An Adaptive Least-Squares Finite Element Method for the Compressible Euler Equations

Farzad Taghaddosi

A Thesis

in

The Department

of

Mechanical Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

June 1996

ISBN 0-612-18448-X

Canada

Abstract

An Adaptive Least-Squares Finite Element Method for the

Compressible Euler Equations

Farzad Taghaddosi

A two-dimensional code is developed for the solution of the system of unsteady Euler equations in the primitive variables form, using the least squares finite element method. The unsteady terms are discretized in time using backward differences. The equations written in the form of a first-order system of PDEs, are then discretized in space by the least-squares method and are linearized with the Newton method.

The resulting system of linear algebraic equations, which is symmetric and positive definite, is solved using the Conjugate Gradient iterative method. The convergence properties of the iterative solver are improved by applying incomplete Cholesky and diagonal preconditioners. Moreover, by optimizing the user defined parameters of the iterative solver substantial reduction in the computational time has been achieved.

The performance of the least-squares method in capturing the shocks is demonstrated through three supersonic and transonic test problems. Since, the artificial viscosity mechanism naturally embedded in the least squares formulation is not able to sharply resolve the shocks, a moving-node mesh adaptation technique is employed to improve the quality of the solution. The error estimation of the adaptive method is based on the second derivatives of a given flow variable and is calculated along the edges of the elements. It is very sensitive to oriented structures like shock waves and, along with the moving-node scheme, will create adapted meshes where the element edge(s) are aligned with such directional flow phenomena. It is demonstrated that by using the adaptation, good results could be obtained even on rather coarse grids.

Acknowledgments

I would first like to thank my supervisors Professors W G Habashi and G. Guèvremont for their continuous guidance, support, and encouragement throughout this research. I am especially grateful for their positive and attentive approach whenever I encountered a difficulty during my work.

My special thanks goes to Mr. D. Ait-Ali-Yahia, who provided me with the grid adaptation code, and with whom I had many helpful discussions about adaptation. The help of Dr M G Vallet for providing the grid optimization library is greatly acknowledged.

My thanks are also due to Dr L Dutto, who kindly helped me in better understanding the solution techniques for linear algebraic equations by providing articles and books, and also through e-mails when she was in Ottawa

I would also like to thank my colleagues at the CFD Lab of Concordia University, who were always available to answer my questions and with whom I shared a friendly environment In particular, I would like to thank Mr C Lepage, Dr G Baruzzi, and Mr D Stanescu for their help and suggestions

This work was supported through a scholarship from the Ministry of Culture and Higher Education (MCHE) of the I R Iran, and under Operating Grant OGP 0171328 of the National Sciences and Engineering Research Council of Canada (NSERC) I would like to personally acknowledge them for their financial support

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The motion of a fluid continuum is governed by the full system of the Navier-Stokes (NS) equations. Due to the complex nature of these non-linear PDEs, analytical solutions are only available for very limited, and sometimes impractical, situations. In contrast, the numerical approach, through continuous development, can solve a much wider range of problems.

In practice, the full NS equations are usually simplified to less complex form, by introducing approximations and/or assumptions, before a numerical method is applied. These simplifications are sometimes essential due to the lack of scientific knowledge and/or computational resources as in the case of an exact analysis of a turbulent flow, and are sometimes dictated by logic because the flow can be accurately approximated using a simpler physical model. An example is the Euler equations which are obtained by neglecting viscous terms and heat conduction effects in the Navier-Stokes equations. It is a valid approximation for high-speed or convection-dominated flows, where viscous effects are confined to very thin regions close to the solid body (boundary layers), and the rest of the flow is essentially inviscid.

Consider the following model equation which contains the basic elements of the Navier-Stokes equations:

$$\frac{\partial \phi}{\partial t} + \vec{\Gamma} \cdot \nabla \phi = \nabla \cdot (k \nabla \phi) \tag{1.1}$$

where ϕ is the field variable, $\vec{V}$ is the velocity vector, and k is the diffusion coefficient. As advection dominates the flow, the relative effect of the diffusion term diminishes. In the limit, when $k = 0$, equation (1.1) reduces to the purely hyperbolic convective transport equation, which with $\phi = \vec{V}$ are the Euler equations. Therefore, the problems encountered in the numerical simulation of advection-dominated flows and inviscid flows are similar, and so is the solution to them. For this reason, many numerical schemes developed for the solution of the Navier-Stokes equations with dominant advection terms, have found their application in the solution of the Euler equations, and vice versa.

1.1 Finite Element Method for the Euler Equations

It is well known that the application of the standard Galerkin method, or any centered scheme, to convection-dominated flows leads to node to node spurious oscillations in the solution, known as wiggles. This happens when the grid Peclet number ($Pe = |\vec{V}| h / k$, where h is the characteristic length of the element), which represents the relative importance of convection to diffusion in the flow, exceeds two [26]. The problem arises from the fact that a central-difference type approximation to the convection term (first derivative), in the absence of diffusion (second derivatives), i.e. when $Pe = 1$, leads to equations where adjacent nodal values are decoupled, thus destabilizing the solution.

A straightforward way of overcoming this problem is to severely refine the mesh in the regions where oscillations occur, such that $h = Pe/|\vec{V}|$ and therefore, the convection no longer dominates on an element level. This is, however, impractical in high speed flows due to the very large number of nodes required. The other option is to use a different approximation for the convection term, which is consistent with the hyperbolic character of the flow. The methods developed based on this latter idea can be generally classified into three categories: upwind methods, artificial viscosity methods, and characteristic Galerkin methods.

1.1.1 Upwind Methods

Upwind methods are based on incorporating the directional nature of the convection phenomenon into the structure of the weight function. The first paper in this context was by Christie *et al.* [10], who built asymmetric piecewise linear and quadratic basis functions to create oscillation-free results for a 1-D model equation. The accuracy of their results was dependent on the way the weights were constructed. Heinrich *et al.* [19] later generalized this idea for 1-D and 2-D problems by adding a function of the form $\alpha_{ij}F(x)$ to the standard Galerkin weight, where $F(x)$ is a positive function, and α_{ij} is a parameter which controls the degree of asymmetry. As an alternative method of constructing an upwind effect, Hughes [23] proposed to relocate the quadrature point in the element during the numerical integration of the convection term. The position of this point was determined based on the element Peclet number, and was so selected to provide the proper upwind effect.

A major breakthrough in the application of the finite element schemes to this field was made when Brooks and Hughes [6] introduced the Streamline Upwind Petrov-Galerkin (SUPG) method. This method is based on adding a perturbation p to the standard Galerkin test function N, and applying the modified weight $w = N + p$ to all the terms in the advection-diffusion equation, leading to a consistent formulation. The added perturbation p creates an upwind effect by introducing artificial viscosity in the characteristic directions for 1-D systems and multi-dimensional scalar problems, and in averaged characteristic directions for truly multi-dimensional systems. The SUPG is a high-order method, which means that it does not sacrifice accuracy by adding artificial viscosity for the sake of stability.

Hughes and Tezduyar [24] generalized the SUPG method to first-order hyperbolic systems. In their initial work, several test cases were examined, and acceptable results were obtained. Later developments of the SUPG method mainly consisted of improving its shock capturing ability by adding non-linear operators to the perturbation [5, 25], and optimizing the free parameters in the perturbation p [52].

1.1.2 Artificial Viscosity Methods

Since the application of the Galerkin formulation to the hyperbolic equations leads to under-diffuse solutions, thus creating oscillations, it then seems logical to explicitly add an extra amount of artificial diffusion to balance these under-diffuse results. Kelly *et al.* [32] showed that the asymmetric weighting for the 1-D convection-diffusion model equation is equivalent to using the symmetric Galerkin weight and adding a diffusion term. The concept of balancing dissipation was then extended to multi-dimensions by the definition of an artificial diffusivity tensor. The necessity of defining a tensor was to apply the artificial dissipation only in the flow direction and eliminate the possibility of any cross-wind diffusion [6].

The artificial viscosity methods benefit from the use of the Galerkin formulation, which is easy to implement compared to the upwind methods, where asymmetric weighting complicates the algorithm. The accuracy and stability of the artificial viscosity methods, however, depend on the mechanism which controls the amount of added artificial dissipation. In the ideal case, this mechanism should be so constructed as to smooth out the local oscillations without globally polluting the entire solution. Löhner *et al.* [36] used the flux corrected-transport (FCT) method to obtain accurate solutions for high-speed compressible flows. In the FCT method, which is a two-step algorithm, an amount of artificial viscosity is first added, and the excess amount is then removed in a second step. This can also be interpreted as combining a high-order scheme in smooth regions of the flow with a lower-order method near discontinuities or in under-diffuse regions.

In another approach, Peraire *et al.* [45] used an explicit method and the Galerkin formulation to solve the unsteady Euler equations in 3-D. Their artificial dissipation model was based on smoothing the solution in the vicinity of discontinuities at the end of each time-step, using a pressure coefficient as a sensor to detect sharp gradients. Another way of adding artificial viscosity was proposed by Baruzzi *et al.* [3]. In their method, the Laplacian of dependent variables, i.e., ρ, u, and v was added to the continuity, $x-$momentum, and $y-$momentum equations, respectively. The amount of artificial viscosity was then con-

trolled by a single parameter as the coefficient of the Laplacians. They later extended this first-order artificial viscosity method to second-order [4].

1.1.3 Characteristic Galerkin Methods

The problems encountered in the numerical solution of hyperbolic equations governing convection-dominated flows are due to the presence of asymmetric or non-self-adjoint operators, i.e., first-order convection terms, for which the Galerkin method is not optimal. These asymmetric operators, however, when written along characteristic directions, gain the self-adjoint property, and therefore the application of the standard Galerkin method will provide the best approximation. This fact has led to the development of Characteristic Galerkin methods.

Löhner *et al.* [35] used this concept to develop an explicit algorithm for the solution of a system of hyperbolic equations. They tested different problems including flow in a nozzle and in a Riemann shock tube. In both cases, an additional artificial viscosity term was needed to stabilize the solution near the shocks.

To utilize the concept of the characteristics for the advection-diffusion equations, the equations are split into two parts: an advection equation which is treated by the characteristics method, and a diffusion equation which is treated by the standard Galerkin method. The solution of the advection equation is used as an initial guess for the diffusion equation [18, 34].

This split-operator approach was used by Zienkiewicz *et al.* [55], and Zienkiewicz and Wu [56] to solve a range of problems from incompressible flow to transonic and supersonic flow around airfoils and the cylinder. In their approach, the compressible Navier-Stokes equations were split into two parts. The first contained the pure transport terms in addition to the (nearly) self-adjoint diffusion terms, and the other equation contained the remaining term(s). An explicit method based on the characteristics was then used to solve the first system of equations. The second system, which retains the compressible terms, was solved in an implicit manner to avoid time-step limitations.

1.1.4 Other Methods

There also exists other schemes for the solution of the hyperbolic equations, which do not fit exactly into the above categories. One of these is the popular Taylor-Galerkin (TG) method proposed by Donea [12]. It uses the ideas behind the Lax-Wendroff finite difference scheme, in a finite element context, to stabilize the solution of the convective-transport equation. The scheme is third-order accurate in time and exhibits particularly high phase accuracy with minimal numerical damping. Löhner *et al.* [36] and Oden *et al.* [42] used the Taylor-Galerkin method for the solution of the Euler equations in the supersonic regime. The TG method, being a high-order method, produces oscillations at discontinuities, and therefore, both Löhner and Oden used a flux-corrected-transport approach to avoid non-physical oscillations in the solution.

Rice and Schnipke [51] proposed a direct approach for the solution of convection-dominated flows. In their method, the convection terms are transformed to streamline coordinates, where they become one-dimensional, i.e., $\rho u_s \frac{\partial \phi}{\partial s}$, and are assumed constant within each element. The governing equation is then discretized using the Galerkin method. The calculation of the convection term, however, requires the pattern of the streamline passing through one of the downstream corner nodes in the element. This pattern is found using a search algorithm based on the mass flow across the element boundaries. The method was extended to quadratic elements by Hill and Baskharone [20].

Another approach is based on the least-squares weighted residual method, where the L_2-norm of the governing equations residual is minimized with respect to the dependent variables. The least-squares method has very good stability properties due to its minimization nature, and has been applied for the solution of a variety of problems [30, 33, 53]. It is the method adopted in this thesis and will be discussed in detail in the next section.

The methods reviewed above use different approaches to tackle the problems involved in the numerical solution of hyperbolic equations. It can be shown, nevertheless, that many of them can be deduced from each other, or reduced to very similar or even identical algorithms when applied to model equations. For example, Löhner *et al.* [35] showed that the

Taylor-Galerkin method can be justified as a characteristic Galerkin method. As mentioned earlier, Kelly *et al.* [32], and Brooks and Hughes [6] showed the equivalence of the Streamline Upwind Petrov-Galerkin (SUPG) method and the balancing dissipation method for a 1-D convection-diffusion equation. A comparison between the least-squares method, the Taylor-Galerkin, and the Petrov-Galerkin methods is made by Carey and Jiang [9]. Further analysis of the similarity between the Taylor-Galerkin, the Petrov-Galerkin, and the characteristic Galerkin methods is carried out by Morton [38] and Comini *et al.* [11].

1.2 Least-Squares Methods

As one of the earliest efforts in this field, one can mention the technique presented by Polk and Lynn [48] for the solution of the unsteady expanding flow in a tube. They used triangular elements constructed in both space and time, with their base at t_0, their vertex at $t_0 + \Delta t$, and their sides corresponding to the characteristics: $dx = (x \pm c)\,dt$. The elements, therefore, can be considered as the domain of dependence for the calculation of the variables at vertex at $t_0 + \Delta t$. The first-order system was then discretized by the least-squares method. Another space-time finite element scheme was presented by Nguyen and Reynen [40], and was applied to the solution of convection-dominated problems in one- and two-dimensions. They showed that by extending the least-squares formulation to the time domain, highly accurate and stable results can be obtained up to very large grid Peclet numbers.

Fletcher [15] used the least-squares method to solve the Euler equations for subcritical compressible flows around the airfoil and cylinder. The special feature of his method was to represent groups of variables rather than single variables. By doing so, the linearity of the Euler equations was retained and lower-order numerical quadrature could be employed. Bruneau *et al.* [7] used a rather similar method to study the vortical phenomena created by the subsonic and supersonic flow over a flat plate at different angles of attack. They solved the system of Euler equations with the least-squares method, again by using group

variables.

Park and Liggett [44] combined the Taylor-Galerkin and the least-squares methods, and proposed a new scheme for the solution of the convection-dominated flows, called Taylor-Least-Squares (TLS). The method uses a third-order time approximation as in the Taylor-Galerkin method, but the semi-discretized equations are discretized in space by the least-squares method rather than the standard Galerkin approach. The appearance of the fourth-order spatial derivatives in the weak formulation, which reduce to second-order after integration by parts, necessitates the use of higher-degree polynomials, e.g., cubic Hermite shape functions. The method is accurate and can be extended to higher dimensions with little difficulty. The disadvantages are, however, that it is an explicit method with limitations on the time step, and further is computationally expensive due to the use of higher-degree shape functions.

Application of the least-squares method to a governing equation of the general form $\mathcal{L}(\phi) = f$ leads to the favorable result of a symmetric and positive-definite coefficient matrix, if $\mathcal{L}$ is a first-order differential operator. If $\mathcal{L}$ is a higher-order operator, however, this property is completely lost during the integration by parts, and moreover, elements with higher-order continuity requirements, e.g., C^1 must be employed.

Lynn and Arya [37] proposed to break down the high-order system to its first order counterpart as a way of eliminating this disadvantage. This idea, and the fact that almost all physical problems can be recast in the form of a first-order system, was used by Carey and Jiang [8] to formulate an algorithm for the general system of first-order PDEs using the least-squares finite element method.

An analysis of the method for the wave equation is performed by Carey and Jiang [9], including stability analysis, error estimation, and comparison with the Taylor-Galerkin method for simple 1-D test cases. To overcome the problem of non-linear instability, which occurs when a developing shock steepens, Jiang and Carey [28] proposed to minimize the H_1-norm of the residual rather than the L_2-norm. This will lead to the addition of an artificial viscosity that is proportional to the solution gradient, and will stabilize the

solution

Jiang and Carey [29] used the least-squares method for the solution of the 2-D compressible Euler equations, and studied several supersonic test cases with shocks. They also extended the H^1-residual method to 2-D in order to suppress the oscillations created when higher-order elements were used. Lefebvre *et al.* [33] applied a similar least-squares method for the compressible Euler equations. To stabilize the solution near discontinuities when quadratic elements were used, an amount of artificial viscosity (like the one employed in [45]) was explicitly added to the solution at the end of each time-step. They also adopted an adaptive refinement strategy based on the least-squares residual as the error estimator, in a similar fashion proposed by Jiang and Carey [27].

1.3 Present Method

Among different finite element methods used for the solution of the Euler equations, the least-squares method has been chosen in the present work due to its favorable features, including:

- ease of implementation.
- naturally produces artificial dissipation.
- contains no free parameter(s).
- stable with equal-order interpolation of variables.
- the resulting system of algebraic equations is symmetric and positive-definite.

The symmetry and positive-definite property of the coefficient matrix is one of the most important features of the least-squares method, since:

1. Only half the matrix needs to be stored, thus reducing the memory requirements by 50%. This is a very desirable advantage, especially for large-scale problems, where storage limitations are a major concern.

2. The resulting system can be solved very efficiently by the Conjugate Gradient (CG) iterative method. Application of other finite element methods, such as the Galerkin [3], or the SUPG [6], lead to nonsymmetric matrices. Efficient iterative methods used for the solution of nonsymmetric systems are variants of the optimal Conjugate Gradient method, attempting to imitate its properties [2]. Nevertheless, they are not as robust as the CG method in terms of convergence and stability properties, storage requirements, and CPU time. For example, the Generalized Minimum Residual (GMRES) method, which is considered one of the most efficient iterative solvers for nonsymmetric matrices, is *at least* two times slower and needs *at least* three times more memory compared to the CG method for solving the same number of equations [49, Chap. 3].

By applying preconditioning, the stability of the matrix iteration process is further enhanced, and the computation time is further reduced.

The artificial viscosity produced by the least-squares method does not allow discontinuities such as shock waves to be sharply resolved, unless an impractically fine grid is used. This is one of the main reasons why very little work has been done so far in applying the least-squares method to the Euler equations, despite its various obvious advantages [29, 33].

In the present work, a very robust adaptive method is used in order to improve the resolution near shocks. The employed directionally adaptive approach is a very recent and important development, which significantly improves the grid efficiency compared to traditional anisotropic methods [1]. It uses second derivatives of a given flow variable as the error estimator, compared to first derivatives used by most other adaptive methods. Since the variables vary linearly along each element edge, the truncation error is second-order. Employing second derivatives thus provides a more accurate measure for the approximation error.

The sensitivity of the error estimator to directional flow phenomena, such as shock waves, is achieved by calculating the error along the element edges rather than the com-

monly used approaches based on nodes or elements. When combined with the moving-node scheme, it leads to alignment of the element edges with such directional structures. As a result, the shocks will be very thin and very sharply captured.

The present work, therefore, is an attempt to investigate the least-squares finite element method for compressible flows in the transonic and supersonic regimes with shocks, and improve its performance and accuracy via an adaptive grid method, and a preconditioned Conjugate Gradient iterative solver.

1.4 Thesis Content

In Chapter 2, theoretical aspects of the least-squares method will be explained, along with its comparison with other methods. The development of the finite element algorithm for a general first-order system of PDEs, and subsequently, the Euler equations is followed. Description of different ways of applying the boundary conditions ends this chapter.

Chapter 3 gives a brief review of different adaptive strategies and, in particular, the directionally-adaptive moving-node scheme used in our calculations.

Chapter 4 describes the solution method, including discussion about the choice of matrix solver, i.e., direct vs. iterative, different iterative solvers, preconditioning, and stopping criteria.

In Chapter 5, the numerical results for three test cases, including supersonic and transonic flows, are presented and the results are discussed.

In Chapter 6, the performance and robustness of the least-squares method for the compressible Euler equations is discussed, and conclusions are drawn.

Chapter 2

Least-Squares Finite Element Method Formulation

Before developing a least-squares finite element method (LSFEM) for a general system of first order partial differential equations in two dimensions and its implementation for the Euler equations, a simple model equation is first analyzed to discuss the properties of the least-squares method, and compare it with other methods.

2.1 Analysis of the Method

Consider the following scalar first-order equation in one dimension

$$\frac{\partial u}{\partial t} + A(u)\frac{\partial u}{\partial x} = 0 \tag{2.1}$$

Using backward differences in time to discretize the unsteady term, equation (2.1) can be approximated by .

$$\frac{u^{n+1} - u^n}{\Delta t} + A^n \frac{\partial u^{n+1}}{\partial x} = 0 \tag{2.2}$$

where superscript n denotes an evaluation at time t^n, $\Delta t = t_{n+1} - t_n$ is the time step, and A^n is the linearized coefficient based on the solution at the previous time level: $A(u^{n+1}) \approx A(u^n) = A^n$. This form of time discretization leads to a fully implicit method which is

unconditionally stable for all Courant numbers [9]. The effect of different Courant numbers (or different Δt 's) on the solution will be discussed in detail in section 2.1.2.

The least-squares method is based on minimizing the L_2-norm of the residual of the governing equation(s). For equation (2.2), the least-squares functional is:

$$I(u^{n+1}) = \int_{\Omega} R^2 d\Omega = \int \left(u^{n+1} - u^n + \Delta t A^n \frac{\partial u^{n+1}}{\partial x} \right)^2 dx \tag{2.3}$$

where R is the residual, and Ω is the solution domain. To minimize this functional, its variation with respect to u^{n+1} is set to zero

$$\delta I = \frac{\partial I}{\partial u^{n+1}} \delta u^{n+1} = 0 \tag{2.4}$$

Setting the test function $w = \delta u^{n+1}$, gives the weak form :

$$\int \left(u^{n+1} - u^n + \Delta t \theta^n \frac{\partial u^{n+1}}{\partial t} \right) \left(1 + \Delta t A^n \frac{\partial}{\partial x} \right) w \, dx = 0 \tag{2.5}$$

Equation (2.5) is the weighted residual statement of equation (2.2) with the weight function.

$$w = w + \Delta t \theta^n w_t \tag{2.6}$$

The least squares method can therefore be considered as a Petrov-Galerkin formulation

2.1.1 Least-Squares vs. SUPG

At this point a comparison can be made between the least-squares method and the Streamline Upwind Petrov-Galerkin (SUPG) method. The essence of the SUPG method, as mentioned in the previous chapter, is to create an upwind effect by directionally weighting the element upstream of a node more heavily than the downstream element (Fig. 2.1). The word "directionally" means that this asymmetric weighting is only employed in the flow direction, thus avoiding cross-wind diffusion in two and three dimensions when the flow is skewed to the mesh. The weight of the SUPG method in one-dimension is :

$$w_{\text{SUPG}} = N + \tau u \frac{\partial N}{\partial x} \tag{2.7}$$

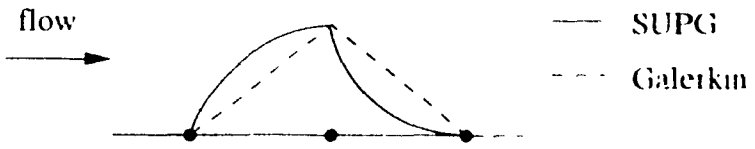

Figure 2.1: Galerkin and SUPG weighting functions

or in vector form:

$$w_{\text{SUPG}} = N + \tau \vec{u} \cdot \nabla N \tag{2.8}$$

where N is the shape function, and $\vec{u}$ is the velocity vector. The parameter τ has dimensions of time and is called the *intrinsic time scale*. It is generally a function of element size and local flow velocity and controls the amount of upwinding to be applied. As a result, its value highly affects the accuracy of the solution.

If the trial function in equation (2.5) is selected as the shape function, an expression similar to (2.8) will be obtained:

$$w_{\text{LS}} = N + \Delta t \, A^n \frac{\partial N}{\partial x} \tag{2.9}$$

For the case of the inviscid Burgers equation, where $A(u) = u$ in (2.1), the weight becomes

$$w_{\text{LS}} = N + \Delta t \, u \frac{\partial N}{\partial x} \tag{2.10}$$

or in vector form:

$$w_{\text{LS}} = N + \Delta t \, \vec{u} \cdot \nabla N \tag{2.11}$$

The inviscid Burgers equation and its general form, the inviscid transport equation, are the only cases where nearly identical weights for both methods are obtained.

Although the general form of the weights in the least-squares method (Eq. (2.9)) and the SUPG method (Eq. (2.8)) are different, they are analogous in some respect. For example, they are both proportional to the solution gradient and both produce asymmetric weighting. Moreover, A^n in multi-dimensions is a matrix, called the Jacobian matrix, that contains velocity components. Similarly, in multi-dimensional problems, τ is a matrix whose components are calculated using the Jacobian matrices [5, 24].

Despite these similarities there are some major differences between these two methods.

- The weight in the least-squares method is produced naturally as part of the formulation, while it has to be defined in the SUPG method.

- The least-squares method is a minimization problem and therefore has very good stability properties. For the SUPG method, however, stability depends on the amount of upwinding applied and, hence is not intrinsic as in the least-squares method.

- The coefficient matrices of the least-squares method are always symmetric and positive-definite, while they are nonsymmetric in the SUPG method.[1]

- There is no unique way of defining the key parameter τ, and different methods are proposed for its calculation [5, 52]. In contrast, the least-squares method contains no free parameters and leads to a rather general formulation for all kinds of flow

2.1.2 Artificial Viscosity

Some other features of the least-squares method can be demonstrated by deriving the Euler-Lagrange equation[2] corresponding to equation (2.5)

$$\frac{u^{n+1} - u^n}{\Delta t} + u^n \frac{\partial u^n}{\partial x} - \Delta t (u^n)^2 \frac{\partial^2 u^n}{\partial x^2} + (\Delta t u^n)^2 \frac{\partial^2}{\partial x^2}\left(\frac{u^{n+1} - u^n}{\Delta t}\right) \tag{2.12}$$

or

$$\frac{u^{n+1} - u^n}{\Delta t} + u^n \frac{\partial u^n}{\partial x} = \Delta t (u^n)^2 \frac{\partial^2 u^{n+1}}{\partial x^2} \tag{2.13}$$

In this equation, the left-hand side is an approximation to (2.1), and the right-hand side is the artificial viscosity. The artificial viscosity term

- is produced naturally as part of the least-squares formulation without any special treatment or weighting,

- is first-order,

[1] This is one of the most important advantages of the least-squares method, and will be discussed in Chapter 4.

[2] See Appendix A for the derivation.

- contains no free parameter(s),
- is directly proportional to the time step,
- contains the non-linear properties of the governing equation due to the presence of the non-linear coefficient A'', and
- indirectly depends on the solution due to the coefficient A''

It is clear from (2.13) that the only parameter which explicitly controls the amount of artificial viscosity in the flow is the time step. Very large or very small Δt's (or CFL numbers) will lead to diffusive or oscillatory solutions, respectively. The value of Δt directly affects the numerical accuracy and numerical stability of the solution. In equation (2.13), as Δt approaches zero, the numerical viscosity becomes negligible, and the equation resembles the Euler-Lagrange equation of the Galerkin formulation, which leads to the occurrence of oscillations.

To obtain accurate transient solutions, convergence should be obtained at each time level using, for example, a Newton method [22]. This is especially necessary in the present formulation, since it is only first-order accurate in time. If only the steady state solution is sought, one Newton iteration per time level suffices. In the problems with discontinuities (e.g. shock waves), the choice of Δt required for accurate steady state solutions is not very straightforward. In such cases, it is desired on one hand to use very small Δt's for better resolution of sharp layers, and on the other hand to use higher Δt's to avoid oscillations and reduce the overall computation time. The suitable time step will be therefore obtained based on trial and error.

2.1.3 Least-Squares vs. Taylor-Galerkin

It would also be useful to compare briefly the least-squares method with the well-known Taylor-Galerkin method [12]. This method is in fact an extension to finite elements of the ideas behind the Lax-Wendroff finite difference scheme. It uses higher-order time integration by including second and third time derivatives of the Taylor series expansion to

improve the stability properties of the Galerkin formulation for convective-transport equations.

For equation (2.1), with a linearized coefficient (i.e., $u_t + A^n u_x = 0$), the (forward-time or Euler) Taylor-Galerkin method gives the following equation:

$$\frac{u^{n+1} - u^n}{\Delta t} + A^n \frac{\partial u^n}{\partial x} = \frac{\Delta t (A^n)^2}{2} \frac{\partial^2 u^n}{\partial x^2} + \frac{(\Delta t A^n)^2}{6} \frac{\partial^2}{\partial x^2} \left(\frac{u^{n+1} - u^n}{\Delta t} \right) \tag{2.14}$$

which is different from equation (2.12) by only two numerical coefficients. This equation, when discretized in space using the Galerkin method, leads to stable solutions for convection-dominated flows. The improved stability properties are due to the first term on the right-hand side which resembles a numerical viscosity, and is obtained by substituting the second-order time derivatives in the Taylor series expansion by space derivatives through successive differentiation of $u_t + A^n u_x = 0$.

The Taylor-Galerkin method is an explicit time-accurate method, and is only conditionally stable compared to the unconditionally stable least-squares formulation. However, both methods contain no free parameters and have very good stability properties (due to their mechanisms of controlling the oscillations).

2.2 Least-Squares Method for First-Order Systems

Consider the following general system of first-order partial differential equations in two dimensions:

$$\mathbf{Lu} = \mathbf{f} \tag{2.15}$$

$$\mathbf{L} = \mathbf{A}_0 \frac{\partial}{\partial t} + \mathbf{A}_1 \frac{\partial}{\partial x} + \mathbf{A}_2 \frac{\partial}{\partial y} + \mathbf{A}$$

where $\mathbf{L}$ is a linear first-order differential operator, $\mathbf{u}^T = (u_1, u_2, \ldots, u_n)$ is the vector of unknowns, $\mathbf{f}$ is the given source function, $\mathbf{A}_0$ is the identity matrix (or its incomplete form), and $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}$ are $n \times n$ non-linear coefficient matrices. Since almost all physical problems can be recast mathematically in the form of (2.15), the developed least-squares method in the following can then be used to solve a variety of problems, by only changing

the coefficient matrices. This will allow to construct a rather general-purpose code. The method can also be easily extended to 3-D by adding the extra term $\mathbf{A}_3\,\partial/\partial z$ to (2.15), and making corresponding changes in the following formulation.

Discretizing the unsteady term in equation (2.15) with backward differences in time and linearizing the coefficient matrices using the Picard method, leads to:

$$\mathcal{L}\mathbf{u}^{n+1} = \mathbf{f}^n \tag{2.16}$$

$$\mathcal{L} = \mathbf{A}_1^n \frac{\partial}{\partial x} + \mathbf{A}_2^n \frac{\partial}{\partial y} + \left(\frac{\mathbf{A}_0}{\Delta t} + \mathbf{A}^n\right) \qquad ; \qquad \mathbf{f}^n = \frac{\mathbf{A}_0}{\Delta t}\mathbf{u}^n + \mathbf{f}$$

It should be noted that if another linearization method (e.g. Newton) were used, or the system were written in Δ-form (i.e., in terms of $\Delta\mathbf{u}^{n+1}$ rather than $\mathbf{u}^{n+1}$), the general form of (2.16) would remain the same. In such cases, an extra matrix or extra vectors would be added to the left- and right-hand sides of (2.16), respectively.

Defining the residual vector as $\mathbf{R} = \mathcal{L}\mathbf{u}^{n+1} - \mathbf{f}^n$, the least-squares functional will be:

$$\mathbf{I}(\mathbf{u}^{n+1}) = \int_\Omega \mathbf{R}^T\mathbf{R}\, d\Omega \tag{2.17}$$

Minimizing this functional by setting $\delta\mathbf{I} = \mathbf{0}$ and $\mathbf{w} = \delta\mathbf{u}^{n+1}$, where $\mathbf{w}$ is the weight function, gives the following weak statement:

$$\int_\Omega (\mathcal{L}\mathbf{w})^T(\mathcal{L}\mathbf{u}^{n+1} - \mathbf{f}^n)\, d\Omega = 0 \tag{2.18}$$

Since the least-squares method is a minimization problem, and therefore is not subject to the Ladyzhenskaya-Babuška-Brezzi (LBB) condition [17], equal-order interpolation can be used for all variables [30]. Introducing the finite element approximation:

$$\mathbf{u}^{n+1} \approx \mathbf{u}_h^{n+1} = \sum_{j=1}^{n_e} \mathbf{N}_j \mathbf{u}_j^{n+1} \tag{2.19}$$

where n_e is the number of nodes per element, and $\mathbf{N}_i = N_i\,\mathbf{I}$ is the element shape function, into the weak form (equation (2.18)) results in the linear algebraic equations:

$$[\mathbf{K}]\{\mathbf{U}\} = \{\mathbf{F}\} \tag{2.20}$$

where $\{\mathbf{U}\}$ is the global vector of unknowns, $[\mathbf{K}]$ is the global coefficient matrix, and $\{\mathbf{F}\}$ is the right-hand side vector. Matrix $[\mathbf{K}]$ and vector $\{\mathbf{F}\}$ are obtained by summing up the element matrices and vectors, respectively:

$$\mathbf{K}_{ij}^e = \int_{\Omega^e} (\mathcal{L}\mathbf{N}_i)^T (\mathcal{L}\mathbf{N}_j)\, d\Omega \qquad ; \qquad \mathbf{F}_i^e = \int_{\Omega^e} (\mathcal{L}\mathbf{N}_i)^T \mathbf{f}^n\, d\Omega \tag{2.21}$$

The expression for $\mathbf{K}_{ij}^e$ indicates that it is a symmetric and positive-definite (spd) matrix. This property is a characteristic of the least-squares method and does not depend on the form of the differential operator $\mathcal{L}$. The least-squares method, when applied to either symmetric or asymmetric differential operators (e.g., second- and first-order derivatives, respectively), will result in a symmetric, positive-definite matrix. In contrast, the Galerkin formulation leads to such a matrix only for symmetric (self-adjoint) differential operators. The importance of $[\mathbf{K}]^e$, and consequently $[\mathbf{K}]$, being symmetric and positive-definite is that very efficient iterative solvers can be used for the solution of equation (2.20). This will be discussed in more detail in Chapter 4.

2.3 System of the Euler Equations

The Euler equations which are the highest level of approximation for inviscid, non-heat conducting fluids, can be written in the following first-order system in terms of primitive variables[3]:

$$\mathbf{A}_0 \frac{\partial \mathbf{u}}{\partial t} + \mathbf{A}_1 \frac{\partial \mathbf{u}}{\partial x} + \mathbf{A}_2 \frac{\partial \mathbf{u}}{\partial y} = 0 \tag{2.22}$$

where $\mathbf{u}^T = (\rho, u, v, p)$, $\mathbf{A}_0 = \mathbf{I}$ (identity matrix), and

$$\mathbf{A}_1 = \begin{bmatrix} u & \rho & 0 & 0 \\ 0 & u & 0 & 1/\rho \\ 0 & 0 & u & 0 \\ 0 & \gamma p & 0 & u \end{bmatrix} \qquad ; \qquad \mathbf{A}_2 = \begin{bmatrix} v & 0 & \rho & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & v & 1/\rho \\ 0 & 0 & \gamma p & v \end{bmatrix}$$

[3]See Appendix B for the derivation of the energy equation in terms of pressure.

in which ρ is the density, (u, v) are the velocity components, p is the pressure, and γ is the specific heat ratio.

Since the coefficient matrices $\mathbf{A_1}$ and $\mathbf{A_2}$ are not constant, equation (2.22) is non-linear. The Newton method is applied to linearize the equations by setting $\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta\mathbf{u}$ in (2.22) and neglecting higher-order terms. After discretizing the unsteady term using backward differences, equation (2.22) can be written as:

$$\mathcal{L}\Delta\mathbf{u} = \mathbf{f}^n \tag{2.23}$$

$$\mathcal{L} = \mathbf{A}_1^n \frac{\partial}{\partial x} + \mathbf{A}_2^n \frac{\partial}{\partial y} + \left(\frac{\mathbf{A}_0}{\Delta t} + \mathbf{C}^n\right) \quad ; \quad \mathbf{f}^n = -\left(\mathbf{A}_1^n \frac{\partial \mathbf{u}^n}{\partial x} + \mathbf{A}_2^n \frac{\partial \mathbf{u}^n}{\partial y}\right)$$

$$\mathbf{C}^n = \begin{bmatrix} \left(\dfrac{\partial u^n}{\partial x} + \dfrac{\partial v^n}{\partial y}\right) & \dfrac{\partial \rho^n}{\partial x} & \dfrac{\partial \rho^n}{\partial y} & 0 \\ \dfrac{1}{\rho^n}\left(u^n \dfrac{\partial u^n}{\partial x} + v^n \dfrac{\partial u^n}{\partial y}\right) & \dfrac{\partial u^n}{\partial x} & \dfrac{\partial u^n}{\partial y} & 0 \\ \dfrac{1}{\rho^n}\left(u^n \dfrac{\partial v^n}{\partial x} + v^n \dfrac{\partial v^n}{\partial y}\right) & \dfrac{\partial v^n}{\partial x} & \dfrac{\partial v^n}{\partial y} & 0 \\ 0 & \dfrac{\partial p^n}{\partial x} & \dfrac{\partial p^n}{\partial y} & \gamma\left(\dfrac{\partial u^n}{\partial x} + \dfrac{\partial v^n}{\partial y}\right) \end{bmatrix}$$

The semi-discrete system (2.23) has the general form of (2.1). Equation (2.21) can therefore be used to calculate the element coefficient matrices and vectors with the differential operator $\mathcal{L}$ defined in (2.23). The integrals in (2.21) are numerically evaluated using Gauss-Legendre quadrature. For linear elements, the integrands in (2.21) are at most quadratic polynomials for undistorted elements, and can be exactly integrated using 2×2 Gauss points.

For the Euler equations, Eq. (2.23), the numerical viscosity inherent in the least-squares formulation can be demonstrated as follows. The right-hand side vector of (2.23) as defined in (2.21) is:

$$\mathbf{F}_i^e = \int_{\Omega^e} (\mathcal{L}\mathbf{N}_i)^T \mathbf{f}^n \, d\Omega$$

$$= -\int_{\Omega^e} \left[\mathbf{A}_1^n \frac{\partial \mathbf{N}_i}{\partial x} + \mathbf{A}_2^n \frac{\partial \mathbf{N}_i}{\partial y} + \left(\frac{\mathbf{A}_0}{\Delta t} + \mathbf{C}^n \right) \mathbf{N}_i \right]^T \left(\mathbf{A}_1^n \frac{\partial \mathbf{u}^n}{\partial x} + \mathbf{A}_2^n \frac{\partial \mathbf{u}^n}{\partial y} \right) d\Omega$$

$$= -\int_{\Omega^e} \left(\mathbf{A}_1^n \frac{\partial \mathbf{N}_i}{\partial x} + \mathbf{A}_2^n \frac{\partial \mathbf{N}_i}{\partial y} \right)^T \left(\mathbf{A}_1^n \frac{\partial \mathbf{u}^n}{\partial x} + \mathbf{A}_2^n \frac{\partial \mathbf{u}^n}{\partial y} \right) d\Omega - \int_{\Omega^e} \cdots \qquad (2.24)$$

After integrating this equation by parts, the first term on the right-hand side yields the following term in the associated Euler-Lagrange equation:

$$\int_{\Omega^e} \left(\mathbf{A}_1^n \frac{\partial}{\partial x} + \mathbf{A}_2^n \frac{\partial}{\partial y} \right)^T \left(\mathbf{A}_1^n \frac{\partial \mathbf{u}^n}{\partial x} + \mathbf{A}_2^n \frac{\partial \mathbf{u}^n}{\partial y} \right) d\Omega \qquad (2.25)$$

which represents the "natural" numerical viscosity of the least-squares method.

As the steady-state is reached by time-marching, the time-dependent term in (2.22), and consequently the right-hand side of (2.23), which is in fact the residual, vanishes. For this case, $\mathbf{u}^{n+1} = \mathbf{u}^n = \mathbf{u}$, and the least-squares weak form will become:

$$\left(\left(\frac{\mathbf{A}_0}{\Delta t} + \mathbf{C} \right) \mathbf{w}, \mathbf{A}_1 \frac{\partial \mathbf{u}}{\partial x} + \mathbf{A}_2 \frac{\partial \mathbf{u}}{\partial y} \right) + \left(\mathbf{A}_1 \frac{\partial \mathbf{w}}{\partial x} + \mathbf{A}_2 \frac{\partial \mathbf{w}}{\partial y}, \mathbf{A}_1 \frac{\partial \mathbf{u}}{\partial x} + \mathbf{A}_2 \frac{\partial \mathbf{u}}{\partial y} \right) = 0 \qquad (2.26)$$

which indicates that the steady-state solution depends on Δt. This is not desirable, as it will restrict the choice of time-step when accurate steady-state solutions are sought

2.4 Boundary Conditions

The definition of the problem governed by the equation (2.23) will only be complete once appropriate initial and boundary conditions are specified. The boundary conditions should provide correct information about all dependent variables on the boundaries in a way compatible with the physical character of the governing equations.

For the Euler equations, this information can be obtained through characteristics – the lines along which information propagates throughout the flow. At a typical node P on the boundary (Fig. 2.2), the right-running characteristic C_- will transport information from inside of the domain towards the node, and will affect the values at P. The left-running characteristic C_+, however, brings information from outside of the domain to the inside, and can affect the flow in the computational domain. The number of (physical) boundary

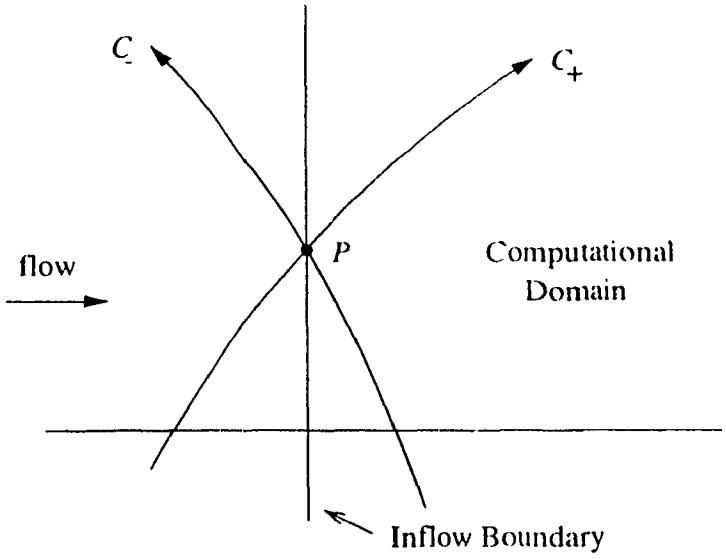

Figure 2.2: Characteristics at the boundary

conditions that can be imposed at a typical boundary should be equal to the number of *incoming* characteristics. The remaining variables at the boundary, which correspond to the number of *outgoing* characteristics, will be determined along with the solution.

Physical boundary conditions are imposed by explicitly specifying the dependent variable(s) at the boundary. In the finite element method, boundary values determined by outgoing characteristics are determined in a natural way through the interpolation functions embedded in the formulation. In the finite difference method, however, it is more difficult to determine these values. Among the different methods are discretizing the characteristics variables, using compatibility relations at the boundaries, or by extrapolation. These methods should be applied in a manner consistent with the internal scheme to keep the same order of accuracy and stability properties.

For the Euler equations (2.22), the number of incoming and outgoing characteristics are determined by the signs of the eigenvalues of the matrix K defined as:

$$K = \mathrm{A}_1 n_x + \mathrm{A}_2 n_y \tag{2.27}$$

in which (n_x, n_y) are components of the normal vector $\vec{n}$ on the given boundary, pointing towards the flow direction. In two dimensions, the eigenvalues of K are:

$$\vec{V}\cdot\vec{n}\ ,\ \vec{V}\cdot\vec{n}\ ,\ \vec{V}\cdot\vec{n}+c\ ,\ \vec{V}\cdot\vec{n}-c \tag{2.28}$$

where $\vec{V}$ is the velocity vector and c is the speed of sound. Depending on the flow velocity and its direction with respect to the boundary, each eigenvalue is either positive, negative, or zero. The number of physical and numerical boundary conditions are then determined by the number of positive and negative eigenvalues, respectively.

2.4.1 Inlet and Outlet Boundary Conditions

For subsonic and supersonic inlets and outlets, the number of physical boundary conditions to be specified are given in Table 2.1. It is common practice to specify density and velocity components at inlet and pressure at exit for subsonic inlet and exit boundaries, respectively [21, Chap. 19].

2.4.2 Solid Wall Boundary Condition

On a solid wall, the normal component of the velocity is zero, and only one of the eigenvalues in (2.28) is positive. The physical boundary condition applied for this case is: $\vec{V} \cdot \vec{n} = 0$ (i.e., no-penetration or flow tangency condition). In the Galerkin method, this condition is easily imposed by neglecting the boundary integral containing the $\vec{V} \cdot \vec{n}$ term. Baruzzi *et al.* [3] have neglected this term only in the continuity equation, while the integrals carrying this term in the momentum equations, which include density in their formulation, are calculated in order to determine the density at a solid boundary. In the least-squares method, however, such an integral does not exist and one of the following methods should be used.

Subsonic		Supersonic	
Inlet:	3 (ρ, u, v)	Inlet:	4 (ρ, u, v, p)
Outlet:	1 (p)	Outlet:	none

Table 2.1: Physical boundary conditions for 2-D Euler equations

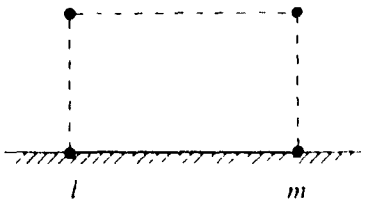

Figure 2.3: Local node numbers

Least-Squares Functional

In this method [33], the no-penetration boundary condition is weakly imposed by adding a least-squares functional to the original integral in (2.17). The new boundary functional, which does not affect the Euler-Lagrange equations, will be

$$\mathbf{I}(\mathbf{u}^{n+1}) = \int_{\Omega} \mathbf{R}^T \mathbf{R}\, d\Omega + w \int_{\Gamma} (\mathbf{V} \cdot \vec{n})^2\, d\Gamma \tag{2.29}$$

where Γ is the wall boundary, and w is the relative weight of the boundary functional with respect to the interior functional. This weight determines the relative importance of the two functionals, and should be of the order of $\mathcal{O}(10^3)$.

Minimizing the boundary functional with respect to the x component of velocity, and setting $N = \delta u$, gives the following for the x-momentum equation

$$w \int_{\Gamma} (un_x + vn_y) n_x N\, d\Gamma \tag{2.30}$$

This integral at the element level, after introducing the finite element approximation, can be written as

$$\underbrace{w \left[\int_{\Gamma^e} N_i N_j n_x^2\, d\Gamma \quad \int_{\Gamma^e} N_i N_j n_y n_x\, d\Gamma \right]}_{[K_x]} \begin{Bmatrix} u_j \\ v_j \end{Bmatrix} \quad i, j = l, m \tag{2.31}$$

where l and m are *local* node numbers of the element at the boundary (Fig. 2.3). Similarly, by minimizing the boundary functional with respect to the y-component of velocity, for the y-momentum equation the following is obtained:

$$\underbrace{w \left[\int_{\Gamma^e} N_i N_j n_x n_y\, d\Gamma \quad \int_{\Gamma^e} N_i N_j n_y^2\, d\Gamma \right]}_{[K_y]} \begin{Bmatrix} u_j \\ v_j \end{Bmatrix} \quad i, j = l, m \tag{2.32}$$

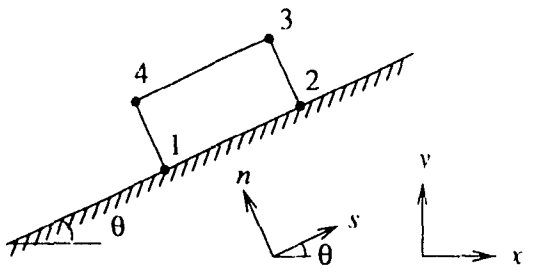

Figure 2.4: Coordinate rotation for wall elements

By assembling the coefficient matrices $[K_x]$ and $[K_y]$ in $[\mathbf{K}]^e$ (as defined in 2.21), and choosing a suitable weight for the functional, the no-penetration boundary condition can be discretely imposed on solid walls.

Rotated Coordinates

Another way of imposing the no-penetration boundary condition is to rotate the coordinate axes at the wall nodes from the *global* $x-y$ frame to the *local* $s-n$ (normal-tangential) coordinates [50, Chap 4]. In the new coordinate system, the flow tangency boundary condition is easily imposed by setting $V_n = 0$

Consider the element on the inclined boundary in Fig 2.4. For nodes 1 and 2, the local and global velocity components, $(\hat{u}, \hat{v})$ and (u, v) respectively, are related by:

$$\begin{Bmatrix} u_1 \\ v_1 \end{Bmatrix} = [R] \begin{Bmatrix} \hat{u}_1 \\ \hat{v}_1 \end{Bmatrix} \quad , \quad \begin{Bmatrix} u_2 \\ v_2 \end{Bmatrix} = [R] \begin{Bmatrix} u_2 \\ \hat{v}_2 \end{Bmatrix} \tag{2.33}$$

where

$$[R] = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

is the rotation matrix, and the angle θ is measured counter-clockwise from the $x-$axis to the $s-$axis. A similar relation can be written between the total vector of nodal variables in local and global coordinates:

$$\{\mathbf{u}^e\} = [\mathbf{R}^e]\{\hat{\mathbf{u}}^e\} \tag{2.34}$$

where

$$\{\mathbf{u}^e\}^T = (\rho_1\, u_1\, v_1\, p_1\ ,\ \rho_2\, u_2\, v_2 p_2\ ,\ \rho_3\, u_3\, v_3\, p_3\ ,\ \rho_4\, u_4\, v_4\, p_4)$$

$$\{\hat{\mathbf{u}}^e\}^T = (\rho_1\ \hat{u}_1\ v_1\ p_1\ ,\ \rho_2\ \hat{u}_2\ \hat{v}_2 p_2\ ,\ \rho_3\ u_3\ v_3\ p_3\ ,\ \rho_4\ u_4\ v_4\ p_4)$$

and $[\mathbf{R}^e]$ is:

$$[\mathbf{R}^e] = \left[\begin{array}{cccc|cccc|cccc|cccc}
1 & & & & & & & & & & & & & & & \\
 & c & -s & & & & & & & & & & & & & \\
 & s & c & & & & & & & & & & & & & \\
 & & & 1 & & & & & & & & & & & & \\
\hline
 & & & & 1 & & & & & & & & & & & \\
 & & & & & c & -s & & & & & & & & & \\
 & & & & & s & c & & & & & & & & & \\
 & & & & & & & 1 & & & & & & & & \\
\hline
 & & & & & & & & 1 & & & & & & & \\
 & & & & & & & & & 1 & & & & & & \\
 & & & & & & & & & & 1 & & & & & \\
 & & & & & & & & & & & 1 & & & & \\
\hline
 & & & & & & & & & & & & 1 & & & \\
 & & & & & & & & & & & & & 1 & & \\
 & & & & & & & & & & & & & & 1 & \\
 & & & & & & & & & & & & & & & 1
\end{array}\right]$$

$$c = \cos\theta \qquad s = \sin\theta$$

Substituting (2.34) into the element equations of system (2.20)

$$[\mathbf{K}^e]\{\mathbf{u}^e\} = \{\mathbf{F}^e\} \tag{2.35}$$

yields:

$$[\mathbf{K}^e][\mathbf{R}^e]\{\hat{\mathbf{u}}^e\} = \{\mathbf{F}^e\} \tag{2.36}$$

In these equations, the coefficient matrix has become nonsymmetric. To retain the symmetry and positive-definite properties of the least-squares formulation, both sides are premultiplied by $[\mathbf{R}^e]^T$. This gives the new element equations:

$$[\mathbf{K}^e]'\{\hat{\mathbf{u}}^e\} = \{\mathbf{F}^e\}' \tag{2.37}$$

where

$$[\mathbf{K}^e]' = [\mathbf{R}^e]^T[\mathbf{K}^e][\mathbf{R}^e] \qquad \text{and} \qquad \{\mathbf{F}^e\}' = [\mathbf{R}^e]^T\{\mathbf{F}^e\}$$

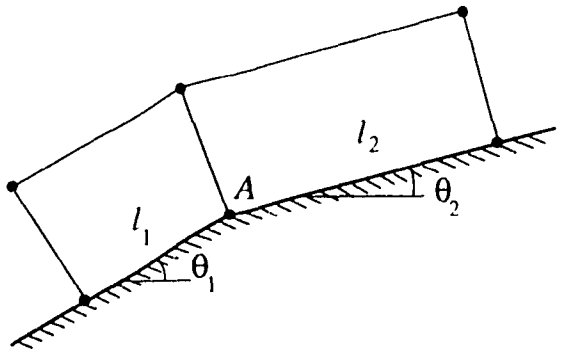

Figure 2.5: Discontinuous angle at wall nodes

In the transformed system (2.37), the no-penetration boundary condition is applied by setting: $\tilde{v}_1 = v_2 = 0$

When a curved boundary is approximated by finite elements, there is a discontinuity in the slope at a typical wall point A (Fig. 2.5), whether linear or higher-order elements are used. This discontinuity can become small by increasing the number of elements or using higher geometry approximation.

In order to apply the rotated coordinates method, each wall point should have a unique angle, so that the velocity vector $\vec{V}$ has a single value at that point. To accomplish this, a weighted average of the angles of the two adjacent wall edges can be used:

$$\theta_A = \frac{l_1 \theta_2 + l_2 \theta_1}{l_1 + l_2} \tag{2.38}$$

The angles θ_1 and θ_2 are multiplied by l_2 and l_1, since the element with the shorter edge will approximate $\vec{V}_A$ better than the element with the longer edge.

The velocity components in the coefficient matrices $\mathbf{A}_1, \mathbf{A}_2$ and $\mathbf{C}$ are in the global coordinate system. Before evaluating these matrices for the elements on a wall, the velocity components of the wall nodes must be counter-rotated from the local to the global coordinate system.

Chapter 3

Adaptive Procedure

3.1 Introduction

In the numerical solution of a well-posed problem, there exists a bounded error in the computed results due to the discretization of the partial differential equations, no matter which scheme is employed (finite element, finite difference, finite volume, etc). In the finite element method, this error can be attributed to three sources:

1. approximation error, which is due to the discrete approximation of the solution, e.g., by linear, quadratic, or other functions,

2. finite arithmetic and quadrature error, which are due to the computer round-off errors and the numerical evaluation of integrals,

3. error in approximating the geometry.

The error in approximating the geometry is mainly dependent on the complexity of the domain and the dimension of the problem, for example 2-D or 3-D. For simple or structured geometries, where the domain can be discretized accurately or exactly, this error can be highly reduced. For complex geometries, and especially for 3-D problems, the grid generation becomes an important issue because the domain approximation error could become significant.

By using an appropriate number of quadrature points, the integrals involved in a finite element discretization can be very accurately evaluated, thus reducing the quadrature error. This is not, however, always practical due to the computational cost of numerical integration. Since the finite arithmetic errors are mostly dependent on the computer hardware, there is less control over them. Nevertheless, it is still possible to reduce the error by using high-precision real variables, and in particular, optimizing the arithmetic operations in the code.

Compared to domain approximation and arithmetic errors, the approximation error is generally the largest. As a result, any attempt to reduce it will improve the accuracy of the results significantly. This error, which is measured in some norm, depends on how well the numerical method approximates the physical model. It can be reduced locally or globally by adapting the grid to the solution. This adaptive procedure which can be done in different ways, also requires information about the quality of the numerical approximation. Any adaptive method has therefore two basic ingredients:

1. Error estimate: In order to assess the quality of the solution throughout the domain, a criterion is needed. This criterion should reflect the error of the numerical approximation, which can be interpreted as the interpolation error. When the solution is approximated by piecewise linear functions, a norm of first- or second-order derivatives of the solution could provide such an error estimate [42, 46]. The estimated error distribution throughout the domain will then provide the required information to apply adaptation.

   The dependent variable whose error is the basis of the adaptation, is usually chosen depending on the flow regime. For inviscid compressible flows, the pressure or Mach number are the frequently used key variables. A norm of the residual of the equations can also be used as the error indicator [27, 33].

2. Adaptive strategy: The goal of adaptation is to improve the quality of the solution by equidistributing the error throughout the domain. This can be achieved using any

or a combination of the following methods:

h-method The analysis begins on a rather coarse mesh, which models the basic geometrical features of the domain. Once the solution is obtained, and the error estimated, the elements with a high level of error are subdivided into more elements of the same type. The refinement might also be coupled with a reverse process, leading to mesh coarsening. The *h*-method generally leads to isotropic meshes, and therefore, is well suited for those problems where the solution gradients are approximately equal in all directions. Major disadvantages of this method are an increased number of unknowns, which is more problematic in 3-D, and a change of node connectivity, which will increase the band-width of the coefficient matrix.

p-method In this approach, existing elements having a large error are replaced by higher-order elements. Use of higher-order polynomials to approximate the solution will then reduce the interpolation error. Despite this favorable property, the *p*-method has the disadvantage of being computationally more expensive. Moreover, constrained or transition elements must be employed to connect lower- and higher-order elements together, adding to the complexity of the algorithm.

r-method This strategy is based on moving the nodes in such a way that the newly oriented mesh has a more uniform error distribution. Node movement, as a result, leads to anisotropic meshes. The possibility of moving nodes allows forming elements whose edge(s) are aligned with an essentially 1-D flow phenomena, such as boundary layers and shocks. In this method node connectivity remains unchanged, since no new nodes are added. Therefore, a drawback is that the accuracy of the final results is limited by the structure and resolution of the initial grid.

- **remeshing** In this approach, as the name implies, the domain is remeshed based on the information obtained from the distribution of the error on the initial grid [46, 47]. As a result, the mesh is refined in some regions, and is coarsened in some others. The position of the nodes on the new mesh are generally completely different from that of the old grid. The main disadvantage of this method is that it is expensive, especially for 3-D problems. Local remeshing, however, can alleviate this problem to some extent, and make the procedure more efficient.

The adaptive strategies described above can also be combined together in an attempt to exploit the advantages of each individual method. Examples include h-p and h-r methods.

3.2 Directionally-Adaptive Approach

The adaptive method used in this thesis is based on the work of Ait-Ali-Yahia *et al.* [1], and uses a moving-node or r-method as the adaptive strategy. The important part, however, is the error estimation since it provides the information based on which the adaptation is done. The special feature of the present error estimator is its sensitivity to directional flow phenomena, such as shock waves. This is achieved by calculating the error on the element edges rather than the commonly used approaches based on nodes or elements. This will provide the moving-node mechanism with more accurate information about the distribution of error, leading to better alignment of the element edge(s) with such oriented structures. As a result, less elements would be required due to the efficient adaptation. Two basic steps of this adaptive procedure are explained in more detail in the following sections.

3.2.1 Edge-based error estimate

Consider the 1-D element shown in Fig 3.1, where the scalar variable σ is assumed to vary linearly within the element. A better approximation for this variable would be a quadratic function. In another words, the variable σ can be corrected by piecewise quadratic poly-

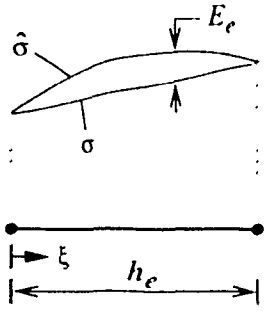

Figure 3.1: Approximation error in a 1-D element

nomials. Therefore, the error E can be estimated as the difference between the quadratic interpolation $\hat{\sigma}$ and the linear one.

After some simple arithmetic, and assuming that the nodal values of the two functions coincide, the approximation error can be expressed as:

$$E_e = \frac{\xi}{2}(h_e - \xi)\left.\frac{d^2\hat{\sigma}}{dx^2}\right|_e \tag{3.1}$$

where ξ is the local element coordinate, and h_e is the element length. Since the first term of the truncation error for a linear approximation is a second order derivative, equation (3.1) can provide a good estimate for the error. The measure of error in each element is then considered to be the root-mean-square value of E_e [46]:

$$E_e^{RMS} = \left(\int_0^{h_e} \frac{E_e^2}{h_e}\, d\xi\right)^{\frac{1}{2}} = \frac{1}{\sqrt{120}} h_e^2 \left|\frac{d^2\sigma}{dx^2}\right|_e \tag{3.2}$$

For an optimum mesh, where the error is equidistributed and is equal over each element, the following should hold for all the elements:

$$h_e^2 \left|\frac{d^2\hat{\sigma}}{dx^2}\right|_e = C \tag{3.3}$$

where C is a positive constant. The second derivative in equation (3.2) is based on $\hat{\sigma}$, which is the desired solution and therefore is not available. It is then approximated by the second derivative of the numerical solution, i.e., $\frac{d^2\sigma}{dx^2}$, by employing a recovery process based on the weighted residual method [1].

The above methodology is extended to 2-D based on the fact that the edge of a 2-D element can be considered as a 1-D element. The second derivative in equation (3.2) is

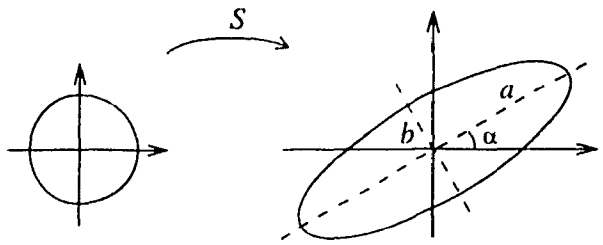

Figure 3.2: Transformation of a unit circle under S

replaced by the Hessian matrix:

$$H = \begin{bmatrix} \frac{\partial^2 \sigma}{\partial x^2} & \frac{\partial^2 \sigma}{\partial x \partial y} \\ \frac{\partial^2 \sigma}{\partial y \partial x} & \frac{\partial^2 \sigma}{\partial y^2} \end{bmatrix} \tag{3.4}$$

whose value is calculated for all the nodes throughout the domain. The Hessian matrix can be decomposed into the form:

$$H = R\,\Lambda\,R^T \tag{3.5}$$

where Λ is the diagonal matrix of eigenvalues, and R is the matrix of eigenvectors. The matrix R can be interpreted as a rotation by the angle α which the eigenvector corresponding to the smaller eigenvalue λ_1 makes with the x-axis.

Since the error should be positive, the Hessian matrix is reconstructed by replacing Λ with $|\Lambda|$ in (3.5):

$$\bar{H} = R|\Lambda|R^T \qquad \text{or} \qquad H = SS^T \tag{3.6}$$

where $S = R\sqrt{|\Lambda|}$ can be considered as a transformation, which when applied to an element stretches it in the direction of the principal axes of $\bar{H}$. This transformation applied to a unit circle is shown in Fig. 3.2, where the semi-major and -minor axes of the ellipse are:

$$a = 1/\sqrt{|\lambda_1|} \quad ; \quad b = 1/\sqrt{|\lambda_2|} \qquad |\lambda_2| > |\lambda_1|$$

Once $\bar{H}$ is calculated for all the nodes, with assumed linear variation throughout the domain, the error on each edge is obtained from:

$$e(\ell) = \int_{\xi_1}^{\xi_2} \sqrt{\mathbf{r}^T(\xi)\,\bar{H}(\xi)\,\mathbf{r}(\xi)}\,d\xi \tag{3.7}$$

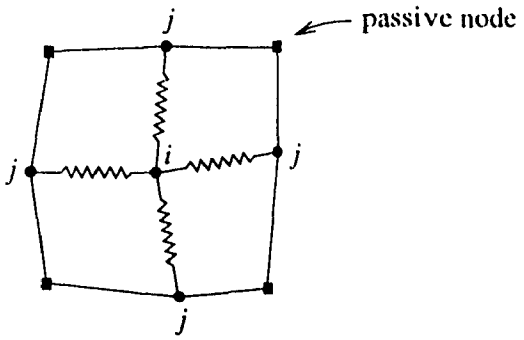

Figure 3.3: Local spring network corresponding to node i

where $\mathbf{r}(\xi)$ is the parametric representation of the edge ℓ. This edge-based error estimate, is then used as input for the moving-node mechanism to reposition the nodes, and accordingly, the elements.

It should be noted that although $e(\ell)$ is a scalar, it represents the error in the direction of the edge. The directional behavior of the flow features is therefore implicitly embedded in it.

3.2.2 Moving-node strategy

The adaptive mechanism used in this work for the equidistribution of error is a moving-node method based on a spring analogy. In this approach, the mesh is interpreted as a network of springs, where each edge is considered to be a fictitious spring with its stiffness representing the measure of the error (Fig. 3.3). The equilibrium of forces in this network will then determine the movement of each node. This idea was first introduced by Gnoffo [16]. Nakahashi and Diewert [39] later complemented his work by incorporating a grid-orthogonality constraint, and applied it in a finite difference context. In the finite element method, however, grid orthogonality is not essential, and hence as explained below, a simpler approach can be adopted.

In the spring network of Fig. 3.3, the new position of point i is determined by minimiz-

ing the potential energy of the system at that node:

$$P_i = \sum_j (\mathbf{x}_i - \mathbf{x}_j)^2 k_{ij} \tag{3.8}$$

where $\mathbf{x}$ is the position vector. The stiffness of the interconnected springs, k_{ij}, is defined as:

$$k_{ij} = \frac{e(\ell)}{\|\mathbf{x}_i - \mathbf{x}_j\|} = \frac{e(\mathbf{x}_i, \mathbf{x}_j)}{\|\mathbf{x}_i - \mathbf{x}_j\|} \tag{3.9}$$

in which $\|\cdot\|$ denotes the Euclidean norm. Minimizing (3.8) with respect to $\mathbf{x}_i$ gives the following equation:

$$\sum_j (\mathbf{x}_i^{m+1} - \mathbf{x}_j^{m+1})\, k_{ij}^{m+1} = 0 \tag{3.10}$$

which expresses the equilibrium of forces in the local spring network at the present adaptive iteration $m+1$. By lagging $\mathbf{x}_j$ and k_{ij} in equation (3.10), it can be written as:

$$\begin{aligned} 0 &= \sum_j (\mathbf{x}_i^{m+1} - \mathbf{x}_i^m + \mathbf{x}_i^m - \mathbf{x}_j^{m+1})\, k_{ij}^m \\ &= \sum_j \Delta\mathbf{x}_i\, k_{ij}^m + \sum_j (\mathbf{x}_i^m - \mathbf{x}_j^m)\, k_{ij}^m \\ &= \Delta\mathbf{x}_i \sum_j k_{ij}^m + \sum_j (\mathbf{x}_i^m - \mathbf{x}_j^m)\, k_{ij}^m \end{aligned}$$

or

$$\Delta\mathbf{x}_i = -\frac{\sum_j (\mathbf{x}_j^m - \mathbf{x}_i^m)\, k_{ij}^m}{\sum_j k_{ij}^m} \tag{3.11}$$

The new position of $\mathbf{x}_i$ is then calculated from:

$$\mathbf{x}_i^{m+1} = \mathbf{x}_i^m + \omega\, \Delta\mathbf{x}_i \tag{3.12}$$

where ω is a relaxation parameter.

The iteration process (3.12) is applied to all nodes in the domain in order to adapt the mesh to the solution. Boundary nodes can also move in the same way as internal nodes, but they are projected back on the boundary to restore the original shape of the domain. The moving-node scheme is applied to grid points in a sweeping manner. The reason is

to allow to check the quality of each newly oriented element during the mesh movement, and thus avoid formation of elements with a negative or nearly zero Jacobian. To obtain an appropriate adapted mesh, the number of adaptive iterations per adaptive cycle is chosen to be in the range of $m = 200 \sim 400$.

The adaptive method uses the solution of one of the scalar variables to adapt the mesh, and then interpolates the input data on the new mesh. The adapted mesh and the interpolated results are then used as initial data for the least-squares code to obtain a more accurate solution. Each mesh adaptation followed by the least-squares solution is called one adaptive cycle. In this work, the pressure is used as the key variable for adaptation.

Chapter 4

Solution Method

The solution of the system of linear algebraic equations:

$$A\,x = b \tag{4.1}$$

resulting from the discretization of the governing PDEs, is one of the most important parts of the numerical algorithm. The importance of this step becomes more evident by noting that it comprises over half of the overall computation time for large problems. The solution method should be robust enough to handle the usually ill-conditioned coefficient matrices encountered in computational fluid dynamics. Equation (4.1) can be solved by either of two different methods: direct or iterative.

4.1 Direct Methods

Direct methods are generally based on the factorization of the coefficient matrix A into two or more matrices having special structure. For example, in the Gaussian elimination A is factored into a lower and an upper triangular matrix, i.e., $A = LU$. Other examples are QR factorization (where Q is an orthogonal matrix and R is upper triangular), and LL^T or Cholesky factorization (for symmetric matrices). In the direct approach, the number of operations as well as the computation time is fixed for a particular matrix. Further, it has the advantage of being numerically stable (at least when pivoting is applied).

grid dimension	type of solver	exponent ν
2-D	iterative	1.021
2-D	direct	1.319
3-D	iterative	1.036
3-D	direct	1.645

Table 4.1: Empirical values for the storage requirement formula

For medium-sized linear systems, where direct and iterative methods both require almost the same amount of CPU time, the direct approach is usually preferred because of its reliability. For large sparse systems, however, direct methods are not suitable. The factorization of A creates nonzero elements, called fill-in, which requires additional memory. Pommerell [49, Chap. 2] has conducted an analysis of more than 200 large sparse systems (with more than 5000 unknowns) to investigate the memory requirements for direct and iterative linear solvers. His study has led to an empirical formula:

$$s = \mu n^{\nu} \tag{4.2}$$

where s is the amount of storage for a given number of unknowns n, μ is a coefficient, and the exponent ν is given in Table 4.1. It can be seen from this table that in passing from 2-D to 3-D problems, the storage requirement increases "exponentially" for direct solvers, while it still varies linearly for iterative solvers.

The number of operations required to perform factorization grows even faster than the memory requirements, leading to serious round-off errors and large CPU times. The large amount of storage requirements and timing for direct solvers, therefore, makes the solution procedure very expensive, and sometimes beyond available computer resources. As a result, iterative methods are commonly used for the solution of large sparse systems to significantly reduce the memory and CPU time requirements.

4.2 Iterative Methods

The essence of an iterative method is to generate a sequence of approximations $x_0, x_1, \ldots, x_k$ to the exact solution x, with the hope that each successive iterate x_k will be closer to x than the last. In basic iterative methods, the coefficient matrix A is split into two matrices:

$$A = M - N \tag{4.3}$$

where M is a nonsingular and easily invertible matrix. Substituting (4.3) into (4.1), a recurrence formula is obtained for the $(k+1)$th approximation:

$$x_{k+1} = Bx_k + c \tag{4.4}$$

where $B = M^{-1}N$ is called the *iteration matrix*, and $c = M^{-1}b$ is a vector. For example, choosing $M = D$ and $N = L + U$, where D, L, and U are the diagonal, and strictly lower and upper triangular matrices, respectively, leads to the *Jacobi* method.

The performance of the iteration defined by (4.4) basically depends on the spectral radius of B, defined as:

$$\rho(B) = \max|\lambda_i| \tag{4.5}$$

where λ_i is the set of eigenvalues of B. The convergence will be assured, if and only if $\rho(B) < 1$ [13]. Moreover, the smaller the spectral radius the faster the convergence to the exact solution. Based on these facts other methods have been constructed for splitting A, with the aim of a smaller spectral radius for the resulting iteration matrix. For example, if $M = D + L$ and $N = U$, the *Gauss-Seidel* method will be obtained, which performs better than the Jacobi method.

The convergence speed can be further improved by introducing a relaxation parameter ω, as in the *Successive Over-Relaxation* (SOR) method. The parameter is adjusted to make $\rho(B)$ as small as possible. Nevertheless, the optimization of ω in relaxation methods is not easy in general, and further it requires an estimate for the eigenvalues. Despite the improvements that can be obtained by relaxation or using different splitting techniques,

basic iterative methods (also known as matrix splitting methods) have generally slow convergence [2, Chap. 5]. Acceleration techniques such as the Steepest Descent and the Conjugate Gradient are therefore preferred. They are based on changing the structure of the iteration, rather than the iteration matrix, to increase the convergence speed.

4.3 Conjugate Gradient Method

The Conjugate Gradient (CG) method is the best choice for the solution of the linear system (4.1), if matrix A is symmetric and positive-definite (spd). In essence, it generates a set of linearly-independent orthogonal search directions in order to reach the exact solution x. The recurrence formula is of the form:

$$x_{k+1} = x_k + \alpha_k d_k \tag{4.6}$$

where α_k and d_k are the step size and search direction, respectively. In exact arithmetic, the CG method converges to x in n steps, where n is the dimension of A. In practice, however, convergence to at most machine accuracy (and most often a less stringent criteria is used) is sought, and therefore, the CG method will attain the desired convergence in a far less number of iterations than n.

As mentioned in Chapter 2, the least squares method always leads to a symmetric positive-definite coefficient matrix, regardless of the form of the differential operator. Methods based on the Galerkin formulation, however, lead to a symmetric matrix only for a symmetric operator. When convection terms are present, the coefficient matrix will be nonsymmetric. Other methods like the implicit Taylor-Galerkin or SUPG also generate nonsymmetric matrices. Efficient iterative methods used for the solution of nonsymmetric matrices are generalizations of the CG method, in order to exploit its desirable properties. Nevertheless, they are not as efficient as the CG in terms of storage requirements, CPU time, and convergence properties [2, Chap. 1].

The simplest approach to solve an nonsymmetric matrix is to make the coefficient matrix symmetric and positive-definite by premultiplying A by its transpose, and apply the

CG to the normalized equations: $A^T A\,x = A^T b$. This method is referred to as CGNR. It requires an additional transposed matrix-vector multiplication compared to the CG. More importantly, the condition number of $A^T A$ is the square of the condition number of A. This will slow down the convergence significantly by approximately squaring the number of iterations, which will be disastrous if A is ill-conditioned.

Another method which is generally considered the most robust iterative solver for large nonsymmetric systems, is the Generalized Minimum Residual (GMRES). The difficulty with using this method, is that all previous search directions must be stored in memory in order to construct each new one. This means that memory requirements will increase linearly with iteration number. Since this procedure demands a very large amount of storage, which is usually not available, the GMRES is restarted after m iterations. Breaking down the GMRES (or GMRES(∞)) into GMRES(m) will slow down the convergence and increase the number of iterations. A small value of m may also stall the method. There is no way of finding an optimum value for m, and it is usually chosen in the range of 10-50, depending on the problem size and conditioning of the coefficient matrix

Tables 4.2 and 4.3 [49, Chap. 3] show the number of operations and storage requirements, respectively, for different iterative methods ignoring preconditioning. The values given for the GMRES and the GCR are based on the assumption that $m \ll n$. It is clear from the tables that the CG is optimal compared to others in terms of both memory and CPU time. This in fact reflects the superior advantage of having a symmetric, positive-definite coefficient matrix.

4.4 Preconditioning

The success of an iterative method for the solution of system (4.1) largely depends on how well-conditioned the coefficient matrix is. In many practical problems, however, the coefficient matrix is (nearly) ill-conditioned, especially for nonsymmetric matrices. The goal of preconditioning is to create a new linear system with a better eigenvalue spectrum,

Method	Storage requirements
Splitting methods	0
Steepest descent	n
CG	$2n$
CGNR	$2n$
GCR(m)	$(2m+1)n$
GMRES(m)	$mn + \frac{1}{2}m^2 + O(m)$
BiCG	$4n$
Bi-CGSTAB	$4n$

Table 4.2: Storage requirements to solve a system of n linear equations. It is assumed that the storage for the initial approximation x_0 and the right-hand side b can be overwritten at exit by the final approximation and the final residual.

i.e., with eigenvalues as close together as possible. This will speed up the convergence, and in some cases, will allow convergence for a previously unstable system.

Applying the left preconditioning matrix M to system (4.1), gives the following new system:

$$M^{-1}Ax = M^{-1}b \tag{4.7}$$

A good preconditioner should resemble A, reduce the condition number of the original system, i.e., $\kappa(M^{-1}A) \ll \kappa(A)$, and be easily inverted. For example, when $M = A$, $M^{-1}A$ will reduce to the identity matrix whose eigenvalues are all equal, and system (4.7) is solved in just one iteration. This is never done in practice since obtaining M^{-1} is equivalent to using a direct solver with all its disadvantages (large memory and CPU time).

Preconditioning can be applied in three different ways:

1. right preconditioning $\quad (AM^{-1})(Mx) = b \quad \text{or} \quad \tilde{A}\tilde{x} = b$

2. left preconditioning $\quad (M^{-1}A)x = M^{-1}b \quad \text{or} \quad \tilde{A}x = \tilde{b}$

Method	Matrix-vector products	Transposed matrix-vector products	Vector dot products	Linear operations on vectors
Jacobi, GS, SOR	(1)	-	-	1
SGS, SSOR	(2)	-	-	2
Steepest descent	1	-	2	4
CG	1	-	2	6
CGNR	1	1	2	6
GCR(m)	1	-	$\frac{1}{2}m+2$	$m+4$
GMRES(m)	1	-	$\frac{1}{2}m+1$	$m+3$
BiCG	1	1	2	10
Bi-CGSTAB	2	-	4	12

Table 4.3: No. of operations per iteration for some well-known iterative solvers. For the splitting methods, there are no matrix-vector products, but triangular solves with the same complexity.

3. split preconditioning $\qquad (M_1^{-1} A M_2^{-1})(M_2 x) = M_1^{-1} b \quad \text{or} \quad \tilde{A}\tilde{x} = \tilde{b}$

where for the third type M is split into two matrices: $M = M_1 M_2$

Since the Conjugate Gradient method needs a symmetric and positive-definite matrix, the new coefficient matrix $\tilde{A}$ after applying preconditioning should also have this property. In the case of right and left preconditioning, this means that $M^{-1}A$ or AM^{-1} should be symmetric, positive-definite. As a result, the selection of a preconditioner is somewhat limited, since $M^{-1}A$ and AM^{-1} are not generally symmetric nor definite, even when both M and A are. For these cases, however, one can use a diagonal (or Jacobi) preconditioner, which is built by setting:

$$M = \text{Diag}(A) \tag{4.8}$$

and satisfies the above condition of $\tilde{A}$ being symmetric and positive-definite.

The effect of the Jacobi preconditioner is to make A more diagonally dominant, which in turn leads to a more uniform distribution of eigenvalues. Therefore it usually has a favorable effect on the convergence rate. This preconditioner is the simplest and the cheapest since:

- inverting a diagonal matrix is very straightforward,
- memory requirements is limited to a vector of length n, and
- its application only costs n multiplications.

Therefore, it is often recommended to try Jacobi preconditioner first before using more elaborate preconditioners.

In the present work, three different diagonal preconditioners (M) are used:

1. $m_{ii} = a_{ii}$ diagonal (JCG-1)
2. $m_{ii} = \sum_{j=1}^{n} |a_{ij}|$ L_1-norm (JCG-2)
3. $m_{ii} = \left(\sum_{j=1}^{n} (a_{ij})^2 \right)^{1/2}$ L_2-norm (JCG-3)

where a_{ij} are the elements of the original matrix A, and m_{ii} are the diagonal elements of M. The effect of each of these on convergence of the CG method is discussed in Chapter 5.

For a symmetric, positive-definite preconditioner matrix M, the difficulty in obtaining a symmetric, positive-definite $M^{-1}A$ (or AM^{-1}), can be overcome by alternatively splitting M into two matrices:

$$M = EE^T \tag{4.9}$$

and using split preconditioning:

$$(E^{-1} A E^{-T})(E^T x) = E^{-1} b \tag{4.10}$$

where $E^{-1} A E^{-T}$ is symmetric, positive-definite. The eigenvalues of $E^{-1} A E^{-T}$ are equal to those of $M^{-1}A$, thus indicating the equivalence of the two approaches. The factored

matrix E can be obtained by the incomplete Cholesky (IC) factorization, or in general, the incomplete LU (ILU) factorization (for nonsymmetric M).

For an incomplete factorization, the amount of fill-in is restricted in order to keep the cost (time and memory) of implementing the preconditioner low. By definition, ILU(s) or IC(s) keeps fill elements only from levels $1, 2, \ldots, s$. ILU(0) or zero level fill, eliminates all new nonzeros created during factorization. ILU(1) or first level fill, allows nonzeros created only by the original entries, while ILU(2) allows fill-in created by at least one entry from the first level fill, and so on.

ILU(0) and IC(0) preconditioning are the most commonly used incomplete factorizations, since the constructed matrices have the same data structure as A, and thus the same memory requirements. For higher-order fill-ins, the memory requirements are not only increasing, but also are not easily predictable. IC(0) is the preconditioner used in this work, and its performance will be discussed in Chapter 5.

4.5 Stopping Criteria

Contrary to direct solvers, iterative methods provide a sequence of approximations to the exact solution, and hence, it is up to the user to decide whether or not the calculated result is acceptable. For this reason, a criterion is required in order to halt the iterative process when prescribed conditions are met. The mechanisms to impose the criteria are activated when either of the following happens:

1. the maximum number of iterations is reached.
2. the solution is acceptable (within the specified tolerance), or
3. the iterative method is stalled or is diverging.

Case 1 The convergence behavior of a particular problem basically depends on the conditioning of the matrix, and the appropriateness of the chosen solver for that problem. To avoid excessive calculations for the cases where the iterative solver makes no

progress or the convergence is unacceptably slow, the number of solver iterations (or matrix-vector multiplications) should be limited. The value of this maximum number, therefore, depends on the convergence behavior of the problem, which is not known a priori. It also depends on the desired level of solution accuracy, since higher accuracy demands a larger number of iterative steps.

In the numerical results presented here it has been attempted to assign an optimum value for the maximum number of solver iterations for each test case, based on the convergence trend and the preset tolerance.

Case 2 For this case, the criterion is usually based on some norm of residual: $\|r_k\|$ where $r_k = b - A x_k$. The reason for this choice is that for most iterative solvers the residual vector is available at each iteration, and one only needs to calculate its norm.

The most commonly used convergence test is:

$$\|r_k\| < \epsilon \|b\| \tag{4.11}$$

where ϵ is the tolerance. Another common test is:

$$\|r_k\| < \epsilon \|r_0\| \tag{4.12}$$

which is equivalent to (4.11) for the case of a zero initial guess, i.e., when $r_0 = b$. In this work, a more general form of (4.11) is used:

$$\|r_k\| < \epsilon_{rel} \|b\| + \epsilon_{abs} \tag{4.13}$$

where ϵ_{rel} and ϵ_{abs} are relative and absolute tolerances, respectively.

For the case of preconditioning (either right, left, or split), the above criteria still holds but r_k and b will be replaced by the preconditioned values at no extra cost, since the iterative method will be applied to the preconditioned system.

Case 3 By assigning proper upper limit tolerances, the criteria set forth for the convergence can be used to avoid divergence, as well. In situations where the iterative

process seems to be stalled, one should be careful in setting up a mechanism to stop the method. The reason is that some iterative methods after a significant number of iterations with a rather stalled pattern suddenly start to converge. The iterative process might stall if the matrix is ill-conditioned, and in such a case, a more efficient preconditioner should be used.

Chapter 5

Numerical Results

In this chapter, the least-squares method, along with the adaptation method, are applied to three well-known benchmarks, including supersonic and transonic flows. The effects of preconditioning, artificial viscosity, and adaptation are studied for each test case and comparison is made with analytical solutions or published results. All the computations presented are performed on a Silicon Graphics Indy computer, with one 100 MHz R4000 processor.

5.1 The Shock-Reflection Problem

The first test case is the reflection of a shock from a solid wall, as shown in Fig. 5.1. The

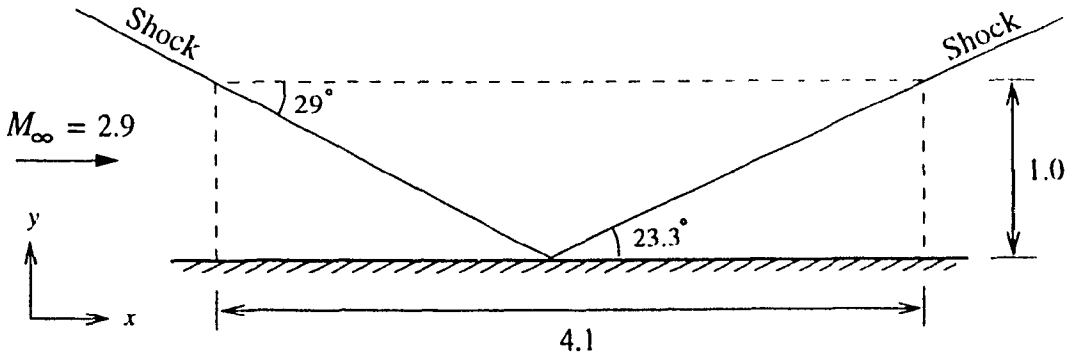

Figure 5.1: Reflection of a shock from a solid wall: the computational domain.

domain is initially discretized uniformly with 60×20 bilinear rectangular elements. The boundary conditions are:

$$\text{inlet} \begin{cases} \rho = 1.0 \\ u = 2.9 \\ v = 0.0 \\ p = 0.7143 \end{cases} \qquad \text{upper boundary} \begin{cases} \rho = 1.7 \\ u = 2.6193 \\ v = -0.5063 \\ p = 1.5282 \end{cases}$$

For the lower boundary, the no-penetration boundary condition is imposed by setting $v = 0$. At the exit, no boundary condition is specified since the flow is supersonic. The free stream density and speed of sound are used for non-dimensionalization. The values at the upper boundary are used as the initial guess.

To study the effect of preconditioning on the performance of the Conjugate Gradient (CG) iterative solver, the shock reflection problem is solved using a Jacobi preconditioner (JCG) and an incomplete Cholesky preconditioner with a zero level fill-in (ICCG(0)). Figure 5.2 shows the convergence history of the iterative solver at the 30th global iteration. The relative and absolute tolerances (section 4.5) for the solver are set to:

$$\epsilon_{rel} = 10^{-6} \quad , \quad \epsilon_{abs} = 10^{-14} \tag{5.1}$$

and the maximum number of solver iterations is set to a very large number in order to obtain an accurate solution by allowing the residual of the iterative solver to drop six orders of magnitude in all global iterations.

The Jacobi preconditioners JCG-1, JCG-2, and JCG-3 (section 4.4) are all applied from the left. As mentioned in Chapter 4, the ICCG(0) should be applied from both left and right using split preconditioning, but unfortunately this option was still not available in the iterative solver package used. The results presented for the ICCG(0) are therefore for the left-preconditioning only.[1]

[1] Applying the incomplete Cholesky preconditioner with zero level of fill-in from the left will destroy the numerical symmetry of the linear system, even though the matrix will still preserve its structural symmetry. As a result, the convergence of the CG method is no longer guaranteed for this asymmetric system. The preconditioned matrix in this case was close to a symmetric and positive-definite matrix, and it was therefore still possible to obtain a converged solution with the left ICCG(0).

Method	CG	JCG-1	JCG-2	JCG-3	ICCG(0)
CPU time (s)	26 34	20.16	20.80	20.47	17.05
Solver iterations	301	200	210	203	40

Table 5.1: Computation time for different preconditioned iterative methods.

Figure 5.2 shows that using the very simple Jacobi preconditioner, the residual of the CG method dropped two orders of magnitude, and the number of iterations were reduced by about 33%. The JCG-2 method performs better than the JCG-1 and JCG-3 methods. The diagonal elements of the JCG-2 preconditioner are constructed using the L_1-norm (section 4.4) and are therefore larger, leading to a more diagonalized coefficient matrix which has better convergence properties. The incomplete Cholesky preconditioner, as expected, has a much better convergence rate. The cost of its usage, however, is of the same order as the Jacobi preconditioner (Table 5.1) because it requires factorization and more vector-matrix products.

The convergence rate and overall time history of different preconditioners are shown in Fig. 5.3. Since the residual of the iterative solver is allowed to drop six orders of magnitude at each iteration, the convergence history of all the methods lie on the same curve.

In the present numerical examples, the solution at steady-state is desired, and therefore only an approximate solution at each time level is required, and less stringent conditions can be imposed on the iterative solver. The shock reflection problem is solved again by setting:

$$\epsilon_{rel} = 10^{-2} \quad , \quad \epsilon_{abs} = 10^{-7} \tag{5.2}$$

and the maximum number of solver iterations to 150. It should be noted that the ϵ_{abs} must be less than the convergence tolerance of the least-squares method ϵ_{ls}. The ϵ_{abs} is the lower limit for the solver residual, and since the initial residual of the solver is that of the least-squares method, $\epsilon_{abs} > \epsilon_{ls}$ would mean that no operations will be performed by the solver on the linear system. Moreover, if ϵ_{abs} is chosen to be equal to the global tolerance, the residual of the least-squares method will oscillate when it becomes of the same order of the

global tolerance, hence not being able to drop below it. The maximum number of solver iterations is chosen based on the convergence rate of the JCG's in Fig. 5.3, and seems to be a proper upper limit.

The results obtained after changing the solver parameters are shown in Fig. 5.4. It is seen that the convergence history of all the methods are almost the same as in Fig. 5.3, except for the CG method which requires 6 more iterations to converge. The important consequence of this optimization is the drop in the overall CPU time by approximately 50%, which is quite significant. To show that the quality of the solution is not affected, the pressure at $y = 0.5$ is plotted in Fig. 5.5. The last item in the legend: "All methods; rel. error=1.0E-6" refers to the solution of all methods before imposing these. As can be seen from the figure, the accuracy of the solution is preserved despite reducing ϵ_{rel} and ϵ_{abs}.

The performance of the JCG-2 method applied as a right-preconditioner is also studied and compared with the left JCG-2 method (Fig. 5.6). The CPU time for the right-preconditioner is 682.1 seconds compared to 632.4 seconds for the left-preconditioner, showing about 8% increase. The reason is that for the left-preconditioning, the right-hand side vector is premultiplied by the inverse of the preconditioner matrix (section 4.4), while it remains intact for the right-preconditioning. Therefore, the initial residual of the iterative solver for the left-preconditioning would be lower than that for the right-preconditioning, thereby reducing the number of solver iterations and, consequently, the computation time. For the rest of the computations, therefore, the JCG-2 method is applied as a left-preconditioner.[2]

The same problem is also solved using a banded Cholesky direct solver. The computation time is 34 seconds per iteration, and 2074 seconds overall, which is 3.28 times higher than the optimized JCG-2 iterative method, clearly indicating the advantage of using iterative solvers. The savings in the computation time for the iterative solver compared to a direct solver would be more pronounced for problems of larger size, as outlined in

[2]Since there is a possibility that ICCG(0) when applied as both left- and right-preconditioner might have a better performance than the JCG-2 preconditioner, it is excluded in making this conclusion.

Δt	CPU time (s)	No. of iterations
0.05	933.172	108
0.1	632.424	61
0.15	554.085	46

Table 5.2: Computation time for different time steps.

Chapter 4.

The effect of artificial viscosity on the solution is examined by using different time steps to attain the steady-state solution. It was mentioned in Chapter 2 that the time step is the only parameter which controls the amount of artificial viscosity in the least-squares method. Figure 5.7 shows the pressure contours using different time steps. As Δt increases, more artificial viscosity is added to the solution, and therefore the shock is more smeared. At the other extreme, a very small amount of artificial viscosity, which corresponds to very small Δt, tends to destabilize the solution by creating oscillations at discontinuities.

The effect of artificial viscosity on the solution near the shock is demonstrated in Figure 5.8, which shows the pressure distribution at $y = 0.5$ for different time steps. The convergence history for different Δt's is shown in Figure 5.9, and the corresponding CPU times in Table 5.2.

The results used for the adaptation should be of such a quality that demonstrate at least the basic phenomena happening in the flow. If the results have poor quality, many adaptive cycles will be required to achieve an acceptable solution. It is also possible that an accurate solution may never be achieved. A more accurate initial flow solution obtained, for example, by using a finer grid would need a lower number of adaptation cycles. Nevertheless, these cycles are generally computationally more expensive, because of increased number of nodes which means larger CPU time.

Based on the results for different time steps, the solution corresponding to $\Delta t = 0.1$ is chosen as the input data for the adaptation, since the shock is not too smeared, and the corresponding CPU time is moderate. The shock for $\Delta t = 0.05$ has better resolution

compared to the others, but the corresponding CPU time is very high. Moreover, the results for larger Δt's after one cycle of adaptation would most likely be as good as those for $\Delta t = 0.05$.

Figures 5.10, and 5.11 show the mesh and the flow solution for the first four adaptive cycles using $\Delta t = 0.1$. The mesh gets finer in the vicinity of the shock after each adaptation. This is especially more pronounced in the stronger incident shock. The pressure distribution after each adaptation is shown in Figure 5.12. There is very little difference between the results of the 3rd and 4th cycles. This indicates that the present amount of artificial viscosity is high for the size of the mesh near the shock, and it must be reduced if better shock resolution is desired. The time step is therefore reduced to $\Delta t = 0.05$ for the 3rd cycle, and the computation is continued up to the 5th cycle with the same time step.

The mesh and the corresponding solution for $\Delta t = 0.05$ are shown in Figures 5.13, and 5.14. The mesh for the 3rd cycle is the same as the one used for $\Delta t = 0.1$. Figure 5.15 shows the improvements obtained in resolving the shock (especially the reflected shock) by reducing the amount of artificial viscosity.

The adapted meshes in Figs 5.10 and 5.13 show how the adaptive method relocates the elements and aligns them with the shock, where the approximation error is high due to the sharp gradients. In other parts of the domain, however, the elements are stretched since the solution is smooth and without any significant gradients. This behavior is consistent with the theory outlined in Chapter 3.

Figure 5.16 shows the evolution of the mesh and the solution during the five cycles of adaptation. The convergence and time histories are shown in Figure 5.17. The CPU time has surprisingly reduced in the 3rd and the 4th adaptive cycles despite using a lower time step. The reason can be seen from Figure 5.18 where the number of solver iterations is reduced significantly in the 3rd cycle, thus leading to a smaller CPU time. The reduction in the number of the CG iterations implies that the conditioning of the coefficient matrix has improved. This improvement could be attributed to larger diagonal elements in the coefficient matrix which contain $1/\Delta t$ terms.

The shock reflection problem is also solved on a four times finer uniform grid (120×40 elements) using $\Delta t = 0.05$, for comparison with the adaptive procedure. The pressure contours are shown in Figure 5.19, and the comparison with the adapted solution is made in Figure 5.20. It is evident from these figures that, the solution on the fine grid is not very accurate compared to the adapted solution, especially for the reflected shock, indicating the importance of compatibility of the grid orientation with the flow structure in obtaining accurate numerical results without using more elements.

The computation time for the fine grid is 4324 seconds compared to the 7036 seconds for the five cycles of adaptation, and the storage requirements is close to four times higher than that of the coarse grid. The high value of the CPU time and the memory, as well as the relative low accuracy of the fine grid clearly demonstrate the advantage of using adaptation

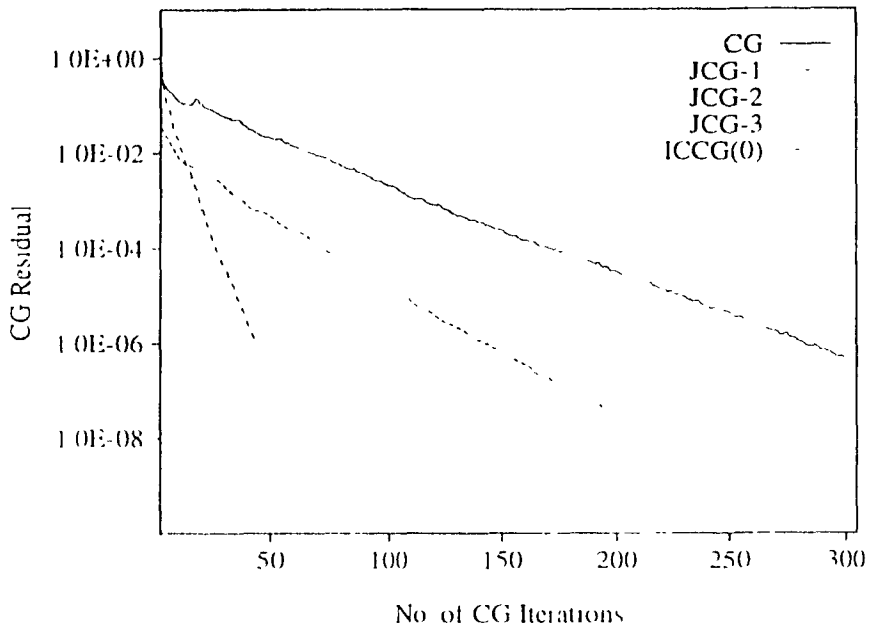

Figure 5.2 Convergence history of the iterative solver with different preconditioners.

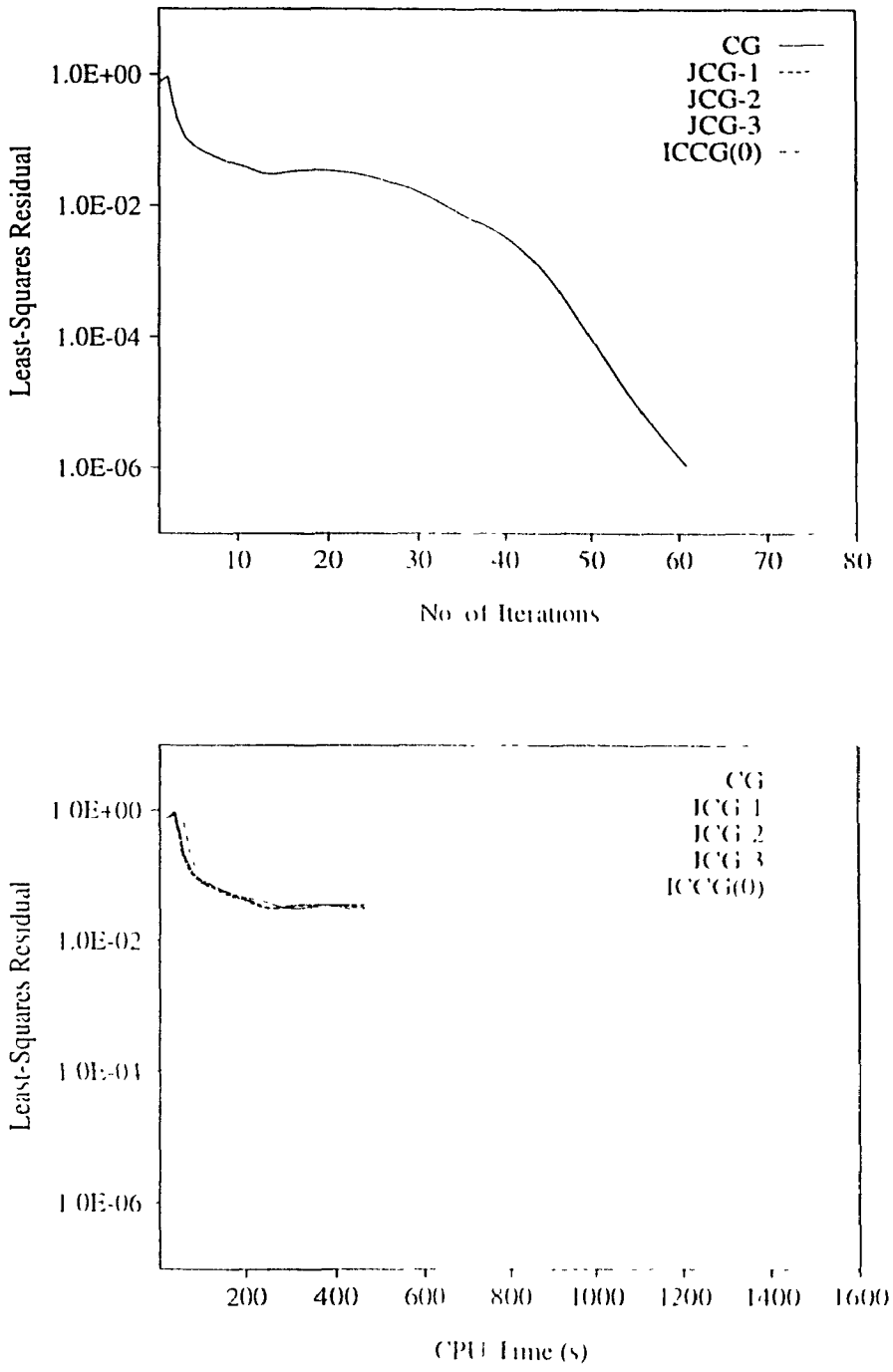

Figure 5.3: Convergence and time histories of different preconditioned iterative methods, $\Delta t = 0.1$.

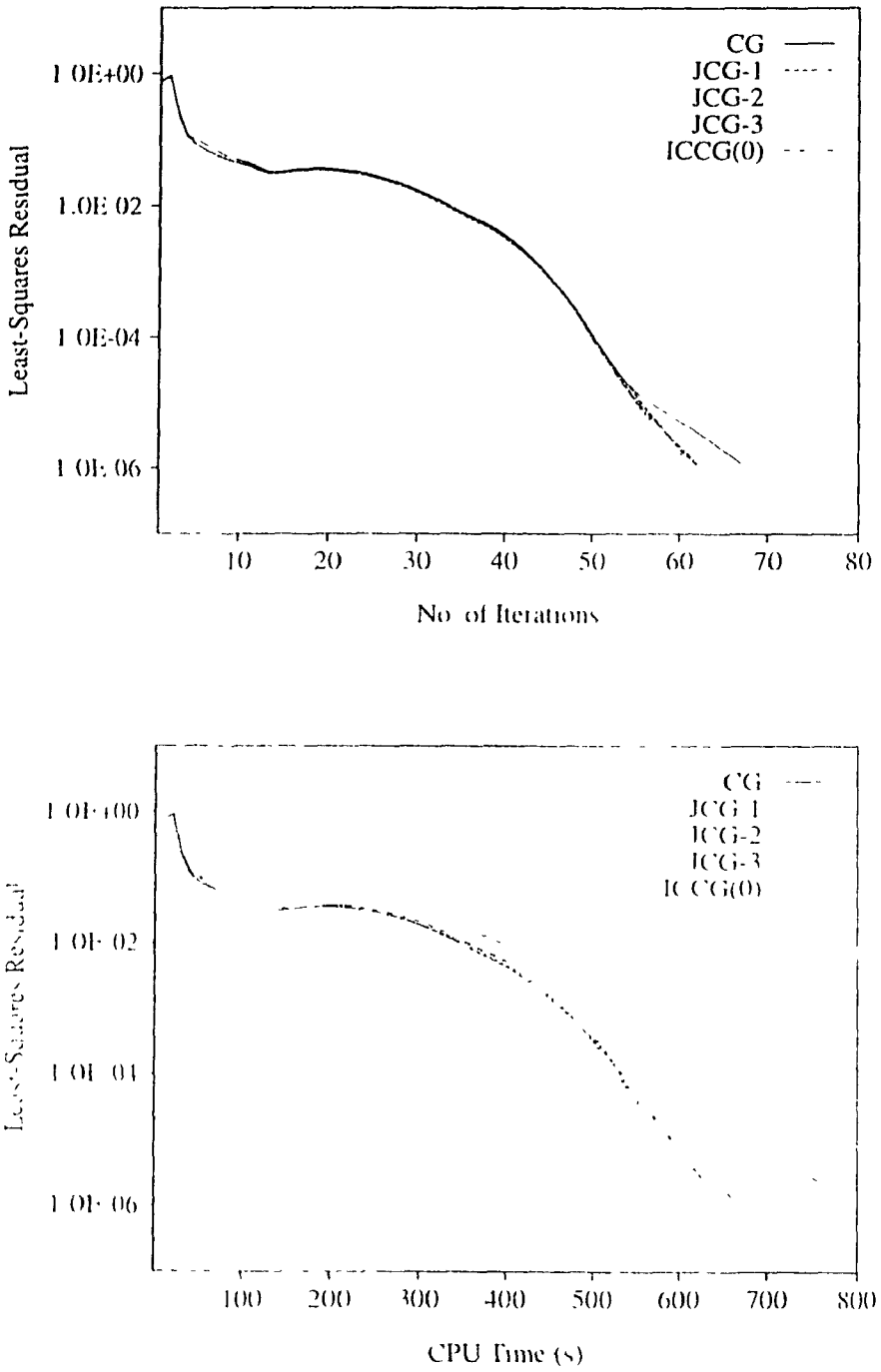

Figure 5.4: Convergence and time histories of different preconditioned iterative methods after optimizing the solver parameters, $\Delta t = 0.1$.

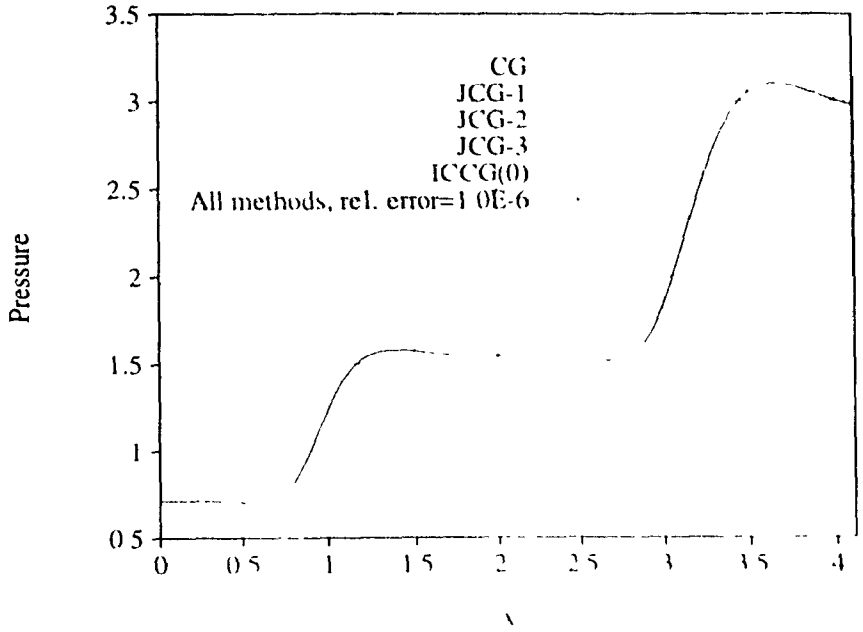

Figure 5.5: Pressure distribution at $y = 0.5$ before and after optimizing the solver parameters, $\Delta t = 0.1$.

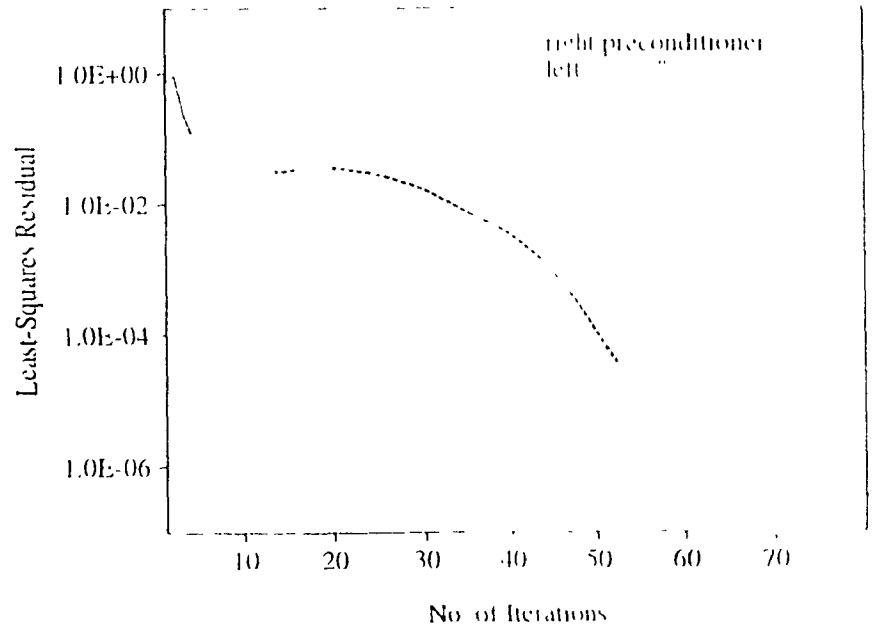

Figure 5.6: Convergence history of the Jacobi left- and right-preconditioners

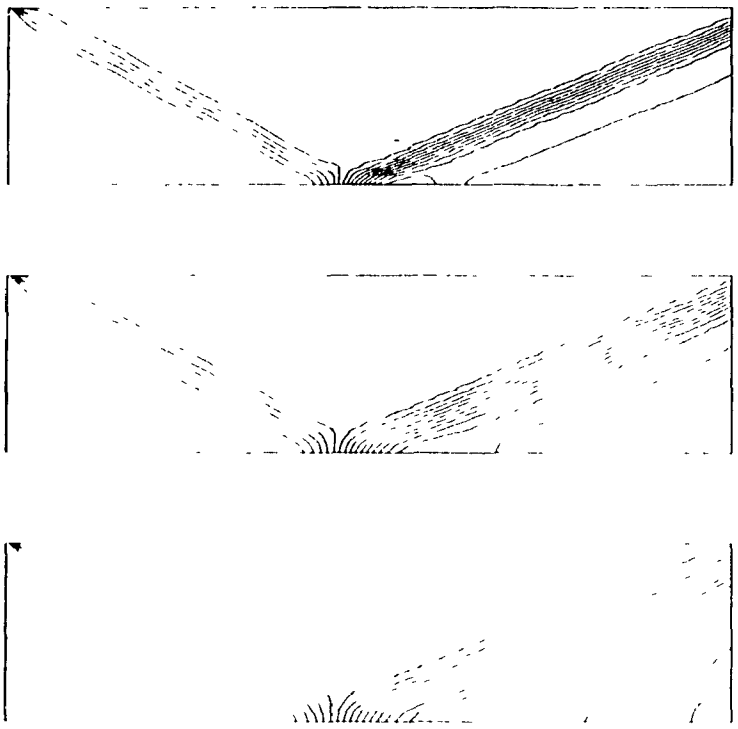

Figure 5.7 Pressure contours for different time steps, from top to bottom. Δt = 0.05, 0.1, 0.15

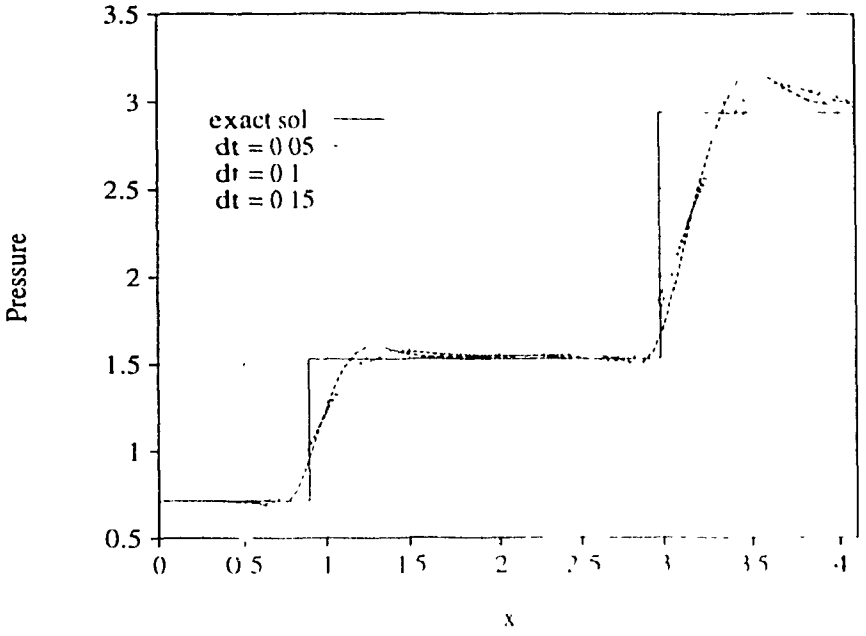

Figure 5.8: Pressure distribution for different time steps at y = 0.5

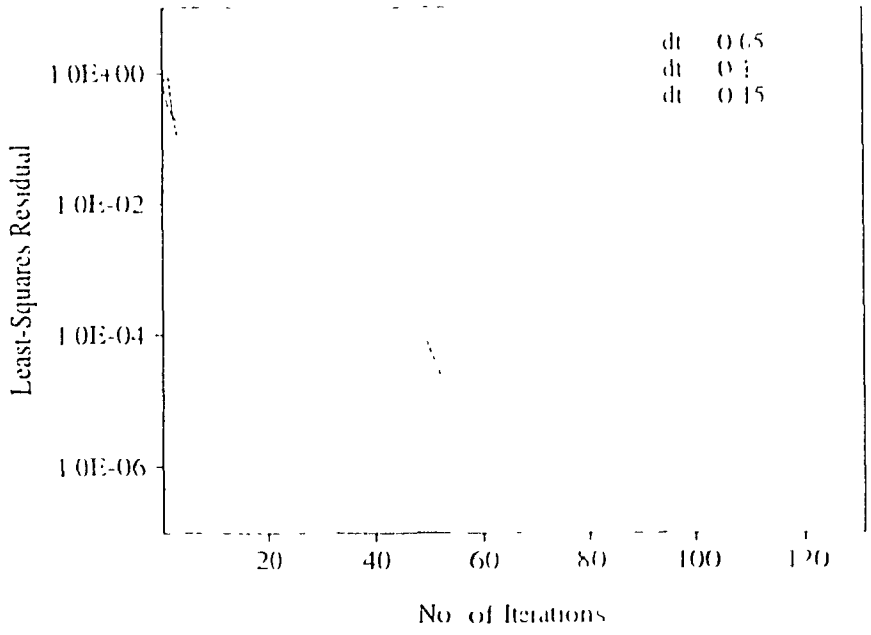

Figure 5.9: Convergence history for different time steps

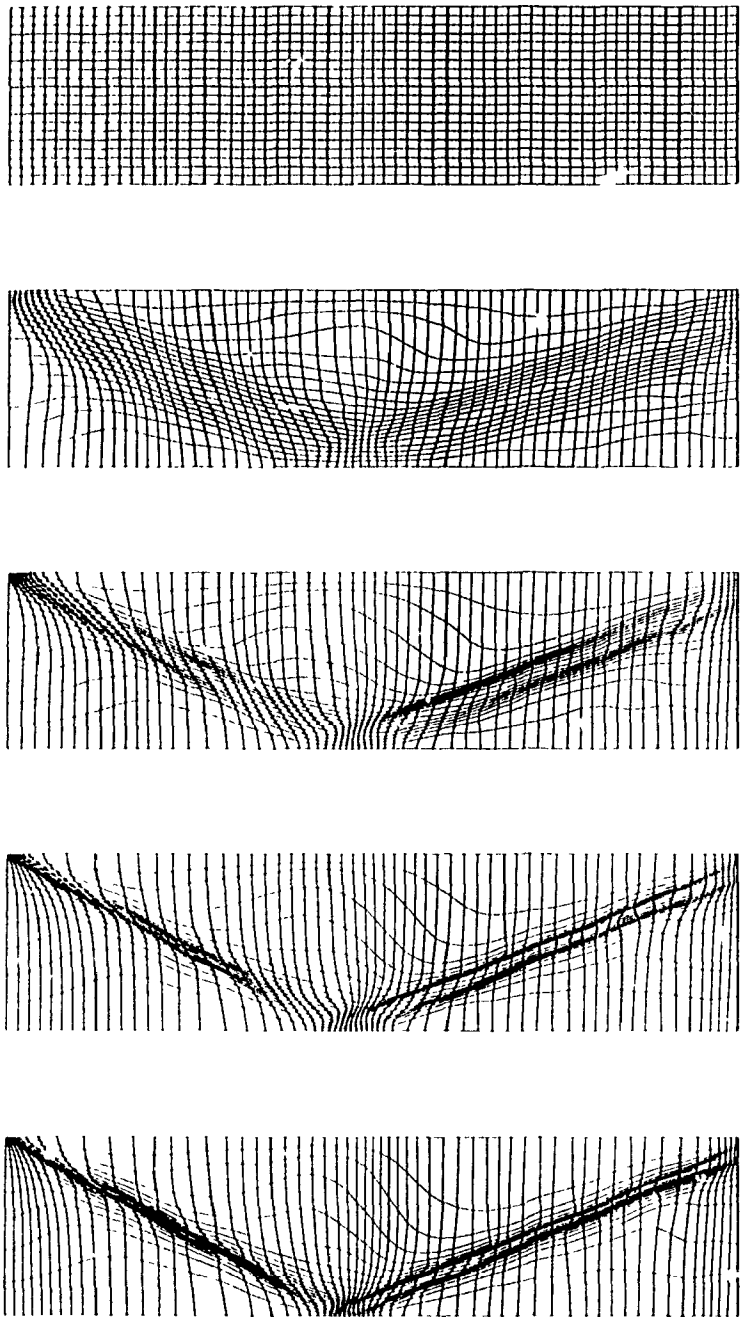

Figure 5.10: Original and adapted grids after four cycles, $\Delta t = 0.1$.

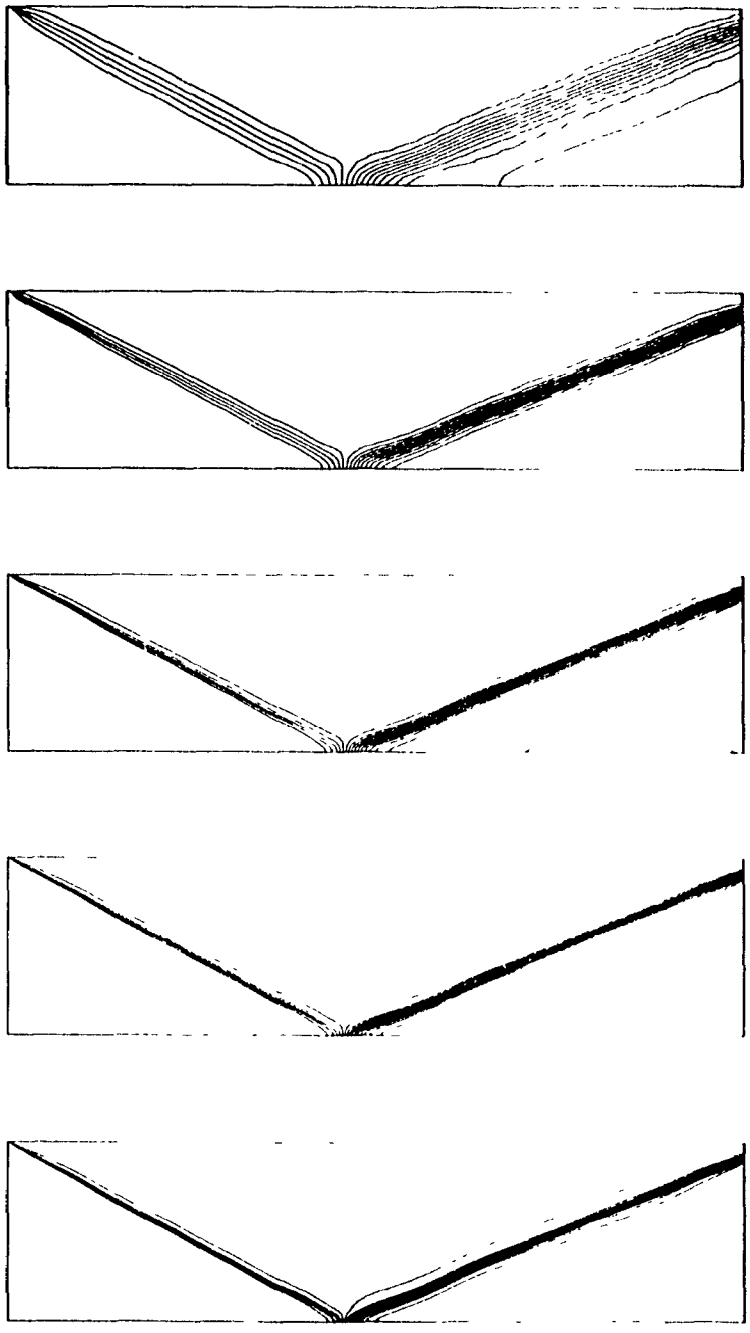

Figure 5.11: Original and adapted solutions after four cycles, $\Delta t = 0.1$.

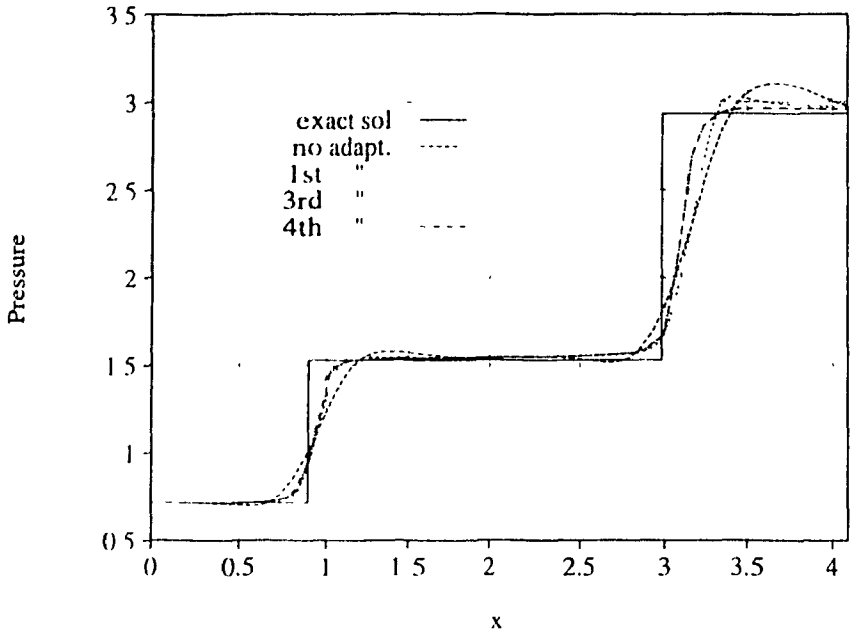

Figure 5.12: Pressure distribution at $y = 0.5$ after four cycles of adaptation, $\Delta t = 0.1$.

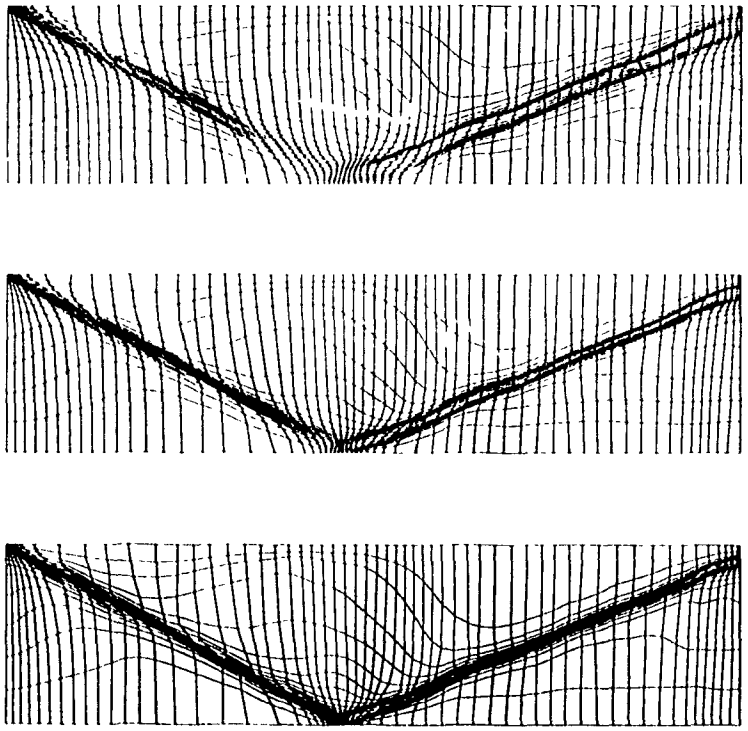

Figure 5.13: Adapted grids after reducing the artificial viscosity, $\Delta t = 0.05$.

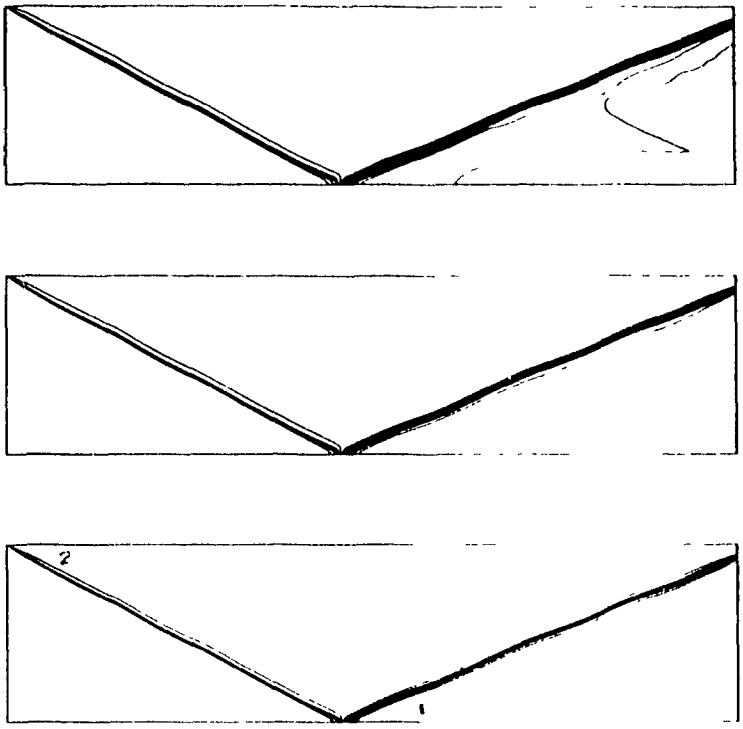

Figure 5.14: Adapted solution after reducing the artificial viscosity, $\Delta t = 0.05$

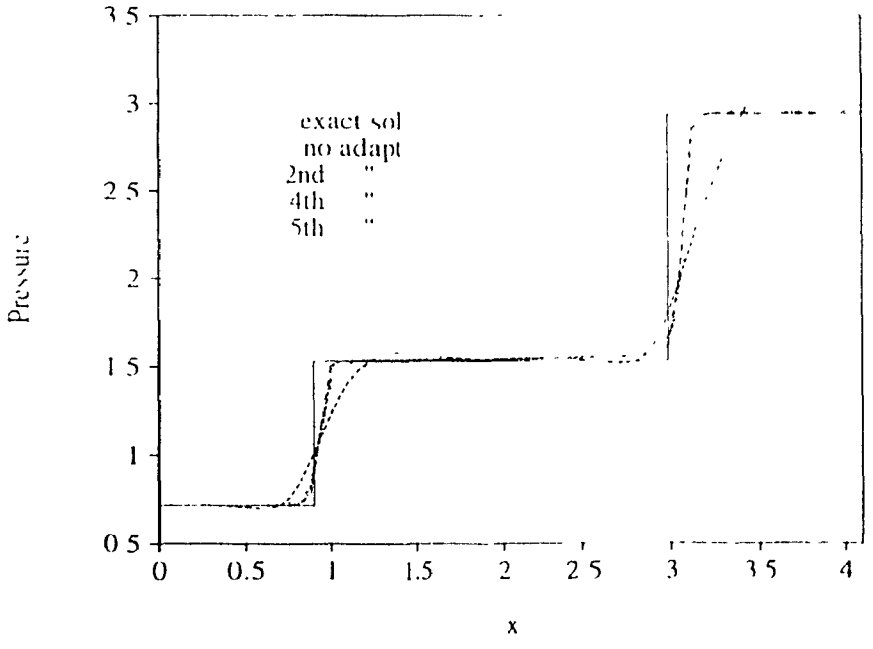

Figure 5.15: Pressure distribution at $y = 0.5$ after reducing the artificial viscosity, $\Delta t = 0.05$.

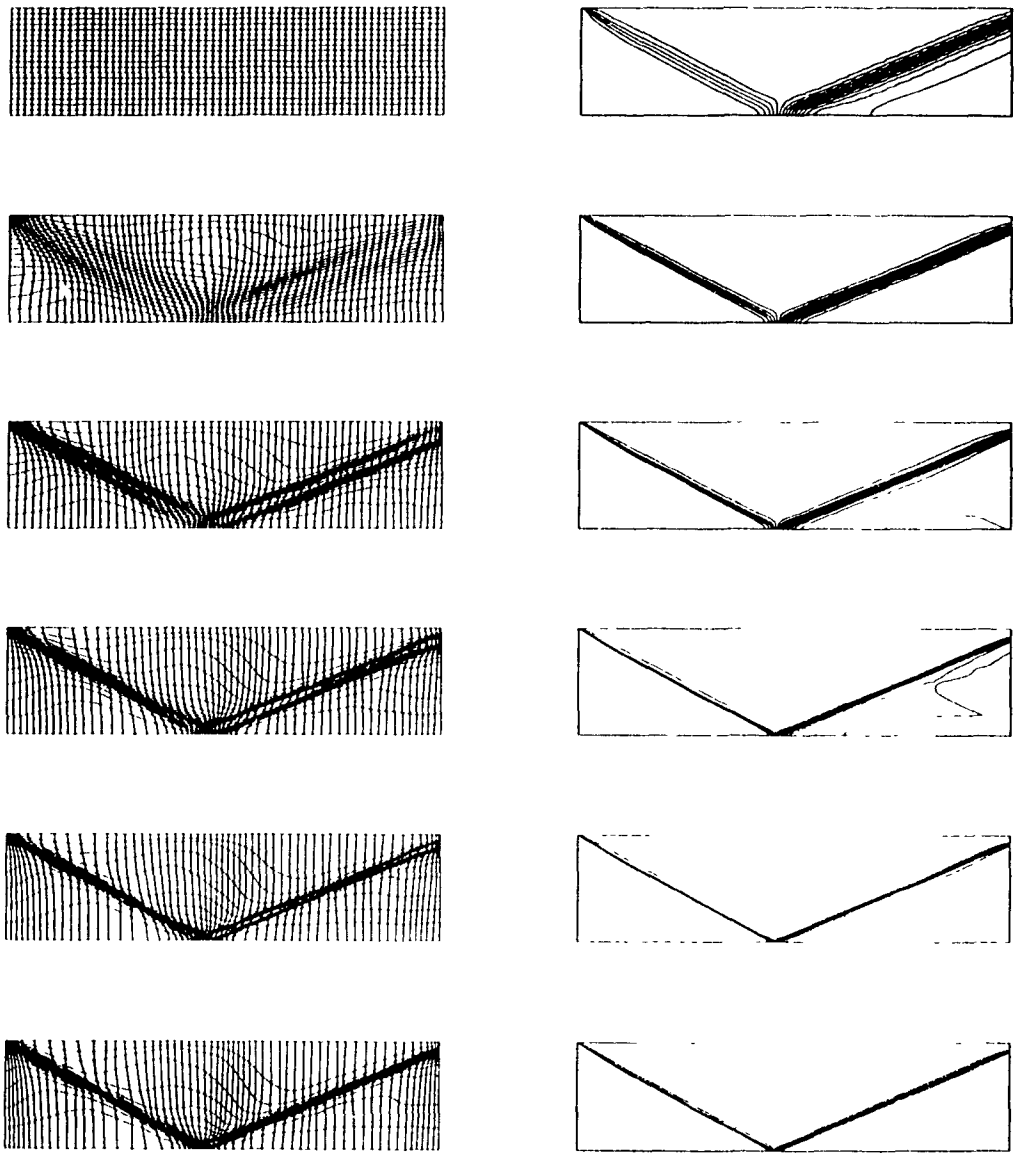

Figure 5.16 Evolution of the grid and the solution during the adaptation; $\Delta t = 0.05$ for the last three cycles.

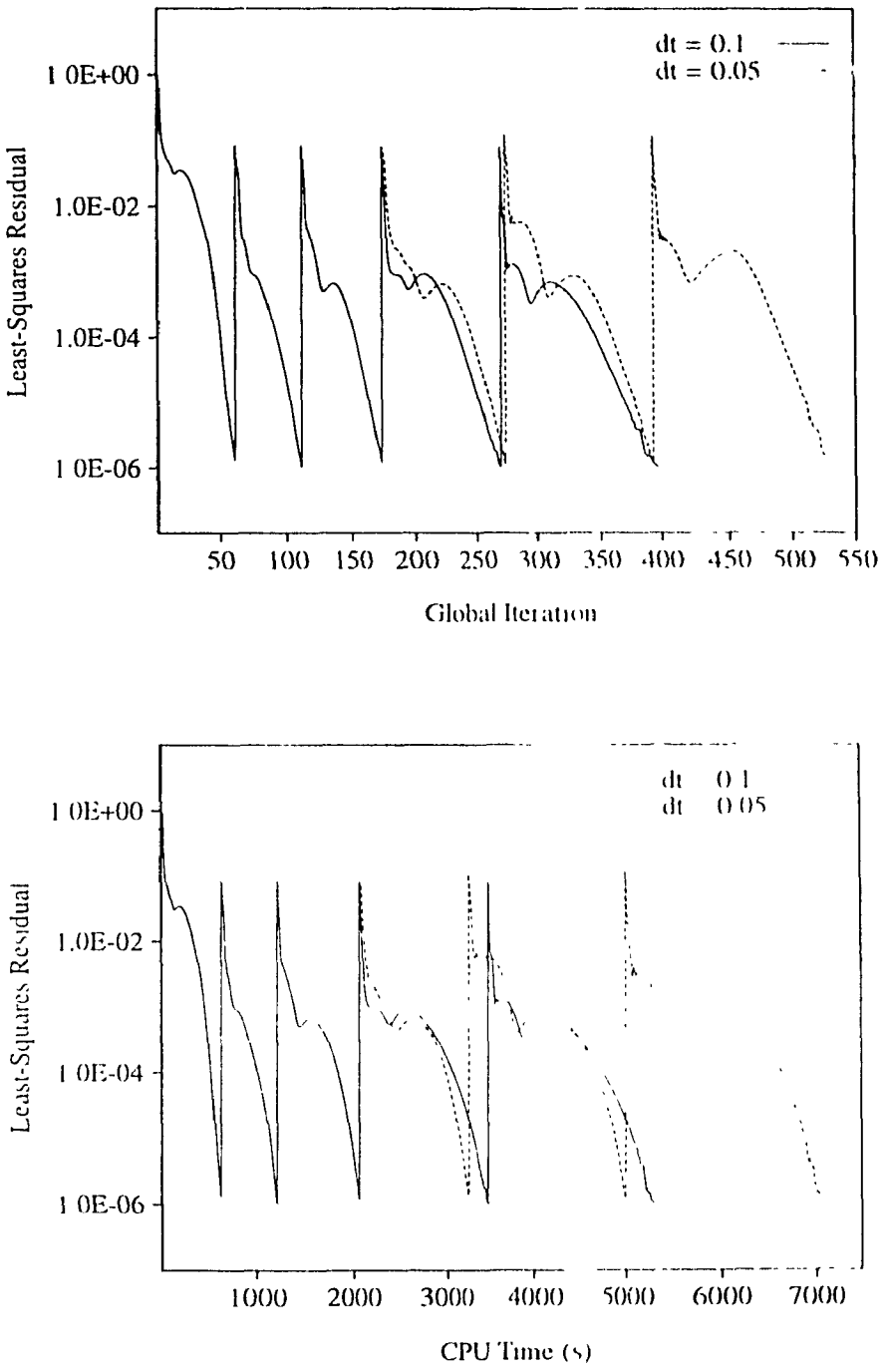

Figure 5.17: Convergence and time histories of the flow solver with adaptive procedure.

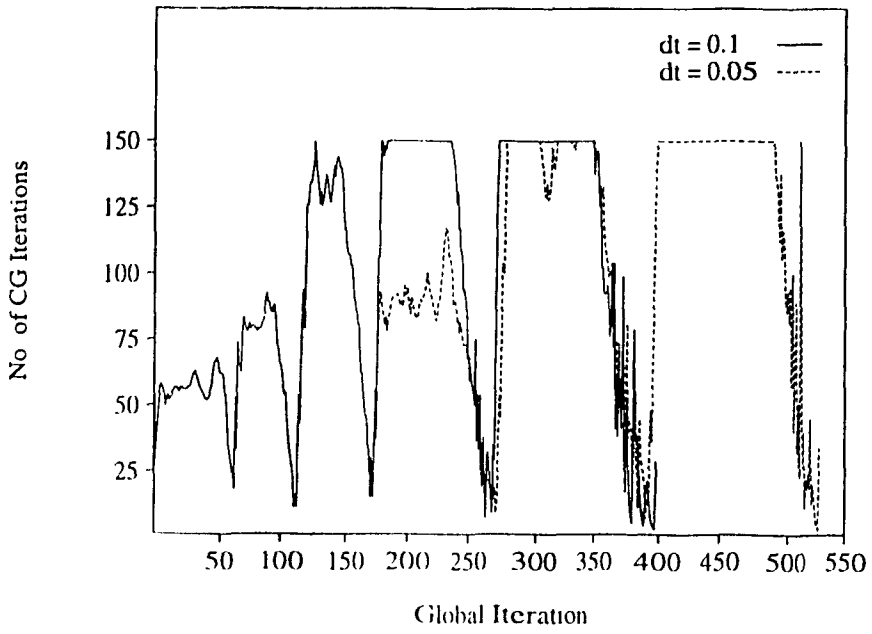

Figure 5.18: Convergence history of the iterative solver with adaptive procedure.

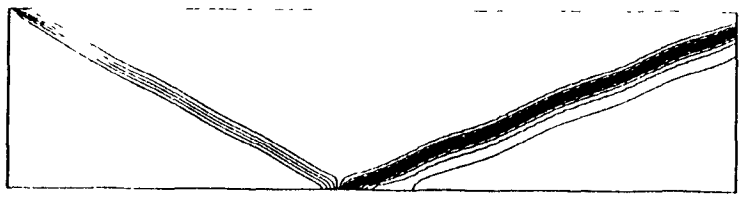

Figure 5.19: Pressure contours for the fine grid, $\Delta t = 0.05$.

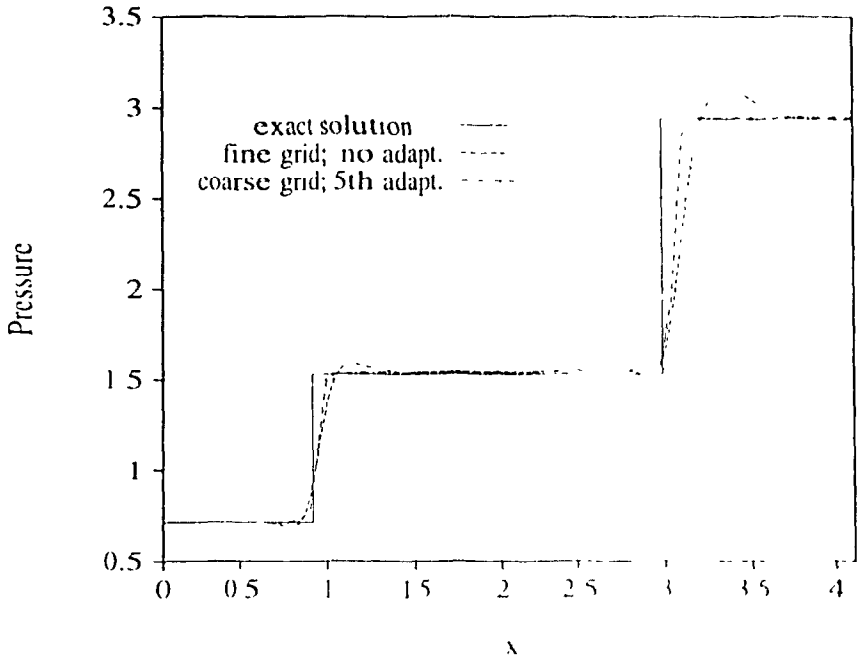

Figure 5.20: Pressure distribution at y 0.5 for the adapted and fine grids

5.2 Supersonic Channel Flow

The second test case is that of supersonic flow in a channel with a circular arc bump on the lower wall. The computational domain for this problem is shown in Figure 5.21, and has the following dimensions:

$$t/c = 0.04 \quad ; \quad L = c = 1.0$$

The boundary conditions at inlet are:

$$\rho = 1.0 \quad , \quad u = 1.65 \quad , \quad v = 0.0 \quad , \quad p = 0.7143$$

On walls the no-penetration boundary condition is imposed, and the exit boundary is free. The free stream density and speed of sound are used to non-dimensionalize the variables. The grid consists of 64×16 uniformly distributed bilinear rectangular elements, with 16 elements in the y-direction, 22 elements on the bump, and 21 elements on each side of the bump.

The test case is run for three different time steps to demonstrate the effect of artificial viscosity on the solution. The results are shown in Figure 5.22. The leading- and trailing-edge shocks, as well as the interaction of the trailing-edge shock with the reflected shock are qualitatively well-captured. As expected, the shocks are smeared for large Δt's, and are sharper with smaller Δt's, which correspond to smaller amounts of artificial viscosity. This

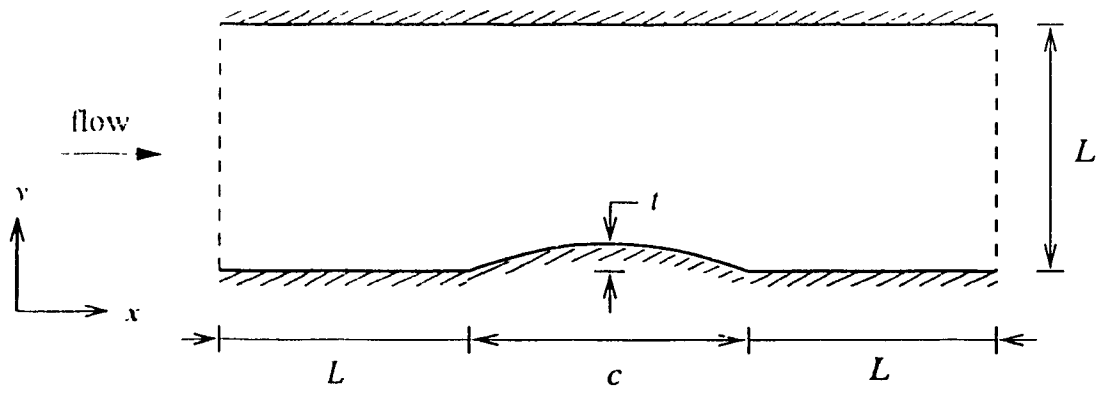

Figure 5.21: The computational domain for the channel flow.

Δt	CPU time(s)	No. of iterations
0.05	666.08	105
0.1	426.11	59
0.2	327.84	36

Table 5.3: Computation time for different time steps.

is evident in Fig. 5.23, where the pressure at the cross-section $y = 0.2$ which cuts through three shocks (leading-edge, trailing-edge, and the reflected one) is plotted. All the shocks are sharper for $\Delta t = 0.05$, but the profile of the leading-edge shock has an undershoot. The convergence history for different time steps is shown in Figure 5.24, and the total CPU time is given in Table 5.3.

A comparison between the quality of the solution and the CPU time for different time steps shows that the results for $\Delta t = 0.1$ are the suitable ones to be used for the adaptation. The reason is that the shocks have a moderate resolution and not too smeared as for Δt 0.2, and basic features of the flow are clearly captured. Moreover, the CPU time for Δt 0.1 is 36% less than that for $\Delta t = 0.05$, and not too high to be considered expensive.

Before starting the adaptation, the parameters of the iterative solver are optimized in order to reduce the computation time. The solver is chosen to be the Jacobi preconditioned CG method (JCG-2) due to its good performance for the previous test case. This assumption is verified later.

The parameters are first set to those of 5.1, with the maximum number of iterations given a very large value. This will ensure high accuracy for the calculated solution. The computation time for this case is 751.4 seconds. The convergence history of the iterative solver with these parameters is shown in Figure 5.25 at selected global iterations. Based on this figure, the new value for the maximum number of iterations is set to 100, and the tolerances are set to:

$$\epsilon_{rel} = 10^{-2} \quad ; \quad \epsilon_{abs} = 10^{-7}$$

This value for ϵ_{rel} is chosen based on the experience from the previous test case.

Iterative method	CPU time(s)
JCG-1	433.35
JCG-2	426.11
JCG-3	434.04

Table 5.4: Computation time for different Jacobi preconditioned iterative methods.

The convergence history and the corresponding pressure distribution with these new parameters are shown in Figures 5.26 and 5.27. The flow solver residual is virtually the same as for the non-optimized case, and it is therefore expected to obtain the same results. This is confirmed by Fig 5.27, where the pressure profiles lie on the same curve. Without losing accuracy, the computation time for the optimized case is 43% less.

The same problem is also solved using the JCG-1 and JCG-3 iterative methods, and with the optimized parameters. The accuracy of the results was found to be the same as for the JCG-2 method. The computation time, however, was higher for these methods (Table 5.4). It is, therefore, justified to use the JCG-2 method as the solver.

Using the pressure as the key variable, the results for $\Delta t = 0.1$ are then used to adapt the mesh. Figures 5.28 and 5.29 show the mesh and the corresponding solution. The adaptation is continued until no major improvements are seen in the adapted results compared to the previous cycle. The mesh and pressure contours of the 3rd and the 4th cycles in Figures 5.28 and 5.29 are only slightly different. This is more evident in Figure 5.30, which shows the pressure distribution across the shocks. The shocks (especially the leading-edge shock) become sharper after each adaptation. Since no significant improvement in the solution is achieved after the 3rd cycle of adaptation, indicating a high amount of artificial viscosity, the time step is reduced to 0.05 for the 3rd cycle, and the computation is continued until the 5th cycle.

Figures 5.31 and 5.32 show the mesh and the results for this lower time step. The improvements in the shock resolution is evident from Figure 5.33, where the leading-edge shock is quite sharp, and the trailing-edge and the reflected shock profiles are close to

vertical. The evolution of the shock and solution during the five cycles of adaptation are demonstrated in Figure 5.34.

Figure 5.35 shows the convergence and time histories before and after reducing the time step. The convergence history of the iterative solver is shown in Figure 5.36. Similar to the previous test case, the number of iterations at the 3rd cycle is reduced after lowering the time step, indicating the improvement of the conditioning of the matrix. This reduction, however, has not led to any savings in the CPU time, because the number of global iterations is increased in this cycle, resulting in an overall increase of the computation time.

The Mach number distribution on the lower and upper walls of the channel before and after the adaptation is plotted in Figures 5.37 and 5.38, respectively. The adaptation has improved the resolution of the shock and damped out the overshoot before the upper-wall shock. On the lower wall, there are some non-physical oscillations near the leading- and trailing-edges, most probably due to the use of non-conservative formulation.

The adapted solution is compared with the published results of Eidelman *et al.* [14] and Jiang *et al.* [31] in Figure 5.39. Since these authors have not used an adaptive method, the shock is more smeared in their results. Overall, the adapted least-squares method shows good agreement, except for the oscillations at the leading- and trailing-edges

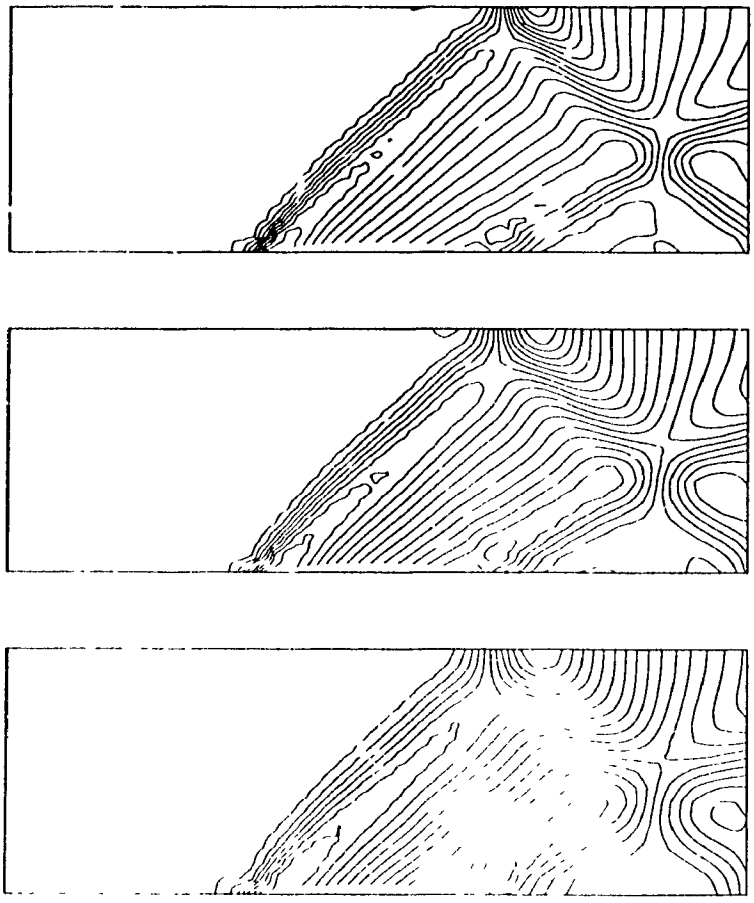

Figure 5.22: Pressure contours for different time steps; from top to bottom: $\Delta t = 0.05$, 0.1, 0.2.

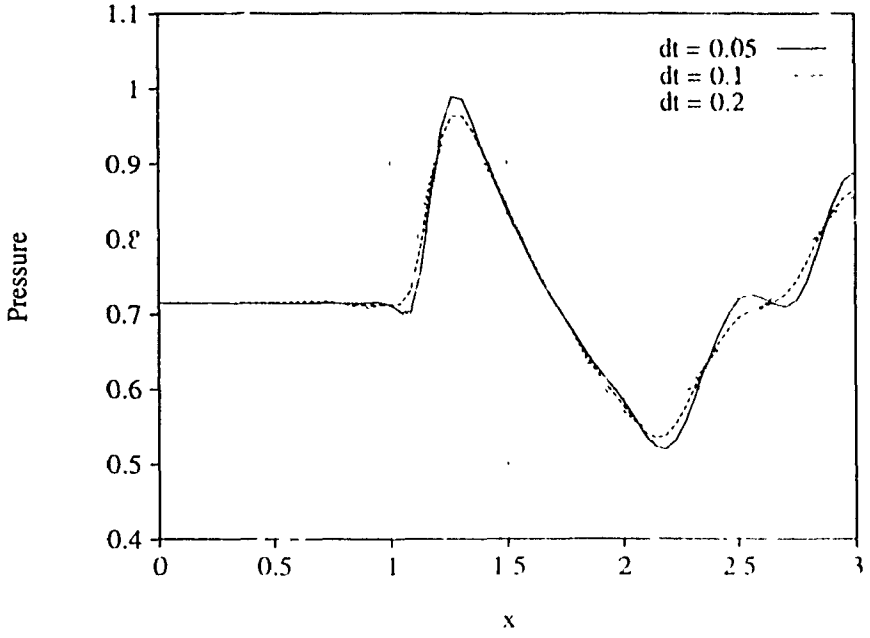

Figure 5.23: Pressure distribution for different time steps at y 0.2.

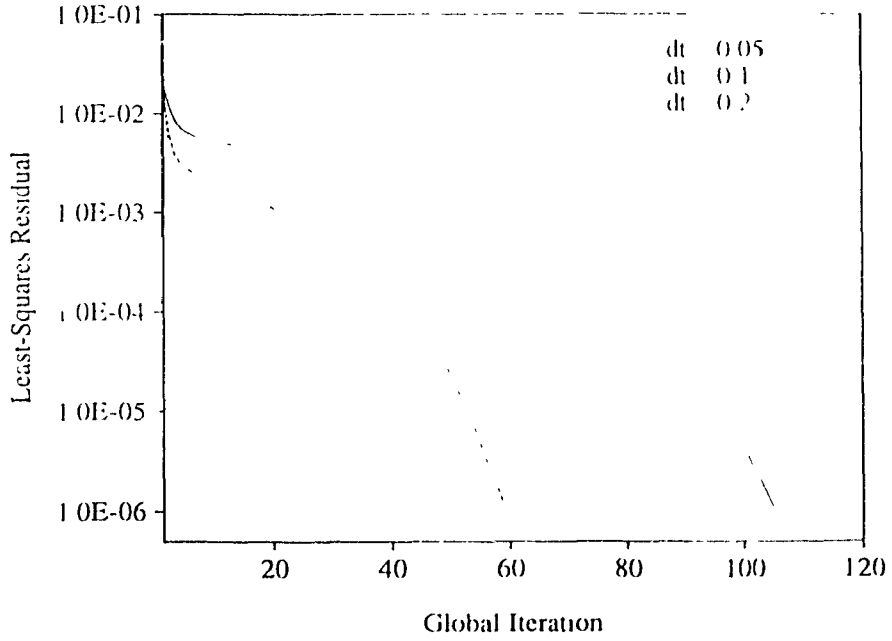

Figure 5.24: Convergence history for different time steps.

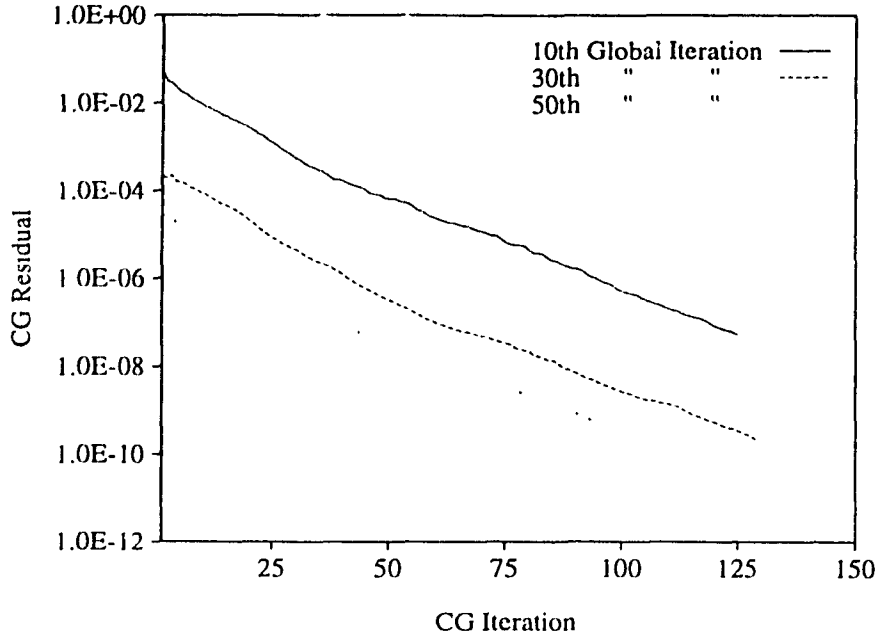

Figure 5.25: Convergence history of the Jacobi preconditioned Conjugate Gradient method (JCG-2), $\Delta t = 0.1$.

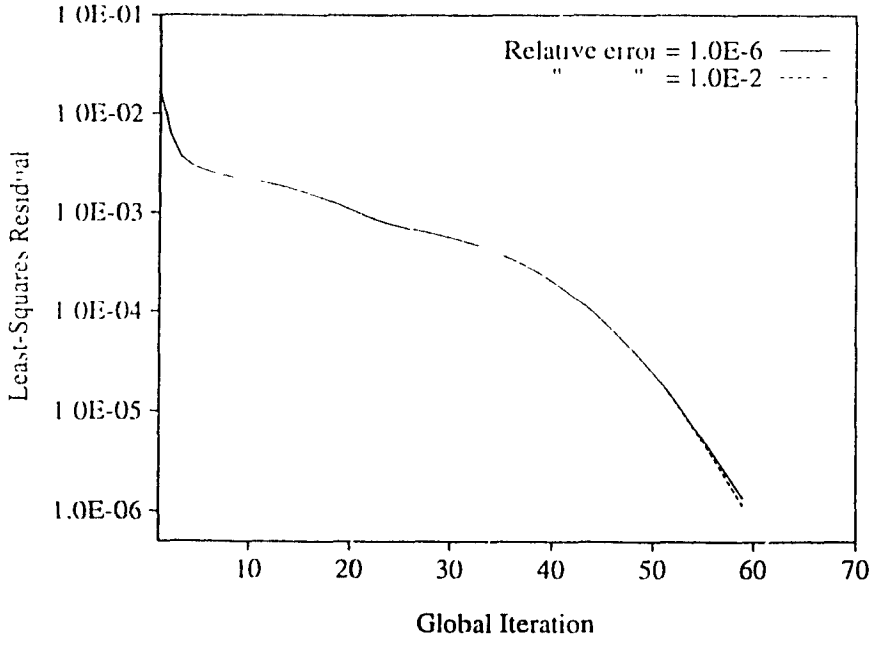

Figure 5.26: Convergence history of the Jacobi preconditioned Conjugate Gradient method before and after optimizing the solver parameters, $\Delta t = 0.1$.

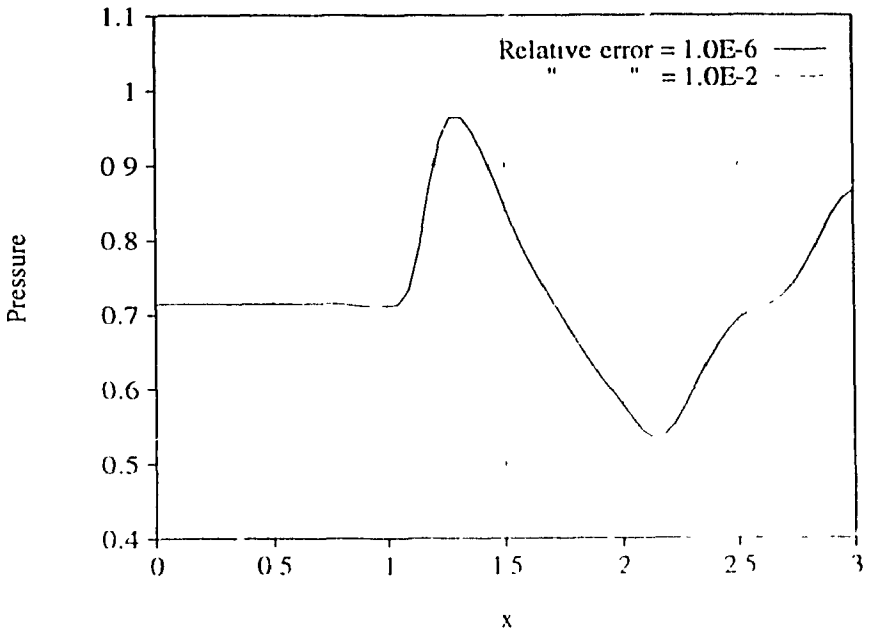

Figure 5.27: Pressure distribution at y 0.2 before and after optimizing the solver parameters, $\Delta t = 0.1$

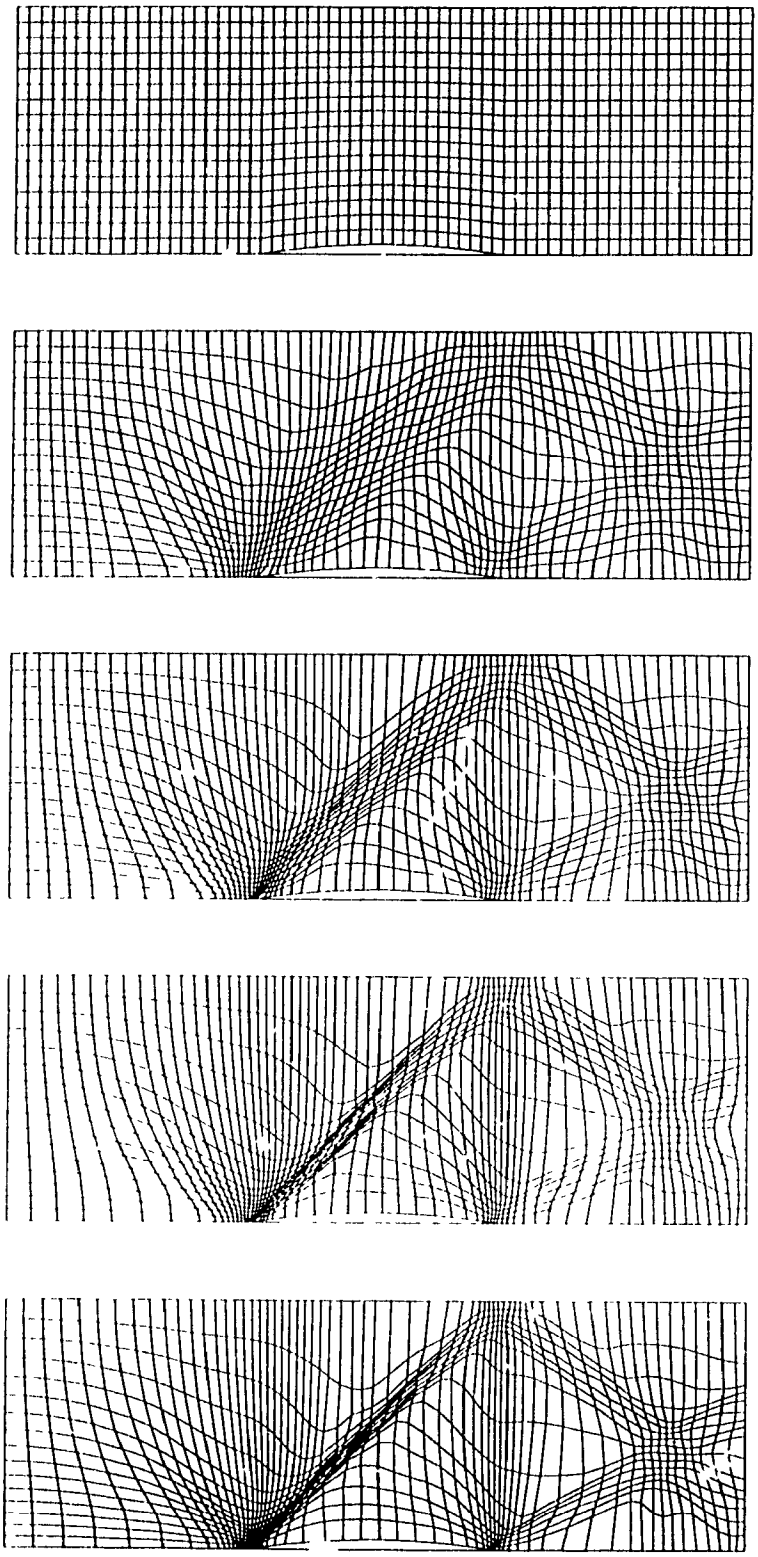

Figure 5.28: Original and adapted grids (after four cycles of adaptation), $\Delta t = 0.1$.

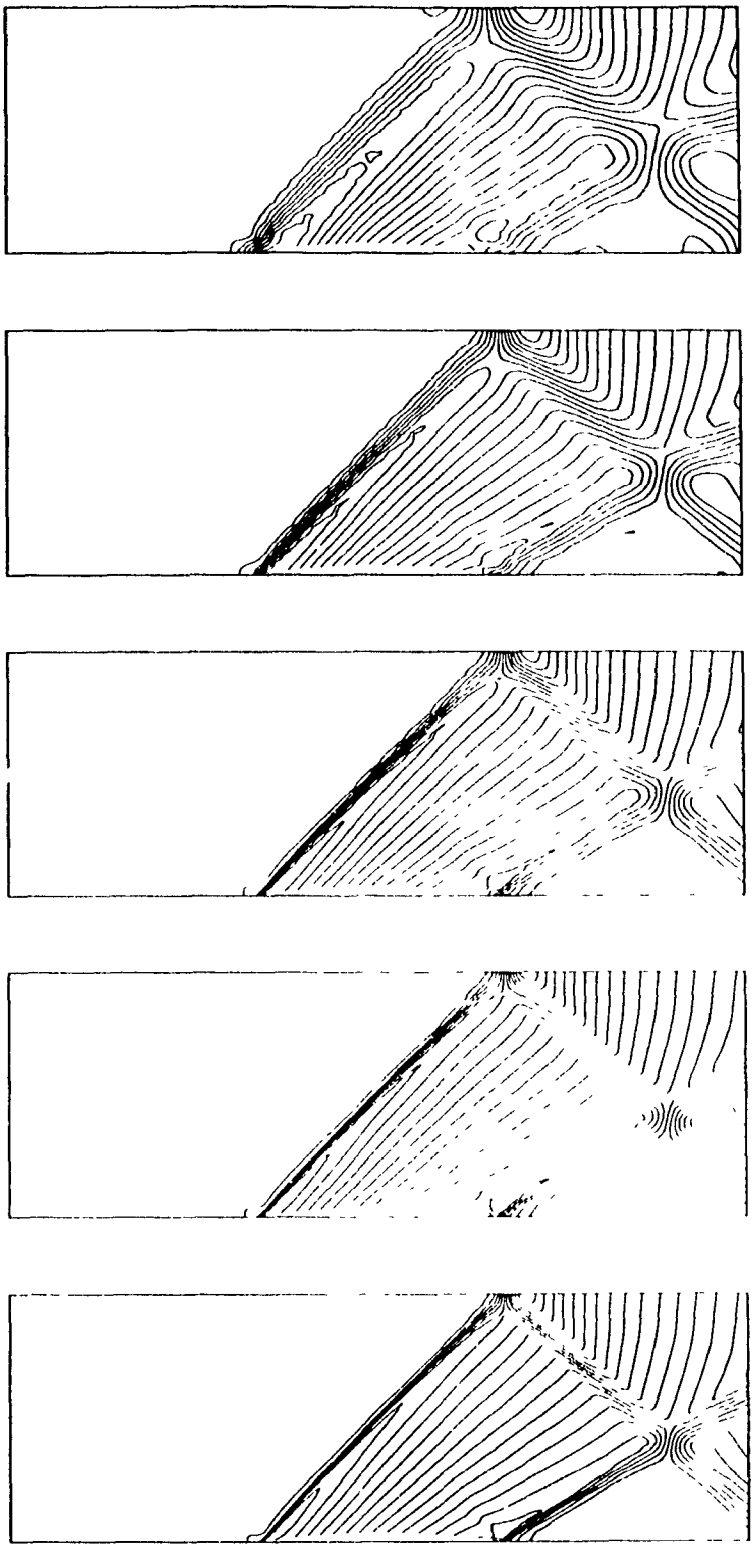

Figure 5.29: Pressure contours of the original and adapted solutions (after four cycles of adaptation), $\Delta t = 0.1$.

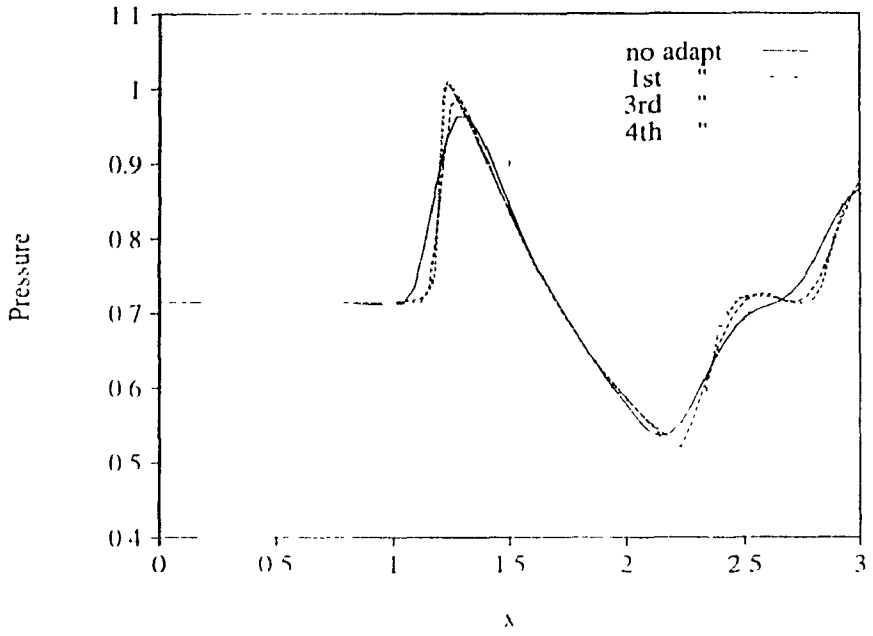

Figure 5.30 Pressure distribution at $y = 0.2$ after four cycles of adaptation, $\Delta t = 0.1$

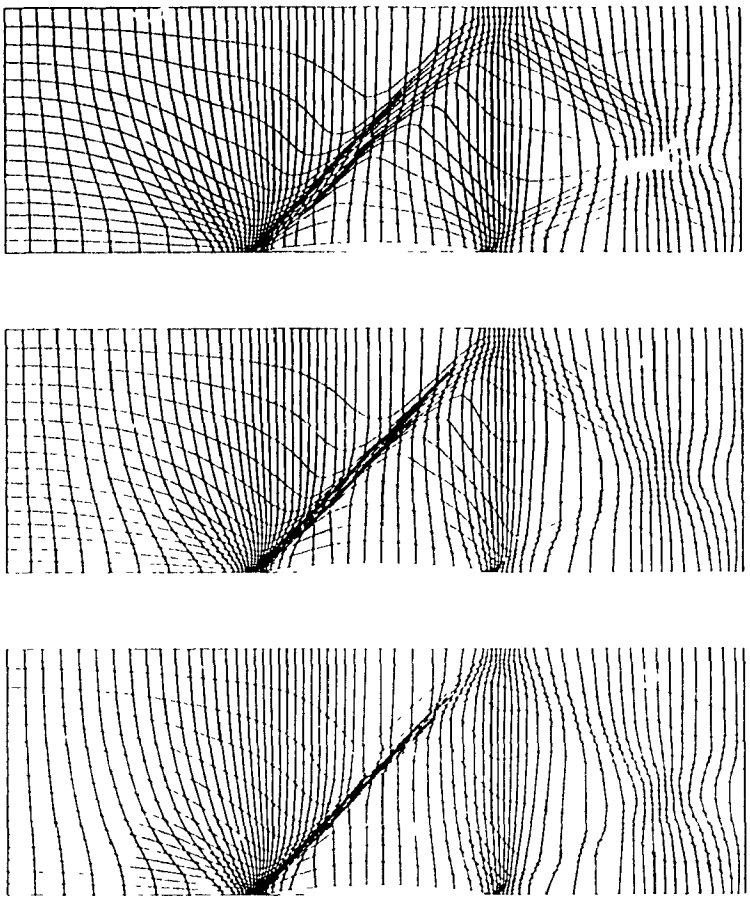

Figure 5.31: Adapted grids after reducing the artificial viscosity, $\Delta t = 0.05$

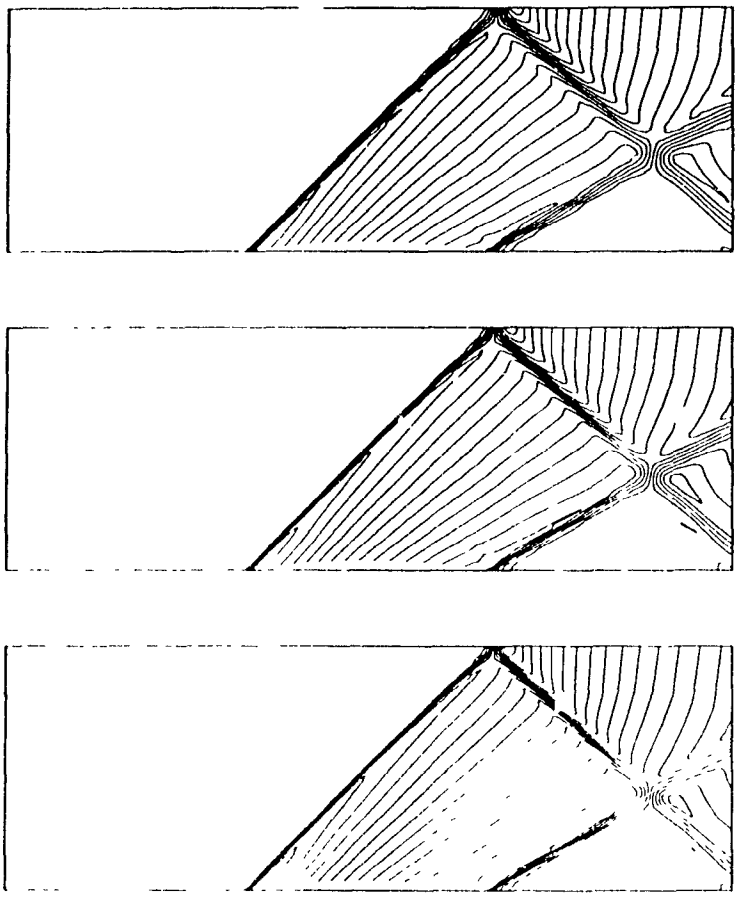

Figure 5.32 Pressure contours of adapted solution after reducing the artificial viscosity, $\Delta t = 0.05$

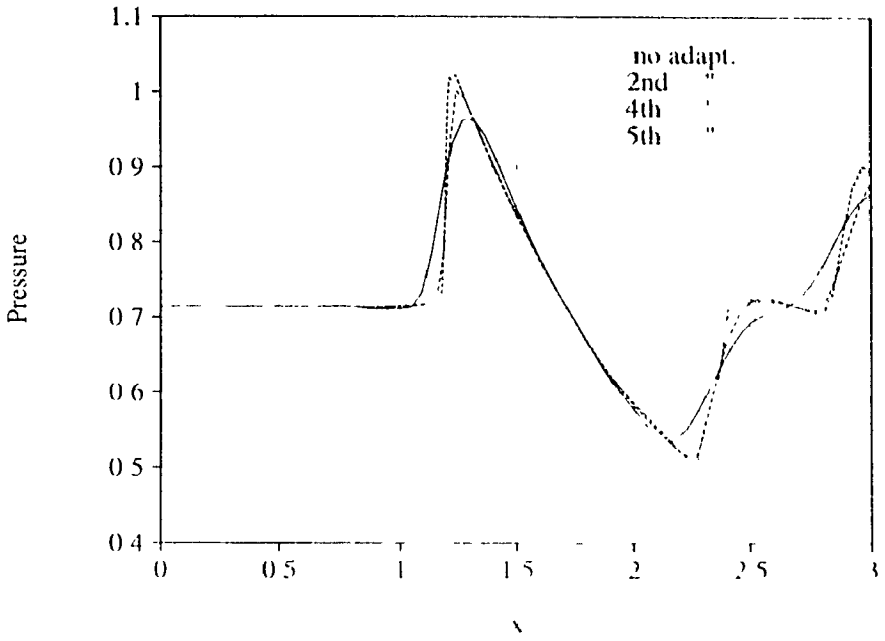

Figure 5 33: Pressure distribution at y 0 2 after reducing the artificial viscosity, Δt = 0.05.

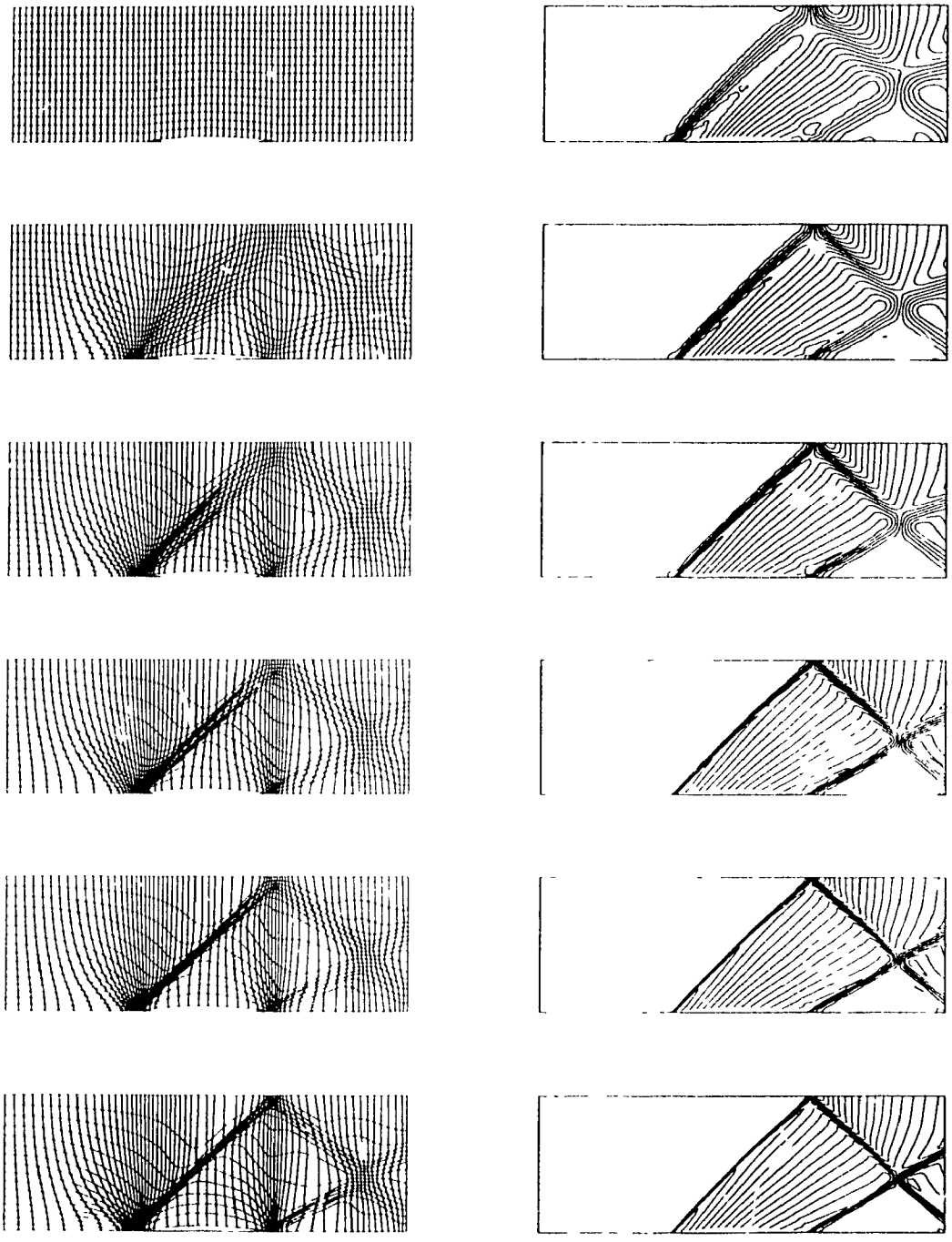

Figure 5.34: Evolution of the grid and the solution during the adaptation; $\Delta t = 0.05$ for the last three cycles.

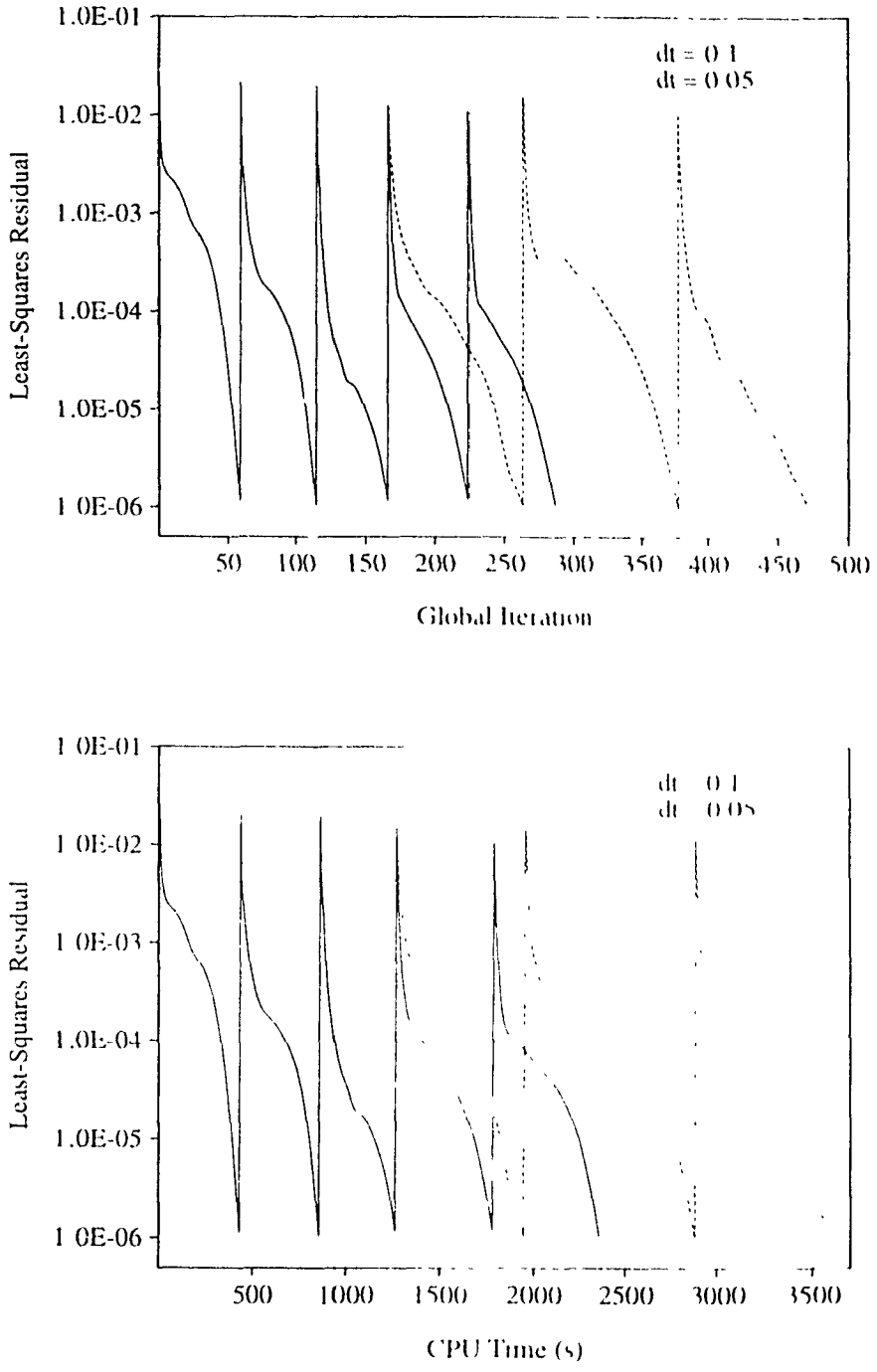

Figure 5.35: Convergence and time histories of the flow solver with adaptive procedure

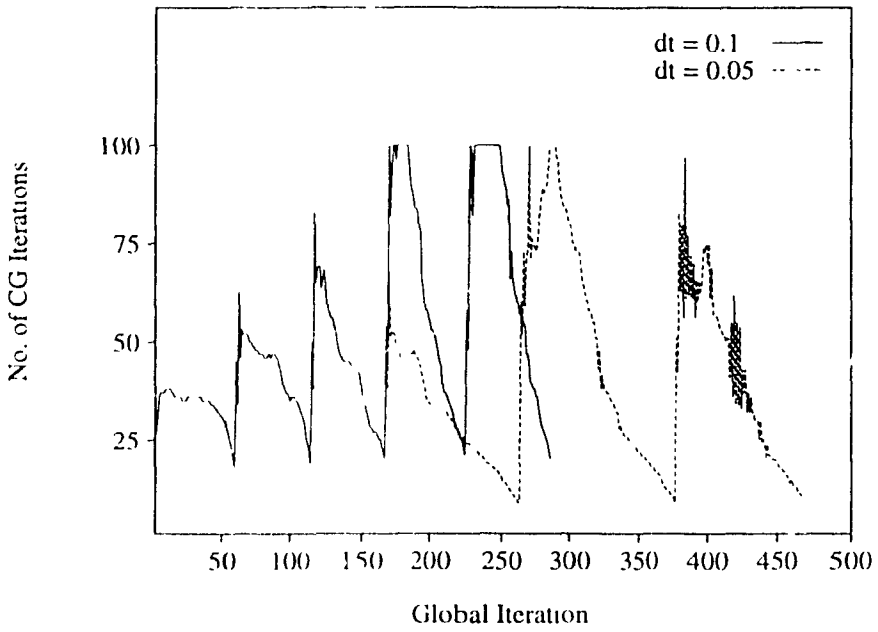

Figure 5.36 Convergence history of the iterative solver with adaptive procedure.

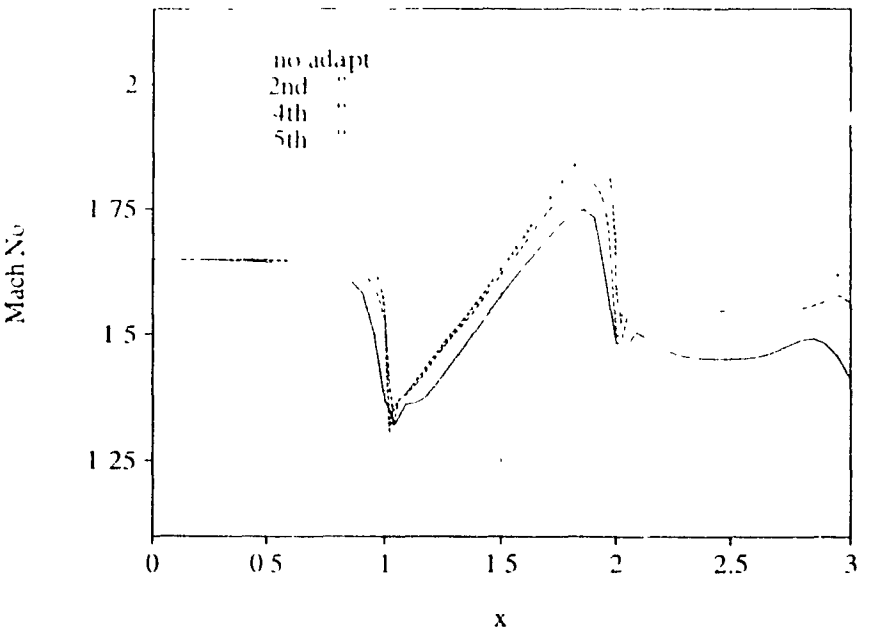

Figure 5.37: Mach number distribution on the lower wall before and after adaptation.

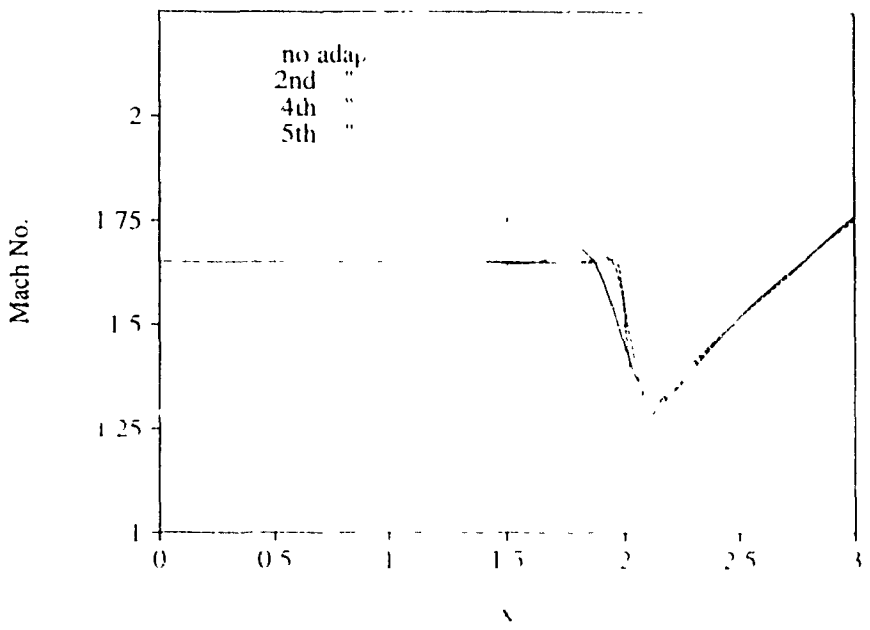

Figure 5.38: Mach number distribution on the upper wall before and after adaptation

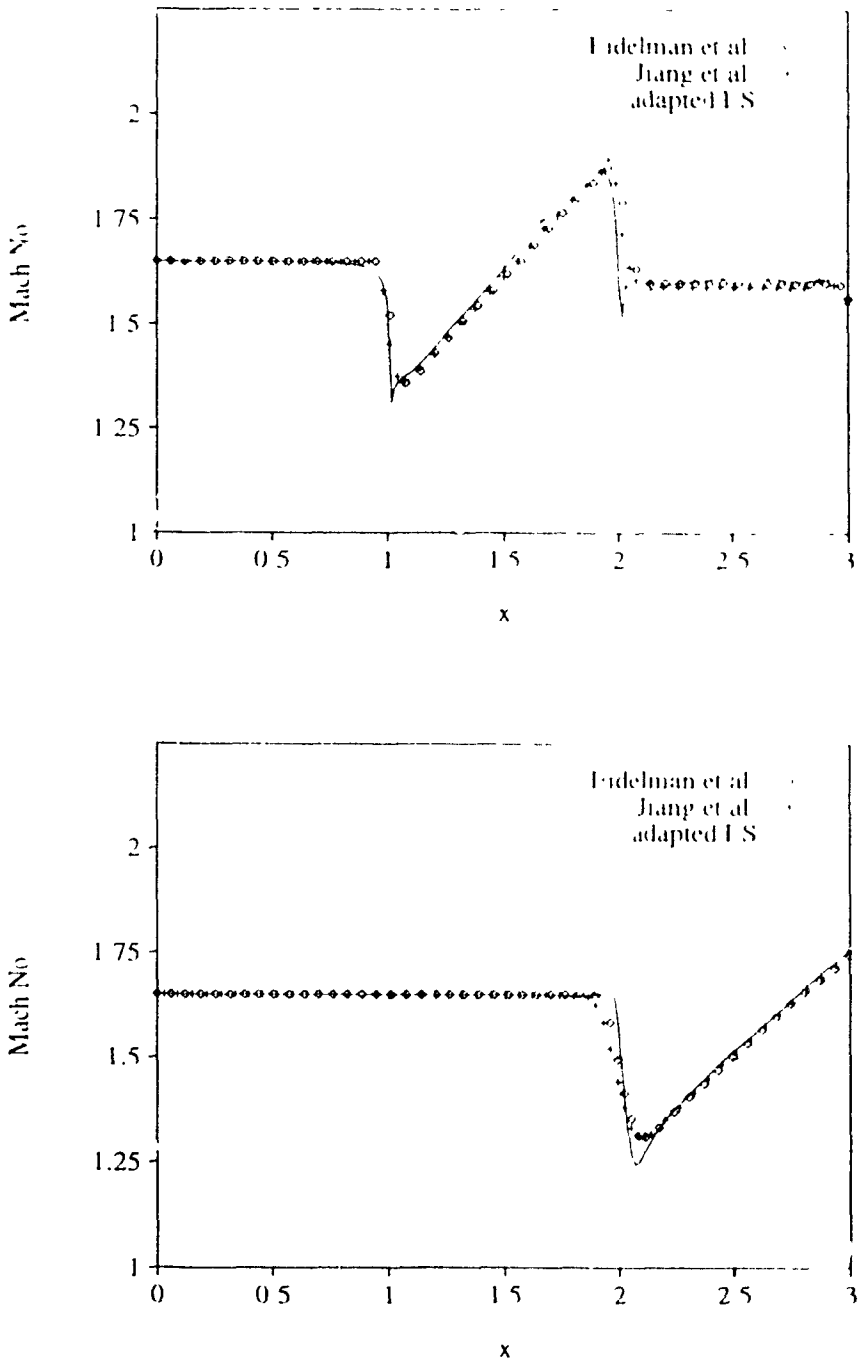

Figure 5.39: Mach number distribution on the lower and upper walls; comparison with the published data.

5.3 Transonic Channel Flow

The least squares method is also applied to the transonic flow in a channel. The computational domain is shown in Fig. 5.21 with a circular arc bump, having the following dimensions

$$t/c = 0.1 \qquad L = c = 1.0$$

The grid consists of bilinear rectangular elements, with 32 elements uniformly distributed on the bump, and 16 elements on each side of it centrally clustered with respect to the bump (Fig. 5.40). The height of the channel is divided into 16 elements which are clustered toward the bottom wall, with the minimum element size on the wall equal to 0.03125. The Mach number at inlet is $M_\infty = 0.675$, which is large enough to create a supersonic pocket on the bump followed by a shock. The inlet and exit boundary conditions are

$$\text{inlet} \begin{cases} \rho = 1.0 \\ u = 0.675 \\ v = 0.0 \end{cases} \qquad \text{exit} \begin{cases} p = 1.5282 \end{cases}$$

On the lower and upper walls, the no-penetration boundary condition is imposed.

To show the effect of artificial viscosity on the solution and also to select proper results for the adaptation, the test case is run for three different time steps. The pressure contours are plotted in Figure 5.41. The top solution corresponds to $\Delta t = 0.1$ where the shock is rather sharp. As the time step increases more artificial viscosity is added to the flow and, therefore, the shock becomes weaker and spreads over a larger number of elements. This is also demonstrated by the Mach number distribution on the lower wall shown in Figure 5.42. The shock Mach number is considerably reduced by increasing Δt. The convergence history of the flow solver is shown in Figure 5.43, and the corresponding CPU time in Table 5.5.

For the solution of the linear equations, the Jacobi preconditioned Conjugate Gradient method (JCG-2) is used with the following reduced tolerances:

$$\epsilon_{rel} = 10^{-2} \qquad , \qquad \epsilon_{abs} = 10^{-7}$$

Δt	CPU time (s)	No. of iterations
0.1	4484.2	628
0.2	2637.5	301
0.3	2131.2	209

Table 5.5 Computation time for different time steps

and the maximum number of solver iterations is set to 100. Based on the CPU time and the resolution of the shock, the solution for $\Delta t = 0.3$ is chosen for the adaptation, with the pressure as the key variable for error estimation.

The grids and corresponding solutions after two cycles of adaptation are shown in Figures 5.44 and 5.45. By adapting, the mesh becomes finer at the shock position and also at the leading- and trailing-edges, where there are singularities in the solution due to the sudden change in the flow direction. Despite having a finer mesh after the second adaptation, the shock resolution is almost the same as in the first adaptation, indicating a high amount of artificial viscosity. The artificial viscosity is lowered by reducing the time step from 0.3 to 0.1, and the computations are repeated on the second adapted mesh.

The pressure contours after reducing the artificial viscosity are shown in Figure 5.46. The Mach number distribution in Figure 5.47 shows the effects of both adaptation and reducing the artificial viscosity on the solution. The adaptation has little effect on the Mach number distribution on the upper wall since there is no significant change in the solution gradients. On the lower wall, however, the shock has become quite sharp and is captured in 6 adapted elements, equivalent to 2 elements of the original mesh. The convergence and time histories of the least-squares method including the adaptive procedure are shown in Figure 5.48.

The adapted results are compared to those of Ni [41] and Eidelman *et al.* [14] in Figure 5.49. The position of the shock is approximately at 72% of the chord, and the maximum Mach number is 1.323, both in very good agreement with these results. The deviation is, however, in the Mach number distribution after the shock. Figure 5.50 shows

the Mach contours for the final adapted case. It is seen that there is a viscous effect near the lower wall (limited to the first row of the elements) similar to a boundary layer. This viscous layer has little effect on the solution before the shock. After the shock, however, the flow becomes rotational near the wall and hence adversely affects this layer, leading to under-estimation of the Mach number.

Figure 5.51 shows that using larger Δt, i.e., more artificial viscosity, worsens the situation both ahead and after the shock. This is more evident from Fig. 5.47, where the Mach number profile behind the bump is elevated after reducing Δt in the second adaptive cycle. The figure also shows that as the shock becomes stronger, thus generating more vorticity, the tail branch of the Mach number profile is reduced.

To gain further insight into the effect of artificial viscosity on the accuracy of the solution near the wall, the mass flow from the solid boundaries was calculated by evaluating the following integral

$$I = \frac{\int_{\Gamma_w} (\vec{V} \cdot \vec{n})\, d\Gamma}{\int_{\Gamma} d\Gamma} \tag{5.3}$$

where Γ_w is the solid wall boundary. This integral is zero in an analytical method. In the numerical approach, however, it has some finite value representing the amount of error in numerical approximation at solid boundaries. For the final adapted solution where $\Delta t = 0.1$ (Fig. 5.49 or 5.50), this integral was of the order of $\mathcal{O}(10^{-1})$, and increased by half orders of magnitude after adding artificial viscosity, i.e., by increasing the time step from 0.1 to 0.3.

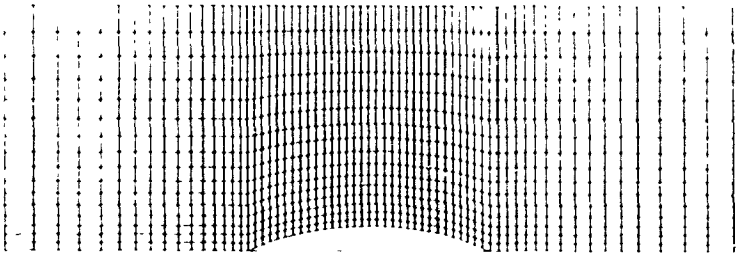

Figure 5.40: Computational grid for the transonic test case

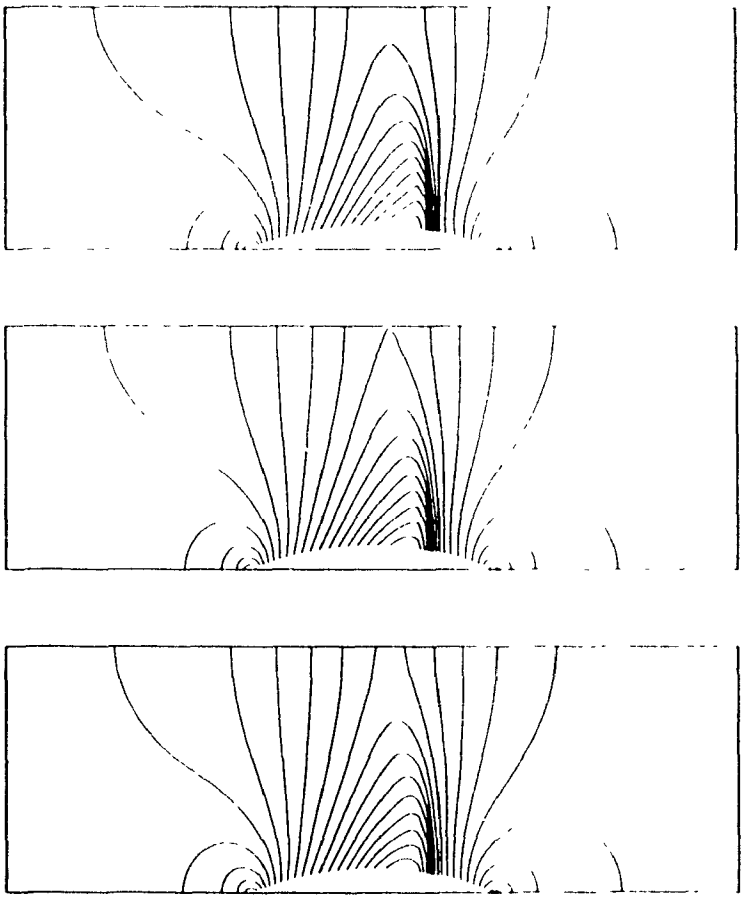

Figure 5.41: Pressure contours for different time steps, from top to bottom. Δt 0.1, 0.2, 0.3.

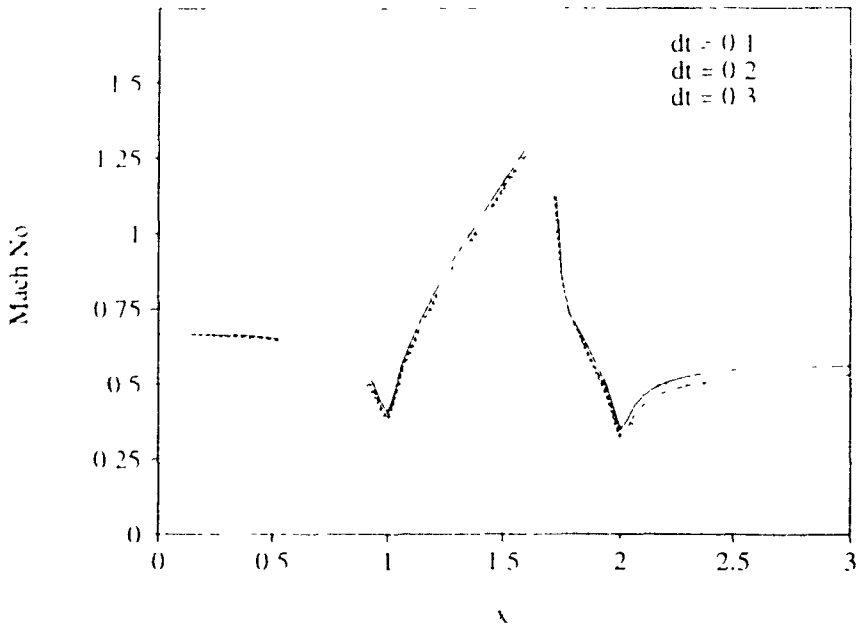

Figure 5.42. Mach number distribution on the lower wall for different time steps

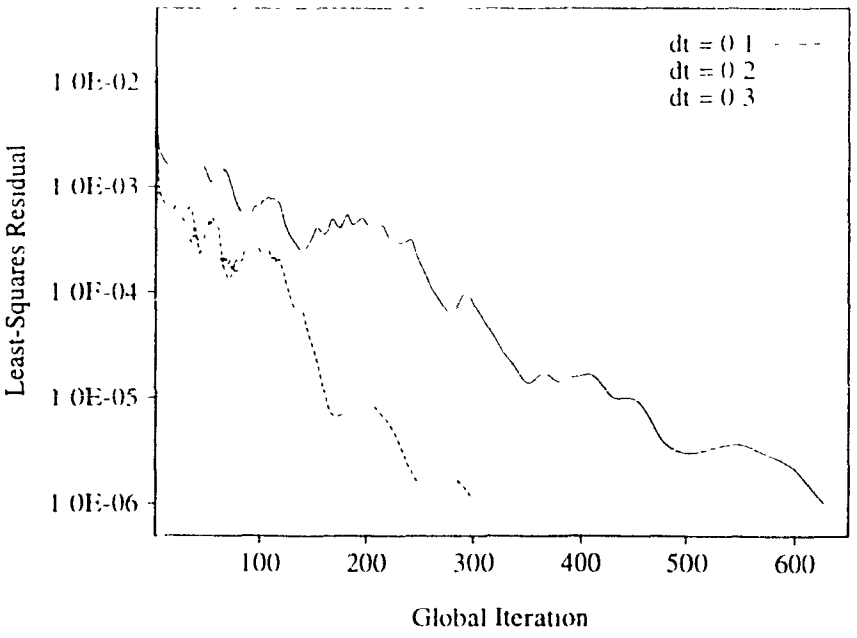

Figure 5.43: Convergence history for different time steps.

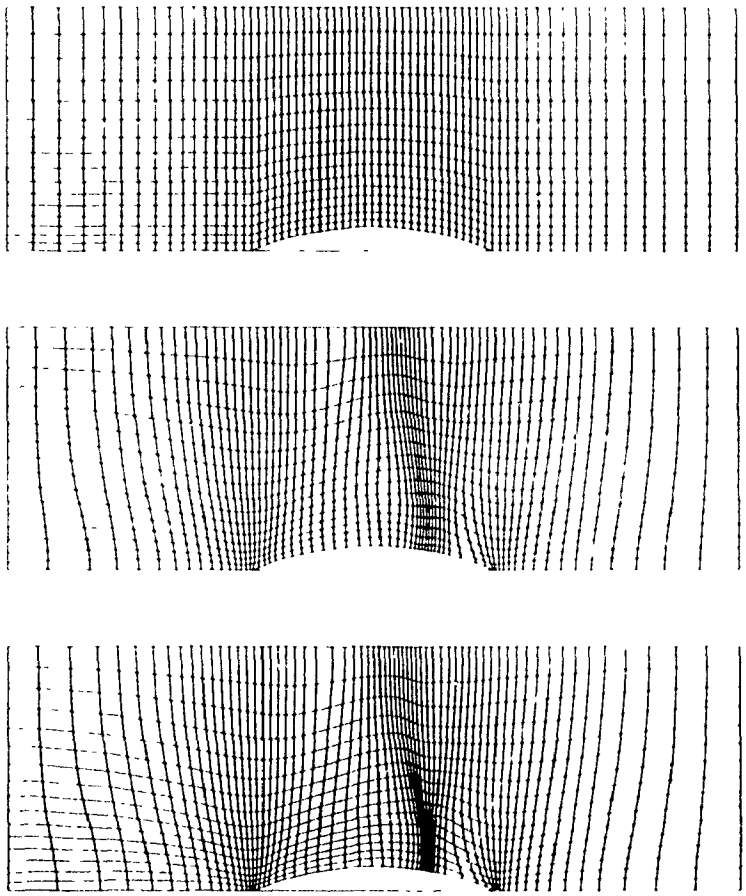

Figure 5.44: Original and adapted grids (after two cycles of adaptation)

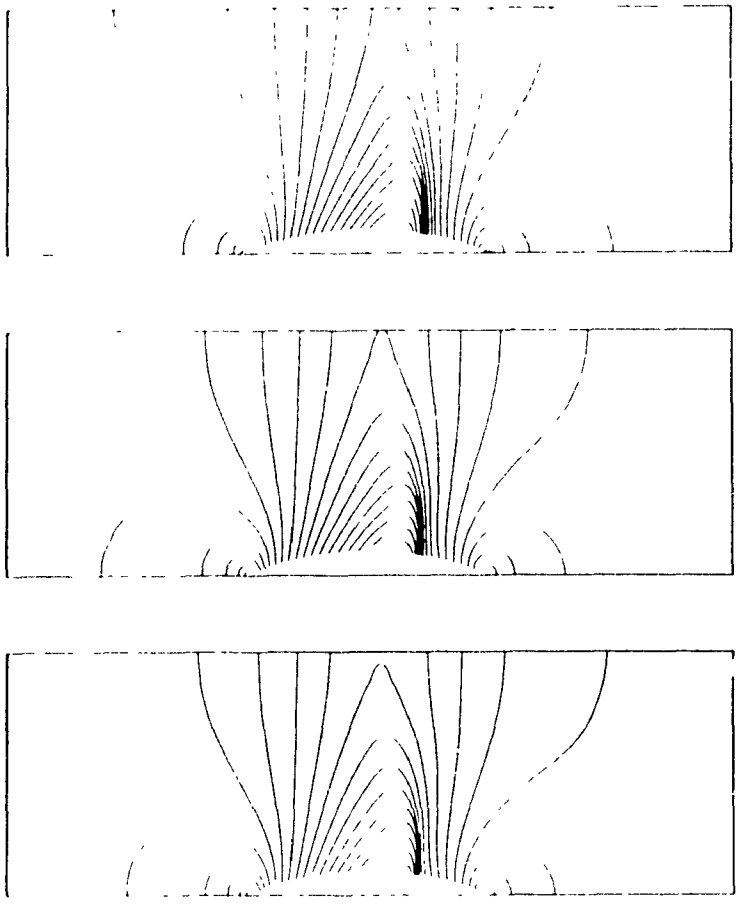

Figure 5.45: Pressure contours of the original and adapted solutions (after two cycles of adaptation), $\Delta t = 0.3$

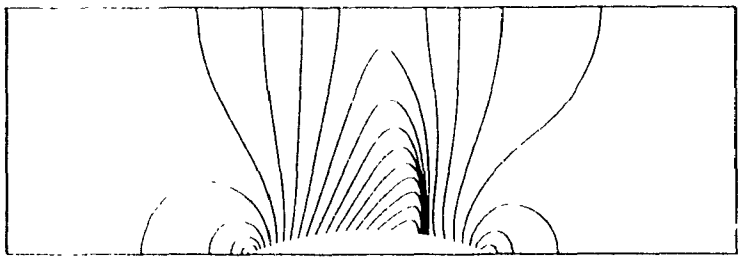

Figure 5.46: Pressure contours of the second adapted solution after reducing the artificial viscosity, $\Delta t = 0.1$.

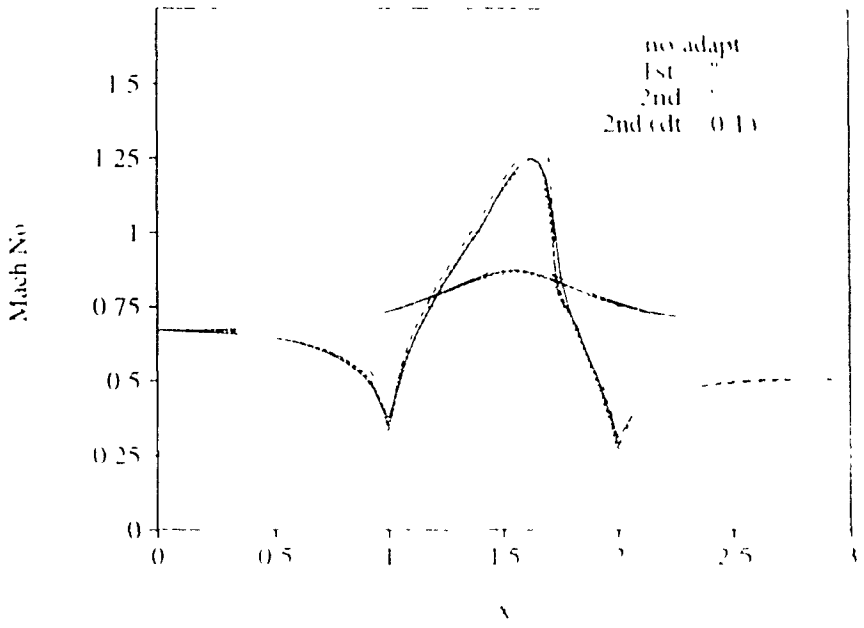

Figure 5.47. Mach number distribution on the lower and upper walls before and after adaptation

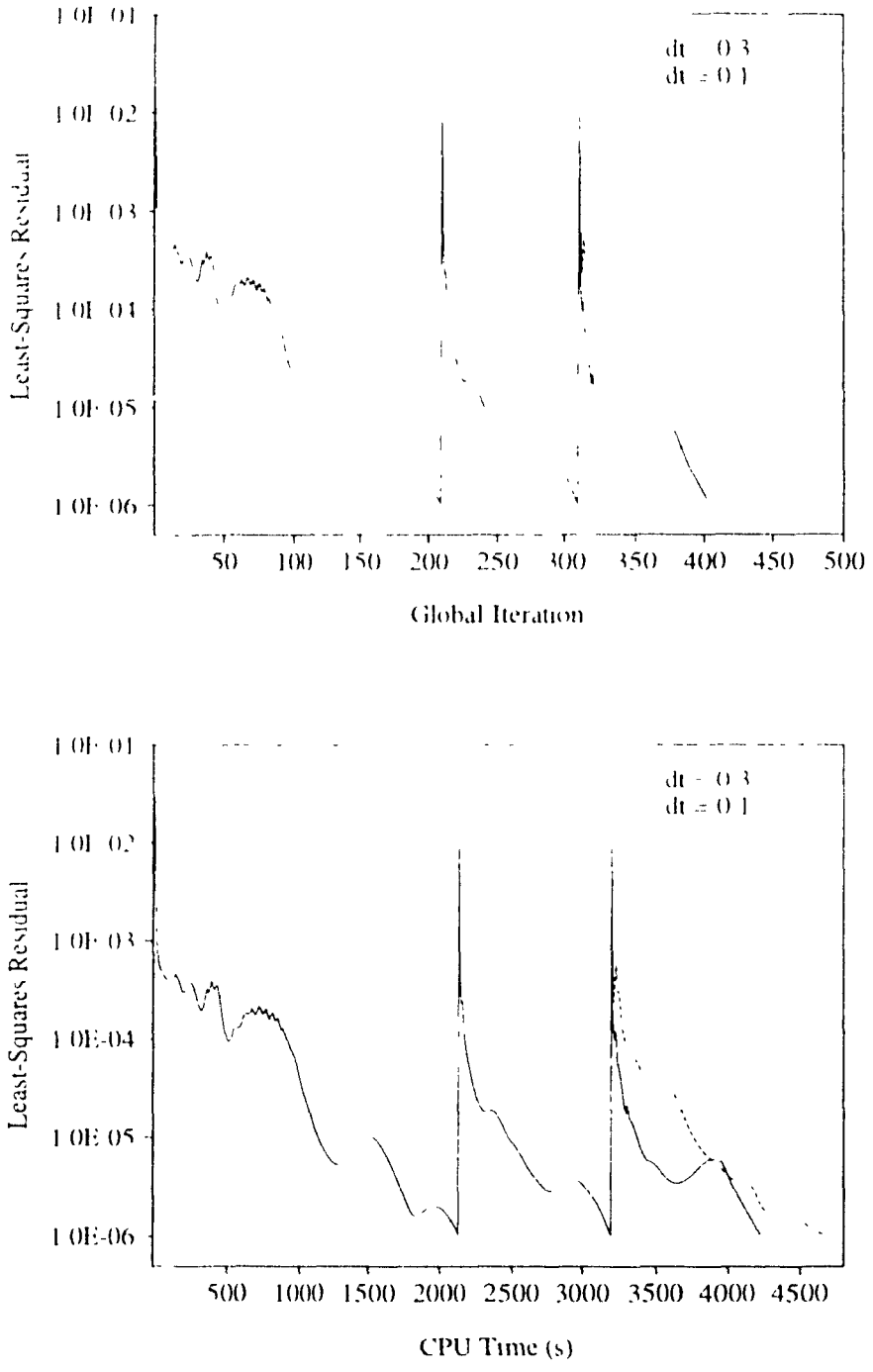

Figure 5.48: Convergence and time histories of the flow solver with adaptive procedure.

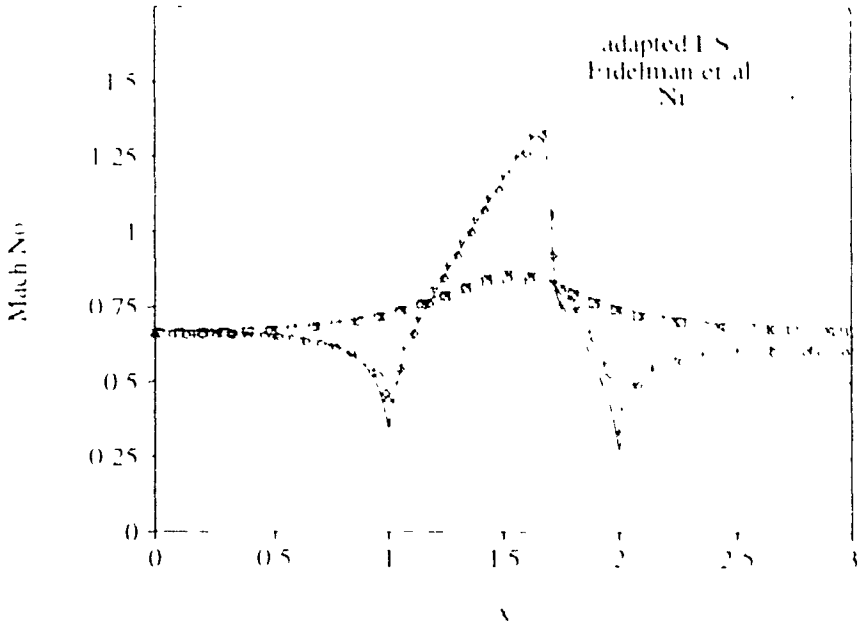

Figure 5.49 Mach number distribution on the lower and upper wall, comparison with the published data

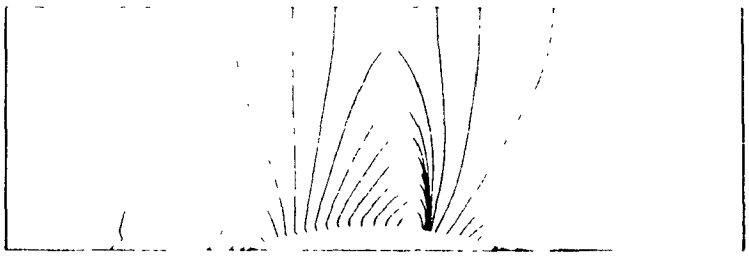

Figure 5.50: Iso-mach lines for the second adapted solution, Δl 0.1

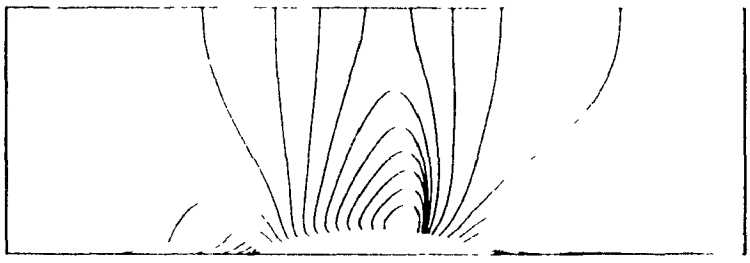

Figure 5.51: Iso-mach lines for the second adapted solution, Δl 0.3

Chapter 6

Conclusions and Future Work

In this thesis the least-squares finite element method was used to solve the compressible Euler equations. The Euler equations, written in the form of a first-order non-linear system of PDEs, are linearized using the Newton method. The application of the least-squares method to this first-order system leads to a system of linear algebraic equations, where the coefficient matrix is symmetric and positive-definite.

The symmetry and positive definite property, one of the major advantages of the least-squares method, allows the linear algebraic equations to be solved very efficiently using the Conjugate Gradient (CG) iterative method. To improve the convergence rate of the iterative solver, two different preconditioners, i.e., diagonal and incomplete Cholesky, were applied.

The emphasis was on the diagonal (or Jacobi) preconditioner, due to its ease of implementation and reduced cost in terms of both memory and CPU time. The diagonal preconditioner JCG-2, whose elements were constructed based on the L_1-norm of the row elements of the coefficient matrix, had better performance and, therefore, was the selected preconditioner.

It was shown that by assigning proper values to the user-defined parameters of the iterative solver, i.e., relative and absolute tolerances and maximum number of solver iterations, the computation time could be reduced by up to 50%. This significant improvement was achieved without any degradation of the solution accuracy.

The major goals of this thesis were 1) to study the ability of the least squares method in capturing shocks, and 2) the performance of the adaptive method and the level of accuracy that could be attained by adapting the solution. To accomplish this, two supersonic test cases including shock reflection and interaction, and a transonic test case were studied. The numerical results without adaptation show that the least squares method is able to correctly capture the shocks and their position. With the minimum amount of artificial viscosity, i.e., on the verge of having oscillations, the quality of the captured shocks was still moderate. This rather low quality is attributed to the artificial viscosity mechanism of the least-squares method, which is first-order with a constant coefficient.

To improve the accuracy, the least-squares method was coupled with an adaptive method, whose distinct feature is being sensitive to the oriented flow structures like shock waves. The results after the adaptation clearly show that, as expected, the elements are clustered near the shocks, with their edge(s) aligned with them. Significant improvements in the shock resolution after the adaptation for all test problems, indicate the robustness and very good performance of this directionally adaptive method, despite using rather coarse grids.

To improve the artificial viscosity mechanism of the least squares method, several options may be considered in the future work.

1. The Flux-Corrected-Transport (FCT) method may be combined with the least squares method to control the amount of artificial viscosity at discontinuities. This leads to a high-resolution scheme, where the shocks are clearly captured. For the Taylor-Galerkin method [12], which like the least-squares method has no free parameters and its artificial viscosity solely depends on the value of the time step, a similar combination was used by Lohner *et al.* [36], leading to accurate results.

2. Since the shock wave is a non-linear phenomenon, more accurate results would be obtained if the artificial viscosity mechanism is non-linear as well. This can be achieved by minimizing the H_1-norm of the equations residuals, which leads to

the addition of an extra artificial viscosity term proportional to the solution gradient rather than a constant amount as in the L_2 method.

3 A pseudo-time step may be defined as the coefficient of the artificial viscosity term, different from the physical time step used for the rest of the terms in the formulation. This pseudo-time step may contain free parameters and vary locally, thus controlling the amount of artificial viscosity added to the flow more accurately. Another advantage of using the pseudo-time step for the artificial viscosity term, is that one can use large time steps to speed up the convergence without deteriorating the solution accuracy by introducing large amounts of artificial viscosity.

The present method of applying the flow tangency boundary condition was found to be accurate. The other method of applying this type of boundary condition, i.e., the penalty or functional method (section 2.4.2) was also employed, leading to virtually the same results. The important point in employing this method is determination of the relative weight w which effects the overall accuracy of the solution. Zeitoun *et al.* [54] recently proposed a systematic method, which can be used to determine the optimal value of the weight.

Bibliography

[1] D. Ait-Ali-Yahia, W.G. Habashi, A. Tam, M.-G. Vallet and M. Fortin, 'A Directionally-Adaptive Finite Element Method for High-Speed Flows,' *AIAA paper 96-2553.*

[2] O. Axelsson, *Iterative Solution Methods*, Cambridge University Press, Cambridge, 1994.

[3] G.S Baruzzi, W.G Habashi and M.M. Hafez, 'Finite Element Solutions of the Euler Equations for Transonic External Flows,' *AIAA J.*, vol. 29, pp 1886-1893, 1991

[4] G.S. Baruzzi, W.G. Habashi and M M Hafez, 'A Second Order Finite Element Method for the Solution of the Transonic Euler and Navier Stokes Equations,' *Int J. Num. Meth. Fluids*, vol. 20, pp. 671-693, 1995

[5] G.J. Le Beau, S.E. Ray, S.K. Aliabadi and T.E. Tezduyar, 'SUPG Finite Element Computation of Compressible Flows with the Entropy and Conservation Variables Formulations,' *Comp. Meth. Appl. Mech. Eng.*, vol. 104, pp. 397-422, 1993.

[6] A.L. Brooks and T.J.R. Hughes, 'Streamline Upwind Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations,' *Comp. Meth. Appl. Mech. Eng.*, vol. 32, pp. 199-259, 1982.

[7] Ch. H. Bruneau, J. Laminie and J.J. Chattot, 'Computation of 3D Vortex Flows Past a Flat Plate at Incidence Through A Variational Approach of the Full Steady Euler Equations,' *Int. J. Num. Meth. Fluids*, vol. 9, pp. 305-323, 1989.

[8] G.F. Carey and B.-N. Jiang, 'Least-Squares Finite Element Method and Preconditioned Conjugate Gradient Solution,' *Int. J. Num. Meth. Eng.*, vol. 24, pp. 1283-1296, 1987.

[9] G.F. Carey and B.-N. Jiang, 'Least-Squares Finite Elements for First-Order Hyperbolic Systems,' *Int. J. Num. Meth. Eng.*, vol. 26, pp. 81-93, 1988.

[10] I. Christie, D.F. Griffiths, A.R. Mitchell and O.C. Zienkiewicz, 'Finite Element Methods for Second Order Differential Equations with Significant First Derivatives,' *Int. J. Num. Meth. Eng.*, vol. 10, pp. 1389-1396, 1976.

[11] G. Comini, M. Manzan and C. Nonino, 'Analysis of Finite Element Schemes for Convection-Type Problems,' *Int. J. Num. Meth. Fluids*, vol. 20, pp. 443-458, 1995.

[12] J Donea, 'A Taylor-Galerkin Method for Convective Transport Problems,' *Int. J. Num. Meth. Eng.*, vol. 20, pp. 101-119, 1984.

[13] L.C. Dutto, 'On the Iterative Methods for Solving Linear Systems of Equations,' *Revue Européenne des Éléments Finis*, vol. 2, pp. 423-448, 1993.

[14] Sh. Eidelman, Ph. Colella and R.P. Shreeve, 'Application of the Godunov Method and Its Second-Order Extension to Cascade Flow Modeling,' *AIAA J.*, vol. 22, pp. 1609-1615, 1984.

[15] C.A.J. Fletcher, 'A Primitive Variable Finite Element Formulation for Inviscid Compressible Flow,' *J. Comp. Phy.*, vol. 33, pp. 301-312, 1979.

[16] P.A. Gnoffo, 'A Finite-Volume, Adaptive Grid Algorithm Applied to Planetary Entry Flowfields,' *AIAA J.*, vol. 21, pp. 1249-1254, 1983.

[17] M.D. Gunzburger, *Finite Element Methods for Viscous Incompressible Flows, A Guide to Theory, Practice, and Algorithms*, Academic Press, San Diego, CA, 1989.

[18] Y. Hasbani, E. Livne and M. Bercovier, 'Finite Elements and Characteristics Applied to Advection-Diffusion Equations,' *Computers and Fluids*, vol. 11, pp. 71-83, 1983.

[19] J.C. Heinrich, P.S. Huyakoran, O.C. Zienkiewicz and A.R. Mitchell, 'An 'Upwind' Finite Element Scheme for Two-Dimensional Convective Transport Equation,' *Int. J. Num. Meth. Eng.*, vol. 11, pp. 131-143, 1977.

[20] D.L. Hill and E.A. Baskharone, 'A Monotone Streamline Upwind Method for Quadratic Finite Elements,' *Int. J. Num. Meth. Fluids*, vol. 17, pp. 463-475, 1993.

[21] Ch. Hirsch, *Numerical Computation of Internal and External Flows, vol.2*, Wiley, Chichester, 1990.

[22] L.-J. Hou, 'A Time-Accurate Least-Squares Finite Element Method for Incompressible Flow,' *AIAA paper 95-0081*.

[23] T.J.R. Hughes, 'A Simple Scheme for Developing 'Upwind' Finite Elements,' *Int J Num. Meth. Eng.*, vol. 12, pp. 1359-1365, 1978.

[24] T.J.R. Hughes and T.E. Tezduyar, 'Finite Element Methods for First-Order Hyperbolic Systems with Particular Emphasis on the Compressible Euler Equations,' *Comp Meth. Appl. Mech. Eng.*, vol. 45, pp. 217-284, 1984.

[25] T.J.R. Hughes, M. Mallet and A. Mizukami, 'A New Finite Element Formulation for Computational Fluid Dynamics: II. Beyond SUPG,' *Comp. Meth. Appl. Mech. Eng.*, vol. 54, pp. 341-355, 1986.

[26] O.K. Jensen and B.A. Finlayson, 'Oscillation Limits for Weighted Residual Methods Applied to Convective Diffusion Equations,' *Int. J. Num. Meth. Eng.*, vol. 15, pp. 1681-1689, 1980.

[27] B.-N. Jiang and G.F. Carey, 'Adaptive Refinement for Least-Squares Finite Elements with Element-by-Element Conjugate Gradient Solution,' *Int. J. Num. Meth. Eng.*, vol. 24, pp. 569-580 , 1987.

[28] B.-N Jiang and G.F. Carey, 'A Stable Least-Squares Finite Element Method for Non-Linear Hyperbolic Problems,' *Int. J Num. Meth. Fluids*, vol. 8, pp. 933-942, 1988.

[29] B.-N. Jiang and G.F. Carey, 'Least-Squares Finite Element Methods for Compressible Euler Equations,' *Int. J. Num. Meth. Fluids*, vol. 10, pp 557-568, 1990.

[30] B.-N Jiang and L.A. Povinelli, 'Least-Squares Finite Element Method for Fluid Dynamics,' *Comp. Meth. Appl. Mech. Eng.*, vol. 81, pp. 13-37, 1990.

[31] Y. Jiang, C.P. Chen and H.M. Shang, 'A New Pressure-Velocity Coupling Procedure for Inviscid and Viscous Flows at All Speeds,' in *Proceedings of the Fourth Int. Sym. on Computational Fluid Dynamics*, vol. 1, pp. 545-550, University of California, Davis, Sep. 9-12, 1991.

[32] D.W Kelly, S. Nakazawa, O.C Zienkiewicz and J.C. Heinrich, 'A Note on Upwinding and Anisotropic Balancing Dissipation in Finite Element Approximations to Convective Diffusion Problems,' *Int. J Num. Meth. Eng.*, vol 15, pp. 1705-1711, 1980.

[33] D Lefebvre and J. Peraire, 'Finite Element Least Squares Solution of the Euler Equations Using Linear and Quadratic Approximations,' *Int J. Comp. Fluid Dyn* , vol. 1, pp. 1-23, 1993.

[34] C.W. Li, 'Least-Squares Characteristics and Finite Elements for Advection-Dispersion Simulation,' *Int. J. Num. Meth. Eng.*, vol. 29, pp. 1343-1358, 1990.

[35] R. Löhner, K. Morgan and O.C. Zienkiewicz, 'The Solution of Non-Linear Hyperbolic Equation Systems by the Finite Element Method,' *Int. J. Num. Meth. Fluids*, vol. 4, pp. 1043-1063, 1984.

[36] R. Löhner, K. Morgan, J. Peraire and M. Vahdati, 'Finite Element Flux-Corrected Transport (FEM-FCT) for the Euler and Navier-Stokes Equations,' in *Finite Elements in Fluids* (R.H. Gallagher, R. Glowinski, P.M. Gresho, J.T. Oden and O.C. Zienkiewicz Eds.), vol. 7, pp. 105-121, Wiley, 1987.

[37] P.P. Lynn and S.K. Arya, 'Use of the Least Squares Criterion in the Finite Element Formulation,' *Int. J. Num. Meth. Eng.*, vol. 6, pp 75-88, 1973.

[38] K.W. Morton, 'Generalised Galerkin Methods for Hyperbolic Problems,' *Comp Meth. Appl. Mech. Eng.*, vol. 52, pp. 847-871, 1985.

[39] K. Nakahashi and G.S. Diewert, 'Self-Adaptive-Grid Method with Application to Airfoil Flow,' *AIAA J.*, vol. 25, pp. 513-520, 1987.

[40] H. Nguyen and J. Reynen, 'A Space-Time Least-Square Finite Element Scheme for Advection-Diffusion Equations.' *Comp. Meth. Appl. Mech. Eng.*, vol. 42, pp. 331-342, 1984.

[41] R.-H. Ni, 'A Multiple Grid Scheme for Solving the Euler Equations,' *AIAA J.*, vol 20, pp. 1565-1571, 1982.

[42] J.T. Oden, T Strouboulis and Ph Devloo, 'Adaptive Finite Element Methods for High-Speed Compressible Flows,' *Int. J. Num. Meth. Fluids*, vol 7, pp 1211-1228, 1987.

[43] B. Palmerio, 'A Two-Dimensional FEM Adaptive Moving-Node Method for Steady Euler Flow Simulations,' *Comp. Meth Appl. Mech. Eng* , vol 71, pp. 315 340, 1988

[44] N.-S. Park and J.A. Liggett, 'Taylor-Least-Squares Finite Element for Two Dimensional Advection-Dominated Unsteady Advection-Diffusion Problems,' *Int. J Num. Meth. Fluids*, vol. 11, pp. 21-38, 1990.

[45] J. Peraire, J. Peiro, L. Formaggia, K. Morgan and O.C. Zienkiewicz, 'Finite Element Euler Computations in Three Dimensions,' *Int. J. Num. Meth. Eng.*, vol. 26, pp. 2135-2159, 1988.

[46] J. Peraire, M. Vahdati, K. Morgan and O.C. Zienkiewicz, 'Adaptive Remeshing for Compressible Flow Computations,' *J. Comp. Phy.*, vol. 72, pp. 449-466, 1987.

[47] J Peraire, J Peiro and K. Morgan, 'Adaptive Remeshing for Three-Dimensional Compressible Flow Computations,' *J. Comp. Phy*, vol. 103, pp. 269-285, 1992.

[48] J F. Polk and P.P. Lynn, 'A Least Squares Finite Element Approach to Unsteady Gas Dynamics,' *Int. J. Num. Meth. Eng.*, vol. 12, pp 3-10, 1978

[49] C. Pommerell, *Solution of Large Unsymmetric systems of Linear Equations* Series in Microelectronics, vol. 17, Hartung-Gorre, 1992.

[50] J N. Reddy, *An Introduction to the Finite Element Method*, McGraw-Hill, New York, 1993.

[51] J G Rice and R.J. Schnipke, 'A Monotone Streamline Upwind Finite Element Method for Convection-Dominated Flows,' *Comp. Meth. Appl Mech. Eng.*, vol. 48, pp. 313-327, 1985

[52] A Soulaimani and M. Fortin, 'Finite Element Solution of Viscous Flows Using Conservative Variables,' *Comp Meth Appl. Mech. Eng.*, vol 118, pp 319-350, 1994.

[53] L Q Tang and T.T.H. Tsang, 'An Efficient Least-Squares Finite Element Method for Incompressible Flows and Transport Processes,' *Int. J. Comp Fluid Dyn.*, vol. 4, pp. 21-39, 1995.

[54] D.G. Zeitoun, J.P. Laible and G.F. Pinder, 'A Weighted least Squares Method for First-Order Hyperbolic Systems,' *Int. J. Num. Meth. Fluids*, vol. 20, pp. 191-212, 1995.

[55] O.C. Zienkiewicz, J. Szmelter and J. Peraire, 'Compressible and Incompressible Flow; An Algorithm for All Seasons,' *Comp. Meth. Appl. Mech. Eng.*, vol. 78, pp. 105-121, 1990.

[56] O.C. Zienkiewicz and J. Wu, 'A General Explicit or Semi-Explicit Algorithm for Compressible and Incompressible Flows,' *Int. J. Num. Meth. Eng.*, vol. 35, pp. 457-479, 1992.

Appendix A

Derivation of the Euler-Lagrange Equation

The Euler-Lagrange equation corresponding to the weak form:

$$\int \left(u^{n+1} - u^n + \Delta t A^n \frac{\partial u^{n+1}}{\partial x} \right) \left(1 + \Delta t A^n \frac{\partial}{\partial x} \right) w \, dx = 0 \qquad (A.1)$$

is derived by expanding the terms and integrating by parts as follows

$$\begin{aligned} 0 &= \int \left(u^{n+1} - u^n + \Delta t A^n u_x^{n+1} \right) w \, dx + \int \left(u^{n+1} - u^n + \Delta t A^n u_t^{n+1} \right) \Delta t A^n \frac{\partial w}{\partial t} \, dt \\ &= \int \left(u^{n+1} - u^n + \Delta t A^n u_t^{n+1} \right) w \, dx - \int \Delta t A^n w \frac{\partial}{\partial x} \left(u^{n+1} - u^n + \Delta t A^n u_t^{n+1} \right) dt \\ &\quad + \text{boundary integral} \\ &= \int \left[\left(1 - \Delta t A^n \frac{\partial}{\partial x} \right) \left(u^{n+1} - u^n + \Delta t A^n u_t^{n+1} \right) \right] w \, dt + \text{boundary integral} \qquad (A.2) \end{aligned}$$

Setting the terms inside the bracket equal to zero gives the Euler-Lagrange equation

$$\begin{aligned} 0 &= \left(1 - \Delta t A^n \frac{\partial}{\partial x} \right) \left(u^{n+1} - u^n + \Delta t A^n u_x^{n+1} \right) \\ &= \left(1 - \Delta t A^n \frac{\partial}{\partial x} \right) \left[\left(1 + \Delta t A^n \frac{\partial}{\partial x} \right) u^{n+1} - u^n \right] \\ &= \left(1 - \Delta t A^n \frac{\partial}{\partial x} \right) \left[\left(1 + \Delta t A^n \frac{\partial}{\partial x} \right) \left(u^{n+1} - u^n \right) + \Delta t A^n \frac{\partial u^n}{\partial x} \right] \end{aligned}$$

$$\left(1 \quad (\Delta t A^n)^2 \frac{\partial^2}{\partial x^2}\right)\left(\frac{u^{n+1} - u^n}{\Delta t}\right) + \left(1 - \Delta t A^n \frac{\partial}{\partial x}\right) A^n \frac{\partial u^n}{\partial x}$$

or

$$\frac{u^{n+1}}{\Delta t} \frac{u^n}{} + A^n \frac{\partial u^n}{\partial x} = \Delta t (A^n)^2 \frac{\partial^2 u^{n+1}}{\partial x^2} \tag{A.3}$$

Appendix B

Derivation of the Energy Equation in Terms of Pressure

For the Euler equations, where the fluid is considered to be inviscid and non heat conducting, the energy equation can be written in the following non conservative form

$$\frac{De}{Dt} = \frac{p}{\rho^2}\frac{D\rho}{Dt} \qquad \text{(B 1)}$$

where e is the internal energy per unit mass, ρ is the density, and p is the pressure. Using the state relation

$$e = e(p, \rho) \qquad \text{(B 2)}$$

one can write

$$\frac{De}{Dt} = \left(\frac{\partial e}{\partial p}\right)_{\rho}\frac{Dp}{Dt} + \left(\frac{\partial e}{\partial \rho}\right)_{p}\frac{D\rho}{Dt} \qquad \text{(B 3)}$$

Substituting for De/Dt from Eq (B 2) into Eq (B 1), and noting that from the continuity equation,

$$\frac{D\rho}{Dt} = -\rho\nabla\cdot\vec{V} \qquad \text{(B 4)}$$

the energy equation can be written as,

$$\frac{Dp}{Dt} + \left[\frac{\frac{p}{\rho^2} - \left(\frac{\partial e}{\partial \rho}\right)_{p}}{\left(\frac{\partial e}{\partial p}\right)_{\rho}}\right]\rho\nabla\cdot\vec{V} = 0 \qquad \text{(B 5)}$$

From the thermodynamic relation:

$$T\,ds = de - \frac{p}{\rho^2}\,d\rho$$

where s is the entropy, we have:

$$\left(\frac{\partial e}{\partial \rho}\right)_s = \frac{p}{\rho^2} \tag{B.6}$$

Using the state relation (B.2), along a line of constant entropy one can write:

$$(de)_s = \left[\left(\frac{\partial e}{\partial \rho}\right)_p d\rho + \left(\frac{\partial e}{\partial p}\right)_\rho dp\right]_s \tag{B.7}$$

or

$$\left(\frac{\partial e}{\partial \rho}\right)_s = \left(\frac{\partial e}{\partial \rho}\right)_p + \left(\frac{\partial e}{\partial p}\right)_\rho \left(\frac{\partial p}{\partial \rho}\right)_s \tag{B.8}$$

By introducing the definition of the speed of sound c,

$$c^2 = \left(\frac{\partial p}{\partial \rho}\right)_s$$

and using Eq (B.6), equation (B.8) becomes

$$\frac{p}{\rho^2} = \left(\frac{\partial e}{\partial \rho}\right)_p - \frac{c^2}{\left(\frac{\partial e}{\partial p}\right)_\rho} \tag{B.9}$$

Substituting this equation into Eq.(B.5) gives:

$$\frac{Dp}{Dt} + \rho c^2 \nabla \cdot \vec{V} = 0 \tag{B.10}$$

For a perfect gas,

$$c^2 = \frac{\gamma p}{\rho}$$

where γ is the specific heat ratio. Therefore, the energy equation can be written as:

$$\frac{Dp}{Dt} + \gamma p \nabla \cdot \vec{V} = 0 \tag{B.11}$$