Enhancements to SIMMOD: A Neural Network Post-processor to Estimate Aircraft Fuel Consumption

Phase I Final Report

NEXTOR Research Report RR-97-8

Antonio A. Trani (Principal Investigator)
Frank Wing-Ho (Graduate Research Assistant)

Department of Civil Engineering
Virginia Tech
Blacksburg, VA 24061

December 1997

Preface

This report documents research undertaken by the National Center of Excellence for Aviation Operations Research, under Federal Aviation Administration Research Grant Number 96-C-001. This document has not been reviewed by the Federal Aviation Administration (FAA). Any opinions expressed herein do not necessarily reflect those of the FAA or the U.S. Department of Transportation.

The authors are appreciative of the support provided by the FAA Office of Operations Research (ASD), and in particular, the guidance and leadership provided by Tony Vanchieri and Norm Fujisaki in funding this effort.

Table of Contents

Preface ii

Table of Contents iii

List of Figures vi

List of Tables viii

Executive Summary ix

CHAPTER 1 Introduction 1
- Background of the Problem 1
- Existing Fuel Consumption Model 2
- Neural Network and Fuel Consumption Modeling 2
- Research Objectives and Approach 3

CHAPTER 2 Modeling Background 5
- Aircraft Performance Based Fuel Consumption Model 5
- Physical Properties of Air 5
- International Standard Atmosphere (ISA) 6
- Speed of Sound and Mach Number 7
- Aircraft Performance 7
- Aircraft Fuel Consumption 9
- Existing SIMMOD Fuel Consumption Model 10
  - Drag Computation 11
- Neural Network Aided Fuel Consumption 15
- Introduction to Neural Networks 15
  - Functionality 16
  - Layers 16
  - Neurons 16
  - Connections 16
  - Weights and Biases 17
  - Recall 17
  - Transfer Functions 17
  - Learning 18
  - Neural Network Types 18
- Neural Learning Using Back-Propagation 19
- Learning Rule and Lavenberg Marquardt Optimization Algorithm 20

Steepest Descent Method 20
Newton's Method 21

CHAPTER 3 Methodology 25
Selection of the Testing Aircraft 25
Fuel Consumption Data Collection 25
Training the Neural Network 26
MATLAB Neural Network Basics 27
Initff 27
trainnlm 28
simuff 29
logsig 29
tansig 29
purelin 30
Selection of Training Algorithms 30
Design of the Appropriate Neural Network Topology 30
Implementation and Generalization of the Neural Network 32
Description of the Neural Network Aided Fuel Consumption Model 32
Correlating Results 35
Statistical Analysis Testing Procedures 35
Implementation of the Outputs to an Actual Flight Trajectory 36

CHAPTER 4 Neural Network Model Development 41
Training of Weight Matrix For Different Flight Phases 41
Testing Preliminary Results Using Statistics 43
Implementation of Neural Network to Actual Flight Trajectory 44

CHAPTER 5 Discussion of Results 49
Training Results 49
Testing Results 58
Correlation of Neural Network Fuel Consumption Results 60
SIMMOD and SIMMOD 2000 Implementation Issues 65
Neural Network Fuel Burn Correction in the Presence of Winds 67

CHAPTER 6 Conclusions and Recommendations 71
Conclusions 71
Remarks and Recommendations 72
Remarks 72
Fuel Efficient Flight Path 72
Neural Network Model Extensions 73

Recommendations 73

Bibliography 75

APPENDIX A Neural Network Templates Source Code 77
Take-off and Climb Out Fuel Estimation 78
Climb Performance Estimation 79
Cruise Specific Air Range 82
Neural Network Testing Program 87
Testing main program 87
Statistical analysis 94
Main Program to Calculate Fuel Consumption 101
Main Program 101
Cruise Cubroutine 112
Descent Subroutine 115

APPENDIX B Neural Network Trained Matrices and Bias Vectors 121
Climb Neural Network Matrices 121
First Layer Climb Fuel Weight Matrix (8 x 4) 121
First Layer Bias Climb Fuel Vector (8 x 1) 122
Second Layer Climb Fuel Weight Matrix (8 x 8) 122
Second Layer Bias Climb Fuel Vector (8 x 1) 122
Third Layer Climb Fuel Weight Matrix (1 x 8) 122
Third Layer Bias Climb Fuel Vector (1 x 1) 122
Cruise Neural Network Matrices 123
First Layer Cruise Fuel Weight Matrix (8 x 3) 123
First Layer Bias Cruise Fuel Vector (8 x 1) 123
Second Layer Cruise Fuel Weight Matrix (8 x 8) 123
Second Layer Bias Cruise Fuel Vector (8 x 1) 124
Third Layer Cruise Fuel Weight Matrix (1 x 8) 124
Third Layer Bias Cruise Fuel Vector (1 x 1) 124
Descent Neural Network Matrices 124
First Layer Descent Fuel Weight Matrix (12 x 4) 124
First Layer Bias Descent Fuel Vector (12 x 1) 125
Second Layer Descent Fuel Weight Matrix (12 x 12) 125
Second Layer Bias Descent Fuel Vector (12 x 1) 125
Third Layer Descent Fuel Weight Matrix (1 x12) 125
Third Layer Bias Descent Fuel Vector (1 x 1) 126

List of Figures

Figure 2.1 Velocity Profile Over an Aerodynamic Surface. 6
Figure 2.2 Typical Forces Acting on the Aircraft. 8
Figure 2.3 Aircraft Characteristic Drag Polar. 9
Figure 2.4 Turboprop and Turbofan Engine Characteristics. 10
Figure 2.5 General Architecture of a Neural Network. 16
Figure 2.6 A Single Neuron Diagram. 17
Figure 2.7 Basic Neural Network Transfer Functions. 18
Figure 2.8 Backpropagation Cycle. 20
Figure 3.1 Fokker F-100 Aircraft Layout and General Characteristics. 26
Figure 3.2 General Three Layer Neural Network. 33
Figure 3.3 Fuel Burn for Takeoff and Climbout. 34
Figure 3.4 Fuel Estimation Procedure for Takeoff and Climbout. 34
Figure 3.5 Fuel Estimation Procedure for Climb Segment. 34
Figure 3.6 Fuel Estimation Procedure for Cruise Segment. 35
Figure 3.7 Selected Neural Network Architecture. 39
Figure 4.1 Sample Input Vector P for Training Purposes. 42
Figure 4.2 Neural Network Training Process. 43
Figure 4.3 Neural Network Testing Procedure. 45
Figure 4.4 Flight Trajectory Implementation with Waypoints. 45
Figure 4.5 Possible Aircraft Climb Procedures. 47
Figure 4.6 Flow Chart for Neural Network Aided Fuel Consumption Model. 48
Figure 5.1 Climb Fuel Information (source: Fokker 100 Performance Manual). 51
Figure 5.2 Specific Air Range Information (source: Fokker 100 Performance Manual). 52
Figure 5.3 Descent Fuel Performance (source: Fokker 100 Performance Information). 53
Figure 5.4 Two-Dimensional View of Climb Fuel Database. 54
Figure 5.5 Three Dimensional View of Training Climb Fuel Database. 54
Figure 5.6 Two-Dimensional View of Climb Distance Database. 55
Figure 5.7 Three-Dimensional View of Climb Distance Database. 55
Figure 5.8 Two Dimensional View of Cruise Segment Specific Range Database. 56
Figure 5.9 Three Dimensional View of Cruise Segment Specific Range Database. 56

Figure 5.10	Two Dimensional View of Descent Fuel Database.	57
Figure 5.11	Three Dimensional View of Descent Fuel Database.	57
Figure 5.12	Estimated and Actual Climb Fuel Results.	61
Figure 5.13	Climb Fuel Error Histogram.	61
Figure 5.14	Climb Distance Correlation of Results.	62
Figure 5.15	Climb Distance Estimation Errors.	62
Figure 5.16	Specific Range Generalization Results.	63
Figure 5.17	Specific Range Error Histogram.	63
Figure 5.18	Descent Fuel Predicted vs. Actual.	64
Figure 5.19	Descent Fuel Error Histogram.	64
Figure 5.20	Frequency Distributions for Actual and Computed Descent Fuel.	65
Figure 5.21	Sample Flight Plan Profile.	66
Figure 5.22	Current SIMMOD Fuel Burn Post-processor Input/Output File Structure.	68
Figure 5.23	New SIMMOD Fuel Burn Post-processor Input/Output File Structure.	69
Figure 5.24	SAR Parameter Corrected for Enroute Winds.	69
Figure B.1	Climb Fuel Consumption Neural Network.	121
Figure B.2	Cruise Fuel Consumption Neural Network.	123
Figure B.3	Descent Fuel Consumption Neural Network.	124

List of Tables

Table 3.1.	Comparison of Optimization Algorithms	30
Table 3.2.	Results of the Topology Sensitivity Study	31
Table 3.3.	Summary of Neural Network Training and Testing	36
Table 4.1.	Summary of Neural Network Input and Output Parameters.	44
Table 5.1.	Training Data Sets.	50
Table 5.2.	Neural Network Testing Data Sets.	59
Table 5.3.	Summary of Errors for All Phases of Flight.	59
Table 5.4.	Flight Plans Used in the Correlation of the Neural Network Model (ISA+10).	67
Table 5.5.	SIMMOD Fuel Burn Postprocessor Routines.	70

Executive Summary

This report documents research undertaken by the National Center of Excellence for Aviation Operations Research to enhance SIMMOD - the FAA airspace and airfield simulation model. The existing SIMMOD fuel consumption model based on aircraft performance parameters was studied. Advantages and disadvantages of this model were reviewed. A representative neural network aided fuel consumption model was developed using data given in the aircraft performance manual. The data used in this study was applicable to the Fokker 100 aircraft powered by Rolls-Royce Tay 650 engines. Nevertheless, the methodology can be extended to any type of aircraft including piston and turboprop type vehicles with confidence.

The neural network was trained to estimate fuel consumption of an example aircraft. Results were compared to the actual performance provided in the aircraft performance manual and found to be accurate for possible implementation in SIMMOD and other fast-time simulation programs.

The following conclusions are derived from this analysis:

1. The disadvantage of the existing SIMMOD advanced fuel consumption model is that the information required to create the data base for this particular algorithm is very difficult to obtain. This fact has been without doubt a constraint in the expansion of the fuel burn database in SIMMOD.

2. Results obtained from the neural network aided fuel consumption model show that a neural network with proper training is an accurate and efficient mean to calculate fuel consumption of fixed wing aircraft. The added benefit of this approach is that only the flight performance manual of the aircraft is needed to characterize the complete fuel burn behavior of the vehicle throughout its flight envelope.

3. A neural network is found to be a viable alternative in fuel consumption estimating application. The computational results obtained in this paper indicate that the neural network approach can be implemented in fast-time simulation models such as SIMMOD, RAMS, TAAM and future products where flight trajectories are described in terms of waypoints. Moreover, neural networks can approximate with good accuracy the complete performance of the vehicle (including climb, cruise, maneuvering, and decent) and simplify the implementation of realistic aircraft models without compromising aircraft sensititive data that is seldom made public.

CHAPTER 1 Introduction

1.1 Background of the Problem

Addressing internal organization needs to execute airport and airspace capacity studies the Federal Aviation Administration (FAA) developed SIMMOD - the airspace and airfield simulation model. This model has been widely used by internal groups within the FAA and over 200 users worldwide. SIMMOD is regarded by many as one of the few airspace and airfield simulation models that have been calibrated against actual scenarios and is regarded as a good tool for airfield and airspace simulation analysis.

Over the years SIMMOD has been both acclaimed and criticized. In the past seven years the FAA has performed sixteen airport/airspace capacity enhancement studies where SIMMOD played a major role (i.e., was the modeling tool of choice) in the analysis (FAA, 1996). In this study all SIMMOD projects combined yield an expected delay savings of 4,703 million dollars in the horizon year (2003-2005). If one factors the life cycle cost savings of all major projects one can easily arrive to figures an order of magnitude higher.While it is not possible to know how much of the benefits claimed by each study are due to the modeling tool itself one could easily argue that SIMMOD brought to these studies a level of airspace and airfield simulation detail that could have been difficult to achieve with other simulation tools.

It is important to note that SIMMOD has been used in numerous airport and airspace studies sponsored by airport authorities as well. These studies have been carried out over the years by engineering consulting firms and claim several hundred million dollars in savings to users and operators of the system. In retrospect, SIMMOD has served and continues to serve an important role in airport/airspace planning studies. The FAA and industry do not have access to other gate-to-gate model like this without paying a high premium. While the technology of this model is effectively showing its age it is important to realize that many improvements and future developments to this model can be made with modest resources. This study tries to accomplish a small improvement to the model. An improvement that could perhaps be more relevant in the future when natural resource consumption become as prominent variable in airport and airspace planning activities.

There are several functions in SIMMOD that perhaps receive little attention from various users mainly because the data to support such post-processor functions is scarce. One of such functions is fuel burn. SIMMOD utilizes a fuel consumption post processor that computes the fuel consumption of an aircraft given a flight profile (Collins, 1982). The advanced fuel burn model-MOD 830725 developed by Bela P. Collins at the MITRE Corporation in the early 1980’s is the primary algorithm behind the fuel burn computations. The theme for this research paper is to look for possible ways to enhance the fuel consumption estimation process by reducing the complexity of the existing fuel-burn model, make the methodology easier to implement using current aircraft data, making the technology more transportable to other models and increasing its accuracy. Moreover, this research attempts to establish a methodology that can be readily applied to any flight vehicle contained in the SIMMOD data base.

1.2 Existing Fuel Consumption Model

SIMMOD’s existing fuel consumption model utilizes the energy balance relation to estimate the fuel consumption of an aircraft. This relation is based on aerodynamics and engine characteristics of an aircraft [Collins 1980]. Since the model is basically a combination of different performance fitted curves, the major task in using this model is to determine all the coefficients involved in describing the nonlinear behavior of the aircraft’s performance curves. However, the information required to determine these coefficients are usually considered proprietary by most aircraft production companies and cannot be obtained from them. Instead, flight testing and wind tunnel testing are used as sources of information. Unfortunately, the cost of this kind of testing is extremely high. The motivation of this thesis is to use information given directly from the aircraft manufacturer to predict fuel consumption accurately and efficiently. This information is contained in a handbook known as the “pilot’s flight manual”.

1.3 Neural Network and Fuel Consumption Modeling

The neural network approach to aviation fuel consumption application is quite simple in principle. The aircraft fuel consumption data from the flight performance manual of individual aircraft are presented to the network. The network, by an iterative process, self-organizes and generalizes its own performance data (i.e., strenghts of connections between various elements of the network denoted as weights and biases are determined). This process is referred to as “network learning”. When sufficient amount of data are presented to the network, the network will becomes a ”trained network” capable of estimating the performance of aircraft in fuel consumption. Neural network training techniques utilized are presented in Chapters 3 and 4. Actual implementation results are presented in Chapter 5.

1.4 Research Objectives and Approach

The objectives of this research project are:

1. To investigate possible alternatives to fuel consumption modeling in SIMMOD and other fast-time simulation programs,
2. To use the information provided in the aircraft performance flight manual, along with neural network methodology to develop an accurate fuel consumption model, and
3. To determine whether neural network should be considered as a viable alternative in fuel consumption estimating applications

To attain these objectives, the existing SIMMOD fuel burn model based on energy balance principles is studied and reviewed. The advantages and disadvantages of this particular model are reviewed. A representative neural network model is then selected by comparing results obtained from different neural network model topologies (see Chapter 5).

CHAPTER 2 Modeling Background

2.1 Aircraft Performance Based Fuel Consumption Model

The aircraft performance based fuel consumption model, also known as the Advanced Fuel Burn Model - MOD 830725 (AFBM), was developed by B. P. Collins of the MITRE Corporation. This is a fuel consumption evaluation model based on an energy balance concept. The energy balance relation is defined as the aircraft travels along a path its energy gains and losses over a distance will be maintained. This concept has been used to develop a core equation that models the energy performance of aircraft. The equation is adapted to a specific aircraft by using constants that represent the relationship of lift to drag and thrust to fuel flow. In order to gain an insight of this model, some important aerodynamic features will be presented before describing this model explicitly.

2.1.1 Physical Properties of Air

The physical properties of air included are:

- temperature (T)
- pressure (P)
- density (ρ)
- viscosity (υ)

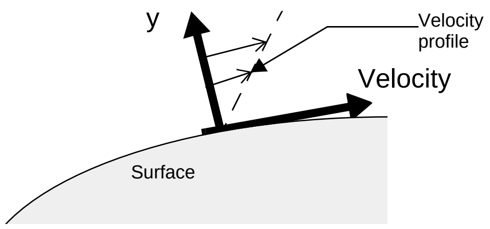
Temperature is a measure of the average kinetic energy of the particle of gas. If KE is the mean molecular kinetic energy, then temperature is given by $KE = \frac{3}{2}kT$, where k is the Boltzmann constant. Common units of temperature are Kelvin (K), degree Celsius $(°C)$, degree Rankine $(°R)$, and degree Fahrenheit $(°F)$. Pressure is the normal force per unit area exerted on a surface due to the time rate of change of momentum of the gas molecules impacting the surface. Common units of pressure are Pascal

(Pa) and pounds per square inch (psi). The density of a substance is the mass of that substance per unit volume. Common units of density are kilogram per cubic meter (kg/m^3) and slug per cubic foot (slug/ft^3). Viscosity of a fluid is defined as the ratio of viscous stress to the velocity gradient. If τ is the viscous stress then

$$\tau = \upsilon\left(\frac{d(velocity)}{d(y)}\right) \qquad (2.1)$$

where υ is the viscous coefficient of a fluid. Figure 2.1 shows a velocity profile of a fluid moving over a surface. Common units of viscosity are kilogram per meter per second (kg/m/sec) and slug per feet per second (slug/ft/sec).

Figure 2.1 Velocity Profile Over an Aerodynamic Surface.



A perfect gas is one where intermolecular forces can be neglected. For a perfect gas, the equation which relates pressure, density and temperature is

$$p = \rho RT \qquad (2.2)$$

where R is the specific gas constant.

2.1.2 International Standard Atmosphere (ISA)

The earth's atmosphere is a dynamically changing system, constantly in a state of flux. The pressure, density, and temperature of the atmosphere in this planet depend on altitude, longitude and latitude, time of the day, and many other factors. To take all the above factors into account during evaluation of flight performance is impractical. Therefore, the international standard atmosphere (ISA) is defined in order to relate flight test, wind tunnel results, and general aircraft design and performance to a common reference.

A standard atmosphere in common use is the 1959 ARDC Model atmosphere. ARDC standards for the U.S, Air-force's previous Air Research and Development Command, which is now the Air Force Systems Command (McCormick 1995).

2.1.3 Speed of Sound and Mach Number

Speed of sound plays a fundamental role in any aircraft performance evaluation. Calculation of the speed of sound is based on the continuity equation and momentum equations. For the steady incompressible flow of frictionless fluid, the relationship of density ρ and velocity V in a stream tube of varying area A can be written as:

$$\rho_1 A_1 V_1 = \rho_2 A_2 V_2 \tag{2.3}$$

The momentum equation is an expression to relate the rate of change momentum to the force based on Newton's second law of motion, force is equal to the mass times the velocity change with respect to time. For a fluid element, after some algebraic manipulations, the differential change in pressure can be written as:

$$dp = -\rho V dV \tag{2.4}$$

The above equation is designated the momentum equation. Based on the above two equations and the assumption that the flow through a sound wave is isentropic (no heat addition), the speed of sound a can be derived, and given by,

$$a = \sqrt{\left(\frac{dp}{d\rho}\right)_{isentropic}} = \sqrt{\gamma R T} \tag{2.5}$$

where γ is the specific heat ratio and R is the specific gas constant.

The speed of sound leads to another, vital definition for high speed gas flows, namely, the Mach number. By definition the Mach number M is the velocity of the flow divided by the speed of sound:

$$M = \frac{V}{a} \tag{2.6}$$

M is one of the most useful quantities used in aerodynamic theory.

2.1.4 Aircraft Performance

This section presents some basic elements of airplane performance.

Figure 2.2 shows an aircraft in climbing flight. The direction of motion of the aircraft is inclined at an angle θ with respect to the horizontal. The flight path direction and the relative wind are along the same line. The angle formed between the mean chord line and the flight path line is the angle of attack α with respect to the flight path direction. There are four physical forces acting any flight vehicle:

1. Lift L, which is perpendicular to flight path direction.

2. Drag D, which is parallel to the flight path direction.

3. Weight W, which acts vertically toward the centre of the earth.

4. Thrust T, which in general is inclined at angle α with respect to the flight path direction.

By summing horizontal and vertical forces with respect to the flight path using Newton's second law of motion, the following can be obtained (Anderson, 1989):

$$T\cos(\alpha) - D - W\sin(\theta) = \frac{W}{g}\frac{dV}{dt} \tag{2.7}$$

$$L + T\sin(\alpha) - W\cos(\theta) = m\frac{V^2}{RC} \tag{2.8}$$

where g is the gravitation constant of the earth and RC is the radius of the curvature of the flight path. The above expressions are known as equations of motion for an airplane in two-dimensional translational flight. Along with these equations, consider level unaccelerated flight. Level flight means that the flight path is along the horizontal, that is, $\theta = 0$. Unaccelerated flight implies that the right hand side of the above equations are equal to zero. Therefore the equations of motion can be reduced to:

$$T\cos(\alpha) = D \tag{2.9}$$

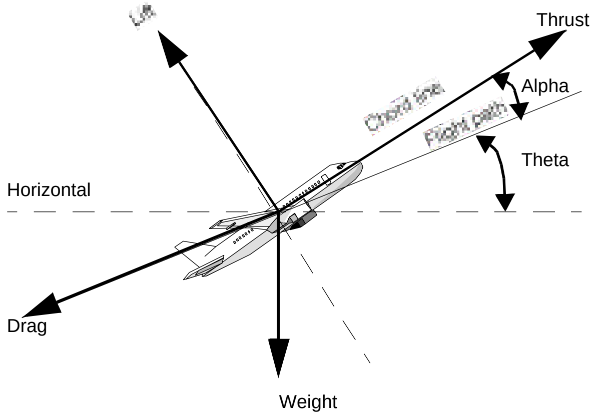
$$L + T\sin(\alpha) = W \tag{2.10}$$

Furthermore, for most conventional airplanes, α is small enough that $\cos(\alpha) \approx 1$ and $\sin(\alpha) \approx 0$. Thus:

$$T = D \tag{2.11}$$

$$L = W \tag{2.12}$$

Figure 2.2 Typical Forces Acting on the Aircraft.



The results above show that during a level unaccelerated flight, the aerodynamic drag is balanced by the weight of the airplane: this result while almost trivial, is extremely useful in the estimation of fuel consumption.

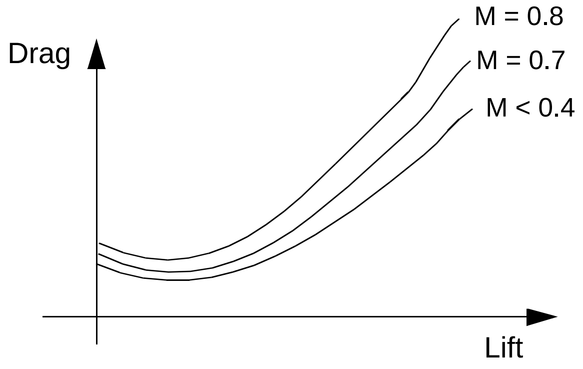
2.1.5 Aircraft Fuel Consumption

Fuel consumption of an aircraft is a function of the following variables:

- aerodynamics characteristics
- engine type
- mission profile

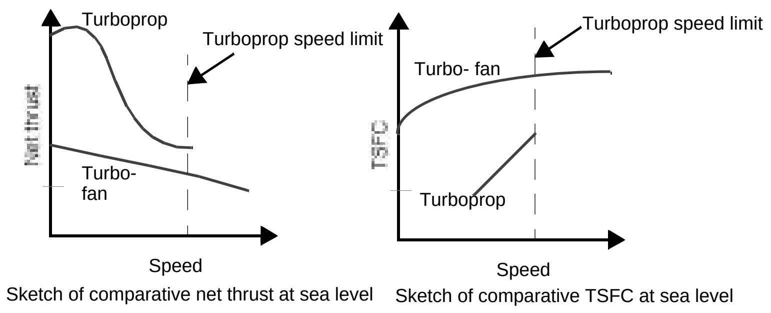
Aerodynamic characteristics depend on individual aircraft design variables. The two major characteristic concerning fuel consumption estimation are lift and drag produced during flight. Generally, lift of an aircraft is a function of the wing-geometric and drag is a function of the entire aircraft. Relationship between these two characteristics are usually presented in a drag polar. Figure 2.3 is a graphical presentation of the drag polar.

Figure 2.3 Aircraft Characteristic Drag Polar.



The powerplant is the only part of the flight vehicle that is responsible for the physical process of fuel consumption. There are two types of engines utilized by civilian aircraft: turbo-fan and turbo-propeller. Figure 2.4 shows the fundamental performance differences between these two types of engines (Torenbeck, 1982). The specific fuel consumption for a turbo-fan engine is expressed as a thrust specific fuel consumption (TSFC). Typical units of TSFC are pounds of fuel per hour per pound of thrust.

Figure 2.4 Turboprop and Turbofan Engine Characteristics.



2.1.6 Existing SIMMOD Fuel Consumption Model

An existing fuel burn model that evaluates fuel consumption is the Advanced Fuel Burn Model - MOD 830725 (AFBM), developed by Bela P. Collins of the MITRE Corporation (Collins, 1982).

This particular aircraft fuel burn evaluation model is based on an energy balance concept. This concept has been used to develop a core equation that models the energy performance of aircraft. The equation is adapted to a specific aircraft by using constants that represent the relationship of lift to drag and thrust to fuel flow. Multi-variate curve fitting techniques are used in conjunction with performance data to derive the aircraft specific constants. Aircraft performance limits are represented by empirical relationships that also utilize aircraft specific constants.

<u>Assumptions and Definitions</u>

As mentioned earlier, the fuel consumption algorithm was designed to incorporate pre-determined coefficients associated with aircraft performance, and dynamic inputs of the flight profile of the aircraft. In order to simplify the computational procedures the following assumptions were made:

- Weight changes over increments will not affect fuel burn calculations.
- Flight path angles are small:

$$\cos(\varphi) \approx 1 \tag{2.13}$$

- Velocity and altitude changes within an increment are linear.
- Energy Balance Approach (Collins, 1984).
- The fuel burn equation has been derived from basic principles of conservation of energy. Energy balance requires:
- (energy change) = (energy in) - (energy loss)

or, when expressed in terms of aircraft flight:

(change in potential energy + change in kinetic energy) = work done by the thrust - work done by the drag

where,

work done by the thrust = thrust force x distance along the vector

work done by the thrust = drag force x distance along the vector

Therefore, the energy balance can be written as:

(change in potential energy + change in kinetic energy) = thrust force x distance along the vector - drag force x distance along the vector

$$\Delta PE + \Delta KE = (F_n \times d) - (D \times d) \tag{2.14}$$

Solving this expression for thrust F_n

$$F_n = \frac{\Delta KE}{d} + \frac{\Delta PE}{d} + D \tag{2.15}$$

where,

$$\Delta KE = \frac{W({V_2}^2 - {V_1}^2)}{2g} \text{ and } \Delta PE = W(h_2 - h_1) \tag{2.16}$$

The distance along the velocity vector can be expressed as

$$d = \bar{V}t \tag{2.17}$$

where,

$$\bar{V} = \frac{(V_1 + V_2)}{2} \tag{2.18}$$

Drag Computation

Since the aircraft is assumed to operate at a small flight path angle, according to basic aircraft performance the lift required is equal to the weight for level flight. In addition, drag and lift are usually expressed as non-dimensional coefficients C_D and C_L, and can be written as:

$$Lift = Weight = \frac{1}{2}\rho V^2 S_w C_L \tag{2.19}$$

Solving for C_L,

$$C_L = \frac{W}{\frac{1}{2}\rho V^2 S_w} \tag{2.20}$$

Similarly, the drag can be written as,

$$D = \frac{1}{2}\rho V^2 S_w C_D \tag{2.21}$$

where,

$$C_D = C_{D,o} + f(C_L, e, AR) \tag{2.22}$$

Here, e is Oswald's efficiency factor and AR is the aspect ratio of the wing. Expanding the above equation, the total drag coefficient can be expressed as:

$$C_D = \sum C_{D_{P_{min}}} + C_{D_t} + C_{D_r} + C_{D_i} + C_{D_{P_{C_L}}} + C_{D_C} \tag{2.23}$$

where,

$$\sum C_{D_{P_{min}}} \tag{2.24}$$

is the summation of the minimum profile drag of the individual aircraft components, for turbulent attached flow. In the previous expressions the following nomenclature has been used.

C_{D_t} = Drag required to trim the aircraft about its centre of gravity.

$C_{D_{int}}$ = Drag due to interference between components.

C_{D_r}= Drag due to surface distributed roughness, steps, gaps, and significant protuberance.

C_{D_i} = Wing vortex induced drag at a given wing lift coefficient corresponding to the spanwise distribution of lift, and is the net effect of elliptic and non-elliptic contributions.

$C_{D_{P_{C_L}}}$ = Net aircraft lift-dependent profile drag, including major contributions from the wing and fuselage, and other components.

C_{D_C} = Compressibility drag, which includes subcritical drag creep, wave drag, and shock-induced separation drag.

Note that the first four terms are not lift dependent, while the remaining two terms are both lift and Mach dependent. In addition, the last term accounts for compressibility drag rise. Along with this information, the following functions are utilized to present the nondimensional drag coefficient (C_D) that consists of a nonlinear drag polar with sensitivity to Mach number:

$$C_D = M_a + M_b C_L^{\ 2} + M_b C_L^{\ 4} \tag{2.25}$$

where M_a, M_b,and M_c are functions of Mach number. Hence, M_a represents the first four terms in Equation 2.2.3, $M_bC_L^{2}$ represents the fifth and sixth terms in Equation 2.23 and $M_bC_L^{4}$ represents the last term. These three M_i functions are defined as follows,

$$M_a = C_1 + C_2\Gamma^2 + C_3\Gamma^4 \tag{2.26}$$

$$M_a = C_4 + C_5\Gamma + C_6\Gamma^2 + C_7\Gamma^3 + C_8\Gamma^4 \tag{2.27}$$

$$M_c = C_9 + C_{10}\Gamma + C_{11}\Gamma^2 + K_{12}\Gamma^3 \tag{2.28}$$

where,

$$\Gamma = \frac{1+\bar{M}}{1-\bar{M}} \tag{2.29}$$

and C_i are drag constants

According to basic aerodynamics, drag will increases dramatically, once the critical Mach number is reached. Therefore, in order to account for this kind of behavior higher order terms are employed.

Now the drag function shown in Equation 2.21 can be re-written as,

$$D = \frac{1}{2}\rho V^2 S_w (M_a + M_b {C_L}^2 + M_c {C_L}^4) \tag{2.30}$$

where the M functions have been previously defined.

The energy balance equation (see Eq. 2.15) now becomes,

$$F_n = \frac{\Delta KE}{d} + \frac{\Delta PE}{d} + \frac{1}{2}\rho V^2 S_w (M_a + M_b {C_L}^2 + M_c {C_L}^4) \tag{2.31}$$

by substituting for Eq. 2.20,

$$F_n = \frac{\Delta KE}{d} + \frac{\Delta PE}{d} + \frac{1}{2}\rho V^2 S_w M_a + M_b\left(\frac{W}{\frac{1}{2}\rho V^2 S_w}\right)^2 + M_c\left(\frac{W}{\frac{1}{2}\rho V^2 S_w}\right)^4 \tag{2.32}$$

Here, the energy balance equation consists of some specific constants and aircraft variables, such as acoustic speed, altitude and weight. Aircraft configuration changes, such as flaps and landing gear deployment can affect the drag polar. To account for these changes the following functions should be utilized:

$$R_1 = 1 + (CF)(CC_1) + (CC_2)(GF) + (CC_3)(FA)(GF) + (CC_4)(FA) + \left(CC_5\right)(FA)^2 + (CC_6)(FA)^3 \tag{2.33}$$

$$R_2 = 1 + (CF)(CC_7) + (CC_8)(GF) + (CC_9)(FA)(GF) + (CC_{10})(FA) + (CC_{11})(FA)^2 + (CC_{12})(FA)^3 \tag{2.34}$$

$$R_3 = 1 + (CF)(CC_{13}) + (CC_{14})(GF) + (CC_{15})(FA)(GF) + (CC_{16})(FA) + (CC_{17})(FA)^2 + (CC_{18})(FA)^3 \tag{2.35}$$

The final drag equation becomes:

$$F_n = \frac{\Delta KE}{d} + \frac{\Delta PE}{d} + \frac{1}{2}\rho V^2 S_w M_a R_1 + R_2 M_b\left(\frac{W}{\frac{1}{2}\rho V^2 S_w}\right)^2 + R_3 M_c\left(\frac{W}{\frac{1}{2}\rho V^2 S_w}\right)^4 \tag{2.36}$$

Since it is desired to compute fuel flow and finally fuel burn over an increment of time, the thrust must be translated into fuel flow.

The fuel burn of an aircraft basically depends on airframe drag, engine specific fuel consumption, distance of the route to be flown, vertical flight path and aircraft weight. The central factor in engine development is to minimize thrust specific fuel consumption (TSFC), or simply called, the specific fuel consumption (SFC). SFC is a measure of engine efficiency, defined as fuel flow rate in pounds per hour divided by engine thrust in pounds of force (lb/hr/lb). The fuel consumption of an aircraft depends on the individual power plant. For turbojet and turbofan engine the fuel flow is a function of two factors, altitude and velocity.

An empirical relationship that has been found to describe the turbojet engine is shown below:

$$\hat{\mu} = \eta_1 + \eta_2 F_n + \eta_3 \eta F_n^2 \tag{2.37}$$

where the η functions are,

$$\eta_1 = k_1 + k_2\bar{M} + k_3\bar{h} + k_4\bar{M}\bar{h} + k_5\bar{h}^2 + k_6\bar{M}\bar{h}^2 \tag{2.38}$$

$$\eta_2 = [k_7 + k_8\bar{M} + k_9\bar{h} + k_{10}\bar{M}\bar{h} + k_{11}\bar{h}^2 + k_{12}\bar{M}\bar{h}^2](N \times 10^4)^{-1} \tag{2.39}$$

$$\eta_3 = [k_{13} + k_{14}\bar{M} + k_{15}\bar{h} + k_{16}\bar{M}\bar{h} + k_{17}\bar{h}^2 + k_{18}\bar{M}\bar{h}^2](N \times 10^4)^{-2} \tag{2.40}$$

According to Collins (1984), The above relationships do not have a direct physical tie to the systems that exist within an engine. The functions were chosen because their behavior is similar to reality. Now the energy balance equation shown in Eq. 2.36 can be substituted into Eq. 2.37. The advanced fuel burn model MOD830725 core equation is written as follows:

$$\begin{aligned}\hat{\mu} = {} & \eta_1 + qS_w M_a R_1 + \frac{R_2\eta_2 M_b W^2}{qS_w} + \frac{R_3\eta_2 M_c W^4}{(qS_w)^3} + \frac{\eta_2(V_2 - V_1)W}{gt} + \frac{(h_2 - h_1)W}{t\bar{V}} + \frac{\eta_3 W(V_2 - V_1)}{gt} + \\ & \frac{(h_2 - h_1)W^2}{t\bar{V}} + \frac{2(V_2 - V_1)W}{gt} + \frac{(h_2 - h_1)W}{t\bar{V}} + \eta_3 M_a qS_w R_1 + \frac{R_2\eta_3 M_b W^2}{qS_w} + \\ & \frac{R_3\eta_3 M_c W^4}{(qS_w)^3} + \eta_3 M_a^2 (qS_w)^2 R_1 + 2R_1\eta_3 M_a M_b W^2 R_2 + \frac{2R_1\eta_3 M_a M_c W^4 R_3}{(qS_w)^2} + \\ & \frac{R_2\eta_3 M_b W^2}{qS_w} + \frac{R_3\eta_3 M_c W^4}{(qS_w)^3} + \eta_3 M_a^2 (qS_w)^2 R_1 + 2R_1\eta_3 M_a M_b W^2 R_2 + \frac{2R_1\eta_3 M_a M_c W^4 R_3}{(qS_w)^2} + \\ & \frac{\eta_3 M_b^2 W^4 R_2^2}{(qS_w)^2} + \frac{2R_2\eta_3 M_b M_c W^6 R_3}{(qS_w)^4} + \frac{\eta_3 M_c^2 W^8 R_3^2}{(qS_w)^6}\end{aligned} \tag{2.41}$$

The core equation combines all aerodynamic properties and engine data in a very compact form. The advantage of using this model is that it can be attached to any flight trajectory generation program for a particular aircraft. Unfortunately, the coefficients in this model must be obtained through several source (including wind tunnel data and/or flight testing) that in many instances are proprietary in nature thus making their inclusion in models like SIMMOD difficult. A further complication of this model is the complexity of the task of modeling low and high speed aircraft performance with the same core equation. Aircraft aerodynamics and engine properties at low Mach number and high Mach number are substantially different. Attempts to describe the behavior of aircraft properties and engine behaviors at high Mach numbers is a complicated task. To determine and justify the above required coefficients, flight testing and wind tunnel testing must be conducted for each aircraft in the market under different flight conditions. Therefore, this model is somewhat impractical.

2.2 Neural Network Aided Fuel Consumption

Using neural networks to calculate fuel consumption rates of aircraft was have been demonstrated before (Schilling, 1996). In his work Schilling used Collin's model to calculate the fuel burn, and then used these results to train the neural network assisted model. Schilling proved the potential of using a neural network to estimate fuel consumption of aircraft. The idea of this research project is to expand Schilling's idea to the level that the calculation of fuel consumption need not involve any flight testing or experiments, i.e. a model independent from Collin's model was developed.

All existing aircraft in the civil aviation industry are tested by the aircraft manufacturer during the flight certification process (i.e. FAR Parts 23, 25 or 27). Unfortunately, many test results of these aircraft are confidential. The flight manual is a common source of information for individual aircraft. Fuel consumption information contained in the flight manual is the most reliable source of data. The main goal of this research is to make use of this common information contained in the flight manual to predict fuel consumption of an aircraft. To make use of the flight manual, a table look-up function must be constructed. Since data given are non-linear and discontinuous, one way to create this function is by using the neural network. The following sections contain some background and fundamental information of neural network in considerable detail. Chapters 3 and 4 contain descriptions on how the neural network is implemented in fuel consumption estimation.

2.2.1 Introduction to Neural Networks

Artificial neural networks (or neural networks for short) are computational models broadly inspired by the organization of the human brain. The most important features of a neural network are its abilities to learn, to associate, and to be error tolerant. Unlike conventional problem solving algorithms, neural networks can be trained to perform a particular task. This is done by presenting the system with a representative set of examples describing the problem, namely pairs of input and output samples; the neural network will then extrapolate the mapping between input and output data. After training, the neural network can be used to recognize data that is similar to any of the examples shown during the training phase. The neural network can even recognize incomplete or noisy data, an important feature that is often used for prediction, diagnosis or control purposes. Furthermore, neural networks have the ability to self-organize, therefore enabling segmentation or coarse coding of data.

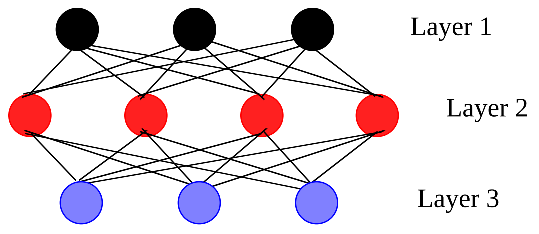
Functionality

At the most abstract level, a neural network can be thought of as a black box, where data is fed in on one side, processed by the neural network which then produces an output according to the supplied input (Candill 1992). Although a neural network can usually process any kind of data, e.g. qualitative or quantitative information, the data fed into the neural network should be preprocessed (e.g. filtered, transformed) to enable faster training and better performance. In fact, the selection, preprocessing, and coding of information is one of the main issues to deal with when working with neural networks.

Layers

A closer look at the black box reveals that its interface to the outside world consists of an input layer and an output layer of neurons. The neurons are the processing units within the neural network and are usually arranged in layers (Alexander, 1989). The information is propagated through the neural network layer by layer, always in the same direction. Besides the input and output layer there can be other intermediate layers of neurons, which are usually called hidden layers. Figure 2.5 illustrates the simplified architecture of a neural network.

Figure 2.5 General Architecture of a Neural Network.



Neurons

A neuron collects information from all preceding neurons relative to the flow of the information and propagates its output to the neurons in the following layer. The output of each preceding neuron (a_{i-1}) is modulated by a correspondent weight (w_i) and bias (b_i) before affecting the activity of the neuron. This process is realized by the formula $n_i = w_i a_{i-1} + b_i$, where n_i represents the activity of the neuron. This activity is then modified by transfer function f() and becomes the final output $a_i = f(n_i) = f(w_i a_{i-1} + b)$ of the neuron (Dayhoff, 1990). This signal is then propagated to the neurons of the next layer. Figure 2.7 depicts this process.

Connections

Connections are the paths between neurons where all the information flows within a neural network. Very often the neurons of two succeeding layers are fully interconnected, but there might exist additional connections going to further layers or even missing connections between certain neurons.

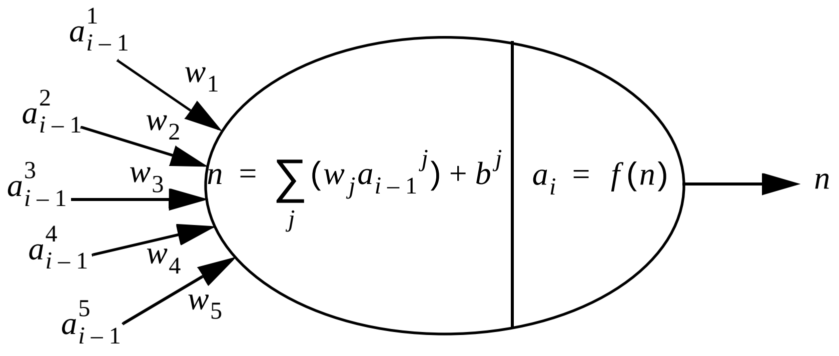
Weights and Biases

One of the most important aspects of neural networks is the storage of information (Khanna, 1996). Each connection is equipped with an individual weight and bias that modifies the signal flow on the respective connection. The weight works as a factor by which the output of the preceding neuron is multiplied. The bias works as a fine adjustment by which the product of weight and output from the preceding layer is added. This means that information is stored and distributed within a neural network and even minor destruction of some of the weights and biases will have large effect on the recall of learned information.

Recall

The phase when a neural network applies the information acquired during the learning phase is called the recall phase. The recall always starts by applying an input pattern to the input layer of the neural network (Khanna, 1996). Each of the input neurons holds a specific component of the input pattern and normally does not process it, but simply sends it directly to all the connected neurons. However, before their output can reach the succeeding neurons, it is modified by the weight and bias on the connection. All the neurons of the second layer then receive modified (i.e. weighted and biased) input values and process them. Afterwards these neurons send their output to succeeding neurons of the next layer. This procedure is repeated until the neurons of the output layer finally produce an output which is the neural network's answer to the presented input pattern.

Figure 2.6 A Single Neuron Diagram.



Transfer Functions

Transfer functions are the processing units of a neuron. These functions can be linear or non-linear. Three of the most common transfer function are depicted in Figure 2.7:

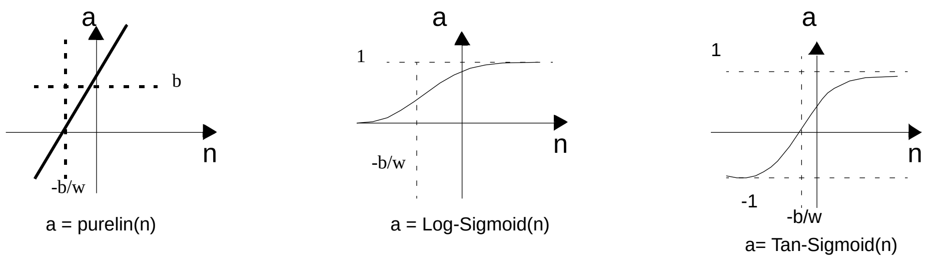
The mathematical formulation of the above functions is given as follows:

Pure-linear: $a = n$ (2.42)

$$\text{Log Sigmoid: } a = \frac{1}{1+e^{-n}} \tag{2.43}$$

$$\text{Tangent Sigmoid } a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}} \tag{2.44}$$

Figure 2.7 Basic Neural Network Transfer Functions.



Learning

The phase when sample patterns of a certain problem are presented to a neural network is called the training phase. During training, the weights and biases of the neural network are adjusted. Depending on the type of the neural network and on the problem it is going to solve, either a supervised or an unsupervised method can be used for adapting the weights (Beal, 1992). In both cases however, every training starts with a recall where the input is propagated through the neural network and all its neurons change their activity accordingly. A supervised training is typically chosen when the neural network to map input to output patterns is desirable. This requires that the output to a given input is known. After the recall phase, the output of the neural network is compared to what the resulting output pattern should be. The observed difference is used to adapt the weights and biases. The adaptation of the weights starts at the output neurons and continues downward toward the input layer. The weight and bias adaptation for one pattern often does not correct the neural network's faulty response completely, but improves it. Then the next input pattern is chosen and the whole process is repeated until the overall response of the neural network is satisfying. It is important to define the point where the training is terminated, because sometimes it is possible to over-train a neural network. Namely, at some point the neural network starts to memorize exactly the training examples with their inherent noise and later on it will not be able to generalize from the trained examples to new patterns presented during recall. An unsupervised training is chosen when the neural network has to classify data on its own. In this fashion the neural network distinguishes certain classes by using the interdependency it detects within the data. Some of these neural networks are even able to reorganize themselves, e.g. by recruiting new neurons to represent unknown patterns or new classes.

Neural Network Types

There are hundreds of different neural network types that can be classified in various ways, e.g. in the way they are trained (supervised, unsupervised, or reinforced), how the information flow in the network is organized (feedback, feedforward), how the topology is built (static or self-organizing). Another way to classify neural networks is by distinguishing between the training algorithms that are used to adjust the weights. In this case, the number of different training algorithms is even larger than the number of neural network types (Khanna, 1996).

The typical steps for creating a neural network application are:

1. Analysis of the problem and collection of all available data
2. Analysis of the collected data
3. Choice of the neural network type that is capable of solving your problem
4. Selection of the important features that will be used
5. Coding of the information, using the result of the data analysis
6. Separation of data basis into training and test set
7. Design of the appropriate neural network topology, choice of the neurons'
8. Functions, and basic decision about the amount of neurons to be used in each layer
9. Training of the neural network and monitoring its performance on the test set
10. Optimization of the neural network by changing the topology, the amount of neurons, the neurons' functions

2.2.2 Neural Learning Using Back-Propagation

One of the most powerful uses of a neural network is function approximation. Neural networks known as neural nets are computing systems which can be trained to learn a complex relationship between input variables and target data sets. Neural nets employ Parallel Distributed Processing (PDP) composed of interconnecting simple processing nodes. Neural net techniques have been successfully applied in various fields such as linear and/or non-linear function approximation, control systems and image processing. As discussed in the previous section, the learning process is the most important part of the entire process. The objective of the learning process is to train the network so that the application of a set of inputs produces the desired or at least a consistent set of outputs. During training the network weights gradually converge to values such that each input vector produces the desired output vector (Freeman, 1992).

A learning cycle starts with applying an input vector to the network, which is propagated in a forward propagation mode which ends with an output vector. Next, the network evaluates the errors between the desired output vector and the actual output vector. It uses these errors to shift the connection weights and biases according to a learning rule that tends to minimize the error. This process is generally referred to as “error back-propagation” or back-propagation for short. The adjusted weights and biases are then used to start a new cycle. A back-propagation cycle, also known as an epoch, in a neural network is illustrated in Figure 2.8. For a finite number of epochs the weights and biases are shifted until the deviations form the outputs are minimized.

2.2.3 Learning Rule and Lavenberg Marquardt Optimization Algorithm

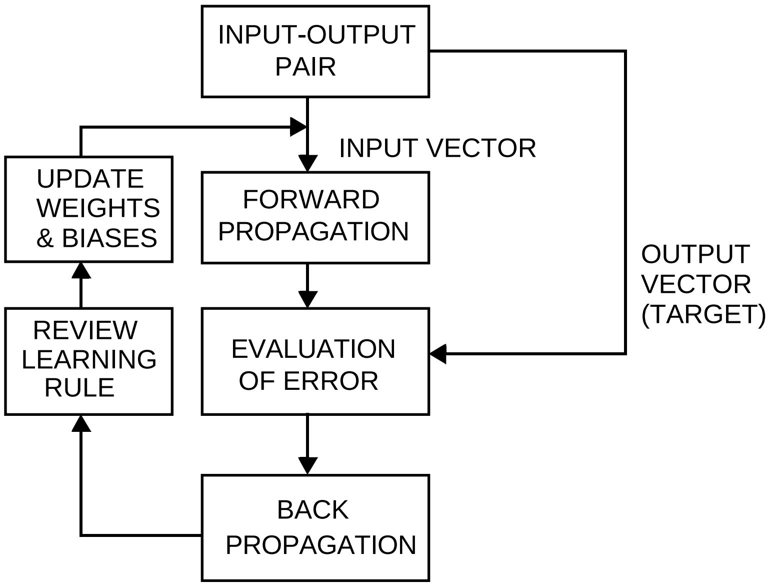
As stated in the previous section, the neural network learning process is actually an iterative process which minimizes the error between the outputs and the targets by shifting weights and biases toward the optimum (Hagan, 1996). This process can be achieved by applying the Levenberg-Marquardt algorithm. The Levenberg-Marquardt algorithm is based on two optimization techniques, the steepest descent algorithm and Newton's method. The difference between these two algorithms is that the steepest descent algorithm is based on the first order Taylor series expansion, and the Newton's method is based on the second order Taylor series.

Suppose a performance index Z($\vec{x}$) is to be minimized. The search of the optimum begins at ($\vec{x}_0$) and then updates the guess in stages according to following rule:

$$\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{p}_k \qquad (2.45)$$

where $\vec{p}_k$ is the search direction and $\vec{\alpha}_k$ is the step size. Note that Newton's method and steepest descent are distinguished by the choice of the search direction, $\vec{p}_k$.

Figure 2.8 Backpropagation Cycle.



<u>Steepest Descent Method</u>

This method is derived from a first order Taylor series expansion of Z($\vec{x}$) about the old guess $\vec{x}_k$:

$$Z(\vec{x}_{k+1}) = Z(\vec{x}_k + \Delta\vec{x}_k) = Z(\vec{x}_k) + \vec{g}_k^T \Delta\vec{x}_k \qquad (2.46)$$

where $\vec{g}_k$ is the gradient evaluated at the previous guess $\vec{x_k}$:

$$\vec{g}_k \equiv \quad Z(\vec{x})\big|_{\vec{x} = x_k} \tag{2.47}$$

For $Z(\vec{x}_{k+1})$ to be less than $Z(\vec{x})$, such that the function decrease at each iteration, the second term on the right hand side must be negative:

$$\vec{g}_k^T \Delta\vec{x}_k = \alpha_k \vec{g}_k^T \vec{p}_k < 0 \tag{2.48}$$

If $\vec{\alpha}_k$ is a small but positive value, then,

$$\vec{g}_k^T \vec{p}_k < 0 \tag{2.49}$$

The above vector will be most negative if:

$$\vec{p}_k = -\vec{g}_k \tag{2.50}$$

Using the above conclusion, an iteration until function Z is optimum produces the method of steepest descent:

$$\vec{x}_{k+1} = \vec{x}_k - \alpha_k \vec{g}_k \tag{2.51}$$

The learning rate, $\vec{\alpha}_k$ can be determined by:

1. Minimizing Z with respect to $\vec{\alpha}_k$, which is basically the same as minimizing along the line $\vec{x}_k - \alpha_k \vec{g}_k$.
2. Fixed learning rate.

Newton’s Method

If $B_k \equiv \quad ^2 Z(\vec{x})\big|_{\vec{x} = \vec{x}_k}$ and $\vec{g}_k \equiv \quad Z(\vec{x})\big|_{\vec{x} = x_k}$, then the second order Taylor series of $Z(\vec{x}_{k+1})$ can be expressed as:

$$Z(\vec{x}_{k+1}) = Z(\vec{x}_k) + \vec{g}_k^T \Delta\vec{x}_k + \frac{1}{2}\Delta\vec{x}_k^T \vec{B}_k \Delta\vec{x}_k \tag{2.52}$$

By taking the gradient of the above equation with respect to $\Delta\vec{x}_k$, and setting it equal to zero, $\Delta\vec{x}_k$ can be written as:

$$\Delta\vec{x}_k = -\vec{B_k}^{-1} \vec{g}_k \tag{2.53}$$

Newton’s method is then defined as:

$$\vec{x}_{k+1} = \vec{x}_k - \vec{B_k}^{-1} \vec{g}_k \tag{2.54}$$

Since error function in neural network is expressed in sum of squares function, for the purpose of this research project, assume that Z is the sum of squares function, such that:

$$Z(\vec{x}) = \sum_{i=1}^{N} \zeta_i^{2}(\vec{x}) = \xi^T(\vec{x})\zeta(\vec{x}) \tag{2.55}$$

and the jth element of the gradient can be written as

$$[\quad Z(\vec{x})]_j = \frac{\partial}{\partial \vec{x}_j} Z(\vec{x}) = 2\sum_{i=1}^{N} \zeta_i(\vec{x}) \frac{\partial}{\partial \vec{x}} \zeta_i(\vec{x}) \tag{2.56}$$

The gradient can $Z(\vec{x}) = 2J^T(\vec{x})\zeta(\dot{\vec{x}})$ (2.57)

where J is the Jacobian matrix as shown below:

$$J(\vec{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1}\zeta_1(\vec{x}) & \frac{\partial}{\partial x_2}\zeta_1(\vec{x}) & & \frac{\partial}{\partial x_n}\zeta_1(\vec{x}) \\ \frac{\partial}{\partial x_1}\zeta_2(\vec{x}) & \frac{\partial}{\partial x_2}\zeta_2(\vec{x}) & & \frac{\partial}{\partial x_n}\zeta_n(\vec{x}) \\ & & & \\ \frac{\partial}{\partial x_1}\zeta_N(\vec{x}) & \frac{\partial}{\partial x_2}\zeta_N(\vec{x}) & & \frac{\partial}{\partial x_n}\zeta_N(\vec{x}) \end{bmatrix} \tag{2.58}$$

The next step is finding the Hessian matrix. The k,j element of the Hessian matrix would be:

$$[\quad ^2 Z(\vec{x})]_{k,j} = \frac{\partial^2}{\partial x_k \partial x_j} Z(\vec{x}) = 2\sum_{i=1}^{N} \left\{ \frac{\partial}{\partial x_k}\zeta_i(\vec{x}) \frac{\partial}{\partial x_j}\zeta_i(\vec{x}) + \zeta_i(\vec{x}) \frac{\partial^2}{\partial x_k \partial x_j}\zeta_i(\vec{x}) \right\} \tag{2.59}$$

The Hessian matrix can then be expressed in matrix form:

$$^2 Z(\vec{x}) = 2J^T(\vec{x})J(\vec{x}) + 2S(\vec{x}) \tag{2.60}$$

where

$$S(\vec{x}) = \sum_{i=1}^{N} \zeta_i(\vec{x}) \quad ^2\zeta_i(\vec{x}) \tag{2.61}$$

If $S(\vec{x})$ is small then the Hessian matrix can be approximated as:

$$^2 Z(\vec{x}) \equiv 2J^T(\vec{x})J(\vec{x}) \tag{2.62}$$

Substituting Equations 2.55 and 2.60 into Equation 2.54, the following can be obtained:

$$\hat{x}_{k+1} = \hat{x}_k - [2J^T(\hat{x}_k)J(\hat{x}_k)]^{-1} 2J^T(\overrightarrow{x_k})\zeta(\hat{x}_k) \tag{2.63}$$

$$\hat{x}_{k+1} = \hat{x}_k - [J^T(\hat{x}_k)J(\hat{x}_k)]^{-1} J^T(\overrightarrow{x_k})\zeta(\hat{x}_k) \tag{2.64}$$

Equation 2.64 is known as the Gauss-Newton algorithm. The main advantage of using this method over Newton's method is that it does not require calculation of second derivatives. However, there is a weakness with the Gauss-Newton method, arising at instances when matrix $H = J^T J$ is not invertible. This can be overcome by using the following modification to approximate the Hessian matrix: $G = H + \mu I$.

To see how this matrix is invertible, suppose that the eigenvalues and eigenvectors of H are $\{\lambda_1, \lambda_2, \quad , \lambda_n\}$ and $\{\xi_1, \xi_2, \quad , \xi_n\}$. Then:

$$G\xi_i = [H + \mu I]\xi_i = H\xi_i + \mu\xi_i = \lambda_i\xi_i + \mu\xi_i = (\lambda_i + \mu)\xi_i. \tag{2.65}$$

Therefore the eigenvectors of G are the same as the eigenvectors of H, and the eigenvalues of G are $(\lambda_i + \mu)$. G can be made positive definite by increasing μ until $\lambda_i + \mu > 0$ for all i, and therefore the matrix will be invertible.

This leads to an important result, known as Levenberg-Marquardt algorithm:

$$\hat{x}_{k+1} = \hat{x}_k - [J^T(\hat{x}_k)J(\hat{x}_k) + \mu_k I]^{-1} J^T(\overrightarrow{x_k})\zeta(\hat{x}_k) \tag{2.66}$$

or

$$\Delta\hat{x}_k = -[J^T(\hat{x}_k)J(\hat{x}_k) + \mu_k I]^{-1} J^T(\hat{x}_k)\zeta(\hat{x}_k) \tag{2.67}$$

The main characteristic of this equation is that when μ_k is increased it approaches the steepest descent algorithm with a small learning rate:

$$\hat{x}_{k+1} \quad \hat{x}_k - \frac{1}{\mu_k}J^T(\hat{x}_k)\zeta(\hat{x}_k) = \hat{x}_k - \frac{1}{2\mu_k} \quad Z(\hat{x}) \tag{2.68}$$

while as μ_k is decreased to zero the algorithm becomes Gauss-Newton.

Typically, the algorithm begins with μ_k set to some small value. If a step does not yield a smaller value for $Z(\hat{x})$, then the step is repeated with μ_k multiplied by some factor κ for $\kappa > 1$. Eventually, $Z(\hat{x})$ should decrease, since a small step would be taken in the direction of steepest descent. If a step does produce a smaller value for $Z(\hat{x})$, then μ_k is divided by κ for the next step, so that the algorithm will approach Gauss-Newton, which should provide faster convergence. The algorithm provides a good compromise between the efficiency of Newton's method and the guaranteed convergence of steepest descent method.

Implementing the Levenberg algorithm method to the neural network training is done by replacing the Z function discussed by the performance index function F. The performance index function for multilayer network is the mean squared error.

Suppose $\vec{p}$ is an input vector, the corresponding target output is $\vec{t}$. The algorithm should adjust the network parameters in order to minimize the mean squared error:

$$F(\vec{x}) = \Phi[e^2] = \Phi[(t-a)^2] \tag{2.69}$$

Here a is the output of the tested value and x is the vector of network weights and biases.If the network has multiple outputs this generalizes to

$$F(\vec{x}) = \Phi[\vec{e}^T\vec{e}] = \Phi[(\vec{t}-\vec{a})^T(\vec{t}-\vec{a})] \tag{2.70}$$

If each target occurs with equal probability, the mean squared error is proportional to the sum of squared errors over Q targets in the training set:

$$\hat{F}(\vec{x}) = \sum_{q=1}^{Q}(\vec{t}_q-\vec{a}_q)^T(\vec{t}_q-\vec{a}_q) \tag{2.71}$$

$$\hat{F}(\vec{x}) = \sum_{q=1}^{Q}\vec{e}_q^{\,T}\vec{e}_q = \sum_{i=1}^{N}(\zeta_i)^2 \tag{2.72}$$

Equation 2.71 is equivalent to the performance index shown in Equation 2.55, for which Levenberg-Marquardt was designed. Therefore it should be a straight forward matter to adapt the algorithm for network training.

CHAPTER 3 Methodology

The general approach to demonstrate the use of neural networks in fuel consumption estimation follows six steps:

1. Selection of testing aircraft
2. Data collection from a flight manual
3. Training neural networks for the corresponding data base
4. Implementation and generalization of the neural network
5. Correlation of results
6. Implementation of the neural network to an actual flight trajectory

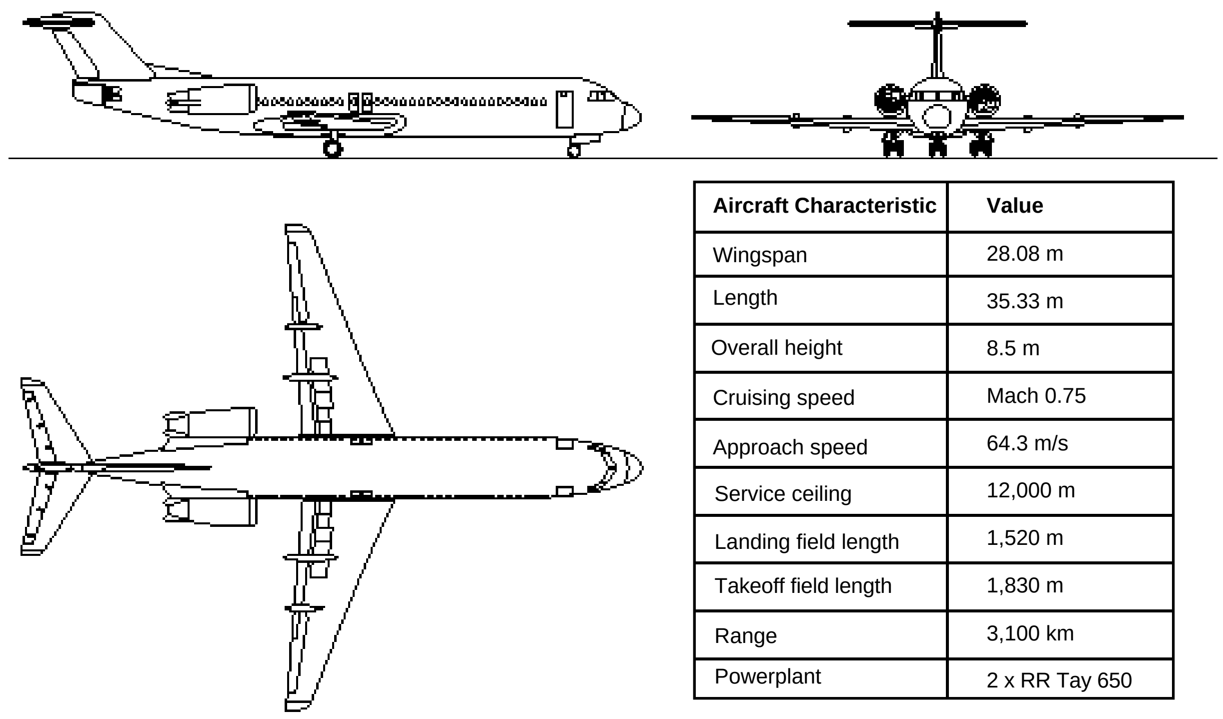
3.1 Selection of the Testing Aircraft

The purpose of this section is to show that neural networks have the potential to approximate various aircraft fuel consumption data sets in a reliable and efficient manner. This chapter illustrates this using data sets taken from the Fokker F-100 - a short range turbofan-powered aircraft. The specifications of this particular aircraft are shown in Figure 3.1

3.2 Fuel Consumption Data Collection

A scanner connected to a personal computer was used to digitize all flight performance chart contained in the flight manual. The flight manual consists of different charts relating fuel consumption to flight performance parameters such as Mach number, altitude, weight and international atmospheric conditions (i.e., ISA, ISA + 10, etc.). Using a data retrieval program all digitized maps were converted into usable table data depicting aircraft performance for various flight regimes. The typical segmentation of flight phases stated in the flight manual was maintained to preserve the accuracy of the tables as much as possible. Typical performance charts included takeoff and climbout, climb, cruise, descent and taxi profiles.

Figure 3.1 Fokker F-100 Aircraft Layout and General Characteristics.



Aircraft Characteristic	Value
Wingspan	28.08 m
Length	35.33 m
Overall height	8.5 m
Cruising speed	Mach 0.75
Approach speed	64.3 m/s
Service ceiling	12,000 m
Landing field length	1,520 m
Takeoff field length	1,830 m
Range	3,100 km
Powerplant	2 x RR Tay 650

3.3 Training the Neural Network

In order to simplify our analysis for training a neural network we used the MATLAB Neural Network Toolbox. MATLAB is a general mathematical package produced by the Mathworks Company. This tool is very efficient to handle matrices and was used throughout this research project to handle data manipulation tasks and neural network computations. Moreover, MATLAB has excellent graphics and visualization routines that made this effort more manageable.

The reader must It must understand that once a neural network is trained and properly calibrated its implementation becomes independent of the mathematical package used. Consequently although our approach was substantially simplified with the use of MATLAB the actual implementation of the neural network applied to aircraft fuel consumption is relatively easy to implement in any higher level programming language (including simulation languages like SIMSCRIPT II.5) that handles arrays.

Throughout this research project several programs or templates were developed in MATLAB to perform the following neural net computations:

1. Network training/learning.
2. Testing and evaluation of a trained network.
3. Implementation to estimate aircraft fuel consumption in any trajectory.

For any given aircraft, the data was split into learning and testing sets. Learning sets are used to train the neural network to recognize patterns. Testing sets are random values of the input parameters to the neural network used to validate the outputs of the trained network. Each template requires certain the following inputs:

1. Number of inputs.
2. Value for the learning coefficient.
3. Number of processing elements (neurons) in the hidden layer and output layers.
4. Maximum number of cycles (epochs) for each run.
5. Required accuracy in the training procedure (i.e., sum of the squared errors for each run).

In general, all programs developed as part of this research effort can be viewed as computational templates that are fully reusable for any number of aircraft. In this research we tested two dissimilar aircraft to validate that the topology of the neural network employed could in fact characterize various performance envelopes for turbofan and turboprop powered aircraft. The Saab 2000 was employed as testing aircraft for the later assessment.

3.3.1 MATLAB Neural Network Basics

MATLAB is a powerful and advanced computational software developed by MathWorks Inc. MATLAB features a well developed and tested neural network toolbox that has multiple built-in functions to ease the training and generalization of neural networks. Some of the relevant commands for neural network basics are included in this chapter for completeness and better understanding of the source code in Appendix A.

Initff

Purpose:

Initializes a feed-forward network.

Synopsis:

```
[w1,b1,w2,b2,w3,b3] = initff(P,S1,F1,S2,F2,S3,T)
```

Description

initff(P,S,T) - takes a matrix of input vectors P, the number of neurons S, and the number of rows in target vector T, and returns the weights and biases for a single layer with S neurons.

trainnlm

Purpose:

Trains a feed-forward network with Levenberg-Marquardt Algorithm.

Synopsis:

[w1,b1,w2,b2,w3,b3] = trainnlm(w1,b1,’F1’,w2,b2,’F2’,w3,b3,’F3’,P,T,tp)

where

w is the weight vector

b is the bias vector

F is the transfer function

P is the input vector

T is the Target vector

tp is the optional training parameter

tp(1) - Epochs between updating display

tp(2) - Maximum number of epochs to train

tp(3) - Sum-squared error goal

tp(4) - Minimum gradient

tp(5) - Initial value for μ , learning rate

tp(6) - Multiplier for increasing μ

tp(7) - Multiplier for decreasing μ

tp(8) Maximum value for μ

Description:

trainlm - a function which employs the Levenberg-Marquart Algorithm in training the weights and biases to map the input vectors.

Training continues until the error goal is met or until the number of epochs reaches a maximum number of epochs.

The variable μ determines whether learning progresses according to Newton’s method or gradient descent. Here is the Levenberg-Marquardt rule for updating parameters (such as weights and biases):

$$V = (J^T J + \mu I)^{-1} J$$

Here, J is the Jacobian matrix, as discussed in Chapter 2. As the error e gets large the $J^T J$ term becomes negligible and learning progresses according to $\mu^{-1} J^T e$, which is gradient descent. Whenever a step is taken with increasing error, μ is increased until a step can be taken without increasing error. However, if μ becomes too large no learning takes place (i.e. $\mu^{-1} J^T e$ approaches zero). This occurs when an error minima has been found. This is why learning stops when μ reaches its maximum value.

simuff

Purpose:

Simulate a feed-forward network.

Synopsis:

simuff(P,w1,b1,F1,w2,b2,F2,w3,b3,F3)

Description

simuff - a feed-forward network consisting of a set of layers, where each layer receives its input from the previous layers.

simuff(P,w1,b1,F1,w2,b2,F2,w3,b3,F3) takes the weights (w), biases (b) and user defined transfer functions of three layers and returns the network outputs.

logsig

Purpose:

Log sigmoid transfer function.

Synopsis:

logsig(n)

Description:

logsig - log-sigmoid is a function used to map a neuron input from the interval $(-\infty, \infty)$ into the interval (0,1). The log-sigmoid is a differentiable function, which makes it suitable for neurons being trained with Levenberg-Marquardt algorithm. The following is the log-sigmoid equation as it is applied to each input element:

$$\log sig(n) = \frac{1}{1 + e^{-n}} \qquad (3.1)$$

tansig

Purpose:

Tangent-sigmoid transfer function.

Synopsis:

tansig(n)

Description:

tansig - a tangent-sigmoid function, used to map a neuron input from the interval $(-\infty, \infty)$ into interval (-1,1). The tangent-sigmoid is a differentiable function, which makes it suitable for neurons being trained with Levenberg-Marquardt algorithm. The following is the tangent-sigmoid equation as it is applied to each input element:

$$\tan sig(n) = \tanh(n)) \qquad (3.2)$$

purelin

Purpose:

linear transfer function.

Synopsis:

purelin(n)

Description:

purelin - the simplest transfer function a neuron can have is the pure linear transfer function, which simply passes a neuron’s input vectors on to its output, being altered only by the neuron’s bias, which is added to it.

3.3.2 Selection of Training Algorithms

Based on the analysis performed with several transfer function algorithms the Levenberg-Marquardt algorithms are found to be the most efficient and reliable means to be used for this study. Table 3.1 shows a comparison of the three most popular supervised algorithms. These numbers are based on MATLAB Version 5.1 run on a PowerMacintosh 8500 computer (i.e., PowerPC 604-120 microprocessor).

TABLE 3.1. Comparison of Optimization Algorithms

Function	Technique	Time (s)	Flops
TRAINBP	Backpropagation	259.1	2.16E+0.8
TRAINBPX	Fast Backprop	42.4	2.97E+0.7
TRAINLM	Levenberg-Marquardt	3.3	315769

Therefore, Levenberg-Marquardt Algorithm was the choice for our neural network design and analysis.

3.3.3 Design of the Appropriate Neural Network Topology

Design of the appropriate neural network topology involves several important steps:

1. Choosing the appropriate neurons' transfer functions,
2. Basic decision about the amount of neurons to be used in each layer,
3. Selecting the amount of hidden layers.

Function approximation has been traditionally one of the most researched uses of neural networks. Typically, a two or three layer network is sufficient to approximate any function with a finite number of discontinuities. In order to gain an insight as to how topology effects the outputs, tangent-sigmoid, logarithmic-sigmoid and pure linear neuron transfer functions were selected and tested for further investigation. Moreover, the amount of neurons each layer depends on the complexity of the target function used. If there are not enough neurons in each layer, the outputs will not be able to fit all the data points (under-fitting). On the other hand, if there are too many neurons in each layer, oscillations may occur between data points (over-fitting). Therefore, a topology study was conducted in order to find the most appropriate architecture neural network to fit aircraft fuel consumption parameters. Note that there are several hundred combinations of neurons and layers but for practical purposes we tested eight candidate topologies shown in Table 3.2. The training input data sets for this part of the experiment are 805 data points selected from the flight manual cruise section and then tested (or generalized) with 805 random data points selected from the same flight performance manual (Fokker, 1995). Since the testing data set was selected randomly these data points were not the same as those used in the training procedure.

TABLE 3.2. Results of the Topology Sensitivity Study

Number of layers	Number of neurons	Transfer functions	Mean Relative Error(%)	Standard Deviation of Error(%)	Floating point operations
2	4	tansig-purelin	0.611	0.0282	4.667E+08
2	6	tansig -purelin	0.606	0.0281	1.631E+09
2	8	tansig - purlin	0.628	0.0288	8.258E+08
3	4	logsig-tansig-pure-lin	0.617	0.0288	1.563E+10
3	6	logsig-tansig-pure-lin	0.610	0.0280	1.809E+10
3	8	logsig-tansig-pure-lin	0.604	0.0279	7.687E+08
3	10	logsig-tansig-pure-lin	0.656	0.0290	2.076E+09
3	12	logsig-tansig-pure-lin	0.667	0.0295	1.271E+09

The network topology study is based on the number of floating point operations (flops) and output errors obtained for each network topology. Table 3.2 lists all the candidates selected and their corresponding computational performance.

Based on the results shown in Table 3.2, the two best candidates are two layers with six neurons and three layers with eight neurons. The corresponding mean errors are 0.606% and 0.604% (for 10,000 epoch). Unfortunately, further testing showed that the two layer architecture does not converge to a minimum error during training with input data from the climb section after 10000 epochs. Therefore, the final topology selected for this research is a three layer model with eight neurons in the first two layers and one neuron in the output (third) layer. The corresponding transfer functions are logsig-tansig-purelin, respectively.

3.4 Implementation and Generalization of the Neural Network

The neural network employed in the fuel burn evaluation model is based on a non-linear optimization technique because of the non-linearity of the transfer functions. This technique is sometimes called continuous-variable non-linear programming. It involves real decision variables but the objective and constraint functions are not necessarily linear. The objective of optimization is to train the network parameters weights (w) and biases (b) so they can be adjusted in an effort to optimize the performance of the network. Neural nets are taught to accommodate changes in the weights and bias to appropriately reconfigure the output. Each time it iterates (teaches), the error between the output and target becomes smaller until a minimized error goal is achieved. The goals of the model are as follows:

- Understandable and easy to use
- Suitable for a wide range of aircraft
- Easy to modify or update the model

3.4.1 Description of the Neural Network Aided Fuel Consumption Model

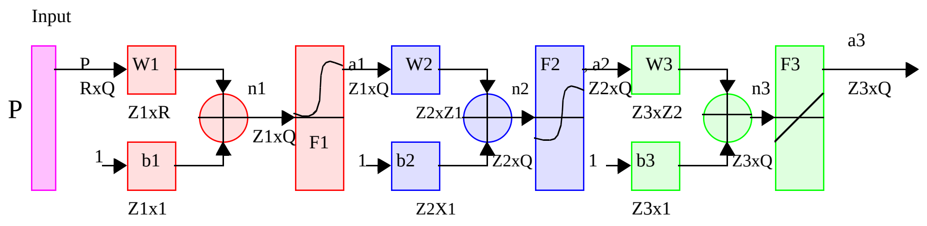
Fuel burn evaluation of a particular aircraft for each mission is divided in six segments:

- Warm Up and Taxi
- Takeoff and Climb-Out
- Climb
- Cruise
- Descent
- Approach and Landing

For each segment, there is a set of trained neural network weights and biases. The raw data for training purposes is obtained from the flight manual of a particular aircraft and used as inputs to train the neural network. A back-propagation neural network with Levenberg-Marquardt approximation model is employed. This model is created by generalizing the Widrow-Hoff learning rule to multiple-layer networks and non-linear differentiable transfer functions. Input vectors and the corresponding output vectors are used to train a network until it can accurately approximate a function. The reason for choosing this model is that this network with biases, sigmoid layers, and a linear output layer is capable of approximating any function with a finite number of discontinuities. The Levenberg-Marquardt approximation is a non-linear optimization algorithm that would reduce the training time using Newton's method.

Figure 3.2 displays the architecture of the three layer fuel burn approximation network.

Figure 3.2 General Three Layer Neural Network.



where,

P is the input vector

Zi is the number of neurons in ith layer, for i=1,2,3

Z1=8

Z2=8

Z3=1

F1 is a logrithmic sigmoid transfer function

F2 is a tangential sigmoid transfer function

and F3 is a pure linear transfer function

Note that, although different flight segments require different inputs, the training procedure remains the same. Once this network is developed, the fuel burn estimation of an aircraft is very simple. The following are assumptions made to estimate the fuel burn:

- Aircraft acceleration is not taken into consideration, therefore all velocity changes are assumed to be instantaneous.
- Aerodynamic effects are not used, all data is taken directly from the flight manual.
- Along with these assumptions, the fuel burn calculation for different flight segments can be established.

For warm up and taxi, the fuel flow rate is calculated using results from a linear fitting technique. The reason for using a linear fitting technique is that the fuel burn rate and initial weight have a linear relationship. Once the slope and the y-intersection of the line is found (see Figure 3.2), for any given initial weight there will be a corresponding fuel rate. Multiplying the fuel flow rate by the taxing time gives the total fuel burn for this particular segment.

For takeoff and climb-out, with the final aircraft weight left from the previous phase, the total fuel burn for takeoff and climb-out to a certain altitude can be found for an aircraft using a trained neural network. This statement is illustrated in Figure 3.3.The fuel burn for climb to cruise altitude can be calculated given an initial weight at the beginning of the flight phase, initial and target altitudes, temperature difference from ISA conditions and Mach number.

Figure 3.3 Fuel Burn for Takeoff and Climbout.

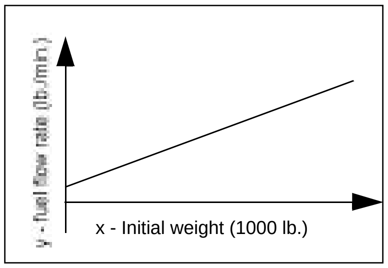


Figure 3.4 Fuel Estimation Procedure for Takeoff and Climbout.

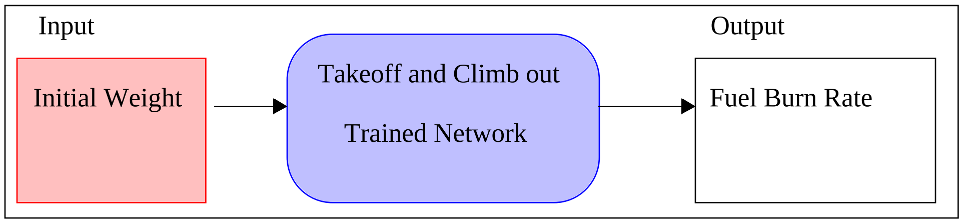
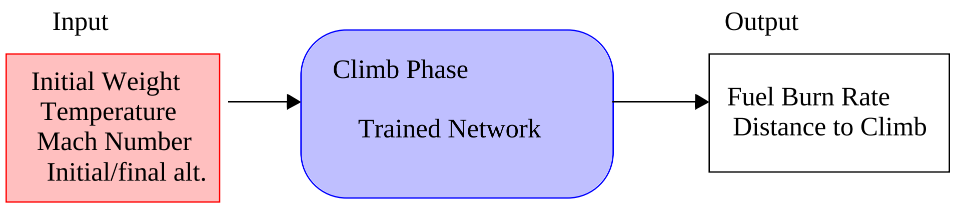


Figure 3.5 Fuel Estimation Procedure for Climb Segment.



For cruise, given the cruise altitude and Mach number, for a given initial weight, the specific air range (SAR) can be calculate using neural network.

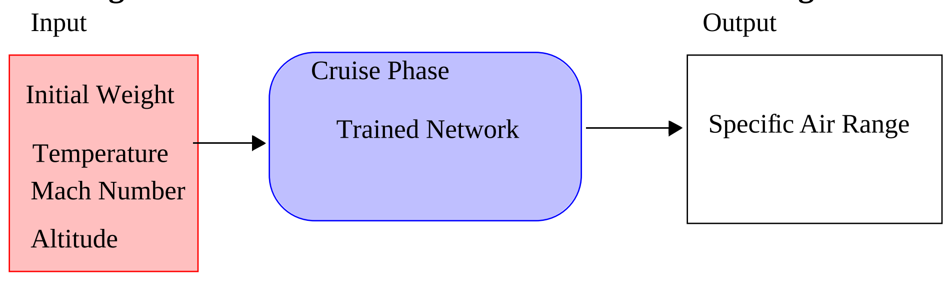
Since,

$$FuelBurn(lb) = Range(nm)/(SAR) \tag{3.3}$$

the fuel burn can be found using the required Mach number for a time interval. The fuel burn for descent and landing is similar to the takeoff and climb-out phase, therefore the details will not be repeated here.

As shown above, fuel burn estimation using neural network is simpler than calculating all the coefficients appearing in Collins algorithm. To demonstrate the capability and reliability of this method, the Fokker 100 flight manual will be used to train the aforementioned networks. Table 3.3 contains a summary of the amount of data used to train and test the network for different segments of flight, and the corresponding inputs and outputs.

Figure 3.6 Fuel Estimation Procedure for Cruise Segment.



3.5 Correlating Results

In order to justify the results standard statistical measures were used in this study. The following are the statistical measures which will be employed for this research.

3.5.1 Statistical Analysis Testing Procedures

For the purpose of this research, neural network predicted fuel burn will be compared to the actual fuel burn for each performance point. Each pair of actual and predicted fuel burn has an error value, i.e.

$$Error_i = \frac{(Fuelburn_{actual})_i - (Fuelburn_{predicted})_i}{(Fuelburn_{actual})_i} \times 100\,\% \tag{3.4}$$

for $i = 1, 2, \quad, N$, where N is the number of testing data points for the corresponding flight phase.

For each flight phase the mean and variance of all the errors was analyzed. Standard hypothesis testing techniques were used to demonstrate that means of the actual and predicted data were the same.

TABLE 3.3. Summary of Neural Network Training and Testing

Flight Phase	Number of Training Points	Number of testing points	Input Parameters	Output
Takeoff and Climb-Out	8	N/A	a) ISA Cond. b) Initial Weight (1000 lb.)	Fuel Burn Rate (lb/min)

Climb to Cruise Altitude	852 (Fuel)
	854(Distance)
	Total 1706
		852(Fuel)
		854(Distance)
		Total 1706
			a) Initial Weight (1000 lb.)
			b) ISA Cond
			c) Mach Number
			d) Target Altitude (1000 ft)
				a) Fuel Burn (lb.)
				b) Distance to Climb (nm)

Cruise	805	805	a) Cruise Mach Number b) Cruise Weight (1000 lb) c) Cruise Altitude (1000 ft)	Specific Air Range (nm/lb)
Descent	1210 (Fuel) 288(Distance) Total 1498	1210 (Fuel) 288 (Distance) Total 498	a) Initial Weight (1000 lb) b) ISA Cond c) Mach Number d) Target Altitude (1000 ft)	a) Fuel Burn (lb) b) Descent Distance (nm)

3.6 Implementation of the Outputs to an Actual Flight Trajectory

The neural network trained weight matrix and bias vector were implemented in a simulation program. This simulation program takes a desired four-dimensional aircraft trajectory as input and uses the corresponding weights and biases of the calibrated neural network model to estimate the fuel consumption. A complete mission includes:

- Taxi and warm up
- Takeoff and climb out
- Climb to cruise altitude
- Cruise at desirable attitude
- Descent
- Approach and landing

The simulation program is basically a feed-forward one step process. For a single neuron, suppose each layer has its own trained weight matrix W and its own bias vector b, along with the transfer function f and input vector p, output a can be expressed as:

$$a = f(Wp + b) \tag{3.5}$$

For the purpose of this research, as noted previously, a three layered network with eight neurons in the first two layers and one neuron in the third layer is utilized. Their corresponding transfer functions are logarithmic-sigmoid for the first layer, tangential-sigmoid for the second layer and pure linear for the third layer. As a demonstration of how the feed-forward process works, the climb phase of flight will be used. As stated previously, inputs of this phase are initial weight, Mach number, target altitude and ISA condition. The output is fuel consumption. Therefore, this model should be able to synthesize four input parameters and give one output. The corresponding architecture is shown in Figure 3.7. Note that there are four inputs, eight neurons in each hidden layer and one neuron in the output layer. The outputs of layers one and two are the inputs for layers two and three. Thus layer two can be viewed as a one-layer network with eight inputs, eight neurons, and an 8x8 weight matrix W^2. The input to layer two is a^1, and the output is a^2. Therefore, the three layer network in this example can be written as the following expression.

$$fuelburn = a^3 = f3(W^3 f2(W^2 f1(W^1 p + b^1) + b^2) + b^3) \tag{3.6}$$

where

$$W^1 = \begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & & w_{1,8}^1 \\ w_{2,1}^1 & w_{2,2}^1 & & w_{2,8}^1 \\ w_{3,1}^1 & w_{3,2}^1 & & w_{3,8}^1 \\ w_{4,1}^1 & w_{4,2}^1 & & w_{4,8}^1 \end{bmatrix}, W^2 = \begin{bmatrix} w_{1,1}^2 & w_{1,2}^2 & & w_{1,8}^2 \\ w_{2,1}^2 & w_{2,2}^2 & & w_{2,8}^2 \\ & & & \\ w_{8,1}^2 & w_{8,2}^2 & & w_{8,8}^2 \end{bmatrix} \text{ and } W^2 = \begin{bmatrix} w_{1,1}^3 \\ w_{2,1}^3 \\ \\ w_{8,1}^3 \end{bmatrix} \tag{3.7}$$

Hence for the first neuron in the first layer the output a$_1^1$can be expressed as:

$$a^1{}_1 = f1\left(\begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & w_{1,2}^1 & w_{1,2}^1 \end{bmatrix} \begin{bmatrix} altitude \\ Mach \\ Weight \\ ISA \end{bmatrix} + b_1^1 \right) \tag{3.8}$$

Since

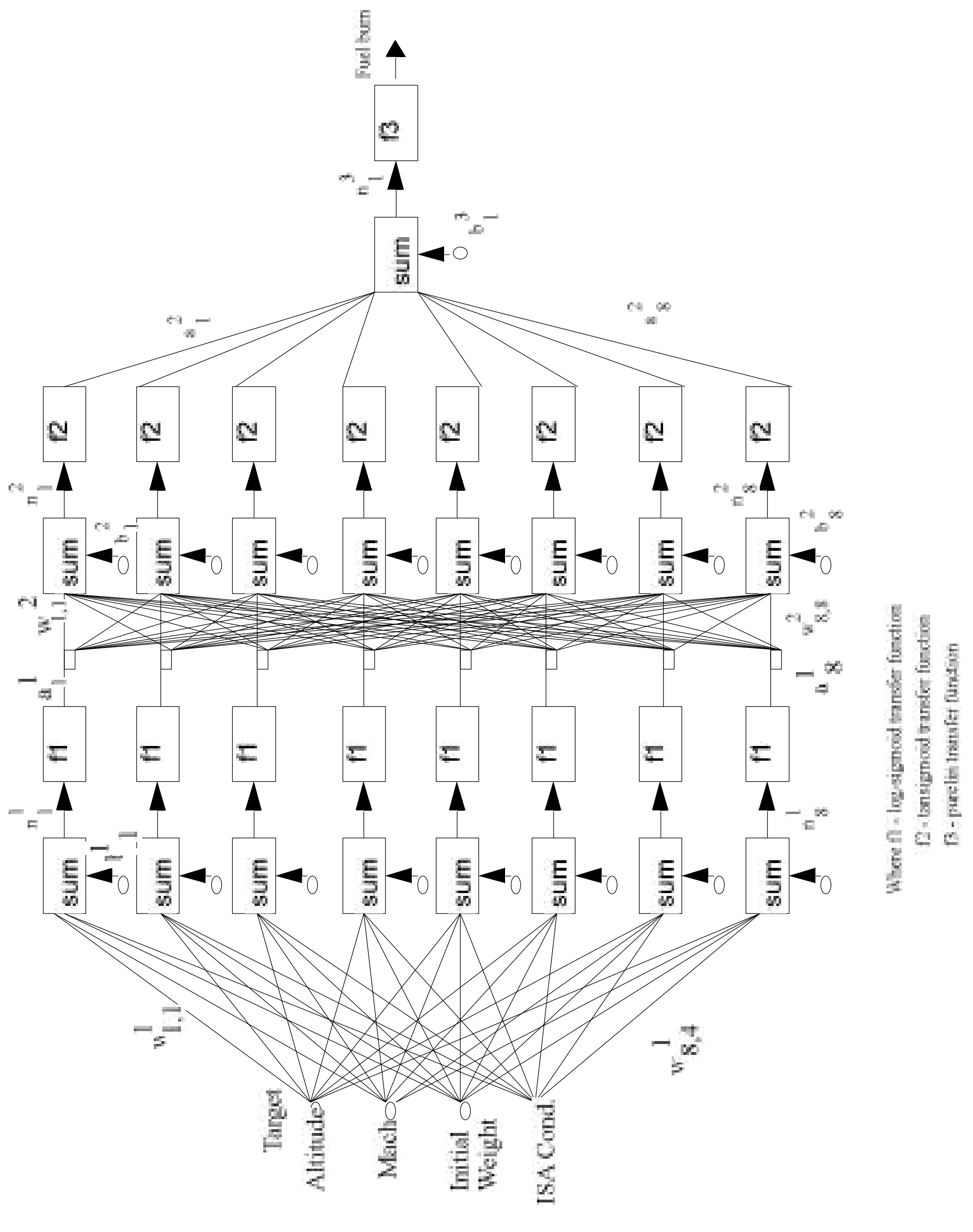
$$f1(n) = \frac{1}{1 + e^{-n}} \tag{3.9}$$

therefore,

$$a^1_1 = \frac{1}{1 + e^{\left(-\begin{bmatrix} w_{1,1}^{1} & w_{1,2}^{1} & w_{1,2}^{1} & w_{1,2}^{1} \end{bmatrix} \begin{bmatrix} altitude \\ Mach \\ Weight \\ ISA \end{bmatrix} + b_1^{1}\right)}} \tag{3.10}$$

Calculation of fuel burn in all other flight phases is based on the same principle and therefore will not be repeated here. Chapter 4 provides a description of all the computer programs created for this research project giving the reader insight into some of the technical details involved in neural network model testing and development.

Figure 3.7 Selected Neural Network Architecture.


Target Altitude
Mach
Initial Weight
ISA Cond.
Fuel burn
sum
f1
f2
f3
Where f1 - log-sigmoid transfer function
f2 - tansigmoid transfer function
f3 - purelin transfer function

CHAPTER 4 Neural Network Model Development

This chapter describes all the computer programs constructed to support the development of a neural network based fuel consumption model. The description of the computer programs is intended to present all major components of this project. The format for this presentation is as follows:

1. Training of weight matrix for different flight phases

    Takeoff and climb out

    Climb to cruise altitude

    Cruise

    Descent

2. Testing of preliminary results using statistics
3. Implementation of results to actual flight trajectory

4.1 Training of Weight Matrix For Different Flight Phases

The program developed in this section was used to perform neural net training. Inputs for the program are the learning set of data obtained from the flight manual. As discussed in Chapter 3, besides the learning set of data, the following data sets are also required as inputs:

a) The number of inputs to the neural network

b) The value of each learning coefficient

c) The number of processing elements in hidden and output layers

d) The number for each run

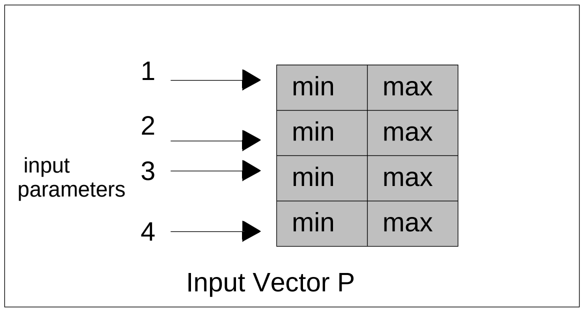
The MATLAB notation such as initff (initialize feed-forward) model, trainlm (training network using Levenberg-Marquardt Algorithm), and other transfer functions described in Chapter 3 are employed throughout this section.

The first step for training the weight matrix or neural network learning process is to initialize the data. Before this task can be done, all the data in the training set must be normalized, such that the training time can be reduced. The data in the training set consists of several input variables. The normalization is performed by dividing the input variables by the maximum value for each input variable. This results in the input field consisting of numbers that are greater than zero, and less than or equal to one, i.e. $0 < Input \leq 1$.

Recall the initialization function from Chapter 3, initff. This particular function takes a matrix of an input vector P, the number of neurons S, and the number of rows in target vector T, and returns the weights and biases for a single layer with S neurons. It is important to note that each row of P contains the minimum and maximum expected values of the network inputs so that the weights and biases can be initialized properly. Figure 4.1 shows an illustration of the input vector P.

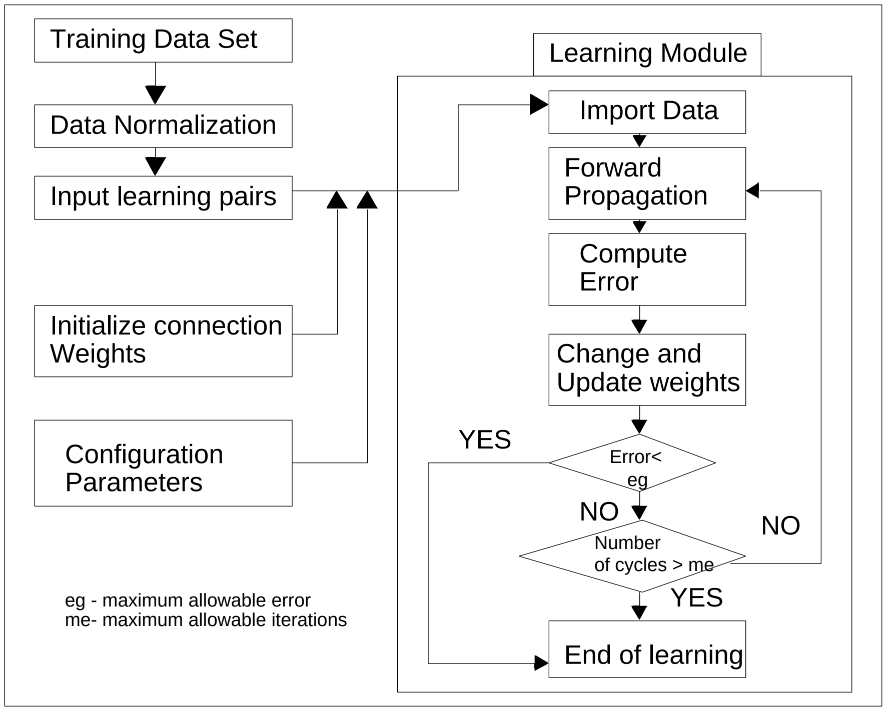
The second step is to train the weight matrix of the neural network using back-propagation with Levenberg-Marquardt algorithm. A complete listing of the MATLAB neural network program is shown in Appendix A. The structure of the program is illustrated in Figure 4.2.

Figure 4.1 Sample Input Vector P for Training Purposes.



Since different flight phases require different inputs, for each flight phase there is an individual training routine to produce a corresponding weight matrix. Table 4.1 lists a summary of the input and output parameters for various flight phases. Throughout this research project a series of training templates were created to simplify future estimation of aircraft specific parameters. While all data presented in this report addresses the Fokker 100 aircraft it is very easy to derive weight and bias parameters for any other aircraft if a training dataset exists. Since most of the aircraft manufacturers present performance information in either tabular or graphical form the training and generalization of neural networks using the template approach developed in this project greatly simplifies future implementation tasks.

Figure 4.2 Neural Network Training Process.



4.2 Testing Preliminary Results Using Statistics

Preliminary results for climb, cruise and descent segments of flight will be used to test the reliability of using the neural network to estimate fuel consumption of an aircraft. For these three segments, testing data sets shall be imported into the testing program developed. Using results from the neural network training, each performance vector in the testing data set will be processed by a feed-forward propagation (simulation) routine which gives an output. Outputs of the neural network are compared with the actual values acquired from the flight manual for the corresponding performance point. Standard statistical methods were to validate that all data points conform to the sam set or not. For the climb segment, a total of 1700 performance points were used to train the neural network. For the cruise segment, a total of 805 performance points were tested given that nonlinearities of the data are less severe than those encountered in climb or descent phases. Finally, for the descent segment, a total of 1210 performance datapoints were be tested. Note that all the performance points were being selected in a random manner. A simple flow diagram is shown in Figure 4.3 to demonstrate the testing process. Moreover, a list of testing programs for this part of the study are shown in Appendix A.

TABLE 4.1. Summary of Neural Network Input and Output Parameters.

Flight Phase	Input Parameters	Output Parameters
Takeoff and Climb Out	Weight (1000lb) ISA Conditions	Fuel Burn Rate(lb/min)
Climb to Cruise Altitude	Weight (1000lb) ISA Conditions Mach Number Target Altitude (1000ft)	Fuel Burn (lb.) Distance to Climb (nm)
Cruise	Altitude (1000 ft.) Weight (1000 lb.) Mach	Specific Air Range (lb/nm)
Descent	Weight (1000 lb.) ISA Conditions Mach Number Target Altitude (1000 ft.)	Fuel Burn (lb.) Descent Distance

4.3 Implementation of Neural Network to Actual Flight Trajectory

Once the training of the neural network is completed and tested, outputs are ready to be implemented to a simulation program. In addition to the outputs from the neural network, a sample flight trajectory is inputted into the simulation program. This flight trajectory is described by the initial weight of the aircraft before takeoff, the velocity and altitude schedules, and the number of way points. The velocity and altitude schedules are described by way points. Way points are points on the flight trajectory for which the velocity and altitude are specified, as shown in Figure 4.4.

With these inputs, the simulation program will calculate the fuel consumed at each way point, in the following manner:

1. For taxi, as already demonstrated, the fuel consumption is a linear function of weight, therefore the neural network is not required. Based on the initial weight and taxi time, the fuel burned can be calculated.

2. Takeoff and climb-out are neural network aided. This flight phase consists of only one way point. The initial weight for this flight segment is the final aircraft weight at the end of the taxi segment. The weight is normalized as previously described, and entered into simuff, which yields a normalized fuel burned.

Figure 4.3 Neural Network Testing Procedure.

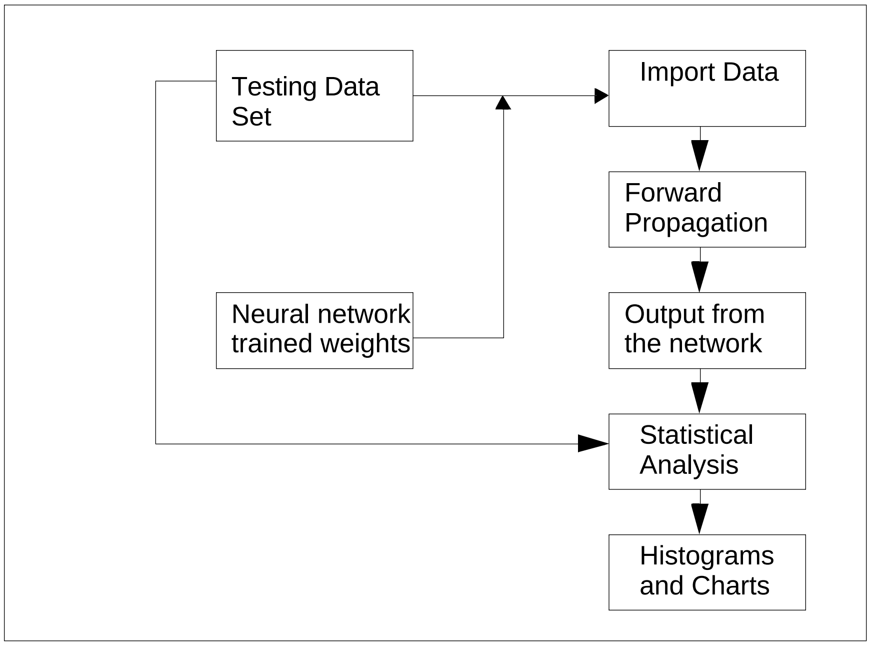
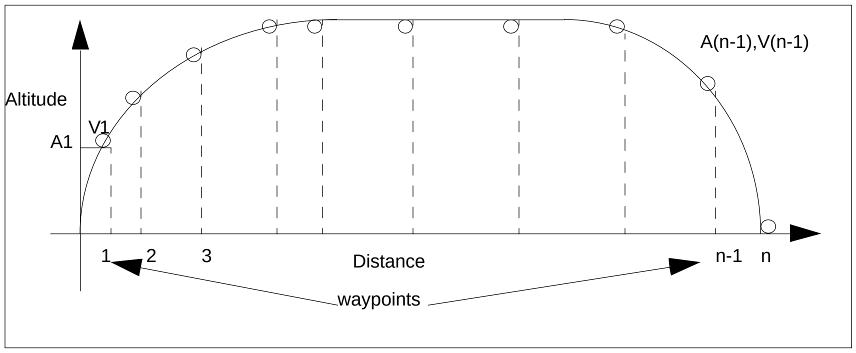


Figure 4.4 Flight Trajectory Implementation with Waypoints.



3. Techniques employed for fuel consumption estimations of the climb phase are based on the actual procedures followed by a pilot during flight. Fuel consumption information conveyed in the flight manual in most cases includes only typical speeds, altitudes and weights. For instance, in the flight manual for the Fokker 100, climb profiles are given only for the following cases:

Mach 0.65, 0.70, 0.73, and 0.75

Weight 62, 66, 70, 74, 78, 82, 86, 90, 94, 98, 102, and 106 (1000 lb.)

ISA conditions 0 and +10

The neural network simulation program is capable of producing results for input values for which it was not trained. However, these results will not be accurate because the network was not trained to interpolate between two discrete input values, so caution must be taken. Therefore, human logic is injected into the program, and the procedures followed by a pilot in actual flight are used. This is done by using weighted averages.

Weighted averaging is performed using the following procedure. Suppose the Fokker F100 is climbing with an initial climb weight W, at average Mach number M, such that W has a value between W1 and W2, and M has a value between M1 and M2, i.e. $W1 < W \leq W2$ and $M1 < M \leq M2$. Also the initial climb altitude is assumed to be A1 and the target altitude is assumed to be A2.

Note that M1, M2, W1, and W2 are input values for which the corresponding information can be found in the flight manual without using any form of interpolation.

To estimate the fuel consumption in this case, the following steps have to be taken:

1. Calculate fuel consumption for climb from A1 to A2 with initial weight W1 using Mach numbers M1 and M2. The resultants is F_{W1}. Using weighted average for Mach number, fuel consumption of an aircraft with initial weight W1 climbing from A1 to A2 is F_{W1}, which can be written as:

$$F_{W1} = \left[F_{(W1, M1, A2)} + \frac{F_{(W1, M1, A2)} - F_{(W1, M2, A2)}}{M1 - M2}(M - M1)\right] - \left[F_{(W1, M1, A1)} + \frac{F_{(W1, M1, A1)} - F_{(W1, M2, A1)}}{M1 - M2}(M - M1)\right] \tag{4.1}$$

2. Replace W1 by W2 and use the same procedure shown above. Fuel consumption for climb with W2 will be F_{W2}, which can be written as:

$$F_{W2} = \left[F_{(W2, M1, A2)} + \frac{F_{(W2, M1, A2)} - F_{(W2, M2, A2)}}{M1 - M2}(M - M1)\right] - \left[F_{(W2, M1, A1)} + \frac{F_{(W2, M1, A1)} - F_{(W2, M2, A1)}}{M1 - M2}(M - M1)\right] \tag{4.2}$$

Finally, apply the same procedure for weight. The final fuel consumption F for an aircraft with initial climb weight W, average climbing Mach number M, climbing from A1 to A2 is:

$$F = F_{W1} + \frac{F_{W2} - F_{W1}}{W2 - W1}(W - W1) \tag{4.3}$$

The climb distance calculation is basically the same as climb fuel calculation except that a new set of weights and biases are used instead. The purpose of calculating the required climb distance is to check if the aircraft will be able to climb to a certain altitude in a certain distance as the desired trajectory requires. In this fashion one can detect if the input trajectory violates the performance limits of a particular aircraft and requires the user to correct the trajectory manually. On the other hand if the aircraft is able to out-perform the desired trajectory such that the target altitude can be attained before reaching the distance limit then the program will add a segment of cruise from the point where the required altitude is attained to the next way point. The extra fuel consumption for this portion is calculated based on the weight of the aircraft after the particular climb segment and the target Mach number of the next way point. Figure 4.5 is used to illustrate the self-adjustment feature of the program.

4. The techniques used to calculate fuel consumption for the cruise segment are very similar to the ones used in the climb phase. The only difference is that data for altitude is in discrete format since performance data is usually stated in terms of Specific Air Range - SAR (see Figure 5.2 in the next chapter). An example of fuel burn calculation for an aircraft cruise at altitude A, with cruise Mach number M and weights W is shown as follows:

Suppose $W1 < W \le W$ and $A1 < A < A2$

where W1, W2, A1 and A2 are performance parameters which can be used to determine the specific air range (SAR) without interpolation. In a similar vein to step(1) we calculate the specific air range for both A1 and A2, such that SAR_{W1} and SAR_{W2} can be written as:

$$SAR_{W1} = \left[SAR_{(W1,M,A1)} + \frac{SAR_{(W1,M,A1)} - SAR_{(W1,M,A2)}}{A1 - A2}(A - A1)\right] \qquad (4.4)$$

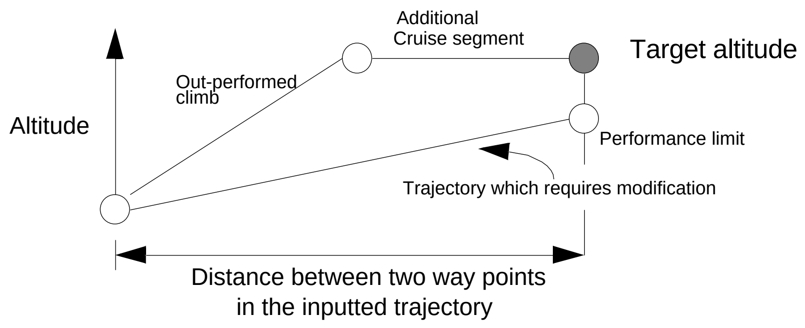
and

$$SAR_{W2} = \left[SAR_{(W2,M,A1)} + \frac{SAR_{(W2,M,A1)} - SAR_{(W2,M,A2)}}{A1 - A2}(A - A1)\right] \qquad (4.5)$$

Therefore, the specific air range SAR for the aircraft cruising at an altitude A, with cruise Mach number M and weights W can be expressed as:

$$SAR_{W} = \left[SAR_{(W1,M,A1)} + \frac{SAR_{W2} - SAR_{W1}}{W2 - W1}(W - W1 \right. \qquad (4.6)$$

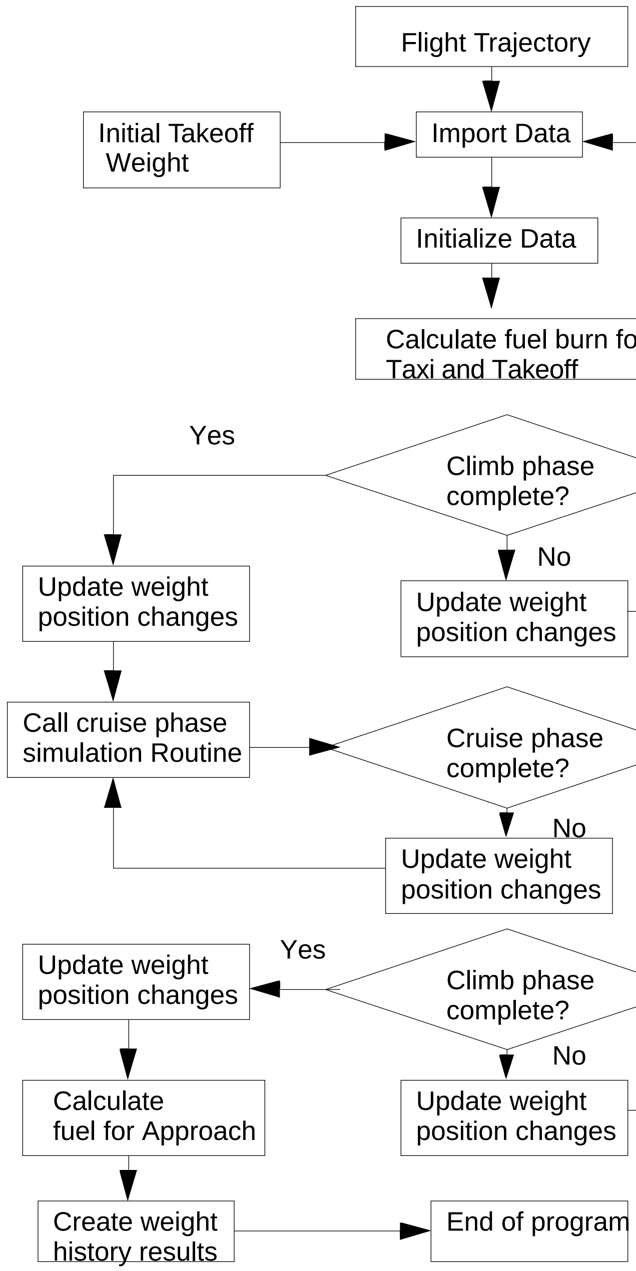
Figure 4.5 Possible Aircraft Climb Procedures.


Additional Cruise segment
Target altitude
Out-performed climb
Altitude
Performance limit
Trajectory which requires modification
Distance between two way points in the inputted trajectory

5. The computation procedure of fuel consumption for the descent phase is the same as shown in the climb phase and it will not be repeated here. A complete flow diagram summarizing the computational procedures outlined in this section is shown in Figure 4.6. Note that implementation of this algorithm in SIMMOD is possible if one parses the outcome file generated in every simulation. This procedure

will be further explained in the following chapter in Section 5.4.The process here is to keep track of the microscopic activities of every aircraft for all link traversals, ground and airspace hold actions and airspace path stretching procedures.

Figure 4.6 Flow Chart for Neural Network Aided Fuel Consumption Model.


Flight Trajectory
Initial Takeoff Weight
Import Data
ISA Condition
Initialize Data
Calculate fuel burn for Taxi and Takeoff
Update weight position changes
Call climb phase simulation Routine
Climb phase complete?
Yes
No
Update weight position changes
Update weight position changes
Call cruise phase simulation Routine
Cruise phase complete?
Yes
No
Update weight position changes
Update weight position changes
Call descent phase simulation Routine
Climb phase complete?
Yes
No
Update weight position changes
Update weight position changes
Calculate fuel for Approach
Create weight history results
End of program

CHAPTER 5 Discussion of Results

This chapter presents the results generated by the neural network aided fuel consumption model described in Chapters 3 and 4. The Fokker 100, a medium size, high by-pass ratio turbofan powered aircraft was used as the test aircraft. Results generated from neural network aided model are compared with the actual performance provided in the flight manual While the results presented in this chapter are complete for the F100, our study suggests that high performance turbopropeller driven aircraft such as many commuter aircraft operating today can also be modeled with the neural network topology used in this paper. In this new instance the results are comparable in precision to those obtained for the turbofan powered aircraft.

5.1 Training Results

The purpose of training the neural network is to create a set of weight matrices and bias vectors (as explained in Chapter 2 of this report) that make up a mathematical model to predict fuel consumption under any set of aircraft flight conditions. These weighs and biases are somewhat equivalent to the regression constants found in may non-linear multivariate estimation models and thus can be easily incorporated in any programming environment that supports array manipulation, including SIMSCRIPT II.5, the native computer simulation language used in SIMMOD.

Training data sets were obtained by either digitizing the flight manual of the test aircraft or through a simple compilation of various table functions representing various aircraft performance characteristics spanning the complete flight envelope of the aircraft in question. Sample flight information characteristics for the Fokker 100 aircraft are shown in Figures 5.1 through 5.3 for climb, cruise and descent conditions, respectively. Other flight phases such as takeoff and climbout to 1,500 ft., taxiing and loiter (i.e., holding) conditions are simpler to analyze because the relationships are in general linear and can perhaps be approximated with a simpler model (i.e., a simple regression model). In fact, these simpler conditions can also be analyzed with lower order neural networks (2 layers and just 4-5 neurons in each layer).

Close examination of Figures 5.1through 5.3 illustrates that under various speed, altitude, weather, and weight conditions any high-performance aircraft exhibits quite a bit of nonlinear behavior in the fuel consumption parameter. This fact coupled with numerous charts usually available for a single aircraft make the use of neural networks desirable.

The sizes of the various training data sets used in the neural network learning process are shown in Table 5.1. Note that all datasets used varied in length according to the characteristic non-linear behavior observed. For example, the cruise phase, while non-linear in nature is more predictable with fewer points that the descent phase because the aircraft velocity profile changes more drastically in a descent from 37000 ft. that cruising at the same altitude. The size of each set was derived directly from performance curves supplied by the manufacturer (Fokker) and reflect knowledge that we obtained through several iterations in the modeling process.

.

TABLE 5.1. Training Data Sets.

Flight Phase	Number of Training Points
Takeoff and Climb-Out	8 (linear)
Climb to Cruise Altitude	852 (Fuel) 854 (Distance)
Cruise	805
Descent	1210 (Fuel) 288 (Distance)

Figures 5.4 to 5.11 show the plots of the training data sets used in this research project for climb, cruise and descent phases respectively. In particular, Figures 5.4-5.5 depict in two and three dimensions data points used to train the climb phase neural network,. Examination of Figure 5.5 reveals that four climb profiles (as stated in the flight manual) are typically employed in the operation of this aircraft for regular climbs (i.e., mach numbers ranging from a low 0.65 to a high speed climb at 0.75 mach). Note also that pressure altitudes vary from 1500 ft. (corresponding to the end of the climb out segment modeled in another dataset) to 40,000 ft. corresponding to the maximum certified service ceiling of this aircraft. In general, variations in fuel consumption are also very sensitive to climb weight and atmospheric conditions prevalent along the climb path. These are represented in the variations along the z-axis in Figure 5.5. For this particular aircraft a low climb weight of 58,000 lb is possible with minimal payload (i.e., short stage lengths) up to a maximum climb weight of 98,000 lb (accounting some minor loses for taxi-

ing and climb-out fuel consumption). Temperature profiles contained in the flight manual accounted for ISA (International Standard Atmosphere) and ISA+10 conditions. Note that in general the more information about varying temperature conditions is presented in the manual the more accurate predictions are possible using the neural network. A minimum data set to predict climb fuel consumption incorporates these four variables: a) pressure altitude at top of climb, b) temperature, c) mach number, and d) aircraft weight.

Figure 5.1 Climb Fuel Information (source: Fokker 100 Performance Manual).

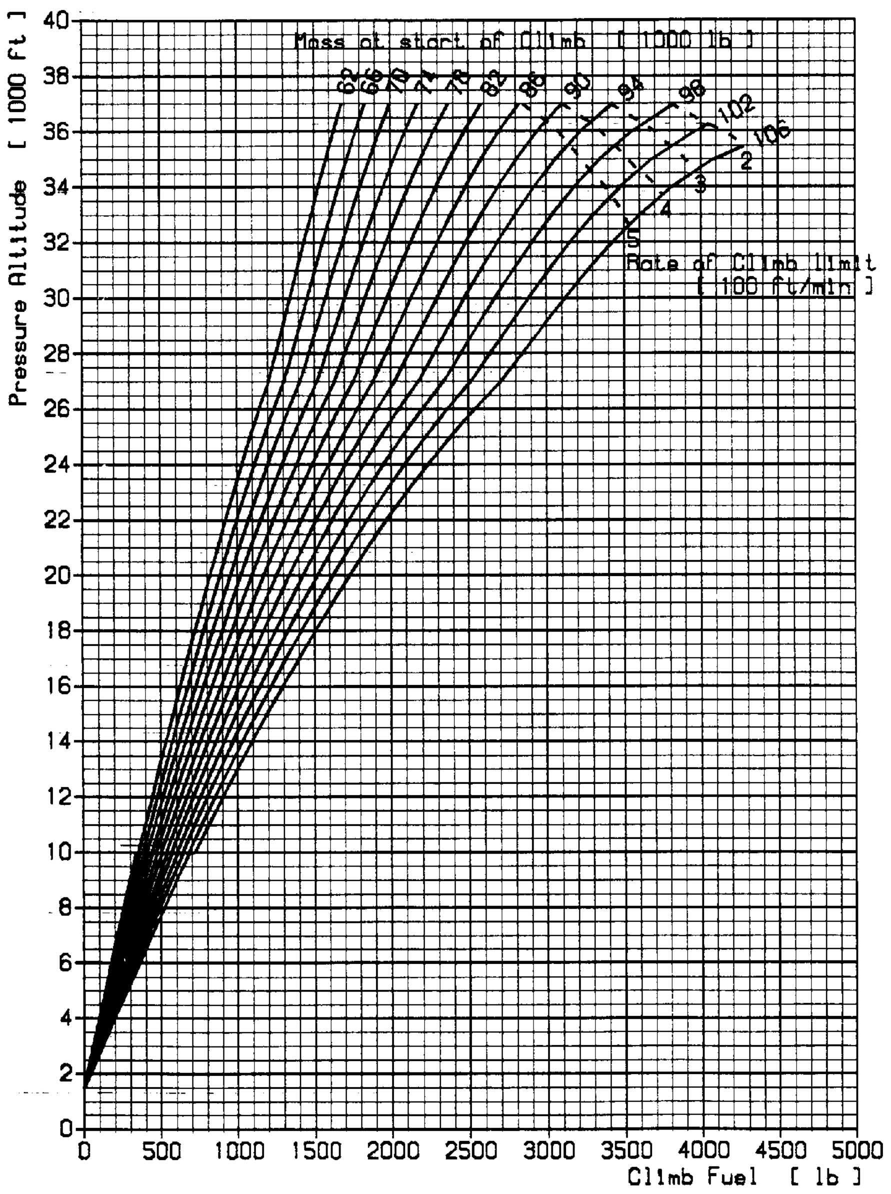
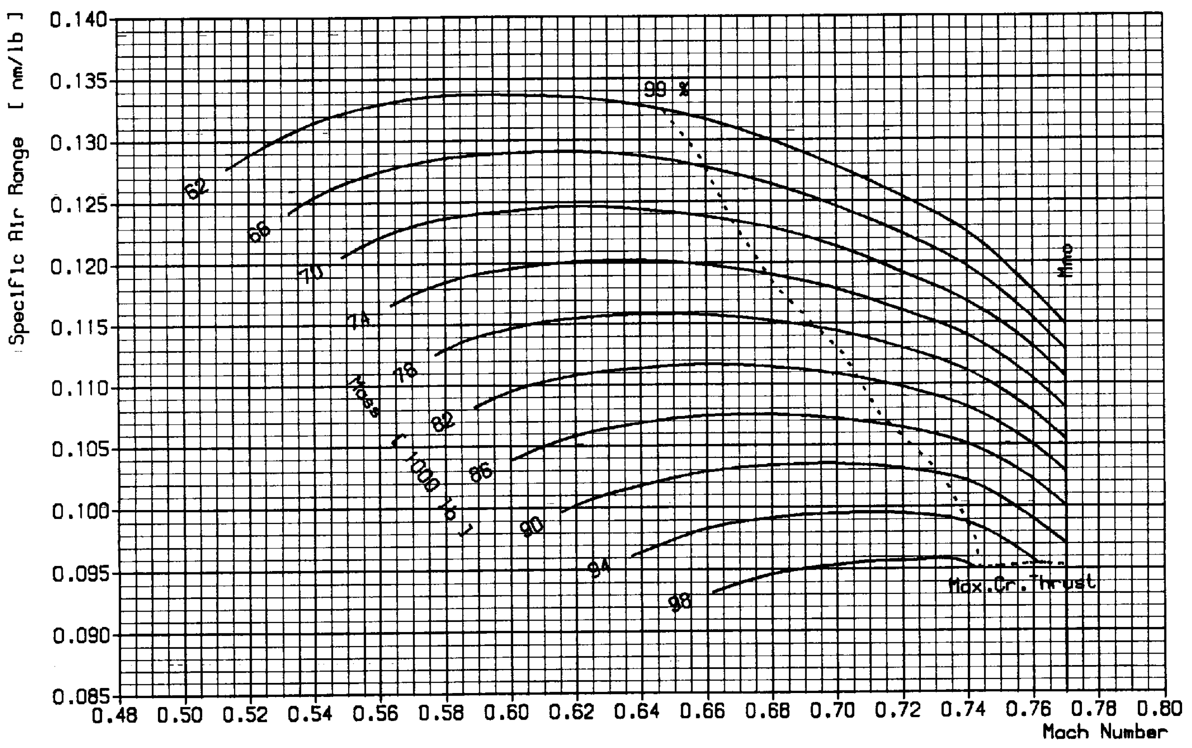


Figure 5.2 Specific Air Range Information (source: Fokker 100 Performance Manual).



(for ISA+10° C subtract 0.4% of the SAR values presented)

Figure 5.3 Descent Fuel Performance (source: Fokker 100 Performance Information).

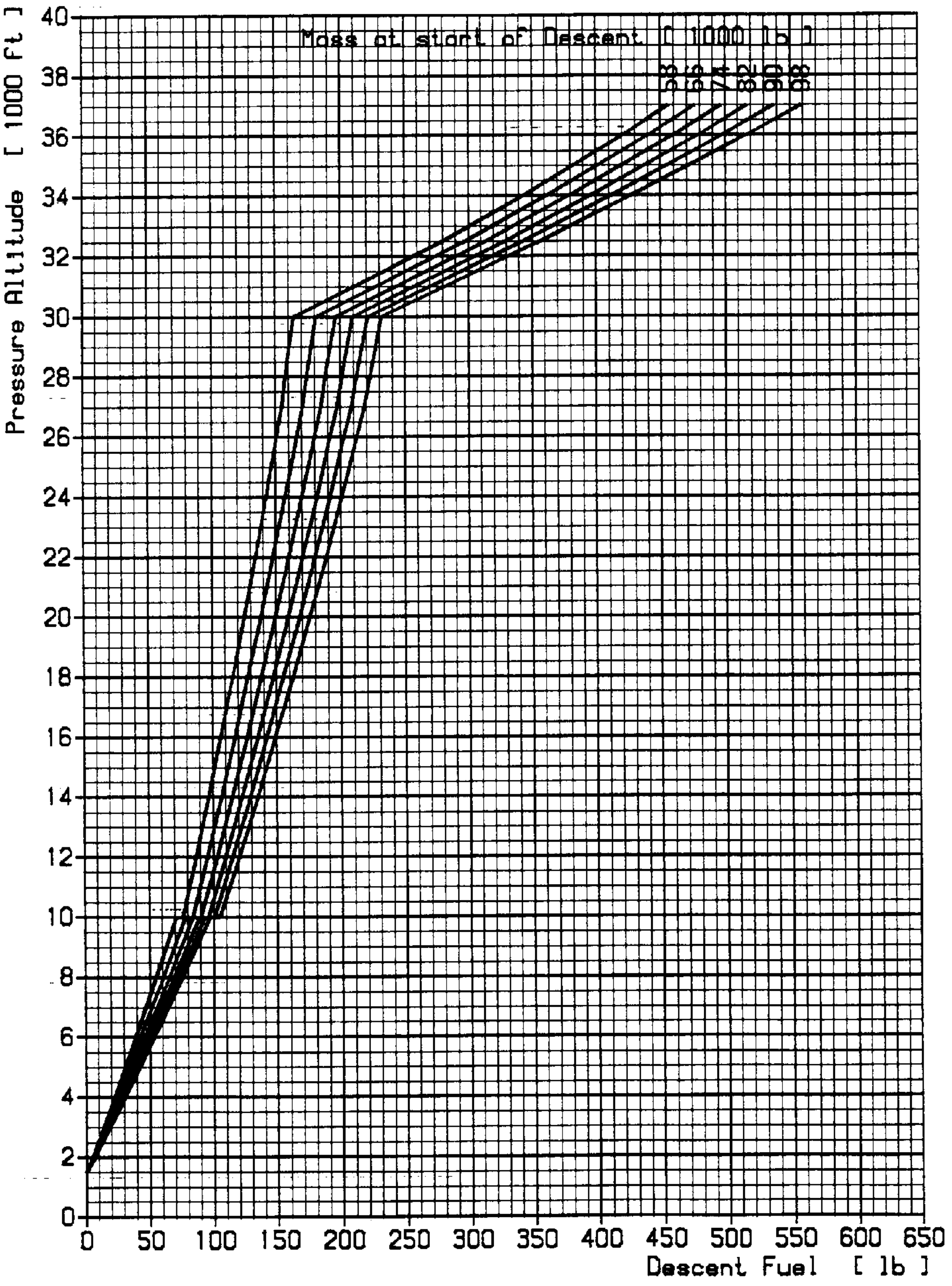


Figure 5.4 Two-Dimensional View of Climb Fuel Database.

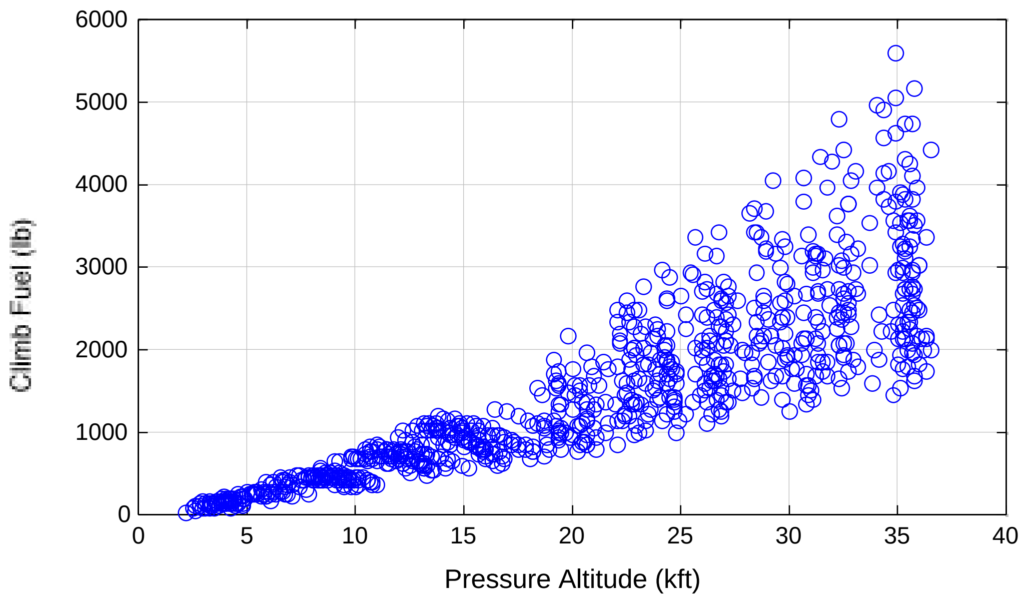


Figure 5.5 Three Dimensional View of Training Climb Fuel Database.

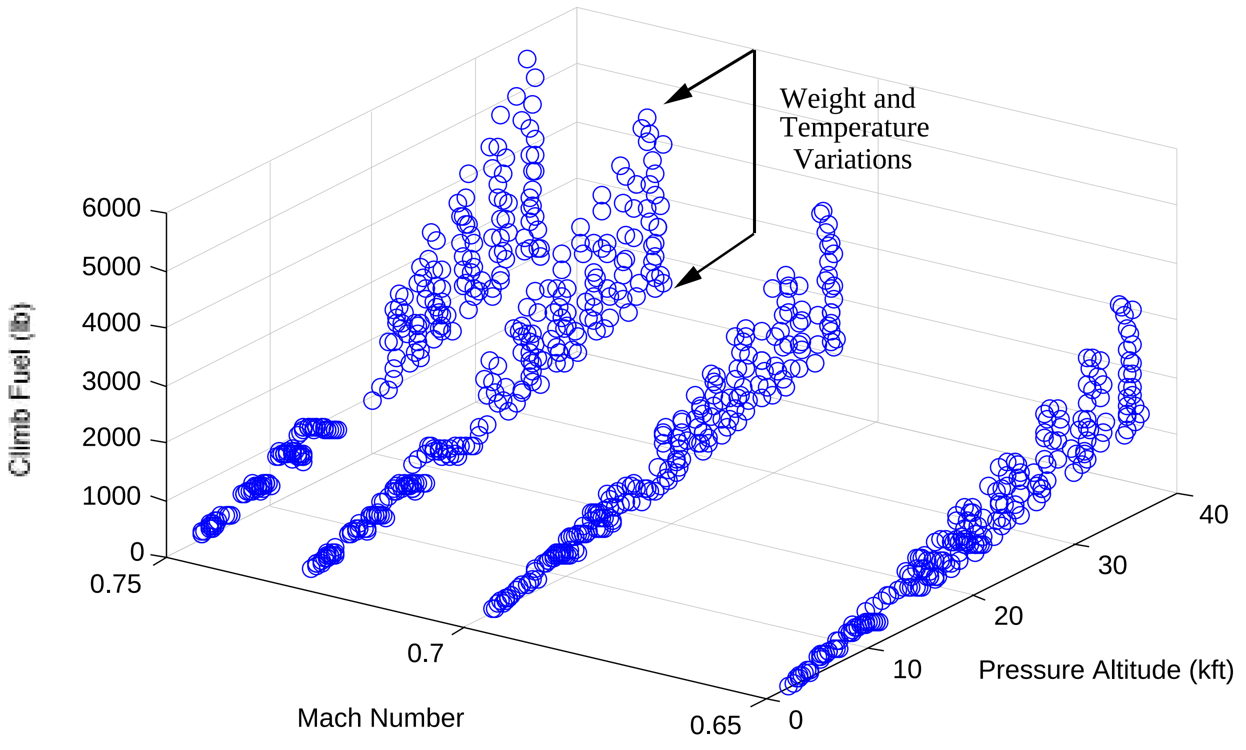


Figure 5.6 Two-Dimensional View of Climb Distance Database.

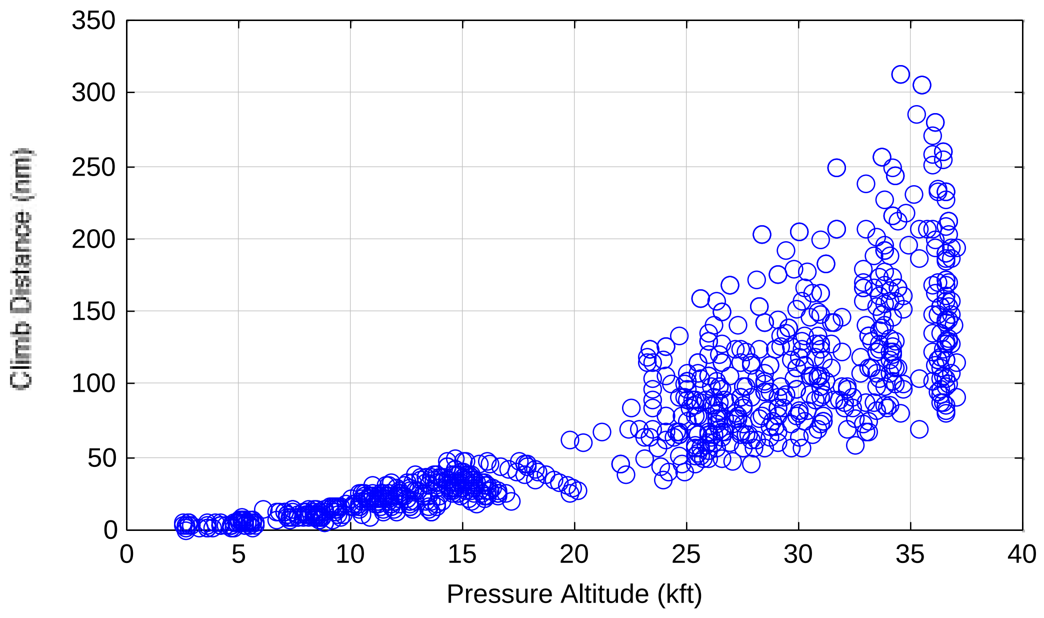


Figure 5.7 Three-Dimensional View of Climb Distance Database.

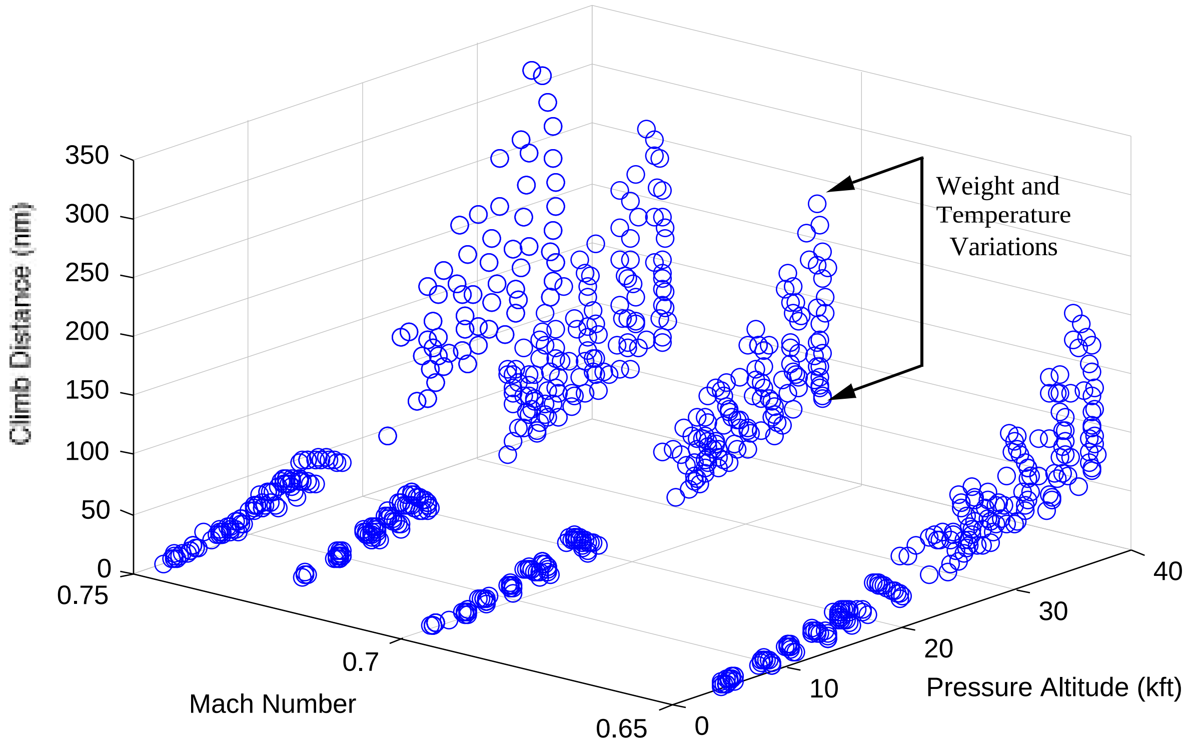


Figure 5.8 Two Dimensional View of Cruise Segment Specific Range Database.

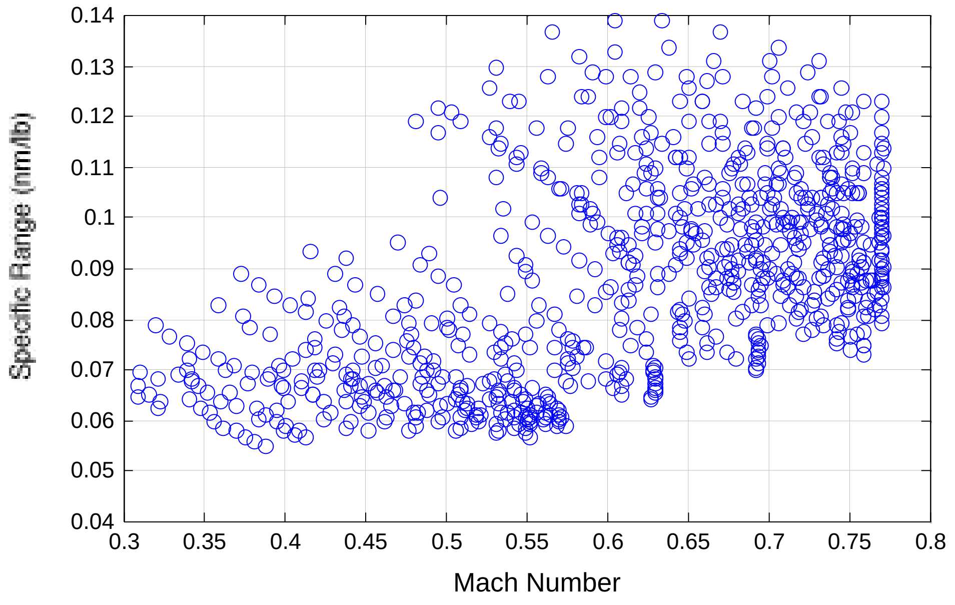


Figure 5.9 Three Dimensional View of Cruise Segment Specific Range Database.

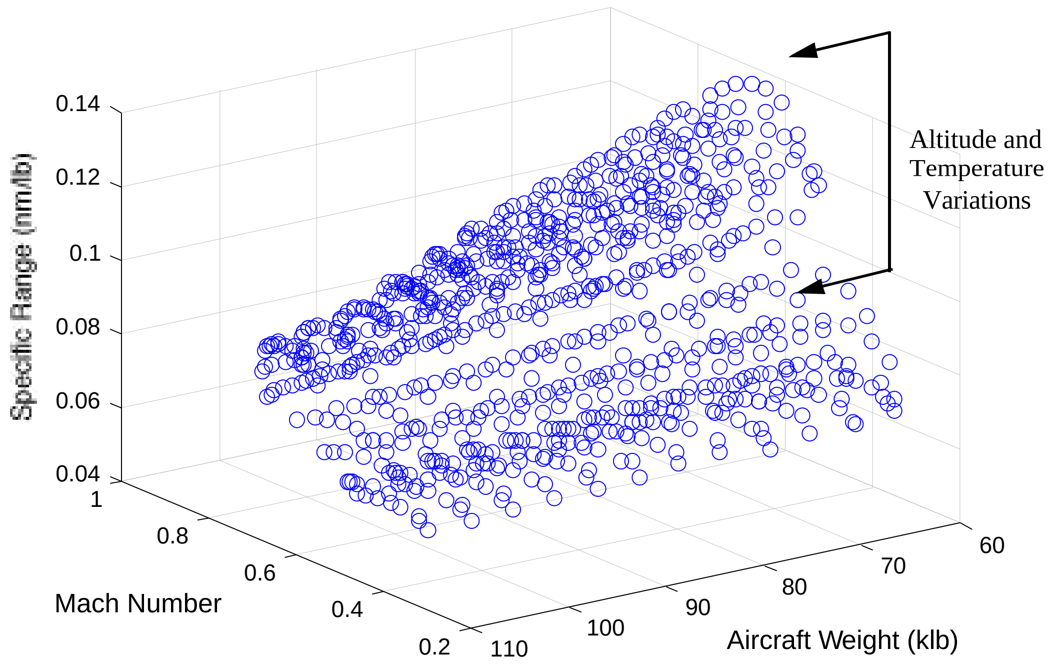


Figure 5.10 Two Dimensional View of Descent Fuel Database.

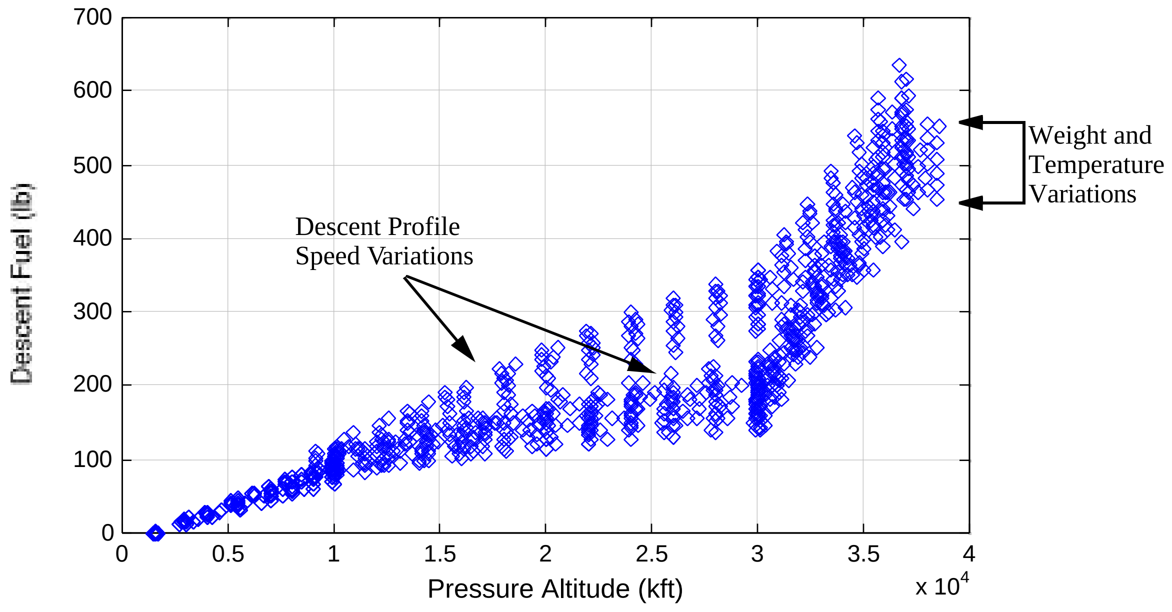
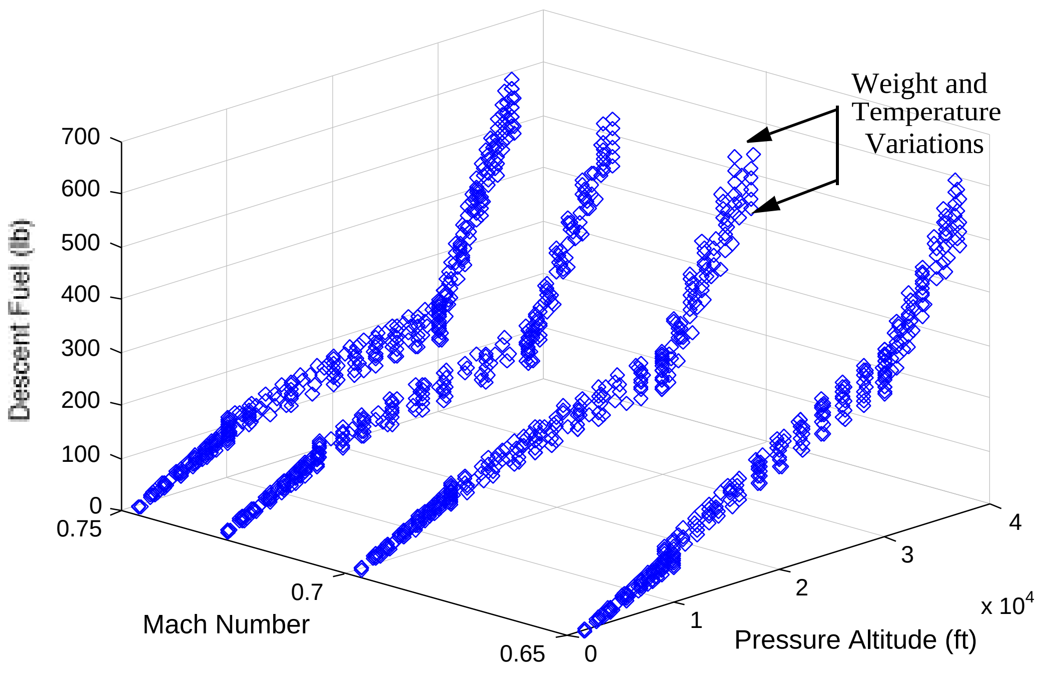


Figure 5.11 Three Dimensional View of Descent Fuel Database.



Figures 5.5 through 5.11 demonstrate that the training data sets cover all the typical performance points in the climb, cruise and descent phases of the mission profile. For the climb phase, fuel consumption and distance estimation are trained for target altitudes ranging from 1,500 ft. to 40,000 ft. For the cruise phase, specific air range (the distance covered for every pound of fuel consumed) is trained for cruise Mach numbers ranging from 0.3 to 0.77 at various altitudes and temperature conditions. Note that training data should be selected carefully such that a wide range of velocities and altitudes are included. Selection of training data is a very important step; whether the neural network can be used to predict fuel consumption accurately depends on how well the trained network can generalize the input data. A good way to check whether the input data is well distributed is through the use of scattered plots for various input parameters as shown in Figures 5.4, 5.6, 5.8, and 5.10.

Sample weight matrices and bias vectors are found for the Fokker 100 in Appendix B. It is worth mentioning that once these matrices and vectors are found and stored, they become a data base which can be utilized in *any* other simulation programs, and do not depend on MATLAB. The computational procedure for doing this has been illustrated in Section 3.7 of Chapter 3.

5.2 Testing Results

The generalization of a neural network involves testing various data sets into the trained neural network to assure the reliability of the fuel consumption estimations. Without doubt, this is one of the most important pieces in any neural network modeling effort. As stated in Section 5.1, whether the network is reliable depends on how well the trained network can generalize the inputs. In other words, a well trained and constructed network should be able to predict fuel consumption conditions dissimilar to those used in the training procedure. The test results were evaluated based on the mean error and the standard deviation of error (see Chapter 3). The results are examined and presented in this section.

Table 5.2 lists all the data sets used in the generalization step for various flight phases. In this research it was decided that the number of data points in the generalization procedure should be equal or more than the number of points used in the training procedure. This heuristic rule was used throughout the generalization procedure to give sound statistical measures of the neural network errors compared to the actual flight performance data. In all cases t-tests performed on the datasets demonstrated that the mean errors came very close to be zero and thus the null hypothesis was an all cases accepted.

Figure 5.12 shows a plot of actual versus predicted climb fuel as a function of top-of-climb altitude. From this figure, it can be seen that the fuel consumption of the aircraft increases in a non-linear manner as the target altitude increases. Similar non-linearities are observed for variations of climb distance and target altitude (see Figure 5.14). Figure 5.13 shows graphically the errors predicted by the three-layer climb neural network. The mean and the standard deviation of the fuel estimation errors are 6.75 and 30.19 lb., respectively. To bring these numbers in perspective, the average computer fuel consumption in the complete database was 1,528 lb. thus indicating a very small average percent error. In fact the climb neural network was trained for nine hours for this aircraft on a 250 MHz PowerPC computer to accuracies as low as .013% (RMS). Table 5.3 shows the mean errors and their respective standard deviations for all phases of flight computed using neural networks.

Figures 5.14 and 5.15 show the errors obtained for the climb distance computation. Climb distance was used as a parameter to demonstrate that neural networks can in fact predict other aircraft performance measures such as climb and descent times that are also important in fast time simulation models. The climb distance error computed was 0.377% with a standard deviation of 0.305%. Examination of Figure 5.15 shows that maximum dispersed errors of up to 10 nautical miles are possible for heavy F100 climbing to high flight levels (i.e., 95,000 lb flying at flying at 37,000 ft, for example). Note however, that in most cases the climb distance error is confined to ∓ 5.4 nautical miles ninety nine percent of the time.Once again, this clearly illustrates that neural networks can yield very reasonable values of fuel consumption for any high performance aircraft.

TABLE 5.2. Neural Network Testing Data Sets.

Flight Phase	Number of Testing Points
Takeoff and Climbout	Nor applicable (linear regression used instead)
Climb to Cruise Altitude	850 (Fuel) 850 (Distance)
Cruise	805
Descent	1210 (Fuel) 140 (Distance)

TABLE 5.3. Summary of Errors for All Phases of Flight.

Flight Phase	Mean Error (%)	Standard Deviation (%)	Null Hypothesis (t-test at $\alpha = 0.01$)
Climb			
• Distance	0.377	0.305	Accept
• Fuel	1.026	0.190	Accept
Cruise Specific Air Range	-0.034	0.334	Accept
Descent			
• Distance	1.760	1.860	Accept
• Fuel	1.423	1.177	Accept

Figures 5.16 and 5.17 show the cruise phase fuel consumption results. Figure 5.16 shows a plot of Specific Air Range (SAR) versus cruise Mach number.This representation is typical in flight performance

manuals of high performance aircraft. This figure encompasses many charts and diagrams contained in the flight performance manual and is also known as the “flight envelope” of the aircraft. In our modeling procedure we selected data points so as to include all possible flight conditions of the aircraft to make sure that the altitude and Mach schedule presented does not violate the aircraft performance limits. The errors between the estimated and actual fuel burn are shown in Figure 5.17. Note that cruise fuel burn predictions are fairly accurate with a mean estimation error of -0.034% and a standard deviation of 0.334%. This is a result of the near quadratic behavior of SAR with Mach number (see Figure 5.2).

Figure 5.18 depicts a plot of estimated versus actual descent fuel resulting from the neural network estimation. Figure 5.19 shows a histogram with the errors resulting from the fuel consumption estimation and shows that most fuel estimation errors are contained to less than ∓7.4 lbs. ninety nine percent of the time. The percent errors for mean and standard deviation are shown in Table 5.3.

Figure 5.20 shows a frequency plot of actual and estimated fuel consumption for the descent phase. This plot demonstrates the general accuracy trends of the neural network estimation procedure. A Chi-Square test of the descent fuel data indicates that both distributions fit well at $\alpha = 0.01$.

5.3 Correlation of Neural Network Fuel Consumption Results

The final step of this research project is to develop a computer program which would perform the following tasks for the test aircraft:

1. Perform feed-forward simulation using weight matrices trained.

2. Calculate fuel consumption the sample aircraft for complete missions and compare the neural network results with actual data (i.e., flight performance data).

A simulation program has been developed to test the validity of the results for a complete flight path; the results obtained are shown in this section.

The input trajectory was developed according to the instructions provided in the flight manual of the test aircraft, the Fokker F100. Each flight considered all typical segments of flight: a) taxi, b) takeoff and climbout, c) climb to cruise altitude, d) cruise, e) descent from cruise altitude, f) landing and taxing at destination airport. Six short stage length trips covering the East Coast of the U.S. were selected to verify the accuracy of the model developed. Trips were chosen as they are typically flown by this type of aircraft in NAS, under ISA+10 temperature conditions and respecting ATC control vertical separation procedures. The aircraft initial aircraft ramp gross weight was set to 95,000 lb for all analyses representing 95% of the maximum allowable. In all cases ample fuel reserves were available at this operating condition. All trips were modeled using Free Flight trajectories (i.e., pseudo globe circle routes with constant heading waypoint legs of 100 nautical miles). A typical flight trajectory illustrating a flight from Dallas-Forth Worth to Miami is shown in Figure 5.21.

Figure 5.12 Estimated and Actual Climb Fuel Results.

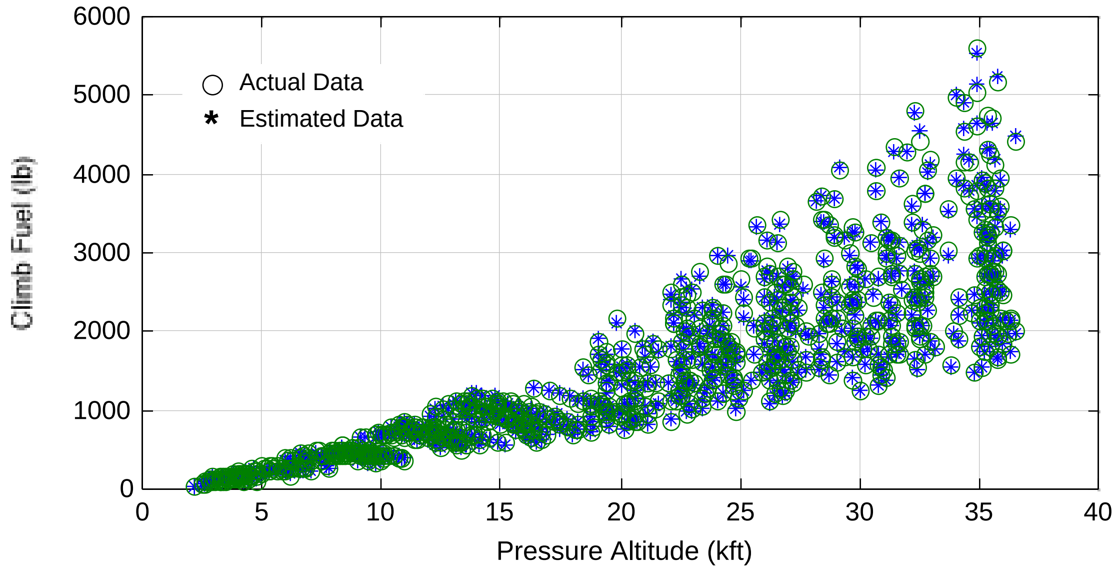


Figure 5.13 Climb Fuel Error Histogram.

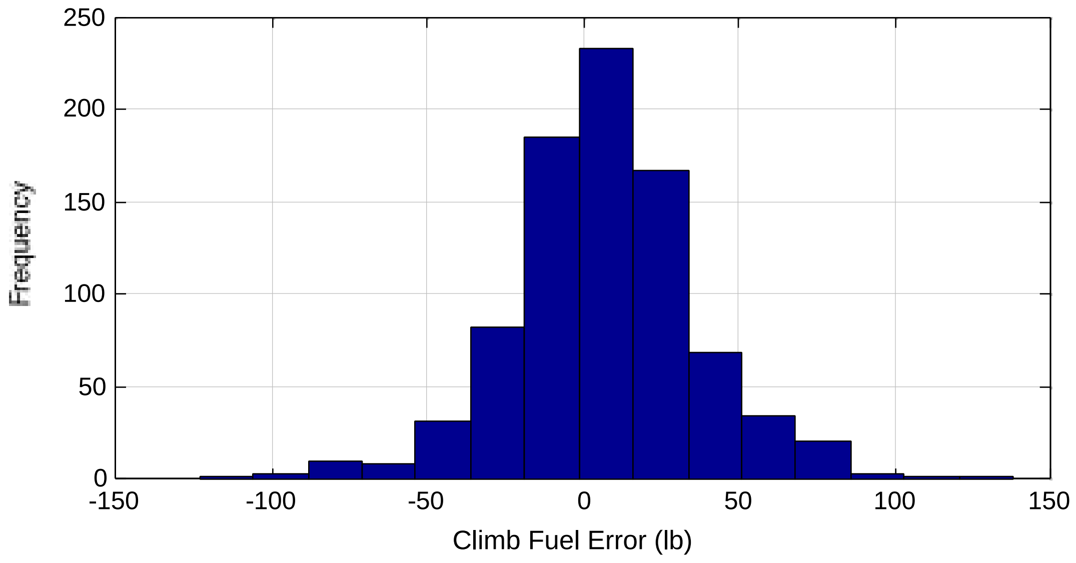


Figure 5.14 Climb Distance Correlation of Results.

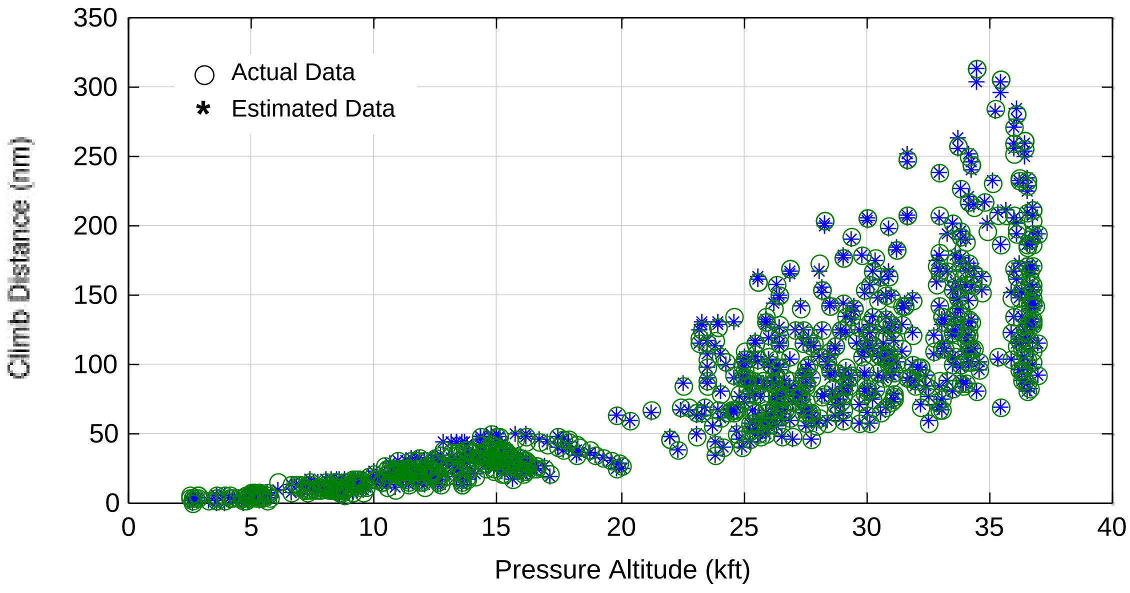


Figure 5.15 Climb Distance Estimation Errors.

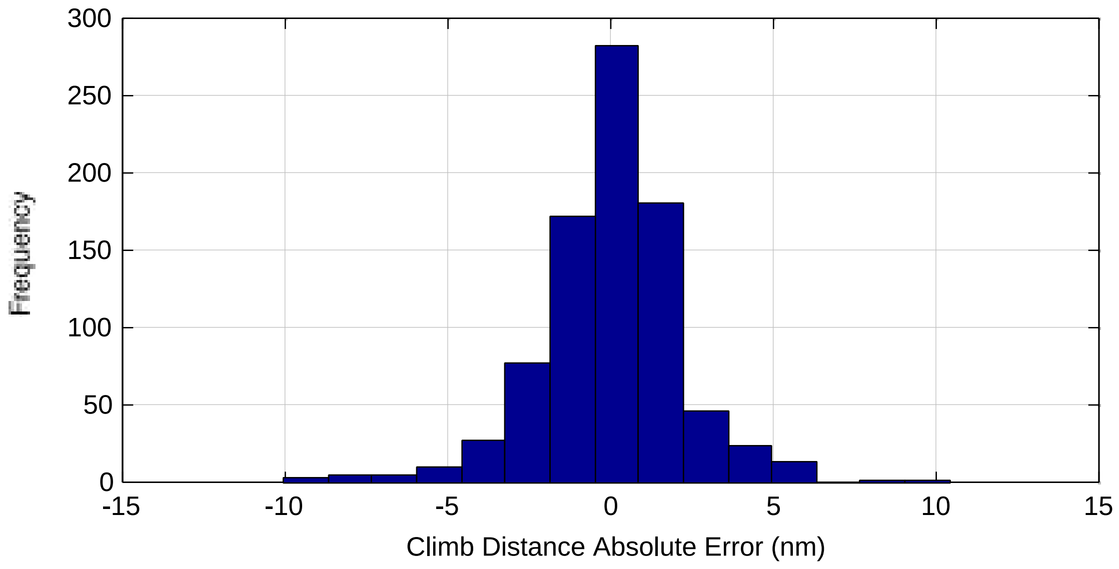


Figure 5.16 Specific Range Generalization Results.

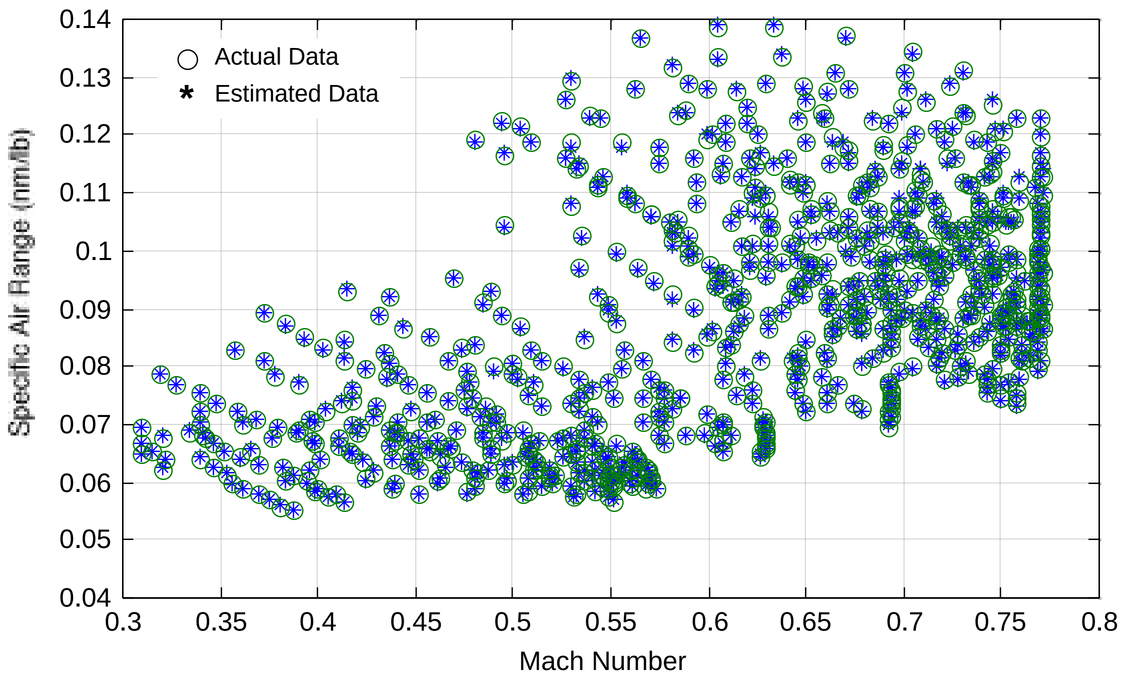


Figure 5.17 Specific Range Error Histogram.

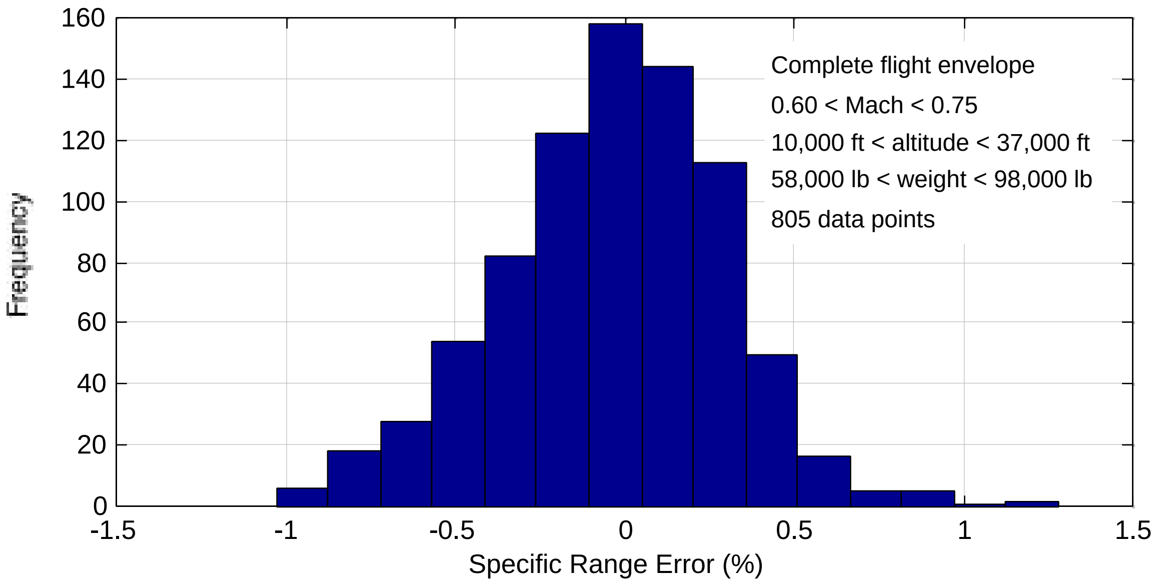


Figure 5.18 Descent Fuel Predicted vs. Actual.

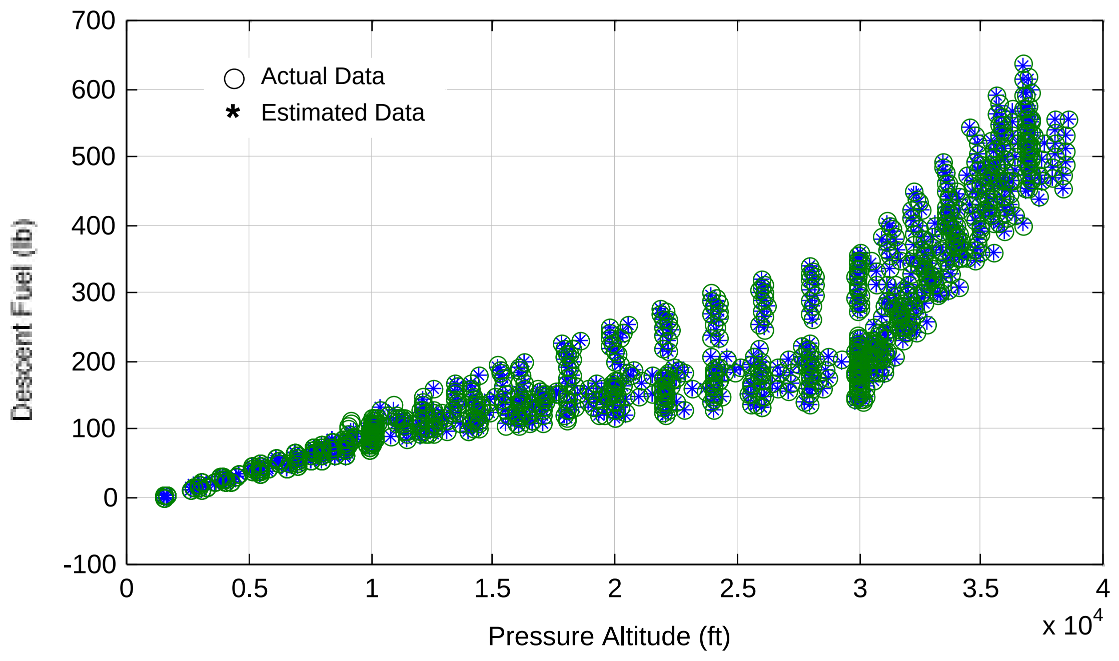


Figure 5.19 Descent Fuel Error Histogram.

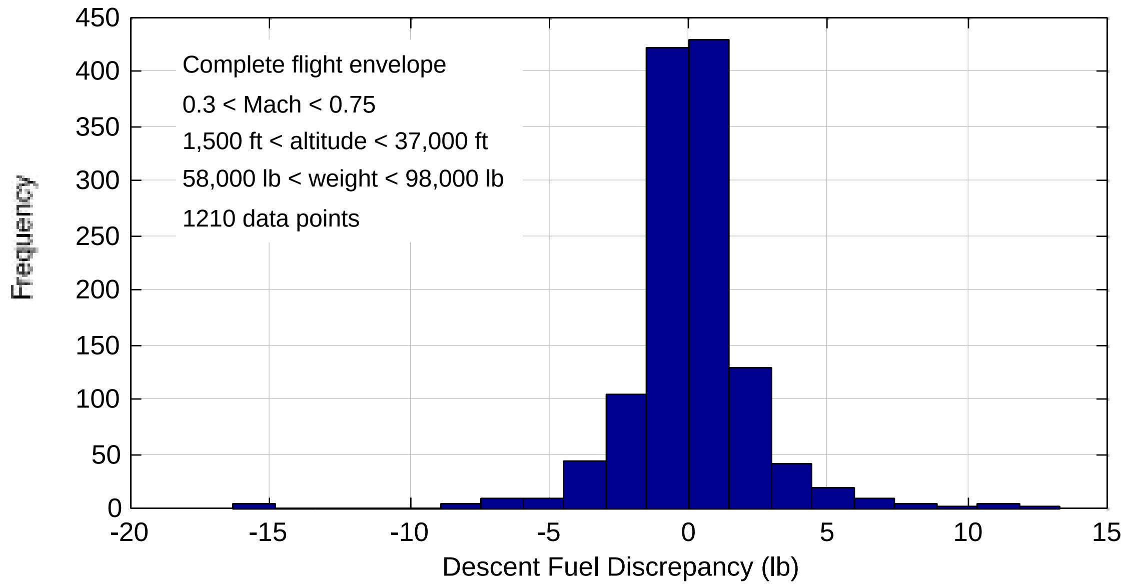
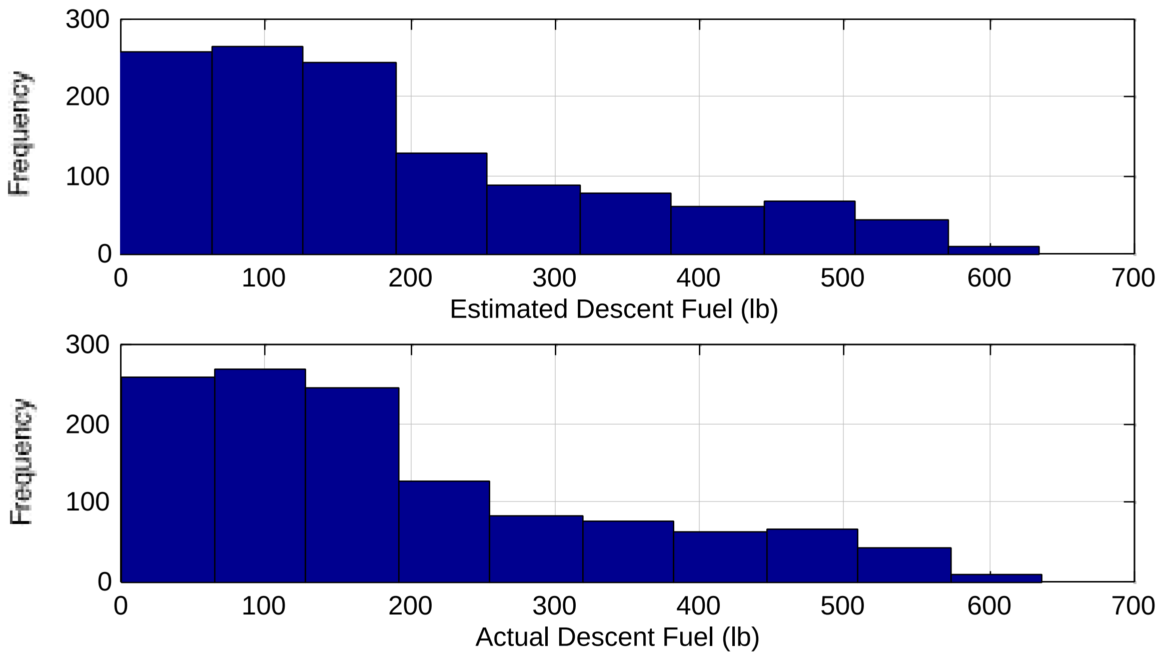


Figure 5.20 Frequency Distributions for Actual and Computed Descent Fuel.

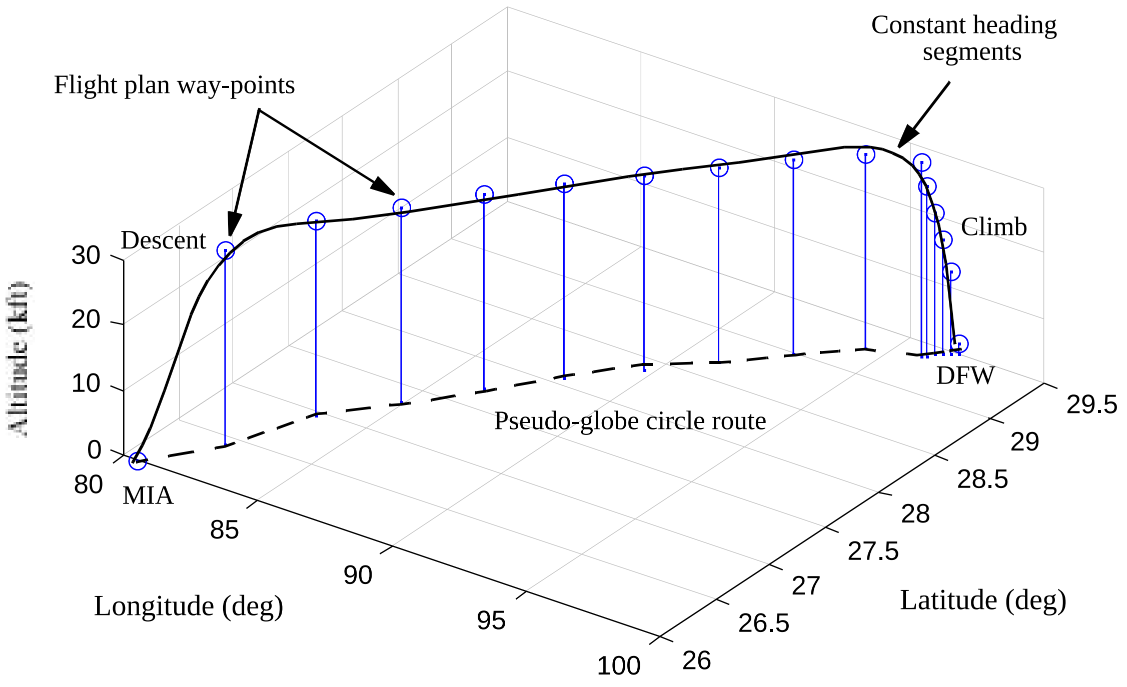


The output of the trajectory simulation program was fed to the neural networks for climb, cruise and descent to estimate the fuel consumption for each flight leg. These results were compared with the performance values contained in the flight manual. The results obtained in this comparison were very encouraging. An average fuel estimation error of less than 0.8% was found in the computations with a maximum error of 1.61%. Table 5.4 presents the flight plan characteristics of the twelve flight plans selected for this study.

5.4 SIMMOD and SIMMOD 2000 Implementation Issues

The implementation of the neural network model can be carried out within the existing SIMMOD framework or in any future version of the model (called SIMMOD 2000 for this discussion). The basic requirements to implement a neural network model are much less restrictive than those found today in the SIMMOD fuel burn post-processor. For example, in the present implementation of the MITRE algorithm to estimate fuel burn the spacing between data points in the flight trajectory has to be tightly adjusted to less than 2,000 ft. (in the vertical dimension) to maintain reasonable accuracy. The proposed neural network model can be implemented with larger vertical spacing requiring less computational effort while maintaining a good level of accuracy. This is because the fuel consumption model using the neural network was generalized with absolute fuel consumption statistics represents altitude changes from sea level up to the cruising altitude. In this fashion large altitude changes are captured more accurately.

Figure 5.21 Sample Flight Plan Profile.



The existing input/output file structure used in SIMMOD can be utilized in the implementation of the neural network algorithm. Figure 5.22 illustrates graphically the current structure of the fuel burn post-processor model. Minor modifications to the input file structure (INP file in large central box) will be required to read weight and biases of the neural network files (routine INP.100.READ.FUELBURN.PARAMETERS in Table 5.5. This operation is analogous to reading large numbers of aircraft-engine specific constants as currently done in SIMMOD

Table 5.5 shows all pertinent routines associated with the fuel burn post-processor. For fuel consumption analysis alone minor changes to routines FBC.100.AIR.FUEL.BURN and FBG.100.GROUND.FUEL.BURN as these two serve the role of fuel burn calculators for airspace and ground actions, respectively. These routines are contained in submodules labeled FBC and FBG in Figure 5.22.

Figure 5.23 shows a proposed implementation methodology to embed the neural network fuel consumption model developed in this project into the current structure of SIMMOD. A new set of files (called Neural Network Fuel Consumption File in Figure 5.23) will be created to store weights and biases for a large aircraft population (100+ aircraft). Using the methodology outlined in Chapters 3 and 4 of this report we generate networks to cover detailed fuel consumptions for every phase of flight. Inside the fuel burn post-processor module (see Figure 5.23) submodules FBC and FBG are modified (routines FBC.100.AIR.FUEL.BURN and FBG.100.GROUND.FUEL.BURN) and labeled MFBC and MFBG.

TABLE 5.4. Flight Plans Used in the Correlation of the Neural Network Model (ISA+10).

Flight	Cruise Flight Level (FL)	Distance (nm) / Time (hr)	Flight Manual Fuel Burn (lb)	Neural Net Fuel Burn (lb)	Percent Difference (%)
ROA[a]-MDW[b]	280	448 / 1:08	6,457	6,546	1.37
	310	448 / 1:10	6,360	6,330	0.46
MIA[c]-DFW[d]	310	972 / 2:24	11,851	11,865	0.12
	350	972 / 2:13	11,510	11,544	0.29
ROA-LGA[e]	290	352 / 0:57	5,298	5,260	0.71
	330	352 / 0:58	5,343	5,429	1.61
ATL[f]-MIA	290	518 / 1:20	6,990	7,047	0.80
	330	518 / 1:21	7,009	7,082	1.04
ATL-DCA[g]	290	475 / 1:13	6,549	6,584	0.54
	330	475 / 1:14	6,590	6,654	0.97
ROA-ATL	280	310 / 0:51	4,938	4,998	1.30
	310	310 / 0:51	4,941	4,933	0.15

a. ROA - Roanoke Regional Airport (Virginia)
b. MDW - Midway Airport (Illinois)
c. MIA - Miami International (Florida)
d. DFW - Dallas-Forth Worth International (Texas)
e. LGA - Laguardia Airport (New York)
f. ATL - Atlanta Hartsfield International Airport (Georgia)
g. DCA - National Airport (Virginia)

5.5 Neural Network Fuel Burn Correction in the Presence of Winds

All performance data points contained in a typical flight manual refer to still air performance conditions. For example, Figure 5.2 illustrates typical SAR profiles with zero wind conditions. Since still air conditions seldom exist is necessary to adjust the resulting values of the neural network fuel consumption model for changing wind conditions throughout the entire flight. One simple correction factor to be introduce here is the well known concept of SAR shift due to enroute wind patterns. An approximation to account for variable winds enroute is,

$$SAR_w = SAR_0\left(1 + \frac{w}{a_0 M\sqrt{\frac{T_h}{T_0}}}\right) \tag{5.1}$$

where,

SAR_w is the specific air range in the presence of winds, SAR_0 is the specific air range without winds, w is the wind component parallel to the flight path (negative for headwinds and positive for tailwinds) and the denominator is the true airspeed of the aircraft in question in terms of mach number (M), speed of sound at sea level conditions (a_0) and temperature ratio T_h/T_0. Figure 5.24 illustrates the variations in SAR for the Fokker 100 as predicted by the neural network at 30,000 ft. and ISA conditions for various winds enroute. This characterization is important because most of the fast-time simulation models incorporate wind patterns in the form of table look-up functions. In the particular case of SIMMOD a series of wind sets are specified by the user for various families of links in the airspace structure. Incorporation of wind requires minor modifications to routines FBC.100.AIR.FUEL.BURN and FBC.900.INTERMEDIATE.VAUES.

Figure 5.22 Current SIMMOD Fuel Burn Post-processor Input/Output File Structure.

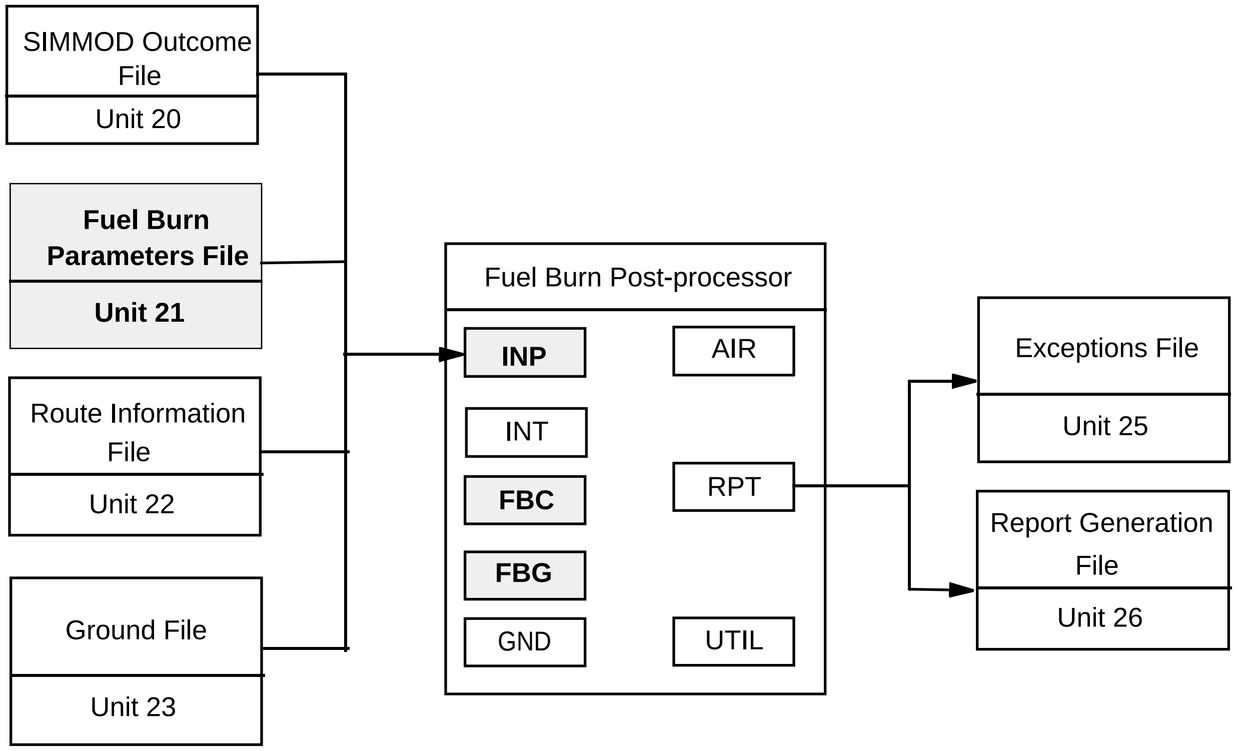


Figure 5.23 New SIMMOD Fuel Burn Post-processor Input/Output File Structure.

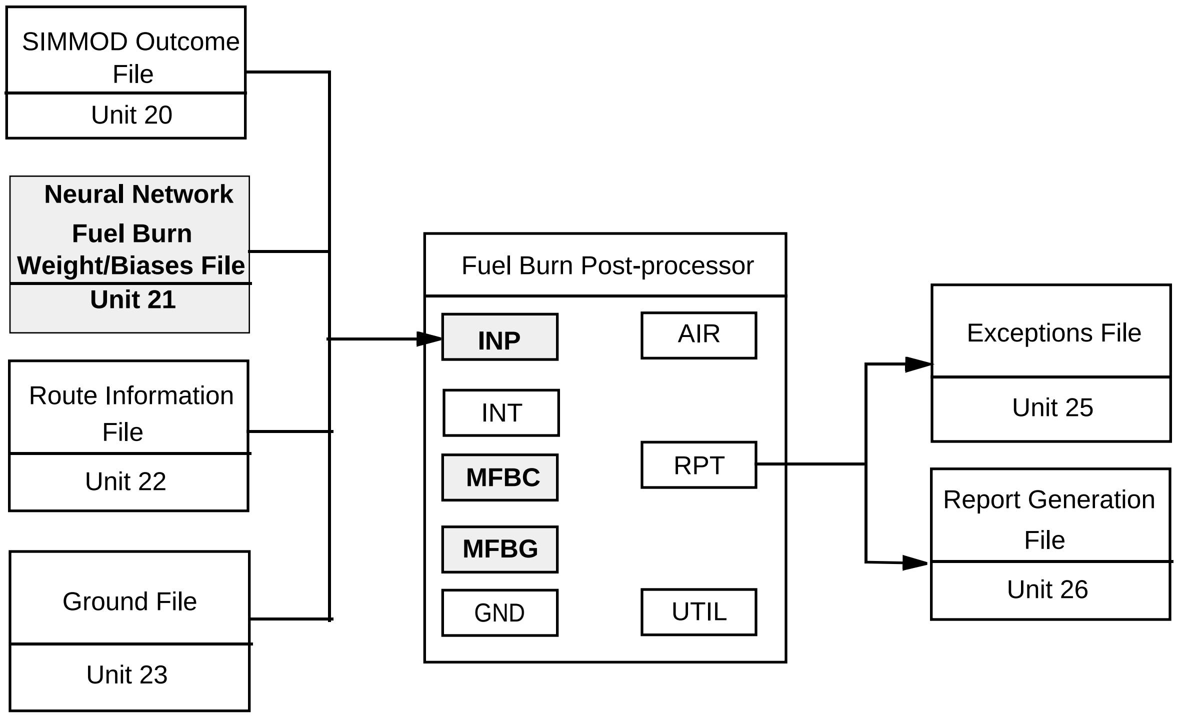


Figure 5.24 SAR Parameter Corrected for Enroute Winds.

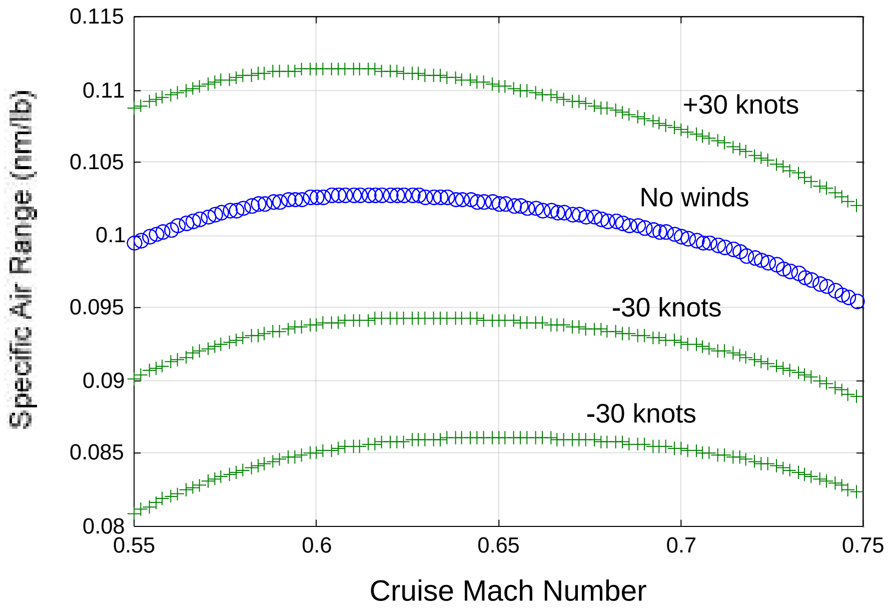


TABLE 5.5. SIMMOD Fuel Burn Postprocessor Routines.

Routine	Purpose	Subroutine Names
AIR	Processes airborne fuel consuming events	AIR.100.AIRSPACE.ACTION ' AIR.110.TRAVERSE.AIR.LINK AIR.120.SPEED.UP.AIR.LINK AIR.130.PATH.STRETCH.AIR.LINK AIR.140.SLOW.DOWN.AIR.LINK AIR.150.HOLD.AT.AIR.NODE AIR.160.NON.SIMOD.NODE AIR.170.RESET.AIR.ACTION
FBC	Processes fuel burn calculations	FBC.100.AIR.FUEL.BURN ** FBC.200.LEG.DISTANCE FBC.900.INTERMEDIATE.VAUES ** FBC.910.DENSITY.ALTTUDE FBC.920.SPEED.OF.SOUND FBC.930.FUEL.FLOW.LIM ** FBC.940.THRUST ** FBC.950.F.CONSTANTS**
GND	Processes ground fuel consuming events	GND.100.GROUND.ACTION GND.110.TAXI.GROUND.LINK GND.120.HOLD.AT.GROUND.NODE GND.130.LANDING.ROLL GND.140.TAKEOFF.ROLL
FBG	Processes ground fuel burn computations	FBG.100.GROUND.FUEL.BURN ** FBG.200.GROUND.THRUST **
INP	Reads an input file and initializes variables	INP.100.READ.FUELBURN.PARAMETERS ** INP.200.READ.ROUTE.INFORMATION INP.210.STD.TEMPERATURE
INT	Initiates an action such as a flight or a cross-over between air and ground or vice versa	INT.100.INITIATE.FLIGHT INT.200.GROUND.THEN.AIR INT.300.AIR.THEN.GROUND
RPT	Reports the results of a fuel burn calculation	RPT.100.TERMINATE.FLIGHT RPT.110.REPORT.AIRSPACE.ACTION RPT.120.REPORT.GROUND.ACTION RPT.200.REPORT.STATISTICS
UTL	Utility routines tracking statistical accumulators needed in the fuel report	UTL.100.GET.AIRLINE.INDEX UTL.110.ADD.AIRLINE UTL.200.GET.ROUTE.INDEX UTL.210.ADD.ROUTE UTL.300.UPDATE.CUMULATIVES

** Routines to be modified to accommodate a neural network model

CHAPTER 6 Conclusions and Recommendations

6.1 Conclusions

The existing SIMMOD fuel consumption model based on aircraft performance parameters was studied. Advantages and disadvantages of this model were reviewed. A representative neural network aided fuel consumption model was developed using data given in the aircraft performance manual. The neural network was trained to estimate fuel consumption of an example aircraft. Results were compared to the actual performance provided in the aircraft performance manual and found to be accurate for possible implementation in SIMMOD and other fast-time simulation programs.

The following conclusions are derived from our analysis:

1. The advantage of the existing advanced fuel consumption model (i.e. those not using neural networks) is that it can be easily transferred to any flight trajectory program, therefore, implementation of this model is simple. The disadvantage of this model is that the information required to create the data base for this particular algorithm is very difficult to obtain. This fact has been without doubt a constraint in the expansion of the fuel burn database in SIMMOD.

2. The information provided in the aircraft performance manual is a reliable source to obtain fuel consumption data of any aircraft. Along with neural network technology, a neural network aided fuel consumption model has been developed.

3. Results obtained from the neural network aided fuel consumption model show that a neural network with proper training is an accurate and efficient mean to calculate fuel consumption of fixed wing aircraft. The added benefit of this approach is that only the

flight performance manual of the aircraft is needed to characterize the complete fuel burn behavior of the vehicle throughout its flight envelope.

4. A neural network is found to be a viable alternative in fuel consumption estimating application. The computational results obtained in this paper indicate that the neural network approach can be implemented in fast-time simulation models such as SIMMOD, RAMS, TAAM and future products where flight trajectories are described in terms of waypoints. Moreover, neural networks can approximate with good accuracy the complete performance of the vehicle (including climb, cruise, maneuvering, and decent) and simplify the implementation of realistic aircraft models without compromising aircraft sensititive data that is seldom made public.

6.2 Remarks and Recommendations

6.2.1 Remarks

One of the advantages of using neural networks to estimate fuel consumption is that neural networks are able to automatically create an internal distributed model of the problem during training. The problem is, however, that this distributed storage of information makes it almost impossible to explain the network response to input patterns. Here, rule-based systems, such as Expert Systems or Fuzzy Logic, offer a better choice. However, neural networks have already been developed that combine both training from examples and definition of knowledge in the form of rules. The trick is to restrict the interconnectivity of the neural network so that its structure can be interpreted as an implementation of a rules set. An example of such neural networks are Neuron-Fuzzy systems. The problem of finding the optimal amount of neurons for most neural network types can only be solved by a time consuming trail and error approach. Nevertheless, in our study we found that once a network topology is identified yielding accurate results the same network topology can be used to model other aircraft.

In general, the amount of neurons should be large enough to store all relevant information within the weights and biases, but at the same time small enough to force the neural network to generalize and not to learn the inherent noise present in the training patterns. There are neural networks that automatically insert new neurons for patterns that are not similar to any of the learned ones. Additionally, there are methods like Genetic Algorithms to automatically optimize neural networks. Many neural network types tend to forget what they've previously learned when only new patterns are presented during training. The only way to prevent this is to store all the patterns, to add new patterns, and then to present the whole set during training. Therefore, the artificial intelligence algorithms mentioned above should also be considered as an alternative to estimate fuel consumption of an aircraft.

Fuel Efficient Flight Path

Fuel efficient trajectory is one of the interesting by-products of the fuel consumption estimation model. For each feasible trajectory of an aircraft there will be a corresponding fuel consumption profile. By comparing different trajectories, it is possible to determine the most fuel efficient trajectory. One way to approach this task is using dynamic programming techniques. The disadvantage of this technique is that the computational procedure is very time consuming and repetitive. Although a fuel efficient trajectory may not be executable from air-traffic controllers' point of view, it is beneficial to determine this particular trajectory. Without any doubt, under future free flight conditions, fuel consumption estimation and flight profile generation will be analyzed interactively in advanced air traffic management systems.

Neural Network Model Extensions

The model developed in this research project purely addressed the fuel burn and performance computations typical of fast time simulation models. A future enhancement to the model presented here is the extension to estimate thrust associated with a fuel burn flight condition. In simple terms thrust and fuel burn are related by a characteristic parameter called Thrust Specific Fuel Consumption (TSFC). This paramater is usually a complex function of mach number, temperature, pressure altitude, among other factors. Preliminary results obtained in our research indicate that thrust and TSFC can also be easily characterized using neural networks (we used a Pratt and Whitney JTD9-7R engine for this purpose) and thus thrust values can be be obtained from operational simulation models to support noise studies.

6.2.2 Recommendations

Due to the time constraints of this project, the neural network approach was used to fully describe the fuel consumption metrics of a single aircraft. Although not shown here, the Saab 2000 turboprop aircraft was also modeled using the same network topology and the results were as accurate as those of the Fokker 100. This is a first order demonstration that neural networks can approximate the performance characteristics for various engine-airframe technologies. The algorithms developed in this research project have merit because they simplify matters to add fuel burn computations to any fast-time simulation program where aircraft trajectories are approximated using waypoint structures.

The evolution of future airport and airspace models is likely to implement fuel consumption models as an integral part of the analysis and not as a post-processor module as currently done in practice. SIMMOD 2000 should implement more complete fuel burn procedures that those found in SIMMOD today. In an environment where scarce economic resources are important is perhaps inadmissible to forget the costs associated with aircraft operations in the National Airspace System (NAS).

Recommendations for future research are:

a) Test the implementation of neural networks to predict fuel consumption for general aviation aircraft. This should be done to ensure that out network topology is robust and applicable to piston engine aircraft.

b) Connect the model developed within the current structure of SIMMOD using standard SIMSCRIPT II.5/C routines. This step should be a formality since we have tested the algorithms in C and MATLAB for full aircraft trajectories. In fact, the SIMMOD outcome file provides a lot more data points than usually required for minimum precision of our program and thus no anticipated surprises should be expected.

c) Validate the model for a large aircraft database. This is a critical step if fuel burn is ever to be used by airspace and airport planners in a reliable fashion. Ironically, SIMMOD was developed as a fuel consumption prediction tool. Yet few people today employ this model for this purpose because the fuel consumption data base is very small compared to the number of aircraft modeled operationally (only 17 aircraft are actually represented in terms of fuel consumption parameters). This trend should be reversed because a model that predicts some of the economic aspects of airport and airspace operations would have more appeal to a wider range of users. Besides, expenditure of fuel resources might become increasingly important in future years as communities around the world are more in tune with the preservation of natural resources and fuel becomes a more expensive commodity.

APPENDIX A Neural Network Templates Source Code

This appendix contains computer algorithms and source code to train and generalize aircraft fuel consumption neural networks. The following sections are included:

A.1-3 Neural Network Training Templates

1. Take-off and climb out fuel estimation
2. Climb performance estimation
3. Cruise specific air range
4. Descent performance estimation

A.4-5 Neural Network Testing Program

1. Testing main program
2. Statistical analysis

A.6 Main Program to Calculate Fuel Consumption

1. Main Program
2. Climb subroutine
3. Cruise subroutine
4. Descent subroutine

A.1 Take-off and Climb Out Fuel Estimation

```
%NEURAL NETWORKS TRAINING FOR TAKEOFF AND CLIMBOUT
%DEVELOPED BY FRANK CHEUNG
%UNDERSPERVISION OF DR. ANTONIO TRANI
%LAST MODIFIED 23/11/97 by Toni Trani
fid = fopen ('cof')
cof = fscanf(fid, '%g %g %g ', [3,inf]);
cof=cof';

fclose(fid)

for i = 1 : 8;
Weightco(i)=cof(i,1);
Fuelco(i)=cof(i,2);
ISAco(i)=cof(i,3);

end

% Data Normalization

W_co = Weightco/max(Weightco);
F_co = Fuelco/max(Fuelco);
ISA_co = ISAco/max(ISAco);


%Set Inputs and Targets

W_co_min = min(W_co);
W_co_max = max(W_co);
ISA_co_min = min(ISA_co);
ISA_co_max = max(ISA_co);
F_co_max = max(F_co);
F_co_min = min(F_co);

P1_co = [W_co_min W_co_max; ISA_co_min ISA_co_max];
T1_co = [F_co_min F_co_max ] ;
P_co = [W_co; ISA_co];
Ta_co = [F_co ];
% Initialize Traning Parameters

df = 10000;   % Frequency of progress displays (in epochs).
me = 10000; % Maximum number of epochs to train.
eg = 0.02; % Sum-squared error goal.
tp = [df me eg ];

%*******************************
%       For Fuel Burn         **
%*******************************

% Initialize Weights and Biasis
```

```
nns = 8; % Number of Neurons in each layer
nns2 = 8;

[ W11_co,b11_co,W12_co,b12_co,W13_co,b13_co ]=initff(P1_co,nns,'logsig',nns2,'tansig',T1_co,'purelin');

% Taining of the neural networks using Lavenberg-Marquardt Alogrithm

[ W11_co,b11_co,W12_co,b12_co,W13_co,b13_co ]= trainlm(W11_co,b11_co,'logsig',W12_co,b12_co,'tan-
sig',W13_co,b13_co,'purelin',P_co,Ta_co,tp);

% Export result

fid=fopen('wbtx.txt','w');

fprintf(fid,'%6.3f %6.3f %6.3f %6.3f %6.3f %6.3f\n',W11_co,b11_co,W12_co,b12_co,W13_co,b13_co);

% Simulate Traning Results

[F1] = simuff (P,W11_co,b11_co,'logsig',W12_co,b12_co,'tansig',W13_co,b13_co,'purelin');

end
```

A.2 Climb Performance Estimation

```
_______________________________________________________________________
% NEURAL NETWORKS TRAINING FOR CLIMB PHASE
%DEVELOPED BY FRANK CHEUNG
%UNDERSPERVISION OF DR. ANTONIO TRANI
%LAST MODIFIED 06/07/97

% Data input

fid = fopen ('CDISTFINALA')
climbd = fscanf(fid, '%g %g %g %g %g', [5,inf]);
climbd=climbd';

for i = 1 : 864;

Mach_cbd(i)=climbd(i,1);
Weight_cbd(i)=climbd(i,2);
Dist_cbd(i)=climbd(i,3);
Alt_cbd(i)=climbd(i,4);
ISA_cbd(i)=climbd(i,5);

  end
fid = fopen ('CF_FINALWA')
climbf = fscanf(fid, '%g %g %g %g %g', [5,inf]);
climbf=climbf';
for i = 1 : 864;

Mach_cbf(i)=climbf(i,1);
Weight_cbf(i)=climbf(i,2);
```

```
Fuel_cbf(i)=climbf(i,3);
Alt_cbf(i)=climbf(i,4);
ISA_cbf(i)=climbf(i,5);

  end
% Data Normalization

W_cbd = Weight_cbd/max(Weight_cbd);
M_cbd = Mach_cbd/max(Mach_cbd);
ISA_cbd = ISA_cbd/max(ISA_cbd);
A_cbd = Alt_cbd/max(Alt_cbd);
D_cbd = Dist_cbd/max(Dist_cbd);

W_cbf = Weight_cbf/max(Weight_cbf);
M_cbf = Mach_cbf/max(Mach_cbf);
ISA_cbf = ISA_cbf/max(ISA_cbf);
A_cbf = Alt_cbf/max(Alt_cbf);
F_cbf = Fuel_cbf/max(Fuel_cbf);

%Set Inputs and Targets

W_cbd_min = min(W_cbd);
W_cbd_max = max(W_cbd);
M_cbd_min = min(M_cbd);
M_cbd_max = max(M_cbd);
ISA_cbd_min = min(ISA_cbd);
ISA_cbd_max = max(ISA_cbd);
A_cbd_min = min(A_cbd);
A_cbd_max = max(A_cbd);
D_cbd_min = min(D_cbd);
D_cbd_max = max(D_cbd);

W_cbf_min = min(W_cbf);
W_cbf_max = max(W_cbf);
M_cbf_min = min(M_cbf);
M_cbf_max = max(M_cbf);
ISA_cbf_min = min(ISA_cbf);
ISA_cbf_max = max(ISA_cbf);
A_cbf_min = min(A_cbf);
A_cbf_max = max(A_cbf);
F_cbf_max = max(F_cbf);
F_cbf_min = min(F_cbf);

P1_cbd = [W_cbd_min W_cbd_max; M_cbd_min M_cbd_max; ISA_cbd_min ISA_cbd_max ...
; A_cbd_min A_cbd_max];
P1_cbf = [W_cbf_min W_cbf_max; M_cbf_min M_cbf_max; ISA_cbf_min ISA_cbf_max ...
; A_cbf_min A_cbf_max];
T1_cbf = [F_cbf_min F_cbf_max ] ;
T1_cbd = [D_cbd_min D_cbd_max ] ;

P_cbd = [W_cbd; M_cbd; ISA_cbd; A_cbd ];
P_cbf = [W_cbf; M_cbf; ISA_cbf; A_cbf ];

Ta_cbf = [F_cbf ];
```

```
Ta_cbd = [D_cbd];

% Initialize Traning Parameters

df = 100;   % Frequency of progress displays (in epochs).
me = 10000; % Maximum number of epochs to train.
eg = 0.02; % Sum-squared error goal.
tp = [df me eg ];


% Initialize Weights and Biasis

nns = 8; % Number of Neurons in each layer
nns2 = 8;


%********************************
%       For Climb Distance    **
%********************************


[ W31_cb_d,b31_cb_d,W32_cb_d,b32_cb_d,W33_cb_d,b33_cb_d ]=initff(P1_cbd,nns,'logsig',nns2 ...
,'tansig',T1_cbd,'purelin');

% Taining of the neural networks using Lavenberg-Marquardt Alogrithm

[ W31_cb_d,b31_cb_d,W32_cb_d,b32_cb_d,W33_cb_d,b33_cb_d ]= trainlm(W31_cb_d,b31_cb_d,'logsig' ...
,W32_cb_d,b32_cb_d,'tansig',W33_cb_d,b33_cb_d,'purelin',P_cbd,Ta_cbd,tp);


%********************************
%       For Climb Fuel        **
%********************************


[ W31_cb_f,b31_cb_f,W32_cb_f,b32_cb_f,W33_cb_f,b33_cb_f ]=initff(P1_cbf,nns,'logsig',nns2,'tansig' ...
,T1_cbf,'purelin');

% Taining of the neural networks using Lavenberg-Marquardt Alogrithm

[ W31_cb_f,b31_cb_f,W32_cb_f,b32_cb_f,W33_cb_f,b33_cb_f ]= trainlm(W31_cb_f,b31_cb_f,'logsig',W32_cb_f ...
,b32_cb_f,'tansig',W33_cb_f,b33_cb_f,'purelin',P_cbf,Ta_cbf,tp);

end
```

A.3 Cruise Specific Air Range

```
% NEURAL NETWORKS TRAINING FOR CRUISE PHASE
%DEVELOPED BY FRANK CHEUNG
%UNDERSPERVISION OF DR. ANTONIO TRANI
%LAST MODIFIED 11/07/97

% Data input

fid = fopen ('CRVFINE')
cruise = fscanf(fid, '%g %g %g %g ', [4,inf]);
cruise=cruise';

for i = 1 : 805;
Alt_cr(i)=cruise(i,1);
Weight_cr(i)=cruise(i,2);
Mach_cr(i)=cruise(i,3);
Fuel_cr(i)=cruise(i,4);

  end

% Data Normalization

W_cr = Weight_cr/max(Weight_cr);
M_cr = Mach_cr/max(Mach_cr);
A_cr = Alt_cr/max(Alt_cr);
F_cr = Fuel_cr/max(Fuel_cr);


%Set Inputs and Targets

W_cr_min = min(W_cr);
W_cr_max = max(W_cr);
M_cr_min = min(M_cr);
M_cr_max = max(M_cr);
A_cr_min = min(A_cr);
A_cr_max = max(A_cr);
F_cr_max = max(F_cr);
F_cr_min = min(F_cr);
P1_cr = [W_cr_min W_cr_max; M_cr_min M_cr_max;  A_cr_min A_cr_max];
T1_cr = [F_cr_min F_cr_max ] ;

P_cr = [W_cr; M_cr; A_cr ];
Ta_cr = [F_cr ];
% Initialize Traning Parameters

df = 10;   % Frequency of progress displays (in epochs).
me = 10000; % Maximum number of epochs to train.
eg = 0.02; % Sum-squared error goal.
tp = [df me eg ];


% Initialize Weights and Biasis
```

```
nns = 10; % Number of Neurons in each layer
nns2 = 10;


%*******************************
%       For Cruise Fuel       **
%*******************************


[ W31_cr,b31_cr,W32_cr,b32_cr,W33_cr,b33_cr ]=initff(P1_cr,nns,'logsig',nns2,'tansig',T1_cr,'purelin');

% Taining of the neural networks using Lavenberg-Marquardt Alogrithm

[ W31_cr,b31_cr,W32_cr,b32_cr,W33_cr,b33_cr ]= trainlm(W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tan-
sig',W33_cr,b33_cr,'purelin',P_cr,Ta_cr,tp);

% Export Result

fid=fopen('wbcr.txt','w');

fprintf(fid,'%6.3f %6.3f %6.3f %6.3f %6.3f %6.3f,%6.3f,%6.3f\n',W31_cr,b31_cr,W32_cr,b32_cr,W33_cr,b33_cr);

% Simulate Traning Results
end
4. Descent performance estimation

% NEURAL NETWORKS TRAINING FOR DESCENT PHASE
%DEVELOPED BY FRANK CHEUNG
%UNDERSPERVISION OF DR. ANTONIO TRANI
%LAST MODIFIED 24/06/97

% Data input

fid = fopen ('dd')
dd = fscanf(fid, '%g %g %g %g %g ', [5,inf]);
dd=dd';

for i = 1 : 288;
Mach_dd(i)=dd(i,1);
Weight_dd(i)=dd(i,2);
Dist_dd(i)=dd(i,3);
Alt_dd(i)=dd(i,4);
ISA_dd(i)=dd(i,5);

  end

% Data Normalization

W_dd = Weight_dd/max(Weight_dd);
M_dd = Mach_dd/max(Mach_dd);
A_dd = Alt_dd/max(Alt_dd);
D_dd = Dist_dd/max(Dist_dd);
ISA_dd = ISA_dd/max(ISA_dd);
```

```
%Set Inputs and Targets

W_dd_min = min(W_dd);
W_dd_max = max(W_dd);
M_dd_min = min(M_dd);
M_dd_max = max(M_dd);
A_dd_min = min(A_dd);
A_dd_max = max(A_dd);
ISA_dd_min = min(ISA_dd);
ISA_dd_max = max(ISA_dd);
Dist_dd_min = min(D_dd);
Dist_dd_max = max(D_dd);
P1_dd = [W_dd_min W_dd_max; M_dd_min M_dd_max;  A_dd_min A_dd_max;ISA_dd_min ISA_dd_max];
T1_dd = [Dist_dd_min Dist_dd_max ] ;

P_dd = [W_dd; M_dd; A_dd;ISA_dd ];
Ta_dd = [D_dd];

% Initialize Traning Parameters

df = 100;   % Frequency of progress displays (in epochs).
me = 10000; % Maximum number of epochs to train.
eg = 0.02; % Sum-squared error goal.
tp = [df me eg ];


% Initialize Weights and Biasis

nns = 8; % Number of Neurons in each layer
nns2 = 8;


%********************************
%       For Descent Distance  **
%********************************


[ W31_dd,b31_dd,W32_dd,b32_dd,W33_dd,b33_dd ]=initff(P1_dd,nns,'logsig',nns2,'tansig',T1_dd,'purelin');

% Taining of the neural networks using Lavenberg-Marquardt Alogrithm

[       W31_dd,b31_dd,W32_dd,b32_dd,W33_dd,b33_dd        ]=        trainlm(W31_dd,b31_dd,'logsig',W32_dd,b32_dd,'tan-
sig',W33_dd,b33_dd,'purelin',P_dd,Ta_dd,tp);

% Export Result

fid=fopen('wbdd.txt','w');

fprintf(fid,'%6.3f %6.3f %6.3f %6.3f %6.3f %6.3f\n',W31_dd,b31_dd,W32_dd,b32_dd,W33_dd,b33_dd);
```

```
fid = fopen ('df')
df = fscanf(fid, '%g %g %g %g %g ', [5,inf]);
df=df';

for i = 1 : 270;
Weight_df(i)=df(i,1);
Mach_df(i)=df(i,2);
fuel_df(i)=df(i,3);
Alt_df(i)=df(i,4);
ISA_df(i)=df(i,5);

  end

% Data Normalization

W_df = Weight_df/max(Weight_df);
M_df = Mach_df/max(Mach_df);
A_df = Alt_df/max(Alt_df);
F_df = fuel_df/max(fuel_df);
ISA_df = ISA_df/max(ISA_df);

%Set Inputs and Targets

W_df_min = min(W_df);
W_df_max = max(W_df);
M_df_min = min(M_df);
M_df_max = max(M_df);
A_df_min = min(A_df);
A_df_max = max(A_df);
ISA_df_min = min(ISA_df);
ISA_df_max = max(ISA_df);
F_df_min = min(F_df);
F_df_max = max(F_df);
P1_df = [W_df_min W_df_max; M_df_min M_df_max;  A_df_min A_df_max;ISA_df_min ISA_df_max];
T1_df = [F_df_min F_df_max ] ;

P_df = [W_df; M_df; A_df;ISA_df ];
Ta_df = [F_df];

% Initialize Traning Parameters

df = 100;   % Frequency of progress displays (in epochs).
me = 10000; % Maximum number of epochs to train.
eg = 0.02; % Sum-squared error goal.
tp = [df me eg ];


% Initialize Weights and Biasis

nns = 8; % Number of Neurons in each layer
nns2 = 8;


%******************************
```

```
%       For Descent Fuel      **
%*******************************


[ W31_df,b31_df,W32_df,b32_df,W33_df,b33_df ]=initff(P1_df,nns,'logsig',nns2,'tansig',T1_df,'purelin');

% Taining of the neural networks using Lavenberg-Marquardt Alogrithm

[        W31_df,b31_df,W32_df,b32_df,W33_df,b33_df          ]=          trainlm(W31_df,b31_df,'logsig',W32_df,b32_df,'tan-
sig',W33_df,b33_df,'purelin',P_df,Ta_df,tp);

% Export Result

fid=fopen('wbdf.txt','w');

fprintf(fid,'%6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f\n',W31_df,b31_df,W32_df,b32_df,W33_df,b33_df);

fid = fopen ('dt')
dt = fscanf(fid, '%g %g %g %g %g ', [5,inf]);
dt=dt';

for i = 1 : 258;
Weight_dt(i)=dt(i,1);
Mach_dt(i)=dt(i,2);
time_dt(i)=dt(i,3);
Alt_dt(i)=dt(i,4);
ISA_dt(i)=dt(i,5);

  end

% Data Normalization

W_dt = Weight_dt/max(Weight_dt);
M_dt = Mach_dt/max(Mach_dt);
A_dt = Alt_dt/max(Alt_dt);
T_dt = time_dt/max(time_dt);
ISA_dt = ISA_dt/max(ISA_dt);

%Set Inputs and Targets

W_dt_min = min(W_dt);
W_dt_max = max(W_dt);
M_dt_min = min(M_dt);
M_dt_max = max(M_dt);
A_dt_min = min(A_dt);
A_dt_max = max(A_dt);
ISA_dt_min = min(ISA_dt);
ISA_dt_max = max(ISA_dt);
T_dt_min = min(T_dt);
T_dt_max = max(T_dt);
P1_dt = [W_dt_min W_dt_max; M_dt_min M_dt_max;  A_dt_min A_dt_max;ISA_dt_min ISA_dt_max];
T1_dt = [T_dt_min T_dt_max ] ;

P_dt = [W_dt; M_dt; A_dt;ISA_dt ];
```

```
Ta_dt = [T_dt];

% Initialize Traning Parameters

df = 100;   % Frequency of progress displays (in epochs).
me = 10000; % Maximum number of epochs to train.
eg = 0.02; % Sum-squared error goal.
tp = [df me eg ];


% Initialize Weights and Biasis

nns = 8; % Number of Neurons in each layer
nns2 = 8;


%********************************
%       For Descent Time      **
%********************************


[ W31_dt,b31_dt,W32_dt,b32_dt,W33_dt,b33_dt ]=initff(P1_dt,nns,'logsig',nns2,'tansig',T1_dt,'purelin');

% Taining of the neural networks using Lavenberg-Marquardt Alogrithm

[ W31_dt,b31_dt,W32_dt,b32_dt,W33_dt,b33_dt ]= trainlm(W31_dt,b31_dt,'logsig',W32_dt,b32_dt,'tan-
sig',W33_dt,b33_dt,'purelin',P_dt,Ta_dt,tp);
% Export Result

fid=fopen('wbdt.txt','w');
fprintf(fid,'%6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f %6.3f\n',W31_dt,b31_dt,W32_dt,b32_dt,W33_dt,b33_dt);
end
```

A.4 Neural Network Testing Program

A.4.1 Testing main program

```
%NEURAL NETWORKS TRAINING FOR DATA TESTING
%DEVELOPED BY FRANK CHEUNG
%UNDERSPERVISION OF DR. ANTONIO TRANI
%LAST MODIFIED 11/07/97


load Climb;
load Cruise3_8;
load descent;

global W31_cb_f b31_cb_f W32_cb_f b32_cb_f W33_cb_f ...
 b33_cb_f W31_cb_d b31_cb_d W32_cb_d b32_cb_d W33_cb_d ...
 b33_cb_d W31_cr b31_cr W32_cr b32_cr W33_cr b33_cr ...
 W31_df b31_df W32_df b32_df W33_df b33_df W31_d
global b31_dd W32_dd b32_dd W33_dd b33_dd Weight_cbd Mach_cbd ISA_cbd Dist_cbd ...
```

```
 Alt_cbd  Weight_cbf Mach_cbf ISA_cbf Alt_cbf Fuel_cbf ...
 Alt_cr Weight_cr Mach_cr Fuel_cr Weight_dd ...
 Mach_dd  Alt_dd  ISA_dd  Weight_df Mach_df fuel_df Alt_df ISA_df;



fid = fopen ('CDTFINAL');
CBD = fscanf(fid, '%g %g %g %g %g', [5,inf]);
CBD=CBD';
for i=1:854;
CBDM(i)=CBD(i,1);
CBDW(i)=CBD(i,2);
TCBD(i)=CBD(i,3);
CBDA(i)=CBD(i,4);
 CBDI(i)=CBD(i,5);

end


fid = fopen ('CFTFINAL');
CBF = fscanf(fid, '%g %g %g %g %g', [5,inf]);
CBF=CBF';
for i=1:852;
CBFM(i)=CBF(i,1);
CBFW(i)=CBF(i,2);
TCBF(i)=CBF(i,3);
CBFA(i)=CBF(i,4);
CBFI(i)=CBF(i,5);

end

fid = fopen ('CRT');
CBT = fscanf(fid, '%g %g %g %g ', [4,inf]);
CBT=CBT';
for i=1:805;
CTA(i)=CBT(i,1);
CTW(i)=CBT(i,2);
CTM(i)=CBT(i,3);
CTF(i)=CBT(i,4);

end

fid = fopen ('DDT');
DD = fscanf(fid, '%g %g %g %g %g', [5,inf]);
DD=DD';
for i=1:140;
DDTM(i)=DD(i,1);
DDTW(i)=DD(i,2);
DDTD(i)=DD(i,3);
DDTA(i)=DD(i,4);
DDTI(i)=DD(i,5);

end
```

```
fid = fopen ('DFT');
DF = fscanf(fid, '%g %g %g %g %g', [5,inf]);
DF=DF';
for i=1:140;
DFTM(i)=DF(i,1);
DFTW(i)=DF(i,2);
DFTF(i)=DF(i,3);
DFTA(i)=DF(i,4);
DFTI(i)=DF(i,5);


end


WMX_cbd = max(Weight_cbd);
MMX_cbd = max(Mach_cbd);
IMX_cbd = max(ISA_cbd);
AMX_cbd = max(Alt_cbd);
WMX_cbf = max(Weight_cbf);
MMX_cbf = max(Mach_cbf);
IMX_cbf = max(ISA_cbf);
AMX_cbf = max(Alt_cbf);
MCBF = max(Fuel_cbf);
MCBD = max(Dist_cbd);
MCRA = max(Alt_cr);
MCRW = max(Weight_cr);
MCRM = max(Mach_cr);
MFCR = max(Fuel_cr);
MWDD = max(Weight_dd);
MMDD = max(Mach_dd);
MDDD = max(Dist_dd);
MADD = max(Alt_dd);
MWDF = max(Weight_df);
MMDF = max(Mach_df);
MFDF = max(fuel_df);
MADF = max(Alt_df);


%For Climb

% Mach Number Normalization

M1 = CBFM./MMX_cbf;

M2 = CBDM./MMX_cbd;


% Altitude Normalization

A1=CBFA./AMX_cbf;

A2= CBDA./AMX_cbd;
```

```
T1N = CBFI./10;

T2N = CBDI./10;

% Weight Normalization

WN1 = CBFW./WMX_cbf;

WN2 = CBDW./WMX_cbd;

P1C = [WN1; M1; T1N; A1 ];

P2C = [WN2; M2; T2N; A2 ];



F1=simuff(P1C,W31_cb_f,b31_cb_f,'logsig',W32_cb_f,b32_cb_f,'tansig',W33_cb_f,b33_cb_f,'purelin');


D1=simuff(P2C,W31_cb_d,b31_cb_d,'logsig',W32_cb_d,b32_cb_d,'tansig',W33_cb_d,b33_cb_d,'purelin');

FC_cal = F1.*MCBF;

DC_cal = D1.*MCBD;

for i= 1:852;

if TCBF(i) <= 0.001;
FC_err(i) = 0;
RFC_err(i) = 0;

else;

FC_err(i) = (TCBF(i) - FC_cal(i))/TCBF(i);
RFC_err(i) = (TCBF(i) - FC_cal(i));
end
end
for i = 1:854;

if TCBD(i) <= 0.001;

DC_err(i) = 0;
RDC_err(i) = 0;

else;

DC_err(i) = (TCBD(i) - DC_cal(i))/TCBD(i);
RDC_err(i)= (TCBD(i) - DC_cal(i));
end
end

AVG_DC = sum(abs(DC_err))/852*100;
```

```
AVG_FC = sum(abs(FC_err))/864*100;

% For Cruise

% Mach Number Normalization

TM3 = CTM./MCRM;


% Altitude Normalization

TA3= CTA./MCRA;

% Weight Normalization

WN3 = CTW./MCRW;

P3 = [WN3; TM3; TA3 ];

F3 =simuff(P3,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin');

CRF_cal = F3.*MFCR;

CRF_err = (CTF-CRF_cal)./CTF;
RCRF_err = (CTF-CRF_cal);
AVG_CRF = sum(abs(CRF_err))/805*100;

% Mach Number Normalization

M4 = DFTM./MMDF;

M5 = DDTM./MMDD;


% Altitude Normalization

A4= DFTA./MADF;

A5 = DDTA./MADD;

%ISA Initialization

for i = 1:140;

T4(i) = DFTI(i)/10;

end

for i = 1:140;

T5(i) = DDTI(i)/10;

end
```

```
% Weight Normalization

WND = DDTW./MWDD;

WNF = DFTW./MWDF;

% Initialization

P4 = [WNF; M4; A4; T4 ];

P5 = [WND; M5; A5; T5 ];


F4=simuff(P4,W31_df,b31_df,'logsig',W32_df,b32_df,'tansig',W33_df,b33_df,'purelin');


D5=simuff(P5,W31_dd,b31_dd,'logsig',W32_dd,b32_dd,'tansig',W33_dd,b33_dd,'purelin');


D5_cal = D5.*MDDD;

F4_cal = F4.*MFDF;

DD_err = (DDTD-D5_cal)./DDTD;
RDD_err = (DDTD-D5_cal);
DF_err = (DFTF-F4_cal)./DFTF;
RDF_err = (DFTF-F4_cal);
AVG_DD = sum(abs(DD_err))/140*100;

AVG_DF = sum(abs(DF_err))/140*100;

i=1:850;

%*********PLOTS**********************

%********For Climb******************




plot(i,RFC_err(i), '-',i,TCBF(i),'o',i,FC_cal(i),'x');
xlabel('Testing Point No.');
ylabel('Climb Fuel (lb)');
title(['Average absolute relative error is ', num2str(AVG_FC),'%']);
legend('Actual Error','Actual Fuel Burn','Estimated Fuel Burn',2);

pause

plot(i,RDC_err(i), '-',i,TCBD(i),'o',i,DC_cal(i),'x');
xlabel('Testing point No.');
ylabel('Climb Distance (nm)');
title(['Average absolute relative error is ',num2str(AVG_DC),'%']);
legend('Relative Error','Actual Climb Distance','Estimated Climb Distance',2);
```

```
pause
plot(CBFA,TCBF,'x');
xlabel('Altitude (1000ft)');
ylabel('Climb Fuel (lb)');
legend('Fuel Burn',2);

pause

plot(CBFA,TCBF,'x',CBFA,FC_cal,'o');
xlabel('Altitude (1000ft)');
ylabel('Climb Fuel (lb)');
title(['Average absolute relative error is ',num2str(AVG_FC),'%']);
legend('Actual Climb Fuel','Estimated Climb Fuel',2);
pause

plot(CBDA,TCBD,'x');
xlabel('Altitude (1000ft)');
ylabel('Climb Distance (nm)');
legend('Climb Distance',2);
pause

plot(CBDA,TCBD,'x',CBDA,DC_cal,'o');
xlabel('Altitude (1000ft)');
ylabel('Actual Climb Distance (nm)');
title(['Average absolute relative error is ',num2str(AVG_DC),'%']);
legend('Actual Climb Distance','Estimated Climb Distance',2);
pause

%************For Cruise**************

i=1:805;

plot(Mach_cr,Alt_cr,'x');
xlabel('MACH NUMBER');
ylabel('Altitude (1000ft)');
title('Cruise Envelope of F100');
legend('Performance Point',2)
pause

plot(CTM,CTF,'x');
xlabel('MACH NUMBER');
ylabel('Specific Air Range (nm/lb)');
title(['Average absolute relative error is ',num2str(AVG_CRF),'%']);
legend('Cruise Specific Air-Range',2)
pause

plot(CTM,CTF,'x',CTM,CRF_cal,'o');
xlabel('MACH NUMBER');
ylabel('Specific Air Range (nm/lb)');
title(['Average absolute relative error is ',num2str(AVG_CRF),'%']);
legend('Actual Cruise Specific Range' ...
,'Estimated Cruise Specific Range',2);
pause
```

```
plot(i,RCRF_err(i), '-',i,CTF(i),'o',i,CRF_cal(i),'x');
xlabel('Tesing Point Number');
ylabel('Specifc Air Range (nm/lb)');
title(['Average absolute relative error is ',num2str(AVG_CRF),'%']);
legend('Relative Error','Actual Cruise Specific Range' ...
,'Estimated Cruise Specific Range',2);
pause

%***************For Descent***********************
i=1:140;
plot(i,RDD_err(i), '-',i,DDTD(i),'o',i,D5_cal(i),'x');
xlabel('Points');
ylabel('Descent Distance (nm)');
legend('Relative Error','Actual Descent Distance','Estimated Descent Distance',2);

title(['Average absolute relative error is ', num2str(AVG_DD),'%']);
pause
plot(i,RDF_err(i),'-',i,DFTF(i), 'o',i,F4_cal(i),'x');
xlabel('Points');
ylabel('Descent Fuel (lb)');

legend('Actual Error','Actual Descent Fuel','Estimated Descent Fuel',2);
title(['Average absolute relative error is ', num2str(AVG_DF),'%']);
pause
```

A.5 Statistical analysis

```
____________________________________________________
%NEURAL NETWORKS TRAINING FOR STATISTICAL ANALYSIS
%DEVELOPED BY FRANK CHEUNG
%UNDERSPERVISION OF DR. ANTONIO TRANI
%LAST MODIFIED 11/07/97

load cruise3_8;
load climb;
load descent;
fid = fopen ('CDTFINAL');
CBD = fscanf(fid, '%g %g %g %g %g', [5,inf]);
CBD=CBD';
for i=1:854;
CBDM(i)=CBD(i,1);
CBDW(i)=CBD(i,2);
TCBD(i)=CBD(i,3);
CBDA(i)=CBD(i,4);
 CBDI(i)=CBD(i,5);

end


fid = fopen ('CFTFINAL');
CBF = fscanf(fid, '%g %g %g %g %g', [5,inf]);
```

```
CBF=CBF';
for i=1:852;
CBFM(i)=CBF(i,1);
CBFW(i)=CBF(i,2);
TCBF(i)=CBF(i,3);
CBFA(i)=CBF(i,4);
CBFI(i)=CBF(i,5);

end

% Simulate Traning Results

fid = fopen ('CRT');
CBT = fscanf(fid, '%g %g %g %g ', [4,inf]);
CBT=CBT';
for i=1:805;
CTA(i)=CBT(i,1);
CTW(i)=CBT(i,2);
CTM(i)=CBT(i,3);
CTF(i)=CBT(i,4);

end
fid = fopen ('DDT');
DD = fscanf(fid, '%g %g %g %g %g', [5,inf]);
DD=DD';
for i=1:140;
DDTM(i)=DD(i,1);
DDTW(i)=DD(i,2);
DDTD(i)=DD(i,3);
DDTA(i)=DD(i,4);
DDTI(i)=DD(i,5);

end

fid = fopen ('DFT');
DF = fscanf(fid, '%g %g %g %g %g', [5,inf]);
DF=DF';
for i=1:140;
DFTM(i)=DF(i,1);
DFTW(i)=DF(i,2);
DFTF(i)=DF(i,3);
DFTA(i)=DF(i,4);
DFTI(i)=DF(i,5);


end


WMX_cbd = max(Weight_cbd);
MMX_cbd = max(Mach_cbd);
IMX_cbd = max(ISA_cbd);
AMX_cbd = max(Alt_cbd);
WMX_cbf = max(Weight_cbf);
MMX_cbf = max(Mach_cbf);
```

```
IMX_cbf = max(ISA_cbf);
AMX_cbf = max(Alt_cbf);
MCBF = max(Fuel_cbf);
MCBD = max(Dist_cbd);
MCRA = max(Alt_cr);
MCRW = max(Weight_cr);
MCRM = max(Mach_cr);
MFCR = max(Fuel_cr);
MWDD = max(Weight_dd);
MMDD = max(Mach_dd);
MDDD = max(Dist_dd);
MADD = max(Alt_dd);
MWDF = max(Weight_df);
MMDF = max(Mach_df);
MFDF = max(fuel_df);
MADF = max(Alt_df);
%For Climb

% Mach Number Normalization

M1 = CBFM./MMX_cbf;

M2 = CBDM./MMX_cbd;


% Altitude Normalization

A1=CBFA./AMX_cbf;

A2= CBDA./AMX_cbd;



T1N = CBFI./10;

T2N = CBDI./10;

% Weight Normalization

WN1 = CBFW./WMX_cbf;

WN2 = CBDW./WMX_cbd;

P1C = [WN1; M1; T1N; A1 ];

P2C = [WN2; M2; T2N; A2 ];


F1=simuff(P1C,W31_cb_f,b31_cb_f,'logsig',W32_cb_f,b32_cb_f,'tansig',W33_cb_f,b33_cb_f,'purelin');


D1=simuff(P2C,W31_cb_d,b31_cb_d,'logsig',W32_cb_d,b32_cb_d,'tansig',W33_cb_d,b33_cb_d,'purelin');
```

```
FC_cal = F1.*MCBF;

DC_cal = D1.*MCBD;

for i= 1:800;

if TCBF(i) <= 0.001;
FC_err(i) = 0;
RFC_err(i) = 0;

else;

FC_err(i) = (TCBF(i) - FC_cal(i))/TCBF(i)*100;

end
end
for i = 1:800;

if TCBD(i) <= 0.001;

DC_err(i) = 0;
RDC_err(i) = 0;

else;

DC_err(i) = (TCBD(i) - DC_cal(i))/TCBD(i)*100;

end
end

AVG_DC = sum(DC_err)/852;

AVG_FC = sum(FC_err)/864;

%difference of actual and trained fuel burn
%The sum of the difference is divided by the sample size

% CALCULATING THE MEANS



% W = THE DIFFERENCE OF THE SAMPLE MEANS

% CALCULATING THE RMS (Standard Deviation)

sd = sqrt(sum(DC_err.^2)-(sum(DC_err-AVG_DC)^2/852))/851
sf = sqrt(sum(FC_err.^2)-(sum(FC_err-AVG_FC)^2/864))/863


hist(DC_err,50)
xlabel('Error (%)');
ylabel('Frequency');
title(['Mean Error is ',num2str(AVG_DC),'% and Standard deviation is'...
,num2str(sd), '%']);

grid
legend('Climb Distance Statistics');
pause

hist(FC_err,50)
xlabel('Error (%)');
ylabel('Frequency');
title(['Mean Error is ',num2str(AVG_FC),'% and Standard deviation is'...
,num2str(sf), '%']);

grid
legend('Climb Fuel Statistics');
pause

%******************** CRUISE ****************************
% Mach Number Normalization

TM3 = CTM./max(Mach_cr);


% Altitude Normalization

TA3= CTA./max(Alt_cr);

% Weight Normalization

WN3 = CTW./max(Weight_cr);

P3 = [WN3; TM3; TA3 ];

F3 =simuff(P3,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr ...
,'purelin')*max(Fuel_cr);


%*************************STATISTICS****************************

%difference of actual and trained fuel burn
%The sum of the difference is divided by the sample size
%sw = sum(w)/600 = mean(w)

% CALCULATING THE MEANS
% CTF = GENERALIZED DATA
% F3 = AFBM AFTER INVOKING THE NEURAL NET GENERALIZED DATA
i = 1:805;
w(i) = (CTF(i) - F3(i))./CTF(i)*100;

m = mean(w(i))
for i=1:805;
me(i)=m;
end
% W = THE DIFFERENCE OF THE SAMPLE MEANS
% CALCULATING THE RMS (Standard Deviation)

s1 = sqrt(sum(w.^2)-(sum(w(i)-me(i))^2/805))/804



hist(w,20)
xlabel('Error %');
ylabel('Frequency');

title(['Mean Error is ',num2str(m),'% and Standard deviation is'...
,num2str(s1), '%'])
legend('Cruise Specific Air Range Statistics');
grid
pause

end

% Mach Number Normalization

M4 = DFTM./MMDF;

M5 = DDTM./MMDD;


% Altitude Normalization

A4= DFTA./MADF;

A5 = DDTA./MADD;

%ISA Initialization

for i = 1:140;

T4(i) = DFTI(i)/10;

end

for i = 1:140;

T5(i) = DDTI(i)/10;

end

% Weight Normalization

WND = DDTW./MWDD;

WNF = DFTW./MWDF;

% Initialization

P4 = [WNF; M4; A4; T4 ];
```

```
P5 = [WND; M5; A5; T5 ];


F4=simuff(P4,W31_df,b31_df,'logsig',W32_df,b32_df,'tansig',W33_df,b33_df,'purelin');


D5=simuff(P5,W31_dd,b31_dd,'logsig',W32_dd,b32_dd,'tansig',W33_dd,b33_dd,'purelin');


D5_cal = D5.*MDDD;

F4_cal = F4.*MFDF;

DD_err = (DDTD-D5_cal)./DDTD*100;
RDD_err = (DDTD-D5_cal);
DF_err = (DFTF-F4_cal)./DFTF*100;
RDF_err = (DFTF-F4_cal);
AVG_DD = sum(DD_err)/140;

AVG_DF = sum(DF_err)/140;

sd = sqrt(sum(DC_err.^2)-(sum(DD_err-AVG_DD)^2/140))/139
sf = sqrt(sum(FC_err.^2)-(sum(DF_err-AVG_DF)^2/140))/139



hist(DD_err,15)
xlabel('Error (%)');
ylabel('Frequency');
title(['Mean Error is ',num2str(AVG_DD),'% and Standard deviation is'...
,num2str(sd), '%']);

grid
legend('Descent Distance Statistics');
pause

hist(DF_err,15)
xlabel('Error (%)');
ylabel('Frequency');
title(['Mean Error is ',num2str(AVG_DF),'% and Standard deviation is'...
,num2str(sf), '%']);

grid
legend('Descent Fuel Statistics');
pause
```

A.6 Main Program to Calculate Fuel Consumption

A.6.1 Main Program

```
%FUEL BURN CALCULATION
%DEVELOPED BY FRANK CHEUNG
%UNDER SUPERVISION OF DR. ANTONIO TRANI
%LAST MODIFIED 25/10/97

% Data input

load climb;
load cruise;
load descent;
load co;

global W31_cb_f b31_cb_f W32_cb_f b32_cb_f W33_cb_f ...
 b33_cb_f W31_cb_d b31_cb_d W32_cb_d b32_cb_d W33_cb_d ...
 b33_cb_d W31_cr b31_cr W32_cr b32_cr W33_cr b33_cr;
 global W31_df b31_df W32_df b32_df W33_df b33_df W31_dd ...
 b31_dd W32_dd b32_dd W33_dd b33_dd Weight_cbf Mach_cbf ISA_cbf Dist_cbf ...
 Alt_cb Fuel_cb  Time_cb  Alt_cr Weight_cr Mach_cr Fuel_cr Weight_dd ...
 global Mach_dd  Alt_dd  ISA_dd  Weight_df Mach_df fuel_df Alt_df ISA_df ...
W31_co b31_co W32_co b32_co W33_co ...
 b33_co Weightco Fuelco ISAco Weight_cbd Mach_cbd ISA_cbd Dist_cbd;
% Data initialization

WMX_cbf = max(Weight_cbf);
MMX_cbf = max(Mach_cbf);
TMX_cbf = max(ISA_cbf);
AMX_cbf = max(Alt_cbf);
WMX_cbd = max(Weight_cbd);
MMX_cbd = max(Mach_cbd);
TMX_cbd = max(ISA_cbd);
AMX_cbd = max(Alt_cbd);
MCBF = max(Fuel_cbf);
MCBD = max(Dist_cbd);
MCRA = max(Alt_cr);
MCRW = max(Weight_cr);
MCRM = max(Mach_cr);
MFCR = max(Fuel_cr);
MWDD = max(Weight_dd);
MMDD = max(Mach_dd);
MDDD = max(Dist_dd);
MADD = max(Alt_dd);
MWDF = max(Weight_df);
MMDF = max(Mach_df);
MFDF = max(fuel_df);
MADF = max(Alt_df);

clear A_cb P1_cb A_cb_max P_cb A_cb_min T1_cb Alt_cb T2_cb ...
 T3_cb D_cb  T_cb Dist_cb T_cb_max Dist_cb_max T_cb_min Dist_cb_min Ta_cb ...
 F_cb Tb_cb F_cb_max Tc_cb W_cb_min F_cb_min Tem_cb Weight_cb Fuel_cb  Tem_cb_max M_cb ...
 Tem_cb_min M_cb_max Temp_cb M_cb_min Time_cb Mach_cb;
clear  A_cr  M_cr me ...
 A_cr_max  M_cr_max  nns ...
 A_cr_min  M_cr_min ...
 Alt_cr Mach_cr W_cr  cruise         tp;
clear  F_cr      P1_cr       W_cr_max        df ...
 F_cr_max  P_cr        W_cr_min        eg ...
 F_cr_min  T1_cr       Weight_cr   ...
Fuel_cr    Ta_cr       ans              i ...
A_dd       ISA_dt     T_dt_max         Weight_dt ...
A_dd_max       ISA_dt_max     T_dt_min      ans ;
clear A_dd_min       ISA_dt_min     Ta_dd           ...
A_df           M_dd           Ta_df ...
A_df_max       M_dd_max       Ta_dt ...
A_df_min       M_dd_min ...
A_dt           M_df ...
A_dt_max       M_df_max ...
A_dt_min       M_df_min ...
Alt_dd         M_dt     ...
Alt_df         M_dt_max ...
Alt_dt         M_dt_min;
clear D_dd           Mach_dd  ...
Dist_dd        Mach_df  ...
Dist_dd_max    Mach_dt        W_dd           eg ...
Dist_dd_min    P1_dd          W_dd_max       fid  ...
F_df           P1_df          W_dd_min       fuel_df;
clear F_df_max       P1_dt          W_df           i ...
F_df_min       P_dd           W_df_max       me ...
ISA_dd         P_df           W_df_min       nns ...
ISA_dd_max     P_dt           W_dt           nns2  ...
ISA_dd_min     T1_dd          W_dt_max       time_dt ...
ISA_df         T1_df          W_dt_min       tp ...
ISA_df_max     T1_dt          Weight_dd  ...
ISA_df_min     T_dt           Weight_df ;


fid = fopen ('finaldata.txt')
path = fscanf(fid, '%g %g %g ', [3,inf]);
path=path';


% Counter N

N=0;

%ISA Condition

ISA=0;

% ISA Normalized

ISAN=ISA/10;
```

```
% Number of waypoints included from the beginning of climb segment to Cruise Segment

X = 10;

% Number of waypoints included from the beginning of cruise segment to descent Segment

Y = 20;

% Number of waypoints included from the beginning of descent to the end

Z = 10;

%Initial take off weight (1000 lb)

W(1) = 95;
A(1)= 0;

% Taxi Time

TT = 5; %Taxi Time (min)

% ********************Taxi Fuel Burn(lb)**************************


F(2) = (W(1)*(2/13)+24.2)*TT;

W(2) = W(1)-F(2)/1000;

A(2) = 0

% *************Take off and Climb out to 1500ft************************



%Take off and climbout fuel Calculation

W_in= W(2);

% Weight Normalization

W_in_N= W_in/max(Weightco);

ISA_in = ISAN;

% Input for the Neural Nets

P= [W_in_N;ISAN];

%Output = Fuel Burn

F(3)= simuff (P,W11_co,b11_co,'logsig',W12_co,b12_co,'tansig',W13_co,b13_co,'purelin');

% Weight after Take off and Climbout to 1500 ft;
```

```
W(3)=W(2)-(F(3)*max(Fuelco))/1000;

% ***********************Climbing, Cruise and descent**************************

Alt(3)= 1.500; % Starting Altitude
Mach(3)=0.65; % Starting Mach Number
WC = W(3);    % Starting Weight
D(3)=0;      % Starting distance

for i = 1:X;



D(i+3)=path(i,1); % nm away from origin
Alt(i+3)=path(i,2);  % 1000ft
Mach(i+3)=path(i,3); % Mach Number



% Weight Normalization

Dist = D(i+3)-D(i+2)

W_in = WC;

M1 = Mach(i+2);

M2 = Mach(i+3);

A1 = Alt(i+2);

A2 = Alt(i+3);

TrueWeight = W(i+2);

%Calculated Fuel Burn

[EX,NW,F,FEXF,D_cb]= cal_cb(Dist,A1, M1, A2, M2, W_in,WMX_cbd,MMX_cbd,AMX_cbd ...
,WMX_cbf,MMX_cbf,AMX_cbf,MCBF ...
,MCRA,MCRW,MCRM,MFCR,MCBD,TrueWeight,ISAN);

%Data Registration

EXDIST(i+3) = EX;
W(i+3) = NW;
TrueWeight = W(i+3);
FB(i+3) = F;
FEXFN(i+3) = FEXF;



end

% ****************************End of Climb*****************************************
```

```
for i = X+1:X+Y;


D(i+3)=path(i,1); % nm away from origin
Alt(i+3)=path(i,2);  % 1000ft
Mach(i+3)=path(i,3); % Mach Number


% Weight Normalization

Dist = D(i+3)-D(i+2);

W_in = W(i+2);

M1 = Mach(i+2);

M2 = Mach(i+3);

A1 = Alt(i+2);

A2 = Alt(i+3);



%Calculated Fuel Burn

[NW,F] = cal_cr(Dist,A1, M1, A2, M2, W_in,MCRA,MCRW,MCRM,MFCR);

%Data Registration

EXDIST(i+3) = 0;
W(i+3) = NW;
FB(i+3) = F;
FEXFN(i+3) = 0;
    W_in=NW;
end

%*************************End of Cruise*********************************
for i = X+Y+1:X+Y+Z;


D(i+3)=path(i,1); % nm away from origin
Alt(i+3)=path(i,2);  % 1000ft
Mach(i+3)=path(i,3); % Mach Number

% Weight Normalization

Dist = D(i+3)-D(i+2);

W_in = W(X+Y);

TrueWeight = W(i+2)
```

```
M1 = Mach(i+2);
M2 = Mach(i+3);
A1 = Alt(i+2);
A2 = Alt(i+3);

%Calculated Fuel Burn

[EX,NW,F,FEXF,D_d] = cal_d(Dist,A1, M1, A2, M2, W_in, ISA, MCRA,MCRW,MCRM,MFCR ...
,MWDF,MMDD,MDDD,MADD,MWDF,MMDF,MFDF,MADF,MWDD,TrueWeight);

%Data Registration

EXDIST(i+3) = EX;
W(i+3) = NW;
FB(i+3) = F;
FEXFN(i+3) = FEXF;

end
% Approach and landing
A(X+Y+Z+4) = 0;
W(X+Y+Z+4)=W(X+Y+Z+3)-((W(X+Y+Z+3)-62)*(.392-.318)/(88-62)+.318);
D(X+Y+Z+4)=1200;
Mach(X+Y+Z+4)=0;
Alt(X+Y+Z+4) =A(X+Y+Z+4);

% Data presentation
plot3(D,Mach,Alt,'-');
xlabel('Distance in NM');
ylabel('Mach Number');
zlabel('Altitude in 1000ft');
title('3D Flight Profile');
grid
pause
plot(D,Alt,'-');
xlabel('Distance (NM)');
ylabel('Altitude in (1000ft)');
title('2D Flight Profile');
grid
pause
plot(D,W,'-');
xlabel('Distance (NM)');
ylabel('Weight (1000lb)');
title('Weight History of F100');
grid
pause
plot(Alt,W,'-');
xlabel('Altitude (1000ft)');
ylabel('Weight (1000lb)');
title('Weight History of F100');
grid
pause
```

2. Climb subroutine

```
function [EX,NW,F,FEXF,D_cb]= CAL_CB(Dist,A1, M1, A2, M2, W_in,WMX_cbd,MMX_cbd,AMX_cbd ...
,WMX_cbf,MMX_cbf,AMX_cbf,MCBF,MCRA,MCRW,MCRM,MFCR,MCBD,TrueWeight,ISAN);

global W31_cb_f b31_cb_f W32_cb_f b32_cb_f W33_cb_f ...
 b33_cb_f W31_cr b31_cr W32_cr b32_cr W33_cr b33_cr W31_cb_d ...
 b31_cb_d W32_cb_d b32_cb_d W33_cb_d b33_cb_d;

if W_in > 62;
if W_in <= 66;
W1 = 62;
W2 = 66;
end
end
if W_in > 66;
if W_in <= 70;
W1 = 66;
W2 = 70;
end
end

if W_in > 70;
if W_in <= 74;
W1 = 70;
W2 = 74;
end
end
if W_in > 74;
if W_in<= 78;
W1 = 74;
W2 = 78;
end
end

if W_in > 78;
if W_in <= 82;
W1 = 78;
W2 = 82;
end
end

if W_in > 82;
if W_in <= 86;
W1 = 82;
W2 = 86;
end
end

if W_in > 86;
if W_in <= 90;
W1 = 86;
W2 = 90;
end
end

if W_in > 90;
if W_in <=94
W1 = 90
W2 = 94
end
end
if W_in > 94;
if W_in <=98
W1 = 94;
W2 = 98;
end
end
if W_in > 98;
if W_in <= 102;
W1 = 98;
W2 = 102;
end
end
if W_in > 102;
if W_in <= 106;
W1 = 102;
W2 = 106;
end
end


% Mach Number Normalization

M1ND = M1/MMX_cbd;

M2ND = M2/MMX_cbd;
M1NF = M1/MMX_cbf;
M2NF = M2/MMX_cbf;


% Altitude Normalization
A1ND= A1/AMX_cbd;
A2ND= A2/AMX_cbd;
A1NF= A1/AMX_cbf;
A2NF= A2/AMX_cbf;

% Weight Normalization

W1ND = W1/WMX_cbd;
W2ND = W2/WMX_cbd;

W1NF = W1/WMX_cbf;
W2NF = W2/WMX_cbf;

P1D1 = [W1ND; M1ND;ISAN; A1ND ];

P2D1 = [W1ND; M2ND;ISAN; A1ND ];

P3D1 = [W1ND; M1ND;ISAN; A2ND ];
```

```
P4D1 = [W1ND; M2ND;ISAN; A2ND ];
P1F1 = [W1NF; M1NF;ISAN; A1NF ];
P2F1 = [W1NF; M2NF;ISAN; A1NF ];
P3F1 = [W1NF; M1NF;ISAN; A2NF ];
P4F1 = [W1NF; M2NF;ISAN; A2NF ];
F1F1 = simuff(P1F1,W31_cb_f,b31_cb_f,'logsig',W32_cb_f,b32_cb_f,'tansig',W33_cb_f,b33_cb_f,'purelin');
F1F2 = simuff(P2F1,W31_cb_f,b31_cb_f,'logsig',W32_cb_f,b32_cb_f,'tansig',W33_cb_f,b33_cb_f,'purelin');
F1F3 = simuff(P3F1,W31_cb_f,b31_cb_f,'logsig',W32_cb_f,b32_cb_f,'tansig',W33_cb_f,b33_cb_f,'purelin');
F1F4 = simuff(P4F1,W31_cb_f,b31_cb_f,'logsig',W32_cb_f,b32_cb_f,'tansig',W33_cb_f,b33_cb_f,'purelin');

D1D1 = simuff(P1D1,W31_cb_d,b31_cb_d,'logsig',W32_cb_d,b32_cb_d,'tansig',W33_cb_d,b33_cb_d,'purelin');
D1D2 = simuff(P2D1,W31_cb_d,b31_cb_d,'logsig',W32_cb_d,b32_cb_d,'tansig',W33_cb_d,b33_cb_d,'purelin');
D1D3 = simuff(P3D1,W31_cb_d,b31_cb_d,'logsig',W32_cb_d,b32_cb_d,'tansig',W33_cb_d,b33_cb_d,'purelin');
D1D4 = simuff(P4D1,W31_cb_d,b31_cb_d,'logsig',W32_cb_d,b32_cb_d,'tansig',W33_cb_d,b33_cb_d,'purelin');
D1_cb = ( ((D1D3+D1D4)-(D1D1+D1D2))/2)*MCBD;
F1 = ((F1F3+F1F4)/2-(F1F1+F1F2)/2)*MCBF/1000;
P1D2 = [W2ND; M1ND;ISAN; A1ND ];
P2D2 = [W2ND; M2ND;ISAN; A1ND ];
P3D2 = [W2ND; M1ND;ISAN; A2ND ];
P4D2 = [W2ND; M2ND;ISAN; A2ND ];
P1F2 = [W2NF; M1NF;ISAN; A1NF ];
P2F2 = [W2NF; M2NF;ISAN; A1NF ];
P3F2 = [W2NF; M1NF;ISAN; A2NF ];
P4F2 = [W2NF; M2NF;ISAN; A2NF ];
F2F1=simuff(P1F2,W31_cb_f,b31_cb_f,'logsig',W32_cb_f,b32_cb_f,'tansig',W33_cb_f,b33_cb_f,'purelin');
F2F2=simuff(P2F2,W31_cb_f,b31_cb_f,'logsig',W32_cb_f,b32_cb_f,'tansig',W33_cb_f,b33_cb_f,'purelin');
F2F3=simuff(P3F2,W31_cb_f,b31_cb_f,'logsig',W32_cb_f,b32_cb_f,'tansig',W33_cb_f,b33_cb_f,'purelin');
F2F4=simuff(P4F2,W31_cb_f,b31_cb_f,'logsig',W32_cb_f,b32_cb_f,'tansig',W33_cb_f,b33_cb_f,'purelin');
```

```
D2D1=simuff(P1D2,W31_cb_d,b31_cb_d,'logsig',W32_cb_d,b32_cb_d,'tansig',W33_cb_d,b33_cb_d,'purelin');

D2D2=simuff(P2D2,W31_cb_d,b31_cb_d,'logsig',W32_cb_d,b32_cb_d,'tansig',W33_cb_d,b33_cb_d,'purelin');

D2D3=simuff(P3D2,W31_cb_d,b31_cb_d,'logsig',W32_cb_d,b32_cb_d,'tansig',W33_cb_d,b33_cb_d,'purelin');

D2D4=simuff(P4D2,W31_cb_d,b31_cb_d,'logsig',W32_cb_d,b32_cb_d,'tansig',W33_cb_d,b33_cb_d,'purelin');

D2_cb = ( (D2D3+D2D4)/2-(D2D1+D2D2)/2)*MCBD;

F2 = (((F2F3+F2F4)-(F2F1+F2F2))/2 )*MCBF/1000;

F = F1+((F2-F1)/(W2-W1))*(W_in-W1);


D_cb = D1_cb+((D2_cb-D1_cb)/(W2-W1))*(W_in-W1);

if Dist< D_cb;

A2 = A2-0.5;


pause
else

TW = TrueWeight-F;


EX = Dist-D_cb ;% Extra distance required

% Normalize inputs

EXDMN = (M2)/MCRM;% Extra Distance Mach Normal


EXDAN = A2/MCRA;  % Extra Distance Altitude Normal

if TW > 62;
if TW <= 66;
W1 = 62;
W2 = 66;
end
end
if TW > 66;
if TW <= 70;
W1 = 66;
W2 = 70;
end
end

if TW > 70;
if TW <= 74;
```

```
W1 = 70;
W2 = 74;
end
end
if TW > 74;
if TW<= 78;
W1 = 74;
W2 = 78;
end
end

if TW > 78;
if TW <= 82;
W1 = 78;
W2 = 82;
end
end

if TW > 82;
if TW <= 86;
W1 = 82;
W2 = 86;
end
end

if TW > 86;
if TW <= 90;
W1 = 86;
W2 = 90;
end
end

if TW > 90;
if TW <=94
W1 = 90
W2 = 94
end
end
if TW > 94;
if TW <=98
W1 = 94;
W2 = 98;
end
end
if TW > 98;
if TW <= 102;
W1 = 98;
W2 = 102;
end
end
if TW > 102;
if TW <= 106;
W1 = 102;
W2 = 106;
```

```
end
end

TWN1= W1/MCRW; % Temporary Weight Normal
TWN2= W2/MCRW; % Temporary Weight Normal

PEXD1 = [TWN1; EXDMN; EXDAN]; % Input for the cruise network
   PEXD2 = [TWN2; EXDMN; EXDAN];

EXF1 =simuff(PEXD1,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin'); % Fuel Burn Estimation
EXF2 =simuff(PEXD2,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin'); % Fuel Burn Estimation

EXF = EXF1+((EXF2-EXF1)/(W2-W1))*(TW-W1);

FEXF = (inv((EXF*MFCR))*EX)/1000; % Actual Extra Fuel Burn

NW = TW - FEXF;

end
```

A.6.2 Cruise Cubroutine

```
function [NW,F]= cal_cr(Dist,A1, M1, A2, M2, W_in  ,MCRA,MCRW,MCRM,MFCR);

global W31_cr b31_cr W32_cr b32_cr W33_cr b33_cr;

% Mach Number Normalization

M1N = M1/MCRM;

M2N = M2/MCRM;


% Altitude Normalization

A1N= A1/MCRA;

A2N= A2/MCRA;


% Weight Normalization

if W_in > 62;
if W_in <= 66;
W1 = 62;
W2 = 66;
end
end
if W_in > 66;
if W_in <= 70;
W1 = 66;
W2 = 70;
```

```
end
end

if W_in > 70;
if W_in <= 74;
W1 = 70;
W2 = 74;
end
end
if W_in > 74;
if W_in<= 78;
W1 = 74;
W2 = 78;
end
end

if W_in > 78;
if W_in <= 82;
W1 = 78;
W2 = 82;
end
end

if W_in > 82;
if W_in <= 86;
W1 = 82;
W2 = 86;
end
end

if W_in > 86;
if W_in <= 90;
W1 = 86;
W2 = 90;
end
end

if W_in > 90;
if W_in <=94;
W1 = 90
W2 = 94
end
end
if W_in > 94;
if W_in <=98;
W1 = 94;
W2 = 98;
end
end
if W_in > 98;
if W_in <= 102;
W1 = 98;
W2 = 102;
end
```

```
end
if W_in > 102;
if W_in <= 106;
W1 = 102;
W2 = 106;
end
end

WN1 = W1/MCRW;

WN2 = W2/MCRW;

P1C1 = [WN1; M1N; A1N ];

P2C1 = [WN1; M2N; A1N ];

P3C1 = [WN1; M1N; A2N ];

P4C1 = [WN1; M2N; A2N ];

F1F1=simuff(P1C1,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin');

F1F2=simuff(P2C1,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin');

F1F3=simuff(P3C1,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin');

F1F4=simuff(P4C1,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin');


F1 = inv((F1F1+F1F2+F1F3+F1F4)/4*MFCR)*Dist;


P1C2 = [WN2; M1N; A1N ];

P2C2 = [WN2; M2N; A1N ];

P3C2 = [WN2; M1N; A2N ];

P4C2 = [WN2; M2N; A2N ];

F2F1=simuff(P1C2,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin');

F2F2=simuff(P2C2,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin');

F2F3=simuff(P3C2,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin');

F2F4=simuff(P4C2,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin');


F2 = inv((F2F1+F2F2+F2F3+F2F4)/4*MFCR)*Dist;

F = (F1+(F2-F1)/(W2-W1)*(W_in-W1))/1000;
```

```
NW = W_in-F;

end
```

A.6.3 Descent Subroutine

```
________________________________________
function [EX,NW,F,FEXF,D_d]= CAL_D(Dist,A1, M1, A2, M2, W_in, ISAN, MCRA,MCRW,MCRM,MFCR ...
,MWDF,MMDD,MDDD,MADD,MWDF,MMDF,MFDF,MADF,MWDD,TrueWeight);

global W31_df b31_df W32_df b32_df W33_df ...
 b33_df W31_dd b31_dd W32_dd b32_dd W33_dd ...
 b33_dd W31_cr b31_cr W32_cr b32_cr W33_cr b33_cr;



% Mach Number Normalization
if W_in > 58;
if W_in <= 66;
W1 = 58;
W2 = 66;
end
end
if W_in > 66;
if W_in <= 74;
W1 = 66;
W2 = 74;
end
end

if W_in > 74;
if W_in <= 82;
W1 = 74;
W2 = 82;
end
end
if W_in > 82;
if W_in <= 90;
W1 = 82;
W2 = 90;
end
end
if W_in > 90;
if W_in<= 98;
W1 = 90;
W2 = 98;
end
end



% Mach Number Normalization
```

```
M1ND = M1/MMDD;

M2ND = M2/MMDD;

M1NF = M1/MMDF;

M2NF = M2/MMDF;


% Altitude Normalization

A1ND= A1/MADD;

A2ND= A2/MADD;

A1NF= A1/MADF;

A2NF= A2/MADF;

% Weight Normalization

W1ND = W1/MWDD;
W2ND = W2/MWDD;

W1NF = W1/MWDF;
W2NF = W2/MWDF;

P1D1 = [W1ND; M1ND; A1ND;ISAN ];

P2D1 = [W1ND; M2ND; A1ND;ISAN ];

P3D1 = [W1ND; M1ND; A2ND;ISAN ];

P4D1 = [W1ND; M2ND; A2ND;ISAN ];

P1F1 = [W1NF; M1NF; A1NF;ISAN ];

P2F1 = [W1NF; M2NF; A1NF;ISAN ];

P3F1 = [W1NF; M1NF; A2NF;ISAN ];

P4F1 = [W1NF; M2NF; A2NF;ISAN ];

F1F1=simuff(P1F1,W31_df,b31_df,'logsig',W32_df,b32_df,'tansig',W33_df,b33_df,'purelin');

F1F2=simuff(P2F1,W31_df,b31_df,'logsig',W32_df,b32_df,'tansig',W33_df,b33_df,'purelin');

F1F3=simuff(P3F1,W31_df,b31_df,'logsig',W32_df,b32_df,'tansig',W33_df,b33_df,'purelin');

F1F4=simuff(P4F1,W31_df,b31_df,'logsig',W32_df,b32_df,'tansig',W33_df,b33_df,'purelin');


D1D1=simuff(P1D1,W31_dd,b31_dd,'logsig',W32_dd,b32_dd,'tansig',W33_dd,b33_dd,'purelin');
```

```
D1D2=simuff(P2D1,W31_dd,b31_dd,'logsig',W32_dd,b32_dd,'tansig',W33_dd,b33_dd,'purelin');

D1D3=simuff(P3D1,W31_dd,b31_dd,'logsig',W32_dd,b32_dd,'tansig',W33_dd,b33_dd,'purelin');

D1D4=simuff(P4D1,W31_dd,b31_dd,'logsig',W32_dd,b32_dd,'tansig',W33_dd,b33_dd,'purelin');

D1_d = ((D1D1+D1D2)/2-(D1D3+D1D4)/2)*MDDD;

F1 = ((F1F1+F1F2)/2 -(F1F3+F1F4)/2)*MFDF;

P1D2 = [W2ND; M1ND; A1ND;ISAN ];

P2D2 = [W2ND; M2ND; A1ND ;ISAN];

P3D2 = [W2ND; M1ND; A2ND ;ISAN];

P4D2 = [W2ND; M2ND; A2ND ;ISAN];

P1F2 = [W2NF; M1NF; A1NF;ISAN ];

P2F2 = [W2NF; M2NF; A1NF;ISAN ];

P3F2 = [W2NF; M1NF; A2NF;ISAN ];

P4F2 = [W2NF; M2NF;A2NF; ISAN ];

F2F1=simuff(P1F2,W31_df,b31_df,'logsig',W32_df,b32_df,'tansig',W33_df,b33_df,'purelin');

F2F2=simuff(P2F2,W31_df,b31_df,'logsig',W32_df,b32_df,'tansig',W33_df,b33_df,'purelin');

F2F3=simuff(P3F2,W31_df,b31_df,'logsig',W32_df,b32_df,'tansig',W33_df,b33_df,'purelin');
F2F4=simuff(P4F2,W31_df,b31_df,'logsig',W32_df,b32_df,'tansig',W33_df,b33_df,'purelin');


D2D1=simuff(P1D2,W31_dd,b31_dd,'logsig',W32_dd,b32_dd,'tansig',W33_dd,b33_dd,'purelin');

D2D2=simuff(P2D2,W31_dd,b31_dd,'logsig',W32_dd,b32_dd,'tansig',W33_dd,b33_dd,'purelin');

D2D3=simuff(P3D2,W31_dd,b31_dd,'logsig',W32_dd,b32_dd,'tansig',W33_dd,b33_dd,'purelin');

D2D4=simuff(P4D2,W31_dd,b31_dd,'logsig',W32_dd,b32_dd,'tansig',W33_dd,b33_dd,'purelin');

D2_d = ( (D2D1+D2D2)/2-(D2D3+D2D4)/2)*MDDD;

F2 = ((F2F1+F2F2)/2-(F2F3+F2F4)/2)*MFDF;

F = (F1+(F2-F1)/(W2-W1)*(W_in-W1))/1000;


D_d = D1_d+(D2_d-D1_d)/(W2-W1)*(W_in-W1);

if Dist<D_d;
```

```
A2 = A2-0.5;

pause
else

TW = (TrueWeight-F);


EX = Dist-D_d; % Extra distance required

% Normalize inputs

EXDMN = (M2)/MCRM;% Extra Distance Mach Normal


EXDAN = A2/MCRA;  % Extra Distance Altitude Normal

if TW > 62;
if TW <= 66;
W1 = 62;
W2 = 66;
end
end
if TW > 66;
if TW <= 70;
W1 = 66;
W2 = 70;
end
end

if TW > 70;
if TW <= 74;
W1 = 70;
W2 = 74;
end
end
if TW > 74;
if TW<= 78;
W1 = 74;
W2 = 78;
end
end

if TW > 78;
if TW <= 82;
W1 = 78;
W2 = 82;
end
end

if TW > 82;
if TW <= 86;
W1 = 82;
W2 = 86;
```

```
end
end

if TW > 86;
if TW <= 90;
W1 = 86;
W2 = 90;
end
end

if TW > 90;
if TW <=94
W1 = 90
W2 = 94
end
end
if TW > 94;
if TW <=98
W1 = 94;
W2 = 98;
end
end
if TW > 98;
if TW <= 102;
W1 = 98;
W2 = 102;
end
end
if TW > 102;
if TW <= 106;
W1 = 102;
W2 = 106;
end
end

TWN1= W1/MCRW; % Temporary Weight Normal
TWN2= W2/MCRW; % Temporary Weight Normal

PEXD1 = [TWN1; EXDMN; EXDAN]; % Input for the cruise network
    PEXD2 = [TWN2; EXDMN; EXDAN];

EXF1 =simuff(PEXD1,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin'); % Fuel Burn Estimation
EXF2 =simuff(PEXD2,W31_cr,b31_cr,'logsig',W32_cr,b32_cr,'tansig',W33_cr,b33_cr,'purelin'); % Fuel Burn Estimation

EXF = EXF1+((EXF2-EXF1)/(W2-W1))*(TW-W1);

FEXF = inv((EXF*MFCR))*EX/1000; % Actual Extra Fuel Burn

NW = TW - FEXF;


end
```

APPENDIX B

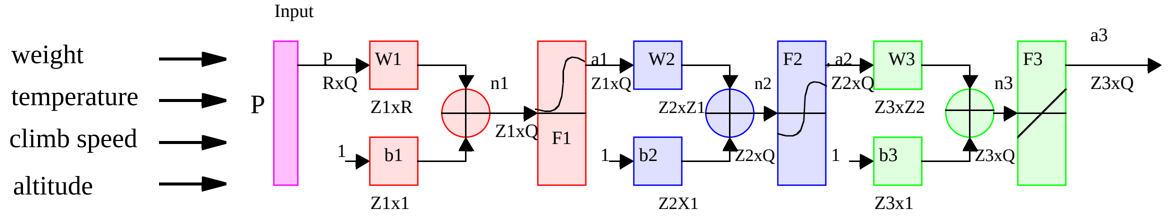
Neural Network Trained Matrices and Bias Vectors

This appendix contains a sample of weights and biases of the neural network estimated for the climb, cruise and descent phases. The data applies to the Fokker 100 aircraft.

B.1 Climb Neural Network Matrices

Recall that out network topolgy consists of 3 layers with a total of eight neurons per layer.

Figure B.1 Climb Fuel Consumption Neural Network.



B.1.1 First Layer Climb Fuel Weight Matrix (8 x 4)

Name of variable in source code: **W31_cb_f**

```
W31_cb_f = [1.4433   16.1224   -5.1308    4.1978
            0.7292  -56.4483    7.5036    6.4436
           -1.2606   31.7945    0.9047    5.3956
            1.3031   37.4262   -0.4690    3.6161
0.7410   29.0485    3.4602   -2.0486
-3.1726  -45.0995   -1.4852   -2.7352
1.9145  -11.7073   12.6932   -3.7325
1.3329   -9.5420    1.7506   -2.4465]
```

B.1.2 First Layer Bias Climb Fuel Vector (8 x 1)

Name of variable in source code: **b31_cb_f**

```
b31_cb_f' = [1-16.7970  1 43.9789 1 -31.6941  1-42.7644 1 -28.6348   147.15951 -2.3517 19.0268]
```

B.1.3 Second Layer Climb Fuel Weight Matrix (8 x 8)

Name of variable in source code: **W32_cb_f**

```
W32_cb_f = [-0.6005   -0.5870   -0.7480   -1.7733   -1.3437    0.2434    0.6188  1.4815
             -0.2233   -0.3751   -0.8264   -1.3783    -1.0784    0.4015    0.9850    -2.3304
             0.9568   -0.7226   -0.5406   -2.4272   -6.1678    1.6676    6.4060       4.1179
             -1.3189   -0.8735   -1.7891     -1.9017   -2.5168    0.5247    0.1467      -3.3560
            -1.2485   -0.9816   -0.8140   -1.7289   -2.2851   -0.0013    1.1800       -0.9163
             1.9170    0.5568    1.5908   -5.6420   -5.4944   -1.6828    4.8786       3.6204
            -0.9679    0.8287   -0.8533    6.6745    0.7747    0.0693    0.9815       -1.4368
             -4.0090   -2.7320   -2.2132   -4.5606    4.1227    2.9075   -2.7943       -5.9217]
```

B.1.4 Second Layer Bias Climb Fuel Vector (8 x 1)

Name of variable in source code: **b32_cb_f**

```
b32_cb_f' = [3.3329    3.1060   -3.0608    6.9925    3.9623   -2.7703   -0.8586    8.8345]
```

B.1.5 Third Layer Climb Fuel Weight Matrix (1 x 8)

Name of variable in source code: **W33_cb_f**

```
W33_cb_f = [-8.9400    2.8457    0.0614    1.4502    2.7007    0.2713   -1.0206  -5.7028]
```

B.1.6 Third Layer Bias Climb Fuel Vector (1 x 1)

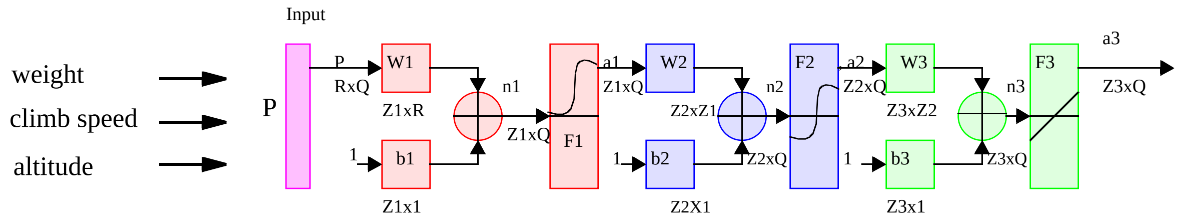
Name of variable in source code: **b33_cb_f**

```
b33_cb_f = [ 6.9078]
```

B.2 Cruise Neural Network Matrices

Recall that out network topolgy consists of 3 layers with a total of eight neurons per layer. In the cruise segment three inputs form the P vector: a) altitude, b) cruise mach number and c) weight. The temperature is implicitly modeled because SAR curves have a constant SAR adjustment factor for non-ISA conditions.

Figure B.2 Cruise Fuel Consumption Neural Network.



B.2.1 First Layer Cruise Fuel Weight Matrix (8 x 3)

Name of variable in source code: **W31_cr**

```
W31_cr = [ 6.3938   -9.8763    0.6649
          -0.1808    7.9561   -8.2477
           1.6851  -11.6343   -2.2276
          10.0500    7.7243   -9.4891
          -5.2912    6.2386   13.3908
           2.7012   12.7482    8.4157
          -9.7159    4.4772   -7.1510
          -7.0059   -1.2290   11.5976]
```

B.2.2 First Layer Bias Cruise Fuel Vector (8 x 1)

Name of variable in source code: **b31_cb_f**

```
b31_cr’ = [ 0.0156   -2.2324   12.6392   -7.5378  -10.2912  -10.7929   11.4901   -5.0219]
```

B.2.3 Second Layer Cruise Fuel Weight Matrix (8 x 8)

Name of variable in source code: **W32_cr**

```
W32_cr = [ -0.6202   -0.5070   -0.7396    0.2002    0.2389    0.1129   -0.1460   -1.1135
            0.4843   -0.0916   -0.7863   -0.4668    0.2533   -0.4232    0.6174   -2.2879
            0.0348   -0.0384   -0.3301    0.1160    0.5629   -0.0947   -0.0742   -0.4790
           -0.7224   -0.1714   -1.0539   -0.1012   -0.9725    0.2464    0.2292   -0.8991
           -1.6445  -19.3290   -8.5896    0.9265    8.6182    3.4082   -4.6841   -0.0982
           -0.4339   -1.0528    0.1404   -0.1411   -0.3719   -0.1410    0.1876   -1.3745
            0.3263   -0.3183   -0.8243   -0.4162    0.2460   -0.8841    1.2335   -2.6649
            0.4940    0.4168   -0.6962    1.1764   -0.2112    0.3459    0.5304   -2.1430]
```

B.2.4 Second Layer Bias Cruise Fuel Vector (8 x 1)

Name of variable in source code: **b32_cr**

b32_cr' = [0.8888 2.6767 0.2862 0.9785 3.4955 0.4767 0.1963 2.3233]

B.2.5 Third Layer Cruise Fuel Weight Matrix (1 x 8)

Name of variable in source code: **W33_cr**

W33_cr = [3.4282 5.3365 -5.4049 -1.7693 0.0286 -1.0977 0.5139 -5.8195]

B.2.6 Third Layer Bias Cruise Fuel Vector (1 x 1)

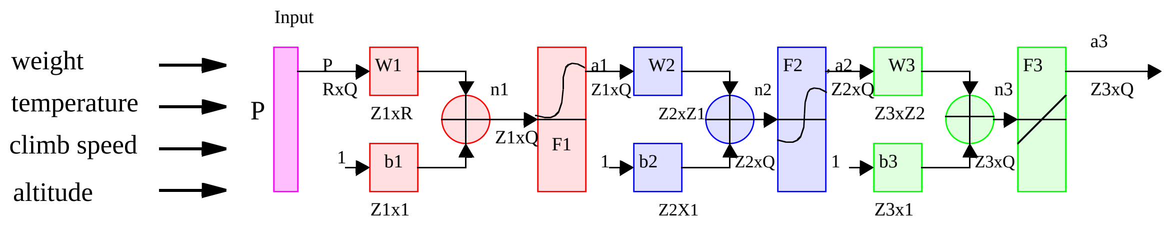
Name of variable in source code: **b33_cr**

b33_cr = [1.3720]

B.3 Descent Neural Network Matrices

This network topolgy consists of 3 layers with a total of twelve neurons per layer. In the descent segment four inputs form the P vector: a) altitude, b) cruise mach number,c) weight, and d) temperature.

Figure B.3 Descent Fuel Consumption Neural Network.



B.3.1 First Layer Descent Fuel Weight Matrix (12 x 4)

Name of variable in source code: **W31_df**

```
W31_df = [ 0.8511   52.8616   -8.6578    5.0650
           -1.5315   33.1232   -4.6842   -1.4033
           -0.0997  -58.3401  -13.7583   -4.5358
            0.1058  -39.4660    8.2775    5.8887
            0.5674   37.1619    2.0485    3.5484
            0.2326   15.8703    7.3495   -2.3180
           -0.5385   -0.4510 -184.0259   88.2574
           -3.8665   38.5448    2.7645    0.7520
            1.3531  -38.4882   -3.1255    8.8929
           -0.4177  -13.1107    9.7991   -2.1331
           -0.2185   53.2039   -5.8740    3.4922
            0.7447  -49.7325   -3.4859   -5.4073]
```

B.3.2 First Layer Bias Descent Fuel Vector (12 x 1)

Name of variable in source code: **b31_df**

```
b31_df’ = [ -49.3891  -26.7955   61.6388   31.1293  -40.9867  -21.9681   50.2503
            -31.9546   30.4832   12.1114  -43.6518   50.9938]
```

B.3.3 Second Layer Descent Fuel Weight Matrix (12 x 12)

Name of variable in source code: **W32_df**

```
W32_df = [ -0.3441    0.0356    0.1744    0.7858    1.6451    4.8408    0.8007
           -0.7591    1.6862   -0.8373    0.9596   -2.2921   -1.3884    0.1174
            4.1915    3.1200   -6.3542   -5.8100   10.2786   21.1127   -3.5153
            1.2613   -0.4000   -0.7177   -1.4973   -1.5431    1.2969    0.7868
           15.7700  -11.6365   -0.0752   25.4629    1.2772  -12.9458    6.1164
            1.2899   -4.1841    2.4559    8.0063   -6.3241    7.7492   -1.6877
           -0.0633    0.0252   -0.6305   -0.5820   -2.1185    1.5752    0.5957
           -0.9591   -0.4096    0.0772    0.5444    1.4693   -0.0506   -0.5681
           35.0054   -3.7473    3.6455  -30.2444  -45.5501  -39.7679   25.2169
           -1.6790   -0.3125    0.8318   -7.8237    6.7393    6.5104    1.4588
            0.0445    0.3169   -0.5943    2.7784   17.2762   -1.4082    0.0629
           -0.4590   -0.2112    0.1123    0.6804    1.5115    4.3249    0.7915

 Columns 8 through 12

           -0.0862    0.4148    0.3094   -1.0727    0.5149
           -0.1962   -0.9792    0.6816    2.6692    2.6671
           -5.0880   -7.2074  -13.2910    5.6920   -2.8350
            0.1593   -0.9023   -0.8160    0.2214   -0.0807
           11.6813  -12.2832   14.2854   24.6876   16.010
            3.4205   -5.7693    4.2349    2.5722   -7.0593
           -0.2754   -1.1515   -2.9731    1.0873   -2.8333
            0.0738    0.4027   -1.0044   -0.6905   -0.4023
           -0.5374    0.0539   17.5249    1.5512   -7.3604
            0.7041   -3.9677    0.1896   -4.2407   -3.9139
           -1.1530    0.4714   -0.8786    4.0312    2.7061
           -0.0035   -0.1634   -0.6757   -1.1861    0.1354]
```

B.3.4 Second Layer Bias Descent Fuel Vector (12 x 1)

Name of variable in source code: **b32_df**

```
b32_df’ = [ -2.3422   -1.4546    3.0836    2.4604  -61.3439   -9.2880    6.1743
             0.6685   19.4247    7.6869   -3.3949   -0.6705]
```

B.3.5 Third Layer Descent Fuel Weight Matrix (1 x12)

Name of variable in source code: **W33_df**

```
W33_df = [ 3.9943    0.6872    0.0370    2.3114    0.0130    0.1381   -1.5006
           3.0206    0.0454    0.1203    1.2498   -4.2761]
```