University of Southern Queensland

Faculty of Health, Engineering & Sciences

Hypersonic Nozzle Characterisation Using CFD

A dissertation submitted by

Alister Webb

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Mechanical)

Submitted: October, 2020

Abstract

The University of Southern Queensland Hypersonic Testing Facility (TUSQ) is used for experimentation involving supersonic and hypersonic aerodynamics. TUSQ has four available nozzles which are designed to produce different flow regimes, they are Mach 2, Mach 4.5, Mach 6, and Mach 7. The facility can produce almost constant stagnation conditions at the nozzle inlet, with stagnation pressure and temperature ranging up to 7MPa and 1100K respectively. As the most frequently used stagnation pressure is 1MPa, flows produced by this stagnation condition are consequently well documented and very well understood.

This project aims to perform CFD simulations of TUSQ using the compressible flow solver `Eilmer4` in order to improve the current understanding of the range of its capabilities. A 2D axisymmetric CFD model of TUSQ's Mach 6 nozzle and test section was created, and simulated for the nominal stagnation condition of 1MPa at 575K. The results were then compared to experimental and theoretical data documented by Birch (2019) and Buttsworth (2010). The Mach 6 nozzle simulation was validated, and found to produce results in the core flow that deviated by less than 1% compared to the experimental and theoretical data.

Further simulations were run for different stagnation conditions for both the Mach 6 nozzle, and the Mach 7 nozzle. The stagnation conditions used for both nozzles included; 4MPa at 575K, and 7MPa at 575K, with 1MPa at 900K also run for the Mach 6 nozzle. The results showed that increasing the stagnation pressure subsequently increased the static pressure in the test section, as well as the pitot pressure, and Reynolds number. It also produced a larger expansion at the nozzle exit. The 1MPa 900K simulation was found to increase the size of the usable area adjacent to the nozzle exit. The usable area has very little local variation in flow parameters and is suitable for use for experimentation.

University of Southern Queensland

Faculty of Health, Engineering & Sciences

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Dean

Faculty of Health, Engineering & Sciences

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Alister Webb

Acknowledgments

I would like to thank my supervisor Dr Fabian Zander for his help and guidance during the project. I would also like to acknowledge the support of the USQ High Performance Computing cluster (HPC) for the computational resources provided to undertake this project.

Alister Webb

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The University of Southern Queensland has a Hypersonic Testing Facility (called TUSQ) which is located at the Toowoomba campus. The facility is used for experimentation involving supersonic and hypersonic aerodynamics. The facility has three modes of operation which allow it to produce a wide range of test flows. They are: atmospheric blow-down, Ludwieg tube, and Ludwieg tube with free piston compression. Each of these modes can produce different conditions at the nozzle inlet, which allows for a wide range of possible flow conditions. There are several available flow regimes, they are Mach 2, Mach 4.5, Mach 6, and Mach 7.

For both modes of operation where the Ludwieg tube is used, a diaphragm is placed between the Ludwieg tube and the nozzle inlet. Initiation of a test flow is achieved by rupturing this diaphragm, which occurs when the pressure inside the Ludwieg tube reaches the burst pressure of the diaphragm. The burst pressure is dependent on both the material and thickness of the diaphragm, which allows for burst pressures and thus test flows to be reproduced repeatedly and accurately. When a test flow is initiated this way the stagnation conditions produced at the nozzle inlet can be almost constant, allowing for large periods of test flow with consistent parameters.

The facility is designed to utilise burst pressures of up to 7MPa. Despite this however, the most frequently used burst pressure is 1MPa. Consequently, the flows produced by the 1MPa burst pressure are well documented and very well understood. The understanding of flows produced by higher burst pressures is limited.

1.1 Project Aims

The aim of this dissertation is to investigate the properties of flows produced by higher stagnation pressures through the use of computational fluid dynamics. As there can be difficulty in measuring certain flow properties experimentally, a CFD model will be able to provide the expected flow properties with reasonable accuracy for a range of burst pressures.

The project aim can be broken down into two key areas. The scope of the project is to:

1. Determine the variation in flow parameters and the range of possible flow conditions achievable in TUSQ.
2. Determine the available core flow area for experimental testing.

1.2 Overview of the Dissertation

This dissertation is organized as follows:

Chapter 2 provides background information on wind tunnels including TUSQ, as well as compressible flow, nozzles, and CFD.

Chapter 3 discusses past works and reviews available literature concerning CFD, hypersonics and TUSQ.

Chapter 4 discusses the setup of the `Eilmer4` simulations and the reasoning for certain design decisions.

Chapter 5 details the methodology, which includes the design and revision of the CFD mesh.

Chapter 6 presents the results of the simulations. It also provides a brief discussion on the validity of the simulations based on existing experimental and theoretical data.

Chapter 7 concludes the dissertation and suggests where further work may be required.

Chapter 2

Background Information

2.1 Chapter Overview

This Chapter provides background information on core topics such as wind tunnels, compressible flow, nozzles, and CFD. It also provides information on TUSQ, including facility hardware and an overview of it's capabilities.

2.2 Wind Tunnels

Wind tunnels are used to perform experiments with fluid flows. These experiments generally involve a scale model and are often undertaken to obtain a greater understanding of the behaviour of the fluid flow and how it interacts with the model (NASA 2015*f*). Full scale testing is normally not performed due to safety issues and associated prototyping costs. As such, full scale testing is generally not feasible (though there can be exceptions). When a scale model is tested, it must be 'matched' to the fluid flow so that the obtained results are representative of the required conditions. This means that the flows are made to be kinetically, dynamically, or geometrically similar to the design criteria (NASA 2018*b*). The most common application of wind tunnel testing is the analysis of aerodynamic drag and lift for aerospace and automotive applications. Experimentation yields data that can be compared with theoretical data and calculations to assess the suitability of the tested design.

Wind tunnel testing can be used to acquire both quantitative and qualitative data about the model under test. Quantitative data may include measurements such as force and bending moment (which are used to calculate drag coefficient), as well as temperature and pressure (NASA 2015*b*). Quantitative data can be compared numerically with theoretical data. Qualitative data can also be acquired, some types include the path of flow streamlines (by using smoke or coloured gas in the flow), and eddy currents and deadzones on aircraft parts such as control surfaces or wings (by coating the surface with a thin layer of oil which the airflow can disturb) (Lewington 2005; Central Aerohydrodynamic Institute 2020). Shockwaves in supersonic flows can also be captured with Schlieren photography, whereby the different densities present in the flow refract light in different directions (Schmidt 2015; NASA 2015*d*). Though these types of data may be quantitative, with continually improving machine vision and object detection and recognition systems it is possible to analyse and reconstruct some of these types of data as computer models for further analysis (Central Aerohydrodynamic Institute 2020).

2.2.1 Types of Wind Tunnel

Wind tunnels are commonly classified by their achievable flow regimes. These regimes can be categorised as: subsonic, transonic, supersonic, and hypersonic (NASA 2015*e*). Most subsonic and transonic tunnels are fan operated open or closed return types. While most supersonic and hypersonic wind tunnels are not fan driven due to the high velocity and pressure requirements, and are instead fed from a pressure vessel. The gas from the pressure vessel is forced through a nozzle which dictates the flow regime inside the test section. The mechanics and theory behind supersonic and hypersonic nozzles are covered in Section 2.5.

Among the supersonic and hypersonic wind tunnels, several types exist including: recirculation, blowdown, and indraft. Recirculating wind tunnels operate with a fan forcing the working fluid around a closed loop. Despite generally being limited to subsonic flows, some are capable of low supersonic flows. Flow duration for this type of tunnel can be up to ten minutes or more. Another type of subsonic and supersonic wind tunnel is the indraft tunnel. This design is somewhat inefficient as it uses a vacuum to force the working fluid into the test chamber (Johl et al. 2004). This can be achieved with a vacuum vessel or a fan placed downstream of the test section. The high pressure side is gener-

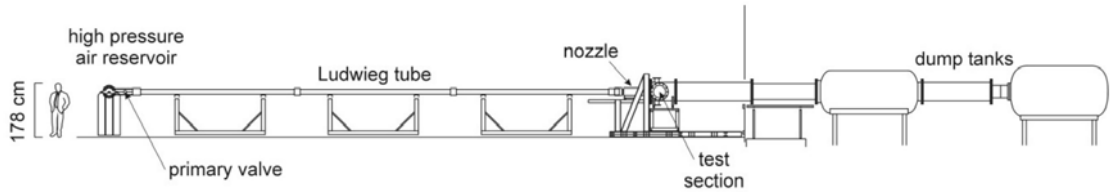
ally atmospheric pressure, while the low pressure side is a partial vacuum. While these two designs can achieve substantial flow durations, they are limited to subsonic and low supersonic flow velocities.

A blowdown tunnel functions in the opposite fashion to an indraft tunnel. The working fluid is still forced through the test section by means of a large pressure difference. However, the key difference with this design is that the low pressure side is at atmospheric pressure (or less, depending on the design) and the high pressure side is an order of magnitude (or more) higher than atmospheric pressure. The high pressure side of a blowdown tunnel is fed from one or more pressure vessels, in which the pressure can range into Megapascals (Robinson et al. 2015).

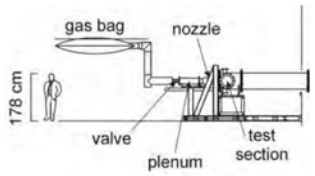
A blowdown tunnel consists of a pressure vessel, nozzle, test section, diffuser, and dump tank. The flow velocity in the test section is generally supersonic or hypersonic. Because of this, a diffuser is often used to decrease the flow velocity before it exits the system (either into a dump tank or into atmosphere) so that shocks don't form at the exit (NASA 2018*a*). Blow-down tunnels are also limited to very short test times due to the capacity of the pressure vessels. Test times are generally in the range of tens of milliseconds but some can produce flows that last as long as a hundred milliseconds or more (Buttsworth 2009; NASA 2015*a*).

2.3 TUSQ Facility

The University of Southern Queensland Hypersonic testing facility (TUSQ) is a low enthalpy Ludwieg tube with free piston compression heating, and is located at the USQ Toowoomba campus (Buttsworth 2009). Other supersonic and hypersonic testing facilities in operation in Australia include The University of Queensland's Centre for Hypersonics, and the University of New South Wales' hypersonic testing facility (of Queensland 2018; UNSW 2020). Despite being able to produce similar flow Mach numbers in some cases, these facilities produce high enthalpy, short duration flows (of less than 10ms) compared to TUSQ which is capable of producing low enthalpy, long duration flows of up to 200ms. As such, these facilities are not directly comparable.



(a) The arrangement of the facility in Ludwieg Tube mode.



(b) The arrangement of the facility in Atmospheric Blow-down mode.

Figure 2.1: Facility setup in both modes of operation. Adapted from Buttsworth (2009).

2.3.1 Hardware Overview and Operation

The facility has three modes of operation: atmospheric pressure blow-down, Ludwieg tube, and Ludwieg tube with free piston compression heating. In atmospheric pressure blow-down mode, a 1.5 m^3 gas bag is mounted to a plenum chamber on the inlet side of the nozzle (as seen in Figure 2.1b). This mode is used for low Mach number and Reynolds number flows. In Ludwieg tube mode a 16m by 130mm diameter Ludwieg tube is connected to the inlet side of the nozzle. High pressure air storage is connected to the other end of the tube via two high flow rate valves (as seen in Figure 2.1a). This storage is in the form of six standard air cylinders.

When operating with free piston compression, the Ludwieg tube and nozzle are separated by a thin Mylar diaphragm (Buttsworth 2009). When the high flow rate valves are opened, the piston (which resides at the opposite end of the tube to the diaphragm) is driven down the tube by the inflow of gas. The gas is then compressed between the piston and diaphragm. When the pressure in this section reaches the critical burst pressure of the diaphragm it ruptures, accelerating the gas into the nozzle and test section. In this mode of operation it is possible to achieve near constant stagnation conditions at the inlet of the nozzle for the test duration.

2.3.2 Capability Overview

The facility is equipped with five nozzles which each produce a specific flow velocity. They are Mach 2 (CO_2), Mach 2 (Air), Mach 4.5, Mach 6, and Mach 7. The Mach 2 CO_2 nozzle is designed for use in atmospheric blow-down mode, while the Mach 4.5 nozzle can only be used with both Ludwieg tube modes. The contoured Mach 6 nozzle (Shown in Figure 2.2) requires the use of free piston compression to achieve the required inlet stagnation conditions. The mode of operation also affects the maximum flow duration. With the Mach 2 nozzle in atmospheric blow-down, flows lasting several seconds are possible due to the (relatively) low pressures. The Mach 4.5 nozzle can produce flows lasting up to 90ms and the Mach 6 nozzle can produce flows that last up to 200ms. The test section measures 830mm in length by 600mm in diameter, and the Mach 4.5 and Mach 6 nozzles have exit diameters of 215.9mm and 217.5mm respectively. As such, experimentation involving reasonably sized models is possible.

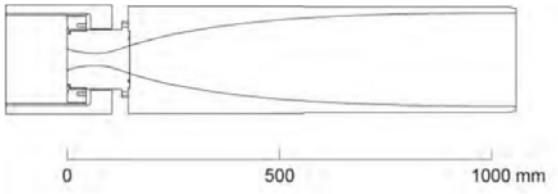


Figure 2.2: Schematic of the Mach 6. Adapted from Buttsworth (2009).

2.4 Compressible Flow

For certain engineering applications, fluid flows can be considered to be incompressible. Some examples include water in pipes, oils and lubricants, and air at low velocities. However, when dealing with very high velocity flows in fluids like air, compressibility becomes an important factor. A flow is considered compressible when a pressure gradient (or differential) can cause the density of the fluid to change. For low velocities, large pressure changes are often not apparent. However, for sufficiently high velocities interesting phenomena can occur. The dimensionless Mach number is used to characterise the flow regimes. It represents the ratio of the fluid velocity and the speed of sound of the surrounding medium, and is expressed as $M = \frac{v}{a}$ where v is the flow velocity and a is the

speed of sound (Anderson 2020).

For flow regimes close to Mach 0.3 and above (that is, 0.3 times the speed of sound), relatively large pressure changes and density changes can occur. Flow velocities less than this can be treated as incompressible, while flow velocities greater must be treated as compressible (if the effects of pressure and density gradients are to be considered). When the flow regime is between Mach 0.8 and Mach 1.2 the flow considered to be transonic (NASA 2015*e*). At Mach 1, localised pressure gradients (or shocks) form. For Mach numbers between 1.2 and 5 the flow is called supersonic, and greater than Mach 5 is called hypersonic.

It has been well noted that compressible flows without discontinuities (such as shocks) can be considered to be reversible and have constant entropy (Anderson 2020; Devenport 2001). Because of this, the properties of a supersonic flow can be described by the isentropic flow equations. These equations use the stagnation properties of the flow, which are defined as the properties of the flow if it was isentropically brought to rest. These stagnation (or sometimes called total) properties are constant. The stagnation temperature, pressure, and density are defined in relation to their actual values at some point in the flow. The isentropic relations for temperature, pressure, and density are shown in Equations 2.1, 2.2, and 2.3.

$$\frac{T_o}{T} = 1 + \frac{\gamma - 1}{2} M^2 \tag{2.1}$$

$$\frac{P_o}{P} = \left(1 + \frac{\gamma - 1}{2} M^2\right)^{\frac{\gamma}{\gamma - 1}} \tag{2.2}$$

$$\frac{\rho_o}{\rho} = \left(1 + \frac{\gamma - 1}{2} M^2\right)^{\frac{1}{\gamma - 1}} \tag{2.3}$$

Where M is the Mach number of the flow, and γ is the ratio of specific heats ($\frac{c_p}{c_v}$) for the ideal gas (Anderson 2020; NASA 2018*a*). Since these equations are based on the flow being isentropic, they do not hold when applied to points either side of a discontinuity such as a normal shock. A shock is not isentropic as it has been noted to be an irreversible process that undergoes a change in entropy. The Rankine-Hugoniot relations are used to describe the properties of points either side of a normal shock and are shown in Equations

2.4, 2.5, and 2.6.

$$\frac{T_2}{T_1} = \frac{\left(1 + M_1^2 \frac{\gamma-1}{2}\right)\left(M_1^2 \frac{2\gamma}{\gamma-1} - 1\right)}{M_1^2\left(\frac{2\gamma}{\gamma-1} + \frac{\gamma-1}{2}\right)} \tag{2.4}$$

$$\frac{P_2}{P_1} = \frac{2\gamma M_1^2 - \gamma + 1}{\gamma + 1} \tag{2.5}$$

$$\frac{\rho_2}{\rho_1} = \frac{v_2}{v_1} = \frac{M_1^2(\gamma+1)}{M_1^2(\gamma-1)+2} \tag{2.6}$$

As noted by Freedman & Greene (n.d.).

2.5 Converging-Diverging Laval Nozzle

A De Laval nozzle consists of a converging section and a diverging section. As shown in Figure 2.3 the converging section accepts the inflow from a chamber or pressure vessel, while the diverging section vents either to atmosphere or to a dump tank. For subsonic flows, the velocity in the converging section increases along its length (as it restricts the area), to a maximum at the throat. The velocity in the diverging section then decreases to a minimum at the nozzle exit. Decreasing the back-pressure (or ambient pressure) for subsonic flows increases the velocity at all points in the nozzle. When the velocity at the throat reaches Mach 1 it cannot be increased further by decreasing the back pressure as it has reached a maximum. At this point the nozzle is considered to be 'choked' (Devenport 2001). Under these conditions the mass flow rate through the nozzle is at a maximum where the pressure ratio is $\frac{P^*}{P_o} = 0.528$ (for air). The quantities denoted by '$*$' are the properties associated with Mach 1, which in the case of a nozzle resides at the throat for flows that are sonic or faster (Anderson 2020; Devenport 2001).

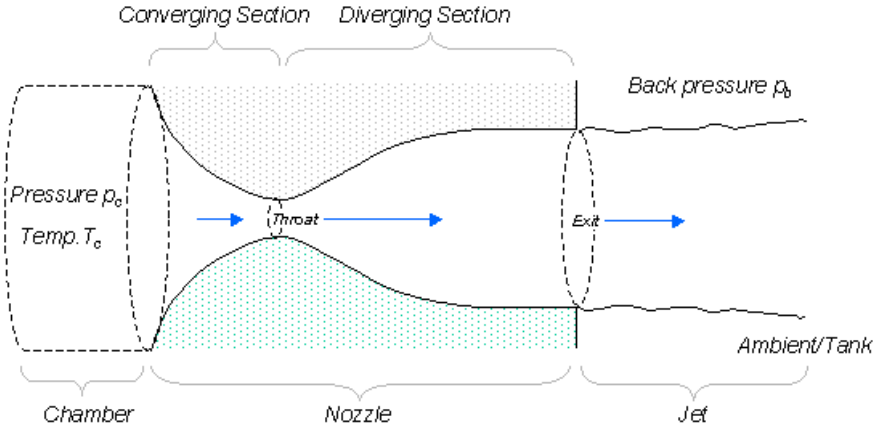


Figure 2.3: Sections of a converging-diverging nozzle. Adapted from Devenport (2001).

By decreasing the back-pressure from a choked state, the supersonic flow in the diverging section of the nozzle will accelerate with an increase in area (the opposite to subsonic flow). Once supersonic conditions have been attained at any point in the diverging section a normal shock will terminate the flow. Altering the back-pressure in this instance will shift the location of the normal shock along the length of the nozzle. In the case where a normal shock resides inside the diverging section of the nozzle, the flow is instantaneously slowed to subsonic speeds as it crosses the shock and ejected in the subsonic core flow/jet. Decreasing the back-pressure moves the shock towards the exit until a critical back-pressure is reached. At this pressure the shock transitions from terminating the flow at the exit plane of the nozzle, into an oblique shock outside the nozzle. The geometry of this shock is dependent on the back-pressure.

As seen in Figure 2.4, for the ratio of nozzle pressure to stagnation pressure at a given location, there are many different phenomena that can occur. Line a shows a completely subsonic flow, where the velocity (and pressure) are at a maximum at the throat. Line b shows a choked flow, where the pressure ratio never decreases to less than 0.528. As Devenport (2001) suggested, this is the critical ratio for air and cannot occur until after the throat of the nozzle. Line c represents a supersonic flow where a normal shock resides inside the diverging portion - as denoted by the sudden increase in pressure ratio. Similar to line c, line d indicates that there is a normal shock present at the exit plane of the nozzle. It is at this critical pressure ratio that the shock can transition from the exit plane to outside the nozzle. Lines e, f, and g are of special interest as the shock (or expansion waves) in these cases reside outside the nozzle.

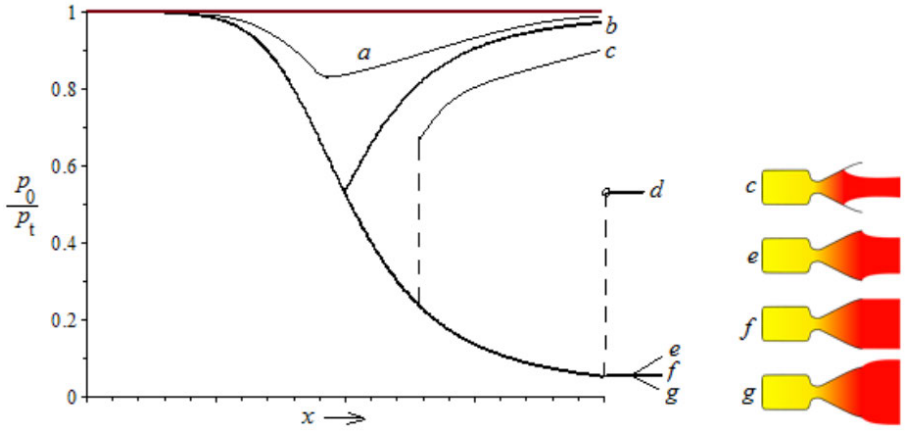


Figure 2.4: Ratio of exit pressure to stagnation pressure for location along the length of a nozzle. Behaviour of the flow changes as the back-pressure is altered. Adapted from Martinez (2020).

The nozzle flow shown by e is called over-expanded, as the oblique shocks formed at the exit converge on the core flow. In this case the pressure in the diverging section of the nozzle decreases to less than the ambient pressure, and the gas in this section 'over-expanded'. An oblique shock forms at the exit to compress the flow as it meets the higher back-pressure. Line f is considered to be 'adapted' as the pressure in the nozzle reduces to the back-pressure at the exit, here there is no (or at least a very small) pressure difference. Line g represents an under-expanded nozzle, where (opposite to e) the pressure in the diverging section of the nozzle is greater than the ambient pressure causing the flow to expand as it exits the nozzle.

Over-expanded flows generally occur during low altitude testing of rocket engines. At operating altitude the nozzles are designed to be adapted, but at lower altitude (sea level for example) the ambient pressure is higher than the design pressure. Both under-expanded and over-expanded flows can generate 'shock diamonds', where the sudden expansion (or compression) of the flow generates a series of expansion and compression fans.

2.6 Fundamentals of CFD

Computational Fluid Dynamics (CFD) is the branch of fluid mechanics concerned with computer simulation of fluids. The capability of most modern CFD packages includes static and dynamic fluid flows, and even thermodynamic simulations involving heat transfer (Chung 2002). It is often used in conjunction with experimental wind tunnel tests as a way of obtaining certain flow parameters that would otherwise be immeasurable such as entropy, enthalpy, and internal energy. It is also frequently used as a standalone method of determining the suitability of engineering designs where an experiment is not feasible.

2.6.1 Governing Equations

The three fundamental equations in fluid mechanics are: the energy equation, momentum equation(s), and continuity equation (NASA 2015*c*). Called the Navier-Stokes equations (NS Equations) after the physicists that created them, they are the basis for (almost) all CFD problems. These equations have no known analytical solution for most problems, but do have well defined solutions for some very specific and simplified problems. It is for

this reason that there are four variations of the NS Equations. In order of decreasing complexity they are: Navier-Stokes Equations, Euler equations, the full Potential Equations, and the linearised potential equations (Jameson 1990; NASA 2015*c*).

Another variation of the NS Equations is called the Reynolds-Averaged Navier-Stokes Equations (abbreviated to RANS Equations). The idea for this variation of the NS Equations was proposed by Osborne Reynolds, and involves using what is called the Reynolds Decomposition to time average the equations. The Reynolds Decomposition is a technique where a value is separated into its average and fluctuating values (Nguyen 2005). The RANS Equations are used to simulate turbulent flows, as Direct Numerical Simulation (DNS) of the NS Equations is only possible for very small or geometrically simple domains (or as mentioned before in very special or simplified cases).

Simplifying the NS equations by removing the viscous terms, produces the Euler equations. Simplifying again, by removing the vorticity terms yields the full potential equations. Linearising these equations then produces the linearised potential equations (Jameson 1990). In early CFD problems, most of the methods involved idealisations and simplifications. The linearised potential equations were adopted as the first governing equations for CFD in the 1940's, though the equations had been derived much earlier. The advancement in computational power in subsequent decades saw the adoption of the full potential equations, Euler equations, and finally the Navier-Stokes equations. Both the Euler equations and Navier-Stokes equations are used in modern CFD as the governing equations for inviscid and viscous models respectively (Chung 2002). Both equations have a variation for compressible and incompressible flow. Because of the nature of these equations, there is normally no analytical solution for complex simulations (though there are exceptional cases). For most CFD problems these equations must be solved numerically.

2.6.2 Meshing

Meshing is the process of dividing the flow domain into smaller discrete parts. Depending on the number of simulation dimensions these parts can be lines (for simple 1 dimensional analysis), areas (for 2 dimensional domains) or volumes (for 3 dimensional domains). Starting with the simplest domain - a line of length L between two points. The line can be subdivided into discrete sections of length $\frac{L}{n}$ where n is the number of sections (or number of nodes minus 1). For some governing equation(s) applied to this domain, the

value(s) can be calculated at each of these nodes. The same basic principle applies to 2 and 3 dimensions - although it is more complicated. The surface or volume is divided into smaller areas or volumes, with properties calculated at each.

This process may be trivial for small or simple geometries, but for large and/or intricate geometries it is very complicated. Flow domains are rarely a uniform shape, and more often comprised of compound curves, Bezier curves, and arcs which prevent consistent element shapes or sizes. In these cases it is very easy for the generated elements to be too large or small, have large aspect ratios, and/or be extremely skewed. Selection of the appropriate size and shape of element (as well as distributing them appropriately) is critical in producing reliable, high accuracy results. There are several mesh parameters that can affect the accuracy of solutions, the most important being: cell aspect ratio, cell skewness, and element size.

A cell's aspect ratio is the ratio of the length of its longest and shortest sides. Bakker (2006) noted that both the compute time of a solution, and the solution accuracy are dependent on the size of the cells that the mesh is comprised of. In general, smaller element or cell sizes produce more accurate solution at the cost of computation time. An improvement in accuracy does not always occur for *very* small elements in *all* cases. When the boundary layer is of interest, the size of elements very close to the wall can be very small to capture the behaviour of the fluid in this region. Maintaining an aspect ratio of close to 1 in this case would not be feasible as the width of the elements would also be very small, leading to unreasonably large element counts.

Chan et al. (2011) mentioned that there is some uncertainty surrounding the appropriate aspect ratio of cells in CFD simulations. In some cases involving viscous boundary layers, element aspect ratios of up to 1000 are not unheard of. In cases such as this, the elements would be aligned lengthwise with the wall, such that the longest dimension is parallel to it. Large aspect ratios would allow for sufficient resolution perpendicular to the wall to capture viscous effects, without increasing compute time significantly. Chan et al. (2011) went on to note that even though aspect ratios may reach up to 1000, the upper limit was roughly 600 before their simulation results became erroneous. Bakker (2006) also noted that for cells in the bulk of the flow domain (excluding the boundary layer due to the restrictions already mentioned) aspect ratios of close to 1 are ideal.

Cell skewness is determined by the optimal cell area (for triangles) or based on the relative

difference of a right angle and the smallest or largest internal angle (for quadrilaterals) (Bakker 2006). A skewness value of (or close to) 1 represents an element that is very skewed and often appears as a sliver or line as opposed to a triangle or quadrilateral. A skewness near 0 represents an element that is very close to triangular or rectangular. According to Bakker (2006) very high skewness values should be avoided as they can cause inaccurate results. To limit introduced error skewness for triangles should be limited to no more than 0.85, and for rectangles to no more than 0.9.

Also of important note is the distribution of cells in a given direction. Sometimes an area of smaller elements may be required at a point of interest. It would be inadvisable to decrease the global element size for a localised increase in resolution/accuracy as this would also increase the compute time considerably. In these cases it would be more appropriate to redistribute the cells so that they decrease in size the closer they are to the area of interest. This is called clustering (or inflation). In a default structured mesh the number of elements are defined along the edges of the domain (either by specifying a number or the maximum cell size), and are generated so that they are all equally spaced. Clustering is a way of changing this spacing so that the bulk of the cells are distributed closer to one side or edge. A direct result of this is that the cell size may change gradually along an edge. This change in cell size is important as having small cells next to large cells can introduce error. Bakker (2006) stated that the size of two adjacent cells should not differ by more than 20%. This equates to a maximum inflation factor or growth rate of 1.2.

2.6.3 Post-processing and Data Visualisation

Once a CFD simulation is complete, all of the cell data in the domain is exported. Analysing the data allows useful information to be extracted from it and interpreted by the user. Though it is dependent on the type of analysis (and purpose of the CFD simulation), the extracted information can take different forms. In some cases a single number or answer may be sufficient. In others, data at points of interest or values summed over an area may be required. For instances where a detailed analysis is required, the simulation data can be processed after the simulation is complete. This can be performed by the CFD program by using an integral utility or function, or by a third party program.

Programs such as MATLAB can be used to process very large datasets. This processing

can be done to fit curves or trend lines and perform large scale validation of results with experimental data. While numerical values can serve to be very useful in calculation they can sometimes be unintuitive when there is limited context. Simulation results can also be processed by visualisation software such as ParaView or GNUPlot (MATLAB also has the capability to produce visualisations of data, however the data often requires pre-processing with ParaView or similar for this to be achieved). Because simulation results often include fields such as coordinate system data, geometry data, and mesh data, visualisation programs can transform the simulation data into easy to interpret forms such as a graph or plot (Kitware n.d.). These visualisations can be very useful for quick visual reference or to convey a large amount of information in a compact form. Some types of data visualisation are however, unable to be easily manipulated to allow for precise results, as colour gradients and lines can be difficult to interpret with repeated accuracy.

Some post-processing programs allow the user to 'probe' points in the flow domain, which can be used to determine the flow parameters in a given location. This can be a very useful feature, however it can be time-consuming to manually select a large number of points which in turn can limit the feature's usefulness. For purposes such as this, many data visualisation packages have the ability to export the data to a different file format which can be read by other data analysis programs.

Chapter 3

Literature Review

The focus of this literature review will be Computational Fluid Dynamics applied in the context of hypersonics. It will also briefly focus on Ludwieg Tubes and shock tunnels, as well as previously conducted experiments and literature relating to TUSQ.

3.1 Hypersonic CFD

Hypersonic CFD is an increasingly important area of study and development over the last 30 years. With the advent of supersonic aircraft and spaceflight, the ability to model and analyse the behaviour of supersonic and hypersonic flows has become a very important advancement. Despite possessing the ability to produce a wide array of test conditions, Supersonic and Hypersonic ground testing has technical limitations. Technical limitations such as instrumentation measurement speed (and resolution), as well as power and performance requirements exist which can restrict potential test conditions (Anderson 2020). As such, CFD is often used in conjunction with ground testing facilities as a way to help improve the understanding of flows produced in these facilities or as a way to accurately model flows that may otherwise be unable to be produced due to technical limitations.

3.1.1 CFD simulation of Hypersonic Nozzles

Numerical simulation of hypersonic nozzles can be used to provide an in-depth spatial and temporal analysis of flow parameters and can be instrumental in improving the under-

standing of the produced flows. Sura (2017) used CFD to analyse the flow field of a CD nozzle and determine the size of the core flow and the effective usable test volume. They found that due to running the simulation as inviscid and with 'ideal' boundary conditions the nozzle was over-expanded. The boundary conditions used were not realistic, with an outlet pressure specified at 0Pa. As evidenced by their results, the boundary condition had a significant effect on the nozzle flow. Improper boundary condition definitions can lead to unexpected (and sometimes unphysical) flow characteristics as well as erroneous results. As such, it is essential that boundary conditions be appropriately selected and defined.

Jéger & Veress (2017) performed and documented axisymmetric simulations of a rocket engine. They investigated the difference between a conical nozzle and a bell nozzle and found that the bell nozzle produced more thrust. They also provided details on their methodology, which included labelling of boundary conditions, comparison to experimental data, and a mesh and geometry independence study. Mesh and geometry (or domain) independence studies are crucial in ensuring that the solution is both accurate and closely representative of real world conditions. The geometry independence study performed by Jéger & Veress (2017) was done to ensure that the flow domain past the nozzle exit did not affect the flow upstream. Their simulation involved nozzle discharge to ambient air, so if the domain was incorrectly sized (i.e. if it was too small) the discharge could affect the flow upstream of the nozzle regardless of the boundary conditions imposed.

They found that the domain geometry did not cause any unwanted effects. A mesh independence study was also performed and involved comparing the results of simulations for each nozzle at cell counts of 300,000, 600,000, and 900,000. However they did not mention which cell count produced mesh independent results.

3.1.2 Viscous Effects and Turbulence Models

There are several ways to accurately capture the effects of viscosity and turbulence in a simulation. One option is to spatially resolve the boundary layer with a suitable number of cells. A study conducted by Bono et al. (2008) into Mach 10.3 flow over a wedge on a flat plate showed that cell sizes of less than 0.5mm were not unreasonable for their simulations. Cell sizes in this order of magnitude can result in simulations that are very accurate, and have very high fidelity results. The solve time however, is often considerably

longer than that of a simulation that utilises larger cells. Because increasing the number of cells also increases the computational load accordingly, methods have been devised to both retain solution accuracy and decrease solve time.

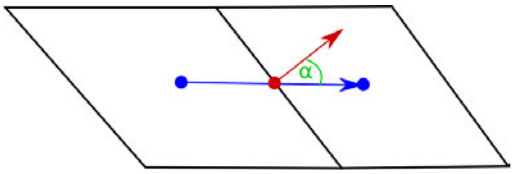
Turbulence models were created to model the effects of boundary layers and turbulence without significantly increasing compute time or requiring such a small cell resolution in areas of interest. A turbulence model uses equations to approximate the behaviour of the fluid in defined regions close to the boundary. These equations are called wall functions and are dependent on the proximity of the fluid element to the wall or boundary. Because wall functions are designed for specific regions in the boundary layer, the size of the elements close to the wall can often be critical. The y^+ value is a non-dimensional distance of the first cell to the wall and is based on the fluid velocity and cell size. Each turbulence model uses different wall functions, and so a certain cell size (and corresponding y^+) may not be suitable for all models.

Wandel (2020) said that the boundary layer can be split into sections defined by their y^+ values. The following regions make up the boundary layer: linear sublayer ($y^+ < 5$), buffer layer ($5 < y^+ < 30$), log-law layer ($30 < y^+ < 500$), and the outer layer ($y^+ > 500$). Each turbulence model is accurate in a certain range of y^+. If the first cell size is outside this range, the solution for these cells may be erroneous. The k-ω turbulence model for example is accurate for a first element y^+ in the range of $1 < y^+ < 300$. Chan et al. (2011) analysed the suitability of the k-ω turbulence model for supersonic and hypersonic flows. They found that despite the model's inherent sensitivity to free stream turbulence, it can be used to model turbulence for supersonic and hypersonic flows. They also noted that for some of their cases, the cell aspect ratio at the boundary was required to be no more than 600, and that a wall y^+ of less than 1 was required to minimise error.

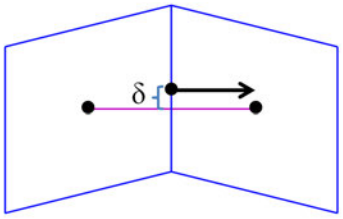
3.1.3 Mesh Quality

Depending on the scope and required accuracy of the simulation, the quality of the final mesh can be crucial. Some solvers are based on different assumptions, and so the overall mesh quality requirement may differ slightly between them. For example, `Eilmer4` uses the cell-centred finite volume method which operates based on the assumption that the cells in its structured grids are orthogonal (Jacobs & Gollan 2018). This discretisation method computes the fluxes at the centres of the cell boundaries and if the direction of the

fluxes are sufficiently different, error can be introduced and the solution can fail or produce erroneous results. In this case the critical mesh quality parameter is orthogonality, which is defined as the difference in angle between the line connecting two cell centres and the normal to the cell boundary between them and is illustrated in Figure 3.1. In other words, if the cells are sufficiently deformed that a line perpendicular to the cell interface does not cross through (or pass sufficiently close to) both cell centres, there is a possibility of introducing error.



(a) Poor cell orthogonality. Adapted from CFD Support (n.d.).



(b) Good cell orthogonality. Adapted from Rhoads (2014).

Figure 3.1: Cell orthogonal quality. Poor quality is denoted by a high boundary normal to cell centre angle.

3.1.4 Mesh Independence

Solution accuracy is normally dependent on cell size. In general, meshes with larger cells tend to produce results that are less representative of real world conditions. Decreasing the cell size can help to improve the overall solution accuracy at the expense of solve time. There is a point at which decreasing the cell size produces minimal improvements in accuracy. As such it is important to determine the smallest practical cell size that produces results with the required accuracy.

Mousavi & Roohi (2014) performed 3D CFD analyses on a square convergent-divergent nozzle. The aim of their analyses was to determine the location and number of shocks that formed in the nozzle. To ensure a grid independent result, they performed a mesh independence study by running simulations for 6 different mesh refinements. By recording the Mach number, centreline pressure and Reynolds number and plotting them against the number of cells, they determined that a minimum cell count of approximately 272×10^4 was required. Increased cell counts were found to produce diminishing returns for simulation accuracy. Their findings were more detailed compared to those of Jéger & Veress (2017)

mentioned above as the final mesh resolution was noted.

Narayana & Reddy (2016) provided a much more detailed process regarding mesh refinement. During their analysis of a CD rocket nozzle they tested five separate meshes, each with different combinations of horizontal and vertical cell counts. The boundary conditions and simulation settings were also mentioned in detail. The simulation results for each mesh were then compared for Mach number, Prandtl number, dynamic pressure, static pressure, Reynolds number, Equi-angle Skew and orthogonal quality with contour plots shown for each. Limited analysis and discussion of results was presented in their article, and the final mesh resolution was not noted. The simulation settings and geometry were presented in adequate detail that reproduction of their results would be possible. Plots shown in their report indicated that the refinement and cell count affected the nozzle flow. Larger cell sizes were shown to produce results that did not have shocks formed in the nozzle throat. This indicated that the more coarse mesh did not sufficiently resolve the flow domain. As the size of the cells was refined, the location and behaviour of the shocks became more apparent and well defined. The mesh refinement study showed that ensuring that the solution to a simulation is independent of the mesh is crucial for hypersonic CFD.

3.2 `Eilmer` Compressible flow Solver

`Eilmer` is a CFD program designed to solve the compressible NS Equations for both 2D and 3D transient simulations utilising the finite volume method. The basis of `Eilmer` is the Compressible Navier Stokes solver (abbreviated as cns4u) (Jacobs et al. 2012). This code originally served to simulate single-block problems such as shock and expansion tubes, and evolved into the Multiple-Block Compressible Navier Stokes solver (MBCNS2). From here development at the University of Queensland saw the creation of Elmer (and its renaming to `Eilmer`). Validation of `Eilmer` has been performed by Jacobs & Gollan (2013). Their process involved simulating supersonic flow interacting with a sphere in a range of gases. The results of their simulations were in agreement with experimental data from three separate studies (Jacobs & Gollan 2013).

The program has a number of simulation capabilities which include (but may not be limited to), 2D and 3D compressible flow, inviscid, laminar, and turbulent flows, and finite

rate chemistry (Jacobs & Gollan 2020*a*). Both the 2D and 3D flow simulations utilise the NS Equations in their integral form, with the main difference being that the 2D equations omit some components. `Eilmer` uses the cell-centred finite volume method, whereby the fluxes are computed at the midpoints of the cell boundaries (Jacobs et al. 2019). For 3D simulations these cells (volumes) are hexahedral and can have planar or non-planar surfaces. 2D cells are quadrilateral and are composed of only straight lines. The boundaries of a 2D cell are labelled as North, East, South, and West. The naming convention for blocks (groups of cells) is identical (Jacobs et al. 2019). `Eilmer` does not have a Graphical User Interface (GUI) and instead simulations are defined using an input script. These scripts are written in the Lua programming language and contain definitions for the geometry, mesh, flow states, and configuration variables as well as other simulation information (Jacobs & Gollan 2020*c*).

3.3 Shock Tunnels and Ludwieg Tubes

A shock tunnel is a type of wind tunnel designed to generate supersonic and hypersonic flow regimes. It consists of five discrete sections: a high pressure gas reservoir, compression tube, shock tube, nozzle, and test section and dump tank(s) (shown in Figure 3.2). The shock tube is separated from the compression tube and nozzle by a diaphragm at each end. The apparatus operates by using the high pressure gas reservoir to drive a piston along the compression tube. Once the piston compresses the driver gas to the burst pressure of the first diaphragm, it ruptures and a shock wave propagates along the shock tube. The shock wave is then reflected off of the secondary diaphragm allowing the test gas in the shock tube to stagnate. The secondary diaphragm then ruptures, allowing the test gas to flow through the nozzle and into the test section (The University of Queensland 2016; Robinson et al. 2015).

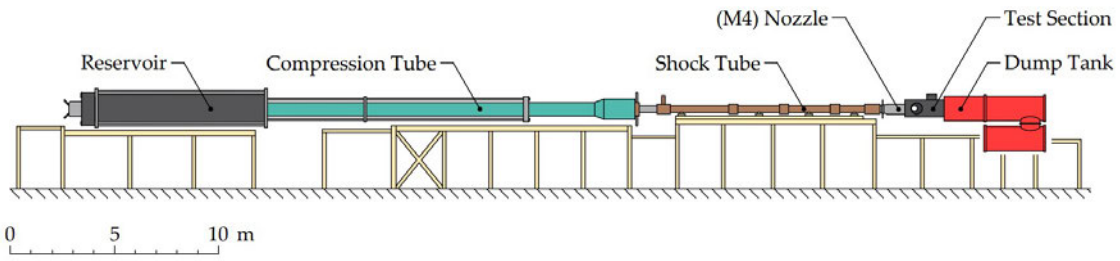


Figure 3.2: The layout of the University of Queensland T4 - Free-piston driven shock tunnel. Adapted from Doherty (2014).

Shock tunnels are used to generate flow regimes of between Mach 4 and Mach 10 (with some even reaching Mach 25 or higher), and can produce flows that last up to 10 milliseconds or more (The University of Queensland 2016). The stagnation pressures used to produce these flows can range up to 44MPa or more resulting in high enthalpy and high temperature flows (Robinson et al. 2015).

A Ludwieg tube operates in a similar fashion to a shock tube, with the main difference being that a Ludwieg tube uses only one diaphragm which separates the driver tube and nozzle. A Ludwieg tube consists of four discrete components: a high pressure gas reservoir, compression/driver tube, nozzle, and test section and dump tank(s). To operate the Ludwieg tube, the high pressure gas reservoir is vented into an evacuated driver tube. A shock wave propagates along the tube and is reflected off of the diaphragm allowing the gas to stagnate. The pressure increases and eventually causes the diaphragm to burst, allowing the gas to flow through then nozzle and into the test section.

Similar to a shock tube, a Ludwieg tube can produce supersonic and hypersonic flow regimes. However, due to the smaller compression ratio (from the reservoir to the nozzle) the produced flows have a lower enthalpy and lower temperature than those of a shock tube. The flow duration can also last up to 200ms or more (Buttsworth & Smart 2010; Buttsworth 2009). Ludwieg tubes can also be operated with 'free piston compression' where the reservoir gas drives a piston along the Ludweig (driver) tube. The piston compresses the driver gas which then ruptures the diaphragm. Free piston compression is used to produce higher stagnation temperatures and thus hotter test flows.

3.4 TUSQ

Previous work has been done involving hypersonic flows in the TUSQ facility. Buttsworth & Smart (2010) performed experiments involving scramjet inlet starting. The TUSQ facility was operated as a Ludwieg tube with free piston compression and utilised the Mach 6 nozzle. During their experiments, the high pressure air reservoir was initially pressurised to 4MPa, which produced a Mach 6 flow which lasted 217ms. The flow stagnation pressure (at the inlet of the nozzle) was measured to be 840kPa and the initial fluid pressure inside the nozzle and test section was measured to be 94.4kPa. They determined that if the Ludwieg tunnel was assumed to be perfectly isentropic that the fluid temperature at the inlet to the nozzle would be 570K once it was compressed by the piston. Actual experimental measurement proved that this was not the case with a measured temperature of 440K. They commented that the Ludwieg tunnel was not operating such a way that the flows could be considered isentropic, and that the performance of the facility was to be determined with temperature measurements and not isentropic relations.

Buttsworth (2009) provided an overview of the tunnel as well as the expected achievable flow conditions. The overview consisted of an explanation of the operating principles, dimensions and important measurements of the facility, the available nozzles, and both experimental and target flow parameters in its various modes of operation. They noted that the achievable stagnation conditions for the operational modes were 70kPa for the blowdown, and 7MPa for Ludwieg tunnel and free piston modes. The nominal stagnation temperature was also listed for Ludwieg and free piston modes as 283K and 500K respectively.

Birch et al. (n.d.) investigated pressure fluctuations in the wind tunnel due to free piston compression. They noted that the nominal flow conditions for a stagnation pressure of 1MPa included a Mach number of 5.9, pitot pressure of 32.9kPa, and a free stream flow velocity of 1013m/s.

The parameters listed by Buttsworth & Smart (2010), Buttsworth (2009), and Birch et al. (n.d.) are somewhat similar to one another. An important point to note is that most (if not all) of these parameters are theoretical conditions. The conditions presented by Birch et al. (n.d.) are all theoretical and based on the assumption of isentropic flow. Similarly, the conditions presented by Buttsworth (2009) are all achievable or nominal conditions.

Birch (2019) also noted the nominal conditions of the Mach 6 nozzle, with most of the parameters being theoretical and based on the assumption of isentropic flow. They did however, present some measured experimental data, of note were their time averaged plots of pitot pressure and Mach number.

3.5 Chapter Summary

A literature review was conducted on hypersonic CFD. It was found that there were several crucial factors in creating a CFD simulation of hypersonic flows with the CFD mesh being the most important. Several sources found that certain cell sizing and boundary condition assumptions lead to erroneous results. They also highlighted the importance of conducting a mesh refinement study to ensure that the results were independent of the CFD mesh.

Information regarding TUSQ and simulation of hypersonic nozzles was also reviewed. There are a number of existing studies detailing the TUSQ free stream, however these studies focus on specialised areas. It was determined that there was a gap in the current knowledge surrounding characterisation of the TUSQ free stream, in particular, the general flow characteristics in the test section.

Chapter 4

Simulation Setup

4.1 Chapter Overview

This chapter details the setup requirements for a simulation using the compressible flow solver `Eilmer4` and also includes a brief overview of the structure of an input script.

4.2 Script Structure

Because many of the components of a simulation input script are dependent on other stages, the order in which certain elements are defined is important. Each script must follow a similar structure. In general the structure is:

1. Set configuration settings such as; title, dimensions, viscosity and turbulence, etc.
2. Create (or import) the geometry.
3. Using the geometry, create quadrilaterals that comprise the flow domain.
4. Create grids by specifying the number of divisions along each axis of the previously defined quadrilaterals.
5. Create fluid blocks (or fluid block arrays, see Section 4.6) and define the boundary conditions.

Depending on the complexity of the simulation the structure of the script my change. For example, it is possible to omit parts 2 to 4 by instead importing an existing grid.

4.3 Configuration Settings

`Eilmer4` has a wide range of available configuration settings. The settings that are of interest for the TUSQ nozzle simulations are shown below, with the appropriate lines of code also listed with each:

- Simulation dimensions can be to set to either 2D or 3D as `Eilmer4` has capabilities for both. A second component to a 2D simulation is the axisymmetric setting. This serves to change the simulation so that the 2D geometry represents a slice of a revolved shape and sets the x axis as the axis of revolution (Jacobs & Gollan 2020*c*). Doing so can reduce the computation time and complexity for revolved shapes.

  ```
  config.dimensions = 2
  config.axisymmetric = true
  ```

- Viscosity and turbulence can be included. The only available turbulence model is the Wilcox 2006 k-ω turbulence model.

  ```
  config.viscous = true
  config.turbulence_model = "k_omega"
  ```

- The solver will terminate at the maximum simulation time, with the maximum number of steps being the limit to the number of iterations that the solver will perform. In general the number of steps is set very high so that the solver does not terminate before the maximum simulation time is reached. The initial dt (Δt) can be set to ensure a stable start to a simulation as in some cases the automatically calculated time step can be too large and cause the simulation to fail. This time step is dynamically calculated using the CFL (Courant–Friedrichs–Lewy) value during each iteration. Smaller values may be required for simulations that have very sudden flow field changes (Jacobs & Gollan 2020*b*).

  ```
  config.max_time = 10.0e-3
  config.max_step = 12000000
  config.dt_init = 1.0e-9
  ```

```
config.cfl_value = 0.5
config.dt_plot = 1.0e-3
config.dt_history = 10.0e-5
```

- The whole flow solution will be written to file after each increase of the plot increment. This is normally set to 1ms so that results may be recorded for each millisecond of flow time.

```
config.dt_plot = 1.0e-3
```

4.4 Flow Domain and Geometry

The flow domain consists of three sections: the inlet side of the nozzle (which also includes the end of the Ludwieg tube), the nozzle, and the test section. As mentioned above, there were two possible methods of creating the flow domain by using `Eilmer4`'s geometry package. A 3D model can allow for calculation of properties across the entire flow domain, but can be very complex to create using the geometry package and can require a significant number of cells to attain a suitable cell resolution. 3D solutions can also be very computationally intensive and often result in very long solve times.

2D simulations can be much less resource intensive and do not require such a large number of cells. They are however, limited to certain usage cases like geometry that exhibits an axis of revolved symmetry. For axisymmetric cases the 2D flow domain is limited to a 'slice' of the original 3D geometry. In other words, if the 2D geometry was revolved around its axis of symmetry, the resulting solid would be the same as the original 3D geometry. As solve times are of significance, the 2D axisymmetric simulations were conducted. Figure 4.1 shows the TUSQ Mach 6 flow domain represented as a 2D axisymmetric 'slice' of the 3D cavity.

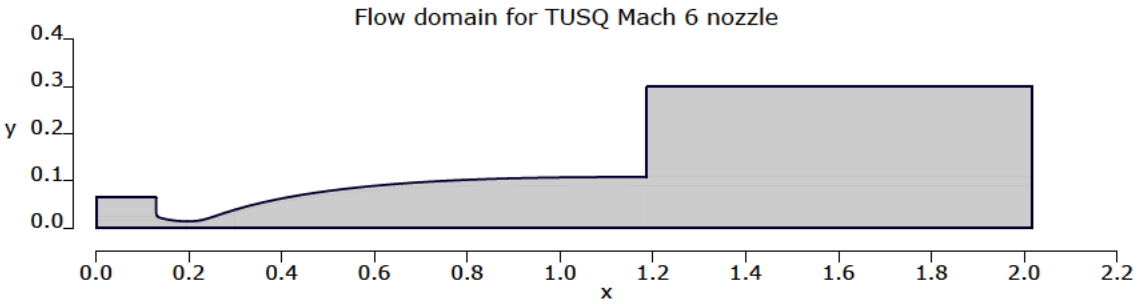


Figure 4.1: Flow domain of the Mach 6 nozzle and test section.

4.5 Boundary Conditions

The boundary conditions for each simulation were specified as shown in Table 4.1. For the Mach 6 nozzle an additional simulation was run using $P = 1\text{MPa}$ and $T = 900\text{K}$ at the inlet.

Table 4.1: Boundary conditions for each stagnation condition.

	Inlet	**Outlet**	**Axis**	**Walls**	**Initial**
1MPa	$P = 1\text{MPa}$ $T = 575\text{K}$	Specified back-pressure $P_\infty = 500$ Pa	Symmetry	Non-slip, Adiabatic	$P = 500$ Pa $T = 320$ K $v_x = 300$ m/s
4MPa	$P = 4\text{MPa}$ $T = 575\text{K}$	Specified back-pressure $P_\infty = 500$ Pa	Symmetry	Non-slip, Adiabatic	$P = 500$ Pa $T = 320$ K $v_x = 300$ m/s
7MPa	$P = 7\text{MPa}$ $T = 575\text{K}$	Specified back-pressure $P_\infty = 500$ Pa	Symmetry	Non-slip, Adiabatic	$P = 500$ Pa $T = 320$ K $v_x = 300$ m/s

Inlet

There are a range of possible inflow boundary conditions available in `Eilmer4`. As the Ludwieg tunnel produces nearly constant stagnation conditions at the nozzle inlet, this boundary condition was set to inflow from stagnation. This was implemented in the input script with `InFlowBC_FromStagnation:new{stagnationState}`. Where the stagnation state was defined with respect to the stagnation pressure and temperature, as well as the turbulent kinetic energy and specific dissipation. This boundary condition operates on the assumption of isentropic conditions at the boundary.

Walls

The wall boundary condition was set as 'no-slip' and with a fixed temperature (as the heat transfer to/from the wall would be negligible). Coupled with the viscous configuration setting, no-slip wall condition allows the viscous boundary layer close to the wall to be captured (cell size notwithstanding). This is of interest as the boundary layer affects the effective core flow exit diameter, and thus the nozzle area ratio. A larger boundary layer decreases the area ratio, $\epsilon = \frac{A_*}{A_e}$, which thus affects the core flow velocity. An

inviscid (and laminar) simulation with a 'slip wall' boundary condition would not produce realistic results close to the wall due to the non-existent boundary layer. Wall functions were also enabled for the turbulence model to reduce both computation time and cell size requirements close to the wall. This was only true for later stages of the simulations that had sufficient spatial resolution close to the wall.

Axis

As the lower edge of the geometry lies on the x axis, it is automatically set as the axis of symmetry by `Eilmer` due to the axisymmetric configuration setting. The solver computes the mass flux across the axis in this case, and if it is not sufficiently small terminates the simulation.

Outlet

The test section and dump tanks are under a partial vacuum during a run. By specifying the back-pressure at the outlet boundary, the partial vacuum can be accounted for during a simulation. This is boundary condition is only suitable for short simulation times, as the back-pressure changes over time due to the mass accumulation in the dump tanks. 'Short' in this case means 60ms or less, as at that point the back-pressure in the test section begins to change. The boundary condition is set with `OutFlowBC_FixedP:new{p}`, where `p` is the back-pressure in Pa.

Initial Domain Properties

The initial conditions are set as $P = 500$Pa, and $T = 320$K for the first run of each simulation. $v_x = 300$m/s was set as the flow in the simulations incorrectly initiated with $v_x = 0$m/s. Unphysical areas of recirculation occurred in the nozzle inlet for $v_x = 0$m/s, so specifying a non-zero velocity was done to mitigate against this. For multi-stage runs, the initial conditions are set to the resultant conditions of the previous run (See Section 4.7).

4.6 Blocks

Fluid blocks are a grouping of cells and are defined in the input script by specifying a grid and any applicable boundary conditions. Grids are defined by a reference quadrilateral and row and column indices for the number of cells. The purpose of a block is to divide the flow domain into discrete areas (or volumes for 3D simulations). For a 2D simulation, each block can have a maximum of four sides (i.e it must be a quadrilateral) and is used

to produce a grid (or mesh). The properties of the mesh can be controlled on a per-block basis, and as such allow for quite complex flow domains and meshes to be created.

4.6.1 Fluid Block Arrays

A fluid block array is a variation of the standard block that further divides the flow domain into smaller 'sub-blocks'. The number of sub-blocks are defined by indices for x and y divisions. For jobs run on conventional consumer grade computers, fluid block arrays provide little additional benefit. Jobs run on clusters (such as the USQ HPC) have the ability to assign multiple processor cores to a block, and thus more cores to a job/simulation. Using multiple processors can drastically reduce compute time. For instances where numerous processor cores are assigned to the same job (i.e. it utilises Parallel computing), it requires that the computational load is balanced across all the processor cores in use. To do so, the following line of code is required at the end of the input script `mpiTasks = mpiDistributeBlocks{ntasks=` $nTasks$ `, dist="load-balance"}`.

4.7 Multi-Stage Simulations

Singular, high resolution simulations can be very computationally intensive. A way of reducing the overall compute time is to run a series of simulations with progressively finer meshes. Each subsequent job will utilise the solution data from the previous job as it's initial state. In other words, each simulation 'picks up' where the last one finished. Doing so can reduce the compute time even further as the simulation is already initialised with the expected (or at least estimated) flow field. This was implemented for all simulations in the scope of this project and was defined in the input script by creating a flow solution object '`FlowSolution:new{}`' based on the previous run. This was then used to create a flow state with the '`makeFlowStateFn()`' function and specified as the initial state of the flow domain.

4.8 Chapter Summary

The nozzle simulations were set as 2D axisymmetric, with viscosity and the k-ω turbulence model enabled. Each simulation was designed so that is was comprised of several stages so that the solve time could be minimised where possible. For each consecutive stage, the cell size was decreased until the required spatial resolution was achieved. The input scripts and thus the simulations are otherwise identical. Wall functions were also enabled for later stages. The boundary conditions were defined as :

Wall - Non-slip with a specified surface temperature, viscous effects, and k-ω turbulence enabled. For later (highly resolved) stages of the multistage simulations wall functions were enabled.

Axis - Nozzle centreline was set as the axis of symmetry. Set with the axisymmetry option in `Eilmer4`.

Inlet - Inflow from stagnation conditions.

Outlet - Outflow with a specified back-pressure, $P = 500$ Pa.

Chapter 5

Methodology

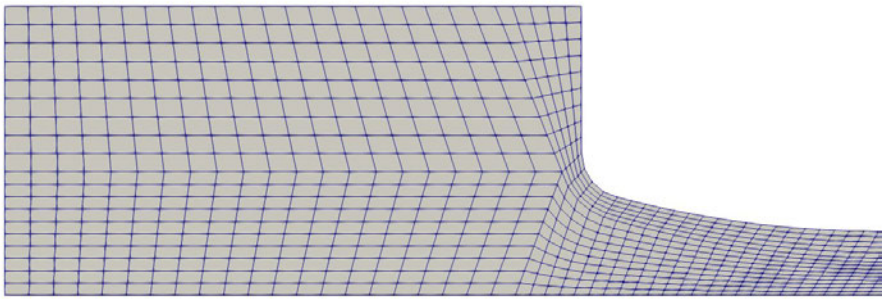
5.1 Chapter Overview

This chapter details the approach taken to design and refine the mesh used in the simulations.

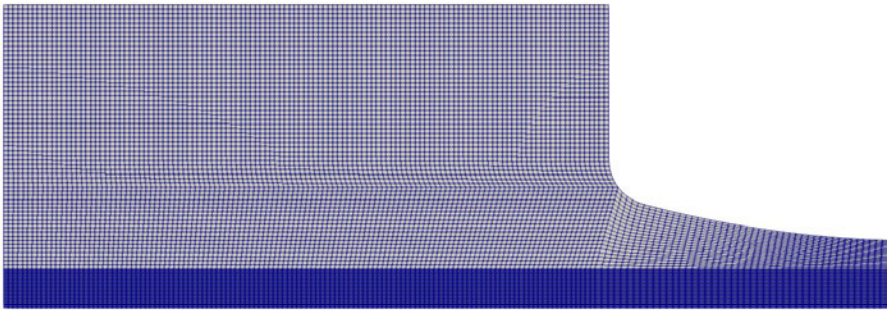
5.2 Initial Mesh Revisions

The first mesh consisted of the nozzle and inlet and was made up of five blocks. As seen in Figure 5.1a, the quality of the cells in this mesh was very poor. There were areas of high skewness and poor orthogonal quality. This was due to the size and shape of the blocks. The second mesh included the test section, and consisted of ten blocks. In order to improve mesh quality near the inlet and at the throat the shape of each block was modified. The initial mesh featured a single point that was the intersection of five separate blocks which led to many cells in this area having high skewness and poor orthogonality. The boundary layer block also terminated at the top of the cylinder at the end of the Ludwieg tube which presented issues in the input script. Assigning the number of vertices to each block was made difficult as some shared their x index with the y index of others. The boundary layer block in the second revision was modified to join to an intermediate block in the inlet which terminated at the inlet face. The top of the cylinder was assigned it's own block, which did not have any effect on the downstream blocks like with the first mesh.

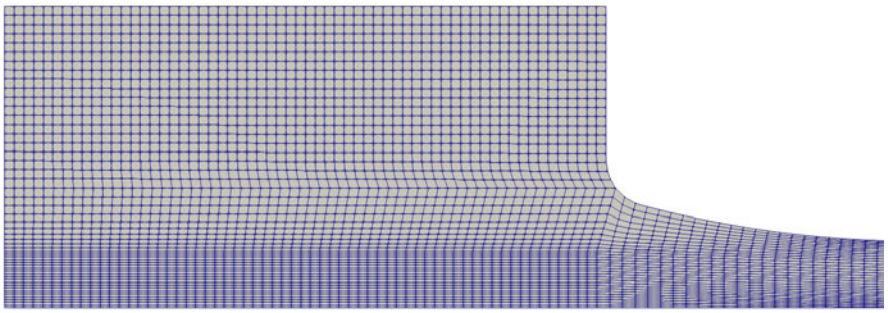
For the purpose of illustrating the variations between each mesh revision, the following figures only show the inlet and throat as these areas are the most geometrically complex parts of the domain.



(a) Revision 1 - Initial mesh and geometry.



(b) Revision 2 - Mesh improved with modified blocks.

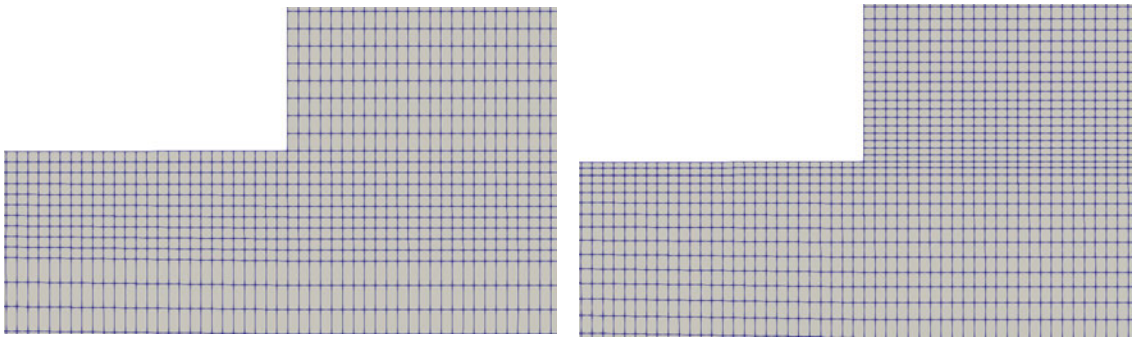


(c) Revision 3 - Mesh improved with clustering and modified blocks.

Figure 5.1: Mesh revisions one to three.

5.3 Clustering

Clustering was implemented to control the distribution of cells in certain blocks. By default, `Eilmer4` automatically distributes the cells along each edge of the blocks while attempting to maintain orthogonality between the cells. In cases where the geometry is already orthogonal (the test section for example) it produces close to perfectly orthogonal cells with minimal skewness. In other cases where block faces are not straight lines, cells can be produced that are problematic. High skewness and poor orthogonality can often be encountered in blocks with sides defined by arcs, splines, or Bezier curves. Clustering can be used to improve mesh quality in these areas by constraining the cells so that they are biased to one side of the block. Clustering can also be used to create areas of high cell density or to reduce severity of the change in cell size between blocks. This is especially useful to allow for the required spatial resolution to capture the effects of the viscous boundary layer without increasing the cell count considerably. An example of clustering can be seen in Figure 5.2.



(a) Un-clustered mesh. The intersection between blocks is visible.

(b) Clustered mesh. Change in cell size between blocks is now gradual.

Figure 5.2: Clustered and un-clustered variations of the mesh at the corner of the nozzle and test section.

5.4 Final Mesh

The third and fourth revisions were quite similar and made use of fluid block arrays and clustering. The fourth version included two more blocks in the nozzle throat as the clustering functions produced uneven cell distributions in the beginning of the diverging section of the nozzle. Introducing two blocks in this area improved the consistency of the clustering. The shown geometry was created entirely with the `Eilmer4` geometry package with the exception of the nozzle contour which was imported from file.

The final mesh consisted of 12 separate blocks (as shown in Figure 5.3). The inlet blocks were all close to rectangular, the right side of block 3 also includes a small section of the nozzle contour to shift block 2 down. This served to minimise the skewness of the cells closest to this point. The inclusion of blocks 6 and 7 in the diverging section of the nozzle served to decrease the overall skewness and reduced the relative size difference of the top and bottom paths for each block. Because `Eilmer4` distributes cells evenly (by default) along the paths that define the quads, having paths on opposite faces of a block that have significantly different lengths can lead to poor quality cells.

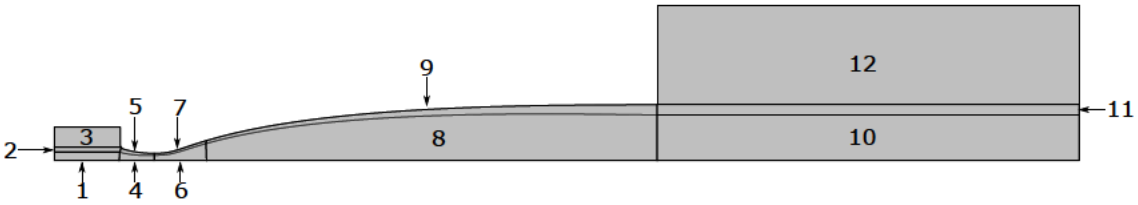


Figure 5.3: The final arrangement of the flow domain. Each block is numbered according to the same scheme as the input script.

The distribution of cells and sub-blocks is shown in Table 5.1 for reference. The number of cells in x and y for each block was defined parametrically with respect to a refine factor (R) so that the resolution of the entire domain could be scaled at once. The refine factor was the global cell size control and was implemented in the input script so that all definitions of cell counts could be modified with one control (See Appendix B, code section 'Grids and Meshing' for implementation). The cell counts for each block are consequently a function of R^2, so the overall cell count could be estimated by $n \approx 2.29 \times 10^6 R^2$. Refine factors of less than 4 were used for all simulations.

Table 5.1: Number of cells (for a refine factor of 1) and sub-blocks in x and y each block. Totals are also included for each block.

Block	**Cells** x	**Cells** y	**Sub-Blocks** x	**Sub-Blocks** y	**Sub-Blocks**	**Cells**
1	125	40	4	2	8	5000
2	125	20	4	1	4	2500
3	125	55	4	1	4	6875
4	60	40	2	2	4	2400
5	60	20	2	1	2	1200
6	120	40	3	2	6	4800
7	120	20	3	1	3	2400
8	900	40	12	2	24	36000
9	900	20	12	1	12	18000
10	830	40	12	2	24	33200
11	830	20	12	1	12	16600
12	830	120	12	3	36	99600
				Total	139	228575

The throat and inlet of the final mesh is shown in Figure 5.4. Clustering was included in this area to create a gradual change in cell size between blocks. In comparison to Figure 5.1c, the final mesh has a more gradual change in cell size and a more consistent cell size in the inlet section.

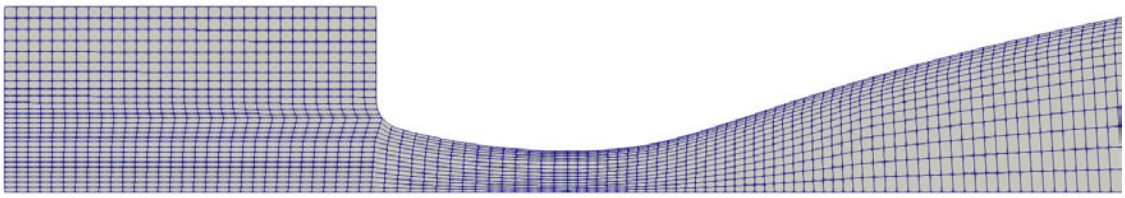


Figure 5.4: Cylinder end and nozzle throat.

5.4.1 Mesh Quality

For this application, mesh quality is a crucial parameter. Figure 5.5 and Figure 5.6 show the cell skewness and edge length ratios for the most geometrically complex area of the domain - the nozzle throat. Due to the nozzle contour and size reduction at the throat, poor quality cells can easily be created. Clustering and appropriate positioning and sizing of blocks has allowed for the creation of cells that are within reasonable mesh quality tolerances.

The largest skewness value occurs in the nozzle inlet in the corners of blocks 3 and 5 where the skewness reaches a maximum of 0.59. Bakker (2006) and ANSYS Inc. (2010) categorised cell skewness into six quality ranges as shown in Table 5.2. As the cell skewness for the final mesh does not exceed 0.6, even the poorest cell skewness is still within the acceptable range. Across the majority of the domain, the cell skewness would be considered to be 'good' as values are less than 0.5. Cell aspect ratio is also no larger than 13, with the highest edge length ratio occurring at the nozzle throat. This is much smaller than the limit found by Chan et al. (2011) of 600. As such the mesh is of suitable quality, and has no exceptionally poor quality cells.

Table 5.2: Cell Skewness ranges. Adapted from Bakker (2006) and ANSYS Inc. (2010).

Skewness	0-0.25	0.25-0.5	0.5-0.8	0.8-0.95	0.95-0.99	0.99-1
Cell Quality	Excellent	Good	Acceptable	Poor	Sliver	Degenerate

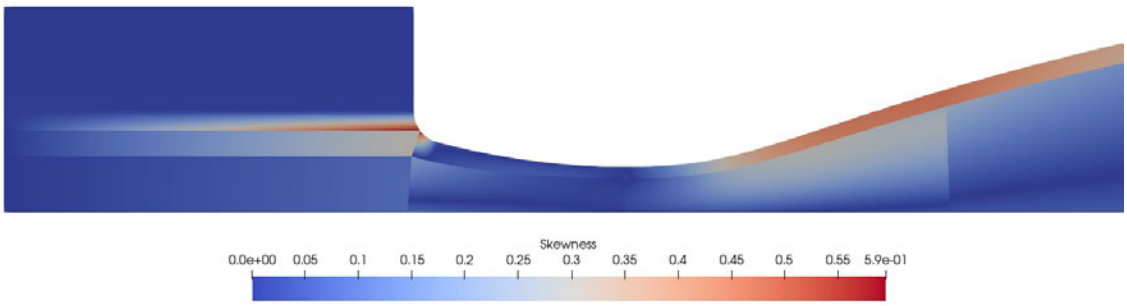


Figure 5.5: ParaView plot of skewness of the nozzle throat and inlet using the 'Mesh Quality' filter with the 'Skew' quadrilateral property. High skewness can be observed at the south east corner of block 3 and the north west corner of block 5.

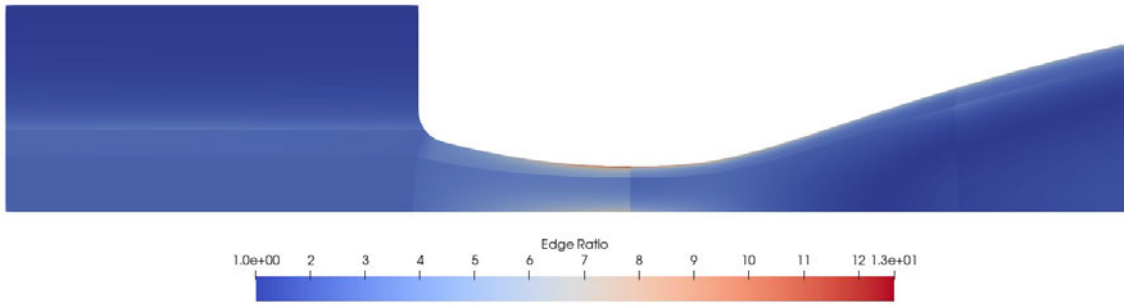


Figure 5.6: ParaView plot of cell edge length ratio of the nozzle throat and inlet using the 'Mesh Quality' filter with the 'Edge Ratio' quadrilateral property. A maximum edge ratio of 12 occurs in the throat close to the wall. Values across the rest of the domain range between 1 and 5.

5.5 Non-Dimensional Distance to the Wall

As the model utilises the k-ω turbulence model as well as wall functions, the non-dimensional distance to the wall (y^+) is crucial in obtaining accurate results. As noted by Nichols & Nelson (2004), a y^+ of less than 100 allows the wall functions to produce reasonable results. Very small y^+ values were also found to be suitable for the k-ω turbulence model by Chan et al. (2011) for supersonic flows. As such $y^+ < 100$ would be suitable for the simulations of the Mach 6 and Mach 7 nozzles. The flow data was analysed using MATLAB and the wall y^+ value for the nozzle contour was calculated using the equations noted by White (2003).

The wall spacing (size of the first cell close to the wall) was estimated as the distance between cell centres of the two rows of cells closest to the wall. This was used with the centreline velocity, density, and viscosity to calculate the wall y^+ for each cell along the length of the diverging section of the nozzle. Shown in Figure 5.7 is the y^+ value as a function of x distance along the nozzle. As seen in the figure, the y^+ value varies between 20 and 60. As the y^+ value does not exceed 100 (shown by the wall spacing subplot in Figure 5.7), the cell size close to the wall is small enough to provide the required spatial resolution for the turbulence model and wall functions.

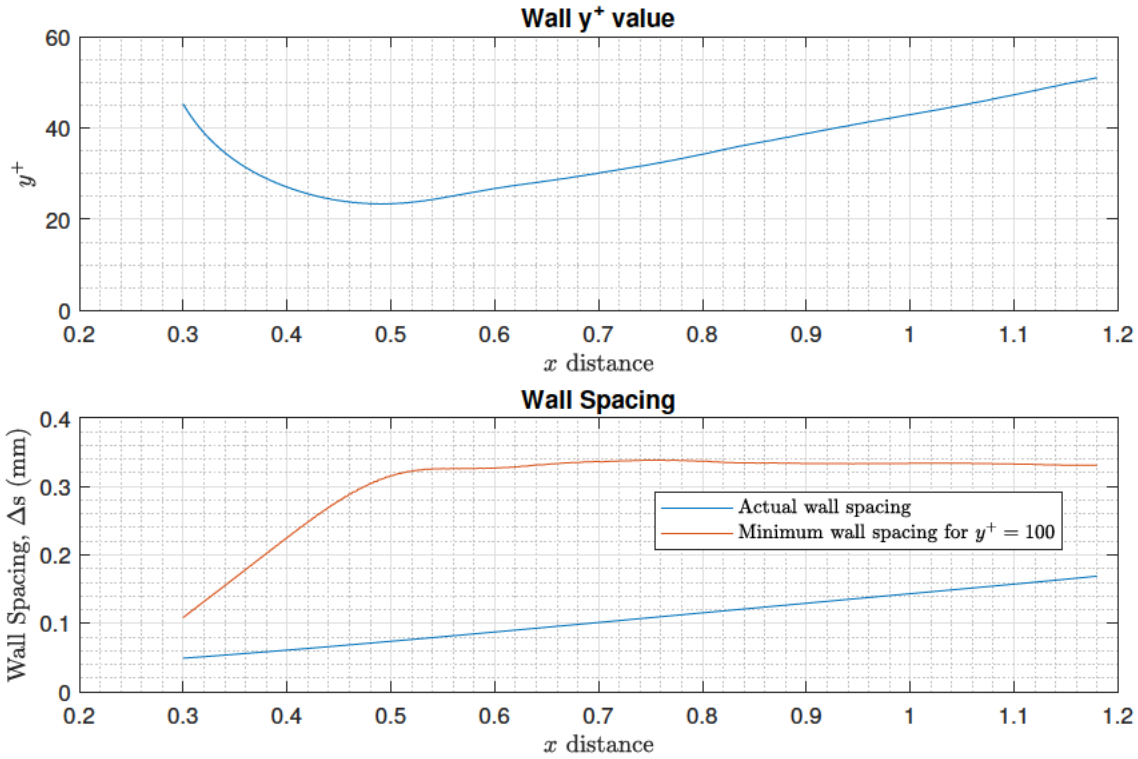


Figure 5.7: Wall y^+ and wall spacing for the Mach 6 nozzle.

5.6 Mesh Refinement

The flow data at the exit plane was extracted using a built in post-processing utility of `Eilmer4`. The 'extract line' function was used, which allowed for the flow properties at each point intersected by a line (defined by it's end points). The output was written to a text file and plotted using MATLAB. Because the number of cells in each mesh was different, the number of points extracted was also different for each. The location of the exit plane was at $x = 1.187$m, and the line spanned the entire radius of the nozzle exit. The flow data for each mesh was used to estimate the boundary layer thickness (and core flow diameter). The average core flow properties were also found. The mesh refinement analysis was performed using the script shown in Appendix C.

The mesh refinement study consisted of a four stage simulation, with each stage having a larger cell count than the preceding stage. The four stages had cell counts of 14×10^3, 57×10^3, 2.1×10^6, and 3.3×10^6 respectively. Because it was run as a multistage simulation, each stage was initialised using the results of the previous stage. Apart from the initial condition and cell count, every remaining factor was the same for each stage. The Mach

7 nozzle was simulated for a stagnation pressure of 7MPa. Properties at the exit plane for each mesh are shown below in Figure 5.8. As seen in the figure, the first two meshes produced results that differed by a large amount. When compared to the third mesh, the first two results would not be considered mesh independent, as flow parameters differ between each stage. The third and fourth meshes however produce mesh independent results, as the difference in flow parameters between each is less than 1% in most cases as shown in Figure 5.9.

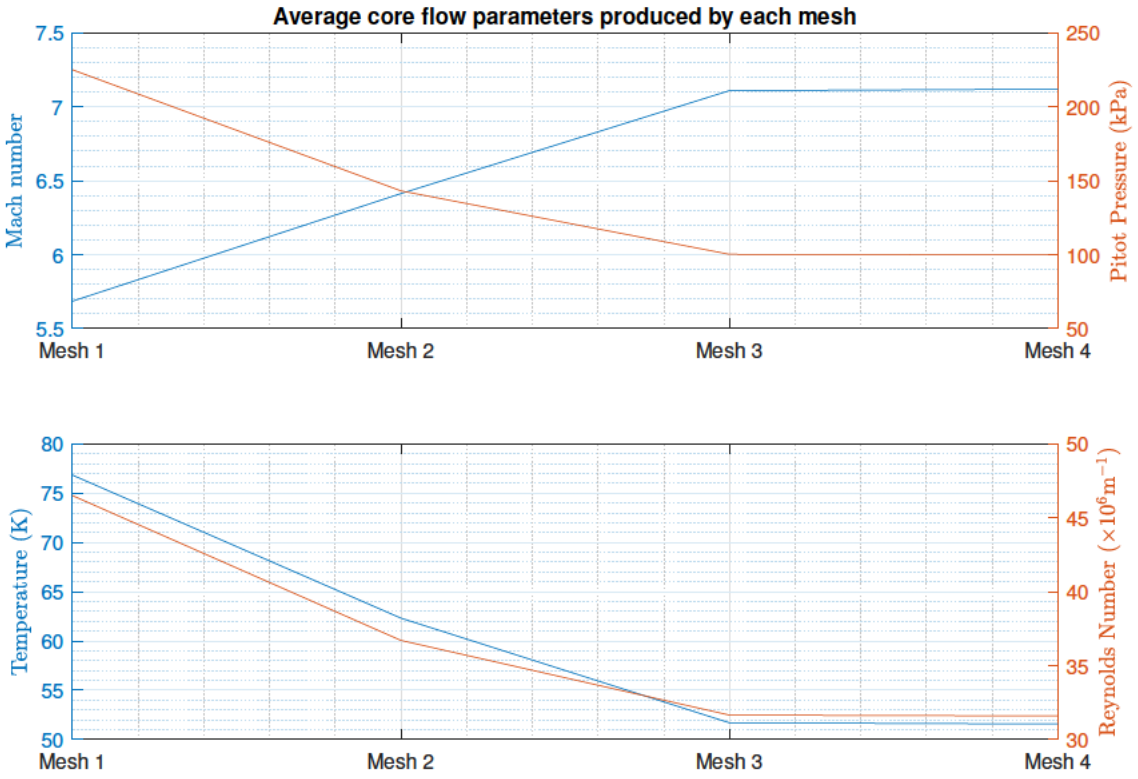


Figure 5.8: Mach number, pitot pressure, temperature, and Reynolds number for each mesh.

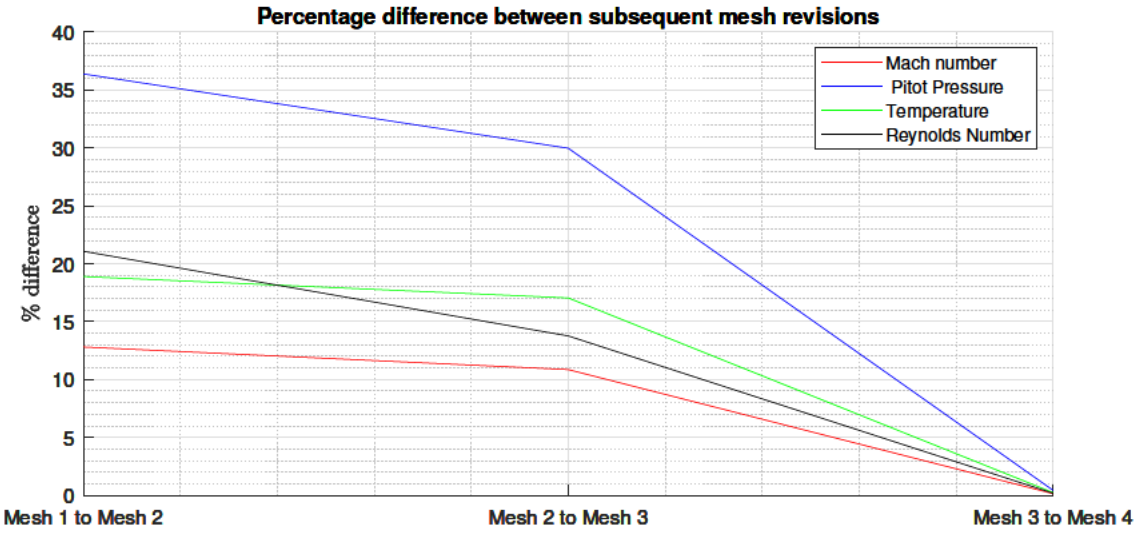


Figure 5.9: Percentage difference in flow parameters between subsequent mesh revisions.

The thickness of the boundary layer is of particular interest, because as the cell count is increased, the boundary layer decreases in size. The first mesh produces a boundary layer that is approximately 50mm in thickness, this decreased the area ratio of the nozzle and thus also decreased the Mach number at the exit plane. For reference, the boundary layer for the Mach 6 nozzle is expected to be roughly 10-20mm in thickness. The first mesh boundary layer is more than twice that value, and is unrealistic. The second mesh decreases the size of the boundary layer slightly to 40mm which is still considered to be unrealistic. The third and fourth stages have wall functions enabled, which also affected the boundary layer thickness. Wall functions were enabled so that the boundary layer did not require such a small spatial resolution, thus decreasing the solve time and cell count while maintaining the required accuracy. For stages 1 and 2, the boundary layer resolution was not sufficient for the use of wall functions.

Because of both the implementation of wall functions and the smaller cell size, the third and fourth meshes have an almost identical boundary layer thickness of approximately 9.5mm, which results in a flow regime of approximately Mach 7.1 at the nozzle exit. Other flow parameters can be seen to differ greatly compared to the first two meshes in Figure 5.10. The difference between the third and fourth meshes however is minimal, and as mentioned before for most parameters is less than 1%. Based on this very small difference, it was reasoned that the third mesh produced the most computationally efficient, yet accurate results. Cell counts of more than 2.1×10^6 were deemed to produce diminishing returns on accuracy with much longer solve times. From the log files, the solve time of

the fourth mesh was more than double that of the third (at 220 hours compared to 96 hours) for a 50% increase in cell count and negligible change in accuracy.

Shown in Figure 5.10 is the exit plane velocity and Mach number distribution. Again, the first two mesh refinements produced significantly different results to the last two. The cell count, and core flow properties for each mesh are shown in Table 5.3. The values for meshes 3 and 4 can be seen to be very similar.

As the Mach 7 mesh is identical to the Mach 6 mesh except for the nozzle contour, it was reasoned that such a small difference in shape would not introduce unwanted error, or significantly affect the mesh independence or quality. As such, this mesh independence study also applies for the Mach 6 simulations.

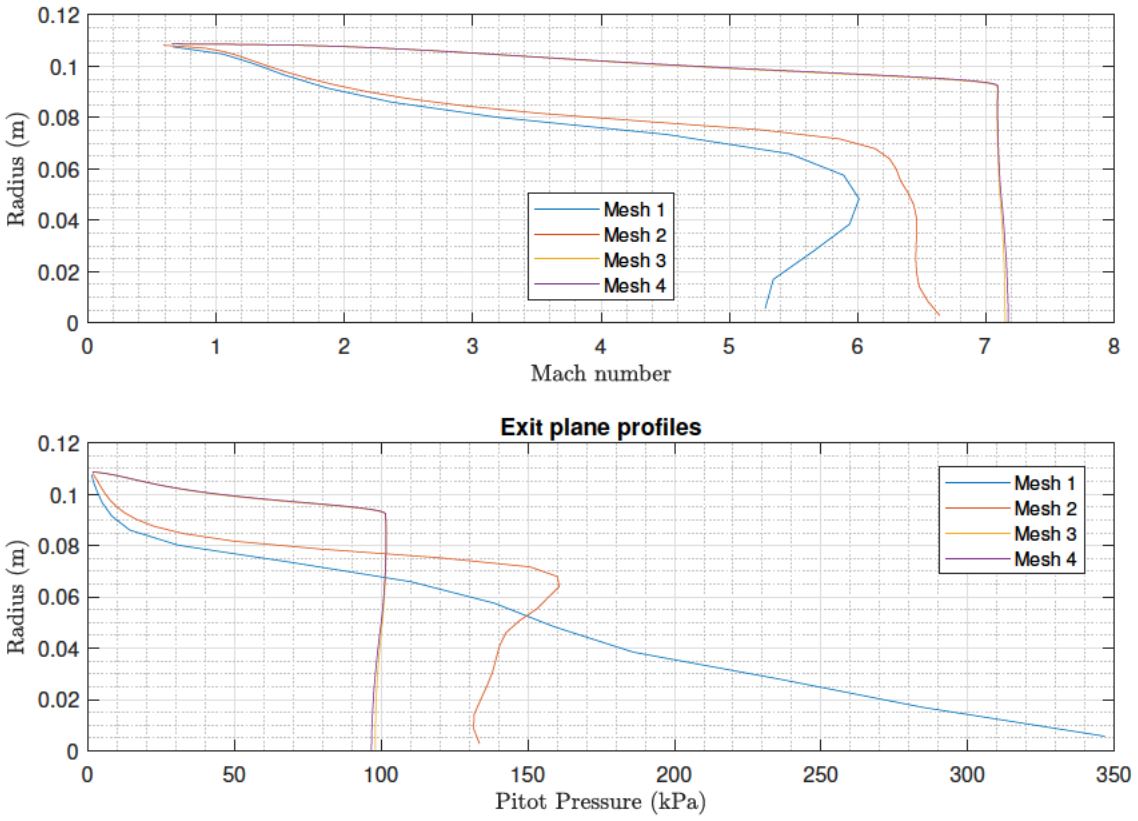


Figure 5.10: Mach number and pitot pressure distributions/profiles at the nozzle exit plane for each mesh. Mesh 3 and 4 produced very similar results.

Table 5.3: Properties of the core flow at the nozzle exit for each mesh. Number of cells in each mesh is also included.

	Cells	δ **(mm)**	$\overline{v}_x$ **(m/s)**	$\overline{M}$	$\overline{P}_{pitot}$ **(kPa)**	$\overline{P}$ **(kPa)**	$\overline{T}$ **(K)**
Mesh 1	14188	50.1	995.4	5.69	225.1	5.561	76.83
Mesh 2	56934	36.6	1014.6	6.41	143.2	2.690	62.32
Mesh 3	2051800	15.2	1024.9	7.11	100.3	1.530	51.70
Mesh 4	3292000	15.1	1025.1	7.12	99.8	1.519	51.58

5.7 Chapter Summary

The mesh used in the simulations went through multiple revisions before it was suitable for use. The final mesh included clustering to improve the uniformity of the cells and utilised sub-blocks to allow for the simulations to be run efficiently using the USQ HPC. The mesh quality was checked using ParaView and the cell skewness and edge length ratio were found to be within acceptable ranges. Finally, a mesh independence study was performed on the Mach 7 mesh and the optimum cell count was found to be 2.1×10^6. Due to the similarity between the Mach 7 and Mach 6 meshes it was reasoned that the Mach 6 mesh would also be of acceptable quality.

Chapter 6

Results

6.1 Chapter Overview

This chapter presents the results of the simulations for both nozzles and provides an analysis on the effects of varying the stagnation condition. It also provides validation of the Mach 6 simulation by comparing the results to historical experimental and theoretical data.

6.2 Mach 6 Nozzle

The Mach 6 nozzle was run using four different stagnation conditions. These were 1MPa 575K, 1MPa 900K, 4MPa 575K, and 7MPa 575K. The plots shown below were produced with MATLAB, all plots use the 'Jet' colour map. For extra information and code relating to the analysis of the flow data, see Appendix D.

Shown in Figure 6.1 is the contour plot for Mach number for each stagnation pressure. It can be noted from this figure that increasing the stagnation pressure also increases the size of the expansion in the test section, and thus the Mach number closer to the outer radius of the test section. This expansion is due to the different nozzle pressure ratios. Nominal stagnation conditions of 1MPa at 575K coupled with a test section back-pressure of 500Pa, produce a pressure ratio ($\frac{P_T}{P}$) of 2000. The nozzle was designed to produce an adapted flow, so assuming the path from inlet to outlet is isentropic (and $\gamma = 1.4$ for air),

the ideal pressure ratio could be found as:

$$\begin{aligned}\frac{P_o}{P} &= \left(1 + \frac{\gamma - 1}{2} M^2\right)^{\frac{\gamma}{\gamma - 1}} \\ &= \left(1 + \frac{1.4 - 1}{2} 6^2\right)^{\frac{1.4}{1.4 - 1}} \\ &= 1578.9\end{aligned}$$

For pressure ratios higher than this the nozzle outflow would be under-expanded. This is noted to be the case for each stagnation pressure result to varying degrees. The nominal condition of 1MPa produces a slightly under-expanded nozzle outflow, with a Mach reflection forming just past the nozzle exit and converging on the nozzle centreline approximately 600mm downstream. Expansion of the flow can be seen to occur at approximately 1.4m. With the 900K result, the expansion is shifted away from the nozzle exit by approximately 200mm to $x = 1.6$m. The 4MPa stagnation pressure produces a pressure ratio of 8000, which causes an even greater expansion than the nominal conditions. Expansion begins at the nozzle exit and which causes the nozzle outflow to increase in diameter significantly. The Mach reflection at the nozzle exit for this condition formed closer to the exit plane than for the nominal condition. The 7MPa stagnation pressure produced a similar Mach number distribution to the 4MPa stagnation pressure.

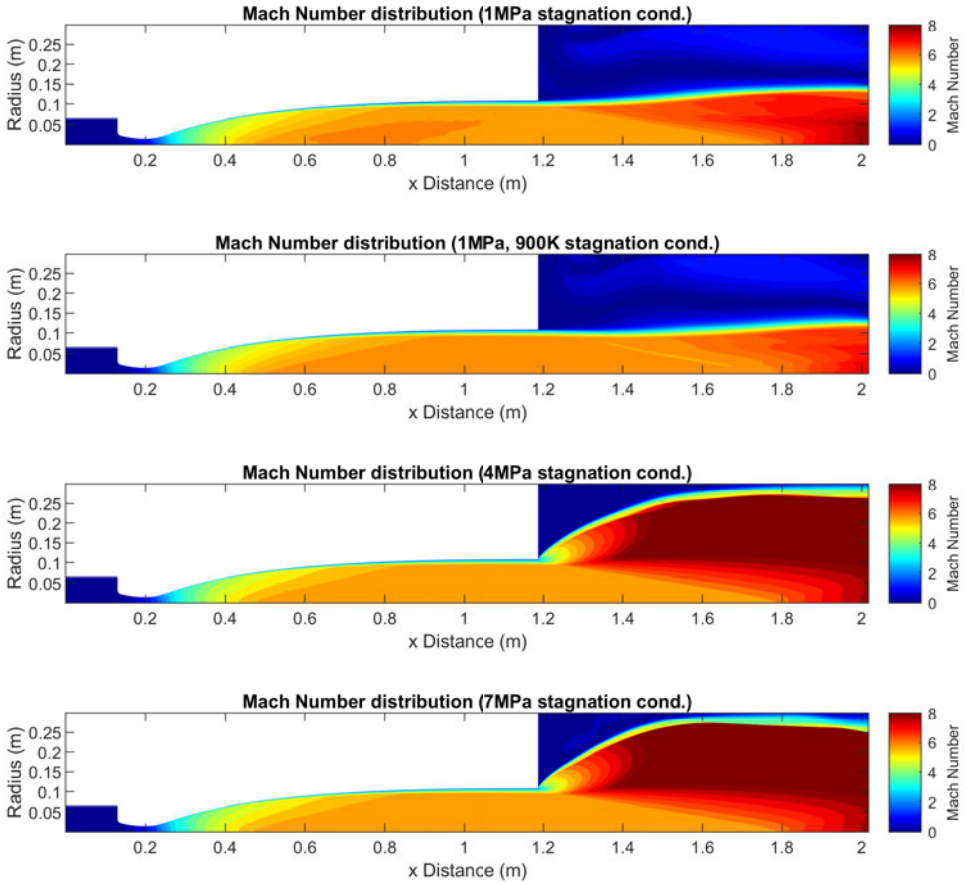


Figure 6.1: Local Mach number contour plots for the test section under the four stagnation pressures.

The nominal core flow regime for each stagnation condition is very similar at approximately Mach 6. The main difference, however is the size of the usable part of the core flow which is shown in Figure 6.2. Locations where the Mach number was greater than 6.1 were replaced with white space. The nominal flow regime is Mach 5.9, so $\pm 3\%$ gives a maximum of approximately Mach 6.1. This area (white space) is not suitable for experimental purposes as the Mach number differs significantly from the required value. The area adjacent to the nozzle exit (shown in red in each case) is the usable volume of the core flow. For the 1MPa result it can be seen that the usable volume has a relatively consistent radius immediately following the nozzle exit, which transitions to a triangular area at $x = 1.3$m. The 4MPa and 7MPa results also show a decrease in radius at this point and are somewhat similar in this regard. The exception to this is the 1MPa 900K result, which has by far the largest usable volume, with the reduction in radius beginning at $x = 1.5$m.

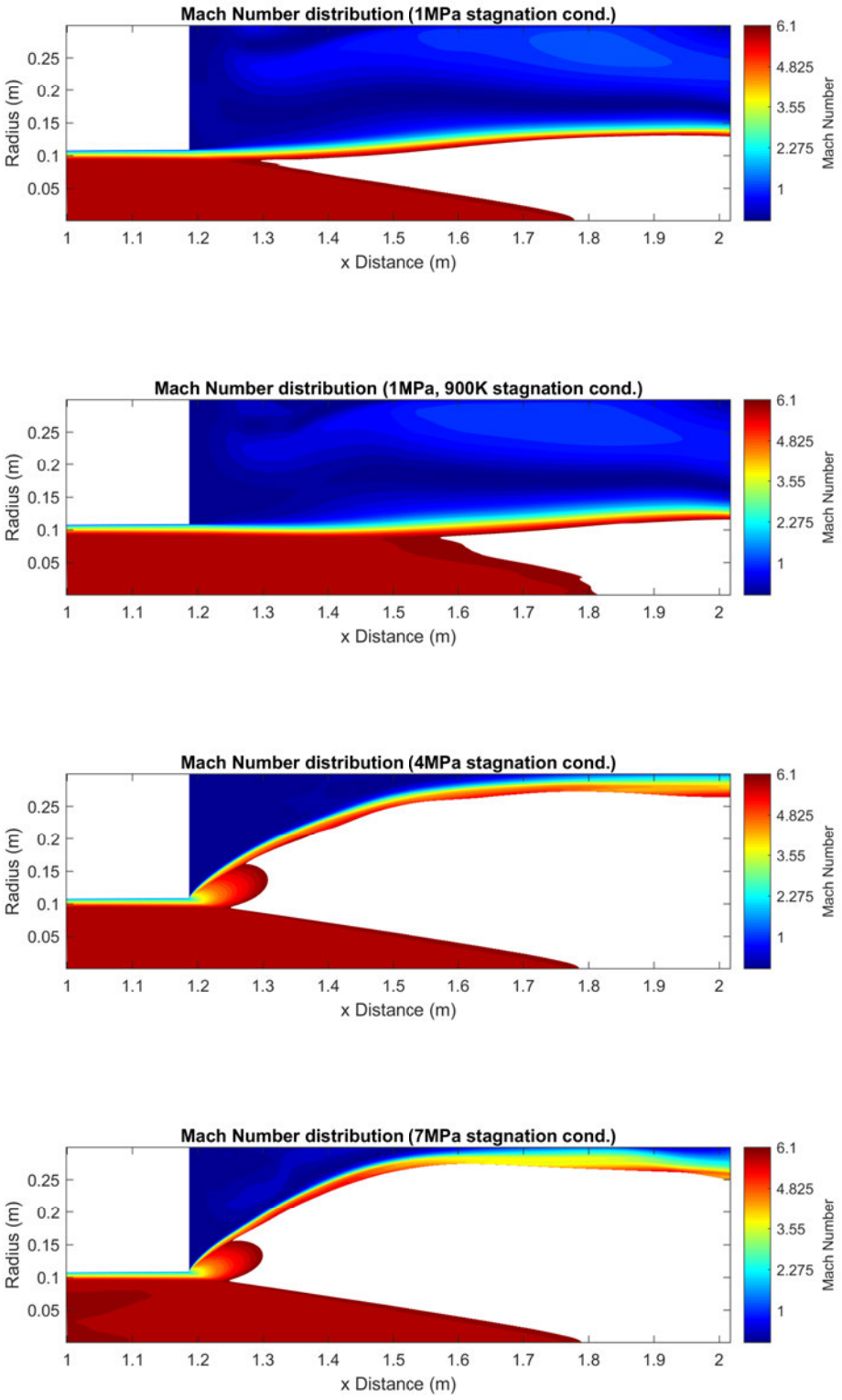


Figure 6.2: Contour plots showing the area of usable core flow where the Mach number does not exceed 6.1.

The 1MPa 900K result also has a higher core flow temperature and smaller Reynolds number as shown in Figure 6.3 and Figure 6.4. This was to be expected, as the temperature directly affects the density of the flow, which was approximately 0.03kg/m^3 for the 1MPa condition and 0.02 kg/m^3 for the 1MPa 900K condition.

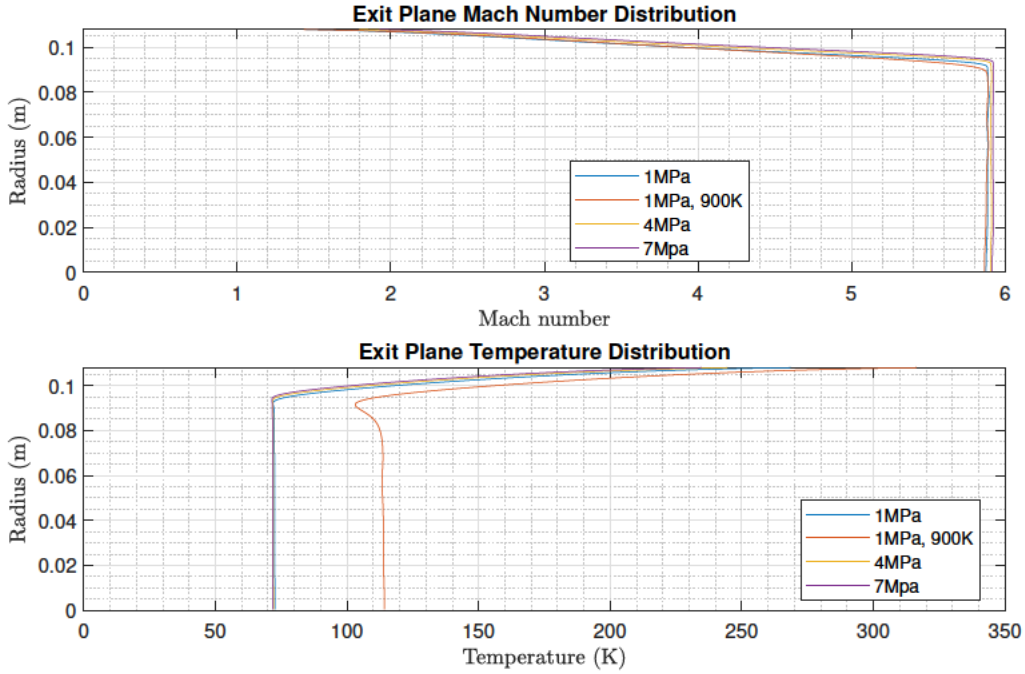


Figure 6.3: Mach number and Temperature distribution at the nozzle exit.

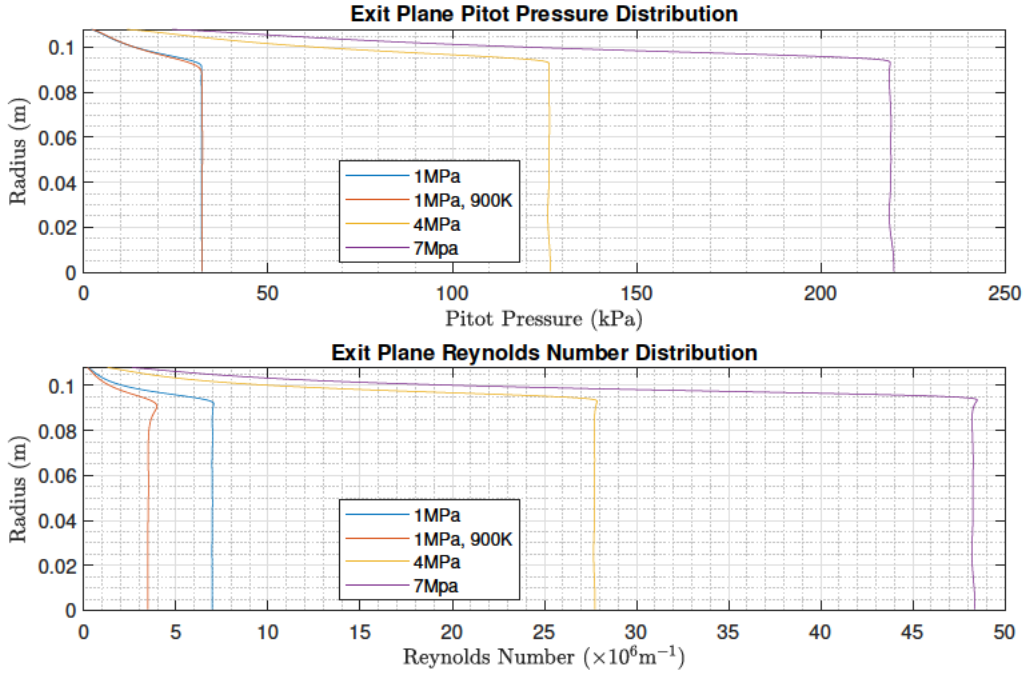


Figure 6.4: Pitot pressure and Reynolds number distribution at the nozzle exit.

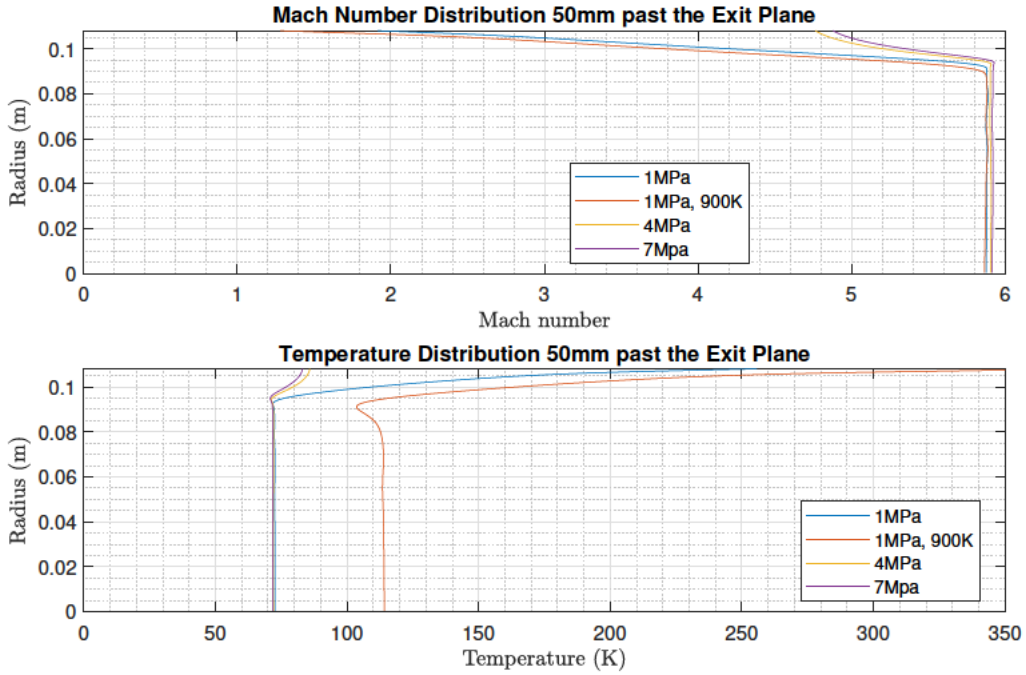


Figure 6.5: Mach number and Temperature distribution at 50mm past the nozzle exit.

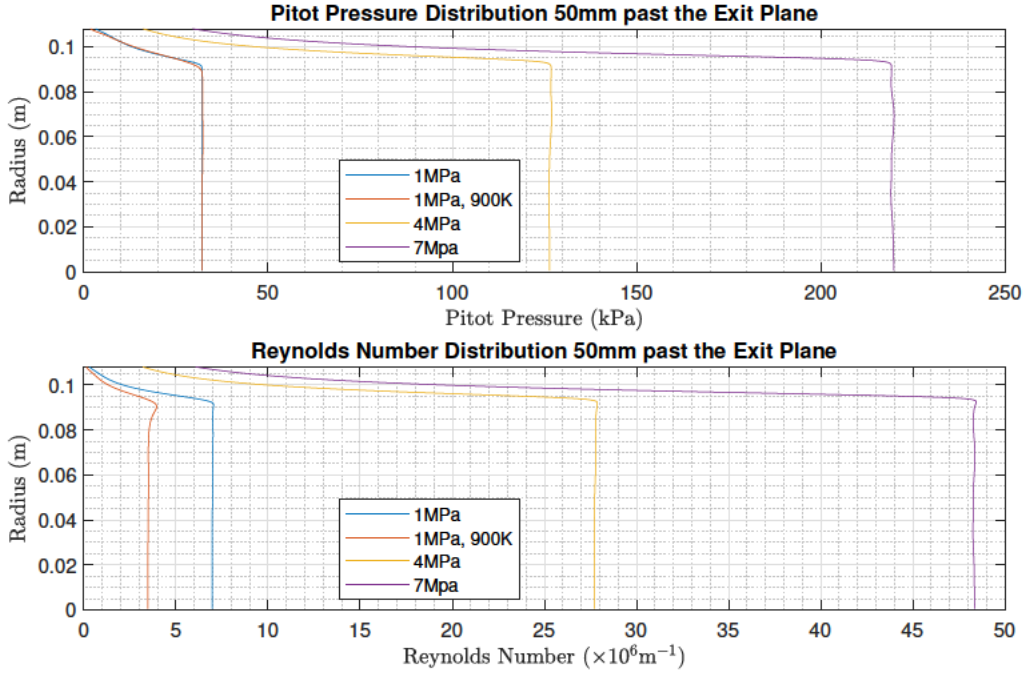


Figure 6.6: Pitot pressure and Reynolds number distribution at 50mm past the nozzle exit.

6.2.1 Model Validation

The simulation results for the Mach 6 nozzle produced with the 1MPa 575K stagnation condition was analysed using MATLAB. The results of the analysis included the average core flow parameters at the nozzle exit plane. These results are shown in Table 6.1.

Table 6.1: Average core flow parameters at the nozzle exit for the viscous Mach 6 nozzle simulation.

Property	Unit	`Eilmer4` (Viscous)
M_∞	-	5.874
P_∞	Pa	708.66
T_∞	K	72.54
ρ_∞	g/m^3	31.812
$U_{x,\infty}$	m/s	1002.9
$U_{r,\infty}$	m/s	-1.568
Re	$\times 10^6$m^{-1}	6.943

Birch (2019) presented results in their study for the Mach 6 nozzle for the same stagnation conditions. They presented the expected nominal flow conditions as well as the results of an inviscid `Eilmer4` simulation. The nominal conditions were calculated based on the assumption of isentropic flow for an outlet Mach number of $M = 5.9$. As such, the properties listed under 'nominal conditions' were not measured experimental quantities, but were instead theoretical estimates based on the expected Mach number. They also presented plots of time averaged pitot pressure and Mach number over the duration of a test run. Average values (for $t < 50$ ms) were interpolated from the plots (thus introducing uncertainty) and are shown below. Flow conditions presented by Birch (2019) are summarised in Table 6.2.

Table 6.2: Flow parameters presented by Birch (2019).

Property	Unit	Nominal Conditions	Time Averaged	`Eilmer4` (Inviscid)
M_∞	-	5.9	5.925	5.98
P_{pitot}	kPa	31.8	30.2	-
P_∞	Pa	702	-	648
T_∞	K	72	-	70.6
U_x	m/s	1006	-	1007
Re	$\times 10^6$ m^{-1}	6.94	-	-

The average core flow parameters were found to be within 1% of the nominal flow conditions. Percentage difference between the nominal conditions and the average core flow values from the simulation results are shown in Table 6.3. The pitot pressure and Mach number differed by 0.18% and 0.66% respectively compared to the time-averaged experimental values. Flow parameters in the usable area in the test section are sufficiently close to the time averaged values noted by Birch (2019), as well as the nominal flow conditions. Thus the 1MPa simulation is a reasonably accurate representation of the flows produced by TUSQ.

Table 6.3: Comparison to results obtained by Birch (2019)

Property	Unit	Nominal (Birch 2019)	Viscous `Eilmer4`	Difference
M	-	5.9	5.886	0.238%
P	Pa	702	708.7	0.954%
P_{pitot}	kPa	31.8	31.942	0.447%
T	K	72	72.42	0.577%
U_x	m/s	1006	1004.2	0.180%
Re	$\times 10^6$m^{-1}	6.94	6.974	0.492%

6.3 Mach 7 Nozzle

The Mach 7 nozzle was run using three different stagnation conditions. These were 1MPa 575K, 4MPa 575K, and 7MPa 575K. Like with the Mach 6 nozzle results, the plots shown in this section were produced with MATLAB and utilise the 'Jet' colour map.

Using the isentropic flow relations again, the ideal pressure ratio for the Mach 7 nozzle is $\frac{P_o}{P} = 4139.8$. For the nominal stagnation pressure of 1MPa and back-pressure of 500Pa, the pressure ratio is 2000. Shown in Figure 6.7 is the Mach number distribution in the nozzle and test section. The 1MPa stagnation condition produced an over-expanded flow (due to the lower pressure ratio), with the outflow decreasing in diameter until $x \approx 1.8$m. An oblique shock formed inside the diverging section of the nozzle, approximately 30mm from the exit.

The 4MPa and 7MPa results appear to be under-expanded, and undergo a significant expansion in the test section. The 7MPa result also shows some artifacting at the outer wall of the test section at $x = 1.5$m. As these plots are produced from an average of three time-steps, if any one time-step is significantly different to another the averaging process will allow for areas with high Mach numbers appear in the final plot. The artifacting in this figure is a direct result of this, meaning that the simulation would require to be run for a longer flow time to prevent this from occurring. However, this does not affect the results near the area of interest (i.e. the nozzle exit), and so the results were suitably temporally resolved for the scope of the project.

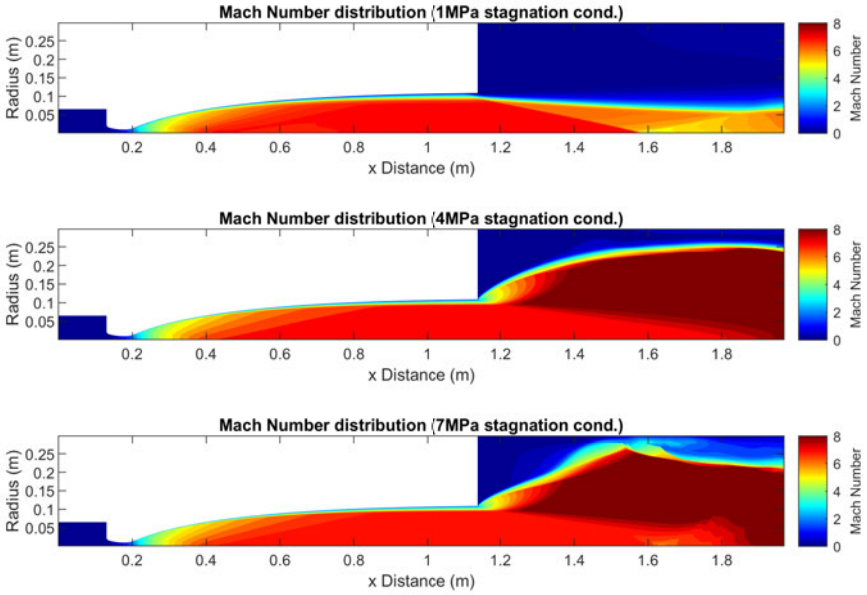


Figure 6.7: Local Mach number contour plots for the test section under the three stagnation pressures. The 7MPa plot shows remnants of previous time-step data near the top of the test section due to the averaging conducted as part of the post processing.

Figure 6.8 shows the usable core flow area (in red), where the Mach number does not exceed Mach 7.2 (nominal flow regime is Mach 7, so ±3% gives a maximum of Mach 7.2). The 1MPa result has the smallest usable area of the three stagnation conditions due to the oblique shock, at approximately 500mm in length ($x = 1.1$m to $x = 1.6$m). The 4MPa result produced the second largest usable area at 700mm in length. The 7MPa result had areas of high and low Mach number towards the end of the usable core flow, again this was caused because the solution was not sufficiently temporally resolved, however based on the behaviour of the core flow area for the Mach 6 nozzle it is expected that the usable area is somewhat similar to that of the 4MPa result.

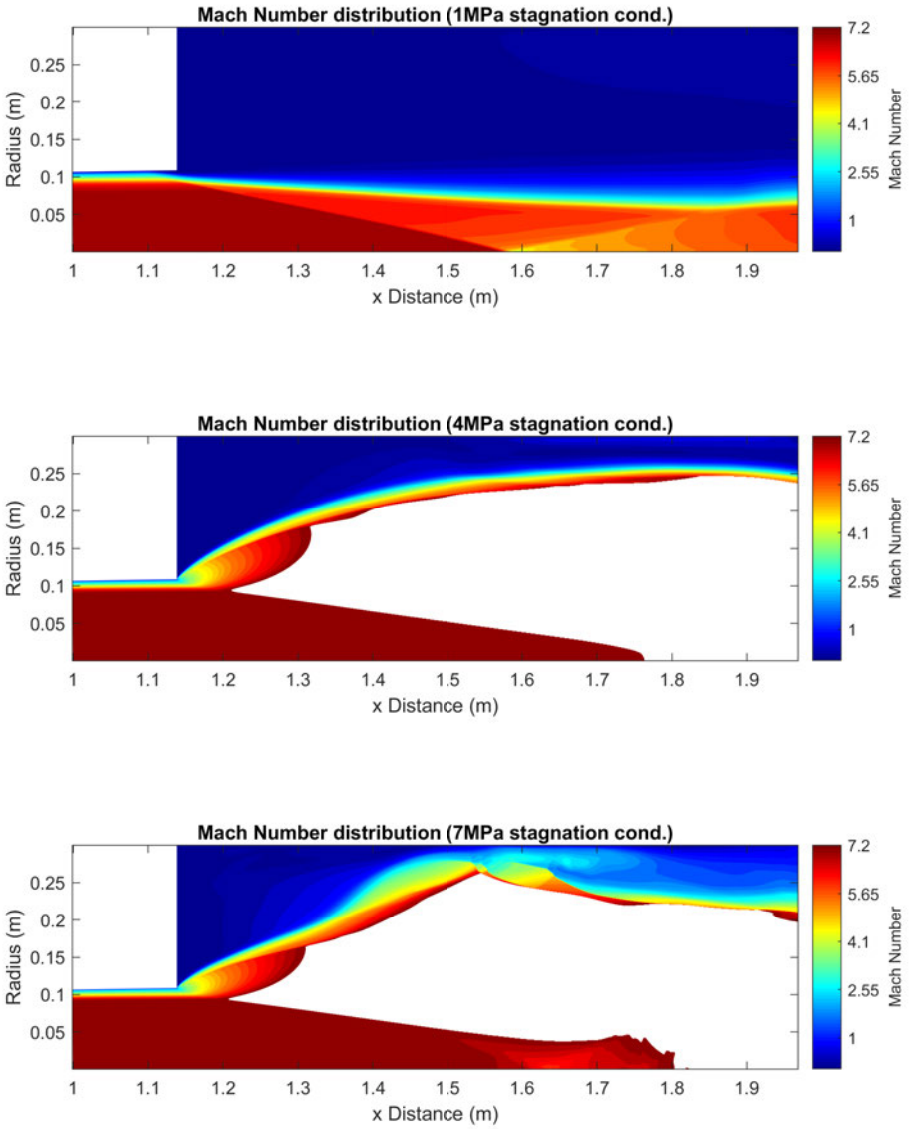


Figure 6.8: Contour plots showing the area of usable core flow where the Mach number does not exceed 7.2.

Figures 6.9 and 6.10 show the flow properties at the nozzle exit as a function of radius. The boundary layer can be seen to begin between $R = 0.08$m and $R = 0.1$m. In this region the flow parameters (particularly Mach number and Reynolds number) decrease to close to zero at the nozzle wall. The effect of varying the stagnation pressure is more apparent with these figures, as the Reynolds number and Pitot pressure however, increase almost in proportion to stagnation pressure. The Mach number and free stream temperature however remain almost unchanged.

The same phenomena can be observed 50mm past the nozzle exit in Figures 6.11 and 6.12. These figures indicate that the flow parameters are similar to those at the nozzle exit (similar to the Mach 6 nozzle). With the main exception being the 1MPa stagnation condition. Here, the oblique shock causes an abrupt change in flow conditions at a radius of approximately 80mm. This phenomenon is not observed with the other stagnation conditions. The Pitot Pressure and Mach number can also be seen to transition more smoothly at a radius of 80mm for the 4MPa and 7MPa conditions.

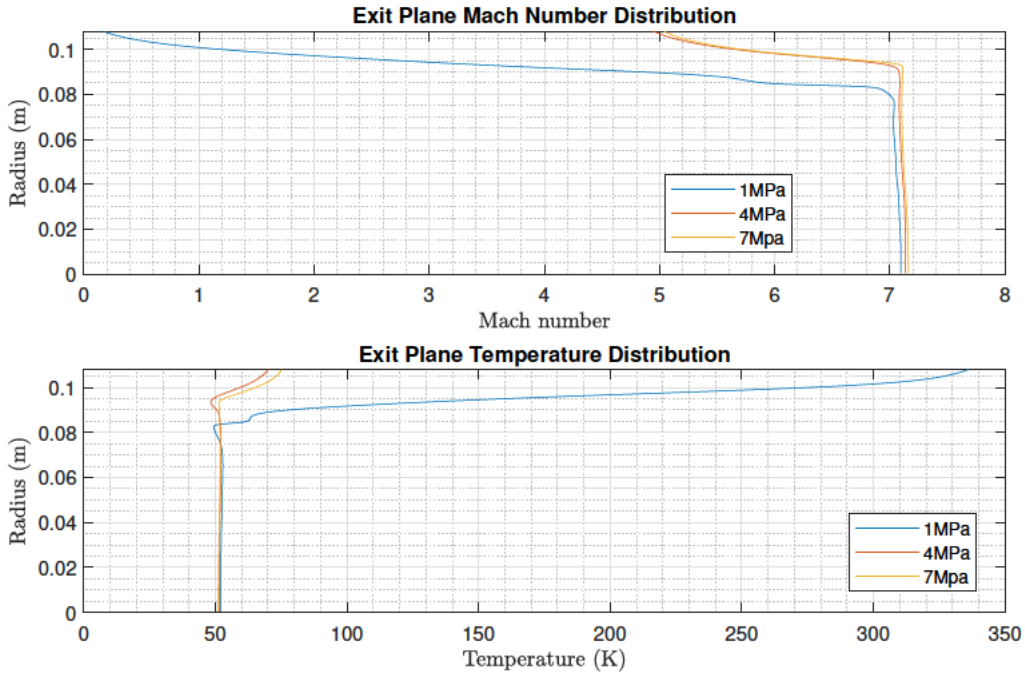


Figure 6.9: Mach number and Temperature distribution at the nozzle exit.

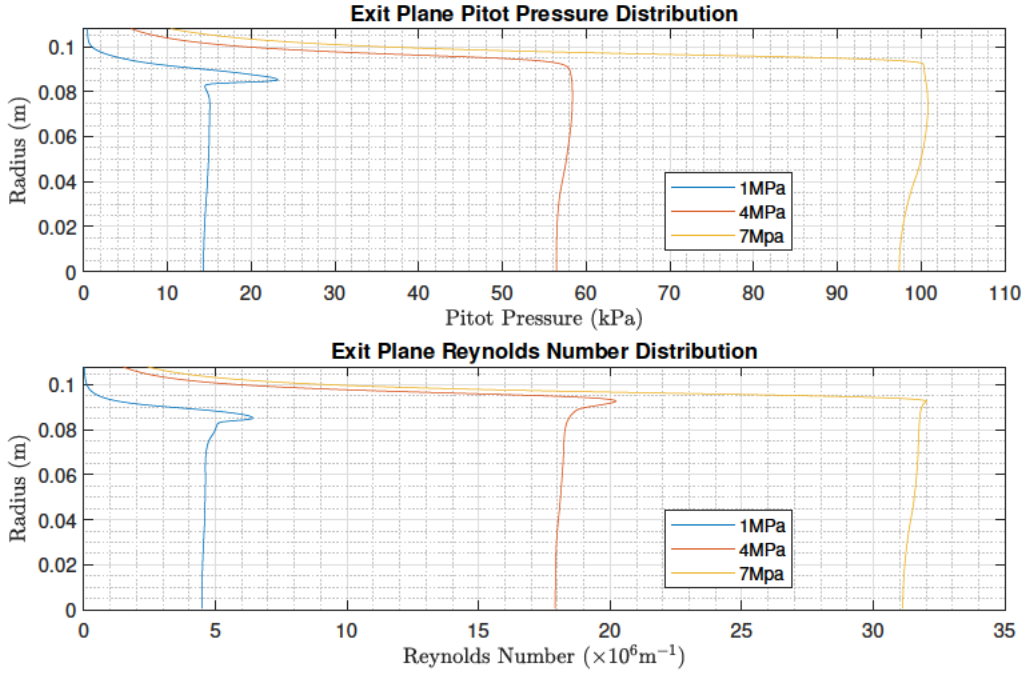


Figure 6.10: Pitot pressure and Reynolds number distribution at the nozzle exit.

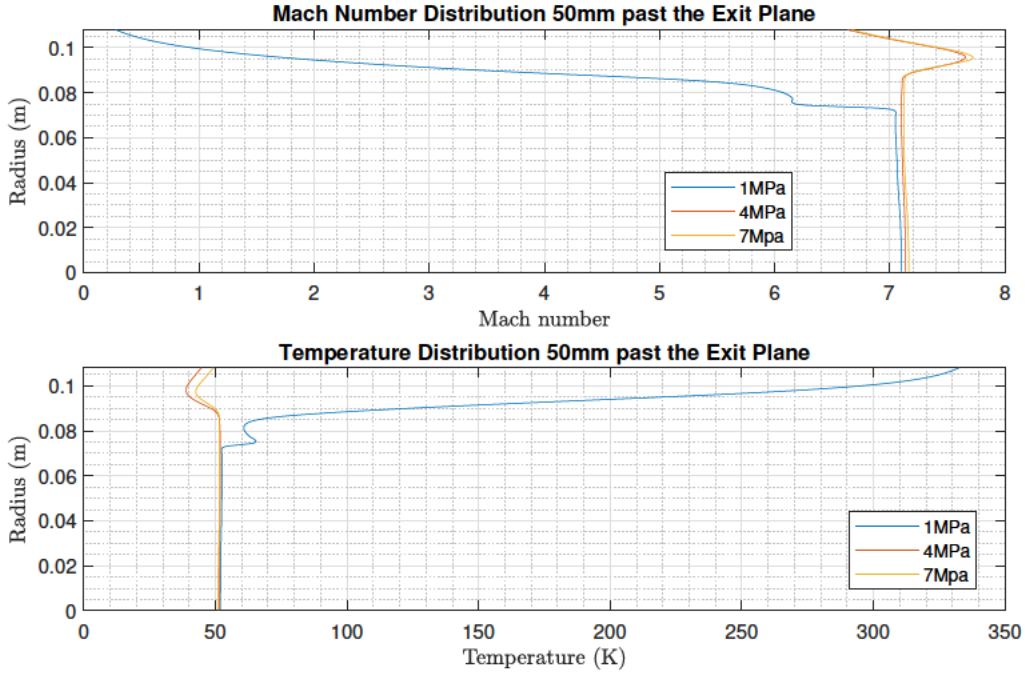


Figure 6.11: Mach number and Temperature distribution at 50mm past the nozzle exit.

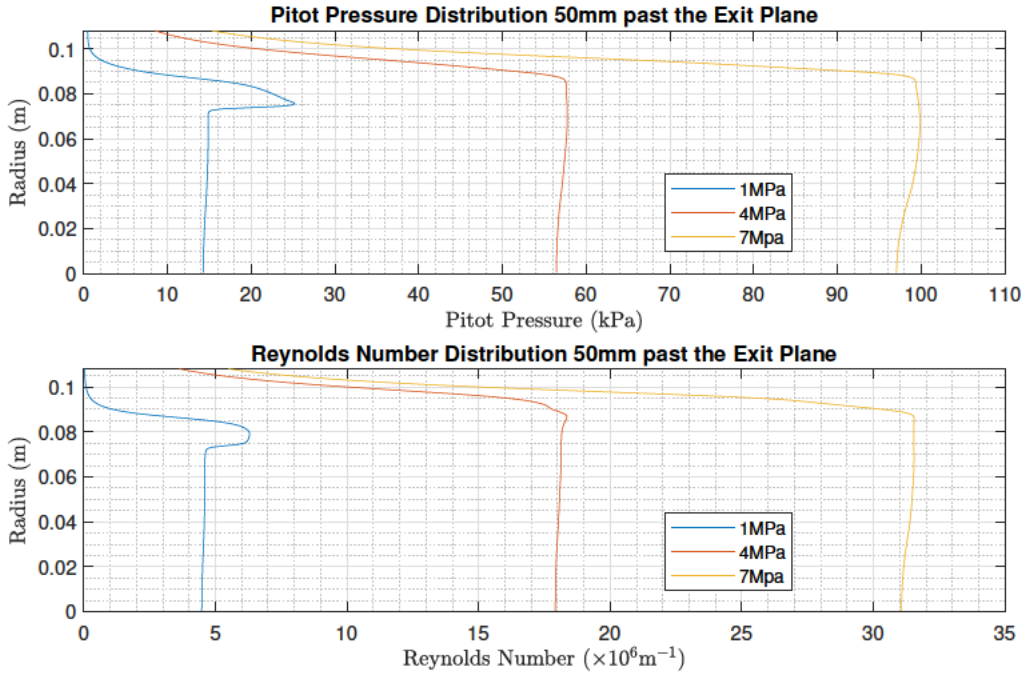


Figure 6.12: Pitot pressure and Reynolds number distribution at 50mm past the nozzle exit.

6.4 Flow Conditions

Table 6.4 and Table 6.5 show the average core flow conditions produced by each stagnation condition for the Mach 6 and Mach 7 nozzles respectively. As Reynolds number is often used as a scaling factor for experimental models, it can be of importance in determining the size of the model. The Reynolds number ranges for the Mach 6 and Mach 7 nozzles based on the current range of simulated stagnation conditions are $3.51\times10^6 < Re_{M6} < 48.2\times10^6$ and $4.59 \times 10^6 < Re_{M7} < 31.52 \times 10^6$ respectively.

Table 6.4: Mean core flow parameters at the nozzle exit for each stagnation condition for the Mach 6 nozzle.

Property	Unit	$P_o = 1$MPa $T_o = 575$K	$P_o = 1$MPa $T_o = 900$K	$P_o = 4$MPa $T_o = 575$K	$P_o = 7$MPa $T_o = 575$K
M	-	5.89	5.88	5.91	5.92
P_{pitot}	kPa	31.94	32.12	126.16	218.84
P	Pa	708.73	713.78	2778.84	4798.69
T	K	72.42	113.32	72.00	71.72
Re	$\times10^6$ m^{-1}	6.97	3.51	27.70	48.24

Table 6.5: Mean core flow parameters at the nozzle exit for each stagnation condition for the Mach 7 nozzle.

Property	Unit	$P_o = 1$MPa $T_o = 575$K	$P_o = 4$MPa $T_o = 575$K	$P_o = 7$MPa $T_o = 575$K
M	-	7.06	7.10	7.12
P_{pitot}	kPa	14.74	57.54	99.51
P	Pa	227.87	879.16	1513.34
T	K	52.28	51.70	51.54
Re	$\times10^6$ m^{-1}	4.59	18.17	31.52

6.5 Chapter Summary

Validation of the Mach 6 nozzle simulation was performed and it was found that the flow parameters at the nozzle exit differ by less than 1% compared to the nominal conditions presented by Birch (2019). It was found that changing both the stagnation pressure and temperature directly affected the flow parameters in the nozzle core flow. By increasing the stagnation pressure and thus the pressure ratio, the flow became under-expanded. The Reynolds number, pitot pressure and static pressure increased in proportion to the stagnation pressure. Increasing the stagnation temperature had the opposite effect, with the Reynolds number, pitot pressure, and static pressure decreasing in proportion to the change in stagnation temperature. Increasing the stagnation temperature also increased the size of the usable area at the nozzle exit by shifting the location of the beginning of the Mach reflection further into the test section.

The simulation results showed that there was a direct correlation between stagnation pressure and temperature and the flow parameters in the test section. The flow Reynolds number (which is often used for scaling) can be increased or decreased by changing the stagnation pressure and temperature without affecting the flow Mach number.

Chapter 7

Conclusions and Further Work

7.1 Conclusions

Several 2D axisymmetric simulations of the University of Southern Queensland Hypersonic Testing Facility (TUSQ) were produced. These simulations were run using the compressible flow solver `Eilmer4` and have been validated against historical experimental and theoretical data. The purpose of running multiple simulations was to determine the variation of the flow parameters in TUSQ and how they were affected by changing the nozzle stagnation conditions.

It was found that increasing the stagnation pressure produced under-expanded flow for both the Mach 6 and Mach 7 nozzles, but did not greatly affect the usable area in the test section. The Reynolds number, pitot pressure, and static pressure were found to increase proportionally to the stagnation pressure. Increasing the stagnation temperature was found to increase the size of the usable area in the test section, as well as decrease the Reynolds Number and pitot pressure proportionally.

For the Mach 6 nozzle, the nominal stagnation conditions (of $P_o = 1$MPa and $T_o = 575$ K) produced a slightly under-expanded Mach 5.89 flow with a Reynolds number of 6.94×10^6 m^{-1} and pitot pressure of 31.8 kPa. The flow Mach number remained almost unchanged for the other stagnation conditions. Stagnation conditions of 4MPa at 575K, and 7MPa at 575K subsequently increased the Reynolds number and pitot pressure in proportion to the increase in stagnation pressure. The remaining stagnation condition of 1MPa at 900K

was found to decrease the Reynolds number and pitot pressure, with the change being inversely proportional to the increase in temperature. This condition was also found to increase the size of the usable area in the test section. The achievable range of Reynolds number was 3.5×10^6 m^{-1} to 48.2×10^6 m^{-1} and the range of pitot pressure was 32.1 kPa to 218.8 kPa.

For the Mach 7 nozzle, the nominal conditions produced an over-expanded Mach 7.05 outflow with an oblique shock formed at the nozzle exit. Like with the Mach 6 nozzle, the Mach number remained unchanged for the other stagnation conditions. The range of Reynolds number was 4.6×10^6 m^{-1} to 31.5×10^6 m^{-1} and the range of pitot pressure was 14.7 kPa to 99.5 kPa. The flow became under-expanded at higher stagnation pressures, with size of the usable area remaining largely the same.

7.2 Further Work

Future work on characterisation of the TUSQ freestream may follow on from this study. Possible areas of interest have been identified as:

- Characterisation of TUSQ's Mach 4.5 nozzle
- Transient analyses of TUSQ

References

Anderson, J. D. (2020), *Modern Compressible Flow with Historical Perspective*, McGraw-hill Education.

ANSYS Inc. (2010), *ANSYS Meshing User's Guide*, ANSYS Inc.

Bakker, A. (2006), 'Applied computational fluid dynamics', `http://www.bakker.org/dartmouth06/engs150/07-mesh.pdf`. [Online; accessed May-2020].

Birch, B. J. C. (2019), Characterisation of the USQ Hypersonic Facility Freestream, PhD thesis, University of Southern Queensland, Toowoomba, Queensland.

Birch, B. J. C., Choudhury, R., Stern, N. & Buttsworth, D. R. (n.d.), 'Pressure fluctuations in a hypersonic ludwieg tube with free piston compression heating'.

Bono, G., Awruch, A. M. & Popiolek, T. L. (2008), 'Computational study of laminar shock/boundary layer interaction at hypersonic speeds', *Mecánica Computacional* **27**, 3135–3150.

Buttsworth, D. R. (2009), Ludwieg tunnel facility with free piston compression heating for supersonic and hypersonic testing, *in* W. Short & I. Cairns, eds, 'Proceedings of the 9th Australian Space Science Conference Sydney 28 - 30 September, 2009', National Space Society of Australia Ltd, pp. 153–162.

Buttsworth, D. R. & Smart, M. K. (2010), Development of a ludwieg tube with free piston compression heating for scramjet inlet starting experiments, *in* '48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition 4 - 7 January 2010, Orlando, Florida', American Institute of Aeronautics and Astronautics, Inc.

Central Aerohydrodynamic Institute (2020), 'Viscid oil method. visualization of surface

streamlines and shear stress', http://www.tsagi.ru/en/research/measurements/oil/. [Online; accessed April-2020].

CFD Support (n.d.), 'Mesh quality controls', https://www.cfdsupport.com/OpenFOAM-Training-by-CFD-Support/node129.html. [Online; accessed June-2020].

Chan, W. Y. K., Jacobs, P. A. & Mee, D. J. (2011), Suitability of the k–ω turbulence model for scramjet flowfield simulations, *in* 'International Journal For Numerical Methods in Fluids', John Wiley & Sons, Ltd.

Chung, T. J. (2002), *Computational Fluid Dynamics*, Cambridge University Press.

Devenport, W. J. (2001), 'Converging diverging nozzle', http://www.engapplets.vt.edu/fluids/CDnozzle/cdinfo.html. [Online; accessed April-2020].

Doherty, L. (2014), An Experimental Investigation of an Airframe Integrated Three-Dimensional Scramjet Engine at a Mach 10 Flight Condition, PhD thesis, The University of Queensland, Brisbane, Queensland.

Freedman, E. & Greene, E. F. (n.d.), '2v. shock waves', https://web.mit.edu/8.13/8.13c/references-fall/aip/aip-handbook-section2v.pdf. [Online; accessed April-2020].

Jacobs, P. A. & Gollan, R. J. (2013), 'About the formulation, verification and validation of the hypersonic flow solver eilmer', *International Journal for Numerical Methods in Fluids* **73**(1).

Jacobs, P. A. & Gollan, R. J. (2018), 'Introduction to cfd for hypersonic flows with eilmer'.

Jacobs, P. A. & Gollan, R. J. (2020*a*), 'About - eilmer', http://cfcfd.mechmining.uq.edu.au/eilmer/about/. [Online; accessed April-2020].

Jacobs, P. A. & Gollan, R. J. (2020*b*), 'Eilmer reference manual for users, v4.0', http://cfcfd.mechmining.uq.edu.au/html/eilmer-reference-manual.html. [Online; accessed July-2020].

Jacobs, P. A. & Gollan, R. J. (2020*c*), 'Guide to the transient flow solver', http://cfcfd.mechmining.uq.edu.au/pdfs/eilmer-user-guide.pdf. [Online; accessed October-2020].

Jacobs, P. A., Gollan, R. J., Denman, A. J., O'Flaherty, B. T., Potter, D. F., Petrie-Repar, P. J. & Johnston., I. A. (2012), 'Eilmer's theory book: Basic models for gas dynamics and thermochemistry.'.

Jacobs, P., Gollan, R. & Damm, K. (2019), 'Progress of the eilmer 4 transient flow solvers', `http://cfcfd.mechmining.uq.edu.au/pdfs/eilmer-talk-pj-aug-2019.pdf`. [Online; accessed April-2020].

Jameson, A. (1990), 'Full-potential, euler, and navier-stokes schemes', **125**, 42–44.

Jéger, C. & Veress, A. (2017), 'Novell application of cfd for rocket engine nozzle optimization', *Periodica Polytechnica Transportation Engineering* **47**, 131–135.

Johl, G., Passmore, M. & Render, P. (2004), 'Design methodology and performance of an indraft wind tunnel', pp. 465–473.

Kitware (n.d.), 'File formats for vtk version 4.2', `https://vtk.org/wp-content/uploads/2015/04/file-formats.pdf`. [Online; accessed April-2020].

Lewington, N. (2005), 'Aerodynamics fundamentals for automotive', `http://www.saea.com.au/resources/Documents/aero101-SAE-A-Seminar.pdf`. [Online; accessed April-2020].

Martinez, I. (2020), 'Nozzles', `http://webserver.dmt.upm.es/~isidoro/bk3/c17/Nozzles.pdf`. [Online; accessed April-2020].

Mousavi, S. M. & Roohi, E. (2014), 'Three dimensional investigation of the shock train structure in a convergent–divergent nozzle', *Acta Astronautica* **105**.

Narayana, K. S. & Reddy, K. S. (2016), 'Simulation of convergent divergent rocket nozzle using cfd analysis', *IOSR Journal of Mechanical and Civil Engineering (IOSR-JMCE)* **13**, 58–65.

NASA (2015*a*), 'Blowdown wind tunnel', `https://www.grc.nasa.gov/WWW/K-12/airplane/tunblow.html`. [Online; accessed April-2020].

NASA (2015*b*), 'Drag measurement', `https://www.grc.nasa.gov/www/k-12/airplane/dragdat.html`. [Online; accessed April-2020].

NASA (2015*c*), 'Euler equations', `https://www.grc.nasa.gov/www/k-12/airplane/eulereqs.html`. [Online; accessed April-2020].

NASA (2015*d*), 'Schlieren flow visualisation', https://www.grc.nasa.gov/www/k-12/airplane/tunvschlrn.html. [Online; accessed April-2020].

NASA (2015*e*), 'Types of wind tunnels', https://www.grc.nasa.gov/www/k-12/airplane/tuntype.html. [Online; accessed April-2020].

NASA (2015*f*), 'Wind tunnel testing', https://www.grc.nasa.gov/www/k-12/airplane/tuntest.html. [Online; accessed April-2020].

NASA (2018*a*), 'Normal shock wave', https://www.grc.nasa.gov/www/k-12/airplane/normal.html. [Online; accessed April-2020].

NASA (2018*b*), 'Similarity parameters', https://www.grc.nasa.gov/WWW/K-12/airplane/airsim.html. [Online; accessed April-2020].

Nguyen, C. (2005), 'Turbulence modeling', http://www.mit.edu/~cuongng/Site/Publication_files/TurbulenceModeling_04NOV05.pdf. [Online; accessed April-2020].

Nichols, R. H. & Nelson, C. C. (2004), 'Wall function boundary conditions including heat transfer and compressibility', **42**, 1107–1113.

of Queensland, U. (2018), 'Centre for hypersonics', http://hypersonics.mechmining.uq.edu.au/. [Online; accessed April-2020].

Rhoads, J. (2014), 'How do you define a good grid?', https://www.cfdsupport.com/OpenFOAM-Training-by-CFD-Support/node129.html. [Online; accessed June-2020].

Robinson, M. J., Rowan, S. A. & Odam, J. (2015), *T4 Free Piston Shock Tunnel Operator's Manual Version 2.0*, The University of Queensland Centre for Hypersonics.

Schmidt, B. (2015), 'Schlieren visualization', https://shepherd.caltech.edu/T5/Ae104/Ae104b_handout2015.pdf. [Online; accessed April-2020].

Sura, J. (2017), 'Cfd simulation of hypersonic shock tunnel nozzle'.

The University of Queensland (2016), 'T4 - free-piston driven shock tunnel', http://hypersonics.mechmining.uq.edu.au/t4. [Online; accessed June-2020].

UNSW (2020), 'Hypersonics | school of engineering and information technology |', https://www.unsw.adfa.edu.

au/school-of-engineering-and-information-technology/school-research-themes/hypersonics. [Online; accessed April-2020].

Wandel, A. (2020), 'Mec5100 study guide', http://usqstudydesk.usq.edu.au. [Online; accessed May-2020].

White, F. M. (2003), *Fluid Mechanics*, McGraw-Hill.

Appendix A

Project Specification

ENG 4111/4112 Research Project

Project Specification

For:	**Alister Webb**
Topic:	Hypersonic Nozzle Characterisation Using CFD
Supervisors:	Dr. Fabian Zander
Sponsorship:	Faculty of Health, Engineering & Sciences
Project Aim:	To create steady-state flow simulations of the standard hypersonic nozzles used in the TUSQ facility (Mach 4.5, 6, and 7) and undertake a high fidelity investigation into the effects of changing stagnation conditions on the test flow.

Program:

1. Review existing literature concerning hypersonic flow and Ludwieg tubes.
2. Install and learn to use the Eilmer CFD program.
3. Create and simulate a steady-state model of the TUSQ facility with a Mach 6 nozzle.
4. Use historical experimental data from the TUSQ facility to validate the model results.
5. Create and simulate models for other nozzle geometries (Mach 4.5 and 7).

As time and resources permit:

1. Extend the simulation region to encompass the whole test section to determine how its size and geometry affects the flow.
2. Perform transient analyses on the nozzle and test section.

Agreed:

Student Name:	Alister Webb
Date:	10/03/2020
Supervisor Name:	Dr Fabian Zander
Date:	10/03/2020

Appendix B

Mach 6 nozzle `Eilmer4` Input Script

B.1 Purpose and Function

This script is used to define and run the simulations in `Eilmer4`. It sets all required configuration variables and defines the flow conditions, as well as the geometry and mesh. This script was used as the basis for the Mach 7 nozzle simulations, with only the contour and a few nozzle specific definitions being changed.

B.2 Script

```
--  Alister Webb
--  10/04/2020
--  Eilmer4 input script for the TUSQ Mach 6 nozzle. Nozzle contour is
--  read from an external file.
--      Requires:
--          -- M6cont.dat
--          -- ideal-air-gas-model.lua
--  ########################################################################
--  Initialisation and Housekeeping
--  ########################################################################
-- For multi-stage simulations, the old flow solution is read in here:
--[[
fsolEnd = FlowSolution:new{jobName="Mach-6", dir="../STAGE_2", tindx=10,
                           nBlocks=139}
initial = makeFlowStateFn(fsolEnd)
--]]
-- Now the standards
config.title = "TUSQ Mach 6 Nozzle, Turbulence and Viscous effects."
print(config.title)
config.dimensions = 2
config.axisymmetric = true
config.viscous = true
config.turbulence_model = "k_omega"
-- Gas model
nsp, nmodes, gm = setGasModel('ideal-air-gas-model.lua')
print("GasModel set to ideal air. Number of species =", nsp,
      "Number of modes=", nmodes)
stagCond = FlowState:new{p=1.0e6, T=575.0, velx=0.0, tke=0.5, omega=5}
--  ########################################################################
--  Geometry
--  ########################################################################
```

```
--  Import Geometry from Space Separated .dat file
importFileName = "M6cont.dat"
contour = Spline2:new{filename=importFileName}
--  ------------------------------------------------------------------------
--  Translate contour into x+ and Create Required Geometry
--  ------------------------------------------------------------------------
cylinderLength = 0.13 -- Offset the nozzle contour into x+
cylinderDiameter = 0.065
shift = Vector3:new{x = (cylinderLength - contour(0).x), y = 0} -- This is
-- the translation distance (in positive x) to move the entire contour
-- (cylinder included) into x+
contour = ArcLengthParameterizedPath:new{underlying_path = contour}
-- Re-distribute the points so they're now evenly spaced
contour = TranslatedPath:new{original_path = contour, shift = shift}
-- Translate the path to the right into positive coordinates.
print('Contour import and alignment... Done.')
--  ------------------------------------------------------------------------
--  Diagram. Blocks and Nodes
--  ------------------------------------------------------------------------
-- number = node, * = wall, - = | =\ = block boundary
--
--                                          19******************23
--   4*********9                            *                    |
--   | Block 3 *                            *      Block 12      |
--   3---------7********12********15********18--------------------22
--   | Block 2 | Block 5 | Block 7 | Block 9 |      Block 11      |
--   2---------6--------11--------14--------17--------------------21
--   | Block 1 | Block 4 | Block 6 | Block 8 |      Block 10      |
--   1---------5--------10--------13--------16--------------------20
--
--   | Driver  | Throat  | Diverging Section |    Test Section    |
--
-- For a more detailed drawing of the flow domain, see
-- Section 4.4 - Flow domain and Geometry
-- More detailed inlet section. Point 8 is located between points 7 and 9
-- and is the beginning of the nozzle contour:
--
--      4*******************************9
--      |                               *
--      |                               *
--      |           Block 3             *
--      |                               8 Contour Start
--      |                                *
```

```
--      3---------------------------------7*****          *
--      |           Block 2               /       **12*****
--      |                                 | Block 5 |
--      2---------------------------------6---------11------
--      |           Block 1               | Block 4 |
--      1---------------------------------5---------10------
--
-- Blocks 1, 2, and 3 make up the cylinder/driver section.
-- Blocks 4 and 5 are the throat blocks.
-- Blocks 6, 7, 8, and 9 are the nozzle blocks
-- Blocks 10, 11, and 12 are the test section blocks.
-- Blocks 5, 7, and 9 are the in the boundary layer (and so are 2 and 11).
--  ------------------------------------------------------------------------
--  Finder function for a required x coordinate - for the path 'contour'
--  ------------------------------------------------------------------------
function findContourX(x)
    -- Will return the value of t (where 0<t<1) for a given x coordinate
    -- of a path object.
    -- Set some local variables
    local error = 1e-7
    local iteration_Max = 100
    local counter = 0
    local min = 0
    local max = 1
    local mean = (min+max)/2
    -- While loop to calculate the value of t
    while math.abs((x-contour(mean).x)/x)>error do
        counter = counter+1 -- A counter to prevent an infinite while loop
        if counter >= iteration_Max then
            print('Maximum iteration reached')
            break -- Error message
        elseif x>contour(1).x then
            print(
            'X coordinate not present in path object. Defaulted to t=0.5')
            break -- Error message
        elseif x<contour(0).x then
            print(
            'X coordinate not present in path object. Defaulted to t=0.5')
            break -- Error message
        elseif contour(mean).x>x then -- Set upper range to the mean value
            max = mean
        elseif contour(mean).x<x then -- Set lower range to the mean value
            min = mean
```

```
        end
        mean = (min+max)/2 -- Recalculate the mean value for the next loop
    end
    return (mean) -- return a single numerical value.
 end

-- Sub-range the contour into two parts either side of the narrowest point
--  of the throat and a small section to be used as part of the cylinder
-- wall and also parameterize each.
throatCenter = 0.1967 -- This distance (mm) is different for each contour:
-- M6 = 0.1967m, M7 = 0.1755m
throatCentreLength = findContourX(throatCenter) -- t value
leftSplitLength = findContourX(0.132) -- t value
-- 132mm is past the cylinder wall and creates an intersection point that
-- minimises skewness.
contourLeft = ArcLengthParameterizedPath:new{underlying_path =
    ReversedPath:new{underlying_path = SubRangedPath:new{underlying_path =
        contour, t0 = 0, t1 = leftSplitLength}}}
throatContour = ArcLengthParameterizedPath:new{underlying_path =
     SubRangedPath:new{underlying_path = contour, t0 = leftSplitLength,
        t1 = throatCentreLength}}
divergingContour = ArcLengthParameterizedPath:new{underlying_path =
    SubRangedPath:new{underlying_path = contour, t0 = throatCentreLength,
        t1 = 1}}
--  ------------------------------------------------------------------------
--  Diverging section Boundary layer path function
--  ------------------------------------------------------------------------
L0 = 0.0035 -- Throat area b.l. thickness
L1 = 0.02 -- Nozzle exit b.l. thickness
function BLContour(t)
    F = divergingContour(t) -- Duplicate the nozzle contour
    dt = 0.001 -- set the increment
    if t < dt then
        -- use the forward difference
       left = t; right = t+dt
    elseif t > 1 - dt then
        -- use the backward difference
       left = t-dt; right = t
    else
        -- otherwise use a point before and after the specified value
       left = t-dt; right = t+dt
    end
    -- Calculate the value at each point and find the gradient
```

```
    rightPoint = divergingContour(right)
    leftPoint = divergingContour(left)
    dx = rightPoint.x-leftPoint.x
    dy = rightPoint.y-leftPoint.y
    -- calculate the normal by swapping the components. The new path will
    -- be offset in negative y (down from the original) hence '-dy'. May
    -- not work with all geometry.
    X = -dy; Y = dx
    -- Calculate the magnitude of the normal and divide the x and y
    -- components to create a unit vector.
    -- Alternatively create a vector and use ':normalise()' after calling
    -- it.
    X = X / (dx^2 + dy^2)^0.5; Y = Y / (dx^2 + dy^2)^0.5;
    -- Calculate the boundary layer thickness based on the two values
    -- specified above.
    L = L0 + t*(L1-L0)
    -- Find the coordinates of the point on the offset boundary layer path
    -- and return those as the outputs in a table
    xval = F.x - X*L
    yval = F.y - Y*L
    return {x=xval, y=yval}
 end
offsetBLContour = ArcLengthParameterizedPath:new{underlying_path =
    LuaFnPath:new{luaFnName="BLContour"}}
--  ------------------------------------------------------------------------
--  Finder function for a required x coordinate - for the path 'BLContour'
--  ------------------------------------------------------------------------
function findBLContourX(x)
    -- Same as above (save for the path object), see comments there.
   local error = 1e-7
   local iteration_Max = 100
   local counter = 0
   local min = 0
   local max = 1
   local mean = (min+max)/2
   while math.abs((x-BLContour(mean).x)/x)>error do
       counter = counter+1
       if counter >= iteration_Max then
           print('Maximum iteration reached')
           break
       elseif x>BLContour(1).x then
           print(
            'X coordinate not present in path object. Defaulted to t=0.5')
```

```
            break
        elseif x<BLContour(0).x then
            print(
            'X coordinate not present in path object. Defaulted to t=0.5')
            break
        elseif BLContour(mean).x>x then
            max = mean
        elseif BLContour(mean).x<x then
            min = mean
        end
        mean = (min+max)/2
    end
    return (mean)
end
--  ------------------------------------------------------------------------
--  Throat Boundary layer path function
--  ------------------------------------------------------------------------
L2 = 0.008 -- Inlet side area b.l. thickness -- M6 = 8mm, M7 = 5.5mm
L3 = L0 -- Throat b.l. thickness - Same as L0 above
-- Same as offset path function above
function smallBLContour(t)
    F = throatContour(t)
    dt = 0.001
    if t < dt then
       left = t; right = t+dt
    elseif t > 1 - dt then
       left = t-dt; right = t
    else
       left = t-dt; right = t+dt
    end
    rightPoint = throatContour(right)
    leftPoint = throatContour(left)
    dx = rightPoint.x-leftPoint.x
    dy = rightPoint.y-leftPoint.y
    X = -dy; Y = dx
    X = X / (dx^2 + dy^2)^0.5; Y = Y / (dx^2 + dy^2)^0.5;
    L = L2 + t*(L3-L2)
    xval = F.x - X*L
    yval = F.y - Y*L
    return {x=xval, y=yval}
 end
throatBLContour = ArcLengthParameterizedPath:new{underlying_path =
    LuaFnPath:new{luaFnName="smallBLContour"}}
```

```
-- Sub-range the path so that the curve does not end perpendicular to a
-- block boundary
throatBLContour = SubRangedPath:new{underlying_path =
    throatBLContour, t0 = 0.07, t1 = 1}
--  ------------------------------------------------------------------------
--  Lines and points for the boundaries
--  ------------------------------------------------------------------------
-- Parameters
NozzleBLSplit = findBLContourX(0.30) -- Corresponds with x=400mm,
-- the separation point for the blocks in the diverging section.
NozzleContSplit = findContourX(0.30) -- Also x=400mm
-- -- Coordinates for the BL separation point, also the end of the nozzle
contourSplitLeft = contourLeft(0)
nozEnd = contour(1)
blEnd = offsetBLContour(1)
-- -- Measurements
testSectionL = 830/1000  -- test section length
testSectionR = 300/1000   -- test section radius
-- -- Corner points/nodes for each block, listed from bottom to top and
-- left to right as seen in the ASCII drawing above.
p1 = Vector3:new{x = 0, y = 0}
p2 = Vector3:new{x = 0, y = throatBLContour(0).y}
p3 = Vector3:new{x = 0, y = contourSplitLeft.y}
p4 = Vector3:new{x = 0, y = cylinderDiameter}
p5 = Vector3:new{x = 0.128, y = 0} -- originally 127mm
p6 = throatBLContour(0)--Vector3:new{x = 0.128, y = .y}
p7 = contourSplitLeft
p8 = Vector3:new{x = cylinderLength, y = contour(0).y}
p9 = Vector3:new{x = cylinderLength, y = cylinderDiameter}
p10 = Vector3:new{x = BLContour(0).x, y = 0}
p11 = offsetBLContour(0)
p12 = divergingContour(0)
p13 = Vector3:new{x = 0.30, y=0}
p14 = offsetBLContour(NozzleBLSplit)
p15 = contour(NozzleContSplit)
p16 = Vector3:new{x = nozEnd.x, y = 0}
p17 = blEnd
p18 = nozEnd
p19 = Vector3:new{x = nozEnd.x, y = testSectionR}
p20 = Vector3:new{x = (nozEnd.x+testSectionL), y = 0}
p21 = Vector3:new{x = (nozEnd.x+testSectionL), y = blEnd.y}
p22 = Vector3:new{x = (nozEnd.x+testSectionL), y = nozEnd.y}
p23 = Vector3:new{x = (nozEnd.x+testSectionL), y = testSectionR}
```

```
-- Lines for block boundaries and interfaces - Naming convention for
-- interfaces is 'bxbyInter', where x is the first block and y is the
-- second block. x and y are always consecutive e.g. 'b1b2Inter' being
-- block 1 and 2 interface, not 'b2b1Inter' or similar.
-- -- Inlet faces
inletBottom = Line:new{p0 = p1, p1 = p2}
inletMid = Line:new{p0 = p2, p1 = p3}
inletTop = Line:new{p0 = p3, p1 = p4}
-- -- Horizontal lines for blocks 1-3 - includes interfaces
axisLeft = Line:new{p0 = p1, p1 = p5}
b1b2Inter = Line:new{p0 = p2, p1 = p6}
b2b3Inter = Line:new{p0 = p3, p1 = p7}
cylinderTop = Line:new{p0 = p4, p1 = p9}
-- -- Vertical cylinder and nozzle separation
b1b4Inter = Line:new{p0 = p5, p1 = p6}
b2b5Inter = Line:new{p0 = p6, p1 = p7}
cylinderRight = Polyline:new{segments =
                            {contourLeft, Line:new{p0 = p8, p1 = p9}}}
-- This is a composite path due to the end of the contour being part of
-- the cylinder end.
-- -- Horizontal lines for throat
axisThroat = Line:new{p0 = p5, p1 = p10}
--b4b5Inter = throatBLContour--Line:new{p0 = p6, p1 = p11}
b4b5Inter = ArcLengthParameterizedPath:new{underlying_path =
     Bezier:new{points =
        {p6, Vector3:new{x = (0.95*(p11.x+p6.x)/2), y = p11.y}, p11}}}
-- wallThroat = throatContour
-- -- Mid nozzle separation
b4b6Inter = Line:new{p0 = p10, p1 = p11}
b5b7Inter = Line:new{p0 = p11, p1 = p12}
-- -- Diverging section horizontal lines
axisDivergingL = Line:new{p0 = p10, p1 = p13}
b6b7Inter = ArcLengthParameterizedPath:new{underlying_path =
SubRangedPath:new{underlying_path = offsetBLContour, t0 = 0,
                  t1 = NozzleBLSplit}}
b8b9Inter = ArcLengthParameterizedPath:new{underlying_path =
SubRangedPath:new{underlying_path = offsetBLContour, t0 = NozzleBLSplit,
                  t1 = 1}}
b7Cont = ArcLengthParameterizedPath:new{underlying_path =
SubRangedPath:new{underlying_path = contour, t0 = throatCentreLength,
                  t1 = NozzleContSplit}}
b9Cont = ArcLengthParameterizedPath:new{underlying_path =
SubRangedPath:new{underlying_path = contour, t0 = NozzleContSplit,
```

```
                    t1 = 1}}
axisDivergingR = Line:new{p0 = p13, p1 = p16}
b6b8Inter = Line:new{p0 = p13, p1 = p14}
b7b9Inter = Line:new{p0 = p14, p1 = p15}
-- -- Test section and nozzle interfaces
b8b10Inter = Line:new{p0 = p16, p1 = p17}
b9b11Inter = Line:new{p0 = p17, p1 = p18}
testSectionLeft = Line:new{p0 = p18, p1 = p19}
-- -- Horizontal test section interfaces
axisTestSec = Line:new{p0 = p16, p1 = p20}
b10b11Inter = Line:new{p0 = p17, p1 = p21}
b11b12Inter = Line:new{p0 = p18, p1 = p22}
testSectionTop = Line:new{p0 = p19, p1 = p23}
-- -- Outlets
outletBottom = Line:new{p0 = p20, p1 = p21}
outletMid = Line:new{p0 = p21, p1 = p22}
outletTop = Line:new{p0 = p22, p1 = p23}
--
print('Points and lines... Done.')
--  ------------------------------------------------------------------------
--  Quads - numbered according to blocks in diagram - Starts at 1
--  ------------------------------------------------------------------------
quad1 = makePatch{north = b1b2Inter, east = b1b4Inter, south = axisLeft,
                    west = inletBottom}
quad2 = makePatch{north = b2b3Inter, east = b2b5Inter, south = b1b2Inter,
                    west = inletMid}
quad3 = makePatch{north = cylinderTop, east = cylinderRight,
                    south = b2b3Inter, west = inletTop}
quad4 = makePatch{north = b4b5Inter, east = b4b6Inter, south = axisThroat,
                    west = b1b4Inter}
quad5 = makePatch{north = throatContour, east = b5b7Inter,
                    south = b4b5Inter, west = b2b5Inter}
quad6 = makePatch{north = b6b7Inter, east = b6b8Inter,
                    south = axisDivergingL , west = b4b6Inter}
quad7 = makePatch{north = b7Cont, east = b7b9Inter, south = b6b7Inter,
                    west = b5b7Inter}
quad8 = makePatch{north = b8b9Inter, east = b8b10Inter,
                    south =axisDivergingR , west = b6b8Inter}
quad9 = makePatch{north = b9Cont, east = b9b11Inter, south = b8b9Inter ,
                    west = b7b9Inter}
quad10 = makePatch{north = b10b11Inter, east = outletBottom,
                    south =axisTestSec , west = b8b10Inter}
quad11 = makePatch{north = b11b12Inter, east = outletMid,
```

```
                    south =b10b11Inter , west = b9b11Inter}
quad12 = makePatch{north = testSectionTop, east = outletTop,
                    south =b11b12Inter , west = testSectionLeft}
print('Quads... Done.')
--  ------------------------------------------------------------------------
--  Grids and Meshing
--  ------------------------------------------------------------------------
-- Mesh size and refine val
refine = 3.0
-- Number of cells in each section, starting at ~1cell/mm, Floor function
-- allows non-integer refine factors by 'rounding' down to the lowest
-- integer.
--block1
nx1 = math.floor(125*refine); ny1 = math.floor(40*refine) -- core region
--block2
nx2 = nx1; ny2 = math.floor(20*refine) -- boundary layer
--block3
nx3 = nx1; ny3 = math.floor(55*refine)
--block4
nx4 = math.floor(60*refine); ny4 = ny1
--block5
nx5 = nx4; ny5 = ny2
--block6
nx6 = math.floor(120*refine); ny6 = ny1
--block7
nx7 = nx6; ny7 = ny5
--block 8
nx8 = math.floor(900*refine); ny8 = ny1
--block9
nx9 = nx8; ny9 = ny2
--block10
nx10 = math.floor(830*refine); ny10 = ny1
--block11
nx11 = nx10; ny11 = ny2
--block12
nx12 = nx10; ny12 = math.floor(120*refine) -- test section
-- Calculate the cells in each block
ncells1 = nx1*ny1
ncells2 = nx2*ny2
ncells3 = nx3*ny3
ncells4 = nx4*ny4
ncells5 = nx5*ny5
ncells6 = nx6*ny6
```

```
ncells7 = nx7*ny7
ncells8 = nx8*ny8
ncells9 = nx9*ny9
ncells10 = nx10*ny10
ncells11 = nx11*ny11
ncells12 = nx12*ny12
-- Calculate and print the total number of cells.
ncells = ncells1+ncells2+ncells3+ncells4+ncells5+ncells6+ncells7+ncells8+
         ncells9+ncells10+ncells11+ncells12
print("Cell sizes defined. Total number of cells =", ncells)
--  ------------------------------------------------------------------------
--  Clustering Functions
--  ------------------------------------------------------------------------
-- -- Boundary layer
BLCluster = RobertsFunction:new{end0 = false, end1 = true, beta = 1.25}
-- -- Core flow
coreClustL = RobertsFunction:new{end0 = true, end1 = false, beta = 1.3}
coreClustMid = RobertsFunction:new{end0 = false, end1 = true, beta = 1.15}
coreClustR = RobertsFunction:new{end0 = false, end1 = true, beta = 1.35}
-- -- Cylinder, M6 = 1.25, M7 = 1.1
topCylClust = RobertsFunction:new{end0 = true, end1 = false, beta = 1.25}
lowerCylClust = RobertsFunction:new{end0 = true, end1 = false, beta = 1.2}
-- Test Section Upper
testSecClust = RobertsFunction:new{end0 = true, end1 = false, beta = 1.15}
--  ------------------------------------------------------------------------
-- Grids
--  ------------------------------------------------------------------------
grid1 = StructuredGrid:new{psurface = quad1, niv = nx1+1, njv = ny1+1}
grid2 = StructuredGrid:new{psurface = quad2, niv = nx2+1, njv = ny2+1}
grid3 = StructuredGrid:new{psurface = quad3, niv = nx3+1, njv = ny3+1,
    cfList = {west = topCylClust, east = topCylClust}}
grid4 = StructuredGrid:new{psurface = quad4, niv = nx4+1, njv = ny4+1,
    cfList = {east = coreClustL}}
grid5 = StructuredGrid:new{psurface = quad5, niv = nx5+1, njv = ny5+1,
    cfList = {east = BLCluster}}
grid6 = StructuredGrid:new{psurface = quad6, niv = nx6+1, njv = ny6+1,
    cfList = {east = coreClustMid, west = coreClustL}}
grid7 = StructuredGrid:new{psurface = quad7, niv = nx7+1, njv = ny7+1,
    cfList = {west = BLCluster, east = BLCluster}}
grid8 = StructuredGrid:new{psurface = quad8, niv = nx8+1, njv = ny8+1,
    cfList = {east = coreClustR, west = coreClustMid}}
grid9 = StructuredGrid:new{psurface = quad9, niv = nx9+1, njv = ny9+1,
    cfList = {west = BLCluster, east = BLCluster}}
```

```
grid10 = StructuredGrid:new{psurface = quad10, niv = nx10+1, njv = ny10+1,
    cfList = {west = coreClustR, east = coreClustR}}
grid11 = StructuredGrid:new{psurface = quad11, niv = nx11+1, njv = ny11+1,
    cfList = {west = BLCluster, east = BLCluster}}
grid12 = StructuredGrid:new{psurface = quad12, niv = nx12+1, njv = ny12+1,
    cfList = {west = testSecClust, east = testSecClust}}
print('Grid... Done.')
--  ------------------------------------------------------------------------
--  Blocks and sub-blocks
--  ------------------------------------------------------------------------
-- Define the number of sub-blocks.
nib1 = 4; njb1 = 2          -- inlet block - bottom of cylinder
nib2 = nib1; njb2 = 1       -- inlet block - middle of cylinder
nib3 = nib1; njb3 = 1       -- upper inlet block
nib4 = 2; njb4 = njb1       -- core flow block in throat
nib5 = nib4; njb5 = njb2    -- bl block in throat
nib6 = 3; njb6 = njb1       -- leftmost core block in diverging section
nib7 = nib6; njb7 = njb2    -- leftmost BL block in diverging section
nib8 = 12; njb8 = njb1      -- rightmost core block in diverging section
nib9 = nib8; njb9 = njb2    -- rightmost bl block in diverging section
nib10 = 12; njb10 = njb1    -- core flow block in test section
nib11 = nib10; njb11 = njb2 -- bl block in test section - middle one
nib12 = nib10; njb12 = 3    -- upper test section block
--
blk1 = FluidBlockArray{grid = grid1, initialState = initial, nib = nib1,
    njb = njb1, bcList = {west =
        InFlowBC_FromStagnation:new{stagnationState=stagCond}}}
blk2 = FluidBlockArray{grid = grid2, initialState = initial, nib = nib2,
    njb = njb2, bcList = {west =
        InFlowBC_FromStagnation:new{stagnationState=stagCond}}}
blk3 = FluidBlockArray{grid = grid3, initialState = initial, nib = nib2,
    njb = njb2, bcList = {north = WallBC_NoSlip_FixedT:new{Twall=300},
            east = WallBC_NoSlip_FixedT:new{Twall=300},
            west = InFlowBC_FromStagnation:new{stagnationState=stagCond}}}
blk4 = FluidBlockArray{grid = grid4, initialState = initial, nib = nib4,
    njb = njb4}
blk5 = FluidBlockArray{grid = grid5, initialState = initial, nib = nib5,
    njb = njb5,
    bcList = {north = WallBC_NoSlip_FixedT:new{Twall=300,
        wall_function=true, group='loads'}}}
blk6 = FluidBlockArray{grid = grid6, initialState = initial, nib = nib6,
    njb = njb6}
blk7 = FluidBlockArray{grid = grid7, initialState = initial, nib = nib7,
```

```
    njb = njb7,
    bcList = {north = WallBC_NoSlip_FixedT:new{Twall=300,
        wall_function=true, group='loads'}}}
blk8 = FluidBlockArray{grid = grid8, initialState = initial, nib = nib8,
    njb = njb8}
blk9 = FluidBlockArray{grid = grid9, initialState = initial, nib = nib9,
    njb = njb9,
    bcList = {north = WallBC_NoSlip_FixedT:new{Twall=300,
        wall_function=true, group='loads'}}}
blk10 = FluidBlockArray{grid = grid10, initialState = initial,
    nib = nib10, njb = njb10,
    bcList = {east = OutFlowBC_FixedP:new{p_outside = 500}}}
blk11 = FluidBlockArray{grid = grid11, initialState = initial,
    nib = nib11, njb = njb11,
    bcList = {east = OutFlowBC_FixedP:new{p_outside = 500}}}
blk12 = FluidBlockArray{grid = grid12, initialState = initial,
    nib = nib12, njb = njb12,
    bcList = {north = WallBC_NoSlip_FixedT:new{Twall=300},
              east = OutFlowBC_FixedP:new{p_outside = 500},
              west = WallBC_NoSlip_FixedT:new{Twall=300}}}
print('Fluid Block Arrays... Done.')
--  ------------------------------------------------------------------
--  Boundary Conditions and config
--  ------------------------------------------------------------------
--
identifyBlockConnections()
-- configuration settings
config.max_time = 3.0e-3     -- Max flow sim time (sec)
config.max_step = 12000000   -- Max num steps
config.dt_init = 1.0e-9      -- Initial time step
config.cfl_value = 0.5       --
config.dt_plot = 1.0e-4      -- Write snaps of the flow using this interval
config.dt_history = 10.0e-5 -- Write history points using this
--
config.adjust_invalid_cell_data = true
config.max_invalid_cells = 20
--
-- Distribute the blocks - For cluster operation
mpiTasks = mpiDistributeBlocks{ntasks=100, dist="load-balance"}
```

Appendix C

Mesh Refinement MATLAB Code

C.1 Purpose and Function

The code in this appendix was used to read text files exported from `Eilmer4` via a post-processing operation. These `.txt` files contained the flow data of the cells at the nozzle exit plane, as well as the vertical line of cells 50mm downstream of the exit. The script then compares the flow properties at these locations for each mesh, and produces the plots shown in Section 5.6.

C.2 Script

```
%% m7MeshRef.m
% Alister Webb
% 5/10/2020
% Used to read and compare data from files created via Eilmer4 post
% processing operations.
%% Housekeeping
clc,clear,close all
% Number of cells in each mesh
nCells = [14188, 56934, 2051775, 3291959];
%% Enable or disable sections, change to true or false
plotVals = 0;
%% Directory and File Management
% This allows files in subfolders to be used without explicitly specifying
% an absolute path. Also works if the parent folder is moved.
S1 = pwd + "\" + ["m7meshRefS1Ex.txt","m7meshRefS1Ex50.txt"];
S2 = pwd + "\" + ["m7meshRefS2Ex.txt","m7meshRefS2Ex50.txt"];
S3 = pwd + "\" + ["m7meshRefS3Ex.txt","m7meshRefS3Ex50.txt"];
S4 = pwd + "\" + ["m7meshRefS4Ex.txt","m7meshRefS4Ex50.txt"];
%% Output file column headings for reference
% These are the headings present in the output files by default:
% 1:pos.x 2:pos.y 3:pos.z 4:volume 5:rho 6:vel.x 7:vel.y 8:vel.z 9:p 10:a
% 11:mu 12:k 13:mu_t 14:k_t 15:S 16:tke 17:omega 18:massf[0]-air 19:u 20:T
% 21:M_local 22:pitot_p 23:total_p 24:total_h
%% Read data
% Importing the data this way yields a warning message about variable
% names. Warnings are turned off for the import sections:
warning('off');
% Now import the data as a table.
stage1 = struct;
stage2 = struct;
```

```
stage3 = struct;
stage4 = struct;
pos = ["ex","ex50"];
% variable names also have to be set as the original ones can't be
% imported into MATLAB properly because they contain special characters.
varNames = {'x', 'y', 'z', 'volume', 'rho', 'vx', 'vy', 'vz', 'p', 'a',...
    'mu', 'k', 'mu_t', 'k_t', 'S', 'tke', 'omega', 'massofair', 'u',...
    'T', 'mach', 'pitot_p', 'total_p', 'total_h'};
for j=1:2
    stage1.(pos(j)) = readtable(S1(j));
    stage1.(pos(j)).Properties.VariableNames = varNames;
    stage2.(pos(j)) = readtable(S2(j));
    stage2.(pos(j)).Properties.VariableNames = varNames;
    stage3.(pos(j)) = readtable(S3(j));
    stage3.(pos(j)).Properties.VariableNames = varNames;
    stage4.(pos(j)) = readtable(S4(j));
    stage4.(pos(j)).Properties.VariableNames = varNames;
end
% Turn warnings back on
warning('on');
%% Average core flow V, M, P_pit, and T for each mesh
% Estimate core flow radius
BL = [findBLThickness([stage1.ex.y,stage1.ex.vx]);...
    findBLThickness([stage2.ex.y,stage2.ex.vx]);...
    findBLThickness([stage3.ex.y,stage3.ex.vx]);...
    findBLThickness([stage4.ex.y,stage4.ex.vx])];
coreS1 = 0.108-BL(1);
coreS2 = 0.108-BL(2);
coreS3 = 0.108-BL(3);
coreS4 = 0.108-BL(4);
% Values in the core flow region are:
s1 = array2table(table2array(stage1.ex((stage1.ex.y<coreS1),:)),...
    'variablenames',varNames);
s2 = array2table(table2array(stage2.ex((stage2.ex.y<coreS2),:)),...
    'variablenames',varNames);
s3 = array2table(table2array(stage3.ex((stage3.ex.y<coreS3),:)),...
    'variablenames',varNames);
s4 = array2table(table2array(stage4.ex((stage4.ex.y<coreS4),:)),...
    'variablenames',varNames);
% And the averages are
s1Av = array2table(mean(table2array(s1)), 'variablenames',varNames);
s2Av = array2table(mean(table2array(s2)), 'variablenames',varNames);
s3Av = array2table(mean(table2array(s3)), 'variablenames',varNames);
```

```
s4Av = (mean(table2array(s3)));
% All in one table (rows are stage number)
averages = array2table([mean(table2array(s1));mean(table2array(s2));...
    mean(table2array(s3));mean(table2array(s4))],'variablenames',...
    varNames);
% And reynolds number is
Re = averages.rho.*averages.vx./averages.mu;
averages = addvars(averages, Re);
% Boundary layer thickness for reference
averages = addvars(averages, BL);
% Then percentage differences are
percDiff = array2table(100*abs(diff(table2array(averages)))./...
    table2array(averages(1:end-1,:)), 'variablenames',...
    averages.Properties.VariableNames);
%% Plot the mean core flow parameters fo each mesh
% Can be toggled on or off
if plotVals == true
    figure(2)
    subplot(2,1,1)
    title('Average core flow parameters produced by each mesh')
    yyaxis left
    plot([1,2,3,4], averages.mach)
    ylabel('Mach number','interpreter','latex')
    axis([-inf,inf,5.5,7.5])
    grid on
    grid minor
    yyaxis right
    plot([1,2,3,4], averages.pitot_p./1000)
    ylabel('Pitot Pressure (kPa)','interpreter','latex')
    xticks([1,2,3,4])
    xticklabels({'Mesh 1', 'Mesh 2', 'Mesh 3', 'Mesh 4'})
    subplot(2,1,2)
    yyaxis left
    plot([1,2,3,4], averages.T)
    ylabel('Temperature (K)','interpreter','latex')
    grid on
    grid minor
    yyaxis right
    plot([1,2,3,4], averages.Re./1e6)
    ylabel('Reynolds Number ($\times10^6$m$^{-1}$)','interpreter','latex')
    xticks([1,2,3,4])
    xticklabels({'Mesh 1', 'Mesh 2', 'Mesh 3', 'Mesh 4'})
    set(gcf, 'PaperUnits', 'centimeters');
```

```
    set(gcf, 'PaperPosition', [0 0 22 15]);
    print('m7properties.eps','-depsc',['-r' num2str(resolution)]);
end
%% Percentage differences between meshes
% Can be toggled on or off
if plotVals == true
    vals = [percDiff.mach, percDiff.pitot_p, percDiff.T, percDiff.Re,...
        percDiff.BL];
    figure(3)
    ax = axes;
    ax.ColorOrder = [1 0 0; 0 0 1;0 1 0; 0 0 0;1 0 1];
    hold on
    for i=1:4
        plot(vals(:,i))
    end
    hold off
    ylabel('\% difference','interpreter','latex')
    xticks([1,2,3])
    xticklabels({'Mesh 1 to Mesh 2', 'Mesh 2 to Mesh 3',...
        'Mesh 3 to Mesh 4'})
    grid on
    grid minor
    title('Percentage difference between subsequent mesh revisions')
    legend('Mach number', ' Pitot Pressure', 'Temperature',...
        'Reynolds Number')
    set(gcf, 'PaperUnits', 'centimeters');
    set(gcf, 'PaperPosition', [0 0 22 10]);
    print('m7perc.eps','-depsc',['-r' num2str(resolution)]);
end
%% Boundary layer thickness
function out = findBLThickness(input)
% Calculates the y coord at which the velocity is less than 99% of the
% average core flow. To start find the values that are within 85% of the
% maximum velocity to find the velocities that are in the core flow.
velAv = mean(input(0.850*max(input(:,2))<input(:,2),2));
% find the values that are within 1% of the average
coreflow = input(0.990*velAv<input(:,2),:);
% The boundary layer thickness is then estimated as the difference between
% the last y value and the y value that corresponds to the slowest core
% flow velocity
out = max(input(:,1))-coreflow(end,1);
end
```

Appendix D

Results Post-processing Code

D.1 Overview

The code in this appendix serves to post process the `Eilmer4` results into a format usable in MATLAB, and then perform analysis of the results using MATLAB. There are four stages to the post processing operation:

- Process results with `Eilmer4`. This involves exporting the results to `vtk-xml` format for ParaView.
- Process results with ParaView. The output of this stage is several `.csv` files for MATLAB.
- Process results with MATLAB. This is an intermediate step for MATLAB. It serves to read and average the flow data from the `.csv` files and store them in `.mat` format. This is to reduce the load time for the MATLAB flow data.
- Create meaningful interpretations of the results using MATLAB.

D.2 Paraview Export

This script is run using ParaView's Python shell, and reads the `vtk-xml` format files produced by the `Eilmer4` post processing operation. This operation produces one `.csv` file for each timestep.

```
# trace generated using paraview version 5.8.0
# Modified by Alister Webb for import from vtk-xml and export to .csv
# 1/10/2020
#
import os
# Set the regime and stagnation condition. These set the output folder
regime = 'M6'
cond = '1mpa'
# Set the filepath of the plot files (Eilmer4 post processed)
#   First is the parent directory
infilePath = ''
outfilePath = ''
#   Second is the destination directory
intermediate = regime + '/' + cond + '/final/plot/'
#Set the output file name and path
```

```
outputName = cond + '.csv'
out = outfilePath + regime + '/' + cond + '/' + outputName
# Get a list of all the files in the plot directory
fileList = os.listdir(infilePath + intermediate)
# returns an array of filenames. Only the .pvtu files will be used, so:
#   Preallocate
SplitNames = []
List = []
inputFNames = []
files = []
# For each filename
for x in fileList:
    SplitNames = x.split('.') # Split it by '.' to get the file extension
    if SplitNames[-1] == 'pvtu':
        List.append(SplitNames)
# 'List' is a list of all the .pvtu files in the directory
# Now re-combine the name and extension
for a in List:
    inputFNames.append(a[0] + '.' + a[1])
# Then add in the absolute filepath
for c in inputFNames:
    files.append(infilePath + intermediate + c)
# Finally pick the last three files (if there are more than three)
if len(files)>=3:
    files = [files[-3],files[-2],files[-1]]
print(files)

# Remaining code was generated by ParaView using the 'trace' functionality
####

#### import the simple module from the paraview
from paraview.simple import *
#### disable automatic camera reset on 'Show'
paraview.simple._DisableFirstRenderCameraReset()

# create a new 'XML Partitioned Unstructured Grid Reader'

Data = XMLPartitionedUnstructuredGridReader(FileName=files)
Data.CellArrayStatus = ['pos.x', 'pos.y', 'pos.z', 'volume', 'rho', 'vel.x
    ', 'vel.y', 'vel.z', 'p', 'a', 'mu', 'k', 'mu_t', 'k_t', 'S', 'tke', '
    omega', 'massf[0]-air', 'u', 'T', 'M_local', 'pitot_p', 'total_p', '
    total_h', 'vel.vector']
```

```
# get active view
renderView1 = GetActiveViewOrCreate('RenderView')
# uncomment following to set a specific view size
# renderView1.ViewSize = [1611, 593]

# get layout
layout1 = GetLayout()

# show data in view
DataDisplay = Show(Data, renderView1, 'UnstructuredGridRepresentation')

# trace defaults for the display properties.
DataDisplay.Representation = 'Surface'
DataDisplay.ColorArrayName = [None, '']
DataDisplay.OSPRayScaleFunction = 'PiecewiseFunction'
DataDisplay.SelectOrientationVectors = 'None'
DataDisplay.ScaleFactor = 0.20168249607086183
DataDisplay.SelectScaleArray = 'None'
DataDisplay.GlyphType = 'Arrow'
DataDisplay.GlyphTableIndexArray = 'None'
DataDisplay.GaussianRadius = 0.010084124803543091
DataDisplay.SetScaleArray = [None, '']
DataDisplay.ScaleTransferFunction = 'PiecewiseFunction'
DataDisplay.OpacityArray = [None, '']
DataDisplay.OpacityTransferFunction = 'PiecewiseFunction'
DataDisplay.DataAxesGrid = 'GridAxesRepresentation'
DataDisplay.PolarAxes = 'PolarAxesRepresentation'
DataDisplay.ScalarOpacityUnitDistance = 0.016032332614375748

# reset view to fit data
renderView1.ResetCamera()

#changing interaction mode based on data extents
renderView1.InteractionMode = '2D'
renderView1.CameraPosition = [1.008412480354309, 0.15000000596046448,
    10000.0]
renderView1.CameraFocalPoint = [1.008412480354309, 0.15000000596046448,
    0.0]

# get the material library
materialLibrary1 = GetMaterialLibrary()

# update the view to ensure updated data information
```

```
renderView1.Update()

# save data
SaveData(out, proxy=Data, WriteTimeSteps=1,
    Filenamesuffix='_%d',
    ChooseArraysToWrite=0,
    PointDataArrays=[],
    CellDataArrays=['M_local', 'S', 'T', 'a', 'k', 'k_t', 'massf[0]-air',
        'mu', 'mu_t', 'omega', 'p', 'pitot_p', 'pos.x', 'pos.y', 'pos.z', '
        rho', 'tke', 'total_h', 'total_p', 'u', 'vel.vector', 'vel.x', 'vel
        .y', 'vel.z', 'volume'],
    FieldDataArrays=[],
    VertexDataArrays=[],
    EdgeDataArrays=[],
    RowDataArrays=[],
    Precision=5,
    UseScientificNotation=0,
    FieldAssociation='Cell Data',
    AddMetaData=0,
    AddTime=0)

# destroy Data
Delete(Data)
del Data

ResetSession()
```

D.3 Flow Data Import

D.3.1 Cell and Block Information Lua Script

This script is used to read an `Eilmer4` flow solution and return the information for each cell and each sub-block in separate text files. Cell information includes location (in x and y), block number, sub-block number, and number of cells in the sub-block in x and y. Block information includes; cells in x and y, and parent block number. These files are used in the MATLAB import function `getFlowData.m` and are specific to each mesh.

```
-- get-cell-data.lua
-- Invoke with the command line:
-- $ e4shared --custom-post --script-file=get-cell-data.lua
-- Alister Webb 8/9/2020
print("Read flow solution")
-- Specify the number of blocks (sub-blocks because of FluidBlockArrays)
-- in the flow solution and read the flow solution
nsb = 139
fsol = FlowSolution:new{jobName="Mach-6", dir="1mpa/final", tindx=last,
                        nBlocks=nsb}
print("Flow solution read","\n", "fsol=", fsol)
-- Create a text file for the output
fName = "cell-data.txt"
-- Find the number of sub-blocks in each block using the indices from the
-- input script.
topCylY = 1; blY = 1; coreY = 2; topTestY = 3
cylX = 4; thrX = 2; intermX = 3; nozX = 12; testX = 12
-- And tabulate them (the row now corresponds to the parent block number)
subBlockDiv = {
    {nib = cylX, njb = coreY}, -- Bottom of the cylinder
    {nib = cylX, njb = blY}, -- Middle of the cylinder
    {nib = cylX, njb = topCylY}, -- Top of the cylinder
    {nib = thrX, njb = coreY}, -- Throat core flow block
    {nib = thrX, njb = blY}, -- Throat boundary layer block
    {nib = intermX, njb = coreY}, -- Intermediate Nozzle core flow block
    {nib = intermX, njb = blY}, -- Intermediate Nozzle BL block
    {nib = nozX, njb = coreY}, -- Nozzle core flow block
    {nib = nozX, njb = blY}, -- Nozzle boundary layer block
    {nib = testX, njb = coreY}, -- Lower test section block
    {nib = testX, njb = blY}, -- Middle test section block
    {nib = testX, njb = topTestY} -- Upper test section block
```

```
}
-- Then the cumulative number of sub-blocks in each block tabulated
Cnb = {0} -- We need a zero point to work out ranges below
for b = 1, #subBlockDiv do
    Cnb[b+1] = Cnb[b] + subBlockDiv[b].nib*subBlockDiv[b].njb
end
-- Open the file and write the comma separated header row
f = assert(io.open(fName, "w"))
f:write("nic", ",", "njc", ",", "subBlock", ",", "nib", ",", "njb", "," ,
        "block", ",", "x", ",", "y", "\n")
-- Print a file creation message
print("File "..fName.." Created")
-- For each sub-block, find the number of cells in x and y
for ind = 0, nsb-1 do
    nj = fsol:get_njc(ind)
    ni = fsol:get_nic(ind)
    -- Then look through the table of cumulative sub-block numbers and
    -- find which indices the sub-block number is between
    for k = 1, #Cnb-1 do
        if (ind+1) >= (Cnb[k]+1) and (ind+1) <= Cnb[k+1] then
        nb = k-- Then set the block number to that index
        end
    end
    -- For each row of cells in y
    for j = 0, nj-1 do
        -- and then for each cell in x
        for i = 0, ni-1 do
            -- Load the cell data for this cell
            cellData = fsol:get_cell_data{ib=ind, i=i, j=j}
            -- Write the number of cells in x and y for that sub-block,
            -- the sub-block number, the parent block number, and the x
            -- and y coordinates for this cell
            f:write(ni, ",", nj, ",", ind+1, ",", subBlockDiv[nb].nib,
                    ",", subBlockDiv[nb].njb, ",", nb,
             ",", cellData["pos.x"], ",", cellData["pos.y"], "\n")
        end
    end
end
-- Close the file and display a message upon completion
f:close()
print("Indices and block numbers for all cells written to "..fName)
--
fName = "sub-block-data.txt"
```

```
-- Create a new file to write the sub-block information to. Open it and
-- write the comma separated header row
f = assert(io.open(fName, "w"))
f:write("nic", ",", "njc", ",", "subBlock", ",", "nib", ",", "njb", "," ,
        "block", "\n")
-- Print a file creation message
print("File "..fName.." Created")
-- For each sub-block, find the number of cells in x and y
for ind = 0, nsb-1 do
    nj = fsol:get_njc(ind)
    ni = fsol:get_nic(ind)
    -- Then look through the table of cumulative sub-block numbers and
    -- find which indices the sub-block number is between
    for k = 1, #Cnb-1 do
        if (ind+1) >= (Cnb[k]+1) and (ind+1) <= Cnb[k+1] then
        nb = k-- Then set the block number to that index
        end
    end
    -- Write the number of cells in x and y for that sub-block, the
    -- sub-block number, and the parent block number
    f:write(ni, ",", nj, ",", ind+1, ",", subBlockDiv[nb].nib, ",",
            subBlockDiv[nb].njb, ",", nb, "\n")
end
-- Close the file and display a message upon completion
f:close()
print("Indices and block numbers for all sub-blocks written to "..fName)
```

D.3.2 MATLAB Flow Data Import Function

The data exported from ParaView is a comma separated list containing the cell data. Each cell and it's data occupies a row. The list can be subdivided into groups of cells for each row in each sub-block. The import script reads all the data, then uses the cell and block information obtained with `get-cell-data.lua` to determine where these groups of cells occur in the `.csv` file. It then arranges the data for each cell into a table containing an array for each property. The output arrays(inside the table) are $m \times n$ where m is the number of cells in the tallest part of the domain and n is the number of cells in the length of the domain. Areas where there no cells are replaced with `NaN`. This function is called in '`dataProcessor.m`'.

```
%% getFlowData.m
% Alister Webb
% 16/09/2020
% Reads the flow data from a .csv file based on the cell and block
% information in two text files.
%   Based on the block arrangement below
%
%                         _________@---------@--------------------@
%   @---------@---------@         |         |                      |
%   | Block 3 |  *NaN*  |  *NaN*  |  *NaN*  |      Block 12      |
%   @---------@---------@---------@---------@--------------------@
%   | Block 2 | Block 5 | Block 7 | Block 9 |      Block 11      |
%   @---------@---------@---------@---------@--------------------@
%   | Block 1 | Block 4 | Block 6 | Block 8 |      Block 10      |
%   @---------@---------@---------@---------@--------------------@
%
%   | Driver  | Throat  | Diverging Section |    Test Section    |
%
% For a more detailed drawing of the flow domain, see
% Section 4.4 - Flow domain and Geometry
%
% Sections shown as *NaN* are filled with NaN later in the script so that
% the entire flow domain can be represented in an array (with x columns
% and y rows)
%% Full flow domain cell data import function
function outTable = getFlowData(datafName, cellDatafName,...
    blockDatafName)
% Read the list of cells. Warnings are turned off for this as MATLAB has
% to change the variablenames due to them having special characters
```

```
warning('off')
data = readtable(datafName);
warning('on')
% Read cell and sub-block information (Generated by get-block-data.lua)
cellinfo = readtable(cellDatafName);
subBlockInfo = readtable(blockDatafName);
% Put the cell data into an array to use in the loops
dataArray = table2array(data);
% Set the number of blocks and sub-blocks
nblocks = cellinfo.block(end);
nsubblocks = cellinfo.subBlock(end);
% Preallocate the cell array that will contain the cell data that is
% sorted by sub-block
subBlocks = cell(nsubblocks, 1);
% Then the number of variables of interest
numVars = size(data,2)-4;
for j = 1:nsubblocks
    % Create a temporary data array containing the cell data of all the
    % cells in block j
    temp = dataArray(cellinfo.subBlock==j,:);
    % Preallocate the initial result matrix
        Z = zeros(subBlockInfo.njc(j), subBlockInfo.nic(j), numVars);
    % Calculate the number of rows of cells in this block
        numRows = subBlockInfo.njc(j);
    for k = 1:numVars
        % Reset the initial row offset for each variable
        row0 = 1;
        for m = 1:numRows
            % Find the rows of the data array that need to be used
            rows = row0:(row0+subBlockInfo.nic(j)-1);
            % Find the row offset for each group of cells
            row0 = rows(:,end)+1;
            % Allocate the cells to their respective rows
            Z(m, :, k) = temp(rows,k)';
        end
        % Allocate the collection of cell data to the index of subBlocks
        % that corresponds to the sub-block number
        subBlocks{j} = Z;
    end
end
% Preallocate a cell array for the block data
output = cell(nblocks,1);
% Set the initial row offset
```

```
row0 = 0;
for k = 1:nblocks
    % Find the block definitions (contains the information for all the
    % sub-blocks in the block)
    blockDefs = subBlockInfo(subBlockInfo.block==k,:);
    blockRows = max(blockDefs.njb(:,:));
    blockCols = max(blockDefs.nib(:,:));
    % Find the number of sub-blocks in this block, as well as the rows to
    % include in the temporary array
    numSubBlocks = blockRows*blockCols;
    inRows = (1:numSubBlocks)+row0;
    row0 = inRows(end);
    temp = subBlocks(inRows,:);
    % Preallocate the second loop output array Z2
    increment = 1:max(blockDefs.njb):size(blockDefs,1);
    columns = sum(blockDefs.nic(increment));
    increment2 = 1:max(blockDefs.nib):size(blockDefs,1);
    rows = sum(blockDefs.njc(increment2));
    Z2 = nan(rows, columns, numVars);
    output0 = 0;
    for m = 1:max(blockDefs.njb(:,:))
        % Another temporary variable
        outRows = (1:(max(blockDefs.njb(:,:))):numSubBlocks)+(m-1);
        intermediate = temp(outRows,:);
        % Find the number of cells in x
        xTot = 0;
        for x = 1:length(intermediate)
            xTot = xTot+size(intermediate{x},2);
        end
        % And in y
        yTot = size(intermediate{x},1);
        % Preallocate
        Z = nan(yTot, xTot, numVars);
        col0 = 1;
        colOffset = 0;
        for n = 1:length(intermediate)
            % Put the sub-block data in it's respective columns in the
            % block data array
            col = (col0:max(size(intermediate{n},2)))+colOffset;
            colOffset = col(end);
            Z(:, col, :) = intermediate{n};
        end
        % Then for instances where there are multiple rows of sub-blocks,
```

```
        % put each row in it's place in the output
        outputrows = (1:max(blockDefs.njc))+output0;
        output0 = outputrows(end);
        Z2(outputrows, :,:) = Z;
    end
    % Place each block's data in a cell in the output cell array
output{k} = Z2;
end
% Combine the blocks into the separate sections in x (i.e. blocks 1, 2,
% and 3 are section 1, and are all stacked on top of each other...
sections = {};
sections{1} = [output{1};output{2};output{3}];
sections{2} = [output{4};output{5}];
sections{3} = [output{6};output{7}];
sections{4} = [output{8};output{9}];
sections{5} = [output{10};output{11};output{12}];
yMax = size(sections{5},1);
% Then add in NaN values to pad out the unused y values so there is
% whitespace in any contour plots that may be made
for p = 1:length(sections)
sections{p} = [sections{p};nan(yMax-(size(sections{p},1)),...
    size(sections{p},2), numVars)];
end
fullDomain = [sections{1},sections{2},sections{3},sections{4},...
    sections{5}];
% Preallocate the final output table
outTable = table(zeros(size(fullDomain,1),0));
for q = 1:size(fullDomain,3)
    outTable = addvars(outTable, fullDomain(:,:,q));
end
% Clip the first empty column and give the table variable names
outTable(:,1) = [];
varNames = {'x', 'y', 'z', 'volume', 'rho', 'vx', 'vy', 'vz', 'p', 'a',...
    'mu', 'k', 'mu_t', 'k_t', 'S', 'tke', 'omega', 'mass_air', 'u',...
    'T', 'mach', 'pitot_p', 'total_p', 'total_h'};
outTable.Properties.VariableNames = varNames;
end
```

D.3.3 MATLAB Processing and Export to `.mat` Format

This script uses the above function to read and average the flow data. Then it exports the flow data as a `.mat` file to be loaded in the individual analysis scripts. There is one output file for each nozzle: `m6flowdata.mat` and `m7flowdata.mat`.

```
%% DataProcessor.m
% Alister Webb
% 18/09/2020
% Read required flow data from csv and save them as .mat files for faster
% loading in data analysis scripts. Also time averages data where required
%   Requires:
%       - getFlowData.m
%{
% Referenced file structure:
 Parent/
 |-- DataProcessor.m
 |-- MATLAB Analysis/
 |     -- flowdata.mat
 |
 |-- M6/
 |    |-- 1mpa/
 |    |-- 4mpa/
 |     -- 7mpa/
  -- M7/
     |-- 1mpa/
     |-- 4mpa/
      -- 7mpa/
%}
%% Housekeeping
clc,clear,close all
%% Set file paths
cellfile = "cell-data.txt";
blockfile = "sub-block-data.txt";
parent = pwd;
child = ["\M6\","\M7\"];
%% Preallocate variables:
% Preallocate output structure
fieldNames = {'m6stag1mpa','m6stag4mpa','m6stag7mpa','m7stag1mpa',...
    'm7stag4mpa','m7stag7mpa'};
% List the stagnation conditions (subfolder names)
subfolder = {'1mpa\','4mpa\','7mpa\'};
```

```
% Output field (iterates during the loop)
field = 0;
% Output file prefix
noz = ["m6","m7"];
%% Read each file and average the values over the number of files (steps)
% This is for both M6 and M7
for nRegs = 1:2
    % Preallocate the output (resets on each outer loop)
    flowData = struct;
    % set the source directory to M6 or M7
    destination = parent + child(nRegs);
    % then for each subfolder:
    cellInfo = destination + "cell-data.txt";
    blockInfo =  destination + "sub-block-data.txt";
    for j = 1:length(subfolder)
        % File information
        cd(destination + subfolder{j})
        folderInfo = dir('*.csv');
        fNames = {folderInfo.name}; % List of .csv files in destination
        % folder
        cd(parent)
        % Read the first flow data to 'preallocate' the output. This is
        % the easiest way to do it
        filepath = destination + subfolder{j} + fNames{1};
        initialData = getFlowData(filepath, cellInfo, blockInfo);
        varNames = initialData.Properties.VariableNames;
        Z = nan(size(initialData.(varNames{1}))); % Set the size based on
        % the first variable. Also preallocate the output cell array.
        output = cell(0,0);
        % Table variables are put into a cell array for ease of use.
        for nVars = 1:(size(initialData,2)) %i.e. number of flow variables
            output{nVars} = initialData.(varNames{nVars});
        end
        % With the variables initialised in a cell array, the values from
        % each timestep can be added to the initial one (If there's only 1
        % step this is skipped):
        if length(fNames)>=2
            for  numFiles = 2:(length(fNames))
                filepath = (destination+subfolder{j}+fNames{numFiles});
                tempData = getFlowData(filepath, cellInfo, blockInfo);
                % sum the flow data into a numeric matrix
                for nVars = 3:(size(tempData,2))
                    %i.e. number of flow variables except the coordinates
```

```
                    output{nVars} = output{nVars} +...
                        tempData.(varNames{nVars});
                end
            end
        end
        % Find the average by dividing each matrix by the number of steps
        data = tempData(:,1:2);
        for nVars = 3:length(output)
            %i.e. the number of flow variables except the coordinates
            output{nVars} = output{nVars}./length(fNames);
            data = addvars(data, output{nVars});
        end
        % Clip the first variable and set the variable names. Assign the
        % flow data to a field inside a structure
        field = field + 1;
        data.Properties.VariableNames =...
            initialData.Properties.VariableNames;
        flowData.(fieldNames{field}) = data;
    end
    % Then save the output variable as a .m file
    fPath = 'MATLAB Analysis\' + noz(nRegs);
    save(fPath + 'flowdata.mat','-struct','flowData','-v7.3')
end
```

D.4 MATLAB Analysis

This script is for the Mach 6 nozzle analysis and is very similar to the Mach 7 nozzle analysis script (except for the comparison to historical data). The Mach 6 analysis file compares the flow data to the data presented by Birch (2019). This script calls the 'makeplots.m script which is a collection of plot functions, for this reason it is not included here. makeplots.m was created to reduce clutter and make debugging easier.

```
%% M6_data_analysis.m
% Alister Webb
% 21/08/2020
% Used to read and compare data from files created via Eilmer4 post
% processing operations for the Mach 6 nozzle.
%   Requires:
%       -makeplots.m
%       -m6flowData.mat
%% Housekeeping
clc,clear,close all
%% Enable or disable sections, change to true or false
plotVals = 0;
% Figure number, increases after each figure
n = 1;
% Flow regime for output filenames
regime = 'M6_';
%% Data from Birch (2019)
% Experimental data and previous Eilmer4 data were tabulated on page 33 of
% Birch (2019). Each row is for a specific Mach number, values can range
% between these. Columns are as follows:
% 1:Mach, 2:P_free, 3:T_free, 4:rho, 5:vel.x, 6:vel.r (vel.r =~= vel.y)
ExpData = [5.95, 666, 71.3, 32.6e-3, 1007, 0;...
           5.85, 740, 73.4, 35.1e-3, 1005, 0];
EilmerData = [5.98, 648, 70.6, 31.8, 1007, 0];
% Nominal test conditions were also listed as:
% 1:Mach, 2:P_free, 3:T_free, 4:vel.x, 5:pitot_p, 6:Re_x
refData = [5.9, 702, 72, 1006, 31.8e3, 6.94e6];
%% Variable names for reference
% 'x', 'y', 'z', 'volume', 'rho', 'vx', 'vy', 'vz', 'p', 'a', 'mu', 'k',
% 'mu_t', 'k_t', 'S', 'tke', 'omega', 'mass_air', 'u', 'T', 'mach',
% 'pitot_p', 'total_p', 'total_h'
%% Read M6 data
% Import data from a a.mat file
loadedData = load('m6flowdata.mat'); % load puts the data into a structure
```

```
% So the field needs to be referenced to extract the flow data:
stag1mpa = loadedData.m6stag1mpa;
stag900k = loadedData.m6stag1mpa900k;
stag4mpa = loadedData.m6stag4mpa;
stag7mpa = loadedData.m6stag7mpa;
% Extract the coordinate matrices to make future calculations easier
x = stag1mpa.x(1,:);
y = stag1mpa.y(:,end)';
% Calculate and append Reynolds number to each stag cond
Re = stag1mpa.rho.*stag1mpa.vx./stag1mpa.mu;
stag1mpa = addvars(stag1mpa, Re);
Re = stag900k.rho.*stag900k.vx./stag900k.mu;
stag900k = addvars(stag900k, Re);
Re = stag4mpa.rho.*stag4mpa.vx./stag4mpa.mu;
stag4mpa = addvars(stag4mpa, Re);
Re = stag7mpa.rho.*stag7mpa.vx./stag7mpa.mu;
stag7mpa = addvars(stag7mpa, Re);
%% Extract Lines
% One at the exit plane and one 50mm downstream. The lines are
%   (x1,y1,x2,y2):
exitPlane = [1.187,0,1.187,0.108];
testSec = [1.237,0,1.237,0.299];
% 1mpa
exit1mpa = extract(exitPlane,x,y,stag1mpa);
test1mpa = extract(testSec,x,y,stag1mpa);
% 1mpa @ 900k
exit900k = extract(exitPlane,x,y,stag900k);
test900k = extract(testSec,x,y,stag900k);
% 4mpa
exit4mpa = extract(exitPlane,x,y,stag4mpa);
test4mpa = extract(testSec,x,y,stag4mpa);
% 7mpa
exit7mpa = extract(exitPlane,x,y,stag7mpa);
test7mpa = extract(testSec,x,y,stag7mpa);
%% Estimate the boundary layer thickness at the exit plane
BL1mpa = findBLThickness([exit1mpa.y,exit1mpa.vx]);
BL900k = findBLThickness([exit900k.y,exit900k.vx]);
BL4mpa = findBLThickness([exit4mpa.y,exit4mpa.vx]);
BL7mpa = findBLThickness([exit7mpa.y,exit7mpa.vx]);
%% Average parameters for each stag cond. at the exit plane
totalVars = {'vx','mach','p','T'};
exit = struct('s1mpa',exit1mpa, 's900k',exit900k, 's4mpa',exit4mpa,...
              's7mpa',exit7mpa);
```

```
fields = fieldnames(exit);
cfDia = [0.108,0.108,0.108,0.108] - [BL1mpa,BL900k,BL4mpa,BL7mpa];
for k = 1:length(fields)
    Z = table('size',[0,0]);
    cfD = cfDia(1,k);
    for names = 1:length(totalVars)
        temp = exit.(fields{k}).(totalVars{names});
        yvals = exit.(fields{k}).y;
        temp = mean(temp((yvals<cfD),:));
        Z = addvars(Z,temp,'NewVariableNames',(totalVars{names}));
    end
    exitmeans.(fields{k}) = Z;
end
%% Average parameters for each stag cond. at 50mm from the exit plane
test = struct('s1mpa',test1mpa, 's900k',test900k, 's4mpa',test4mpa,...
              's7mpa',test7mpa);
fields = fieldnames(test);
cfDia = [0.108,0.108,0.108,0.108] - [BL1mpa,BL900k,BL4mpa,BL7mpa];
for k = 1:length(fields)
    Z = table('size',[0,0]);
    cfD = cfDia(1,k);
    for names = 1:length(totalVars)
        temp = test.(fields{k}).(totalVars{names});
        yvals = test.(fields{k}).y;
        temp = mean(temp((yvals<cfD),:));
        Z = addvars(Z,temp,'NewVariableNames',(totalVars{names}));
    end
    testmeans.(fields{k}) = Z;
end
%% Plot the test section parameters
% Requires makeplots.m
if plotVals == true
    m6makeplots
end
%% Relative Difference to Pre-existing data
% The percentage difference will be found for the core flow at the exit
% plane:
%       Assemble an array of data:
S1mpa = [exit1mpa.mach,exit1mpa.p,exit1mpa.T,exit1mpa.vx,...
    exit1mpa.pitot_p,exit1mpa.Re];
% Find the values in the core flow (outside the boundary layer) and averge
% them
coreFlow = mean(S1mpa(exit1mpa.y<0.09,:));
```

```
% Then the percentage difference
PCD = percentDiff(refData,coreFlow);
% 1:Mach, 2:P_free, 3:T_free, 4:rho, 5:vel.x, 6:vel.r (vel.r =~= vel.y)
S1mpa = [exit1mpa.mach,exit1mpa.p,exit1mpa.T,exit1mpa.rho,...
    exit1mpa.vx,exit1mpa.vy];
coreFlow = mean(S1mpa(exit1mpa.y<0.09,:));
PCD2 = maxDiff(ExpData,coreFlow);
%% Percentage difference function
% Requires two inputs of the same size
function out = percentDiff(dataset1, dataset2)
% percentage difference is the ratio of the difference to the average
out = 200*abs(dataset1-dataset2)./(dataset1+dataset2);
end

%% Boundary layer thickness
function out = findBLThickness(input)
% Takes a matrix of the values of y coordinate and x velocity [coord, vx]
% Calculates the y coord at which the velocity is less than 99% of the
% average core flow
% to start find the values that are within 85% of the maximum velocity to
% find the velocities that are in the core flow. Then find the average
% core flow velocity.
velAv = mean(input(0.850*max(input(:,2))<input(:,2),2));
% find the values that are within 1% of the average
coreflow = input(0.990*velAv<input(:,2),:);
% The boundary layer thickness is then estimated as the difference between
% the last y value and the y value that corresponds to the slowest core
% flow velocity
out = max(input(:,1))-coreflow(end,1);
end
%% Line and Area extraction function
function dataRange = extract(corners, x, y, data)
% Used to extract a smaller area of a larger dataset based on the
% specified corner coordinates, x, y, and dataset. Corner coordinates
% should be diagonally opposed to extract an area. Corner points that lie
% on the same x or y value will be used to extract a line between the
% coordinates.
% Find the coordinates closest to each specified point:
%   x values
[~,xlowerBound] = min(abs(repmat(corners(1), 1, numel(x(1,:)))-x));
[~,xupperBound] = min(abs(repmat(corners(3), 1, numel(x(1,:)))-x));
% Then the range of the indices of the x vlaue input
xRange = xlowerBound:1:xupperBound;
```

```
% Then the x values that correspond to each point
xVals = x(xRange);
%   y values
[~,ylowerBound] = min(abs(repmat(corners(2), 1, numel(y(1,:)))-y));
[~,yupperBound] = min(abs(repmat(corners(4), 1, numel(y(1,:)))-y));
% Then the range of the indices of the y vlaue input
ybound = [ylowerBound,yupperBound];
yRange = min(ybound):1:max(ybound);
% Then the y values that correspond to each point
yVals = y(yRange);
% Extract the smaller area from the complete dataset for all variables
if numel(yVals)==1 %is the line horizontal?
    dataRange = table(zeros(numel(xRange),0));
    for j=1:size(data,2)
    temp = table2array(data(:,j));
    dataRange = addvars(dataRange,temp(yRange,xRange)');
    end
elseif numel(xVals)==1 % is the line vertical
    dataRange = table(zeros(numel(yRange),0));
    for j=1:size(data,2)
    temp = table2array(data(:,j));
    dataRange = addvars(dataRange,temp(yRange,xRange));
    end
else %  the coords corners of a rectangle if the line is not horizontal or
     %  vertical
    dataRange = table(zeros(numel(yRange),0));
    for j=1:size(data,2)
    temp = table2array(data(:,j));
    dataRange = addvars(dataRange,temp(yRange,xRange));
    end
end
% Clip the first column of the table because it contains only zeros due to
% the 'addvars()' function
dataRange(:,1) = [];
% Set the variable names to the originals
dataRange.Properties.VariableNames = data.Properties.VariableNames;
end
%% Percentage difference outside a range
function out = maxDiff(range, dataset)
%Range must be 2*n and dataset must be 1*n.
% Will produce a single value for each entry in the dataset showing the
% maximum deviation outside a range. This value is signed and shows
% whether the data value is above or below the range, or if it resides
```

```
% inside it.
% Create an anonymous percentage difference function
pcD = @(x,y) 100*abs(x-y)./(x);
% Then the min and max values
minVals = min(range);
maxVals = max(range);
%Preallocate the output
out = zeros(size(dataset));
% Find where the values sit in the range
belowRange = dataset<minVals;
aboveRange = dataset>maxVals;
insideRange = or(~belowRange,~aboveRange);
% Then calculate the percentage difference accordingly
out(insideRange) = 0;
out(belowRange) = -pcD(minVals(belowRange),dataset(belowRange));
out(aboveRange) = pcD(maxVals(aboveRange),dataset(aboveRange));
end
```