Project 3

CHUNG,Chak Pong

20015116

Problem

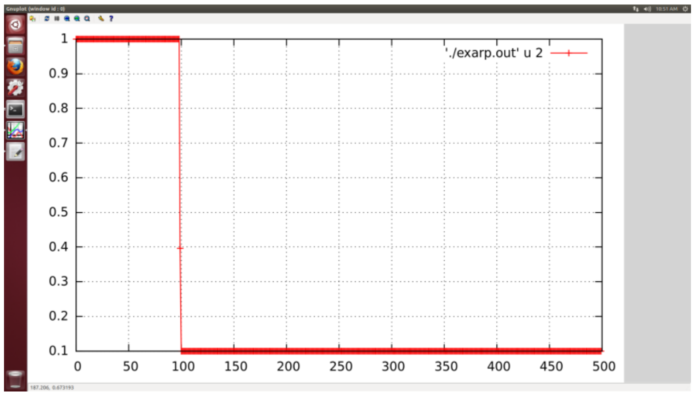
One dimentional shallow water equation with exact Riemann Solver

with Dam-Break Initial State UL=UR=0,HL=1,HR=0.1

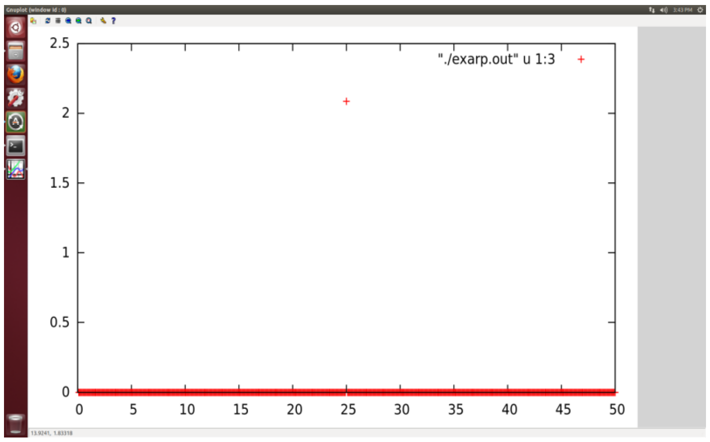
$$\begin{bmatrix} h \\ hu \end{bmatrix}_t + \begin{bmatrix} uh \\ hu^2 + \frac{1}{2}gh^2 \end{bmatrix}_x = 0.$$

Initial Condition

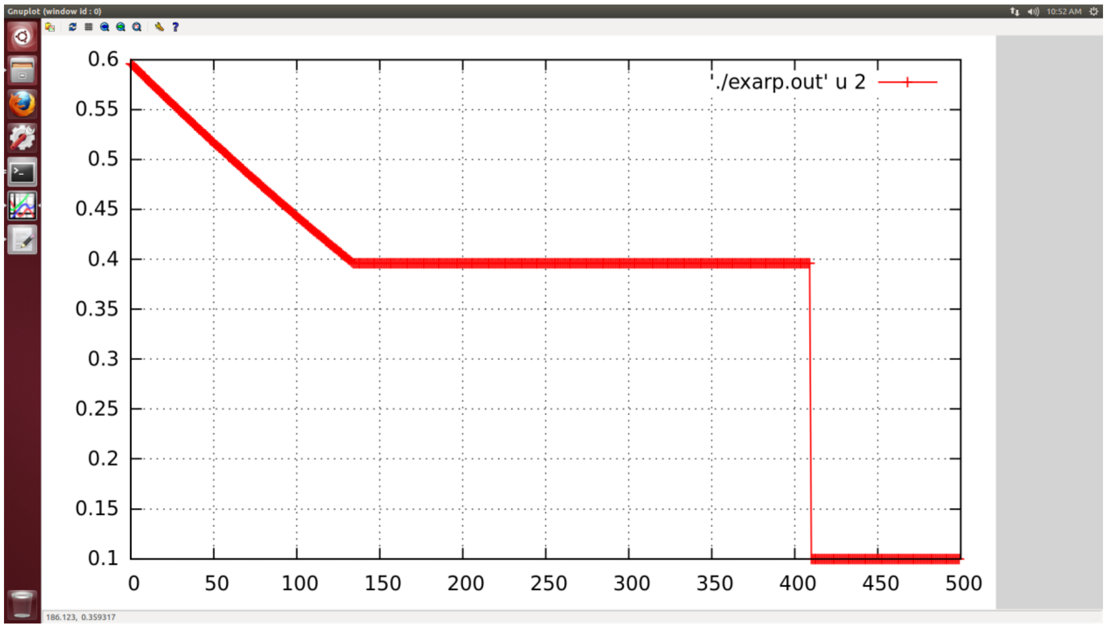
Initial H



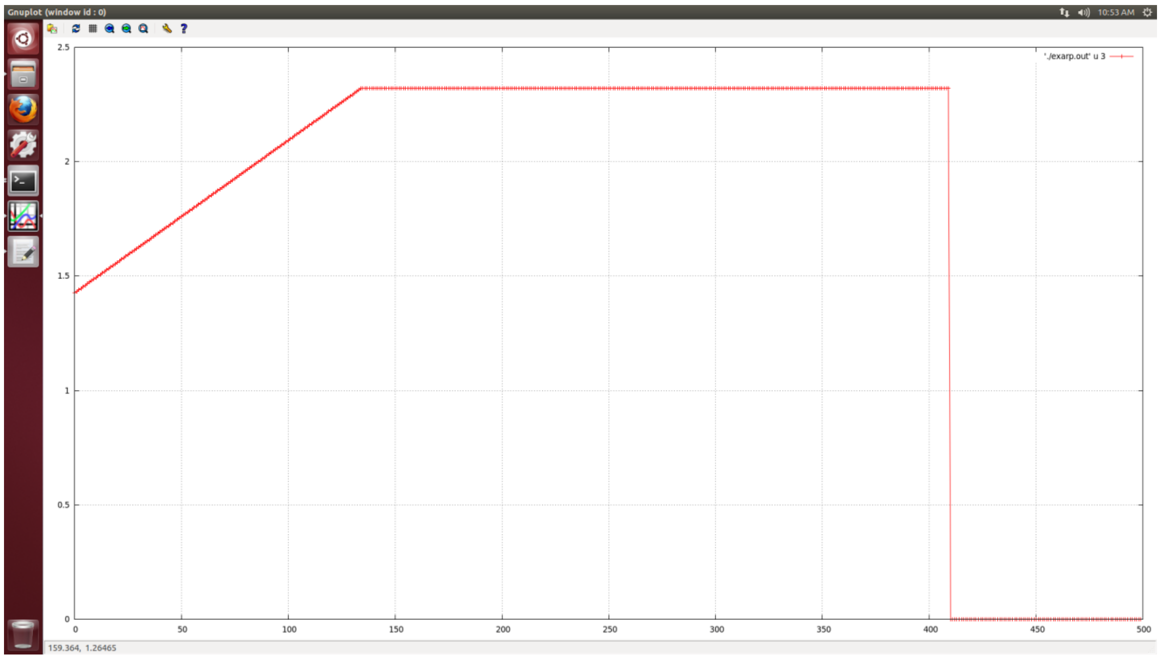
Initial HU



H at 10s



HU at 10s



Key steps in determing the Flux at star region

- Find $h_\star$ such that $\Phi(h_\star) = \Phi_r(h_\star) - \Phi_\ell(h_\star) = 0,$ where

$$\Phi_\ell(h_\star) := \begin{cases} u_\ell - (h_\star - h_\ell)\sqrt{g\left(\frac{1}{2h_\star} + \frac{1}{2h_\ell}\right)} & \text{if } h_\star > h_\ell \\ u_\ell + 2\left(\sqrt{gh_\ell} - \sqrt{gh_\star}\right) & \text{if } h_\star \le h_\ell \end{cases}$$

$$\Phi_r(h_\star) := \begin{cases} u_r + (h_\star - h_r)\sqrt{g\left(\frac{1}{2h_\star} + \frac{1}{2h_r}\right)} & \text{if } h_\star > h_r \\ u_r - 2\left(\sqrt{gh_r} - \sqrt{gh_\star}\right) & \text{if } h_\star \le h_r \end{cases}$$

- Newton iteration: $h_\star^{k+1} = h_\star^k - \frac{\Phi(h_\star^k)}{\Phi'(h_\star^k)}$

u* is determined by

US = 0.5*(UL + UR) + 0.5*(FR - FL)

A more detailed derivation can be found in Toro 's book *Shock-Capturing Methods for Free-Surface Shallow Flows*, section 5.3.,from equation 5.5 to equation 5.12

The code used is attached.

```
*
      IMPLICIT NONE
*
C     Declaration of variables
*
      REAL    CHALEN,CL,CR,DCRIT,DL,DR,GATE,GRAVIT,TIMOUT,
     &        TOL,UL,UR
*
      INTEGER MCELLS, NITER
*
      COMMON /STATES/ CL, DL, UL, CR, DR, UR
      COMMON /ACCELE/ GRAVIT
      COMMON /TOLERA/ NITER, TOL
      COMMON /DOMAIN/ CHALEN, GATE, MCELLS, TIMOUT
*
C     Initial data and computational parameters are read in
*
      OPEN(UNIT=1,FILE='exarp.ini',STATUS='UNKNOWN')
*
      READ(1,*)CHALEN    ! length of channel
      READ(1,*)GATE      ! position of gate
      READ(1,*)GRAVIT    ! acceleration due to gravity
      READ(1,*)MCELLS    ! number of cells in profile
      READ(1,*)TOL       ! tolerance for convergence test
      READ(1,*)NITER     ! iterations in exact solver
      READ(1,*)TIMOUT    ! output time
      READ(1,*)DL        ! depth on left reservoir
      READ(1,*)UL        ! velocity in left reservoir
      READ(1,*)DR        ! depth in right reservoir
      READ(1,*)UR        ! velocity in right reservoir
*
      CLOSE(1)
*
C     Compute celerity on left and right states
*
      CL = SQRT(GRAVIT*DL)
      CR = SQRT(GRAVIT*DR)
*
C     Use the "depth positivity condition" to identify
C     type of data and thus of solution and to call
C     appropriate exact solver
*
      DCRIT = (UR-UL) - 2.0*(CL+CR)
*
      IF(DL.LE.0.0.OR.DR.LE.0.0.OR.DCRIT.GE.0.0)THEN
*
C        Dry bed cases
*
         CALL DRYBED
*
      ELSE
*
C        Wet bed case
*
         CALL WETBED
*
```

```
      ENDIF
*
C     Results are printed out
*
      CALL OUTPUT
*
      END
*
                                    *
*----------------------------------------------------------*
*
      SUBROUTINE OUTPUT
*
C     Purpose: to output exact solution at chosen
C              output time TIMOUT
*
      IMPLICIT NONE
*
C     Declaration of variables
*
      INTEGER  MX, I, MCELLS
*
      REAL     D, U,CHALEN,GATE,TIMOUT,XCOORD
*
      PARAMETER (MX = 3000)
*
      DIMENSION D(MX), U(MX)
*
      COMMON /SOLUTI/ D, U
      COMMON /DOMAIN/ CHALEN, GATE, MCELLS, TIMOUT
*
      OPEN(UNIT=1,FILE='exarp.out',STATUS='UNKNOWN')
*
      DO 10 I = 1, MCELLS
         XCOORD = REAL(I)*CHALEN/REAL(MCELLS)
         WRITE(1,20)XCOORD, D(I), U(I)
 10   CONTINUE
*
 20   FORMAT(3(F10.5,4X))
*
      CLOSE(1)
*
      END
*
                                    *
*----------------------------------------------------------*
*
      SUBROUTINE WETBED
*
C     Purpose: to solve the Riemann problem exactly for
C              the wet-bed case
*
      IMPLICIT NONE
*
C     Declaration of variables
*
      INTEGER  I,IT,MCELLS,MX,NITER
```

```
*
      REAL     CHA,CHALEN,CL,CR,CS,D,D0,DL,DR,DS,DSAM,FL,
     &         FLD,FR,FRD,GATE,GRAVIT,S,TIMOUT,TOL,U,UL,
     &         UR,US,USAM,XCOORD
*
      PARAMETER (MX = 3000)
*
      DIMENSION D(MX), U(MX)
*
      COMMON /SOLUTI/ D, U
      COMMON /STATES/ CL, DL, UL, CR, DR, UR
      COMMON /STARSO/ CS, DS, US
      COMMON /ACCELE/ GRAVIT
      COMMON /TOLERA/ NITER, TOL
      COMMON /DOMAIN/ CHALEN, GATE, MCELLS, TIMOUT
*
C     Find starting value for iteration
*
      WRITE(6,*)
      WRITE(6,*)'Exact Solution in Star Region'
      WRITE(6,*)'=============================='
      WRITE(6,*)
*
      CALL STARTE
*
C     Store starting value in D0
*
      D0 = DS
*
C     Start iteration
*
      WRITE(6,*)'   IT   ','   DS         ', '    CHA '
      WRITE(6,*)
      DO 10 IT = 1, NITER
*
         CALL GEOFUN(FL,FLD,DS,DL,CL)
         CALL GEOFUN(FR,FRD,DS,DR,CR)
         DS  = DS - (FL + FR + UR-UL)/(FLD + FRD)
         CHA = ABS(DS-D0)/(0.5*(DS+D0))
         WRITE(6,30)IT,DS,CHA
         IF(CHA.LE.TOL)GOTO 20
         IF(DS.LT.0.0)DS = TOL
         D0 = DS
*
 10   CONTINUE
*
      WRITE(6,*)'Number of NITER iterations exceeded,
     &           STOP'
*
      STOP
*
 20   CONTINUE
 30   FORMAT(I6,2X,2(F12.7,2X))
*
C     Converged solution for depth DS in Star Region.
C     Compute velocity US in Star Region
*
```

```
      US = 0.5*(UL + UR) + 0.5*(FR - FL)
*
      WRITE(6,*)
      WRITE(6,*)'Depth in Star Region    h* =',DS
      WRITE(6,*)'Velocity in Star Region u* =',US
      WRITE(6,*)
*
      CS = SQRT(GRAVIT*DS)
*
C     Evaluate exact solution at time TIMOUT
*
      DO 40 I = 1, MCELLS
*
         XCOORD = REAL(I)*CHALEN/REAL(MCELLS) - GATE
         S      = XCOORD/TIMOUT
*
C        Sample solution throughout wave structure at
C        time TIMOUT
*
         CALL SAMWET(DSAM,USAM,S)
*
C        Store solution
*
         D(I) = DSAM
         U(I) = USAM
*
 40   CONTINUE
*
      END
*
                                    *
*----------------------------------------------------------*
*
      SUBROUTINE GEOFUN(F,FD,D,DK,CK)
*
C     Purpose: to evaluate functions FL, FR and their
C              derivatives in iterative Riemann solver,
C              for wet-bed case.
*
      IMPLICIT NONE
*
C     Declaration of variables
*
      REAL   C,CK,D,DK,F,FD,GES,GRAVIT
*
      COMMON /ACCELE/ GRAVIT
*
      IF(D.LE.DK)THEN
*
C        Wave is rarefaction wave (or depression)
*
         C  = SQRT(GRAVIT*D)
         F  = 2.0*(C-CK)
         FD = GRAVIT/C
      ELSE
*
C        Wave is shock wave (or bore)
```

```
*
         GES = SQRT(0.5*GRAVIT*(D+DK)/(D*DK))
         F   = (D-DK)*GES
         FD  = GES - 0.25*GRAVIT*(D-DK)/(GES*D*D)
      ENDIF
*
      END
*
                                   *
*----------------------------------------------------------*
*
      SUBROUTINE STARTE
*
C     Purpose: to provide starting value for Newton-Raphson
C              iteration. The Two-Rarefaction Riemann
C              Solver (TRRS) and Two-Shock Riemann Solver
C              (TSRS) are used adaptively
*
      IMPLICIT NONE
*
C     Declaration of variables
*
      REAL     CL,CR,CS,DL,DMIN,DR,DS,GEL,GER, GRAVIT,
     &         UL,UR,US
*
      COMMON /STATES/ CL, DL, UL, CR, DR, UR
      COMMON /STARSO/ CS, DS, US
      COMMON /ACCELE/ GRAVIT
*
      DMIN = MIN(DL,DR)
*
C     Use Two-Rarefaction (TRRS) solution as starting value
*
      DS = (1.0/GRAVIT)*(0.5*(CL+CR)-0.25*(UR-UL))**2
*
      IF(DS.LE.DMIN)THEN
*
C        Use Two-Rarefaction (TSRS) approximation as
C        starting value
*
         WRITE(6,*)'TR approximation, h* =',DS
      ELSE
*
C        Use two-shock (TSRS) solution as starting value
C        with DS as computed from TRRS as estimate
*
         WRITE(6,*)'TS approximation, h* =',DS
*
         GEL = SQRT(0.5*GRAVIT*(DS+DL)/(DS*DL))
         GER = SQRT(0.5*GRAVIT*(DS+DR)/(DS*DR))
         DS  = (GEL*DL + GER*DR - (UR-UL))/(GEL + GER)
*
      ENDIF
      WRITE(6,*)
*
      END
*
```

```
                             *
*----------------------------------------------------------*
*
      SUBROUTINE SAMWET(D,U,S)
*
C     Purpose: to sample solution through wave structure at
C              TIMOUT for wet-bed case
*
      IMPLICIT NONE
*
C     Declaration of variables
*
      REAL    C,CL,CR,CS,D,DL,DR,DS,GRAVIT,QL,QR,S,SHL,
     &        SHR,SL,SR,STL,STR,U,UL,UR,US
*
      COMMON /STATES/ CL, DL, UL, CR, DR, UR
      COMMON /STARSO/ CS, DS, US
      COMMON /ACCELE/ GRAVIT
*
      IF(S.LE.US)THEN
********************************************
C        Sample left wave
********************************************
         IF(DS.GE.DL)THEN
*
C           Left shock
*
            QL = SQRT((DS + DL)*DS/(2.0*DL*DL))
            SL = UL - CL*QL
*
            IF(S.LE.SL)THEN
*
C              Sample point lies to the left of the shock
*
               D = DL
               U = UL
            ELSE
*
C              Sample point lies to the right of the shock
*
               D = DS
               U = US
            ENDIF
         ELSE
*
C           Left rarefaction
*
            SHL = UL - CL
*
            IF(S.LE.SHL)THEN
*
C              Sample point lies to the right of the
C              rarefaction
*
               D = DL
               U = UL
            ELSE
```

```
*
               STL = US - CS
*
               IF(S.LE.STL)THEN
*
C                 Sample point lies inside the rarefaction
*
                  U = (UL + 2.0*CL + 2.0*S)/3.0
                  C = (UL + 2.0*CL - S)/3.0
                  D = C*C/GRAVIT
               ELSE
*
C                 Sample point lies in the STAR region
*
                  D = DS
                  U = US
               ENDIF
            ENDIF
         ENDIF
*
      ELSE
********************************************
C        Sample right wave
********************************************
*
         IF(DS.GE.DR)THEN
*
C           Right shock
*
            QR = SQRT((DS + DR)*DS/(2.0*DR*DR))
            SR = UR + CR*QR
*
            IF(S.GE.SR)THEN
*
C              Sample point lies to the right of the shock
*
               D = DR
               U = UR
            ELSE
*
C              Sample point lies to the left of the shock
*
               D = DS
               U = US
            ENDIF
*
         ELSE
*
C           Right rarefaction
*
            SHR = UR + CR
*
            IF(S.GE.SHR)THEN
*
C              Sample point lies to the right of the
C              rarefaction
*
```

```
               D = DR
               U = UR
            ELSE
*
               STR = US + CS
*
               IF(S.GE.STR)THEN
*
C                 Sample point lies inside the rarefaction
*
                  U = (UR  - 2.0*CR + 2.0*S)/3.0
                  C = (-UR + 2.0*CR + S)/3.0
                  D = C*C/GRAVIT
               ELSE
*
C                 Sample point lies in the STAR region
*
                  D = DS
                  U = US
               ENDIF
            ENDIF
         ENDIF
      ENDIF
*
      END
*
                                  *
*----------------------------------------------------------*
*
      SUBROUTINE DRYBED
*
C     Pupose: to compute the exact solution in the case
C             in which a portion of dry bed is present
*
      IMPLICIT NONE
*
C     Declaration of variables
*
      INTEGER I,MCELLS,MX
*
      REAL    CHALEN,CL,CR,D,DL,DR,DSAM,GATE,S,TIMOUT,
     &        U,UL,UR,USAM,XCOORD
*
      PARAMETER (MX = 3000)
*
      DIMENSION D(MX), U(MX)
*
      COMMON /SOLUTI/ D, U
      COMMON /STATES/ CL, DL, UL, CR, DR, UR
      COMMON /DOMAIN/ CHALEN, GATE, MCELLS, TIMOUT
*
      DO 10 I = 1, MCELLS
*
         XCOORD = REAL(I)*CHALEN/REAL(MCELLS) - GATE
         S      = XCOORD/TIMOUT
C
         IF(DL.LE.0.0)THEN
```

```
*
C           Left state is dry
*
            CALL SAMLEF(DSAM,USAM,S)
         ELSE
            IF(DR.LE.0.0)THEN
*
C              Right state is dry
*
               CALL SAMRIG(DSAM,USAM,S)
            ELSE
*
C              Middle state is dry
*
               CALL SAMMID(DSAM,USAM,S)
            ENDIF
         ENDIF
*
         D(I) = DSAM
         U(I) = USAM
*
 10   CONTINUE
*
      END
*
                                    *
*----------------------------------------------------------*
*
      SUBROUTINE SAMLEF(D,U,S)
*
C     Purpose: to sample the solution through the wave
C              structure at time TIMOUT, for the case in
C              which the left state is dry. Solution
C              consists of single right rarefaction
*
      IMPLICIT NONE
*
C     Declaration of variables
*
      REAL   C,CL,CR,D,DL,DR,GRAVIT,S,SHR,STR,U,UL,UR
*
      COMMON /STATES/ CL, DL, UL, CR, DR, UR
      COMMON /ACCELE/ GRAVIT
*
      SHR = UR + CR
*
      IF(S.GE.SHR)THEN
*
C        Sampling point lies to the right of the
C        rarefaction
*
         D = DR
         U = UR
      ELSE
*
         STR = UR-2.0*CR
*
```

```
         IF(S.GE.STR)THEN
*
C           Sampling point lies inside the rarefaction
*
            U = ( UR - 2.0*CR + 2.0*S)/3.0
            C = (-UR + 2.0*CR + S)/3.0
            D = C*C/GRAVIT
         ELSE
*
C           Sampling point lies in dry-bed state
*
            D = DL
            U = UL
         ENDIF
      ENDIF
*
      END
*
                                 *
*----------------------------------------------------------*
*
      SUBROUTINE SAMMID(D,U,S)
*
C     Purpose: to sample the solution through the wave
C              structure at time TIMOUT, for the case in
C              which the middle state is dry. Solution
C              consists of a left and a right rarefaction
C              with a dry portion in the the middle
*
      IMPLICIT NONE
*
C     Declaration of variables
*
      REAL    C,CL,CR,D,DL,DR,GRAVIT,S,SHL,SHR,SSL,SSR,
     &        U,UL,UR
*
      COMMON /STATES/ CL, DL, UL, CR, DR, UR
      COMMON /ACCELE/ GRAVIT
*
C     Compute wave speeds
*
      SHL = UL - CL
      SSL = UL + 2.0*CL
      SSR = UR - 2.0*CR
      SHR = UR + CR
*
      IF(S.LE.SHL)THEN
*
C        Sampling point lies to the left of the left
C        rarefaction
*
         D = DL
         U = UL
      ENDIF
*
      IF(S.GT.SHL.AND.S.LE.SSL)THEN
*
```

```
C        Sampling point lies inside the left rarefaction
*
         U = (UL + 2.0*CL + 2.0*S)/3.0
         C = (UL + 2.0*CL - S)/3.0
         D = C*C/GRAVIT
      ENDIF
*
      IF(S.GT.SSL.AND.S.LE.SSR)THEN
*
C        Sampling point lies inside the middle dry bed region
*
         D = 0.0
         U = 0.0
      ENDIF
*
      IF(S.GT.SSR.AND.S.LE.SHR)THEN
*
C        Sampling point lies inside the right rarefaction
*
         U = ( UR - 2.0*CR + 2.0*S)/3.0
         C = (-UR + 2.0*CR + S)/3.0
         D = C*C/GRAVIT
      ENDIF
*
      IF(S.GT.SHR)THEN
*
C        Sampling point lies to the right of the right
C        rarefaction
*
         D = DR
         U = UR
      ENDIF
*
      END
*
                                  *
*----------------------------------------------------------*
*
      SUBROUTINE SAMRIG(D,U,S)
*
C     Purpose: to sample the solution through the wave
C              structure at time TIMOUT, for the case in
C              which the right state is dry. Solution
C              consists of single left rarefaction
*
      IMPLICIT NONE
*
C     Declaration of variables
*
      REAL   C,CL,CR,D,DL,DR,GRAVIT,S,SHL,STL,U,UL,UR
*
      COMMON /STATES/ CL, DL, UL, CR, DR, UR
      COMMON /ACCELE/ GRAVIT
*
      SHL = UL - CL
*
      IF(S.LE.SHL)THEN
```

```
*
C         Sampling point lies to the left of the rarefaction
*
          D = DL
          U = UL
       ELSE
*
          STL = UL + 2.0*CL
*
          IF(S.LE.STL)THEN
*
C            Sampling point lies inside the rarefaction
*
             U = (UL + 2.0*CL + 2.0*S)/3.0
             C = (UL + 2.0*CL - S)/3.0
             D = C*C/GRAVIT
          ELSE
*
C            Sampling point lies in right dry-bed state
*
             D = DR
             U = UR
          ENDIF
       ENDIF
*
       END
*
                                    *
*----------------------------------------------------------*
*

SUB
```

Project 2 MATH5350

CHUNG,Chak Pong.

20015116

Problem

Use Riemann solver to code Linearised gas dynamics equation

$$\mathbf{U}_t + \mathbf{A}\mathbf{U}_x = \mathbf{0}\,,$$

with

$$\mathbf{U} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \equiv \begin{bmatrix} \rho \\ u \end{bmatrix}\,, \quad \mathbf{A} = \begin{bmatrix} 0 & \rho_0 \\ a^2/\rho_0 & 0 \end{bmatrix}\,.$$

Finite volume scheme is used in this project:

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x_i}(F^*_{i+1/2} - F^*_{i-1/2})$$

The flux F is A*U,where U is

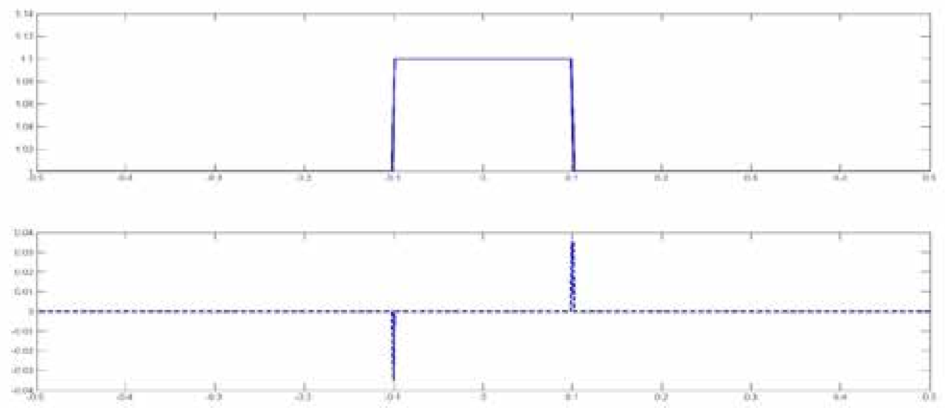
$$\mathbf{U}^* = \begin{bmatrix} \rho^* \\ u^* \end{bmatrix} = \beta_1 \begin{bmatrix} \rho_0 \\ -a \end{bmatrix} + \alpha_2 \begin{bmatrix} \rho_0 \\ a \end{bmatrix}\,.$$

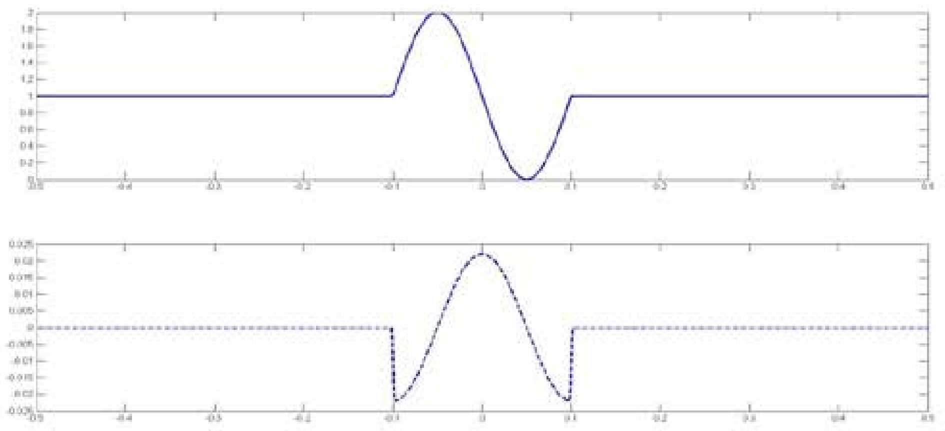
Where beta and alpha are given by

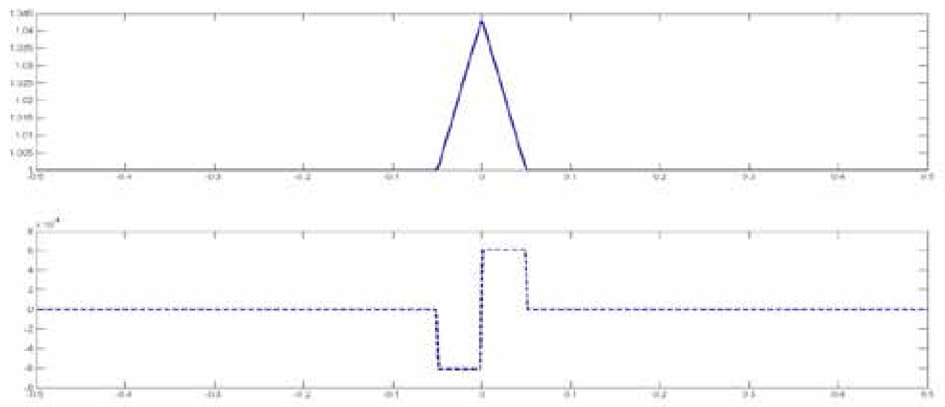
$$\beta_1 = \frac{a\rho_R - \rho_0 u_R}{2a\rho_0}\,, \quad \beta_2 = \frac{a\rho_R + \rho_0 u_R}{2a\rho_0}\,.$$

$$\alpha_1 = \frac{a\rho_L - \rho_0 u_L}{2a\rho_0}\,, \quad \alpha_2 = \frac{a\rho_L + \rho_0 u_L}{2a\rho_0}\,.$$

Three different initial conditions



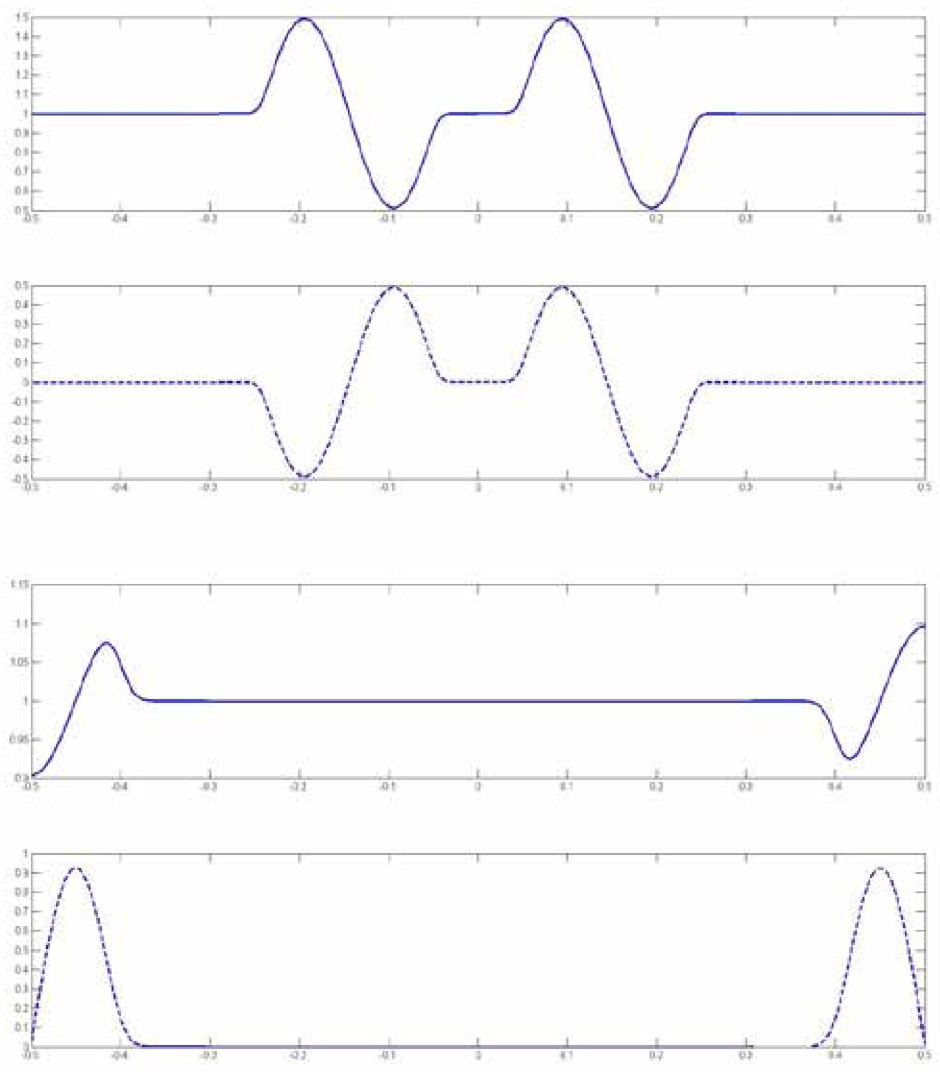




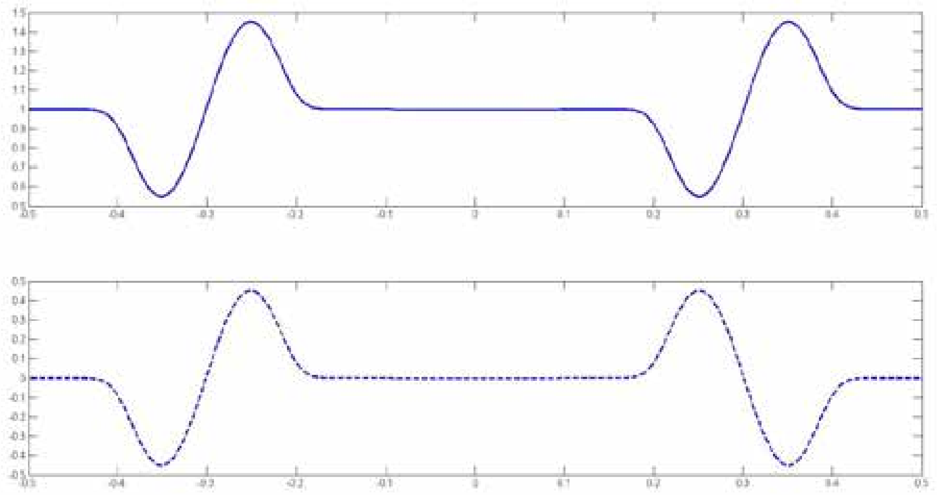
Boundary condition

The difference between **open boundary** and **reflection boundary** lies in the **two far end** of the grid. Only two ghost points are used for this Riemann solver used. But 4 ghost points will be used if the flux depends on neighboring four points. For open boundary, we need to copy the value of the point next to the ghost point to the ghost point. For reflection boundary, we can fill in the ghost point in a way that there is a wave coming in opposite direction.

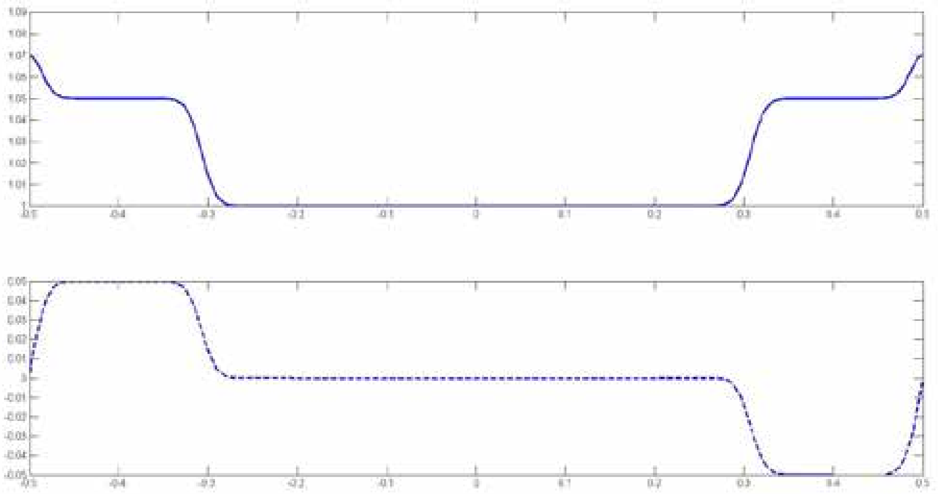
Sine:



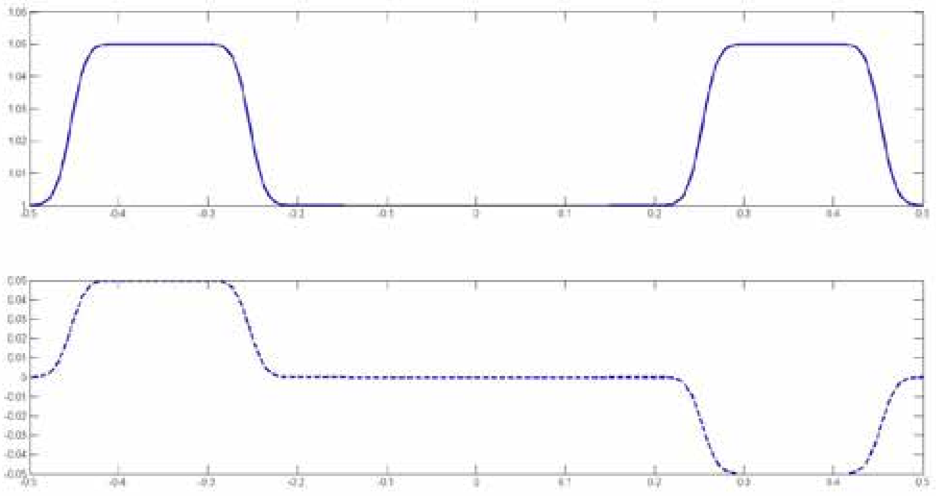
Reflection starts from here:



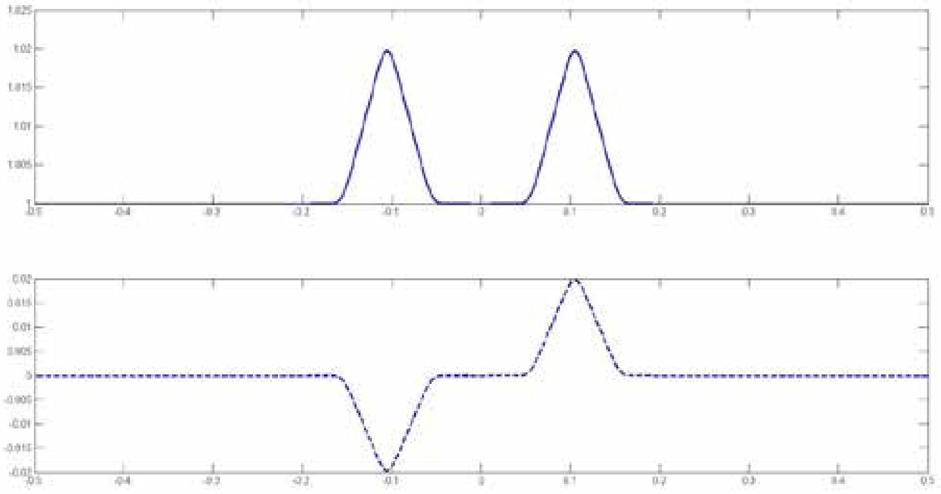
Square (spread into two half square gradually then moving to different end. Below is the graph when approaching to the end and then reflected.)



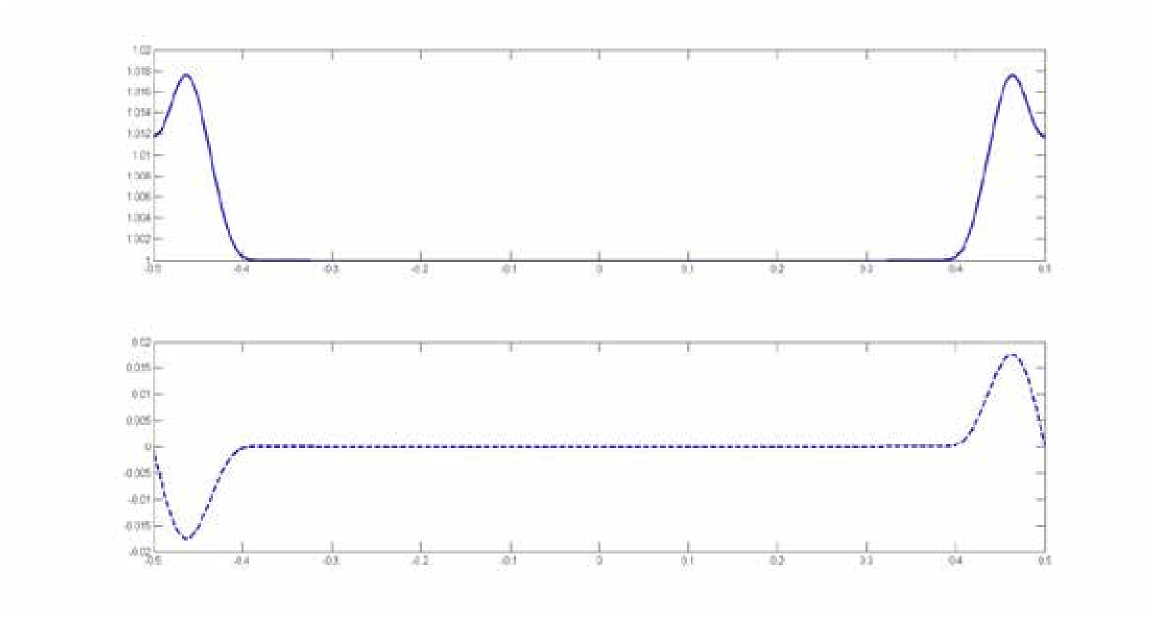
Reflection starts from here:



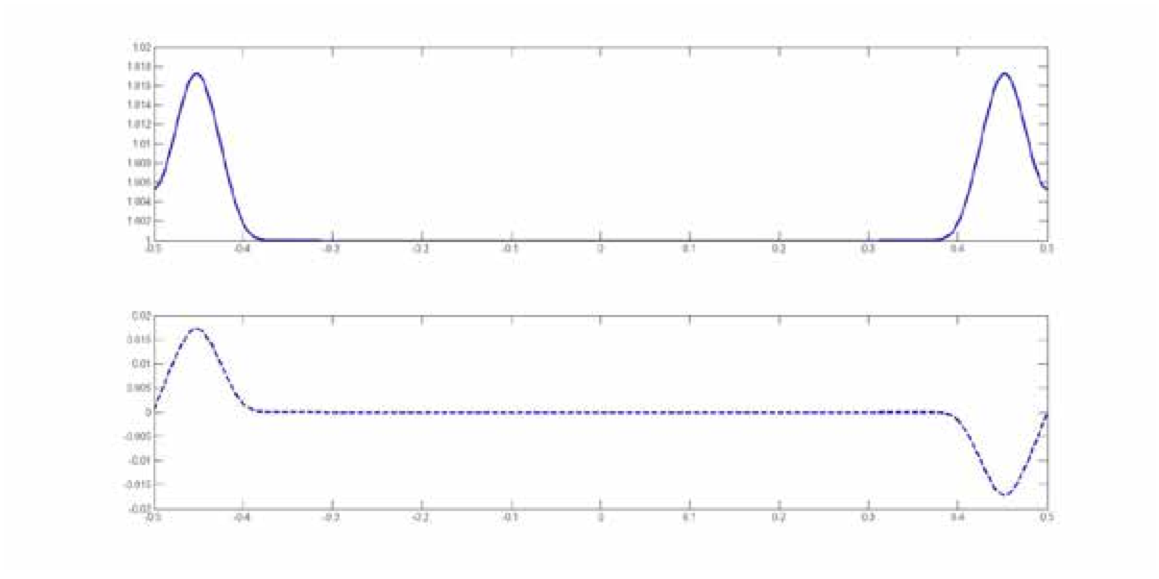
Triangle



Reflection starts from here:



Reflection starts from here:



Code

The **FORTRAN code** is attached in the end

The subroutine for drawing the graph is commented out for easy reading of the main part.

```
module datas
    real(kind=8),parameter :: PI = 4.0*atan(1.0)
    real(kind=8),parameter :: SMV = 1.0E-20
    real(kind=8),parameter :: a= 0
    real(kind=8),allocatable,dimension(:,:) :: w !solution variable
    real(kind=8),allocatable,dimension(:,:) :: flux !flux
    real(kind=8),allocatable,dimension(:) :: x
    real(kind=8),allocatable,dimension(:,:) :: u0
    real(kind=8),allocatable,dimension(:,:) :: u1
    real(kind=8) :: dx !spacing in x-direction
    real(kind=8) :: dt !time step
    real(kind=8) :: cfl !cfl number
    real(kind=8) :: lambda
    real(kind=8) :: t
    integer :: iter !iterations
end module datas

module solver
    contains
    function Riemann(ul,ur)
    real(kind=8),dimension(2) ::ul
    real(kind=8),dimension(2) ::ur
    real(kind=8) ::a=1.0
    real(kind=8) ::rho0=1.0
    real(kind=8) ::alpha2
    real(kind=8) ::beta1
    real(kind=8) ::u1
    real(kind=8) ::u2
    real(kind=8) ,dimension(2) ::Riemann

    alpha2=(a*ul(1)+rho0*ul(2))/(2*a*rho0)
    beta1=(a*ur(1)-rho0*ur(2))/(2*a*rho0)

    u1=beta1*rho0+alpha2*rho0
    u2=beta1*(-a)+alpha2*a
    Riemann(1)=rho0*u2
    Riemann(2)=a*a/(rho0)*u1
end function Riemann
end module solver

program main
    use datas
    use solver
    integer :: i,itmax,t_F,bc,flg
    real :: xmin,xmax
    integer :: m,n

    cfl = 0.7

    !geometry
    bc=2
    n =1000
    flg=1
    itmax=100
    m=1
    xmax=0.5
    xmin=-0.5
    t_F=1;bc=2;flg=1
    dx=(xmax-xmin)/(n-1)


    allocate(x(n))
    allocate(u0(n+2*m,2))
    allocate(u1(n+2*m,2))

    ! I.C.
open(unit=10,file="out.dat")
if (flg==1) then

     do i=1,n

        x(i)=-0.5+(i-1)*dx
      if (x(i)<-0.1) then
        u0(m+i,1)=0
        u0(m+i,2)=0
        elseif (x(i)<=0.1) then
            u0(m+i,1)=0
            u0(m+i,1)=0.10  !! wanring!
            u0(m+i,2)=0

        else
            u0(m+i,1)=0
            u0(m+i,2)=0
        endif
        !write(10,*) i, u0(m+i,1),u0(m+i,2)
        !write(*,*) "i,u0(m+i,1),u1(m+i,2)", i, u0(m+i,1),u0(m+i,2)
    end do
endif

! B.C.
if (bc==2) then
do i=1,m

    u0(i,1)=u0(2*m+1-i,1)
    u0(i,2)=u0(2*m+1-i,2)

    u0(m+n+i,1)= u0(m+n+1-i,1);
    u0(m+n+i,2)=-u0(m+n+1-i,2);

    enddo
endif

t=0;
dt=0.001
lambda=dt/dx    ! for constant dt,put it outside the loop

do while(t+dt<=t_F .or. it<=itmax)
```

```
      t=t+dt

    do i=1,n
        u1(m+i,:)=u0(m+i,:)-lambda*(Riemann(u0(m+i,:),u0(m+i+1,:))-Rieman
        n(u0(m+i-1,:),u0(m+i,:)))

        !write(10,*) i, u1(m+i,1),u1(m+i,2)

    enddo

if (bc==2) then
do i=1,m

    u1(i,1)=u1(2*m+1-i,1)
    u1(i,2)=u1(2*m+1-i,2)

    u1(m+n+i,1)= u1(m+n+1-i,1);
    u1(m+n+i,2)=-u1(m+n+1-i,2);

    enddo
endif

u0(:,:)=u1(:,:)

enddo

end program main


!subroutine timestep()
!    use datas
!    integer :: i
!    real(kind=8) :: umax
!
!    umax = 0.0
!    do i=1,num
!        umax=max(umax,w(i))
!    end do
!    dt = cfl*dx/umax
!end subroutine timestep
!
!subroutine calc_flux()
!    use datas
!    integer :: i
!
!    !boundary
!    flux(1) = 0.5*w(i)**2
!    flux(num+1) = 0.5*w(num)**2
!
!    !inner
!    do i=2,num
!        if (w(i-1)>=w(i)) then !form a shock
!            if (0<0.5*(w(i-1)+w(i))) then
!                flux(i) = 0.5*w(i-1)**2
!            else
!                flux(i) = 0.5*w(i)**2
!            end if
!        else !form a rarefaction wave
!            if (0<w(i-1)) then
!                flux(i) = 0.5*w(i-1)**2
!            else if (0>w(i)) then
!                flux(i) = 0.5*w(i)**2
!            else
!                flux(i) = 0.0
!            end if
!        end if
!    end do
!end subroutine calc_flux
!
!subroutine update()
!    use datas
!    integer :: i
!
!    do i=1,num
!        w(i) = w(i)+(flux(i)-flux(i+1))*dt/dx
!    end do
!end subroutine update
!
!subroutine writeout()
!    use datas
!    integer :: i
!
!    open(unit=10,file="out.dat")
!    do i=1,num
!        xpos = (-1.0+dx/2.0)+(4.0-dx)*(i-1.0)/(num-1.0+SMV)
!        write(10,*) xpos,w(i)
!    end do
!end subroutine writeout
```

MATH5350

CHUNG,Chak Pong

SID:20015116

The project is to solve the burgers Equation：

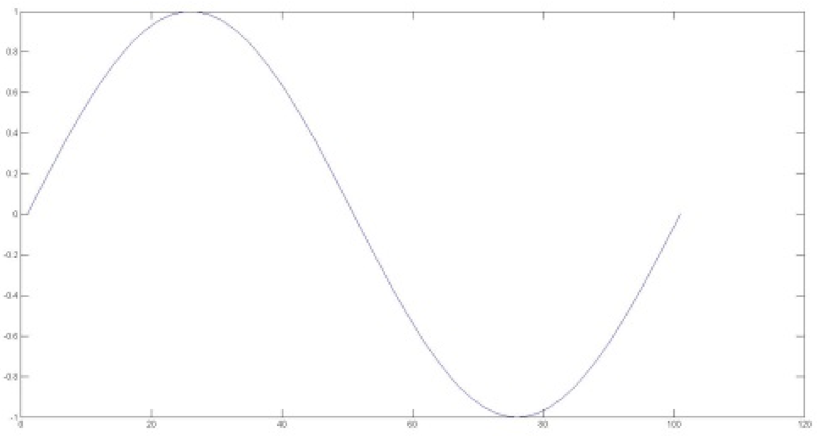
$$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial}{\partial x}(u^2) = 0.$$

Using the final volume method:

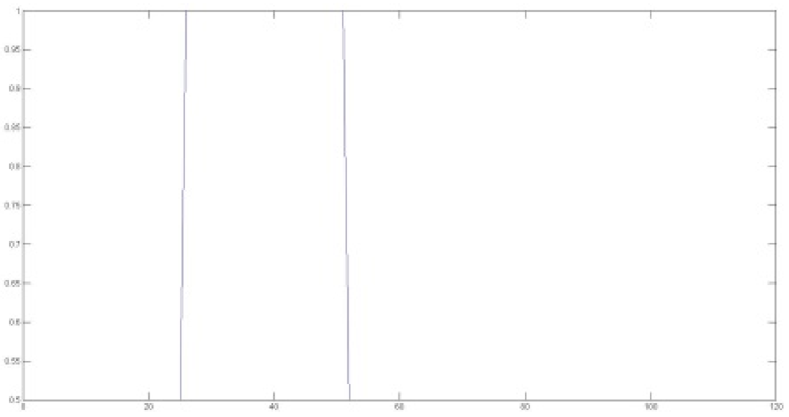
$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}\left(F_{i+1/2}^n - F_{i-1/2}^n\right),$$

With Initial condition:

1.Sine wave 2.Square Pulse

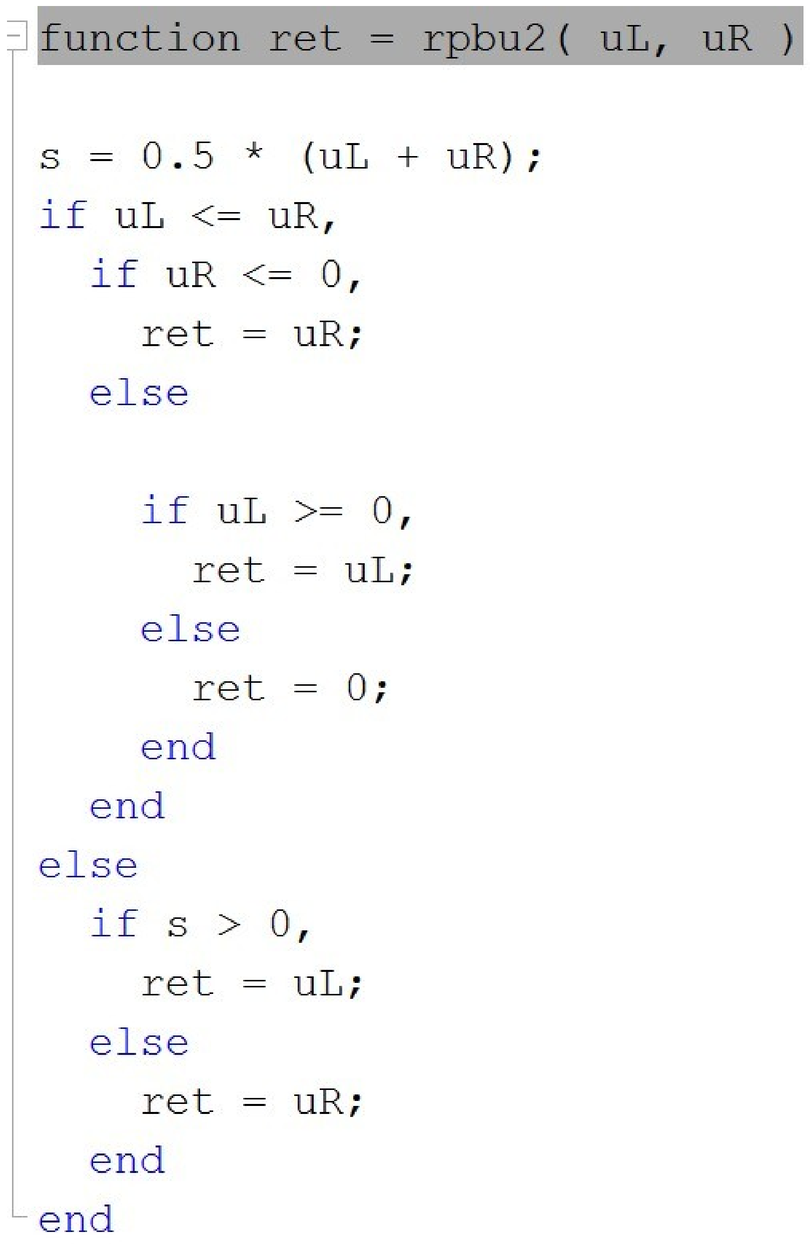


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
20
40
60
80
100
120



1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0
20
40
60
80
100
120

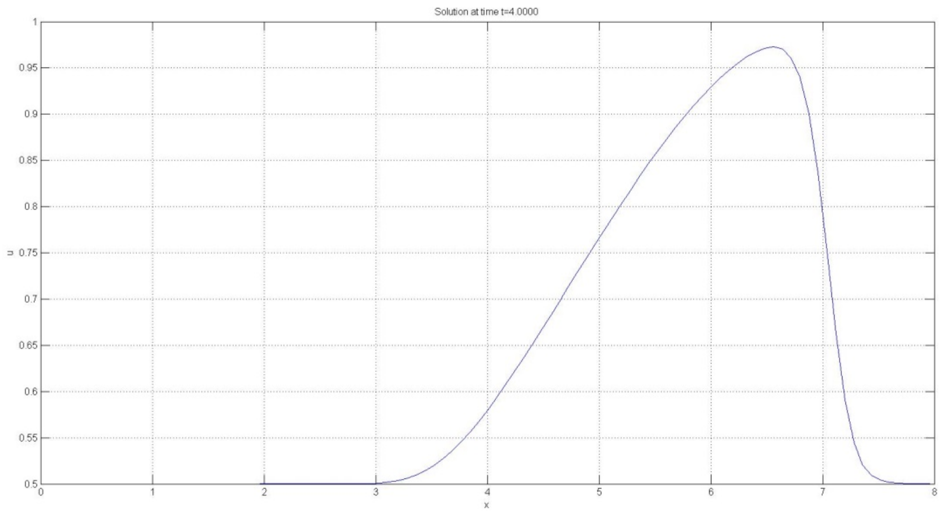
Flux function:

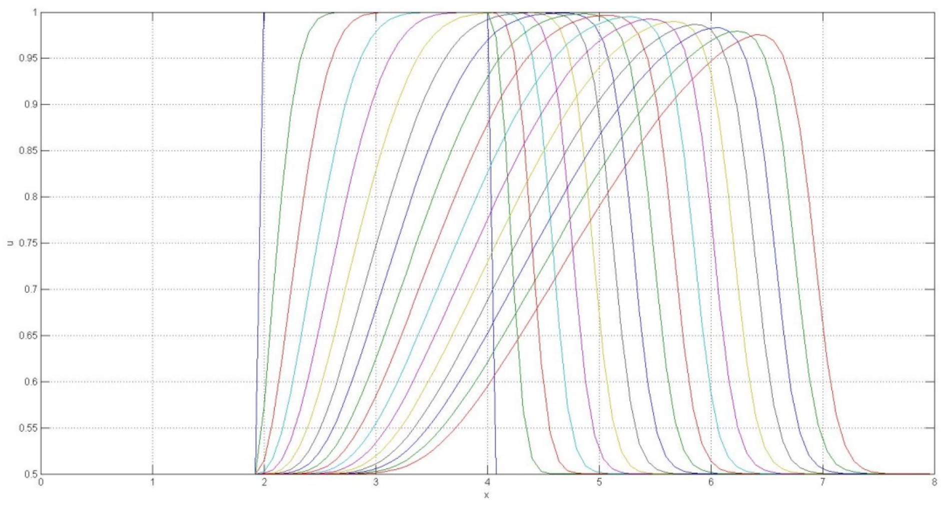


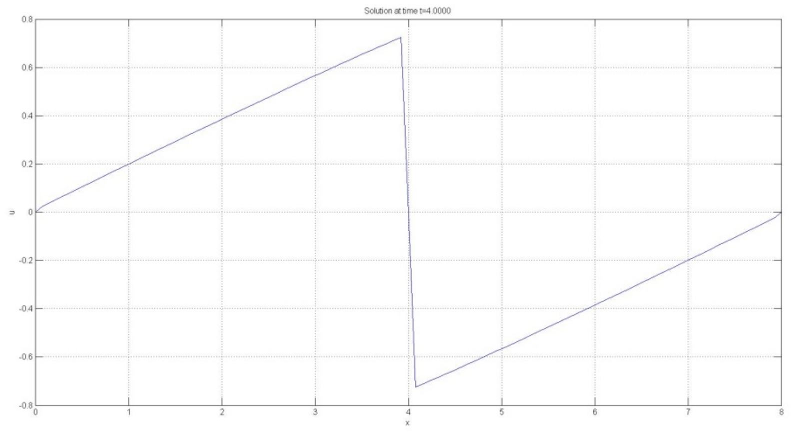
```
function ret = rpbu2( uL, uR )

s = 0.5 * (uL + uR);
if uL <= uR,
  if uR <= 0,
    ret = uR;
  else

    if uL >= 0,
      ret = uL;
    else
      ret = 0;
    end
  end
else
  if s > 0,
    ret = uL;
  else
    ret = uR;
  end
end
```

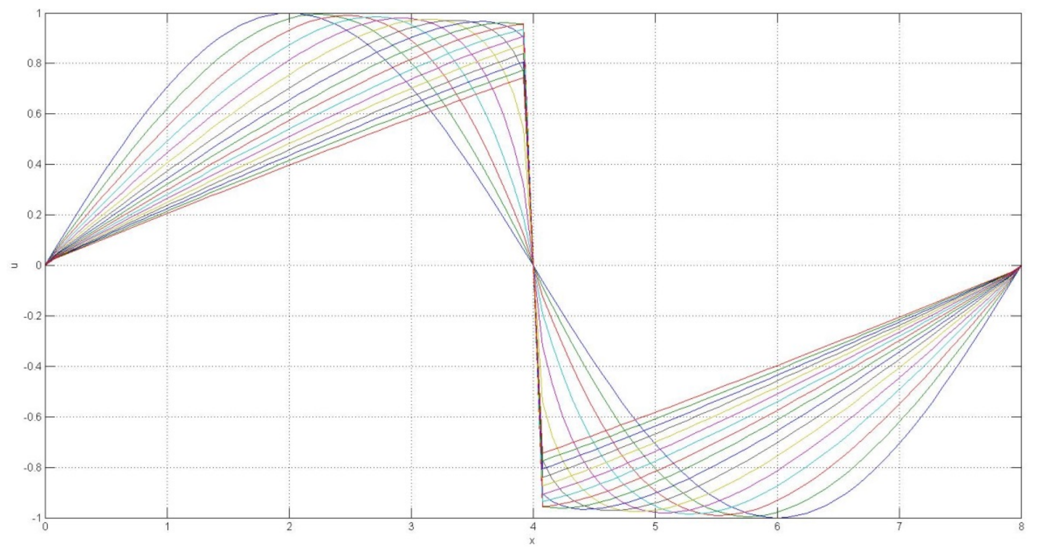
Result:

For the two schemes of flux,the result for the square wave initial condition is the same,which is shown below






Solution at time t=4.0000
u
x
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
1
2
3
4
5
6
7
8


u
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6
7
8

The matlab code is shown below:

```
clear;

nx    = 100;
 dt    = 0.01;
 ictype= 5;     % 1 = shock; 2 = expansion;
                                  % 3 = sonic expansion; 4 = square pulse;5 =
sine

 tend = 4;     % end time
xmax = 8;     % domain length [0,xmax]
dx = xmax/nx; % mesh spacing (constant)
x  = [0 : dx : xmax];
nt = floor(tend/dt);
dt = tend / nt;
ntprint = 50; % for printing

u0   = uinit(x,ictype);
u    = u0;
unew = 0*u;
us   = unew(1:end-1);

disp( ['   dx = ', num2str(dx)] );
disp( ['   dt = ', num2str(dt)] );

ntprint = min(nt, ntprint);
dtprint = tend / ntprint;

uall = zeros(ntprint+1,nx+1);

uall(1,:) = u0;

ip = 1;
figure(1)
for i = 1 : nt,
  t = i*dt;

  us = rpbu2(u(1:end-1), u(2:end));
  unew(2:end-1) = u(2:end-1) + dt/dx * (f(us(1:end-1)) - f(us(2:end)));
  unew(1)   = u(1);
  unew(end) = u(end);

  % Plot the solution profiles.
  if t >= ip*dtprint,
    plot(x, unew)
    xlabel('x'), ylabel('u')
    title( ['Solution at time t=', num2str(t,'%9.4f')] )
    grid on, shg
    pause(0.1)
    ip = ip + 1;

    uall(ip,:) = unew;
  end
  u = unew;
end

figure(2)
nskip = 3;
plot(x,uall(1:nskip:end,:));
xlabel('x'), ylabel('u')
grid on, shg
```