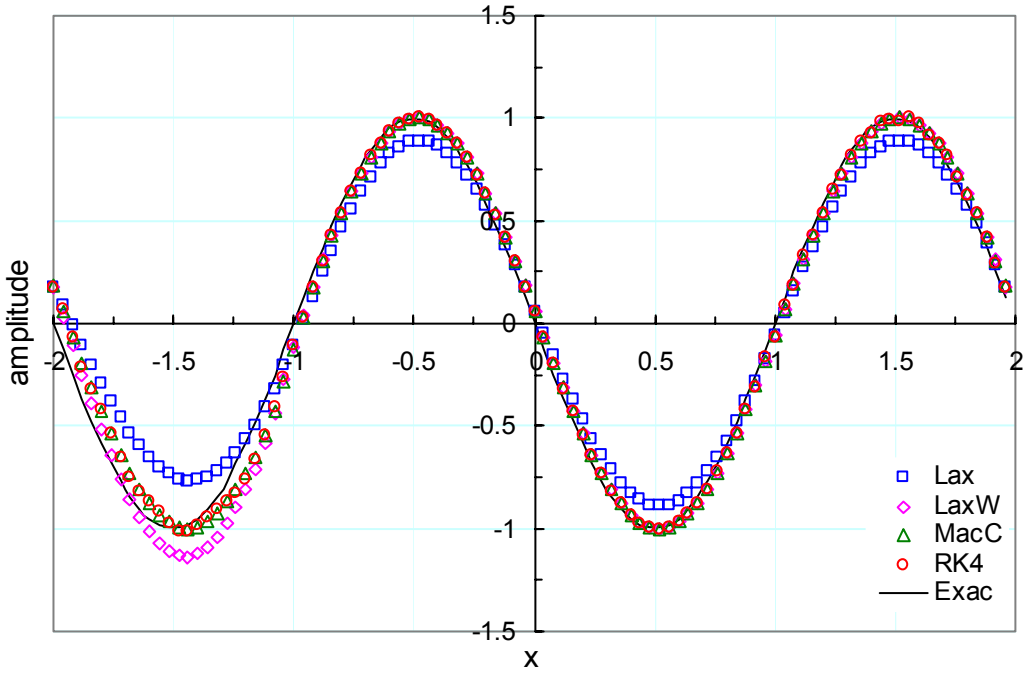

Fig.1c Comparative Plots for Courant No., ν = 0.75

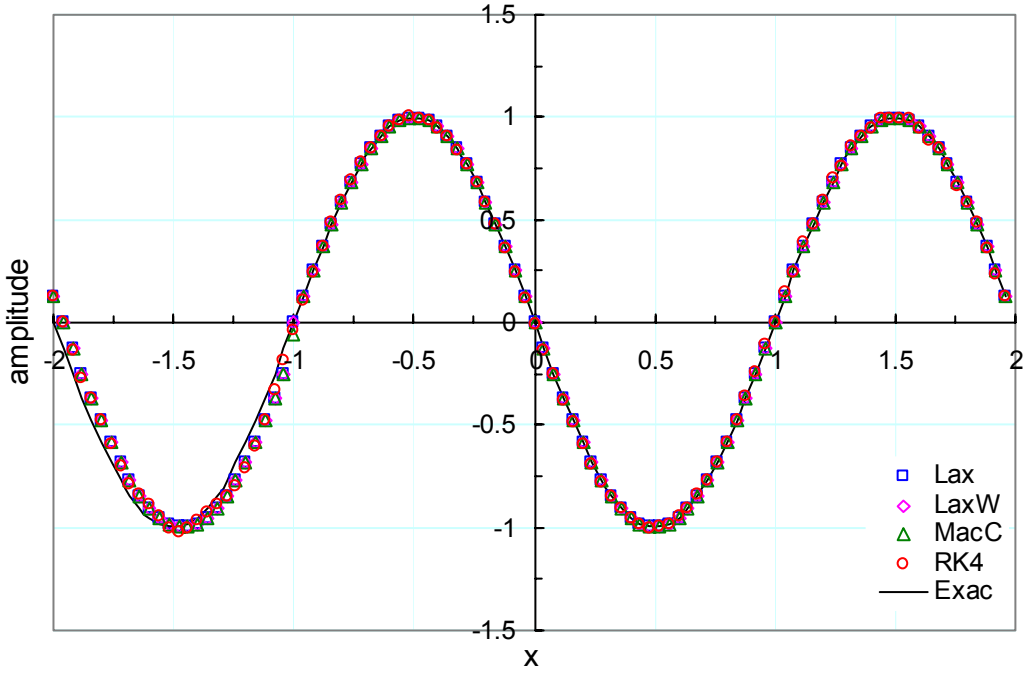

Fig.1d Comparative Plots for Courant No., ν = 1.0

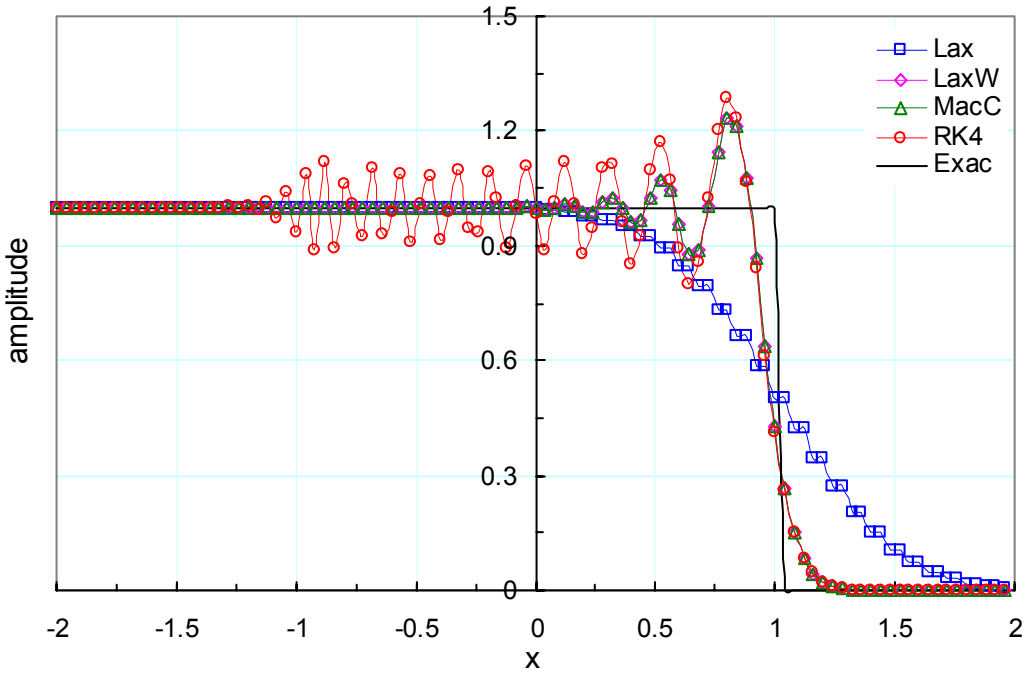

Fig.2a Comparative Plots for Courant No., ν = 0.25

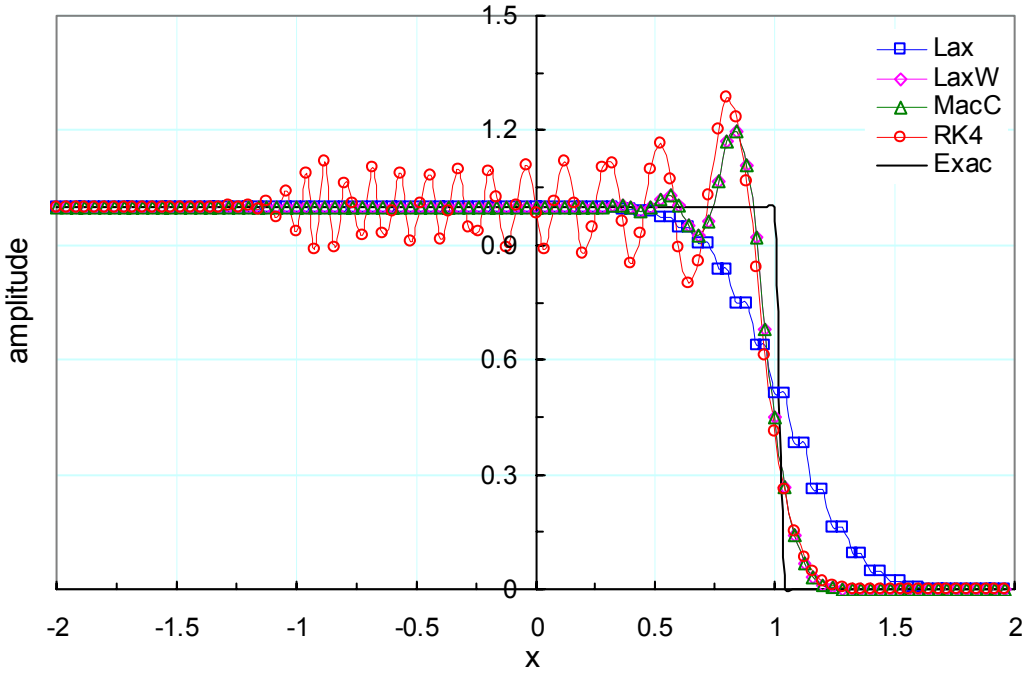

Fig.2b Comparative Plots for Courant No., ν = 0.5

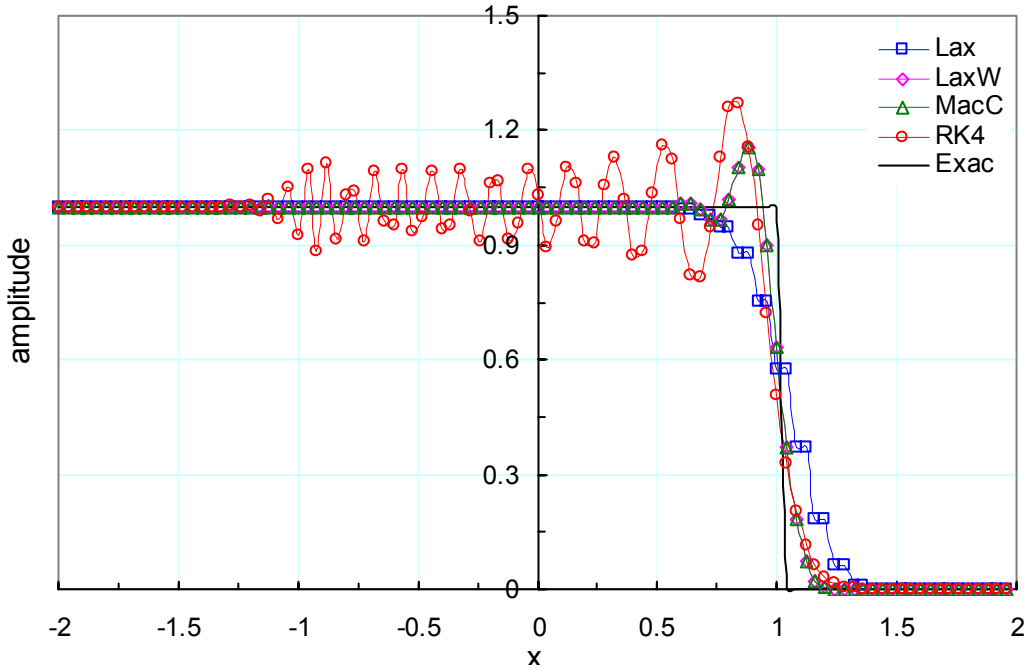

Fig.2c Comparative Plots for Courant No., ν = 0.75

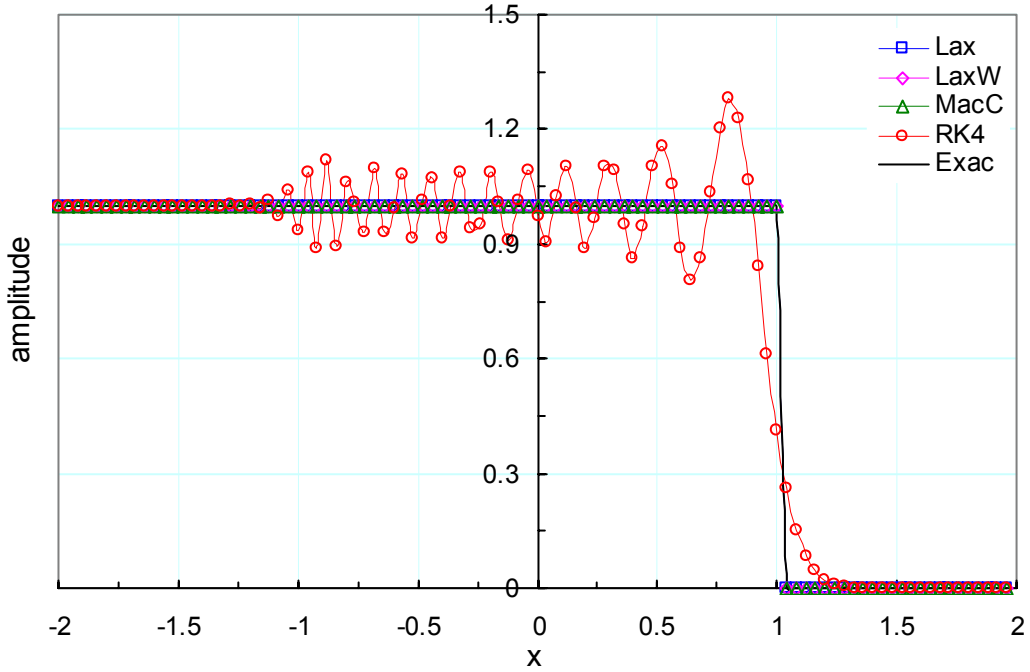

Fig.2d Comparative Plots for Courant No., ν = 1.0

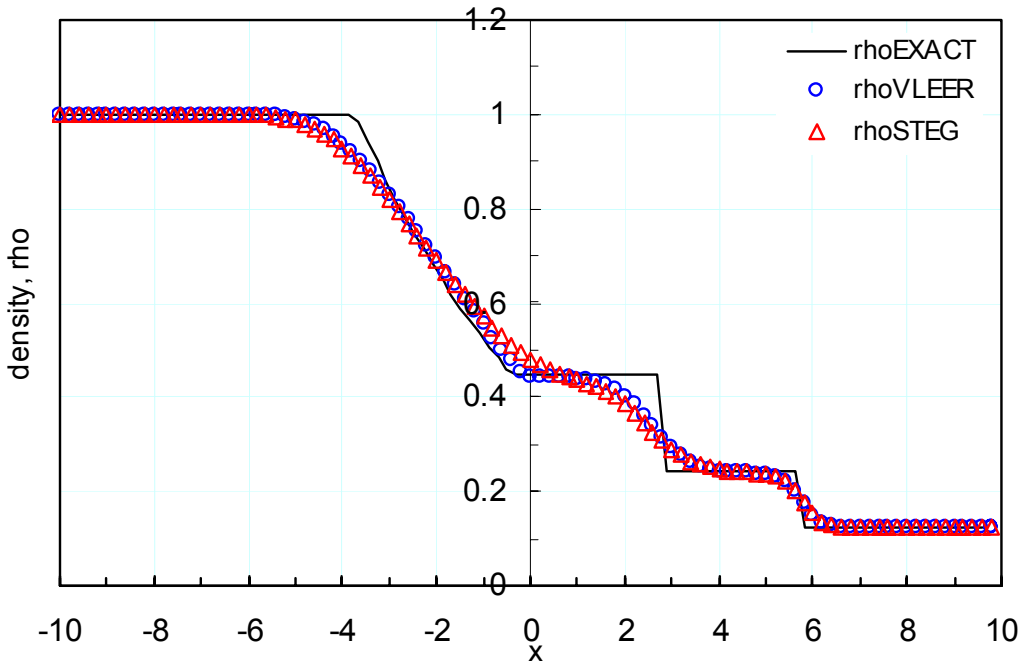

Fig.3a Comparative Plots for density, ρ

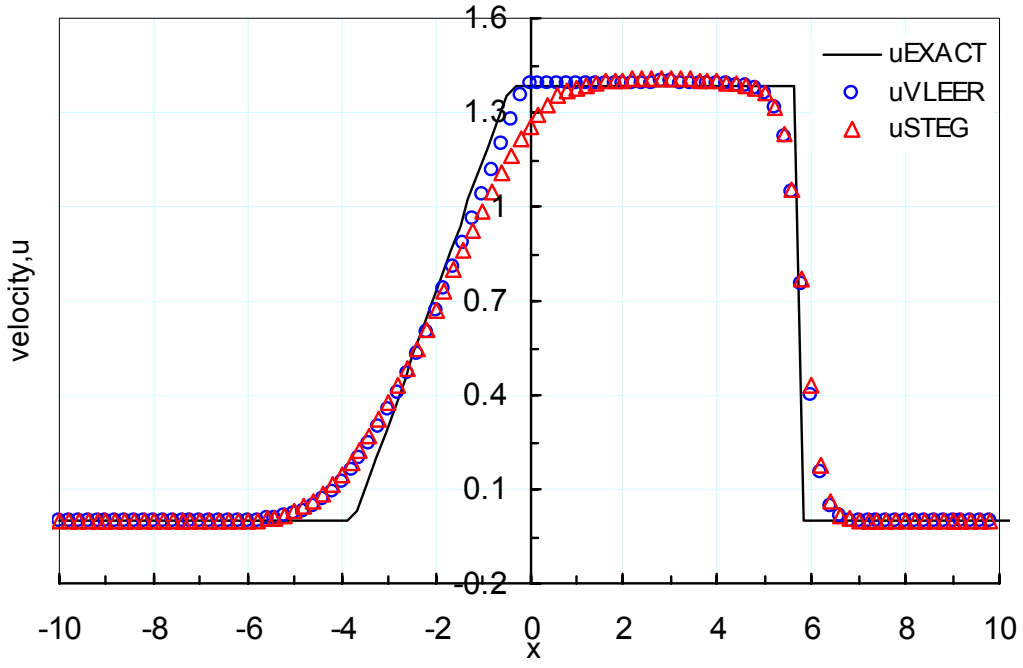

Fig.3b Comparative Plots velocity, u

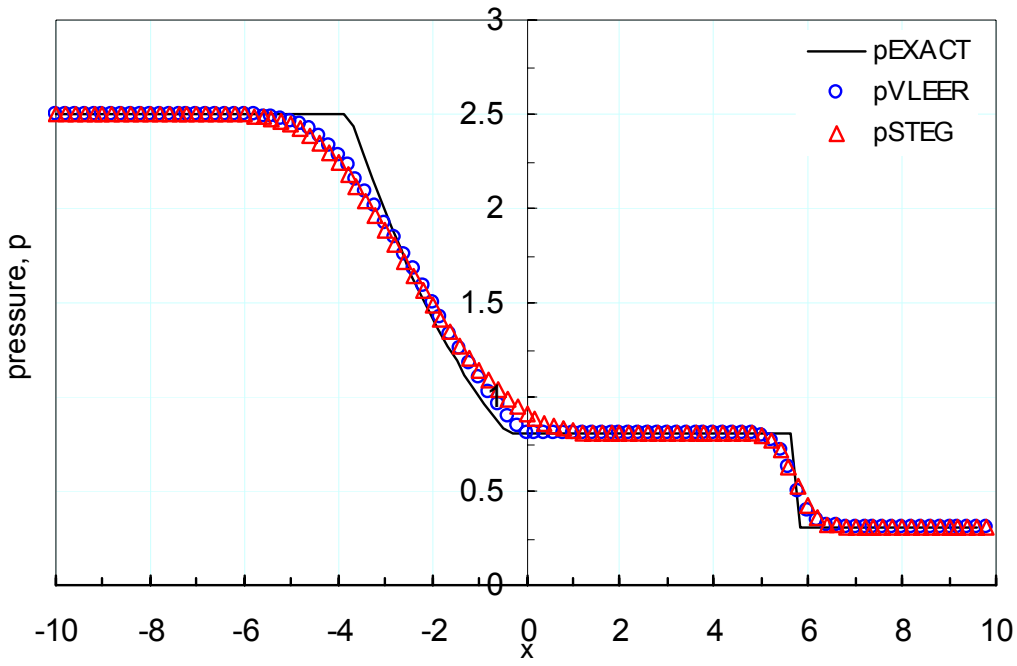

Fig.3b Comparative Plots for pressure, p

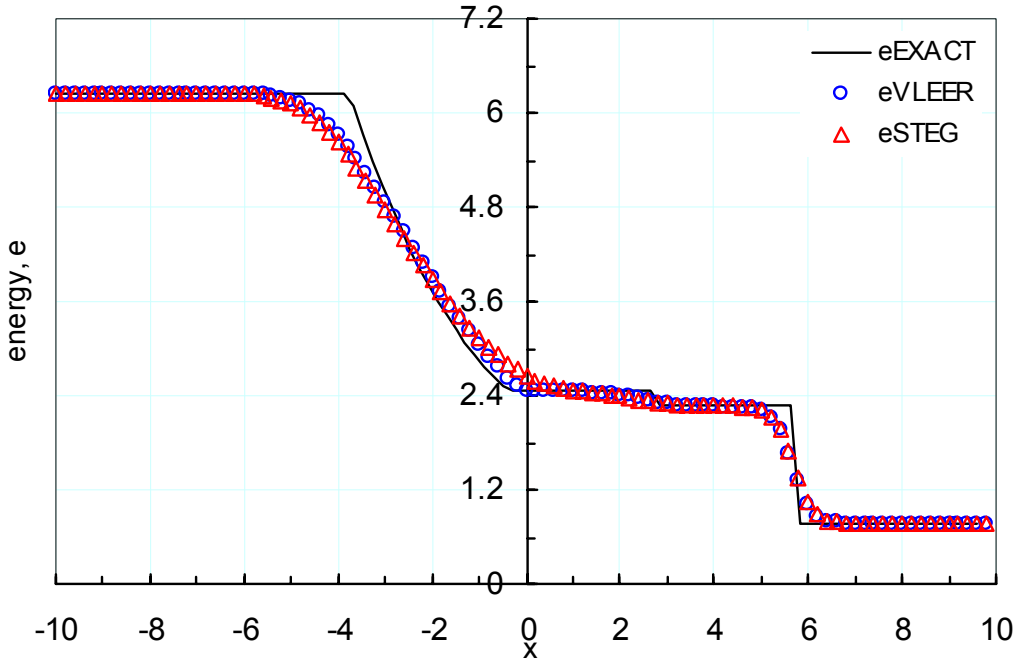

Fig.3b Comparative Plots for energy, e

APPENDIX C-Codes

```
/*********************************************************************/
/* This is LAX scheme to solve 1D wave eq.
/*********************************************************************/
#include<stdio.h>
#include<string.h>
#include<math.h>
#define  IMAX 100
#define SQR(X) ((X) * (X))
double PI = 2.*asin(1.);
FILE *pt1,*pt2;
int main()
{
 int  i,j;
 double   c,t,dt,dx,nu,beta,x[IMAX+1],u[IMAX+1],u1[IMAX+1],uExact[IMAX+1];
 char outnam1[20],outnam2[20];

 pt1 = fopen ("laxDisc_nu1.dat", "w");
 /********* //Input values//******************/
      c = 1./2.;  // Wave speed
      nu = 1.;    // CFL number
      dx = 4./100.; // domain length is 20
      dt = nu*dx/c;

/********** Initialization for SINE WAVE *******/
      /*for(i=0;i<=IMAX;i++){
            x[i] = 0.0;
            u[i] = 0.0;
            x[i] = -2. + i*dx;
            u[i] = sin(PI*x[i]);
      }*/

/********** Initialization for DISCONTINUITY *******/
      for(i=0;i<=IMAX;i++){
            x[i] = 0.0;
            u[i] = 0.0;
            x[i] = -2. + i*dx;
            if(x[i]<= 0.)
            u[i] = 1.0;
            else u[i] = 0.;
      }

/**************** Calculations *****************/
      t = 0.;
      while(t < 2.){
            for(i=1;i<IMAX;i++)
                  u1[i]=0.5*(u[i+1]+u[i-1])-nu*0.5*(u[i+1]-u[i-1]);

                  /* Periodic BC for SINE WAVE*/
                  /*u1[0]=u1[IMAX-1];
                  u1[IMAX]=u1[1]; */
```

```
            /* BC for DISCONTINUITY*/
                  u1[0]=1.;
                  u1[IMAX]=0.;

                  for(j=0;j<IMAX;j++)
                        u[j]=u1[j];
                        t+=dt;
                        if(t > 1.999) t = 2.0;
      }
      printf("t=%f\n",t);
      /*Generate Exact solution for Discontinuity*/
      for(j=0;j<IMAX;j++){
            if(x[j]<= c*t)
                  uExact[j] = 1.;
            else
                  uExact[j] = 0.;
      }
      /*Generate Exact solution for SINE WAVE*/
      /*for(j=0;j<=IMAX;j++)
            uExact[j] = sin(PI*(x[j]-c*t));*/

                  fprintf(pt1,"After t=%f\n",t);
                  for(i=0;i<IMAX;i++){
                  fprintf(pt1,"%6.4f %6.4f %6.4f \n",x[i],u[i],uExact[i]);
      }
      /*******************************************************/
      fclose(pt1);
 }

      /---------------------------- END-------------------------------*/



/*********************************************************************/
/*          This is LAX WENDROFF scheme to solve wave eq. using
/*********************************************************************/
#include<stdio.h>
#include<string.h>
#include<math.h>
#define  IMAX 100
#define SQR(X) ((X) * (X))
double PI = 2.*asin(1.);
FILE *pt1;
int main()
{
 int  i,j;
 double   c,t,dt,dx,nu,beta,x[IMAX+1],u[IMAX+1],u1[IMAX+1],uExact[IMAX+1];
 char outnam1[20],outnam2[20];

      pt1 = fopen ("laxWendDisc_nu025.dat", "w");

 /********* //Input values//*******************/
      c = 1./2.;  // Wave speed
      nu = 0.25;  // CFL number
      dx = 4./100.; // domain length is 20
            dt = nu*dx/c;
```

```
 /********** Initialization for SINE WAVE *******/
    /*for(i=0;i<=IMAX;i++){
          x[i] = 0.0;
          u[i] = 0.0;
          x[i] = -2. + i*dx;
          u[i] = sin(PI*x[i]);
    }*/
    /********** Initialization for DISCONTINUITY *******/
    for(i=0;i<=IMAX;i++){
          x[i] = 0.0;
          u[i] = 0.0;
          x[i] = -2. + i*dx;
          if(x[i]<= 0.)
          u[i] = 1.0;
          else u[i] = 0.;
    }
    /**************** Calculations *****************/
    t = 0.;
 while(t < 2.){
    for(i=1;i<IMAX;i++)
    u1[i]=u[i]-0.5*nu*(u[i+1]-u[i-1])+0.5*SQR(nu)*(u[i+1]-2.*u[i]+u[i-1]);

                /* Periodic BC for SINE WAVE*/
                /*u1[0]=u1[IMAX-1];
                u1[IMAX]=u1[1]; */

                /* BC for DISCONTINUITY*/
                u1[0]=1.;
                u1[IMAX]=0.;

                for(j=0;j<IMAX;j++)
                      u[j]=u1[j];
                      t+=dt;
                      if(t > 1.999) t = 2.0;
    }
    printf("t=%f\n",t);
    /*Generate Exact solution for Discontinuity*/
    for(j=0;j<IMAX;j++){
          if(x[j]<= c*t)
                uExact[j] = 1.;
          else
                uExact[j] = 0.;
    }
    /*Generate Exact solution for SINE WAVE*/
    /*for(j=0;j<IMAX;j++)
          uExact[j] = sin(PI*(x[j]-c*t));*/

                fprintf(pt1,"After t=%f\n",t);
                for(i=0;i<IMAX;i++){
                fprintf(pt1,"%6.4f %6.4f %6.4f \n",x[i],u[i],uExact[i]);
    }
    /*****************************************************/
    fclose(pt1);
}

    /---------------------- END----------------------*/
```

```c
/*************************************************************************/
// This is MacCormack's predictor Corrector method to solve linear wave
// equation
/*************************************************************************/

#include<stdio.h>
#include<string.h>
#include<math.h>
#define  IMAX 100
double PI = 2.*asin(1.);
FILE *pt1;
int main()
{
      int   i,j,k;
      double      c,t,dx,dt,nu;
      double
      x[IMAX+1],u[IMAX+1],uPred[IMAX+1],uCorr[IMAX+1],uExact[IMAX+1];
      char  outnam1[20],outnam2[20];

      pt1 = fopen ("macCormSine_nu025.dat", "w");

      /*********  //Input values//*******************/
      c = 1./2.;  // Wave speed
      nu = 0.25;  // CFL number
      dx = 4./100.; // domain length is 20
      dt = nu*dx/c;
      /********** Initialization for SINE WAVE *******/
      for(i=0;i<=IMAX;i++){
            x[i] = 0.0;
            u[i] = 0.0;
            x[i] = -2. + i*dx;
            u[i] = sin(PI*x[i]);
      }
      /********** Initialization for DISCONTINUITY *******/
      /*for(i=0;i<=IMAX;i++){
            x[i] = 0.0;
            u[i] = 0.0;
            x[i] = -2. + i*dx;
            if(x[i]<= 0.)
            u[i] = 1.0;
            else u[i] = 0.;
      }*/
      /**************** Predictor step ****************/
      t = 0.;
      while(t<2.){
            for(i=0;i<IMAX;i++)
                  uPred[i]=u[i]-nu*(u[i+1]-u[i]);

                  // BCs for Discontinuity
                  /*uPred[0] = 1.;
                  uPred[IMAX] = 0.;  */

                  // Periodic BC for Sine Wave
                  uPred[0]=uPred[IMAX-1];
                  uPred[IMAX]=uPred[1];
      /**************** Corrector step *****************/
                  for(i=1;i<=IMAX;i++)
                  uCorr[i]=0.5*((u[i]+uPred[i])-nu*(uPred[i]-uPred[i-1]));
```

```
            // BCs for Discontinuity
                  /*uCorr[0] = 1.;
                  uCorr[IMAX] = 0.;         */

                  // Periodic BC for Sine Wave
                  uCorr[0]=uCorr[IMAX-1];
                  uCorr[IMAX]=uCorr[1];
      /*********************************************/
                  for(j=0;j<=IMAX;j++)
                        u[j]=uCorr[j];
                        t+=dt;
                        if(t > 1.999) t = 2.0;
      }
      /*Generate Exact solution for Discontinuity*/
      /*for(j=0;j<IMAX;j++){
            if(x[j]<= c*t)
                  uExact[j] = 1.;
            else
                  uExact[j] = 0.;
      }*/
      /*Generate Exact solution for SINE WAVE*/
      for(j=0;j<IMAX;j++)
            uExact[j] = sin(PI*(x[j]-c*t));

                  fprintf(pt1,"After t=%f\n",t);
                  for(i=0;i<IMAX;i++){
                  fprintf(pt1,"%6.4f %6.4f %6.4f \n",x[i],u[i],uExact[i]);
      }
      /*****************************************************/
      fclose(pt1);
 }

      /---------------------- END----------------------*/

/*************************************************************************/
// This is Runge-Kutta 4th order Scheme method to solve linear wave
// equation
/*************************************************************************/

#include<stdio.h>
#include<string.h>
#include<math.h>
#define  IMAX 100
double PI = 2.*asin(1.);
FILE *pt1,*pt2;

void rungk4(double [IMAX+1],double [IMAX+1], double, double, double);
void getR(double [IMAX+1], double [IMAX+1],double, double, double);

int main()
{
 int  i,j;
 double c,t,dx,dt,nu;
 double  x[IMAX+1],u[IMAX+1],uNew[IMAX+1],uCorr[IMAX+1],uExact[IMAX+1];
 char outnam1[20];

 pt1 = fopen ("rungkDisc_nu1.dat", "w");
```

```
/*********  //Input values//*******************/
      c = 1./2.;  // Wave speed
      nu = 1.;    // CFL number
      dx = 4./100.; // domain length is 20
      dt = nu*dx/c;

      /********** Initialization for SINE WAVE *******/
      /*for(i=0;i<=IMAX;i++){
            x[i] = 0.0;
            u[i] = 0.0;
            x[i] = -2. + i*dx;
            u[i] = sin(PI*x[i]);
      }*/
      /********** Initialization for DISCONTINUITY *******/
      for(i=0;i<=IMAX;i++){
            x[i] = 0.0;
            u[i] = 0.0;
            x[i] = -2. + i*dx;
            if(x[i]<= 0.)
            u[i] = 1.0;
            else u[i] = 0.;
      }

      /*************** Calculations ****************/
      t = 0.;
      while(t<2.){
            rungk4(u,uNew,dx,dt,c);
            for(i=0;i<=IMAX;i++)
                  u[i]=uNew[i];

                  /* BCs for Discontinuity */
                  /*u[0] = 1.;
                  u[IMAX] = 0.; */

                  /* Periodic BC for Sine Wave */
                  /*u[0]=u[IMAX-1];
                  u[IMAX]=u[1];  */
                  t+=dt;
                        if(t > 1.999) t = 2.0;
      }
      /*Generate Exact solution for Discontinuity*/
      for(j=0;j<IMAX;j++){
            if(x[j]<= c*t)
                  uExact[j] = 1.;
            else
                  uExact[j] = 0.;
      }

      /*Generate Exact solution for SINE WAVE*/
      /*for(j=0;j<=IMAX;j++)
            uExact[j] = sin(PI*(x[j]-c*t));*/

                  fprintf(pt1,"After t=%f\n",t);
                  for(i=0;i<IMAX;i++){
                  fprintf(pt1,"%6.4f %6.4f %6.4f \n",x[i],u[i],uExact[i]);
      }
      /*************************************************/
      fclose(pt1);
 }
```

```
void rungk4(double y[IMAX+1],double yNew[IMAX+1], double dX, double dT,
double C)
{
	int i,j,k;
	double R1[IMAX+1],R2[IMAX+1],R3[IMAX+1],R4[IMAX+1];

	for(j=0;j<=IMAX;j++){
			yNew[j]=0.;
			R1[j]=0.;
			R2[j]=0.;
			R3[j]=0.;
			R4[j]=0.;
	}
	getR(R1,y,C,dT,dX);
	for(i=1;i<IMAX;i++)
		yNew[i] = y[i] + 0.5*dT*R1[i];

	/*-----BC for Sine Wave ----*/
		/*yNew[0] = yNew[IMAX-1];
		yNew[IMAX] = yNew[1];*/

	/*--BC for Discontinuity ---*/
		yNew[0] = 1.;
		yNew[IMAX] = 0.;

	getR(R2,yNew,C,dT,dX);
	for(i=1;i<IMAX;i++)
		yNew[i]= y[i] + 0.5*dT*R2[i];

	/*-----BC for Sine Wave ----*/
		/*yNew[0] = yNew[IMAX-1];
		yNew[IMAX] = yNew[1];*/

	/*--BC for Discontinuity ---*/
		yNew[0] = 1.;
		yNew[IMAX] = 0.;

	getR(R3,yNew,C,dT,dX);
	for(i=1;i<IMAX;i++)
		yNew[i]= y[i] + dT*R3[i];

	/*-----BC for Sine Wave ----*/
		/*yNew[0] = yNew[IMAX-1];
		yNew[IMAX] = yNew[1];*/

	/*--BC for Discontinuity ---*/
		yNew[0] = 1.;
		yNew[IMAX] = 0.;

	getR(R4,yNew,C,dT,dX);
	for(i=1;i<IMAX;i++)
		yNew[i]= y[i] + (dT/6.)*(R1[i]+2.*R2[i]+2.*R3[i]+R4[i]);
```

```
      /*-----BC for Sine Wave ----*/
            /*yNew[0] = yNew[IMAX-1];
            yNew[IMAX] = yNew[1];*/

      /*--BC for Discontinuity ---*/
            yNew[0] = 1.;
            yNew[IMAX] = 0.;
}
void getR(double R[IMAX+1],double y[IMAX+1], double c, double dt, double dx)
{
      int i;
      for(i=1;i<IMAX;i++)
            R[i]= -0.5*c*(y[i+1]-y[i-1])/dx;
}



      /---------------------------- END-------------------------------*/



/*************************************************************************/
// This program solves 1D shock problem eploying StegerWarming Splitting/
// Scheme and Van Leer Flux Splitting Schemes as subroutines            /
/*************************************************************************/

#include<stdio.h>
#include<string.h>
#include<math.h>
#define  N 100
#define  IMAX 102
FILE *pt1;
      float t1=0.0;
     void  vleer(float fiph[][IMAX],float fimh[][IMAX],double
u[][IMAX],float a[IMAX],float H[IMAX],int M,float gam);
    void  steger(float fiph[][IMAX],float fimh[][IMAX],double
u[][IMAX],float a[IMAX],float H[IMAX],int M,float gam);
int main()
{
 int  i,j,M=N ;
 float   dx,dt=0.0,cfl=0.636,a[IMAX],H[IMAX],gam=1.4;
 float      x,y,t,te,U[3][IMAX][2],fiph[3][IMAX],fimh[3][IMAX];
 double     w[3][IMAX];
 char choice,outnam1[20],outnam2[20];

 float timestep(double w[][IMAX],float a[IMAX],float dx,int M,float cfl);

 printf("Enter output filename for Results......");
 scanf("%s",outnam1);
 pt1 = fopen (outnam1,"w");

 dx= 20.0/N;
 x = -10.;
```

```
/* INITIALIZATION */
 for(i=0;i<=N+1;i++){
      if(x<=0.){
            w[0][i]=1.0;
      w[1][i]=0.0;
      w[2][i]=1./(gam-1.);
      }
      else {
            w[0][i]=0.125;
            w[1][i]=0.0;
            w[2][i]=0.125/(gam-1.);
      }
      x=x+dx;
 }
 for(i=1;i<=N;i++){
    U[0][i][0]=w[0][i];
    U[1][i][0]=w[0][i]*w[1][i];
    U[2][i][0]=((w[2][i]/(gam-1))+0.5*w[0][i]*w[1][i]*w[1][i]);
    a[i] = sqrt((w[2][i]*gam)/w[0][i]);
    H[i] = (U[2][i][0]+w[2][i])/w[0][i];
 }
    H[0]=H[1];
    a[0]=a[1];
    H[N+1]=H[N];
    a[N+1]=a[N];
    t1=0.0;
    while(t1<=2.){
      dt=timestep(w,a,dx,N,cfl);

      printf(" dt = %f\n",dt);
    //steger(fiph,fimh,w,a,H,N,gam);
      vleer(fiph,fimh,w,a,H,N,gam);
      for(i=1;i<=N;i++){
            for(j=0;j<3;j++)
                  U[j][i][1]=U[j][i][0]- (dt/dx)*(fiph[j][i]-fimh[j][i]);
                  w[0][i]=U[0][i][1];
                  w[1][i]=U[1][i][1]/U[0][i][1];

            w[2][i]=(U[2][i][1]-0.5*U[0][i][1]*w[1][i]*w[1][i])*(gam-1);
                  H[i]=(U[2][i][1]+w[2][i])/w[0][i];
                  a[i]=sqrt((w[2][i]*gam)/w[0][i]);
      }
      for(j=0;j<3;j++){
            w[j][0]=w[j][1];
            w[j][N+1]=w[j][N];
      }

      for(i=1;i<=N;i++)
            for(j=0;j<3;j++)
                  U[j][i][0]=U[j][i][1];

      /*printf(" dt = %f t1 = %f\n",dt,t1);*/
      t1=t1+dt;
      printf(" dt = %f t1 = %f\n",dt,t1);
      }
```

```
}
      for(i=0;i<N;i++){
            y = (-10.+i*dx);
            fprintf(pt1,"%4.3f  %5.4f  %5.4f  %5.4f
%5.4f\n",y,w[0][i+1],w[1][i+1],w[2][i+1],U[2][i+1][1]/*/U[0][i+1][1]*/);

      }
      fclose(pt1);
}

float timestep(double x[][IMAX], float a[IMAX],float dx,int m,float c)
{
int i;
float s=0.0,temp=0.0,t;
for(i=1;i<=m;i++){
      temp= fabs(x[1][i])+a[i];
       if(temp>=s){
       s=temp;
       }
     else
       temp=0.0;
}

t=(c*dx)/s;
return(t);
}

void steger(float fph[][IMAX],float fmh[][IMAX],double u[][IMAX],float
                     aa[IMAX], float h[IMAX],int m,float g)
{

int i,j;
double l1[IMAX],l2[IMAX],l3[IMAX];
float lp1[IMAX],lp2[IMAX],lp3[IMAX],fac=0.0;
float lm1[IMAX],lm2[IMAX],lm3[IMAX],fp[3][IMAX],fm[3][IMAX];

for(i=0;i<=m+1;i++){
   fac=(0.5*u[0][i])/g;
   l1[i]=u[1][i]-aa[i];
   l2[i]=u[1][i];
   l3[i]=u[1][i]+aa[i];
   lp1[i]=0.5*(l1[i]+fabs(l1[i]));
   lp2[i]=0.5*(l2[i]+fabs(l2[i]));
   lp3[i]=0.5*(l3[i]+fabs(l3[i]));
   lm1[i]=0.5*(l1[i]-fabs(l1[i]));
   lm2[i]=0.5*(l2[i]-fabs(l2[i]));
   lm3[i]=0.5*(l3[i]-fabs(l3[i]));

   fp[0][i]=fac*(lp1[i]+2.0*(g-1)*lp2[i]+lp3[i]);
   fp[1][i]=fac*((u[1][i]-aa[i])*lp1[i]+2.0*(g-
                         1)*u[1][i]*lp2[i]+(u[1][i]+aa[i])*lp3[i]);
   fp[2][i]=fac*((h[i]-u[1][i]*aa[i])*lp1[i]+(g-
                     1)*u[1][i]*u[1][i]*lp2[i]+(h[i]+u[1][i]*aa[i])*lp3[i]);

   fm[0][i]=fac*(lm1[i]+2.0*(g-1)*lm2[i]+lm3[i]);
   fm[1][i]=fac*((u[1][i]-aa[i])*lm1[i]+2.0*(g-
                       1)*u[1][i]*lm2[i]+(u[1][i]+aa[i])*lm3[i]);
   fm[2][i]=fac*((h[i]-u[1][i]*aa[i])*lm1[i]+(g-
               1)*u[1][i]*u[1][i]*lm2[i]+(h[i]+u[1][i]*aa[i])*lm3[i]);
```

```
       fac=0.0;

  }

for(i=1;i<=m;i++){
      for(j=0;j<3;j++){
        fph[j][i]=0.0;
        fmh[j][i]=0.0;
            fph[j][i]=fp[j][i]+fm[j][i+1];
        fmh[j][i]=fp[j][i-1]+fm[j][i];
      }
}
}

void vleer(float fph[][IMAX],float fmh[][IMAX],double u[][IMAX],float
aa[IMAX], float h[IMAX],int m,float g)
{

int i,j;
double mach;
float co_p=0.0,co_m=0.0,gf1=0.0,gf2=0.0,fp[3][IMAX],fm[3][IMAX];

for(i=0;i<=m+1;i++){
    mach=u[1][i]/aa[i];
    gf1=(2.0*aa[i])/g;
    gf2=(2.0*aa[i]*aa[i])/(g*g-1.0);
    co_p= 0.25*u[0][i]*aa[i]*pow((1.0+mach),2.0);
    co_m= 0.25*-1*u[0][i]*aa[i]*pow((1.0-mach),2.0);

      fp[0][i]=co_p;
      fp[1][i]=co_p*gf1*(0.2*mach+1.0);;
      fp[2][i]=co_p*gf2*pow((0.2*mach+1.0),2.0);

      fm[0][i]=co_m;
      fm[1][i]=co_m*gf1*(0.2*mach-1.0);
      fm[2][i]=co_m*gf2*pow((0.2*mach-1.0),2.0);
      mach=0.0;
      gf1=0.0;
      gf2=0.0;
      co_p=0.0;
      co_m=0.0;
 }

for(i=1;i<=m;i++){

      for(j=0;j<3;j++){
        fph[j][i]=0.0;
        fmh[j][i]=0.0;
            fph[j][i]=fp[j][i]+fm[j][i+1];
        fmh[j][i]=fp[j][i-1]+fm[j][i];
      }
 }
}



      /---------------------------- DOCUMENT END-------------------------------*/
```