The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

Declaration of Original Work

This is to certify that the calculations, computer code, results and other work presented in this dissertation are essentially my own work, except where otherwise indicated, and has not been submitted for a degree at any other university.

Signed by candidate

Prakash Parbhoo

April 2000

Title

A Numerical Investigation into the Tactical Advantages of Supercruise for Combat Aircraft

By

P. Parbhoo

A dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science in Engineering

University of Cape Town

Department of Mechanical Engineering

University of Cape Town

Rondebosch 7700

South Africa

Acknowledgements

I wish to thank the following for their contribution to this dissertation:

My supervisor, Dr C. Redelinghuys for his support, encouragement, and the opportunity to work with him and gain some insight into the attributes that a good engineer should posses.

My mother and late father who provided substantial support and encouragement.

Kirsten Winkler, without whom this document would not have been completed.

Shelley Adams for providing motivation and encouragement over many years.

Alfred Hoernle and Clem van der Riet for demanding that this dissertation be completed.

University of Cape Town

Terms of Reference

Funding for this work was provided by the Denel Group. The objective of the work would be to determine the tactical advantages, if any, of providing a Mirage III class combat aircraft with supercruise capability. This would be determined by developing numerical models of the aircraft components and producing performance predictions from these.

University of Cape Town

Synopsis

Developments in the field of combat aircraft design have resulted in a new generation of fighter aircraft. These new aircraft sport better aerodynamic design, advanced avionics and modern power plants. This leaves legacy combat aircraft vulnerable in a combat scenario. Although updated avionics provide some improvement for legacy combat aircraft, the aerodynamic design cannot be substantially modified. The power plant, or turbojet engine, can however be upgraded to a new model.

This dissertation describes a numerical investigation into the tactical advantages offered by upgrading a hypothetical aircraft with a newer engine model. The airframe is largely based on the Dassault Mirage III, and the engines are based on the SNECMA Atar 9k50 and M53.

A brief overview of recent developments in combat aircraft design and engine performance is provided which illustrates the trend towards high thrust to weight ratio aircraft engines with lower fuel consumption.

A thermodynamic model of a turbojet / low bypass ratio turbofan engine is developed and a parametric exercise is provided to illustrate the key factors influencing engine performance. A numerical model is developed for the supersonic intake which allows the prediction of both pressure recovery and mass flow rate through the component. The combined result of these models is shown to produce predictions which compare favourably with published data.

An aerodynamic model is built around the US Airforce Digital DATCOM. Some example output from this program is provided and compared to expected analytical trends.

The propulsion, intake and aerodynamic models are combined to provide predictions of key performance indicators such as Thrust Specific Fuel Consumption, Sustainable and Attainable Turn Rate, Specific Excess Power and Range.

These predictions indicate that upgrading a hypothetical combat airframe with a new propulsion unit offers some tactical advantages such as increased Specific Excess Power, Sustained Turn Rate and Range.

A recommendation is made that further simulations be performed to obtain a more detailed indication of how the suggested improvements in performance will be of benefit in a combat scenario.

Shortcomings of the models developed in this investigation are identified and suggestions are made as to where they should be improved.

All the source code for the various simulations is provided in the appendices.

University of Cape Town

Table of Contents

Declaration of Original Work	I
Title	II
Acknowledgements	III
Terms of Reference	IV
Synopsis	V
Table of Contents	VI
List of Illustrations	XI
List of Tables	XIII
List of Symbols	XIV
CHAPTER 1: INTRODUCTION	**1**
CHAPTER 2: BACKGROUND	**4**
2.1 Military Aircraft Propulsion	**4**
2.2 Air Combat	**5**
CHAPTER 3: SIMULATION OVERVIEW	**7**
3.1 Performance Indicators	**7**
3.1.1 Range	7
3.1.2 Thrust Specific Fuel Consumption	8
3.1.3 Specific Excess Power	8
3.1.4 Sustained Turn Rate and Attained Turn Rate	9
3.1.5 Thrust - Drag - Load Factor chart	10
3.2 Modelling the entire Aircraft	**11**
CHAPTER 4: PROPULSION MODEL	**12**
4.1 Turbojet Model	**12**
4.2 Typical Thermodynamic Cycle	**12**
4.3 Engine components and properties	**13**

University of Cape Town

4.4	**ESTIMATING THE THRUST**	**14**
4.4.1	MODELLING ASSUMPTIONS AND SIMPLIFICATIONS	14
4.4.2	CALCULATION OF THE THRUST PER UNIT AIRFLOW INTO THE GAS GENERATOR	14
4.4.3	CALCULATION OF NET THRUST OUTPUT	20
4.5	**ENGINE PERFORMANCE INDICATORS**	**21**
4.5.1	THRUST SPECIFIC FUEL CONSUMPTION (TSFC) AND SPECIFIC IMPULSE	21
4.5.2	T-S DIAGRAM CALCULATIONS	22
4.6	**FACTORS AFFECTING ENGINE PERFORMANCE**	**22**
4.6.1	REFERENCE CONFIGURATION FOR THE PARAMETRIC INVESTIGATION	22
4.6.2	TURBINE INLET TEMPERATURE	24
4.6.3	COMPRESSOR PRESSURE RATIO	24
4.6.4	INTAKE PRESSURE RECOVERY	25
4.7	**SOFTWARE IMPLEMENTATION OF THE PROPULSION MODEL**	**26**
CHAPTER 5: INTAKE MODELLING		**27**
5.1	**OVERVIEW OF INTAKE OPERATION**	**27**
5.1.1	SUBSONIC FLIGHT	27
5.1.2	DETACHED SHOCK MODE	28
5.1.3	CRITICAL MODE	28
5.1.4	SUB CRITICAL MODE	29
5.1.5	SUPER CRITICAL MODE	29
5.1.6	MASS FLOW AND PRESSURE RECOVERY FOR DIFFERENT OPERATING MODES	30
5.2	**CONICAL SHOCK PROPERTIES**	**31**
5.3	**STANDARD INTAKE PERFORMANCE CURVES**	**33**
5.4	**NUMERICAL ESTIMATION OF INTAKE PERFORMANCE**	**34**
5.4.1	DETACHED SHOCK SOLUTION	34
5.4.2	CRITICAL SOLUTION	35
5.4.3	SUB CRITICAL SOLUTION	37
5.4.4	SUPER CRITICAL SOLUTION	37
5.5	**SOFTWARE IMPLEMENTATION OF THE INTAKE MODEL**	**38**
CHAPTER 6: AERODYNAMIC MODEL		**39**
6.1	**OVERVIEW OF BASIC AERODYNAMICS**	**39**
6.1.1	SUBSONIC FLIGHT AND WING SECTIONS	40

University of Cape Town

6.1.2 VARIATION OF DRAG WITH MACH NUMBER 42

6.2 DIGITAL DATCOM MODEL 42

6.2.1 DESCRIBING THE AIRFRAME 42

6.2.2 FLIGHT CONDITIONS 46

6.2.3 LIMITATIONS OF THE MODEL 46

6.3 RESULTS OF THE DATCOM MODEL 47

6.3.1 C_L VS. AOA 47

6.3.2 C_D VS. AOA 47

6.3.3 $C_{L\,MAX}$ VS. M 48

6.3.4 C_D VS. MACH NUMBER 49

6.3.5 C_L/C_D VS. C_L 50

6.4 SOFTWARE IMPLEMENTATION OF THE AERODYNAMIC MODEL 51

CHAPTER 7: SIMULATION RESULTS 52

7.1 COMPARISON OF ENGINE PERFORMANCE 52

7.1.1 PROPULSION MODEL INPUT PARAMETERS 52

7.1.2 THERMODYNAMIC CYCLE: T-S DIAGRAMS 54

7.1.3 THRUST SPECIFIC FUEL CONSUMPTION (TSFC) 56

7.1.4 NET THRUST 58

7.2 COMPARISON OF THE INTAKE MODEL WITH STANDARD CURVES 60

7.3 THRUST - DRAG - LOAD FACTOR CHART 61

7.4 SPECIFIC EXCESS POWER 64

7.5 SUSTAINED TURN RATE 65

7.6 RANGE 66

CHAPTER 8: CONCLUSIONS AND RECOMMENDATIONS 68

8.1 CONCLUSIONS 68

8.2 RECOMMENDATIONS 69

REFERENCES 70

APPENDIX A: INTAKE MODEL 72

APPENDIX A-1 FLOW CHART 72

University of Cape Town

APPENDIX A-2 DETACHED SHOCK EQUATIONS 73

APPENDIX A-3 SAMPLE CONICAL SHOCK DATA 75

APPENDIX A-4 INTAKE MODEL: CLASS CCONEINTAKE 76

A.4.1 HEADER FILE: CONEINTAKE.H 76
A.4.2 IMPLEMENTATION FILE: CONEINTAKE.CPP 77

APPENDIX B: DATCOM MODEL 92

APPENDIX B-1 SCHEMATIC DRAWINGS 92

APPENDIX B-2 SAMPLE INPUT FILE 94

APPENDIX B-3 SAMPLE OUTPUT FILE 95

APPENDIX B-4 DATCOM POST-PROCESSING SOURCE CODE 101

B.4.1 PROGRAM: DATCOM POST 101
B.4.2 PROGRAM: CD_CL 107
B.4.3 PROGRAM: CL_CD_FULL 111

APPENDIX C: PROPULSION MODEL 116

FLOW CHART FOR ESTIMATING THRUST OUTPUT 116

APPENDIX C-2 PROPULSION MODEL: CLASS TURBOJET1D 117

C.2.1 HEADER FILE: 1DPROPMODEL.HPP 117
C.2.2 IMPLEMENTATION FILE: 1DPROPMODEL.CPP 120

APPENDIX C-3 PROPULSION SIMULATION CODE 138

C.3.1 PROGRAM: TSFC 138
C.3.2 PROGRAM: THRUST 142
C.3.3 PROGRAM: CALCTSDIAGRAM 148

APPENDIX D: PERFORMANCE ESTIMATION SOURCE CODE 152

APPENDIX D-1 SUSTAINED TURN RATE 152

APPENDIX D-2 SPECIFIC EXCESS POWER 158

APPENDIX D-3 DRAG - THRUST - LOAD FACTOR CHART 168

D.3.1 PROGRAM: CDMNALT 168
D.3.2 PROGRAM: CLVSMVSN 172

University of Cape Town

APPENDIX E: GENERAL SOURCE CODE 176

APPENDIX E-1 TAYLOR-MACOLL FLOW 176

E.1.1 MAIN FILE: TMFLOWPROPSMAIN.CPP 176
E.1.2 HEADER FILE: TMFLOWPROPS.H 178
E.1.3 IMPLEMENTATION FILE: TMFLOWPROPS.CPP 179

APPENDIX E-2 LOOKUP TABLES 187

E.2.1 HEADER FILE: LOOKUPTB.HPP 187
E.2.2 IMPLEMENTATION FILE: LOOKUPTB.CPP 188
E.2.3 HEADER FILE: LOOKUP4D.H 194
E.2.4 IMPLEMENTATION FILE: LOOKUP4D.CPP 195

APPENDIX E-3 IDEAL GAS BEHAVIOUR 203

E.3.1 HEADER FILE: IDEALGAS.H 203
E.3.2 IMPLEMENTATION FILE: IDEALGAS.CPP 204
E.3.3 HEADER FILE: IDEALGASSTREAM.H 211
E.3.4 IMPLEMENTATION FILE: IDEALGASSTREAM.CPP 212

APPENDIX E-4 DATA INPUT FILE SYSTEM 213

E.4.1 HEADER FILE: DATAFILEINPUT.HPP 213
E.4.2 IMPLEMENTATION FILE: DATAFILEINPUT.CPP 215

University of Cape Town

List of Illustrations

Figure 2.1: Trends in Thrust to Weight Ratio 4

Figure 2.2: Typical Aircraft Operating Envelopes and Stagnation Temperature limits. 5

Figure 2.3: Trends in TSFC 5

Figure 3.1: STR and ATR in simulated short range combat 10

Figure 4.1: Schematic of a turbojet engine with bypass. 12

Figure 4.2: Typical T-s diagram for the Brayton cycle, with bypass and losses 12

Figure 4.3: Variation of TSFC and $F/\dot{m}_a$ with turbine inlet temperature. 24

Figure 4.4: Variation of TSFC and $F/\dot{m}_a$ with compressor pressure ratio. 24

Figure 4.5: Variation of TSFC and $F/\dot{m}_a$ with diffuser pressure recovery. 25

Figure 5.1: Schematic of a supersonic conical intake with two-shock system 27

Figure 5.2: Detached shock arrangement 28

Figure 5.3: Critical mode shock arrangement 28

Figure 5.4: Two flow arrangements in sub critical mode 29

Figure 5.5: Super critical mode shock arrangement 30

Figure 5.6: Mass flow and Pressure Recovery for different operating modes. 30

Figure 5.7: Streamlines for supersonic flow over a cone 31

Figure 5.8: Comparison between published conical shock wave data and calculated results 32

Figure 5.9: Standard pressure recovery curves 33

Figure 5.10: Schematic representation of a detached shock and its discretisation 34

Figure 5.11: Pressure recovery in critical mode operation 35

Figure 5.12: Parameters for determining the mass flow through the intake 36

Figure 6.1: Basic aerodynamic forces acting on a wing. 39

Figure 6.2: Variation of lift coefficient with angle of attack 40

Figure 6.3 : Idealised variation of drag coefficient with angle of attack. 41

Figure 6.4: Variation of Lift-to-Drag ratio with C_L 41

Figure 6.5: Variation of C_{D0} with Mach number 42

Figure 6.6: Schematic aircraft model for use in the Digital DATCOM 43

University of Cape Town

Figure 6.7 : Airfoil section used for wing specification 44

Figure 6.8: Lift coefficient (C_L) for various angles of attack, at various Mach numbers 47

Figure 6.9: Drag coefficient (untrimmed) for various angles of attack and Mach numbers 48

Figure 6.10: Maximum lift coefficient vs. Mach number. 48

Figure 6.11: C_D for maximum C_L at 4000m 49

Figure 6.12: Variation of C_{D0} with Mach Number at 4000m 50

Figure 6.13: Variation of Lift to Drag ratio with Lift in the supersonic regime at sea level 50

Figure 7.1 : T-s Diagram for the 9k50, M=0.66, A/B = Off, Sea Level 54

Figure 7.2: T-s Diagram for the M53, M=0.66, A/B = Off, Sea Level 55

Figure 7.3: T-s Diagram for the 9k50, M=0.66, A/B = On, Sea Level 55

Figure 7.4: T-s Diagram for the M53, M=0.66, A/B = On, Sea Level 56

Figure 7.5: Comparison of TSFC between the 9k50 and M53 at Sea Level 56

Figure 7.6: Comparison of TSFC between the 9k50 and M53 at 14 km 57

Figure 7.7: Comparison of Thrust output between the 9k50 and M53 at Sea Level 58

Figure 7.8: Comparison of Thrust output between the 9k50 and M53 at 4 km 59

Figure 7.9: Comparison of Thrust output between the 9k50 and M53 at 14km 60

Figure 7.10: Diffuser performance comparison 60

Figure 7.11: C_L required for a given load factor and Mach number at 4km 61

Figure 7.12: C_D for n=1 to n=9 with C_T superimposed at 4km 62

Figure 7.13: Load Factor Map for Mach number and Altitude 63

Figure 7.14: Specific Excess Power (SEP) [m/s] for the 9k50 and M53, n=1 64

Figure 7.15: Sustained Turn rate comparison for the 9k50 and M53 at 4km 65

Figure 7.16: Sustained Turn rate, with Attainable Turn Rate (ATR) superimposed 66

Figure 7.17: Relative Range for different engine configurations at 4000m 67

Figure B.1: Plan view of the hypothetical fighter 92

Figure B.2: Elevation view of the hypothetical fighter 92

Figure B.3: Schematic outline superimposed on drawings from Jane's [3] 93

University of Cape Town

List of Tables

Table 4.1: Turbojet engine components and properties 13

Table 4.2 : Parameter values for the reference configuration.. 23

Table 4.3: Parameters for Function CycleThermodynamics 26

Table 6.1: DATCOM parameters for the wing planform.. 44

Table 6.2: DATCOM parameters for modelling the Elevators 45

Table 6.3: DATCOM parameters for modelling the Tail... 46

Table 7.1: Propulsion model input parameters... 52

Table 7.2: Rated and Predicted Static Thrust for the 9k50 and M53 at Sea Level..................... 58

Table 7.3: Flight conditions for Range estimation 67

University of Cape Town

List of Symbols

Roman

A	:	Area
C_p	:	Specific heat
D	:	Drag
F	:	Thrust
f	:	Fuel/air ratio
g	:	Gravitational acceleration
h	:	Enthalpy or Height
L	:	Lift
M	:	Mach number
m	:	Mass
n	:	Load factor
p	:	Pressure
Q_R	:	Lower heating value of fuel
R	:	Gas constant or Range
r	:	Radius (capture streamtube)
S	:	Aerodynamic reference area
s	:	Entropy or Distance
T	:	Temperature or Thrust
t	:	Time
u	:	Fluid velocity
V	:	Speed
W	:	Weight
x	:	Linear position along conical centrebody axis

Aerodynamic Coefficients

C_L	:	Lift coefficient
C_D	:	Drag coefficient
C_T	:	Thrust coefficient
C_W	:	Weight coefficient

Subscripts

0	:	Actual intake capture streamtube
0-9	:	Station identifiers
a	:	Afterburner or Primary air stream entering gas generator
avg	:	Average
b	:	Burner or Bypass stream
c	:	Compressor or Cone surface or Capture
d	:	Diffuser or Design
f	:	Fan or Fuel
lip	:	Cowl / Intake lip position
n	:	Nozzle
t	:	Stagnation value or Turbine

University of Cape Town

∞	:	Free stream

<u>Greek</u>

β	:	Bypass ratio or Conical shock angle
δ	:	Stagnation / static pressure ratio
ε	:	Drag to Lift ratio or Shock angle (Taylor-Macoll flow)
γ	:	Specific heat ratio
η	:	Efficiency
π	:	Stagnation pressure ratio
θ	:	Stagnation / static temperature ratio
ρ	:	Density
τ	:	Stagnation temperature ratio

University of Cape Town

Chapter 1: Introduction

Recent advances in the areas of computational fluid dynamics, stability and control, structural and material design have had a great impact on the aircraft industry. In particular, the design of combat aircraft, which have always been at the forefront of technology, has benefited greatly from innovations stemming from these advances.

New fighter aircraft are designed to be operated into the post-stall region to afford high agility, but remain controllable through advanced aerodynamic design and control systems. New combat aircraft engines provide greater thrust to weight ratios and exhibit lower fuel consumption [1]. More advanced technologies such as *thrust vectoring* [2] provide for an even more manoeuvrable aircraft.

New developments in armaments have also taken place with a wide selection of short-range air-to-air and ground-to-air missiles available. These weapons typically employ advanced guidance/seeking systems which render them lethal and allow them to be *all aspect* weapons [1]. Developments have also been made in the field of all aspect guns. Because these weapons need not be fired with the aircraft pointing towards the target, the traditional combat scenario of gaining an advantage over the opponent and firing from behind no longer applies to the same extent as it did in the 1950's through 1980's.

New combat aircraft are typically capable of supersonic flight without the use of an afterburner. This capability is termed *supercruise* and has arisen from the development of propulsion unit technology. Older aircraft do not have this capability and rely on the utilisation of an afterburner to gain the necessary thrust to perform supersonic flight.

Thus, older aircraft are becoming increasingly more vulnerable in short-range combat. New aircraft employing the latest technology are expensive, and generally subject to export restrictions from the country of manufacture. One component of these older aircraft that can be readily upgraded, however, is the propulsion unit. Newer propulsion units offer the high thrust to weight ratios and low fuel consumption that the new aircraft have as standard.

However, combat performance is the integrated response of aerodynamics, avionics, structural strength and propulsion. The investigation described in this report therefore aims to determine what benefits and, possibly, drawbacks upgrading only the engine would have for an aircraft with an old airframe design. I.e. would an old airframe reap any tactical advantages by having supercruise capability?

University of Cape Town

In order to perform this task, a simulation model had to be developed. An overview of the simulation model will be given in Chapter 3. The airframe, aerodynamics and propulsion unit are identified as key components of this model. Chapter 3 also describes some of the performance metrics identified as being indicative of the performance of an engine, and engine-airframe combination.

The Dassault Mirage III was chosen as the basis of a hypothetical aircraft due to its age (the first Mach 2 capable prototype was rolled out in 1956 [3]) and the large number that were deployed by numerous airforces around the world, making it one of the most successful fighter aircraft ever produced. Two engines from French manufacturer SNECMA were chosen for evaluation, the Atar 9k50 and the M53. The 9k50, a pure afterburning turbojet, was deployed in the Mirage III fleet obtained by South Africa and subsequently utilised in the Cheetah. The M53 was first used in the Mirage 2000 and is a low bypass ratio afterburning turbofan engine.

The development of the propulsion model is detailed in Chapter 4. The model developed is one-dimensional and ignores three-dimensional flow effects. Instead the development concentrates on the thermodynamic cycle and a theoretical analysis of it to arrive at a set of equations which fully describe its behaviour. Some of the key factors affecting a turbojet's performance are detailed through some simple simulation tests. The software implementation is described at the end of the chapter.

The analysis of the propulsion unit identifies the air intake as a key factor in the performance of the propulsion unit. Chapter 5 deals with this complex component. The various modes of operation are described and a numerical method for determining both the pressure recovery and mass flow is developed. A brief description of the software implementation of this model is then provided.

Chapter 6 provides some background on basic aerodynamic theory in order to put into context the results obtained later in the chapter. The use of a numerical tool developed by the US Airforce for estimating stability and control parameters, the Digital DATCOM, is described. The model parameters and software for output processing from the DATCOM are described, along with some of the primary aerodynamic coefficient results obtained.

The combined model results are detailed in Chapter 7. Results from the combined intake and propulsion unit models are used to compare two engine models, and an explanation of the differences in their performance is given. The combined aerodynamic, intake and propulsion model results are then presented, highlighting the changes in overall performance due a change in engine.

University of Cape Town

Chapter 8 describes some of the conclusions that can be drawn from the results obtained and offers some suggestions with regard to areas that require more detailed study and how the model can be improved.

University of Cape Town

Chapter 2: Background

This chapter provides a brief background to developments in military aircraft propulsion and some of the developments in air combat practices.

2.1 Military Aircraft Propulsion

Early fighter aircraft were developed with pure turbojet engines. A reheat, or afterburner, section was generally provided to boost the maximum thrust that the propulsion unit could deliver. In general, these engines provided insufficient dry (without afterburning) thrust to achieve supersonic cruising [4].

Furthermore, these aircraft tended to have specialised roles such as, for example, bombing, combat, interception and reconnaissance. Combat typically took place at high subsonic speeds at fairly low altitudes in a head-to-tail chase arrangement [1].

University of Cape Town

Developments in military aircraft propulsion have focussed around providing engines which have higher thrust to weight ratios (see Figure 2.1) and higher thrust per unit airflow through the engine [4]. Maximum speed has not changed greatly due to limitations imposed by supersonic aerodynamics and aerodynamic heating due to high stagnation temperatures associated with high supersonic flow (see Figure 2.2). Thus, higher thrust to weight ratios and higher thrust per unit airflow allow a smaller, lighter engine to be utilised. This reduces the wing loading and allows an aircraft to have increased agility.

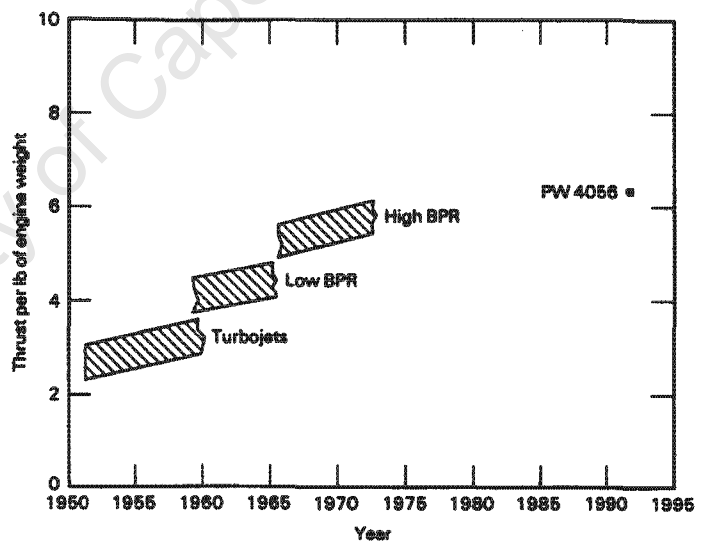


Figure 2.1: Trends in Thrust to Weight Ratio

Source: [5]

Modern combat aircraft also tend to have multi-role requirements [4]. This multi-role capability imposes supersonic flight and cruise requirements on the aircraft. The supersonic flight capability can be achieved at the expense of engine mass. By utilising the gains in thrust to weight ratio and thrust per unit airflow, a modern engine of the same dimensions and mass as an older engine can deliver significantly more thrust. This ability to cruise at supersonic speeds without the use of an afterburner is termed *supercruise*.

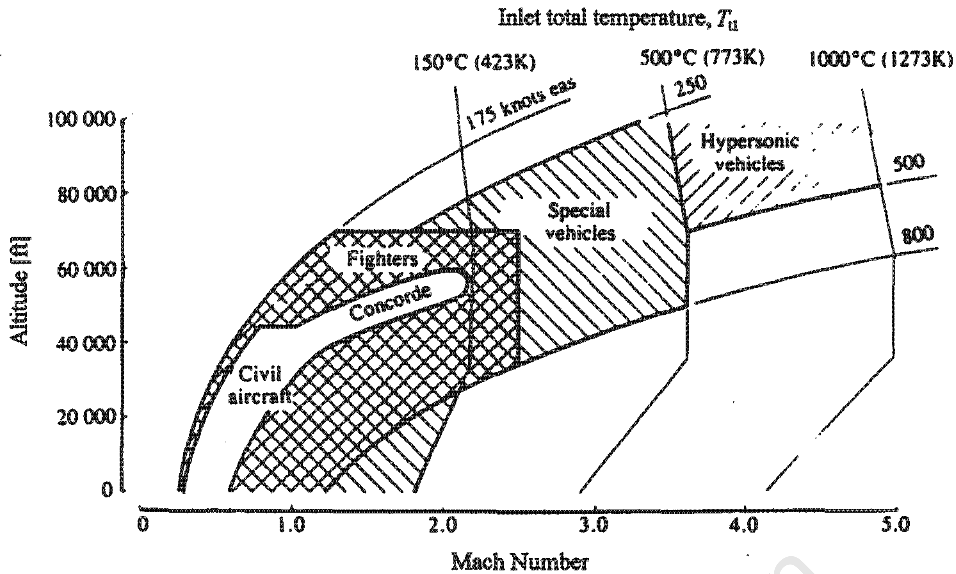


Figure 2.2: Typical Aircraft Operating Envelopes and Stagnation Temperature limits.
Source: [4]

These gains in engine performance have been achieved by moving from pure turbojet engines to low bypass ratio turbofans (see Figure 2.3), with or without an afterburner. In addition, new materials and design techniques a have resulted in more efficient and higher pressure-ratio compressors, and turbines with higher working temperatures. Old engines such as the SNECMA 9k50 operate with compressor pressure ratios of around 6 and turbine inlet temperatures around 1200K. Modern engines can have compressor pressure ratios in excess of 10 and turbine inlet temperatures approaching the stoichiometric burning temperature of the fuel (approximately 2400K) [4].

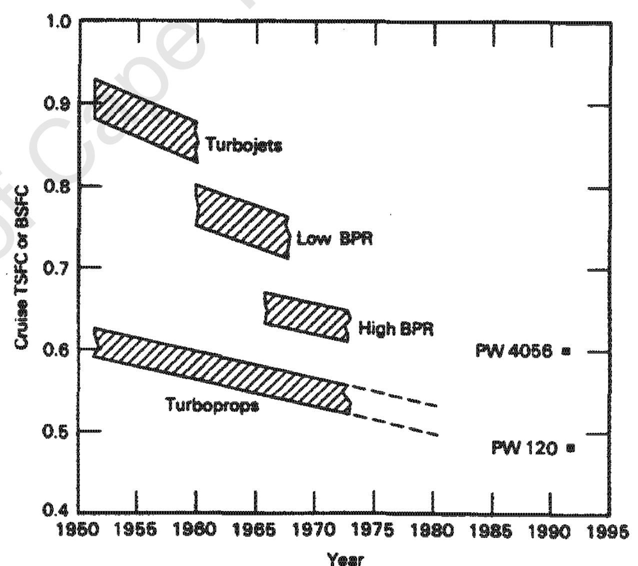


Figure 2.3: Trends in TSFC
Source: [5]

2.2 Air Combat

Traditional combat scenarios have typically involved a head-to-tail chase arrangement with firing on the target taking place from the rear. Armaments utilised were fired with the aircraft nose pointing towards the rear of the target, in a similar flight path. These weapons are termed *rear-aspect* weapons. Combat of this nature required high sustainable performance in, for

University of Cape Town

example, a turn, in order to maintain a position of superiority over the opponent. High thrust to weight ratios were therefore critical [1,6].

Recent developments in air-to-air weapons and fire control systems have resulted in *all-aspect* capability short-range missiles and guns. These weapons no longer need to be fired in the classical head-to-tail configuration. In fact, Herbst [1] has shown that these weapons are far better utilised in head-to-head combat where they score high hit rates.

Furthermore, all-aspect weapons leave no part of an aircraft safe from attack - attacks can be made from all directions. The best way to avoid being shot down is to respond aggressively and achieve a position from which to fire before the opponent does. The ability to respond rapidly requires the utilisation of unsustainable transient performance in order to gain an advantage over the opponent. The energy lost through these manoeuvres can be recovered at a later stage [1,6]. This has, in recent years, placed more emphasis on high agility aircraft, and has placed greater requirements on the aerodynamic and avionic performance of the aircraft. The performance of the aircraft in combat is now somewhat less sensitive to thrust to weight ratio, but it is still important in order to be able to recover lost energy as rapidly as possible.

In general, combat manoeuvres require a constant interchange of potential and kinetic energy. High engine performance is essential for [1]:

- Turning: excess thrust is needed for high drag.
- Climbing: excess power is required for the accumulation of potential energy.
- Acceleration: excess power is required for the accumulation of kinetic energy.

University of Cape Town

Chapter 3: Simulation Overview

This chapter describes the approach utilised for the numerical modelling of the aircraft's performance. The indicators used to evaluate the performance are described along with a brief description of how they are calculated. The manner in which the aircraft is discretised for modelling purposes is also covered.

3.1 Performance Indicators

3.1.1 Range

The maximum flight range of a fighter aircraft is an important tactical parameter since it determines the furthest distance from which an airforce can launch a strike against an enemy. The range for a military aircraft is usually determined by a mission profile [5]. However, a simple estimate of the cruising range at constant altitude and airspeed is given by the *Breguet range equation* for propeller driven aircraft, and the modified Breguet equation for turbojet aircraft.

While the aircraft is cruising in level flight, the engine provides sufficient thrust only to counteract the drag. The drag in turn is dependent on the required lift, which is in turn dependent on the weight of the aircraft. As the aircraft burns fuel, the total mass, and therefore the total lift required is reduced. The range of the aircraft can therefore be calculated as follows:

The fuel weight consumption rate is given by Eq. 3.1

$$\dot{W}_f = (TSFC)D \qquad \textbf{Eq. 3.1}$$

where (TSFC) = Thrust specific fuel consumption [N/Ns]

D = Drag [N]

The weight of the aircraft then varies according to:

$$\frac{dW}{W} = -\frac{(TSFC)\cdot\varepsilon\cdot ds}{V} \qquad \textbf{Eq. 3.2}$$

where ε = Drag to Lift ratio

V = Cruising speed

s = Distance

University of Cape Town

Assuming that the aircraft operates at constant ε/V and TSFC, Eq. 3.2 can be integrated [5] to yield the modified Breguet range equation for turbojet driven aircraft:

$$R = \frac{V}{(TSFC)\varepsilon}\ln\left(1+\frac{W_F}{W_E}\right) \quad \textbf{Eq. 3.3}$$

where: W_F = Weight full

W_E = Weight empty

3.1.2 Thrust Specific Fuel Consumption

Thrust Specific Fuel Consumption (TSFC) is a measure of the rate at which fuel is consumed for each unit of thrust generated. Given two turbojet engines, the one with the lower TSFC value will consume less fuel for a given thrust output. This has implications for range, as discussed in Section 3.1.1.

$$TSFC = \frac{\dot{m}_f}{F} \quad \textbf{Eq. 3.4}$$

where: $\dot{m}_f$ = Fuel flow rate

F = Engine thrust

The calculation of TSFC from the thermodynamic cycle for a turbojet engine with a bypass stream will be discussed in more detail in Section 4.4.3.

3.1.3 Specific Excess Power

The rate of climb for an aircraft can be derived from the equations of motion [5] as:

$$\frac{dh}{dt} = V\left[\frac{T-D}{W} - \frac{1}{g}\cdot\frac{dV}{dt}\right]$$

or

$$(T-D)V = W\frac{dh}{dt} + \frac{d}{dt}\left(\frac{W}{2g}V^2\right)$$

writing

$$h_e = h + \frac{V^2}{2g}$$

h_e represents the total energy per unit weight of the aircraft, i.e. both the kinetic and potential energy.

University of Cape Town

The Specific Excess Power (SEP) is then given by

$$SEP = \frac{dh_e}{dt} = \frac{V(T-D)}{W}$$ **Eq. 3.5**

It can be seen that $T \cdot V$ is the available power and that $D \cdot V$ is the required power [5], the difference being the *excess* power. This excess power can be used either to climb or to accelerate and is therefore an important parameter for combat aircraft.

Eq. 3.5 also shows that for two aircraft with the same airframe, SEP is affected primarily by the thrust term T, but also through W which is affected by engine mass. In addition, the lift required is affected by W and therefore the drag term D is also affected by engine mass. A lighter engine therefore has benefits for the aircraft's performance in terms of SEP.

3.1.4 Sustained Turn Rate and Attained Turn Rate

The Sustained Turn Rate (STR) of an aircraft is the maximum constant rate at which it can turn, at constant altitude, forward speed and bank angle. This parameter is critical for combat aircraft since it influences engagement with enemy aircraft, as will be shown later. Sustained turn rate is calculated from Eq. 3.6.

$$STR = g\sqrt{\frac{\rho_\infty C_L (n^2 - 1)}{2n(W/S_W)}}$$ **Eq. 3.6**

where n is the load factor and is given by Eq. 3.7:

$$n = \frac{L}{W}$$ **Eq. 3.7**

for a condition where the thrust matches the drag, i.e. $T = D$, Eq. 3.7 can be written as:

$$\begin{aligned} n &= \frac{L}{D} \cdot \frac{T}{W} \\ &= \frac{C_L}{C_D} \cdot \frac{C_T}{C_W} \end{aligned}$$ **Eq. 3.8**

In order to compare the performance of different aircraft engines, a plot of STR vs. Mach number can be generated [6]. The following algorithm is used to calculate this plot:

For a given altitude and free stream Mach number, M_∞, the thrust coefficient, C_T can be determined from the propulsion model. From the aerodynamic performance data, the angle of attack at which a matching drag coefficient equal to the thrust coefficient occurs can be determined. The value of the lift coefficient at this angle of attack can then be determined. The STR can then be calculated from Eq. 3.6.

University of Cape Town

The Attainable Turn Rate (ATR) is primarily governed by aerodynamic and structural strength considerations. ATR is the highest achievable turn rate that the aircraft is capable of, although it might not be sustainable. ATR is calculated from the following algorithm:

For a particular Mach number, the maximum value for C_L and its corresponding C_D are determined. A value for weight coefficient, C_W, is calculated. C_L/C_D and n (from Eq. 3.7) are calculated. If n exceeds the structural limit of the airframe, it is set to n_{max}, and a new value, C_{Leff} is calculated as the required lift coefficient. The ATR is then calculated from Eq. 3.9, which is seen to be similar to Eq. 3.6.

$$ATR = g\sqrt{\frac{\rho_{\infty} C_{Leff}(n^2 - 1)}{2n(W/S_W)}} \qquad \text{Eq. 3.9}$$

Figure 3.1 illustrates a simulated short-range combat scenario. It can be clearly seen how the pilot utilises the attainable turn rate limits, and how he then flies below the STR line in order to regain energy lost by flying above the STR line.

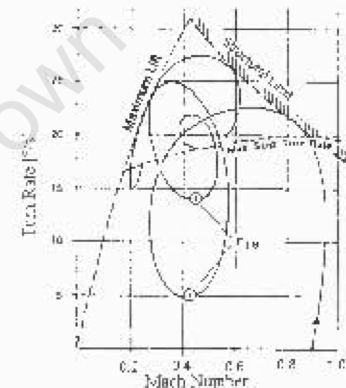


Figure 3.1: STR and ATR in simulated short range combat

Source: [1]

The figure also clearly shows the two limits which bound the attainable turn rate: The maximum lift condition at low Mach numbers and the maximum load factor, or structural limit, at higher Mach numbers.

Furthermore, the maximum attainable turn rate occurs at relatively low Mach numbers and is significantly larger than the sustained rate.

University of Cape Town

3.1.5 Thrust - Drag - Load Factor chart

At a given altitude and Mach number, a value for C_L can be calculated which yields flight at a given load factor, n, as given by Eq. 3.7.

Utilising the aerodynamic data for the airframe, values for C_D can be determined from the calculated values for C_L. These can be plotted against Mach number, as lines of constant n, as shown in Figure 7.12. Values of the thrust coefficient, C_T can then be superimposed on this graph.

From this plot, the maximum attainable Mach number in level flight can be determined as the intersection of the C_T and $C_{D\,n=1}$ curves. The maximum attainable load factor can also be determined. Thus by plotting the C_T curves for various engines, a quick comparison of their performance can be obtained.

3.2 Modelling the entire Aircraft

An aircraft in flight can be viewed as a free body with only four primary forces acting upon it. These are *Lift* opposing *Weight* in the vertical direction, and *Drag* opposing *Thrust* in the horizontal direction. Of these, the Lift and Drag components are dependent on the aircraft's aerodynamic properties, while weight is governed by the airframe and fuel consumption. Thrust is dependent upon the output of the engine. The aircraft model can therefore be broken down into the following components:

Airframe

Of the three aircraft sub-models, the airframe is the simplest to model for the applications listed in Section 3.1 above. The required parameters are:

- Airframe mass (including or excluding armaments).
- Current fuel load.
- Maximum load factor that the airframe can withstand.

Aerodynamic Model

The aerodynamic model must provide values for the lift and drag coefficients (C_L and C_D respectively) at various Mach numbers and altitudes.

The trimmed coefficients are desired, as these would give the values of the parameters in steady flight. However, for the purposes of this investigation, the assumption can be made that the aircraft is neutrally stable and the coefficients can be utilised as calculated, without taking into account trim effects. This is acceptable because the same aerodynamic data is used for all the simulations, and no dynamic aerodynamic effects are being evaluated.

Propulsion Model

The propulsion model must provide values for the thrust and fuel consumption at various Mach numbers and altitudes.

The assumption is made that each of these aspects of the aircraft can be modelled independently and that the results can then be combined to yield an integrated simulated response for the entire aircraft.

University of Cape Town

Chapter 4: Propulsion Model

4.1 Turbojet Model

The turbojet model used for the development of a one-dimensional propulsion model is based on a turbojet model developed by Kerrebrock [7], but has been modified to account for a bypass stream. Figure 4.1 shows a schematic diagram on which the model and thermodynamic analysis are based.

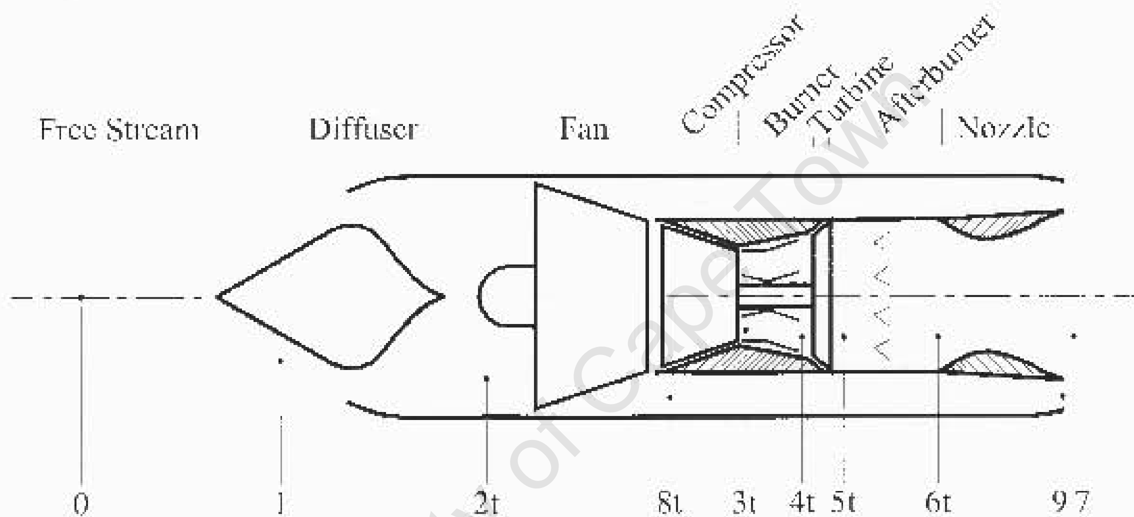


Figure 4.1: Schematic of a turbojet engine with bypass.

4.2 Typical Thermodynamic Cycle

The ideal cycle for the turbojet engine is the *Brayton* cycle [8]. A typical T-s diagram for this cycle, modified for a bypass stream, and including component losses, is shown in Figure 4.2. Station numbers correspond to those shown in Figure 4.1, with isentropic end states indicated by numbers subscripted with 's'.

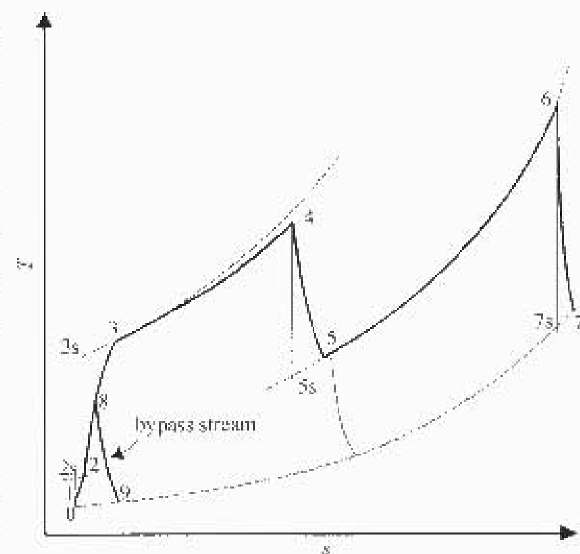


Figure 4.2: Typical T-s diagram for the Brayton cycle, with bypass and losses

University of Cape Town

4.3 Engine components and properties

Table 4.1: Turbojet engine components and properties

COMPONENT	STAGE	PRESSURE RATIO	TEMP RATIO	EFFICIENCY
Diffuser	0-2	$\pi_d = \frac{p_{t2}}{p_{t0}} = \frac{p_{t2}}{p_0 \delta_0}$	$\frac{T_{t2}}{T_{t0}} = 1$ (adiabatic)	$\eta_d = \frac{h_{t2s} - h_{t0}}{h_{t2} - h_{t0}}$
Fan	2-8	$\pi_f = \frac{p_{t8}}{p_{t2}}$ (known)	$\tau_f = \frac{T_{t8}}{T_{t2}}$	$\eta_f = \frac{h_{t8s} - h_{t2}}{h_{t8} - h_{t2}} = \frac{\pi_f^{(\gamma_c - 1)/\gamma_c} - 1}{(\tau_f - 1)}$
Compressor	2-3	$\pi_c = \frac{p_{t3}}{p_{t2}}$ (known)	$\tau_c = \frac{T_{t3}}{T_{t2}}$	$\eta_c = \frac{h_{t3s} - h_{t2}}{h_{t3} - h_{t2}} = \frac{\pi_c^{(\gamma_c - 1)/\gamma_c} - 1}{(\tau_c - 1)}$
Burner	3-4	$\pi_b = \frac{p_{t4}}{p_{t3}}$ (assumed)	$\tau_b = \frac{T_{t4}}{T_{t3}}$	$\eta_b = \frac{C_{p_b}[(\dot{m}_a + \dot{m}_{f_b})T_{t4} - \dot{m}_a T_{t3}]}{\dot{m}_{f_b} Q_R}$
Turbine	4-5	$\pi_t = \frac{p_{t5}}{p_{t4}}$	$\tau_t = \frac{T_{t5}}{T_{t4}}$	$\eta_t = \frac{h_{t4} - h_{t5}}{h_{t4} - h_{t5s}}$
Afterburner	5-6	$\pi_a = \frac{p_{t6}}{p_{t5}}$	$\tau_a = \frac{T_{t6}}{T_{t5}}$	$\eta_a = \frac{C_{p_a}[(\dot{m}_a + \dot{m}_{f_b} + \dot{m}_{f_a})T_{t6} - (\dot{m}_a + \dot{m}_{f_b})T_{t5}]}{\dot{m}_{f_a} Q_R}$
Nozzle	6-7		$\frac{T_{t7}}{T_{t6}} = 1$ (adiabatic)	$\eta_n = \frac{h_{t6} - h_{t7}}{h_{t6} - h_{t7s}}$

University of Cape Town

4.4 Estimating the Thrust

4.4.1 Modelling assumptions and simplifications

Stagnation conditions

The assumption is made that the flow velocity is sufficiently small at the station numbers suffixed with a 't' in Figure 4.1, that the thermodynamic state of the gas at these points is represented by the stagnation properties [9]. However, this is not actually the case, and internal velocities might be quite large [9].

Average specific heats

The assumption is made that the specific heats and specific heat ratio γ can be accurately represented by an average value for each component or groups of components.

Furthermore, the gas composition is assumed to remain constant, whereas the molar mass of the working fluid in fact changes with the addition of fuel in the burner and afterburner.

Adiabatic engine components

The engine components, especially the compressor and turbine are assumed to be adiabatic [9]. In modern engines, however, turbine blade cooling may result in heat transfer in the turbine [7]. Air cooling of the afterburner stage by the bypass stream might also be present [4]. Other components might also utilise air-cooling.

4.4.2 Calculation of the thrust per unit airflow into the gas generator

From manipulation of the momentum and energy laws [9], Eq. 4.1 can be derived:

$$F = \dot{m}_7 u_7 - \dot{m} u_0 + A_7(p_7 - p_0) + \dot{m}_9 u_9 + A_9(p_9 - p_0) \qquad \textbf{Eq. 4.1}$$

Let:

$\dot{m}_a$ – mass flow rate into gas generator, i.e. mass flow rate at station 3.

$$f = \frac{\dot{m}_f}{\dot{m}_a} = \text{fuel/air ratio} \qquad \textbf{Eq. 4.2}$$

$$\beta = \frac{\dot{m}_9}{\dot{m}_a} = \text{bypass ratio} \qquad \textbf{Eq. 4.3}$$

University of Cape Town

This gives:

$$\begin{aligned}\dot m_7 &= \dot m_f + \dot m_a \\ &= f\dot m_a + \dot m_a \\ &= \dot m_a(1+f)\end{aligned} \qquad \textbf{Eq. 4.4}$$

$$\dot m_9 = \beta \dot m_a \qquad \textbf{Eq. 4.5}$$

$$\begin{aligned}\dot m &= \dot m_a + \dot m_9 \\ &= \dot m_a(1+\beta)\end{aligned} \qquad \textbf{Eq. 4.6}$$

Also,

$$\dot m_7 = \rho_7 A_7 u_7 \qquad \textbf{Eq. 4.7}$$

Using the substitution $\rho = \dfrac{p}{RT}$, Eq. 4.7 can be rewritten as:

$$\begin{aligned}A_7 &= (1+f)\dot m_a \frac{1}{\rho_7 u_7} \\ \frac{A_7}{\dot m_a} &= (1+f)\frac{\rho_0 u_0}{\rho_7 u_7}\cdot\frac{1}{\rho_0 u_0} \\ &= (1+f)\frac{1}{\rho_0 u_0}\frac{p_0}{p_7}\frac{T_7}{T_0}\frac{R_t}{R_c}\frac{u_0}{u_7}\end{aligned} \qquad \textbf{Eq. 4.8}$$

where

R_t = gas constant for combustion products

R_c = gas constant for incoming air

Similarly for the bypass stream,

$$\begin{aligned}\dot m_9 &= \rho_9 A_9 u_9 \\ \frac{A_9}{\dot m_a} &= \beta\frac{1}{\rho_0 u_0}\frac{\rho_0}{\rho_9}\frac{u_0}{u_9} \\ &= \beta\frac{1}{\rho_0 u_0}\frac{p_0}{p_9}\frac{T_9}{T_0}\frac{u_0}{u_9}\end{aligned} \qquad \textbf{Eq. 4.9}$$

Rewriting Eq. 4.1, making substitutions with Eq. 4.2 to Eq. 4.6, an expression for the thrust per unit airflow into the gas generator is obtained.

$$\begin{aligned}\frac{F}{\dot m_a} &= \frac{\dot m_7}{\dot m_a}u_7 - \frac{\dot m}{\dot m_a}u_0 + \frac{A_7}{\dot m_a}(p_7 - p_0) + \frac{\dot m_9}{\dot m_a}u_9 + \frac{A_9}{\dot m_a}(p_9 - p_0) \\ &= (1+f)u_7 - (1+\beta)u_0 + \frac{A_7}{\dot m_a}(p_7 - p_0) + \beta u_9 + \frac{A_9}{\dot m_a}(p_9 - p_0)\end{aligned} \qquad \textbf{Eq. 4.10}$$

University of Cape Town

The ratios $A_7 / \dot{m}_a$ and $A_9 / \dot{m}_g$ are readily determined from Eq. 4.8 and Eq. 4.9 once the exit state of the fluid has been determined. In order to determine these unknown quantities, the energy balance in the engine will be considered, with reference to the T-s diagram given in Figure 4.2.

From compressible flow relations,

$$\theta_0 = \frac{T_{t0}}{T_0} = 1 + \tfrac{1}{2}(\gamma_c - 1)M_0^2 \qquad \text{Eq. 4.11}$$

$$\delta_0 = \frac{p_{t0}}{p_0} = [1 + \tfrac{1}{2}(\gamma_c - 1)M_0^2]^{\frac{\gamma_c}{\gamma_c - 1}} \qquad \text{Eq. 4.12}$$

The compressor pressure ratio and efficiency are known, so from the definition of the compressor efficiency:

$$\begin{aligned}
\eta_c &= \frac{h_{t3s} - h_{t2}}{h_{t3} - h_{t2}} \\
&= \frac{C_{p_c} T_{t2}\left[\left(\dfrac{p_{t3}}{p_{t2}}\right)^{\frac{\gamma_c - 1}{\gamma_c}} - 1\right]}{C_{p_c}(T_{t3} - T_{t2})} \\
&= \frac{\pi_c^{(\gamma_c - 1)/\gamma_c} - 1}{(\tau_c - 1)}
\end{aligned}$$

Rearranging this, an expression for the temperature ratio across the compressor is obtained:

$$\tau_c = \frac{T_{t3}}{T_{t2}} = \left[\pi_c^{(\gamma_c - 1)/\gamma_c} - 1\right] / \eta_c + 1 \qquad \text{Eq. 4.13}$$

The total temperature ratio across the diffuser is unity (adiabatic diffuser assumption), and thus T_{t3} may be found from:

$$\begin{aligned}
T_{t3} &= \frac{T_{t3}}{T_{t2}} \frac{T_{t2}}{T_{t0}} \frac{T_{t0}}{T_0} T_0 \\
&= \tau_c \theta_0 T_0 \qquad \text{Eq. 4.14}
\end{aligned}$$

The maximum turbine inlet temperature, T_{t4}, is known (determined by material limitations). The average temperature in the burner, as given by

$$\overline{T_b} = \tfrac{1}{2}(T_{t3} + T_{t4})$$

is used to determine an average specific heat for the burner, i.e. $C_{pb} = C_{ph}(\overline{T_b})$.

University of Cape Town

From the definition of burner efficiency [7],

$$\eta_b = \frac{C_{p_t}\left[(\dot{m}_a + \dot{m}_{f_b})T_{t4} - \dot{m}_a T_{t3}\right]}{\dot{m}_{f_b} Q_R}$$

$$= \frac{C_{p_t}\left[(1+f_b)T_{t4} - T_{t3}\right]}{f_b Q_R}$$

Rearranging this equation, an expression for the air/fuel ratio can be determined:

$$f_b = \frac{C_{p_t}(T_{t4} - T_{t3})}{Q_R \eta_b - C_{p_t} T_{t4}} \qquad \textbf{Eq. 4.15}$$

Now,

$$\tau_b = \frac{T_{t4}}{T_{t3}} \qquad \textbf{Eq. 4.16}$$

Using a similar reasoning as that for τ_c, Eq. 4.13, an expression for the temperature ratio across the bypass compression stage can be determined:

$$\tau_f = \frac{T_{t8}}{T_{t2}} = \left[\pi_f^{(\gamma_c - 1)/\gamma_c} - 1\right]/\eta_f - 1 \qquad \textbf{Eq. 4.17}$$

To determine the temperature ratio across the turbine, the energy balance in the compressor turbine system is considered:

$$\beta \dot{m}_a C_{p_c}(T_{t8} - T_{t2}) + \dot{m}_a C_{p_c}(T_{t3} - T_{t2}) = \dot{m}_a(1+f_b)C_{p_t}(T_{t4} - T_{t5})$$

$$\frac{C_{p_c}}{C_{p_t}} T_{t2}\left[\beta\left(\frac{T_{t8}}{T_{t2}} - 1\right) + \left(\frac{T_{t3}}{T_{t2}} - 1\right)\right] = (1+f_b)T_{t4}\left(1 - \frac{T_{t5}}{T_{t4}}\right)$$

$$\frac{C_{p_c}}{C_{p_t}} \frac{1}{(1-f_b)}\left[\beta(\tau_f - 1) + (\tau_c - 1)\right] = \tau_b \tau_r (1 - \tau_t)$$

Solving for τ_t,

$$\tau_t = 1 - \frac{C_{p_c}}{C_{p_t}} \frac{1}{(1-f_b)} \frac{1}{\tau_b \tau_r}\left[\beta(\tau_f - 1) + (\tau_c - 1)\right] \qquad \textbf{Eq. 4.18}$$

If the afterburner is operational, then T_{t6} will be known from the afterburner performance data (this is again limited by the materials and design of the afterburner). If the afterburner is not operational, then $T_{t6} = T_{t5}$. Since the nozzle is assumed to be adiabatic,

$$T_{t7} = T_{t6} \qquad \textbf{Eq. 4.19}$$

University of Cape Town

Analogous to Eq. 4.15, an expression for the fuel/air ratio for the afterburner can be developed:

$$f_a = \frac{C_{p_t}(1+f_b)(T_{t6}-T_{t5})}{Q_R\eta_a - C_{p_t}T_{t6}} \qquad \textbf{Eq. 4.20}$$

The total fuel/air ratio is then simply

$$f = f_b + f_a \qquad \textbf{Eq. 4.21}$$

From the definition of turbine efficiency,

$$\eta_t = \frac{h_{t4}-h_{t5}}{h_{t4}-h_{t5s}} = \frac{C_{p_t}(T_{t4}-T_{t5})}{C_{p_t}(T_{t4}-T_{t5s})} = \frac{\left(1-\dfrac{T_{t5}}{T_{t4}}\right)}{\left(1-\dfrac{T_{t5s}}{T_{t4}}\right)}$$

$$= \frac{(1-\tau_t)}{1-\left(\dfrac{p_{t5s}}{p_{t4}}\right)^{\frac{\gamma_t-1}{\gamma_t}}}$$

Rearranging,

$$\pi_t = \frac{p_{t5}}{p_{t4}} = [1-(1-\tau_t)/\eta_t]^{\frac{\gamma_t}{\gamma_t-1}} \qquad \textbf{Eq. 4.22}$$

The stagnation pressure at station 6 can now be determined from the following identity:

$$p_{t6} = p_0 \frac{p_{t0}}{p_0}\frac{p_{t2}}{p_{t0}}\frac{p_{t3}}{p_{t2}}\frac{p_{t4}}{p_{t3}}\frac{p_{t5}}{p_{t4}}\frac{p_{t6}}{p_{t5}}$$

$$= p_0\delta_0\pi_d\pi_c\pi_b\pi_t\pi_a \qquad \textbf{Eq. 4.23}$$

where π_a, π_b are the stagnation pressure losses due to heat addition and are assumed or estimated (close to unity). π_d is calculated from the diffuser pressure recovery.

There are two limiting cases for the exhaust state [7]:

1. The nozzle is fully expanded, and $p_7 = p_0$.
2. The nozzle is choked and $M_7 = 1$.

University of Cape Town

Case 1: Fully expanded nozzle

The enthalpy drop of the exhaust gases at the exit determines the exhaust velocity.

Thus:

$$\frac{u_7^2}{2} = h_{t6} - h_{t7} = \eta_n C_{p_n}(T_{t6} - T_{t7})$$

$$= \eta_n C_{p_n} T_{t6}\left[1 - \left(\frac{p_7}{p_{t6}}\right)^{(\gamma_n - 1)/\gamma_n}\right]$$

or

$$u_7 = \sqrt{2\eta_n C_{p_n} T_{t6}\left[1 - \left(\frac{p_7}{p_{t6}}\right)^{(\gamma_n - 1)/\gamma_n}\right]} \qquad \textbf{Eq. 4.24}$$

$$T_7 = T_{t6} - \frac{u_7^2}{2C_{p_n}} \qquad \textbf{Eq. 4.25}$$

$$M_7 = \frac{u_7}{\sqrt{\gamma_n R_n T_7}} \qquad \textbf{Eq. 4.26}$$

Case 2: Nozzle choked and $M_7 = 1$

From Eq. 4.26:

$$u_7^2 = M_7^2 \gamma_n R_n T_7$$

substituting into Eq. 4.25:

$$M_7^2 \gamma_n R_n T_7 = 2C_{p_n}[T_{t6} - T_7]$$

$$T_7 = \frac{2C_{p_n} T_{t6}}{[M_7^2 \gamma_n R_n + 2C_{p_n}]} \qquad \textbf{Eq. 4.27}$$

u_7 can now be found from Eq. 4.25 or Eq. 4.26.

From Eq. 4.24,

$$p_7 = p_{t6}\left[1 - \frac{u_7^2}{2\eta_n C_{p_n} T_{t6}}\right]^{\frac{\gamma_n}{\gamma_n - 1}} \qquad \textbf{Eq. 4.28}$$

A similar approach can be adopted to determine the exit condition of the bypass stream.

University of Cape Town

Case 1: Fully expanded nozzle

Analogous to Eq. 4.24,

$$u_9 = \sqrt{2\eta_{n_b} C_{p_c} T_{t8}\left[1-\left(\frac{p_9}{p_{t8}}\right)^{(\gamma_c-1)/\gamma_c}\right]} \qquad \textbf{Eq. 4.29}$$

Analogous to Eq. 4.25,

$$T_9 = T_{t8} - \frac{u_9^2}{2C_{p_c}} \qquad \textbf{Eq. 4.30}$$

Case 2: Choked bypass stream nozzle, $M_9 = 1$.

Analogous to Eq. 4.27,

$$T_9 = \frac{2C_{p_c}T_{t8}}{[M_9^2\gamma_c R_c + 2C_{p_c}]} \qquad \textbf{Eq. 4.31}$$

Analogous to Eq. 4.28,

$$p_9 = p_{t8}\left[1-\frac{u_9^2}{2\eta_{n_b}C_{p_c}T_{t8}}\right]^{\frac{\gamma_c}{\gamma_c-1}} \qquad \textbf{Eq. 4.32}$$

4.4.3 Calculation of Net Thrust Output

In section 4.4.2 an expression for the thrust *per unit airflow* into the gas generator was derived as given by Eq. 4.10. In order to determine the net thrust output, F, the mass flow parameter $\dot{m}_a$ must be determined.

The mass flow through the engine is controlled by the following mechanisms:

- **The intake chokes the flow**

  In this case, the mass flow rate into the engine is limited by the capture streamtube diameter, free stream density and Mach number. Determination of the mass flow in this condition will be deferred to Section 5.4.

- **The exhaust nozzle chokes the flow**

  In this case the mass flow through each nozzle (primary and bypass) is determined by the exit conditions as given in Eq. 4.8 and Eq. 4.9 which provide a relationship between the exit nozzle areas and $\dot{m}_a$. A maximum nozzle area is specified, and it is assumed that this area is variable between this maximum and some smaller value. The maximum mass flow rate for a given thermodynamic exit condition can therefore be calculated.

University of Cape Town

- **The engine chokes the flow**

  The engine components such as the compressor, burner and turbine are capable of passing certain maximum mass flow rate. This flow rate is given as a specification by the manufacturer.

- **The compressor 'suction' at low free stream velocities**

  When the engine is stationary, the mass flow as given by the product of the capture streamtube diameter and forward velocity is zero. The compressor, however, acts as a 'pump' in this condition and a certain mass flow is achieved.

The mass flow through the engine is therefore determined in the following manner:

- **Subsonic Flight Regime:**

$$If\ \dot{m}_{diff} < \dot{m}_{pump}\ \ then\ \ \dot{m} = \dot{m}_{pump}$$

$$Else\ \ \dot{m} = \dot{m}_{diff}$$

$$If\ \dot{m} > \dot{m}_{exh}\ \ then\ \ \dot{m} = \dot{m}_{exh}$$

$$If\ \dot{m} > \dot{m}_{max}\ \ then\ \ \dot{m} = \dot{m}_{max}$$

where,

$\dot{m}_{diff}$ = maximum flow through the diffuser

$\dot{m}_{pump}$ = mass flow that the engine is capable of 'pumping'

$\dot{m}_{exh}$ = maximum mass flow through the exhaust

$\dot{m}_{max}$ = maximum mass flow through the compressor & turbine

- **Supersonic Flight Regime:**

  In this case, the lesser of $\dot{m}_{diff}$, $\dot{m}_{exh}$, $\dot{m}_{max}$ determines the total net flow through the engine, and hence the resultant thrust.

4.5 Engine Performance Indicators

4.5.1 Thrust Specific Fuel Consumption (TSFC) and Specific Impulse

A frequently used measure of engine performance is the thrust specific fuel consumption (TSFC) value. This value indicates the rate at which fuel is used for every unit of thrust produced. Thus,

University of Cape Town

$$TSFC = \frac{\dot{m}_f}{F} = \frac{f\dot{m}_a}{\dot{m}_a\left(\frac{F}{\dot{m}_a}\right)} = \frac{f}{\left(\frac{F}{\dot{m}_a}\right)}$$ **Eq. 4.33**

This parameter is easily calculated once Eq. 4.10 and Eq. 4.21 have been evaluated.

Specific impulse is the inverse of TSFC.

4.5.2 T-s Diagram Calculations

The thermodynamic cycle of a turbojet engine can be plotted on a T-s diagram such as Figure 4.2. After the temperatures and pressures at the various engine stations have been calculated, the entropy generation of each stage can be evaluated. In order to plot the T-s diagram, the entropy at each station was calculated as the sum of the entropy at the previous station and the entropy change across the relevant component:

$$s_i = s_{i-1} + C_p \ln\left(\frac{T_i}{T_{i-1}}\right) - R\ln\left(\frac{p_i}{p_{i-1}}\right)$$ **Eq. 4.34**

The isobaric lines on the T-s diagram can be plotted from Eq. 4.34, by setting $p_i = p_{i-1}$. This can then be rearranged as follows:

$$T = T_{ref}e^{(s-s_{ref})/C_p}$$ **Eq. 4.35**

The isobaric line passing through a particular station point can be plotted using Eq. 4.35 by choosing s_{ref} and T_{ref} to be the conditions at the station point, and then plotting T as a function of s.

4.6 Factors Affecting Engine Performance

This section provides the results of a brief parametric investigation into the factors affecting the overall performance of the engine.

4.6.1 Reference configuration for the parametric investigation

Table 4.2 provides a list of the engine parameters and their values utilised for the parametric investigation. These values are largely based upon the SNECMA 9k50 engine which will be discussed in more depth in Chapter 7.

University of Cape Town

Table 4.2 : Parameter values for the reference configuration

Parameter	Symbol	Value	Units
Altitude		0	m
Pressure ratios			
Fan	π_f	1	
Compressor	π_c	6.15	
Burner	π_b	0.97	
Afterburner	π_a	0.98	
Diffuser	π_d	AIA Curve	
Bypass Ratio	β	0	
Efficiencies:			
Fan	η_f	N/A	
Compressor	η_c	0.85	
Burner	η_b	0.96	
Turbine	η_t	0.90	
Afterburner	η_a	0.96	
Primary Nozzle	η_n	0.97	
Bypass Nozzle	η_{bn}	N/A	
Temperatures:			
Turbine Inlet Temperature	T_{t4}	1203	K
Maximum Afterburner Temperature	T_{t6}	2500	K
Heating value of fuel	Q_R	43 961.4	kJ/kg
Primary Nozzle Type		Convergent	
Bypass Nozzle Type		N/A	

University of Cape Town

4.6.2 Turbine Inlet Temperature

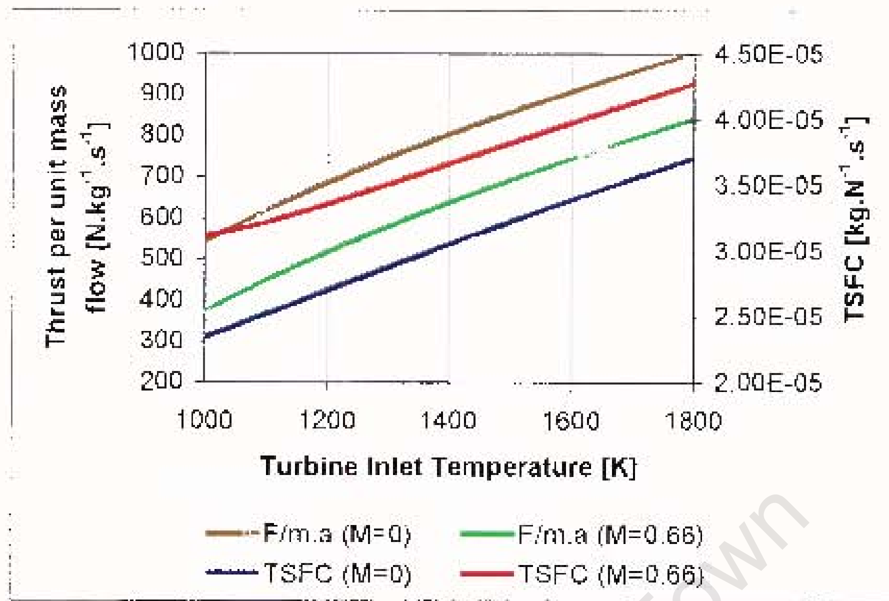


Figure 4.3: Variation of TSFC and $F/\dot{m}_a$ with turbine inlet temperature.

Figure 4.3 shows that both $F/\dot{m}_a$ and TSFC increase with T_{t4}. $F/\dot{m}_a$ however, increases at a greater rate than TSFC and thus higher turbine temperatures provide better engine performance.

4.6.3 Compressor Pressure Ratio

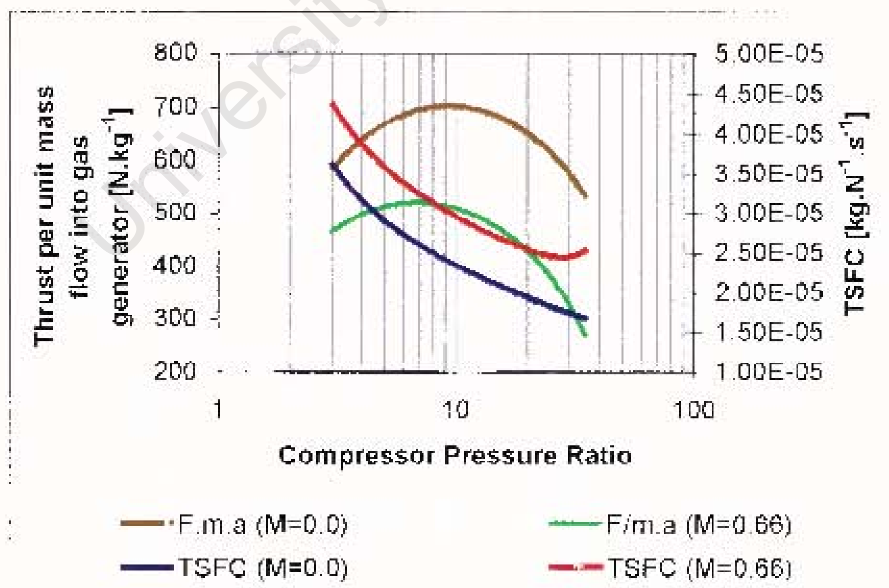


Figure 4.4: Variation of TSFC and $F/\dot{m}_a$ with compressor pressure ratio

University of Cape Town

Figure 4.4 shows that an optimal compressor pressure ratio exists for a given engine. $F/\dot{m}_a$ at first increases due to the gains from the higher compressor exit pressure, but then decreases as more energy is removed from the working fluid in order to power the compressor.

TSFC shows an overall decrease with an increase in the compressor pressure ratio, and this would be beneficial for extending the range of the aircraft.

The trends indicated in Figure 4.4 are similar to those found in Fig 6-14 of Hill and Petersen [9]. In order to check the model for errors, the parameters for Hill and Petersen's Fig 6-14 was entered into the propulsion model and closely matching results were obtained.

4.6.4 Intake Pressure Recovery

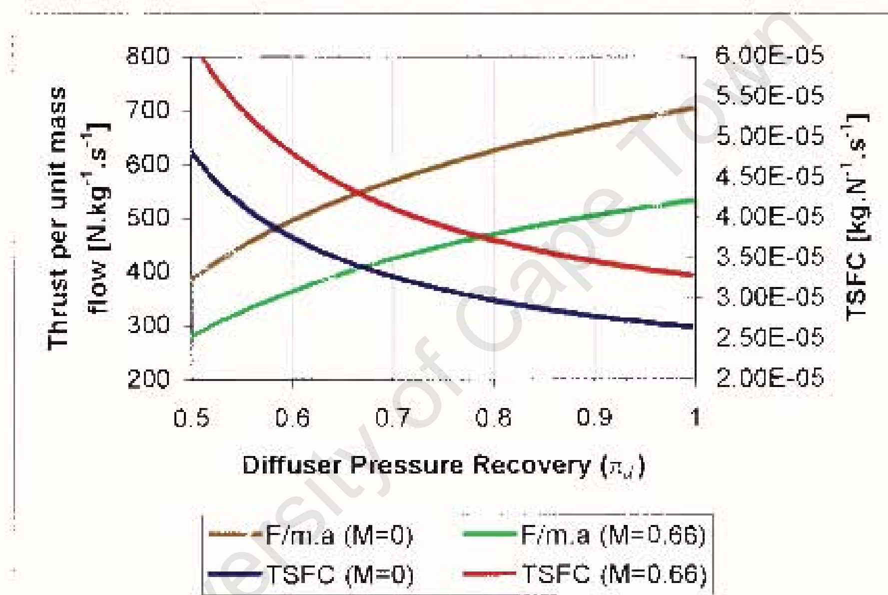


Figure 4.5: Variation of TSFC and $F/\dot{m}_a$ with diffuser pressure recovery.

An increase in the diffuser pressure recovery, π_d, is seen to increase $F/\dot{m}_a$ and decrease TSFC. An improvement in intake performance therefore benefits the overall performance of the engine and it is for this reason that intake performance is such a critical part of the propulsion system and must be designed correctly. The intake will be discussed in more detail in Chapter 5.

University of Cape Town

4.7 Software Implementation of the Propulsion Model

The one-dimensional propulsion model described in this chapter was coded into a C++ class representing the entire turbojet. The source code for this class, "TurboJet1D" can be found in Appendix C-2.

A flow chart indicating the algorithm for determining the net thrust output can be found in Appendix C-1.

The class stores all the data for the component efficiencies, maximum temperatures, pressure ratios, fuel energy, etc. i.e. all the data required to represent the turbojet. The `TurboJet1D` class also contains a `CConeIntake` object that describes the intake geometry and behaviour. The `CConeIntake` class will be discussed in Section 5.5.

The primary function in this class is `CycleThermodynamics`, and its parameters are given in Table 4.3. This function is called by the `TSFC` and `Thrust` functions to calculate the overall thermodynamics of the cycle. It follows the procedures, and utilises the equations, described in this chapter to perform its calculations.

University of Cape Town

Table 4.3: Parameters for Function CycleThermodynamics

	Parameter Name	Units	In/Out	Type
Return Value:	TSFC - Thrust Specific Fuel Consumption	kg/(N.s)	Out	Double precision
Arguments:	M0 - Free stream Mach number		In	Double precision
	T0 - Free stream temperature	K	In	Double precision
	p0 - Free stream pressure	Pa	In	
	ABOnOff - Afterburner: On = 1, Off = 0		In	Integer
	pi_d - diffuser pressure recovery		In	Double precision
	ex7 - exit stream properties at station 7		Out	CIdealGasStream
	ex9 - exit stream properties at station 9		Out	CIdealGasStream
	A7_ma - exit area to mass flow ratio	m²s/kg	Out	Double precision
	A9_ma - exit area to mass flow ratio	m²s/kg	Out	Double precision
	F_ma - Thrust to mass flow ratio	N s/kg	Out	Double precision
	f - fuel/air ratio		Out	Double precision

Chapter 5: Intake Modelling

In Chapter 4 it was shown that the intake performance is critical to the overall performance of the propulsion unit in terms of its pressure recovery and the mass flow that it permits. This section describes the manner in which the conical-centrebody, supersonic intake is modelled. It starts with an overview of intake operation modes and proceeds to describe how these are mathematically modelled. A method for determining the mass flow through the intake is also presented.

5.1 Overview of intake operation

Figure 5.1 shows a schematic of a typical supersonic, conical centrebody intake. The supersonic free stream airflow over the cone generates a conical shockwave. A second, normal, shock then occurs at the cowl lip, or in the diverging passage. This two-shock system, discussed in more detail later, results in a favourable stagnation pressure ratio from the free stream to the compressor face [10].

The centrebody can translate along the X-axis, in order to vary the design angle, β_d.

University of Cape Town

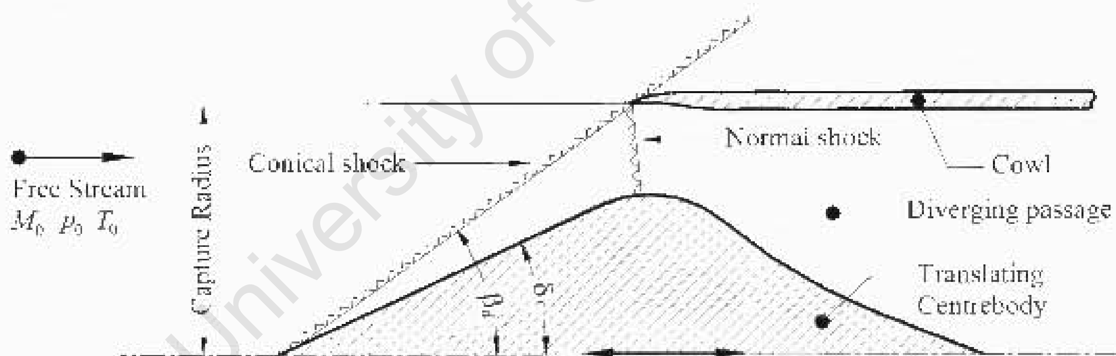


Figure 5.1: Schematic of a supersonic conical intake with two-shock system

A conical centrebody, supersonic intake, operates in one of the following modes, dependent on flight condition and turbomachine performance [10, 11]:

5.1.1 Subsonic flight

This mode of operation is effective in the free stream Mach number range from $0 \leq M_0 \leq 1$. The stagnation pressure recovery, π_d is nominally constant [10] and close to unity.

5.1.2 Detached Shock Mode

A detached bow shock is formed in the Mach number range from $1 \leq M_0 < M_{attach}$, where M_{attach} is the Mach number at which a shock wave attaches to the conical centrebody. This shock wave has a hyperbolic geometry and asymptotes towards the sonic (Mach) angle μ.

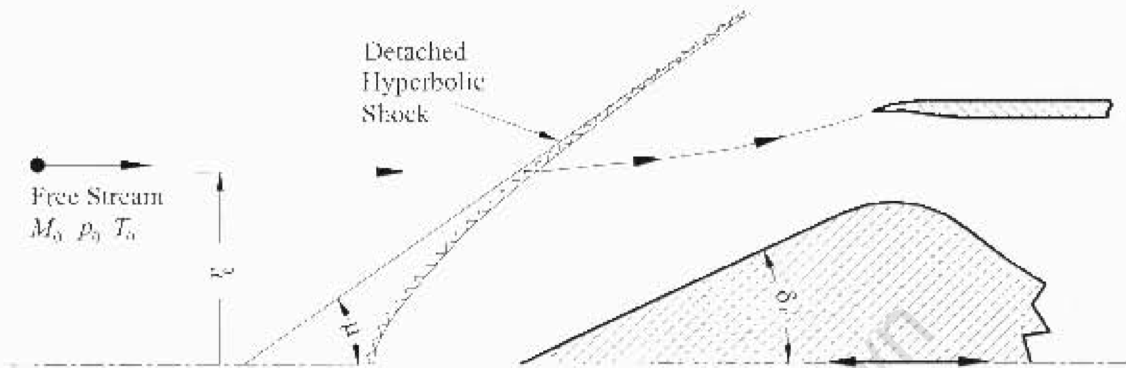


Figure 5.2: Detached shock arrangement

5.1.3 Critical Mode

Critical mode pressure recovery occurs when the intake is operating at its design condition. Therefore, the Mach number is given by the design Mach number M_d, which is variable and dependent upon the centrebody position.

The back pressure on the intake at the compressor face is such that the normal shock occurs at the cowl lip. This results in the capture stream tube having the same diameter as the cowl lip and the maximum possible mass flow rate through the intake is achieved.

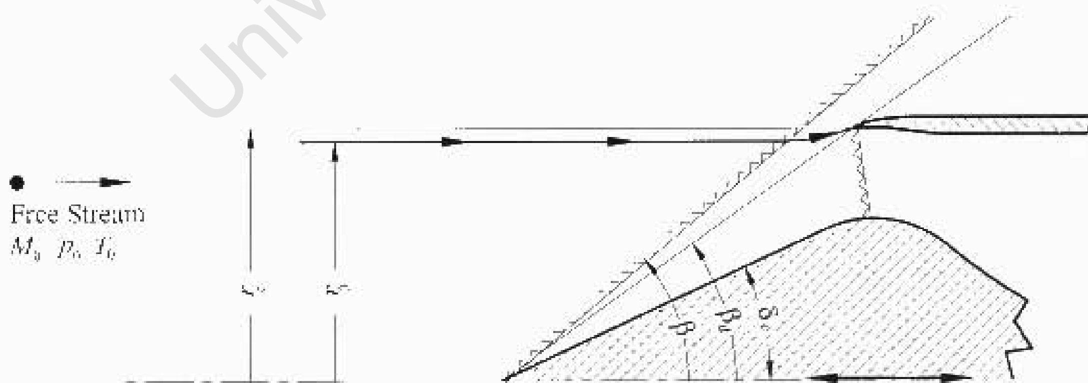


Figure 5.3: Critical mode shock arrangement

University of Cape Town

5.1.4 Sub Critical Mode

Sub critical mode operation occurs when the back pressure on the intake is too high. This results in the normal shock being pushed forward, outside the cowl. Part of the capture streamtube therefore passes through the two-shock system and the rest only passes through a stronger, single shock. The stagnation pressure recovery is therefore lower than at critical operation for the same Mach number. In addition, streamlines passing through the single shock are curved, and the capture streamtube diameter is therefore smaller than in critical mode operation, resulting in a lower mass flow rate through the intake.

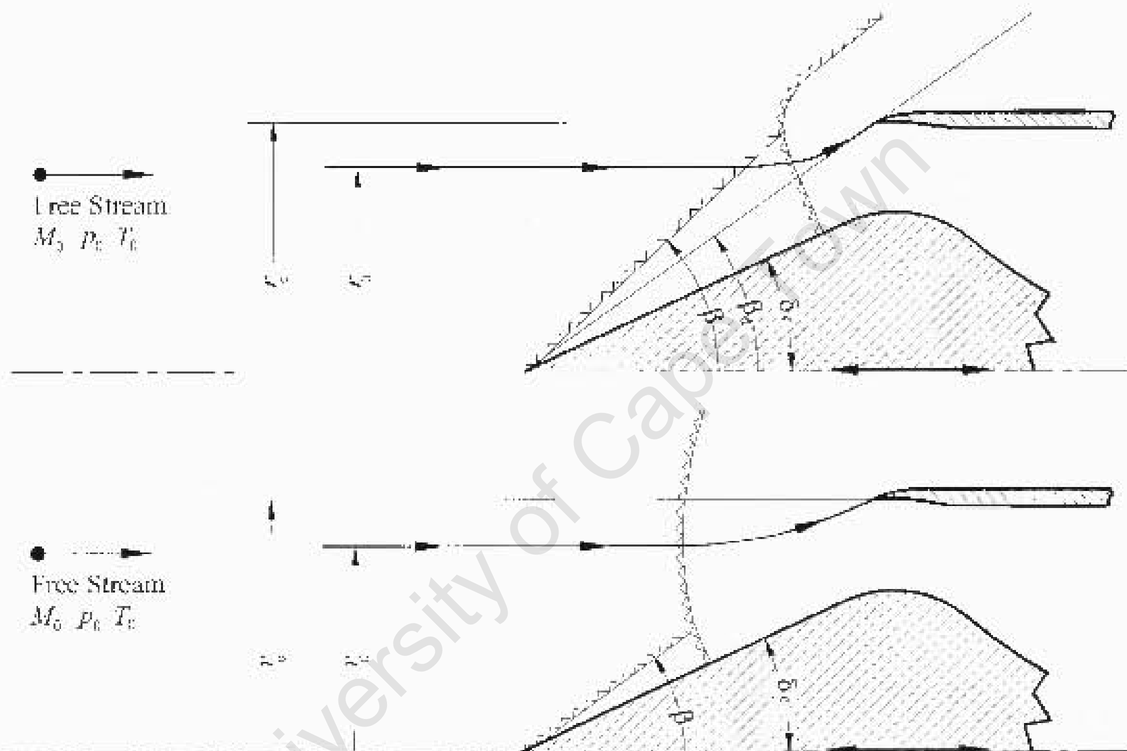


Figure 5.4: Two flow arrangements in sub critical mode

University of Cape Town

5.1.5 Super Critical Mode

In super critical mode, the back pressure acting on the intake is lower than in critical mode operation. The normal shock therefore moves to a location further down the diverging passage inside the cowl, as illustrated in Figure 5.5. The Mach number at this location is higher than at the cowl lip and the normal shock is therefore stronger. The overall pressure recovery is therefore lower than at the critical condition. The mass flow rate through the intake remains constant and equal to the mass flow rate obtained at the critical condition.

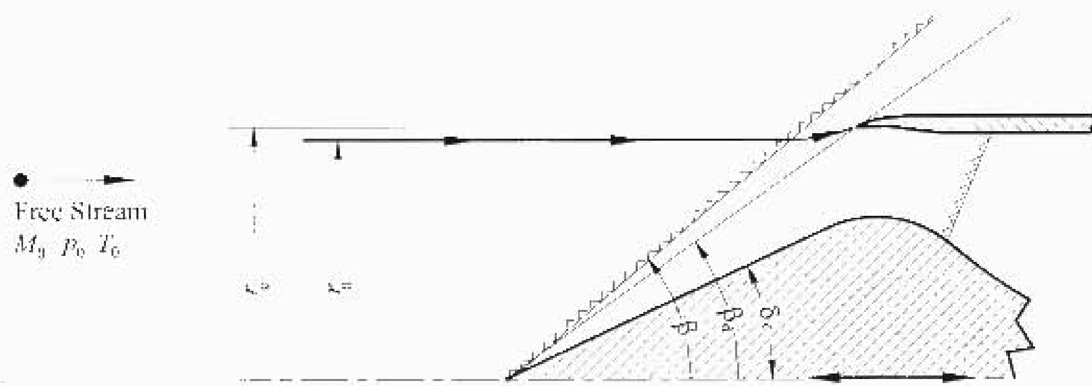


Figure 5.5: Super critical mode shock arrangement

5.1.6 Mass flow and pressure recovery for different operating modes

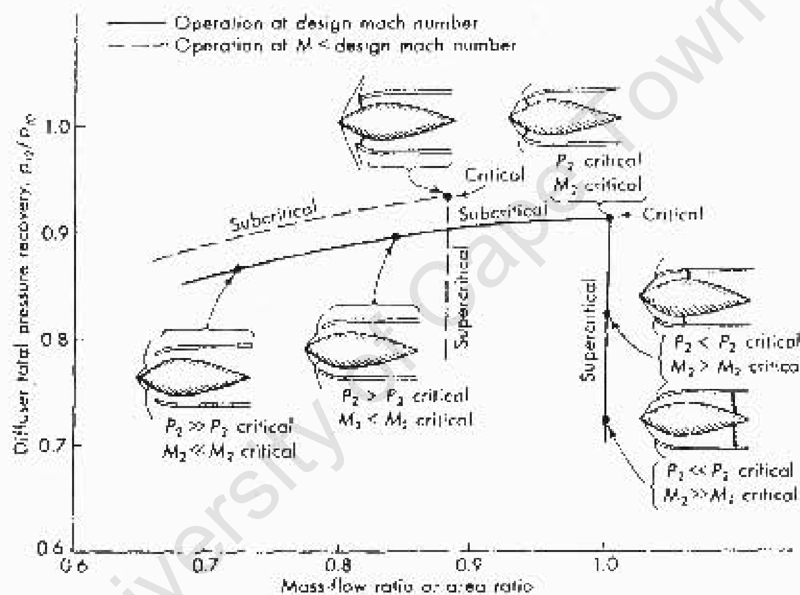


Figure 5.6: Mass flow and Pressure Recovery for different operating modes.

Source: [10]

University of Cape Town

Figure 5.6 illustrates the various operating modes that have been described in this section. The effect of subcritical and supercritical operation on the total pressure recovery is clearly illustrated. The reduced mass flow due to operation at an off-design condition is also indicated.

For this reason, the design of modern intakes is extremely complicated. Generally, facilities are provided for bleeding excess intake mass flow when the engine is choked, so as to avoid a subcritical operational mode. Additional intake gates are provided for engine starting and low Mach number operation such as to allow sufficient mass flow through the intake [10, 5].

Determination of the exact mass flow through the engine is therefore difficult to calculate, and so simplifications and assumptions regarding the intake behaviour have been incorporated into the model developed in this section to estimate the actual mass flow rate.

5.2 Conical Shock Properties

In order to develop a numerical model for a conical intake, some understanding of the behaviour of the conical shock is required. This section will provide a very brief overview of the key properties of this shock type.

When a supersonic stream encounters a wedge, an oblique shock wave forms and the stream is diverted such that the streamlines are parallel to the wedge disturbance. In the case of a conical body in the supersonic stream, the flow around the cone is axisymmetric. From the conservation of mass principle as applied to the fluid between streamlines, the streamlines converge towards the cone surface as the their radius increases. This situation is shown in Figure 5.7.

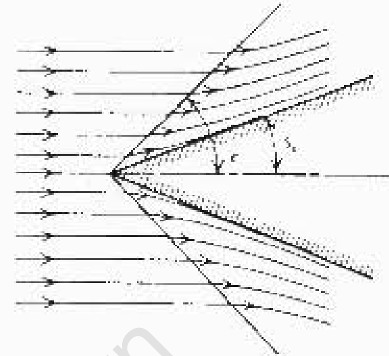


Figure 5.7: Streamlines for supersonic flow over a cone

Source: [12]

Due to the curvature of the streamlines, compression of the gas takes place between the shock wave and the cone surface. Thermodynamic properties are therefore not constant along a streamline, but are instead constant along rays emanating from the cone apex.

The shock angle is dependent on the free stream Mach number and the cone semi-angle. Unlike the simple oblique shock, the shock angle cannot be determined from a simple gas relationship. Instead, it requires an iterative procedure such as that described in Section 16-5 of reference 13, based on the work of Taylor & Macoll. This flow pattern is termed Taylor-Macoll flow.

Once the shock angle for a given cone and free stream Mach number have been determined, the properties behind the shock are readily calculated from the standard ideal gas shock relationships. Furthermore, the properties along any ray can be determined by performing the required numerical integration between the shock and the ray in question.

In order to provide data for the behaviour of a conical shock system, a computer program was written based on the algorithm presented in reference 13. It utilises the Taylor-Macoll approach to solve for the shock angles corresponding to a range of Mach numbers at a given free stream temperature. A comparison of the output of this program to the data published in reference 12 is given in Figure 5.8. This shows that the calculated output matches the published data very well. The gaps in the curves for the 10° and 20° cone semi-angle are due to the algorithm not converging within the specified convergence period for that particular data point. This missing data is readily calculated by increasing the maximum number of iterations the software is allowed to perform.

University of Cape Town

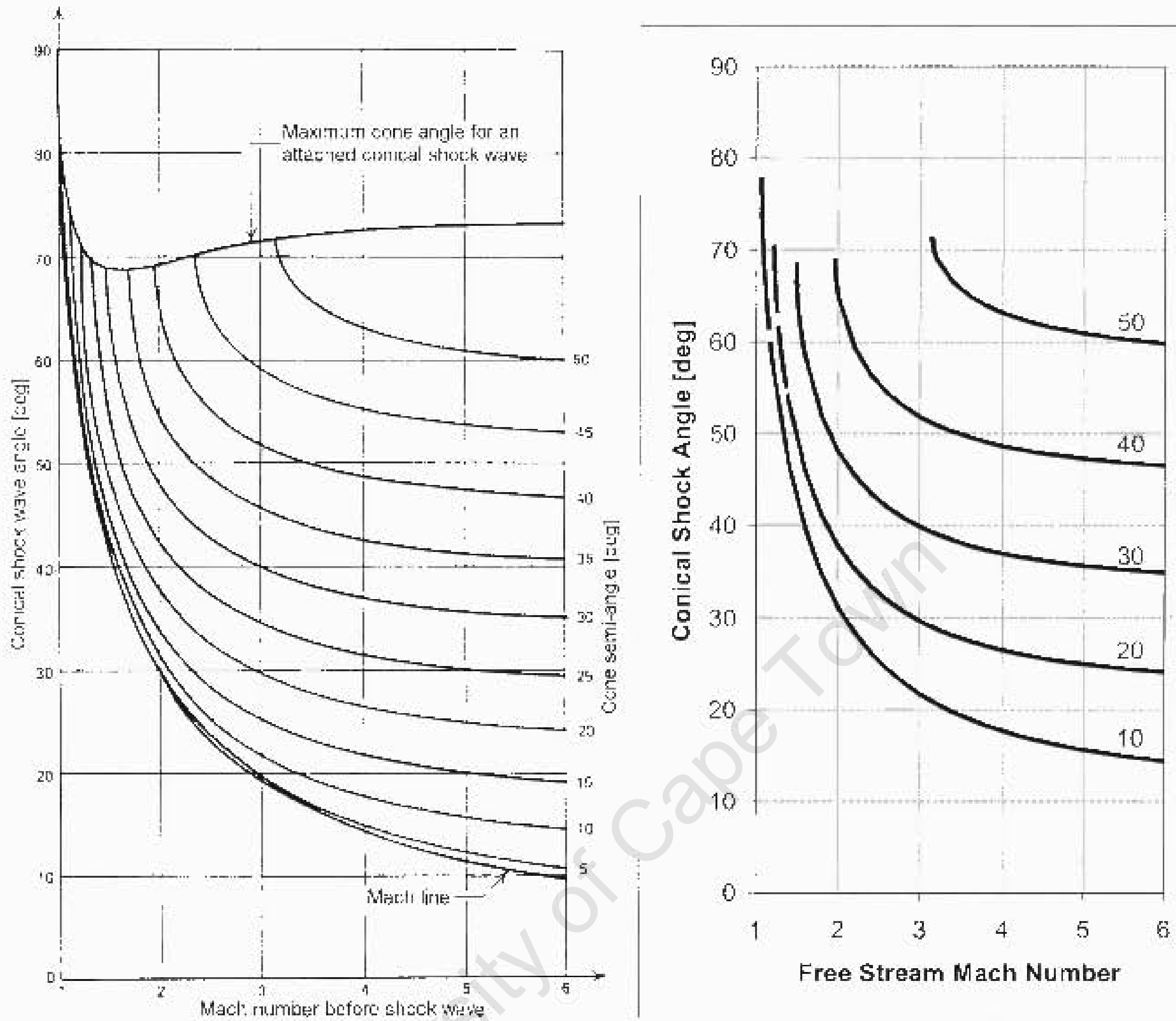


Figure 5.8: Comparison between published conical shock wave data and calculated results

The code for program which calculates these properties is presented in Appendix E-1.

A sample output file from the program for a 30° cone semi-angle is presented in Appendix A-3.

The calculation of properties along any ray was included in the ideal gas utility code which is presented in Appendix E-3.

University of Cape Town

5.3 Standard intake performance curves

Due to the complex nature of intake performance, several standard intake pressure recovery curves have been developed. These allow a quick estimate of how an intake should behave at different Mach numbers. However, they do not predict the mass flow that the intake will be capable of passing. As discussed earlier, the mass flow rate through the intake is affected by operational mode and is critical to determining the net thrust output of the engine. Two standard curves [11] are presented below for comparison with the intake model developed in this chapter.

AIA Pressure Recovery Curve

The Aircraft Industries Association (AIA) standard pressure recovery curve approximates the subsonic recovery as unity, and utilises Eq. 5.1 for the supersonic region.

$$\pi_d = 1 \qquad 0 < M_0 \leq 1$$
$$\pi_d = 1 - 0.1(M_0 - 1)^{1.5} \qquad M_0 > 1 \qquad \textbf{Eq. 5.1}$$

MIL-E-5008B curve

This curve represents a revision of the AIA curve, intended to be somewhat more indicative of actual intake performance. The curve is intended to provide a guide to good, optimum intake performance behaviour. In the subsonic region, the recovery is unity, and Eq. 5.2 applies in the supersonic region.

$$\pi_d = 1 - 0.0075(M_0 - 1)^{1.35} \qquad 1 < M_0 < 5 \qquad \textbf{Eq. 5.2}$$

University of Cape Town

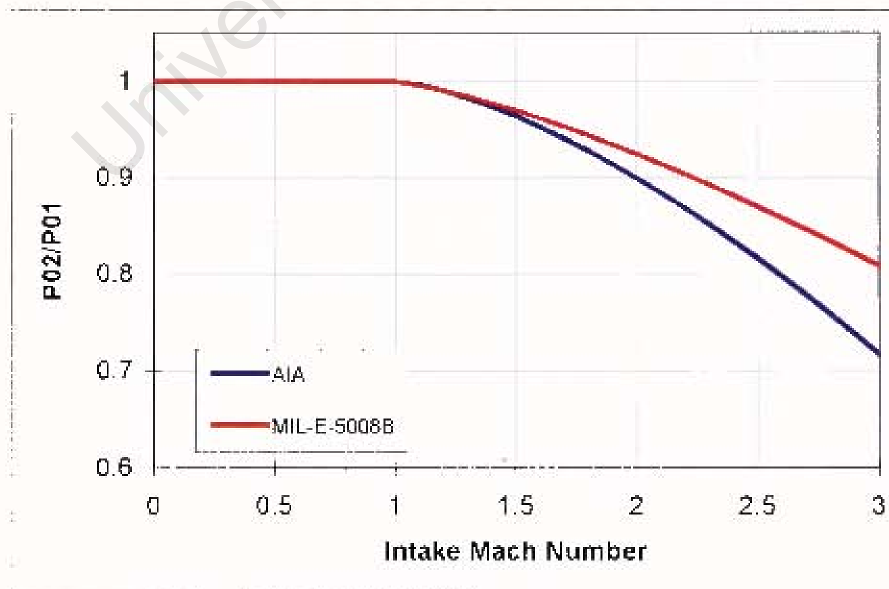


Figure 5.9: Standard pressure recovery curves

5.4 Numerical estimation of intake performance

The equations and algorithms utilised in the numerical intake performance model are described below. Detailed derivations of some of the equations have been placed in appendices for further reference. A comparison between the performance predicted by the algorithms presented in this section and the standard curves is given in Section 7.2.

5.4.1 Detached Shock Solution

Figure 5.10 below shows a schematic diagram of a detached shock system upstream from a conical body. The shock has a hyperbolic geometry [10] which asymptotes towards the Mach angle, μ.

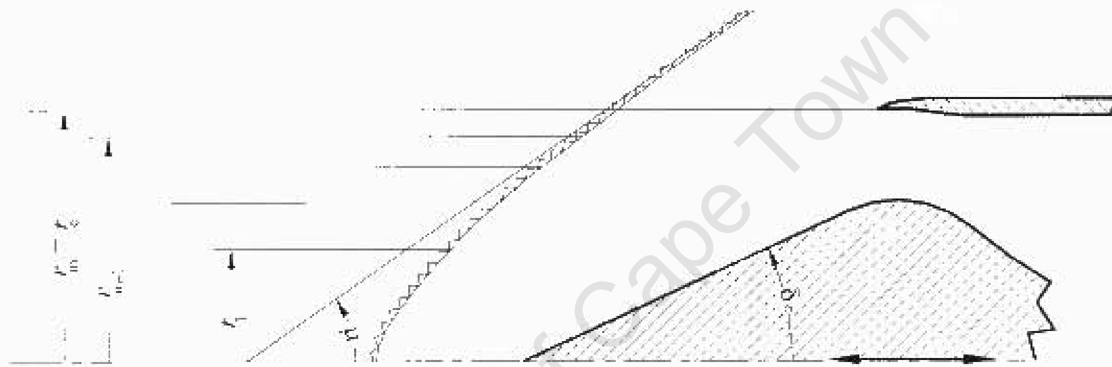


Figure 5.10: Schematic representation of a detached shock and its discretisation

University of Cape Town

At the apex of the hyperbola, the shock properties are similar to those of a normal shock [12]. Towards the outer reaches, the shock behaves as a Mach shock. Between these two positions, the properties behind the shock can be estimated from the oblique shock relations.

The solution algorithm utilises the assumption that the shock angle at the capture radius, r_c, is a linear function of Mach number between a normal shock (90°) and the first attached shock (β at M_{attach}). In reality, the shock asymptotes to the Mach angle at a very large distances from the apex of the cone.

The calculation of the parameters describing the hyperbola is given in Appendix A-2.

Pressure recovery is calculated by dividing the capture streamtube into m annuli of equal area as given by Eq. 5.3. The flow properties before and after the oblique shock are calculated for each section, and the results are averaged to yield the pressure recovery for the capture streamtube.

$$A_n = \pi(r_n^2 - r_{n-1}^2) \qquad \text{Eq. 5.3}$$

where r_n is given by

$$r_n = r_m \sqrt{n/m}$$

and where

A_n = area of annulus n

m = number of annuli

$r_m = r_i$ = streamtube capture radius

r_n, r_{n-1} are the inner and outer radius of annulus n

Thus the pressure recovery is given by Eq. 5.4 below:

$$\frac{p_{t1}}{p_{t0}} = \frac{1}{m}\sum_{n=1}^{m}\left(\frac{p_{t1}}{p_{t0}}\right)_n \qquad \textbf{Eq. 5.4}$$

Mass flow rate in this condition is assumed to be governed by the engine and exhaust.

5.4.2 Critical Solution

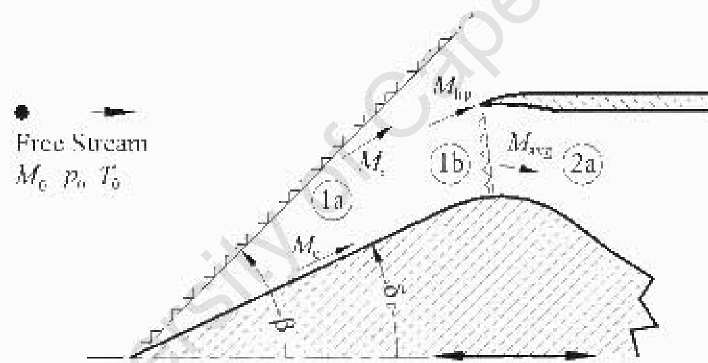


Figure 5.11: Pressure recovery in critical mode operation

University of Cape Town

Figure 5.11 shows the pressures at various locations in the intake shock system during critical mode operation. The solution for the stagnation pressure p_{t2a} is as follows:

- The conical shock wave angle β is determined from previously calculated Taylor-Macoll conical flow properties, as described in Section 5.2.
- The algorithm determines whether or not the conical centrebody can be positioned such that β_d matches β. If this is not possible, β_d is adjusted to a value that is as close to β as possible.
- The average Mach number of the airstream entering the cowl is determined as the average of the Mach number at the cowl lip, M_{lip} and the cone surface, M_c. If β_d matches β, then M_{lip} is given by the Mach number immediately after the conical shock. However, if β_d is less than β, M_{lip} must be determined by integrating the Taylor-Macoll flow equations between β and β_d.

The pressure recovery across the conical shock is calculated from Eq. 5.5:

$$\frac{p_{t1}}{p_0} = \frac{p_{t0}}{p_0} \cdot \frac{p_{t1}}{p_{t0}} \qquad \textbf{Eq. 5.5}$$

$$M_{avg} = \frac{M_{lip} + M_c}{2} \qquad \textbf{Eq. 5.6}$$

- The average Mach number of the stream entering the cowl is then calculated Eq. 5.6. If $M_{avg} < 1$, the stream entering the intake is subsonic, and there will be no normal shock. It is possible that $M_{lip} > 1$ while $M_{avg} < 1$, however, in this case, M_{lip} will be close to unity and the pressure ratio across the shock would be approximately unity as well.
- If $M_{avg} > 1$, a normal shock occurs and the pressure ratio across the shock is caluclated from the normal shock equations.

The total pressure recovery for the intake is therefore given by Eq. 5.7:

$$\pi_d = \frac{p_{t2}}{p_{t0}} = \frac{p_{t2}}{p_{t1}} \cdot \frac{p_{t1}}{p_0} \cdot \frac{p_0}{p_{t0}} \qquad \textbf{Eq. 5.7}$$

The maximum mass flow rate that the intake is capable of passing is calculated by considering the position of the shock system, which determines the maximum capture streamtube diameter.

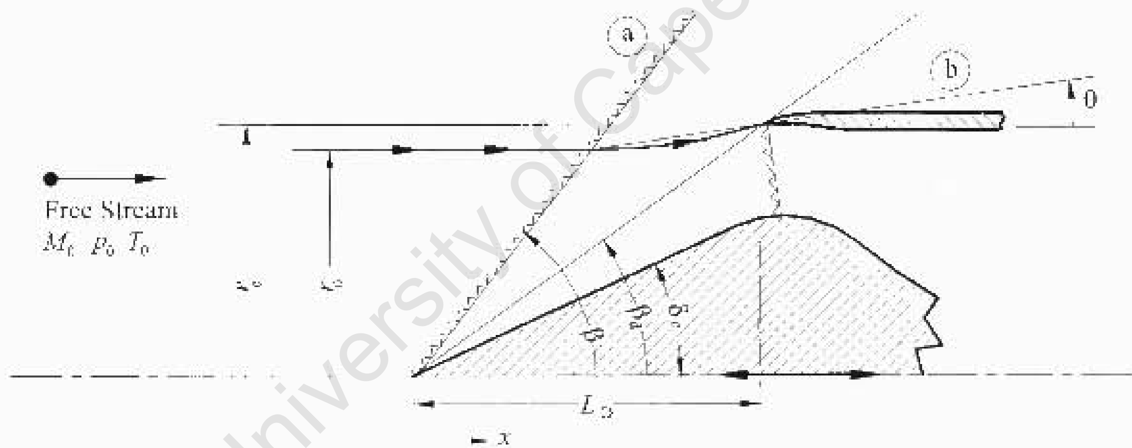


Figure 5.12: Parameters for determining the mass flow through the intake

Figure 5.12 illustrates the shock arrangement and flow streamline of the actual capture streamtube, r_0. The conical shock (a) is described by Eq. 5.8 while the average streamline (b) is described by Eq. 5.9.

$$r_a(x) = x \tan\beta \qquad \textbf{Eq. 5.8}$$

$$r_b(x) = r_c + (x - L_D)\tan\theta \qquad \textbf{Eq. 5.9}$$

Solving for the intersection of these lines yields Eq. 5.10 below:

$$r_0 = \frac{r_c - L_D \tan\theta}{1 - \dfrac{\tan\theta}{\tan\beta}} \qquad \textbf{Eq. 5.10}$$

University of Cape Town

The only unknown on the right hand side in Eq. 5.10 is θ, the flow angle. θ represents the average flow direction of a streamline after the conical shock. This streamline curves, starting at the flow direction given by the oblique shock equations, and approaching the slope of the cone as it nears the cone surface. The *average* flow angle is given by the average of the flow angle immediately after the shock and the flow angle at the intake lip. The flow angle at the lip can be calculated by integrating the Taylor-Macoll flow equations.

The maximum flow possible through the intake is then given by Eq. 5.11.

$$\begin{aligned} \dot{m}_a &= \rho_0 A_0 v_0 \\ &= \rho_0 \pi r_0^2 M_0 \sqrt{\gamma R T_0} \end{aligned} \qquad \textbf{Eq. 5.11}$$

5.4.3 Sub Critical Solution

When the maximum flow possible through the engine is less than the maximum flow through the intake operating in critical mode, as given by Eq. 5.11, the engine appears as a high back pressure to the intake, and it will operate in sub critical mode as discussed in Section 5.1.4. The engine effectively acts as a choke, and the mass flow rate through the intake is limited to the maximum flow through the engine. The maximum flow in turn determines the capture streamtube diameter.

The shock arrangement is illustrated in Figure 5.4. The pressure recovery is therefore dependent on the location of the normal shock as this determines what proportion of the capture streamtube passes through two shocks, and what passes through only the normal shock.

The simulation procedure utilised assumes that for zero mass flow through the engine, the normal shock is located at the apex of the conical centrebody. The position of the normal shock for a given flow ratio is then interpolated between the normal shock placed at the cowl lip (flow ratio = 1) and the cone apex (flow ratio = 0).

5.4.4 Super Critical Solution

In super critical mode, the mass flow is limited to the maximum mass flow obtained in critical mode, as discussed in Section 5.1.5 above. The pressure recovery in this case depends on the Mach number at which the normal shock occurs, which in turn is dependent upon the throat area and the pressure at the compressor face.

University of Cape Town

5.5 Software Implementation of the Intake Model

The methods, procedures and equations of this chapter were coded into a C++ class called "CConeIntake". This class has data members which represent the geometry and fluid properties which affect the performance of the intake. It also has member functions which provide pressure recovery calculations in the various operating modes described in the preceding sections.

A flow chart representing the procedure utilised for determining the pressure recovery of the intake is presented in Appendix A-1.

Conical shock properties are provided in a lookup-table which can be changed at run-time to allow the intake to be used at different altitudes within the same simulation. The conical shock properties are generated by a standalone program "TMFlowProps" which implements the Taylor-Macoll solution [13] for conical shock properties. This was done due to the iterative nature of the conical shock solution algorithm. Conical shock properties along rays at angles other than the shock angle are then easily calculated by Runge-Kutte integration from the previously calculated shock angle and properties. The functions for determining these properties are included in the IdealGas class which encapsulates the behaviour of an ideal gas.

The code for the CConeIntake class is provided in Appendix A-4.

The code for the IdealGas class is provided in Appendix E-3.

University of Cape Town

Chapter 6: Aerodynamic Model

This chapter describes the manner in which the aerodynamic model is implemented in the simulation. It covers some basic aerodynamic concepts and then proceeds to discuss the development of the Digital DATCOM model used, and how the output from this model was processed for use in the overall simulation.

6.1 Overview of Basic Aerodynamics

Figure 6.1 illustrates the basic forces acting on a wing section [14]. The nomenclature is as follows:

V_∞ = free stream velocity

α = angle of attack

c = cord length

R = resultant aerodynamic force

N = component of R perpendicular to c

A = component of R parallel to c

L = *Lift* - the component of R perpendicular to V_∞

D = *Drag* - component of R parallel to V_∞

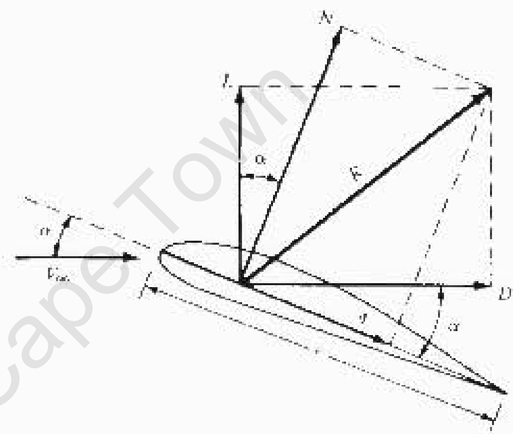


Figure 6.1: Basic aerodynamic forces acting on a wing.
Source: [14]

By defining a dynamic pressure, q_∞, as given by Eq. 6.1, and dividing the relevant force (e.g. L) by q_∞ and S, the reference area, a dimensionless coefficient is obtained. Dimensionless coefficients of this form are a more fundamental indicator of the aerodynamic properties of a wing than the actual forces themselves and allow comparison between different sizes of wings and their behaviour at different altitudes.

$$q_\infty = \tfrac{1}{2}\rho_\infty V_\infty^2 \qquad \text{Eq. 6.1}$$

University of Cape Town

6.1.1 Subsonic flight and wing sections

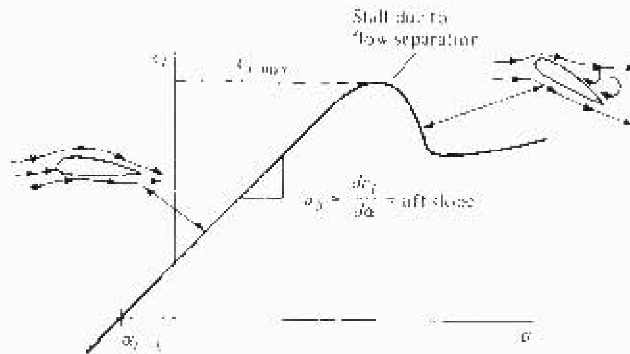


Figure 6.2: Variation of lift coefficient with angle of attack
Source: [14]

In the subsonic flight regime, a typical subsonic airfoil behaves as shown in Figure 6.2. The variation of C_L with α is largely linear up to the point where flow separation begins. This is the point at which the maximum lift C_{Lmax} is achieved. Beyond this point, the lift coefficient decreases as the wing stalls.

By defining a lift slope as given by Eq. 6.2, the lift coefficient can be easily calculated up to C_{Lmax}. This approach, however, does not allow the value of C_{Lmax} to be determined.

$$C_{L_\alpha} = \frac{dC_L}{d\alpha} \qquad \textbf{Eq. 6.2}$$

The variation of drag with angle of attack is schematically shown in Figure 6.3. For a general wing, this can be estimated by Eq. 6.3 [14]:

$$C_D = C_{D0} + \frac{C_L^2}{\pi \cdot AR \cdot e} \qquad \textbf{Eq. 6.3}$$

where,

AR	–	aspect ratio of the wing (b^2/S)
e	–	span efficiency factor

University of Cape Town

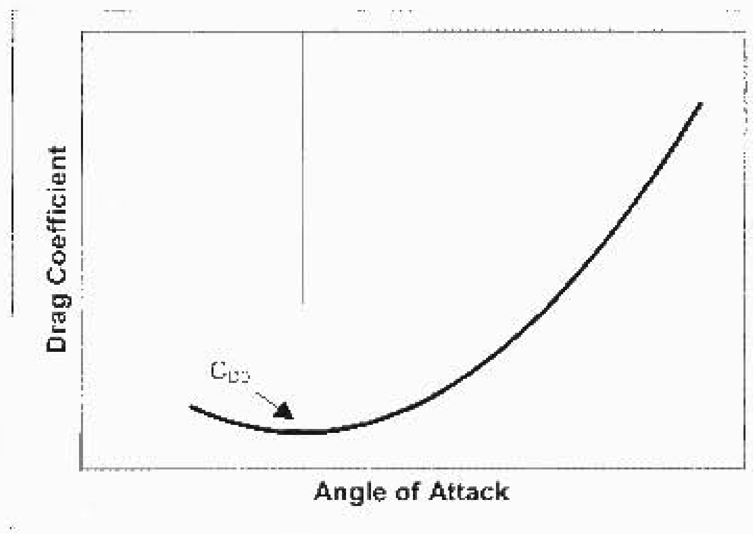


Figure 6.3 : Idealised variation of drag coefficient with angle of attack.

The lift-to-drag ratio is an indicator of how efficiently a wing generates lift. Generally, the higher the lift to drag ratio of an aircraft, the better its performance. [14] Using Eq. 6.3, an idealised curve of lift-to-drag vs. lift can be plotted, as shown in Figure 6.4.

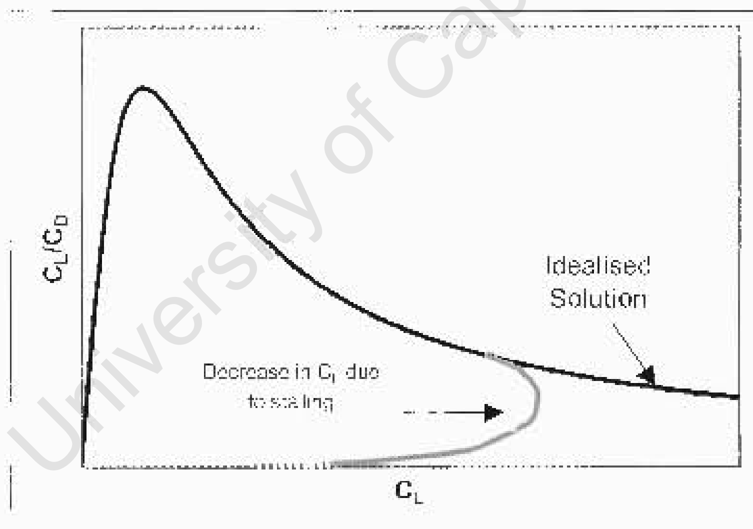


Figure 6.4: Variation of Lift-to-Drag ratio with C_L

Although Eq. 6.3 predicts a gradually decreasing C_L/C_D with an infinitely increasing value of C_L, this does not occur in reality. As the angle of attack increases, a condition where the flow separates from the wing occurs and the wing stalls, as indicated in Figure 6.2. The value of C_L therefore decreases. This results in the curve shown in Figure 6.4 curling back towards zero. For delta wing aircraft, this stall condition occurs at high angles of attack.

University of Cape Town

6.1.2 Variation of Drag with Mach number

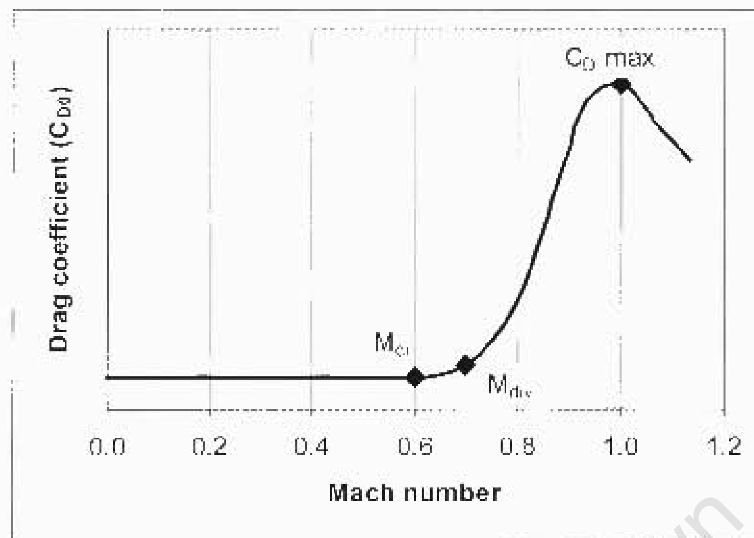


Figure 6.5: Variation of C_{D0} with Mach number

Figure 6.5 shows a schematic of the variation of drag coefficient C_{D0} with Mach number. It shows that C_{D0} is nominally constant until some critical Mach number M_{cr} after which it starts to increase, reaching a maximum value at sonic speed. Thereafter the value decreases. One of the factors influencing the ratio of C_{Dmax} to C_{Dinit} is the so called 'area rule' [14]. The application of the area rule results in the waisted appearance of many combat aircraft.

University of Cape Town

6.2 Digital DATCOM model

The Digital DATCOM [15, 16] was used to provide estimates of the aerodynamic coefficients of a hypothetical delta-winged aircraft loosely based on the Dassault Mirage III. The following sections describe how the aircraft was idealised for modelling with the DATCOM. Some of the limitations of the model are discussed, along with some results.

6.2.1 Describing the airframe

The airframe model was generally based on the Dassault Mirage III, which was flown in the 1960's to 1980's by many of the world's airforces. The approximate model is shown in Figure 6.6, with the dimensions used. A more detailed set of drawings with dimensions is available in Appendix B-1. The modelling of various aspects of the airframe such as the fuselage, wings and tail section are described in more detail in the sections to follow.

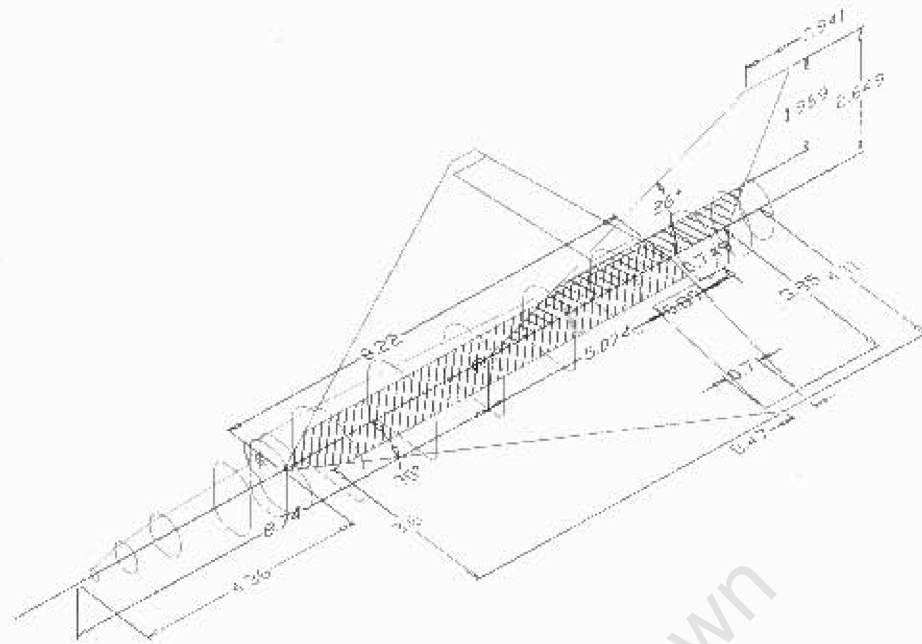


Figure 6.6: Schematic aircraft model for use in the Digital DATCOM

University of Cape Town

Body

The body was modelled by providing the Digital DATCOM with four parameter arrays. These are x-ordinate, cross-section half-width, cross-sectional area and perimeter as indicated in Figure 6.6. Data for these was obtained from scaled drawings from Jane's [3] and a 1/48th-scale model of a Mirage IIIC.

Centre of mass was placed such that it was within the aircraft's wheelbase, and on the aircraft centreline, even though it probably lies somewhat lower when taking into account the position of the wings, and any armaments.

Wing

Planform Geometry

Planform geometry is that of a swept-back tapered wing, resulting in a delta shape. Dimensions were specified such that the planform would be similar to the Mirage III, dimensions of which were obtained from drawings from Jane's [3].

Dimensions are given in Table 6.1:

Table 6.1: DATCOM parameters for the wing planform

Description	Notes	Eng. Symbol	Datcom Namelist	Datcom Variable	Value	Units
Longitudinal location of wing apex	1		SYNTHS	XW	4.36	m
Vertical location of wing apex			SYNTHS	ZW	0	m
Wing incidence	2	i	WGPLNF	ALIW	0	Deg
Tip chord		c_t	WGPLNF	CHRDTP	0.47	m
Root chord		c_r	WGPLNF	CHRDR	8.22	m
Semi-span	2	b	WGPLNF	SSPN	4.11	m
Exposed semi-span			WGPLNF	SSPNE	3.6	m
Sweepback angle		Λ	WGPLNF	SAVSI	60	Deg
Reference location for sweepback angle			WGPLNF	CHSTAT	0	% chord
Dihedral	3		WGPLNF	DHDADI	0	Deg
Twist			WGPLNF	TWISTA	0	Deg
Planform type			WGPLNF	TYPE	1	

Notes:

1. *Measured from aircraft nose to theoretical vertex of delta.*
2. *From Jane's [3].*
3. *Mirage III has 1° anhedral, but this value is only used for lateral coefficient estimation and could therefore be ignored.*

Airfoil Section

Jane's [3] states that the airfoil section is of *conical camber*, and varies in thickness from 4.5% to 3.5% chord, root to tip. The airfoil used in the Digital DATCOM model has a 4% maximum thickness at 50% chord, with a maximum camber of 1%.

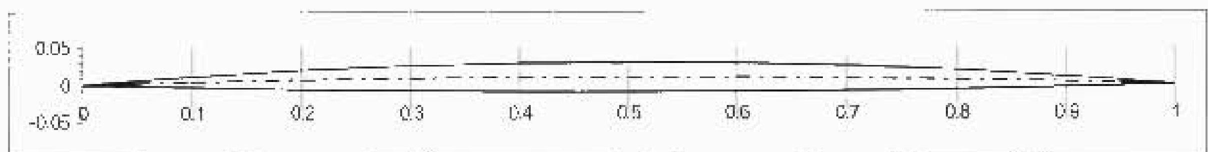


Figure 6.7 : Airfoil section used for wing specification

University of Cape Town

The airfoil was entered by specifying the chord position and corresponding upper and lower surface co-ordinates in the DATCOM.

The ksharp wave drag factor was taken from the DATCOM User's manual [15] as 16/3.

Elevators

Elevators were modelled as simple symmetric flaps. Parameters for the elevators are as follows:

Table 6.2: DATCOM parameters for modelling the Elevators

Description	Eng. Symbol	Datcom Namelist	Datcom Variable	Value	Units
Flap type – simple		SYMFLP	FTYPE	1	
Trailing edge tangent (90 & 99% chord)	$Tan(\varphi_{TE}^{\,\prime})$	SYMFLP	PHETE	0.0522	
Trailing edge tangent (95 & 99% chord)	$Tan(\varphi_{TE}^{\,\prime\prime})$	SYMFLP	PHETEP	0.0523	
Inboard span location of flap		SYMFLP	SPANFI	0.749	m
Outboard span location of flap		SYMFLP	SPANFO	3.85	m
Flap chord at inboard edge		SYMFLP	CHRDFI	0.89	m
Flap chord at outboard edge		SYMFLP	CHRDFO	0.7	m
Average chord of the balance		SYMFLP	CB	0.05	m
Average thickness of the flap		SYMFLP	TC	0.1	m
Flap nose type – round nose		SYMFLP	NTYPE	1.0	

Tail

A vertical tail was added to the model to improve the drag estimate. The airfoil section used was a standard symmetric supersonic airfoil of 4% maximum thickness available in DATCOM. The parameters for modelling the tail are given in Table 6.3.

University of Cape Town

Table 6.3: DATCOM parameters for modelling the Tail

Description	Eng. Symbol	Datcom Namelist	Datcom Variable	Value	Units
Longitudinal location of tail apex on centreline		SYNTHS	XV	8.74	m
Vertical tail above reference line?		SYNTHS	VERTUP	True	
Tip chord	c_t	VTPLNF	CHRDTP	0.941	m
Root chord	c_r	VTPLNF	CHRDR	5.074	m
Semi-span	b	VTPLNF	SSPN	2.649	m
Exposed semi-span		VTPLNF	SSPNE	1.969	m
Sweepback angle	Λ	VTPLNF	SAVSI	64	Deg
Reference location for sweepback angle		VTPLNF	CHSTAT	0	% chord
Planform type		VTPLNF	TYPE	1	
Wave drag factor		VTSCHR	KSHARP	16/3	

6.2.2 Flight conditions

The Digital DATCOM was instructed to produce C_L and C_D data for Mach numbers in the range from 0 to 2.2 and in the altitude range from sea level to 20km.

6.2.3 Limitations of the model

The Digital DATCOM only provides trimmed data for the subsonic flight regime. The assumption was therefore made that the aircraft in question was neutrally stable at all flight conditions. Since the model is required to produce only limited hypothetical aerodynamic data that does not vary for the various simulations, this is a reasonable assumption.

In the transonic region ($0.7 < M_0 < 1.4$), the DATCOM does not provide an estimate of stall behaviour. Instead it utilises a constant value for the lift slope and does not limit the maximum value that C_L achieves. In addition, C_D is not calculated beyond a certain angle of attack. C_{Lmax} and the missing values of C_D were therefore estimated, as described later.

In general, the output data from the DATCOM was not always smooth, and this is evident in some of the results presented in Chapter 7.

University of Cape Town

6.3 Results of the DATCOM model

The following section presents selected results from the Digital DATCOM model. Due to the large volume of data generated, it is not possible to present all the results in this report. However, the selection presented below particularly covers the data most relevant for the simulations described in this document.

6.3.1 C_L vs. AoA

Figure 6.8 shows the untrimmed lift coefficient data for various angles of attack and various Mach numbers. Of particular interest are the straight lines obtained for flight in the transonic range. Due to the complexities of transonic aerodynamics, the Digital DATCOM only provides lift data based on a constant value for $C_{L\alpha}$ in the transonic range as discussed earlier.

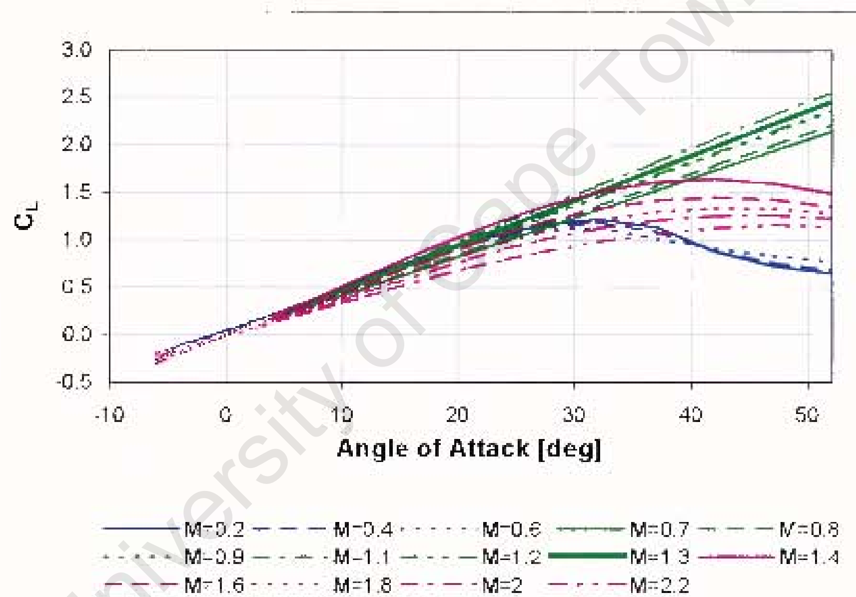


Figure 6.8: Lift coefficient (C_L) for various angles of attack, at various Mach numbers.

University of Cape Town

6.3.2 C_D vs. AoA

Figure 6.9 shows the Digital DATCOM results for untrimmed drag coefficient for various Mach numbers at a varying angle of attack. The lack of data at high angles of attack mentioned earlier can be observed. The noisy curve for $M = 1.1$ is also illustrated.

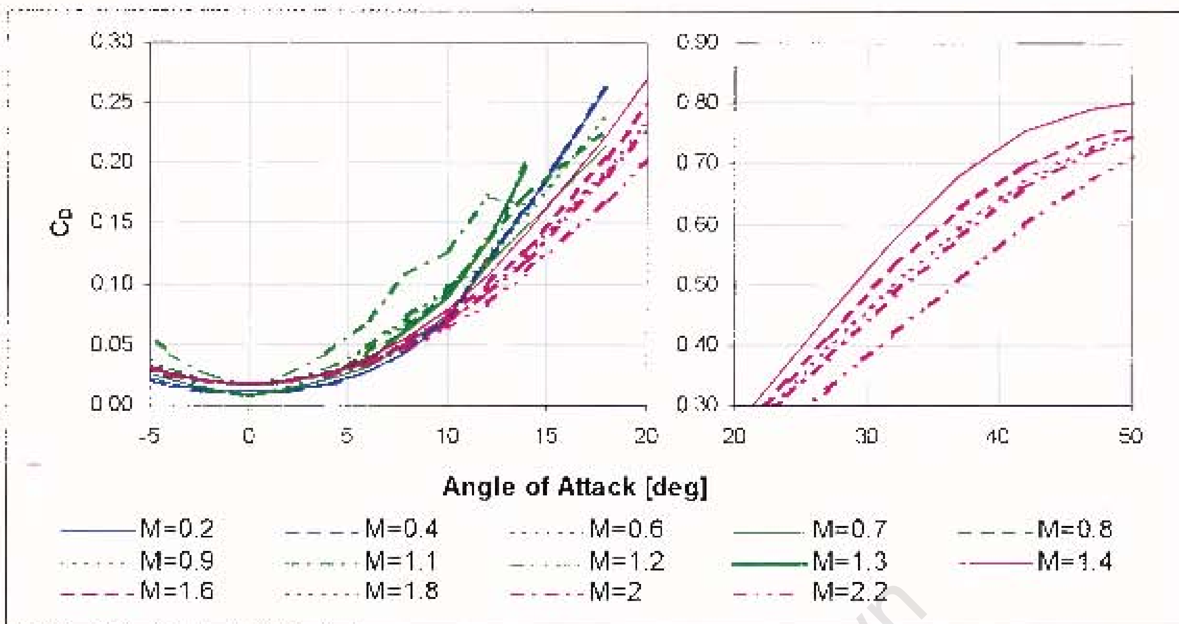


Figure 6.9: Drag coefficient (untrimmed) for various angles of attack and Mach numbers

6.3.3 $C_{L\ max}$ vs. M

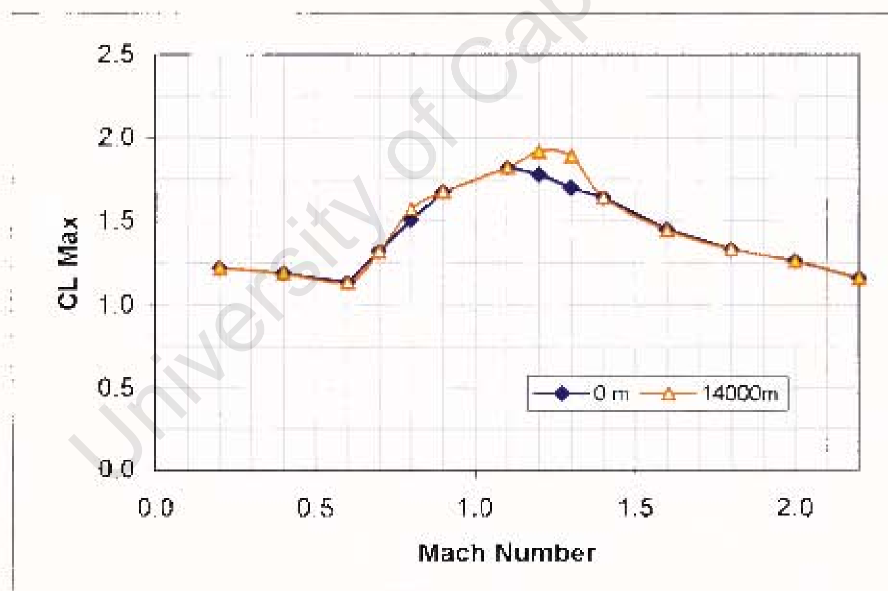


Figure 6.10: Maximum lift coefficient vs. Mach number.

Figure 6.10 shows the variation of C_{Lmax} with Mach number. The estimated values shown in the transonic range are based on interpolating the angle of attack between $M=0.6$ and $M=1.4$. C_{Lmax} was then calculated based on this angle of attack.

University of Cape Town

6.3.4 C_D vs. Mach Number

Figure 6.11 and Figure 6.12 show the variation of drag with Mach number at an altitude of 4000m. The increase in drag as the Mach number approaches unity is quite evident. Figure 6.11, the value of C_D at C_{Lmax} also shows a good correlation to Figure 6.5. In the transonic range this is in part due to the fact that C_D was estimated based on the value of C_L, as given by Eq. 6.3.

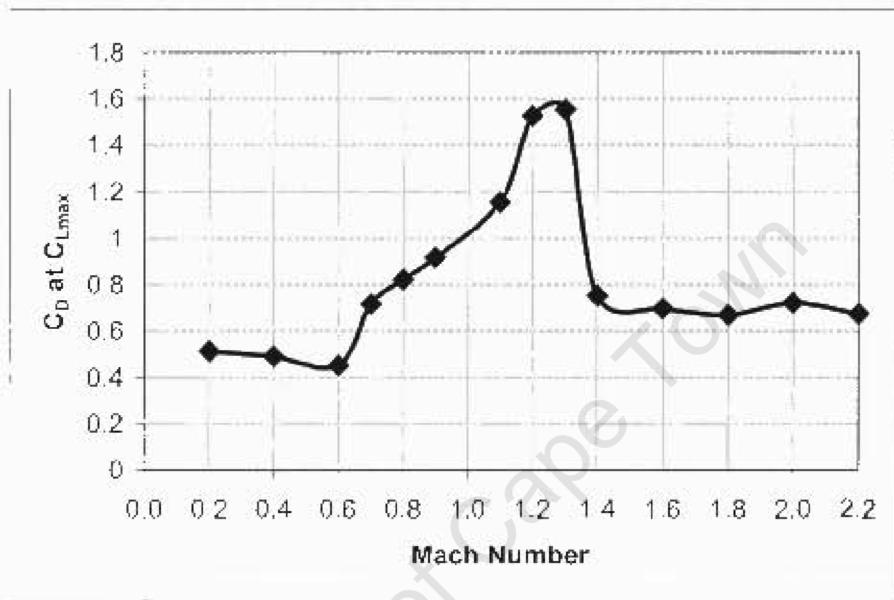


Figure 6.11: C_D for maximum C_L at 4000m

Figure 6.12 does not show as good a correlation with Figure 6.5 as Figure 6.11. However, it must be borne in mind that Figure 6.5 is only an indication of an idealised behaviour pattern. In the transonic region the aerodynamic behaviour is complex and reliable estimates are difficult to obtain from empirically or theoretically based formulations. The values indicated in Figure 6.12 are therefore unlikely to be accurate, but the trends indicated are not necessarily incorrect.

Furthermore, for the purposes of the numerical investigation under discussion, these aerodynamic effects are common to all the simulations and are therefore of secondary importance to the results and can therefore be tolerated.

University of Cape Town

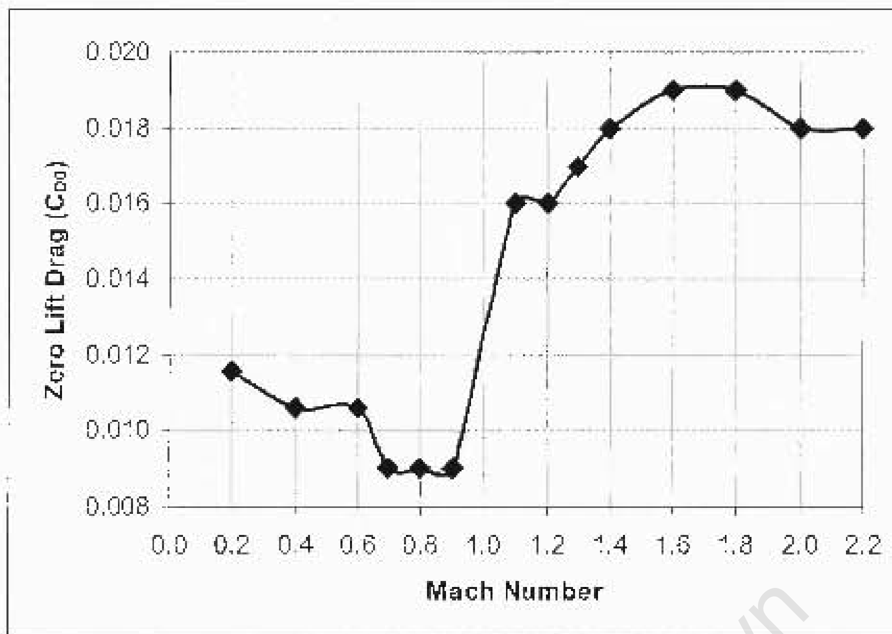


Figure 6.12: Variation of C_{D0} with Mach Number at 4000m

University of Cape Town

6.3.5 C_L/C_D vs. C_l

Figure 6.13 shows the Digital DATCOM prediction for variation of Lift to Drag ratio with increasing Lift. It can be seen that the trends indicated resemble those suggested in Figure 6.4. All the DATCOM data utilised in the simulations was limited to values of C_L up to and including C_{Lmax}. The start of the curling of the Lift to Drag ratio curve can be seen at high values of C_L, but is not completed due to the data having been removed.

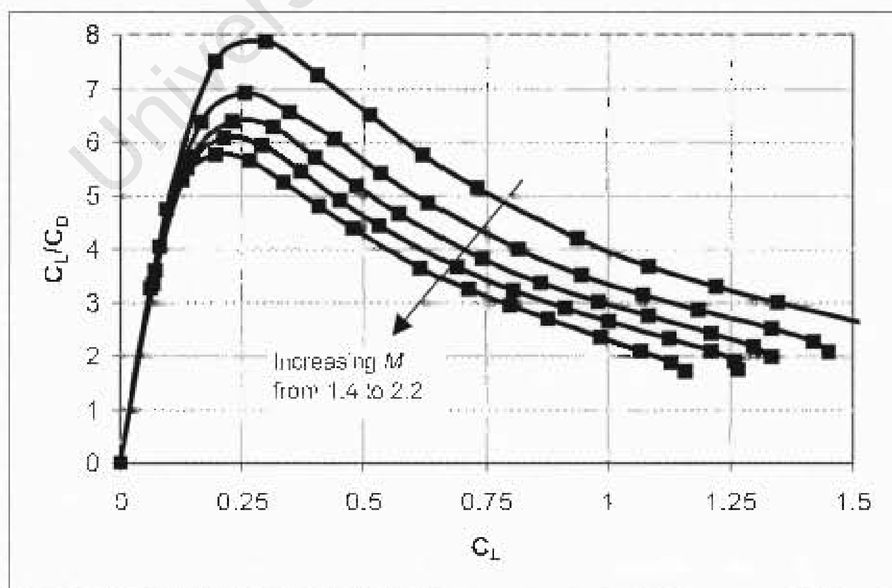


Figure 6.13: Variation of Lift to Drag ratio with Lift in the supersonic regime at sea level

6.4 Software Implementation of the Aerodynamic Model

As discussed in this chapter, most of the aerodynamic coefficient calculations were performed with the aid of the Digital DATCOM. The DATCOM, however, produces output which is human-readable. Software was therefore written to extract the lift, drag and angle of attack values for each flight condition and automatically store these in a format suitable for accessing from a lookup-table.

Each lookup table contained one flight condition, i.e. one Mach number and altitude. Separate lookup tables were generated to allow data to be accessed via C_L, C_D and angle of attack.

Furthermore, the Digital DATCOM does not provide values for C_D at high angles of incidence. These were therefore estimated using Eq. 6.3. Data beyond the stall point was not required, so the data was cropped during processing to the maximum value of C_L. In the transonic regime, the DATACOM does not provide maximum values for C_L as it utilises a constant value for the lift slope as discussed in Section 6.3.1. These were then estimated by determining a stall angle of attack by linearly interpolating between the angle of attack for C_{Lmax} at M=0.6 and at M=1.4. The maximum value for C_L was then determined from the stall angle.

A multidimensional lookup table system was developed to allow interpolation across tables, thus allowing aerodynamic coefficients to be determined at any Mach number, altitude and angle of attack.

The code for the single- and multi-file lookup tables is presented in Appendix E-2.

The code for extracting and processing the Digital DATCOM data is presented in Appendix A-1.

University of Cape Town

Chapter 7: Simulation Results

Two engine models were constructed based on production engines manufactured by French company SNECMA [17, 18, 19]. These were compared from an engine-only performance perspective, as detailed in Section 7.1, and combined with the aerodynamic data obtained in Section 6.3 to yield the integrated aerodynamic - intake - propulsion-unit performance comparisons presented in the remainder of this chapter.

7.1 Comparison of Engine Performance

This section presents the results obtained from engine-only simulations such as the thermodynamic cycle, thrust specific fuel consumption and net thrust output. Results are based on the SNECMA Atar 9k50 and the SNECMA M53. The Atar series was widely used in the Mirage III, while the M53 has been used in the Mirage 2000. The source code for this section can be found in Appendix C-3.

University of Cape Town

7.1.1 Propulsion Model Input Parameters

Table 7.1 lists the input parameters utilised for the propulsion model. Notes are listed at the end of the table, indicating sources for the values and assumptions utilised.

Table 7.1: Propulsion model input parameters.

Paramter Name	Variable Name	Unit	9k50	M53
Afterburner	AB-ON-OFF		0 – OFF; 1 = ON	
Fan pressure ratio	pi-f		1	3
Compressor pressure ratio	pi-c		6.15	9.8
Burner pressure ratio	pi-b		0.97	0.97
A/B pressure ratio	pi-a		0.98	0.98
Bypass ratio	Beta		0	0.36
Fan efficiency	eta-f		1	0.95
Compressor efficiency	eta-c		0.85	0.88
Burner efficiency	eta-b		0.96	0.97
Turbine efficiency	eta-t		0.9	0.92
Afterburner efficiency	eta-a		0.96	0.96
Primary nozzle efficiency	eta-n		0.97	0.97
Secondary nozzle efficiency	eta-nb		0.97	0.97

Paramter Name	Variable Name	Unit	9k50	M53	
Max turbine inlet temperature	MaxTt4	K	1203	1600	
Max afterburner temperature	MaxtTt6	K	2500	2500	
Generator mass flow max	MaxMassFlow		72	94	1
Compressor suction flow	PumpMassFlow		71	85	
Fuel energy	JetFuelQR	J/kg	4396.400		2
Primary nozzle type (converging = 1)	PrimaryNozzleType		1	1	5
Bypass nozzle type	BypassNozzleType		1	1	5
Maximum primary nozzle area	A7-max	m^2	0.32	0.24	6
Minimum primary nozzle area	A7-min	m^2	0.15	0.1	
Maximum bypass nozzle area	A9-max	m^2	0	0.04	
Minimum bypass nozzle area	A9-min	m^2	0	0.04	
Aerodynamic reference area	SREF	m^2	35.716	35.716	3
Takeoff mass	MASS	kg	13700	13700	4
Maximum intake Mach number	Intake_Mmax		2.3	2.3	
Mach number for specified flow rate	Intake_Mmdot		0.5	0.6	
Mass flow rate specification	Intake_mdot	kg/s	72	94	
Density spec for flow rate	Intake_rho	kg/m^3	1.01	1.01	
Subsonic pressure recovery	Intake_SubPiD		0.99	0.99	
Intake cone angle (half angle)	IntakeConeAngle	°	25	25	

University of Cape Town

Notes:

1. *Source: [17, 18].*
2. *Source: [9].*
3. *From the reference area calculated by the Digital DATCOM.*
4. *This is the maximum rated takeoff mass for the Mirage III [3].*
5. *Source: Jane's [3]*
6. *Estimated to produce correct net thrust output*
7. *Other parameters are estimated based on typical values in the literature.*

7.1.2 Thermodynamic Cycle: T-s Diagrams

Figure 7.1 through to Figure 7.4 show the thermodynamic cycles for the 9k50 and M53 on Temperature - Entropy (T-s) diagrams at a free stream Mach number of 0.66 and at an altitude of 0m.

Figure 7.1 shows the T-s diagram for the 9k50 with the afterburner off. This diagram shows that the compressor exit pressure, p_{t3}, is 791 kPa and that the turbine inlet temperature is 1200 K. It also shows that p_7, the exit pressure is not the same as the free stream pressure as the nozzle is a converging flap type, and the exit stream is therefore not fully expanded. i.e. the nozzle is choked.

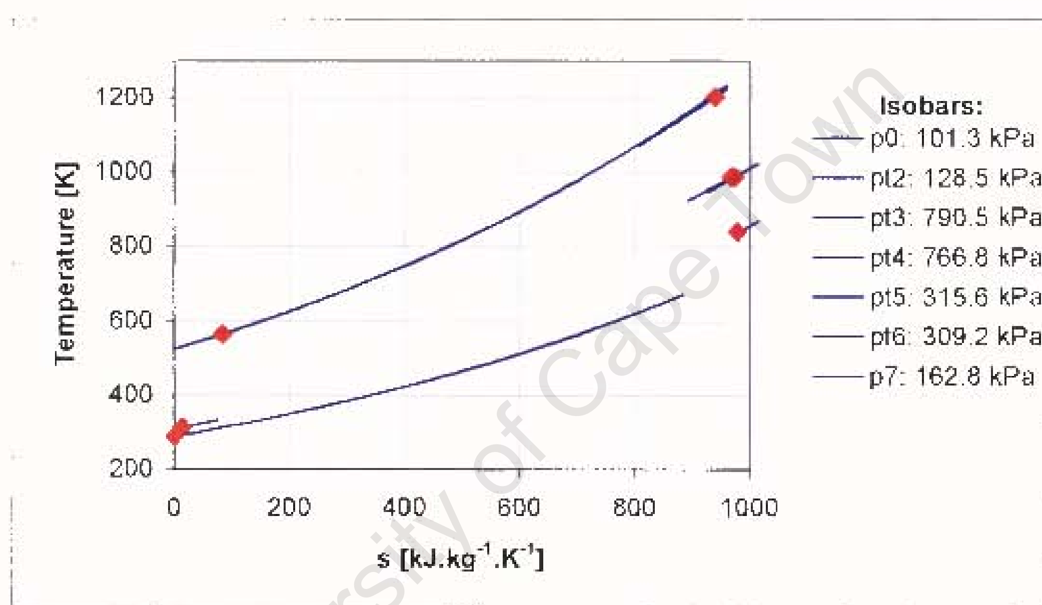


Figure 7.1 : T-s Diagram for the 9k50, M=0.66, A/B = Off, Sea Level

University of Cape Town

Figure 7.2 shows the T-s diagram for the M53. Key differences between the 9k50 and the M53 are also evident - the M53 has:

- a compressor exit pressure of 1260 kPa
- a turbine inlet temperature of 1600K.
- a bypass stream

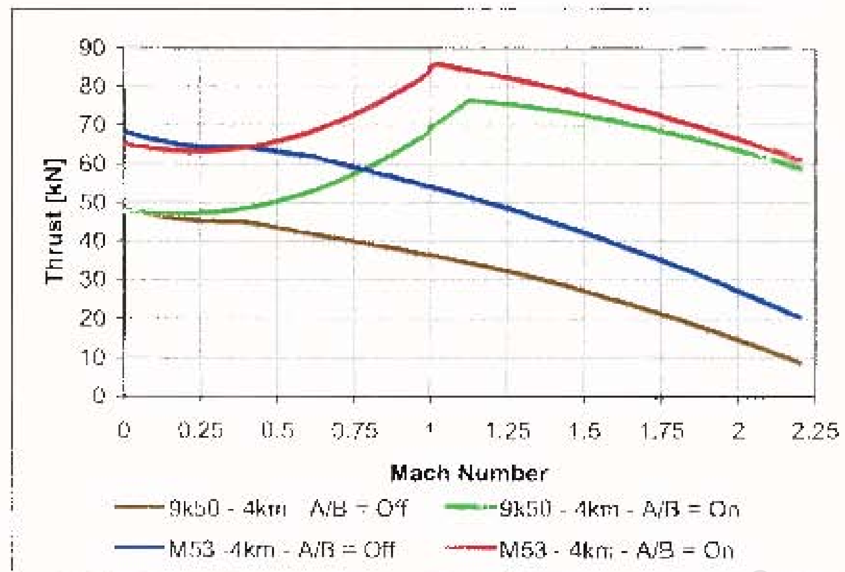


Figure 7.8: Comparison of Thrust output between the 9k50 and M53 at 4 km

At an altitude of 4km, the M53 again outperforms the 9k50. For the non-afterburning case, the performance difference is significant at high Mach numbers, but not so pronounced in the afterburning case. Below M=0.5, the net thrust produced by the M53 with the afterburner operational is marginally lower than with the afterburner off. This is due to exhaust choking.

Figure 7.9 shows the thrust output at 14km. In the non-afterburning case, the mass flow is choked by the exhaust in the subsonic regime and by the intake (low density at high altitude, thus requiring a large capture streamtube) in the supersonic regime. The discontinuity in this case is due to a discontinuity in the algorithm that calculates the mass flow rate as the intake shock pattern changes from detached to attached. However, since this discontinuity is small, the error introduced is deemed acceptable for this investigation. It is therefore also clear that the mass flow in this condition is being choked by the intake.

In the afterburning case, the mass flow is choked by the exhaust in the supersonic regime due to the converging nozzle, and the highly energised exhaust gasses.

Note:

The results presented in this section represent a theoretical solution to the equations presented in Chapter 4 and Chapter 5. Although the trends indicated in Figure 7.7, Figure 7.8 and Figure 7.9 are similar to those for certain turbojet and turbofan engines, the reader is cautioned from using these results directly. Empirical data for the performance of these engines obtained after the compilation of this document indicate that thrust output at high Mach numbers is larger than that predicted by the model. The empirical data for these engines should therefore be consulted should accurate performance data for these engines be required.

University of Cape Town

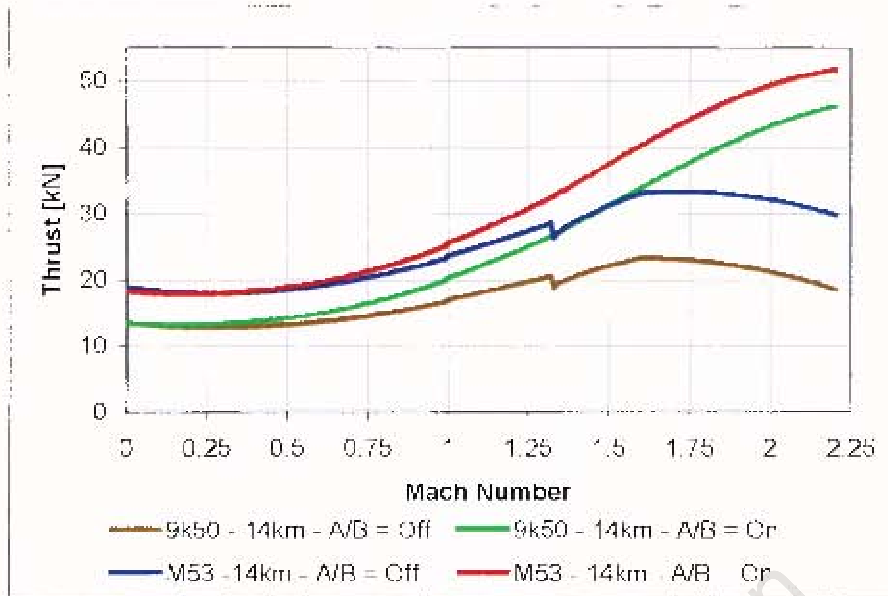


Figure 7.9: Comparison of Thrust output between the 9k50 and M53 at 14km

7.2 Comparison of the Intake Model with Standard curves

University of Cape Town

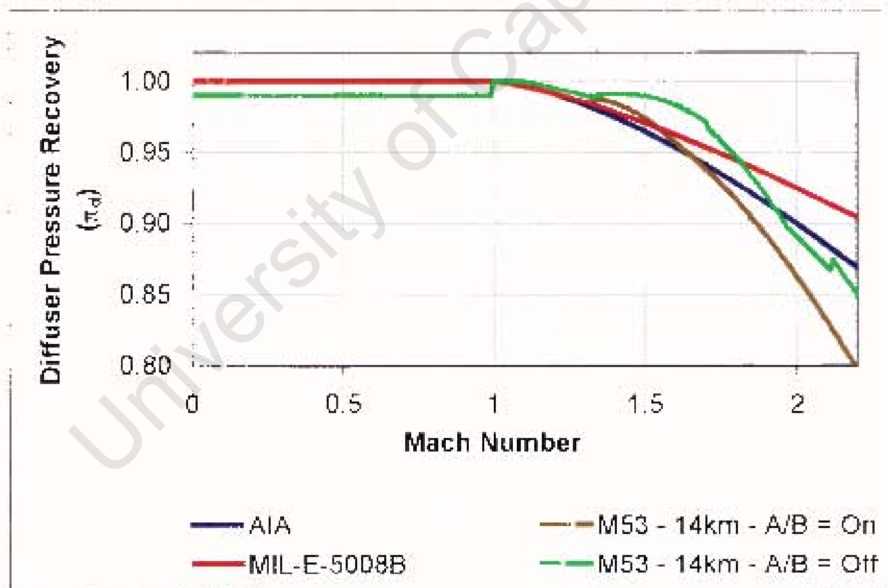


Figure 7.10: Diffuser performance comparison

Standard intake performance curves were presented in Section 5.3. Figure 7.10 presents a comparison of these curves with two intake performance curves calculated for the M53 engine configuration at an altitude of 14km. The subsonic recovery was specified to be 0.99 in this configuration.

In the afterburner off condition, the curve displays small discontinuities as the intake operating mode switches from subsonic to detached, to critical and then supercritical. It can be seen that the deviation of this curve from the standard curves is small (between 3% and 5%).

In the afterburner on condition, the engine is choked by the exhaust and the intake operates in subcritical mode. Thus the lower pressure recovery indicated by this curve is expected

7.3 Thrust - Drag - Load Factor chart

From the relationship given in Eq. 3.7, Figure 7.11 can be plotted for a given aircraft mass and altitude. It shows that as the load factor, n, is increased at a given Mach number, a higher lift coefficient C_L is required. The achievable load factor is therefore limited by the maximum lift coefficient that the aircraft is capable of at a given flight condition.

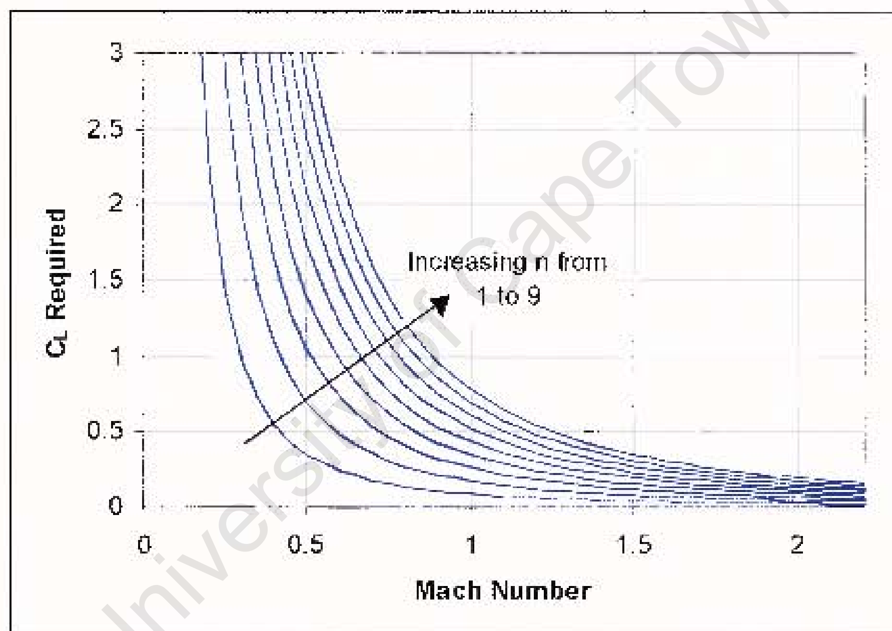


Figure 7.11: C_L required for a given load factor and Mach number at 4km

From Figure 7.11, and the aerodynamic data, where the calculated value for C_L lies within the aircraft's capabilities, the corresponding drag coefficient, C_D, can be determined. This is plotted in Figure 7.12 where the lines of constant n shown for C_L in Figure 7.11 are now plotted for C_D.

University of Cape Town

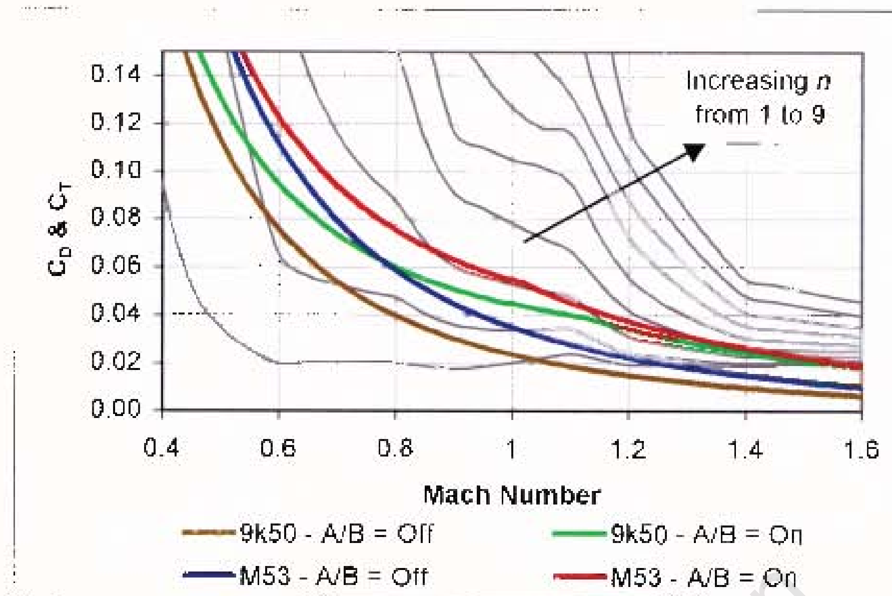


Figure 7.12: C_D for $n=1$ to $n=9$ with C_T superimposed at 4km

In addition, values for the thrust coefficient, C_T, are also plotted for the two engines under consideration, for both the afterburning and non-afterburning cases. From this plot, the maximum Mach number that the aircraft can attain in level flight ($n = 1$) at an altitude of 4000m for the various engine configurations can be determined from the intersection of the C_T and C_D at $n = 1$ curves. This shows that the M53 can attain a higher Mach number than the 9k50 in both the afterburning and non-afterburning cases, with the effect being more pronounced for the non-afterburning case. This is as expected from the results shown in Figure 7.8.

Furthermore, Figure 7.12 shows the maximum attainable load factor for a given flight condition. Here it can be seen that at an altitude of 4km, the maximum load factor that the aircraft can attain with the 9k50 is just in excess of 2 in the subsonic range with for both the afterburning and non-afterburning cases, and just in excess of 3 in the low supersonic (transonic) range with the afterburner operational.

The M53 is seen to perform considerably better, achieving a load factor well in excess of 2 in the subsonic case, and in excess of three in the supersonic case with the afterburner operational.

The maximum load factors indicated by these results would appear to be quite low compared to the maximum structural load factor of 8 specified in the simulations. However, for all these simulations, the *maximum takeoff mass* of a Mirage III, 13700kg, was utilised. The *'clean'* mass is closer to 9000kg, and the empty mass is around 7000kg. This heavy aircraft mass would therefore account for the apparently poor performance of this hypothetical aircraft.

Figure 7.13 extends the concepts illustrated in Figure 7.12 to various altitudes for the four engine configurations under consideration. The outer boundaries indicate the maximum

University of Cape Town

attainable Mach number for a unity load factor (where the C_T and C_D curves intersect for unity n). The internal boundaries represent the maximum Mach number achievable for a given load factor, or conversely, the maximum load factor that can be achieved for a given Mach number and altitude.

Many of the detailed features of these plots are common to the SEP plot given by Figure 7.14, and will be discussed in more depth in Section 7.4.

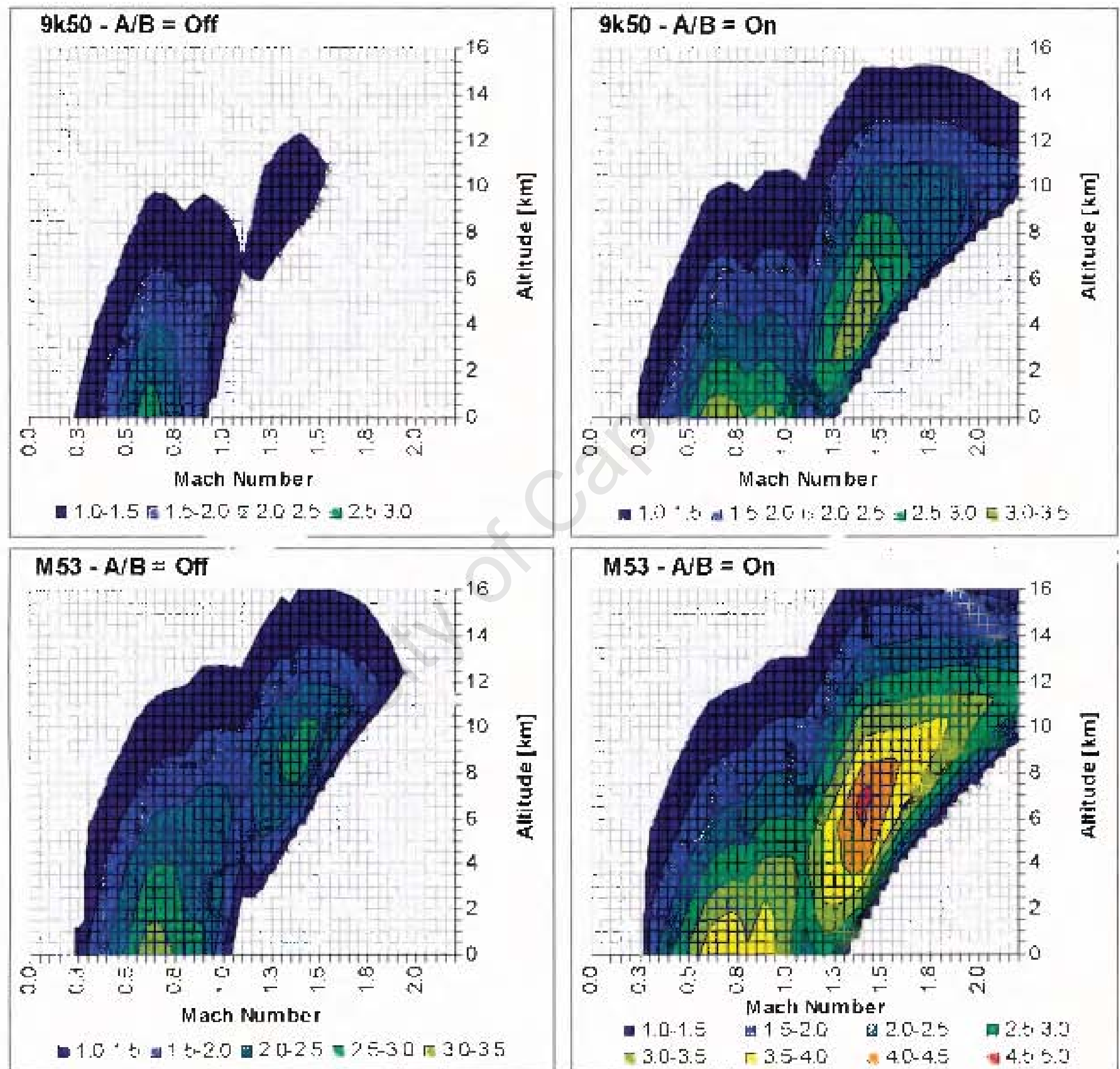


Figure 7.13: Load Factor Map for Mach number and Altitude

7.4 Specific Excess Power

The calculation of specific excess power (SEP) was discussed in Section 3.1.3, and is plotted in Figure 7.14 for the two power plants under discussion for both the afterburning and non-afterburning cases. A load factor of 1 was utilised for these plots.

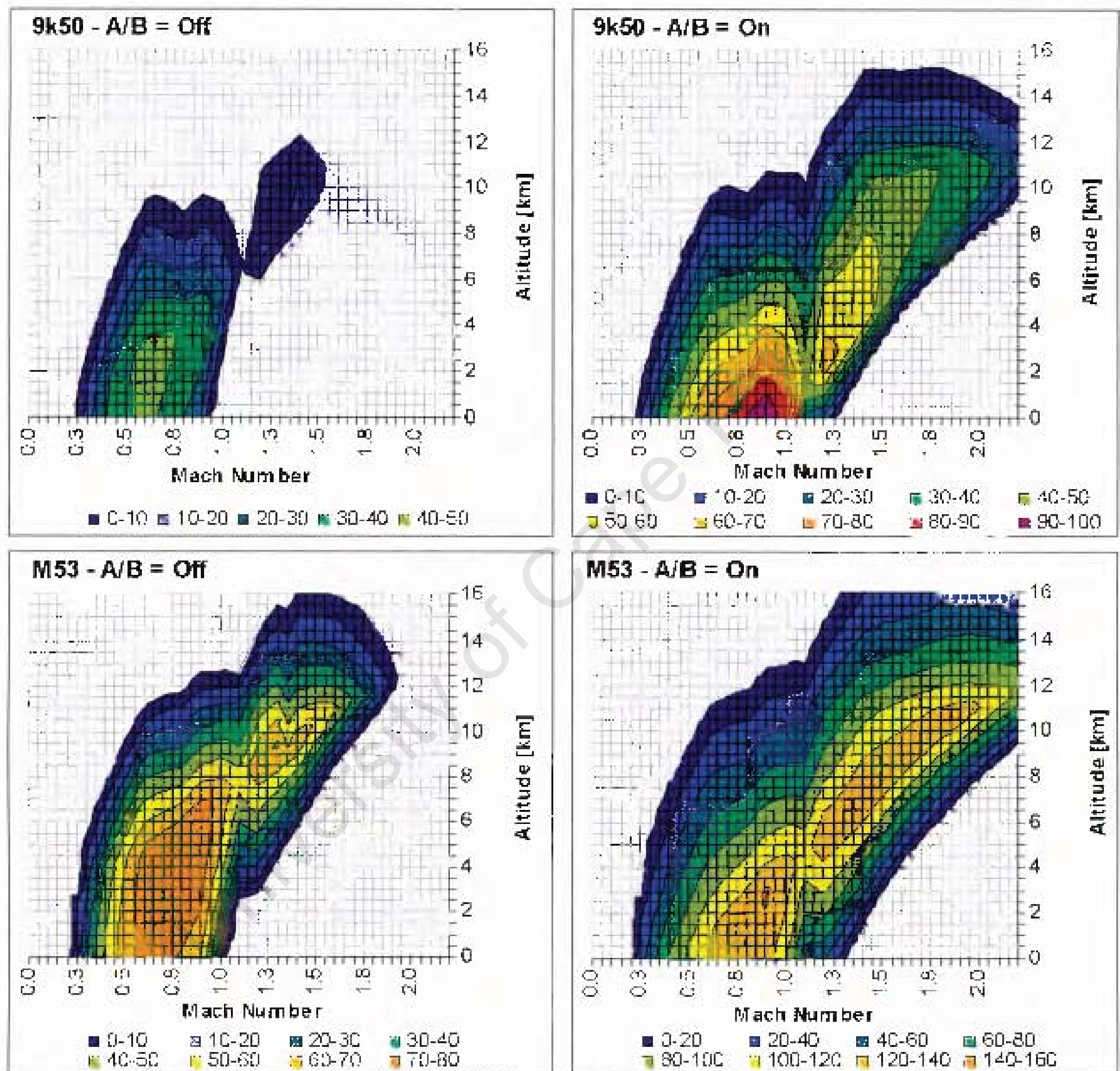


Figure 7.14: Specific Excess Power (SEP) [m/s] for the 9k50 and M53, *n*=1

Figure 7.14 reflects the data already presented in Section 7.2. The outer boundary representing an SEP of zero reflects the condition where C_T equals C_D on the high Mach number side of the plot. This was seen to be the Mach number at which the C_T and C_D curves intersected in Figure 7.12, and the maximum attainable Mach number for a load factor of unity.

Furthermore, it can be seen that the aircraft has a significantly higher service ceiling with the M53 than with the 9k50. Maximum SEP is also significantly higher with the M53 than with the

9k50. The non-afterburning M53 is able to offer SEP values close to those for the afterburning 9k50.

The decrease in SEP evident in all four plots around $M=1$ is primarily due to the significant rise in C_D in the transonic region.

7.5 Sustained Turn Rate

The concept and calculation of Sustained Turn Rate (STR) was discussed in Section 3.1.4. STR is plotted in Figure 7.15 for the two powerplants under discussion for both the afterburning and non-afterburning conditions.

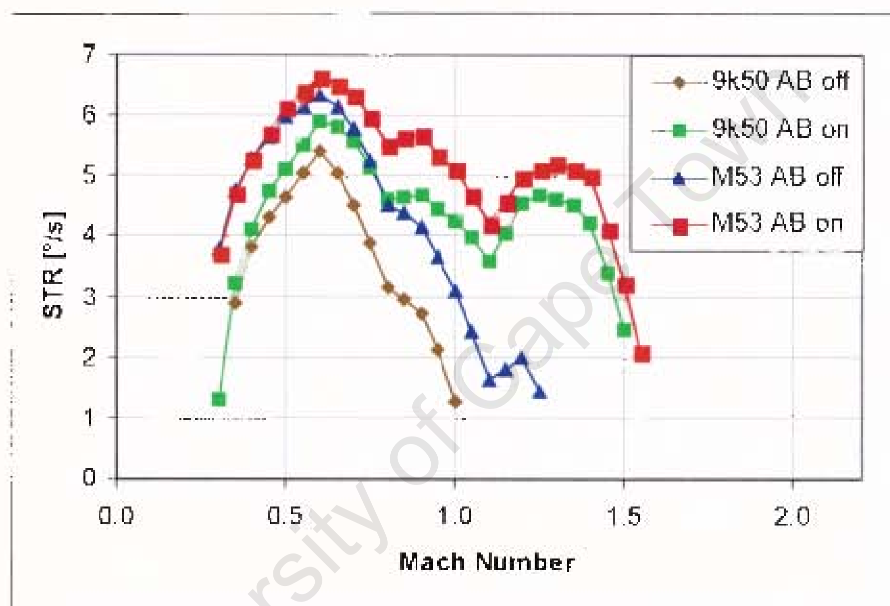


Figure 7.15: Sustained Turn rate comparison for the 9k50 and M53 at 4km

University of Cape Town

The effect of drag rise in the transonic regime is again evident. Furthermore, the highest STR occurs at high subsonic Mach numbers, one of the primary reasons for combat taking place in this regime [6]. Again, the M53 appears to offer a higher STR in both the afterburning and non-afterburning configurations. At the peak STR condition, the non-afterburning M53 offers even better performance than the afterburning 9k50.

It is also interesting to note that the values for STR shown in Figure 7.15 are significantly lower than those typically found in literature [1, 6]. Eq. 3.8 indicates that if the thrust matches the drag, the maximum attainable load factor is directly proportional to the thrust to weight ratio. For the aircraft and engine configurations under consideration, the thrust to weight ratio (weight between 94kN and 134kN, Thrust < 65kN for the 9k50 and < 94kN for the M53 at sea level) is significantly less than unity, and hence the low sustained turn rate capability.

Figure 7.16 contains the same results as Figure 7.15, however, the Attainable Turn Rate (ATR) has been superimposed. Where STR occurs at a thrust to drag ratio of one, the ATR is limited by the maximum lift attainable and the structural strength of the aircraft, i.e. the maximum load factor that it is capable of withstanding. The left hand boundary (low Mach numbers) is limited by maximum lift constraint, while the maximum load factor constraint is manifested at higher Mach numbers (right hand boundary).

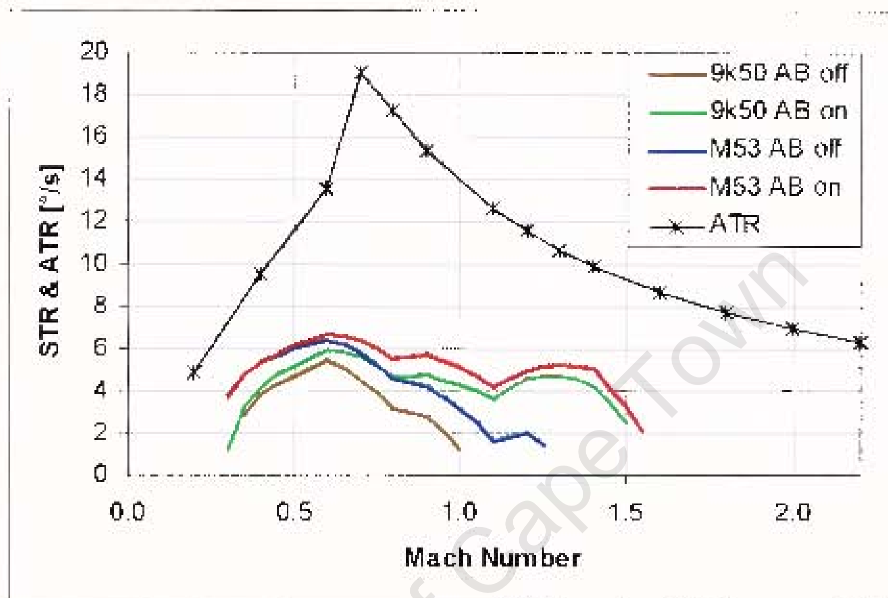


Figure 7.16: Sustained Turn rate, with Attainable Turn Rate (ATR) superimposed

University of Cape Town

ATR is significantly faster than STR, but the aircraft operates in a condition where the drag is higher than the thrust and therefore loses velocity or energy height in SEP terms. ATR is primarily an aerodynamic effect, and is only indirectly influenced through power plant by virtue of mass differences. However, since the M53 and 9k50 are similar in mass, this difference was ignored for this investigation.

7.6 Range

As discussed in Section 3.1.1, range is an important characteristic of a combat aircraft since it determines the maximum distance from which an airforce can strike at an opponent and also how deep into an opponent's territory strikes can be made.

Utilising the assumptions and simplifications given in Section 3.1.1, Eq. 3.3 can provide an estimate of the cruising range of an aircraft. In order to compare the engines under discussion, a baseline range was calculated for the 9k50 without the afterburner at a Mach number of 0.6 and altitude of 4000m. The other engine configurations were then rated against this, as shown in Figure 7.17. The velocities and drag to lift ratios for the two Mach numbers considered is shown in Table 7.3. The 'full' and 'empty' masses were estimated at 13700kg (maximum rated

take-off mass) and a fuel load of 3300 litres, yielding an empty mass of 11320kg. These parameters indicate a range of 3211km for the baseline range (green bar in Figure 7.17). Jane's [3] suggests a combat radius of 1200km or 2400km round trip for the MHI. This would compare favourably with the range obtained for a Mach number just in excess of 1

Table 7.3: Flight conditions for Range estimation

Mach Number	Velocity [m/s]	Drag/Lift Ratio (ε)
0.6	194.8	0.151
1.1	327.8	0.329
1.2	389.5	0.373

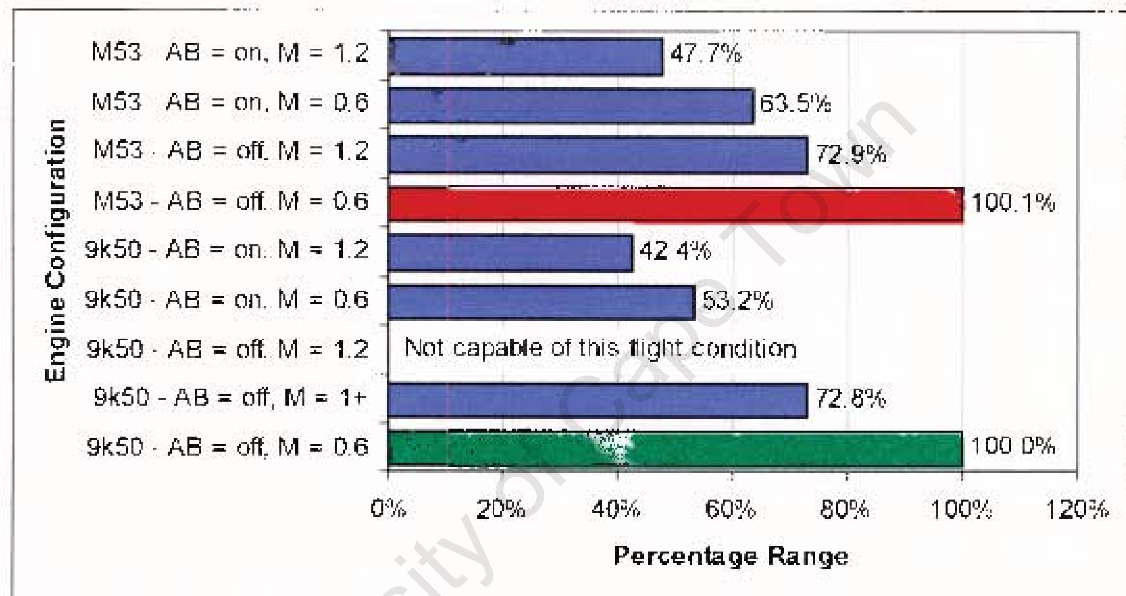


Figure 7.17: Relative Range for different engine configurations at 4000m

Since only the TSFC parameter was varied in Eq. 3.3 for the two different engines, Figure 7.17 reflects the trends shown for TSFC in Section 7.1.3. At 4000m, the M53 is seen to provide only a marginal improvement over the 9k50 for dry operation at M=0.6. The 9k50 is unable to provide sufficient thrust for the aircraft to cruise at a Mach number of 1.2, and the M53 therefore shows a significant improvement over the 9k50. Furthermore, the non-afterburning M53 at a Mach number of 1.2 provides marginally more range than the non-afterburning 9k50 at its maximum Mach number (just in excess of 1). However, in the afterburning condition, the M53 shows an all-round increased range over the 9k50.

The drop in range from a Mach number of 0.6 to 1.2 is due to the increase in the drag/lift ratio.

University of Cape Town

Chapter 8: Conclusions and Recommendations

8.1 Conclusions

From the results presented in Section 7.6, the M53 results in a range capability which is only marginally better than with the 9k50 in the dry case. However, the M53 does allow a higher Mach number to be achieved in the dry case. In the afterburning case, the M53 shows a distinct improvement over the 9k50 in terms of range. The M53 therefore presents a tactical advantage in an interception role where range and speed are required.

When the two engines are compared in terms of Specific Excess Power, as given in Section 7.4, the M53 presents a clear advantage over the 9k50. It allows a larger flight envelope, and presents an opportunity for attainable performance to be exploited by allowing a more rapid re-acquisition of lost energy.

The results obtained for Sustained Turn Rate in Section 7.5 again indicate that an airframe equipped with an M53 engine has an advantage over one equipped with the 9k50. The M53 offers a dry peak STR which outperforms the 9k50 with an afterburner. In traditional short range combat, this would offer significant benefits.

Section 7.3 illustrated that the M53 allows significantly higher load factors to be achieved which is reflected in the STR and SEP data. Furthermore, the M53 allows higher maximum speeds to be achieved than the 9k50.

In general, the M53 appears to offer better all-round performance for the hypothetical aircraft. Its ability to produce significantly more dry thrust than the 9k50 allows supersonic cruising without the use of an afterburner (supercruise). This has additional benefits beyond the scope of detailed discussion in this dissertation such as minimising the aircraft's infrared signature.

A conclusion can therefore be reached from this preliminary investigation that supercruise capability for an old airframe does offer tactical advantages over other similarly classed combat aircraft. However, these performance gains should be viewed in light of a new generation of aircraft that have superior handling and agility that arises not only from their superior power plants, but also from their aerodynamics and control systems.

A conclusion can also be drawn from the results obtained that the simulation algorithms and routines developed performed satisfactorily and the results obtained compare favourably to published data, but that scope remains for improvement.

University of Cape Town

8.2 Recommendations

This simulation exercise has provided a preliminary indication that supercruise capability for an old airframe offers some tactical advantages. A more detailed simulation of the aircraft in combat should be performed to obtain a more comprehensive evaluation of the performance gains that are potentially offered by this technology.

In terms of the simulations developed in this investigation, the following shortcomings should be addressed in future work:

Aerodynamic Model:

Although the Digital DATCOM provided useful information, it has serious shortcomings. In particular, an improved aerodynamic model should provide:

- Values for the lift and drag coefficient for trimmed flight at all flight conditions.
- A better estimation of the lift curve in the transonic region.
- An improved drag behaviour prediction.

Propulsion Model:

The propulsion model performed well and provided reasonable estimates when compared to published data. However, performance corrections for operation at less than full throttle should be developed. The interaction with the aerodynamic model should then be tightened, such that flight conditions such as lift, drag and thrust are better matched.

The model should also be enhanced to provide a better estimate of the mass flow rate through the engine.

Intake Model:

The intake model should be further developed to better account for the following:

- Mass flow rate through the intake under the various operating modes.
- Corrections for the pressure recovery at angles of attack other than zero. This should then be linked to the aerodynamic data and propulsion model to provide more realistic values for the thrust at an angle of attack.
- The effect of bleeds and auxiliary intake gates should also possibly be added.

University of Cape Town

References

1. Herbst, W.B., Dynamics of Air Combat, *Journal of Aircraft*, Vol. 20, No. 7, July 1983, pp. 594-598.
2. Braybrook, R., X-31 Update, *Air International*, Dec. 1995, pp. 331-334.
3. Various, Dassault-Breguet, *Jane's All the World's Aircraft*, S. Low, Marston & Company Ltd.,1981
4. Denning, R.M. & Mitchell, N.A, Trends in military aircraft propulsion, *Proc Instn Mech Engrs*, Vol. 203, 1989, pp. 11-23.
5. McCormick, B.W., Aerodynamics, Aeronautics and Flight Mechanics, John Wiley & Sons, New York, 1995.
6. Hancock, G.J., An introduction to the flight dynamics of rigid aeroplanes, Ellis Horwood, Hertfordshire, 1995.
7. Kerrebrock, J.L, Aircraft Engines and Gas Turbines, MIT Press, Cambridge Mass. 1977.
8. Çengel, Y.A. & Boles, M.A. Thermodynamics: An Engineering Approach, International Ed. McGraw-Hill Book Co, Singapore, 1989.
9. Hill, P.G. & Petersen, C.R., Mechanics and Thermodynamics of Propulsion, Addison-Wesley, Reading Mass. 1965.
10. Seddon, J. & Goldsmith, E.L., Intake Aerodynamics: An Account of the Mechanics of Flow in and around the Air Intakes of Turbine-engined and Ramjet Aircraft and Missiles, Collins, London, 1985.
11. Hesse, W.J. & Mumford, N.V.S., Jr., Jet Propulsion for Aerospace Applications, 2nd Ed. Pitman Publishing Corp., New York, 1964.
12. Zucrow, J.M. & Hoffman, J.D., Gas Dynamics, Vol 1, John Wiley & Sons, New York, 1976.
13. Zucrow, J.M. & Hoffman, J.D., Gas Dynamics - Multidimensional Flow, Vol 2, John Wiley & Sons, New York, 1977.
14. Anderson, J.D. Jr., Fundamentals of Aerodynamics, Second Edition, McGraw-Hill Book Co, New York, 1991.
15. Williams, J.E. & Vukelich, S.R., The USAF Stability and Control Digital Datcom, Vol. 1, NTIS, U.S. Dept. of Commerce, Springfield, Va., 1976.

University of Cape Town

16. Williams, J.E. & Vukelich, S.R., The USAF Stability and Control Digital Datcom, Vol. 2, NTIS, U.S. Dept. of Commerce, Springfield, Va., 1976.

17. SNECMA, M53-P2 Engine, http://www.snecma-moteurs.com/en/produits/moteur_milirair/m53.htm

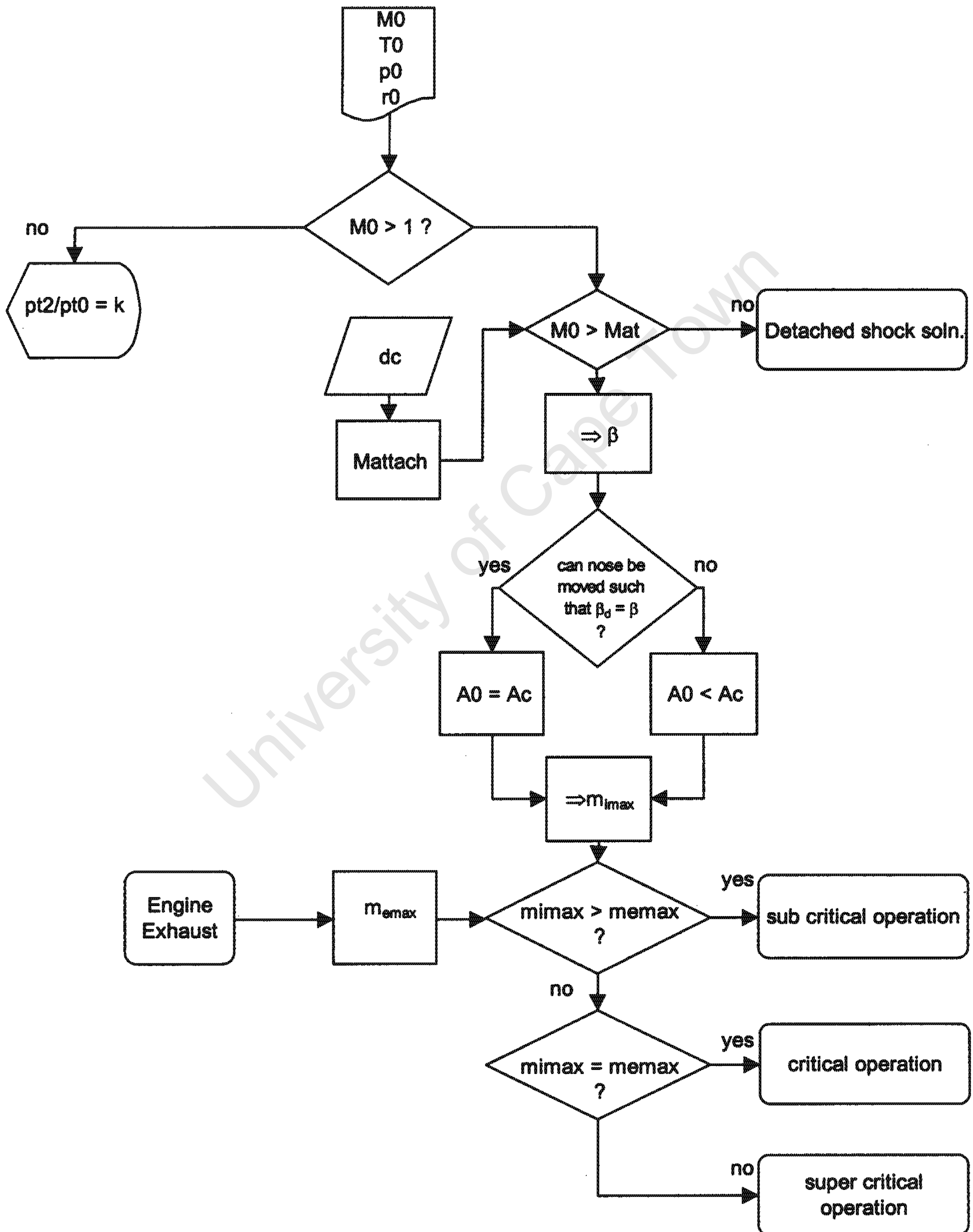
18. SNECMA, ATAR 9k50 Engine, http://www.snecma-moteurs.com/en/produits/moteur_milirair/atar.htm

19. Wilkinson, P.H., Aircraft Engines of the World 1970, Paul H. Wilkinson, Washington, D.C., 1970.

University of Cape Town

Appendix A: Intake Model

Appendix A-1 Flow Chart



University of Cape Town

Appendix A-2 Detached Shock Equations

The general equation of an hyperbola is given by

$$\frac{x^2}{a^2}-\frac{r^2}{b^2}=1$$

rewriting:

$$r=\pm\frac{b}{a}\sqrt{x^2-a^2}$$

considering only the positive portion of the hyperbola, the slope is found by taking the derivative with respect to x:

$$\frac{dr}{dx}=\frac{b}{a}\frac{x}{\sqrt{x^2-a^2}}$$

The slope of the asymptote is then given by

$$\left.\frac{dr}{dx}\right|_{\infty}=\lim_{x\to\infty}\frac{b}{a}\frac{x}{\sqrt{x^2-a^2}}$$
$$=\frac{b}{a}$$

From the Mach line condition,

$$\left.\frac{dr}{dx}\right|_{x=\infty}=\tan\mu$$

where

$$\mu=\sin^{-1}\left(\frac{1}{M_\infty}\right)$$

Thus,

$$\frac{b}{a}=\tan\left(\sin^{-1}\left(\frac{1}{M_\infty}\right)\right)$$

The value for a must now be chosen such that the slope at $r = r_c$ is greater than the slope of the conical shock wave generated at M_{attach}. Solving for x in terms of r,

$$\frac{dr}{dx}=\frac{b}{a}\frac{1}{r}\sqrt{r^2+b^2}$$

University of Cape Town

Now,

$$b^2 = \left(\frac{b}{a}\right)^2 a^2$$

Substituting, and solving for a,

$$a = \frac{r\sqrt{\left(\frac{dr}{dx}\right)^2 - \left(\frac{b}{a}\right)^2}}{\left(\frac{b}{a}\right)^2}$$

It will be assumed that the slope of the hyperbola at $r = r_c$ is a linear function of Mach number between 90° (normal) and ε_m, the angle of the conical shock at M_{attach}.

Thus the angle of the detached shock at the capture radius is given by:

$$\theta_{r_c} = \frac{\pi/2 - \varepsilon_m}{1 - M_{attach}}(M_\infty - 1) + \frac{\pi}{2}$$

and the slope is therefore:

$$\left.\frac{dr}{dx}\right|_{r=r_c} = \tan(\theta_{r_c})$$

This completely defines the shape of the hyperbola, but not its position relative to the apex of the cone. However, since the mass flow rate the detached shock condition is assumed to be controlled by the mass flow rate through the engine, the capture streamtube diameter is not required, and therefore the position of the shock is not required.

In order to calculate the pressure recovery, a capture streamtube with radius r_c is assumed, and this area is then divided into annuli of equal area as described in Section 5.4.1.

University of Cape Town

Appendix A-3 Sample Conical Shock Data

The following data shows the typical output from the Taylor Macoll flow calculation program. The data represents solutions for a 30° cone semi-angle at sea level (temperature of 288.15K). The Mach number of attachment may be determined by requesting output between Mach numbers of 1.4 and 1.6 with a smaller increment size.

```
M1   eps      M2        p2/p1     r2/r1     Mc        pc/p1     rc/r1
1.2  No Solution at this Mach Number
1.4  No Solution at this Mach Number
1.6  58.288   1.07131   1.99486   1.62199   0.932775  2.34512   1.82061
1.8  51.9199  1.22905   2.17556   1.71871   1.10661   2.54597   1.92294
2    48.0819  1.36293   2.41734   1.84163   1.2534    2.80687   2.04898
2.2  45.4174  1.48558   2.69797   1.97573   1.38651   3.10648   2.18501
2.4  43.4488  1.60008   3.01169   2.11574   1.50954   3.43949   2.32625
2.6  41.9378  1.70754   3.3562    2.25867   1.62403   3.80392   2.46996
2.8  40.7461  1.80853   3.73034   2.40243   1.73085   4.19884   2.61423
3    39.7866  1.90341   4.13351   2.54546   1.83062   4.62375   2.75757
3.2  39.0012  1.99247   4.56533   2.68654   1.92381   5.07841   2.89884
3.4  38.3493  2.07598   5.0256    2.82474   2.01084   5.56267   3.03716
3.6  37.8018  2.15423   5.51415   2.95934   2.0921    6.07642   3.17182
3.8  37.3374  2.22749   6.03089   3.0898    2.16796   6.61961   3.30231
4    36.9398  2.29604   6.57576   3.21576   2.23877   7.1922    3.42825
4.2  36.5968  2.36015   7.14872   3.33694   2.30486   7.79417   3.54942
4.4  36.2988  2.4201    7.74972   3.45319   2.36655   8.4255    3.66564
4.6  36.0382  2.47614   8.37875   3.56446   2.42413   9.0862    3.77687
4.8  35.8089  2.52855   9.03579   3.67073   2.4779    9.77624   3.88311
5    35.6062  2.57756   9.7208    3.77207   2.52812   10.4956   3.98441
5.2  35.4261  2.62339   10.4338   3.86858   2.57504   11.2443   4.08088
5.4  35.2653  2.66628   11.1749   3.96039   2.61891   12.0224   4.17264
5.6  35.1212  2.70643   11.9439   4.04764   2.65994   12.8299   4.25985
5.8  34.9916  2.74403   12.7408   4.13051   2.69834   13.6666   4.34268
6    34.8745  2.77926   13.5658   4.20917   2.73429   14.5327   4.42131
###Start Time: 956488692
###End Time: 956488721
Total Time: 29seconds
```

University of Cape Town

Appendix A-4 Intake Model: Class CConeIntake

A.4.1 Header File: ConeIntake.h

```
// ConeIntake.h: interface for the CConeIntake class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_CONEINTAKE_H__4B68A533_2017_11D1_BE68_000000000000__INCLUDED_)
#define AFX_CONEINTAKE_H__4B68A533_2017_11D1_BE68_000000000000__INCLUDED_

#include "..\LOOKUPTABLE\LOOKUPTB.HPP" // Added by ClassView
#include "..\IdealGas\IdealGas.h"   // Added by ClassView
#include "../IdealGas/IdealGasStream.h"

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CConeIntake
{
public:

   void GetIntakeGeometry(double &rc, double &L, double &dx);

   LookupTable * ReplaceLUT(const char *lutfname);
   double PiD(CIdealGasStream* fluid, double aoa, const double mdotmin, const double mdotmax, double& mdot);
   double PiD(double M0, CIdealGasStream* fluid);
   void SubsonicPiD(double newPiD);
   double PiD(double M0);

   double GetRc(void) const { return m_rc; }

   void SetMassFlows(double mdmax, double mdsuck) { m_mdotmax = mdmax; m_mdotsuction = mdsuck; }

   void ManDesign(double rcapture, double dL, double travel);
   int AutoDesign(double massflow, double mdotM, double rho, double Mmax);
   int AutoDesign(double massflow, double mdotM, double rho, double Mmax, double BetaMin);
    CConeIntake(double ConeAngle, CIdealGasStream *Fluid, const char* lutfname);
   CConeIntake();
   virtual ~CConeIntake();
```

University of Cape Town

```
    typedef enum enum_OPMODE_tag {   eOM_SUBSONIC, eOM_DETACHED,
                                     eOM_SUBCRITICAL, eOM_CRITICAL,
                                     eOM_SUPERCRITICAL } eOPMODE;

    eOPMODE LastOpMode(void) { return m_opmode; }

protected:
    int CalcSuperCritPiD(double massflow, double flowratio, double & pid);
    int SubCritPiD(double massflow, double flowratio, double &pid);
    int CalcCritCapture(double &r0);
    double DetachedShock(double M0, double *pmassflowrate=NULL);
    double m_betadmax;
    double m_betadmin;
    double m_subpid;
    CIdealGasStream* m_pfluid;
    double m_L;
    double m_dx;
    double m_rc;
    LookupTable *m_pLUTCSP;
    double m_deltac;

    double m_mdotmax;
    double m_mdotsuction;

    double m_crittol;    // critical mass flow tolerance (Fraction of critical mass flow)

    eOPMODE m_opmode;

    enum enum_LUTCOLS {MACHNO0, SANGLE, MACHNOS, P2_P1S, R2_R1S, MACHNOC, PC_P1, RC_R1};

};

#endif // !defined(AFX_CONEINTAKE_H__4B68A533_2017_11D1_BE68_000000000000__INCLUDED_)
```

A.4.2 Implementation File: ConeIntake.cpp

```
// ConeIntake.cpp: implementation of the CConeIntake class.
//
//////////////////////////////////////////////////////////////////////
/*
#include "LookupTable.hpp"
#include "IdealGas.h"
```

```
*/
#include <stdlib.h>
#include <math.h>
#include <mathutil.h>
#include <fstream.h> // for cout
#include "ConeIntake.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CConeIntake::CConeIntake()
{

   m_deltac = 0.0;
   m_pfluid = NULL;
   m_pLUTCSP = NULL;

   m_crittol = 0.05;
}

CConeIntake::~CConeIntake()
{

   // if we've created a lookup table, free the memory it uses
   if(m_pLUTCSP) delete m_pLUTCSP;

}

CConeIntake::CConeIntake(double ConeAngle, CIdealGasStream *Fluid, const char * lutfname)
{

   m_deltac = ConeAngle;
   m_pfluid = Fluid;
   m_pLUTCSP = new LookupTable(lutfname, LookupTable::LINEAR, LookupTable::D1D);

   m_crittol = 0.05;

}


int CConeIntake::AutoDesign(double massflow, double mdotM, double rho, double Mmax)
{

   if(!m_pLUTCSP->IsOK()) return 0;
```

University of Cape Town

```
    // get the shock angle for the max Mach number (degrees)
    if(m_pLUTCSP->OutRange(Mmax)) return 0;
    double MmaxBeta = DTR(m_pLUTCSP->Lookup(Mmax, (long)SANGLE));
    double Mmin = m_pLUTCSP->RangeMin();
    double MminBeta = DTR(m_pLUTCSP->Lookup(Mmin, (long)SANGLE));

    // calculate the capture radius based on mass flow and a Mach number for that flow
    m_rc = sqrt( massflow/(rho*M_PI*mdotM*m_pfluid->SonicV()) );

    // calculate the distance from the cone apex to the intake lip,
    // assuming that the conical shock touches the intake lip at Mmax
    m_L = m_rc/tan(MmaxBeta);

    // calc the travel dist of the cone as half the dist
    // required to make the lowest attached Mach number touch the intake lip
    m_dx = 0.5*(m_L - m_rc/tan(MminBeta) );
//  m_dx = (m_L - m_rc/tan(MminBeta) );

    // store the minimum and maximum values of betad
    m_betadmin = MmaxBeta;        // note: smallest beta at largest Mach
    m_betadmax = atan(m_rc/(m_L - m_dx)); // vice versa

    return 1;
}

void CConeIntake::ManDesign(double rcapture, double dL, double travel)
{

    m_rc = rcapture;
    m_L = dL;
    m_dx = travel;

    // store the minimum and maximum values of betad
    m_betadmin = atan(m_rc/m_L);        // note: smallest beta at largest Mach
    m_betadmax = atan(m_rc/(m_L - m_dx)); // vice versa

}

// determine the critical mode pressure recovery
double CConeIntake::PiD(double M0)
{

    double Mattach;
    double beta;
    double Mc, Ms;  // Mach numbers just after shock and on cone surface
    double Mlip;   // Mach number at intake lip
```


University of Cape Town

```
double Mav;    // average Mach number
double Thetalip; // flow angle at the lip

double pt1_p0; // first stage stagnation/static pressure recovery
double pt2_pt1;   // second stage stagnation pressure recovery

// if flow is subsonic, return constant recovery
if(M0<1.0) {
   m_opmode = eOM_SUBSONIC;
   return m_subpid;
}

// check if shock is attached.
Mattach = m_pLUTCSP->RangeMin();
if(M0 < Mattach) {
   m_opmode = eOM_DETACHED;
   return DetachedShock(M0);
}

// we now have an attached shock. determine its geometry
beta = DTR(m_pLUTCSP->Lookup(M0, (long)SANGLE));

// determine the Mach number on the cone surface
Mc = m_pLUTCSP->Lookup(M0, (long)MACHNOC);

// check if betadmin <= beta <= betadmax
if( (m_betadmin <= beta) && (beta <= m_betadmax) ) {
   Mlip = m_pLUTCSP->Lookup(M0, (long)MACHNOS); // Mlip = Ms
}
else if(beta > m_betadmax) {
   m_pfluid->TMRayProps(M0, beta, m_betadmax, Mlip, Thetalip);
}

// calc the average Mach number at the cowl inlet.

Mav = 0.5*(Mlip + Mc);  // note that this might not be quite accurate
                        // since the Mach number distribution probably
                        // isn't linear across the cowl opening.

// calculate the pressure recovery across the shock system

pt1_p0 = m_pLUTCSP->Lookup(M0, (long)PC_P1)*m_pfluid->pt_p(Mc);

// check if a normal shock will occur:
if(Mav > 1.0) pt2_pt1 = m_pfluid->NS_pt2_pt1(Mav);
else pt2_pt1 = 1.0;
```

University of Cape Town

```
   return pt2_pt1*pt1_p0/m_pfluid->pt_p(M0);

}

void CConeIntake::SubsonicPiD(double newPiD)
{
// specify a constant pressure recovery for subsonic operation

   if(newPiD <= 1.0) m_subpid = newPiD;
   else m_subpid = 1.0;

}

double CConeIntake::DetachedShock(double M0, double *pmassflowrate)
{

// this calculates the pressure recovery when a detached bow shock is present

   double a; // constant in equation of hyperbola
   double b_a; // ratio of b/a - constant in hyperbola
   double bsq; // b^2
   double theta_rc;  // slope of shock at capture radius
   double betaatt;   // conical shock angle for attached shock at smallest Mach number
   double Mmin;    // lowest Mach number which results in attached shock
   int n;
   double rn;      // radius being evaluated
   double dr_dx;  // slope at given radius
   double pt2_pt1;
   double r2_r1;

   // from the Mach line condition at x = infinity,
   b_a = tan(asin(1.0/M0));

   Mmin = m_pLUTCSP->RangeMin();
   betaatt = DTR(m_pLUTCSP->Lookup(Mmin, (long)SANGLE));

   // estimate the slope at the capture radius
   theta_rc = (M_PI_2 - betaatt)*(M0-1.0)/(1.0-Mmin) + M_PI_2;

   a = m_rc*sqrt(SQ(tan(theta_rc)) - SQ(b_a))/SQ(b_a);

   bsq = SQ(b_a*a);

   // divide the capture streamtube into 3 rings of equal area, evaluate props
   // at the weighted mean radius of each ring
```

University of Cape Town

```
   pt2_pt1  = 0.0;

   for(n=0;n<6;n+=2) {
      rn = m_rc*sqrt((n+1.0)/6.0);     // calc the radius
      dr_dx = b_a*sqrt(SQ(rn)+bsq)/m_rc;     // calc the slope
      pt2_pt1  += m_pfluid->OS_pt2_pt1(M0, atan(dr_dx)); // get the pressure recovery
   }

// cout << M0 << "\t" << b_a << "\t" << a << "\t" << RTD(theta_rc) << "\t" << pt2_pt1/3.0 << endl;

   // calculate the mass flow rate if we've been asked for it
   if(pmassflowrate)

      /* Note that if we assume that the flow angle is negligible up to
         the intake lip, then the mass flow into the intake is the same
         as the mass flow for the capture streamtube BEFORE the shock.
         this follows from the fact that v2/v1 = r1/r1 (JEA John Eq 4.12)
      */

      *pmassflowrate = m_pfluid->r()*m_pfluid->v()*M_PI*SQ(m_rc);


   // return the average pressure recovery
   return pt2_pt1/3.0;

}

double CConeIntake::PiD(double M0, CIdealGasStream * fluid)
{
   m_pfluid = fluid;
   return PiD(M0);
}

double CConeIntake::PiD(CIdealGasStream *intakefluid, double aoa, const double mdotmin,
                        const double mdotmax, double & mdot)
{

   double pid_crit;
   double M0;
   double Mattach;

   m_pfluid = intakefluid;

   // just do some basic stuff to make life simpler
   M0 = intakefluid->M();
```

University of Cape Town

```
   // calculate the critical mode pressure recovery
   pid_crit = PiD(M0);


   /* This function must return the pressure recover under all operating modes.
      Thus it takes as arguments the mass flow rate through the rest of the
      engine.  This is then used to check whether or not the back pressure
      presented by the compresser face is too high or too low resulting in
      subcritical or supercritical performance respectively.
   */

   /* The approach will be to calculate the critical mode pressure recover and then
      correct this for operating condition
   */

   /* This function must also calculate the maximum mass flow rate possible.  This
      is determined by the capture streamtube diameter.  This must therefore be
      calculated.
   */

   // follow the same initial steps as used in the critical mode recovery to determine

   /******************/
   /* SUBSONIC      */
   /******************/

   if(M0<1) {
      m_opmode = eOM_SUBSONIC;

      double mdotIntake_Max; // maximum mass flow possible into the intake
      mdotIntake_Max = SQ(m_rc)*M_PI*m_pfluid->r()*m_pfluid->v();

      // return the flow through the diffuser as the max flow through the engine
//    mdot = mdotmax>mdotIntake_Max?mdotIntake_Max:mdotmax;
//    mdot = M0*mdotIntake_Max + (1.0 - M0)*mdotmax;
//    mdot = mdotmax;

      if(mdotIntake_Max < mdotmax) {

         double v = m_pfluid->v();
         double r = m_pfluid->r();

         double vint = m_mdotmax/( SQ(m_rc)*M_PI*r);

         mdot = m_mdotsuction + v/vint*(m_mdotmax - m_mdotsuction);
```

University of Cape Town

```
        if(mdot > mdotmax) mdot = mdotmax;
    }

    else mdot=mdotmax;

    return m_subpid;
}

// now check if the shock is attached
Mattach = m_pLUTCSP->RangeMin();
if(M0 < Mattach) {

/******************/
/* DETACHED SHOCK */
/******************/

    m_opmode = eOM_DETACHED;

    // The mass flow in this case is determined by the mass flow through the
    // engine or the mass flow possible through the capture streamtube.

    double masstemp, p2_p1;

    p2_p1 = DetachedShock(M0, &masstemp);

    mdot = (masstemp<mdotmax)?masstemp:mdotmax;

    return p2_p1;

}

/******************/
/* ATTACHED SHOCK */
/******************/

/*
    Here the critical part is to determine the operational mode, i.e. whether
    or not we are sub, super or just critical.  Once we determine this, we need
    to apply some correction for pressure recovery from the critical value
*/

double mdotIntake_Max;  // maximum mass flow possible into the intake
double flowratio;
double r0;
```

University of Cape Town

```
    // determine the capture radius for critical flow
    if(CalcCritCapture(r0)==0) r0 = m_rc;

    // the max flow is given by the capture streamtube area.
    mdotIntake_Max = SQ(r0)*M_PI*m_pfluid->r()*m_pfluid->v();

    flowratio = (mdotmax - mdotIntake_Max)/mdotIntake_Max;

    if(flowratio > m_crittol)
       /* The engine can handle more mass flow than the intake.  Therefore, back pressure
          that the diffuser sees will be low and we will probably operate in supercritcal
          mode
       */
       m_opmode =  eOM_SUPERCRITICAL;
    else if(flowratio < -m_crittol)
       /* The maximum engine mass flow is less than the critical intake flow rate. The
          engine therefore appears as a high back pressure to the intake and the intake
          will operate in subcritical mode
       */
       m_opmode =  eOM_SUBCRITICAL;
    else
       m_opmode = eOM_CRITICAL;

    switch(m_opmode) {
       case eOM_CRITICAL:
          mdot = (mdotmax > mdotIntake_Max)?mdotIntake_Max:mdotmax;
          return pid_crit;

       case eOM_SUPERCRITICAL:
          mdot = mdotIntake_Max;
//        return pid_crit * sqrt(1.0 - flowratio);
          double pisup;
          CalcSuperCritPiD(mdot, flowratio, pisup);
          return pisup;

       case eOM_SUBCRITICAL:
          mdot = mdotmax;
//        return pid_crit * pow((1.0 + flowratio), 1.0/3.0);
          double pisub;
          SubCritPiD(mdot, flowratio, pisub);
          return pisub;
    }

    return pid_crit;

}
```

University of Cape Town

```
int CConeIntake::CalcCritCapture(double & r0)
{

   double beta;
   double Mlip, M0;

   // flow angle after the shock and at the lip
   double thetas, Thetalip;
   double theta;
   double Ld;

   M0 = m_pfluid->M();


   // we now have an attached shock. determine its geometry
   beta = DTR(m_pLUTCSP->Lookup(M0, (long)SANGLE));

   // check if betadmin <= beta <= betadmax
   if( (m_betadmin <= beta) && (beta <= m_betadmax) ) {

      // we can move the cone, so the capture radius will be the
      // same as the cowl inlet

      r0 = m_rc;
      return 1;

   }
   else if(beta > m_betadmax) {

      // get the mach number at the lip as well as the flow angle
      m_pfluid->TMRayProps(M0, beta, m_betadmax, Mlip, Thetalip);


      // get the flow angle after the shock
      thetas = m_pfluid->ShockFlowAngle(M0, beta);

      theta = 0.5*(thetas + Thetalip);

      // get the cone position - it will be at an extreme
      Ld = m_L - m_dx;

      r0 = (m_rc - Ld*tan(theta)) / (1.0 - tan(theta)/tan(beta));
      return 1;

   }
```

University of Cape Town

```
   return 0;
}

int CConeIntake::SubCritPiD(double massflow, double flowratio, double & pid)
{

   double M0, v0, rho;
   double Mc, Ms, Mav;
   double beta;

   double r0;

   // determine the capture streamtube for the massflow given
   v0 = m_pfluid->v();
   rho = m_pfluid->r();


   r0 = sqrt( massflow / (M_PI*rho*v0) );

   // we now have an attached shock. determine its geometry
   M0 = m_pfluid->M();
   beta = DTR(m_pLUTCSP->Lookup(M0, (long)SANGLE));

   // intersection between the normal and conical shocks
   double rint;
// rint = -m_rc * flowratio;
   rint = m_rc * (1.0+flowratio);
   // note that rint is based on an assumption that the normal shock moves in a
   // linear fashion to flow ratio.

   // everything at a capture radius lower than rint passes through both
   // shocks.  everything above, through only the normal;


   // areas
   double Aboth;
   double Anorm;
   double Atotal;

   Atotal = M_PI*SQ(r0);

   if( r0 < rint) { // the entire capture streamtube passes through both

      Aboth = Atotal;
      Anorm = 0.0;
```

University of Cape Town

```
   }
   else {

      Aboth = M_PI*SQ(rint);
      Anorm = Atotal - Aboth;
   }

   // recoveries
   double piconical;
   double pinormal;
   double pt1_p0_c;
   double pt2_pt1_c;

   // determine the Mach number on the cone surface
   Mc = m_pLUTCSP->Lookup(M0, (long)MACHNOC);

   // determine the mach number just after the conical shock
   Ms = m_pLUTCSP->Lookup(M0, (long)MACHNOS);

   // calculate the average mach number
   Mav = 0.5*(Mc + Ms);

   // pressure ratio across the conical shock portion
   pt1_p0_c = m_pLUTCSP->Lookup(M0, (long)PC_P1)*m_pfluid->pt_p(Mc);

   // check if a normal shock will occur:
   if(Mav > 1.0) pt2_pt1_c = m_pfluid->NS_pt2_pt1(Mav);
   else pt2_pt1_c = 1.0;

   piconical =  pt2_pt1_c*pt1_p0_c/m_pfluid->pt_p(M0);

   pinormal = m_pfluid->NS_pt2_pt1(M0);

   pid = (piconical*Aboth + pinormal*Anorm)/Atotal;

   return 1;

}


int CConeIntake::CalcSuperCritPiD(double massflow, double flowratio, double & pid)
{

   double Mattach;
   double M0;
   double beta;
```

University of Cape Town

```
double Mc, Ms;  // Mach numbers just after shock and on cone surface
double Mlip;    // Mach number at intake lip
double Mav;     // average Mach number
double Thetalip; // flow angle at the lip

double pt1_p0; // first stage stagnation/static pressure recovery
double pt2_pt1;   // second stage stagnation pressure recovery


M0 = m_pfluid->M();

// we now have an attached shock. determine its geometry
beta = DTR(m_pLUTCSP->Lookup(M0, (long)SANGLE));

// determine the Mach number on the cone surface
Mc = m_pLUTCSP->Lookup(M0, (long)MACHNOC);

// check if betadmin <= beta <= betadmax
if( (m_betadmin <= beta) && (beta <= m_betadmax) ) {
   Mlip = m_pLUTCSP->Lookup(M0, (long)MACHNOS); // Mlip = Ms
}
else if(beta > m_betadmax) {
   m_pfluid->TMRayProps(M0, beta, m_betadmax, Mlip, Thetalip);
}

// calc the average Mach number at the cowl inlet.

Mav = 0.5*(Mlip + Mc);  // note that this might not be quite accurate
                        // since the Mach number distribution probably
                        // isn't linear across the cowl opening.

double Mns;
Mns = (M0-Mav)*(flowratio) + Mav;


// calculate the pressure recovery across the shock system

pt1_p0 = m_pLUTCSP->Lookup(M0, (long)PC_P1)*m_pfluid->pt_p(Mc);

// check if a normal shock will occur:
// note that if Mav is subsonic, then there will be no normal shock further down
// so we check on Mav, but use Mns in the calculation
if(Mav > 1.0) {
   pt2_pt1 = m_pfluid->NS_pt2_pt1(Mns);
   m_opmode =  eOM_SUPERCRITICAL;
}
```

University of Cape Town

```
   else {
      pt2_pt1 = 1.0;
      m_opmode = eOM_CRITICAL;
   }

   pid = pt2_pt1*pt1_p0/m_pfluid->pt_p(M0);

   return 1;

}

LookupTable * CConeIntake::ReplaceLUT(const char * lutfname)
{

   if(m_pLUTCSP) delete m_pLUTCSP;

   m_pLUTCSP = new LookupTable(lutfname, LookupTable::LINEAR, LookupTable::D1D);

   return m_pLUTCSP;

}

int CConeIntake::AutoDesign(double massflow, double mdotM, double rho, double Mmax, double BetaMin)
{

   if(!m_pLUTCSP->IsOK()) return 0;

   // get the shock angle for the max Mach number (degrees)
   if(m_pLUTCSP->OutRange(Mmax)) return 0;
   //double MmaxBeta = DTR(m_pLUTCSP->Lookup(Mmax, (long)SANGLE));
   double Mmin = m_pLUTCSP->RangeMin();
   double MminBeta = DTR(m_pLUTCSP->Lookup(Mmin, (long)SANGLE));

   // calculate the capture radius based on mass flow and a Mach number for that flow
   m_rc = sqrt( massflow/(rho*M_PI*mdotM*m_pfluid->SonicV()) );

   // calculate the distance from the cone apex to the intake lip,
   // assuming that the conical shock touches the intake lip at Mmax
// m_L = m_rc/tan(MmaxBeta);
   m_L = m_rc/tan(BetaMin);

   // calc the travel dist of the cone as half the dist
   // required to make the lowest attached Mach number touch the intake lip
   m_dx = 0.5*(m_L - m_rc/tan(MminBeta) );
// m_dx = (m_L - m_rc/tan(MminBeta) );
```


University of Cape Town

```
   // store the minimum and maximum values of betad
// m_betadmin = MmaxBeta;        // note: smallest beta at largest Mach
   m_betadmin = BetaMin;         // note: smallest beta at largest Mach
   m_betadmax = atan(m_rc/(m_L - m_dx)); // vice versa

   return 1;
}

void CConeIntake::GetIntakeGeometry(double &rc, double &L, double &dx)
{

   rc = m_rc;
   L = m_L;
   dx = m_dx;
}
```

University of Cape Town

Appendix B: DATCOM Model

Appendix B-1 Schematic Drawings

Figure B.1 and Figure B.2 present plan and elevation views of the hypothetical combat aircraft utilised in the Digital DATCOM model. These dimensions are based on schematic drawings presented in reference 3, as shown in Figure B.3

These views also indicate the cross-sectional area estimations which the DATCOM uses to estimate drag related to the 'area' rule. Hatched areas indicate portions of the wing and tail section which are considered to be hidden by the fuselage

All dimensions are indicated in metres.

University of Cape Town

Figure B.1: Plan view of the hypothetical fighter

Figure B.2: Elevation view of the hypothetical fighter

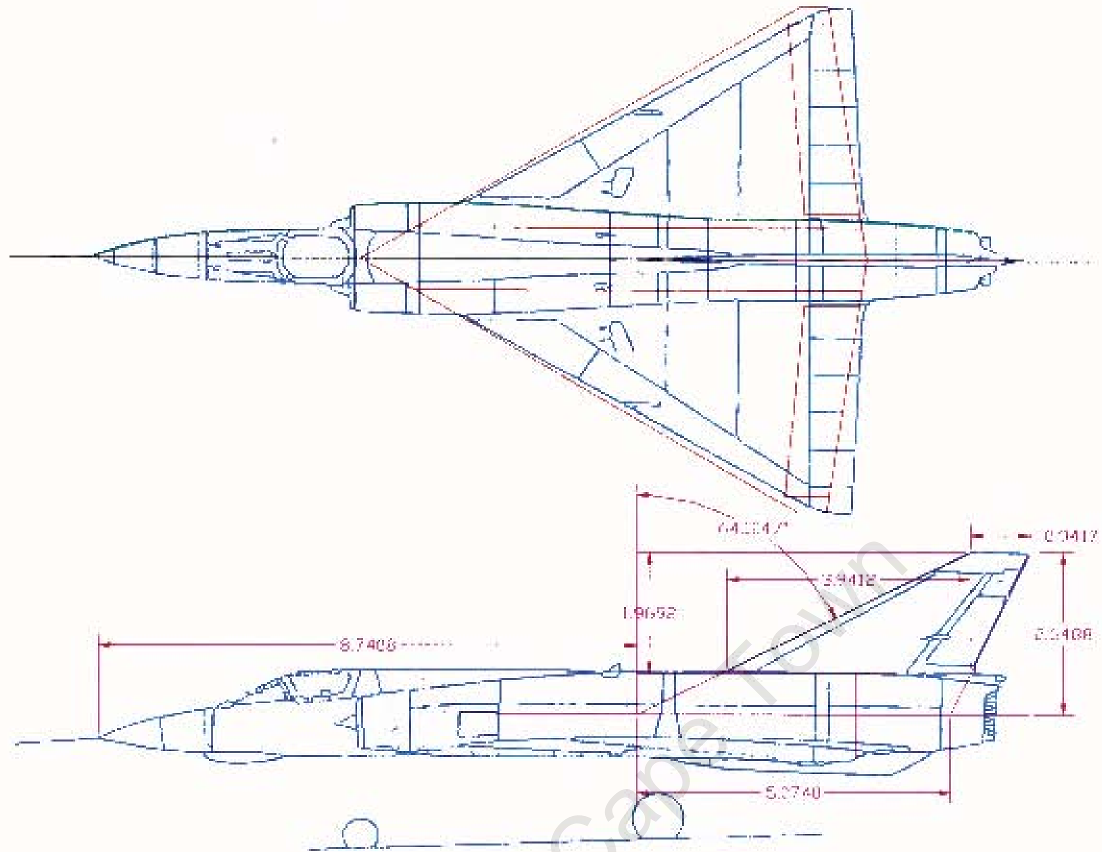


Figure B.3: Schematic outline superimposed on drawings from Jane's [3]

University of Cape Town

Appendix B-2 Sample Input File

```
DIM M
 $FLTCON NMACH=3.0,MACH(1)=.2,.4,.6,
  NALPHA=20.0,ALSCHD(1)=-6.0,-4.0,-2.0,0.0,2.0,4.0,6.0,8.0,10.0,
  12.0,14.0,16.0,21.0,24.0,27.0,30.0,37.0,42.0,47.0,52.0,
  LOOP=2.0,HYPERS=.FALSE.,
  NALT=11.0,ALT(1)=0.0,2000.0,4000.0,6000.0,8000.0,10000.0,
  12000.0,14000.0,16000.0,18000.0,20000.0$
 $SYNTHS XCG=6.5,ZCG=0.0$
 $BODY NX=15.0,
  X(1)=0.0,0.2573,1.0317,1.0615,3.2732,4.6573,4.2599,5.3762,6.8947,
  8.4163,10.0164,11.4436,13.0742,13.8758,14.4085,
  R(1)=0.0,0.1119,0.3366,0.3992,0.4512,0.4595,0.8465,0.8972,0.8557,
  0.7514,0.6473,0.6523,0.5797,0.5041,0.4276,
  S(1)=0.0,0.0393,0.2581,0.5008,1.0987,1.2808,1.6488,2.1235,2.1691,
  1.8842,1.5642,1.5519,1.1552,0.9565,0.7583,
  P(1)=0.0,0.7009,1.0000,2.3006,3.8443,4.0304,4.9566,5.4757,5.4958,
  5.0876,4.5043,4.5646,3.8161,3.4923,3.1315,
  BNOSE=2.0,BTAIL=1.0,BLN=4.259,BLA=10.148,ITYPE=3.0$
 $OPTINS ROUGFC=0.0000016$
 $SYNTHS XW=4.36,ZW=-0.54,ALIW=0.0$
 $WGPLNF CHRDTP=0.47,SSPNE=3.6,SSPN=4.11,CHRDR=3.02,SAVSI=60.0,
  CHSTAT=0.0,DHDADI=0.0,TWISTA=0.0,TYPE=1.0$
 $WGSCHR TYPEIN=1.0,NPTS=21.0,
  XCORD(1)=0.00,0.01,0.02,0.03,0.04,0.06,0.10,0.20,0.30,0.40,0.50,
  0.60,0.70,0.80,0.90,0.95,0.96,0.97,0.98,0.99,1.00,
  YUPPER(1)=0.0,0.001192,0.002360,0.003503,0.004622,0.005717,0.010625,
  0.019225,0.025214,0.028804,0.03,0.028804,0.025214,0.019225,0.010625,
  0.005717,0.004622,0.003503,0.002360,0.001192,0.0,
  YLOWER(1)=0.0,-0.0004,-0.000791,-0.001174,-0.001549,-0.001915,
  -0.003523,-0.006422,-0.008413,-0.009604,-0.01,-0.009604,-0.008413,
  -0.006422,-0.003523,-0.001915,-0.001549,-0.001174,-0.000791,-0.0004,0.0,
  KSHARP=5.3339
 $SYNTHS XV=8.74,VERTUP=.TRUE.$
NACA V 5-3-30.0-4.0-20.0
 $VTPLNF CHRDTP=0.941,SSPNE=1.989,SSPN=2.649,CHRDR=5.074,SAVSI=54.0,
  CHSTAT=0.0,TYPE=1.0$
 $VTSCHR KSHARP=5.3339
 $SYMFLP FTYPE=1.0,NDELTA=9.,DELTA(1)=-60.,-40.,-20.,-10.,0.,10.,
  20.,40.,60.,PHETE=.0522,PHETEP=.0523,SPANFI=.719,SPANFO=3.65,
  CHRDFI=0.39,CHRDFO=0.7,CB=0.05,TC=0.1,NTYPE=1.0,$
TRIM
CASEID SUBSONIC, BWTE, TRIM
```

University of Cape Town

Appendix B-3   Sample Output File

```
*******************************************************
                                                 *    USAF STABILITY AND CONTROL  DIGITAL DATCOM    *
                                                 *    PROGRAM REV. JAN 96   DIRECT INQUIRIES TO:    *
                                                 *   WRIGHT LABORATORY  (WL/FIGC)  ATTN: W. BLAKE   *
                                                 *         WRIGHT PATTERSON AFB, OHIO  45433        *
                                                 *    PHONE (513) 255-6764,    FAX (513) 255-4054   *
                                                 ****************************************************
1                                  CONERR - INPUT ERROR CHECKING
0 ERROR CODES   N* DENOTES THE NUMBER OF OCCURENCES OF EACH ERROR
0 A - UNKNOWN VARIABLE NAME
0 B - MISSING EQUAL SIGN FOLLOWING VARIABLE NAME
0 C - NON-ARRAY VARIABLE HAS AN ARRAY ELEMENT DESIGNATION - (N)
0 D - NON-ARRAY VARIABLE HAS MULTIPLE VALUES ASSIGNED
0 E - ASSIGNED VALUES EXCEED ARRAY DIMENSION
0 F - SYNTAX ERROR

0****************************  INPUT DATA CARDS  ****************************

  $FLTCON NMACH=1.0,MACH(1)=0.60,NALPHA=11.,ALSCHD(1)=-6.0,-4.0,-2.0,0.0,2.0,
   4.0,8.0,12.0,16.0,20.0,24.0,RNNUB(1)=4.28E6$
  $OPTINS SREF=8.85,CBARR=2.45,BLREF=4.28$
  $SYNTHS XCG=4.14,ZCG=-0.20$
  $BODY NX=10.0,
   X(1)=0.0,0.258,0.589,1.26,2.26,2.59,2.93,3.59,4.57,6.26,
   S(1)=0.0,0.080,0.160,0.323,0.751,0.883,0.939,1.032,1.032,1.032,
   P(1)=0.0,1.00,1.42,2.01,3.08,3.34,3.44,3.61,3.61,3.61,
   R(1)=0.0,.186,.286,.424,.533,.533,.533,.533,.533,.533$
  $BODY BNOSE=1.,BLN=2.59,BLA=3.67$
 CASEID APPROXIMATE AXISYMMETRIC BODY SOLUTION, EXAMPLE PROBLEM 1, CASE 1
 SAVE
 DUMP CASE
 NEXT CASE
  $BODY ZU(1)=-.595,-.476,-.372,-.138,0.200,.334,.343,.343,.343,.343,
         ZL(1)=-.595,-.715,-.754,-.805,-.868,-.868,-.868,-.868,-.868,-.868$
 CASEID ASYMMETRIC (CAMBERED) BODY SOLUTION, EXAMPLE PROBLEM 1, CASE 2
 SAVE
 NEXT CASE
  $FLTCON NMACH=3.0,MACH(1)=0.90,1.40,2.5,RNNUB(1)=6.4E6,9.96E6,17.9E6$
 SAVE
 CASEID ASYMMETRIC (CAMBERED) BODY SOLUTION, EXAMPLE PROBLEM 1, CASE 3
 NEXT CASE
  $FLTCON NMACH=1.0,MACH(1)=2.5,RNNUB(1)=17.9E6,HYPERS=.TRUE.$
```

University of Cape Town

```
 $BODY DS=0.0$
CASEID HYPERSONIC BODY SOLUTION, EXAMPLE PROBLEM 1, CASE 4
NEXT CASE
 $FLTCON NMACH=4.0,MACH(1)=0.60,0.90,1.40,2.50,LOOP=1.,NALT=4.0,
  ALT(1)=0.,2000.,40000.,90000.,HYPERS=.FALSE.,
  NALPHA=11.,ALSCHD(1)=-6.0,-4.0,-2.0,0.0,2.0,4.0,8.0,12.0,16.0,20.0,24.0$
 $OPTINS SREF=8.85,CBARR=2.46,BLREF=4.28$
 $SYNTHS XW=3.61,ZW=-.80,ALIW=2.0,XCG=4.14$
 $WGPLNF CHRDTP=0.64,SSPNE=1.59,SSPN=1.59,CHRDR=2.90,SAVSI=55.0,CHSTAT=0.0,
  TWISTA=0.0,SSPNDD=0.0,DHDADI=0.0,DHDADO=0.0,TYPE=1.0$
 $WGSCHR DELTAY=2.85,XOVC=0.40,CLI=0.127,ALPHAI=0.123,CLALPA(1)=.1335,
  TOVC=0.11,CLMAXL=1.55,
  CLMAX(1)=1.195,CMO=-.0262,LERI=.0134,CAMBER=.TRUE.,CLAMO=.105,TCEFF=0.055$
CASEID STRAIGHT TAPERED EXPOSED WING SOLUTION, EXAMPLE PROBLEM 2, CASE 1
SAVE
DUMP A
NEXT CASE
 $FLTCON NMACH=2.0,MACH(1)=0.60,2.5,LOOP=2.,NALT=2.,ALT(1)=0.,90000.$
 $SYNTHS XW=2.497,ZW=-.75$
 $WGPLNF SSPNOP=1.11,CHRDBP=2.24,CHRDR=4.01,SAVSI=75.1,SAVSO=55.0,TYPE=2.0$
 $WGSCHR TOVC=.10,LERI=0.011,LERO=.0158,TOVCO=0.10,XOVCO=0.40,CMOT=-.0262$
CASEID EXPOSED CRANKED WING SOLUTION, EXAMPLE PROBLEM 2, CASE 2
SAVE
NEXT CASE
 $FLTCON LOOP=3.$
 $WGPLNF TYPE=2.0$
CASEID EXPOSED DOUBLE DELTA WING SOLUTION, EXAMPLE PROBLEM 2, CASE 3
NEXT CASE
BUILD
 $FLTCON NMACH=2.0,MACH(1)=.60,.80,NALPHA=9.0,ALSCHD(1)=-2.0,0.0,2.0,
  4.0,8.0,12.0,16.0,20.0,24.0,RNNUB(1)=2.28E6,3.04E6$
 $FLTCON NMACH=3.0,MACH(1)=0.60,0.80,1.5,RNNUB(1)=4.28E6,6.4E6,
  9.96E6,$
 $OPTINS SREF=2.25,CBARR=0.822,BLREF=3.00$
 $SYNTHS XCG=2.60,ZCG=0.0,XW=1.70,ZW=0.0,ALIW=0.0,XH=3.93,
  ZH=0.0,ALIH=0.0,XV=3.34,VERTUP=.TRUE.$
 $BODY NX=10.0,BNOSE=2.0,BTAIL=1.0,BLN=1.46,BLA=1.97,
  X(1)=0.0,.175,.322,.530,.850,1.460,2.50,3.43,3.97,4.57,
  S(1)=0.0,.00547,.0220,.0491,.0872,.136,.136,.136,.0993,.0598,
  P(1)=0.0,.262,.523,.785,1.04,1.305,1.305,1.305,1.12,.856,
  R(1)=0.0,.0417,.0833,.125,.1665,.208,.208,.208,.178,.138$
 $WGPLNF CHRDTP=0.346,SSPNE=1.29,SSPN=1.50,CHRDR=1.16,SAVSI=45.0,CHSTAT=0.25,
  TWISTA=0.0,SSPNDD=0.0,DHDADI=0.0,DHDADO=0.0,TYPE=1.0$
 $WGSCHR TOVC=.060,DELTAY=1.30,XOVC=0.40,CLI=0.0,ALPHAI=0.0,CLALPA(1)=0.131,
  CLMAX(1)=.82,CMO=0.0,LERI=.0025,CLAMO=.105,YCM=0.0,
```

University of Cape Town

```
   SLOPE(1)=70.7,2.7,0.0,-2.5,-3.8,-4.0$
  $VTPLNF CHRDTP=.420,SSPNE=.63,SSPN=.849,CHRDR=1.02,SAVSI=28.1,
   CHSTAT=.25,TWISTA=0.0,TYPE=1.0$
  $VTSCHR TOVC=.09,XOVC=0.40,CLALPA(1)=2*0.141,DELTA=.0075$
  $WGSCHR CLMAXL=0.78$
  $HTPLNF CHRDTP=.253,SSPNE=.52,SSPN=.67,CHRDR=.42,SAVSI=45.0,CHSTAT=0.25,
   TWISTA=0.0,SSPNDD=0.0,DHDADI=0.0,DHDADO=0.0,TYPE=1.0$
  $HTSCHR TOVC=0.060,DELTAY=1.30,XOVC=0.40,CLI=0.0,ALPHAI=0.0,CLALPA(1)=.121,
   CLMAX(1)=0.82,CMO=0.0,LERI=.0025,CLAMO=.105,YCM=0.5
 CASEID CONFIGURATION BUILDUP, EXAMPLE PROBLEM 3, CASE 1
 SAVE
 NEXT CASE

// DATA TRIMMED HERE FOR BREVITY

NEXT CASE
  $FLTCON NMACH=1.,MACH(1)=10.,NALPHA=5.,ALSCHD(1)=0.,5.,10.,15.,20.,
   RNNUB(1)=1.06E5,HYPERS=.TRUE.$
  $OPTINS SREF=1.,CBARR=1.$
  $HYPEFF ALITD=150000.,XHL=8.,TWOTI=2.122,CF=2.5,HDELTA(1)=0.,2.,4.,6.,
   10.,12.,16.,20.,25.,30.,LAMNR=.TRUE.,HNDLTA=10.$
 CASEID FLAT PLATE WITH FLAP IN HYPERSONIC FLOW, EXAMPLE PROBLEM 13
 NEXT CASE
1           THE FOLLOWING IS A LIST OF ALL INPUT CARDS FOR THIS CASE.
0
  $FLTCON NMACH=1.0,MACH(1)=0.60,NALPHA=11.,ALSCHD(1)=-6.0,-4.0,-2.0,0.0,2.0,
   4.0,8.0,12.0,16.0,20.0,24.0,RNNUB(1)=4.28E6$
  $OPTINS SREF=8.85,CBARR=2.46,BLREF=4.28$
  $SYNTHS XCG=4.14,ZCG=-0.20$
  $BODY NX=10.0,
   X(1)=0.0,0.258,0.589,1.26,2.26,2.59,2.93,3.59,4.57,6.26,
   S(1)=0.0,0.080,0.160,0.322,0.751,0.883,0.939,1.032,1.032,1.032,
   P(1)=0.0,1.00,1.42,2.01,3.08,3.34,3.44,3.61,3.61,3.61,
   R(1)=0.0,.186,.286,.424,.533,.533,.533,.533,.533,.533$
  $BODY BNOSE=1.,BLN=2.59,BLA=3.67$
 CASEID APPROXIMATE AXISYMMETRIC BODY SOLUTION, EXAMPLE PROBLEM 1, CASE 1
 SAVE
 DUMP CASE
 NEXT CASE
0 INPUT DIMENSIONS ARE IN FT, SCALE FACTOR IS 1.0000


    BD(  1)= 6.26000E+00    BD(  2)= 3.59000E+00    BD(  3)= 1.03200E+00    BD(  4)= 1.03200E+00    BD(  5)= 6.26000E+00
    BD(  6)= 1.03200E+00    BD(  7)= 3.66530E+00    BD(  8)= 1.00000E-30    BD(  9)= 8.43444E-01    BD( 10)= 2.00339E-02
    BD( 11)= 0.00000E+00    BD( 12)= 1.00000E-30    BD( 13)= 1.00000E-30    BD( 14)= 1.00000E-30    BD( 15)= 1.00000E-30
```

University of Cape Town

```
    BD( 16)= 1.00000E-30     BD( 17)= 1.00000E-30     BD( 18)= 1.00000E-30     BD( 19)= 1.00000E-30     BD( 20)= 1.00000E-30
    BD( 21)= 1.00000E-30     BD( 22)= 1.00000E-30     BD( 23)= 1.00000E-30     BD( 24)= 1.00000E-30     BD( 25)= 1.00000E-30
    BD( 26)= 1.00000E-30     BD( 27)= 1.00000E-30     BD( 28)= 1.00000E-30     BD( 29)= 1.00000E-30     BD( 30)= 1.00000E-30
    BD( 31)= 1.00000E-30     BD( 32)= 1.00000E-30     BD( 33)= 4.14000E+00     BD( 34)= 1.00000E-30     BD( 35)= 1.00000E-30

// DATA TRIMMED HERE FOR BREVITY

  BODY(391)= 1.00000E-30  BODY(392)= 1.00000E-30  BODY(393)= 1.00000E-30  BODY(394)= 1.00000E-30  BODY(395)= 1.00000E-30
  BODY(396)= 1.00000E-30  BODY(397)= 1.00000E-30  BODY(398)= 1.00000E-30  BODY(399)= 1.00000E-30  BODY(400)= 1.00000E-30
1                             AUTOMATED STABILITY AND CONTROL METHODS PER APRIL 1976 VERSION OF DATCOM
                                           CHARACTERISTICS AT ANGLE OF ATTACK AND IN SIDESLIP
                                                 DATCOM BODY ALONE CONFIGURATION
                              APPROXIMATE AXISYMMETRIC BODY SOLUTION, EXAMPLE PROBLEM 1, CASE 1

 ----------------------- FLIGHT CONDITIONS ------------------------            -------------- REFERENCE DIMENSIONS ---------
  MACH    ALTITUDE   VELOCITY    PRESSURE    TEMPERATURE      REYNOLDS            REF.     REFERENCE LENGTH   MOMENT REF. CENTER
 NUMBER                                                        NUMBER             AREA       LONG.     LAT.     HORIZ      VERT
             FT        FT/SEC     LB/FT**2      DEG R           1/FT              FT**2       FT        FT        FT        FT
0 0.600                                                      4.2800E+06           8.950     2.460     4.280     4.140    -0.200
0                                                         -------------------DERIVATIVE (PER DEGREE)-------------------
0 ALPHA     CD       CL        CM        CN        CA       XCP        CLA          CMA          CYB          CNB          CLB
0
   -6.0    0.023   -0.021   -0.0207   -0.023    0.021    0.903    3.433E-03    3.443E-03   -3.433E-03   -1.979E-03    0.000E+00
   -4.0    0.022   -0.014   -0.0138   -0.015    0.021    0.905    3.433E-03    3.443E-03   -3.433E-03   -1.979E-03    0.000E+00
   -2.0    0.021   -0.007   -0.0069   -0.008    0.021    0.906    3.433E-03    3.443E-03   -3.433E-03   -1.979E-03    0.000E+00
    0.0    0.021    0.000    0.0000    0.000    0.021   ******    3.433E-03    3.443E-03   -3.433E-03   -1.979E-03    0.000E+00
    2.0    0.021    0.007    0.0069    0.008    0.021    0.906    3.433E-03    3.443E-03   -3.433E-03   -1.979E-03    0.000E+00
    4.0    0.022    0.014    0.0138    0.015    0.021    0.905    3.433E-03    3.443E-03   -3.433E-03   -1.979E-03    0.000E+00
    8.0    0.025    0.027    0.0275    0.031    0.021    0.899    3.433E-03    3.443E-03   -3.433E-03   -1.979E-03    0.000E+00
   12.0    0.029    0.041    0.0413    0.046    0.020    0.890    3.433E-03    3.443E-03   -3.433E-03   -1.979E-03    0.000E+00
   16.0    0.036    0.057    0.0551    0.063    0.019    0.878    3.433E-03    3.443E-03   -3.433E-03   -1.979E-03    0.000E+00
   20.0    0.044    0.069    0.0689    0.080    0.018    0.864    3.433E-03    3.443E-03   -3.433E-03   -1.979E-03    0.000E+00
   24.0    0.054    0.082    0.0826    0.097    0.016    0.849    3.433E-03    3.443E-03   -3.433E-03   -1.979E-03    0.000E+00
1          THE FOLLOWING IS A LIST OF ALL INPUT CARDS FOR THIS CASE
0
 $BODY ZU(1)=-.595,-.476,-.372,-.138,0.200,.334,.343,.343,.343,.343,
       ZL(1)=-.595,-.715,-.756,-.805,-.869,-.868,-.868,-.869,-.868,-.868$
 CASEID ASYMMETRIC (CAMBERED) BODY SOLUTION, EXAMPLE PROBLEM 1, CASE 2
 SAVE
 NEXT CASE
0 INPUT DIMENSIONS ARE IN FT, SCALE FACTOR IS 1.0000

1                             AUTOMATED STABILITY AND CONTROL METHODS PER APRIL 1976 VERSION OF DATCOM
                                           CHARACTERISTICS AT ANGLE OF ATTACK AND IN SIDESLIP
                                                 DATCOM BODY ALONE CONFIGURATION
```

University of Cape Town

```
                                 ASYMMETRIC (CAMBERED) BODY SOLUTION, EXAMPLE PROBLEM 1, CASE 2

 ---                       FLIGHT CONDITIONS          -----------------                   ---------------  REFERENCE DIMENSIONS  ------------
  MACH     ALTITUDE    VELOCITY     PRESSURE    TEMPERATURE      REYNOLDS                  REF.      REFERENCE LENGTH    MOMENT REF. CENTER
 NUMBER                                                           NUMBER                   AREA       LONG.      LAT.      HORIZ       VERT
              FT         FT/SEC      LB/FT**2       DEG R          1/FT                    FT**2       FT         FT         FT         FT
0 0.600                                                         4.2800E+06                 8.050      2.460      4.280      4.140      0.200
0                                                                                          DERIVATIVE (PER DEGREE) ------------------
0 ALPHA      CD          CL          CM          CN         CA         XCP        CLA         CMA          CYB          CNB           CLB
0
   -6.0     0.022     -0.012     -0.0239     -0.014      0.021      1.551     2.227E-03    4.111E-03    3.227E-03    2.363E-03    0.000E+00
   -4.0     0.021      0.007      0.0157      0.008      0.021      1.951     2.837E-03    4.092E-03   -2.837E-03   -2.352E-03    0.000E+00
   -2.0     0.021     -0.007     -0.0075     -0.002      0.021      4.656     2.859E-03    4.107E-03   -2.859E-03   -2.366E-03    0.000E+00
    0.0     0.021      0.005      0.0037      0.005      0.021      0.147     2.852E-03    4.102E-03    2.853E-03    2.357E-03    0.000E+00
    2.0     0.021      0.011      0.0089      0.011      0.021      0.790     2.818E-03    4.077E-03   -2.810E-03   -2.343E-03    0.000E+00
    4.0     0.022      0.016      0.0170      0.018      0.021      0.955     2.751E-03    4.039E-03   -2.751E-03   -2.315E-03    0.000E+00
    8.0     0.026      0.027      0.0229      0.030      0.022      1.093     2.537E-03    3.874E-03    2.537E-03    2.227E-03    0.000E+00
   12.0     0.031      0.036      0.0480      0.042      0.022      1.146     2.233E-03    3.651E-03   -2.233E-03   -2.092E-03    0.000E+00
   16.0     0.037      0.045      0.0621      0.053      0.023      1.171     3.100E-03    3.731E-03    3.100E-03    3.144E-03    0.000E+00
   20.0     0.043      0.061      0.0779      0.074      0.024      1.054     6.089E-03    4.440E-03   -6.089E-03    7.557E-03    0.000E+00
   24.0     0.070      0.093      0.0976      0.114      0.026      0.859     9.984E-03    5.440E-03   -9.984E-03   -3.177E-03    0.000E+00
1          THE FOLLOWING IS A LIST OF ALL INPUT CARDS FOR THIS CASE.
0
  $FLTCON NMACH=3.0,MACH(1)=0.90,1.40,2.5,RNNUB(1)=6.4E5,9.96E5,17.8E6$
 SAVE
 CASEID ASYMMETRIC (CAMBERED) BODY SOLUTION, EXAMPLE PROBLEM 1, CASE 3
 NEXT CASE
0 INPUT DIMENSIONS ARE IN FT, SCALE FACTOR IS 1.0000

-                              AUTOMATED STABILITY AND CONTROL METHODS PER APRIL 1976 VERSION OF DATCOM
                                            CHARACTERISTICS AT ANGLE OF ATTACK AND IN SIDESLIP
                                                    DATCOM BODY ALONE CONFIGURATION
                                 ASYMMETRIC (CAMBERED) BODY SOLUTION, EXAMPLE PROBLEM 1, CASE 3

                   -------  FLIGHT CONDITIONS  -------------------------                   ---------------  REFERENCE DIMENSIONS  ------------
  MACH     ALTITUDE    VELOCITY     PRESSURE    TEMPERATURE      REYNOLDS                  REF.      REFERENCE LENGTH    MOMENT REF. CENTER
 NUMBER                                                           NUMBER                   AREA       LONG.      LAT.      HORIZ       VERT
              FT         FT/SEC      LB/FT**2       DEG R          1/FT                    FT**2       FT         FT         FT         FT
0 0.900                                                         6.4000E+06                 8.050      2.460      4.280      4.140     -0.200
0                                                                                   -------DERIVATIVE (PER DEGREE)-------------------
0 ALPHA      CD          CL          CM          CN         CA         XCP        CLA         CMA          CYB          CNB           CLB
0
   -6.0     NDM        NDM         NDM         NDM        NDM        1.003     2.433E-03    2.443E-03    3.433E-03   -1.979E-03    0.000E+00
   -4.0     NDM                                                                                                                    0.000E+00
   -2.0     NDM                                                                                                                    0.000E+00
```

0.0	NDM	0.000E+00
2.0	NDM	0.000E+00
4.0	NDM	0.000E+00
8.0	NDM	0.000E+00
12.0	NDM	0.000E+00
16.0	NDM	0.000E+00
20.0	NDM	0.000E+00
24.0	NDM	0.000E+00

0*** NDM PRINTED WHEN NO DATCOM METHODS EXIST

University of Cape Town

Appendix B-4 DATCOM Post-Processing Source Code

B.4.1 Program: DATCOM Post

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fstream.h>
#include <strstrea.h>
#include <math.h>

int stripspace(char *outbuf, char *inbuf);


int main(void)
{

   double mach;
   double alt;
   double vel;
   double pres;
   double temp;
   double Rn;
   double Sref;
   double Cref;
   double span;
   double bmomref;
   double vmomref;

   int ncols;
   int currcol;
   int lineno=0;
   int pos;
   int cnt;

   int colwid;

   int coldelims[] = {2, 11, 18, 28, 37, 46, 55, 64, 76, 89, 102, 115, 127};
// int coldelims[] = {2, 13, 22, 32, 45, 50, 59, 72, 84, 94, 106, 118};

   char buf[512];
   char buf1[256];
```

University of Cape Town

```
char buf3[256];
char infname[512];
char outfname[512];
char config[513];
char caseid[512];

char alphabuf[40];
char clbuf[40];
char cdbuf[40];
double cl, clold;
int cdlen;

int rowstart;


ofstream OutFile;

int freadytox;

cout << "Enter the filename: ";
cin >> infname;

ifstream InFile(infname, ios::in, filebuf::sh_read);

buf[0] = ' ';       // set first char to space

while(!InFile.eof()) {

   freadytox = 0;

   while(!freadytox) {

      // extract all data until we find a line starting with '1'
      while(buf[0] != '1') {
         InFile.getline(buf, sizeof(buf));
         lineno++;
         if(InFile.eof()) break;
      }

      if(InFile.eof()) break;

      // extract the next line
      InFile.getline(buf, sizeof(buf));
      lineno++;
```


University of Cape Town

```
            // check if it says "CHARACTERISTICS AT ANGLE OF ATTACK AND IN SIDESLIP"
            if( strstr(buf, "CHARACTERISTICS AT ANGLE OF ATTACK AND IN SIDESLIP") )
//          if( strstr(buf, "CHARACTERISTICS OF HIGH LIFT AND CONTROL DEVICES") )
               freadytox = 1;

            buf[0] = ' ';      // set first char to space

         }

         if(InFile.eof()) continue;

         // get job description data
         InFile.eatwhite();
         InFile.getline(buf, sizeof(buf));
         lineno++;
//       stripspace(config, buf);
         strcpy(config, buf);

/*       InFile.eatwhite();
         InFile.getline(buf, sizeof(buf));
         lineno++;
//       stripspace(caseid, buf);
         strcpy(caseid, buf);
*/
         cout << lineno << " " << buf << endl;
         cout << lineno 1 << " " << config << endl;
//       cout << lineno << " " << caseid << endl;

         // extract rubbish till we encounter data, marked with a zero.
         buf[0] = ' ';
         while(buf[0] != '0') {
            InFile.getline(buf, sizeof(buf));
            lineno++;
         }

/*
         // extract the zero
         InFile >> pos;

         // now extract some useful data
         InFile >> mach >> alt >> vel >> pres >> temp >> Rn >> Sref;
         InFile >> Lref >> span >> xmomref >> vmomref;
*/
         mach = cos(1.3456); // just to let the compiler know we need fp support
```


University of Cape Town

```
//    sscanf(buf1, "%lf", &mach);

      sscanf(buf, "%d %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf", &pos,
         &mach, &alt, &vel, &pres, &temp, &Rn, &Sref, &Lref, &span, &hmomref, &vmomref);

      // extract another line
      InFile.getline(buf, sizeof(buf));
      lineno++;


      // get number of columns
      ncols = sizeof(coldelims)/sizeof(coldelims[0]);

      // create the output file name

      sprintf(outfname, "%s-M%.2lf-%.1lfm.lut.dat", infname, mach, alt);

      // open the output file
      OutFile.open(outfname, ios::app|ios::out, filebuf::openprot);
/*
      // output the state parameters read in earlier
      OutFile << "Input File:\t" << infname << endl;
      OutFile << "This File:\t" << outfname << "\n" << endl;
*///     OutFile << config << "\n" /*<< caseid << "\n" */<< endl;
/*
      OutFile << "Mach Number:\t" << mach << endl;
      OutFile << "Altitude:\t" << alt << endl;
      OutFile << "Velocity:\t" << vel << endl;
      OutFile << "Pressure:\t" << pres << endl;
      OutFile << "Temperature:\t" << temp << endl;
      OutFile << "Reynolds No.:\t" << Rn << endl;
      OutFile << "Ref. Area:\t" << Sref << endl;
      OutFile << "Lat. Ref. Length:\t" << Lref << endl;
      OutFile << "Span:\t" << span << endl;
      OutFile << "Horizontal Moment Ref.:\t" << hmomref << endl;
      OutFile << "Vertical Moment Ref.:\t" << vmomref << "\n" << endl;
*/
      _strset(buf1, ' ');
      _strset(buf2, ' ');

      // get the line with the headers
      InFile.getline(buf, sizeof(buf));
      lineno++;

//    OutFile << lineno << "\t";
```


University of Cape Town

```
/*
      // extract headers
      for(currcol=0;currcol<ncols-1;currcol++) {
         colwid = coldelims[currcol+1]-coldelims[currcol];
         strncpy(buf1, buf+coldelims[currcol], colwid);
         buf1[colwid] = '\0';  // append terminating null
         stripspace(buf2, buf1);
//       cout << buf2 << "\t";
         OutFile << buf2 << "\t";
      }
//    cout << endl;
      OutFile << endl;
*/
      // strip another line
      InFile.getline(buf, sizeof(buf));
      lineno++;

      // now we want to start processing the data
      buf[0] = ' ';
      InFile.getline(buf, sizeof(buf));
      lineno++;

      rowstart = lineno;  // record the starting line

      // create a temporary output stream
      ostrstream TempOut;

      // set old cl to very negative number
      clold = -99000.0;

      while(buf[0] != '1') {

         currcol=0;
         // alpha
         colwid = coldelims[currcol+1]-coldelims[currcol];
         strncpy(buf1, buf+coldelims[currcol], colwid);
         buf1[colwid] = '\0';  // append terminating null
         stripspace(alphabuf, buf1);
         // cd
         currcol++;
         colwid = coldelims[currcol+1]-coldelims[currcol];
         strncpy(buf1, buf+coldelims[currcol], colwid);
         buf1[colwid] = '\0';  // append terminating null
         cdlen = stripspace(cdbuf, buf1);
         if(cdlen==0) sprintf(cdbuf, "-999.0");  // there was no number
```


University of Cape Town

```
            // cl
            currcol++;
            colwid = coldelims[currcol-1]-coldelims[currcol];
            strncpy(buf1, buf+coldelims[currcol], colwid);
            buf1[colwid] = '\0';  // append terminating null
            stripspace(clbuf, buf1);
            sscanf(clbuf, "%lf", &cl);
            if(cl<clold) {  // check if we've reached the peak in the lift curve
               buf[0] = '1';
               continue;
            }
            clold = cl;

            TempOut << clbuf << "\t" << cdbuf << "\t" << alphabuf << endl;

            // read the next line
            InFile.getline(buf, sizeof(buf));
            lineno++;

         }

         char *tempstream;
         int pcount;
         pcount = TempOut.pcount();
         tempstream = TempOut.str();
         // append null after the bit we've stored
         tempstream[pcount-1] = '\0';

         OutFile << lineno - rowstart << endl;
         OutFile << "3\n";
         OutFile << tempstream;

         free(tempstream);

         OutFile.close();
      }

   return lineno;

}

int stripspace(char *outbuf, char *inbuf)
{

   char *revstr;
```

University of Cape Town

```
    char *buf;
    int len;
    int strt, endd;

    // duplicate the string
    buf = _strdup(inbuf);

    // get total length
    len = strlen(inbuf);

    // reverse the string so that we can strip trailing spaces
    revstr = _strrev(buf);

    // find first non-space
    endd = len - strspn(revstr, " ");

    strt = strspn(inbuf, " ");

    // clear the output buffer
    _strset(outbuf, 0);

    // this will happen if string is all blank
    if(endd < strt) endd = strt = 0;
    else strncpy(outbuf, inbuf+strt, endd-strt);

    // append null character to string
    outbuf[endd-strt] = '\0';

    // free memory allocated by strdup()
    free(buf);

    return endd-strt;
}
```

B.4.2 Program: CD_CL

```
#include <iostream.h>
#include <stdlib.h>
#include <string.h>
#include "../LookupTable/LookupTB.hpp"

bool GetMachNumFromFName(const char *lpszName, double &Mach);
```

```
int main(void)
{

   char infilelist[512];
   char infile[512];
   char outfile[512];

   int cnt, tcnt;
   int rows, cols;
   int row, col;

   double Cl[100], Cd[100], alpha[100], Cd0;;
   double lCd, lCl;
// double Clmax;
   double factor;

   double *Clmax, *Cd_Clmax, *Alpha_Clmax, *MList;

   int clp, nn;

   cout << "Enter An input file: ";
   cin >> infilelist;

   ifstream List(infilelist);

   List >> tcnt;

   // allocate memory for the clmax stuff
   Clmax       = new double[tcnt];
   Cd_Clmax    = new double[tcnt];
   Alpha_Clmax = new double[tcnt];
   MList       = new double[tcnt];

   bool bMach;
   double Mtemp;

   for(cnt=0;cnt<tcnt;cnt++) {

      List >> infile;

      bMach = GetMachNumFromName(infile, Mtemp);
      if( bMach ) MList[cnt] = Mtemp;

      clp = 0;
      nn = 0;
```

University of Cape Town

```
// this block will ensure that the lookup table is
// created and then destroyed when we no longer need it.
{
   // create a lookup table
   LookupTable CLCD(infile, LookupTable::LINEAR, LookupTable::D1D);

   Cd0 = CLCD.Lookup(0.0, (long)1);
}



ifstream Data(infile);
strcpy(outfile, infile);
strcpy(outfile + strlen(infile), ".CDCL.lut.dat");

ofstream Out(outfile);

Data >> rows;
Data >> cols;

Clmax[cnt] = 0.0;

for(row=0;row<rows;row++) {
   Data >> Cl[row];
   Data >> Cd[row];
   Data >> alpha[row];

   rn++;

   if(Cd[row] < 0.0) {

      // classical equation:
      //Cd = Cd0 + Cl^2/(pi*AR*e)

      factor = (lCd - Cd0)/(lCl*lCl);

      Cd[row] = Cd0 + Cl[row]*Cl[row]*factor;
   }
   else {

      lCd = Cd[row];
      lCl = Cl[row];
   }
```

University of Cape Town

```
            // extract maximum cl
            if(Cl[row] > Clmax[cnt]) {
               Clmax[cnt] = Cl[row];
               Cd_Clmax[cnt] = Cd[row];
               Alpha_Clmax[cnt] = alpha[row];
            }

            if(Cl[row] >= 0.0) clp++; // only write out for + Cl
         }

         // dump the CD CL Alpha lut
         Out << clp << endl;
         Out << cols << endl;

         for(row=0;row<rows;row++)
            if(Cl[row] >= 0)
               Out << Cd[row] << "\t" << Cl[row] << "\t" << alpha[row] << endl;

         Out.close();



         Data.close();
//       Out.close();

      }

      // dump the clmax and cd at cl max data
      strcpy(outfile, infile);
      strcpy(outfile + strlen(infile), ".ClmaxCD.lut.dat");

      ofstream ClMaxCd(outfile);

      ClMaxCd << tcnt << "\n" << 4 << endl;

      for(cnt=0;cnt<tcnt;cnt++)
         ClMaxCd << MList[cnt] << "\t" << Clmax[cnt] << "\t" << Cd_Clmax[cnt] << "\t" << Alpha_Clmax[cnt] << endl;

      ClMaxCd.close();

      return 0;

}
```


University of Cape Town

```
bool GetMachNumFromFName(const char *lpszName, double &Mach)
{

   char *start;
   char *endd;

   start = strstr(lpszName, "-M");

   if(!start) return false;

   // move along by 2 to pass over the -M
   start += 2;

   Mach = strtod(start, &endd);

   return true;

}
```

B.4.3 Program: CL_CD_Full

```
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include "../LookupTable/LookupTB.hpp"

int main(void)
{

   char infilelist[512];
   char infile[512];
   char outfile[512];

   int cnt, tcnt;
   int rows, cols;
   int row, col;

   double Cl[100], Cd[100], alpha[100], Cd0;;
   double lCd, lCl;
   double factor;

   double Clt, Cdt;
```


University of Cape Town

```
   int clp, cn;

   char namefmt[] = "D:/DIGDAT/MIII/CLCD/MIII M%.2lf %.1fm.lut.dat";

// cout << "Enter an input file: ";
// cin >> infilelist;

// ifstream List(infilelist);

// List >> cent;

   double M_List[] = { 0.2, 0.4, 0.6, 0.7, 0.8, 0.9, 1.1, 1.2, 1.3, 1.4, 1.6, 1.8, 2.0, 2.2 };
   double A_List[] = {  0.0,      2000.0,  4000.0,  6000.0,
                        8000.0,  10000.0, 12000.0, 14000.0,
                        16000.0, 18000.0, 20000.0 };
   int nM = 14;
   int nA = 11;

   int Mcnt, Acnt;
   double Alt, M;
   double alpha_06, alpha_14, alphaclip;

   // go by altitude
   for(Acnt=0;Acnt<nA;Acnt++) {

      Alt = A_List[Acnt];

      // get the stuff at 0.6 and 1.4 Mach
      {
         char buf[1024];
         int r, c;
         int n;
         float at, bt, ct;
         sprintf(buf, namefmt, 0.6, Alt);

         ifstream stuff(buf);
         stuff >> r >> c;

         stuff.getline(buf, 1024); // just get the rest of the line on which cols is.

         for(n=0;n<r;n++) stuff.getline(buf, 1024);

         sscanf(buf, "%f %f %f", &at, &bt, &ct);
         alpha_06 = ct;
```

University of Cape Town

```
    stuff.close();

}

{
    char buf[1024];
    int r, c;
    int n;
    float at, bt, ct;
    sprintf(buf, namefmt, 1.4, Alt);

    ifstream stuff(buf);
    stuff >> r >> c;

    stuff.getline(buf, 1024); // just get the rest of the line on which data is.

    for(n=0;n<r;n++) stuff.getline(buf, 1024);

    sscanf(buf, "%f %f %f", &at, &bt, &ct);
    alpha_14 = ct;

    stuff.close();
}

// now go by Mach number
for(Mcnt=0;Mcnt<nM;Mcnt++) {

    M = M_List[Mcnt];

    // generate the filename
    sprintf(infile, namefmt, M, Alt);


    // this block will ensure that the lookup table is
    // created and then destroyed when we no longer need it.
    {
        // create a lookup table
        LookupTable CLCD(infile, LookupTable::LINEAR, LookupTable::D1D);

        Cd0 = CLCD.Lookup(0.0, (long)1);
    }
```


University of Cape Town

```
ifstream Data(infile);
strcpy(outfile, infile);
strcpy(outfile + strlen(infile), ".CLCD.lut.dat");

ofstream Out(outfile);

// get the number of rows and columns
Data >> rows;
Data >> cols;

// if we're 0.6 < M < 1.4 then we want to clip alpha
if( (0.6 < M) && (M<1.4) )
   alphaclip = (M-0.6)*(alpha_06 - alpha_14)/(0.6-1.4) + alpha_06;

clp = 0;
nn = 0;

for(row=0;row<rows;row++) {
   Data >> Cl[row];
   Data >> Cd[row];
   Data >> alpha[row];

   if( (0.6 < M) && (M<1.4) ) {
      if(alpha[row] < alphaclip) nn++;
   }
   else nn++;

   if(Cd[row] < 0.0) {

      // classical equation:
      //Cd = Cd0 + Cl^2/(pi*AR*e)

      factor = (lCd - Cd0)/(lCl*lCl);

      Cd[row] = Cd0 + Cl[row]*Cl[row]*factor;
   }
   else {

      lCd = Cd[row];
      lCl = Cl[row];
   }

//       if(Cl[row] >= 0.0) clp++; // only write out for + Cl
   }
```

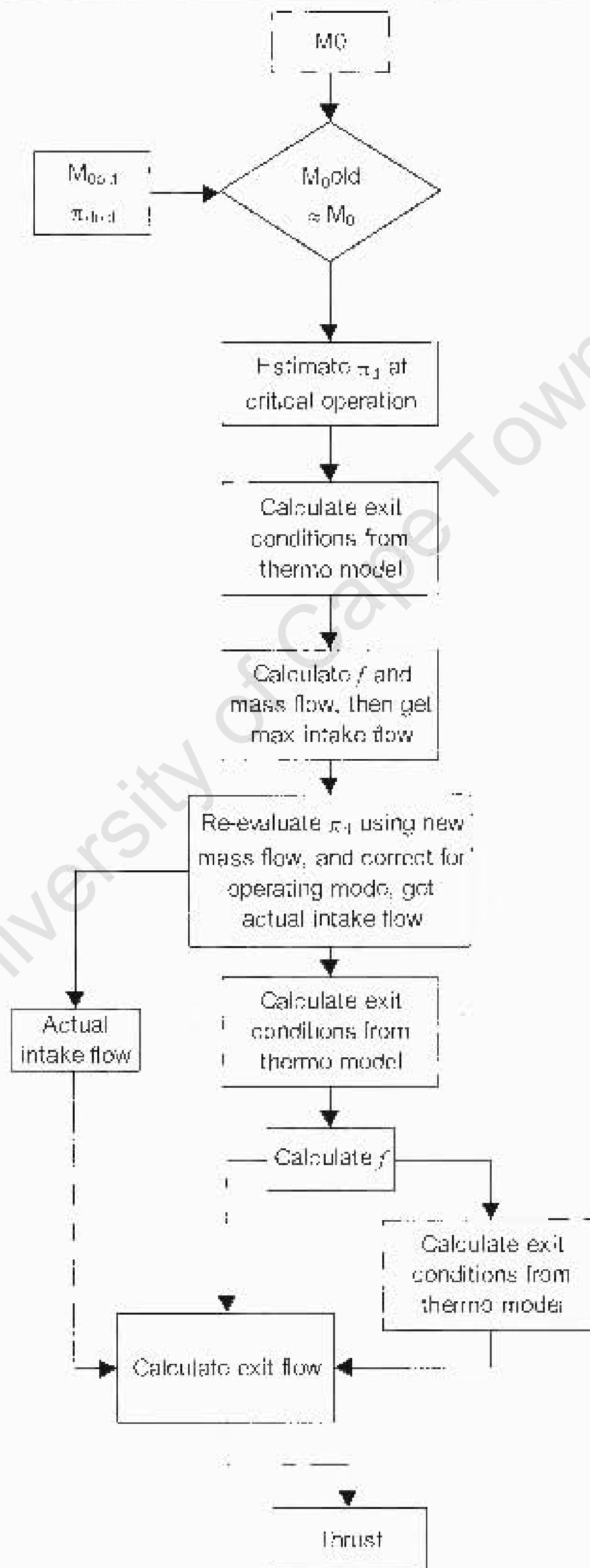

University of Cape Town

```
        if( (0.6 < M) && (M<1.4) ) {
           nn++;
           Clt = (alphaclip-alpha[nn-2])*(Cl[nn-1]-Cl[nn-2])/(alpha[nn-1]-alpha[nn-2]) + Cl[nn-2];
           Cdt = (alphaclip-alpha[nn-2])*(Cd[nn-1]-Cd[nn-2])/(alpha[nn-1]-alpha[nn-2]) + Cd[nn-2];

           Cl[nn-1] = Clt;
           Cd[nn-1] = Cdt;
           alpha[nn 1] = alphaclip;
        }

        Out << nn << endl;
        Out << cols << endl;

        for(row=0;row<nn;row++)
//      if(Cl[row] >= 0)
           Out << Cl[row] << "\t" << Cd[row] << "\t" << alpha[row] << endl;


        Data.close();
        Out.close();
     }
  }

  return 0;

}
```


University of Cape Town

Appendix C: Propulsion Model

Appendix C-1 Flow Chart for estimating Thrust Output



University of Cape Town

Appendix C-2 Propulsion Model: Class TurboJet1D

C.2.1 Header File: 1DPropModel.hpp

```
/*******************************************************************************
  FILENAME:  1DPropModel.hpp                                                   |
  NAME:      Class 1DPropModel                                                 |
  DESCRIPTION: header file defining class 1DPropModel - a one dimensional      |
               gas turbine propulsion model.  Model accounts for component     |
               inefficiencies and allows for a bypass stream.                  |
  AUTHOR:    Prakash Parbhoo                                                   |
  DATE:      23 April 1997                                                     |
  REVISION HISTORY:                                                            |
*******************************************************************************/

#include <math.h>
#include <iostream.h>
#include "..\CONEINTAKE\Coneintake.h"  // Added by ClassView
#include "..\IdealGas\IdealGasStream.h"

// define a square macro
#if !defined( SQ )
#define SQ(x) ((x)*(x))
#endif

const double Rair = 287;

class Turbojet1D {

   protected:
      double m_machmax;

   private:

      /*** Component Parameters ***/

      // Static Pressure Ratios
      double pi_f;    // bypass fan pressure ratio
      double pi_c;    // compressor pressure ratio
      double pi_b;    // burner pressure ratio
      double pi_a;    // afterburner pressure ratio
```

University of Cape Town

```
// Efficiencies
double eta_f;  // fan
double eta_c;  // compressor
double eta_b;  // burner
double eta_t;  // turbine
double eta_a;  // afterburner
double eta_n;  // nozzle
double eta_nb; // bypass nozzle

// Maximum Temperatures
double Tt4max; // max turbine inlet temp
double Tt6max; // max afterburner temp

// Maximum Nozzle Areas
double m_A7max;    // max primary nozzle area
double m_A9max;    // max bypass nozzle area

// Minimum Nozzle Areas
double m_A7min;    // min primary nozzle area
double m_A9min;    // min bypass nozzle area


// Fuel
double QR;  // lower heating value of jet fuel

// Nozzle Types
enum NozzleEnum nozzleprimary;
enum NozzleEnum nozzlebypass;

// diffuser pressure loss function
//      double (Turbojet1D::*P) Diffuser(double, double, double);


/*** Operational Parameters ***/

double beta;    // bypass ratio for bleed air



/*** Gas Property Utility Functions ***/

//const double Rair = 287.0;
// Cengel & Boles   Table A2   note: Only really valid for T = 273K to 1800K
//      double Cp(double temp) { return (15000.0/287.1)*(28.11 + 0.001967*temp + (0.4802e-5)*(temp*temp) - (1.966e-9)*(temp*temp*temp)); }
double Cp(double t);
double gamma(double temp, double R) { double Cptemp = Cp(temp);  return Cptemp/(Cptemp - R); }
```

University of Cape Town

```
    double Tt_T(double M, double gama) { return ( 1.0 + 0.5*(gama-1.0)*SQ(M) ); }  // C&B 16-18 pg 719
    double pt_p(double M, double gama) { return pow( 1.0 + 0.5*(gama-1.0)*SQ(M), gama/(gama-1.0) ); }  // C&B 16-19 pg 719
    double deltas(double T2_T1, double p2_p1, double Cpes, double R) { return ( Cpes*log(T2_T1)-R*log(p2_p1) ); }

    double isobaric(double T1, double s1, double s2, double Cpav) { return T1 * exp((s2 - s1) / Cpav); }



    // standard AIA diffuser pressure recovery function
    double Pi_d_AIA(double M, double T, double p) { return (M<=1.0)?0.95:(.95 - 0.1*pow((M-1.0), 1.5)); }
    double Pi_diffuser(double M, double T, double p) { return (M<=1.0)?m_Pid_const:(m_Pid_const - 0.1*pow((M-1.0), 1.5)); }

  public:
    virtual CConeIntake * CreateIntake(double ConeAngle, CIdealGasStream *Fluid, const char * lutfname);
    Turbojet1D();

//  Turbojet1D(void) { Pi_diffuser = Pi_d_AIA; }

    // functions to set component parameters
    void Pi_f(double val)   {pi_f = val;}
    void Pi_c(double val)   {pi_c = val;}
    void Pi_b(double val)   {pi_b = val;}
    void Pi_a(double val)   {pi_a = val;}
//  void Pi_d_func(double (Turbojet1D::*Pi_func)(double, double, double)) { Pi_diffuser = Pi_func; }

    void Eta_f(double val)  { eta_f = val;}
    void Eta_c(double val)  { eta_c = val;}
    void Eta_b(double val)  { eta_b = val;}
    void Eta_t(double val)  { eta_t = val;}
    void Eta_a(double val)  { eta_a = val;}
    void Eta_n(double val)  { eta_n = val;}
    void Eta_nb(double val) { eta_nb = val;}

    void BypassRatio(double val)  { beta = val;}
    void A7_max(double val) { m_A7max = val; }     // exit areas
    void A9_max(double val) { m_A9max = val; }
    void A7_min(double val) { m_A7min = val; }
    void A9_min(double val) { m_A9min = val; }

    void Tt_4(double val);
    void Tt_6(double val);

    void MaxMassFlow(double val) { m_mdotmax = val; }

    void FuelQ(double val)  { Q4 = val;};
```

University of Cape Town

```
        void MainNozzleType( enum NozzleEnum noztype ) { nozzleprimary = noztype; }
        void BPNozzleType( enum NozzleEnum noztype ) { nozzlebypass = noztype; }

        enum NozzleEnum { CONDIVERGENT, CONVERGENT };
        enum ABEnum { OFF, ON };

        int DrawTs(double M0, double T0, double p0, ABEnum ABOnOff, int plotisobars, ofstream& OutputFile);
        double TSFC(double M0, double T0, double p0, ABEnum ABOnOff, double& F_ma, double& f);
        int Thrust(double M0, IdealGas *atm, double p0, double aca, double throttle, ABEnum ABOnOff, double &F, double &mdf, double &A7_ma);

        // state variables - use for diagnosis
        double o_A7;
        double o_A9;
        double o_A7_ma;
        double o_mdotaengmax;
        double o_mdotmax;
        double o_mdotaengmin;
        double o_mdotmin;
        double o_mdotactual;
        double o_pi_d;
        double o_TSFC;

        double m_Pid_const;

    protected:
        double pi_d_last;
        double M0_last;
        CConeIntake *pCCIdiff;

        double CycleThermodyamics(double M0, double T0, double p0, ABEnum ABOnOff, double pi_d, CIdealGasStream& ex7, CIdealGasStream& ex9, double&
A7_ma, double& A9_ma, double& V_ma, double& f);
};
```

C.2.2 Implementation File: 1DPropModel.cpp

```
/**********************************************************************************\
| FILENAME: 1DPropModel.cpp
| NAME:      Class TurboJet1D
| DESCRIPTION: class TurboJet1D - a one dimensional
|              gas turbine propulsion model.  Model accounts for component   |
|              inefficiencies and allows for a bypass stream.                 |
| AUTHOR:    Prakash Parbhoo                                                  |
| DATE:      22 April 1997                                                    |
| REVISION HISTORY:                                                           |
```

University of Cape Town

```
          01/05/97: Added dummy Cp function to return constant Cp         |
/*************************************************************************/

#include "1DPropModel.hpp"


// Cp - calculates specific heat for air at a given temperature
double Turbojet1D::Cp(double t)
{

   // This is based on data presented in Zucrow & Hoffman Vol. 1,
   // Section 1-16 (a), pg 57.
   if(t<1000.0) return (3.65359 - (1.33736e-3)*t + (3.29421e-6)*SQ(t) - (1.91142e-9)*t*SQ(t) + (0.275462e-12)*SQ(SQ(t)) )*Rair;
   return (3.04473 + (1.33805e-3)*t - (0.488256e-6)*SQ(t) + (0.0855475e-9)*t*SQ(t) - (0.00570132e-12)*SQ(SQ(t)) )*Rair;

}

/*
// This function used to see that everything ties up if a constant specific heat is used.
double Turbojet1D::Cp(double t)
{
   return 1004;
}
*/

// we may later want to add to this function, hence out of line inline definition
void Turbojet1D::Tt_4(double val)
{
   Tt4max = val;
}
void Turbojet1D::Tt_5(double val)
{
   Tt5max = val;
}


int Turbojet1D::DrawTs(double M0, double T0, double p0, ABEnum ABOnOff, int plotisobars, ofstream& OutputFile)
{

   // specific heat ratios
   double gammac; // compressor
   double gammat;
   double gamman;

   // specific cp's
   double Cpc;     // compressor
```

University of Cape Town

```
double Cpb;    // burner
double Cpt;    // turbine
double Cpa;    // afterburner
double Cpn;    // nozzle exit

double theta0;
double delta0;

// stagnation temperature ratios
double tau_c;  // compressor
double tau_b;  // burner
double tau_f;  // fan
double tau_t;  // turbine

// temperatures
double Tt3;               // compressor exit
double Tt4 = Tt4max;      // turbine entry
double Tt5;               // turbine exit
double Tt6;               // afterburner exit
double Tt7;               // afterburner exit
double T7;
double Tt8;               // bypass fan exit
double T9;                // bypass nozzle exit

// other
double fb;  // fuel/air ratio for primary burner
double fa;  // fuel/air ratio for afterburner

double pt6;
double p7;
double p9;
double pi_d;
double pi_t;
double pt8;

double u7sq;   // exit velocity of primary stream
double u9sq;   // exit velocity of bypass stream
double M7;     // exit mach number
double M9;


// determine stagnation conditions of intake air
theta0 = Tt_T(M0, gamma(T0, Rair));        // PP 28/4/97 Eq. 11
delta0 = pt_p(M0, gamma(T0, Rair));        // PP 28/4/97 Eq. 12

// determine avg. air properties for compressor
```


University of Cape Town

```
gammac = gamma(0.5*T0*(1.0+theta0), Rair);
Cpc = Cp(0.5*T0*(1.0+theta0));

// calculate temperature ratio across the compressor
tau_c = ( pow(pi_c, (gammac-1.0)/gammac) - 1.0)/eta_c + 1.0;      // PP 28/4/97 Eq. 13
Tt3 = tau_c*theta0*T0;                                             // PP 28/4/97 Eq. 14

// determine average burner specific heat
Cpb = Cp(0.5*(Tt3+Tt4));

// determine burner fuel/air ratio
fb = Cpb*(Tt4 - Tt3)/(QR*eta_b - Cpb*Tt4);    // PP 28/4/97 Eq. 15
tau_b = Tt4/Tt3;

// calculate temperature ratio across the fan
tau_f = ( pow(pi_f, (gammac-1.0)/gammac) - 1.0)/eta_f + 1.0;      // PP 28/4/97 Eq. 17
Tt8 = tau_f*theta0*T0;

// determine average turbine specific heat, assume drop in temp roughly equal to compressor temp rise
Cpt = Cp(Tt4 - 0.5*(Tt3-theta0*T0));       // Tt2 = theta0*T0
gammat = gamma(Tt4 - 0.5*(Tt3-theta0*T0), Rair);

// determine temperature ratio across the turbine
tau_t = 1.0 - Cpc*(beta*(tau_f-1.0)+(tau_c-1.0))/(Cpt*tau_b*tau_c*(1.0+fb)); // PP 28/4/97 Eq. 18
Tt5 = tau_t*Tt4;

// check if afterburner is on or off and apply appropriate condition
if(ABOnOff == ON) {
   Tt7 = Tt6 = Tt6max;
   Cpa = Cp(0.5*(Tt5+Tt6));
   fa = Cpa*(1+fb)*(Tt6-Tt5)/((QR*eta_a-Cpa*Tt6));     // PP 28/4/97 Eq. 20
}
else {   // afterburner is off
   Tt6 = Tt7 = tau_t*Tt4;
   Cpa = Cp(Tt6);
   fa = 0.0;           // no fuel is used
}

// we could now calculate f
// f = fa + fb;

// determine the pressure ratio across the turbine
pi_t = pow( (1.0-(1.0-tau_t)/eta_t), gammat/(gammat-1.0) );// PP 28/4/97 Eq. 22

// determine the stagnation pressure ratio across the diffuser
pi_d = Pi_diffuser(M0, T0, p0);
```


University of Cape Town

```
pt6 = p0*delta0*pi_d*pi_c*pi_b*pi_t*pi_n;                    // PP 28/4/97 Eq. 23

// the two different exit conditions must now be considered.
/*
if the nozzle is convergent, we need to check if it is choked.  We thus consider
it as if it were a convergent divergent nozzle and check the exit Mach number.
If the mach number exceeds 1, we then apply the choked solution, otherwise we use the
fully expanded solution.
*/

/**** evaluate the fully expanded case: ****/

// estimate Cpn by first estimating T7, using turbine constants

gamman = gammat;
Cpn = Cpt;
p7 = p0;
u7sq = (2*eta_n*Cpn*Tt6*(1.0-pow(p7/pt6, (gamman-1.0)/gamman)));     // PP 28/4/97 Eq. 24
T7 = Tt6 - u7sq/(2*Cpn);                                               // PP 28/4/97 Eq. 25

// now reestimate Cpn
gamman = gamma(0.5*(Tt6-T7), Rair);
Cpn = Cp(0.5*(Tt6-T7));
u7sq = (2*eta_n*Cpn*Tt6*(1.0-pow(p7/pt6, (gamman-1.0)/gamman)));     // PP 28/4/97 Eq. 24
T7 = Tt6 - u7sq/(2*Cpn);                                               // PP 28/4/97 Eq. 25

M7 = sqrt(u7sq/(gamman*Rair*T7));   // PP 28/4/97 Eq. 26

/**** now evaluate convergent, choked solution only if M7 > 1 */
if( (nozzleprimary == CONVERGENT) && (M7>=1.0) ) {

   T7 = 2.0*Cpn*Tt6/(gamman*Rair+2.0*Cpn);   // PP 28/4/97 Eq. 27

   u7sq = gamman*Rair*T7;

   p7 = pt6*pow( (1.0-u7sq/(2.0*eta_n*Cpn*Tt6)), gamman/(gamman-1.0)); // PP 28/4/97 Eq. 28
}

/*** Consider the by-pass stream exit now ***/
// fully expanded case
p9 = p0;
pt8 = pi_f*pi_d*delta0*p0;
u9sq = (2*eta_nb*Cpc*Tt8*(1.0-pow(p9/pt8, (gammac-1.0)/gammac)));      // PP 28/4/97 Eq. 29
T9 = Tt8 - u9sq/(2*Cpc);                                                 // PP 28/4/97 Eq. 30

M9 = sqrt(u9sq/(gammac*Rair*T9));
```

University of Cape Town

```
/*** now evaluate convergent, choked relations only if M7 > 1 */
if( (nozzlebypass == CONVERGENT) && (M9>=1.0) ) {

   T9 = 2.0*Cpc*Tt8/(gammac*Rair+2.0*Cpc);   // FP 28/4/97 Eq. 31

   u9sq = gammac*Rair*T9;

   p9 = pt8*pow( (1.0-u9sq/(2.0*eta_nb*Cpc*Tt8)), gammac/(gammac-1.0));    // FP 28/4/97 Eq. 32
}

// we have now calculated everything that we need.
// so, we can output our results to the file specified.

double s[10];  // we will ignore point 1

s[0] = 0;
s[2] = deltas(thetaO, pi_d*deltaO, Cp(0.5*T0*(1+thetaO)), Rair);
s[3] = s[2] + deltas(tau_c, pi_c, Cpc, Rair);
s[4] = s[3] + deltas(tau_b, pi_b, Cpb, Rair);
s[5] = s[4] + deltas(tau_t, pi_t, Cpt, Rair);
s[6] = s[5] + deltas(Tt6/Tt5, pi_a, Cpa, Rair);
s[7] = s[6] + deltas(T7/Tt6, p7/pt5, Cpm, Rair);
// the following eq for s[7] looks better on the graph, but
// is actually less accurate since it does not take into account
// the various specific heats which were present along the process
// path
// s[7] = deltas(T7/T0, p7/p0, Cp(0.5*(T0+T7)), Rair);
s[8] = s[2] + deltas(tau_f, pi_f, Cpc, Rair);
s[9] = s[8] + deltas(T9/Tt8, p9/pt8, Cpc, Rair);


double ss;

if(OutputFile.good()) {
   OutputFile << "s\tT";

   // if we must plot the isobars, output the pressure labels for each isobar
   if(plotisobars) {
      OutputFile << "\tp0: " << p0/1000. << "kPa\tpt2: " << delta0*p0*pi_d/1000.
      << "kPa\tpt3: " << delta0*p0*pi_d*pi_c/1000. << "kPa\tpt4: " << delta0*p0*pi_d*pi_c*pi_b/1000. << "kPa\tpt5: "
      << delta0*p0*pi_d*pi_c*pi_b*pi_t/1000. << "kPa\tpt6: " << pt6/1000. << "kPa\tp7: " << p7/1000. << "kPa";
      if(beta!=0.0) OutputFile << "\tpt8: " << pt8/1000. << "kPa\tp9: " << p9/1000. << "kPa\t";
   }

   OutputFile << endl;
```


University of Cape Town

```
        OutputFile << s[0] << "\t" << T0 << endl;
        OutputFile << s[2] << "\t" << T0*theta0 << endl;
        OutputFile << s[3] << "\t" << Tt3 << endl;
        OutputFile << s[4] << "\t" << Tt4 << endl;
        OutputFile << s[5] << "\t" << Tt5 << endl;
        OutputFile << s[6] << "\t" << Tt6 << endl;
        OutputFile << s[7] << "\t" << T7 << endl;
        if(beta) OutputFile << s[8] << "\t" << Tt8 << endl;
        if(beta) OutputFile << s[9] << "\t" << T9 << endl;

        if(plotisobars) {
            for(ss=0.0;ss<=1.1*s[7];ss+=(1.1*s[7]/100.0)){
                OutputFile << ss << "\t\t";
                if( ss<0.95*s[4] ) OutputFile << isobaric(T0, s[0], ss, Cp(0.5*(T0-T7)));
                OutputFile << "\t";
                if( ss<s[3] ) OutputFile << isobaric(T0*theta0, s[2], ss, Cpc);
                OutputFile << "\t";
                if( ss<s[5] ) OutputFile << isobaric(Tt3, s[3], ss, Cpb);
                OutputFile << "\t";
                if( (ss>0.85*s[4]) && (ss<s[5]) ) OutputFile << isobaric(Tt4, s[4], ss, Cpt);
                OutputFile << "\t";
                if( (ss>0.95*s[4]) && (ss<s[7]) ) OutputFile << isobaric(Tt5, s[5], ss, Cpa);
                OutputFile << "\t";
                if( (ss>0.95*s[6]) && (ss<1.05*s[6]) ) OutputFile << isobaric(Tt6, s[6], ss, Cpa);
                OutputFile << "\t";
                if( /*(p7!=p0) && (ss>0.95*s[6])*/ (ss>1.05*s[4]) && (ss<1.05*s[7]) ) OutputFile << isobaric(T7, s[7], ss, Cp(T7));
                OutputFile << "\t";
                if( (ss<0.9*s[3]) && (beta!=0.0) ) OutputFile << isobaric(Tt8, s[8], ss, Cpc);
                OutputFile << "\t";
                if( (p9!=p0) && (ss>0.95*s[8]) && (ss<1.05*s[9]) && (beta!=0.0) ) OutputFile << isobaric(T9, s[9], ss, Cpc);
                OutputFile << endl;
            }
        }
        return 1;
    }
    else return 0;
}

double TurboJet1D::TSFC(double M0, double T0, double p0, ABEnum ABOnOff, double& F_ma, double& f)
{

    // specific heat ratios
    double gammac; // compressor
    double gammat;
```

University of Cape Town

```
double gamman;

// specific heats
double Cpc;     // compressor
double Cpb;     // burner
double Cpt;     // turbine
double Cpa;     // afterburner
double Cpn;     // nozzle exit

double theta0;
double delta0;

// stagnation temperature ratios
double tau_c;  // compressor
double tau_b;  // burner
double tau_f;  // fan
double tau_t;  // turbine

// temperatures
double Tt3;                  // compressor exit
double Tt4 = Tt4max;         // turbine entry
double Tt5;                  // turbine exit
double Tt6;                  // afterburner exit
double Tt7;                  // afterburner exit
double T7;
double Tt8;                  // bypass fan exit
double T9;                   // bypass nozzle exit

// other
double fb;  // fuel/air ratio for primary burner
double fa;  // fuel/air ratio for afterburner

double pt6;
double p7;
double p9;
double pi_d;
double pi_t;
double pt8;

double u7sq;   // exit velocity of primary stream
double u9sq;   // exit velocity of bypass stream
double M7;     // exit mach number
double M9;


// determine stagnation conditions of intake air
```


University of Cape Town

```
theta0 = Tt_T(M0, gamma(T0, Rair));       // PP 28/4/97 Eq. 11
delta0 = pt_p(M0, gamma(T0, Rair));       // PP 28/4/97 Eq. 12

// determine avg. air properties for compressor
gammac = gamma(0.5*T0*(1.0+theta0), Rair);
Cpc = Cp(0.5*T0*(1.0+theta0));

// calculate temperature ratio across the compressor
tau_c = ( pow(pi_c, (gammac-1.0)/gammac) - 1.0)/eta_c + 1.0;      // PP 28/4/97 Eq. 13
Tt3 = tau_c*theta0*T0;                                            // PP 28/4/97 Eq. 14

// determine average burner specific heat
Cpb = Cp(0.5*(Tt3+Tt4));

// determine burner fuel/air ratio
fb = Cpb*(Tt4 - Tt3)/(QR*eta_b - Cpb*Tt4);   // PP 28/4/97 Eq. 15
tau_b = Tt4/Tt3;

// calculate temperature ratio across the fan
tau_f = ( pow(pi_f, (gammac-1.0)/gammac) - 1.0)/eta_f + 1.0;      // PP 28/4/97 Eq. 17
Tt8 = tau_f*theta0*T0;

// determine average turbine specific heat, assume drop in temp roughly equal to compressor temp rise
Cpt = Cp(Tt4 - 0.5*(Tt3-theta0*T0));      // Tt2 = theta0*T0
gammat = gamma(Tt4 - 0.5*(Tt3-theta0*T0), Rair);

// determine temperature ratio across the turbine
tau_t = 1.0 - Cpc*(beta*(tau_f-1.0)+(tau_c-1.0))/(Cpt*tau_b*tau_c*(1.0+fb)); // PP 28/4/97 Eq. 18
Tt5 = tau_t*Tt4;

// check if afterburner is on or off and apply appropriate condition
if(ABOnOff == ON) {
   Tt7 = Tt6 = Tt6max;
   Cpa = Cp(0.5*(Tt5+Tt6));
   fa = Cpa*(1+fb)*(Tt6-Tt5)/((QR*eta_a-Cpa*Tt6));    // PP 28/4/97 Eq. 20
}
else {   // afterburner is off
   Tt6 = Tt7 = tau_t*Tt4;
   Cpa = Cp(Tt6);
   fa = 0.0;           // no fuel is used
}

// we could now calculate f
f = fa + fb;

// determine the pressure ratio across the turbine
```


University of Cape Town

```
p_t = pow( (1.0-(1.0-tau_t)/eta_t), gammat/(gammat-1.0) );// PP 28/4/97 Eq. 22

// determine the stagnation pressure ratio across the diffuser
pi_d = Pi_diffuser(M0, T0, p0);
pt6 = p0*delta0*pi_d*pi_c*pi_b*pi_t*pi_a;                    // PP 28/4/97 Eq. 23

// the two different exit conditions must now be considered.
/*
If the nozzle is convergent, we need to check if it is choked.  We thus consider
it as if it were a convergent-divergent nozzle and check the exit Mach number.
If the mach number exceeds 1, we then apply the choked solution, otherwise we use the
fully expanded solution.
*/

/**** evaluate the fully expanded case: ****/

// estimate Cpn by first estimating T7, using turbine constants

gamman = gammat;
Cpn = Cpt;
p7 = p0;
u7sq = (2*eta_n*Cpn*Tt6*(1.0-pow(p7/pt6, (gamman-1.0)/gamman)));     // PP 28/4/97 Eq. 24
T7 = Tt6 - u7sq/(2*Cpn);                                               // PP 28/4/97 Eq. 25

// now reestimate Cpn
gamman = gamma(0.5*(Tt6+T7), Rair);
Cpn = Cp(0.5*(Tt6+T7));
u7sq = (2*eta_n*Cpn*Tt6*(1.0-pow(p7/pt6, (gamman-1.0)/gamman)));     // PP 28/4/97 Eq. 24
T7 = Tt6 - u7sq/(2*Cpn);                                               // PP 28/4/97 Eq. 25

M7 = sqrt(u7sq/(gamman*Rair*T7));   // PP 28/4/97 Eq. 26

/**** now evaluate convergent, choked condition only if M7 > 1  */
if( (nozzleprimary == CONVERGENT) && (M7>=1.0) ) {

   T7 = 2.0*Cpn*Tt6/(gamman*Rair+2.0*Cpn);   // PP 28/4/97 Eq. 27

   u7sq = gamman*Rair*T7;

   p7 = pt6*pow( (1.0-u7sq/(2.0*eta_n*Cpn*Tt6)), gamman/(gamman-1.0)); // PP 28/4/97 Eq. 28
}

/*** Consider the bypass stream exit now ***/
// Fully expanded case
p9 = p0;
pt8 = pi_f*pi_d*delta0*p0;
```


University of Cape Town

```
    u9sq = (2*eta_nb*Cpc*Tt8*(1.0-pow(p9/pt8, (gammac-1.0)/gammac)));        // PP 28/4/97 Eq. 29
    T9 = Tt8 - u9sq/(2*Cpc);                                                   // PP 28/4/97 Eq. 30

    M9 = sqrt(u9sq/(gammac*Rair*T9));

    /*** now evaluate convergent, choked solution only if M9 > 1 */
    if( (nozzlebypass == CONVERGENT) && (M9> 1.0) ) {

        T9 = 2.0*Cpc*Tt8/(gammac*Rair+2.0*Cpc);   // PP 28/4/97 Eq. 31

        u9sq = gammac*Rair*T9;

        p9 = pt8*pow( (1.0-u9sq/(2.0*eta_nb*Cpc*Tt8)), gammac/(gammac-1.0));   // PP 28/4/97 Eq. 32
    }

    // we have now calculated everything that we need.

    double rho0 = p0/(Rair*T0);
    double u0 = M0*sqrt(gamma(T0,Rair)*T0*Rair);
    double u7 = sqrt(u7sq);
    double u9 = (u9sq>=0)?sqrt(u9sq):0.0;

    double A7_ma = (1+f)*(p0*T7)/(rho0*p7*T0*u7);            // PP 28/4/97 Eq. 8
    double A9_ma = (u9>0)?(beta*(p0*T9)/(rho0*p9*T0*u9)):0.0;            // PP 28/4/97 Eq. 9

    F_ma = (1.0-f)*u7 - (1.0-beta)*u0 - A7_ma*(p7-p0) + beta*u9 + A9_ma*(p9-p0);     // PP 28/4/97 Eq. 10

    return F_ma;

}

// Calculate the engine thrust in Newtons [N]
int Turbojet1D::Thrust(double M0, IdealGas *atm, double p0, double aoa, double throttle, ABEnum ABOnOff, double& F, double& mdf, double &F_ma)
{

    /*
    M0 = Free stream Mach No.
    atm - atmospheric gas (includes temperature)
    p0 - free stream ambient pressure  [Pa]
    aoa - Angle of attack [radians]
    throttle = throttle position [0 <  throttle <  1.0]
    ABOnOff - Afterburner flag [On/Off]
    F = thrust [N]
    mdf = fuel consumption [kg/s]
```


University of Cape Town

```
    Return value: Error evaluation: 0 = Error
                                    1 = Success
    */

    /* Refer to ThrustFlowChar.xls for a flow chart of
       the procedure used in this routine
    */

    double pi_d_crit;
    double pi_d_act;

    double mdot;
    double mdota;

    double mdotmax;
    double mdotaengmax;

    double mdotmin;
    double mdotaengmin;

    double A7_ma;
    double A9_ma;

//  double F_ma;
    double f;

    int retval;

    // create input fluid object i.e. intake stream
    CIdealGasStream FreeStream(*atm);
    FreeStream.M(M0);
    FreeStream.p(p0);

    // calculate critical mode pressure recovery
    pi_d_crit = pCCIdiff->PiD(M0, &FreeStream);
//  pi_d_crit = Pi_d_AIA(M0, 0.0, 0.0);

    // create the exit fluid objects from the ambient gas
    CIdealGasStream GasStream7(*atm);
    CIdealGasStream GasStream9(*atm);

    // determine exit conditions
    CycleThermodyamics(M0, atm->T(), p0, ABOnOff, pi_d_crit, GasStream7, GasStream9, A7_ma, A9_ma, F_ma, f);

    // the ratio of exit area to mass flow through the gas generator has now been calculated
```


University of Cape Town

```
    // Since we know the maximum primary exit area, we can calculate the mass flow rate

    // max flow possible through the engine  i.e. exhaust full open
    mdotaengmax = m_A7max/A7_ma;
//  if(mdotaengmax > 72) mdotaengmax = 72;

    mdotmax = mdotaengmax*(1+beta);        // PP eq 6
    if(mdotmax > m_mdotmax) mdotmax = m_mdotmax;

    // determine the minimum flow possible through the engine.
    mdotaengmin = m_A7min/A7_ma;
    mdotmin = mdotaengmin*(1+beta);

    // create an intake gas stream
    double mdotactual;

    // Now see how the intake is really performing
    pi_d_act = pCCTdiff->PiD(&FreeStream, 0.0, mdotmin, mdotmax, mdotactual);
//  pi_d_act = Pi_d_AIA(M0, 0.0, 0.0);

    // determine exit conditions
    double f_F_ma;
    f_F_ma = CycleThermodynamics(M0, atm->T(), p0, ABOnOff, pi_d_act , GasStream7, GasStream9, A7_ma, A9_ma, F_ma, f);

    // determine the exit mass flow rates
    double mdot7, mdot9;
//  double mdota;

    mdota = mdotactual/(1+beta);

    // and the fuel consumption
    mdf = f_F_ma * F_ma*mdota;


    mdot7 = mdota + mdf;
//  mdot9 = mdotactual - mdot7;
    mdot9 = mdota*beta;

    // determine the exit areas
    double A7, A9;
//  A7 = A7_ma * mdot7;
//  A9   A9_ma * mdot9;

    A7 = A7_ma * mdota;
    A9 = A9_ma * mdota;
```

University of Cape Town

```
    // dump some stuff to an output list
    o_A7                = A7;;
    o_A9                = A9;;
    o_A7_ma             = A7_ma;
    o_mdotaengmax       = mdotaengmax;;
    o_mdotmax           = mdotmax;;
    o_mdotaengmin       = mdotaengmin;;
    o_mdotmin           = mdotmin;;
    o_mdotactual        = mdotactual;;
    o_pi_d              = pi_d_act;
    o_TSFC              = f_F_ma;

    // now calculate the thrust
    F =     F_ma * mdota;


    return 1;

}


double Turbojet1D::CycleThermodyamics(double M0, double T0, double p0, ABEnum ABOnOff, double pi_d, CIdealGasStream& ex7, CIdealGasStream& ex9,
double& A7_ma, double& A9_ma, double& F_ma, double& f)
{

    // specific heat ratios
    double gammac; // compressor
    double gammat;
    double gamman;

    // specific heats
    double Cpc;     // compressor
    double Cpb;     // burner
    double Cpt;     // turbine
    double Cpa;     // afterburner
    double Cpn;     // nozzle exit

    double theta0;
    double delta0;

    // stagnation temperature ratios
    double tau_c;  // compressor
    double tau_b;  // burner
    double tau_f;  // fan
    double tau_t;  // turbine
```


University of Cape Town

```
// temperatures
double Tt3;                  // compressor exit
double Tt4 = Tt4max;         // turbine entry
double Tt5;                  // turbine exit
double Tt6;                  // afterburner exit
double Tt7;                  // afterburner exit
double T7;
double Tt8;                  // bypass fan exit
double T9;                   // bypass nozzle exit

// other
double fb;  // fuel/air ratio for primary burner
double fa;  // fuel/air ratio for afterburner

double pt6;
double p7;
double p9;
double pi_t;
double pt8;

double u7sq;   // exit velocity of primary stream
double u9sq;   // exit velocity of bypass stream
double M7;     // exit mach number
double M9;


// determine stagnation conditions of intake air
theta0 = Tt_T(M0, gamma(T0, Rair));        // PP 28/4/97 Eq. 11
delta0 = pt_p(M0, gamma(T0, Rair));        // PP 28/4/97 Eq. 12

// determine avg. air properties for compressor
gammac = gamma(0.5*T0*(1.0-theta0), Rair);
Cpc = Cp(0.5*T0*(1.0+theta0));

// calculate temperature ratio across the compressor
tau_c = ( pow(pi_c, (gammac-1.0)/gammac) - 1.0)/eta_c - 1.0;      // PP 28/4/97 Eq. 13
Tt3 = tau_c*theta0*T0;                                           // PP 28/4/97 Eq. 14

// determine average burner specific heat
Cpb = Cp(0.5*(Tt3+Tt4));

// determine burner fuel/air ratio
fb = Cpb*(Tt4 - Tt3)/(QR*eta_b - Cpb*Tt4);  // PP 28/4/97 Eq. 15
tau_b = Tt4/Tt3;

// calculate temperature ratio across the fan
```

```
tau_f = ( pow(pi_f, (gammac-1.0)/gammac) - 1.0)/eta_f + 1.0;      // PP 28/4/97 Eq. 17
Tt8 = tau_f*theta0*T0;

// determine average turbine specific heat. assume drop in temp roughly equal to compressor temp rise
Cpt = Cp(Tt4 - 0.5*(Tt3-theta0*T0));       // Tt2 = theta0*T0
gammat = gamma(Tt4 - 0.5*(Tt3-theta0*T0), Rair);

// determine temperature ratio across the turbine
tau_t = 1.0 - Cpc*(beta*(tau_f-1.0)+(tau_c-1.0))/(Cpt*tau_b*tau_c*(1.0+fb)); // PP 28/4/97 Eq. 18
Tt5 = tau_t*Tt4;

// check if afterburner is on or off and apply appropriate condition
if(ABOnOff == ON) {
   Tt7 = Tt6 = Tt6max;
   Cpa = Cp(0.5*(Tt5+Tt6));
   fa = Cpa*(1+fb)*(Tt6-Tt5)/((QR*eta_a-Cpa*Tt6));    // PP 22/4/97 Eq. 20
}
else {   // afterburner is off
   Tt6 = Tt7 = tau_t*Tt4;
   Cpa = Cp(Tt6);
   fa = 0.0;          // no fuel is used
}

// we could now calculate f
f = fa + fb;

// determine the pressure ratio across the turbine
pi_t = pow( (1.0-(1.0-tau_t)/eta_t), gammat/(gammat-1.0) );// PP 28/4/97 Eq. 22

// determine the stagnation pressure ratio across the diffuser
// pi_d = Pi_diffuser(M0, T0, p0);
pt6 = p0*delta0*pi_d*pi_c*pi_b*pi_t*pi_a;                  // PP 28/4/97 Eq. 23

// the two different exit conditions must now be considered.
/*
If the nozzle is convergent, we need to check if it is choked.  We thus consider
it as if it were a convergent divergent nozzle and check the exit Mach number.
If the mach number exceeds 1, we then apply the choked solution, otherwise we use the
fully expanded solution.
*/

/**** evaluate the fully expanded case: ****/

// estimate Cpn by first estimating T7, using turbine constants

gamman = gammat;
```

University of Cape Town

```
Cpn = Cpt;
p7 = p0;
u7sq = (2*eta_n*Cpn*Tt6*(1.0-pow(p7/pt6, (gamman-1.0)/gamman)));     // PP 28/4/97 Eq. 24
T7 = Tt6 - u7sq/(2*Cpn);                                              // PP 28/4/97 Eq. 25

// now recalculate Cpn
gamman = gamma(0.5*(Tt6+T7), Rair);
Cpn = Cp(0.5*(Tt6+T7));
u7sq = (2*eta_n*Cpn*Tt6*(1.0-pow(p7/pt6, (gamman-1.0)/gamman)));     // PP 28/4/97 Eq. 24
T7 = Tt6 - u7sq/(2*Cpn);                                              // PP 28/4/97 Eq. 25

M7 = sqrt(u7sq/(gamman*Rair*T7));   // PP 28/4/97 Eq. 26

/**** now evaluate convergent, choked solution only if M7 > 1  */
if( (nozzleprimary == CONVERGENT) && (M7>=1.0) ) {

   T7 = 2.0*Cpn*Tt6/(gamman*Rair+2.0*Cpn);   // PP 28/4/97 Eq. 27

   u7sq = gamman*Rair*T7;

   p7 = pt6*pow( (1.0-u7sq/(2.0*eta_n*Cpn*Tt6)), gamman/(gamman-1.0)); // PP 28/4/97 Eq. 28
}

/*** Consider the bypass stream exit now ***/
// fully expanded case
p9 = p0;
pt8 = pi_f*pi_d*delta0*p0;
u9sq = (2*eta_nb*Cpc*Tt8*(1.0-pow(p9/pt8, (gammac-1.0)/gammac)));     // PP 28/4/97 Eq. 29
T9 = Tt8 - u9sq/(2*Cpc);                                               // PP 28/4/97 Eq. 30

M9 = sqrt(u9sq/(gammac*Rair*T9));

/**** now evaluate convergent, choked solution only if M7 > 1  */
if( (nozzlebypass == CONVERGENT) && (M9>=1.0) ) {

   T9 = 2.0*Cpc*Tt8/(gammac*Rair+2.0*Cpc);   // PP 28/4/97 Eq. 31

   u9sq = gammac*Rair*T9;

   p9 = pt8*pow( (1.0-u9sq/(2.0*eta_nb*Cpc*Tt8)), gammac/(gammac-1.0));   // PP 28/4/97 Eq. 32
}

// we have now calculated everything that we need.

double rho0 = p0/(Rair*T0);
double u0 = M0*sqrt(gamma(T0,Rair)*T0*Rair);
```

University of Cape Town

```
    double u7 = sqrt(u7sq);
    double u9 = (u9sq>=0)?sqrt(u9sq):0.0;

    A7_ma = (1+f)*(p0*T7)/(rho0*p7*T0*u7);            // PF 28/4/97 Eq. 8
    A9_ma = (u9>0)?(beta*(p0*T9)/(rho0*p9*T0*u9)):0.0;               // PF 28/4/97 Eq. 9

    F_ma = (1.0+f)*u7 - (1.0+beta)*u0 + A7_ma*(p7-p0) + beta*u9 + A9_ma*(p9-p0);      // PF 28/4/97 Eq. 10

    ex7.T(T7);
    ex7.p(p7);
    ex7.v(u7);

    ex9.T(T9);
    ex9.p(p9);
    ex9.v(u9);

    return f/F_ma;

}

Turbojet1D::Turbojet1D()
{

    pCCIdiff = NULL;
    m_Pid_const = 0.95;

}

CConeIntake * Turbojet1D::CreateIntake(double ConeAngle, CIdealGasStream *Fluid, const char * lutfname)
{

    // create the intake
    pCCIdiff = new CConeIntake(ConeAngle, Fluid, lutfname);

    return pCCIdiff;
}
```

University of Cape Town

Appendix C-3 Propulsion Simulation Code

C.3.1 Program: TSFC

```
/*****************************************************************************\
| FILENAME: 1DPropModelTSFC.cpp                                               |
| NAME:       main                                                            |
| DESCRIPTION: Test for class Turbojet1D                                      |
| AUTHOR:     Prakesh Parbhoo                                                 |
| DATE:       1 May 1997                                                      |
| REVISION HISTORY:                                                           |
\*****************************************************************************/

#include "..\1DPropModel.hpp"
#include "..\..\DataFileInput\DataFileInput.hpp"

int main( int argc, char *argv[])
{

   if(argc<2) {
      cout << "No input file specified on command line.\n";
      cout << "Usage: 1DPropModelTs inputfile" << endl;
      return 1;
   }

   InputFile TurbojetData(argv[1], ios::in | ios::nocreate, filebuf::sh_read);

   if(!TurbojetData.good()) {
      cerr << "Inputfile: " << argv[1] << endl;
      cerr << "INPUTFILE NOT GOOD" << endl;
      return 2;
   }


   double M0_1;
   double M0_2;
   double T0;
   double p0;
   int afterburner;
   // Static Pressure Ratios
   double pi_f;   // bypass fan pressure ratio
   double pi_c;
```

University of Cape Town

```
// double pi_c_start;   // compressor pressure ratio
// double pi_c_end;  // compressor pressure ratio
   double pi_d_start;   // compressor pressure ratio
   double pi_d_end;  // compressor pressure ratio
   double pi_b;   // burner pressure ratio
   double pi_a;   // afterburner pressure ratio
   // efficiencies
   double eta_f;  // fan
   double eta_c;  // compressor
   double eta_b;  // burner
   double eta_t;  // turbine
   double eta_a;  // afterburner
   double eta_n;  // nozzle
   double eta_nb; // bypass nozzle

   double beta;
   // Maximum Temperatures
   double Tt4max; // max turbine inlet temp
// double Tt4_start;
// double Tt4_end;

   double Tt7max; // max afterburner temp
   // Fuel
   double QR;  // lower heating value of jet fuel
   // Nozzle Types
   int nozzleprimary;
   int nozzlebypass;

   char *outname[512];

   TurbojetData >> new FileInputVar("M0-1", &M0_1);
   TurbojetData >> new FileInputVar("M0-2", &M0_2);
   TurbojetData >> new FileInputVar("T0", &T0);
   TurbojetData >> new FileInputVar("p0", &p0);
   TurbojetData >> new FileInputVar("AB ON OFF", &afterburner);

   TurbojetData >> new FileInputVar("pi-f", &pi_f);
// TurbojetData >> new FileInputVar("pi-c-start", &pi_c_start);
// TurbojetData >> new FileInputVar("pi-c-end", &pi_c_end);
   TurbojetData >> new FileInputVar("pi-c", &pi_c);
   TurbojetData >> new FileInputVar("pi-b", &pi_b);
   TurbojetData >> new FileInputVar("pi-a", &pi_a);

   TurbojetData >> new FileInputVar("pi-d-start", &pi_d_start);
   TurbojetData >> new FileInputVar("pi-d-end", &pi_d_end);
```

University of Cape Town

```
    TurbojetData >> new FileInputVar("Beta", &beta);

    TurbojetData >> new FileInputVar("eta-f", &eta_f);
    TurbojetData >> new FileInputVar("eta-c", &eta_c);
    TurbojetData >> new FileInputVar("eta-b", &eta_b);
    TurbojetData >> new FileInputVar("eta-t", &eta_t);
    TurbojetData >> new FileInputVar("eta-a", &eta_a);
    TurbojetData >> new FileInputVar("eta-n", &eta_n);
    TurbojetData >> new FileInputVar("eta-nb", &eta_nb);

    TurbojetData >> new FileInputVar("MaxTt4", &Tt4max);
//  TurbojetData >> new FileInputVar("Tt4_start", &Tt4_start);
//  TurbojetData >> new FileInputVar("Tt4_end", &Tt4_end);
    TurbojetData >> new FileInputVar("MaxTt6", &Tt6max);
    TurbojetData >> new FileInputVar("JetFuelQR", &QR);

    TurbojetData >> new FileInputVar("PrimaryNozzleType", &nozzleprimary);
    TurbojetData >> new FileInputVar("BypassNozzleType", &nozzlebypass);

    TurbojetData >> new FileInputVar("OUTPUTFILE", foutname);

    if(TurbojetData.ReadData()<=0) {
       cout << "Error retrieving data" << endl;
       return 3;
    }

    TurbojetData.DeleteVarList();
    TurbojetData.close();

    // create the turbojet
    Turbojet1D TestJet;

    TestJet.Pi_f(pi_f);
    TestJet.Pi_c(pi_c);
    TestJet.Pi_b(pi_b);
    TestJet.Pi_a(pi_a);

    TestJet.BypassRatio(beta);

    TestJet.Eta_f(eta_f);
    TestJet.Eta_c(eta_c);
    TestJet.Eta_b(eta_b);
    TestJet.Eta_t(eta_t);
    TestJet.Eta_a(eta_a);
    TestJet.Eta_n(eta_n);
    TestJet.Eta_nb(eta_nb);
```

```
   TestJet.Tt_4(Tt4max);
   TestJet.Tt_6(Tt6max);

   TestJet.FuelQ(QR);
   TestJet.MainNozzleType((NozzleEnum)nozzleprimary);
   TestJet.BFNozzleType((NozzleEnum)nozzlebypass);

   ofstream dataout(foutname, ios::out, filebuf::openprot);

   double TSFC[2];
   double F_mdota[2];
   double f[2];
// double Tt4;
   double pi_d;

// dataout << "pi-c\tM0-1\tF/m.a-1\tM0-2\tF/m.a-2\n";
/*
   for(pi_c=pi_c_start;pi_c<pi_c_end;pi_c+=(pi_c_end-pi_c_start)/100.) {
      TestJet.Pi_c(pi_c);
      TSFC[0] = TestJet.TSFC(M0_1, T0, p0, (TurbojetID::ABEnum)afterburner, F_mdota[0], f[0]);
      TSFC[1] = TestJet.TSFC(M0_2, T0, p0, (TurbojetID::ABEnum)afterburner, F_mdota[1], f[1]);
      dataout << pi_c << "\t" << TSFC[0] << "\t" << F_mdota[0] << "\t" << TSFC[1] << "\t" << F_mdota[1] << endl;
   }
*/
/*
   dataout << "Tt4\tTSFC (M="<<M0_1 << ")\tF/m.a (M="<<M0_1 << ")\tTSFC (M="<<M0_2 << ")\tF/m.a (M="<<M0_2 << ")\n";

   for(Tt4=Tt4_start;Tt4 <= Tt4_end;Tt4+=(Tt4_end-Tt4_start)/100.) {
      TestJet.Tt_4(Tt4);
      TSFC[0] = TestJet.TSFC(M0_1, T0, p0, (TurbojetID::ABEnum)afterburner, F_mdota[0], f[0]);
      TSFC[1] = TestJet.TSFC(M0_2, T0, p0, (TurbojetID::ABEnum)afterburner, F_mdota[1], f[1]);
      dataout << Tt4 << "\t" << TSFC[0] << "\t" << F_mdota[0] << "\t" << TSFC[1] << "\t" << F_mdota[1] << endl;
   }
*/
   dataout << "PiA\tTSFC (M="<<M0_1 << ")\tF/m.a (M="<<M0_1 << ")\tTSFC (M="<<M0_2 << ")\tF/m.a (M="<<M0_2 << ")\n";

   for(pi_d=pi_d_start;pi_d<pi_d_end;pi_d+=(pi_d_end-pi_d_start)/100.) {
      TestJet.m_PiA_const = pi_d;
      TSFC[0] = TestJet.TSFC(M0_1, T0, p0, (TurbojetID::ABEnum)afterburner, F_mdota[0], f[0]);
      TSFC[1] = TestJet.TSFC(M0_2, T0, p0, (TurbojetID::ABEnum)afterburner, F_mdota[1], f[1]);
      dataout << pi_d << "\t" << TSFC[0] << "\t" << F_mdota[0] << "\t" << TSFC[1] << "\t" << F_mdota[1] << endl;
   }
```

University of Cape Town

```
   datаout.close();

   return 1;

}
```

C.3.2 Program: Thrust

```
/**************************************************************************************\
  FILENAME: 1DPropModel\Thrust.cpp
  NAME:        main                                                                    |
  DESCRIPTION: Test for class Turbojet1D                                               |
  AUTHOR:      Prakash Parshoo                                                         |
| DATE:        1 January 2003                                                          |
| REVISION HISTORY:                                                                    |
\**************************************************************************************/

#include "..\1DPropModel.hpp"
#include "..\..\DatafileInput\DatafileInput.hpp"
#include "../../ConeIntake/ConeIntake.h"
#include <mathutil.h>

int main( int argc, char *argv[])
{


   if(argc<2) {
      cout << "No input file specified on command line.\n";
      cout << "Usage: 1DPropModelThrust inputfile" << endl;
      return 1;
   }

   InputFile TurbojetData(argv[1], ios::in | ios::nocreate, filebuf::sh_read);

   if(!TurbojetData.good()) {
      cerr << "Inputfile: " << argv[1] << endl;
      cerr << "INPUTFILE NOT GOOD" << endl;
      return 2;
   }

   char *opmodes[] = {  "eOM_SUBSONIC",        "eOM_DETACHED",
                        "eOM_SUBCRITICAL",     "eOM_CRITICAL",
```


University of Cape Town

```
                        "COM_SUPERCRITICAL" };

    enum {ALTITUDE, TEMPERATURE, PRESSURE, DENSITY, GRAVACC, VISCOSITY, KKK, SONICV };

    double M0_start;
    double M0_end;
    double M0_step;
//  double T0;
//  double p0;

    double Alt;
    int afterburner;

    // Static Pressure Ratios
    double pi_f;    // bypass fan pressure ratio
    double pi_c;    // compressor pressure ratio
    double pi_b;    // burner pressure ratio
    double pi_a;    // afterburner pressure ratio

    // Efficiencies
    double eta_f;  // fan
    double eta_c;  // compressor
    double eta_b;  // burner
    double eta_t;  // turbine
    double eta_a;  // afterburner
    double eta_n;  // nozzle
    double eta_nb; // bypass nozzle

    double beta;    // bypass ratio

    // Maximum Temperatures
    double Tt4max; // max turbine inlet temp
    double Tt6max; // max afterburner temp

    // maximum mass flow and suction capability
    double mdotmax;    // maximum mass flow the engine is capable of
    double mdotsuc;    // mass flow at M=0 due to pumping effect

    // Fuel
    double QR;  // lower heating value of jet fuel

    // Nozzle types
    int nozzleprimary;
    int nozzlebypass;

    // Exit areas
```

University of Cape Town

```
   double A7_max;
   double A9_max;
   double A7_min;
   double A9_min;

   // reference area
   double Sw;

   // Intake / Diffuser
   double coneangle; // angle of the intake cone (degrees)
   char lutIntake[512]; // lookup table for the intake

   double iMmax;
   double iMmdot;
   double imdot;
   double irho;
   double iSubPID;

   char atmname[512];  // atmosphere data lookup

   char foutname[512];

   TurbojetData >> new FileInputVar("M0 Start", &M0_start);
   TurbojetData >> new FileInputVar("M0 End", &M0_end);
   TurbojetData >> new FileInputVar("M0-Step", &M0_step);
// TurbojetData >> new FileInputVar("T0", &T0);
// TurbojetData >> new FileInputVar("p0", &p0);
   TurbojetData >> new FileInputVar("AB-ON-OFF", &afterburner);
   TurbojetData >> new FileInputVar("Altitude", &Alt);

   TurbojetData >> new FileInputVar("pi-f", &pi_f);
// TurbojetData >> new FileInputVar("pi-c-start", &pi_c_start);
// TurbojetData >> new FileInputVar("pi-c-end", &pi_c_end);
   TurbojetData >> new FileInputVar("pi-c", &pi_c);
   TurbojetData >> new FileInputVar("pi-b", &pi_b);
   TurbojetData >> new FileInputVar("pi-a", &pi_a);

   TurbojetData >> new FileInputVar("Beta", &beta);

   TurbojetData >> new FileInputVar("eta-f", &eta_f);
   TurbojetData >> new FileInputVar("eta-c", &eta_c);
   TurbojetData >> new FileInputVar("eta-b", &eta_b);
   TurbojetData >> new FileInputVar("eta-t", &eta_t);
   TurbojetData >> new FileInputVar("eta-a", &eta_a);
   TurbojetData >> new FileInputVar("eta-n", &eta_n);
   TurbojetData >> new FileInputVar("eta-mb", &eta_mb);
```

```
    TurbojetData >> new FileInputVar("MaxMassFlow",  &mdotmax );
    TurbojetData >> new FileInputVar("PumpMassFlow", &mdotsuc );

    TurbojetData >> new FileInputVar("MaxTt4", &Tt4max);
    TurbojetData >> new FileInputVar("MaxTte", &Ttemax);
    TurbojetData >> new FileInputVar("JetFuelQR", &QR);

    TurbojetData >> new FileInputVar("PrimaryNozzleType", &nozzleprimary);
    TurbojetData >> new FileInputVar("BypassNozzleType", &nozzlebypass);

    TurbojetData >> new FileInputVar("A7 max", &A7_max);
    TurbojetData >> new FileInputVar("A9 max", &A9_max);
    TurbojetData >> new FileInputVar("A7-min", &A7_min);
    TurbojetData >> new FileInputVar("A9 min", &A9_min);

    TurbojetData >> new FileInputVar("SREF", &Sw);

    TurbojetData >> new FileInputVar("IntakeConeAngle", &coneangle);
    TurbojetData >> new FileInputVar("IntakeLookupTableName", lutIntake);

    TurbojetData >> new FileInputVar("Intake_Mmax", &iMmax);
    TurbojetData >> new FileInputVar("Intake Mmdot", &iMmdot);
    TurbojetData >> new FileInputVar("Intake_mdot", &imdot);
    TurbojetData >> new FileInputVar("Intake_rho", &irho);
    TurbojetData >> new FileInputVar("Intake SubPID", &iSubPID);

    TurbojetData >> new FileInputVar("OUTPUTFILE", foutname);

    // atmospheric data
    TurbojetData >> new FileInputVar("ATMOSPHEREDATA", atmfname);


    if(TurbojetData.ReadData()<=0) {
       cout << "Error retrieving data" << endl;
       return 3;
    }

    TurbojetData.DeleteVarList();
    TurbojetData.close();

/****************************************************************************/

    // create the atmospheric data lookup
    LookupTable Atmosphere(atmfname, LookupTable::LINEAR, LookupTable::D1D,
```

```
    cout << "Atmosphere data loaded" << endl;

/*************************************************************************/

    // create the turbojet
    Turbojet1D TestJet;

    IdealGas Air(287, CpAir);
    Air.T(Atmosphere.Lookup(Alt, (long)TEMPERATURE));
    CIdealGasStream FreeStream(Air);
    FreeStream.T(Atmosphere.Lookup(Alt, (long)TEMPERATURE));
    FreeStream.p(Atmosphere.Lookup(Alt, (long)PRESSURE));
    FreeStream.M(0.0);

    // create the intake
    CConeIntake *pIntake = TestJet.CreateIntake(DTR(coneangle), &FreeStream, iatIntake);

//  pIntake->AutoDesign(85.0, 1.1, 1.01, 2.3);
//  pIntake->AutoDesign(89.0, 1.6, 0.25, 2.2);
//  pIntake->AutoDesign(72.0, 0.6, 1.01, 2.2);
    pIntake->AutoDesign(imdot, iMmdot, irho, iMmax);
    pIntake->SubsonicPiD(iSubPiD);
    pIntake->SetMassFlows(mdotmax, mdotout);

    TestJet.Pi_f(pi_f);
    TestJet.Pi_c(pi_c);
    TestJet.Pi_b(pi_b);
    TestJet.Pi_a(pi_a);

    TestJet.BypassRatio(beta);

    TestJet.Eta_f(eta_f);
    TestJet.Eta_c(eta_c);
    TestJet.Eta_b(eta_b);
    TestJet.Eta_t(eta_t);
    TestJet.Eta_a(eta_a);
    TestJet.Eta_n(eta_n);
    TestJet.Eta_nb(eta_nb);

    TestJet.A7_max(A7_max);
    TestJet.A9_max(A9_max);
    TestJet.A7_min(A7_min);
    TestJet.A9_min(A9_min);

    TestJet.Tt_4(Tt4max);
```

University of Cape Town

```
    TestJet.Tt_6(Tt6max);

    TestJet.MaxMassFlow(mdotmax);

    TestJet.FuelQ(QR);
    TestJet.MainNozzleType((NozzleEnum)nozzleprimary);
    TestJet.BPNozzleType((NozzleEnum)nozzlebypass);

/*******************************************************************/

    ofstream dataout(outname, ios::out, filebuf::openprot);

//  double pi_c;
    double TSFC[2];
    double F_mdota[2];
    double f[2];
    double f_ma;

    double p0;
    p0 = Atmosphere.Lookup(Alt, (long)PRESSURE);

    // dump some info
    dataout << "Altitude:\t" << Alt;
    dataout << "\tReference area\t" << Sw << endl;

    double rc_, Ld_, dx_;

    pIntake->GetIntakeGeometry(rc_, Ld_, dx_);

    dataout << "rc:\t" << rc_ << endl;
    dataout << "Ld:\t" << Ld_ << endl;
    dataout << "dx:\t" << dx_ << endl;

    dataout << endl;

    dataout << "M0\tThrust\tmdotf\tF_ma\tIntakeMode\tmdotmax\tmdotmin\tmdotact\tpid\tA7_ma\tCT\tTSFC\tA7\tA9\n";

    double M, T, mdotf, CT;
    for(M=M0_start;M<=M0_end;M+=M0_step) {

        TestJet.Thrust(M, &Air, p0, 0.0, 1.0, (TurbojetAB::ABEnum)afterburner, T, mdotf, F_ma);

        CT = T/(0.5*FreeStream.Gamma()*p0*SQ(M)*Sw);

        dataout << M << "\t" << T << "\t" << mdotf << "\t" << F_ma << "\t" << opmodes[pIntake->LastOpMode()] << "\t";
        dataout << TestJet.o_mdotmax << "\t" << TestJet.o_mdotmin << "\t" << TestJet.o_mdotactual << "\t";
```


University of Cape Town

```
      dataout << TestJet.o_pi_d << "\t" << TestJet.o_A7_ma << "\t";
      dataout << CT << "\t" << TestJet.o_TSFC << "\t" << TestJet.o_A7 << "\t" << TestJet.o_A9 << endl;

   }

/* for(pi_c=pi_c_start;pi_c<=pi_c_end;pi_c+=(pi_c_end-pi_c_start)/100.) {
      TestJet.Pi_c(pi_c);
      TSFC[0] = TestJet.TSFC(M0[0], T0, p0, (Turbojet1D::ABEnum)afterburner, F_mdota[0], f[0]);
      TSFC[1] = TestJet.TSFC(M0[1], T0, p0, (Turbojet1D::ABEnum)afterburner, F_mdota[1], f[1]);
      dataout << pi_c << "\t" << TSFC[0] << "\t" << F_mdota[0] << "\t" << TSFC[1] << "\t" << F_mdota[1] << endl;
   }
*/
   dataout.close();

   return 1;

}
```

C.3.3 Program: CalcTSDiagram

```
/*****************************************************************************\
| FILENAME: 1DPropModelTs.cpp                                                 |
| NAME:      main                                                             |
| DESCRIPTION: Test for class Turbojet1D                                      |
| AUTHOR:    Prakash Parbhoo                                                  |
| DATE:      29 April 1997                                                    |
| REVISION HISTORY:                                                           |
\*****************************************************************************/

#include "..\1DPropModel.hpp"
#include "..\..\DatafileInput\DatafileInput.hpp"

int main( int argc, char *argv[])
{


   if(argc<2) {
      cout << "No input file specified on command line.\n";
      cout << "Usage: 1DPropModelTs inputfile" << endl;
      return 1;
   }
```

University of Cape Town

```
InputFile TurbojetData(argv[1], ios::in | ios::nocreate, filebuf::sh_read);

if(!TurbojetData.good()) {
   cerr << "Inputfile: " << argv[1] << endl;
   cerr << "INPUTFILE NOT GOOD " << endl;
   return 2;
}


double M0;
double T0;
double p0;
int afterburner;
// Static Pressure Ratios
double pi_f;   // bypass fan pressure ratio
double pi_c;   // compressor pressure ratio
double pi_b;   // burner pressure ratio
double pi_a;   // afterburner pressure ratio
// Efficiencies
double eta_f;  // fan
double eta_c;  // compressor
double eta_b;  // burner
double eta_t;  // turbine
double eta_a;  // afterburner
double eta_n;  // nozzle
double eta_nb; // bypass nozzle

double beta;
// Maximum Temperatures
double Tt4max; // max turbine inlet temp
double Tt6max; // max afterburner temp
// Fuel
double QR;  // lower heating value of jet fuel
// Nozzle Types
int nozzleprimary;
int nozzlebypass;

char fontname[512];

TurbojetData >> new FileInputVar("M0", &M0);
TurbojetData >> new FileInputVar("T0", &T0);
TurbojetData >> new FileInputVar("p0", &p0);
TurbojetData >> new FileInputVar("AB-ON-OFF", &afterburner);

TurbojetData >> new FileInputVar("pi_f", &pi_f);
TurbojetData >> new FileInputVar("pi-c", &pi_c);
```

```
TurbojetData >> new FileInputVar("pi-b", &pi_b);
TurbojetData >> new FileInputVar("pi-a", &pi_a);

TurbojetData >> new FileInputVar("Beta", &beta);

TurbojetData >> new FileInputVar("eta-f", &eta_f);
TurbojetData >> new FileInputVar("eta-c", &eta_c);
TurbojetData >> new FileInputVar("eta-b", &eta_b);
TurbojetData >> new FileInputVar("eta-t", &eta_t);
TurbojetData >> new FileInputVar("eta-a", &eta_a);
TurbojetData >> new FileInputVar("eta-n", &eta_n);
TurbojetData >> new FileInputVar("eta-nb", &eta_nb);

TurbojetData >> new FileInputVar("MaxTt4", &Tt4max);
TurbojetData >> new FileInputVar("MaxTt6", &Tt6max);
TurbojetData >> new FileInputVar("JetFuelQR", &QR);

TurbojetData >> new FileInputVar("PrimaryNozzleType", &nozzleprimary);
TurbojetData >> new FileInputVar("BypassNozzleType", &nozzlebypass);

TurbojetData >> new FileInputVar("OUTPUTFILE", foutname);

if(TurbojetData.ReadData()<=0) {
   cout << "Error retrieving data" << endl;
   return 3;
}

TurbojetData.DeleteVarList();
TurbojetData.close();

// create the turbojet
Turbojet1D TestJet;

TestJet.Pi_f(pi_f);
TestJet.Pi_c(pi_c);
TestJet.Pi_b(pi_b);
TestJet.Pi_a(pi_a);

TestJet.BypassRatio(beta);

TestJet.Eta_f(eta_f);
TestJet.Eta_c(eta_c);
TestJet.Eta_b(eta_b);
TestJet.Eta_t(eta_t);
TestJet.Eta_a(eta_a);
TestJet.Eta_n(eta_n);
```

University of Cape Town

```
    TestJet.Eta_nb(eta_nb);

    TestJet.Tt_4(Tt4max);
    TestJet.Tt_6(Tt6max);

    TestJet.FuelQ(QR);
    TestJet.MainNozzleType((NozzleEnum)nozzleprimary);
    TestJet.BPNozzleType((NozzleEnum)nozzlebypass);

    ofstream dataout(foutname, ios::out, filebuf::openprot);

    int result = TestJet.DrawTs(M0, T0, p0, (Turbojet1D::ABEnum)afterburner, 1, dataout);

    dataout.close();

    return (result==1)?0:4;

}
```

University of Cape Town

Appendix D: Performance Estimation Source Code

Appendix D-1 Sustained Turn Rate

```
#include <stdlib.h>
#include <fstream.h>
#include <stdio.h>
#include <math.h> // for dummy math call and float support

#include "../IdealGas/IdealGas.h"
#include "../Lookup4D/Lookup4D.h"
#include "../DataFileInput/DataFileInput.hpp"
#include <mathutil.h>

int main(void) {


   double M;
   double alt;
   double Clreq;
   double Cd;
   double n;
   double Cl;
   double Cw;
   double CL;


// char fulname[512];

/* double MList[] = {0.2,0.4,0.6,0.8,0.9,1.1,1.2,1.4,1.6,1.8,2.0,2.2};
   double AltList[] = {0.0,2000.0,4000.0,6000.0,8000.0,10000.0,12000.0,14000.0,16000.0,18000.0,20000.0};
*/

   double *MList;
   double *AltList;

   int nM, nAlt;
   int Mcnt, Altcnt;
   char txtMach[1024];  // buffers to hold text form of Mach and Alt lists
```

University of Cape Town

```
char pszAlt[1024];

int retstat=0;

char infname[512];       // input file
char outfname[512];      // output file
char atmfname[512];      // atmosphere file
char aerofnamefmt[512]; // aerodynamic data name format
char propfname[512];     // propulsion data

double altstep;
double MStep;

double nstart;
double nend;
double nstep;

// double Mstep;

double p;
double g;
double gamma;
double rho;
double mass;
double Sref;

double STR, STRnew;

int errval;
int fNoData=0;

enum {ALTITUDE, TEMPERATURE, PRESSURE, DENSITY, GRAVACC, VISCOSITY, XXX, SONICV };
enum {MACHNUMBER, THRUST, CT, PID, MDOT};

// Get the Input File

cout << "Enter the name of the input file: ";
cin >> infname;

InputFile InData(infname);

// check that input file is okay
if(InData.bad() || !InData.good() ) {
   cerr << "Bad input file: " << outfname << endl;
```


University of Cape Town

```
        return 0;
    }

    // extract the first bit of data
    InData >> new FileInputVar("OUTPUTFILE", outfname);
    InData >> new FileInputVar("ATMOSPHEREDATA", atmfname);
    InData >> new FileInputVar("PROPULSIONDATA", propfname);


    InData >> new FileInputVar("N-MACH", &nM);
    InData >> new FileInputVar("MACHNOS", pszMach);
    InData >> new FileInputVar("N-ALT", &nAlt);
    InData >> new FileInputVar("ALTNOS", pszAlt);

    InData >> new FileInputVar("AEROFNAMEFMT", aerofnamefmt);
    // get primary variables
/*  InData >> new FileInputVar("ALT0", &altstart);
    InData >> new FileInputVar("ALT1", &altend);

    InData >> new FileInputVar("M0", &Mstart);
    InData >> new FileInputVar("M1", &Mend);
*/  InData >> new FileInputVar("MSTEP", &MStep);
    InData >> new FileInputVar("ALTSTEP", &altstep);

    InData >> new FileInputVar("n0", &nstart);
    InData >> new FileInputVar("n1", &nend);
    InData >> new FileInputVar("nSTEP", &nstep);

    // get other data needed
    InData >> new FileInputVar("MASS", &mass);
    InData >> new FileInputVar("SREF", &Sref);

    InData >> new FileInputVar("BLANKNODATA", &fNoData);

    // perform the read
    InData.ReadData();

    // close the input file
    InData.close();


    // generate the lists of Altitude and Mach Number

    char *tempptr1, *tempptr2;

    AltList = new double[nAlt];
```


University of Cape Town

```
    tempptr1 = pszAlt;
    for(Altcnt=0;Altcnt<nAlt;Altcnt++) {
       AltList[Altcnt] = strtod(tempptr1, &tempptr2);
       tempptr1 = tempptr2+1;
    }

    MList = new double[nM];
    tempptr1 = pszMach;
    for(Mcnt=0;Mcnt<nM;Mcnt++) {
       MList[Mcnt] = strtod(tempptr1, &tempptr2);
       tempptr1 = tempptr2+1;
    }

    // Create a 4D lookup table to hold the Aero Data at Mach number and Altitude
    CLookup4D AeroData(MList, AltList, nM, nAlt, aeroframefmt);

    if(errval = AeroData.LastError()) { // note: not ==, but assignment
       cerr << "Error creating Lookup Table: " << errval;
       return 0;
    }

    cout << "All Data Loaded" << endl;

/*********************************/

// now the tables have been loaded, so we can do all the other processing required.


    // create the output file
    ofstream OutFile(outfname);
    if(OutFile.bad()) {
       cerr << "Unable to properly create output file" << endl;
       return 0;
    }

    // now create the atmosphere lookup
    LookupTable Atmosphere(atmfname, LookupTable::LINEAR, LookupTable::D1D);

    // create the propulsion lookup table
    LookupTable Thrust(propfname, LookupTable::LINEAR, LookupTable::D1D);

    // create an ideal gas - we need this for gamma
    IdealGas Air(287.0, CpAir);

       alt = AltList[0];
```


University of Cape Town

```
    cout << endl;
    cout << "Alt: " << alt;

    // get the gas properties at that altitude
    gamma = Air.Gamma(Atmosphere.Lookup(alt, (long)TEMPERATURE));
    p = Atmosphere.Lookup(alt, (long)PRESSURE);
    g = Atmosphere.Lookup(alt, (long)GRAVACC);
    rho = Atmosphere.Lookup(alt, (long)DENSITY);

    // print headers
    OutFile << "Altitude:\t" << alt << "\tgamma:\t" << gamma << "\tPressure:\t" << p << endl;
    OutFile << "Mass:\t" << mass << "\tRho:\t" << rho << "\tSref:\t" << Sref << endl;

    OutFile << "\n\n";

    OutFile << "M\tSTR\tSTRnew\tn\tCl\tCt\tCw\tCt/Cw\tCl/Cw\n";

  /*********************************************/

    for(M=0.0;M<=MList[nM-1];M+=MStep) {

       cout << ".";

//     OutFile << M;

       // get Ct
       Ct = Thrust.Lookup(M, (long)CT);

       // now, for sustained turn rate, Cd = Ct,
       // so look up a Cl corresponding to Cd = Ct

          // if the Mach number is lower than the data we have available
          // then use the data available
          retstat = AeroData.Lookup((M<MList[0])?MList[0]:M, alt, Ct, IL, Cl);
          Cw = mass*g/(0.5*gamma*p*SQ(M)*Sref);
          switch(retstat) {
             case CLookup4D::errNONE:

                // load factor
                n = Cl / Cw ;  // Ct / Cw = 1

                STR = g*sqrt(0.5*rho*Cl*n/ (mass*g/Sref) );
                STRnew = g*sqrt(0.5*rho*Cl*(SQ(n)-1.0)/ (n*mass*g/Sref) );

                OutFile << M << "\t";
```


University of Cape Town

```
            OutFile << STR << "\t";
            OutFile << STRnew << "\t";
            OutFile << n << "\t";
            OutFile << Cl << "\t";
            OutFile << Ct << "\t";
            OutFile << Cw << "\t";
            OutFile << Ct/Cw << "\t";
            OutFile << Cl/Cw << endl;

            break;
        case CLookup4D::errOUTOFRANGE:
            OutFile << M << "\t";
            OutFile << "\t" << "\t";
            OutFile << "\t" << "\t";
            OutFile << "\t" << "\t";
            OutFile << "\t" << "\t";
            OutFile << Ct << "\t";
            OutFile << Cw << endl;
            break;

        default:                    OutFile << M << endl;
                                    break;
        }

    }
    OutFile << endl;


  return 1;

}
```

University of Cape Town

Appendix D-2 Specific Excess Power

```
/**************************************************************************\
| FILENAME: 1DPropModelThrust.cpp                                          |
| NAME:      main                                                          |
| DESCRIPTION: Test for class Turbojet1D                                   |
| AUTHOR:    Prakash Parohoo                                               |
| DATE:      1 January 2000                                                |
| REVISION HISTORY:                                                        |
\**************************************************************************/

#include "../1DPropModel/1DPropModel.hpp"
#include "../DatafileInput\DatafileInput.hpp"
#include "../ConeIntake/ConeIntake.h"
#include "../Lookup1D/Lookup1D.h"
#include <mathutil.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main( int argc, char *argv[])
{


   if(argc<2) {
      cout << "No input file specified on command line.\n";
      cout << "Usage: 1DPropModelThrust inputfile" << endl;
      return 1;
   }

   inputfile TurbojetData(argv[1], ios::in | ios::nocreate, filebuf::sh_read);

   if(!TurbojetData.good()) {
      cerr << "Inputfile: " << argv[1] << endl;
      cerr << "INPUTFILE NOT GOOD" << endl;
      return 2;
   }

   char *opmodes[] = {  "eOM_SUBSONIC",       "eOM_DETACHED",
                        "eOM_SUBCRITICAL",    "eOM_CRITICAL",
                        "eOM_SUPERCRITICAL"   };

   enum {ALTITUDE, TEMPERATURE, PRESSURE, DENSITY, GRAVACC, VISCOSITY, KKK, SONICV };
```

University of Cape Town

```
/*****************************************************************************************/
// INPUT PARAMETERS

   double M0_start;
   double M0_end;
   double M0_step;
   double Alt_start;
   double Alt_end;
   double Alt_step;

   // double T0;
// double p0;
   int afterburner;

   // Static Pressure Ratios
   double pi_f;   // bypass fan pressure ratio
   double pi_c;   // compressor pressure ratio
   double pi_b;   // burner pressure ratio
   double pi_a;   // afterburner pressure ratio

   // Efficiencies
   double eta_f;  // fan
   double eta_c;  // compressor
   double eta_b;  // burner
   double eta_t;  // turbine
   double eta_a;  // afterburner
   double eta_n;  // nozzle
   double eta_nb; // bypass nozzle

   double beta;    // bypass ratio

   // Maximum Temperatures
   double Tt4max; // max turbine inlet temp
   double Tt6max; // max afterburner temp

   // maximum mass flow and suction capability
   double mdotmax;   // maximum mass flow the engine is capable of
   double mdotsuc;   // mass flow at M=0 due to pumping effect

   // Fuel
   double QR;  // lower heating value of jet fuel

   // Nozzle types
   int nozzleprimary;
   int nozzlebypass;
```


University of Cape Town

```
	// Exit areas
	double A7_max;
	double A9_max;
	double A7_min;
	double A9_min;

	// reference area
//	double Sw;
	double mass;
	double Sref;

	// Intake / Diffuser
	double coneangle; // angle of theintake cone (degrees)
	char lutIntake[512]; // lookup table for the intake
	char aeroTableName[512];
	char atmfname[512];

	double iMmax;
	double iMmdot;
	double imdot;
	double irho;
	double isubPiD;

	// operating conditions
	double *MList;
	double *AltList;

	int nM, nAlt;
	char pszMach[1024];  // buffers to hold text form of Mach and Alt lists
	char pszAlt[1024];

	int iNoData=0;
	char foutname[512];
	char tabname[1024];

/************************************************************************/
// LOCAL VARIABLES

	int errval;
	int Mcnt, Altcnt;

/************************************************************************/
// GET THE DATA
```

University of Cape Town

```
    // atmospheric data
    TurbojetData >> new FileInputVar("ATMOSPHERICDATA", atmfname);

    // aerodynamic data stuff
    TurbojetData >> new FileInputVar("N-MACH", &nM);
    TurbojetData >> new FileInputVar("MACHNOS", pszMach);
    TurbojetData >> new FileInputVar("N-ALT", &nAlt);
    TurbojetData >> new FileInputVar("ALTNOS", pszAlt);
    TurbojetData >> new FileInputVar("AEROFNAMEFMT", aerofnamefmt);

    // Mach numbers
    TurbojetData >> new FileInputVar("M0 Start", &M0_start);
    TurbojetData >> new FileInputVar("M0-End", &M0_end);
    TurbojetData >> new FileInputVar("M0 Step", &M0_step);

    // Altitudes
    TurbojetData >> new FileInputVar("Alt Start", &Alt_start);
    TurbojetData >> new FileInputVar("Alt-End", &Alt_end);
    TurbojetData >> new FileInputVar("Alt-Step", &Alt_step);

//  TurbojetData >> new FileInputVar("T0", &T0);
//  TurbojetData >> new FileInputVar("P0", &P0);
    TurbojetData >> new FileInputVar("AB-ON-OFF", &afterburner);

    TurbojetData >> new FileInputVar("pi-f", &pi_f);
//  TurbojetData >> new FileInputVar("pi-c-start", &pi_c_start);
//  TurbojetData >> new FileInputVar("pi-c-end", &pi_c_end);
    TurbojetData >> new FileInputVar("pi-c", &pi_c);
    TurbojetData >> new FileInputVar("pi-b", &pi_b);
    TurbojetData >> new FileInputVar("pi-a", &pi_a);

    TurbojetData >> new FileInputVar("Beta", &beta);

    TurbojetData >> new FileInputVar("eta-f", &eta_f);
    TurbojetData >> new FileInputVar("eta-c", &eta_c);
    TurbojetData >> new FileInputVar("eta-b", &eta_b);
    TurbojetData >> new FileInputVar("eta-t", &eta_t);
    TurbojetData >> new FileInputVar("eta-a", &eta_a);
    TurbojetData >> new FileInputVar("eta-n", &eta_n);
    TurbojetData >> new FileInputVar("eta-mb", &eta_mb);

    TurbojetData >> new FileInputVar("MaxMassFlow",  &mdotmax );
    TurbojetData >> new FileInputVar("PumpMassFlow", &mdotsuc );

    TurbojetData >> new FileInputVar("MaxTt4", &Tt4max);
    TurbojetData >> new FileInputVar("MaxTt6", &Tt6max);
```

University of Cape Town

```
   TurbojetData >> new FileInputVar("JetFuelQR", &QR);

   TurbojetData >> new FileInputVar("PrimaryNozzleType", &nozzleprimary);
   TurbojetData >> new FileInputVar("BypassNozzleType", &nozzlebypass);

   TurbojetData >> new FileInputVar("A7-max", &A7_max);
   TurbojetData >> new FileInputVar("A9-max", &A9_max);
   TurbojetData >> new FileInputVar("A7-min", &A7_min);
   TurbojetData >> new FileInputVar("A9-min", &A9_min);

   // aircraft characteristics
   TurbojetData >> new FileInputVar("SREF", &Sref);
   TurbojetData >> new FileInputVar("MASS", &mass);

   TurbojetData >> new FileInputVar("IntakeConeAngle", &coneangle);
   TurbojetData >> new FileInputVar("IntakeLookupTableName", IntIntake);

   TurbojetData >> new FileInputVar("Intake_Mmax", &iMmax);
   TurbojetData >> new FileInputVar("Intake_Mmdot", &iMmdot);
   TurbojetData >> new FileInputVar("Intake_mdot", &imdot);
   TurbojetData >> new FileInputVar("Intake_rho", &irho);
   TurbojetData >> new FileInputVar("Intake_SubPID", &iSubPID);

   TurbojetData >> new FileInputVar("OUTPUTFILE", foutname);
   TurbojetData >> new FileInputVar("OUTPUTTABLE", tabname);

   TurbojetData >> new FileInputVar("BLANKNODATA", &fNoData);

   if(TurbojetData.ReadData()<=0) {
      cout << "Error retrieving data" << endl;
      return 3;
   }

   TurbojetData.DeleteVarList();
   TurbojetData.close();
/**************************************************************************************************/
// CREAT THE PRIMARY LOOKUP TABLES

   // generate the lists of Altitude and Mach Number

   char *tempptr1, *tempptr2;

   AltList = new double[nAlt];
   tempptr1 = pszAlt;
   for(Altcnt=0;Altcnt<nAlt;Altcnt++) {
```

University of Cape Town

```
      AltList[Altcnt] = strtod(tempptr1, &tempptr2);
      tempptr1 = tempptr2+1;
   }

   MList = new double[nM];
   tempptr1 = pszMach;
   for(Mcnt=0;Mcnt<nM;Mcnt++) {
      MList[Mcnt] = strtod(tempptr1, &tempptr2);
      tempptr1 = tempptr2+1;
   }

   // Create a 4D lookup table to hold the Aero Data at Mach number and Altitude
   CLookup4D AeroData(MList, AltList, nM, nAlt, aerofnamefmt);

   if(errval = AeroData.LastError()) { // note: not ==, but assignment
      cerr << "Error creating Lookup Table: " << errval;
      return 0;
   }

   cout << "Aerodynamic data loaded" << endl;

   // now create the atmosphere lookup
   LookupTable Atmosphere(atmfname, LookupTable::LINEAR, LookupTable::DIE);

   cout << "Atmosphere data loaded" << endl;


   cout << "All Data loaded" << endl;
/*****************************************************************************************************/

   // create the turbojet
   Turbojet1D TestJet;

   // Set up the intake
   IdealGas Air(287, CpAir);
// Air.T(T0);
   Air.T(Atmosphere.Lookup(0.0, (long)TEMPERATURE));
   CIdealGasStream FreeStream(Air);
// FreeStream.T(T0);
// FreeStream.p(p0); // p at 15 km
   FreeStream.T(Atmosphere.Lookup(0.0, (long)TEMPERATURE));
   FreeStream.p(Atmosphere.Lookup(0.0, (long)PRESSURE));
   FreeStream.M(0.5);

   // create the intake
```


University of Cape Town

```
   CConeIntake *pIntake = TestJet.CreateIntake(DTR(coneangle), &FreeStream, iutIntake);

// pIntake->AutoDesign(89.0, 1.1, 1.01, 2.2);
// pIntake->AutoDesign(85.0, 1.6, 0.25, 2.2);
// pIntake->AutoDesign(73.0, 0.6, 1.01, 2.2);
   pIntake->AutoDesign(imdot, iMmdot, irho, iMmax);
   pIntake->SubsonicPiD(iSubPiD);
   pIntake->SetMassFlows(mdotmax, mdotsub);

   TestJet.Pi_f(pi_f);
   TestJet.Pi_c(pi_c);
   TestJet.Pi_b(pi_b);
   TestJet.Pi_a(pi_a);

   TestJet.BypassRatio(beta);

   TestJet.Eta_f(eta_f);
   TestJet.Eta_c(eta_c);
   TestJet.Eta_b(eta_b);
   TestJet.Eta_t(eta_t);
   TestJet.Eta_a(eta_a);
   TestJet.Eta_n(eta_n);
   TestJet.Eta_nb(eta_nb);

   TestJet.A7_max(A7_max);
   TestJet.A9_max(A9_max);
   TestJet.A7_min(A7_min);
   TestJet.A9_min(A9_min);

   TestJet.Tt_4(Tt4max);
   TestJet.Tt_6(Tt6max);

   TestJet.MaxMassFlow(mdotmax);

   TestJet.FuelQ(QR);
   TestJet.MainNozzleType((NozzleEnum)nozzleprimary);
   TestJet.BPNozzleType((NozzleEnum)nozzlebypass);

/*****************************************************************************************/
// CREATE THE OUTPUT FILE(S)

   ofstream dataout(foutname, ios::out, filebuf::openprot);
   ofstream tabout (tabname , ios::out, filebuf::openprot);

/*********************************************************************************************/
```


University of Cape Town

```
// double pi c;
   double TSFC[2];
   double F_mdota[2];
   double f[2];
   double F_ma;

// dataout << "M0\tThrust\tmdotf\tF_ma\tIntakeMode\tmdotmax\tmdotmin\tmdotact\tpld\tA7_ma\tC2\n";

   double M, T, mdotf, CT;

   double p, g, Temp; // , gamma;
   double Cw, Cd, SEP;

   double Alt;
   int nnAlt;

   int retstat;

   // we want the same number of altitude and Mach number points
// double AltInc;
   int stepss;

   stepss = (M0_end - M0_start)/M0_step;

   // dump the header for the table
   for(M=M0_start;M<=M0_end;M+=M0_step) tabout << "\t" << M;
   tabout << endl;

// for(nnAlt=0;nnAlt<nAlt;nnAlt++) {
   for(Alt=Alt_start;Alt<=Alt_end;Alt+=Alt_step) {
// for(nnAlt=0;nnAlt<=stepss;nnAlt++) {

//    Alt = AltList[nnAlt];
//    Alt = AltList[0] + ((double)nnAlt)*(AltList[nAlt-1] - AltList[0])/((double)stepss);

      // get the atmospheric properties
      p     = Atmosphere.Lookup(Alt, (long)PRESSURE);
      g     = Atmosphere.Lookup(Alt, (long)GRAVACC);
      Temp  = Atmosphere.Lookup(Alt, (long)TEMPERATURE);

      Air.T(Temp);

      dataout << "Altitude:\t"   << Alt   << "\t";
      dataout << "g:\t"          << g     << "\t";
      dataout << "p:\t"          << p     << "\t";
```

University of Cape Town

```
dataout << "Temp:\t"        << Temp  << endl;

dataout << "M\tCT\tCw-Cl\tCd\tSEP" << endl;

tabout << endl;
tabout << Alt;

cout << "\nAlt: " << Alt << " ";

for(M=M0_start;M<=M0_end;M+=M0_step) {

   cout << ".";

   // set the stream properties
   FreeStream.T(Temp);
   FreeStream.p(p);
   FreeStream.M(M);

   TestJet.Thrust(M, &Alt, p, 0.0, 1.0, (TurbojetID::ABEnum)afterburner, T, mdotf, F_na);

   // calculate the thrust and weight coefficients
   CT = T/(0.5*FreeStream.Gamma()*p*SQ(M)*Sref);
   Cw = mass*g/(0.5*FreeStream.Gamma()*p*SQ(M)*Sref);

   // now use Cw = Cl for n=1 flight to find Cd.
   retstat = AeroData.Lookup(!M<MList[0]?MList[0]:M, Alt, Cw, 1L, Cd);

   dataout << M << "\t" << CT << "\t" << Cw;

   switch(retstat) {
      case CLookup4D::errNONE:
                              SEP = FreeStream.v()*(CT-Cd)/Cw;
                              dataout << "\t" << Cd << "\t" << SEP << endl;
                              tabout << "\t" << SEP;
                              break;
      case CLookup4D::errOUTOFRANGE:
                              if(fNoData) {
                                 dataout << endl;
                                 tabout << "\t-9";
                              }
                              else {
                                 dataout << "\tDNIR:"<< Cw << endl;
                                 tabout << "\tDNIR:"<< Cw;
                              }
                              break;
      default:                dataout << endl;
```


University of Cape Town

```
                    break;
        }
    }
  }
  dataout.close();
  return 1;
}
```

University of Cape Town

Appendix D-3 Drag - Thrust - Load Factor Chart

D.3.1 Program: CdMnAlt

```
#include <stdlib.h>
#include <iostream.h>
#include <stdio.h>
#include <math.h> // for dummy math call and float support

#include "../IdealGas/IdealGas.h"
#include "../Lookup4D/Lookup4D.h"
#include "../DataFileInput/DataFileInput.hpp"
#include <mathutil.h>

int main(void) {

   double M;
   double alt;
   double Clreq;
   double Cd;
   double n;

// char infname[512];

/* double MList[] = {0.2,0.4,0.6,0.8,0.9,1.1,1.2,1.4,1.6,1.8,2.0,2.2};
   double AltList[] = {0.0,2000.0,4000.0,6000.0,8000.0,10000.0,12000.0,14000.0,16000.0,18000.0,20000.0};
*/

   double *MList;
   double *AltList;

   int nM, nAlt;
   int Mcnt, Altcnt;
   char pszMach[1024]; // buffers to hold text form of Mach and Alt lists
   char pszAlt[1024];


   int retstat=0;

   char infname[512];
   char outfname[512];
```

University of Cape Town

```
char atmfname[512];
char aerofnameofm[512];


double altstep;
double MStep;

double nstart;
double nend;
double nstep;

// double Mstep;

double p;
double g;
double gamma;
double mass;
double Sref;

int errval;
int fNoData=0;

enum {ALTITUDE, TEMPERATURE, PRESSURE, DENSITY, GRAVACC, VISCOSITY, KKK, SONICV };

// Get the Input file

cout << "Enter the name of the input file: ";
cin >> infname;

inputFile InData(infname);

// check that input file is okay
if(InData.bad() || !InData.good() ) {
   cerr << "Bad input file: " << outfname << endl;
   return 0;
}

// extract the first bit of data
InData >> new FileInputVar("OUTPUTFILE", outfname);
InData >> new FileInputVar("ATMOSPHEREDATA", atmfname);

InData >> new FileInputVar("N-MACH", &nM);
InData >> new FileInputVar("MACHNOS", pszMach);
InData >> new FileInputVar("N-ALT", &nAlt);
InData >> new FileInputVar("ALTNOS", pszAlt);
```

University of Cape Town

```
   InData >> new FileInputVar("AEROFNAMEFMT", aerofnamefmt);
   // get primary variables
/* InData >> new FileInputVar("ALT0", &altstart);
   InData >> new FileInputVar("ALT1", &altend);

   InData >> new FileInputVar("M0", &Mstart);
   InData >> new FileInputVar("M1", &Mend);
*/ InData >> new FileInputVar("MSTEP", &MStep);
   InData >> new FileInputVar("ALTSTEP", &altstep);

   InData >> new FileInputVar("n0", &nstart);
   InData >> new FileInputVar("n1", &nend);
   InData >> new FileInputVar("nSTEP", &nstep);

   // get other data needed
   InData >> new FileInputVar("MASS", &mass);
   InData >> new FileInputVar("SREF", &Sref);

   InData >> new FileInputVar("BLANKNODATA", &fNoData);

   // perform the read
   InData.ReadData();

   // close the input file
   InData.close();


   // generate the lists of Altitude and Mach Number

   char *tempptr1, *tempptr2;

   AltList = new double[nAlt];
   tempptr1 = pszAlt;
   for(Altcnt=0;Altcnt<nAlt;Altcnt++) {
      AltList[Altcnt] = strtod(tempptr1, &tempptr2);
      tempptr1 = tempptr2+1;
   }

   MList = new double[nM];
   tempptr1 = pszMach;
   for(Mcnt=0;Mcnt<nM;Mcnt++) {
      MList[Mcnt] = strtod(tempptr1, &tempptr2);
      tempptr1 = tempptr2+1;
   }

   // Create a 4D lookup table to hold the Aero Data at Mach number and Altitude
```


University of Cape Town

```
    CLookup4D AeroData(MList, AltList, nM, nAlt, aerofname1nt);

    if(errval = AeroData.LastError()) { // note: not ==, but assignment
       cerr << "Error creating Lookup Table: " << errval;
       return 0;
    }

    cout << "All Data Loaded" << endl;

/*********************************/

// now the tables have been loaded, so we can do all the other processing required.


    // create the output file
    ofstream OutFile(outfname);
    if(OutFile.bad()) {
       cerr << "Unable to properly create output file" << endl;
       return 0;
    }

    // now create the atmosphere lookup
    LookupTable Atmosphere(atmfname, LookupTable::LINEAR, LookupTable::DLC);

    // create an ideal gas - we need this for gamma
    IdealGas Air(287.0, CpAir);


    // now work through the Altitude and Mach Number list
    for(alt=0.0;alt<=AltList[nAlt-1];alt+=altstep) {

       cout << endl;
       cout << "Alt: " << alt;

       // get the gas properties at that altitude
       gamma = Air.Gamma(Atmosphere.Lookup(alt, (long)TEMPERATURE));
       p = Atmosphere.Lookup(alt, (long)PRESSURE);
       g   Atmosphere.Lookup(alt, (long)GRAVACC);

       // print headers
       OutFile << "Altitude:\t" << alt << "\tgamma:\t" << gamma << "\tPressure:\t" << p << endl;
       OutFile << "Alt\tMachNo";
       for(n=nstart;n<=nend;n+=nstep) OutFile << "\tn=" << n;
       OutFile << endl;
```

```
/****************************************/

    for(M=0.0;M<=MList[nM-1];M+=MStep) {

        cout << ".";

        OutFile << alt << "\t" << M;

        for(n=nstart;n<=nend;n+=nstep) {
//          Clreq = 0.5*n*mass*g/(gamma*p*SQ(M)*Sref);
            Clreq = n*mass*g/(0.5*gamma*p*SQ(M)*Sref);

            // now do the lookup for Cd
            // if the Mach number is lower than the data we have available
            // then use the data available
            retstat = AeroData.Lookup((M<MList[0])?MList[0]:M, alt, Clreq, 1L, Cd);
            switch(retstat) {
                case CLookup4D::errNONE:
                                        if(Cd<0.0) //cout << "No Data Available\n";
                                            if(fNoData) OutFile << "\t";
                                            else OutFile << "\tNDM:"<< Clreq;
                                        else OutFile << "\t" << Cd;
                                        break;
                case CLookup4D::errOUTOFRANGE:
                                        if(fNoData) OutFile << "\t";
                                        else OutFile << "\tDNTR:"<<Clreq;
                                        break;
                default:                OutFile << "\t";
                                        break;
            }

        }
        OutFile << endl;
    }

  }
  return 1;

}
```

D.3.2 Program: ClvsMvsn

```
#include <fstream.h>
#include "../LookupTable/LookupTS.hpp"
```


University of Cape Town

```
#include "../IdealGas/IdealGas.h"
#include "../DataFileInput/DataFileInput.hpp"
#include <mathutil.h>

int main(void) {

   char infname[512];

   char outfname[512];
   char atmfname[512];

   double altstart;
   double altend;
   double Mstart;
   double Mend;
   double nstart;
   double nend;

   double M, n, alt;
   double Mstep, nstep, altstep;

   double Clreq;
   double p;
   double q;
   double gamma;
   double mass;
   double Sref;

   enum {ALTITUDE, TEMPERATURE, PRESSURE, DENSITY, GRAVACC, VISCOSITY, KKK, SONICV };

   cout << "Enter the name of the input file: " << endl;
   cin >> infname;

   InputFile inData(infname);

   // check that input file is okay
   if(inData.bad() || !inData.good() ) {
      cerr << "Bad input file: " << outfname << endl;
      return 0;
   }

   inData >> new FileInputVar("OUTPUTFILE", outfname);
   InData >> new FileInputVar("ATMOSPHEREDATA", atmfname);

   // get primary variables
   InData >> new FileInputVar("ALT0", &altstart);
```


University of Cape Town

```
InData >> new FileInputVar("ALT1", &altend);
InData >> new FileInputVar("ALTSTEP", &altstep);

InData >> new FileInputVar("M0", &Mstart);
InData >> new FileInputVar("M1", &Mend);
InData >> new FileInputVar("MSTEP", &Mstep);

InData >> new FileInputVar("n0", &nstart);
InData >> new FileInputVar("n1", &nend);
InData >> new FileInputVar("nSTEP", &nstep);

// get other data needed
InData >> new FileInputVar("MASS", &mass);
InData >> new FileInputVar("SREF", &Sref);


InData.ReadData();

InData.close();

// create the output file
ofstream OutFile(outfname);
if(OutFile.bad()) {
   cerr << "Unable to properly create output file" << endl;
   return 0;
}


LookupTable Atmosphere(atmfname, LookupTable::LINEAR, LookupTable::CLIP);

if(altstart < Atmosphere.RangeMin()) {
   cerr << "Specified start altitude is lower than start of Atmospheric data.  Adjusting to start of atmospheric data." << endl;
   altstart = Atmosphere.RangeMin();
}

if(altend > Atmosphere.RangeMax()) {
   cerr << "Specified end altitude is higher than end of Atmospheric data.  Adjusting to end of atmospheric data." << endl;
   altend = Atmosphere.RangeMax();
}

OutFile << "Input File:\t" << infname << endl;
OutFile << "This File:\t" << outfname << endl;
OutFile << "\nMass:\t" << mass << endl;
OutFile << "Sref:\t" << Sref << "\n" << endl;
```

```
// create an ideal gas - we need this for gamma
IdealGas Air(287.0, CpAir);

for(alt=altstart;alt<=altend; alt+=altstep) {
   gamma = Air.Gamma(Atmosphere.Lookup(alt, (long)TEMPERATURE));
   p = Atmosphere.Lookup(alt, (long)PRESSURE);
   g = Atmosphere.Lookup(alt, (long)GRAVACC);

   // print headers
   OutFile << "Altitude:\t" << alt << "\tgamma:\t" << gamma << "\tPressure:\t" << p << endl;
   OutFile << "Alt\tMachNo";
   for(n=nstart;n<=nend;n+=nstep) {
      Clreq = n*mass*g/(0.5*gamma*p*SQ(M)*Sref);
      OutFile << "\tn=" << n;
   }
   OutFile << endl;

   for(M=Mstart;M<=Mend;M+=Mstep) {

      OutFile << alt << "\t" << M;

      for(n=nstart;n<=nend;n+=nstep) {
         Clreq = n*mass*g/(0.5*gamma*p*SQ(M)*Sref);
         OutFile << "\t" << Clreq;
      }
      OutFile << endl;
   }
   OutFile << endl;
}

OutFile.close();
return 1;
}
```

University of Cape Town

Appendix E: General Source Code

Appendix E-1 Taylor-Macoll Flow

E.1.1 Main File: TMFlowPropsMain.cpp

```
/* standard includes */
#include <fstream.h>
#include <time.h>    // for time() -no kidding!
#include <mathutil.h>

/* custom includes */
#include "../IdealGas/IdealGas.h"   // for class IdealGas
#include "TMFlowProps.h"

double CpAir(double T);

int main(void)
{

    // input parameters
    double Mstart, Mend;
    double stepsize;
    double M;
    double delta;
    double Tinf;

    // output variables
    double epsilon;         // conical shock wave angle
    double M2;              // Mach number behind the shock
    double Mc;              // Mach number on the cone surface
    double p2_p1;
    double pc_p1;
    double r2_r1;
    double rc_r1;

    int Ndivs = 10;
    int Ndivsmax = 1000;
```

University of Cape Town

```
int nConvMin;
int nConvMax;
int nBetaConvMax;

long sstime, eetime; // time stats

char outfname[512];  // output filename

sstime = time(NULL);

cout << "Enter the start Mach number: ";
cin >> Mstart;
cout << "Enter the end Mach number: ";
cin >> Mend;
cout << "Enter the interval size: ";
cin >> stepsiz;
cout << "Enter the cone angle [deg]: ";
cin >> delta;
cout << "Enter the free stream static temperature [K]: ";
cin >> Tinf;
cout << "Enter an output filename: ";
cin >> outfname;

IdealGas Air(287, CpAir);
Air.T(Tinf);

ofstream datafile(outfname, ios::out, filebuf::openprot);

datafile << "M1\teps\tM2\tp2/p1\tr2/r1\tMc\tpc/p1\trc/r1\n";

// get the current solution options
GetTMFlowProps(&Ndivs, &nConvMin, &nConvMax, &nBetaConvMax);

for(M=Mstart;M<=Mend;M+=stepsiz) {
   Ndivs = 10;

   int res;

   do {
      // set the number of integration steps
      SetTMFlowProps(Ndivs, nConvMin, nConvMax, nBetaConvMax);
      // find the solution
      res = TMFlowProps(M, DTR(delta), &Air, epsilon, M2, p2_p1, r2_r1, Mc, pc_p1, rc_r1);
      Ndivs*=10;
   } while (!res && (Ndivs <= Ndivsmax));
```

```
        if(res) {    // there was a solution
            datafile << M << "\t" << RTD(epsilon) << "\t" << M2 << "\t" << p2_p1 << "\t" << r2_r1 << "\t" << Mc << "\t" << pc_p1 << "\t" << rc_r1 <<
endl;
        }
        else datafile << M << "\tNo Solution at this Mach Number\n";
    }

    etime = time(NULL);

    datafile << "##Start Time: " << stime << endl;
    datafile << "##End Time: " << etime << endl;
    datafile << "Total Time: " << etime - stime << "seconds" << endl;

    datafile.close();


    return 1;
}

// Cp - calculates specific heat for air at a given temperature
double CpAir(double t)
{

    // this is based on data presented in Zucrow & Hoffman Vol. 1,
    // Section 1.14 (4), pg 57.
    if(t<1000.0) return (3.65359 - (1.33736e-3)*t + (3.29421e-6)*SQ(t) - (1.91142e-9)*t*SQ(t) + (0.275462e-12)*SQ(SQ(t)) )*287.;
    return (3.04473 + (1.33805e-3)*t - (0.488256e-6)*SQ(t) + (0.0855475e-9)*t*SQ(t) - (0.00570132e-12)*SQ(SQ(t)) )*287.;

}
```

E.1.2 Header File: TMFlowProps.h

```
#if !defined(_TMFLOWPROPS_H_PPARRUCO_23061997_)
#define _TMFLOWPROPS_H_PPARRUCO_23061997_

int TaylorMaccoll(double M1, double deltac, IdealGas *ConeFlowGas, double &epsilon, double &M2, double &Mc);
int TMFlowProps(double M1, double deltac, IdealGas *ConeFlowGas, double &epsilon, double &M2, double &p2_p1, double &r2_r1, double &Mc, double
&pc_p1, double &rc_r1);

void SetTMFlowProps(int intsteps, int ConvCntMin, int ConvCntMax, int betaConvCnt);
void GetTMFlowProps(int *intsteps, int *ConvCntMin, int *ConvCntMax, int *betaConvCnt);


#endif // !defined(_TMFLOWPROPS_H_PPARRUCO_23061997_)
```

University of Cape Town

E.1.3 Implementation File: TMFlowProps.cpp

```
/**************************************************************/
/*     TAYLOR-MACCOLL FLOW OVER A CONE                        */
/*     Determines properties behind a conical                 */
/*     shock system.                                          */
/*     Based on the method presented in                       */
/*     Zucrow, Maurice J. and Hoffman, Joe D.,                */
/*     Gas Dynamics Vol 2, Sec 16-5 pp 173-183                */
/**************************************************************/

#include <stdio.h>
#include <math.h>
#include <mathutil.h>
#include <float.h> // for _isnan()
#include <iostream.h>
#include "../routines/rk4.hpp"
#include "../IdealGas/idealGas.h"   // for class IdealGas
#include "TMFlowProps.h"

#define FALSE 0
#define TRUE 1


const double ZeroTol = 1e-6;     // Zero tolerance for iterative convergence
inline int SemiZero(double val) { return (fabs(val)<ZeroTol); }


/* Local Function Prototypes */
void TMFlow_eqn(Vector *pow, Vector *derivs, double psi);

// global variables

double gammaRK;           // gamma for RK4 derivatives
int Ndivs = 10;       // number of integration steps
int ccConvCntMin = 20;  // convergence count
int ccConvCntMax = 100; // convergence count
int betaConvCntMax = 1000; // convergence count for valid epsilon >= beta

void SetTMFlowProps(int intsteps, int ConvCntMin, int ConvCntMax, int betaConvCnt)
{
   Ndivs = intsteps;
```

```
    dcConvCntMin = ConvCntMin;
    dcConvCntMax = ConvCntMax;
    betaConvCntMax = betaConvCnt;
}

void GetTMFlowProps(int *intsteps, int *ConvCntMin, int *ConvCntMax, int *betaConvCnt)
{
    *intsteps = Ndivs;
    *ConvCntMin = dcConvCntMin;
    *ConvCntMax = dcConvCntMax;
    *betaConvCnt = betaConvCntMax;
}


int TMFlowProps(double M1, double deltac, IdealGas *ConeFlowGas, double &epsilon, double &M2, double &p2_p1, double &r2_r1, double &Mc, double
&pc_p1, double &rc_r1)
{
    int nResult;
    double gamma = ConeFlowGas->Gamma();

    nResult = TaylorMaccoll(M1, deltac, ConeFlowGas, epsilon, M2, Mc);
    // check for success
    if(nResult!=1) return nResult;

    p2_p1 = ConeFlowGas->OS_p2_p1(M1, epsilon);
    r2_r1 = ConeFlowGas->OS_r2_r1(M1, epsilon);
    pc_p1 = ConeFlowGas->pt_p(M2)*p2_p1/ConeFlowGas->pt_p(Mc);
    rc_r1 = ConeFlowGas->rt_r(M2)*r2_r1/ConeFlowGas->rt_r(Mc);

    return nResult;
}

inline double PRR(double x, double gam) { return pow( (1.0-(gam-1.0)/(gam+1.0)*SQ(x)), gam/(gam-1.0)); }
inline double RRR(double x, double gam) { return pow( (1.0-(gam-1.0)/(gam+1.0)*SQ(x)), 1.0/(gam-1.0)); }

#define ERRDATA M1 << "\t" << deltac << "\t" << e[cnt] << "\t" << deltai[cnt]

int TaylorMaccoll(double M1, double deltac, IdealGas *ConeFlowGas, double &epsilon, double &M2, double &Mc)
{
    double e[3], tan_e, sin_e;        // shock wave angles
    double dpsi[3];    // integration step sizes
    double ua[3], vs[3], rkysi[3];
    double deltai[3];
    double M1s, M2s;
    double beta, betaold;
```

```
	double thetas;
	int cnt;		// loop counter
	int seccnt;
	int dcontmin = 0;
	int dcontmax = 0;
	int betacnt;
	int fDone;

	double gamma = ConeFlowGas->Gamma();

	long iterationno=0;

	Vector state(2);  // state vector for RungeKutta integration
	RungeKutta4 ConeFlow(&state, TMFlow_equ);    // RungeKutta integration instance

	// use this to check the integration process
/*	ofstream outputfile("RKCheckData.out", ios::out, filebuf::openprot);
	ConeFlow.AppendLog(&outputfile);
	ConeFlow.OutputDerivs(2);
	ConeFlow.LogTime(0);
*/
	double alpha = asin(1.0/M1);  // free stream Mach angle;
//	M1s = sqrt((gamma+1.0)*SQ(M1)/(2.0+(gamma-1.0)*SQ(M1)));    // Z&H 3.169
	M1s = ConeFlowGas->M2Ms(M1);
	// cout << "M1*: " << M1s << endl;


	cnt = 0;
	e[cnt] = deltac + 0.5*alpha;  // Z&H 16.64

	fDone = FALSE;
	beta = 0.5*e[cnt];

	do {

//		outputfile << "Iteration #" << (iterationno++) << endl;

		betaold = beta;

		betacnt = 0;

		do {

			// if the previous value for e yielded an impossible value for beta i.e. beta > e, then re-estimate
			// e based on beta
			if(e[cnt]<beta) {
```


University of Cape Town

```
//          cout << "epsilon:" << e[cnt] << " Will be adjusted to " << beta << endl;
            e[cnt] = beta; // if we have an impossible
         }
         if( (cnt) && (e[cnt-1]==betaold) ) {
//       cout << "Previous e[cnt] was also = beta " << endl;
               e[cnt] = 1.01*betaold;
         }


         tan_e = tan(e[cnt]);
         sin_e = sin(e[cnt]);

         beta = atan( tan_e*2.0*(1.0/SQ(M1*sin_e) - 0.5*(gamma-1.0))/(gamma+1.0) );  // from ZSH 16.67


//       cout << "epsilon:" << e[cnt] << endl;
//       cout << "beta: " << beta << endl;

      } while ((beta>e[cnt]) && ((betacnt++)<betaConvCntMax) );

      if(betacnt>=betaConvCntMax) { // the estimation of beta has not converged
         cerr << "Estimation of valid epsilon has not converged" << endl;
         return 0;
      }

      // calculate the step size
      dpsi[0] =  (e[cnt] - deltac)/((double)Ndivs);

      thetas = e[cnt] - beta;


      M2s = M1s*sin_e*(2.0/((gamma+1.0)*SQ(M1*sin_e)) + (gamma-1.0)/(gamma-1.0) )/sin(beta);

//    cout << "M2s: " << M2s << endl;

      ConeFlow.SetSVEl(0, M2s*cos(beta));
      ConeFlow.SetSVEl(1, -M2s*sin(beta));
      ConeFlow.SetStep(dpsi[0]);
      ConeFlow.SetAccumulatedSteps(e[cnt]);  // set the current position to the shock angle

      // do the integration
      gammaRK = gamma;     // set the value of gamma used by the RK4 derivative function

      // get the current position - M2s*sin(beta)
      va[0] = ConeFlow.GetSVEl(1);
```


University of Cape Town

```
// v* for the last step is less than zero, so we need to iterate to find solution where v* =0

vs[1] = vs[2];

while(vs[1] < 0.0) {
   // get the current status
   us[0] = ConeFlow.GetSVEl(0);
   vs[0] = ConeFlow.GetSVEl(1);
   rkpsi[0] = ConeFlow.GetAccumulatedSteps();
   // do one more step
   ConeFlow.Run(1);
   // get the updated status
   us[1] = ConeFlow.GetSVEl(0);
   vs[1] = ConeFlow.GetSVEl(1);

   if(_isnan(vs[1])) {
      cerr << "NaN encountered after integration step." << endl;
      return 0;
   }
}

// v* has now gone positive.  we need to discard the last step and reset the state vector.
// we then change the step length.  The new step length is determined by linear interpolation
// between points currently stored in [0] and [1]

//    cout << "Doing linear interpolation on dpsi[1]\n";

if(vs[1]==vs[0]) {
   cout << "DIVIDE BY 0: v*[1] = v*[0] : " << vs[1] << endl;
   return 0;
}
else dpsi[1] = dpsi[0]*(0.0 - vs[0])/(vs[1] vs[0]);

// set the new step length
ConeFlow.SetStep(dpsi[1]);
// return the state of the integration scheme to the position at which v* is still negative
ConeFlow.SetAccumulatedSteps(rkpsi[0]);
ConeFlow.SetSVEl(0, us[0]);
ConeFlow.SetSVEl(1, vs[0]);

ConeFlow.Run(1);

us[2] = ConeFlow.GetSVEl(0);
vs[2] = ConeFlow.GetSVEl(1);

secent =0 ;
```


University of Cape Town

```
      while(!SemiZero(vs[2])) {
//       cout << "Applying secant method to dpsi[2]\n";

         // use secant method to find next step length to be used.
         if(vs[2]==vs[1]) {
            cout << "DIVIDE BY 0: v*[2] - v*[1] : " << vs[2] << endl;
            return 0;
         }
         else dpsi[2] = (dpsi[1]-dpsi[0])*(0.0-vs[2])/(vs[2]-vs[1]) + dpsi[1];

         ConeFlow.SetStep(dpsi[2]);
         // return the state of the integration scheme to the position at which v* is still negative
         ConeFlow.SetAccumulatedSteps(-kpsi[0]);
         ConeFlow.SetSVEl(0, us[0]);
         ConeFlow.SetSVEl(1, vs[0]);

         ConeFlow.Run(1);

         dpsi[0] = dpsi[1];
         dpsi[1] = dpsi[2];
         vs[1] = vs[2];

         us[2] = ConeFlow.GetSVEl(0);
         vs[2] = ConeFlow.GetSVEl(1);

         // secant method is not yielding a solution
         if( (secant++)>100 ) {
            cout << "Secant method not converging" << endl;
            return 0;
         }
      }

//    cout << "v*[2] should be within SemiZero\n";

      // at this point, we should be close enough to where we want to be.  Now get the current
      // cone angle which satisfies the wave angle and estimate a new wave angle based on that.

      deltai[cnt] = ConeFlow.GetAccumulatedSteps();

      if(!SemiZero(deltac-deltai[cnt])) {
         switch(cnt) {
            case 0:  e[cnt+1] = e[cnt] + 0.5*(deltac - deltai[cnt]);
                     cnt++;
                     break;
            case 1:  e[cnt+1] = e[cnt] + (e[cnt] - e[cnt-1])*(deltac - deltai[cnt])/(deltai[cnt]-deltai[cnt-1]);
```


University of Cape Town

```
                    e[0] = e[1];
                    e[1] = e[2];
                    if( (deltai[1]<deltac) && (deltai[0]<deltac) ) {
                       dccntmin++;
                       dccntmax--;
                    }
                    if( (deltai[1]>deltac) && (deltai[0]>deltac) ) {
                       dccntmin--;
                       dccntmax++;
                    }
                    deltai[0] = deltai[1];
                    if( (dccntmin>dcConvCntMin) || (dccntmax>dcConvCntMax) ) {     // if the number of iterations has exceeded a certain
                       cout << "Iteration count has been exceeded: Max:" << dccntmax << " of " << dcConvCntMax << " and Min: " << dccntmin << "
of" << dcConvCntMin << endl;
                       return 0;                              // count, fail.
                    }
                    break;
            }
         }
         else tDone = TRUE;

         // check to see that the value just calc'd for e is still less than 90 deg.
/*
         if( (e[cnt]>M_PI_2) && (e[cnt-1]==beta) ) {
            cout << "e = 90 and previous e = beta.  Adjusting e to 1.01 beta.\n";
            e[cnt] = 1.01*beta;
         }
*/
         if(e[cnt]>M_PI_2) {
//          cout << "e[cnt]>90\n";
            e[cnt] = 0.5*(M_PI_2 + beta);
         }

      } while (!tDone);


      epsilon = e[cnt]; // set the shock wave angle

      if(epsilon>(Alpha+2.0*deltac)) {
         cout << "Calculated Angle exceeds maximum possible. " << endl;
         cout << ERRDATA << endl;
         return 0;
      }

      M2 = ConeFlowGas->Ms2M(M2s);
      Mc = ConeFlowGas->Ms2M(sqrt(SQ(us[2])+SQ(vs[2])));
```

University of Cape Town

```
    return 1;

}

// Z&H 16-62
inline double AA(double x, double y, double gam) { return ( (gam+1.0)-(gam-1.0)*(SQ(x)+SQ(y) ))/2.0; }

void TMFlow_equ(Vector *pos, Vector &derivs, double psi)
{

    double dvsdpsi;
    double a_as_sq;
    double us = pos->Get(0);
    double vs = pos->Get(1);

    derivs.Set(0, vs);        // Z&H 16 69

    // note that gammaRK is a global variable which must be set before calling this func.
    // i.e. set before calling RK4::Run()

    a_as_sq = AA(us, vs, gammaRK);

// dvsdpsi = us + a_as_sq*(us + vs/tan(psi))/(SQ(vs) a_as_sq);   // Z&H 16-61

    dvsdpsi = ((-vs*(SQ(vs)*(gammaRK-1.0)-SQ(us) gammaRK-1.0+gammaRK*SQ(us)))/tan(psi) + (-2.0*us*(SQ(us)*(gammaRK-1.0)-gammaRK+gammaRK*SQ(vs)-
1.0)) )/(SQ(vs) gammaRK 1.0+gammaRK*SQ(us)+gammaRK*SQ(vs) SQ(us));
    derivs.Set(1, dvsdpsi);

}
```


University of Cape Town

Appendix E-2 Lookup Tables

E.2.1 Header File: LookupTB.hpp

```
#if !defined(LOOKUPTABLE_H_PFARBHOO_LONCTIMEACO__INCLUDED_)
#define LOOKUPTABLE_H_PFARBHOO_LONCTIMEACO__INCLUDED_

#define TRUE 1
#define FALSE 0

class LookupTable {

   private:
      char *fname;     // data file name

      long rows;       // array dimensions
      long columns;

      long lastpos[2];  // last position in table

      double **arrayinput;
      double *arraydata;

      int fInterp;     // interpolation flag   LINEAR, LOWER, UPPER
      int f2DData;     // data format flag   either DIO i.e. 1 input column and multiple output colu
                                    //   or DIO i.e. input row and column to find data in table

      int loaddata(void);  // loads the data required

      int fOkayStat; // all okay status flag - TRUE = okay, FALSE = problem
      int errcode;     // an integer error code

   public:
      LookupTable(const char *filename, int interpolation, int datalayout);
      ~LookupTable();
//    int RefreshData(void);

      // main access functions
      double Lookup(const double keyval, long col);
      double Lookup(const double xval, const double yval);

      // functions to see if desired value is in the table's range
```

University of Cape Town

```
    int OutRange(const double keyval);
    int OutRange(const double xval, const double yval);

    // functions which return the min and max range.
    double LookupTable::RangeMin(void);
    double LookupTable::RangeMax(void);

    // error detection functions
    int IsOK(void) { return fOkayStat; }
    int LastError(void) { return errcode; }


    // enumerations
    enum {LINEAR, LOWER, UPPER};      // interpolation types
    enum {D1D, D2D};                  // data formats
    // error codes
    enum {FILENOTFOUND, BADMALLOC, OUTOFRANGE};

};

#endif
```

E.2.2 Implementation File: LookupTB.cpp

```
#include <string.h>
#include <fstream.h>

#include "lookuptb.hpp"

LookupTable::LookupTable(const char *filename, int interpolation, int datalayout)
{

   fInterp = interpolation;
   f2DData = datalayout;

   lastpos[0] = lastpos[1] = 0;

   // check to see if we have a file name
   if(!filename) {
      fOkayStat = FALSE;
// 27 Oct 97
      arraydata = NULL; // so that we don't generate errors when destructing
      arrayinput = NULL;
      return;
   }
```

University of Cape Town

```
   fname = new char[strlen(filename)+1];
   strcpy(fname, filename);

   ifstream InputFile(fname, ios::in | ios::nocreate);


   // check that our input stream is good for input
   if(!InputFile.good()) {
      fOkayStat = FALSE;
// 27 Oct 97
      arraydata = NULL; // so that we don't generate errors when destructing
      arrayinput = NULL;
      if(fname) {
         delete fname;
         fname = NULL;
      }
      return;
   }

   // read in the dimension data
   InputFile >> rows;
   InputFile >> columns;

   switch(f2DData) {    // use a switch in case we add more data layouts later

      case D2D:    // the data has an input row and column set and data in a 2D array.

      case D1D:    // the data is arranged with one input column and a set of data columns
         {
            long cnt, rcnt, ccnt;
            arraydata = new double[rows*columns];
            arrayinput = new double *[rows];
            if(arraydata && arrayinput) {
               arrayinput[0] = arraydata;
               for(cnt=1; cnt<rows ; cnt++) arrayinput[cnt] = arrayinput[cnt-1] + columns;
               for(rcnt=0;rcnt<rows;rcnt++)
                  if(!InputFile.eof())
                     for(ccnt=0;ccnt<columns;ccnt++)
                        if(!InputFile.eof()) InputFile >> arrayinput[rcnt][ccnt];   // note swap on rows and cols
                        else break;
                  else break;
               rows = rcnt;   // set the number of rows to the number actually read in.
            }
            else {
               if(arraydata) delete [] arraydata;
```

```
            if(arrayinput) delete [] arrayinput;
         }
      }
   }
   // 27 10 97
   InputFile.close();
}

LookupTable::~LookupTable()
{
   if(arraydata) delete [] arraydata;
   if(arrayinput) delete [] arrayinput;
   if(frame) delete [] frame;
   rows = columns = 0;
   iOkayStat = FALSE;
}


double LookupTable::Lookup(const double keyval, long col)
{
   long startt, endd, cpos;
   double val;

   // check that we have the right data format for the use of this function
   if(f2DData!=D1D) return 0.0;

   startt = (lastpos[0]>0)?(lastpos[0]-1):0;       // expand our search area by one
   endd = (lastpos[0]<rows)?(lastpos[0]+1):rows;   // on either side of the last pos

   // check if the key falls outside this expanded search zone. If so, search entire table.
   if( (keyval<arrayinput[startt][0]) || (keyval>arrayinput[endd][0]) ) {
      startt = 0;
      endd = rows;
   }

   while ((endd-startt) > 1) {
      cpos = (endd-startt)/2 + startt;
      val = arrayinput[cpos][0];
      if(val>keyval) endd = cpos;
      else if(val<keyval) startt = cpos;
      else if(val==keyval) {
```

University of Cape Town

```
            lastpos[0] = cpos;
            return arrayinput[cpos][col];
        }
    }

    switch(finterp) {
        case LINEAR:
            {
                double dx = arrayinput[startt][0] - arrayinput[endd][0];
                double dy = arrayinput[startt][col] - arrayinput[endd][col];
                lastpos[0] = startt;
                return arrayinput[startt][col] + (keyval-arrayinput[startt][0])*dy/dx;
            }
        case LOWER:     lastpos[0] = startt;
                        return arrayinput[startt][col];
        case UPPER:     lastpos[0] = endd;
                        return arrayinput[endd][col];

    }
    return 0.;
}

double LookupTable::Lookup(const double xval, const double yval)
{

    long startt, endd, cpos;
    long startty, enddy, cposy;
    double val, valy;


    if(t2DData!=D2D) return 0.0;

    /********************/
    // get the row header position

    startty = (lastpos[1]>0)?lastpos[1]-1:0;
    enddy = (lastpos[1]<(columns-1))?(lastpos[1]-1):(columns-1);

    // check if the key falls outside this expanded search zone.  If so, search entire table.
    if( (yval<arrayinput[0][startty]) || (yval>arrayinput[0][enddy]) ) {
        startty = 0;
        enddy = columns-1;
    }

    while ((enddy-startty) > 1) {
        cposy = (enddy-startty)/2 + startty;
```

University of Cape Town

```
    valy = arrayinput[0][cposy];
    if(valy>yval) enddy = cposy;
    else if(valy<yval) startty = cposy;
    else if(valy==yval) lastpos[1] = enddy = startty = cposy;
}

/***********************/
// get the row input position

startt = (lastpos[0]>0)?(lastpos[0]-1):0;            // expand our search area by one
endd = (lastpos[0]<(rows-1))?(lastpos[0]+1):(rows-1); // on either side of the last pos

// check if the key falls outside this expanded search zone.  If so, search entire table.
if( (xval<arrayinput[startt][0]) || (xval>arrayinput[endd][0]) ) {
    startt = 0;
    endd = rows;
}

while ((endd-startt) > 1) {
    cpos = (endd-startt)/2 + startt;
    val = arrayinput[cpos][0];
    if(val>xval) endd = cpos;
    else if(val<xval) startt = cpos;
    else if(val==xval) lastpos[0] = endd = startt = cpos;
}

// check to see if we have an exact match.
if( (endd == startt) && (enddy == startty) ) return arrayinput[cpos][cposy];

switch(f_interp) {
    case LINEAR:
        {
            if( (endd != startt) && (enddy != startty) ) { // interpolate on both
                double tempy1, tempy2;

                double dy = arrayinput[0][startty] - arrayinput[0][enddy];
                double dx = arrayinput[startt][startty] - arrayinput[startt][enddy];
                tempy1 = arrayinput[startt][startty] + (yval-arrayinput[0][startty])*dx/dy;

                dx = arrayinput[endd][startty] - arrayinput[endd][enddy];
                tempy2 = arrayinput[endd][startty] + (yval-arrayinput[0][startty])*dx/dy;

                dx = arrayinput[startt][0] - arrayinput[endd][0];
                dy = tempy1 - tempy2;
```

```
                lastpos[0] = startt;
                lastpos[1] = startty;
//              return tempy1 + (xval-tempx1)*dy/dx;
                return tempy1 + (xval-arrayinput[startt][0])*dy/dx;
            }
            else if( (endd == startt)  && (enddy != startty) ) { // interpolate on y
                double dy = arrayinput[0][startty] - arrayinput[0][enddy];
                double dx = arrayinput[cpos][startty] - arrayinput[cpos][enddy];
                lastpos[1] = startty;
                return arrayinput[cpos][startty] + (yval-arrayinput[0][startty])*dx/dy;
            }
            else { // interpolate on x - y exact
                double dx = arrayinput[startt][0] - arrayinput[endd][0];
                double dy = arrayinput[startt][cposy] - arrayinput[endd][cposy];
                lastpos[0] = startt;
                return arrayinput[startt][cposy] + (xval-arrayinput[startt][0])*dy/dx;
            }
        }

    case LOWER:     lastpos[0] = startt;
                    lastpos[1] = startty;
                    return arrayinput[startt][startty];
    case UPPER:     lastpos[0] = endd;
                    lastpos[0] = enddy;
                    return arrayinput[endd][enddy];
    }

    return 0.;
}

int LookupTable::OutRange(const double keyval)
{
    return ( (keyval<arrayinput[0][0]) || (keyval>arrayinput[rows-1][0]) );
}

int LookupTable::OutRange(const double xval, const double yval)
{

    int retval1;
    int retval2;

    /* check to see if the x and y values are in the ranges contained in the
       table borders.  Note that the upper left cell is ignored since this
       does not contain a table coordinate
    */
```


University of Cape Town

```
    retval1 = ( (xval<arrayinput[1][0]) || (xval>arrayinput[rows-1][0]) );
    retval2 = ( (yval<arrayinput[0][1]) || (yval>arrayinput[0][columns-1]) );

    return retval1 + retval2;

}

double LookupTable::RangeMin(void)
{
    return arrayinput[0][0];
}

double LookupTable::RangeMax(void)
{
    return arrayinput[rows-1][0];
}
```

E.2.3 Header File: Lookup4D.h

```
// Lookup4D.h: interface for the CLookup4D class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_LOOKUP4D_H__B50464D2_55D0_11D1_BED5_000000000000__INCLUDED_)
#define AFX_LOOKUP4D_H__B50464D2_55D0_11D1_BED5_000000000000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "../LookupTable/LookupTb.hpp"

class CLookup4D
{

public:
    enum eErrCodes { errNONE, errFILENOTFOUND, errBADMALLOC, errOUTOFRANGE, errNODATA, errBADDIMENSIONS, errNONAMETEMPLATE, errTABLECREATION,
errUNKNOWN };
    enum eInterp { LINEAR, LOWER, UPPER };

public:
    int OutRange(const double rowval, const double colval);
    int LastError(void);
```

University of Cape Town

```
    // methods
    int Lookup(double keyrow, double keycol, double tablekey, long tablecol, double& result);

    // constructors & destructor
    CLookup4D(double* rowkeys, double* colkeys, long rows, long cols, const char* fnametemplate, eInterp InterpType=LINEAR, int
LayoutType=LookupTable::D1D);
    CLookup4D();
    virtual ~CLookup4D();

protected:
    void FreeArray(void);
    LookupTable ***m_ArrayBase;
    eErrCodes m_errcode;

    long m_columns;
    long m_rows;
    double *m_pcolkeys;  // column headers - i.e. across
    double *m_prowkeys;  // row headers - i.e. down

    long lastpos[2];  // last position in table.

    eInterp m_eInterp;

    char *m_pfnameformat;

};

#endif // !defined(AFX_LOOKUP4D_H__B5346408_5508_11D1_8E05_000000000000__INCLUDED_)
```

E.2.4 Implementation File: Lookup4D.cpp

```
// Lookup4D.cpp: implementation of the CLookup4D class.
//
//////////////////////////////////////////////////////////////////////

#include <stdlib.h>
#include <stdio.h>
//#include <memory.h>
#include <string.h>
#include "Lookup4D.h"

//////////////////////////////////////////////////////////////////////
```


University of Cape Town

```
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CLookup4D::CLookup4D()
{

   // set all pointer members to null and everything else to zero

   m_ArrayBase = NULL;
   m_columns = 0L;
   m_eInterp = LINEAR;
   m_errcode = errNONE;
   m_pcolkeys = NULL;
   m_pfnameformat = NULL;
   m_prowkeys = NULL;
   m_rows = 0L;
   lastpos[0] = lastpos[1] = 0;

}

CLookup4D::~CLookup4D()
{

   FreeArray();

   // release the memory used for the filename format string - allocated with _strdup()
   if(m_pfnameformat) free(m_pfnameformat);

}

CLookup4D::CLookup4D(double* rowkeys, double* colkeys, long rows, long cols, const char* fnametemplate, eInterp InterpType, int LayoutType)
{

   long rowcnt;
   long colcnt;


   char framebuf[512];

   // check that we've been supplied with key data
   if( (!rowkeys) || (!colkeys) ) {
      m_errcode = errNODATA;
      return;
   }

   // check that the dimensions are valid
```


University of Cape Town

```
if( (rows<1) || (cols<1) ) {
   m_errcode = errBADDIMENSIONS;
   return;
}

// check that a name template has been supplied
if(!fnametemplate) {
   m_errcode = errNONAMETEMPLATE;
   return;
}

// allocate the memory
m_ArrayBase = new LookupTable **[rows];   // array of pointer to pointer
if(m_ArrayBase) { // successfully allocated
   m_ArrayBase[0] = new LookupTable *[rows*cols]; // array of pointers
   if(!m_ArrayBase[0]) { // allocation failed
      delete m_ArrayBase;
      m_errcode = errBADMALLOC;
      return;
   }
   // set all the pointers to NULL
   memset(m_ArrayBase[0], 0, rows*cols);
}
else {          // if m_ArrayBase was not allocated
   m_errcode = errBADMALLOC;
   return;
}

// set up the table of pointers
for(rowcnt=1;rowcnt<rows;rowcnt++) m_ArrayBase[rowcnt] = m_ArrayBase[rowcnt-1] + cols;

m_rows = rows;
m_columns = cols;

for(rowcnt=0;rowcnt<rows;rowcnt++) {
   for(colcnt=0;colcnt<cols;colcnt++) {
      // create the filename
      sprintf(fnamebuf, fnametemplate, rowkeys[rowcnt], colkeys[colcnt]);
      m_ArrayBase[rowcnt][colcnt] = new LookupTable(fnamebuf, InterpType, LayoutType);
      if(!m_ArrayBase[rowcnt][colcnt]) {
         m_errcode = errTABLECREATION;
         FreeArray();
         return;
      }
      if(!m_ArrayBase[rowcnt][colcnt]->IsOK()) { // table not happy
         m_errcode = (eErrCodes)m_ArrayBase[rowcnt][colcnt]->LastError();
```


University of Cape Town

```
            FreeArray();
            return;
         }
      }

   }

   m_eInterp = InterpType;
   // may want to add code to actually copy the data from colkeys to our own location
   m_pcolkeys = colkeys;
   m_prowkeys = rowkeys;
   m_pfnameformat = _strdup(fnametemplate);

   lastpos[0] = lastpos[1] = 0;

}

void CLookup4D::FreeArray()
{

   long rowcnt, colcnt;

   if(m_ArrayBase) {
      if(m_ArrayBase[0]) {
         for(rowcnt=0;rowcnt<m_rows;rowcnt++)
            for(colcnt=0;colcnt<m_columns;colcnt++)
               if(m_ArrayBase[rowcnt][colcnt]) delete m_ArrayBase[rowcnt][colcnt];
         delete m_ArrayBase[0];
         m_ArrayBase[0] = NULL;
      }
      delete m_ArrayBase;
      m_ArrayBase = NULL;
   }

   m_rows = 0L;
   m_columns = 0L;

}

int CLookup4D::Lookup(double keyrow, double keycol, double tablekey, long tablecol, double& result)
{

   long startl, endl, cpos;
   long startty, enddy, cposy;
   double val, valy;
```

University of Cape Town

```
   // check that we have a valid set of lookup keys
   if(OutRange(keyrow, keycol)) {
      m_errcode = errOUTOFRANGE;
      return m_errcode;
   }

// if(!fLUData(4D)) return 0.0;

   /*********************/
   // get the row border position

   startty = (lastpos[1]>0)?(lastpos[1]-1):0;
   enddy = (lastpos[1]<(m_columns-1))?(lastpos[1]+1):(m_columns-1);

   // check if the key falls outside this expanded search zone.  If so, search entire table.
   if( (keycol<m_pcolkeys[0]) || (keycol>m_pcolkeys[enddy]) ) {
      startty = 0;
      enddy = m_columns-1;
   }

   // could be that we now have two adjacent columns, one of which is an exact match.
   // however, since (enddy-startty) == 1, we would never enter the following while loop.
   // so, check each column for an exact match first.

   if( (valy=m_pcolkeys[enddy]) == keycol) cposy = startty = enddy;
   else if( (valy=m_pcolkeys[startty]) == keycol) cposy = enddy = startty;

   while ((enddy-startty) > 1) {
      cposy = (enddy-startty)/2 + startty;
      valy= m_pcolkeys[cposy];
      if(valy>keycol) enddy = cposy;
      else if(valy<keycol) startty = cposy;
      else if(valy==keycol) lastpos[1] = enddy = startty = cposy;
   }


   /*********************/
   // get the row input position

   startt = (lastpos[0]>0)?(lastpos[0]-1):0;              // expand our search area by one
   endd = (lastpos[0]<(m_rows-1))?(lastpos[0]+1):(m_rows-1);  // on either side of the last pos

   // check if the key falls outside this expanded search zone.  If so, search entire table.
   if( (keyrow<m_prowkeys[startt]) || (keyrow>m_prowkeys[endd]) ) {
```

University of Cape Town

```
      startt = 0;
      endd = m_rows;
   }

   // do the same check here for exact row matches
   if( (val=m_prowkeys[endd]) == keyrow ) cpos = startt = endd;
   else if((val=m_prowkeys[startt]) == keyrow ) cpos = endd = startt;

   while ((endd-startt) > 1) {
      cpos = (endd-startt)/2 + startt;
      val = m_prowkeys[cpos];
      if(val>keyrow) endd = cpos;
      else if(val<keyrow) startt = cpos;
      else if(val==keyrow) lastpos[0] = endd = startt = cpos;
   }

   // check to see if we have an exact match.
   if( (endd == startt) && (enddy == startty) ) {
      // check that the other value is in range
      if(m_ArrayBase[cpos][cposy]->OutRange(tablekey)) return m_errcode = errOUTOFRANGE;
      // get the value we're looking for
      result = m_ArrayBase[cpos][cposy]->Lookup(tablekey, tablecol);
      return errNONE;
   }

   int frange=0;

   switch(m_eInterp) {
      case LINEAR:
         {
            if( (endd != startt) && (enddy != startty) ) { // interpolate on both

               // do a quick check to make sure that all the tables we are
               // interested in have a valid range for our query
               frange =  m_ArrayBase[startt][startty]->OutRange(tablekey);
               frange += m_ArrayBase[startt][enddy]->OutRange(tablekey);
               frange += m_ArrayBase[endd][startty]->OutRange(tablekey);
               frange += m_ArrayBase[endd][enddy]->OutRange(tablekey);

               if(frange) {
                  m_errcode = errOUTOFRANGE;
                  return m_errcode;
               }

               double tempy1, tempy2;
```


University of Cape Town

```
        double dy = m_pcolkeys[startty] - m_pcolkeys[enddy];
        double dx = m_ArrayBase[startt][startty]->Lookup(tablekey, tablecol) - m_ArrayBase[startt][enddy]->Lookup(tablekey, tablecol);
        tempy1 = m_ArrayBase[startt][startty]->Lookup(tablekey, tablecol) + (keycol-m_pcolkeys[startty])*dx/dy;

        dx = m_ArrayBase[endd][startty]->Lookup(tablekey, tablecol) - m_ArrayBase[endd][enddy]->Lookup(tablekey, tablecol);
        tempy2 = m_ArrayBase[endd][startty]->Lookup(tablekey, tablecol) + (keycol-m_pcolkeys[startty])*dx/dy;

        dx = m_prowkeys[startt] - m_prowkeys[endd];
        dy = tempy1 - tempy2;

        lastpos[0] = startt;
        lastpos[1] = startty;

        result = tempy1 + (keyrow-m_prowkeys[startt])*dy/dx;
        return errNONE;
    }
    else if( (endd == startt) && (enddy != startty) ) { // interpolate on y
        // do a outer range check
        frange = m_ArrayBase[cpos][startty]->OutRange(tablekey);
        frange += m_ArrayBase[cpos][enddy]->OutRange(tablekey);
        if(frange) {
            m_errcode = errOUTOFRANGE;
            return m_errcode;
        }

        double dy = m_pcolkeys[startty] - m_pcolkeys[enddy];
        double dx = m_ArrayBase[cpos][startty]->Lookup(tablekey, tablecol) - m_ArrayBase[cpos][enddy]->Lookup(tablekey, tablecol);
        lastpos[1] = startty;
        result = m_ArrayBase[cpos][startty]->Lookup(tablekey, tablecol) + (keycol-m_pcolkeys[startty])*dx/dy;
        return errNONE;
    }
    else { // interpolate on x - y exact
        // range check
        frange = m_ArrayBase[startt][cposy]->OutRange(tablekey);
        frange += m_ArrayBase[endd][cposy]->OutRange(tablekey);
        if(frange) {
            m_errcode = errOUTOFRANGE;
            return m_errcode;
        }

        double dx = m_prowkeys[startt] - m_prowkeys[endd];
        double dy = m_ArrayBase[startt][cposy]->Lookup(tablekey, tablecol) - m_ArrayBase[endd][cposy]->Lookup(tablekey, tablecol);
        lastpos[0] = startt;
        result = m_ArrayBase[startt][cposy]->Lookup(tablekey, tablecol) + (keyrow-m_prowkeys[startt])*dy/dx;
        return errNONE;
    }
```

University of Cape Town

```
        }

    case LOWER:     lastpos[0] = startt;
                    lastpos[1] = startty;
                    if(m_ArrayBase[startt][startty]->OutRange(tablekey)) return m_errcode = errOUTOFRANGE;

                    result = m_ArrayBase[startt][startty]->Lookup(tablekey, tablecol);
                    return errNONE;
    case UPPER:     lastpos[0] = endd;
                    lastpos[0] = enddy;
                    if(m_ArrayBase[endd][enddy]->OutRange(tablekey)) return m_errcode = errOUTOFRANGE;

                    result = m_ArrayBase[endd][enddy]->Lookup(tablekey, tablecol);
                    return errNONE;
    }

    return errUNKNOWN;

}

int CLookup4D::LastError()
{
    return m_errcode;
}

int CLookup4D::OutRange(const double rowval, const double colval)
{

    int retval1;
    int retval2;

    retval1 = ( (rowval<m_prowkeys[0]) | (rowval>m_prowkeys[m_rows-1]) );
    retval2 = ( (colval<m_pcolkeys[0]) | (colval>m_pcolkeys[m_columns-1]) );

    return retval1 + retval2;
}
```

University of Cape Town

Appendix E-3 Ideal Gas Behaviour

E.3.1 Header File: IdealGas.h

```
// IdealGas.h: interface for the IdealGas class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_IDEALGAS_H__03B7AC73_E906_11D0_BDDC_A6BFBB6C9664__INCLUDED_)
#define AFX_IDEALGAS_H__03B7AC73_E906_11D0_BDDC_A6BFBB6C9664__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class IdealGas
{
  public:
    double ShockFlowAngle(double M0, double shockangle);
    virtual void CopyFrom(IdealGas &other);
    double OS_pt2_pt1(double M1, double epsilon);
    double SonicV(void);
    double NS_pt2_pt1(double M);
    // properties - Mach # and flow angle for Talyor Macoll flow, for a ray
    int TMRayProps(double M1, double shockangle, double rayangle,  double &Mray, double &theta);
    // static density ratio across an oblique shock wave
    double OS_r2_r1(double M1, double epsilon);
    // static pressure ratio across an oblique shock wave
    double OS_p2_p1(double M1, double epsilon);
    double M2Ms(double M);
    double Ms2M(double Mstar);
    double Tt_T(double M);
    double rt_r(double M);
    double pt_p(double M);
    double Cp(void);
    void R(double newR);
    double R(void);
    double Gamma(void);
    double Gamma(double T);
    double T(void);
    void T(double newT);
    double Cp(double T);
    IdealGas(const double gasconst, double (*CpUserFunc)(double));
```

```
	IdealGas();
	virtual ~IdealGas();



protected:
	double (*CpUser)(double); // user specified Cp function taking T in [K]
	double mCp;	// specific heat [J/kg.K]
	double mgamma; // specific heat ratio
	double mT; // static temperature [K]
	double mR; // gas constant [J/kg.K]
};

/***************************/
/* GLOBAL FUNCTIONS FOR CP */
/***************************/

double CpAir(double t);


#endif // !defined(AFX_IDEALGAS_H__6387AC73_E506_11D0_BUDC_A6BFB6C9661__INCLUDED_)
```

E.3.2 Implementation File: IdealGas.cpp

```
// IdealGas.cpp: implementation of the IdealGas class.
//
//////////////////////////////////////////////////////////////////////


#include "IdealGas.h"
#include <math.h>
#include <mathutil.h>	// for SQ and constants
#include "../routines/rk4.hpp" // for the RK4 integration used in the
					// Taylor Maccoll flow stuff

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////
```


University of Cape Town

```
IdealGas::IdealGas()
{

}

IdealGas::~IdealGas()
{

}

IdealGas::IdealGas(const double gasconst, double (*CpUserFunc)(double))
{
   mR = gasconst;
   CpUser = CpUserFunc;
}

double IdealGas::Cp(double newT)
{
   return CpUser(newT);
}

void IdealGas::T(double newT)
{
   mT = newT;

   // determine the new properties
   mCp = Cp(newT);
   mgamma = Gamma(newT);
}

double IdealGas::T()
{
   return mT;
}

double IdealGas::Gamma(double newT)
{
   double Cptemp = Cp(newT);
    return Cptemp/(Cptemp - mR);
}

double IdealGas::Gamma()
{
   return mgamma;
}
```

University of Cape Town

```
double IdealGas::R()
{
   return mR;
}

void IdealGas::R(double newR)
{
   mR=newR;
}

double IdealGas::Cp()
{
   return mCp;
}


double IdealGas::pt_p(double M)
{
   return pow(( 1.0 + 0.5*(mgamma-1.0)*SQ(M)), mgamma/(mgamma-1.0) );    // C&B 16-19 pg 719
}

double IdealGas::rt_r(double M)
{
   return pow(( 1.0 + 0.5*(mgamma-1.0)*SQ(M)), 1.0/(mgamma-1.0) );    // C&B 16-20 pg 719
}

double IdealGas::Tt_T(double M)
{
   return ( 1.0 + 0.5*(mgamma-1.0)*SQ(M) );  // C&B 16-18 pg 719
}

double IdealGas::Ms2M(double Mstar)
{
   return Mstar*sqrt( 2.0/(1.0+mgamma+SQ(Mstar)*(1.-mgamma)) );       // Z&H 3.189
}

double IdealGas::M2Ms(double M)
{
   return M*sqrt( (mgamma+1.)/(2.+(mgamma-1)*SQ(M)) );      // Z&H 3.189
}

double IdealGas::OS_p2_p1(double M1, double epsilon)
{
   // static pressure ratio across an oblique shock wave
   // M1 is Mach number before shock wave
   // epsilon is wave angle
```

University of Cape Town

```
   double sin_eps = sin(epsilon);
   return 2.*mgamma*(SQ(M1*sin_eps)-(mgamma-1.)/(2.*mgamma))/(mgamma+1.);   // Z&H 7.66
}

double IdealGas::OS_r2_r1(double M1, double epsilon)
{
   double sin_eps = sin(epsilon);
   return (mgamma+1.)*SQ(M1*sin_eps)/(2.+(mgamma-1.)*SQ(M1*sin_eps));   // Z&H 7.97
}

double IdealGas::OS_pt2_pt1(double M1, double epsilon)
{
   // stagnation pressure ratio across an oblique shock wave
   // M1 is Mach number before shock wave
   // epsilon is wave angle

   double M1n; // Mach number perpendicular to the shock

   M1n = M1*sin(epsilon);

   // stagnation pressure is lost as if flow entered a normal shock
   // in a stream flowing at M1n

   return NS_pt2_pt1(M1n);

}

//**************************************************
// TAYLOR-MACCOLL (CONICAL SHOCK) GAS PROPERTIES
// The use of these functions requires the initial
// Mach number and the conical shock angle to be
// known.  The conical shock angle must have been
// determined elsewhere, such as by the iterative
// procedure used in TMFlowProps.exe
//**************************************************

void TMFlow_equ(Vector *pos, Vector &derivs, double psi);

double TMgammaRK; // global value for gamma used in integration derivative function

int IdealGas::TMRayProps(double M1, double shockangle, double rayangle, double & Mray, double & theta)
{

   double M1s, M2s;
   double eps, beta;
   double tan_e, sin_e;
```

University of Cape Town

```
double dpsi;
double us, vs;
double us1, vs1;
double thetas;

long steps = 101;

Vector state(2);  // state vector for RungeKutta integration
RungeKutta4 ConeFlow(&state, TMFlow_eqs);    // RungeKutta Integration instance

M1s = M2Ms(M1);

// note the notation: this follows the notation in ZGH

eps = shockangle;

tan_e = tan(eps);
sin_e = sin(eps);

beta = atan( tan_e*2.0*(1.0/SQ(M1*sin_e) + 0.5*(mgamma-1.0))/(mgamma+1.0) );  // from ZGH 16.67

// step size
dpsi = -(eps - rayangle)/((double)steps);

// flow angle after the shock
thetas = eps - beta;

M2s = M1s*sin_e*(2.0/((mgamma+1.0)*SQ(M1*sin_e)) + (mgamma-1.0)/(mgamma+1.0) )/sin(beta);

ConeFlow.SetSVEl(0, M2s*cos(beta));
ConeFlow.SetSVEl(1, -M2s*sin(beta));
ConeFlow.SetStep(dpsi);
ConeFlow.SetAccumulatedSteps(eps);  // set the current position to the shock angle

// do the integration
TMgammaRK = mgamma;     // set the value of gamma used by the RK4 derivative function

ConeFlow.Run(steps);

// velocities along ray
us = ConeFlow.GetSVEl(0);
vs = ConeFlow.GetSVEl(1);

// rectangular velocities
us1 = us*cos(rayangle) - vs*sin(rayangle);
vs1 = us*sin(rayangle) + vs*cos(rayangle);
```

University of Cape Town

```
   Mray = Ms2M(sqrt(SQ(us)+SQ(vs)));

   theta = atan(vs1/us1);

   return 1;
}

// ZSW 16-62
inline double AA(double x, double y, double gam) { return ( (gam+1.0)-(gam-1.0)*(SQ(x)+SQ(y) ))/2.0; }

void TMFlow_equ(Vector *pos, Vector &derivs, double psi)
{

   double dvsdpsi;
   double a_as_sq;
   double us = pos->Get(0);
   double vs = pos->Get(1);

   derivs.Set(0, vs);        // Z&H 16-69

   // note that gammaRK is a global variable which must be set before calling this func.
   // i.e. set before calling RK4::Run()

   a_as_sq = AA(us, vs, TMgammaRK);

// dvsdpsi = -us - a_as_sq*(us + vs/tan(psi))/(SQ(vs) a_as_sq);  // Z&H 16-62

   dvsdpsi = ((-vs*(SQ(vs)*(TMgammaRK-1.0)-SQ(us)-TMgammaRK-1.0+TMgammaRK*SQ(us)))/tan(psi) + (-2.0*us*(SQ(us)*(TMgammaRK-1.0)-
TMgammaRK+TMgammaRK*SQ(vs)-1.0)) )/(SQ(vs) TMgammaRK 1.0+TMgammaRK*SQ(us)+TMgammaRK*SQ(vs) SQ(us));
   derivs.Set(1, dvsdpsi);

}

double IdealGas::NS_pt2_pt1(double M)
{

   // taken from Z&H 7.40

   double part1 = ( (mgamma-1.0)/(mgamma+1.0) + 2.0/( (mgamma+1.0)*SQ(M) ) );
   double part2 = ( 2.0*mgamma*SQ(M)/(mgamma-1) - (mgamma-1.0)/(mgamma-1.0) );

   return pow( (pow(part1, mgamma)*part2), 1.0/(mgamma 1.0) );

}
```

```
/*****************************************************************/
/* GLOBAL FUNCTIONS FOR SPECIFIC HEAT PROPERTIES                */
/*****************************************************************/

// Cp   calculates specific heat for air at a given temperature
double CpAir(double t)
{

   // this is based on data presented in Zucrow & Hoffman Vol. 1.
   // Section 1.16 (a), pg 77.
   if(t<1000.0) return (3.65359 - (1.33736e-3)*t + (3.29421e-6)*SQ(t) - (1.91142e-9)*t*SQ(t) + (0.275462e-12)*SQ(SQ(t)) )*287.;
   return (3.04473 + (1.33805e-3)*t - (0.488256e-6)*SQ(t) - (0.0855475e-9)*t*SQ(t) - (0.00570132e-12)*SQ(SQ(t)) )*287.;

}

double IdealGas::SonicV()
{

   return sqrt(mgamma*mR*mT);

}


void IdealGas::CopyFrom(IdealGas & other)
{

   CpUser = other.CpUser;
   mCp = other.Cp;
   mgamma = other.mgamma;
   mR = other.mR;
   mT = other.mT;
}

double IdealGas::ShockFlowAngle(double M0, double shockangle)
{


   double eps, tan_e, sin_e, beta;


   eps = shockangle;

   tan_e = tan(eps);
   sin_e = sin(eps);
```

University of Cape Town

```
    beta = atan( tan_e*2.0*(1.0/SQ(M0*sin_e) - 0.5*(mgamma-1.0))/(mgamma+1.0) );  // from Z&H 16-47

    // flow angle after the shock
    return eps - beta;

}
```

E.3.3 Header File: IdealGasStream.h

```
// IdealGasStream.h: interface for the CIdealGasStream class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_IDEALGASSTREAM_H__40D610F1_60E0_11D1_B8E8_00000000000__INCLUDED_)
#define AFX_IDEALGASSTREAM_H__40D610F1_60E0_11D1_B8E8_00000000000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "IdealGas.h"

class CIdealGasStream : public IdealGas
{
public:
    CIdealGasStream(IdealGas& Gas);
    CIdealGasStream();
    virtual ~CIdealGasStream();

    double p(void) { return m_p; }
    double v(void) { return m_v; }
    double M(void) { return m_v/SonicV(); }
    double r(void) { return m_p/(mR*mT); }

    void p(double pressure) { m_p = pressure; }
    void v(double velocity) { m_v = velocity; }
    void M(double MachNo) { m_v = MachNo*SonicV(); }

protected:
    double m_v; // velocity
    double m_p; // pressure
```


University of Cape Town

```
};

#endif // !defined(AFX_IDEALGASSTREAM_H__40D610F1_65E0_11D1_BE55_00C000C0C550__INCLUDED_)
```

E.3.4 Implementation File: IdealGasStream.cpp

```
// IdealGasStream.cpp: implementation of the CIdealGasStream class.
//
//////////////////////////////////////////////////////////////////////

#include "IdealGasStream.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CIdealGasStream::CIdealGasStream()
{

}

CIdealGasStream::~CIdealGasStream()
{

}

CIdealGasStream::CIdealGasStream(IdealGas & Gas)
{

	m_p = 0.0;
	m_v = 0.0;

	// copy the values from the input gas
	CopyFrom(Gas);


}
```


University of Cape Town

Appendix E-4 Data Input File System

E.4.1 Header File: DatafileInput.hpp

```
/***************************************************************************\
| FileName: DatafileInput.hpp                                               |
| Name:     Classes InputFile, FileInputVar                                 |
| Description: Implements two classes for inputing data from a              |
|              structured, labeled data file                                |
| Author:   Prakash Parbhoo                                                 |
| Date:     18/04/97                                                        |
| Revision history:                                                         |
\***************************************************************************/

#include <string.h>

class FileInputVar
{

   private:
      char varname[256];
      union {
         int *intptr;
         double *doubleptr;
         char *charptr;
      } var;

      FileInputVar *nextvar;

   public:
      // public data
      enum vartype { varINT, varDOUBLE, varSTRING };

      // constructors
      FileInputVar(const char *vname, int *varvar);
      FileInputVar(const char *vname, int *varvar, FileInputVar *next);
      FileInputVar(const char *vname, double *varvar);
      FileInputVar(const char *vname, double *varvar, FileInputVar *next);
      FileInputVar(const char *vname, char *varvar);
      FileInputVar(const char *vname, char *varvar, FileInputVar *next);

      // utility functions
      FileInputVar *NextVar(void) {return nextvar; }
```


University of Cape Town

```
    int IsName(const char *nametocheck);

    void AttachData(int captured) { *(var.intptr) = captured; }
    void AttachData(double captured) { *(var.doubleptr) = captured; }
    void AttachData(const char *captured) { strcpy(var.charptr, captured); }
    void Tack(FileInputVar *pIivVarToTack);

  private:
    vartype vardatatype;

  public:
    vartype type(void) {return vardatatype; }

};


class InputFile : public ifstream
{
  private:
    FileInputVar *headVar;
    FileInputVar *tailVar;

  public:
    InputFile(const char* szName, int nMode = ios::in | ios::nocreate, int nProt = filebuf::sh_read );
    ~InputFile(void) { ifstream::~ifstream();
                       DeleteVarList();
    }

//  InputFile(const char *name, FileInputVar *first);

    InputFile& operator >>( FileInputVar *pIivNewVar );

    // we need to re implement the base class' extraction operators if we wish to
    // make them available.
    inline InputFile& operator >>( char* psz ) {
      return (InputFile&)istream:: operator >>( psz );
    }
    inline InputFile& operator >>( int& n ) {
      return (InputFile&)istream:: operator >>( n );
    }
    inline InputFile& operator >>( long& ln ) {
      return (InputFile&)istream:: operator >>( ln );
    }
    inline InputFile& operator >>( double& d ) {
```

University of Cape Town

```
        return (InputFile&)istream:: operator >>( c );
    }

    int AttachVar(FileInputVar *vardata) { if(vardata) headVar = vardata; return 0; }
    int ReadData(void);
    void DeleteVarList(void);

};
```

E.4.2 Implementation File: DatafileInput.cpp

```
/*****************************************************************************\
| FileName: DatafileInput.cpp                                                 |
| Summary:     Classes InputFile, FileInputVar                                |
| Description: Implements two classes for inputing data from a                |
|              structured, labeled data file                                  |
| Author:    Prakash Parbhoo                                                  |
| Date:      18/04/97
| Revision History:
\*****************************************************************************/


#include <string.h>
#include <iostream.h>
#include "DatafileInput.hpp"

FileInputVar::FileInputVar(const char *vname, int *varvar)
{
   if(vname) strcpy(varname, vname);
   var.intptr = varvar;
   vardatatype = varINT;
   nextvar = NULL;
}
FileInputVar::FileInputVar(const char *vname, int *varvar, FileInputVar *next)
{
   if(vname) strcpy(varname, vname);
   var.intptr = varvar;
   vardatatype = varINT;
   nextvar = next;
}
FileInputVar::FileInputVar(const char *vname, double *varvar)
{
   if(vname) strcpy(varname, vname);
```

```
    var.doubleptr = varvar;
    vardatatype = varDOUBLE;
    nextvar = NULL;
}
FileInputVar::FileInputVar(const char *vname, double *varvar, FileInputVar *next)
{
    if(vname) strcpy(varname, vname);
    var.doubleptr = varvar;
    vardatatype = varDOUBLE;
    nextvar = next;
}
FileInputVar::FileInputVar(const char *vname, char *varvar)
{
    if(vname) strcpy(varname, vname);
    var.charptr = varvar;
    vardatatype = varSTRING;
    nextvar = NULL;
}
FileInputVar::FileInputVar(const char *vname, char *varvar, FileInputVar *next)
{
    if(vname) strcpy(varname, vname);
    var.charptr = varvar;
    vardatatype = varSTRING;
    nextvar = next;
}


int FileInputVar::isName(const char *nametocheck)
{

    if(!strcmp(nametocheck, varname)) return 1;
    return 0;

}

void FileInputVar::Tack(FileInputVar *pfivVarToTack)
{

    // check if we are INSERTING
    if(nextvar) {
        // check if we are inserting a list
        if(pfivVarToTack->nextvar) {
            FileInputVar *pfivTemp = pfivVarToTack->nextvar;
            FileInputVar *pfivVarToTackTail = NULL;
            // find the tail of the list
```

University of Cape Town

```
            while(pfivTemp) {
               pfivVarToTackTail = pfivTemp;
               pfivTemp = pfivTemp->nextvar;
            }
            // set the tail to point to the one ahead of the insertion point
            pfivVarToTackTail->nextvar = nextvar;
         }
         // inserting a single item
         else {
            pfivVarToTack->nextvar = nextvar;
         }
      }
      // point the one behind the insertion point at the inserted data
      nextvar = pfivVarToTack;
}


InputFile::InputFile(const char* szName, int nMode, int nProt)
: ifstream(szName, nMode, nProt)
{

   headVar = NULL;
   tailVar = NULL;

}
/*
InputFile::inputFile(const char *fname, FileInputVar *first)
{

   if(fname) strcpy(datafilename, fname);
   firstvar = first;

}
*/
int InputFile::ReadData(void)
{

   int comch = ';';
   int spcch = ' ';
   int astch = '*';

   char buf1[512];
   char buf2[128];

   char *lpszStr1;
   char *lpszStr2;
```


University of Cape Town

```
int len1;

FileInputVar *tempvar;

// check to see that we have a variable list to fill, else return
if(!headVar) return -1;

// return with error code if the file is bad
if(bad()||(!good())) {
   cerr << "InputFile::ReadData: bad stream " << /* datafilename << */ endl;
   return 0;
}

int tempint;
double tempdouble;
char tempstr[512];

while(!eof()) {

   getline(buf1, sizeof(buf1), '\n');

   // check for ';'
   lpszStr1 = strchr(buf1, comch);
   // check for ' '
   lpszStr2 = strchr(buf1, spcch);

   // if either of these chars are first on the line, go to next line
   if( (lpszStr1 == buf1) || (lpszStr2 == buf1) ) continue;

   // if not, find which occurs first, space or ; and then extract chars up to that point
   if( (!lpszStr1) && (!lpszStr2) ) len1 = strlen(buf1);
   else len1 = ((lpszStr1>lpszStr2)?lpszStr2:lpszStr1) - buf1;

   // a variable name will start like this:  *VAR, so if len1 is only 1, we only have a *
   if(len1<2) continue;

   // check that buf1 starts with *
   if(buf1[0] != astch) continue;

   // now copy len1-1 characters into buf2
```

University of Cape Town

```
      len1--;

      strncpy(buf2, buf1+sizeof(char), len1);
      // we need to append a null char to buf2 (strncpy doesn't do this automatically)
      buf2[len1] = 0x0;

      // now look for the variable name in our list of variables
      tempvar = headVar;
      while(tempvar) {
         if(tempvar->IsName(buf2)) {   // we have the variable we want
            switch (tempvar->Type()) {
               case FileInputVar::varINT:
                     *this >> tempint;
                     tempvar->AttachData(tempint);
                     break;
               case FileInputVar::varDOUBLE:
                     *this >> tempdouble;
                     tempvar->AttachData(tempdouble);
                     break;
               case FileInputVar::varSTRING:
                     *this >> tempstr;
                     tempvar->AttachData(tempstr);
                     break;
            }
            tempvar = NULL;
         }
         else tempvar = tempvar->NextVar();
      }
   }

   return 1;
}

void InputFile::DeleteVarList(void)
{

   FileInputVar *pfivTemp = headVar;
   FileInputVar *pfivDel = NULL;
   while(pfivTemp) {
      pfivDel = pfivTemp;
      pfivTemp = pfivTemp->NextVar();
      delete pfivDel;
   }
   headVar = NULL;
   tailVar = NULL;
```

University of Cape Town

```
:

InputFile& InputFile::operator >>( FileInputVar *pfivNewVar )
{
   // check that we have a valid variable
   if(!pfivNewVar) return *this;

   if(tailVar) {
      tailVar->Tack(pfivNewVar);
      tailVar = pfivNewVar;
   }
   else {
      headVar = pfivNewVar;
      tailVar = pfivNewVar;
   }

   return *this;
}
```

University of Cape Town