Adaptive Stable Finite Element Methods for the Compressible Navier-Stokes Equations

by

Philip John Capon

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy

The University of Leeds
School of Computer Studies
December 1995

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others.

Abstract

Many problems involving fluid flow can now be simulated numerically, providing a useful predictive tool for a wide range of engineering applications. Of particular interest in this thesis are computational methods for solving the problem of compressible fluid flow around aerodynamic configurations.

A finite element method is presented for solving the compressible Navier-Stokes equations in two dimensions on unstructured meshes. The method is stabilized by the addition of a least-squares operator (an inexpensive simplification of the Galerkin least-squares method), leading to solutions free of spurious oscillations. Convergence to steady state is reached via a backward Euler time-stepping scheme, and the use of local time-steps allows convergence to be accelerated. The choice of both the nonlinear solver, which is employed to solve the algebraic system arising at each time-step, and the iterative method used within this solver, is fully discussed, along with an inexpensive technique for approximating the Jacobian matrix.

In order to obtain accurate solutions more efficiently, the idea of adapting the mesh is investigated, and two distinct methods of mesh refinement are described in detail. These are the addition of nodes to the mesh in regions determined by an error indicator (h-refinement) and the local repositioning of existing nodes using the value of this error indicator across neighbouring elements (r-refinement). As well as considering these adaptive techniques separately, we introduce an original algorithm which combines the two ideas, with results indicating that this combination is an effective approach. The example problems used consist mainly of steady transonic flow at low to moderate Reynolds numbers.

Transient flow problems are also considered, and we examine the difficulties which occur when the method of lines is used as a solution technique and h-refinement (including derefinement of elements) is carried out.

Acknowledgements

I would like to thank my supervisor Dr. Peter Jimack for his guidance and encouragement throughout the course of this research. I would also like to thank Phil Woods of British Aerospace Sowerby Research Centre, Dr. Shaun Forth and Dr. Martin Berzins for their helpful advice and discussions during this time.

Thanks are also due to my office colleagues Paul Pratt, Gary Stead, David Hodgson and Jeremy Littler for their support and friendship.

This research was supported by EPSRC and British Aerospace plc through a EPSRC CASE award.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many physical processes may be modelled by a set of partial differential equations (p.d.e.'s) and the solution of these equations allows us to make predictions about the behaviour of such processes. In general, the p.d.e.'s cannot be solved analytically, and so a numerical method is required to obtain solutions. The rapid growth in the power and availability of computers in recent years has led to the development of many algorithms for solving these problems successfully, and as a result numerical simulation is beginning to complement or even replace experimental measurement.

One area which has benefited from this development is that of fluid dynamics, allowing many types of fluid flows to be modelled accurately. For example, in the aerospace industry, it may soon be possible to use computational fluid dynamics (c.f.d.) as a cost-effective alternative to building physical models in the design stage of aircraft production.

A common problem in aerodynamics is simulation of compressible flow around a body (such as an aerofoil in two dimensions or a wing in three dimensions). In this thesis we describe a numerical method for solving this type of problem. The compressible Navier-Stokes equations in two dimensions are solved using a stabilized finite element method on unstructured meshes. Solutions are obtained implicitly via Newton's method and a GMRES iterative solver, allowing convergence to steady state in a small number of time-steps. The algorithm is made more efficient by adapting the mesh as solution progresses. The two distinct techniques used to modify the mesh are the addition of extra grid points and the relocation of existing points to those regions where the error is estimated to be large. We focus mainly on steady problems, but also consider transient flow.

In the next section, we present a brief overview of techniques for solving compressible flow problems, mentioning some of the methods in common use and identifying which issues are addressed by this thesis. The contents of each chapter in the thesis are outlined in §1.2, and in §1.3 we discuss some important aspects of the software implementations performed during this work.

1.1 Numerical Simulation of Compressible Flow

The process by which the numerical solution of any system of p.d.e.'s may be obtained can be divided into four distinct stages. First a precise formulation of the problem in terms of equations, boundary conditions and solution domain is needed. Next, the domain is usually divided up into elements or cells to form a discrete grid on which the numerical scheme operates. The third stage is the actual solution procedure, and in the final step, any post-processing, such as visualizing the results, is carried out. We consider each of the four stages separately for the case of compressible flow problems around aerodynamic configurations.

1.1.1 Problem Specification

The Euler equations (defined in [101] for example) are the fundamental equations for describing the motion of an inviscid ideal gas. This hyperbolic system of p.d.e.'s is often used to obtain flow solutions around aerodynamic bodies, based on the assumption that air is both ideal and inviscid. For many flows, this inviscid approach leads to solutions which adequately predict properties such as the lift on the body and the flow field away from the body.

However viscosity does have an effect close to the surface of the body, where frictional forces contribute to the drag, and in some circumstances the boundary layer which is formed can become separated from the surface leading to the creation of a wake. Hence there may be a need to account for viscous effects in the equations which are modelling the flow, and addition of the viscous terms to the Euler equations leads to the Navier-Stokes equations (see [101] or chapter 2 of this thesis).

When a flow is considered to be viscous, the question arises of whether there is turbulence present in the flow. Laminar (i.e. non-turbulent) flow is smooth and

usually occurs when the Reynolds number is below a problem-dependent critical value. Above this value the flow becomes turbulent and is characterized by irregular, apparently chaotic motion of the fluid. For most practical aerodynamic problems, turbulence is present and leads to significantly different behaviour from laminar flow near the body surface, so it is important that the phenomenon is correctly modelled. For most problems, the computational mesh (see §1.1.2) is not fine enough to correctly resolve turbulent effects, so the Navier-Stokes equations are averaged and extra algebraic or differential equations are used to approximate the terms introduced by the presence of turbulence.

Although the effects of the compressibility of air at low speeds (i.e. when the Mach number, M, (the ratio of flow velocity to speed of sound) is less than approximately 0.3) may be ignored, at higher speeds these effects need to be taken into account and so the full equations for compressible flow are used. Compressible flow may be placed in one of several flow regimes, depending on the Mach number, and these range from subsonic (where $M < 1$ everywhere) to hypersonic ($M >> 1$). The properties of the flow in these different regimes vary significantly (see [1] for further details) and will affect the choice of flow solver used.

In many cases there exists a steady state solution to the equations, but it may be necessary for some applications to seek time-dependent solutions, in which case a numerical method is needed which is accurate in time as well as space (and will be more expensive as a consequence).

Having specified the equations which are to be used, the solution domain needs to be defined. This includes the geometry of the aerodynamic configuration which could be as simple as a two-dimensional aerofoil or as complex as a complete three-dimensional aircraft. It is only recently that the level of computing power to deal with three-dimensional flow has been easily available, and previously simulations have been limited to two-dimensions. The solution domain is enclosed by a boundary, which is far enough away from the body that the flow here is that of the freestream. Correct boundary conditions along both the freestream and surface need to be specified so that the p.d.e.'s being solved are well-posed.

In this thesis, the equations which we wish to solve are the Navier-Stokes equations, at low to moderate Reynolds numbers, so that the flow is laminar and, in most cases, steady. Example flows considered lie in either the subsonic, transonic or supersonic flow regimes, with the solution geometry consisting of a domain around

two-dimensional aerofoils.

1.1.2 Grid Generation

Most numerical schemes for solving p.d.e.'s require the solution domain to be partitioned into a large number of cells, where the nodes or cells of the resulting grid are used to form an approximate solution. Except for very simple domains, the generation of such a grid is not straightforward and for complex three dimensional domains can be more expensive than the actual solution of the flow. Weatherill [125] gives a general overview of mesh generation, but here we briefly describe the two classes (structured and unstructured) that grids fall into.

On a structured grid, points are stored in an ordered fashion so that the neighbours of each node may be easily referenced, and each non-boundary node has the same number of neighbours. Usually the shape of the cells are quadrilateral and the grid has a regular appearance (see figure 4.1 for example). Flow solvers based upon structured grids lead to efficient algorithms, because neighbouring node values can be accessed quickly, but the generation of grids is expensive. The usual approach taken is to set up a mapping which transforms the complex physical domain into a simpler computational domain. For more complicated geometries this may not be possible so the domain is divided into blocks, each of which is transformed separately.

Unstructured meshes differ from structured meshes in the data structure used to store the mesh. There is no longer any correspondence between the physical positions of the node points and the order in which they are stored, so that the data structure requires details of the connectivity of the mesh, i.e. knowledge of which nodes surround each element. The appearance of unstructured meshes is usually irregular, and nodes may have a variable number of neighbours. The usual shape of elements is either quadrilateral or more commonly triangular, and an example of a such a mesh is shown in figure 4.2. Two common methods for generating this type of mesh are the advancing front technique, where nodes and connectivity are both defined as the mesh is being generated across the domain, and Delaunay triangulation, where the node positions are determined first, and their connectivity is defined subsequently.

Because the neighbours of each node are no longer directly accessible, use of

these meshes is computationally more expensive than the use of structured grids, but there are a number of advantages when solving on unstructured grids. The major advantage is the ability to generate them for complex domains more quickly than structured meshes. In addition, this approach means that there can be large variations in the mesh density throughout the mesh, allowing regions where the solution is nearly constant to have far fewer nodes than the regions of greatest activity. Although the location of these regions may not be known in advance, it is possible to dynamically modify the mesh according to the flow as the solution progresses.

We do not address the issue of grid generation in this thesis, but use unstructured triangular meshes generated elsewhere. The technique of modifying the mesh during the solution is discussed in some depth, and two methods of doing this are presented.

1.1.3 Flow Solution

In order to obtain a good solution to the p.d.e.'s representing the flow, two important properties that the flow solver should possess are accuracy and stability. An accurate scheme should reproduce the analytical solution as the size of the mesh elements tends to zero, and if it is stable then the numerical solution will be be free of non-physical oscillations. The issue of stability is particularly significant for inviscid flows or flows at high Reynolds numbers, and numerical methods which are successful for other types of p.d.e.'s often need to be modified for these hyperbolic or convection-dominated problems.

In general, the solver transforms the differential operators in the p.d.e.'s into discrete operators. Often the spatial and temporal terms are dealt with separately (semi-discretization), and we now discuss the most popular methods used for the discretization process.

Finite Difference Methods

This class of methods was the first to be used in fluid flow problems, and many variations have since been developed to deal with different types of problem. An introduction to finite differences is given by Smith in [109], but the basic idea is to replace the partial derivatives by expressions written in terms of the unknowns, using Taylor expansions. Only regular structured meshes are suitable for use with

these methods.

In order to deal with first order convection terms, which lead to instabilities when the usual central difference operator is applied to each point, upwinding (where a bias towards the neighbouring upstream node is applied to the operator) has been used. By itself, this is less accurate (as the mesh size tends to zero) than using central differences, which has led to improved methods such as flux limiter schemes, Godunov's scheme [49], involving the solution of a Riemann problem at each cell, and approximate Riemann solvers (e.g. [103]).

Finite Volume Methods

Possibly the most widely used methods in flow simulation, one of the first such schemes being that of Jameson *et al.* [74]. Methods are either cell-centred, using the element as a control volume over which to be integrated or cell-vertex, where the region surrounding each node is the control volume. The equations to be discretized are written in a conservative form, and integrated over each control volume. The surface integrals containing the advective and diffusive fluxes are rewritten as line integrals using the divergence theorem so that the fluxes may be evaluated along the volume faces. This calculation of the numerical fluxes often uses finite difference (or finite element) techniques. Peyret and Taylor [102] give a description of finite volume methods applied to fluid problems.

Historically, finite volume schemes have been used mainly on structured meshes, but many methods have been developed for unstructured triangular meshes in recent years (see [123] for example).

Finite Element Methods

The finite element method is a relative latecomer to c.f.d., its main application having traditionally been in solid mechanics. For a detailed introduction to the method see, for example, [77], [115] or [98]. First, the original p.d.e.'s are multiplied by a test function and integrated over the domain, resulting in the weak formulation of the problem. The infinite dimensional subspace which contains the unknown function is then replaced by a subspace of finite dimension, chosen so that the approximate solution consists of a piecewise polynomial function. Suitable choice of the test function leads to a system of algebraic equations to be solved.

Usually integration is carried out on each element of the mesh (which may well be unstructured) and the global equations are assembled from the local element ones.

In the same way as finite difference and finite volume methods, modifications are needed so that stability is maintained for convection-dominated flows. This has been done using artificial viscosity, which is simple to implement but not very accurate, or streamline upwind Petrov Galerkin methods (e.g. Hughes [65], Johnson [78]). One advantage of using finite elements is that there exists a solid theoretical foundation to the methods, allowing analysis of convergence, stability and error estimation (e.g. [115], [42]).

Use of the above methods, combined with an accurate ordinary differential equation solver (or suitable alternative) if transient solutions are being sought, leads to an explicit scheme, if the algebraic equations depend on the known values at the previous time-step, or an implicit scheme, where the equations depend on the unknown current solution values. In the former case, which is very quick since only updates are being performed at each time-step, convergence to a steady state is slow since the time-steps are limited by stability constraints. Implicit solution requires solving a nonlinear set of equations, which is more expensive, but very large time-steps may be taken. The nonlinear system may be solved using Newton's method, for example, and the resulting linear systems solved by either direct methods or iterative techniques.

Alternatively the technique of multigrid (see [16]) may be used. This makes use of the fact that different components of the error in a solution are most effectively reduced by meshes of differing length-scale. Several meshes, ranging from a very coarse to very fine, are used in the solution process. Since the solution needs to be interpolated between different meshes, the algorithm is more naturally suited to structured grids, and has been used to solve compressible flows around aerofoils (e.g. Jameson and Mavriplis [73]), but an unstructured equivalent has also been developed by Mavriplis [92].

Here, we use finite elements for the spatial discretization, utilizing a variation of the Galerkin least-squares method of Hughes to add stability. A steady state solution is obtained by time-stepping with an implicit backward Euler scheme until the steady solution is reached. The resulting nonlinear system is solved with Newton's method, with preconditioned GMRES as the linear iterative solver.

1.1.4 Post-processing

Once a numerical solution for the flow of interest has been obtained, it is clear that some means of presenting this solution is required. At the simplest level this may be a plot of the pressure coefficients over an aerofoil, but the ability to view the entire flow-field is needed in order to identify the location of features such as shocks and separated flows. The visualization of two-dimensional flows includes the use of contour and shaded plots, and vector arrows. In three dimensions, visualizing the data becomes more difficult and properties such as colour and opacity are used to represent flow variables.

It is important that the numerical method being used is validated by comparing results against both experiment (e.g. wind tunnel) and other numerical codes; usually for some well-defined test problems. As well as simple visual comparisons of solutions, values such as the lift and drag on the body may be used as checks of accuracy.

We are not concerned with such issues as visualization in this thesis, and use a software package "Viz" [112] to display solutions graphically. Results featured here include values such as lift and drag coefficients for comparison with previous results.

The four stages outlined above encompass the complete process for finding numerical solutions to a compressible flow problem. At each stage we have pointed out which issues are addressed in this thesis, and in the following section list the order in which these issues appear.

1.2 Contents of Thesis

We now outline the contents of each chapter in this thesis. In chapter 2, we state the compressible Navier-Stokes equations, and their formulation for different variables. The stabilized finite element method used throughout the remainder of the thesis is then described. This method leads to a system of nonlinear algebraic equations, and chapter 3 contains a discussion of the nonlinear and linear schemes which are used to solve this system.

Some results for a number of well known test cases solved on fixed unstructured

meshes are presented in chapter 4. The next two chapters both address the issue of adaptivity, where the mesh is altered in some way in order to improve the efficiency and/or accuracy of the solution process. In chapter 5 we consider the addition of extra points to the mesh (h-refinement), and give results for comparison with those obtained in chapter 4. Chapter 6 contains a description of a node movement algorithm (r-refinement) and we present results of using r-refinement alone and a combination of the two techniques (hr-refinement).

Prior to chapter 7, only steady flow problems are considered, but in this chapter, we discuss some methods, and associated difficulties, for accurately solving transient flows. In chapter 8, three separate issues are discussed as areas for future work. These include turbulence modelling, implementation of the node movement algorithm of chapter 6 for unsteady problems and extension of the work to three dimensions. A brief summary is given in chapter 9.

1.3 Implementation Details

This section contains some comments concerning the practical implementation of the methods presented in the remainder of the thesis. The code has been written entirely in Fortran-77, rather than a more recent computer language such as C or Fortran-90. Most established numerical software packages have been written in Fortran-77, including the ones used in later chapters (NKSOL, SLAP and SPRINT), so using (and modifying) these packages in conjunction with our code is straightforward.

However, Fortran-77 does not contain features found in more modern languages, including dynamic memory allocation. Because problems are solved on unstructured meshes and the number of unknowns vary (due to mesh refinement), the lack of this property means that the storage space is not always used efficiently. In addition, it causes some of the algorithms to be more complicated than the equivalent C code (for example, construction and storage of the sparse matrix structure used to store the Jacobian matrix in §3.3.6).

All the code has been written to be run on a serial machine, and no consideration has been given the implementation of these methods on parallel machines. The timings for results presented in later chapters have been obtained on a single MIPS R4400 processor of a Silicon Graphics Challenge/XL eight-processor machine.

Chapter 2

A Stable Finite Element Method for the Steady Navier-Stokes Equations

2.1 Introduction

The purpose of this chapter is to present a finite element method for the solution of the compressible Navier-Stokes equations in two dimensions. The type of flow which we consider in this chapter is both steady and laminar: time-dependent and turbulent solutions are discussed in subsequent chapters. We begin by stating the equations in §2.2, a system of four (in two dimensions) p.d.e.'s, and since the major application of this work is in aerodynamics, suitable boundary conditions for external flows are defined. Alternative formulations of the equations are also given, which use different physical properties as the dependent variables.

The Galerkin finite element method is discussed in §2.3, along with details of how the method is applied to the Navier-Stokes equations. The problems arising from using the Galerkin method on its own are also considered here. In §2.4 and §2.5, some improvements to the standard Galerkin method are presented, including a modified version of the Galerkin least-squares method of Hughes [66], which is used in later chapters.

Although we are concerned with steady solutions of the equations, solving the steady equations directly can be computationally very expensive, so it is usually necessary to obtain such solutions via some form of time-stepping where the time

derivative is included in the equations. This is discussed in §2.6 where the technique of local time-stepping is also described.

2.2 The Navier-Stokes Equations for Compressible Flow

In this section, we state the compressible Navier-Stokes equations and describe a number of alternative variable formulations. The form of the equations using primitive variables is discussed first, with associated boundary conditions. Other formulations are presented, including a generalized formulation for an arbitrary set of variables.

The five primary variables involved in the study of the compressible flow of a gas are velocity ($\mathbf{u}$), density (ρ), pressure (p), internal energy (e) and temperature (T). In two dimensions, $\mathbf{u}$ consists of two components (u and v) so six governing equations are required. We make the following assumptions about the gas: it is ideal, so that

$$p = \rho RT, \tag{2.1}$$

where R is the specific gas constant; it is calorically perfect, which means that the specific heats at constant volume and pressure c_v and c_p are constant, so that

$$e = c_v T. \tag{2.2}$$

These assumptions are reasonable in the case of air provided the temperature is moderate and the flow is not hypersonic. The relations also mean that only a further four equations are required, and these are derived from the principles of conservation of mass, momentum and energy:

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho \mathbf{u}) = 0 \tag{2.3}$$

$$\frac{\partial \rho u}{\partial t} + \nabla.(u\rho \mathbf{u}) + \frac{\partial p}{\partial x} = \frac{\partial \tau_{11}}{\partial x} + \frac{\partial \tau_{21}}{\partial y} \tag{2.4}$$

$$\frac{\partial \rho v}{\partial t} + \nabla.(v\rho \mathbf{u}) + \frac{\partial p}{\partial y} = \frac{\partial \tau_{12}}{\partial x} + \frac{\partial \tau_{22}}{\partial y} \tag{2.5}$$

$$\begin{aligned} &\tfrac{\partial}{\partial t}(e + |\mathbf{u}|^2/2) + \nabla.((e + |\mathbf{u}|^2/2)\rho \mathbf{u}) + \nabla.(p\mathbf{u}) = \\ &\nabla.(k\nabla T) + \tfrac{\partial}{\partial x}(\tau_{11}u + \tau_{12}v) + \tfrac{\partial}{\partial y}(\tau_{21}u + \tau_{22}v) \quad , \end{aligned} \tag{2.6}$$

where κ is the coefficient of thermal conductivity and τ_{ij} is the viscous-stress tensor given by

$$\begin{bmatrix} \lambda(u_x + v_y) + 2\mu u_x & \mu(u_y + v_x) \\ \mu(v_x + u_y) & \lambda(u_x + v_y) + 2\mu v_y \end{bmatrix}. \tag{2.7}$$

For the viscosity coefficients λ and μ, we assume Stoke's hypothesis ($\lambda = -2/3\mu$). For further details of how the above equations are obtained, see for example Batchelor [12], O'Neill and Chorlton [101] or Anderson [1].

The four p.d.e.'s (2.3)–(2.6) form the Navier-Stokes equations, which can be rewritten in a number of ways, depending on the choice of unknown variables. We first consider a primitive variable formulation.

2.2.1 Primitive formulation

Following the approach of Bristeau *et al.* [17], the equations can be non-dimensionalized and written so that the primitive variables of ρ (density), $\mathbf{u}$ (velocity) and T (temperature) are the primary unknowns:

$$\frac{\partial \rho}{\partial t} + \mathbf{u}.\nabla\rho + \rho\nabla.\mathbf{u} = 0, \tag{2.8}$$

$$\rho\frac{\partial \mathbf{u}}{\partial t} + \rho(\mathbf{u}.\nabla)\mathbf{u} + (\gamma - 1)(T\nabla\rho + \rho\nabla T) = \frac{1}{Re}[\Delta\mathbf{u} + \frac{1}{3}\nabla(\nabla.\mathbf{u})], \tag{2.9}$$

$$\rho\frac{\partial T}{\partial t} + \rho\mathbf{u}.\nabla T + (\gamma - 1)\rho T\nabla.\mathbf{u} = \frac{1}{Re}(\frac{\gamma}{Pr}\Delta T + F(\nabla\mathbf{u})). \tag{2.10}$$

Note that ρ, $\mathbf{u}$ and T refer to non-dimensionalized quantities here, rather than the physical variables denoted in (2.3)–(2.6). The non-dimensionalization introduces two dimensionless parameters Re and Pr. The Reynolds number

$$Re = \frac{\rho_\infty u_\infty l_\infty}{\mu}, \tag{2.11}$$

where ρ_∞, u_∞ and l_∞ are the freestream density, freestream speed and a suitable length scale respectively, represents physically the ratio of inertia forces to viscous forces, while the Prandtl number

$$Pr = \frac{\mu c_p}{\kappa} \tag{2.12}$$

measures the ratio of energy dissipated by friction to the energy transported by thermal conduction. The ratio of specific heats (c_p/c_v) is denoted by γ, and throughout this work we assume $Pr = 0.72$ and $\gamma = 1.4$. The Reynolds number (with the

Mach number defined below) governs the type of flow—the amount of viscosity in the flow decreases with increasing Re and the flow becomes inviscid in the limit as Re tends to infinity.

For two dimensional flows, $F(\nabla \mathbf{u})$ has the form

$$F(\nabla \mathbf{u}) = \frac{4}{3}\left[(\frac{\partial u}{\partial x})^2 + (\frac{\partial v}{\partial y})^2 - \frac{\partial u}{\partial x}\frac{\partial v}{\partial y}\right] + \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)^2. \tag{2.13}$$

The equations are to be solved in a region $\Omega \subset \mathbf{R}^2$, with a boundary Γ. In the case of an external flow around a solid body such as an aerofoil, we define Γ_∞ to be the farfield boundary, which is split up into the inflow boundary,

$$\Gamma_\infty^- = \{\mathbf{x} : \mathbf{x} \in \Gamma_\infty, \mathbf{u}_\infty.\mathbf{n} < 0\}, \tag{2.14}$$

and outflow boundary

$$\Gamma_\infty^+ = \Gamma_\infty \backslash \Gamma_\infty^-, \tag{2.15}$$

where $\mathbf{n}$ is the unit vector of the outward normal to the farfield boundary. The internal wall boundary is denoted by Γ_B, as shown in figure 2.1.

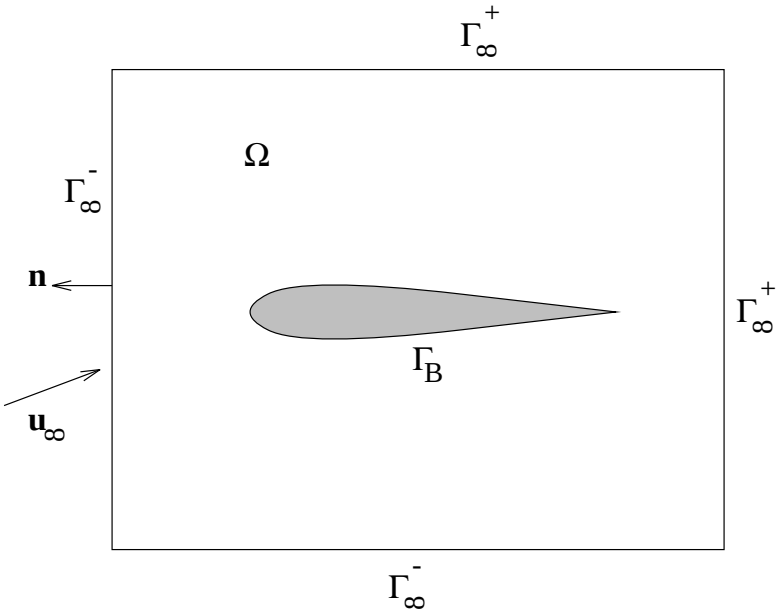

Figure 2.1: Typical domain and boundaries for external flow around a body

For a given angle of attack α,

$$\mathbf{u}_\infty = \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix} \tag{2.16}$$

is the normalized freestream velocity. As boundary conditions, on the inflow boundary Γ_∞^-, we have

$$\mathbf{u} = \mathbf{u}_\infty \tag{2.17}$$

$$\rho = 1 \tag{2.18}$$

$$T = T_\infty = \frac{1}{\gamma(\gamma-1)M_\infty^2} \tag{2.19}$$

where the parameter M_∞ is the freestream Mach number, the ratio of flow speed to the speed of sound. On the outflow boundary, Γ_∞^+,

$$\frac{\partial \mathbf{u}}{\partial n} = 0 \tag{2.20}$$

$$\frac{\partial T}{\partial n} = 0. \tag{2.21}$$

If $M_\infty < 1$ then we also specify $\rho = 1$. On the internal boundary, Γ_B, the no-slip condition of

$$\mathbf{u} = 0 \tag{2.22}$$

and

$$T = T_B = T_\infty[1 + (\gamma-1)M_\infty^2/2] \tag{2.23}$$

are enforced. We specify these boundary conditions the same way as Bristeau *et al.* [17].

Here we are interested in steady solutions of the Navier-Stokes equations, and in §2.3 we consider the time-independent set of equations. However in practice steady solutions are often obtained using time-stepping (see §2.6) and in this case initial conditions are also required:

$$\rho(\mathbf{x}, 0) = \rho_0(\mathbf{x}) \tag{2.24}$$

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x}) \tag{2.25}$$

$$T(\mathbf{x}, 0) = T_0(\mathbf{x}). \tag{2.26}$$

2.2.2 Alternative formulations

We demonstrate in this section that the Navier-Stokes equations can be written in a general form for any appropriate set of variables as the unknowns in the equations. If the conservative variables of density, momentum and energy are used then the Navier-Stokes equations may be expressed as

$$\mathbf{U}_{,t} + \mathbf{F}_{i,i}^{\text{adv}} = \mathbf{F}_{i,i}^{\text{diff}} + \mathcal{F} \tag{2.27}$$

where "$_{,i}$" denotes partial differentiation by x_i and the summation convention is being used. $\mathbf{U}$, $\mathbf{F}_i^{\text{adv}}$ and $\mathbf{F}_i^{\text{diff}}$ are defined as follows

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho(e + |\mathbf{u}|^2/2) \end{pmatrix} \tag{2.28}$$

$$\mathbf{F}_i^{\text{adv}} = \rho u_i \begin{pmatrix} 1 \\ u_1 \\ u_2 \\ (e + |\mathbf{u}|^2/2) \end{pmatrix} + \begin{pmatrix} 0 \\ p\delta_{1i} \\ p\delta_{2i} \\ pu_i \end{pmatrix} \quad , \quad \mathbf{F}_i^{\text{diff}} = \begin{pmatrix} 0 \\ \tau_{1i} \\ \tau_{2i} \\ \tau_{ij}u_j + (\kappa T)_{,i} \end{pmatrix}, \tag{2.29}$$

$\mathcal{F}$ is the source vector containing body forces present and heat supplied. Note that the Euler equations are obtained from (2.27) simply by removing the terms $\mathbf{F}_{i,i}^{\text{diff}}$ and $\mathcal{F}$.

It is possible to rewrite (2.27) in quasi-linear form as

$$\mathbf{U}_{,t} + A_i \mathbf{U}_{,i} = (K_{ij}\mathbf{U}_{,j})_{,i} + \mathcal{F} \tag{2.30}$$

with $A_i = F_{i,\mathbf{U}}$ (i.e. the Jacobian matrix $\partial \mathbf{F}_i^{\text{adv}}/\partial \mathbf{U}$) and the K_{ij}'s satisfying $K_{ij}\mathbf{U}_{,j} = \mathbf{F}_i^{\text{diff}}$. These matrices have the following form:

$$A_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ (\gamma-1)|\mathbf{u}|^2/2 - u^2 & (3-\gamma)u & (1-\gamma)v & (\gamma-1) \\ -uv & v & u & 0 \\ -u(\gamma e + |\mathbf{u}|^2(\frac{\gamma}{2}-1)) & \gamma e - (\gamma-1)u^2 + \frac{|\mathbf{u}|^2}{2} & (1-\gamma)uv & u\gamma \end{pmatrix} \tag{2.31}$$

$$A_2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ -uv & v & u & 0 \\ (\gamma-1)|\mathbf{u}|^2/2 - v^2 & (1-\gamma)u & (3-\gamma)v & (\gamma-1) \\ -v(\gamma e + |\mathbf{u}|^2(\frac{\gamma}{2}-1)) & (1-\gamma)uv & \gamma e - (\gamma-1)v^2 + \frac{|\mathbf{u}|^2}{2} & v\gamma \end{pmatrix} \tag{2.32}$$

$$K_{11} = \frac{\mu}{\rho} \begin{pmatrix} 0 & 0 & 0 & 0 \\ -\frac{4}{3}u & \frac{4}{3} & 0 & 0 \\ -v & 0 & 1 & 0 \\ -(\frac{4}{3}u^2 + v^2 + \frac{\gamma}{Pr}(e - \frac{|\mathbf{u}|^2}{2})) & u(\frac{4}{3} - \frac{\gamma}{Pr}) & v(1 - \frac{\gamma}{Pr}) & \frac{\gamma}{Pr} \end{pmatrix} \tag{2.33}$$

$$K_{12} = \frac{\mu}{\rho}\begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{2}{3}v & 0 & -\frac{2}{3} & 0 \\ -u & 1 & 0 & 0 \\ -\frac{uv}{3} & v & -\frac{2}{3}u & 0 \end{pmatrix}, K_{21} = \frac{\mu}{\rho}\begin{pmatrix} 0 & 0 & 0 & 0 \\ -v & 0 & 1 & 0 \\ \frac{2}{3}u & -\frac{2}{3} & 0 & 0 \\ -\frac{uv}{3} & -\frac{2}{3}v & u & 0 \end{pmatrix} \tag{2.34}$$

$$K_{22} = \frac{\mu}{\rho}\begin{pmatrix} 0 & 0 & 0 & 0 \\ -u & 1 & 0 & 0 \\ -\frac{4}{3}v & 0 & \frac{4}{3} & 0 \\ -(\frac{4}{3}v^2 + u^2 + \frac{\gamma}{Pr}(e - \frac{|\mathbf{u}|^2}{2})) & u(1-\frac{\gamma}{Pr}) & v(\frac{4}{3} - \frac{\gamma}{Pr}) & \frac{\gamma}{Pr} \end{pmatrix} \tag{2.35}$$

More generally, Hauke and Hughes [61] note that any independent set of variables $\mathbf{Y}$ can be used to obtain the equations in quasi-linear form:

$$A_0\mathbf{Y}_{,t} + A_i\mathbf{Y}_{,i} = (K_{ij}\mathbf{Y}_{,j})_{,i} + \mathcal{F}, \tag{2.36}$$

where $A_0 = \mathbf{U}_{,\mathbf{Y}}$, $A_i = F^{\text{adv}}_{i,\mathbf{Y}}$ and $K_{ij}\mathbf{Y}_{,j} = \mathbf{F}^{\text{diff}}_i$.

So the primitive variables of velocity, density and temperature, used in §2.2.1, may be used, when the matrices are expressed as follows:

$$A_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \rho & 0 & 0 \\ 0 & 0 & \rho & 0 \\ 0 & 0 & 0 & \rho \end{pmatrix} \tag{2.37}$$

$$A_1 = \begin{pmatrix} u & \rho & 0 & 0 \\ (\gamma-1)T & \rho u & 0 & (\gamma-1)\rho \\ 0 & 0 & \rho u & 0 \\ 0 & \{\rho(\gamma-1)T - 2\frac{2u_x - v_y}{3Re}\} & \{-\frac{v_x+u_y}{Re}\} & \rho u \end{pmatrix} \tag{2.38}$$

$$A_2 = \begin{pmatrix} v & 0 & \rho & 0 \\ 0 & \rho v & 0 & 0 \\ (\gamma-1)T & 0 & \rho v & \rho(\gamma-1) \\ 0 & \{-\frac{v_x+u_y}{Re}\} & \{\rho(\gamma-1)T - 2\frac{2v_y - u_x}{3Re}\} & \rho v \end{pmatrix} \tag{2.39}$$

$$K_{11} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{4}{3Re} & 0 & 0 \\ 0 & 0 & \frac{1}{Re} & 0 \\ 0 & 0 & 0 & \frac{\gamma}{RePr} \end{pmatrix}, K_{12} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3Re} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.40}$$

$$K_{21} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3Re} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, K_{22} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{1}{Re} & 0 & 0 \\ 0 & 0 & \frac{4}{3Re} & 0 \\ 0 & 0 & 0 & \frac{\gamma}{RePr} \end{pmatrix} \tag{2.41}$$

$$\mathcal{F} = 0 \tag{2.42}$$

The above set of matrices are nonsymmetric, as are those defined for the conservative variables. However there does exist a set of variables for which the advective A_i and diffusive K_{ij} matrices are symmetric. This is called the set of entropy variables, and such variables have been investigated by Harten [59], Hughes *et al.* [67] and Johnson *et al.* [81]. Shakib and Hughes give the derivation and form of the advective and diffusive matrices in [108]. This symmetric property of entropy variables is useful for mathematical analysis and is exploited by Shakib and Hughes [108] when developing the Galerkin/least-squares finite element method (see §2.5). The disadvantage of using entropy variables is their complexity compared to other variables, especially in dealing with boundary conditions, since these are specified in terms of physical variables and the equations relating the two sets of variables are highly nonlinear.

Another formulation which may be used consists of the primitive variables of velocity, pressure and temperature. In this form the equations are well-posed in the incompressible limit and so this offers the possibility of solving both compressible and incompressible flow problems within the same code. Both this and the previous primitive form mentioned have the advantage that the advective and diffusive matrices are relatively sparse, hence computationally more efficient than using conservative or entropy variables (this can be seen by comparing equations (2.31)–(2.35) with (2.38)–(2.41)).

The finite element methods described in §2.3 and §2.5 are to be applied to the general quasi-linear form (2.36), thus allowing any of the above mentioned formulations to be used, provided the advective and diffusive matrices have been defined and correct boundary conditions implemented. We will use mainly the primitive variables of velocity, density and temperature, as well as the conservative variables (2.28) for comparison.

Since we are interested in steady flow, we ignore the time dependent term $A_0\mathbf{Y}_{,t}$ in (2.36) in the next three sections and only consider spatial discretizations of the

steady equations. The use of time-stepping in the transient equations to reach steady state is discussed in §2.6.

2.3 The Galerkin Finite Element Method

In this section a finite element method for solving the time-independent version of the system (2.36) of p.d.e.'s is described. For the particular example of primitive variables, for which boundary conditions were stated in §2.2.1, precise details of how the method has been implemented are given. The problems with using the standard Galerkin method for convection-dominated flows are discussed in §2.3.1.

We wish to solve the nonlinear system of p.d.e.'s

$$A_i \mathbf{U}_{,i} = (K_{ij}\mathbf{U}_{,j})_{,i} + \mathcal{F}, \quad i = 1,2 \tag{2.43}$$

over the domain Ω (figure 2.1 for example) with suitable boundary conditions defined on the boundary Γ. The vector $\mathbf{U}$ consists of four unknowns, and the 4×4 matrices A_i and K_{ij} are dependent upon the choice of physical variables in $\mathbf{U}$. We first define the function spaces $\mathcal{U}$ and $\mathcal{V}$:

$$\mathcal{U} = \{\mathbf{U} : u_i \in H^1(\Omega), i = 1,..,4, \mathbf{U} \in \mathcal{S}_\Gamma\}. \tag{2.44}$$

The boundary conditions $\mathcal{S}_\Gamma$ will depend upon the choice of variables, but in the case of the primitive formulation (§2.2.1), where $\mathbf{U} = (\rho, u, v, T)$,

$$\mathcal{S}_\Gamma = \{\mathbf{U} : \mathbf{U} = (1, \cos\alpha, \sin\alpha, T_\infty)^T \text{on } \Gamma_\infty^-, u_2, u_3 = 0, u_4 = T_B \text{ on } \Gamma_B\} \tag{2.45}$$

($u_1 = 1$ on Γ_∞^+ is also required when $M_\infty < 1$). We define $\mathcal{V}$ as

$$\mathcal{V} = \{\mathbf{V} : v_i \in H^1(\Omega), i = 1,..,4, \mathbf{V} \in \mathcal{T}_\Gamma\} \tag{2.46}$$

where (in the case of primitive variables)

$$\mathcal{T}_\Gamma = \{\mathbf{V} : \mathbf{V} = \mathbf{0} \text{ on } \Gamma_\infty^-, v_2, v_3, v_4 = 0 \text{ on } \Gamma_B\} \tag{2.47}$$

(also $v_1 = 0$ on Γ_∞^+ if $M_\infty < 1$). A weak formulation for (2.43) can now be defined by multiplying by a test function $\mathbf{V} \in \mathcal{V}$ and integrating over Ω: find $\mathbf{U} \in \mathcal{U}$, such that for all $\mathbf{V} \in \mathcal{V}$, the following is satisfied

$$\int_\Omega (A_i \mathbf{U}_{,i}.\mathbf{V} + \mathbf{V}_{,i}.K_{ij}\mathbf{U}_{,j} - \mathcal{F}.\mathbf{V})d\Omega - \int_\Gamma \mathbf{V}.(K_{ij}\mathbf{U}_{,j})n_i d\Gamma = 0. \tag{2.48}$$

Some discrete finite element subspaces can now be defined. The domain Ω is partitioned into n_{el} non-overlapping triangles Ω_e, so that $\Omega = \cup_e \Omega_e$, and each triangle has an associated mesh size parameter h. The solution $\mathbf{U}$ is approximated by $\mathbf{U}^h$, a vector of functions which are linear over each element and continuous along element edges. A basis for the space of each of these functions is the set of piecewise linear "hat" functions ϕ_i, $i = 1, .., n_{nod}$, where n_{nod} is the total number of nodes. These have the value $\phi_i = \delta_{ij}$ at node j.

Hence the discrete trial space may be defined as

$$\mathcal{U}^h = \{\mathbf{U}^h : \mathbf{U}^h \in (C^0(\Omega))^4, \mathbf{U}^h|_{\Omega^e} \in (P_1)^4, \mathbf{U}^h \in \mathcal{S}_\Gamma\}, \tag{2.49}$$

where P_1 is the space of linear polynomials, and the corresponding discrete test space is

$$\mathcal{V}^h = \{\mathbf{V}^h : \mathbf{V}^h \in (C^0(\Omega))^4, \mathbf{V}^h|_{\Omega^e} \in (P_1)^4, \mathbf{V}^h \in \mathcal{T}_\Gamma\}. \tag{2.50}$$

An equivalent discrete problem to (2.48) can now be stated: find $\mathbf{U}^h \in \mathcal{U}^h$, such that for all $\mathbf{V}^h \in \mathcal{V}^h$, the following is satisfied

$$\int_\Omega A_i \mathbf{U}^h_{,i}.\mathbf{V}^h + \mathbf{V}^h_{,i}.K_{ij}\mathbf{U}^h_{,j} - \mathcal{F}.\mathbf{V}^h)d\Omega - \int_\Gamma \mathbf{V}^h.(K_{ij}\mathbf{U}^h_{,j})n_i d\Gamma = 0. \tag{2.51}$$

Thus the problem is reduced to one of finite dimension, and requires a nonlinear algebraic equation solver such as Newton's method (see chapter 3). We now consider the specific example of using primitive variables, and give the equations that arise from this choice.

If the total number of nodes in the mesh is n_{nod}, then

$$n_{nod} = n_{int} + n_{out} + n_{wall} + n_{in} \tag{2.52}$$

where n_{int}, n_{wall}, n_{in}, and n_{out} are the number of nodes away from the boundaries, on the wall boundary (Γ_B), on the inflow boundary (Γ_∞^-) and on the outflow boundary (Γ_∞^+) respectively. These are ordered as they appear in (2.52) so that nodes $1 \ldots n_{int}$ denote the internal nodes, nodes $n_{int}+1 \ldots n_{out}$ denote the outflow nodes etc. Now

define the piecewise linear approximation $\mathbf{U}^h$ of $\mathbf{U} = (\rho, u, v, T)^T$ as

$$\mathbf{U}^h = \begin{pmatrix} \sum_{i=1}^{n_{int}} \rho_i \phi_i + \sum_{i=n_{int}+1}^{n_{int}+n_{out}} \phi_i + \sum_{i=n_{int}+n_{out}+1}^{n_{nod}-n_{in}} \rho_{i-n_{out}} \phi_i + \sum_{i=n_{nod}-n_{in}+1}^{n_{nod}} \phi_i \\ \sum_{i=1}^{n_{int}+n_{out}} u_i \phi_i + \sum_{i=n_{nod}-n_{in}+1}^{n_{nod}} \cos\alpha \phi_i \\ \sum_{i=1}^{n_{int}+n_{out}} v_i \phi_i + \sum_{i=n_{nod}-n_{in}+1}^{n_{nod}} \sin\alpha \phi_i \\ \sum_{i=1}^{n_{int}+n_{out}} T_i \phi_i + \sum_{i=n_{int}+n_{out}+1}^{n_{nod}-n_{in}} T_B \phi_i + \sum_{i=n_{nod}-n_{in}+1}^{n_{nod}} T_\infty \phi_i \end{pmatrix} \tag{2.53}$$

where ρ_i, u_i, v_i and T_i are the unknown coefficients to be determined (note that we have assumed the case $M_\infty < 1$, so $\rho = 1$ on Γ_∞^+). Thus the total number of unknowns m is

$$\begin{aligned} m &= n_{int} + n_{wall} + n_{int} + n_{out} + n_{int} + n_{out} + n_{int} + n_{out} && (2.54) \\ &= 4n_{int} + 3n_{out} + n_{wall}, && (2.55) \end{aligned}$$

and we require m equations, obtained by setting $\mathbf{V}^h$ to the following vectors in turn:

$$\begin{gathered} (\phi_i, 0, 0, 0)^T,\ i = 1, \ldots, n_{int},\ n_{int} + n_{out} + 1, \ldots, n_{int} + n_{out} + n_{wall} \\ (0, \phi_i, 0, 0)^T,\ i = 1 \ldots n_{int} + n_{out} \\ (0, 0, \phi_i, 0)^T,\ i = 1 \ldots n_{int} + n_{out} \\ (0, 0, 0, \phi_i)^T,\ i = 1 \ldots n_{int} + n_{out} \end{gathered}$$

in the variational form (2.51). This leads to the following equations being obtained (where we define $\mathbf{U}^h = (\rho^h, u^h, v^h, T^h)^T$ and $\mathbf{u}^h = (u^h, v^h)$ for convenience):

$$\int_\Omega (\mathbf{u}^h.\nabla \rho^h + \rho^h \nabla.\mathbf{u}^h)\phi_i d\Omega = 0,$$

$$i = 1, .., n_{int},\ n_{int} + n_{out} + 1, ..n_{int} + n_{out} + n_{wall} \tag{2.56}$$

$$\int_\Omega [(\rho^h(\mathbf{u}^h.\nabla)u^h + (\gamma - 1)(T^h \frac{\partial \rho^h}{\partial x} + \rho \frac{\partial T^h}{\partial x}))\phi_i + \frac{1}{Re}(\nabla u^h.\nabla \phi_i + \frac{1}{3}(\nabla.\mathbf{u}^h)\frac{\partial \phi_i}{\partial x})]d\Omega$$

$$- \frac{1}{3Re}\int_{\Gamma_\infty^+} (\nabla.\mathbf{u}^h)\phi_i n_1 d\Gamma = 0, \quad i = 1, .., n_{int} + n_{out} \tag{2.57}$$

$$\int_\Omega [(\rho^h(\mathbf{u}^h.\nabla)v^h + (\gamma - 1)(T^h \frac{\partial \rho^h}{\partial y} + \rho \frac{\partial T^h}{\partial y}))\phi_i + \frac{1}{Re}(\nabla v^h.\nabla \phi_i + \frac{1}{3}(\nabla.\mathbf{u}^h)\frac{\partial \phi_i}{\partial y})]d\Omega$$

$$- \frac{1}{3Re}\int_{\Gamma_\infty^+} (\nabla.\mathbf{u}^h)\phi_i n_2 d\Gamma = 0, \quad i = 1, .., n_{int} + n_{out} \tag{2.58}$$

$$\int_\Omega [(\rho^h(\mathbf{u}^h.\nabla T^h) + (\gamma - 1)\rho^h t^h(\nabla.\mathbf{u}^h))\phi_i + \frac{1}{Re}(\frac{\gamma}{Pr}\nabla T^h.\nabla \phi_i$$

$$- F(\nabla \mathbf{u}^h)\phi_i)]d\Omega = 0, \quad i = 1, .., n_{int} + n_{out}. \tag{2.59}$$

These equations may be written in the form

$$\mathbf{G}(\mathbf{W}) = \mathbf{0} \tag{2.60}$$

where $\mathbf{W}$ is the vector of all the unknown coefficients ρ_i, u_i, v_i and T_i, and $\mathbf{G}$ is the nonlinear function representing (2.56)–(2.59). Chapter 3 describes how such a system may be solved.

Any method used to solve (2.60) will require evaluation of $\mathbf{G}$ given $\mathbf{W}$. This is calculated element-wise, and assembled to form the global vector. Details of such an assembly process are given in [77], for single equations, and for systems the procedure is similar. The major complication is that nodes have a variable number of unknowns associated with them, i.e. most have four, but nodes on the wall boundary have one, and nodes on the outflow boundary may have three. This requires an extra data structure to keep track of the node with which each unknown is associated.

The numerical integration which is required over each element is carried out using a three point Gaussian quadrature rule from the list of rules given by Cowper [32]. Results indicate that this appears to be sufficient for the problems considered here.

2.3.1 Inadequacy of the Galerkin method

The method outlined above, in conjunction with a suitable nonlinear solver, provides a means of finding solutions to the Navier-Stokes equations. However this approach does not typically work very well in practice, even for problems where the Reynolds number is small ($Re = O(1)$). Results, given in §4.6 and also by Bristeau *et al.* [17], contain spurious oscillations in the values of the density, and other variables at higher Reynolds numbers (e.g. $Re = 73$).

The reason for these non-physical oscillations appears to be the presence of the convection terms in the Navier-Stokes equations. This is the dominant term in the continuity equation, and the other equations at higher Reynolds numbers. When the standard Galerkin method (or equivalently second order differencing) is applied to the convection terms, the resulting set of equations may be decoupled between

adjacent nodes, leading to node-by-node oscillations. This effect is studied in detail for a simple one dimensional problem in [52].

Another consequence of using the standard Galerkin method on its own for convection dominated flows is that convergence of the linear solver (GMRES for example) is very slow, even with preconditioners such as ILU or Jacobi (§3.3.3). This is due to the effect of the convection terms on the Jacobian matrix which causes it to be non-diagonally dominant, i.e. the Jacobian has very small values in the diagonal compared to other values along a row. Hence preconditioners in which the matrix diagonal plays an important role fail to improve convergence.

In the following sections, some modifications of the Galerkin method which overcome these difficulties are discussed.

2.4 Use of bubble functions

For low Reynolds numbers, it is possible to use the Galerkin method successfully, provided a suitable higher order approximation for the velocity fields is chosen. Such an approach has been used for incompressible flow, and in this section we discuss the use of bubble functions, used by Bristeau *et al.* [17], and how they can be used to obtain a stable Galerkin scheme for compressible flow.

Numerical simulations of the *incompressible* Navier-Stokes equations suffer from undesirable oscillations arising from two sources. One, outlined in §2.3.1, is due to the presence of a convection term in the equations. The other is due to the incompressibility constraint of the equations ($\nabla.\mathbf{u} = 0$), and requires that only certain combinations of approximations should be used for the velocity and pressure fields. Permissible combinations are those that satisfy the Babuska-Brezzi condition [4], such as quadratic approximation of velocity and linear approximation of pressure, whereas linear approximations for both variables fail to meet the condition. Further details concerning the use of finite elements in incompressible flow are given by Gunzburger [53].

The Babuska-Brezzi condition is also satisfied by using certain bubble functions. For example, the space of cubic bubbles for a triangle K is defined as

$$B_K = \{\phi_B : \phi_B|_K \in P_3, \phi_B = 0 \text{ on } \partial K\} \tag{2.61}$$

where P_3 is the set of cubic polynomials. The spaces used to approximate the

velocity and pressure fields are defined as

$$\mathcal{V}^h = \{\mathbf{v} : \mathbf{v} \in (C^0(\Omega))^2, \mathbf{v}|_K \in (P_1 \oplus B_K)^2\} \tag{2.62}$$

$$\mathcal{P}^h = \{p : p \in (C^0(\Omega)), p|_K \in P_1\}. \tag{2.63}$$

In [17], Bristeau *et al.* use this approach for the primitive formulation of the compressible Navier-Stokes equations. By choosing (2.62) as the approximation space for velocity and (2.63) for the other variables of density and temperature, the spurious oscillations which occur using piecewise linear basis functions everywhere are eliminated.

In this work, we have attempted to use bubble functions in the same way. One benefit of using bubble functions rather than other higher order polynomials is that static condensation, in which the extra unknowns introduced by the bubble functions can be eliminated from the linear system, can be used.

Static condensation uses the fact that the bubble functions are zero on each triangle boundary, so have no effect on neighbouring triangles. This means that the extra unknowns introduced by using this extra function can be eliminated from the linear system and recovered afterwards. To see this consider a simple example, which, using piecewise linear basis functions, leads to a linear, symmetric system:

$$A\mathbf{x} = \mathbf{b} \tag{2.64}$$

where A is a matrix (the stiffness matrix for example), $\mathbf{x}$ consists of the unknowns at each node and $\mathbf{b}$ is a vector. If bubble functions are now incorporated into the approximation, the system becomes

$$\begin{bmatrix} A & a^\phi \\ (a^\phi)^T & \Phi \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{x}^\phi \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{b}^\phi \end{bmatrix} \tag{2.65}$$

where $\mathbf{x}^\phi$ are the extra unknowns situated at the centroid of each triangle, $\mathbf{b}^\phi$ is another vector, a^ϕ is a matrix containing only three nonzero elements per row (since each bubble function will affect the value of the three vertices of that triangle only), and Φ is a diagonal matrix . Since Φ is diagonal, it is straightforward to calculate $\mathbf{x}^\phi$ from

$$\mathbf{x}^\phi = \mathbf{b}^\phi - \Phi^{-1}(a^\phi)^T\mathbf{x} \tag{2.66}$$

once $\mathbf{x}$ has been found by solving the system

$$(A - a^\phi\Phi^{-1}(a^\phi)^T)\mathbf{x} = \mathbf{b} - a^\phi\mathbf{b}^\phi. \tag{2.67}$$

This idea can be extended to the more complicated Navier-Stokes system, and means that the linear (or nonlinear) system being solved is no larger than if only piecewise linear approximations were being used.

In numerical experiments, results indicate that the use of bubble functions does indeed remove non-physical oscillations at fairly low Reynolds numbers ($Re = 73$), as well as improving convergence, but as the Reynolds number is increased ($Re = 500$) oscillations begin to re-appear. See §4.6 for details.

The reason why the Galerkin method with bubble functions leads to stable solutions at fairly low Reynolds numbers is discussed in detail by Brezzi *et al.* in [15]. For the linearized compressible Navier-Stokes equations, they show that this technique is equivalent, in the diffusive limit, to streamline-upwind/Petrov-Galerkin methods (see next section). This equivalence is established in a more abstract framework by Baiocchi *et al.* [8].

In addition to the use of bubble functions, Bristeau *et al.* [17] also consider using different grids for the velocity and other variables. In particular, they use a fine grid for velocity, where this grid has been obtained from a coarse one by dividing each triangle into four, and the coarse grid for density and temperature. All the approximations are still linear but there are about twice as many velocity unknowns as density and temperature unknowns. Results in [17] indicate that, as one might expect, this approach also leads to oscillation-free solutions, at least at fairly low Reynolds numbers.

The approach outlined in this section of using the standard Galerkin method with different approximations for different variables is adequate at fairly low Reynolds numbers but becomes less useful when the Reynolds number is increased, as spurious oscillations begin to reappear. In the next section, we consider a modified Galerkin method by which accurate solutions can be obtained using piecewise linear approximations for all the variables.

2.5 Stable Schemes

The problem with using the Galerkin method for convection-dominated problems can be overcome by using a stabilized finite element method. This consists of adding extra mesh-dependent terms to the standard Galerkin method. It is designed to ensure that consistency is maintained, so that the solution of the original p.d.e.'s

is a still a solution to the discrete equations obtained via the modified method. We describe this approach in this section, first demonstrating it for a linear scalar problem, and then using it for the full Navier-Stokes equations.

Consider the following problem in two dimensions:

$$\mathbf{a}.\nabla u - \epsilon \Delta u = f \tag{2.68}$$

in the domain $\Omega \in \mathbf{R}^2$, with Dirichlet boundary conditions

$$u = g \tag{2.69}$$

given on $\partial\Omega$. The coefficient $\mathbf{a} = (a_1, a_2)^T$ is smoothly varying, and ϵ is a small constant. Even when the source term f and boundary data g are smooth, in general the solution u will vary rapidly in a layer of width $O(\epsilon)$ at the outflow boundary. For simplicity, we assume that $g = 0$, and first formulate the usual Galerkin method. Given a triangulation of the domain Ω, and a suitable function space $\mathcal{V}^h$, consisting of piecewise polynomial functions, find $u \in \mathcal{V}_0^h$ such that

$$(\mathbf{a}.\nabla u, v) + \epsilon(\nabla u, \nabla v) = (f, v) \; \forall v \in \mathcal{V}_0^h \tag{2.70}$$

where $\mathcal{V}_0^h = \{v \in \mathcal{V}^h : v = 0 \text{ on } \partial\Omega\}$ and

$$(u, v) = \int_\Omega uv d\Omega. \tag{2.71}$$

As noted in §2.3.1, when ϵ is small and the exact solution is not smooth, then the resulting numerical solution from this method contains spurious oscillations. A very simple way to overcome this is to use artificial diffusion, so the weak form (when $\epsilon < h$) becomes: find $u \in \mathcal{V}_0^h$ such that

$$(\mathbf{a}.\nabla u, v) + h(\nabla u, \nabla v) = (f, v) \; \forall v \in \mathcal{V}_0^h \tag{2.72}$$

where h is the mesh size parameter. The extra diffusion adds stability (which eliminates the oscillations) but modifies the problem actually being solved so the method is first order accurate at most.

A better approach is to only add extra diffusion in the direction of the streamlines, thus avoiding excess crosswind diffusion. This technique, introduced by Hughes and Brookes [65], modifies the weak form as follows: find $u \in \mathcal{V}_0^h$ such that

$$(\mathbf{a}.\nabla u, v) + \epsilon(\nabla u, \nabla v) + h(\mathbf{a}.\nabla u, \mathbf{a}.\nabla v) = (f, v) \; \forall v \in \mathcal{V}_0^h. \tag{2.73}$$

Although this method introduces less crosswind diffusion than (2.72) it is still an inconsistent modification of the original problem, hence an alternative way of introducing the term $h(\mathbf{a}.\nabla u, \mathbf{a}.\nabla v)$ into the weak form is needed. This may be done by altering the test function v to $v + \tau\mathbf{a}.\nabla v$ so that the weak form is now: find $u \in \mathcal{V}_0^h$ such that

$$\begin{aligned}(\mathbf{a}.\nabla u, v) + \epsilon(\nabla u, \nabla v) + \tau(\mathbf{a}.\nabla u, \mathbf{a}.\nabla v) + \epsilon\tau(\Delta u, \mathbf{a}.\nabla v) \\ = (f, v + \tau\mathbf{a}.\nabla v)\ \forall v \in \mathcal{V}_0^h.\end{aligned} \tag{2.74}$$

We need to define

$$(\Delta u, \mathbf{a}.\nabla v) = \sum_{e=1}^{n_{el}} \int_{\Omega_e} \Delta u \mathbf{a}.\nabla v \tag{2.75}$$

where n_{el} is the number of elements. There are several ways of defining τ, but for convection-dominated flows, it should have the property that

$$\tau = O\left(\frac{h}{|\mathbf{a}|}\right). \tag{2.76}$$

This method is both consistent (i.e. the exact solution u of the original problem is a solution of the weak form (2.74)) and stable, so that the spurious oscillations of the Galerkin method are avoided. Hughes and Brookes [65],[19] introduced the method, and refer to it as Streamline Upwind/Petrov-Galerkin (SUPG). Proofs concerning stability and error estimates for the problem above are given by Johnson *et al.* [78],[79] where the method is referred to as streamline diffusion.

In [66], Hughes *et al.* developed a successor to SUPG, known as Galerkin least-squares, where the test function is modified further to $(v + \tau(\mathbf{a}.\nabla v - \Delta v - f))$. In the cases where $\epsilon = 0$ or piecewise linear approximations are being used, this is identical to the SUPG method, but leads to a formulation which is more amenable to mathematical analysis.

These methods have been extended to systems of equations, see [68], and [70], as well as nonlinear problems [81]. The use of these methods has also been considered for problems such as the Euler equations [56], the incompressible Navier-Stokes equations [58] and the compressible Navier-Stokes equations [108]. In §2.5.1, we describe how we have used the Galerkin least-squares method for compressible flow in this thesis.

For problems involving sharp boundary layers and shocks, use of the methods outlined above still lead to overshooting and undershooting in these regions,

and so an extra term, in addition to the streamline diffusion term, is used. This discontinuity-capturing term acts in the direction of the solution gradient, rather than the streamline and reduces the oscillations that appear in sharp layers. It is described in detail in [67] and [71].

2.5.1 A Stable Method for the Navier-Stokes Equations

As discussed above, the Galerkin least-squares method for the compressible Navier-Stokes equations is given in [108] and is based upon the use of entropy variables. Hauke and Hughes [61] state that this method may also be used for other sets of variables, and in this section we describe a modified Galerkin least-squares method for the steady equations using primitive and conservative variables. The main differences from the method of Shakib *et al.* [108] are that no temporal discretization is carried out, a simpler choice of the parameter τ is used and the discontinuity capturing term is omitted. Further discussion of time-dependent problems, which require accurate time discretization, is given in chapter 7.

The Galerkin method (2.51) is modified by the addition of one term, the least-squares operator, and so the new variational form is: find $\mathbf{U}^h \in \mathcal{U}^h$, such that for all $\mathbf{V}^h \in \mathcal{V}^h$, the following is satisfied

$$\int_\Omega A_i \mathbf{U}^h_{,i}.\mathbf{V}^h + \mathbf{V}^h_{,i}.K_{ij}\mathbf{U}^h_{,j} - \mathcal{F}.\mathbf{V}^h)d\Omega + \sum_{e=1}^{n_{el}} \int_{\Omega_e} (\mathcal{L}^*\mathbf{V}^h.\tau\mathcal{L}^*\mathbf{U}^h)d\Omega$$

$$- \int_\Gamma \mathbf{V}^h.(K_{ij}\mathbf{U}^h_{,j})n_i d\Gamma = 0. \tag{2.77}$$

The choice of τ, which is not to be confused with the viscous stress tensor τ_{ij} in (2.7), is calculated over each element and is discussed below. The steady-state compressible Navier-Stokes operator $\mathcal{L}^*$ is defined as:

$$\mathcal{L}^* = A_i \partial/\partial x_i - (\partial/\partial x_i)(K_{ij}\partial/\partial x_j) - \mathcal{F}. \tag{2.78}$$

The additional least-squares term follows from generalizing the method for a single equation given in §2.5. In Shakib *et al.* [108], two definitions for the 4×4 matrix τ are derived, which involve the solution of eigenvalue problems, and so are computationally expensive. Hauke and Hughes [61] show how this τ may be transformed into a form suitable for other formulations, and an algebraic form of τ suitable for conservative variables is given by Soulaimani and Fortin [111]. In this

work however, we simply define

$$\tau = \begin{bmatrix} \frac{h}{2} & 0 & 0 & 0 \\ 0 & \frac{h}{2} & 0 & 0 \\ 0 & 0 & \frac{h}{2} & 0 \\ 0 & 0 & 0 & \frac{h}{2} \end{bmatrix} \tag{2.79}$$

where h is a mesh size parameter for each element. This has the advantage of being very cheap to evaluate in comparison to the more complicated definitions of τ. The disadvantage of this simple choice of τ is that it does not provide the optimal value required for *each* component of the system, however numerical results (see §4.6) suggest that for the types of flow under consideration here, the solutions obtained are not adversely affected by the use of (2.79). In addition, the term may need modification if used on meshes containing highly distorted elements.

2.6 Time-stepping

The previous sections (§2.3, §2.4 and §2.5) contained details of discretizations of the time-independent Navier-Stokes equations. The resulting nonlinear system of algebraic equations can then be solved by using techniques discussed in Chapter 3. However, this approach is only practical for very simple problems on coarse meshes. In most cases, the nonlinear solver will not be able to converge to a solution. This may be due to the initial solution estimate being too far away from the steady solution for the Newton solver to converge, or, if the linear solver being used is GMRES, having insufficient memory to store a large enough basis for the Krylov subspace.

An alternative to solving the steady-state problem directly is to solve the time-dependent equations, by approximating $\partial \mathbf{U}/\partial t$ in some way and marching forward in time, until a steady solution has been reached. Time-accuracy is not important, and this approach may be viewed as a form of numerical continuation. Problems involving transient solutions, where accuracy in time is required are considered in chapter 7.

Hence we now return to the full Navier-Stokes equations in quasi-linear form (2.36), which we wish to solve until time T, by which time the solution will have become steady. We divide the time interval $[0, T]$ into N sub-intervals $I_n =$

$[t_n, t_{n+1}]$, $n = 0 \ldots (N-1)$ with $0 = t_0 < t_1 < \ldots < t_N = T$. At each t_n, the solution of (2.36) is required, with the time-derivative $\partial \mathbf{U}/\partial t$ being approximated in some way, using a backward Euler scheme for example

$$\mathbf{U}_t \approx \frac{\mathbf{U}_n - \mathbf{U}_{n-1}}{\Delta t} \tag{2.80}$$

where $\mathbf{U}_n$ and $\mathbf{U}_{n-1}$ are the estimated discrete solutions at the current and previous time-steps, t_n and t_{n-1} respectively and Δt is the time-step size.

Using (2.80), a discrete variational equation for (2.36) can now be stated: at each t_n, $n = 1, \ldots, N$ find $\mathbf{U}_n^h \in \mathcal{U}^h$, such that for all $\mathbf{V}^h \in \mathcal{V}^h$, the following is satisfied

$$\begin{aligned} \int_\Omega \left[(A_0 \frac{\mathbf{U}_n^h - \mathbf{U}_{n-1}^h}{\Delta t} + A_i \mathbf{U}_{n,i}^h).\mathbf{V}^h + \mathbf{V}_{,i}^h . K_{ij} \mathbf{U}_{n,j}^h - \mathcal{F}.\mathbf{V}^h \right] d\Omega \\ + \sum_{e=1}^{n_{el}} \int_{\Omega^e} \mathcal{L}\mathbf{V}^h . \tau \mathcal{L}\mathbf{U}_n^h d\Omega - \int_\Gamma \mathbf{V}^h . (K_{ij}\mathbf{U}_{n,j}^h) n_i d\Gamma = 0, \end{aligned} \tag{2.81}$$

where suitable choices for Δt are discussed below in §2.6.1. The compressible Navier-Stokes operator $\mathcal{L}$ is defined as $A_0 \partial/\partial t + \mathcal{L}^*$. Rather than fix the final time value T in advance, it is sensible to detect when the solution has converged to steady-state, either by evaluating the steady-state residual (given by (2.56)-(2.59)) or monitoring $\mathbf{U}_t^h$, both of which should be zero at steady-state.

A simpler way to approximate $\partial \mathbf{U}^h/\partial t$ than the backward Euler scheme is to use of an explicit method, so that

$$\mathbf{U}_t^h = \frac{\mathbf{U}_{n+1}^h - \mathbf{U}_n^h}{\Delta t}, \tag{2.82}$$

which means that a very simple linear system, involving the mass matrix, is solved at each time-step. However the method is only conditionally stable, so that the time-step size Δt needs to be very small, and thus many steps are required to reach steady solution. This approach is commonly used, as some time-accuracy is maintained due to the small time-steps. However when only convergence to steady state is of importance then it is desirable to be able to take large time-steps.

The implicit backward Euler approach (2.80) is unconditionally stable, so there is no restriction imposed by stability requirements on the size of Δt and far fewer, larger steps are allowed. At each step, a nonlinear system needs to be solved, but this system is far more manageable than the one formed from the steady state equations directly. This is because the solution at the last time-step, which is used

as an initial guess at the current time-step, is close to the current solution and so convergence of the nonlinear system is easier to achieve (e.g. a smaller basis for the Krylov subspace is needed if GMRES is used). Although not implemented here, the calculation of a predicted initial guess based on an explicit Euler step would also help convergence.

For this reason, we have used backward Euler time-stepping to reach steady solutions of the Navier-Stokes equations. In the next section, we consider how to define the time-step, which can either be fixed in value throughout the domain or allowed to vary from element to element.

2.6.1 Global and Local Time-Stepping

For the time-stepping approach described above, the size of the time-step Δt needs to be chosen. A straightforward approach is to globally specify the same value everywhere in the domain. Since we wish to reach steady state as quickly as possible, this should be as large as the constraints on the nonlinear solver allow. Usually this means starting with an initial value, when the features of the flow are developing most quickly and increasing this as the solution begins to reach a steady-state.

A more efficient strategy when solving for steady-state solutions is to allow the value of Δt to vary over the domain. In problems which have large variations in mesh size, convective speed and diffusive properties, the use of a fixed time-step means that information about the flow propagates at different rates in different parts of the domain. As a result, convergence is slowed down because the rate may not be optimal on all parts of the domain.

If we determine Δt locally and in a suitable way, the flow information will propagate at nearly optimal rate throughout the domain. Shakib *et al.* [108] take the following approach for choosing Δt.

The algorithmic Courant number C_τ is defined as

$$C_\tau = 2(\Delta t/h^2)\hat{\lambda}_{\max} + 2(\Delta t/h^2)\bar{\lambda}_{\max}, \tag{2.83}$$

where h is the spatial mesh size parameter, and $\hat{\lambda}$ and $\bar{\lambda}$ are the upper bounds on the eigenvalues of two eigenvalue problems, involving the advective and diffusive matrices for the entropy formulation of the Navier-Stokes equations, stated in [108]. This definition is motivated by analysis of a one-dimensional linear convection-

diffusion problem discussed in [106]. Solution of the eigenvalue problems leads to

$$C_\tau = 2(\Delta t/h^2)\max(2\mu/\rho, \kappa/c_v\rho) + (\Delta t/h)u_\tau. \tag{2.84}$$

where μ is the viscosity coefficient, ρ is the density, κ is the coefficient of thermal conductivity, c_v is the specific heat at constant volume and

$$u_\tau = (u^2 + \frac{3}{2}c^2 + c\sqrt{16u^2 + c^2})^{\frac{1}{2}}, \tag{2.85}$$

with u and c being the particle and acoustic speeds. Since $\gamma = c_p/c_v = 1.4$, $Pr = \mu c_p/\kappa = 0.72$, and $Re = 1/\mu$

$$\frac{\kappa}{c_v} = \frac{\mu\gamma}{Pr} < 2\mu \tag{2.86}$$

and so

$$\max(2\mu/\rho, \kappa/c_v\rho) = \frac{2}{\rho Re}. \tag{2.87}$$

The local time-step size can now be chosen for each element so that C_τ is equal to a predetermined value set by the user. Rewriting (2.84),

$$\Delta t = \frac{C_\tau}{\left(\frac{2}{h^2\rho Re}\right) + \left(\frac{u_\tau}{h}\right)} \tag{2.88}$$

The choice for C_τ is discussed in [108], and values of between 20 and 100 appear to be suitable. As with global time-stepping, it is useful to allow C_τ (and hence Δt) to increase as steady-state is approached. Note that this method loses all accuracy in time, as it is meaningless to associate a solution $\mathbf{U}_n^h$ with a particular time, as different elements have different values of Δt, whereas global time-stepping still retains some consistency in time. However this doesn't matter when seeking steady solutions as the steady solution of both problems (local and global) is the same.

The implementation of this local time-stepping technique is simple due to the use an element-by-element approach when evaluating the residual. For each element, the local time-step size is computed and used in the element integral, from which the global residual vector is assembled. Local time-stepping is significantly quicker than global time-stepping to converge to a steady solution, as results in §4.6.2 show.

2.7 Summary

We have presented a finite element method for the solution of the compressible Navier-Stokes equations in two-dimensions, which leads to one or more finite sys-

tems of algebraic equations. These can then be solved using the algorithms discussed in the next chapter.

There are several ways of expressing the Navier-Stokes equations, each with advantages and disadvantages, but they can all be written in a general form to which a finite element method can be applied.

Use of the Galerkin finite element method on its own leads to unwanted oscillations appearing in the flow solution, so further techniques to overcome this are required. The addition of higher order approximations, such as bubble functions, for a subset of the variables eliminates the oscillations for flows at fairly low Reynolds numbers, but is insufficient when the flow is more convection-dominated. Stabilized methods such as SUPG, streamline diffusion and Galerkin least-squares are superior to bubble functions, as they work at higher Reynolds numbers and improve convergence, and this is demonstrated in chapter 4. We have implemented a variant of Galerkin least-squares.

In practice, the steady problem cannot be solved directly despite the global convergence techniques described in the next chapter, so time-stepping is used, and if the time-step size is chosen locally, convergence to steady-state can be rapid.

Results and comparisons using the methods outlined above for a number of standard test cases appear in chapter 4.

Chapter 3

Solution of a Nonlinear System of Algebraic Equations

3.1 Introduction

In the previous chapter, a finite element method was applied to the Navier-Stokes equations, leading to a system of nonlinear algebraic equations. Two approaches to obtaining steady-state solutions were discussed—solving the time-independent Navier-Stokes equations directly, or using an implicit time-stepping scheme to reach a steady solution. In the former case, a single nonlinear system of equations needs to be solved, while the latter case involves solution of a nonlinear system at each time-step. The solution technique for the nonlinear problem arising in either case is the same, and is the subject of this chapter.

We use Newton's method, which is described in §3.2, to solve each nonlinear problem. Some improvements to the basic method are outlined, and the use of the software package NKSOL discussed. Newton's method requires solution of a linear system at each iteration of the process, and a number of algorithms for solving this linear system are discussed in §3.3. The solver chosen for use in this work is the iterative solver GMRES which is described in detail, along with a suitable preconditioning matrix to improve convergence of GMRES. We also discuss some of the practical implementation issues.

3.2 Newton's Method

The algorithm for Newton's method is detailed in §3.2.1 below. Brown and Saad have developed NKSOL [21], a software package in which Newton's method is implemented, along with some enhancements to the basic method to improve its performance, and these are outlined in §3.2.2 and §3.2.3. A description of NKSOL is contained in the subsequent section, followed by some practical details of how the Jacobian matrix, which is required at each iteration of Newton's method, may be evaluated (given in §3.2.5).

3.2.1 The Algorithm

This section contains a description of the standard Newton algorithm which we use to solve the nonlinear system of equations arising from discretization of the Navier-Stokes equations (with or without time-stepping). The problem under consideration may be stated as: find $\mathbf{u}^* \in \mathbf{R}^n$ such that

$$\mathbf{G}(\mathbf{u}^*) = 0, \tag{3.1}$$

where $\mathbf{G} : \mathbf{R}^n \to \mathbf{R}^n$ is a nonlinear operator. Newton's method for solving (3.1) is as follows:

- Choose $\mathbf{u}_0$, an initial estimate of the solution,
- For i=0,1,... until convergence
  1. Solve
  $$J(\mathbf{u}_i)\delta\mathbf{u} = -\mathbf{G}(\mathbf{u}_i). \tag{3.2}$$
  2. Update
  $$\mathbf{u}_{i+1} = \mathbf{u}_i + \delta\mathbf{u}. \tag{3.3}$$

The $n \times n$ matrix $J(\mathbf{u})$ is the Jacobian of $\mathbf{G}(\mathbf{u})$,

$$[J(\mathbf{u})]_{ij} = \frac{\partial G_i(\mathbf{u})}{\partial u_j}. \tag{3.4}$$

Provided that the initial guess $\mathbf{u}_0$ is close enough to the exact solution $\mathbf{u}^*$, then Newton's method will converge to $\mathbf{u}^*$, and furthermore, the rate of convergence will generally be quadratic. For proofs of these results, see for example Dennis and Schnabel [37, chapter 5].

At each Newton step, the linear system of equations (3.2) needs to be solved, and many methods exist to do this. These can be divided into two classes—direct methods based on the use of Gaussian elimination, and iterative methods where the current approximation is updated at each iteration. These techniques are discussed in §3.3.

Newton's method is a very effective algorithm for solving a problem such as (3.1), and the quadratic rate of convergence allows solutions to be obtained quickly. However, as mentioned above, convergence will only occur when the initial estimate $\mathbf{u}_0$ is close enough to $\mathbf{u}^*$, so ideally some sort of modification is required to allow convergence when $\mathbf{u}_0$ is outside the local region around $\mathbf{u}^*$. In the next section, one such modification is outlined.

3.2.2 Obtaining global convergence

In this section, a modified Newton's method, which allows more global convergence than the standard method but retains its fast convergence rate near $\mathbf{u}^*$, is considered. This modification, known as linesearch backtracking, is described in detail by Dennis and Schnabel [37].

The basic idea of the algorithm is to perform Newton iteration as before, but rather than perform the update (3.3), a scalar λ is chosen so that the update becomes

$$\mathbf{u}_{i+1} = \mathbf{u}_i + \lambda\delta\mathbf{u} \tag{3.5}$$

instead. By choosing λ according to certain conditions given below, results exist [37] that show this modification can allow the method to obtain global convergence.

To see how this might work, the problem (3.1) is reformulated as a minimization problem: if

$$g(\mathbf{u}) = \frac{1}{2}\mathbf{G}^T(\mathbf{u})\mathbf{G}(\mathbf{u}), \tag{3.6}$$

then seek $\mathbf{u}^*$ such that

$$g(\mathbf{u}^*) = \min_{\mathbf{u}\in\mathbf{R}^n} g(\mathbf{u}). \tag{3.7}$$

A vector $\mathbf{p}$ is defined as a search direction if

$$\nabla g(\mathbf{u})^T\mathbf{p} < 0, \tag{3.8}$$

where

$$\nabla g(\mathbf{u}) = \left(\frac{\partial g}{\partial u_1}, \frac{\partial g}{\partial u_2}, .., \frac{\partial g}{\partial u_n}\right)^T. \tag{3.9}$$

If this holds, then it is guaranteed that for sufficiently small λ,

$$g(\mathbf{u} + \lambda\mathbf{p}) < g(\mathbf{u}). \tag{3.10}$$

It is clear that a search direction $\mathbf{p}$ may be useful in the solution of (3.7) (and hence the original problem (3.1)), provided λ can be determined. The update vector $\delta\mathbf{u}$ is shown to be such a search direction in the following way: at step i of the Newton process, $\mathbf{p}$ is a search direction if

$$\mathbf{G}(\mathbf{u}_i)^T J(\mathbf{u}_i)\mathbf{p} < 0, \tag{3.11}$$

because $\nabla g(\mathbf{u}) = J(\mathbf{u})^T\mathbf{G}(\mathbf{u})$. Since $\delta\mathbf{u} = -J(\mathbf{u}_i)^{-1}\mathbf{G}(\mathbf{u}_i)$,

$$\begin{aligned} \mathbf{G}(\mathbf{u}_i)^T J(\mathbf{u}_i)\delta\mathbf{u} &= -\mathbf{G}(\mathbf{u}_i)^T J(\mathbf{u}_i)J(\mathbf{u}_i)^{-1}\mathbf{G}(\mathbf{u}_i) && (3.12)\\ &= -\mathbf{G}(\mathbf{u}_i)^T\mathbf{G}(\mathbf{u}_i) && (3.13)\\ &< 0. \end{aligned}$$

Hence $\delta\mathbf{u}$ is a search direction. Now the scalar λ needs to be chosen such that

$$g(\mathbf{u} + \lambda\delta\mathbf{u}) < g(\mathbf{u}). \tag{3.14}$$

However, this won't always be a sufficient condition for global convergence (see examples in §6.3 of [37]), and further conditions, developed by Goldstein [50] are required:

$$\begin{aligned} g(\mathbf{u} + \lambda\delta\mathbf{u}) &\leq g(\mathbf{u}) + \alpha\lambda\nabla g(\mathbf{u})^T\delta\mathbf{u} && (3.15)\\ g(\mathbf{u} + \lambda\delta\mathbf{u}) &\geq g(\mathbf{u}) + \beta\lambda\nabla g(\mathbf{u})^T\delta\mathbf{u} && (3.16) \end{aligned}$$

for $0 < \alpha < \beta < 1$. These ensure that the relative size of λ compared to the rate of decrease of g is neither too large nor too small.

If at each Newton step, a value for λ is chosen in this way, then proofs given in [37] show that this method will ensure global convergence. Moreover, near the exact solution $\mathbf{u}^*$, if $\lambda = 1$, then the method reverts to standard Newton and the rate of convergence becomes quadratic. A practical implementation of linesearch backtracking, including an algorithm to choose a λ to satisfy (3.15) and (3.16) is given by Brown and Saad in [21]. An alternative technique is also given, using a model trust region strategy.

Thus the modified Newton method described above overcomes one of the main problems associated with using Newton's method alone by no longer requiring $\mathbf{u}_0$ to be in the neighbourhood of $\mathbf{u}^*$ (although the rate of convergence will only be quadratic once $\mathbf{u}_i$ is close enough to $\mathbf{u}^*$).

3.2.3 Inexact Methods

The linear system (3.2) can be solved in a number of ways. In particular, an iterative method, where the current solution is made more and more accurate at each iteration, may be used. In this case, another modification, which reduces the time taken by Newton's method to reach a solution, can be made and is described in this section.

The inexact Newton method of Dembo *et al.* [34] only requires the iterative solver to find an approximate solution to (3.2), rather than solving the linear system to a high degree of accuracy. This will reduce the time spent solving the linear system, and, if the level of approximation is chosen carefully, won't affect the rate of convergence of the Newton iteration.

Define the residual $\mathbf{r}_i$ at the ith Newton step as

$$\mathbf{r}_i = J(\mathbf{u}_i)\delta\mathbf{u} + \mathbf{G}(\mathbf{u}_i), \tag{3.17}$$

and a sequence of scalars $\{\eta_i\}$. The equations to be solved at each Newton step become:

$$J(\mathbf{u}_i)\delta\mathbf{u} = -\mathbf{G}(\mathbf{u}_i) + \mathbf{r}_i, \text{ with } \frac{||\mathbf{r}_i||}{||\mathbf{G}(\mathbf{u}_i)||} \leq \eta_i, \tag{3.18}$$

where $||\cdot||$ denotes an arbitrary norm in $\mathbf{R}^n$. This may be seen as iteratively solving (3.2) until $||\mathbf{r}_i|| \leq \eta_i||\mathbf{G}(\mathbf{u}_i)||$. In [34] it is shown that for a sequence $\{\eta_i\} \leq 1$ convergence is guaranteed, when $\mathbf{u}_0$ is sufficiently close to $\mathbf{u}^*$. However, the rate of convergence will depend on the choice of the sequence $\{\eta_i\}$, and it can be shown (see [34]) that to obtain quadratic convergence,

$$\eta_i = O(||\mathbf{G}(\mathbf{u}_i)||), \text{ as } i \to \infty. \tag{3.19}$$

Hence in practice, the sequence $\{\eta_i\} \to 0$ in order to preserve a quadratic rate of convergence and as $\mathbf{u}_i$ approaches $\mathbf{u}^*$, η_i will be very small, so the method will effectively revert to exact Newton.

This technique is used in NKSOL (see next section) and significantly reduces the amount of time spent solving the linear systems, especially in the early stages of the algorithm.

3.2.4 NKSOL

The software package NKSOL (Nonlinear Krylov SOLver) has been developed by Brown and Saad [21] to implement some of the ideas described above. It uses an

inexact Newton method combined with the iterative solvers GMRES (§3.3.2) or Arnoldi's method ([2]) as the linear solver. In addition, the software offers a choice of either the linesearch backtracking algorithm (§3.2.2) or a strategy based on model trust region techniques to improve the otherwise local convergence of Newton ([21]).

As this is an inexact Newton method, the sequence η_i used in (3.18) needs to be defined:

$$\eta_i = \frac{1}{2^i}. \tag{3.20}$$

Results indicate that quadratic convergence is usually achieved using (3.20).

In §3.3.3, the importance of preconditioners for iterative solvers (such as GMRES) are discussed. However NKSOL does not contain any preconditioning routines, so we use another set of subroutines (SLAP—see §3.3.5) to provide a preconditioned GMRES routine.

NKSOL only requires the user to write subroutines to evaluate the residual $\mathbf{G}(\mathbf{u})$, and (optionally) provide the Jacobian matrix. This solver (in combination with SLAP) has been used to obtain all of the results shown in this thesis.

3.2.5 Evaluation of Jacobian

This section contains details of how the Jacobian matrix $J(\mathbf{u}_i)$, which is used in the solution of the linear system (3.2) arising at each Newton step, is evaluated. A number of methods are shown, most of which form the sparse matrix J explicitly, requiring storage space (§3.3.6 gives details of sparse matrix storage). The final method assumes that J will only be referred to as part of a matrix-vector multiplication, and so the matrix is not calculated or stored. This method can only be used with an iterative type of linear solver, as direct solvers require the full matrix to be available.

(i) Exact Jacobian

In this approach the Jacobian is evaluated by analytically finding the derivatives of $\mathbf{G}(\mathbf{u})$ in advance and coding them as a Fortran subroutine. For example, a typical term in the residual obtained from (2.56)–(2.59) in §2.3 might look like

$$G_i = \ldots + \int_\Omega \rho^h \left(\sum_{k=1}^{n_{int}+n_{out}} u_k \phi_k + \sum_{k=n_{nod}-n_{in}+1}^{n_{nod}} \cos\alpha\phi_k \right) \left(\sum_{k=1}^{n_{int}+n_{out}} u_k \phi_{k,x} + \sum_{k=n_{nod}-n_{in}+1}^{n_{nod}} \cos\alpha\phi_{k,x} \right) \phi_i d\Omega + \ldots \tag{3.21}$$

In this case the entry in J would be

$$\frac{\partial G_i}{\partial u_j} = \ldots + \int_\Omega \rho^h \phi_j \left(\sum_{k=1}^{n_{int}+n_{out}} u_k \phi_{k,x} + \sum_{k=n_{nod}-n_{in}+1}^{n_{nod}} \cos\alpha\phi_{k,x} \right) \phi_i d\Omega + \int_\Omega \rho^h \left(\sum_{k=1}^{n_{int}+n_{out}} u_k \phi_k + \sum_{k=n_{nod}-n_{in}+1}^{n_{nod}} \cos\alpha\phi_k \right) \phi_{j,x} \phi_i d\Omega. \tag{3.22}$$

The matrix J is formed by assembling the contributions from each local element matrix. For details on assembling global matrices in this way see Johnson [77]. This matrix is sparse, and although nonsymmetric, has a symmetric structure. The sparsity of the Jacobian allows it to be stored in a relatively compact format (see §3.3.6), but it still uses up a large amount of memory (approximately 20–30 $\times$ the number of unknowns).

One difficulty associated with calculating the Jacobian in this way is that it is prone to errors from incorrect coding of derivatives. Using an approach such as Galerkin least-squares (see §2.5), where extra terms are incorporated into the variational formulation, the algebra becomes very involved, increasing the likelihood of mistakes. This can be seen in (3.21)—if the test function ϕ_i is replaced by $\phi_i + \tau(\ldots)$ then the dependency of the term on u_j may become highly nonlinear and algebraically complicated. Furthermore if τ is calculated using the current solution in some way, then obtaining this dependence analytically may be impossible.

This evaluation difficulty can be overcome (at least when τ is independent of $\mathbf{u}$) by using one of the computer algebra packages now available, such as Maple [44] or Mathematica [127]. They will symbolically evaluate the Jacobian, given a residual function, and generate source code which can be directly inserted into a Fortran subroutine. Even this is not a straightforward task however, when the problem being solved consists of a system of p.d.e.'s, with several different sets of unknowns (density, velocity and temperature for example).

(ii) Global Finite Difference Approximation

An alternative to the above is to approximate the derivatives in some way, and given a routine to calculate $\mathbf{G}(\mathbf{u})$, the following approximation can be used to evaluate the product $J(\mathbf{u})\mathbf{v}$:

$$J(\mathbf{u})\mathbf{v} \approx \frac{\mathbf{G}(\mathbf{u}+\sigma\mathbf{v}) - \mathbf{G}(\mathbf{u})}{\sigma}, \tag{3.23}$$

where σ is a small scalar, the choice of which is discussed at the end of this section. This approximation (3.23) can also be used to form the full Jacobian matrix ([37]): the jth column of $J(\mathbf{u}_i)$ is given by

$$J(\mathbf{u}_i)\mathbf{e}_j \approx \frac{\mathbf{G}(\mathbf{u}_i+\sigma\mathbf{e}_j) - \mathbf{G}(\mathbf{u}_i)}{\sigma}, \tag{3.24}$$

where $\mathbf{e}_j$ is the vector length of length n with 1 in the jth position, and 0 everywhere else. However this involves $n+1$ evaluations of the residual $\mathbf{G}$ each time the Jacobian is recalculated, and so is too expensive to be practical.

(iii) Element Finite Difference Approximation

A more useful idea is to use (3.24) to compute *element* Jacobian matrices by finite differences and then assemble these to form an approximate global Jacobian. For the Navier-Stokes equations on a triangular mesh, each element matrix is of size 12 $\times$ 12 (there are four unknowns on each of the three vertices), and so 13 evaluations of the element residual are required for each element. This is far quicker than method (ii) and also turns out to be about 30% quicker than using method (i) (see results in §4.6.2). This method also overcomes the problems associated with method (i) of coding the analytic derivatives exactly. We are unaware of this approach having been used before, although it seems to be a natural progression from the methods (i) and (ii).

(iv) Matrix-Free Methods

If an iterative linear solver is being used in (3.2), then usually the only time that the Jacobian is needed is for a matrix-vector multiply operation, hence avoiding the need for it to be formed explicitly. The approximation (3.23) is all that is required, and no matrix needs to be stored. However, certain preconditioners, such as incomplete LU decomposition (see §3.3.3), work by factorizing the matrix in some way. If a matrix-free method is implemented, then the matrix entries are not

available. In this case such preconditioners may not be used, and an alternative, such as domain decomposition preconditioning [29] is required.

For all of the approximate methods using (3.23), σ needs to be chosen so that the approximation is accurate and the fast convergence of Newton is not lost. In [37] the choice of σ is discussed and it is shown that the quadratic convergence rate of Newton is preserved for small enough σ. In [20], Brown shows that for a suitable choice of σ, use of (3.23) retains local convergence for inexact Newton methods which use Krylov-subspace iterative solvers. In our numerical experiments, it appears to be sufficient, in the case of the Navier-Stokes equations, to choose σ simply to be $\sqrt{\text{macheps}}$ where macheps is a machine specific value defined as the smallest positive number ϵ such that $1 + \epsilon > 1$ on that machine.

Four methods for evaluating the Jacobian matrix have been outlined, and of these, methods (iii) and (iv) seem to be the most useful. If an iterative solver combined with a preconditioner which doesn't require the Jacobian matrix is being used, then a matrix-free method will be suitable as no matrix storage is required, otherwise method (iii) is possibly the best choice. In this work we use an iterative solver (GMRES), but the preconditioner is an incomplete factorization of the Jacobian, so method (iii) has been selected.

3.3 Solution of Linear System

Newton's method for solving a nonlinear set of equations, described in §3.2, leads to a linear system (3.2) at each Newton step, and in this section, methods for solving such systems are discussed. Various methods are compared in §3.3.1 and in §3.3.2 GMRES, an iterative solver, is described in further detail. Preconditioning a linear system in order to improve the convergence rate of an iterative solver is often necessary, and some common types are given in §3.3.3, along with a more detailed explanation of incomplete LU factorization in §3.3.4. The use of SLAP, a set of subroutines containing a preconditioned GMRES algorithm, is discussed in §3.3.5. One example of a data structure used to store a sparse matrix is given in §3.3.6.

The general form of the problem under consideration may be written as

$$A\mathbf{x} = \mathbf{b}, \tag{3.25}$$

where A is a non-singular $n \times n$ sparse matrix, $\mathbf{b}$ is a known vector and $\mathbf{x}$ is the

unknown vector. We denote the exact solution to (3.25) as $\mathbf{x}^*$. The Newton step (3.2) is one example of such a system, and in this case the matrix is nonsymmetric and indefinite.

3.3.1 Iterative versus Direct Solvers

Methods of solution to (3.25) can be classed in two distinct groups.

Direct Methods

Direct methods use Gaussian elimination (see [22] for example) to obtain a factorization of the matrix A in terms of lower(L) and upper(U) triangular matrices, so that $A = LU$. This can then easily be solved by performing forward and back substitutions,

$$L\mathbf{v} = \mathbf{b},\ U\mathbf{x} = \mathbf{v}. \tag{3.26}$$

When n is large, it is impractical to store the entire matrix in memory and carry out the elimination in this way. To overcome this problem, the frontal method [72] may be used, where the assembly of A from the element matrices is carried out at the same time as the elimination process is performed. Only that part of the matrix currently being worked on is stored in fast memory, the remainder of A is stored in secondary memory.

Another technique for reducing the amount of memory required is to reorder the unknowns, so that the structure of the matrix is altered. If done in the right way, this has the effect of reducing the number of nonzero entries needed during the elimination. One effective type of reordering is nested dissection, described in [47].

Iterative Methods

In iterative methods, the current approximation $\mathbf{x}_i$ to the solution of (3.25) is updated at each step to obtain a better approximate solution $\mathbf{x}_{i+1}$. A number of iterative algorithms are discussed below, but for a general overview of such methods and details of practical implementation, see Barret *et al.* [11].

The simplest types of these methods (see [121]), such as Jacobi or Gauss-Seidel, are inadequate for most practical problems because the rate of convergence is very slow, so a more complex class of nonstationary methods which use Krylov subspaces

is considered. A Krylov subspace K_k is defined as

$$K_k = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0\} \tag{3.27}$$

where $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$. At each iteration k, if the current solution of (3.25) is $\mathbf{x}_k$, a Krylov method will try to find an update vector $\mathbf{z}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ from K_k, by generating an orthogonal set of basis vectors for K_k.

For positive definite symmetric matrices the conjugate gradient method, developed by Hestenes and Stiefel [62], is widely used. Updates $\mathbf{z}_k \in K_k$ are generated in such a way that for the updated solution $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{z}_k$,

$$(\mathbf{x}_{k+1} - \mathbf{x}^*)^T A(\mathbf{x}_{k+1} - \mathbf{x}^*) = \min_{\mathbf{z} \in K_k} (\mathbf{x}_k + \mathbf{z} - \mathbf{x}^*)^T A(\mathbf{x}_k + \mathbf{z} - \mathbf{x}^*). \tag{3.28}$$

One major feature of the conjugate gradient method is that the orthogonal basis for the Krylov subspace K_k can be constructed with only three-term recurrences, so very little storage is required. In the algorithm this is implemented with two two-term recurrences—one to update the residual using the current search direction (the latest orthogonal basis vector) and another to update the search direction using the newly computed residual.

For matrices which are not symmetric, such as the Jacobian in (3.2), the basis for K_k can no longer be built up from a short-term recurrence in such a straightforward way. One approach to overcome this is used in the Biconjugate gradient method (BiCG) [88], and involves generating two mutually orthogonal sequences of residuals. The short-term recurrences are retained but the minimization property of conjugate gradient is lost. As a result, convergence is irregular and breakdowns in the algorithm are liable to occur. Several variants and improvements of BiCG have been developed, including Bi-CGSTAB [119], CGS [110] and Quasi-Minimal Residual (QMR) [45].

Another approach is to construct an orthogonal basis for K_k by storing all the previously computed vectors, and using them in the recurrence relation. Saad and Schultz have introduced GMRES (Generalized Minimal Residual) [104], which does this, and hence there is a similar minimization property to that of the conjugate gradient method. The drawback is that the storage costs are much higher for GMRES than for other nonsymmetric solvers. This algorithm is described in more detail in the next section.

Although direct methods provide a reliable means of reaching a solution, for large problems their storage requirements are very high. Iterative methods use less memory, and may be more suitable for use in Newton's method, where the linear system doesn't always have to be solved to high accuracy (see §3.2.3), thus reducing the number of iterations. Shakib *et al.* [107] have carried out a comparison between a direct solver and GMRES in the case of compressible viscous flow, and show that the iterative solver offers both faster solution time and reduced memory requirements.

3.3.2 GMRES

This section describes the algorithm for GMRES, an iterative method for solving (3.25) when A is nonsymmetric. The algorithm (as given in [104]) for GMRES is concerned with obtaining an update vector at each iteration until the solution is sufficiently accurate so that the iteration process ends and the new solution $\mathbf{x}$ can be formed. It can be stated as follows:

1. Choose $\mathbf{x}_0$ and calculate $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\beta = ||\mathbf{r}_0||_2$, and $\mathbf{v}_1 = \mathbf{r}_0/\beta$.

2. For j=1,2,..,until $||\mathbf{r}_j||_2 < \epsilon$,

$$\begin{aligned} h_{ij} &= (A\mathbf{v}_j, \mathbf{v}_i),\ i = 1, 2, .., j, \\ \hat{\mathbf{v}}_{j+1} &= A\mathbf{v}_j - \sum_{i=1}^{j} h_{ij}\mathbf{v}_i, \\ h_{j+1,j} &= ||\hat{\mathbf{v}}_{j+1}||_2, \\ \mathbf{v}_{j+1} &= \hat{\mathbf{v}}_{j+1}/h_{j+1,j}. \end{aligned} \tag{3.29}$$

3. Form the approximate solution, with $k = j$,

$$\mathbf{x}_k = \mathbf{x}_0 + V_k\mathbf{y}_k, \tag{3.30}$$

where $\mathbf{y}_k$ minimizes

$$||\beta\mathbf{e}_1 - \bar{H}_k\mathbf{y}||_2. \tag{3.31}$$

Here, $\mathbf{r}_j = \mathbf{b} - A\mathbf{x}_j$, ϵ is the specified tolerance, $\bar{H}_k$ is a $(k+1)\times k$ upper Hessenberg matrix whose elements are given by h_{ij}, $V_k = [\mathbf{v}_1, \mathbf{v}_2, \ldots \mathbf{v}_m]$, an orthonormal basis for the Krylov subspace K_k, and $\mathbf{e}_1 = (1, 0, 0, \ldots, 0)^T$. We now show that $\mathbf{y}_k$ is chosen such that the residual norm $||\mathbf{r}_k||_2$ is minimized over $\mathbf{y} \in K_k$.

H_k is $k \times k$ upper Hessenberg matrix whose elements are given by h_{ij} and

$$\begin{aligned}
AV_k &= [A\mathbf{v}_1, \ldots, A\mathbf{v}_k] \\
&= [\hat{\mathbf{v}}_2 + \sum_{i=1}^{1} h_{i1}\mathbf{v}_i, \ldots, \hat{\mathbf{v}}_{k+1} + \sum_{i=1}^{k} h_{ik}\mathbf{v}_i] \quad \text{(from (3.29))} \\
&= [\sum_{i=1}^{2} h_{i1}\mathbf{v}_i, \ldots, \sum_{i=1}^{k+1} h_{ik}\mathbf{v}_i] \\
&= [\sum_{i=1}^{2} h_{i1}\mathbf{v}_i, \ldots, \sum_{i=1}^{k} h_{ik}\mathbf{v}_i] + \hat{\mathbf{v}}_{k+1}\mathbf{e}_k^T \\
&= [\sum_{i=1}^{k} h_{i1}\mathbf{v}_i, \ldots, \sum_{i=1}^{k} h_{ik}\mathbf{v}_i] + h_{k+1,k}\mathbf{v}_{k+1}\mathbf{e}_k^T \\
&= V_k H_k + h_{k+1,k}\mathbf{v}_{k+1}\mathbf{e}_k^T \\
&= V_{k+1}\bar{H}_k, \qquad (3.32)
\end{aligned}$$

where $\mathbf{e}_k$ is the vector which has the value 1 in the k'th component, and 0 elsewhere. Minimizing (3.31) is equivalent to minimizing $||\mathbf{r}_k||_2$ because

$$\begin{aligned}
||\bar{H}_k\mathbf{y}_k - \beta\mathbf{e}_1||_2 &= ||V_{k+1}(\bar{H}_k\mathbf{y}_k - \beta\mathbf{e}_1)||_2 \quad (V_{k+1} \text{ is orthonormal}) \\
&= ||V_{k+1}\bar{H}_k\mathbf{y}_k - \beta\mathbf{v}_1)||_2 \\
&= ||AV_k\mathbf{y}_k - \mathbf{r}_0||_2 \quad \text{(using (3.32))} \\
&= ||A(\mathbf{x}_k - \mathbf{x}_0) - (\mathbf{b} - A\mathbf{x}_0)||_2 \quad \text{(using (3.30))} \\
&= ||A\mathbf{x}_k - \mathbf{b}||_2 \\
&= ||\mathbf{r}_k||_2. \qquad (3.33)
\end{aligned}$$

Hence minimization of (3.31) also minimizes the current residual. This minimization problem is solved using a QR factorization of $\bar{H}_k$ and more details are given in [104]. The factorization of $\bar{H}_k$ is updated as each column is computed, which allows the residual norm $||\mathbf{r}_k||_2$ of the approximate solution to be obtained without actually forming $\mathbf{x}_k$. Only when $||\mathbf{r}_k||_2 < \epsilon$ will the solution vector $\mathbf{x}_k$ be calculated. When used as part of an inexact Newton method, the tolerance ϵ will be determined by the sequence η_i discussed in §3.2.3.

If the iteration in step 2 above is allowed to continue long enough, then the exact solution will always be found in at most n steps, in the absence of rounding errors. However since the dimension of V_k is increased at each step, memory restrictions prevent k becoming large (each iteration requires storage of an additional vector of length n). There is also a drop in speed as k increases since the formation of V_k

and the QR factorization take longer at each step. To overcome this problem, it is common to restart the whole algorithm after m steps with the current solution $\mathbf{x}_m$ used as $\mathbf{x}_0$ in the new process. This means that no more than $n \times m$ storage locations are required but, for large problems, this may not always be successful (because the solution may not converge), so it is desirable to set m to as large a value as the storage facilities allow (see §3.3.5 for more details).

In spite of the large memory overheads, GMRES, in combination with a suitable preconditioner, has been shown to be an effective solver to use in the case of compressible flow. For example Bristeau *et al.* [17], Venkatakrishnan and Mavriplis [122], Shakib *et al.* [107] and Soulaimani and Fortin [111] have all successfully obtained results using GMRES.

3.3.3 Preconditioning

In many practical applications, the matrix A in (3.25) is badly conditioned, and as a consequence the convergence of an iterative solver will either be very slow, or the method will fail to converge altogether. In addition, when the storage costs of a solver depend upon the number of iterations (as with GMRES for example), then it is important to reduce the number of steps taken by increasing the rate of convergence.

In this section the concept of preconditioning, which helps to overcome this slow convergence by transforming the linear system, is discussed. A preconditioning matrix M is applied to the problem (3.25), which leads to a matrix with a smaller condition number than that of the original. This preconditioning of the system is done either by multiplying through by M^{-1},

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b} \quad \text{(left preconditioning)}, \tag{3.34}$$

or rewriting it as

$$AM^{-1}M\mathbf{x} = \mathbf{b} \quad \text{(right preconditioning)}. \tag{3.35}$$

Alternatively, both left and right preconditioning can be combined. The additional costs in using a preconditioner are twofold—the initial construction of M and the solution of $M\mathbf{y} = \mathbf{z}$ at each step. M should ideally be chosen such that these costs are minimized, and that the convergence of (3.25) is significantly improved (which can be done by ensuring $M \approx A$).

Jacobi preconditioning yields one of the simplest forms that M may take,

$$M = \text{diag}\{A\}. \tag{3.36}$$

This preconditioner is trivial to implement, and in a case where A is diagonally dominant works well, otherwise the gains in rate of convergence are relatively small compared to more sophisticated choices of M.

A very common such class of preconditioners are those consisting of incomplete LU factorizations of A, which involve forming M as a product of upper and lower triangular matrices based upon an approximate decomposition of A, so that certain entries are ignored. This idea was introduced initially for symmetric matrices by Meijerink and van der Vorst [94] and subsequently extended to nonsymmetric systems [118]. A more detailed description of ILU factorization may be found in §3.3.4.

Alternatives to the this type of preconditioner include the element-by-element (EBE) approach, where the element matrix is preconditioned in some way, before assembly of the global matrix (if the assembly stage is done at all). Hughes *et al.* [69] introduced the concept of EBE factorizations in a structural mechanics context, and in [124] Wathen considered EBE preconditioners for the conjugate gradient method. Other work in this area includes that of Gustafsson and Lindskog [55] and van Gijzen [120]. For two-dimensional problems, global preconditioners are usually preferable, but the situation may be different in three dimensions [124].

Another preconditioning technique is that of domain decomposition. Particularly effective for parallel computers, the approach here is to partition the domain into subdomains which can be handled separately. The two alternative methods are Schwarz methods, where the domains are overlapping, and Schur complement methods where the domains are separated by interfaces. For further details and references concerning domain decomposition methods see [29].

3.3.4 ILU Factorization

As mentioned above, one important class of preconditioner consists of using approximate, or incomplete, factorizations of A, as described in this section. To obtain an incomplete factorization, a set S containing a subset of elements of A is chosen, so that during the factorization process, any elements lying outside S will be discarded. A convenient choice for S when A is sparse is the set of nonzero entries

in A, so that the sparsity pattern is preserved and the same data structure can be used to store the factorization. These elements have fill level zero, where a nonzero entry in the factorization is said to have a fill-level of $k+1$ if it is caused by an entry of fill-level k. Hence this is referred to as the ILU(0) preconditioner, whereas ILU(1) will contain more nonzero entries than the original matrix A. As the level of fill increases, the preconditioner becomes more effective as well as needing more memory to store the factorization.

The incomplete decomposition can be written as

$$\text{for each } k = 1, .., n,\ i, j > k,\ a_{ij} \leftarrow \begin{cases} a_{ij} - a_{ik} a_{kk}^{-1} a_{kj} & \text{if} (i,j) \in S \\ a_{ij} & \text{otherwise.} \end{cases} \tag{3.37}$$

In some cases, where structured meshes are being used (and so the matrix has a regular structure), it is possible to show that the ILU(0) preconditioner will have the same off-diagonal elements as the original matrix [95] but for unstructured meshes this will not be so. Having computed the approximate factorization, $LU\mathbf{y} = \mathbf{z}$ can be solved at each iteration as

$$L\mathbf{v} = \mathbf{z},\ U\mathbf{y} = \mathbf{v}. \tag{3.38}$$

A variation of this incomplete factorization is the modified incomplete LU preconditioner (MILU), where rather than discard any nonzero elements lying outside S, they are subtracted from the diagonal. In this way, the preconditioner has the same rowsum as the original matrix. Although there is a risk of breakdown during the factorization, it has been shown for certain cases (by Gustafsson [54] for example) that the behaviour of the condition number of the preconditioned system improves.

The type of ILU factorization described so far refers to treating each unknown separately, but another approach is to group together sets of unknowns and perform blockwise (rather than pointwise) factorization. There are a number ways of grouping together sets of unknowns, including by node (giving four unknowns in each block for the 2-d compressible Navier-Stokes equations) or by processor if the algorithm is to be run in parallel. The decomposition is then based on blocks, which means that the blocks will have to be inverted. If the block size is small, then inversion will not be too slow, however some type of approximate inverse may be needed for larger sized blocks. Further details on such block preconditioners are found in Axelsson [3] and Concus *et al.* [31]

Dutto [40] has looked at the importance of how the unknowns are ordered in the case of block ILU preconditioning for the Navier-Stokes equations and has shown that certain ordering algorithms, such as reverse Cuthill-Mcgee, will improve the preconditioner and accelerate convergence of GMRES. Ordering of unknowns has also been investigated by Venkatakrishnan and Mavriplis [122], with similar results. They also compared a number of preconditioners for both inviscid and viscous compressible flow, with results showing that block ILU was the most successful.

We have used a pointwise ILU(0) preconditioner, as this has been implemented in SLAP (see next section). Results in chapter 4 seem to indicate that this is a successful approach, although the use of block ILU, node reordering or ILU(1) would possibly lead to a further acceleration in the convergence rate.

3.3.5 SLAP

GMRES (along with a predecessor of GMRES, Arnoldi [2]) is implemented in NKSOL, but without a preconditioner. SLAP (Sparse Linear Algebra Package) written by Seager and Greenbaum [105], is a set of routines for solving sparse linear systems and includes several iterative solvers including GMRES. It also contains some preconditioners such as diagonal scaling and pointwise ILU(0). We have used the ILU and GMRES routines so that they are called by NKSOL, at each Newton step, to approximately solve the linear system.

One parameter required by the SLAP routine is m, the maximum dimension of the Krylov subspace. If this is too small, then the GMRES solver will fail to converge, but since the storage required by the solver is proportional to $m \times n$, m is restricted by the amount of memory available. In §4.6.2 the choice of m is investigated for the Navier-Stokes equations, along with the question of whether restarting the GMRES process is worthwhile.

3.3.6 Storage of Sparse Matrices

The $n \times n$ Jacobian matrix J arising in (3.2) is sparse, the structure of which is determined by the current mesh on which the problem is being solved. Typically less than 1% of the matrix elements will be nonzero, so clearly a more efficient type of matrix data storage is needed than simply storing all of the matrix entries. One possibility is to use a matrix-free method (see §3.2.5), but in this section a variation

of the Harwell-Boeing sparse matrix format [39] is described. It is the format used by SLAP (see §3.3.5) and throughout this work.

The matrix J is specified by three arrays. Nonzero elements are stored in `VAL`, in columnwise order, with the diagonal on each column being stored before the remaining elements of that column. `ROW` stores the row index of the corresponding element in `VAL`. The values in `COL` point to elements in `VAL` and `ROW` which represent the diagonals of each column.

So a diagonal entry $J_{i,i}$ is stored in `VAL`[`COL`[i]], and the other elements in column i are stored in `VAL`[`COL`[i]+1]...`VAL`[`COL`[i+1]-1], and their row indices in `ROW`[`COL`[i]+1]...`ROW`[`COL`[i+1]-1]. In addition, if $nzelt$ is the number of nonzero elements in J, the assignment `COL`[n+1]=$nzelt+1$ is made. For example if

$$J = \begin{bmatrix} 4 & 0 & 2 & 0 & 0 \\ -1 & 2 & 0 & 0 & 1 \\ 0 & 1 & 5 & 0 & 0 \\ 3 & 0 & 2 & 2 & 0 \\ 0 & 2 & -1 & 0 & 2 \end{bmatrix}$$

then the arrays `VAL`,`ROW` and `COL` become:

```
VAL = 4 -1 3 2 1 2 5 2 2 -1 2 2 1
ROW = 1 2 4 2 3 5 3 1 4 5 4 5 2
COL = 1 4 7 11 12 14
```

This format uses a minimum amount of memory in that no nonzero elements are stored, but is inefficient since procedures such as matrix vector multiplications involve an indirect addressing step for every scalar operation. It differs from the Harwell-Boeing format in that the diagonal elements (which will always be nonzero in this case) are always listed as the first element in the column. The advantage of this modification, which is used by SLAP, is that the diagonals can be obtained more quickly than if they appear further down the list of column entries (useful, for example, if Jacobi preconditioning is being used).

The creation of the arrays `VAL`, `ROW` and `COL` is achieved using information about the unstructured mesh. In fact, it is efficient to separate the construction of `ROW` and `COL` (the structure of J) which need only be done once per new mesh, from the calculation of `VAL` which is done every time the Jacobian is re-evaluated, at every Newton step.

It is a nontrivial task to construct ROW and COL because the length of each column is not known in advance. The stages for doing this can be summarised as follows.

- Initially assume that each column will have no more than MAXNZ elements in it, and create a two dimensional $n \times$ MAXNZ array.
- Loop through all the mesh triangles. On each triangle, all 12 (4 variables for each vertex) unknowns have a global number (between 1 and n) associated with them, $g_1..g_{12}$. The entries $[J]_{g_i,g_j}$ will all be nonzero elements and hence their row indices should be stored in the appropriate column, if they have not already been stored. This is complicated by the fact that on boundaries, one or two of the vertices will have a reduced number of unknowns, due to Dirichlet boundary conditions.
- The two dimensional array still contains many unused entries (since only very few of the columns will have MAXNZ elements) and so the data is rewritten as a continuous one dimensional array ROW with elements in the correct order. COL can be formed as this process is carried out.

The temporary two dimensional array which is initially created is much larger than the final array ROW but this extra space can be used elsewhere (in the GMRES linear solver, for example). If a language other than Fortran is used (such as C or Fortran-90), then dynamic memory allocation can be used, and the above process is unnecessary.

Once COL and ROW have been created, the array VAL can be updated each time the Jacobian is recalculated. This is obtained from the assembly of the mesh element Jacobians, by adding values to the entries of J. Addition of a value x to row i, column j of J is as follows.

- Address of diagonal of column j is COL[j]
- If $i = j$ then VAL[COL[j]]=VAL[COL[j]]+x
- Otherwise $k = 1, 2, \ldots$ until ROW[COL[j]+k]=i
- VAL[COL[j]+k]=VAL[COL[j]+k]+x

This involves a short search along a column looking for the relevant row number—another result of using a storage format which is efficient in memory terms but not in time.

Other routines needed for handling sparse matrix formats include a matrix-vector multiply algorithm and possibly an incomplete factorization routine if that is the preconditioner being used. Details of how these may be implemented can be found in Barret *et al.* [11].

3.4 Summary

In this chapter, the problem of solving systems of nonlinear equations has been addressed. Newton's method (§3.2) has been described, along with two techniques for improving its performance (global convergence and inexact solution of each linear problem). The software package NKSOL gives a practical implementation of Newton's method and is used in this work.

The solution of the linear system obtained at each Newton step has also been considered (§3.3), and of the iterative and direct solvers discussed, GMRES was chosen as the solver to use in this work. This has been implemented in SLAP, along with an ILU preconditioner.

This combination of NKSOL and SLAP provides a satisfactory means of solving the set of equations arising from the finite element discretization of the Navier-Stokes equations and is used to obtain the results shown in subsequent chapters.

Chapter 4

Results on Fixed Meshes

4.1 Introduction

The previous two chapters addressed the issues of spatial discretization and the solution of nonlinear equations respectively. We are now in a position to solve the steady compressible Navier-Stokes equations and this chapter presents some results for several examples of transonic and supersonic flow. There are two reasons for doing this; firstly to demonstrate why certain methods or algorithms have been used in place of others (for example the use of local rather than global time-stepping, or need for a stable finite element scheme) and secondly to compare with well known standard test cases for which many previous results exist in the literature.

It is important to emphasise that this chapter only contains results on fixed meshes—the idea of adapting the mesh as solution proceeds is introduced and discussed in chapters 5 and 6. We begin in §4.2 with a short description of the type of mesh being used. The solution algorithm is briefly discussed in §4.3, and techniques for evaluating flow properties are outlined in §4.4. Before considering the Navier-Stokes equations fully we first present some results for a simpler system, consisting of Burgers' equations, in §4.5. The exact solution is known in this case, enabling the accuracy of solutions to be checked. In §4.6 we show the effect of using different algorithms and parameters within the code, using known test cases, so that results can be compared with existing results.

4.2 Unstructured Meshes

In chapter 1, we briefly discussed the generation of suitable grids on which to carry out the spatial discretization (which is the finite element method in this case). Most of the examples presented here concern flow around a NACA0012 aerofoil and typical meshes for such a domain are given in figures 4.1 (a structured mesh of quadrilaterals) and 4.2 (an unstructured mesh of triangles).

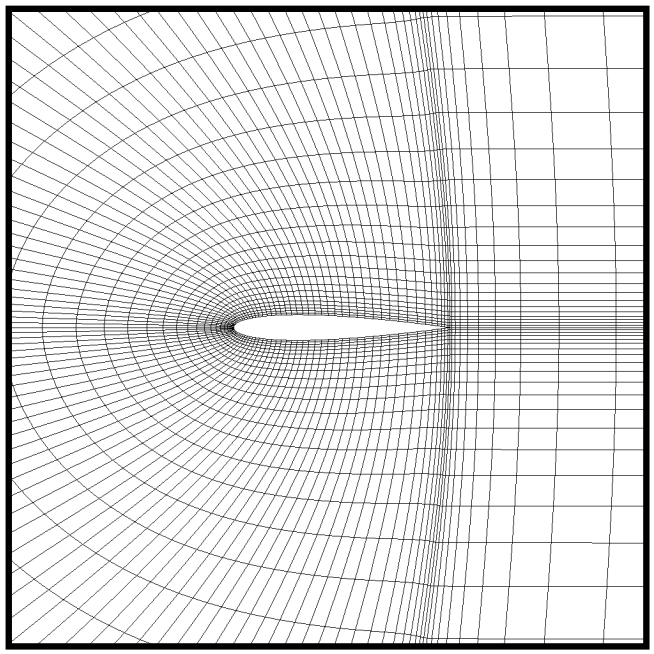

Figure 4.1: A structured mesh around a NACA0012 Aerofoil (4096 elements).

In all of the following results, we use unstructured meshes such as figure 4.2. All the meshes used have been generated at the British Aerospace Sowerby Research centre, with the BAe Release 2.2 FLITE surface mesh generator from Swansea University [26].

4.3 Algorithm for Solution

Here we summarize how the methods seen in the previous chapters are combined so that a converged steady-state solution to the Navier-Stokes equations may be reached. The method of solution may be briefly stated as follows.

1. Initialization. This includes reading in data files containing details of the mesh and setting up the structure of the Jacobian matrix (which is dependent upon

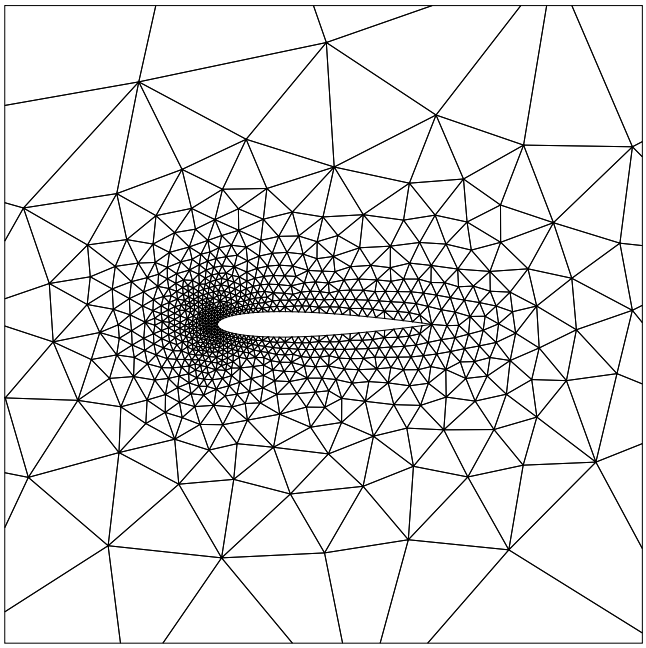

Figure 4.2: An unstructured mesh around a NACA0012 Aerofoil (1617 elements).

the mesh). The initial conditions are also specified, which are usually set to be equal to the freestream values, and as a consequence there is initially a very high gradient within the elements adjacent to the wall boundary.

2. Nonlinear problem at each time-step. With either local or global timestepping implemented, use of a suitable finite element method (such as Galerkin least-squares) applied to a particular variable formulation leads to a system of nonlinear equations, solved using Newton iteration (which has been implemented in NKSOL, see §3.2.4). A maximum of six Newton iterations are performed per time-step, and a linear solver used to solve the linear system at each iteration. The iterative solver used is the preconditioned GMRES algorithm (see §3.3.2).

3. Stopping test. If the L_2 norm of the current solution residual of the steady state problem is small enough ($< 10^{-8}$) then steady state has been reached.

Unless otherwise stated, results in this chapter have been obtained using the following parameters:

- The primitive variable formulation given by (2.8)–(2.10), along with associated boundary conditions.

- The finite element method used is the modified form of Galerkin least-squares discussed in §2.5.1, with the parameter $\tau = hI/2$ (where h is the mesh size parameter, defined here as length of the longest element edge) and where a shock capturing term has been omitted.

- The local timestepping procedure, described in §2.6.1, is used with an algorithmic Courant number of 50.

- The stopping tolerance for the nonlinear solver is 10^{-5}.

- The Jacobian matrix is evaluated using approach (iii) described in §3.2.5.

- The preconditioning used with the GMRES solver is incomplete LU factorization with no fill-in. The maximum dimension of the Krylov subspace is 25, and no restarts are performed (so that a maximum of 25 linear iterations are performed per Newton step). The linear system is not solved exactly (see §3.2.3), and the stopping tolerance at the ith Newton iteration is set to be

$$\frac{||\mathbf{G}(\mathbf{u}_i)||_2}{2^i}, \tag{4.1}$$

  where $\mathbf{G}(\mathbf{u})$ is the residual of the nonlinear problem at the current time-step and $\mathbf{u}_i$ is the solution to this problem at the ith Newton step.

- The numerical integration over each element is carried out using a three point Gaussian quadrature rule, given in [32].

4.4 Flow Evaluation

In order to evaluate the quality of a flow solution, a number of comparisons and checks may be made against previous numerical and experimental results. In addition to the visualization of the solution variables, numerical parameters may exist and can be compared.

In the case of flow around aerofoils, there are many ways of comparing results. When viewing the solution over all or part of the domain, there are several variables which may be compared, including velocity vectors, Mach number, density, temperature and pressure. In order to visualize these values, we use the package "Viz", version 2.11 [112], developed for internal use by the Sowerby Research Centre at

British Aerospace, which has been specifically designed for viewing aerodynamics simulations. This program allows visualization of solutions on both structured and unstructured meshes, for two (and three) dimensional problems. A numerical solution may be input in a simple format, and the choice of output includes contours and shaded plots for any flow variable, velocity vectors and the mesh itself.

Of particular interest in flow around aerofoils is the evaluation of forces exerted on the aerofoil surface. These forces originate from two sources—pressure which acts normally to the surface, and stress, which acts tangentially—and are usually plotted as non-dimensionalized quantities c_p (coefficient of pressure) and c_f (coefficient of friction) against the distance along the aerofoil chord. We define

$$c_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty u_\infty^2}, \quad c_f = \frac{\tau_w}{\frac{1}{2}\rho_\infty u_\infty^2}, \tag{4.2}$$

where p_∞, ρ_∞ and u_∞ are the respective values of pressure, density and speed in the freestream, and p and τ_w are the wall pressure and shear stress. The value of τ_w is calculated on each element adjacent to the aerofoil wall as

$$\tau_w = -\frac{1}{Re}\frac{\partial V}{\partial n} \tag{4.3}$$

where V is the velocity tangential to the aerofoil surface, and n is the outward normal to the surface. From the values c_f and c_p calculated on each element adjoining the aerofoil, the coefficients of lift and drag, C_l and C_d, may be derived:

$$N = \sum_{n_c} c_p s \cos\theta + \sum_{n_c} c_f s \sin\theta \tag{4.4}$$

$$A = -\sum_{n_c} c_p s \sin\theta + \sum_{n_c} c_f s \cos\theta \tag{4.5}$$

$$C_l = N\cos\alpha - A\sin\alpha \tag{4.6}$$

$$C_d = N\sin\alpha - A\cos\alpha \tag{4.7}$$

where n_c is the number of elements adjacent to the aerofoil, α is the angle of attack, s is the length of each aerofoil segment and θ is the angle of the segment with the horizontal (see figure 4.3).

The test cases which are considered in this and subsequent chapters mainly originate from a GAMM workshop set up to consider the numerical solution of the compressible Navier-Stokes equations [18]. These cases concern steady transonic flow around a NACA0012 aerofoil at low to moderate Reynolds numbers. Table 4.1 shows the list of cases for which we present results in this chapter. An additional

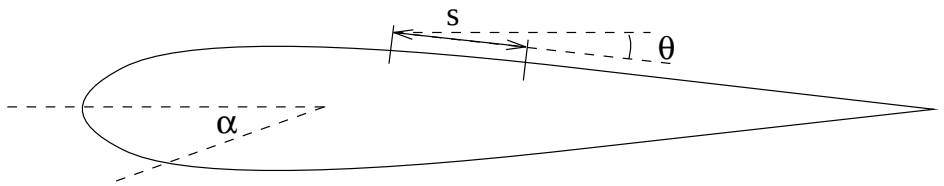

Figure 4.3: Values used in calculation of C_d and C_l.

example that we look at is the problem of supersonic flow over a flat plate [25], for which many previous results have been obtained. All timings given refer to the total CPU time in seconds for the code to converge on a single Silicon Graphics MIPS R4400 processor.

Case	Re	M	α
A1	73	0.8	10°
A2	500	0.8	10°
A3	106	2.0	10°
A6	2000	0.85	0°

Table 4.1: Test cases considered from [18].

4.5 A System of Burgers' Equations

Before giving results for the full system of Navier-Stokes equations, in this section some results showing the numerical solution of a simpler system are given. This system of Burgers' equations consists of two coupled nonlinear convection-diffusion equations, and hence contains some of the features of the Navier-Stokes equations, but also has a known solution, so we can reliably measure the accuracy of a numerical scheme. Consider the following system of equations:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - \epsilon(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) = f \tag{4.8}$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} - \epsilon(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) = g, \tag{4.9}$$

where ϵ is a small constant, and the source terms f and g are given by

$$f = -g = \frac{\exp((-4x+4y)/(32\epsilon))}{128\epsilon(1+\exp((-4x+4y)/(32\epsilon)))^2} \tag{4.10}$$

This system has the exact solution

$$u = \frac{3}{4} - \frac{1}{4}\frac{1}{1+\exp((-4x+4y)/(32\epsilon))} \tag{4.11}$$

$$v = \frac{3}{4} + \frac{1}{4}\frac{1}{1+\exp((-4x+4y)/(32\epsilon))} \tag{4.12}$$

which represents a wave front at $y = x$.

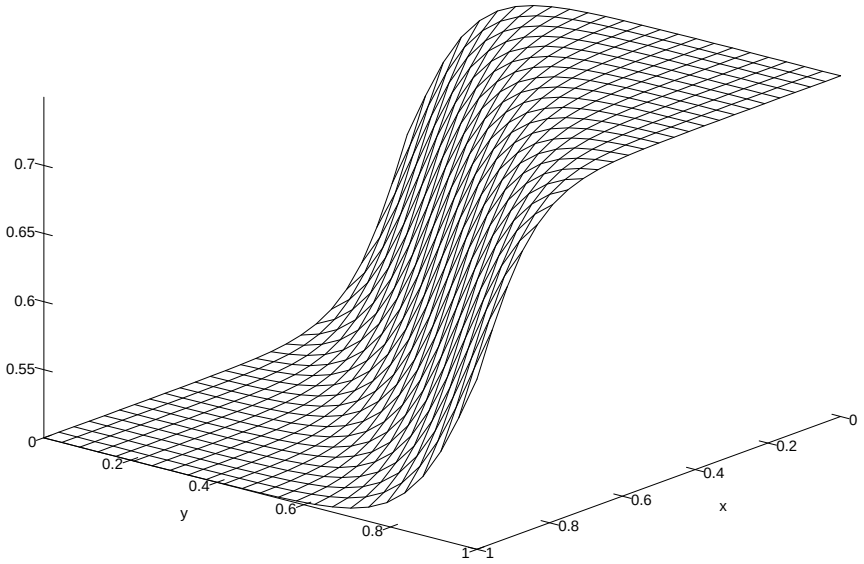

Figure 4.4: u values of exact solution of Burgers' System ($\epsilon = 0.01$).

This problem is to be solved on a square domain $[0,1] \times [0,1]$, with the exact solution being prescribed as Dirichlet conditions along the boundary. The smaller the value of ϵ, the sharper the front along $y = x$, and the more difficult the problem becomes to solve using standard Galerkin finite elements. The first component of the exact solution with $\epsilon = 0.01$ is shown in figure 4.4, and we solve the problem for values of ϵ ranging from 0.1 to 0.001 on a series of unstructured meshes, using both standard Galerkin methods and the stabilized finite element methods described in chapter 2. The three meshes consist of 146, 568 and 2310 elements respectively, and the third of these is shown in figure 4.5.

Given the numerical solution, we define the error norm E as

$$E = \frac{1}{n}\sum_{i=1}^{n}\sqrt{(u_i - u_i^e)^2 + (v_i - v_i^e)^2} \tag{4.13}$$

where n is the number of nodes in the mesh, (u_i, v_i) is the exact solution at node i and (u_i^e, v_i^e) is the numerical solution at node i. Table 4.2 lists the error norms obtained using both the standard Galerkin method (§2.3) and the Galerkin least-squares modification (§2.5) for nine problems (ϵ= 0.1, 0.01 and 0.001 solved on three meshes).

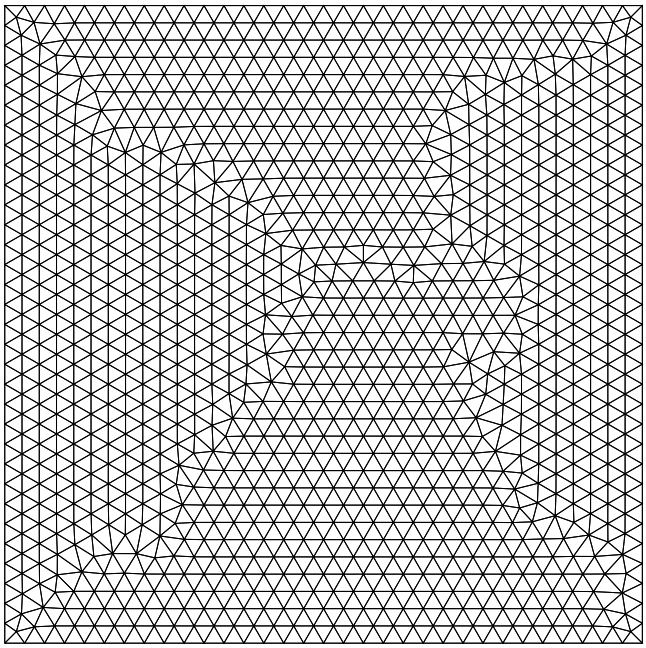

Figure 4.5: Mesh 3 (2310 elts).

For the problem when $\epsilon = 0.001$ on mesh 3, figures 4.6(a) and 4.6(b) show the contours of the first component of the numerical solution obtained using Galerkin only and Galerkin least-squares respectively. These clearly show that spurious oscillations, introduced by the Galerkin method for convection-dominated problems, can be eliminated to a large extent by the introduction of the extra stabilizing term into the variational form.

Mesh	ϵ	Gal.	GLS
1	0.1	8.00e-6	2.01e-5
2	0.1	2.33e-6	1.39e-5
3	0.1	6.83e-7	8.16e-6
1	0.01	2.47e-3	1.41e-3
2	0.01	5.47e-4	2.92e-4
3	0.01	1.33e-4	9.10e-5
1	0.001	1.55e-1	1.18e-2
2	0.001	3.64e-2	8.27e-3
3	0.001	7.91e-3	2.98e-3

Table 4.2: Comparison of errors using Galerkin and GLS.

It can be seen from table 4.2 that for diffusive problems, when ϵ is large in relation to the mesh size, the Galerkin method is more accurate than the Galerkin least-squares technique used here (which indicates that a more sophisticated choice

of the parameter τ should depend on the amount of diffusion already present). However as ϵ is decreased, and the non-physical oscillations introduced by the Galerkin method begin to dominate the solution, the Galerkin least-squares method is more stable (and hence there is less error in the solution). The solutions obtained on the meshes used here are not as accurate as we would like, which demonstrates the need for adaptivity, considered in later chapters. We mention here that stability on coarse meshes provides a basis for a mesh refinement strategy, since oscillatory solutions fail to provide accurate information about where to refine.

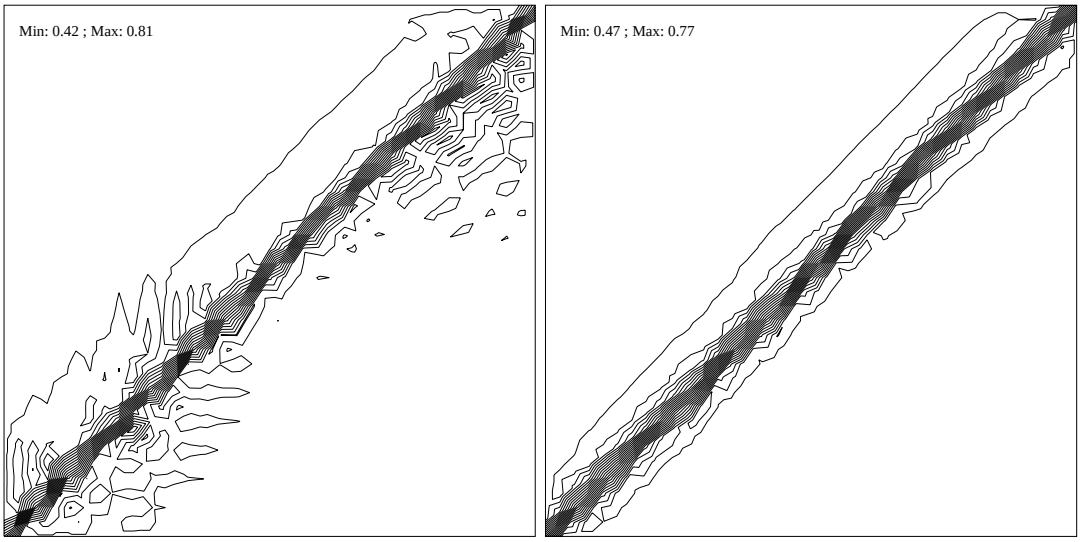

Figure 4.6: Case $\epsilon = 0.001$ on mesh 3. u component of solution using (a) Galerkin and (b) GLS.

4.6 Comparison of Test Cases

Results given above for the system of Burgers' equations indicate that improved solutions may be obtained using a stabilized finite element method such as Galerkin least-squares. We now return to the steady compressible Navier-Stokes equations in this section and show results on fixed meshes for the test cases given in table 4.1, as well as flow over a flat plate [25]. The effect of certain parameters and methods on the accuracy and efficiency of the solution algorithm outlined in §4.3 is also considered. Two meshes are used in the examples of flow around aerofoils—mesh 1 consists of just 1617 elements, a subsection of which is shown in figure 4.2, and mesh 2, consisting of 5436 elements, in which the mesh spacing near the aerofoil is

approximately half that of mesh 1 (see figure 4.7).

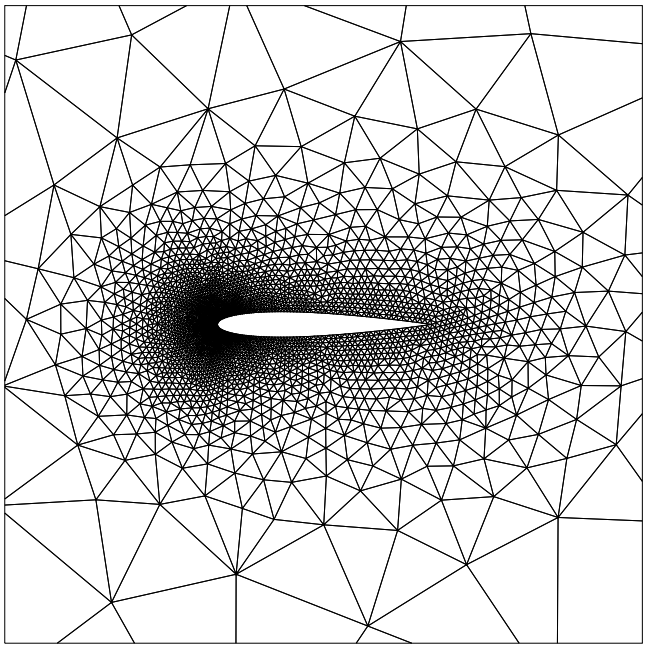

Figure 4.7: Mesh 2 (5436 elements).

4.6.1 Case A1

This problem consists of transonic flow at a fairly low Reynolds number, Re=73, and we first attempt to obtain a solution using the standard Galerkin method only, as described in §2.3.1. As expected, this leads to spurious oscillations being produced in the solution, and these can be seen in figure 4.8(a), which shows a contour plot of the density variable from the converged solution. The mesh used here is mesh 1, consisting of 1617 elements, and even at this Reynolds number, the entire solution is corrupted by the presence of these non-physical oscillations.

In §2.4, we discussed how higher order approximations for the velocity variables may be employed to avoid the oscillations, and figure 4.8(b) shows the improvement in the quality of the solution when bubble functions are used in addition to piecewise linear approximations. We also consider the solution obtained from using Galerkin least-squares (see §2.5).

Figures 4.9 and 4.10 show plots of the pressure and friction coefficients for the solutions obtained using bubble functions and Galerkin least-squares on the finer mesh 2. In comparison to the results obtained in the GAMM workshop [18], these appear to be within the range of values given. The lift and drag coefficients are

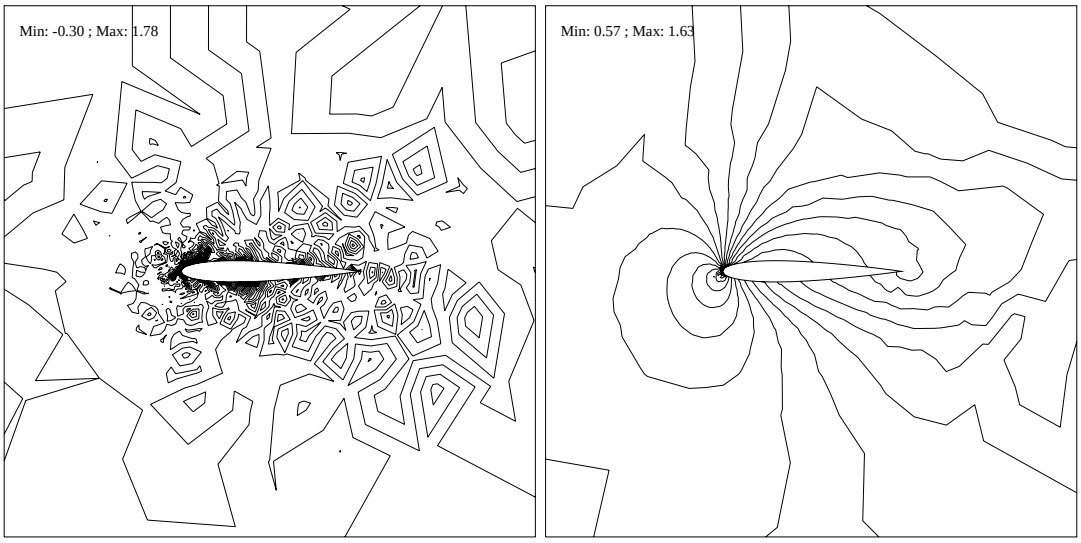

Figure 4.8: Test case A1: Density contours around aerofoil using Galerkin method (a) without and (b) with bubble functions on mesh 1.

shown in table 4.3, which also gives results for the other test cases considered below. It should be noted that in all the examples both the size of the Krylov dimension used in GMRES and the algorithmic Courant number are fixed (at 25 and 50 respectively), and hence the CPU times quoted could be improved upon in each case by tuning these two parameters (e.g. increasing the Krylov dimension would accelerate convergence at the cost of additional memory required).

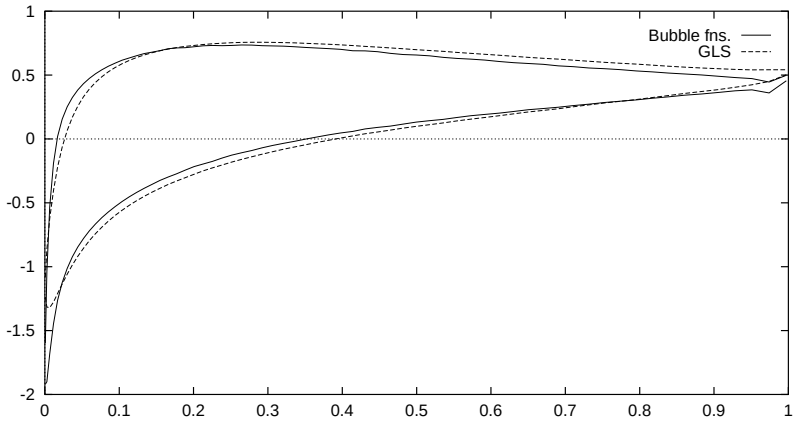

Figure 4.9: Pressure coefficients for case A1 using bubble functions and GLS on mesh 2.

We now consider the effect on these methods of increasing the Reynolds number by looking at test case A2.

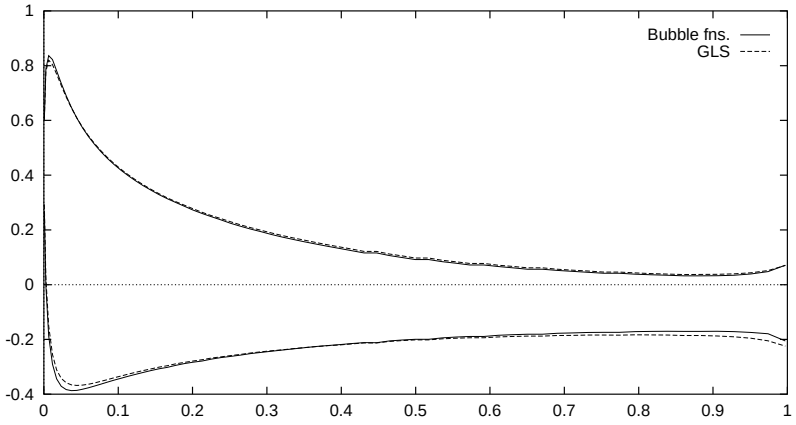

Figure 4.10: Friction coefficients for case A1 using bubble functions and GLS on mesh 2.

Case	Method	C_l	C_d	CPU time (s)
A1	Bubble functions	0.51	0.61	6802
	GLS	0.55	0.62	775
	Range of values in [18]	0.52-0.66	0.58-0.70	
A2	GLS (Prim. vars.)	0.52	0.28	780
	GLS (Cons. vars.)	0.54	0.29	917
	GLS (New form of τ)	0.52	0.29	3810
	Range in [18]	0.41-0.52	0.24-0.29	
A3	GLS (Prim. vars.)	0.32	0.46	1190
	Range in [18]	0.31-0.40	0.41-0.49	
A6	GLS (Prim. vars.)	0.003	0.12	685
	Range in [18]	0	0.10-0.14	

Table 4.3: Results for case A1, A2, A3, A6 on mesh 2.

4.6.2 Case A2

The only change from the previous case A1 is that the Reynolds number is increased to 500, and we again use the approach involving piecewise linear approximations with higher order approximations (bubble functions) for the velocity unknowns. The density contours from the solution (obtained on mesh 1) are plotted in figure 4.11(a), and it can be seen that the use of bubble functions is insufficient, as oscillations begin to reappear. This suggests that an improved method for generating solutions free of unnecessary oscillations is required, such as the schemes discussed in §2.5. If the modified Galerkin least-squares method is used to solve case A2, then an improved solution is obtained, as shown in figure 4.11(b).

The use of an alternative variable formulation to the primitive variables of den-

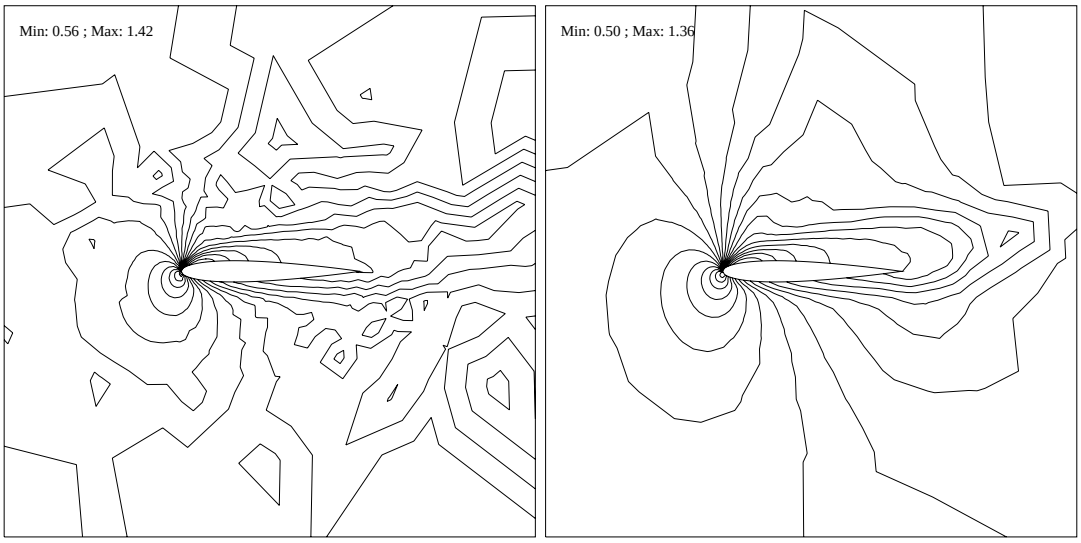

Figure 4.11: Test case A2: Density contours around aerofoil using (a) Galerkin with bubble functions and (b) Galerkin least-squares on mesh 1.

sity, velocity and temperature leads to similar results. Figures 4.12 and 4.13 show the pressure and friction coefficients for both the primitive and the conservative variables given in §2.2.2. Mesh 2, consisting of 5436 elements, is used in this case, and the lift and drag coefficients are given in table 4.3.

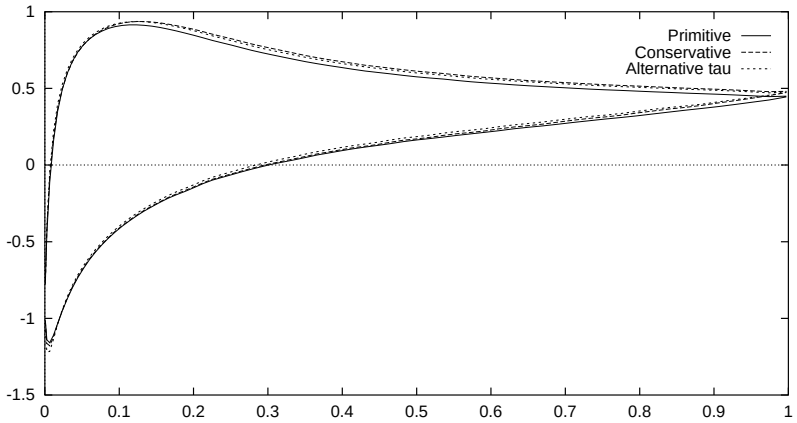

Figure 4.12: Pressure coefficients around the aerofoil for primitive and conservative formulations (case A2).

The form of the matrix τ used in the extra terms of the modified Galerkin least-squares formulation is $hI/2$ where h is the mesh size parameter and I is the identity matrix. More sophisticated forms of this matrix are implemented by both Shakib

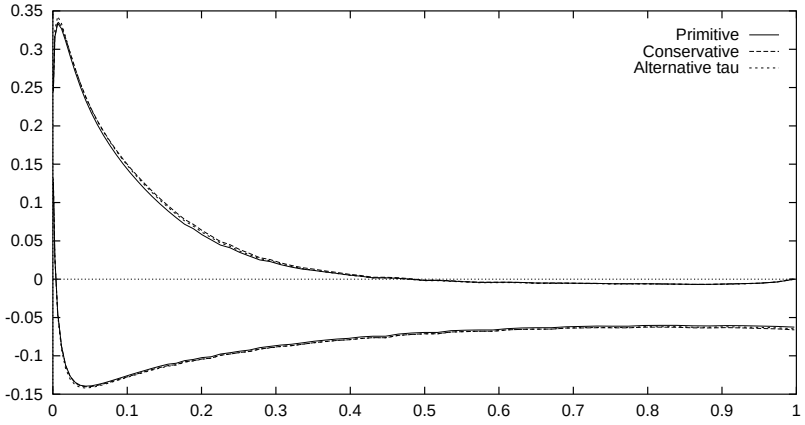

Figure 4.13: Friction coefficients around the aerofoil for primitive and conservative formulations (case A2).

et al. [108] and Hansbo and Johnson [57]. For example, in [57],

$$\tau = \frac{h}{2}(I + \sum_{i=1}^{2} A_i^2)^{-\frac{1}{2}} \tag{4.14}$$

where the A_i's are the convection matrices in (2.36). In the case of primitive variables, the convection matrices are not positive definite and so we use the conservative formulation to implement (4.14). The pressure and friction coefficients with this alternative form of τ are also shown in figures 4.12 and 4.13, and the coefficients C_l and C_d are given in table 4.3.

For this particular test case, the results above show the need for a stable finite element method, such as Galerkin least-squares, in order to avoid solutions containing spurious oscillations. There does not appear to be much difference between using primitive and conservative variables at these moderate Reynolds numbers, and choosing a diagonal form for τ appears to give satisfactory results. The C_l and C_d values are largely within the range of values obtained at the GAMM workshop, although more accuracy would be expected on finer grids.

We discuss below a number of issues concerning the efficiency of the algorithm and see how convergence is affected by some of the parameters chosen.

Time-stepping

In §2.6, two approaches to marching towards a steady-state solution were considered—global time-stepping where a constant time-step is used throughout the domain, and local time-stepping, so that the time-step is allowed to vary spatially. Figure 4.14

shows a comparison between the two approaches for the test case A2, solved on mesh 1. For global timestepping, the time-step size Δt is set to be 1.0, and for local time-stepping the algorithmic Courant number defined in §2.6.1 is 50.0. The graph shows the 2-norm of the residual vector and the cumulative number of nonlinear iterations, which is directly proportional to the CPU time taken. There is clearly a substantial gain in efficiency when using local time-steps to reach steady solutions. The optimal choice of the algorithmic Courant number (and Δt) largely depends on two factors—the size of the mesh and the Krylov subspace dimension used in the GMRES iterative solver (see below). Although a larger Courant number leads to even faster convergence for the mesh used here, a value of 50.0 appears to be suitable for a wide range of mesh sizes.

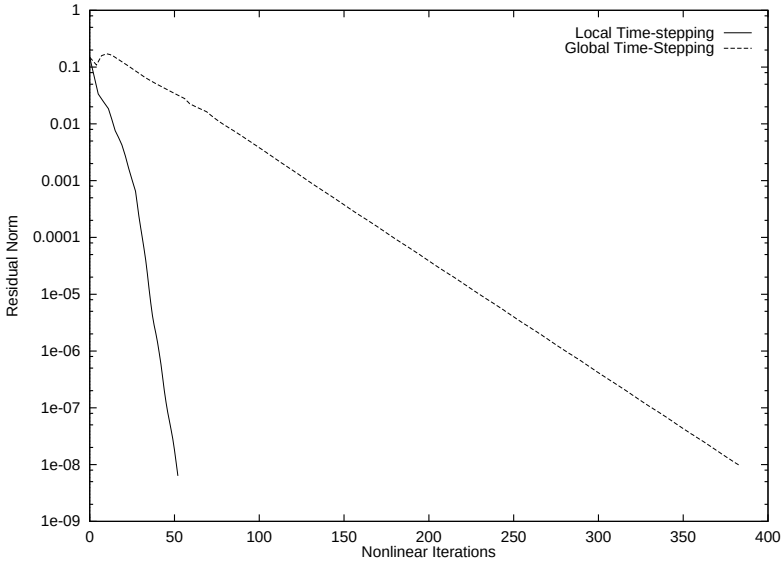

Figure 4.14: Comparison of number of nonlinear iterations taken to converge using local and global time-stepping.

Further acceleration in convergence may be obtained by allowing the Courant number to increase as the time-stepping nears steady-state. We have implemented this by increasing the Courant number by a factor of ten when only one nonlinear iteration is required per time-step. This leads to a large saving in time since the flow solution is changing very slowly by this point and the GMRES solver can cope with the increased time-step size without needing to increase the size of the Krylov dimension.

GMRES

The linear system generated at each Newton step needs to be solved, and the GMRES iterative solver which solves this system was described in §3.3.2. In this section we show the need to incorporate a preconditioner in the solver and discuss how much memory is required by the algorithm. This memory is needed since the update vector computed by GMRES at each iteration lies in a Krylov subspace, and the basis vectors of this subspace need to be stored by the algorithm.

We first consider solving case A2 on mesh 1 without using time-stepping i.e. as one nonlinear problem, using GMRES with and without a preconditioner. Using a value of 200 as the dimension of the Krylov subspace and restarting four times (so that up to 1000 iterations are performed per Newton step), three approaches to solving the linear problem at each nonlinear iteration are tried:

- GMRES with no preconditioning,
- GMRES with Jacobi preconditioning, and
- GMRES with incomplete LU decomposition (with no fill-in) as a preconditioner.

Figure 4.15 shows the residual norm of the nonlinear problem against the total number of linear iterations for these three approaches. Clearly, the use of a preconditioner improves the rate of convergence significantly, and for large meshes is essential, as the Krylov dimension is restricted by the amount of memory (for example a mesh of 4000 nodes and a Krylov dimension of 200 requires approximately 25 megabytes of storage). Only by using an effective preconditioner (such as ILU) can the Krylov subspace be substantially reduced, although some extra storage space is needed for the decomposition process. This also adds to the CPU time, though overall time taken is still considerably reduced: the total time in seconds for convergence (on mesh 1) to steady state is shown in table 4.4.

	CPU time (s)
No precon.	1736
Jacobi	593
ILU(0)	88

Table 4.4: Timings for different preconditioners.

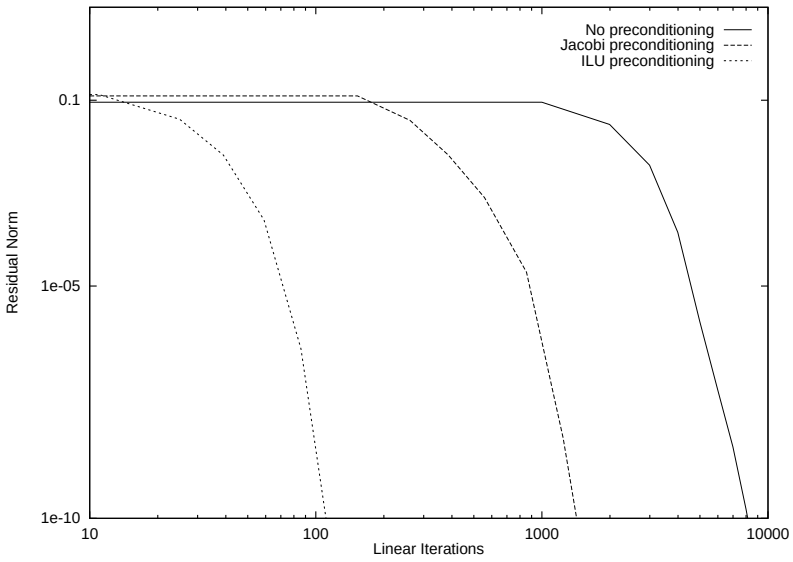

Figure 4.15: Linear iterations to solve a nonlinear problem using GMRES with and without a preconditioner.

Only two types of preconditioning have been implemented here. Further investigations could be carried out on other algorithms such as element by element techniques or a more advanced incomplete LU factorization involving higher levels of fill-in. Re-ordering the unknowns may also lead to convergence acceleration (see Dutto [40] for details).

We now see how the size of Krylov dimension affects the rate of convergence when using GMRES with ILU(0) preconditioning. Table 4.5 shows the CPU times for convergence of case A2 on mesh 2 (with a fixed algorithmic Courant number of 50 used in the local timestepping) for different sizes of the Krylov dimension. Also shown is the effect of allowing two restarts within the GMRES algorithm.

Kry. Dim.	RestartΓ	Nonlinear itrns	Linear itrns	CPU time (s)
5	n	89	444	1424
5	y	57	736	1053
10	n	59	574	1045
10	y	54	813	1041
25	n	55	806	1056
25	y	54	899	1089
50	n	54	855	1086
50	y	54	879	1089

Table 4.5: Convergence comparison using different sizes of Krylov dimension.

The time taken to reach a converged solution is only significantly affected when the Krylov dimension is as small as 5, and no restarts are carried out. Above this, the size of Krylov dimension makes very little difference to the rate of convergence, for this mesh, consisting of 5436 elements. For larger meshes, a larger maximum dimension may be required, although the use of restarting the GMRES algorithm may offer some improvement in convergence.

The two parameters of Krylov dimension size and algorithmic Courant number both depend the size of the mesh, and on each other, so that as the Krylov dimension size is reduced, the Courant number may need to be decreased. Further study might provide a better indication of how these three values are inter-related, so that the code could assign nearly optimal values to the Courant number at run time in order to converge more quickly.

It should be noted that for many test cases at moderate Reynolds numbers, time-stepping is not required when solving on coarse meshes, and the steady Navier-Stokes equations can be solved directly as one nonlinear problem. As the mesh size increases however, a larger Krylov dimension is required and the gains in solution time are outweighed by the extra memory requirements. Test case A2 can be solved directly on both meshes 1 and 2, with dimension of the Krylov subspace equal to 25.

Evaluation of Jacobian Matrix

The Newton solver used to solve the nonlinear problem arising at each time-step requires knowledge of the Jacobian matrix, and in §3.2.5 several possible approaches for calculating this matrix were outlined. These include

- (i) computing the element Jacobian matrix from the analytical derivatives and assembling,
- (ii) forming the global Jacobian from finite difference approximations using the residual function,
- (iii) forming the element Jacobian matrix from finite difference approximations and then assembling, and
- (iv) not computing the matrix explicitly, but finding the matrix-vector product when needed.

To see how these methods compare, case A2 is solved on mesh 1, without any preconditioning of GMRES (since Jacobi or ILU require access to the matrix and so cannot be used with method (iv)), using methods (i), (iii) and (iv) (method (ii) is substantially slower as each Jacobian evaluation requires the residual to be evaluated as many times as there are unknowns). The CPU times are shown in table 4.6, and indicate that forming the Jacobian explicitly appears to be much faster than the matrix-free method (iv). In addition, there is a small gain in time by using the approximate element Jacobian matrix rather than the exact form, because calculating the element residual twelve times is faster than computing the element Jacobian exactly. This suggests that method (iii) is the best approach to use, since it does not require any derivatives to be computed analytically.

Type of Jacobian evaluation	CPU time (s)
(i)	4684
(iii)	4166
(iv)	13955

Table 4.6: Convergence times using different methods to evaluate the Jacobian.

4.6.3 Case A3

This case involving supersonic flow around an aerofoil contains a detached bow shock. In order to solve this problem, we again cannot use the Galerkin method combined with bubble functions without seeing unwanted oscillations appear again, so use the modified Galerkin least-squares method as for case A2. The parameters used are as stated previously and are based on the results presented in the previous section (concerning the use of local time-stepping, the GMRES algorithm etc.).

Primitive variables are employed in the formulation of the problem, which is solved on mesh 2, consisting of 5436 elements. We show the density contours of the solution near the aerofoil in figure 4.16 and the pressure and friction coefficients in figures 4.17 and 4.18. The coefficients of lift and drag, and the time taken to converge are given in table 4.3.

Although the shock has been clearly resolved near the aerofoil where the mesh is fine, in regions where the mesh is coarser the shock is not well defined at all. This indicates the need for either a finer mesh everywhere, or some means of refining the mesh where it is required (this topic is discussed further in chapter 5).

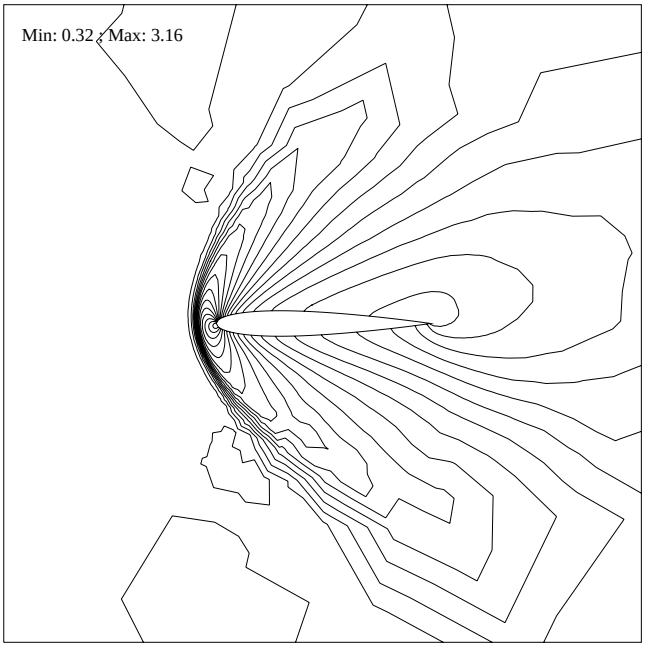

Figure 4.16: Test case A3: Density contours around aerofoil.

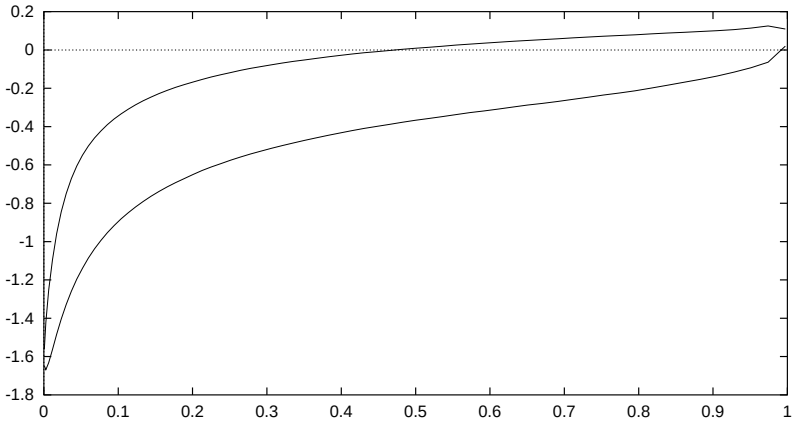

Figure 4.17: Pressure coefficients for case A3 on mesh 2.

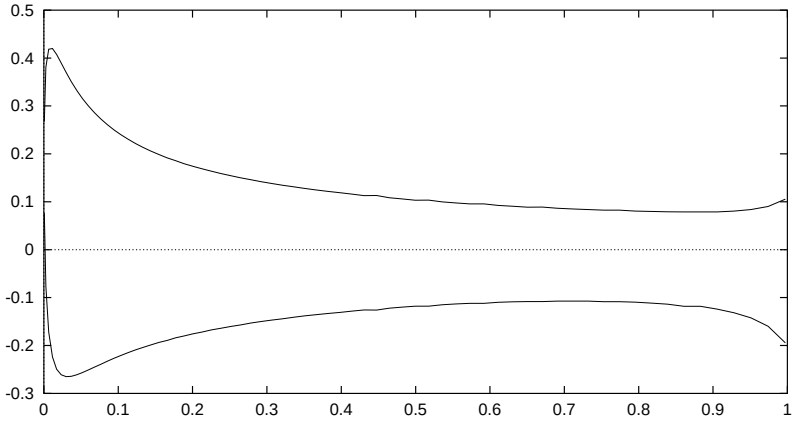

Figure 4.18: Friction coefficients for case A3 on mesh 2.

4.6.4 Case A6

The test case A6 concerns transonic flow at the higher Reynolds number of 2000, with no angle of attack, and as a consequence there is a well-defined wake behind the aerofoil. Again, we use primitive variables, and values of the numerical parameters are as given in §4.3. The solution obtained on mesh 2 is shown in figure 4.19 as a plot of the Mach number contours, and due to the coarseness of the mesh away from the aerofoil, the wake is not resolved at all. We use the Mach number rather than density contours in this test case in order to more clearly show the wake when it appears in the results in later chapters. Figures 4.20 and 4.21 show the pressure and friction coefficients around the aerofoil. Details of the lift and drag coefficients are given in table 4.3.

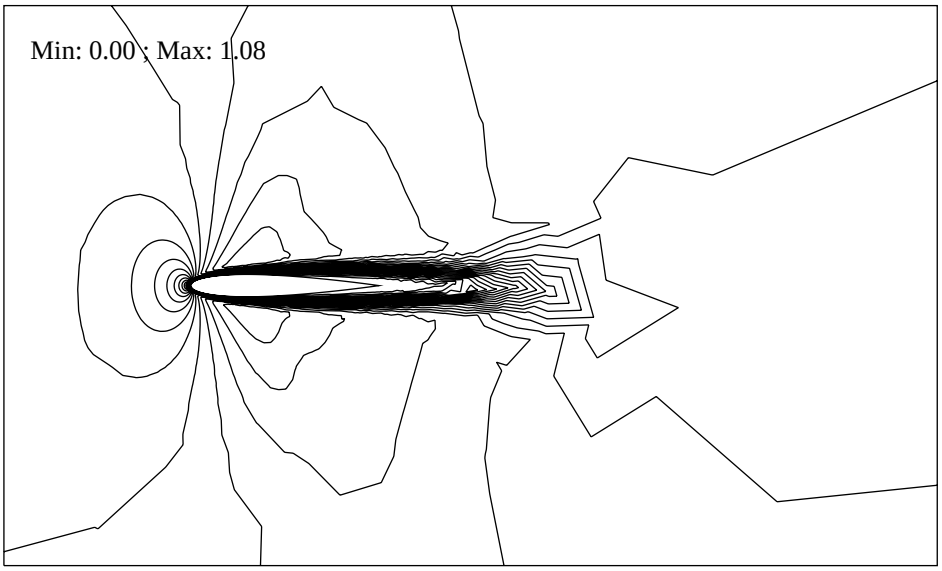

Figure 4.19: Test case A6: Mach number contours around aerofoil.

In order to detect the wake fully, a finer mesh may be used, and we use a triangulated version of the structured mesh shown in figure 4.1, which consists of 8192 elements. Figure 4.22 shows the Mach contours of the solution obtained on this mesh, where the elements are more evenly spread over the entire domain than the unstructured meshes used here. The wake is clearly defined even far from the aerofoil trailing edge. This again demonstrates the need to have a mesh which is fine enough to detect all the features of the flow, either by using a mesh which is fine everywhere or only refining the mesh where necessary.

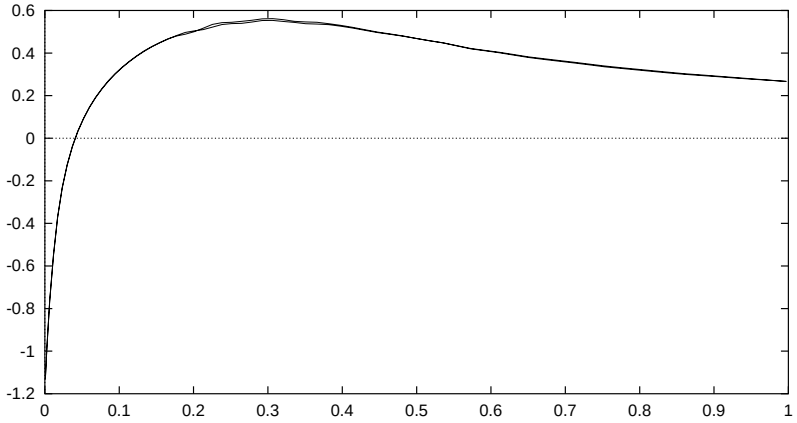

Figure 4.20: Pressure coefficients for case A6 on mesh 2.

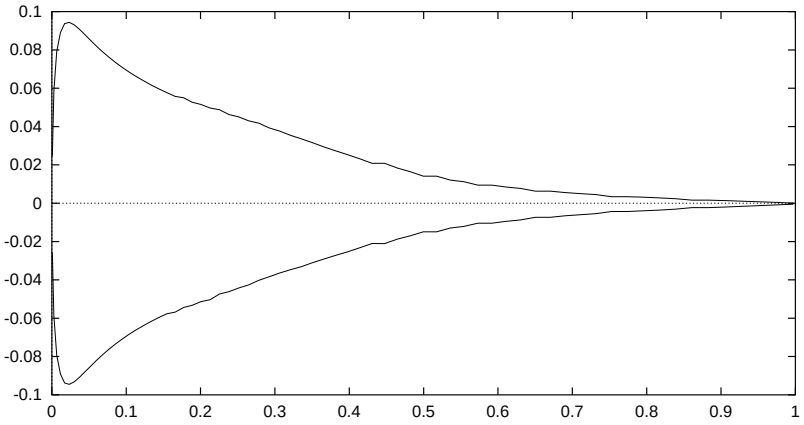

Figure 4.21: Friction coefficients for case A6 on mesh 2.

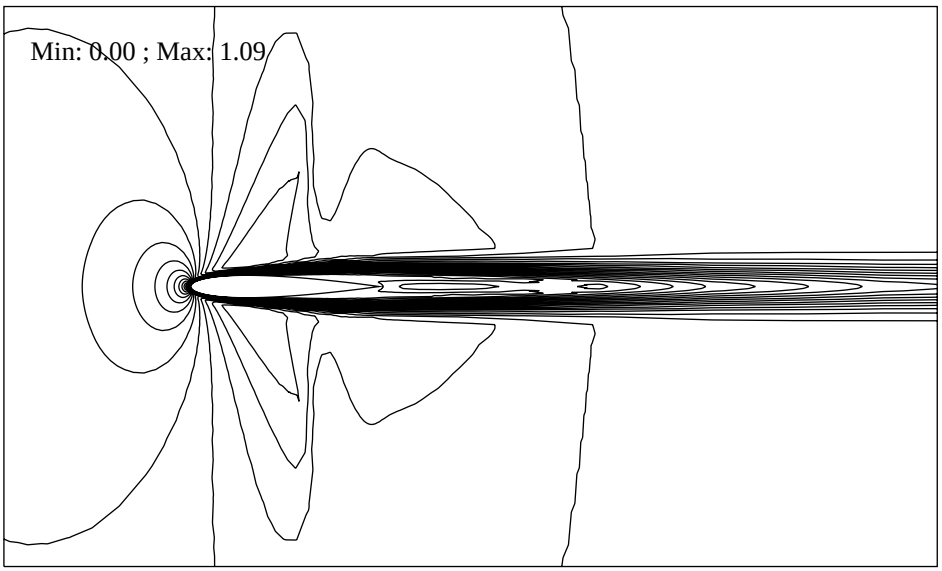

Figure 4.22: Test case A6: Mach contours around aerofoil on finer structured mesh.

4.6.5 Flow Over a Flat Plate

This problem first considered by Carter [25] consists of supersonic flow (a Mach number of 3) passing over an infinitely thin plate causing a shock and a boundary layer to develop from the leading edge of the plate. The computational domain is the region $-0.2 \leq x \leq 1.2$, $0 \leq y \leq 0.8$, and the plate is positioned along $y = 0$, $0 \leq x \leq 1.2$. The Reynolds number used is 1000, and the angle of attack is zero. The following boundary conditions apply. The inflow boundary is defined as the line $x = -0.2$, where $u = 1$, $v = 0$, $\rho = 1$ and $T = T_\infty = (\gamma(\gamma - 1)M_\infty^2)^{-1}$. Along the plate, $u = v = 0$ and $T = T_\infty[1 + (\gamma - 1)M_\infty^2/2]$. Neumann conditions apply to remaining nodes on the boundaries.

We solve the problem on five different meshes, from a coarse mesh consisting of 8×8 nodes (98 elements) to the finest mesh of 25088 elements. These correspond to mesh spacings of $h = 0.2$, $h = 0.1$, $h = 0.05$, $h = 0.025$ and $h = 0.0125$ respectively. One of the intermediate meshes (29×29 nodes) is shown in figure 4.23. The formulation we use is that of primitive variables, and the usual default values for the parameters concerning local time-stepping and the GMRES solver are applied. The CPU time taken to solve the problem on each mesh is given in table 4.7.

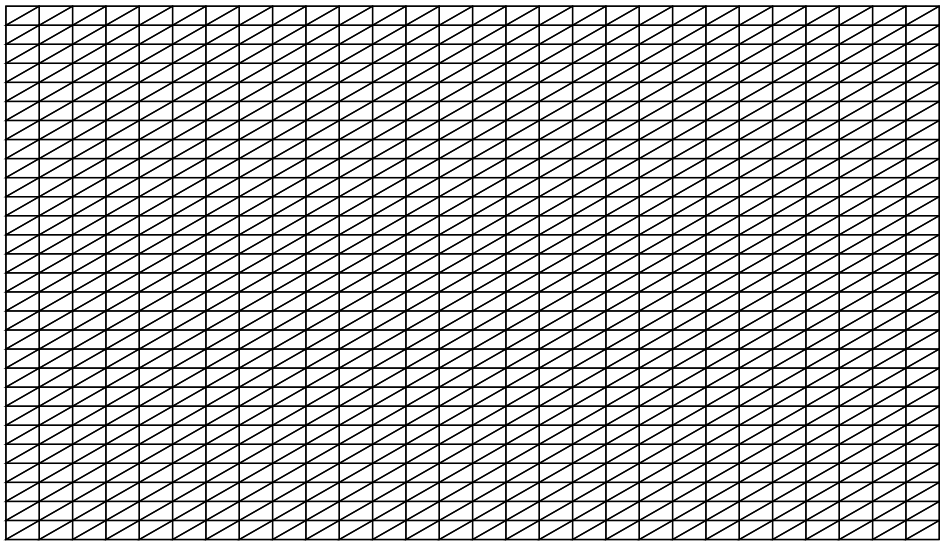

Figure 4.23: Middle mesh ($h = 0.05$) used for flat plate flow problem.

In figures 4.24 and 4.25, the contours of pressure and the Mach number are shown for the solution obtained on the finest mesh, and these demonstrate the presence of both the shock and the boundary layer. We also plot the pressure and

Mesh	1	2	3	4	5
Elts.	98	392	1568	6272	25088
Time (s)	4	15	81	443	2462

Table 4.7: Time taken to solve flat plate flow problem on each mesh

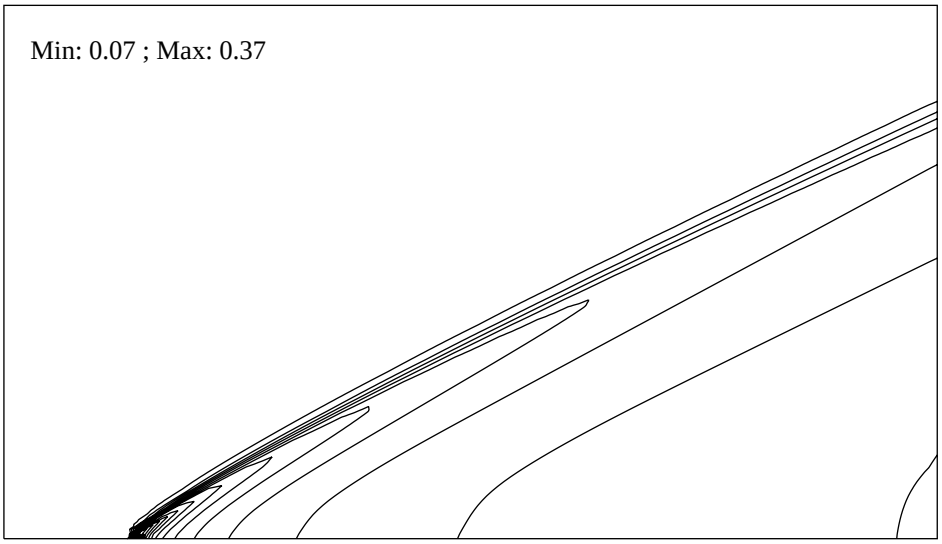

Figure 4.24: Pressure contours of solution on finest mesh ($h = 0.0125$).

friction coefficients in figure 4.26 and 4.27.

This problem has been used many times as a test case in previous work (see Shakib *et al.* [107] and Demkowicz *et al.* [35] for example), and so we may compare solutions obtained here with these other results. On inspection of the contour plots, the location of the shock appears to be correct, and the boundary layer seems to be approximately the right width. However the plots of pressure and skin friction coefficients do not agree very well with other results—values near the leading edge are too low. Possible reasons for these inaccuracies may be the incorrect specification of the boundary conditions, particularly in front of the flat plate, or the simplified form of the Galerkin least-squares parameter τ (although the more sophisticated choice of (4.14) was tried and no improvement observed). In this case we would expect that more accurate solutions may be obtained on finer grids, and in chapters 5 and 6 results on refined meshes do show greater accuracy. The stability of the method even on coarse meshes will allow the mesh to be refined in the correct regions.

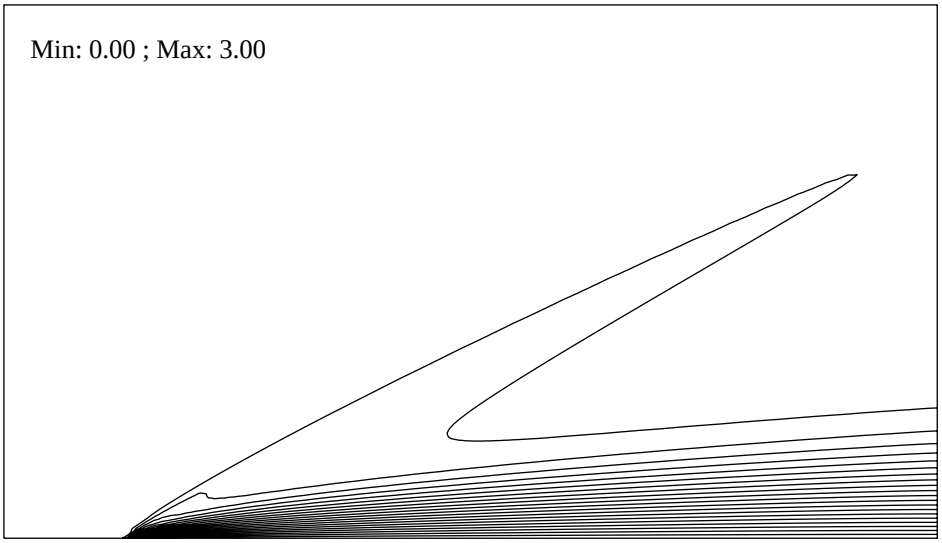

Figure 4.25: Mach contours of solution on finest mesh ($h = 0.0125$).

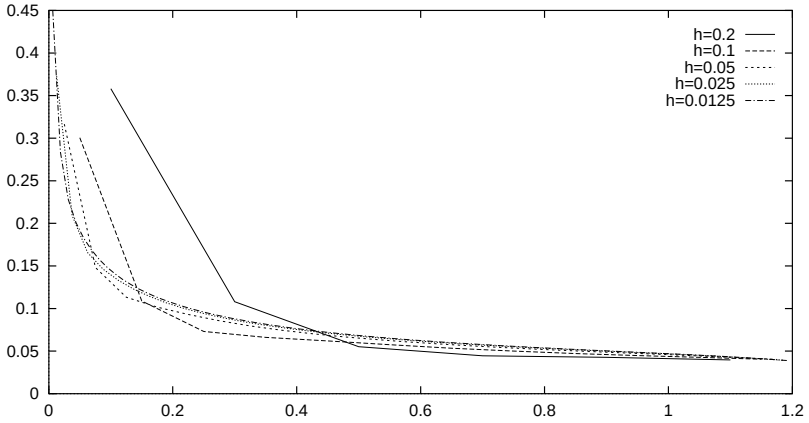

Figure 4.26: Pressure coefficients for flat plate flow problem.

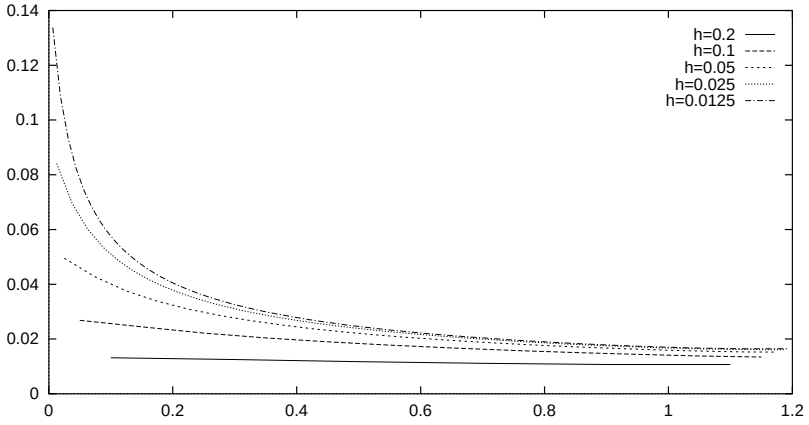

Figure 4.27: Friction coefficients for flat plate flow problem.

4.7 Summary

In this chapter, we have presented a number of solutions to standard test cases for the Navier-Stokes equations. We have also introduced a smaller problem, the coupled system of Burgers' equations, which contains similar features to the full Navier-Stokes equations but has a known exact solution. Results given for this problem show that the use of a stabilized finite element method such as Galerkin least-squares enables solutions which are free of oscillations to be obtained for convection-dominated flows on a coarse mesh. For meshes which are sufficiently fine, the standard Galerkin method is, as expected, stable and accurate.

For the compressible Navier-Stokes equations, the Galerkin method is insufficient, even when higher order functions (such as bubble functions) are used to approximate the velocity variable, see [17]. Our simplified version of the Galerkin least-squares method, where the shock capturing term has been omitted and a simple, inexpensive choice of the parameter τ is used, appears to eliminate the spurious oscillations which appear with standard Galerkin. Results of the four test cases selected from the GAMM workshop on compressible flow [18] appear to be within the range of solutions obtained there, thus verifying the general accuracy of the scheme, even on coarse meshes, for these types of flow. However the solutions obtained for Carter's flat plate flow problem are clearly inaccurate, and it is possible that the method with the current choice of τ is inadequate for this problem, which involves supersonic flow.

The use of local time-stepping offers a significant acceleration in the time to converge to steady-state solutions. The optimal value of the algorithmic Courant number depends largely on the mesh size and the size of the Krylov dimension (used by the GMRES solver), but it is not yet clear how this value might be chosen. Further gains in efficiency are also possible by allowing the Courant number to vary dynamically, so that after the initial few time-steps it could be increased so as to reach steady state more quickly. This is already done in a limited way as the solution approaches convergence.

The preconditioner used in the GMRES linear solver is incomplete LU decomposition with no fill-in, and allows a large improvement in the rate of convergence of GMRES over Jacobi (or no) preconditioning. More sophisticated preconditioners (e.g. ILU with some fill-in) may allow even faster convergence at the cost of extra

memory used to store the decomposition.

The GMRES solver requires a large amount of memory to store the Krylov subspace basis vectors, so it is desirable to reduce the dimension of the subspace as much as possible. Results show that a value of 5 (with restarting) may be sufficient to allow convergence, provided the number of unknowns and/or the algorithmic Courant number are not too large.

The results obtained for the last two test cases (A3 and A6), in which the shock and the wake fail to be completely resolved on the unstructured mesh used, indicate that finer meshes are needed. Two possible approaches are to

- use a very fine mesh everywhere so that no flow features are missed, or
- refine the mesh only in regions where the numerical solution is inaccurate.

This latter approach is discussed in detail in the next two chapters.

Chapter 5

Adaptivity I: h-Refinement

5.1 Introduction

In the previous chapter, some solutions of the steady compressible Navier-Stokes equations were presented. These results were generated on fixed unstructured meshes, so that accurate solutions were only obtained in regions where the mesh was fine enough. For example, in §4.6.4, the wake behind the aerofoil was only resolved when the mesh density behind the aerofoil was high enough.

One possible solution to this problem is to use a single fixed mesh which will be dense enough to accurately reproduce the correct solution. However the exact location of the flow features such as shocks and wakes is not always known, which means the mesh has to be fine everywhere. This is expensive and inefficient as many more unknowns may be solved for than are actually needed.

An alternative idea is to adapt the mesh in some way in order to improve the quality of the solution (or reduce the solution error). Initially the problem might be solved on a coarse mesh, which is subsequently modified according to the first approximate solution. It is particularly important that a stabilized numerical method is used to obtain solutions on coarse meshes, as solutions containing spurious oscillations will lead to refinement of the mesh in regions where it is unnecessary. The solve-adapt iteration may be repeated until a solution of the desired accuracy has been reached. We discuss such an approach in this chapter, and describe methods for obtaining steady solutions on adapted meshes. The use of adaptivity for transient problems involves some additional difficulties, and these are considered further in chapter 7.

There has been a great deal of interest in adaptive finite element methods, both for strongly elliptic problems (e.g. Babuska [6] and Eriksson and Johnson [42]) and compressible flow (e.g. Lohner [89], Oden *et al.* [100] and Hassan *et al.* [60]).

This adaptive approach requires a number of issues to be addressed. Firstly the concept of adaptivity implies that we can measure the error of a given numerical solution, which in general we cannot know exactly. Instead, some means of estimating where the error is large is required—and there exist a wide range of techniques which have been used to do this, from simply using physical properties of the solution to sophisticated *a posteriori* error estimates. Some of these are discussed in §5.2.

Once the error indicator has been used to determine where the solution error is largest, there are several ways of modifying the mesh to reduce this error.

- *Addition of points* (h-refinement). Here the mesh is refined by adding more nodes to the mesh in regions where the error is large, thus reducing the mesh size parameter h for elements in these regions. Conversely removal of nodes where the error is small is also possible.

- *Movement of points* (r-refinement). No new points are added, but existing nodes are redistributed in order to ensure that the mesh density is high where the error is large.

- *Order enrichment* (p-refinement). The mesh remains the same, but the local order of approximation is increased, i.e. in elements where the error is large, higher order polynomial basis functions are used.

It is also possible to combine these methods as well as use them individually. In this chapter we discuss the use of h-refinement and consider a form of r-refinement (and the hr combination) in the following chapter. We do not consider p-refinement here, but see [35] and [36] for examples of hp-refinement in compressible flow problems.

When carrying out h-refinement, it is important to have a suitable data structure to store the details of the mesh, so that addition and removal of points may be done efficiently. This is discussed in §5.3 where an algorithm for spatial mesh refinement is outlined. In §5.4, results of using h-refinement with a number of error indicators are presented for some of the test cases given in chapter 4. These demonstrate the advantages of using adaptive meshes rather than fixed meshes.

5.2 Error Indicators

The two main issues to be considered when taking an adaptive approach, such as h-refinement, are the refinement algorithm used (including a suitable mesh data structure) and how to determine which regions of the mesh are to be refined. The first of these is discussed in §5.3, and in this section we describe some choices of error indicator (which should reflect which regions need to be refined).

The refinement algorithm outlined in the next section works by dividing specified triangles of the mesh into four sub-triangles, and so we require an error indicator which will return a value for each element (rather than each node or edge). Given a tolerance TOL, an element will be refined if

$$\theta_K > TOL \times \theta_K^{\mathrm{MAX}}, \tag{5.1}$$

where θ_K is the value returned by the error indicator for element K and θ_K^{MAX} is the maximum value of θ_K over all the elements in the mesh. This refinement rule is given by Oden *et al.* in [100] and makes the choice of the value TOL less dependent on the exact form of θ_K than the fixed criterion of $\theta_K > TOL$.

There exist many possible choices for this indicator, and one common idea is to use the physical properties of the current numerical solution. Very often these will be changing most rapidly in parts of the mesh where the error will be large such as shocks or boundary layers. For example, in [100], Oden *et al.* use the gradient of the density to define the following:

$$\theta_K^g = \frac{\sqrt{A_K}}{\rho_K} \sup_{i=1,2} \left| \frac{\partial \rho}{\partial x_i} \right|, \tag{5.2}$$

where A_K denotes the element area and ρ_K is the average value of the density on the element. In [17], the local mach number is used in the following way by Bristeau *et al.*

$$\theta_K^m = \frac{\mathbf{u}_K \times \nabla M_K}{|\mathbf{u}_K|}, \tag{5.3}$$

where M_K is the local mach number obtained from the average element velocity $\mathbf{u}_K$ ($M_K = M_\infty |\mathbf{u}_K|$). Another similar indicator is the flow vorticity

$$\theta_K^v = |\nabla \times \mathbf{u}_K|. \tag{5.4}$$

A more sophisticated approach than relying on the change in solution variables is to develop *a posteriori* error estimates, where the error of a numerical solution U

is estimated by some function $\mathcal{E}$ of U and h (the mesh size parameter):

$$||u - U|| \leq \mathcal{E}(U, h). \tag{5.5}$$

Here u is the exact solution and $\mathcal{E}$ will depend on the p.d.e. being solved. Such estimates have been developed for a wide range of p.d.e.'s (e.g. by Babuska [5], Strouboulis and Oden [116] and Johnson *et al.* [42]). For convection-dominated problems where the streamline diffusion method is being used, Eriksson and Johnson [41] prove an *a posteriori* error estimate for a scalar linear convection-diffusion equation, and show how this estimate for the solution error may be used as the basis of an adaptive algorithm which aims to construct a mesh such that $\mathcal{E}(U, h)$ equals a user defined tolerance. Error estimates have also been proved by Hughes *et al.* [68] for linear symmetric advective-diffusive systems.

However the theoretical justification for the extension of results from linear to nonlinear problems is still being investigated (see for example the work of Johnson and co-workers, [80] and [43]). Instead, attempts have been made to generalize estimates for linear problems and numerically demonstrate their effectiveness. In [57] Hansbo and Johnson give results of an adaptive method for the compressible Euler equations based on extending the work done in [41].

The error indicator in [57] is based on $R(\mathbf{U})$, the residual obtained by inserting the finite element solution into the original equation, and we consider here a similar but generalized type of indicator, so that formulations other than conservative variables may be used. Since we are interested in an element-based indicator, we wish to calculate this residual over each element. For the Navier-Stokes equations, we define for each element

$$R(\mathbf{U}) = \mathcal{L}^* \mathbf{U} \tag{5.6}$$

where $\mathcal{L}^*$ is defined in (2.78) as the steady-state compressible Navier-Stokes operator. However since $\mathbf{U}$ is approximated with piecewise linear functions, the second derivatives vanish inside the elements and

$$R(\mathbf{U}) = A_1 \frac{\partial \mathbf{U}}{\partial x_1} + A_2 \frac{\partial \mathbf{U}}{\partial x_2} - \mathcal{F}, \tag{5.7}$$

where the A_i's are the convection matrices and $\mathcal{F}$ is a source term. The error indicator θ_K has the form

$$\theta_K^r = \left(\int_K |R(\mathbf{U})|^2 d\mathbf{x} \right)^{\frac{1}{2}} \tag{5.8}$$

where, if $R(\mathbf{U}) = (r_1, .., r_4)^T$,

$$|R(\mathbf{U})|^2 = \sum_{i=1}^{4} r_i^2. \tag{5.9}$$

In order to include the diffusive effects, which are likely to be important near boundaries for example, we define the following term for the scalar variable u, defined in [41],

$$D_h^2 u|_K = \left(\frac{1}{2} \sum_{\delta \in \partial K} (|[\nabla u]_\delta|/h_\delta)^2 \right)^{\frac{1}{2}} \tag{5.10}$$

where ∂K represents the three element edges, $[.]_\delta$ denotes the jump across edge δ and h_δ is the length of edge δ. This measures the jump across element boundaries in the gradient of u, and may be included within a modified form of (5.8),

$$\theta_K^d = (1-\beta) \left(\int_K |R(\mathbf{U})|^2 d\mathbf{x} \right)^{\frac{1}{2}} + \beta \left(\int_K \frac{1}{Re} \sum_{i=1}^{4} D_h^2 u_i|_K d\mathbf{x} \right)^{\frac{1}{2}}, \tag{5.11}$$

where $\beta > 0$ is a specified parameter. In §5.4 results using several of the element error indicators discussed above are shown.

5.3 A Local Spatial Refinement Algorithm

Having discussed suitable choices of error indicator, we now briefly describe a refinement algorithm which may be used to obtain a converged solution on an adapted mesh. The original version of the code used to carry out the mesh refinement here was written by Jimack and is described in detail in [76]. The algorithm and associated data structures have been designed to handle both refinement and derefinement of the mesh, but for steady problems, we only utilize the refinement routines (in chapter 7, we consider the use of derefinement in unsteady problems). The algorithm has been developed for use on meshes which are unstructured and contain triangular elements.

The approach taken, which is similar to that of Lohner [89], involves refining elements selected by the error indicator by subdividing them into four smaller elements as shown in figure 5.1. In order to keep the mesh conforming, any triangle which then contains a node on any of its edge midpoints is divided into two (see Figure 5.1). If one of these bisected triangles needs to be subsequently refined then the parent triangle is divided into four, to avoid the occurrence of very thin triangles. A tree-like data structure, consisting of parent and child elements is used to

store the layout of the mesh. An initial mesh, which may be very coarse, forms the top level, and the current mesh used for the finite element discretization is given by the leaves of the tree structure. Figure 5.2 shows the tree for the mesh section in figure 5.1.

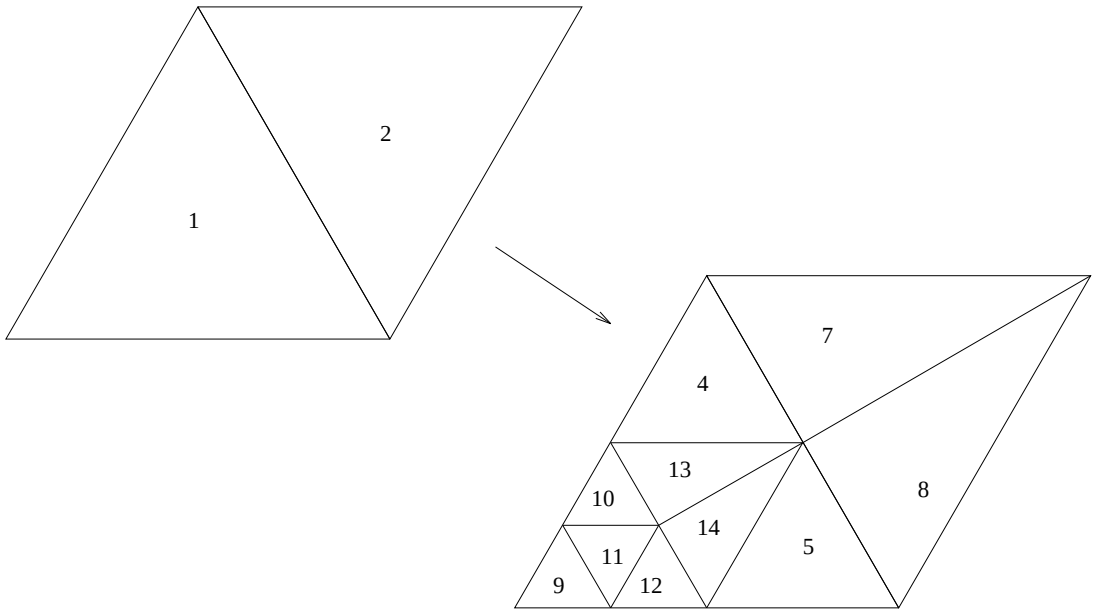

Figure 5.1: Refinement of triangles into two or four sub-triangles.

The new nodes added upon refinement appear at the midpoints of element edges and linear interpolation of the solution values at the two nodes connected by the edge is used to generate solution values at the new point. The positions of additional points along the domain boundary are calculated directly from the polynomial (or other function) used to define the shape of the boundary.

At each refinement step, only one level of refinement is carried out, and no initial coarse element is allowed to be refined more than a predetermined number of times. This is used in conjunction with the error indicator tolerance TOL to specify the desired accuracy of the solution (by limiting the size of the smallest element).

In order to reach a solution on a refined mesh, the above spatial refinement algorithm is combined with the finite element solver which was discussed in previous chapters. The problem is first solved on the initial mesh, and then a refinement step based on the current converged solution is performed. A converged solution is then found on the new mesh, and another refinement step carried out. This solve-refine process is continued until no more refinement is done on the mesh (i.e. when for each element either the error is sufficiently small or the maximum level of

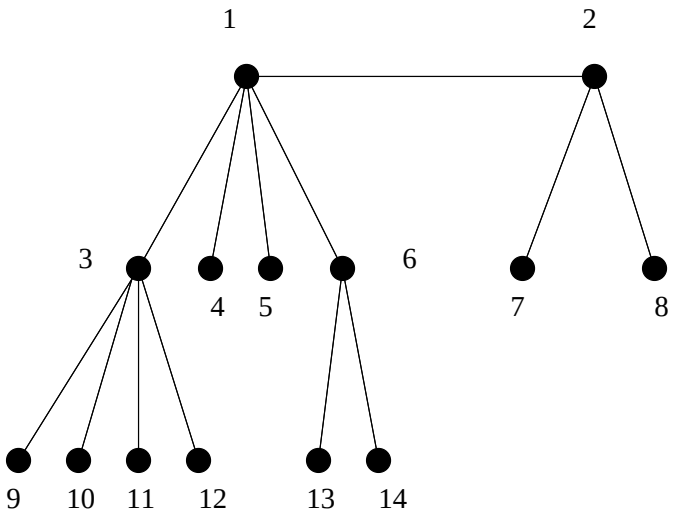

Figure 5.2: Tree structure for triangles shown in figure 5.1.

refinement has been reached).

It is only on the final mesh that a fully converged solution is required and so solving to convergence on all the intermediate meshes is unnecessary. Instead it is possible to only partially solve the problem on each mesh before refining, thus providing a more efficient way of reaching the steady solution. The solution scheme we use here is based on this idea, so that local time-stepping is carried out for a number of steps on a given mesh, followed by a refinement step, after which the time-stepping procedure continues.

The solution procedure is similar to the algorithm given in §4.3 except that a refinement step is performed each time the solution on the current mesh has converged to within a given tolerance (which is larger than the tolerance used to determine convergence on the final mesh). This is measured by the L_2 norm of current solution residual (i.e. the residual of the nonlinear system generated from discretizing the steady Navier-Stokes equations). In practice usually only one time-step is needed to reduce the residual norm to below the required tolerance, and so, after a few initial steps reaching a partially converged solution on the starting mesh, refinement is carried out almost every step.

By starting with a coarse mesh, and applying the above method of solve and refine, we may obtain a converged solution on a mesh which allows all the features of a particular flow to be resolved. In the following section results demonstrating this are given.

5.4 Results

We have so far discussed how spatial refinement may be carried out on a mesh, given an indicator to measure where the error is large. We now show the effect on the numerical solution of refining meshes using similar examples to those introduced in chapter 4.

The system of Burgers' equations given in §4.5, for which an exact solution is known, is first considered, followed by the test cases A2, A3 and A6 (see table 4.1) from the GAMM workshop [18]. We don't consider case A1 here, as it is a similar but less difficult problem than A2. Finally, results from the problem of flow over a flat plate are shown.

The values of the numerical parameters (including the algorithmic Courant number, GMRES Krylov dimension and solver tolerances) in the code are exactly as were stated in §4.3 along with the modified Galerkin least-squares method of §2.5.1 using primitive variables. The following additional parameters are needed in the refinement routines.

- On each mesh, local time-stepping continues until the L_2 norm of the current solution residual is below 10^{-4}.
- The tolerance TOL for the error indicator varies for different problems but is in the range 0.1–0.4, so that an element is refined if $\theta_K > \text{TOL} \times \theta_K^{\text{MAX}}$.
- Similarly, the maximum level of refinement allowed (MAXLEV) for any initial element differs between problems (but is no more than 5).
- For the error indicator θ_K^d defined in (5.11), the value of the parameter β is always 0.5.

5.4.1 A System of Burgers' Equations

This set of two coupled Burgers' equations appear as (4.8)–(4.9) in §4.5, and have a known exact solution so that the accuracy of a numerical solution can be measured precisely. In order to see the effect of mesh refinement, the value of the diffusion constant ϵ is fixed as 0.001 and the problem is solved on several adapted meshes, starting with a coarse mesh consisting of 40 elements (see figure 5.3). Three different

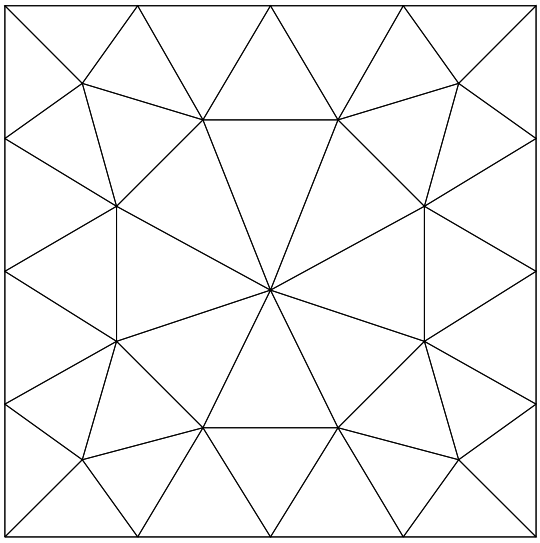

Figure 5.3: Initial coarse mesh (40 elements).

error indicators are used, and the problem is also solved on two fixed meshes of 2310 and 9350 elements for comparison.

- Since the exact solution is known, precise values of the error on each element may be calculated and so we use the indicator

$$\theta_K^e = \sqrt{(u-u_e)^2 + (v-v_e)^2} \tag{5.12}$$

  where u, v are the average values of the numerical solution over the element and u_e, v_e are the exact known values at the centroid of the element.

- The indicator equivalent to (5.8) is also used:

$$\theta_K^r = \left(\int_K (u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - f)^2 + (u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} - g)^2 d\mathbf{x} \right)^{\frac{1}{2}}. \tag{5.13}$$

- Thirdly, we use a simple indicator based on the divergence of the velocity,

$$\theta_K^g = \left| \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right|. \tag{5.14}$$

The error norm used to calculate the error on the final mesh is

$$E = \sum_{K \in \Omega} A_K \sqrt{(u-u_e)^2 + (v-v_e)^2} \tag{5.15}$$

where A_K is the element area, u, v are the average values of the numerical solution over the element and u_e, v_e are the exact values of the solution at the centroid.

Err. Ind.	MAXLEV	Err. Norm	CPU time	Final elts
Fixed mesh 1		4.13e-3	43	2310
θ_K^e	3	3.38e-3	17	948
θ_K^r	3	3.43e-3	20	897
θ_K^g	3	3.43e-3	18	778
Fixed mesh 2		1.03e-3	199	9350
θ_K^e	4	8.56e-4	60	2399
θ_K^r	4	8.62e-4	51	1975
θ_K^g	4	8.77e-4	34	1575

Table 5.1: Results on adapted meshes for Burgers' equations.

For each of the indicators, two values of MAXLEV, the maximum refinement level (3 and 4) are used, so that the finest mesh spacing allowed in each case is comparable to the spacing on the two fixed meshes, and the value of TOL is 0.1.

Table 5.1 gives the results obtained using both fixed and adapted meshes, showing the error E of the final solution, the number of elements in the mesh and the CPU time (on a MIPS R4400 processor) to reach convergence.

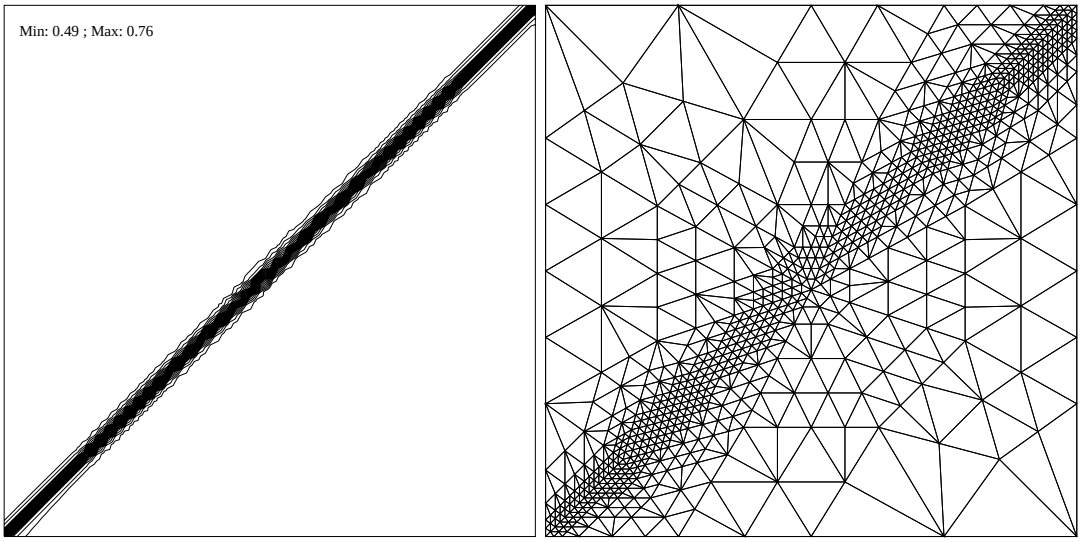

Figure 5.4: Contour plot of u and final mesh (1575 elements) when using θ_K^g and MAXLEV=4.

It may be seen from the table that introducing refinement has a significant effect on the rate of convergence. For all three error indicators tried, a solution of slightly better accuracy to the fixed mesh solution is obtained between two and six times more quickly when starting with a coarse mesh and refining only where necessary. Figure 5.4 shows the final solution and the mesh generated when MAXLEV=4 and

θ_K^g is the error indicator used. For this particular problem, when MAXLEV=3, the three error indicators all perform similarly, but with MAXLEV=4, θ_K^g, based on the gradient of the solution, is the most efficient since it only refines along the front at $y = x$, whereas the other indicators will additionally refine either side of the front.

This example demonstrates quantitatively that adaptive methods can be an efficient way to solve p.d.e.'s. We now return to the Navier-Stokes equations, where precise comparisons of accuracy are less easily available, and show results for the test cases first discussed in §4.6.

5.4.2 A2

In all the cases concerning flow around the NACA0012 aerofoil, we use the mesh shown in figure 5.5 as the initial mesh. It contains only 547 elements and the mesh spacing along the wall boundary is approximately twice that of mesh 1 (1617 elements) used in §4.6.

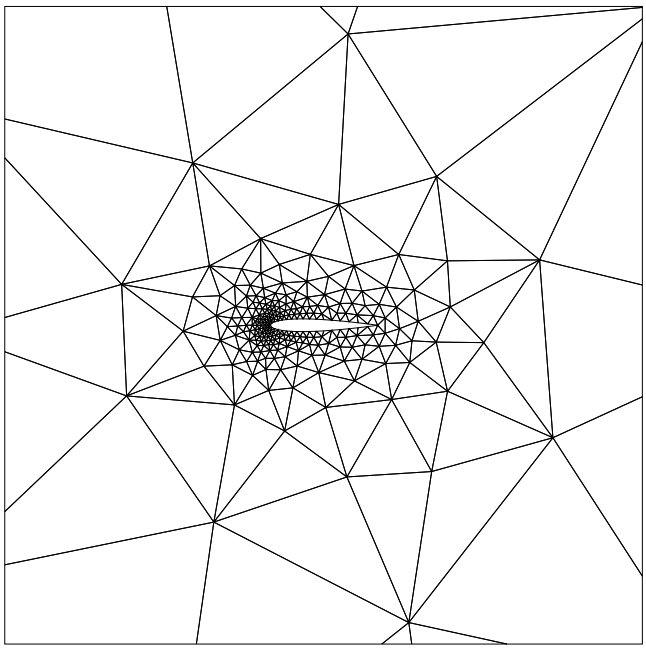

Figure 5.5: Initial coarse mesh around NACA0012 aerofoil (547 elements).

We first use the test case A2 to compare the error indicators described in §5.2. The same parameters (TOL=0.1, MAXLEV=3) are used in all cases (with the exception of the density gradient measure θ_K^g where TOL=0.4), and table 5.2 shows the coefficients C_l and C_d, total time taken and the final mesh size for each indicator. Also shown are the results from the fixed mesh 2 (see §4.6).

Err. Ind.	C_l	C_d	CPU time	Final elts
Fixed	0.52	0.28	780	5436
θ_K^g	0.48	0.27	3426	10274
θ_K^m	0.64	0.27	842	4008
θ_K^v	0.60	0.31	1463	6558
θ_K^r	0.47	0.27	1212	6468
θ_K^d	0.47	0.27	1208	8370
Values in [18]	0.41-0.52	0.24-0.29		

Table 5.2: Results on adapted meshes for test case A2.

Figure 5.6 shows the density contours obtained when using the residual indicator θ_K^r. Plots of the pressure and friction coefficients are shown in figures 5.7 and 5.8, which may be compared with plots for a fixed mesh shown in figures 4.12 and 4.13.

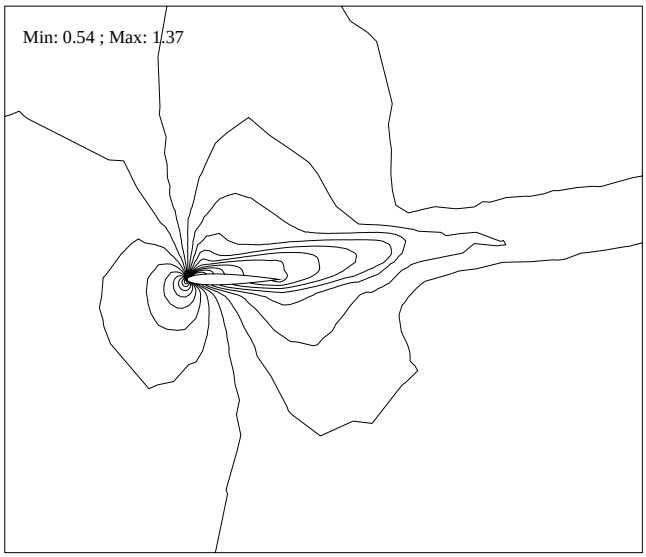

Figure 5.6: Density contours for case A2 using residual indicator.

Figure 5.9 shows a subsection of the final mesh obtained when using the density gradient (θ_K^g) and vorticity (θ_K^v) indicators. It is clear from the C_l values in table 5.2 that when starting from a coarse mesh the indicators θ_K^v and θ_K^m fail to resolve the flow adequately. The density gradient indicator θ_K^g performs better but at a cost of using many more elements than the other indicators (this could be reduced by using an even larger value of TOL). The residual indicator θ_K^r appears to give a solution with a lower (and more accurate, compared with the results in [18]) value of the lift coefficient without using an excessive number of elements. Like θ_K^g, there is some refinement behind the aerofoil so the wake is partially resolved. The addition

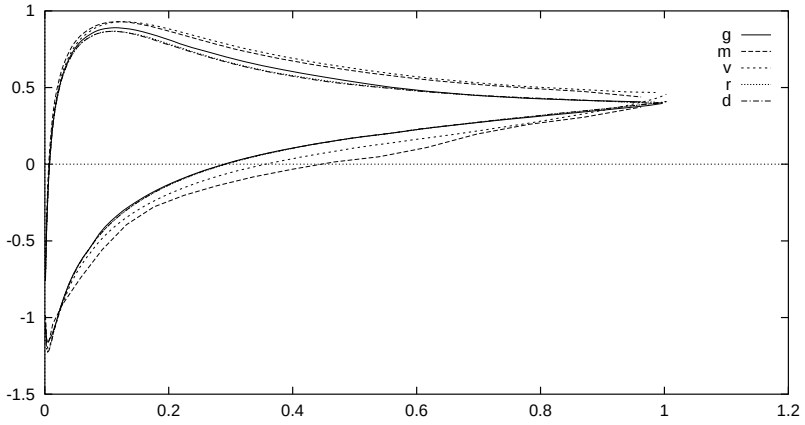

Figure 5.7: Pressure coefficients using different error indicators for case A2.

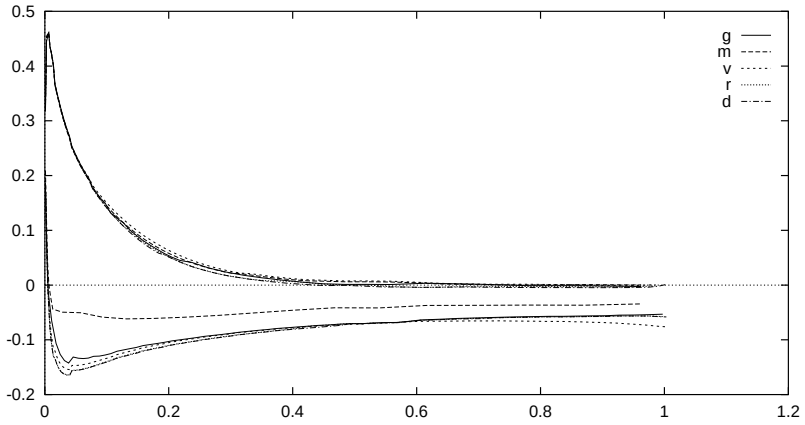

Figure 5.8: Friction coefficients using different error indicators for case A2.

of diffusive terms (θ_K^d) seems to make very little difference for this example, apart from adding extra elements near the leading edge of the aerofoil.

Overall, θ_K^r and θ_K^d are the most satisfactory of the indicators tried, as well as being the only ones with any theoretical basis. The indicator θ_K^g also works well, but at a greater cost.

We now look at the effect of changing the parameter MAXLEV when using the residual indicator. Table 5.3 shows how the values of C_l and C_d become more accurate as elements are allowed to be refined by extra levels, and demonstrates that the solution is converging in some sense as the mesh spacing $h \to 0$. We observe that the two fixed meshes used in §4.6 have a mesh spacing at the aerofoil of one half and one quarter of the initial coarse mesh used in this chapter. Hence they are approximately equivalent to adapted meshes obtained with MAXLEV = 1 and 2 respectively. In fact the final mesh generated for MAXLEV=2 has a slightly

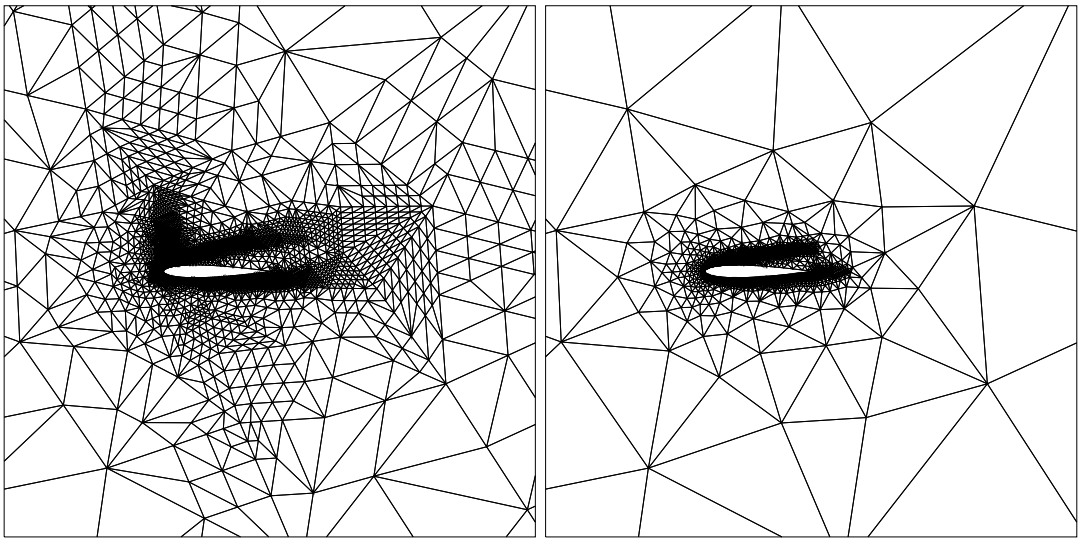

Figure 5.9: Mesh sections using (a) density gradient (θ_K^g) and (b) vorticity (θ_K^v).

more accurate value of C_l than the fixed mesh value and takes less than half the time to converge.

MAXLEV	TOL	C_l	C_d	CPU time (s)	Final elts
Fixed		0.52	0.28	780	5436
0		0.77	0.29	43	547
1	0.1	0.57	0.28	92	1467
2	0.1	0.50	0.28	320	3230
3	0.1	0.47	0.27	1212	6468
4	0.2	0.45	0.27	1632	9586

Table 5.3: Case A2: Effect of different MAXLEV.

It should be emphasised that the choice of parameters (in particular TOL and MAXLEV) has not been optimized and faster or more accurate solutions may be obtained for other values of these quantities. The values chosen here do however appear to show that the use of h-refinement offers a more efficient way of solving the problem than a fixed mesh approach.

5.4.3 A3

The flow in this case is supersonic and involves a bow shock forming near the leading edge. In chapter 4, when a fixed mesh which is very coarse away from the aerofoil is used, this feature is not completely resolved. We would like to use an adapted mesh to correctly refine near the shock, and so use the gradient and residual error

indicators θ_K^g and θ_K^r in the refinement algorithm. The two indicators based on the velocity gradients (θ_K^v and θ_K^m) are not able to resolve the shock, and the modified residual indicator θ_K^d gives very similar results to θ_K^r.

The value of MAXLEVEL for this example is 4, so that large triangles away from the aerofoil may be refined to a small enough size. The value TOL is correspondingly increased to 0.4 so that the number of elements used is not too large. Table 5.4 lists the results obtained. The pressure and friction coefficients look very similar to the fixed mesh results in §4.6 and the final meshes are shown in figure 5.10. The density contours for both indicators are given in figure 5.11.

Err. Ind.	C_l	C_d	CPU time	Final elts
Fixed	0.32	0.46	1266	5436
θ_K^g	0.33	0.44	1368	8360
θ_K^r	0.33	0.45	1661	8819
Values in [18]	0.31-0.40	0.41-0.49		

Table 5.4: Results on adapted meshes for test case A3.

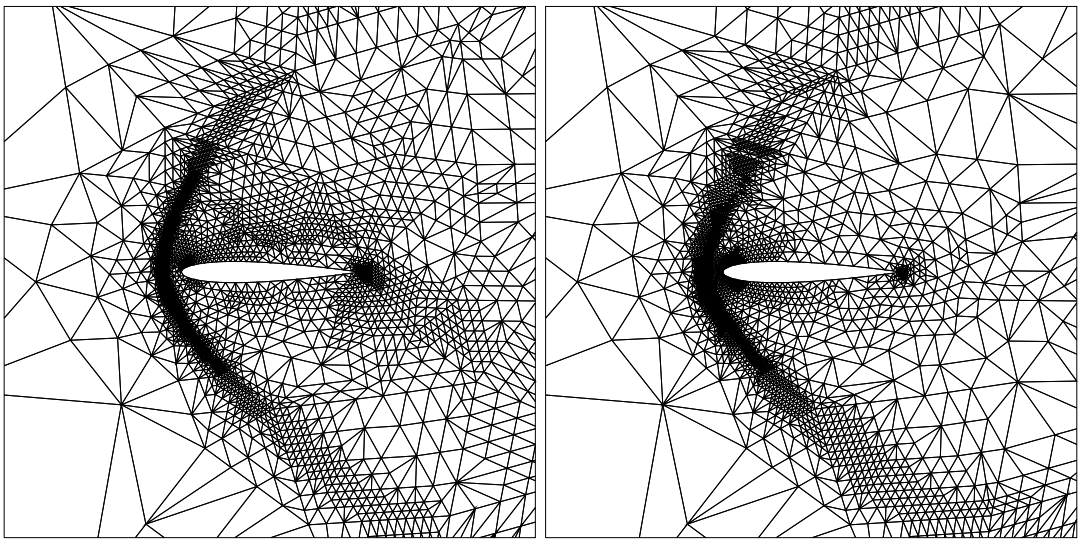

Figure 5.10: Mesh sections for case A3 using (a) density gradient (θ_K^g) and (b) residual (θ_K^r).

Both error indicators appear to resolve the shock further away from the aerofoil to a greater extent than the fixed mesh used in §4.6, although the convergence time is increased. The residual indicator adds many more nodes along the shock and at the leading edge of the aerofoil whereas the density gradient indicator put extra unnecessary nodes in regions above the aerofoil where the flow is smooth.

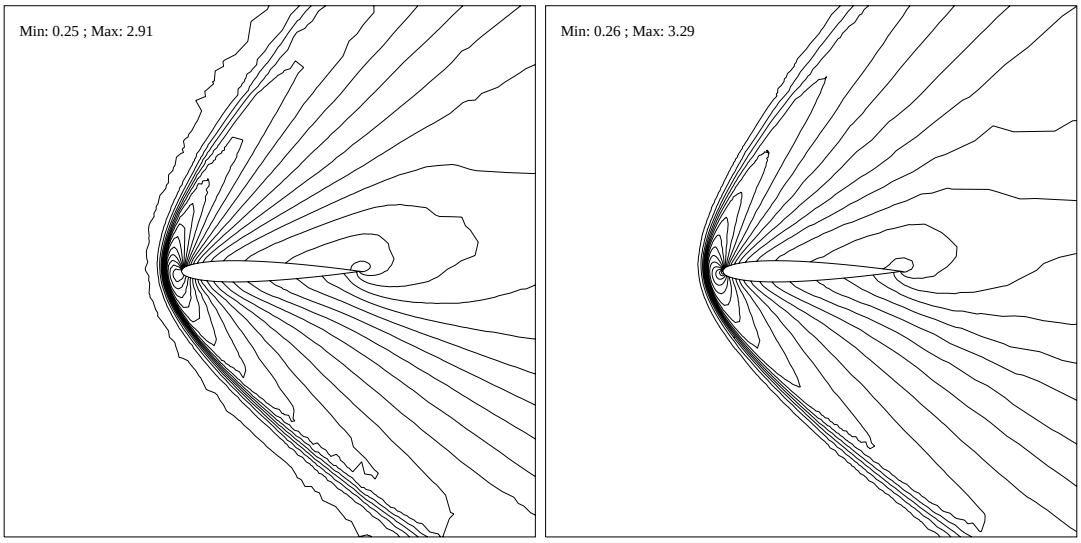

Figure 5.11: Density contours for case A3 using (a) density gradient (θ_K^q) and (b) residual (θ_K^r).

5.4.4 A6

In §4.6 this test case was solved on an unstructured mesh, when the wake behind the aerofoil was not resolved properly, and on a more dense structured mesh, when this wake could be clearly seen. We now see how refinement can be used on an unstructured mesh to detect this wake.

The error indicator chosen here is the residual based θ_K^r, since the density gradient indicator appears to completely fail to detect the wake when starting on such a coarse mesh. The value of TOL is 0.3 and MAXLEV is set as 5, and table 5.5 shows the results for this problem, along with the results obtained on fixed (structured and unstructured) meshes in §4.6. A section of the final adapted mesh containing 8928 elements is shown in figure 5.12, and the contours of the Mach number are shown in figure 5.13.

Err. Ind.	C_l	C_d	CPU time	Final elts
Fixed (unstr.)	0.003	0.12	685	5436
Fixed (struct.)	0.0	0.11	4237	8192
θ_K^r	0.002	0.12	2211	8928
Values in [18]	0	0.10-0.14		

Table 5.5: Results on meshes for test case A6.

The wake formed behind the aerofoil is clearly detected using the residual error

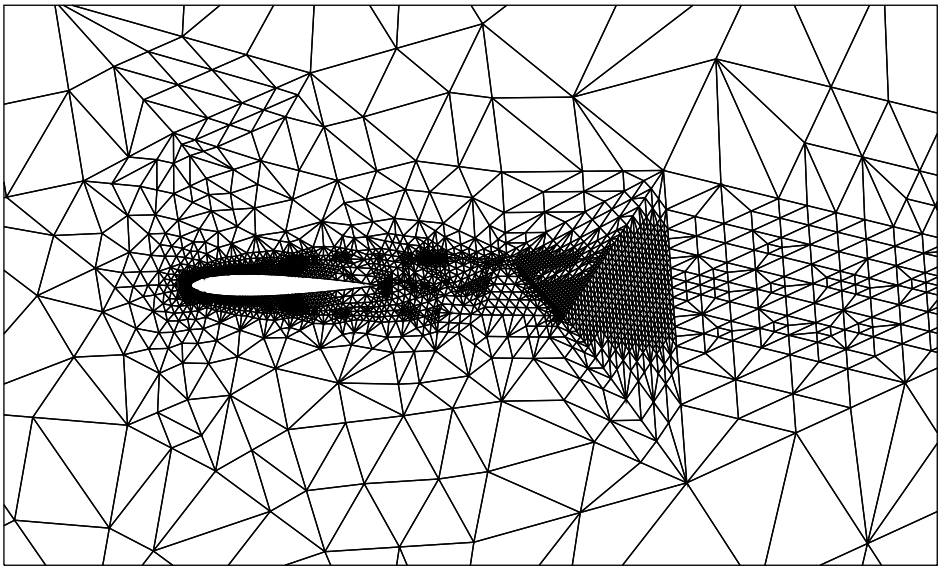

Figure 5.12: Mesh sections of final mesh for case A6.

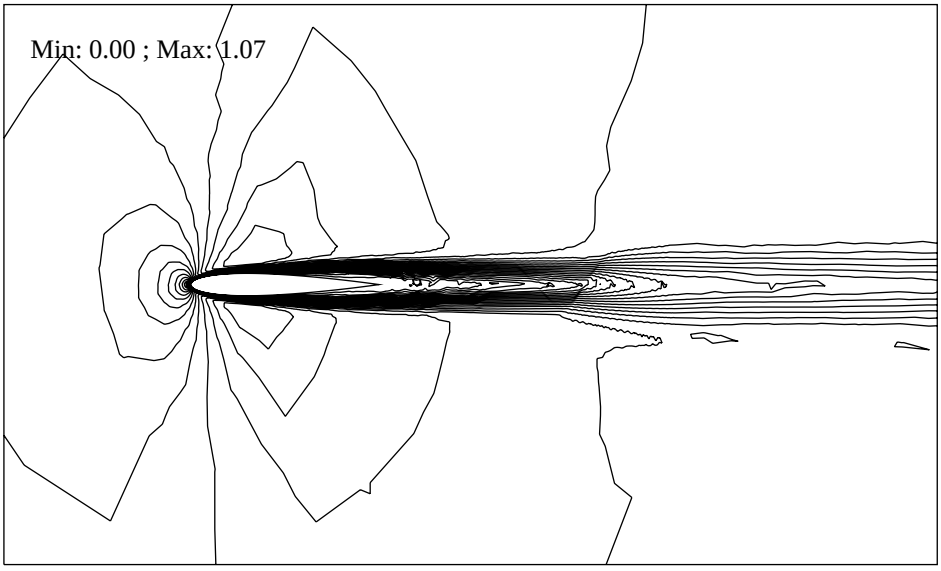

Figure 5.13: Mach contours for case A6 using residual error indicator.

indicator θ_K^r, although it is not resolved as well as the structured mesh solution in figure 4.22. The structured grid elements are highly distorted and aligned horizontally so that they capture the wake more accurately. An unstructured mesh automatically refined by adding extra nodes will never be able to match the solution on a structured mesh in this case, where the wake is aligned with the structure of the mesh. Only by increasing MAXLEV substantially would there be enough elements to fully capture the wake, but in practice this would exceed the amount of available memory. In the next chapter, we consider the possibility of allowing the elements to change shape, which could potentially overcome this problem.

5.4.5 Flow over a Flat Plate

This problem, known as Carter's problem [25], was considered for fixed meshes in §4.6.5. We now try to obtain solutions of greater accuracy using the adaptive approach of this chapter. The initial starting mesh, shown in figure 5.14, consists of 98 elements, and the boundary conditions are those defined in §4.6.5. The two error indicators used are θ_K^g and θ_K^r (the indicator θ_K^d gave very similar results to θ_K^r), with MAXLEV=5 and either TOL=0.05 for θ_K^r or TOL=0.5 for θ_K^g. Table 5.6 shows the final mesh sizes and CPU times obtained in these two cases, along with the equivalent values for the finest fixed mesh tried in §4.6.5.

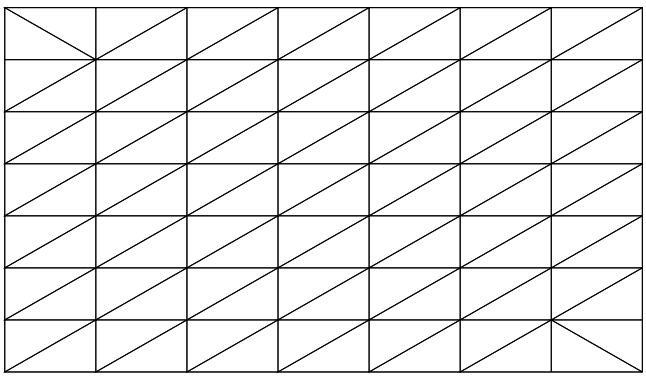

Figure 5.14: Initial mesh used for flat plate flow problem.

The pressure and friction coefficients are shown in figures 5.15 and 5.16. The contours of pressure and Mach numbers using the residual indicator are plotted in figures 5.17 and 5.18 and the final meshes for both indicators are shown in figures 5.19 and 5.20.

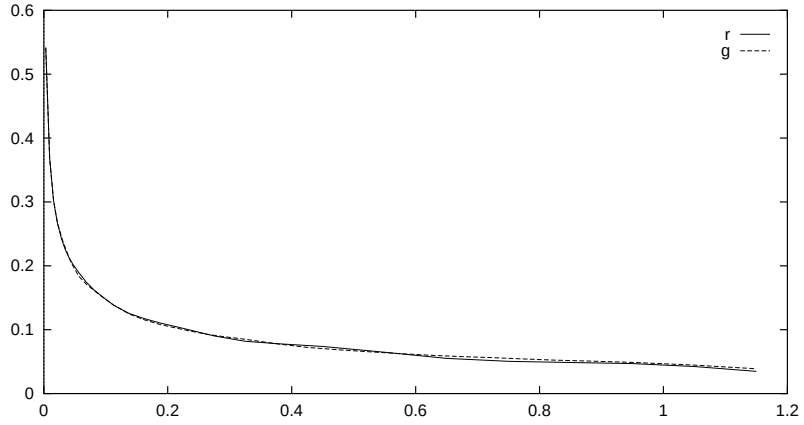

Figure 5.15: Pressure coefficients for flat plate flow problem.

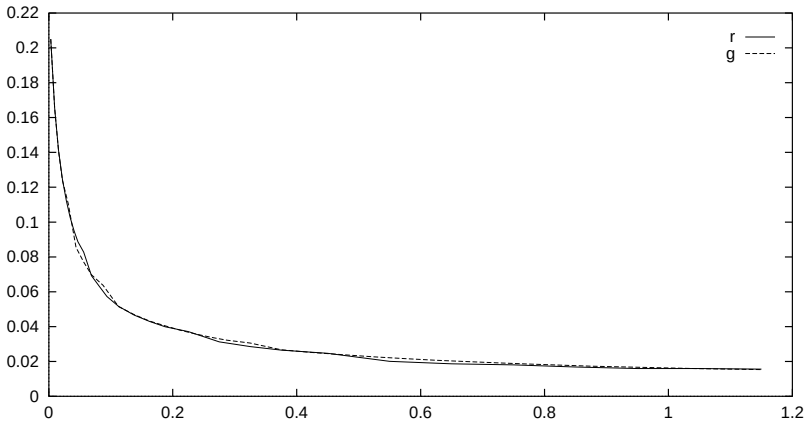

Figure 5.16: Friction coefficients for flat plate flow problem.

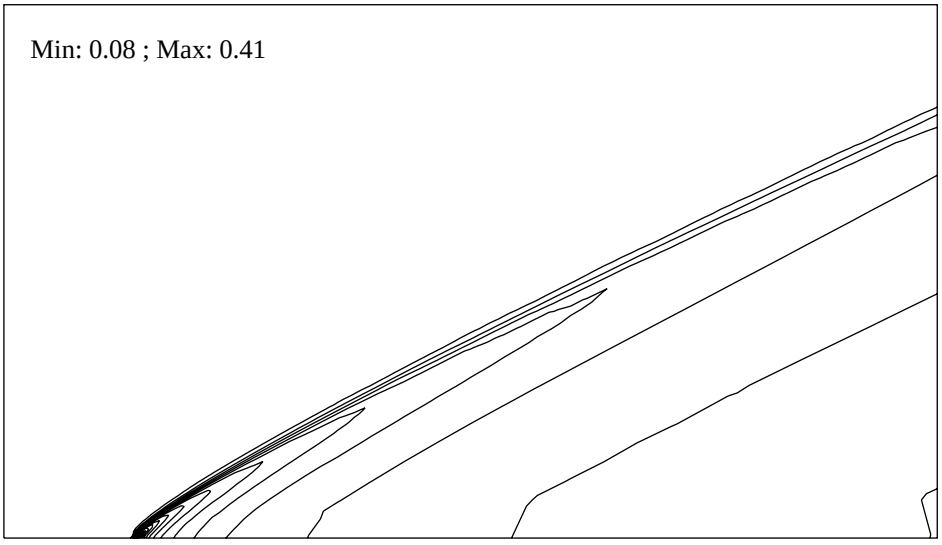

Figure 5.17: Flat plate flow: contours of pressure on final mesh.

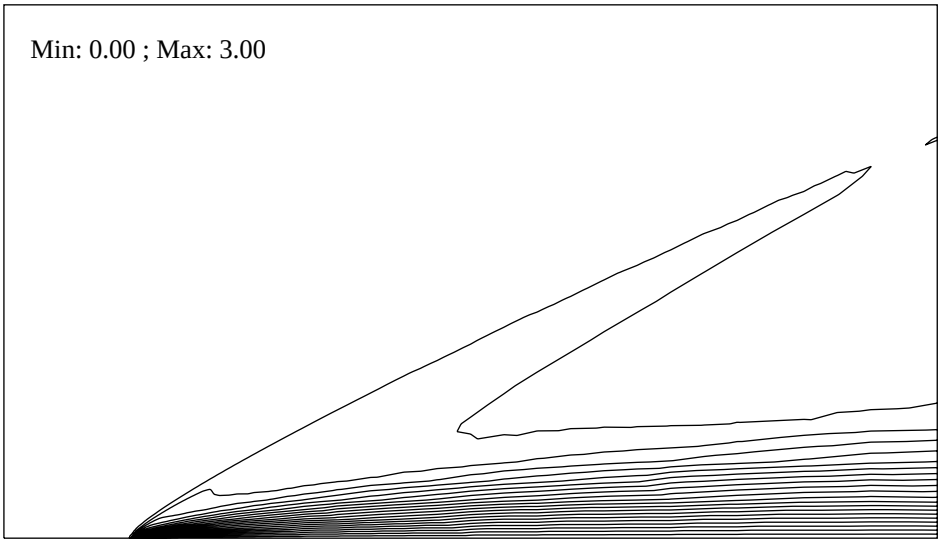

Figure 5.18: Flat plate flow: contours of Mach number on final mesh.

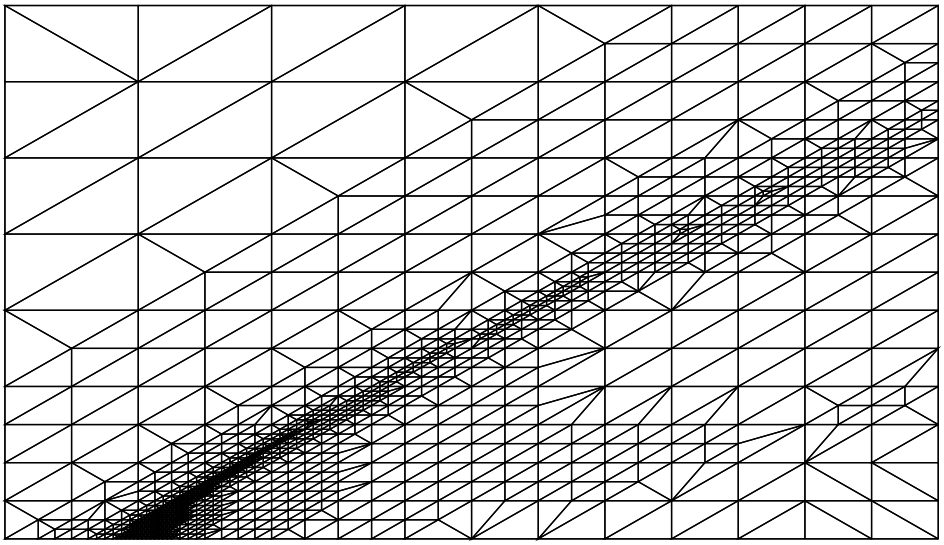

Figure 5.19: Flat plate flow: final mesh using θ_K^r indicator.

Err. Ind.	CPU time	Final elts
Fixed	2462	25088
θ_K^r	146	2087
θ_K^g	251	2389

Table 5.6: Results on adapted meshes for flat plate flow.

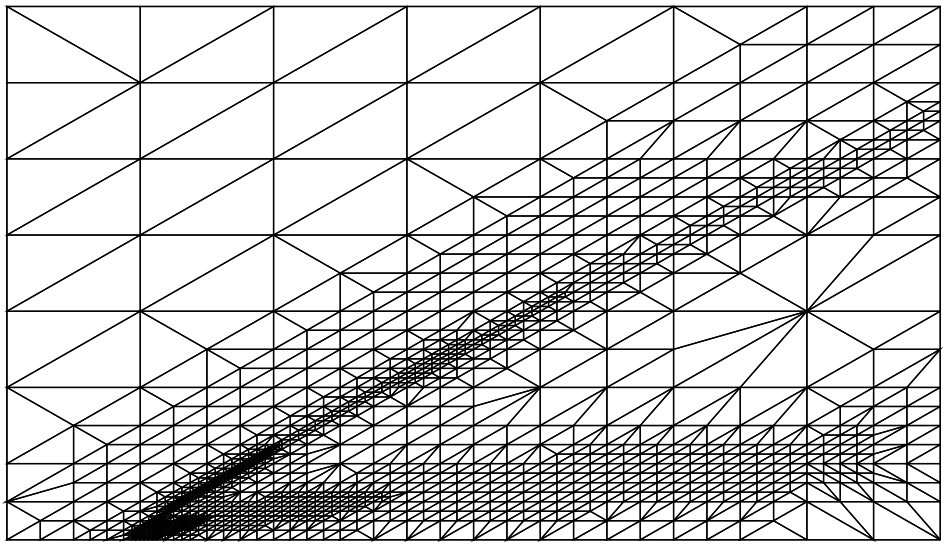

Figure 5.20: Flat plate flow: final mesh using θ_K^g indicator.

The value of MAXLEV used here (=5) allows the mesh to be refined so that the finest elements have half the spacing of the finest mesh used in §4.6.5. Hence we might expect more accurate solutions in those regions where the most refinement has taken place. Comparison of the coefficients of pressure and friction between fixed (figures 4.26 and 4.27) and h-refined (figures 5.15 and 5.16) meshes indicates that the solution on the adapted mesh is indeed slightly different. In particular the values of the pressure coefficients are higher near the leading edge of the plate, and this compares more favourably with results obtained elsewhere (e.g. [107],[35]). More significantly the time taken to fully converge on the refined meshes is reduced by at least a factor of ten. The two indicators tried behave very similarly, although the gradient indicator puts more nodes in the region of the boundary layer. This example further demonstrates the importance of stability on the coarse mesh, so that refinement, which is essential for accuracy, may be carried out efficiently.

5.5 Summary

In this chapter we have discussed a method for adaptively refining a mesh in order to obtain a numerical solution in an efficient way. The two most important aspects of any adaptive approach are the method for indicating where to refine and the exact way in which the mesh is refined. Here we considered several error indicators based on the current numerical solution (§5.2) and outlined an algorithm for adding extra nodes (h-refinement) to an unstructured mesh (see §5.3).

Results presented here have been compared with the solutions obtained on fixed meshes in chapter 4. For example, finding an adaptive solution for a system of Burgers' equations was shown to be several times quicker than obtaining a solution of equivalent accuracy on a fixed mesh.

For the Navier-Stokes equations, most results given here contained more elements than the fixed meshes used in the previous chapter, and hence the time taken to reach a solution was greater. However the accuracy of solutions was shown to be improved, so that the values of the lift and drag coefficients were more in line with results obtained elsewhere, and flow features away from the aerofoil were able to be detected.

This suggests that an adaptive approach can either be used to provide an equally good solution as a fixed mesh but with fewer elements, or used to obtain more accurate solutions in an efficient way—by using a coarse initial mesh and a good error indicator, no prior knowledge about the location of flow features is required.

Of the indicators used, perhaps the most suitable for using when the mesh is initially very coarse is the residual error indicator θ_K^r. The others based on gradients in the solution failed either to detect shocks or wakes, and at least at moderate Reynolds numbers, the indicator incorporating diffusive effects appeared to have little additional effect.

There are some aspects of the approach taken here which need to be addressed further. The optimal values for the two parameters which control the amount of refinement, MAXLEV and TOL, vary for different problems and it is not clear how best to choose them in each case. When starting on coarse meshes, the maximum level of refinement, MAXLEV, needs to be large enough to allow large triangles to be refined so that they resolve flow features accurately. The refinement tolerance TOL is used in conjunction with MAXLEV to specify the required accuracy of a

solution.

One drawback of using an initial mesh which is coarse is that the final mesh reflects the shape of the elements used in the starting mesh, since the the shape of each child element is determined by the shape of its parent. As a result, the mesh looks non-smooth and irregular, and stretched elements (which may improve convergence of a solution along boundary layers for example) cannot occur, unless present on the original mesh. This issue is considered in chapter 6, where we describe an approach for allowing nodes to move locally around the mesh.

Chapter 6

Adaptivity II: r-Refinement

6.1 Introduction

A method for adding extra nodes to a mesh (h-refinement) was described in the previous chapter, and this approach was shown to be more efficient than using a fixed mesh throughout the solution process. However there are some problems with h-refinement which we would like to overcome. These concern the quality of the mesh and shape of the elements, which are determined by the initial mesh—the geometry of final mesh is very dependent on this mesh, and there is no scope for allowing elements to change their shape and align themselves with flow features in the solution.

Another strategy for mesh refinement is to allow nodes on the existing mesh to be relocated (r-refinement). Some popular techniques for doing this, such as the method of moving finite elements [97] and node equidistribution [33] are outlined in §6.2. The remainder of this chapter is concerned with an alternative approach (similar to that described by Hubbard in [64]), where nodes are repositioned according to a local weighting formula between time-steps. Because it is a local process, it is inexpensive, and has no effect on the mesh connectivity, but allows elements to change shape and hence resolve features of the solution more accurately than a fixed mesh.

In §6.3, the algorithm for local node movement is fully described. Further details of using the method and some results which show that improved solutions may be obtained appear in §6.4. However for the examples we have tried there appear to be some limitations with this method, for example convergence is slow during node

movement and there is no way of controlling the error in the solution.

A novel idea is to combine the two types of refinement, h and r, to give an algorithm which both adds extra nodes to the mesh in the regions where they are required and moves nodes in order to allow the mesh to resolve the solution more efficiently. In practice the method involves a modification of the h-refinement algorithm given in §5.3, and it is discussed in §6.5. In §6.6, for the examples considered in previous chapters (Burgers' system, transonic flow around NACA0012 aerofoil and supersonic flat plate flow), some results are presented and compared with results obtained using h and r-refinement separately.

This technique overcomes some of the problems arising from using either of the methods separately. The ability to add extra nodes allows the error in the solution to be controlled in some way, and the node movement means that the final mesh is no longer coupled to the initial mesh, and elements may become aligned with flow features.

6.2 Techniques of r-Refinement

There are a number of methods which use the idea of moving nodes around a mesh rather than adding extra nodes, and we briefly describe some of them in this section. One of the most common approaches is that of moving finite elements (MFE), which was introduced by Miller and Miller [97]. This method is suitable for any time-dependent p.d.e., and although considered for 1-d problems in [97], it has subsequently been extended to higher dimensions. A full description of MFE and its development, along with properties of the method, is given by Baines in [7].

The difference between MFE and standard finite elements is that the positions of the nodes are no longer fixed but become additional unknowns which need to be determined along with the nodal coefficient values. In 1-d, this means that there are $2N$ unknowns (if N is the number of internal nodes) so an extra N equations are required in addition to the N equations obtained from the usual Galerkin method. The $2N$ equations form a system of ordinary differential equations (o.d.e's) which are then solved using an o.d.e. solver (usually implicitly).

This approach allows the nodes to be automatically moved to regions of the solution where high resolution is required, e.g. near shocks, without the need of any further refinement procedures. However, for certain values of the solution,

the extra equations introduced can cause the mass matrix in the o.d.e. system to be singular. In [97] this is overcome by the addition of a penalty function in the formulation, involving a numerical parameter.

The extension of the method to general systems in 1-d is given by Gelinas *et al.* in [46], where several example problems are considered and the development of the method for two dimensional problems is summarized by Miller in [96]. In 2-d, elements may become highly distorted, leading to mesh tangling, so a further parameter can be added to the penalty function to avoid this (and the choice of these parameters may be problem dependent). The number of equations to be solved at each step is now $3N$, significantly more expensive than the fixed finite element approach.

Many results for a variety of problems in one and two dimensions, solved using MFE, are presented by Zegeling [129]. Further examples are given in [83] where Johnson *et al.* avoid the use of a penalty function and use explicit time-stepping, which limits the time-step size to prevent mesh-tangling.

Node equidistribution is an alternative way of generating a new mesh at each time-step to MFE. This is described by Coyle *et al.* in [33] and involves positioning the nodes so that a weighting function (such as the gradient or curvature of the numerical solution) is evenly distributed over the entire mesh. The new positions of the nodes are obtained from the solution of either an algebraic or differential set of equations at each step, so that the finite element equations may be solved on the new mesh.

A similar approach is used by Lohner *et al.* [90] for the compressible Euler equations by considering element sides as springs of prescribed stiffness and moving the nodes until the spring system is in equilibrium. Another way of moving nodes for hyperbolic conservation laws is outlined by Lucier in [91], where nodes are moved according to the method of characteristics.

In [64], Hubbard outlines a local method for moving nodes, so that each node is repositioned according to the neighbouring patch of elements surrounding it. We adopt a similar approach in this chapter as it appears to overcome some of the problems of associated with a method such as MFE. It does not require the expensive solution of extra equations, singularities in the matrix system will not occur and mesh tangling can be avoided more easily.

6.3 A Local Node Movement Algorithm

As noted in the previous section, one simple idea, described in [64], for adapting the mesh is to reposition each node by a displacement based upon information from only the neighbouring nodes and elements. In this section we discuss the algorithm for 2-d unstructured meshes in detail—the simpler 1-d version is described in [64].

Like the equidistribution methods (e.g. [33]) mentioned in §6.2, this method seeks to position the nodes so that some function of the solution is more evenly distributed across the mesh. However, rather than compute the new mesh globally, it moves nodes locally, which is a much less expensive approach.

The formula used to compute the new position $\mathbf{s}_{\text{new}}$ of a node is

$$\mathbf{s}_{\text{new}} = \frac{1}{\sum_{i=1}^{m} w_i} \sum_{i=1}^{m} w_i \mathbf{s}_i \tag{6.1}$$

where m is the number of surrounding elements, $\mathbf{s}_1, .., \mathbf{s}_m$ are the centroids of these elements and $w_1, .., w_m$ are the weighting functions defined below. The result of this procedure carried out for each node is to place the node at the weighted average of the centroids of the surrounding triangles as shown in figure 6.1. In practice, all the new node positions are computed prior to being updated (a Jacobi-type process), rather than in a Gauss-Seidel fashion which is dependent on the order in which the nodes are updated.

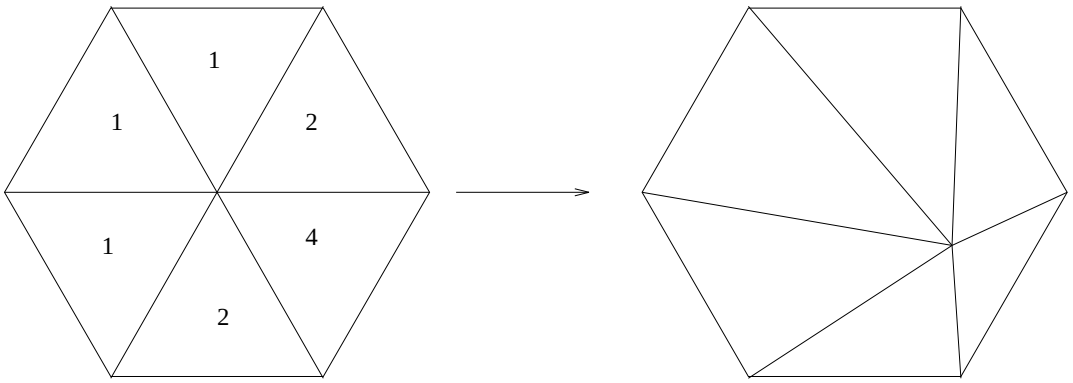

Figure 6.1: Local node movement, showing weighting functions for each element.

The weighting functions w_i are values defined on each element, and in [64], quantities such as the arc length or the curvature of the local solution are used. In §5.2, we considered a selection of error indicators used in h-refinement to specify which elements are to be refined, and it seems natural to use one of these element

based quantities as our choice for w_i. For example, if the residual based indicator θ_K^r is being used,

$$w_K = \left(\int_K |R(\mathbf{U})|^2 d\mathbf{x} \right)^{\frac{1}{2}} \tag{6.2}$$

where $R(\mathbf{U})$ is the residual resulting from inserting the current numerical solution into the original p.d.e.'s, and for the Navier-Stokes equations is defined by the expression (5.7). The effect of using an error indicator is to allow nodes to move to regions where the error is largest, so that the mesh should be able to more accurately resolve flow features.

The implementation of this node movement algorithm within a solution procedure is straightforward—one or more iterations of the routine are performed after selected time-steps (see §6.4 and §6.5 for further details). Once the node positions have been updated, no interpolation of the solution values is carried out, and the next time-step continues immediately. Because the node connectivity is unchanged and the algorithm is a local process, the additional computational cost is small and results suggest that this is outweighed by the benefits of having a more suitable mesh.

If the weighted average formula (6.1) is used as shown, then the situation known as mesh tangling may occur in some circumstances. These arise when the triangle centroids surrounding a node no longer form a convex polygon around the node. In this case it is possible for nodes to be moved such that triangles lose their positive orientation and elements become tangled (see figure 6.2). This can be avoided by restricting the amount that a node may move in one node movement iteration and, as in [64], we ensure that a node never moves more than

$$\Delta \mathbf{s}_{\max} = \min_i \frac{A_i}{\max_{j=1,2,3} l_{ij}} \tag{6.3}$$

where i denotes all the triangles neighbouring the node, A_i is the triangle area, and l_{ij} are the lengths of the triangle sides. This ensures that a node will never move a distance more than half the length of the shortest height of any of the surrounding elements. The new weighting formula becomes

$$\mathbf{s}_{\text{new}} = \alpha \frac{1}{\sum_{i=1}^m w_i} \sum_{i=1}^m w_i \mathbf{s}_i + (1-\alpha)\mathbf{s}_{\text{old}} \tag{6.4}$$

where

$$\alpha = \frac{\min(\Delta \mathbf{s}_{\max}, \Delta \mathbf{s})}{\Delta \mathbf{s}} \tag{6.5}$$

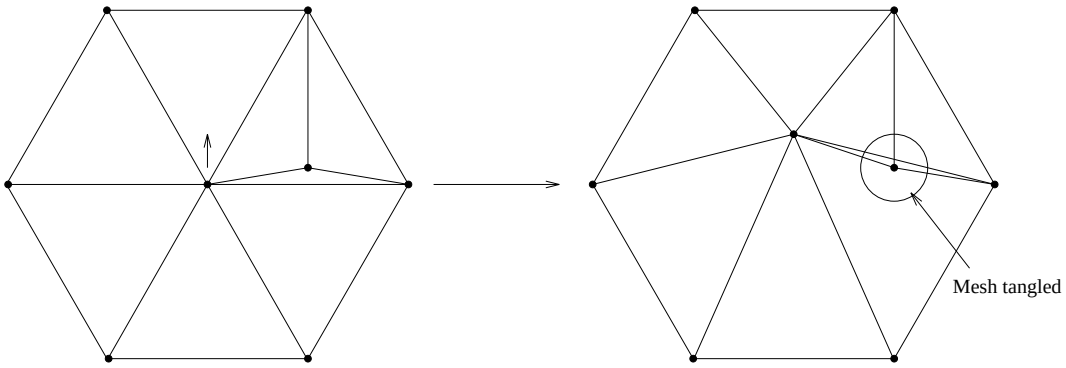

Figure 6.2: Example of mesh tangling occurring.

and $\Delta\mathbf{s}$ is the node displacement calculated using (6.1).

Boundary nodes are treated in the following way. On straight boundaries a new node position is calculated as above, and then projected back onto the boundary. However for curved boundaries (e.g. aerofoils) we fix the nodes, rather than try to relocate them along the wall surface, since our initial attempts to allow them to move caused tangling to occur in highly curved regions. We believe that this problem can be solved, but becomes more difficult to do so if the geometry is especially complicated (e.g. multi-element aerofoils). By ensuring there are sufficient nodes along the boundary, either using h-refinement to add the extra nodes, or using a grid which is initially fine near the boundary, it appears to be satisfactory to keep these nodes fixed.

In the following section, we show some results of using this r-refinement algorithm on a mesh with a fixed number of nodes, and in §6.5 discuss a method which combines the node movement introduced here with the h-refinement techniques introduced in the previous chapter.

6.4 r-Refinement Only

The node movement algorithm introduced in §6.3 may be used within the time-stepping solution strategy described in chapter 2, and here we state details of how this is done in practice and show some results for the system of Burgers' equations (where we know the exact solution) and the full Navier-Stokes equations.

No node movement is carried out initially and the solution is left to converge on the initial mesh until the L_2 norm of the steady-state residual has been reduced

to below 10^{-4}. After this point, the node movement algorithm is called after every time-step until a fixed number (N_U) of movement updates have been performed. Each time the node movement algorithm is called, one or more updates (updates per step or UPS) are carried out. In §6.4.1, the effect of different values of these parameters is observed.

Since the solution only converges very slowly whilst movement updates are being carried out it is necessary to continue time-stepping after N_U updates have been made (after which no further node movement is carried out), until the solution has fully converged.

For some problems, the shape of the triangles becomes highly distorted and so when the minimum height of any element has gone below 10^{-3}, the surrounding nodes are no longer moved.

6.4.1 Numerical Examples

We present here some results of using the solution scheme outlined above. As in chapters 4 and 5, the finite element method used is that of modified Galerkin least-squares, local time-stepping is used (with a Courant number of 50), and the GMRES linear solver has Krylov dimension of 25 (these values are not necessarily optimal, but are robust).

Burgers' System

The coupled system of Burgers' equations introduced in §4.5 has an exact known solution and so it may be used to quantitatively measure the accuracy of a method. The mesh on which the problem (in which the diffusion constant ϵ=0.001) is solved contains 568 elements and is shown in figure 6.4(a). The three error indicators given in §5.4.1 are each tried as the weighting function in the node movement formula (6.4). These are the residual based θ^r_K, the exact error θ^e_K and the solution gradient θ^g_K.

Numerical experiments indicate that it is more efficient to carry out more than one update in between each time-step, but doing too many updates causes the error to increase (as the mesh becomes too distorted). This is shown in figure 6.3, where the error norm of the converged solution (defined by the formula in §5.4.1) is plotted against the total number of updates carried out. Several values of UPS (updates

per time-step) ranging from 1 to 10 are shown, and the error indicator used here is the residual based θ_K^r.

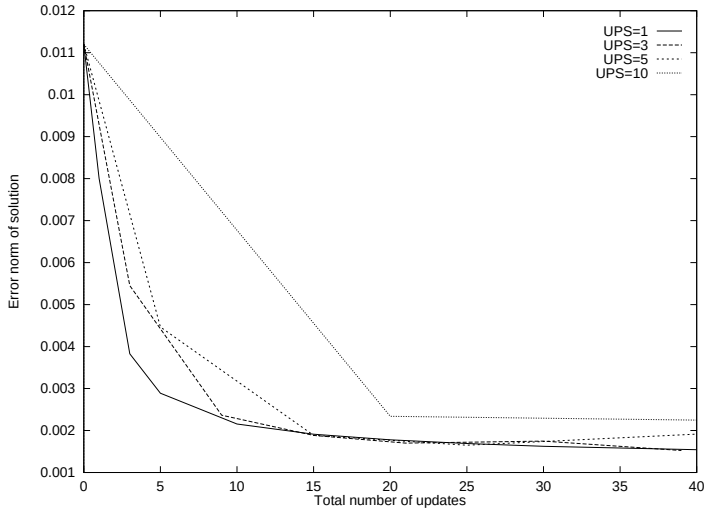

Figure 6.3: Solution error norm for different values of UPS and N_U.

On the basis of these results, we choose UPS to be five (as this will require fewer time-steps than if UPS is one), and carry out a total number of 25 updates, as further updates only reduce the error by very small amounts (in the case of the gradient indicator, only 10 updates are done as the error increases after this point). Hence node movement is performed for five (or two) time-steps, and then the mesh is fixed, to allow rapid convergence. Boundary nodes are allowed to move vertically or horizontally along the boundaries.

Table 6.1 shows the error norm of the final solution on the meshes generated using each of the error indicators, along with the error obtained on a stationary mesh. The final mesh in each case is shown in figure 6.4(b)–(d), and figure 6.5 shows the solution obtained using θ_K^r (mesh (b)).

Err. Ind.	UPS	N_U	Err. Norm	CPU time (s)
No mode movement			1.12e-2	10
θ_K^r	5	25	1.65e-3	33
θ_K^e	5	25	1.75e-3	36
θ_K^g	5	10	4.80e-3	20

Table 6.1: Results on r-refined meshes for Burgers' equations.

It is apparent that the node movement has a significant effect on the shape of the mesh, as elements shift towards the $y = x$ line along which the sharp front

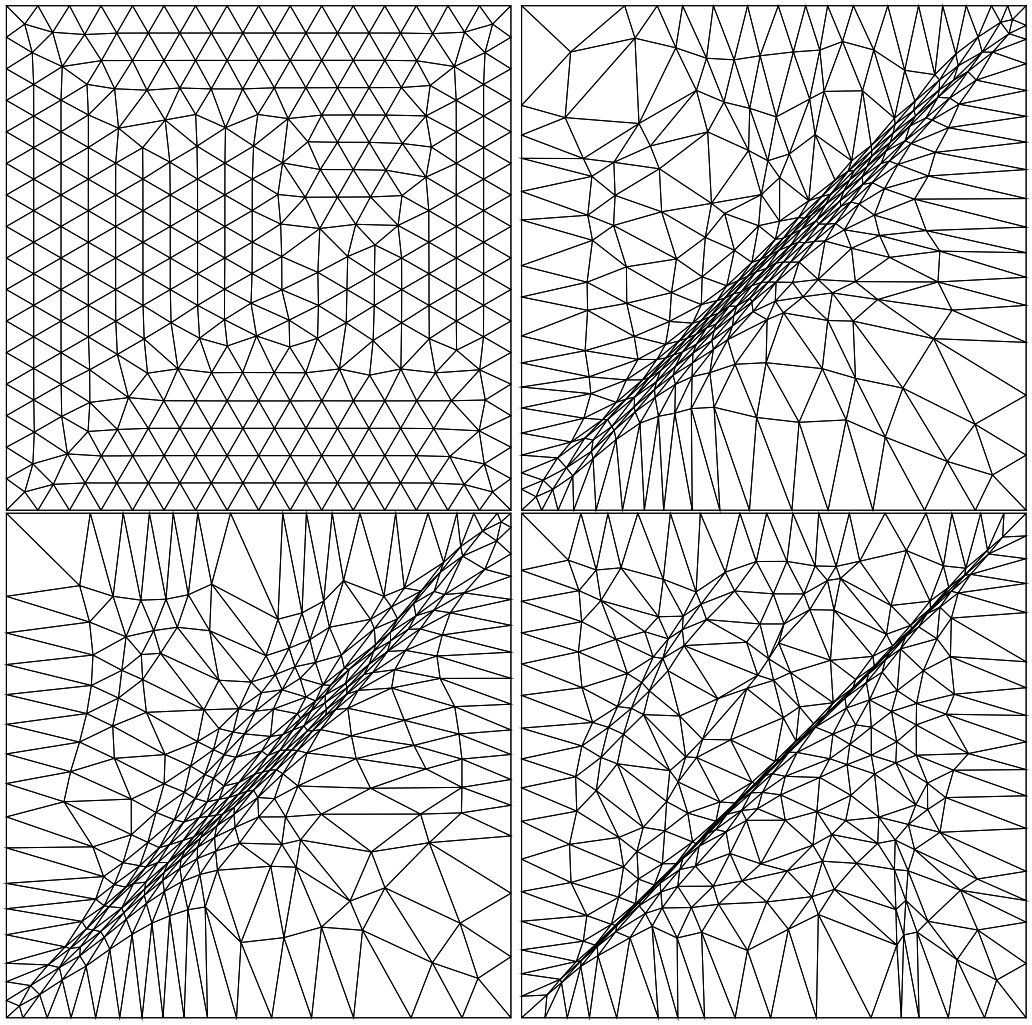

Figure 6.4: Final meshes obtained when using r-refinement only for Burgers' system (a) No movement (b) θ_K^r (c) θ_K^e (d) θ_K^g

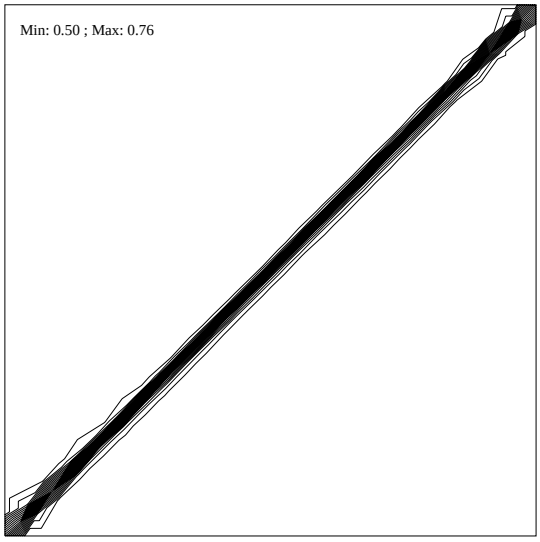

Figure 6.5: Contour plot of u using r-refinement only with θ_K^r.

lies. They also become distorted and begin to align themselves with this front, which allows better resolution of the front. This may be seen in the drop in the error norm, when compared to the stationary mesh, although this extra accuracy involves a much larger cost, as convergence during the mesh movement stage is very slow. The solutions are most accurate away from the boundaries due to the restricted movement of the boundary nodes.

The exact error indicator and the residual based indicator both behave similarly, whereas the gradient based error indicator θ_K^g gives the largest error as nodes are pulled in too closely to the front and elements become too highly distorted. A comparison with results obtained in previous chapters is given in §6.6.1.

Navier-Stokes Equations

We return to the full Navier-Stokes equations and attempt to use the node movement algorithm in same way as for the system of Burgers' equations. The first example we choose to use is the test case A2 from [18] which we have previously considered in both chapters 4 and 5. It consists of transonic flow around a NACA0012 aerofoil with Reynolds number of 500, a Mach number of 0.8 and an angle of attack of 10°.

The initial mesh used, containing 5436 elements, is the fixed mesh used in chapter 4 (and shown in figure 4.7). We use the primitive variable formulation of the Navier-Stokes equations here, and compare the three error indicators θ_K^r (the convective residual), θ_K^d (residual with diffusive terms) and θ_K^g (the density gradient). In practice, the solutions obtained using θ_K^d are very similar to those using θ_K^r and we do not show them here.

Based on the experiments carried out for the system of Burgers' equations above, we set the node movement parameters UPS=5 and N_U=25. Increasing the number of updates N_U distorts the elements further, but with very little change in the solution. If the number of updates per step UPS is reduced to 1 (but keeping N_U=25) then node movement will be carried out prior to 25 rather than 5 time-steps. The final meshes and solutions obtained in both case are nearly identical, but the latter example takes almost twice as long to converge. Figure 6.6 shows the residual norm plotted against the total nonlinear iterations for three cases—no movement, node movement with UPS=5 and node movement with UPS=1.

This shows the initial convergence on the starting mesh, followed by very slow convergence during the r-refinement stage, and the final solution being reached rapidly. The error indicator used in this case is θ_K^r.

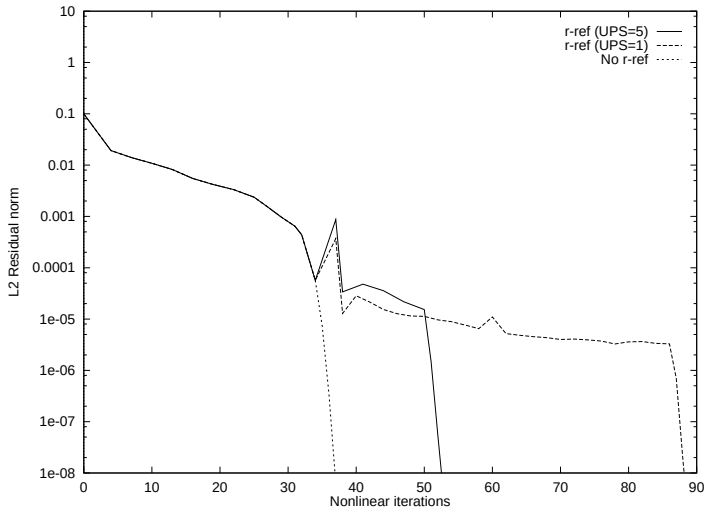

Figure 6.6: Convergence to steady state with no r-refinement, UPS=1 and UPS=5.

For the three error indicators mentioned, along with the stationary mesh, table 6.2 shows the times to fully converge and the lift and drag coefficients. Pressure and skin friction coefficients are shown in figure 6.7 and 6.8.

Err. Ind.	C_l	C_d	CPU time (s)
No mode movement	0.52	0.28	780
θ_K^r	0.49	0.28	1133
θ_K^g	0.55	0.29	1124
θ_K^d	0.49	0.28	1084

Table 6.2: Results on r-refined meshes for Navier-Stokes equations (case A2).

The refined mesh obtained using θ_K^r is shown in figure 6.9. In this case, the nodes are moved towards a region surrounding the aerofoil so that a clearly defined boundary appears between the very coarse outer elements and the dense mesh close to the aerofoil. Along this interface, the elements are very thin, although there is no obvious flow feature in this region that causes the interface to develop at this point. One possible reason the mesh has this appearance is that on the initial mesh nodes are already concentrated around the aerofoil, so that there is a wide variation between fine elements near the aerofoil and very coarse ones away from it. A more suitable starting mesh might be one where nodes are more evenly distributed. There

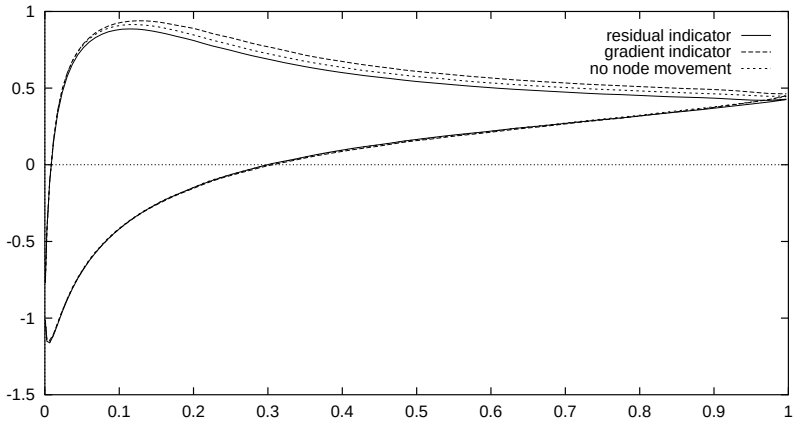

Figure 6.7: Pressure coefficients for case A2 using r-refinement.

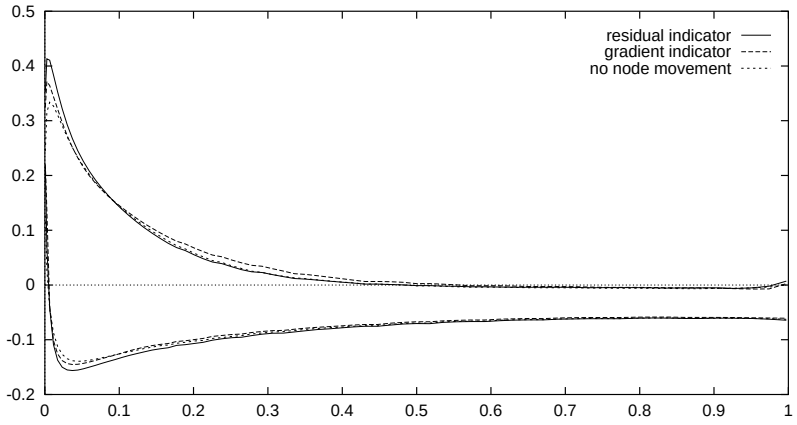

Figure 6.8: Friction coefficients for case A2 using r-refinement.

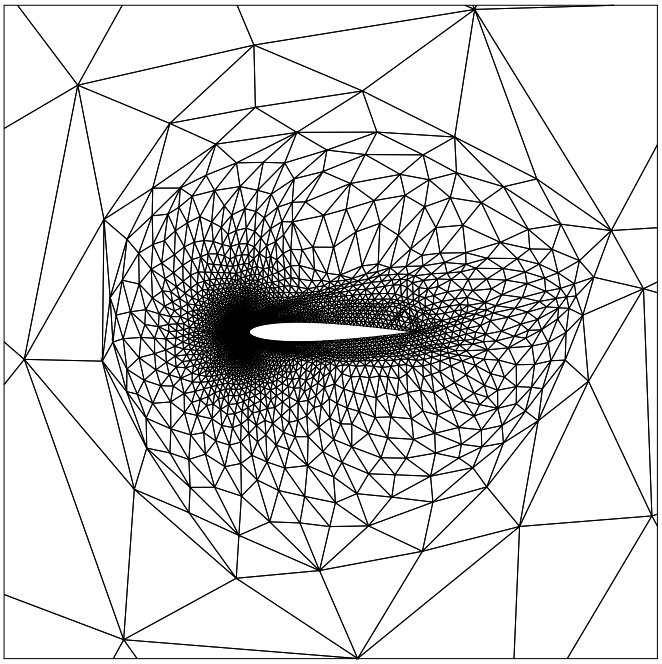

Figure 6.9: Final refined mesh using θ_K^r.

is a small decrease (towards a more accurate value) in the lift coefficient but at the cost of a large increase in CPU time.

Figure 6.10 shows the mesh refined using θ_K^g, and it can be seen that the elements have not become so highly distorted. However along the aerofoil wall, nodes appear to be pulled away from the boundary, and this is reflected in the rather high value for the lift coefficient.

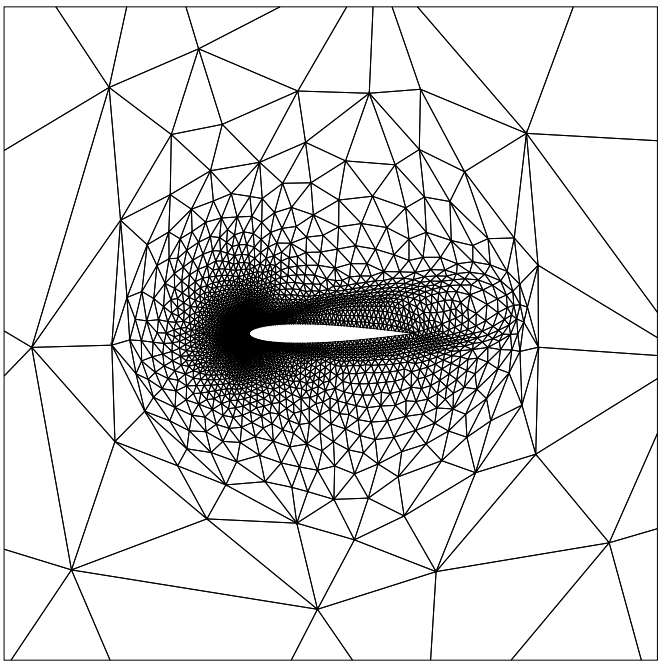

Figure 6.10: Final refined mesh using θ_K^g.

The second example considered here is the test case A3 involving a bow shock (Re=106, M=2, $\alpha = 10°$). The error indicator used as the weighting function is the residual based measure, θ_K^r, which gives better results than θ_K^g in the case A2 above. A similar initial mesh as before is used, and both the final mesh and the contours of the density are shown in figure 6.11. There is a marked improvement in the solution compared to the fixed mesh (see figure 4.16), since the nodes are moved into the shock location so that elements become aligned with the shock, which is clearly resolved. The time taken to reach convergence is 1485 seconds, compared with 1266 seconds for the fixed mesh.

In the examples considered above, there are a number of problems with using the node movement algorithm of §6.3 as the sole refinement technique. In all the cases, there was a considerable increase in the time taken to reach a final solution, due to a sharp drop in the rate of convergence when the mesh is being moved (see

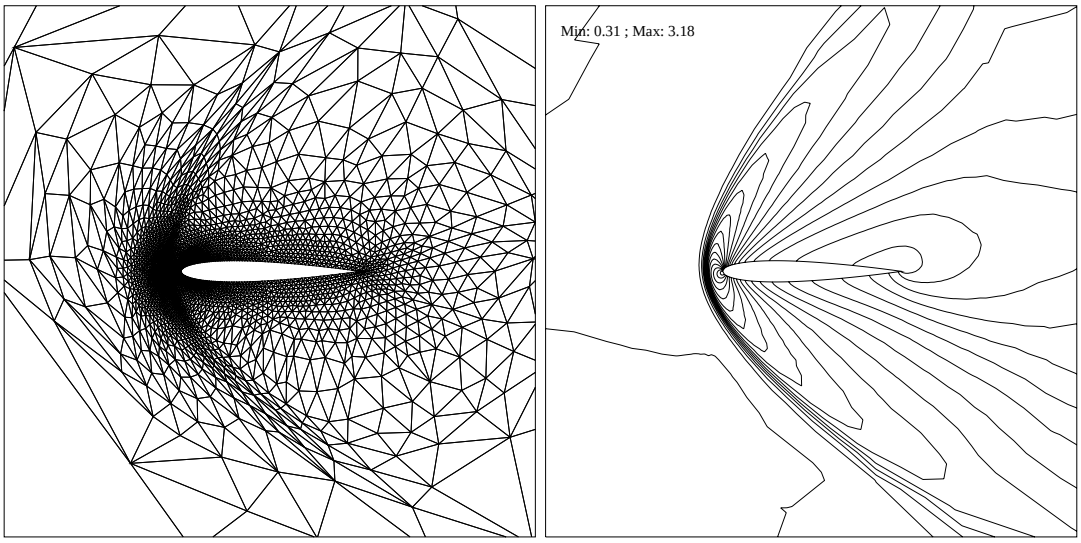

Figure 6.11: Final refined mesh and density contours for case A3.

figure 6.6). Also, it is not clear, for the Navier-Stokes equations, how many node updates should be carried out, and we base the selected values of N_U upon the results obtained for the system of Burgers' equations. However it does appear that performing several updates per step is more efficient than taking only one. In the case of flow around the NACA0012 aerofoil, a more even distribution of nodes in the original mesh may prevent the highly distorted elements occurring in smooth regions, for example in case A2. The final example, case A3, demonstrates that node movement can be used to refine meshes effectively, by altering the shape of elements so that they are more aligned with a particular flow feature.

We conclude that as a general refinement technique, node relocation *by itself* doesn't appear to be entirely successful. In the next section we discuss the possibility of combining two different refinement strategies—the node movement described in this chapter and the addition of extra nodes to the mesh (h-refinement) as outlined in chapter 5.

6.5 hr-Refinement

We have now introduced two separate approaches for mesh refinement. In h-refinement, extra nodes are added to the mesh in positions determined by some form of error indicator. As more nodes are placed in this region then it is hoped that the total error in the numerical solution will be reduced. One problem with the

type of refinement we have used here is that elements cannot change their shape, which is determined by the shape of the elements in the initial mesh. As a consequence, a mesh contains more nodes than are necessary in regions such as shocks or boundary layers—if elements could be reshaped so that they align themselves according to the flow, then the number of nodes needed could be reduced. In the method of r-refinement, no new nodes are added, but nodes are moved towards regions where the error indicator is large. The node repositioning is carried out on a local basis, and hence is relatively cheap. This approach also allows elements to change shape, but lacks the error control obtained by using h-refinement, i.e. there is no way of reducing the error below a certain point.

It would appear sensible to attempt to combine these two techniques in order to retain a level of error control without putting in too many additional unnecessary nodes. We describe in this section how such a combination can be implemented in practice.

The procedure is based upon the h-refinement algorithm presented in chapter 5. The algorithm is modified by repositioning the nodes prior to each refinement step. This provides a natural stopping point for the mesh movement algorithm, so that when extra nodes are no longer being added to the mesh, no further mesh movement is applied. This overcomes the problem of when to stop moving the mesh in the r-refinement algorithm (see §6.4).

A very simple flowchart of the complete algorithm is given in figure 6.12. As for the individual h and r refinement methods, no refinement is carried out until a partially converged solution on the initial mesh (which is usually coarse) has been obtained. Here, partial convergence means that the L_2 norm of the residual vector of the nonlinear system (generated from the finite element discretization of the steady p.d.e.'s) is below 10^{-4}. The tolerance for full convergence is as in chapters 4 and 5 (10^{-8}).

Once this initial stage is over, the algorithm enters a solve/move/refine iteration until no more elements are to be refined, and this is determined by the error indicator, the tolerance TOL and maximum level of refinement MAXLEV (see §5.3). Each time the node movement algorithm is called, one or more updates are carried out and the error indicator, already calculated to determine which elements are to be refined, is used as the weighting function to determine the new node positions.

No interpolation is performed on the relocated nodes, as there is no noticeable

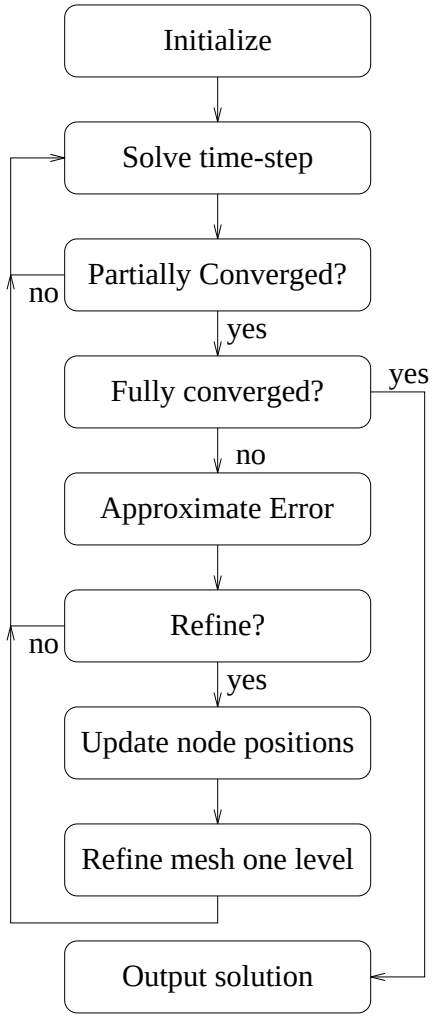

Figure 6.12: Flowchart of hr-refinement algorithm.

improvement in convergence if solution values are interpolated to take account of the adjusted node positions. Immediately after the node update, the mesh is refined by one level, and solution at the next time-step follows.

6.6 Results using hr-Refinement

In order to present results of using the algorithm outlined above we adopt a similar approach to the previous two chapters. Firstly we consider the system of Burgers' equations where the analytical solution is known. We then show solutions for some of the GAMM test cases [18], and finally give results for the problem of flow over a flat plate.

6.6.1 Burgers' Equations

Once again, before presenting results for the Navier-Stokes equations, we first show some solutions of the system of Burgers' equations introduced in §4.5, obtained using the combination of h and r refinement outlined in §6.5. The value of the diffusive constant ϵ is 0.001 and, as in §5.4.1, we use a coarse initial mesh of 40 elements, shown in figure 5.3. The three error indicators tried (which also serve as weighting functions for node movement) are the residual based θ_K^r, the exact error θ_K^e and the gradient indicator θ_K^g.

Because we can quantitatively measure the accuracy of a given numerical solution for this problem, we may precisely monitor the effect of varying the algorithm and parameters. We first fix the h-refinement parameters TOL and MAXLEV to 0.5 and 3 respectively, and take a similar approach to the r-refinement algorithm considered in §6.4, i.e. we specify in advance how many updates are performed. The h-refinement is implemented exactly as shown in chapter 5. Table 6.3 shows the error norm (defined by equation (5.15)), the time taken to fully converge and the number of elements on the final mesh, when UPS and N_U are fixed in this way, and the indicator used is θ_K^r. This shows firstly that the node movement reduces the error norm, and secondly that doing too many updates of the node positions can increase the error, as too many nodes are pulled towards the front along $y = x$ and the elements a short distance from this line begin to be aligned in the wrong direction.

UPS	N_U	Error Norm	CPU (s)	Elts
1	20	1.21e-3	64	684
5	25	1.40e-3	35	892
5	5	1.02e-3	34	828
1	5	8.64e-4	21	684
0	0	3.97e-3	18	548

Table 6.3: hr-refinement with predetermined N_U for system of Burgers' Equations.

The number of updates which appear to give the lowest error in table 6.3 approximately corresponds to the number of refinement steps which occur, hence we carry out the node updating prior to each refinement step. There is no advantage in only relocating nodes at alternate time-steps or less frequently, which serves only to increase the solution time since more time-steps are required. In addition, per-

forming more than one update per step does not reduce the solution error, as table 6.4 shows.

UPS	Error Norm	CPU (s)	Elts
1	7.82e-4	23	684
2	9.24e-4	27	799
3	1.21e-3	27	749
5	1.65e-3	34	597
10	1.98e-3	53	844

Table 6.4: hr-refinement with varying UPS for system of Burgers' Equations.

The final set of results demonstrates the effect of varying the different values of the h-refinement parameters MAXLEV and TOL, and using other error indicators. In this case, one update prior to each refinement step is performed, and table 6.5 shows the total number of updates performed (N_U), the error norm, the solution time and the number of elements on the final mesh. The solution and mesh from the third case given in the table are shown in figure 6.13. Figure 6.14 shows subsections of the meshes obtained using (a) the residual and (b) the gradient indicators.

Err Ind	MAXLEV	TOL	N_U	Err Nrm	CPU (s)	Elts
θ^r_K	3	0.1	4	6.77e-4	24	909
θ^r_K	3	0.3	6	7.76e-4	31	727
θ^r_K	3	0.5	6	7.82e-4	23	684
θ^r_K	4	0.1	8	2.97e-4	111	2861
θ^r_K	4	0.5	9	3.93e-4	71	2051
θ^e_K	3	0.1	7	6.51e-4	45	988
θ^e_K	3	0.5	7	8.47e-4	30	688
θ^g_K	3	0.1	8	1.29e-3	38	912
θ^g_K	3	0.3	16	2.35e-3	73	958
θ^g_K	3	0.5	25	4.56e-3	161	965

Table 6.5: Results on hr-refined meshes for system of Burgers' Equations.

As would be expected, in all the cases not only are nodes moved towards the front $y = x$ but extra nodes are also added in this region too. The exact error and residual based indicators perform similarly, but the indicator which measures the solution gradient is substantially slower and less accurate, perhaps because as figure 6.14(b) shows there is a sharp contrast in the shape of the triangles between those lying precisely on the front and the remainder. If the number of node updates per time-step (UPS) is increased, then again the error norm also increases, for similar

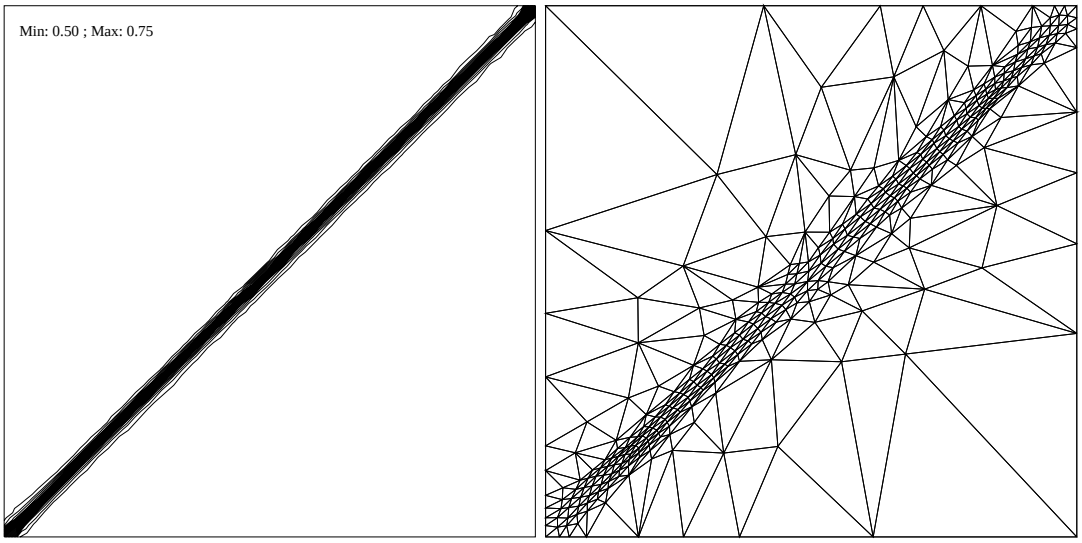

Figure 6.13: Contour plot of u and final mesh obtained when using hr-refinement for Burgers' system with θ_K^r indicator.

reasons to those just mentioned. By increasing the maximum level of refinement, a more accurate solution can be obtained, but reducing the tolerance has only a small effect on the accuracy (and solution time). Overall, the best approach is to use θ_K^r as the error indicator, with one node update step prior to each refinement of the mesh.

These results compare favourably with those obtained using the refinement techniques individually, and this is demonstrated by table 6.6 which shows the norm of the exact error and solution times of examples from the four methods discussed—fixed mesh, h-refinement, r-refinement and hr-refinement. (Note that in any of the cases involving a choice of parameters, it is possible that these could be optimized to allow minor improvements in the solution time.)

Ref Type	Err Ind	MAXLEV	TOL	UPS	Err Nrm	CPU (s)	Elts
None					1.03e-3	199	9350
h	θ_K^g	4	0.1		8.77e-4	34	1575
r	θ_K^r			5	1.82e-3	37	568
hr	θ_K^r	3	0.5	1	7.82e-4	23	684

Table 6.6: Results on refined meshes for Burgers' Equations.

Clearly all three adaptive approaches are an improvement over a fixed mesh, as they use far fewer elements (and hence less time) to obtain solutions of similar (or better) accuracy. However it appears that the combined strategy of both moving

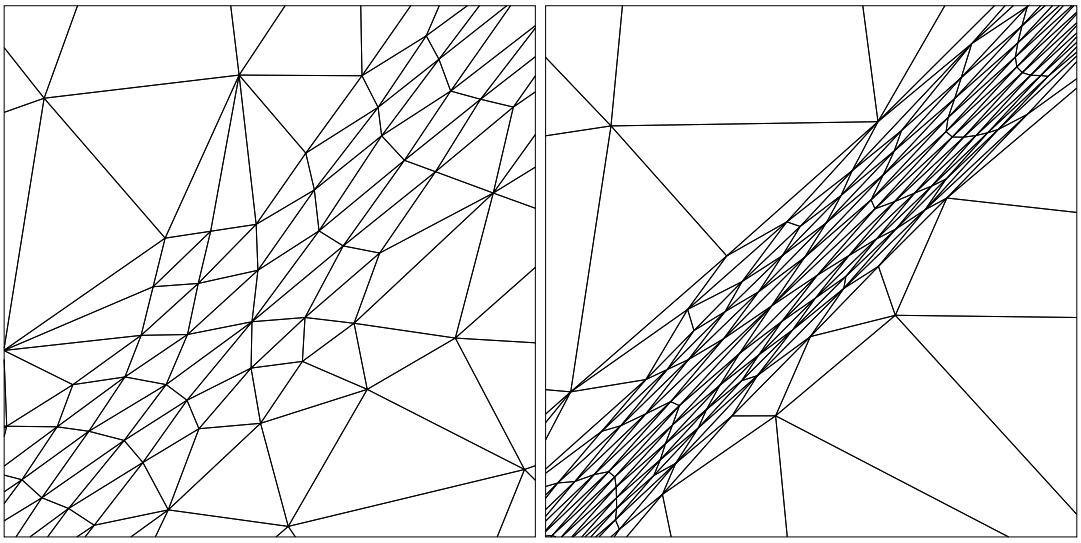

Figure 6.14: Subsection $[0.4, 0.6] \times [0.4, 0.6]$ of final meshes using (a) θ_K^r and (b) θ_K^g.

and adding nodes works best, giving both the lowest error norm and fastest solution time. It also appears that in general, θ_K^r is the best choice of error indicator. Of course this is a relatively simple problem compared to the full Navier-Stokes equations, with only a single stationary flow feature on a small part of the domain, but it serves to demonstrate the potential improvements which a combined hr approach might offer.

6.6.2 GAMM Test Cases

The cases we consider here are A2, A3 and A6. For the first of these, we see the effect of changing some of the numerical parameters, but in general we use the results from Burgers' system above to derive suitable starting values for these parameters. The initial starting mesh in all cases is the coarse mesh of 547 elements shown in figure 5.5.

Case A2

As explained in §6.5, we carry out the node movement prior to each refinement step, and initially set UPS=1, MAXLEV=3 and TOL=0.4, with the residual based error indicator θ_K^r being used. The final mesh and density contours obtained in this case are shown in figure 6.15. The most obvious effect of the node movement here is that the wake has been detected and elements have begun to align themselves along this flow feature. In table 6.7, the effect of increasing UPS, so that more than one

updating of the nodes is performed before refining, is shown. In fact the pressure and friction coefficients for these cases are very similar and so there appears to be little benefit in choosing UPS to be more than one.

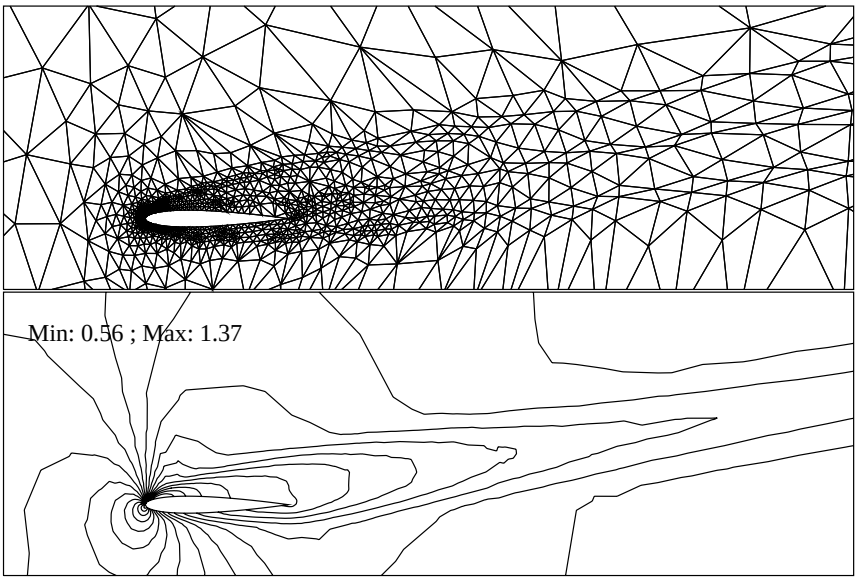

Figure 6.15: Final mesh and density contours obtained when using hr-refinement for case A2 with θ_K^r indicator.

Err Ind	MAXLEV	TOL	UPS	C_l	C_d	CPU (s)	Elts
θ_K^r	3	0.4	1	0.46	0.27	658	3693
θ_K^r	3	0.4	2	0.45	0.27	843	4492
θ_K^r	3	0.4	5	0.45	0.27	2116	5117
θ_K^r	3	0.2	1	0.44	0.26	1877	7655
θ_K^r	3	0.3	1	0.45	0.27	1239	4991
θ_K^d	3	0.2	1	0.46	0.27	740	4043
θ_K^g	3	0.7	1	0.53	0.28	2635	5235

Table 6.7: Results on hr-refined meshes for case A2.

For the system of Burgers' equations above, the value of TOL, the refinement tolerance, makes only small difference to solutions (see table 6.5), but as table 6.7 shows, changing TOL in this case has a large effect on the size of the final mesh and the solution time. However the pressure and friction coefficients for the three values of TOL shown are again very similar.

The two other error indicators also considered are θ_K^d (which includes diffusive terms) and θ_K^g (based on the density gradient). Different values of TOL are used to obtain meshes of similar size, and the pressure and friction coefficients are plotted

in figures 6.16 and 6.17. The final meshes around the aerofoil in each case is shown in figure 6.18.

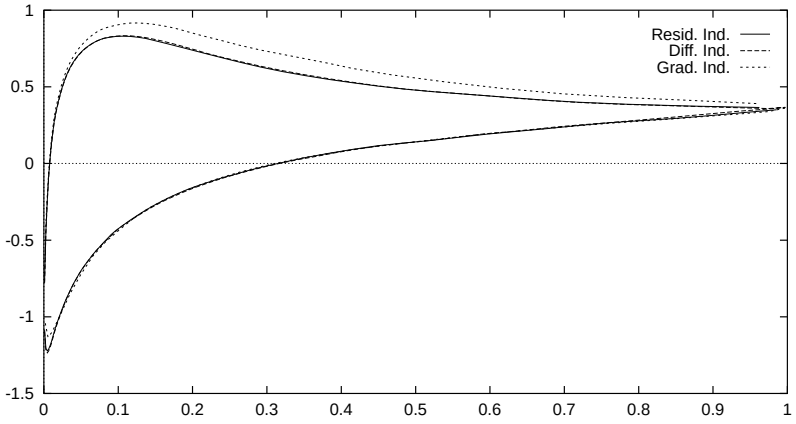

Figure 6.16: Pressure coefficients for case A2 using hr-refinement for different error indicators.

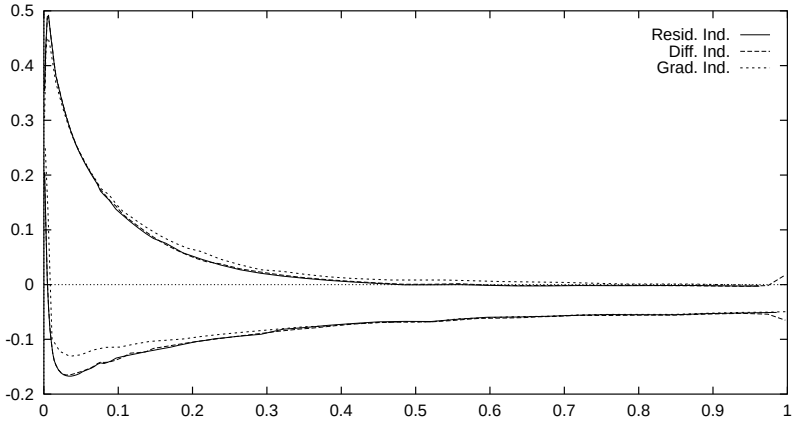

Figure 6.17: Friction coefficients for case A2 using hr-refinement for different indicators.

All the error indicators we use detect the wake to some extent and the node movement automatically move nodes towards this region. Near the aerofoil, the gradient indicator appears to pull nodes away from the wall boundary, and so is less accurate than the other indicators. The diffusive effects incorporated in θ_K^d make only a small difference compared to θ_K^r.

When choosing the numerical parameters, the value which is probably most significant in determining the accuracy and solution time is MAXLEV, which here is equal to 3 in order to compare solutions with those previously obtained. A

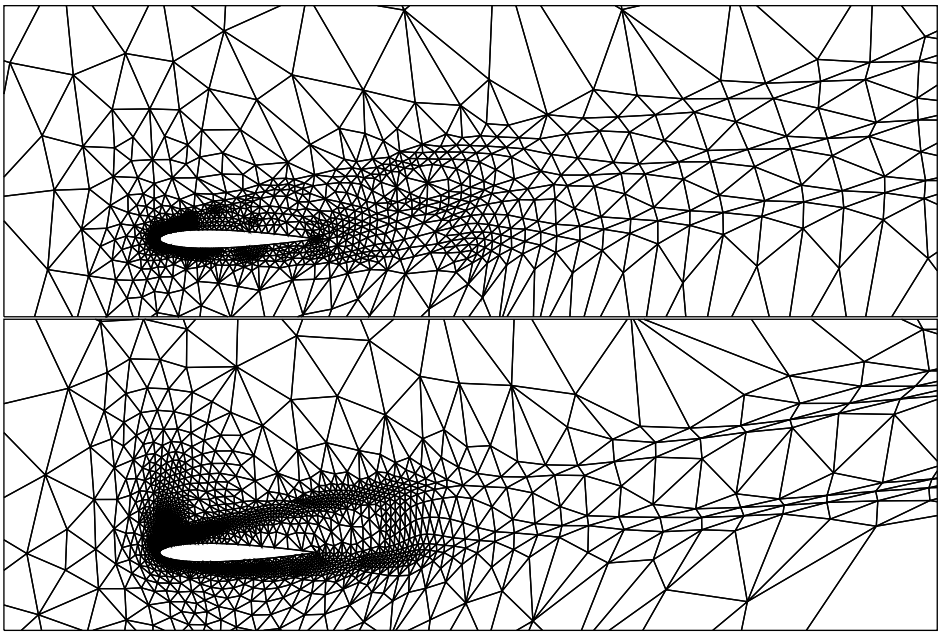

Figure 6.18: Final meshes for case A2 with (a) θ_K^d and (b) θ_K^g.

comparison of hr-refinement with previous results obtained is made at the end of this section.

Case A3

This test case contains a bow shock which we wish to resolve using fewer elements than when h-refinement is used. The two indicators θ_K^r and θ_K^g are employed to determine where to refine and as weighting functions in the node updating. In this case, the solution time and mesh quality are particularly sensitive to the choice of TOL and MAXLEV, and the values chosen (for comparison with the results obtained previously) are MAXLEV=3 and TOL=0.6 (with θ_K^r) or TOL=0.5 (with θ_K^g). Table 6.8 shows the solution coefficients, times and mesh sizes.

Err. Ind.	C_l	C_d	CPU time	Final elts
θ_K^r	0.33	0.45	1018	4660
θ_K^g	0.33	0.43	912	4801

Table 6.8: Test case A3: results on hr-refined meshes.

The final meshes in each case are shown in figure 6.19, and the density contours plotted in figure 6.20. In both cases nodes are clustered along the shock, even away from the aerofoil, enabling it to be clearly resolved. However the elements in the

centre of this shock region are not being stretched so that they become aligned with the flow, perhaps because there are not enough updates being done before the refinement procedure stops.

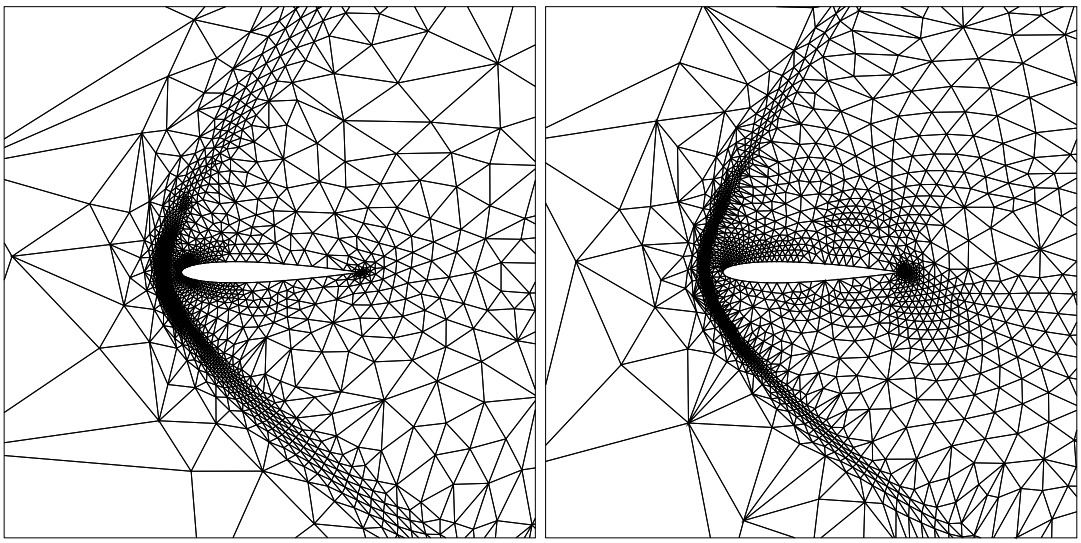

Figure 6.19: Final meshes for case A3 with (a) θ_K^r and (b) θ_K^g.

Case A6

For this case, where the Reynolds number is 2000, the error indicator based on the convective residual (θ_K^r) is used, with MAXLEV=4 and TOL=0.6. The final mesh obtained with these values consists of 5223 elements and the solution time is 1398 seconds. The solution and the final mesh are shown in figures 6.21 and 6.22.

The mesh is refined in such a way as to clearly resolve the wake, within which elements are beginning to be stretched so that they become more aligned with the direction of the wake.

Comparison of GAMM Test Cases

Any comparison of the solutions obtained using h, r and hr refinement cannot be precisely made since we are unable to quantify the error in a solution. However, by looking at the coefficient values plotted around the aerofoil and by inspection of the mesh and solution values, some idea of the relative quality of the solutions may be obtained. For the three GAMM testcases considered, table 6.9 summarises the C_d and C_l values obtained by using the different types of refinement. Plots of the pressure and friction coefficients for these examples are shown in figures 6.23–6.28.

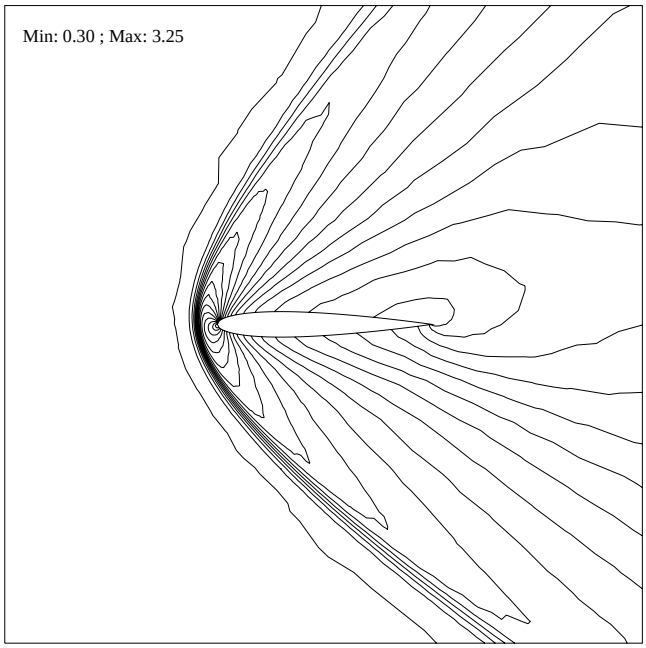

Figure 6.20: Density contours for case A3 with θ_K^r.

The cases given in table 6.9 have been chosen so that the hr-refined mesh contains fewer elements (and takes less time as a result of this) than the other meshes, but there does not appear to be a deterioration in the quality of the solution. All the coefficient plots show similar results for the different refinement techniques, and all are within the range of results obtained in [18]. Where there are differences (most obviously in figure 6.28), it is the fixed mesh solution which seems to be the most inaccurate.

The major difference between meshes obtained using h-refinement and hr-refinement is that flow features such as shocks or wakes are being detected more accurately with hr-refinement. Far from the aerofoil where elements are initially coarse, there is some element stretching, whereas in the initially fine regions, elements retain their initial shape.

Flow over a flat plate

We have already considered Carter's problem [25] on fixed meshes and h-refined meshes, and now present some results using the combined hr-refinement method of this chapter. The initial mesh used consists of 98 elements (and is shown in figure 5.14) and the boundary conditions for the problem are defined in §4.6.5.

When using the indicator θ_K^r with TOL=0.05 and MAXLEV=5, full convergence

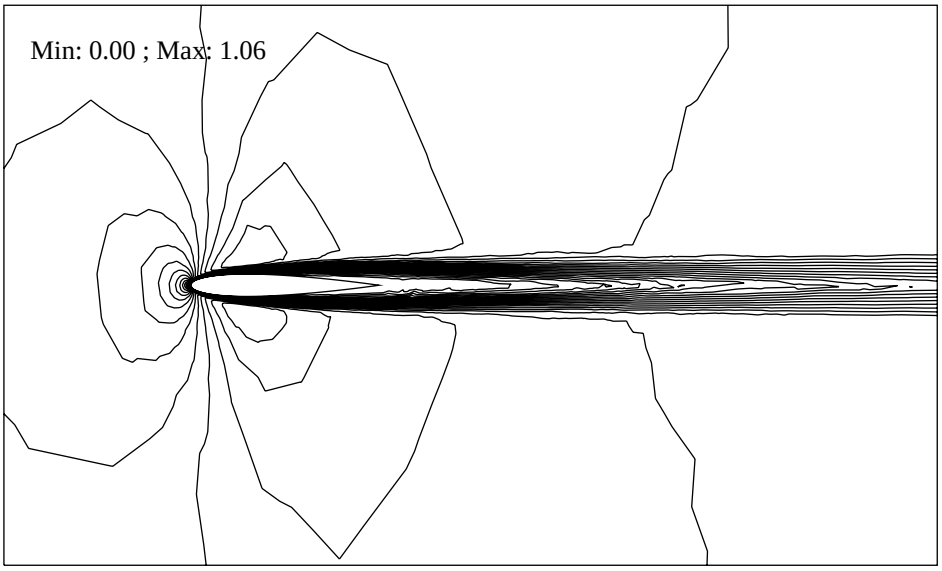

Figure 6.21: A6: Mach contours of solution on *hr*-refined mesh.

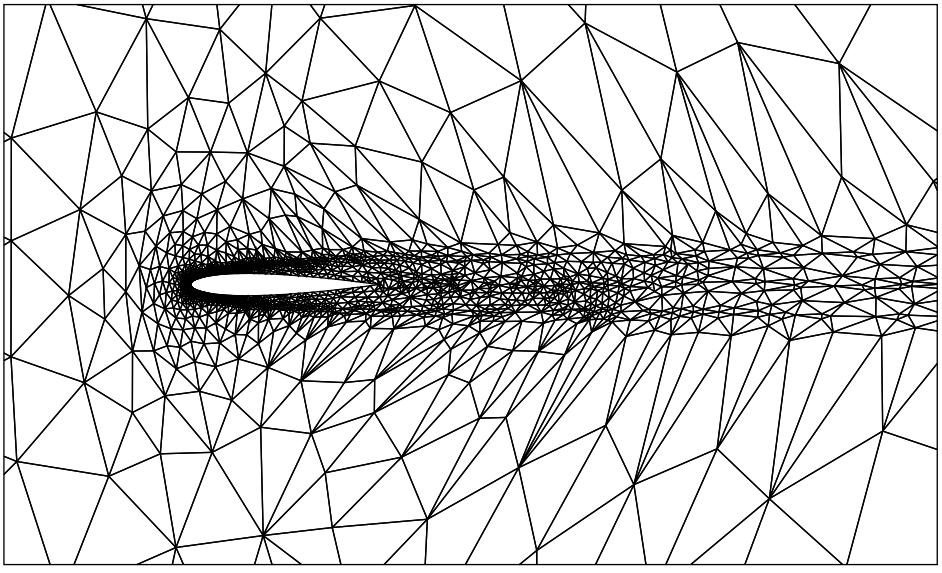

Figure 6.22: A6: Final mesh using *hr*-refinement.

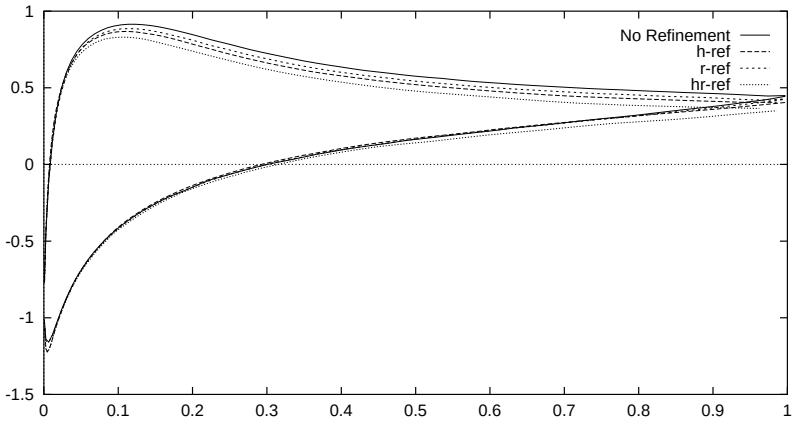

Figure 6.23: Pressure coefficients for case A2.

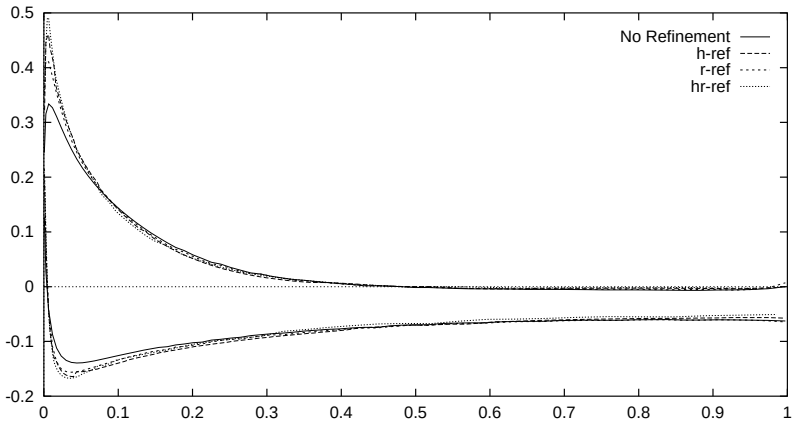

Figure 6.24: Friction coefficients for case A2.

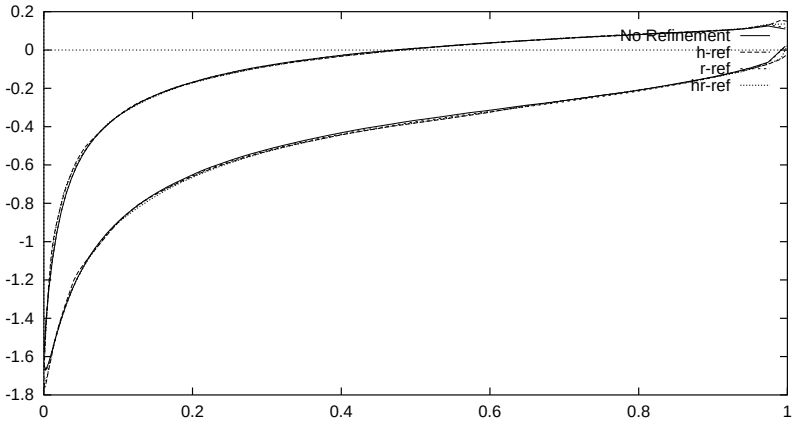

Figure 6.25: Pressure coefficients for case A3.

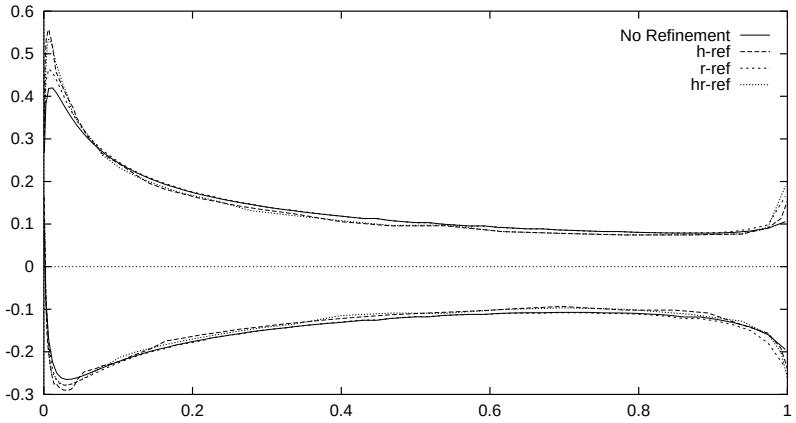

Figure 6.26: Friction coefficients for case A3.

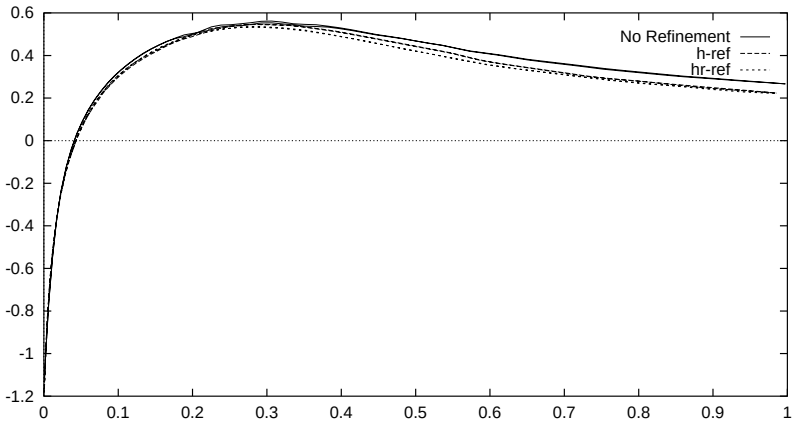

Figure 6.27: Pressure coefficients for case A6.

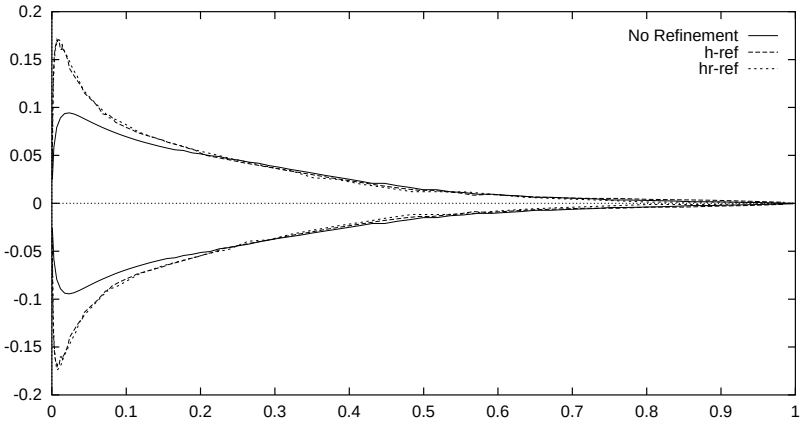

Figure 6.28: Friction coefficients for case A6.

Case	Refinement	C_l	C_d	CPU time (s)	Elements
A2	Fixed	0.52	0.28	780	5436
	h	0.47	0.27	1212	6468
	r	0.49	0.28	1133	5436
	hr	0.46	0.27	658	3693
	Range in [18]	0.41-0.52	0.24-0.29		
A3	Fixed	0.32	0.46	1266	5436
	h	0.33	0.45	1368	8360
	r	0.32	0.47	1485	5436
	hr	0.33	0.45	1018	4660
	Range in [18]	0.31-0.40	0.41-0.49		
A6	Fixed	0.003	0.12	685	5436
	h	0.002	0.12	2211	8928
	hr	0.00	0.12	1398	5223
	Range in [18]	0	0.10-0.14		

Table 6.9: Results on refined meshes for test cases A2, A3 and A6.

is reached in 151 seconds, and the final mesh of 2278 elements is shown in figure 6.29. Nodes are moved towards the shock, and to a lesser extent, towards the boundary layer. Table 6.10 summarizes the results on fixed, h-refined and hr-refined meshes, with the corresponding plots of pressure and friction coefficients shown in figures 6.31 and 6.32. We also show the additional result when MAXLEV (the maximum level of refinement) is increased from 5 to 7. Clearly there is a large saving in computational time when using h or hr-refinement, although if MAXLEV isn't large enough, the pressure and friction coefficients are inaccurate when compared to [107] or [35] for example. However, increasing MAXLEV does improve the accuracy (as we would expect).

	CPU time	Final elts
Fixed	2462	25088
h	146	2087
hr	151	2278
hr/MAXLEV=7	408	3987

Table 6.10: Results on hr-refined meshes for flat plate flow.

If the gradient indicator θ_K^q is used instead of θ_K^r, then the mesh becomes quickly distorted and convergence fails completely. The mesh in this case is shown in figure 6.30.

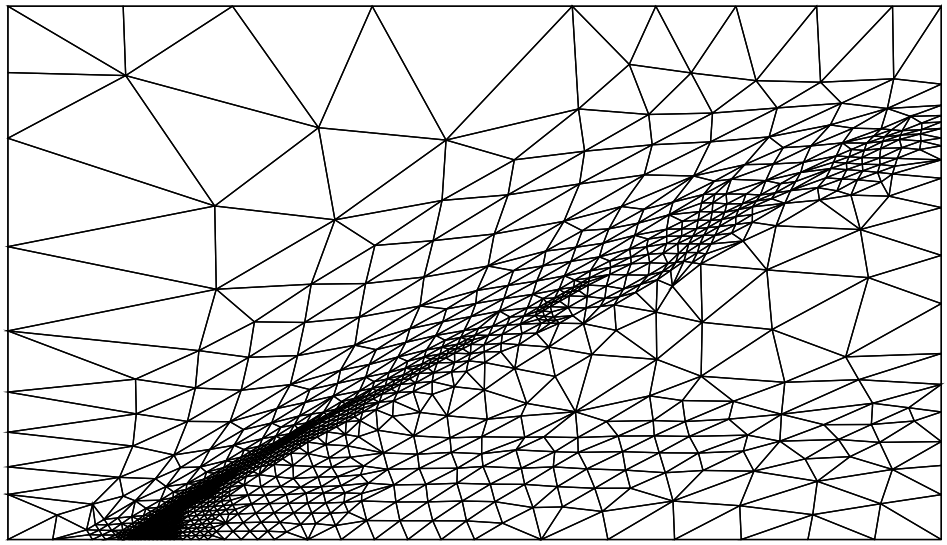

Figure 6.29: Flat plate flow: final mesh using θ_K^r indicator.

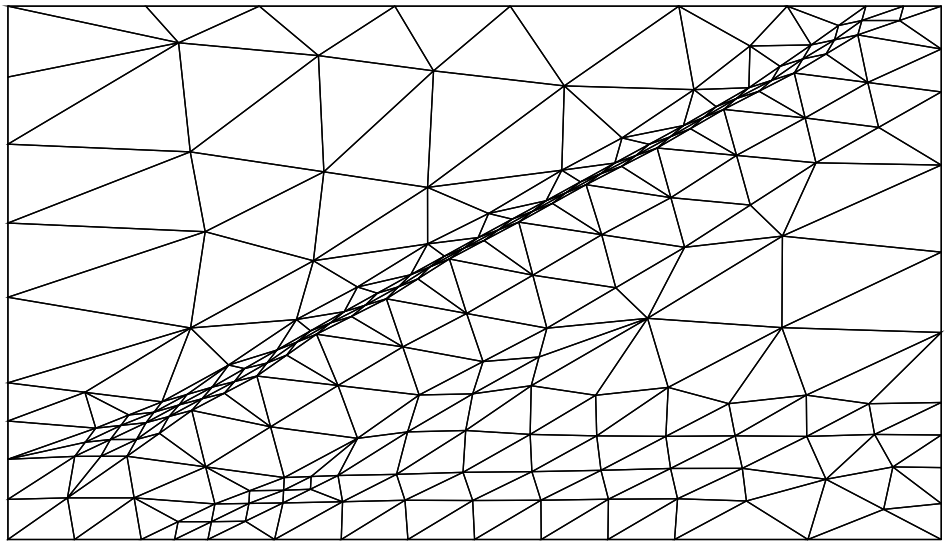

Figure 6.30: Flat plate flow: final mesh using θ_K^g indicator.

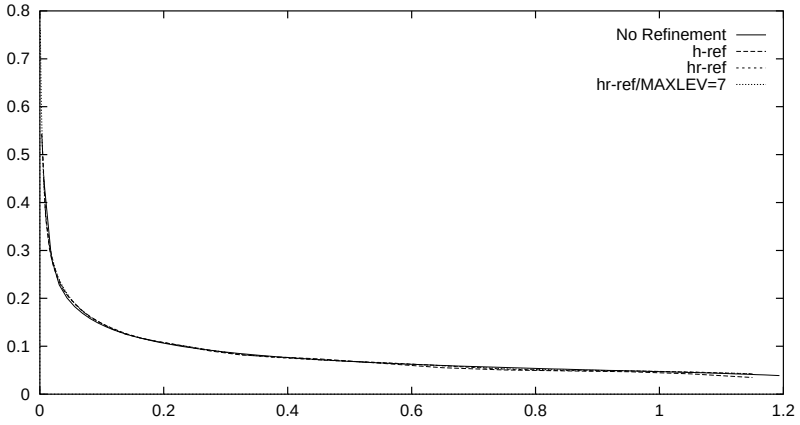

Figure 6.31: Pressure coefficients for flat plate flow problem.

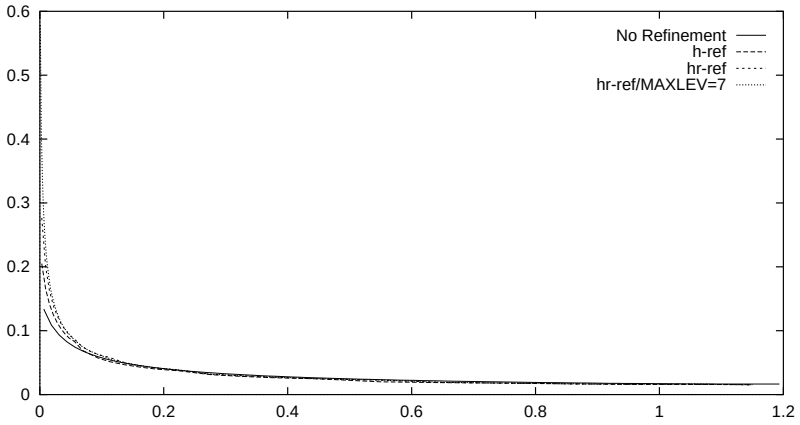

Figure 6.32: Friction coefficients for flat plate flow problem.

6.7 Summary

We have introduced in this chapter a r-refinement technique based on the method described in [64]. By using only the local solution values, each node is repositioned according to a weighted average of neighbouring element centroids. This offers certain advantages over other mesh movement methods, by being inexpensive to compute and avoiding mesh tangling.

If this technique is used on its own, as described in §6.4, then the mesh is refined so that nodes are moved towards regions where the error is large. This improves the quality of the solution, at the expense of extra computational cost as the rate of convergence drops while the node relocation is still being carried out.

An alternative to using the node movement technique by itself is to combine it with h-refinement, and this approach is described in §6.5. It involves a simple modification of the h-refinement algorithm of the previous chapter, so that a node

update step is performed prior to each h-refinement step. The results presented in §6.6 indicate that solutions can be obtained more efficiently with this new hr-refinement method, because fewer nodes are needed. This is shown quantitatively for the system of Burgers' equations, but for the Navier-Stokes equations, where exact solutions are not known, it is less easy to make precise comparisons. For the examples we have looked at, the solutions obtained more efficiently using hr-refinement appear to be no worse than those generated using h(or r)-refinement. In addition features such as wakes and shocks away from the aerofoils have been resolved more accurately, with some stretching of elements, which allows them to be more aligned with the flow.

Another advantage of allowing nodes to move is that the geometry of the final mesh is uncoupled from the initial coarse mesh, so that by inspection the mesh is of better quality. The reduction in convergence rate noticed when using r-refinement only is not such a significant problem when h-refinement is also being carried out, since the addition of nodes slows convergence more dramatically than any node movement which may be performed.

Several error indicators, used as both an h-refinement criterion and a weighting function in the node updating, have been tried, and the most versatile appears to be the convective residual indicator θ_K^r. There are problems with the gradient indicator when moving nodes towards shocks, leading to highly distorted elements. Incorporation of the diffusive terms of the residual (θ_K^d) makes very little difference for the particular problems considered here.

Although this combined approach seems to offer some potential for improving the efficiency of the solution process, there remain some issues still to be resolved. As in the h-refinement method, the best choice of parameters (for a desired level of accuracy) varies widely for different types of problems, and in some cases it is only by experiment that sensible values can be obtained. We mention above that some element stretching occurs within flow features, but only in regions where the initial mesh is coarse. Further gains in efficiency might be possible if more elements are allowed to be stretched (so that fewer nodes are needed). This might be achieved by altering the error indicator, or no longer using the indicator as the weighting function. Finally, as the Reynolds number is increased, the boundary layer becomes thinner, and further refinement is needed along the aerofoil. This is another region where element stretching would be beneficial, but experiments show that the error

indicators we have considered here do not seem to allow this to happen. Alternatives include directional indicators, perhaps based upon an idea discussed by Weatherill in [126] of detecting and tracking streamlines in the flow.

Chapter 7

Time-Dependent Problems

7.1 Introduction

In preceding chapters we sought steady solutions to flow problems consisting mainly of transonic flows at moderate Reynolds numbers ($Re < 2000$). However in other flow regimes, for example where the Reynolds number is larger than this, a steady laminar solution to the Navier-Stokes equations may not exist. One approach, considered in §8.2, is to work with the time-averaged Navier-Stokes equations and incorporate a turbulence model into the system, so that a form of steady solution for flows at high Reynolds number may be found. Alternatively, the full Navier-Stokes equations may be solved directly, and in this case, the solution is unsteady and a numerical method is needed which is accurate in both time and space in order to capture the transient solution as it evolves. For flows where the Reynolds number is too high, this method is currently prohibitively expensive, but at lower Reynolds numbers, transient problems can be accurately resolved. In this chapter we discuss some techniques which may be suitable for this type of problem.

We have already considered the use of time-stepping to march towards steady solutions in §2.6, where an implicit backward Euler scheme was used to approximate the time derivative. If global (rather than local) time-steps are used, then it is possible to obtain a time-dependent solution, but such a method is only first order accurate in time and offers no control over the size of the temporal error (other than through the choice of the constant time-step). Hence a more sophisticated algorithm is needed, and some possible approaches are described in §7.2. These include the use of space-time finite elements, developed by Hughes [66] and Johnson

[77], Taylor-Galerkin methods [38] and the method of lines.

The method of lines is a general method which reduces a system of p.d.e.'s to a system of ordinary differential equations (o.d.e.'s), by only discretizing in space in the first instance. The spatial and temporal discretization are thus independent, allowing a variety of spatial discretizations (e.g. finite elements or finite volume) to be used with any standard o.d.e. solver. We attempt to follow this approach to obtain transient solutions using the the finite element methods presented in chapter 2.

In §7.3, we briefly outline some methods for the solution of o.d.e.'s, mentioning in particular the software package SPRINT [13] which includes several algorithms for time integration, such as the theta method and the Gear method, and contains built-in temporal error control.

We use the method of lines in §7.4 to obtain a solution for an example of transient flow around an aerofoil. The use of SPRINT allows adaptivity in time, and we would also like to incorporate the h-refinement methods of chapter 5, to allow both temporal and spatial error control. One extra complication faced when refining meshes for unsteady problems is the need for derefinement—the removal of nodes from meshes. This requires a more complex data structure than refinement only, but allows unsteady problems to be solved more efficiently.

When refinement is carried out between time-steps, the solution on the old mesh is interpolated onto the new mesh, and the error introduced by this interpolation affects the local error estimate thus significantly reducing the size of the current time-step, slowing down the solution process. In §7.5, this is discussed in detail, including a theoretical analysis of a simple one-dimensional problem and numerical results for a two-dimensional example, which indicate that more accurate interpolation will overcome this problem. We then return to the Navier-Stokes equations and suggest how the interpolation might be improved in this case.

7.2 Time Accurate Finite Element Methods

In this section we outline a number of methods which use finite elements in space, and approximate time-dependent solutions accurately.

7.2.1 Space-time Finite Elements

This approach uses finite element methods to approximate the unknown variables both spatially and temporally. The usual way of doing this is the discontinuous Galerkin method, introduced by Johnson *et al.* [78], where the variable is discretized in time by a polynomial which is discontinuous at discrete time levels. Much work on convection-dominated flows has been done using a combination of a stabilized finite element method (e.g. streamline diffusion or Galerkin least-squares) and the discontinuous Galerkin method (see for example Johnson *et al.* [78], Shakib and Hughes [106] and Hughes *et al.* [66]))

We now briefly discuss the discontinuous method for the compressible Navier-Stokes equations, as described in [108]. For a given time T, the time interval $I = [0, T]$ over which the equations are to be solved, I may be partitioned into $0 = t_0 < t_1 < \ldots < t_N = T$, so that $I_n = [t_n, t_{n+1}]$ is the nth time interval. If Ω denotes the spatial domain, and Γ its spatial boundary then we can define space-time "slabs" $Q_n = \Omega \times I_n$ with boundary $P_n = \Gamma \times I_n$. For each Q_n, the spatial domain is subdivided into $(n_{el})_n$ elements , $\Omega_n^e, e = 1, \ldots, (n_{el})_n$, so that

$$Q_n^e = \Omega_n^e \times I_n, \quad e = 1, \ldots, (n_{el})_n. \tag{7.1}$$

The test and trial functions are continuous within each "slab" Q_n, but discontinuous across the interface of between Q_n and Q_{n+1}, i.e. at times $t_1, t_2, \ldots, t_{N-1}$. Because of these discontinuities we define, for a time-dependent vector function $\mathbf{V}$,

$$\mathbf{V}(t_n^{\pm}) = \lim_{\epsilon \to 0^{\pm}} \mathbf{V}(t_n + \epsilon). \tag{7.2}$$

The Galerkin least-squares method, described in chapter 2, modifies the standard Galerkin method by the addition of a least squares operator. As given in [108], this method is extended for a complete space-time discretization so that the variational formulation becomes: within each Q_n, $n = 0, \ldots, N-1$, find $\mathbf{U}^h \in \mathcal{U}^h$ such that for all $\mathbf{V}^h \in \mathcal{V}^h$ the following is satisfied:

$$\begin{aligned}
&\int_{Q_n} (-\mathbf{V}^h_{,t}.A_0\mathbf{U}^h - \mathbf{V}^h.A_i\mathbf{U}^h_{,i} + \mathbf{V}^h_{,i}.K_{ij}\mathbf{U}^h_{,j} - \mathcal{F}.\mathbf{V}^h)dQ \\
&\quad + \int_{\Omega} (\mathbf{V}^h(t_{n+1}^-).A_0\mathbf{U}^h(t_{n+1}^-) - \mathbf{V}^h(t_n^+).A_0\mathbf{U}^h(t_n^+))d\Omega \\
&+ \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} (\mathcal{L}\mathbf{V}^h.\tau\mathcal{L}\mathbf{U}^h)dQ - \int_{P_n} \mathbf{V}^h.(K_{ij}\mathbf{U}^h_{,j})n_i dP = 0,
\end{aligned} \tag{7.3}$$

for suitably defined test and trial functions $\mathcal{V}^h$ and $\mathcal{U}^h$, and where $\mathcal{L}$ and τ are defined as in chapter 2. The second integral is the term introduced by the extra time discretization, by which the flow information is propagated from one space-time "slab" to the next. At each time level, a nonlinear system is solved. If the approximation functions are constant or linear in time, then it can be shown [108] that the method is unconditionally stable. In [108] it is advocated that the constant in time approach is used for steady problems, whereas the additional accuracy gained by using linear in time approximations makes the higher order method suitable for transient problems.

7.2.2 Taylor-Galerkin Methods

Rather than use finite elements in both time and space, it is possible to use a different technique to approximate in time, whilst retaining the spatial finite element approximation. This can be done either by discretizing in space first, and then in time (the method of lines outlined below) or carrying out the time discretization prior to the spatial discretization.

One example of this second approach is the Taylor-Galerkin method, first used by Donea [38], where the time derivative is approximated by a Taylor expansion, and then rewritten in terms of spatial derivatives. The resulting equations can then be discretized in space by a Galerkin method. The method has been used by Demkowicz *et al.* to solve both the Euler [36] and the Navier-Stokes equations [35] adaptively and by Tworzydlo *et al.* [117].

For example, if $\mathbf{U}$ is a solution vector for a system of equations, then a Taylor series expansion leads to

$$\mathbf{U}(t+\Delta t)-\alpha\frac{\Delta t^2}{2}\mathbf{U}_{,tt}(t+\Delta t) = \mathbf{U}(t)+\Delta t\mathbf{U}_{,t}(t)+(1-\alpha)\frac{\Delta t^2}{2}\mathbf{U}(t)_{,tt}+O(\Delta t^3). \quad (7.4)$$

The value $\alpha \in [0,1]$ is an implicitness parameter, so that $\alpha = 1$ corresponds to a fully implicit algorithm, and $\alpha = 0$ leads to an explicit algorithm. The time derivative in the equations can be rewritten in terms of spatial derivatives, and then substituted into (7.4) leading to a set of equations containing only spatial terms at each time-step.

7.2.3 Method of Lines

In this approach, the p.d.e.'s, which consist of several variables, are reduced to a system of o.d.e.'s, of just one independent variable. Because the spatial and temporal unknowns are approximated separately, the method allows a particular spatial discretization technique to be used with any of the wide range of o.d.e. solvers which have been developed (see §7.3.1). For example, use of the stabilized Galerkin method given in §2.5 for the Navier-Stokes equations can lead to a system of o.d.e.'s. The steady problem is a nonlinear system of equations which may be written as

$$\mathbf{G}(\mathbf{W}) = \mathbf{0} \tag{7.5}$$

where $\mathbf{W}$ is the vector of all the unknown coefficients and $\mathbf{G}$ is the nonlinear function representing the variational equations (this is shown in more detail in §2.3 for the Galerkin method). The unsteady equations introduce a time derivative so the system becomes

$$M(\mathbf{W})\frac{\partial \mathbf{W}}{\partial t} + \mathbf{G}(\mathbf{W}) = (0) \tag{7.6}$$

where the dependence of the matrix M on $\mathbf{W}$ is due to the presence of the least squares operator. The solution of this system of o.d.e.'s is the subject of the following section.

7.3 Temporal Discretization

When the method of lines is used, the system of p.d.e.'s is semi-discretized to reduce it to a system of o.d.e.'s such as (7.6). We now consider some methods to solve this resulting initial value problem. The particular software package used in this work is SPRINT, discussed in §7.3.2.

7.3.1 Solution of O.D.E.'s

The theory and implementation of methods for solving systems of o.d.e.'s is well developed and there are a wide range of techniques available (see for example [86] for an overview of the subject). In this section we briefly summarize some methods which are available within SPRINT, the software package which we are using as the o.d.e. solver.

Once a system of o.d.e.'s has been obtained from spatial discretization of the original p.d.e.'s, it may be written as follows,

$$\frac{\partial \mathbf{y}}{\partial t} = \mathbf{g}(\mathbf{y}, t), \quad \mathbf{y}(0) = \mathbf{k}, \tag{7.7}$$

for some function $\mathbf{g}$ and vector $\mathbf{k}$ (note that for convenience this is a simpler system than that given in (7.6)). The general approach to solving such problems is to compute estimates of $\mathbf{y}$ ($\mathbf{y}_1, \mathbf{y}_2, .., \mathbf{y}_n, ..$, say) at discrete time levels, $t_1, t_2, .., t_n, ..$, using previous values of $\mathbf{y}$. The simplest method for solving (7.7) is the forward Euler method,

$$\mathbf{y}_n = \mathbf{y}_{n-1} + \delta \mathbf{g}(\mathbf{y}_{n-1}, t_{n-1}) \tag{7.8}$$

where δ is the time-step size. For stability, this method usually requires very small time-steps to be taken, but this problem may be overcome by using the implicit backward Euler method (used in §2.6 to march towards steady state) which is stable for any size of time-step:

$$\mathbf{y}_n = \mathbf{y}_{n-1} + \delta \mathbf{g}(\mathbf{y}_n, t_n). \tag{7.9}$$

The stiffness of an o.d.e. is a property which we do not define here (see [86, chapter 6]) but stiff o.d.e.'s require a suitable o.d.e. solver, and in particular an implicit method is required to avoid very small time-steps. A survey of methods for solving stiff o.d.e.'s is given by Byrne and Hindmarsh in [23]. One method which is provided by SPRINT and may be used for stiff problems is the theta method, defined as

$$\mathbf{y}_n = \mathbf{y}_{n-1} + (1-\theta)\delta \mathbf{g}(\mathbf{y}_{n-1}, t_{n-1}) + \theta \delta \mathbf{g}(\mathbf{y}_n, t_n). \tag{7.10}$$

with $0 \leq \theta \leq 1$. When $\theta < 0.5$, the method is conditionally stable, otherwise it is stable, and the optimal value of θ depends on the particular problem. If $\theta = 0.5$, then the method is second order rather than first order accurate, but this is not always the best choice. We use the default value within SPRINT of $\theta = 0.55$, although Berzins and Furzeland have developed an adaptive theta method which automatically selects the value of θ [14].

Multistep methods for solving o.d.e.'s use more than one of the previous values of $\mathbf{y}$ to calculate the current value, and a common technique when dealing with stiff problems is to use the backward differentiation formulae (b.d.f.) of Gear. Here the current value $\mathbf{y}_n$ is calculated using

$$\sum_{j=0}^{k} \alpha_{k-j} \mathbf{y}_{n-j} = \delta \beta_k \mathbf{g}(\mathbf{y}_n, t_n). \tag{7.11}$$

For $k = 1, \ldots, 6$, the coefficients α_{k-j} and β_k are given in [86]. This is also available within SPRINT, and if $k = 1$, then the method reverts to backward Euler.

Other numerical schemes for the solution of o.d.e.'s include explicit and implicit Runge-Kutta methods, which are single step but have a more complicated structure than linear multistep methods, and Adams methods, which are multistep methods suitable for nonstiff problems. Further details are given in [86].

For any of the implicit methods mentioned above, a typical approach to computing $\mathbf{y}$ at the latest time-step involves two stages. First, a predictor step uses an explicit method to give an approximate value of $\mathbf{y}$ cheaply. This is then used as an initial guess for the corrector step, in which the implicit method is used to obtain a more accurate approximation of $\mathbf{y}$.

In order to ensure that the numerical solution is accurate in time, an adaptive strategy is adopted. This usually means that the local error is estimated in some way, and the size of the time-step and/or the order of the method is allowed to change accordingly.

7.3.2 SPRINT

The software package SPRINT (Software for PRoblems IN Time) has been developed by Berzins and Furzeland [13] for the numerical solution of problems that involve mixed systems of time-dependent algebraic, ordinary and partial differential equations. It has been designed in a modular way so that individual modules can be replaced by the user's own routines.

It can solve systems of o.d.e.'s of the following form (which is more general than (7.7) above)

$$A(\mathbf{y}, t)\frac{\partial \mathbf{y}}{\partial t} = \mathbf{g}(\mathbf{y}, t), \quad \mathbf{y}(0) = \mathbf{k}, \tag{7.12}$$

for a given matrix A (which may be singular, although the matrix M in (7.6) is positive definite), function $\mathbf{g}$ and vector $\mathbf{k}$. A number of techniques for integrating in time are included in SPRINT, such as the theta method, b.d.f. and Adams methods. SPRINT also has the ability to control the size of the temporal error, by adapting the size of the time-step and the order of the method, as the solution progresses, in order to keep the error estimate within a user specified tolerance.

In order to solve the nonlinear system of equations arising at each time-step, either functional iteration or Newton's method may be used. Functional iteration is

computationally cheaper, but may be insufficient when dealing with stiff problems, in which case Newton's method, which requires access to the Jacobian matrix, is more suitable. Within the theta method an optional switch has been implemented, allowing the code to change between these two nonlinear solvers according to the degree of stiffness. At each iteration of Newton's method, the linear systems are solved using one of a number of iterative methods provided.

We now describe the use of SPRINT with the finite element method of chapter 2 to solve unsteady flow problems.

7.4 The Method of Lines for Unsteady Flow

In this section, we consider an example of unsteady flow and show how solutions may be obtained using the method of lines, in which the Navier-Stokes equations are first semi-discretized using a stabilized finite element method, leading to a system of o.d.e.'s which are solved within SPRINT. We also wish to use the spatial h-refinement techniques of chapter 5, extending them to enable derefinement as well as refinement.

The design of SPRINT is such that using it in conjunction with a spatial discretization technique is straightforward. Once initialization has been performed (e.g. setting up of the mesh details), SPRINT requires a user defined subroutine to interface with the spatial discretization. This routine provides the following residual

$$\mathbf{r}(\mathbf{y}, \dot{\mathbf{y}}) = A(\mathbf{y}, t)\dot{\mathbf{y}} - \mathbf{g}(\mathbf{y}, t), \tag{7.13}$$

which is obtained from semi-discretizing the p.d.e.'s. In our case this is done by Galerkin least-squares, the stabilized finite element method discussed in chapter 2, which leads to the system (7.6).

We use the theta method for time integration, with $\theta = 0.55$, as recommended in [13], and a tolerance of 10^{-3} when estimating the local errors in time, which should ensure that spatial rather than temporal errors dominate the solution. At the end of each time-step, SPRINT allows a monitoring routine to be called, and we use this to carry out any refinement of the mesh. In the current example, refinement is only performed every 0.5 units of time. The approach for refining is similar to that used in steady problems—an error indicator is computed for every element and any elements in which the indicator exceeds the specified tolerance are refined. As

with the steady case, values at the new nodes are obtained by linear interpolation. Following the refinement step, SPRINT attempts to continue integration using the same step size and order as before.

Since we are considering unsteady flows, flow features are likely to change position, and so derefinement of the mesh (i.e. removal of nodes) is desirable. Derefinement has been implemented in the algorithm, developed by Jimack [76], which we use for h-refinement (see §5.3). Elements to be derefined are those for which the value of the error indicator is below a derefinement tolerance. The tree-like data structure employed to store the mesh simplifies the extra complexity of the derefinement process, which arises because of the need to keep the mesh conforming. Before being derefined, an element must be checked that it can be derefined without leaving a nonconforming mesh. The refinement and derefinement tolerances used in the example below are 0.05 and 0.025 respectively, with a maximum level of refinement (MAXLEV) of 4. Note that these are absolute tolerances rather than relative values dependent on the maximum error which we used for steady problems in chapter 5. The error indicator is a time-dependent version of θ_K^r (see §5.2), where the time derivative is included in the convective residual term.

The example we consider is subsonic flow ($M_\infty = 0.55$) around the NACA0012 aerofoil. The angle of attack α is 8.34°, which ensures that the flow is unsteady when the Reynolds number Re is 5000. This case has previously been considered elsewhere, e.g. [111]. The initial mesh consists of 1617 elements (and is shown in figure 4.2), and we allow the code to solve 5000 time-steps, which corresponds to a final time reached of 13.8 units. Figures 7.1–7.4 show the meshes and Mach number contours of the solution as it evolves. The final mesh shown contains 5989 elements, and the contour plots clearly show the unsteadiness of the flow.

The refined meshes suffer from the same problem as meshes obtained using h-refinement for steady flows, which is a lack of smoothness and a clear dependence on the original mesh structure. For steady problems, we used r-refinement to overcome this poor quality, but the inclusion of derefinement into the algorithm means that r-refinement cannot be used in the same way. This is because derefined elements may be deformed due to the node movement, a problem which is discussed further in §8.3.

Figure 7.5 plots the value of the time variable against the number of time-steps. Immediately after the steps in which mesh refinement occurs, the time-step

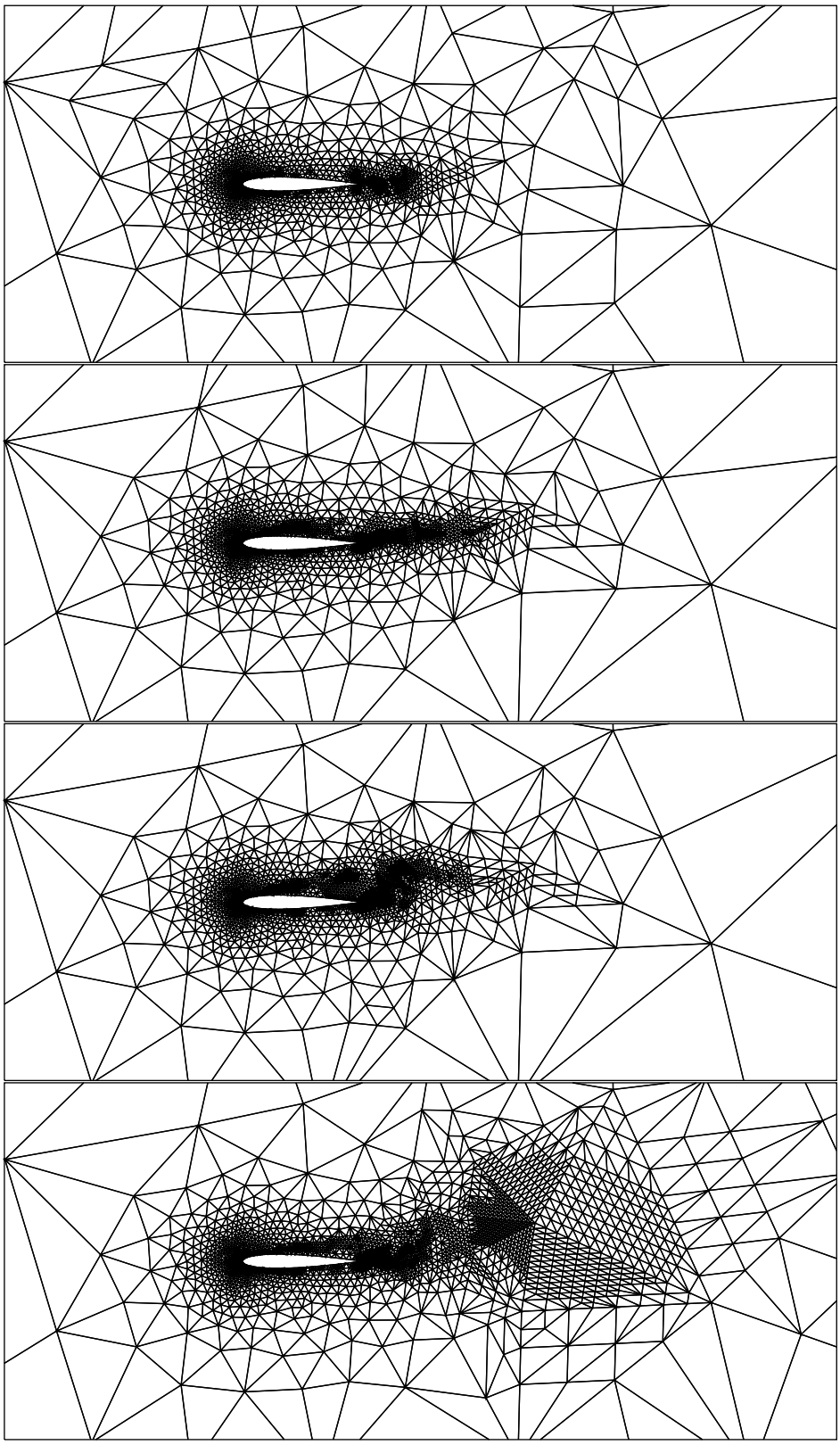

Figure 7.1: Unsteady flow: meshes at t=1, t=3, t=5, t=7.

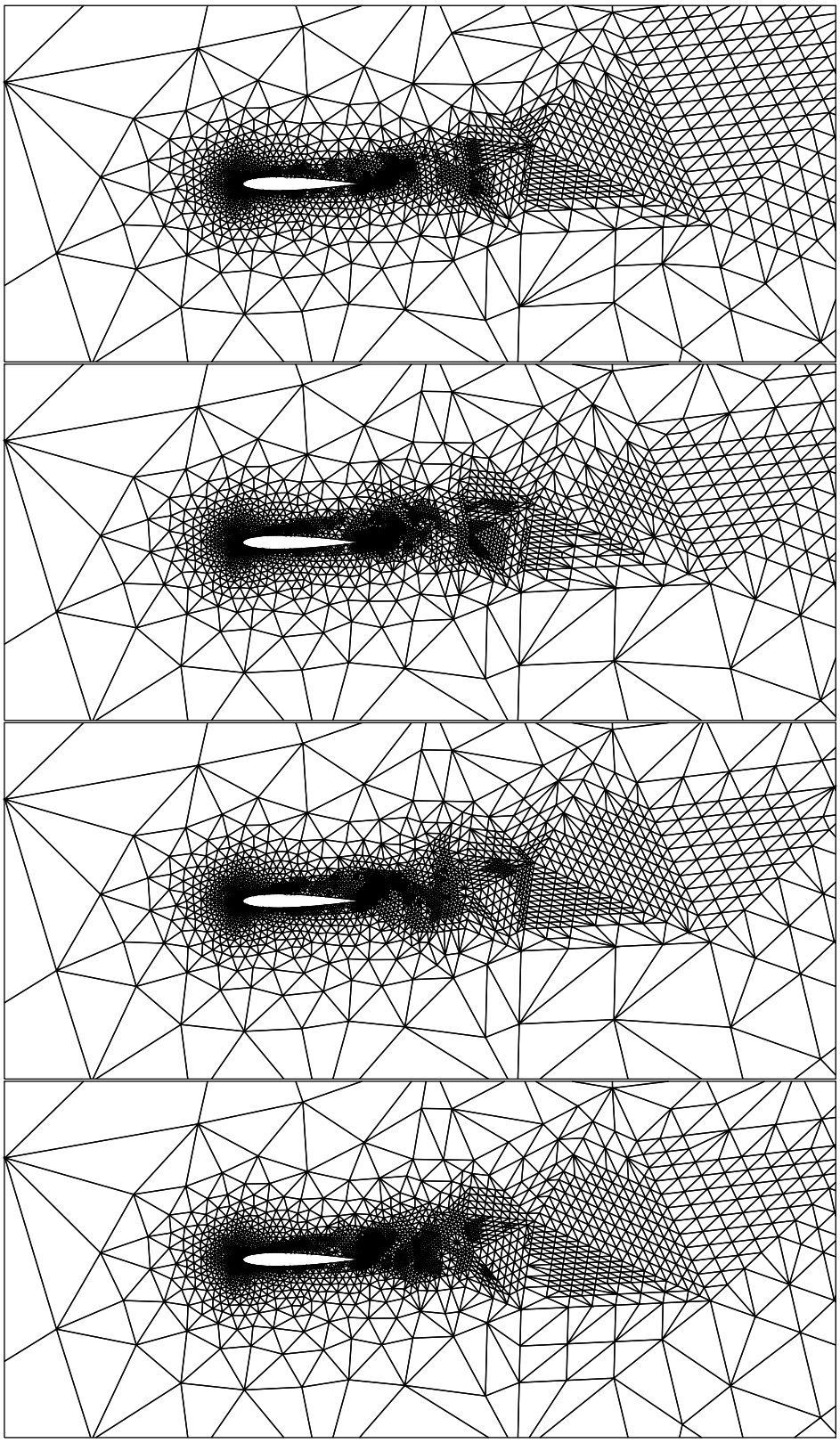

Figure 7.2: Unsteady flow: meshes at t=9, t=10.5, t=12, t=13.5.

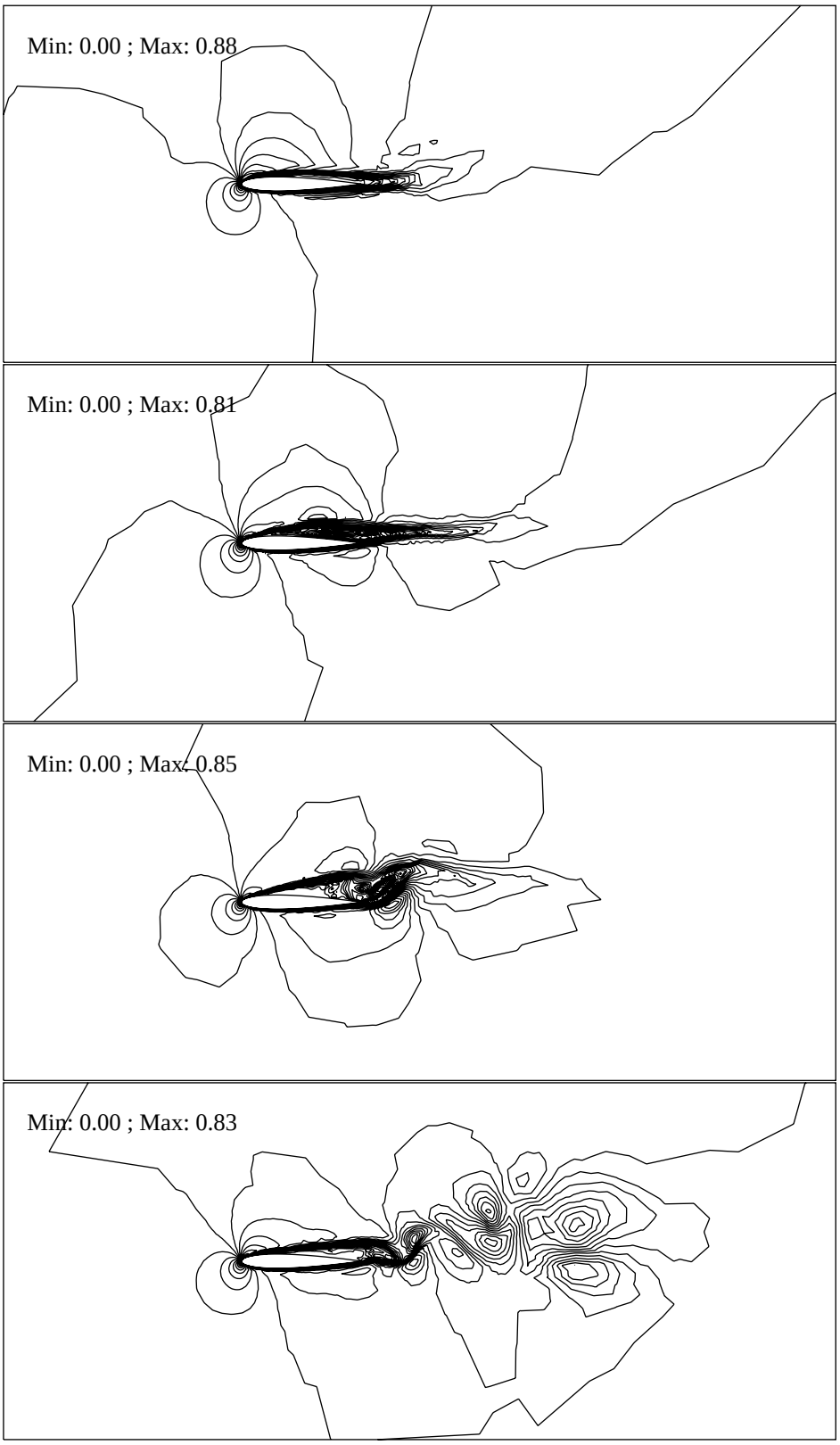

Figure 7.3: Unsteady flow: Mach contours of solution at t=1, t=3, t=5, t=7.

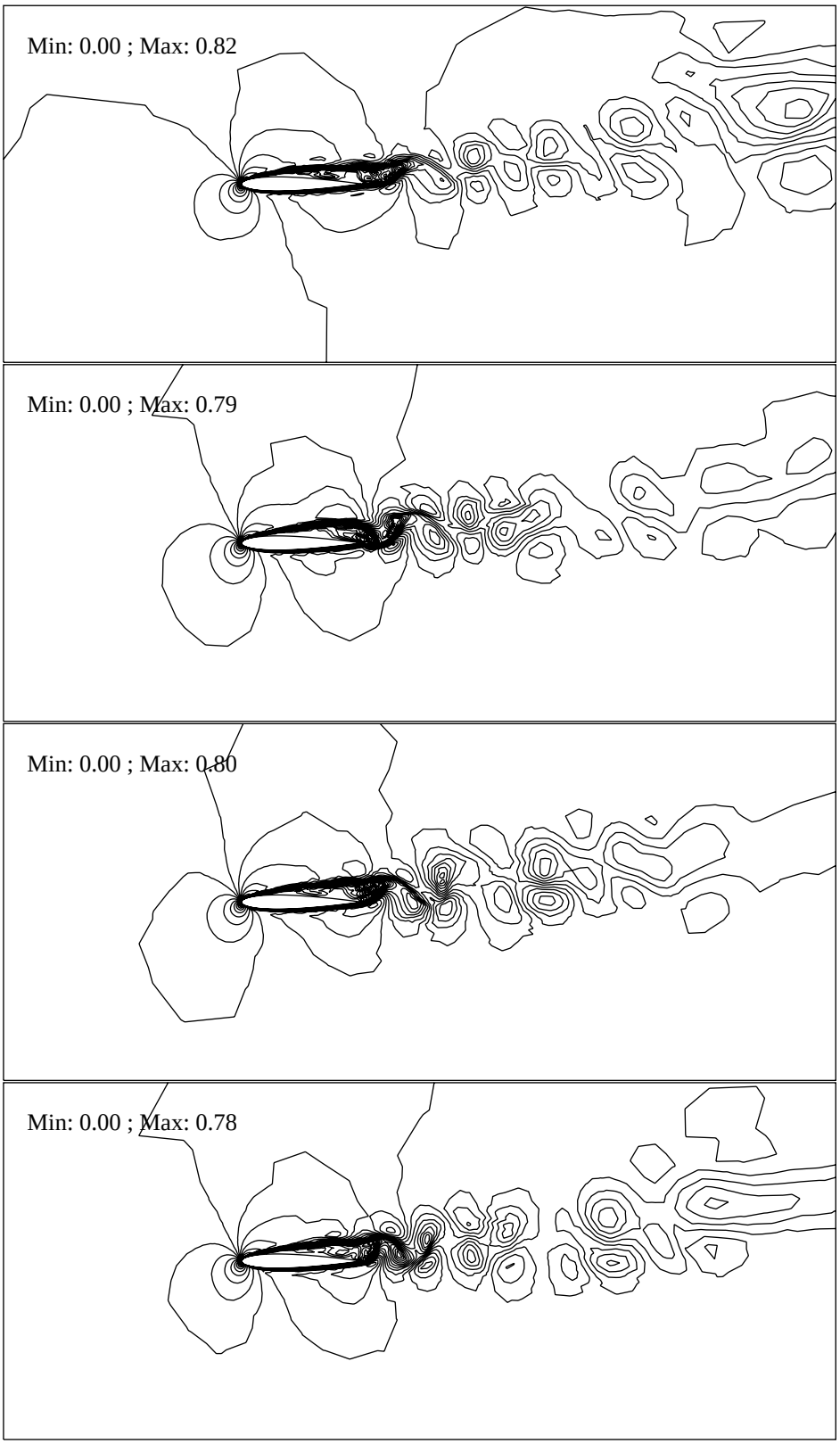

Figure 7.4: Unsteady flow: Mach contours of solution at t=9, t=10.5, t=12, t=13.5.

is reduced by a factor of between 100 and 1000, as can be seen from figure 7.5. This significantly slows down the solution time, which is about 16 hours, and in the remainder of the chapter we attempt to show why this drop in the time-step occurs and how it might be avoided.

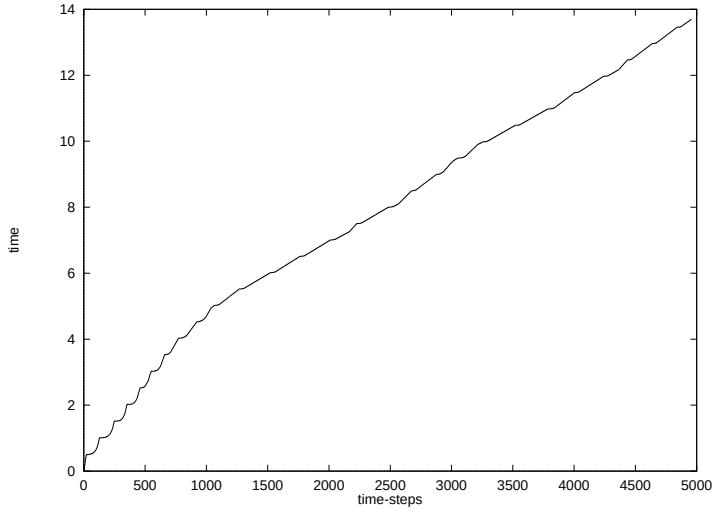

Figure 7.5: Plot of time against time-steps.

An immediate gain in the solution time might be found by using a more efficient linear solver than the general linear algebra package that SPRINT uses within Newton's method (for example, the approach we used for steady problems of GMRES with ILU preconditioning).

An indication as to why the time-step reduces so much after spatial refinement may be obtained from examination of the norm of the residual vector (calculated from (7.13)) before and after refinement. This indicates that there is a substantial increase in its size after interpolation onto the refined mesh. It would seem reasonable that this increase in the residual is causing the local error estimate to jump dramatically, thus leading to the sharp reduction in time-step size. It therefore follows that, if we can refine the mesh so that the increase in residual size is restricted, the size of the time-step will not fall so dramatically. We discuss in further detail this relationship between residual and time-step size in §7.5.3, but first consider alternatives to using linear interpolation in order to obtain values at the new nodes, and observe the effect of these alternatives on the residual.

7.5 Interpolation on Refined Meshes

We have seen in the previous section that the large drop in the size of the time-step immediately after a mesh refinement step coincides with a sharp increase in size of the residual vector, and have suggested why the two may be connected. In this section we show that when new nodal values are obtained using a better quality interpolant than linear, then there is not such a large increase in the norm of the residual vector (and there is a correspondingly smaller drop in the size of the time-step). This is shown theoretically for a simple one-dimensional example, and then numerically for a problem in two-dimensions on both structured and unstructured meshes. Finally, the relationship between this increase in the residual vector and the decrease in time-step size is studied more carefully for the simple case of a backward Euler scheme.

7.5.1 A One Dimensional Convection-Diffusion Problem

In order to observe the effect upon the o.d.e. residual of interpolating a solution from the old mesh onto the new mesh, we first consider a one dimensional linear convection-diffusion problem on a uniform mesh. This mesh is refined and the new residual vectors obtained using both linear and higher order interpolants are compared.

The equation

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} - \epsilon \frac{\partial^2 u}{\partial x^2} = 0 \tag{7.14}$$

for $0 \leq x \leq 1$, $u(0,t) = 0$, $u(1,t) = 1$, initial conditions $u(x,0) = x$ and where ϵ is a constant, has a steady-state solution

$$u(x,t) = \frac{e^{\frac{x}{\epsilon}} - 1}{e^{\frac{1}{\epsilon}} - 1}. \tag{7.15}$$

The Galerkin finite element method for obtaining a spatial discretization of this problem at each time-step can be stated in the following way. We first define a trial function space for u,

$$\mathcal{U} = \{u : u \in C(H^1([0,1]), [0,\infty)), u(0,t) = 0, u(1,t) = 1\}. \tag{7.16}$$

The weak form of (7.14) is: find $u \in \mathcal{U}$ such that

$$(u_t, v) + (u_x, v) + \epsilon(u_x, v_x) = 0 \quad \forall v \in \mathcal{U}_0, \tag{7.17}$$

where $\mathcal{U}_0 = \{v : v \in C(H^1([0,1]),[0,\infty)), v(0,t) = v(1,t) = 0\}$ and $(u,v) = \int_0^1 uvdx$.

The interval [0,1] is partitioned into n equally spaced sub-intervals, with node points at $0 = x_0 < x_1 < \ldots < x_{n-1} < x_n = 1$. The length of each sub-interval is $\hat{h}$. If u and u_t are approximated in space by piecewise linear functions u^h and u_t^h then the space of trial functions is given by

$$\mathcal{U}^h = \{u^h : u^h \in C(C^0(0,1),[0,\infty)), u^h|_I \in P_1, u^h(0,t) = 0, u^h(1,t) = 1\}, \quad (7.18)$$

and the space of test functions is

$$\mathcal{V}^h = \{v^h : v^h \in C(C^0(0,1),[0,\infty)), v^h|_I \in P_1, v^h(0,t) = v^h(1,t) = 0\} \quad (7.19)$$

where I denotes each sub-interval and P_1 is the space of linear polynomials. The discrete variational formulation is: find $u^h \in \mathcal{U}^h$ such that

$$(u_t^h, v^h) + (u_x^h, v^h) + \epsilon(u_x^h, v_x^h) = 0 \quad \forall v^h \in \mathcal{V}^h. \quad (7.20)$$

Define ϕ_i $(i = 0 \ldots n)$ as the usual piecewise linear basis functions so that $\phi_i(x_j) = \delta_{ij}$, and write u^h and u_t^h as

$$u^h = \sum_{i=0}^{n} a_i(t)\phi_i, \;\; u_t^h = \sum_{i=0}^{n} b_i(t)\phi_i \quad (7.21)$$

where $\mathbf{a} = (a_1, a_2, \ldots, a_{n-1})^T$ and $\mathbf{b} = (b_1, b_2, \ldots, b_{n-1})^T$ are the unknown coefficients to be found at each time-step and $\mathbf{b}(t) = \dot{\mathbf{a}}(t)$ in the exact temporal solution. As boundary conditions, we define $b_0 = b_n = 0$, $a_0 = 0$ and $a_n = 1$. The vector $\mathbf{b}$ will be determined by current and previous values of $\mathbf{a}$, depending on the time integration scheme being used.

A linear system of $n-1$ equations is obtained from (7.20) by choosing $v^h = \phi_i$ for $i = 1, 2, \ldots, n-1$:

$$\left(\sum_{j=1}^{n-1} b_j\phi_j, \phi_i\right) + \left(\sum_{j=1}^{n-1} a_j\phi_{j,x}, \phi_i\right) + \epsilon\left(\sum_{j=1}^{n-1} a_j\phi_{j,x}, \phi_{i,x}\right) + \hat{c}_i = 0 \quad (7.22)$$

where $\hat{c}_i$ is a component of the boundary vector $\hat{\mathbf{c}}$ determined by the Dirichlet boundary conditions. In this case

$$\hat{\mathbf{c}} = (0, 0, \ldots, 0, (\phi_{n,x}, \phi_{n-1}) + \epsilon(\phi_{n,x}, \phi_{n-1,x}))^T. \quad (7.23)$$

Now define the matrices $\hat{M}$, $\hat{D}$ and $\hat{K}$, where $\hat{m}_{ij} = (\phi_i, \phi_j)$, $\hat{d}_{ij} = (\phi_i, \phi_{j,x})$ and $\hat{k}_{ij} = (\phi_{i,x}, \phi_{j,x})$:

$$\hat{M} = \begin{pmatrix} \frac{2\hat{h}}{3} & \frac{\hat{h}}{6} & 0 & \dots & 0 & 0 & 0 \\ \frac{\hat{h}}{6} & \frac{2\hat{h}}{3} & \frac{\hat{h}}{6} & \dots & 0 & 0 & 0 \\ 0 & \frac{\hat{h}}{6} & \frac{2\hat{h}}{3} & \dots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \dots & \frac{\hat{h}}{6} & \frac{2\hat{h}}{3} & \frac{\hat{h}}{6} \\ 0 & 0 & 0 & \dots & 0 & \frac{\hat{h}}{6} & \frac{2\hat{h}}{3} \end{pmatrix}, \tag{7.24}$$

$$\hat{D} = \begin{pmatrix} 0 & \frac{1}{2} & 0 & \dots & 0 & 0 & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & \dots & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 & \dots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \dots & -\frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \dots & 0 & -\frac{1}{2} & 0 \end{pmatrix}, \tag{7.25}$$

$$\hat{K} = \begin{pmatrix} \frac{2}{\hat{h}} & -\frac{1}{\hat{h}} & 0 & \dots & 0 & 0 & 0 \\ -\frac{1}{\hat{h}} & \frac{2}{\hat{h}} & -\frac{1}{\hat{h}} & \dots & 0 & 0 & 0 \\ 0 & -\frac{1}{\hat{h}} & \frac{2}{\hat{h}} & \dots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \dots & -\frac{1}{\hat{h}} & \frac{2}{\hat{h}} & -\frac{1}{\hat{h}} \\ 0 & 0 & 0 & \dots & 0 & -\frac{1}{\hat{h}} & \frac{2}{\hat{h}} \end{pmatrix}. \tag{7.26}$$

The linear system can be written as

$$\hat{M}\mathbf{b} + \hat{D}\mathbf{a} + \epsilon\hat{K}\mathbf{a} + \hat{\mathbf{c}} = \mathbf{0} \tag{7.27}$$

which needs to be solved at each time-step until a final time has been reached. This system defines the residual function $\hat{\mathbf{R}}$:

$$\hat{\mathbf{R}}(\mathbf{a}, \mathbf{b}) = \hat{M}\mathbf{b} + \hat{D}\mathbf{a} + \epsilon\hat{K}\mathbf{a} + \hat{\mathbf{c}}. \tag{7.28}$$

The mesh is now refined by dividing each sub-interval in two, so that there are now $2n$ sub-intervals each of length h $(= \hat{h}/2)$. Given an interpolation operator $I_h : \mathbf{R}^{n-1} \to \mathbf{R}^{2n-1}$ which acts upon a $n-1$ length vector to generate interpolated values at the new node points, we define the residual vector on the new mesh as

$$\mathbf{R}(I_h(\mathbf{a}), I_h(\mathbf{b})) = MI_h(\mathbf{b}) + DI_h(\mathbf{a}) + \epsilon KI_h(\mathbf{a}) + \mathbf{c} \tag{7.29}$$

where the $(2n-1)\times(2n-1)$ matrices M, D and K are similar to the $(n-1)\times(n-1)$ matrices $\hat{M}$,$\hat{D}$ and $\hat{K}$, with h replacing $\hat{h}$. We now define three types of interpolation operators.

Linear Interpolation

Define $I_h^L : \mathbf{R}^{n-1} \to \mathbf{R}^{2n-1}$ as

$$I_h^L(\mathbf{y}) = \begin{pmatrix} \frac{1}{2}(y_0 + y_1) \\ \vdots \\ y_i \\ \frac{1}{2}(y_i + y_{i+1}) \\ y_{i+1} \\ \vdots \\ \frac{1}{2}(y_{n-1} + y_n) \end{pmatrix}, \tag{7.30}$$

where $\mathbf{y} = (y_1, \ldots, y_{n-1})^T$ and y_0 and y_n are specified Dirichlet boundary conditions.

Hermite Interpolation

The interpolant $S(x)$ is constructed using a piecewise cubic polynomial whose first derivative is continuous at each node. On each sub-interval $[x_i, x_{i+1}]$

$$S(x) = y_i H_i(x) + y_{i+1} H_{i+1}(x) + y_i' K_i(x) + y_{i+1}' K_{i+1}(x) \tag{7.31}$$

where the cardinal functions are defined in terms of the piecewise linear basis functions ϕ_i and ϕ_{i+1} as follows:

$$\begin{aligned} H_i &= \phi_i^2(3 - 2\phi_i) && (7.32)\\ H_{i+1} &= \phi_{i+1}^2(3 - 2\phi_{i+1}) && (7.33)\\ K_i &= \phi_i^2(x - x_i) && (7.34)\\ K_{i+1} &= \phi_{i+1}^2(x - x_{i+1}). && (7.35) \end{aligned}$$

These take the following values at x_i and x_{i+1}:

$$\begin{array}{llll} H_i(x_i) = 1 & K_i(x_i) = 0 & H_{i+1}(x_i) = 0 & K_{i+1}(x_i) = 0 \\ H_i'(x_i) = 0 & K_i'(x_i) = 1 & H_{i+1}'(x_i) = 0 & K_{i+1}'(x_i) = 0 \\ H_i(x_{i+1}) = 0 & K_i(x_{i+1}) = 0 & H_{i+1}(x_{i+1}) = 1 & K_{i+1}(x_{i+1}) = 0 \\ H_i'(x_{i+1}) = 0 & K_i'(x_{i+1}) = 0 & H_{i+1}'(x_{i+1}) = 0 & K_{i+1}'(x_{i+1}) = 1 \end{array}$$

and at the midpoint $(x_i + \frac{\hat{h}}{2})$ (i.e. the new node point)

$$H_i(x_i + \tfrac{\hat{h}}{2}) = \tfrac{1}{2} \quad K_i(x_i + \tfrac{\hat{h}}{2}) = \tfrac{\hat{h}}{8} \quad H_{i+1}(x_i + \tfrac{\hat{h}}{2}) = \tfrac{1}{2} \quad K_{i+1}(x_i + \tfrac{\hat{h}}{2}) = -\tfrac{\hat{h}}{8}. \tag{7.36}$$

The approximations y_i' and y_{i+1}' to the derivatives are computed using second order differencing

$$y_i' = \frac{y_{i+1} - y_{i-1}}{2\hat{h}}, \quad i = 1, \ldots, n-1 \tag{7.37}$$

and at the boundaries,

$$y_0' = \frac{4y_1 - 3y_0 - y_2}{2\hat{h}}, \quad y_n' = \frac{-4y_{n-1} + 3y_n + y_{n-2}}{2\hat{h}}. \tag{7.38}$$

Hence from (7.31), and (7.36)–(7.38)

$$S(x_i + \frac{\hat{h}}{2}) = \frac{9y_i}{16} + \frac{9y_{i+1}}{16} - \frac{y_{i-1}}{16} - \frac{y_{i+2}}{16}, \quad i = 1, .., n-2 \tag{7.39}$$

$$S(x_0 + \frac{\hat{h}}{2}) = \frac{3y_0}{8} + \frac{3y_1}{4} - \frac{y_2}{8} \tag{7.40}$$

$$S(x_{n-1} + \frac{\hat{h}}{2}) = \frac{3y_{n-1}}{4} + \frac{3y_n}{8} - \frac{y_{n-2}}{8}. \tag{7.41}$$

So the interpolant is:

$$I_h^H(\mathbf{y}) = \begin{pmatrix} \frac{1}{8}(3y_0 + 6y_1 - y_2) \\ \vdots \\ y_i \\ \frac{1}{16}(9y_i + 9y_{i+1} - y_{i-1} - y_{i+2}) \\ y_{i+1} \\ \vdots \\ \frac{1}{8}(6y_{n-1} + 3y_n - y_{n-2}) \end{pmatrix}. \tag{7.42}$$

Cubic Spline Interpolation

Again, a piecewise cubic polynomial is constructed but now both the first and second derivatives of $S(x)$ are required to be continuous at the nodes. Instead of (7.31) we have

$$S(x) = y_i H_i(x) + y_{i+1} H_{i+1}(x) + m_i K_i(x) + m_{i+1} K_{i+1}(x), \tag{7.43}$$

where the terms $m_0, m_1, \ldots, m_n$ are obtained from the solution of the tridiagonal system

$$\begin{pmatrix} 2 & 4 & 0 & \ldots & 0 & 0 & 0 \\ \frac{1}{2} & 2 & \frac{1}{2} & \ldots & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 2 & \ldots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \ldots & \frac{1}{2} & 2 & \frac{1}{2} \\ 0 & 0 & 0 & \ldots & 0 & 4 & 2 \end{pmatrix} \begin{pmatrix} m_0 \\ \vdots \\ m_n \end{pmatrix} = \begin{pmatrix} \frac{1}{2\hat{h}}(4y_1 - 5y_0 + y_2) \\ \vdots \\ \frac{3}{2\hat{h}}(y_{i+1} - y_{i-1}) \\ \vdots \\ -\frac{1}{2\hat{h}}(4y_{n-1} + y_{n-2} - 5y_n) \end{pmatrix}. \quad (7.44)$$

See Lancaster and Salkauskas [87, chapter 4] for details on how this system, which leads to clamped splines, is obtained. Using (7.36), at the new node points,

$$S(x_i + \frac{\hat{h}}{2}) = \frac{1}{2}(y_i + y_{i+1}) + \frac{\hat{h}}{8}(m_i - m_{i+1}), \quad (7.45)$$

and the interpolant $I_h^C : \mathbf{R}^{n-1} \to \mathbf{R}^{2n-1}$ is defined as

$$I_h^C(\mathbf{y}) = \begin{pmatrix} \frac{1}{2}(y_0 + y_1) + \frac{\hat{h}}{8}(m_0 - m_1) \\ \vdots \\ y_i \\ \frac{1}{2}(y_i + y_{i+1}) + \frac{\hat{h}}{8}(m_i - m_{i+1}) \\ y_{i+1} \\ \vdots \\ \frac{1}{2}(y_{n-1} + y_n) + \frac{\hat{h}}{8}(m_{n-1} - m_n) \end{pmatrix}. \quad (7.46)$$

Theoretical Comparison of Interpolants

We wish to compare the residual given by (7.29) using the three interpolants defined above with the residual vector on the original mesh. In order to make a direct nodal comparison, we linearly interpolate the old mesh residual $\hat{\mathbf{R}}(\mathbf{a}, \mathbf{b})$:

$$I_h^L(\hat{\mathbf{R}}(\mathbf{a}, \mathbf{b})) = I_h^L(\hat{M}\mathbf{b}) + I_h^L(\hat{D}\mathbf{a}) + I_h^L(\hat{K}\mathbf{a}) + I_h^L(\hat{\mathbf{c}}). \quad (7.47)$$

For each node (new and old) the four expressions (the nodal component of the interpolated residual and residual components of the interpolated solution vectors for the three interpolants I_h^L, I_h^H and I_h^C) can be compared. In the following analysis, the three matrices M, D and K are considered separately.

Mass Matrix M

Given the vector $\mathbf{b}$ with boundary nodes $b_0 = b_n = 0$ we obtain the following expressions (the two terms shown in each vector refer to the old node point at x_i and the new point at $x_{i+\frac{1}{2}} = x_i + h$ respectively).

$$I_h^L(\hat{M}\mathbf{b}) = \begin{pmatrix} \vdots \\ \frac{\hat{h}}{6}b_{i-1} + \frac{2\hat{h}}{3}b_i + \frac{\hat{h}}{6}b_{i+1} \\ \frac{\hat{h}}{12}b_{i-1} + \frac{5\hat{h}}{12}b_i + \frac{5\hat{h}}{12}b_{i+1} + \frac{\hat{h}}{12}b_{i+2} \\ \vdots \end{pmatrix}, \tag{7.48}$$

$$MI_h^L(\mathbf{b}) = \begin{pmatrix} \vdots \\ \frac{h}{12}b_{i-1} + \frac{5h}{6}b_i + \frac{h}{12}b_{i+1} \\ \frac{h}{2}b_i + \frac{h}{2}b_{i+1} \\ \vdots \end{pmatrix}, \tag{7.49}$$

$$MI_h^H(\mathbf{b}) = \begin{pmatrix} \vdots \\ \frac{h}{12}b_{i-1} + \frac{41h}{48}b_i - \frac{h}{96}b_{i-2} + \frac{h}{12}b_{i+1} - \frac{h}{96}b_{i+2} \\ \frac{13h}{24}b_i + \frac{13h}{24}b_{i+1} - \frac{h}{24}b_{i-1} - \frac{h}{24}b_{i+2} \\ \vdots \end{pmatrix}, \tag{7.50}$$

$$MI_h^C(\mathbf{b}) = \begin{pmatrix} \vdots \\ \frac{h}{12}b_{i-1} + \frac{5h}{6}b_i + \frac{h}{12}b_{i+1} + \frac{h^2}{24}m_{i-1} - \frac{h^2}{24}m_{i+1} \\ \frac{h}{12}b_{i-1} + \frac{5h}{6}b_i + \frac{h}{12}b_{i+1} + \frac{h^2}{24}m_{i-1} - \frac{h^2}{24}m_{i+1} \\ \vdots \end{pmatrix}. \tag{7.51}$$

Since $\hat{h} = 2h$, and using a Taylor series expansion to approximate terms b_{i-2},b_{i-1},b_{i+1} and b_{i+2} in terms of b_i and the approximations b_i', b_i'' and b_i''' to the solution derivatives ($b_{i+1} = b_i + \hat{h}b_i' + \hat{h}^2 b_i''/2 + \hat{h}^3 b_i'''/6 + O(\hat{h}^4)$ for example), terms away from the boundaries can be directly compared.

	At node x_i	At new node $x_i + h$
Interpolated residual	$2hb_i + O(h^3)$	$2hb_i + 2h^2 b_i' + O(h^3)$
Linear interpolation	$hb_i + O(h^3)$	$hb_i + h^2 b_i' + O(h^3)$
Hermite interpolation	$hb_i + O(h^3)$	$hb_i + h^2 b_i' + O(h^3)$
Cubic Spline	$hb_i + \frac{h^2(m_{i-1} - m_{i+1})}{24} + O(h^3)$	$hb_i + h^2(b_i' + \frac{m_i - m_{i+1}}{6}) + O(h^3)$

For linear and Hermite interpolation, the values of the terms in the component of the new residual are half of the equivalent terms of the interpolated residual (to second order). In the case of cubic splines, an approximate expression for the terms involving m_i can be derived: we make the assumption that, as $h \to 0$, $b''_{i+\frac{1}{2}} = S''(x_i + h) + O(h^2)$, which can be shown numerically. We then write

$$b''_{i+\frac{1}{2}} = \frac{b_{i+2} - b_{i+1} - b_i + b_{i-1}}{8h^2} + O(h^2) \tag{7.52}$$

and

$$\begin{aligned} S''(x_i + h) &= y_i H''_i(x_i + h) + y_{i+1} H''_{i+1}(x_i + h) + m_i K''_i(x_i + h) \\ &\quad + m_{i+1} K''_{i+1}(x_i + h) && (7.53) \\ &= \frac{1}{2h}(-m_i + m_{i+1}). && (7.54) \end{aligned}$$

Hence using the assumption above and from (7.52) and (7.54) we have that

$$\begin{aligned} m_i - m_{i+1} &= \frac{-b_{i+2} + b_{i+1} + b_i - b_{i-1}}{4h} + O(h^3) && (7.55) \\ &= -2(hb''_i + h^2 b'''_i) + O(h^3). && (7.56) \end{aligned}$$

Replacing the $m_i - m_{i+1}$ by (7.56), the residual of the cubic spline interpolation becomes $hb_i + O(h^3)$ at x_i and $hb_i + h^2 b'_i + O(h^3)$ at $x_i + h$. Thus again the terms in each component of the residual of the interpolated solution are half those of the interpolation of the residual on the original mesh.

Convection Matrix D

Given the vector $\mathbf{a}$ with boundary nodes $a_0 = 0$ and $a_n = 1$, expressions for $I_h^L(\hat{D}\mathbf{a})$, $DI_h^L(\mathbf{a})$, $DI_h^H(\mathbf{a})$ and $DI_h^C(\mathbf{a})$ are

$$I_h^L(\hat{D}\mathbf{a}) = \begin{pmatrix} \vdots \\ \frac{1}{2}(-a_{i-1} + a_{i+1}) \\ \frac{1}{4}(-a_{i-1} - a_i + a_{i+1} + a_{i+2}) \\ \vdots \end{pmatrix}, \tag{7.57}$$

$$DI_h^L(\mathbf{a}) = \begin{pmatrix} \vdots \\ \frac{1}{4}(-a_{i-1} + a_{i+1}) \\ \frac{1}{2}(-a_i + a_{i+1}) \\ \vdots \end{pmatrix}, \tag{7.58}$$

$$DI_h^H(\mathbf{a}) = \begin{pmatrix} \vdots \\ \frac{1}{32}(a_{i-2} - 10a_{i-1} + 10a_{i+1} - a_{i+2}) \\ \frac{1}{2}(-a_i + a_{i+1}) \\ \vdots \end{pmatrix}, \tag{7.59}$$

$$DI_h^C(\mathbf{a}) = \begin{pmatrix} \vdots \\ \frac{1}{4}(-a_{i-1} + a_{i+1}) + \frac{h}{8}(-m_{i-1} + 2m_i - m_{i+1}) \\ \frac{1}{2}(-a_i + a_{i+1}) \\ \vdots \end{pmatrix}. \tag{7.60}$$

Using (7.56), we first obtain an expression for $-m_{i-1} + 2m_i - m_{i+1}$ in terms of a_i:

$$\begin{aligned} -m_{i-1} + 2m_i - m_{i+1} &= -(m_{i-1} - m_i) + (m_i - m_{i+1}) && (7.61)\\ &= \frac{1}{4h}(-a_{i+2} + 2a_{i+1} - 2a_{i-1} + a_{i-2}) + O(h^3) && (7.62)\\ &= O(h^3). && (7.63) \end{aligned}$$

The terms for each type of interpolation can now be compared as before.

	At node x_i	At new node $x_i + h$
Interpolated residual	$2ha_i' + O(h^3)$	$2ha_i' + 2h^2a_i'' + O(h^3)$
Linear interpolation	$ha_i' + O(h^3)$	$ha_i' + h^2a_i'' + O(h^3)$
Hermite interpolation	$ha_i' + O(h^3)$	$ha_i' + h^2a_i'' + O(h^3)$
Cubic Spline	$ha_i' + O(h^3)$	$ha_i' + h^2a_i'' + O(h^3)$

As for the mass matrix, the residual terms are, to second order, half the size of the terms in the interpolated residual for all three interpolants.

Stiffness Matrix K

For the vector $\mathbf{a}$, expressions for $I_h^L(\hat{K}\mathbf{a})$, $KI_h^L(\mathbf{a})$, $KI_h^H(\mathbf{a})$ and $KI_h^C(\mathbf{a})$ are

$$I_h^L(\hat{K}\mathbf{a}) = \begin{pmatrix} \vdots \\ \frac{1}{\hat{h}}(-a_{i-1} + 2a_i - a_{i+1}) \\ \frac{1}{2\hat{h}}(-a_{i-1} + a_i + a_{i+1} - a_{i+2}) \\ \vdots \end{pmatrix}, \tag{7.64}$$

$$KI_h^L(\mathbf{a}) = \begin{pmatrix} \vdots \\ \frac{1}{h}(-a_{i-1} + 2a_i - a_{i+1}) \\ 0 \\ \vdots \end{pmatrix}, \tag{7.65}$$

$$KI_h^H(\mathbf{a}) = \begin{pmatrix} \vdots \\ \frac{1}{16h}(a_{i-2} - 8a_{i-1} + 14a_i - 8a_{i+1} + a_{i+2}) \\ \frac{1}{8h}(-a_{i-1} + a_i + a_{i+1} - a_{i+2}) \\ \vdots \end{pmatrix}, \tag{7.66}$$

$$KI_h^C(\mathbf{a}) = \begin{pmatrix} \vdots \\ \frac{1}{2h}(-a_{i-1} + 2a_i - a_{i+1} - \frac{1}{4}(m_{i-1} - m_{i+1}) \\ \frac{1}{2}(m_i - m_{i+1}) \\ \vdots \end{pmatrix}. \tag{7.67}$$

As before we use (7.56) to obtain $m_{i-1} - m_{i+1}$ in terms of a_i.

$$m_{i-1} - m_{i+1} = (m_{i-1} - m_i) + (m_i - m_{i+1}) \tag{7.68}$$

$$= \frac{1}{4h}(a_{i+2} + 2a_i - a_{i-2}) + O(h^3) \tag{7.69}$$

$$= -4ha_i'' + O(h^3) \tag{7.70}$$

So for the stiffness matrix, a comparison of interpolants may also be made.

	At node x_i	At new node $x_i + h$
Interpolated residual	$-2ha_i'' + O(h^3)$	$-2(ha_i'' + h^2a_i''') + O(h^3)$
Linear interpolation	$-2ha_i'' + O(h^3)$	0
Hermite interpolation	$-ha_i'' + O(h^3)$	$-ha_i'' - h^2a_i''' + O(h^3)$
Cubic Spline	$-ha_i'' + O(h^3)$	$-ha_i'' - h^2a_i''' + O(h^3)$

The Hermite and cubic spline interpolants behave in the same way as for the mass and convection matrices. However at old nodes, the component of the residual obtained from linear interpolation is equal (to second order) to the equivalent term in the interpolated residual. At new nodes this component is zero, and so in neither case does the linear interpolant behave as it has done for the other matrices. It is this property that causes the residual to dramatically increase when linear interpolation is used and the mesh is refined. For both Hermite and cubic spline interpolation the terms from all three matrices are approximately reduced by a factor of two, for small enough h, and hence the overall residual is of the same order.

Numerical Experiments

This section describes the numerical solution of (7.14) using SPRINT, in order to see if the above analysis can be shown to hold in practice. The o.d.e solver selected in SPRINT is the backward Euler scheme, and the residual routine required by SPRINT is that defined by the residual function $\hat{\mathbf{R}}$ in (7.28). The value of ϵ used is 0.2, which is large enough to allow the standard unstabilized Galerkin method to obtain solutions free of oscillations on the meshes used.

The initial mesh consists of 32 elements, and the time-stepping is allowed to continue until $t = 1000$, by which time the solution has reached steady-state. Without any refinement of the mesh, the time-step size δ increases steadily, as would be expected for a problem with a steady-state solution.

We wish to see the effect of refining the mesh on both the residual norm and the time-step size, and so after a specified time-step, every element is sub-divided into two, solution values at the new nodes are obtained by interpolation from the old nodes and time-stepping is allowed to continue on the refined mesh. The two points at which refinement is carried out are during the early transient stage (i.e. as soon as $\delta > 0.1$) and near steady state (when $\delta > 10$).

	δ	Residual L_2 norm	$(M\mathbf{b})_{31}$	$(K\mathbf{a})_{31}$	$(D\mathbf{a})_{31}$
Before refinement	0.113	8.2e-09	-6.64e-5	-0.135	0.135
Linear	2.41e-5	190.7	-3.37e-5	-0.135	6.74e-2
Hermite	0.089	5.91	-3.39e-5	-6.98e-2	6.73e-2
Cubic Spline	0.089	0.441	-3.39e-5	-6.73e-2	6.72e-2
Before refinement	11.7	1.77e-12	-6.19e-9	-0.135	0.135
Linear	4.08e-5	191.9	-3.14e-9	-0.135	6.77e-2
Hermite	0.349	5.92	-3.16e-9	-7.91e-2	6.76e-2
Cubic Spline	0.350	0.440	-3.16e-9	-6.77e-2	6.75e-2

Table 7.1: Effect of interpolants on refining a) during transient stage and b) near steady state

For these two cases, table 7.1 shows the value of δ immediately before and after refinement when the three types of interpolation discussed above are used. The table also shows the L_2 norm of the residual vector in each example, as well as individual terms contained in the residual at the 31st node of the refined mesh (i.e. one lying near the middle of the interval).

In the first case, when the mesh is refined during the transient stage and linear

interpolation is used to form the new solution values, there is a huge reduction in the step size δ. There is also a large increase in the residual norm, which is explained by the relative changes in the separate components which make up the residual. As predicted by the analysis above, the terms produced by the mass and convection matrices both halve in value following interpolation, but the term from the stiffness matrix is unchanged (to 3 d.p.) and it is this behaviour which causes the residual norm to increase. In §7.5.3 we discuss the question of why a large increase in the size of the residual leads to a drop in the step size δ.

If higher order interpolants (Hermite or cubic spline) are used instead of a linear one, then δ only drops slightly. The separate components of the residual term at the 31st node all halve in value (to second order) so the increase in the residual norm is less than with linear interpolation. There will always be some increase due to boundary effects and the fact that there are twice as many nodes. As might be expected, the cubic spline is a better interpolant than the Hermite type, but resulting step size is the same in both cases. These results provide further evidence that if the residual can be kept relatively small following interpolation, then the time-step size will not be badly affected.

Results are similar in the second case, when the mesh is refined near to steady state. When linear interpolation is used, there is the expected sharp fall in δ, but when cubic spline or hermite interpolation are used, a less dramatic drop in δ occurs. This drop ($11.7 \rightarrow 0.35$) is more significant than that in the transient stage ($0.113 \rightarrow 0.089$) because the steady state solution on the refined mesh is different to the one on the original mesh. The residual terms all behave as predicted by the theory, as in the first case.

The analysis of the effects of interpolation for the simple one-dimensional problem (7.14) has been verified numerically by the results above, and explains the growth in the residual size following mesh refinement. We now consider more complicated problems in one and two dimensions.

Use of Galerkin Least-Squares and Nonlinear problems

If the Galerkin spatial discretization used above is insufficient (due to the mesh being too coarse for the chosen value of ϵ), then a stabilized scheme such as Galerkin least-squares (see §2.5) may be used to obtain stable solutions. For the linear problem

(7.14) the variational formulation would become: find $u^h \in \mathcal{U}^h$ such that for all $v^h \in \mathcal{U}_0^h$

$$(u_t^h, v^h + \tau v_x^h) + (u_x^h, v^h + \tau v_x^h) + \epsilon(u_x^h, v_x^h) = 0, \tag{7.71}$$

where $(u, v) = \int_0^1 uv dx$, and τ is suitably chosen. As with the Galerkin method in (7.28), we can write the above in terms of matrices as a residual function $\hat{\mathbf{R}}$:

$$\hat{\mathbf{R}}(\mathbf{a}, \mathbf{b}) = (\hat{M} + \tau \hat{D}^T)\mathbf{b} + \hat{D}\mathbf{a} + (\epsilon + \tau)\hat{K}\mathbf{a} + \hat{\mathbf{c}}, \tag{7.72}$$

(using a similar notation to (7.28)). The only new matrix which appears here is $\hat{D}^T$, which behaves in exactly the same way as $\hat{D}$ when applied to an interpolated vector, i.e. the components of $D^T I_h(\mathbf{b})$ are half the size of the interpolated vector $I_h(\hat{D}^T\mathbf{b})$. Hence the same conclusions concerning the change in the size of the residual terms apply to Galerkin least-squares as to the standard Galerkin method.

These results also follow for a nonlinear convection-diffusion problem. Consider

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \epsilon\frac{\partial^2 u}{\partial x^2} = f \tag{7.73}$$

for $0 \leq x \leq 1$, $u(0,t) = 0$, $u(1,t) = 1$, initial conditions $u(x,0) = x$ and where ϵ is a constant, and f is a source term. The nonlinear term in the residual is given by

$$\hat{N}_i(\mathbf{a}) = \int_0^1 \sum_{k=0}^{n} a_k \phi_{k,x} \sum_{j=0}^{n} a_j \phi_j \phi_i dx, \quad i = 1, \ldots n-1. \tag{7.74}$$

As for the matrices M, D and K, we can show what happens to the size of this term when different interpolation operators are applied:

$$I_h^L(\hat{N}(\mathbf{a})) = \begin{pmatrix} \vdots \\ \frac{1}{6}(-a_{i-1}^2 - a_{i-1}a_i + a_i a_{i+1} + a_{i+1}^2) \\ \frac{-a_{i-1}^2 - a_{i-1}a_i - a_i^2 + a_{i+1}^2 + a_{i+1}a_{i+2} + a_{i+2}^2}{12} \\ \vdots \end{pmatrix}, \tag{7.75}$$

$$N(I_h^L(\mathbf{a})) = \begin{pmatrix} \vdots \\ \frac{1}{24}(-a_{i-1}^2 - 4a_{i-1}a_i + 4a_i a_{i+1} + a_{i+1}^2) \\ \frac{1}{4}(-a_i^2 + a_{i+1}^2) \\ \vdots \end{pmatrix}, \tag{7.76}$$

$$N(I_h^H(\mathbf{a})) = \begin{pmatrix} \vdots \\ \frac{1}{1536}(-80a_{i-1}^2 - 340a_{i-1}a_i + 18a_{i-2}a_{i-1} \\ +34a_{i-2}a_i + 340a_ia_{i+1} - a_{i-2}^2 - 2a_{i-2}a_{i+1} \\ +80a_{i+1}^2 - 34a_ia_{i+2} - 18a_{i+1}a_{i+2} \\ +2a_{i-1}a_{i+2} + a_{i+2}^2) \\ \frac{1}{96}(-25a_i^2 + a_{i-1}a_i + a_ia_{i+2} + 25a_{i+1}^2 \\ -a_{i-1}a_{i+1} - a_{i+1}a_{i+2}) \\ \vdots \end{pmatrix}, \tag{7.77}$$

$$N(I_h^C(\mathbf{a})) = \begin{pmatrix} \vdots \\ \frac{1}{96}(-4a_{i-1}^2 - 4a_{i-1}hm_{i-1} + 4a_{i-1}hm_i \\ -h^2m_{i-1}^2 + 2h^2m_{i-1}m_i - 16a_{i-1}a_i \\ -8a_ihm_{i-1} + 16a_ihm_i + 16a_ia_{i+1} \\ -8a_ihm_{i+1} + 4a_{i+1}^2 + 4a_{i+1}hm_i \\ -4a_{i+1}hm_{i+1} - 2h^2m_im_{i+1} + h^2m_{i+1}^2) \\ \frac{1}{24}(-6a_i^2 - a_ihm_i + a_ihm_{i+1} + 6a_{i+1}^2 \\ +a_{i+1}hm_i - a_{i+1}hm_{i+1}) \\ \vdots \end{pmatrix}. \tag{7.78}$$

Using the computer algebra package Maple [44] to perform the complicated algebraic manipulation, we make use of (7.56) and rewrite $a_{i-2}, .., a_{i+2}$ in terms of a_i to obtain a comparison of the interpolants, as previously.

	At node x_i	At new node $x_i + h$
Interpolated residual	$2ha_ia_i' + O(h^3)$	$2(ha_ia_i' + a_ih^2a_i'' + h^2a_i'^2) + O(h^3)$
Linear interpolation	$ha_ia_i' + O(h^3)$	$ha_ia_i' + a_ih^2a_i'' + h^2a_i'^2 + O(h^3)$
Hermite interpolation	$ha_ia_i' + O(h^3)$	$ha_ia_i' + a_ih^2a_i'' + h^2a_i'^2 + O(h^3)$
Cubic Spline	$ha_ia_i' + O(h^3)$	$ha_ia_i' + a_ih^2a_i'' + h^2a_i'^2 + O(h^3)$

In the same way as the mass matrix and convection matrix terms were halved in the linear problem, the interpolants all halve $I_h^L(\hat{N}(\mathbf{a}))$ to second order, and so the behaviour of the interpolation operators for the nonlinear problem follows that of the linear problem.

Conclusions for the One-Dimensional Problem

We have demonstrated in this section that the choice of interpolant has a large effect on the o.d.e. residual when the spatial mesh is refined. For the one-dimensional

example considered, we have shown analytically that use of linear interpolation to find the solution values at the new nodes causes the size of the residual to jump sharply. However, if cubic spline or Hermite interpolants are used instead, then this increase is not great.

Numerical experiments verify this behaviour, and also show the connection between a large residual and a large reduction in time-step size following refinement. The reason why a large increase in the residual causes a sharp drop in the step size is considered in §7.5.3. Further analysis indicates that the residual will behave in the same way for nonlinear problems and when the Galerkin least-squares method is used.

Having established that in one-dimension, higher order interpolation offers improved performance to linear interpolation, we now compare Hermite with linear interpolation for a two-dimensional example on both structured and unstructured meshes.

7.5.2 Interpolation on Two Dimensional Meshes

In two dimensions linear interpolation of two nodal values to obtain a solution value at an edge midpoint (which is where new nodes appear in the refinement strategy used) is straightforward. We do not consider using cubic spline, due to its complexity in two dimensions, and results in 1-d show little improvement over using Hermite interpolation. Formation of a cubic Hermite interpolant is more complicated than the linear case, as within each element ten degrees of freedom are required, which consist of values at the three nodes, the partial derivatives at each node, and a value at the triangle centroid. The interpolant $S(x, y)$ on each element may be written as

$$S(x,y) = \sum_{i=1}^{3} u_i H_i(x,y) + \sum_{i=1}^{3} \frac{\partial u_i}{\partial x} K_{i,1}(x,y) + \sum_{i=1}^{3} \frac{\partial u_i}{\partial y} K_{i,2}(x,y) + L(x,y)u_c \quad (7.79)$$

where u_i and u_c are the solution values at the nodes and the element centroid, and the basis functions may be defined in terms of the piecewise linear basis functions ϕ_i, $i = 1, 2, 3$ (see [98]):

$$H_i = \phi_i(3\phi_i - 2\phi_i^2 - 7\phi_j\phi_k) \quad (7.80)$$

$$K_{i,1} = \phi_i((x_i - x_j)\phi_j(\phi_k - \phi_i) + (x_i - x_k)\phi_k(\phi_j - \phi_i)) \quad (7.81)$$

$$K_{i,2} = \phi_i((y_i - y_j)\phi_j(\phi_k - \phi_i) + (y_i - y_k)\phi_k(\phi_j - \phi_i)) \quad (7.82)$$

$$L = 27\phi_1\phi_2\phi_3 \quad (7.83)$$

where (i, j, k) is any cyclic numbering of $(1, 2, 3)$ and the positions of the three nodes are given by $\mathbf{s}_j = (x_j, y_j)$, $j = 1, 2, 3$. These functions have the following values at the nodes $\mathbf{s}_j$,

$$\begin{array}{lll} H_i(\mathbf{s}_j) = \delta_{ij}, & \frac{\partial H_i}{\partial x}(\mathbf{s}_j) = 0, & \frac{\partial H_i}{\partial y}(\mathbf{s}_j) = 0, \\ K_{i,1}(\mathbf{s}_j) = 0, & \frac{\partial K_{i,1}}{\partial x}(\mathbf{s}_j) = \delta_{ij}, & \frac{\partial K_{i,1}}{\partial y}(\mathbf{s}_j) = 0, \\ K_{i,2}(\mathbf{s}_j) = 0, & \frac{\partial K_{i,2}}{\partial x}(\mathbf{s}_j) = 0, & \frac{\partial K_{i,2}}{\partial y}(\mathbf{s}_j) = \delta_{ij}, \\ L(\mathbf{s}_j) = 0, & \frac{\partial L}{\partial x}(\mathbf{s}_j) = 0, & \frac{\partial L}{\partial y}(\mathbf{s}_j) = 0. \end{array} \quad (7.84)$$

The refinement process adds new nodes at the edge midpoints, so we require the value of the interpolant $S(x, y)$ at these points. Denoting these midpoints by $\mathbf{m}_i$ where $\mathbf{m}_i$ is the midpoint of the edge opposite $\mathbf{s}_i$, the linear functions $\phi_i, i = 1, 2, 3$ have the values

$$\phi_i(\mathbf{m}_j) = \begin{cases} 0 \text{ if } i = j \\ \frac{1}{2} \text{ otherwise} \end{cases} . \quad (7.85)$$

Substitution of these values into (7.79) leads to the following expression for obtaining solution values at edge midpoints

$$S(\mathbf{m}_i) = \frac{1}{2}(u_j + u_k) + \frac{1}{8}\left\{(x_k - x_j)(\frac{\partial u_j}{\partial x} - \frac{\partial u_k}{\partial x}) + (y_k - y_j)(\frac{\partial u_j}{\partial y} - \frac{\partial u_k}{\partial y})\right\}, \quad (7.86)$$

where (i, j, k) is any cyclic numbering of $(1, 2, 3)$. This equation requires approximations of the partial derivatives of the solution at the three element nodes. In the case of a structured Cartesian mesh, these values may be obtained from second order difference formulae in the same way as for the one dimensional case (equations (7.37)–(7.38)). For an unstructured mesh, there are no equivalent formulae. For this reason, the approach taken here is to make use of the piecewise constant partial derivatives on the surrounding elements. The formula used is

$$\nabla u = \frac{1}{\sum_{i=1}^{k} A_i} \sum_{i=1}^{k} A_i (\nabla u)_i, \quad (7.87)$$

where A_i and $(\nabla u)_i$ denote the area and gradient of the k elements surrounding the node. This idea is an extension of a similar formula for regular meshes, for which certain convergence results exist (details are given by Goodsell and Whiteman in [51]).

We now use the interpolants outlined above in the refinement of a two dimensional mesh when solving an unsteady problem. In §4.5, a coupled system of Burgers' equations was introduced. We consider a time-dependent version of the equations here, obtained by removing the source terms. The equations become

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - \epsilon(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) = 0 \tag{7.88}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} - \epsilon(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) = 0, \tag{7.89}$$

where ϵ is a small constant, and the exact solution is given by

$$u = \frac{3}{4} - \frac{1}{4}\frac{1}{1+\exp((-4x+4y-t)/(32\epsilon))} \tag{7.90}$$

$$v = \frac{3}{4} + \frac{1}{4}\frac{1}{1+\exp((-4x+4y-t)/(32\epsilon))}, \tag{7.91}$$

which represents a moving front situated at $y = x + 0.25t$. The spatial domain consists of the unit square $[0,1] \times [0,1]$, and the Dirichlet boundary conditions are continuously updated as the solution progresses.

Using both structured and unstructured meshes, we observe the effect that uniformly refining the entire mesh has on the size of the time-step δ. The three structured meshes used consist of regular grids of 9×9, 17×17 and 33×33 nodes (i.e. 128, 512 and 2048 elements respectively). The three unstructured meshes consist of 146, 568 and 2310 elements (the third of these is shown in figure 4.5).

Using two values (0.01 and 0.001) of ϵ, the problem is solved via the method of lines (so that the equations are semi-discretized using finite elements leading to a system of o.d.e.'s solved using the backward Euler method in SPRINT) until 0.25 units of time have passed. The entire mesh is then refined one level and the solution values at the new nodes are formed using either linear or Hermite interpolation. In the latter case, the partial derivatives are formed using second order differences if the mesh is structured or the formula (7.87) if it is unstructured.

For both types of mesh, we run twelve cases (as there are three meshes, two values of ϵ and two interpolants) and monitor the time-step size immediately before and after refinement. Table 7.2 shows the values of δ for all the cases.

On the structured meshes, when ϵ=0.01, there is a clear improvement in using Hermite interpolants rather than linear ones, as it causes a smaller reduction in the time-step size on all three meshes. This improvement becomes more apparent as

ϵ	Mesh	δ_{old}	δ_{new} Linear	δ_{new} Hermite	Ratio (Her/Lin)
Structured meshes (p.d.'s from difference formulae)					
0.01	1	2.66e-2	2.22e-4	1.68e-3	7.57
0.01	2	1.84e-2	1.80e-4	2.07e-3	11.5
0.01	3	1.32e-2	1.69e-5	9.29e-3	550
0.001	1	1.12e-2	3.96e-4	7.26e-5	0.183
0.001	2	5.87e-3	3.30e-5	3.42e-5	1.03
0.001	3	2.99e-3	1.62e-5	1.93e-5	1.19
Unstructured meshes (p.d.'s from weighted average)					
0.01	1	2.80e-2	1.92e-4	2.96e-4	1.54
0.01	2	1.78e-2	1.40e-4	1.40e-3	10.0
0.01	3	1.29e-2	1.30e-4	1.50e-3	11.5
0.001	1	7.82e-3	5.01e-5	2.89e-5	0.57
0.001	2	4.75e-3	2.54e-5	2.43e-5	0.96
0.001	3	3.38e-3	1.47e-5	1.82e-5	1.24

Table 7.2: Effect on the time-step size using linear and Hermite Interpolation.

the mesh becomes finer, as shown by the ratio in the last column of table 7.2, and on mesh 3 (when $h = 1/32$), there is only a very small drop in δ.

When ϵ is reduced to 0.001, the superiority of Hermite interpolation is less obvious, and on the coarsest mesh 1, linear interpolation results in a larger step size. However on the finest mesh, Hermite is slightly better than linear interpolation, suggesting that as $h \to 0$, it definitely leads to better results. In order to see this improvement clearly though, a very fine mesh is needed when ϵ is small.

The partial derivatives required for the Hermite interpolant on the structured meshes are found by second order difference formulae, which allow an accurate interpolant to be formed. For the cases on unstructured meshes, the formula (7.87) is used, and as a consequence the results for these cases, shown in the second half of table 7.2, are less conclusive. Improvements, particularly when ϵ=0.01, in the step-size can still be seen for Hermite interpolation, but there is less difference between the linear and Hermite interpolants than on structured meshes.

If a more accurate method for computing the partial derivatives on unstructured meshes were to be used, particularly near the boundary where (7.87) is very inaccurate, then it is hoped that the results on unstructured meshes would be very similar to those on structured meshes.

Overall, these results show that the behaviour of the step-size in two dimensions

is similar to what happens in one dimension—refining with linear interpolation results in a large o.d.e. residual which causes the time-step size to drop sharply. The residual is smaller when higher order interpolants are used (provided they are accurate) which leads to less severe reductions in the step size. In the following section, we show how the step-size is affected by the residual.

7.5.3 Connection between Residual and Time-Step

In the previous two sections we have seen, in one and two dimensions, how interpolation affects the size of the new residual vector, after refining a mesh. As a result, the size of the time-step immediately after refinement can drop significantly, and in this section we briefly discuss why the size of the new residual affects the time-step δ. We use the backward Euler method as a simple example, but the approach is typical of that used in general o.d.e. software such as SPRINT. Consider the o.d.e. system

$$\dot{\mathbf{y}} = \mathbf{g}(\mathbf{y}, t). \tag{7.92}$$

This may be solved using the backward Euler scheme in two stages: an explicit (predictor) step is carried out to calculate an initial guess, $\mathbf{y}_n^p$ for the second stage,

$$\mathbf{y}_n^p = \mathbf{y}_{n-1} + \delta \dot{\mathbf{y}}_{n-1}, \tag{7.93}$$

where $\mathbf{y}_{n-1}$ and $\dot{\mathbf{y}}_{n-1}$ are vectors saved from the previous time-step and δ is the current time-step size. The second (corrector) step is the backward Euler step, requiring solution of a nonlinear equation,

$$\mathbf{y}_n^c = \mathbf{y}_{n-1} + \delta \mathbf{g}(\mathbf{y}_n^c, t_n). \tag{7.94}$$

Once this has been solved, a local error estimate is used to decide whether the current step is acceptable. This is defined as

$$\tau_n = \frac{1}{2}(\mathbf{y}_n^c - \mathbf{y}_n^p) \tag{7.95}$$

$$= \frac{\delta}{2}(\mathbf{g}(\mathbf{y}_n^c, t_n) - \dot{\mathbf{y}}_{n-1}). \tag{7.96}$$

If $||\tau_n|| < \epsilon$, for some error tolerance ϵ and choice of norm, then the integration continues with the next step (possibly increasing the step-size if $||\tau_n||$ sufficiently small), otherwise the step size is reduced and the current step repeated, until $||\tau_n||$ is sufficiently small.

When the mesh has not been refined,

$$||\dot{\mathbf{y}}_{n-1} - \mathbf{g}(\mathbf{y}_{n-1}, t_{n-1}))|| = ||\mathbf{r}_{n-1}|| << \epsilon \tag{7.97}$$

and so

$$\tau_n \approx \frac{\delta}{2}(\mathbf{g}(\mathbf{y}_n, t_n) - \mathbf{g}(\mathbf{y}_{n-1}, t_{n-1})) \tag{7.98}$$

and there is usually no need to reduce the step size δ.

When the mesh is refined at the end of the previous time-step, then the size of the o.d.e. system is changed and we define new vectors via an interpolation operator I_h,

$$\tilde{\mathbf{y}}_{n-1} = I_h(\mathbf{y}_{n-1}),\ \tilde{\dot{\mathbf{y}}}_{n-1} = I_h(\dot{\mathbf{y}}_{n-1}). \tag{7.99}$$

The residual vector $\tilde{\mathbf{r}}_{n-1}$ is given by

$$\tilde{\mathbf{r}}_{n-1} = \tilde{\dot{\mathbf{y}}}_{n-1} - \tilde{\mathbf{g}}(\tilde{\mathbf{y}}_{n-1}, t_{n-1}) \tag{7.100}$$

and we can no longer assume that $||\tilde{\mathbf{r}}_{n-1}||$ is small—exactly how large it is will depend on the type of interpolant used to define I_h, as seen in the previous section. As before, a predictor is formed,

$$\tilde{\mathbf{y}}_n^p = \tilde{\mathbf{y}}_{n-1} + \delta\tilde{\dot{\mathbf{y}}}_{n-1}, \tag{7.101}$$

and the corrector is obtained via solution of the nonlinear system

$$\tilde{\mathbf{y}}_n^c = \tilde{\mathbf{y}}_{n-1} + \delta\tilde{\mathbf{g}}(\tilde{\mathbf{y}}_n^c, t_n), \tag{7.102}$$

using $\tilde{\mathbf{y}}_n^p$ as the initial guess.

The local error estimate $\tilde{\tau}_n$ is now

$$\begin{aligned}
\tilde{\tau}_n &= \frac{1}{2}(\tilde{\mathbf{y}}_n^c - \tilde{\mathbf{y}}_n^p) && (7.103)\\
&= \frac{\delta}{2}(\tilde{\mathbf{g}}(\tilde{\mathbf{y}}_n^c, t_n) - \tilde{\dot{\mathbf{y}}}_{n-1}) && (7.104)\\
&= \frac{\delta}{2}(\tilde{\mathbf{g}}(\tilde{\mathbf{y}}_n^c, t_n) - \tilde{\mathbf{g}}(\tilde{\mathbf{y}}_{n-1}, t_{n-1}) - \tilde{\mathbf{r}}_{n-1}). && (7.105)
\end{aligned}$$

Because of the error introduced by the interpolation, $||\tilde{\mathbf{r}}_{n-1}||$ is large relative to $||\tilde{\mathbf{g}}(\tilde{\mathbf{y}}_n^c, t_n) - \tilde{\mathbf{g}}(\tilde{\mathbf{y}}_{n-1}, t_{n-1})||$, and hence $||\tilde{\tau}_n||$ is too large. The step size may have to be significantly reduced before the local error estimate is below the tolerance ϵ, especially if the residual vector is particularly big (which it is in the case of linear

interpolation). By using an improved interpolant to lower the size of the residual, a smaller reduction in step size can be made.

Although we only consider the simple method of backward Euler here, it should be possible to show a similar result for more sophisticated schemes, such as those considered in §7.3.1.

7.5.4 Time-Dependent Navier-Stokes Equations

Having discussed the relationship between interpolation and the size of the time-step which follows any refinement of the mesh for simple problems, we now return to the Navier-Stokes equations. In §7.4, solutions are shown for an unsteady test case, but the solution process is slow due to the large reduction in the time-step size after each refinement. On the basis of the results and analysis shown above, we would expect that accurate use of higher order interpolation than linear should reduce the size of the residual and hence cause the step size not to fall so sharply.

To test this in practice, we now solve the unsteady test case used previously (where $Re = 5000$, $M_\infty = 0.55$ and $\alpha = 8.34°$) in exactly the same way as in §7.4, except that Hermite interpolation rather than linear interpolation is used when refining the spatial mesh. This involves the use of expression (7.86) which provides the solution values at element mid-points (i.e. the positions of new nodes). In addition the partial derivatives at each node are required, and these are approximated using the formula (7.87) since this example is solved on an unstructured mesh.

The results are disappointing, as there is no apparent improvement in the step size reduction when compared to the case where linear interpolation was used. Presumably, this is due to the poor approximations of the partial derivatives of the solution at the nodes. These are least accurate near the boundaries, including the aerofoil surface, where accurate interpolation is especially important since a great deal of the refinement will take place in this region.

The averaging formula (7.87) is an attempt to extend to unstructured meshes a result obtained for regular structured meshes [51]. This gradient recovery technique followed from the phenomenon of superconvergence in finite element methods (where the rate of convergence at certain points within the domain exceed the global rate) and there has been much work on these subjects for regular meshes (see [85] for a general survey). The corresponding analysis for unstructured meshes is still

an open problem however, and development of alternatives to (7.87) might provide more accurate derivatives, leading to improved interpolation.

In [24], Carey describes a method for obtaining derivatives along the boundary for elliptic equations. It uses the boundary integrals, in which the boundary flux appears, computed in the Galerkin finite element discretization. However if we attempt to extend this to the system of Navier-Stokes equations, and use the computed boundary derivatives for the Hermite interpolation, then no noticable improvement is observed in the test case used above.

Although we are not able to currently overcome this problem of poor estimates to the partial derivatives on unstructured meshes, it is hoped that Hermite interpolation on structured meshes around aerofoils might be more successful, as the second order difference formula (suitably transformed if necessary) would provide more accurate approximations to the derivatives. This is a possible area for further research in the short-term, although in the longer term a more reliable method for recovering derivatives on unstructured meshes should be sought.

7.6 Summary

In this chapter we have considered unsteady problems, where the ability to obtain time-accurate solutions is more important than being able to march the solution forward in time quickly so as to reach a converged steady solution. We have outlined some methods which allow such time-accuracy in §7.2, including complete space-time finite element methods and the method of lines. The latter has been used as the main approach in this chapter, and the software package SPRINT [13] has provided the methods for solving the system of o.d.e.'s obtained after the p.d.e.'s have been semi-discretized (using a stabilized finite element method).

Taking this approach, the solutions obtained for the test case in §7.4 appeared to be satisfactory. The unsteadiness in the flow has been clearly shown, and the use of h-refinement (including derefinement) has detected the wake successfully, at least qualitatively. We have not considered the implementation of r-refinement in this chapter, as using it with derefinement introduces additional complications, which are discussed in §8.3.

However, solution of this problem was slow, due mainly to the sharp fall in the size of the time-step immediately after refinement. Because of the corresponding

increase in the size of the new residual vector, it appeared that reducing this value might prevent such a reduction in the time-step.

The analysis of a simple one dimensional problem in §7.5.1 showed that improved interpolation reduces the size of the residual, and this has also been shown numerically for a two dimensional problem. For the simple time integration scheme of backward Euler, the link between a large residual vector and a large reduction in the time-step has been demonstrated. The next step was to return to the Navier-Stokes equations and attempt to use a higher order of interpolation than linear in order to prevent the large reduction of the time-step size.

Although we have made several attempts to implement Hermite interpolation for the Navier-Stokes equations on unstructured meshes, we have not been able to demonstrate any significant gain over using linear interpolation. It would seem that this is due to insufficiently accurate approximations of the gradients which are needed at each node. If improved estimates to the partial derivatives could be found, then we would expect that use of a Hermite interpolant would lower the o.d.e. residual and prevent a sharp drop in the step size, leading to faster solution times.

On the basis of these results, and in the absence of a suitable interpolant in 2-d, we conclude that this combination of a stabilized finite element method with a separate o.d.e. solver is currently rather inefficient and another method such as the discontinuous Galerkin (which uses finite elements in both space and time) [77] should be considered as an alternative. One possible reason why this method might be better is that a discontinuity in the o.d.e. is introduced when the mesh is refined, and since the discontinuous Galerkin method is discontinous in time, no smoothness is imposed at the point where the mesh is refined. It would be of interest to make a comparison between these two techniques.

Chapter 8

Future Areas of Research

8.1 Introduction

This chapter contains discussions on three possible areas in which the work described in previous chapters might be extended. We have considered compressible flow at low to moderate Reynolds numbers, usually for problems which have steady solutions. When simulating high Reynolds number flows, in which turbulence is present and has a significant effect on the flow features, direct solution of the unsteady Navier-Stokes equations is computationally too expensive. Instead, a turbulence model is needed, and in §8.2 we describe the most common models which are used. We also briefly consider some aspects of implementing a turbulence model within the finite element approach we have described in chapter 2.

In chapter 7, the use of h-refinement in methods for time-dependent flows was demonstrated and we may wish to also make use of the r-refinement techniques presented in chapter 6. In §8.3, this issue is addressed, and some of the possible difficulties which might arise are discussed.

Only two-dimensional flows are considered in this thesis, and in §8.4 we consider what changes would be involved if the ideas and algorithms presented here were extended to three dimensions.

8.2 Turbulence Modelling

A viscous flow may be classified as one of two types —laminar or turbulent. Below a critical Reynolds number (which usually depends on the particular flow conditions

and geometry), the flow is laminar and remains smooth and regular. At this critical value, the flow enters a transition region in which instabilities begin to appear. These instabilities cause turbulence as the Reynolds number increases further.

It is important to account for the effects of turbulence, such as increased friction on surfaces, and a decreasing likelihood of flow separation from a wall surface (see for example Hinze [63] for a fuller description of turbulence). In principle, turbulent solutions may be obtained from the unsteady Navier-Stokes equations, using the approach of chapter 7 for example. However, with the exception of relatively simple flows this would require a very fine grid and far more computational power than is currently available in practice. Instead, average values for the unknown variables are sought, by means of the time (or Reynolds) averaged Navier-Stokes equations. These contain additional terms, which are modelled by further equations (which may be algebraic or differential). The averaged values obtained from these equations are usually sufficient to show how the flow is affected by turbulence.

In §8.2.1 we state the averaged Navier-Stokes equations, and some alternatives for modelling the additional terms in the equations are outlined in §8.2.2. Various implementation issues are discussed in §8.2.3.

8.2.1 The Reynolds Averaged Navier-Stokes Equations

We wish to obtain time-averaged values of the unknowns, ignoring the fluctuations caused by turbulent effects. Hence we split the unknown variable (e.g. u) into two parts

$$u = \overline{u} + u' \tag{8.1}$$

where $\overline{u}$ is either the time average

$$\overline{u}(t) = \frac{1}{\mathcal{T}} \int_{t-\frac{\mathcal{T}}{2}}^{t+\frac{\mathcal{T}}{2}} u(\tau) d\tau \tag{8.2}$$

for a suitable time-scale $\mathcal{T}$, or ensemble average

$$\overline{u}(t) = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} u^i(t) \tag{8.3}$$

which is obtained by repeating the transient process many times, so that $u^i(t)$ denotes the measured value of u at time t in the ith experiment. These two averaging operators may be assumed to be equivalent for most types of turbulent flow. It

follows that for the fluctuating term u',

$$\overline{u'} = 0. \tag{8.4}$$

This type of averaging is adequate when the density, ρ, is constant (i.e. incompressible flows), but for variable density flows, the resulting averaged equations contain some complex terms which may be avoided by using Favre averaging, where the unknowns are weighted by the density. In this case

$$u = \tilde{u} + u'' \tag{8.5}$$

where $\tilde{u}$ is the Favre average

$$\tilde{u}(t) = \frac{1}{\overline{\rho}\mathcal{T}} \int_{t-\frac{\mathcal{T}}{2}}^{t+\frac{\mathcal{T}}{2}} \rho(\tau)u(\tau)d\tau \tag{8.6}$$

or in ensemble form,

$$\tilde{u}(t) = \lim_{N\to\infty} \frac{1}{\overline{\rho}N} \sum_{i=1}^{N} \rho^i(t)u^i(t). \tag{8.7}$$

We may also write

$$\tilde{u} = \frac{\overline{\rho u}}{\overline{\rho}} \qquad \text{(from (8.6))}. \tag{8.8}$$

Using the conservative form of the Navier-Stokes equations (given in vector form by equation (2.27) in §2.2.2), we wish to rewrite the equations in terms of the averaged quantities $\overline{\rho}$, $\tilde{u_1}$, $\tilde{u_2}$, $\tilde{e}$, $\overline{T}$ and $\overline{p}$. This is done by averaging the entire equations, and then simplifying. For example the continuity equation becomes

$$\overline{\rho_{,t} + (\rho u_i)_{,i}} = 0 \tag{8.9}$$

where "$_{,i}$" denotes partial differentiation by x_i and the summation convention is being used. Making use of (8.8), this may be rewritten as

$$\overline{\rho}_{,t} + (\overline{\rho}\tilde{u}_i)_{,i} = 0. \tag{8.10}$$

Hence no new terms are introduced in this equation, but this is not the case in the remaining equations due to the presence of nonlinear terms. The Reynolds averaged momentum equations

$$\overline{(\rho u_i)_{,t} + (\rho u_i u_j + p\delta_{ij})_{,j} - \mu S_{ij}(\mathbf{u})_{,j}} = 0, \tag{8.11}$$

where μ is the molecular viscosity and $S_{ij}(\mathbf{u})$ is defined as

$$S_{ij}(\mathbf{u}) = u_{i,j} + u_{j,i} - \frac{2}{3}\delta_{ij}u_{k,k}, \tag{8.12}$$

become

$$(\overline{\rho}\tilde{u}_i)_{,t} + (\overline{\rho}\tilde{u}_i\tilde{u}_j + \overline{\rho}\widetilde{u_i''u_j''} + \overline{p}\delta_{ij})_{,j} - \mu S_{ij}(\tilde{\mathbf{u}})_{,j} = 0 \tag{8.13}$$

introducing the Reynolds stress tensor $\overline{\rho}\widetilde{u_i''u_j''}$ which has to be modelled.

The instantaneous energy equation is

$$(\rho e_{\text{tot}})_{,t} + (\rho u_i e_{\text{tot}} + p u_i)_{,i} - (\mu S_{ij}(\mathbf{u})u_j + \kappa T_{,i})_{,i} = 0. \tag{8.14}$$

where κ is the thermal conductivity and

$$e_{\text{tot}} = e + \frac{|\mathbf{u}|^2}{2}. \tag{8.15}$$

The Reynolds averaged equation may be written as

$$\begin{aligned}(\overline{\rho}\tilde{e}_{\text{tot}})_{,t} + (\overline{\rho}\tilde{u}_i\tilde{e}_{\text{tot}} + \overline{\rho}\widetilde{u_i''e_{\text{tot}}''} + \overline{p}\tilde{u}_i + \overline{pu_i''})_{,i} \\ -(\mu(S_{ij}(\tilde{\mathbf{u}})\tilde{u}_j + S_{ij}(\tilde{\mathbf{u}})\overline{u''}_j + S_{ij}(\overline{\mathbf{u}''})\tilde{u}_j + \overline{S_{ij}(\mathbf{u}'')u_j''} + \kappa\overline{T}_{,i})_{,i} = 0\end{aligned} \tag{8.16}$$

which introduces even more extra terms to be modelled. We are now faced with the well-known closure problem—six equations (the four Navier-Stokes equations and two expressions relating p and T in terms of ρ, u_i and e) but many more unknowns to be determined, including the four (in two dimensions) components of the Reynolds stress tensor and the turbulent flux terms appearing in (8.16).

A common approach is to make the assumption about the Reynolds stress tensor that it may be described in the same way as molecular viscous effects (the eddy viscosity hypothesis, see [27] for example), so that

$$\overline{\rho}\widetilde{u_i''u_j''} = -\mu_T S_{ij}(\tilde{\mathbf{u}}) + \frac{2}{3}\overline{\rho}K\delta_{ij}, \tag{8.17}$$

where μ_T is the eddy viscosity and the turbulent kinetic energy K is defined by

$$K = \frac{1}{2}\widetilde{u_i''u_i''}. \tag{8.18}$$

Similarly the eddy diffusivity hypothesis assumes that the turbulent fluxes are proportional to the gradient of the mean values, so for example

$$\overline{\rho}\widetilde{u_i''e_{\text{tot}}''} = \frac{\mu_T}{Pr_T}\tilde{e}_{\text{tot,i}} \tag{8.19}$$

Hence we can now rewrite the averaged momentum equation as

$$(\overline{\rho}\tilde{u}_i)_{,t} + (\overline{\rho}\tilde{u}_i\tilde{u}_j + + (\overline{p} + \frac{2}{3}\overline{\rho}K)\delta_{ij})_{,j} - (\mu + \mu_T)S_{ij}(\tilde{\mathbf{u}})_{,j} = 0. \tag{8.20}$$

After some algebraic manipulation (see Jansen *et al* [75] for details), the energy equation can also be written in a simpler form,

$$(\overline{\rho}\tilde{e}_{\rm tot})_{,t} + (\overline{\rho}\tilde{u}_i\tilde{e}_{\rm tot} + (\overline{p} + \tfrac{2}{3}\overline{\rho}K)\tilde{u}_i)_{,i}$$
$$-((\mu+\mu_T)\tilde{u}_j S_{ij}(\tilde{\mathbf{u}}) + (\kappa+\kappa_T)\overline{T}_{,i} + (\mu + \tfrac{\mu_T}{Pr_K})K_{,i})_{,i} = 0. \qquad (8.21)$$

The turbulent Prandtl number Pr_T is a constant and the turbulent thermal conductivity is defined by

$$\kappa_T = \gamma c_v \frac{\mu_T}{Pr_T}. \qquad (8.22)$$

This set of averaged equations (8.10), (8.20) and (8.21) now only require additional modelling of the turbulent kinetic energy K and the eddy viscosity μ_T. Turbulence models which approximate these values are categorized by the number of additional differential equations of which they consist. The most common types are zero, one and two equation models, but all models contain several constants which need to be empirically determined, by measuring quantities in practical experiments involving simple turbulent flows. This highlights the fact that particular models are usually better suited to certain flows than others. In the next section we briefly outline and give examples of the different types of model, and also mention some of the alternative approaches.

8.2.2 Types of Turbulence Models

We discuss in this section some of the models used to obtain the eddy viscosity μ_T and turbulent kinetic energy K, which are then used in the Reynolds averaged Navier-Stokes equations, as shown above. Of particular interest is their suitability for unstructured meshes, as historically most solvers incorporating turbulence models have been based on the use of structured meshes in the turbulent regions.

Zero-Equation Models

Models in this category use only algebraic expressions to approximate the turbulent quantities. Typically, the eddy viscosity is modelled so that

$$\mu_T \propto \rho l_t u_t \qquad (8.23)$$

where l_t is the turbulent length scale and u_t is the turbulent velocity scale, which are obtained from the local mean flow quantities. In this case, it appears to be

common to rewrite the Reynolds averaged Navier-Stokes equations so that the term $\frac{2}{3}\overline{\rho}K$ appearing in (8.20) and (8.21) is "absorbed" into the definition of pressure and the diffusive term involving K is no longer present. This avoids the need to obtain K directly.

This type of model is really only suited to flows near walls, where diffusive effects dominate, since transport effects are ignored. One early model is that of Cebeci and Smith [28], which splits the boundary layer into two regions, and defines the eddy viscosity for each region separately. One disadvantage with this model is the necessity to locate the boundary layer in order to determine the length scale in the outer region. The Baldwin-Lomax model [10] avoids this need, instead using the distribution of the vorticity to obtain length scales.

For more complex flows, results obtained using the model of Johnson and King [82] are superior to both the Cebeci-Smith and Baldwin-Lomax models. This model transforms a simplified form of the turbulent kinetic energy transport equation into a o.d.e. which describes the development of the maximum Reynolds stress in the streamwise direction.

For high Reynolds number flows which remain attached to the wall, these models give good results, but cannot accurately predict highly separated flows and turbulence in wakes. Since they are essentially algebraic models, they are computationally inexpensive, but require turbulent length and velocity scales. This requires knowledge of the distance of each point from the wall surface (y) which is easily available when using structured meshes, but not when the mesh is unstructured. In [93], Mavriplis introduces a method for using the Baldwin-Lomax model with unstructured meshes. This is accomplished by generating lines normal to the wall and interpolating solution values from the unstructured mesh points onto the normal mesh lines in order to obtain the necessary length scales.

One-Equation Models

As for algebraic models, the approximation (8.23) is made, but in this case the turbulent velocity scale is usually defined to be

$$u_t = \sqrt{K} \tag{8.24}$$

where the turbulent kinetic energy K is found from a transport equation. It is common to introduce the turbulence dissipation rate ϵ to indirectly represent l_t, via

the following:

$$l_t = C\frac{K^{\frac{3}{2}}}{\epsilon} \tag{8.25}$$

for some constant C.

The model of Wolfshtein [128] is a simple one-equation model, consisting of a p.d.e. with K as the unknown, with the length scale being determined in terms of y, the distance from the wall. A slightly more advanced model is given by Mitcheltree *et al.* [99], where the length scale is determined in different ways for attached and separated flows. A blending function is used to switch between the two formulations.

The Norris-Reynolds model is used by Jansen *et al.* [75] within the context of the Galerkin least-squares finite element method. The extra equation is combined with the Reynolds averaged Navier-Stokes equations and the resulting system is transformed using entropy variables so that the advective and diffusive matrices are symmetric (see §2.2.2).

All the models mentioned so far approximate the length scale in terms of y, hence are less suitable for unstructured meshes than structured grids. The Baldwin-Barth model [9] overcomes this problem to a large extent, and consists of an equation modelling the turbulent Reynolds number R_T, from which the eddy viscosity may be found directly. A length scale is not needed, although there is still some dependence on y, as it is used by damping functions which simulate the effect of solid walls on turbulence. It is recommended in [9] that y need only be stored for the nodes very close to the wall, since the damping functions reach their asymptotic value away from the wall.

Another model which has only a limited dependence on y is that of Spalart and Almaras [113], in which a p.d.e. is used to obtain a value for μ_t directly.

In general, one-equation models can cope with a wider range of flows than algebraic models, though at extra expense, and some have the ability to be used on unstructured meshes with relative ease.

Two-Equation Models

Two-equation models eliminate the need to find algebraic length and velocity scales, and instead rely on two transport equations to obtain values from which μ_T may be found. The most common approach is the K-ϵ model where the two equations involve K and ϵ, the rate of dissipation of turbulent kinetic energy. The eddy

viscosity is then determined from

$$\mu_T = \rho C \frac{K^2}{\epsilon}. \tag{8.26}$$

The standard K-ϵ model is generally applicable in regions of fully developed turbulence, but not near wall boundaries. There are two ways of overcoming this. A low Reynolds number model incorporates extra terms into the ϵ equation, and redefines the eddy viscosity near walls. This approach requires particularly fine grids close to the wall. The second idea is to either employ a separate simplified model or impose logarithmic wall laws near the wall (a high Reynolds number model).

The model of Jones and Launder [84] is a well known version of the K-ϵ model, and can either be used as a low Reynolds number model, or may be simplified and treated as a high Reynolds number model. In [30], another low Reynolds number model is presented by Chien, similar to [84] but taking a different approach to modelling terms near the wall.

The advantages of two-equation models such as K-ϵ are that they are applicable to a wide range of flows, and have been in use for over twenty years, so that their uses and limitations are well understood. However they are more difficult to use than the zero and one-equation models discussed above, they often require finer meshes near wall boundaries, and numerical difficulties may occur during their solution. Because there is no need for length scales to be determined algebraically from wall distances, this type of model may be used on unstructured grids.

Other Turbulence Models

If the eddy viscosity hypothesis is no longer assumed to hold, then a Reynolds stress model may be used. In this type of model, each component of the Reynolds stress tensor is modelled by a separate transport equation. An example of this type is given by Gibson and Launder in [48]. Although mathematically and physically rigorous, the complexity of the extra equations has led to this approach not usually being used in practice.

Another technique used is that of large-eddy simulation. Here, the large scales of motion are computed accurately, while the smaller scale motions occurring on a subgrid scale are modelled.

The most computationally demanding approach is the direct numerical simulation involving the unsteady Navier-Stokes equations, so that no turbulence model

is used. This requires a very fine grid in order to resolve the turbulent fluctuations, and is currently limited to very simple turbulent flows at low Reynolds numbers.

8.2.3 Implementation Issues

In practice, turbulent flows occur at high Reynolds numbers (i.e. $Re > 10^5$), and this will require modifications to the numerical scheme used to discretize the Reynolds averaged Navier-Stokes equations. A discontinuity capturing term (such as that included in the Galerkin least-squares method of Hughes and Mallet [71]) needs to be added so that the solver will converge at these high Reynolds numbers.

If we are to incorporate a turbulence model into the method outlined in previous chapters, then it must be one that is suitable for use on unstructured meshes. This excludes the algebraic models (unless the approach of Mavriplis [93] where a background structured mesh is used), and some of the one-equation models. It is also important that the mesh is sufficiently fine, especially near the body surface to allow the turbulence model to resolve the eddy viscosity correctly. One-equation models such as Baldwin-Barth or Spalart-Almaras generally require fewer elements near the wall than many of the $K - \epsilon$ models.

Whichever model is used, there is choice of whether to couple the averaged Navier-Stokes equations with the model equation(s) or solve them as two separate sets of equations. Jansen *et al.* [75] take the former approach, which is less likely to cause numerical difficulties during convergence, but is more complicated to implement. In either case the numerical scheme used to solve the turbulence equations needs to be chosen with care, so as to avoid negative values of the eddy viscosity. If adaptive methods such as hr-refinement are to be used, then consideration needs to be given as to whether the error indicator should be changed in any way. Use of the node movement algorithm requires that any wall distances being stored may need to be changed accordingly.

Turbulence models are notoriously difficult to implement correctly, and we have attempted to discuss the most common types that are used and some of the difficulties from a practical point of view. However the inclusion of turbulence models within compressible flow solvers is essential for simulating realistic flows around many practical aerodynamic configurations.

8.3 r-Refinement for Time-Dependent Problems

A method for solving transient problems, where time-accuracy is important, was demonstrated in chapter 7. We also implemented h-refinement for such problems, which used both refinement and derefinement of the mesh to add and subtract nodes as the solution changed over time. We consider in this section some possible issues which might arise if node movement (r-refinement) were to be used as an alternative or addition to h-refinement.

8.3.1 r-Refinement Only

The inclusion of the mesh relocation algorithm into the time-accurate solver (which in chapter 7 was based upon the software package SPRINT [13]) is straightforward in principle. The node update routine would be called at the end of each time-step (or after every n steps for some n). One problem with using h-refinement, discussed in detail in §7.5, is the drop in the size of the time-step which occurs immediately after refinement. This may also occur when the nodes are relocated, although a possible solution would be to interpolate (perhaps using a better interpolant than linear) the solution values at the new node positions.

When using r-refinement only for solving steady problems, a number of difficulties were noted (see §6.4), and these could apply equally to unsteady problems. For example the large variation in element size on meshes around aerofoils appeared to cause distorted elements in regions where they were not required.

For simplicity, we treated nodes which lie on boundaries as stationary (i.e. their positions were not changed by the node updating step) when considering steady flows, but movement along the boundaries would probably have to be introduced in the case of time-dependent flows, in order to follow any non-stationary features near the boundaries.

The error indicator which determines where refinement should take place is used for both node movement and node addition/removal, and is based upon the current time-dependent solution. Hence the refined mesh often lags slightly behind the newly computed solution, so that for example in a flow where there is a sharp front moving across the domain, the most heavily refined region of the mesh might be just behind the front itself. In the case of h-refinement this is compensated for by ensuring that sufficient refinement is carried out either side of the region where the

error indicator is highest. This cannot be done in the same way when r-refinement is being used by itself.

The points discussed above suggest that, just as for steady flows, r-refinement would possibly not be very suitable as an adaptive technique on its own. We now consider whether a combined approach involving both h and r refinement might be taken.

8.3.2 hr-Refinement

As results in chapter 6 indicate, the use of hr-refinement for problems involving steady flows leads to an efficient method for obtaining solutions. However there is an additional complication when solving for unsteady flows, because of the need to derefine elements. This is demonstrated in figure 8.1, which shows a typical subdivision of an element into four, the subsequent repositioning of the nodes and then the derefinement of the parent element. This deformed element is no longer triangular, causing the finite element solver to fail.

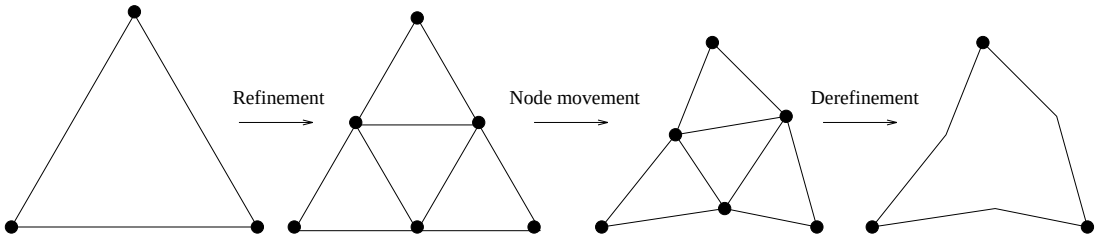

Figure 8.1: h-refinement–node movement–derefinement leading to a deformed triangle.

Hence we cannot use r-refinement in the same way as for steady problems, and we need to modify the technique in some way. One idea is to restrict node movement to the coarse initial mesh, which will never be derefined. Access to the coarse elements is available via the tree-like data structure being used, so this approach is relatively straightforward, but it is not clear how much effect this would have on the mesh. A possible complication is that node tangling between original nodes and new nodes might occur.

Another way of allowing derefinement is to recover the triangular shape of the derefined deformed element. This is more complicated to implement than the previous suggestion, but would be more effective. At its simplest, a triangle could be

recovered by forming the three edges between the element nodes, however situations might arise where this would lead to overlapping elements.

Other aspects to consider even if derefinement could be dealt with successfully include the question of when to refine (for both h and r-refinement) and how the size of the time-step would behave following refinement.

If derefinement could be made to work successfully when node movement is being used, then the advantages gained by using hr-refinement for steady flows would be likely to follow for transient flows, although the reduction in time-step size might still be a problem to overcome.

8.4 Solution of 3-D Flow

In this section we briefly discuss how three-dimensional problems might be solved using the approach we have taken in this thesis. Although only two-dimensional flows have been considered, the extension to 3-d should, in principle, follow without any major difficulties.

The Navier-Stokes equations in 3-d are very similar to the 2-d version, with the addition of an extra momentum equation. The conservative formulation follows in an obvious way from the 2-d form given by equation (2.27). The primitive formulation (equations (2.8)–(2.10)) is extended in a similar way, with a modification to the term $F(\nabla \mathbf{u})$:

$$\begin{aligned} F(\nabla \mathbf{u}) = \tfrac{4}{3}\left[(\tfrac{\partial u}{\partial x})^2 + (\tfrac{\partial v}{\partial y})^2 + (\tfrac{\partial w}{\partial z})^2\right] + \left(\tfrac{\partial v}{\partial x} + \tfrac{\partial u}{\partial y}\right)^2 + \left(\tfrac{\partial w}{\partial x} + \tfrac{\partial u}{\partial z}\right)^2 \\ + \left(\tfrac{\partial v}{\partial z} + \tfrac{\partial w}{\partial y}\right)^2 - \tfrac{4}{3}\left(\tfrac{\partial u}{\partial x}\tfrac{\partial v}{\partial y} + \tfrac{\partial u}{\partial x}\tfrac{\partial w}{\partial z} + \tfrac{\partial v}{\partial y}\tfrac{\partial w}{\partial z}\right). \end{aligned} \tag{8.27}$$

where $\mathbf{u} = (u, v, w)^T$.

In 2-d, we used triangles to form the unstructured meshes, so in 3-d, the elements are tetrahedra, consisting of four triangular-shaped faces. The generation of 3-d meshes is clearly more complicated than the 2-d case, especially for complex geometries, although unstructured meshes make this task less difficult than if structured grids were used.

The modified Galerkin least-squares finite element method which we have used as the discretization scheme follows directly from 2-d to 3-d. Integration is performed over element volumes rather than areas, but the assembly process works in

exactly the same way. Discretization leads to a system of nonlinear algebraic equations which can be solved using Newton's method with preconditioned GMRES as for 2-d problems.

So solution of 3-d flows on fixed meshes introduces few extra complications, except that the process is far more expensive, since many more nodes are required in 3-d than 2-d for a given resolution. This emphasises the need for some form of adaptivity so that solution can made more efficient.

Both forms of adaptive refinement discussed in chapters 5 (h-refinement) and 6 (r-refinement) may be extended to 3-d. Node addition involves sub-division of the tetrahedra into eight smaller elements, with alternative types of sub-division used to avoid the appearance of hanging nodes at edge midpoints. In [114], Speares and Berzins present an algorithm to carry out h-refinement for 3-d problems using this approach. Node movement in 3-d should be very similar to the 2-d technique, though extra care may be needed to ensure that node tangling does not occur.

In theory at least, the methods we have used in 2-d generally extend to 3-d, and the benefits of adaptivity and unstructured meshes become more pronounced when solving problems in three dimensions.

Chapter 9

Summary

In this thesis we have presented a set of methods for the efficient solution of a particular system of p.d.e.'s; the compressible Navier-Stokes equations. These have been combined in a numerical code which solves compressible flow problems at low to moderate Reynolds numbers, with freestream Mach numbers in the range 0.5–3. The code consists of the following components.

At the heart of the approach is a finite element method which is stable and accurate for flows which are dominated by convection terms. The scheme is based upon the Galerkin least-squares method of Hughes [66], but simplified so that it may be applied to nonsymmetric formulations of the equations (such as a primitive or conserved variable formulation). The discontinuity capturing term is omitted and a less complex parameter τ (which controls the size of the least-squares term) is used. We also decouple the space and time discretizations, so that the temporal derivatives are dealt with separately (the method of lines approach). This allows the use of a backward Euler local time-stepping scheme to be used for solving steady problems, resulting in rapid convergence to steady state.

The nonlinear solver used to solve the resulting algebraic equations is an inexact Newton method so that the linear system at each iteration is not always solved exactly, but without affecting convergence at each time-step. The GMRES iterative solver is used to solve the linear system, but a preconditioner (an incomplete LU factorization) is required so that the memory requirements of GMRES are not excessive and convergence may be accelerated. The formation of the Jacobian matrix at each Newton step is carried out via assembly of approximate element Jacobians, thus achieving a decrease in CPU time (compared to other techniques)

and avoiding the need to analytically code the derivatives (although this is also done for the purposes of comparison).

On fixed, unstructured meshes, the above methods may be used to converge to steady solutions quickly, for problems involving transonic flow and Reynolds numbers in the range 73–2000. Results for the test cases tried appear to lie within the range of results previously obtained [18], even when using the simplified form of the least-squares parameter τ (there seems to be little difference when a more complex form is implemented). In addition, provided the mesh is not too fine, we can solve the steady problem directly without time-stepping (i.e. in one nonlinear solve). However to fully resolve flow features, such as shocks and wakes, a finer mesh is needed, which can be unnecessarily expensive if it is of uniform density throughout the entire domain.

An efficient means of solution is to use adaptivity. We make use of the stability of the Galerkin least-squares method to obtain solutions free of spurious oscillations on a coarse mesh and adapt this mesh accordingly. The mesh is refined dynamically during solution so that the final mesh reflects the particular flow being solved.

We combine two types of refinement, the first being addition of extra nodes (h-refinement). An element error indicator based on the residual obtained from inserting the current solution into the original p.d.e.'s (though other indicators are considered) determines where extra nodes are to be added. This gives a form of error control to the method, but, on its own, leads to meshes over-dependent on the original coarse mesh structure and shape.

The second form of refinement is movement of existing nodes (r-refinement). Each node is relocated to a new position based on an average of the neighbouring elements, weighted by the element error indicator. This does not appear to be very suitable for problems we consider when used as the sole refinement technique, but complements h-refinement very well, as it removes the dependence of the final mesh on the initial mesh. It also allows the stretching of elements in flow features such as wakes, shocks and boundary layers. Currently the effect of this stretching is limited, though it may be possible to enhance this.

The backward Euler time-stepping scheme which we use for steady problems is not suitable for transient flows where time-accuracy is needed. Instead we need to use a more accurate o.d.e. solver to solve the semi-discretized system of equations. The adaptive methods also need modifying, as removal of nodes, as well as addition,

is needed, and this prevents node movement from being carried out in the same way as before. Investigation of this approach indicates that accurate interpolation is needed when computing nodal values on refined meshes, in order to avoid a sharp fall the current time-step size, which is determined adaptively. However, an interpolant such as piecewise Hermite cubic approximation requires estimates of partial derivatives at nodes, and we are currently unable to find an accurate method to do this on unstructured meshes. In the absence of a cheap, accurate interpolant on unstructured meshes, we conclude that the method of lines might not be the best approach for this type of problem.

We have used the methods outlined above for relatively simple flows (i.e. laminar two-dimensional flow around aerofoils) to obtain results efficiently. It is expected that these techniques, suitably modified, should be equally applicable to more realistic problems, such as those involving turbulence or three dimensional geometries.

Bibliography

[1] J.D. Anderson. *Fundamentals of Aerodynamics*. McGraw-Hill, 1984.

[2] W.E. Arnoldi. The principle of minimized iteration in the solution of the matrix eigenvalue problem. *Quart. Appl. Math.*, 9:17–29, 1951.

[3] O. Axelsson. Incomplete block matrix factorization preconditioning methods. The ultimate answerΓ *J. Comput. Appl. Math.*, 12&13:3–18, 1985.

[4] I. Babuska. The finite element method with Lagrangian multipliers. *Numer. Math.*, 20:179–192, 1973.

[5] I. Babuska and W.C. Rheinboldt. A posteriori error estimates for the finite element method. *International Journal for Numerical Methods in Engineering*, 12:1597–1615, 1978.

[6] I. Babuska and W.C. Rheinboldt. Reliable error estimation and mesh adaption for the finite element method. In *Computational Methods in Nonlinear Mechanics*, pages 67–108. North-Holland, New York, 1980.

[7] M.J. Baines. *Moving Finite Elements*. Oxford University Press, 1994.

[8] C. Baiocchi, F. Brezzi, and L.P. Franca. Virtual bubbles and Galerkin-least-squares type methods (Ga.L.S.). *Computer Methods in Applied Mechanics and Engineering*, 105:125–141, 1993.

[9] B.S. Baldwin and T.J. Barth. A one-equation turbulence transport model for high Reynolds number wall-bounded flows. Technical Report 91-0610, AIAA, 1991.

[10] B.S. Baldwin and H. Lomax. Thin-layer approximation and algebraic model for separated turbulent flows. 78-257, AIAA, 1978.

[11] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods.* SIAM, 1993.

[12] G.K. Batchelor. *An Introduction to Fluid Mechanics.* Cambridge, 1967.

[13] M. Berzins and R.M. Furzeland. A user's manual for SPRINT - A versatile software package for solving systems of algebraic, ordinary and partial differential equations: Part 1 - algebraic and ordinary differential equations. Technical Report TNER.85.058, Thornton Reserch Centre, Shell Research Limited, 1985.

[14] M. Berzins and R.M. Furzeland. An adaptive theta method for the solution of stiff and nonstiff differential equations. *Applied Numerical Mathematics*, 9:1–19, 1992.

[15] F. Brezzi, M.O. Bristeau, L.P. Franca, M. Mallet, and G. Roge. A relationship between stabilized finite element methods and the Galerkin method with bubble functions. *Computer Methods in Applied Mechanics and Engineering*, 96:117–129, 1992.

[16] W.L. Briggs. *A Multigrid Tutorial.* SIAM, 1987.

[17] M.O. Bristeau, R. Glowinski, L. Dutto, J. Periaux, and G. Roge. Compressible viscous flow calculations using compatible finite element approximations. *International Journal for Numerical Methods in Fluids*, 11:719–749, 1990.

[18] M.O. Bristeau, R. Glowinski, J. Periaux, and H. Viviand, editors. *Numerical Simulation of Numerical Navier-Stokes Flows, A GAMM Workshop.* Freidr. Vieweg and Sohn, Braunschweig/Weisbaden, 1987.

[19] A.N. Brookes and T.J.R. Hughes. Streamline upwind/Petrov-Galerkin methods for advection-dominated flows. In *Proceedings Third International Conference on Finite Element in Fluid Flows*, Canada, 1980. Banff.

[20] P.N. Brown. A local convergence theory for combined inexact-Newton/finite difference projection methods. *SIAM Journal of Numerical Analysis*, 24:407–434, 1987.

[21] P.N. Brown and Y. Saad. Hybrid Krylov methods for nonlinear systems of equations. Technical report, Laurence Livermore National Laboratory, November 1987.

[22] R.L. Burden and J.D. Faires. *Numerical Analysis.* Prindle, Weber and Schmidt, Boston, Mass., 4th edition, 1989.

[23] G.D. Byrne and A.C. Hindmarsh. Stiff ODE solvers: A review of current and coming attractions. *Journal of Computational Physics*, 70:1–62, 1987.

[24] G.F. Carey. Derivative calculation from finite element solutions. *Computer Methods in Applied Mechanics and Engineering*, 35:1–14, 1982.

[25] J.E. Carter. Numerical solutions of the Navier-Stokes equations for the supersonic laminar flow over a two-dimensional compression corner. Technical Report NASA TR R-385, NASA, July 1972.

[26] CDR, Innovation Centre, University College Swansea. *The Provison of an Unstructured Grid Capability for Modelling High Mach Number Flows. Issue 1.0*, December 1990.

[27] T. Cebeci and P. Bradshaw. *Momentum Transfer in Boundary Layers.* McGraw-Hill, New York, 1977.

[28] T. Cebeci and A.M.O. Smith. A finite-difference method for calculating compressible laminar and turbulent boundary layers. *Journal of Basic Engineering*, 92(3):523–535, 1970.

[29] T.F. Chan, R. Glowinski, J. Periaux, and O.B. Widlund, editors. *Domain Decomposition Methods. Proceedings of the Second International Symposium on Domain Decomposition Methods.* SIAM, 1989.

[30] K.Y. Chien. Predictions of channel and boundary-layer flows with a low-Reynolds-number turbulence model. *AIAA Journal*, 20:33–38, 1982.

[31] P. Concus, G. Golub, and G. Meurant. Block preconditioning for the conjugate gradient method. *SIAM Journal on Scientific and Statistical Computing*, 6:220–252, 1985.

[32] G.R. Cowper. Gaussian quadrature formulae for triangles. *International Journal for Numerical Methods in Fluids*, 7:405–408, 1973.

[33] J.M. Coyle, J.E. Flaherty, and R. Ludwig. On the stability of mesh equidistribution strategies for time-dependent partial differential equations. *Journal of Computational Physics*, 62:26–39, 1986.

[34] R.S. Dembo, S.C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM Journal of Numerical Analysis*, 19(2):400–408, 1982.

[35] L. Demkowicz, J.T. Oden, and W. Rachowicz. A new finite element method for solving compressible Navier-Stokes equations based on an operator splitting method and h-p adaptivity. *Computer Methods in Applied Mechanics and Engineering*, 84:275–326, 1990.

[36] L. Demkowicz, J.T. Oden, W. Rachowicz, and O. Hardy. An h-p Taylor-Galerkin finite element method for compressible Euler equations. *Computer Methods in Applied Mechanics and Engineering*, 88:363–396, 1991.

[37] J.E. Dennis and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice Hall, Englewood Cliffs, NJ, 1983.

[38] J. Donea. A Taylor-Galerkin method for convective-transport problems. *International Journal for Numerical Methods in Engineering*, 20:101–120, 1984.

[39] I. Duff, R. Grimes, and J. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15:1–14, 1989.

[40] L.C. Dutto. The effect of ordering on preconditioned GMRES algorithm for solving the compressible Navier-Stokes equations. *International Journal for Numerical Methods in Engineering*, 36:457–497, 1993.

[41] K. Erikkson and C. Johnson. Adaptive streamline diffusion finite element methods for stationary convection-diffusion problems. *Mathematics of computation*, 60:167–188, 1993.

[42] K. Eriksson and C. Johnson. An adaptive finite element method for linear elliptic problems. *Mathematics of Computation*, 50:361–383, 1988.

[43] K. Eriksson and C. Johnson. Adaptive finite element methods for parabolic problems. IV. nonlinear problems. *SIAM Journal of Numerical Analysis*, To appear, 1995.

[44] B. W. Char et al. *MAPLE Reference Manual.* WATCOM Publications Ltd., Waterloo, Canada, 5th edition, 1988.

[45] R. Freund and N. Nachtigal. QMR: A quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, 60:315–339, 1991.

[46] R.J. Gelinas, S.K. Doss, and K. Miller. The moving finite element method: applications to general partial differential equations with multiple large gradients. *Journal of Computational Physics*, 40(202-249), 1981.

[47] A. George. Nested dissection of regular finite element meshes. *SIAM Journal of Numerical Analysis*, 11:345–363, 1973.

[48] M.M. Gibson and B.E. Launder. Ground effects on pressure fluctuations in the atmospheric boundary layer. *Journal of Fluid Mechanics*, 86:491–511, 1978.

[49] K. Godunov. Finite-difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics. *Mat. Sbornik*, 47:271–306, 1959. (In Russian).

[50] A.A. Goldstein. *Constructive Real Analysis.* Harper and Row, New York, 1967.

[51] G. Goodsell and J.R. Whiteman. Superconvergent recovered gradient functions. *International Journal for Numerical Methods in Engineering*, 27:469–481, 1989.

[52] P.M. Gresho and R.L. Lee. Don't surpress the wiggles - they're telling you something! In T.J.R. Hughes, editor, *Finite Element methods for Convection Dominated Flows, AMD Vol. 34*, pages 37–61, New York, 1979. ASME.

[53] M.D. Gunzburger. *Finite Elements for Viscous Incompressible Flows: a Guide to Theory, Practice and Algorithms.* Academic Press, 1989.

[54] I. Gustafsson. A class of first-order factorization methods. *BIT*, 18:142–156, 1978.

[55] I. Gustafsson and G. Lindskog. A preconditioning technique based on element matrix factorizations. *Computer Methods in Applied Mechanics and Engineering*, 55:201–220, 1986.

[56] P. Hansbo. Explicit streamline diffusion finite element methods for the compressible Euler equations in conservation variables. *Journal of Computational Physics*, 109(2):274–288, 1993.

[57] P. Hansbo and C. Johnson. Adaptive streamline diffusion methods for compressible flow using conservation variables. *Computer Methods in Applied Mechanics and Engineering*, 87:267–280, 1991.

[58] P. Hansbo and A. Szepessy. A velocity-pressure streamline diffusion finite element method for the incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 84:175–192, 1990.

[59] A. Harten. On the symmetric form of systems of conservation laws with entropy. *Journal of Computational Physics*, 49:151–164, 1983.

[60] O. Hassan, K. Morgan, J. Peraire, E.J. Probert, and R.R. Thareja. Adaptive unstructured mesh methods for steady viscous flow. Technical Report AIAA-91-1538-CP, AIAA, 1991.

[61] G. Hauke and T.J.R. Hughes. A unified approach to compressible and incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 113:389–395, 1994.

[62] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand.*, 49:409–436, 1952.

[63] J.O. Hinze. *Turbulence: an Introduction to its Mechanism and Theory.* McGraw-Hill, 1975.

[64] M.E. Hubbard. Grid adaptation and multidimensional upwinding. Numerical analysis report 8/94, Department of Mathematics, University of Reading, 1994.

[65] T.J.R. Hughes and A.N. Brooks. A multidimensional upwind scheme with no crosswind diffusion. In T.J.R. Hughes, editor, *Finite Element Methods for Convection Dominated Flows, AMD Vol. 34*, pages 19–35. ASME, New York, 1979.

[66] T.J.R. Hughes, L.P. Franca, and G.M. Hulbert. A new finite element formulation for computational fluid dynamics: VIII. The Galerkin/least-squares method for advective diffusive equations. *Computer Methods in Applied Mechanics and Engineering*, 73:173–189, 1989.

[67] T.J.R. Hughes, L.P. Franca, and M. Mallet. A new finite element formulation for computational fluid dynamics: I. Symmetric forms of the compressible Euler and Navier-stokes equations and the second law of thermodynamics. *Computer Methods in Applied Mechanics and Engineering*, 54:223–234, 1986.

[68] T.J.R. Hughes, L.P. Franca, and M. Mallet. A new finite element formulation for computational fluid dynamics: VI. Convergence analysis of the generalized SUPG formulation for linear time dependent multi-dimensional advective-diffusive systems. *Computer Methods in Applied Mechanics and Engineering*, 63:97–112, 1987.

[69] T.J.R. Hughes, I. Levit, and J. Winget. An element-by element solution algorithm for problems of structural and solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 36:241–254, 1983.

[70] T.J.R. Hughes and M. Mallet. A new finite element formulation for computational fluid dynamics: III. The generalized streamline operator for multidimensional advective-diffusive systems. *Computer Methods in Applied Mechanics and Engineering*, 58:305–328, 1986.

[71] T.J.R. Hughes and M. Mallet. A new finite element formulation for computational fluid dynamics: IV. A discontinuity-capturing operator for multidimensional advective-diffusive systems. *Computer Methods in Applied Mechanics and Engineering*, 58:329–336, 1986.

[72] B. Irons. A frontal solution program for finite element analysis. *International Journal for Numerical Methods in Engineering*, 2(5-32), 1970.

[73] A. Jameson and D. Mavriplis. Finite volume solution of the two-dimensional Euler equations on a regular triangular mesh. *AIAA Journal*, 24(4):611–618, 1986.

[74] A. Jameson, W. Schmidt, and E. Turkel. Numerical solution of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes. Technical Report 81-1259, AIAA, 1981.

[75] K. Jansen, Z. Johan, and T.J.R. Hughes. Implementation of a one-equation turbulence model within a stabilized finite element formulation of a symmetric advective-diffusive system. *Computer Methods in Applied Mechanics and Engineering*, 105:405–433, 1993.

[76] P.K. Jimack. Adaptive error control in the finite element method for time-dependent problems. Technical Report 92.11, School of Computer Studies, University of Leeds, 1992.

[77] C. Johnson. *Numerical Solutions of Partial Differential Equations by the Finite Element Method.* Cambridge University Press, 1987.

[78] C. Johnson, U. Navert, and J. Pitkaranta. Finite element methods for linear hyperbolic problems. *Computer Methods in Applied Mechanics and Engineering*, 45:285–312, 1984.

[79] C. Johnson and J. Saranen. Streamline diffusion methods for problems in fluid mechanics. *Mathematics of Computation*, 47:1–18, 1986.

[80] C. Johnson and A. Szepessy. Adaptive finite element methods for conservation laws based on a posteriori error estimates. *Communications on Pure and Applied Mathematics*, XLVIII(3):199–234, 1995.

[81] C. Johnson, A. Szepessy, and P. Hansbo. On the convergence of shock capturing streamline diffusion finite element methods for hyperbolic conservation laws. *Mathematics of Computation*, 54:107–129, 1990.

[82] D.A. Johnson and J.S. King. A mathematically simple turbulence closure model for attached and separated turbulent boundary layers. *AIAA Journal*, 23(11):1684–1692, 1985.

[83] I.W. Johnson, A.J. Wathan, and M.J. Baines. Moving finite element methods for evolutionary problems . II. applications. *Journal of Computational Physics*, 79:270–297, 1988.

[84] W.P. Jones and B.E. Launder. The prediction of laminarization with a 2-equation model of turbulence. *International Journal of Heat and Mass Transfer*, 15:301–314, 1972.

[85] M. Krizek. Superconvergence phenomena in the finite element method. *Computer Methods in Applied Mechanics and Engineering*, 116:157–163, 1994.

[86] J.D. Lambert. *Numerical Methods for Ordinary Differential Systems.* Wiley, Chichester, 1991.

[87] P. Lancaster and K. Salkauskas. *Curve and Surface Fitting - An Introduction.* Academic Press, 1986.

[88] C. Lanczos. Solutions of systems of linear equations by minimized iterations. *J. Res. Natl. Bur. Stand.*, 49:33–53, 1952.

[89] R. Lohner. An adaptive finite element scheme for transient problems in CFD. *Computer Methods in Applied Mechanics and Engineering*, 61:323–338, 1987.

[90] R. Lohner, K. Morgan, and O.C. Zienkiewicz. Adaptive grid refinement for the compressible Euler equations. In I. Babuska, editor, *Accuracy estimates and adaptive refinements in finite element computations*, chapter 15, pages 281–297. John Wiley, 1986.

[91] B.J. Lucier. A moving mesh numerical method for hyperbolic conservation laws. *Mathematics of Computation*, 46(173):59–69, 1986.

[92] D.J. Mavriplis. Multigrid solution of the two-dimensional Euler equations on unstructured triangular meshes. *AIAA Journal*, 26(7):824–831, 1988.

[93] D.J. Mavriplis. Turbulent flow calculations using unstructured and adaptive meshes. *International Journal for Numerical Methods in Fluids*, 13:1131–1152, 1991.

[94] J.A. Meijerink and H.A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, 31(137):148–162, 1977.

[95] J.A. Meijerink and H.A. van der Vorst. Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *Journal of Computational Physics*, 44:134–155, 1981.

[96] K. Miller. Recent results on finite element methods with moving nodes. In I. Babuska, editor, *Accuracy estimates and adaptive refinements in finite element computations*, chapter 18, pages 325–338. John Wiley, 1986.

[97] K. Miller and R.N. Miller. Moving finite elements. I. *SIAM Journal of Numerical Analysis*, 18(6):1019–1032, 1981.

[98] A.R. Mitchell and R. Wait. *The Finite Element Method in Partial Differential Equations*. Wiley, 1977.

[99] R.A. Mitcheltree, M.D. Salas, and H.A. Hassan. One-equation turbulence model for transonic airfoil flows. *AIAA Journal*, 28(9):1625–1632, 1990.

[100] J.T. Oden, T. Strouboulis, and P.H. Devloo. Adaptive finite elements for high-speed compressible flow. *International Journal for Numerical Methods in Fluids*, 7:1211–1228, 1987.

[101] M.E. O'Neill and F. Chorlton. *Viscous and Compressible Fluid Mechanics*. Ellis Horwood, 1989.

[102] R. Peyret and T.D. Taylor. *Computational Methods for Fluid Flow*. Springer-Verlag, 1983.

[103] P.L. Roe. Approximate Riemann solvers, parameter vectors and difference schemes. *Journal of Computational Physics*, 43:357–372, 1981.

[104] Y. Saad and M.H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 7(3):856–869, 1986.

[105] M. Seager. A SLAP for the masses. Technical Report UCRL-100267, Lawrence Livermore National Laboratory, December 1988.

[106] F. Shakib and T.J.R. Hughes. A new finite element formulation for computational fluid dynamics: IX. Fourier analysis of space-time Galerkin/least-squares algorithms. *Computer Methods in Applied Mechanics and Engineering*, 87:35–58, 1991.

[107] F. Shakib, T.J.R. Hughes, and Z. Johan. A multi-element group preconditioned GMRES algorithm for nonsymmetric systems arising in finite element analysis. *Computer Methods in Applied Mechanics and Engineering*, 75:415–456, 1989.

[108] F. Shakib, T.J.R. Hughes, and Z. Johan. A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 89:141–219, 1991.

[109] G.D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods.* Oxford, 1985.

[110] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 10:36–52, 1989.

[111] A. Soulaimani and M. Fortin. Finite element solution of compressible viscous flows using conservative variables. *Computer Methods in Applied Mechanics and Engineering*, 118:319–350, 1994.

[112] Sowerby Research Centre. *User Guide for the Scientific Visualisation Package Viz*, April 1993.

[113] P.R. Spalart and S.R. Allmaras. A one-equation turbulence model for aerodynamic flows. 92-0439, AIAA, 1992.

[114] W. Speares and M. Berzins. A 3D unstructured mesh adaptation algorithm for time dependent shock dominated problems. Technical Report Report 95.33, School of Computer Studies, University of Leeds, 1995.

[115] W.G. Strang and G.J. Fix. *An Analysis of the Finite Element Method.* Prentice-Hall, 1973.

[116] T. Strouboulis and J.T. Oden. A posteriori error estimates of finite element approximations in fluid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 78:201–247, 1990.

[117] W.W. Tworzydlo, J.T. Oden, and E.A. Thornton. Adaptive implicit/explicit finite element methods for compressible viscous flows. *Computer Methods in Mechanics and Engineering*, 95:397–440, 1992.

[118] H.A. van der Vorst. Iterative solution methods for certain sparse linear systems with a non-symmetric matrix arising from PDE-problems. *Journal of Computational Physics*, 44:1–19, 1981.

[119] H.A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 13:631–644, 1992.

[120] M.B. van Gijzen. An analysis of element-by-element preconditioners for non-symmetric problems. *Computer Methods in Applied Mechanics and Engineering*, 105:23–40, 1993.

[121] R. Varga. *Matrix Iterative Analysis*. Prentice-Hall Inc, Englewood Cliffs, NJ, 1962.

[122] V. Venkatakrishnan and D.J. Mavriplis. Implicit solvers for unstructured meshes. *Journal of Computational Physics*, 105:83–91, 1993.

[123] J.M. Ware. *The Adaptive Solution of Time-Dependent Partial Differential Equations in Two Space Dimensions*. PhD thesis, School of Computer Studies, University of Leeds, 1993.

[124] A.J. Wathen. An analysis of some element-by-element techniques. *Computer Methods in Applied Mechanics and Engineering*, 74:271–287, 1989.

[125] N.P. Weatherill. Grid generation: Structured, unstructured or bothΓ In P.Stow, editor, *Computational Methods in Aeronautical Fluid Dynamics*. Oxford Science Publications, 1990.

[126] N.P. Weatherill and M.J. Marchant. A streamline based approach to the adaptation of unstructured grids for viscous flow simulation. In *ICFD Conference on Numerical Methods for Fluid Dynamics*, 1995.

[127] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley, 2nd edition, 1991.

[128] M. Wolfshtein. The velocity and temperature distribution in one-dimensional flow with turbulence augmentation and pressure gradient. *International Journal of Heat and Mass Transfer*, 12:301–312, 1969.

[129] P. Zegeling. *Moving Grid methods for time-dependent partial differential equations.* PhD thesis, University of Amsterdam, 1992.