A Computer Model to Estimate Commercial Aviation Fuel Consumption and Emissions in the Continental United States

by
Zhihao Zou

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

Antonio A. Trani, Chair
Linbing Wang,
Montasir M. Abbas,

November 13, 2012
Blacksburg, Virginia

Keywords: Fuel Consumption, Emissions, TASM, BADA, Aircraft, Air Transportation

A Computer Model to Estimate Commercial Aviation Fuel Consumption and Emissions in the Continental United States

Zhihao Zou

ABSTRACT

A comprehensive model is developed to estimate and predict the fuel consumption and emissions by domestic commercial aviation in the Continental United States. Most of the existing fuel consumption and emission models are limited in their ability to predict the annual fuel burn for air transportation at the national level. For example, those models either require real track data or are developed only to model single flight scenarios. The model developed in this thesis is part of a software framework called the Transportation Systems Analysis Model (TSAM). The model has the capability to estimate fuel consumption and emissions for millions of domestic flights in a year in the continental U.S. TSAM is a nationwide, long-distance, multimodal travel demand forecast model developed at Virginia Tech. The model enables TSAM to quantify fuel and emission metrics for various modes of transportation.

The EUROCONTROL Base of Aircraft Data (BADA) is employed as the Aircraft Performance Model to simulate individual flight profiles and calculate fuel burn rates. Fuel consumption on the ground (taxi mode) is estimated separately. Different operational conditions like wind states, terminal area detour, cruise altitude and airport elevation are considered in the model. Emissions of HC, CO, NOx and SOx are computed inside the Landing/Take-off (LTO) cycle based on the fuel consumption estimate, while greenhouse gas of CO_2 is calculated for the complete flight cycle.

ACKNOWLEDGEMENTS

I must first thank Dr.Trani for offering me the opportunity to work at his lab, guiding me and supporting me through my graduate study at Virginia Tech. His patience and humbleness have left an everlasting impression. I would also like to thank Dr.Abbas and Dr.Wang for being my committee members and providing me suggestion in the research.

Second, I would like to thank Maria Rye, a student in Aerospace Engineering. Her contributions in the initial stages of the model development are greatly appreciated. Additionally, I want to thank my lab mate Saloni Chirania especially for helping me with my English language skills.

Finally, I am grateful to my family. Without your support, I cannot have done so many things and I could not complete my graduate studies in the United States. I am always indebted to you. This work is also dedicated to my aunt and uncle who died when I was studying in the United States. I missed the chance to see you last time, but you are in my memory forever, R.I.P.

ATTRIBUTION

Maria Rye, a student in Aerospace Engineering, has helped in the development of this model. Following is a brief description of her contributions to the research project.

Section 3.4: Flight Profile Generation

Maria developed the code to create climb and descent profiles in flight profile generation. She also created a custom Runge Kutta 4^{th} order ODE solver to integrate the wind vectors into flight profile.

Section 3.7: Wind Effect Analysis

She developed a function in MATLAB to calculate the wind vector at the given coordinates and altitude.

Section 3.8: Analysis of Terminal Area Detour Factor

She did the cluster analysis for the airports in the region where PDARS data are available. She created the mechanism and developed the code to extract detour factor for each airport.

Table of Contents

LIST of FIGURES

LIST of TABLES

1. INTRODUCTION

1.1 Background

High fuel prices have been a major concern for the aviation community for many years, which is now a significant hurdle. As we can see in Figure 1, the price of JetA1 fuel keeps at high levels since 2005 (Bureau of Transportation Statistics, 2012).

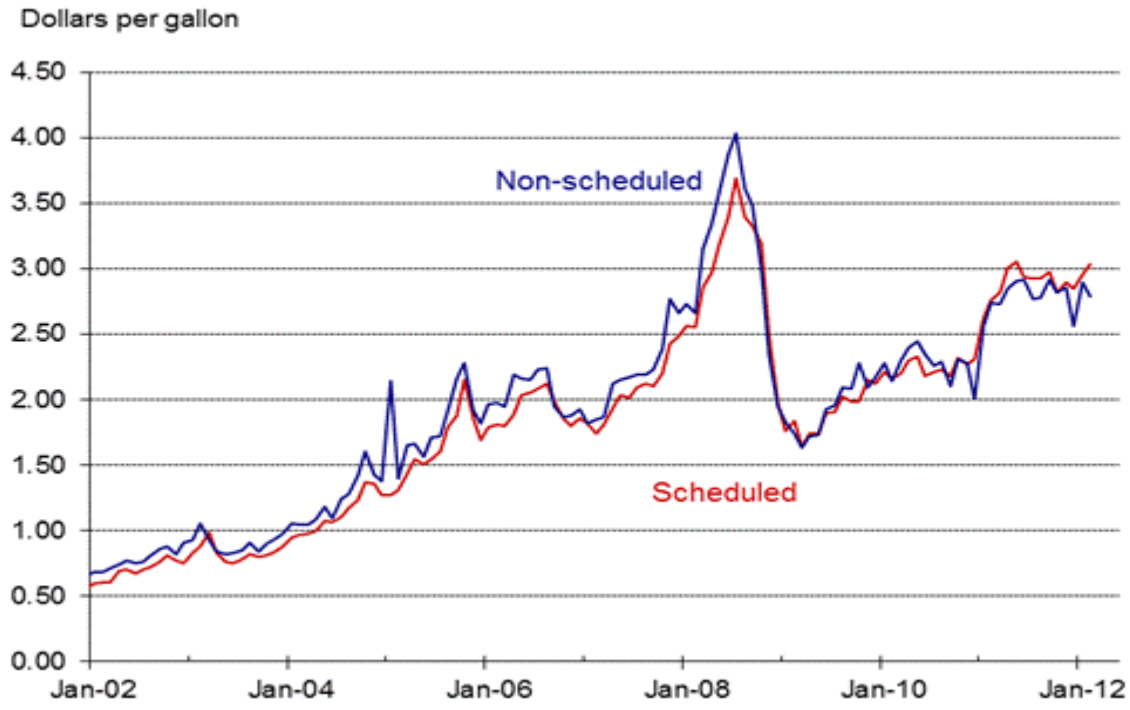


Figure 1. Trend Of JetA1 Prices Over The Years (Source: BTS 2012). Used Under Fair Use, 2012.

It is also reported that jet fuel prices accounted for 25.6% of airline operating expenses in2010. (Bureau of Transportation Statistics, 2012) Changes in fuel costs significantly affect ticket prices

and airline's profitability. Figure 2 demonstrates the percentage of the fuel costs among airline's total operating cost from 2003 to 2012 (IATA, 2012).

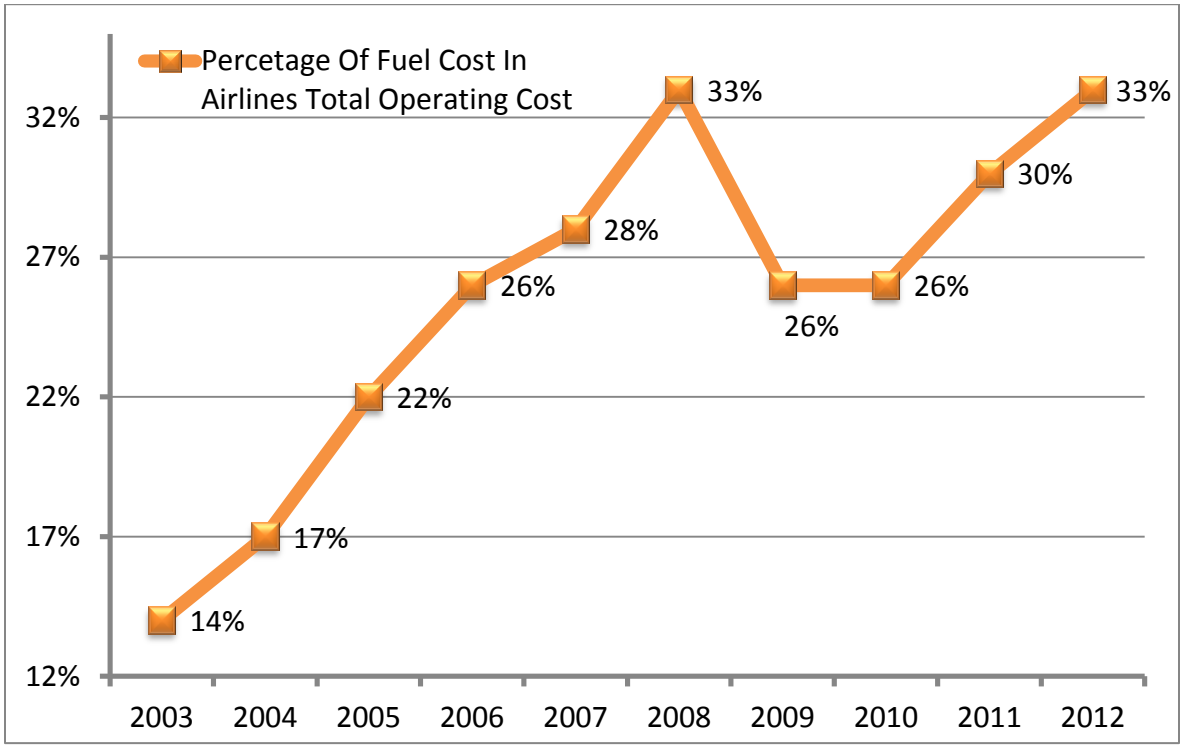


Figure 2 Percentage Of Fuel Expense In The Airline's Total Operating Cost (Source: IATA). Used Under Fair Use, 2012.

A desire for new aircraft that have better performance in speed, payload and range is limited by the fuel efficiency. Aircraft manufacturers are focusing their effort in marketing fuel-efficient and environment-friendly airliners. The unveiling of Boeing 787 Dreamliner is a good example.

Additionally, from the perspective of transportation mode choice, energy consumption is also of more concern than ever with the constant increases in fuel prices. Reduced vehicle energy consumption means lower travel cost for long distance travelers, resulting in a potential increase demand for the air transportation mode. To predict the future demand and energy consumption, a multimodal intercity model is needed to capture the effect of fuel consumption and emissions on the market share of different transportation modes in United States.

The Transportation System Analysis Model (TSAM) is a multi-mode, national-level, transportation framework developed at Virginia Tech using the classical four-step planning process (Wing-Ho, 2008). TSAM currently evaluates demand for commercial airline, automobile and high-speed rail. TSAM estimates county-to-county, nationwide demand for each mode of transportation using county-level socio-economic data forecast available until 2040. With the model presented in this thesis, TSAM is now able to estimate the energy consumption and emissions for various modes of transportation there by allowing users to analyze the benefits of technologies that could impact energy efficiency and the environment.

1.2 TSAM Overview

The transportation systems analysis model (TSAM) was developed to understand the interrelationships among ground and air transportation systems in the United States (Trani, 2008). There are 3091 counties along with 443 commercial service airports modeled in TSAM (Ashiabor, Baik and Trani, 2007). Each county is associated with n candidate airports including the closest airports by driving distance, the cheapest airports by average fare and the airports with the highest enplanements (Trani, 2008). TSAM forecasts annual person round-trips by air taxi, commercial airline, rail and automobile between all counties in the U.S. The process of demand estimation is separated into business and nonbusiness trip purposes and five household income group levels (Trani, 2008). The mode choice model in TSAM outputs person round-trips for each mode by trip purpose and household income group for all county pairs. The commercial airline network assignment model in TSAM uses the origin-destination airport person round-trip demand to calculate the number of passengers on each commercial airline route. TSAM employs a multinomial logit model to calculate the probability of each commercial airline route being selected by a traveler on the basis of travel time and airfare. With the demand on each

commercial airline route, it is possible to estimate the fuel consumption and emissions for the commercial airline mode. The structure of TSAM is shown in Figure 3 (Chirania, 2012).

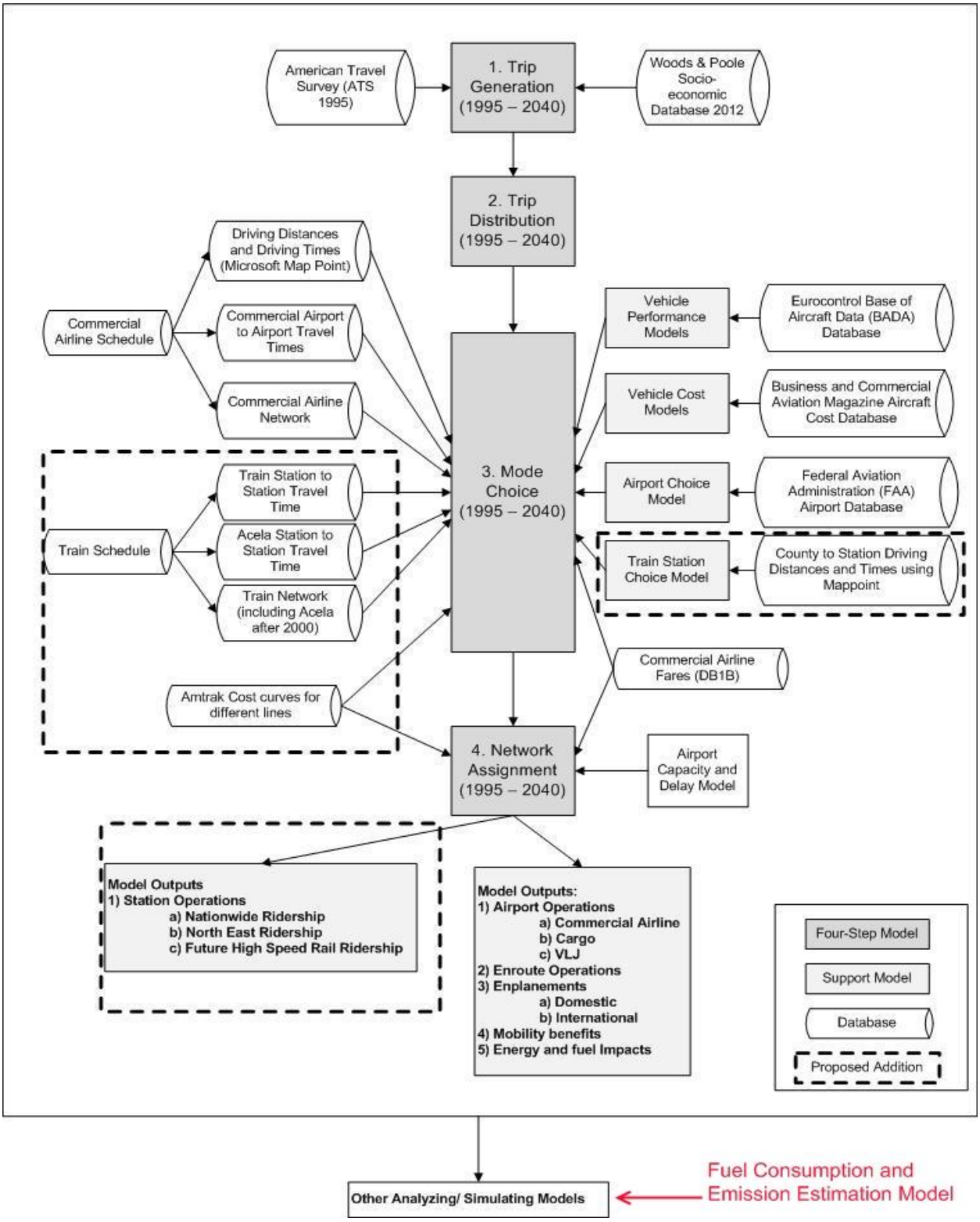


Figure 3. Structure Of TSAM (Chirania, 2012). Used With Permission Of Saloni Chirania, 2012.

1.3 Research Scope

The main objective of the computer model presented is to estimate commercial aviation fuel and emissions for all phases of flight. An accurate estimation of fuel and emissions will help to analyze the benefits of each transportation mode in terms of energy efficiency and environmental impacts. Additionally, the model developed can help policy makers to decide the future investment and developments for each transportation mode. The scope of this research model can be summarized as follows:

- Apply the BADA model along with other models and databases to estimate the fuel consumption based on the flight demand produced by TSAM.
- Validate the model by using T100 data as model input (flight information) and validate the result with Bureau of Transportation Statistics (BTS) airline fuel consumption reports.
- Develop a sub-model to calculate the emissions by produced commercial airlines based on the output of fuel consumption estimation model.

To reach these objectives, the EUROCONTROL BADA Aircraft Performance Model is used to produce an aircraft flight profile. FAA Enhanced Traffic Management System (ETMS) data are used to obtain flight waypoints and assign flown cruise altitudes. The performance Data Analysis and Reporting System (PDARS) data is used to extract terminal area detour factors. The FAA Aviation System Performance Metrics (ASPM) provides data for airport taxi out and taxi in time which is essential for ground fuel consumption and estimation. Finally, wind states are also taken into account in the model. The NCAR/NCEP database provides wind data for a given day for all coordinates and altitudes in the continental US. To estimate emissions, we used the emission factors database contained in the FAA Emissions and Dispersion Modeling System (EDMS).

1.4 Research Objective

The aim of the work is to develop a model to estimate current and future fuel consumptions and emissions for the commercial airline demand predicted by TSAM. The model should be generic to also be able to handle other input such as NASA ACES data or T100 data. Econometric models use factors such as demography and social economics to predict the energy consumption in the future. This work aims at developing a more realistic and comprehensive model based on parameters like demand, change of the route, cruise altitude assignment, to name a few. Further, another practical potential of the model is the sensitivity analysis to aid decision making for Air Traffic Management. The current state of the model can perform sensitivity tests on parameters like speed, take-off weight, and cruise altitude.

2. LITERATURE REVIEW

2.1 Overview

The fuel and emissions model developed has two sub-models: 1) a fuel consumption sub-model and 2) an emissions sub-model. In this chapter, we present past efforts to model aviation fuel consumption and emissions. Consulting companies, research institutes, governments and international organizations have developed models and calculators to estimate fuel consumption and emissions. However, most of the existing models are limited in their capability predict annual fuel burn for air transportation at the national level. For example, those models either require real track data or are developed only to model single flights.

2.2 Existing Fuel Consumption and Emissions Estimation Model

2.2.1 Fuel Consumption Model

In the early 1980s, B.P.Collins proposed an algorithm to predict aircraft fuel consumption derived from the basic concept of energy balance which can be expressed as $(energy\ in)$ – $(energy\ loss)$ = $(energy\ change)$. When expressed in terms of aircraft physical variables, $f(E_T) - f(E_D) = f(\Delta KE) + f(\Delta PE)$ where E_T is thrust energy, E_D is drag energy, and ΔKE and ΔPE are the changes in kinetic and potential energies respectively (Collins, 1982). It is assumed that the changes in the consumption of fuel energy continuously balance the energy losses and gains as a system during the aircraft flight. The only variables that need to be considered in Collin's algorithm are the aircraft configuration, weight, and path profile. The true airspeed, altitude and time are employed to describe the aircraft path profile. In his model, the

total path profile is divided into increments of altitude for every 2000 feet during climb and descent phase or 200 seconds of time increment during the cruise phase. The fuel estimation accuracy of the algorithm has been verified using cockpit data and air traffic control en-route data and terminal area radar data.

Wing-Ho and Trani developed a method to estimate fuel consumption using an artificial neural network (Wing-Ho, 2004). The model was developed using the data extracted from aircraft performance manual. The technique of neural network was introduced in their work and it could be trained to estimate fuel consumption of example aircraft. In the model, the aircraft fuel consumption data was obtained from flight performance manual of individual aircraft is imported into the neural network. The network was trained using LMQ algorithms and the output of the model was fuel flow. The data used in the artificial neural network model was applicable to the Fokker 100 equipped with Rolls-Royce Tay 650 engines and the SAAB 2000 turboprop aircraft.

More recently, Chatterji proposed a way of fuel burn estimation using real track data (Chatterji, 2011). In his work, the BADA fuel consumption model is applied to determine the fuel flow rate. It is developed in Chatterji's model that the variables of altitude, airspeed and thrust are required to estimate the fuel consumption. Airspeed is determined by latitude, longitude and altitude which can be expressed as a function of time. The latitude, longitude, altitude is extracted from flight trajectory data. Drag and lift which depends on estimated aircraft and wind states and weight are calculated to estimate the aircraft engine thrust. Once the thrust, altitude and airspeed are available, fuel flow rate can be estimated using BADA model. The model is validated by comparing the result to the actual data from Flight Data Recorder (FDR). However, due to the limit of aircraft types in BADA, the author compares the modeled aircraft of Bombardier Global

5000 to a Bombardier RJ-900 Regional Jet which can lead to some inconsistencies in characteristics of aircraft performance.

Senzig presented a model for estimating terminal area airplane fuel consumption which is integrated in FAA aviation environmental tool – the Aviation Environmental Design Tool (AEDT) (Senzig, Fleming and Iovinelli, 2009). In this model, a thrust specific fuel consumption algorithm has been developed as follows:

$$TSFC/\sqrt{\theta} = K_1 + K_2 M + K_3 h_{MSL} + K_4 F/\delta, \qquad (1)$$

where θ is the temperature ratio, M is the Mach number, h_{MSL} is the height above mean sea level, F/δ is the net corrected thrust, δ is the pressure ratio. The coefficients K_i are determined for individual aircraft types. Similarly, the arrival TSFC algorithm is expressed as

$$TSFC/\sqrt{\theta} = \alpha + \beta_1 M + \beta_2 e^{-\beta_3 (F/\delta/F_0)} \qquad (2)$$

in the model, α is the arrival thrust specific fuel consumption constant coefficient, β_1 is the arrival thrust specific fuel consumption Mach number coefficient, β_2 is the arrival thrust specific fuel consumption thrust term coefficient. β_3 is the arrival thrust specific fuel consumption thrust coefficient, F_0 is the static thrust at sea level standard conditions. A major disadvantage of this model is that it requires data from the airplane manufacturer which limit the general application of the model to many aircraft types.

2.2.2 Emission Model

Several models and calculators have been developed to estimate emissions. One such model is the ICAO Carbon Emissions Calculator. The calculator employs a distance-based approach to

estimate fuel consumption first and then compute the carbon emission using a multiplicative constant of 3.157, which represents the amount of CO_2 produced per ton of aviation fuel on combustion (ICAO, 2010). In this model, a great circle distance is assumed for the flight distance between the origin and destination airport. The fuel burn rate per kilometer is obtained from the CORINAIR database. Many of the coefficients, factors and parameters are from databases maintained by ICAO.

The ICAO Carbon Emissions Calculator only estimates the emission of greenhouse gas CO_2. Other efforts have been done to estimate other emissions inside the Landing/Take-off (LTO) cycle (ICAO, 2008). The FAA Emissions and Dispersion Modeling System (EDMS) is a model that is capable of estimating emissions like CO, HC, NO_X and SO_X for a given aircraft inside the LTO cycle (FAA, 2010; CSSI, 2010). In EDMS, the aircraft activities are classified into six modes, approach, taxi-in, startup, taxi-out, takeoff and climb out. For each of the modes, an emission factor is associated with a combination of aircraft type and engine. The emissions are estimated based on the calculation of fuel consumption in the model and the emissions rate database in the model.

2.3 Base of Aircraft Data (BADA)

The Base of Aircraft Data (BADA) is an Aircraft Performance Model (APM) developed and maintained by EUROCONTROL through active cooperation with aircraft manufactures and operating airlines (EUROCONTROL, 2011). Some of the capabilities of BADA are designed to calculate aircraft trajectory simulations and predictions as well as to better plan traffic flows, reduce delays, operating costs, and minimize adverse environmental impact and environmental studies.

BADA comprises of two components: model specifications and datasets. Model specifications are the basic aerodynamic equations that characterize the motions of an aircraft in flight. BADA uses a total energy model. The datasets contain model coefficients associated with each aircraft. The model specifications apply to 90% of the current aircraft types operating in the European Civil Aviation Conference (ECAC) airspace.

BADA provides three different kinds of datasets for each modeled aircraft type. The Operations Performance Files (OPF) contains the thrust, drag and fuel coefficients with information on aircraft weights, speeds and maximum altitude for the specified aircraft type. The Airline Performance File (APF) presents a default operational climb, cruise and descent speed schedule which is normally flown by airlines. The Performance Tables File (PTF) provides the nominal performance of the modeled aircraft in the form of a look-up table. It enables the user to obtain the aircraft average performance data directly without implementing the BADA Total Energy Model.

Many aircraft types can be mapped to the 117 BADA aircraft types (Version 3.9) enabling BADA to evaluate up to 339 different aircraft types.

3. METHODOLOGY

3.1 Introduction

This chapter describes the methodology used to estimate fuel and emissions. The application of the BADA model for generating flight profiles is explained in this chapter. The modeling of wind states, terminal area detours, airport taxi time and cruise altitude assignment are explained in this chapter.

3.2 Model Structure

The model developed includes several modules and nested procedures. A flight profile is modeled using equations and coefficients from the BADA database. Taxi fuel burn is estimated by analyzing real airport taxi times contained in the FAA Airport Performance Metrics (APM) Database. Operational conditions are considered in the model such as wind states, terminal area detour effect, cruise altitude assignment, and airport elevation. Figure 4 shows the structure of the model. The required model inputs are Origin-Destination (OD) pair information and the aircraft type. The model treats each unique OD pair and aircraft type combination as one flight. The model input can have hundreds of thousands flights in the form of a daily or yearly flight schedule. The flight demand data is obtained from the TSAM's mode choice model output.

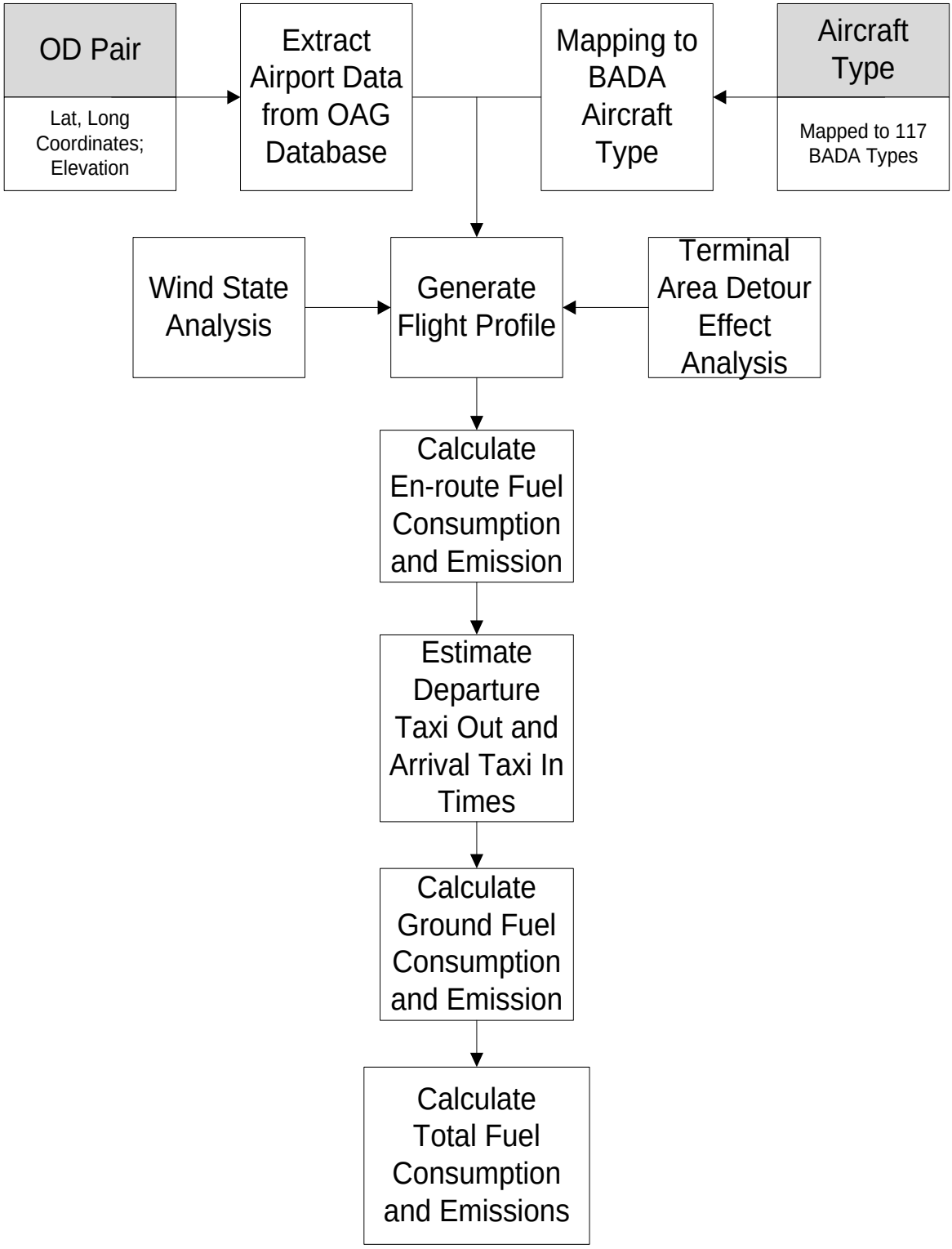


Figure 4. Structure And Components Of TSAM Fuel And Emissions Model.

3.3 Application of BADA Model

3.3.1 Introduction

In order to estimate the fuel consumption, we estimate the fuel flow rate for all flights conditions. Fuel flow rate is a very dynamic parameter. Fuel flow rate is a function of atmospheric parameters, speed and weight.

The BADA model provides a parameter called Thrust Specific Fuel Consumption (η [kg/(min·KN)]) that helps calculate the fuel flow rate at different flying conditions. The model is specified as a function of the true airspeed (TAS) for jet and turboprop engines. The BADA model provides typical profiles for true airspeed.

3.3.2 BADA Thrust Specific Fuel Consumption Model

BADA specifies the thrust specific fuel consumption (η) as a function of the TAS for jet and turboprop engines as shown in equations 3 – 4 (EUROCONTROL, 2011).

Jet Aircraft: $$\eta = C_{f1} \times (1 + \frac{V_{TAS}}{C_{f2}}) \quad (3)$$

Turboprop Aircraft: $$\eta = C_{f1} \times \left(1 - \frac{V_{TAS}}{C_{f2}}\right) \times (\frac{\mathrm{V_{TAS}}}{1000}) \quad (4)$$

Where C_{f1} and C_{f2} are model constants fuel flow factors, V_{TAS} is the true airspeed and η is the thrust specific fuel consumption.

The true airspeed can be calculated by correcting calibrated airspeed (CAS) by density and compressibility effects. BADA provides airline procedure models and parameters in the Airlines

Procedures File (APF) to calculate the aircraft calibrated airspeed in climb, cruise and descent phases. All the fuel flow factors C_f can be obtained from the BADA Operational File (OPF).

The nominal fuel flow rate f_{nom}, is applicable to all phases of flight except in idle descent and cruise. f_{nom} can then be calculated using the thrust Thr for jet and turboprop engine in Equation 5. For piston engines, f_{nom} is specified as a constant (Equation 6) (EUROCONTROL, 2011).

Jet/Turboprop Aircraft: $$f_{nom} = \eta \times Thr \quad (5)$$

Piston Aircraft: $$f_{norm} = C_{f1} \quad (6)$$

In the descent phase, idle thrust conditions are assumed and the minimum fuel flow rate f_{min} is the function of geopotential pressure altitude H_p [ft] for jet and turboprop engines (Equation 7). For piston aircraft, an additional constant is used as shown in Equation 8 (EUROCONTROL, 2011).

Jet/Turboprop Aircraft: $$f_{min} = C_{f3}(1 - \frac{H_p}{C_{f4}}) \quad (7)$$

Piston Aircraft: $$f_{min} = C_{f3} \quad (8)$$

The cruise fuel flow in the BADA model is calculated using the thrust specific fuel consumption η, the thrust Thr, and a cruise fuel flow factor $C_{f\mathrm{cr}}$ for jet and turboprop engine aircraft (Equation 9). For piston engine, the cruise fuel flow is obtained by multiplying the constant to the cruise fuel flow factor which is shown in equation (10) (EUROCONTROL, 2011).

Jet/Turboprop: $$f_{cr} = \eta \times Thr \times C_{fcr} \quad (9)$$

Piston: $$f_{cr} = C_{f1} \times C_{fcr} \quad (10)$$

Engine thrust is calculated using the coefficients and equations provided by BADA for three different thrust levels: maximum climb and take-off, maximum cruise, and descent.

The model assumes that an aircraft climbs and descents using standard procedures. Additionally, since it is impossible to know the takeoff weight for future flights, nominal mass conditions in the BADA Performance Table File (PTF) are employed.

3.3.3 BADA Airline Procedures Model

As stated in the previous section, calibrated airspeed is required to calculate the true airspeed. BADA provides an Airline Procedures Model to estimate CAS for three separate flight phases: climb, cruise and descent. For each of these phases and each aircraft model, the Airline Procedure Model requires the following information to determine the aircraft speed schedule.

I. BADA airline procedure default speeds which can be found in BADA Airline Procedure File (APF).

   Three values are obtained from APF. V_1 is the standard value of CAS [knots] below 10000 feet; V_2 is the standard value of CAS between 10000 feet and Mach transition altitude; and M is the standard Mach number above the Mach transition altitude.

   Table 1 shows the value of these variables for Airbus A320 in different phases of flight extracted from BADA APF (EUROCONTROL, 2011).

Table 1. Parameters Of BADA Airline Procedure Model For Airbus A320.

	Climb			Cruise			Descent		
	Below 10k ft	Between 10K ft and Mach transition altitude		Below 10k ft	Between 10K ft and Mach transition altitude		Below 10k ft	Between 10K ft and Mach transition altitude	
MASS	CAS	CAS	MACH	CAS	CAS	MACH	CAS	CAS	MACH
Low	310	310	0.78	250	310	0.78	300	300	0.79
Average	310	310	0.78	250	310	0.78	300	300	0.79
High	310	310	0.78	250	310	0.78	300	300	0.79

Calculation of Mach transition altitude is described in Section 3.3.4.2.

II. Stall speeds for take-off and landing configurations are found in the BADA Operations Performance File (OPF).

III. Minimum speed coefficients and speed schedules are also included.

Two minimum speed coefficients are specified in the user manual of the BADA Model. One is the minimum speed coefficient for take-off ($C_{Vmin,TO}$), the value of which is 1.2. The other is the minimum speed coefficient for all other phases which is set to a value of 1.3.

The BADA user manual states that "the speed schedules applicable below FL100 for climb and descent are based on a factored stall speed plus increment valid for a specified altitude range." Table 2 lists the values of speed increments used in the model for the climb and descent profile (EUROCONTROL, 2011).

Table 2. Incremental Variables For Speed Schedules.

Climb			Descent		
Variable Name	Description	Value [Kts]	Variable Name	Description	Value [Kts]
$Vd_{CL,1}$	Climb Speed Increment below 1500 ft (jet)	5	$Vd_{DES,1}$	Descent Speed Increment below 1000 ft (jet/turboprop)	5
$Vd_{CL,2}$	Climb Speed Increment below 3000 ft (jet)	10	$Vd_{DES,2}$	Descent Speed Increment below 1500 ft (jet/turboprop)	10
$Vd_{CL,3}$	Climb Speed Increment below 4000 ft (jet)	30	$Vd_{DES,3}$	Descent Speed Increment below 2000 ft (jet/turboprop)	20
$Vd_{CL,4}$	Climb Speed Increment below 5000 ft (jet)	60	$Vd_{DES,4}$	Descent Speed Increment below 3000 ft (jet/turboprop)	50
$Vd_{CL,5}$	Climb Speed Increment below 6000 ft (jet)	80	$Vd_{DES,5}$	Descent Speed Increment below 500 ft (piston)	5
$Vd_{CL,6}$	Climb Speed Increment below 500 ft (turbo/piston)	20	$Vd_{DES,6}$	Descent Speed Increment below 1000 ft (piston)	10
$Vd_{CL,7}$	Climb Speed Increment below 1000 ft (turbo/piston)	30	$Vd_{DES,7}$	Descent Speed Increment below 1500 ft (piston)	20
$Vd_{CL,8}$	Climb Speed Increment below 1500 ft (turbo/piston)	35			

Having obtained these parameters, the speed schedule for climb, cruise and descent can then be calculated using the Equations 11-48 (EUROCONTROL, 2011):

- **CLIMB:**

  For jet aircraft:

From 0 to 1499 ft: $C_{V_{min}} \cdot (V_{stall})_{TO} + V_{d_{CL,1}}$ (11)

From 1500 to 2999 ft: $C_{V_{min}} \cdot (V_{stall})_{TO} + V_{d_{CL,2}}$ (12)

From 3000 to 3999 ft: $C_{V_{min}} \cdot (V_{stall})_{TO} + V_{d_{CL,3}}$ (13)

From 4000 to 4999 ft: $C_{V_{min}} \cdot (V_{stall})_{TO} + V_{d_{CL,4}}$ (14)

From 5000 ft to 5999 ft: $C_{V_{min}} \cdot (V_{stall})_{TO} + V_{d_{CL,5}}$ (15)

From 6000 ft to 9999 ft: $\min(V_{cl,1}, 250\ kt)$ (16)

From 10000 ft to Mach transition altitude: $V_{cl,2}$ (17)

Above Mach transition altitude: M_{cl} (18)

For turboprop and piston aircraft:

From 0 to 499 ft:	$C_{Vmin} \cdot (V_{stall})_{TO} + V_{d_{CL,6}}$	(19)
From 500 to 999 ft:	$C_{Vmin} \cdot (V_{stall})_{TO} + V_{d_{CL,7}}$	(20)
From 1000 to 1499 ft:	$C_{Vmin} \cdot (V_{stall})_{TO} + V_{d_{CL,8}}$	(21)
From 1500 to 9999 ft:	$\min(V_{cl,1}, 250 \text{ kt})$	(22)
From 10000 ft to Mach transition altitude:	$V_{cl,2}$	(23)
Above Mach transition altitude:	M_{cl}	(24)

- **CRUISE**

For jet aircraft:

From 0 to 2999 ft:	$min(V_{cr,1}, 170\ kt)$	(25)
From 3000 to 5999 ft:	$min(V_{cr,1}, 220\ kt)$	(26)
From 6000 to 13999 ft:	$min(V_{cr,1}, 250kt)$	(27)
From 14000 ft to Mach transition altitude:	$V_{cr,2}$	(28)
Above Mach transition altitude:	M_{cr}	(29)

For turboprop and piston aircraft:

From 0 to 2999 ft:	$min(V_{cr,1}, 150\ kt)$	(30)
From 3000 to 5999 ft:	$min(V_{cr,1}, 180\ kt)$	(31)
From 6000 to 9999 ft:	$min(V_{cr,1}, 250\ kt)$	(32)
From 10000 ft to Mach transition altitude:	$V_{cr,2}$	(33)
Above Mach transition altitude:	M_{cr}	(34)

- **DESCENT**

For jet and turboprop aircraft:

From 0 to 999 ft:	$C_{Vmin} * (V_{stall})_{LD} + Vd_{DES,1}$	(35)
From 1000 to 1499 ft:	$C_{Vmin} * (V_{stall})_{LD} + Vd_{DES,2}$	(36)
From 1500 to 1999 ft:	$C_{Vmin} * (V_{stall})_{LD} + Vd_{DES,3}$	(37)
From 2000 to 2999 ft:	$C_{Vmin} * (V_{stall})_{LD} + Vd_{DES,4}$	(38)
From 3000 ft to 5999 ft:	$min(V_{des,1}, 220\ kt)$	(39)
From 6000 ft to 9999 ft:	$min(V_{des,1}, 250\ kt)$	(40)
From 10000 ft to Mach transition altitude:	$V_{des,2}$	(41)
Above Mach transition altitude:	M_{des}	(42)

For piston aircraft:

From 0 to 499 ft:	$C_{Vmin} * (V_{stall})_{LD} + Vd_{DES,5}$	(43)
From 500 to 999 ft:	$C_{Vmin} * (V_{stall})_{LD} + Vd_{DES,6}$	(44)
From 1000 to 1499 ft:	$C_{Vmin} * (V_{stall})_{LD} + Vd_{DES,7}$	(45)
From 1500 to 9999 ft:	$V_{\text{des},1}$	(46)
From 10000 ft to Mach transition altitude:	$V_{\text{des},2}$	(47)
Above Mach transition altitude:	M_{des}	(48)

3.3.4 Atmospheric Model

Conversion from CAS to TAS and Mach number requires several atmospheric parameters which are a function of altitude.

3.3.4.1 International Standard Atmosphere (ISA)

The International Standard Atmosphere (ISA) published by the International Organization for Standardization (ISO) is an atmospheric model of how the pressure, temperature, density, and viscosity of the Earth's atmosphere change over a wide range of altitudes (EUROCONTROL, 2011).

Mean Sea Level (MSL) Standard atmospheric conditions are those that occur in the ISA model at the point where the geopotential pressure altitude H_p is zero. Sea level ISA values are:

Standard atmospheric temperature at MSL: $T_0 = 288.15$ [K]
Standard atmospheric pressure at MSL: $p_0 = 101325$ [Pa]
Standard atmospheric density at MSL: $\rho_0 = 1.225$ [kg/m^3]
Speed of sound: $a_0 = 340.294$ [m/3]

Non-ISA atmospheric conditions follow the same model as the ISA atmosphere but differ in temperature and pressure. ΔT represents the difference in atmospheric temperature at MSL between a given non-standard atmosphere and ISA conditions. Similarly, Δp represents the

difference in atmospheric pressure at MSL between a given non-standard atmosphere and ISA conditions.

3.3.4.2 ISA and Non-ISA Conversion

Some physical constants are defined in the conversion of atmospheric parameters between ISA and non-ISA, they are:

Adiabatic index of air:	$\kappa = 1.4$
Real gas constant for air:	$R = 287.05287\ [m^2/(K \cdot s^2)]$
Gravitational acceleration:	$g_0 = 9.80665\ [m/s^2]$
ISA temperature gradient with altitude below the tropopause:	$\beta_{T,<} = -0.0065\ [K/m]$

The subindex < denotes values below the tropopause and > denotes values above the tropopause which is a layer located at a geopotential pressure altitude, $H_{p,trop}$ of 11000 meters.

With these parameters, the temperature can be determined using the following equations:

$$T_< = T_0 + \Delta T + \beta_{T,<} H_{p,<} \quad (49)$$

$$T_{ISA,trop} = T_0 + \beta_{T,<} H_{p,trop} \quad (50)$$

$$T_{trop} = T_0 + \Delta T + \beta_{T,<} H_{p,trop} \quad (51)$$

$$T_> = T_{trop} \quad (52)$$

The air pressure can be determined using the following equations:

$$p_< = p_0 \left(\frac{T_< - \Delta T}{T_0}\right)^{\frac{g_0}{\beta_{T,<} R}} \quad (53)$$

$$p_{trop} = p_0 \left(\frac{T_{trop} - \Delta T}{T_0}\right)^{\frac{g_0}{\beta_{T,<} R}} \quad (54)$$

$$p_> = p_{trop} \exp\left[-\frac{g_0}{R T_{ISA,trop}} (H_{p,>} - H_{p,trop})\right] \quad (55)$$

Air density can be calculated using the pressure and the temperature at a given altitude:

$$\rho = \frac{p}{RT} \quad (56)$$

The speed of sound can be determined using the following expression:

$$a = \sqrt{\kappa RT} \tag{57}$$

Having the equations above, CAS can be converted to TAS using the standard Equation:

$$V_{TAS} = [\frac{2}{\mu}\frac{p}{\rho}\{\left(1 + \frac{p_0}{p}[\left(1 + \frac{\mu}{2}\frac{\rho_0}{p_0}{V_{CAS}}^2\right)^{\frac{1}{\mu}} - 1]\right)^{\mu} - 1\}]^{\frac{1}{2}} \tag{58}$$

$$\mu = \frac{\kappa - 1}{\kappa} \tag{59}$$

Mach number can be obtained from TAS using Equation 60:

$$V_{TAS} = M \times \sqrt{\kappa RT} \tag{60}$$

The Mach transition altitude is defined as the geopotential pressure altitude at which CAS and M represent the same TAS value, and can be calculated as follows:

$$H_{p,trans} = \left(\frac{1000}{0.3048 \cdot 6.5}\right) \cdot [T_0 \cdot (1 - \theta_{trans})] \tag{61}$$

where θ_{trans} is the temperature ratio at the transition altitude,

$$\theta_{trans} = (\delta_{trans})^{-\frac{\beta_{T,<}R}{g_0}} \tag{62}$$

where δ_{trans} is the pressure ratio at the transition altitude,

$$\delta_{trans} = \frac{[1 + (\frac{\kappa - 1}{2})(\frac{V_{CAS}}{a_0})^2]^{\frac{\kappa}{\kappa - 1}} - 1}{[1 + \frac{\kappa - 1}{2}M^2]^{\frac{\kappa}{\kappa - 1}} - 1} \tag{63}$$

3.4 Flight Profile Generation

The model uses a numerical integration approach to generate flight profiles to compute the fuel consumption and emissions. The entire flight procedure is divided into increments of a variable time changes in all phases of climb, cruise and descent. TAS, fuel flow rate, thrust and aircraft weight are calculated in each iteration using the equations explained in the previous sections. The aircraft position state along the track and altitude are calculated incrementally considering the wind states.

In order to calculate the instantaneous parameters of altitude, distance and weight in each iteration, an Ordinary Differential Equation (ODE) solver is required. Traditional ODE15s solver takes about five seconds to complete this work which leads to an unacceptable model running time. A customized Runge Kutta 4^{th} order (RK4) ODE solver has been developed to handle this job. It takes only one second for the new solver to do the computation (Rye, 2010).

The RK4 solver takes inputs of derivatives of aircraft altitude, distance traveled, aircraft weight and time span, and outputs the instantaneous values of those parameters.

For calculating the aircraft ground speed and total distance traveled, the customized RK4 solver also runs to integrate the wind vector (Described in Section 3.7) into the aircraft true airspeed.

Before the generation of flight profiles, every flight is assigned a cruise altitude stochastically based on the distance flown and aircraft type. The climb profile is nested in flight profile generation when the altitude reaches the assigned cruise altitude in the calculation. The descent profile is created as a second step as a reversed climb profile using descent parameters and equations in the BADA model. The cruise profile is executed next for the distance left between the climb and descent profile.

The total fuel consumed in the flight is obtained by subtracting the final aircraft weight from the initial weight. Figure 5 demonstrates the flight profile of an Airbus A320 flying from Atlanta (ATL) to Los Angeles (LAX) generated by the model.

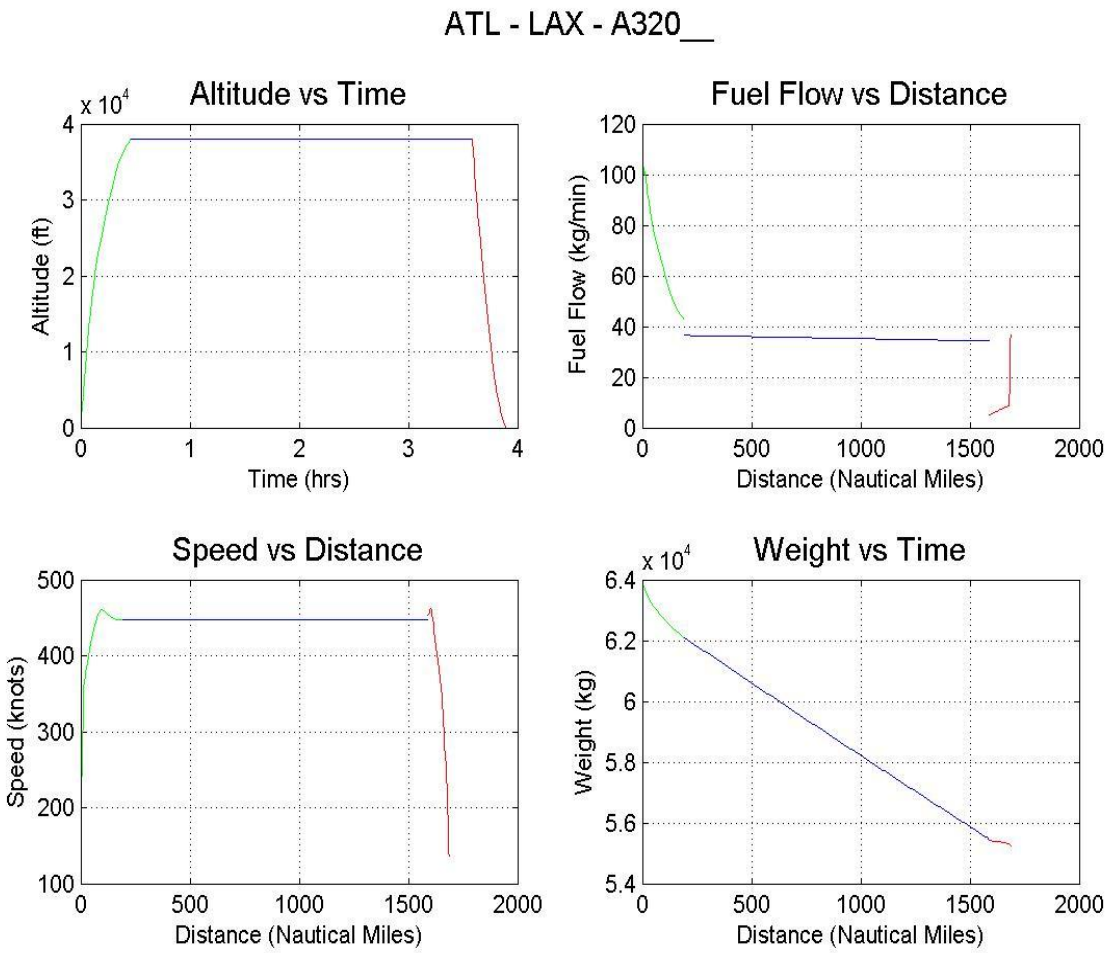


Figure 5. Flight Profile Of Airbus A320 From ATL To LAX.

3.5 Stochastic Assignment of Cruise Altitude

In the flight profile generation, each flight is assigned a cruise altitude stochastically. To prepare the randomness observed in cruise altitude assignment, we use the Enhanced Traffic Management System (ETMS) data as empirical reference. ETMS is a database of the Federal Aviation Administration (FAA) developed to monitor air traffic operations both real-time and off-line (FAA, 2012). The ETMS database contains flight waypoint information and cruise altitude. In the analysis of cruise altitude assignment, the ETMS data are sorted for each unique OD pair and aircraft type combination and segregated for different headings. The ETMS

database includes thousands of distinct aircraft. ETMS aircraft types are mapped to 117 BADA aircraft types included in version 3.9. Figure 6 shows a sample cruise altitude assignment trajectory for Airbus A320 extracted from the ETMS dataset.

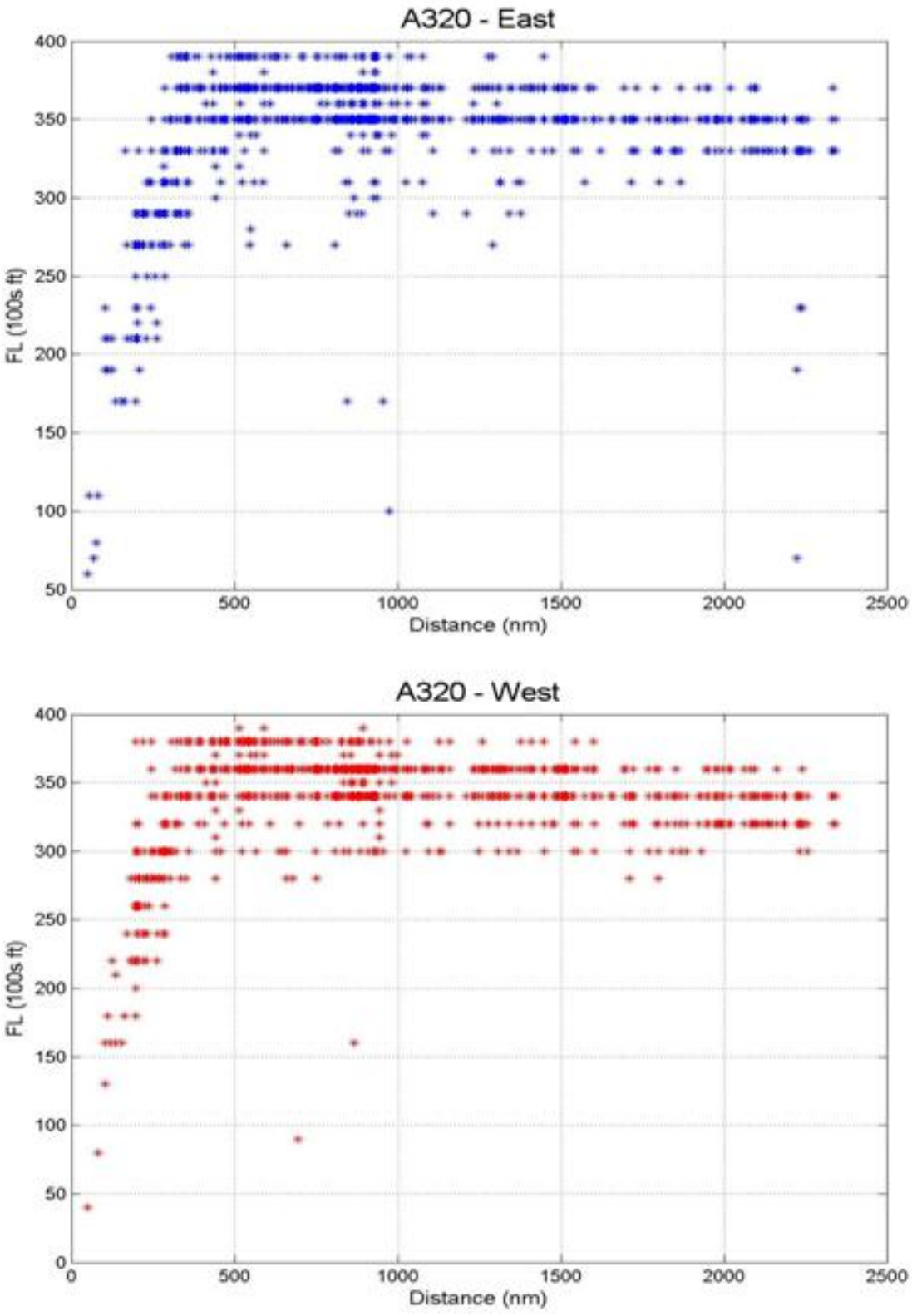


Figure 6. Airbus A320 ETMS Cruise Altitude Assignment.

It is observed that flights tend to fly at FL350 and FL370 when heading east and FL340 and FL360 when heading west for A320. Most of the flights that have a travel distance greater than 400 nautical miles flew above 24000 feet. A simulation technique employing an empirical cumulative distribution function is used in order to replicate these tendencies.

3.5.1 Crossover Point

A Crossover Point is observed in the plots of ETMS cruise altitude data in figure 7. The Crossover Point depicts the minimum flight distance when an aircraft can achieve the highest flight level in the ETMS data for a specified aircraft type. Figure 7 demonstrates the concept of Crossover Point in the plot.

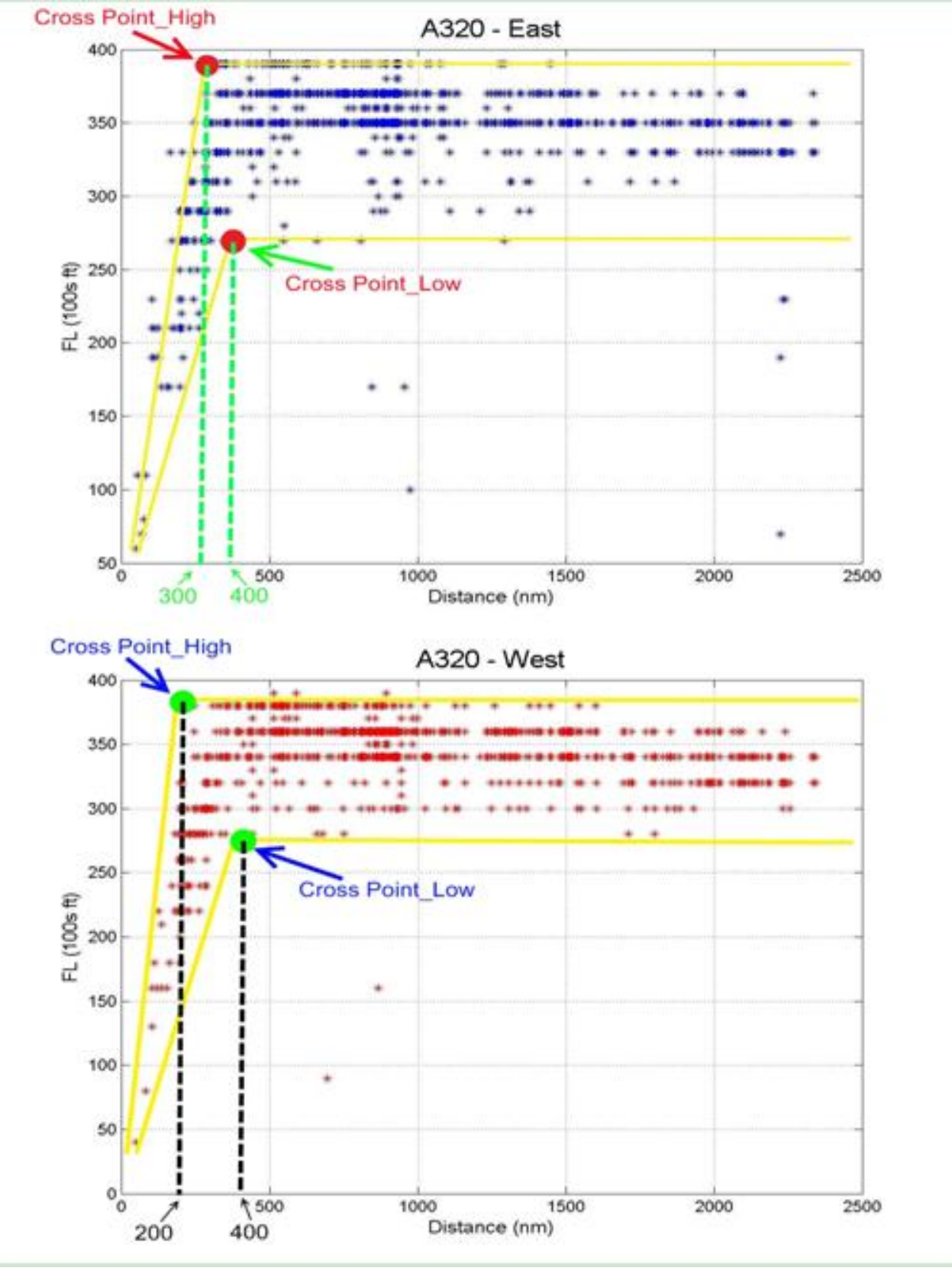


Figure 7. Crossover Point For Airbus A320.

3.5.2 Regression Analysis for the Boundary

The cruise altitudes corresponding to a flight distance show an upper and a lower boundary. The crossover point is accordingly split into two points denoted high and low crossover point. Having identified the crossover points for each BADA aircraft type, a linear regression analysis is performed to model the cruise altitude boundaries. Computationally, the distance flown is divided into several bins with size of 50 nautical miles each. In each bin, the highest and lowest altitudes are sorted out. The slope and intercept of upper bound and lower bound are calculated using the highest and lowest altitude in each bin. Figure 8 shows an example of highest and lowest altitude in a bin for an Airbus A320 heading east.

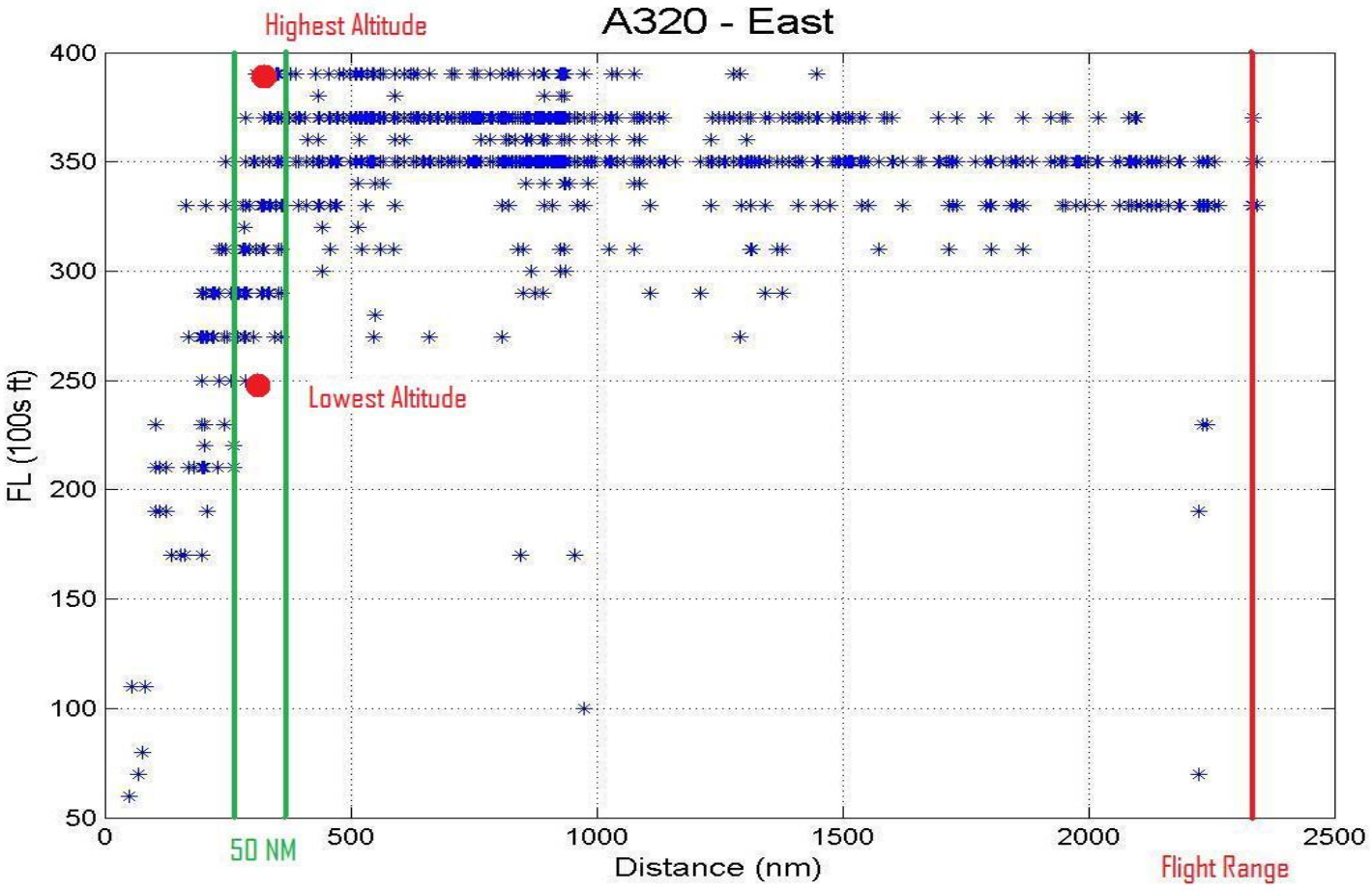


Figure 8. Highest And Lowest Altitudes In A Bin.

These upper and lower cruise altitude boundaries are used to generate a random cruise altitude for each flight in the model. This process is a Monte Carlo Simulation. It is assumed that the slope of the upper and lower bounds after the crossover point is 0. Figure 9 demonstrates the upper and lower cruise altitude boundaries for Airbus A320 heading east.

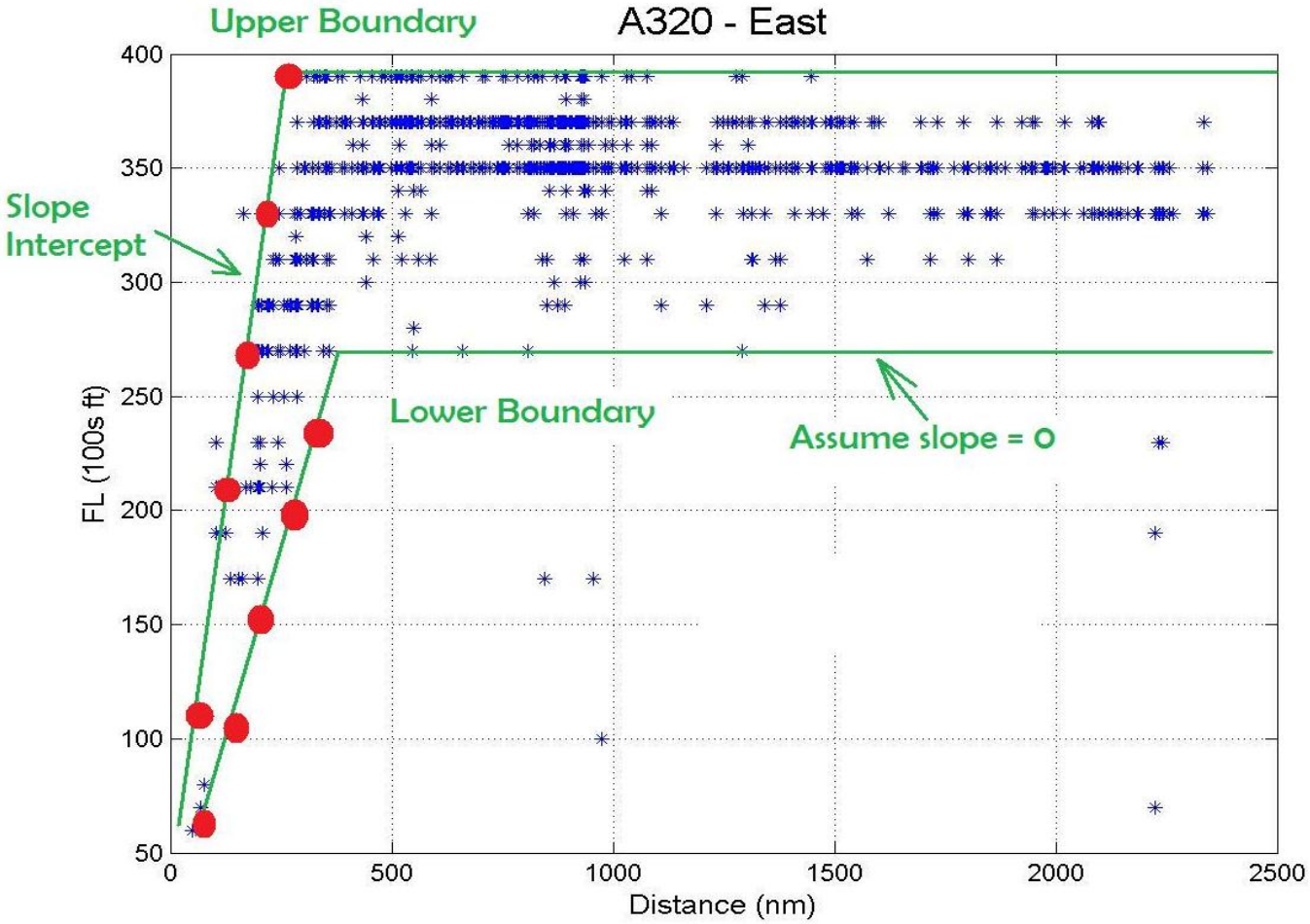


Figure 9. Upper Bound And Lower Bound For The Cruise Altitudes For Airbus A320.

3.5.3 Empirical CDF for Cruise Altitudes

The generation of stochastic cruise altitudes is done using an inverse generation method. Figure 10 shows that for every aircraft type, four distance ranges are used to generate a cruise altitude.

The cruise altitude in every bin can be generated using simulation techniques following an empirical cumulative distribution function (CDF) between the upper and lower bounds.

When assigning cruise altitude for the flights in the first section, a uniform distribution is used. A CDF is not required for first section. Figure 10 shows the cruise flight level assignment for Airbus A320 heading East.

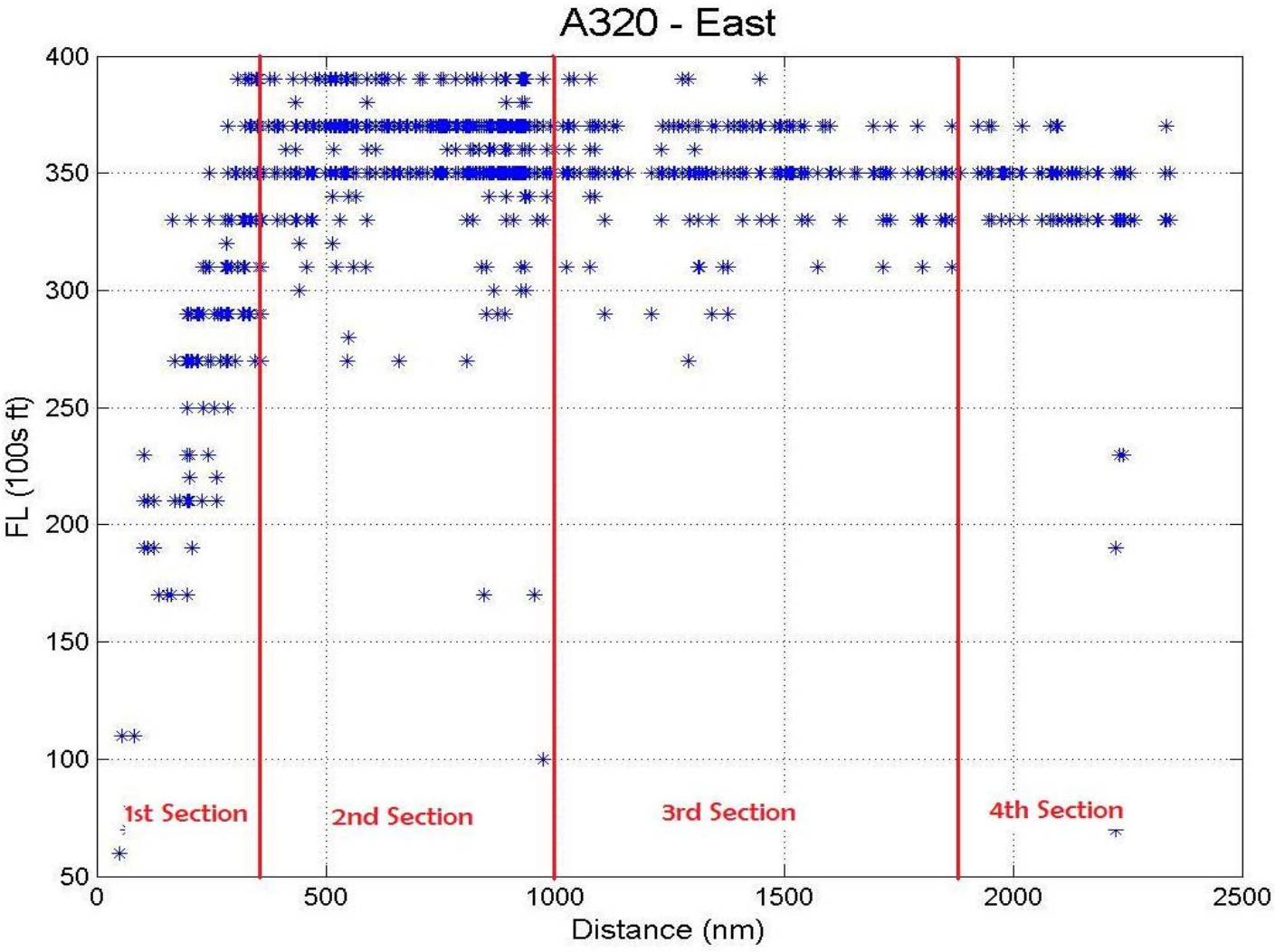


Figure 10. Sections Assignment For Airbus A320 Heading East.

The empirical CDF corresponding to the cruise altitude assignment for the Airbus A320 heading east is shown in Figure 11.

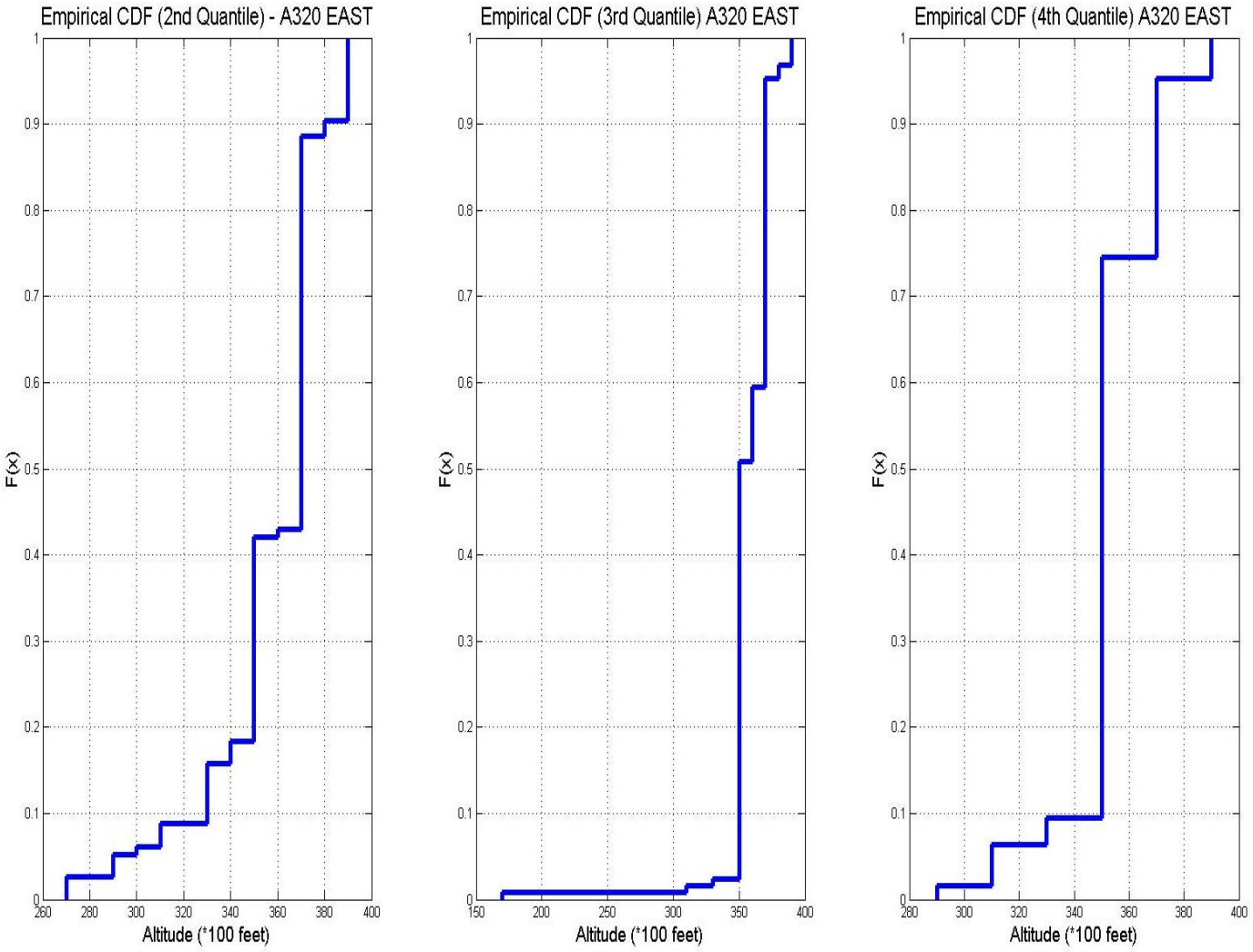


Figure 11. Empirical CDFs For Cruise Altitude Assignment Of Airbus A320 Heading East.

Figure 12 shows the stochastic assignment of cruise altitudes for Airbus A320 generated by the model.

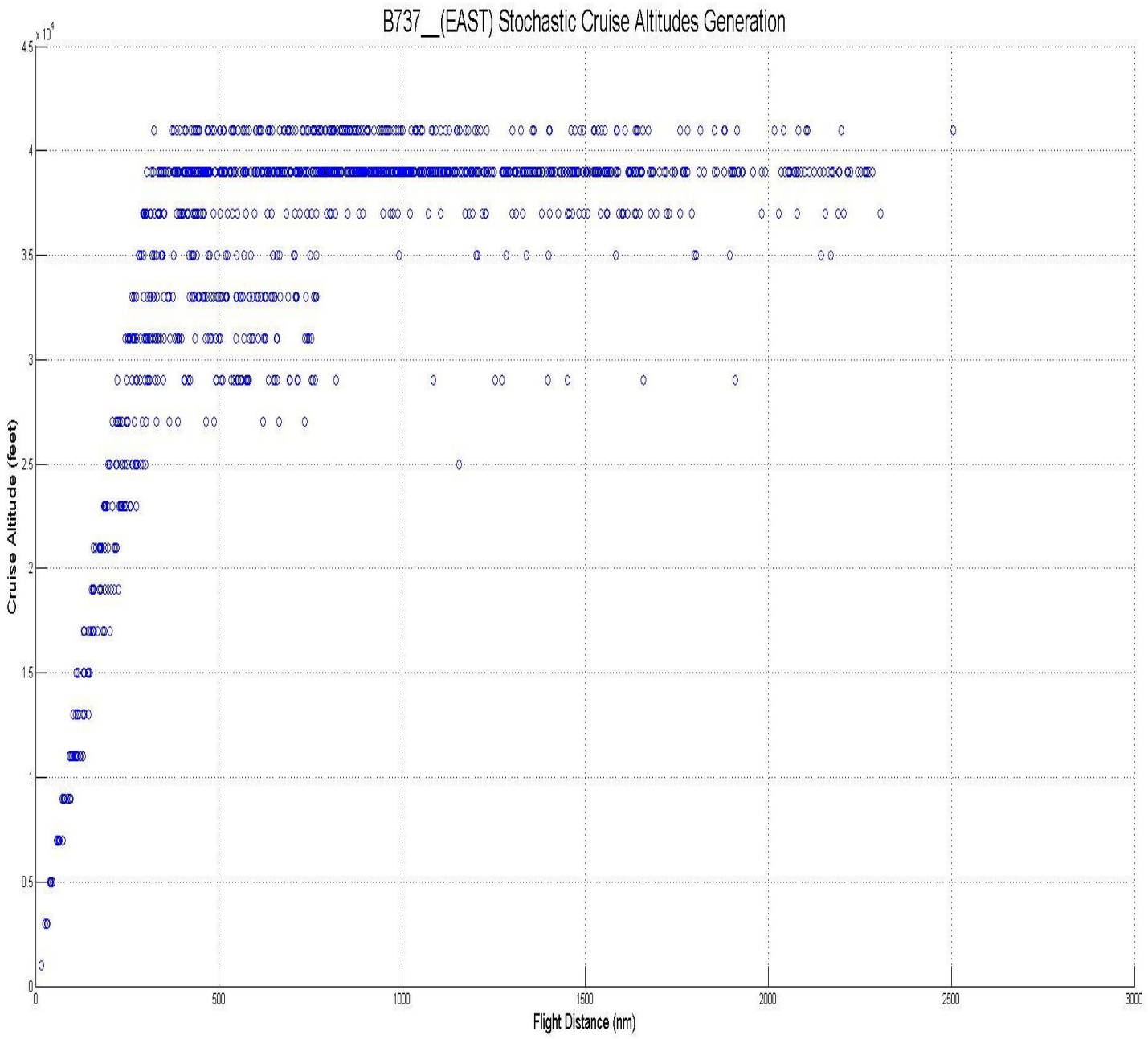


Figure 12. Cruise Altitude Assignment Of Airbus A320 Flying East Generated By The Monte Carlo Simulation.

3.6 Flight Track Waypoints

Most of the commercial flights fly along prescribed waypoints which are part of a flight plan. The waypoints determine not only the flight distance traveled but also the wind states that the aircraft encounters as it moves. Beside the cruise altitude, the ETMS data also contains waypoints for all of the commercial flights in the U.S. The ETMS data provides trajectory of the commercial flight in the flight profile generation. Figure 13 demonstrates the trajectory of a commercial flight flown by Airbus A320 from Atlanta to Los Angeles generated by the model using the ETMS waypoints data in a 3D view.

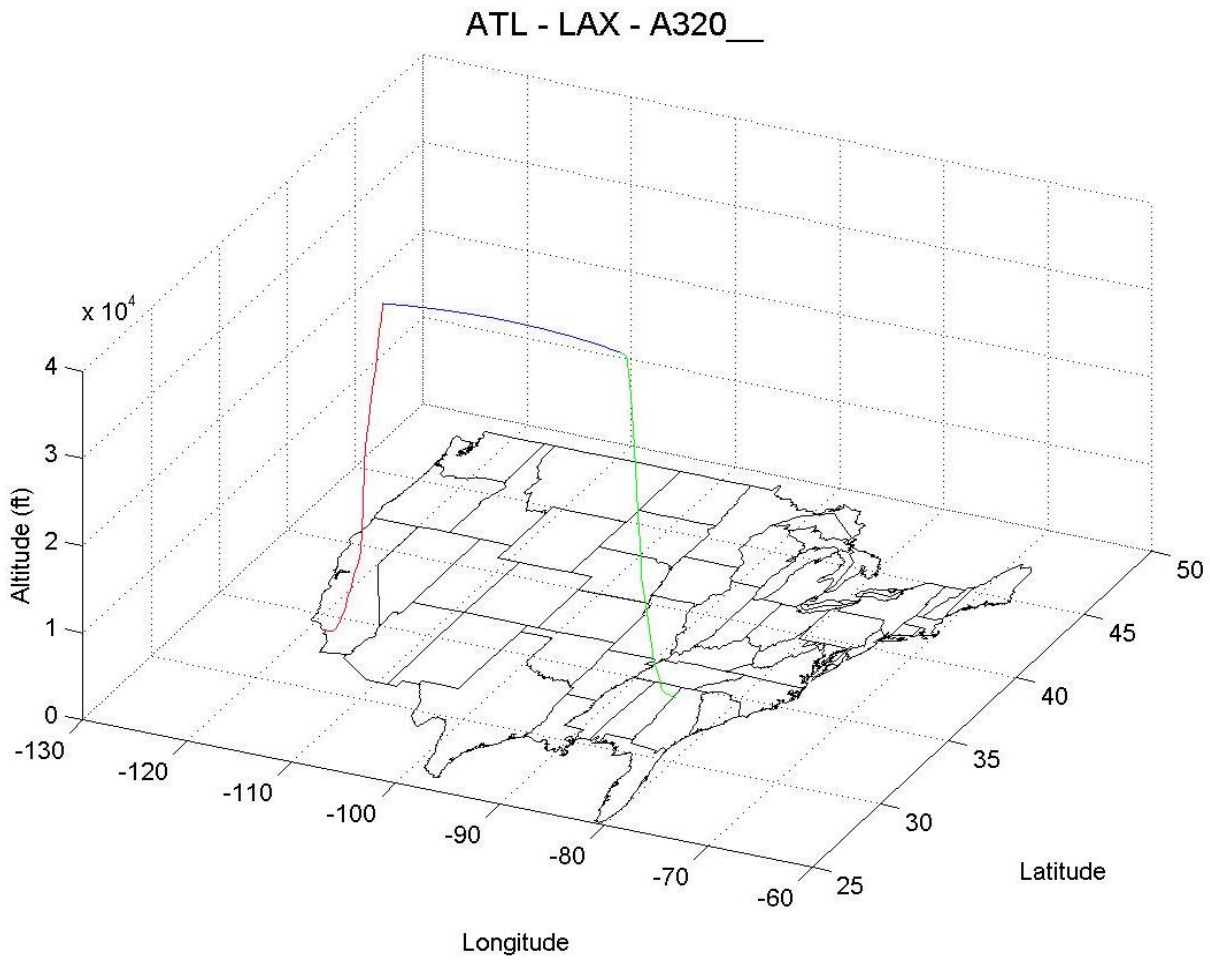


Figure 13. 3D View Of Airbus A320 Flies From ATL To LAX Generated By The Model.

3.7 Wind Effect Analysis

As the aircraft moves through different latitude and longitude coordinates along its route and climbs or descends, it passes through different wind vectors and flight levels. The states of wind affect the true airspeed, hence affecting fuel consumption. In the developed model, wind effect is analyzed at each coordinates corresponding to the time increment when generating the flight profile.

To incorporate wind effect into flight profile analysis, NCEP/NCAR Climate Dataset for wind is employed. The dataset provides wind vectors on 17 pressure levels above Continental US for every 2.5 degree of longitude and latitude coordinates (NOAA, ESRL and PSD 2012). Figure 14 demonstrates the annual average direction and speed of wind above continental U.S.

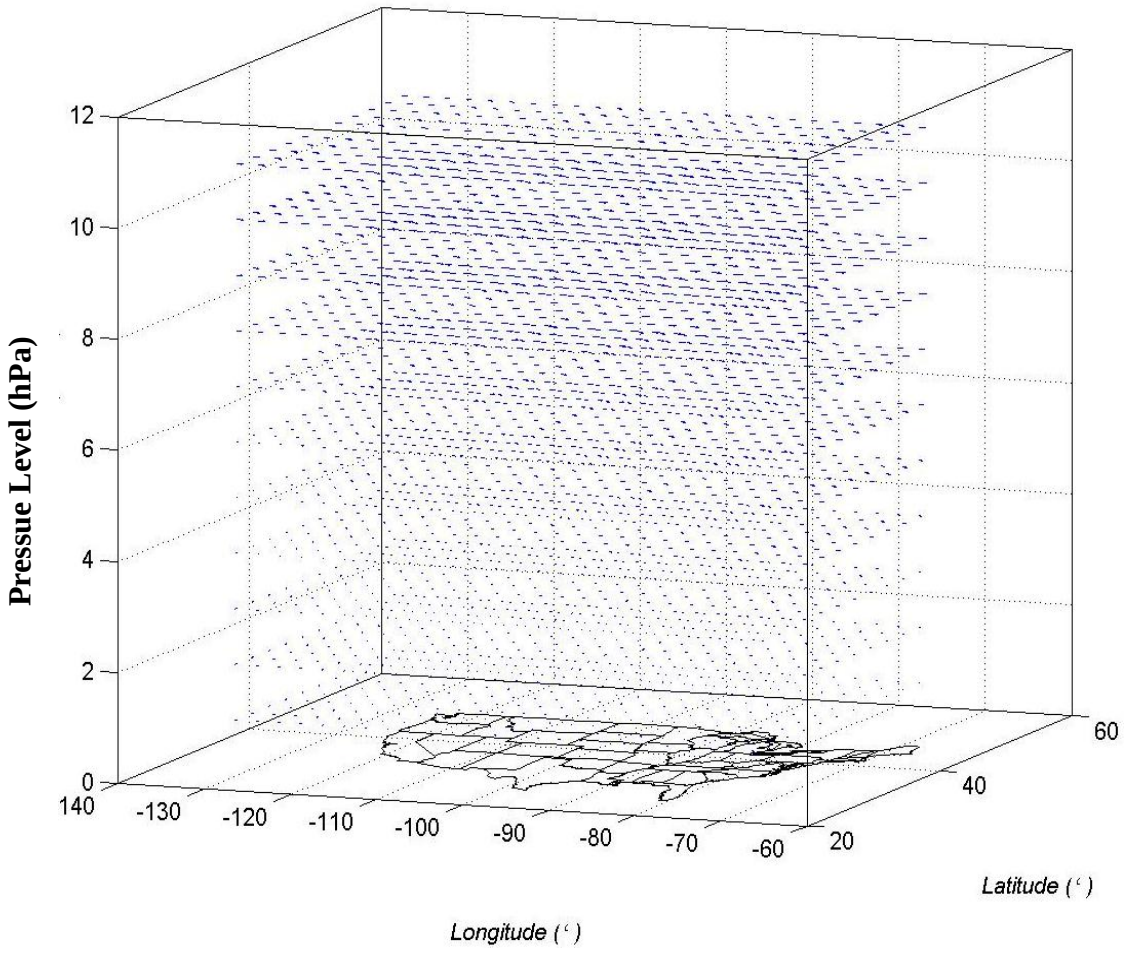


Figure 14. 3-D Plot Of Annual Average Wind Effect At Different Pressure Levels.

In the NCAR wind dataset, the U-wind component presents the East-West component of wind whereas the V-wind presents the North-South component of wind. The U-wind and V-wind components are combined in the analysis to determine the wind effects on the aircraft. Figure 15 shows the annual average wind vector at 40°N, -95°W at different altitudes.

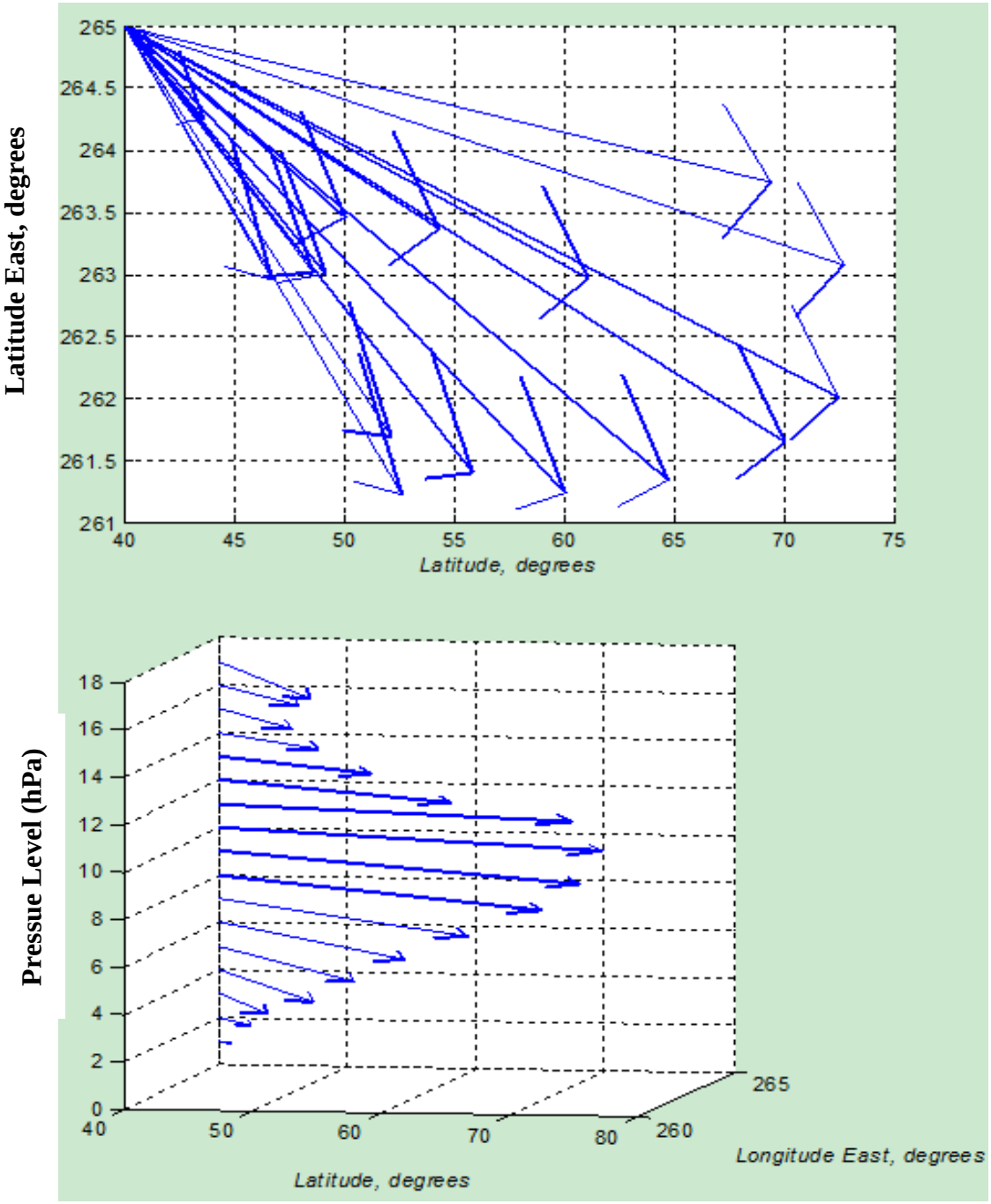


Figure 15. Wind Vector Plot At 40°N, -95°W At Different Altitudes.

Wind vector is projected on the speed vector of aircraft to determine the value of ground speed. Figure 16 illustrates the process for combining the wind and the aircraft speed vectors.

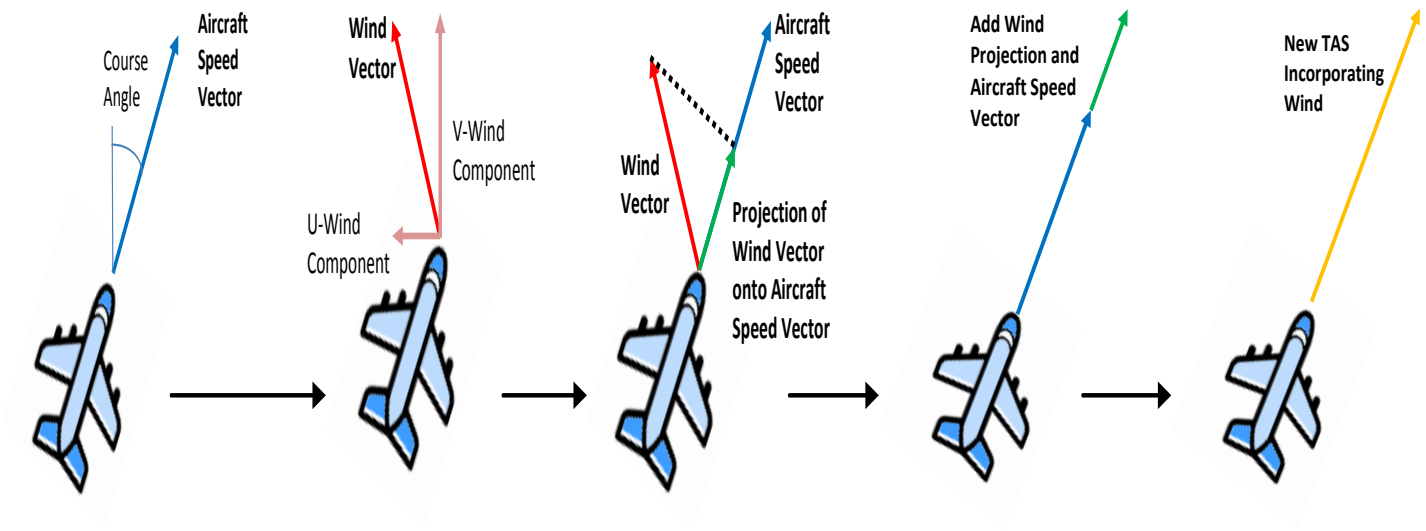


Figure 16. Process For Combining The Wind And Aircraft Speed Vectors.

When generating the flight profile, the aircraft position coordinates are adjusted in order to account the wind vector. Ground speed is used to estimate travel time. Figure 17 shows different flight profiles for an Airbus A320 flying from PHX to ATL with and without wind effects. Table 1 summarizes the differences of flight profiles under different wind conditions for the same flight.

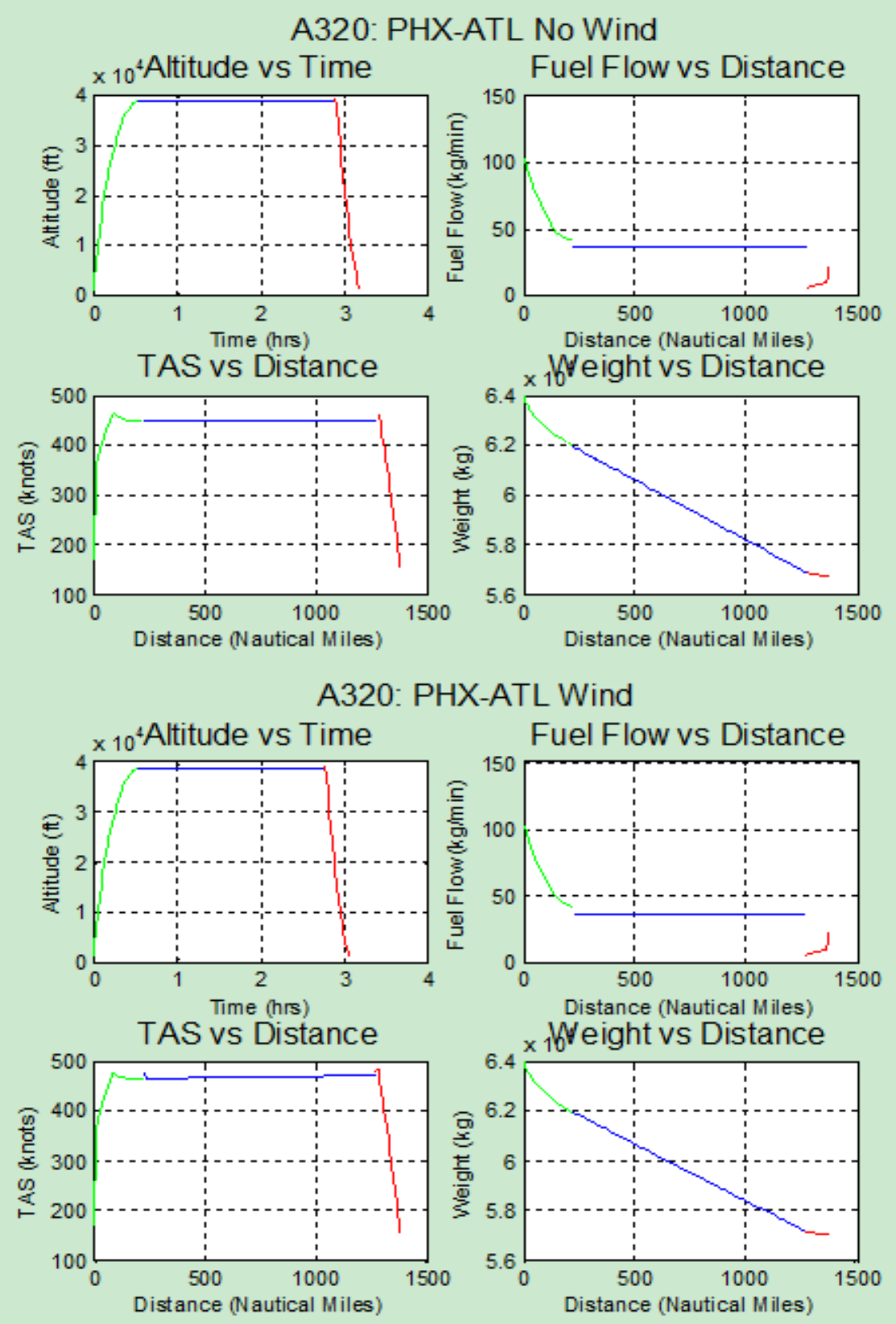


Figure 17. Different Flight Profiles Of A320 From PHX To ATL Under Wind And No-Wind Conditions.

Table 3. Different Flight Profiles Of A320 From PHX To ATL With And Without Wind Effect.

Wind Condition	Average Wind Projection	Average Ground Speed	Travel Time	Distance Travelled	Fuel Burn
No Wind	0 knots	433.2 knots	3:10 hrs	1376.7 nm	7267 kg
Tail Wind	19.19 knots	451.8 knots	3:02 hrs	1376.7nm	6980 kg

In Table 3 we observe that the projection of a tail wind vector increases the aircraft ground speed compared to the no wind condition. This reduces travel time and saves fuel. Figure 18 shows ground speed profiles for an Airbus A320 flying from Phoenix to Atlanta with and without wind for the same flight. The wind state is also shown for reference.

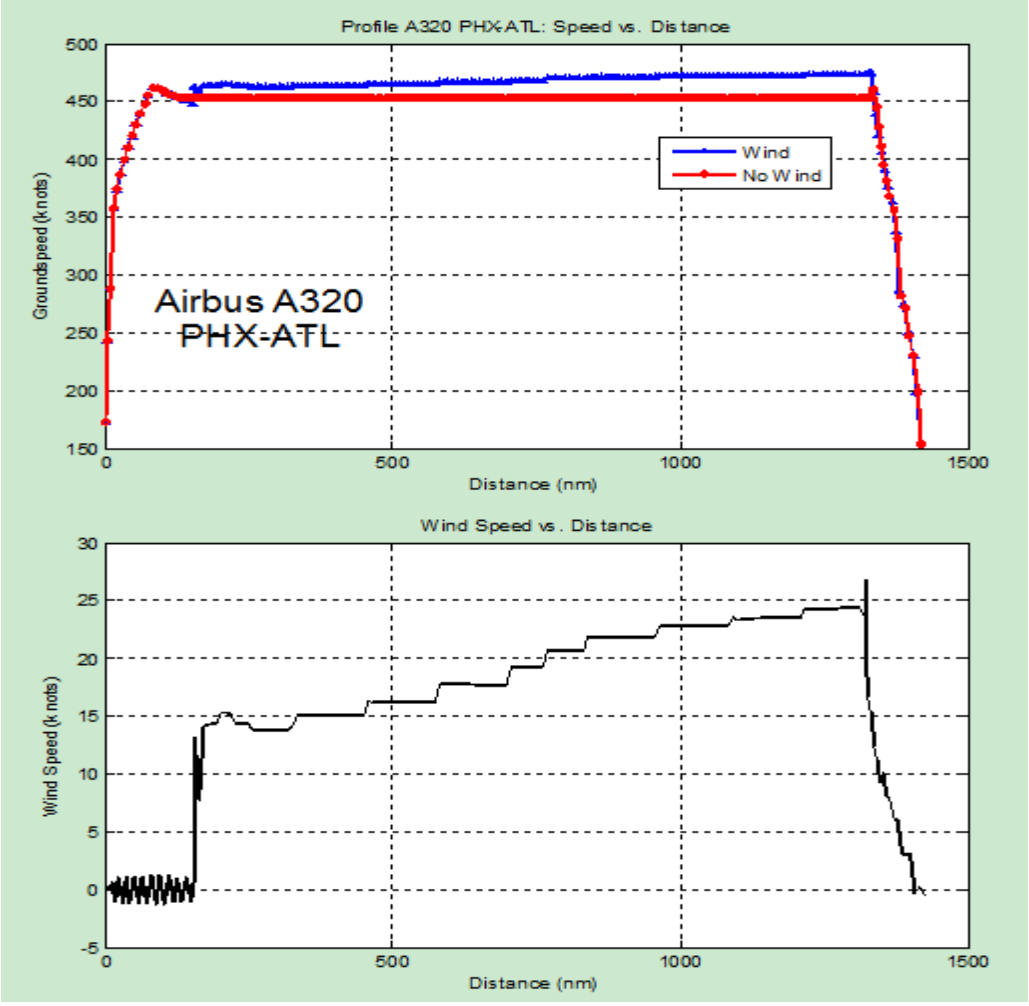


Figure 18. Ground And Wind Speeds Of An A320 Flying From PHX To ATL With And Without Wind.

3.8 Analysis of Terminal Area Detour Factor

Traffic congestion in the terminal area affects fuel consumption. Sometimes an aircraft may be instructed to "hold" at a pre-determined altitude or fly a detour for separation purposes. This is especially true when operating in regions of heavy demand such as Los Angles, Atlanta and New York. The extra travel time adds to the fuel burn due to the additional thrust utilized to keep the aircraft straight and leveled and due to more circuitous tracks flown.

The terminal area detour distance for a flight and its effect in fuel burn are estimated by analyzing historical radar data extracted from the Performance Data Analysis and Reporting System (PDARS) across various airports. PDARS is a comprehensive set of software tools for gathering aviation performance data to measure and baseline operations, helping design, implement, and evaluate operational improvements (Browder, Gutterrud and Schade, 2010). Figure 19 demonstrates the installation of PDARS system in the United States (Schroeder, 2011). When an aircraft is 200-300 feet Above Ground Level (AGL) in the terminal area, radar data is collected every 5 seconds. Three variables in PDARS data are of most interest to this modeling process: 1) the airport name and type of operation; the flight path data including latitude and longitude coordinates; and 3) the speed profile and aircraft type.

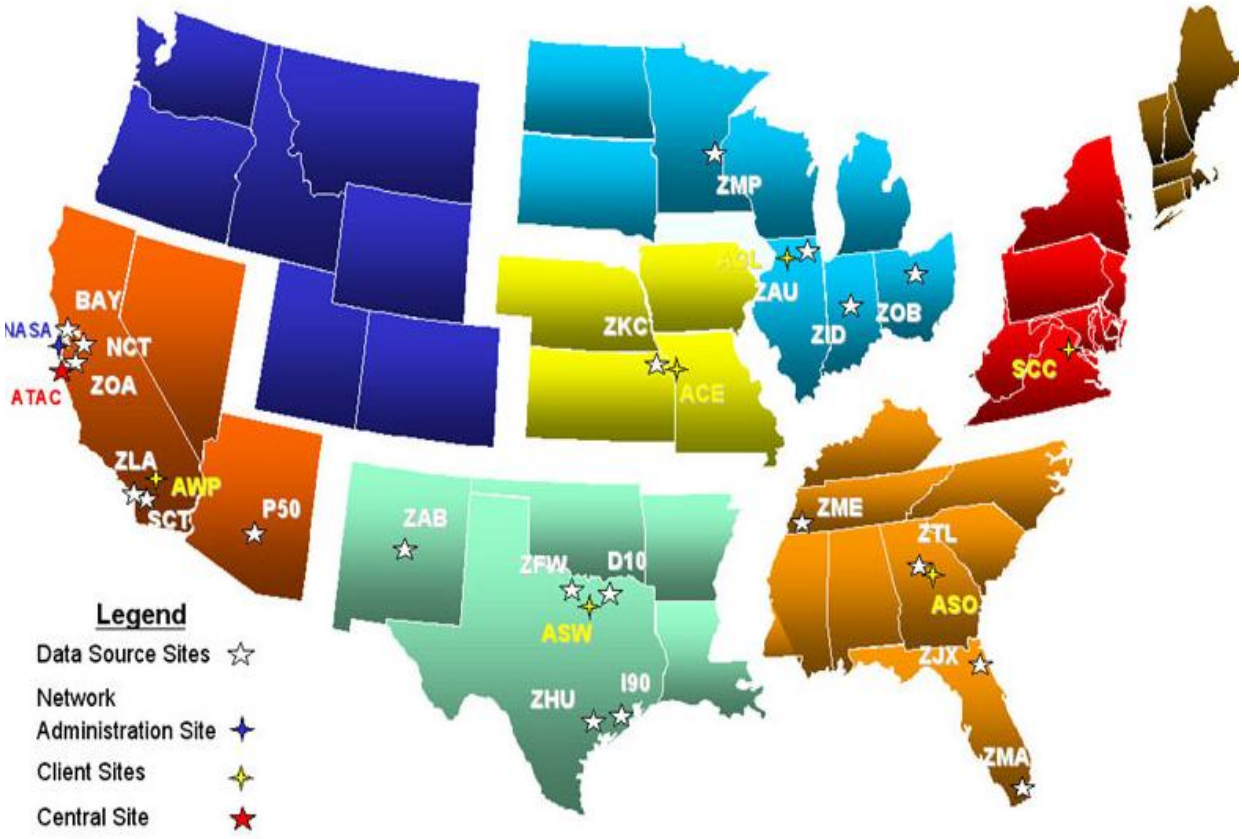


Figure 19. PDARS Installation In The U.S. As Of 2004 (Schroeder, 2011). Used Under Fair Use, 2012

A cluster analysis, based on departure and arrival frequency, proximity to other airports within same TRACON, and proximity to major hub airport, has been done for processing the PDARS data. The analysis allows detour factors to be matched from PDARS airports to OAG airports of similar type that are used in fuel consumption estimation. In order to calculate the detour factor for each individual PDARS track, the distance traveled within the TRACON according to the PDARS data was divided by the Great Circle Distance between the same starting and ending points as the PDARS data (Rye, 2011). Detour factors were then averaged for each airport (the only distinction made was for departure or arrival status). Airport characteristics were observed across all the PDARS airports to determine whether there was a correlation between these

characteristics and the airport detour factor. Figure 20 shows the comparisons between GCD tracks and PDARS arrival tracks at some airports and the corresponding analyzed detour factors.

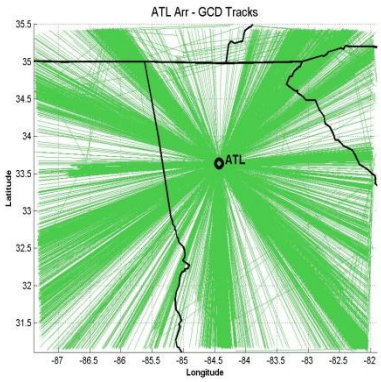
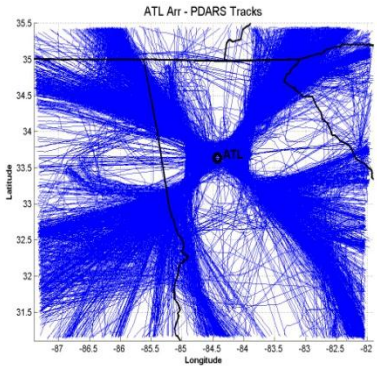
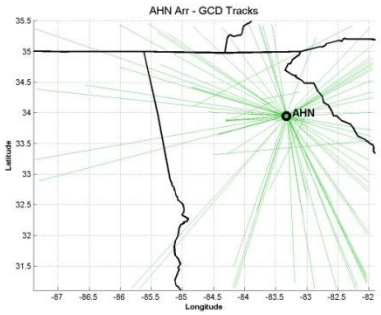
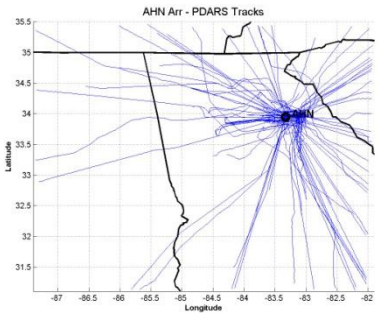
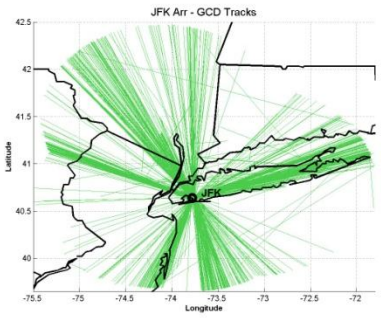
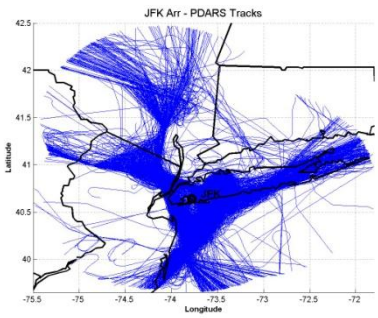
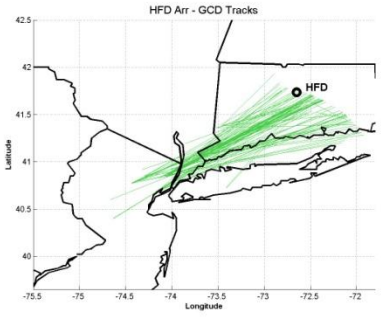
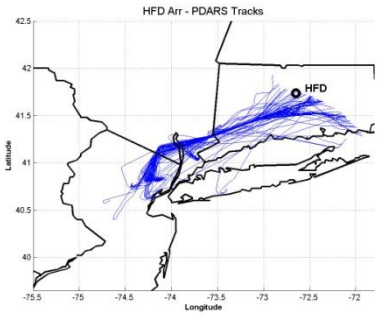
GCD Routes	PDARS Routes	Detour Factor
		ATL Arrivals (Atlanta TRACON) Average Detour Factor: 1.4648 Number of Sample: 8047
		AHN Arrivals (Atlanta TRACON) Average Detour Factor: 2.5247 Number of Sample: 103
		JFK Arrivals (New York TRACON) Average Detour Factor: 1.9153 Number of Sample: 2647
		HFD Arrivals (New York TRACON) Average Detour Factor: 1.2278 Number of Sample: 210

Figure 20. GCD Tracks Compare To PDARS Tracks For Arrivals.

The PDARS variables considered in the cluster analysis are shown in Table 4.

Table 4. Variables In PDARS Dataset That Used In Cluster Analysis.

Name	Description	Example
STATUS	Departure/Arrival Boolean	'Dep', 'Arr'
LESS_100SM	Number of airports within 100 stat. mi. radius	>=20 (greater than or equal to 20 airports)
FREQUENCY	Number of incoming/ outgoing aircraft per day	<100 (less than 100 flights)
DIST_TO_HUB	Distance to major hub (such as ATL, LAX, JFK,…) in degrees	<2 (less than 2 degrees)

These variables were included, along with the airport code and its detour factor, in a list of all PDARS airports. JMP statistical software was then used to sort all the airports into individual clusters, depending on their characteristics and detour factors (Rye, 2011). Cluster characteristics are defined in Table 5.

Table 5. Cluster Characteristics And Associated Detour Factor.

Cluster No.	Cluster Characterisitics	Avg Detour Factor
1)	STATUS: 'Dep' -- LESS_100SM: <20 -- FREQUENCY: <100 -- DIST_TO_HUB: <2	1.132
2)	STATUS: 'Dep' -- LESS_100SM: >=20 -- FREQUENCY: <100 -- DIST_TO_HUB: <2	1.099
3)	STATUS: 'Arr' -- LESS_100SM: <20 -- FREQUENCY: <100 -- DIST_TO_HUB: <2	1.186
4)	STATUS: 'Arr' -- LESS_100SM: >=20 -- FREQUENCY: <100 -- DIST_TO_HUB: <2	1.128
5)	STATUS: 'Arr' -- LESS_100SM: <20 -- FREQUENCY: >=100 -- DIST_TO_HUB: <2	1.415
6)	STATUS: 'Arr' -- LESS_100SM: >=20 -- FREQUENCY: >=100 -- DIST_TO_HUB: <2	1.204
7)	STATUS: 'Dep' -- FREQUENCY: >=100 -- DIST_TO_HUB: <2	1.122
8)	FREQUENCY: <100 -- DIST_TO_HUB: >=2	1.148

It is observed that large hub airports have relatively moderate detour factors. Smaller airports around large hubs have higher detour factors while those farther away from hubs have lower factors. The reason is that air traffic patterns and procedures are organized around large hub facilities and thus in some sense receive higher level of service when entering the complex airspace. It also can be concluded that in most cases, detour factors are higher for arrivals compared to departures (Rye, 2011).

3.9 Ground Fuel Consumption Estimation

The taxi-out and taxi-in operations contribute to the total fuel burn in commercial flights. This is important at large hub airports. Pollutants emitted by aircraft on the ground are of concern. In fuel consumption and emissions model, ground operations fuel consumption and emissions are calculated by estimating taxi-out and taxi-in times at specific airports. Taxi times at different airports are either extracted from FAA Aviation System Performance Metrics (ASPM) or predicted through yearly number of departures and arrivals at the specific airport using regression analysis.

The ASPM database provides both unimpeded and historical taxi-out and taxi-in times at 77 airports which handle the majority of the commercial flights every year (FAA, 2012). The actual taxi time includes delays. As a result, the fuel consumption calculated based on taxi times includes the fuel burn for ground delays.

For non-ASPM 77 airports, taxi times are predicted through regression analysis of the number of operations reported in T100 data. Figure 21 shows a linear trend between taxi-in time and yearly number of arrivals at the ASPM 77 airports.

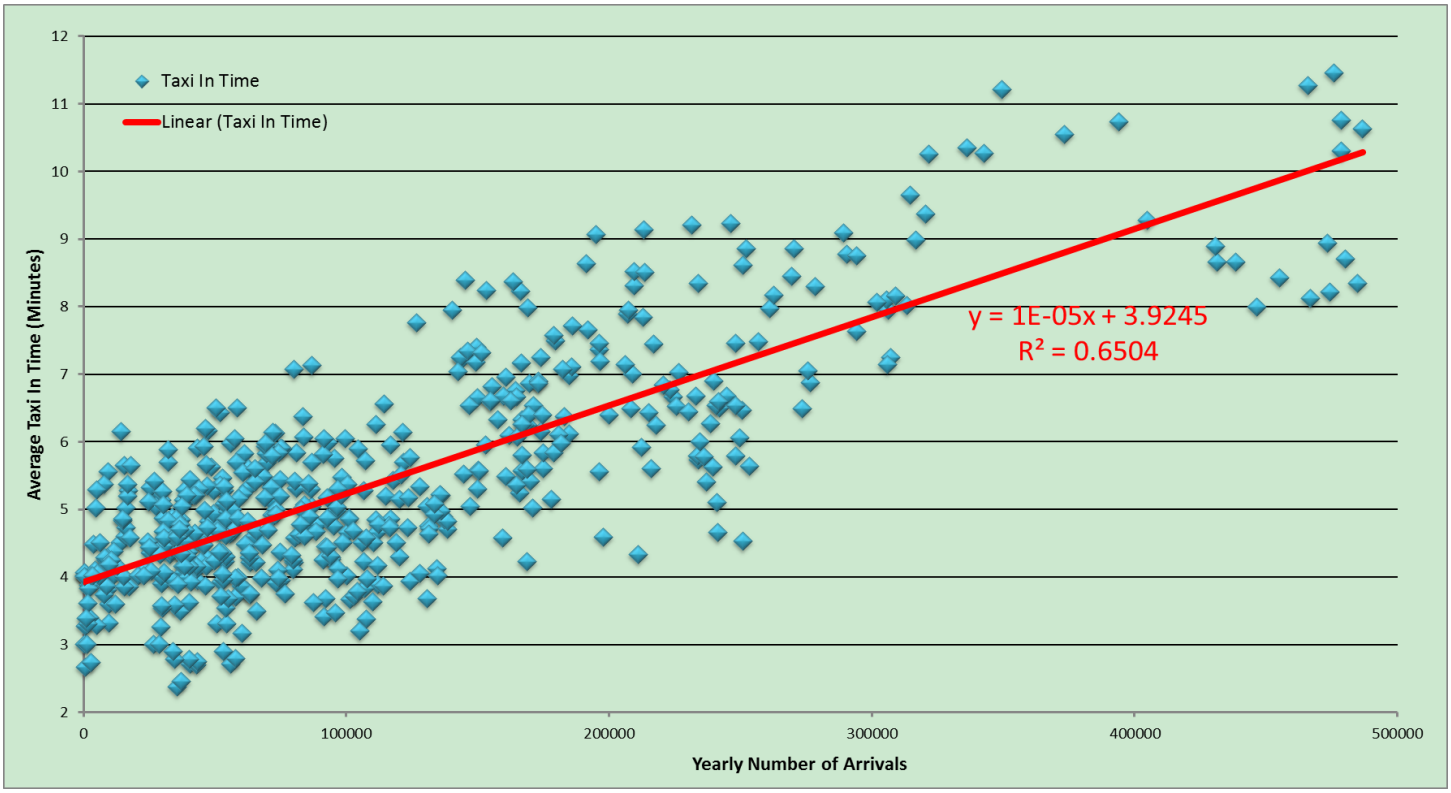


Figure 21. Relationship Between Taxi-in Time And Number Of Arrivals At ASPM 77 Airport.

A similar upward trend has also been observed in the relationship between taxi-out time and number of departures. Assuming that smaller airports maintain the same trend, taxi time at airports other than ASPM 77 can be predicted by the linear regression shown in Figure 21.

The BADA model does not provide fuel burn rate for taxi or ground maneuvering so an alternative source is used to estimate ground fuel burn rates. The FAA Emission and Dispersion Modeling System (EDMS) has been selected as the source of fuel burn data for ground operations. Aircraft fuel burn rate at idle conditions in the EDMS dataset is used to calculate the commercial flight fuel consumption for ground operations.

3.10 Emission Model

In the emission model, CO, HC, NOx and SOx emissions are calculated in the Landing/Take-off (LTO) cycle which includes all activities near the airport that take place up to 3000ft AGL. The calculation is simply the conversion of fuel consumption to emissions by multiplying the emission factor of the pollutant to the computed fuel burn expressed in Equation 64.

$$Emission\ (g) = Fuel\ Burn\ (kg) \times Emission\ Factor\ (g/kg\ fuel\ burn)$$

or (64)

$$Emission\ (g) = Fuel\ Flow\ (kg/s) \times Operational\ Time\ (s) \times Emission\ Factor\ (g/kg\ fuel\ burn)$$

The emission factors for the emissions model are obtained from the FAA EDMS database which provides factors of CO, HC, NOx and SOx for hundreds of aircraft types and engine combinations. EDMS provides emission factors for takeoff, climb-out, approach, and idle conditions. EDMS is a combined emissions and dispersion model for assessing air quality at civilian airports and military air bases (FAA, 2011). EDMS 5 uses the same BADA data for aircraft performance modeling which is consistent with previously stated flight profile analysis. Table 6 demonstrates the emission factors for commonly used commercial aircraft types extracted from the EDMS database.

Table 6. Sample Of Emission Factors For Some Aircraft Types Extracted From EDMS.

BADA Code	Takeoff_ CO (g/Kg)	Takeoff _HC (g/Kg)	Takeoff_ NOx (g/Kg)	Takeoff_ SOx (g/Kg)	Approach _CO (g/Kg)	Approach _HC (g/Kg)	Approach_ Nox (g/Kg)	Approach _Sox (g/Kg)
A318__	0.8	0.1	21.4	1	3.7	0.8	8.8	1
A319__	0.57	0.041	24.5	1	2.6	0.062	8.7	1
A320__	0.9	0.23	24.6	1	2.5	0.4	8	1
A333__	0.05	0.04	28.72	1	1.85	0.11	12.66	1
A343__	0.8	0.1	23.3	1	23.1	3.7	7.3	1
A345__	0.02	0	44.91	1	0.46	0	11.78	1
A346__	0.02	0	44.91	1	0.46	0	11.78	1
B734__	0.9	0.04	20.7	1	3.1	0.08	8.7	1
B735__	0.9	0.03	20.7	1	3.1	0.07	9.1	1
B737__	0.4	0.1	25.3	1	2.2	0.1	10.1	1
B738__	0.2	0.1	28.8	1	1.6	0.1	10.8	1
B739__	0.2	0.1	30.9	1	1.4	0.1	11	1
B742__	0.45	0.14	28.97	1	3.71	0.28	10.16	1
B743__	0.45	0.14	28.97	1	3.71	0.28	10.16	1
B744__	0.44	0.06	28.1	1	2	0.13	11.6	1
B752__	0.33	0.02	29.41	1	1.95	0.11	9.77	1
B753__	0.3	0.01	37.77	1	1.32	0.09	10.7	1
B762__	1	0.3	29.6	1	2.8	0.45	10.8	1
B763__	0.37	0.1	32.8	1	1.78	0.14	12	1
B764__	0.05	0.05	27.38	1	1.93	0.11	12.63	1
B772__	0.19	0.03	61	1	0.44	0.06	13.19	1
B773__	0.19	0.03	61	1	0.44	0.06	13.19	1
B77W__	0.19	0.03	61	1	0.44	0.06	13.19	1

The fuel consumption for take-off, climb, and descent up to 3000 feet above ground and taxi are extracted from flight profile fuel burn data. Emissions are then calculated accordingly.

For CO_2, a traditional way of calculating emission inventories for an aircraft in the LTO cycle is using the ICAO Carbon Emissions Calculator. Since the fuel module has estimated the fuel consumption for the complete flight cycle, CO_2 emissions can be converted from fuel burn data. The model adopts the CO_2 emission factor of 3.157 (kg/kg fuel burn) which is a chemical

constant relating the mass of CO_2 produced by stoichiometric combustion of a known amount of fuel (Jardine 2003).

4. MODEL RESULTS AND VALIDATION

4.1 Fuel Consumption Results and Validation

The model has been validated using T100 data from 2003 to 2010 as input to the model. The model results are compared to the statistics of Certified Air Carrier Fuel Consumption and Travel reported by the Bureau of Transportation Statistics (BTS), Research and Innovative Technology Administration (RITA). The T-100 data is the Air Carrier Statistics database reported by RITA as well. The T100 data contains domestic and international airline market and segment data. The T-100 Domestic Segment (All Carriers) table is used in the model validation (RITA, 2012). The table contains domestic non-stop segment data including origin, destination and aircraft type which are three inputs required by the fuel and emissions model.

Each T-100 aircraft is mapped into one of 117 equivalent aircraft types available in version 3.9 of the BADA model. ETMS waypoints data are applied to T-100 origin-destination pair if a flight between those OD pairs exists in ETMS data. Otherwise, GCD track data is assumed for that flight.

National Transportation Statistics presents statistics on the U.S transportation system, including its physical components, safety record, economic performance, human and natural environment, and national security (RITA, 2012). Among all the datasets in the National Transportation Statistics, Table 4-8, Certificated Air Carrier Fuel Consumption and Travel is used as baseline in this validation effort.

Table 7 lists the fuel consumption and emission and other relevant parameters estimated by the model along with the corresponding values reported by BTS for years 2003 to 2010.

Table 7. Model Outputs Compare To RITA Fuel Burn Report.

	Total Yearly Mileage (Millions)			Total Fuel Burn (Million Gallons)		
	Model	RITA	Error	Model	RITA	Error
2003	6,197	5,896	5.11%	12,318	13,082	-5.84%
2004	6,710	6,366	5.40%	12,974	14,091	-7.93%
2005	6,844	6,529	4.82%	12,923	13,976	-7.54%
2006	6,716	6,423	4.56%	12,562	13,694	-8.27%
2007	6,849	6,534	4.82%	12,752	13,682	-6.80%
2008	6,542	6,247	4.73%	12,085	12,686	-4.74%
2009	6,033	5,757	4.79%	11,181	11,339	-1.39%
2010	6,072	5,807	4.56%	11,193	11,256	-0.56%

Figure 22 shows the comparison of modeled fuel consumption to the RITA Report.

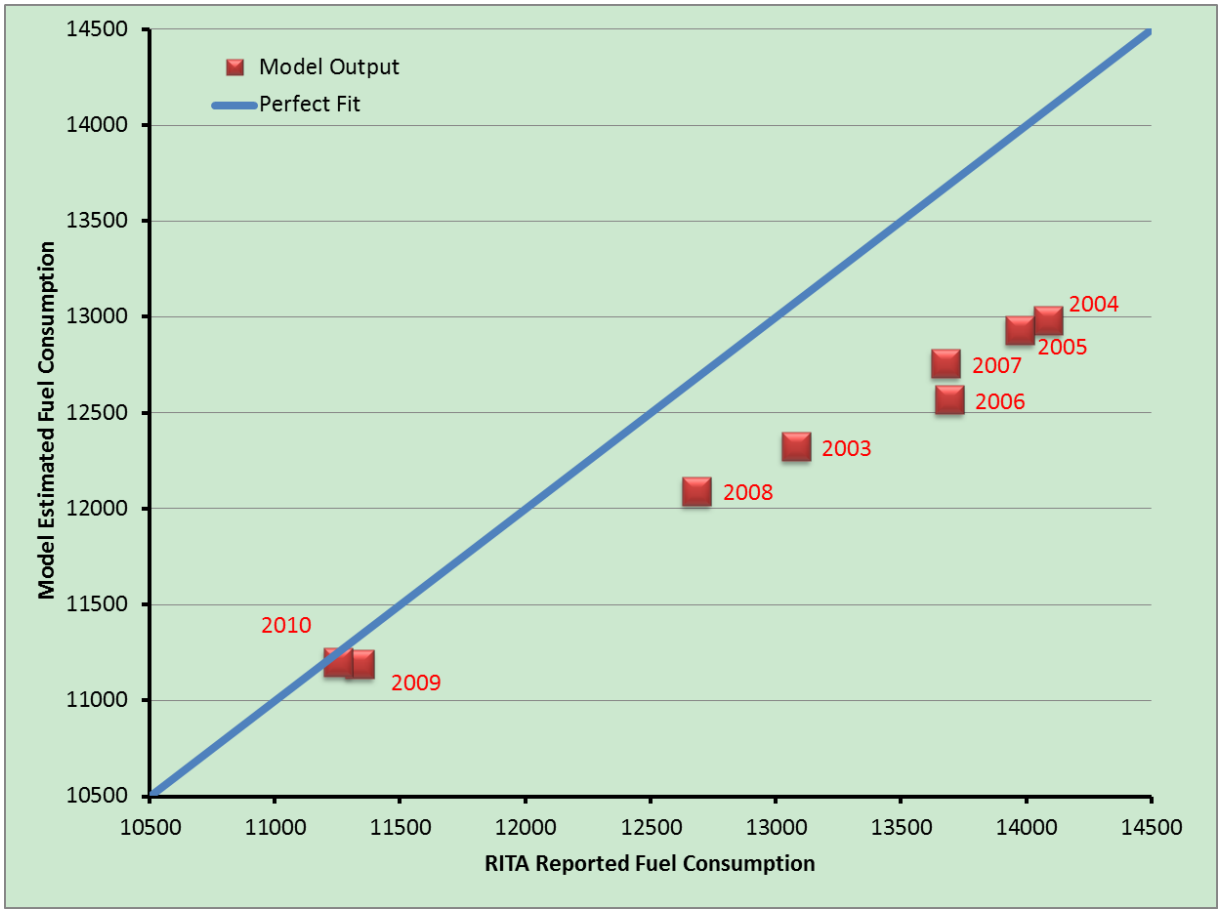


Figure 22. Comparison Of Modeled Fuel Consumption To The RITA Reported Number.

It is observed that the model underestimates the fuel consumption. The errors are around 5%. Consider that the average errors of BADA model (version 3.9) is about 3% (EUROCONTROL, 2011), the output of the model is satisfactory. Besides, the model captures the trend of fuel consumption and mileage flown very well through all years. The errors for year 2009 and 2010 are down to within 1.5% of the values reported by BTS.

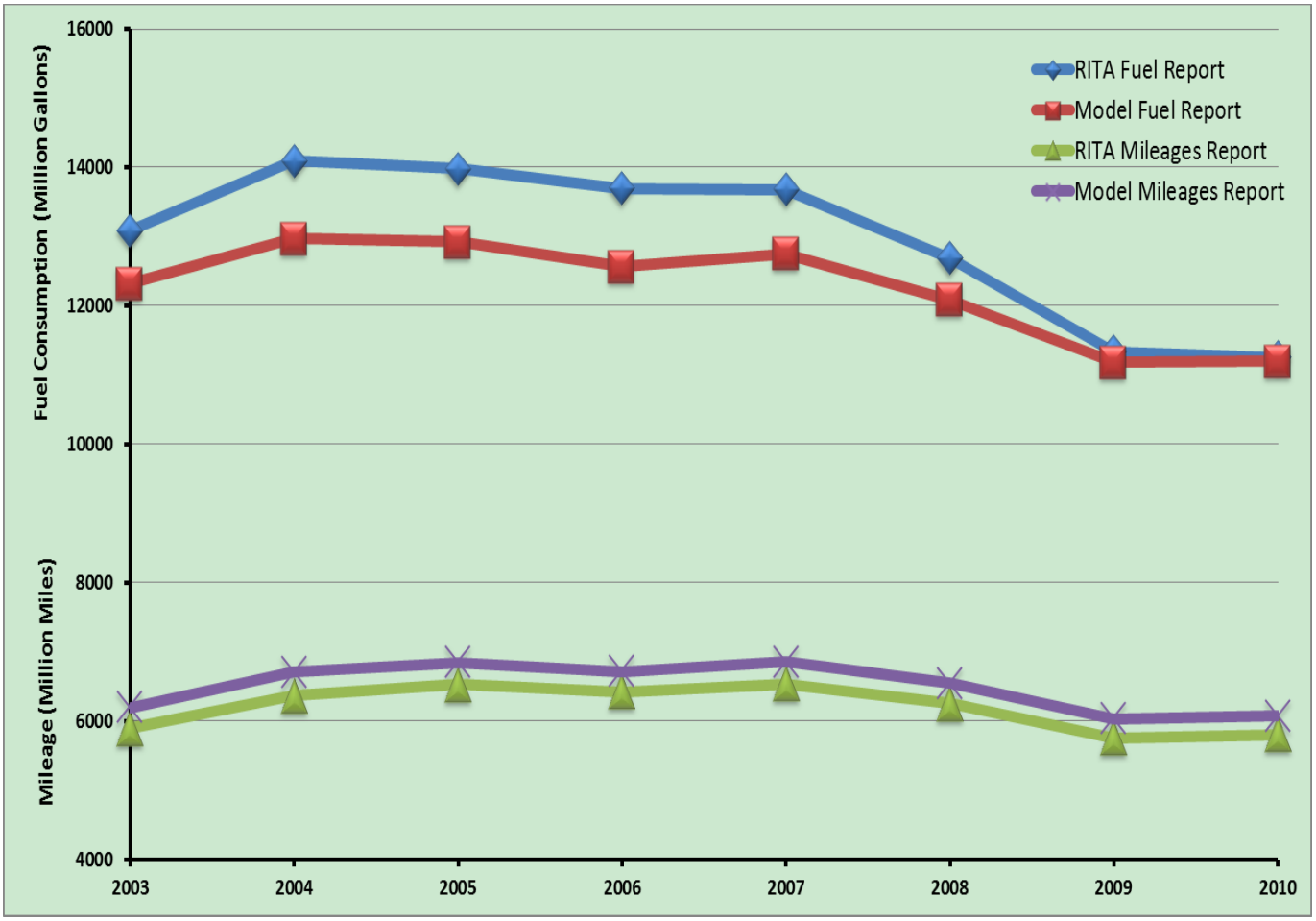


Figure 23. Yearly Trend Of Fuel Consumption Estimated By Model And Reported By RITA.

4.2 Comparison of Fuel Flow Rates between BADA Data and Model Output

In the developed fuel consumption model, fuel flow rate is calculated using equations and coefficients from BADA model specifications. The model computed fuel flow rates are validated by comparing the value to the BADA Performance Table File (PTF). Figure 25 shows the comparison for Airbus A320.

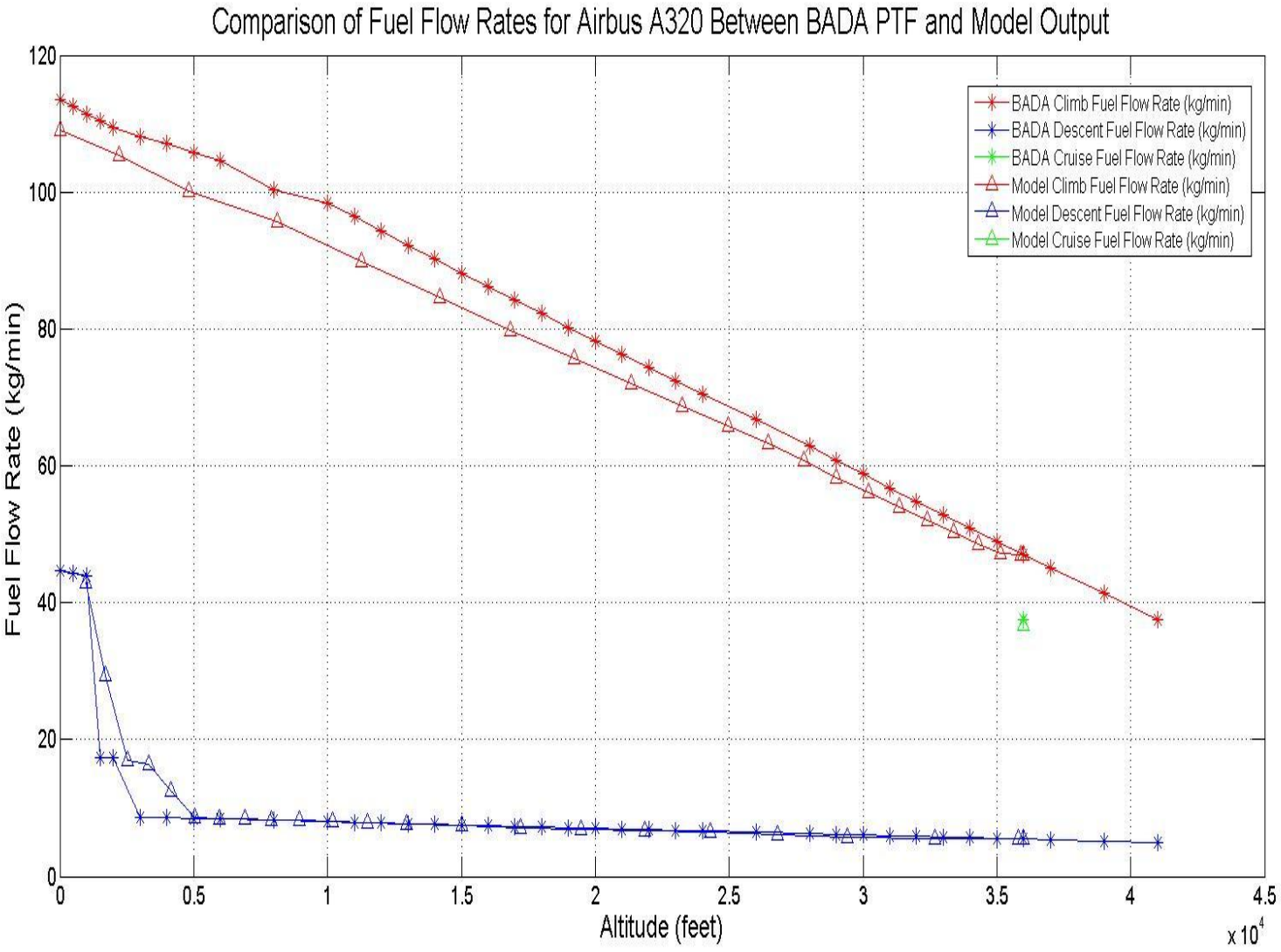


Figure 24. Comparison of Fuel Flow Rates for Airbus A320 Between BADA PTF and Model Output.

Figure 26 shows the comparison for Boeing 737-300.

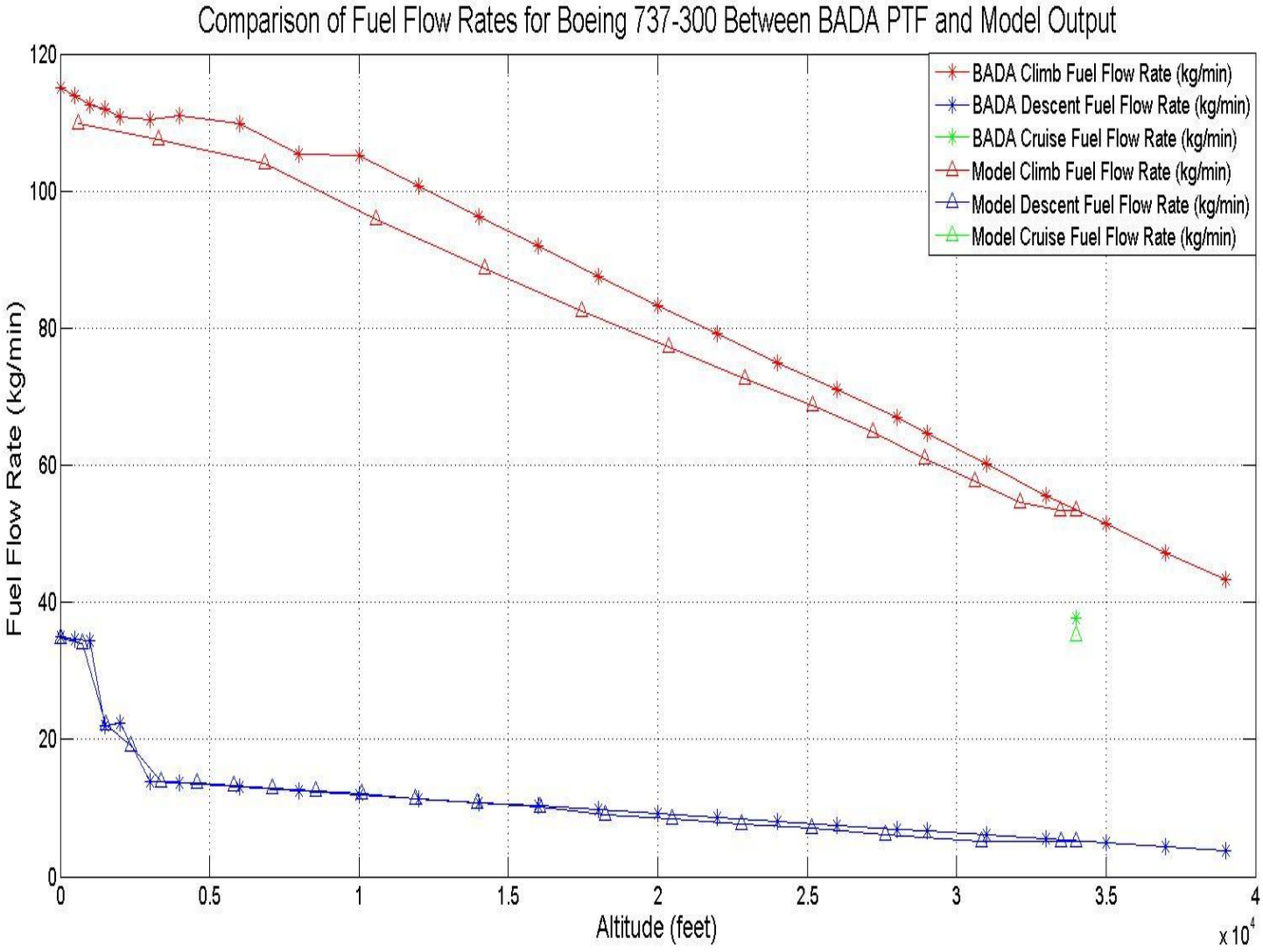


Figure 25. Comparison of Fuel Flow Rates for Boeing 737-300 Between BADA PTF and Model Output.

Same comparisons have been done for the other BADA aircraft types. All of the plots show that model computed fuel flow rates have consistent trends compare to the values provided by BADA PTF. Consider that the aircraft weight is decreasing when calculating the fuel flow rate in the model, the fuel flow rate also reduces. In BADA PTF, the constant nominal weight is assumed to

obtain the fuel flow rate. This is the reason that the discrepancies between BADA PTF and model output are observed in the Figures 24 and 25.

Additionally, the plot of fuel consumption against flight distance for each aircraft type also shows reasonable trend. For example, Figure 26 shows the fuel burn trend for Airbus A320, and Figure 27 shows the same plot for Boeing 737-300.

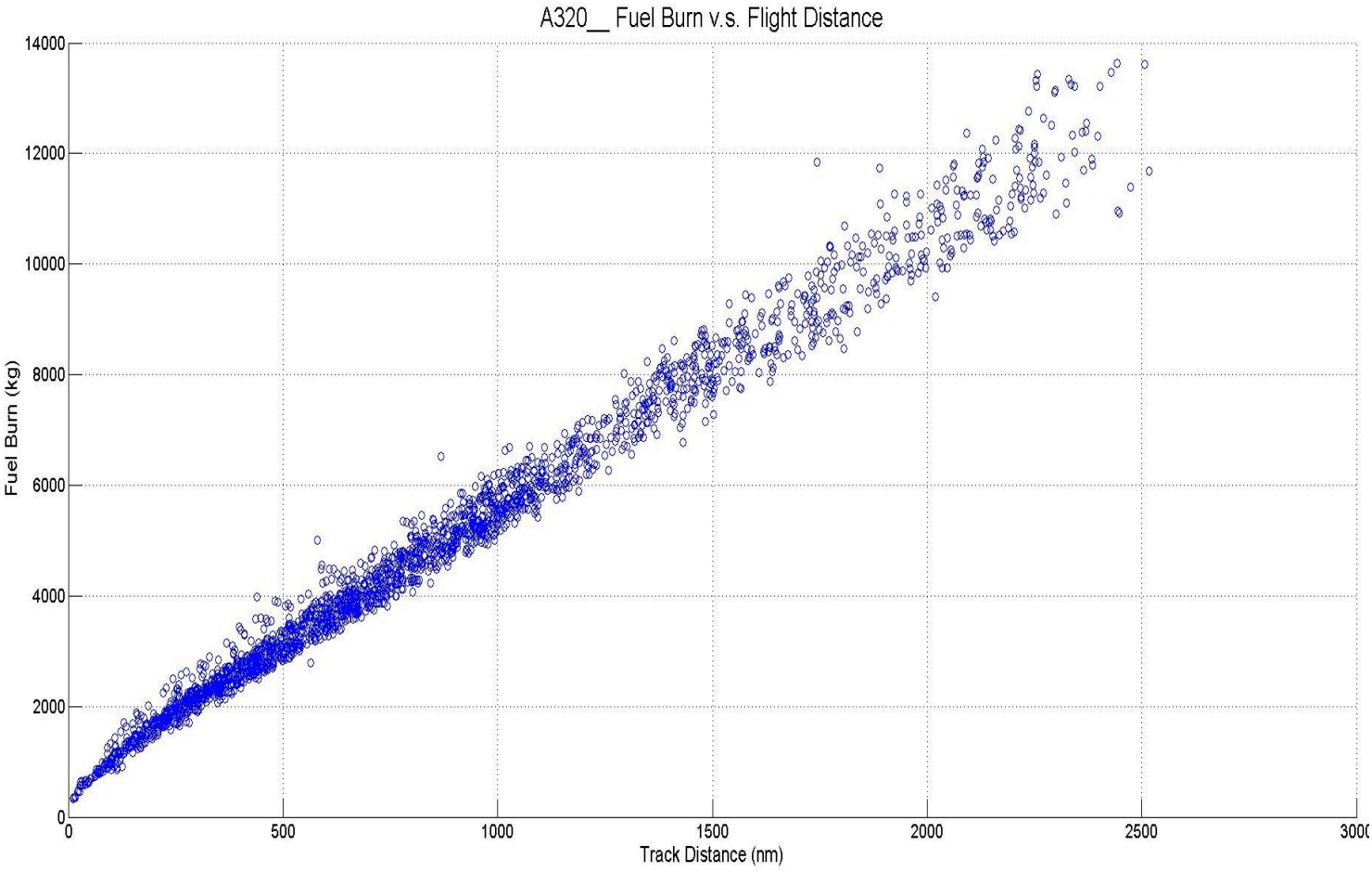


Figure 26. Fuel Burn Trend For Airbus A320

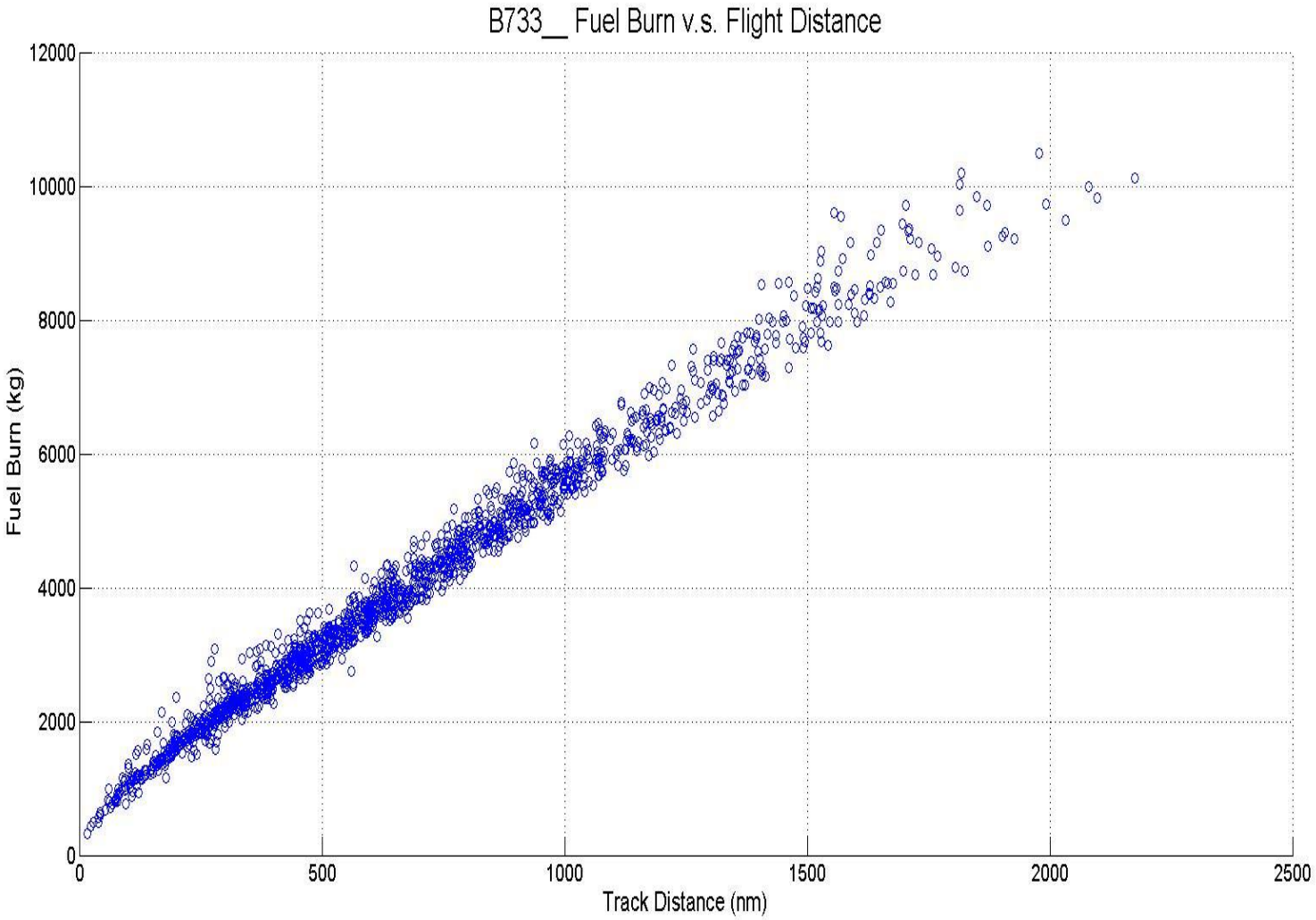


Figure 27. Fuel Burn Trend For Boeing 737-300

Non-linear trends are observed for the short distance flights. The short cruise distance for the flights combined with the non-linear trend of fuel burn rate for climb, are the reasons for this.

For the long distance flights, two main branches appear in the plot. They explain the head and tail wind effects projected on the aircraft when heading West and East.

5. CONCLUSION

A model to estimate fuel and emissions for commercial flights in the continental U.S. has been developed. The model can work either as a stand-alone calculator or as a sub-module in TSAM - the Transportation System Analysis Model. The addition of fuel computations in TSAM provides the basis to study benefits of aviation technology in a multimode model from the perspective of energy efficiency and environment impacts. The fuel and emissions model can help NASA estimate the fuel consumption and emissions using other models such as ACES.

The fuel and emissions model developed employs the BADA aircraft performance model to calculate fuel consumption considering operational conditions such as wind states, terminal area detour factors, etc. The model serves as a decision tool where sensitivity analyses can be done for various operational conditions to study the impact of new technologies on fuel and emission metrics.

Unlike other existing models, fuel consumption and emissions on the ground are also considered using the FAA EDMS database. The model fuel consumption output is an estimate of the complete flight cycle and not just the en-route component.

Airlines and Air Traffic Controllers can use the model to run sensitivity analysis for the aeronautic factors such as route and cruise level assignment, take-off weight and speed.

Considering the average errors of 3% for BADA model reported by EUROCONTROL (EUROCONTROL, 2011), the validation results show that the model output is satisfactory in estimating fuel consumption and has good performance in the generation of a flight profile.

This model has been programmed in MATLAB. The model can calculate fuel consumption and emissions for a single flight in two to five seconds depends on the flight distance. The MATLAB code is flexible to be embedded into other software or simulation tools. Integrated with TSAM, the model is an effective and quick tool to predict the fuel consumption and emissions by commercial air flights in the future.

6. RECOMMENDATIONS

This section provides some detail on potential improvements to the fuel and emissions model developed.

6.1 Terminal Area Flight Profile

As stated in Section 3.3.2, an assumption has been made that all airplanes depart and arrive using standard procedures which is defined in the BADA Airline Procedures Model. The effects of air traffic management (ATM) in the terminal area on these operations are not modeled by the BADA which can lead to discrepancies in estimating the fuel consumption and emissions.

Future work could involve developing an algorithm to model more details of an ATM-influenced operation so that a better representation of terminal area operations can be provided. The model could be enhanced to include procedures expected to be in place for the next generation (NextGen). This could include tailored arrivals and continuous descent approaches. It is recommended that this model adopts BADA version 4 coefficients. This will enable modeling and simulating advanced concepts.

6.2 Wind States

Stated in Section 3.7, a tradeoff between program execution time and accuracy has been made. Under current assumptions we estimate fuel and emissions using long-term averages of the wind data. More efficient computer code could be developed to account for wind datasets with higher fidelity, such as considering daily averages, without slowing down the program execution speed.

6.3 Ground Fuel Consumption and Emissions

Additional efforts can be placed in the estimation of the ground fuel consumption and emissions. In the current model, activities such as stop and go operations are not fully modeled during the ground taxi operation. Additionally, the taxi time estimated in the model is the OOOI time (gate out, wheels off, wheel on, gate in) which does not include the time for the aircraft waiting at the gate. This would add to the fuel consumption and emissions when the aircraft is at the gate with the engines running. Further analysis of gate operational times can be carried out for different aircraft types in the model.

REFERENCES

Ashiabor, Senanu, Hojong Baik, and Antonio A Trani. "Logit Models for Forecasting Nationwide Intercity Travel Demand in the United States." *Transportation Research Record*, 2007: 1-12.

Braven, den, and W S John. "Concept and Operation of the Performance Data Analysis and Reporting System (PDARS)." 2003.

Browder, J, R Gutterrud, and J Schade. "Performance Data Analysis Reporting System (PDARS) - A Valuable Addition to FAA Manager's Toolsets." *Managing the Skies - Journal of the FAA Managers Association* , 2010.

Bureau of Transportation Statistics. *Domestic Airline Jet Fuel Prices.* 2012. http://www.bts.gov/publications/multimodal_transportation_indicators/april_2012/html/domestic_airline_jet_fuel_prices.html (accessed 2012).

Chatterji, Gano B. "Fuel Burn Estimation Using Real Track Data." *11th AIAA Aviation Technology, Integration and Operations Conference.* Virginia Beach, 2011.

Chirania, Saloni Ramesh. "Forecasting Model for High-Speed Rail In the United States." October 2012.

Collins, Bela P. "Estimation of Aircraft Fuel Consumption." *Journal of Aircraft,* November 1982: 969.

EUROCONTROL. *Base of Aircraft Data (BADA).* 2011. http://www.eurocontrol.int/services/bada (accessed 2012).

EUROCONTROL. "Model Accuracy Report For The Base of Aircraft Data (BADA) Revision 3.9." 2011.

EUROCONTROL. "User Manual for the Base of Aircraft Data (BADA) Revision 3.9." User Manual, 2011.

Federal Aviation Administration (FAA). *ASPM System Overview.* October 17, 2012. http://aspmhelp.faa.gov/index.php/ASPM_System_Overview (accessed 2012).

Federal Aviation Administration (FAA). *Emissions and Dispersion Modeling System (EDMS).* Feburary 16, 2011.

http://www.faa.gov/about/office_org/headquarters_offices/apl/research/models/edms_model/ (accessed 2012).

Federal Aviation Administration (FAA). *Enhanced Traffic Management System (ETMS).* May 03, 2012. http://hf.tc.faa.gov/projects/etms.htm (accessed 2012).

Federal Aviation Administration (FAA), and CSSI Inc. "Emissions and Dispersion Modeling System (EDMS) User's Manual." User's Manual, Washington D.C, 2010.

Flightaware. *Live Flight Tracker.* 2012. http://flightaware.com/ (accessed 2012).

IATA. *Fact Sheet: Fuel.* 2012. http://www.iata.org/pressroom/facts_figures/fact_sheets/pages/fuel.aspx (accessed 2012).

International Civil Aviation Organization (ICAO). "ICAO Annex 16: Environmental Protection, Volume II -- Aircraft Engine Emissions." ICAO Annex, 2008.

International Civil Aviation Organization (ICAO). "ICAO Carbon Emissins Calculator." Manual, 2010.

Jardine, Christian N. "Calculating The Carbon Dioxide Emissions Of Flights." Oxford, 2003.

NOAA, ESRL, and PSD. "NCEP/NCAR Reanalysis Monthly Means and Other Derived Variables: Pressure Level." *Earth System Research Laboratory.* 2012. http://www.esrl.noaa.gov/psd/data/gridded/data.ncep.reanalysis.derived.pressure.html (accessed 2012).

Oaks, Robert D, Hollis F Ryan, and Mike Paglione. "Prototype Implementation and Concept Validation of a 4-D Trajectory Fuel Burn Model Application." *AIAA Guidance, Navigation and Control Conference.* Toronto: AIAA, 2010. AIAA 2010-8164.

Patterson, Judith, George J Noel, David A Senzig, Christopher J Roof, and Gregg G Fleming. "Analysis of Departure and Arrival Profiles Using Real-Time Aircraft Data." *Journal of Aircraft*, 2009: Vol.46, No.4.

RITA, and BTS. *Introduction.* 2012. http://www.bts.gov/publications/national_transportation_statistics/html/introduction.html (accessed 2012).

RITA, and Bureau of Transportation Statistics (BTS). "T-100 Segment (All Carriers)." *TranStats.* 2012. http://www.transtats.bts.gov/Fields.asp?Table_ID=293 (accessed 2012).

Rye, Maria. "Fuel Consumption and Emissions Model Whitepaper." Whitepaper, 2011.

Rye, Maria. "Fuel Consumption and Emissions Project Progress Report." 2010.

Schroeder, Nataliya T. *Analysis of Potential Wake Turbulence Encounters in Current and NextGen Flight Operations.* Master Thesis, Blacksburg: Virginia Tech, 2011.

Senzig, David A, Gregg G Fleming, and Ralph J Iovinelli. "Fuel Consumption Modeling in Support of ATM Environmental Decision-making." 8th USA/Europe Air Traffic Management Research and Development Seminar, 2009.

Senzig, David A, Gregg G Fleming, and Ralph J Iovinelli. "Modeling of Terminal-Area Airplane Fuel Consumption." *Journal of Aircraft* 46 (2009).

Trani, Antonio A, Hojong Baik, Howard Swingle, and Senanu Ashiabor. "Integrated Model for Studying Small Aircraft Transportation System." *Transportaton Research Record,* 2003: 1-10.

Trani, Antonio A., Nicolas Hinze, Howard Swingle, Senanu Ashiabor, and Anand Seshadri. "Forecasting Model for Air Taxi, Commercial Airline, and Automobile Demand in the United States." *Transportation Research Record,* 2008: 9-20.

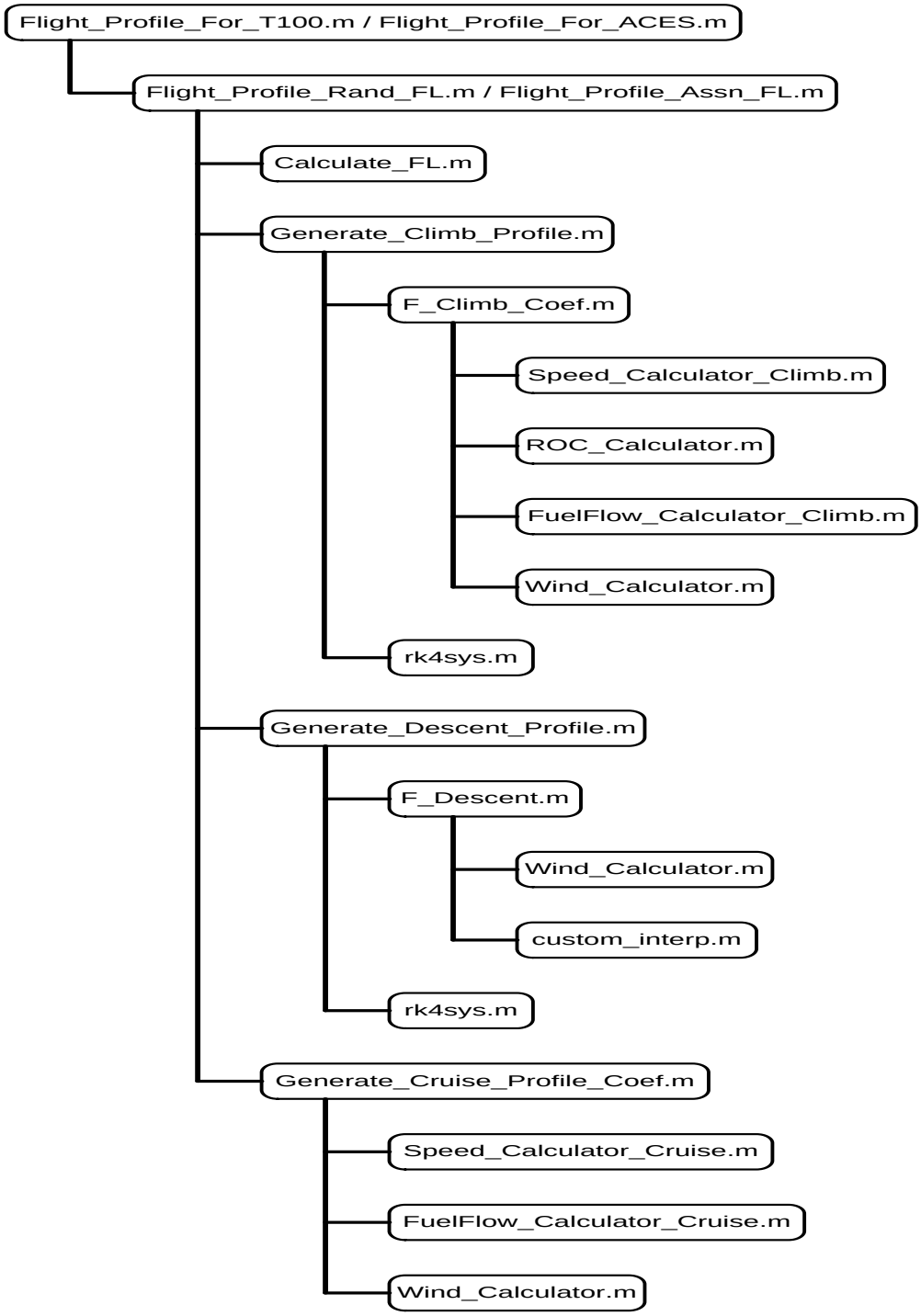
UNIQUE. *Aircraft NOx-Emissions within the Operational LTO Cycle.* Zurich: Unique (Flughafen Zurich AG) , 2004.

Wikipedia. *International Standard Atmosphere.* July 12, 2012. http://en.wikipedia.org/wiki/International_Standard_Atmosphere (accessed 2012).

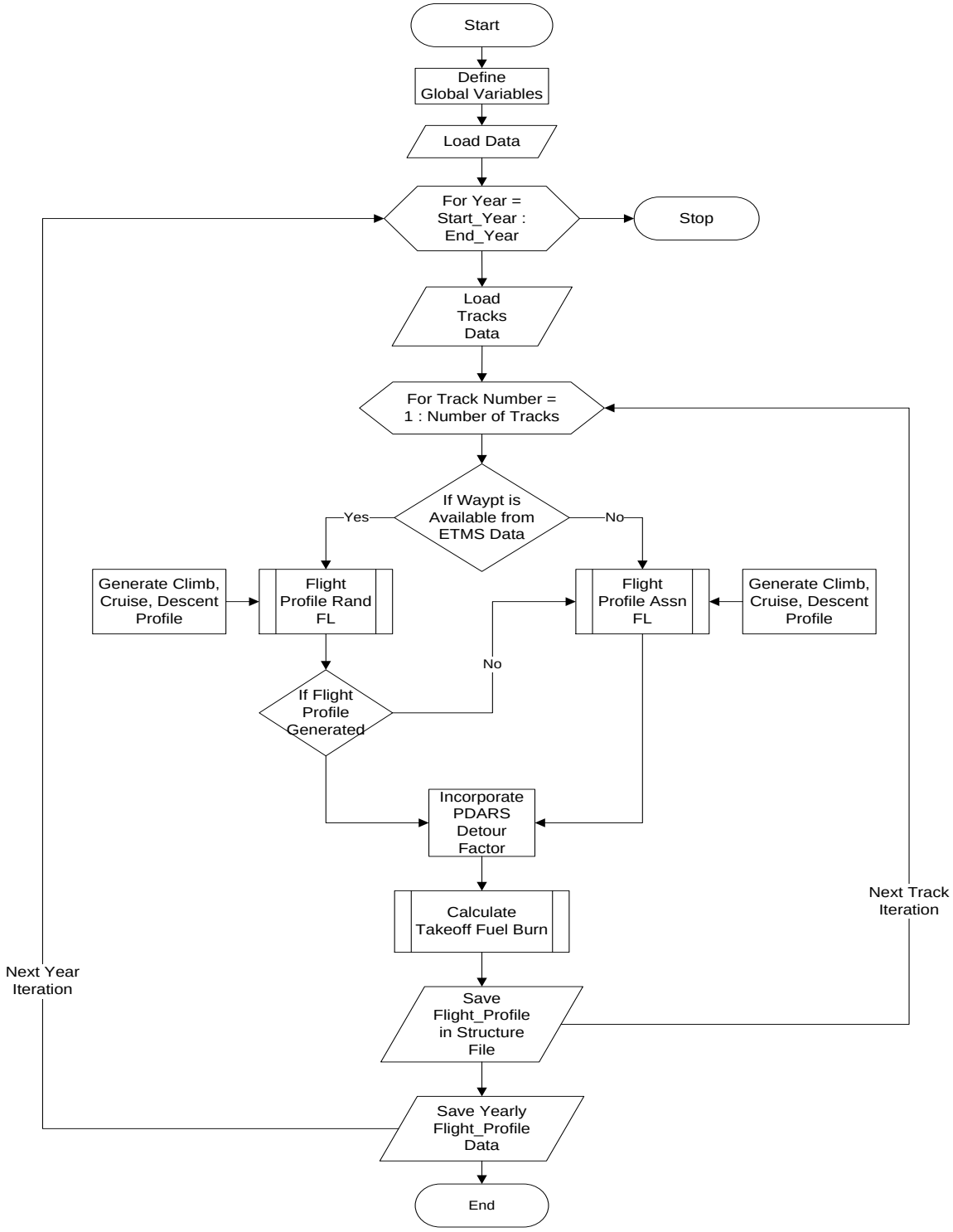
Wing-Ho, F C, Antonio A Trani, G Schilling, Hojong Baik, and A Seshadri. "A Neural Network Model to Estimate Aircraft Fuel Consumption." *AIAA 4th Aviation Technology, Integration and Operations Forum.* Chicago, 2004.

APPENDIX A: Model Flowcharts

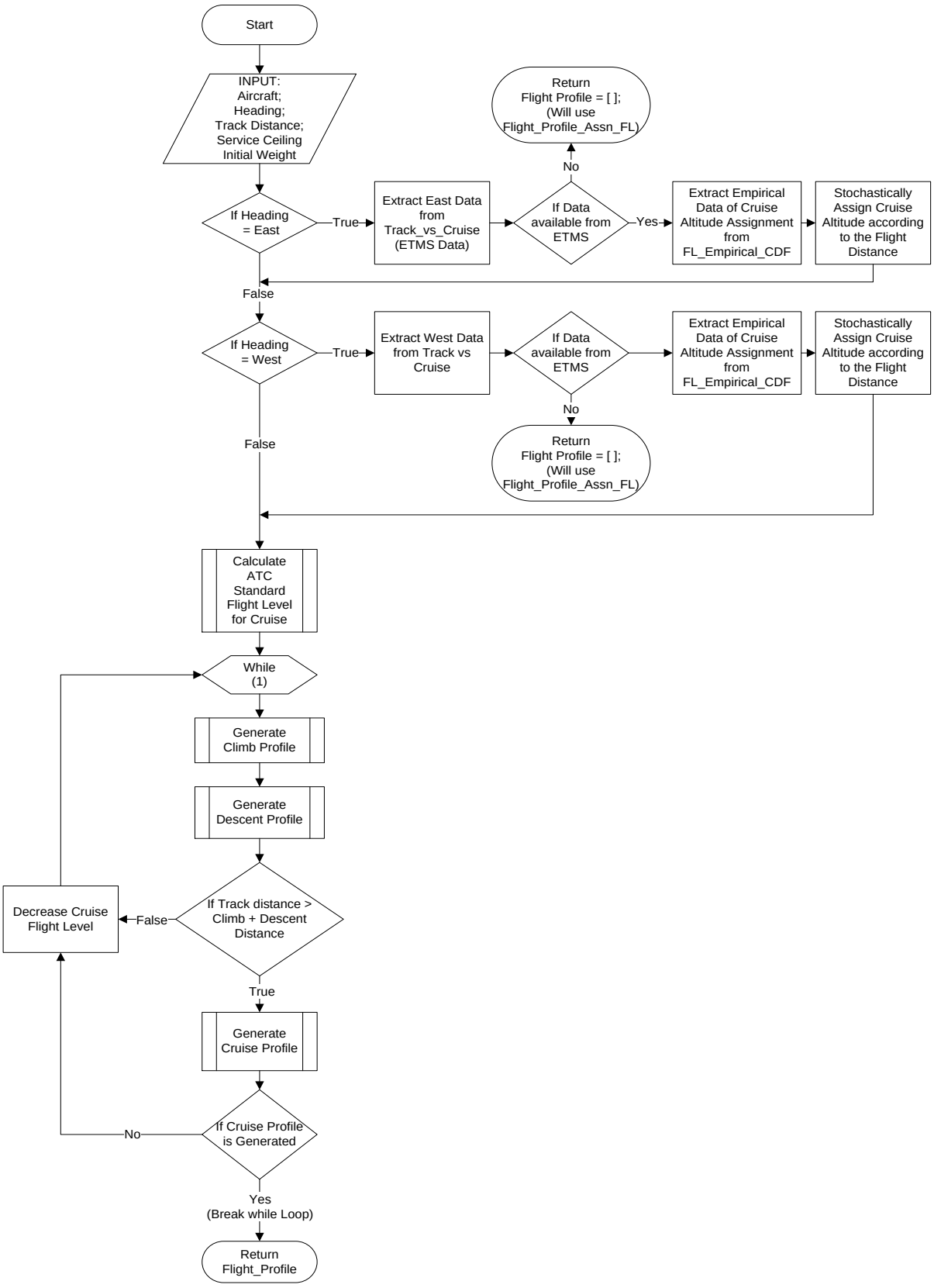
A.1 Function Dependency



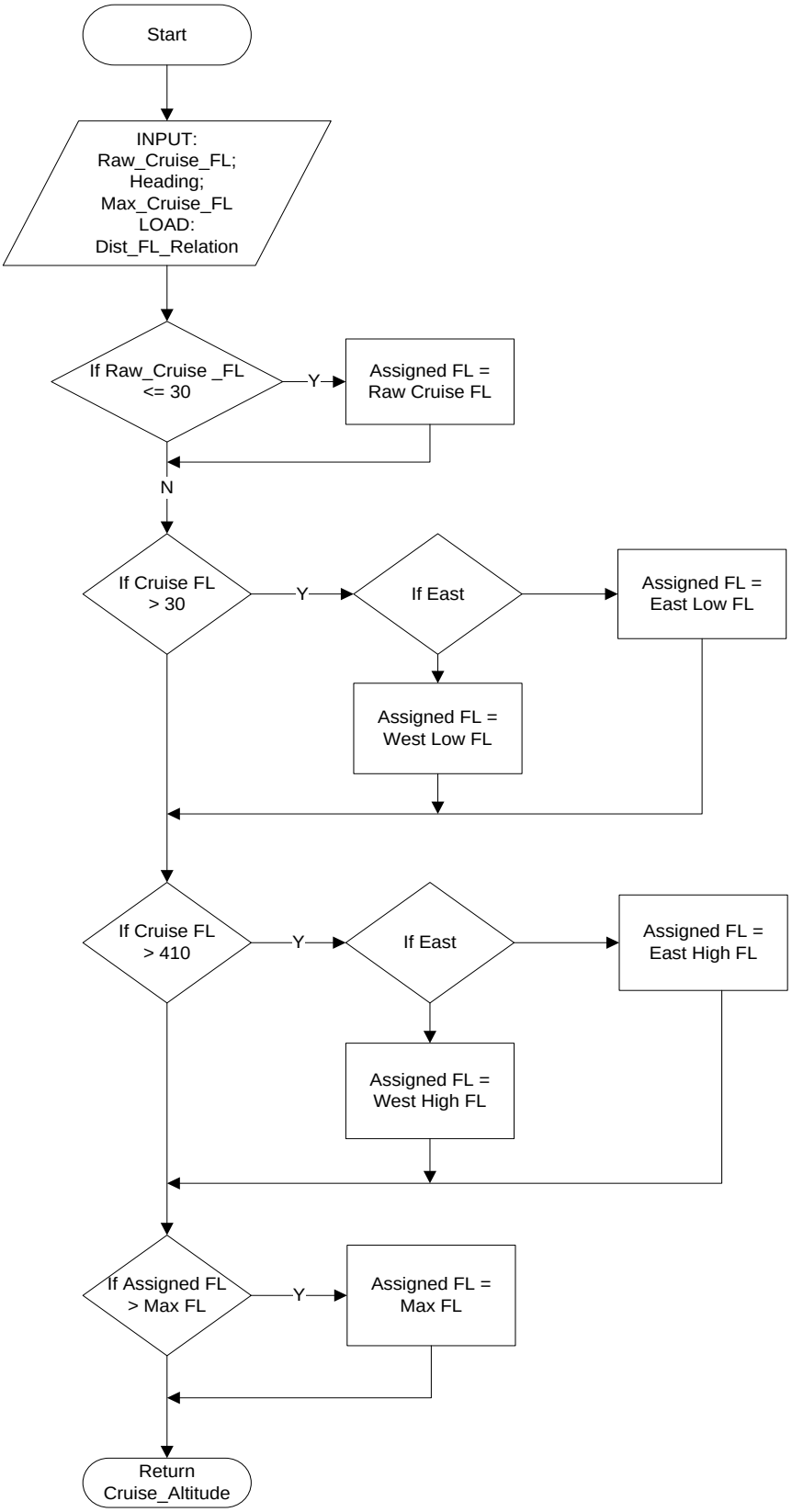
A.2 Flight Profile Generator



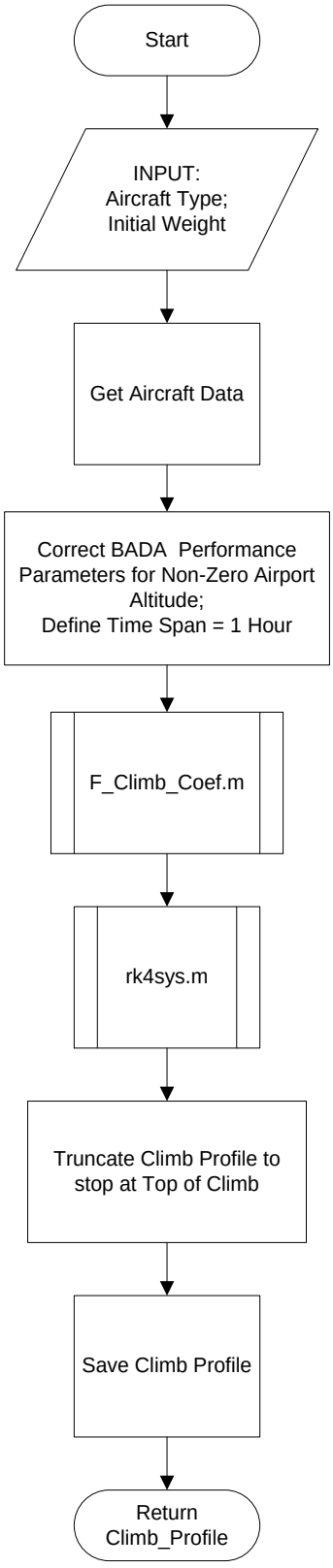
A.3 Flight_Profile_Rand_FL



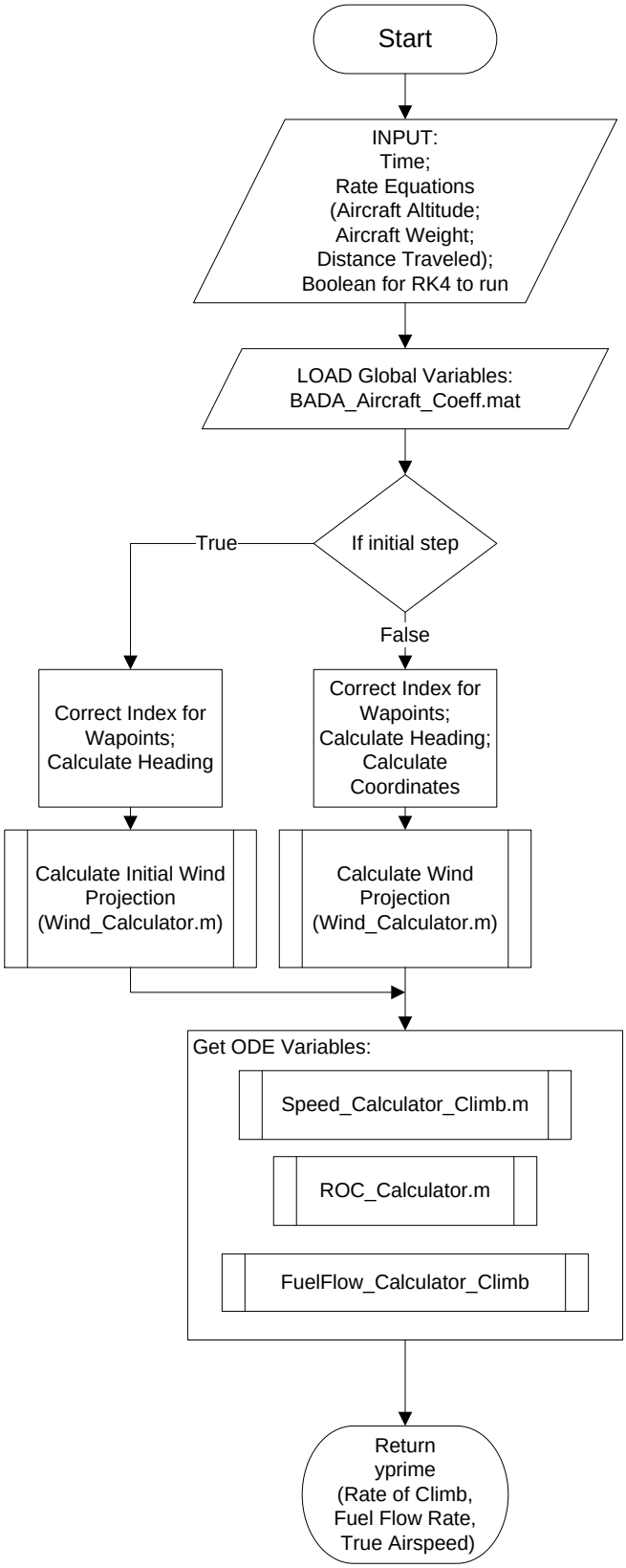
A.4 Calculate_FL



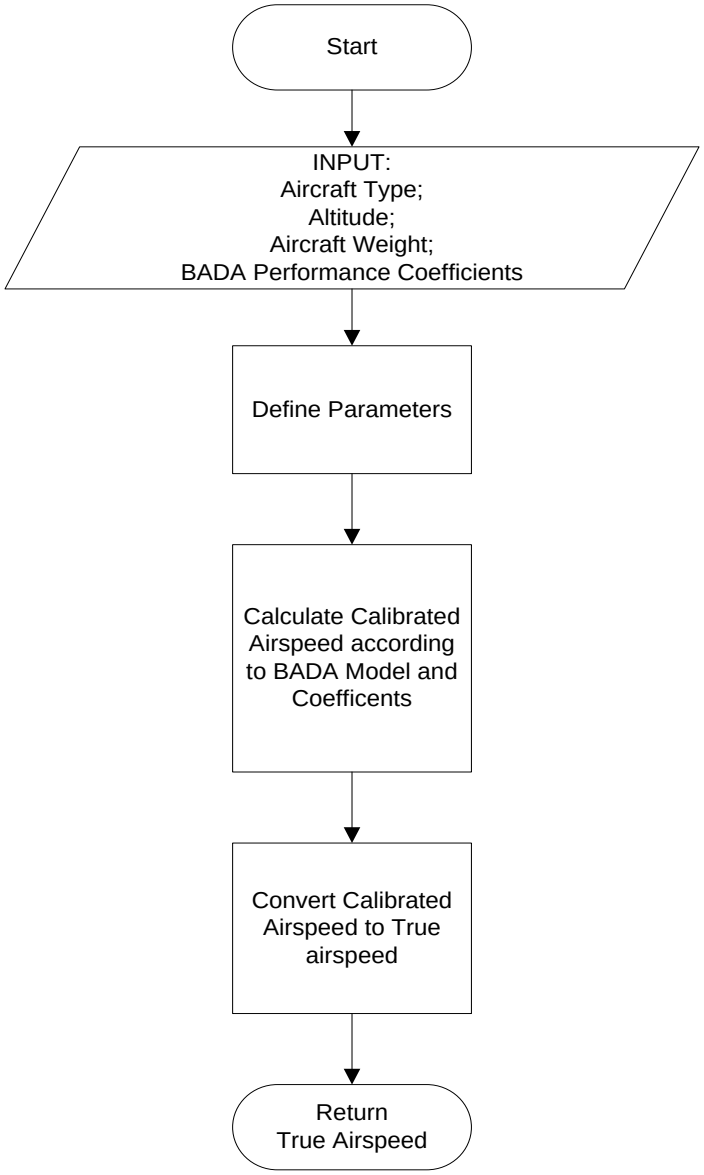
A.5 Generate_Climb_Profile



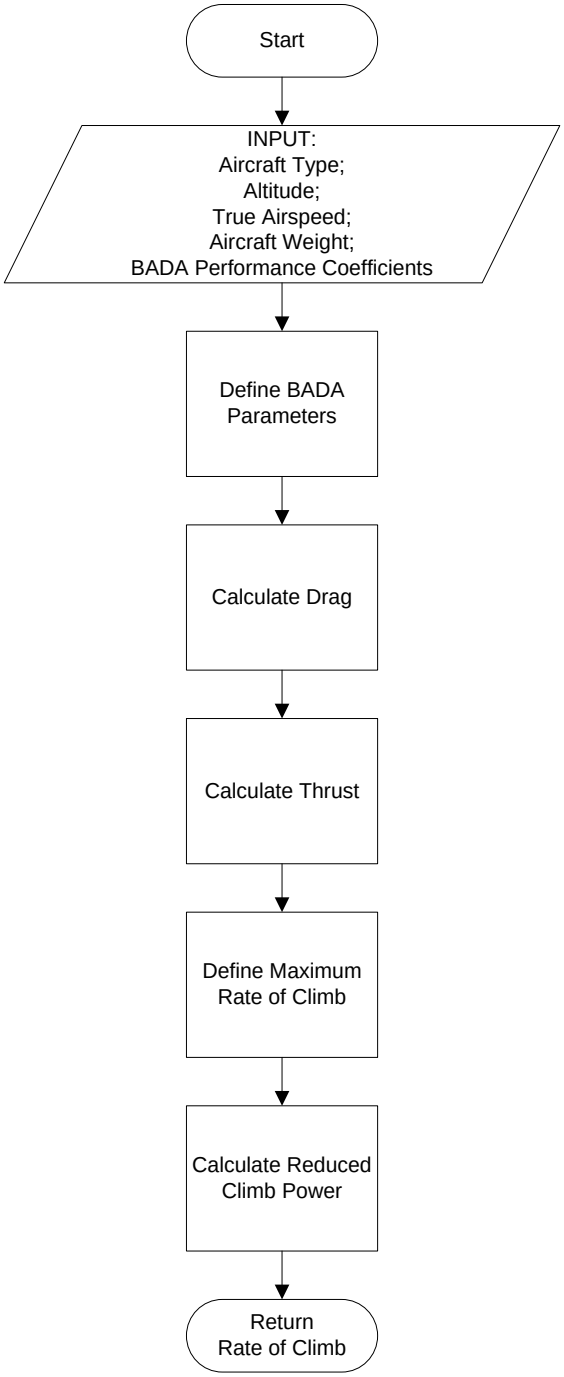
A.6 F_Climb_Coef



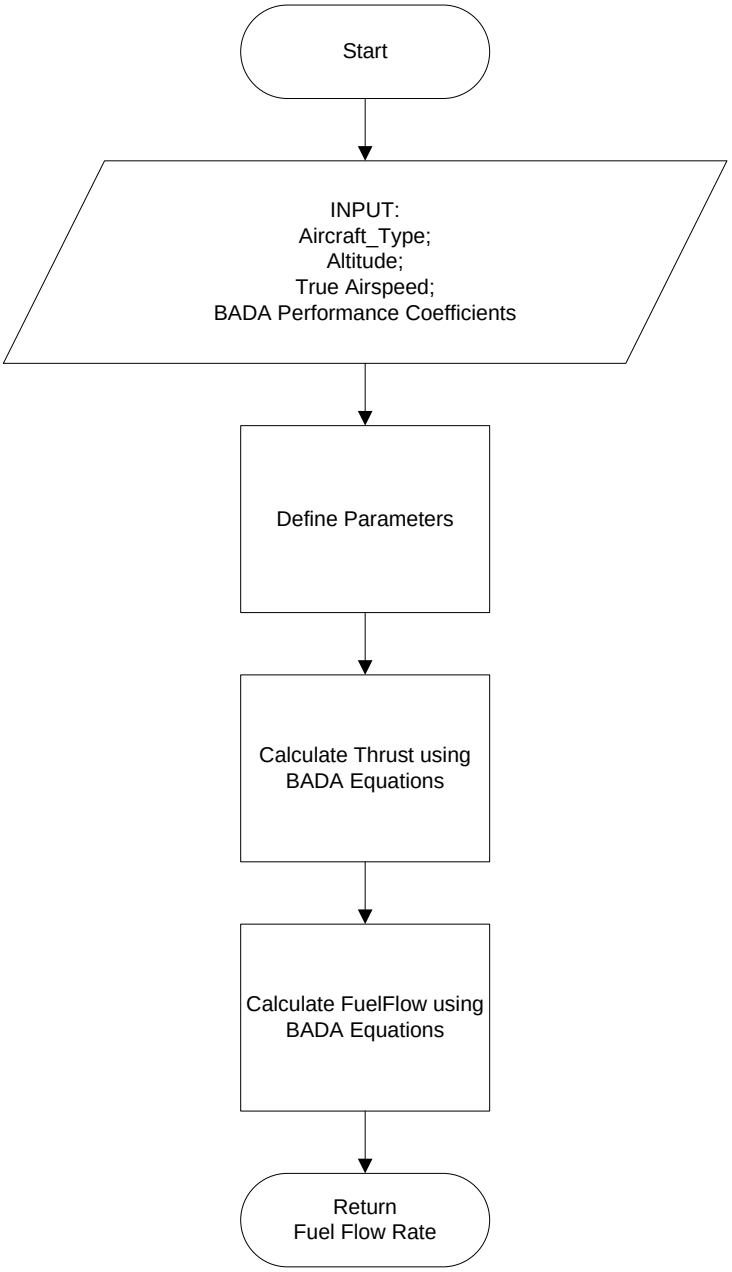
A.7 Speed_Calculator_Climb



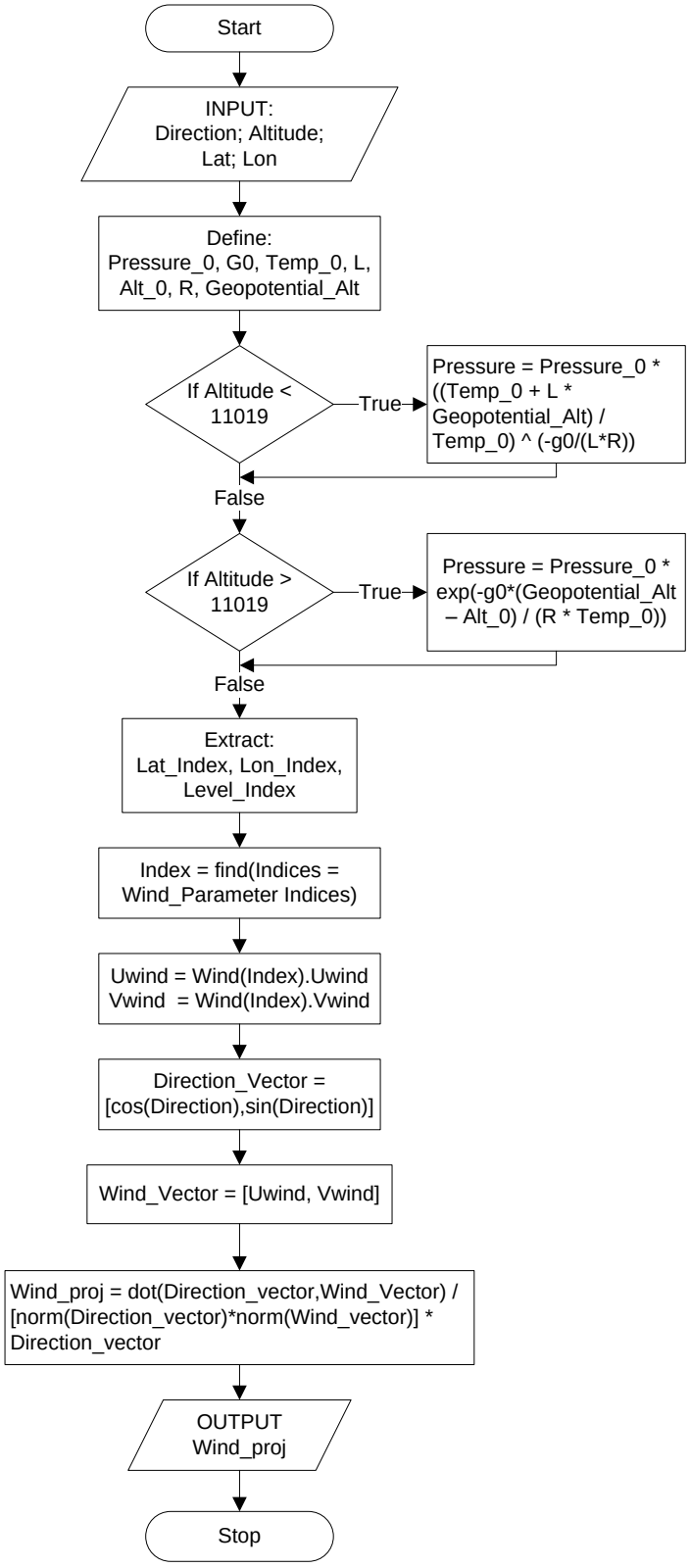
A.8 ROC_Calculator



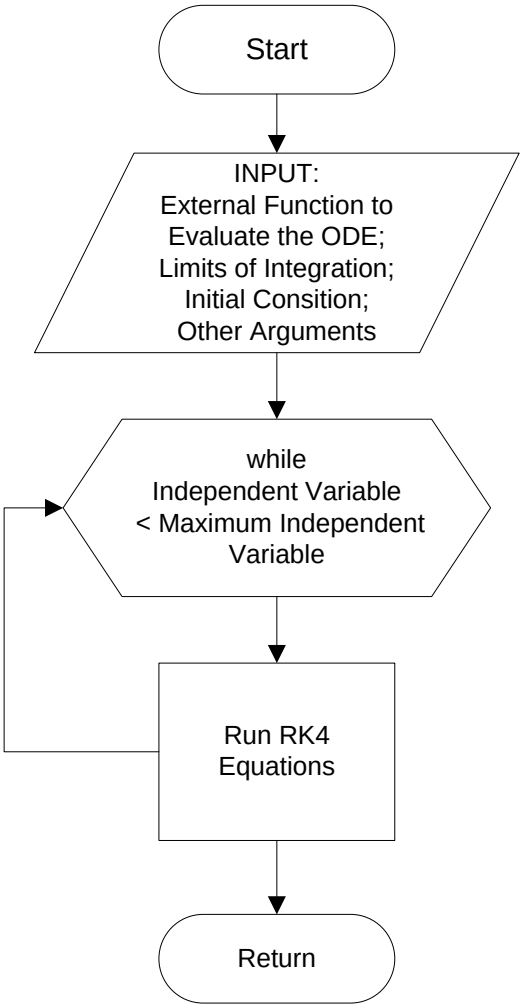
A.9 FuelFlow_Calculator_Climb



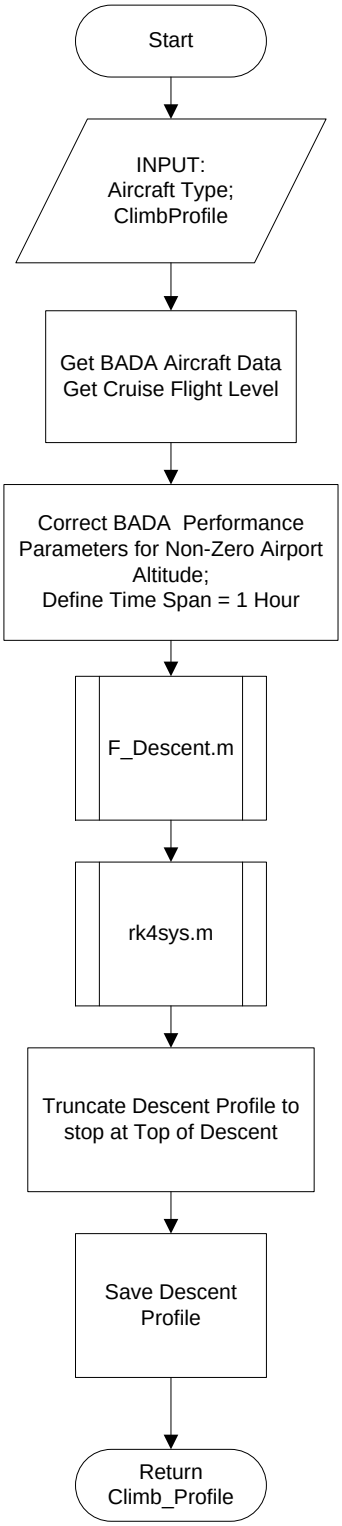
A.10 Wind_Calculator



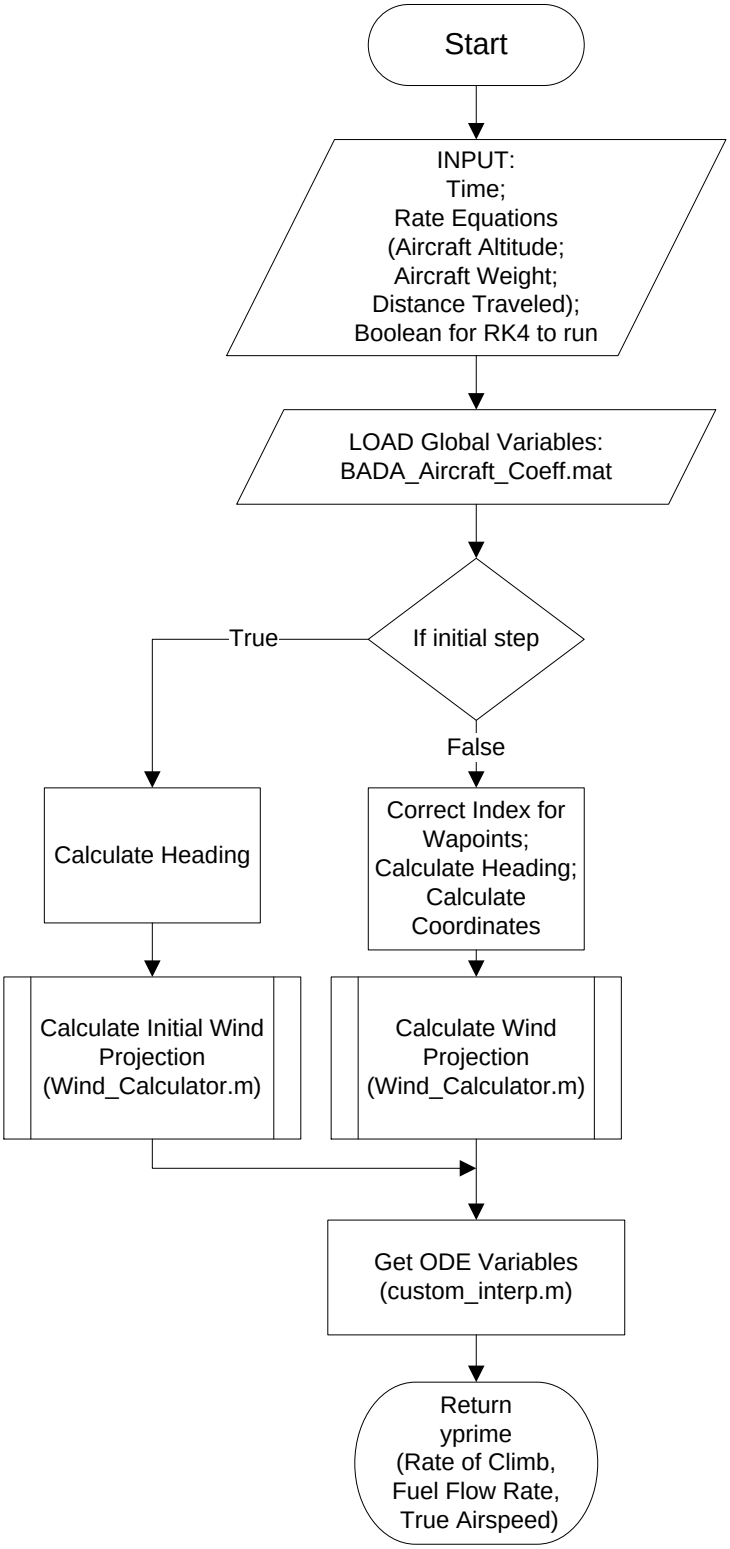
A.11 rk4sys


Start
INPUT:
External Function to
Evaluate the ODE;
Limits of Integration;
Initial Consition;
Other Arguments
while
Independent Variable
< Maximum Independent
Variable
Run RK4
Equations
Return

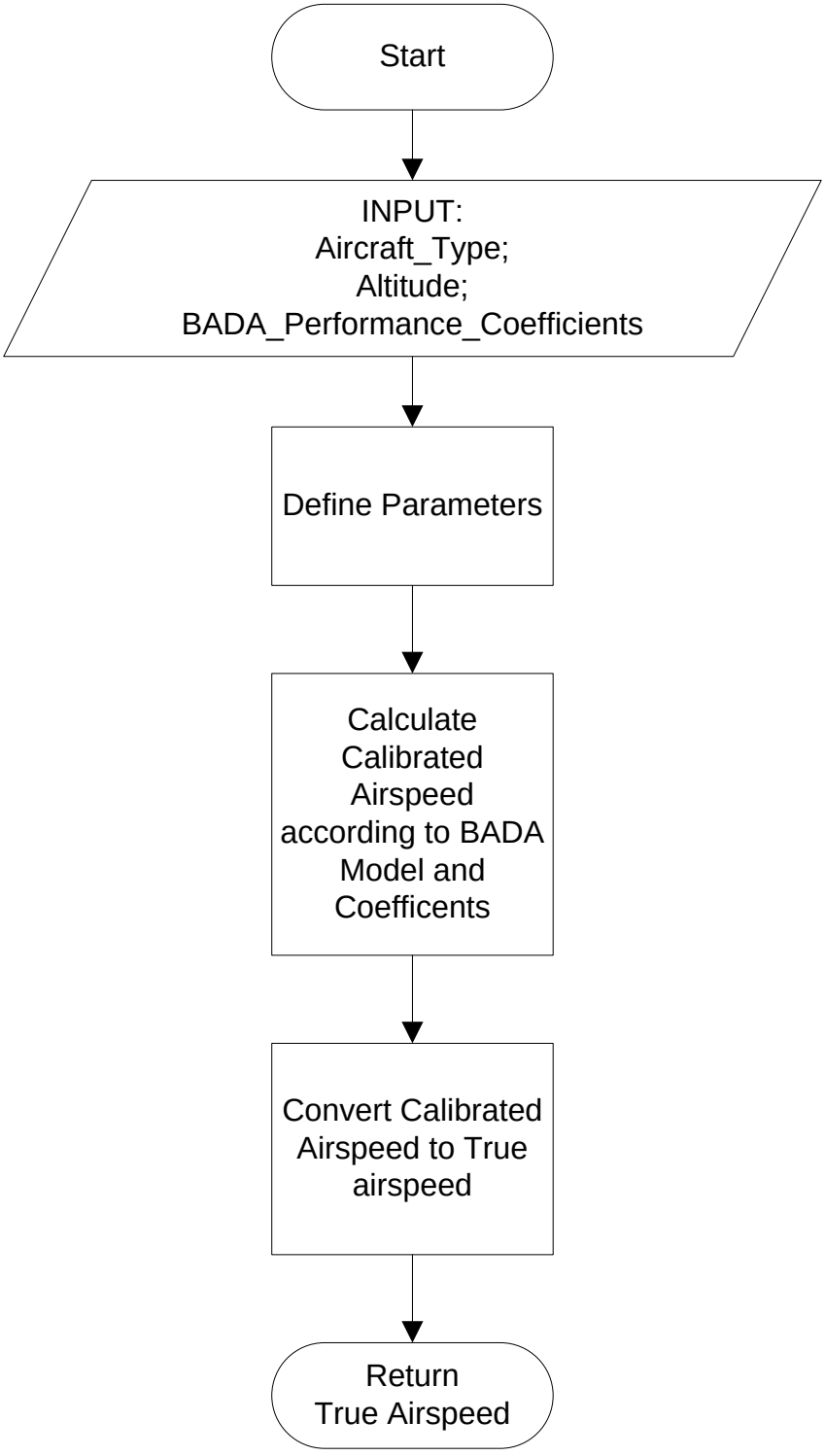
A.12 Generate_Descent_Profile



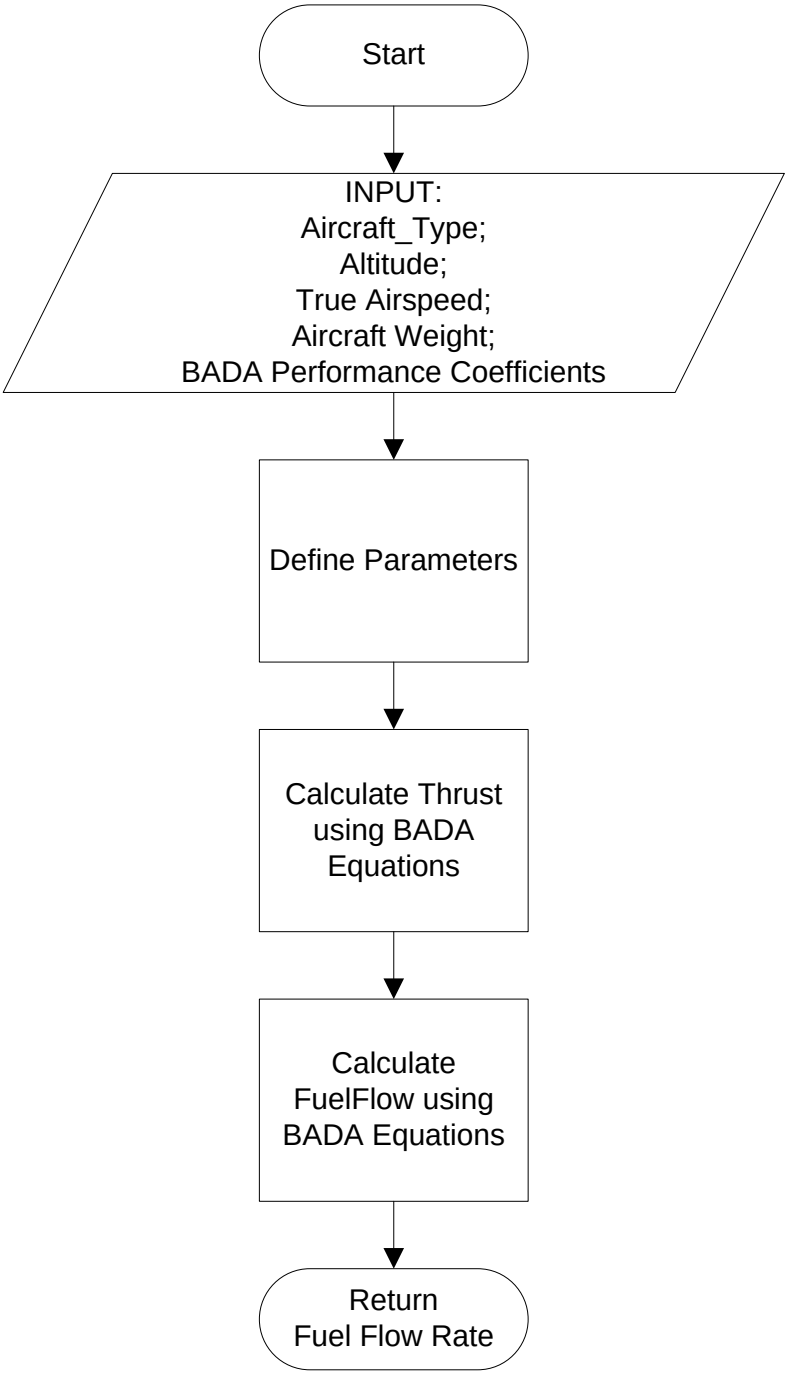
A.13 F_Descent



A.14 Generate_Cruise_Profile_Coef



A.15 FuelFlow_Calculator_Cruise



APPENDIX B: MATLAB Source Code

B.1 Flight Profile Generator

```
% THIS M-FILE GENERATES FLIGHT PROFILES FOR SPECIFIC FLIGHT TRACKS
% --------------
% RECORD START TIME
% --------------
time = clock;
month1 = time(2); date1 = time(3); year1 = time(1);
hour1 = time(4); minutes1 = time(5); seconds1 = time(6);
fprintf('\n%.0f/%.0f/%.0f\t%.0f:%02.0f:%02.0f\n',month1, date1, year1, hour1, minutes1, seconds1)
% ------------------------------------------------------------------------

% --------------
% DEFINE GLOBAL VARIABLES
% --------------
global BADA_Aircraft_Data BADA_Aircraft_List ETMS_BADA_List Dist_FL_Relation FL_Empirical_CDF
BADA_Aircraft_Coef
global Origin_Airport_Altitude Destination_Airport_Altitude Assn_Cruise_FL
global FeetInNauticalMiles Track_vs_Cruise Course Waypoints
global Wind_Boolean Wind Wind_Parameters % Month
global time_increment Aircraft_Index

global true_air_speed rate_of_desc fuel_flow

% NAUTICAL MILES TO FEET
FeetInNauticalMiles = 6076.11549;
time_increment = 1; % THE DESIRE STEP SIZE WHEN GENERATING CLIMB, CRUISE AND DESCENT
PROFILE (in minute)
% ------------------------------------------------------------------------

% --------------
% LOAD DATA
% --------------
load BADA_Aircraft_Data.mat
load BADA_Aircraft_List.mat
% load conus.mat
load Dist_FL_Relation.mat
load ETMS_Tracks.mat
load ETMS_Info.mat          % variable name: Dept_Arr_AC
load Track_vs_Cruise.mat
load ETMS_BADA_List.mat
load Wind.mat
load Wind_Parameters.mat
load FL_Empirical_CDF
load BADA_Aircraft_Coef.mat
% ------------------------------------------------------------------------

% --------------
% SPECIFY DIRECTORY
% --------------
```

```
%Output_Dir  = 'D:\fuelconsumption_and_emissions\Output2';
%BADA_Dir    = 'D:\fuelconsumption_and_emissions\bada_3_9\bada_39';
%Figures_Dir = 'D:\fuelconsumption_and_emissions\Figures3';
% ------------------------------------------------------------------------

% --------------
% WIND INCORPORATION
% --------------
% true = WIND; false = NO WIND
Wind_Boolean = true;
Month = 7; % TEST OF PROGRAM
% ------------------------------------------------------------------------

% Remove Unused Wind Months
Months_To_Remove = Wind.Time ~= Month;
Wind.Time(Months_To_Remove) = [];
Wind.Level(Months_To_Remove) = [];
Wind.Lat(Months_To_Remove) = [];
Wind.Lon(Months_To_Remove) = [];
Wind.Uwind(Months_To_Remove) = [];
Wind.Vwind(Months_To_Remove) = [];

% --------------
% YEAR LOOP
% --------------
Start_Year = 1995;
End_Year = 2008;

for Year_As_Num = Start_Year : End_Year


   % --------------
   % LOAD YEAR T100 TRACKS
   % --------------
   Year = int2str(Year_As_Num);
   disp('Loading T100_Tracks...')
   T100_Tracks_To_Load = sprintf('T100_Tracks_%s',Year);
   load(T100_Tracks_To_Load)
   disp('Finished Loading T100_Tracks.')
   % ------------------------------------------------------------------------

   % LOAD PDARS DETOUR FACTOR INFO
   load(['Airport_Status_DF_',Year,'.mat'])

   [NOT,Number_Of_Track_Entries] = size(T100_Tracks);

   ETMS_Waypoints = 0; % Count how many flights use ETMS_Waypoints
   Great_Circle_Points = 0; % Count how many flights use GCD_Points

   % -------------------------------------
   % FLIGHT PROFILE LOOP
   % -------------------------------------

   for Track_Number = 1:Number_Of_Track_Entries

      disp(Track_Number)
```

```
%       % BEGIN STOPWATCH
%       tic

      % DISPLAY PROGRESS EVERY 100 TRACKS
      if mod(Track_Number,100) == 0
         disp([num2str(Track_Number),'/',num2str(Number_Of_Track_Entries)])
      end

      % --------------
      % GET AIRCRAFT DATA
      % --------------
      Aircraft_Index = strcmp(T100_Tracks(Track_Number).BADA_Code,BADA_Aircraft_List);
      Current_Aircraft_Type = char(BADA_Aircraft_List(Aircraft_Index));

      if isempty(Aircraft_Index) == 1
         continue
      end

      Max_Cruise_FL = max(BADA_Aircraft_Data(Aircraft_Index).FL);
      Initial_Weight_kg = BADA_Aircraft_Data(Aircraft_Index).Nominal_Mass;

      % ---------------------------------------------------------------

      % --------------
      % GET AIRPORT DATA
      % --------------
      % ORIGIN AIRPORT
      Origin                        = T100_Tracks(Track_Number).Origin_Airport_ID;
      Origin_Airport_Altitude       = T100_Tracks(Track_Number).Origin_Airport_Altitude;
      Airport_Origin_Longitude      = T100_Tracks(Track_Number).Airport_Origin_Longitude;
      Airport_Origin_Latitude       = T100_Tracks(Track_Number).Airport_Origin_Latitude;

      % DESTINATION AIRPORT
      Destination                   = T100_Tracks(Track_Number).Destination_Airport_ID;
      Destination_Airport_Altitude  = T100_Tracks(Track_Number).Destination_Airport_Altitude;
      Airport_Destination_Longitude = T100_Tracks(Track_Number).Airport_Destination_Longitude;
      Airport_Destination_Latitude  = T100_Tracks(Track_Number).Airport_Destination_Latitude;
      % ---------------------------------------------------------------

      % --------------
      % CALCULATE TRACK DISTANCE AND COURSE
      % --------------
      % EXTRACT AIRPORT INFO FROM ETMS DATA
      Dept_Airport_Comp = strcmp(Origin, Dept_Arr_AC(:,1));
      Arr_Airport_Comp = strcmp(Destination, Dept_Arr_AC(:,2));

      Find_Airport = find(Dept_Airport_Comp == 1 & Arr_Airport_Comp == 1, 1, 'first');

      % FIND DISTANCE BETWEEN AIRPORTS
      if isempty(Find_Airport) == 0

         % USE ETMS DATA FIRST
         Course = ETMS_Tracks(Find_Airport).Course;
         Track_Distance_nm = ETMS_Tracks(Find_Airport).Waypoint_Distance_nm;
         Waypoints = [ETMS_Tracks(Find_Airport).Latitude, ETMS_Tracks(Find_Airport).Longitude];
```

```
        ETMS_Waypoints = ETMS_Waypoints + 1;

        if isempty(Waypoints) == 1
           % FALL BACK ON GC CALCULATIONS IF THERE ARE NO ETMS WAYPOINTS
           [Course Track_Distance_nm] = legs([Airport_Origin_Latitude Airport_Destination_Latitude],...
              [Airport_Origin_Longitude Airport_Destination_Longitude], 'gc');
           [lat, lon] =
gcwaypts(Airport_Origin_Latitude,Airport_Origin_Longitude,Airport_Destination_Latitude,Airport_Destination_L
ongitude);
           Waypoints = [lat,lon];
           GCD_Great_Circle_Points = Great_Circle_Points + 1;
        end

     else

        % FALL BACK ON GC CALCULATIONS IF THERE IS NO ETMS DATA
        [Course Track_Distance_nm] = legs([Airport_Origin_Latitude Airport_Destination_Latitude],...
           [Airport_Origin_Longitude Airport_Destination_Longitude], 'gc');
        [lat, lon] =
gcwaypts(Airport_Origin_Latitude,Airport_Origin_Longitude,Airport_Destination_Latitude,Airport_Destination_L
ongitude);
        Waypoints = [lat,lon];
        GCD_Great_Circle_Points = Great_Circle_Points + 1;

     end

     % ASSIGN COURSE TO NAVIGATIONAL DIRECTION
     if Course>=0 && Course<180
        Heading = 'EAST';
     elseif Course>=180 && Course<360
        Heading = 'WEST';
     end
     % -----------------------------------------------------------------

     % ---------------------------------------------------------------------------------------------------------------
     % GENERATE FULL FLIGHT PROFILE
     % ---------------------------------------------------------------------------------------------------------------


        [Climb_And_Descent_Distance Flight_Profile] =
Flight_Profile_Rand_FL(Aircraft_Index,Heading,Track_Distance_nm,Max_Cruise_FL,Initial_Weight_kg);


     if isempty(Flight_Profile) == 1
        [Climb_And_Descent_Distance Flight_Profile] =
Flight_Profile_Assn_FL(Aircraft_Index,Heading,Track_Distance_nm,Max_Cruise_FL,Initial_Weight_kg);
     end

     % ---------------------------------------------------------------------------------------------------------------

     % SEPARATE PROFILES
     Climb_Profile = Flight_Profile.Climb_Profile;
     Cruise_Profile = Flight_Profile.Cruise_Profile;
     Descent_Profile = Flight_Profile.Descent_Profile;
     % -----------------------------------------------------------------
```

```
    % CALCULATE TAKEOFF FUEL BURN
    Takeoff_Fuel_Burn_kg = Takeoff_Fuel_Calculator(Current_Aircraft_Type);
    % ------------------------------------------------------------------------------------------------

    % --------------
    % INCORPORATE PDARS DETOUR FACTOR
    % --------------
    Dep_Index = find(strcmp(Airport_Status_DF(:,4),[Origin,'Dep'])==1);
    Arr_Index = find(strcmp(Airport_Status_DF(:,4),[Origin,'Arr'])==1);
    Climb_DF = cell2mat(Airport_Status_DF(Dep_Index,3));
    Descent_DF = cell2mat(Airport_Status_DF(Arr_Index,3));

    Climb_Profile.Total_Fuel_kg = Climb_Profile.Total_Fuel_kg*Climb_DF;
    Climb_Profile.Distance_For_Climb_nm = Climb_Profile.Distance_For_Climb_nm*Climb_DF;
    Descent_Profile.Total_Fuel_kg = Descent_Profile.Total_Fuel_kg*Descent_DF;
    Descent_Profile.Distance_For_Descent_nm = Descent_Profile.Distance_For_Descent_nm*Descent_DF;

    Flight_Profile.Climb_Profile.Total_Fuel_kg = Flight_Profile.Climb_Profile.Total_Fuel_kg*Climb_DF;
    Flight_Profile.Climb_Profile.Distance_For_Climb_nm =
Flight_Profile.Climb_Profile.Distance_For_Climb_nm*Climb_DF;
    Flight_Profile.Descent_Profile.Total_Fuel_kg = Flight_Profile.Descent_Profile.Total_Fuel_kg*Descent_DF;
    Flight_Profile.Descent_Profile.Distance_For_Descent_nm =
Flight_Profile.Descent_Profile.Distance_For_Descent_nm*Descent_DF;

    Profile_Track_Distance = Climb_Profile.Distance_For_Climb_nm + Cruise_Profile.Cruise_Distance_nm +
Descent_Profile.Distance_For_Descent_nm;
    % ------------------------------------------------------------------

    % --------------
    % SAVE PROFILES IN STRUCTURE
    % --------------
    Tracks_Profile(Track_Number).Origin_Airport      = Origin;
    Tracks_Profile(Track_Number).Destination_Airport = Destination;
    Tracks_Profile(Track_Number).Aircraft_Type       = Current_Aircraft_Type;

    Tracks_Profile(Track_Number).Climb_Profile       = Climb_Profile;
    Tracks_Profile(Track_Number).Cruise_Profile      = Cruise_Profile;
    Tracks_Profile(Track_Number).Descent_Profile     = Descent_Profile;

    Tracks_Profile(Track_Number).Takeoff_Fuel_Burn_kg = Takeoff_Fuel_Burn_kg;

    Total_Fuel = Flight_Profile.Climb_Profile.Total_Fuel_kg + Flight_Profile.Cruise_Profile.Total_Fuel_kg +
Flight_Profile.Descent_Profile.Total_Fuel_kg + Takeoff_Fuel_Burn_kg;
    Tracks_Profile(Track_Number).Yearly_Frequency      = T100_Tracks(Track_Number).Yearly_Frequency;
    Tracks_Profile(Track_Number).Fuel_Consumption_kg   = T100_Tracks(Track_Number).Yearly_Frequency *
Total_Fuel;
    Tracks_Profile(Track_Number).Track_Distance_nm     = Profile_Track_Distance;

    Tracks_Profile(Track_Number).Waypt_Latitude    = ETMS_Tracks(Find_Airport).Latitude;
    Tracks_Profile(Track_Number).Waypt_Longitude   = ETMS_Tracks(Find_Airport).Longitude;
    Tracks_Profile(Track_Number).Heading = Heading;
    % ------------------------------------------------------------------

    close all

%       % END STOPWATCH
```

```
%         toc

    end % for Track_Number = 1:Number_Of_Track_Entries
    % ------------------------------------------------------------------

    % --------------
    % SAVE STRUCTURE
    % --------------

     save (['Tracks_Profile', '_', Year, '.mat'], 'Tracks_Profile');

end % for Year_As_Num = Start_Year : End_Year
% ------------------------------------------------------------------------

% --------------
% RECORD END TIME
% --------------
time = clock;
month2 = time(2); date2 = time(3); year2 = time(1);
hour2 = time(4); minutes2 = time(5); seconds2 = time(6);
fprintf('\nStart Time:\n%.0f/%.0f/%.0f\t%.0f:%02.0f:%02.0f\n',month1, date1, year1, hour1, minutes1, seconds1)
fprintf('\nEnd Time:\n%.0f/%.0f/%.0f\t%.0f:%02.0f:%02.0f\n',month2, date2, year2, hour2, minutes2, seconds2)
% ------------------------------------------------------------------------
```

B.2 Flight_Profile_Rand_FL.m

```
% This function runs through the flight profile of the aircraft/airport
% combination by assigning a random cruise altitude
%
% [Climb_And_Descent_Distance, Flight_Profile] =
Flight_Profile_Rand_FL(Aircraft_Index,Heading,Track_Distance_nm,Max_Cruise_FL)
%
% INPUT:
%   Aircraft_Index      = index of AC/AP combo from T100 tracks structure
%   Heading             = general direction of path, EAST/WEST
%   Track_Distance_nm   = distance along track
%   Max_Cruise_FL       = aircraft possible altitude
%
% OUTPUT:
%   Climb_And_Descent_Distance  = track distance minus the cruise distance
%   Flight_Profile              = profile of aircraft/airport combo

function [Climb_And_Descent_Distance, Flight_Profile] =
Flight_Profile_Rand_FL(Aircraft_Index,Heading,Track_Distance_nm,Max_Cruise_FL,Initial_Weight_kg)

% --------------
% DEFINE GLOBAL VARIABLES
% --------------
global Track_vs_Cruise Assn_Cruise_FL BADA_Aircraft_List ETMS_BADA_List FL_Empirical_CDF
% ------------------------------------------------------------------------

% --------------
% GET AIRCRAFT AND TRACK DATA
% --------------
Current_Aircraft_Type = char(BADA_Aircraft_List(Aircraft_Index));
ETMS_BADA_Match = find(strcmp(Current_Aircraft_Type(1:4), ETMS_BADA_List) == 1);

% ROUND TRACK DISTANCE TO NEAREST TEN FOR USE WITH ETMS DATA
Rounded_Track_Dist = round(Track_Distance_nm/10)*10;
% ------------------------------------------------------------------------

% --------------
% CHECK FOR ETMS MATCH
% --------------
if isempty(ETMS_BADA_Match) == 1;
   Flight_Profile = [];
   Climb_And_Descent_Distance = [];
   return
end
% ------------------------------------------------------------------------
% VERY SHORT DISTANCE FLIGHT
% ------------------------------------------------------------------------
if Track_Distance_nm <= 5.1;
   Flight_Profile = [];
   Climb_And_Descent_Distance = [];
   return
end
% ------------------------------------------------------------------------
```

```
% --------------
% GENERATE RANDOM FLIGHT LEVEL
% --------------
% EAST/WEST DIRECTIONS RESULT IN DIFFERENT CRUISE ALTITUDES
if strcmp('EAST',Heading) == 1 % HEADING EAST
   % EXTRACT DATA FROM ETMS
   East_Track = Track_vs_Cruise(ETMS_BADA_Match).East_Track;
   East_Track_Pts = Track_vs_Cruise(ETMS_BADA_Match).East_Track_Pts;
   East_Min_FL_Pts = Track_vs_Cruise(ETMS_BADA_Match).East_Min_FL_Pts;
   East_Max_FL_Pts = Track_vs_Cruise(ETMS_BADA_Match).East_Max_FL_Pts;
   % LOAD EMPIRICAL CDF FROM FL_EMPIRICAL_CDF.MAT (EXTRACTED FROM ETMS DATA,
MANUALLY MAINTAINED)
   East_FL_2nd_Normalized = FL_Empirical_CDF(1,2).East_FL_Normalized; % Normalized Fligh Levels
   East_F_2nd = FL_Empirical_CDF(1,2).East_F; % CDF Corresponding to the Normalized Fligh Levels
   East_FL_3rd_Normalized = FL_Empirical_CDF(1,3).East_FL_Normalized;
   East_F_3rd = FL_Empirical_CDF(1,3).East_F;
   East_FL_4th_Normalized = FL_Empirical_CDF(1,4).East_FL_Normalized;
   East_F_4th = FL_Empirical_CDF(1,4).East_F;

%%
   if isempty(East_Track_Pts) == 1
      Flight_Profile = [];
      Climb_And_Descent_Distance = [];
      return
   elseif Rounded_Track_Dist < East_Track_Pts(1) || Rounded_Track_Dist > East_Track_Pts(end)
      Flight_Profile = [];
      Climb_And_Descent_Distance = [];
      return
% ------------------------------------------------------------------------------------------------
% TRACK DISTANCE IS IN FIRST QUARTILE, USE UNIFORM DISTRIBUTION
   elseif Track_Distance_nm <= 500
      % GET RANDOM FLIGHT LEVEL BY UNIFORM DISTRIBUTION
      Index_ETMS_FL = find(Rounded_Track_Dist == East_Track_Pts);
      Min_FL = East_Min_FL_Pts(Index_ETMS_FL);
      Max_FL = East_Max_FL_Pts(Index_ETMS_FL);
      if Min_FL > Max_FL
         Raw_Cruise_FL = randi([round(Max_FL) round(Min_FL)]);
      else
         Raw_Cruise_FL = randi([round(Min_FL) round(Max_FL)]);
      end
      if Raw_Cruise_FL > Max_Cruise_FL
         Raw_Cruise_FL = Max_Cruise_FL;
      end
% ------------------------------------------------------------------------------------------------ ----------
----------
% TRACK DISTANCE IS IN SECOND QUARTILE, USE THE SECOND SECTION OF EMPIRICAL CDF
DISTRIBUTION
   elseif Track_Distance_nm <= 1000
       % GET RANDOM FLIGHT LEVEL BY EMPIRICAL CDF DISTRIBUTION IN 2ND QUARTILE
       East_FL_2nd = East_FL_2nd_Normalized .* Max_Cruise_FL;
       Random_Number = rand(1);
       East_FL_2nd_idx = find(East_F_2nd>=Random_Number,1,'first');
       Raw_Cruise_FL = East_FL_2nd(East_FL_2nd_idx);
%        Raw_Cruise_FL_temp = round(East_FL_2nd(East_FL_2nd_idx)/10)*10;
%        if mod(Raw_Cruise_FL_temp,20) == 0
%            Raw_Cruise_FL = Raw_Cruise_FL_temp + 10;
```

```
%        else
%            Raw_Cruise_FL = Raw_Cruise_FL_temp;
%        end
       if Raw_Cruise_FL > Max_Cruise_FL
         Raw_Cruise_FL = Max_Cruise_FL;
       end
% ------------------------------------------------------------------------------------------------------------------------
% TRACK DISTANCE IS IN THIRD QUARTILE, USE THE THIRD SECTION OF EMPIRICAL CDF
DISTRIBUTION
   elseif Track_Distance_nm <= 1500
       % GET RANDOM FLIGHT LEVEL BY EMPIRICAL CDF DISTRIBUTION IN 3RD QUARTILE
       East_FL_3rd = East_FL_3rd_Normalized .* Max_Cruise_FL;
       Random_Number = rand(1);
       East_FL_3rd_idx = find(East_F_3rd>=Random_Number,1,'first');
       Raw_Cruise_FL = East_FL_3rd(East_FL_3rd_idx);
%        Raw_Cruise_FL_temp = round(East_FL_3rd(East_FL_3rd_idx)/10)*10;
%        if mod(Raw_Cruise_FL_temp,20) == 0
%            Raw_Cruise_FL = Raw_Cruise_FL_temp + 10;
%        else
%            Raw_Cruise_FL = Raw_Cruise_FL_temp;
%        end
       if Raw_Cruise_FL > Max_Cruise_FL
         Raw_Cruise_FL = Max_Cruise_FL;
       end
% ------------------------------------------------------------------------------------------------------------------------
% TRACK DISTANCE IS IN FOURTH QUARTILE, USE THE FOURTH SECTION OF EMPIRICAL CDF
DISTRIBUTION
   else
       % GET RANDOM FLIGHT LEVEL BY EMPIRICAL CDF DISTRIBUTION IN 4TH QUARTILE
       East_FL_4th = East_FL_4th_Normalized .* Max_Cruise_FL;
       Random_Number = rand(1);
       East_FL_4th_idx = find(East_F_4th>=Random_Number,1,'first');
       Raw_Cruise_FL = East_FL_4th(East_FL_4th_idx);
%         Raw_Cruise_FL_temp = round(East_FL_4th(East_FL_4th_idx)/10)*10;
%        if mod(Raw_Cruise_FL_temp,20) == 0
%            Raw_Cruise_FL = Raw_Cruise_FL_temp + 10;
%        else
%            Raw_Cruise_FL = Raw_Cruise_FL_temp;
%        end
       if Raw_Cruise_FL > Max_Cruise_FL
         Raw_Cruise_FL = Max_Cruise_FL;
       end
   end


elseif strcmp('WEST',Heading) == 1 % HEADING WEST
   % EXTRACT DATA FROM ETMS
   West_Track = Track_vs_Cruise(ETMS_BADA_Match).West_Track;
   West_Track_Pts = Track_vs_Cruise(ETMS_BADA_Match).West_Track_Pts;
   West_Min_FL_Pts = Track_vs_Cruise(ETMS_BADA_Match).West_Min_FL_Pts;
   West_Max_FL_Pts = Track_vs_Cruise(ETMS_BADA_Match).West_Max_FL_Pts;
   % LOAD EMPIRICAL CDF FROM FL_EMPIRICAL_CDF.MAT (EXTRACTED FROM ETMS DATA,
MANUALLY MAINTAINED)
   West_FL_2nd_Normalized = FL_Empirical_CDF(1,2).West_FL_Normalized;
```

```
    West_F_2nd = FL_Empirical_CDF(1,2).West_F;
    West_FL_3rd_Normalized = FL_Empirical_CDF(1,3).West_FL_Normalized;
    West_F_3rd = FL_Empirical_CDF(1,3).West_F;
    West_FL_4th_Normalized = FL_Empirical_CDF(1,4).West_FL_Normalized;
    West_F_4th = FL_Empirical_CDF(1,4).West_F;
 %%
    if isempty(West_Track_Pts) == 1
        Flight_Profile = [];
        Climb_And_Descent_Distance = [];
        return
    elseif Rounded_Track_Dist < West_Track_Pts(1) || Rounded_Track_Dist > West_Track_Pts(end)
        Flight_Profile = [];
        Climb_And_Descent_Distance = [];
        return
% ------------------------------------------------------------------------------------------------------
% TRACK DISTANCE IS IN FIRST QUARTILE, USE UNIFORM DISTRIBUTION
    elseif Track_Distance_nm <= 500
        % GET RANDOM FLIGHT LEVEL BY UNIFORM DISTRIBUTION
        Index_ETMS_FL = find(Rounded_Track_Dist == West_Track_Pts);
        Min_FL = West_Min_FL_Pts(Index_ETMS_FL);
        Max_FL = West_Max_FL_Pts(Index_ETMS_FL);
        if Min_FL > Max_FL
            Raw_Cruise_FL = randi([round(Max_FL) round(Min_FL)]);
        else
            Raw_Cruise_FL = randi([round(Min_FL) round(Max_FL)]);
        end
        if Raw_Cruise_FL > Max_Cruise_FL
          Raw_Cruise_FL = Max_Cruise_FL;
        end
% ------------------------------------------------------------------------------------------------------------------------
% TRACK DISTANCE IS IN SECOND QUARTILE, USE THE SECOND SECTION OF EMPIRICAL CDF
DISTRIBUTION
    elseif Track_Distance_nm <= 1000
         % GET RANDOM FLIGHT LEVEL BY EMPIRICAL CDF DISTRIBUTION IN 2ND QUARTILE
         West_FL_2nd = West_FL_2nd_Normalized .* Max_Cruise_FL;
         Random_Number = rand(1);
         West_FL_2nd_idx = find(West_F_2nd>=Random_Number,1,'first');
         %Raw_Cruise_FL = West_FL_2nd(West_FL_2nd_idx);
        Raw_Cruise_FL_temp = round(West_FL_2nd(West_FL_2nd_idx)/10)*10;
        if mod(Raw_Cruise_FL_temp,20) ~= 0
            Raw_Cruise_FL = Raw_Cruise_FL_temp + 10;
        else
            Raw_Cruise_FL = Raw_Cruise_FL_temp;
        end
        if Raw_Cruise_FL > Max_Cruise_FL
          Raw_Cruise_FL = Max_Cruise_FL;
        end
% ------------------------------------------------------------------------------------------------------------------------
% TRACK DISTANCE IS IN THIRD QUARTILE, USE THE THIRD SECTION OF EMPIRICAL CDF
DISTRIBUTION
    elseif Track_Distance_nm <= 1500
         % GET RANDOM FLIGHT LEVEL BY EMPIRICAL CDF DISTRIBUTION IN 3RD QUARTILE
         West_FL_3rd = West_FL_3rd_Normalized .* Max_Cruise_FL;
         Random_Number = rand(1);
```

```
        West_FL_3rd_idx = find(West_F_3rd>=Random_Number,1,'first');
        Raw_Cruise_FL = West_FL_3rd(West_FL_3rd_idx);
%         Raw_Cruise_FL_temp = round(West_FL_3rd(West_FL_3rd_idx)/10)*10;
%         if mod(Raw_Cruise_FL_temp,20) ~= 0
%             Raw_Cruise_FL = Raw_Cruise_FL_temp + 10;
%         else
%             Raw_Cruise_FL = Raw_Cruise_FL_temp;
%         end
        if Raw_Cruise_FL > Max_Cruise_FL
          Raw_Cruise_FL = Max_Cruise_FL;
        end
% ------------------------------------------------------------------------------------------------------------------------
% TRACK DISTANCE IS IN FOURTH QUARTILE, USE THE FOURTH SECTION OF EMPIRICAL CDF DISTRIBUTION
    else
        % GET RANDOM FLIGHT LEVEL BY EMPIRICAL CDF DISTRIBUTION IN 4TH QUARTILE
        West_FL_4th = West_FL_4th_Normalized .* Max_Cruise_FL;
        Random_Number = rand(1);
        West_FL_4th_idx = find(West_F_4th>=Random_Number,1,'first');
        Raw_Cruise_FL = West_FL_4th(West_FL_4th_idx);
%         Raw_Cruise_FL_temp = round(West_FL_4th(West_FL_4th_idx)/10)*10;
%         if mod(Raw_Cruise_FL_temp,20) ~= 0
%             Raw_Cruise_FL = Raw_Cruise_FL_temp + 10;
%         else
%             Raw_Cruise_FL = Raw_Cruise_FL_temp;
%         end
        if Raw_Cruise_FL > Max_Cruise_FL
          Raw_Cruise_FL = Max_Cruise_FL;
        end
    end
end
% ------------------------------------------------------------------------

% --------------
% ASSIGN ATC FL
% --------------
Assn_Cruise_FL = Calculate_FL(Raw_Cruise_FL, Heading, Max_Cruise_FL);

% CONVERT FL TO FEET
Assn_Cruise_FL = Assn_Cruise_FL * 100;
% ------------------------------------------------------------------------

% --------------
% PROFILE LOOP
% --------------
while(1)

    % --------------
    % GENERATE CLIMB PROFILE
    % --------------
    Climb_Profile = Generate_Climb_Profile(Aircraft_Index,Initial_Weight_kg);
    % ------------------------------------------------------------------

    % --------------
    % GENERATE DESCENT PROFILE
```

```
   % --------------
   Descent_Profile = Generate_Descent_Profile(Aircraft_Index, Climb_Profile);
   % ------------------------------------------------------------------------

   % --------------
   % GENERATE CRUISE PROFILE
   % --------------
   Climb_And_Descent_Distance = Climb_Profile.Distance_For_Climb_nm +
Descent_Profile.Distance_For_Descent_nm;
   Cruise_Distance_nm = Track_Distance_nm - Climb_And_Descent_Distance;

   % CHECK THAT CRUISE DISTANCE EXISTS
   if sign(Cruise_Distance_nm) == -1
      if Assn_Cruise_FL < 3000
         Assn_Cruise_FL = Assn_Cruise_FL - 500;
      elseif Assn_Cruise_FL >= 41000
         Assn_Cruise_FL = Assn_Cruise_FL - 4000;
      else
         Assn_Cruise_FL = Assn_Cruise_FL - 2000;
      end
      continue
   end

   Cruise_Profile = Generate_Cruise_Profile_Coef(Aircraft_Index, Climb_Profile, Descent_Profile,
Cruise_Distance_nm);
   % ------------------------------------------------------------------------

   % --------------
   % DECREASE CRUISE FL IF EMPTY CRUISE
   % --------------
   if isempty(Cruise_Profile) == 0
      break
   else
      if Assn_Cruise_FL < 3000
         Assn_Cruise_FL = Assn_Cruise_FL - 500;
      elseif Assn_Cruise_FL >= 41000
         Assn_Cruise_FL = Assn_Cruise_FL - 4000;
      else
         Assn_Cruise_FL = Assn_Cruise_FL - 2000;
      end
   end
   % ------------------------------------------------------------------------

end % while(1)
% ------------------------------------------------------------------------

% --------------
% CORRECT DESCENT PROFILE
% --------------
Initial_Descent_Weight               = Cruise_Profile.Weight_kg(end);
Descent_Profile.Weight_kg              = Descent_Profile.Weight_kg + Initial_Descent_Weight;
Descent_Profile.Time_hrs               = Descent_Profile.Time_hrs + Cruise_Profile.Time_hrs(end);
Descent_Profile.Distance_nm            = Descent_Profile.Distance_nm + Cruise_Profile.Distance_nm(end);
Descent_Profile.Distance_nm_For_Speed   = Descent_Profile.Distance_nm_For_Speed +
Cruise_Profile.Distance_nm(end);
% ------------------------------------------------------------------------
```

```
% --------------
% CREATE FLIGHT PROFILE STRUCTURE
% --------------
Flight_Profile.Climb_Profile    = Climb_Profile;
Flight_Profile.Cruise_Profile   = Cruise_Profile;
Flight_Profile.Descent_Profile  = Descent_Profile;
% ------------------------------------------------------------------------
return
```

B.3 Calculate_FL.m

```
% This function calculates the cruise flight level of an aircraft depending
% on its heading and track distance between two airports
%
% [Assn_Cruise_FL Cruise_FL] = Calculate_FL(Track_Distance_nm, Heading, Aircraft, Max_Cruise_FL)
%
% INPUT:
%   Track_Distance_nm   = distance along track
%   Heading             = general direction of path, EAST/WEST
%   Aircraft            = BADA aircraft being considered
%   Max_Cruise_FL       = aircraft possible altitude
%
% OUTPUT:
%   Assn_Cruise_FL      = the assigned cruise FL, rounded
%   Cruise_FL           = the unrounded or maximum aircraft FL

function Assn_Cruise_FL = Calculate_FL(Raw_Cruise_FL, Heading, Max_Cruise_FL)

% --------------
% DEFINE GLOBAL VARIABLES
% --------------
global Dist_FL_Relation
% ------------------------------------------------------------------------

% --------------
% GET FLIGHT LEVEL DATA
% --------------
% Field List:
% Field 1: Aircraft
% Field 2: EAST_Total_Dist
% Field 3: EAST_Max_FL
% Field 4: EAST_Dist_Max_FL
% Field 5: EAST_Slope
% Field 6: WEST_Total_Dist
% Field 7: WEST_Max_FL
% Field 8: WEST_Dist_Max_FL
% Field 9: WEST_Slope

BADA_AC = Dist_FL_Relation{1};
EAST_Slope = Dist_FL_Relation{5};
WEST_Slope = Dist_FL_Relation{9};

Flight_Assn_East_Lo = [30;50;70;90;110;130;150;170;190;210;230;250;270;...
                290;310;330;350;370;390;410];
Flight_Assn_West_Lo = [30;40;60;80;100;120;140;160;180;200;220;240;260;...
                280;300;320;340;360;380;400];
Flight_Assn_East_Hi = [410;450;490;530;570;610];
Flight_Assn_West_Hi = [400;430;570;510;550;590];
% ------------------------------------------------------------------------

% AC_Index = strmatch(Aircraft, BADA_AC, 'exact');

% % --------------
% % GET HEADING
```

```
% % --------------
% if Course>=0 && Course<180
%     Slope = EAST_Slope(AC_Index);
%     Heading = 'EAST';
% elseif Course>=180 && Course<360
%     Slope = WEST_Slope(AC_Index);
%     Heading = 'WEST';
% end
% % ------------------------------------------------------------------------

% --------------
% ASSIGN FL FROM ATC RULES
% --------------
if Raw_Cruise_FL <= 30
   Assn_Cruise_FL = Raw_Cruise_FL;
elseif Raw_Cruise_FL > 30
   if strcmp('EAST', Heading) == 1
      [no, Row] = min(abs(Flight_Assn_East_Lo - Raw_Cruise_FL));
      Assn_Cruise_FL = Flight_Assn_East_Lo(Row);
   elseif strcmp('WEST', Heading) == 1
      [no, Row] = min(abs(Flight_Assn_West_Lo - Raw_Cruise_FL));
      Assn_Cruise_FL = Flight_Assn_West_Lo(Row);
   end
elseif Raw_Cruise_FL > 410
   if strcmp('EAST', Heading) == 1
      [no, Row] = min(abs(Flight_Assn_East_Hi - Raw_Cruise_FL));
      Assn_Cruise_FL = Flight_Assn_East_Hi(Row);
   elseif strcmp('WEST', Heading) == 1
      [no, Row] = min(abs(Flight_Assn_West_Hi - Raw_Cruise_FL));
      Assn_Cruise_FL = Flight_Assn_West_Hi(Row);
   end
end
% ------------------------------------------------------------------------

% --------------
% CHECK THAT ASSIGNED FL < MAX FL OF AIRCRAFT
% --------------
if Assn_Cruise_FL >= Max_Cruise_FL
   Assn_Cruise_FL = Max_Cruise_FL;
end

if Assn_Cruise_FL < 0
   Assn_Cruise_FL = 0;
end
% ------------------------------------------------------------------------
return
```

B.4 Generate_Climb_Profile.m

```
% This function generates the climb profile of the aircraft/airport
% combination being called
%
% Climb_Profile = Generate_Climb_Profile(Aircraft_Index)
%
% INPUT:
%   Aircraft_Index = index of AC/AP combo from T100 tracks structure
%
% OUTPUT:
%   Climb_Profile  =  motion profile of aircraft's climb

function Climb_Profile = Generate_Climb_Profile(Aircraft_Index,Initial_Weight_kg)

% --------------
% DEFINE GLOBAL VARIABLES
% --------------
global BADA_Aircraft_Data rocTable CLIMB_TAS Altitude_Table CLIMB_FUEL_NOM
global FeetInNauticalMiles Time Assn_Cruise_FL Distance
global Current_Lat Current_Lon i Origin_Airport_Altitude Wind_Proj
global Waypoints Initial_Lat Initial_Lon BADA_Aircraft_Coef


global j true_air_speed rate_of_climb fuel_flow

ftpsecTOknots = 0.592483801;
% ------------------------------------------------------------------------

%load BADA_Aircraft_Data

% --------------
% GET AIRCRAFT DATA
% --------------
CLIMB_TAS      = BADA_Aircraft_Data(Aircraft_Index).CLIMB_TAS;
CLIMB_FUEL_NOM = BADA_Aircraft_Data(Aircraft_Index).CLIMB_FUEL_NOM;
Altitude_Table = BADA_Aircraft_Data(Aircraft_Index).FL * 100; % IN FEET
rocTable       = BADA_Aircraft_Data(Aircraft_Index).CLIMB_ROC_NOM; % RATE OF CLIMB
% ------------------------------------------------------------------------

% --------------
% CORRECT AIRCRAFT DATA FOR NON-ZERO AIRPORT ALTITUDE
% --------------
if Origin_Airport_Altitude > 0
   % USE HYDROSTATICS TO GET DENSITY RATIO
   h = Origin_Airport_Altitude*0.3048; % IN METERS ABOVE SEALEVEL
   L = -.0065; % LAPSE RATE IN TROPOSHERE< IN (KELVIN/METER)
   T_SL = 288.15; % SEA LEVEL TEMPERATURE IN KELVIN, ACCORDING TO BADA
   g0 = 9.81; % BASE GRAVITATIONAL CONSTANT, IN m/s^2
   R = 287.0368; % GAS CONSTANT, IN N.m/kg.K

   T_h = T_SL + L*h; % TEMPERATURE AT AIRPORT ALTITUDE, IN KELVIN
   sigma = (T_h/T_SL)^(-(g0/(L*R)+1)); % DENSITY RATIO, rho_h/rho_SL

   % CORRECT ALTITUDE TABLE TO START AT ABOUT AIRPORT ALTITUDE
```

```
    [~,Data_Fix_Index,~] = find(Altitude_Table>floor(Origin_Airport_Altitude),1);
    temp_TAS = CLIMB_TAS(Data_Fix_Index:end);
    temp_FUEL = CLIMB_FUEL_NOM(Data_Fix_Index:end);
    temp_ROC = rocTable(Data_Fix_Index:end);
    temp_ALT = Altitude_Table(Data_Fix_Index:end);
    CLIMB_TAS_new = temp_TAS;
    CLIMB_FUEL_NOM_new = temp_FUEL;
    rocTable_new = temp_ROC;
    Altitude_Table_new = temp_ALT;

    % REPLACE DATA WITH DENSITY RATIO-CORRECTED VALUES
    Length = length(CLIMB_TAS_new);
    CLIMB_TAS_new = CLIMB_TAS(1:Length)/sigma;
    CLIMB_FUEL_NOM_new = CLIMB_FUEL_NOM(1:Length)*sigma;
    rocTable_new = rocTable(1:Length)*sigma;

    % FOR ALTITUDES ABOVE 20000 FT, USE ORIGINAL BADA DATA
    [~,Index_Old_Data,~] = find(Altitude_Table == 20000);
    [~,Index_New_Data,~] = find(Altitude_Table_new == 20000);

    CLIMB_TAS_new(Index_New_Data:end) = CLIMB_TAS(Index_Old_Data:end);
    CLIMB_FUEL_NOM_new(Index_New_Data:end) = CLIMB_FUEL_NOM(Index_Old_Data:end);
    rocTable_new(Index_New_Data:end) = rocTable(Index_Old_Data:end);

    Altitude_Table = Altitude_Table_new;
    CLIMB_TAS = CLIMB_TAS_new;
    CLIMB_FUEL_NOM = CLIMB_FUEL_NOM_new;
    rocTable = rocTable_new;

end
% -------------------------------------------------------------------------

% --------------
% DEFINE BOUNDARY AND INITIAL CONDITIONS
% --------------
% INITIAL LATITUDE AND LONGITUDE
Initial_Lat = Waypoints(1,1);
Initial_Lon = Waypoints(1,2);

% DEFINE TIME SPAN (ONE HOUR IN LENGTH)
Time_Initial = 0.0;                    % INITIAL TIME (SECONDS)
Time_Final   = 3600.0;                 % FINAL TIME (SECONDS)
Time_Span    = [Time_Initial Time_Final];  % SPAN TIME

% DEFINE INITIAL MASS
Aircraft_Mass_Initial   = Initial_Weight_kg; % KG

% DEFINE INITIAL STATE VARIABLES
yClimb_Initial  = [Origin_Airport_Altitude Aircraft_Mass_Initial 0];
% y(1) - ALTITUDE (FEET)
% y(2) - WEIGHT (KG)
% y(3) - DISTANCE TRAVELLED (FEET)

% DEFINE INITIAL F_Climb VARIABLES
i = 0;
j = 0;
```

```
Current_Lat = 0;
Current_Lon = 0;
Distance = 0;
Time = 0;
Wind_Proj = 0;
% ------------------------------------------------------------------------

% --------------
% GENERATE CLIMB PROFILE
% --------------
[Climb_Time,Climb_Data] = rk4sys('F_Climb_Coef', Time_Span, yClimb_Initial, 60, []);

% FIND INDEX FOR REACHING CRUISE FLIGHT LEVEL
Altitude_ft = Climb_Data(:,1);
Index_To_Reach_MaxFL = find(Altitude_ft >= Assn_Cruise_FL, 1, 'first');
if isempty(Index_To_Reach_MaxFL) == 1
   Index_To_Reach_MaxFL = length(Altitude_ft)-1; % GET INDEX OF LAST ITEM IF NOTHING TO
TRUNCATE
end

true_air_speed = true_air_speed(1:Index_To_Reach_MaxFL);
rate_of_climb = rate_of_climb(1:Index_To_Reach_MaxFL);
fuel_flow = fuel_flow(1:Index_To_Reach_MaxFL);

% TRUNCATE CLIMB PROFILE TO STOP AT CRUISE FLIGHT LEVEL; CONVERT DATA
Altitude_ft           = Altitude_ft(1:Index_To_Reach_MaxFL);
Distance_nm           = Climb_Data(1:Index_To_Reach_MaxFL,3)/FeetInNauticalMiles;
Time_hrs              = Climb_Time(1:Index_To_Reach_MaxFL) / 3600;
Weight_kg             = Climb_Data(1:Index_To_Reach_MaxFL,2);

Rate_of_Climb_ftmin   = rate_of_climb';
Speed_knots           = true_air_speed';%(Distance_nm(2:end) - Distance_nm(1:end-1)) ./ (Time_hrs(2:end) -
Time_hrs(1:end-1));
FuelFlow_kgmin        = fuel_flow';%-(Weight_kg(2:end) - Weight_kg(1:end-1)) ./ (Time_hrs(2:end) -
Time_hrs(1:end-1)) / 60;
Distance_nm_For_Speed = (Distance_nm(2:end) + Distance_nm(1:end-1)) / 2;
Total_Fuel_kg         = Weight_kg(1) - Weight_kg(end);

Latitude_Pts          = Current_Lat(1:Index_To_Reach_MaxFL);
Longitude_Pts         = Current_Lon(1:Index_To_Reach_MaxFL);
Wind_Vectors          = Wind_Proj(1:Index_To_Reach_MaxFL);
% ------------------------------------------------------------------------

% --------------
% SAVE CLIMB PROFILE IN STRUCTURE
% --------------
Climb_Profile.Distance_For_Climb_nm = max(Distance_nm);
Climb_Profile.Time_For_Climb_hrs    = max(Time_hrs);
Climb_Profile.Total_Fuel_kg         = Total_Fuel_kg;

Climb_Profile.Altitude_ft           = Altitude_ft;
Climb_Profile.Distance_nm           = Distance_nm;
Climb_Profile.Time_hrs              = Time_hrs;
Climb_Profile.Weight_kg             = Weight_kg;

Climb_Profile.Rate_of_Climb_ftmin   = Rate_of_Climb_ftmin;
```

```
Climb_Profile.Speed_knots           = Speed_knots;
Climb_Profile.FuelFlow_kgmin        = FuelFlow_kgmin;
Climb_Profile.Distance_nm_For_Speed = Distance_nm_For_Speed;

Climb_Profile.Latitude_Pts          = Latitude_Pts;
Climb_Profile.Longitude_Pts         = Longitude_Pts;
Climb_Profile.Wind_Vectors_knots    = Wind_Vectors * ftpsecTOknots;
% ------------------------------------------------------------------------
return
```

B.5 F_Climb_Coef.m

```
% This function generates points for the climb profile of the
% aircraft/airport combination
%
% yprime = F_Climb(t,y,Add)
%
% INPUT:
%   t   = time
%   y   = rate equations
%       y(1) = Aircraft altitude (feet)
%       y(2) = Aircraft weight (kilograms)
%       y(3) = Distance traveled along the path (feet)
%   Add = boolean for RK4 to run F_Climb
%
% OUTPUT:
%   yprime = derivative of y (rate equations)

function yprime = F_Climb_Coef(t,y,Add)

% --------------
% DEFINE GLOBAL VARIABLES
% --------------
global rocTable Altitude_Table CLIMB_TAS CLIMB_FUEL_NOM
global Waypoints i  Distance FeetInNauticalMiles Assn_Cruise_FL
global Current_Lat Current_Lon Wind_Boolean Wind_Proj Month
global Initial_Lat Initial_Lon BADA_Aircraft_Coef Aircraft_Index

global j true_air_speed rate_of_climb fuel_flow azimuth_vec

pmTops = 1/60;        % CONVERT FROM FEET PER MINUTE TO FEET PER SECOND
knotsTofps = 1.6874;  % CONVERT KNOTS TO FEET PER SECOND
mpsTofps = 3.2808;    % CONVERT WIND SPEED FROM METERS PER SECOND TO FEET PER SECOND
mpsToknots = 1.94384449;    %convert wind speed from m/s to knots.
% ------------------------------------------------------------------------

% --------------
% EXTRACT CURRENT ALTITUDE
% --------------
Current_Altitude = min(y(1), Assn_Cruise_FL);
% ------------------------------------------------------------------------

% --------------
% CALCULATE WIND PROJECTION
% --------------
if i == 0 % CALCULATE INITIAL WIND PROJECTION
   i = i+1; % COUNTER
   Distance(i,:) = y(3);
   % INITIAL LAT/LON POINTS
   Current_Lat(i,:) = Initial_Lat; %Waypoints(1,1);
   Current_Lon(i,:) = Initial_Lon; %Waypoints(1,2);

   [~,Index] = min(abs(Current_Lat(i,:) - Waypoints(:,1)) + abs(Current_Lon(i,:) - Waypoints(:,2))); % Find index of
Waypoints corresponding to lat/lon
```

```
% --------------
% CORRECT INDEX
% --------------
% WAYPOINTS OF INDEX MUST BE FARTHER ALONG THAN CURRENT LAT/LON TO
% AVOID BACKTRACKING
if Index == length(Waypoints)
   Index = Index - 1;
elseif Index <= length(Waypoints) - 1 % FOR INDICES BEFORE END OF WAYPOINTS

   % DETERMINE IF WAYPOINTS IS INCREASING OR DECREASING IN SIZE

   if Waypoints(Index+1,1) > Waypoints(Index,1)
      if Current_Lat(i,:) >= Waypoints(Index,1)
         Index = Index + 1;
      end
   elseif Waypoints(Index+1,1) < Waypoints(Index,1)
      if Current_Lat(i,:) <= Waypoints(Index,1)
         Index = Index + 1;
      end
   end

   if Index ~= length(Waypoints)
      if Waypoints(Index+1,2) > Waypoints(Index,2)
         if Current_Lon(i,:) >= Waypoints(Index,2)
            Index = Index + 1;
         end
      elseif Waypoints(Index+1,2) < Waypoints(Index,2)
         if Current_Lon(i,:) <= Waypoints(Index,2)
            Index = Index + 1;
         end
      end
   end

else % FOR INDICES APPROACHING END OF WAYPOINTS
   if Waypoints(Index,1) > Waypoints(Index-1,1)
      if Current_Lat(i,:) >= Waypoints(Index,1)
         Index = Index + 1;
      end
   elseif Waypoints(Index,1) < Waypoints(Index-1,1)
      if Current_Lat(i,:) <= Waypoints(Index,1)
         Index = Index + 1;
      end
   end

   if Index == length(Waypoints)
      if Waypoints(Index,2) > Waypoints(Index-1,2)
         if Current_Lon(i,:) >= Waypoints(Index,2)
            Index = Index + 1;
         end
      elseif Waypoints(Index,2) < Waypoints(Index-1,2)
         if Current_Lon(i,:) <= Waypoints(Index,2)
            Index = Index + 1;
         end
      end
   end
```

```
    end

    % CORRECT INDEX IF GREATER THAN WAYPOINTS LENGTH
    if Index > length(Waypoints(:,1))
        Index = length(Waypoints(:,1));
    end
    % ---------------------------------------------------------------------

    % --------------
    % CALCULATE HEADING
    % --------------
    Azimuth = azimuth(Current_Lat(i,:), Current_Lon(i,:), Waypoints(Index,1), Waypoints(Index,2)); %
INSTANTANEOUS ANGLE OF HEADING

    % INITIAL WIND PROJECTION
    if Wind_Boolean == true
        Wind_Proj(i,:) = Wind_Calculator(Azimuth, Month, y(1), Current_Lat(i,:), Current_Lon(i,:))*mpsToknots;
    end

elseif Add == true % RUN THIS CODE ONLY WHEN F_CLIMB IS CALLED THE FOURTH TIME BY THE
rk4sys ODE FUNCTION
    i = i+1; % COUNTER
    Distance(i,:) = y(3);

    % FIND INDEX OF WAYPOINTS CORRESPONDING TO CURRENT LAT/LON POINTS
    Change_in_Distance = (Distance(i,:) - Distance(i-1,:))/FeetInNauticalMiles;
    [~,Index] = min(abs(Current_Lat(i-1,:) - Waypoints(:,1)) + abs(Current_Lon(i-1,:) - Waypoints(:,2))); % Find
index of Waypoints corresponding to lat/lon

    % --------------
    % CORRECT INDEX
    % --------------
    % WAYPOINTS OF INDEX MUST BE FARTHER ALONG THAN CURRENT LAT/LON TO
    % AVOID BACKTRACKING
    if Index == length(Waypoints)
        Index = Index - 1;
    elseif Index <= length(Waypoints) - 1 % FOR INDICES BEFORE END OF WAYPOINTS

        % DETERMINE IF WAYPOINTS IS INCREASING OR DECREASING IN SIZE

        if Waypoints(Index+1,1) > Waypoints(Index,1)
            if Current_Lat(i-1,:) >= Waypoints(Index,1)
                Index = Index + 1;
            end
        elseif Waypoints(Index+1,1) < Waypoints(Index,1)
            if Current_Lat(i-1,:) <= Waypoints(Index,1)
                Index = Index + 1;
            end
        end

        if Index ~= length(Waypoints)
            if Waypoints(Index+1,2) > Waypoints(Index,2)
                if Current_Lon(i-1,:) >= Waypoints(Index,2)
                    Index = Index + 1;
                end
            elseif Waypoints(Index+1,2) < Waypoints(Index,2)
```

```
            if Current_Lon(i-1,:) <= Waypoints(Index,2)
                Index = Index + 1;
            end
        end
    end

    else % FOR INDICES APPROACHING END OF WAYPOINTS
        if Waypoints(Index,1) > Waypoints(Index-1,1)
            if Current_Lat(i-1,:) >= Waypoints(Index,1)
                Index = Index + 1;
            end
        elseif Waypoints(Index,1) < Waypoints(Index-1,1)
            if Current_Lat(i-1,:) <= Waypoints(Index,1)
                Index = Index + 1;
            end
        end

        if Index == length(Waypoints)
            if Waypoints(Index,2) > Waypoints(Index-1,2)
                if Current_Lon(i-1,:) >= Waypoints(Index,2)
                    Index = Index + 1;
                end
            elseif Waypoints(Index,2) < Waypoints(Index-1,2)
                if Current_Lon(i-1,:) <= Waypoints(Index,2)
                    Index = Index + 1;
                end
            end
        end

    end

    while (1)
        [~,Dist_to_Next_Waypt] = legs([Current_Lat(i-1,:) Waypoints(Index,1)], [Current_Lon(i-1,:)
Waypoints(Index,2)]);
        if Dist_to_Next_Waypt < Change_in_Distance
            Index = Index + 1;
            if Index >= length(Waypoints)
                Index = length(Waypoints)-1;
                break
            end
        else
            break
        end
    end

    % CORRECT INDEX IF GREATER THAN WAYPOINTS LENGTH
    if Index > length(Waypoints(:,1))
        Index = length(Waypoints(:,1));
    end
    % ------------------------------------------------------------------------

    % --------------
    % CALCULATE HEADING
    % --------------
    Azimuth = azimuth(Current_Lat(i-1,:), Current_Lon(i-1,:), Waypoints(Index,1), Waypoints(Index,2)); %
INSTANTANEOUS ANGLE OF HEADING
```

```
    % ------------------------------------------------------------------

    % --------------
    % CALCULATE LAT/LON
    % --------------
    Change_in_Degrees = nm2deg(Change_in_Distance); % CONVERT DISTANCE NM TO DEGREES FOR USE
IN RECKON FUNCTION
    [Current_Lat(i,:),Current_Lon(i,:)] = reckon(Current_Lat(i-1,:),Current_Lon(i-1),Change_in_Degrees,Azimuth);
    % ------------------------------------------------------------------

    % --------------
    % CALCULATE WIND PROJECTION
    % --------------
    if Wind_Boolean == true
        Wind_Proj(i,:) = Wind_Calculator(Azimuth, Month, y(1), Current_Lat(i,:), Current_Lon(i,:))*mpsToknots;
    end
    % ------------------------------------------------------------------
end

% WIND VS NO WIND
if Wind_Boolean == false
    Wind_Proj(i,:) = 0;
end
% ------------------------------------------------------------------

% % --------------
% % GET ODE VARIABLES
% % --------------
% if Current_Altitude < min(Altitude_Table)
%     vtas       = custom_interp(fliplr(Altitude_Table),fliplr(CLIMB_TAS),Current_Altitude) * knotsTofps +
Wind_Proj(i,:)*knotsTofps;   % SPEED IS TRUE AIRSPEED (TAS) IN FPS
%     rateOfClimb = custom_interp(fliplr(Altitude_Table),fliplr(rocTable),Current_Altitude) * pmTops;
% IN FEET PER SECOND UNITS
%     fuelFlow    = custom_interp(fliplr(Altitude_Table),fliplr(CLIMB_FUEL_NOM),Current_Altitude)  * pmTops;
% KILOGRAMS PER SECOND
% else
%     vtas       = custom_interp(Altitude_Table,CLIMB_TAS,Current_Altitude) * knotsTofps +
Wind_Proj(i,:)*knotsTofps;                % SPEED IS TRUE AIRSPEED (TAS) IN FPS
%     rateOfClimb = custom_interp(Altitude_Table,rocTable,Current_Altitude) * pmTops;
% IN FEET PER SECOND UNITS
%     fuelFlow    = custom_interp(Altitude_Table,CLIMB_FUEL_NOM,Current_Altitude)  * pmTops;
% KILOGRAMS PER SECOND
% end
% % ------------------------------------------------------------------

% --------------
% GET ODE VARIABLES
% --------------
vtas_kts    = Speed_Calculator_Climb(Aircraft_Index, Current_Altitude, y(2), BADA_Aircraft_Coef);
vtas       = vtas_kts * knotsTofps + Wind_Proj(i,:)*knotsTofps;   % SPEED IS TRUE AIRSPEED (TAS) IN FPS
rateOfClimb = ROC_Calculator(Aircraft_Index, Current_Altitude, vtas_kts, y(2), BADA_Aircraft_Coef) * pmTops;
% IN FEET PER SECOND UNITS
fuelFlow    = FuelFlow_Calculator_Climb(Aircraft_Index, Current_Altitude, vtas_kts, BADA_Aircraft_Coef)  *
pmTops;              % KILOGRAMS PER SECOND

% ------------------------------------------------------------------
```

```
if Add == true
   j = j+1;
   true_air_speed(j) = vtas/knotsTofps;
   rate_of_climb(j) = rateOfClimb/pmTops;
   fuel_flow(j) = fuelFlow/pmTops;
   azimuth_vec(j) = Azimuth;
end

% --------------
% ASSIGN ODE VARIABLES
% --------------
% CHECK IF CRUISE ALTITUDE IS REACHED
if y(1) < Assn_Cruise_FL
   yprime(1) = rateOfClimb;
else
   yprime(1) = 0; % ZERO RATE OF CLIMB (FEET/S)
end

% MASS FLOW RATE (KG/S)
yprime(2) = - fuelFlow;

% DISTANCE TRAVELLED ALONG FLIGHT PATH (FEET)
yprime(3) = vtas;

% TRANSPOSE THE ARRAY GOING OUT
yprime = yprime';
% ------------------------------------------------------------------------
return
```

B.6 Speed_Calculator_Climb.m

```
% Speed_Calculator
% IMPORTANT: currently configured only for CRUISE conditions
%
% [VTAS] = Speed_Calculator(Aircraft_Index, Altitude)
%
% INPUT:
%   Aircraft_Index      = index of current aircraft in BADA
%   Altitude            = current altitude of aircraft <ft>
%   AC_Mass             = current mass of aircraft <kg>
%   BADA_Aircraft_Coef  = structure of BADA coefficients from APF file for
%                         aircraft
%
% OUTPUT:
%   VTAS = current true airspeed <kts>

function [VTAS] = Speed_Calculator_Climb(Aircraft_Index, Altitude, AC_Mass, BADA_Aircraft_Coef)
% global BADA_Aircraft_Coef

% Convert
kts2m_s = 0.514444444444; % kts to m/s
ft2m = 0.3048; % ft to m

% Deviations from ISA for now assumed to be zero:
delta_pres = 0;
delta_temp = 0;

% --------------
% MSL STANDARD CONDITIONS
% --------------
temp_0 = 288.15; % standard temperature at MSL <K>
pres_0 = 101325; % standard pressure at MSL <Pa>
dens_0 = 1.225; % standard density at MSL <kg/m^3>
a_0 = 340.294; % speed of sound at MSL <m/s>
% ------------------------------------------------------------------------

% --------------
% PHYSICAL CONSTANTS
% --------------
gamma = 1.4; % adiabatic index of air (kappa in BADA User Manual)
R = 287.05287; % real gas constant for air <m^2/K.s^2>
g0 = 9.80665; % gravitational acceleration <m/s^2>
beta = -0.0065; % ISA temerature gradient below tropo (lapse rate L) <K/m>
r_earth = 6.371e+6; % RADIUS OF EARTH
% ------------------------------------------------------------------------

% --------------
% ATMOSPHERIC CONDITIONS
% --------------
Altitude_m = ft2m * Altitude;
Hp = (r_earth / (r_earth + Altitude_m)) * Altitude_m; % geopotential altitude <m>
Hp_tropo = 11000; % geopotential alt at tropopause <m>

Hp = Altitude_m;
```

```
% MEAN SEA LEVEL (MSL)

if Hp < Hp_tropo
   temp = temp_0 + delta_temp + beta*Hp;
   pres = pres_0*((temp - delta_temp)/temp_0)^(-g0/(beta*R));
elseif Hp == Hp_tropo
   temp = temp_0 + delta_temp + beta*Hp_tropo;
   pres = pres_0*((temp - delta_temp)/temp_0)^(-g0/(beta*R));
elseif Hp > Hp_tropo
   temp_tropo = temp_0 + delta_temp + beta*Hp_tropo;
   pres_tropo = pres_0*((temp_tropo - delta_temp)/temp_0)^(-g0/(beta*R));
   temp_ISA_tropo = temp_0 + beta*Hp_tropo;
   temp = temp_tropo;
   pres = pres_tropo * exp(-g0/(R*temp_ISA_tropo)*(Hp-Hp_tropo));
end
dens = pres/(R*temp);
% ------------------------------------------------------------------------

% --------------
% CONSTANTS ACROSS ALL AIRCRAFT
% --------------
% MINIMUM SPEED COEFFICIENT
Cv_min = 1.3;

% CLIMB SPEED INCREMENTS <KCAS>
Vd_cl1 = 5;
Vd_cl2 = 10;
Vd_cl3 = 30;
Vd_cl4 = 60;
Vd_cl5 = 80;
Vd_cl6 = 20;
Vd_cl7 = 30;
Vd_cl8 = 35;
% ------------------------------------------------------------------------

% --------------
% PROCEDURES SPECIFICATION CAS SCHEDULE
% --------------
eng_type = BADA_Aircraft_Coef(Aircraft_Index).AC_Type.eng_type;
V_cl1 = BADA_Aircraft_Coef(Aircraft_Index).Procedures.V_cl1;
V_cl2 = BADA_Aircraft_Coef(Aircraft_Index).Procedures.V_cl2;
M_cl = BADA_Aircraft_Coef(Aircraft_Index).Procedures.M_cl;
m_ref = BADA_Aircraft_Coef(Aircraft_Index).Mass.m_ref * 1000; % tonnes to kg
V_stall_TO_ref = BADA_Aircraft_Coef(Aircraft_Index).Aero.V_stall_TO;

% ACTUAL V_STALL_TO FOR AC_Mass
V_stall_TO = V_stall_TO_ref * sqrt(AC_Mass/m_ref);

% CALCULATE MACH TRANSITION ALTITUDE
V_cl2_m_s = kts2m_s * V_cl2;
delta_trans = ((1+(gamma-1)/2*(V_cl2_m_s/a_0)^2)^(gamma/(gamma-1))-1)...
   /((1+(gamma-1)/2*M_cl^2)^(gamma/(gamma-1))-1);
theta_trans = (delta_trans)^(-beta*R/g0);
Hp_trans = (-1/(ft2m*beta))*(temp_0*(1-theta_trans));
```

```
mu = (gamma-1)/gamma;

Altitude_trans = Hp_trans*r_earth/(r_earth-Hp_trans);

% CAS SCHEDULE
if strcmp(eng_type,'Jet') == 1
   if Altitude < 1500
      VCAS = Cv_min*V_stall_TO + Vd_cl1;
   elseif Altitude >= 1500 && Altitude < 3000
      VCAS = Cv_min*V_stall_TO + Vd_cl2;
   elseif Altitude >= 3000 && Altitude < 4000
      VCAS = Cv_min*V_stall_TO + Vd_cl3;
   elseif Altitude >= 4000 && Altitude < 5000
      VCAS = Cv_min*V_stall_TO + Vd_cl4;
   elseif Altitude >= 5000 && Altitude < 6000
      VCAS = Cv_min*V_stall_TO + Vd_cl5;
   elseif Altitude >= 6000 && Altitude < 10000
      VCAS = min(V_cl1,250);
   elseif Altitude >= 10000 && Altitude < Altitude_trans
      VCAS = V_cl2;
   elseif Altitude >= Altitude_trans
      M = M_cl;
   end
else
   if Altitude < 500
      VCAS = Cv_min*V_stall_TO + Vd_cl6;
   elseif Altitude >= 500 && Altitude < 1000
      VCAS = Cv_min*V_stall_TO + Vd_cl7;
   elseif Altitude >= 1000 && Altitude < 1500
      VCAS = Cv_min*V_stall_TO + Vd_cl8;
   elseif Altitude >= 1500 && Altitude < 10000
      VCAS = min(V_cl1,250);
   elseif Altitude >= 10000 && Altitude < Altitude_trans
      VCAS = V_cl2;
   elseif Altitude >= Altitude_trans
      M = M_cl;
   end
end
% ---------------------------------------------------------------------

% --------------
% CONVERT CAS OR MACH TO TAS
% --------------
if Altitude < Altitude_trans
   VCAS_m_s = kts2m_s * VCAS;
   VTAS = 1/kts2m_s * (2/mu*pres/dens*((1+pres_0/pres*((1+mu/2*dens_0/pres_0*VCAS_m_s^2)^(1/mu)-
1))^mu-1))^(1/2);
else
   VTAS = 1/kts2m_s*M*sqrt(gamma*R*temp);
end
% ---------------------------------------------------------------------
```

B.7 ROC_Calculator.m

```
% ROC Calculator
%
% [ROC] = ROC_Calculator(Aircraft_Index, Altitude, VTAS, AC_Mass, BADA_Aircraft_Coef)
%
% INPUT:
%   Aircraft_Index      = index of current aircraft in BADA
%   Altitude            = current altitude of aircraft <ft>
%   VTAS                = current true airspeed <kts>
%   AC_Mass             = current mass of aircraft <kg>
%   BADA_Aircraft_Coef  = structure of BADA coefficients from APF file for
%                         aircraft
%
% OUTPUT:
%   ROC                 = current rate of climb <ft/min>

function [ROC] = ROC_Calculator(Aircraft_Index, Altitude, VTAS, AC_Mass, BADA_Aircraft_Coef)

% DEVIATIONS FROM ISA FOR NOW ASSUMED TO BE ZERO:
delta_temp = 0;
ESF = 1; % energy share factor

% CONVERT
kts2m_s = 0.514444444444; % kts to m/s
ft2m = 0.3048; % ft to m

% --------------
% MSL STANDARD CONDITIONS
% --------------
temp_0 = 288.15; % standard temperature at MSL <K>
pres_0 = 101325; % standard pressure at MSL <Pa>
dens_0 = 1.225; % standard density at MSL <kg/m^3>
a_0 = 340.294; % speed of sound at MSL <m/s>
% ------------------------------------------------------------------------

% --------------
% PHYSICAL CONSTANTS
% --------------
gamma = 1.4; % adiabatic index of air (kappa in BADA User Manual
R = 287.05287; % real gas constant for air <m^2/K.s^2>
g0 = 9.80665; % gravitational acceleration <m/s^2>
beta = -0.0065; % ISA temerature gradient below tropo (lapse rate L) <K/m>
r_earth = 6.371e+6; % RADIUS OF EARTH
C_Tcr = 0.95; % BADA-defined maximum cruise thrust coefficient for all aircraft
% ------------------------------------------------------------------------

% --------------
% ATMOSPHERIC CONDITIONS
% --------------
Altitude_m = ft2m * Altitude;
% Hp = (r_earth / (r_earth + Altitude_m)) * Altitude_m; % geopotential altitude <m>
Hp_tropo = 11000; % geopotential alt at tropopause <m>

Hp = Altitude_m;
```

```
% MEAN SEA LEVEL (MSL)

if Hp < Hp_tropo
   temp = temp_0 + delta_temp + beta*Hp;
   pres = pres_0*((temp - delta_temp)/temp_0)^(-g0/(beta*R));
elseif Hp == Hp_tropo
   temp = temp_0 + delta_temp + beta*Hp_tropo;
   pres = pres_0*((temp - delta_temp)/temp_0)^(-g0/(beta*R));
elseif Hp > Hp_tropo
   temp_tropo = temp_0 + delta_temp + beta*Hp_tropo;
   pres_tropo = pres_0*((temp_tropo - delta_temp)/temp_0)^(-g0/(beta*R));
   temp_ISA_tropo = temp_0 + beta*Hp_tropo;
   temp = temp_tropo;
   pres = pres_tropo * exp(-g0/(R*temp_ISA_tropo)*(Hp-Hp_tropo));
end
dens = pres/(R*temp);
% ---------------------------------------------------------------------

% ----------------
% DRAG
% ----------------
eng_type = BADA_Aircraft_Coef(Aircraft_Index).AC_Type.eng_type;

S = BADA_Aircraft_Coef(Aircraft_Index).Aero.S; % planform area <m^2>
VTAS_m_s = kts2m_s * VTAS;

C_L = 2*AC_Mass*g0/(dens*VTAS_m_s^2*S);
C_D0_CR = BADA_Aircraft_Coef(Aircraft_Index).Aero.C_D0_CR;
C_D2_CR = BADA_Aircraft_Coef(Aircraft_Index).Aero.C_D2_CR;
C_D = C_D0_CR + C_D2_CR*C_L^2;

Drag = 1/2*C_D*dens*VTAS_m_s^2*S;
% ---------------------------------------------------------------------

% ----------------
% THRUST
% ----------------
% MAX CLIMB THRUST
C_Tc1 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc1;
C_Tc2 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc2;
C_Tc3 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc3;
C_Tc4 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc4;
C_Tc5 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc5;

Hp_ft = Altitude;%(r_earth / (r_earth + Altitude)) * Altitude;

if strcmp(eng_type,'Jet') == 1
   Thrust_max_cl_ISA = C_Tc1*(1 - Hp_ft/C_Tc2 + C_Tc3*Hp_ft^2);
elseif strcmp(eng_type,'Turboprop') == 1
   Thrust_max_cl_ISA = C_Tc1/VTAS*(1 - Hp_ft/C_Tc2) + C_Tc3;
elseif strcmp(eng_type,'Piston') == 1
   Thrust_max_cl_ISA = C_Tc1*(1 - Hp_ft/C_Tc2) + C_Tc3/VTAS;
end

% IF NOT ISA:
```

```
if delta_temp ~= 0
   delta_temp_eff = delta_temp - C_Tc4;
   Thrust_max_cl = Thrust_max_cl_ISA * (1 - C_Tc5*delta_temp_eff);
else
   Thrust_max_cl = Thrust_max_cl_ISA;
end

% % MAX POSSIBLE CRUISE THRUST
% Thrust_max_cr = C_Tcr * Thrust_max_cl;

Thrust = Thrust_max_cl;
% ------------------------------------------------------------------------

% --------------
% MAX ROC
% --------------
dh_dt = (Thrust-Drag)*VTAS_m_s/(AC_Mass*g0) * ESF;
% ------------------------------------------------------------------------

% --------------
% REDUCED CLIMB POWER
% --------------
h_MO = BADA_Aircraft_Coef(Aircraft_Index).Flight_Env.h_MO;
h_max = BADA_Aircraft_Coef(Aircraft_Index).Flight_Env.h_max;
G_t = BADA_Aircraft_Coef(Aircraft_Index).Flight_Env.G_t;
C_Tc4 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc4;
G_w = BADA_Aircraft_Coef(Aircraft_Index).Mass.G_w;
m_min = BADA_Aircraft_Coef(Aircraft_Index).Mass.m_min*1000;
m_max = BADA_Aircraft_Coef(Aircraft_Index).Mass.m_max*1000; % from tonnes to kg

Max_Alt_Actual_m = min(h_MO,h_max+G_t*(C_Tc4)+G_w*(m_max-AC_Mass))*ft2m;

if Hp < (0.8*Max_Alt_Actual_m)

   if strcmp(eng_type,'Jet') == 1
      C_red = 0.15;
   elseif strcmp(eng_type,'Turboprop') == 1
      C_red = 0.25;
   elseif strcmp(eng_type,'Piston') == 1
      C_red = 0;
   end

   C_pow_red = 1 - C_red * (m_max-AC_Mass)/(m_max-m_min);
   dh_dt = dh_dt*C_pow_red;
end
% ------------------------------------------------------------------------

ROC = dh_dt*60/ft2m; % convert from <m/s> to <ft/min>
```

B.8 FuelFlow_Calculator_Climb.m

```
% Fuel Flow Calculator
% IMPORTANT: currently configured only for CRUISE conditions
%
% [VTAS] = Speed_Calculator(Aircraft_Index, Altitude)
%
% INPUT:
%   Aircraft_Index      = index of current aircraft in BADA
%   Altitude            = current altitude of aircraft <ft>
%   VTAS                = current true airspeed <kts>
%   AC_Mass             = current mass of aircraft <kg>
%   BADA_Aircraft_Coef  = structure of BADA coefficients from APF file for
%                         aircraft
%
% OUTPUT:
%   Fuel_Flow           = current fuel flow <kg/min>

function [Fuel_Flow] = FuelFlow_Calculator_Climb(Aircraft_Index, Altitude, VTAS, BADA_Aircraft_Coef)

% DEVIATIONS FROM ISA FOR NOW ASSUMED TO BE ZERO:
delta_temp = 0;

% CONVERT
kts2m_s = 0.514444444444; % kts to m/s
ft2m = 0.3048; % ft to m
% --------------
% MSL STANDARD CONDITIONS
% --------------
temp_0 = 288.15; % standard temperature at MSL <K>
pres_0 = 101325; % standard pressure at MSL <Pa>
dens_0 = 1.225; % standard density at MSL <kg/m^3>
a_0 = 340.294; % speed of sound at MSL <m/s>
% ------------------------------------------------------------------------

% --------------
% PHYSICAL CONSTANTS
% --------------
gamma = 1.4; % adiabatic index of air (kappa in BADA User Manual
R = 287.05287; % real gas constant for air <m^2/K.s^2>
g0 = 9.80665; % gravitational acceleration <m/s^2>
beta = -0.0065; % ISA temerature gradient below tropo (lapse rate L) <K/m>
r_earth = 6.371e+6; % RADIUS OF EARTH
C_Tcr = 0.95; % BADA-defined maximum cruise thrust coefficient for all aircraft
% ------------------------------------------------------------------------

% % --------------
% % ATMOSPHERIC CONDITIONS
% % --------------
% Altitude_m = ft2m * Altitude;
% % Hp = (r_earth / (r_earth + Altitude_m)) * Altitude_m; % geopotential altitude <m>
% Hp_tropo = 11000; % geopotential alt at tropopause <m>
%
% Hp = Altitude_m;
%
```

```
% % MEAN SEA LEVEL (MSL)
%
% if Hp < Hp_tropo
%     temp = temp_0 + delta_temp + beta*Hp;
%     pres = pres_0*((temp - delta_temp)/temp_0)^(-g0/(beta*R));
% elseif Hp == Hp_tropo
%     temp = temp_0 + delta_temp + beta*Hp_tropo;
%     pres = pres_0*((temp - delta_temp)/temp_0)^(-g0/(beta*R));
% elseif Hp > Hp_tropo
%     temp_tropo = temp_0 + delta_temp + beta*Hp_tropo;
%     pres_tropo = pres_0*((temp_tropo - delta_temp)/temp_0)^(-g0/(beta*R));
%     temp_ISA_tropo = temp_0 + beta*Hp_tropo;
%     temp = temp_tropo;
%     pres = pres_tropo * exp(-g0/(R*temp_ISA_tropo)*(Hp-Hp_tropo));
% end
% dens = pres/(R*temp);
% % ------------------------------------------------------------------------

% ----------------
% THRUST
% ----------------
eng_type = BADA_Aircraft_Coef(Aircraft_Index).AC_Type.eng_type;

% MAX CLIMB THRUST - PTF tables don't consider this??
C_Tc1 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc1;
C_Tc2 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc2;
C_Tc3 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc3;
C_Tc4 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc4;
C_Tc5 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc5;

Hp_ft = Altitude;%(r_earth / (r_earth + Altitude)) * Altitude;

if strcmp(eng_type,'Jet') == 1
   Thrust_max_cl_ISA = C_Tc1*(1 - Hp_ft/C_Tc2 + C_Tc3*Hp_ft^2);
elseif strcmp(eng_type,'Turboprop') == 1
   Thrust_max_cl_ISA = C_Tc1/VTAS*(1 - Hp_ft/C_Tc2) + C_Tc3;
elseif strcmp(eng_type,'Piston') == 1
   Thrust_max_cl_ISA = C_Tc1*(1 - Hp_ft/C_Tc2) + C_Tc3/VTAS;
end

% IF NOT ISA:
if delta_temp ~= 0
   delta_temp_eff = delta_temp - C_Tc4;
   Thrust_max_cl = Thrust_max_cl_ISA * (1 - C_Tc5*delta_temp_eff);
else
   Thrust_max_cl = Thrust_max_cl_ISA;
end

% % MAX POSSIBLE CRUISE THRUST
% Thrust_max_cr = C_Tcr * Thrust_max_cl;

Thrust = Thrust_max_cl;
% ------------------------------------------------------------------------

% ----------------
% FUEL FLOW
```

```
% ----------------
C_f1 = BADA_Aircraft_Coef(Aircraft_Index).Fuel.C_f1;
C_f2 = BADA_Aircraft_Coef(Aircraft_Index).Fuel.C_f2;
C_fcr = BADA_Aircraft_Coef(Aircraft_Index).Fuel.C_fcr;

% THRUST SPECIFIC FUEL CONSUMPTIONS
if strcmp(eng_type,'Jet') == 1
   TSFC = C_f1*(1+VTAS/C_f2);
elseif strcmp(eng_type,'Turboprop') == 1
   TSFC = C_f1*(1-VTAS/C_f2)*(VTAS/1000);
end

% FUEL FLOW
Thrust_kN = Thrust/1000; % convert thrust from <N> to <kN>

if strcmp(eng_type,'Piston') == 1
   f_cr = C_f1;
else
   f_cr = TSFC * Thrust_kN;
end
% ------------------------------------------------------------------------

Fuel_Flow = f_cr;

end
```

B.9 Wind_Calculator.m

```
% This function calculates the magnitude of the wind vector projected onto
% the direction vector, as well as the sign of the wind relative to the
% direction
%
% wind_proj = Wind_Calculator(Course, Month, Altitude, Latitude, Longitude)
%
% INPUT:
%   Course      = general direction of aircraft, in degrees from north
%   %Month       = month of flight, to reference in wind data
%   Altitude    = current altitude of aircraft in flight
%   Latitude    = current latitude coordinate of aircraft in flight
%   Longitude   = current longitude coordinate of aircraft in flight
%
% OUTPUT:
%   wind_proj   = magnitude and sign of wind vector to be added (or
%                 subtracted) from direction vector, unit in meters per
%                 second

% Coded by Maria Rye, Modified by Nicolas Hinze

function wind_proj = Wind_Calculator(Course, Altitude, Latitude, Longitude)

% --------------
% DEFINE GLOBAL VARIABLES
% --------------
global Wind Wind_Parameters %STRUCTURES
% ------------------------------------------------------------------------

% --------------
% CORRECT DIRECTION
% --------------
% COURSE ANGLES CANNOT BE USED TO COMPARE DIRECTION COMPONENTS WITH WIND
% COMPONENTS
% SOUTH IS (+) (DIRECTION = 90 DEGREES)
% EAST IS (+) (DIRECTION = 0 DEGREES)

if Course >= 0 && Course <= 180
   Direction = Course - (180 - 2*(180 - Course));
elseif Course > 180 && Course < 360
   Direction = Course + (180 + 2*(180 - Course));
end
% ------------------------------------------------------------------------

% --------------
% FIND WIND DATA INDEX OF LAT/LON
% --------------
% CORRECT LONGITUDE SIGN
Longitude = Longitude + 360;

[~, Lat_Index] = min(abs(Latitude - Wind_Parameters.Lat));
[~, Lon_Index] = min(abs(Longitude - Wind_Parameters.Lon));
% ------------------------------------------------------------------------
```

```
% --------------
% DEFINE CONSTANTS AND CONVERSION FACTORS
% --------------
m_to_ft = 3.2808399;
R = 287; % Nm/kgK
g = 9.81; % m/s^2
Pa_to_mbar = .01; % WIND DATA IS IN MILLIBARS, CONVERT TO PASCALS
r_earth = 6.357e+6; % RADIUS OF EARTH

Altitude = Altitude/m_to_ft; % CONVERT TO METERS FOR EASIER USE
Geopotential_Alt = (r_earth / (r_earth + Altitude)) * Altitude;
      % CONVERT ALTITUDE TO GEOPOTENTIAL HEIGHT FOR USE IN ATMOSPHERIC
      % HYDROSTATIC EQUATIONS
% ------------------------------------------------------------------------

% --------------
% CALCULATE PRESSURE
% --------------
if Altitude < 11019 %m

   % DEFINE SEA LEVEL CONSTANTS
   Pressure_0 = 1.0131e+5; % Pa
   Temp_0 = 288.16; % K
   L = -.0065; % K/m

   Pressure = Pressure_0 * ((Temp_0 + L*Geopotential_Alt) / Temp_0) ^ (-g/(L*R)); % Pa

   Pressure = Pressure * Pa_to_mbar; % CONVERT FROM Pa TO MILLIBARS

else

   % DEFINE TROPOSPHERIC END VALUES (TROPOPAUSE INITIAL CONDITIONS)
   Temp_0 = 216.5; % K
   Alt_0 = 11000; % m
   Pressure_0 = 22632; % Pa

   Pressure = Pressure_0 * exp(-g * (Geopotential_Alt - Alt_0) / (R*Temp_0)); % Pa

   Pressure = Pressure * Pa_to_mbar; % CONVERT FROM Pa TO MILLIBARS

end
% ------------------------------------------------------------------------

% --------------
% FIND INDEX AND U/V-WIND
% --------------
[~,Level_Index] = min(abs(Pressure - Wind_Parameters.Level));

Index_Level = find(Wind.Level == Level_Index);
Index_Lat = find(Wind.Lat(Index_Level) == Lat_Index);
Index_Lon = Wind.Lon(Index_Level(Index_Lat)) == Lon_Index;
% Index_Time = find(Wind.Time(Index_Level(Index_Lat(Index_Lon))) == Month);
% Index = Index_Level(Index_Lat(Index_Lon(Index_Time)));
Index = Index_Level(Index_Lat(Index_Lon));
Uwind = Wind.Uwind(Index);
Vwind = Wind.Vwind(Index);
```

```
% ------------------------------------------------------------------------

% --------------
% CALCULATE WIND PROJECTION
% --------------
% FIND PROJECTION OF WIND ONTO DIRECTION UNIT VECTOR
Direction_Vector = [cosd(Direction), sind(Direction)];
Wind_Vector = [Vwind, Uwind];
   % BE AWARE OF ORDER OF COMPONENTS: Vwind WILL BE THE X-COMPONENT, Uwind
   % WILL BE THE Y-COMPONENT.  THIS IS COMPATIBLE WITH THE Direction ANGLE
   % CALCULATED ABOVE

% PROJECTION ONTO Direction_vector
wind_proj = dot(Direction_Vector, Wind_Vector) / (norm(Direction_Vector));
% ------------------------------------------------------------------------

return;
```

B.10 rk4sys.m

```
% Solution of 1st order ODE using Runge Kutta 4th order with adaptive step
% size
%
%   dy/dt = dtdt(tp,yp)
%
% [tp,yp] = rk4sys(dydt,tspan,y0,h,varargin)
%
% INPUT:
%   dydt    = name of external function to evaluate the solution of the ODE
%   tspan   = limits of integration
%   y0      = initial condition
%   h       = initial stepsize
%   varargin= other arguments
%
% OUTPUT:
%   [tp,yp] = solution vectors

function [tp,yp] = rk4sys(dydt,tspan,y0,h,varargin)

n = length(tspan);
ti = tspan(1); tf = tspan(n);

if n==2,
   t=(ti:h:tf)'; n = length(t);

   if t(n) < tf
      t(n+1) = tf;
      n = n+1;
   end

else
   t = tspan;
end

tt = ti; y(1,:) = y0;
np = 1; tp(np) = tt; yp(np,:) = y(1,:);
i = 1;

while(1)
   tend = t(np+1);
   hh = t(np+1)-t(np);
   if hh>h, hh=h; end

   while(1) % Run RK4 equations
      if tt+hh>tend, hh=tend-tt;end

      k1=feval(dydt,tt,y(i,:),false)';
      ymid = y(i,:)+k1.*hh/2;
      k2 = feval(dydt,(tt+hh/2),ymid,false)';
      ymid = y(i,:)+k2.*hh/2;
      k3 = feval(dydt,(tt+hh/2),ymid,false)';
      yend = y(i,:)+k3.*hh;
      k4 = feval(dydt,(tt+hh),yend,true)';
```

```
      phi = (k1+2*(k2+k3)+k4)/6;
      y(i+1,:) = y(i,:)+phi*hh;
      tt = tt+hh;
      i=i+1;

      if tt>=tend,break,end % Break while loop when independent variable is greater than max indep. var.
   end

   np = np+1; tp(np,:)=tt; yp(np,:) = y(i,:);
   if tt>=tf,break,end

end
```

B.11 Generate_Descent_Profile.m

```
% This function generates the descent profile of the aircraft/airport
% combination being called
%
% Descent_Profile = Generate_Descent_Profile(Aircraft_Index, Climb_Profile)
%
% INPUT:
%   Aircraft_Index = index of AC/AP combo from T100 tracks structure
%   Climb_Profile  = inputs climb information for reference in descent
%
% OUTPUT:
%   Climb_Profile  =  motion profile of aircraft's climb

function Descent_Profile = Generate_Descent_Profile(Aircraft_Index, Climb_Profile)

% --------------
% DEFINE GLOBAL VARIABLES
% --------------
global BADA_Aircraft_Data rodTable DESCENT_TAS Altitude_Table DESCENT_FUEL
global FeetInNauticalMiles Time Assn_Cruise_FL Distance Waypoints
global Current_Lat Current_Lon i Destination_Airport_Altitude Wind_Proj

global j true_air_speed rate_of_desc fuel_flow

ftpsecTOknots = 0.592483801;
% ------------------------------------------------------------------------

% --------------
% GET AIRCRAFT DATA
% --------------
DESCENT_TAS    = BADA_Aircraft_Data(Aircraft_Index).DESCENT_TAS;
DESCENT_FUEL   = BADA_Aircraft_Data(Aircraft_Index).DESCENT_FUEL;
Altitude_Table = BADA_Aircraft_Data(Aircraft_Index).FL * 100; % IN FEET
rodTable       = BADA_Aircraft_Data(Aircraft_Index).DESCENT_ROD; % RATE OF DESCENT
% ------------------------------------------------------------------------

% --------------
% CORRECT AIRCRAFT DATA FOR NON-ZERO AIRPORT ALTITUDE
% --------------
if Destination_Airport_Altitude > 0
   % USE HYDROSTATICS TO GET DENSITY RATIO
   h = Destination_Airport_Altitude*0.3048; % IN METERS ABOVE SEALEVEL
   L = -.0065; % LAPSE RATE IN TROPOSHERE< IN (KELVIN/METER)
   T_SL = 288.15; % SEA LEVEL TEMPERATURE IN KELVIN, ACCORDING TO BADA
   g0 = 9.81; % BASE GRAVITATIONAL CONSTANT, IN m/s^2
   R = 287.0368; % GAS CONSTANT, IN N.m/kg.K

   T_h = T_SL + L*h; % TEMPERATURE AT AIRPORT ALTITUDE, IN KELVIN
   sigma = (T_h/T_SL)^(-(g0/(L*R)+1)); % DENSITY RATIO, rho_h/rho_SL

   % CORRECT ALTITUDE TABLE TO START AT ABOUT AIRPORT ALTITUDE
   [~,Data_Fix_Index,~] = find(Altitude_Table>floor(Destination_Airport_Altitude),1);
   temp_TAS = DESCENT_TAS(Data_Fix_Index:end);
   temp_FUEL = DESCENT_FUEL(Data_Fix_Index:end);
```

```
    temp_ROD = rodTable(Data_Fix_Index:end);
    temp_ALT = Altitude_Table(Data_Fix_Index:end);
    DESCENT_TAS_new = temp_TAS;
    DESCENT_FUEL_new = temp_FUEL;
    rodTable_new = temp_ROD;
    Altitude_Table_new = temp_ALT;

    % REPLACE DATA WITH DENSITY RATIO-CORRECTED VALUES
    Length = length(DESCENT_TAS_new);
    DESCENT_TAS_new = DESCENT_TAS(1:Length)/sigma;
    DESCENT_FUEL_new = DESCENT_FUEL(1:Length)*sigma;
    rodTable_new = rodTable(1:Length)/sigma;

    % FOR ALTITUDES ABOVE 20000 FT, USE ORIGINAL BADA DATA
    [~,Index_Old_Data,~] = find(Altitude_Table == 20000);
    [~,Index_New_Data,~] = find(Altitude_Table_new == 20000);

    DESCENT_TAS_new(Index_New_Data:end) = DESCENT_TAS(Index_Old_Data:end);
    DESCENT_FUEL_new(Index_New_Data:end) = DESCENT_FUEL(Index_Old_Data:end);
    rodTable_new(Index_New_Data:end) = rodTable(Index_Old_Data:end);

    Altitude_Table = Altitude_Table_new;
    DESCENT_TAS = DESCENT_TAS_new;
    DESCENT_FUEL = DESCENT_FUEL_new;
    rodTable = rodTable_new;

end
% ---------------------------------------------------------------------

% --------------
% DEFINE BOUNDARY AND INITIAL CONDITIONS
% --------------
% DEFINE TIME SPAN (ONE HOUR IN LENGTH)
Time_Initial = 0.0;                    % INITIAL TIME (SECONDS)
Time_Final   = 3600.0;                   % FINAL TIME (SECONDS)
Time_Span    = [Time_Initial Time_Final];  % SPAN TIME

% DEFINE INITIAL MASS
Aircraft_Mass_Initial = 0; % KG

% DEFINE INITIAL STATE VARIABLES
yDescent_Initial = [Destination_Airport_Altitude Aircraft_Mass_Initial 0];
% y(1) - ALTITUDE (FEET)
% y(2) - WEIGHT (KG)
% y(3) - DISTANCE TRAVELLED (FEET)

% FLIP WAYPOINTS TO DO BACKWARDS CALCULATIONS FOR DESCENT PROFILE
Waypoints = flipud(Waypoints);

% DEFINE INITIAL F_Climb VARIABLES
i = 0;
j = 0;
Current_Lat = 0;
Current_Lon = 0;
Distance = 0;
Time = 0;
```

```
Wind_Proj = 0;
% ------------------------------------------------------------------------

% --------------
% GENERATE DESCENT PROFILE
% --------------
[Descent_Time,Descent_Data] = rk4sys('F_Descent', Time_Span, yDescent_Initial, 60, []);

% CORRECT ORDER OF WAYPOINTS BACK TO TRUE ORDER
Waypoints = flipud(Waypoints);

% FIND INDEX FOR REACHING CRUISE FLIGHT LEVEL
Altitude_ft = Descent_Data(:,1);
Index_To_Reach_MaxFL = find(Altitude_ft >= Assn_Cruise_FL, 1, 'first');
if isempty(Index_To_Reach_MaxFL) == 1
   Index_To_Reach_MaxFL = length(Altitude_ft)-1; % GET INDEX OF LAST ITEM IF NOTHING TO
TRUNCATE
end

true_air_speed = true_air_speed(1:Index_To_Reach_MaxFL);
rate_of_desc = rate_of_desc(1:Index_To_Reach_MaxFL);
fuel_flow = fuel_flow(1:Index_To_Reach_MaxFL);

% TRUNCATE DESCENT PROFILE TO STOP AT CRUISE FLIGHT LEVEL; CONVERT DATA
Altitude_ft           = Altitude_ft(1:Index_To_Reach_MaxFL);
Distance_nm           = Descent_Data(1:Index_To_Reach_MaxFL,3)/FeetInNauticalMiles;
Time_hrs              = Descent_Time(1:Index_To_Reach_MaxFL) / 3600;
Weight_kg             = Descent_Data(1:Index_To_Reach_MaxFL,2);

Rate_of_Descent_ftmin   = rate_of_desc';
Speed_knots           = true_air_speed';%(Distance_nm(2:end) - Distance_nm(1:end-1)) ./ (Time_hrs(2:end) -
Time_hrs(1:end-1));
FuelFlow_kgmin        = fuel_flow';%-(Weight_kg(2:end) - Weight_kg(1:end-1)) ./ (Time_hrs(2:end) -
Time_hrs(1:end-1)) / 60;
Distance_nm_For_Speed   = (Distance_nm(2:end) + Distance_nm(1:end-1)) / 2;
Total_Fuel_kg         = Weight_kg(1) - Weight_kg(end);

Latitude_Pts          = Current_Lat(1:Index_To_Reach_MaxFL);
Longitude_Pts         = Current_Lon(1:Index_To_Reach_MaxFL);
Wind_Vectors          = Wind_Proj(1:Index_To_Reach_MaxFL);

% ------------------------------------------------------------------------

% --------------
% SAVE DESCENT PROFILE IN STRUCTURE
% --------------
Descent_Profile.Distance_For_Descent_nm = max(abs(Distance_nm));
Descent_Profile.Time_For_Descent_hrs    = max(abs(Time_hrs));
Descent_Profile.Total_Fuel_kg           = Total_Fuel_kg;

Descent_Profile.Altitude_ft           = flipud(Altitude_ft);
Descent_Profile.Distance_nm           = -1*(flipud(Distance_nm) - max(Distance_nm));
Descent_Profile.Time_hrs              = Time_hrs;
Descent_Profile.Weight_kg             = -1*((flipud(Weight_kg) - min(Weight_kg)));

Descent_Profile.Rate_of_Descent_ftmin   = flipud(Rate_of_Descent_ftmin);
```

```
Descent_Profile.Speed_knots             = flipud(Speed_knots);
Descent_Profile.FuelFlow_kgmin          = flipud(FuelFlow_kgmin);
Descent_Profile.Distance_nm_For_Speed   = -1*(flipud(Distance_nm_For_Speed) -
max(Distance_nm_For_Speed));

Descent_Profile.Latitude_Pts          = flipud(Latitude_Pts);
Descent_Profile.Longitude_Pts         = flipud(Longitude_Pts);
Descent_Profile.Wind_Vectors_knots    = flipud(Wind_Vectors) * ftpsecTOknots;
% ------------------------------------------------------------------------

return
```

B.12 F_Descent.m

```
% This function generates points for the descent profile of the
% aircraft/airport combination
%
% yprime = F_Descent(t,y,Add)
%
% INPUT:
%   t   = time
%   y   = rate equations
%       y(1) = Aircraft altitude (feet)
%       y(2) = Aircraft weight (kilograms)
%       y(3) = Distance traveled along the path (feet)
%   Add = boolean for RK4 to run F_Descent
%
% OUTPUT:
%   yprime = derivative of y (rate equations)

function yprime_d = F_Descent(t,y,Add)

% --------------
% DEFINE GLOBAL VARIABLES
% --------------
global rodTable Altitude_Table DESCENT_TAS DESCENT_FUEL Assn_Cruise_FL
global Waypoints i Distance FeetInNauticalMiles Destination_Airport_Altitude
global Current_Lat Current_Lon Wind_Boolean Wind_Proj Month

global j true_air_speed rate_of_desc fuel_flow

pmTops = 1/60;        % CONVERT FROM FEET PER MINUTE TO FEET PER SECOND
knotsTofps = 1.6874;  % CONVERT KNOTS TO FEET PER SECOND
metersTofps = 3.280839895; %CONVERT METERS PER SECOND TO FEET PER SECOND

% ------------------------------------------------------------------------

% --------------
% EXTRACT CURRENT ALTITUDE
% --------------
Current_Altitude = min(y(1), Assn_Cruise_FL);
% ------------------------------------------------------------------------

% --------------
% CALCULATE WIND PROJECTION
% --------------
if i == 0 % CALCULATE INITIAL WIND PROJECTION
   i = i+1; % COUNTER

   % INITIAL LAT/LON POINTS
   Current_Lat(i,:) = Waypoints(1,1);
   Current_Lon(i,:) = Waypoints(1,2);

   % INITIAL HEADING
   Azimuth = azimuth(Current_Lat(i,:), Current_Lon(i,:), Waypoints(2,1), Waypoints(2,2));

   % INITIAL WIND PROJECTION
```

```
    Wind_Proj(i,:) = Wind_Calculator(Azimuth, Month, y(1), Current_Lat(i,:), Current_Lon(i,:));

elseif Add == true % RUN THIS CODE ONLY WHEN F_DESCENT IS CALLED THE FOURTH TIME BY THE
rk4sys ODE FUNCTION
    i = i+1; % COUNTER
    Distance(i,:) = y(3);

    % FIND INDEX OF WAYPOINTS CORRESPONDING TO CURRENT LAT/LON POINTS
    Change_in_Distance = (Distance(i,:) - Distance(i-1,:))/FeetInNauticalMiles;
    [no,Index] = min(abs(Current_Lat(i-1,:) - Waypoints(:,1)) + abs(Current_Lon(i-1,:) - Waypoints(:,2))); % Find
index of Waypoints corresponding to lat/lon

    % --------------
    % CORRECT INDEX
    % --------------
    % WAYPOINTS OF INDEX MUST BE FARTHER ALONG THAN CURRENT LAT/LON TO AVOID
BACKTRACKING
    if Index == length(Waypoints)
        Index = Index - 1;
    elseif Index <= length(Waypoints) - 1 % FOR INDICES BEFORE END OF WAYPOINTS

        % DETERMINE IF WAYPOINTS IS INCREASING OR DECREASING IN SIZE

        if Waypoints(Index+1,1) > Waypoints(Index,1)
            if Current_Lat(i-1,:) >= Waypoints(Index,1)
                Index = Index + 1;
            end
        elseif Waypoints(Index+1,1) < Waypoints(Index,1)
            if Current_Lat(i-1,:) <= Waypoints(Index,1)
                Index = Index + 1;
            end
        end

        if Index ~= length(Waypoints)
            if Waypoints(Index+1,2) > Waypoints(Index,2)
                if Current_Lon(i-1,:) >= Waypoints(Index,2)
                    Index = Index + 1;
                end
            elseif Waypoints(Index+1,2) < Waypoints(Index,2)
                if Current_Lon(i-1,:) <= Waypoints(Index,2)
                    Index = Index + 1;
                end
            end
        end

    else % FOR INDICES APPROACHING END OF WAYPOINTS
        if Waypoints(Index,1) > Waypoints(Index-1,1)
            if Current_Lat(i-1,:) >= Waypoints(Index,1)
                Index = Index + 1;
            end
        elseif Waypoints(Index,1) < Waypoints(Index-1,1)
            if Current_Lat(i-1,:) <= Waypoints(Index,1)
                Index = Index + 1;
            end
        end
```

```
        if Index == length(Waypoints)
            if Waypoints(Index,2) > Waypoints(Index-1,2)
                if Current_Lon(i-1,:) >= Waypoints(Index,2)
                    Index = Index + 1;
                end
            elseif Waypoints(Index,2) < Waypoints(Index-1,2)
                if Current_Lon(i-1,:) <= Waypoints(Index,2)
                    Index = Index + 1;
                end
            end
        end

    end

    % CORRECT INDEX IF GREATER THAN WAYPOINTS LENGTH
    if Index > length(Waypoints(:,1))
        Index = length(Waypoints(:,1));
    end
    % -----------------------------------------------------------------

    % --------------
    % CALCULATE HEADING
    % --------------
    Azimuth = azimuth(Current_Lat(i-1,:), Current_Lon(i-1,:), Waypoints(Index,1), Waypoints(Index,2)); %
INSTANTANEOUS ANGLE OF HEADING
    % -----------------------------------------------------------------

    % --------------
    % CALCULATE LAT/LON
    % --------------
    Change_in_Degrees = nm2deg(Change_in_Distance); % CONVERT DISTANCE NM TO DEGREES FOR USE
IN RECKON FUNCTION
    [Current_Lat(i,:),Current_Lon(i,:)] = reckon(Current_Lat(i-1,:),Current_Lon(i-1),Change_in_Degrees,Azimuth);
    % -----------------------------------------------------------------

    % --------------
    % CALCULATE WIND PROJECTION
    % --------------
    % SINCE DESCENT PROFILE IS NOT IN DIRECTION OF ORIGIN, MULTIPLY BY -1
    Wind_Proj(i,:) = Wind_Calculator(Azimuth, Month, y(1), Current_Lat(i,:), Current_Lon(i,:)) * -1;
    % -----------------------------------------------------------------
end

% WIND VS NO WIND
if Wind_Boolean == false
    Wind_Proj(i,:) = 0;
end
% -----------------------------------------------------------------

% --------------
% GET ODE VARIABLES
% --------------
if Current_Altitude < min(Altitude_Table)
    vtas        = custom_interp(fliplr(Altitude_Table),fliplr(DESCENT_TAS),Current_Altitude) * knotsTofps +
Wind_Proj(i,:) * metersTofps;   % SPEED IS TRUE AIRSPEED (TAS) IN FPS
```

```
   rateOfdescent = custom_interp(fliplr(Altitude_Table),fliplr(rodTable),Current_Altitude) * pmTops;
% IN FEET PER SECOND UNITS
   fuelFlow      = custom_interp(fliplr(Altitude_Table),fliplr(DESCENT_FUEL),Current_Altitude)  * pmTops;
% KILOGRAMS PER SECOND
else
   vtas          = custom_interp(Altitude_Table,DESCENT_TAS,Current_Altitude) * knotsTofps + Wind_Proj(i,:) *
metersTofps;                  % SPEED IS TRUE AIRSPEED (TAS) IN FPS
   rateOfdescent = custom_interp(Altitude_Table,rodTable,Current_Altitude) * pmTops;
% IN FEET PER SECOND UNITS
   fuelFlow      = custom_interp(Altitude_Table,DESCENT_FUEL,Current_Altitude)  * pmTops;
% KILOGRAMS PER SECOND
end
% ------------------------------------------------------------------------

if Add == true
   j = j+1;
   true_air_speed(j) = vtas/knotsTofps;
   rate_of_desc(j) = rateOfdescent/pmTops;
   fuel_flow(j) = fuelFlow/pmTops;
end

% --------------
% ASSIGN ODE VARIABLES
% --------------
% CHECK IF CRUISE ALTITUDE IS REACHED
if y(1) < Assn_Cruise_FL
   yprime_d(1) = rateOfdescent;
else
   yprime_d(1) = 0; % ZERO RATE OF DESCENT (FEET/S)
end

% MASS FLOW RATE (KG/S)
yprime_d(2) = - fuelFlow;

% DISTANCE TRAVELLED ALONG FLIGHT PATH (FEET)
yprime_d(3) = vtas;

% TRANSPOSE THE ARRAY GOING OUT
yprime_d = yprime_d';
% ------------------------------------------------------------------------

return
```

B.13 Generate_Cruise_Profile_Coef.m

```
% This function generates the cruise profile of the aircraft/airport
% combination being called
%
% Cruise_Profile = Generate_Cruise_Profile(Aircraft_Index, Climb_Profile, Descent_Profile, Cruise_Distance_nm)
%
% INPUT:
%   Aircraft_Index      = index of current aircraft in BADA
%   Climb_Profile       = inputs climb info for initial conditions of cruise
%   Descent_Profile     = inputs descent info for final conditions of cruise
%   Cruise_Distance_nm  = Track distance minus climb and descent distances
%
% OUTPUT:
%   Cruise_Profile  =  motion profile of aircraft's cruise

function Cruise_Profile = Generate_Cruise_Profile_Coef(Aircraft_Index, Climb_Profile, Descent_Profile,
Cruise_Distance_nm)

% --------------
% DEFINE GLOBAL VARIABLES
% --------------
global BADA_Aircraft_Data BADA_Aircraft_Coef Assn_Cruise_FL
global i Waypoints Wind_Boolean Month

ftpsecTOknots = 0.592483801;
mpsToknots = 1.94384449;    %convert wind speed from m/s to knots.
% ------------------------------------------------------------------------

% --------------
% DETERMINE WAYPOINTS FOR CRUISE ALTITUDE
% --------------
% REMOVE WAYPOINTS
Start_Lat_Lon = [Climb_Profile.Latitude_Pts(end) Climb_Profile.Longitude_Pts(end)];
End_Lat_Lon = [Descent_Profile.Latitude_Pts(1) Descent_Profile.Longitude_Pts(1)];
[no,Index_Start] = min(abs(Start_Lat_Lon(:,1) - Waypoints(:,1)) + abs(Start_Lat_Lon(:,2) - Waypoints(:,2)));
[no,Index_End] = min(abs(End_Lat_Lon(:,1) - Waypoints(:,1)) + abs(End_Lat_Lon(:,2) - Waypoints(:,2)));
if Index_Start == Index_End
   Cruise_Waypoints = [Start_Lat_Lon;End_Lat_Lon];
else
   Cruise_Waypoints = Waypoints;
   Cruise_Waypoints(Index_Start,:) = Start_Lat_Lon;
   Cruise_Waypoints(Index_End,:) = End_Lat_Lon;
   Cruise_Waypoints = Cruise_Waypoints(Index_Start:Index_End,:);
end

if isempty(Cruise_Waypoints) == 1
   Cruise_Profile = [];
   return
end
% ------------------------------------------------------------------------

% --------------
% DEFINE BOUNDARY AND INITIAL CONDITIONS
% --------------
```

```
% DEFINE INITIAL MASS
Initial_Cruise_Weight_kg = Climb_Profile.Weight_kg(end); % KG
Current_Aircraft_Weight = Initial_Cruise_Weight_kg;

% DEFINE TIME INTERVAL
deltaTime_hrs = 2/60; % INTERVAL OF 2/60 HOUR = 2 MINUTES

% DEFINE INITIAL LOOP VARIABLES
i = 0;
Distance_nm = 0;
Time_hrs = 0;
Current_Lat = 0;
Current_Lon = 0;
Wind_Proj = 0;
Ground_speed_knots = 0;
FuelFlow_kgmin = 0;
Weight_kg = 0;
Current_Distance_nm = 0;
Azimuth = 0;
% -------------------------------------------------------------------------

% --------------
% GET CRUISE FL
% --------------
Max_Climb_Altitude_ft      = Climb_Profile.Altitude_ft(end);
Cruise_Altitude_ft         = min(Assn_Cruise_FL, Max_Climb_Altitude_ft);
% -------------------------------------------------------------------------

% --------------
% GET CRUISE SPEED % put in while loop if Altitude changes
% --------------
Cruise_Speed_knots = Speed_Calculator_Cruise(Aircraft_Index, Cruise_Altitude_ft, BADA_Aircraft_Coef);
% -------------------------------------------------------------------------

% --------------
% CRUISE LOOP
% --------------
while Current_Distance_nm < Cruise_Distance_nm

   % COUNTER
   i = i+1;

   % --------------
   % GENERATE TIME
   % --------------
   if i == 1
      Time_hrs(i,:) = 0;
   else
      Time_hrs(i,:) = Time_hrs(i-1,:) + deltaTime_hrs;
   end
   % -------------------------------------------------------------------------

   % --------------
   % GENERATE DISTANCE
   % --------------
   if i == 1
```

```
    Distance_nm(i,:) = 0;
  else
    Change_In_Dist = Ground_speed_knots(i-1) * (Time_hrs(i) - Time_hrs(i-1));
    Distance_nm(i,:) = Distance_nm(i-1) + Change_In_Dist;
  end
  Current_Distance_nm = Distance_nm(i,:);
  % ---------------------------------------------------------------------

  % --------------
  % CALCULATE WIND PROJECTION AND LAT/LON WAYPOINT
  % --------------
  if i == 1 % CALCULATE INITIAL WIND PROJECTION

    % INITIAL LAT/LON POINTS
    Current_Lat(i,:) = Cruise_Waypoints(1,1);
    Current_Lon(i,:) = Cruise_Waypoints(1,2);

    % INITIAL HEADING
    Azimuth(i,:) = azimuth(Current_Lat(i,:), Current_Lon(i,:), Cruise_Waypoints(2,1), Cruise_Waypoints(2,2));

    % INITIAL WIND PROJECTION
    if Wind_Boolean == 1
      Wind_Proj(i,:) = Wind_Calculator(Azimuth(i,:), Month,Cruise_Altitude_ft, Current_Lat(i,:),
Current_Lon(i,:))*mpsToknots;
    end

  else % CALCULATE REMAINING WIND PROJECTIONS

    % FIND INDEX OF WAYPOINTS CORRESPONDING TO CURRENT LAT/LON POINTS
    Change_in_Distance = (Distance_nm(i,:) - Distance_nm(i-1,:));
    [no,Index] = min(abs(Current_Lat(i-1,:) - Cruise_Waypoints(:,1)) + abs(Current_Lon(i-1,:) -
Cruise_Waypoints(:,2)));

    % --------------
    % CORRECT INDEX
    % --------------
    % WAYPOINTS OF INDEX MUST BE FARTHER ALONG THAN CURRENT LAT/LON TO
    % AVOID BACKTRACKING
    if Index == length(Cruise_Waypoints)
      Index = Index - 1;
    elseif Index <= length(Cruise_Waypoints) - 1 % FOR INDICES BEFORE END OF WAYPOINTS

      % DETERMINE IF WAYPOINTS IS INCREASING OR DECREASING IN SIZE

      if Cruise_Waypoints(Index+1,1) > Cruise_Waypoints(Index,1)
        if Current_Lat(i-1,:) >= Cruise_Waypoints(Index,1)
          Index = Index + 1;
        end
      elseif Cruise_Waypoints(Index+1,1) < Cruise_Waypoints(Index,1)
        if Current_Lat(i-1,:) <= Cruise_Waypoints(Index,1)
          Index = Index + 1;
        end
      end

      if Index ~= length(Cruise_Waypoints)
        if Cruise_Waypoints(Index+1,2) > Cruise_Waypoints(Index,2)
```

```
            if Current_Lon(i-1,:) >= Cruise_Waypoints(Index,2)
               Index = Index + 1;
            end
         elseif Cruise_Waypoints(Index+1,2) < Cruise_Waypoints(Index,2)
            if Current_Lon(i-1,:) <= Cruise_Waypoints(Index,2)
               Index = Index + 1;
            end
         end
      end

   else % FOR INDICES APPROACHING END OF WAYPOINTS
      if Cruise_Waypoints(Index,1) > Cruise_Waypoints(Index-1,1)
         if Current_Lat(i-1,:) >= Cruise_Waypoints(Index,1)
            Index = Index + 1;
         end
      elseif Cruise_Waypoints(Index,1) < Cruise_Waypoints(Index-1,1)
         if Current_Lat(i-1,:) <= Cruise_Waypoints(Index,1)
            Index = Index + 1;
         end
      end

      if Index == length(Cruise_Waypoints)
         if Cruise_Waypoints(Index,2) > Cruise_Waypoints(Index-1,2)
            if Current_Lon(i-1,:) >= Cruise_Waypoints(Index,2)
               Index = Index + 1;
            end
         elseif Cruise_Waypoints(Index,2) < Cruise_Waypoints(Index-1,2)
            if Current_Lon(i-1,:) <= Cruise_Waypoints(Index,2)
               Index = Index + 1;
            end
         end
      end

   end

   while (1)
      [~,Dist_to_Next_Waypt] = legs([Current_Lat(i-1,:) Cruise_Waypoints(Index,1)], [Current_Lon(i-1,:)
Cruise_Waypoints(Index,2)]);
      if Dist_to_Next_Waypt < Change_in_Distance
         Index = Index + 1;
         if Index >= length(Cruise_Waypoints)
            Index = length(Cruise_Waypoints)-1;
            break
         end
      else
         break
      end
   end

   % CORRECT INDEX IF GREATER THAN WAYPOINTS LENGTH
   if Index > length(Cruise_Waypoints(:,1))
      Index = length(Cruise_Waypoints(:,1));
   end
   % -----------------------------------------------------------------

   % --------------
```

```
        % CALCULATE HEADING
        % --------------
        Azimuth(i,:) = azimuth(Current_Lat(i-1,:), Current_Lon(i-1,:), Cruise_Waypoints(Index,1),
Cruise_Waypoints(Index,2)); % Instantaneous angle of heading

        % CORRECT ANGLE IF IT POINTS IN OPPOSITE DIRECTION OF AIRCRAFT
        % DIRECTION
%         if Azimuth(i,:) > (Azimuth(i-1,:)+100) || Azimuth(i,:) < (Azimuth(i-1,:)-100)
%             Azimuth(i,:) = Azimuth(i-1);
%         end
        % -----------------------------------------------------------------

        % --------------
        % CALCULATE LAT/LON
        % --------------
        Change_in_Degrees = nm2deg(Change_in_Distance); % CONVERT DISTANCE NM TO DEGREES FOR
USE IN RECKON FUNCTION
        [Current_Lat(i,:),Current_Lon(i,:)] = reckon(Current_Lat(i-1,:),Current_Lon(i-
1),Change_in_Degrees,Azimuth(i,:));
        % -----------------------------------------------------------------

        % --------------
        % CALCULATE WIND PROJECTION
        % --------------
        if Wind_Boolean == 1
            Wind_Proj(i,:) = Wind_Calculator(Azimuth(i,:), Month,Cruise_Altitude_ft, Current_Lat(i,:),
Current_Lon(i,:))*mpsToknots;
        end
        % -----------------------------------------------------------------

    end

    % WIND VS NO WIND
    if Wind_Boolean == false
        Wind_Proj(i,:) = 0;
    end
    % -----------------------------------------------------------------

    % --------------
    % GENERATE GROUND SPEED
    % --------------
    Ground_speed_knots(i,:) = Cruise_Speed_knots + Wind_Proj(i,:);
    % -----------------------------------------------------------------

    % --------------
    % GENERATE WEIGHT
    % --------------
    if i == 1
        Weight_kg(i,:) = Initial_Cruise_Weight_kg;
    else
        Weight_kg(i,:) = Weight_kg(i-1,:) - FuelFlow_kgmin(i-1,:)*deltaTime_hrs*60; % weight change over entire
delta t
    end
    % -----------------------------------------------------------------

    % --------------
```

```
   % GENERATE FUEL CONSUMPTION
   % --------------
   FuelFlow_kgmin(i,:) = FuelFlow_Calculator_Cruise(Aircraft_Index, Cruise_Altitude_ft, Cruise_Speed_knots,
Weight_kg(i,:), BADA_Aircraft_Coef);
   % ------------------------------------------------------------------------

   % --------------
   % GENERATE ALTITUDE
   % --------------
   Altitude_ft(i,:) = Cruise_Altitude_ft;
   % ------------------------------------------------------------------------

end % while Current_Distance_nm < Cruise_Distance_nm
% ------------------------------------------------------------------------

% --------------
% EXTRACT FINAL PROFILE POINTS
% --------------
Weight_kg(i,:)      = Weight_kg(i-1,:) - FuelFlow_kgmin(i-1,:);
FuelFlow_kgmin(i,:) = FuelFlow_Calculator_Cruise(Aircraft_Index, Cruise_Altitude_ft, Cruise_Speed_knots,
Weight_kg(i,:), BADA_Aircraft_Coef);
Distance_nm(i,:)    = Cruise_Distance_nm;
Time_hrs(i,:)       = Time_hrs(i-1,:) + (Distance_nm(i)-Distance_nm(i-1))/Ground_speed_knots(i);
Total_Fuel_kg       = Weight_kg(1) - Weight_kg(end);
% ------------------------------------------------------------------------

% --------------
% SAVE CRUISE PROFILE IN STRUCTURE
% --------------
Cruise_Profile.Cruise_Distance_nm   = Cruise_Distance_nm;
Cruise_Profile.Cruise_Time_hrs      = Time_hrs(end) - Time_hrs(1);
Cruise_Profile.Total_Fuel_kg        = Total_Fuel_kg;
Cruise_Profile.Cruise_Speed_knots   = Distance_nm(end)/Time_hrs(end);
Cruise_Profile.Altitude_ft          = Altitude_ft;

% GET LAST STEP
Cruise_Profile.Distance_nm          = (Distance_nm + Climb_Profile.Distance_nm(end));
Cruise_Profile.Time_hrs             = (Time_hrs + Climb_Profile.Time_hrs(end));
Cruise_Profile.Weight_kg            = Weight_kg;
Cruise_Profile.FuelFlow_kgmin       = FuelFlow_kgmin;
Cruise_Profile.Ground_speed_knots   = Ground_speed_knots;

Cruise_Profile.Latitude_Pts         = Current_Lat;
Cruise_Profile.Longitude_Pts        = Current_Lon;
Cruise_Profile.Wind_Vectors_knots   = Wind_Proj;
% ------------------------------------------------------------------------

return
```

B.14 FuelFlow_Calculator_Cruise.m

```
% Fuel Flow Calculator
%
% [VTAS] = Speed_Calculator(Aircraft_Index, Altitude)
%
% INPUT:
%   Aircraft_Index      = index of current aircraft in BADA
%   Altitude            = current altitude of aircraft <ft>
%   VTAS                = current true airspeed <kts>
%   AC_Mass             = current mass of aircraft <kg>
%   BADA_Aircraft_Coef  = structure of BADA coefficients from APF file for
%                         aircraft
%
% OUTPUT:
%   Fuel_Flow           = current fuel flow <kg/min>

function [Fuel_Flow] = FuelFlow_Calculator_Cruise(Aircraft_Index, Altitude, VTAS, AC_Mass,
BADA_Aircraft_Coef)

% DEVIATIONS FROM ISA FOR NOW ASSUMED TO BE ZERO:
delta_temp = 0;

% CONVERT
kts2m_s = 0.514444444444; % kts to m/s
ft2m = 0.3048; % ft to m
% --------------
% MSL STANDARD CONDITIONS
% --------------
temp_0 = 288.15; % standard temperature at MSL <K>
pres_0 = 101325; % standard pressure at MSL <Pa>
dens_0 = 1.225; % standard density at MSL <kg/m^3>
a_0 = 340.294; % speed of sound at MSL <m/s>
% ------------------------------------------------------------------------

% --------------
% PHYSICAL CONSTANTS
% --------------
gamma = 1.4; % adiabatic index of air (kappa in BADA User Manual
R = 287.05287; % real gas constant for air <m^2/K.s^2>
g0 = 9.80665; % gravitational acceleration <m/s^2>
beta = -0.0065; % ISA temerature gradient below tropo (lapse rate L) <K/m>
r_earth = 6.371e+6; % RADIUS OF EARTH
C_Tcr = 0.95; % BADA-defined maximum cruise thrust coefficient for all aircraft
% ------------------------------------------------------------------------

% --------------
% ATMOSPHERIC CONDITIONS
% --------------
Altitude_m = ft2m * Altitude;
Hp = (r_earth / (r_earth + Altitude_m)) * Altitude_m; % geopotential altitude <m>
Hp_tropo = 11000; % geopotential alt at tropopause <m>

Hp = Altitude_m;
```

```
% MEAN SEA LEVEL (MSL)

if Hp < Hp_tropo
   temp = temp_0 + delta_temp + beta*Hp;
   pres = pres_0*((temp - delta_temp)/temp_0)^(-g0/(beta*R));
elseif Hp == Hp_tropo
   temp = temp_0 + delta_temp + beta*Hp_tropo;
   pres = pres_0*((temp - delta_temp)/temp_0)^(-g0/(beta*R));
elseif Hp > Hp_tropo
   temp_tropo = temp_0 + delta_temp + beta*Hp_tropo;
   pres_tropo = pres_0*((temp_tropo - delta_temp)/temp_0)^(-g0/(beta*R));
   temp_ISA_tropo = temp_0 + beta*Hp_tropo;
   temp = temp_tropo;
   pres = pres_tropo * exp(-g0/(R*temp_ISA_tropo)*(Hp-Hp_tropo));
end
dens = pres/(R*temp);
% ---------------------------------------------------------------------

% ----------------
% THRUST
% ----------------
eng_type = BADA_Aircraft_Coef(Aircraft_Index).AC_Type.eng_type;

% % % MAX CLIMB THRUST - PTF tables don't consider this??
% % C_Tc1 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc1;
% % C_Tc2 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc2;
% % C_Tc3 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc3;
% % C_Tc4 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc4;
% % C_Tc5 = BADA_Aircraft_Coef(Aircraft_Index).Thrust.C_Tc5;
% %
% % Hp_ft = (r_earth / (r_earth + Altitude)) * Altitude;
% %
% % if strcmp(eng_type,'Jet') == 1
% %     Thrust_max_cl_ISA = C_Tc1*(1 - Hp_ft/C_Tc2 + C_Tc3*Hp_ft^2);
% % elseif strcmp(eng_type,'Turboprop') == 1
% %     Thrust_max_cl_ISA = C_Tc1/VTAS*(1 - Hp_ft/C_Tc2) + C_Tc3;
% % elseif strcmp(eng_type,'Piston') == 1
% %     Thrust_max_cl_ISA = C_Tc1*(1 - Hp_ft/C_Tc2) + C_Tc3/VTAS;
% % end
% %
% % % IF NOT ISA:
% % if delta_temp ~= 0
% %     delta_temp_eff = delta_temp - C_Tc4;
% %     Thrust_max_cl = Thrust_max_cl_ISA * (1 - C_Tc5*delta_temp_eff);
% % else
% %     Thrust_max_cl = Thrust_max_cl_ISA;
% % end
% %
% % % MAX POSSIBLE CRUISE THRUST
% % Thrust_max_cr = C_Tcr * Thrust_max_cl;

% THRUST = DRAG IN CRUISE
S = BADA_Aircraft_Coef(Aircraft_Index).Aero.S; % planform area <m^2>
VTAS_m_s = kts2m_s * VTAS;

C_L = 2*AC_Mass*g0/(dens*VTAS_m_s^2*S);
```

```
C_D0_CR = BADA_Aircraft_Coef(Aircraft_Index).Aero.C_D0_CR;
C_D2_CR = BADA_Aircraft_Coef(Aircraft_Index).Aero.C_D2_CR;
C_D = C_D0_CR + C_D2_CR*C_L^2;

Drag = 1/2*C_D*dens*VTAS_m_s^2*S;

Thrust_drag = Drag;

Thrust = Thrust_drag;
% Thrust = min(Thrust_drag,Thrust_max_cr);
% -------------------------------------------------------------------------

% ----------------
% FUEL FLOW
% ----------------
C_f1 = BADA_Aircraft_Coef(Aircraft_Index).Fuel.C_f1;
C_f2 = BADA_Aircraft_Coef(Aircraft_Index).Fuel.C_f2;
C_fcr = BADA_Aircraft_Coef(Aircraft_Index).Fuel.C_fcr;

% THRUST SPECIFIC FUEL CONSUMPTIONS
if strcmp(eng_type,'Jet') == 1
   TSFC = C_f1*(1+VTAS/C_f2);
elseif strcmp(eng_type,'Turboprop') == 1
   TSFC = C_f1*(1-VTAS/C_f2)*(VTAS/1000);
end

% FUEL FLOW
Thrust_kN = Thrust/1000; % convert thrust from <N> to <kN>

if strcmp(eng_type,'Piston') == 1
   f_cr = C_f1*C_fcr;
else
   f_cr = TSFC * Thrust_kN * C_fcr;
end
% -------------------------------------------------------------------------

Fuel_Flow = f_cr;

end
```