OBJECT-ORIENTED IMPLEMENTATION OF A COMPUTATIONAL FLUID DYNAMICS METHOD

A thesis submitted for the degree of

Master of Engineering

By

Stephen McMahon

B.Sc.(Hons)

School of Aerospace and Mechanical Engineering,

University College,

The University of New South Wales,

Australian Defence Force Academy.

June 2000

Abstract

An algorithm for flow simulation, based on convection of the fundamental flow quantities mass, momentum and energy, has been described by Milthorpe [22]. The algorithm, originally implemented using FORTRAN, has been re-implemented using the object oriented programming language C++. The aim was to produce a computer program that is more flexible and extensible, allowing further modifications to be carried out in a more efficient manner.

This new computer code was used to simulate an experiment, outlined by Skinner [31], that investigated the early arrival of driver gas at the test section of a shock tunnel.

While tailoring the computer program to the driver gas contamination problem it was discovered that a certain set of parameters needed to be tuned in order to model the problem more accurately. An automated tuning process was used to accomplish this.

The functionality of Milthorpe's original computer program was extended to included parallel processing. The computer program was also made portable so that it could run on several types of computers and operating systems to make the parallel processing capabilities more flexible.

The resulting computer code was able to simulate many aspects of the flow in a shock tunnel but there was not enough accuracy to draw any conclusions about driver gas contamination. It was indeed more efficient to add new features to the new code.

Certificate of Originality

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of a university or other institute of higher learning, except where due acknowledgement is made in the text.

I also declare that the intellectual content of this thesis is the product of my own work, even though I may have received assistance from others on style, presentation and language expression.

Stephen McMahon

Acknowledgements

I wish to thank my supervisor Dr. John Milthorpe for his guidance and encouragement; my wife Patty for her endless patience and support; Dr. Sam Mallinson and Dr. Phil Palma for their advice; and all my friends and family for their understanding.

Contents

List of Figures

Chapter 1

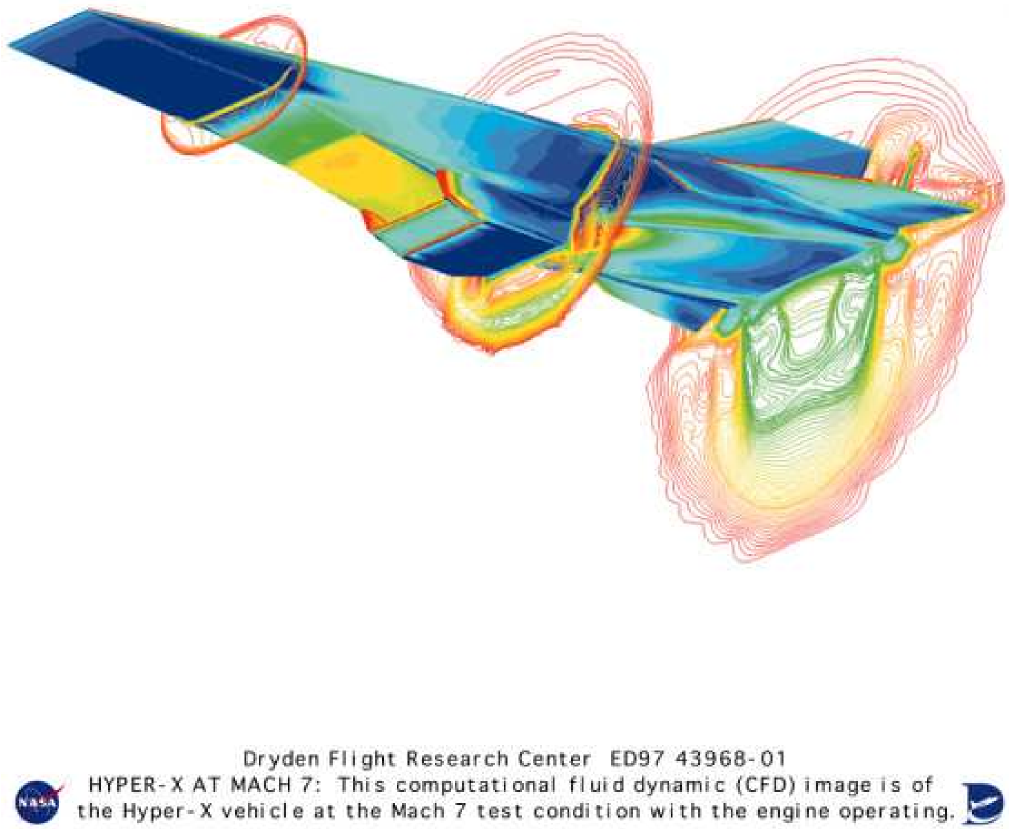
Introduction

1.1 Hypersonic aircraft and spacecraft

The world's fastest air-breathing aircraft, the SR-71, cruises slightly above Mach 3. New air breathing aircraft capable of flying at hypersonic speeds (greater than Mach 5) are currently being developed, which will make use of an experimental engine called a SCRAMJET (supersonic combusting ramjet) [10]. A SCRAMJET starts operating at speeds over about Mach 7. SCRAMJETs use the speed of the aircraft to compress air, which then combusts with the fuel, usually hydrogen. An aircraft using this technology could be used as a single-stage-to-orbit (SSTO) space launch vehicle. Such an aircraft could take off at a conventional airport. Using its SCRAMJET, it could climb to a sufficient altitude and speed to engage a more conventional rocket which would take it into Earth orbit. Since there are no expendable stages and since they could use existing airports, vehicles such as this could greatly reduce the cost of access to space. A hypersonic aircraft could also be used as a fast, commercial intercontinental transport because of the speeds they attain.

A prototype of one such vehicle, which should start to undergo flight tests in 2000, is HyperX [4] (show here in Figure 1.1). HyperX is intended to be a demonstration of air-breathing hypersonic flight. It will be attached to a booster rocket which itself is launched from an aircraft. When the booster completes its burn the SCRAMJET will be started and the test flight of the hypersonic vehicle will begin.

Figure 1.1: HyperX at Mach 7 (reproduced from Dryden Flight Research Center [3])



Another example of a flight test of a SCRAMJET is the HyShot project [21], which is being undertaken by the University of Queensland. The purpose of the HyShot project is to compare results from a SCRAMJET flight test against those from shock tunnel experiments and those generated in computer simulations. The HyShot project launches a SCRAMJET engine as part of the payload of a rocket. When the trajectory of the rocket reaches Mach 8, which is actually when it is on the way down, the SCRAMJET experiment will begin.

Testing SCRAMJET designs is just one application of a shock tunnel. A shock tunnel is an aerodynamic testing facility whose test times are measured in fractions of a second. The short test times keep the energy cost of running the facility down as well as preserving the facility and test models from over heating. A diaphragm separating a high pressure driver gas from the test gas is ruptured causing a shock wave to be sent through the test gas. This shock wave travels down a tube and reflects at the end, creating a slug of high pressure and high temperature test gas which then expands through a nozzle into the test section. The reflected shock can interact with driver gas behind the interface sending disturbances back into the test gas. To avoid this a driver gas with the right speed of sound is chosen. The required speed of sound of the driver gas increases with the speed of the primary shock. Shock tunnels are discussed in greater detail in chapter 3.

1.2 The role of Computational Fluid Dynamics, CFD

Hypersonic flight experiments allow engineers and researchers to investigate all aspects of aerodynamics relating to the prototype vehicle. The problem is that they are costly and time consuming. To reduce future costs, one aim of these kind of flight experiments is to allow engineers to validate computational fluid dynamics (CFD) models of the engine designs. With working and reliable CFD simulations-tions available, further development of SCRAMJETs and hypersonic aircraft can be validated at a greatly reduced cost.

For many subsonic flow problems, such as those encountered in conventional aircraft, the numerical methods available are simpler and more robust. In these cases CFD simulations can largely replace wind tunnel experiments.

Hypersonic flow problems, on the other hand, present greater problems for both wind tunnel experiments and CFD simulations. There are much higher speeds and temperatures involved. The flow is more complex with shock waves introducing their own aerodynamic effects. The increased energy of the flow itself can cause the molecules in air to dissociate and ionise, which further destabilises the flow. Aerodynamic heating of an aircraft at these velocities also presents further design problems.

Experiments in shock tunnels can present researchers with some difficulties, such as short test times and the presence of debris from a burst metal diaphragm. Various experiments have shown that the driver gas contaminates the test flow earlier than expected [31]. Thus, it is useful to validate shock tunnel test data against CFD simulations to rule out or explain unwanted effects that may be present in the results.

CFD calculations to simulate hypersonic flight are much more complex than their subsonic counterparts. The model equations being solved are hyperbolic and hence non-linear. Shock waves, which appear as discontinuities in flow solutions, present a range of problems to numerical researchers. Such problems include oscillation or instability in the solutions. To accurately simulate hypersonic flow, chemical reactions must also be taken into consideration.

The comparison of results obtained by numerical and experimental hypersonic researchers is a two way process. Each validates the other and increases the understanding of the problems involved.

1.3 Outline of the current work

Milthorpe [23] developed an algorithm for modelling unsteady flows containing discontinuities, such as those found in shock tunnel research. The algorithm he uses is a kind of direct numerical simulation and calculates flow values based on the conservation of mass, momentum and energy. His computer program has been used as a comparison for a number of shock tunnel experiments [11],[18],[26].

The current work is an extension of Milthorpe's method. The primary aim was to produce a computer program that was more flexible and extensible, allowing further modifications and extensions to be made in an efficient manner. To achieve this it was decided to implement the algorithm in C++, an object-oriented and modular computer language [33].

The functionality of Milthorpe's original computer program was extended to included parallel processing. One reason for this was to investigate the ease with which the new version of the computer program could be extended. Another reason is that a computer program that can make use of parallel processing takes much less time to complete when run on a set of networked workstations or on a multiprocessor computer. The computer program was also made portable so that it could run on several types of computers and operating systems to make the parallel processing capabilities more flexible.

To contribute in another way to the field of hypersonics research, and to validate the computer code with a new problem, a particular shock tunnel simulation problem was tackled. The problem of driver gas contamination (discussed in more detail in Section 3.3) in the shock tunnel test section was modelled, in the hope that experimentalists might gain some useful insights to the problem. The driver gas contamination problem involves several interesting flow features such as shock waves, shock reflection, shock-boundary layer interaction and contact surface mixing, making this a good test for the code. To simulate this problem, the computer code had to deal with interactions between two species of gas - the driver gas and the test gas.

While tailoring the computer program to the driver gas contamination problem, it was discovered that a certain set of parameters needed to be tuned in order to model the problem more accurately. It was decided to automate the tuning process because there were a number of parameters. A simplified version of the problem had to be run to test the parameters as they were changed. Included in this work is a description of the tuning algorithm used and the results it produced.

Chapter 2

Basic Principles

2.1 Fluid dynamics

To construct a model for fluid motion one looks at properties of matter in a control volume. These properties of a fluid are often expressed in terms of mass, momentum and energy. In most cases the fluid is considered to be a continuum, whereas for rarefied gases one needs to take into account the behaviour of molecules in a statistical way [15, p. 34].

To arrive at a mathematical model of fluid motion one looks at the properties of mass, momentum and energy in a control volume (Figure 2.1) and considers the following physical laws:

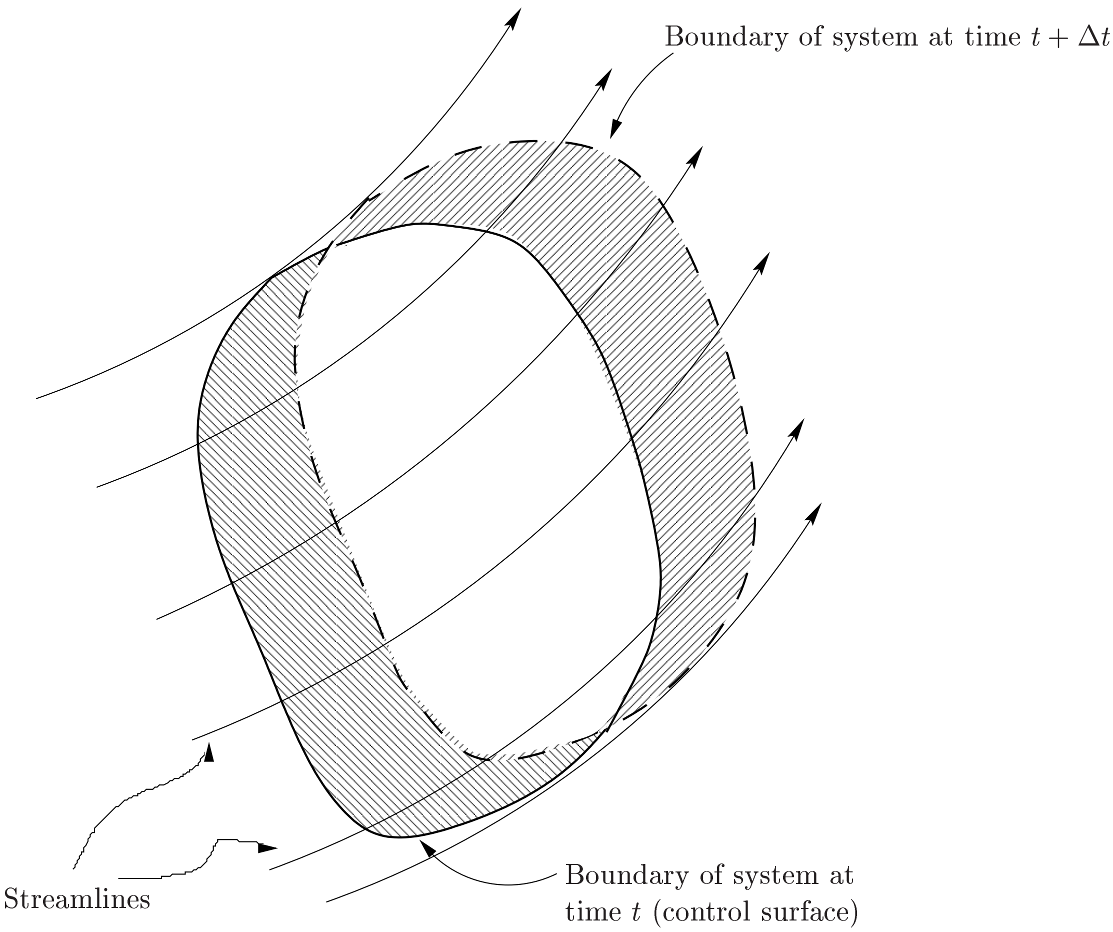
1. Conservation of mass,
2. Newton's second law of motion,
3. Conservation of energy, and
4. Second law of thermodynamics.

2.1.1 Conservation of mass

An expression for the conservation of mass is arrived at by considering the rate of change of mass in the control volume; that is, the flow of mass in and the flow of mass out of our control volume [15]. In differential form this is

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0 \tag{2.1}$$

Figure 2.1: Fluid control volume



where ρ is the density and $\mathbf{V}$ is the velocity vector. Equation 2.1 is called the continuity equation.

If the flow is incompressible this becomes:

$$\nabla \cdot \mathbf{V} = 0 \tag{2.2}$$

2.1.2 Navier-Stokes equation

Equation 2.3, generally known as the Navier-Stokes equation, is obtained by considering the effect of Newton's second law of motion ($F = ma$) on the control volume and by taking into account all the forces acting upon it [15, p. 54]. It is expressed here in cartesian tensor notation.

$$\rho \left(\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} \right) = -\frac{\partial p}{\partial x_i} + B_i + \frac{\partial}{\partial x_j} \left[\mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3} \delta_{ij} \frac{\partial u_k}{\partial x_k} \right) \right]$$

$$+\frac{\partial}{\partial x_i}\left(\zeta\frac{\partial u_k}{\partial x_k}\right) \tag{2.3}$$

B_i represents body forces, μ is the coefficient of viscosity and ζ is the second coefficient of viscosity. ζ is zero for a monatomic gas.

The Navier-Stokes equation is often simplified for various fluid dynamics problems by using the incompressible condition (Equation 2.2). This reduces the above equation to:

$$\rho\frac{D\mathbf{V}}{Dt} = -\nabla p + \mathbf{B} + \mu\nabla^2\mathbf{V}$$

where D/Dt is called the substantial or material derivative, and is given by:

$$\frac{D\mathbf{V}}{Dt} = \frac{\partial\mathbf{V}}{\partial t} + (\mathbf{V}\cdot\nabla)\,\mathbf{V} \tag{2.4}$$

Another way to simplify solutions to the Navier-Stokes equation is to keep μ constant.

2.1.3 Conservation of energy

The energy equation is derived by looking at the heat flux into and out of the control volume and by looking at the work done on the control volume by pressure and shear stresses [15, p. 56].

$$\rho C_v\frac{DT}{Dt} = -p\nabla\cdot\mathbf{V} + \kappa\nabla^2 T - \nabla\cdot\mathbf{q}_r + \Phi + q'''$$

Here, p is the pressure, C_v is the specific heat at constant volume, T is the temperature, κ is the thermal conductivity, $\mathbf{q}_r$ is the radiation heat flux vector, Φ is the dissipation function and q''' is the internal heat production rate per unit volume.

2.1.4 Equation of state

To fully track the state of a fluid, it is necessary to have an equation of state that relates fluid quantities to each other. For low and moderate density gases, the ideal gas equations (Equations 2.5, 2.6 and 2.7) can be used. In practice, these equations are also used in the solution of problems at higher densities, at the expense of complete accuracy.

The ideal gas equation of state is:

$$p = \rho RT \tag{2.5}$$

where $R = C_p - C_v$ is the gas constant. C_p and C_v are the specific heats at constant pressure and temperature respectively.

For an ideal gas the energy is related to the temperature by:

$$E = \rho C_v T \tag{2.6}$$

and pressure is related to density by:

$$\frac{p}{\rho^\gamma} = constant \tag{2.7}$$

2.2 Current numerical methods

To exactly simulate fluid flow on a computer it would be necessary to solve the above equations with infinite accuracy. In reality, numerical researchers must choose a method to discretise the problem. Some of the general numerical methods used in computational fluid dynamics are described here.

The setting up of a numerical simulation begins with creating a computational grid. The flow variables are calculated at the node points of this grid and, in some methods, at some intermediate points as well. The spacing between grid points has to be fine enough to attain a high enough degree of accuracy. There are advantages, however, to keeping the number of grid points small since more grid points means more computer memory is required and a greater time is needed to perform each iteration of the calculation. Most computer architectures deal with smaller amounts of memory much more efficiently and cheaply than with large amounts of memory. The simplest computational grid is a rectangular lattice with fixed spacing between node points in each dimension. There are a range of methods that use unstructured grids where the density of the node points is not constant and is higher in the regions where more accuracy is required. Unstructured meshes often end up being connected in a triangular or tetrahedral fashion because these shapes fill space well and they require a minimum number of vertices. Some methods even use adaptive meshes where node points are created and destroyed as flow features move through the computational domain. This keeps the total number of nodes to a minimum, whilst still providing the required resolution for certain flow features.

A numerical simulation of a fluid flow problem requires that all flow variables that are tracked by the algorithm are given an initial value. In one scenario, just the free stream flow values are specified. Other scenarios need the specification of an 'initial' condition which is specified at most points in the computational domain and an 'inlet' condition which is specified only along an inlet boundary. Together with the initial conditions, boundary conditions define the nature of the flow problem being simulated. They include inlet conditions, various kinds of walls and outflow conditions. There are also special kinds of conditions that arise from the geometry of the problem like an axi-symmetric boundary.

2.2.1 Discretising the model equations

Finite differences are discrete approximations to the model equations. They are formed by performing a Taylor series expansion of the equations to be solved. All derivatives incorporated in the expansion need to be continuous. The Taylor series can be expanded to attain almost any order of accuracy, however practical methods are only first or second order accurate.

Table 2.1 shows three different approaches to discretising a partial derivative in a space variable [30, p. 18].

As an example the one dimensional model transport equation (Equation 2.8) can be discretised to give Equation 2.9. Equation 2.8 is an expression for the transport of vorticity that is applicable in one dimensional incompressible flow.

Table 2.1: Three methods for discretising

Method	expression	accuracy
Forward difference	$\left.\frac{\delta f}{\delta x}\right\|_{ij} = \frac{f_{i+1,j}-f_{ij}}{\Delta x}$	$O(\Delta x)$
Backward difference	$\left.\frac{\delta f}{\delta x}\right\|_{ij} = \frac{f_{i,j}-f_{i-1,j}}{\Delta x}$	$O(\Delta x)$
Centred difference	$\left.\frac{\delta f}{\delta x}\right\|_{ij} = \frac{f_{i+1,j}-f_{i-1,j}}{2\Delta x}$	$O(\Delta x^2)$

$$\frac{\partial \zeta}{\partial t} = -u\frac{\partial \zeta}{\partial x} + \alpha\frac{\partial^2 \zeta}{\partial x^2} \tag{2.8}$$

$$\frac{\zeta_i^{n+1} - \zeta_i^n}{\Delta t} = -\frac{u\zeta_{i+1}^n - u\zeta_{i-1}^n}{2\Delta x} + \alpha\frac{\zeta_{i+1}^n + \zeta_{i-1}^n - 2\zeta_i^n}{\Delta x^2} \tag{2.9}$$

This is known as the forward time, centred space (FTCS) form of this equation. It is second order accurate ($O(\Delta x^2)$).

2.2.2 Control volume approach to discretisation

Another approach to formulating the finite difference equations is to use a control volume. This process is similar to the way the continuous fluid dynamics equations themselves are formulated (Section 2.1). The advantage of control volume approaches are that they are based on macroscopic physical laws, rather than on continuum mathematics. When flow discontinuities occur (eg. shock waves) a discretised form of a continuous equation often does not represent the feature well. On the other hand, by directly representing each physical law in a finite difference equation, insights may be obtained on how to deal with difficult flow features like shock waves. This approach highlights the 'numerical simulation' process, exemplified by the PIC (Particle-in-Cell) and FLIC (Fluid-in-Cell) codes developed at Los Alamos ([14],[12]). Roache [30, p. 28] gives an example where he arrives at the same equation as Equation 2.9 for vorticity transport by using a control volume approach.

The control volume approach, or direct numerical simulation, is essentially the approach Milthorpe used in formulating his algorithm (Chapter 4).

2.2.3 Method of Characteristics

Roache [30, p. 226] outlined another approach to solving the fluid equations, namely the 'Method of Characteristics' [5]. He states that for supersonic inviscid shock-free flow, where the equations are purely hyperbolic, it is the natural choice for numerical computation. In this well-known method, the computational mesh is not rectangular, nor is its shape known beforehand, but it develops as the solution is explicitly stepped out during the computation. This method gives the most accurate results possible, since the calculation points lie along 'characteristics,' across which derivatives may be discontinuous. The greatest limitation of this method is that viscous effects cannot be included, except by boundary-layer methods. Another disadvantage of this method is the complexity of the process of evaluating the location of solution points from the intersection of the characteristic lines.

2.2.4 Godunov type methods

Methods developed for dealing with incompressible flow can often be used for compressible flow problems. However, problems arise when there are strong shocks or high Reynolds' numbers. Errors, in the form of oscillations, occur close to discontinuities due to limitations in the approximation.

Godunov methods calculate the flow in two different reference frames. These methods are complicated but they do treat flow discontinuities better than some other methods. A solution is first found in a Lagrangian frame and then re-mapped to an Eulerian grid. This method can be thought of as operator splitting based on convective and sound waves.

The following is an outline of the method:

1. Initialisation Step - average the initial distribution over the computational cells:

$$u_j^0 = \frac{1}{\Delta x_j} \int_{x_j - \Delta x_j/2}^{x_j + \Delta x_j/2} u(x) dx. \tag{2.10}$$

2. Reconstruction Step - reconstruct the initial distribution as piecewise polynomials over the computational cells.

$$u_j(x) = P_j(x), \tag{2.11}$$

where $P_j \in [x_j - \Delta x_j/2, x_j + \Delta x_j/2]$ is a polynomial in cell j.

3. Solution in the Small Step - solve the initial-value problem at each cell interface where discontinuities can exist:

$$u(x,t) = E(x, t - t^n) \cdot u(x, t^n), \tag{2.12}$$

where $E(x, t-t^n)$ symbolically represents the evolution operator given by the solution to the Riemann shock-tube problem.

4. Averaging Step - re-average the solution over the grid cells. Given the solution operator in the previous step.
5. Go back to the reconstruction step.

Rider [29] reviewed a number of approximate Riemann solvers that can be used in Step 3. He showed that the choice of a good solver can be used to trade off accuracy with computational efficiency. Care must be taken with some of these solvers because of their inherent smearing of contact discontinuities.

2.2.5 Particle-in-Cell, PIC

Harlow [14] outlined the PIC method, developed at Los Alamos Laboratory in 1955. It is another combined Eulerian-Lagrangian scheme which has been applied successfully to solve a wide variety of multi-fluid problems where the fluid distortions are large. It utilises Lagrangian fluid particles to transport mass, momentum and energy through an Eulerian mesh of cells. While the use of these particles facilitates the calculation of multi-fluid problems, it also results in non-physical fluctuations of the fluid quantities. The dual-mesh system of the PIC method makes great demands on computer memory capacity and on calculation time.

2.2.6 Fluid-in-Cell, FLIC

Gentry, et al. [12] described an Eulerian finite difference method called 'Fluid-in-Cell' that can be used to solve time-dependent equations of motion for the compressible flow of a fluid. It is based on the Particle-in-Cell method (Section 2.2.5), though the transport calculation is different and it does not require the use of particles. Elimination of particles allows solutions that are free of the fluctuations characteristic of PIC. The FLIC method will be described in some detail because it has similarities with Milthorpe's method.

The difference equations

Step 1 Pressure is calculated for each cell and intermediate values for u, v, and I are calculated based on pressure gradient effects alone. An artificial viscosity in the form of a pressure term is included. This viscosity term is of the form:

$$
\begin{aligned}
q^n_{i+\frac{1}{2},j} &= Bc^n_{i+\frac{1}{2},j}\rho^n_{i+\frac{1}{2},j}(u^n_{ij} - u^n_{i+1,j}) && \text{if } K(u^2+v^2)^n_{i+\frac{1}{2},j} < (c^2)^n_{i+\frac{1}{2},j} \\
& && \text{and if } u^n_{ij} > u^n_{i+1,j} \\
&= 0 \text{ otherwise} \\
q^n_{i,j+\frac{1}{2}} &= Bc^n_{i,j+\frac{1}{2}}\rho^n_{i,j+\frac{1}{2}}(v^n_{ij} + v^n_{i,j+1} && \text{if } \mathrm{K}(u^2+v^2)^n_{i,j+\frac{1}{2}} < (c^2)^n_{i,j+\frac{1}{2}} \\
& && \text{and if} v^n_{ij} > v^n_{i,j+1} \\
&= 0 \text{ otherwise}
\end{aligned}
\tag{2.13}
$$

The quantity K determines the maximum value of the Mach number at a cell interface for which the artificial viscosity will be applied. The quantity B determines the magnitude of the viscous pressure term. It should be large enough to ensure stability but small enough to avoid obscuring important details of the solution. Generally, B need not exceed a value of 0.5.

Step 2 Transport effects are then calculated. The mass, which flows from cell to cell, is directly proportional to the density of the donor cell (the cell from which the fluid is flowing) and the velocity. Transport of momentum components and energy are then accomplished, assuming that the mass which crosses the cell boundaries carries the velocity components and the specific internal energy of the donor cell.

Boundary conditions

Three types of mesh boundaries have been used in the FLIC method: input boundaries, continuative output boundaries, and reflective boundaries. In all three cases the boundary criteria reduce to a determination of the values of the flow variables in fictitious cells outside the calculating mesh.

At any given time, input boundary cells are constant along the boundary, but they may vary with time to represent, for example, a decaying shock wave. At a continuative output boundary, the flow quantities in the fictitious boundary cells are defined in such a way that the normal space derivatives of the variables vanish at the boundary. If the output boundary coincides with cell boundaries, the flow variables in each fictitious exterior cell are set to equal the value in the adjacent interior cell.

The difference equations for those cells adjacent to the bounding surface of a rigid body must be modified to ensure that there is no flow of mass or energy across the boundary. This requires the normal velocity component to be zero at the body surface and is also achieved using fictitious cells for those cases where the reflective surface coincides with a cell boundary. A more complicated approach is used for a general body shape.

Stability and accuracy

The dominant error terms that are introduced by the numerical approximations can be obtained by performing a Taylor series expansion of the finite difference equations, neglecting all terms that are of order δt or δx^2. The form of these error terms is shown below for the case where the solution is a function of one space variable and time [12]:

	Artificial viscosity terms	Truncation error terms
$\frac{D\rho}{Dt} + \rho\frac{\partial u}{\partial x}$	$= \dots$	$+\frac{\partial}{\partial x}(\epsilon\frac{\partial \rho}{\partial x})$
$\rho\frac{Du}{Dt} + \frac{\rho P}{\partial x}$	$= -\frac{\partial q}{\partial x}$	$+\frac{\partial}{\partial x}(\rho\epsilon\frac{\partial u}{\partial x}) + \epsilon\frac{\partial \rho}{\partial x}\frac{\partial u}{\partial x}$
$\rho\frac{DI}{Dt} + P\frac{\partial u}{\partial x}$	$= -q\frac{\partial u}{\partial x}$	$+\frac{\partial}{\partial x}(\rho\epsilon\frac{\partial I}{\partial x}) + \rho\epsilon(\frac{\partial u}{\partial x})^2 + \epsilon\frac{\partial \rho}{\partial x}\frac{\partial I}{\partial x}$

(2.14)

where,

$$\epsilon = \frac{1}{2}|u|\delta x \tag{2.15}$$

There are two types of error terms: artificial viscosity terms that contain q and $\delta q/\delta x$; and truncation terms, which contain ϵ and $\partial/\partial x$ [12]. The effect of the error terms is to cause an artificial diffusion of mass, momentum and energy to occur in regions where there are large variations.

Chapter 3

The Free Piston Shock Tunnel

3.1 Shock tunnel development

The development of the free piston shock tunnel has been outlined by Stalker [32]. He stated that some new generation aircraft and spacecraft were being developed that travelled at 'hypervelocities', which generally means speeds in excess of 3 $km\,s^{-1}$. Aircraft and spacecraft development requires the testing of models. This testing has traditionally been carried out in wind tunnels. Vehicles travelling at these speeds would generally be at very high altitudes, since aerodynamic heating would render flight at low altitudes impossible. Thus, to develop these new aircraft and spacecraft, it is necessary to use a facility that can simulate the fluid dynamic effects of air at high altitudes and at hypervelocities.

To simulate the flow pattern around an aircraft travelling at supersonic speeds it is necessary to reproduce the freestream Mach number. Where the behaviour of boundary layer flow is important, it is also necessary to reproduce the Reynolds number. Large power sources are required to satisfy these conditions at the test section of a conventional, recirculating wind tunnel. Hypersonic wind tunnels were developed in which air was released from a high pressure reservoir rather than being recirculated, and a heater stage was added making it possible to reproduce the required Mach numbers [32]. What was missing was the ability to reproduce real gas effects - the dissociation and ionisation of molecules - which would occur at these flight speeds and altitudes. To represent the real gas effects it was also necessary to match the energy of the test flow to that in flight. However, air at such energies can destroy wind tunnel components within a few seconds. One method of avoiding this destruction was to use an 'impulse' facility which reduces the test time to fractions of a second.

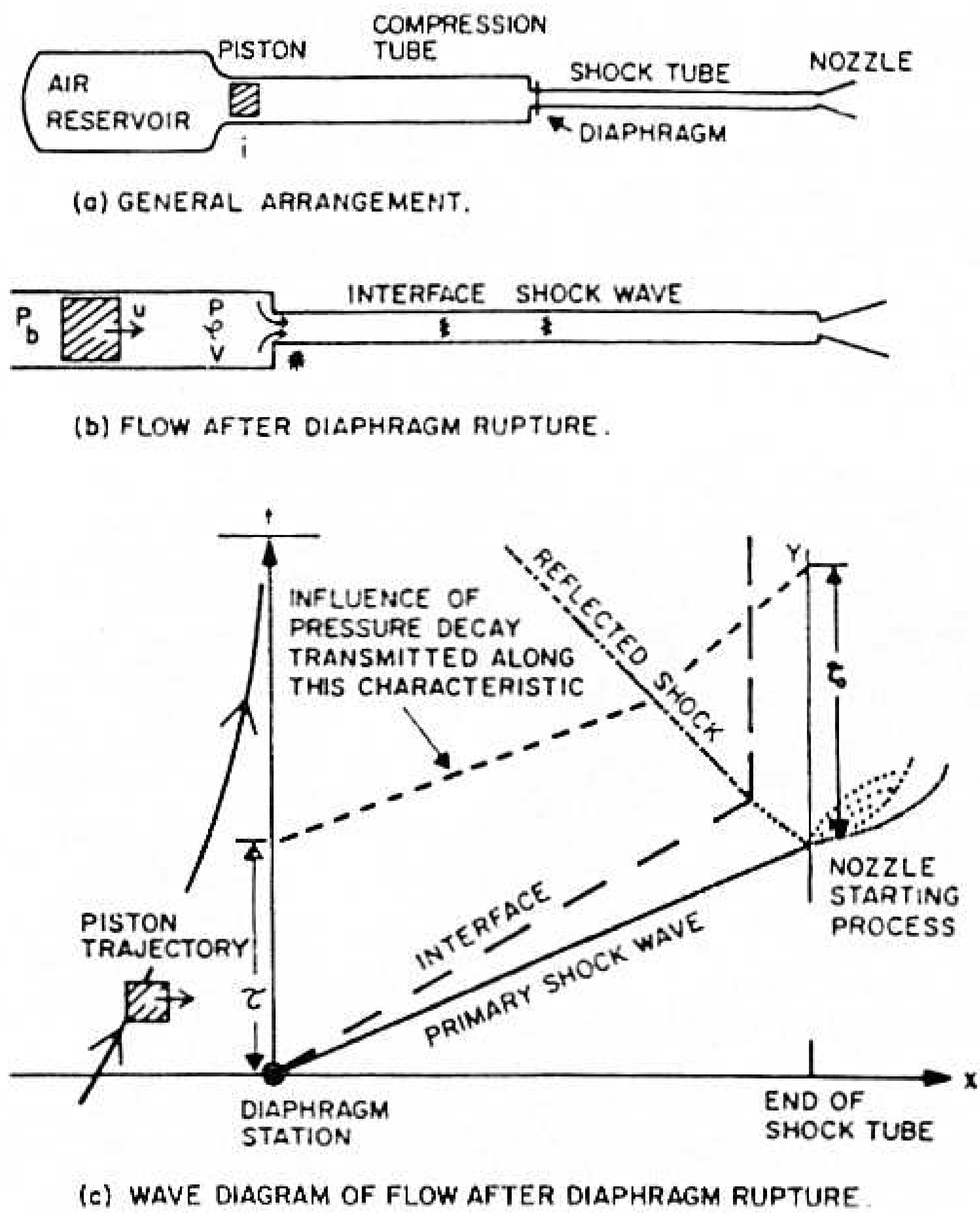
A shock tunnel is one such impulse facility and consists of a shock tube driving a hypersonic wind tunnel nozzle. Operation is initiated by rupture of a high pressure diaphragm that separates the driver gas from the test gas. The driver gas expands violently sending a shock wave through the test gas. When this primary shock reflects at the end wall, the test gas is brought to rest creating a slug at high pressure and high temperature, which then expands through the nozzle creating the required conditions in the test section. As it moves upstream, the reflected shock interacts with the driver gas behind the interface, and, in general, causes a disturbance to travel downstream into the test gas. When this effect occurs, it limits the time for which gas conditions remain constant in the test gas to a period that is too short to allow useful experiments in the test section. Fortunately, for a given primary shock speed, it is possible to choose driver gas conditions such that the reflected shock that brings the test gas to rest is of just the right strength to also bring the gas behind the interface to rest. This is known as 'tailored interface' operation. To achieve this condition for high shock speeds it is necessary to have a high speed of sound in the driver gas. The required speed of sound of the driver gas increases with the speed of the primary shock.

3.2 The free piston shock tunnel

High temperatures can be achieved for a short time in the driver gas by heating the driver gas with a rapid isentropic compression. This is all that is required to achieve even higher speeds in a shock tunnel. Such compression has been achieved by using a simple free piston apparatus [32].

In the free piston shock tunnel (Figure 3.1) the piston is initially stationed at the upstream end of the tube and the compression tube is filled with driver gas at a relatively low pressure. A reservoir is filled with high pressure air and then the piston is released. The piston accelerates down the compression tube until the

Figure 3.1: Operation of a free piston shock tunnel (reproduced from Stalker [32])



pressure of the compressed driver gas exceeds that of the compressed air behind it. Approximately constant thermodynamic conditions are maintained by allowing the diaphragm to rupture before the piston completes its compression stroke, and arranging that the subsequent movement of the piston face approximately compensates for the loss of the driver gas into the shock tube. Essentially, the shock tube is operated with a constant pressure driver rather than the more conventional constant volume driver.

Operation of the shock tunnel after diaphragm rupture is illustrated in Figure 3.1(b) and Figure 3.1(c). It can be seen that the nozzle starting process commences after shock reflection at the downstream end of the shock tube. Since the shock tube is operated in the tailored interface mode, a period of steady nozzle flow follows the nozzle starting process. The testing period is terminated by the arrival of an expansion wave, generated by falling pressure in the driver section. The time of arrival of this wave is dictated by the need to bring the moving piston to rest before it strikes the end of the compression tube. It has been found that this can be achieved, while still delaying any large variations in driver gas pressure until approximately half of the volume of driver gas available at diaphragm rupture has passed into the shock tube.

Table 3.1 shows the dimensions and some other specifications of the T4 free piston shock tunnel at the University of Queensland.

Table 3.1: Specifications of T4 (reproduced from Skinner [31])

Cavity	Length (m)	Diameter (mm)	Volume (litres)	Filled on	Error	Initial Pressure
Air Reservoir	11.3	173	266	bourdon gauge	±5%	-
Manifold	1	168	22.1	-	-	-
Compression Tube	25.53	229	1052	diaphragm strain gauge	±5%	< 1 torr
Shock Tube	10	76	45	bourdon gauge	±7%	< 1 torr
Nozzle	0.87	25 → 262	15	-	-	< 1 torr
Test Section	1.2	450 mm^2	243	-	-	< 1 torr
Dump Tank (x 2)	2.5	650	1104	-	-	< 1 torr
Piston: 90 kg mass, 0.47 m long, nylatron bearings Primary Diaphragm: 1 → 6 mm mild steel, burst pressure ~ 14.2 MPa per mm Secondary Diaphragm: mylar located at nozzle throat						

3.3 Test time limitation

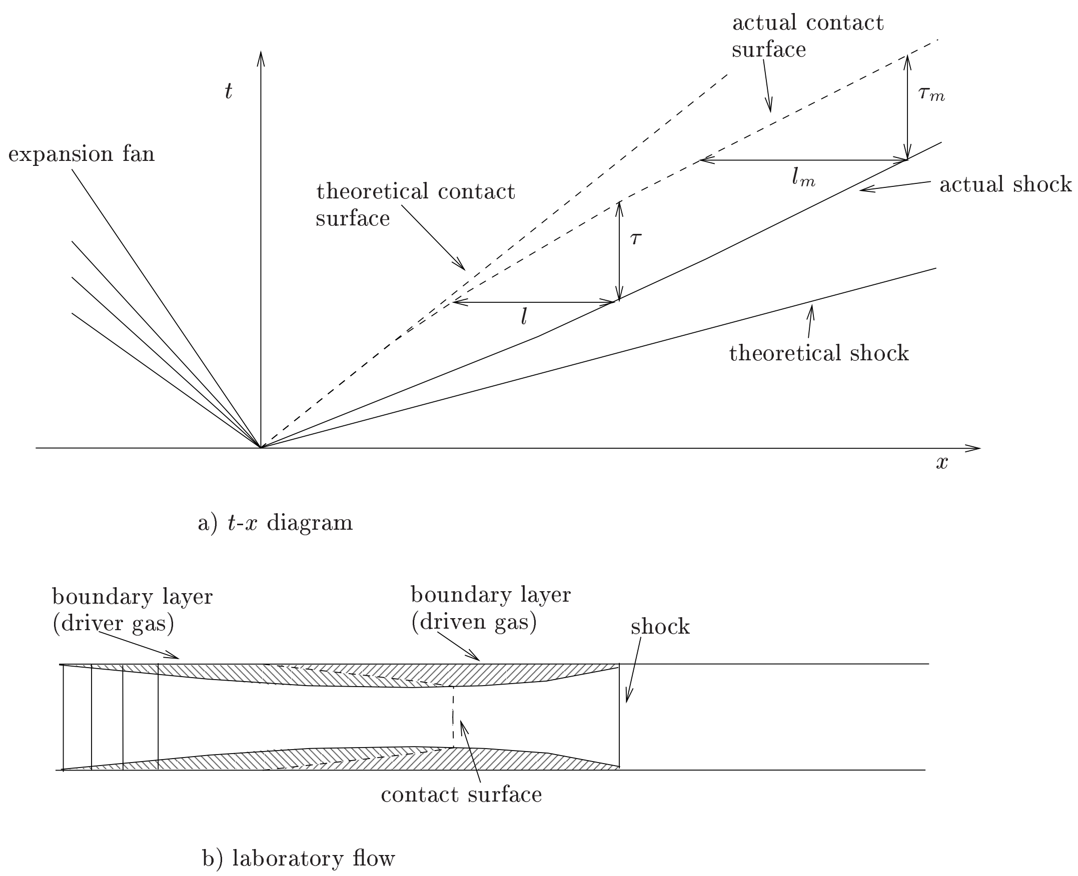
Researchers using a free piston shock tunnel facility need to know what proportion of all their measurements occur under the conditions they are trying to simulate.

The time interval over which these conditions occur is called the 'test time'. Measurements of steady pressure in the test section have traditionally been used to estimate the effective test time, however Crane and Stalker [7] showed that the arrival of contaminating driver gas occurs earlier than the loss of steady pressure.

The loss of test gas to the boundary layer in a shock tube facility was investigated by Mirels [28]. He showed that the wall boundary layer between the shock and the contact surface acts as an aerodynamic sink and absorbs mass from the post-shock region. This causes the contact surface to accelerate, the shock to decelerate and the separation distance, l (Figure 3.2), to drop below an ideal value. The separation distance approaches a limiting maximum value, l_m.

In Figure 3.2, τ is the effective test time and τ_m is the maximum test time, as determined by Mirels.

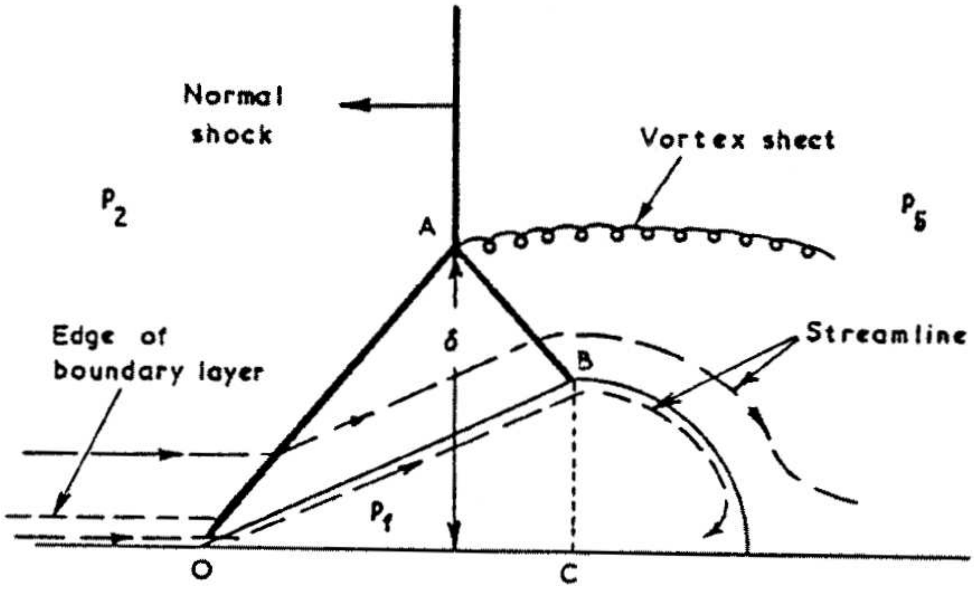
Figure 3.2: Boundary layer effect on shock tube flow (reproduced from Mirels [28])



The interaction of the reflected shock with the boundary layer was investigated by Davies and Wilson [9]. They discussed a mechanism first presented by Mark

[19], where under high enough free-stream Mach numbers the reflected shock can bifurcate (Figure 3.3) as it interacts with the boundary layer. Davies and Wilson proposed that this would cause the driver gas to 'jet' through the bifurcated shock system as it interacts with the contact surface. They concluded that the early arrival of driver gas is at least partially due to this mechanism.

Figure 3.3: Schematic diagram of bifurcated shock system (reproduced from Davies and Wilson [9])



Skinner [31] examined the possible causes of mixing of the test gas and the driver gas during the operation cycle of the shock tunnel. He investigated a number of possible mechanisms for driver gas contamination, namely:

1. Driver gas jet through the opening diaphragm;
2. Turbulence from the driver flow through shock tube entrance;
3. Acceleration and deceleration of the contact surface;
4. Loss of test gas to boundary layer;
5. Reflected shock vortex formation;
6. Reflected shock-boundary layer interaction; and
7. Shock deceleration of the contact surface.

Skinner used a purpose built mass spectrometer mounted in the test section of the T4 shock tunnel to measure the time taken for significant concentrations of driver gas to arrive there. He conducted a series to investigate each of the above possible effects. In these experiments he varied the primary shock Mach number, the density of the driver gas and a number of other mechanical parameters of the shock tunnel facility. He conclude that the effects of reflected shock processes and the jetting through the diaphragm were not probable causes of test time limitation. He also concluded that the loss of test gas to the boundary layer seemed to produce reasonable estimates of the arrival of the centre of the mixing region but cannot be applied to the arrival of the leading edge of a spatially smooth contact region. Skinner discounted the mechanism proposed by Davies and Wilson because it should have shown a sudden increase in contaminants, whereas his results showed that the contamination levels rose smoothly. Skinner concluded that the most likely candidate for the loss of test time is the evolution of a turbulent mixing region around the contact surface. He stated that the source of the initial turbulence is most likely the flow of the driver gas through the opening diaphragm.

3.4 A particular driver gas contamination problem

A design goal of the computer program developed in the current work was to produce a code that could be used to investigate driver gas contamination. Since Skinner's [31] experimental data was available, one of his shock tunnel runs was used to provide conditions for a simulation. Table 3.2 details the conditions for the run chosen.

Table 3.2: Run# 37

Driver gas composition	15% Ar, 85% He
Shock tube fill pressure	15kPa
Test gas	Air
Mach number of primary shock M_s	10.4
Length of shock tube	10m
Diameter of shock tube	76mm

A standard temperature of 300K in the shock tube was assumed. Shock tube relations from Liepmann and Roshko [17] were used to calculate the appropriate inlet conditions required to generate such a shock. These conditions were then normalised to set the post shock flow variables to unity (Appendix C).

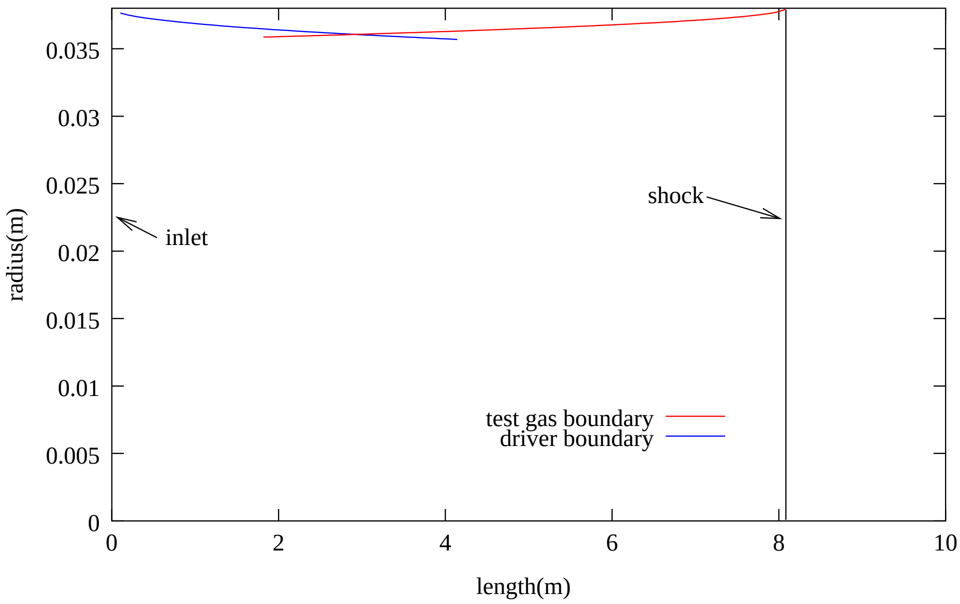
3.4.1 *Expected boundary layer thickness*

Since the computer program developed does not model turbulence, a flat plate laminar boundary layer profile is expected behind the shock and also leading downstream from the inlet. A plot of this expected result is shown in Figure 3.4 and was produced using Equation 3.1 [15] for the computation conditions shown in Table 3.2.

$$\frac{\delta}{x} = \frac{4.64}{\sqrt{Re_x}} \tag{3.1}$$

Here, δ is the boundary layer thickness and Re_x is the Reynolds number based on length x.

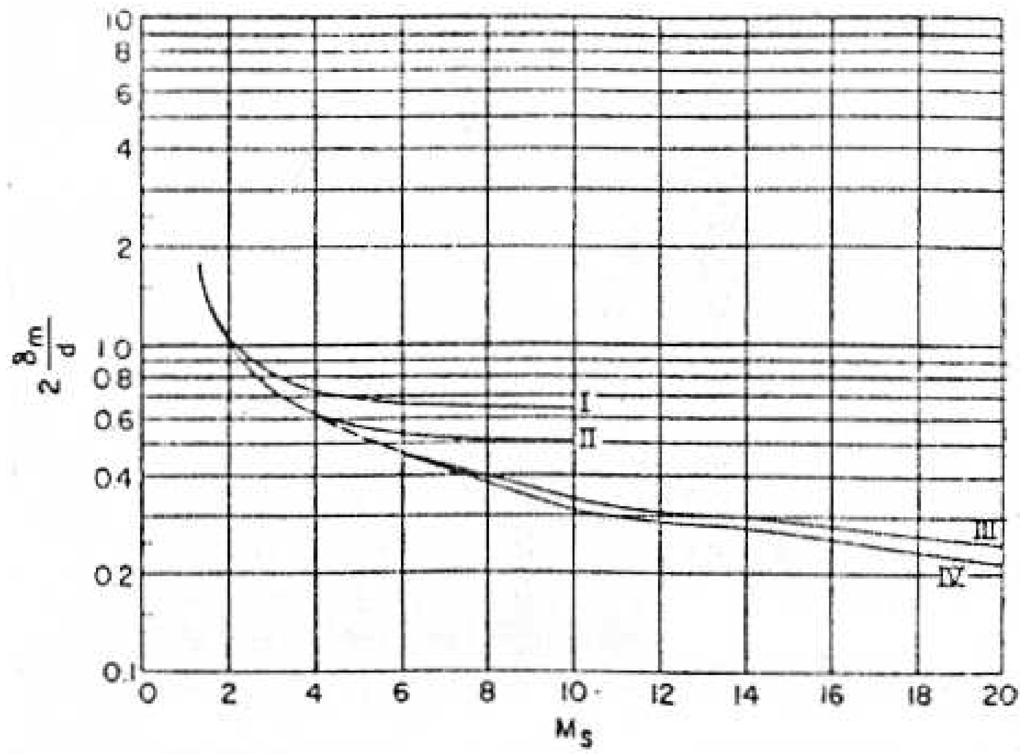
Figure 3.4: Expected boundary layer profile assuming laminar flat plate flow



In his determination of test time limitation, Mirels [28] developed a way to determine the expected boundary layer thickness at the contact surface, based on turbulent boundary layers. Figure 3.2(b) is a qualitative view, according to Mirels, of the expected boundary layer development in a shock tube. This diagram shows the boundary layer forming behind the primary shock in the test gas and the other boundary layer forming in the driver gas according to normal pipe flow processes. Figure 3.5 can be used to predict the boundary layer thickness at the contact surface, δ_m, for a given primary shock Mach number. For a primary shock Mach number of

10 in ideal air (Case II), $\delta_m = \frac{d}{4}$, where d is the tube diameter. Though the contact surface location is not drawn, Figure 3.4 shows that there is a discrepancy between Mirels' value and the expected boundary layer profile. This is due to the fact that Mirels' calculations were for a turbulent boundary layer - which would be the case in a real shock tube at these Reynolds numbers.

Figure 3.5: Boundary layer thickness at l_m (reproduced from Mirels [28])



Chapter 4

Milthorpe's Algorithm

4.1 Overview

Milthorpe [23] noted that many existing methods at the time for computing supersonic flows were based on continuum mechanics. These methods predicted a flow by solving a set of partial differential equations subject to certain boundary conditions. When these methods were applied to hyperbolic problems, as those encountered in hypervelocity research, instabilities often occurred near shocks or other discontinuities.

Milthorpe set out to produce a method that would capture flow discontinuities accurately. His method does not attempt to solve the partial differential equations, but rather examines the behavior of the fundamental quantities: mass, momentum and energy. He accomplished direct numerical simulation where the physical laws are applied to the fluid variables on small volume elements or cells.

The approach is similar to a finite difference method derived from the same physical laws, except that conditions have been placed on many of the calculations. For example, diffusion is limited to a certain proportion of a fluid quantity in a cell. Another example is that pressure difference effects are applied with an upwinding coefficient, which depends on some locally averaged quantities. It could be said that the progress of the flow calculation is managed so that the calculation proceeds in a realistic fashion. This is in contrast to a brute force finite difference, or other equation solving method where complete trust is placed on the governing equations and only the solution of these equations is deemed important.

The algorithm used to calculate the flow values is based on conservation of mass, momentum and energy as the simulation proceeds over discrete time steps. The flow domain is divided into rectangular cells. Each cell contains, at any given time, a certain block of material that possesses a certain amount of momentum and energy. In the course of a time step this block is modified by pressure gradient effects and diffusion. This block, with its associated mass, momentum, and energy, is then convected a small distance depending on the local velocity vector. Greatest in magnitude, at the velocities modelled, is convective transport of mass, momentum and energy due to the velocity of the fluid. Following this are the effects of pressure gradients with the effect of diffusive transport being least significant in magnitude.

4.2 Conservation algorithm

Calculation of fluid transport is conducted in three phases at each iteration.

4.2.1 Phase 1 - Diffusion

The fluid quantities of mass, momentum and energy are modified by the process of diffusion according to Equation 4.1:

$$dQ^k_{i,j} = \delta t \alpha^k [dy(Q^k_{i-1,j} - 2Q^k_{i,j} + Q^k_{i+1,j}) + dx(Q^k_{i,j-1} - 2Q^k_{i,j} + Q^k_{i,j+1})] \tag{4.1}$$

where, $Q^k_{i,j}$ is the quantity of mass, momentum or energy in (i,j)th cell and α^k is one of the diffusion coefficients shown in Table 4.1. The cell sides have dimensions dx and dy and the time step is δt.

Table 4.1: Diffusion coefficients

viscosity	μ
molecular diffusivity	D
thermal conductivity	k

Sutherland's law (Equation 4.2) [16] is used to calculate the temperature dependence of the viscosity.

$$\mu = \mu_0 \left(\frac{T}{T_0}\right)^{3/2} \frac{T_0 + S}{T + S} \tag{4.2}$$

where, μ_0, T_0 and S are coefficients that depend on the species of gas in question and T is the local temperature of that gas species.

The molecular diffusivity, D, is calculated using:

$$D = \frac{\mu}{\rho S_c}$$

where, S_c is the Schmidt number.

The thermal conductivity, k, is calculated using:

$$k = \frac{\mu c_p}{P_r}$$

where, P_r is the Prandtl number.

In addition, momentum and energy are modified by diffusion of those quantities along velocity gradients and energy is modified by diffusion along temperature gradients. These are calculated using an expression similar to Equation 4.1.

In all cases, the amount of a fluid quantity that is transported by diffusion is limited to half the amount of that quantity in the donor cell.

4.2.2 Phase 2 - Pressure gradient effects

The momentum and energy in the cell are modified by the action of the pressure gradients. The pressure difference between adjacent cells affects the momentum in the cells according to the following relations:

$$\delta(\rho u_{i,j}) = \delta t dy(p_{i-1,j} - 2p_{i,j} + p_{i+1,j}) \tag{4.3}$$

$$\delta(\rho v_{i,j}) = \delta t dx(p_{i,j-1} - 2p_{i,j} + p_{i,j+1}) \tag{4.4}$$

where, $\delta(\rho u_{i,j})$ and $\delta(\rho v_{i,j})$ are the changes in x- and y-momentum at the (i,j)th cell.

At the same time as the pressure gradients are transporting momentum between adjacent cells, if the local velocity is non-zero they are also doing work, and hence transporting energy between the cells. This energy transport is obtained from the momentum transport calculated in Equations 4.3 and 4.4 by multiplying by the local velocity.

Upwinding factor

Milthorpe found that by including some upwinding terms in the calculation of pressure gradient effects the accuracy of the solution was enhanced. Adding an upwinding factor to Equation 4.3 gives:

$$\delta(\rho u) = \frac{\delta t dy}{2}\left[(1+F)p_{i-1,j} - 2Fp_{i,j} - (1-F)p_{i+1,j}\right] \tag{4.5}$$

where, F is the upwinding factor, such that $F = 1$ gives complete upwinding with a positive u-velocity. The y-momentum was calculated in the same way. A series of tuning runs calculated the optimum upwinding value for ranges of local Mach numbers. For each direction (x,y) the sign of the upwinding value that caused the least change in momentum was chosen.

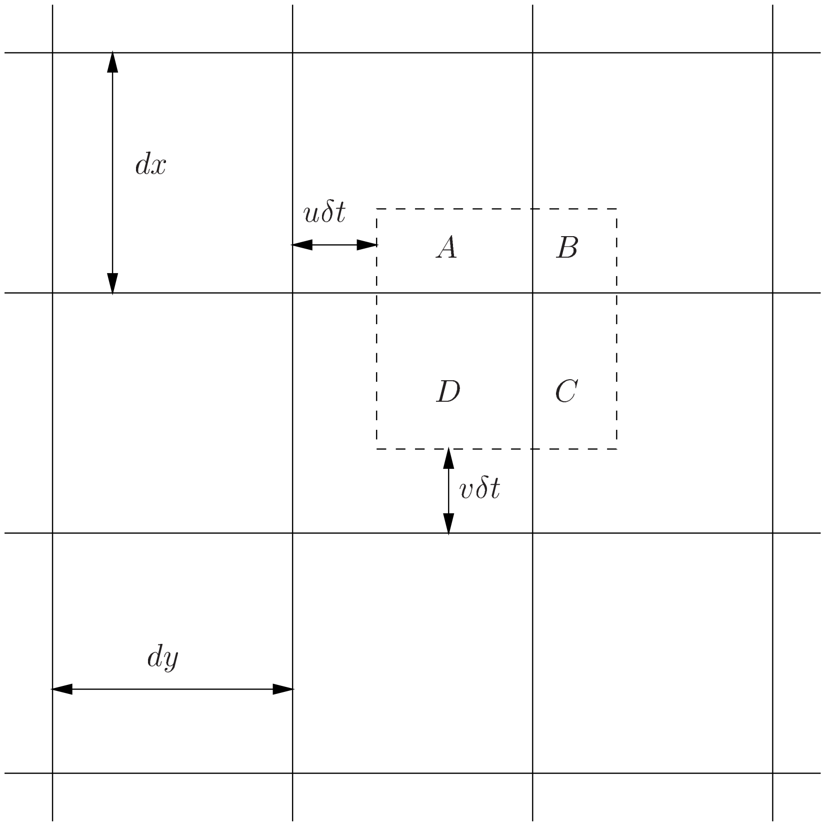
The expression used to calculate the energy transport due to pressure gradients also included a second upwinding parameter, e_f, that is used to combine different amounts of upstream and downstream velocity in the calculation. The rather complicated equation for this is:

$$\begin{aligned}
\delta E \;=\; & \frac{\delta t dy}{2}\Big\{-\left[(1+F)p_{i,j} + (1-F)P_{i+1,j}\right] \\
& \times\left[\frac{(1-e_f)}{2}\left[(1+F)u_{i,j} + (1-F)u_{i-1,j}\right] - e_f u_{i+1,j}\right] \\
& + \left[(1+F)p_{i-1,j} + (1-F)p_{i,j}\right] \\
& \times\left[\frac{(1-e_f)}{2}\left[(1+F)u_{i-1,j} + (1+F)u_{i,j}\right] + e_f u_{i-1,j}\right]\Big\}
\end{aligned}$$

4.2.3 Phase 3 - Convection

Convection of the cells is illustrated in Figure 4.1 for convection with positive u- and v-velocities. A cell is regarded as moving with unchanged dimensions relative to the mesh of cell boundaries, so that it overlaps adjacent cells. The letter A indicates the area of the cell that remains in the original cell at the end of the time step $u\delta t$, while B, C and D represent the areas of a cell that are located in adjacent cells. The quantities of mass, momentum and energy in the cell are divided in proportion to the areas A, B, C and D. Those proportions represented by B, C and D are subtracted from the quantities in the original cell and added to the appropriate adjacent cells.

Figure 4.1: Illustration of cell convection



4.2.4 Calculation of dependent flow variables

When the adjustments to mass, momentum and energy have been completed for all cells, the dependent flow variables density (ρ), velocity (u and v), pressure (p) and temperature (T) can then be calculated. These values are then used in the next iteration. The variables ρ, u and v are calculated in the usual way. However, T is calculated from the cell's energy and velocity using:

$$T = \frac{E - \frac{1}{2}\left(u^2 + v^2\right)}{mc_p}$$

The ideal gas equation of state, Equation 2.5, is used to calculate p.

4.2.5 Stability limits

It is computationally more economical to use the largest possible time step in the computations, but there are a number of limits imposed by accuracy and stability considerations. The first limit is that since convection occurs only into the adjacent cells, the distance travelled in a time step must not exceed the cell dimension. The second limit is that due to over prediction of diffusion, as described by Roache [30]. This restricts the time step to:

$$\Delta t < \frac{\rho \Delta x^2}{2\alpha_k}, \tag{4.6}$$

where, Δx is the step size, and ρ and α_k are as defined earlier.

A third limit arises from pressure gradients causing a sufficiently large change in momentum to alter the sign of the momentum in a cell.

At each iteration, these three limits, and several others, are checked to see if the time step is too large. If this is so, the time step is reduced by a certain amount, otherwise it is increased by a smaller amount. This process stabilises the time step to a particular value while flow conditions are effectively steady and adjusts to changes, such as when the flow reaches an object or a wall. This self adjusting time step allows the calculation to progress as fast as possible whilst keeping the system stable.

4.3 Boundary conditions

Milthorpe implemented boundary conditions by testing for certain array indices as the algorithm looped through the arrays of fluid values.

4.3.1 Upstream boundary

The simplest boundary implemented was the 'upstream' boundary. Most of the problems that Milthorpe's method has been applied to involve a transient solution of a flow where the fluid enters from one side of the computational domain. The side where the fluid enters is called the upstream boundary. Upstream boundary cells are reset to their initial values at the start of each iteration. They take part in the process of diffusion and convection into their neighbouring cells, though their own fluid values are not updated.

4.3.2 Symmetric boundary

A feature that often appears in the kind of problems Milthorpe's algorithm has been applied to is a symmetry surface or slip plane. There is no flow across this surface, no shear stress and no diffusion of energy or material. These conditions occur at a plane of symmetry, on the centre line of an axisymmetric flow and at a solid surface at which it is not required to enforce a no-slip condition. To implement this, values of all variables in a 'dummy' layer immediately outside the computational domain are set equal to those inside, ensuring no diffusion across the boundary. The only restriction on the values in the surface layer is that the normal velocity component cannot be directed into the boundary. The values in any cell are, of course, only a representative average of the point values, so non-zero velocities into the computation domain do not violate the condition of zero flow outward across the boundary. The normal velocity in the dummy layer is set to zero and hence there is no flow inwards across the boundary.

4.3.3 Freestream boundary

This boundary condition is by far the most difficult to implement successfully. In contrast to the predetermined downstream condition, such as a set value of pressure in the freestream, no information is known about the conditions at a free downstream boundary, except that no influence from the boundary can be allowed to propagate into the flow upstream. Many experimental results are concerned with external flows and these require free downstream boundaries if the size of the computation domain is to be at all reasonable.

Roache [30, p. 165] discussed what he called the ‘downstream paradox’. Normally in supersonic flow, downstream disturbances cannot effect the flow upstream. The paradox in CFD is that the outflow condition, which is downstream, is more important in supersonic flow than in subsonic flow. Roache described an example scenario demonstrating how this can be the case. He outlined [30, p. 279] various approaches used to deal with a downstream boundary in compressible flow - most relying on some kind of extrapolation. Roache described an ‘upper’ boundary separately and outlined a method that follows simple wave characteristics to determine a direction of extrapolation.

Milthorpe implemented a freestream boundary by projecting the contours of the flow variables across the boundary. The direction of the contours at the boundary was determined for each cell by calculating the direction that gave the minimum average deviation from the contour value. The value outside the boundary was taken as the mean value along that direction for a small distance into the region. This use of extrapolation in the direction of a contour sets Milthorpe’s method apart from those described by Roache. Milthorpe’s freestream boundary implementation produced quite a straight projection of the contours over the boundary [24].

4.4 Comparison with FLIC

Milthorpe’s method is similar to the Fluid-in-Cell (FLIC) [12] method described in Section 2.2.6, both being derived from a control volume approach (Section 2.2.2). They are both intended to be used for compressible flow problems and the implementation of convection is very similar in the two methods. The main differences, however, are:

- FLIC uses an artificial viscosity term to maintain stability in the calculation whereas, Milthorpe’s method explicitly models viscosity and other diffusion phenomena. Milthorpe’s method has an equivalent to the parameter B in that diffusion is limited to no more than half of the fluid quantity in the donor cell.

Stability is maintained in Milthorpe's method by dynamically adjusting the time step according to various criteria.

- FLIC does not implement any kind of upwinding. Milthorpe has two upwinding parameters, F and e_f, that are calculated according to local conditions.
- FLIC stores density, velocity and specific internal energy between iterations whereas Milthorpe's method stores total mass, energy and momentum.

4.4.1 Pressure gradient effects

The difference equation used in the FLIC method to calculate pressure gradient effects in velocity is:

$$\tilde{u}^n_{i,j} = u^n_{i,j} - \frac{\delta t}{\rho^n_{i,j}\delta x}(P_{i+\frac{1}{2},j} - P_{i-\frac{1}{2},j}) \tag{4.7}$$

where, $P = p + q$ and q is given in Equation 2.13.

An analysis of Equations 4.3 and 4.7, shows that they differ by a factor of $\delta x \delta y$, which is the cell volume. The FLIC version has an artificial pressure term and Milthorpe's has an upwinding factor, F. The discretisation scheme is also a little different. The Milthorpe method calculates values at cell-centred node points, whereas FLIC calculates values at cell boundaries.

4.4.2 Transport effects

The FLIC method calculates a mass flux at each cell boundary proportional to the mass of the donor cell and the intermediate value of the velocity calculated ($\tilde{u}^n_{i,j}$). Transport of velocity and internal energy is calculated based on the quantity of mass transported.

This differs from Milthorpe's convection algorithm in that Milthorpe transports a proportion of mass, momentum and energy independently of each other - depending only on the factor $u\Delta t$. Here u is equivalent to FLIC's u^n not the intermediate value $\tilde{u}^n$.

Chapter 5

Implementation in C++

5.1 Overview

Using object-oriented design principles Milthorpe's algorithm has been implemented using the computer programming language C++[33]. The name 'CONVECT' was given to the computer program since this seemed to be an important feature of Milthorpe's algorithm and this is how it will be referred to. Objects have been used to specify the properties and behavior of gas species, computational cells and of regions within the computational domain. Features of C++, such as data-hiding and inheritance, should reduce the time needed for further development of the code. This is because the code modules that have already been tested need not be changed, minimising the introduction of new bugs. Object-oriented programming is a paradigm where data structures and the computer code that operates on those data structures are kept together. Being a modular approach, object-oriented programming allows a programmer, or team of programmers, to deal with one part of a computer program at time.

One of the objectives of the new implementation was to investigate whether this approach to computer programming would aid in the process of computational research - in this case with Milthorpe's algorithm. Milthorpe's computer program had become complex and somewhat difficult for an outside researcher to grapple and add to. An object-oriented version of this computer program would allow other researchers to identify and modify only those parts of the code that are relevant to the problem they are trying to solve; potentially saving them a great deal of time.

If this approach did indeed make the further development of Milthorpe's algorithm simpler and less prone to bugs, time could then be concentrated on dealing with the more important fluid dynamics and numerical aspects of the work.

5.2 Object-oriented programming

5.2.1 Procedural programming

Procedural programming languages have been widely used for decades. They grew out of a perceived need for computer programs to be more structured than earlier programming languages allowed. The process of writing and re-writing code using procedural languages would be more efficient. In such a language, procedures are used to introduce order to a computer program. A computer program is divided into a set of logical tasks and procedures are written to carry out these tasks. Procedures can have parameters, or arguments, passed to them. They can be grouped together in libraries that can then be used as part of many different computer programs separated in time and space. In this way, the libraries represent a kind of tool box that can be used by computer programmers who do not need to know how a tool functions, just how to use it. This saves a computer programmer time, since time is not wasted 're-inventing the wheel'. In the computer language Fortran, these procedures are called subroutines - in other languages they may be called procedures or functions.

5.2.2 Modular programming

A set of related procedures, together with the data they manipulate, can be grouped together into a 'module'. Modular programming is also referred to as 'data hiding'. A computer programmer using a pre-written module does not necessarily need to know the structure of the data that the module uses internally. The programmer just needs to deal with the module's 'interface' - a set of procedures and the parameters they require. A computer program written to use a particular module need not be modified if the internal data representation of the module is later refined. Another advantage of modular programming is that computer programmers do not have to keep track of data that may be global (accessible by all the procedures within a module) and there is no danger that they would accidentally introduce bugs by modifying this global data unnecessarily.

5.2.3 Data abstraction

A programming language that supports modular programming could also support the idea of 'user defined data types', known as 'data abstraction'. For example, a language may not know how to deal with complex numbers so a computer programmer could define a new data type for this purpose. This new data type would define how a complex number is stored and what needs to be done when you indicate that you want to add two of them together. Data abstraction has some powerful applications in areas such as linear algebra where complex matrix operations can be expressed in a concise form, analogous to expressions used by mathematicians.

A problem with data abstraction is that an abstract data type defines a sort of black box and that there is no way of adapting it to new uses except by modifying its definition. This can lead to severe inflexibility.

5.2.4 Objects

Objects are program modules that provide data abstraction and also allow 'inheritance'. A computer programmer can create a new kind of object by inheriting the properties and behaviour of a parent object and then customising only those properties and behaviour that make the new object different from the parent. This process can be used to create a whole hierarchy of objects that handle a range of different aspects of a computer application. In fact, serious converts to the object-oriented programming paradigm delegate all aspects of their computer application to objects that belong to a single hierarchy.

A good example of a set of objects is the idea of a 'shape' in a computer graphics system. Each shape could have a colour and a point defining its location on the screen. This can be represented by an object with actions associated with it such as move, draw and rotate. To implement a circle a computer programmer would inherit a shape and redefine what draw and rotate mean for it. A circle would need to have an extra property defined for it called a radius. Similarly, a square can be set up by redefining only those parts of a shape that are needed to specify a square.

When defining objects in an object-oriented computer language the term 'class' is used. This creates a distinction between the definition of a class of objects and actual 'instances' of that class.

Inheritance allows an object that has already been developed and debugged to be re-used and enhanced. This feature lends itself to a kind of incremental, or 'building block', approach to software development. Object-oriented programming allows different programmers to use the same code and add to it without having to spend too much time analysing the existing code and without the danger of introducing new bugs to the existing code (which is also the case with other styles of modular programming).

In designing an object-oriented computer program, a programmer divides the problem into tasks, each to be handled by a set of objects. A list of behaviours, or 'methods', are defined for these objects. During this process, commonality between types of objects are identified so that a hierarchy of objects can be set up using inheritance.

5.3 The design of CONVECT

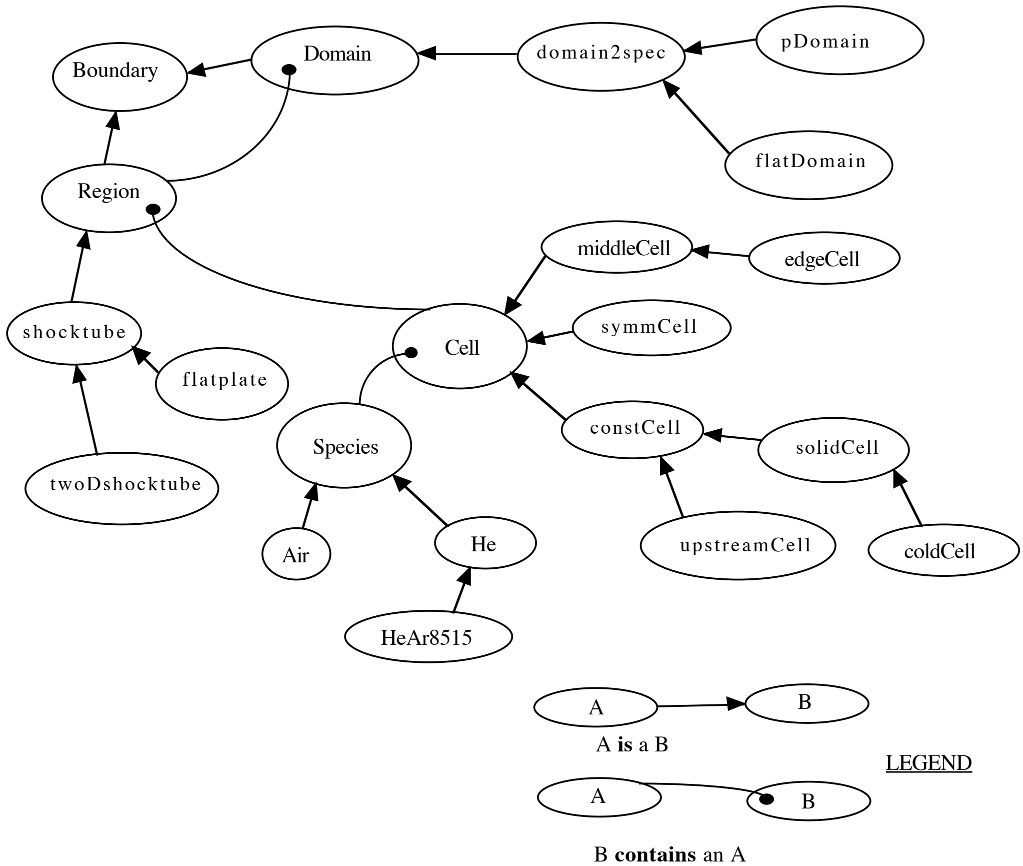
Although the fundamental nature of Milthorpe's algorithm has been preserved, the code has been completely redesigned and rewritten in the object-oriented computer language C++. As described above, the design process involved analysing Milthorpe's algorithm and delegating parts of its functionality to a set of objects. A hierarchy of these objects was set up so that where objects had common features, inheritance could be used.

The original Fortran program stored all fluid quantities in separate arrays. There was also a proliferation of parameters, loop counters and other variables. Using the data abstraction inherent in an object-oriented programming approach, keeping track of variables and arrays when dealing with the program code is vastly simplified.

It was decided to use physical concepts to guide the choice of objects used in CONVECT. Each Species of gas is represented by an object class. The various physical subdivisions of the problem are represented by different object classes. These include Domain, Region and Cell. Objects that represent tasks, not physical quantities have also been defined. Convector is one example and it carries out the process of convection. In most cases, it is the responsibility of a particular object to modify, input or output its own data. A representation of the main objects can be seen in Figure 5.1 and an explanation follows.

In general, Milthorpe's algorithm has been directly translated into this new program structure. Section 5.5 outlines the main differences between the current implementation and Milthorpe's.

Figure 5.1: A representation of the object hierarchy in CONVECT



5.4 The object classes

5.4.1 Domain

A Domain holds most of the properties of the entire computational domain. This includes the overall size and shape of the problem. Domain methods setup and ensure the execution of the whole problem. Many command line parameters to the computer program translate directly into calls to Domain methods.

5.4.2 Region

A Region was originally distinguished from a Domain to allow the code to be implemented in parallel. The idea was that a Domain would contain several Regions that would be executed in parallel. When the parallel version was implemented, however, the method used resulted in single Regions occupying a single Domain. There was in the end no need for these two objects to be separate but the distinction was kept to reduce unnecessary debugging.

Regions are given the responsibility of keeping track of the Cells they contain and looping through those Cells, instructing them to be processed or updated. Regions 'know' how to input and output their data to binary or text files. They keep track of Cell dimensions dx and dy as well as the adjustable timestep δt. It was envisaged that different Regions in a parallel problem may have different values of dx, dy or δt. This would allow the computer algorithm to create finer grids and smaller timesteps, as required, similar to an adaptive mesh. This has not yet been implemented.

Inheritance has been used to implement different kinds of problems with slightly different kinds of Regions and Domains. Two examples of specialised Regions are the flatplate and the shocktube.

5.4.3 Cell

A Cell holds the property of Momentum and a list of the gas Species in it (see Section 5.4.4). Cell methods carry out the actual fluid dynamics by either implementing the various physical processes themselves or calling other objects that do. There are also methods that aid a Region in determining whether to adjust the timestep by checking certain stability criteria. Boundary conditions are implemented by populating a Region with different kinds of Cells. There are upstream Cells, wall Cells and symmetric boundary Cells. Inheritance has been widely used in the implementation of the hierarchy of Cell types.

5.4.4 Species

A Species holds the properties of Mass and Energy. Different kinds of Species represent the different kinds of gas species that are present in a particular problem. Each kind of Species can have its own definitions for certain gas properties such as molecular weight, the Prandtl number and the Schmidt number.

5.4.5 Diffuser

A Diffuser is given the addresses of two Cells as an argument and carries out the process of diffusion between them, updating Cell quantities accordingly. Different kinds of Diffusers are implemented to handle different kinds of diffusion process in certain boundary Cells. For example, there is a solidDiffuser for adiabatic no slip diffusion and a symmDiffuser for diffusion at a symmetric boundary.

5.4.6 Convector

A Convector is given the address of a Cell and convects mass, momentum and energy out of that Cell into neighbouring Cells according to the velocity in the parent Cell. The calculation of the convection velocity factor, a parameter that determines the proportion of material being convected, was delegated back to the Cell object class since special boundary Cells needed to deal with this calculation in different ways. This is a good example of how the identity of an object can determine how a particular method is executed. On the other hand, a more traditional language needs sometimes to go through an elaborate series of conditional statements to determine what needs to be done.

5.4.7 *Squeezer*

A Squeezer is given the address of a Cell and implements pressure gradient effects on that Cell. The name Squeezer was assigned to this object because a single worded descriptive name was needed.

5.4.8 *Transport*

A Transport object is used by a Diffuser to encapsulate the local transport properties of viscosity, thermal conductivity and mass diffusivity. It is 'contained' within a Diffuser object. When the Diffuser is initialised with the addresses of those Cells it will operate on, the Transport object then calculates the effective local viscosity, thermal conductivity and mass diffusivity, based on the local temperature and species concentrations. These values are then called on by the Diffuser for its calculations.

5.5 Implementation differences between Milthorpe's method and CONVECT

5.5.1 *Dynamic memory allocation*

Memory for the Cell objects was allocated dynamically as this is standard C++ programming practice [33], and an array of pointers to these objects was kept. This meant that adjacent Cells in the computational domain were probably, but not necessarily, adjacent in memory. This can have performance effects, since memory caching schemes on modern computers read in chunks of adjacent memory, assuming that operations are often performed on contiguous data.

Beyond that, modern compilers have built-in optimisation features that use knowledge of particular computer architectures to get the best performance out of computer code. When data structures are abstracted, as they have been in CONVECT, it is much less likely that an optimising compiler will be able to achieve as much speed-up as with an ordinary array storage.

5.5.2 *De-referencing*

Much of the data in CONVECT is accessed using pointers or a series of pointers. This 'de-referencing' of memory can slow down the reading of the data into the processor. An example of this is a Region wanting to access the energy of the first Species in a particular Cell. This would be written something like:

```
elem(i,j)->spec(0).Energy()
```

Functions are de-referenced also and this too has a computational overhead. It is for this reason that 'inlined' functions were used where possible. Inlined functions are compiled into sections of computer code that are then inserted, as is, in place of a branch instruction. This means that the compiled program has to execute less instructions, making it much more efficient. The C++ compiler gcc was used and extensive use of its explicit and automatic inlining features drastically reduced the function de-referencing overhead.

5.5.3 Momentum and velocity as cell properties

Momentum in CONVECT is stored for each Cell as opposed to each gas species, as in Milthorpe's method. This was done so as to save a little on storage. Velocity is calculated as $Momentum/TotalMass$ and is also a Cell property. This has not had a significant impact on the accuracy of solutions.

5.5.4 Timestep adjustments

Timestep adjustments were essentially carried out in the same way as in Milthorpe's computer program. Log files of program output were examined to determine which adjustments were actually being used. Where it was found that a timestep adjustment was not used it was disabled. Generally, the CFL stability adjustment (Section 5.5.8) and an adjustment that looks at local oscillations seemed to dominate. Some parameters, including the minimum allowed timestep and the minimum density allowed in a diffusion calculation, were also adjusted to improve the stability of the particular problems being implemented.

5.5.5 Inputs file

To save having to re-compile the program for different problem conditions, most of the program parameters can be entered from a kind of free-form inputs file. The inputs file is a text file with a series of parameter names and values. For parameters that are not present in the inputs file, sensible defaults are used (see Appendix B).

5.5.6 *Calcinit.pl - calculation of initial conditions*

Since the code was being used to model a shock tube, it was necessary to determine initial conditions such that there would be a main shock heading downstream with no upstream shock. The inlet condition was set to be the condition immediately behind the contact surface. This required conditions to be 'matched' so that only the primary shock would propagate and there would be no shock or expansion wave travelling upstream. A Perl program was written to calculate initial conditions given the shock tube relations [17] and to produce a valid inputs file for CONVECT (see Appendix C).

5.5.7 *Cross platform features*

The code was, at different times, developed on Sun workstations or under Linux on a PC, or on a Macintosh. It was therefore necessary to make sure that CONVECT could be compiled and run on the different platforms without major changes. Another reason for implementing CONVECT on different platforms was to allow for parallel processing of the program on a heterogeneous workstation farm. It was necessary to modify the format of the binary data files produced so that they could be read on different platforms. Intel based Linux uses a different byte ordering than Sun so without the new data format data files could not be read on the other platform.

5.5.8 *Stability*

During testing of CONVECT, it was found that the solution seemed to be less stable when $dx \neq dy$. A two dimensional, compressible version of the CFL (Courant, Friedrichs and Lewey[6]) condition was briefly presented by Roache [30, p. 229]. This condition was used in place of a one dimensional version that Milthorpe used during the convection step of the calculation. The condition is:

$$C = \frac{(|\tilde{u}| + a)\,\delta t}{\Delta} \leq \frac{\sqrt{dxdy}}{\sqrt{dx^2 + dy^2}}$$

where Δ is replaced with dx or dy depending on the direction in which convection is currently being applied, a is the sound speed and $|\tilde{u}|$ is the magnitude of the velocity in the current cell.

5.5.9 Artificial viscosity

During the development and testing of CONVECT, it was found that the boundary layer thickness being produced was not nearly large enough. It was much smaller than that predicted by Mirels [28] (see Section 3.4.1). It was even much smaller than the theoretical laminary boundary layer shown in Figure 3.4. Analysis of this showed that there was not enough diffusion occurring and that this was not due to a programming bug. Furthermore, this discrepancy in boundary layer thickness only seemed to result from flows with high Reynolds numbers or, in other words, slightly viscous flows. For example, a flat plate calculation with a much lower Reynolds number produced the expected boundary layer profile.

Roache [30] discussed artificial viscosity for incompressible flow problems in some detail. He described it as a truncation error that appeared in some finite difference analogs. He used a Taylor series expansion of the governing finite difference equation to develop an expression for this truncation error. In the case of Equation 2.9 the transient artificial viscosity term is:

$$\alpha_e = -\frac{u^2 \Delta t}{2} \tag{5.1}$$

This kind of stability analysis is much more difficult for compressible flow since the compressible flow equations cannot be linearised [30, p. 228]. The governing equations could be expressed as a set of simultaneous linearised equations and analysed as such but that is beyond the scope of this work. The conclusion is therefore that there is some kind of artificial viscosity effect of the form shown in

Equation 5.1. Since this term is negative, it would have the effect of canceling out some of the explicitly modelled viscosity. This provides one explanation for the smaller than expected boundary layer.

Another change in the viscosity calculation implemented at the same time was a change in the values of μ_0 used in Equation 4.2 and the way that viscosity was calculated for a gas mixture. Previous values of μ_0 were calculated based on a Reynolds number that was supplied as a parameter for the problem. Further reading about Sutherland's law [16] revealed that this use of μ_0 was incorrect. The correct values were obtained from the GASP User's Manual [16]. The value for the helium argon driver gas was taken to be that of helium and the value for air was taken to be that of nitrogen (N_2). To deal with the change in the use of Sutherland's law, it was necessary to use real flow values to specify the problem. Previously, inlet flow values were scaled to unity to reduce the chance of roundoff error occurring in some calculations.

Previously, an average of μ_0 was used in the Sutherland's law calculation when calculating transport between gas species. Wilke's equation is now used to calculate the mixture viscosity [1, p. 24]. Wilke's equation is given by:

$$\mu_{mix} = \sum_{i=1}^{n} \frac{x_i \mu_i}{\sum_{j=1}^{n} x_j \Phi_{ij}}$$

where,

$$\Phi_{ij} = \frac{1}{\sqrt{8}} \left(1 + \frac{M_i}{M_j}\right)^{-1/2} \left[1 + \left(\frac{\mu_i}{\mu_j}\right)^{1/2} \left(\frac{M_i}{M_j}\right)^{1/4}\right]^2$$

and, the subscripts i and j denote the different gas species, x_i is the mass fraction of species i and M_i is the molecular weight of species i.

5.6 The tuning algorithm

During the development of the code it was found that the amounts of energy and momentum being generated by the pressure gradient effects were incorrect in regions close to shocks. After some investigation, it appeared that Milthorpe's upwinding coefficients were not appropriate for the driver gas contamination problem. A tuning algorithm was developed to tune the upwinding coefficients to produce more accurate results.

The particular tuning algorithm used was given the label 'hill climbing' (Figure 5.2). To visualise it imagine that we want to vary two parameters until some function depending on them was maximised. We can depict this function as a graphical surface and the maximum would then be a hill. To find the 'top' of the hill, one parameter is varied using a certain increment while we are still climbing the hill. Before we start going down, we change direction - or vary the second parameter. If we can't go any higher, we reduce the size of the steps we are taking and start varying the first parameter again. The top of the hill is reached when we can't go any higher and the step size has reached some minimum value.

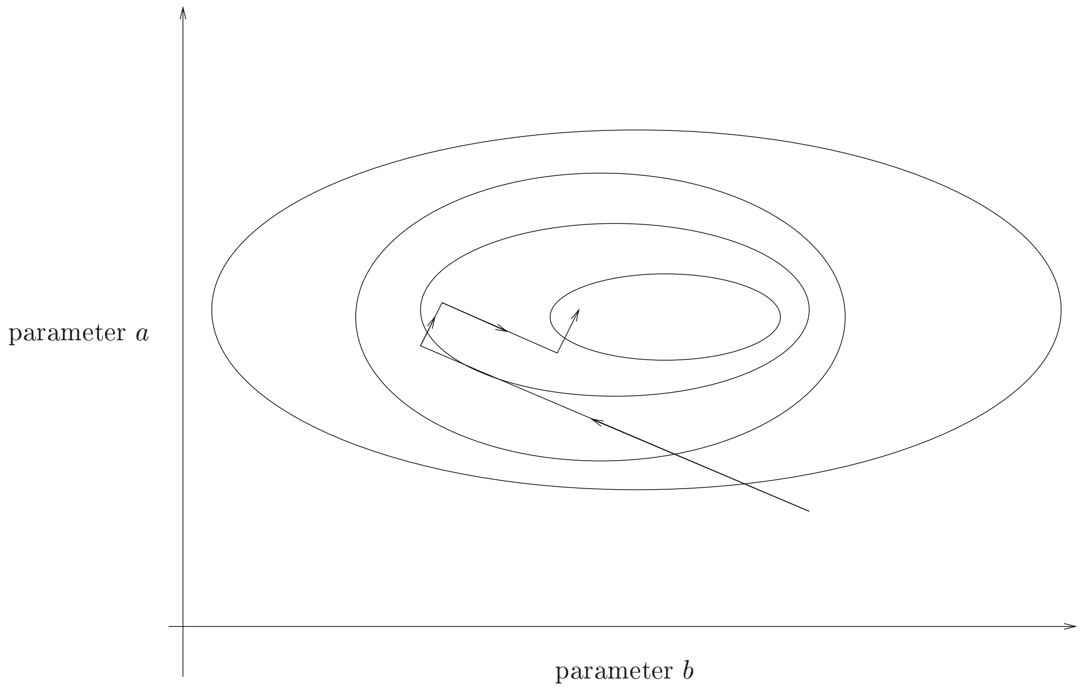
This algorithm is applied to the tuning of the upwinding parameters by getting CONVECT to repeatedly calculate the propagation of a one dimensional shock at the conditions of interest. At the end of each calculation, an error function is produced by comparing the shock profile against an analytical solution. The aim is to minimise this error function and, to use the analogy of the hill, when it is minimised we are at the top of the hill. By producing a one dimensional version of CONVECT that is in turn called by a program implementing the hill climbing algorithm, the whole process was automated quite efficiently. There were actually 30 parameters to vary so the hill climbing algorithm was effectively climbing a 31-dimensional hill!

5.7 Parallel implementation

5.7.1 Overview

One of the principal objectives of re-implementing Milthorpe's code was to introduce parallel processing capabilities. A parallel program is able to spread its processing load across two or more computer processors. The extra processors may be part of a multi-processor computer or a set of computers that are networked together or some combination of both. The ideal 'speedup' of a parallel program is the number

Figure 5.2: Illustration of hill climbing algorithm showing contours of a two parameter function



of processors - that is, running an ideal parallel program on three processors would go three times as fast as it would have on one processor. In reality, a somewhat less than ideal speedup is usually achieved due to the finite communication time between processors, the fact that parts of a computer program do not execute in parallel and that there is sometimes extra code to execute to implement the parallel behavior.

5.7.2 Implementation points

Message Passing Interface (MPI)

It was decided to use a freely available programming interface called MPI to implement the parallel nature of CONVECT. It is a set of computer program routines that provide the framework for writing parallel programs. MPI was used rather than other programming interfaces because it is freely available and it has been implemented on most flavours of UNIX. It is even becoming available for the Windows and Macintosh platforms. Using MPI makes CONVECT a fairly portable computer program. A computer program using MPI could technically run on a mixture of high performance multi-processor computers, several UNIX workstations, some

Windows based PCs and some Macintoshes, all at once and in parallel. In practice, it is simpler to get programs written with MPI to run on a single multi-processor computer or a set of networked UNIX workstations.

Data structures

When running under MPI, CONVECT runs as a set of separate processes, whether on a multi-processor computer or a set of different workstations. Each process has a single Domain that contains a single Region. These are special parallel versions of a Domain and a Region that were constructed by inheriting the existing non-parallel versions. One of the Domains has the responsibility of orchestrating the input and output of the whole parallel computer program. It instructs the parallel Regions to process themselves, and to exchange boundary information and other parameters between each other.

Figure 5.3: How Region boundaries are handled

Figure 5.3 shows how cells along the boundaries of two neighbouring parallel Regions are associated with each other. The cells adjacent to a parallel boundary are essentially ghost cells that are updated only from their associated cells in the

neighbouring Region. The current parallel version of CONVECT synchronises the timestep and some other parameters between all parallel Regions. Care has been taken during the development of this computer program to ensure that the fact splitting amongst several parallel Regions does not affect the solution.

Output file processing

The simplest way to extend the handling of data files when implementing CONVECT in parallel was to simply get each parallel Region to output its results to separate, numbered data files. This meant that some post-processing was required to produce a single, meaningful graphics plot. The ghost cells have to be stripped from the output and the data from the parallel Regions must be pasted together.

Chapter 6

Results

6.1 Tuning and development

6.1.1 Object-oriented development times and issues

A primary objective of this current work was to develop and test a C++ implementation of Milthorpe's algorithm. The practical reality of this software development is that the development and testing goes in cycles. Some computer code is developed, then it is tested. The result of the testing might show that there was some error in the code development so another cycle of development and testing occurs. Even if the testing shows that the code is working, the next cycle starts with development of the next feature of the code. It is more practical and efficient to develop a complex computer program in stages, adding and testing each feature or section one at a time.

Much of the time spent on this current work was in this development cycle. Features were added one at a time and there were several times where a feature that was initially thought to be working had to be re-developed and re-tested. In a fluid dynamics simulation such as this, it is quite difficult to isolate which calculation might be producing an erroneous result since most calculations are very much interdependent.

There was a reasonably steep learning curve that needed to be tackled to implement Milthorpe's algorithm in the new language. Once CONVECT was written and working to a certain point, however, it was relatively quick to implement further enhancements or bug-fixes. The 'building block' approach that an object-oriented programming language offers, meant that rarely did changes need to be made to existing test object classes.

6.1.2 Tuning results

The tuning algorithm described in Section 5.6 was applied to a one dimensional version of the shock tube problem. CONVECT was set to run on the driver gas contamination problem described in Section 3.4, except that only one gas species was used - the test gas. This shock tube problem involved the propagation of just the primary shock. The inlet conditions were set so that they matched the post shock conditions that resulted in no secondary shocks or expansion waves being present. This should theoretically produce a shock that appears as a step function in all flow variables. An error function, based on the theoretical shock location, was minimised as the upwinding parameters were varied. Figures 6.1 and 6.2 show the results before and after a series of tuning runs. Flow quantities have been normalised to show agreement with the ideal step function shape of the shock wave being simulated. The theoretical shock location is also shown in both diagrams. It can be seen that the tuning vastly improved the results, showing greater agreement with the theoretical shock.

Some experimentation was done with the number of cells in the tuning runs. Not surprisingly, it was found that agreement with the theoretical shock improved as the number of cells was increased.

The original upwinding coefficients are shown in Figure 6.3. The final coefficients produced by the tuning process are shown in Figure 6.4.

6.1.3 Flat plate calculations

To verify that the CONVECT code was indeed producing the same results as Milthorpe's original Fortran computer program, the calculation presented by Mallinson and Milthorpe [18] was reproduced (Figure 6.5). Mallinson and Milthorpe were investigating hypersonic laminar boundary layer flow both experimentally and numerically. This included a comparison of heat transfer and pressure data obtained in a free piston shock tunnel to the numerical results produced by Milthorpe's computer

Figure 6.1: Shock wave problem before tuning

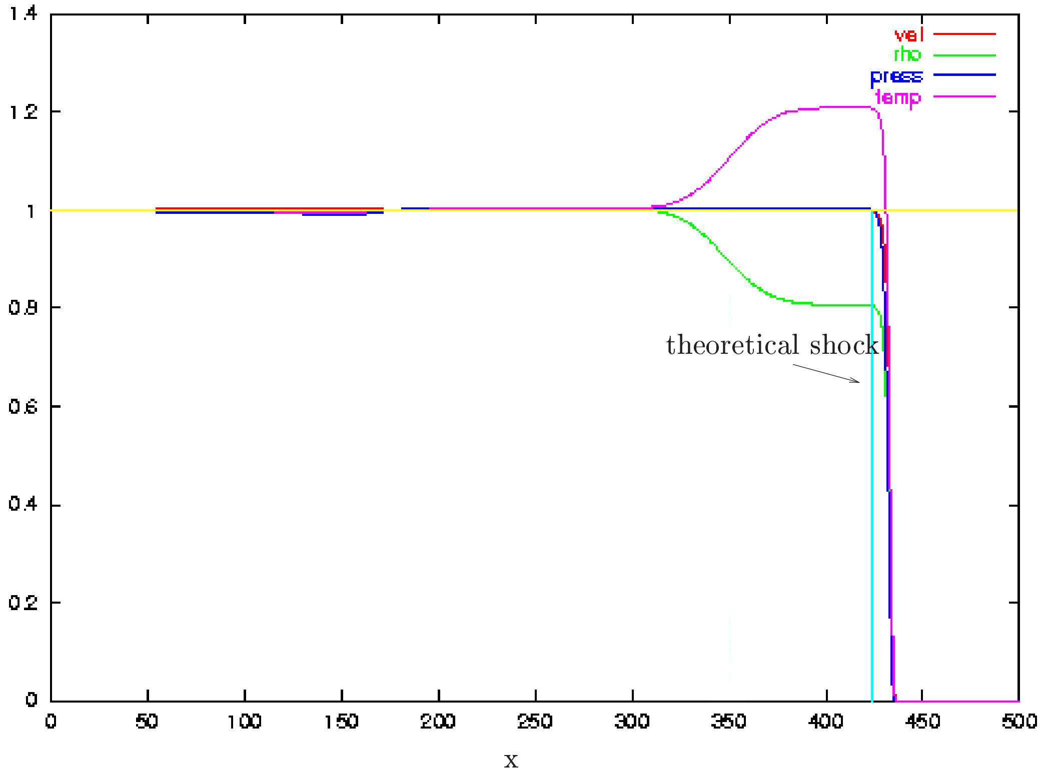


Figure 6.2: Shock wave problem after tuning

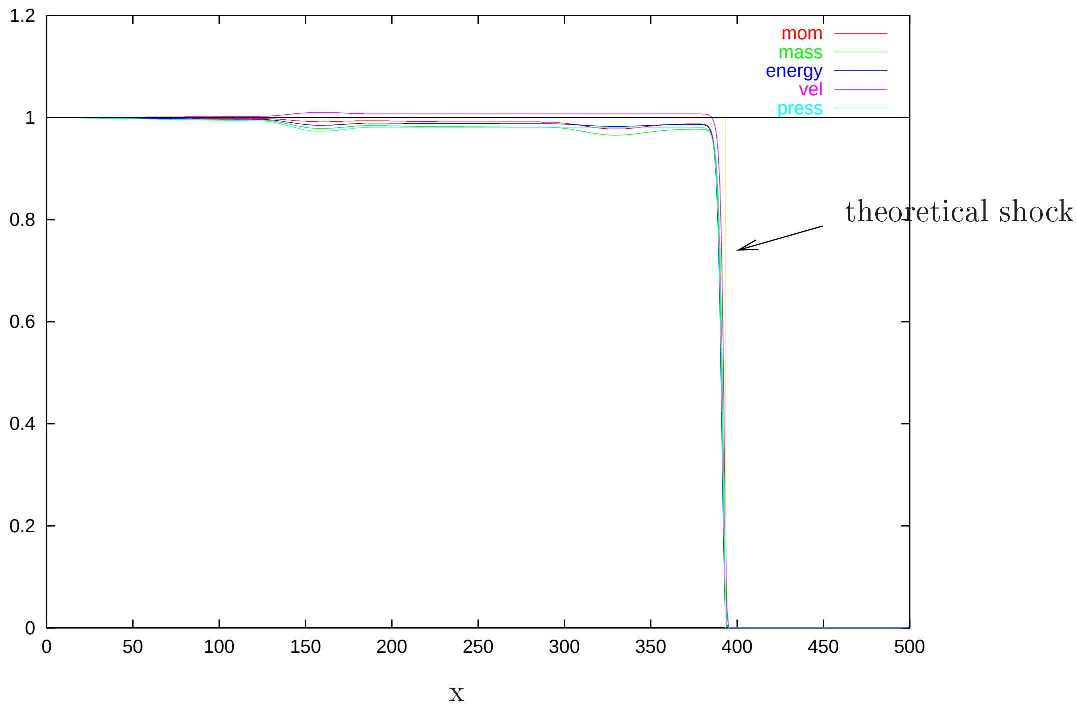


Figure 6.3: Original upwinding coefficients

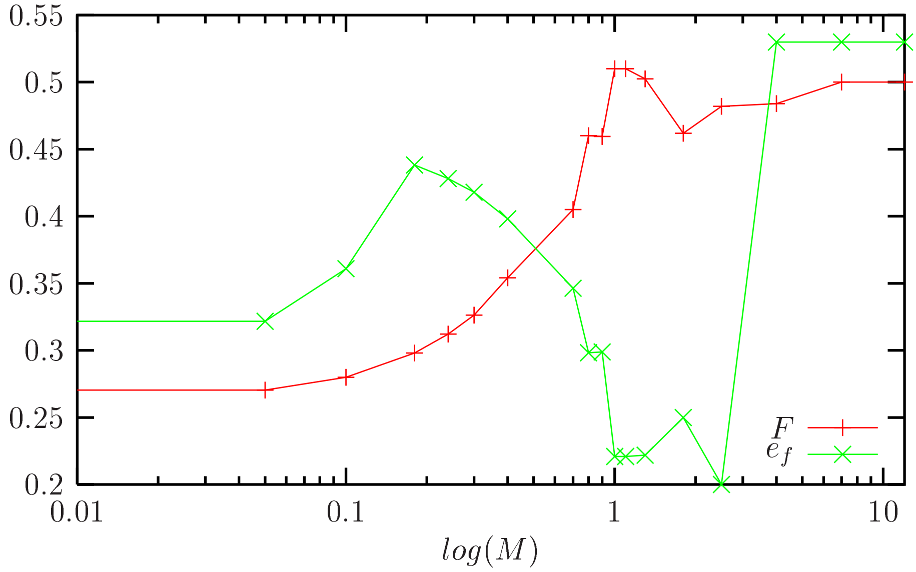
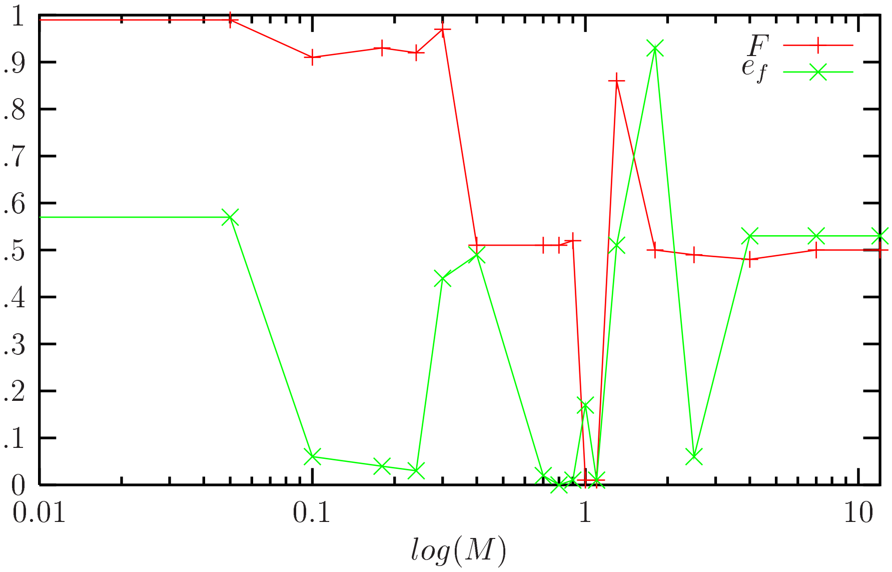


Figure 6.4: Upwinding coefficients after tuning



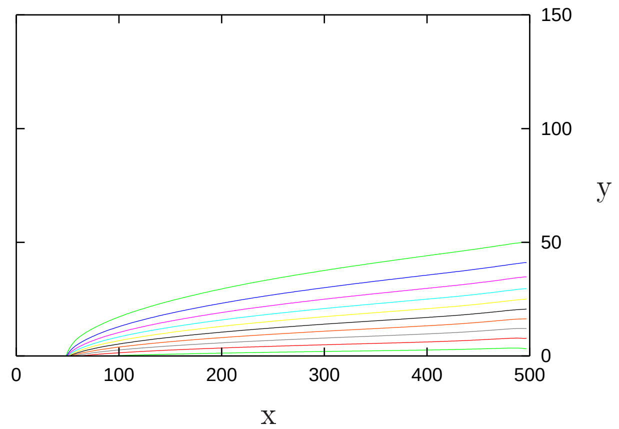


Figure 6.5: Velocity contours of flow over a flat plate

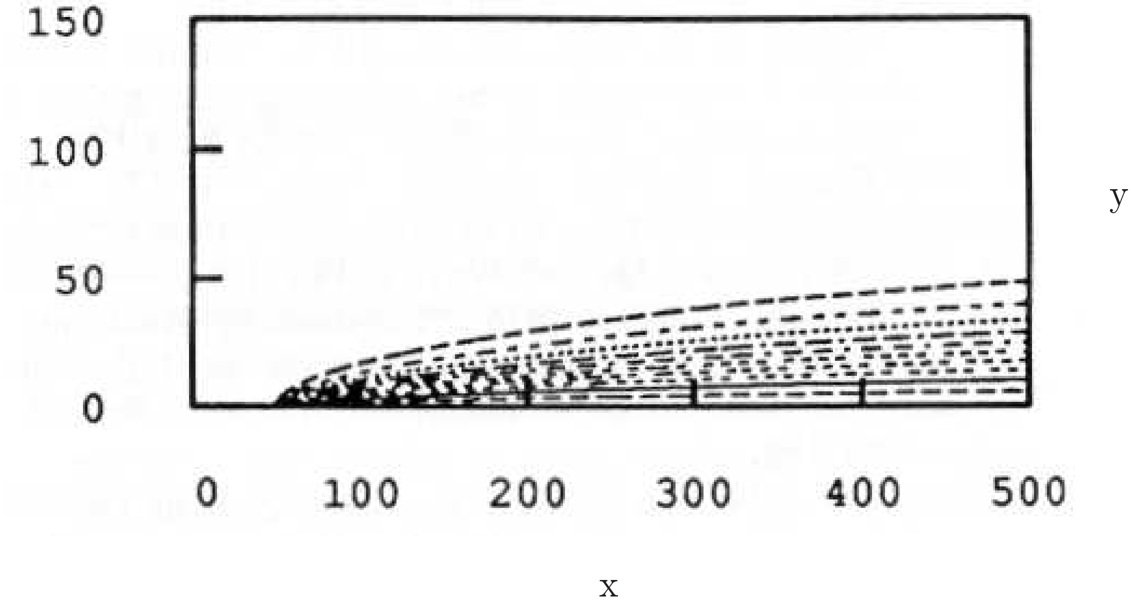
program. They showed that there was good agreement between their experiment and the numerical simulation. This was a good test case for demonstrating that CONVECT does indeed implement Milthorpe's algorithm and, indirectly, that the code simulates flow conditions produced in a free piston shock tunnel.

The numerical problem was a calculation over a domain of 500×150 cells. The left hand boundary was set to an inlet condition at the freestream flow values (Table 6.1). The lower boundary was set to a symmetry condition for the left most 50 cells and the remainder was set to a solid boundary condition. The upper and right hand boundaries were set to the freestream outflow condition. The results produced by CONVECT are shown in Figure 6.5.

Table 6.1: Freestream conditions for flatplate problem

p_∞	$0.69\,kPa$
T_∞	$155\,K$
ρ_∞	$g\,m^{-3}$
u_∞	$2.30\,km\,s^{-1}$
M_∞	9.2

Figure 6.6: Original contour plot of velocity field (scanned from Mallinson and Milthorpe [18])

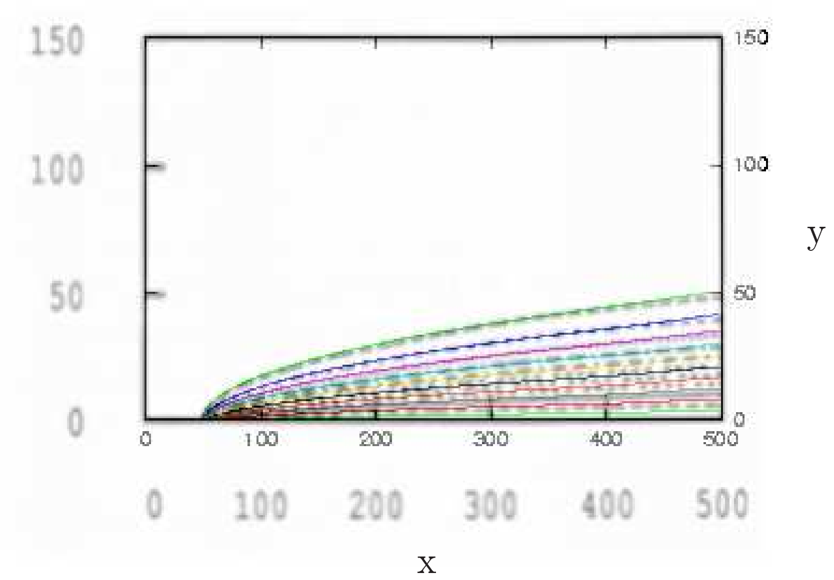


To enable overlays of these results with the original results obtained by Mallinson and Milthorpe [18], the output was scanned (Figure 6.6). Mallinson and Milthorpe stated (pers. comm., October 1999) that their plot was produced with 'Gnuplot' [13] and that the velocity contours were generated at automatically determined values of the velocity data. Gnuplot was also used to plot the results obtained by CONVECT (Figure 6.5) with the contours being generated automatically. If the contours of the results produced by CONVECT coincide with the Mallinson and Milthorpe plot, it can be concluded that both simulations were in fact producing the same results, even though we do not know the values at the contours. To show the coincidence of the contour lines the scanned plot was scaled, made transparent using an image manipulation package and overlaid on the CONVECT results. This is shown in Figure 6.7.

6.2 Early CONVECT results

The results of running the driver gas contamination problem outlined in Section 3.4 with an early version of CONVECT are shown in this section. These results were first presented at the 21st International Symposium on Shock Waves [20].

Figure 6.7: Original plot scaled, rendered transparent and overlaid on current plot



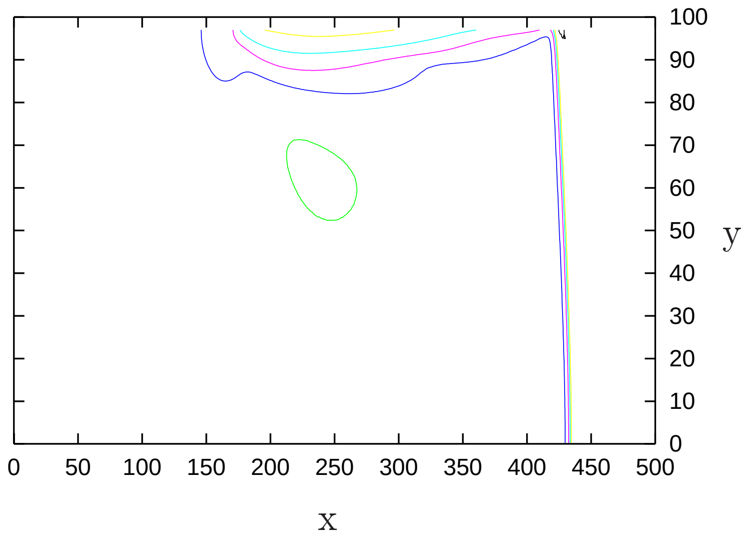
At that time it was apparent that the boundary layer produced in the simulation was much smaller than the theoretical prediction. Some of the parameters that the CONVECT code accepts in its input file are the Reynolds numbers for each gas species. This gets translated into the base viscosity, μ_0, used in the Sutherland viscosity calculation. In these early results the Reynolds number was adjusted so that the boundary layer was a more realistic size.

There was also some concern about the effect of non square cells and of round-off errors produced by small cell dimensions. In order to disregard these possible effects, cell dimensions were set so that $dx = dy = 0.5\,m$. This meant that the dimensions being simulated were different from a real shock tube. Table 6.2 shows some of the parameters used.

Table 6.2: Early CONVECT parameters

Re driver gas (HeAr)	427
Re test gas (Air)	2177
grid dimensions	500×100
problem dimensions (m)	75×25

Figure 6.8: Velocity contours before shock reflection



Figures 6.8, 6.9 and 6.10 show a velocity contour plot of the problem before and after shock reflection off the end wall. There is quite a sharp shock, which shows the effectiveness of Milthorpe's method to deal with this kind of problem. The boundary layer after the shock is well defined, however there should also be a boundary layer in the driver gas starting at the inlet. It is interesting to note the interaction of the reflected shock with the boundary layer. Shock bifurcation does not occur, but there are changes in the shock angle as it interacts with the boundary layer.

6.3 Parallel CONVECT results

6.3.1 Timings

The main reason for implementing a computer program in parallel is to achieve faster execution. As already stated, the ideal 'speedup' for a parallel computer program is the number of processors it is running on. In reality, the speedup is

Figure 6.9: Velocity contours just after shock reflection

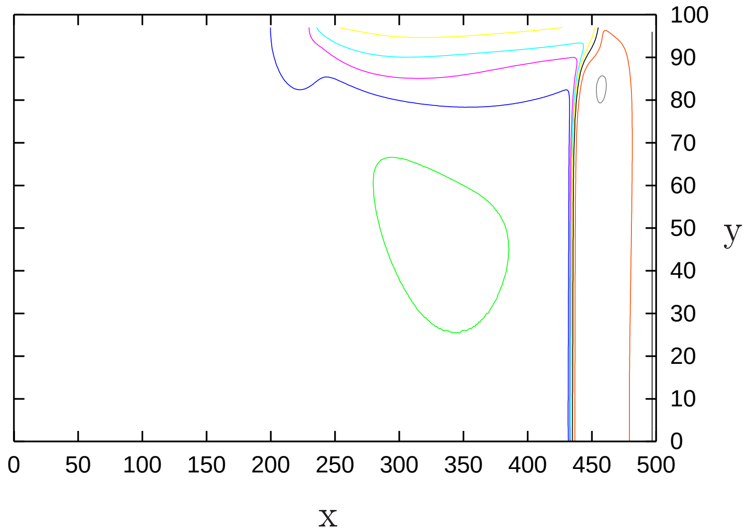
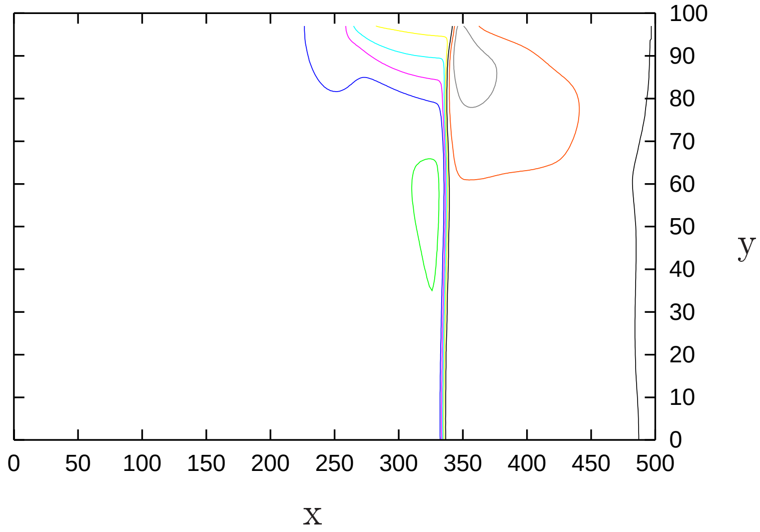


Figure 6.10: Velocity contours at a later time



often less than that due mainly to overheads in the parallel implementation. Table 6.3 shows some timings for a typical small problem using 150×50 cells using the parallel code.

Table 6.3: Parallel timings

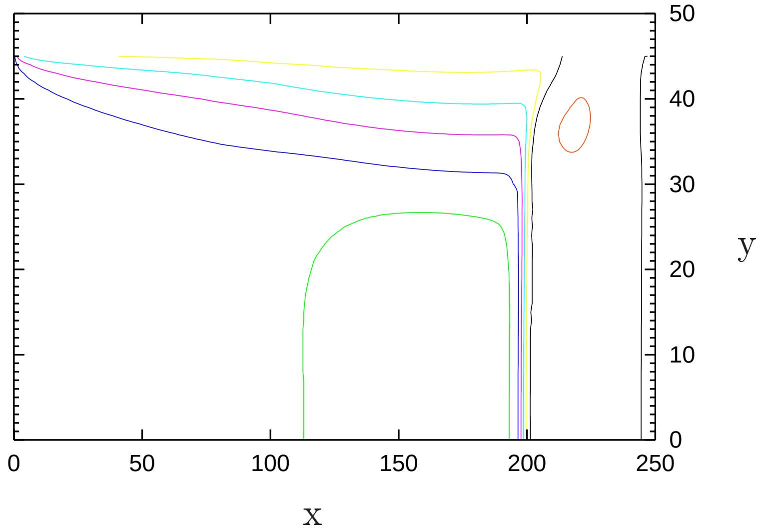
number of processors	time (min)	speedup
2	252	1.02
3	193	1.34
4	140	1.85
7	94	2.75

From this speedup information, it is clear that there are significant overheads for running the parallel code, and these are only overcome by using enough processors. A better indicator of the overheads associated with the parallel implementation is the fact that it took the original single processor version of CONVECT only 133 minutes to run this same problem! It is possible that these overheads would be less dominant on problems with many more cells to calculate over, but there is no timing data for this at the present.

6.3.2 Comparison with previous results

The driver gas contamination problem, whose results appear in Section 6.2, was re-run using the parallel version of CONVECT to verify that the parallel code worked. This was done first on four Sun Ultra1 workstations networked together. In that case the computational domain consisted of 250×50 cells. Figure 6.11 shows the solution after shock reflection. It can be compared with Figure 6.9, noting that the time at which this output was captured was a little further into the calculation. The results show a boundary layer growing from the inlet. This boundary layer appears in Figure 6.11 and not in Figure 6.9 because some bugs in the diffusion code had been fixed by the time the parallel version of CONVECT was implemented. Apart from the discrepancy in the boundary layers, the results are much the same suggesting that the parallel code is working. The main problem during development of the parallel code was dealing with the information flow across the parallel boundaries. The parallel boundaries in this case divide the domain into four equal slices in the

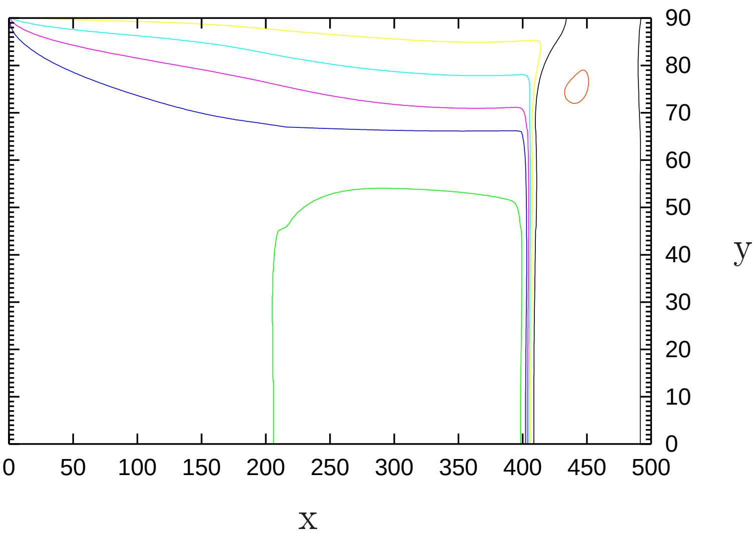
Figure 6.11: Velocity contours for a smaller parallel run



x direction. No fluctuations or other numerical artifacts appear in this result, so the parallel boundaries must be passing information correctly.

To verify that the parallel code could handle different sized problems, the same problem was run on a larger computational domain of 500×100 cells. This was done on a Sun Enterprise 4500 symmetric multi-processor computer using four of its processors. A contour plot of the calculated velocity is shown in Figure 6.12. It can be seen that changing the number of cells does not significantly change the results. It does, however, greatly impact on how long the calculation takes. Increasing the number of cells does make the solution more accurate, however the smaller timestep required means that it takes much longer to get to the same stage of the simulation. The number of calculations and the amount of computer memory the problem uses also increases with the number of cells, adding to the calculation time.

Figure 6.12: Velocity contours for a larger parallel run

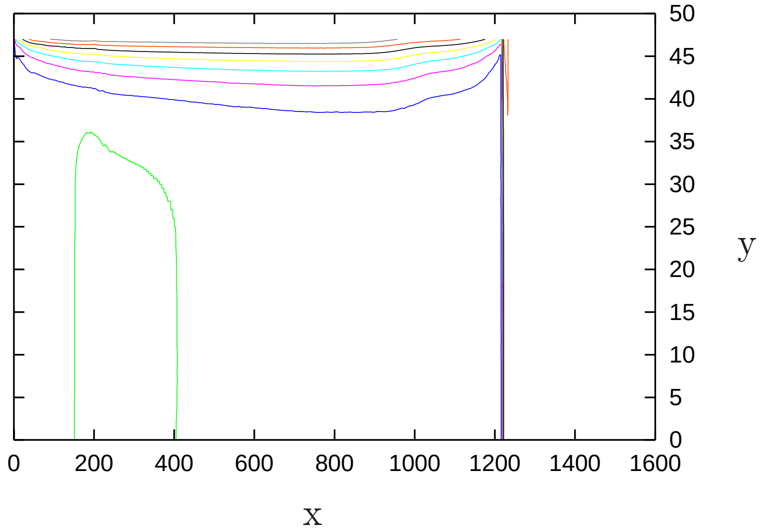


6.4 Artificial viscosity study

A series of runs exploring the inclusion of the artificial viscosity term (Equation 5.1), and other changes to the way viscosity was calculated (as outlined in Section 5.5.9), were carried out. Since the error term is negative, a positive term of the same form was added. This positive term was multiplied by a factor, G, which was varied until the expected boundary layer thickness was achieved. After some experimenting, a value of $G = 0.1$ was found to produce the best results. Shown here in this section are the results of a calculation of the driver gas problem outlined in Section 3.4. The computational domain used here was 1500×50 cells. To simplify the problem a little, and to observe the boundary layer development, just the first 1 meter of the shock tube was simulated.

Figures 6.13 and 6.14 show the flow variable velocity and pressure after the boundary layer has developed and before the shock has reflected from the end wall. Figure 6.15 shows a combined contour plot of the mass fraction of the driver gas and

Figure 6.13: Velocity contours for a shortened shock tube with artificial viscosity



the pressure (also shown in Figure 6.14). This also illustrates the distance between the primary shock and the contact surface. In these results, the contact surface is quite smeared, leaving only a small amount of uncontaminated test gas behind the primary shock.

Figures 6.16 and 6.17 show the flow values of velocity and pressure at a time after the shock has reflected off the end wall bringing the test gas at the end of the shock tube to rest. These results were calculated on a 150×50 cell grid due to time constraints.

6.5 Comparison to General Aerodynamic Simulation Program, GASP

Even though CONVECT was reproducing the results of Milthorpe's original code, there was some concern that there were inaccuracies in the shock tube driver gas contamination problem. A commercial computational fluid dynamics code, GASP

Figure 6.14: Surface plot of pressure for a shortened shock tube with artificial viscosity

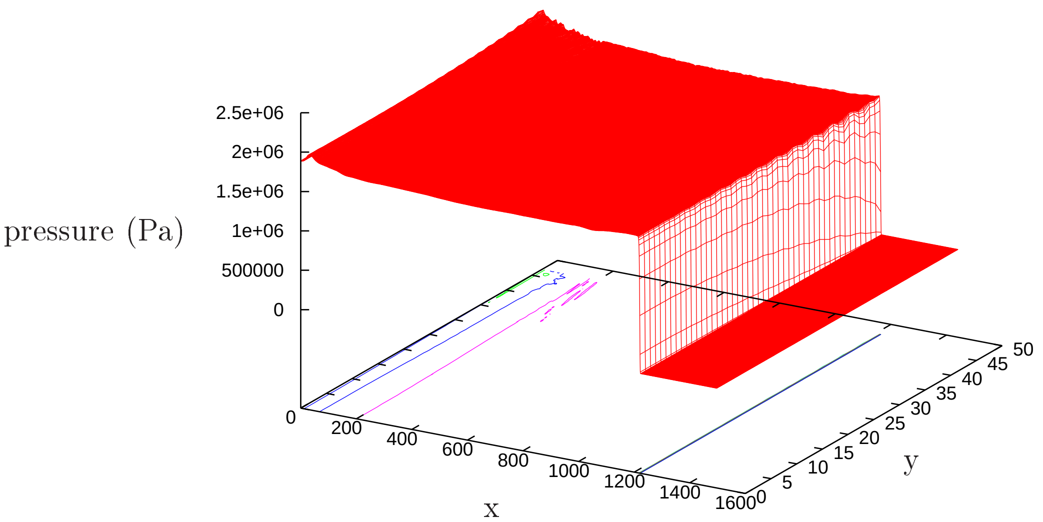


Figure 6.15: Combined contour plot of mass fraction of driver gas (X_a) and pressure, artificial viscosity

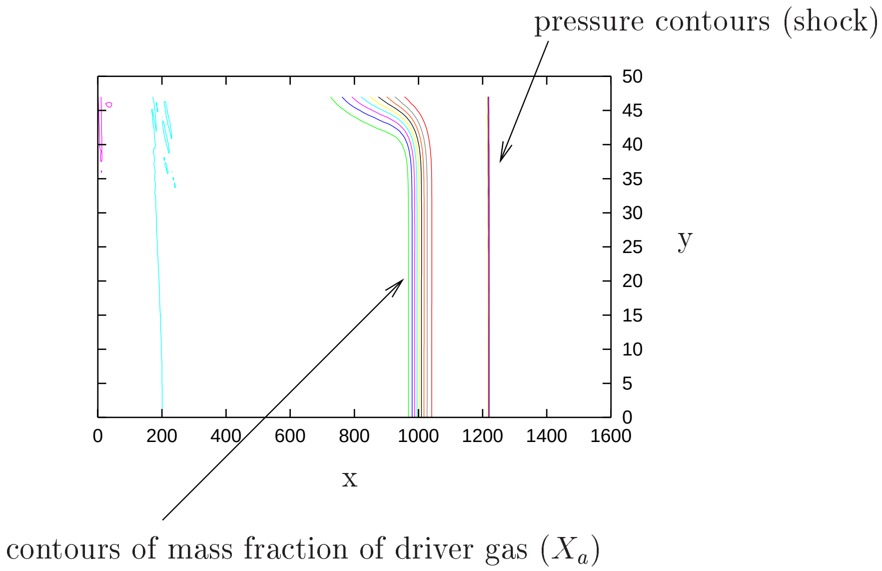


Figure 6.16: Velocity contours after shock reflection with artificial viscosity

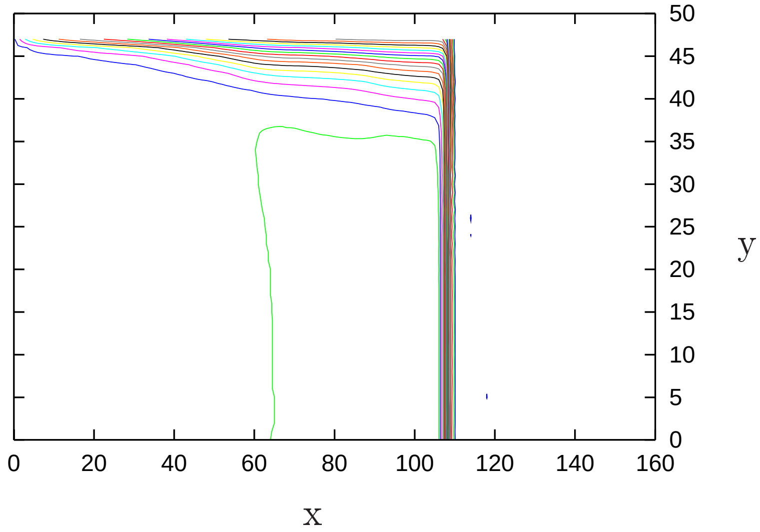
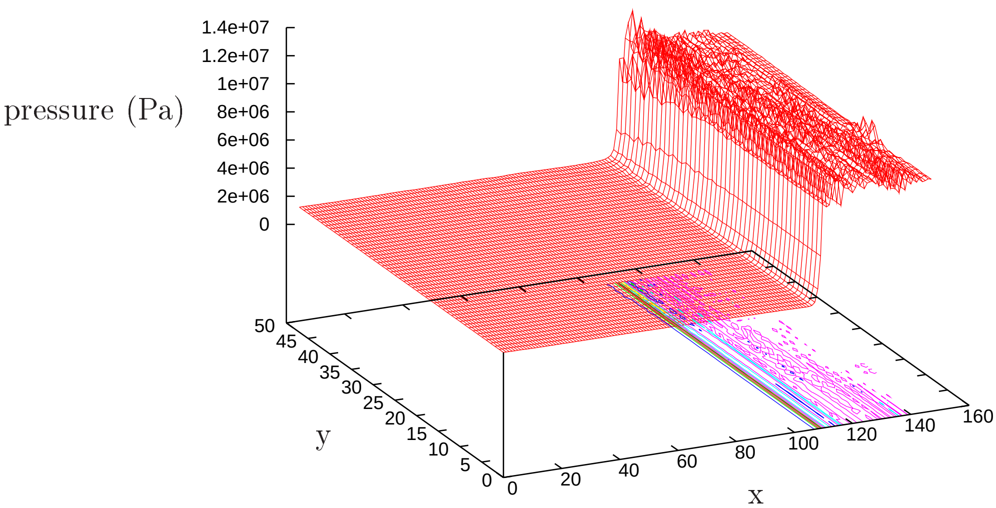


Figure 6.17: Surface plot of pressure after shock reflection with artificial viscosity



[16], was used to provide a comparison. GASP is a very capable computer program, but it takes some time to learn how to use it and apply it to particular problems. Transient problems, such as the driver gas contamination problem, seemed particularly difficult to set up since the documentation only gave examples of solutions to steady state problems.

As a preliminary step towards a full solution of the driver gas contamination problem the following calculation was set up. The flow conditions were taken from Section 3.4. The rest of the options were as follows:

- 129×33 grid points.
- inviscid calculation, ie. no boundary layer produced
- all boundaries except the inlet are specified as 'tangency'

The 'mesh sequencing' feature of GASP was used to improve the accuracy of the calculation. Mesh sequencing involves carrying out the calculation on a coarse grid first then interpolating the results up to a finer grid. In this test case, two levels of grid coarsening were used. The results of this calculation are shown in Figures 6.18 and 6.19. The calculation was completed in only a matter of minutes on a Pentium II 450 running Linux due to the small grid size.

With a successful one dimensional calculation completed, an attempt was made to fully simulate the driver gas contamination problem using GASP. The following options were used:

- 113×113 grid points on a grid that becomes finer as it approaches the wall
- lower boundary is 'tangency'
- upper boundary is 'no-slip' and adiabatic
- turbulent viscosity used but no cross-terms

It took 73 hours to complete the calculation on a Pentium II 450 running Linux. The plots in Figures 6.20 and 6.21 show the velocity results and mass fraction of the driver gas results respectively for this run.

This does not seem to be a good simulation of the problem since the level of the velocity (and other flow variables not presented) should have stayed at the inlet

Figure 6.18: Velocity plot for simple shock tube problem using GASP

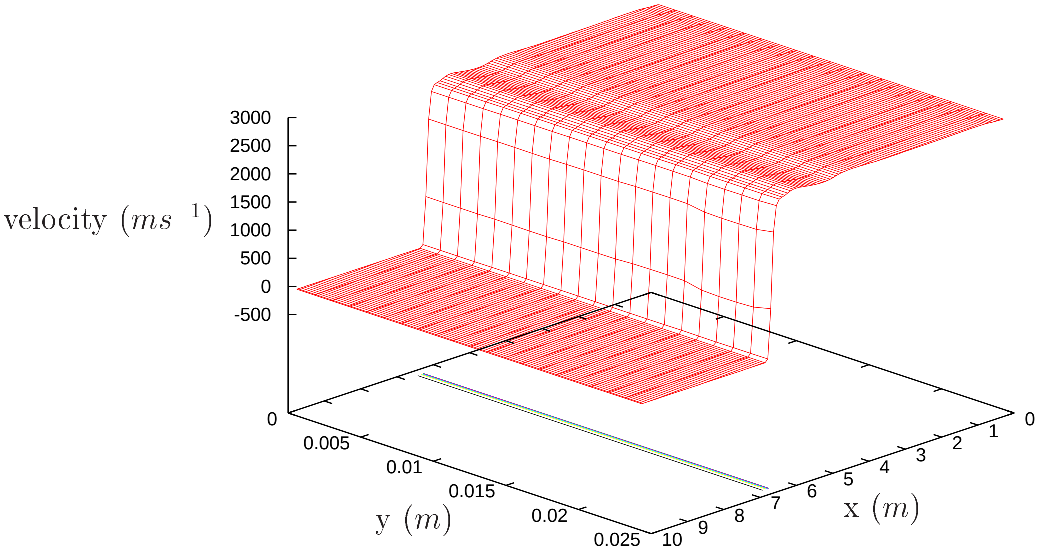


Figure 6.19: Plot of mass fraction of driver gas for simple shock tube problem using GASP

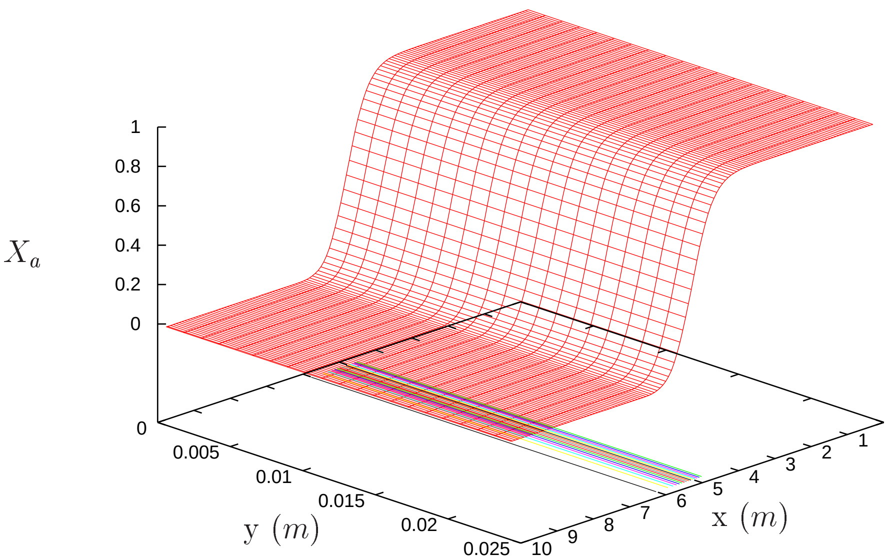


Figure 6.20: Velocity plot for GASP output

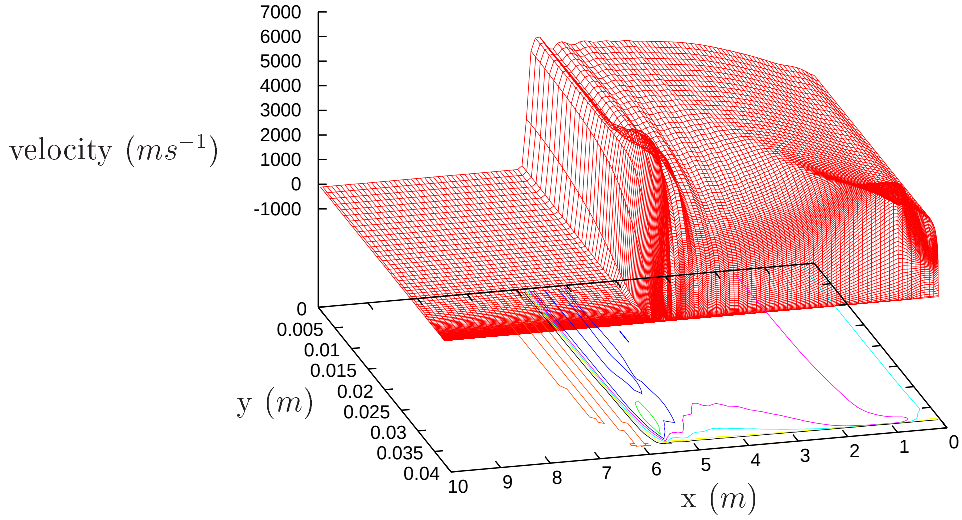


Figure 6.21: Plot of mass fraction of driver gas for GASP output

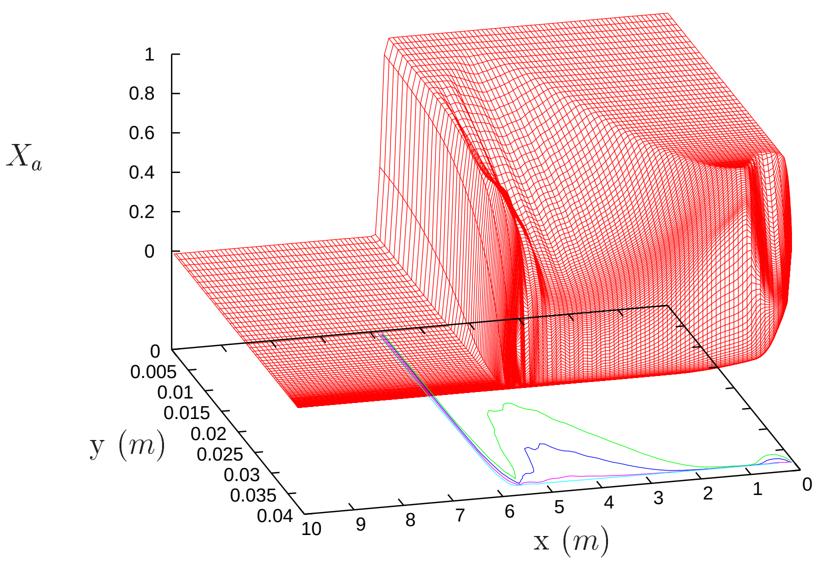
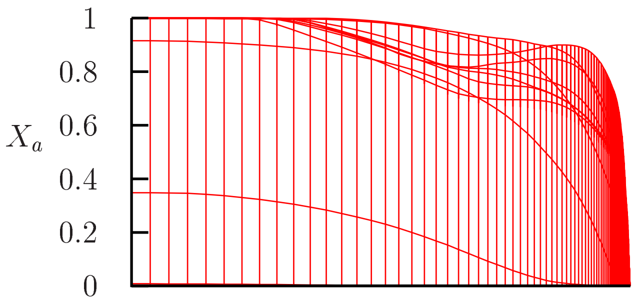


Figure 6.22: End on plot of mass fraction of driver gas for GASP output



condition up to the shock - except in the boundary layer. There seems to be a peak in the value of velocity at the shock near the wall. Figure 6.21 shows that there is a dip in the mass fraction of driver gas behind the contact surface. This result indicates that some air has made its way behind the driver gas. Figure 6.22 shows the same plot as Figure 6.21 with the view point at the downstream end of the shock tube. The mass fraction of the driver gas drops off near the wall. This could be interpreted as a vortex forming where the shock meets the wall, pulling the air into the boundary layer and leaving it behind the contact surface.

Chapter 7

Discussion

7.1 The implementation

7.1.1 Development time issues

The principal reason behind implementing Milthorpe's algorithm in C++ was to arrive at a code that would allow a computer programmer to implement changes in a more reliable and more efficient fashion. This goal has been achieved.

Once CONVECT was producing the same results as Milthorpe's computer program it was possible to start enhancing it so that it could tackle more difficult problems, such as driver gas contamination in a shock tunnel. In this section a number of examples are given to show how object-oriented programming techniques have reduced development time for CONVECT and ancillary code.

To implement the tuning algorithm it was not necessary to change CONVECT at all. The tuning algorithm was implemented as a drop-in replacement to CONVECT and managed the tuning of the parameters that it passed to CONVECT's objects. One of the design features of CONVECT was quite useful here. Through most of the development, testing and execution of CONVECT, the computational domain was two dimensional. The tuning took place on a one dimensional domain that was implemented by making one dimensional versions of only some of the objects - through the object-oriented feature of inheritance. Most of the code was written to deal with an arbitrary problem dimension that was specified as a compile time parameter. Thus, there was no time wasted in writing a new version of the computer program and there was no danger of introducing new bugs to the existing code.

Again, the implementation of CONVECT as a parallel computer program involved very little change to the existing code. In fact, the modular nature of the computer program made it possible to continue developing and testing the original version of the code whilst implementing the parallel version. As with the implementation of the tuning algorithm, the parallel version was implemented as a drop-in replacement for CONVECT that implemented the parallel code through inheritance of existing objects.

The implementation of the artificial viscosity term was not particularly enhanced by the object-oriented nature of CONVECT. All that was required was to introduce a new input parameter for certain functions and to rewrite some of the Transport object's methods. This kind of change would have probably taken about the same time to implement in Milthorpe's original Fortran program.

For a team of researchers all using versions of the same code, the C++ implementation is a better choice because of its modular nature. A researcher does not need to change the core code, which may affect the other people using it. The researcher need only create a new version of the objects required to implement the new features. This approach is not really possible with Milthorpe's Fortran program. If more than one person were developing it at the same time then they would have to work on different versions of the code which would quickly diverge. This would make it harder to merge the changes each researcher created into a single version of the code.

C++ does have a role to play in scientific computer programming, especially for dealing with user interfaces and for modelling particular abstract concepts. For the purposes of understanding certain fluid dynamics problems better, it was not worthwhile rewriting existing Fortran code. It would have been better to take a hybrid approach where new data structures, such as those needed in a parallel implementation, could be incorporated with C++ whilst leaving the existing Fortran code to do the underlying computations.

This approach has been taken by Crutchfield and Welcome [8] in their Adaptive Mesh Refinement computer program. They used existing Fortran solvers to integrate the solution over subsets of the problem domain. The overall data structure and adaptive mesh algorithm was handled by a C++ program. Their methodical approach to development in this way created a set of computational tools that they and other researchers have developed further and successfully applied to several problems.

Another example of mixing the features of C++ and Fortran code is the implementation of the commercial computational fluid dynamics package GASP [16]. Looking at the source code, it appears that all the problem solving code is written in Fortran, due to historical reasons. In commercialising the product a graphical interface to the problem solving code was constructed and this has been implemented in C++.

Several lessons have been learnt from this work. It is not good computer programming practice to 're-invent the wheel'. It is better to use existing code that works, rather than spend time writing code that does the same thing. There are several good computer programming languages available. If new code is being developed it is good to use the right language for the job. One needs to weigh the benefits of using the right language against the time it takes for researchers involved to learn that language.

7.2 Tuning

The tuning process worked very well. The results presented in Section 6.1.2 show clearly how the new upwinding parameters produce a much better agreement with the theoretical predictions.

The parameters that were originally taken from Milthorpe's computer program gave the required upwinding value for particular local Mach numbers. Extra parameters defined at specific Mach numbers were introduced to refine the solutions for particular flow problems. In particular, several parameters were specified at subsonic Mach numbers to allow for the correct shock standoff distance to be calculated in a blunt body problem that Milthorpe investigated [25]. Milthorpe also found that existing parameters sometimes had to be adjusted to suit new problems [27].

Figures 6.3 and 6.4 show that all the parameters, except those for the highest Mach numbers, had to be adjusted. For the low Mach numbers, the change went from a small amount of upwinding in the momentum calculation to full upwinding. The current work had quite a different range of Mach numbers - generally much higher. Few of the original parameters were left unchanged by the process of tuning them to suit the driver gas contamination problem.

It would be good to achieve a more versatile approach to choosing upwinding coefficients, one in which the upwinding coefficients would not need to be tuned for each new problem. The Mach numbers at which upwinding parameters are defined could be investigated to see if a better set could be found. It is possible that using the local Mach number as a basis for choosing these parameters is not the best approach. An alternative that would be worth investigating is to use some function of the local cell Reynolds number (Equation 7.1).

$$Re_c = \frac{\rho u \, dx}{\mu} \tag{7.1}$$

7.3 Artificial viscosity study

Comparing Figure 6.13 with Figure 3.4 it can be seen that, with the artificial viscosity included, CONVECT produces reasonable laminar boundary layers when simulating a shock tube. The size of the boundary layer that was produced by CONVECT was larger than that predicted by theory. A better match could be achieved by reducing the value of G.

To tune G in a way similar to that used for the upwinding parameters, it would be necessary to construct an error function that would be at a minimum when G was optimised. This would take more time to tune since this is a two dimensional problem. It is quite possible to do this.

7.4 Comparison with GASP

The fact that GASP did not produce results that were as good as the CONVECT results was not necessarily due to a lack of capability in the product. The intention was to run GASP to produce a comparison for discussion but not to spend a great deal of time doing so. In the time allowed, it was difficult to get an acceptable solution using GASP. Those results presented in Section 6.5 that include viscosity show that a transient, if somewhat unstable, solution was achieved. There was no separation of the shock from the contact surface. This, and the appearance of air behind the contact surface, show that the problem was not simulated adequately.

GASP is widely used and is well validated for a range of compressible flow problems. It is believed that GASP might produce better results for this driver gas contamination problem given enough time. However, for certain problems it would be simpler to use more specialised code, such as CONVECT or Milthorpe's computer program since they are already set up for the job.

7.5 Driver gas contamination

The best results for simulating the driver gas contamination problem were those shown as part of the artificial viscosity study. The features of a shock, boundary layer and contact surface are all present. The contact surface is quite smeared and it was found that this occurred over a constant number of cells. Thus, it was possible to produce a better defined contact surface by running a problem with more cells in the x-direction.

No bifurcated shock system appears as the reflected shock interacts with the boundary layer. Better examples of a simulation of the reflected shock-boundary layer interaction can be found in [34] and [2]. Both of these show the jetting of driver gas through the bifurcated shock system. They both also start the calculation near the end wall assuming an initial boundary layer profile and a test gas slug length. Vortex flow formations that were present in these other works have not appeared in any of the results produced by CONVECT. This may be because Milthorpe's algorithm does not, at this stage, deal with turbulence.

7.6 Conclusions

CONVECT is an improvement on Milthorpe's original computer program. It produced the same results and it also introduces some improvements on the functionality of the original program such as the ability to compute in parallel. The fact that it has been re-written in C++ should make further development of the code more efficient and less prone to errors.

C++ is generally a good programming language to use for new projects in CFD. If there are existing routines that have been developed in another language, however, it is best to leave these intact and interface new C++ code with them. Where there is no optimised C++ compiler for a particular computer architecture, it would be best to use the optimised compilers that are available.

Milthorpe's algorithm, as implemented in CONVECT, was able to simulate a shock tunnel to some degree. It certainly had more success than the attempt made with the commercial CFD package GASP. The results produced by CONVECT, however, did not provide any further insight into possible causes of test time limitation due to driver gas contamination in shock tunnels.

Further work could gain more insight into the driver gas contamination problem by introducing features such as a turbulence model. Computations over a greater number of cells would improve accuracy. The availability of faster computers and the use of the parallel code would speed up the process of producing results for larger problems. The shock tunnel could be modelled better by including a representation of its nozzle in the simulation.

References

[1] R.B. Bird, W.E. Stewart, and E.N. Lightfoot. *Transport Phenomena*. John Wiley & Sons, Inc., Department of Chemical Engineering University of Wisconsin Madison, Wisconsin, 1960.

[2] M. Cardoso, Y. Burtschell, D. Zeitoun, and R. Abgrall. Numerical simulation about different methods for avoiding driver gas pollution in shock tunnels. In A.F.P. Houwing, editor, *Proceedings of the 21st International Symposium on Shock Waves*, July 1997.

[3] Dryden Flight Research Center. Hyper-X at mach 7. `http://www.dfrc.nasa.gov/gallery/photo/Hyper-X/Small/ED97-43968-1.jpg`, April 1998.

[4] M. Conner. Hypersonic experimental vehicle. `http://www.dfrc.nasa.gov/Projects/HyperX/`, November 1998.

[5] R. Courant and K.O. Freidrichs. *Supersonic Flow and Shock Waves*. Interscience Publishers, Inc., 1948.

[6] R. Courant, K. Friedrichs, and H. Lewy. On the partial differential equations of mathematical physics. *IBM Journal*, pages 215–324, March 1967.

[7] K.C.A Crane and R.J Stalker. Mass-spectrometric analysis of hypersonic flows. *J. Phys. D: Appl. Phys.*, 10(5):679–695, 1977.

[8] W.Y. Crutchfield and M.L. Welcome. Object-oriented implementations of adaptive mesh refinement algorithms. *Scientific Programming*, 2:145–156, 1993.

[9] L. Davies and J.L. Wilson. Influence of reflected shock and boundary-layer interaction on shock-tube flows. *The Physics of Fluids Supplement I*, 12(6):I–37–I–43, June 1969.

[10] J.I. Erdos. Scramjet testing in shock-heated tunnels. In A.F.P. Houwing, editor, *Proceedings of the 21st International Symposium on Shock Waves*, July 1997.

[11] S.L. Gai and J.F. Milthorpe. Hypersonic high enthalpy flow over a blunt-stepped cone. unpublished paper from Milthorpe.

[12] R.A. Gentry, R.E. Martin, and B.J. Daly. An eulerian differencing method for unsteady compressible flow problems. *Journal of Computational Physics*, 1:87–118, 1966.

[13] Gnuplot. `http://www.cs.dartmouth.edu/gnuplot_info.html`, May 1999.

[14] F.H. Harlow. The particle-in-cell computing method for fluid dynamics. In Berni Alder, Sydney Fernbach, and Manuel Rotenberg, editors, *Methods of Computational Physics*, volume 3, page 319. Academic Press, New York, 1964.

[15] W. F. Hughes and J. A. Brighton. *Theory and Problems of Fluid Dynamics*. McGraw-Hill, Inc., 1991.

[16] AeroSoft Inc. *General Aerodynamic Simulation Program User's Manual*. March 1997. export version 3.

[17] H.W. Liepmann and A. Roshko. *Elements of Gasdynamics*. John Wiley and Sons, New York, 1957.

[18] S. G. Mallinson and J. F. Milthorpe. An experimental and numerical study of hypersonic flat plate flow. In *Twelfth Australian Fluid Mechanics Conference*, pages 803–806. University of Sydney, December 1995.

[19] H. Mark. The interaction of a reflected shock wave with the boundary layer in a shock tube. Technical report, NACA, 1958.

[20] S.J. McMahon and J.F. Milthorpe. A numerical investigation of shock tunnel driver gas contamination. In A.F.P. Houwing, editor, *Proceedings of the 21st International Symposium on Shock Waves*, July 1997.

[21] D. Mee, H. Alesi, and M. Blair. HyShot project. `http://www.mech.uq.edu.au/hyper/hyshot/`, October 1999.

[22] J.F. Milthorpe. Numerical simulation of two-dimensional compressible flows. In C. Taylor, J.H. Chin, and G.M. Homsy, editors, *7th International Conference on Numerical Methods in Laminar and Turbulent Flows*. Pineridge Press, 1991.

[23] J.F. Milthorpe. Simulation of supersonic and hypersonic flows. *International Journal for Numerical Methods in Fluids*, 14:267–288, 1992.

[24] J.F. Milthorpe. Simulation of supersonic and hypersonic flows. *International Journal for Numerical Methods in Fluids*, 14:267–288, 1992.

[25] J.F. Milthorpe. Simulation of supersonic flows with detached shocks using a property convection method. *International Journal for Numerical Methods in Fluids*, 19:815–825, 1994.

[26] J.F. Milthorpe. Numuerical simulation of hypersonic flow around a hemispherical body. In A.F.P. Houwing, editor, *Proceedings of the 21st International Symposium on Shock Waves*, August 1997.

[27] J.F. Milthorpe. personal communication, November 1999.

[28] H. Mirels. Shock tube test time limitation due to turbulent-wall boundary layer. *AIAA Journal*, 2(1):84–93, 1964.

[29] William J. Rider. A review of approximate riemann solvers with godunov's method in lagrangian coordinates. *Computers Fluids*, 23(2):397–413, 1994.

[30] P. J. Roache. *Computational Fluid Dynamics*. Hermosa Publishers, Albuquerque, New Mexico, 1972.

[31] K.A Skinner. *Mass Spectrometry in Shock Tunnel Experiments of Hypersonic Combustion*. PhD thesis, University of Queensland, 1994.

[32] R.J. Stalker. Development of a hypervelocity wind tunnel. *The Aeronautical Journal*, 76(738):374–384, 1972.

[33] B. Stroustrup. *The C++ Programming Language*. Addison Wesley, 2nd edition, August 1992.

[34] M. Takahashi and K. Itoh. Numerical simulation of driver gas contamination in a reflected shock tunnel. In Allan Paull, editor, *The 4th International Workshop on Shock Tube Technology*, University of Queensland, Brisbane, 1994.

Appendix A

Files that make up the CONVECT computer program

These are the files that make up the current version of CONVECT and a brief description of each of them.

boundary.gp Gnuplot file producing a "lamindary flat plate" plot of the shock tube boundary layer

calcinit.pl Perl program to produce an inputs file containing the matched shock conditions for a shock tube problem

cell.c Methods for the Cell class and its children

cell.h Header file for the Cell class and its children

cellbuffer.c

cellbuffer.h Header file for the CellBuffer class and associated classes

conditions.c Declarations and functions dealing with global constants and parameters

conditions.h Header file associated with conditions.c

convect.c Main program file set up for a two dimensional, two species shock tube problem

convect1.c Main program file set up for a two dimensional, single species shock tube problem

convect1d.c Superseded by oned.c

convection.h Implements the Convector class

diffusion.h Implements the Diffuser class

domain.c Methods for the Domain class and its children

domain.h Head file for the Domain class and its children

dstrip.pl Perl program which simply removes the edge cells from CONVECT results in preparation for plotting with Gnuplot.

energy.gp Gnuplot file plotting a cross section of energy for a two species problem

flatplate.gp Gnuplot file plotting contours of velocity for the flat plate problem

globals.h Definition of global parameters

graph.gp Gnuplot file plotting contours of velocity, total pressure and mass fraction of driver gas for a two species problem. Uses output of graph.pl

graph1.gp Gnuplot file plotting contours of several flow variables for a single species problem. Uses output of dstrip.pl

graph.pl Perl program which prepares data files for plotting several variables in a two species problem. Originally written to work with graph.gp

inputs Allows user to specify a variety of parameters and problem conditions

makefile Used to build and maintain the application

mass.gp Gnuplot file plotting a cross section of mass for a two species problem

mom.gp Gnuplot file plotting a cross section of momentum

multiplot.gp Gnuplot file plotting contours of mass fraction of driver gas as well as velocity on the same plot. Uses output of graph.pl

myvector.h The vector template class

oned.c Main program file set up for a one dimensional two species shock tube problem

oned.h Head file for one dimensional versions of various classes

oned1.c Main program file set up for a one dimensional one species shock tube problem

onedimp.c Implementation of methods for one dimensional version of various classes

par.gp Gnuplot file which plots contours of velocity for a parallel computation. It uses the output of parprep.pl parprep.pl Perl program which merges the results file from several parallel instances of the CONVECT program to produce a single data file ready for plotting.

pconvect.c Main program file set up for a parallel run of a two dimensional, two species shock tube run

pdomain.c Implementation of methods for parallel version of several classes

pdomain.h Header file for parallel version of several classes

predict.c superseded by calcinit.pl

press.gp Gnuplot file plotting cross sections of partial pressure of each of two species as well as the total pressure

rcheck.gp Gnuplot file which plots a cross section of several flow variables normalised as well as a theoretical shock location. It uses output from rcheck.pl

region.c Implementation of methods for Region class and its children

region.h Head file for Region class and its children

rsync_excludes

setup.gp Gnuplot file preparing the plotting environment to do contour plots

species.c Implementation of methods of Species class and its children

species.h Head file for Species class and its children

squeezing.h Implements the Squeezer class

timecontrol.c

timecontrol.h

transport.c Implementation of methods for Transport class

transport.h Header file for Transport class

tune.c Main program file for running the tuning algorithm over a set of parameters in two dimensions

tune1d.c Main program file for running the tuning algorithm over a set

```
of parameters in one dimension
```

Appendix B

An example inputs file

```
#inputs file generated by calcinit.pl
reynold1 167252828.281643
reynold2 2961009
MAXX 50
MAXY 50
DX 10
DY 0.038
RHO0 0.265109860271596
T0 0.602669125246347
U0 2.1673292633449
PRESS0 1
RHOINT 0.174371301775148
TINT 0.045507765856486
tmax 1
C13 1.2
CHANGETOL 5e-7
writefreq 500
maxiter 3
```

Appendix C

Calcinit.pl

```
#!/usr/bin/perl -w
# $Id: thesis.tex,v 1.21 1999/11/07 01:40:18 sjm Exp $
# calcinit.pl
#    Calculates initial conditions for use with convect.
use Getopt::Std;
# parameters
$visc0m = 4e-3;# set to reduce numerical fluctuations.
$press1m = $temp1m = 1;
$MAXX = 500; $MAXY = 100;

# given conditions
$leng = 75; $wid = 25;
$rconst = 101325*28.966/1.225014/288.16;
$mach = 10.4;
$gamma = 1.4;
$gammaHeAr = 2.5/1.5;
$MWair = 28.966;
$MWHeAr8515 = 4.628;

# condition 0. test gas
$press0 = 15000;
$temp0 = 288.16;
$rho0 = $press0*$MWair/$rconst/$temp0;

# Calculations from J.K.Wright "Shock Tubes"
$mu = ($gamma-1)/($gamma+1);
$y = $mach*$mach*(1+$mu)-$mu;
$uona0 = (1-$mu)*($mach-1/$mach);
$eta = ($mu+$y)/(1+$mu*$y);
$T1onT0 = $y*(1+$mu*$y)/($mu+$y);

# condition 1. post shock test gas
$press1 = $y*$press0;
$rho1 = $eta*$rho0;
$temp1 = $T1onT0*$temp0;
$a0 = sqrt($gamma*$press0/$rho0);
$u = $uona0*$a0;

# Air calculations
```

```
# from Bird, Steward & Lightfoot "Transport Phenomena"
# this is about 2.97
$ktoneair = $temp0/97;

$funcair = 1.048+($ktoneair-2.9)/0.1*(1.039-1.048);
$visc0air = 0.1*0.000026693
                  *sqrt($MWair*$temp0)/3.617/3.617/$funcair;
$reynoldsair = $rho0*$leng*$u/$visc0air;

# Condition 3. drive gas
# reservoir pressure and temperature from Skinner
# reservoir pressure and temperature from Skinner
$temp3 = 4598;
$a3 = sqrt($gammaHeAr * $temp3 * $rconst / $MWHeAr8515);
$press3 = $press0*$y*(1-($gammaHeAr-1)*($a0/$a3)
                 *($y-1)/sqrt(2*$gamma)
                 /sqrt(2*$gamma+($gamma+1)*($y-1)))
                 **(-2*$gammaHeAr/($gammaHeAr-1));

# Condition 2. post contact surface, driver gas
$press2 = $press1;# same as press1
# from Liepmann & Roshko
$temp2 = $temp3*($press2/$press3)**(($gammaHeAr-1)/$gammaHeAr);
$rho2 = $press2*$MWHeAr8515/$rconst/$temp2;

# HeAr8515 calculations
$ktoneHeAr = $temp2/(10.2*0.85+124*0.15);
$funcHeAr = 0.6335+($ktoneHeAr-60)/10*(0.6194-0.6335);
$visc0HeAr = 0.1*0.000026693*sqrt($MWHeAr8515*$temp2)
                      /2.576/2.576/$funcHeAr;
$reynoldsHeAr = $rho2*$leng*$u/$visc0HeAr;

# check command line options
getopts('rsn:');
if (defined $opt_r) {
    undef $opt_r;
    # just use the real conditions for output
    $DX = $wid;
    $DY = $leng;
    $rho1m = $rho1;
    $press1m = $press1;
    $temp1m = $temp1;
    $rho0m = $rho0;
    $um = $u;
    $temp0m = $temp0;
    $rho2m = $rho2;
    $temp2m = $temp2;
    $press2m = $press2;
} else {
```

```
    # calculate model conditions
    $rconstm = $MWair;
    $rho1m = $press1m*$MWair/$rconstm/$temp1m;
    $press0m = $press1m/$y;
    $temp0m = $temp1m/$T1onT0;
    $rho0m = $press0m*$MWair/$rconstm/$temp0m;
    $a0m = sqrt($gamma*$press0m/$rho0m);
    $um = $uona0*$a0m;
    $cs = $mach*$a0m;# shock speed

    # Condition 3. drive gas
    # reservoir pressure and temperature from Skinner
    $temp3m = 1;  # arbitrarily chosen
    $a3m = sqrt($gammaHeAr * $temp3m * $rconstm / $MWHeAr8515);
    $press3m = $press0m*$y
*(1-($gammaHeAr-1)*($a0m/$a3m)*($y-1)/
  sqrt(2*$gamma)/sqrt(2*$gamma+($gamma+1)*($y-1)))
    **(-2*$gammaHeAr/($gammaHeAr-1));

    # Condition 2. post contact surface, driver gas
    $press2m = $press1m;# same as press1
    # from Liepmann & Roshko
    $temp2m = $temp3m*($press2m/$press3m)
                       **(($gammaHeAr-1)/$gammaHeAr);
    $rho2m = $press2m*$MWHeAr8515/$rconstm/$temp2m;

    $DX = $reynoldsair * $visc0m / $rho0m / $um;
    $DY = $DX * $wid / $leng;
}

sub usage
{
    print <<EOF;
usage: calcinit.pl -s | [-r] -n #
   -s => print shock speed
   -n # => output conditions file for # species (# is 1 or 2).
   -r => print real conditions
EOF
}

# figure out what to output based on command line options
if (defined $opt_s)
{
    undef $opt_s;
    print "shock speed: $cs\n";
} elsif (defined $opt_n)
{
    if ($opt_n == 1)
    {
```

```
        # output conditions file for one species
print <<"EOF";
#inputs file generated by calcinit.pl
reynold1 $reynoldsHeAr
reynold2 $reynoldsair
MAXX $MAXX
MAXY $MAXY
DX $DX
DY $DY
RHO0 $rho1m
T0 $temp1m
PRESS0 $press1m
U0 $um
RHOINT $rho0m
TINT $temp0m
EOF
    } elsif ($opt_n ==2)
    {
        # output conditions file
print <<"EOF";
#inputs file generated by calcinit.pl
reynold1 $reynoldsHeAr
reynold2 $reynoldsair
MAXX $MAXX
MAXY $MAXY
DX $DX
DY $DY
RHO0 $rho2m
T0 $temp2m
U0 $um
PRESS0 $press2m
RHOINT $rho0m
TINT $temp0m
EOF
}
else
{
    &usage;
    exit(1);
}
} else
{
    &usage;
    exit(1);
}
EOF
;
```