
Neural Network Aided Aviation Fuel Consumption Modeling

by

Wing Ho Cheung

Thesis submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Civil Engineering

APPROVED:

Dr. A. A. Trani, Chairman

Dr. D. R. Drew

Dr. R. G. Greene

Blacksburg, August 1997

Key words: neural network, fuel consumption, aviation

Neural Network Aided Aviation Fuel Consumption Modeling

By

Wing Ho Cheung

Committee Chairman: Dr. Antonio Trani

Civil Engineering

<ABSTRACT>

This thesis deals with the potential application of neural network technology to aviation fuel consumption estimation. This is achieved by developing neural networks representative jet aircraft. Fuel consumption information obtained directly from the pilot's flight manual was trained by the neural network. The trained network was able to accurately and efficiently estimate fuel consumption of an aircraft for a given mission. Statistical analysis was conducted to test the reliability of this model for all segments of flight. Since the neural network model does not require any wind tunnel testing nor extensive aircraft analysis, compared to existing models used in aviation simulation programs, this model shows good potential. The design of the model is described in depth, and the *MATLAB* source code are included in appendices.

ACKNOWLEDGEMENTS

I would like to thank Dr. Antonio Trani, the chairman of my advisory committee, for his continual effort and guidance throughout this research. I would like to express my sincere appreciation to my advisory committee members Dr. D.R.Drew and Dr. R.G. Greene for their kind review of this document and their valuable remarks.

I would also like to thank Mr.Ahsen Abbasi for his technical opinions contributed to this document.

Finally, I give my greatest appreciation to my parents, Yat Ming and Pik Shan Cheung. They taught me the value of education and their support and encouragement allowed me to get this far in my academic career.

Table of Contents

CHAPTER 1	<i>Introduction</i>	<i>1-1</i>
	1.1 Background	1-1
	1.2 Existing Fuel Consumption Model	1-1
	1.3 Neural Network and Fuel Consumption Modeling	1-2
	1.4 Research Objectives and Approach	1-2
CHAPTER 2	<i>Literature Review</i>	<i>2-1</i>
	2.1 Aircraft Performance Based Fuel Consumption Model	2-2
	2.1.1 Physical Properties of Air	2-2
	2.1.2 International Standard Atmosphere (ISA)	2-3
	2.1.3 Speed of Sound and Mach Number	2-3
	2.1.4 Aircraft Performance	2-4
	2.1.5 Fuel Consumption of an Aircraft	2-6
	2.1.6 Existing Fuel Consumption Model	2-7
	2.2 Neural Network Aided Fuel Consumption	2-14
	2.2.1 Introduction to Neural Network	2-14
	2.2.2 Neural Learning Using Back-Propagation	2-19
	2.2.3 Learning Rule and Lavenberg Marquardt Optimization Algorithm	2-20
CHAPTER 3	<i>Methodology</i>	<i>3-1</i>
	3.1 Selection of the Testing Aircraft	3-2
	3.2 Data Collection from the Flight Manual	3-2
	3.3 Training the Neural Network	3-3
	3.3.1 Selection of a Programming Language	3-3
	3.3.2 MATLAB Basics	3-3
	3.3.3 Selection of Algorithms	3-7
	3.3.4 Design of the Appropriate Neural Network Topology	3-8
	3.4 Implementation and Generalization of the Neural Network	3-9
	3.4.1 Description of the Neural Network Aided Fuel Consumption Model	3-10
	3.4.2 Summary	3-13
	3.5 Results Justification	3-14
	3.5.2 Mean and Standard Deviation	3-15
	3.5.3 Histogram	3-16
	3.5.4 t-test	3-16
	3.5.5 Implementation of Statistical Analysis	3-17
	3.6 Implementation of the Outputs to the Actual Flight Trajectory	3-18

CHAPTER 4	<i>Model Implementation</i>	4-1
	4.1 Training of Weight Matrix For Different Flight Phases	4-2
	4.2 Testing Preliminary Results Using Statistics	4-4
	4.3 Implementation of Neural Network to Actual Flight Trajectory	4-6
CHAPTER 5	<i>Discussion of Results</i>	5-1
	5.1 Training Results	5-2
	5.2 Testing Results	5-6
	5.3 Neural Network Aided Fuel Consumption Results	5-19
CHAPTER 6	<i>Conclusions and Recommendations</i>	6-1
	6.1 Conclusions	6-1
	6.2 Remarks and Recommendations	6-2
	6.2.1 Remarks	6-2
	6.2.2 Recommendations	6-3
CHAPTER 6	<i>Vita</i>	7-1
	<i>Bibliography</i>	<i>Bi-1</i>
	<i>Appendix A: MATLAB Programs</i>	<i>A-1</i>
	<i>Appendix B: Neural Network Trained Matrices and Bias Vectors</i>	<i>B-1</i>
	<i>Appendix C: Statistical Results</i>	<i>C-1</i>
	<i>Appendix D: Fuel Efficient Path</i>	<i>D-1</i>

List of Figures

Figure 2.1 Velocity profile.....	page.....	2-3
Figure 2.2 Force diagram of an aircraft.....		2-5
Figure 2.3 Sketch of a drag polar.....		2-6
Figure 2.4 Comparative performance of turbo-propeller and turbo-fan.....		2-6
Figure 2.5 Neural network.....		2-15
Figure 2.6 Architecture of neural network.....		2-16
Figure 2.7 A single neuron.....		2-17
Figure 2.8 Typical transfer function.....		2-17
Figure 2.9 Weight training.....		2-18
Figure 2.10 Back-propagation cycle.....		2-20
Figure 3.1 Fokker F-100.....		3-2
Figure 3.2 General three layers neural network.....		3-10
Figure 3.4 Fuel estimation of procedure for take-off and climb out.....		3-11
Figure 3.5 Fuel estimation procedure for climb.....		3-12

List of Figures

Figure 3.6 Fuel estimation procedure for cruise.....	3-12
Figure 3.7 Select neural network architecture.....	3-19
Figure 4.1 Example of input vector.....	4-2
Figure 4.2 Flow chart illustrating neural network training process.....	4-3
Figure 4.3 Flow chart illustrating neural network testing process.....	4-5
Figure 4.4 Illustration of way-points on trajectory.....	4-6
Figure 4.5 Possible phase scenarios.....	4-8
Figure 4.6 Flow chart for neural network aided fuel consumption.....	4-10
Figure 5.1 Training climb distance.....	5-3
Figure 5.2 Training climb fuel.....	5-4
Figure 5.3 Training cruise fuel.....	5-5
Figure 5.4 Climb fuel testing.....	5-8
Figure 5.5 Testing climb distance.....	5-9
Figure 5.6 Testing flight envelope.....	5-10
Figure 5.7 Climb fuel comparison.....	5-12
Figure 5.8 Climb distance comparison.....	5-13
Figure 5.9 Specific air-range comparison.....	5-14
Figure 5.10 Descent distance comparison.....	5-15
Figure 5.11 Descent fuel comparison.....	5-16
Figure 5.12 Histogram of error for climb fuel estimation.....	5-18
Figure 5.13 Three dimensional flight profile.....	5-22
Figure 5.14 Two-dimensional flight profile.....	5-23
Figure 5.15 Weight verses distance.....	5-24
Figure 5.16 Weight verses altitude.....	5-25
Figure C.1 Error Distribution of Climb Distance Estimation.....	C-2

Figure C.2 Error Distribution of Cruise Specific Air Range.....	C-3
Figure C.3 Error Distribution of Descent Distance Estimation.....	C-4
Figure C.4 Error Distribution of Descent Fuel Estimation.....	C-5
Figure D.1 Matrix Definition.....	D-1
Figure D.2 Matrix Simplification.....	D-2
Figure D.3 X3 Step.....	D-2
Figure D.4 X2 Step.....	D-3
Figure D.5 X2 Step Cont'.....	D-3
Figure D.6 X1 Step.....	D-4
Figure D.7 Optimal Flight Path.....	D-4
Figure D.8 Calculation Procedure.....	D-8

List of Tables

Table 3.1 Comparison of optimization algorithm.....	Page.....	3-8
Table 3.2 Results of topology study.....		3-9
Table 3.3 Summary of neural network training and testing.....		3-13
Table 4.1 Summary of input and output parameters.....		4-3
Table 5.1 Training data sets.....		5-2
Table 5.2 Testing data sets.....		5-6
Table 5.3 Summary of errors for all flight phases.....		5-19
Table 5.4 Flight trajectory		5-21

1.1 Background

The Federal Aviation Administration (FAA) airport and airspace simulation model (SIMMOD) utilizes a fuel consumption post processor that computes the fuel consumption of an aircraft given a flight profile [Collins 1982]. SIMMOD utilizes the advanced fuel burn model-MOD 830725 developed by Bela P Collins at the MITRE Corporation in the early 1980's. The theme for this research is to seek for possible ways to enhance the fuel consumption estimation process by reducing the complexity of the existing fuel-burn model 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

The 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 non-linear 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 manual of individual aircraft are presented to the network. The network, by an iterative process, self-organizes and generalizes its own performance data. This process is referred to as “network learning”. When sufficient amount of data are presented to the network, the network will become a “trained network” capable of estimating the performance of aircraft in fuel consumption. Neural network training techniques utilized will be presented in Chapters 3 and 4.

1.4 Research Objectives and Approach

The objectives of this research project are:

1. To investigate the advantages and disadvantages of the existing aviation fuel consumption model.
2. To use the information provided in the pilot flight manual, along with neural network methodology to develop an accurate fuel consumption model.
3. To examine results obtained from the neural network assisted fuel consumption model.
4. To determine whether neural network should be considered as a viable alternative in fuel consumption estimating applications.

To attain these objectives, the existing model based on aircraft performance parameters is studied. The advantages and disadvantages of this particular model are reviewed. A representative neural network model is then selected by comparing results obtained from different models. Along with this model, aircraft’s information given in the pilot’s flight manual is digitized and presented to the neural network. Results obtained from the network are then compared to the actual performance in the flight manual for justification.

The purpose of the literature review is to present the basic fundamentals of aviation fuel consumption modeling. The literature review of this thesis conveys background information for two major subjects:

1. Aircraft performance based fuel consumption model.
2. Neural network aided fuel consumption model.

For the first part of this chapter, the aircraft performance based fuel consumption model and vital information related to this model will be presented. The second part of this chapter focuses on issues related to the basic theory and optimization techniques for neural network computing.

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 Bela 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 [Anderson 1989]

The physical properties of air included are:

1. temperature (T)
2. pressure (P)
3. density (ρ)
4. viscosity (ν)

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 ($^{\circ}C$), degree Rankine ($^{\circ}R$), and degree Fahrenheit ($^{\circ}F$). 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 $\tau = \nu \left(\frac{d(velocity)}{d(y)} \right)$, where ν 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 ($slug/ft/sec$).

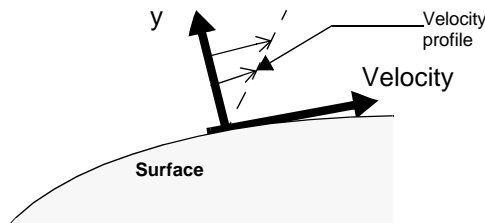


Figure 2.1 Velocity Profile.

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

$$p = \rho RT$$

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 [Anderson 1989]

Speed of sound plays a pivotal role in 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:

$$\rho_1 A_1 V_1 = \rho_2 A_2 V_2 \quad (\text{continuity equation})$$

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:

$$dp = -\rho V dV$$

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

$$a = \sqrt{\left(\frac{dp}{d\rho}\right)_{isentropic}} = \sqrt{\gamma RT}$$

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:

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

M is one of the most powerful 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 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.

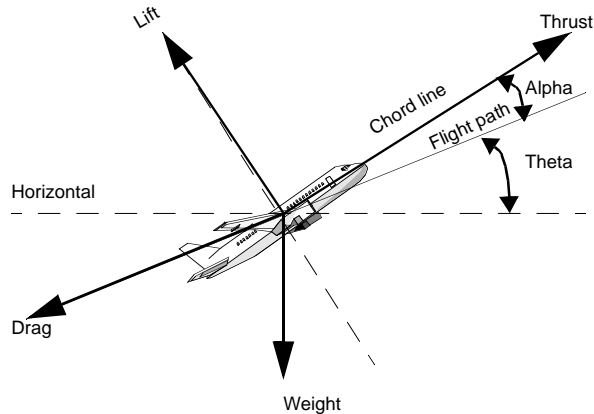


Figure 2.2 Force Diagram of an Aircraft.

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]:

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

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

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:

$$T \cos(\alpha) = D$$

$$L + T \sin(\alpha) = W$$

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

$$T = D$$

$$L = W$$

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 Fuel Consumption of an Aircraft

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

1. aerodynamics characteristics
2. engine type
3. 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.

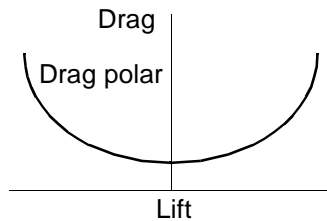


Figure 2.3 Sketch of a Drag Polar.

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.

2.1 Aircraft Performance Based Fuel Consumption Model

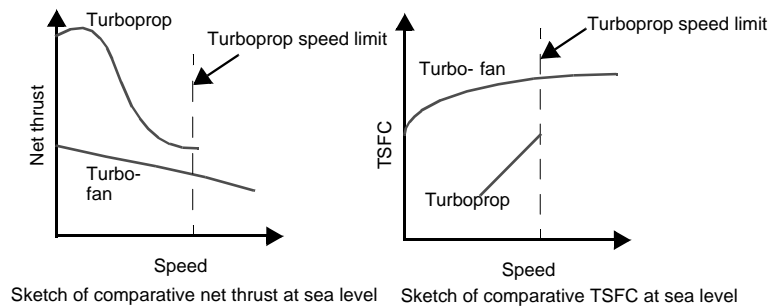


Figure 2.4 Comparative Performance of Turbo-Propeller and Turbo-Fan.

Fuel consumption of an aircraft engine is a function of the powerplant utilized and thrust required from the engine to perform certain tasks at certain altitudes and speeds. Thrust required is a function of the aerodynamic characteristics of an aircraft. The aerodynamic characteristics of an aircraft depend on the aircraft configuration, speed and altitude. Therefore, fuel consumption of an aircraft depends on the aerodynamic characteristics, engine type and the mission profile which defines the speed and altitude schedule of a mission.

2.1.6 Existing Fuel Consumption Model [Collins 1982]

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.

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.

Assumptions and Definitions

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:

$$\cos(\phi) \approx 1$$

- 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:

$$(\text{energy change}) = (\text{energy in}) - (\text{energy loss})$$

or, when expressed in terms of aircraft flight:

$$(\text{change in potential energy} + \text{change in kinetic energy}) = \text{work done by the thrust} - \text{work done by the drag}$$

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:

$$(\text{change in potential energy} + \text{change in kinetic energy}) = \text{thrust force} \times \text{distance along the vector} - \text{drag force} \times \text{distance along the vector}$$

$$\Delta PE + \Delta KE = (F_n \times d) - (D \times d)$$

Solving this expression for thrust F_n

$$F_n = \frac{\Delta KE}{d} + \frac{\Delta PE}{d} + D \quad \text{EQ 2.1}$$

where,

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

The distance along the velocity can be expressed as

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

where,

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

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:

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

Solving for C_L ,

$$C_L = \frac{W}{\frac{1}{2}\rho V^2 S_w} \quad \text{EQ 2.2}$$

Similarly, the drag can be written as,

$$D = \frac{1}{2}\rho V^2 S_w C_D \quad \text{EQ 2.3}$$

where,

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

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:

$$C_D = \sum C_{D_p} + C_{D_i} + C_{D_r} + C_{D_i} + C_{D_p c_L} + C_{D_c} \quad \text{EQ 2.4}$$

where,

$\sum C_{D_{p_{min}}}$ = Summation of the minimum profile drag of the individual aircraft components, for smooth turbulent attached flow.

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:

$$C_D = M_a + M_b C_L^2 + M_c C_L^4 \quad \text{EQ 2.5}$$

where M_a , M_b , and M_c are functions of Mach number. Hence, M_a represents the first four terms of EQ 2.4, $M_b C_L^2$ represents the fifth and sixth term of EQ 2.4 and $M_c C_L^4$ represents the last term. These three M_i functions are defined as follows,

$$M_a = C_1 + C_2 \Gamma^2 + C_3 \Gamma^4$$

$$M_b = C_4 + C_5 \Gamma + C_6 \Gamma^2 + C_7 \Gamma^3 + C_8 \Gamma^4$$

2.1 Aircraft Performance Based Fuel Consumption Model

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

where,

$$\Gamma = \frac{1+M}{1-M} \text{ and } C_i \text{ are drag constants}$$

According to basic aerodynamics, drag will increase 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 EQ 2.3 can be re-written as,

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

where the M functions are as previously defined.

The energy balance equation appearing in EQ 2.1 now becomes,

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

by substituting for EQ 2.2

$$F_n = \frac{\Delta KE}{d} + \frac{\Delta PE}{d} + \frac{1}{2}\rho V^2 S_w M_a + M_b \left(\frac{W}{\frac{1}{2}\rho V^2 S_w} \right)^2 + M_c \left(\frac{W}{\frac{1}{2}\rho V^2 S_w} \right)^4 \quad \text{EQ 2.5}$$

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:

$$R_1 = 1 + (CF)(CC_1) + (CC_2)(GF) + (CC_3)(FA)(GF) + (CC_4)(FA) + (CC_5)(FA)^2 + (CC_6)(FA)^3$$

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

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

The final drag equation becomes:

$$F_n = \frac{\Delta KE}{d} + \frac{\Delta PE}{d} + \frac{1}{2}\rho V^2 S_w M_a R_1 + R_2 M_b \left(\frac{W}{\frac{1}{2}\rho V^2 S_w} \right)^2 + R_3 M_c \left(\frac{W}{\frac{1}{2}\rho V^2 S_w} \right)^4 \quad \text{EQ 2.6}$$

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 turbo jet engine is shown below:

$$\dot{\lambda} = \eta_1 + \eta_2 F_n + \eta_3 \eta F_n^2 \quad \text{EQ 2.7}$$

where the η functions are,

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

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

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

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. 2.6 can be substituted into EQ. 2.7. The advanced fuel burn model MOD830725 core equation is shown as follows:

2.1 Aircraft Performance Based Fuel Consumption Model

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

The core equation combines all aerodynamics 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 wind tunnel testing and/or flight testing. This is because aircraft aerodynamics and engine properties at low Mach number and high Mach number are substantially different. Theoretical models for lift and drag are usually limited to incompressible flows of air, that is, to Mach numbers less than approximately 0.4. 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 of aircraft was first introduced by Mr. Glenn Schilling [Schilling, 1996]. Mr. Schilling used Collin's model to calculate the fuel burn, and then used these results to train the neural network assisted model. Mr. Schilling proved the potential of using a neural network to estimate fuel consumption of aircraft. The idea of this research project is to expand Mr. 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 be 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 Network

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. Figure 2.5 shows the functionality of the neural network.

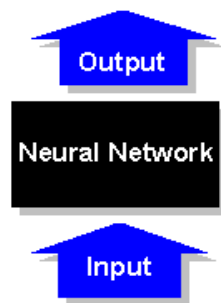


Figure 2.5 Neural Network.

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 2.6 illustrates the simplified architecture of a neural network.

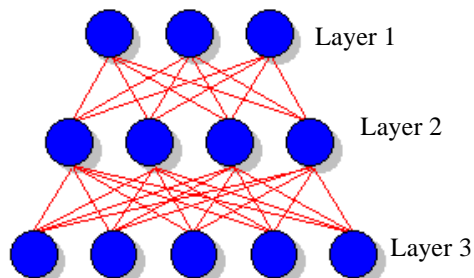


Figure 2.6 General Architecture of Neural Network.

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_j) and bias (b_j) 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.

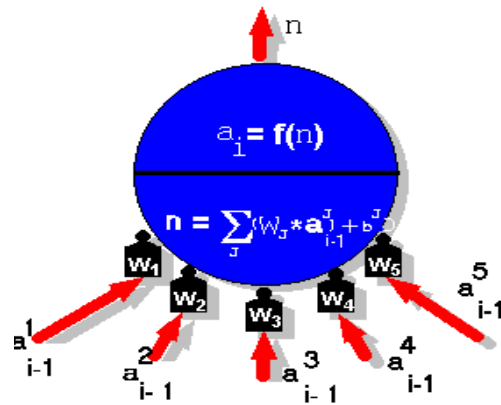


Figure 2.7 A Single Neuron.

Transfer Functions [Mathworks 1997]

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 2.8:

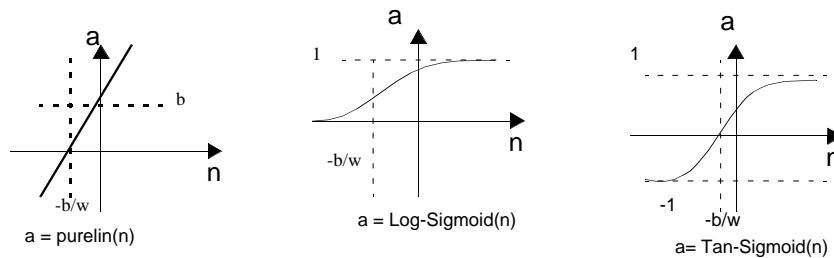


Figure 2.8 Typical Transfer Functions.

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

$$\begin{aligned} \text{Pure-linear: } a &= n \\ \text{Log Sigmoid: } a &= \frac{1}{1 + e^{-n}} \\ \text{Tangent Sigmoid } a &= \frac{e^n - e^{-n}}{e^n + e^{-n}} \end{aligned}$$

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. Figure 2.9 depicts how the weights are trained.

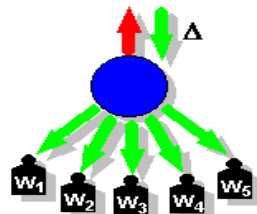


Figure2.9 Weight Training.

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 [Freeman 1992]

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.

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 2.10. For a finite number of epochs the weights and biases are shifted until the deviations from the outputs are minimized.

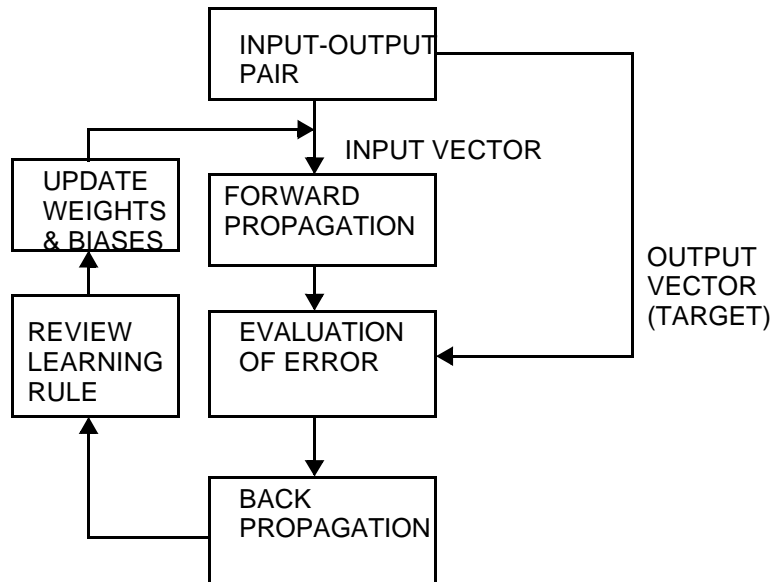


Figure 2.10 Back-Propagation Cycle.

2.2.3 Learning Rule and Lavenberg Marquardt Optimization Algorithm [Hagan 1996]

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. 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:

$$\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{p}_k \quad \text{EQ.2.8}$$

where \vec{p}_k is the search direction and α_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 .

Steepest Descent

Consider the first order Taylor series expansion $Z(\hat{x})$ about the old guess \vec{x}_k :

$$Z(\hat{x}_{k+1}) = Z(\hat{x}_k + \Delta \hat{x}_k) = Z(\hat{x}_k) + \vec{g}_k^T \Delta \hat{x}_k \quad \text{EQ.2.9}$$

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

$$\vec{g}_k \equiv \nabla Z(\hat{x}) \Big|_{\hat{x} = \vec{x}_k} \quad \text{EQ.2.10}$$

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

$$\vec{g}_k^T \Delta \hat{x}_k = \alpha_k \vec{g}_k^T \vec{p}_k < 0 \quad \text{EQ.2.11}$$

If α_k is a small but positive value, then

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

The above vector will be most negative if:

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

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

$$\hat{x}_{k+1} = \hat{x}_k - \alpha_k \vec{g}_k \quad \text{EQ.2.12}$$

Learning rate, α_k can be determined by:

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

Newton's Method

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

$$Z(\hat{x}_{k+1}) = Z(\hat{x}_k) + \hat{g}_k^T \Delta \hat{x}_k + \frac{1}{2} \Delta \hat{x}_k^T \hat{B}_k \Delta \hat{x}_k \quad \text{EQ.2.13}$$

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

$$\Delta \hat{x}_k = -\hat{B}_k^{-1} \hat{g}_k \quad \text{EQ.2.14}$$

Newton's method is then defined as:

$$\hat{x}_{k+1} = \hat{x}_k - \hat{B}_k^{-1} \hat{g}_k \quad \text{EQ.2.15}$$

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:

$$Z(\hat{x}) = \sum_{i=1}^{iv} \zeta_i^2(\hat{x}) = \xi^T(\hat{x}) \zeta(\hat{x}) \quad \text{EQ.2.16}$$

and the j^{th} element of the gradient can be written as

$$[\nabla Z(\hat{x})]_j = \frac{\partial}{\partial \hat{x}_j} Z(\hat{x}) = 2 \sum_{i=1}^{iv} \zeta_i(\hat{x}) \frac{\partial}{\partial \hat{x}} \zeta_i(\hat{x}) \quad \text{EQ.2.17}$$

The gradient can also be written in matrix form:

$$\nabla Z(\hat{x}) = 2J^T(\hat{x}) \zeta(\hat{x}) \quad \text{EQ.2.18}$$

where \mathbf{J} is the Jacobian matrix as shown below:

$$J(\hat{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} \zeta_1(\hat{x}) & \frac{\partial}{\partial x_2} \zeta_1(\hat{x}) & \dots & \frac{\partial}{\partial x_n} \zeta_1(\hat{x}) \\ \frac{\partial}{\partial x_1} \zeta_2(\hat{x}) & \frac{\partial}{\partial x_2} \zeta_2(\hat{x}) & \dots & \frac{\partial}{\partial x_n} \zeta_2(\hat{x}) \\ \dots & \dots & \dots & \dots \\ \frac{\partial}{\partial x_1} \zeta_N(\hat{x}) & \frac{\partial}{\partial x_2} \zeta_N(\hat{x}) & \dots & \frac{\partial}{\partial x_n} \zeta_N(\hat{x}) \end{bmatrix}$$

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

$$[\nabla^2 Z(\hat{x})]_{k,j} = \frac{\partial^2}{\partial x_k \partial x_j} Z(\hat{x}) = 2 \sum_{i=1}^N \left\{ \frac{\partial}{\partial x_k} \zeta_i(\hat{x}) \frac{\partial}{\partial x_j} \zeta_i(\hat{x}) + \zeta_i(\hat{x}) \frac{\partial^2}{\partial x_k \partial x_j} \zeta_i(\hat{x}) \right\}. \quad \text{EQ.2.19}$$

The Hessian matrix can then be expressed in matrix form:

$$\nabla^2 Z(\hat{x}) = 2J^T(\hat{x})J(\hat{x}) + 2S(\hat{x}) \quad \text{EQ.2.20}$$

where

$$S(\hat{x}) = \sum_{i=1}^N \zeta_i(\hat{x}) \nabla^2 \zeta_i(\hat{x})$$

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

$$\nabla^2 Z(\hat{x}) \cong 2J^T(\hat{x})J(\hat{x}) \quad \text{EQ.2.21}$$

By substituting equation (2.21) and equation (2.18) into equation (2.15), the following can be obtained:

$$\begin{aligned} \hat{x}_{k+1} &= \hat{x}_k - [2J^T(\hat{x}_k)J(\hat{x}_k)]^{-1} 2J^T(\hat{x}_k)\zeta(\hat{x}_k) \\ &= \hat{x}_k - [J^T(\hat{x}_k)J(\hat{x}_k)]^{-1} J^T(\hat{x}_k)\zeta(\hat{x}_k) \end{aligned} \quad \text{EQ.2.22}$$

Equation(2.22) 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 weak-

ness with the Gauss-Newton method, which is the matrix $H = J^T J$ may not be 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 \mathbf{H} are $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ and $\{\xi_1, \xi_2, \dots, \xi_n\}$. Then:

$$G\xi_i = [H + \mu I]\xi_i = H\xi_i + \mu\xi_i = \lambda_i\xi_i + \mu\xi_i = (\lambda_i + \mu)\xi_i. \quad \text{EQ.2.23}$$

Therefore the eigenvectors of \mathbf{G} are the same as the eigenvectors of \mathbf{H} , and the eigenvalues of \mathbf{G} are $(\lambda_i + \mu)$. \mathbf{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:

$$\hat{x}_{k+1} = \hat{x}_k - [J^T(\hat{x}_k)J(\hat{x}_k) + \mu_k I]^{-1} J^T(\hat{x}_k)\zeta(\hat{x}_k) \quad \text{EQ.2.25}$$

or

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

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

$$\hat{x}_{k+1} \cong \hat{x}_k - \frac{1}{\mu_k} J^T(\hat{x}_k)\zeta(\hat{x}_k) = \hat{x}_k - \frac{1}{2\mu_k} \nabla Z(\hat{x}) \quad \text{EQ.2.26}$$

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 \hat{t} . The algorithm should adjust the network parameters in order to minimize the mean squared error:

$$F(\hat{x}) = \Phi[e^2] = \Phi[(t - a)^2] \quad \text{EQ.2.27}$$

Here a is the output of the tested value and \mathbf{x} is the vector of network weights and biases.

If the network has multiple outputs this generalizes to

$$F(\hat{x}) = \Phi[\vec{e}^T \vec{e}] = \Phi[(\hat{t} - \hat{a})^T (\hat{t} - \hat{a})] \quad \text{EQ.2.28}$$

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:

$$\begin{aligned} \widehat{F}(\hat{x}) &= \sum_{q=1}^Q (\vec{t}_q - \vec{a}_q)^T (\vec{t}_q - \vec{a}_q) & \text{EQ.2.29} \\ &= \sum_{q=1}^Q \vec{e}_q^T \vec{e}_q = \sum_{i=1}^N (\zeta_i)^2 \end{aligned}$$

Equation (2.29) is equivalent to performance index, equation(2.16), for which Levenberg-Marquardt was designed. Therefore it should be a straight forward matter to adapt the algorithm for network training.

This chapter outlines the procedures used to develop the model for estimating aircraft fuel consumption using data given in the flight manual. The general approach as well as the means and methods that were used to achieve the goals of this thesis are outlined through the following 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 outputs from the neural network to fuel consumption calculation
5. Results justification
6. Implementation of the outputs to the actual flight trajectory

3.1 Selection of the Testing Aircraft

For the purpose of this research a medium sized jet aircraft was selected for testing. The basic plane characteristics of this particular aircraft are shown in Figure 3.1 [Fokker 1993]. The Fokker F-100 is powered by two Rolls-Royce Tay Mk650 turbo-fan engines. Its has been certified for operations at altitudes up to 35000 ft and take-off weights up to 98000 lb. In addition the minimum rate of climb is 300 ft/sec, maximum operating speed 320 kts. or Mach 0.77.

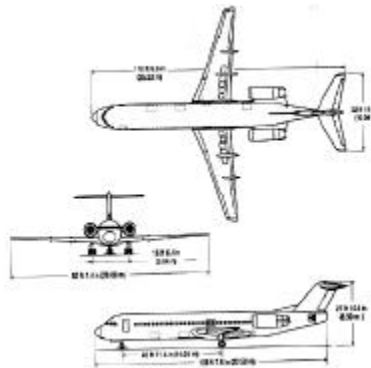


Figure 3.1 Basic Plane Characteristics.

3.2 Data Collection from the Flight Manual

The flight manual consists of different charts related to fuel consumption which are digitized through a scanner. Using a program known as "Data thief", raw data is obtained. The next step is to sort the raw data obtained according to the corresponding flight phase, Mach number, altitude, weight and international standard atmosphere (ISA) conditions.

3.3 Training the Neural Network

3.3.1 Selection of a Programming Language [MATLAB 1995], [Mathworks 1994]

The implementation of neural networks can be expedited with the use of a commercially available software. Examples of these are *Neural Forecaster*, *WinCrain* and *Neuralyst*. Another approach is to code networks in high level computer programming languages such as C or PASCAL. Programming in *MATLAB* could be considered an intermediate approach for experimenting with neural networks. This approach lies closer to the programming approach than it does to the prewritten, commercial-software approach.

Programs that were developed in *MATLAB* to perform neural net computation will enable us to perform the following tasks:

1. Network training/learning.
2. Testing and evaluation of trained network.
3. Implementation to calculate fuel consumption of an aircraft.

For any given problem, the data will be split into a learning set and testing set. The program will require the user to give the following:

1. The number of inputs.
2. A value for the learning coefficient.
3. The number of processing elements (neurons) in the hidden layer and output layers.
4. The maximum number of cycles (epochs) for each run.
5. The required maximum allowable sum squared error for each run.

3.3.2 MATLAB Basics

MATLAB is a powerful and advanced computational software developed by MathWorks Inc.

MATLAB Features:

- Heavy-duty numeric computation
- Advanced graphics and visualization
- A high level programming language based on vectors, arrays, and matrices
- A useful collection of application functions

The following are some important *MATLAB* notations used to construct the learning and testing functions of the neural network program:

1. Initff

Purpose:

Initializes feed-forward network.

Synopsis:

$[w1,b1,w2,b2,w3,b3] = \text{initff}(P,S1,F1,S2,F2,S3,T)$

Description

$\text{initff}(P,S,T)$ - 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.

2. trainlm

Purpose:

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

Synopsis:

$[w1,b1,w2,b2,w3,b3] = \text{trainlm}(w1,b1,'F1',w2,b2,'F2',w3,b3,'F3',P,T,tp)$

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

3.3 Training the Neural Network

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-Marquardt 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 or gradient descent methods. Here is the Levenberg-Marquardt rule for updating parameters (such as weights and biases):

$$\Delta W = (J^T J + \mu I)^{-1} J^T e$$

where J is the Jacobian matrix, as discussed in Chapter 2. Note that, as the error e gets large the $J^T J$ term becomes negligible and learning progresses according to $\mu^{-1} J^T e$ becomes a gradient descent method. 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.

3. 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.

4. 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 applied to each input element:

$$\text{logsig}(n) = \frac{1}{1 + e^{-n}}$$

5. 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:

$$\text{tansig}(n) = \tanh(n)$$

6. 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.

3.3.3 Selection of Algorithms

Based on one of the studies using a demonstration package provided by *MATLAB*, Levenberg-Marquardt algorithms are found to be the most efficient and reliable means to be used for this study [Mathworks 1994]. Table 3.1 shows a comparison of the three most popular supervised algorithms. These numbers are based on *MATLAB* Version 4.2 being run on a Macintosh Quadra 950.

TABLE 3.1 Comparison of Optimization Algorithms

Function	Technique	Time (sec)	Flops*
TRAINBP	Back-propagation	259.1	2.16E+0.8
TRAINBPX	Fast Back-prop	42.4	2.97E+0.7
TRAINLM	Levenberg-Marquardt	3.3	315769

* Floating Point Operations

Therefore, Levenberg-Marquardt Algorithm is the choice of design.

3.3.4 Design of the Appropriate Neural Network Topology

The design of the appropriate neural network topology involves the following steps [Dayhoff 1990]:

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

Function approximation is one of the most powerful 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 affects the outputs, tangent-sigmoid, logarithmic-sigmoid and pure linear neuron (transfer) functions were selected for further investigation. Moreover, the amount of neurons each layer depends on the complexity of the target function. 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 must be conducted in order to find the most appropriate architecture for this project. Note that there are an infinite amount of combinations between the number of neurons and layers. For this reason, some typical architectures are considered as candidates for this project. Training inputs for this part of the experiment are 805 data points selected from the flight manual cruise section and then tested and generalized with another 805 data points selected from the same source. Results are evaluated based on the number of floating point operations (flops) and output errors.

3.4 Implementation and Generalization of the Neural Network

Table 3.2 shows a list of all the candidate neural network topologies selected and their corresponding computational performance.

TABLE 3.2 Results of Topology Study

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

Based on the above information, the two best candidate topologies are NN 2-6 and NN 3-8. The corresponding mean errors are 0.606% and 0.604%. 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.

3.4 Implementation and Generalization of the Neural Network

The neural network employed in the fuel burn evaluation model is based on non-linear transfer functions because of the non-linearity of the input data. The process to estimate all weights associated with the neural network uses a non-linear programming technique. It involves real decision variables but the objective and constraint functions are not necessarily linear. The objective of this 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 biases to appropriately reconfigure the output at each iteration; the error between the output and the target becomes smaller until a minimized error goal is achieved. The goal of the model is to achieve an accuracy between the inputs and outputs of a sum squared error of 0.015, typically. The general conditions behind a neural network model are as follows:

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

For the model to be useful the above statements must be achieved.

3.4.1 Description of the Neural Network Aided Fuel Consumption Model

Fuel burn evaluation of a particular aircraft for each mission is divided into 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 was obtained from the aircraft flight manual and was used as inputs to train the neural network. A back-propagation neural network with Levenberg-Marquardt approximation model was 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 [Hagan 1996]. Figure 3.2 displays the architecture of the three layer fuel burn approximation network.

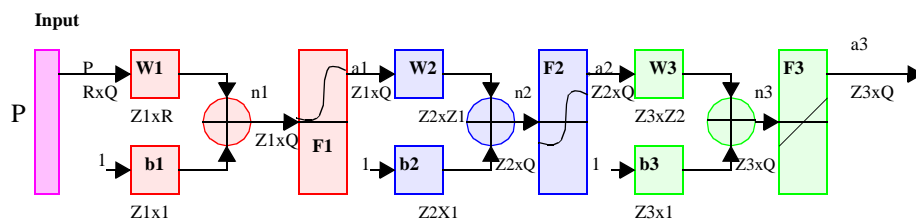


Figure 3.2 General Three Layered Neural Network.

Where:

3.4 Implementation and Generalization of the Neural Network

\mathbf{P} is the input vectors

Z_i is the number of neurons in i^{th} layer, for $i=1,2,3$. For example, $Z_1=8$, $Z_2=8$ and $Z_3=1$.

F_1 is a logarithmic sigmoid transfer function

F_2 is a tangential sigmoid transfer function

and F_3 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 follow linear relationships in these regimes. Once the slope and the y-intersection of the line is found (see Figure 3.3), 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.

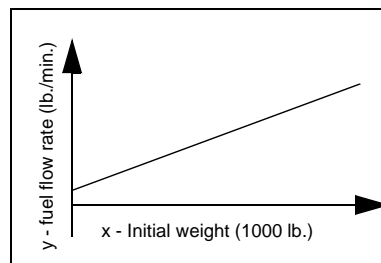


Figure 3.3 Fuel Burn for Takeoff and Climb-Out.

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

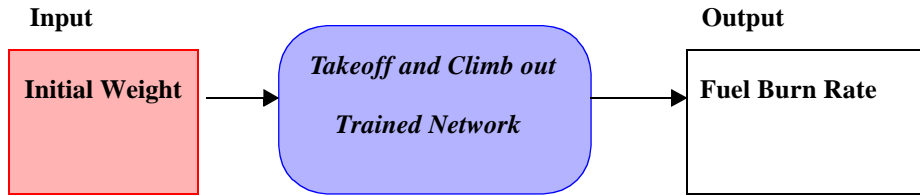


Figure 3.4 Fuel Estimation Procedure for Takeoff and Climb-Out.

is illustrated in Figure 3.4.

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.

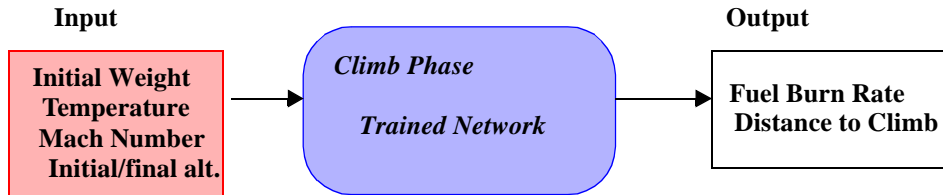


Figure 3.5 Fuel Estimation Procedure for Climb.

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

Since

$$FuelBurn(lb) = Range(nm)/(SAR)$$

the fuel burn can be found using the required Mach number for a given time interval.

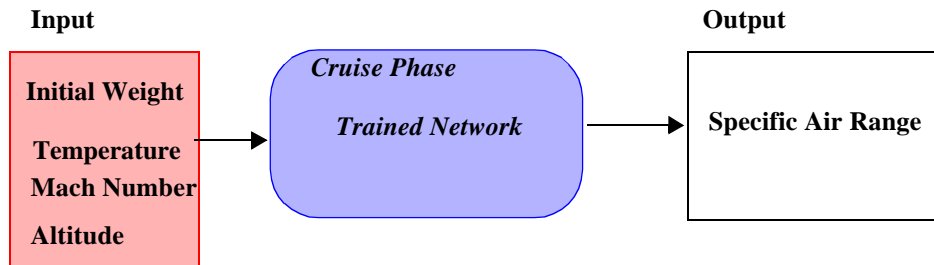


Figure 3.6 Fuel Estimation Procedure for Cruise

3.4 Implementation and Generalization of the Neural Network

The fuel burn for descent and landing is similar to the takeoff and climb-out phase, therefore the details will not be repeated here.

The steps outlined above show that 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.

3.4.2 Summary

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.

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	864 (Fuel) 864(Distance) Total 1728	850(Fuel) 850(Distance) Total 1700	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	288(Fuel) 288(Distance) Total 576	140 (Fuel) 140 (Distance) Total 280	a) Initial Weight (1000 lb) b) ISA Cond c) Mach Number d) Target Altitude (1000 ft.)	a) Fuel Burn (lb) b) Distance to Descent (nm)

The number of training and testing points were selected based on the complexity of the performance charts provided in the aircraft flight manual. The climb and cruise phases of flight contain performance curves which

are discontinuous and non-linear, therefore it is necessary to include all typical performance points for training and testing the neural network. For descent, the fuel consumption is linearly related to aircraft weight and flight Mach number, so a comparatively smaller amount of data points are required.

3.5 Results Justification

In order to justify the results, a confidence study is conducted using statistics. The following are the statistical measures employed in this research.

3.5.1 The Normal (Gaussian) Distribution

The normal distribution provides a useful statistical model that can be utilized to study the results and interpretation of experimental data. Assumptions required to derive the normal distribution are as follows [Drew 1993]:

1. The final error in any observation is the result of combining a large number of very small errors, all of equal magnitude.
2. These small errors have equal probability of being positive and negative.

The density function of the normal distribution is shown below

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\bar{x}}{\sigma}\right)^2}, -\infty \leq x \leq \infty$$

The mean \bar{x} and standard deviation σ are used as parameters of the distribution. The density function is a bell shaped curve that is symmetric around \bar{x} .

3.5 Results Justification

The cumulative distribution function for a normal distributed random variable can be written as

$$P(k_0 < x < k) = \int_{k_0}^k \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\bar{x}}{\sigma}\right)^2} dx$$

Hence to find the probability that x is between k_0 and k , the above formula can be utilized.

Making a substitution

$$\eta = \frac{x-\bar{x}}{\sigma} \text{ and } d\eta = \frac{dx}{\sigma}$$

the following can be obtained

$$P(k_0 < x < k) = \frac{1}{\sigma\sqrt{2\pi}} \int_{\frac{k_0-\bar{x}}{\sigma}}^{\frac{k-\bar{x}}{\sigma}} e^{-\frac{\eta^2}{2}} d\eta \sigma$$

which can also be written as

$$P(k_0 < x < k) = I\left(\frac{k-\bar{x}}{\sigma}\right) - I\left(\frac{k_0-\bar{x}}{\sigma}\right)$$

where

$$I(\eta) = \frac{1}{\sqrt{2\pi}} \int_0^{\eta} e^{-\frac{\eta^2}{2}} d\eta$$

The above integral is usually listed in table form.

3.5.2 Mean and Standard Deviation

Once the type of model is hypothesized, a good next step is to evaluate the actual value of the parameters involved.

For a set of N discrete samples of a quantity x , the mean \bar{x} and the standard deviation σ are given by the relations

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i \quad \text{and} \quad \sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$$

Note that standard deviation is called the variance. In addition, the standard deviation is calculated dividing by $N-1$ rather than N . Technically, this is because only $N-1$ of the samples are independent of the mean [Devenport 1995].

3.5.3 Histogram

There are many other ways of presenting data in a statistical way. Perhaps the most revealing is a histogram. To construct a histogram the range of the quantity being measured is divided up into a number of equal intervals. For discrete data, by adding up the number of samples falling into each interval, the result is a graph showing how the data are distributed. The mean, being a measure of the average, lies near the centre of the histogram. The standard deviation, being a measure of the spread, is usually about one quarter to one sixth of the width of the histogram [Devenport 1995].

3.5.4 t-test

t-test is used in comparing two means. The purpose of this comparison is to tell whether the means of two sets of data are significantly different. The probability level corresponds to calculated t value is the probability that the two means are truly different. In testing significance of means, the error probability α must be declared (typically 5 or 1%). This testing procedure is to justify a hypothesis that two means are the same. This hypothesis is known as "null hypothesis". The following example is given to illustrate the testing procedure.

Two samples of size $N1$ and $N2$, respectively, are independent of one another and assumed to come from normally distributed population. In addition, the deviations of their variance $S1$ and $S2$ are assumed to be random. The null hypothesis, that their mean values $A1$ and $A2$ differ only randomly with an error probability α , is

accepted if the calculated value t_c for the test variable $t = \frac{|A1 - A2|}{Sd} \sqrt{\frac{N1N2}{N1 + N2}}$ with $Sd = \frac{S1(N1 - 1) + S2(N2 - 1)}{N1 + N2 - 2}$ is

less the tabulated value t_t for α and $f = N1 + N2 - 2$ degrees of freedom. Note that the tabulated t value can be found in most statistics text book.

3.5.5 Implementation of Statistical Analysis

For the purpose of this research, the neural networks used to predict fuel burn were compared to the actual fuel burn for each performance point. Each pair of actual and predicted fuel burn has an error value $Error_i$ given by

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

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

The above error value will be recorded. For each flight phase there will be a corresponding mean and variance of all the errors. According to the previous discussion, the errors are assumed to be distributed normally. The probability that an error falls in between twice the value of standard deviation, i.e. $P(\bar{x} - 2\sigma < x < \bar{x} + 2\sigma)$, would be 95% [Devenport 1995]. This probability is independent from the value of the mean and variance. Moreover, t-test will be used to compare means of both actual and predicted fuel burn. Therefore, the error band can be predicted and the results can be justified.

3.6 Implementation of the Outputs to the Actual Flight Trajectory

The neural network trained weight matrix and bias vector was implemented through a simulation program. This simulation program takes a desirable aircraft trajectory as inputs and use the corresponding weights and biases to estimate the fuel consumption of the aircraft. To complete a mission, the aforementioned six flight phases should be considered.

The simulation program is basically a feed-forward one step process. For a single neuron, suppose each layer has its own trained weight matrix \mathbf{W} and its own bias vector \mathbf{b} , along with the transfer function f and input vector \mathbf{p} , output a can be expressed as [Hagan 1996]:

$$a = f(Wp + b)$$

A three layered network with eight neurons in the first two layers and one neuron in the third layer was 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.

Figure 3.7 shows a three layer network which will be utilized to calculate the aircraft fuel consumption. 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 \mathbf{W}^2 . The input to layer two is \mathbf{a}^1 , and the output is \mathbf{a}^2 . Therefore, the three layer network in this example can be written as the following expression.

$$fuelburn = a^3 = f3(W^3f2(W^2f1(W^1p + b^1) + b^2) + b^3)$$

where

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

Hence for the first neuron in the first layer the output a_1^1 can be expressed as:

$$a_1^1 = f1 \left(\begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & w_{1,2}^1 & w_{1,2}^1 \end{bmatrix} \begin{bmatrix} altitude \\ Mach \\ Weight \\ ISA \end{bmatrix} + b_1^1 \right)$$

Since

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

therefore

$$a_1^1 = \frac{1}{1 + e^{- \left(\begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & w_{1,2}^1 & w_{1,2}^1 \end{bmatrix} \begin{bmatrix} altitude \\ Mach \\ Weight \\ ISA \end{bmatrix} + b_1^1 \right)}}$$

Calculation of fuel burn in all other flight phases will be based on the same principle and therefore will not be repeated here.

3.6 Implementation of the Outputs to the Actual Flight Trajectory

Chapter 4 provides a description of all the computer programs created for this research project which will give the reader an insight into some of the technical details involved.

This chapter describes in detail all the computer programs constructed for this research project. The description of programs is intended to present a detailed description of the major components of this project. The following steps summarize the procedures adopted in the implementation of the fuel consumption neural net:

1. Training of neural net 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 an actual flight trajectory

4.1 Training of Weight Matrix For Different Flight Phases

The program developed in this section will be utilized 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, the following decisions regarding the neural network were also required as inputs:

1. The number of inputs.
2. The value for the learning coefficient.
3. The number of processing elements in the hidden and output layers.
4. The number of cycles for each run.

MATLAB was used as the modelling language to implement the neural network. In this *MATLAB* functions in the neural net toolbox. For example, **initff** initializes a feed-forward model, **trainlm** trains a network using Levenberg-Marquardt Algorithm. Other transfer functions will be employed in this section as well.

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 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. Refer to Figure 4.1 for an illustration of the input vector **P**.

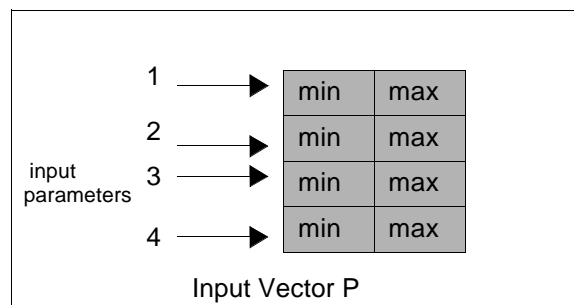


Figure 4.1: Example of Input Vector

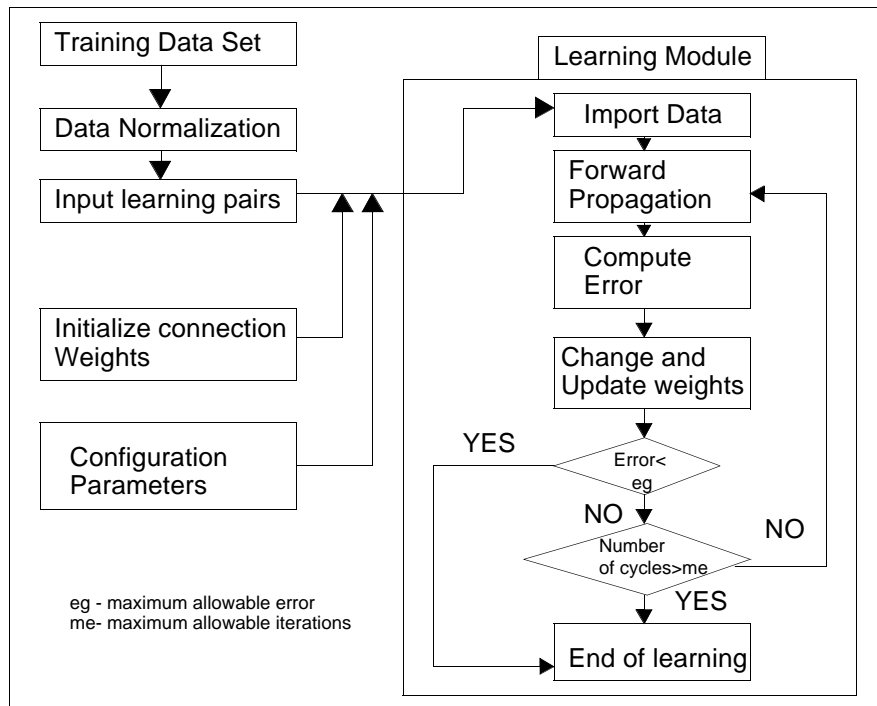


Figure 4.2: Flow Chart Illustrating Neural Network Training Process

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 developed is shown in Appendix A. The structure of the program is illustrated in Figure 4.2.

Since different flight phases require different inputs, for each flight phase there will be an individual training routine to produce the corresponding weight matrix. Table 4.1 presents a summary of the input and output parameters for all flight phases.

TABLE 4.1 Summary of Input and Output Parameters

Flight Phases	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)

TABLE 4.1 Summary of Input and Output Parameters

Flight Phases	Input parameters	Output parameters
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

4.2 Testing Preliminary Results Using Statistics

Preliminary results for the climb, cruise and descent segments of flight were also used to test the reliability of the neural network to estimate aircraft fuel consumption. 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 are processed by a feed-forward propagation (simulation) routine which gives an output. Outputs are then compared with the actual values obtained from the flight manual for the corresponding performance point. Statistical methods were used to verify the accuracy of the model. For the climb segment, a total of 1700 performance points were tested. For the cruise segment, a total of 805 performance points were used in the generalization of the neural network. Finally, for the descent segment, a total of 280 performance points were tested to bias the generalization. All the performance points were selected in a random manner. A 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 is contained in Appendix A.

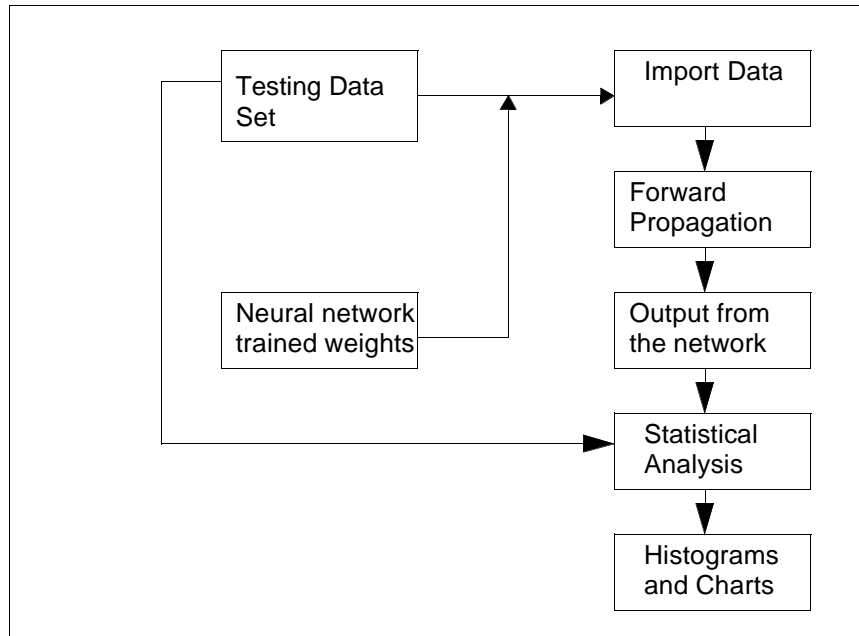


Figure 4.3: Flow Chart Illustrating Neural Network Testing Process

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 into 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 a number of way points. The velocity and altitude schedules are described at each way point. Way points are imaginary points in the airspace demarking the three dimensional trajectory of a vehicle. An example flight trajectory is shown in Figure 4.4.

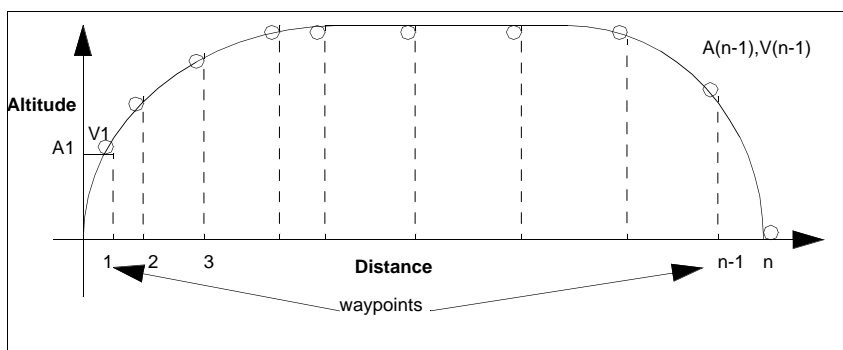


Figure 4.4: Illustration of Way Points on Trajectory

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 a neural network is not required. Based on the initial weight and taxi time, the fuel burned can be calculated.
2. The takeoff and climb-out fuel consumptions are estimated with a neural network. This flight phase consists of only one way point (climb-out ends at the point where the aircraft reaches 1500 ft above the elevation of the departing runway threshold). 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 burn value.
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 F100, data for climb are given only for the following cases:

4.3 Implementation of Neural Network to Actual Flight Trajectory

- 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:

First, calculate fuel consumption for climb from $A1$ to $A2$ with initial weight $W1$ using Mach numbers $M1$ and $M2$. Let this result be F_{W1} . Using the weighted average for Mach number, the fuel consumption of an aircraft with initial weight $W1$ climbing from $A1$ to $A2$ (F_{W1}) can be written as:

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

Next, 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:

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

Finally, apply the same idea 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:

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

The climb distance calculation is analogous to the climb fuel calculation, replacing F_{W1} and F_{W2} by D_{W1} and D_{W2} , where D represents the estimated distance. 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 inputted trajectory requires. The program will be able to 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 inputted 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.

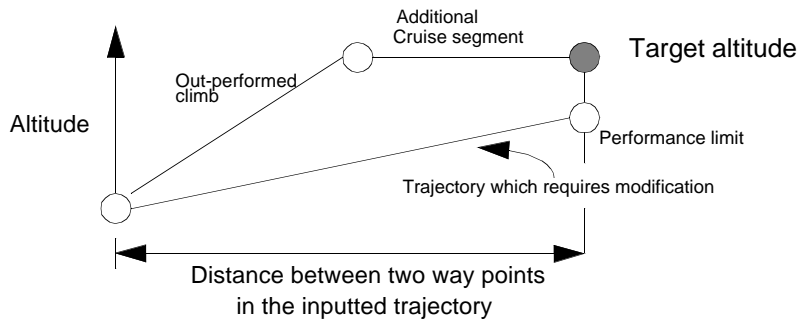


Figure 4.5: Possible Climb Phase Scenarios

- The techniques used to calculate fuel consumption for the cruise segment are very similar to the ones used in the climb phase, with the only difference being that data is given for altitude instead of Mach number. Applying the same idea previously developed, an example of fuel burn calculation for an aircraft cruising at altitude A , with cruise Mach number M and weights W follows.

Suppose $W1 < W \leq W2$ and $A1 < A < A2$

4.3 Implementation of Neural Network to Actual Flight Trajectory

where $W1$, $W2$, $A1$ and $A2$ are performance parameters which can be used to determine the specific air range (SAR) without interpolation. As previously stated, the first step is to calculate the specific air range for both $A1$ and $A2$, such that SAR_{W1} and SAR_{W2} can be written as:

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

and

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

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

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

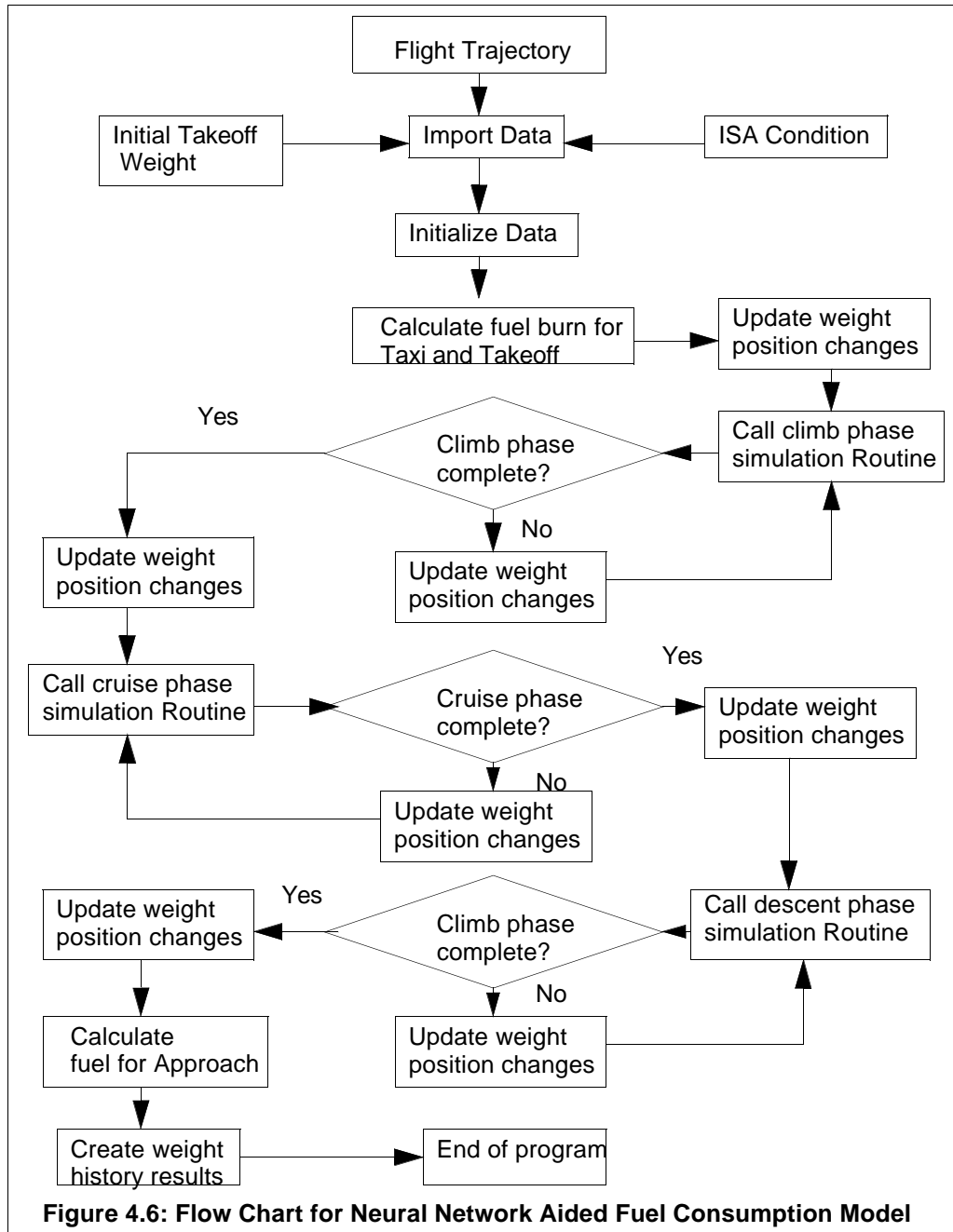
SAR can be translated into fuel burn using the following equation:

$$F_W = \frac{distance}{SAR}$$

where *distance* refers to the displacement of an aircraft between two nodes.

5. Since the computation procedure of fuel consumption for the descent phase is the same as shown in the climb phase, it will not be repeated here.

A complete flow diagram summarizing the computational procedure of this section is shown in Figure 4.6.



Discussion of Results

This chapter presents the results generated by the neural network aided fuel consumption model. The results of this research will contain graphical representations and statistical analysis of the data. The Fokker F100, 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. This chapter includes training results, testing results and neural network aided fuel consumption results.

5.1 Training Results

The purpose of training the neural network is to create sets of weight matrices and bias vectors. Training data sets are obtained by digitizing the flight manual of the test aircraft. The sizes of the various training data sets used in the neural network learning are shown in Table 5.1:

TABLE 5.1 Training Data Sets

Flight Phase	Number of Training Points
Takeoff and Climb-Out	8
Climb to Cruise Altitude	864 (Fuel) 864(Distance) Total 1728
Cruise	805
Descent	288(Fuel) 288(Distance) Total 576

Figures 5.1 to 5.3 are plots showing the distributions of training data used for the cruise and climb phases of flight.

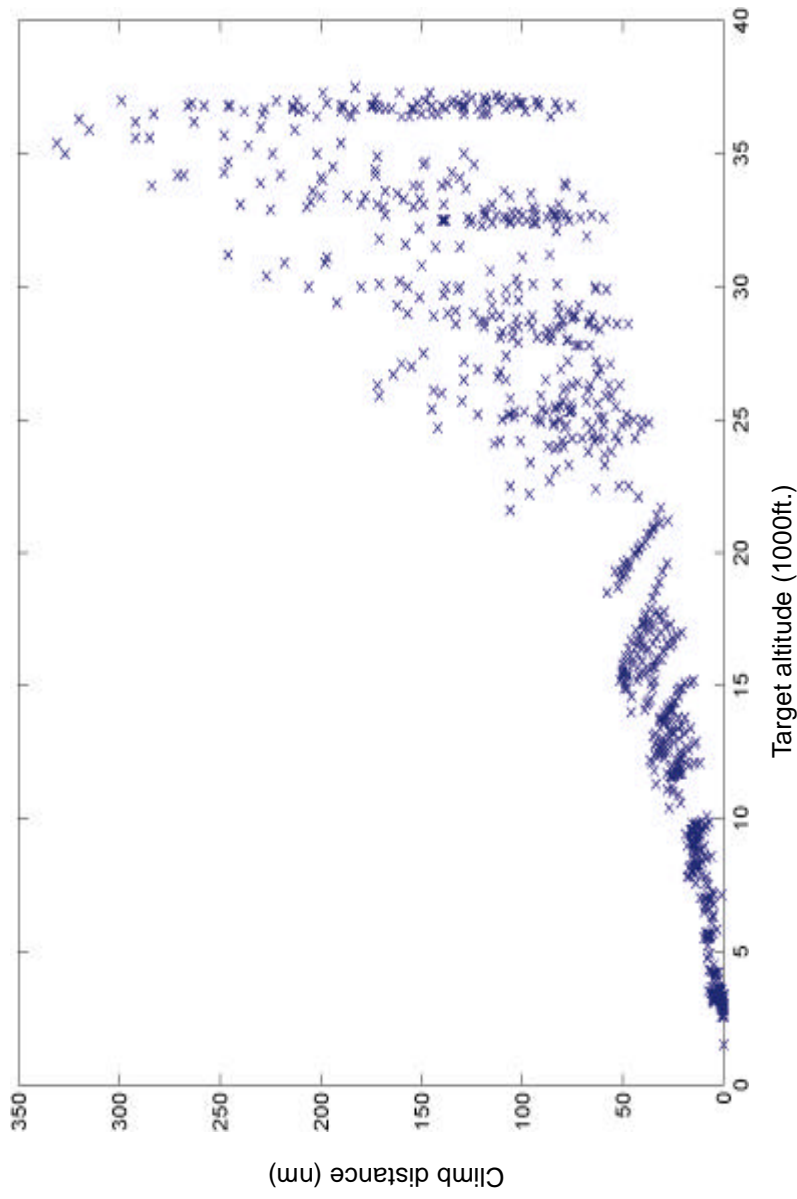


Figure 5.1 Training Climb Distance.

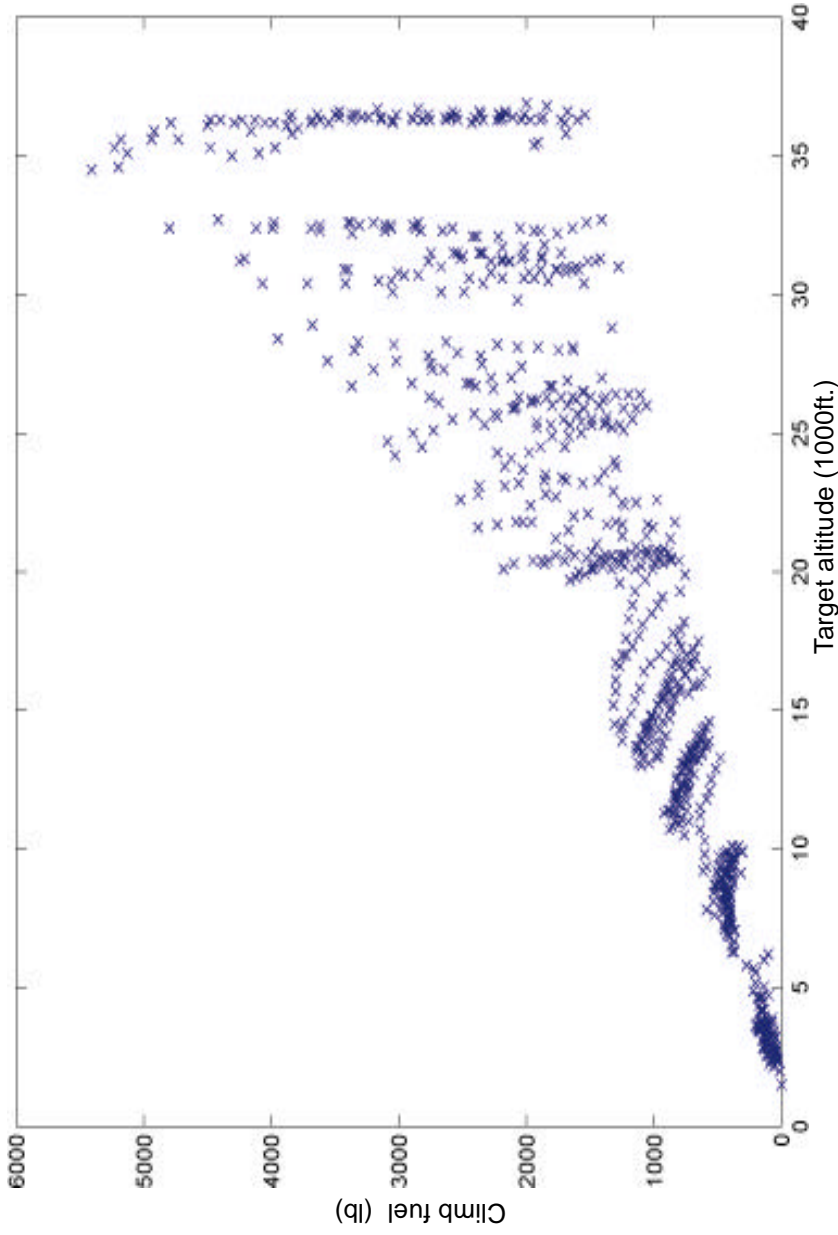


Figure 5.2 Training Climb Fuel.

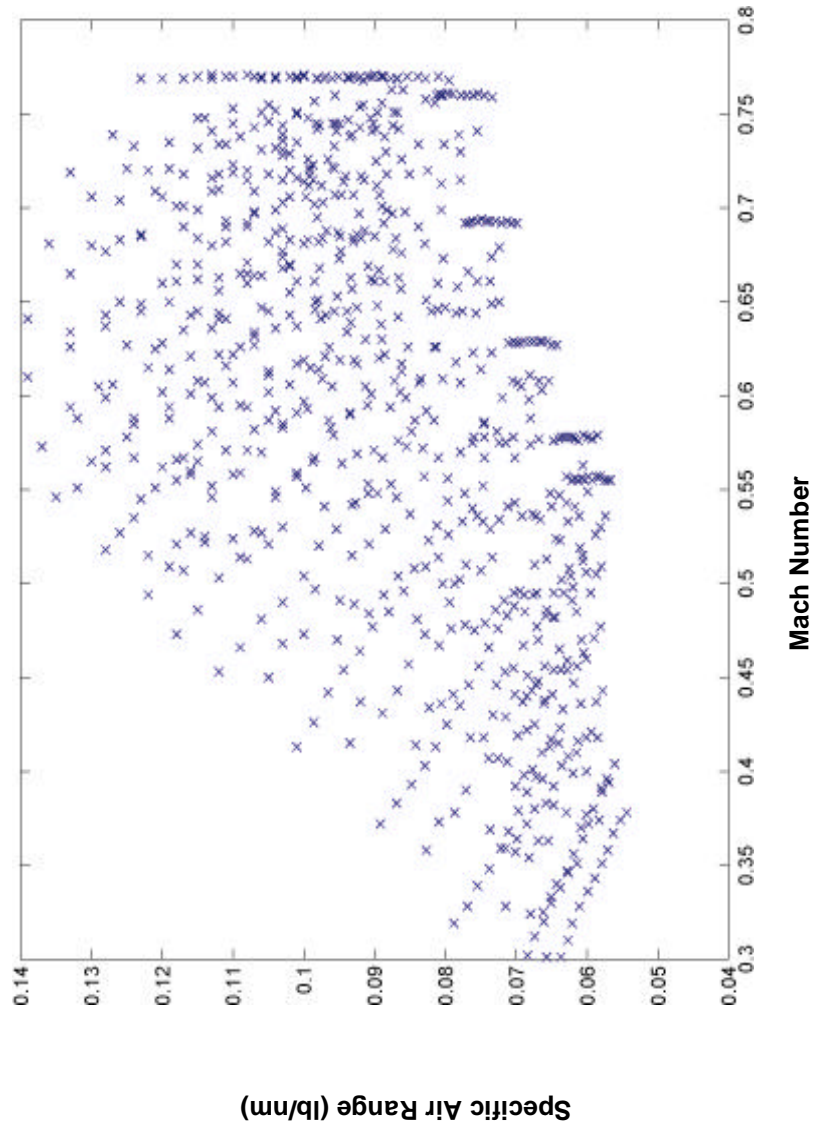


Figure 5.3 Training Cruise Fuel.

As shown in these figures, training data sets have covered all the typical performance points in the climb and cruise phases of the mission. For the climb phase, fuel consumption and distance estimation are trained for target altitudes ranging from 1500 ft. to 40000 ft. For the cruise phase, specific air range (the distances covered for every pound of fuel consumed) is trained for cruise Mach numbers ranging from 0.3 to 0.77. 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 5.1 through 5.3.

Sample weight matrices and bias vectors are found for the Fokker F100 in Appendix B. It is worth noting that once these matrices and vectors are found and stored, they become a data base which can be utilized in many other simulation programs, and do not depend on *MATLAB*.

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. Perhaps this is the most important part of the research. As stated in Section 5.1, whether the network is reliable depends on how well the trained network can generalize the inputs. The test results were evaluated based on the mean error and the standard deviation of error introduced in Chapter 3. The results are examined and presented in this section.

Table 5.2 is a list of the data sets used in the generalization step for various flight phases.

TABLE 5.2 Testing Data Sets

Flight Phase	Number of testing points
Takeoff and Climbout	N/A
Climb to Cruise Altitude	850(Fuel) 850(Distance) Total 1700

5.2 Testing Results

TABLE 5.2 Testing Data Sets

Flight Phase	Number of testing points
Cruise	805
Descent	140 (Fuel) 140 (Distance) Total 280

Figures 5.4 to 5.6 are plots illustrating the test data set.

Figure 5.4 shows a plot of actual climb fuel versus target 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 5.5).

Note that the climb fuel and climb distance are parameters taken from the flight manual which are then compared with the estimated values from the neural network. Figures 5.4 and 5.5 are intended to give the reader evidence that the altitudes and Mach numbers tested in this research are sufficient to represent all missions flown by this aircraft.

Figure 5.6 is a plot of cruise altitude vs. cruise Mach number. This figure is also known as the “flight envelope”. Performance limits such as service ceiling and critical Mach number can be found from this figure [Anderson 1989]. Therefore, it is important to refer to the flight envelope when designing a trajectory for an aircraft to make sure that the altitude and Mach schedule does not violate the aircraft performance limits. The results of comparing estimated fuel burn to actual fuel burn for various flight phases are shown in Figures 5.7 through 5.11.

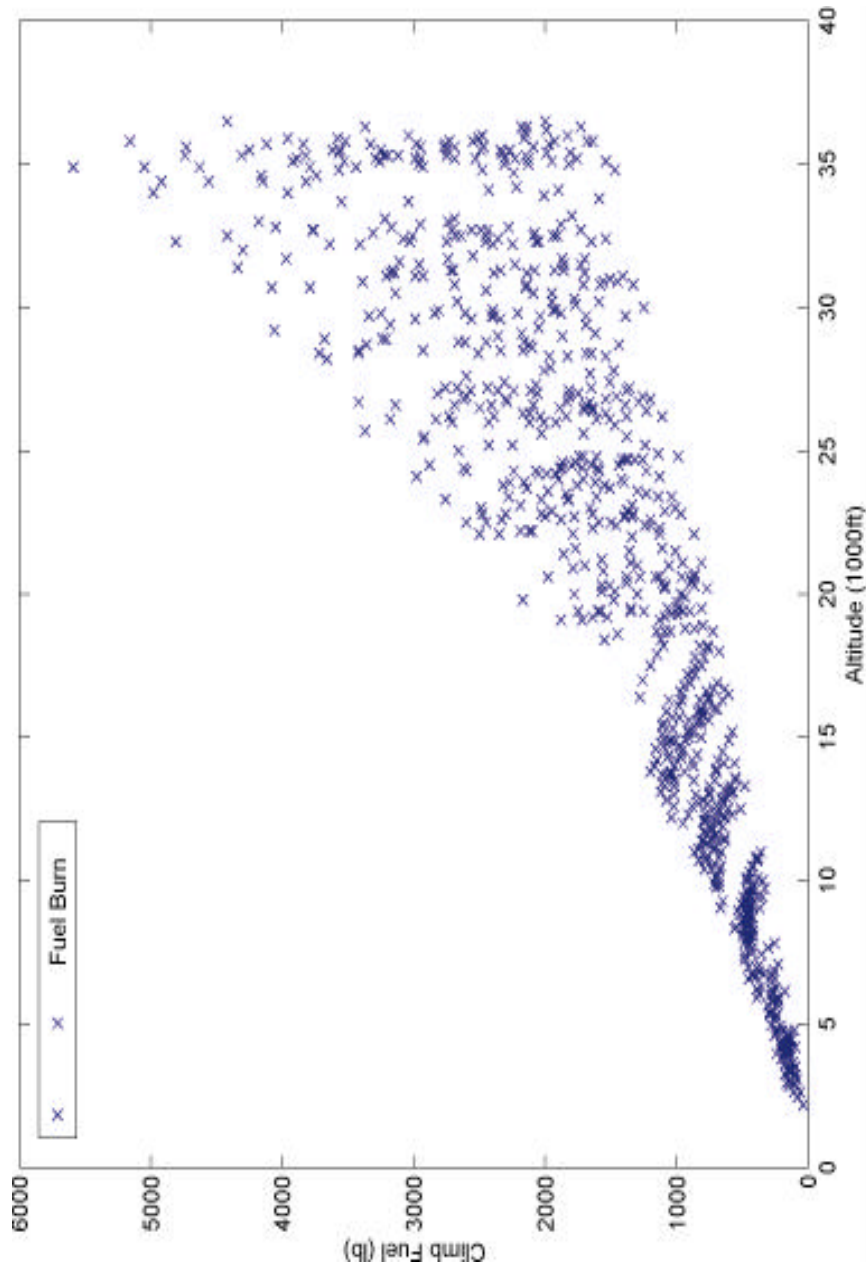


Figure 5.4 Climb Fuel Testing.

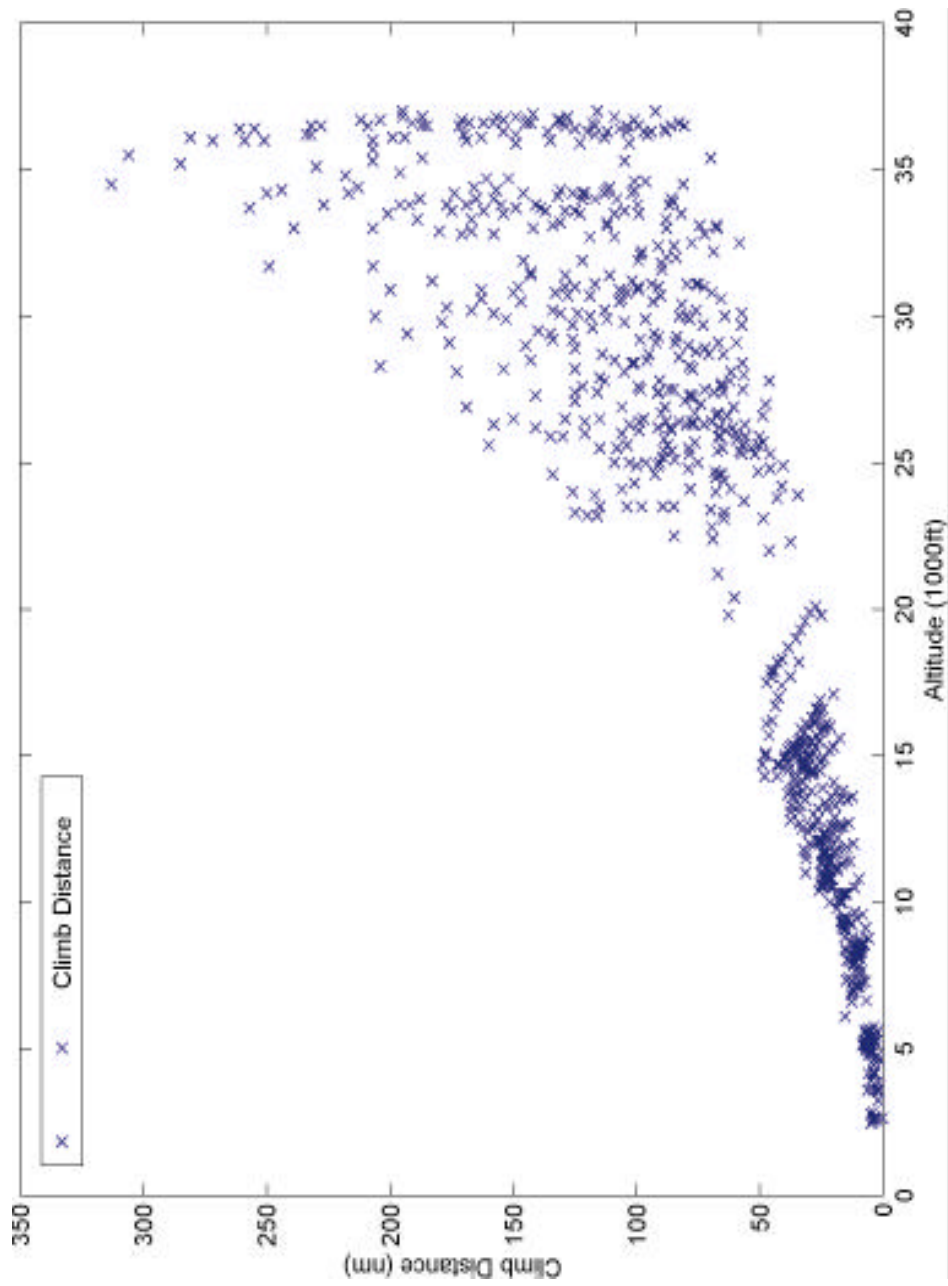


Figure 5.5 Testing Climb Distance.

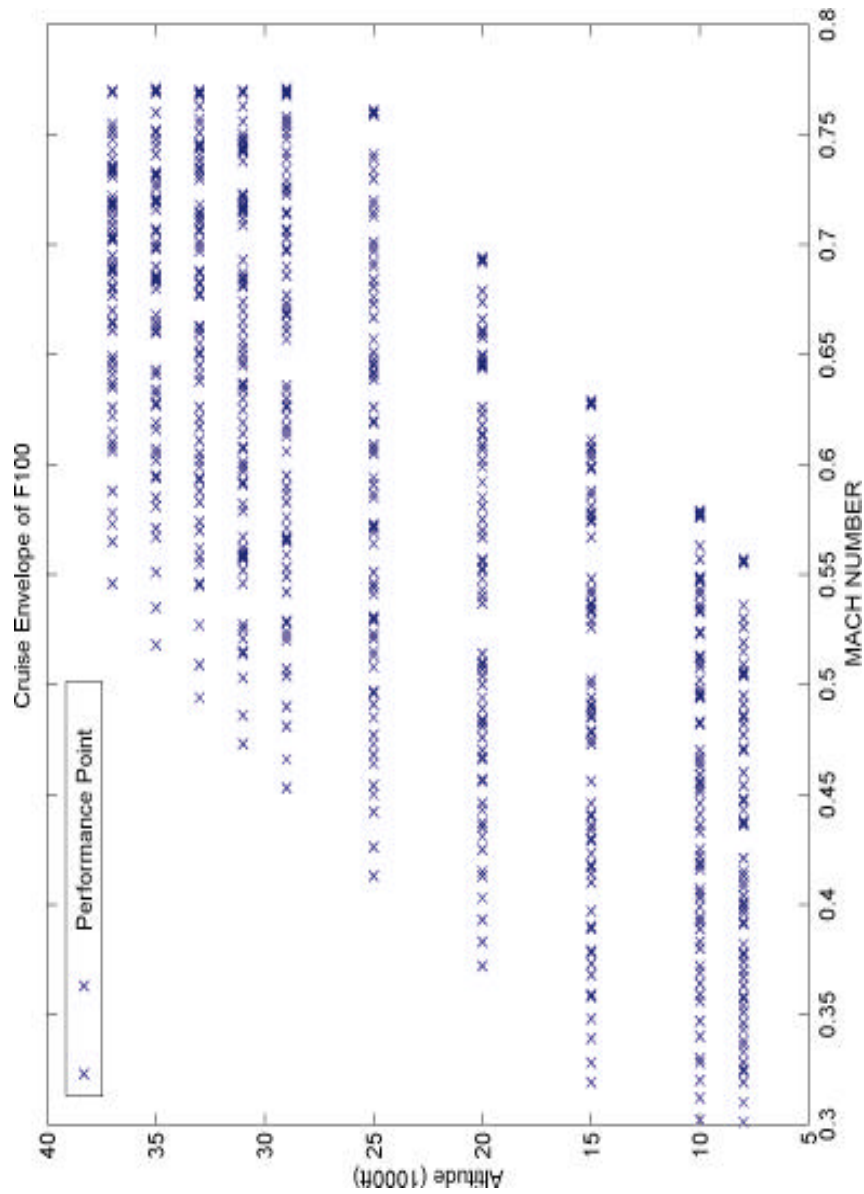


Figure 5.6 Testing Flight Envelope.

5.2 Testing Results

Figure 5.7 is a plot of estimated fuel burn versus actual fuel burn for various climb profiles. Figure 5.8 presents a scattered plot of estimated climb distance versus actual climb distance. The abscissa of these figures represents the data point number. The ordinate represents the actual values of the outputs from both neural network and data acquired from the flight manual. The left hand side of these two figures illustrates the low altitude regime of the climb phase while the right hand side of the figure represents the high altitude regime of the same flight phase. It can be seen that a trained neural network output is able to map the actual fuel or distance required to climb very closely. The error line at the bottom of these figure shows the magnitude of error produced when compared to the actual performance. Taking the difference between each actual and estimated value, the average relative error is calculated to be 3.11% for fuel estimation and 4.91% for distance estimation. The sources of error here are probably due to the random error in the neural network calibration.

Figure 5.9 shows a plot of estimated specific air range versus actual specific air range. Again, the abscissa of this figure represents the number of data points and the ordinate represents the specific air range value. From this figure, it can be seen that the neural network outputs are accurately predicting the specific air range of the cruise phase. The corresponding average relative error is approximately 0.604%, which is considered very accurate for most simulation and modeling requirements.

Figure 5.10 depicts a plot of estimated descent versus actual descent distances. Figure 5.11 is a plot of estimated descent fuel versus actual descent fuel. The abscissa of these figures represents the number of data points and the ordinate represents the actual performance values. Since the behavior of descent fuel consumption and descent distance is linear, i.e. descent fuel/distance is linearly related to the target altitude, the size of the training and testing data set is reduced accordingly. The corresponding average relative error for descent fuel and descent distance are 3.39% and 1.99%, respectively. One reason for showing these figures is to demonstrate that a trained neural network is capable of generalizing the learning inputs accurately.

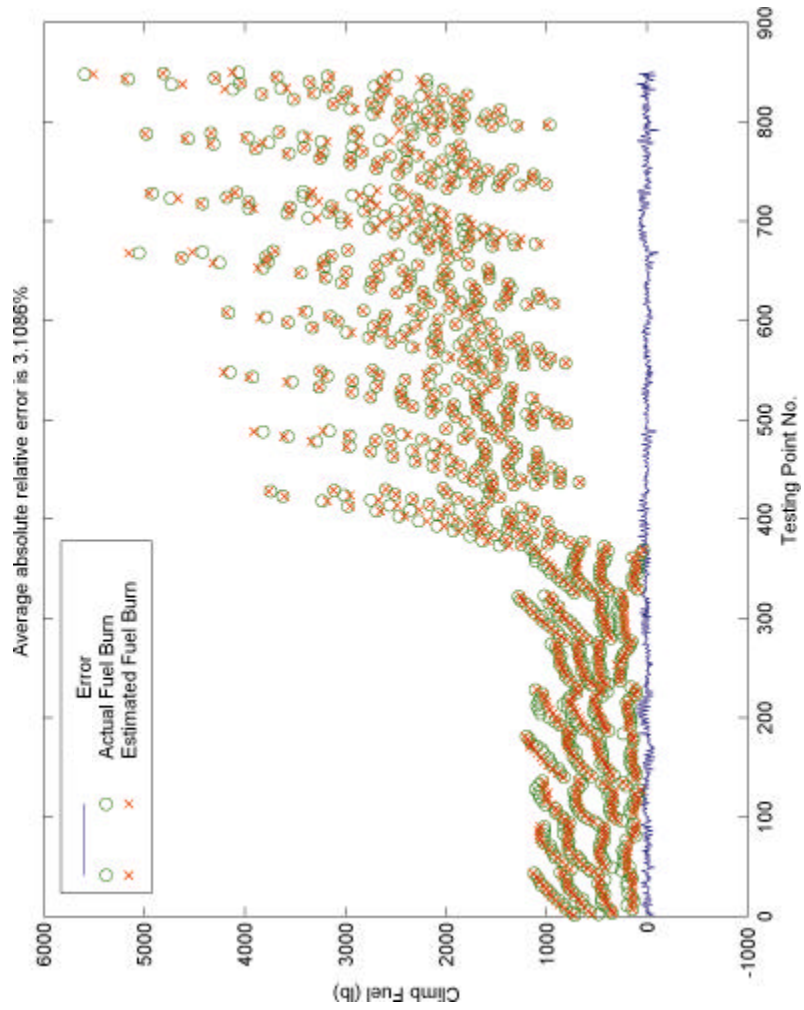


Figure 5.7 Climb Fuel Comparison.

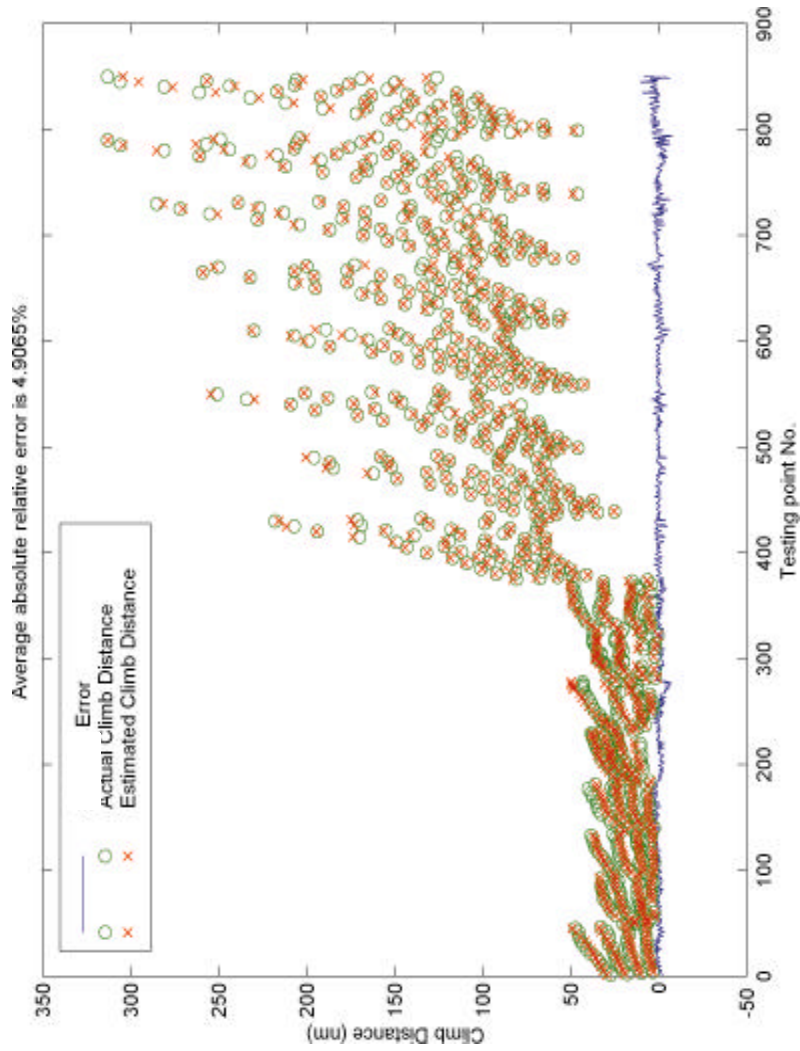


Figure 5.8 Climb Distance Comparison.

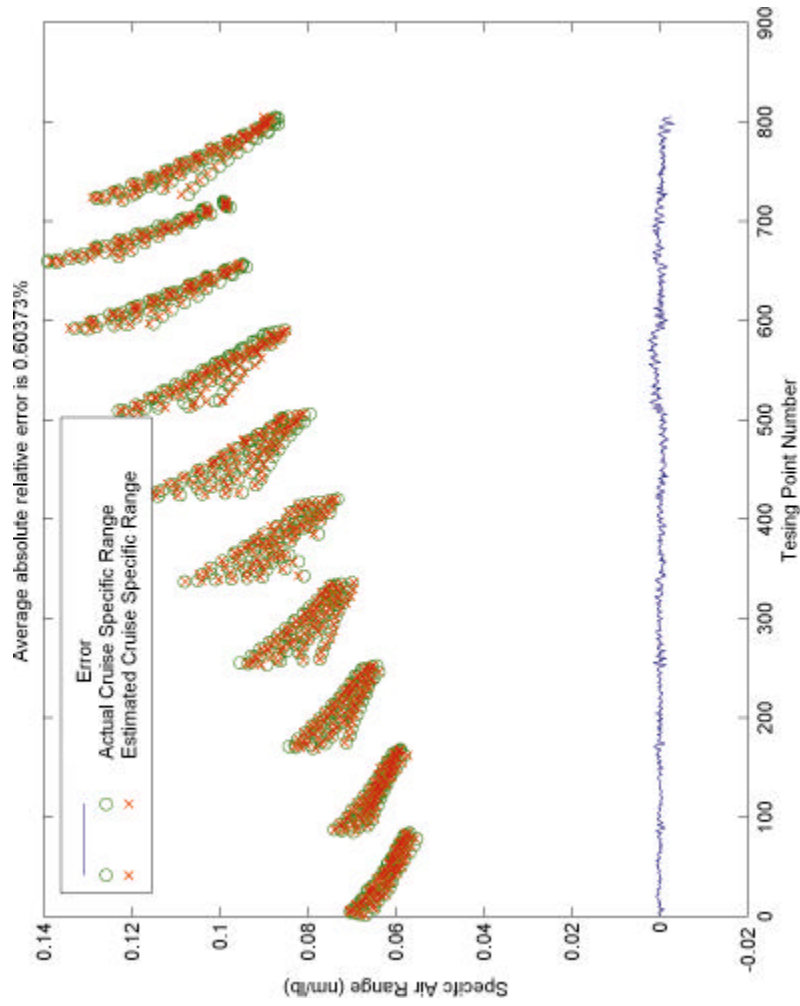


Figure 5.9 Specific Air Range Comparison.

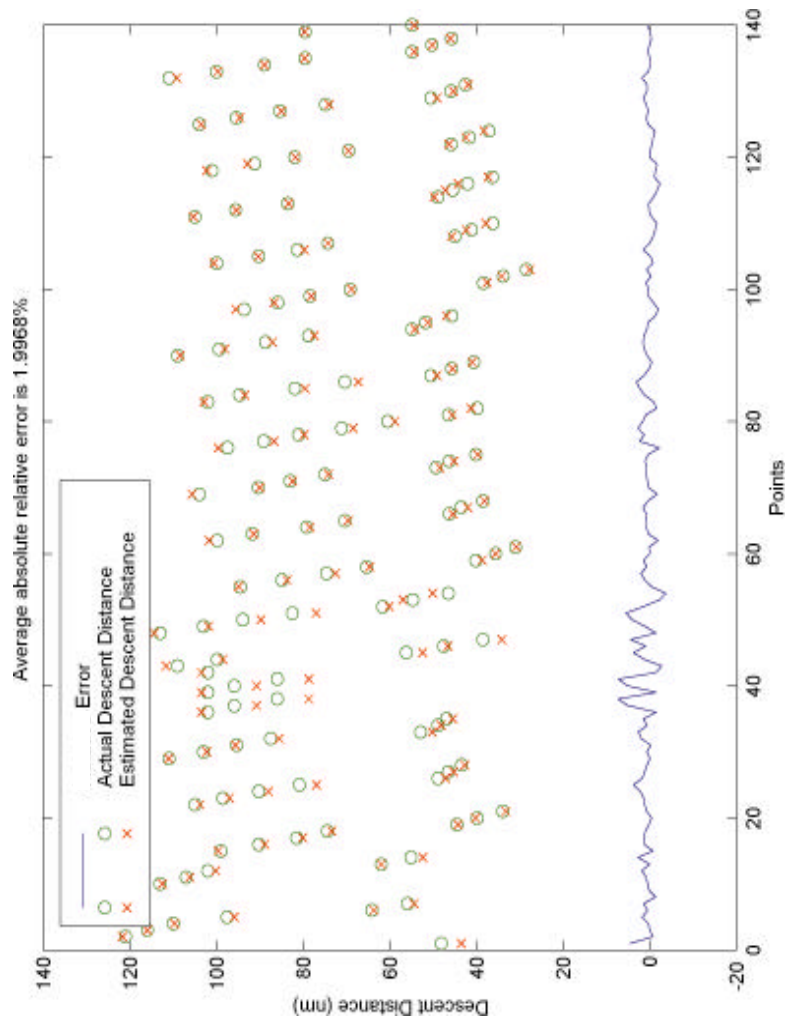


Figure 5.10 Descent Distance Comparison.

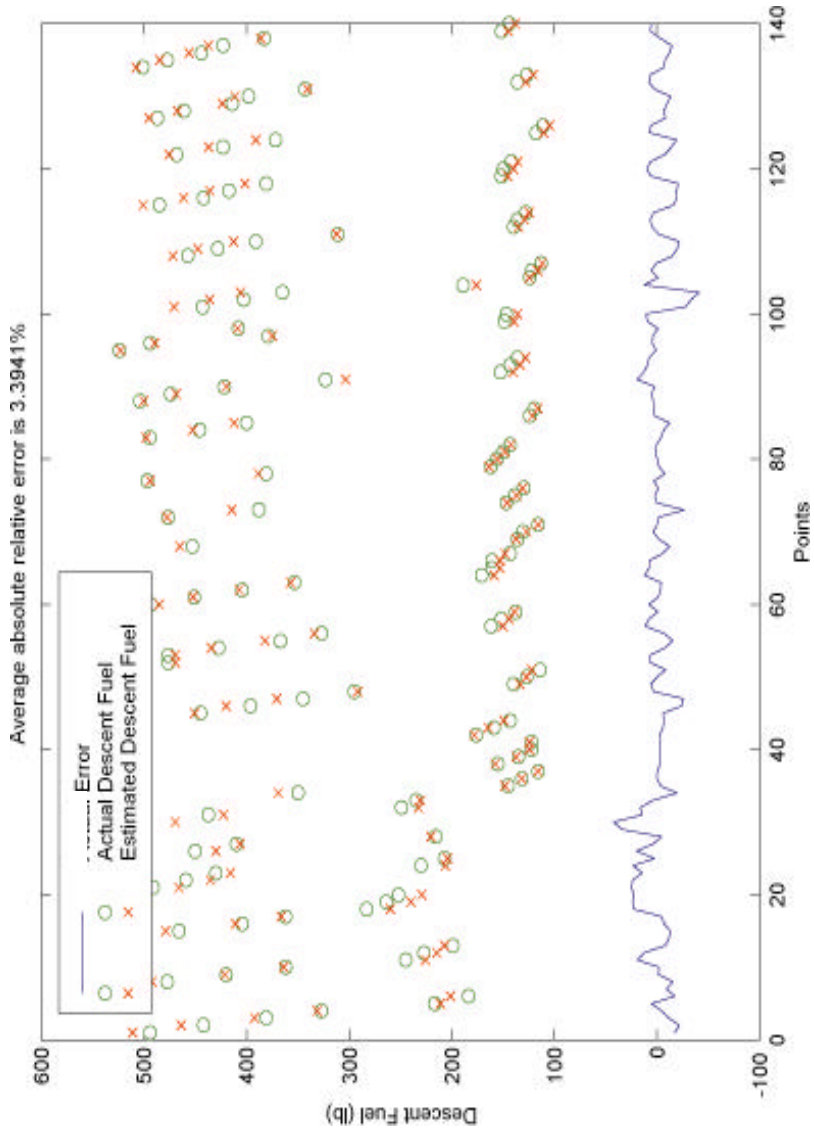


Figure 5.11 Descent Fuel Comparison.

5.2 Testing Results

A necessary step to investigate the degree of uncertainty of the neural network is to determine the distribution of errors. As stated in Chapter 3, a histogram is a compact and revealing medium to present the distribution of errors. Figure 5.12 contains a histogram illustrating the distribution of errors for the climb flight phase. Since histograms for other flight phases are very similar to each other, they are shown as attachments in Appendix C. For Figure 5.12 and Appendix C, the abscissa represents the relative error and the ordinate represents the frequency of observations. On top of each figure the corresponding mean relative error and standard deviation of relative error are displayed.

Figure 5.12 is a histogram that shows the error of climb fuel in percentage. The mean average error is calculated to be 1.03% and the standard deviation is 0.19%. As previously discussed, if the relative error is assumed to be normally distributed, the probability that an error generated from the neural network is within twice the value of the standard deviation is 95%. Therefore, for this segment of flight, 95% of the time the neural network induced error is within $1.03 \pm 0.38\%$. Table 5.3 shows the mean relative errors, standard deviations and t-test results for all the other flight phases.

Finally, it is important to note that, according to the t-test results discussed in Chapter 3, the means of the fuel consumption and distance data from the flight manual and estimated value belong to the same set.

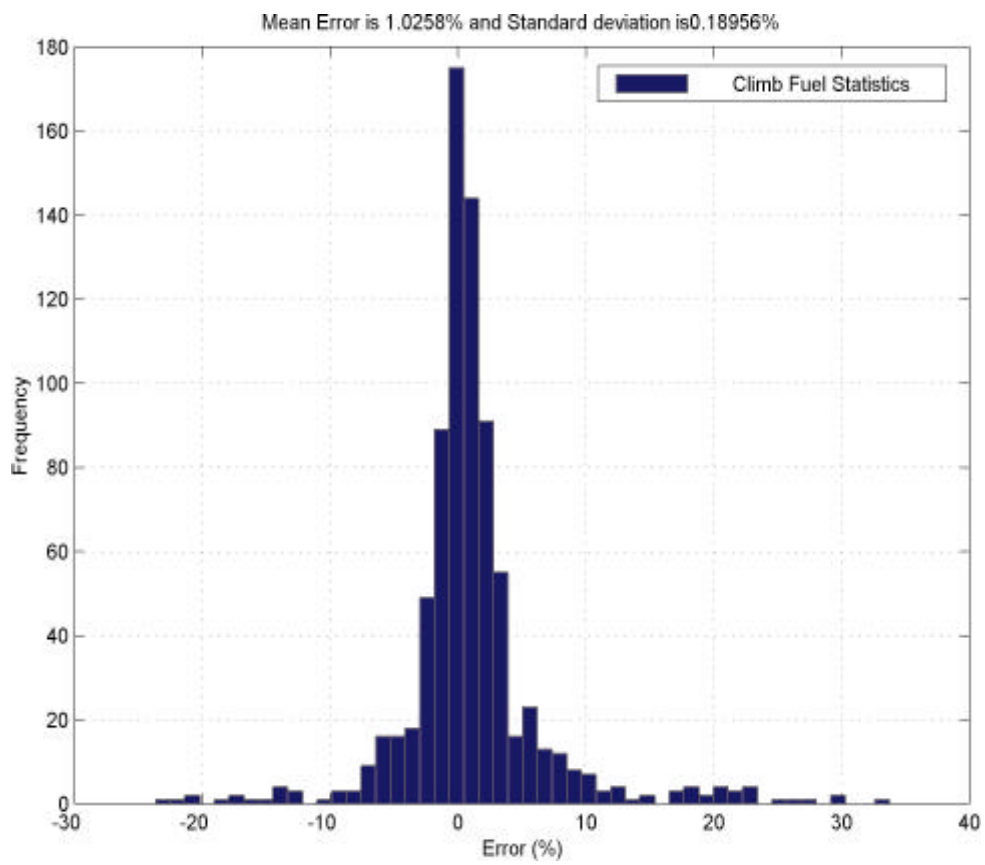


Figure 5.12 Histogram of Error for Climb Fuel Estimation.

5.3 Neural Network Aided Fuel Consumption Results

TABLE 5.3 Summary of Errors for All Flight Phases

Flight Phase	Mean Error(%)	Standard Deviation(%)	Null Hypothesis (t-test with 1% error probability)
Climb			
• Distance	-0.377	0.305	Accept
• Fuel	1.026	0.190	Accept
Cruise specific air range	-0.045	0.028	Accept
Descent			
• Distance	0.910	1.86	Accept
• Fuel	0.418	1.177	Accept

From Table 5.3, it can be seen that the neural network is a reliable means for estimating fuel consumption. The t-test shows that on comparing the mean of the actual and predicted fuel consumption, they are not significantly different, with 99% significance level or 1% error probability.

5.3 Neural Network Aided 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 of an aircraft for a complete mission.

Details of the program have been discussed in Chapter 4, therefore they will not be repeated here. A simulation program has been developed; the results obtained are shown in this section.

The input trajectory was developed under the instructions provided in the flight manual of the test aircraft, the Fokker F100. The total number of way points are 43. Table 5.4 contains a list of way points and the corresponding schedule. The initial gross weight is assumed to be 95000 lb. Figures 5.13 and 5.14 are plots of the intended flight trajectory in three-dimensions and two-dimensions, respectively. Figure 5.13 shows the speed and altitude as a function of distance from the origin in a three dimensional view. Figure 5.14 plots the altitude as a function of distance from the origin. The climb phase of flight is represented by the line on the left hand

side with a positive slope. The horizontal plateau in the middle represents the cruise segment and the line with negative slope represents the descent segment. It can be seen that the intended trajectory has the majority of the range spent on cruise. This trajectory is consistent with most commercial aircraft missions.

Figures 5.15 and 5.16 illustrate the weight changes of the test aircraft during the mission. Figure 5.15 shows aircraft weight as a function of the distance from the origin. Close evaluation of Figure 5.15 reveals that the slope of the curve changes as distance increases. The steepest segment of the curve represents the climbing phase. The linear segment of the curve represents the cruise segment because the test aircraft is cruising at a fixed altitude and speed. Figure 5.16 plots the aircraft weight as a function of altitude. From this figure, it can be seen that a substantial amount of fuel is consumed during the climb and cruise phases of flight.

Results obtained were compared with the performance summary of the flight manual. The results obtained are reasonable when compared to the information given in the flight manual. Since the information given in the flight manual does not include every single point for a given flight trajectory, at this point in time quantification of error for the entire mission is not possible. When compared to the similar trajectory given in the flight manual the differences between estimated fuel consumption and flight manual data are approximately 5%.

5.3 Neural Network Aided Fuel Consumption Results

Table 5.4 Flight Trajectory

Waypoints	Distance From Origin(nm)	Altitude(1000ft.)	Mach Number
1	0	0	0
2	0	0	0.25
3	0	0	0.35
4	0.1	1.5	0.65
5	10	4	0.65
6	25	10	0.65
7	45	13	0.75
8	80	15	0.75
9	100	17	0.75
10	130	20	0.75
11	160	22	0.75
12	340	34	0.77
13	340	34	0.77
14	400	34	0.77
15	450	34	0.77
16	500	34	0.77
17	550	34	0.77
18	560	34	0.77
19	570	34	0.77
20	580	34	0.77
21	620	34	0.77
22	630	34	0.77
23	670	34	0.77
24	680	34	0.77
25	690	34	0.77
26	700	34	0.77
27	720	34	0.77
28	740	34	0.77
29	757	34	0.77
30	807	34	0.77
31	857	34	0.77
32	860	34	0.77
33	870	34	0.75
34	900	32	0.75
35	980	25	0.75
36	1025	20	0.75
37	1060	15	0.75
38	1100	10	0.68
39	1130	9	0.68
40	1150	7	0.68
41	1170	3	0.68
42	1190	2.5	0.68
43	1200	1.5	0.68

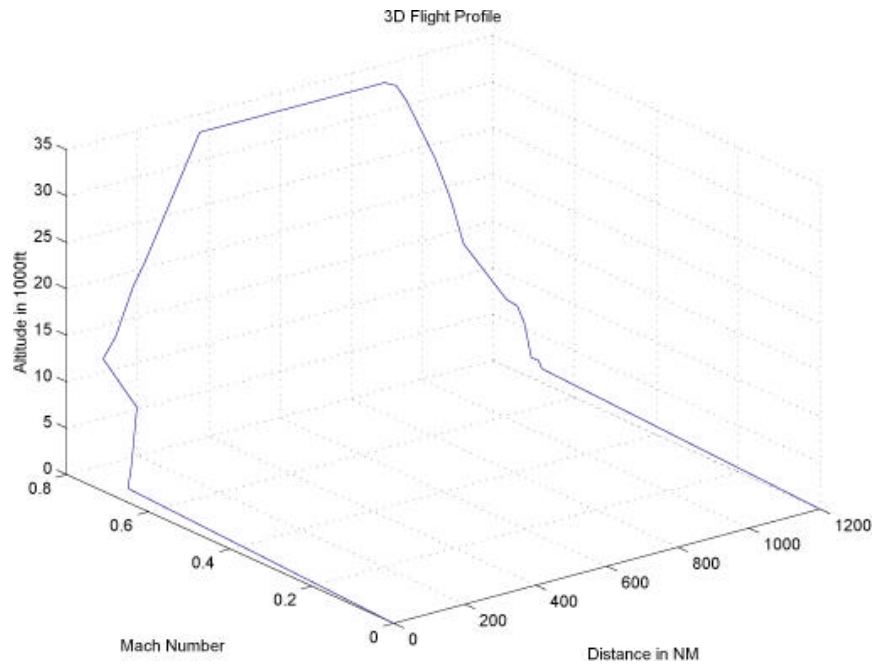


Figure 5.13 Three Dimensional Flight Profile.

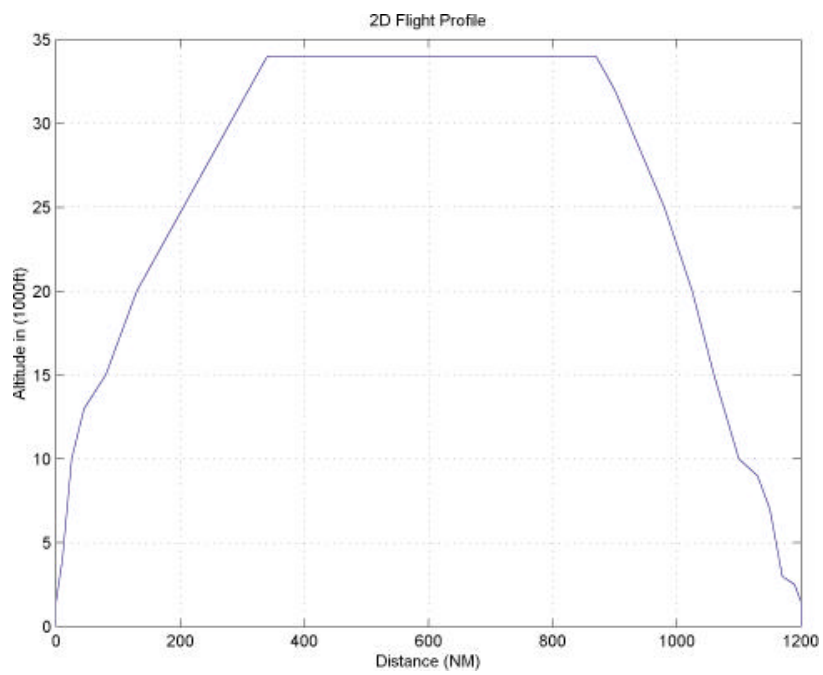


Figure 5.14 Two Dimension Flight Profile.

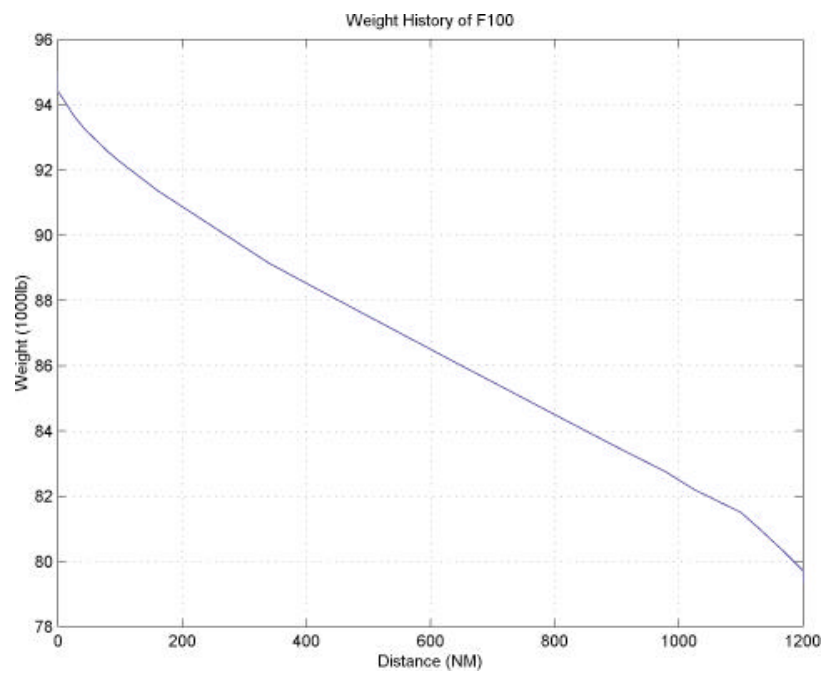


Figure 5.15 Weight vs Distance.

5.3 Neural Network Aided Fuel Consumption Results

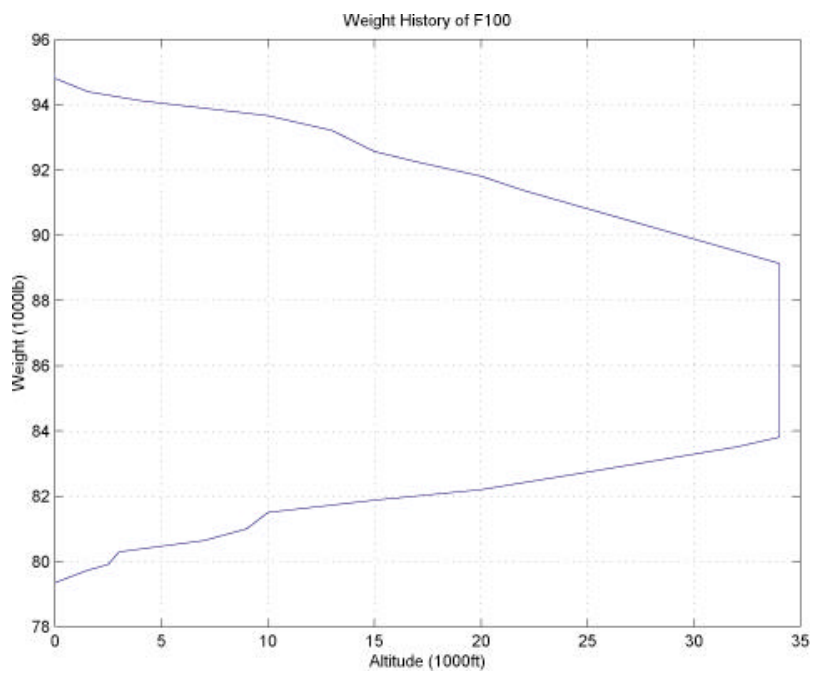


Figure 5.16 Weight vs Altitude.

For this research project, a neural network was used to calculate the fuel consumption of an aircraft. Results show that using neural network is a reliable and efficiency mean to estimate fuel consumption for a complex system with large output non-linearities. In addition, this research also provides an example on how expert systems can be implemented into engineering problems. Some conclusions of this research project will be stated in Chapter 6.

Conclusions and Recommendations

6.1 Conclusions

The existing 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 pilot flight manual, the neural network was trained to estimate fuel consumption of an example aircraft. Results were compared to the actual performance provided in the flight manual.

The following conclusions are drawn:

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.
2. The information provided in the pilot flight manual is a reliable source to obtain fuel consumption data of an aircraft. Along with neural network technology, a neural network aided fuel consumption model can be 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 an aircraft.

4. Neural network is found to be a viable alternative in fuel consumption estimating application.
5. Neural network is found to be a more efficient and accurate mean to calculate fuel consumption than using basic aircraft performance technique.

6.2 Remarks and Recommendations

6.2.1 Remarks

Neural Network

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 test and error approach. 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 exist 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. Details on how to implement this technique to fuel consumption mod-

6.2 Remarks and Recommendations

eling is described in Appendix D [Collins 1982]. 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.

6.2.2 Recommendations

Due to the time constraint of this project, the network was trained for one example aircraft. In order to complete this model, data from all other existing civil aviation aircraft should also be trained. In addition, a unique topology study for neural network aided fuel consumption should also be conducted. This study should select a pertinent topology which would be able to accommodate all possible aircraft. So, once the network is trained for other aircraft, a data base of weights and biases can be established. Then, by developing a routine to simulate the feed-forward process, neural network aided fuel consumption modeling would become a universal post processor (or in line processor) for any flight trajectory generation program. 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.

Bibliography

-
1. Alexander, I., "Neural Computing Architecture." , MIT Press, 1989.
 2. Anderson, John D. Jr., "Introduction To Flight.", McGraw Hill, Inc.,1989.
 3. Beal, Mark. and Demuth, Howard., "Training Functions in a Neural Networks.", Academic/Industrial/NASA. Defense'92., SPE Vol.1721,1992.
 4. Caudill, M., "What is a Neural Network.", AI Expert, Neural Network Special Report, 1992.
 5. Collins, B.P., Haines, A.L., and Wales W.J., "A Concept for a Fuel Efficient Flight Planning Aid for General Aviation.", NASA Contractor Report 3533,1982.
 6. Collins, B.P., " Derivation and Current Capabilities of the Path Profile Fuel Consumption Algorithm.", The MITRE Corp, McLean., VA MTR-80W195, 1980.
 7. Dayhoff, J.E., Neural Networks Architecture, Van Nostrand Reinhold, New York., 1990.
 8. Devenport, William., " AOE 3054 Lecture Notes.", Virginia Polytechnic Institute and State University., 1995.

9. Drew, Donald., "CE 2604 Notes.", Virginia Polytechnic Institute and State University., 1993.
10. Fokker F-100 Pilot's Flight Manual, Fokker Aircraft, Inc.
11. Freeman, J.A., "Back Propagation in a Neural Network." AI Expert Neural Network Special Report., 1992.
12. Hagan, Martin T., " Neural Network Design.", DWS Publishing Company., 1996.
13. Khanna, T., "Foundations of Neural Networks.", New York., Addison-Wesley., 1996.
14. MATLAB User's Guide., Version 4.0., The Mathworks, Inc., Prentice Hall., 1995.
15. McCormick, Barns W. "Aerodynamics, Aeronautics and Flight Mechanics." New York, London: John Wiley and Sons, 1995.
16. Neural Network Toolbox User's Guide, The Mathworks, January 1994.
17. Schilling, Glenn., "Modeling Fuel Consumption with a Neural Network.", Virginia Polytechnic Institute and State University, 1996.
18. Torenbeek, " Synthesis of Subsonic Airplane Design." Delft University Press, 1982.

Appendix A

MATLAB programs

Contents:

A. Neural Network Training programs

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

B. Neural Network testing program

1. Testing main program
2. Statistical analysis

C. Main program to calculate fuel consumption

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

A. Neural Network Training programs for;

1. Take-off and climb out fuel estimation

```
%NEURAL NETWORKS TRAINING FOR TAKEOFF AND CLIMBOUT
%DEVELOPED BY FRANK CHEUNG
%UNDERSUPERVISION OF DR. ANTONIO TRANI
%LAST MODIFIED 23/06/97
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);
ISAc(i)=cof(i,3);

end

% Data Normalization

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

%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 Training Parameters

df = 10000; % Frequency of progress displays (in epochs).
```

1. Take-off and climb out fuel estimation

```
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
```

2. *Climb performance estimation*

```
% NEURAL NETWORKS TRAINING FOR CLIMB PHASE
%DEVELOPED BY FRANK CHEUNG
%UNDERSUPERVISION 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);
```

2. Climb performance estimation

%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 Training 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 Bias

```
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

```

3. Cruise specific air range

3. *Cruise specific air range*

```
% NEURAL NETWORKS TRAINING FOR CRUISE PHASE
%DEVELOPED BY FRANK CHEUNG
%UNDERSUPERVISION 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 Training 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 Bias

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
%UNDERSUPERVISION 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 Training 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 Bias

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');

% Training of the neural networks using Lavenberg-Marquardt Algorithm

[ 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 Training 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 Bias

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');

% Training 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 Training 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

```

B. Neural Network testing program

1. Testing main program

```
%NEURAL NETWORKS TRAINING FOR DATA TESTING
%DEVELOPED BY FRANK CHEUNG
%UNDERSUPERVISION 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);
```

1. Testing main program

```
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 = CBFL./10;

T2N = CBDL./10;

% Weight Normalization

WN1 = CBFW./WMX_cbf;

WN2 = CBDW./WMX_cbd;

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

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

1. Testing main program

```
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 ];

```

1. Testing main program

```
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');

```

1. Testing main program

```
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
```

2. Statistical analysis

```
%NEURAL NETWORKS TRAINING FOR STATISTICAL ANALYSIS
%DEVELOPED BY FRANK CHEUNG
%UNDERSUPERVISION 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 Training Results
```

```
fid = fopen('CRT');
CBT = fscanf(fid, '%g %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');
```

2. Statistical analysis

```
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 = CBFL./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;

```

2. Statistical analysis

```
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)

```

2. Statistical analysis

```
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

```

C. Main program to calculate fuel consumption

1. Main Program

```
%FUEL BURN CALCULATION
%DEVELOPED BY FRANK CHEUNG
%UNDER SUPERVISION OF DR. ANTONIO TRANI
%LAST MODIFIED 25/06/97

% 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

```

1. Main Program

```
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);

```

1. Main Program

%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,MDFD,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;

```

1. Main Program

```
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;
```

2. Climb subroutine

```
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;

```

2. Climb subroutine

$F1 = ((F1F3+F1F4)/2-(F1F1+F1F2)/2)*MCBF/1000;$

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

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

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

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

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

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

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

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

$F2F1 = \text{simuff}(P1F2, W31_cb_f, b31_cb_f, 'logsig', W32_cb_f, b32_cb_f, 'tansig', W33_cb_f, b33_cb_f, 'purelin');$

$F2F2 = \text{simuff}(P2F2, W31_cb_f, b31_cb_f, 'logsig', W32_cb_f, b32_cb_f, 'tansig', W33_cb_f, b33_cb_f, 'purelin');$

$F2F3 = \text{simuff}(P3F2, W31_cb_f, b31_cb_f, 'logsig', W32_cb_f, b32_cb_f, 'tansig', W33_cb_f, b33_cb_f, 'purelin');$

$F2F4 = \text{simuff}(P4F2, W31_cb_f, b31_cb_f, 'logsig', W32_cb_f, b32_cb_f, 'tansig', W33_cb_f, b33_cb_f, 'purelin');$

$D2D1 = \text{simuff}(P1D2, W31_cb_d, b31_cb_d, 'logsig', W32_cb_d, b32_cb_d, 'tansig', W33_cb_d, b33_cb_d, 'purelin');$

$D2D2 = \text{simuff}(P2D2, W31_cb_d, b31_cb_d, 'logsig', W32_cb_d, b32_cb_d, 'tansig', W33_cb_d, b33_cb_d, 'purelin');$

$D2D3 = \text{simuff}(P3D2, W31_cb_d, b31_cb_d, 'logsig', W32_cb_d, b32_cb_d, 'tansig', W33_cb_d, b33_cb_d, 'purelin');$

$D2D4 = \text{simuff}(P4D2, W31_cb_d, b31_cb_d, 'logsig', W32_cb_d, b32_cb_d, 'tansig', W33_cb_d, b33_cb_d, 'purelin');$

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

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

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

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

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;

2. Climb subroutine

```
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

3. Cruise subroutine

3. Cruise subroutine

```
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;
```

3. Cruise subroutine

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

4. 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,MDFD,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
```

4. Descent subroutine

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;

4. Descent subroutine

$F = (F1 + (F2 - F1) / (W2 - W1) * (W_{in} - W1)) / 1000;$

$D_d = D1_d + (D2_d - D1_d) / (W2 - W1) * (W_{in} - W1);$

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

4. Descent subroutine

```
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

An example of the Neural Network Trained Weight Matrices and Bias Vectors

For the cruise segment of flight, there are three inputs. Therefore, for the first hidden layer weight matrix the number of columns is three and since eight neurons configuration is selected, there are eight rows.

Weight matrix and bias vector for the first layer is shown as the following;

Weight matrix =

-13.8112	13.7230	-1.7968
17.6559	13.0522	-3.8619
21.9976	6.9291	2.9611
-22.1868	8.9597	-1.6210
5.9747	-7.3308	9.0976
-15.2327	8.7406	6.6544
11.0292	10.7460	-9.2543
-1.9639	9.3912	12.2075

Bias vector for the first layer:

$$\text{bias vector} = \begin{pmatrix} 5.3226 \\ -20.2318 \\ -25.4196 \\ 12.1188 \\ -4.3895 \\ -1.9267 \\ -14.0892 \\ -7.7072 \end{pmatrix}$$

Since inputs of the second layer come from eight different neurons in the first layer, therefore the size of the matrix is 8X8.

$$\text{Weight of the second layer} = \begin{pmatrix} 0.2868 & -0.5916 & -0.3281 & 0.5151 & 0.0712 & -0.1757 & 0.7190 & -0.1698 \\ 0.1426 & -0.0674 & 0.1017 & 0.1996 & 0.2077 & 1.0992 & 0.3701 & -0.1607 \\ -1.6738 & 0.6003 & -0.0541 & -0.4932 & -0.2116 & -0.9090 & 3.8835 & -0.0803 \\ 0.6498 & -1.0703 & 0.2790 & 0.4042 & 0.3626 & -2.4492 & -0.5578 & -0.4066 \\ 0.3993 & -0.6693 & 0.1188 & -1.0067 & -0.3354 & 0.4921 & 0.1003 & 0.8228 \\ -1.1876 & 0.8426 & -0.9094 & 0.9315 & -1.4362 & 1.7100 & -0.4895 & -1.1878 \\ 0.4817 & -0.4705 & 0.3483 & -0.8956 & -0.0160 & 0.7544 & -0.0971 & 0.7029 \\ -1.1102 & -1.1550 & -0.0401 & 1.1413 & -0.1158 & 1.2824 & -0.7206 & -0.3389 \end{pmatrix}$$

The corresponding bias vector is shown as the following:

$$\text{bias of the second layer} = \begin{pmatrix} 0.6120 \\ -1.4202 \\ 3.8455 \\ 1.6905 \\ -0.2560 \\ 2.1781 \\ -0.5618 \\ -0.6967 \end{pmatrix}$$

Finally, for the output(third) layer, there is one neuron in this layer. The weight matrix for this layer is shown as the following:

$$\text{Weight matrix} = [0.5015 \quad -1.6234 \quad -2.5576 \quad -0.2638 \quad -1.1681 \quad 0.1226 \quad 1.4716 \quad -0.2856]$$

and the corresponding bias vector is

$$\text{Bias vector} = [1.4329]$$

Appendix C

Statistical Results

Contents

Histogram of

1. Climb Distance Estimation
2. Cruise Specific Air range
3. Descent Distance Estimation
4. Descent Fuel Estimation

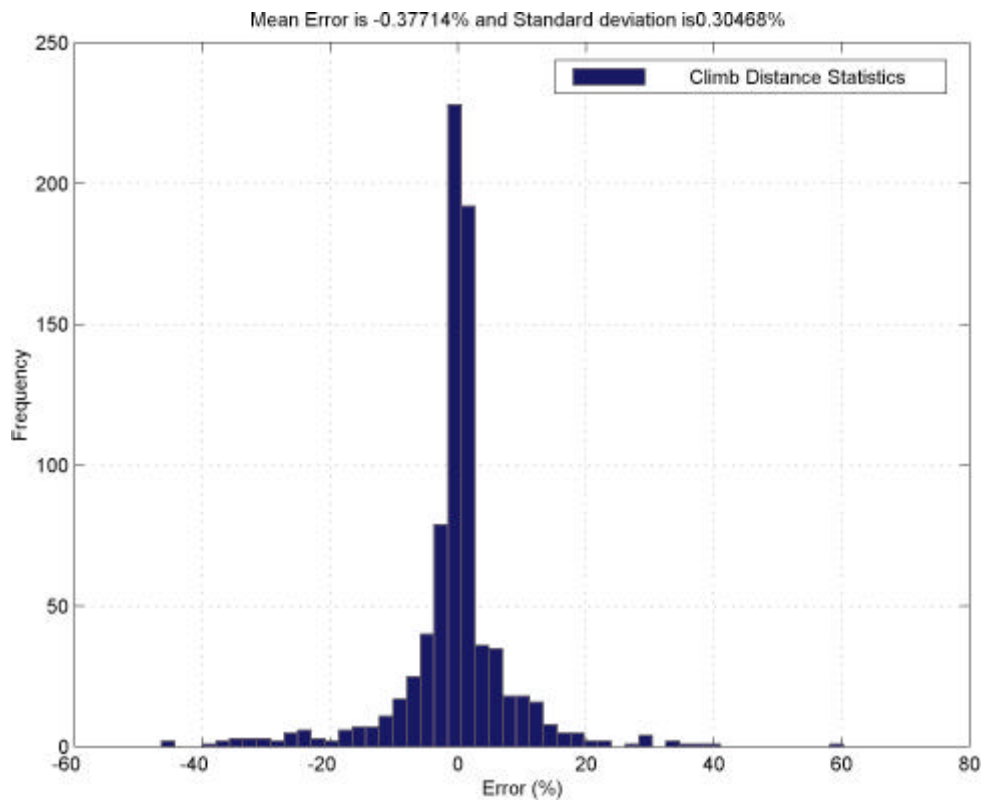


Figure C.1 Error Distribution of Climb Distance Estimation.

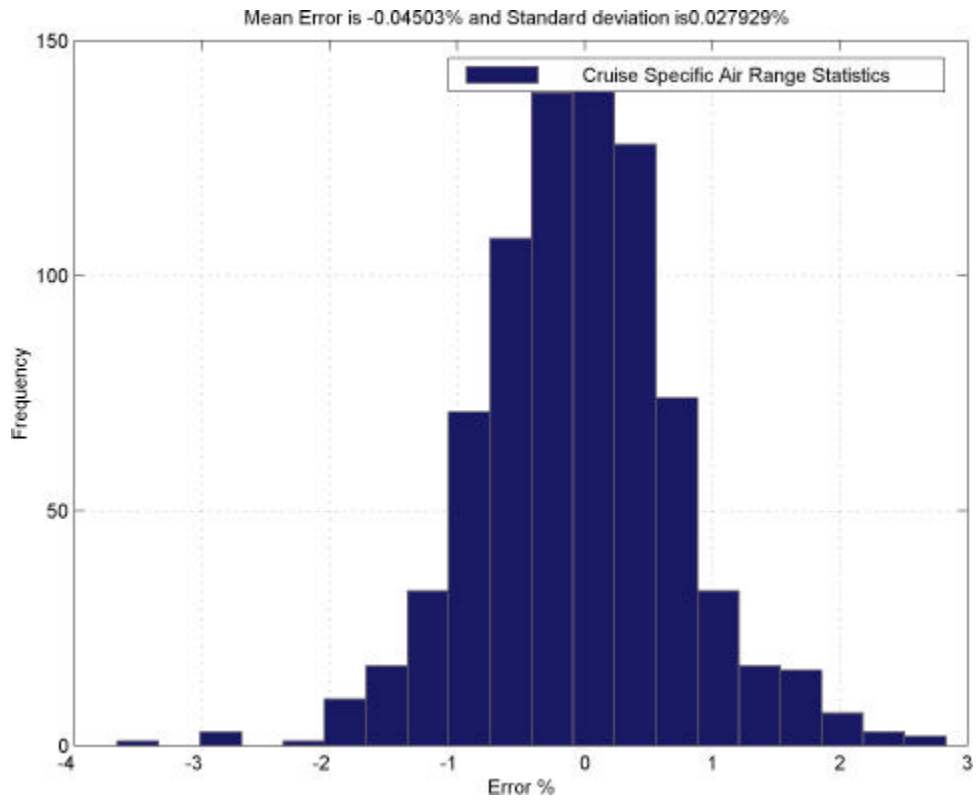


Figure C.2 Error Distribution of Cruise Specific Air Range Estimation.

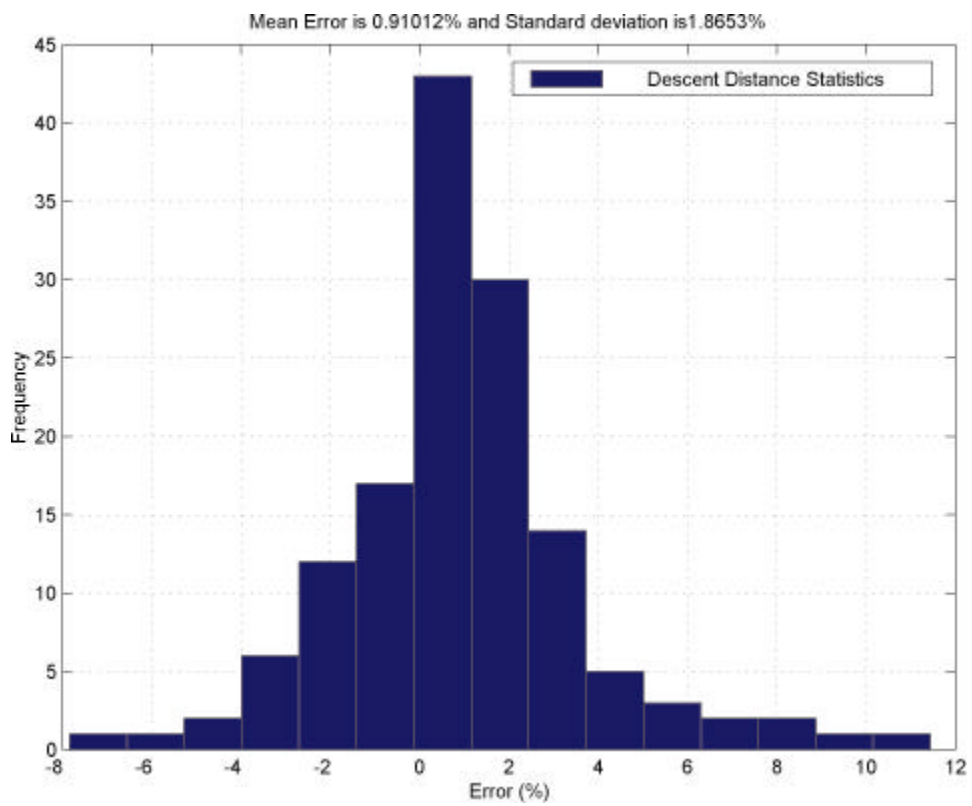


Figure C.3 Error Distribution of Descent Distance Estimation.

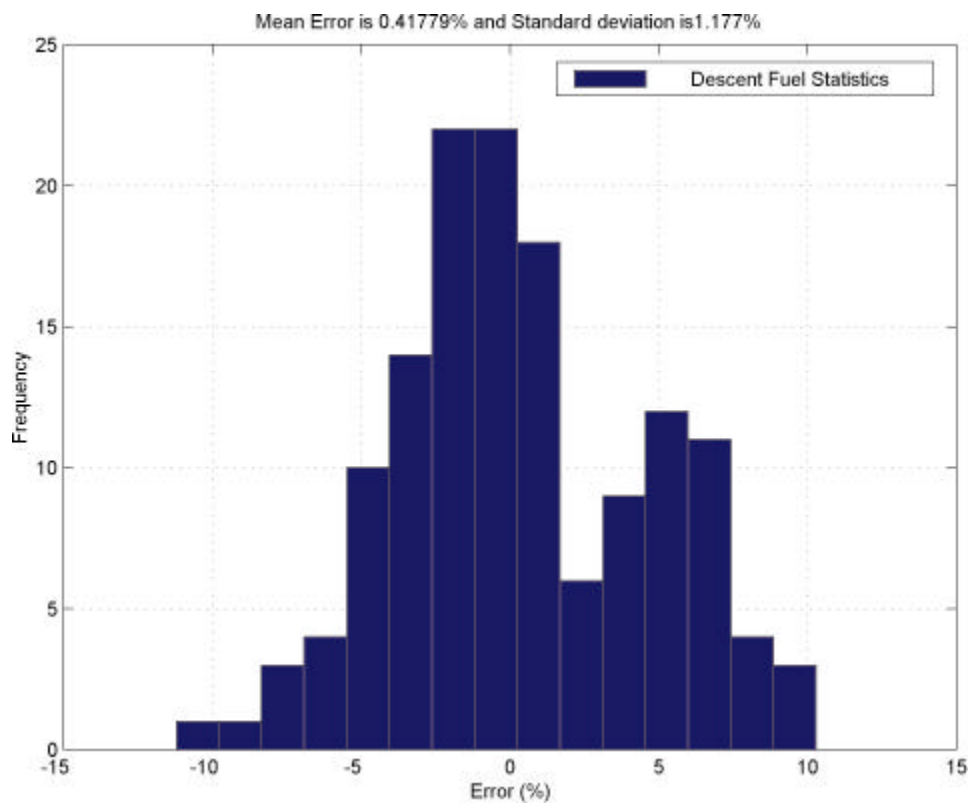


Figure C.4 Error Distribution of Descent Fuel Estimation.

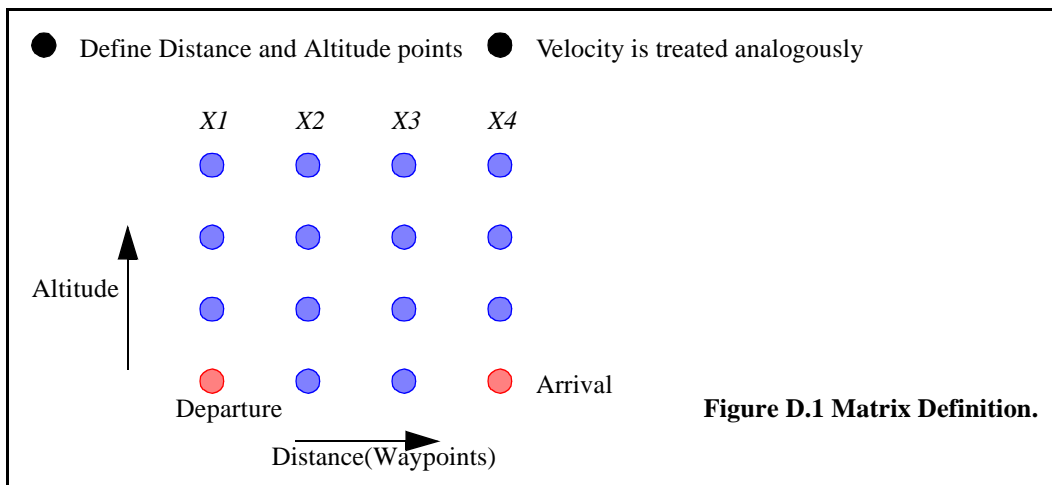
Appendix D: Fuel Efficient Path

D.1 Philosophy

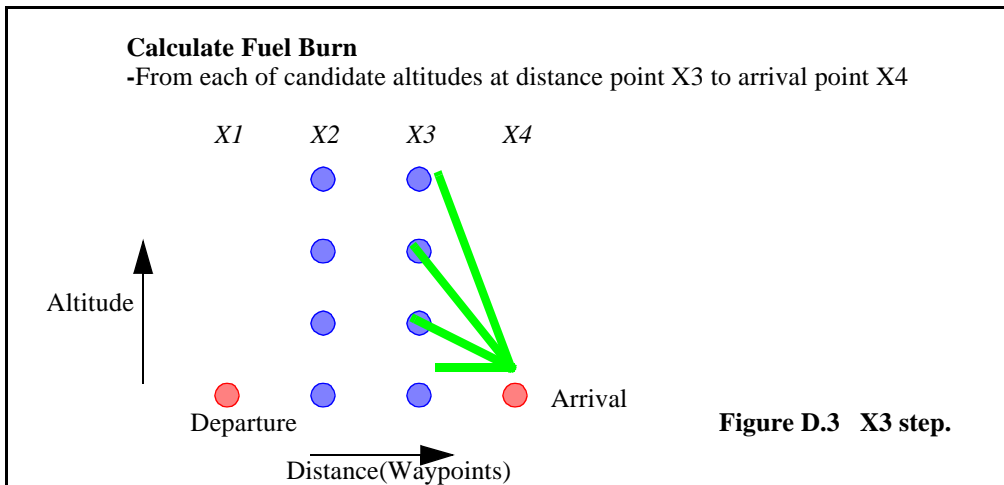
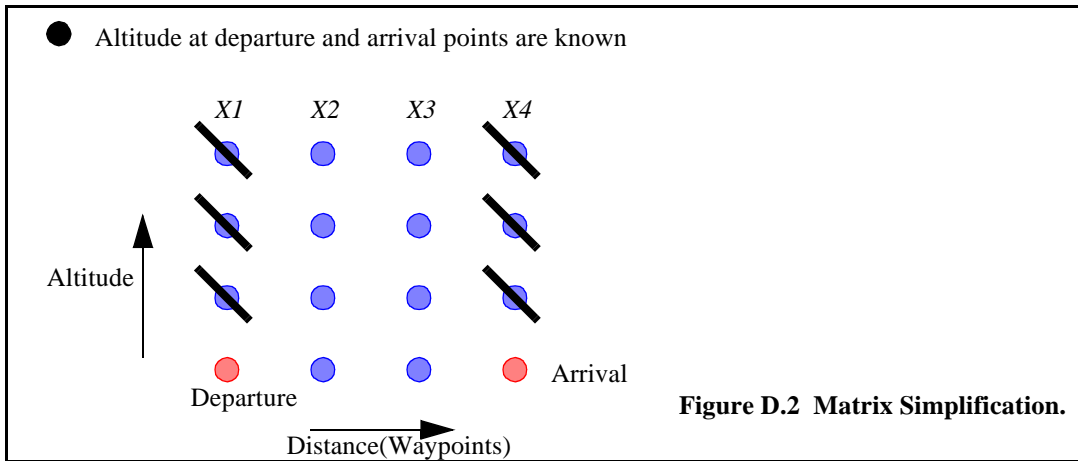
The solution approach is to develop a new fuel burn model as a basis for calculation of fuel burns over specific segments. The fuel efficient path would be established by a dynamic programming technique. This technique iteratively and systematically develops a set of feasible flight paths that merge to the single desirable optimal fuel efficient trajectory.

The new fuel burn model mentioned above is a model that employs neural networks to generate a table lookup function. This function would be able to estimate the fuel burn over a period of time, corresponding to the trajectory generator. In this way, the optimal flight path can be generated in an effective and accurate manner.

This section demonstrates the fundamental idea of the approach to the development of a fuel efficient flight profile. For the purpose of depicting the method, velocity is assumed to be known at each distance way point. The dynamic programming approach is illustrated for a four-step problem in Figures D.1 to D.6.

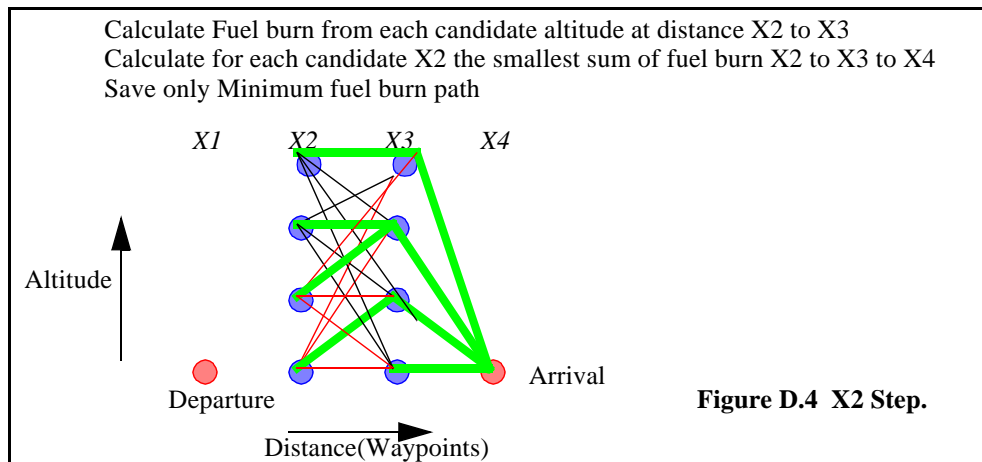


The initial step is to define a set of potential density altitudes for each user defined distance way point as shown in Figure D.1. The altitudes for departure and arrival are known, and therefore immediate simplification is possible, as shown in Figure D.2. Starting from the destination (arrival) point, selectively and iteratively, the fuel burns are calculated from each candidate (performance point) very next to the destination and so on.

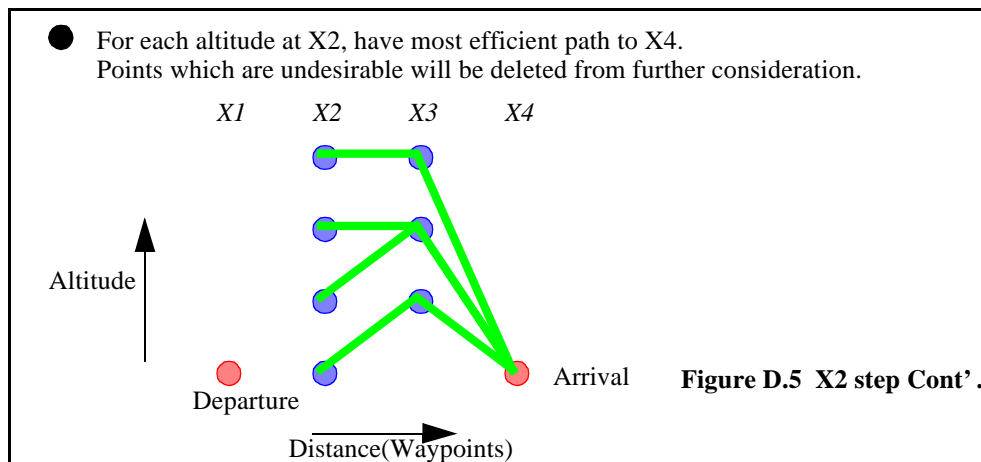


Firstly, the fuel burn is calculated from each candidate altitude at distance X_3 to the arrival point at distance X_4 , as depicted in Figure D.3. The next step is to consider distance X_2 . For each candidate altitude at X_2 , the fuel

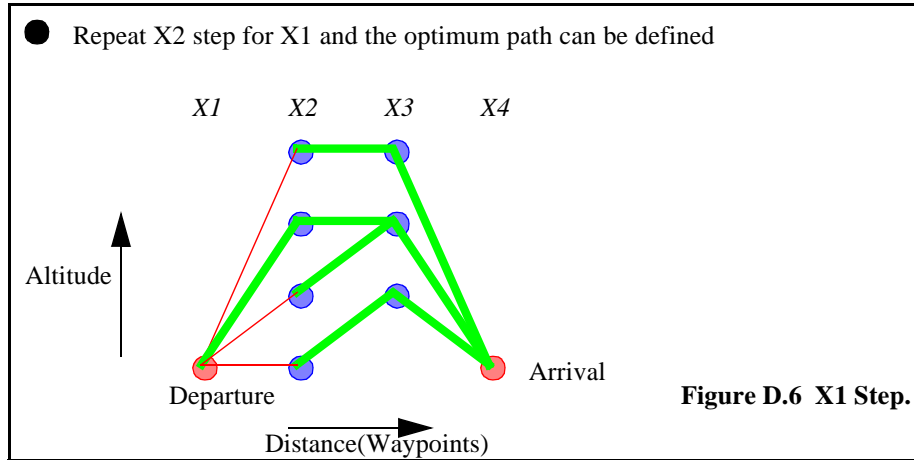
burn is calculated to distance X_3 . Since the fuel burns from distance X_3 to the arrival point at distance X_4 for



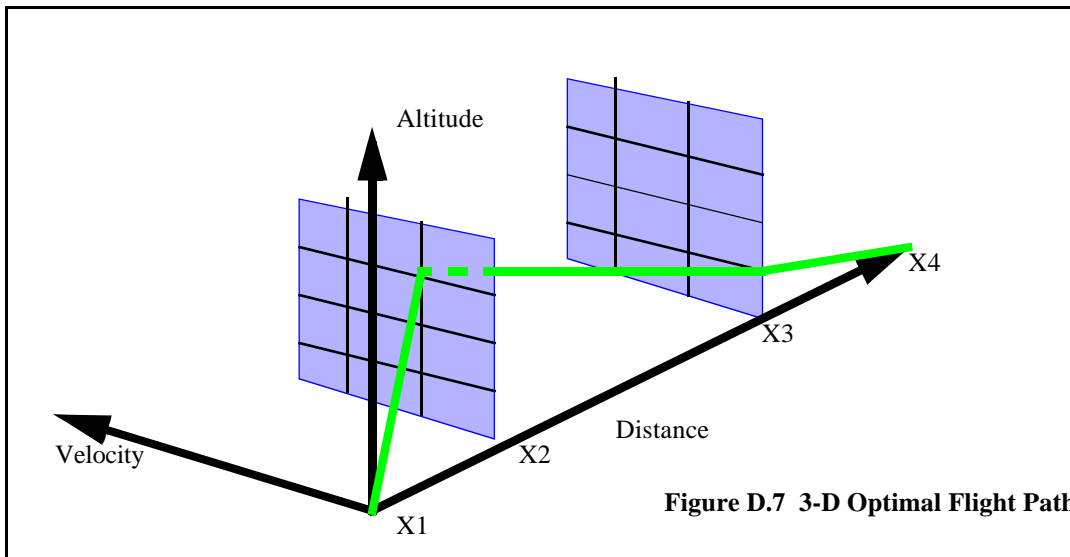
each candidate at X_3 are known, the most fuel efficient route from each altitude point at X_2 to X_4 can be calculated. The solutions for one element in distance X_2 is given as shown in Figure D.4. Iterating the aforementioned scheme for each element (desirable altitude) in X_2 the most fuel efficient route path for each can be



found as shown in Figure D.5. Notice that, during the process, altitudes which do not lie on any fuel efficient path segment can be eliminated from further consideration. Finally, the fuel burn from a single altitude at departure point X_1 to all altitude candidates at distance X_2 can be calculated. Since the most efficient fuel paths from altitudes X_2 to X_4 are known, therefore the most optimal from X_1 to X_4 can be found, shown by the line in Figure D.6.



Altitude is not the only parameter which affects the fuel consumption; velocity is also a factor. Therefore, the determination of both velocity and altitude is required for each way point X_i , for i = number of way points. However, the logic remains the same, except for the fact that the model is now 3-dimensional, as shown in Figure D.7. Here the primary modification is in the number of fuel burn calculations that are required at each step. For instance, in the demonstration shown above at distance X_3 the number of candidates are 4 altitudes times 4



velocities which equals to 20 combinations versus the original 4 altitude candidates.

D.2 Mathematics

This section presents the computational procedures involved in the model discussed above.

D.2.1 Input initialization

Inputs required from the user are as follows:

- Departure and Arrival - altitude and velocity (DA,AA,DV,AV)
- Number of nodes for altitude, velocity and way points (X,Y,Z)
- Maximum Altitude, Rate of Climb, Velocity and Rate of Descent (MA,MRC,MV,and MRD)
- Landing Gross Weight

With the listed inputs, a three dimensional matrix can be established.

For each way point,

$$Altitude(i) = Altitude_{min} + \frac{(i-1)}{X-1}(Altitude_{max} - Altitude_{min})$$

$$Velocity(j) = Velocity_{min} + \frac{(j-1)}{Y-1}(Velocity_{max} - Velocity_{min})$$

where $i=1,2,\dots,X$ and $j=1,2,\dots,Y$

and,

$$Distance(k) = Distance_{min} + \frac{(k-1)}{Z-1}(Distance_{max} - Distance_{min})$$

Therefore, each velocity and altitude at a specific way point can be represented by (i,j,k) .

D.2.2 Calculation required for the first segment

For the purpose of this discussion, assume the destination point is at Z , i.e.

$\text{Distance}(k = Z)$ = Distance between the origin and destination.

The first step is to calculate the fuel burn required from way point $Z-1$ to Z . Since at way point Z there is a defined velocity and altitude which has to be satisfied, along with the initial altitude and velocity $(i, j, Z-1)$, fuel burns can be estimated using the neural network fuel consumption model, denoted as $\text{FB}(i, j, Z-1)$.

D.2.3 Generic Segment

Having calculated fuel burns required for the first segment, the next step is to calculate fuel burn over generic segments. The approach is very similar to the one previously discussed.

Perform calculations between two way points, i.e. $\text{Distance}(k)$ to $\text{Distance}(k-1)$. The specific pair of points under consideration would be denoted as (i, j, k) and $(m, n, k+1)$.

Before proceed any further, these two points must be checked to see if any of the performance constraints are violated. These constraints are:

- Maximum fuel flow for climb and cruise
- Descent prior to ascent, if $\text{Altitude}(i) > \text{Altitude}(m)$ and $\text{Altitude}(i, j, k-1) = 0$
- Deceleration prior to acceleration, if $\text{Velocity}(j) > \text{Velocity}(n)$ and $\text{Velocity}(i, j, k-1) = 0$
- Climb gradient $((\text{Altitude}(i) - \text{Altitude}(m)) / (\text{Distance}(k) - \text{Distance}(k-1)))$ greater than 10%. This can easily be changed to a function of altitude.

If any of the above are violated:

$$\text{FB}(m, n) = 999999999$$

Otherwise fuel burn from $(i, j, k-1)$ to (m, n, k) is calculated and denoted as $\text{FB}(m, n)$.

Then,

$$W(i,j,k-1) = \min_{m,n}[B(m,n) + W(m,n,k)]$$

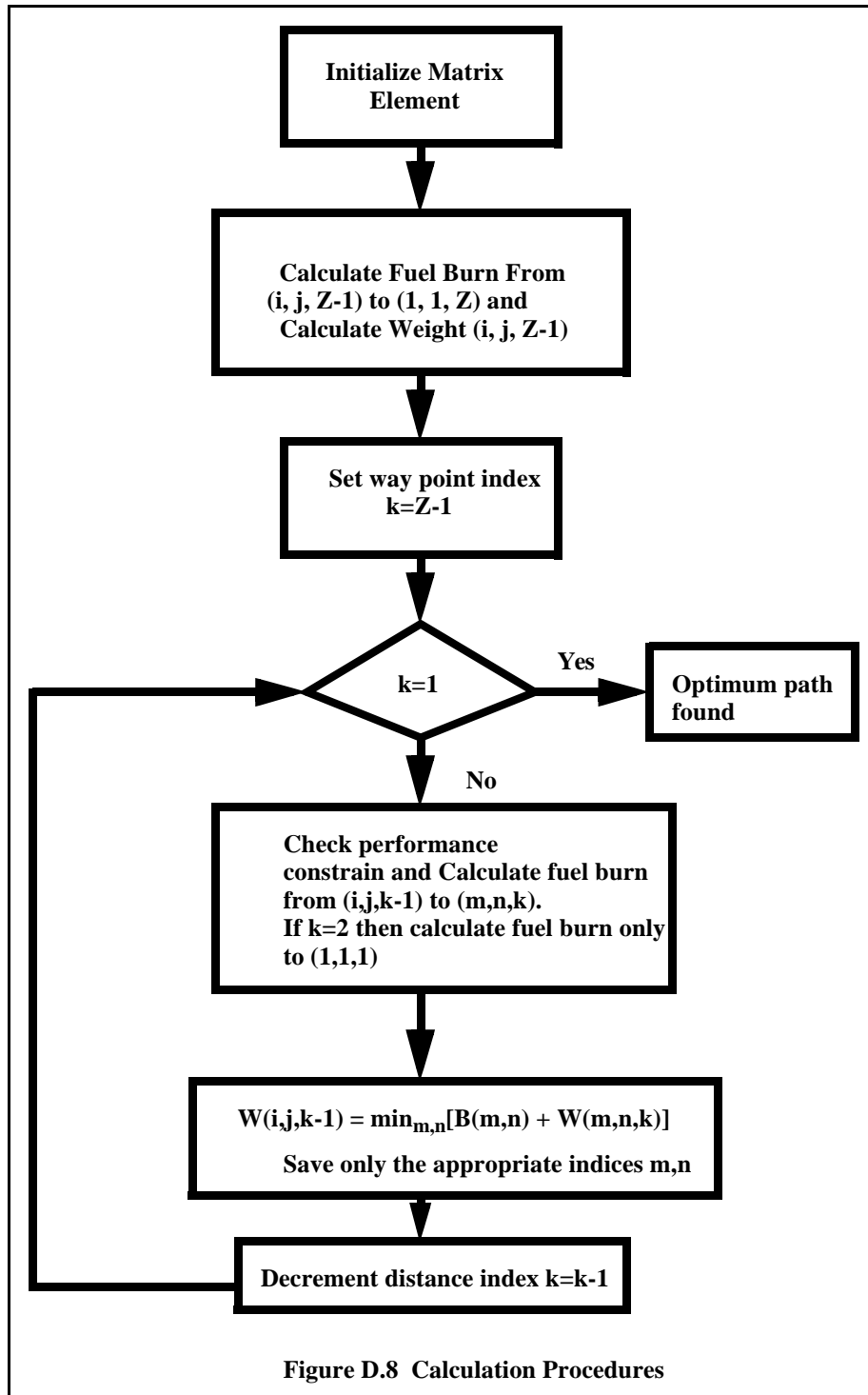
The above steps should be repeated for each $(i, j, k-1)$ until distance $(i, j, k-1) = 0$ or $k-1 = 1$.

Again, calculate fuel burn between (m,n,k) and a single point $(1,1,1)$, so that

$$W(1,1,1) = \min_{m,n}[B(m,n) + W(m,n,k)]$$

Since each point $(i,j,k-1)$ has only one optimum, having reached the point $(1,1,1)$ means a single fuel efficient optimal path has been obtained.

The calculation steps are summarized in Figure D.8.



Vita

Wing Ho Cheung (Frank) was born on Halloween, 1973 in Hong Kong. He graduated from Tsuen Wan Government Secondary School, in July 1991 and graduated again from Natural Bridge high school, in May 1992. He received the degree of Bachelor of Science in Aerospace Engineering from Virginia Polytechnic Institute and State University in May 1996. In August 1996, he joined the graduate transportation engineering in the Civil Engineering Department at the same university. After completing the requirements of Master of Science degree in Civil Engineering in August 1997, he is working as a consultant for Ricondo and Associates, Inc., in Chicago Illinois.

Frank Cheung

August 1997