AEROSPACE ENGINEERING CFD – PROJECT 4

2D Euler Equation- A Computational Solution

Gandharv Kashinath

Iowa State University, Ames, IA

12/7/2007

In this project a numerical solution to the 2D Euler equation has been developed and implemented on a 2D mesh to solve a supersonic flow around a blunt body.

Introduction

In this project, an attempt was made to solve the 2-D Euler equation around a blunt body in a supersonic flow. The governing equation of the flow was stated as follows;

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0$$

Where;

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix}, F = \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ u(E+P) \end{bmatrix}, G = \begin{bmatrix} \rho v \\ \rho uv \\ \rho u^2 + P \\ v(E+P) \end{bmatrix}$$

$$E = \frac{P}{\gamma - 1} + \frac{1}{2}\rho(u^2 + v^2)$$

The above equations were solved using two numerical schemes namely;

- Second order Runge-Kutta (TVD) in time and first order in space.
- Second order Runge-Kutta (TVD) in time and second order in space.

The inflow conditions of the free stream were given as follows;

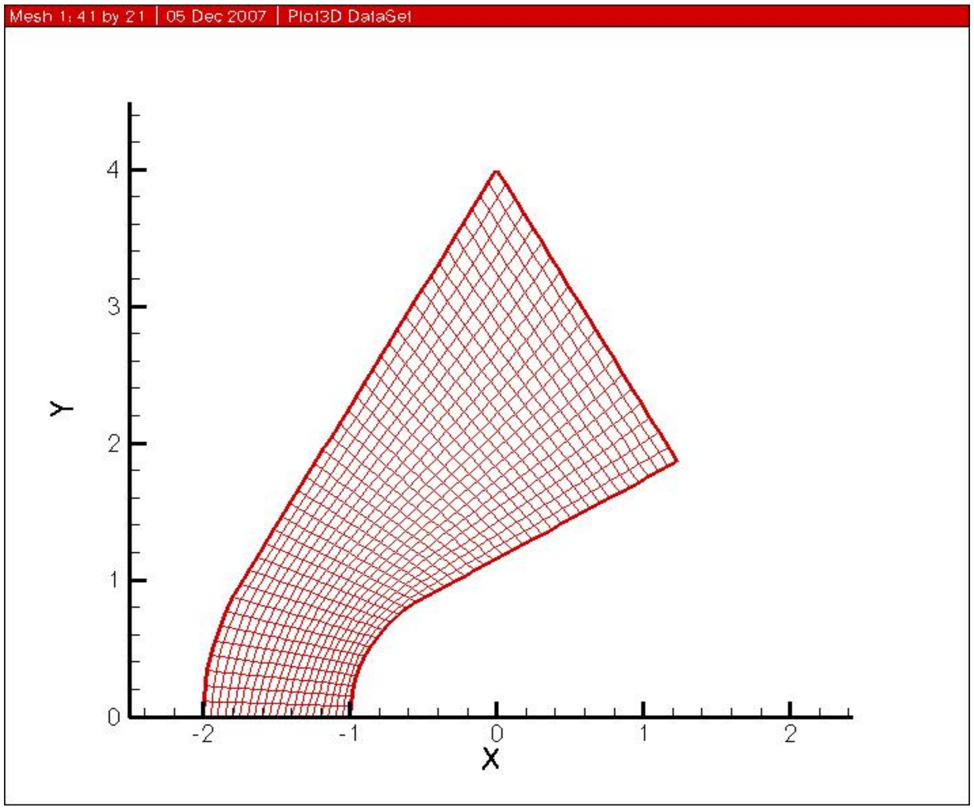
- Ratio of specific heats, $\gamma = 1.4$
- Pressure, $P = 1$
- Density, $\rho = 1.4$
- X – velocity, $u = 4$
- Y – velocity, $v = 0$

In order to solve this problem, two meshes with dimensions 41X21 and 81X41, with the following boundary conditions were used;

- At ξ min boundary, symmetry
- At ξ max boundary, extrapolation
- At η min boundary, wall
- At η max boundary, fix inflow

Note: ξ is the radial direction and η is the stream-wise direction.

Presented in the following page are the two meshes used to solve the 2D problem.



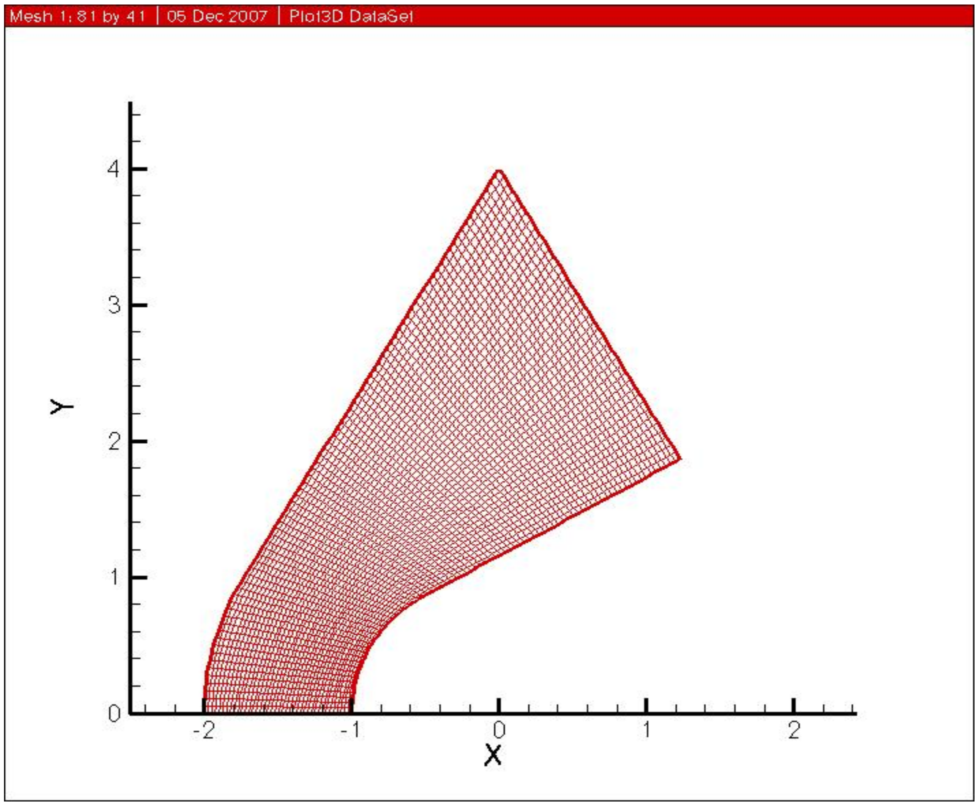
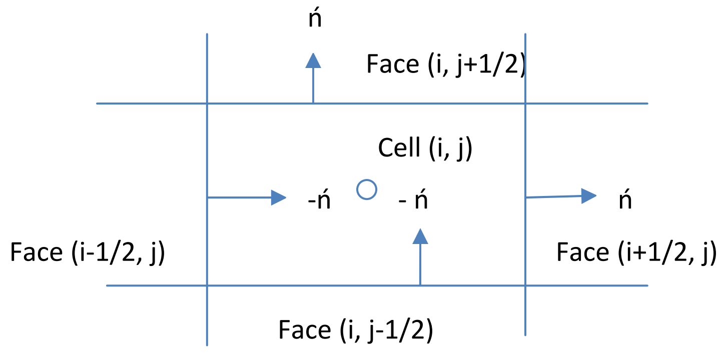


Figure 1: Mesh 1 (41 by 21), Mesh 2 (81 by 41) used for computation

Approach

In order to solve the above 2D Euler equation, it was firstly discretized into a cell centered, finite volume formulation as follows;



Finite Volume Representation

Note: The direction of the normal on the 2 faces has been reversed to modify the governing equations for computational simplicity.

Governing Differential Equation:

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0$$

Apply: Gauss Divergence Theorem

Define: Flux = $\vec{f}.\vec{n} = Fn_x + Gn_y = f_n$

Governing Finite Volume Equation:

$$\frac{\partial \overline{Q_{i,j}}}{\partial t} V_{i,j} + \left(f_{i+\frac{1}{2},j} S_{i+\frac{1}{2},j}\right) - \left(f_{i-\frac{1}{2},j} S_{i-\frac{1}{2},j}\right) + \left(f_{i,j+\frac{1}{2}} S_{i,j+\frac{1}{2}}\right) - \left(f_{i,j-\frac{1}{2}} S_{i,j-\frac{1}{2}}\right) = 0$$

where; V – Volume of the Cell (Area in 2D), S – Area of the face of the Cell (Length in 2D)

From the above representation, the flux f is defined by the Rusonov Flux which is dependent on the average Q values of the left and right cells and is defined by the following equations;

$$f_{i+\frac{1}{2},j} = \tilde{f}\left(Q^L_{i+\frac{1}{2},j}, Q^R_{i+\frac{1}{2},j}, \acute{n}\right) = \frac{1}{2}[F_n(Q^L) + F_n(Q^R) - \lambda\,(Q^R - Q^L)]$$

where;

$$\lambda = \frac{1}{2}\left(|v_{n_L}| + |v_{n_R}| + c_L + c_R\right), \qquad v_n = un_x + vn_y, \qquad c = \sqrt{\gamma \frac{P}{\rho}}$$

$$F_n(Q) = \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ u(E+P) \end{bmatrix} n_x + \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ v(E+P) \end{bmatrix} n_y = \begin{bmatrix} \rho v_n \\ \rho u v_n + P n_x \\ \rho v v_n + P n_y \\ v_n(E+P) \end{bmatrix}$$

Now in order to compute the Rusonov Flux we need the cell average $Q_{i+1/2,j}$. In order to represent this cell average we consider two schemes and the reconstruction as follows;

❖ **First Order Scheme:**
- ➢ $Q^L_{i+\frac{1}{2},j} = Q_{i,j}$
- ➢ $Q^R_{i+\frac{1}{2},j} = Q_{i+1,j}$

❖ **Second Order Scheme: (Linear Reconstruction)**
- ➢ $Q^L_{i+\frac{1}{2},j} = Q_{i,j} + \frac{1}{2} minmod\left(Q_{i+1,j} - Q_{i,j}, Q_{i,j} - Q_{i-1,j}\right)$
- ➢ $Q^R_{i+\frac{1}{2},j} = Q_{i,j} - \frac{1}{2} minmod\left(Q_{i+1,j} - Q_{i,j}, Q_{i,j} - Q_{i-1,j}\right)$

$$minmod(x,y) = \begin{cases} 0 & x.y < 0 \\ sign(x)\min(|x|,|y|) & otherwise \end{cases}$$

Now in order complete solve the above equation the boundary conditions need to be considered and incorporated in the above formulation.

From the given boundary conditions the following equations can be developed to solve for the four boundaries in order to complete the formulation;

❖ **At ξ=1 face, Symmetry**

Along this boundary the v-component of velocity, v=0, ∂u/∂y = 0, ∂P/∂y = 0, ∂ρ/∂y =0. With these conditions the following equations for the cells along this boundary can be constructed;

- ➢ **First Order Scheme**

  $Q^R_{1-\frac{1}{2},j} = Q_{1,j}$
- ➢ **Second Order Scheme**

  $Q^R_{1-\frac{1}{2},j} = Q_{1,j} - \frac{1}{2}\left(Q_{2,j} - Q_{1,j}\right)$

For both these schemes the Q value on the left side of this face is given by;

$$Q^L = \begin{Bmatrix} \rho^L \\ \rho u^L \\ \rho v^L \\ E^L \end{Bmatrix} = \begin{Bmatrix} \rho^R \\ \rho u^R \\ -\rho v^R \\ E^R \end{Bmatrix}$$

With this Q^L, Q^R values the Rusonov Flux can be computed for this boundary face.

- **At ξ=ID+1 face, Supersonic Exit (Where ID is the number of cells in the ξ direction)**

  Along this boundary the values of Q are extrapolated from the interior cells and the following equations for the cells along this boundary can be constructed;

  - **First Order Scheme**

    $Q^L_{ID+\frac{1}{2},j} = Q_{ID,j}$

  - **Second Order Scheme**

    $$Q^L_{ID+\frac{1}{2},j} = Q_{ID,j} - \frac{1}{2}\left(Q_{ID,j} - Q_{ID-1,j}\right)$$

  For both these schemes the Q value on the right face is set equal to the Q value on the left face as calculated above. Again, with this Q^L, Q^R values the Rusonov Flux can be computed for this boundary face.

- **At η=JD+1 face, Supersonic Inlet (Where JD is the number of cells in the η direction)**

  Along this boundary the values of Q are set to the free stream inflow conditions and the following equations for the cells along this boundary can be constructed;

  - **First Order Scheme**

    $Q^L_{i,JD+\frac{1}{2}} = Q_{i,JD}$

  - **Second Order Scheme**

    $$Q^L_{i,JD+\frac{1}{2}} = Q_{i,JD} - \frac{1}{2}\left(Q_{i,JD} - Q_{i,JD-1}\right)$$

  For both these schemes the Q value on the right face is set equal to the inflow conditions. With this Q^L, Q^R values the Rusonov Flux can be computed for this boundary face.

- **At η=1 face, Wall Conditions**

  Along this boundary for every differential length of the boundary the quantity (v.ń) = 0 and with this assumption the following equations for the cells along this boundary can be constructed;

  - **First Order Scheme**

    $Q^R_{i,1-\frac{1}{2}} = Q_{i,1}$

- **Second Order Scheme**

$$Q^{L}_{i,1-\frac{1}{2}} = Q_{i,1} - \frac{1}{2}\left(Q_{i,2} - Q_{i,1}\right)$$

For both these schemes the Q value on the left side of this face is given by;

$$Q^{L} = \begin{Bmatrix} \rho^{L} \\ \rho u^{L} \\ \rho v^{L} \\ E^{L} \end{Bmatrix} = \begin{Bmatrix} \rho^{R} \\ \rho(u^{R} - 2(\overrightarrow{v_R}.\vec{n})n_x \\ \rho(v^{R} - 2(\overrightarrow{v_R}.\vec{n})n_y \\ E^{R} \end{Bmatrix}$$

With this Q^L, Q^R values the Rusonov Flux can be computed for this boundary face.

The Residual function is defined from the numerical governing equation and with the help of the above constructed equations the Residual is calculated at each cell.

$$\frac{\partial \overline{Q_{i,j}}}{\partial t} V_{i,j} + \left(f_{i+\frac{1}{2},j} S_{i+\frac{1}{2},j}\right) - \left(f_{i-\frac{1}{2},j} S_{i-\frac{1}{2},j}\right) + \left(f_{i,j+\frac{1}{2}} S_{i,j+\frac{1}{2}}\right) - \left(f_{i,j-\frac{1}{2}} S_{i,j-\frac{1}{2}}\right) = R_{i,j}(Q) = Residual\ fn.$$

Further a time integration process is initiated and this procedure is carried out using the 2 stage Runge-Kutta TVD scheme which is presented below;

$$Q^{*}_{i,j} = Q^{n}_{i,j} + \Delta t_{i,j} R_{i,j}(Q^{n})$$

$$Q^{n+1}_{i,j} = \frac{1}{2}[Q^{n}_{i,j} + Q^{*}_{i,j} + \Delta t_{i,j} R_{i,j}(Q^{*})]$$

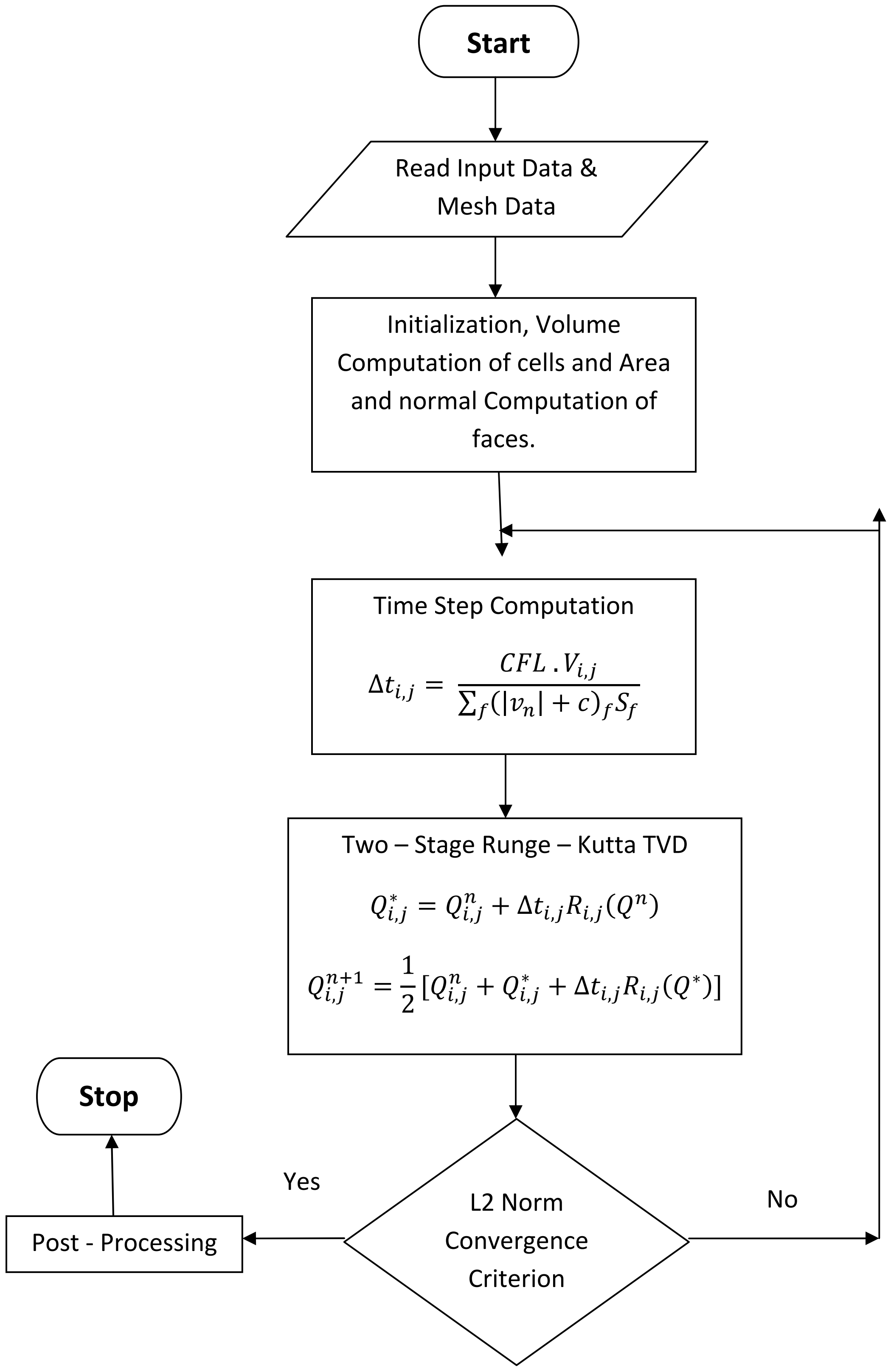
In the above equation the time step $\Delta t_{i,j}$ is computed for each cell and is given by;

$$\Delta t_{i,j} = \frac{CFL\ .V_{i,j}}{\sum_f (|v_n| + c)_f S_f}$$

where f stands for each face of cell.

The equations thus constructed was coded and checked for convergence using the L2 Norm density residual. Simulation was designed to output pressure and Mach number contours for each of the Numerical schemes. In the following sections; a flowchart describing the code structure and results are presented.

Flow Chart used for Program Construction


Start
Read Input Data & Mesh Data
Initialization, Volume Computation of cells and Area and normal Computation of faces.
Time Step Computation
$\Delta t_{i,j} = \frac{CFL\,.V_{i,j}}{\sum_f (|v_n| + c)_f S_f}$
Two – Stage Runge – Kutta TVD
$Q_{i,j}^{*} = Q_{i,j}^{n} + \Delta t_{i,j} R_{i,j}(Q^n)$
$Q_{i,j}^{n+1} = \frac{1}{2}[Q_{i,j}^{n} + Q_{i,j}^{*} + \Delta t_{i,j} R_{i,j}(Q^*)]$
L2 Norm Convergence Criterion
Yes
No
Post - Processing
Stop

Results

Presented below are the Pressure and Mach Number Contours for the two schemes namely the first order and the second order schemes;

- First Order Scheme:
  - Pressure Plots

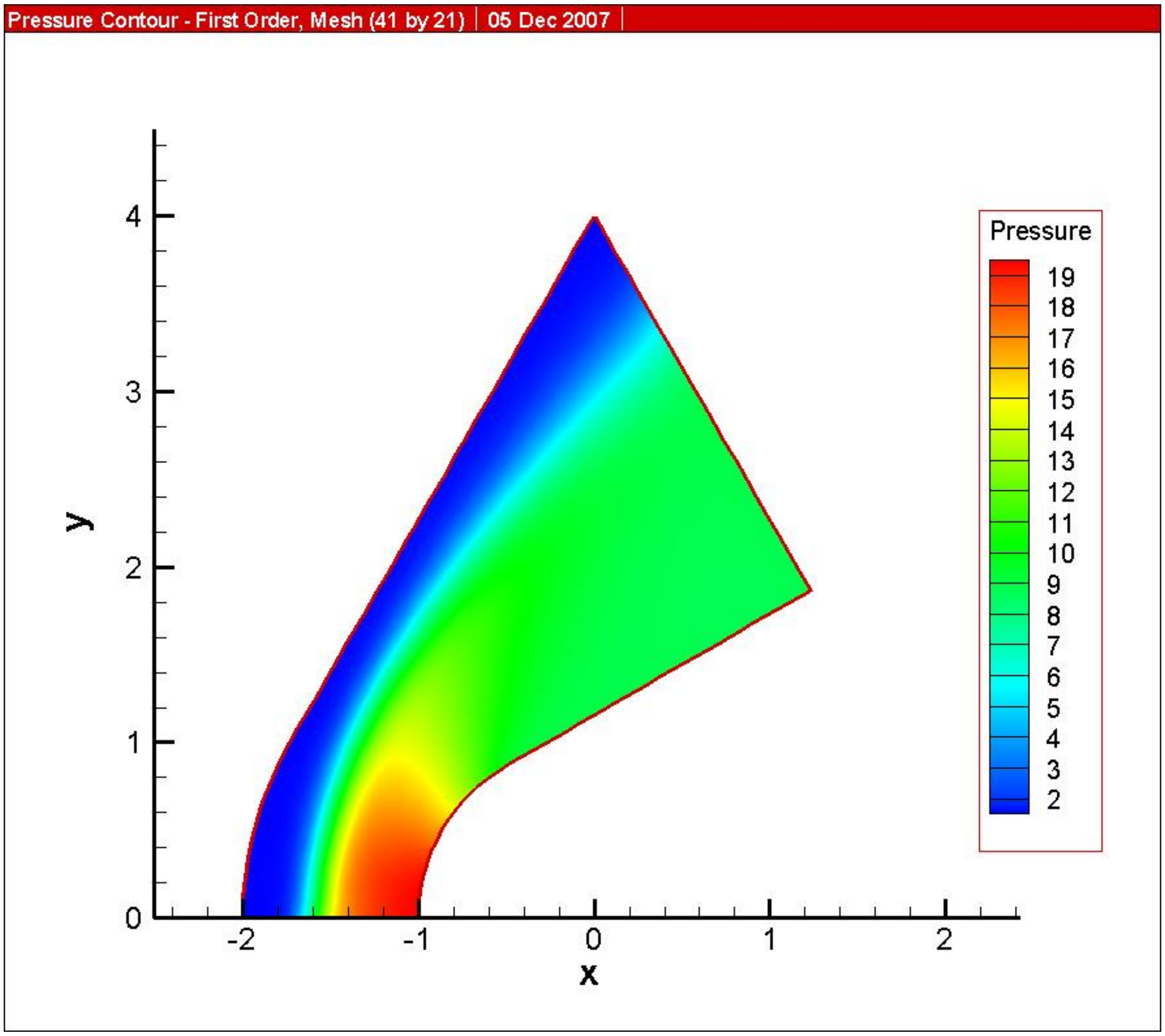


Figure 2: Pressure Contour for a mesh 41 by 21, first order scheme

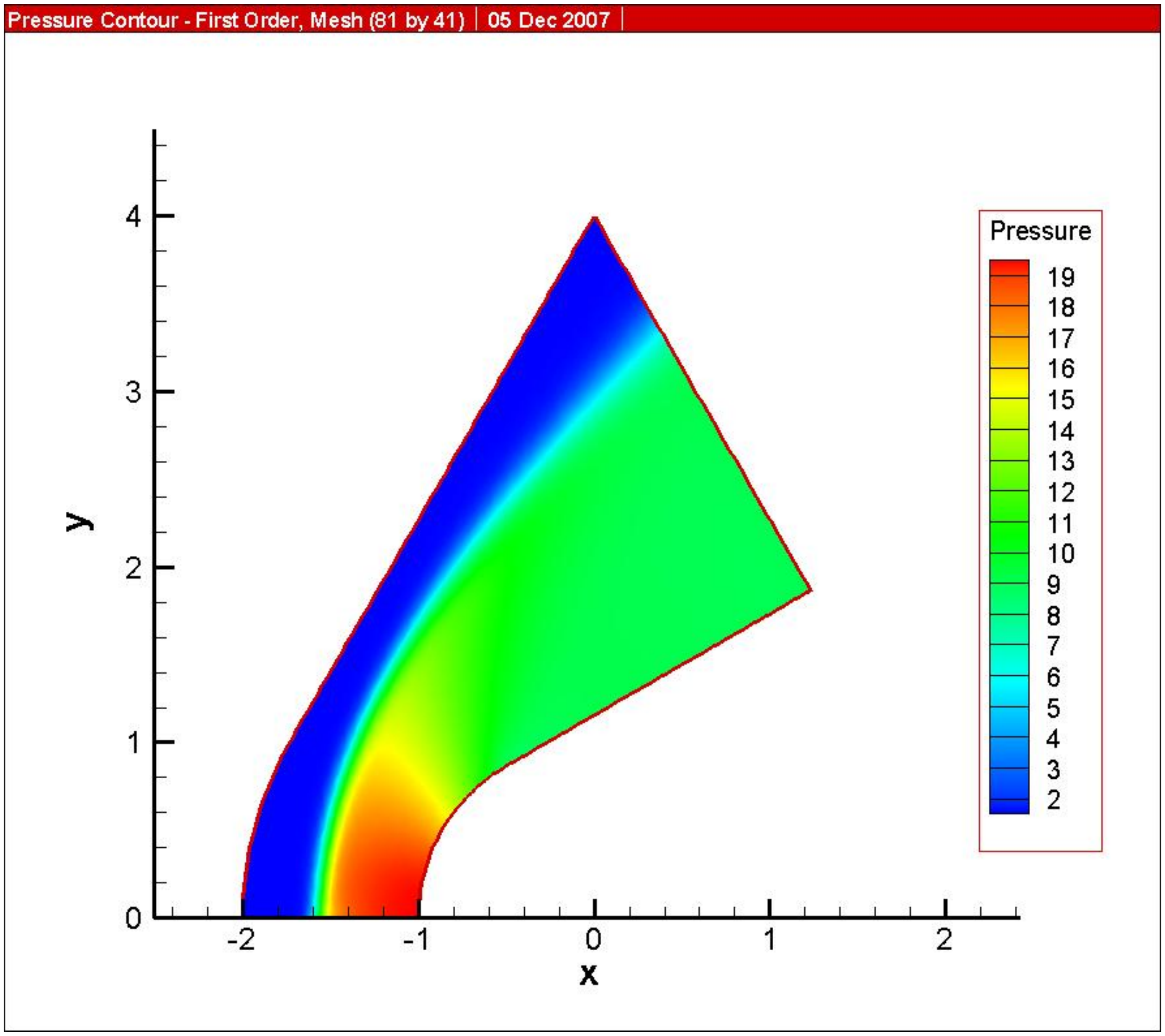


Figure 3: Pressure Contour for a mesh 81 by 41, first order scheme

- Mach Number Plots

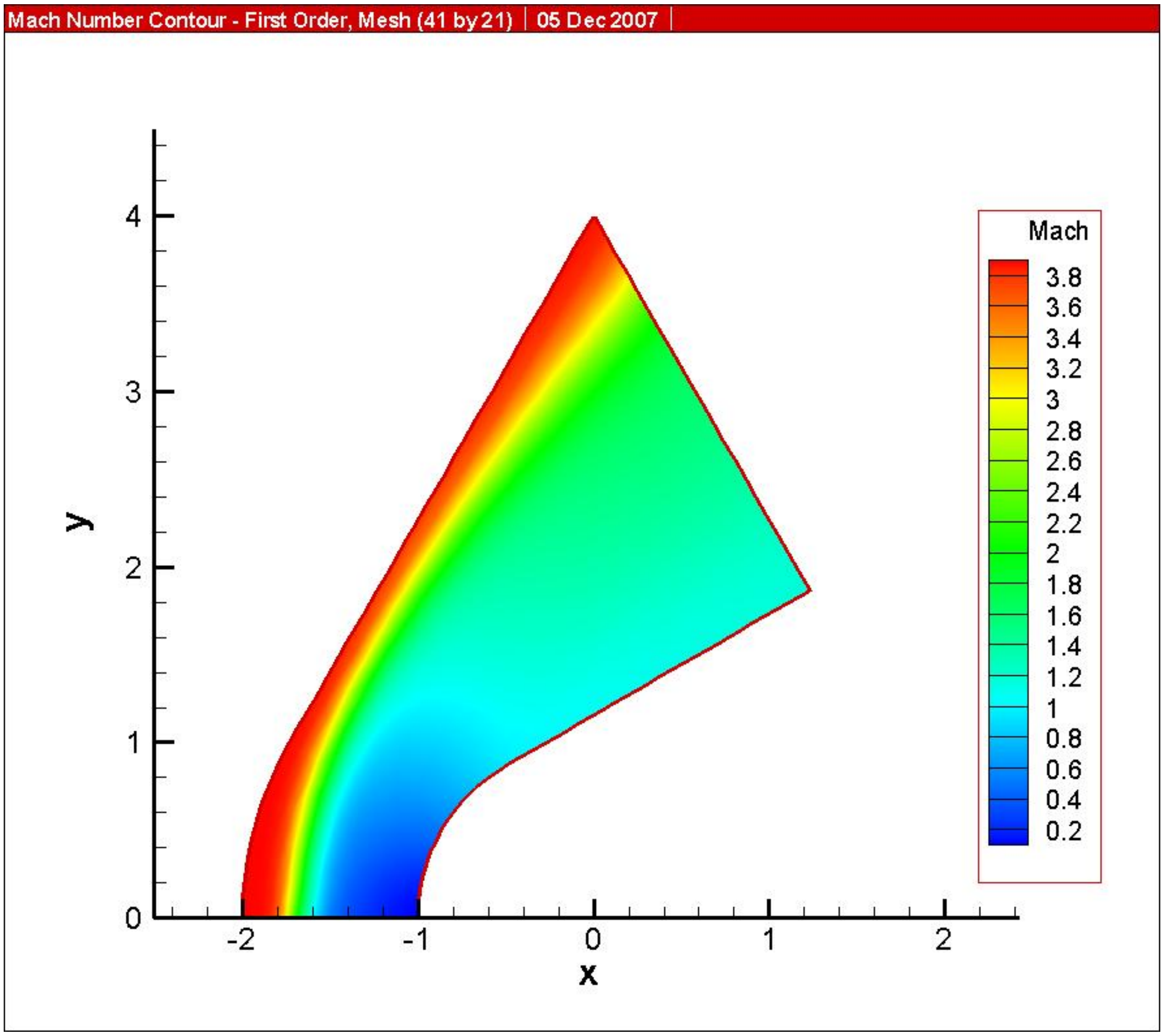


Figure 4: Mach Number Contour for a mesh 41 by 21, first order scheme

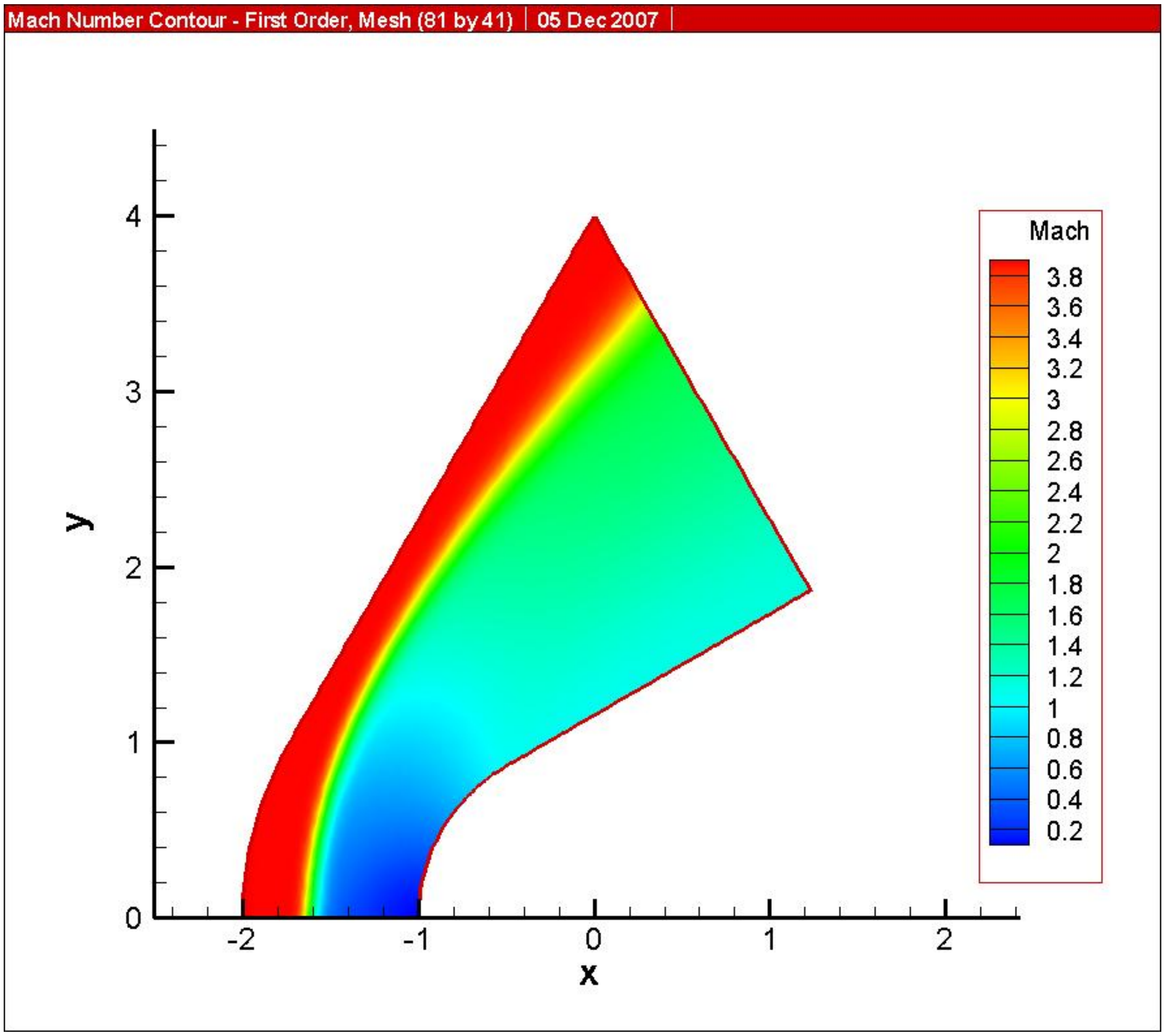


Figure 5: Mach number Contour for a mesh 81 by 41, first order Scheme

Second Order Scheme:

- Pressure Plots

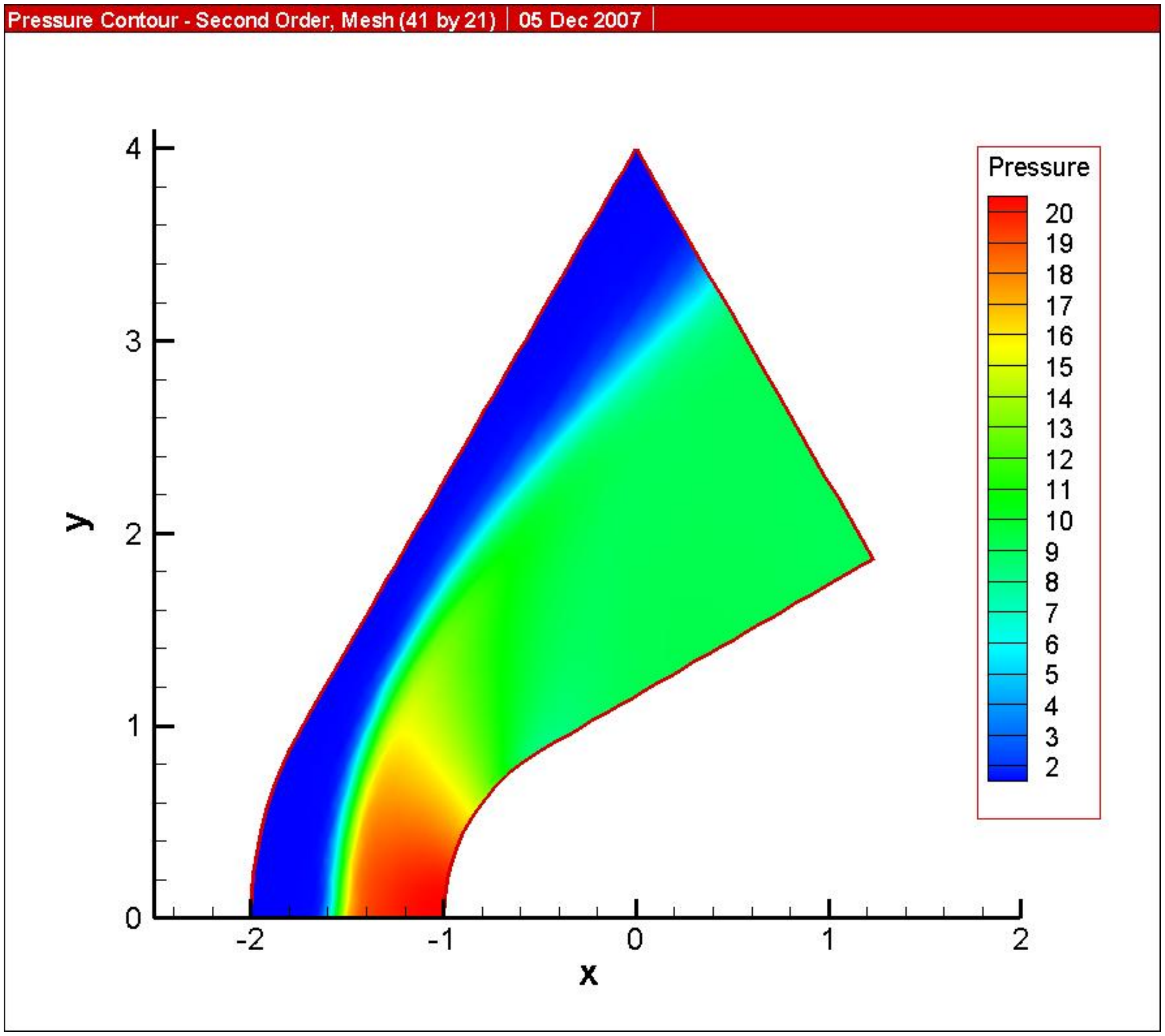


Figure 6: Pressure Contour for a mesh 41 by 21, second order scheme

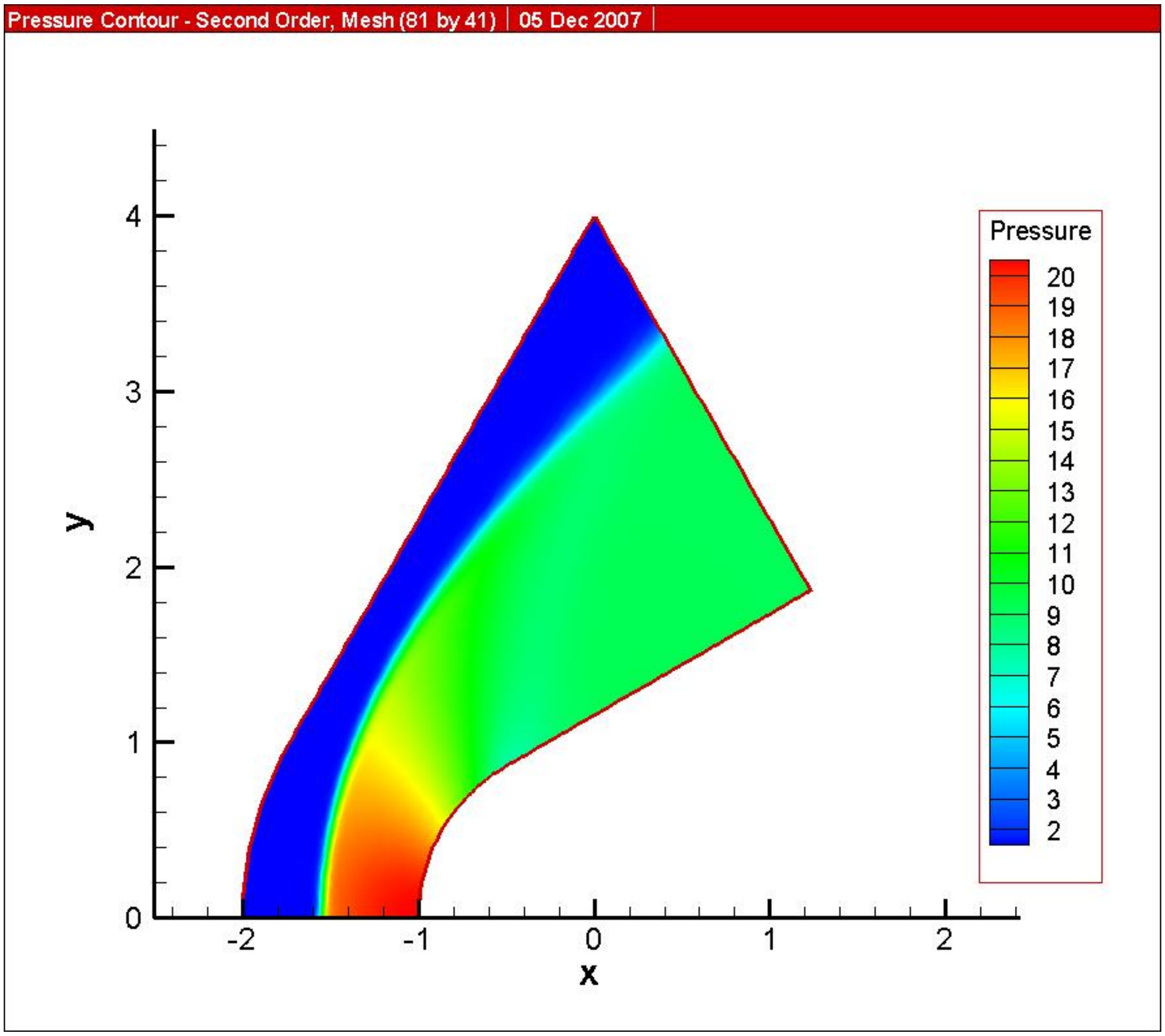


Figure 7: Pressure Contour for a mesh 81 by 41, second order scheme

- Mach Number Plots

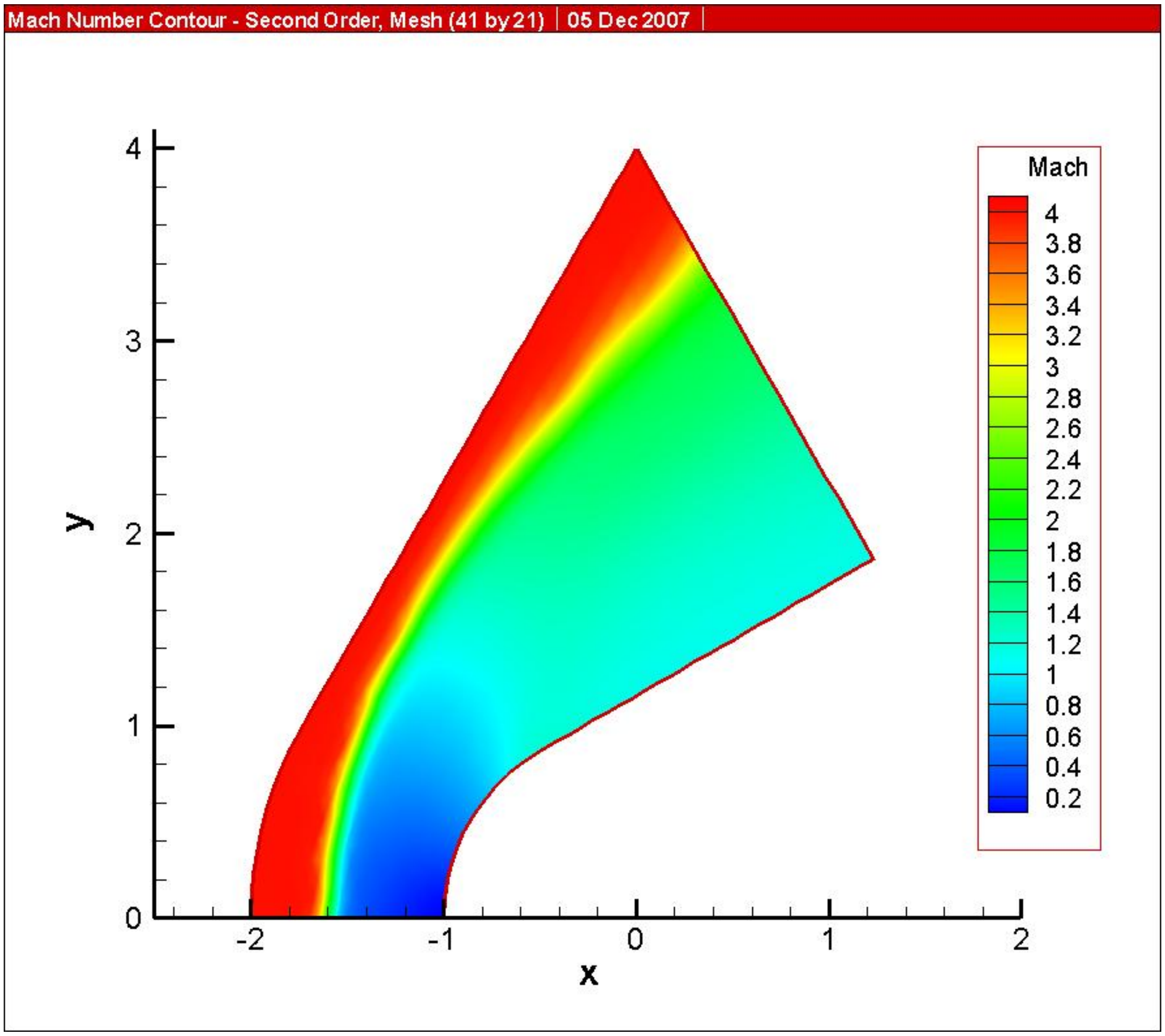


Figure 8: Mach Number Contour for a mesh 41 by 21, second order scheme

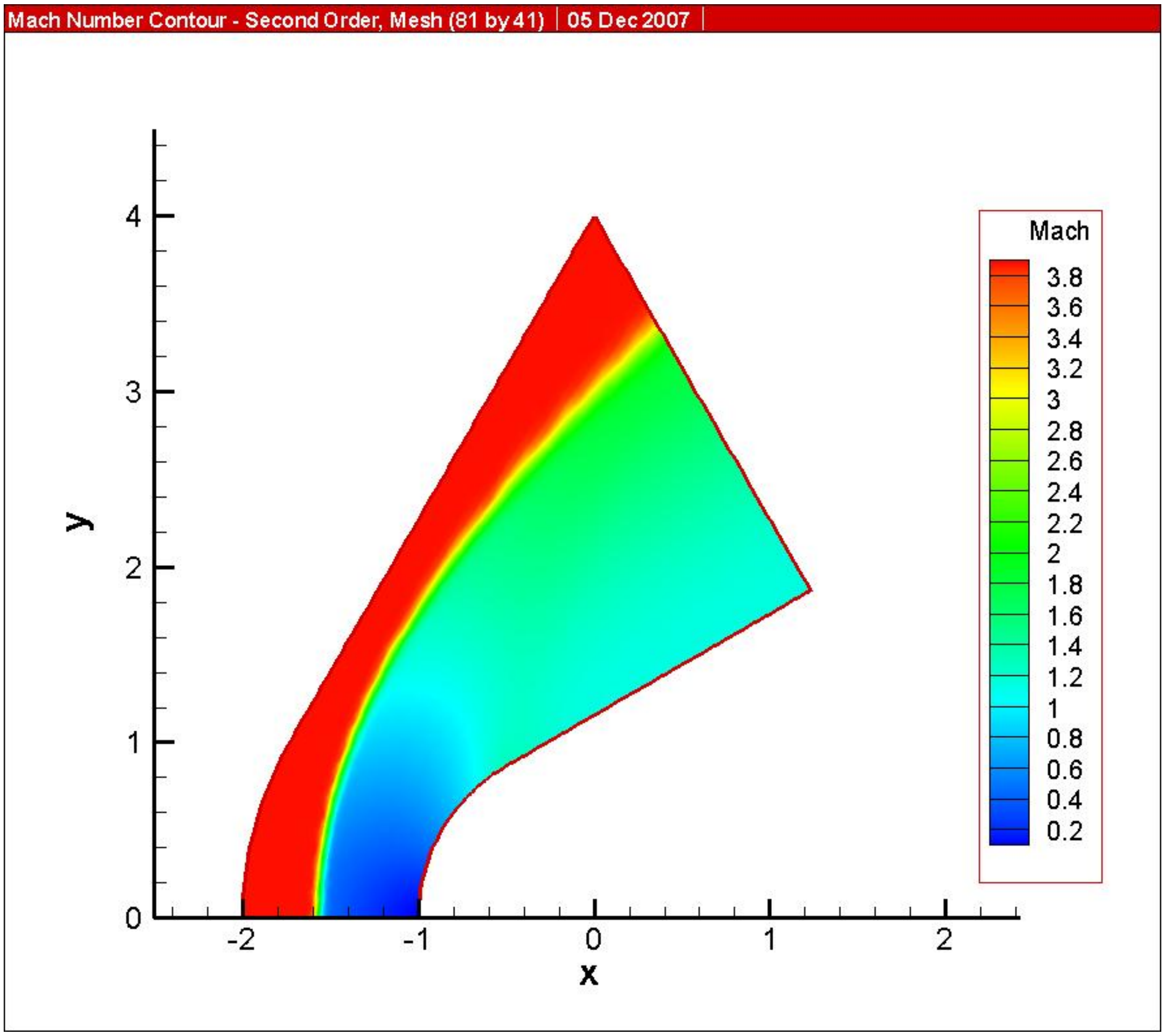


Figure 9: Mach Number Contour for a mesh 81 by 41, second order scheme

Presented below are the density residual plots for each scheme and each mesh.

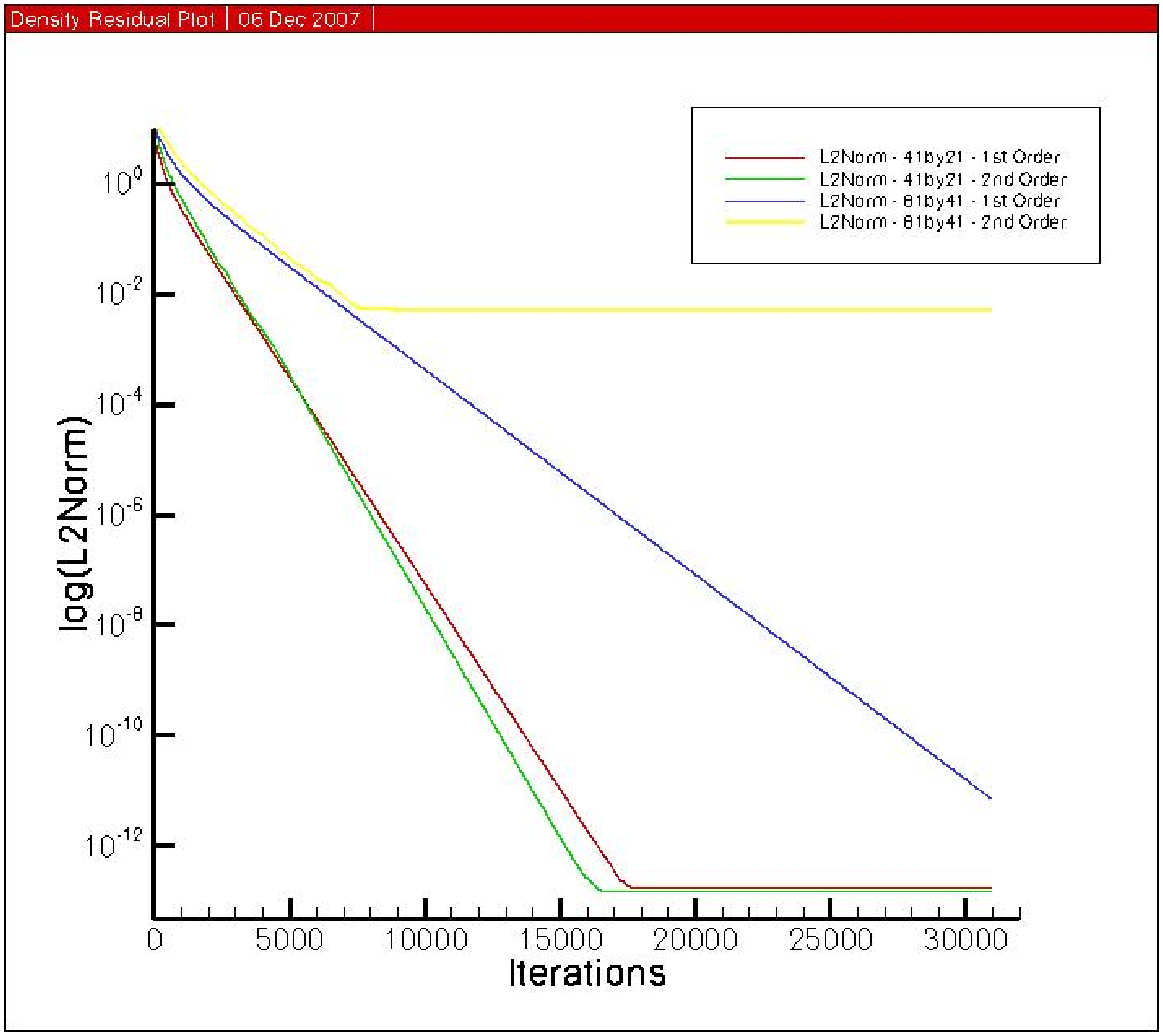


Figure 10: Density Residual Plot

Discussion

From the above results it can be concluded that the code constructed was reliable and solutions obtained were stable. The shock wave is completely contained in the mesh and can be clearly seen in both the first and second order schemes. The second order scheme shows the shock wave is more accurate then the first order scheme and this can be concluded by the curvature of the shock.

Appendix

The following is the code written in FORTRAN in order to solve the 2D Euler equation.

❖ Main Program

```
Program main
Implicit None
Real (Kind=8) :: rhof,pf,uf,vf,gam,Ef,CFL,R2Norm
Real (Kind=8),Allocatable:: XY(:,:,:),SI(:,:,:),SJ(:,:,:),Vol(:,:)
Real (Kind=8),Allocatable:: Q(:,:,:),deltat(:,:)
Real (Kind=8),Allocatable:: Mach(:,:),Pressure(:,:),rho(:,:),Energy(:,:)
Real (Kind=8):: u,v,c ,k
Integer :: i,j,IDp1,JDp1,ID,JD,scheme,Mesh_option,iter

Write(*,*) "2D Euler Solver - Supersonic Flow Solver"
!***********Reading from Input file*******************************
Write(*,*) "Choose Mesh Type: 1-Coarse, 2-Fine"
Read(*,*) Mesh_option

If (Mesh_option == 1) Then
 Open (unit = 2, file = "../dat/Coarse.dat")
 Read(2,*) IDp1,JDp1
 ID = IDp1-1
 JD = JDp1-1
 Allocate (XY(2,IDp1,JDp1))
 Read(2,*)((XY(1,i,j),i=1,IDp1),j=1,JDp1),&
 &((XY(2,i,j),i=1,IDp1),j=1,JDp1)
Else
 Open (unit = 7, file = "../dat/Fine.dat")
 Read(7,*) IDp1,JDp1
 ID = IDp1-1
 JD = JDp1-1
 Allocate (XY(2,IDp1,JDp1))
 Read(7,*)((XY(1,i,j),i=1,IDp1),j=1,JDp1),&
 &((XY(2,i,j),i=1,IDp1),j=1,JDp1)
End If

Open (unit = 8, file = "../dat/CheckMesh.dat")
Write(8,*) IDp1,JDp1
Write(8,*)((XY(1,i,j),i=1,IDp1),j=1,JDp1),&
&((XY(2,i,j),i=1,IDp1),j=1,JDp1)

Close (2)
Close (7)
Close (8)
!**********************************************************************
!***************Geometry Calculations********************************
Allocate(SI(3,ID+1,JD),SJ(3,JD+1,ID),Vol(ID,JD))
Write(*,*) "The Input Mesh was successfully read, entering&
&Geometry Routine"
Call Geometry (ID,JD,XY,SI,SJ,Vol)
!***********************************************************************
Write(*,*) "The Mesh geometry was successfully computed,&
&entering Solver"
```

```
!*************Q Initialization*****************************************
!Write(*,*) "Input CFL number:"
!Read(*,*) CFL
CFL =0.5d0
!Write(*,*) "Input Free-Stream density:"
!Read(*,*) rhof
rhof = 1.4d0
!Write(*,*) "Input Free-Stream pressure:"
!Read(*,*) pf
pf=1.d0
!Write(*,*) "Input Free-Stream u-velocity:"
!Read(*,*) uf
uf=4.d0
!Write(*,*) "Input Free-Stream v-velocity:"
!Read(*,*) vf
vf=0.d0
!Write(*,*) "Input gas constant:"
!Read(*,*)gam
gam=1.4d0
Write(*,*) "Input Scheme Order; 1-First, 2-Second"
Read(*,*) scheme
Ef = (pf/(gam-1.d0))+(0.5d0*rhof*(uf*uf+vf*vf))
!*********************************************************************
Allocate (Q(4,ID,JD))
Do i = 1,ID
 Do j=1,JD
  Q(1,i,j)=rhof
  Q(2,i,j)=rhof*uf
  Q(3,i,j)=rhof*vf
  Q(4,i,j)=Ef
 End Do
End Do

!*********************************************************************
Open (unit=20, file="../dat/Residual.dat")
Allocate (deltat(ID,JD))
R2Norm=1.d0
iter=0
Do iter = 1, 33000!while (R2Norm >= 10.d-13)
!****************Time Step *******************************************
Call timestep(Q,ID,JD,CFL,gam,Vol,SI,SJ,deltat)
!*************Runge - Kutta*******************************************
Call TVDRK(Q,SI,SJ,Vol,deltat,ID,JD,R2Norm,scheme)
!iter = iter+1
Write(20,*) iter,R2Norm
Write(*,*) R2Norm
End Do
!*********************************************************************


!****************Post-Processing**************************************
Allocate (Mach(ID,JD),Pressure(ID,JD),rho(ID,JD),Energy(ID,JD))
Open(unit=15,file="../dat/Output.dat")
Do i=1,ID
 Do j=1,JD
  rho(i,j)=Q(1,i,j)
  u=Q(2,i,j)/Q(1,i,j)
  v=Q(3,i,j)/Q(1,i,j)
  Pressure(i,j) = (Q(4,i,j)-(0.5d0*Q(1,i,j)*(u*u+v*v)))*(gam-1.d0)
  c=DSQRT(gam*Pressure(i,j)/rho(i,j))
  Mach(i,j)=DSQRT(u*u+v*v)/c
  Energy(i,j)=Q(4,i,j)
 End Do
End Do

Write(15,*)"VARIABLES = x, y, Pressure, rho, Energy, Mach"
Write(15,*) "ZONE i=",ID+1,",j=",JD+1
Write(15,*) "DATAPACKING= BLOCK,
VARLOCATION=(3=CELLCENTERED,4=CELLCENTERED,5=CELLCENTERED,6=CELLCENTERED)"

Do j= 1,JD+1
 Do i = 1,ID+1
   Write(15,*) XY(1,i,j)
 End Do
End Do
Do j= 1, JD+1
 Do i = 1, ID+1
   Write(15,*) XY(2,i,j)
 End Do
End Do
Do j =1,JD
 Do i =1,ID
   Write(15,*) Pressure(i,j)
 End Do
End Do
Do j =1,JD
 Do i =1,ID
   Write(15,*) rho(i,j)
 End Do
End Do
Do j =1,JD
 Do i =1,ID
   Write(15,*) Energy(i,j)
 End Do
End Do
Do j =1,JD
 Do i =1,ID
   Write(15,*) Mach(i,j)
 End Do
End Do
Write(*,*) "Code Complete"
Close(20)
Close(15)
Stop
End Program
```

❖ Geometry Subroutine

```
Subroutine Geometry(ID,JD,XY,SI,SJ,Vol)
Implicit None
Integer, Intent (IN):: ID,JD
Integer :: i,j
Real (kind=8), Intent (IN):: XY(2,ID+1,JD+1)
Real (kind=8), Intent (OUT) :: SI(3,ID+1,JD),SJ(3,JD+1,ID),Vol(ID,JD)

Do i = 1,ID+1
 Do j = 1, JD
  SI(1,i,j) = DSQRT((XY(1,i,j+1)-XY(1,i,j))**2.d0 + (XY(2,i,j+1)-
XY(2,i,j))**2.d0)
  SI(2,i,j) = (XY(2,i,j+1)-XY(2,i,j))/SI(1,i,j)
  SI(3,i,j) = -(XY(1,i,j+1)-XY(1,i,j))/SI(1,i,j)
 End Do
End Do

Do i = 1,ID
 Do j = 1, JD+1
  SJ(1,j,i) = DSQRT((XY(1,i+1,j)-XY(1,i,j))**2.d0 + (XY(2,i+1,j)-
XY(2,i,j))**2.d0)
  SJ(2,j,i) = -(XY(2,i+1,j)-XY(2,i,j))/SJ(1,j,i)
  SJ(3,j,i) = (XY(1,i+1,j)-XY(1,i,j))/SJ(1,j,i)
 End Do
End Do

Do i = 1,ID
 Do j = 1,JD
  Vol(i,j) = 0.5d0*DABS(((XY(1,i+1,j+1)-XY(1,i,j))*(XY(2,i,j+1)-
XY(2,i+1,j)))&
            &-((XY(1,i,j+1)-XY(1,i+1,j))*(XY(2,i+1,j+1)-XY(2,i,j))))
 End Do
End Do

Return
End Subroutine
```

❖ Time-Step Subroutine

```
Subroutine timestep(Q,ID,JD,CFL,gam,Vol,SI,SJ,deltat)
Implicit None

Integer, Intent(IN):: ID,JD
Real (kind=8),Intent(IN):: Q(4,ID,JD),Vol(ID,JD),SI(3,ID+1,JD),SJ(3,JD+1,ID)
Real (kind=8),Intent(OUT):: deltat(ID,JD)
Real (kind=8),Intent(IN):: CFL,gam
Real (kind=8),Allocatable::u_vel(:,:),v_vel(:,:),c_cell(:,:)
Real (kind=8),Allocatable::sumi(:,:),sumj(:,:)
Real (kind=8):: velnr,velnl,cf,veln,ul,vl,p_cell
Integer::i,j

Allocate (u_vel(ID,JD),v_vel(ID,JD))
Allocate (c_cell(ID,JD))
Allocate (sumi(ID+1,JD),sumj(ID,JD+1))

Do i = 1,ID
 Do j = 1,JD
  u_vel(i,j)= Q(2,i,j)/Q(1,i,j)
  v_vel(i,j)= Q(3,i,j)/Q(1,i,j)
  p_cell = (gam-1.d0)*(Q(4,i,j)-&
  &0.5d0*Q(1,i,j)*(u_vel(i,j)**2.d0+v_vel(i,j)**2.d0))
  c_cell(i,j) = DSQRT(gam*p_cell/Q(1,i,j))
 End Do
End Do

Do j=1,JD
 Do i=1,ID+1
  If (i==1) Then
   velnr = DABS(u_vel(i,j)*SI(2,i,j)+v_vel(i,j)*SI(3,i,j))
   velnl = DABS(u_vel(i,j)*SI(2,i,j)-v_vel(i,j)*SI(3,i,j))
   cf = c_cell(i,j)
  Else If (i==ID+1) Then
   velnl = DABS(u_vel(i-1,j)*SI(2,i,j)+v_vel(i-1,j)*SI(3,i,j))
   velnr = velnl
   cf = c_cell(i-1,j)
  Else
   velnl = DABS(u_vel(i-1,j)*SI(2,i,j)+v_vel(i-1,j)*SI(3,i,j))
   velnr = DABS(u_vel(i,j)*SI(2,i,j)+v_vel(i,j)*SI(3,i,j))
   cf = 0.5d0*(c_cell(i-1,j)+c_cell(i,j))
  End If
   veln = 0.5d0*(velnl+velnr)
   sumi(i,j) = (veln+cf)*SI(1,i,j)
 End Do
End Do

Do i=1,ID
 Do j=1,JD+1
  If (j==1) Then
   velnr = u_vel(i,j)*SJ(2,j,i)+v_vel(i,j)*SJ(3,j,i)
   ul = u_vel(i,j)-2.d0*velnr*SJ(2,j,i)
   vl = v_vel(i,j)-2.d0*velnr*SJ(3,j,i)
   velnl = ul*SJ(2,j,i)+vl*SJ(3,j,i)
   cf = c_cell(i,j)
  Else If (j==JD+1) Then
   velnl = 4.d0*SJ(2,j,i)!u_vel(i,j-1)*SJ(2,j,i)+v_vel(i,j-1)*SJ(3,j,i)
   velnr = 4.d0*SJ(2,j,i)
   cf = 1.d0!c_cell(i,j-1)
  Else
   velnl = u_vel(i,j-1)*SJ(2,j,i)+v_vel(i,j-1)*SJ(3,j,i)
   velnr = u_vel(i,j)*SJ(2,j,i)+v_vel(i,j)*SJ(3,j,i)
   cf = 0.5d0*(c_cell(i,j-1)+c_cell(i,j))
  End If
   veln = 0.5d0*(DABS(velnl)+DABS(velnr))
   sumj(i,j) = (veln+cf)*SJ(1,j,i)
 End Do
End Do

Do i=1,ID
 Do j=1,JD
  deltat(i,j)=CFL*Vol(i,j)/(sumi(i,j)+sumi(i+1,j)+sumj(i,j)+sumj(i,j+1))
 End Do
End Do

Deallocate (u_vel,v_vel,c_cell,sumi,sumj)
Return
End Subroutine
```

❖ Runge – Kutta TVD Time-Integration Subroutine

```
Subroutine TVDRK(Q,SI,SJ,Vol,deltat,ID,JD,R2Norm,scheme)
Implicit None

Integer, Intent(IN) :: ID,JD,scheme
Real(kind=8),
Intent(IN)::SI(3,ID+1,JD),SJ(3,JD+1,ID),Vol(ID,JD),deltat(ID,JD)
Real(kind=8), Intent(IN OUT)::Q(4,ID,JD)
Real(kind=8), Intent(IN OUT)::R2Norm
Real(kind=8), Allocatable::QO(:,:,:),Res(:,:,:)
Real(kind=8)::Rij
Integer:: i,j,k

Allocate (QO(4,ID,JD),Res(4,ID,JD))
Do i=1,ID
 Do j=1,JD
  Do k =1,4
   QO(k,i,j) = Q(k,i,j)
  End Do
 End Do
End Do

Call Residual(Q,SI,SJ,Vol,ID,JD,Res,scheme)

Do i = 1,ID
 Do j= 1,JD
  Do k= 1,4
   Q(k,i,j) = QO(k,i,j)+(deltat(i,j)*Res(k,i,j))
  End Do
 End Do
End Do

Call Residual(Q,SI,SJ,Vol,ID,JD,Res,scheme)

Do i = 1,ID
 Do j= 1,JD
  Do k= 1,4
   Q(k,i,j) = 0.5d0*(QO(k,i,j)+Q(k,i,j)+(deltat(i,j)*Res(k,i,j)))
  End Do
 End Do
End Do

Rij=0.d0
Do i=1,ID
 Do j=1,JD
  Rij=Rij+((Res(1,i,j))**2.d0)
 End Do
End Do

R2Norm = DABS(DSQRT(Rij/Real(ID*JD)))

Deallocate (QO,Res)
Return
End Subroutine
```

❖ Residual Subroutine

```
Subroutine Residual(Q,SI,SJ,Vol,ID,JD,Res,scheme)
Implicit None

Integer, Intent(IN) :: ID,JD,scheme
Real(kind=8), Intent(IN)::Q(4,ID,JD),SI(3,ID+1,JD),SJ(3,JD+1,ID),Vol(ID,JD)
Real(kind=8), Intent(OUT):: Res(4,ID,JD)
Integer :: i,j,k

Do i = 1,ID
 Do j = 1,JD
  Do k = 1,4
   Res(k,i,j) = 0.d0
  End Do
 End Do
End Do

Call Iflux(Q,SI,ID,JD,Res,scheme)

Call Jflux(Q,SJ,ID,JD,Res,scheme)

Do i = 1,ID
 Do j = 1,JD
  Do k = 1,4
   Res(k,i,j) = Res(k,i,j)/Vol(i,j)
  End Do
 End Do
End Do

Return
End Subroutine
```

❖ Flux in the I- Direction Subroutine

```
Subroutine Iflux(Q,SI,ID,JD,Res,scheme)
Implicit None

Integer, Intent(IN)::ID,JD,scheme
Real (kind=8), Intent(IN)::Q(4,ID,JD),SI(3,ID+1,JD)
Real (kind=8), Intent(IN OUT):: Res(4,ID,JD)
Real (kind=8), Allocatable:: QL(:,:),QR(:,:),Flux(:,:)
Real (kind=8)::Sksi
Integer ::i,j,k

Allocate (QL(4,ID+1),QR(4,ID+1),Flux(4,ID+1))
If (scheme == 1) Then

Do j=1,JD
 Do i=2,ID
  Do k=1,4
    QL(k,i)=Q(k,i-1,j)
    QR(k,i)=Q(k,i,j)
  End Do
 End Do

 i=1
 Do k=1,4
   QR(k,i)=Q(k,i,j)
 End Do
 QL(1,i)=QR(1,i)
 QL(2,i)=QR(2,i)
 QL(3,i)=-QR(3,i)
 QL(4,i)=QR(4,i)

 i=ID+1
 Do k=1,4
   QL(k,i)=Q(k,i-1,j)
   QR(k,i)=QL(k,i)
 End Do

 Call Rusonov_Flux(QL,QR,SI(1,1,j),Flux,ID+1)

 Do i=1,ID
  Do k=1,4
    Res(k,i,j)=Res(k,i,j)-(Flux(k,i+1)*SI(1,i+1,j)-Flux(k,i)*SI(1,i,j))
  End Do
 End Do

End Do

Else

Do j=1,JD
 Do i=2,ID-1
  Do k=1,4
```

```
    Sksi = minmod((Q(k,i+1,j)-Q(k,i,j)),(Q(k,i,j)-Q(k,i-1,j)))
    QL(k,i+1) = Q(k,i,j) + 0.5d0*Sksi !Goes from 3 to ID
    QR(k,i) = Q(k,i,j) - 0.5d0*Sksi !Goes from 2 to ID-1
  End Do
 End Do
 i = 2
 Do k=1,4
    QL(k,i) = Q(k,i-1,j) + 0.5d0*(Q(k,i,j)-Q(k,i-1,j))
 End Do
 i = ID
 Do k=1,4
    QR(k,i) = Q(k,i,j) - 0.5d0*(Q(k,i,j)-Q(k,i-1,j))
 End Do

 i = 1
 Do k=1,4
    QR(k,i) = Q(k,i,j) - 0.5d0*(Q(k,i+1,j)-Q(k,i,j))
 End Do
    QL(1,i) = QR(1,i)
    QL(2,i) = QR(2,i)
    QL(3,i) = -QR(3,i)
    QL(4,i) = QR(4,i)

 i=ID+1
 Do k=1,4
    QL(k,i) = Q(k,i-1,j) + 0.5d0*(Q(k,i-1,j)-Q(k,i-2,j))
    QR(k,i) = QL(k,i)
 End Do

 Call Rusonov_Flux(QL,QR,SI(1,1,j),Flux,ID+1)
 Do i=1,ID
  Do k=1,4
   Res(k,i,j)=Res(k,i,j)-(Flux(k,i+1)*SI(1,i+1,j)-Flux(k,i)*SI(1,i,j))
  End Do
 End Do

End Do
End If

Deallocate(QL,QR,Flux)
Return

Contains
Function minmod(x,y)
Real(kind=8), Intent(IN):: x,y
Real(kind=8)::minmod
If (x*y .LE. 0.d0) Then
      minmod = 0.d0
Else
      minmod = (x/DABS(x))*DMIN1(DABS(x),DABS(y))
End If
Return
End Function

End Subroutine Iflux
```

❖ Flux in the J- Direction Subroutine

```
Subroutine JFlux(Q,SJ,ID,JD,Res,scheme)
Implicit None

Integer, Intent(IN) :: ID,JD,scheme
Real(kind=8), Intent(IN)::Q(4,ID,JD),SJ(3,JD+1,ID)
Real(kind=8), Intent(IN OUT)::Res(4,ID,JD)
Real(kind=8), Allocatable:: QL(:,:),QR(:,:),Flux(:,:)
Real(kind=8)::Sksi
Integer::i,j,k
Real(kind=8)::NM

Allocate (QL(4,JD+1),QR(4,JD+1),Flux(4,JD+1))

If (scheme==1) Then

Do i=1,ID
 Do j=2,JD
  Do k=1,4
    QL(k,j)=Q(k,i,j-1)
    QR(k,j)=Q(k,i,j)
  End Do
 End Do

 j=1
 Do k=1,4
   QR(k,j)=Q(k,i,j)
 End Do
 QL(1,j)=QR(1,j)
 QL(4,j)=QR(4,j)
 NM=QR(2,j)*SJ(2,j,i)+QR(3,j)*SJ(3,j,i)
 QL(2,j)=QR(2,j)-2.d0*NM*SJ(2,j,i)
 QL(3,j)=QR(3,j)-2.d0*NM*SJ(3,j,i)

 j=JD+1
 Do k=1,4
   QL(k,j)=Q(k,i,j-1)
 End Do
 QR(1,j)=1.4d0
 QR(2,j)=1.4d0*4.d0
 QR(3,j)=0.d0
 QR(4,j)=(1.d0/0.4d0)+(0.5d0*1.4d0*4.d0*4.d0)
!Do k=1,4
!   QL(k,j) = QR(k,j)
!End Do

 Call Rusonov_Flux(QL,QR,SJ(1,1,i),Flux,JD+1)

 Do j=1,JD
  Do k=1,4
    Res(k,i,j)=Res(k,i,j)-(Flux(k,j+1)*SJ(1,j+1,i)-Flux(k,j)*SJ(1,j,i))
  End Do
 End Do
End Do

Else

Do i=1,ID
 Do j=2,JD-1
  Do k=1,4
    Sksi = minmod((Q(k,i,j+1)-Q(k,i,j)),(Q(k,i,j)-Q(k,i,j-1)))
    QL(k,j+1) = Q(k,i,j) + 0.5d0*Sksi! Goes from 3 to JD
    QR(k,j) = Q(k,i,j) - 0.5d0*Sksi! Goes from 2 to JD-1
  End Do
 End Do
 j=2
 Do k=1,4
   QL(k,j) = Q(k,i,j-1) + 0.5d0*(Q(k,i,j)-Q(k,i,j-1))
 End Do
 j=JD
 Do k=1,4
   QR(k,j) = Q(k,i,j) - 0.5d0*(Q(k,i,j)-Q(k,i,j-1))
 End Do

 j=1
 Do k=1,4
   QR(k,j) = Q(k,i,j) - 0.5d0*(Q(k,i,j+1)-Q(k,i,j))
 End Do
   QL(1,j)=QR(1,j)
   QL(4,j)=QR(4,j)
   NM=QR(2,j)*SJ(2,j,i)+QR(3,j)*SJ(3,j,i)
   QL(2,j)=QR(2,j)-2.d0*NM*SJ(2,j,i)
   QL(3,j)=QR(3,j)-2.d0*NM*SJ(3,j,i)

 j=JD+1
 Do k=1,4
   QL(k,j) = Q(k,i,j-1) + 0.5d0*(Q(k,i,j-1)-Q(k,i,j-2))!Q(k,i,j-1)
 End Do
 QR(1,j)=1.4d0
 QR(2,j)=1.4d0*4.d0
 QR(3,j)=0.d0
 QR(4,j)=(1.d0/0.4d0)+(0.5d0*1.4d0*4.d0*4.d0)
 !Do k=1,4
 !  QL(k,j) = QR(k,j)
 !End Do

 Call Rusonov_Flux(QL,QR,SJ(1,1,i),Flux,JD+1)

 Do j=1,JD
  Do k=1,4
    Res(k,i,j)=Res(k,i,j)-(Flux(k,j+1)*SJ(1,j+1,i)-Flux(k,j)*SJ(1,j,i))
  End Do
 End Do

End Do

End If
Deallocate (QL,QR,Flux)
Return
```

```
Contains

Function minmod(x,y)
Real(kind=8), Intent(IN):: x,y
Real(kind=8)::minmod

If (x*y .LE. 0.d0) Then
      minmod = 0.d0
Else
      minmod = (x/DABS(x))*DMIN1(DABS(x),DABS(y))
End If

Return
End Function

End Subroutine Jflux
```

❖ Rusonov Flux Subroutine

```
Subroutine Rusonov_Flux(QL,QR,n,Flux,len)
Implicit None

Integer, Intent(IN)::len
Real (kind=8), Intent(IN)::QL(4,len),QR(4,len),n(3,len)
Real (kind=8), Intent(OUT):: Flux(4,len)
Real (kind=8)::ul,vl,Pl,cl,ur,vr,Pr,cr,vnl,vnr,lambda
Integer::g

Do g =1,len

 ul=QL(2,g)/QL(1,g)

 vl=QL(3,g)/QL(1,g)

 Pl=(QL(4,g)-(0.5d0*QL(1,g)*((ul*ul)+(vl*vl))))*(1.4d0-1.d0)

 cl=DSQRT((1.4d0*Pl)/QL(1,g))

 ur=QR(2,g)/QR(1,g)

 vr=QR(3,g)/QR(1,g)

 Pr=(QR(4,g)-(0.5d0*QR(1,g)*((ur*ur)+(vr*vr))))*(1.4d0-1.d0)

 cr=DSQRT((1.4d0*Pr)/QR(1,g))

 vnl=(ul*n(2,g))+(vl*n(3,g))

 vnr=(ur*n(2,g))+(vr*n(3,g))

 lambda=0.5d0*(DABS(vnl)+DABS(vnr)+cl+cr)

 Flux(1,g) = 0.5d0*((QL(1,g)*vnl)+(QR(1,g)*vnr)-(lambda*(QR(1,g)-QL(1,g))))

 Flux(2,g) = 0.5d0*(((QL(2,g)*vnl)+(Pl*n(2,g)))+((QR(2,g)*vnr)+(Pr*n(2,g)))-&
             & (lambda*(QR(2,g)-QL(2,g))))

 Flux(3,g) = 0.5d0*(((QL(3,g)*vnl)+(Pl*n(3,g)))+((QR(3,g)*vnr)+(Pr*n(3,g)))-&
             & (lambda*(QR(3,g)-QL(3,g))))

 Flux(4,g) = 0.5d0*((vnl*(QL(4,g)+Pl))+(vnr*(QR(4,g)+Pr))-&
             & (lambda*(QR(4,g)-QL(4,g))))

End Do

Return
End Subroutine
```