Solving the 2D Euler Equation over the NACA 0012 airfoil numerically

Yifei Guan

Mechanical Engineering

University of Washington

Mar 2016

Abstract

This report studies the transonic flow over the NACA 0012 airfoil by modeling the 2-D Euler equation using Jameson's Finite volume scheme with artificial viscosity and 4-stage Runge - Kutta time stepping algorithm. The CFL number is given for the whole domain, but local time-step size is determined based on the local velocity, local speed of sound and cell size.

Table of Contents:

1. Introduction

This report studies the pressure, density, velocity, enthalpy, entropy, Mach number and the pressure coefficient of the transonic flow over the airfoil. Boundary conditions are specified for the physical value or numerical extrapolation following the multidimensional compatibility relations and the Riemann variables associated with the normal direction are implemented. The 4-stage Runga-Kutta scheme is implemented to advance in time which is theoretical stable for CFL number less or equal to 2.8.

2. Initial condition

The 2-D grid is generated by solving the Poisson's equation for the initial transfinite grid using the Jameson's algorithm. The initial condition is that for each cell inside the domain, the properties are equal to the ones in the main flow.

3. Boundary conditions

Since the attack angle is 0 degree, the inlet velocity is set to be the same as the main stream velocity. The speed of sound is taken as the average of the main stream speed of sound and the local sound speed of the first adjacent interior cell. Pressure, density and enthalpy are taken as the main stream value.

As for the outlet condition, the velocity outlet and the density are linearly extrapolated by the velocity and density of the first two adjacent interior cells. The speed of sound is taken as the average of the main stream speed of sound and the local sound speed of the first adjacent interior cell. The pressure is taken as the main stream pressure. And the enthalpy is taken as the same as the first adjacent interior cell.

For the airfoil, the solid wall boundary condition should be implemented. No flux is going out of the solid wall boundary and a slip condition is implemented.

3.1 Numerical scheme for 2-D Euler equation

The 2-dimensional Euler equation:

$$\frac{\partial U}{\partial t}+\frac{\partial F}{\partial x}+\frac{\partial G}{\partial y}=0 \tag{1}$$

$$\frac{d\ U_{i,J}\Omega_{i,J}}{dt}+\sum_{face}\underset{-}{F}^{*(AV)}\bullet\Delta\underset{-}{S}=0 \tag{2}$$

The residual is defined by

$$\frac{d\ U_{i,J}\Omega_{i,J}}{dt}=-\sum_{face}\underset{-}{F}^{*(AV)}\bullet\Delta\underset{-}{S}=-R_{i,J} \tag{3}$$

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho u \\ \rho E \end{bmatrix},\ F = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uH \end{bmatrix} \text{and}\, G = \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + P \\ \rho vH \end{bmatrix} \tag{4}$$

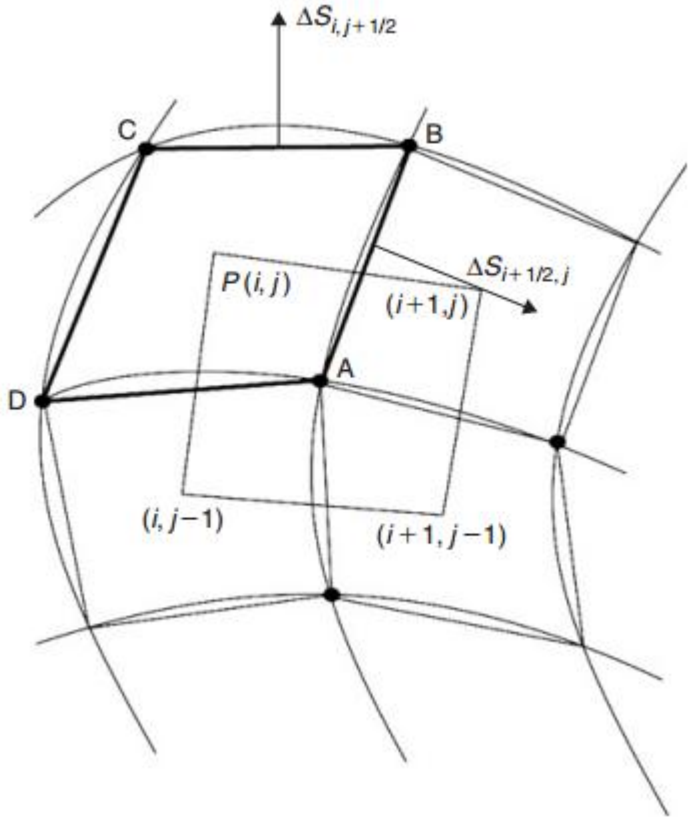

Figure 1. Cell centered space discretization

$$\Delta\vec{S}_{i+1/2,j} = \Delta\vec{S}_{\mathrm{AB}} = \Delta y_{\mathrm{AB}}\vec{1}_x - \Delta x_{\mathrm{AB}}\vec{1}_y = (y_{\mathrm{B}} - y_{\mathrm{A}})\vec{1}_x - (x_{\mathrm{B}} - x_{\mathrm{A}})\vec{1}_y$$

$$\begin{aligned} \Omega_{i,j} &= \Omega_{\mathrm{ABCD}} = \frac{1}{2}|\vec{x}_{\mathrm{AC}} \times \vec{x}_{\mathrm{BD}}| \\ &= \frac{1}{2}[(x_{\mathrm{C}} - x_{\mathrm{A}})(y_{\mathrm{D}} - y_{\mathrm{B}}) - (x_{\mathrm{D}} - x_{\mathrm{B}})(y_{\mathrm{C}} - y_{\mathrm{A}})] \\ &= \frac{1}{2}(\Delta x_{\mathrm{AC}}\Delta y_{\mathrm{BD}} - \Delta x_{\mathrm{BD}}\Delta y_{\mathrm{AC}}) \end{aligned}$$

Numerical flux:

$$(\vec{F}^* \cdot \Delta\vec{S})_{i+1/2,j} \equiv \left[\frac{1}{2}(\vec{F}_{i,j} + \vec{F}_{i+1,j}) - D_{i+1/2,j}\right] \cdot \Delta\vec{S}_{i+1/2,j}$$

Artificial viscosity:

$$D_{i+1/2,j} = \gamma_{i+1/2,j}(U_{i+2,j} - 3U_{i+1,j} + 3U_{i,j} - U_{i-1,j})$$

Where the coefficient is determined by:

$$\gamma_{i+1/2,j} = \frac{1}{2}\kappa^{(4)}[\vec{v} \cdot \Delta\vec{S} + c|\Delta\vec{S}|]_{i+1/2,j}$$

Time integration, Runge-Kutta Scheme:

$$U_{i,j}^{(1)} = U_{i,j}^{n} - \frac{\Delta t}{\Omega_{i,j}}\alpha_1 R_{i,j}^{n}$$

$$U_{i,j}^{(2)} = U_{i,j}^{n} - \frac{\Delta t}{\Omega_{i,j}}\alpha_2 R_{i,j}^{(1)}$$

$$U_{i,j}^{(3)} = U_{i,j}^{n} - \frac{\Delta t}{\Omega_{i,j}}\alpha_3 R_{i,j}^{(2)}$$

$$U_{i,j}^{n+1} = U_{i,j}^{n} - \frac{\Delta t}{\Omega_{i,j}}\alpha_4 R_{i,j}^{(3)}$$

Where

$$\alpha_1 = \frac{1}{8} \quad \alpha_2 = 0.306 \quad \alpha_3 = 0.587 \quad \alpha_4 = 1$$

For optimization of the dissipation.

The local time step size is determined by:

$$\Delta t_{i,j} \leq \text{CFL}\frac{\Omega_{i,j}}{|(\vec{v}+c)_{i,j} \cdot \Delta\vec{S}_i| + |(\vec{v}+c)_{i,j} \cdot \Delta\vec{S}_{j+1/2}|}$$

Where:

$$\Delta\vec{S}_i = \frac{1}{2}(\Delta\vec{S}_{i+1/2,j} + \Delta\vec{S}_{i-1/2,j}) \quad \Delta\vec{S}_j = \frac{1}{2}(\Delta\vec{S}_{i,j+1/2} + \Delta\vec{S}_{i,j-1/2})$$

After updating for the matrix U, properties can be extract from matrix U and can be calculated afterwards,

$$\rho = U \;\; 1,: \;\; , \; u = U \;\; 2,: \; / \rho \, , \; c = \sqrt{\gamma \frac{P}{\rho}} \, , \; Ma = \frac{u}{c} \tag{5}$$

$$P = \;\; \gamma - 1 \, \left[\rho E - \rho u^2 / 2 \right], \; H = E + P / \rho \, , \; S - S_L = \frac{R}{\gamma - 1} \ln \left(\frac{P / P_L}{\rho / \rho_L \;\; ^{\gamma}} \right) \tag{6}$$

3.2 Computational result

The comparison of the analytical solution and the numerical results are visualized in **Figure 2** to **Figure 7**.

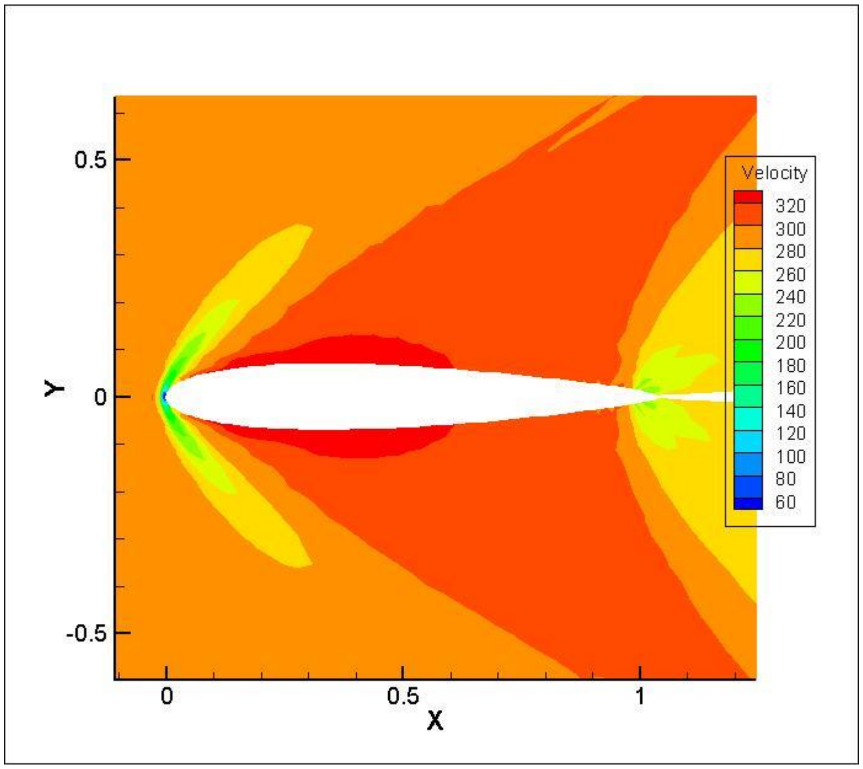

Figure 2. Velocity contour

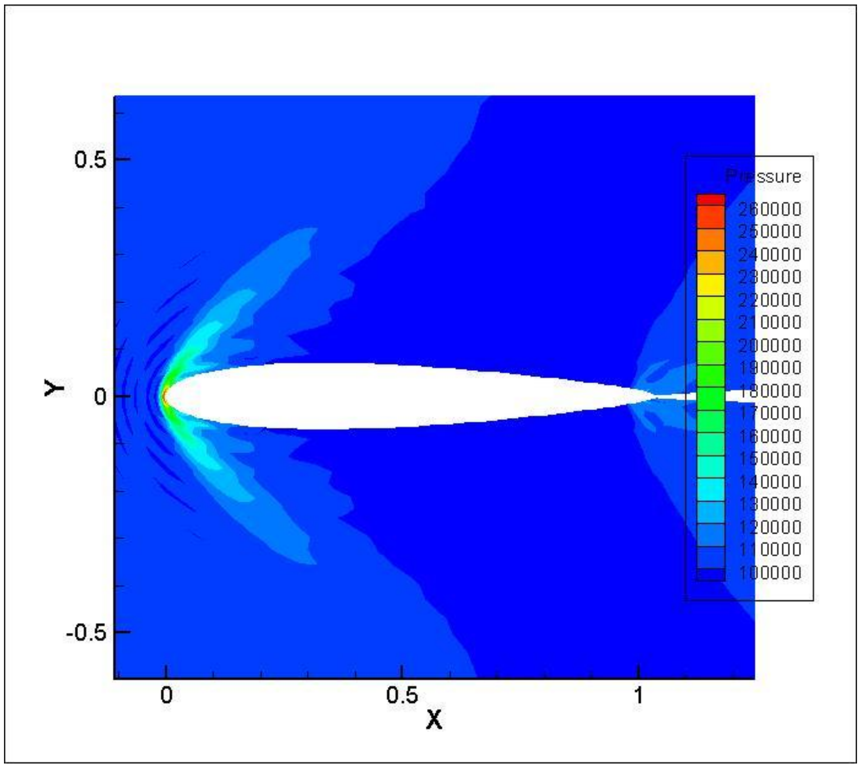

Figure 3. Pressure contour

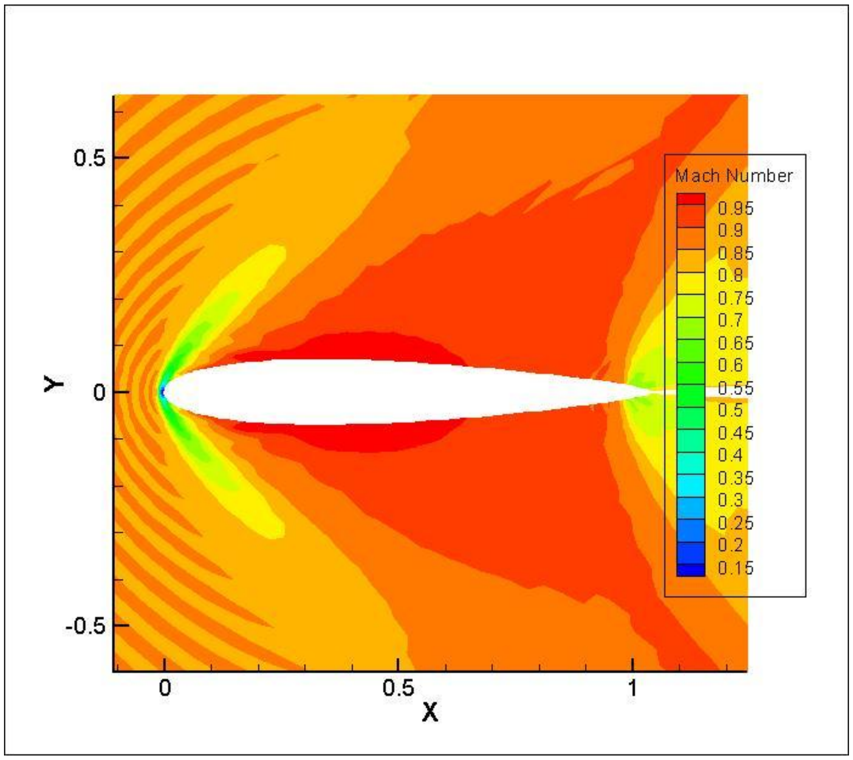

Figure 4. Mach number

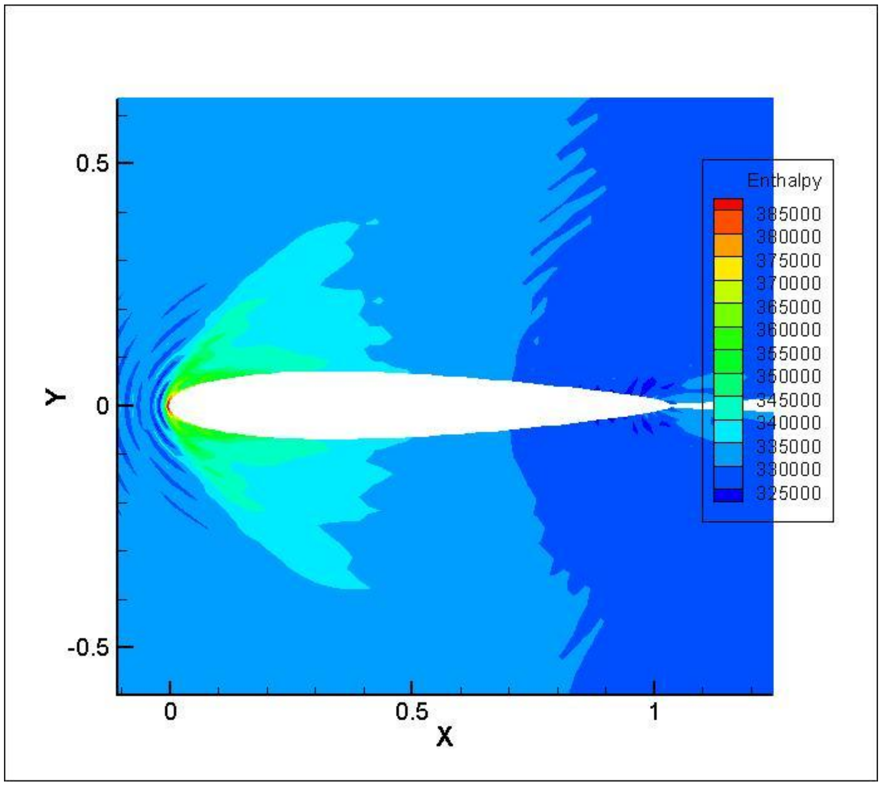

Figure 5. Enthalpy

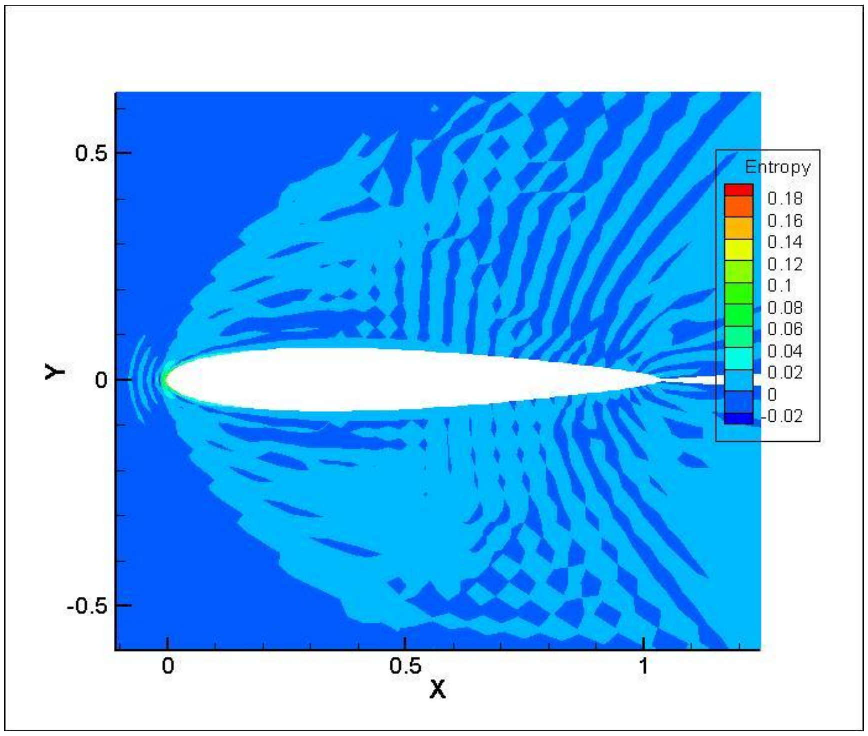

Figure 6. Entropy

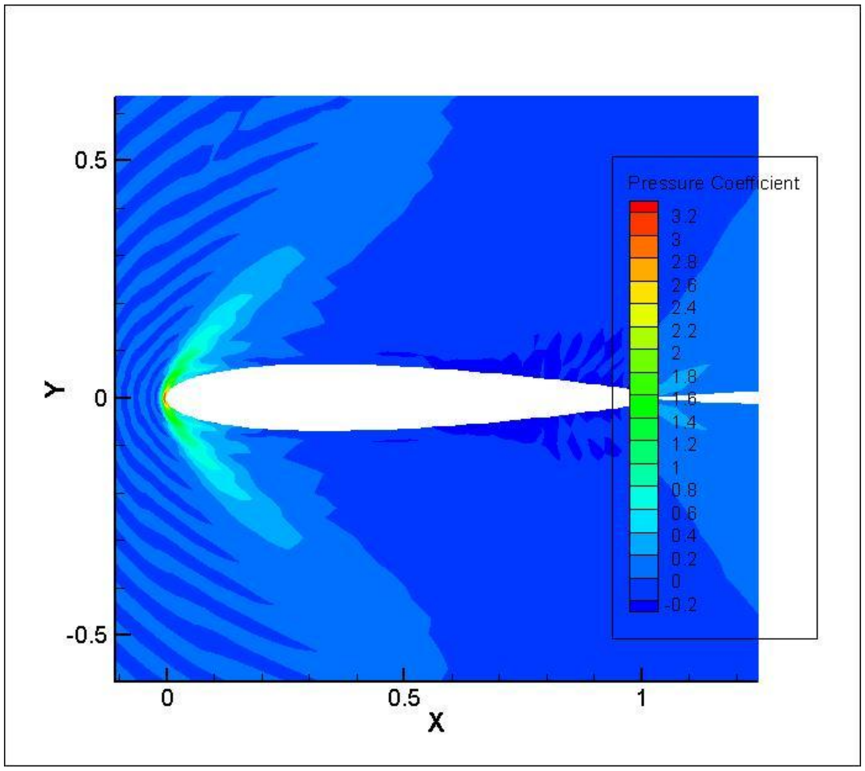

Figure 7. Pressure coefficient

4. Conclusion

This report shows the numerical solution for the transonic flow over the NACA 0012 airfoil. Unfortunately, I had a hard time getting the solution to converge. The computational result here is definitely not what is expected. The flow failed to develop into supersonic and create a shock on and below the airfoil. The contours are rough rather than smooth. The reason for this remains unknown for me. But I guess it is because that the inlet and outlet boundary conditions are off. The artificial viscosity is implemented but doesn't help much about this convergence.

Nonetheless, there are few points making sense about this result. Even though the shock wave hasn't been developed yet. A velocity increase and thus the Mach number is almost unity at the shock region. The other thing is the contours are all symmetric along the airfoil which means the numerical scheme should be correct. And a stagnation point which zero velocity and high pressure at the leading edge can also be distinguished.

Appendix Fortran code

```
! A fortran95 program for solving the 2-D Euler equation
! over the NACA0012 airfoil
program main
  implicit none

  integer :: i, J, k, iplus1, iplus2, iplus3, iminus1, iminus2, iminus3, m, n
  integer, parameter :: Imax=128, Jmax = 104 ! i and J direction coordinates of the cells
                                  ! X and Y are cell coordinate, XX and YY are grid point coordinates
  real(kind = 8), allocatable    :: X(:, :), Y(:, :), XX(:, :), YY(:, :)
  real(kind = 8), allocatable    :: omega(:, :), surf(:, :, :, :), deltt(:, :)
  real(kind = 8), allocatable    :: u(:, :), rho(:, :), v(:, :), P(:, :), H(:, :), E(:, :), c(:, :)
  real(kind = 8), allocatable    :: UU(:, :, :), FF(:, :, :), GG(:, :, :), Fstar(:, :, :, :), UUU(:)
  real(kind = 8), allocatable    :: D(:, :), Fsum(:, :, :, :), FS(:, :, :, :, :), Res(:, :, :), al(:)
  real(kind = 8) alpha, Machinf, CFL, uinf, cinf, pinf, rhoinf, ga, vinf, einf, hinf, hinter
  real(kind = 8) eps2core, eps2, eps4, k2, k4, v1, v2, v3, v4, delt
  real(kind = 8) RnBplus, RnIminus, UnB, UlB
  allocate(X(1:Imax+7, 1:Jmax+7),Y(1:Imax+7, 1:Jmax+7), u(1:Imax+7, 1:Jmax+7) &
        , v(1:Imax+7, 1:Jmax+7),  E(1:Imax+7, 1:Jmax+7)&
        , rho(1:Imax+7, 1:Jmax+7), P(1:Imax+7, 1:Jmax+7), H(1:Imax+7, 1:Jmax+7), c(1:Imax+7,
1:Jmax+7) &
        , XX(Imax + 2, Jmax + 1), YY(Imax + 2, Jmax + 1)) ! flow properties
  allocate (UU(4, 1:Imax+7, 1:Jmax+7), FF(4, 1:Imax+7, 1:Jmax+7))
  allocate (GG(4, 1:Imax+7, 1:Jmax+7), Fstar(2, 4, 1:Imax+7, 1:Jmax+7))
  allocate (D(4, 4), Res(Imax+7, Jmax+7, 4), al(4), UUU(4), deltt(Imax + 7, Jmax + 7))
  allocate (surf(Imax+7, Jmax+7, 4, 2), omega(Imax+7, Jmax+7), FS(Imax+7, Jmax+7, 4, 2, 4),
Fsum(Imax+7, Jmax+7, 2, 4))
  alpha = 0 ! angle of attach
  Machinf = 0.85 ! out stream Mach number
  CFL = 0.01 ! CFL number
  pinf = 101325.0 ! outer pressure
  rhoinf = 1.225 ! outer density
  !pinf = 1.0
  !rhoinf = 1.0
  ga = 1.4 ! specific heat ratio
  cinf = sqrt(ga * pinf / rhoinf) ! outer speed of sound
  uinf = cinf * Machinf ! outer uniform velocity in x direction
  vinf = 0.0 !o outer uniform velocity in y direction
  einf = 1.0 / (ga - 1) * pinf / rhoinf + 0.5 * (uinf * uinf + vinf * vinf)
  hinf = einf + pinf / rhoinf
  k2 = 1.0 / 4.0
  k4 = 1.0 / 256.0
  !k2 = 0.0
  !k4 = 0.0 / 10.0
  al(1) = 1.0 / 8.0
  al(2) = 0.306
  al(3) = 0.587
  al(4) = 1.0
  !al(1) = 1.0 / 4.0
  !al(2) = 1.0 / 3.0
  !al(3) = 1.0 / 2.0
  !al(4) = 1.0
  !delt = 0.000001

  open(unit = 10, file='mesh.dat')
  do J=1, Jmax + 1
    do i=1, Imax + 1
      read(10, *) XX(i, J), YY(i, J) ! read grids
    enddo
      XX(Imax + 2, J) = XX(1, J)
      YY(Imax + 2, J) = YY(1, J)
  enddo
 ! CLOSE (unit = 10, STATUS='KEEP')
!  do J = 1, Jmax + 1
!    XX(Imax + 1, J) = XX(1, J)
!    YY(Imax + 1, J) = YY(1, J)
!  enddo

  ! space discretization, surface direction & cell area
  do J = 1, Jmax
    do i = 1, Imax
       iplus1 = i + 1
       !if (i .GT. Imax - 1) then
       !    iplus1 = 1
       !endif
      ! space discretization
      X(i+4, J+4) = 0.25 * (XX(i, J) + XX(iplus1, J) + XX(i, J + 1) + XX(iplus1, J + 1))
      Y(i+4, J+4) = 0.25 * (YY(i, J) + YY(iplus1, J) + YY(i, J + 1) + YY(iplus1, J + 1))
      ! compute surface direction
      ! x - direction
      surf(i+4, J+4, 1, 1) = YY(iplus1, J) - YY(iplus1, J + 1) ! i + 1/2, J
      surf(i+4, J+4, 2, 1) = YY(iplus1, J + 1) - YY(i, J + 1) ! i, J + 1/2
      surf(i+4, J+4, 3, 1) = YY(i, J + 1) - YY(i, J) ! i - 1/2, J
      surf(i+4, J+4, 4, 1) = YY(i, J) - YY(i + 1, J) ! i, J - 1/2
      ! y - direction
      surf(i+4, J+4, 1, 2) = - XX(iplus1, J) + XX(iplus1, J + 1)
      surf(i+4, J+4, 2, 2) = - XX(iplus1, J + 1) + XX(i, J + 1)
      surf(i+4, J+4, 3, 2) = - XX(i, J + 1) + XX(i, J)
      surf(i+4, J+4, 4, 2) = - XX(i, J) + XX(iplus1, J)

      ! compute cell areas
      omega(i+4, J+4) = 0.5 * abs((XX(i, J + 1) - XX(iplus1, J)) * (YY(i, J) - YY(iplus1, J + 1)) &
                  - (XX(i, J) - XX(iplus1, J + 1)) * (YY(i, J + 1) - YY(iplus1, J)))
    enddo
  enddo

  ! Periodic condition
  !X(0, :) = (X(Imax, :) + X(1, :)) * 0.5
  !X(Imax + 1, :) = (X(1, :) + X(Imax, :)) * 0.5
  !Y(0, :) = (Y(Imax, :) + Y(1, :)) * 0.5
  !Y(Imax + 1, :) = (Y(1, :) + Y(Imax, :)) * 0.5
  ! initial condition
  do J = 1, Jmax + 7
    do i = 1, Imax + 7
        u(i, J) = uinf
        v(i, J) = 0.0
        rho(i, J) = rhoinf
        E(i, J) = einf
        H(i, J) = hinf
        P(i, J) = pinf
        c(i, J) = cinf
    enddo
  enddo
  UU(1, :, :) = rho
  UU(2, :, :) = rho * u
  UU(3, :, :) = rho * v
  UU(4, :, :) = rho * E


  !write(*,*) Y(Imax + 1, Jmax)
!stop
  open(20,file='initial.dat')
  write(20,*) 'VARIABLES = "X", "Y", "U", "V", "surface"'
  write(20,*) 'ZONE I=', Imax,   ', J=', Jmax, 'F=POINT'
  do J=5, Jmax + 4
    do i=5, Imax + 4
      write(20,603) X(i, J), Y(i, J), UU(2, i, J) / UU(1, i, J), UU(3, i, J) / UU(1, i, J), surf(i, J, 1, 1)
    enddo
  enddo
  CLOSE (20 , STATUS='KEEP')
!stop
! write(*, *) uinf

  open(30, file='residuals.dat')
  write(30,*) 'VARIABLES = "Iterations", "Residual"'

do n = 1, 100
    ! update solution
          P = (ga - 1) * (UU(4, :, :) &
            - 0.5 * (UU(2, :, :) * UU(2, :, :) + UU(3, :, :) * UU(3, :, :)) / UU(1, :, :))

          H = ga * UU(4, :, :) / UU(1, :, :) &
            - 0.5 * (ga - 1) * (UU(2, :, :) * UU(2, :, :) &
            + UU(3, :, :) * UU(3, :, :)) / (UU(1, :, :) * UU(1, :, :))

          FF(1, :, :) = UU(2, :, :)
          FF(2, :, :) = UU(2, :, :) * UU(2, :, :) / UU(1, :, :) + P
          FF(3, :, :) = UU(2, :, :) * UU(3, :, :) / UU(1, :, :)
          FF(4, :, :) = UU(2, :, :) * H
          GG(1, :, :) = UU(3, :, :)
          GG(2, :, :) = FF(3, :, :)
          GG(3, :, :) = UU(3, :, :) * UU(3, :, :) / UU(1, :, :) + P
          GG(4, :, :) = UU(3, :, :) * H

          Fstar(1, :, :, :) = FF
          Fstar(2, :, :, :) = GG

          rho = UU(1, :, :)
          U = UU(2, :, :) / rho
          V = UU(3, :, :) / rho
          E = H - p / rho
          c = sqrt(ga * p / rho)

    ! boundary condition
    do i = 4, Imax + 4
      if (surf(i, Jmax + 4, 2, 1) * UU(2, i, Jmax + 4) .LT. 0) then ! inlet condition
        !theta = atan((surf(i, Jmax, 2, 2) / surf(i, Jmax, 2, 1)))
        !write(*,*) theta
        !RnBplus = uinf * cos(theta) + 2 * cinf / (ga - 1)
        !RnBplus = uinf * surf(i, Jmax + 4, 2, 1) / sqrt(surf(i, Jmax + 4, 2, 1) * surf(i, Jmax + 4, 2, 1) &
        !        + surf(i, Jmax + 4, 2, 2) * surf(i, Jmax + 4, 2, 2)) + 2 * cinf / (ga - 1)
        !RnIminus = UU(2, i, Jmax - 1) / UU(1, i, Jmax - 1) * cos(theta) &
        !         - UU(3, i, Jmax - 1) / UU(1, i, Jmax - 1) * sin(theta) &
        !         - 2 * c(i, Jmax - 1) / (ga - 1)
        !RnIminus = (UU(2, i, Jmax + 4) / UU(1, i, Jmax + 4) * surf(i, Jmax + 4, 2, 1) &
        !         + UU(3, i, Jmax + 4) / UU(1, i, Jmax + 4) * surf(i, Jmax + 4, 2, 2)) &
        !         / sqrt(surf(i, Jmax, 2, 1) * surf(i, Jmax, 2, 1) &
        !        + surf(i, Jmax, 2, 2) * surf(i, Jmax, 2, 2)) &
        !        - 2 * c(i, Jmax) / (ga)
        !UnB = 0.5 * (RnBplus + RnIminus)
        !C(i, Jmax) = 0.25 * (ga - 1) * (RnBplus - RnIminus)
        !UlB = - uinf * surf(i, Jmax + 4, 2, 2) / sqrt(surf(i, Jmax + 4, 2, 1) * surf(i, Jmax + 4, 2, 1) &
        !    + surf(i, Jmax + 4, 2, 2) * surf(i, Jmax + 4, 2, 2))

        !write(*, *) UnB
        !U(i, Jmax) = UnB * cos(theta) + UlB * sin(theta)
        !V(i, Jmax) = UnB * sin(theta) - UlB * cos(theta)
        !U(i, Jmax + 1) = (uinf + UU(2, i, Jmax ) / UU(1, i, Jmax)) * 0.5
        !U(i, Jmax + 1) = uinf

        !U(i, Jmax + 5) =  UnB * surf(i, Jmax + 4, 2, 2) / sqrt(surf(i, Jmax + 4, 2, 1) * surf(i, Jmax + 4, 2, 1)
&
        !        + surf(i, Jmax + 4, 2, 2) * surf(i, Jmax + 4, 2, 2)) - UlB * surf(i, jmax + 4, 2, 1) &
        !        / sqrt(surf(i, Jmax + 4, 2, 1) * surf(i, Jmax + 4, 2, 1) &
        !        + surf(i, Jmax + 4, 2, 2) * surf(i, Jmax + 4, 2, 2))
        !V(i, Jmax + 5) = - (UnB * surf(i, Jmax + 4, 2, 2) + UlB * surf(i, jmax + 4, 2, 1)) &
        !           / sqrt(surf(i, Jmax + 4, 2, 1) * surf(i, Jmax + 4, 2, 1) &
        !           + surf(i, Jmax + 4, 2, 2) * surf(i, Jmax + 4, 2, 2))
        U(i, Jmax + 5) = uinf
        V(i, Jmax + 5) = 0.0
        U(i, Jmax + 6) = U(i, Jmax + 5)
        U(i, Jmax + 7) = U(i, Jmax + 5)
        !V(i, Jmax + 1) = 0
        V(i, Jmax + 6) = V(i, Jmax + 5)
        V(i, Jmax + 7) = V(i, Jmax + 5)
        !C(i, Jmax + 1) = sqrt((hinf - uinf * uinf) / 2.0)
        C(i, Jmax + 5) = 0.5 * (cinf + C(i, Jmax + 4))
        rho(i, Jmax + 5) = rhoinf
        !rho(i, Jmax + 5) = pinf / (C(i, Jmax + 5) * C(i, Jmax + 5) / ga)
        E(i, Jmax + 5) = hinf - pinf / rho(i, Jmax + 5)
        !E(i, Jmax + 5) = einf
        UU(1, i, Jmax + 5) = rho(i, Jmax + 5)
        UU(2, i, Jmax + 5) = rho(i, Jmax + 5) * U(i, Jmax + 5)
        UU(3, i, Jmax + 5) = rho(i, Jmax + 5) * V(i, Jmax + 5)
        UU(4, i, Jmax + 5) = rho(i, Jmax + 5) * E(i, Jmax + 5)
        UU(:, i, Jmax + 6) = UU(:, i, Jmax + 5)! + UU(:, i, Jmax + 1) - UU(:, i, Jmax)
        UU(:, i, Jmax + 7) = UU(:, i, Jmax + 6)! + UU(:, i, Jmax + 2) - UU(:, i, Jmax + 1)
      else ! outlet condition
        !theta = atan((surf(i, Jmax, 2, 2) / surf(i, Jmax, 2, 1)))
        !write(*,*) theta
        !RnBplus = - abs(uinf * cos(theta)) + pinf / (rhoinf * cinf)
        !RnIminus = - abs(UU(2, i, Jmax - 1) / UU(1, i, Jmax - 1) * cos(theta) &
        !        + UU(3, i, Jmax - 1) / UU(1, i, Jmax - 1) * sin(theta)) &
        !        - 2 * c(i, Jmax - 1) / (ga - 1)


       ! RnBplus = - abs(uinf * surf(i, Jmax, 4, 1) / sqrt(surf(i, Jmax, 4, 1) * surf(i, Jmax, 4, 1) &
       !        + surf(i, Jmax, 4, 2) * surf(i, Jmax, 4, 2))) + pinf / rhoinf / cinf
       ! RnIminus = - abs((UU(2, i, Jmax) / UU(1, i, Jmax) * surf(i, Jmax, 4, 1) &
       !         + UU(3, i, Jmax) / UU(1, i, Jmax) * surf(i, Jmax, 4, 2)) &
       !         / sqrt(surf(i, Jmax, 4, 1) * surf(i, Jmax, 4, 1) &
       !        + surf(i, Jmax, 4, 2) * surf(i, Jmax, 4, 2))) - 2 * c(i, Jmax) / (ga - 1)
       ! UnB =  - 0.5 * (abs(uinf * surf(i, Jmax, 4, 1) / sqrt(surf(i, Jmax, 4, 1) * surf(i, Jmax, 4, 1) &
       !    + surf(i, Jmax, 4, 2) * surf(i, Jmax, 4, 2))) &
       !    + abs((UU(2, i, Jmax) / UU(1, i, Jmax) * surf(i, Jmax, 4, 1) &
       !    + UU(3, i, Jmax) / UU(1, i, Jmax) * surf(i, Jmax, 4, 2)) &
       !    / sqrt(surf(i, Jmax, 4, 1) * surf(i, Jmax, 4, 1) &
       !    + surf(i, Jmax, 4, 2) * surf(i, Jmax, 4, 2))))
        !C(i, Jmax) = 0.25 * (ga - 1) * (RnBplus - RnIminus)
        C(i, Jmax + 5) = 0.5 * (cinf + C(i, Jmax + 4))
       ! UlB = (- UU(2, i, Jmax) / UU(1, i, Jmax) *  surf(i, Jmax, 4, 2) &
       !    + UU(3, i, Jmax) / UU(1, i, Jmax) *  surf(i, Jmax, 4, 1)) &
       !    / sqrt(surf(i, Jmax, 4, 1) * surf(i, Jmax, 4, 1) &
       !        + surf(i, Jmax, 4, 2) * surf(i, Jmax, 4, 2))
        !write(*, *) UlB
        !U(i, Jmax) = - UnB * cos(theta) + UlB * sin(theta)
        !V(i, Jmax) =  UnB * sin(theta) - UlB * cos(theta)
        !U(i, Jmax + 1) = (UnB * surf(i, Jmax, 4, 1) - UlB * surf(i, Jmax, 4, 2)) &
        !       / sqrt(surf(i, Jmax, 4, 1) * surf(i, Jmax, 4, 1) &
        !       + surf(i, Jmax, 4, 2) * surf(i, Jmax, 4, 2))
```

```
        !V(i, Jmax + 1) = (UnB * surf(i, Jmax, 4, 2) + UlB * surf(i, jmax, 4, 1)) &
        !       / sqrt(surf(i, Jmax, 4, 1) * surf(i, Jmax, 4, 1) &
        !       + surf(i, Jmax, 4, 2) * surf(i, Jmax, 4, 2))
        U(i, Jmax + 5) = UU(2, i, Jmax + 4) / UU(1, i, Jmax + 4)! * 2 - UU(2, i, Jmax + 3) / UU(1, i, Jmax +
3)
        V(i, Jmax + 5) = UU(3, i, Jmax + 4) / UU(1, i, Jmax + 4)! * 2 - UU(3, i, Jmax + 3) / UU(1, i, Jmax +
3)
        !U(i, Jmax + 5) = uinf
        !V(i, Jmax + 5) = 0.0
        !U(i, Jmax + 1) = UU(2, i, Jmax) / UU(1, i, Jmax)
        !V(i, Jmax + 1) = UU(3, i, Jmax) / UU(1, i, Jmax)

        !write(*, *) V(i, Jmax)
        U(i, Jmax + 6) = U(i, Jmax + 5)! + U(i, Jmax + 1) - U(i, Jmax)
        U(i, Jmax + 7) = U(i, Jmax + 6)! + U(i, Jmax + 2) - U(i, Jmax + 1)
        V(i, Jmax + 6) = V(i, Jmax + 5)! + V(i, Jmax + 1) - V(i, Jmax)
        V(i, Jmax + 7) = V(i, Jmax + 6)! + V(i, Jmax + 2) - V(i, Jmax + 1)
        !U(i, Jmax) = UU(2, i, Jmax - 1) / UU(1, i, Jmax - 1)
        !V(i, Jmax) = UU(3, i, Jmax - 1) / UU(1, i, Jmax - 1)
        hinter = ga * UU(4, i, Jmax + 4) / UU(1, i, Jmax + 4) &
              - 0.5 * (ga - 1) * (UU(2, i, Jmax + 4) * UU(2, i, Jmax + 4) &
              + UU(3, i, Jmax + 4) * UU(3, i, Jmax + 4)) / (UU(1, i, Jmax + 4) * UU(1, i, Jmax + 4))
        !C(i, Jmax) = sqrt( 0.5 * (hinter - (UU(2, i, Jmax - 1) / UU(1, i, Jmax - 1)) * (UU(2, i, Jmax - 1) /
UU(1, i, Jmax - 1)) &
        !         + (UU(3, i, Jmax - 1) / UU(1, i, Jmax - 1)) * (UU(3, i, Jmax - 1) / UU(1, i, Jmax - 1))) * (ga -
1))
        p(i, Jmax + 5) = pinf
        rho(i, Jmax + 5) = rho(i, Jmax + 4)
        !rho(i, Jmax + 5) = p(i, Jmax + 5) / (C(i, Jmax + 5) * C(i, Jmax + 5) / ga)
        E(i, Jmax + 5) = hinter - p(i, Jmax + 4) / rho(i, Jmax + 4)
        UU(1, i, Jmax + 5) = rho(i, Jmax + 5)
        UU(2, i, Jmax + 5) = rho(i, Jmax + 5) * U(i, Jmax + 5)
        UU(3, i, Jmax + 5) = rho(i, Jmax + 5) * V(i, Jmax + 5)
        UU(4, i, Jmax + 5) = rho(i, Jmax + 5) * E(i, Jmax + 5)
        UU(:, i, Jmax + 6) = UU(:, i, Jmax + 5)
        UU(:, i, Jmax + 7) = UU(:, i, Jmax + 5)
    endif
  enddo
!stop
  do i = 5, Imax + 4 ! airfoil boundary condition
   ! if (X(i, 1) .GT. 0 .AND. Y(i, 1) .GT. 0) then
    !    theta = abs(atan((surf(i, 1, 4, 2) / surf(i, 1, 4, 1))))
    !elseif (X(i, 1) .LT. 0 .AND. Y(i, 1) .GT. 0) then
    !    theta = abs(atan((surf(i, 1, 4, 2) / surf(i, 1, 4, 1))))
    !elseif (X(i, 1) .LT. 0 .AND. Y(i, 1) .LT. 0) then
    !    theta = - abs(atan((surf(i, 1, 4, 2) / surf(i, 1, 4, 1))))
    !else
    !    theta = - abs(atan((surf(i, 1, 4, 2) / surf(i, 1, 4, 1))))
    !endif
    UnB = 0
    UlB = (UU(3, i, 5) / UU(1, i, 5) * surf(i, 5, 4, 1) - UU(2, i, 5) / UU(1, i, 5) * surf(i, 5, 4, 2)) &
    / sqrt(surf(i, 5, 4, 1) * surf(i, 5, 4, 1) + surf(i, 5, 4, 2) * surf(i, 5, 4, 2)) * 2 &
    - (UU(3, i, 6) / UU(1, i, 6) * surf(i, 6, 4, 1) - UU(2, i, 6) / UU(1, i, 6) * surf(i, 6, 4, 2)) &
    / sqrt(surf(i, 6, 4, 1) * surf(i, 6, 4, 1) + surf(i, 6, 4, 2) * surf(i, 6, 4, 2))



  p(i, 4) = p(i, 5) + p(i, 5) - p(i, 6)
  !rho(i, 4) = p(i, 5) / (C(i, 5) * C(i, 5) / ga)
  rho(i, 4) = rho(i, 5) + rho(i, 5) - rho(i, 6)

  E(i, 4) = E(i, 5) + E(i, 5) - E(i, 6)
  !E(i, 4) = 0.0
  U(i, 4) = (UnB * surf(i, 5, 4, 1) - UlB * surf(i, 5, 4, 2)) &
        / sqrt(surf(i, 5, 4, 1) * surf(i, 5, 4, 1) &
        + surf(i, 5, 4, 2) * surf(i, 5, 4, 2))
  V(i, 4) = (UnB * surf(i, 5, 4, 2) + UlB * surf(i, 5, 4, 1)) &
        / sqrt(surf(i, 5, 4, 1) * surf(i, 5, 4, 1) &
        + surf(i, 5, 4, 2) * surf(i, 5, 4, 2))
  !U(i, 4) = 0.0
  !V(i, 4) = 0.0
  !write(*, *) theta
  UU(1, i, 4) = rho(i, 4)
  UU(2, i, 4) = rho(i, 4) * U(i, 4)
  UU(3, i, 4) = rho(i, 4) * V(i, 4)
  UU(4, i, 4) = rho(i, 4) * E(i, 4)
  UU(:, i, 3) = UU(:, i, 4)! + UU(:, i, 4) - UU(:, i, 5)
  UU(:, i, 2) = UU(:, i, 3)! + UU(:, i, 3) - UU(:, i, 4)
enddo

!periodic boundary
UU(:, 4, :) = UU(:, Imax + 4, :)
UU(:, 3, :) = UU(:, Imax + 3, :)
UU(:, 2, :) = UU(:, Imax + 2, :)
UU(:, 1, :) = UU(:, Imax + 1, :)
UU(:, Imax + 5, :) = UU(:, 5, :)
UU(:, Imax + 6, :) = UU(:, 6, :)
UU(:, Imax + 7, :) = UU(:, 7, :)

!stop
! interior points
do J = 5, Jmax + 4
  do i = 5, Imax + 4
    iplus1 = i + 1
    iplus2 = i + 2
    iplus3 = i + 3
    iminus1 = i - 1
    iminus2 = i - 2
  !   if (iplus1 .GT. Imax) then! periodic boundary in i
  !       iplus1 = 1
  !       iplus2 = 2
  !       iplus3 = 3
```

```
    !    elseif (iplus2 .GT. Imax) then
    !        iplus2 = 1
    !        iplus3 = 2
    !    elseif (iplus3 .GT. Imax) then
    !        iplus3 = 1
    !    endif
    !    if (iminus1 .LT. 1) then
    !        iminus1 = Imax
    !        iminus2 = Imax - 1
    !        iminus3 = Imax - 2
    !    elseif (iminus2 .LT. 1) then
    !        iminus2 = Imax
    !        iminus3 = Imax - 1
    !    elseif (iminus3 .LT. 1) then
    !        iminus3 = Imax
    !    endif
        UUU(:) = UU(:, i, J)
    !    delt = CFL / (abs((UU(2, i, J) / UU(1, i, J) + c(i, J)) * 0.5 * (surf(i, J, 1, 1) - surf(i, J, 3, 1)) &
    !         + (UU(3, i, J) / UU(1, i, J) + c(i, J) * 0.5 * (surf(i, J, 1, 2) - surf(i, J, 3, 2)))) &
    !         + abs((UU(2, i, J) / UU(1, i, J) &
    !         + c(i, J)) * 0.5 * (surf(i, J, 2, 1) - surf(i, J, 4, 1)) &
    !         + (UU(3, i, J) / UU(1, i, J) + c(i, J) * 0.5 * (surf(i, J, 2, 2) - surf(i, J, 4, 2)))))
        do m = 1, 4 ! R-K 4 stage
            do k = 1, 2 ! k = 1 (x - direction) & k = 2 (y - direction)
                !do face = 1, 4
                    ! i + 1/2, J (face 1)
                    eps2core = 0.5 * k2 * (UU(2, i ,J) / UU(1, i, J) * surf(i, J, 1, 1) &
                            + UU(3, i ,J) / UU(1, i, J) * surf(i, J, 1, 2) &
                            + c(i, J) * sqrt(surf(i, J, 1, 1) * surf(i, J, 1, 1) &
                            + surf(i, J, 1, 2) * surf(i, J, 1, 2)))
                    v1 = abs((P(i, J) - 2 * P(iminus1, J) + P(iminus2, J)) / (P(i, J) + 2 * P(iminus1, J) +
P(iminus2, J)))
                    v2 = abs((P(iplus1, J) - 2 * P(i, J) + P(iminus1, J)) / (P(iplus1, J) + 2 * P(i, J) + P(iminus1,
J)))
                    v3 = abs((P(iplus2, J) - 2 * P(iplus1, J) + P(i, J)) / (P(iplus2, J) + 2 * P(iplus1, J) + P(i, J)))
                    v4 = abs((P(iplus3, J) - 2 * P(iplus2, J) + P(iplus1, J)) / (P(iplus3, J) + 2 * P(iplus2, J) +
P(iplus1, J)))
                    eps2 = eps2core * max(v1, v2, v3, v4);
                    eps4 = max(0.0, (eps2core / k2 * k4 - eps2))
                    !eps4 = eps2core / k2 * k4
                    D(1, :) = eps2 * (UU(:, iplus1, J) - UU(:, i, J)) - eps4 * (UU(:, iplus2, J) &
                            - 3 * UU(:, iplus1, J) + 3 * UU(:, i, J) - UU(:, iminus1, J))
                        !write(*, *)
                        !stop
                    !D(1, :) = eps4 * (UU(:, iplus2, J) &
                    !        - 3 * UU(:, iplus1, J) + 3 * UU(:, i, J) - UU(:, iminus1, J))
                    !if (J .EQ. 5) then
                    !    D(1, :) = 0
                    !endif
                    FS(i, J, 1, k, :) = 0.5 * (Fstar(k, :, i, J) + Fstar(k, :, iplus1, J)) &
                                    * surf(i, J, 1, k) - D(1, :)
                    ! i - 1/2, J (face 3)
                    eps2core = 0.5 * k2 * (UU(2, i ,J) / UU(1, i, J) * surf(i, J, 3, 1) &
                            + UU(3, i ,J) / UU(1, i, J) * surf(i, J, 3, 2) &
                            + c(i, J) * sqrt(surf(i, J, 3, 1) * surf(i, J, 3, 1) &
                            + surf(i, J, 3, 2) * surf(i, J, 3, 2)))
                    v1 = abs((P(iminus1, J) - 2 * P(iminus2, J) + P(iminus3, J)) &
                      / (P(iminus1, J) + 2 * P(iminus2, J) + P(iminus3, J)))
                    v2 = abs((P(i, J) - 2 * P(iminus1, J) + P(iminus2, J)) / (P(i, J) + 2 * P(iminus1, J) +
P(iminus2, J)))
                    v3 = abs((P(iplus1, J) - 2 * P(i, J) + P(iminus1, J)) / (P(iplus1, J) + 2 * P(i, J) + P(iminus1,
J)))
                    v4 = abs((P(iplus2, J) - 2 * P(iplus1, J) + P(i, J)) / (P(iplus2, J) + 2 * P(iplus1, J) + P(i, J)))
                    eps2 = eps2core * max(v1, v2, v3, v4);
                    eps4 = max(0.0, (eps2core / k2 * k4 - eps2))
                    !eps4 = eps2core / k2 * k4
                    D(3, :) = eps2 * (UU(:, i, J) - UU(:, iminus1, J)) - eps4 * (UU(:, iplus1, J) &
                        - 3 * UU(:, i, J) + 3 * UU(:, iminus1, J) - UU(:, iminus2, J))
                        !write(*, *)
                        !stop
                    !D(3, :) = eps4 * (UU(:, iplus1, J) &
                    !          - 3 * UU(:, i, J) + 3 * UU(:, iminus1, J) - UU(:, iminus2, J))
                    !if (J .EQ. 5) then
                    !    D(3, :) = 0
                    !endif
                    FS(i, J, 3, k, :) = 0.5 * (Fstar(k, :, iminus1, J) + Fstar(k, :, i, J)) &
                                   * surf(i, J, 3, k) - D(3, :)

                    ! i, J + 1/2 (face2)
                    eps2core = 0.5 * k2 * (UU(2, i ,J) / UU(1, i, J) * surf(i, J, 2, 1) &
                            + UU(3, i ,J) / UU(1, i, J) * surf(i, J, 2, 2) &
                            + c(i, J) * sqrt(surf(i, J, 2, 1) * surf(i, J, 2, 1) &
                            + surf(i, J, 2, 2) * surf(i, J, 2, 2)))
                    v1 = abs((P(i, J) - 2 * P(i, J - 1) + P(i, J - 2)) &
                      / (P(i, J) + 2 * P(i, J - 1) + P(i, J - 2)))
                    v2 = abs((P(i, J + 1) - 2 * P(i, J) + P(i, J - 1)) / (P(i, J + 1) + 2 * P(i, J) + P(i, J - 1)))
                    v3 = abs((P(i, J + 2) - 2 * P(i, J + 1) + P(i, J)) / (P(i, J + 2) + 2 * P(i, J + 1) + P(i, J)))
                    v4 = abs((P(i, J + 3) - 2 * P(i, J + 2) + P(i, J + 1)) / (P(i, J + 3) + 2 * P(i, J + 2) + P(i, J + 1)))

                    eps2 = eps2core * max(v1, v2, v3, v4);
                    eps4 = max(0.0, (eps2core / k2 * k4 - eps2))
                    !eps4 = eps2core / k2 * k4
                    D(2, :) = eps2 * (UU(:, i, J + 1) - UU(:, i, J)) - eps4 * (UU(:, i, J + 2) &
                        - 3 * UU(:, i, J + 1) + 3 * UU(:, i, J) - UU(:, i, J - 1))
                        !write(*, *)
                        !stop
              ! if (J .GT. 1) then
                    !    FS(i, J, 2, k, :) = 0.5 * (Fstar(k, :, i, J - 1) + Fstar(k, :, i, J)) &
                    !                  * surf(i, J, 2, k) - D(2, :)
                    !else
                    !D(2, :) = eps4 * (UU(:, i, J + 2) &
                    !       - 3 * UU(:, i, J + 1) + 3 * UU(:, i, J) - UU(:, i, J - 1))
                    !if (J .EQ. 5) then
                    !   D(2, :) = 0
                    !endif
                    FS(i, J, 2, k, :) = 0.5 * (Fstar(k, :, i, J + 1) + Fstar(k, :, i, J)) &
                                   * surf(i, J, 2, k) - D(2, :)
                    !endif

                  ! i, J - 1/2 (face4)
                    eps2core = 0.5 * k2 * (UU(2, i ,J) / UU(1, i, J) * surf(i, J, 4, 1) &
                            + UU(3, i ,J) / UU(1, i, J) * surf(i, J, 4, 2) &
                            + c(i, J) * sqrt(surf(i, J, 4, 1) * surf(i, J, 4, 1) &
                            + surf(i, J, 4, 2) * surf(i, J, 4, 2)))
                    v1 = abs((P(i, J - 1) - 2 * P(i, J - 2) + P(i, J - 3)) &
                      / (P(i, J - 1) + 2 * P(i, J - 2) + P(i, J - 3)))
                    v2 = abs((P(i, J) - 2 * P(i, J - 1) + P(i, J - 2)) / (P(i, J) + 2 * P(i, J - 1) + P(i, J - 2)))
                    v3 = abs((P(i, J + 1) - 2 * P(i, J) + P(i, J - 1)) / (P(i, J + 1) + 2 * P(i, J) + P(i, J - 1)))
                    v4 = abs((P(i, J + 2) - 2 * P(i, J + 1) + P(i, J)) / (P(i, J + 2) + 2 * P(i, J + 1) + P(i, J)))
                    eps2 = eps2core * max(v1, v2, v3, v4);
                    eps4 = max(0.0, (eps2core / k2 * k4 - eps2))
                    !eps4 = eps2core / k2 * k4
                    D(4, :) = eps2 * (UU(:, i, J) - UU(:, i, J - 1)) - eps4 * (UU(:, i, J + 1) &
                            - 3 * UU(:, i, J) + 3 * UU(:, i, J - 1) - UU(:, i, J - 2))
                        !write(*, *)
                        !stop
              ! if (J .GT. 1) then
                    !   FS(i, J, 4, k, :) = 0.5 * (Fstar(k, :, i, J - 1) + Fstar(k, :, i, J)) &
                     !                * surf(i, J, 2, k) - D(4, :)
                    !else
                    !D(4, :) = eps4 * (UU(:, i, J + 1) &
                    !       - 3 * UU(:, i, J) + 3 * UU(:, i, J - 1) - UU(:, i, J - 2))
                    !if (J .EQ. 5) then
                    !   D(4, :) = 0
                    !endif
                       FS(i, J, 4, k, :) = 0.5 * (Fstar(k, :, i, J - 1) + Fstar(k, :, i, J)) &
                                     * surf(i, J, 4, k) - D(4, :)
                    !endif

                 !enddo
                 ! flux of 4 faces of cell i, J
                 Fsum(i, J, k, :) = FS(i, J, 1, k, :) + FS(i, J, 2, k, :) + FS(i, J, 3, k, :) + FS(i, J, 4, k, :)
           enddo
              Res(i, J, :) = (Fsum(i, J, 1, :) + Fsum(i, J, 2, :))! + Fsum(i, J, 1, 2) + Fsum(i, J, 2, 2) &
                     !+ Fsum(i, J, 1, 3) + Fsum(i, J, 2, 3) + Fsum(i, J, 1, 4) + Fsum(i, J, 2, 4))
               !Res(i, J) = abs(Fsum(i, J, 1, 1) + Fsum(i, J, 2, 1))
               !write(*, 601) Res(i, J)

              P(i, J) = (ga - 1) * (UU(4, i, J) &
                - 0.5 * (UU(2, i, J) * UU(2, i, J) + UU(3, i, J) * UU(3, i, J)) / UU(1, i, J))

              rho(i, J) = UU(1, i, J)
```

```
        c(i, J) = sqrt(ga * p(i, J) / rho(i, J))

        if (Y(i, J) .GT. 0) then
        delt = CFL / (abs((UU(2, i, J) / UU(1, i, J) + c(i, J)) * 0.5 * (surf(i, J, 1, 1) - surf(i, J, 3, 1)) &
            + (UU(3, i, J) / UU(1, i, J) + c(i, J)) * 0.5 * (surf(i, J, 1, 2) - surf(i, J, 3, 2))) &
            + abs((UU(2, i, J) / UU(1, i, J) &
            + c(i, J)) * 0.5 * (surf(i, J, 2, 1) - surf(i, J, 4, 1)) &
            + (UU(3, i, J) / UU(1, i, J) + c(i, J)) * 0.5 * (surf(i, J, 2, 2) - surf(i, J, 4, 2))))
        else
        delt = CFL / (abs((UU(2, i, J) / UU(1, i, J) + c(i, J)) * 0.5 * (surf(i, J, 1, 1) - surf(i, J, 3, 1)) &
            + (UU(3, i, J) / UU(1, i, J) - c(i, J)) * 0.5 * (surf(i, J, 1, 2) - surf(i, J, 3, 2))) &
            + abs((UU(2, i, J) / UU(1, i, J) &
            + c(i, J)) * 0.5 * (surf(i, J, 2, 1) - surf(i, J, 4, 1)) &
            + (UU(3, i, J) / UU(1, i, J) - c(i, J)) * 0.5 * (surf(i, J, 2, 2) - surf(i, J, 4, 2))))
        endif

        !if (J .EQ. 30) then
        !    write(*, *) delt
        !    stop
        !endif

        UU(:, i, J) = UUU(:) - delt * al(m) * Res(i, J, :) ! update U matrix

        !write(*, *) FF(2, i, J)
        P(i, J) = (ga - 1) * (UU(4, i, J) &
          - 0.5 * (UU(2, i, J) * UU(2, i, J) + UU(3, i, J) * UU(3, i, J)) / UU(1, i, J))

        H(i, J) = ga * UU(4, i, J) / UU(1, i, J) &
          - 0.5 * (ga - 1) * (UU(2, i, J) * UU(2, i, J) &
          + UU(3, i, J) * UU(3, i, J)) / (UU(1, i, J) * UU(1, i, J))

        rho(i, J) = UU(1, i, J)

        c(i, J) = sqrt(ga * p(i, J) / rho(i, J))

        FF(1, i, J) = UU(2, i, J)
        FF(2, i, J) = UU(2, i, J) * UU(2, i, J) / UU(1, i, J) + P(i, J)
        FF(3, i, J) = UU(2, i, J) * UU(3, i, J) / UU(1, i, J)
        FF(4, i, J) = UU(2, i, J) * H(i, J)
        GG(1, i, J) = UU(3, i, J)
        GG(2, i, J) = FF(3, i, J)
        GG(3, i, J) = UU(3, i, J) * UU(3, i, J) / UU(1, i, J) + P(i, J)
        GG(4, i, J) = UU(3, i, J) * H(i, J)
        Fstar(1, :, i, J) = FF(:, i, J)
        Fstar(2, :, i, J) = GG(:, i, J)
    enddo
   ! stop
   !write(*, *) UU(4, i, J)
enddo
!stop
```

```
    !periodic boundary
          UU(:, 4, J) = UU(:, Imax + 4, J)
          UU(:, 3, J) = UU(:, Imax + 3, J)
          UU(:, 2, J) = UU(:, Imax + 2, J)
          UU(:, 1, J) = UU(:, Imax + 1, J)
          UU(:, Imax + 5, J) = UU(:, 5, J)
          UU(:, Imax + 6, J) = UU(:, 6, J)
          UU(:, Imax + 7, J) = UU(:, 7, J)
  enddo


  write(*, *) n, sum(Res) !Res(30, 6, 3)! - Res(98, 6, 2)
  write(30,601) sum(Res)
  !CLOSE (30, STATUS='KEEP')

enddo

  open(40,file='velocity.dat')
  write(40,*) 'VARIABLES = X", "Y", "U", "V", "P", "Res", "delt", "C"'
  write(40,*) 'ZONE I=', Imax,   ', J=', Jmax, 'F=POINT'
  do J=5 ,Jmax + 4
    do i=5 ,Imax + 4
      write(40,603) X(i, J), Y(i, J), UU(2, i, J) / UU(1, i, J), UU(3, i, J) / UU(1, i, J), P(i, J), Res(i, J, 1) &
      + Res(i, J, 2) + Res(i, J, 3) + Res(i, J, 4), deltt(i, J), c(i, J)
    enddo
  enddo

!  602 format(2e16.8)
  601 format(E16.8)
  603 format(8e16.8)
end program
```