FINAL REPORT

on

SOLID ROCKET BOOSTER INTERNAL FLOW ANALYSIS BY HIGHLY ACCURATE ADAPTIVE COMPUTATIONAL METHODS

Contract Number NAS8–37682

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Marshall Space Flight Center
Huntsville, Alabama

TR-91-05

March 1991

C. Y. Huang, W. Tworzydlo, J. T. Oden
J. M. Bass, C. Cullen, S. Vadaketh

The Computational Mechanics Company, Inc.
7701 North Lamar, Suite 200
Austin, Texas 78752
(512) 467–0618

FINAL REPORT

on

SOLID ROCKET BOOSTER INTERNAL FLOW ANALYSIS BY HIGHLY ACCURATE ADAPTIVE COMPUTATIONAL METHODS

Contract Number NAS8-37682

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Marshall Space Flight Center
Huntsville, Alabama
TR-91-05

March 1991

C. Y. Huang, W. Tworzydlo, J. T. Oden
J. M. Bass, C. Cullen, S. Vadaketh

The Computational Mechanics Company, Inc.
7701 North Lamar, Suite 200
Austin, Texas 78752
(512) 467-0618

Contents

1 Introduction

Over the past three years, the Computational Mechanics Co., Inc. has designed and produced a new computational tool for modeling internal flows in solid rocket motors as part of the project "Internal Flow Analysis by Highly–Accurate Adaptive Computational Methods" (NAS8-37682). During the course of this effort several new algorithms for studying the effects of moving boundaries on flow characteristics within solid propellant rocket motors have been developed and tested on both two–dimensional planar and axisymmetric computational domains. Of particular interest has been the development of a receding boundary algorithm which successfully models the changing flow domain during the erosion of the propellant within a solid rocket motor. This method relies on adaptive finite element methods that combine node relocation techniques (r-methods), mesh refinement techniques (h-methods), and moving boundaries to form a very powerful analysis package.

In addition to the research and development efforts in the area of moving boundaries, considerable effort has also been focused on the following topics:

- Implementation of a fully adaptive implicit/explicit finite element methodology to optimize the computational effort required to advance the solution forward in time.
- Formulation and implementation of a generalized algebraic turbulence model and data structure compatible with adaptive methodologies and applicable to completely unstructured grids.
- Testing and validation of the computational algorithms for several benchmark problems.

The success of the two–dimensional and axisymmetric analysis package in modeling the benchmark problems suggests that extensions to realistic three–dimensional flows in cavities with eroding boundaries is both feasible and a natural extension of the project currently underway. The final section of this report describes some possible extensions of this project in terms of enhancements of the two–dimensional code and the development of a fully three–dimensional code for use in the design and analysis of solid rocket motors.

The remainder of this report presents a detailed description of the theoretical formulation, numerical methods, and representative examples of the analysis of complex flow phenomena occurring in solid rocket motors. The equations that model flow phenomena for this class of problems are derived in Section 2. The associated boundary conditions are then briefly discussed in Section 3. An implicit/explicit flow algorithm is presented next in Section 4, while the moving boundary and remeshing algorithms are highlighted in Section 5. Section 6 introduces adaptive strategies and the related data structures. The implementation of a

simple algebraic turbulence model is presented in Section 7 along with a discussion of the data structure and storage requirements. Following the discussion of the algebraic turbulence model are several numerical examples which demonstrate some of the modeling capabilities available in the code.

2 Governing Equations

For the class of the fluid dynamic problems we are going to solve, the computational domain may be continuously changed due to boundary motion. It is well known that for fluid dynamic problems involving grid motion, the original Navier-Stokes equations derived from the Eulerian approach need to be modified by using the Arbitrary Lagrangian-Eulerian (ALE) formulation, see reference [1] for detailed derivation. It should be noted that the ALE formulation derived in [1] is expressed in the form of Cartesian tensor notations. To derive its axisymmetric counterpart, we first convert the Cartesian tensor notations to vector operator forms. These vector operators can be readily transformed into axisymmetric form using orthogonal curvilinear transformations.

Two–Dimensional Formulation of the Navier-Stokes Equations

The time–dependent Navier-Stokes equations, including grid motion, can be expressed in tensor notations as

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho(u_i - u_i^G)}{\partial x_i} + \sigma\rho = 0 \tag{2.1}$$

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial}{\partial x_j}\left[\rho u_i(u_j - u_j^G) - \tau_{ij}\right] + \sigma\rho u_i = 0 \tag{2.2}$$

$$\frac{\partial \varepsilon}{\partial t} + \frac{\partial}{\partial x_i}\left[\varepsilon(u_i - u_i^G) - u_j\tau_{ij} + \dot{q}_i\right] + \sigma\varepsilon = 0 \tag{2.3}$$

where ρ denotes the density of the fluids, u_i is the i-th component of flow velocity, u_i^G is the i-th component of grid velocity, and ε represents the total energy per unit mass. The shear stress tensor τ_{ij} can be expressed as

$$\tau_{ij} = -p\delta_{ij} + \lambda\delta_{ij}\frac{\partial u_k}{\partial x_k} + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

and the heat flux q_i is

$$\dot{q}_i = -k\frac{\partial T}{\partial x_i}$$

Here μ and λ represent the molecular and secondary viscosity, respectively. The source terms in equations (2.1)–(2.3) are due to the grid volumetric dilatation, $\sigma = \nabla \cdot \boldsymbol{u}^G$. For simplicity and clarity, we will omit these terms in subsequent discussions.

Axisymmetric Formulation of the Navier-Stokes Equations

In vector operator form, equations (2.1–2.3) can be expressed as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho(V - V^G) = 0 \tag{2.4}$$

$$\begin{aligned} \frac{\partial \rho V}{\partial t} \; &+ \; \nabla \cdot (V - V^G) \otimes \rho V + \nabla p - \nabla(2\mu + \lambda)\nabla \cdot V \\ &+ \; \nabla \times \mu(\nabla \times V) = 0 \end{aligned} \tag{2.5}$$

$$\begin{aligned} \frac{\partial \varepsilon}{\partial t} \; &+ \; \nabla \cdot \varepsilon(V - V^G) + \nabla \cdot pV - \nabla \cdot (\lambda V \nabla \cdot V) \\ &- \; \nabla \cdot \mu \nabla (V \cdot V) + \nabla \cdot (\mu V \times \nabla \times V) + \nabla \cdot \dot{q} = 0 \end{aligned} \tag{2.6}$$

Here V is the velocity vector and V^G represents grid motion.

In orthogonal curvilinear coordinate systems (e_1, e_2, e_3), these vector operators can be expressed as:

$$\nabla = \frac{1}{h_i}\frac{\partial}{\partial x_i} e_i$$

$$\nabla \cdot V = \frac{1}{h_1 h_2 h_3}\left[\frac{\partial}{\partial x_i}(h_2 h_3 V_1) + \frac{\partial}{\partial x_2}(h_1 h_3 V_2) + \frac{\partial}{\partial x_3}(h_1 h_2 V_3)\right]$$

$$\nabla \times V = \frac{1}{h_1 h_2 h_3}\begin{vmatrix} h_1 e_1 & h_2 e_2 & h_3 e_3 \\ \dfrac{\partial}{\partial x_1} & \dfrac{\partial}{\partial x_2} & \dfrac{\partial}{\partial x_3} \\ h_1 V_1 & h_2 V_2 & h_3 V_3 \end{vmatrix}$$

where h_i are metric scale factors.

For a cylindrical coordinate system,

$$h_1 = 1, \quad h_2 = r, \quad h_3 = 1$$

By expanding these vector operators in equations (2.4–2.6), and neglecting all terms related to θ, the axisymmetric form of Navier-Stokes equations is obtained:

Equation of Continuity

$$\boxed{\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u_r)}{\partial r} + \frac{\partial(\rho u_z)}{\partial z} + \frac{\rho u_r}{r} = 0} \tag{2.7}$$

Multiplying equation (2.7) by r and rearranging terms, we have

$$\boxed{\frac{\partial(\rho r)}{\partial t} + \frac{\partial(\rho r u_r)}{\partial r} + \frac{\partial(\rho r u_z)}{\partial z} = 0} \tag{2.8}$$

Equation of motion (r-component)

$$\boxed{\begin{aligned} &\frac{\partial \rho u_r(u_r - u_r^G)}{\partial t} + \frac{\partial \rho u_r(u_z - u_r^G)}{\partial r} + \frac{\partial \rho u_r(u_z - u_z^G)}{\partial z} + \\ &\frac{\rho u_r(u_r - u_r^G)}{r} + \frac{\partial p}{\partial r} - \frac{\partial \tau_{rr}}{\partial r} - \frac{\partial \tau_{rz}}{\partial z} \\ &-\frac{2\mu}{r}\left(\frac{\partial u_r}{\partial r} - \frac{u_r}{r}\right) = 0 \end{aligned}} \tag{2.9}$$

where

$$\tau_{rr} = 2\mu\frac{\partial u_r}{\partial r} + \lambda \nabla \cdot \boldsymbol{V}, \qquad \nabla \cdot \boldsymbol{V} = \frac{\partial u_r}{\partial r} + \frac{\partial u_z}{\partial z} + \frac{u_r}{r}$$

$$\tau_{rz} = \mu\left(\frac{\partial u_r}{\partial z} + \frac{\partial u_z}{\partial r}\right)$$

$$\lambda = -2\mu/3, \text{ if Stokes' hypothesis is applied}$$

Multiplying equation (2.9) by r and rearranging terms, we obtain

$$\boxed{\begin{aligned}&\frac{\partial(\rho r u_r)}{\partial t}+\frac{\partial \rho r u_r(u_r-u_r^G)}{\partial r}+\frac{\partial \rho_r u_r(u_z-u_z^G)}{\partial z}+\frac{\partial(rp)}{\partial r}\\&-\frac{\partial(r\tau_{rr})}{\partial r}-\frac{\partial(r\tau_{rz})}{\partial z}-p+\tau_{\theta\theta}=0\end{aligned}} \tag{2.10}$$

where

$$\tau_{\theta\theta}=\frac{+2\mu u_r}{r}+\lambda\nabla\cdot V$$

and

$$\nabla\cdot V=\frac{\partial u_r}{\partial r}+\frac{\partial u_z}{\partial z}+\frac{u_r}{r}$$

Equation of motion (z-component)

$$\boxed{\begin{aligned}&\frac{(\rho u_z)}{\partial t}+\frac{\partial \rho u_z(u_r-u_r^G)}{\partial r}+\frac{\partial \rho u_z(u_z-u_z^G)}{\partial z}\\&+\frac{\rho u_z(u_r-u_r^G)}{r}+\frac{\partial p}{\partial z}-\frac{\partial \tau_{zr}}{\partial r}-\frac{\partial \tau_{zz}}{\partial z}-\frac{\tau_{zr}}{r}=0\end{aligned}} \tag{2.11}$$

where

$$\begin{aligned}\tau_{zz} &= 2\mu\frac{\partial u_z}{\partial z}+\lambda\nabla\cdot V\\ \tau_{zr} &= \mu\left(\frac{\partial u_r}{\partial z}+\frac{\partial u_z}{\partial r}\right)\end{aligned}$$

Multiplying by r and rearranging terms, we have

$$\boxed{\begin{aligned}&\frac{\partial(\rho r u_z)}{\partial t}+\frac{\partial \rho_r u_z(u_r-u_r^G)}{\partial r}+\frac{\partial \rho r u_z(u_z-u_z^G)}{\partial z}+\frac{\partial(rp)}{\partial z}\\&-\frac{\partial(r\tau_{zr})}{\partial r}-\frac{\partial(r\tau_{zz})}{\partial z}=0\end{aligned}} \tag{2.12}$$

Energy Equation

$$\boxed{\begin{aligned}&\frac{\partial \varepsilon}{\partial t}+\frac{\partial[\varepsilon(u_r-u_r^G)+pu_r]}{\partial r}+\frac{\partial[\varepsilon(u_z-u_z^G)+pu_z]}{\partial z}+\\&\frac{\varepsilon(u_r-u_r^G)+pu_r}{r}-\frac{\partial(u_r\tau_{rr}+u_z\tau_{rz})}{\partial r}-\frac{\partial(u_z\tau_{zz}+u_r\tau_{rz})}{\partial z}\\&-\frac{u_r\tau_{rr}}{r}-\frac{u_z\tau_{rz}}{r}-\frac{\partial}{\partial r}\left(k\frac{\partial T}{\partial r}\right)-\frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right)-\frac{k}{r}\frac{\partial T}{\partial r}=0\end{aligned}} \tag{2.13}$$

Multiplying by r and rearranging terms, we have

$$\boxed{\begin{aligned}&\frac{\partial(r\varepsilon)}{\partial t}+\frac{\partial[r\varepsilon(u_r-u_r^G)+ru_rp]}{\partial r}+\frac{\partial[r\varepsilon(u_z-u_z^G)+ru_zp]}{\partial z}\\&-\frac{\partial(ru_r\tau_{rr}+ru_z\tau_{rz})}{\partial r}-\frac{\partial(ru_z\tau_{zz}+ru_r\tau_{rz})}{\partial z}\\&-\frac{\partial}{\partial r}\left(rk\frac{\partial T}{\partial r}\right)-\frac{\partial}{\partial z}\left(rk\frac{\partial T}{\partial z}\right)=0\end{aligned}} \tag{2.14}$$

To make these equations consistent with the two–dimensional planar counterpart, the notation is changed from z to x and r to y. Depending upon the arrangement of the terms

in the governing equations, two different forms may be obtained. The first form is concluded from equations (2.7), (2.9), (2.11), and (2.13) as

$$\dot{\boldsymbol{u}} + \boldsymbol{F}^c_{i,i} - \boldsymbol{F}^v_{i,i} + (\boldsymbol{S}^c - \boldsymbol{S}^v) = 0 \tag{2.15}$$

where

$$\boldsymbol{u} = [\rho \ \rho u_1 \ \rho u_2 \ \varepsilon]^T \tag{2.16}$$

$$\boldsymbol{F}^c_i = \begin{bmatrix} \rho\left(u_i - u_i^G\right) \\ \rho u_1\left(u_i - u_i^G\right) + p\delta_{1i} \\ \rho u_2\left(u_i - u^G\right) + p\delta_{2i} \\ \varepsilon\left(u_i - u_i^G\right) + u_i p \end{bmatrix} \tag{2.17}$$

$$\boldsymbol{F}^v_i = \begin{bmatrix} 0 \\ \tau_{1i} \\ \tau_{2i} \\ u_m \tau_{mi} - q_i \end{bmatrix} \tag{2.18}$$

$$\boldsymbol{S}^c = \frac{1}{y}\begin{bmatrix} \rho\left(u_2 - u_2^G\right) \\ \rho u_1\left(u_2 - u_2^G\right) \\ \rho u_2\left(u_2 - u_2^G\right) \\ \varepsilon\left(u_2 - u_2^G\right) + p u_2 \end{bmatrix} \tag{2.19}$$

$$\boldsymbol{S}^v = \frac{1}{y} = \begin{bmatrix} 0 \\ \tau_{12} \\ 2\mu\left(u_{2,2} - \dfrac{u_2}{y}\right) \\ u_m \tau_{m2} + kT_{,2} \end{bmatrix} \tag{2.20}$$

and the viscous shear stresses are

$$\tau_{11} \ = \ \mu_R u_{1,1} + \lambda u_{2,2} \tag{2.21}$$

$$\tau_{22} = \mu_R u_{2,2} + \lambda u_{1,1} \tag{2.22}$$

$$\tau_{21} = \tau_{12} = \mu \left(u_{1,2} + u_{2,1} \right) \tag{2.23}$$

$$\mu_R = (2\mu + \lambda) = \text{ longitudinal viscosity} \tag{2.24}$$

It is observed that in this version, both convective and viscous fluxes are identical to the two–dimensional planar counterpart and the additional source terms include the grid velocity. The advantage of applying these equations in the code development is that only the source terms need to be incorporated into the code and, therefore, the code integrity and efficiency may be maintained. Unfortunately, it can be shown that this form is not conservative, and thus unacceptable for the numerical implementation.

The second form is derived from equations (2.8), (2.10), (2.12), and (2.14) as

$$\dot{\hat{\boldsymbol{u}}} + \widehat{\boldsymbol{F}}^c_{i,i} - \widehat{\boldsymbol{F}}^v_{i,i} + \boldsymbol{S}^c - \boldsymbol{S}^v = 0 \tag{2.25}$$

where

$$\hat{\boldsymbol{u}} = y\boldsymbol{u} = y \left[\rho \ \rho u_1 \ \rho u_2 \ \varepsilon \right]^T \tag{2.26}$$

$$\widehat{\boldsymbol{F}}^c_i = y\boldsymbol{F}^c_i = y \begin{bmatrix} \rho \left(u_i - u^G_i \right) \\ \rho u_1 \left(u_i - u^G_i \right) + p\delta_{1i} \\ \rho u_2 \left(u_i - u^G_i \right) + p\delta_{2i} \\ \varepsilon \left(u_i - u^G_i \right) + u_i p \end{bmatrix} \tag{2.27}$$

$$\widehat{\boldsymbol{F}}^v_i = y\boldsymbol{F}^v_i = y \begin{bmatrix} 0 \\ \tau_{1i} \\ \tau_{2i} \\ u_m \tau_{mi} - q_i \end{bmatrix} \tag{2.28}$$

$$\boldsymbol{S}^c = -\left[0 \ 0 \ p \ 0 \right]^T \tag{2.29}$$

$$\boldsymbol{S}^v = -\left[0 \ 0 \ \frac{\mu_R u_2}{y} + \lambda u_{m,m} \ 0 \right]^T \tag{2.30}$$

and the viscous shear stresses are

$$\tau_{11} = \mu_R u_{1,1} + \lambda u_{2,2} + \frac{\lambda u_2}{y} \tag{2.31}$$

$$\tau_{22} = \mu_R u_{2,2} + \lambda u_{1,1} + \frac{\lambda u_2}{y} \tag{2.32}$$

$$\tau_{21} = \tau_{12} = \mu \left(u_{1,2} + u_{2,1} \right) \tag{2.33}$$

$$\mu_R = (2\mu + \lambda) = \text{ longitudinal viscosity} \tag{2.34}$$

This form has the advantage that the grid velocity makes no contribution to the source terms and there is no formal difference between the axisymmetric and planar formulations due to the grid velocity. Although this form is conservative and will be employed for the code development, the finite element interpolation based on $\widehat{\boldsymbol{u}} = y \left(\rho \ \rho u_1 \ \rho u_2 \ \varepsilon \right)^T$ will cause severe numerical problems near the axis of singularity, $y = 0$, since the conservation variables $\boldsymbol{u} = \left(\rho \ \rho u_1 \ \rho u_2 \ \varepsilon \right)^T$ for all nodes that lie on this axis can not be recovered directly from $\widehat{\boldsymbol{u}}$. We have resolved this problem by interpolating $\boldsymbol{u}$ and y separately, see Section 4.1 for a detailed description.

3 Boundary Conditions

It is well known that the well-posedness of fluid dynamic problems can not be established by solely considering the governing equations without investigating the associated boundary conditions [2,3,4,5]. Moreover, the accuracy and the rate of convergence of a numerical algorithm are also affected by the proper treatment of the numerical boundary conditions [6–10]. On the other hand, the ability of a CFD code to simulate fluid dynamic problems is strongly dependent on the flexibility of treating various types of boundary conditions. In this report, instead of presenting a detailed mathematical background about these boundary conditions, we will rather point out various types of boundary conditions that are most often encountered in fluid dynamic problems and discuss how these boundary conditions are implemented within the context of finite element methods.

The following boundary conditions are common to most fluid dynamic problems:

1. Open boundaries or inflow/outflow boundaries.
2. No–flow or no–penetration boundary.
3. No–slip isothermal/adiabatic boundary.
4. Porous wall boundary with mass injection.

5. Moving boundary with prescribed velocity.

Boundary condition type 1 is usually encountered when artificial boundaries are introduced to a problem in order to reduce the computational domain. In this case, boundary conditions have to be treated carefully to handle wave reflection and disturbance propagation for both accuracy and rate of convergence considerations [6–10].

Boundary condition type 2 is popular for inviscid flow problems and for problems involving symmetric geometry. On this type of boundary, flow is prohibited from penetrating the boundary and is allowed only in the direction tangent to the boundary. This condition is quite natural for solid boundaries immersed in an inviscid flow.

Boundary condition type 3 is a typical boundary condition for viscous flow problems. The isothermal boundary simply implies that the temperature of the wall is a constant, while adiabatic boundary means that the heat flux across the wall is zero.

Boundary conditions 4 and 5 are special features associated with the numerical simulation of eroding or moving boundaries. Generally speaking, the eroding surface of a solid propellant may be regarded as a porous wall with boundary motion. The burning rate and the mass flow rate across the surface are, theoretically, dependent on the local flow conditions such as pressure and temperature and the composition of the solid propellant. A notable empirical formula known as Saint-Robert's law which correlates the burning rate and chamber pressure is given by [11]. In cases where chemical reactions and combustion phenomena are not considered, the surface of the solid propellant is usually treated as a porous wall with mass injection. the mass injection rate being obtained from experimental data.

To avoid ambiguity in identifying the type of boundary condition at a node where two different types of boundaries intersect, a hierarchical procedure is set up as follows based on their relative priority:

1. porous wall boundary with boundary motion,

2. no–slip boundary condition,

3. no–flow or no–penetration boundary condition, and

4. open boundary condition.

A detailed discussion of the theoretical formulation and implementation of most of these boundary conditions can be found in reference [20]. In this work we will discuss boundary conditions specific to the solid rocket booster applications, namely porous/burning wall conditions (Section 4.4).

4 Numerical Methods

Over the past few years, significant progress has been made in the development of new computational methods for solving compressible Navier-Stokes equations. The approaches described in the literature vary from fully explicit algorithms, which are computationally inexpensive but often severely limited by stability restrictions, to fully implicit algorithms, which are unconditionally stable but are much more expensive per time step. The selection of which type of algorithms is optimal for a given application is generally not known *a priori* and may in fact change as the features of the flowfield develop. As a result, a domain decomposition approach is gaining popularity in the CFD community which employs an explicit formulation in one region of the mesh and a fully implicit formulation in another, see references [12–15,20].

In this section, a family of implicit/explicit finite element algorithms developed by Tworzydlo, Oden, and Thornton [20] will be generalized for the solution of axisymmetric problems with moving meshes and porous/burning wall boundary conditions. Within this family, a fully explicit method or various versions of implicit algorithms can be obtained by appropriate selection of implicitness parameters. This general finite element algorithm is combined with adaptive mesh refinement (as presented in references [18,19]) and adaptive selection of implicit/explicit zones within the computational domain. Several approaches to the selection of implicit and explicit zones are presented.

Following the description of implicit/explicit schemes, the artificial dissipation added to the system to suppress possible spurious solutions (due to shocks) will be outlined. The final solution describes the implementation of the porous wall boundary condition and pressure outflow boundary condition.

4.1 A General Family of Implicit Taylor-Galerkin Methods

In this section a general family of implicit Taylor-Galerkin methods will be derived. This family is based on a combination of second–order Taylor series expansions in time with a Galerkin approximation in space (one–, two–, or three–dimensional). Several implicitness parameters are introduced, so that, depending on the particular choice, a fully explicit scheme or a variety of implicit schemes can be recovered. The family of algorithms presented here is a generalization to axisymmetric problems of the algorithms developed previously.

Second–Order Taylor Expansion in Time

Assume that the solution $\widehat{\boldsymbol{u}}^n$ is given at the time moment t^n and the solution at time t^{n+1} is to be calculated. Formally, the values of the solution at moments t^n and t^{n+1} can be expressed

by the second–order Taylor expansion around an arbitrary moment $t^{n+\alpha}$ (see Fig. 4.1), where α is the implicitness parameter with values between zero and one:

$$\begin{aligned} \hat{\boldsymbol{u}}^{n+1} &= \hat{\boldsymbol{u}}^{n+\alpha} + (1-\alpha)\Delta t\dot{\hat{\boldsymbol{u}}}^{n+\alpha} + (1-\alpha)^2\frac{\Delta t^2}{2}\ddot{\hat{\boldsymbol{u}}}^{n+\alpha} + O(\Delta t^3) \\ \hat{\boldsymbol{u}}^{n} &= \hat{\boldsymbol{u}}^{n+\alpha} - \alpha\Delta t\dot{\hat{\boldsymbol{u}}}^{n+\alpha} + \alpha^2\frac{\Delta t^2}{2}\ddot{\hat{\boldsymbol{u}}}^{n+\alpha} + O(\Delta t^3) \end{aligned} \tag{4.1}$$

By subtracting these two formulas one obtains a formula for the increment of the solution between steps n and $n+1$:

$$\Delta\hat{\boldsymbol{u}} = \hat{\boldsymbol{u}}^{n+1} - \hat{\boldsymbol{u}}^{n} = \Delta\dot{\hat{\boldsymbol{u}}}^{n+\alpha} + (1-2\alpha)\frac{\Delta t^2}{2}\ddot{\hat{\boldsymbol{u}}}^{n+\alpha} + O(\Delta t^3) \tag{4.2}$$

Now it is easy to observe that:

$$\ddot{\hat{\boldsymbol{u}}}^{n+\alpha} = \ddot{\hat{\boldsymbol{u}}}^{n+\beta} + O((\alpha-\beta)\Delta t)$$

so that—still preserving second–order accuracy—a second implicitness parameter β can be introduced into equation (4.2):

$$\Delta\hat{\boldsymbol{u}} = \Delta t\dot{\hat{\boldsymbol{u}}}^{n+\alpha} + (1-2\alpha)\frac{\Delta t^2}{2}\ddot{\hat{\boldsymbol{u}}}^{n+\beta} + O(\Delta t^3) \tag{4.3}$$

The next step of the derivation is to express the quantities evaluated at time moments $t^{n+\alpha}$ and $t^{n+\beta}$ by quantities evaluated at the basic steps t^n and t^{n+1}. It is easy to show, using a Taylor series expansion, that:

$$\left.\begin{aligned} \dot{\hat{\boldsymbol{u}}}^{n+\alpha} &= \dot{\hat{\boldsymbol{u}}}^{n} + \alpha\boldsymbol{\Delta}\dot{\hat{\boldsymbol{u}}} + O(\Delta t^2) \\ \ddot{\hat{\boldsymbol{u}}}^{n+\beta} &= \ddot{\hat{\boldsymbol{u}}}^{n} + \beta\Delta\ddot{\hat{\boldsymbol{u}}} + O(\boldsymbol{\Delta} t^2) \end{aligned}\right\}$$

Substituting these formulas into equation (4.2) yields a two–parameter expansion:

$$\boldsymbol{\Delta}\hat{\boldsymbol{u}} = \Delta t\left(\dot{\hat{\boldsymbol{u}}}^{n} + \alpha\boldsymbol{\Delta}\dot{\hat{\boldsymbol{u}}}\right) + (1-2\alpha)\frac{\Delta t^2}{2}\left(\ddot{\hat{\boldsymbol{u}}}^{n} + \beta\boldsymbol{\Delta}\ddot{\hat{\boldsymbol{u}}}\right) + O(\Delta t^3) \tag{4.4}$$

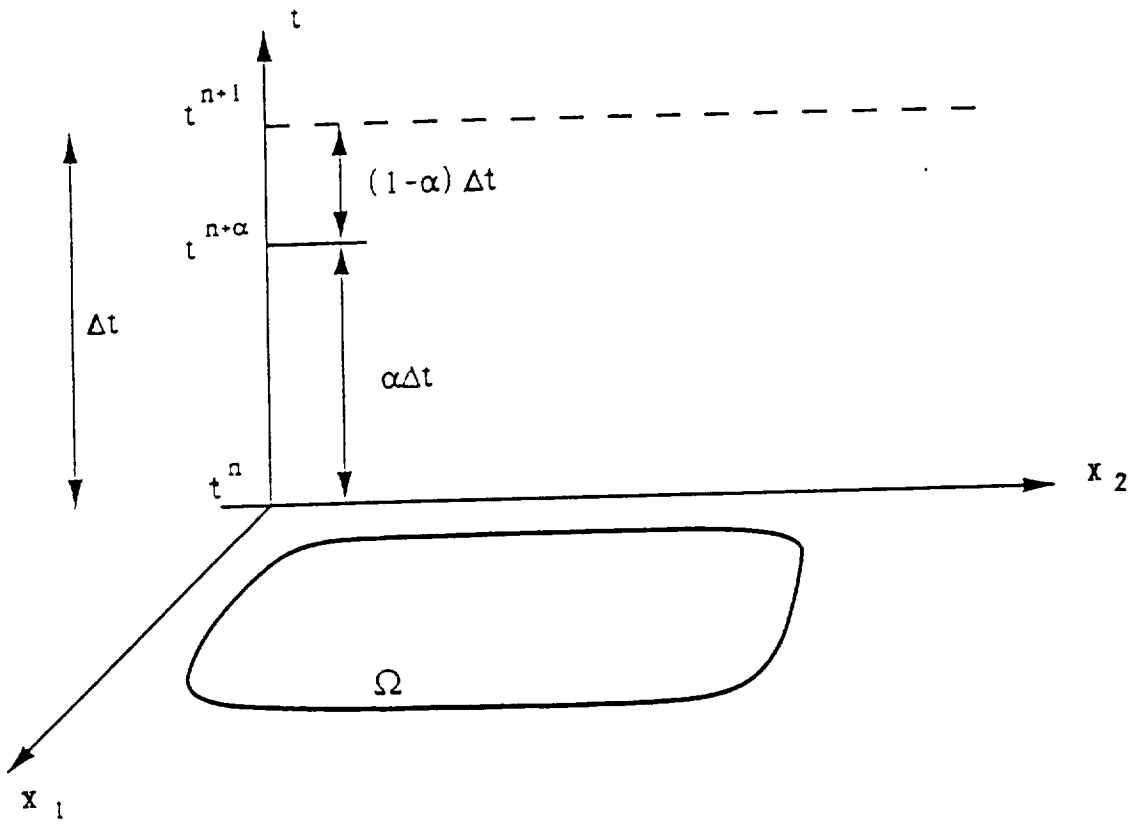


Figure 4.1: General Taylor series expansion in time

Now, following the original idea of Lax and Wendroff, the original equation (4.1) will be substituted to equation (4.4) to replace time derivatives by space derivatives. This substitution yields a formula for the first derivatives:

$$\dot{\hat{\boldsymbol{u}}} = \widehat{\boldsymbol{F}}^{v}_{i,i} - \widehat{\boldsymbol{F}}^{c}_{i,i} + \boldsymbol{S}^{v} - \boldsymbol{S}^{c} \tag{4.5}$$

$$\begin{aligned} \Delta\dot{\hat{\boldsymbol{u}}} &= \Delta\widehat{\boldsymbol{F}}^{v}_{i,i} - \Delta\widehat{\boldsymbol{F}}^{c}_{i,i} + \Delta\boldsymbol{S}^{v} - \Delta\boldsymbol{S}^{c} \\ &= \left(\widehat{\boldsymbol{R}}_{ij}\Delta\hat{\boldsymbol{u}}_{,j}\right)_{,i} + \left(\widehat{\boldsymbol{P}}_{i}\Delta\hat{\boldsymbol{u}}\right)_{,i} - \left(\widehat{\boldsymbol{A}}_{i}\Delta\hat{\boldsymbol{u}}\right)_{,i} \\ &+ \widehat{\boldsymbol{Q}}_{j}\Delta\hat{\boldsymbol{u}}_{,j} + \hat{\boldsymbol{T}}\Delta\hat{\boldsymbol{u}} - \widehat{\boldsymbol{B}}\Delta\hat{\boldsymbol{u}} \end{aligned} \tag{4.6}$$

and for the second derivatives:

$$\begin{aligned} \ddot{\hat{\boldsymbol{u}}} &= \dot{\widehat{\boldsymbol{F}}}^{v}_{i,i} - \dot{\widehat{\boldsymbol{F}}}^{c}_{i,i} + \dot{\boldsymbol{S}}^{v} - \dot{\boldsymbol{S}}^{c} \\ &= \left(\widehat{\boldsymbol{R}}_{ij}\dot{\hat{\boldsymbol{u}}}_{,j} + \widehat{\boldsymbol{P}}_{i}\dot{\hat{\boldsymbol{u}}}\right)_{,i} - \left(\widehat{\boldsymbol{A}}_{i}\dot{\hat{\boldsymbol{u}}}\right)_{,i} \\ &+ \left(\widehat{\boldsymbol{Q}}_{j}\dot{\hat{\boldsymbol{u}}}_{,j} + \hat{\boldsymbol{T}}\dot{\hat{\boldsymbol{u}}}\right) + \widehat{\boldsymbol{B}}\dot{\hat{\boldsymbol{u}}} \end{aligned} \tag{4.7}$$

where the relevant Jacobians are defined as:

$$\begin{aligned} \widehat{\boldsymbol{R}}_{ij} &= \frac{\partial\widehat{\boldsymbol{F}}^{v}_{i}}{\partial\hat{\boldsymbol{u}}_{,j}} \quad , \quad \widehat{\boldsymbol{P}}_{i} = \frac{\partial\widehat{\boldsymbol{F}}^{v}_{i}}{\partial\hat{\boldsymbol{u}}} \\ \widehat{\boldsymbol{A}}_{i} &= \frac{\partial\widehat{\boldsymbol{F}}^{c}_{i}}{\partial\hat{\boldsymbol{u}}} \quad , \quad \widehat{\boldsymbol{Q}}_{j} = \frac{\partial\boldsymbol{S}^{v}}{\partial\hat{\boldsymbol{u}}_{,j}} \\ \hat{\boldsymbol{T}} &= \frac{\partial\boldsymbol{S}^{v}}{\partial\hat{\boldsymbol{u}}} \quad , \quad \widehat{\boldsymbol{B}} = \frac{\partial\boldsymbol{S}^{c}}{\partial\hat{\boldsymbol{u}}} \end{aligned}$$

It can be shown that the multiplier, y, will not affect the linearization of nonlinear inviscid fluxes. Moreover, Jacobians $\widehat{\boldsymbol{R}}_{ij}$ will remain the same as its two–dimensional planar counterpart. This implies that

$$\widehat{\boldsymbol{R}}_{ij} = \boldsymbol{R}_{ij} \ , \qquad \widehat{\boldsymbol{A}}_{i} = \boldsymbol{A}_{i}$$

where $\boldsymbol{R}_{ij}$ and $\boldsymbol{A}_{i}$ are the planar counterpart and can be found in reference [23,24]. The Jacobians $\widehat{\boldsymbol{P}}_{i}$ can be expressed as the sum of two components as

$$\widehat{\boldsymbol{P}}_{i} = \boldsymbol{P}_{i} + \boldsymbol{P}^{A}_{i}$$

where $\boldsymbol{P}_i$ is the two-dimensional planar counterpart and the additional term $\boldsymbol{P}_i^A$ is due to the axisymmetric formulation. Details of $\widehat{\boldsymbol{P}}_i$, $\widehat{\boldsymbol{Q}}_j$, $\hat{\boldsymbol{T}}$, and $\widehat{\boldsymbol{B}}$ are given in Appendix A.

After further substitutions, the second order time derivative can be expressed as

$$\begin{aligned}
\ddot{\hat{\boldsymbol{u}}} &= \left[\boldsymbol{R}_{ij}\left(\widehat{\boldsymbol{F}}^v_{k,k} - \widehat{\boldsymbol{F}}^c_{k,k} + \boldsymbol{S}^v - \boldsymbol{S}^c\right)_{,j}\right]_{,i} \\
&- \left[\widehat{\boldsymbol{P}}_i\left(\widehat{\boldsymbol{F}}^v_{k,k} - \widehat{\boldsymbol{F}}^c_{k,k} + \boldsymbol{S}^v - \boldsymbol{S}^c\right)\right]_{,i} \\
&+ \left[\boldsymbol{A}_i\left(\widehat{\boldsymbol{F}}^v_{k,k} - \widehat{\boldsymbol{F}}^c_{k,k} + \boldsymbol{S}^v - \boldsymbol{S}^c\right)\right]_{,i} \\
&+ \left[\widehat{\boldsymbol{Q}}_j\left(\widehat{\boldsymbol{F}}^v_{k,k} - \widehat{\boldsymbol{F}}^c_{k,k} + \boldsymbol{S}^v - \boldsymbol{S}^c\right)\right]_{,i} \\
&- \left[\hat{\boldsymbol{T}}\left(\widehat{\boldsymbol{F}}^v_{k,k} - \widehat{\boldsymbol{F}}^c_{k,k} + \boldsymbol{S}^v - \boldsymbol{S}^c\right)\right] \\
&- \left[\widehat{\boldsymbol{B}}\left(\widehat{\boldsymbol{F}}^v_{k,k} - \widehat{\boldsymbol{F}}^c_{k,k} + \boldsymbol{S}^v - \boldsymbol{S}^c\right)\right]
\end{aligned} \tag{4.8}$$

Since bilinear elements are used in the finite element code, any spatial derivatives higher than order 3 will be neglected in equation (4.8). This results in the following approximation for the second order time derivatives.

$$\begin{aligned}
\ddot{\hat{\boldsymbol{u}}} &= \left(\boldsymbol{A}_i\widehat{\boldsymbol{F}}^c_{k,k}\right)_{,i} + \left(\boldsymbol{A}_i\boldsymbol{S}^c\right)_{,i} + \widehat{\boldsymbol{B}}\widehat{\boldsymbol{F}}^c_{k,k} + \widehat{\boldsymbol{B}}\boldsymbol{S}^c + O(\mu,k) \\
\ddot{\hat{\boldsymbol{u}}} &= \left(\boldsymbol{A}_i\boldsymbol{A}_k\hat{\boldsymbol{u}}_{,k}\right)_{,i} + \left(\boldsymbol{A}_i\widehat{\boldsymbol{B}}\hat{\boldsymbol{u}}\right)_{,i} + \widehat{\boldsymbol{B}}\boldsymbol{A}_k\hat{\boldsymbol{u}}_{,k} + \widehat{\boldsymbol{B}}\widehat{\boldsymbol{B}}\hat{\boldsymbol{u}} + O(\mu,k)
\end{aligned} \tag{4.9}$$

and the incremental form of the second order time derivative can be expressed as

$$\begin{aligned}
\Delta\ddot{\hat{\boldsymbol{u}}} &= \left(\boldsymbol{A}_i\boldsymbol{A}_k\Delta\hat{\boldsymbol{u}}_{,k}\right)_{,i} + \left(\boldsymbol{A}_i\widehat{\boldsymbol{B}}\Delta\hat{\boldsymbol{u}}\right)_{,i} + \widehat{\boldsymbol{B}}\boldsymbol{A}_k\Delta\hat{\boldsymbol{u}}_{,k} \\
&+ \widehat{\boldsymbol{B}}\widehat{\boldsymbol{B}}\Delta\hat{\boldsymbol{u}} + O(\Delta t) + o(\mu,k)
\end{aligned} \tag{4.10}$$

Substituting (4.5)-(4.6) and (4.9)-(4.10) into (4.2) and regrouping explicit and implicit terms, we get

$$\begin{aligned}
&\Delta\hat{\boldsymbol{u}} + \alpha\Delta t\left(\boldsymbol{A}_i\Delta\hat{\boldsymbol{u}}\right)_{,i} + \alpha\Delta t\widehat{\boldsymbol{B}}\Delta\hat{\boldsymbol{u}} - \gamma\Delta t\left(\boldsymbol{R}_{ij}\Delta\hat{\boldsymbol{u}}_{,j}\right)_{,i} \\
&- \gamma\Delta t\left(\widehat{\boldsymbol{P}}_i\Delta\hat{\boldsymbol{u}}\right)_{,i} - \gamma\Delta t\widehat{\boldsymbol{Q}}_j\Delta\hat{\boldsymbol{u}}_{,j} - \gamma\Delta t\hat{\boldsymbol{T}}\Delta\hat{\boldsymbol{u}} \\
&- \frac{(1-2\alpha)\beta\Delta t^2}{2}\left[\left(\boldsymbol{A}_i\boldsymbol{A}_k\Delta\hat{\boldsymbol{u}}_{,k}\right)_{,i} + \left(\boldsymbol{A}_i\widehat{\boldsymbol{B}}\Delta\hat{\boldsymbol{u}}\right)_{,i}\right. \\
&\left.+\widehat{\boldsymbol{B}}\boldsymbol{A}_k\Delta\hat{\boldsymbol{u}}_{,k} + \widehat{\boldsymbol{B}}\widehat{\boldsymbol{B}}\Delta\hat{\boldsymbol{u}}\right] = \Delta t\left(\widehat{\boldsymbol{F}}^v_{i,i} - \widehat{\boldsymbol{F}}^c_{i,i} + \boldsymbol{S}^v - \boldsymbol{S}^c\right) \\
&+ \frac{(1-2\alpha)\Delta t^2}{2}\left[\left(\boldsymbol{A}_i\widehat{\boldsymbol{F}}^c_{k,k}\right)_{,i} + \left(\boldsymbol{A}_i\boldsymbol{S}^c\right)_{,i} + \widehat{\boldsymbol{B}}\widehat{\boldsymbol{F}}^c_{k,k} + \widehat{\boldsymbol{B}}\boldsymbol{S}^c\right] \\
&+ (1-2\alpha)O(\mu,k)\Delta t^2 + (\alpha-\gamma)O(\mu,k)\Delta t^2 + O(\Delta t^3)
\end{aligned} \tag{4.11}$$

Notice that a third implicit parameter, γ, was introduced in (4.11) to control the implicitness of the viscous terms without affecting the order of accuracy in time.

Variational Formulation

Neglecting higher order terms and multiplying by an arbitrary test function v through (4.11) and integrating over the domain we get

$$
\begin{aligned}
&\int_\Omega \{\Delta\hat{\boldsymbol{u}}\cdot\boldsymbol{v} + \alpha\Delta t[(-\boldsymbol{A}_i\Delta\hat{\boldsymbol{u}})\boldsymbol{v}_{,i} + (\widehat{\boldsymbol{B}}\Delta\hat{\boldsymbol{u}})\boldsymbol{v}] \\
&+ \gamma\Delta t[(\boldsymbol{R}_{ij}\Delta\hat{\boldsymbol{u}}_{,j})\boldsymbol{v}_{,i} + (\widehat{\boldsymbol{P}}_i\Delta\hat{\boldsymbol{u}})\boldsymbol{v}_{,i} - (\widehat{\boldsymbol{Q}}_j\Delta\hat{\boldsymbol{u}}_{,j})\boldsymbol{v} \\
&- (\widehat{\boldsymbol{T}}\Delta\hat{\boldsymbol{u}})\boldsymbol{v}] + \frac{(1-2\alpha)\beta\Delta t^2}{2}[(\boldsymbol{A}_i\boldsymbol{A}_k\Delta\hat{\boldsymbol{u}}_{,k})\boldsymbol{v}_{,i} \\
&+ (\boldsymbol{A}_i\widehat{\boldsymbol{B}}\Delta\hat{\boldsymbol{u}})\boldsymbol{v}_{,i} - (\widehat{\boldsymbol{B}}\boldsymbol{A}_j\Delta\hat{\boldsymbol{u}}_{,j})\boldsymbol{v} - (\widehat{\boldsymbol{B}}\widehat{\boldsymbol{B}}\Delta\hat{\boldsymbol{u}})\boldsymbol{v}]\}d\Omega \\
&+ \int_{\partial\Omega}\{\alpha\Delta t(n_i\boldsymbol{A}_i\Delta\hat{\boldsymbol{u}})\boldsymbol{v} - \gamma\Delta t[(n_i\boldsymbol{R}_{ij}\Delta\hat{\boldsymbol{u}}_{,j})\boldsymbol{v} \\
&+ (n_i\widehat{\boldsymbol{P}}_i\Delta\hat{\boldsymbol{u}})\boldsymbol{v}] - \frac{(1-2\alpha)\beta\Delta t^2}{2}[(n_i\boldsymbol{A}_i\widehat{\boldsymbol{B}}\Delta\hat{\boldsymbol{u}})\boldsymbol{v} + (n_i\boldsymbol{A}_i\boldsymbol{A}_j\Delta\hat{\boldsymbol{u}}_{,j})\boldsymbol{v}]\}ds \\
&= \int_\Omega\{\Delta t[(\widehat{\boldsymbol{F}}_i^c - \widehat{\boldsymbol{F}}_i^v)\boldsymbol{v}_{,i} - (\boldsymbol{S}^c - \boldsymbol{S}^v)\boldsymbol{v}] \\
&+ \frac{(1-2\alpha)\Delta t^2}{2}[-(\boldsymbol{A}_i\boldsymbol{A}_j\hat{\boldsymbol{u}}_{,j})\boldsymbol{v}_{,i} - (\boldsymbol{A}_i\boldsymbol{S}^c)\boldsymbol{v}_{,i} + (\widehat{\boldsymbol{B}}\boldsymbol{A}_j\hat{\boldsymbol{u}}_{,j})\boldsymbol{v} \\
&+ (\widehat{\boldsymbol{B}}\boldsymbol{S}^c)\boldsymbol{v}]\}d\Omega + \int_{\partial\Omega}\left\{\Delta t n_i(\widehat{\boldsymbol{F}}_i^v - \widehat{\boldsymbol{F}}_i^c)\boldsymbol{v} + \frac{(1-2\alpha)\Delta t^2}{2}[(n_i\boldsymbol{A}_i\boldsymbol{A}_j\hat{\boldsymbol{u}}_{,j})\boldsymbol{v} + (n_i\boldsymbol{A}_i\boldsymbol{S}^c)\boldsymbol{v}]\right\}ds
\end{aligned}
\tag{4.12}
$$

In the variational formulation (4.12), the unknowns selected for finite element interpolation is $\hat{\boldsymbol{u}} = y\boldsymbol{u}$, or in discrete form

$$
\hat{\boldsymbol{u}}(\boldsymbol{x}) = \sum_{I=1}^{N}\hat{u}_I\phi_I(\boldsymbol{x}) \tag{4.13}
$$

The choice of $\hat{\boldsymbol{u}}$ for the finite interpolation leads to the following computational problems: $\hat{\boldsymbol{u}}$ becomes zero as $y \to 0$. This implies that the conservation variable $\boldsymbol{u}$ becomes singular,

$$
\boldsymbol{u} = \lim_{y\to 0}\frac{\hat{u}}{y} = \lim_{y\to 0}\frac{y\boldsymbol{u}}{y} \tag{4.14}
$$

To avoid this singularity problem, we will interpolate y and $\boldsymbol{u}$ separately for $\hat{\boldsymbol{u}}$ so that $\boldsymbol{u}$ becomes the actual vector of unknowns in the problem. The resulting variational form becomes

$$
\begin{aligned}
&\int_\Omega \{y\Delta\boldsymbol{u}\cdot\boldsymbol{v} + \alpha\Delta t[y(-\boldsymbol{A}_i^n\Delta\boldsymbol{u})\boldsymbol{v}_{,i} + y(\widehat{\boldsymbol{B}}\Delta\boldsymbol{u})\boldsymbol{v}] \\
&+ \gamma\Delta t[y(\boldsymbol{R}_{ij}\Delta\boldsymbol{u}_{,j})\boldsymbol{v}_{,i} + (\boldsymbol{R}_{ij}y_{,j}\Delta\boldsymbol{u})\boldsymbol{v}_{,i} + y(\widehat{\boldsymbol{P}}_i\Delta\boldsymbol{u})\boldsymbol{v}_{,i} \\
&- y(\widehat{\boldsymbol{Q}}_j\Delta\boldsymbol{u}_{,j})\boldsymbol{v} - (\widehat{\boldsymbol{Q}}_j y_{,j}\Delta\boldsymbol{u})\boldsymbol{v} - y(\hat{\boldsymbol{T}}\Delta\boldsymbol{u})\boldsymbol{v}] \\
&+ \frac{(1-2\alpha)\beta\Delta t^2}{2}[y(\boldsymbol{A}_i\boldsymbol{A}_j\Delta\boldsymbol{u}_{,j})\boldsymbol{v}_{,i} + (\boldsymbol{A}_i\boldsymbol{A}_j y_{,j}\Delta\boldsymbol{u})\boldsymbol{v}_{,i} \\
&+ y(\boldsymbol{A}_i\widehat{\boldsymbol{B}}\Delta\boldsymbol{u})\boldsymbol{v}_{,i} - y(\widehat{\boldsymbol{B}}\boldsymbol{A}_j\Delta\boldsymbol{u}_{,j})\boldsymbol{v} - (\widehat{\boldsymbol{B}}\boldsymbol{A}_j y_{,j}\Delta\boldsymbol{u})\boldsymbol{v} - y(\widehat{\boldsymbol{B}}\widehat{\boldsymbol{B}}\Delta\boldsymbol{u})\boldsymbol{v}]\}d\Omega \\
&\int_{\partial\Omega}\{\alpha\Delta t y(n_i\boldsymbol{A}_i\Delta\boldsymbol{u})\boldsymbol{v} - \gamma\Delta t[y(n_i\boldsymbol{R}_{ij}\Delta\boldsymbol{u}_{,j})\boldsymbol{v} + (n_i\boldsymbol{R}_{ij}y_{,j}\Delta\boldsymbol{u})\boldsymbol{v} + y(n_i\widehat{\boldsymbol{P}}_i\Delta\boldsymbol{u})\boldsymbol{v}] \qquad (4.15) \\
&- \frac{(1-2\alpha)\beta\Delta t^2}{2}[y(n_i\widehat{\boldsymbol{A}}_i\widehat{\boldsymbol{A}}_j\Delta\boldsymbol{u}_{,j})\boldsymbol{v} + (n_i\boldsymbol{A}_i\boldsymbol{A}_j y_{,j}\Delta\boldsymbol{u})\cdot\boldsymbol{v} + y(n_i\boldsymbol{A}_i\widehat{\boldsymbol{B}}\Delta\boldsymbol{u})\boldsymbol{v}]ds \\
&= \int_\Omega \Delta t[y(\widehat{\boldsymbol{F}}_i^c - \widehat{\boldsymbol{F}}_i^v)\boldsymbol{v}_{,i} - (\boldsymbol{S}^c - \boldsymbol{S}^v)\boldsymbol{v}] - \frac{(1-2\alpha)\Delta t^2}{2}[y(\boldsymbol{A}_i\boldsymbol{A}_j\boldsymbol{u}_{,j})\boldsymbol{v} + (\boldsymbol{A}_i\boldsymbol{A}_j y_{,j}\boldsymbol{u})\boldsymbol{v}_{,i} \\
&+ \boldsymbol{A}_i\boldsymbol{S}^c\boldsymbol{v}_{,i} - y(\widehat{\boldsymbol{B}}\boldsymbol{A}_j\boldsymbol{u}_{,j})\boldsymbol{v} - (\widehat{\boldsymbol{B}}\boldsymbol{A}_j y_{,j}\boldsymbol{u})\boldsymbol{v} - (\widehat{\boldsymbol{B}}\boldsymbol{S}^c)\boldsymbol{v}]d\Omega \\
&+ \int_{\partial\Omega}\{\Delta t y n_i(\widehat{\boldsymbol{F}}_i^v - \widehat{\boldsymbol{F}}_i^c)\boldsymbol{v} + \frac{(1-2\alpha)\Delta t^2}{2}[y(n_i\boldsymbol{A}_i\boldsymbol{A}_j\boldsymbol{u}_{,j})\boldsymbol{v} \\
&+ n_i\boldsymbol{A}_i\boldsymbol{A}_j y_{,j}\boldsymbol{u})\cdot\boldsymbol{v} + (n_i\boldsymbol{A}_i\boldsymbol{S}^c)\cdot\boldsymbol{v}]\}ds
\end{aligned}
$$

Since matrices $\widehat{\boldsymbol{B}}$, $\hat{\boldsymbol{T}}$, $\widehat{\boldsymbol{Q}}$, and $\widehat{\boldsymbol{P}}$ contain multiplier $\frac{1}{y}$ and/or $\frac{1}{y^2}$ (see Appendix A) and $y_{,1} = 0$, $y_{,2} = 1$, the integrals containing these terms in equation (4.15) can be regrouped as:

1. Left hand side interior integrals

$$
\begin{aligned}
&-\int_\Omega \left[b\left(\boldsymbol{B}^2+\boldsymbol{B}\boldsymbol{A}_2\right)+c(\boldsymbol{Q}_2+\boldsymbol{T}_2)\right]\frac{\Delta\boldsymbol{u}\cdot\boldsymbol{v}}{y}d\Omega \\
&+\int_\Omega (a\boldsymbol{B}-c\boldsymbol{T}_1)\Delta\boldsymbol{u}\cdot\boldsymbol{v}d\Omega+\int \boldsymbol{I}(y\Delta\boldsymbol{u}\cdot\boldsymbol{v})d\Omega \\
&+\int_\Omega \left(-a\boldsymbol{A}_i+c\boldsymbol{P}_i\right)\left(y\Delta\boldsymbol{u}\cdot\boldsymbol{v}_{,i}\right)d\Omega \\
&+\int_\Omega \left[(c\boldsymbol{R}_{i2}+b\boldsymbol{A}_i\boldsymbol{A}_2)+(b\boldsymbol{A}_i\boldsymbol{B})+c\boldsymbol{P}_i^A\right](\Delta\boldsymbol{u}\cdot\boldsymbol{v}_{,i})d\Omega \\
&-\int_\Omega \left(c\boldsymbol{Q}_j+b\boldsymbol{B}\boldsymbol{A}_j\right)\left(\Delta\boldsymbol{u}_{,j}\cdot\boldsymbol{v}\right)d\Omega \\
&+\int_\Omega \left(c\boldsymbol{R}_{ij}+b\boldsymbol{A}_i\boldsymbol{A}_j\right)\left(y\Delta\boldsymbol{u}_{,j}\boldsymbol{v}_{,i}\right)d\Omega \\
&\text{where}\quad a=\alpha\Delta t,\ b=\frac{(1-2\alpha)\beta\Delta t^2}{2},\ c=\gamma\Delta t
\end{aligned}
\tag{4.16}
$$

2. Left hand side boundary integrals

$$
\begin{aligned}
&\int_{\partial\Omega}\left(an_i\boldsymbol{A}_i-cn_i\boldsymbol{P}_i\right)\left(y\Delta\boldsymbol{u}\cdot\boldsymbol{v}\right)ds \\
&-\int_{\partial\Omega}\left[(cn_i\boldsymbol{R}_{i2}+bn_i\boldsymbol{A}_i\boldsymbol{A}_2)+bn_i\boldsymbol{A}_i\boldsymbol{B}-cn_i\boldsymbol{P}_i^A\right](\Delta\boldsymbol{u}\cdot\boldsymbol{v})ds \\
&-\int_{\partial\Omega}\left(cn_i\boldsymbol{R}_{ij}+bn_i\boldsymbol{A}_i\boldsymbol{A}_j\right)\left(y\Delta\boldsymbol{u}_{,j}\boldsymbol{v}\right)ds
\end{aligned}
\tag{4.17}
$$

3. Right hand side interior integrals

$$
\begin{aligned}
&\int_\Omega \Delta t \left(\boldsymbol{S}_1^v - \boldsymbol{S}^c\right) \cdot \boldsymbol{v} d\Omega \\
+ &\int_\Omega \left\{ \Delta t \boldsymbol{S}_2^v \boldsymbol{v} + \frac{(1-2\alpha)\Delta t^2}{2} \left[\left(\boldsymbol{B}\boldsymbol{A}_2\right) \boldsymbol{u} \cdot \boldsymbol{v} + \boldsymbol{B}\boldsymbol{S}^c \cdot \boldsymbol{v} \right] \right\} \frac{1}{y} d\Omega \\
+ &\int_\Omega \Delta t \left[y \left(\boldsymbol{F}_i^c - \boldsymbol{F}_i^{vc} \right) \cdot \boldsymbol{v}_{,i} \right] d\Omega \\
- &\int_\Omega \left\{ \Delta t \left(\boldsymbol{F}_i^{vA} \cdot \boldsymbol{v}_{,i} \right) + \frac{(1-2\alpha)\Delta t^2}{2} \left[\boldsymbol{A}_i \left(\boldsymbol{A}_2 \boldsymbol{u} + \boldsymbol{S}^c \right) \cdot \boldsymbol{v}_{,i} \right] \right\} d\Omega \\
+ &\int_\Omega \frac{(1-2\alpha)\Delta t^2}{2} \boldsymbol{B}\boldsymbol{A}_j \left(\boldsymbol{u}_{,j} \cdot \boldsymbol{v} \right) d\Omega \\
- &\int \frac{(1-2\alpha)\Delta t^2}{2} \boldsymbol{A}_i \boldsymbol{A}_j \left(\boldsymbol{u}_{,j} \cdot \boldsymbol{v}_{,i} \right) y d\Omega
\end{aligned}
\tag{4.18}
$$

$$
\begin{aligned}
\text{where} \quad S_1^v &= -[0 \; 0 \; \lambda u_{k,k} \; 0]^T \\
S_2 &= -[0 \; 0 \; \mu_R u_2 \; 0]^T
\end{aligned}
\tag{4.19}
$$

$\boldsymbol{F}_i^{vc}$ is the Cartesian counterpart of viscous fluxes and $\boldsymbol{F}_i^{vA}$ is the additional viscous fluxes associated with the multiplier, $\frac{1}{y}$, where

$$
\begin{aligned}
\boldsymbol{F}_1^{vA} &= [0 \quad \lambda v \quad 0 \quad \lambda u v]^T \\
\boldsymbol{F}_2^{vA} &= [0 \quad 0 \quad \lambda v \quad \lambda v^2]^T
\end{aligned}
\tag{4.20}
$$

4. Right hand side boundary integrals

$$
\begin{aligned}
&\int_{\partial\Omega} \Delta t \left[y n_i \left(\boldsymbol{F}_i^{vc} - \boldsymbol{F}_i^c \right) \cdot \boldsymbol{v} \right] ds + \int_{\partial\Omega} \Delta t n_i \boldsymbol{F}_i^{vA} \cdot \boldsymbol{v} ds \\
+ &\int_{\partial\Omega} \frac{(1-2\alpha)\Delta t^2}{2} \left(n_i \boldsymbol{A}_i \boldsymbol{A}_2 \right) \left(\boldsymbol{u} \cdot \boldsymbol{v} \right) ds \\
+ &\int_{\partial\Omega} \frac{(1-2\alpha)\Delta t^2}{2} n_i \boldsymbol{A}_i \left(\boldsymbol{S}^c \cdot \boldsymbol{v} \right) ds \\
+ &\int_{\partial\Omega} \frac{(1-2\alpha)\Delta t^2}{2} n_i \boldsymbol{A}_i \boldsymbol{A}_j \left(y \boldsymbol{u}_{,j} \cdot \boldsymbol{v} \right) ds
\end{aligned}
\tag{4.21}
$$

4.2 Implicit/Explicit Procedures

The basic idea of implicit/explicit algorithms is simple: combine the two methods to take advantage of the superior features of each. The major advantage of the explicit method is that element computations are relatively inexpensive and simple. Unfortunately this method suffers from stability limitations of the time step, which in some problems leads to prohibitively large numbers of time steps.

The implicit algorithm allows for an application of larger time steps than the explicit method. Moreover, due to the existence of implicit boundary terms, it offers easy and straightforward control of natural boundary conditions, particularly those involving the viscous fluxes. An additional advantage is that with larger time steps no explicit artificial dissipation is necessary, which is very important in the calculation of boundary fluxes, particularly by wall heating rates. The major disadvantage of implicit methods is a much higher cost of element operations and a more complex and expensive solution of the resulting system of equations.

In this section, the formulation and numerical implementation of an adaptive implicit/explicit algorithm for compressible flows is presented. The algorithm will be based on the general family of Taylor-Galerkin methods discussed in the previous sections.

Formulation of Implicit/Explicit Schemes

The algorithms for determining the partition must be designed so as to preserve stability, the conservative properties, and the required order of approximation. We begin by partitioning the domain Ω into subdomains $\Omega^{(E)}$ and $\Omega^{(I)}$ where explicit and implicit schemes are to be applied, respectively.

Some of the possible procedures are examined below.

Procedure I

The first possible approach, applied for example by Hassan, Morgan, and Peraire [25], is based on the following two steps:

1. Perform the explicit step computations on all nodes in the mesh ($\Omega^{(E)} = \Omega$).

2. Perform the implicit computations in the subregions, where the stability criterion for the explicit scheme: $C \leq 1$ is violated. The solution in remaining nodes is "frozen" at this step.

This simple procedure has one basic disadvantage: it appears to be nonconservative and this disturbs the regularity of the solution in the transition zone. This is caused by the fact that during Step 2 the "frozen" explicit nodes impose the actual Dirichlet boundary condition on the edge of the implicit zone. Prescribing the Dirichlet boundary condition for the Navier-Stokes equation means that there must exist an (external) source of fluxes to support the prescribed state of the solution. Since no such external source exists within the computational domain, the solution will not be conservative across the implicit/explicit line. Due to enforcement of the Dirichlet boundary conditions, the solution may exhibit a "ramp" or "kink" along this line.

Procedure II

Again Ω is considered to be the union of two subregions $\Omega^{(I)}$ and $\Omega^{(E)}$ (see Fig. 4.2), such that:

$$\Omega^{(E)} \cap \Omega^{(I)} = \Gamma_{EI} \ , \ \Omega^{(E)} \cup \Omega^{(I)} = \Omega$$

It is convenient to assume that the interface between the two regions coincides with the element boundaries.

It can easily be observed that the differential equations to be solved on the two subregions are different due to different implicitness parameters applied in each zone: $\alpha^{(I)}, \beta^{(I)}, \gamma^{(I)}$ in the implicit zone and $\alpha^{(E)}, \beta^{(E)}, \gamma^{(E)} = 0$ in the explicit zone. Therefore, the variational formulation (4.15), based on the assumption of constant implicitness parameters, cannot be applied to the domain Ω. Instead, it can be applied separately to each subdomain with additional continuation conditions across the interface. These conditions represent continuity of the solution and satisfaction of the conservation laws across the interface and are of the form:

$$\left.\begin{aligned} \boldsymbol{u}^{(E)} &= \boldsymbol{u}^{(I)} \\ \boldsymbol{F}_i^{(E)C} &= \boldsymbol{F}_i^{(I)C} \\ \boldsymbol{A}_i^{(E)} &= \boldsymbol{A}_i^{(I)} \\ \boldsymbol{F}_n^{(E)V} &= \boldsymbol{F}_n^{(I)V} \end{aligned}\right\} \quad \text{on} \quad \Gamma_{EI} \tag{4.22}$$

where index n refers to the outward normal for the corresonding region ($\boldsymbol{n}^{(E)} = -\boldsymbol{n}^{(I)}$). The continuity requirement also pertains to the test function, so that $\boldsymbol{v}^{(E)} = \boldsymbol{v}^{(I)} = \boldsymbol{v}$. Note that for general weak solutions of Euler equations the solution $\boldsymbol{u}$ need not be continuous across the interface. However, for regularized problems and finite element interpolation, the continuity of $\boldsymbol{u}$ is actually satisfied.

If the variational statement is formulated for this problem, then in addition to interior integrals for each subdomain and regular boundary integrals, jump integrals across the in-

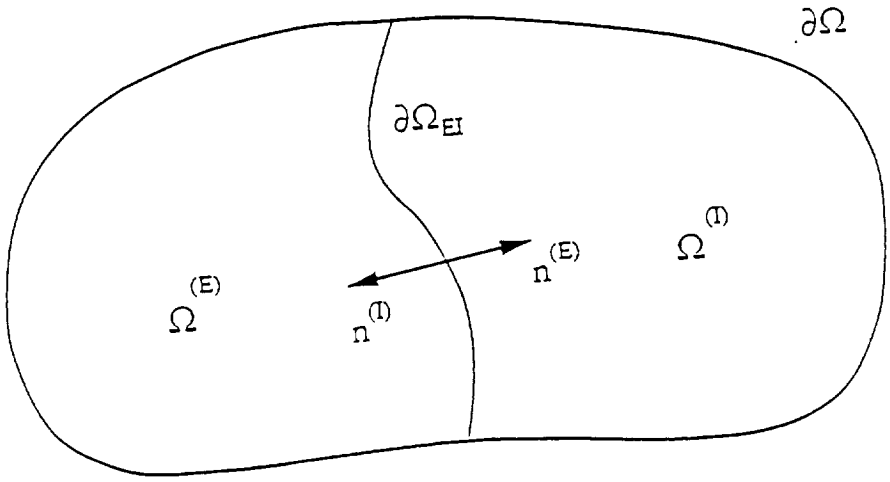


Figure 4.2: Implicit/explicit zones in a computational domain.

terface appear in the formulation. Note that in the practical implementation of this scheme we set all the interface components to zero. This procedure preserves the continuity of fluxes and time accuracy across the interface up to the first order.

Procedure III

The last procedure considered here is based on a generalization of the weak formulation, according to which the implicitness parameters are not constant, but are continuous functions of the position $\boldsymbol{x}$. For the finite element computations, it is convenient to limit the choice of these parameters to the finite element subspace, so that:

$$\alpha(\boldsymbol{x}) = \sum_{I=1}^{N} \alpha_I \Psi_I(\boldsymbol{x})$$

and the same holds for the other implicitness parameters β, γ and δ. With this assumption additional terms show up in the variational formulation. Note that there are no additional boundary terms resulting from the variable implicitness.

The above approach seems to be the most general and clear, with no ambiguities concerning the interface conditions. To make the practical application cheaper, the implicitness coefficients are held constant in most of the elements, and the additional terms are actually evaluated only in the transition zones.

Selection of Implicit and Explicit Zones

The basic criterion for selection of implicit and explicit zones is simple: for a given time step all nodes which violate the stability criterion for an explicit scheme should be treated with the implicit scheme. According to this criterion, several options for an automatic adaptive selection of implicit/explicit zones were implemented:

1. User-prescribed time step DT:

   Within this option, the user prescribes the time step. All nodes satisfying stability criterion for the explicit scheme (with s certain safety factor) are explicit. This means that all the elements connected to these nodes are treated with the explicit scheme. On all other elements the implicit scheme is applied.

2. Prescribed maximum CFL number:

   In this option, the user prescribes the maximum CFL number that can occur for elements in Ω. The time step is automatically selected as the maximum step satisfying this condition. The choice of a maximum CFL number may be suggested by the time

accuracy arguments or the quality of results (it is known that, for a Taylor-Galerkin scheme, too large a CFL number tends to smear shocks).

3. A prescribed percentage of the domain is implicit.

   In this version, the user specifies the fraction of the domain which is to be treated implicitly. The elements with the strongest stability limitation (usually the smallest ones) are treated implicitly, the others are explicit. The time step is selected to guarantee stability of the explicit zone.

4. Minimization of the cost of computations.

   In this option, the time step and the implicit/explicit subzones are selected to minimize the cost of advancing the solution in time (say one time unit). The algorithm is based on the fact that, for an increased time step, an increasing number of elements must be analyzed with the (expensive) implicit algorithm. The typical situation is presented in Fig. 4.3, which shows for different time steps the relative number of nodes that must be treated with the implicit scheme (to preserve stability). On the abscissa, the Δt_{FE} denotes the longest time step allowable for the fully explicit scheme (with certain safety factors). Δt_{FI} denotes the shortest time step requiring a fully implicit procedure. The relative number of implicit nodes increases as a step function from zero for $\Delta t \leq \Delta t_{FE}$ to one for $\Delta t \geq t_{FJ}$. Now assume that the ratio r of the computational cost of processing one implicit node to the cost of processing one explicit node is given. This ratio can be estimated relatively well by comparing the calculation time of element matrices and adding, for implicit nodes, a correction for the solution of the system of equations. Then the reduction of the cost of advancing the solution in time with the implicit/explicit scheme, as compared to the fully explicit scheme, is given by the formula:

$$R(\Delta t) = \frac{\Delta t_{FE}}{\Delta t}\left(n^{(E)} + r n^{(I)}\right)$$

   Typical plots of the function $R(\Delta t)$ are presented in Fig. 4.4. Shown here are the two cases:

   (a) the case of a small difference between fully explicit and fully implicit time steps—an almost uniform mesh

   (b) the case of a large difference between fully explicit and fully implicit time steps

   Note that in either case, restrictions on the length of the time step should be applied, for example, from the maximum CFL condition. Otherwise the cheapest procedure would always be to reach the final time with one implicit step.

   From the plots in Fig. 4.4, the following observation can be made: for an essentially uniform mesh, the mixed implicit/explicit procedure does not provide savings of the

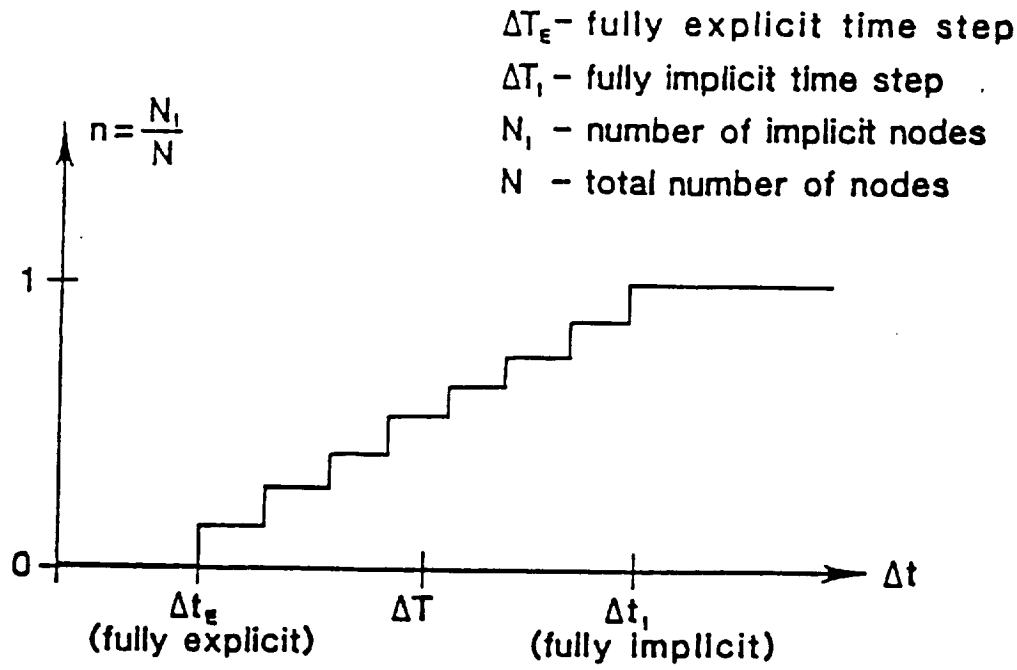


Figure 4.3: Relative number of implicit elements for increasing time step.

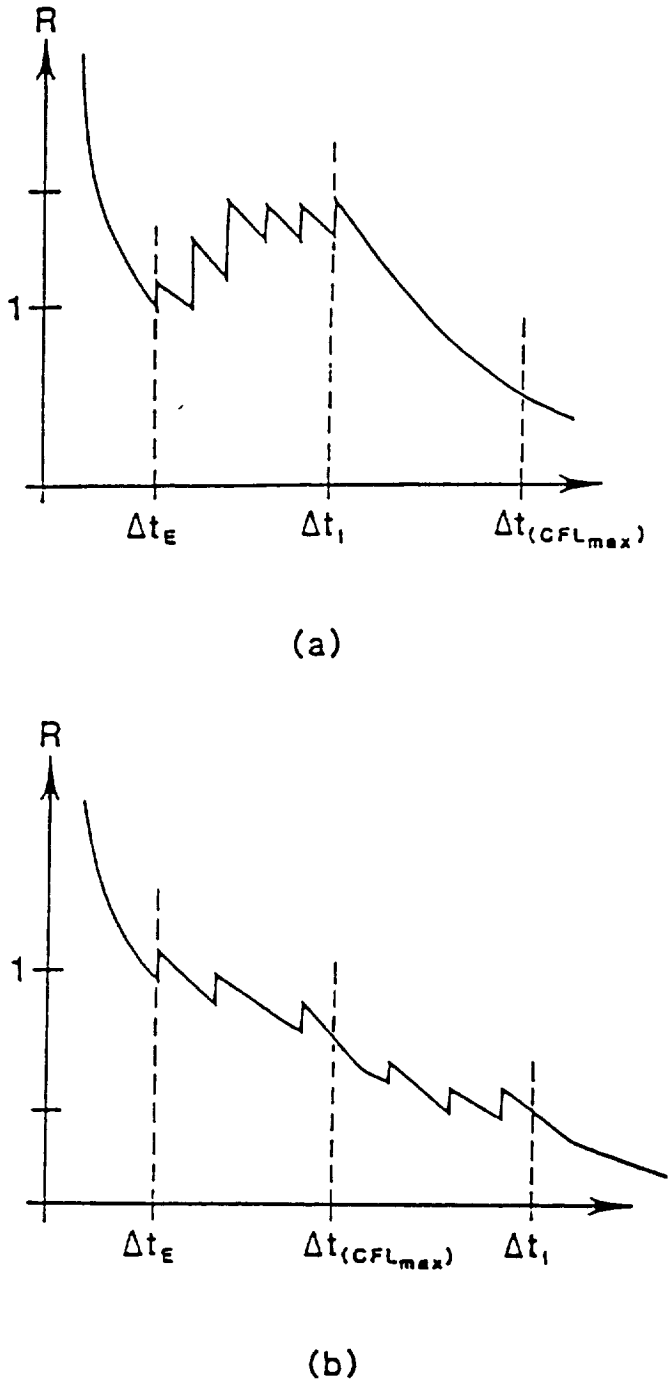


Figure 4.4: Reduction of the cost of computations due to implicit/explicit procedure.

computational cost—either a fully implicit or fully explicit scheme is the cheapest depending on the time step restriction. On the other hand, for very diverse mesh sizes the mixed procedure provides considerable savings. This means that the effectiveness of the mixed implicit/explicit scheme will be the best for large–scale computations with both very large and very small elements present in the domain. In the practical implementation of this method, the approximation of the function $R(\Delta t)$ is automatically estimated for a given mesh. Then, the time step corresponding to the smallest $R(\Delta t)$ is selected automatically (subject to additional constraints, in particular the $\mathrm{CFL}_{\max}$ constraint).

In addition to the above criteria, based purely on a stability analysis, some other criteria for application of implicit schemes can be applied. For example, within boundary layers the implicit scheme may be preferred, because it provides faster convergence of the boundary fluxes and offers direct control of the natural boundary conditions. Many of these issues are yet to be studied.

4.3 Artificial Dissipation

In order to suppress spurious oscillations of the solution, a variety of models of artificial dissipation are used. In this work, we assume that the artificial dissipation can be introduced as the additional flux in the Navier-Stokes equations in the form:

$$\dot{\boldsymbol{u}} + \boldsymbol{F}^C_{i,i} = \boldsymbol{F}^V_{i,i} + \boldsymbol{F}^A_{i,i}$$

where the artificial dissipation flux is the function of the solution vector and its derivatives:

$$\boldsymbol{F}^A_i = \boldsymbol{F}^A_i(\boldsymbol{u}, \boldsymbol{u}_{,j})$$

with corresponding Jacobians defined as

$$\boldsymbol{P}^A_i = \frac{\partial \boldsymbol{F}^A_i}{\partial \boldsymbol{u}}$$

$$\boldsymbol{R}^A_{ij} = \frac{\partial \boldsymbol{F}^A_i}{\partial \boldsymbol{u}_{,j}}$$

The advantage of this approach is that the artificial dissipation can be treated using exactly the same formulation and procedures as for the natural viscosity. In the implicit algorithm, for the sake of generality, a fourth implicitness parameter δ is introduced for the terms associated with the artificial dissipation. In the calculation of the stiffness matrices, right–hand sides and boundary terms, the same formulas are used as for the natural viscosity.

Within the above framework, various models of artificial dissipation can be formulated relatively easily. For example, a straightforward extension of the original Lapidus dissipation [26] to multidimensional cases yields artificial fluxes defined as

$$\boldsymbol{F}_i^A = k^{ii}\boldsymbol{u}_{,j} \tag{4.23}$$

with

$$k^{ii} = c_k A_e \, |v_{i,i}|$$

where c_k is a coefficient (usually between zero and one), A_e is the element area, and v_i are the components of velocity vector (no summation on i). The Jacobians $\boldsymbol{P}$ and $\boldsymbol{R}$ can be defined by a straightforward differentiation of formula (4.23).

Another generalization of the Lapidus concept was proposed by Löhner, Morgan, and Peraire [27]. Within the framework proposed here, the fluxes corresponding to their model are of the form:

$$\boldsymbol{F}^A = \boldsymbol{l}k\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{l}}$$

or

$$\boldsymbol{F}_i^A = kl_i l_j \boldsymbol{u}_{,j} \tag{4.24}$$

where $\boldsymbol{l}$ is the normalized gradient of the magnitude of velocity,

$$\boldsymbol{l} = \frac{grad|\boldsymbol{v}|}{|grad|\boldsymbol{v}||}$$

and the coefficient k is calculated from the formula

$$k = c_k A_e(\boldsymbol{l} \cdot grad(\boldsymbol{v} \cdot \boldsymbol{l}))$$

The Jacobians $\boldsymbol{P}$ and $\boldsymbol{R}$ can be defined by differentiation of formula (4.23). If, for simplicity, dependence of k and $\boldsymbol{l}$ on the solution is disregarded, then:

$$\begin{aligned} \boldsymbol{P}_i &= 0 \\ \boldsymbol{R}_{ij} &= kl_i l_j \boldsymbol{I} \end{aligned} \tag{4.25}$$

where $\boldsymbol{I}$ is the identity matrix of dimension M. In the incremental equation (4.11), the above approach leads to an additional term of the form

$$\Delta t \left[l_i k \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{l}} \right]_{,i}$$

which differs slightly from the original formula proposed by Löhner and Morgan (equation (9) in [27]). These two versions are equivalent only for $\boldsymbol{l}$ constant throughout the domain. It can be verified, however, that the directional derivative used in [27] is not in divergence form and thus cannot be directly used in the variational formulation for arbitrary $\boldsymbol{l}$.

4.4 Implementation of Boundary Conditions

Before discussion of boundary conditions for the implicit Taylor-Galerkin method, it is useful to quote the general result of Strikwerda [3], which specifies the number of boundary conditions necessary for well-posedness of the linearized Euler and Navier-Stokes equations. These results are summarized for two-dimensional problems in the table below.

Type of Boundary	Euler (not regularized)	Navier-Stokes
supersonic inflow	4 ess	4 ess
subsonic inflow	3 ess	3 ess + 1 nat
subsonic outflow	1 ess	1 ess + 2 nat
supersonic outflow	0	0 ess + 3 nat
no-flow	1 ess	1 ess + 2 nat
solid wall		
—isothermal	—	3 ess
—heat flux	—	2 ess + 1 nat

In this table "ess" denotes the essential boundary conditions and "nat" denotes natural boundary conditions. The essential conditions are to be imposed on the characteristic variables rather than the conservation variables. It is of importance to note that the numbers presented in the table are true for problems that are not regularized. If—as is the cae of virtually all computational techniques—some artificial diffusion is built into the algorithm or added explicitly, natural boundary conditions should be imosed on these terms even for Euler problems. Moreover, since artificial diffusion (in contrast to the natural viscosity) affects all the conservation variables, the number of natural boundary conditions for these terms should actually be one more than for the (nonregularized) Navier-Stokes equations.

In the jargon of finite difference methods, essential boundary conditions are equivalent to the boundary conditions to be specified, and the natural boundary conditions are equivalent to the boundary conditions to be extrapolated from the interior of the computational domain.

Within the finite element context, the basic idea of implementing these boundary conditions is based on the concept of constrained minimization. The essential boundary conditions are treated as constraint functions associated with the variational formulation. Penalty methods are usually applied to enforce these conditions in the original system and result in additional stiffness matrices and right hand load vectors. This method has been applied for implementing the open (inflow/outflow) boundary conditions, no–flow boundary conditions, and no–slip boundary conditions.

In this section, the numerical implementation of pressure outflow boundary conditions and porous wall boundary conditions are presented.

Enforcement of Prescribed Pressure

For the case of subsonic outflow with a specified pressure there is one essential boundary condition to be specified, namely the prescribed pressure. The procedure currently being applied in this case is the following:

(a) Apply a supersonic outflow procedure to account for corresponding boundary integrals (continuation from the interior condition).

(b) Impose the one essential boundary condition (pressure) by the penalty method.

The supersonic outflow procedure is described elsewhere and it amounts to rigorous calculation of boundary integrals resulting from variational formulation.

Beginning with the constitutive relation for the pressure

$$p = (\gamma - 1)(E_t - E_k) = \overline{\gamma}(E_t - E_k) \tag{4.26}$$

the condition for a prescribed pressure $\overline{p}$ is (in incremental form)

$$\Delta p = \overline{p} - p^n \tag{4.27}$$

The corresponding penalty term (in energy form) is

$$2\frac{1}{\varepsilon}\left[\Delta p - (\overline{p} - p^n)\right]^2 \tag{4.28}$$

and in variational formulation is

$$\frac{1}{\varepsilon}\left[\Delta p - (\overline{p} - p^n)\right] \cdot \delta(\Delta p) \tag{4.29}$$

where the test function for pressure is its own variation.

Now observe that Δp can be expressed in terms of conservation variables as:

$$\Delta p = \frac{\partial p}{\partial \boldsymbol{u}}\, \boldsymbol{\Delta u} = \boldsymbol{d} \cdot \boldsymbol{\Delta u} \tag{4.30}$$

where $\boldsymbol{d} = \overline{\gamma}\left\{\frac{E_k}{\rho},\ v_1, v_2, 1\right\}^T$ (in the two–dimensional case). Therefore the penalty term (4.29) becomes

$$\frac{1}{\varepsilon}\left[\boldsymbol{d} \cdot \boldsymbol{\Delta u} - (\overline{p} - p^n)\right](\boldsymbol{d} \cdot \boldsymbol{u}) \tag{4.31}$$

where $\boldsymbol{u}$ is a test function. Using the standard requirement that the variational equation be satisfied for every $\boldsymbol{v}$ leads to the *vector* equation:

$$\frac{1}{\varepsilon}\left[\boldsymbol{d} \cdot \boldsymbol{\Delta u} - (\overline{p} - p^n)\right] \boldsymbol{d} = 0 \tag{4.32}$$

and after regrouping

$$\underbrace{\frac{1}{\varepsilon}[\boldsymbol{d}\otimes\boldsymbol{d}]\,\boldsymbol{\Delta u}}_{\text{penalty stiffness matrix}} \quad = \quad \underbrace{\frac{1}{\varepsilon}\boldsymbol{d}\,(\overline{p}-p^n)}_{\text{right hand side}} \tag{4.33}$$

Note: The first component of $\boldsymbol{d}$ has density in the denominator, so that this approach will blow up if the density is close to zero. To stabilize this procedure, one can multiply (4.33) by ρ^2, to get

$$\frac{1}{\varepsilon}\left[\widehat{\boldsymbol{d}}\otimes\widehat{\boldsymbol{d}}\right]\boldsymbol{\Delta u} \quad = \quad \frac{\rho}{\varepsilon}\widehat{\boldsymbol{d}}\,(\overline{p}-p^n) \tag{4.34}$$

where $\widehat{\boldsymbol{d}} = \rho\boldsymbol{d} = \{E_k, m_1, m_2, \rho\}$.

Porous Wall Boundary Conditions

Generally speaking, the burning surface of a solid propellant may be regarded as a porous wall with boundary motion. The burning rate and the mass flow rate across the burning surface are, theoretically, dependent on the local flow conditions such as pressure and temperature and the composition of the solid propellant. In order to implement the porous wall boundary condition with possible boundary motion simultaneously and consistently, a total number of four quantities are prescribed (see Fig. 4.5). These quantities are the tangential and normal velocities of the wall ($\overline{u}_{TW}$ and $\overline{u}_{NW}$), the mass flux across the wall ($\overline{m}_N$), and the temperature of the wall ($\overline{T}_W$). The requirement that these prescribed quantities have to be satisfied at all times on the burning surface results in three boundary conditions to be imposed.

(a) The tangential velocity of the fluid is equal to the tangential velocity of the wall,

$$u_T = \overline{u}_{TW}$$

(b) The balance of the net mass flux,

$$\rho\,(u_N - \overline{u}_{NW}) = \overline{m}_N = \text{normal momentum}$$

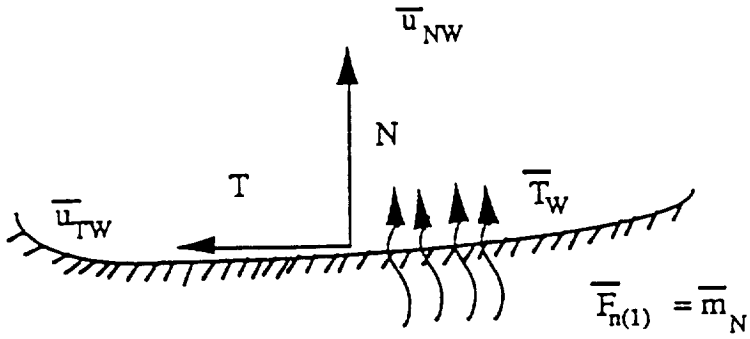


Figure 4.5: Prescribed quantities on a burning boundary.

(c) The temperature of the fluid is equal to the temperature of the wall,

$$T = \overline{T}_W$$

It is important to note that the normal velocity of the fluid is not equal to the normal velocity of the wall due to the prescribed mass flux. The difference of these two quantities contributes to the net mass flux across the wall.

The numerical implementation of these boundary conditions results in the modifications of the stiffness matrices and global right hand side associated with boundary integrals. For condition (a), the incremental form consistent with the incremental time stepping algorithm can be expressed as

$$\Delta u_T = \overline{u}_{TW} - u_T^n \tag{4.35}$$

The constraint function associated with this condition is defined by

$$g = \Delta u_T - (\overline{u}_{TW} - u_T^n) \tag{4.36}$$

The variational form of the penalty term resulting from the constraint function g (tested with its own variation $\delta g = \delta \Delta u_T$) is

$$\frac{1}{\varepsilon}[\Delta u_T - (\overline{u}_{TW} - u_T^n)] \cdot \delta(\Delta U_T) \tag{4.37}$$

The function g tested with the variation of the incremental tangential velocity, (4.35), will result in penalty terms affecting both the continuity and momentum equations. To avoid this unphysical situation, the constraint function, g, may be tested with the variation of the tangential momentum, that is

$$\frac{1}{\varepsilon}\left[\Delta\overline{u}_T - (u_{TW} - u_T^n)\right] \cdot \delta\Delta m_T \tag{4.38}$$

where m_T denotes the tangential component of momentum. Note that

$$\left.\begin{aligned} \Delta u_T &= \frac{\partial U_T}{\partial \boldsymbol{U}} \quad \Delta \boldsymbol{u} = \boldsymbol{d}\Delta\boldsymbol{u} \\ \Delta m_T &= \frac{\partial m_T}{\partial \boldsymbol{U}} \quad \Delta \boldsymbol{u} = \widehat{\boldsymbol{d}}\Delta\boldsymbol{u} \end{aligned}\right\} \tag{4.39}$$

where

$$\left.\begin{aligned} \boldsymbol{d} &= \frac{\partial u_T}{\partial \boldsymbol{u}} = \left[-\frac{u_T}{\rho}, \frac{T_1}{\rho}, \frac{T_2}{\rho}, 0\right] \\ \widehat{\boldsymbol{d}} &= \frac{\partial m_T}{\partial \boldsymbol{u}} = [0,\ T_1,\ T_2,\ 0] \end{aligned}\right\} \tag{4.40}$$

and (T_1, T_2) represent the x, y components of the unit tangential vector. Substituting (4.39) into (4.38), the penalty term becomes

$$\frac{1}{\varepsilon}\left[\boldsymbol{d}\cdot\Delta\boldsymbol{u} - (u_{TW} - u_T^n)\right]\widehat{\boldsymbol{d}}\cdot\boldsymbol{v} \tag{4.41}$$

Since the global variational formulation of the problem (not shown here) has to be satisfied for any test function $\boldsymbol{v}$, the corresponding kernel of the penalty term in the stiffness matrix can be found from (4.41) as

$$\boldsymbol{k} = \frac{1}{\varepsilon}\widehat{\boldsymbol{d}} \otimes \boldsymbol{d} \tag{4.42}$$

and the corresponding right hand side is

$$\boldsymbol{r} = \frac{1}{\varepsilon}\left(u_{TW} - u_T^n\right)\widehat{\boldsymbol{d}} \tag{4.43}$$

The condition (b) to be imposed on the porous wall states that

$$\rho\left(u_N - \overline{u}_{NW}\right) = \overline{m}_N$$

or

$$m_N - \rho\overline{u}_{NW} = \overline{m}_N \tag{4.44}$$

The incremental form of this condition can be written as

$$\Delta m_N - \Delta\rho\overline{u}_{NW} = \overline{m}_N - \left(m_N^n - \rho^n\overline{u}_{NW}\right) \tag{4.45}$$

This results in a constraint function L, given by

$$L = \left(\Delta m_N - \Delta\rho\overline{u}_{NW}\right) - \left[\overline{m}_N - \left(m_N^n - \rho^n\overline{u}_{NW}\right)\right] = 0 \tag{4.46}$$

By applying a typical penalty procedure as described for imposing condition (a), we obtain (L is tested with δm_N) the corresponding kernel, $\boldsymbol{k}$, and the right hand side, $\boldsymbol{r}$, as

$$\boldsymbol{k} = \frac{1}{\varepsilon}\hat{\boldsymbol{d}} \otimes \boldsymbol{d} \tag{4.47}$$

$$\boldsymbol{r} = \frac{1}{\varepsilon}\left[\overline{m}_N - \left(m_N^n - \rho^n\overline{u}_{NW}\right)\right]\boldsymbol{d} \tag{4.48}$$

where

$$\begin{aligned} \boldsymbol{d} &= \frac{\partial L}{\partial U} = [-\overline{u}_{NW},\ N_1,\ N_2,\ 0] \\ \hat{\boldsymbol{d}} &= \frac{\partial m_N}{\partial U} = [0,\ N_1,\ N_2,\ 0] \end{aligned} \tag{4.49}$$

and (N_1, N_2) denote the x, y components of the outward unit normal vector.

For condition (c), we apply the same penalty procedure to enforce the prescribed temperature. The resulting penalty term is (tested with the variation of total energy)

$$\frac{1}{\varepsilon}\left[\Delta T - \left(\overline{T}_W - T^n\right)\right] \cdot \delta(E_t) \tag{4.50}$$

The temperature can be expressed in terms of total energy, E_t, and kinetic energy, E_k, as

$$T = \gamma(\gamma - 1)\left(\frac{E_t}{\rho} - \frac{E_k}{\rho}\right)$$

or

$$T = \gamma(\gamma - 1)\left(\frac{E_t}{\rho} - \frac{m_i m_i}{2\rho^2}\right) \tag{4.51}$$

where m_i represents the momentum components. The standard vector $\boldsymbol{d} = \dfrac{\partial T}{\partial \boldsymbol{u}}$ can be derived from (4.51) as

$$\begin{aligned} d_1 &= \frac{\partial T}{\partial \rho} = \overline{\gamma}\left(-\frac{E_t}{\rho^2} + \frac{E_k}{\rho^2}\right) \\ d_2 &= \frac{\partial T}{\partial m_1} = -\frac{\overline{\gamma} u}{\rho}\ ,\ d_3 = \frac{\partial T}{\partial m_2} = -\frac{\overline{\gamma} v}{\rho} \\ d_4 &= \frac{\partial T}{\partial E_t} = \frac{\overline{\gamma}}{\rho}\ ,\ \overline{\gamma} = \gamma(\gamma - 1) \end{aligned}$$

The corresponding kernel and right hand side result from this condition is

$$\begin{aligned} \boldsymbol{k} &= \frac{1}{\varepsilon}\,\hat{\boldsymbol{d}} \otimes \boldsymbol{d} \\ \boldsymbol{r} &= \frac{1}{\varepsilon}\left(\overline{T}_W - T^n\right)\boldsymbol{d} \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{d} &= \overline{\gamma}\left[-\frac{E_t}{\rho^2} + \frac{E_k}{\rho^2},\ \frac{-u}{\rho},\ \frac{-v}{\rho},\ \frac{1}{\rho}\right] \\ \hat{\boldsymbol{d}} &= [0,\ 0,\ 0,\ 1] \end{aligned}$$

5 Moving Grid/Eroding Boundary Algorithms

As mentioned earlier, the surface of the solid propellant is treated as a porous wall characterized by mass injection and boundary motion. The mass injection rate and the speed of the boundary motion are dependent on the local flow conditions and the composition of the solid propellant. If combustion phenomena are not considered, these quantities are usually determined according to empirical data.

As combustion phenomena has not been included as part of this project, and empirical approach has been implemented whereby the motion/erosion of the solid propellant boundary is prescribed. The moving grid algorithm that incorporates the motions into the deformations of the domain can be stated as follows:

For a given computational domain, Ω, but $\partial\Omega_1$ represent a part of the boundary which is subjected to a prescribed boundary motion and $\partial\Omega_2$ represent the stationary component, such that $\partial\Omega_2 = \partial\Omega_1 \cup \partial\Omega_2$ (see Fig. 5.1).

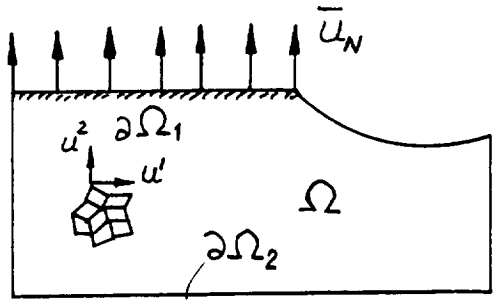


Figure 5.1: Computational domain with a moving boundary.

In order to prevent unacceptable mesh distortions near the moving boundary, we have implemented an algorithm whereby the whole computational mesh is stretched. The grid velocities u^k for the stretching are calculated by solving a boundary–value problem defined by the Laplace operator (which has certain smoothing properties):

$$\nabla^2 u^k = 0 \quad \text{in } \Omega \ , \ k = 1, 2$$

$$u_n = \bar{u}_N \qquad \text{on } \partial\Omega_1$$

$$u_N = 0 \qquad \text{on } \partial\Omega_2$$

where u^k represents the k-th component of grid velocity and $\bar{u}_N$ is a prescribed normal velocity. Note that only the normal component of velocity is prescribed on the moving and stationary boundaries. This allows the grid to stretch along the boundary and better adjust to the erosion process. However, corner nodes belonging to two different boundary sections have two linearly independent normal vectors and thus two components of velocity are automatically prescribed. Note that the two equations are coupled due to the specified normal boundary velocity, $\bar{u}_N$.

A preconditioned block Jacobi-CG iterative method was employed to solve the final system of linear equations for grid velocity. Once the grid velocities for each node point are determined, the new nodal positions are updated by using the time step size and the calculated grid velocity.

$$X^k = X^k + Dt \times u^k$$

Presently, two options for determining the time step size were implemented in:

(a) In the first option, the time step size Dt is calculated from from the fluid stability criteria. Although an accurate transient solution may be obtained, the boundary motion may be extremely slow due to the small time step size and the relatively slow motion of the moving boundary. Thus, this version may require extremely long solution times and high computational costs to move the boundary large distances.

(b) In the second option, we specify a certain amount of time to advance the moving boundary without solving the fluid problem. During the boundary motion, the grid velocities are recalculated periodically to account for changes of the normal vector $\boldsymbol{n}$ and to prevent mesh distortion. Currently, the criterion for the corresponding time interval Dt is that the mesh can move only a prescribed distance before recalculation of the grid velocities. This distance is usually taken as a fraction of the averaged element size.

6 Adaptive Mesh Strategies and Data Management Schemes

Adaptive methods are an efficient means for improving the accuracy of a computational solution. In the implementation of such methods, the relative accuracy of a solution is determined by calculating an error indicator for each element in some appropriate norm. Once an error is determined, adaptive methods are used to change the structure of the approximation of the problem to reduce the error below a preassigned limit. Changing the structure of the approximation may involve increasing or decreasing the number of elements (h-methods), shifting the grid points (r-methods), altering the order of the local finite element approximation (p-methods), or any of several other techniques.

The h-adaptive scheme incorporated into the solid rocket booster code uses a normalized error indicator to estimate the relative magnitude of the local element error. This type of error indicator has proven to have the capability of capturing shock waves with variable strengths and shock wave interactions. This normalized error indicator is defined as

$$\theta_e = h_e \frac{1}{U} \sup_{i=1,2} \left| \frac{\partial U}{\partial x_i} \right| \tag{6.1}$$

where h_e is the measure of the element size and U is the independent variable such as density, pressure, etc., in the error calculations.

Based on this local error indicator, the h-refinement and unrefinement strategy can be summarized by the following steps:

An h-Refinement/Unrefinement Method

The h-procedure involves the following steps:

1. For a given domain Ω, such as that shown in Fig. 6.1, a coarse finite element mesh is constructed which contains only a number of elements sufficient to model basic geometrical features of the flow domain.

2. As the adaptive process is designed to handle groups of four elements at a time, a finer starting grid is generated by by a bisection process, indicated in Fig. 6.1b, to obtain an initial set of element groups.

3. The numerical solution is calculated on this initial coarse grid, and the error indicators θ_e are computed over all M elements in the grid. Let

$$\theta_{\text{MAX}} = \max_{1 \le e \le M} \theta_e$$

4. Next, the groups of elements are scanned and the group errors are computed

$$\theta^k_{\text{GROUP}} = \sum_{k=1}^{P} \theta_{e_k}$$

where e_k is an element number in group k and $P = 4$.

5. Error tolerances are defined by two real numbers, $0 < \alpha, \beta < 1$. If

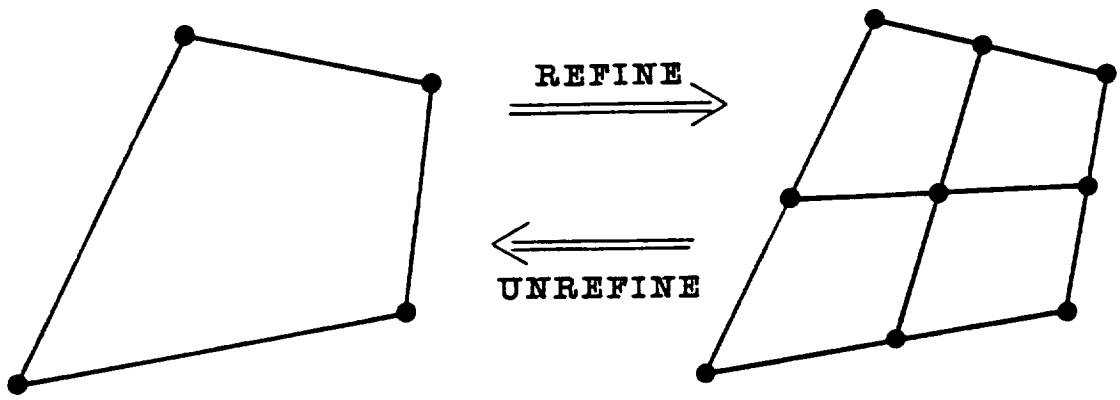
$$\theta_e \ge \beta \theta_{\text{MAX}}$$

element e is **refined**. This is done by bisecting element e into four new subelements. If

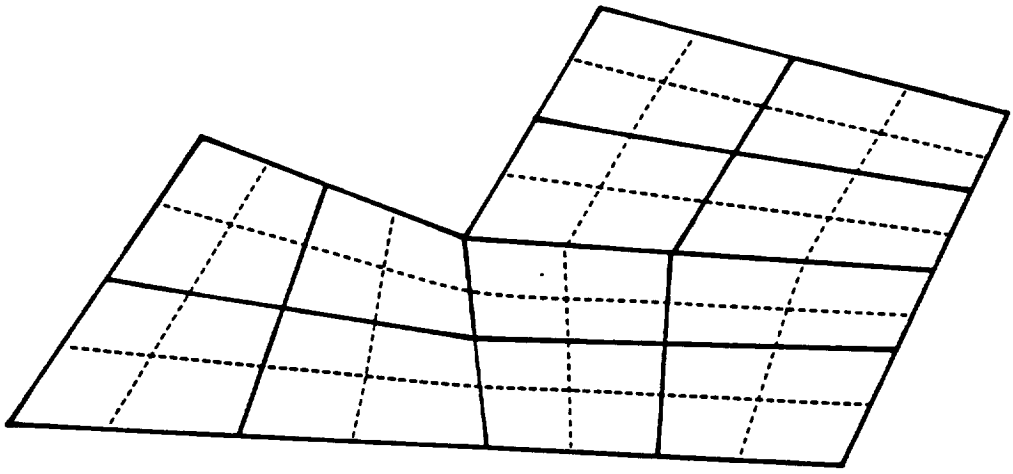
$$\theta^k_{\text{GROUP}} \le \alpha \theta_{\text{MAX}}$$

group k is **unrefined** by replacing this group with a single new element with nodes coincident with the corner nodes of the group.

This general process can be followed for any choice of an error indicator. Moreover, it can also be implemented with any prescribed frequency.



(a)



(b)

Figure 6.1: (a) Refinement and unrefinement of a four–element group, and (b) a coarse initial mesh consisting of a four–element group.

Data Structures

An important consideration in all adaptive schemes is the data structure and associated algorithms needed to handle the changing number of elements, their node locations and numbers, and the element labels.

As noted in the preceding paragraphs, the algorithm is designed to process (refine or unrefine) in groups of four elements at each local refinement/unrefinement step. Consider, for example, the case of an initial mesh of 20 square elements shown in Fig. 6.2. We assign to each element in this mesh an element number, NEL = 1,2,...,NELEM and to each **global** node a lable NODE. This array, NODES(J,NEL) relates the local number J(J = 1,2,3,4) of element NEL to the global node number NODES. In addition, the coordinates X_J, Y_J of each node are also provided relative to a fixed global coordinate system. These numbers are filed in two arrays:

NODES(J,NEL) = the array of global node numbers assigned to node J of element NEL

XCO(JCO,NODE) = the array of JCO — coordinates of global node NODE(JCO = 1 or 2)

Suppose that an error indicator is computed that signals that an element should be refined, say element 11, in the example. We must have some system for assigning appropriate labels to the new elements and nodes. Toward this end, a convention has been established that defines the connectivity of the specified element with its neighbors in the mesh. This information is provided by a third connectivity array.

Thus, the bookkeeping of element and node numbers evolving in a refinement process is monitored by the arrays NODES(. , .) , XCO(. , .) , NELCON(. , .), and an array LEVEL(NEL) which assigns a level number to element NEL. Initially, the same level can be assigned to all elements, and this level is an arbitrary parameter prescribed in advance by the user. Thus, provisions are now in hand for an arbitrary dynamic renumbering of elements and nodes. If, for example, for the mesh in Fig. 6.2 (element 11) is to be refined, we proceed through the following steps:

1. Loop over the neighbors of element 11 (which is made possible with the NELCON array), checking the level of the neighboring elements relative to the level of element 11;

2. If any neighboring element has a level lower than element 11, then element 11 cannot be refined at this stage;

3. If element 11 can be refined (as is the case in Fig. 6.2), generate new element numbers (thus changing NELEM and new node numbers for unconstrained nodes);

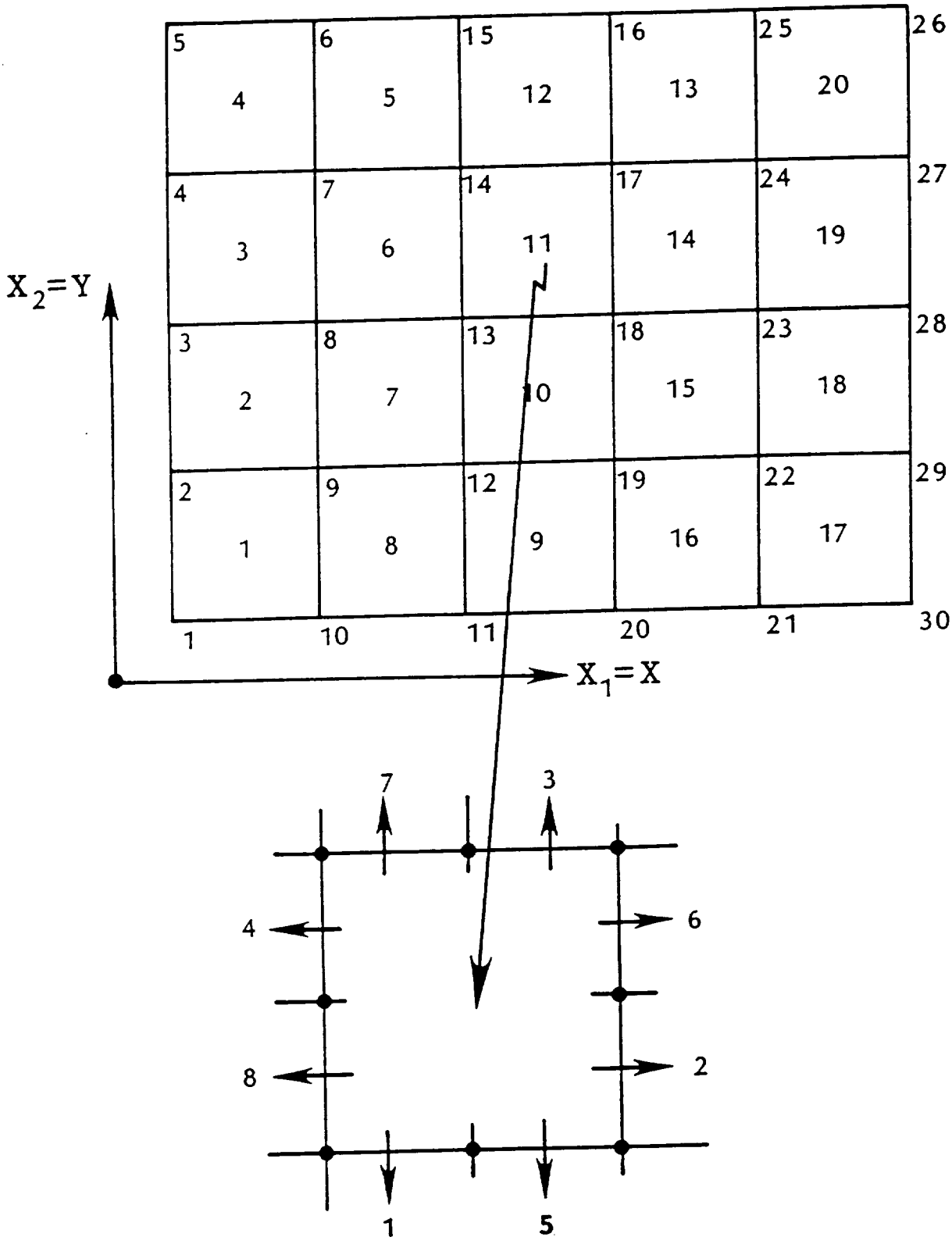


Figure 6.2: Mesh, node, and connectivity numbering in a model problem.

4. Compute the connectivity matrix NELCON for the new elements;

5. Adapt the connectivity matrices for the neighboring elements (since the refinement of element 11 has now changed this connectivity); and

6. Interpolate the solution for the new nodes.

Consider, as an example, the uniform grid of four elements shown in Fig. 6.3a and suppose that the error estimators dictate that element A is to be refined. Thus, A is divided into four elements, I, II, III, IV, as shown, and the solution values at the junction nodes, shown circled in the figure, are constrained to coincide with the averaged values between those marked X. Note that the connectivities change in the process, e.g., the connectivities 4 and 8 of element B are different.

Next, assume that an additional refinement is required, and that we must next refine element III. We impose the restriction that each element side can have no more than two elements connected to it. Thus, before III can be refined, element B must first be refined, as indicated in Fig. 6.3b. The constrained node B1 in Fig. 6.3a now becomes active, while node C1 remains a constrained node. With B bisected, we proceed to refine III into subelements $\alpha, \beta, \gamma, \delta$, and new constrained nodes, again circled in Fig. 6.3c, are produced. In this case, only element B had to be refined first in order to refine III, but, in general, the number of elements that must be refined in order to refine a particular element cannot be specified.

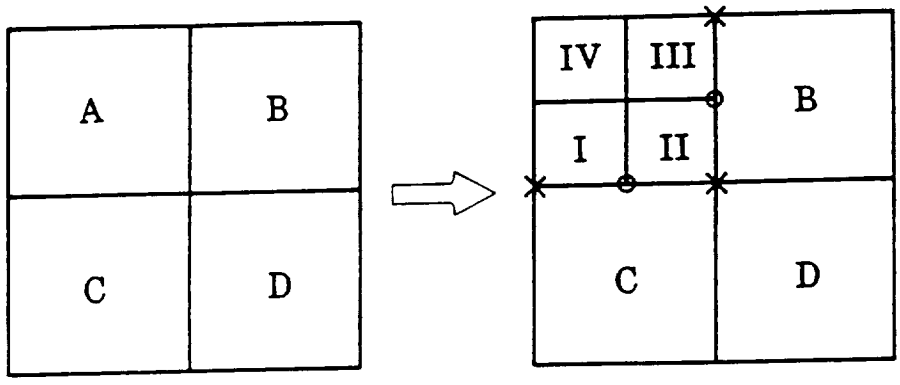
7 Turbulence Modeling

Reynolds–Averaged Navier-Stokes Equations

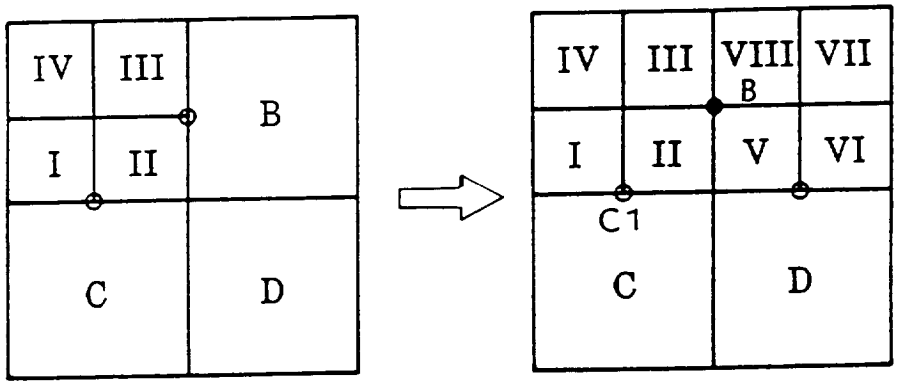
For most engineering analyses, only the mean motion of a turbulent flow is of interest. The governing equations of the mean motion of a turbulent flow are usually derived by applying the Reynolds decomposition technique and an averaging procedure to the Navier-Stokes equations.

The time–dependent, mass–averaged, full Navier-Stokes equations can be expressed as

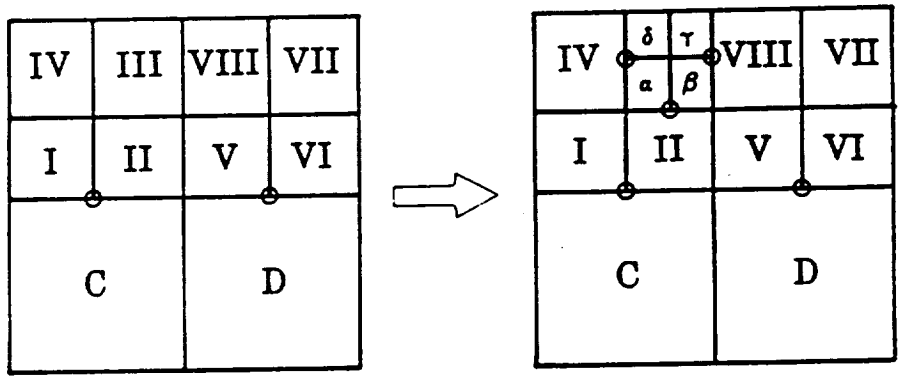
$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \dfrac{\partial \rho \left(u_i - u_i^G\right)}{\partial x_i} = 0 \\ \dfrac{\partial \rho u_i}{\partial t} + \dfrac{\partial}{\partial x_j}\left[\rho u_i \left(u_j - u_j^G\right) - \tau_{ij}\right] = 0 \\ \dfrac{\partial \varepsilon}{\partial t} + \dfrac{\partial}{\partial x_i}\left[\varepsilon \left(u_i - u_i^G\right) - u_j \tau_{ij} + \dot{q}_i\right] = 0 \end{cases}$$



(a)



(b)



(c)

Figure 6.3: Sequence of refinements of uniform mesh.

where

$$\tau_{ij} = -p\delta_{ij} - \frac{2}{3}(\mu + \mu_e)\,\delta_{ij}\frac{\partial u_k}{\partial x_k} + (\mu + \mu_e)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

$$\dot{q}_i = -\left(\frac{\mu}{Pr} + \frac{\mu_e}{Pr_e}\right)\frac{\partial h}{\partial x_i}, \quad h \text{ is the specific enthalpy}$$

and μ_e is the turbulent eddy viscity, Pr and Pr_e are the laminar and turbulent Prandtl numbers, respectively. These equations are exactly the same as for the laminar case except as an additional eddy viscosity has been added to the molecular viscosity. The eddy viscosity may be calculated using a wide range of turbulence models which vary from algebraic models to k-ε models. We have selected a simple algebraic model [29] to implement within the context of the implicit/explicit flow solver described in Section 4.

Algebraic turbulence models, although simple in formulation, are very difficult to implement if complicated geometries are to be handled. The following sections present details of the numerical implementation of Prandtl-van Driest inner layer turbulence model in the context of adaptive unstructured grids.

Prandtl-van Driest Turbulence Model

In the Prandtl-van Driest turbulence model, the turbulent eddy viscosity is calculated according to:

$$\mu_i = \rho * |w| * \ell^2$$

where

ρ = local fluid density
$|w|$ = local magnitude of vorticity
ℓ = Prandtl mixing length

and

$\ell = kyD$
k = 0.4, von Karman constant
y = distance normal to a solid (no-slip) wall
D = van-Driest damping function
$= 1.0 - \exp(-y^+/A)$

and

$$
\begin{aligned}
A &= 26.0, \text{ van-Driest constant} \\
y^+ &= \text{wall function} = \rho_w \cdot v_w \, y/\mu_w \\
\rho_w &= \text{density on the wall} \\
\mu_w &= \text{laminar viscosity on the wall} \\
v_w &= \text{viscous wall velocity} \\
&= \sqrt{(\tau_w/\rho_w)} \\
\tau_w &= \text{shear stress on the wall}
\end{aligned}
$$

It can be shown that if all flow variables are normalized by some reference quantities, for instance, a_∞ for velocity, ρ_∞ for density, Lc for length, and μ_∞ for the viscosity, the formula for calculating μ_w and v_w will be modified as follows:

$$
\begin{aligned}
\mu_i &= Re_\infty \, \rho \, |w| \, \ell^2 \\
v_w &= \sqrt{Re_\infty \cdot \tau_w/\rho_w}
\end{aligned}
$$

For the case where a node has multiple length scales, say ML, the Prandtl mixing length will be calculated by taking the harmonic mean value of these length scales

$$
\frac{1}{\ell} = \sum \left(\frac{1}{\ell_i}\right), \quad i = 1 \sim ML
$$

It should be mentioned that for structured grids used in finite difference methods, the data structures applied for indicating a wall are essentially built up by the grid indices with appropriate flags, for instance, $I = 1$, or, $I = IMAX$; $J = 1$, or $J = JMAX$. This simple data structure, together with an *ad hoc* assumption that the grid system is orthogonal, allows the calculation of the length scales associated with a grid point (or node) to be made relatively easy.

In finite element methods, the calculations of ρ and $|w|$ is straightforward within an element. However, the calculation of Prandtl mixing length is very difficult, especially for adaptive unstructured grids, due to its strong dependence on the boundaries. This geometry depenence makes the calculation of length scales for a node very expensive. One reasonable method to reduce this dependence and to increase the computational efficiency is to build up a data structure that can be readily used to calculate the length scales for a given point. By having the data structure built up, the calculation of the Prandtl mixing length can be performed in a manner that is consistent with ρ and $|w|$ in an efficient manner.

Requirements in Designing a Data Structure for Implementing an Inner Layer Turbulence Model

In designing a data structure there are several requirements which should be kept in mind:

1. Efficient (ready to use)

   Once the data structure is built up, the length scales for a given point (nodal points or integration points) should be readily decoded from the data structure.

2. Economic (minimum storage)

   The storage should be kept at a minimum and be flexible (dynamic allocation).

3. Geometry independent (complex geometry)

   The data structure should be designed such that it is independent of the complexity of the geometry.

4. Modularized (easy to couple with various flow algorithms)

   The data structure should be designed such that it could be easily coupled with any flow algorithms.

5. Extendable, reusable (ready to be used with a two-equation model)

   The data structure should be designed such that it can be used with other turbulence models in which length scale is one of the key parameters for calculating turbulent eddy viscosity.

6. General (multiple length scales)

   The data structure should be designed so that it allows for variable length scales for each node.

7. Grid-structure dependent

   The data structure should be dependent on the global grid structure only. This implies that the data structure should be rebuilt only when the global grid structure has been changed, for instance, after a grid adaptation.

8. Readable

   The arrays used in the data structure should be easy to read for engineers and the designer.

9. Integrity, compactness, etc.

Criteria in Building Up the Data Structure

It is important to set up geometrical criterion in building up the data structure for implementing the inner layer turbulence model. Without using these criteria, erroneous length scales and eddy viscosities may be determined. Presently, the following criteria are used.

- A length scale for a node is a valid projection and/or the minimum distance to the viscous boundary (Fig. 7.1).
- A valid length scale should not break the boundary of the computational domain (Fig. 7.2).
- The maximum allowable number of length scales per node is limited to four (4).
- Each node has at least one length scale associated with a viscous boundary (Fig. 7.3).
- If no valid projection can be found, the minimum distance to a wall will be taken as the length scale associated with a viscous boundary.
- Length scales for each node are selected from all potential length scales that satisfy the above criteria starting from the minimum value.

Design of a Data Structure for Implementing an Inner Layer Model

With the above functional specifications and criteria in mind, we have designed a data structure for efficiently implementing an inner layer turbulence model.

The first set of data is utilized to store information about the no-slip boundaries and the solutions on these boundaries. All the viscous boundaries are stored in a discrete form, that is, a boundary is composed by many line segments (this is true because bilinear elements are used in our finite element code). Each line segment will be referred to as a *viscous panel.* Each viscous panel can be identified by four integers, the element number it belongs to, the side number, and the node numbers (with respect to total number of viscous nodes) of its two end points. The reason for storing the node numbers of the end points for each viscous panel is that they could be readily used for calculating the wall function (y^+) for each interior node by using interpolation.

The second set of data is utilized to store information that can be readily used to calculate the Prandtl mixing length for each node during the solution procedure. This set of data consists of a pointer, the number of length scales, and the values of the length scales

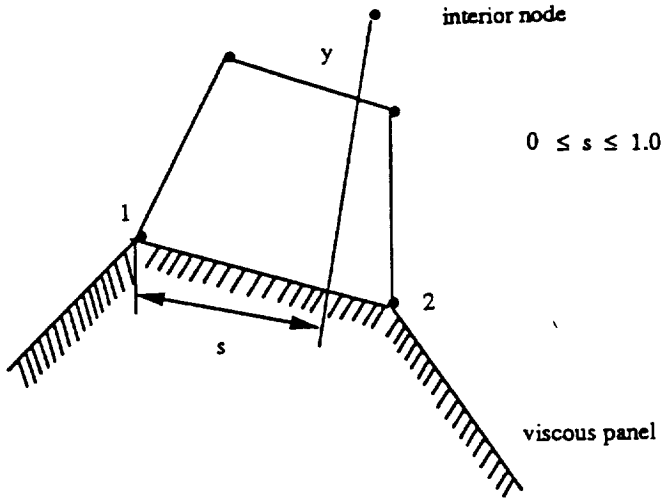


Figure 7.1: A valid projection of an interior node with respect to a viscous boundary.

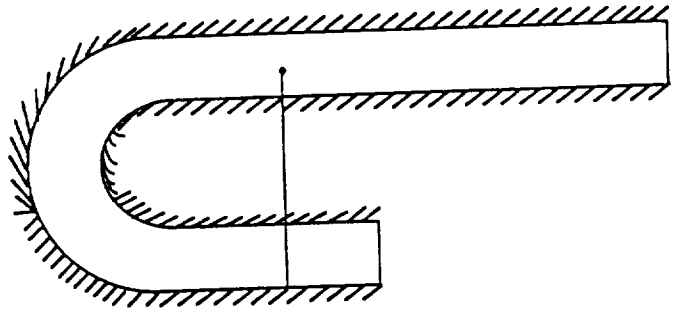


Figure 7.2: A valid projection breaks the boundary.

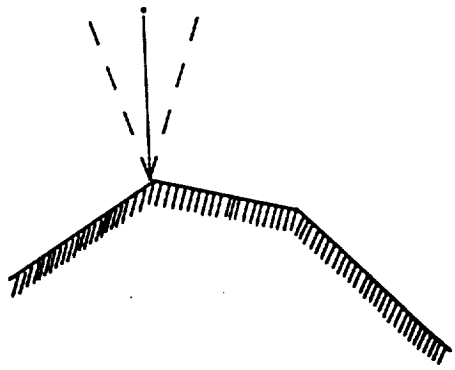


Figure 7.3: The minimum distance is selected as the length scale.

associated with a node. For each length scale, the associated viscous panel number and the local coordinate of the projection are also stored. A pointer array was designed to take care of variable length scales, to efficiently allocate the storage and to direct access to the data to be used to calculate the length scale.

Details of the data structure are described as follows:

```
   COMMON /TURBZR/  RHOWALL(2*MXBCD),TAUWALL(2*MXBCD),
.                   CMUWALL(2*MXBCD),RTZDAT(2,2*MAXND),
.                   TLENX(MAXND)
   COMMON /TURBZI/  NVISPAN,NVISNOD,NODVPAN(4,2*MXBCD),
.                   NMVISP(MAXND),ITZDAT(2*MAXND),
.                   ITZPTR(MAXND)
```

DEFINITIONS OF ARRAYS

New Flow Property:

```
TLENX  :  TURBULENT LENGTH SCALE FOR EACH NODE
```

Wall Variables:

```
RHOWALL  :  DENSITY FOR EACH NODE ON VISCOUS BOUNDARY
TAUWALL  :  WALL SHEAR STRESS FOR EACH NODE ON VISCOUS BOUNDARY
CMUWALL  :  LAMINAR VISCOSITY FOR EACH NODE ON VISCOUS BOUNDARY
```

Global Arrays:

```
NMVISP  :  NUMBER OF VISCOUS PANELS ASSOCIATED WITH A NODE
           (NUMBER OF LENGTH SCALES FOR A NODE),
ITZPTR  :  THE POINTER FOR EACH NODE (IN ARRAYS RTZDAT AND ITZDAT)
RTZDAT  :  VALUE OF Y-NORMAL AND LOCAL PARAMETRIC COORDINATES
           ON A VISCOUS PANEL FOR EACH NODE,
ITZDAT  :  LIST OF VISCOUS PANELS ASSOCIATED WITH A NODE
```

Information About the Viscous Panel:

```
NVISPAN  :  TOTAL NUMBER OF VISCOUS PANELS = NUMELD+NUMELE
NVISNOD  :  TOTAL NUMBER OF VISCOUS NODES = NUMBCD+NUMBCE
NODVPAN  :  INFORMATION TO BE DECODED TO IDENTIFY THE NODES ON
            EACH VISCOUS PANEL,
            1 :  ELEMENT NUMBER,     2 :  SIDE NUMBER
            3 :  FIRST NODE NUMBER,  4 :  SECOND NODE NUMBER
```

Summary of Storage:

1. A number of 7*MAXND words are required for storing information associated with the length scales for each node (assume averaged two length scales for each node).

2. A number of 3*MXVISP words are required for storing solutions on the solid (no-slip) walls for efficiently interpolating solutions.

Technical Comments

For a grid system with MAXND=2500, it requires 140 kbytes of storage (assuming an average of two length scales for each node). This storage requirement is comparable to the storage required by the data structure designed by P. Rostand [30]: 90 kbytes for the same grid size and a "single" length scale for each node. However, it must be noted that Rostand's data structure was designed for implementing both an inner layer and outer layer turbulence model and for simple geometries only (convex geometry was assumed). In addition, after a closer examination, one finds that there are some hidden storage requirements in Rostand's calculations (for instance, solutions on the normals to be used for interpolation) not taken into account. This needed data requires additional storage and extra computational time if Rostand's data structure were to be generalized to handle more complicated geometries. We estimate that an additional 140 kbytes of storage may be required for our data structures if the outer layer turbulence model is to be implemented.

Numerical Implementation

The major tasks involved in the numerical implementation of the inner layer turbulence model are basically composed of two parts: 1) how to build up the data structure, and 2) how to use the data structure to calculate the eddy viscosity. Flowchart 1 (Fig. 7.4) shows how the data structure is constructed in a step-by-step fashion. Flowchart 2 (Fig. 7.5) presents a global view of how the inner layer turbulence model can be coupled with other flow algorithms with just a few extra subroutine calls. Once the data structure is built

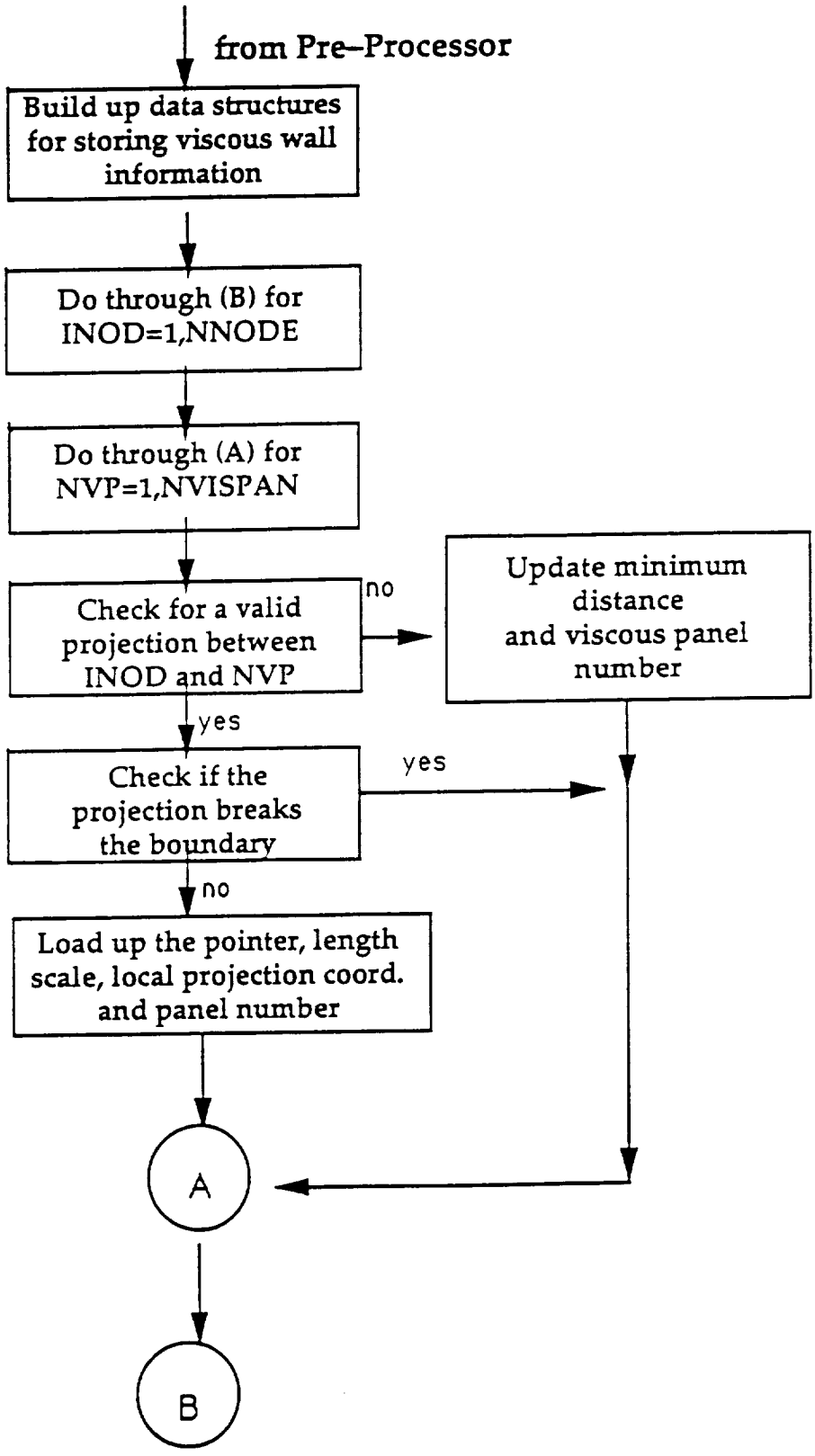


Figure 7.4: Flowchart for constructing the data structures for inner layer turbulence model.

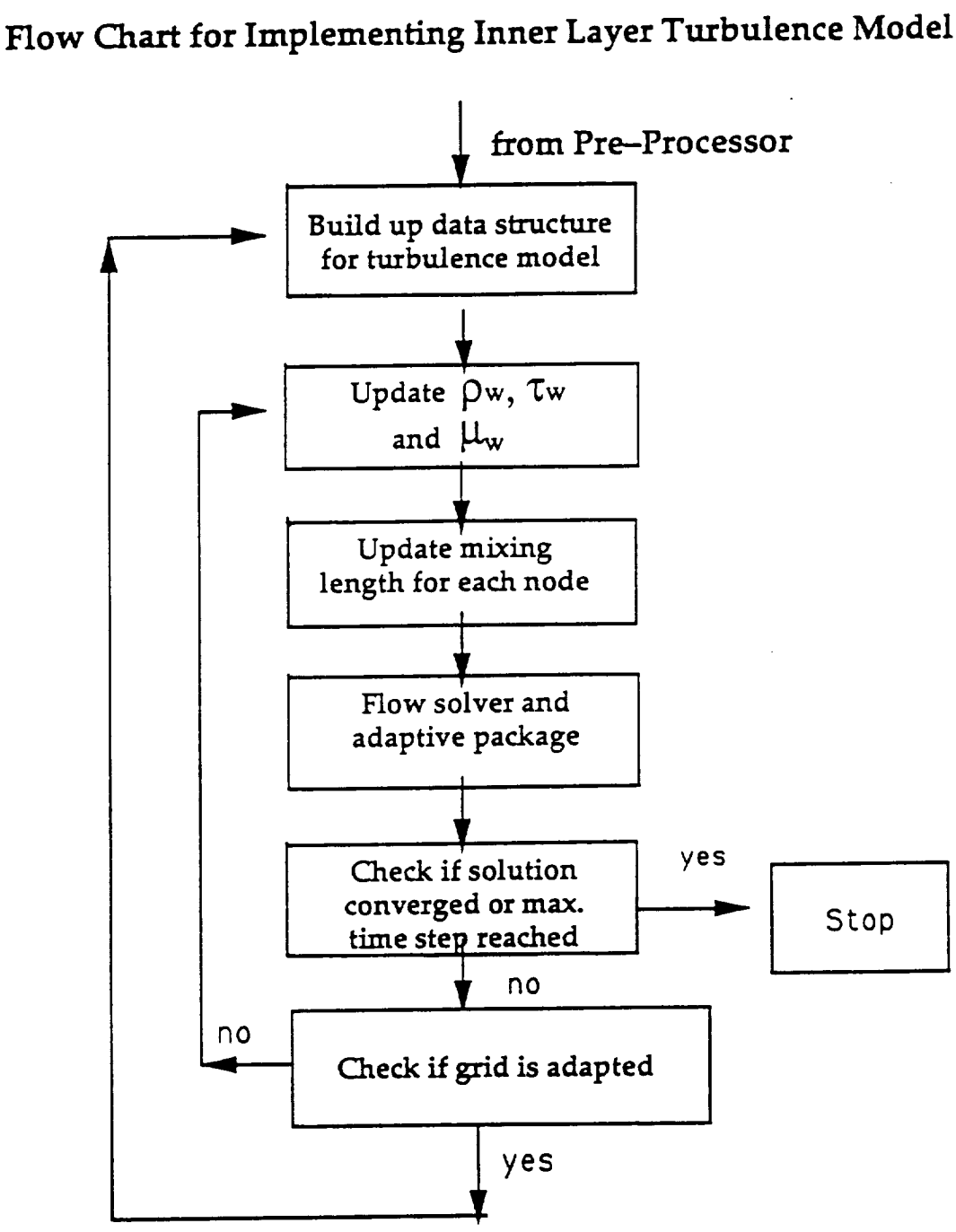


Figure 7.5: Flowchart for implementing inner layer turbulence model.

up, the calculation of the eddy viscosity for a given point (usually the integration point) is straightforward within the flow solver.

8 Numerical Examples

8.1 Supersonic Nozzle With Small Throat Radius of Curvature

A 45°–15° convergent/divergent conical nozzle was used in this test case (see reference [31] for detailed nozzle specifications). This geometry is characterized by its rapid contraction in the convergent part, sharp wall curvature at the nozzle throat and near parallel at the nozzle exit. The experimental work for this type of nozzle flow has been performed by Cuffel, et al. [32] and numerical results have been reported by Serra [31].

An initial grid of 61 × 21 nodes was used to solve this problem. Flow was initialized by using the analytic solution concluded from a quasi–one–dimensional isentropic flow [33]. The nozzle was assumed to connect to a reservoir such that the flow condition at the nozzle inlet could be treated as a uniform subsonic inflow. This inlet flow condition remained constant during the solution process. At the nozzle exit, a supersonic outflow condition was specified. After 400 time steps (with a minimum cost option and CFLBOUND = 5.0), the flow pattern was well developed in the nozzle. The adaptive package was then switched on and the grid was adapted (every ten steps to the first level) for another 200 time steps with $\alpha = 0.50$, $\beta = 0.70$. For the next 100 time steps, the grid was adapted every 10 time steps to the second level with $\alpha = 0.40$, $\beta = 0.60$. Finally, the grid was adapted to the third level for another 100 time steps. The final adapted grid consists of 3930 elements and 3750 nodes.

As evidenced by the adapted grid shown in Fig. 8.1a, the grid is automatically refined and is well aligned with the shock pattern in the divergent part of the nozzle. This pattern is best illustrated by the Mach number contours, as shown in Fig. 8.2c. An oblique shock was triggered by a junction between the circular circular–arc throat and the divergent section. This oblique shock hit the nozzle centerline and reflected back into the flow domain. The streamline distribution over the entire nozzle passage is shown in Fig. 8.1b. The rapid expansion of the flow in the region close to the nozzle throat can be seen from a closeup view of the Mach contours in Fig. 8.2d. The comparisons of experimental data [32], numerical results from Serra [31], and our prediction for the Mach number distribution along the nozzle centerline and the nozzle wall are presented in Fig. 8.3.

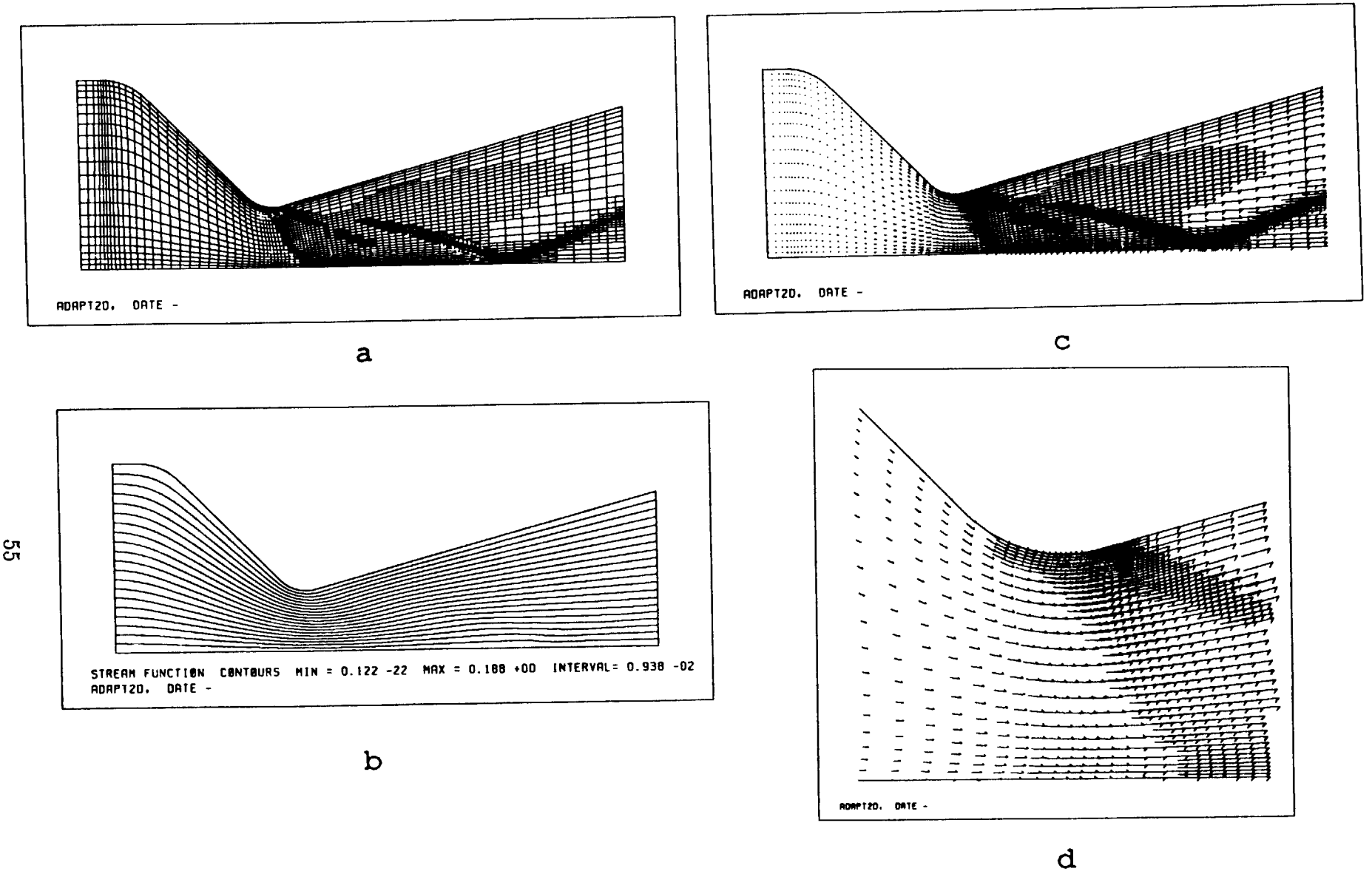


Figure 8.1: (a) Adapted grid, (b) streamlines, (c) velocity vector plot, and (d) closeup view of velocity vectors in the nozzle throat.

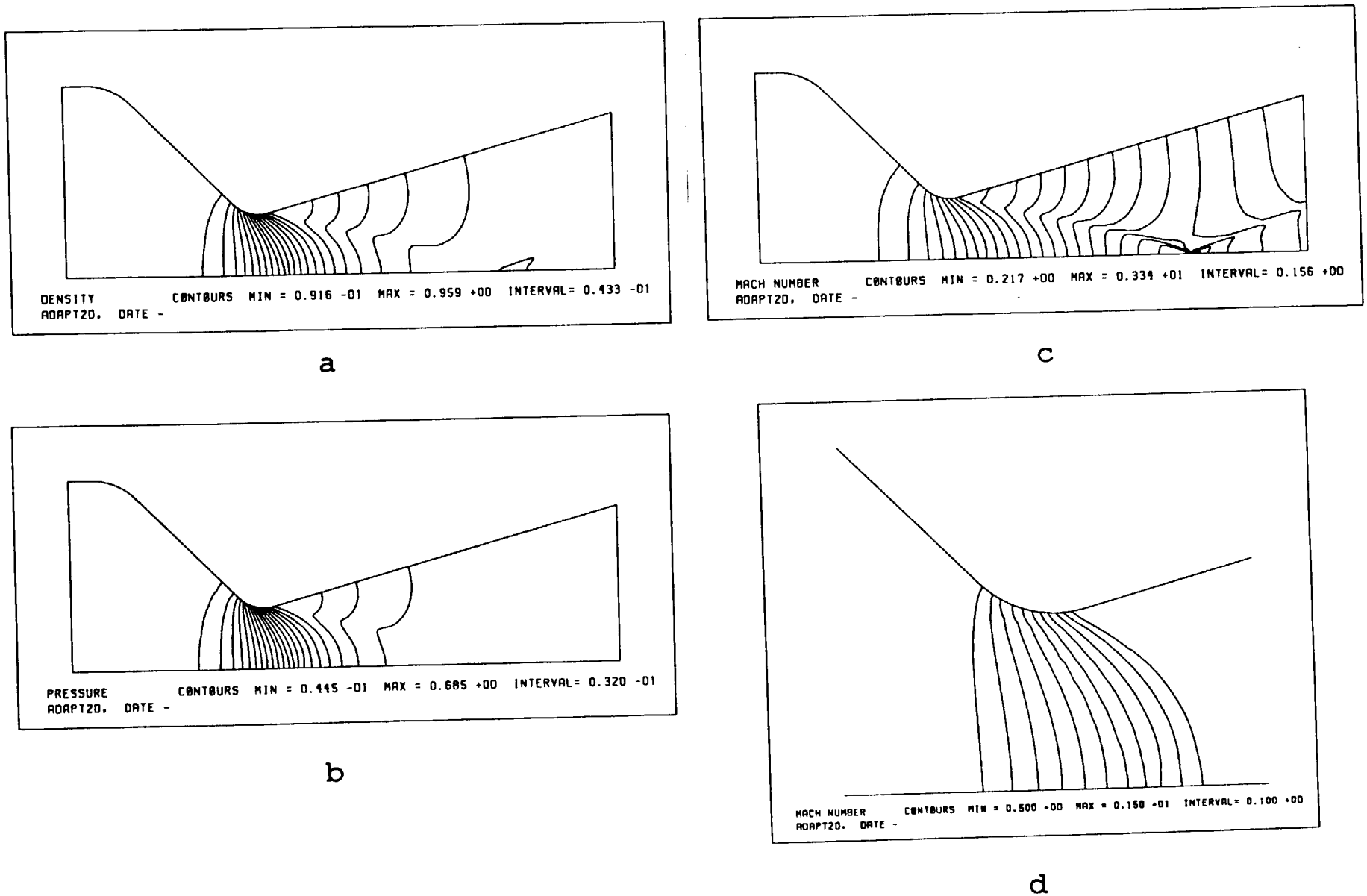


Figure 8.2: (a) Density contours, (b) pressure contours, (c) Mach contours, and (d) closeup view of Mach contours in the nozzle throat.

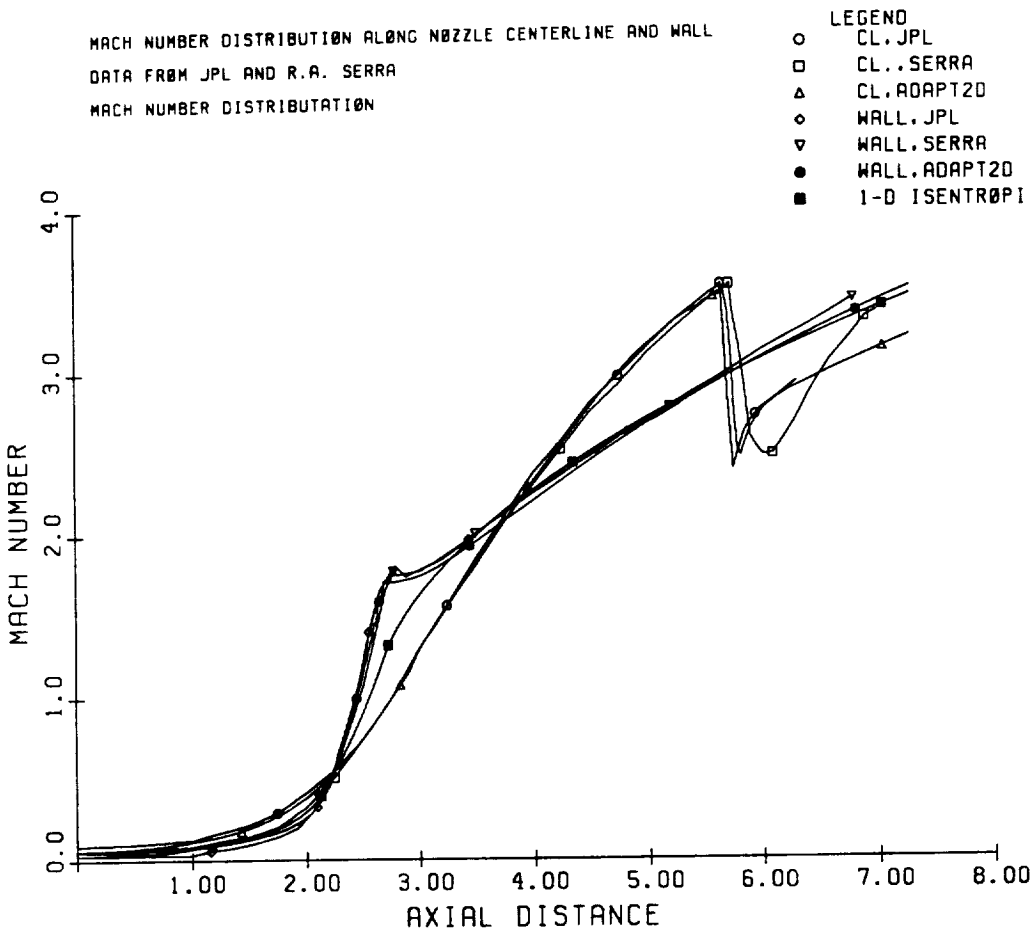


Figure 8.3: Comparison of Mach number distribution for a 45°–15° conical nozzle.

8.2 Viscous Flow Over a Sphere

Flow over a sphere has been a popular benchmark problem for validating CFD codes due to the facts that the fluid physics for this problem is well understood [34] and that many numerical results are available for comparison [34,35]. The flow conditions selected for this case are $M = 0.1$ and $Re = 100.0$. The computational domain was first discretized by using a mixed structured/unstructured grid. After the flowfield developed to a certain stage, the grid was then adapted to the first level in the region close to the solid boundary and in the separation region. The surface of the sphere was treated as a no–slip isothermal wall. The axis of symmetry was treated as a no–flow or no–penetration boundary. Characteristic boundary conditions were imposed on the rest of the artificial boundaries. No artificial dissipation was added for this test case.

Figure 8.4a shows the adapted grid which consists of 2457 elements and 2535 nodes. The recirculation region on the lee–side of the sphere can be seen clearly from the velocity vector plot, as shown in Fig. 8.4d. From the streamline plot shown in Fig. 8.4c, it can be observed that the flow separates at an approximate angle of 123° (measured from the leading edge stagnation point) and that the dividing streamline extends into the wake region with the distance $s/D = 0.8$. The vorticity distribution and pressure distribution along the surface of the sphere (as shown in Fig. 8.5) agree extremely well with the data from [34,35]. It should be mentioned that, in the solid rocket booster code, the reference velocity used to nondimensionalize the Navier-Stokes equations is the speed of sound at farfield. The characteristic length is based on the diameter of the sphere. Therefore, the vorticity and the pressure coefficient calculated by the code must be rescaled according to the following formulae:

1. for the vorticity

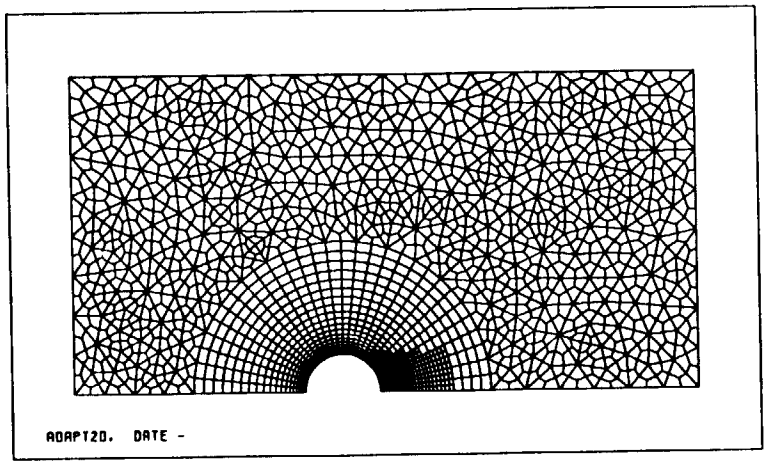
$$\omega = \frac{\omega_A}{2M_\infty}$$

   where ω_A is obtained from the code and ω is the normalized vorticity using freestream velocity.

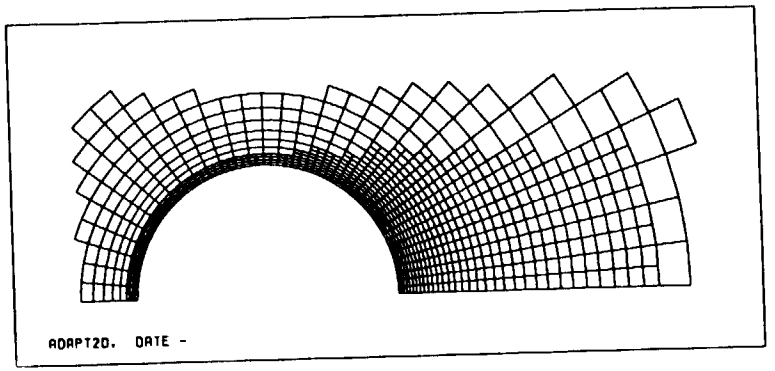
2. for the pressure coefficient

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty u_\infty^2} = \frac{2}{M_\infty^2}\left(P_A - \frac{1}{\gamma}\right)$$

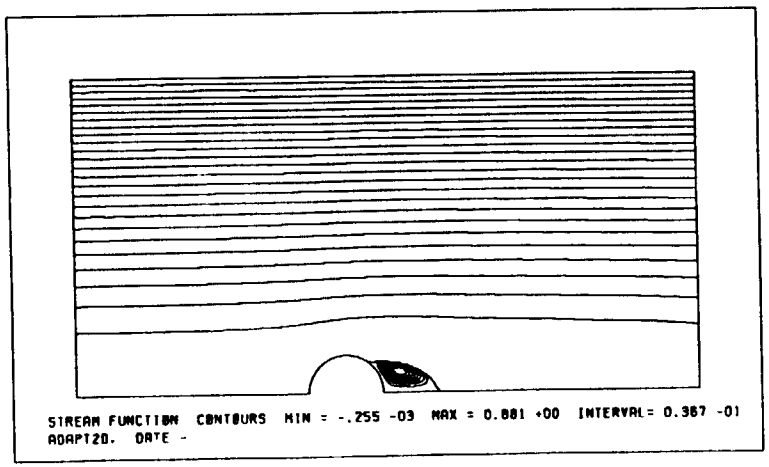
   where P_A is obtained from the $ADAPT2D^{TM}$ code.



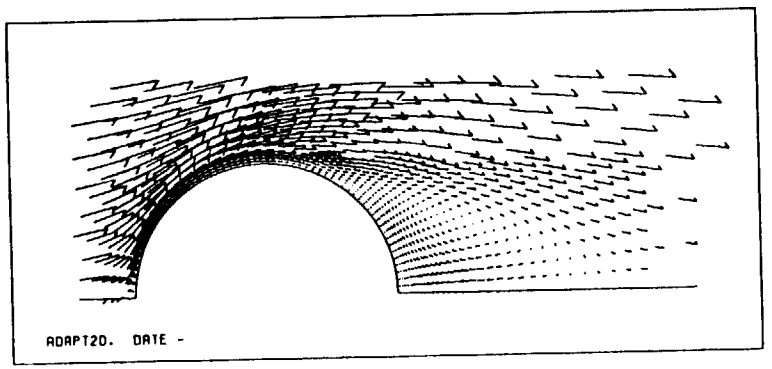
(a)



(b)

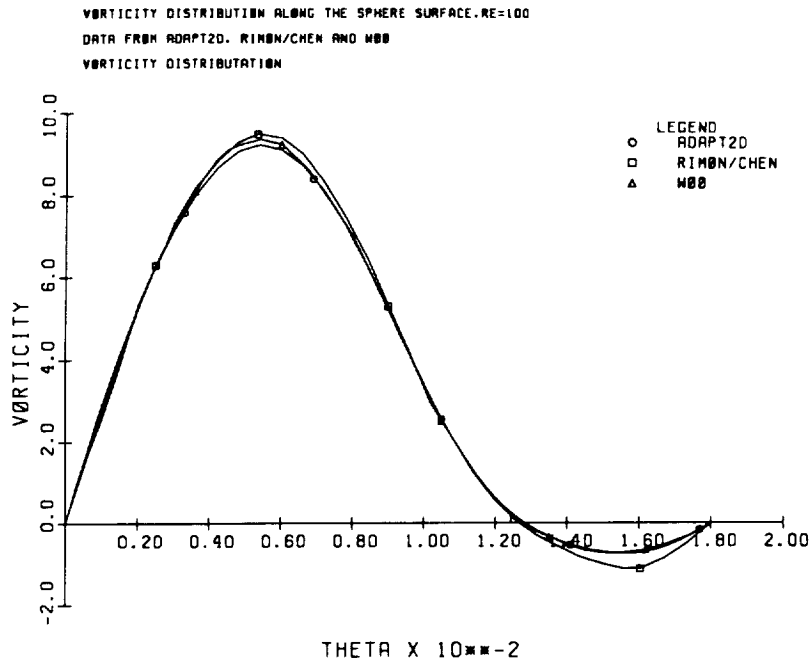


(c)

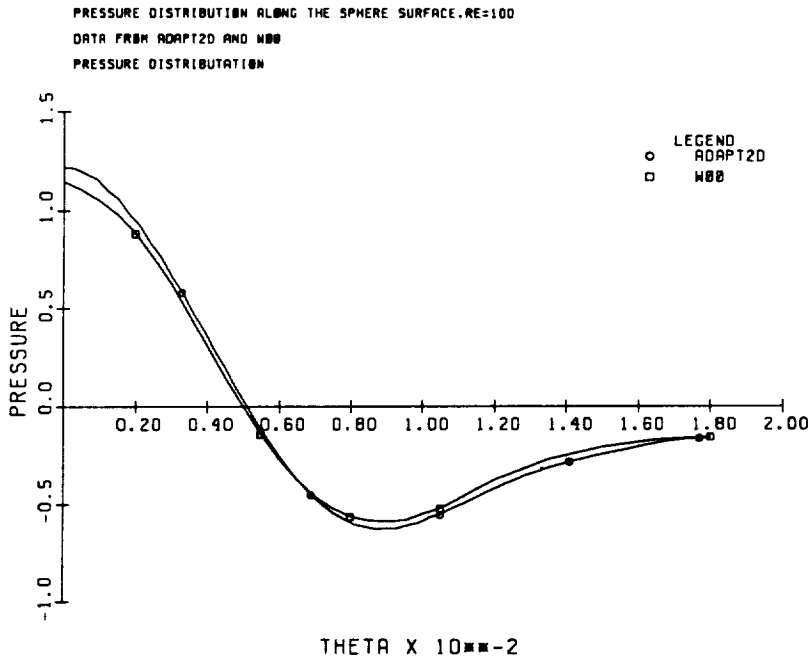


(d)

Figure 8.4: Flow over a sphere, $M = 0.1$, $Re = 100.0$. (a) Adapted structured/unstructured grid, (b) close-up view of adapted grid, (c) streamline plot shows separation bubble, (d) close-up view of velocity vector plot in separation region.



a



b

Figure 8.5: Flow over a sphere, $M = 0.1$, $Re = 100$. (a) vorticity distribution on the sphere surface, and (b) pressure distribution on the sphere surface.

8.3 Simulation of Vortex Shedding Due to Motor Inhibitor

The purpose of this test case is to verify the axisymmetric capability of the code. The problem of vortex shedding due to a motor inhibitor protruding from the wall to the port flowfield studied by Majumdar, et al. [36] will be utilized as the benchmark problem.

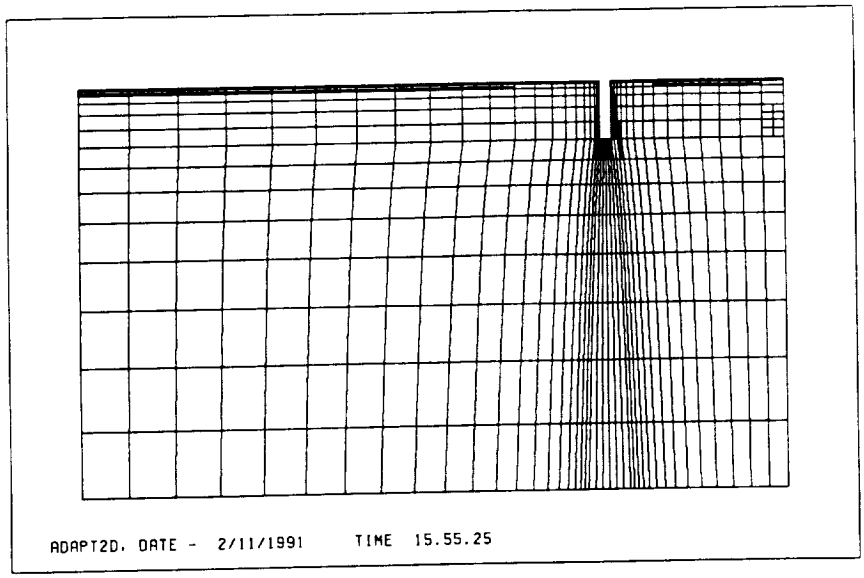
Due to the nature of axisymmetry, an axial/radial section of the SRM was taken as the computational domain. For the sake of fair comparison, the initial mesh size used for solving this problem is exactly the same as it was used in [36]. The flow conditions employed in the simulation are $M = 0.09897$, $Re = 10^5$. No artificial dissipation was added for this case. Flow was initialized by using one–seventh power law velocity distribution. This type of velocity distribution was also used to specify the inflow condition. At the outflow, a pressure boundary condition was imposed.

The grid was adapted to the first level during the solution process. The final grid consists of 722 elements and 772 nodes. As seen in Fig. 8.6a, grids were automatically refined along the no–slip boundary to resolve the viscous boundary layer. Strong circulation due to the inhibitor can be seen clearly from the velocity vector plot shown in Fig. 8.6b. The streamline plot indicates a large separation region formed in the lee–side of the inhibitor. The pressure distribution is displayed in Fig. 8.7a. These results qualitatively agree well with the results obtained by Majumdar, et al. [36] as exhibited in Fig. 8.8.

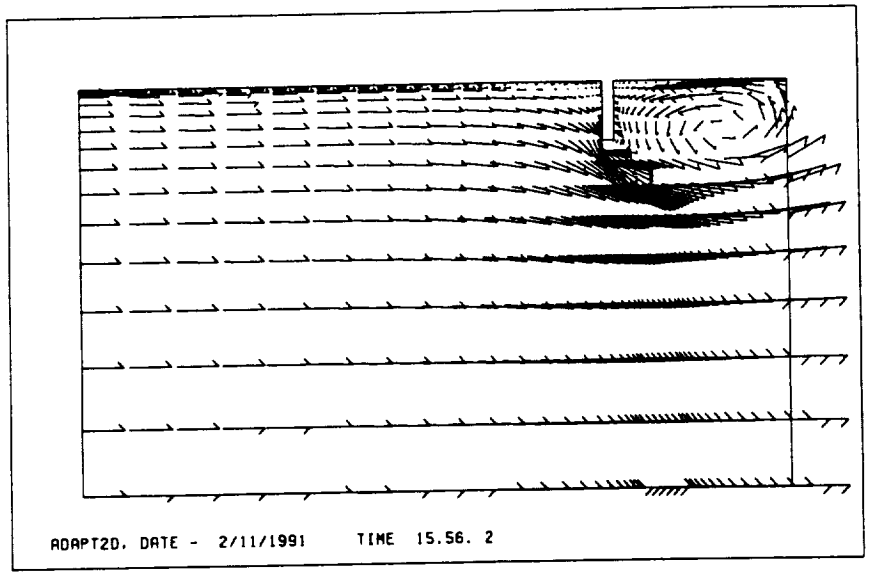
8.4 Internal Flow in the Turnaround Duct of Space Shuttle Main Engine

Due to the highly restricted space, the most important design specifications for the Space Shuttle Main Engine (SSME) are minimal weight and size. This requirement results in the complicated design of engine components which are usually combined with structural and geometrical complexities. A typical example is the 180° bend turnaround duct (TAD). The strong curvature of flow passage in the TAD will cause high levels of turbulence, flow unsteadiness and flow separation, etc. The ability to model turbulent flows is therefore important for any CFD code in order to predict internal flow in the turnaround duct. However, as pointed out by Monson, et al. [29], turbulence closure models have been developed and optimized for external flows. The type of model needed and its importance for internal flows with strong curvature still remains to be established. The purpose of this test case is to verify the accuracy of the simple algebraic turbulence model that has been implemented within the code.

Specification of the computational domain is given in [29]. Two test cases with $Re = 10^5$ and $Re = 10^6$ have been performed. The initial grids for both cases are shown in Fig. 8.9. Flow was initialized by using the one–seventh power law velocity distribution [37]. The

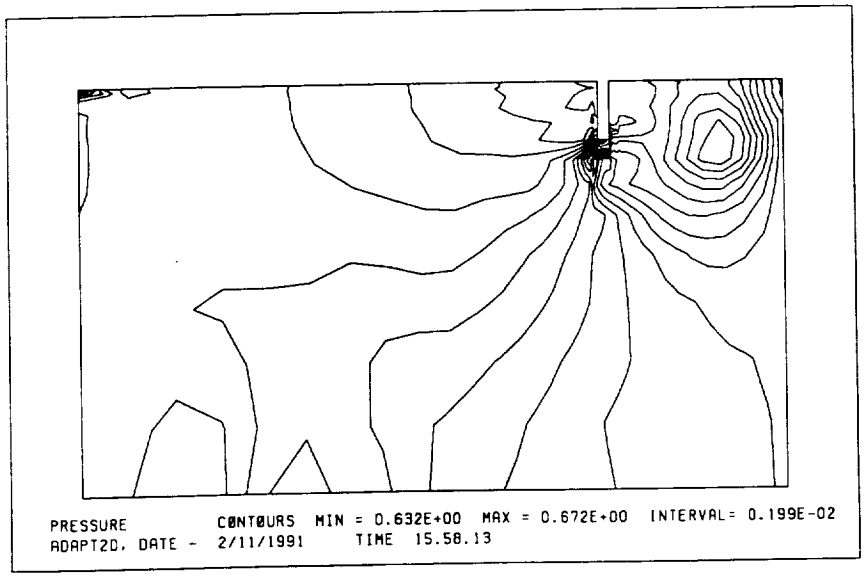


a

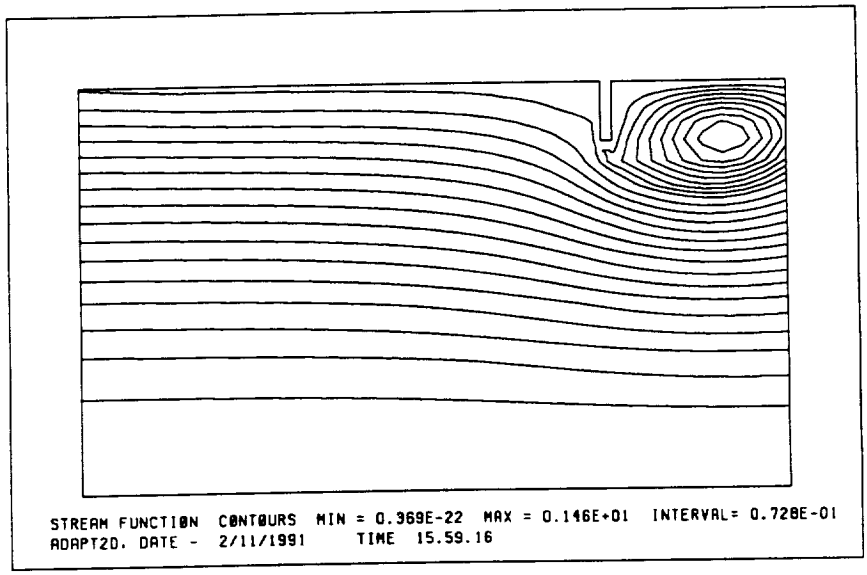


b

Figure 8.6: Simulation of vortex shedding due to motor inhibitor. (a) adapted grid, and (b) velocity distribution.



a



b

Figure 8.7: Simulation of vortex shedding due to motor inhibitor. (a) pressure contours, and (b) streamlines.

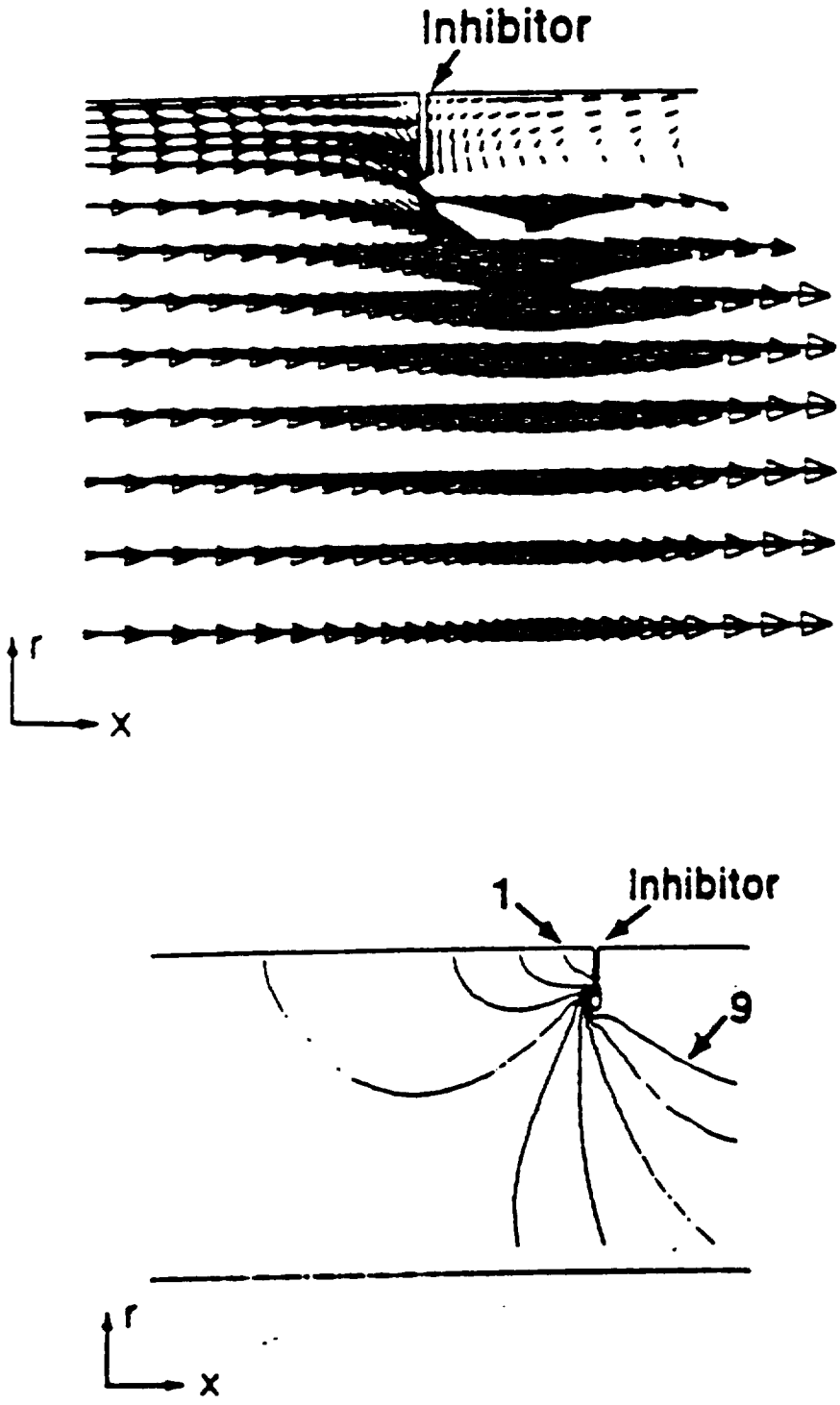


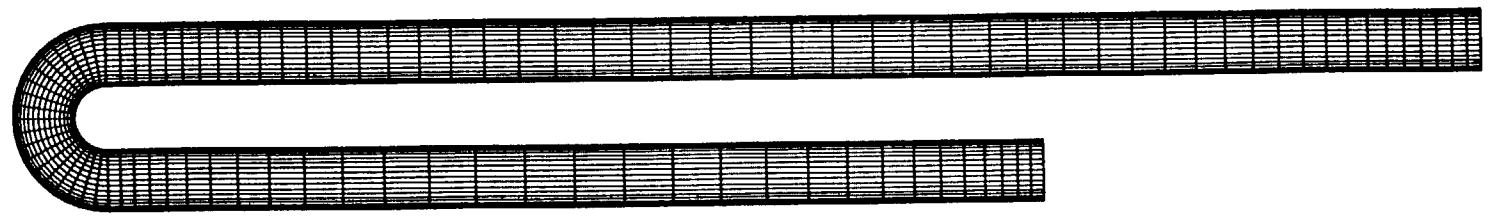
Figure 8.8: Simulation of vortex shedding to to motor inhibitor [36]. (a) velocity distribution, and (b) pressure contours.

maximum Mach number in this velocity profile is $M = 0.1$. This amounts to an averaged uniform flow at the inlet with $M = 0.0875$. This velocity is used to normalize the longitudinal velocity distribution. At the inlet of the TAD, the boundary condition was specified by using the one–seventh power law velocity distribution. The pressure outflow boundary condition was imposed at the exit of TAD. A fully implicit scheme was used to solve this problem. To let the flow develop smoothly, the CFL number was gradually increased from 1.0 to 50.0. After 400 time steps, the grid was adapted to the first level in the regions where active flow events are likely to occur, see Figs. 8.10, 8.11. The specified convergence tolerance was aligned within 600 time steps (the convergence tolerance is in the order of 10^{-6}). The pressure contours, vorticity contours, velocity distribution and streamlines are shown in Figs. 8.12, 8.13. The results shown here agree qualitatively very well with the numerical results predicted by Chen by using an extended k-ε model as shown in Fig. 8.14. It is interesting to note that in the case of $Re = 10^5$, a small secondary separation bubble was predicted by our code, which is consistent with the observation in the experiment performed by Sandborn (as shown in Fig. 8.15). The comparisons with the data predicted by other CFD codes [29] for the longitudinal velocity distributions at several axial locations are shown in Figs. 8.16–8.21. Good agreement can be observed for those locations far away from the separation bubble while some discrepancy can be seen in the regions close to the separation bubble. Further numerical studies are required to investigate this difference.

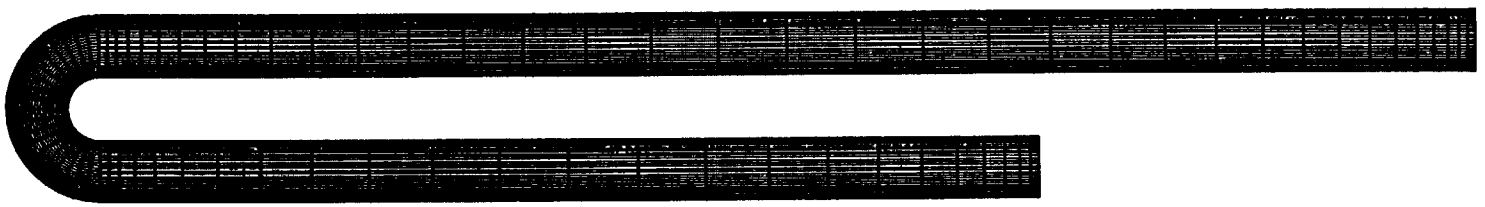
8.5 Porous Cylinder with Nozzle (Planar Case)

The internal flow in a nozzle is usually driven by the mass injection from the surface of the solid propellant and the positive pressure gradient between the rocket chamber and the environment. The purpose of this test case is to apply the porous wall boundary condition and the pressure outflow boundary condition for simulating the internal flowfield in a cylindrical–port cold–flow model. This cylindrical–port model is connected to a nozzle which was constructed using piecewise analytic functions. This type of problem has been studied by Sabnis, et al. [38] and will be resolved here.

The initial grid consists of 50×20 elements in the cylindrical part and 30×20 elements in the nozzle section. The flow conditions employed for this case are the mass injection rate $= 0.0018$ and Reynolds number $= 8.0 \times 10^5$. No artificial dissipation was added and laminar flow was assumed. Flow was initialized as a quiescent condition over the entire computational domain. The head end of the cylinder and nozzle wall were treated as a no–slip isothermal boundary. The axis of the cylinder was treated as a no–flow boundary. In order to drive the flow in the nozzle smoothly, the pressure at the outflow boundary was first set to a value slightly below its corresponding quasi–one–dimensional isentropic pressure. During the solution process, this value was gradually decreased until supersonic flow was



a



b

Figure 8.9: Initial grids used for simulating internal flow in the TAD. (a) $Re = 10^5$, (b) $Re = 10^6$.

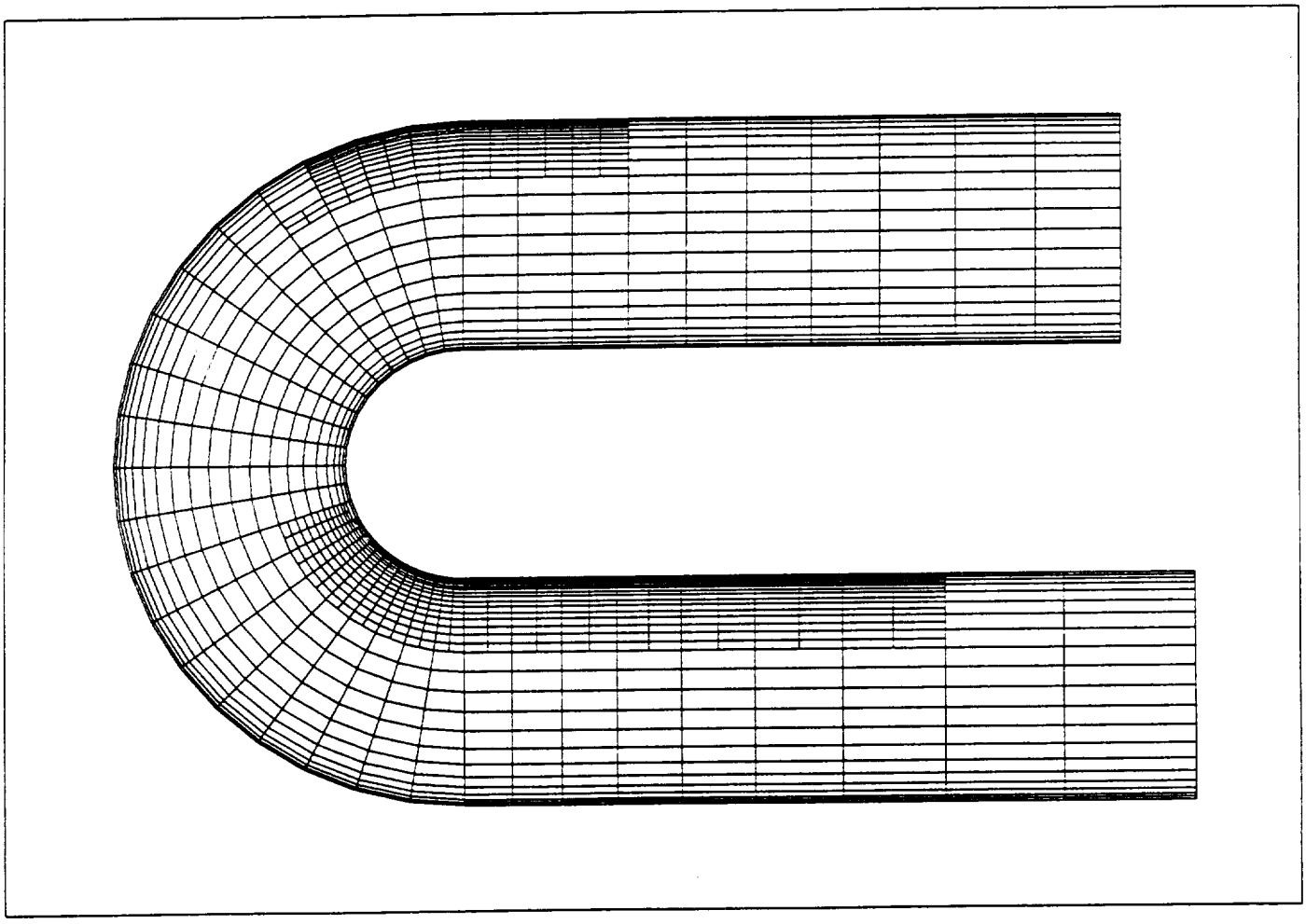


Figure 8.10: Adapted grid, $Re = 10^5$.

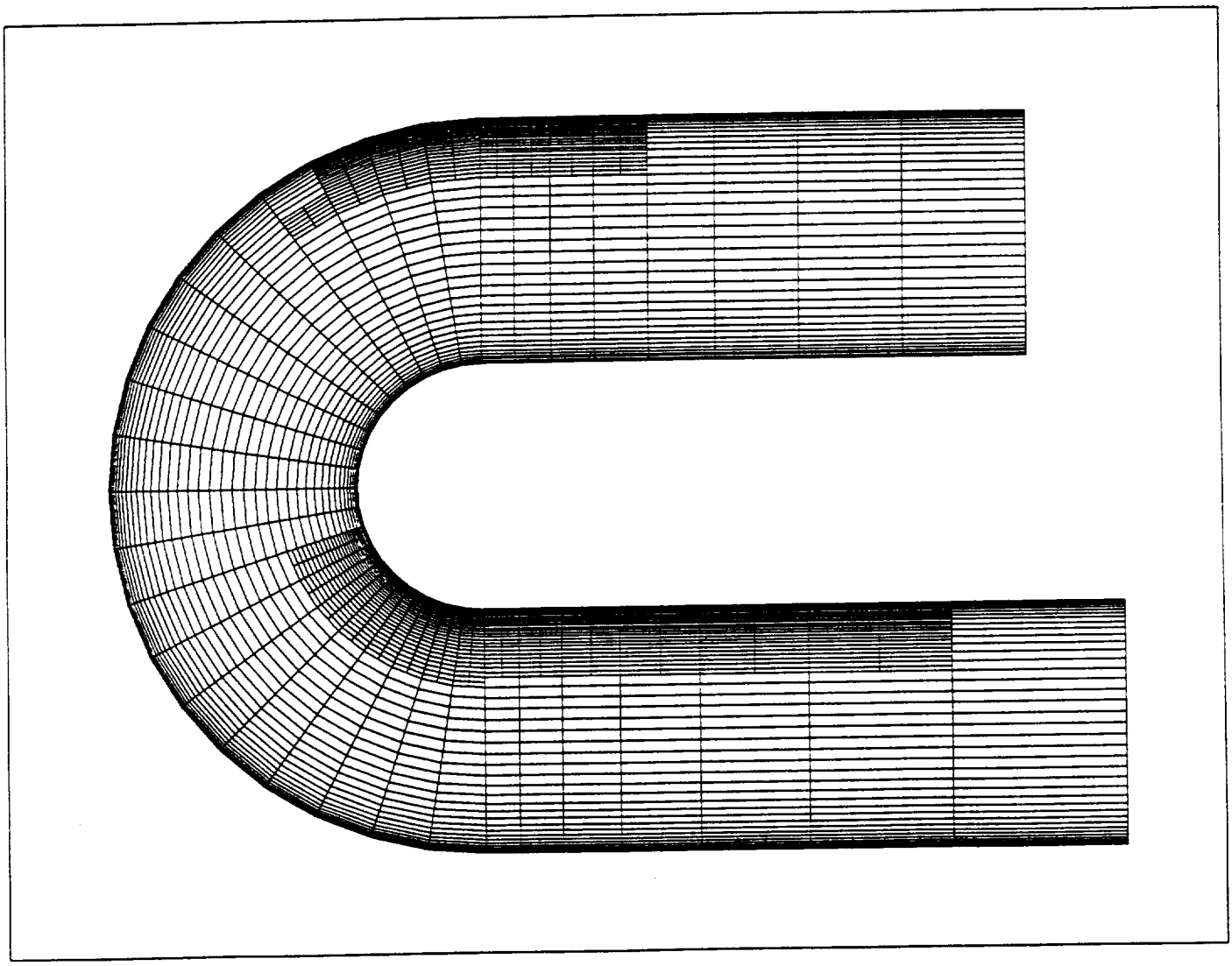
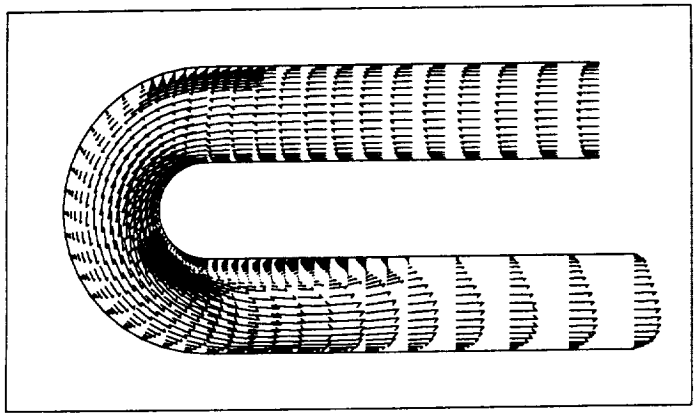
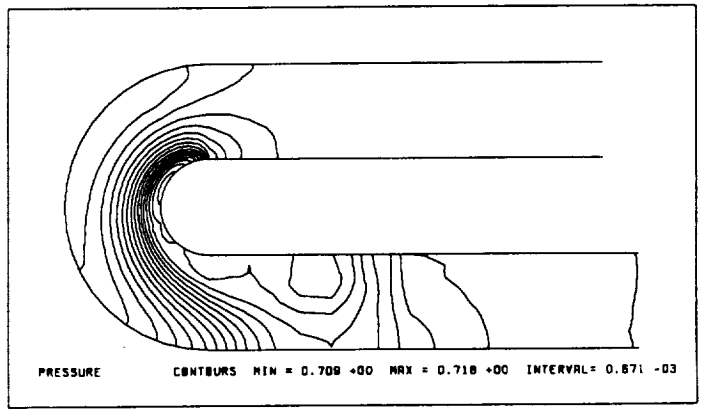


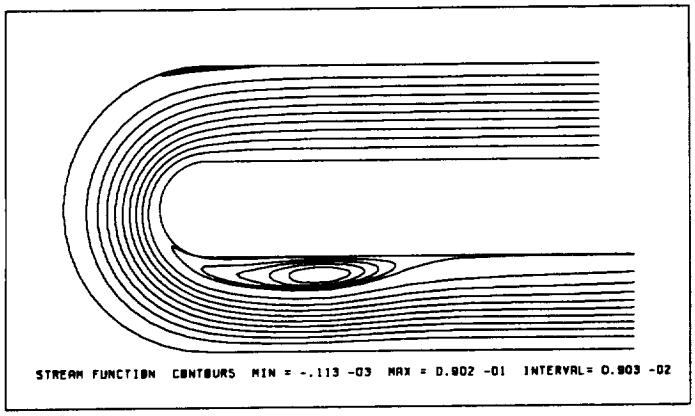
Figure 8.11: Adapted grid, $Re = 10^6$.



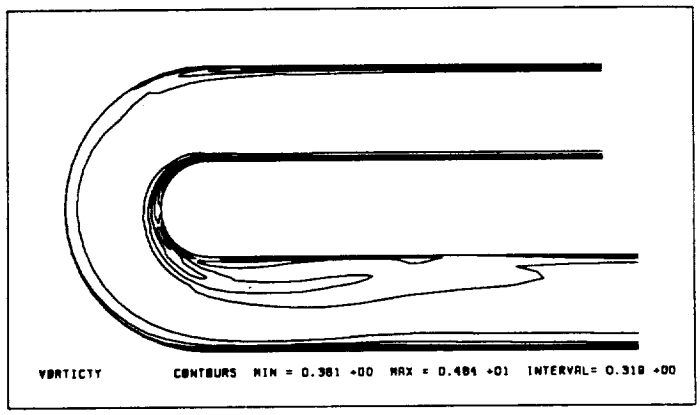
a



c

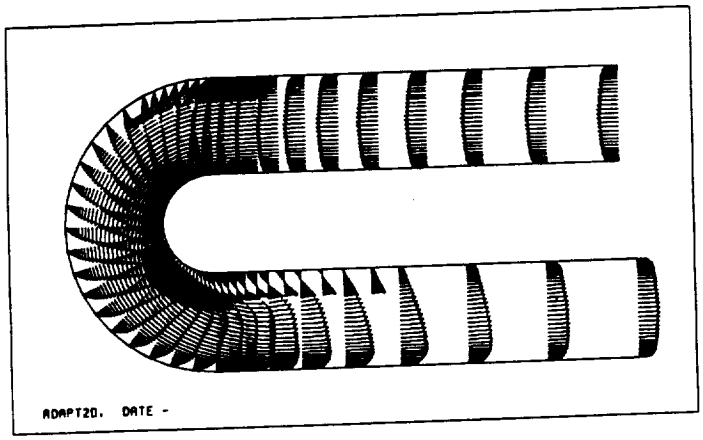


b

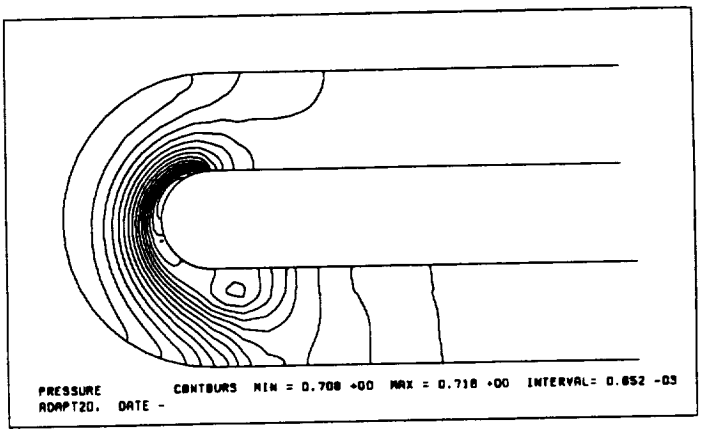


d

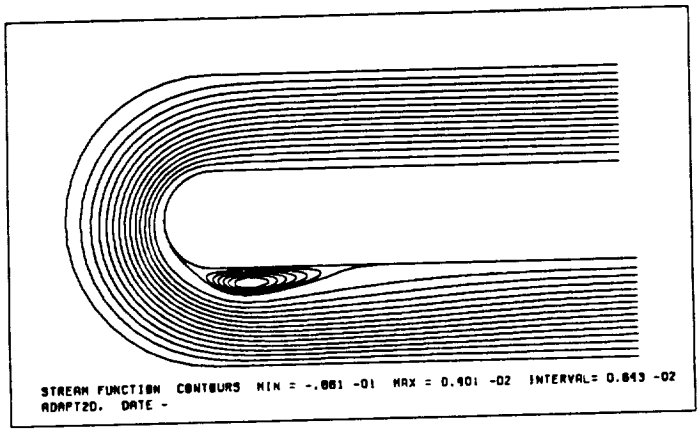
Figure 8.12: Internal flow in the TAD, $Re = 10^5$. (a) velocity distribution, (b) streamlines, (c) pressure contours, and (d) vorticity contours.



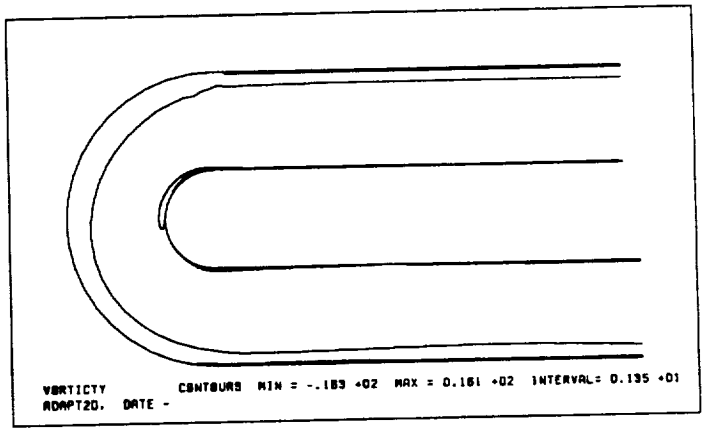
a



c

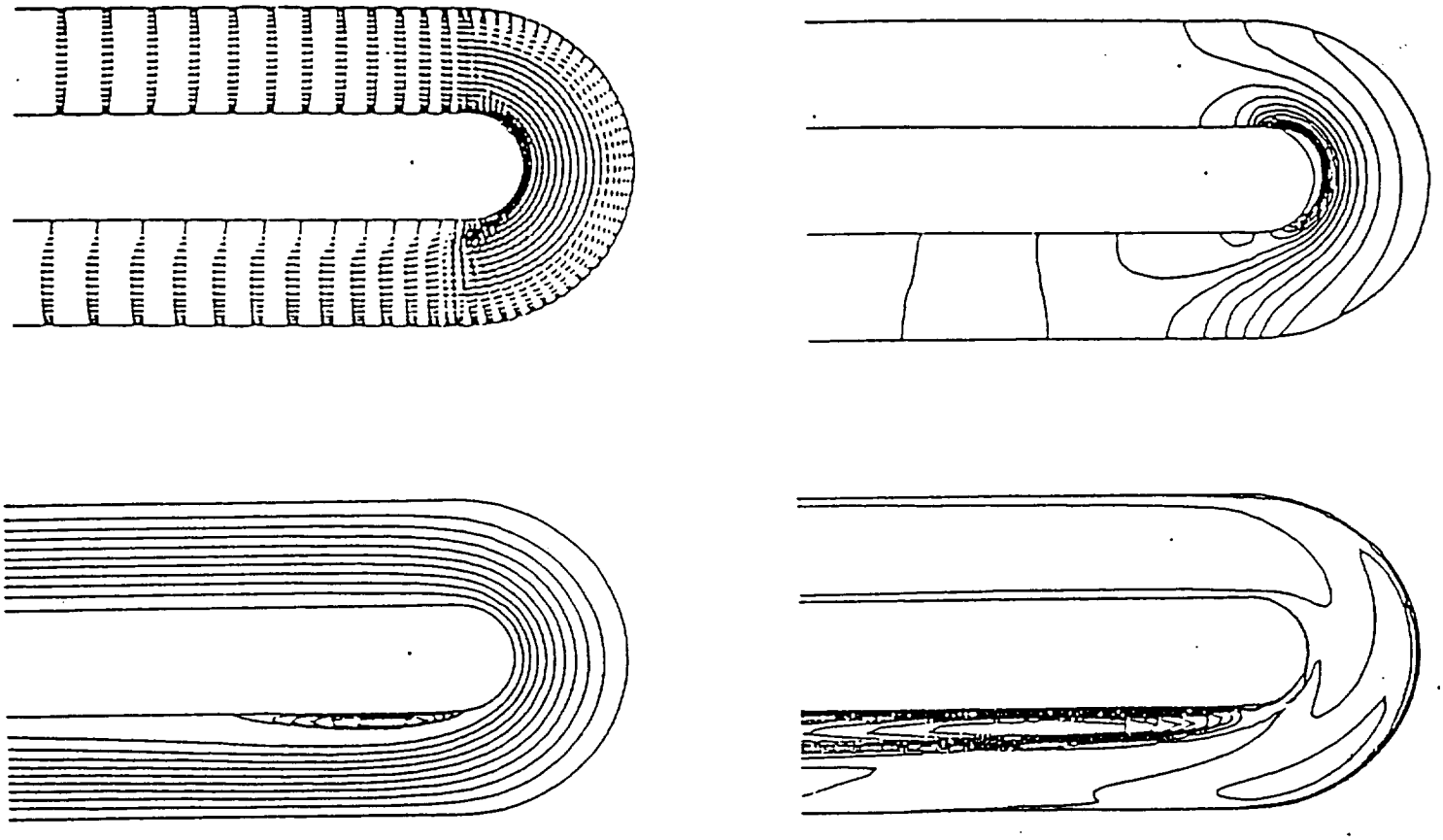


b



d

Figure 8.13: Internal flow in the TAD, $Re = 10^6$. a) velocity distribution, (b) streamlines, (c) pressure contours, and (d) vorticity contours.



- $Re = 10^5$
- Extended k-ε Model
- 81 x 21 Grid

From Y. S. Chen '87

Figure 8.14: Internal flow in the TAD (Y. S. Chen), $Re = 10^5$, k-ε model. (a) velocity distribution, (b) streamlines, (c) pressure contours, and (d) vorticity contours.

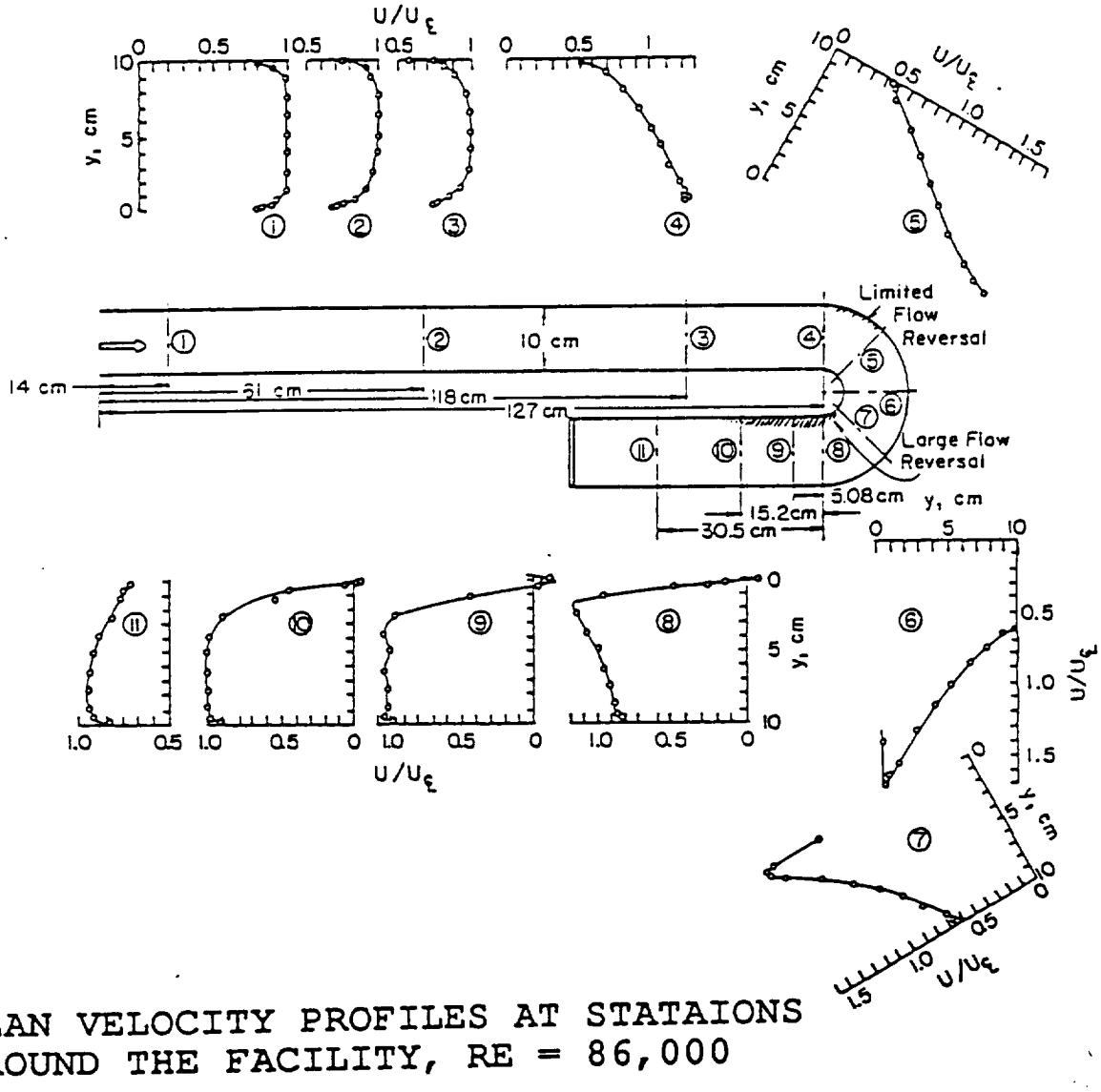
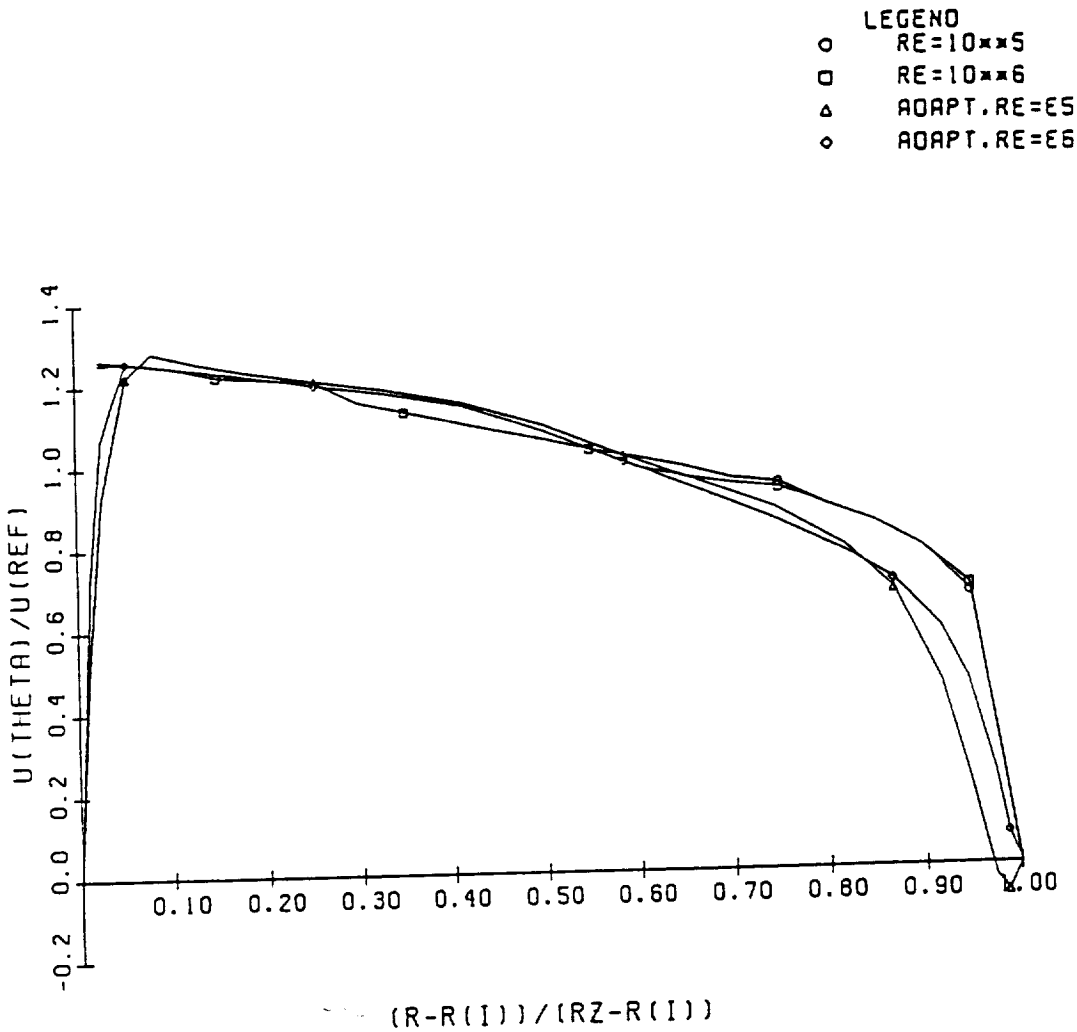
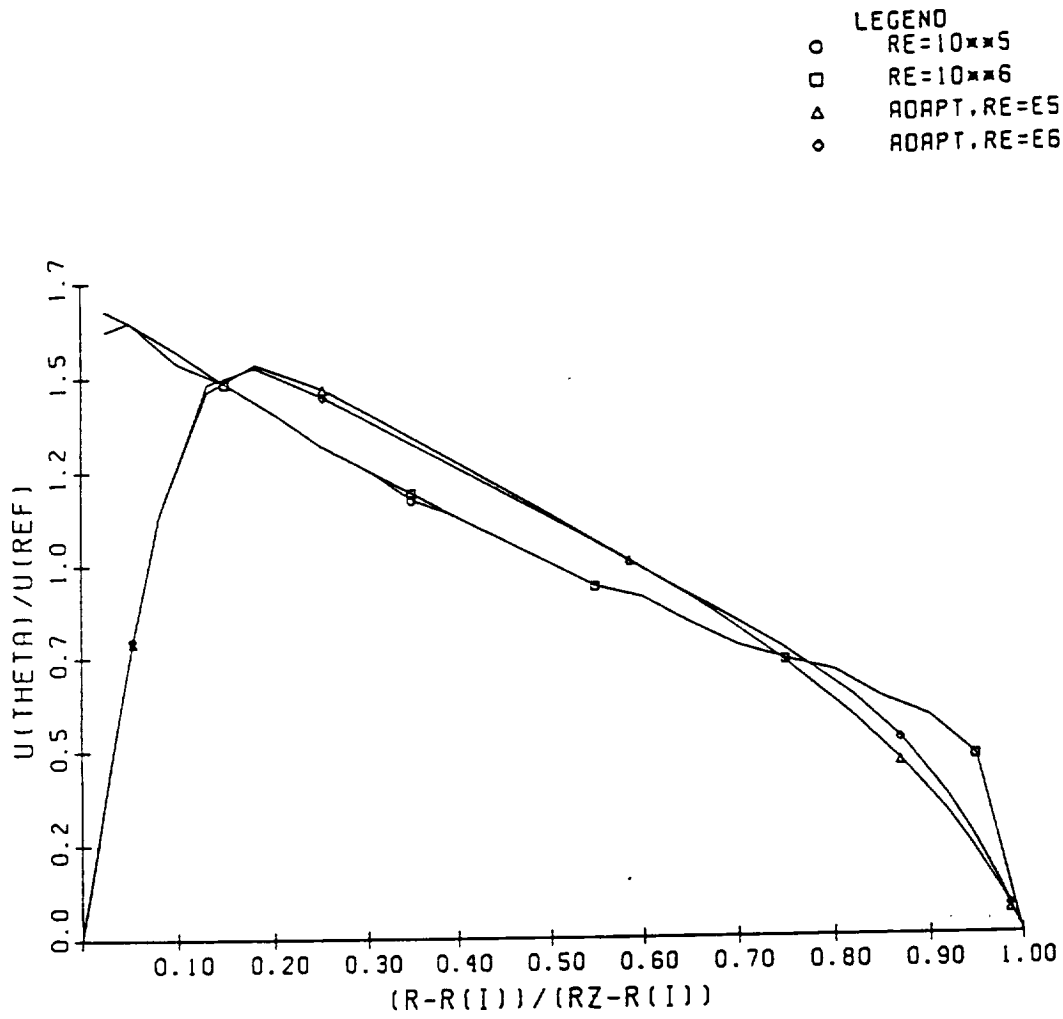


Figure 8.15: Internal flow in the TAD (Sandborn), $Re = 86,000.0$, mean velocity profiles at different axial locations.



LOGITUDINAL VELOCITY DISTRIBUTION
AT THETA = 0.0

Figure 8.16: Normalized longitudinal velocity distribution at $\theta = 0.0$ deg.



LOGITUDINAL VELOCITY DISTRIBUTION
AT THETA = 90.0

Figure 8.17: Normalized longitudinal velocity distribution at $\theta = 90.0$ deg.

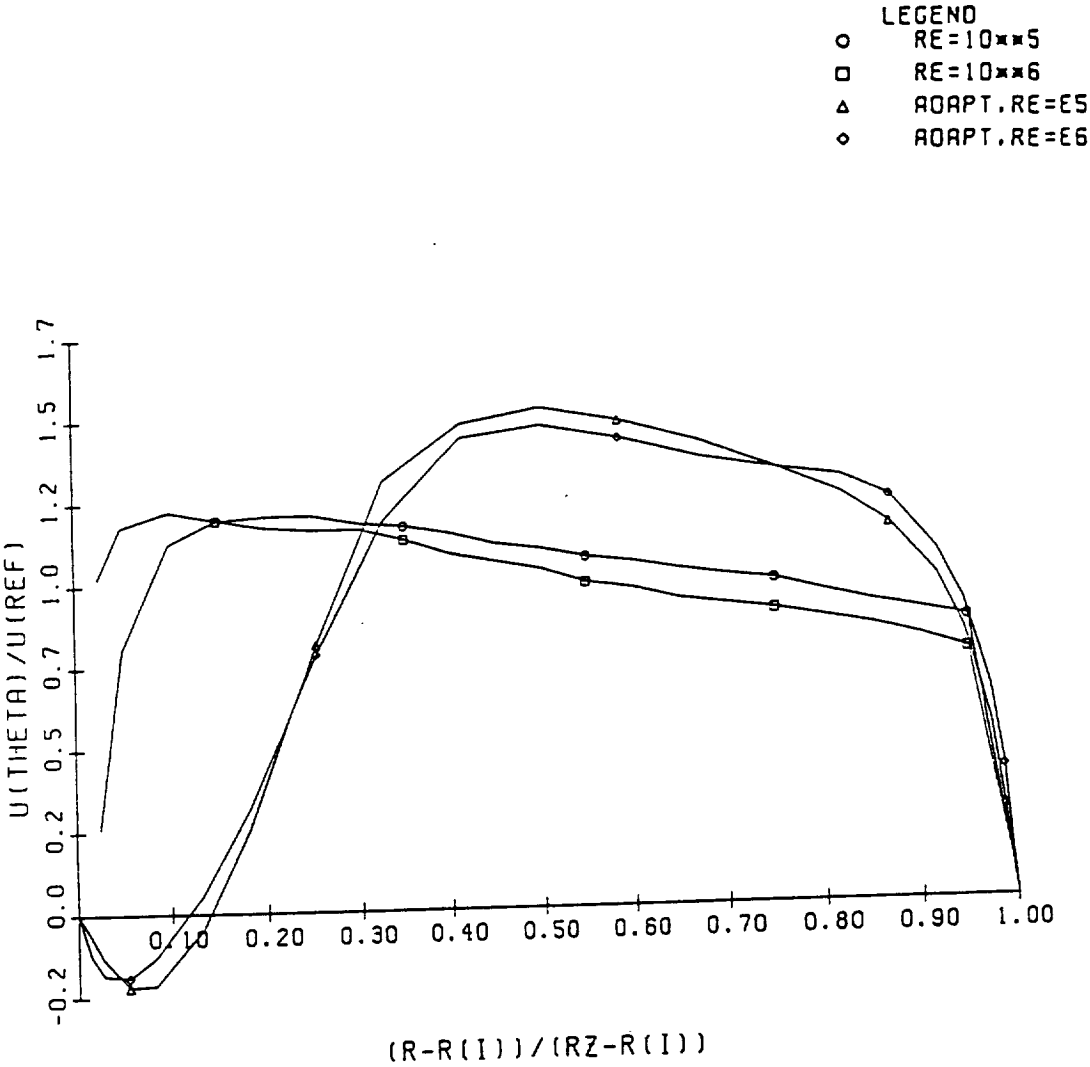
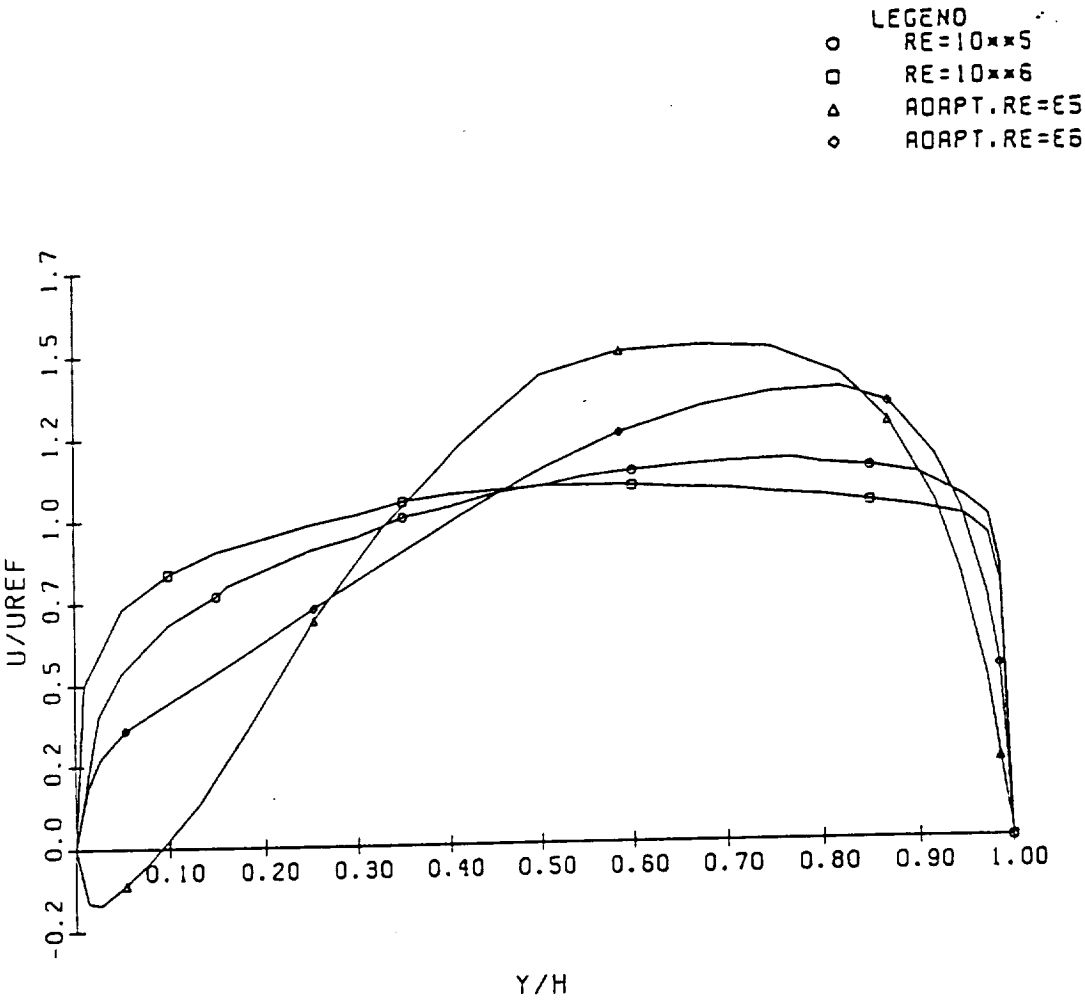
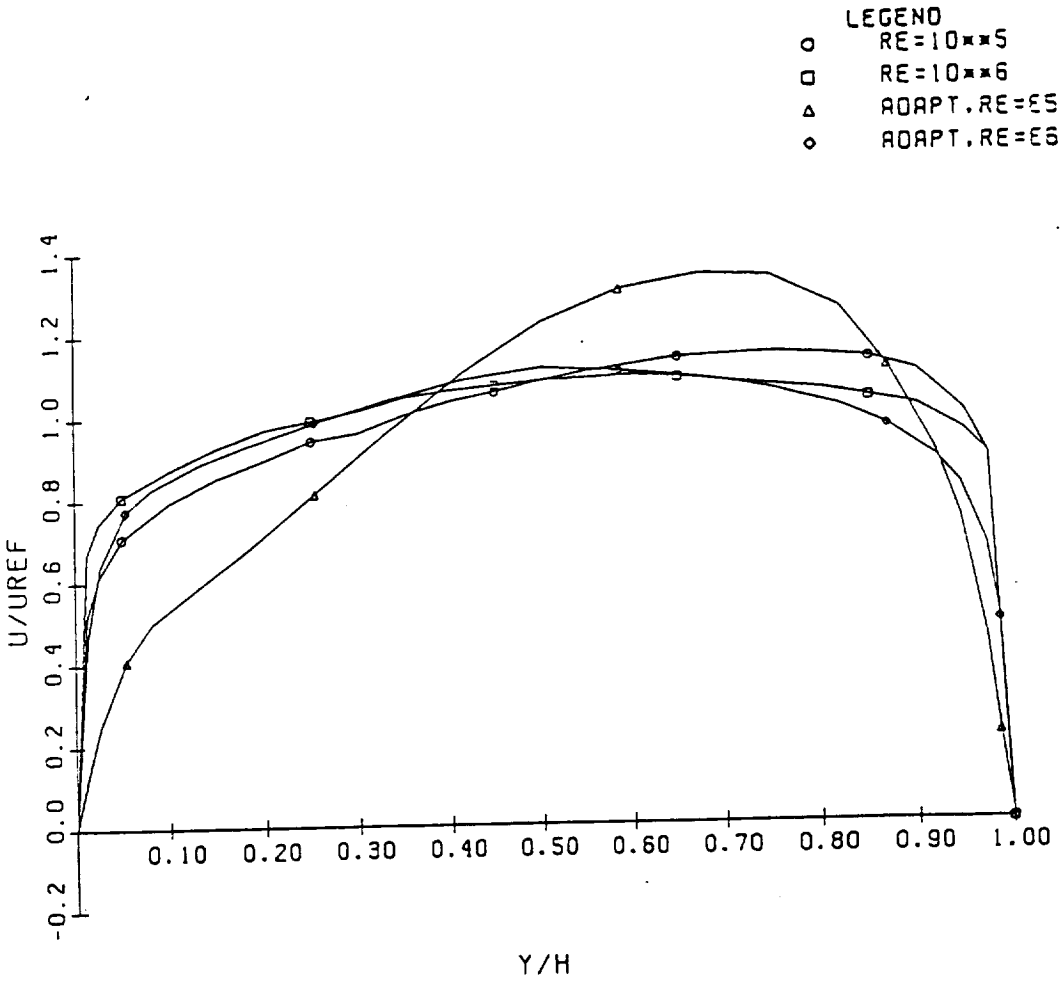


Figure 8.18: Normalized longitudinal velocity distribution at $\theta = 180.0$ deg.



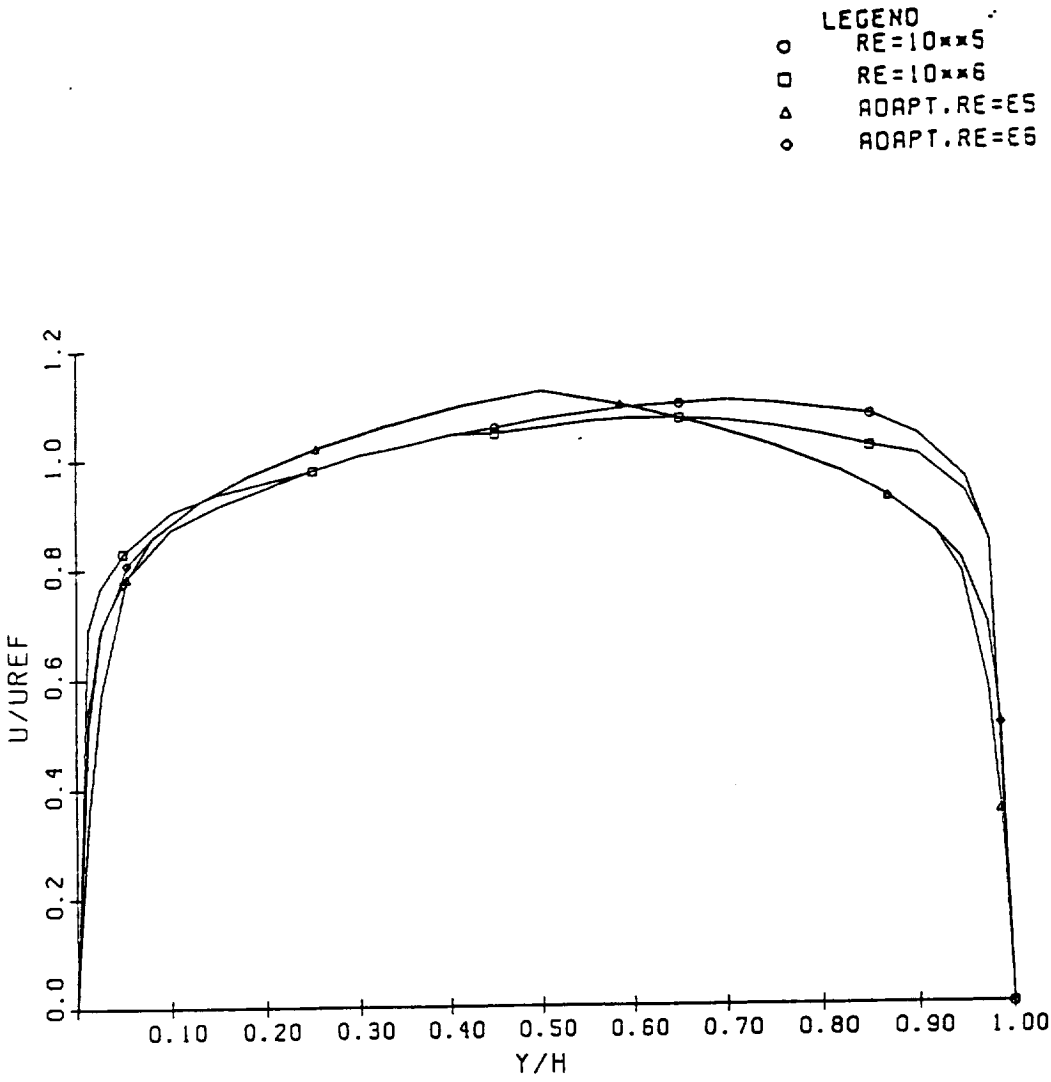
LOGITUDINAL VELOCITY DISTRIBUTION
AT X/H = 2.0

Figure 8.19: Normalized longitudinal velocity distribution at $x/H = 2.0$.



LOGITUDINAL VELOCITY DISTRIBUTION
AT X/H = 4.0

Figure 8.20: Normalized longitudinal velocity distribution at $x/H = 4.0$.



LOGITUDINAL VELOCITY DISTRIBUTION
AT X/H = 12.0

Figure 8.21: Normalized longitudinal velocity distribution at $x/H = 12.0$.

developed at the outflow. Once the flow becomes supersonic at the nozzle exit, the pressure outflow boundary condition was turned off. A method that is similar to the extrapolation boundary condition procedure used by the finite difference community is then imposed on the supersonic outflow boundary. Due to the nature of the subsonic flow within the cylindrical port field, the pressure disturbance will be propagated from the nozzle section upstream toward the head end and reflected back and forth within the chamber. After 1000 fully implicit time steps, this type of disturbance still can be observed in the numerical solution. The numerical solutions presented here were obtained after 2000 time steps.

Figure 8.22 shows the grid and a plot of the streamlines. It can be seen clearly that the internal flow was driven by the mass injection from the surface of the cylinder. The comparisons of the normalized axial velocity distribution at several axial locations are shown in Figs. 8.23–8.26. Very good agreement between our code, MINT and experimental data are evident.

The adaptive package was activated after 200 time steps and the grid was refined to the first level. The resulting mesh contained 2560 elements and 2594 nodes as shown in Fig. 8.27. As can be seen from Fig. 8.27a, refinement is clustered in the near wall region in the divergent section of the nozzle to resolve the viscous boundary layer. The viscous boundary layer grows rapidly and significantly reduces the effective cross section area in the divergent part as shown in Fig. 8.27b.

8.6 Porous Cylinder with Nozzle (Axisymmetric Case)

To test the axisymmetric option in conjunction with a porous boundary condition, the previous test case was resolved with the grid size and all flow conditions, boundary conditions, and time stepping procedures exactly the same as those used in test case 8.5.

Very similar numerical results were obtained as compared with the previous test case. Figure 8.28 shows the initial grid and the streamline. It can clearly be seen that the internal flow was driven by the mass injection from the surface of the cylinder. The pressure contours shown in Fig. 8.29 indicates that in the nozzle chamber the pressure is nearly constant and the pressure changes rapidly in the vicinity of the nozzle throat due to a fast flow expansion. Figure 8.30 shows the velocity distribution in the nozzle section. A relatively thin boundary layer extended from the nozzle wall can be observed. After the internal flow was developed to a certain stage, we turned on the adaptive options with the maximum level of grid refinement limited to the first level. The grid was adapted in the region close to the viscous boundary to resolve the rapid change of flow velocity as shown in Fig. 8.31.

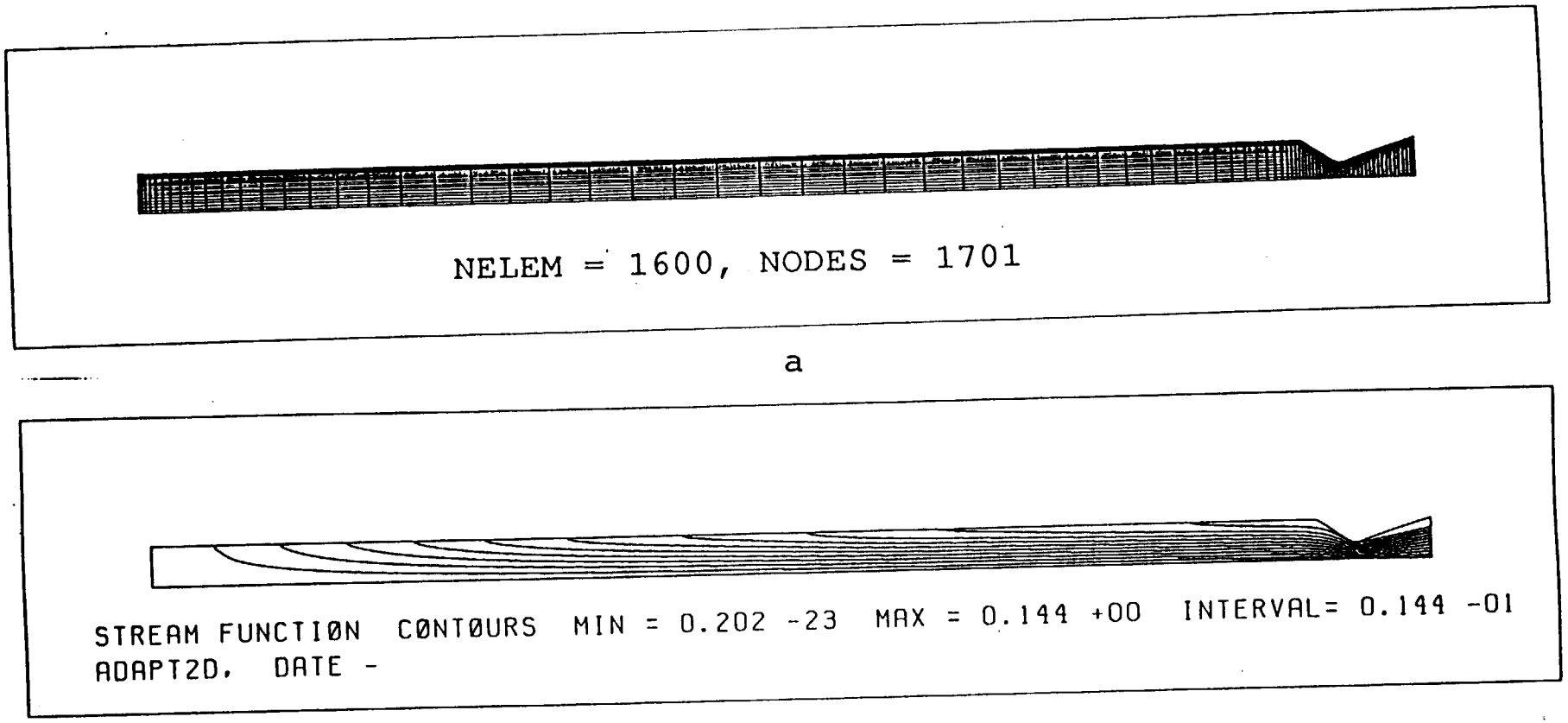
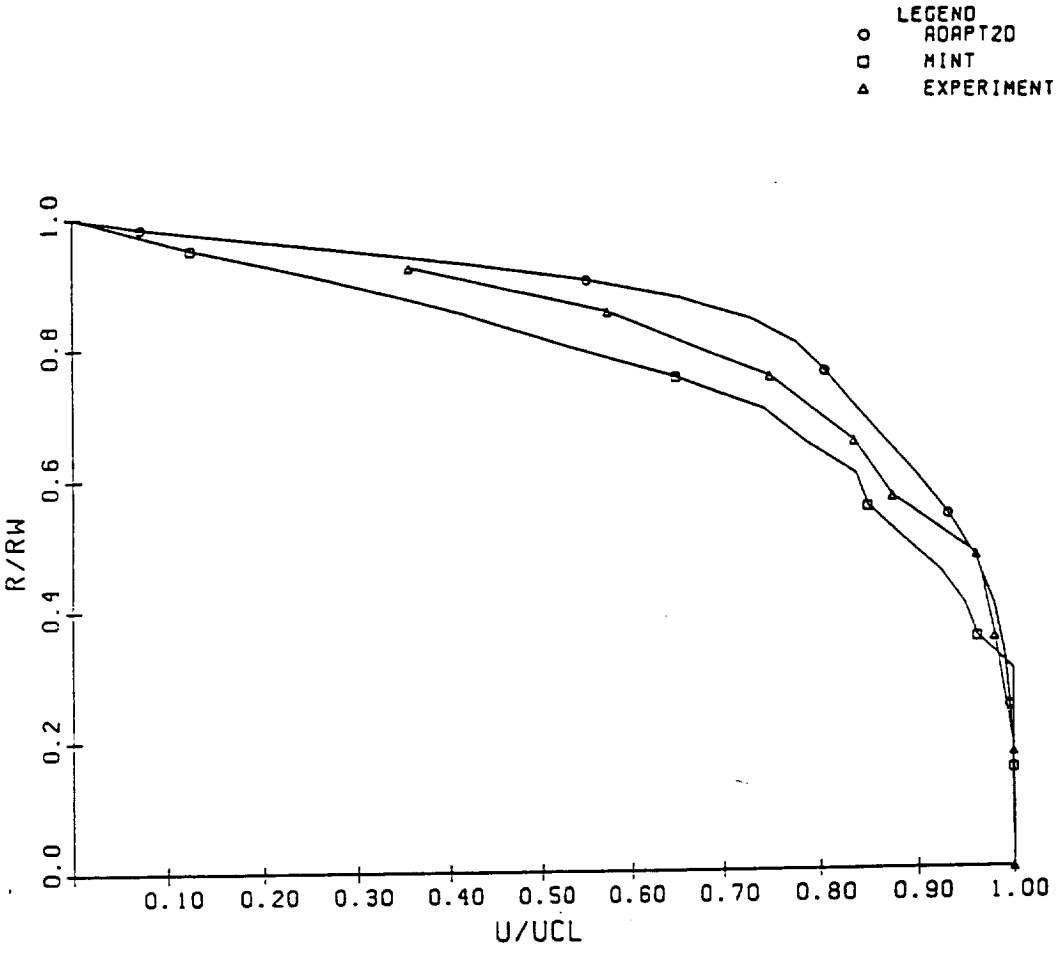
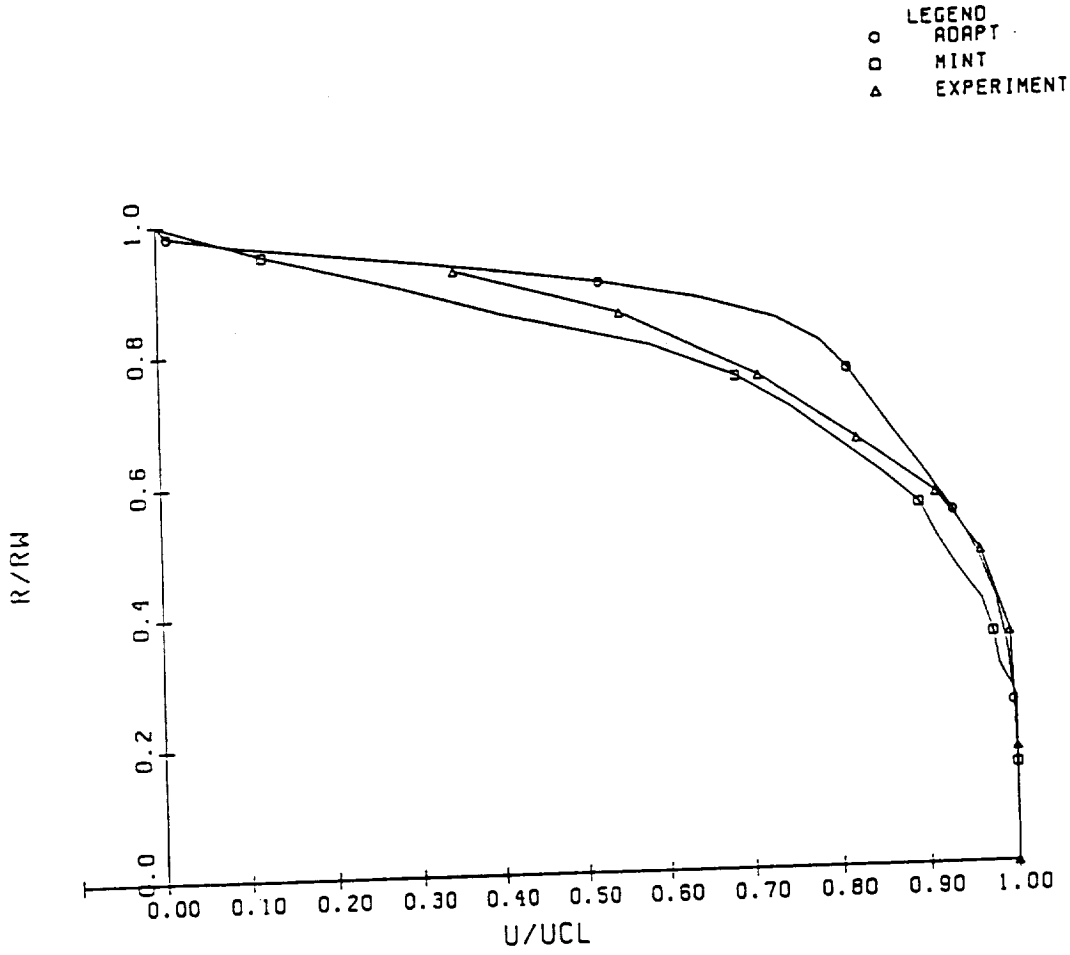


Figure 8.22: Porous cylinder with a nozzle. (a) grid, (b) streamlines.



NORMALIZED AXIAL VELOCITY DISTRIBUTION
AT Z/D = 4.22

Figure 8.23: Normalized axial velocity distribution at $Z/D = 4.22$.



NORMALIZED AXIAL VELOCITY DISTRIBUTION
AT Z/D = 6.64

Figure 8.24: Normalized axial velocity distribution at $Z/D = 6.64$..

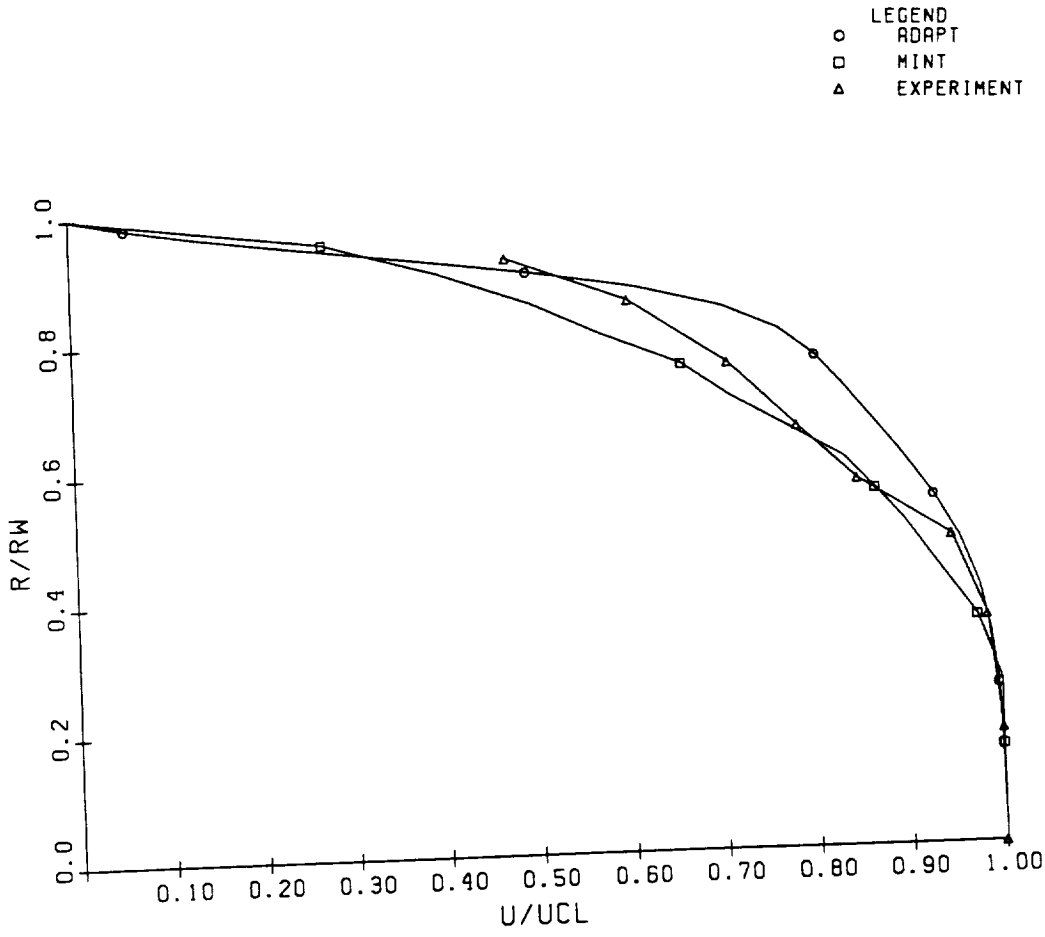


Figure 8.25: Normalized axial velocity distribution at $Z/D = 9.06$.

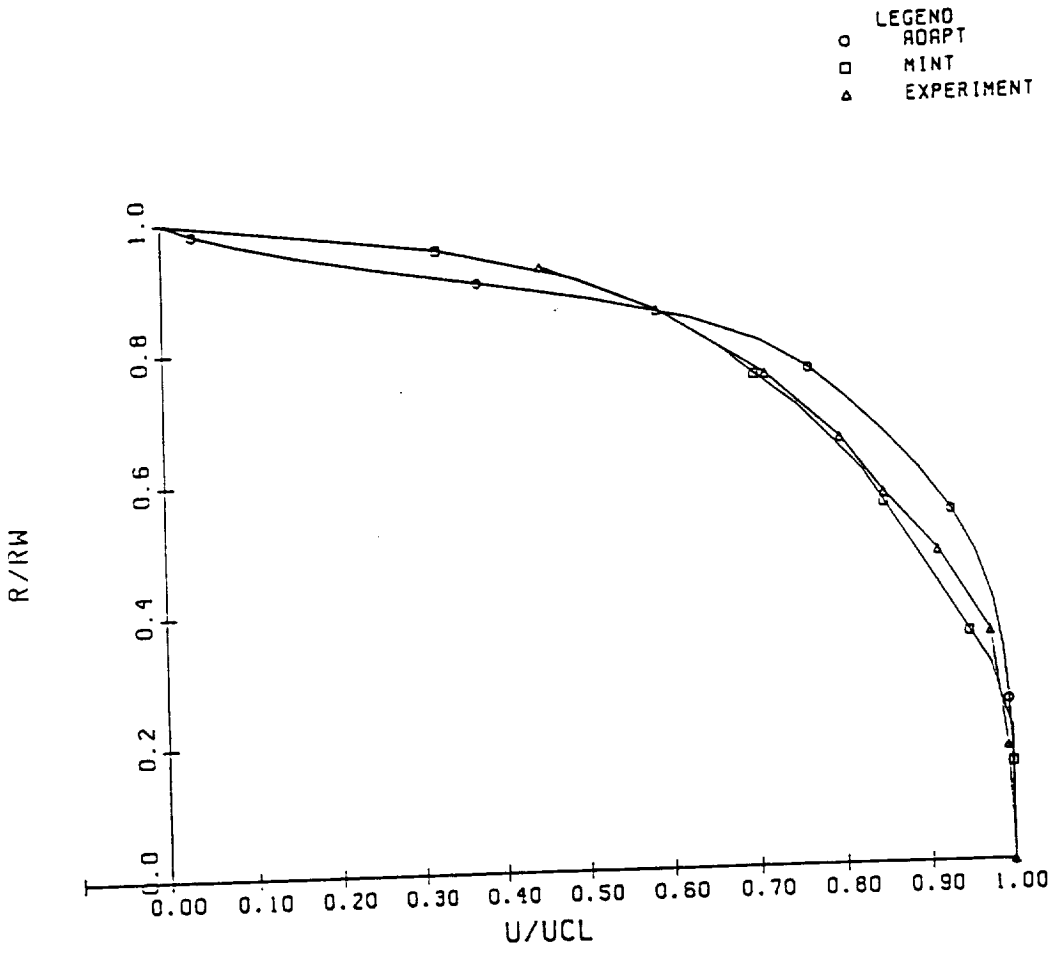
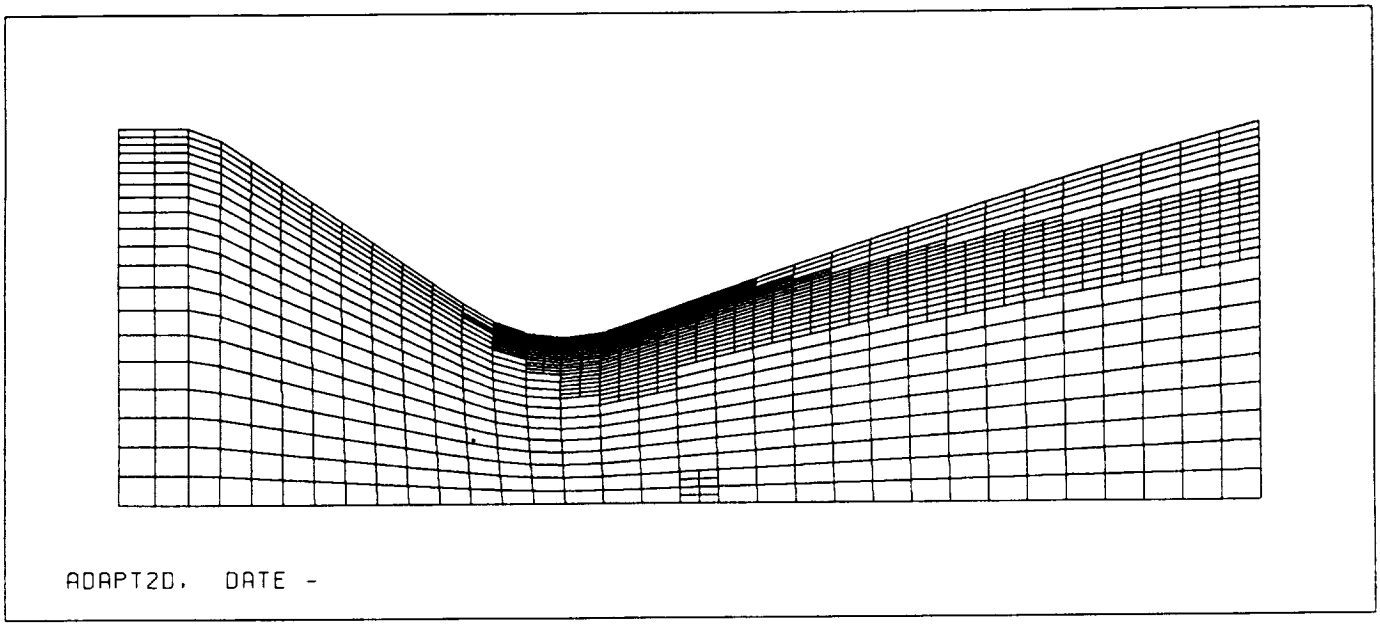
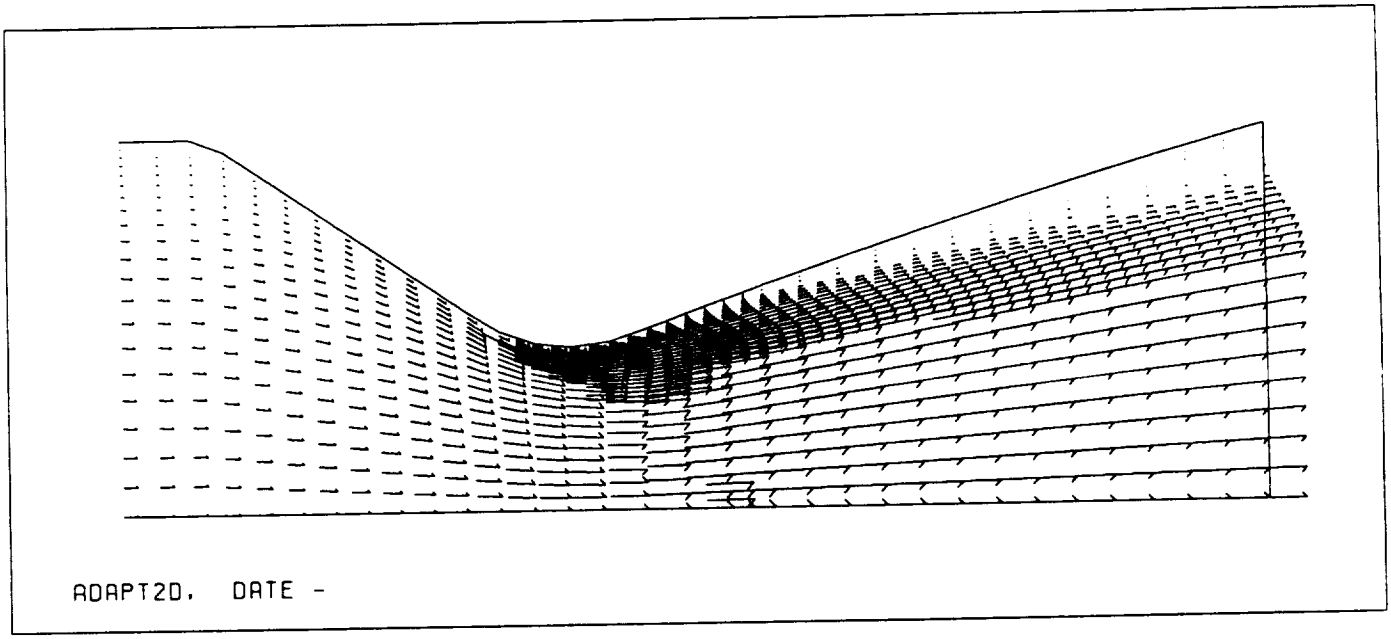


Figure 8.26: Normalized axial velocity distribution at $Z/D = 12.72$.

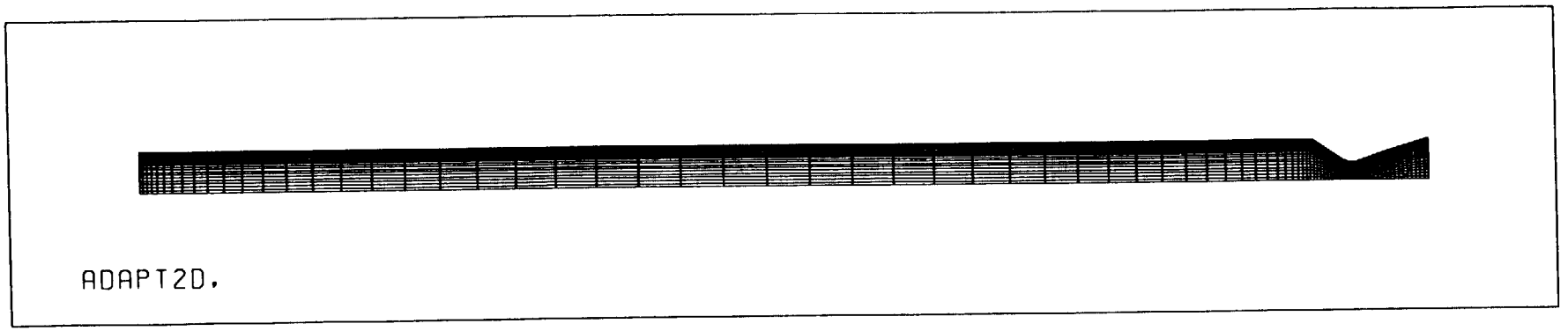


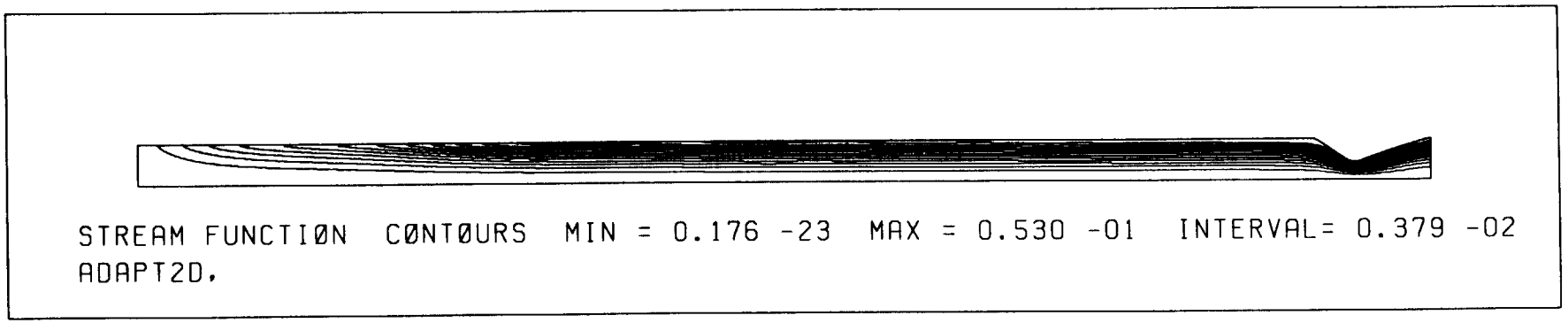
a



b

Figure 8.27: Closeup view of (a) adapted grid, and (b) velocity profiles in the nozzle section.





PØRØUS CYLINDER WITH A NØZZLE, AXISYMMETRIC CASE
STRUCTURED GRID, EXPLICIT-IMPLICIT ALGØRITHM

Figure 8.28: Initial grid and the stream function contours.

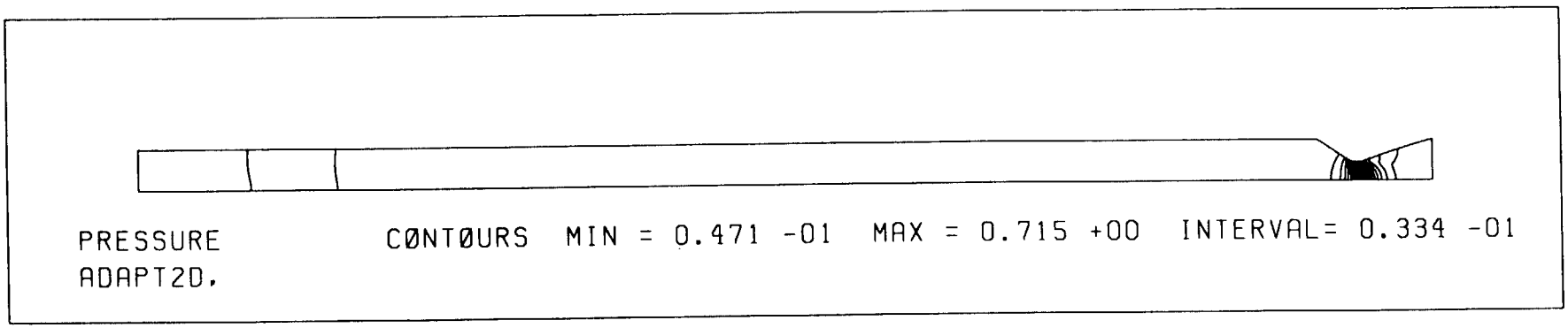
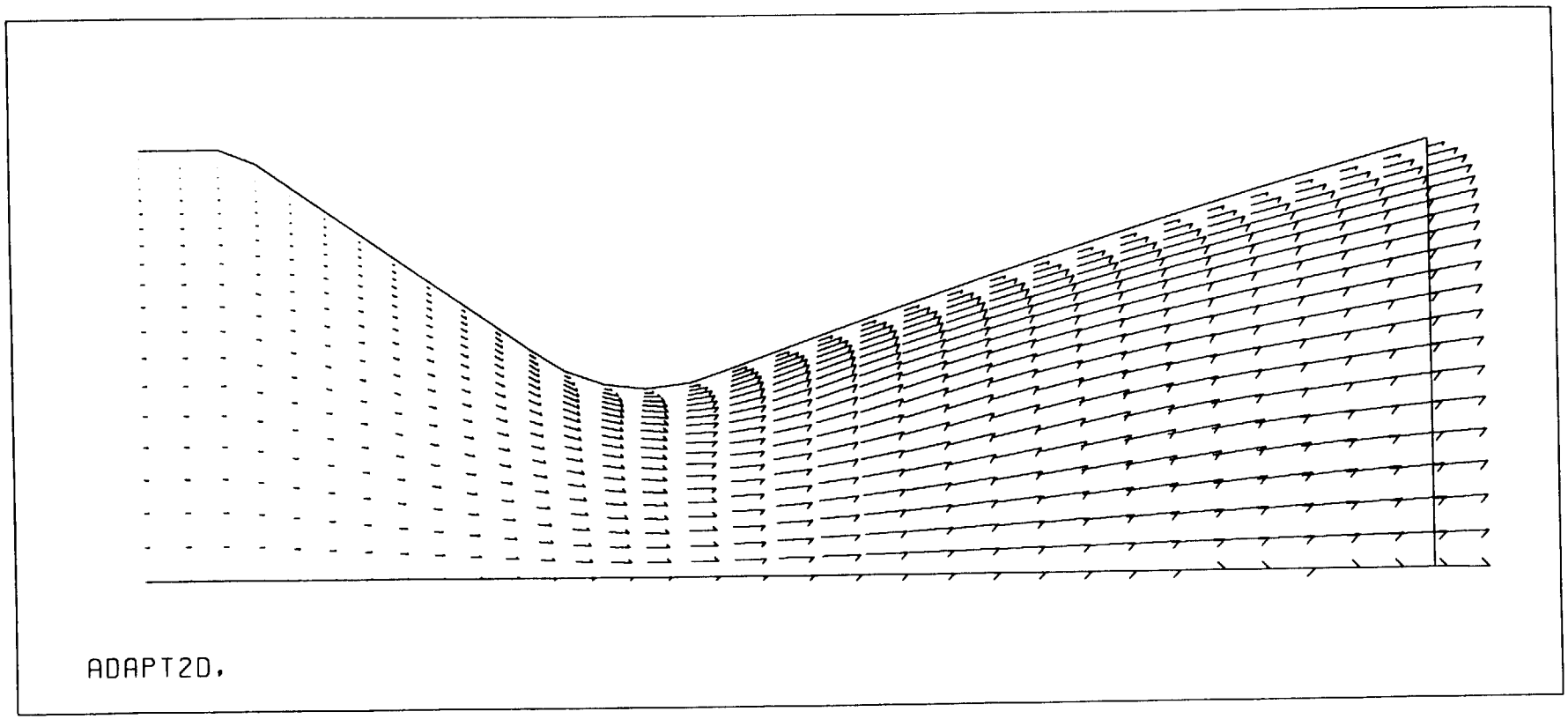
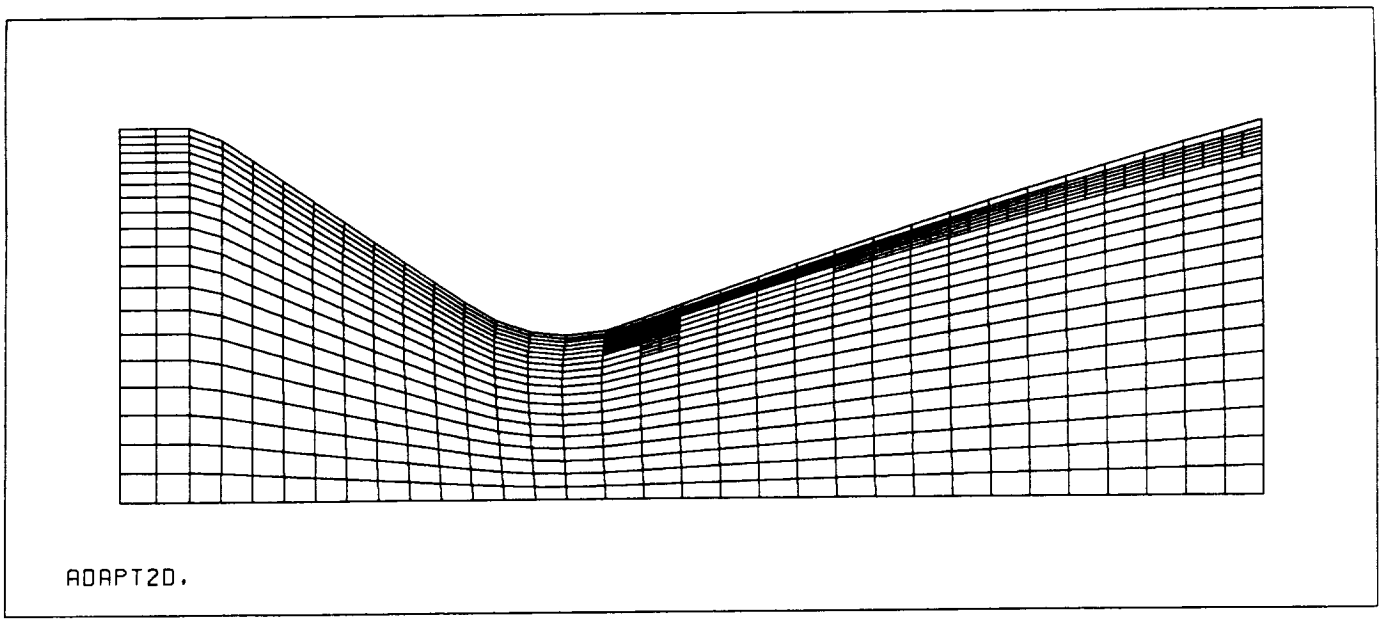


Figure 8.29: Pressure contours.



PØRØUS CYLINDER WITH A NØZZLE, CARTESIAN CASE
STRUCTURED GRID, EXPLICIT-IMPLICIT ALGØRITHM

Figure 8.30: Velocity distribution in the nozzle section.



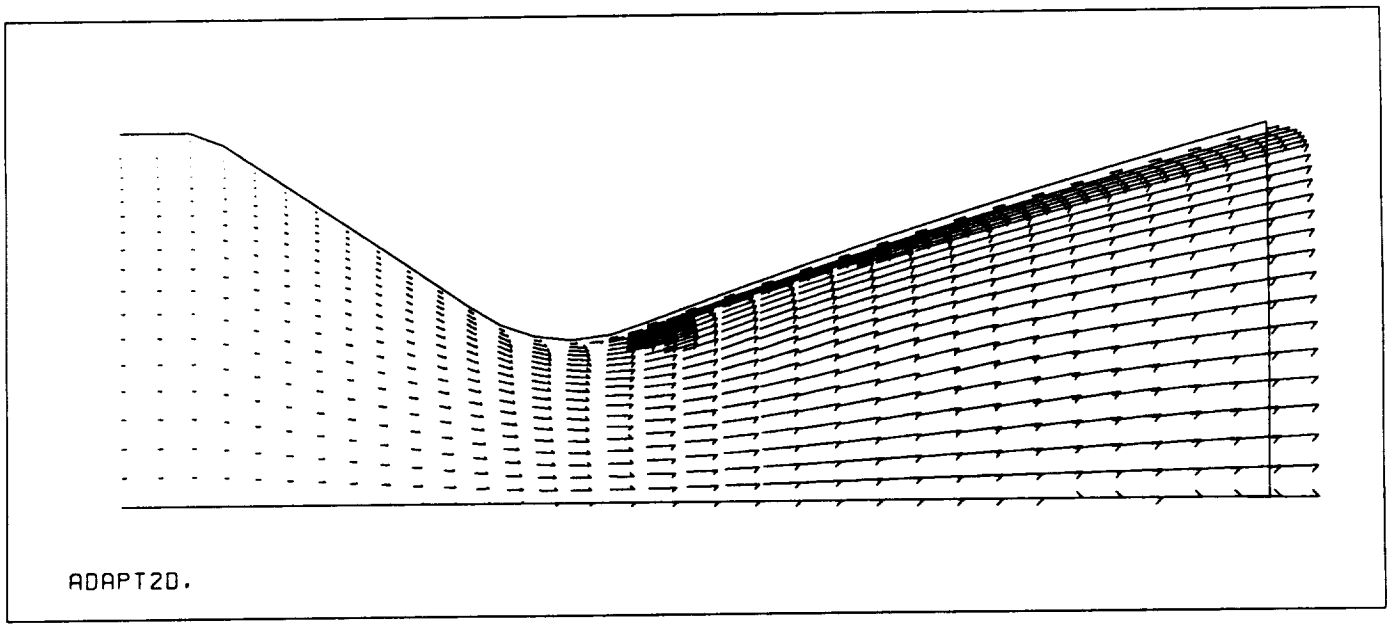


Figure 8.31: Adapted grid and velocity distribution in the nozzle section.

8.7 Slotted Chamber with Nozzle (Axisymmetric Case)

The purpose of this test case is to validate the axisymmetric modeling capability for simulating internal nozzle flows with mass injection. In addition, an algebraic turbulence model was applied. The nozzle geometry was constructed using piecewise analytic functions. The inhibitor was treated as a radial slot and attached to the cylinder (see [39] for detailed geometry specifications).

The initial grid consists of 70×20 elements in the combustion chamber region and 30×20 elements in the nozzle section. The flow conditions employed includes the normalized mass injection rate = 0.0027 and Reynolds number = 8.0×10^5. The mass injection rate is held constant for all porous boundaries. The initial condition and the rest of the boundary conditions are exactly the same as the previous test case. The surfaces of the inhibitor are treated as porous boundaries. The numerical solutions presented here were obtained after 4000 time steps. Figure 8.32 shows the grid an streamline plots. It can be seen that the internal flow was driven by the mass injection from the surface of the cylinder. A global velocity vector plot and details of the flowfield around the aft-end of the slot are displayed in Fig. 8.33a. The high speed flow injected from the slot to the chamber caused a reversed flow in the vicinity of the joint of the slot and the cylinder, see Fig. 8.33b. Chamber pressure is nearly constant and varies rapidly in the nozzle section as shown in Fig. 8.33c. The closed-up view of the velocity vector plots in the nozzle section and Mach number contours are shown in Fig. 8.34. These figures confirm that the large contraction of the convergent part leads to a rapid flow expansion.

The comparisons of the normalized axial velocity distribution at several axial locations are shown in Figs. 8.35–8.43. Generally speaking, the velocity boundary layers at all locations predicted by our code are much thinner. This discrepancy may be caused by many reasons such as the grid resolution, distribution of the mass injection rate, the use of appropriate turbulence models, and flow unsteadiness, etc. Further numerical studies are necessary to explain these differences.

After 4000 time steps, the grid was adapted to the first level. The final grid consists of 2315 elements and 2437 nodes. As expected, most of the adaptation occurred along the solid wall of the divergent part of the nozzle as shown in Fig. 8.44a. A very thin velocity boundary layer can be observed in Fig. 8.44b.

8.8 Moving Grid Algorithm

To test the moving grid algorithm, a 45°–15° convergent–divergent nozzle [31], with an extended cylindrical part for modeling the solid propellant section, was used. As mentioned in Section 5, if the grid motion is coupled with the fluids problem, the boundary motion could

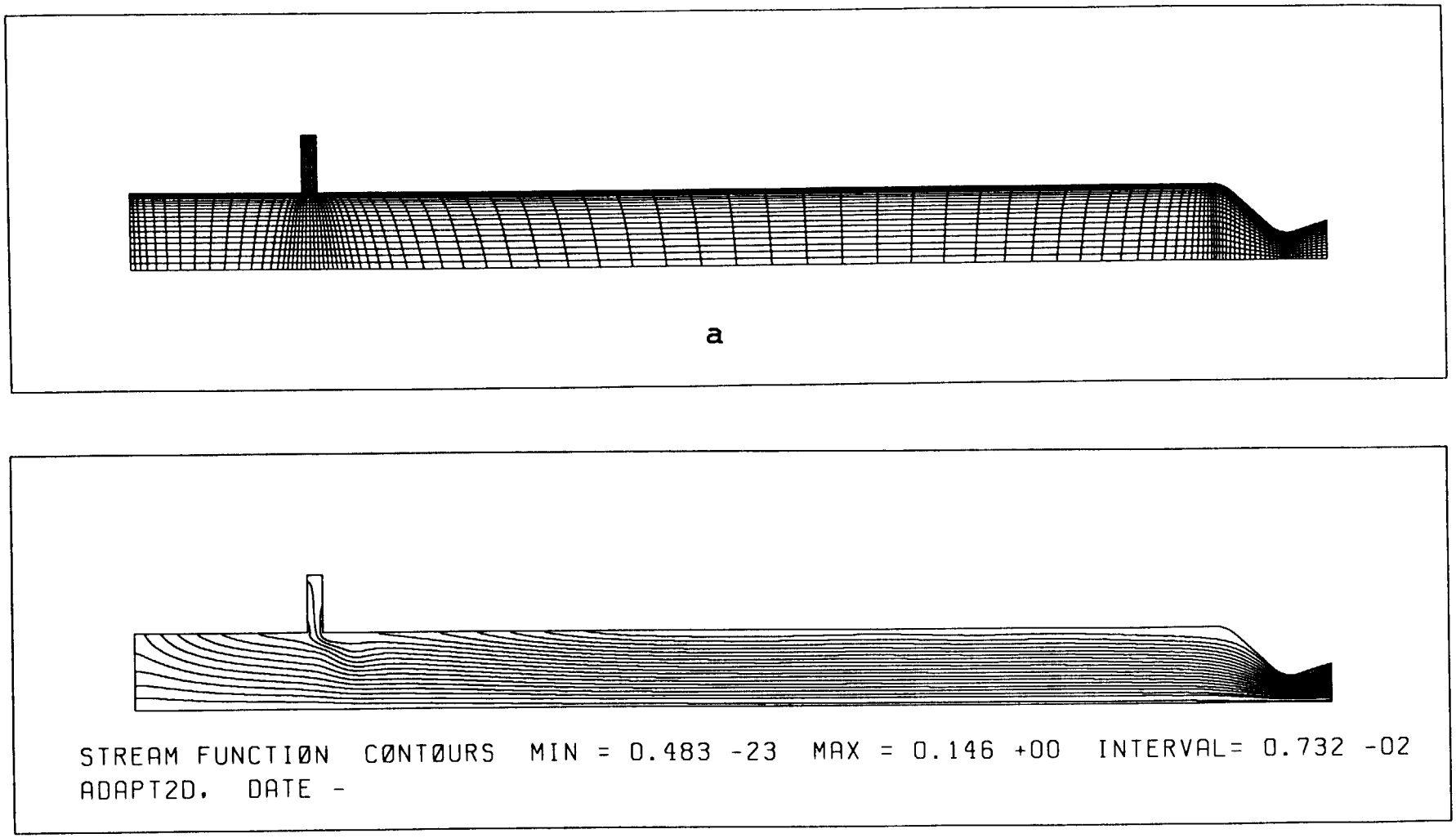
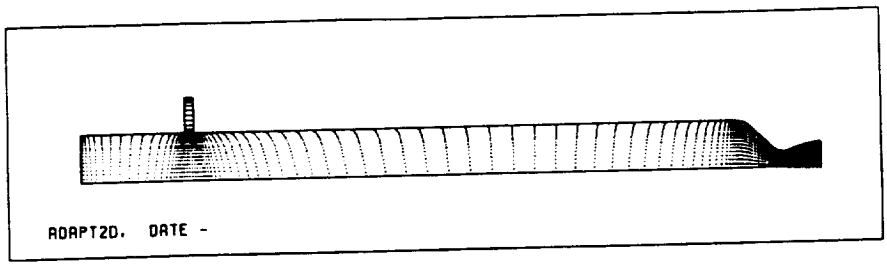
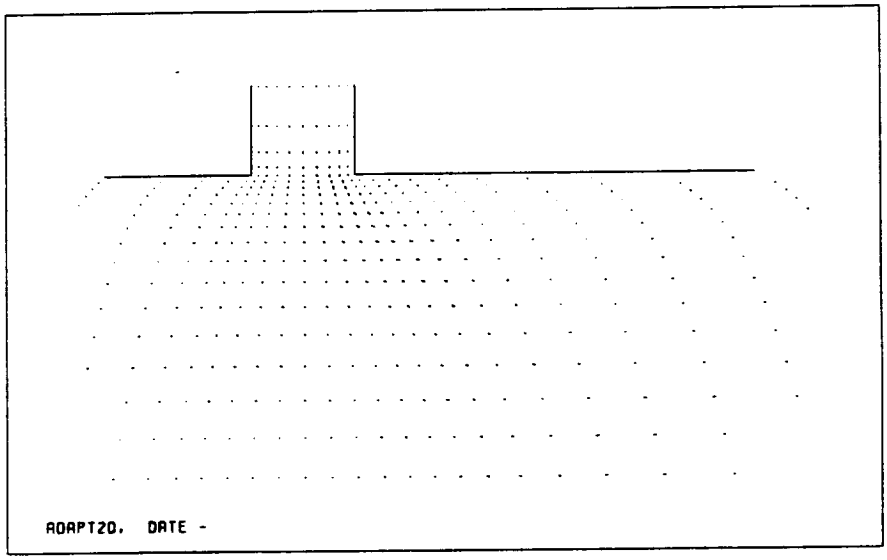


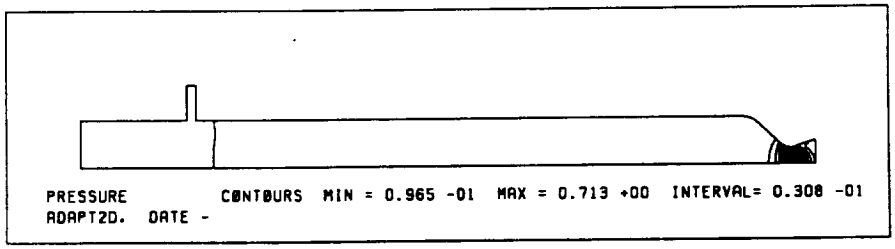
Figure 8.32: Slotted chamber with a nozzle. (a) grid, and (b) streamlines.



a

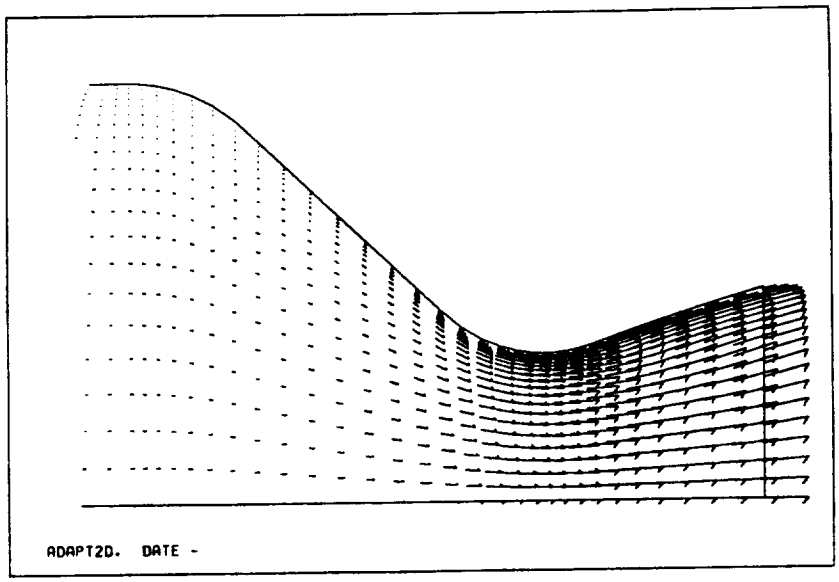


b

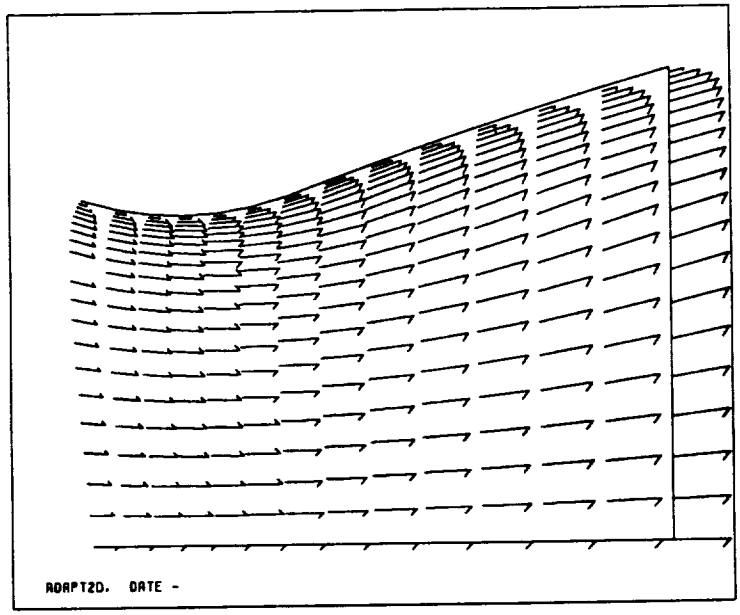


c

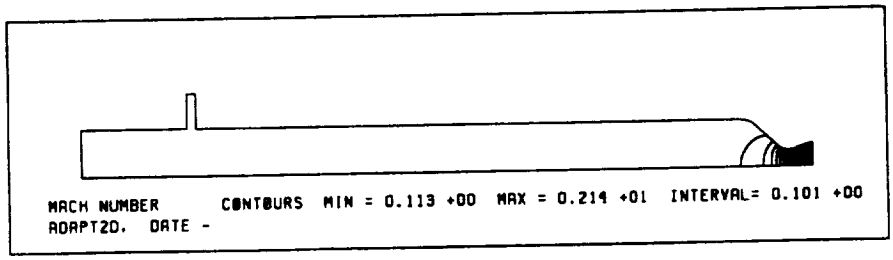
Figure 8.33: Slotted chamber with a nozzle. (a) velocity vectors, (b) velocity vectors in the vicinity of the slot, and (c) pressure contours.



a



b



c

Figure 8.34: Slotted chamber with a nozzle. (a) velocity vectors in the nozzle section, (b) velocity vectors in the divergent section, and (c) Mach contours.

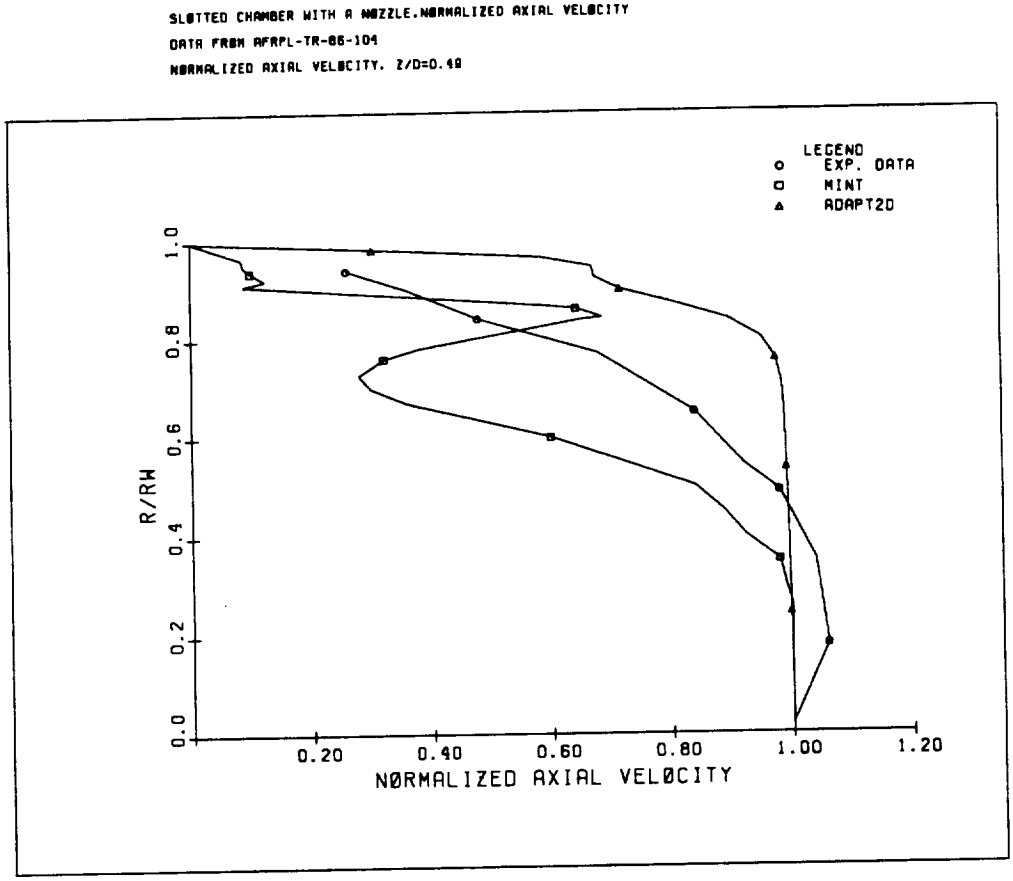


Figure 8.35: Normalized axial velocity distribution at $Z/D = 0.49$.

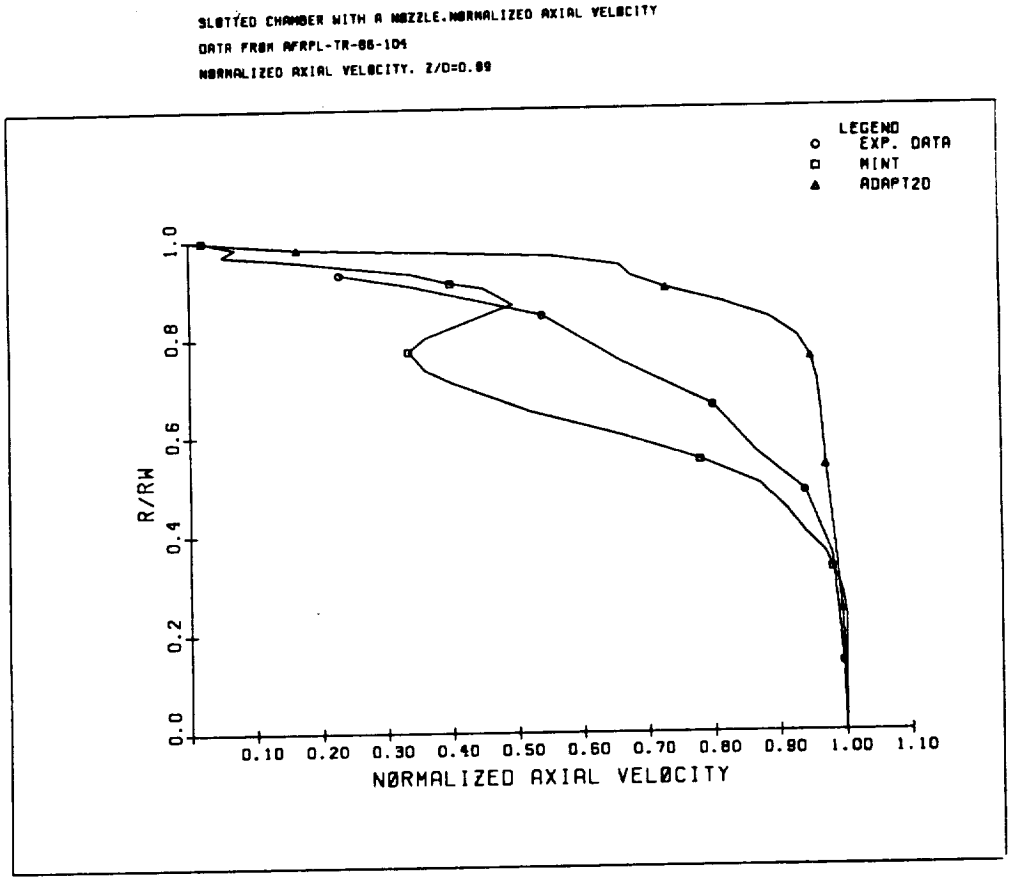


Figure 8.36: Normalized axial velocity distribution at $Z/D = 0.89$.

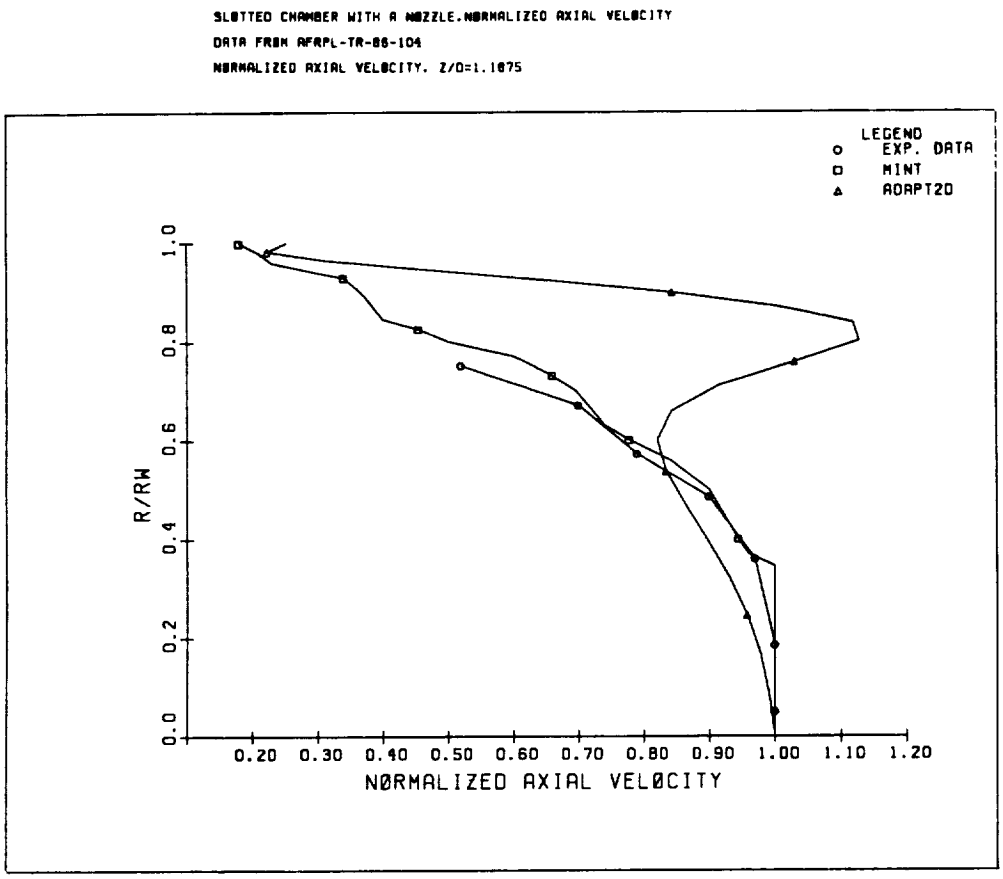


Figure 8.37: Normalized axial velocity distribution at $Z/D = 1.1875$.

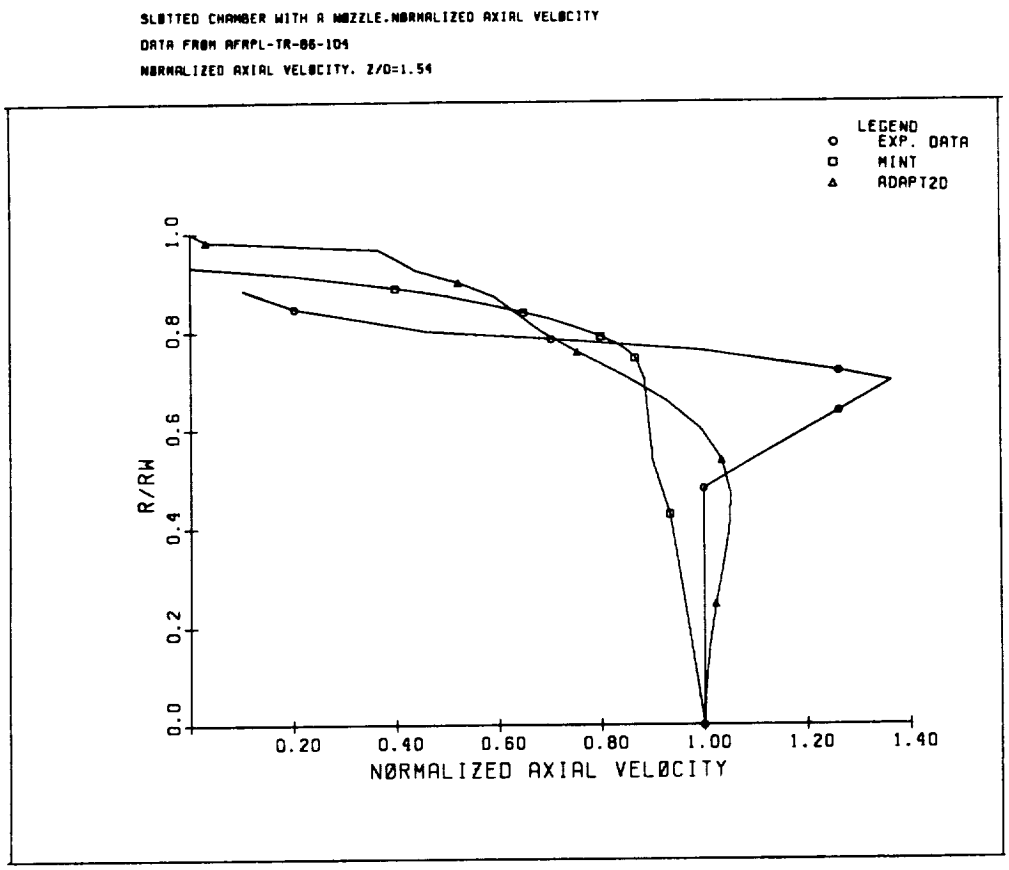


Figure 8.38: Normalized axial velocity distribution at $Z/D = 1.54$.

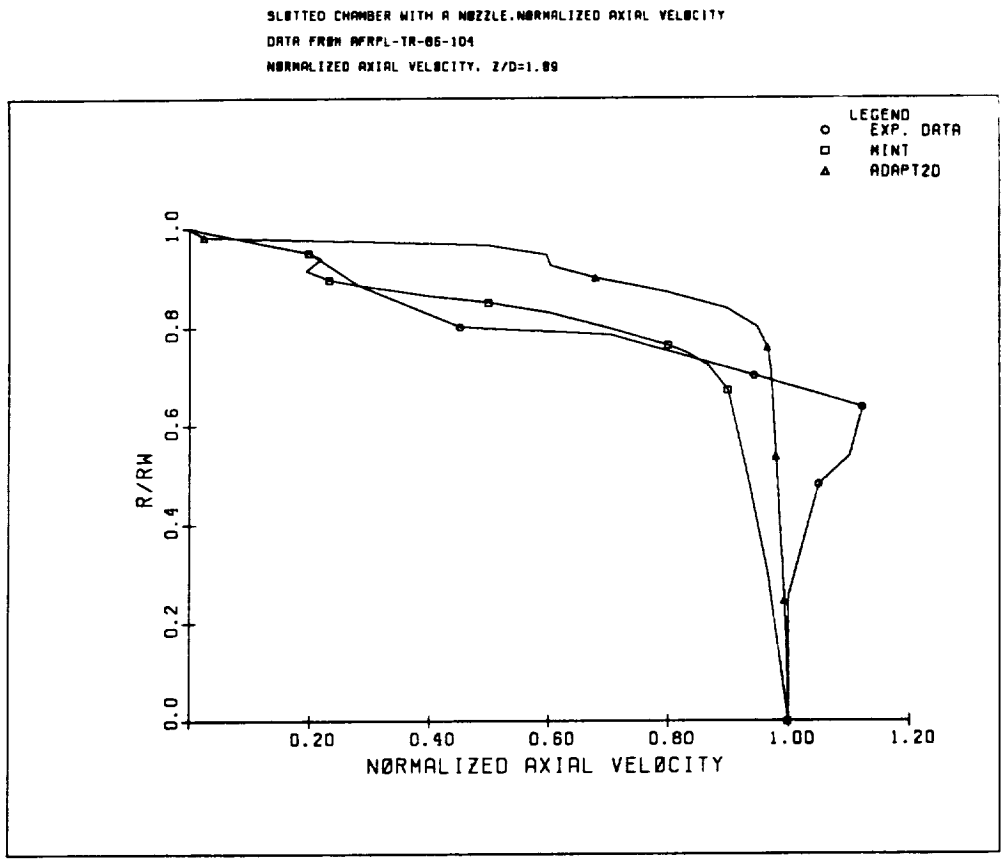


Figure 8.39: Normalized axial velocity distribution at $Z/D = 1.89$.

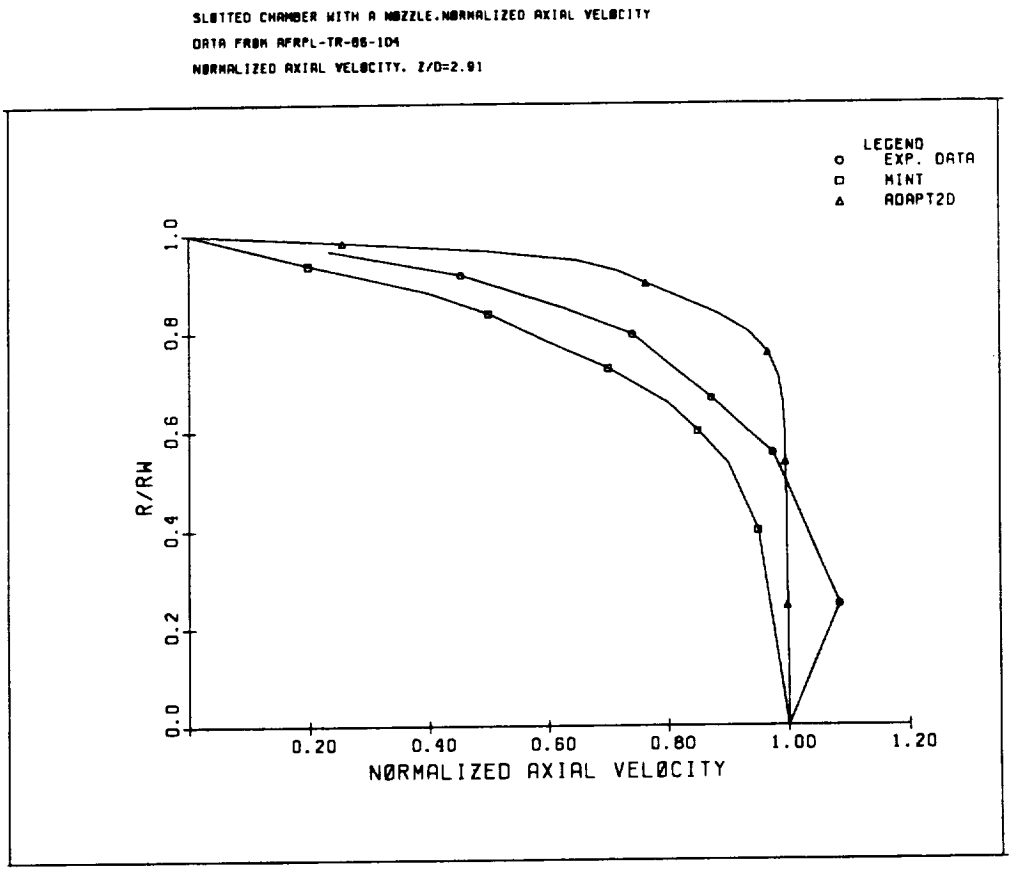


Figure 8.40: Normalized axial velocity distribution at $Z/D = 2.91$.

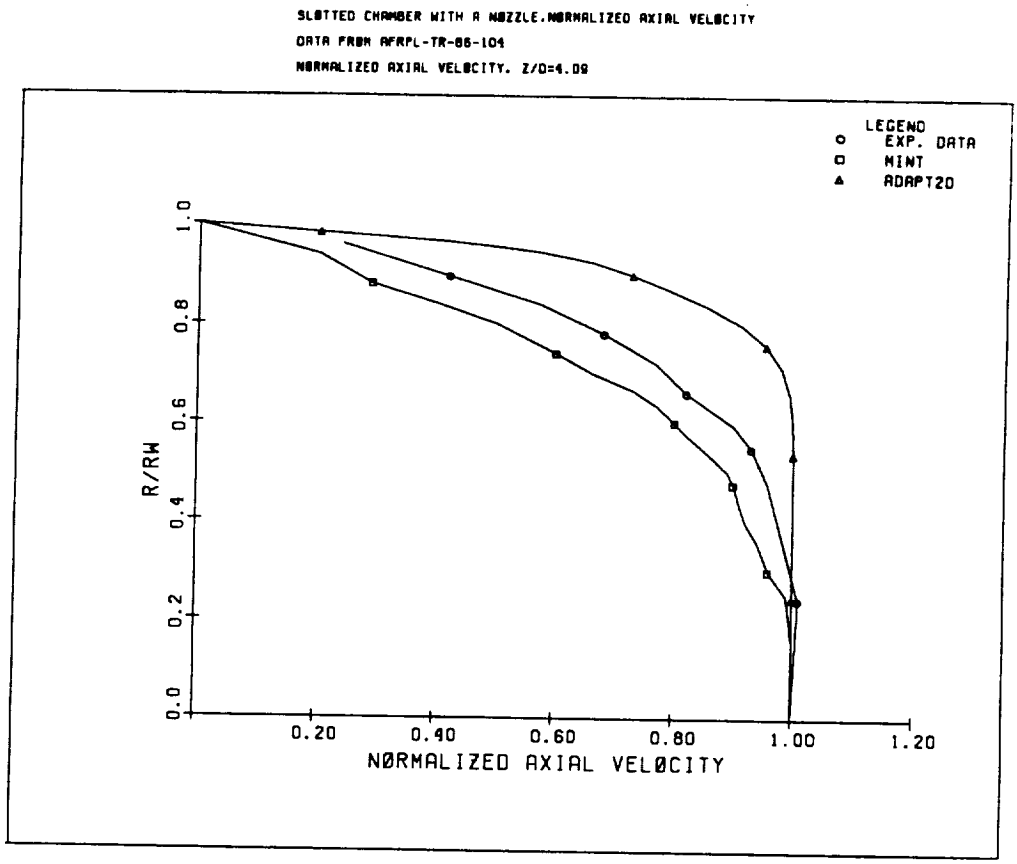


Figure 8.41: Normalized axial velocity distribution at $Z/D = 4.09$.

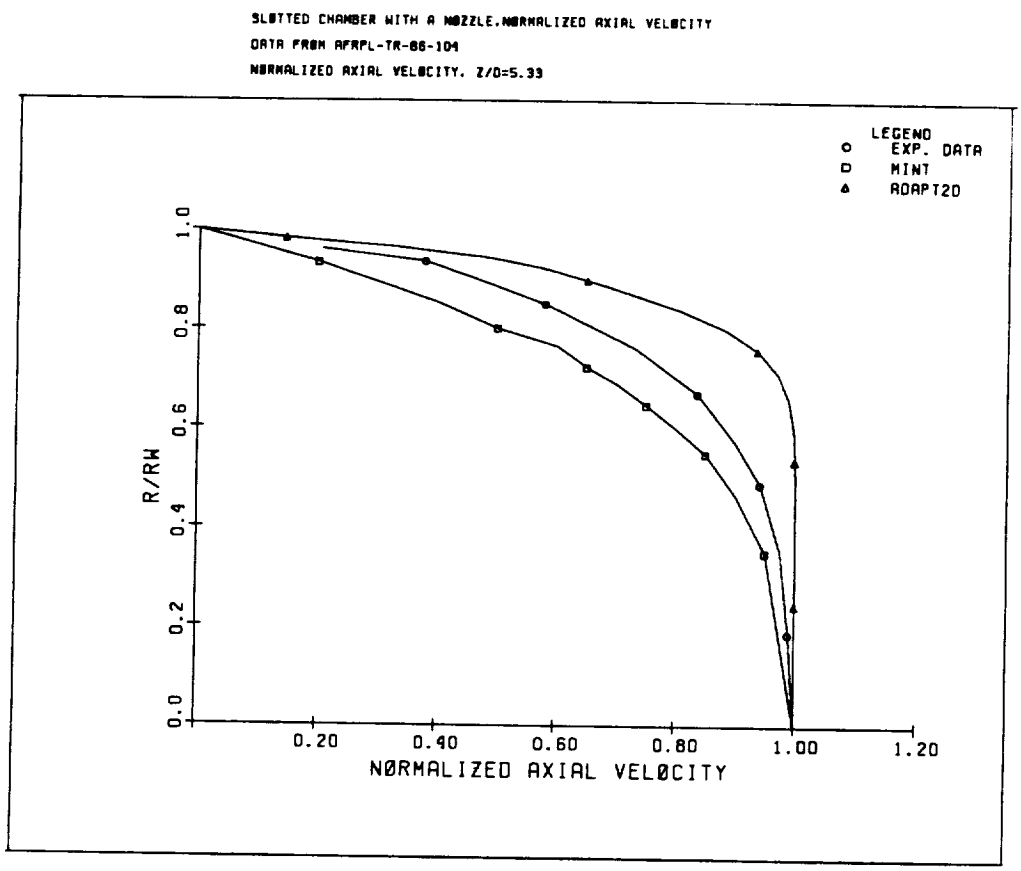


Figure 8.42: Normalized axial velocity distribution at $Z/D = 5.33$.

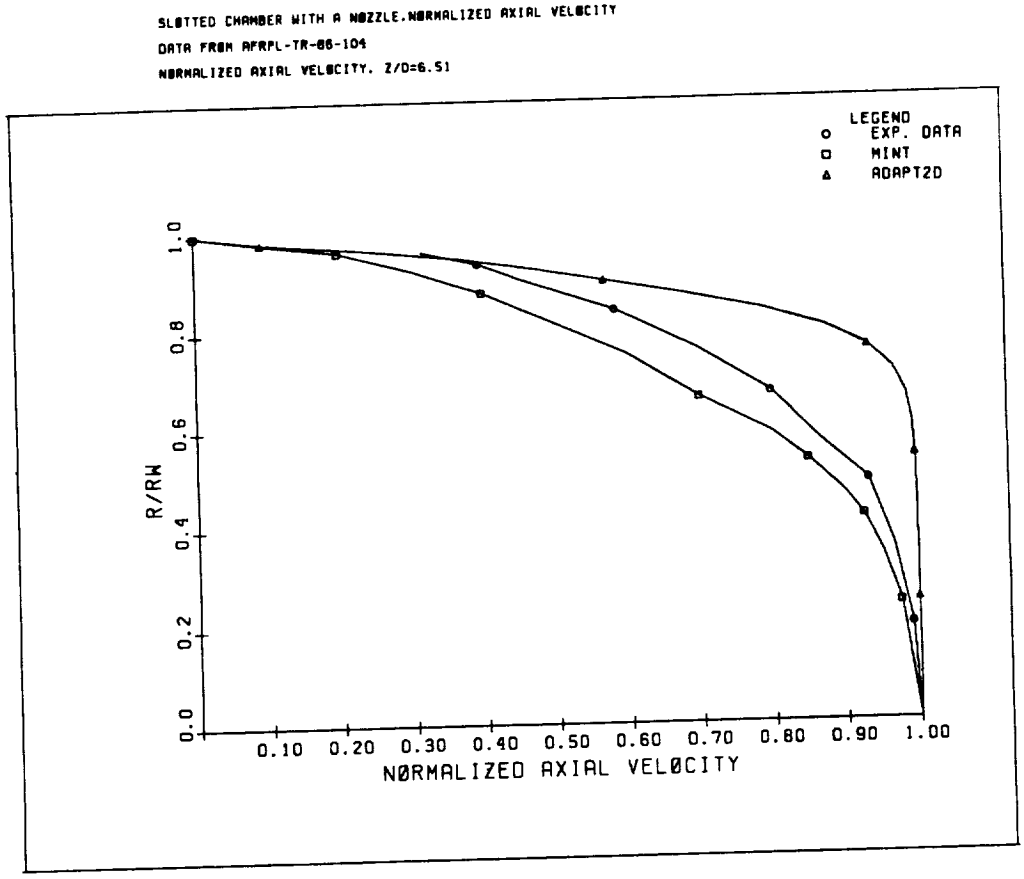
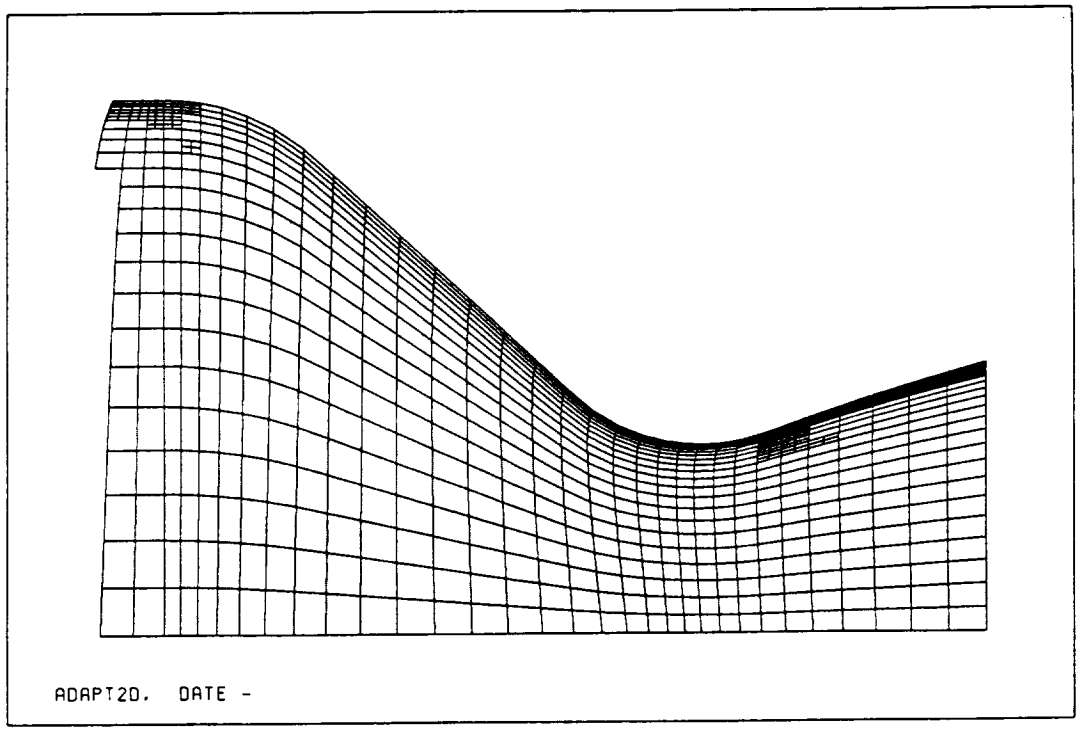
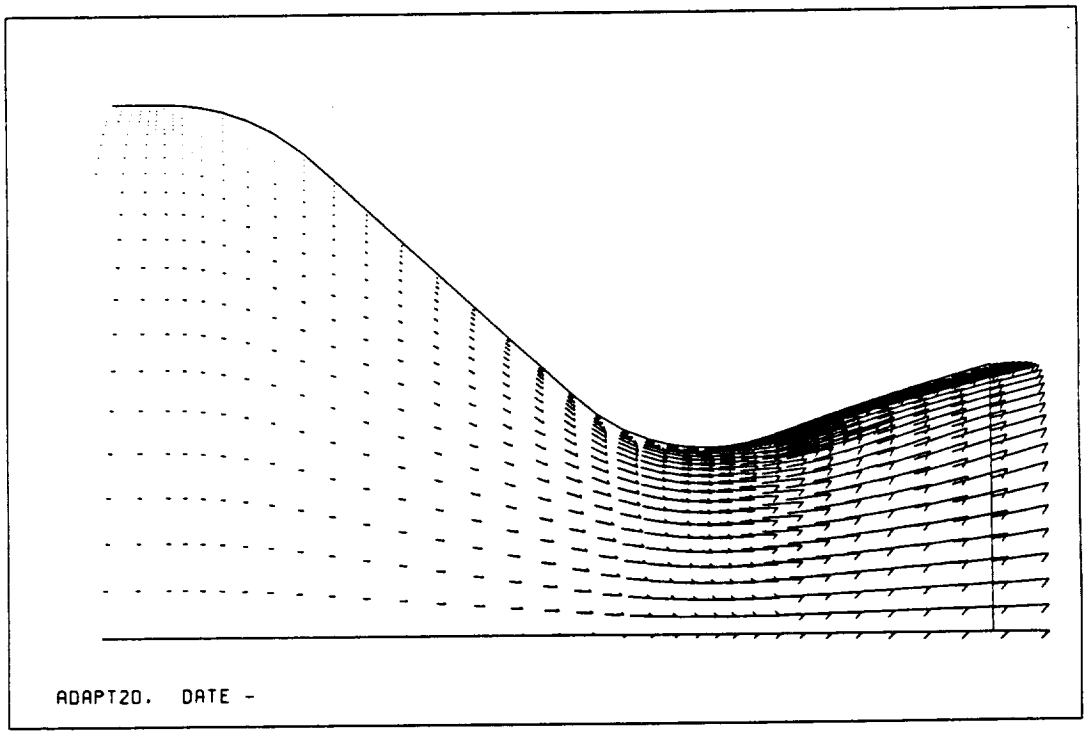


Figure 8.43: Normalized axial velocity distribution at $Z/D = 6.51$.



a



b

Figure 8.44: Closeup view of (a) adapted grid, and (b) velocity profiles in the nozzle section.

be extremely slow and is not suitable for large boundary motions. Therefore, the moving grid algorithm, Option 2, was used. The initial grid is shown in Fig. 8.45. After 1.5 time units of advancement, the grid distribution is shown in Fig. 8.46. It should be noted that the Laplacian operator that governs the grid motion is a smoothing operator and, therefore, the grid motion does not affect the quality of the initial grid as can be seen from Fig. 8.46.

9 Future Extensions

The success of the two–dimensional and axisymmetric analysis package in modeling various benchmark problems as demonstrated in Section 8 suggests that extensions of current simulation capability to realistic three–dimensional flows in cavities with eroding boundaries is feasible. In this section, we will briefly describe some possible extensions of this project with respect to the enhancement of the two–dimensional code and for the development of a fully three–dimensional capability for use in the design and analysis of solid rocket motors.

Two–Dimensional Enhancements

The two–dimensional code has numerous special capabilities for modeling subsonic to supersonic flow regions. To enhance these current capabilities the following options are proposed.

1. Develop a menu driven, x-window–based interactive preprocessor and postprocessor.
2. Develop a user–friendly two–dimensional structured/unstructured mesh generation package.
3. Optimize (vectorize/parallelize) the implicit/explicit solution module to provide peak performance on the MSFC Convex and/or Cray computers.
4. Enhance the moving boundary algorithm to include a physically based regression velocity which is a function of the flow characteristics such as pressure and temperature.
5. Implementation of higher–order turbulence models such as a k-ℓ or k-ε model.

Three–Dimensional Code

Many of the configurations used in solid rocket motors are three–dimensional in nature due to the shape of the combustion chamber. Thus a three–dimensional analysis capability which extends the two–dimensional/axisymmetric capabilities would clearly be of great interest and benefit. The following steps outline the development of a three–dimensional Navier-Stokes solid rocket motor code:

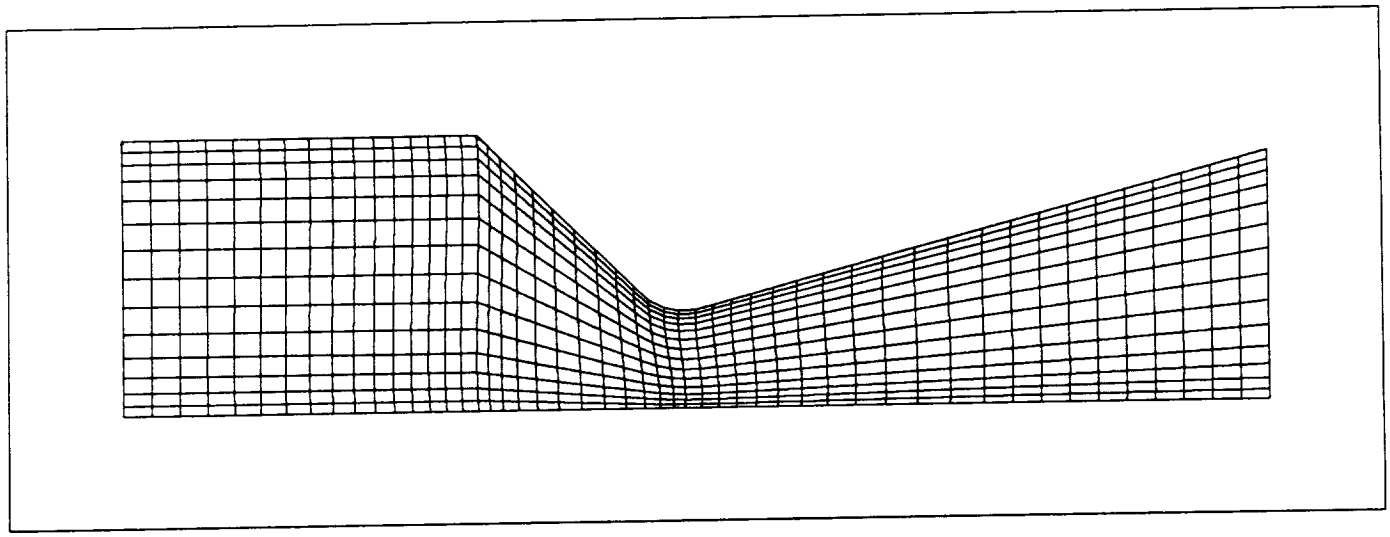


Figure 8.45: Initial grid used for simulating moving grid/eroding boundary algorithm.

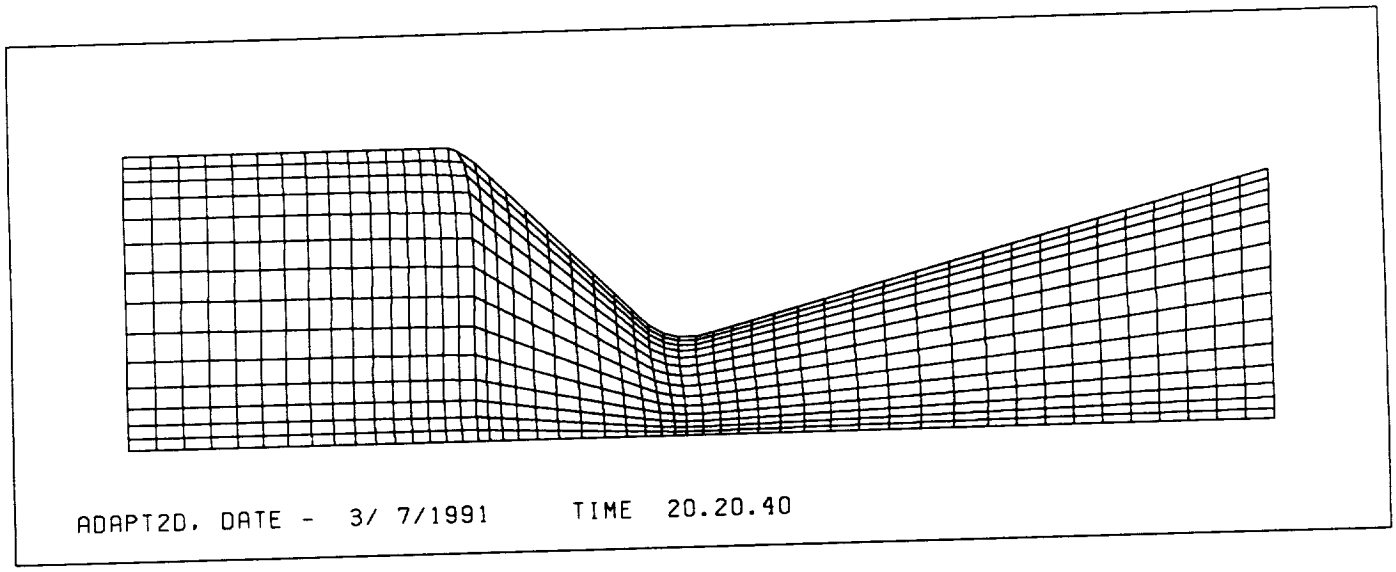


Figure 8.46: Grid distribution after 1.5 time units of advancement.

1. Develop three–dimensional discrete models of the Navier-Stokes equations for compressible flow in domains with boundaries unergoing arbitrary motions, particularly with burning or receding surfaces where the burning rate is determined by local flow physics.

2. Develop a three–dimensional mesh generation package for moving boundary simulations.

3. Develop a grid visualization package for viewing moving grids in three dimensions.

4. Develop a complete three–dimensional anisotropic h-adaptive package including the data structure, refinement/unrefinement package, and directional error estimates.

5. Extend the current two–dimensional burning boundary algorithm to three dimensions.

6. Assemble the functional packages developed in Tasks 1 to 5 into user–friendly three–dimensional analysis and design package for modeling solid rocket motors.

7. Develop a set of code validation problems to be supported by experimental results supplied by MSFC (if available).

8. Develop complete user documentation.

Appendices

A Jacobians Due to Source Terms

Four groups of additional Jacobian matrices are required in the axisymmetric formulation.

1. Jacobian matrix due to inviscid source term, $\widehat{\boldsymbol{B}}$, is defined as

$$\begin{aligned}\widehat{\boldsymbol{B}} &= \frac{\partial \boldsymbol{S}^c}{\partial \hat{\boldsymbol{u}}} \\ &= \frac{1}{y}\boldsymbol{B} \\ \boldsymbol{B} &= \overline{\gamma}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -q^2/2 & u & v & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}\end{aligned}$$

   where

   $\overline{\gamma} = \gamma - 1$, $q^2 = u^2 + v^2$

   and γ is the ratio of specific heats.

2. Jacobian matrix due to viscous source terms, $\widehat{\boldsymbol{T}}$, is defined as

$$\widehat{\boldsymbol{T}} = \frac{\partial \boldsymbol{S}^v}{\partial \hat{\boldsymbol{u}}} = \frac{1}{y}\boldsymbol{T} = \frac{1}{y}\left(\boldsymbol{T}_1 + \frac{1}{y}\boldsymbol{T}_2\right)$$

   where

$$\boldsymbol{T}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \dfrac{\lambda}{\rho^2}\dfrac{\partial \rho u}{\partial x} - \dfrac{2\lambda \rho u}{\rho^3}\dfrac{\partial \rho}{\partial x} & & & \\ + \dfrac{\lambda}{\rho^2}\dfrac{\partial \rho v}{\partial y} - \dfrac{2\lambda \rho v}{\rho^3}\dfrac{\partial \rho}{\partial y} & \dfrac{\lambda}{\rho^2}\dfrac{\partial \rho}{\partial x} & \dfrac{\lambda}{\rho^2}\dfrac{\partial \rho}{\partial y} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$T_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{2\mu\rho v}{\rho^2} & 0 & \frac{-2\mu}{\rho} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

3. Jacobian matrices due to viscous source terms, $\widehat{Q}$, are defined as

$$\widehat{Q}_1 = \frac{\partial S^v}{\partial \hat{u}_{,1}} = \frac{1}{y} Q_1$$

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{\lambda\rho u}{\rho^2} & \frac{-\lambda}{\rho} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\widehat{Q}_2 = \frac{\partial S^v}{\partial \hat{u}_{,2}} = \frac{1}{y} Q_2$$

$$Q_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{\lambda\rho v}{\rho^2} & 0 & \frac{-\lambda}{\rho} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

4. The viscous Jacobians, $\boldsymbol{P}^i = \dfrac{\partial \widehat{\boldsymbol{F}}_i^v}{\partial \boldsymbol{u}}$, are defined as

$$\boldsymbol{P}^1 = \frac{\partial \widehat{\boldsymbol{F}}_1^v}{\partial \boldsymbol{u}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ P_{21}^1 & P_{22}^1 & P_{23}^1 & 0 \\ P_{31}^1 & P_{32}^1 & P_{33}^1 & 0 \\ P_{41}^1 & P_{42}^1 & P_{43}^1 & P_{44}^1 \end{bmatrix}$$

where the matrix components of $\boldsymbol{P}^1$ are given by

$$P_{21}^1 = \frac{1}{\rho^2}\left(-\mu_R m_{1,1} - \lambda m_{2,2} + 2\mu_R \frac{m_1 \rho_{,1}}{\rho} + 2\lambda \frac{m_2 \rho_{,2}}{\rho}\right)$$

$$P_{22}^1 = -\frac{\mu_R}{\rho^2}\rho_{,1}$$

$$P_{23}^1 = -\frac{\lambda}{\rho^2}\frac{\partial \rho}{\partial y}$$

$$P_{31}^1 = -\frac{\mu}{\rho^2}\left(-m_{2,1} - m_{1,2} + 2m_1 \frac{\rho_{,2}}{\rho} + 2m_2 \frac{\rho_{,1}}{\rho} + \frac{m_1}{y}\right)$$

$$P_{32}^1 = -\frac{\mu}{\rho}\left(\frac{\rho_{,2}}{\rho} + \frac{1}{y}\right)$$

$$P_{33}^1 = -\frac{\mu}{\rho^2}\rho_{,1}$$

$$P_{41}^1 = uP_{21}^1 = uP_{21}^1 + vP_{31}^1 - \frac{1}{\rho^2}\left(m_1 \tau_{11} + m_2 \tau_{21}\right)$$

$$+ \frac{k}{\rho^2 c_v}\left[-(\rho e)_{,1} + 2u_1 m_{1,1} + 2u_2 m_{2,1} + \left(2e - 3u_1^2 - 3u_2^2\right)\rho_{,1}\right]$$

$$P_{42}^1 = \frac{\tau_{11}}{\rho} + uP_{22}^1 + vP_{32}^1 + \frac{k}{\rho^2 c_v}\left(-m_{1,1} + 2u_1 \rho_{,1}\right)$$

$$P_{43}^1 = \frac{\tau_{12}}{\rho} + uP_{23}^1 + vP_{33}^1 + \frac{k}{\rho^2 c_v}\left(-m_{2,1} + 2u_2 \rho_{,1}\right)$$

$$P_{44}^1 = -\frac{k}{\rho^2 c_v}\rho_{,1}$$

Setting $i = 2$ in the definition above for the viscous Jacobians gives:

$$\boldsymbol{P}^2 = \frac{\partial \widehat{\boldsymbol{F}}_2^v}{\partial \boldsymbol{u}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ P_{21}^2 & P_{22}^2 & P_{23}^2 & 0 \\ P_{31}^2 & P_{32}^2 & P_{33}^2 & 0 \\ P_{41}^2 & P_{42}^2 & P_{43}^2 & P_{44}^2 \end{bmatrix}$$

where the matrix components are defined by

$$\begin{aligned}
P_{21}^2 &= P_{31}^1 \\
P_{22}^2 &= P_{32}^1 \\
P_{23}^2 &= P_{33}^1 \\
P_{31}^2 &= \frac{1}{\rho^2}\left(-u_R m_{,2} + 2\mu_R u_2 \rho_{,2} + 2\frac{\mu m_2}{y}\right) \\
P_{32}^2 &= -\frac{\lambda}{\rho^2}\rho_{,1} \\
P_{33}^2 &= -\frac{\mu_R}{\rho^2}\rho_{,2} - \frac{2\mu}{\rho y} \\
P_{41}^2 &= -\frac{m_1}{\rho^2}\tau_{12} - \frac{m_2}{\rho}\tau_{22} + u_1 P_{21}^2 + u_2 P_{31}^2 \\
&+ \frac{k}{\rho^2 c_v}\left[-(\rho e)_{,2} + 2u_1 m_{1,2} + 2u_2 m_{2,2} + \left(2e - 3u_1^2 - 3u_2^2\right)\rho_{,2}\right] \\
&+ \frac{1}{y}\frac{k}{\rho^2 c_v}\left(\rho e - \rho u_1^2 - \rho u_2^2\right) \\
P_{42}^2 &= \frac{\tau_{12}}{\rho} + u_1 P_{22}^2 + u_2 P_{32}^2 + \frac{k}{\rho^2 c_v}\left(-m_{1,2} + 2u_1\rho_{,2} + \frac{m_1}{y}\right) \\
P_{43}^2 &= \frac{\tau_{22}}{\rho} + u_1 P_{23}^2 + u_2 P_{33}^2 + \frac{k}{\rho^2 c_v}\left(-m_{2,2} + 2u_2\rho_{,2} + \frac{m_2}{y}\right) \\
P_{44}^2 &= -\frac{k}{\rho^2 c_v}\rho_{,2} - \frac{1}{y}\frac{k}{\rho c_v}
\end{aligned}$$

B Methods for Treating Constrained or Hanging Nodes

Suppose that one is interested in performing element calculations for the element NEL shown shaded in Fig. B.1. The solution within this element can be expressed as

$$\begin{aligned} \boldsymbol{u}^e &= \sum_{I=1}^{4} \boldsymbol{u}_{\mathrm{NODES}(I,NEL)} \Psi_I \\ &= \boldsymbol{u}_1 \Psi_1 + \boldsymbol{u}_2 \Psi_2 + \boldsymbol{u}_3 \Psi_3 + \boldsymbol{u}_4 \Psi_4 \end{aligned} \tag{B.1}$$

where Ψ_I is the local shape function associated with node I, and $\boldsymbol{u}_i$ is the solution vector at node I. However, notice that nodes 1 and 3 are hanging nodes and the numerical solution at these two nodes are constrained by

$$\left.\begin{aligned} \boldsymbol{u}_1 &= \frac{\boldsymbol{u}_4 + \boldsymbol{u}_9}{2} \\ \boldsymbol{u}_3 &= \frac{\boldsymbol{u}_4 + \boldsymbol{u}_8}{2} \end{aligned}\right\} \tag{B.2}$$

Substituting (B.2) into (B.1), we have

$$\boldsymbol{u}^e = \left(\frac{\boldsymbol{u}_4 + \boldsymbol{u}_9}{2}\right) \Psi_1 + \boldsymbol{u}_2 \Psi_2 + \left(\frac{\boldsymbol{u}_4 + \boldsymbol{u}_8}{2}\right) \Psi_3 + \boldsymbol{u}_4 \Psi_4 \tag{B.3}$$

Rearranging terms, we obtain

$$\boldsymbol{u}^e = \boldsymbol{u}_9 \frac{\Psi_1}{2} + \boldsymbol{u}_2 \Psi_2 + \boldsymbol{u}_8 \frac{\Psi_3}{2} + \boldsymbol{u}_4 \left(\frac{\Psi_1}{2} + \frac{\Psi_3}{2} + \Psi_4\right) \tag{B.4}$$

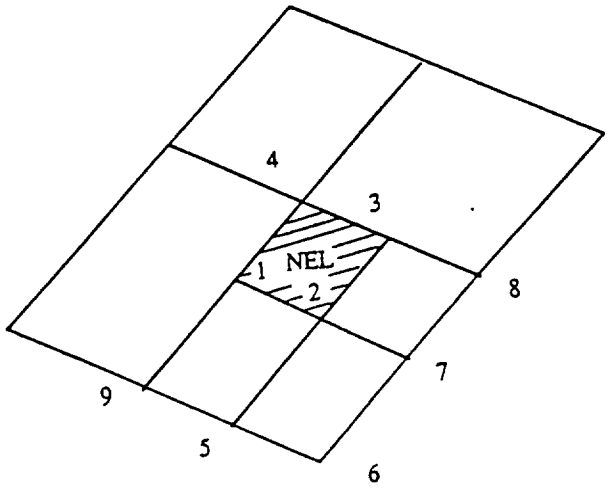


Figure B.1: Elements consisting of constrained nodes.

By defining new shape functions, Ψ'_i, as

$$\left.\begin{aligned} \Psi'_1 &= \frac{\Psi_1}{2} \\ \Psi'_2 &= \Psi_2 \\ \Psi'_3 &= \frac{\Psi_3}{2} \\ \Psi'_4 &= \left(\frac{\Psi_1}{2} + \frac{\Psi_3}{2} + \Psi_4\right) \end{aligned}\right\} \tag{B.5}$$

and defining (u_9, u_2, u_8, u_4) as the physical degrees of freedom associated with this element, then the solution vector within NEL can be interpreted as

$$u^e = \sum_{I=1}^{4} u_{\mathrm{NODES}(I,NEL)} \Psi'_I \tag{B.6}$$

where

NODES(1,NEL) = 9 , NODES(2,NEL) = 2
NODES(3,NEL) = 8 , NODES(4,NEL) = 4

and Ψ'_I are the modified shape function defined by (B.5). The method being applied to handle the constraint nodes for the Cartesian problems is based on equations (B.5) and (B.6). That is, one modifies the data structure such that element NEL was defined by the

physical degrees of freedom (nodes 9, 2, 8, 4) rather than its original geometrical degrees of freedom (nodes 1, 2, 3, 4). The local shape functions for these physical degrees of freedom have to be consistently modified according to equation (B.5).

In the axisymmetric case, the conservation variable $\hat{\boldsymbol{u}} = y\boldsymbol{u}$ was interpolated separately for y and $\boldsymbol{u}$. It is important to note that y is a purely geometrical quantity. The use of the modified shape functions to interpolate y within an element will cause a severe inconsistency for the conservation variable $\hat{\boldsymbol{u}}$. Therefore, y should be interpolated by using the original element shape function, that is

$$y^e = \sum_{i=1}^{4} y_i \Psi_i \tag{B.7}$$

Thus, equations (B.5), (B.6), and (B.7) completely define the method for treating the constrained nodes for the axisymmetric problem.

C Projection of Surface Nodes

The basic idea behind an adaptive methodology is that during the solution process, the grid is automatically refined and/or unrefined according to certain error criteria. These criteria are developed such that when the solution process is repeated using an updated grid the accuracy of the solution is improved.

Due to the refinement and unrefinement of the grid, many new nodes and elements are generated and some existing nodes are eliminated. This in general does not cause any difficulties if the new nodes are generated interior to the domain but may lead to problems if the nodes are adjacent to a prescribed boundary and no precautions are taken to ensure that the new grid points lie on the given surface. Obviously, it is necessary that as the refinement continues the resolution of the profile of the object must be improved as well as the accuracy of the solution. Without a precise interpolation or projection, these new nodes generated on the prescribed boundaries during the adaptive procedures can destroy the accuracy of the original geometrical representeation of a two–dimensional object and thus the solution.

If the surface of a two–dimensional object can be expressed by a few simple algebraic equations, an accurate interpolation or projection is easy to perform. Unfortunately, for most realistic problems the surfaces of a two–dimensional object are usually defined by a finite number of discrete points rather than a set of continuous algebraic functions. Thus we are faced with the problem, how does one accurately and efficiently project nodal points onto the surface of the two–dimensional object defined by a finite number of discrete points. One popular method used by the CAD/CAM industry [40] which we will also employ is described below.

The Parametric Cubic (PC) Curve in Space

The algebraic form of a PC space curve can be expressed as

$$\boldsymbol{r}(u) = \sum_{i=0}^{3} a_i u^i \tag{C.1}$$

where $\boldsymbol{r} = (x, y, z)^T$, u is the parameter that characterizes the curve and we assume it varies from 0 to 1, and a_i are the constant coefficients.

Suppose that the end conditions $\boldsymbol{r}$ are given, that is

$$\left.\begin{aligned} \boldsymbol{r}(0) &= a_0 \\ \boldsymbol{r}(1) &= a_o + a_1 + a_2 + a_3 \\ \boldsymbol{r}'(0) &= a_1 \\ \boldsymbol{r}'(1) &= 3a_3 + 2a_2 + a_1 \end{aligned}\right\} \tag{C.2}$$

or in matrix notation

$$\begin{bmatrix} \boldsymbol{r}(0) \\ \boldsymbol{r}(1) \\ \boldsymbol{r}'(0) \\ \boldsymbol{r}'(1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} \tag{C.3}$$

Solving (C.3) for a_i yields

$$\begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = (M) \begin{bmatrix} \boldsymbol{r}(0) \\ \boldsymbol{r}(1) \\ \boldsymbol{r}'(0) \\ \boldsymbol{r}'(1) \end{bmatrix} \tag{C.4}$$

where

$$M = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{C.5}$$

Substituting (C.4) into (C.1) and rearranging, we obtain the geometric form of $\boldsymbol{r}$

$$\boldsymbol{r}(u) = \boldsymbol{r}(0)b_1(u) + \boldsymbol{r}(1)b_2(u) + \boldsymbol{r}'(0)b_3(u) + \boldsymbol{r}'(1)b_4(u) \tag{C.6}$$

where

$$\left.\begin{aligned} b_1(u) &= 2u^3 - 3u^2 + 1 \\ b_2(u) &= -2u^3 + 3u^2 \\ b_3(u) &= u^3 - 2u^2 + u \\ b_4(u) &= u^3 - u^2 \end{aligned}\right\} \tag{C.7}$$

are the so–called *blending* functions. In matrix notation, equation (C.6) can be written as

$$\boldsymbol{r}(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} (M) \begin{bmatrix} \boldsymbol{r}(0) \\ \boldsymbol{r}(1) \\ \boldsymbol{r}'(0) \\ \boldsymbol{r}'(1) \end{bmatrix} \tag{C.8}$$

Replacing $\boldsymbol{r}$ by x, y, and z in (C.8), the full representation of a PC space curve becomes

$$\begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} (M) \begin{bmatrix} x(0) & y(0) & z(0) \\ x(1) & y(1) & z(1) \\ x'(0) & y'(0) & z'(0) \\ x'(1) & y'(1) & z'(1) \end{bmatrix} \tag{C.9}$$

Note that the PC plane curve is the special case of (C.9) by neglecting the z-coordinate.

Segmenting a PC Curve

In most practical engineering applications, PC curves are usually defined in a piecewise manner. Assuming that the end values of a segmented PC curve are given by u_1 and u_2, then the geometric coefficients of the segmented curve in terms of the given curve are modified as

$$\begin{bmatrix} \boldsymbol{R}(0) \\ \boldsymbol{R}(1) \\ \boldsymbol{R}'(0) \\ \boldsymbol{R}'(1) \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}(u_1) \\ \boldsymbol{r}(u_2) \\ \Delta u \boldsymbol{r}'(u_1) \\ \Delta u \boldsymbol{r}'(u_2) \end{bmatrix} \tag{C.10}$$

where $\Delta u = u_2 - u_1$ and $\boldsymbol{R}$ denotes the x, y, and z coordinates for the segmented PC curve.

Projection of Surface Nodes

Usually, the arc length coordinate is commonly used as the parameter that characterizes a PC curve. This implies that those points to be supplied to construct a PC curve have been pre-sorted in certain order such that every point on the curve can be assigned a unique arc length coordinate. The element connectivity array as explained in Section 6 can be used to

identify if a node lies on the prescribed surface. For a point to be projected onto a PC curve, the method we applied is first to find the closest segment in the PC curve to the point. Once this segment was identified, the projection of the surface node is equivalent to the following statement:

For a given point with Cartesian coordinate $\boldsymbol{r} = (x, y)$, find its arc length coordinate, s, and associated Cartesian coordinate $\boldsymbol{r}(s)$ such that

$$d = \|\boldsymbol{r}(s) - \boldsymbol{r}\| \tag{C.11}$$

is a minimum, where

$$\boldsymbol{r}(s) = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} \boldsymbol{M} \begin{bmatrix} \boldsymbol{r}(s1) \\ \boldsymbol{r}(s2) \\ \Delta s\boldsymbol{r}(s1) \\ \Delta s\boldsymbol{r}(s2) \end{bmatrix} \tag{C.12}$$

$$\Delta s = s2 - s1 \tag{C.13}$$

and $\boldsymbol{M}$ is given by equation (C.5).

Obviously, to minimize d is equivalent to find the zero–root of a fifth order polynomial $f(s)$ defined as

$$f(s) = [\boldsymbol{r}(s) - \boldsymbol{r}] \cdot \boldsymbol{r}'(s) \tag{C.14}$$

Newton-Ralphson's iterative method may be applied to determine its zero–root as follows:

1. Use $s = 0.5(s1 + s2)$ as an initial guess.
2. Calculate $f(s)$ and $f'(s)$, where $f'(s) = [\boldsymbol{r}(s) - \boldsymbol{r}] \cdot \boldsymbol{r}''(s) + \boldsymbol{r}'(s) \cdot \boldsymbol{r}'(s)$
3. Calculate $\delta = -\dfrac{f(s)}{f'(s)}$
4. Update $s = s + \delta$
5. Check if $|\delta| <$ tolerance,
   if yes, root $= s$ and stop
   if no, go to step 2.

By knowing the arc length coordinate of an arbitrary point on the PC curve, its corresponding Cartesian coordinate can be easily determined by using equation (C.12).

D Interface With GAMMA2D

Presently, the interface with the SRB2D code is through a user supplied grid file which contains node definition, element definition, element connectivity and, possibly, some geometric data for defining user prescribed profiles such as a nozzle contour or the profile of an eroding pocket. These profiles are treated as parametric cubic splines so that ruled curves may be readily established to perform surface nodal projections (for grid refinement) and surface node redistribution (for remeshing), see Appendix C for details. COMCO has developed an in-house grid generator, GAMMA2D, to automatically generate this grid file which can be readily interfaced with SRB2D. For details see the GAMMA2D user's manual and the SRB2D user's manual.

E Summary of the Postprocessing Capabilities

For the sake of completeness of the final report, we summarize the current postprocessing capabilities of the code.

1. Pointwise or nodewise aerodynamic data extraction.

   For the purpose of engineering analysis, it is highly desirable that the CFD codes can provide quantitative information about the flowfield. Nodewise data extraction allows a user to extract desired flow prooperties from the computed solution for a given point or a given boundary in the computational domain. These aerodynamic data include all conservation variables, all primitive variables, pressure coefficient, Mach number, shear stresses, entropy change, total enthalpy change, total energy loss, etc.

2. Image of the initial grid and adapted grid with an option to show a section of the computational domain..

3. Contour plots.

   Contour plotting of flow variables is a very popular method to show the distribution of flow properties throughout the computational domain. The code is capable of plotting the contours for the following flow properties: all conservation variables, all primitive variables, Mach number, vorticity, entropy change, total enthalpy change, total energy loss, molecular viscosity, eddy viscosity, turbulent length scale, etc.

4. Velocity vector plot and streamline plot.

   These plots are very useful to resolve complicated flow topologies such as vortex shedding, flow separation, flow reattachment, etc.

5. Total mass flux, total force, total heat transfer, total moment calculations.

To activate these postprocessing capabilities in the code, the user provides key words in semantic form in the input deck. These key words are usually accompanied with some user specified options. See the SRB2D user's manual for details.

References

1. Donea, J., Giuliani, S., and Halleux, J. P., "An Arbitrary Lagrangian-Eulerian Finite Element Method for Transient Dynamic Fluid–Structure Interactions," *Computer Methods in Applied Mechanics and Engineering*, **33**, pp. 689–723, 1982.

2. Kreiss, H. O., "Stability Theory for Difference Approximations of Mixed Initial Boundary Value Problems, Part I," *Mathematics of Computation*, Vol. 22, pp. 703–714, 1968.

3. Strikwerda, J. C., "Initial Boundary Value Problems for Incompletely Parabolic Systems," **Ph.D. Thesis**, Mathematics Department, Stanford University, 1976.

4. Gustafsson, B., and Sudstrom, A., "Incompletely Parabolic Problems in Fluid Dynamics," *SIAM J. Appl. Math.*, Vol. 35, No. 2, pp. 343–357, September 1978.

5. Dutt, P., "Stable Boundary Conditions and Difference Schemes for Navier-Stokes Equations," *SIAM J. Numer. Anal.*, Vol. 25, No. 2, April 1988.

6. Rudy, D. H., and Strikwerda, J. C., "A Nonreflecting Outflow Boundary Condition for Subsonic Navier-Stokes Calculations," *Journal of Computational Physics*, **36**, pp. 55–70, 1980.

7. Chu, C. K., and Sereny, A., "Boundary Conditions in Finite Difference Fluid Dynamics Codes," *Journal of Computational Physics*, **15**, pp. 476–491, 1974.

8. Gustafsson, B., and Kreiss, H. O., "Boundary Conditions for Time–Dependent Problems With an Artificial Boundary," *Journal of Comutational Physics*, **30**, pp. 333–351, 1979.

9. **Numerical Boundary Condition Procedures**, NASA-CP-2201, 1981.

10. Yee, H. C., "Numerical Approximation of Boundary Conditions with Applications to Inviscid Equations of Gasdynamics," NASA-TM-81265.

11. Hesse, W. J., and Nicholas V. S. Mumford, **Jet Propulsion for Aerospace Applications**, 2nd ed., 1964.

12. Tezduyar, T. E., and Liu, J., **Domain Decomposition Methods for Partial Differential Equations**, R. Glowinski, et al. (eds.), SIAM, 1988.

13. Tezduyar, T. E., and Liu, J., **Recent Developments in Computational Fluid Dynamics**, T. E. Tezduyar and T. J. R. Hughes (eds.), ASME, 1988.

14. Tezduyar, T. E., Liu, J., Nguyen, T., and Poole, S., **Domain Decomposition Methods**, T. F. Chan, et al. (eds.), SIAM, 1989.

15. Tezduyar, T. E., and Liu, J., **Finite Element Analysis in Fluids**, T. J. Chung and G. R. Karr (eds.), University of Alabama, Huntsville Press, 1989.

16. Lerat, A., "Implicit Methods of Second-Order Accuracy for the Euler Equations," *AIAA J.*, **23**, pp. 33–40, 1985.

17. Hollander, H., Lerat, A., and Peyret, R., "Three-Dimensional Calculation of Transonic Viscous Flows by an Implicit Method," *AIAA J.*, **23**, pp. 1670–1678, 1985.

18. Oden, J. T., Strouboulis, T., and Devloo, P., *Computer Meth. in Appl. Mech. and Engrg.*, Vol 59 (3), 1986.

19. Oden, J. T., Strouboulis, T., and Devloo, P., *Intl. J. of Numer. Meth. in Fluids*, Vol. 7 (11), Nov. 1987.

20. Tworzydlo, W., Oden, J. T., and Thornton, E. A., "Adaptive Implicit/Explicit Finite Element Method for Compressible Viscous Flows," submitted for publication to *Comp. Meths. Appl. Mech. Engrg.*

21. Oden, J. T., and Bass, J. M., **Ninth Annual Conference on Computing Methods in Applied Sciences and Engineering**, Paris, France, Jan. 1990.

22. Oden, J. T., and Bass, J. M., "New Developments in Adaptive Methods for Computational Fluid Dynamics," in **Computing Methods in Applied Sciences and Engineering**, ed. by R. Glowinski and A. Lichnewsky, SIAM, Philadelphia, 1991.

23. Anderson, D. A., Tannehill, J. C., and Pletcher, R. H., **Computational Fluid Mechanics and Heat Transfer**, McGraw-Hill Book Co., New York, 1984.

24. Demkowicz, L., Oden, J. T., and Rachowicz, W., "A New Finite Element Method for Solving Compressible Navier-Stokes Equations Based on an Operator Splitting Method and *h-p* Adaptivity," *Comp. Meths. Appl. Mech. and Eng.*, to appear.

25. Hassan, O., Morgan, K., and Peraire, J., "An Adaptive Implicit/Explicit Finite Element Scheme for Compressible Viscous High Speed Flows," AIAA 27th Aerospace Sciences Meeting, Reno, Nevada, Jan. 9–12, 1989.

26. Lapidus, A., "A Detached Shock Calculation by Second–Order Finite Differences," *J. Comp. Physics*, **2** pp. 154–177, 1967.

27. Löhner, R., Morgan, K., and Peraire, J., "A Simple Extension to Multidimensional Problems of the Artificial Viscosity Due to Lapidus," *Com. Appl. Num. Meths.*, **1**, pp. 141–147, 1985.

28. Lo, S. H., "A New Mesh Generation Scheme for Arbitrary Planar Domains," *Int. J. Num. Meths. in Eng.*, Vol. 21, pp. 1403–1426, 1985.

29. Monson, D. J., Seegmiller, H. L., and McConnaughey, P. K., AIAA Paper 89-0275.

30. Rostand, P., "Algebraic Turbulence Models for the Computation of Two–Dimensional High–Speed Flows Using Unstructured Grids," *International Journal for Numerical Methods in Fluids*, Vol. 9, pp. 1121–1143, 1989.

31. Serra, R. A., *AIAA J.*, pp. 603–611, May 1972.

32. Cuffel, R. F., Back, L. H., and Massier, P. F., *AIAA J.*, Vol. 7, No. 7, 1969.

33. Ames Research Staff, "Equations, Tables, and Charts for Compressible Flow," NACA Report 1135, Moffett Field, CA, 1953.

34. Clift, R., Grace, J. R., and Weber, M. E., "Bubbles, Drops and Particles," Academic Press, New York, 1978.

35. Rimon, Y., and Chen, S. I., "Numerical Solution of a Uniform Flow Over a Sphere at Intermediate Reynolds Numbers," *The Physics of Fluids*, Vol. 12, No. 5, pp. 949–959.

36. Majumdar, A. K., Whitesides, R. H., and Jenkins, S. L., *J. Propulsion*, Vol. 6, No. 1, Jan–Feb., 1990.

37. Pao, R. H. F., **Fluid Dynamics**, Charles E. Merrill Books, Inc., 1967.

38. Sabnis, J. S., Gibeling, H. J., and McDonald, H., *J. of Propulsion*, Vol. 5, No. 6, Nov.–Dec., 1989.

39. Dunlap, R., et al., "Internal Flowfield Investigation," AFRPL-TR-86-104.

40. Peters, G. J., "Parametric Bi-Cubic Surface, in **Computer Aided Geometric Design**, Barnhill/Riesenfeld, eds., 1974.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
TR-91-05		

4. Title and Subtitle	5. Report Date
Final Report	March 15, 1991
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
C.Y. Huang, W. Tworzydlo, J.T. Oden, J.M. Bass, C. Cullen, S. Vadaketh	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
Computational Mechanics Co., Inc. 7701 N. Lamar, Suite 200 Austin, Texas 78752	NAS8-37682
	13. Type of Report and Period Covered Final Report March, 1990
12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
NASA Marshall Space Flight Center Huntsville, Alabama 35812	

15. Supplementary Notes

None

16. Abstract

This is the final report for a research project entitled "Solid Rocket Booster Internal Flow Analysis by Highly Accurate Adaptive Computational Methods" conducted for the NASA MSFC under Contract NAS8-37682. The primary objective of this project was to develop an adaptive finite element flow solver for simulating internal flows in the solid rocket booster. This report describes a unique flow-simulator code for analyzing highly complex flow phenomena in the SRB. New methodologies and features incorporated in this analysis tool are:

1) An adaptive implicit/explicit finite element method for solving both two-dimensional and axisymmetric Navier-Stokes equations..
2) Adaptive algorithms and the data management systems for h-adaptive methods.
3) Moving boundary and remeshing algorithms for simulating the erosive burning of a solid propellant.
4) Inclusion of an algebraic turbulence model.
5) Both two-dimensional and axisymmetric versions of finite element flow solvers, all operational and tested on representative flow problems. All of these topics are discussed in detail in this report.

17. key Words (Suggested by Author(s))	18. Distribution Statement
solid rocket booster, adaptive finite element methods, implicit/explicit schemes, moving boundary algorithms, porous wall with mass injection, turbulence model, adapted structured-unstructured grids.	Unclassified-Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
None	None		N/A