DEPARTMENT OF MECHANICAL ENGINEERING AND MECHANICS
COLLEGE OF ENGINEERING AND TECHNOLOGY
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529

INTERACTIVE REAL TIME FLOW SIMULATIONS

P. 47

By

I. Sadrehaghighi, Graduate Research Assistant

and

S. N. Tiwari, Principal Investigator

Progress Report
For the period ended September 30, 1990

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Under
Research Grant NCC1-68
Dr. Robert E. Smith, Jr., Technical Monitor
ACD-Computer Applications Branch

October 1990

DEPARTMENT OF MECHANICAL ENGINEERING AND MECHANICS
COLLEGE OF ENGINEERING AND TECHNOLOGY
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529

INTERACTIVE REAL TIME FLOW SIMULATIONS

By

I. Sadrehaghighi, Graduate Research Assistant

and

S. N. Tiwari, Principal Investigator

Progress Report
For the period ended September 30, 1990

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Under
Research Grant NCC1-68
Dr. Robert E. Smith, Jr., Technical Monitor
ACD-Computer Applications Branch

Submitted by the
Old Dominion University Research Foundation
P.O. Box 6369
Norfolk, Virginia 23508-0369

October 1990

ABSTRACT

INTERACTIVE REAL TIME FLOW SIMULATIONS

Ideen Sadrehaghighi
Surendra N. Tiwari

An interactive real time flow simulation technique is developed for an unsteady channel flow. A finite-volume algorithm in conjunction with a Runge-Kutta time stepping scheme has been developed for two-dimensional Euler equations. A global time step has been used to accelerate convergence of steady-state calculations. A raster image generation routine has been developed for high speed image transmission which allows user to have direct interaction with the solution development. In addition to theory and results, the hardware and software requirements are discussed.

ACKNOWLEDGMENTS

This is a progress report on the research project "Numerical Solutions of Three-Dimensional Navier-Stokes Equations for Closed-Bluff Bodies" for the period ended September 30, 1990. Specific efforts during this period were directed in the area of "Interactive Real Time Flow Simulations."

This work was supported by the NASA Langley Research Center through Cooperative Agreement NCC1-68. The cooperative agreement was monitored by Dr. Robert E. Smith, Jr., of the Analysis and Computation Division (Computer Applications Branch), NASA Langley Research Center, Mail Stop 125.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

$\mathbf{Q}$	= vector of conservative variables
$\mathbf{F}$,$\mathbf{G}$	= flux vectors for coordinate directions
x , y	= physical coordinates
h	= height of channel
$\bar{x}$, $\bar{y}$	= computational coordinates
u , v	= velocity components in x and y directions
p	= static pressure
M_∞	= free stream Mach number
ρ	= density
E	= total energy
e	= energy per unit volume
S	= cell area
γ	= ratio of specific heats
α	= angle of attack
β	= stretching parameter
η	= clustering parameter
δ	= boundary-layer thickness

Chapter 1

INTRODUCTION

Since an interactive design process is one of the ultimate goals of Computational Fluid Dynamics (CFD), real time flow simulation with direct user interaction is an ideal approach in achieving this goal [1]*. This process requires a supercomputer with vast amount of memory, an extremely high bandwidth communication network and highly capable graphic workstations. Even with today's rapid advances in the supercomputer developments, some of the components are not fast or large enough for a realistic three-dimensional problem. However, real time simulation of medium size two-dimensional flow problems (Euler equations) are possible today.

Accurate simulations are critical to the development and understanding of highly unsteady flow. For the reasons outlined above, a simple unsteady two-dimensional channel type flow has been studied here. The relative motion of the shocks and other strong gradients have been examined as the solution being computed. Once the solution has been examined, the user will inform the flow solver to take different action or to continue on the existing course. This requires large amount of data to be examined by the user, and small amount of information in form of instruction, to be send back to the mainframe computer. Consequently, this procedure requires a fast mainframe computer, an extremely fast network for data communication and a relatively fast workstation.

The computational process is initiated on the mainframe computer under

* The numbers in brackets indicate references.

interactive control by the user at a workstation. The equations of motion are integrated step by step on the mainframe computer. The choice of a variable to be visualized is instructed from the workstation and a separate rasterization program computes a raster image of the variable. The image is transmitted over the communication link to a raster display device to be viewed by the user. Images are continuously created, transmitted, and if desired, stored on a recording device. Having viewed the images on the raster display device, the user responds. If mainframe's computational and communication rates are sufficient, a real time interaction can be achieved. This, of course , depends on the size of the problem under investigation (i.e. number of grid points, equations of motion, and solution technique) .

Chapter 2

GRID GENERATION

A suitable transformation for a channel type flow can be obtained using an analytical function. A simple case would be to use uniform spacing in x direction. For y direction, the spacing is obtained by [2],

$$y = h\frac{(\beta+2\eta)[(\beta+1)/(\beta-1)]^{(\bar{y}-\eta)/(1-\eta)} - \beta + 2\eta}{(2\eta+1)(1+[(\beta+1)/(\beta-1)]^{(\bar{y}-\eta)/(1-\eta)})} \tag{2.1}$$

where h is the height of channel and η is the parameter which controls the grid clustering in y direction. Variable $\bar{y}$ corresponds to normal coordinate in the computational domain. The stretching parameter, β, is related approximately to the non-dimensional boundary layer thickness (δ/h) by

$$\beta = (1-\frac{\delta}{h})^{\frac{-1}{2}} \quad , \quad 0 < \frac{\delta}{h} < 1 \tag{2.2}$$

The amount of stretching for various values of δ/h is illustrated in Fig. 2.1. For this transformation, if $\eta = 0$ the mesh will be refined near y = h , whereas, if $\eta = 0.5$ the mesh will be refined equally near y = 0 and y = h . For this study, values of $\eta = 0.5$ and β= 0.30 been chosen and the resulting grid is shown in Fig. 2.2.

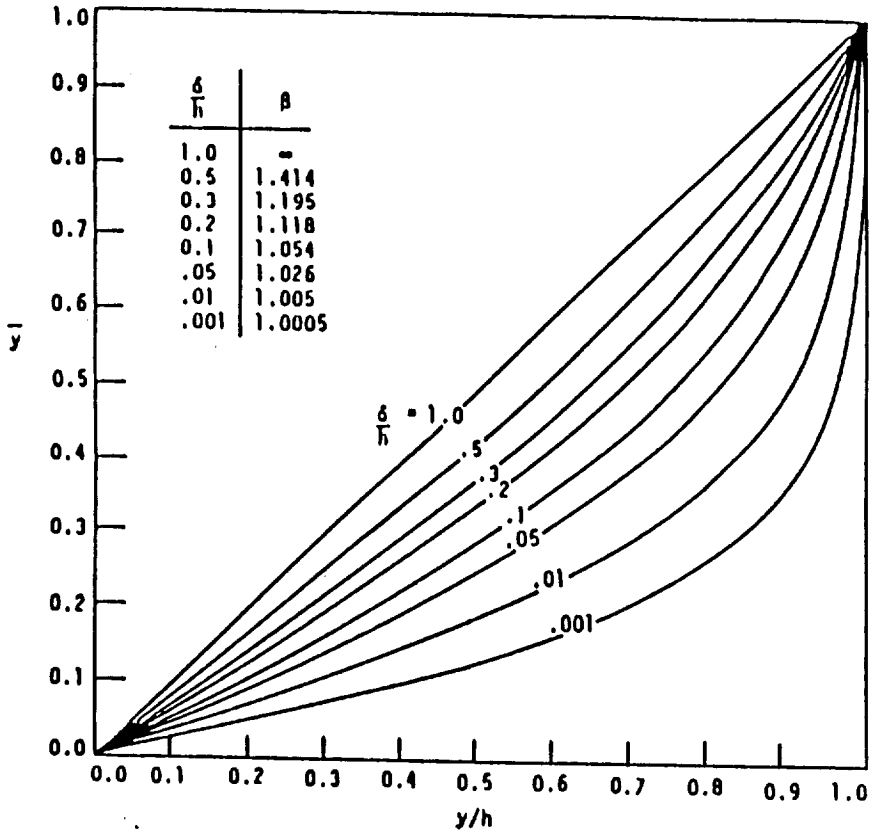


Fig. 2.1 Stretching transformation ($\eta = 0$)

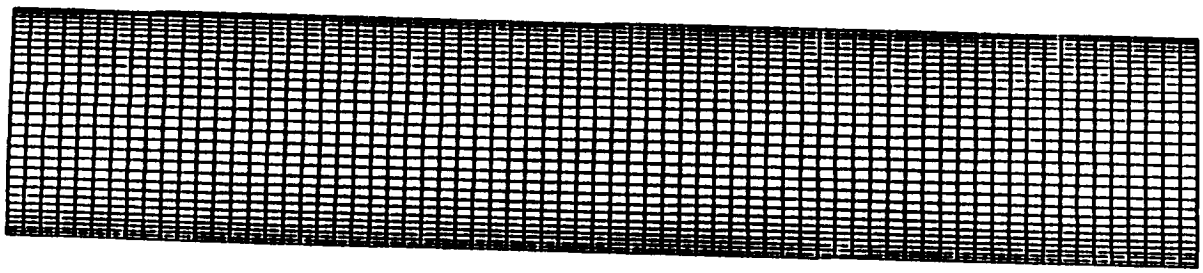


Fig. 2.2 Computational grid for a channel flow

Chapter 3

EQUATIONS OF MOTION AND METHOD OF SOLUTION

3.1 Governing Equations

The governing equations are time-dependent Euler equations [3]. The unsteady two-dimensional equations for a compressible perfect gas can be written in an integral form for a region Ω with boundary $\delta\Omega$ as

$$\frac{\partial}{\partial t}\int\int_{\Omega}\mathbf{Q}\,dxdy + \int_{\delta\Omega}(\mathbf{F}dy - \mathbf{G}dx) = 0 \tag{3.1}$$

where vectors $\mathbf{Q}$, $\mathbf{F}$, and $\mathbf{G}$, are

$$\mathbf{Q} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix} \quad \mathbf{F} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (E+p)u \end{bmatrix} \quad \mathbf{G} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (E+p)v \end{bmatrix} \tag{3.2}$$

The total energy, E, is defined using the ideal gas relation as

$$E = \frac{p}{(\gamma-1)} + \frac{1}{2}\rho(u^2+v^2) \tag{3.3}$$

where γ is the ratio of specific heats.

3.2 Finite-Volume Scheme

The governing equations in the integral form are applied to an arbitrary quadrilateral and the line integrals are approximated with the midpoint rule. The discretized result is

$$\frac{d}{dt}(S_{ij}\mathbf{Q}_{i,j}) + \angle\mathbf{Q}_{i,j} = 0 \tag{3.4}$$

where $\angle$ is defined as the spatial discretization operator and S_{ij} is area of the cell. The components of $\mathbf{Q}_{i,j}$ represent the cell-averaged quantities and obtained by the mean values of the fluxes crossing the cell.

In order to overcome the instabilities associated with central differencing, a fourth order dissipative model is used. In this study, the dissipative model developed by Jameson, Schmidt, and Turkel [4] has been used. The finite-volume formulation, Eq. (3.4) is now expressed as

$$\frac{d}{dt}(S_{i,j}\mathbf{Q}_{i,j}) + \angle\mathbf{Q}_{i,j} - D\mathbf{Q}_{i,j} = 0 \tag{3.5}$$

where D represents the artificial dissipation operator. This operator can be written in the following way

$$D\mathbf{Q}_{i,j} = D_x\mathbf{Q}_{i,j} + D_y\mathbf{Q}_{i,j} \tag{3.6}$$

where $D_x\mathbf{Q}_{i,j}$ and $D_y\mathbf{Q}_{i,j}$ are the contributions from each of the coordinate directions. In conservation form

$$D_x\mathbf{Q}_{i,j} = \mathbf{d}_{i+1/2,j} - \mathbf{d}_{i-1/2,j} \tag{3.7}$$

$$D_y\mathbf{Q}_{i,j} = \mathbf{d}_{i,j+1/2} - \mathbf{d}_{i,j-1/2} \tag{3.8}$$

where the terms on the right hand side have the form

$$\mathbf{d}_{i+1/2,j} = \frac{S_{i+1/2,j}}{\Delta t}[\varepsilon^{(2)}_{i+1/2,j}(\mathbf{Q}_{i+1,j} - \mathbf{Q}_{i,j}) - \varepsilon^{(4)}_{i+1/2,j}(\mathbf{Q}_{i+2,j} - 3\mathbf{Q}_{i+1,j} + 3\mathbf{Q}_{i,j} - \mathbf{Q}_{i-1,j})] \tag{3.9}$$

The coefficients $\varepsilon^{(2)}$ and $\varepsilon^{(4)}$ are determined from the pressures gradients as

$$\nu_{i,j} = \frac{\mid P_{i+1,j} - 2P_{i,j} + P_{i-1,j} \mid}{P_{i+1,j} + 2P_{i,j} + P_{i-1,j}} \tag{3.10}$$

$$\varepsilon^{(2)}_{i+1/2,j} = \alpha^{(2)} max(\nu_{i+1}, \nu_{i,j}) \tag{3.11}$$

$$\varepsilon^{(4)}_{i+1/2,j} = max(0, (\alpha^{(4)} - \varepsilon^{(2)}_{i+1/2,j})) \tag{3.12}$$

where typical values of the constants $\alpha^{(2)}$ and $\alpha^{(4)}$ are $\frac{1}{4}$ and $\frac{1}{256}$, respectively.

3.3 Time-Stepping Scheme

A modified four stage Runge-Kutta time stepping scheme was used to advance the solution in time. Due to its explicit character, this scheme is very simple and flexible . Hence, it is ideal for interactive computation purposes. At time level n,

$$\mathbf{Q}^{(0)} = \mathbf{Q}^{(n)} \tag{3.13}$$

$$\mathbf{Q}^{(1)} = \mathbf{Q}^{(0)} - \alpha_1 \Delta t R\mathbf{Q}^{(0)} \tag{3.14}$$

$$\mathbf{Q}^{(2)} = \mathbf{Q}^{(0)} - \alpha_2 \Delta t R\mathbf{Q}^{(1)} \tag{3.15}$$

$$\mathbf{Q}^{(3)} = \mathbf{Q}^{(0)} - \alpha_3 \Delta t R\mathbf{Q}^{(2)} \tag{3.16}$$

$$\mathbf{Q}^{(4)} = \mathbf{Q}^{(0)} - \Delta t R\mathbf{Q}^{(3)} \tag{3.17}$$

$$\mathbf{Q}^{(n+1)} = \mathbf{Q}^{(4)} \tag{3.18}$$

where on the (q+1)st stage

$$R\mathbf{Q}^{(q)} = \frac{1}{S}(L\mathbf{Q}^{(q)} - D\mathbf{Q}^{(0)}) \tag{3.19}$$

and

$$\alpha_1 = \frac{1}{4} \quad , \quad \alpha_2 = \frac{1}{3} \quad , \quad \alpha_3 = \frac{1}{2} \tag{3.20}$$

3.4 Boundary Conditions

The boundary conditions are implemented by applying a slip velocity condition at the solid walls [5]. Also, the stagnation enthalpy is assumed to be constant along the solid surface and equal to that of the free-stream, i.e.,

$$h_t = h + \frac{1}{2}(u^2 + v^2) = h_{t\infty} \tag{3.21}$$

For solid wall, $(h_t)_{wall}$, the free-stream values(u=1 and v=0) are used, i.e.,

$$(h_t)_{wall} = \gamma e_{F.S.} + \frac{1}{2} \tag{3.22}$$

Because of the supersonic nature of the flow, the outflow boundary is determined by extrapolating from interior points.

Chapter 4

SOFTWARE AND HARDWARE CONSIDERATIONS

The implemented software and hardware are discussed in [1]. Currently, the mainframe computer is a CRAY-2 which generally performs at 200-250 MFLOPS (Million Floating Point Operations Per Second) . The mainframe is connected to UltraNet Graphics Display Device (UGDD), which is a network based frame buffer through a High-Speed Channel (HSC) . The channel can support transfer rates up to 100 MBytes/sec. The frame buffer is a high-speed display system supporting high resolution monitors. The UGDD contains two memory arrays. While one array is displayed, the other is updated over the network. The user determines which array to display and controls the situation over the network. The UGDD supports a single user at a time and may be used to display color images at animation rates. The frame buffer can display RGB (Red, Green, Blue) images up to 1280 pixels horizontally by 1024 Scan-lines vertically. Figure 4.1 shows a pictorial representation of the hardware setup for a single hub (UltraNet 1000). Figure 4.2 shows the configuration using a CRAY supercomputer in conjunction with several mini-supercomputers and workstations.

There are several pieces of software required for a real time CFD simulation. They are : a grid generator, a flow solver, an image rasterizer, and an image manipulator. Each software component has been written and applied separately. However, for a true interactive process, these softwares must be integrated into one working

unit or be presented in parallel.

With the current CFD technology, it is possible to solve the Euler equations at $16\frac{\mu sec}{(gridpoints)*(iterations)}$ [6]. For a grid of 70 X 30, it will take one second for 62 iterations. This will result in 62 frames per second.

A rasterization program called PLOTD is developed to produce a raster image from the solution. A complete listing of the source code is provided in Appendix A. This software converts a specified variable defined on a surface to a raster RGB images (1024 X 768). The required CPU time for this step depends on the size of the grid and the complexity of the image. This step may take from few microseconds to a couple of seconds. Consequently, the image must not be very complex.

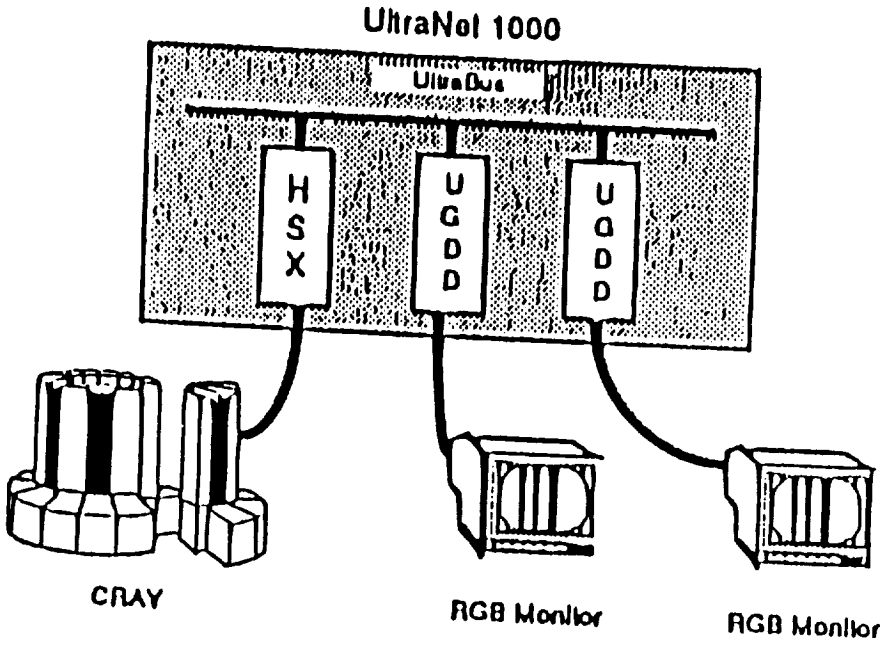


Fig. 4.1 CRAY and UltraNet Graphics Display Device

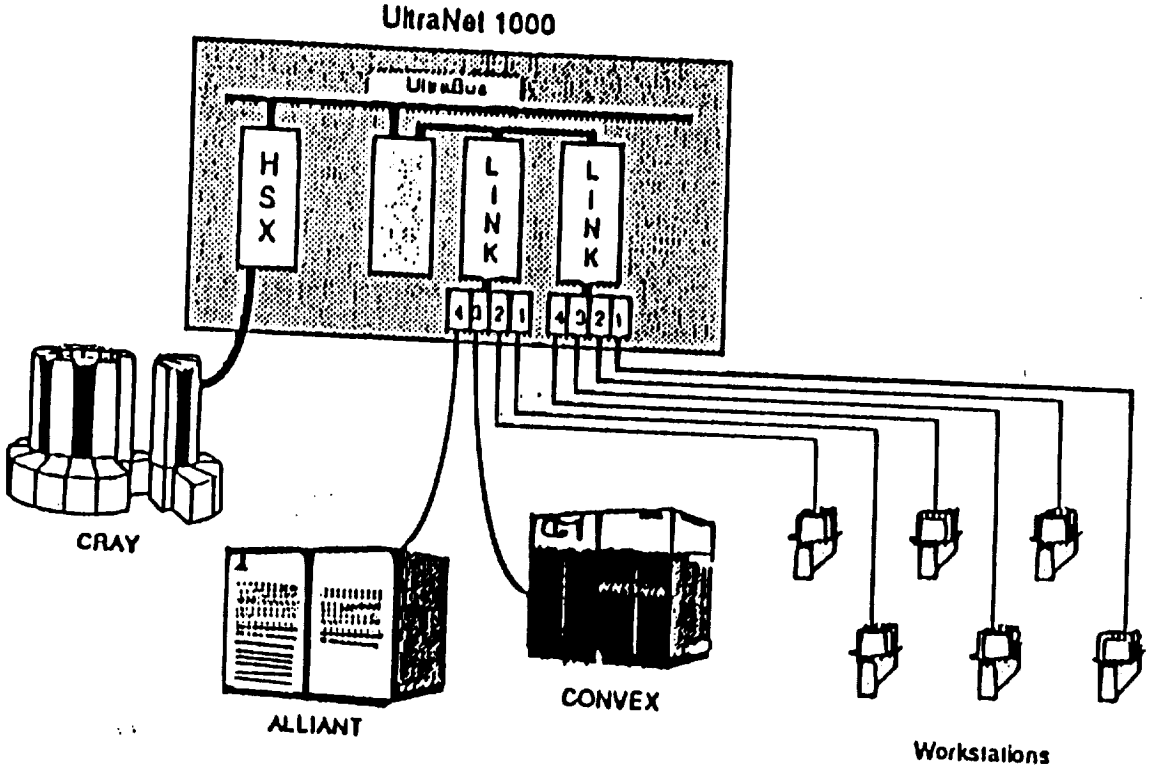


Fig. 4.2 High performance multicomputer network

Chapter 5

RESULTS AND DISCUSSION

The unsteady two-dimensional Euler equations are solved for a channel flow. In order to test the flow solver, the solution has been obtained by keeping all the flow parameters constant. The results are satisfactory and the shocks are captured very accurately. Figures 5.1 through 5.4 show pressure and Mach number contours at different iterations for different angles of attack. A more comprehensive test can be achieved by changing one of the flow conditions as the solution evolves. An ideal and relatively simple case would be rotating the angle of attack. The angle of attack can be expressed as

$$\alpha = \alpha_{min} + (\alpha_{max} - \alpha_{min}) sin(\omega \pi) \tag{5.1}$$

where α_{min} and α_{max} are intial and final angles of attack, and ω is the specified frequency. Initially, the solution starts with a uniform flow everywhere (impulse start). At this initial stage of the computation, none of the flow features has been altered. This, in fact, is the continuation of the case described earlier for testing the flow solver. After some iterations, once the flow features are established, the angle of attack is allowed to rotate using Eq. (5.1). It takes between 3-5 cycles for flow properties to reach a cyclic behavior.

Each frame takes at least 0.5 second of CPU time, which is relatively long time for a multiuser machine. One way to alleviate this is to take advantage of the parallel capability of the CRAY 2. This can be accomplished by allowing the flow

solution, rasterization and image transmission to be performed on different processors.

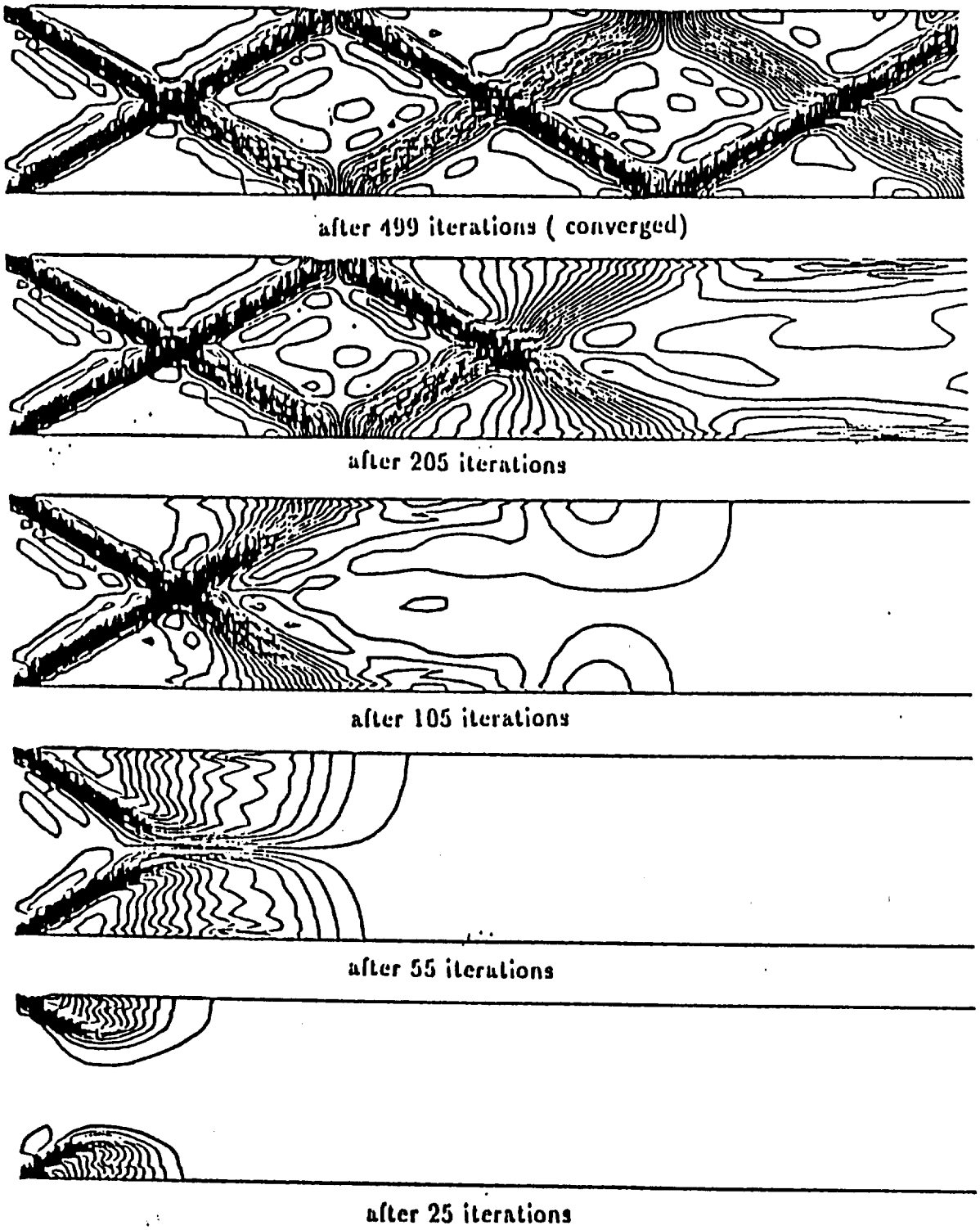


Fig. 5.1 Pressure contours for different iterations ($\alpha = 0.8^{\circ}$, $M_{\infty} = 2.0$)

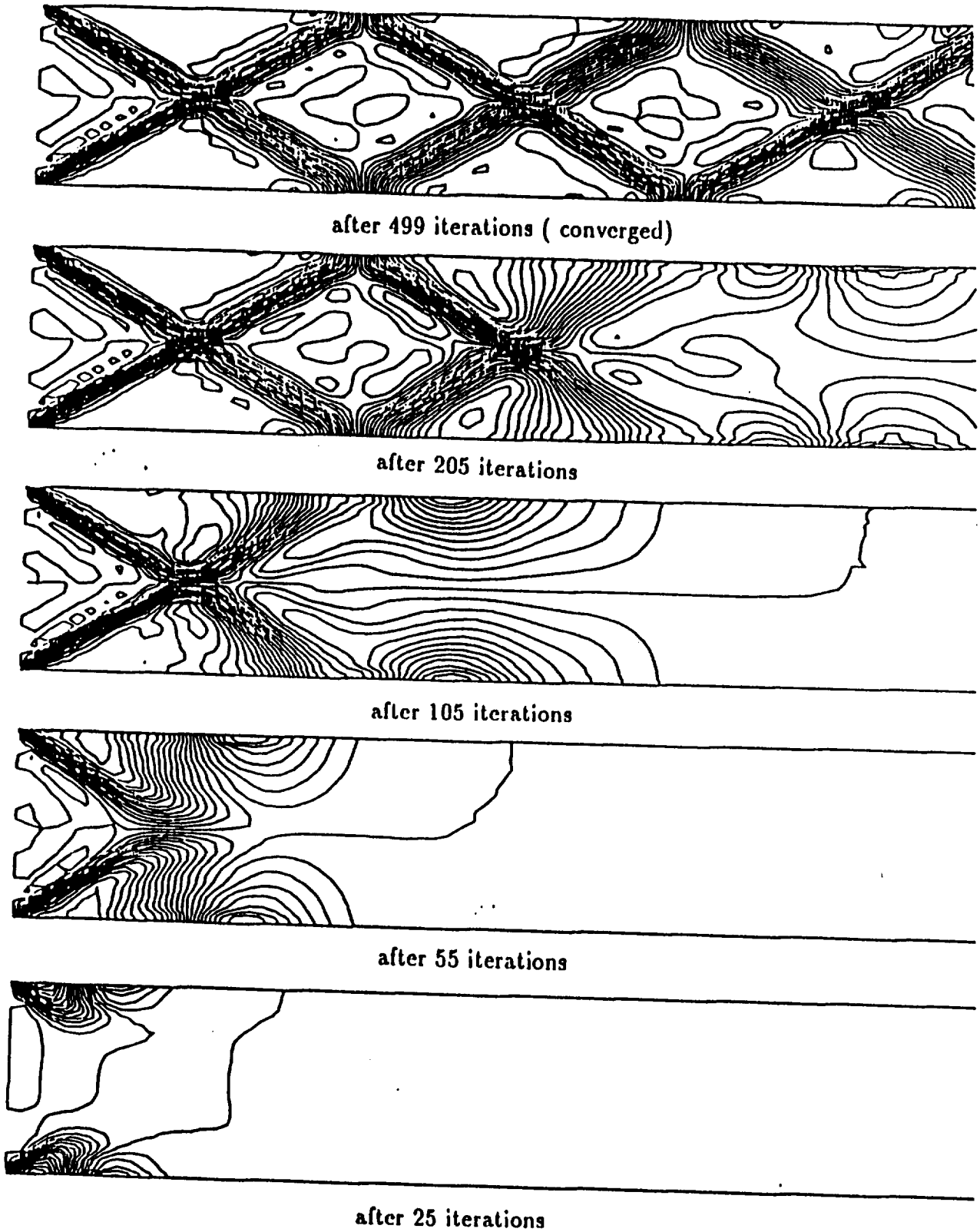


Fig. 5.2 Mach number contours for different iterations ($\alpha = 0.8^{\circ}$, $M_{\infty} = 2.0$)

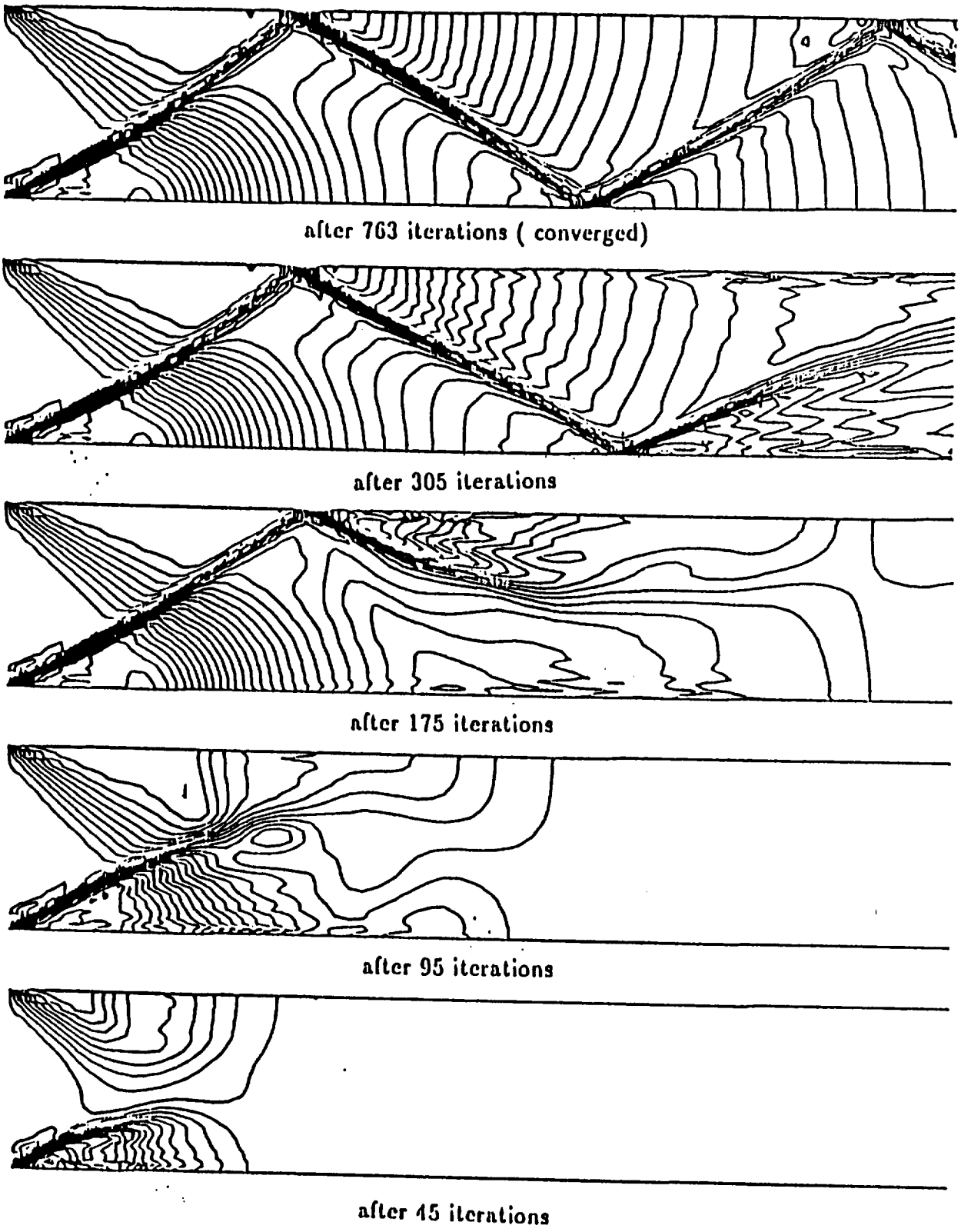
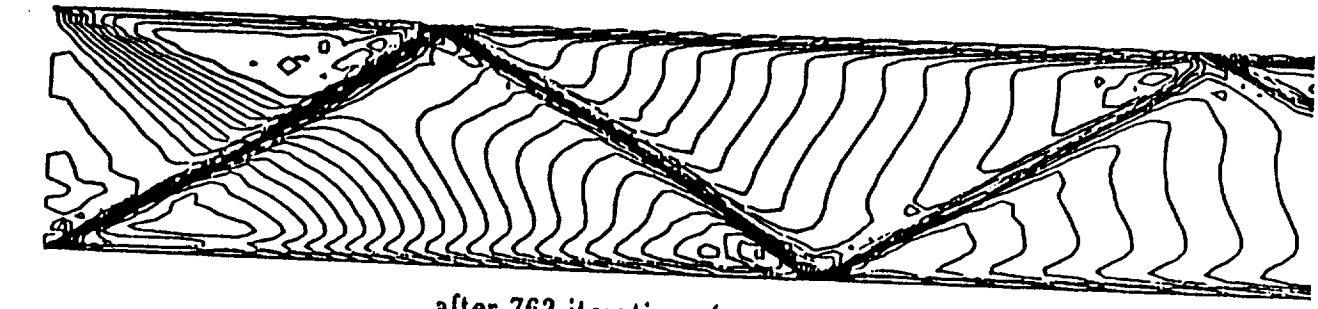
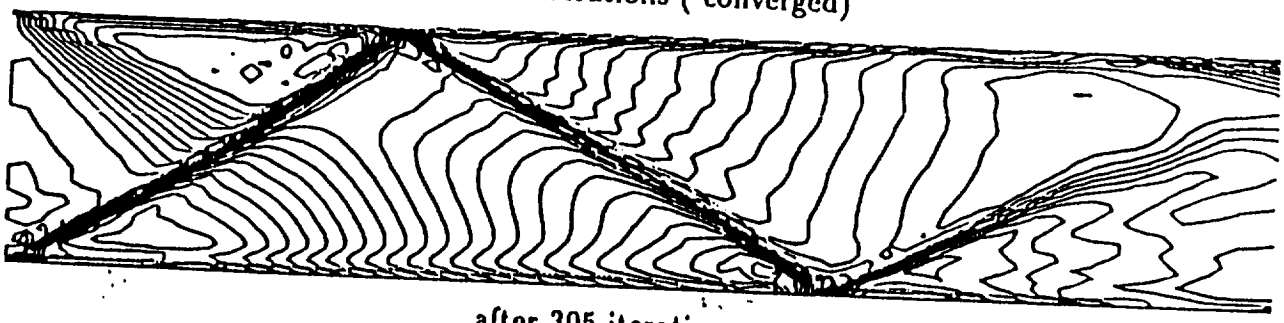


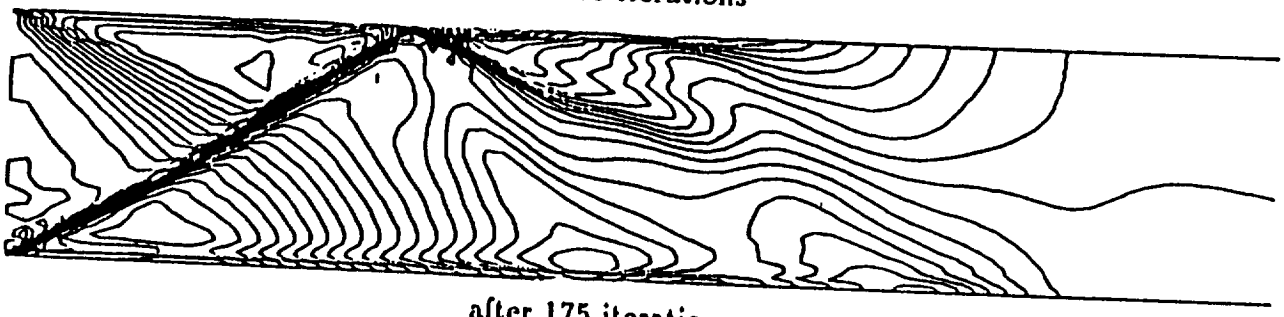
Fig. 5.3 Pressure contours for different iterations ($\alpha = 15.0^{\circ}$, $M_{\infty} = 2.0$)



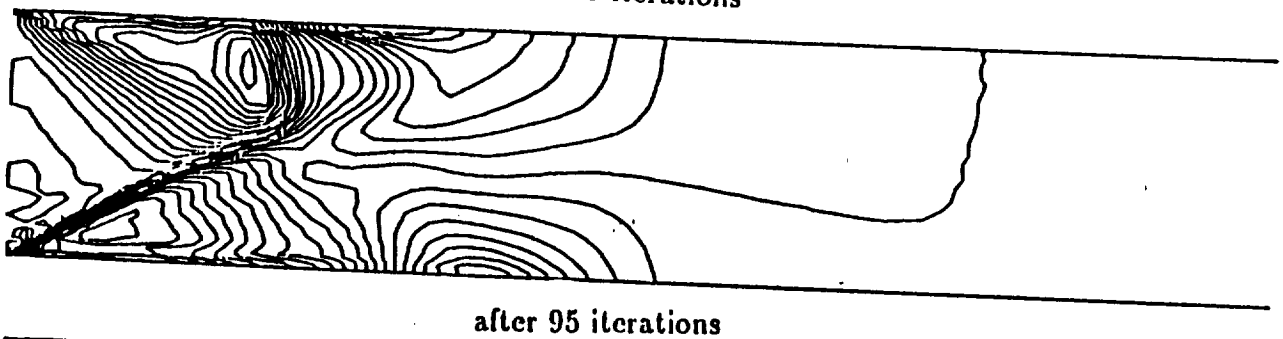
after 763 iterations (converged)



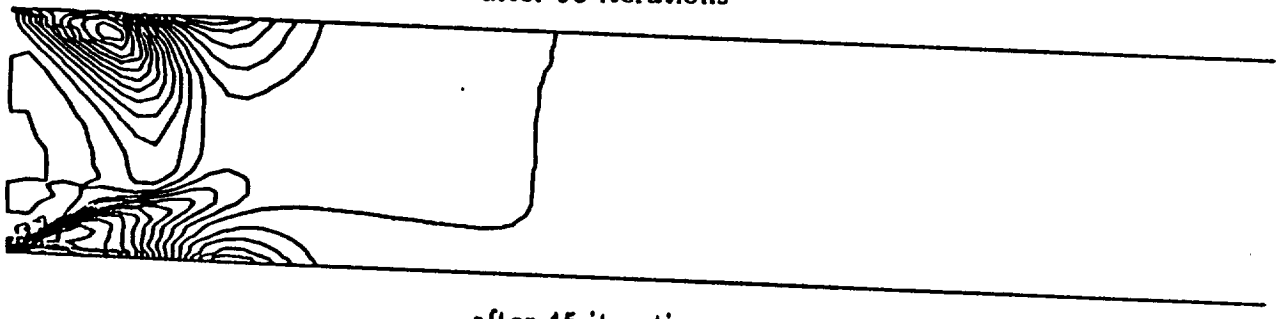
after 305 iterations



after 175 iterations



after 95 iterations



after 45 iterations

Fig. 5.4 Mach number contours for different iterations ($\alpha = 15.0$, $M_\infty = 2.0$)

REFERENCES

1. Abolhassani, J. S. and Everton, E. L., "*Interactive Grid Adaption* , " The Third International Congress of Fluid Mechanics, Vol. I, January 1990, pp. 309-317.

2. Anderson, D. A., Tannehill, J. C., and Pletcher, R. H., Computational Fluid Mechanics and Heat Transfer, Hemisphere Publishing Corporation, New York, 1984.

3. Lavante, E. V., Bruns, R. L., Sanetrik, M. D., and Lam, T., "*Numerical Analysis of Flow About a Total Temperature Sensor*," AIAA Journal, March 1989, pp 692 - 714.

4. Jamson, A., Schmidt, W. and Turkel, E., "*Numerical Solutions of the Euler Equations By Finite-Volume Using Runge-Kutta Time-Stepping Schames*," AIAA Paper 81-1259, June 1981.

5. Hoffmann, K. A., Computational Fluid Dynamics For Engineers, Engineering Education System, Austin, TX ,1989.

6. Rumsey, C. L. and Anderson, W. K., "*Some Numerical and Physical Aspects of Unsteady Navier-Stokes Computations Over Airfoils Using Dynamic Meshes*," AIAA Paper 88-0329, January 1988.

APPENDIX A

RASTER IMAGE GENERATION ROUTINE : PLOTD

Following subroutines are developed for a raster image generation. Main arguments are defined as:

x,y	= grid coordinates
d	= flow variable to be contoured
n	= number of points in x direction
m	= number of points in y direction
is,ie	= desired start and end points for contouring in I-direction
js,je	= desired start and end points for contouring in J-direction
nnc	= number of contour levels

```
      subroutine conplt(x,y,d,m,n,is,ie,js,je,a,b,nnc)
c
      PARAMETER(NPMAX=10000,NPSMAX=100,NSMAX=100)
c
      dimension x(m,n),y(m,n),d(m,n)
c
      common /extrem/   xmin,   xmax,   ymin,   ymax,   dmin,   dmax
      real              xmin,   xmax,   ymin,   ymax,   dmin,   dmax
c
      common /maping/ scalex, ofsetx, scaley, ofsety, scaled, ofsetd
      real            scalex, ofsetx, scaley, ofsety, scaled, ofsetd
c
      dimension ix1(npsmax,3), iy1(npsmax,3)
      dimension ix2(npsmax,3), iy2(npsmax,3)
c
      logical   ind(npsmax,2), flag(npsmax)
c
c     contour plot
c     written by: Eric L. Everton & Jamshid S. Abolhassani
c
      nc=abs(nnc)
c
c     check for zero range
c
      if (dmax.ne.dmin) then
c
c        compute delta
c
         delf=(dmax-dmin)/float(nc)
c
      else
c
c        zero range, issue error message and stop
c
         write(*,*) 'maximum and minimum are equal'
         stop
c
      endif
c
c     set starting contour level
c
      cl=dmin
c
c     set error flag to false
c
      do 100 i=is,iel
c
      flag(i)=.false.
c
  100 continue
c
c     plot contours
c
  150 continue
c
c     checks for contour level out of range
```

```
c
      if(cl.lt.dmin) go to 150
c
      if(cl.gt.dmax) return
c
c     convert contour level to look up table (lut) index
c
      icolor=cl*scaled+ofsetd
c
c     checks for lut index out of range
c
      if(icolor.lt.int(a)) icolor=int(a)
c
      if(icolor.gt.int(b)) icolor=int(b)
c
c     set current lut index (color)
c
      call ufbcol(icolor)
c
c     set do loop end controls
c
      iel=ie-1
      jel=je-1
c
c     upper triangle do loop
c
      do 200 j=js,jel
c
c     reset line plot flags
c
      do 250 k=1,2
      do 250 i=is,iel
c
      ind(i,k)=.false.
c
  250 continue
c
c     vectorized do loop
c
      do 300 i=is,iel
c
c     initialize temporary variables
c
      x1=x(i  ,j  )
      y1=y(i  ,j  )
      f1=d(i  ,j  )
      x2=x(i+1,j  )
      y2=y(i+1,j  )
      f2=d(i+1,j  )
      x3=x(i+1,j+1)
      y3=y(i+1,j+1)
      f3=d(i+1,j+1)
c
c     check to see if contour line intersects this triangle
c
      if ((cl.le.max(f1,f2,f3)).and.(cl.ge.min(f1,f2,f3))) then
```

```
c
c        check to see if contour line intersects point 1
c
         if (cl.eq.f1) then
c
c           check to see if contour line intersects point 2
c
            if (cl.eq.f2) then
c
c              check to see if contour line intersects point 3
c
               if (cl.eq.f3) then
c
c                 contour line intersects all three verticies
c
                  ix1(i,1)=ofsetx+scalex*x1
                  iy1(i,1)=ofsety+scaley*y1
                  ix2(i,1)=ofsetx+scalex*x2
                  iy2(i,1)=ofsety+scaley*y2
                  ind(i,1)=.true.
                  ix1(i,2)=ofsetx+scalex*x2
                  iy1(i,2)=ofsety+scaley*y2
                  ix2(i,2)=ofsetx+scalex*x3
                  iy2(i,2)=ofsety+scaley*y3
                  ind(i,2)=.true.
                  ix1(i,3)=ofsetx+scalex*x3
                  iy1(i,3)=ofsety+scaley*y3
                  ix2(i,3)=ofsetx+scalex*x1
                  iy2(i,3)=ofsety+scaley*y1
c
               else
c
c                 contour line intersects point 1 and point 2
c
                  ix1(i,1)=ofsetx+scalex*x1
                  iy1(i,1)=ofsety+scaley*y1
                  ix2(i,1)=ofsetx+scalex*x2
                  iy2(i,1)=ofsety+scaley*y2
                  ind(i,1)=.true.
c
               endif
c
c           contour line did not intersect point 2
c           check to see if contour line intersects point 3
c
            else if (cl.eq.f3) then
c
c              contour line intersects point 1 and point 3
c
               ix1(i,1)=ofsetx+scalex*x1
               iy1(i,1)=ofsety+scaley*y1
               ix2(i,1)=ofsetx+scalex*x3
               iy2(i,1)=ofsety+scaley*y3
               ind(i,1)=.true.
c
            else
```

```
c
c              check to see if contour line intersects edge
c              between point 2 and point 3
c
               if ((c1-f2)*(c1-f3).le.0.) then
c
c                 contour line intersects point 1 and
c                 edge between point 2 and point 3
c
                  ix1(i,1)=ofsetx+scalex*x1
                  iy1(i,1)=ofsety+scaley*y1
                  ix2(i,1)=ofsetx+scalex*(x2+(x3-x2)*(c1-f2)/(f3-f2))
                  iy2(i,1)=ofsety+scaley*(y2+(y3-y2)*(c1-f2)/(f3-f2))
                  ind(i,1)=.true.
c
               endif
c
            endif
c
c           contour line did not intersect point 1
c           check to see if contour line intersects point 2
c
            else if (c1.eq.f2) then
c
c              contour line intersects point 2
c
               ix1(i,1)=ofsetx+scalex*x2
               iy1(i,1)=ofsety+scaley*y2
c
c              check to see if contour line intersects point 3
c
               if (c1.eq.f3) then
c
c                 contour line intersects point 2 and point 3
c
                  ix2(i,1)=ofsetx+scalex*x3
                  iy2(i,1)=ofsety+scaley*y3
                  ind(i,1)=.true.
c
               else
c
c                 check to see if contour line intersects edge
c                 between point 1 and point 3
c
                  if ((c1-f1)*(c1-f3).le.0.) then
c
c                    contour line intersects point 2 and
c                    edge between point 1 and point 3
c
                     ix2(i,1)=ofsetx+scalex*(x1+(x3-x1)*(c1-f1)/(f3-f1))
                     iy2(i,1)=ofsety+scaley*(y1+(y3-y1)*(c1-f1)/(f3-f1))
                     ind(i,1)=.true.
c
                  endif
c
               endif
```

```
c
c          check to see if contour line intersects point 3
c
           else if (c1.eq.f3) then
c
c             check to see if contour line intersects edge
c             between point 1 and point 2
c
              if ((c1-f1)*(c1-f2).le.0.) then
c
c                contour line intersects point 3 and
c                edge between point 1 and point 2
c
                 ix1(i,1)=ofsetx+scalex*x3
                 iy1(i,1)=ofsety+scaley*y3
                 ix2(i,1)=ofsetx+scalex*(x1+(x2-x1)*(c1-f1)/(f2-f1))
                 iy2(i,1)=ofsety+scaley*(y1+(y2-y1)*(c1-f1)/(f2-f1))
                 ind(i,1)=.true.
c
              endif
c
           else
c
c             contour line does not intersect any of the verticies
c             check to see if contour line intersects edge
c             between point 2 and point 3
c
              if ((c1-f2)*(c1-f3).le.0.) then
c
c                contour line intersects edge
c                between point 2 and point 3
c
                 ix1(i,1)=ofsetx+scalex*(x2+(x3-x2)*(c1-f2)/(f3-f2))
                 iy1(i,1)=ofsety+scaley*(y2+(y3-y2)*(c1-f2)/(f3-f2))
c
c                check to see if contour line intersects edge
c                between point 1 and point 3
c
                 if ((c1-f1)*(c1-f3).le.0.) then
c
c                   contour line intersects edges
c                   between point 2 and point 3 and
c                   between point 1 and point 3
c
                    ix2(i,1)=ofsetx+scalex*(x1+(x3-x1)*(c1-f1)/(f3-f1))
                    iy2(i,1)=ofsety+scaley*(y1+(y3-y1)*(c1-f1)/(f3-f1))
                    ind(i,1)=.true.
c
c                contour line does not intersect edge
c                between point 1 and point 3
c                check to see if contour line intersects edge
c                between point 1 and point 2
c
                 else if ((c1-f1)*(c1-f2).le.0.) then
c
c                   contour line intersects edges
```

```
c                between point 2 and point 3 and
c                between point 1 and point 2
c
                 ix2(i,1)=ofsetx+scalex*(x1+(x2-x1)*(c1-f1)/(f2-f1))
                 iy2(i,1)=ofsety+scaley*(y1+(y2-y1)*(c1-f1)/(f2-f1))
                 ind(i,1)=.true.
c
c             must be a problem
c
              else
c
c                set error flag
c
                 flag(i)=.true.
c
              endif
c
c           contour line does not intersect edge
c           between point 2 and point 3
c           check to see if contour line intersects edge
c           between point 1 and point 3
c
            else if ((c1-f1)*(c1-f3).le.0.) then
c
c             contour line intersects edge
c             between point 1 and point 3
c
              ix1(i,1)=ofsetx+scalex*(x1+(x3-x1)*(c1-f1)/(f3-f1))
              iy1(i,1)=ofsety+scaley*(y1+(y3-y1)*(c1-f1)/(f3-f1))
c
c             check to see if contour line intersects edge
c             between point 1 and point 2
c
              if ((c1-f1)*(c1-f2).le.0.) then
c
c                contour line intersects edges
c                between point 1 and point 3 and
c                between point 1 and point 2
c
                 ix2(i,1)=ofsetx+scalex*(x1+(x2-x1)*(c1-f1)/(f2-f1))
                 iy2(i,1)=ofsety+scaley*(y1+(y2-y1)*(c1-f1)/(f2-f1))
                 ind(i,1)=.true.
c
c             must be a problem
c
              else
c
c                set error flag
c
                 flag(i)=.true.
c
              endif
c
c           contour line did not intersect edges
c           between point 2 and point 3 and
c           between point 1 and point 3
```

```
c            check to see if contour line intersects edge
c            between point 1 and point 2
c
             else if ((c1-f1)*(c1-f2).le.0.) then
c
c               contour line only intersects edge
c               between point 1 and point 2
c               must be a problem
c               set error flag
c
                flag(i)=.true.
c
             endif
c
          endif
c
       endif
c
  300 continue
c
      do 350 i=is,iel
c
c        check to see if an error occured
c
         if (flag(i)) then
c
c            issue error message and stop
c
             write(*,*) 'There is something fishy about conplt!'
             stop
c
         endif
c
  350 continue
c
      do 360 i=is,iel
c
c     check to see if line is to be plotted
c
      if (ind(i,1)) then
c
c        plot a single line
c
         call ufblin(ix1(i,1),iy1(i,1),ix2(i,1),iy2(i,1))
c
c        check for more lines
c
         if (ind(i,2)) then
c
c            plot two more lines
c
             call ufblin(ix1(i,2),iy1(i,2),ix2(i,2),iy2(i,2))
             call ufblin(ix1(i,3),iy1(i,3),ix2(i,3),iy2(i,3))
c
         endif
c
```

```
      endif
c
  360 continue
c
  200 continue
c
c     lower triangle do loop
c
      do 400 j=js,je1
c
c     reset line plot flags
c
      do 450 k=1,2
      do 450 i=is,ie1
c
      ind(i,k)=.false.
c
  450 continue
c
c     vectorized do loop
c
      do 500 i=is,ie1
c
c     initialize temporary variables
c
      x1=x(i  ,j  )
      y1=y(i  ,j  )
      f1=d(i  ,j  )
      x3=x(i+1,j+1)
      y3=y(i+1,j+1)
      f3=d(i+1,j+1)
      x4=x(i  ,j+1)
      y4=y(i  ,j+1)
      f4=d(i  ,j+1)
c
c     check to see if contour line intersects this triangle
c
      if ((cl.le.max(f1,f3,f4)).and.(cl.ge.min(f1,f3,f4))) then
c
c        check to see if contour line intersects point 1
c
         if (cl.eq.f1) then
c
c           check to see if contour line intersects point 3
c
            if (cl.eq.f3) then
c
c              check to see if contour line intersects point 4
c
               if (cl.eq.f4) then
c
c                 contour line intersects all three verticies
c
                  ix1(i,1)=ofsetx+scalex*x1
                  iy1(i,1)=ofsety+scaley*y1
                  ix2(i,1)=ofsetx+scalex*x3
```

```
                  iy2(i,1)=ofsety+scaley*y3
                  ind(i,1)=.true.
                  ix1(i,2)=ofsetx+scalex*x3
                  iy1(i,2)=ofsety+scaley*y3
                  ix2(i,2)=ofsetx+scalex*x4
                  iy2(i,2)=ofsety+scaley*y4
                  ind(i,2)=.true.
                  ix1(i,3)=ofsetx+scalex*x4
                  iy1(i,3)=ofsety+scaley*y4
                  ix2(i,3)=ofsetx+scalex*x1
                  iy2(i,3)=ofsety+scaley*y1
c
               else
c
c                 contour line intersects point 1 and point 3
c
                  ix1(i,1)=ofsetx+scalex*x1
                  iy1(i,1)=ofsety+scaley*y1
                  ix2(i,1)=ofsetx+scalex*x3
                  iy2(i,1)=ofsety+scaley*y3
                  ind(i,1)=.true.
c
               endif
c
c           contour line did not intersect point 3
c           check to see if contour line intersects point 4
c
            else if (cl.eq.f4) then
c
c              contour line intersects point 1 and point 4
c
               ix1(i,1)=ofsetx+scalex*x1
               iy1(i,1)=ofsety+scaley*y1
               ix2(i,1)=ofsetx+scalex*x4
               iy2(i,1)=ofsety+scaley*y4
               ind(i,1)=.true.
c
            else
c
c              check to see if contour line intersects edge
c              between point 3 and point 4
c
               if ((cl-f3)*(cl-f4).le.0.) then
c
c                 contour line intersects point 1 and
c                 edge between point 3 and point 4
c
                  ix1(i,1)=ofsetx+scalex*x1
                  iy1(i,1)=ofsety+scaley*y1
                  ix2(i,1)=ofsetx+scalex*(x3+(x4-x3)*(cl-f3)/(f4-f3))
                  iy2(i,1)=ofsety+scaley*(y3+(y4-y3)*(cl-f3)/(f4-f3))
                  ind(i,1)=.true.
c
               endif
c
            endif
```

```
c
c          contour line did not intersect point 1
c          check to see if contour line intersects point 3
c
           else if (cl.eq.f3) then
c
c             contour line intersects point 3
c
              ix1(i,1)=ofsetx+scalex*x3
              iy1(i,1)=ofsety+scaley*y3
c
c             check to see if contour line intersects point 4
c
              if (cl.eq.f4) then
c
c                contour line intersects point 3 and point 4
c
                 ix2(i,1)=ofsetx+scalex*x4
                 iy2(i,1)=ofsety+scaley*y4
                 ind(i,1)=.true.
c
              else
c
c                check to see if contour line intersects edge
c                between point 1 and point 4
c
                 if ((cl-f1)*(cl-f4).le.0.) then
c
c                   contour line intersects point 3 and
c                   edge between point 1 and point 4
c
                    ix2(i,1)=ofsetx+scalex*(x1+(x4-x1)*(cl-f1)/(f4-f1))
                    iy2(i,1)=ofsety+scaley*(y1+(y4-y1)*(cl-f1)/(f4-f1))
                    ind(i,1)=.true.
c
                 endif
c
              endif
c
c          check to see if contour line intersects point 4
c
           else if (cl.eq.f4) then
c
c             check to see if contour line intersects edge
c             between point 1 and point 3
c
              if ((cl-f1)*(cl-f3).le.0.) then
c
c                contour line intersects point 4 and
c                edge between point 1 and point 3
c
                 ix1(i,1)=ofsetx+scalex*x4
                 iy1(i,1)=ofsety+scaley*y4
                 ix2(i,1)=ofsetx+scalex*(x1+(x3-x1)*(cl-f1)/(f3-f1))
                 iy2(i,1)=ofsety+scaley*(y1+(y3-y1)*(cl-f1)/(f3-f1))
                 ind(i,1)=.true.
```

```
c
            endif
c
         else
c
c           contour line does not intersect any of the verticies
c           check to see if contour line intersects edge
c           between point 3 and point 4
c
            if ((c1-f3)*(c1-f4).le.0.) then
c
c              contour line intersects edge
c              between point 3 and point 4
c
               ix1(i,1)=ofsetx+scalex*(x3+(x4-x3)*(c1-f3)/(f4-f3))
               iy1(i,1)=ofsety+scaley*(y3+(y4-y3)*(c1-f3)/(f4-f3))
c
c              check to see if contour line intersects edge
c              between point 1 and point 4
c
               if ((c1-f1)*(c1-f4).le.0.) then
c
c                 contour line intersects edges
c                 between point 3 and point 4 and
c                 between point 1 and point 4
c
                  ix2(i,1)=ofsetx+scalex*(x1+(x4-x1)*(c1-f1)/(f4-f1))
                  iy2(i,1)=ofsety+scaley*(y1+(y4-y1)*(c1-f1)/(f4-f1))
                  ind(i,1)=.true.
c
c              contour line does not intersect edge
c              between point 1 and point 4
c              check to see if contour line intersects edge
c              between point 1 and point 3
c
               else if ((c1-f1)*(c1-f3).le.0.) then
c
c                 contour line intersects edges
c                 between point 3 and point 4 and
c                 between point 1 and point 3
c
                  ix2(i,1)=ofsetx+scalex*(x1+(x3-x1)*(c1-f1)/(f3-f1))
                  iy2(i,1)=ofsety+scaley*(y1+(y3-y1)*(c1-f1)/(f3-f1))
                  ind(i,1)=.true.
c
c              must be a problem
c
               else
c
c                 set error flag
c
                  flag(i)=.true.
c
               endif
c
c           contour line does not intersect edge
```

```
c            between point 3 and point 4
c            check to see if contour line intersects edge
c            between point 1 and point 4
c
             else if ((c1-f1)*(c1-f4).le.0.) then
c
c               contour line intersects edge
c               between point 1 and point 4
c
                ix1(i,1)=ofsetx+scalex*(x1+(x4-x1)*(c1-f1)/(f4-f1))
                iy1(i,1)=ofsety+scaley*(y1+(y4-y1)*(c1-f1)/(f4-f1))
c
c               check to see if contour line intersects edge
c               between point 1 and point 3
c
                if ((c1-f1)*(c1-f3).le.0.) then
c
c                  contour line intersects edges
c                  between point 1 and point 4 and
c                  between point 1 and point 3
c
                   ix2(i,1)=ofsetx+scalex*(x1+(x3-x1)*(c1-f1)/(f3-f1))
                   iy2(i,1)=ofsety+scaley*(y1+(y3-y1)*(c1-f1)/(f3-f1))
                   ind(i,1)=.true.
c
c               must be a problem
c
                else
c
c                  set error flag
c
                   flag(i)=.true.
c
                endif
c
c            contour line did not intersect edges
c            between point 3 and point 4 and
c            between point 1 and point 4
c            check to see if contour line intersects edge
c            between point 1 and point 3
c
             else if ((c1-f1)*(c1-f3).le.0.) then
c
c               contour line only intersects edge
c               between point 1 and point 2
c               must be a problem
c               set error flag
c
                flag(i)=.true.
c
             endif
c
          endif
c
       endif
c
```

```
  500 continue
c
      do 550 i=is,iel
c
c        check to see if an error occured
c
         if (flag(i)) then
c
c           issue error message and stop
c
            write(*,*) 'There is something fishy about conplt!'
            stop
c
         endif
c
  550 continue
c
      do 560 i=is,iel
c
c     check to see if line is to be plotted
c
      if (ind(i,1)) then
c
c        plot a single line
c
         call ufblin(ix1(i,1),iy1(i,1),ix2(i,1),iy2(i,1))
c
c        check for more lines
c
         if (ind(i,2)) then
c
c           plot two more lines
c
            call ufblin(ix1(i,2),iy1(i,2),ix2(i,2),iy2(i,2))
            call ufblin(ix1(i,3),iy1(i,3),ix2(i,3),iy2(i,3))
c
         endif
c
      endif
c
  560 continue
c
  400 continue
c
c     increment contour level
c
      c1=c1+delf
c
c     plot contours
c
      go to 150
c
      end
c*****************************************************************
      subroutine fminmax(f,fmin,fmax,is,ie,js,je,m,n,iflag)
c
```

```
      dimension f(m,n)
c
      data eps/1e14/
c
c     check to reset minimums and maximums
c
      if(iflag.eq.0) then
c
         fmin=eps
         fmax=-eps
c
      end if
c
      do 100 j=js,je
      do 100 i=is,ie
c
c        check to reset minimums and maximums
c
         fmax=amax1(f(i,j),fmax)
         fmin=amin1(f(i,j),fmin)
c
  100 continue
c
      return
      end
C
C*************************************************************************
*
C
      subroutine getf(td,tu,tv,te,dd,m,n,ivar)
c
      dimension td(*), tu(*), tv(*), te(*), dd(*)
c
      common/fluid/gamma,gm1,gp1,gm1g,gp1g,ggm1
c
      data cv/4390./
c
      mn=m*n
c
      rcv=1./cv
      rggm1=1./ggm1
      coe1=(gamma-1.0)*cv
      coe2=gamma*(gamma-1.0)*cv
c
      if(ivar.eq.1)then
         do 10  i=1,mn
            dd(i)=td(i)
 10      continue
         return
      end if
c
      if(ivar.eq.2)then
         do 20  i=1,mn
            rrho=1./td(i)
            u1=tu(i)*rrho
            v1=tv(i)*rrho
```

```
            dd(i)=(te(i)*rrho-0.5*(u1*u1+v1*v1))*rcv
 20       continue
          return
       end if
c
       if(ivar.eq.3)then
          do 30  i=1,mn
             rrho=1./td(i)
             u1=tu(i)*rrho
             v1=tv(i)*rrho
             t1=(te(i)*rrho-0.5*(u1*u1+v1*v1))*rcv
             dd(i)=td(i)*gm1*cv*t1
 30       continue
          return
       end if
c
       if(ivar.eq.4)then
          do 40  i=1,mn
             rrho=1./td(i)
             u1=tu(i)*rrho
             v1=tv(i)*rrho
             uv=u1*u1+v1*v1
             t1=(te(i)*rrho-0.5*uv)*rcv
             c1=abs(t1*ggm1*cv)
             dd(i)=sqrt(uv/c1)
 40       continue
       end if
c
       if(ivar.eq.5)then
          do 50  i=1,mn
             rrho=1./td(i)
             u1=tu(i)*rrho
             v1=tv(i)*rrho
             t1=(te(i)*rrho-0.5*(u1*u1+v1*v1))*rcv
             p1=td(i)*gm1*cv*t1
             dd(i)=log(t1**rggm1/p1)
 50       continue
       end if
c
       if(ivar.eq.6)then
          do 60  i=1,mn
             uv=tu(i)*tu(i)+tv(i)*tv(i)
             dd(i)=0.5*uv/td(i)
 60       continue
       end if
c
       if(ivar.eq.7)then
          do 70  i=1,mn
             rrho=1./td(i)
             u1=tu(i)*rrho
             v1=tv(i)*rrho
             t1=(te(i)*rrho-0.5*(u1*u1+v1*v1))*rcv
             p1=td(i)*gm1*cv*t1
             dd(i)=(te(i)+p1)*rrho
 70       continue
       end if
```

```
c
      return
      end
c*************************************************************
      subroutine gridxy(x,y,m,n,is,ie,js,je,iflag)
c
      dimension x(m,n),y(m,n)
c
      common /maping/ scalex, ofsetx, scaley, ofsety, scaled, ofsetd
      real            scalex, ofsetx, scaley, ofsety, scaled, ofsetd
c
      dimension ix(4),iy(4)
c
c     iflag=0    draw the boundaries   only
c     iflag=1    draw constant-i lines only
c     iflag=2    draw constant-j lines only
c     iflag=3    draw the entire gird
c
c     set grid line look up table (lut) index (color)
c
      call ufbcol(255)
c
c     compute do loop end and increment
c
      ii1=ie-1
      jj1=je-1
      ii=1
      jj=1
c
c     check to reset increments
c
      if(iflag.eq.0.or.iflag.eq.1) jj=je-js
      if(iflag.eq.0.or.iflag.eq.2) ii=ie-is
c
c     check to plot grid lines
c
      if((iflag.ne.2).and.(ii.gt.0).and.(js.ne.je)) then
c
         do 100 i=is,ie,ii
         do 100 j=js,jj1
c
c           convert world coordinates to screen coordinates
c
            ix(1)=scalex*x(i,j  )+ofsetx
            ix(2)=scalex*x(i,j+1)+ofsetx
            iy(1)=scaley*y(i,j  )+ofsety
            iy(2)=scaley*y(i,j+1)+ofsety
c
c           plot grid line
c
            call ufblin(ix(1),iy(1),ix(2),iy(2))
c
 100     continue
c
      endif
c
```

```
c     check to plot grid lines
c
      if((iflag.ne.1).and.(jj.gt.0).and.(is.ne.ie)) then
c
         do 200 j=js,je,jj
         do 200 i=is,iil
c
c           convert world coordinates to screen coordinates
c
            ix(1)=scalex*x(i  ,j)+ofsetx
            ix(2)=scalex*x(i+1,j)+ofsetx
            iy(1)=scaley*y(i  ,j)+ofsety
            iy(2)=scaley*y(i+1,j)+ofsety
c
c           plot grid line
c
            call ufblin(ix(1),iy(1),ix(2),iy(2))
c
 200     continue
c
      endif
c
      return
      end
c
c*****************************************************************
c
      subroutine linmap(a1,a2,b1,b2,c1,c2)
c
c     compute scale factor and offset
c
      if(a2.ne.a1) then
           c1=(b2-b1)/(a2-a1)
           c2=b1 - c1*a1 + .5
      else
          c1=0.
          c2=(b1+b2)*0.5
      end if
      return
      end
c*****************************************************************
****
      subroutine plotd(x,y,q,d,m,n,ivar,is,ie,js,je,nc,sides,
     *                 id,idmin,idmax,it,iflag)
c
      dimension x(m,n),y(m,n),q(m,n,4),d(m,n)
c
      common /extrem/   xmin,   xmax,   ymin,   ymax,   dmin,   dmax
      real              xmin,   xmax,   ymin,   ymax,   dmin,   dmax
c
      common /maping/ scalex, ofsetx, scaley, ofsety, scaled, ofsetd
      real            scalex, ofsetx, scaley, ofsety, scaled, ofsetd
c
      common/fluid/gamma,gm1,gp1,gm1g,gp1g,ggm1
c
      real idmin, idmax
```

```
      save odmin, odmax
c
      data gamma,odmin,odmax/1.4,+1e30,-1e30/
c
c  contour plot
c  written by: Eric L. Everton & Jamshid S. Abolhassani
c
      gm1=gamma-1.
      gp1=gamma+1.
      gm1g=gm1/gamma
      gp1g=gp1/gamma
      ggm1=gamma*gm1
c
c     ivar=0  write out the grid
c          1  density
c          2  temperature
c          3  pressure
c          4  mach number
c          5  entropy
c          6  dynamic pressure
c          7  total enthalpy
c
c --------------- scaling the grid & flow variables
c
      if (it.eq.0) then
c
c        compute x and y minimums and maximums
c
         call fminmax(x,xmin,xmax,is,ie,js,je,m,n,0)
         call fminmax(y,ymin,ymax,is,ie,js,je,m,n,0)
c
c        compute aspect ratio and
c        x and y ranges
c
         yoverx=1024./1280.
         arange=xmax-xmin
         brange=ymax-ymin
c
c        check for zero ranges
c
         if ((arange.eq.0.).or.(brange.eq.0.)) then
c
c           a range is zero
c           issue an error message and stop
c
            write(*,*) 'xmin - xmax or ymin - ymax eq 0'
            stop
c
         endif
c
c        check aspect ratio with range ratio
c
         if (brange/arange.gt.yoverx) then
c
c           adjust xmin and xmax
c           recompute range
```

```
c
            hafdif= .5 * (brange/yoverx - arange)
            xmin  =xmin-hafdif
            xmax  =xmax+hafdif
            arange=xmax-xmin
c
         else
c
c           adjust ymin and ymax
c           recompute range
c
            hafdif= .5 * (arange*yoverx - brange)
            ymin  =ymin-hafdif
            ymax  =ymax+hafdif
            brange=ymax-ymin
c
         endif
c
c        add border
c
         xmin=xmin - sides*arange
         xmax=xmax + sides*arange
         ymin=ymin - sides*brange
         ymax=ymax + sides*brange
c
c        compute scale and offset for x and y
c
         call linmap(xmin,xmax,0.,1279.,scalex,ofsetx)
         call linmap(ymin,ymax,0.,1023.,scaley,ofsety)
c
      endif
c
c     get d variable
c
      call getf(q(1,1,1),q(1,1,2),q(1,1,3),q(1,1,4),d,m,n,ivar)
c
c     compute d minimum and maximum
c
      call fminmax(d,cdmin,cdmax,is,ie,js,je,m,n,0)
      idflag=1
      if(cdmin.eq.cdmax) idflag=0
c
c     save overall d minimum and maximum
c
      if (cdmin.lt.odmin) odmin=cdmin
      if (cdmax.gt.odmax) odmax=cdmax
c
c     check to use current d minimum and maximum
c
      if (id.eq.0) then
c
c        set dmin and dmax to current
c
         dmin=cdmin
         dmax=cdmax
c
```

```
      else
c
c        set dmin and dmax to input
c
         dmin=idmin
         dmax=idmax
c
      endif
c
c     compute scale and offset for d
c
      if(idflag.eq.1)
     1   call linmap(dmin,dmax,2.,255.,scaled,ofsetd)
c
c     write current, overall & used d minimums and maximums
c
      write(62,*) ' current dmin & dmax = ', cdmin, cdmax
      write(62,*) ' overall dmin & dmax = ', odmin, odmax
      write(62,*) '     used dmin & dmax = ',  dmin,  dmax
c
c     clear frame buffer
c
      call ufbcle
c
c     write frame number
c
      write(62,*)' Plotting frame number ',it
c
c     plot contours
c
      if(idflag.eq.1)
     1   call conplt(x,y,d,m,n,is,ie,js,je,2.,255.,nc)
c
c     plot grid lines
c
      call gridxy(x,y,m,n,is,ie,js,je,iflag)
c
c     send frame buffer to display
c
      call ufbwri
c
      return
      end
```