GLOBAL COMMERCIAL AIRCRAFT FUEL BURN AND EMISSIONS FORECAST: 2016 TO 2040

Rahul Padalkar

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
Gerardo Flintsch
Montasir M. Abbas

August 4, 2017
Blacksburg, Virginia

Keywords: Simulation Model, Fuel Consumption, Emissions Model, NASA's N+2 Vehicles, N+1 New Generation Aircraft, TPADS Table

Copyright 2017, Rahul Padalkar

GLOBAL COMMERCIAL AIRCRAFT FUEL BURN AND EMISSIONS FORECAST: 2016 TO 2040

Rahul Padalkar

ABSTRACT

This thesis discusses enhancements to the Global Demand Model (GDM). The model addresses the need to predict: a) number of flights Worldwide by Origin-Destination (OD) airport pair, b) the number of seats (surrogate of demand) by OD airport pair, c) the fleet evolution over time, d) fuel consumption by OD pair and aircraft type, and emissions by OD pair and aircraft type. The model has developed an airline fleet assignment module to predict changes to the airline fleet in the future. Specifically, the model has the capability to examine the fuel and emission benefits if next generation N+1 aircraft and advanced NASA's N+2 aircraft are adopted in the future.

GLOBAL COMMERCIAL AIRCRAFT FUEL BURN AND EMISSIONS FORECAST: 2016 TO 2040

Rahul Padalkar

GENERAL AUDIENCE ABSTRACT

This thesis discusses enhancements to a model, Global Demand Model (GDM), developed at Air Transportation Systems Laboratory at Virginia Tech. The model addresses the need to predict: a) number of flights Worldwide by Origin-Destination (OD) airport pair, b) the number of seats (surrogate of demand) by OD airport pair, c) the fleet evolution over time, d) fuel consumption by OD pair and aircraft type, and emissions by OD pair and aircraft type. The model has developed an airline fleet assignment module to predict changes to the airline fleet in the future. Specifically, the model has the capability to examine the fuel and emission benefits if next generation N+1 aircraft and advanced NASA's N+2 aircraft are adopted in the future.

ACKNOWLEGEMENTS

First, I must thank Dr. Antonio Trani for offering me the opportunity to work at his lab, guiding me and supporting me through my graduate study at Virginia Tech. It is a great honor to work with him in the Air Transportation Systems Laboratory. His knowledge and easy-going personality benefited me a lot. His patience and humbleness have left an everlasting impression.

My sincere gratitude to my committee members: Dr. Montasir M. Abbas and Dr. Gerardo Flintsch for their patience and feedback on my thesis.

The research and development effort in this study was carried out as a part of a research project for NASA Langley Research Center. I am grateful for the opportunity to work on this project and would like to acknowledge Ty Vincent Marien, Sam Dollyhigh and Tech-Seng Kwa for their guidance and valuable input during the development of the project

I am particularly thankful to Nicolas Hinze my lab mates Edwin Freire and Arman Izadi for their assistance in the research task. It has been a great experience working with these people in the enhancement of the Global Demand Model.

Last and most importantly, I must express my profound gratitude to my family for their unconditional support and continuous encouragement through my years of study. To my parents, thank you for trusting me all the time and letting me choose my career.

ATTRIBUTION

Edwin Freire, a PhD student in Civil Engineering (Transportation Infrastructure and Systems Engineering), collaborated in the development of the Global Demand Model. Sections 3.2.1 and 3.2.2 are contributions provided by Edwin.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

The Aeronautics Research Mission Directorate (ARMD) at NASA Headquarters is responsible for establishing a strategic systems analysis capability focused on understanding the system-level impacts of NASA programs, the potential for integrated solutions, and the development of high-leverage options for new investment and partnership. To this end, ARMD's Portfolio Analysis Management Office (PAMO) has tasked the Systems Analysis and Concepts Directorate at NASA Langley to formalize, develop, and utilize, a framework that efficiently employs a variable fidelity analysis capability to aid in such assessments.

PAMO has developed a global aviation demand forecasting capability (called Global Demand Model hereon) that can forecast the annual commercial air traffic operations between 3,630 airports worldwide. The Global Demand Model (GDM) can predict trip demand considering Gross Domestic Product (GDP), population trends and distribute those trips to other airports worldwide, creating a global air transportation network. This thesis explains enhancements made to the GDM model, taking it from prototype stage to a more mature demand prediction tool. The research explains the following new features of the GDM model:

- Developed a fuel burn module to predict global fuel consumption due to commercial airline operations.
- Developed a global emissions module to predict commercial airline greenhouse emissions (CO_2) and Landing and Takeoff Cycle emissions (LTO).
- Added capability to the GDM model to summarize results in NASA's TPADS format.

This thesis describes enhancements to the Global Demand Model (GDM) with a focus on the fuel consumption and emissions module. The enhancements made address the need to predict fuel consumption by OD pair and aircraft type, and emissions by OD pair and aircraft type. A model flowchart of the new features of the GDM model

are shown in Figure 1-1. The figure shows the computational modules added to GDM including: a) a global trip distribution module, b) a fleet evolution analysis module, c) a flight trajectory generation, d) and modules to estimate fuel and emissions produced by commercial aviation operations. The highlighted part inFigure 1-1: Flowchart of the GDM Model. Figure 1-1 shows the scope of this paper.

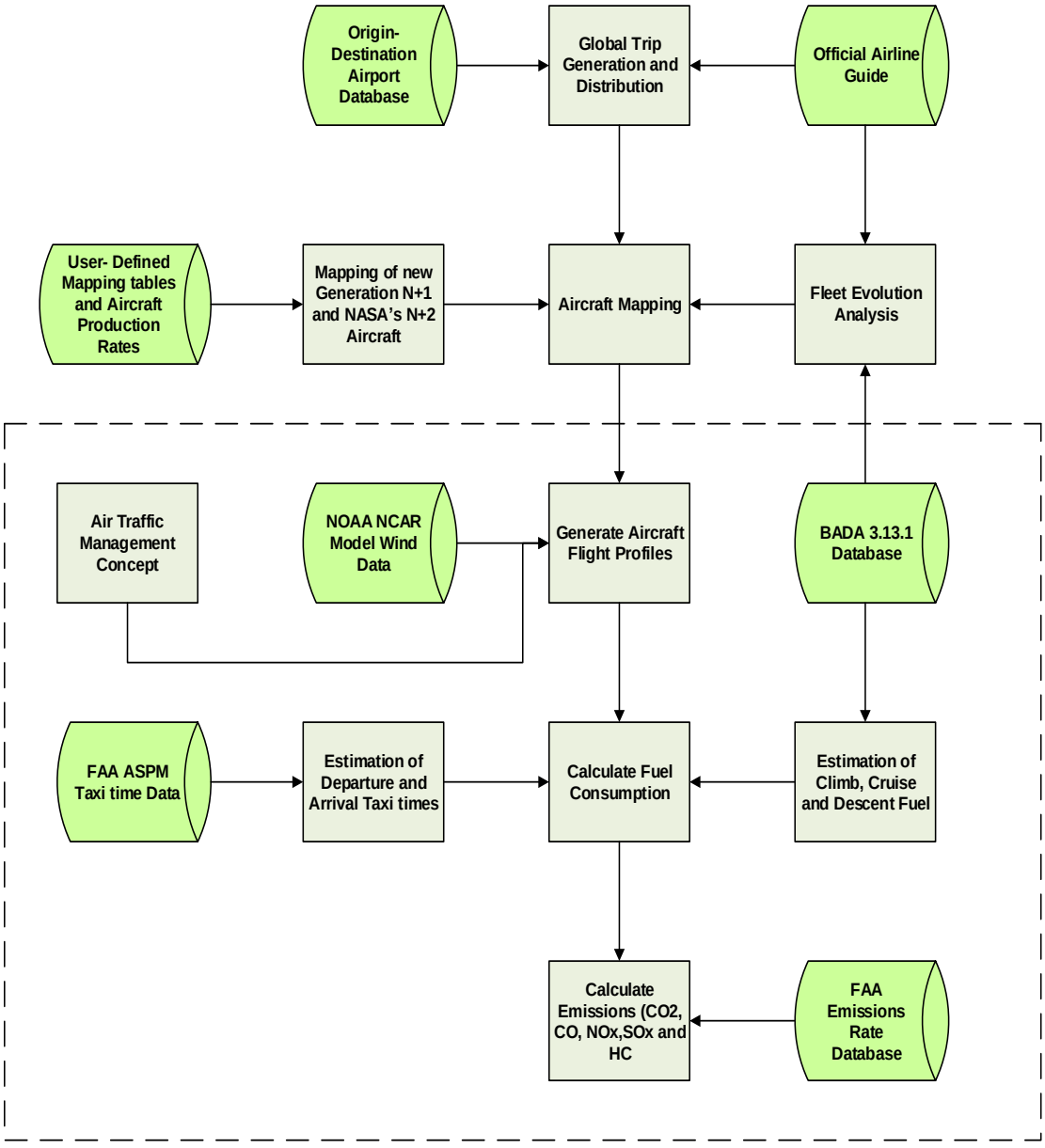


Figure 1-1: Flowchart of the GDM Model.

CHAPTER 2 LITERATURE REVIEW

2.1 Overview

The fuel and emissions model developed has two sub-modules: 1) a fuel consumption module and 2) an emissions module. In this chapter, we present past efforts to model aviation fuel consumption and emissions. In the recent years, 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 2004 Wing-Ho and Trani developed a method to estimate fuel consumption using an artificial neural network (Wing-Ho, 2004). The technique of neural network was introduced in their work and it could be trained to estimate fuel consumption for a specific aircraft. In the model, the aircraft fuel consumption data was obtained from the flight performance manual of individual aircraft which would be then imported into the neural network. The network was trained using the Levenberg-Marquardt Algorithm.(LMQ) 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.

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

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

where 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.

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. Chatterji models altitude, airspeed and thrust which are used 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 depend 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 a Bombardier Global 5000 to a Bombardier RJ-900 Regional Jet which can lead to some inconsistencies in characteristics of aircraft performance.

Committee on Aviation and Environmental Protection (CAEP), in 2013, presented present and future trends in aircraft noise and emissions. The paper discusses 9 different scenarios in terms of aircraft technology advancement rates and advanced operational

improvements, with the base year 2010. According to the CAEP, approximately 65 per cent of global aviation fuel consumption was from international aviation for the year 2010 and is to increase up to 70 per cent by 2050. FAA's Aviation Environmental Design Tool (AEDT), EUROCONTROL's Advanced Emissions Model (AEM) and Manchester Metropolitan University's Future Civil Aviation Scenario Software Tool (FAST) were the models employed to make the forecast (ICAO, 2013).

In 2014, efforts were made to compare the fuel burn rates with variations by seat configuration and stage distance. (Kelly, Park, 2014) The authors use EMEP/ EEA (European Monitoring and Evaluation Programme for European Environment Agency) inventory database for the fuel burn calculations per nautical for aircraft and OAG flight schedule data to calculate the fuel burn by routes and aircraft types.

Most recently, Wasuik et al proposed a model for the estimation of global and regional commercial fuel burn and emissions (Wasuik et al., 2015). A software named Aircraft Performance Model Implementation (APMI) was developed for the same which includes a research for a mathematical model of aircraft performance for all phases of the flight, a mission fuel burn algorithm, domestic and international reserve fuel requirement calculations, a cruise altitude allocation method and a cruise distance calculation procedure.

Tools used for the development of APMI were: CAPSTATS database for obtaining commercial air traffic movements from 2005 to 2011, EUROCONTROL Base of Aircraft Data (BADA) model for the aircraft performance for all phases of a flight and Boeing Fuel Flow Method 2 (BFFM2) for the emission calculations. The final results were obtained from the APMI software were compared with estimates published by System of assessing Global Aircraft Emissions(SAGE), a model developed by US Federal Aviation Administration (FAA), for the year 2006.

2.2.2 Emission Model

Several models and calculators have been developed to estimate emissions. One such model is the ICAO's (International Civil Aviation Organization) paper presented by CAEP. The committee presented CO2 emissions for international aviation from 2005 to 2040, and then extrapolated to 2050. The trends also account for sustainable alternative fuels from 2020 to 2050. The emissions created from the production of jet fuel are assumed to be 0.51 times the fuel amount and from their combustion, compute the carbon emission using a multiplicative constant of 3.16 times the fuel amount. The projections made suggest that by 2040 607Mt and by 2050 1,210Mt of CO2 emissions are expected. This could be reduced substantially with the use of alternative fuels by 25% (ICAO, 2013).

2.3 Data Analysis

2.3.1 OAG Data

The Official Airline Guide (OAG) is an air travel intelligence company that provides digital information and applications to the world's airlines, airports, government agencies and travel-related service companies. OAG compiles airline schedules into a database consisting of future and historical flight details for more than 900 airlines and over 4,000 airports. Because OAG demand set is comprehensive, it is used to identify gaps in the TFMS demand set.

Edwin Freire forecasted airport demand by using the (OAG) database spanning from 1996 to 2015. He then developed a Fratar trip distribution model to distribute commercial airline trips worldwide between over 3600 airports using the aircraft fleet assigned to the airport pairs in the OAG 2015 data.

2.3.2 BADA Data

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, 2015). 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 of the 194 modeled aircraft types in EUROCONTROL BADA 3.13.1. 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[1].

In this study EUROCONTROL BADA 3.13.1 has been employed for 39 aircraft types. The list of the aircraft included in GDM are presented in Table 2-1.

2.3.3 TFMS Data

Traffic Flow Management System (TFMS, previously ETMS) is a data exchange system for supporting the management and monitoring of national air traffic flows. TFMS processes all available data sources such as flight plan messages, flight plan amendment messages, and departure and arrival messages. The FAA Airspace Lab

[1] Total Energy Model equates the rate of work done by forces acting on the aircraft with the rate of increase in potential and kinetic energy. It can be considered as a reduced point-mass model.

assembles TFMS flight messages into one record per flight. TFMS is restricted to the subset of flights that fly under Instrument Flight Rules (IFR) and are captured by the FAA enroute computers (FAA, 2014).

Table 2-1: Global Demand Model Aircraft Fleet Based on OAG 2015 Commercial Flight Data

Aircraft			
Airbus 310	Boeing 717-200	Boeing 767-300	Bombardier Regional Jet CRJ-200
Airbus 319	Boeing 737-300	Boeing 767-400	Bombardier Regional Jet CRJ-900
Airbus 320	Boeing 737-500	Boeing 777-200	Bombardier Havilland Dash 8-300
Airbus 321	Boeing 737-700	Boeing 777-300	Bombardier Havilland Dash 8-200
Airbus 330-200	Boeing 737-800	Boeing 777-200L	Embraer ERJ-135
Airbus 330-300	Boeing 737-900	Boeing 777-300W	Embraer ERJ-145
Airbus 340-600	Boeing 747-400	Boeing 787-800	Embraer ERJ-170
Airbus 380-800	Boeing 747-800	Beechcraft 99	Embraer ERJ-190
Avions-de-Transport-Régional42-500	Boeing 757-200	Cessna 208	McDonnell Douglas MD-82
Avions-de-Transport Régional72-5005	Boeing 767-200	Canadair Challenger 600	

The FAA provided the Air Transportation Systems Laboratory at Virginia Tech with demand sets for eight seed days of traffic (see Table 2-2). Each demand set includes flight information one day before and after the seed day.

The TFMS database contains flight waypoint information and cruise altitude. In the analysis of cruise altitude assignment, the TFMS data are sorted for each unique OD pair and aircraft type combination and segregated for different headings (East or West). The TFMS database includes thousands of distinct aircraft. TFMS aircraft types are mapped to 39 BADA aircraft types.

Table 2-2: Eight TFMS Seed Days.

01/28/2016	03/28/2016	03/31/2016	05/28/2016
06/25/2016	08/16/2016	10/17/2016	11/01/2016

2.3.4 Wind Data

Wind data used for modeling is collected from the National Center for Atmospheric Research (NCAR) Reanalysis model (NOAA/ESRL/PSD 2014). The reanalysis model was developed by Earth System Research Laboratory Physical Science Division. The data provides the magnitude and direction of the wind every 2.5 degrees latitude and longitude at 17 geopotential heights (Li, 2014).

A map of wind vector over the earth on June 25, 2016 is shown in Figure 2-1.

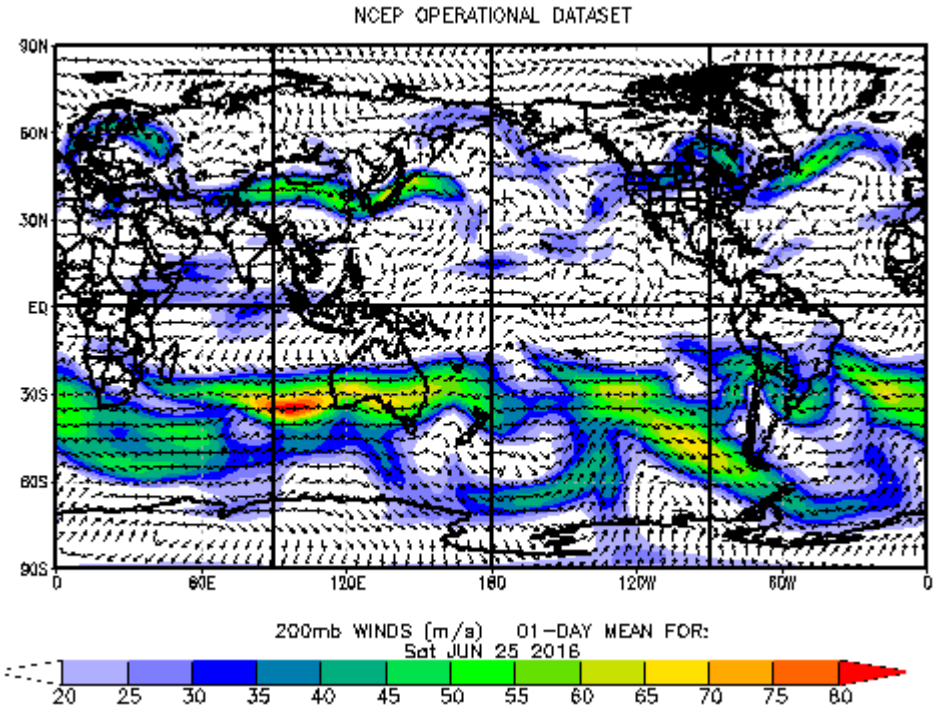


Figure 2-1: Wind Vector at 200 mb (39,000 ft.) on June 25, 2016 from NCAR Model (Speed units in metre/second)

2.3.5 EDMS Data

To obtain fuel burn data for ground operations the FAA Emission and Dispersion Modeling System (EDMS) has been selected. Aircraft fuel burn rate at idle conditions in the EDMS dataset is used to calculate the commercial flight fuel consumption for ground operations. 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).

CHAPTER 3 METHODOLOGY

3.1 Global Demand Model

In 2015 Osama Alsalous developed an econometric regression model that predicts the number of air passenger seats worldwide using Gross Domestic Product (GDP), population, and airline market share as the explanatory variable called the Global Demand Model. (GDM). The model estimated the number of seats offered at 3,017 airports Worldwide (Alsalous 2015).

In 2016 the Air Transportation System Laboratory further enhanced the Global Demand Model. The current model has five main modules. The contributions of this thesis is in steps 4 and 5 in Figure 3-1.

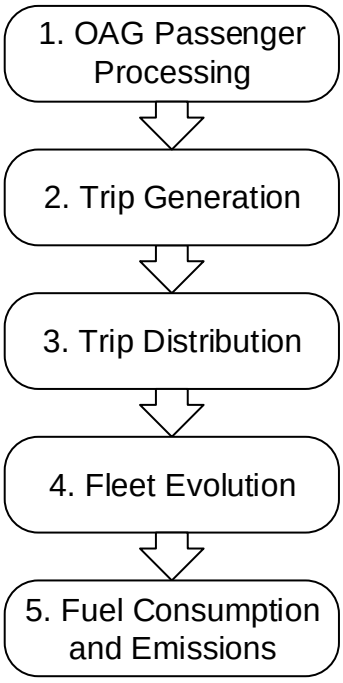


Figure 3-1: Flowchart of Steps Employed in the Global Demand Model Process.

The GDM is a five-step simulation procedure as shown in Figure 3-1. The OAG Passenger Processing module uses the OAG data to generate the airport and aircraft list to be further used by the other modules. The trip generation module forecasts demand between the airports and creates an Origin-Destination (OD) matrix. The aircraft fleet assignment and seat distribution is done in the trip distribution module. The fleet

evolution module studies the effect of retirement and replacement of existing aircraft with the new generation aircraft from years 2016 to 2040. The fuel consumption and emission module predicts global fuel consumption for each aircraft operating between all the OD pairs generated in the trip generation module.

The fuel consumption and emission module works with the aid of several user defined input data tables used for its simulation. Users can define several input parameters like the detour factor, flight assignment separation, the flight level increment for step cruise and cruise check distance for the step cruise. Separate tables for the new generation aircraft and the NASA's N+2 advanced vehicles are provided for the application of fuel consumption and emission saving factors, which is further discussed in section 3.2.3.1.

A matrix consisting of the 39 aircraft (Table 2-1) was developed for the study. The purpose of this matrix was to expand the traditional BADA files to model oceanic operations and flight assignment, fuel burn calculations. The BADA model contains equations, coefficients and speed limitations in four separate files. Some of these aircraft parameters are summarized into this file. BADA does not contain several operational parameters that are introduced in this file. For example, BADA does not account for buffer boundaries in the estimation of the maximum altitude the aircraft can reach for a given mass. Similarly, BADA does not contain information about takeoff mass that is critical in oceanic operations. The purpose of this file is to expand on those topics and provide a richer set of information to each aircraft to model oceanic operations more realistically.

3.2 Improvements to Global Demand Model

3.2.1 Airport Demand Forecast Predictions (by Edwin Freire)

The airline network database – OAG was upgraded from 1996 to 2015, to improve the airport demand forecasts. The enhanced version of the GDM employs a Fratar trip distribution model to distribute future airline seats among airports. Seats are used as a

surrogate for passenger demand in this model since there are no global databases containing the number of passengers between each origin and destination airport for 3,630 airports.

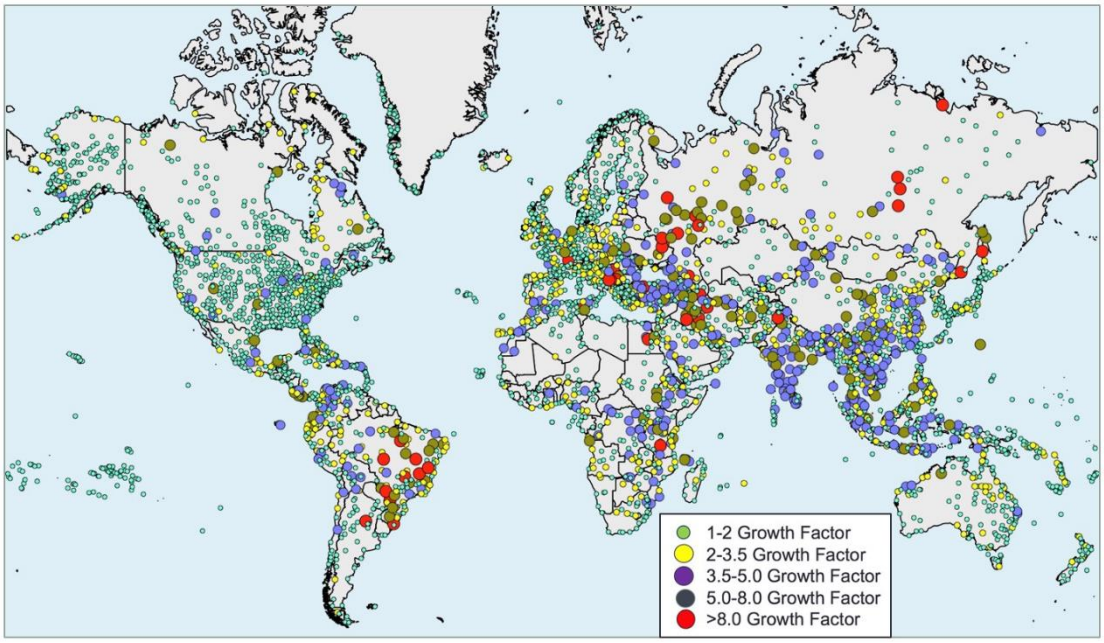


Figure 3-2: Predicted Growth in Air Transportation Demand for Airports in the GDM Model.

Figure 3-2 shows the predicted growth of global air transportation demand by airport predicted in the GDM model. Regions of the world with mature aviation markets (i.e., United States and Europe) are expected to grow less than emerging markets like India and China. The GDM model contains information about the number of flights and aircraft types used to fly those routes for 55,638 routes flown commercially. For example, the Atlanta Hartsfield-Jackson International airport has 256 possible airport destinations. Small airports in isolated regions of the world may have 2 to 3 destinations.

3.2.2 Aircraft Fleet Assignment Module (by Edwin Freire)

The GDM model predicts demand (i.e., number of seats) for more than fifty-five thousand unique origin-destination pairs worldwide. In order to estimate fuel and emission impacts, we developed an aircraft fleet evolution model to allocate flights between origin-destination pairs that satisfy the GDM demand forecast. The aircraft fleet assigned to each origin-destination airport pair was selected using OAG 2015 data.

Analysis of the OAG data shows that 39 unique commercial aircraft types represent 98% of the fleet worldwide. To predict aircraft performance and to estimate fuel and greenhouse emissions we employ, the Eurocontrol BADA 3.13.1 database (Eurocontrol, 2016).

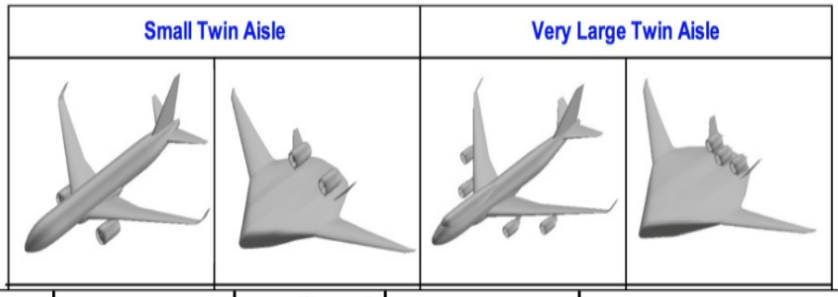
Aircraft fleets are expected to evolve over time. The GDM has the capability to examine the fuel and emission benefits if advanced NASA's N+2 aircraft and new generation N+1 aircraft are adopted in the future. New generation aircraft such as the Boeing 737-8Max, Airbus A320neo and the Bombardier CS100/300 have started to replace older generation commercial aircraft. This trend will continue in the next decade as Boeing and Airbus have received combined orders for more than 8,500 single-aisle aircraft. In order to establish a credible aircraft evolution pattern in GDM, we studied the aircraft utilization trends using OAG data spanning years 2000 to 2015.

Aircraft manufacturers continuously upgrade their existing aircraft products. Newly introduced aircraft are expected to replace older generation aircraft flying today. For example, Airbus certified the Airbus A320 with a new engine option (neo) in 2016. Airbus provides two high by-pass ratio engine options: the GE/Snecma CFM International LEAP-1A and the Pratt and Whitney PW1000G. The new engines provide 12-15% in fuel savings over older generation aircraft. Similarly, Boeing introduced the Boeing 737-8 Max in June of 2017 with the new GE/Snecma CFM International LEAP-1B engine. Airbus has more than 5,054 Airbus A320neo variant orders and Boeing has over 3,500 orders of the 737 Max family. These new generation aircraft are used in the GDM model to replace older generation aircraft. The new generation aircraft are named N+1 aircraft in the analysis. The list of new generation aircraft (N+1) are shown in Table 3-1. The table does not include future new generation aircraft expected from Russia (Irkut MC-21) and China (Comac 919). Nevertheless, the GDM model predicts commercial aviation demand for Russia and China using N+1 aircraft to satisfy the growing demand in emerging economies. The single-aisle commercial aircraft in development in Russia and China are likely to have similar fuel and emissions

performance than the new generation N+1 aircraft from Boeing and Airbus. Both aircraft use new generation CFM-LEAP and PW1000G engines similar to those used in the latest generation of Airbus and Boeing aircraft.

Table 3-1: New Generation Aircraft, Maximum Annual Production Rates and Year of Introduction into Service.

Aircraft Generation	Aircraft Name	Production Rate per Year	Introduction Year
	Airbus 319neo	60	2021
	Airbus 320neo	504	2017
	Airbus 321neo	200	2019
	Airbus 330neo	120	2017
	Airbus 350-1000	48	2019
	Boeing 737 MAX 7	100	2018
	Boeing B737 MAX 8	200	2018
	Boeing B737 MAX 9	100	2018
	Boeing 777-8X	50	2021
	Boeing 777-9X	50	2021
	Bombardier CS100	45	2017
	Bombardier CS300	45	2017
	Embraer E190E2	48	2019
	Embraer 195E2	48	2019
	T+W98	60	2030
	T+W160	240	2030
	T+W216	108	2030
	T+W300	96	2030
	T+W400	36	2030



Vehicle Class	Abbreviation	Number of Passengers	N+2 T+W Nomenclature	Unconventional	Abbreviation
Regional Jet	RJ	98	T+W98	Over-Wing-Nacelle	OWN98
Single Aisle	SA	160	T+W160	Over-Wing-Nacelle	OWN160
Small Twin Aisle	STA	216	T+W216	Hybrid-Wing-Body	HWB216
Large Twin Aisle	LTA	301	T+W301	Hybrid-Wing-Body Mid-Fuselage Nacelle	HWB301 MFN301
Very Large Twin Aisle	VLTA	400	T+W400	Hybrid-Wing-Body	HWB400

Figure 3-3: NASA's N+2 Aircraft (Nikol and Halley, 2016)

Each new generation aircraft (N+1) has a maximum production rate estimated from publicly available aircraft manufacturer data. Table 3-1 shows the production rates for the new generation aircraft. The total numbers of flight hours supplied by the aircraft fleet can be estimated as the product of the annual hours flown by each aircraft type and the number of aircraft available. Table 3-1 provides the year when each new generation aircraft (N+1) is expected to be introduced. Finally Table 3-1, contains maximum annual production rates for five advanced NASA aircraft designs introduced by Nickol and Haller (2016). The five NASA advanced designs (called N+2) are shown in Figure 3-3: NASA's N+2 Aircraft (Nikol and Halley, 2016). N+2 aircraft designs are expected to offer significant fuel consumption reductions compared to the best aircraft flying today. In this analysis, we assume that NASA's N+2 aircraft could be available in the year 2030 (see Table 3-1).

Depending upon the scenario studied, the model assigns N+1 and N+2 aircraft to each route subject to an available flight-hours constraint. Using data from the "MIT Airline Data Project" (http://web.mit.edu/airlinedata/www/default.html) a total of 3,630 hours per year are assigned to each commercial aircraft. The annual aircraft utilization is used to estimate the fleet size needed to satisfy the global demand forecast

in the GDM model. The model uses the calculated travel time between each OD pair to convert from flights to aircraft.

Scenario 1

Scenario 1 is a "do nothing" alternative with no N+1 and N+2 aircraft introduced into the network. The objective of the baseline scenario is to understand a future where the current aircraft fleet mix continues to operate without new aircraft technology that could reduce fuel and emissions. Scenario 1 assumes a 5% detour factor for flights globally. This factor accounts for weather avoidance, route inefficiencies and maneuvering in the terminal area.

Scenario 1.5

Scenario 1.5 assumes that N+1 aircraft are introduced into the commercial aviation network according to the information supplied in Table 3-1. This scenario assumes N+1 aircraft are introduced in routes replacing aircraft with similar seating capacity. This scenario limits the production constraints of N+1 aircraft to those shown in Table 3-1. Priority is given to routes with growth greater than 50% between years 2016 and 2040. Scenario 1.5 assumes a 3% detour factor for flights globally. This assumes a more advanced air traffic control system in the future.

Scenario 2

Scenario 2 assumes that both N+1 and N+2 aircraft are introduced into the commercial aviation network according to the information supplied in Table 3-1. This scenario assumes N+1 and N+2 aircraft are introduced in routes replacing aircraft with similar seating capacity. This scenario limits the introduction of N+1 and N+2 aircraft to production constraints specified in Table 3-1. N+2 aircraft production capacity limits are modest in this scenario (see Table 3-3: Maximum Annual Production Rates for NASA's N+2 Aircraft for Scenarios 2 and 3.). Scenario 2 assumes a 2% detour factor for flights globally. This implies a more advanced Air Traffic Management system in place than today.

Scenario 3

Scenario 3 assumes that both N+1 and N+2 aircraft are introduced into the commercial aviation network according to the information supplied in Table 3-1. This scenario assumes N+1 and N+2 aircraft are introduced in routes replacing aircraft with similar seating capacity. This scenario limits the introduction of N+1 and N+2 aircraft to production constraints specified in Table 3-1. N+2 aircraft production capacity limits are higher than those used in Scenario 2 as shown in Table 3-3. Scenario 3 assumes a high production rate for N+2 aircraft after their introduction in the year 2030. Scenario 3 assumes a 2% detour factor for flights globally. This implies a more advanced Air Traffic Management system in place than today.

Table 3-2: Summary of the Four Scenarios Analyzed in the Global Demand Model.

Parameters	Scenario			
	1	1.5	2	3
Do Nothing	x			
Introduction of N+1 aircraft into the network		x	x	x
Introduction of NASA's N+2 aircraft into the network			x	x
Introduce of new aircraft to routes with growth > 50%		x	x	
Introduce new aircraft in all routes				x
Flight-Hour constraint for N+1 and N+2 aircraft		x	x	x
Add N+1 and N+2 aircraft to routes flown with similar aircraft		x	x	x
Baseline production rate of NASA's N+2 aircraft			x	
High production rate of NASA's N+2 aircraft				x

Table 3-3: Maximum Annual Production Rates for NASA's N+2 Aircraft for Scenarios 2 and 3.

NNASA's N+2 Aircraft	Annual Production Rate Scenario 2	Annual Production Rate Scenario 3
T+W98	60	200
T+W160	240	600
T+W216	108	128
T+W300	96	126
T+W400	36	66

3.2.3 Fuel Consumption and Emissions Module

The fuel consumption and emission modules developed for the GDM model have three components: 1) a mathematical model to estimate aircraft performance for various phases of flight including climb, cruise, descent and taxi-in and taxi-out, 2) a regression analysis for the calculation of taxi times and 3) a procedure to calculate aircraft

emissions

3.2.3.1 Fuel Consumption Modelling

A flight profile is modeled using equations of motion and coefficients from the Base of Aircraft Data (BADA) database (version 3.13.1). Operational conditions considered in the model include a flight detour factor and cruise altitude assignment. The required model inputs are Origin-Destination (OD) pair information, the aircraft type and the annual departures of the aircraft type at the corresponding origin airport. The model treats each unique OD pair and aircraft type combination as one flight instance. Climb, cruise and descent profiles are estimated for each flight using numerical integration techniques built into Matlab. The BADA model provides a set of ASCII files containing performance and operating procedure coefficients for individual aircraft types. The coefficients in the BADA model are used to calculate thrust, drag and fuel flow. Flight speeds for cruise, climb and descent are also contained in the BADA model (Eurocontrol, 2015). In this analysis we use nominal cruise speeds contained in the BADA database for each aircraft.

In order to calculate the instantaneous parameters of altitude, distance and weight in each iteration, an Ordinary Differential Equation (ODE) solver is required. A customized Runge Kutta 4^{th} order (RK4) ODE solver is used to perform the calculations. The solver takes inputs of derivatives of aircraft altitude, distance traveled, aircraft weight and time span, and outputs the instantaneous values of those parameters. (Zou Z. and Trani A.A., 2012).

The BADA mathematical model does not provide aircraft performance that varies with geographical location, airline and specific airspace procedures and policies. In this analysis, we assume all flights are subject to the same Air Traffic Control rules. The cruise flight levels assigned to a flight follow hemispherical rules. The flight profile generator assumes the aircraft take-off mass is a random variable and is a function of the stage length for the flight. This way flights are not assigned the same cruise flight level even when flying the same distance. Moreover, in the flight profile generation,

each flight is also assigned a cruise altitude stochastically based on real cruise altitude data collected from the FAA Traffic Flow Management System (TFMS) data.

Figure 1-1 provides an overview of the steps followed in the fuel consumption and emission model. Trip distribution and fleet evolution outputs produce a number of flights across all OD airport pairs worldwide. Each route can be flown by multiple aircraft. The fuel consumption model estimates the fuel used for each distinct route-aircraft combination. In a typical scenario with 55,638 routes, the fuel consumption model estimates more than 140,000 individual flights to calculate the global fuel consumption.

The following are important assumptions and limitations made in fuel consumption model:

- Freighter aircraft flights are not included in the analysis.
- Military and non-scheduled air traffic are not included in the analysis.
- Delays and cancellations are not modeled.
- Each simulated trajectory follows a flight profile consisting of a continuous climb out to cruise altitude, followed by a cruise, followed by a continuous descent. Each flight is penalized with a detour factor to make the fuel calculations realistic. This simulates flight deviations from a Great Circle route.
- No wind optimal flight trajectories are simulated in this analysis. The GDM model has the capability to create wind optimal flight trajectories. However, given the large number of origin-destination pairs involved in the analysis we did not employed this feature of the model. Instead, different detour factors were used for each scenario to account for more advanced air traffic control systems in the future.

- Step climb procedures were allowed in the simulations. Step climbs are allowed when the mass of the aircraft allows a minimum climb rate of 500 ft/minute to the next flight level. The selected climb rate threshold requires 80-90 minutes of fuel burn before a wide-body aircraft climbs one flight level.

- All airports were assumed to be at sea level conditions.

- No direct modeling of terminal area maneuvering. Each flight is penalized with a detour factor to make the fuel calculations realistic.

- Ground movement fuel calculations employ regression models of taxi-in and taxi-out as a function of airport operations. The ground taxi data has been obtained from the FAA Airport System Performance Metrics database (ASPM) (Zou Z. and Trani A.A., 2012).

3.2.3.1.1 Mapping of National Aeronautics and Space Administration (NASA) N+2 Advanced Vehicles

In 2009, NASA created the Environmentally Responsible Aviation (ERA) project. This project focused on new vehicle concepts and enabling technologies to reduce fuel burn, noise and Landing Takeoff (LTO) NOx emissions. The “N+2” aircraft technology was defined as to contain technologies with Technology Readiness Level (TRL) of 4-6 by 2020. (Nickol and Haller, 2016). There are 5 such tube and wing (T+W) NASA N+2 aircraft models considered in this analysis. Table 3-4 summarizes the NASA N+2 aircraft models compared with the best-in class baseline aircraft and the corresponding block fuel savings per flight. The same block fuel savings are used in the GDM fuel and emissions model.

Table 3-4: NASA N+2 Aircraft with Block Fuel Savings Compared with the Best-in Class Aircraft Today.

NASA N+2 Vehicles	Block Fuel Savings (%)	Best-in Class Aircraft Today
T+W98	46.0	Embraer Regional Jet190
T+W160	42.8	Boeing 737-800
T+W216	44.5	Boeing 767-200ER
T+W301	39.1	Boeing 777-200LR
T+W400	46.2	Boeing 747-400

The fuel consumption and emissions model incorporates N+1 aircraft that are to be introduced in future years. Table 3-5 shows the N+1 aircraft compared with baseline aircraft flying today. The table provides the corresponding fuel burn savings provided by each aircraft manufacturer. For example, the Airbus 330neo with new generation Trent 7000 engines is 11-14% more fuel efficient than a standard Airbus A330-300.

Table 3-5: New Generation Aircraft with Block Fuel Savings Compared with the Base Aircraft.

Aircraft	Fuel Savings Stated by the Manufacturers (%)	Baseline Aircraft
Airbus 319neo	20	Airbus 319
Airbus 320neo	20	Airbus 320ceo
Airbus 321neo	20	Airbus 321
Airbus 330neo	11-14	Airbus 330
Airbus 350-1000	25	Boeing 777-300ER
Boeing 737 Max Series (737 MAX 7,8,9)	16	Airbus 320
Boeing 777X series	20	Boeing 777-300ER
Bombardier C series (CS100 and CS300)	20	Airbus 320ceo
Embraer 190 – E2	16	Embraer 190
Embraer 195 – E2	24	Embraer 195

3.2.3.2 Ground Fuel Burn Estimation

Aircraft taxiing on the surface can burn modest to large amount of fuel. This is because jet engines are designed for efficient power generation at high speeds and high altitudes, but are far less efficient at the surface-level ground operations.

The taxi-out and taxi-in operations are most important at large hub airports. The major concern being the emission caused by these ground operations. In this model, the ground fuel burn and emissions are calculated by estimating taxi-out and taxi-in times at specific airports i.e. the origin and destination (OD matrix) obtained from the global demand model. 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, 2015). The actual taxi time includes delays. As a result, the fuel consumption calculated based on taxi times includes the fuel burn for ground delays. (Zou Z. and

Trani A.A., 2012). The taxi-in and taxi-out travel times are calculated as a function of flight operations and shown in Figure 3-4 and 3-5 using a linear trend.

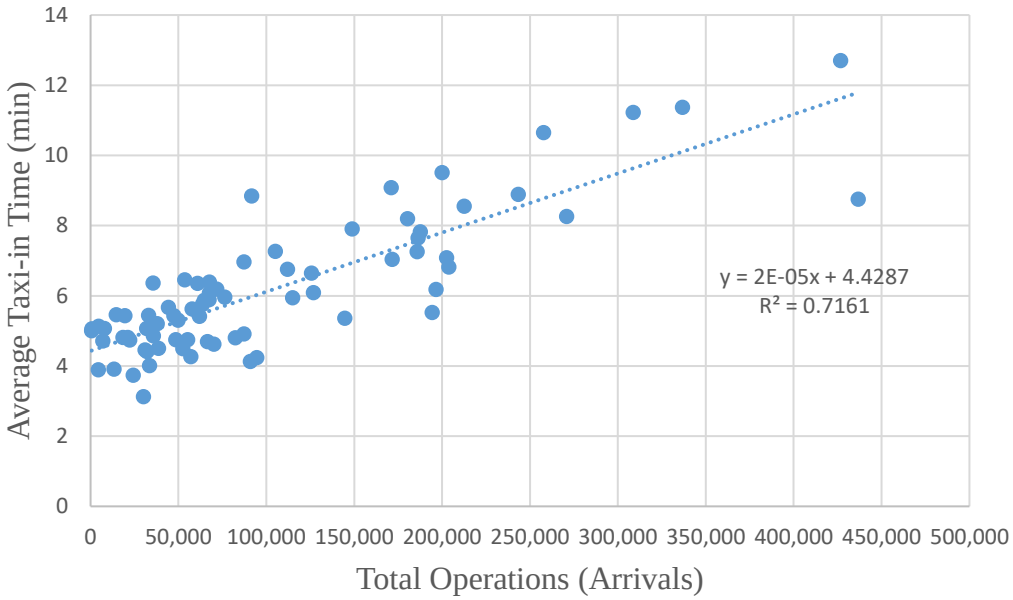


Figure 3-4: Linear Trend between Taxi-In Times and the Annual Number of Arrivals at ASPM 77 Airports for the Year 2015.

Fuel burn data for ground operations was obtained from FAA Emission and Dispersion System (EDMS). The fuel burn rates at idle conditions were considered in the development of the global commercial ground fuel consumption. The ground fuel consumption for the NASA N+2 advanced vehicles is calculated with reference to the baseline best-in class synonym aircraft shown in Table 3-4.

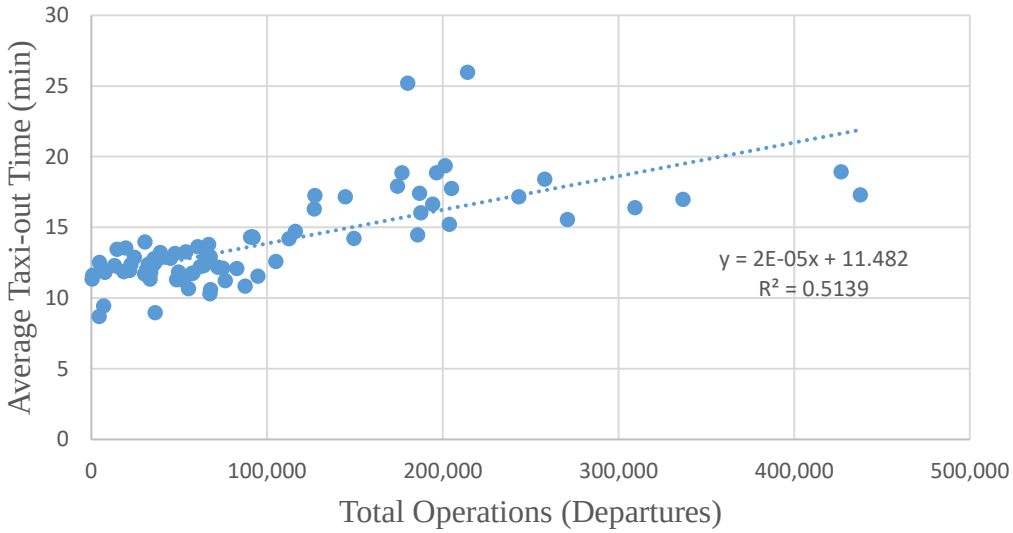


Figure 3-5: Linear Trend between Taxi-Out Times and the Annual Number of Arrivals at ASPM 77 Airports for the Year 2015.

3.2.3.3 Emissions Model

Aircraft engines emit particles such as carbon dioxide (CO_2), water vapor, hydrocarbons (HCs), carbon monoxide (CO), nitrogen oxides (NOx), sulfur oxides (SOx) and black carbon. The GDM model estimates CO, HC, NOx and SOx emissions in the landing and take-off (LTO) cycle. The LTO cycle covers activities of a flight near the airport and up to 3,000 feet above ground level. Equation (2) is used to estimate emissions in the GDM model.

$$Emission\ \ (g) = Fuel\ \ Burn\ \ (kg) \times Emission\ \ Factor\ \ (g/kg\ \ fuel\ \ burn) \quad (3)$$

The emission factors in Equation (3) are obtained from the FAA Emissions and Dispersion Modeling System (EDMS) database (version 5.2). EDMS contains emission factors for 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 5.2 uses the same

BADA data for aircraft performance modeling used to estimate the flight profiles in the GDM model. Table 3-6 shows the emission factors for commonly used commercial aircraft types extracted from the EDMS database The first-order estimation of CO_2 emissions adopts an emission factor of 3.157 (kg/kg fuel burn) which relates the mass of CO_2 produced by stoichiometric combustion of a known amount of fuel.

Table 3-6: Sample of Emission Factors for Some Aircraft Types Extracted from EDMS.

Aircraft	Takeoff_Co (G/K)	Takeoff_Hc (G/K)	Takeoff_Nox (G/Kg)	Takeoff_Sox (G/Kg)	Approach_Co (G/Kg)	Approach_Hc (G/Kg)	Approach_Nox (G/Kg)	Approach_Sox (G/Kg)
Airbus 310	1	0.3	29.6	1	2.8	0.45	10.8	1
Airbus 319	0.57	0.041	24.5	1	2.6	0.062	8.7	1
Airbus 320	0.9	0.23	24.6	1	2.5	0.4	8	1
Airbus 321	0.8	0.1	35.1	1	1.9	0.5	10.9	1
Airbus 330-200	0.72	0.03	42.39	1	1.75	0.15	14.66	1
Airbus 330-300	0.05	0.04	28.72	1	1.85	0.11	12.66	1
Airbus 340-600	0.02	0[2]	44.91	1	0.46	0[1]	11.78	1
Airbus 380-800	0.02	0[1]	44.91	1	0.46	0[1]	11.78	1
Avions De Transport Régional 42-500	0.71	0[1]	9.7	1	4.8	0[1]	6.2	1
Avions De Transport Régional 72-500'	0.71	0[1]	9.7	1	4.8	0[1]	6.2	1
Boeing 717-200	0.78	0[1]	23.97	1	3.76	0.01	11.19	1
Boeing 737-300	0.9	0.04	18.5	1	3.5	0.1	8.4	1
Boeing 737-500	0.9	0.03	20.7	1	3.1	0.07	9.1	1
Boeing 737-700	0.4	0.1	25.3	1	2.2	0.1	10.1	1
Boeing 737-800	0.2	0.1	28.8	1	1.6	0.1	10.8	1
Boeing 737-900	0.2	0.1	30.9	1	1.4	0.1	11	1
Boeing 747-400	0.44	0.06	28.1	1	2	0.13	11.6	1
Boeing 747-800	0.225	0.02	27.5	1.292	3.47	0.085	8.58	1.292

[2] Hydrogen Carbon (HC) Emission rates for aircraft Airbus 340-600, Airbus 380-800, Avions de Transport Régional 42-500, Avions de Transport Régional 72-500, Bombardier Havilland Dash 8-300, Bombardier Havilland Dash 8 Q-400 and McDonnell Douglas MD 82 were absent in the EDMS data.

Boeing 757-200	0.33	0.02	29.41	1	1.95	0.11	9.77	1
Boeing 767-200	1	0.3	29.6	1	2.8	0.45	10.8	1
Boeing 767-300	0.37	0.1	32.8	1	1.78	0.14	12	1
Boeing 767-400	0.05	0.05	27.38	1	1.93	0.11	12.63	1
Boeing 777-200	0.19	0.03	61	1	0.44	0.06	13.19	1
Boeing 777-300	0.19	0.03	61	1	0.44	0.06	13.19	1
Boeing 777-200l	0.19	0.03	61	1	0.44	0.06	13.19	1
Boeing 777-300w	0.19	0.03	61	1	0.44	0.06	13.19	1
Boeing 787-800	0.49	0.0001	46.7	1.292	0.69	0.0001	18.36	1.292
Beechcraft 99	0.9	0	7.5	1	3.5	0.2	5.6	1
Cessna 750 Citation X	0.97	0.29	16.92	1	4.5	0.83	6.63	1
Canadair Challenger 60	0	0.06	11.28	1	1.88	0.13	6.63	1
Bombardier Regional Jet Crj-200	0	0.06	11.61	1	1.9	0.13	6.86	1
Bombardier Regional Jet Crj-900	0.64	0.02	14.69	1	4.24	0.06	10.75	1
Bombardier Havilland Dash 8-300	1.9	0[1]	17.7	1	3.6	0[1]	10.1	1
Bombardier Havilland Dash 8 Q-400	2.1	0[1]	16.8	1	3.5	0[1]	9.4	1
Embraer E135	0.84	0.27	18.82	1	4.04	0.76	6.99	1
Embraer E145	0.75	0.25	20.54	1	3.28	0.64	7.79	1
Embraer E170	0.59	0.02	13.6	1	4.52	0.07	10.29	1
Embraer E190	0.71	0.02	15.81	1	4.05	0.06	11.06	1
Mcdonnell Douglas Md 11	0.37	0.1	32.8	1	1.78	0.14	12	1
Mcdonnell Douglas Md 82	0.42	0[1]	16.49	1	3.79	0[1]	7.65	1
Piper Pa31 Navajo	1442	12.4	0.36	0.11	1260	13.4	1.39	0.11
Socata Tbm-700	5.1	1.75	7.98	0.54	34.77	22.69	4.64	0.54

3.2.3 TPADS Input Tables

The results of the GDM model are compatible with NASA's demand and emission input tables of the Technology Portfolio Assessment and Decision Support (TPADS) tool. The TPADS input tables are aggregated by aircraft class, year of operation, and associated fuel and emissions outputs. Figure 3-6 compares the trends between five aircraft including the Boeing 737-800, Boeing 777-200, Bombardier CRJ-900, Embraer

E190 and Airbus A321 for years 2016-2040 for Scenario 3. Because of the large number of flights worldwide operated by Boeing 737-800 aircraft, the Boeing 737-800 contributes the most in terms fuel burn, emissions and air traffic for the years simulated. The fuel burn results of the TPADS input are shown in Figure 3-7.

The GDM model results for fuel and emissions were combined into the Technology Portfolio Assessment and Decision Support (TPADS) tool emission tables and provided to NASA Langley Research Center.

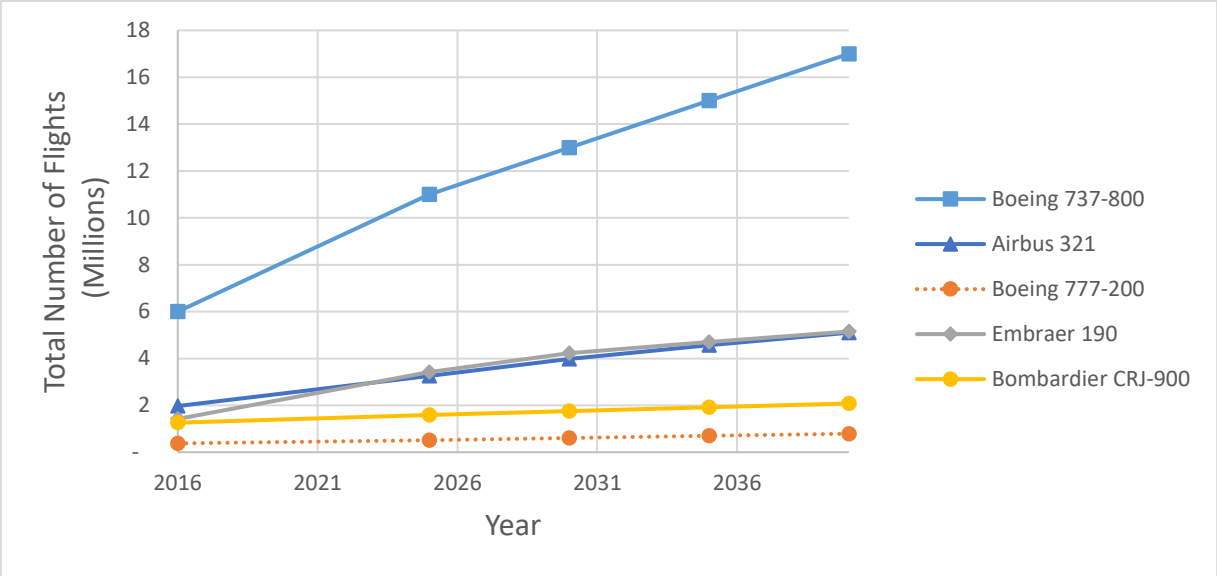


Figure 3-6: Annual Flights for Selected Aircraft in the Demand and Emissions Input Tables of the TPADS Tool.

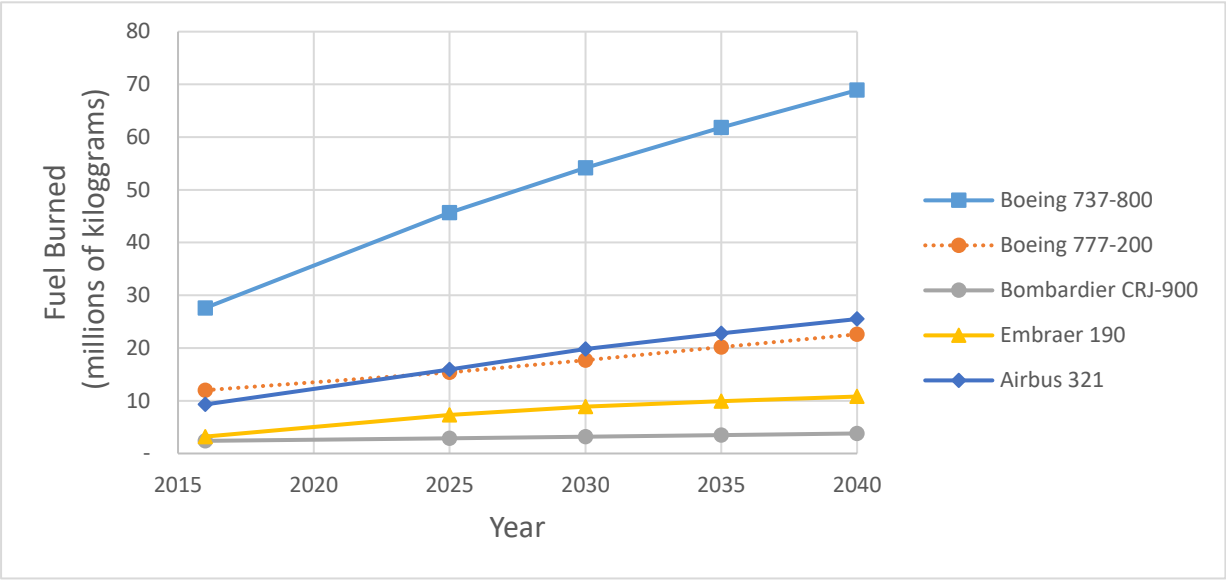


Figure 3-7: Annual Fuel Burned for Selected Aircraft in the Demand and Emissions Input Tables of the TPADS

3.3 Results

3.3.1 Fuel Consumption Results

Annual worldwide fuel burn results were calculated for each scenario described in Table 3-2. Table 3-7 summarizes the annual fuel consumption estimates for all four scenarios investigated.

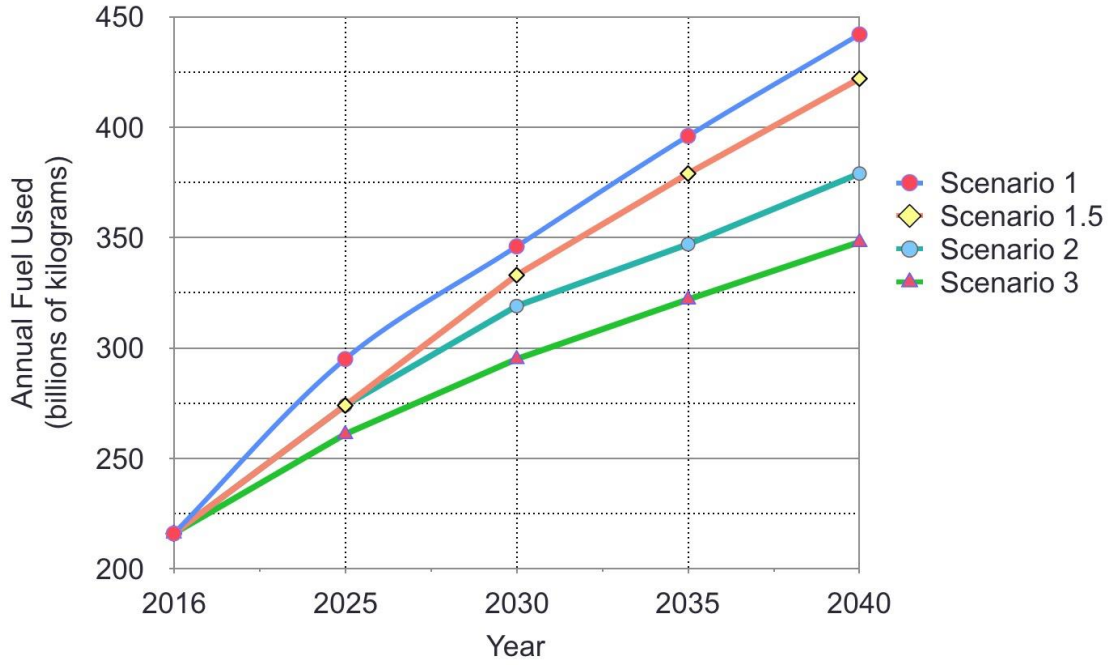


Figure 3-8: Annual Global Fuel Consumption for All Scenarios Modeled.

Figure 3-8 shows the same information in graphical form. It is clear that Scenario 3 – using high adoption rate of NASA N+2 aircraft has the most the positive impact on the environment. Figure 3-9 shows the annual fuel use by commercial aviation for seven regions of the world under assumptions made for Scenario 3. The trends indicate that Asia is the region that is expected to have more flights and hence more fuel used in the year 2040. In Scenario 3, the annual fuel used in Asia is expected to increase by 86% from 63 billion kilograms to 117.5 billion kilograms.

Annual fuel use in North America is expected to increase by 20% from 56 to 67 billion kilograms under Scenario 3. The difference in fuel use can be explained by the differences in economic growth between the two regions (i.e., faster growth in Asian Economies); the distinct population growth of the two regions (i.e., faster population growth in Asia than in North America); and the distinct fleet mix composition (i.e., larger aircraft in Asia than in North America).

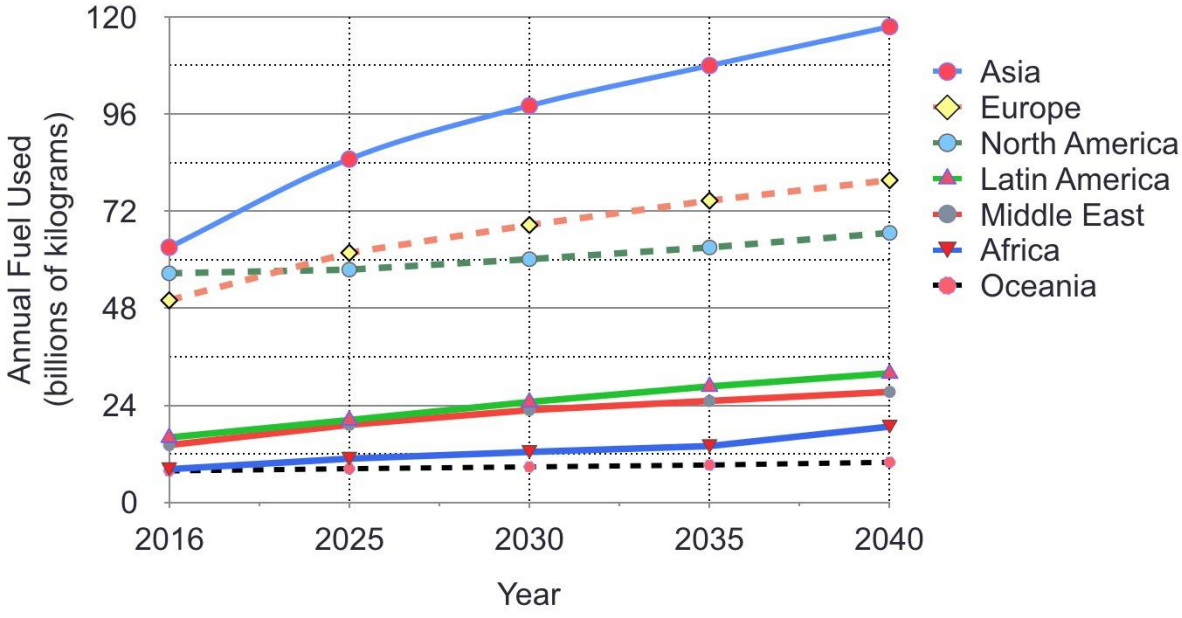


Figure 3-9: Annual Fuel Consumption by World Region for Scenario 3.

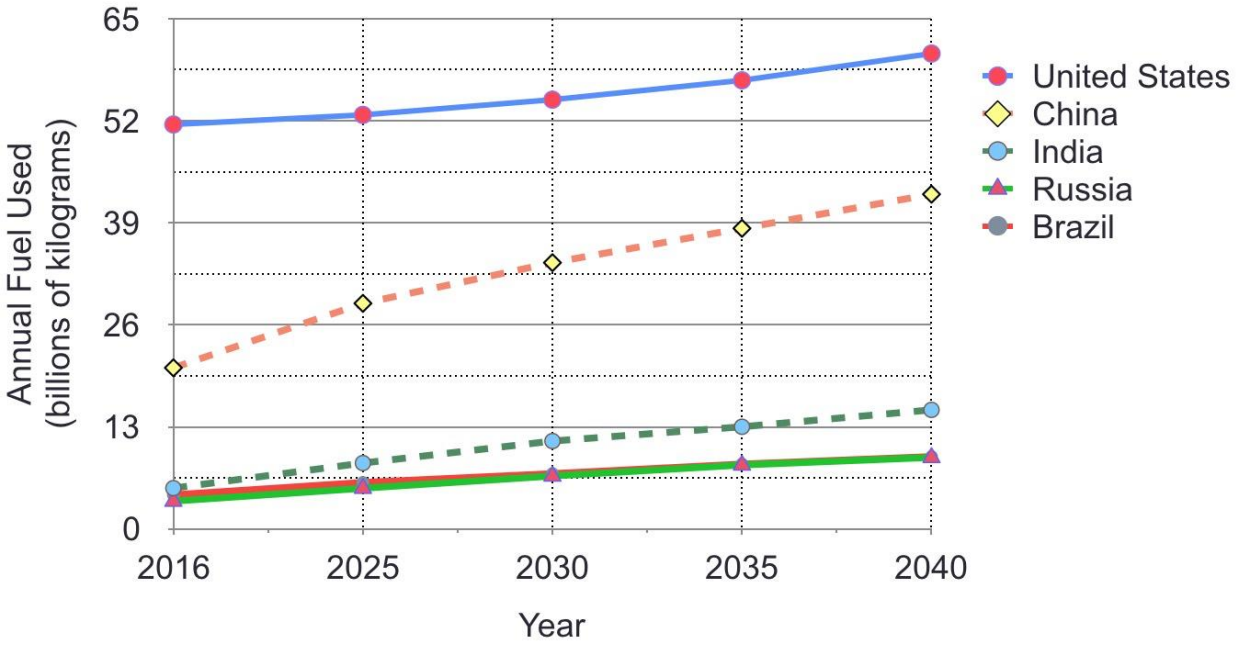


Figure 3-10: Annual Fuel Consumption for Selected Countries for Scenario 3.

Figure 3-10 shows the annual fuel use for selected countries. The graphic compares the annual fuel use between US and the so-called BRIC countries. China is the country with the largest change in annual fuel use for commercial aviation between years 2016 and 2040. Even with the introduction of NASA N+2 aircraft in 2030, China is expected to use 113% more aviation fuel in 2040 than in 2016 due to the large increase in aviation operations in the country.

Figure 3-11 shows the annual fuel use in year 2040 attributed to every 1x1 degree tile on Earth for Scenario 3. The map shows regions of the world (including airspace corridors) with high intensity fuel use. For example, the map highlights China, Southeast Asia and the Middle East as three regions where high concentration of fuel use is expected in 2040.

Other salient worldwide fuel impacts are:

1. Introducing NASA N+2 aircraft in modest numbers (Scenario 2) produces a reduction in annual fuel consumption in the year 2040 of 14.3% compared to Scenario 1 and 10.2% compared to Scenario 1.5.

2. Introducing NASA N+2 aircraft in large numbers (Scenario 3) produces a reduction in annual fuel consumption in the year 2040 of 21.3% compared to Scenario 1 and 17.5% compared to Scenario 1.5.

3. The regions of the world that could benefit the most from the adoption of NASA N+2 aircraft are Asia and the Middle East. The number of flights in Asia is expected to grow by 170% in the next 24 years. A 200% increase in the number of flights is expected in Middle East. By comparison, the number of flights in North America are expected to grow by 54% in the same period.

A reduction of 17.5% between Scenario 1.5 and Scenario 3 shows the future challenge to reduce fuel consumption for a large and established aircraft population (around 20,000 aircraft today). A 17.5% reduction in fuel annually represents saving 74 billion kilograms of aviation fuel. According to the US Environmental Protection Agency, 74 billion kilograms produces the same Greenhouse gas emissions as 45.5 million passenger cars driven in one year. Similarly, 74 billion kilograms of fuel saved annually is equivalent to the CO_2 emissions of 22.7 million homes in the United States or 63 coal-fired power-plants per year.

Table 3-7: Annual Worldwide Fuel Consumption (In billion kg) Results for All Scenarios Modeled.

Year	Scenario 1	Scenario 1.5	Scenario 2	Scenario 3
2016	216	216	216	216
2025	295	278	278	261
2030	346	333	319	295
2035	396	379	347	322
2040	442	422	379	348

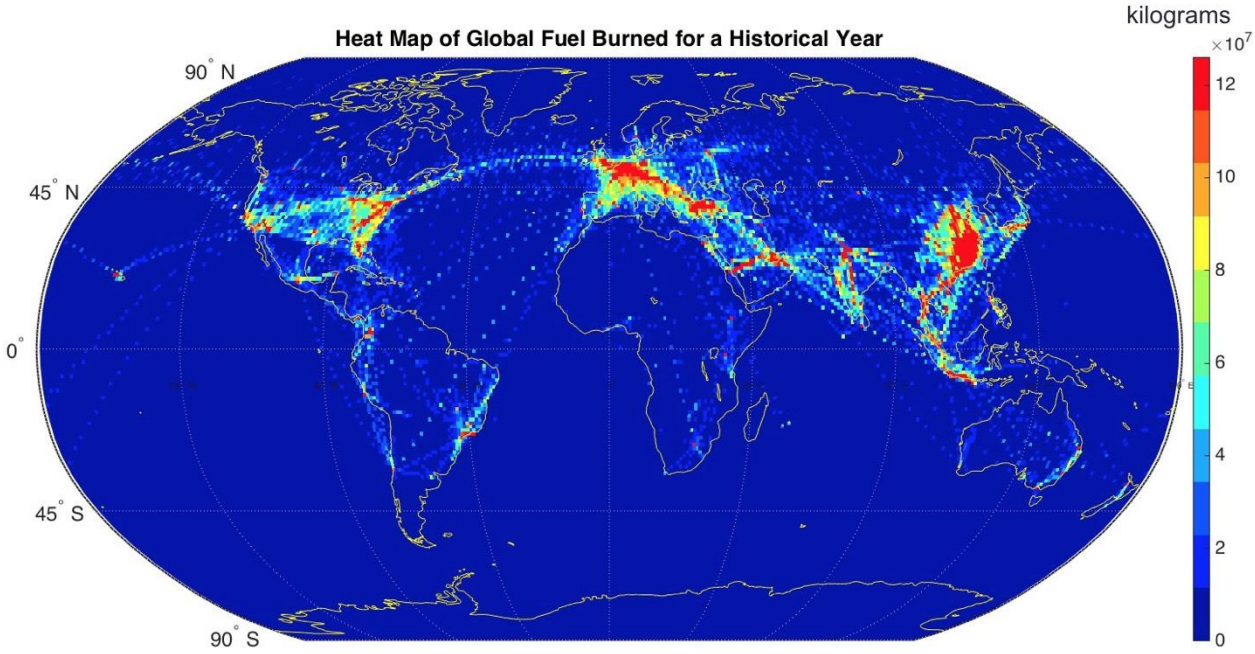


Figure 3-11: Annual Fuel Consumption Calculated at each 1 Degree by 1 Degree tile for Scenario 3.

2.4.2 Emission Results

Table 3-8 contains LTO emission results obtained for each scenario described in Table 3-2. The table compares the LTO emissions obtained in Scenarios 2 and 3 versus Scenario 1.5 (the baseline scenario). According to Table 3-8 Scenario 3 could reduce LTO emissions of HC, NO_x and SO_x by more than 10%. The same information is shown graphically in Figure 3-12.

Table 3-8: Potential Emission Reductions (in Percent) for Scenarios 2 and 3 compared to Scenario 1.5.

Year	Scenario 2				Scenario 3			
	CO	HC	NOx	SOx	CO	HC	NOx	SOx
2030	0.80	5.3	0.97	1.16	0.35	9.55	7.37	2.82
2035	3.45	4.17	6.51	5.76	5.66	9.90	9.82	7.35
2040	4.54	6.62	8.48	7.16	9.03	12.29	11.25	10.44

Figure 3-13 shows the global annual CO_2 emissions for all four scenarios

investigated in the analysis. Introducing NASA N+2 aircraft in large numbers (Scenario 3), could reduce the global CO_2 emissions due to aviation activity in 2040 by 17.6% compared to Scenario 1.5 (i.e., introducing just N+1 aircraft). A total of 234 billion kilograms of CO_2 could be saved annually if NASA N+2 aircraft operate in large numbers in the year 2040.

Figure 3-14 shows the CO_2 emission results by region of the world for Scenario 3. Figure 3-15 shows the annual CO_2 emissions for selected countries. The graphic compares the annual CO_2 emissions produced by the US and the BRIC countries. China is the country with the largest change in annual fuel use for commercial aviation between years 2016 and 2040. Even with the introduction of NASA N+2 aircraft in 2030, China is expected to produce more than 113% in CO_2 emissions in the 2040 than today.

Figure 3-16 shows the annual CO_2 emissions in the year 2040 attributed to every 1x1 degree tile on Earth for Scenario 3. The map shows regions of the world (including airspace corridors) with high intensity of CO_2 emissions. For example, the map highlights China, Southeast Asia and the Middle East as regions where high concentration of CO_2 emissions are expected in 2040.

Other salient worldwide CO_2 impacts are:

1. Introducing NASA N+2 aircraft in modest numbers (Scenario 2) produces a reduction in annual CO_2 emissions in the year 2040 of 14.3% compared to Scenario 1 and 10.2% compared to Scenario 1.5.

2. Introducing NASA N+2 aircraft in large numbers (Scenario 3) produces a reduction in annual CO_2 emissions in the year 2040 of 21.3% compared to Scenario 1 and 17.5% compared to Scenario 1.5.

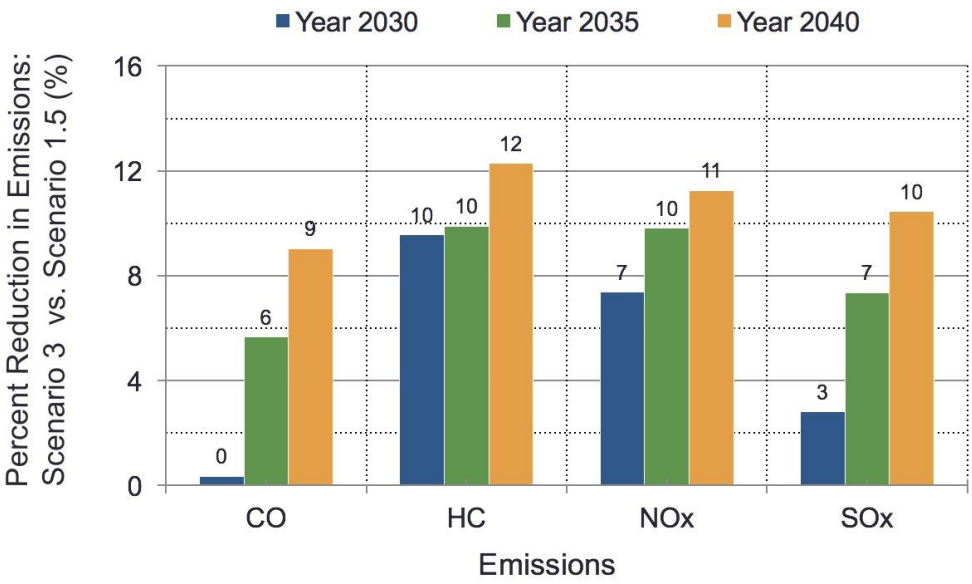


Figure 3-12: Annual Reductions in LTO Cycle Emissions Between Scenarios 1.5 and 3.

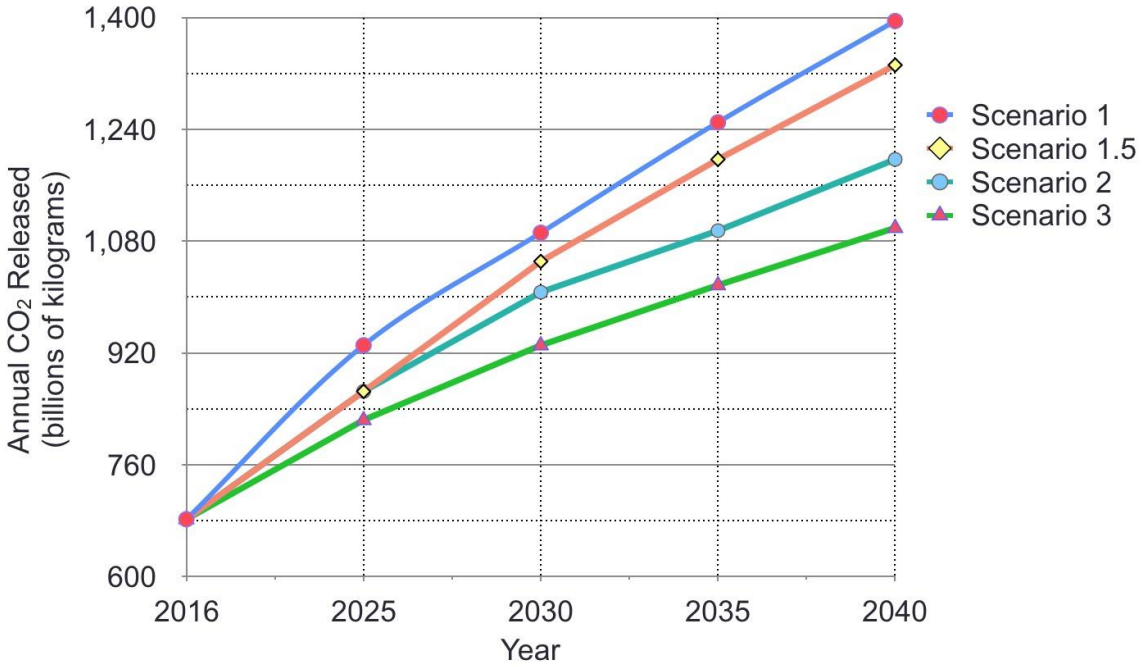


Figure 3-13: Annual CO_2 Produced by Commercial Aviation for All Scenarios Modeled.

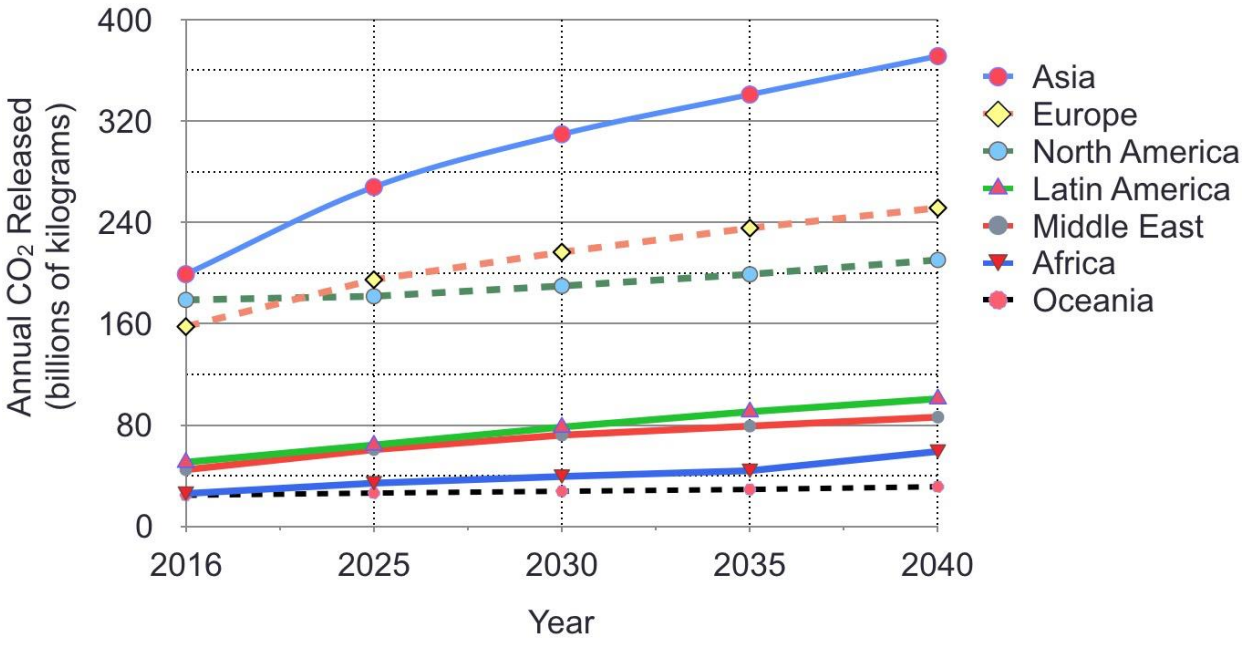


Figure 3-14: Annual CO_2 Produced by World Region and Scenario 3.

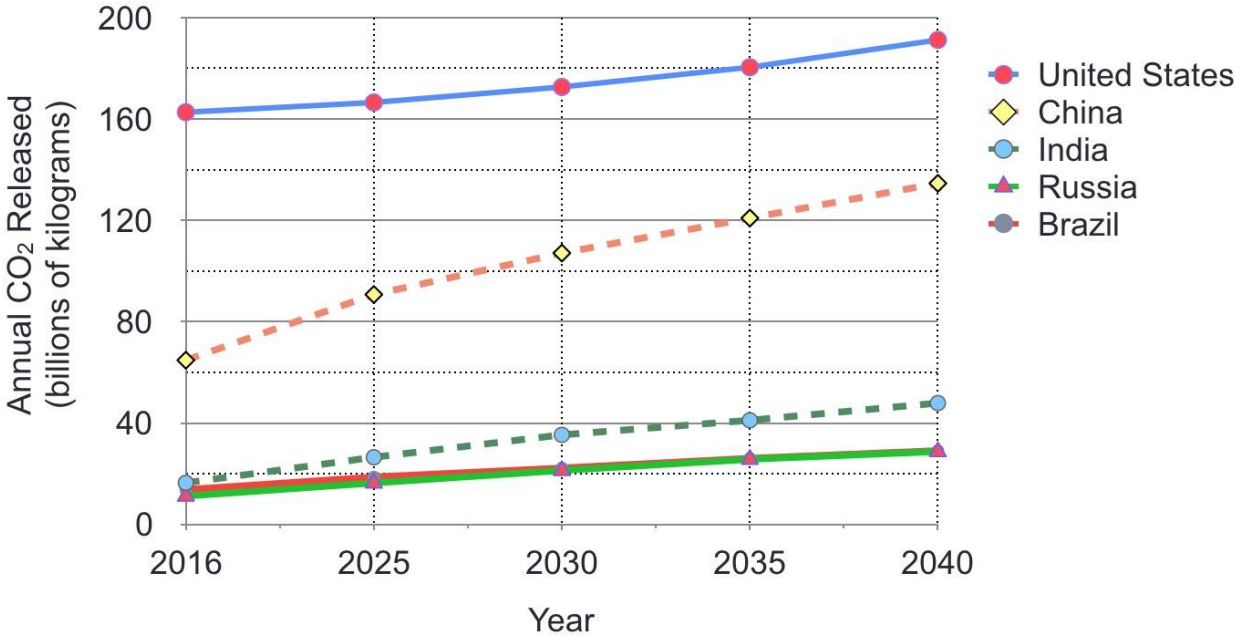


Figure 3-15: Annual CO_2 Produced for Selected Countries and Scenario 3.

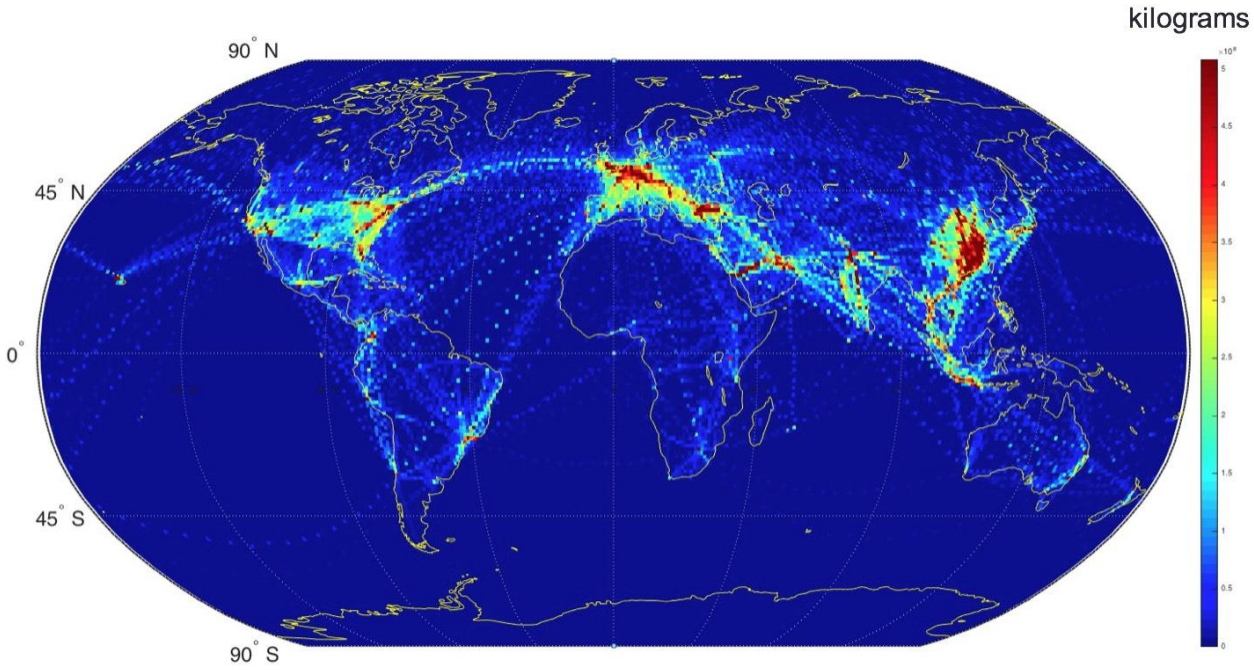


Figure 3-16: Annual CO_2 Calculated at each 1 Degree by 1 Degree tile for Scenario 3.

CHAPTER 4 VALIDATION

4.1 Basic Aircraft Performance

Many inaccuracies were observed in the OAG data obtained for years 1996-2005. We observed that certain OD pairs were unrealistic in terms of the distance between them: either too short or too long. Also incorrect flights were assigned to OD pairs where the distance between the OD pair is greater than the maximum range of the aircraft. All such errors in data were taken care of and eliminated in the analysis. This section presents a simple validation of the aircraft performance in the Global Demand Model. Figure 4-1 shows the total flight times (in minutes) of all the flights for 55638 OD pairs simulated in the study. The longest flight simulated is between Dallas and Sydney flown by Airbus A380-800 which is around 7451nm.The flight times reported by Qantas airlines for flight QF8 is around 1010 to 1030 minutes and that obtained from the simulations is 1003 minutes.

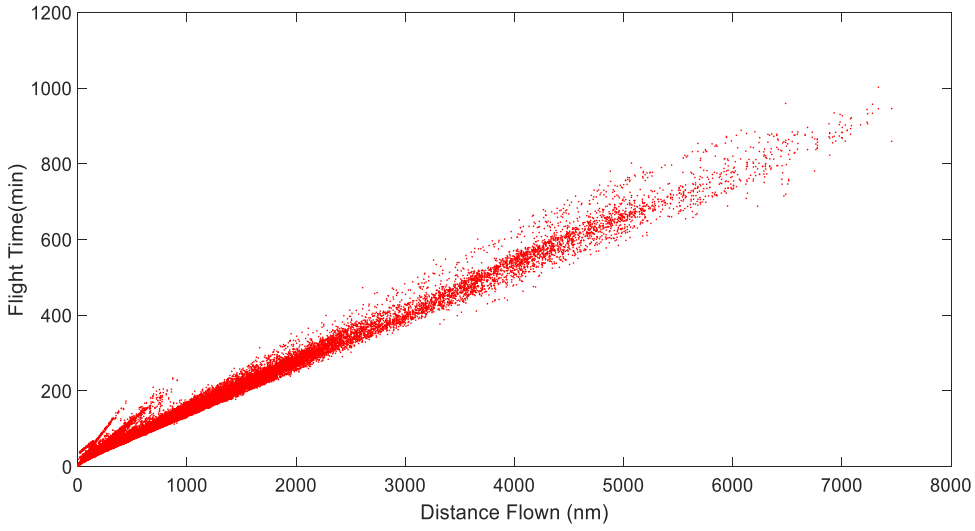


Figure 4-1 Flight Times against Distance Flown for all the Flights Simulated in the GDM

4.2 Comparison of GDM model Results with ICAO Projections

The GDM fuel consumption model results follow monotonically increasing trends even when NASA N+2 aircraft are deployed in large numbers in the world aircraft fleet.

This is the result of growing air transportation demand that outpaces the improvements in fuel consumption obtained by the introduction of more advanced and fuel efficient aircraft. Nevertheless, introduction of N+2 aircraft (Scenario 3) produces tangible benefits compared to Scenario 1.5 where only N+1 aircraft replace older aircraft types (a 17.5% reduction in fuel use in the year 2040).

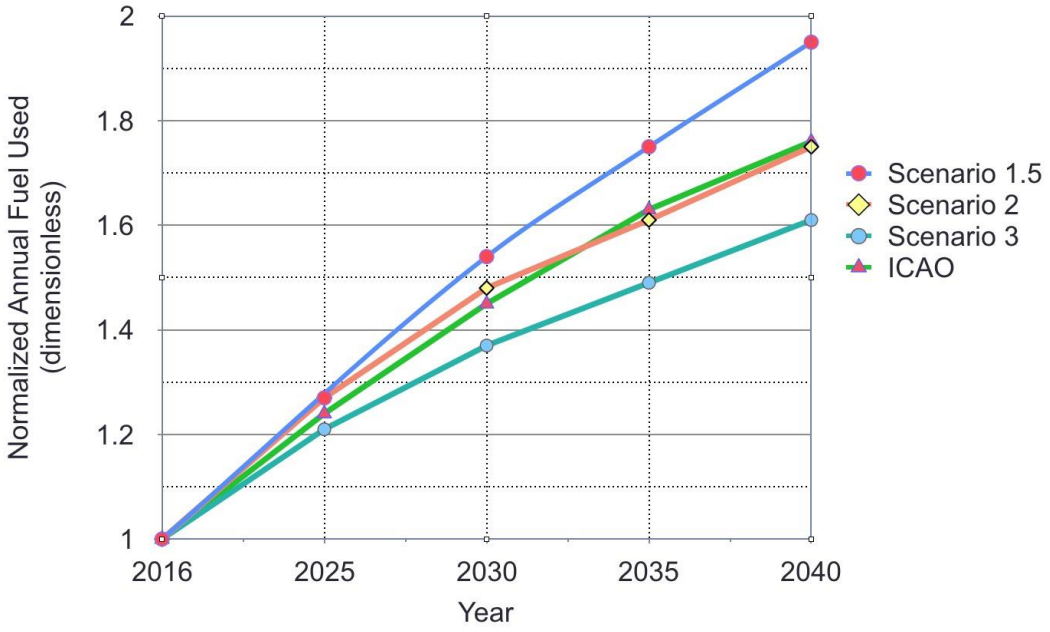


Figure 4-2: Comparison of Normalized Aircraft Fuel Burn Trends for Scenarios 1.5, 2, 3 and ICAO's Goal of 2% Global Annual Fuel Efficiency Improvement

Figure 4-2 compares the GDM fuel use model results with ICAO aspirational fuel efficiency data. The ICAO model assumes a 2% reduction in fuel used every year (ICAO, 2013). The results are normalized to make the comparisons objective since we do not know the assumptions of the commercial aircraft fleet included in the ICAO projections. The figure shows that ICAO's 2% aspirational reduction in fuel consumption per year produces results similar to those of Scenario 2 (i.e., introducing NASA N+2 aircraft in modest quantities starting in the year 2030). Scenario 3 achieves higher fuel use savings compared to ICAO's projection as shown in Figure 4-2.

4.3 Performance of NASA's Advanced Vehicles against their Baseline Aircraft

Figure 4-3, illustrates the simulation results for NASA T+W 98 and for the Embraer E190 flying the same OD pair and Figure 4-4 shows the same between NASA T+W400 and the Boeing 747-400. The results are in accordance with the block fuel savings described in Table 3-4.

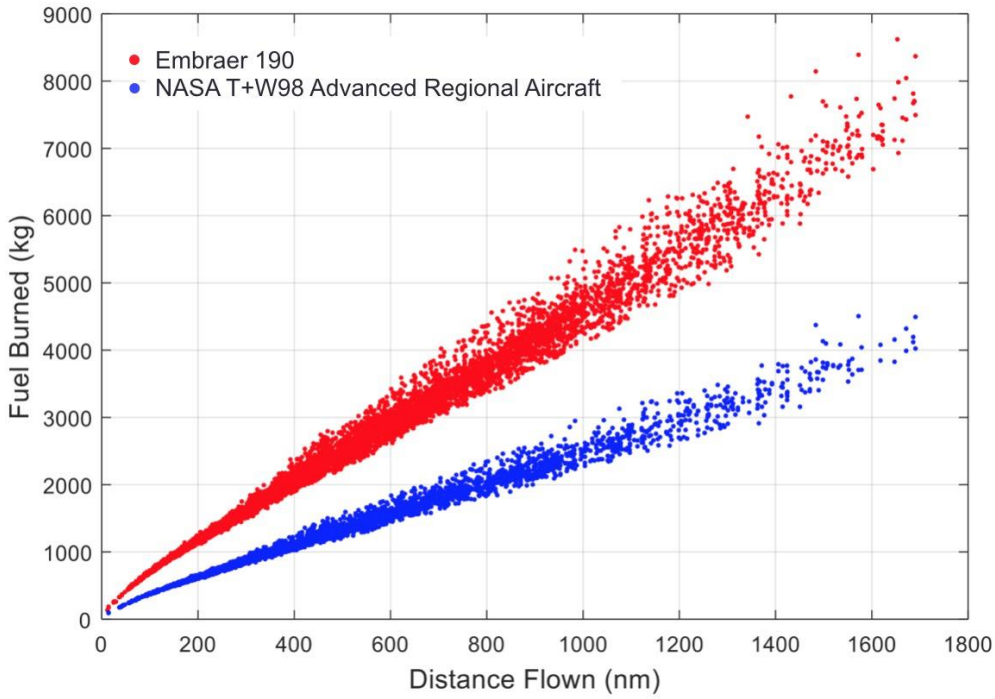


Figure 4-3: Comparison of Fuel Burn Vs Range between T+W98 and Embraer 190.

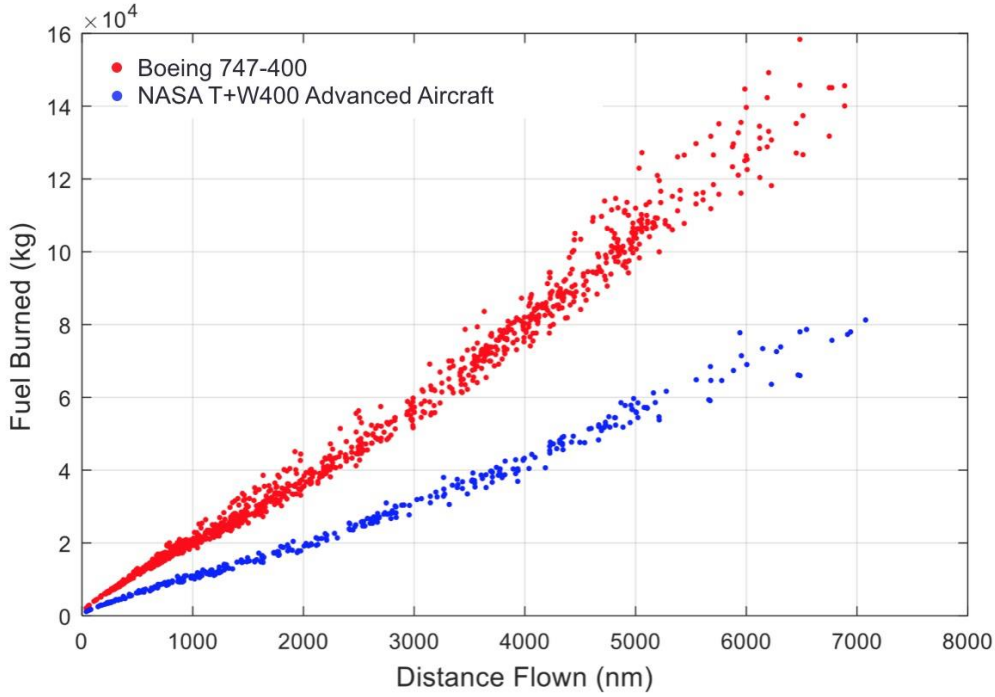


Figure 4-4: Comparison of Fuel Burn vs Range between T+W400 and Boeing 747-400.

CHAPTER 5 CONCLUSIONS

This thesis developed and integrated important improvements to the Global Demand Model. The improvements in the trip distribution model and fleet assignment model has made the GDM more realistic and accurate. The addition of the fuel burn and emission module has added on to the capability of the model. The performance results of the model are comparable with the ICAO's projections.

The model results can be summarized as follows:

Fuel Consumption Projections

- Introducing NASA N+2 aircraft in modest numbers (Scenario 2) produces a reduction in annual fuel consumption in the year 2040 of 14.3% compared to Scenario 1 and 10.2% compared to Scenario 1.5.

- Introducing NASA N+2 aircraft in large numbers (Scenario 3) produces a reduction in annual fuel consumption in the year 2040 of 21.3% compared to Scenario 1 and 17.5% compared to Scenario 1.5.

- The regions of the world that could benefit the most from the adoption of NASA N+2 aircraft are regions with faster growth such as Asia and the Middle East.

- A reduction of 17.5% in the fuel used between Scenario 1.5 and Scenario 3 shows the challenge ahead to reduce fuel consumption for a large and established aircraft population. The most aggressive scenario studied introduced N+2 aircraft in large number but at the end of a 10-year production run (year 2040) only 31% of the worldwide fleet could be replaced with N+2 aircraft.
- A 17.5% reduction in fuel annually represents saving 74 billion kilograms of aviation fuel. According to the US Environmental Protection Agency, 74 billion kilograms produces the same Greenhouse gas emissions as 45.5 million passenger cars driven in one year.

Emission Projections

The introduction of five advanced NASA N+2 aircraft in the system in 2030, could reduce global CO_2 emissions produced by aviation by 21% by the year 2040 compared to the baseline scenario and 17.5% compared to Scenario 1.5. This is a significant reduction in emissions and fuel used worldwide. The results presented are sensitive to initial entry dates of the N+2 aircraft, the rate of production of N+2 aircraft, and the replacement strategy used. The model described in this report, provides NASA ISAAC with a tool to perform strategic assessments of future aviation technology.

CHAPTER 6 RECOMMENDATIONS

The model does not incorporate the freighter aircraft flights. The model can be improved if an OD demand for cargo services are included as well.

The GDM includes a demand elasticity factor to predict induces commercial air transportation demand if fares are reduced. The introduction of NASA N+2 could, in theory, reduce air fares if the fuel savings associated with the operation of more fuel-efficient aircraft are passed on to consumers. Induced demand will produce more air travel and higher global and emissions. This feedback effect can be further studied and integrated into the model.

Virginia Tech developed a simple GDM interface to facilitate the execution of the model. However, a graphical interface that includes Geographic Information Systems maps would make the model more useful.

To improve the estimates of the aircraft fleet worldwide, adoption of OAG publishes travel time data, which includes padding times, can be implemented. Airlines add travel time to flights to account for random schedule effects and to account for re-current airport delays. These additional travel times will require a larger global fleet to service the demand estimated in the model.

All the airports in the model are assumed to be at the sea level conditions. The model will be more realistic if the actual conditions, according to the altitude locations of the airports, are considered.

The model has mapped all the aircraft used for commercial aviation into 39 aircraft. Better results could be obtained if each aircraft is individually studied.

REFERENCES:

Alsalous,O. *Global Demand Forecast Model*, Virginia Polytechnic Institute And State University, December 3, 2015

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

Department of Transportation, Essential Air Service, https://www.transportation.gov/policy/aviation-policy/small-community-rural-air-service/essential-air-service, Accessed in 2015.

EUROCONTROL. "User Manual for the Base of Aircraft Data (BADA) Revision 3.13." User Manual, 2015.

FAA. (2014). Traffic Flow Management System (TFMS). Retrieved from http://aspmhelp.faa.gov/index.php/Traffic_Flow_Management_System_(TFMS)

Federal Aviation Administration (FAA). *Aviation System Performance Metrics.*https://aspm.faa.gov/apm/sys/main.asp (accessed 2016).

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

International Civil Aviation Organization (ICAO), '*Present and Future Trends In Aircraft Noise and Emissions,* 2013, Assembly — 38th Session)

Li, T. (2014). *General Aviation Demand Forecasting Models and A Microscopic North Atlantic Air Traffic Simulation Model*. Virginia Polytechnic Institute and State University.

Nickol, C., Haller, W., "Assessment of the Performance Potential of Advanced Subsonic Transport Concepts for NASA's Environmentally Responsible Aviation Project", 2016.

NOAA, ESRL, and PSD. "NCEP/NCAR Reanalysis Monthly Means and Other Derived Variables: Pressure Level." *Earth System Research Laboratory.* 2017. http://www.esrl.noaa.gov/psd/data/gridded/data.ncep.reanalysis.derived.pressu

re.html (accessed 2017).

Park, Yongha: O'Kelly, Mortana E. Fuel Burn Rates of Comercial Passenger Aircraft: Variations by Seat Configuration and Stage Distance. *Journal of Transport Geography, 2014, Vol 4*1.

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 Semina*r, 2009.

Swelbar, W., Update on US Airport and Regional Jet Capacity Trends, MIT Presentation, June 30, 2015.

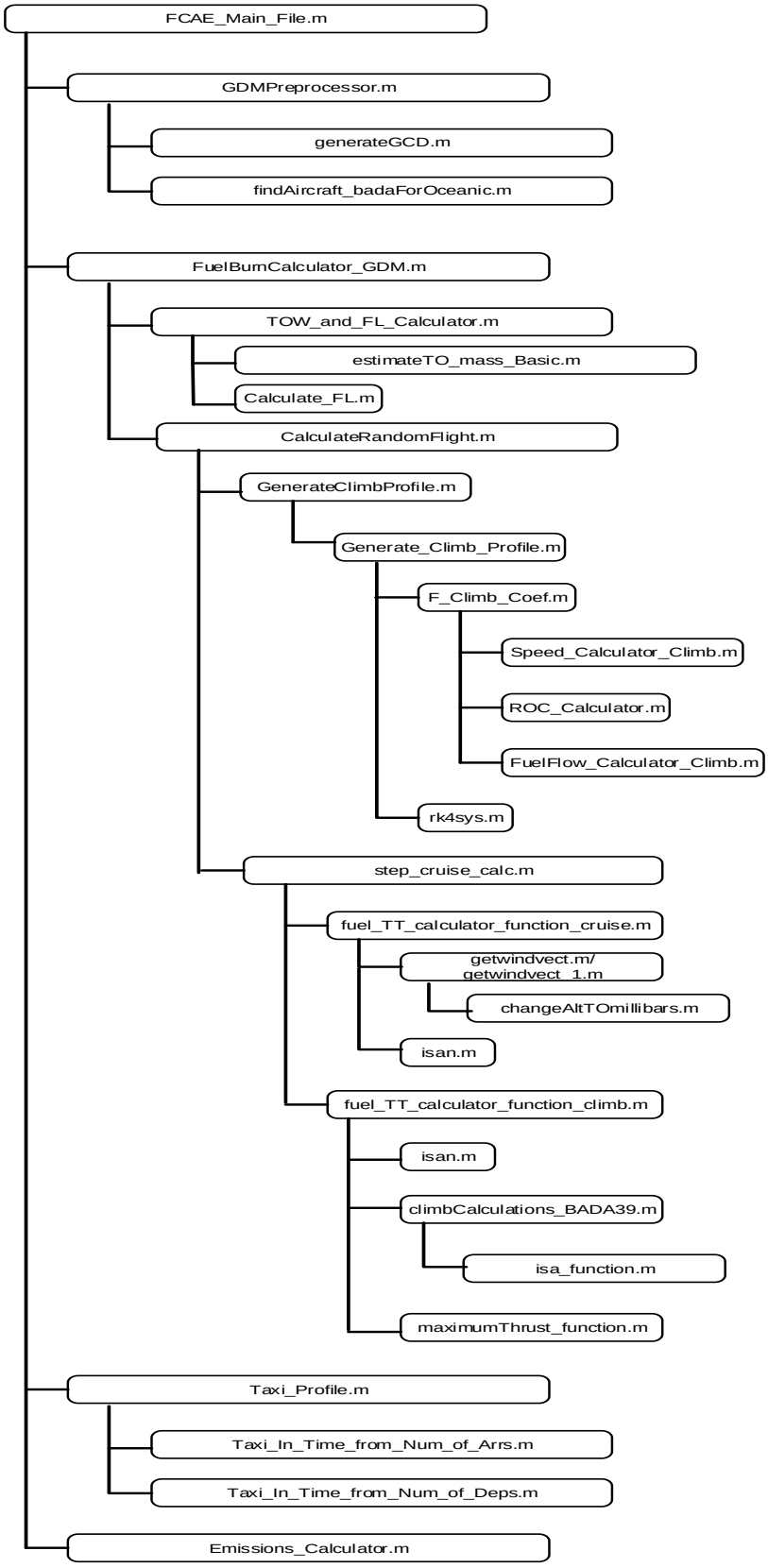
Wasiuk, D., Lowenberg, M., Shallcross, D. An Aircraft Performance Model implementation for the Estimation of Global and Regional Commercial Aviation Furl burn and Emission*s*. *Transportation research Part-D*, 2015

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.

Zou Z., Trani A.A. (2012). “A Computer Model to Estimate Commercial Aviation Fuel Consumption and Emissions in the Continental United States.

APPENDIX A – MODEL FLOWCHART

Functional Dependencies


FCAE_Main_File.m
GDMPreprocessor.m
generateGCD.m
findAircraft_badaForOceanic.m
FuelBurnCalculator_GDM.m
TOW_and_FL_Calculator.m
estimateTO_mass_Basic.m
Calculate_FL.m
CalculateRandomFlight.m
GenerateClimbProfile.m
Generate_Climb_Profile.m
F_Climb_Coef.m
Speed_Calculator_Climb.m
ROC_Calculator.m
FuelFlow_Calculator_Climb.m
rk4sys.m
step_cruise_calc.m
fuel_TT_calculator_function_cruise.m
getwindvect.m/
getwindvect_1.m
changeAltTOmillibars.m
isan.m
fuel_TT_calculator_function_climb.m
isan.m
climbCalculations_BADA39.m
isa_function.m
maximumThrust_function.m
Taxi_Profile.m
Taxi_In_Time_from_Num_of_Arrs.m
Taxi_In_Time_from_Num_of_Deps.m
Emissions_Calculator.m

APPENDIX B – SOURCE CODE

FCAE_Main_File.m

```
% This file runs the Fuel Consumption And Emissions Model
 function FCAE_Main_File(Install_Dir, Project_Dir, Case_Folder, Case_Name, Case_Year)
% function flight = FCAE_Main_File(Year)
clc;
warning off;

% ----------------------
% Settings if run inside MATALB
% ----------------------
% IMPORTANT: All inputs are now organized in "..\GDM_DATA\" folder. This
% the Git project GDM_DATA and contains the official inputs for this model
% for the GDM_GUI.
% Model runs are now saved in the following directory structure:
% Save_Dir = Project_Dir\Output\Case_Folder\Case_Name (See below)

if isdeployed == 0
    Install_Dir = '..\GDM_DATA\DATA';

    GOM_Dir        = '.\GOM';
    GOM_Project_Dir = 'D:\GDM Project\';

    Project_Dir = 'D:\GDM Project\Passenger';
    Case_Folder = 'Test_Folder_Scenario';
    Case_Name   = 'Passenger_FCAE';

    % Trip Generation Output Folder
    TD_Case_Folder = 'Test_Folder';
    TD_Case_Name   = 'Passenger Trip Distribution';

    FE_Case_Folder = 'Test_Folder';
    FE_Case_Name   = 'Passenger Fleet Evolution';

    % Change Scenarios and year for analysis
    Scenario = 1;
    Case_Year = 2025;
end

% Create Output Dir
Save_Dir = [Project_Dir, '\Fuel_Consumption_And_Emissions\Output\', Case_Folder, '\',
Case_Name];
mkdir(Save_Dir);
```

```
Trip_Distribution_Dir = [Project_Dir, '\Trip_Distribution\Output\', TD_Case_Folder,
'\', TD_Case_Name];
Fleet_Evolution_Dir   = [Project_Dir, '\Fleet_Evolution\Output\', FE_Case_Folder, '\',
FE_Case_Name];

rng(0);

% --------------------
% GDM Trip Distribution Data Pre Processing
% --------------------
%   GDMPreProcessor(Install_Dir, Save_Dir, Trip_Distribution_Dir, Fleet_Evolution_Dir,
Case_Year, Scenario);

% --------------------
% Calculate Fuel Consumtpion
% --------------------

%         FuelBurnCalculator_GDM(Install_Dir, Save_Dir, GOM_Dir, GOM_Project_Dir,
Case_Year);

% --------------------
% Taxi Fuel Burn
% --------------------
%       Taxi_Profile(Install_Dir, Save_Dir, Case_Year)

% --------------------
% Emissions Calculation
% --------------------
%
Emissions_Calculator(Install_Dir, Save_Dir, Case_Year);

return;
```

GDMPreProcessor.m

```
function GDMPreProcessor(Install_Dir, Save_Dir, Trip_Distribution_Dir,
Fleet_Evolution_Dir, Case_Year, Scenario)

% global aircraftArray;.... 2/20/17 for new Acft

global NASAaircraft aircraftArray
```

```
rng(0);

Case_Year_Str = num2str(Case_Year);

disp('Running Global Demand Preprocessor..');
disp('Loading GDM trip distribution output, please wait...');
disp(' ')
if Scenario ==1
load ([Fleet_Evolution_Dir, '\Aircraft_Distribution_All.mat']);
load ([Trip_Distribution_Dir, '\Trip_Distribution_All_Years.mat']);
load ([Trip_Distribution_Dir, '\Flight_by_each_aircraft_all.mat']);
else
load ([Fleet_Evolution_Dir,
'\Aircraft_by_year_Updated_NASA_Scenario',num2str(Scenario),'.mat']);
load ([Trip_Distribution_Dir, '\Trip_Distribution_All_Years.mat']);
load ([Trip_Distribution_Dir,
'\Flight_by_year_Updated_NASA_Scenario',num2str(Scenario),'.mat']);
end
load ([Install_Dir, '\Passenger\OAG_Passenger_Processing\airport_list.mat'])
load ([Install_Dir, '\Passenger\Fuel_Consumption_And_Emissions\NASAacft.mat']);
disp(' ')
disp('Data Loaded')
disp(' ')
load ('aircraftArray.mat')
load ('ACMaxRange.mat')
% load ([Input_Dir,'\Aircraft_Under_Production.mat']);
NASAaircraft = (NASAaircraft);
% nextgenacft = (Aircraft_Under_Production);
% load([Input_Dir,'\AircraftMatching_List.mat']);
% Initialize the variables here:
% GCDdist=zeros(length(Trip_Distribution_All.Year_2016.Departure_Airport),1);

load([Install_Dir,
'\Passenger\Fuel_Consumption_And_Emissions\badaForOceanic_4_17_17.mat']) % loading the
badaForOceanic 42Aircraft

% aircraftArray = {badaForOceanic.aircraftName};... 2/20/17 for new Acft

% Flights_by_each_acft_new = Flights_by_each_acft_all(1:1000);
if Scenario==1
    Aircraft_by_year_Updated  = Aircraft_Distribution_All;
```

```
    Flight_by_year_Updated = Flight_by_each_aircraft_all;
    Number_Of_OD_Pairs = length(Aircraft_by_year_Updated);
else
    Aircraft_by_year_Updated =
['Aircraft_by_year_Updated_NASA_Scenario_',num2str(Scenario)] ;
    Flight_by_year_Updated =
['Flight_by_year_Updated_NASA_Scenario',num2str(Scenario)];
    Number_Of_OD_Pairs = length(Aircraft_by_year_Updated);
end
flight = struct;
OD_Time = struct;

for od = 1:Number_Of_OD_Pairs

    if mod(od, 1000) == 0
        disp([num2str(od), '/', num2str(Number_Of_OD_Pairs)]);
    end

    % Skip if no data
    if isempty(Aircraft_by_year_Updated(od).(['Year_', Case_Year_Str]))
        continue;
    end

    % -------------
    % Common OD pair data
    % -------------
    origin = Trip_Distribution_All.(['Year_', Case_Year_Str]).Departure_Airport{od};
    dest   = Trip_Distribution_All.(['Year_', Case_Year_Str]).Arrival_Airport{od};
    seats  = Trip_Distribution_All.(['Year_', Case_Year_Str]).Seats(od);
    % Find the airport in airport_list:
    airportIdIndexOrigin = find(strcmp(airport_list.Airport_IDs,origin),1,'first');
    airportIdIndexDest   = find(strcmp(airport_list.Airport_IDs,dest),1,'first');

    % Get the OD Continent and Country

%     origincountryindex = find(strcmp(airport_list.Country,origin),1,'first');
%     origincontinentindex = find(strcmp(airport_list.G_Region,origin),1,'first');
%     destcountryindex = find(strcmp(airport_list.Country,dest),1,'first');
%     destcontinentindex = find(strcmp(airport_list.G_Region, dest),1,'first');

    origincountry = airport_list.Country(airportIdIndexOrigin);
    origincontinent = airport_list.G_Region(airportIdIndexOrigin);
```

```
    destcountry = airport_list.Country(airportIdIndexDest);
    destcontinent = airport_list.G_Region(airportIdIndexDest);

    % Get the OD lat & lon:
    originLat = airport_list.Apt_Lat(airportIdIndexOrigin);
    if originLat == 0
       disp(num2str(od));
    end
    originLon = airport_list.Apt_Lon(airportIdIndexOrigin);

    destLat = airport_list.Apt_Lat(airportIdIndexDest);
    if destLat == 0
       disp(num2str(od));
    end
    destLon = airport_list.Apt_Lon(airportIdIndexDest);
    noWaypoints = 20;

    [GCDdistance, latWpts, lonWpts] = generateGCD(originLat, originLon, destLat,
destLon, noWaypoints);

    % Enter in the fields of the Flight structure:
    flight(od).Origin     = origin;
    flight(od).Destination = dest;
    flight(od).Seats      = seats;

    flight(od).Destination_Country = destcountry;
    flight(od).Destination_Continent = destcontinent;
    flight(od).Origin_Country = origincountry;
    flight(od).Origin_Continent = origincontinent;

    flight(od).originLat = originLat;
    flight(od).originLon = originLon;
    flight(od).destLat   = destLat;
    flight(od).destLon   = destLon;

    % Indexing (0,0) airports
    if originLat == 0 || originLon==0 ||  destLat==0 || destLon==0
    OD_Time(od).Incorrect_Flight = -999;
    continue
    end
    flight(od).Flight_Plan_Type = 'GC';
    flight(od).GCDdistance     = GCDdistance;
```

```
    flight(od).Equipage = 'E';

    flight(od).GC_FP_Lat = latWpts;
    flight(od).GC_FP_Lon = lonWpts;

    % Assign directions
    Azimuth = azimuth(originLat, originLon, destLat, destLon);
    if Azimuth >= 0 && Azimuth < 180
        flight(od).direction = 'Easterly';
    else
        flight(od).direction = 'Westerly';
    end

    % -------------
    % Aircraft Data
    % -------------
    Number_Of_Aircraft = length(Aircraft_by_year_Updated(od).(['Year_',
Case_Year_Str]));
    %    Index_to_delete = false(1, Number_Of_Aircraft);
    for acft = 1:Number_Of_Aircraft

        % Enter in the fields of the Flight structure:
        originalAircraft = Aircraft_by_year_Updated(od).(['Year_',
Case_Year_Str])(acft);
        flight(od).badaForOceanic_Index(1,acft) =
findAircraft_badaForOceanic(originalAircraft);
        frequency = Flight_by_year_Updated(od).(['Year_', Case_Year_Str])(acft);

        if strcmp(originalAircraft, 'BE99__') == 1
            Aircraft_replaced = 'BE30__';
            originalAircraft = cellstr(Aircraft_replaced);
        end
        flight(od).originalAircraft(1,acft) = originalAircraft;
        flight(od).Frequency(1,acft) = frequency;
        flight(od).Frequency(1,acft) = ceil(flight(od).Frequency(acft));

        acindex =  find(strcmp (ACMaxrange.Aircraft, originalAircraft));
        aircraftmaxrange = ACMaxrange(acindex,2).MaxRange;
        if GCDdistance < 3
            OD_Time(od).Incorrect_Flight = -777;
            continue
```

```
        else
        end
        if GCDdistance > 8000
            OD_Time(od).Incorrect_Flight = -555;
            continue
        else
        end
        if (GCDdistance > aircraftmaxrange) & (GCDdistance < 8000)
%           OD_Time(od).Incorrect_Flight = zeros(1, Number_Of_Aircraft);
            OD_Time(od).Incorrect_Flight(1,acft) = -888;
%           false (OD_Time(od).Incorrect_Flight(1,length (Number_Of_Aircraft)));
            continue
        else
            OD_Time(od).Incorrect_Flight(1,acft) = 0;
        end



    end % for i = 1:Number_Of_Aircraft
    %    flight(od).originalAircraft(Index_to_delete) = [];
    %    flight(od).Frequency(Index_to_delete) = [];
    %    flight(od).badaForOceanic_Index(Index_to_delete) = [];
end % for od = 1:Number_Of_OD_Pairs

% -------------
% Remove short OD pairs
% -------------
% Min_GCD_Distance_nm = 3;
%
% GCD_Distances_nm = [flight.GCDdistance];
%
% Index_To_Delete = GCD_Distances_nm < Min_GCD_Distance_nm;
%
% % flight(Index_To_Delete) = [];
% OD_Time(Index_To_Delete) = -777;
% % -------------
% % Remove long OD pairs
% % -------------
% Min_GCD_Distance_nm = 7500;
%
% GCD_Distances_nm = [flight.GCDdistance];
%
```

```
% Index_To_Delete = GCD_Distances_nm > Min_GCD_Distance_nm;
%
% % flight(Index_To_Delete) = [];
% OD_Time(Index_To_Delete) = -555;
% % -------------
% % Remove zero latlon OD pairs
% % -------------
% Min_Origin_latlon = 0;
% Origin_Lat = [flight.originLat];
% Index_To_Delete = (Origin_Lat == Min_Origin_latlon);
% % flight(Index_To_Delete) = [];
% OD_Time(Index_To_Delete) = -999;
%
% Min_Dest_latlon = 0;
% dest_Lat = [flight.destLat];
% Index_To_Delete = (dest_Lat == Min_Dest_latlon);
% % flight(Index_To_Delete) = [];
% OD_Time(Index_To_Delete) = -999;
% -------------
% Saving
% -------------
disp(' ')
disp('Saving GDM and Pre-Processor and OD Time Results..');

disp(' ')
save ([Save_Dir, '\flight_1_', Case_Year_Str, '.mat'], 'flight', '-v7.3');
save ([Save_Dir, '\flight_1_OD_Time_', Case_Year_Str, '.mat'], 'OD_Time');

disp('Files Saved')
disp(' ')
return;
```

generateGCD.m

```
% Function to generate equally spaced waypoings for a flight
% from points A to B

function [distanceGCD, latWpts, lonWpts] = generateGCD(originLat,originLon,destLat,
destLon, noWaypoints)

% Generate equally spaced waypoint for the route

[latWpts,lonWpts] = gcwaypts(originLat,originLon,destLat,destLon,noWaypoints);

[~,distanceBetweenNodes] = legs(latWpts,lonWpts);
cumulativeDistance = cumsum(distanceBetweenNodes);          % cumulative distance

% calculate the GCD distance here as well

distdeg     =
distance(latWpts(1),lonWpts(1),latWpts(noWaypoints+1),lonWpts(noWaypoints+1));
distanceGCD  = deg2nm(distdeg);                  % GCD distance in nm

return;
```

findAircraft_badaForOceanic.m

```
% Function to estimate the aircraft position in the BADA file
structure
function [aircraftPosition] = findAircraft_badaForOceanic(aircraftType)

 global aircraftArray ....2/20/17 for new Acft
%  aircraftArray =
{'B763__';'B772__';'B744__';'B752__';'A333__';'A346__';'MD11__';'A388__';'B764__';'B76
2__';'B737__';'A320__';'A310__';'C750__';'FA7X__';'B773__';'B77W__';'B77L__';'CL60__';
'B788__';'B748__';'B733__';'A321__';'E145__';'E190__';'E170__';'A319__';'E135__';'B712
__';'B735__';'B738__';'B739__';'A332__';'AT45__';'AT72__';'CRJ2__';'CRJ9__';'DH8C__';'
DH8D__';'C208__';'BE30__';'MD82__';
'CS100__';'CS300__';'B737Max__';'B738Max__';'B739Max__';'A319neo__';'A320neo__';'A321n
eo__';'B777-8X__';'B777-9X__';'A350-
1000__';'A330neo__';'E190E2__';'E195E2__';'TW098__';'TW160__';'TW216__';'TW301__';'TW4
00__'};
%  aircraftPositionTable =[ 1  2  3 4   5   6   7   8   9   10  11  12  13  14  15  16  17
18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41
42  12  12  12  12  12  27  12  23  16  16  16  33  25  25  25  31  10  2  3];
```

```
aircraftindex = strcmp(aircraftArray.aircraftName, aircraftType);% finds position
index

aircraftPosition = aircraftArray(aircraftindex,2).aircraftPosition ;


% aircraftPosition = find(strcmp(aircraftArray, aircraftType));% finds position index


% Check for no match (means all zeros in position indices)

%     % no match - all zeros

if isempty(aircraftPosition)
    aircraftPosition = find(strcmp(aircraftArray.aircraftName, 'BE30__'));
    disp(['WARNING: ', aircraftType{1}, ' not found! BE30__ assigned.']);
end

return;
```

FuelBurnCalculator_GDM.m

```
%    Generates fuel Burn calculations for all stages of the flight with the Arman's
(Used in GOM)step climb approach for the cruise mode of the flight for the GDM
%     Codes written and edited by Rahul P 9/13/2016

function FuelBurnCalculator_GDM(Install_Dir, Save_Dir, GOM_Dir, GOM_Project_Dir,
Case_Year)

global NASAaircraft

Case_Year_Str = num2str(Case_Year);

rng(0);

% Loading the Preprocessor file for Fuel Burn Calculations
disp(' ')
disp('Running Fuel Burn Calculator..');
disp('Loading Flight Data, please wait...');
disp(' ')
load([Save_Dir, '\flight_1_', Case_Year_Str, '.mat'])
load([Save_Dir, '\flight_1_OD_Time_', Case_Year_Str,'.mat'])
disp('Data Loaded')
disp(' ')


if (ismac == 1)
    SLASH = '/';
elseif (ismac == 0)
    SLASH = '\';
end

%%
================================================================================
================================================================== %%
%%                          This part defines some variables as global so that they
can be used in parallel computing                                   %%

global Wind badaForOceanic NAT_Tracks  Constants Waypoints
global BADA_Aircraft_Data  BADA_Aircraft_Coef atmosphere
global WestTracksLatLon EastTracksLatLon ListOfFlightLevel_East_West_ft
global AC_List Track_vs_Cruise FL_Empirical_CDF InputParameters
CruisePhaseforShortRangeFlights
```

```
%%
================================================================================
================================================================== %%
%%                                                    This loads all the required
files                                                   %%

Date4Evaluation = 20140815;

% 1962 International Standard Atmosphere File
load([GOM_Dir, '\atmosphere']);
% BADA Aircraft Performance Operational File (OPF) (21 aircraft types)
%    load ([GOM_Dir, '\GOM\', Scenario_Settings.BADA_File, '.mat'])
% BADA Performance Table File (PTF) for all available aircraft (192 aircraft types)
load([GOM_Dir, '\BADA_Aircraft_Data.mat'])
% List with all available BADA Aircraft (192 aircraft types)
load([GOM_Dir, '\BADA_Aircraft_List.mat'])
% Coefficients for all available BADA Aircraft (192 aircraft types)
load([GOM_Dir, '\BADA_Aircraft_Coef.mat'])
load([GOM_Dir, '\badaForOceanic_4_17_17.mat'])
%Load Wind Parameters
load([GOM_Project_Dir, '\GOM\Input\Wind\20140815\OTSWind_20140815.mat'])
load([GOM_Project_Dir, '\GOM\Input\Wind\20140815\Wind_08_15_2014_DM.mat'])
load([GOM_Project_Dir, '\GOM\Input\Wind\20140815\WindByInterval_20140815.mat'])
load([GOM_Project_Dir, '\GOM\Input\OTS\20140815\NAT_Tracks_20140815.mat'])
load([GOM_Project_Dir, '\GOM\Input\OTS\20140815\trackArray_20140815_Basic.mat'])
load([GOM_Project_Dir, '\GOM\Input\OTS\20140815\trackArray_20140815_RLatSM.mat'])
load([GOM_Project_Dir, '\GOM\Input\OTS\20140815\NAT_OTS_20140815.mat'])

load([Install_Dir, '\Passenger\Fuel_Consumption_And_Emissions\AC_List.mat']); % List
of Aircraft in Track_Vs_Cruise
load([Install_Dir,
'\Passenger\Fuel_Consumption_And_Emissions\InputParameters.mat']); % Detour, Flight
Separation and Step Climb Check distance
load([Install_Dir,
'\Passenger\Fuel_Consumption_And_Emissions\nextgenacft_FuelburnSaving_Factors.mat']);
% Fuel Burn Coefficients for NASA acts
load([Install_Dir,
'\Passenger\Fuel_Consumption_And_Emissions\Track_vs_Cruise_Combined_nochangeincrosspoi
nt.mat']); % Used for Flight Level Assignment
```

```
load([Install_Dir,
'\Passenger\Fuel_Consumption_And_Emissions\FL_Empirical_CDF.mat']); % Used for Flight
Level Assignment. Four quarters
load([Install_Dir,
'\Passenger\Fuel_Consumption_And_Emissions\Cruise_Phase_ShortRange_Flights.mat']); %
Percent of Cruise for Short Range Flights
load([Install_Dir, '\Passenger\Fuel_Consumption_And_Emissions\NASAacft.mat']); % NASA
acfts



% Temporary Wind Set For Initial Calculations (1st of 4)
WindTemporary      = Time.Hrs_0to6;
WindTemporary.Level = Wind.Level;
WindTemporary.Lat   = Wind.Lat;
WindTemporary.Lon   = Wind.Lon;
Wind = WindTemporary;

% OTS Valid Time Period
Constants.Eastbound_OTS_Opening_Time_hrs = 1;
Constants.Eastbound_OTS_Closing_Time_hrs = 8;
Constants.Westbound_OTS_Opening_Time_hrs = 11.50;
Constants.Westbound_OTS_Closing_Time_hrs = 19.00;



%%                          Create Polygon Around Each
Track                                   %%
Polygon_Width_deg = 0.5;

% Eastbound
No_of_EastboundTracks = length(trackArrayEast);
for Eastbound_Track = 1:No_of_EastboundTracks
    This_Track_Lat = NAT_Tracks.(char(trackArrayEast(Eastbound_Track))).lat;
    This_Track_Lon = NAT_Tracks.(char(trackArrayEast(Eastbound_Track))).lon;
    This_Track_Polygon_Lat = [This_Track_Lat + Polygon_Width_deg, flip(This_Track_Lat -
Polygon_Width_deg), This_Track_Lat(1) + Polygon_Width_deg];
    This_Track_Polygon_Lon = [This_Track_Lon, flip(This_Track_Lon), This_Track_Lon(1)];
    NAT_Tracks.(char(trackArrayEast(Eastbound_Track))).Polygon_Lat =
This_Track_Polygon_Lat;
    NAT_Tracks.(char(trackArrayEast(Eastbound_Track))).Polygon_Lon =
This_Track_Polygon_Lon;
end % for Eastbound_Track = 1:No_of_EastboundTracks
```

```
% Westbound
No_of_WestboundTracks = length(trackArrayWest);
for Westbound_Track = 1:No_of_WestboundTracks
    This_Track_Lat = NAT_Tracks.(char(trackArrayWest(Westbound_Track))).lat;
    This_Track_Lon = NAT_Tracks.(char(trackArrayWest(Westbound_Track))).lon;
    This_Track_Polygon_Lat = [This_Track_Lat + Polygon_Width_deg, flip(This_Track_Lat -
Polygon_Width_deg), This_Track_Lat(1) + Polygon_Width_deg];
    This_Track_Polygon_Lon = [This_Track_Lon, flip(This_Track_Lon), This_Track_Lon(1)];
    NAT_Tracks.(char(trackArrayWest(Westbound_Track))).Polygon_Lat =
This_Track_Polygon_Lat;
    NAT_Tracks.(char(trackArrayWest(Westbound_Track))).Polygon_Lon =
This_Track_Polygon_Lon;
end % for Westbound_Track = 1:No_of_WestboundTracks


%%                              This part contains all essential
constants                         %%

Constants.FeetInNauticalMiles = 6076.11549;
% Detour factor
Constants.Detour = InputParameters.Value(1); % Detour Value for entire analysis
extracted from InputParameters
Constants.Wind_Boolean = false;
%  ConvergeDist_deg = nm2deg(-250); % 250 nm in deg

% Big Number To Use For Cost Matrix At Airways Impossible For A Flight
Constants.M = 999999;
nm2km = 1.852;


%%                                          Track Input (2
Sets)                                 %%

% Number of Tracks
NoTracksEast = numel(trackArrayEast);
NoTracksWest = numel(trackArrayWest);

% Tracks for wind data
% WndTrk_Map_East = trackArrayEast;
% WndTrk_Map_West = trackArrayWest;
```

```
%%                    Computation of Track Characteristics (Lat, Lon, Distance
etc)                %%

EastTracksLatLon =
zeros(NoTracksEast,10);                                                %...Matrix for
input(Rows: Tracks, Columns: Entry Lat, Entry Lon, Exit Lat, Exit Lon)
for j = 1:NoTracksEast
    [~,Lat,Lon] =
findTrackEntryExitCoordinates(trackArrayEast{j},'Easterly');           %...Calculate
Entry and Exit Point Coordinates
    EastTracksLatLon(j,1:4) =
[Lat(1),Lon(1),Lat(end),Lon(end)];                                  %...Save Entry and Exit
Point Coordinates
    EastTracksLatLon(j,5:6) =
distance('gc',EastTracksLatLon(j,1:2),EastTracksLatLon(j,3:4));     %...Saves GC
distance of the track in degrees(two intances (why???))
    p = find(Lon(1:end-1) <=-30 & Lon(2:end) >-30);
    k = (Lon(p+1)-Lon(p))/(Lat(p+1)-Lat(p)) ;
    EastTracksLatLon(j,7) = Lat(p) + (-30-Lon(p))/k;
    EastTracksLatLon(j,8) = -
30;                                                          %...30W is a point of
decision whether to use NAT or not...%
    [ArcLen,~] =
distance('gc',EastTracksLatLon(j,1:2),EastTracksLatLon(j,7:8));              %...Latit
ude for 30W Longitude...%
    EastTracksLatLon(j,9)    =  deg2nm(ArcLen); % to 30W
point                                %...Saves GC distance in nm between entry point and
30W latitude
    [~,dist] =
legs(Lat,Lon);                                                          %...Calculate
the distance between NAT waypoints (nm)
    EastTracksLatLon(j,10) =
sum(dist);                                                  %...Save the total length
of the track (nm)
end

WestTracksLatLon = zeros(NoTracksWest,10);
for j = 1:NoTracksWest
    [~,Lat,Lon] = findTrackEntryExitCoordinates(trackArrayWest{j},'Westerly');
    WestTracksLatLon(j,1:4) = [Lat(1),Lon(1),Lat(end),Lon(end)];
```

```
    WestTracksLatLon(j,5:6) =
distance('gc',WestTracksLatLon(j,1:2),WestTracksLatLon(j,3:4));
    p = find(Lon(1:end-1) >-30 & Lon(2:end) <=-30);
    if ~isempty(p)
        k = (Lon(p+1)-Lon(p))/(Lat(p+1)-Lat(p));
        WestTracksLatLon(j,7) = Lat(p) + (-30-Lon(p))/k;
    else
        p=find(Lon==-30);
        k = (Lon(p+1)-Lon(p))/(Lat(p+1)-Lat(p));
        WestTracksLatLon(j,7) = Lat(p) + (-30-Lon(p))/k;
    end
    WestTracksLatLon(j,8) = -30;
    [ArcLen,~] = distance('gc',WestTracksLatLon(j,1:2),WestTracksLatLon(j,7:8));
    WestTracksLatLon(j,9)    =  deg2nm(ArcLen);% to 30W point
    [~,dist] = legs(Lat,Lon);
    WestTracksLatLon(j,10) = sum((dist));

end


%%
Number_Of_OD_Pairs = length(flight);
% East flight altitudes according to hemispherical rules
Flight_Assn_East_Lo = InputParameters.Value(2)*[1:2:39,41:4:57]';
% West flight altitudes according to hemispherical rules
Flight_Assn_West_Lo = InputParameters.Value(2)*[2:2:40,43:4:59]';
ListOfFlightLevel_East_West_ft = [Flight_Assn_East_Lo,Flight_Assn_West_Lo];
Number_Random_Stream =
Number_Of_OD_Pairs;
       % A large Number to pre-allocate the Random Stream
%...Variable to inform user about the nature of errors.
% noFlights =
numel(flight);
   %...Number of flights to be calculated
TOM_Random_stream =
randn(Number_Of_OD_Pairs,1);
         %...Random stream of numbers for use in Takeoff Mass calculations. Using
rng(xxx) defined above this stream can be reproduced in any run, on any computer
%...percent of random flights executing step climbs inside the MNPS

OD_Time(Number_Of_OD_Pairs).Flight_Time = 0;
```

```
% OD_Time = zeros (1, length(Number_Of_OD_Pairs));
%% Computational loop for cost matrices                          %%
for i = 1:Number_Of_OD_Pairs
    if OD_Time(i).Incorrect_Flight < 0
        continue % Skip too short &/or too long flights &/or (0,0) latlon flights
    else
        if mod(i,1000) == 0
            disp([num2str(i), '/', num2str(Number_Of_OD_Pairs)]);
        end

        %        clear TakeOff_Mass Aircraft_Index Direction


        %%                                Find Act type and Bada
Index                              %%

        %        aircraftType = flight(i).originalAircraft;
        %    disp(['Calculating aircraft ',': ',aircraftType,'
(',num2str(i),'/',num2str(noFlights),')'])
        %        Direction = flight(i).direction;
        %        flight(IndexMatrix{j,1}).Origin_Airport_Altitude=0;
        flight(i).Origin_Airport_Altitude     = 0;
        flight(i).Destination_Airport_Altitude = 0;

        if isempty(flight(i).Origin) || isempty(flight(i).Destination) ||
(flight(i).originLon == 0 && flight(i).originLat == 0) || (flight(i).destLon == 0 &&
flight(i).destLat == 0)
            disp(['WARNING: Flight index ', num2str(i), ' has missing coordinates for
origin/destination airports.']);
            continue;
        end

        %           nodeletedrows = numel(isempty(flight(i).Origin));
        %           badaForOceanic_Index = cell2mat(flight(i).badaForOceanic_Index(1));

        % Continue If The Aircraft Was Not Found In The BADA List
        %           if isempty(badaForOceanic_Index)
        %               comments(comment).info = char(['aircraft ',aircraftType, ' not
found', ' for flight ',num2str(i)]);
        %               comment=comment+1;
        %               disp([' Problem in Record',num2str(i)])
        %               continue
```

```
%           end
%           Minimum_Stage_Length_nm = 1000;
% %               Continue If The Flight Is Short
%           if cell2mat(flight(i).GCDdistance) < Minimum_Stage_Length_nm
%               disp(['  Short Flight (< ', num2str(Minimum_Stage_Length_nm), '
nm)'])
%               flight(i).NewID = 0;
% %               %           Index_To_Delete = isempty(flight(i).NewID )==1;
% %               %           flight(Index_To_Delete) = [];
% %               Continue
%           end
      Equipped = strcmp(flight(i).Equipage,'E');  %...1 for equipped aircraft, 0 for
non equipped                                                                %...New
field in flight matrix for TOmass
      flight(i).Origin_LatLon(1)= flight(i).originLat;
      flight(i).Origin_LatLon(2)= flight(i).originLon;
      flight(i).Destination_LatLon(1)= flight(i).destLat;
      flight(i).Destination_LatLon(2)= flight(i).destLon;
      if strcmp(flight(i).Flight_Plan_Type, 'TFMS') == 1
          FP_Lat = [flight(i).Origin_LatLon(1); flight(i).TFMS_FP_Lat;
flight(i).Destination_LatLon(1)];
          FP_Lon = [flight(i).Origin_LatLon(2); flight(i).TFMS_FP_Lon;
flight(i).Destination_LatLon(2)];
      elseif strcmp(flight(i).Flight_Plan_Type, 'Flight_Plan_Generator') == 1
          Flight_Plan_Index = max(1,min((13-(41000 - Assn_Cruise_FL)/1000),13));
          FP_Lat = [flight(i).Origin_LatLon(1);
flight(i).Flight_Plan_Generator(Flight_Plan_Index).Time.Waypoint(:,1);
flight(i).Destination_LatLon(1)];
          FP_Lon = [flight(i).Origin_LatLon(2);
flight(i).Flight_Plan_Generator(Flight_Plan_Index).Time.Waypoint(:,2);
flight(i).Destination_LatLon(2)];
      elseif strcmp(flight(i).Flight_Plan_Type, 'GC') == 1 % Great Circle Route
          FP_Lat = [flight(i).Origin_LatLon(1); flight(i).GC_FP_Lat;
flight(i).Destination_LatLon(1)];
          FP_Lon = [flight(i).Origin_LatLon(2); flight(i).GC_FP_Lat;
flight(i).Destination_LatLon(2)];
      else
          error(['No flight plan assigned for Internal ID: ',
num2str(flight(i).GCDdistance)]);
      end

      % Add intermediate pointsd along flight plan
```

```
        [FP_Lat, FP_Lon] = track(FP_Lat, FP_Lon);
        FP_Lat(isnan(FP_Lat)) = [];
        FP_Lon(isnan(FP_Lon)) = [];
        if FP_Lat > FP_Lat
            disp(num2str(i))
        end
        flight(i).Simulation_FP_Lat = FP_Lat;
        flight(i).Simulation_FP_Lon = FP_Lon;

        [~, flight(i).Simulation_Dist_Between_Points_nm] = legs(FP_Lat,FP_Lon);
        flight(i).Simulation_Stage_Length_nm =
sum(flight(i).Simulation_Dist_Between_Points_nm);

        %%                                      %%
        %           [flight_fuel] = findAircraftIndex(flight);
        for aIndex =1:length(flight(i).originalAircraft)
            if isempty(OD_Time(i).Incorrect_Flight)== 0
                if OD_Time(i).Incorrect_Flight(1, aIndex) < 0
                    continue % Skip Incorrect Flight Assignment
                else
                    %%  Find Aircraft Index In Old Bada (192 Aircraft)
                    AircraftInBadaList = {BADA_Aircraft_Data.Aircraft_Name};
                    %               aircraftType = flight(i).originalAircraft;
                    badaForOceanic_Index = flight(i).badaForOceanic_Index(aIndex);
                    AircraftName      =
badaForOceanic(badaForOceanic_Index).aircraftName;
                    [~, flight(i).Aircraft_Index_in_BADA(aIndex)] =
intersect(AircraftInBadaList,AircraftName);

                    %%                              Takeoff Weight
Calculations                                        %%
%                       if badaForOceanic_Index  == 8
%                           if flight(i).GCDdistance > 4000
%                               disp(num2str(i));
%                           end
%                       end

                    try
                        [TakeOff_Mass, Assn_Cruise_FL] = TOW_and_FL_Calculator(flight(i),
AircraftName, TOM_Random_stream(i), Equipped, aIndex);
                    catch
                        i
```

```
                end
                flight(i).TOmass(aIndex) = TakeOff_Mass;
                flight(i).Assigned_Cruise_FL(aIndex) = Assn_Cruise_FL;
                %               [Climb_Profile] = GenerateClimbProfile(flight(i),
TakeOff_Mass, aIndex);

                %%                                  Calculate Random
Flight                              %%

                %    try
                [Climb_Profile,Step_Cruise_Phase,Descent_Profile, LandingMass] =
CalculateRandomFlight(flight(i),TakeOff_Mass,Assn_Cruise_FL,...
                    badaForOceanic(badaForOceanic_Index).typicalMach,Equipped,aIndex);

                % Enter in the fields of the Flight structure:
                flight(i).Step_Cruise_Phase(aIndex) = Step_Cruise_Phase;
                flight(i).Climb_Profile(aIndex) = Climb_Profile;
                flight(i).Descent_Profile(aIndex) = Descent_Profile;
                flight(i).LandingMass(aIndex) = LandingMass;

                %    Fuel Burn reductions for Next generation and NASA Acft
                if strcmp(flight(i).originalAircraft(aIndex),'CS100__') == 1 ||
strcmp(flight(i).originalAircraft(aIndex),'CS300__') == 1 ||
strcmp(flight(i).originalAircraft(aIndex),'A319neo__') ==
1||strcmp(flight(i).originalAircraft(aIndex),'A320neo__') ==
1||strcmp(flight(i).originalAircraft(aIndex),'A321neo__') ==
1||strcmp(flight(i).originalAircraft(aIndex),'B777-8X__') ==
1||strcmp(flight(i).originalAircraft(aIndex),'B777-9X__') == 1
                    flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent  ) -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(sum(flight(i).Step_Cruise_P
hase(aIndex).Fuel)+ flight(i).Descent_Profile(aIndex).fuelUsedDescent ));
                elseif  strcmp(flight(i).originalAircraft(aIndex),'B737Max__') == 1 ||
strcmp(flight(i).originalAircraft(aIndex),'B738Max__') == 1 ||
strcmp(flight(i).originalAircraft(aIndex),'B739Max__') ==
1||strcmp(flight(i).originalAircraft(aIndex),'E190E2__') == 1    %
                    flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(sum(flight(i).Step_Cruise_P
hase(aIndex).Fuel)+ flight(i).Descent_Profile(aIndex).fuelUsedDescent ));
```

```
                elseif  strcmp(flight(i).originalAircraft(aIndex),'A330neo__') == 1
                    flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(sum(flight(i).Step_Cruise_P
hase(aIndex).Fuel)+ flight(i).Descent_Profile(aIndex).fuelUsedDescent  ));
                elseif   strcmp(flight(i).originalAircraft(aIndex),'E195E2__') == 1
                    flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(sum(flight(i).Step_Cruise_P
hase(aIndex).Fuel)+ flight(i).Descent_Profile(aIndex).fuelUsedDescent  ));
                elseif  strcmp(flight(i).originalAircraft(aIndex),'A350-1000__') == 1
                    flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(sum(flight(i).Step_Cruise_P
hase(aIndex).Fuel)+ flight(i).Descent_Profile(aIndex).fuelUsedDescent ));
                elseif   strcmp(flight(i).originalAircraft(aIndex),'TW098__') == 1
                    flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
((NASAaircraft(1).Fuel_Burn_Efficiency)*(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)
+ flight(i).Descent_Profile(aIndex).fuelUsedDescent));
                elseif   strcmp(flight(i).originalAircraft(aIndex),'TW160__') == 1
                    flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
((NASAaircraft(2).Fuel_Burn_Efficiency)*(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)
+ flight(i).Descent_Profile(aIndex).fuelUsedDescent ));
                elseif   strcmp(flight(i).originalAircraft(aIndex),'TW216__') == 1
                    flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent) -
((NASAaircraft(3).Fuel_Burn_Efficiency)*(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)
+ flight(i).Descent_Profile(aIndex).fuelUsedDescent ));
                elseif   strcmp(flight(i).originalAircraft(aIndex),'TW301__') == 1
                    flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
((NASAaircraft(4).Fuel_Burn_Efficiency)*(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)
+ flight(i).Descent_Profile(aIndex).fuelUsedDescent  ));
```

```
                elseif   strcmp(flight(i).originalAircraft(aIndex),'TW400__') == 1
                    flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
((NASAaircraft(5).Fuel_Burn_Efficiency)*(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)
+ flight(i).Descent_Profile(aIndex).fuelUsedDescent  ));
                elseif ismember(AircraftName, BADA_Aircraft_List)==1
                    flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent );
                end
                flight(i).Total_Fuel_Burnt_kg(aIndex) = (flight(i).Frequency(aIndex))
* (flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex));
                flight(i).Total_Time_of_Flight_min(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Time)+
flight(i).Descent_Profile(aIndex).timeToDescend );
                OD_Time(i).Flight_Time(1,aIndex) =
flight(i).Total_Time_of_Flight_min(aIndex);

            end
          else
            %%  Find Aircraft Index In Old Bada (192 Aircraft)
            AircraftInBadaList = {BADA_Aircraft_Data.Aircraft_Name};
            %              aircraftType = flight(i).originalAircraft;
            badaForOceanic_Index = flight(i).badaForOceanic_Index(aIndex);
            AircraftName      = badaForOceanic(badaForOceanic_Index).aircraftName;
            [~, flight(i).Aircraft_Index_in_BADA(aIndex)] =
intersect(AircraftInBadaList,AircraftName);

            %%                              Takeoff Weight Calculations and Flight
Level Assignmnet                                    %%
            %       if badaForOceanic_Index  == 31
            %           disp(num2str(i));
            %       end

            try
                [TakeOff_Mass, Assn_Cruise_FL] = TOW_and_FL_Calculator(flight(i),
AircraftName, TOM_Random_stream(i), Equipped, aIndex);
            catch
                i
            end
            flight(i).TOmass(aIndex) = TakeOff_Mass;
```

```
            flight(i).Assigned_Cruise_FL(aIndex) = Assn_Cruise_FL;
            %               [Climb_Profile] = GenerateClimbProfile(flight(i),
TakeOff_Mass, aIndex);

            %%                                  Calculate Random
Flight                                  %%

            %    try
            [Climb_Profile,Step_Cruise_Phase,Descent_Profile, LandingMass] =
CalculateRandomFlight(flight(i),TakeOff_Mass,Assn_Cruise_FL,...
                badaForOceanic(badaForOceanic_Index).typicalMach,Equipped,aIndex);

            % Enter in the fields of the Flight structure:
            flight(i).Step_Cruise_Phase(aIndex) = Step_Cruise_Phase;
            flight(i).Climb_Profile(aIndex) = Climb_Profile;
            flight(i).Descent_Profile(aIndex) = Descent_Profile;
            flight(i).LandingMass(aIndex) = LandingMass;

            %    Fuel Burn reductions for Next generation and NASA Acft
            if strcmp(flight(i).originalAircraft(aIndex),'CS100__') == 1 ||
strcmp(flight(i).originalAircraft(aIndex),'CS300__') == 1 ||
strcmp(flight(i).originalAircraft(aIndex),'A319neo__') ==
1||strcmp(flight(i).originalAircraft(aIndex),'A320neo__') ==
1||strcmp(flight(i).originalAircraft(aIndex),'A321neo__') ==
1||strcmp(flight(i).originalAircraft(aIndex),'B777-8X__') ==
1||strcmp(flight(i).originalAircraft(aIndex),'B777-9X__') == 1
                flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent  ) -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(sum(flight(i).Step_Cruise_P
hase(aIndex).Fuel)+ flight(i).Descent_Profile(aIndex).fuelUsedDescent ));
            elseif  strcmp(flight(i).originalAircraft(aIndex),'B737Max__') == 1 ||
strcmp(flight(i).originalAircraft(aIndex),'B738Max__') == 1 ||
strcmp(flight(i).originalAircraft(aIndex),'B739Max__') ==
1||strcmp(flight(i).originalAircraft(aIndex),'E190E2__') == 1    %
                flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(sum(flight(i).Step_Cruise_P
hase(aIndex).Fuel)+ flight(i).Descent_Profile(aIndex).fuelUsedDescent ));
            elseif  strcmp(flight(i).originalAircraft(aIndex),'A330neo__') == 1
```

```
                flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(sum(flight(i).Step_Cruise_P
hase(aIndex).Fuel)+ flight(i).Descent_Profile(aIndex).fuelUsedDescent  ));
            elseif   strcmp(flight(i).originalAircraft(aIndex),'E195E2__') == 1
                flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(sum(flight(i).Step_Cruise_P
hase(aIndex).Fuel)+ flight(i).Descent_Profile(aIndex).fuelUsedDescent  ));
            elseif  strcmp(flight(i).originalAircraft(aIndex),'A350-1000__') == 1
                flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(sum(flight(i).Step_Cruise_P
hase(aIndex).Fuel)+ flight(i).Descent_Profile(aIndex).fuelUsedDescent ));
            elseif   strcmp(flight(i).originalAircraft(aIndex),'TW098__') == 1
                flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
((NASAaircraft(1).Fuel_Burn_Efficiency)*(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)
+ flight(i).Descent_Profile(aIndex).fuelUsedDescent));
            elseif   strcmp(flight(i).originalAircraft(aIndex),'TW160__') == 1
                flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
((NASAaircraft(2).Fuel_Burn_Efficiency)*(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)
+ flight(i).Descent_Profile(aIndex).fuelUsedDescent ));
            elseif   strcmp(flight(i).originalAircraft(aIndex),'TW216__') == 1
                flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent) -
((NASAaircraft(3).Fuel_Burn_Efficiency)*(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)
+ flight(i).Descent_Profile(aIndex).fuelUsedDescent ));
            elseif   strcmp(flight(i).originalAircraft(aIndex),'TW301__') == 1
                flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
((NASAaircraft(4).Fuel_Burn_Efficiency)*(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)
+ flight(i).Descent_Profile(aIndex).fuelUsedDescent  ));
            elseif   strcmp(flight(i).originalAircraft(aIndex),'TW400__') == 1
```

```
                flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent ) -
((NASAaircraft(5).Fuel_Burn_Efficiency)*(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)
+ flight(i).Descent_Profile(aIndex).fuelUsedDescent  ));
            elseif ismember(AircraftName, BADA_Aircraft_List)==1
                flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Fuel)+
flight(i).Descent_Profile(aIndex).fuelUsedDescent );
            end
            flight(i).Total_Fuel_Burnt_kg(aIndex) = (flight(i).Frequency(aIndex)) *
(flight(i).Total_Fuel_Burnt_kg_oneflight(aIndex));
            flight(i).Total_Time_of_Flight_min(aIndex) =
(sum(flight(i).Step_Cruise_Phase(aIndex).Time)+
flight(i).Descent_Profile(aIndex).timeToDescend );
            OD_Time(i).Flight_Time(1,aIndex) =
flight(i).Total_Time_of_Flight_min(aIndex);
            %
            %           end
         end %  OD_Time(i).Incorrect_Flights(1,aIndex) < 0
      end%for aIndex=1:length(flight(i).originalAircraft)
      % calculation of Average Flight time fot the OD
      OD_Times = OD_Time(i).Flight_Time;

      incorrect_flight_indices = find(OD_Times == 0);
      OD_Time(i).Avg_Flight_time_min = sum
(flight(i).Total_Time_of_Flight_min)/( numel (flight(i).Total_Time_of_Flight_min)-
numel (incorrect_flight_indices));
   end % if OD_Time(i).Incorrect_Flights < 0
end % for i = 1:Number_Of_OD_Pairs

% Save Output
disp(' ')
disp('Saving Fuel Burn and OD Time Results..');

disp(' ')
save([Save_Dir, '\flight_2_', Case_Year_Str, '.mat'], 'flight', '-v7.3');
save([Save_Dir, '\flight_2_OD_Time', Case_Year_Str, '.mat'], 'OD_Time');

disp('Files Saved')
disp(' ')
```

```
return;
```

TOW_and_FL_Calculator.m

```
 %%
================================================================================
========= %%
%%     This Function Is Used by The Fuel Burn Calculator To Estimate Takeoff Mass And
Cruise Altitude     %%

%%
================================================================================
========= %%
%%
================================================================================
========= %%
function [Take_Off_Mass_kg, Assn_Cruise_FL]=
TOW_and_FL_Calculator(flight,AircraftName , TOM_Random_stream, Equipped, aIndex)

global badaForOceanic Constants InputParameters
global AC_List Track_vs_Cruise FL_Empirical_CDF

% 1) Direction-wise Desired takeoff weight
if  strcmp(flight.direction,'Easterly')
    DTW_Dist4ThisAircraft =
badaForOceanic(flight.badaForOceanic_Index(aIndex)).DTWEast;
else
    DTW_Dist4ThisAircraft =
badaForOceanic(flight.badaForOceanic_Index(aIndex)).DTWWest;
end

% 2) Takeoff Mass
TakeOff_Mass =
estimateTO_mass_Basic(badaForOceanic(flight.badaForOceanic_Index(aIndex)),
flight.GCDdistance * InputParameters.Value(1), TOM_Random_stream,
DTW_Dist4ThisAircraft);
if isnan(TakeOff_Mass)
    TakeOff_Mass =
estimateTO_mass_Alternative(badaForOceanic(flight.badaForOceanic_Index(aIndex)),
flight.GCDdistance * InputParameters.Value(1), TOM_Random_stream,
DTW_Dist4ThisAircraft);
end
flight.TOmass = TakeOff_Mass;
Take_Off_Mass_kg = TakeOff_Mass;
%%
```

```
Track_Distance_nm = (flight.GCDdistance)*InputParameters.Value(1);
Max_Cruise_FL = (badaForOceanic(flight.badaForOceanic_Index(aIndex)).maxAltitude);

%% EDMS Matching
%3) Maximum operational altitude based on TOW USing Flights recorded in
%TFMS 2016


ETMS_BADA_Match = find(strcmp(AircraftName,AC_List) == 1);
if ETMS_BADA_Match > 0
    % 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 <= 3
%         Flight_Profile = [];
%         Climb_And_Descent_Distance = [];
%         return
%     end
    % -------------------------------------------------------------------------

    % --------------
    % GENERATE RANDOM FLIGHT LEVEL
    % --------------
    % EAST/WEST DIRECTIONS RESULT IN DIFFERENT CRUISE ALTITUDES
    Direction = flight.direction;
    if strcmp('Easterly',Direction) == 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;

        %%

            %       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
        if 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 isempty (Index_ETMS_FL) == 1
                Min_FL = interp1(East_Track_Pts, East_Min_FL_Pts,Rounded_Track_Dist,
'pchip');
                Max_FL = interp1(East_Track_Pts, East_Max_FL_Pts,Rounded_Track_Dist,
'pchip');
            end
            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
            Max_Cruise_FL = Max_Cruise_FL/100;
            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
            Max_Cruise_FL = Max_Cruise_FL/100;
            Raw_Cruise_FL =  Raw_Cruise_FL/100;
            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
            Max_Cruise_FL = Max_Cruise_FL/100;
            Raw_Cruise_FL =  Raw_Cruise_FL/100;
            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)/100;
            %         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
            Max_Cruise_FL = Max_Cruise_FL/100;
            if Raw_Cruise_FL > Max_Cruise_FL
                Raw_Cruise_FL = Max_Cruise_FL;
            end
        end


    elseif strcmp('Westerly',Direction) == 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
    if 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 isempty (Index_ETMS_FL) == 1
            Min_FL = interp1(West_Track_Pts, West_Min_FL_Pts,Rounded_Track_Dist,
'pchip');
            Max_FL = interp1(West_Track_Pts, West_Max_FL_Pts,Rounded_Track_Dist,
'pchip');
        end
        if (Min_FL < 0)
            disp( AircraftName);
        end
        if (Max_FL < 0)
            disp( AircraftName);
        end
        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
        Max_Cruise_FL = Max_Cruise_FL/100;
        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
            Max_Cruise_FL = Max_Cruise_FL/100;
            Raw_Cruise_FL =  Raw_Cruise_FL/100;
            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
            Max_Cruise_FL = Max_Cruise_FL/100;
            Raw_Cruise_FL =  Raw_Cruise_FL/100;
            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)/100;
            %         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
            Max_Cruise_FL = Max_Cruise_FL/100;
            if Raw_Cruise_FL > Max_Cruise_FL
                Raw_Cruise_FL = Max_Cruise_FL;
            end
        end
    end
    % ------------------------------------------------------------------------


    % --------------
    % ASSIGN ATC FL
    % --------------
    % % Check that Raw_FL does not violate airport altitudes
    % Max_Airport_Altitude = max(Origin_Airport_Altitude,Destination_Airport_Altitude)
/ 100;
    % if Raw_Cruise_FL < Max_Airport_Altitude + 20
    %     Raw_Cruise_FL = Max_Airport_Altitude + 20;
    %     % Reject flight if cannot be flown
    %     if Raw_Cruise_FL > Max_Cruise_FL
    %         Flight_Profile = [];
    %         Climb_And_Descent_Distance = [];
    %         return;
    %     end
    % end
    %     if Raw_Cruise_FL > 500
    %         disp(num2str(aIndex));
    %     end
    Raw_Cruise_FL = Raw_Cruise_FL*100;


    Max_Cruise_FL = Max_Cruise_FL*100;
    Assn_Cruise_FL = Calculate_FL(Raw_Cruise_FL, Direction, Max_Cruise_FL);


    % Direction
    if strcmp(flight.direction,'Easterly')
        DirectionIndex = 1;
```

```
    else
        DirectionIndex =2;
    end
    %% Checking if the flight level assigned does not allow any cruise phase i.e.
GCDdistance < (Climb distance + Descend Distance)
    % New Flight Level assigned if required

    Dist_to_climb   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,3),badaForOc
eanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,2),(Assn_Cruise_FL*0.3048),'p
chip');
    Dist_to_descent   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,3),badaFo
rOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,2),(Assn_Cruise_FL*0.30
48),'pchip');
    if flight.GCDdistance < (Dist_to_climb + Dist_to_descent)
        if DirectionIndex == 1
            Flight_Assn_East_Lo = InputParameters.Value(2)*[1:2:82];
            for index =1: length (Flight_Assn_East_Lo)
                Dist_to_descent   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,3),badaFo
rOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,2),(Flight_Assn_East_Lo
(index)*0.3048),'pchip');
                Dist_to_climb   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,3),badaForOc
eanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,2),(Flight_Assn_East_Lo(index
)*0.3048),'pchip');
                if  (Dist_to_climb + Dist_to_descent) - flight.GCDdistance > 0
                    indice = index - 1;
                    if index == 1
                        indice = 1;
                    end
                    Assn_Cruise_FL = Flight_Assn_East_Lo(indice);
                    break;
                end
            end

        else
            Flight_Assn_West_Lo = InputParameters.Value(2)*[2:2:80];
            for index =1: length (Flight_Assn_West_Lo)
                Dist_to_descent   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,3),badaFo
```

```
rOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,2),(Flight_Assn_West_Lo
(index)*0.3048),'pchip');
            Dist_to_climb   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,3),badaForOc
eanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,2),(Flight_Assn_West_Lo(index
)*0.3048),'pchip');
            if  (Dist_to_climb + Dist_to_descent) - flight.GCDdistance > 0
                indice = index - 1;
                if index == 1
                    indice = 1;
                end
                Assn_Cruise_FL = Flight_Assn_West_Lo(indice);
                break;
                %                       typ_alt_in_m  =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).nominalClimb(:,2),badaForO
ceanic(flight.badaForOceanic_Index(aIndex)).nominalClimb(:,3),Dist_to_climb,'pchip');
                %                       PreNATalt = floor(typ_alt_in_m
*3.2808/1000)*1000;
                %                       Assn_Cruise_FL = Calculate_FL(PreNATalt,
Heading, Max_Cruise_FL);
                %                   end
            end
        end


    end%    [flight_fuel] = findAircraftIndex (flight);

end

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


% --------------
% PROFILE LOOP
% --------------
% while(1)
%
%    % --------------
%    % GENERATE CLIMB PROFILE
%    % --------------
```

```
    %     Climb_Profile =
Generate_Climb_Profile(Current_BADA_Aircraft_Data,Current_BADA_Aircraft_Coef,Initial_W
eight_kg,Origin_Airport_Altitude,Waypoints,Assn_Cruise_FL,Wind,Wind_Parameters,Wind_Bo
olean);
    %     % ------------------------------------------------------------------
    %
    %     % --------------
    %     % GENERATE DESCENT PROFILE
    %     % --------------
    %     Descent_Profile =
Generate_Descent_Profile(Current_BADA_Aircraft_Data,Destination_Airport_Altitude,Waypo
ints,Assn_Cruise_FL,Wind,Wind_Parameters,Wind_Boolean);
    %     % ------------------------------------------------------------------
    %
    %     % --------------
    %     % 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(Current_BADA_Aircraft_Coef,
Climb_Profile, Descent_Profile,
Cruise_Distance_nm,Waypoints,Assn_Cruise_FL,Wind,Wind_Parameters,Wind_Boolean);
    %     % ------------------------------------------------------------------
    %
    %     % --------------
    %     % 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;
    % ------------------------------------------------------------------------

else
    % Flight Level assignment for Aircraft not in the TFMS (formerly ETMS) list of
Track_Vs_Cruise
```

```
    maxOperationalAltitude =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).operationalAltMass,
badaForOceanic(flight.badaForOceanic_Index(aIndex)).maxOperationalAltitude,
TakeOff_Mass, 'pchip');
    typ_alt = min(badaForOceanic(flight.badaForOceanic_Index(aIndex)).typicalAltitude,
maxOperationalAltitude);
    PreNATalt = floor(typ_alt/1000)*1000;
    typ_alt_in_m = typ_alt * 0.3048;
    if typ_alt_in_m <
max(badaForOceanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,3))
    Dist_to_climb   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,3),badaForOc
eanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,2),typ_alt_in_m,'pchip');
    Dist_to_descent   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,3),badaFo
rOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,2),(maxOperationalAltit
ude*0.3048),'pchip');
    else
    Dist_to_climb   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).nominalClimb(:,3),badaForO
ceanic(flight.badaForOceanic_Index(aIndex)).nominalClimb(:,2),typ_alt_in_m,'pchip');
    Dist_to_descent   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,3),badaFo
rOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,2),(maxOperationalAltit
ude*0.3048),'pchip');

    end

    if flight.GCDdistance < (Dist_to_climb + Dist_to_descent)
        typ_alt_in_m  =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,2),badaForOc
eanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,3),flight.GCDdistance,'pchip'
);
        PreNATalt = floor(typ_alt_in_m *3.2808/1000)*1000;
    end

    % 4) Maximum Cruise Flight Level based on equipage
    if Equipped
        Max_Cruise_FL =
badaForOceanic(flight.badaForOceanic_Index(aIndex)).maxAltitude;
    else
```

```
       Max_Cruise_FL =
min(badaForOceanic(flight.badaForOceanic_Index(aIndex)).maxAltitude,
Constants.MNPS_Unequipped_max);
    end

    % 5) Set Heading and DirectionIndex based on flight direction
    if strcmp(flight.direction,'Easterly')
       Heading = 'Easterly';
       DirectionIndex = 1;
    else Heading = 'Westerly';
       DirectionIndex =2;
    end
%    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];
%

    % 6) Assign FL using hemispherical rules
    Assn_Cruise_FL = Calculate_FL(PreNATalt, Heading, Max_Cruise_FL);
    %% Checking if the flight level assigned does not allow any cruise phase i.e.
GCDdistance < (Climb distance + Descend Distance)
    % New Flight Level assigned if required

    Dist_to_climb   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,3),badaForOc
eanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,2),(Assn_Cruise_FL*0.3048),'p
chip');
    Dist_to_descent   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,3),badaFo
rOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,2),(Max_Cruise_FL*0.304
8),'pchip');
    if flight.GCDdistance < (Dist_to_climb + Dist_to_descent)
       if DirectionIndex == 1
          Flight_Assn_East_Lo = InputParameters.Value(2)*[1:2:82];
          for index =1: length (Flight_Assn_East_Lo)
             Dist_to_descent   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,3),badaFo
rOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,2),(Flight_Assn_East_Lo
(index)*0.3048),'pchip');
             Dist_to_climb   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,3),badaForOc
```

```
eanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,2),(Flight_Assn_East_Lo(index
)*0.3048),'pchip');
             if  (Dist_to_climb + Dist_to_descent) - flight.GCDdistance > 0
                 indice = index - 1;
                 if index == 1
                     indice = 1;
                 end
                 Assn_Cruise_FL = Flight_Assn_East_Lo(indice);
                 break;
             end
          end

       else
          Flight_Assn_West_Lo = InputParameters.Value(2)*[2:2:80];
          for index =1: length (Flight_Assn_West_Lo)
             Dist_to_descent   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,3),badaFo
rOceanic(flight.badaForOceanic_Index(aIndex)).nominalDescent(:,2),(Flight_Assn_West_Lo
(index)*0.3048),'pchip');
             Dist_to_climb   =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,3),badaForOc
eanic(flight.badaForOceanic_Index(aIndex)).hiMassClimb(:,2),(Flight_Assn_West_Lo(index
)*0.3048),'pchip');
             if  (Dist_to_climb + Dist_to_descent) - flight.GCDdistance > 0
                 indice = index - 1;
                 if index == 1
                     indice = 1;
                 end
                 Assn_Cruise_FL = Flight_Assn_West_Lo(indice);
                 break;
                 %                          typ_alt_in_m  =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).nominalClimb(:,2),badaForO
ceanic(flight.badaForOceanic_Index(aIndex)).nominalClimb(:,3),Dist_to_climb,'pchip');
                 %                          PreNATalt = floor(typ_alt_in_m
*3.2808/1000)*1000;
                 %                          Assn_Cruise_FL = Calculate_FL(PreNATalt,
Heading, Max_Cruise_FL);
                 %                      end
             end
          end
```

```
        end%    [flight_fuel] = findAircraftIndex (flight);

    end

end
```

estimateTO_mass_Basic.m

```
% Function to estimate the takeoff mass of the aircraft

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% badaForOceanic4ThisFlight=badaForOceanic(aircraftPosition)
% StageLength=flight.gcd
% rand_num=TOM_Random_stream(i)
% DTW_Dist4ThisAircraft=badaForOceanic(aircraftPosition).DTW
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [TOmass] =
estimateTO_mass_Basic(badaForOceanic4ThisFlight,StageLength,rand_num,DTW_Dist4ThisAirc
raft)

if StageLength > max(badaForOceanic4ThisFlight.range)
    StageLength= max(badaForOceanic4ThisFlight.range);

elseif StageLength < min(badaForOceanic4ThisFlight.range)
    StageLength= min(badaForOceanic4ThisFlight.range);
end



TOmassNom = pchip(badaForOceanic4ThisFlight.range, DTW_Dist4ThisAircraft,StageLength);

% Add standard deviation analysis to the TO mass


randomMass = badaForOceanic4ThisFlight.stdTOmass *rand_num;

if randomMass < 0
    randomMass = randomMass/2;            % penalizes fewer negative numbers
end

% TOmass calculation

TOmass = TOmassNom + randomMass;


% Check if initial mass is greater than maximum allowable takeoff mass
% (also in BADA)
```

```
if TOmass > badaForOceanic4ThisFlight.maximumMass * 1000;        % bada data is in
tons so divide by 1000
    TOmass = badaForOceanic4ThisFlight.maximumMass * 1000;
end
end
```

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 InputParameters
% ------------------------------------------------------------------------

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

% -------------------------------------------------------
% Altitudes are in feet. Change them to hundreds of feet.
Raw_Cruise_FL = Raw_Cruise_FL/100;
Max_Cruise_FL = Max_Cruise_FL/100;

% BADA_AC = Dist_FL_Relation{1};
% EAST_Slope = Dist_FL_Relation{5};
% WEST_Slope = Dist_FL_Relation{9};
if InputParameters.Value(2) ==1000

Flight_Assn_East_Lo =
(InputParameters.Value(2)/100)*[3;5;7;9;11;13;15;17;19;21;23;25;27;...
   29;31;33;35;37;39;41];
Flight_Assn_West_Lo =
(InputParameters.Value(2)/100)*[2;4;6;8;10;12;14;16;18;20;22;24;26;...
   28;30;32;34;36;38;40];
Flight_Assn_East_Hi = (InputParameters.Value(2)/100)*[410;450;490;530;570;610];
Flight_Assn_West_Hi = (InputParameters.Value(2)/100)*[400;430;470;510;550;590];
else
   Flight_Assn_East_Lo = (InputParameters.Value(2)/100)*[3:2:82];
Flight_Assn_West_Lo = (InputParameters.Value(2)/100)*[2:2:81];
Flight_Assn_East_Hi = (InputParameters.Value(2)/50)*[410;450;490;530;570;610];
Flight_Assn_West_Hi = (InputParameters.Value(2)/50)*[400;430;470;510;550;590];
end
% -------------------------------------------------------------------------

% 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('Easterly', Heading) == 1
        [no, Row] = min(abs(Flight_Assn_East_Lo - Raw_Cruise_FL));
        Assn_Cruise_FL = Flight_Assn_East_Lo(Row);
    elseif strcmp('Westerly', 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('Easterly', Heading) == 1
        [no, Row] = min(abs(Flight_Assn_East_Hi - Raw_Cruise_FL));
        Assn_Cruise_FL = Flight_Assn_East_Hi(Row);
    elseif strcmp('Westerly', 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
% ------------------------------------------------------------------------
% Change the altitude from 100's of feet to feet.
Assn_Cruise_FL = Assn_Cruise_FL*100;
% _____________________________________________________

Return
```

CalculateRandomFlight.m

```
function [ Climb_Profile,Step_Cruise_Phase,Descent_Profile, LandingMass] =...
    CalculateRandomFlight(flight,TOmass,flightLevel,typicalMach,~, aIndex)
% declare global variables
global badaForOceanic Waypoints Constants InputParameters

% load or define variables
% Aircraft_Index_in_BADA = flight.Aircraft_Index_in_BADA
for i = 1:length(flight)
flight(i).Origin_LatLon(1)= flight(i).originLat;
flight(i).Origin_LatLon(2)= flight(i).originLon;
flight(i).Destination_LatLon(1)= flight(i).destLat;
flight(i).Destination_LatLon(2)= flight(i).destLon;
%     latlondest          = flight.Destination_LatLon;
%     latlonDestination     = flight.Destination_LatLon;
%  latorigin          = flight.originLat;
%  lonorigin          = flight.originLon;
%  latdest          = flight.destLat;
%  londest          = flight.destLon;

% p_stepclimb_inMNPS= rand();
%
latFPRoute = flight(i).Simulation_FP_Lat;
lonFPRoute = flight(i).Simulation_FP_Lon;

Waypoints = [];
Waypoints(:,1) = latFPRoute;
Waypoints(:,2) = lonFPRoute;

GCDdist =
(flight(i).Simulation_Dist_Between_Points_nm)*Constants.Detour;
          %%%facotoring detour factor here,
cumGCDdist_final = cumsum(GCDdist);                                    %...Cumulitive GC
Distance
cumGCDdist_Inv   = sum(GCDdist) - cumsum(GCDdist);                        %...Inverted
Cumulitive GC Distance
%     routeDistance    = cumGCDdist_final(end);                              %...GC
Distance (nm)
GCDdistance = flight(i).GCDdistance * Constants.Detour;
% altitude_m = flightLevel/3.28;           % altitude in meters
% if altitude_m > badaForOceanic(flight.badaForOceanic_Index).nominalClimb(end,3)
%     altitude_m = badaForOceanic(flight.badaForOceanic_Index).nominalClimb(end,3);
```

```
% end
%%
================================================================================
================ %%
% CLIMB SEGMENT
TakeOff_Mass=TOmass;
Climb_Profile = GenerateClimbProfile(flight(i),TakeOff_Mass, aIndex);
FuelClimb1 = Climb_Profile.Total_Fuel_kg;
timeClimb1 = Climb_Profile.Time_For_Climb_hrs*3600; %timeClimb1 is in secs
DistClimb1 = Climb_Profile.Distance_For_Climb_nm;
Climb_Profile.Time_For_Climb = Climb_Profile.Time_For_Climb_hrs*60;
MassAfterClimb = TOmass - FuelClimb1;
[~, tempIndex] = min(abs(cumGCDdist_final - DistClimb1));
TOC_latlon = tempIndex+2;


%% Descent Segment for flights less than 500nm ...Rahul P 3/25

if GCDdistance < (InputParameters.Value(4))

maxTimeDescent =
max(badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,1));   %
finds final time in descent profile (seconds)
maxFuelDescent =
max(badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,4));   %
finds final fuel in descent profile (kg)
altitude_TOD = flightLevel * 0.3048; %assuming that random flight(i)s make atleast 2
or three step climbs. however the fuel consumption in descent is small
fuelUsedDescent = maxFuelDescent -
interp1(badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,3),...

badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,4),altitude_TO
D,'pchip');
timeToDescend = maxTimeDescent -
interp1(badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,3),...

badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,1),altitude_TO
D,'pchip');
distanceToDescend =
interp1(badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,3),bad
aForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,2),altitude_TOD,'
pchip');
```

```
% Flights greater than 500nm. i.e. Check Distance for step Climb
else
%    if 500<(GCDdistance)&&(GCDdistance)<550
%        if flight(i).badaForOceanic_Index(aIndex) == 16
%        disp(num2str(flightLevel));
%        end
%    end
maxTimeDescent =
max(badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,1));   %
finds final time in descent profile (seconds)
maxFuelDescent =
max(badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,4));   %
finds final fuel in descent profile (kg)
altitude_TOD =
badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(1,3); %assuming
that random flight(i)s make atleast 2 or three step climbs. however the fuel
consumption in descent is small
fuelUsedDescent = maxFuelDescent -
interp1(badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,3),...

badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,4),altitude_TO
D,'pchip');
timeToDescend = maxTimeDescent -
interp1(badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,3),...

badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,1),altitude_TO
D,'pchip');
distanceToDescend =
interp1(badaForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,3),bad
aForOceanic(flight(i).badaForOceanic_Index(aIndex)).nominalDescent(:,2),altitude_TOD,'
pchip');
end
%%
================================================================================
================ %%
%% Determining the New Cruise before Top of Descent Profile

if (DistClimb1 + distanceToDescend) < GCDdistance
    [Step_Cruise_Phase] = step_cruise_calc( flight(i),Climb_Profile, latFPRoute,
lonFPRoute,TOC_latlon,DistClimb1, distanceToDescend, GCDdist, flightLevel, FuelClimb1,
timeClimb1, MassAfterClimb, typicalMach, aIndex, GCDdistance);
```

```
    Descent_Profile.fuelUsedDescent = fuelUsedDescent;
    Descent_Profile.timeToDescend = timeToDescend/60;
    Descent_Profile.distanceToDescend = distanceToDescend;
    Descent_Profile.mass = Step_Cruise_Phase.Current_Mass(end);
    LandingMass = Step_Cruise_Phase.Current_Mass(end)- fuelUsedDescent;
elseif (DistClimb1 + distanceToDescend) >= GCDdistance
    [ Step_Cruise_Phase, Descent_Profile, Climb_Profile_new] =
step_cruise_shortrange(flight(i),Climb_Profile, latFPRoute, lonFPRoute,TOC_latlon,
DistClimb1, GCDdist, flightLevel, FuelClimb1, timeClimb1, MassAfterClimb, typicalMach,
aIndex, TakeOff_Mass, maxFuelDescent,maxTimeDescent, GCDdistance);
    LandingMass = Step_Cruise_Phase.Current_Mass(end)- fuelUsedDescent;
%     Descent_Profile = Descent_Profile;
    Climb_Profile.Total_Fuel_kg = Climb_Profile_new.Total_Fuel_kg;
    Climb_Profile.Distance_For_Climb_nm = Climb_Profile_new.Distance_For_Climb_nm;
    Climb_Profile.Time_For_Climb = Climb_Profile_new.Time_For_Climb;
end
end
```

GenerateClimbProfile.m

```
function [Climb_Profile] = GenerateClimbProfile(flight, TakeOff_Mass, aIndex)

global Waypoints BADA_Aircraft_Coef Origin_Airport_Altitude Aircraft_Index_in_BADA

Assn_Cruise_FL        = flight.Assigned_Cruise_FL(aIndex);
Origin_Airport_Altitude = flight.Origin_Airport_Altitude;
Aircraft_Index_in_BADA  = flight.Aircraft_Index_in_BADA(aIndex);

% [~, FP_Dist] = legs(flight.Simulation_FP_Lat, flight.Simulation_FP_Lon);

Waypoints = [];
Waypoints(:,1) = flight.Simulation_FP_Lat;
Waypoints(:,2) = flight.Simulation_FP_Lon;

Climb_Profile = Generate_Climb_Profile(flight,TakeOff_Mass, Assn_Cruise_FL, aIndex);

end
```

Generate_Climb_Profile.m

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

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial_Weight_kg=TakeOff_Mass
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Climb_Profile = Generate_Climb_Profile(flight, Initial_Weight_kg,
Assn_Cruise_FL, aIndex)

% --------------
% DEFINE GLOBAL VARIABLES
% --------------
global BADA_Aircraft_Data rocTable CLIMB_TAS Altitude_Table CLIMB_FUEL_NOM
BADA_Aircraft_Coef
global Time Distance
global Current_Lat Current_Lon i Origin_Airport_Altitude Wind_Proj
global Waypoints Initial_Lat Initial_Lon Constants
global j true_air_speed rate_of_climb fuel_flow

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

%load badaForOceanic

% --------------
% GET AIRCRAFT DATA
% --------------
% load ('inputForClimb.mat')

CLIMB_TAS     = BADA_Aircraft_Data(flight.Aircraft_Index_in_BADA(aIndex)).CLIMB_TAS;
CLIMB_FUEL_NOM =
BADA_Aircraft_Data(flight.Aircraft_Index_in_BADA(aIndex)).CLIMB_FUEL_NOM;
```

```
Altitude_Table = BADA_Aircraft_Data(flight.Aircraft_Index_in_BADA(aIndex)).FL * 100; %
IN FEET
rocTable      =
BADA_Aircraft_Data(flight.Aircraft_Index_in_BADA(aIndex)).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;
Aircraft_Index_in_BADA = flight.Aircraft_Index_in_BADA(aIndex);
```

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

% --------------
% GENERATE CLIMB PROFILE
% --------------
[Climb_Time,Climb_Data] = rk4sys('F_Climb_Coef', Time_Span, yClimb_Initial, 30, [],
aIndex, Assn_Cruise_FL, Aircraft_Index_in_BADA);

% 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)
   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) /
Constants.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
```

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, aIndex, Assn_Cruise_FL,
Aircraft_Index_in_BADA)

% --------------
% DEFINE GLOBAL VARIABLES
% --------------
global Waypoints i  Distance
global Current_Lat Current_Lon Wind_Proj Month
global Initial_Lat Initial_Lon BADA_Aircraft_Coef
global Constants

global j true_air_speed rate_of_climb fuel_flow azimuth_vec

% AircraftInBadaList={BADA_Aircraft_Data.Aircraft_Name};
% AircraftName=badaForOceanic(Aircraft_Index).aircraftName;
% [~,Aircraft_Index_Current]=intersect(AircraftInBadaList,AircraftName);


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 Constants.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,:)) /
Constants.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 Constants.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 Constants.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(Current_Altitude, y(2), aIndex,
Aircraft_Index_in_BADA);
vtas       = vtas_kts * knotsTofps + Wind_Proj(i,:)*knotsTofps;   % SPEED IS TRUE
AIRSPEED (TAS) IN FPS
rateOfClimb = ROC_Calculator(Current_Altitude, vtas_kts, y(2), aIndex,
Aircraft_Index_in_BADA) * pmTops;                     % IN FEET PER SECOND UNITS
fuelFlow    = FuelFlow_Calculator_Climb(Current_Altitude, vtas_kts, aIndex,
Aircraft_Index_in_BADA)  * pmTops;               % KILOGRAMS PER SECOND

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

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


% --------------
% GET ODE VARIABLES
% --------------
% ------------------------------------------------------------------------
% -------------------------VTAS-------------------------------------------
% ------------------------------------------------------------------------

% 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 * Current_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_in_BADA).AC_Type.eng_type;
V_cl1         = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Procedures.V_cl1;
V_cl2         = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Procedures.V_cl2;
M_cl          = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Procedures.M_cl;
m_ref         = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Mass.m_ref * 1000; % tonnes
to kg
V_stall_TO_ref = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Aero.V_stall_TO;

% ACTUAL V_STALL_TO FOR AC_Mass
AC_Mass=y(2);
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
Altitude=Current_Altitude;
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
vtas_kts=VTAS;
% ------------------------------------------------------------------------
vtas       = vtas_kts * knotsTofps + Wind_Proj(i,:)*knotsTofps;   % SPEED IS TRUE
AIRSPEED (TAS) IN FPS
rateOfClimb = ROC_Calculator(Current_Altitude, vtas_kts, y(2), aIndex,
Aircraft_Index_in_BADA) * pmTops;                      % IN FEET PER SECOND UNITS
fuelFlow    = FuelFlow_Calculator_Climb(Current_Altitude, vtas_kts, aIndex,
Aircraft_Index_in_BADA)  * pmTops;               % KILOGRAMS PER SECOND
% rateOfClimb = pchip(Altitude_Table,rocTable,Current_Altitude) *
pmTops;                                    % IN FEET PER SECOND UNITS
% fuelFlow    = pchip (Altitude_Table,CLIMB_FUEL_NOM,Current_Altitude)  * pmTops;
% ------------------------------------------------------------------------

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
```

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(Altitude, AC_Mass, aIndex,
Aircraft_Index_in_BADA)
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_in_BADA).AC_Type.eng_type;
V_cl1         = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Procedures.V_cl1;
V_cl2         = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Procedures.V_cl2;
M_cl          = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Procedures.M_cl;
m_ref         = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Mass.m_ref * 1000; % tonnes
to kg
V_stall_TO_ref = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).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
% -------------------------------------------------------------------------
```

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(Altitude, VTAS, AC_Mass, aIndex,
Aircraft_Index_in_BADA)

global 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_in_BADA).AC_Type.eng_type;

S = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).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_in_BADA).Aero.C_D0_CR;
C_D2_CR = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).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_in_BADA).Thrust.C_Tc1;
C_Tc2 = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Thrust.C_Tc2;
C_Tc3 = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Thrust.C_Tc3;
C_Tc4 = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Thrust.C_Tc4;
C_Tc5 = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).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_in_BADA).Flight_Env.h_MO;
h_max = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Flight_Env.h_max;
G_t = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Flight_Env.G_t;
C_Tc4 = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Thrust.C_Tc4;
G_w = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Mass.G_w;
m_min = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Mass.m_min*1000;
m_max = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).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>
```

FuelFlow_Calculator.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(Altitude, VTAS, aIndex,
Aircraft_Index_in_BADA)

global 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_in_BADA).AC_Type.eng_type;

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

```
C_Tc4 = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Thrust.C_Tc4;
C_Tc5 = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).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_in_BADA).Fuel.C_f1;
C_f2 = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).Fuel.C_f2;
C_fcr = BADA_Aircraft_Coef(Aircraft_Index_in_BADA).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
```

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, aIndex, Assn_Cruise_FL,
Aircraft_Index_in_BADA)

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;
ZeroROC = 0;%%%Breaks the while loop if ROC is Zero.

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, aIndex,Assn_Cruise_FL, Aircraft_Index_in_BADA)';
        ymid = y(i,:)+k1.*hh/2;
        k2 = feval(dydt,(tt+hh/2),ymid,false, aIndex,
Assn_Cruise_FL,Aircraft_Index_in_BADA)';
        ymid = y(i,:)+k2.*hh/2;
        k3 = feval(dydt,(tt+hh/2),ymid,false, aIndex,
Assn_Cruise_FL,Aircraft_Index_in_BADA)';
        yend = y(i,:)+k3.*hh;
        k4 = feval(dydt,(tt+hh),yend,true, aIndex,
Assn_Cruise_FL,Aircraft_Index_in_BADA)';
        phi = (k1+2*(k2+k3)+k4)/6;
        y(i+1,:) = y(i,:)+phi*hh;
        tt = tt+hh;
        if (y(i+1,1) - y(i,1)) < 100%%% if climb rate is zero/less than 100 ft, no
point in climbing higher
            ZeroROC = 1;break;
        end
        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 ZeroROC,break,end%if tt>=tf,break,end

end
```

step_cruise_calc.m

```
% Code for (DistClimb1 + distanceToDescend) < sum(GCDdist)
% Step Climb according to Arman's codes with check distance of 500nm
% Written and Edited by Rahul P



function [Step_Cruise_Phase] = step_cruise_calc(flight,Climb_Profile, latFPRoute,
lonFPRoute,TOC_latlon, DistClimb1, distanceToDescend, GCDdist, flightLevel,
FuelClimb1, timeClimb1, MassAfterClimb, typicalMach, aIndex, GCDdistance)
global badaForOceanic  Constants InputParameters
%%
cumGCDdist_Inv   = sum(GCDdist) - cumsum(GCDdist);
[~, tempIndex] = min(abs(cumGCDdist_Inv - distanceToDescend));    %...Find TOD Waypoint
TOD_latlon = tempIndex+1;
latFPRoute_cruise = [Climb_Profile.Latitude_Pts(end);
latFPRoute(TOC_latlon:TOD_latlon)];             %...Find The Cruise Segment
(Latitudes)
lonFPRoute_cruise =
[Climb_Profile.Longitude_Pts(end);lonFPRoute(TOC_latlon:TOD_latlon)];             %...
Find The Cruise Segment (Longitudes)
% %    [~, Cruise_GCDdist] = legs(latFPRoute_cruise,lonFPRoute_cruise); %...Azimuth and
Distance Between Waypoints of The Cruise Segment
[~, Cruise_GCDdist] = legs(latFPRoute_cruise,lonFPRoute_cruise);
% %%%%%%%%%%%%%%%%%Increase the GCD dist by (detour_factor) times for
cruise%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Cruise_GCDdist = Cruise_GCDdist;
%     %%%%%%%%%%%%%%Find Lat/Lon for first 1000 nm of cruise%%%%%%%%%%%%%%%%%%%
cumGCD_dist = cumsum(Cruise_GCDdist);   %...Cumulitive GC Cruise Distance
current_fl = flightLevel;

%
FL_increment = InputParameters.Value(3);
%%

Cruise_Phase = (GCDdistance)-(DistClimb1 + distanceToDescend );
Check_Distance_nm= InputParameters.Value(4);

% Define the Number of segments
Segment=floor(Cruise_Phase/Check_Distance_nm);

dist2travel=zeros(1,Segment+1);
```

```
dist2travel(1,1:Segment)=Check_Distance_nm;
dist2travel(1,end)=Cruise_Phase-(Segment*Check_Distance_nm);
Cum_dist2travel=cumsum(dist2travel);

Cruise_before_TOD_Coordination=cell([1,Segment+1]);

tempIndex=zeros(1,numel(dist2travel)); % Test from rahul
% Find the closest waypoints for these Investigation p
for i=1:numel(dist2travel) % Test from rahul

    [~, tempIndex(i)] = min(abs(cumGCD_dist - Cum_dist2travel(i)));

end

% Gather the whole Segments
Cruise_Before_TOD_Index=[1 tempIndex ];

% Define the Segments Coordinations
for i=1:size(Cruise_Before_TOD_Index,2)-1


Cruise_before_TOD_Coordination{i}(:,1)=latFPRoute_cruise(Cruise_Before_TOD_Index(i):Cr
uise_Before_TOD_Index(i+1));

Cruise_before_TOD_Coordination{i}(:,2)=lonFPRoute_cruise(Cruise_Before_TOD_Index(i):Cr
uise_Before_TOD_Index(i+1));

end
% if tempIndex < 2
%        disp(num2str(flight.Aircraft_Index_in_BADA));
%
% end
% create Step_Cruise_Phase struct
%    Step_Cruise_Phase = struct([]);
%    Descent_Profile = struct([]);

% Import Climb Profile as the first segment of Step_Cruise_Phase
Step_Cruise_Phase.Distance(1)=DistClimb1;
Step_Cruise_Phase.Fuel(1)=FuelClimb1;
Step_Cruise_Phase.Time(1)=timeClimb1/60; % time in min
Step_Cruise_Phase.Wind(1)=0;
Step_Cruise_Phase.Current_Mass(1)=MassAfterClimb ;
```

```
Step_Cruise_Phase.Flight_Level(1)=flightLevel ;

% Define the Cruise profile (Time, Fuel, Distance) of Before TOD Trajectory
for i=1:size(Cruise_Before_TOD_Index,2)-1

   [FuelUsedStep_Cruise_Phase,TravelTimeStep_Cruise_Phase,Wind_speed_before_TOD,
Dist_travelled_before_TOD] =  ...

fuel_TT_calculator_function_cruise(Step_Cruise_Phase.Current_Mass(i),dist2travel(i),St
ep_Cruise_Phase.Flight_Level(i),typicalMach,...

Cruise_before_TOD_Coordination{i}(:,1),Cruise_before_TOD_Coordination{i}(:,2),2,0,flig
ht.badaForOceanic_Index(aIndex));



   Step_Cruise_Phase.Distance(i+1)= Dist_travelled_before_TOD;
   Step_Cruise_Phase.Fuel(i+1) = FuelUsedStep_Cruise_Phase;
   Step_Cruise_Phase.Time (i+1)=TravelTimeStep_Cruise_Phase;
   Step_Cruise_Phase.Wind(i+1)=Wind_speed_before_TOD;
   Step_Cruise_Phase.Current_Mass(i+1)=Step_Cruise_Phase.Current_Mass(i)-
Step_Cruise_Phase.Fuel(i+1) ;

   % check for feasible climbs At Investigation Points

   maxOperationalAltitude =
interp1(badaForOceanic(flight.badaForOceanic_Index(aIndex)).operationalAltMass,badaFor
Oceanic(flight.badaForOceanic_Index(aIndex)).maxOperationalAltitude,Step_Cruise_Phase.
Current_Mass(i+1));
   if isnan(maxOperationalAltitude)
      maxOperationalAltitude =
pchip(badaForOceanic(flight.badaForOceanic_Index(aIndex)).operationalAltMass,badaForOc
eanic(flight.badaForOceanic_Index(aIndex)).maxOperationalAltitude,Step_Cruise_Phase.Cu
rrent_Mass(i+1));
   end
   temp_FL = floor(maxOperationalAltitude/1000)*1000;
   Step_Cruise_Phase.Flight_Level(i+1) = temp_FL - mod((temp_FL -
current_fl),FL_increment);

   % Calculate the profile of feasible Climbs

   if Step_Cruise_Phase.Flight_Level(i+1)>Step_Cruise_Phase.Flight_Level(i)
```

```
        [fuelUsedInClimb,timeToClimb,~,distanceInClimb] = ...

fuel_TT_calculator_function_climb(Step_Cruise_Phase.Current_Mass(i+1),Step_Cruise_Phas
e.Flight_Level(i+1),Step_Cruise_Phase.Flight_Level(i+1),...

typicalMach,Step_Cruise_Phase.Wind(i+1),flight.badaForOceanic_Index(aIndex));

        % ADD the Climb profile (Time, Fuel, Distance) to precceding Cruise Profile

        Step_Cruise_Phase.Distance
(i+1)=Step_Cruise_Phase.Distance(i+1)+distanceInClimb;
        Step_Cruise_Phase.Fuel(i+1)=Step_Cruise_Phase.Fuel(i+1)+fuelUsedInClimb;
        Step_Cruise_Phase.Time(i+1)=Step_Cruise_Phase.Time(i+1)+(timeToClimb/60);
        Step_Cruise_Phase.Current_Mass(i+1)=Step_Cruise_Phase.Current_Mass(i+1)-
fuelUsedInClimb ;
    end

end
end
```

fuel_TT_calculator_function_cruise.m

```
% Program to estimate fuel burn and travel time for an aircraft
% flying a long haul route
function
[totalFuelUsed,totalTravelTime,averageWindSpeed,distanceTraveledCruise,lrsTime,lrsDist
] = ...

fuel_TT_calculator_function_cruise(initialMass,distanceToCover,cruiseAltitude,machNumb
er,Lat,Lon,InterpWind,T_FL_wind,BADAAircaftType)

global badaForOceanic atmosphere
% load('D:\GOM_Git\GOM_GUI\GOM_GUI\DATA\GOM\BADA_Aircraft_Coef_badaForOceanic.mat')
% Inputs:
% badaForOceanic = BADA struct file with aerodynamic coefficients for each aircraft
% cruiseAltitude = desired cruise altitude (feet)
% initialMass = initial aircraft mass (kg)
% routeFlown = route to be studied (from FAA routes for now)

% Outputs

% fuelUsed = total fuel used in trip (kg)
% travelTime = total travel time in trip (minutes)
if distanceToCover <=0
    totalFuelUsed = 0;
    totalTravelTime = 0;
    averageWindSpeed = 0;
    distanceTraveledCruise= 0;
    lrsTime = 0;
    lrsDist =0;
else

    % _________ Load the files with aerodynamic data _____________

    %load badaForOceanic

    % Constants

    k_ft_to_m = 1/3.28;
    mps_to_knots = 3600/1852;   % meters per second to knots
    g = 9.81;   % gravity constant (m/s-s)
    deltaTime = 0.5;                % initial cruise delta time (minutes)
```

```
    % Detect routes between the OD pairs

    % routeDistance = distanceToCover;    % extracts the information for a given route

    % Select aircraft index from BADA data set

    %     [aircraftPosition] = findAircraft(aircraftType);

    % define badaData

    % badaData = badaForOceanic;         % change in variable name
    %clear badaForOceanic                   % clear the old variable

    % Do the climb calculation for travel time and fuel used

    % Nominal climb profile contains 4 columns:
    % 1 = time (seconds)
    % 2 = distance traveled (nm)
    % 3 = altitude (m)
    % 4 = fuel used (kg)

    %cruiseDistance = distanceToCover;       % distance in cruise in nm
    % averageWindSpeed = interp1(windAltitudeVector,averageWind,cruiseAltitude);

    if InterpWind == 0
        averageWindSpeed = T_FL_wind;
    elseif InterpWind == 1
        averageWindSpeed = getwindvect_1(cruiseAltitude,Lat,Lon);
    elseif InterpWind ==2
        averageWindSpeed = getwindvect(cruiseAltitude,Lat,Lon);
    else
        error('Track is not assigned: Check NAT_Tracks.mat weather the tracks exists or
not')
    end
    % Calculate the true airspeed given mach number
    cruiseAltitude_m = cruiseAltitude*k_ft_to_m;      % in meters
    [speedOfSound,rho_Density] = isan(cruiseAltitude_m);         % altitude in meters
    cruiseSpeed_mps = machNumber * speedOfSound;
    cruiseSpeed_knots = cruiseSpeed_mps* mps_to_knots;        % in knots
    groundSpeed_knots = ((cruiseSpeed_knots + averageWindSpeed));              %  in
knots
    % Start Cruise integration algorithm calling the BADA function to estimate
```

```
    % fuel burn

    actualFuel= 0;                                  % initial value of Fuel state in
cruise mode (kg)
    mass= initialMass;                              % starting mass for cruise mode (kg)
    actualDistance = 0;                            % initial value of distance traveled
in cruise mode (m)
    % cruiseTimeCount(1) =0;
    i = 0  ;                                                % index used in while loop
for numerical integration

    % Solve ODEs here using simple Euler first-order method (crude but thrustworthy)

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    aircraftPosition = BADAAircaftType(1);
    % [aircraftPosition] = findAircraft(aircraftType);
    % Select the aircraft parameters based on the position identified in the
    % struct file
    S_WingArea = badaForOceanic(aircraftPosition).wingArea;      % wing area
    cdo = badaForOceanic(aircraftPosition).config.cdo;       % BADA 3.13.1 cruise
    cd2 = badaForOceanic(aircraftPosition).config.cd2;       % BADA 3.13.1 cruise
    % Thrust coefficients
    cf1 = badaForOceanic(aircraftPosition).tsfc_1;
    cf2 = badaForOceanic(aircraftPosition).tsfc_2;                  % bada 3.13.1
    cfr =badaForOceanic(aircraftPosition).cruiseCorrection;  % bada 3.13
    % Lines 105-110 added by Rahul P. According to BADA 3.13
     if strcmp(badaForOceanic(aircraftPosition).aircrafType, 'Jet') == 1
        nu = cf1 * (1 + cruiseSpeed_knots ./ cf2);
     elseif strcmp(badaForOceanic(aircraftPosition).aircrafType, 'Turboprop') == 1
        nu = cf1 * (1 - cruiseSpeed_knots ./ cf2) * ( cruiseSpeed_knots /1000);
     end
%     nu = cf1 * (1 + cruiseSpeed_knots ./ cf2);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % CrossIndicator = 1;

    while actualDistance < distanceToCover              % iterate until the aircraft
reaches final cruise condition

        %     if CrossIndicator && (actualDistance>=DistFromEntry230W)
```

```
        %        Cross30WTime = i*deltaTime;
        %        CrossIndicator = 0;
        %    end
        % calculate fuel and aerodynamic values
        cl = 2*mass*g / (rho_Density * S_WingArea * cruiseSpeed_mps^2);
        cd = cdo + cd2 * cl^2;     % drag coefficient
        drag = 1/2 * cd * S_WingArea* cruiseSpeed_mps^2 *rho_Density;
        fuelFlow = nu .* drag/1000 * cfr;  %Calculations for fuel flow (kg/min)

        %    [~,fuelFlow] =
cruiseCalculations_BADA39(cruiseSpeed,cruiseAltitude,mass,aircraftType,badaData);

        % update accumulators
        actualFuel = actualFuel + fuelFlow * deltaTime;                        % fuel
state (kg)
        mass= mass - fuelFlow * deltaTime ;                                    % Distance
traveled along the flight path (m)

        actualDistance = actualDistance +groundSpeed_knots* deltaTime/60 ;     %
distance traveled  (nm)
        % speed is in nm/min
        i=i+1;                                        % update index to keep vector for
state variables

    end

    %% Correct for the distance overflown in the last segment in the cruise

    if actualDistance >distanceToCover
        DiscountRatio = (actualDistance -distanceToCover )/(groundSpeed_knots
*deltaTime/60);
    else
        DiscountRatio = 0;
    end

    lrsTime = deltaTime*[0:i];
    lrsTime(i+1) = deltaTime - DiscountRatio*deltaTime;
    lrsDist = groundSpeed_knots*lrsTime./60;

    cl = 2*mass*g / (rho_Density * S_WingArea * cruiseSpeed_mps^2);
    cd = cdo + cd2 * cl^2;     % drag coefficient
    drag = 1/2 * cd * S_WingArea* cruiseSpeed_mps^2 *rho_Density;
```

```
    fuelFlow = nu .* drag/1000 * cfr;  %Calculations for fuel flow (kg/min)


    totalTravelTime  = (i*deltaTime - DiscountRatio*deltaTime) ;               %
minutes
    totalFuelUsed = actualFuel - DiscountRatio *fuelFlow * deltaTime;
    % finalMass = massEndOfCruiseTemp +DiscountRatio *fuelFlow *
deltaTime;            % corrected mass
    distanceTraveledCruise = actualDistance - DiscountRatio *groundSpeed_knots*
deltaTime/60;

end

return;
```

getwindvect.m

```
function [wind_vect] = getwindvect(altitude,lat,lon)
global Wind
% load Wind
%calculate the correct wind vectors
altlen = length(altitude);
[Alt_mb] = changeAltTOmillibars(altitude);
for k = 1:length(Alt_mb)
    [val AltInd(k)] = min(abs(Wind.Level - Alt_mb(k)));
end
tempind = find(lon<0);
lon(tempind) = lon(tempind) +360;
tempind = find(lon>357.5);
lon(tempind) = 0;

% for i =1:numel(lat)
%     Uwind(i) = interp3(Wind.Lat,Wind.Lon,Wind.Level,Wind.Uwnd,lat(i),lon(i),Alt_mb);
%     Vwind(i) = interp3(Wind.Lat,Wind.Lon,Wind.Level,Wind.Vwnd,lat(i),lon(i),Alt_mb);
% end
%
% Uwind = Uwind';
% Vwind = Vwind';

Alt_mb_vect = Alt_mb*ones(numel(lat),1);

Uwind = interp3(Wind.Lat,Wind.Lon,Wind.Level,Wind.Uwnd,lat,lon,Alt_mb_vect);
Vwind = interp3(Wind.Lat,Wind.Lon,Wind.Level,Wind.Vwnd,lat,lon,Alt_mb_vect);

%        Uwind = interp3(Wind.Lat,Wind.Lon,Wind.Level,Wind.Uwnd,lat,lon,Alt_mb);
%        Vwind = interp3(Wind.Lat,Wind.Lon,Wind.Level,Wind.Vwnd,lat,lon,Alt_mb);

%        1
% %     LatInd = round(37 - (lat/2.5));
% %     LonInd = floor((lon/2.5) + 1);
% %     if altlen > 1
% %        for l = 1:altlen
% %        Uwind(l,1) = Wind.Uwnd(LonInd(l),LatInd(l),AltInd(l));
% %        Vwind(l,1) = Wind.Vwnd(LonInd(l),LatInd(l),AltInd(l));
% %        end
% %     else
% %
% %     Uwind = diag(Wind.Uwnd(LonInd,LatInd,AltInd));
```

```
% %     Vwind = diag(Wind.Vwnd(LonInd,LatInd,AltInd));
% %
% %     end

if length(lat) < 2
    wind_vect = 0;
    return
end
[Course, dist] = legs(lat,lon);
dist = dist';

Direction_Vector = [cosd(Course), sind(Course)];
Wind_Vector = [Vwind(1:end-1), Uwind(1:end-1)];
% Pressure
% Geopotential_Alt

% 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
for w = 1:length(Course)
    wind_proj(w) = dot(Direction_Vector(w,:), Wind_Vector(w,:)) /
(norm(Direction_Vector(w,:)));
end

wind_vect = 1.9438*sum(wind_proj.*dist)/sum(dist);     % 1.9438 is the conversion
factor for M/s to knots.

end
```

getwindvect_1.m

```
function [wind_vect] = getwindvect(altitude,lat,lon)
global Wind
% load Wind
%calculate the correct wind vectors
altlen = length(altitude);
[Alt_mb] = changeAltTOmillibars(altitude);
for k = 1:length(Alt_mb)
    [val AltInd(k)] = min(abs(Wind.Level - Alt_mb(k)));
end
tempind = find(lon<0);
lon(tempind) = lon(tempind) +360;

% %     parfor i =1:numel(lat)
% %         Uwind(i) =
interp3(Wind.Lat,Wind.Lon,Wind.Level,Wind.Uwnd,lat(i),lon(i),Alt_mb);
% %         Vwind(i) =
interp3(Wind.Lat,Wind.Lon,Wind.Level,Wind.Vwnd,lat(i),lon(i),Alt_mb);
% %     end
% %         Uwind = Uwind';
% %         Vwind = Vwind';
LatInd = round(37 - (lat/2.5));
LonInd = floor((lon/2.5) + 1);
if altlen > 1
    for l = 1:altlen
        Uwind(l,1) = Wind.Uwnd(LonInd(l),LatInd(l),AltInd(l));
        Vwind(l,1) = Wind.Vwnd(LonInd(l),LatInd(l),AltInd(l));
    end
else

    Uwind = diag(Wind.Uwnd(LonInd,LatInd,AltInd));
    Vwind = diag(Wind.Vwnd(LonInd,LatInd,AltInd));

end

if length(lat) < 2
    wind_vect = 0;
    return
end
[Course dist] = legs(lat,lon);
dist = dist';
```

```
Direction_Vector = [cosd(Course), sind(Course)];
Wind_Vector = [Vwind(1:end-1), Uwind(1:end-1)];
% Pressure
% Geopotential_Alt

% 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
for w = 1:length(Course)
    wind_proj(w) = dot(Direction_Vector(w,:), Wind_Vector(w,:)) /
(norm(Direction_Vector(w,:)));
end

wind_vect = 1.9438*sum(wind_proj.*dist)/sum(dist);     % 1.9438 is the conversion
factor for M/s to knots.

end
```

isan.m

```
% Computes atmospheric values

% altitude in meters
% speed of sound in m/s
% density = kg/cu. meter

function [a,rho,temp] = isan(alt)

global atmosphere;            % loads ISA atmospheric tables

h = atmosphere(:,1);          % vector with values of altitude
t = atmosphere(:,2);          % vector with values of temperature
r = atmosphere(:,3);          % vector with values of density
a_alt = atmosphere(:,4);       % vector with values of speed

rho    = pchip(h,r,alt);   % interpolates to get density
a      = pchip(h,a_alt,alt);   % gets speed of sound
temp   = pchip(h,t,alt);   % gets temperature
end
```

fuel_TT_calculator_function_climb.m

```
% Program to estimate fuel burn and travel time for an aircraft
% flying a long haul route

function [fuelUsedInClimb,timeToClimb,averageFuelBurn_kg_min,distanceInClimb] =
fuel_TT_calculator_function_climb(initialMass,initialAltitude_ft,finalAltitude_ft,mach
Number,averageWind,badaForOceanic_Index)

% Inputs:
% badaForOceanic = BADA struct file with aerodynamic coefficients for each aircraft
% cruiseAltitude = desired cruise altitude (feet)
% initialMass = initial aircraft mass (kg)
% routeFlown = route to be studied (from FAA routes for now)

% Outputs

% fuelUsed = total fuel used in trip (kg)
% travelTime = total travel time in trip (minutes)

% _________ Load the files with aerodynamic data ____________

% Constants

k_ft_to_m = 1/3.28;
mps_to_knots = 3600/1852;   % meters per second to knots
deltaTime = 5;                    % step size for climb analysis (seconds)

initialAltitude_m = initialAltitude_ft * k_ft_to_m;
finalAltitude_m   = finalAltitude_ft * k_ft_to_m;

% Do the climb calculation for travel time and fuel used

% Nominal climb profile contains 4 columns:
% 1 = time (seconds)
% 2 = distance traveled (nm)
% 3 = altitude (m)
% 4 = fuel used (kg)

% Start Cruise integration algorithm calling the BADA function to estimate
% fuel burn
```

```
actualFuel(1) = 0;                              % initial value of Fuel state in
climbmode (kg)
mass(1) = initialMass;                          % starting mass for cruise mode (kg)
actualDistance(1)   = 0;                        % initial value of distance traveled
in cruise mode (m)
climbTimeCount(1)   = 0;
actualAltitude_m(1) = initialAltitude_m;        % initial altitude (meters)
ROC(1) = 0;

i = 1;                                          % index used in while loop for
numerical integration

% Solve ODEs here using simple Euler first-order method (crude but thrustworthy)
BADAAircaftType = badaForOceanic_Index;
% aircraftPosition= BADAAircaftType; %Added by Rahul P
FlightCounter = 0;
while actualAltitude_m <= finalAltitude_m             % iterate until the aircraft
reaches the final altitude

   % calculate fuel and aerodynamic values

   % averageWindSpeed =
interp1(windAltitudeVector,averageWind,actualAltitude_m(i)*3.28);      % gets wind
speed interpolated
   averageWindSpeed = averageWind;

   [speedOfSound,~,~] = isan(actualAltitude_m);
   climbSpeed = machNumber * speedOfSound;          % in m/s
   climbSpeedKnots = climbSpeed * 3600/1852;        % climb speed in Knots

   %% @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
   % Calculate the rate of climb
   actualAltitude_ft = actualAltitude_m * 3.28;
   [ROC,~,fuelFlow,~,~] = climbCalculations_BADA39(climbSpeedKnots, actualAltitude_ft,
mass, badaForOceanic_Index);
   %% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     actualFuel = zeros();
%     mass = zeros();
%     climbTimeCount = zeros();
%     actualDistance = zeros();
%     actualAltitude_m = zeros();
```

```
   actualFuel = actualFuel + fuelFlow/60 * deltaTime;                  % fuel state
(kg)
   mass = mass - fuelFlow/60 * deltaTime ;                              % mass state
variable
%    climbTimeCount(i+1) = climbTimeCount(i) + deltaTime;                        %
counts climb time (seconds)

   actualDistance   = actualDistance + climbSpeed * deltaTime +
(averageWindSpeed/mps_to_knots) * deltaTime;          % distance traveled  (meters)
   actualAltitude_m = actualAltitude_m + ROC *deltaTime;                % new altitude
(meters)

   climbTimeCount = climbTimeCount + deltaTime;                         % update
counter for time (to measure when simulation stops)

i=i+1;
% update index to keep vector for state variables

   %           FLight Counter
   %        if mod(FlightCounter, 50) == 0
   %           disp(['Iterations: ', num2str(FlightCounter)]);
   %        end
   FlightCounter = FlightCounter + 1; % Counts the number of instances in loop


end

% write the final time
timeToClimb = max(climbTimeCount);                    % time when climb simulation
stops (seconds)
fuelUsedInClimb = max(actualFuel);                        % kilograms
distanceInClimb = max(actualDistance)/1852;           % in nm
% massEndOfClimb = min(mass);

% Collect travel time statistics
averageFuelBurn_kg_min = fuelUsedInClimb/ (timeToClimb/60);             % in
kilograms/min
% averageFuelBurn_lbhr = fuelUsedInClimb/ (timeToClimb) * 3600
*2.2;                              % in pounds /hrend
```

climbCalculations_BADA39.m

```
% Calculations of drag and fuel burn
% in the cruise condition. Uses the BADA 3.8 model
% Programmer : Toni Trani

% inputs

% tas = knots
% altitude = feet
% mass = kilograms

% Outputs

% drag = Newtons
% fuelFlow = kg/minute
% cl = dimensionless
% cd = dimensionless

function [ROC,drag,fuelFlow,cl,cd] = climbCalculations_BADA39(tas, altitude_ft, mass,
badaForOceanic_Index, BADAAircaftType)

global badaForOceanic
% Select the aircraft parameters based on the position identified in the
% struct file

% Select aircraft index from BADA data set
badaForOceanic_Index = badaForOceanic_Index(1);
BADAAircaftType = badaForOceanic_Index;
aircraftPosition= BADAAircaftType ;
% [aircraftPosition] = findAircraft(aircraftType);
% Select the aircraft parameters based on the position identified in the
% struct file
S = badaForOceanic(aircraftPosition).wingArea;      % wing area
cdo = badaForOceanic(aircraftPosition).config.cdo;       % BADA 3.9 cruise
cd2 = badaForOceanic(aircraftPosition).config.cd2;       % BADA 3.8.1 cruise
% Thrust coefficients
cf1 = badaForOceanic(aircraftPosition).tsfc_1;
cf2 = badaForOceanic(aircraftPosition).tsfc_2;                  % bada 3.9
cfr =badaForOceanic(aircraftPosition).cruiseCorrection;

% Constants and conversion factors
g = 9.81;   % gravity constant (m/s-s)
```

```
kmstoknots = 3600/1852;

% convert the appropriate units

altitude_m = altitude_ft / 3.28;      % in meters
V = tas / kmstoknots;              % speed in m/s needed for the drag calculations

% Call ISA function

[speedOfSound,rho] = isa_function(altitude_m);          % in meters

% Calculations

cl = 2*mass*g / (rho * S * V^2);          % dimensionless

cd = cdo + cd2 * cl^2;                      % drag coefficient

drag = 1/2 * cd * S * V^2 * rho;          % in Newtons

% Estimate the thrust produced

[maxThrust] = maximumThrust_function(tas,altitude_ft,badaForOceanic_Index);     % in
Newtons

% Calculations for fuel flow (kg/min)
% Convert speed to true airspeed Vtas

Vtas = V * kmstoknots;
if strcmp(badaForOceanic(aircraftPosition).aircrafType, 'Jet') == 1
   nu = cf1 * (1 + Vtas ./ cf2);
elseif strcmp(badaForOceanic(aircraftPosition).aircrafType, 'Turboprop') == 1
   nu = cf1 * (1 - Vtas ./ cf2) * ( Vtas /1000);
end
% nu = cf1 * (1 + Vtas ./ cf2);
fuelFlow = nu .* maxThrust/1000 * cfr;

ROC = (maxThrust - drag) * V / (mass*g);          % rate of climb in m/second
end
```

maximumThrust_function.m

```
% Function to determine the maximum altitude possible by the aircraft

% Inputs: acftType and altitude
% Output: max climb thrust (N)

function [maxThrust] = maximumThrust_function(tas,altitude_ft,badaForOceanic_Index)

% _________ Load the files with aerodynamic data ____________

global badaForOceanic
kmstoknots = 3600/1852;
V = tas/ kmstoknots;
Vtas = V*kmstoknots ;
% Find the maximum thrust according to BADA
BADAAircaftType = badaForOceanic_Index;
aircraftPosition= BADAAircaftType;
if strcmp(badaForOceanic(aircraftPosition).aircrafType, 'Jet') == 1
maxThrustperEngine = badaForOceanic(aircraftPosition).Ctc_1 * (1- ...
                altitude_ft/badaForOceanic(aircraftPosition).Ctc_2 +
badaForOceanic(aircraftPosition).Ctc_3 ...
                *altitude_ft ^2) ;

elseif strcmp(badaForOceanic(aircraftPosition).aircrafType, 'Turboprop') == 1
maxThrustperEngine = (badaForOceanic(aircraftPosition).Ctc_1/Vtas) * (1- ...
                altitude_ft/badaForOceanic(aircraftPosition).Ctc_2) +
(badaForOceanic(aircraftPosition).Ctc_3) ;
end

maxThrust = maxThrustperEngine;     %*
badaForOceanic(aircraftPosition).numberOfEngines;
end
```

Taxi_Profile.m

```
% THE FOLLOWING CODE CALCULATES TAXI-OUT AND TAXI-IN TIMES FOR THE
% AIRCRAFT FLEET OF THE ORIGIN AND DESTINATION AIRPORT RESPECTIVELY USING A
% REGRESSION ANALYSIS BASED ON THE ASPM DATA OBTAINED FOR 77 AIRPORTS FOR THE YEAR
2014

% CODED AND EDITED BY Rahul_P


function Taxi_Profile(Install_Dir, Save_Dir, Case_Year)

global aircraftArray

Case_Year_Str = num2str(Case_Year);




disp('Running Taxi Profile Calculator..');
disp('Loading Flight Data, please wait...');
disp(' ')
load ([Save_Dir, '\flight_2_', Case_Year_Str, '.mat'])
load ('nextgenacft.mat')
disp('Data Loaded')
disp(' ')
CaseYear = '2014';
%-----------------------------------------------------------------
%Average Taxi Time for ASPM 77 Airports
%-----------------------------------------------------------------

%-----------------------------------------------------------------
%Step 1: Read the APM-Report for Taxi Time
%-----------------------------------------------------------------
if str2double(CaseYear) < 1998
   APMYear = '1998';
elseif str2double(CaseYear) > 2016
   APMYear = '2016';
else
   APMYear = CaseYear;
end

% Alter according to APM-Report folder
```

```
File_Name = [Install_Dir, '\Passenger\Fuel_Consumption_And_Emissions\APM-Report-',
APMYear, '.txt'];

fileID = fopen(File_Name);

% Skip first 8 lines
LinetoDelete = fgetl(fileID);
LinetoDelete = fgetl(fileID);
LinetoDelete = fgetl(fileID);
LinetoDelete = fgetl(fileID);
LinetoDelete = fgetl(fileID);
LinetoDelete = fgetl(fileID);
LinetoDelete = fgetl(fileID);
LinetoDelete = fgetl(fileID);

APM_Report_Temp = textscan(fileID,'%s%f%f%f%f%f%f%f%f%f%s%f%f%f%f%f%f');

clear LinetoDelete

% Field List:
% Field 1: Airport Code
% Field 2: Number of Departures
% Field 15: Average Taxi Out Time
% Field 16: Number of Arrivals
% Field 18: Average Taxi In Time

APM_Airport_Code = APM_Report_Temp{1};
APM_Num_Dep = APM_Report_Temp{2};
APM_Avg_Taxi_Out_Time = APM_Report_Temp{15};
APM_Num_Arr = APM_Report_Temp{16};
APM_Avg_Taxi_In_Time = APM_Report_Temp{18};

% Delete the entry of "Total" in APM_Airport_Code

Entry_to_Delete = strcmp('Total', APM_Airport_Code);
APM_Airport_Code(Entry_to_Delete) = [ ];

Taxi_Time_Report = {APM_Airport_Code,APM_Num_Dep, APM_Avg_Taxi_Out_Time, APM_Num_Arr,
APM_Avg_Taxi_In_Time};

% clear Entry_to_Delete File_Name Entry_to_Delete File_Name
% flight_To_Load
```

```
flight_Size = length(flight);
Taxi_Profile =
struct('Origin_Airport',{},'Destination_Airport',{},'Aircraft_Type',{},'Frequency',{},
'Avg_Taxi_Out_Time',{},'Avg_Taxi_In_Time',{},'Taxi_Fuel_Consumption_gal',{});

for i = 1: flight_Size

   if isempty(flight(1,i).Origin)==0 && isempty(flight(1,i).Destination)==0 &&
isempty(flight(1,i).originalAircraft)==0 && isempty(flight(1,i).Frequency)==0

       Taxi_Profile(1,i).Origin_Airport = flight(1,i).Origin;
       Taxi_Profile(1,i).Destination_Airport = flight(1,i).Destination;
       Taxi_Profile(1,i).Aircraft_Type = flight(1,i).originalAircraft;
       Taxi_Profile(1,i).Frequency = flight(1,i).Frequency;

   else

       Taxi_Profile(1,i).Origin_Airport = 'n/a';
       Taxi_Profile(1,i).Destination_Airport = 'n/a';
       Taxi_Profile(1,i).Aircraft_Type = 'n/a';
       Taxi_Profile(1,i).Frequency = flight(1,i).Frequency;

       disp(['Error: Track ', num2str(i), ' missing!']);

   end

end

%--------------------------------------------------------------------------
% Step 3: Import APM taxi time report into Taxi_Profile
%--------------------------------------------------------------------------

disp('Data Processing, please wait...')

Taxi_Profile_Size = length(Taxi_Profile);
% if ismember({Taxi_Profile.Origin_Airport},Taxi_Time_Report{1,1}) == 1
%    n = find(cell2mat(Taxi_Time_Report{1,1})== Taxi_Profile.Origin_Airport);
%   Taxi_Profile.Avg_Taxi_Out_Time = Taxi_Time_Report{1,3}(org_index(n),1);
% elseif ismember({Taxi_Profile.Destination_Airport},Taxi_Time_Report{1,1}) ==1
```

```
%     n = find(cell2mat(Taxi_Time_Report{1,1})== Taxi_Profile.Destination_Airport);
%     Taxi_Profile.Avg_Taxi_In_Time = Taxi_Time_Report{1,5}(des_index(n),1);
% end
% pre-index the origin and destinations Taxi Profile airports to the ASPM 77
[~,org_index] = ismember({Taxi_Profile.Origin_Airport},Taxi_Time_Report{1,1});
[~,des_index] = ismember({Taxi_Profile.Destination_Airport},Taxi_Time_Report{1,1});

for n = 1:Taxi_Profile_Size

    if org_index(n) ~= 0    % an ASPM 77 origin
        Taxi_Profile(n).Avg_Taxi_Out_Time = Taxi_Time_Report{1,3}(org_index(n),1);
    end

    if des_index(n) ~= 0    % an ASPM 77 destination
        Taxi_Profile(n).Avg_Taxi_In_Time = Taxi_Time_Report{1,5}(des_index(n),1);
    end

end

disp('    ')
disp('Data Imported')

clear org_index des_index

fclose(fileID);

% clear Num_of_Airports

% clc

%-----------------------------------------------------------------------
% Average Taxi Time for Airports other than ASPM 77
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% Step 4: Find Out the Airport without Avg Taxi Out and Taxi
% In time. Call function Taxi_Out_Time_from_Num_of_Deps.m and
% Taxi_In_Time_from_Num_of_Arr.m, give value to those airports.
%-----------------------------------------------------------------------


% Average Taxi Out Time
```

```
disp('Processing Taxi Out Time for Airports other than ASPM 77, please wait')

% identify the unique origins and their locations
[U_org,I_org,J_org] = unique({Taxi_Profile.Origin_Airport});
N_org = length(U_org);
disp([num2str(N_org) ' unique origin airports need to be processed'])

% for each unique origin compute an average taxi out time
for or = 1:length(U_org)

    % DISPLAY PROGRESS EVERY 100 AIRPORTS
    if mod(or,100) == 0
        disp([num2str(or), '/', num2str(N_org)]);
    end

    % identify if the taxi out time is empty; if it is not, move to next airport
    if ~isempty(Taxi_Profile(I_org(or)).Avg_Taxi_Out_Time)
        continue;
    end

    % get the yearly departures and the avg taxi time
    ind2profiles = find(J_org==or);
    total_sum = 0;
    departures = 0;
    for index = 1: length(ind2profiles)
        total_sum = departures + total_sum;
        departures = sum(Taxi_Profile(index).Frequency(1:end));
    end
    avg_taxi_out_time_temp = Taxi_Out_Time_from_Num_of_Deps(departures);

    % set this avg taxi time for the appropriate taxi profiles
    for ind = 1:length(ind2profiles)
        Taxi_Profile(ind2profiles(ind)).Avg_Taxi_Out_Time = avg_taxi_out_time_temp;
    end

end



% Average Taxi In Time
```

```
disp('Processing Taxi In Time for Airports other than ASPM 77, please wait')

% identify the unique destinations and their locations
[U_des,I_des,J_des] = unique({Taxi_Profile.Destination_Airport});
N_des = length(U_des);
disp([num2str(N_des) ' unique destinations need to be processed'])

% for each unique destination compute an average taxi in time
for de = 1:length(U_des)

    % DISPLAY PROGRESS EVERY 100 AIRPORTS
    if mod(de,100) == 0
        disp([num2str(de), '/', num2str(N_des)]);
    end

    % identify if the taxi in time is empty; if it is not, move to next airport
    if ~isempty(Taxi_Profile(I_des(de)).Avg_Taxi_In_Time)
        continue;
    end

    % get the yearly arrivals and the avg taxi time
    ind2profiles = find(J_des==de);
    for index = 1: length(ind2profiles)
        total_sum = departures + total_sum;
        yearly_arrivals = sum(Taxi_Profile(index).Frequency(1:end));
    end
    %    yearly_arrivals = sum([Taxi_Profile(ind2profiles).Frequency]);
    avg_taxi_in_time_temp = Taxi_In_Time_from_Num_of_Arrs(yearly_arrivals);

    % set this avg taxi time for the appropriate taxi profiles
    for ind = 1:length(ind2profiles)
        Taxi_Profile(ind2profiles(ind)).Avg_Taxi_In_Time = avg_taxi_in_time_temp;
    end

end


%-----------------------------------------------------------------------
% Step 5:  Calculate yearly fuel consumption in each record in Taxi_Profile
%-----------------------------------------------------------------------

disp('Calculating Fuel Consumption for Each Record')
```

```
% Load BADA_Aircraft_Data.mat


load ([Install_Dir,
'\Passenger\Fuel_Consumption_And_Emissions\BADA_Aircraft_Data_for_Taxi.mat'])
load ([Install_Dir,
'\Passenger\Fuel_Consumption_And_Emissions\BADA_Aircraft_List_for_Taxi.mat'])
BADA_Aircraft_Data = BADA_Aircraft_Data_for_Taxi;


% BADA_Aircraft_List = BADA_Aircraft_List_for_Taxi;
% load ([Install_Dir,
'\Passenger\Fuel_Consumption_And_Emissions\BADA_Aircraft_List_for_Taxi.mat'])

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

% pre-index the aircraft types


% [~,ac_index] =
ismember({Taxi_Profile.Aircraft_Type},{BADA_Aircraft_Data.Aircraft_Name});
% AC_Type_string = zeros(Taxi_Profile_Size);
% for each unique Taxi Profile, compute the fuel consumption
for n = 1:Taxi_Profile_Size
    if mod(n,100) == 0
        disp([num2str(n), '/', num2str(Taxi_Profile_Size)]);
    end
    for aIndex = 1:length(Taxi_Profile(n).Aircraft_Type)

        %     AC_Type_string(n,1) = horzcat(Taxi_Profile(n).Aircraft_Type{1:end});
        %     AC_Type_char = char(AC_Type_string);

        %       Aircraft_indice =
find(AC_Type_char)==(BADA_Aircraft_Data.Aircraft_Name);
        if strcmp(Taxi_Profile(n).Aircraft_Type, 'n/a')==1
            continue
        else
        aircraftType = (Taxi_Profile(n).Aircraft_Type{aIndex});
        %       AircraftName      = Taxi_Profile(AC_Index).aircraftName;
        %       aircraftArray =
{'B763__';'B772__';'B744__';'B752__';'A333__';'A346__';'MD11__';'A388__';'B764__';'B76
```

```
2__';'B737__';'A320__';'A310__';'C750__';'FA7X__';'B773__';'B77W__';'B77L__';'CL60__';
'B788__';'B748__';'B733__';'A321__';'E145__';'E190__';'E170__';'A319__';'E135__';'B712
__';'B735__';'B738__';'B739__';'A332__';'AT45__';'AT72__';'CRJ2__';'CRJ9__';'DH8C__';'
DH8D__';'C208__';'BE30__';'MD82__';
'CS100__';'CS300__';'B737Max__';'B738Max__';'B739Max__';'A319neo__';'A320neo__';'A321n
eo__';'B777-8X__';'B777-9X__';'A350-
1000__';'A330neo__';'E190E2__';'E195E2__';'TW098__';'TW160__';'TW216__';'TW301__';'TW4
00__'};

       if isempty(find(ismember(BADA_Aircraft_List, aircraftType), 1))==1
          aircraftindex = strcmp(nextgenacft.aircraftName, aircraftType);% finds
position index

          aircraftPosition = nextgenacft(aircraftindex,2).aircraftPosition ;
       else
          aircraftPosition = find(ismember(BADA_Aircraft_List, aircraftType));
       end


       %       [~, Aircraft_Indice] = intersect(BADA_Aircraft_List,AC_Index);

       % compute the fel consumption
       flight(n).Taxi_Fuel_Consumption_kg(aIndex)  = (Taxi_Profile(n).Avg_Taxi_Out_Time
+ Taxi_Profile(n).Avg_Taxi_In_Time) * 60 *
BADA_Aircraft_Data_for_Taxi(aircraftPosition).Taxi_Fuel_Burn *
Taxi_Profile(n).Frequency(aIndex);
       %       flight(n).Taxi_Fuel_Consumption_gal =
flight(n).Taxi_Fuel_Consumption_kg * lbs_kg * fuelgal_lb;
       end
   end
end


disp(' ')
disp('Saving Taxi Times and Taxi fuel Consumption..');

disp(' ')
save ([Save_Dir, '\flight_3_', Case_Year_Str, '.mat'], 'flight', '-v7.3');
disp('Taxi profiles saved ')
disp(' ')

return
```

Taxi_In_Time_from_Num_of_Arrs.m

```
function avg_taxi_in_time_temp = Taxi_In_Time_from_Num_of_Arrs(num_of_arr)



% TAXI_TIME_FROM_NUM_OF_OPS Summary
% Input: Number of arrivals
% Output: Average Taxi In Time

avg_taxi_in_time_temp = 0.00001683*num_of_arr+4.4247; % as per the ASPM 77 airports in
2015
```

Taxi_Out_Time_from_Num_of_Deps.m

```
function avg_taxi_out_time_temp = Taxi_Out_Time_from_Num_of_Deps(num_of_dep)

%  TAXI_TIME_FROM_NUM_OF_OPS Summary
%  Input: Number of departures
%  Output: Average Taxi Out Time

avg_taxi_out_time_temp = 0.00002379*num_of_dep+17.4824; % as per the ASPM 77 airports
in 2015
```

Emissions_Calculator.m

```
% THIS M-FILE CALCULATES CO2 EMISSSION OF THE WHOLE FLIGHT CYCLE AND
% OTHER EMISSIONS INSIDE THE LTO CYCLE FOR EACH FLIGHT RECORD IN
% GDM INCLUDING GROUND EMISSIONS.

% Coded by Rahul_P

function Emissions_Calculator(Install_Dir, Save_Dir, Case_Year)

global aircraftArray NASAaircraft nextgenacftFuelburnSavingFactors
global lbs_kg fuelgal_lb

Case_Year_Str = num2str(Case_Year);

load ([Install_Dir,
'\Passenger\Fuel_Consumption_And_Emissions\Emission_Factor_Table.mat']);
load([Save_Dir, '\flight_2_OD_Time', Case_Year_Str,'.mat'])
load('D:\GDM_Git\GDM_DATA\DATA\Passenger\Fuel_Consumption_And_Emissions\nextgenacft_Fu
elburnSaving_Factors.mat')
load('D:\GDM_Git\GDM_DATA\DATA\Passenger\Fuel_Consumption_And_Emissions\NASAacft.mat')
;
% load([Install_Dir, '\Emission_Factor_Table.mat']);
% Variables included in the Emission_Factor_Table: Aircraft_Emission_Factor
%     Aircraft_List
%         Aircraft_Operational_Parameter
%   Aircraft_Fuel_Flow_Rate

% Fields in 'Aircraft_Emission_Factor':
% Field 1: Takeoff_CO_factor (g/kg)
% Field 2: Takeoff_HC_factor (g/kg)
% Field 3: Takeoff_NOx_factor (g/kg)
% Field 4: Takeoff_SOx_factor (g/kg)
% Field 5: Climb_Out_CO_factor (g/kg)
% Field 6: Climb_Out_HC_factor (g/kg)
% Field 7: Climb_Out_NOx_factor (g/kg)
% Field 8: Climb_Out_SOx_factor (g/kg)
% Field 9: Approach_CO_factor (g/kg)
% Field 10: Approach_HC_factor (g/kg)
% Field 11: Approach_NOx_factor (g/kg)
% Field 12: Approach_SOx_factor (g/kg)
% Field 13: Idle_CO_factor (g/kg)
% Field 14: Idle_HC_factor (g/kg)
```

```
% Field 15: Idle_NOx_factor (g/kg)
% Field 16: Idle_SOx_factor (g/kg)

% Fields in 'Aircraft_Fuel_Flow_Rate':
% Field 1: Takeoff_Fuel_Flow (kg/s)
% Field 2: Climb_Out_Fuel_Flow (kg/s)
% Field 3: Approach_Fuel_Flow (kg/s)
% Field 4: Idle_Fuel_Flow (kg/s)

% Fields in 'Aircraft_Operational_Parameter':
% Field 1: Takeoff Weight (lbs)
% Field 2: Takeoff Time (mins)
% Field 3: Climb Out Time (mins)
% Field 4: Approach Time (mins)
% Field 5: Landing Roll Time (mins)
disp(' ')
disp('Running Emissions Calculator..');
disp('Loading Flight Data, please wait...');
disp(' ')

load([Save_Dir, '\flight_3_', Case_Year_Str, '.mat'])

% load([Install_Dir,'\Aircraft_List.mat'])
% load([Output_Dir, '\Taxi_Profile_', Case_Year_Str, '.mat'])
disp(' ')
disp('Data Loaded')
disp(' ')

flight_Size = length(flight);
% Taxi_Profile_Size   = length(Taxi_Profile);


for od = 1: flight_Size
    if mod(od,10000) == 0
        disp([num2str(od), '/', num2str(flight_Size)]);
    end
    if OD_Time(od).Incorrect_Flight < 0
        continue % Skip too short &/or too long flights &/or (0,0) latlon flights
    else
        if isempty (flight(od).Climb_Profile) == 1
            continue
        end
```

```
        for aIndex = 1: length(flight(od).Aircraft_Index_in_BADA)
            if OD_Time(od).Incorrect_Flight(1, aIndex) < 0
                continue % Skip Incorrect Flight Assignment
            else
                % Frequency
                Frequency = flight(od).Frequency(aIndex);
                if Frequency == 0
                    continue
                else

                    % Takeoff, Climb, Cruise, Descent emissions
                    current_aircraft_type = flight(od).originalAircraft(aIndex);
                    if strcmp(flight(od).originalAircraft(aIndex),'CS100__') == 1 ||
strcmp(flight(od).originalAircraft(aIndex),'CS300__') == 1 ||
strcmp(flight(od).originalAircraft(aIndex),'A320neo__') ==
1||strcmp(flight(od).originalAircraft(aIndex),'B737Max__') == 1 ||
strcmp(flight(od).originalAircraft(aIndex),'B738Max__') == 1 ||
strcmp(flight(od).originalAircraft(aIndex),'B739Max__') == 1
                        current_aircraft_type = 'A320__';
                    elseif  strcmp(flight(od).originalAircraft(aIndex),'A319neo__') ==
1   %
                        current_aircraft_type = 'A319__';
                    elseif  strcmp(flight(od).originalAircraft(aIndex),'A321neo__') == 1
                        current_aircraft_type = 'A321__';
                    elseif  strcmp(flight(od).originalAircraft(aIndex),'E190E2__') == 1
                        current_aircraft_type = 'E190__';
                    elseif strcmp(flight(od).originalAircraft(aIndex),'E195E2__') == 1
                        current_aircraft_type = 'E195__';
                    elseif strcmp(flight(od).originalAircraft(aIndex),'B777-8X__') ==
1||strcmp(flight(od).originalAircraft(aIndex),'B777-9X__') == 1||
strcmp(flight(od).originalAircraft(aIndex),'A350-1000__') == 1
                        current_aircraft_type = 'B773__';
                    elseif strcmp(flight(od).originalAircraft(aIndex),'A330neo__') == 1
                        current_aircraft_type = 'A332__';
                    elseif   strcmp(flight(od).originalAircraft(aIndex),'TW098__') == 1
                        current_aircraft_type = 'E190__';
                    elseif   strcmp(flight(od).originalAircraft(aIndex),'TW160__') == 1
                        current_aircraft_type = 'B738__';
                    elseif   strcmp(flight(od).originalAircraft(aIndex),'TW216__') == 1
                        current_aircraft_type = 'B762__';
                    elseif   strcmp(flight(od).originalAircraft(aIndex),'TW301__') == 1
                        current_aircraft_type = 'B772__';
```

```
                elseif   strcmp(flight(od).originalAircraft(aIndex),'TW400__') == 1
                    current_aircraft_type = 'B744__';
                elseif  strcmp(flight(od).originalAircraft(aIndex),'BE30__') == 1
                    current_aircraft_type = 'PA31__';
                elseif strcmp(flight(od).originalAircraft(aIndex),'C208__') == 1
                    current_aircraft_type = 'TBM7__';
                end
                if ismember( current_aircraft_type,Aircraft_List(:,1))==1
                    aircraft_idx        = strcmp(current_aircraft_type,
Aircraft_List(:,1));

                    if isempty
(flight(od).Climb_Profile(aIndex).Distance_For_Climb_nm) == 1
                        continue
                    end

                    % Takeoff Emissions for single flight (LTO)
                    Takeoff_Fuel_Burn_kg  =
flight(od).Climb_Profile(aIndex).Total_Fuel_kg;
                    %       Takeoff_Fuel_Burn_kg  =
flight(od).Takeoff_Fuel_Burn_kg(aIndex);
                    Takeoff_Fuel_Burn_gal = Takeoff_Fuel_Burn_kg * lbs_kg *
fuelgal_lb;

                    Takeoff_CO_kg  = Takeoff_Fuel_Burn_kg *
Aircraft_Emission_Factor(aircraft_idx,1) / 1000;
                    Takeoff_HC_kg  = Takeoff_Fuel_Burn_kg *
Aircraft_Emission_Factor(aircraft_idx,2) / 1000;
                    Takeoff_NOx_kg = Takeoff_Fuel_Burn_kg *
Aircraft_Emission_Factor(aircraft_idx,3) / 1000;
                    Takeoff_SOx_kg = Takeoff_Fuel_Burn_kg *
Aircraft_Emission_Factor(aircraft_idx,4) / 1000;
                    Takeoff_CO2_kg = Takeoff_Fuel_Burn_kg * 3.157;

                    % Climb-out Emissions (AGL) for single flight (LTO)
                    [~, climb_weight_idx] =
min(abs(flight(od).Climb_Profile(aIndex).Altitude_ft -
(flight(od).Climb_Profile(aIndex).Altitude_ft(1) + 3000)));

                    ClimbOut_Fuel_Burn_kg_AGL  =
flight(od).Climb_Profile(aIndex).Weight_kg(1) -
flight(od).Climb_Profile(aIndex).Weight_kg(climb_weight_idx);
```

```
                    ClimbOut_Fuel_Burn_gal_AGL = ClimbOut_Fuel_Burn_kg_AGL * lbs_kg *
fuelgal_lb;

                    LTO_ClimbOut_CO_kg  = ClimbOut_Fuel_Burn_kg_AGL *
Aircraft_Emission_Factor(aircraft_idx,5) / 1000;
                    LTO_ClimbOut_HC_kg  = ClimbOut_Fuel_Burn_kg_AGL *
Aircraft_Emission_Factor(aircraft_idx,6) / 1000;
                    LTO_ClimbOut_NOx_kg = ClimbOut_Fuel_Burn_kg_AGL *
Aircraft_Emission_Factor(aircraft_idx,7) / 1000;
                    LTO_ClimbOut_SOx_kg = ClimbOut_Fuel_Burn_kg_AGL *
Aircraft_Emission_Factor(aircraft_idx,8) / 1000;
                    LTO_ClimbOut_CO2_kg = ClimbOut_Fuel_Burn_kg_AGL * 3.157;

                    % Descent Emissions (AGL) for single flight (LTO)
                    %       [~, descent_weight_idx] =
min(abs(flight(od).Descent_Profile(aIndex).Altitude_ft -
(flight(od).Descent_Profile(aIndex).Altitude_ft(end) + 3000)));

                    Descent_Fuel_Burn_kg_AGL  =
flight(od).Descent_Profile(aIndex).mass;
                    Descent_Fuel_Burn_gal_AGL = Descent_Fuel_Burn_kg_AGL * lbs_kg *
fuelgal_lb;

                    LTO_Descent_CO_kg  = Descent_Fuel_Burn_kg_AGL *
Aircraft_Emission_Factor(aircraft_idx,9)  / 1000;
                    LTO_Descent_HC_kg  = Descent_Fuel_Burn_kg_AGL *
Aircraft_Emission_Factor(aircraft_idx,10) / 1000;
                    LTO_Descent_NOx_kg = Descent_Fuel_Burn_kg_AGL *
Aircraft_Emission_Factor(aircraft_idx,11) / 1000;
                    LTO_Descent_SOx_kg = Descent_Fuel_Burn_kg_AGL *
Aircraft_Emission_Factor(aircraft_idx,12) / 1000;
                    LTO_Descent_CO2_kg = Descent_Fuel_Burn_kg_AGL * 3.157;

                    % Full Cycle
                    Climb_Fuel_Burn_kg    =
flight(od).Climb_Profile(aIndex).Total_Fuel_kg;
                    Cruise_Fuel_Burn_kg   = flight(od).Step_Cruise_Phase(aIndex).Fuel;
                    Descent_Fuel_Burn_kg  = flight(od).Descent_Profile(aIndex).mass;

                    % Total airborne fuel consumption (for all departures, prior to
taxi numbers)
```

```
                    Total_Airborne_Fuel_Consumption_kg =
flight(od).Total_Fuel_Burnt_kg_oneflight(aIndex);



                else

                    % Takeoff Emissions for single flight (LTO)
                    Takeoff_Fuel_Burn_kg  = 0;
                    Takeoff_Fuel_Burn_gal = 0;

                    Takeoff_CO_kg  = 0;
                    Takeoff_HC_kg  = 0;
                    Takeoff_NOx_kg = 0;
                    Takeoff_SOx_kg = 0;
                    Takeoff_CO2_kg = 0;

                    % Climb-out Emissions (AGL) for single flight (LTO)
                    LTO_ClimbOut_CO_kg  = 0;
                    LTO_ClimbOut_HC_kg  = 0;
                    LTO_ClimbOut_NOx_kg = 0;
                    LTO_ClimbOut_SOx_kg = 0;
                    LTO_ClimbOut_CO2_kg = 0;

                    % Descent Emissions (AGL) for single flight (LTO)
                    LTO_Descent_CO_kg  = 0;
                    LTO_Descent_HC_kg  = 0;
                    LTO_Descent_NOx_kg = 0;
                    LTO_Descent_SOx_kg = 0;
                    LTO_Descent_CO2_kg = 0;

                    % Full Cycle
                    Climb_Fuel_Burn_kg    = 0;
                    Cruise_Fuel_Burn_kg   = 0;
                    Descent_Fuel_Burn_kg  = 0;

                    % Total airborne fuel consumption (for all departures, prior to
taxi numbers)
                    Total_Airborne_Fuel_Consumption_kg = 0;

                    % Frequency
```

```
                    %       Frequency = 1;      % need a 1 in cases of empty frequency
for un-computed tracks

                end % if isempty(flight(od).Fuel_Consumption_kg) == 0

                % Taxi emissions
                if isempty(flight(od).Taxi_Fuel_Consumption_kg) == 0

                    % Taxi Emissions for single flight (LTO)
                    Total_Taxi_Fuel_Burn_kg =
flight(od).Taxi_Fuel_Consumption_kg(aIndex);  % all track departures
                    Taxi_Fuel_Burn_kg  = Total_Taxi_Fuel_Burn_kg /
flight(od).Frequency(aIndex);
                    Taxi_Fuel_Burn_gal = Taxi_Fuel_Burn_kg * lbs_kg * fuelgal_lb;

                    Taxi_CO_kg  = Taxi_Fuel_Burn_kg *
Aircraft_Emission_Factor(aircraft_idx,13) / 1000;
                    Taxi_HC_kg  = Taxi_Fuel_Burn_kg *
Aircraft_Emission_Factor(aircraft_idx,14) / 1000;
                    Taxi_NOx_kg = Taxi_Fuel_Burn_kg *
Aircraft_Emission_Factor(aircraft_idx,15) / 1000;
                    Taxi_SOx_kg = Taxi_Fuel_Burn_kg *
Aircraft_Emission_Factor(aircraft_idx,16) / 1000;
                    Taxi_CO2_kg = Taxi_Fuel_Burn_kg * 3.157;

                else

                    Total_Taxi_Fuel_Burn_kg = 0;
                    Taxi_Fuel_Burn_kg  = 0;
                    Taxi_Fuel_Burn_gal = 0;

                    Taxi_CO_kg  = 0;
                    Taxi_HC_kg  = 0;
                    Taxi_NOx_kg = 0;
                    Taxi_SOx_kg = 0;
                    Taxi_CO2_kg = 0;

                end % if isempty(Taxi_Profile(od).Taxi_Fuel_Consumption_kg) == 0

                % combine the airborne and taxi fuel consumption numbers (for all
departures; total)
```

```
                Total_Fuel_Consumption_kg = Total_Airborne_Fuel_Consumption_kg +
Total_Taxi_Fuel_Burn_kg;

                % Emissions for all flights
                LTO_CO_kg  = (Takeoff_CO_kg  + LTO_ClimbOut_CO_kg  + LTO_Descent_CO_kg
+ Taxi_CO_kg)  * Frequency;
                LTO_HC_kg  = (Takeoff_HC_kg  + LTO_ClimbOut_HC_kg  + LTO_Descent_HC_kg
+ Taxi_HC_kg)  * Frequency;
                LTO_NOx_kg = (Takeoff_NOx_kg + LTO_ClimbOut_NOx_kg +
LTO_Descent_NOx_kg + Taxi_NOx_kg) * Frequency;
                LTO_SOx_kg = (Takeoff_SOx_kg + LTO_ClimbOut_SOx_kg +
LTO_Descent_SOx_kg + Taxi_SOx_kg) * Frequency;
                LTO_CO2_kg = (Takeoff_CO2_kg + LTO_ClimbOut_CO2_kg +
LTO_Descent_CO2_kg + Taxi_CO2_kg) * Frequency;

                if  strcmp(flight(od).originalAircraft(aIndex),'TW098__') == 1
                    flight(od).LTO_CO_Emission_kg(aIndex)  = LTO_CO_kg -
(NASAaircraft(1).LTO_CO_kg_factor*(LTO_CO_kg)) ;
                    flight(od).LTO_HC_Emission_kg(aIndex)  = LTO_HC_kg -
(NASAaircraft(1).LTO_HC_kg_factor*(LTO_HC_kg));
                    flight(od).LTO_NOx_Emission_kg(aIndex) = LTO_NOx_kg -
(NASAaircraft(1).LTO_NOx_kg_factor*(LTO_NOx_kg));
                    flight(od).LTO_SOx_Emission_kg(aIndex) = LTO_SOx_kg  -
(NASAaircraft(1).LTO_SOx_kg_factor*(LTO_SOx_kg));
                    flight(od).LTO_CO2_Emission_kg(aIndex) = LTO_CO2_kg -
(NASAaircraft(1).LTO_CO2_kg_factor*(LTO_CO2_kg));

                    flight(od).Takeoff_Emissions_Single_Flight.CO_kg(aIndex)  =
Takeoff_CO_kg -(NASAaircraft(1).Takeoff_CO_kg_factor*(Takeoff_CO_kg)) ;
                    flight(od).Takeoff_Emissions_Single_Flight.HC_kg(aIndex)  =
Takeoff_HC_kg - (NASAaircraft(1).Takeoff_HC_kg_factor*(Takeoff_HC_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.NOx_kg(aIndex) =
Takeoff_NOx_kg - (NASAaircraft(1).Takeoff_NOx_kg_factor*(Takeoff_NOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.SOx_kg(aIndex) =
Takeoff_SOx_kg - (NASAaircraft(1).Takeoff_SOx_kg_factor*(Takeoff_SOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.CO2_kg(aIndex) =
Takeoff_CO2_kg - (NASAaircraft(1).Takeoff_CO2_kg_factor*(Takeoff_CO2_kg));

                    flight(od).Climb_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_ClimbOut_CO_kg -(NASAaircraft(1).LTO_ClimbOut_CO_kg_factor*(LTO_ClimbOut_CO_kg));
                    flight(od).Climb_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_ClimbOut_HC_kg -(NASAaircraft(1).LTO_ClimbOut_HC_kg_factor*(LTO_ClimbOut_HC_kg));
```

```
                    flight(od).Climb_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_ClimbOut_NOx_kg -
(NASAaircraft(1).LTO_ClimbOut_NOx_kg_factor*(LTO_ClimbOut_NOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_ClimbOut_SOx_kg -
(NASAaircraft(1).LTO_ClimbOut_SOx_kg_factor*(LTO_ClimbOut_SOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_ClimbOut_CO2_kg -
(NASAaircraft(1).LTO_ClimbOut_CO2_kg_factor*(LTO_ClimbOut_CO2_kg));

                    flight(od).Descent_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_Descent_CO_kg -(NASAaircraft(1).LTO_Descent_CO_kg_factor*(LTO_Descent_CO_kg));
                    flight(od).Descent_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_Descent_HC_kg -(NASAaircraft(1).LTO_Descent_HC_kg_factor*(LTO_Descent_HC_kg));
                    flight(od).Descent_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_Descent_NOx_kg -(NASAaircraft(1).LTO_Descent_NOx_kg_factor*(LTO_Descent_NOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_Descent_SOx_kg -(NASAaircraft(1).LTO_Descent_SOx_kg_factor*(LTO_Descent_SOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_Descent_CO2_kg - (NASAaircraft(1).LTO_Descent_CO2_kg_factor*(LTO_Descent_CO2_kg));

                    flight(od).Taxi_Emissions_Single_Flight.CO_kg(aIndex)  =
Taxi_CO_kg - (NASAaircraft(1).Taxi_CO_kg_factor* (Taxi_CO_kg));
                    flight(od).Taxi_Emissions_Single_Flight.HC_kg(aIndex)  =
Taxi_HC_kg - (NASAaircraft(1).Taxi_HC_kg_factor* (Taxi_HC_kg));
                    flight(od).Taxi_Emissions_Single_Flight.NOx_kg(aIndex) =
Taxi_NOx_kg - (NASAaircraft(1).Taxi_NOx_kg_factor* (Taxi_NOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.SOx_kg(aIndex) =
Taxi_SOx_kg - (NASAaircraft(1).Taxi_SOx_kg_factor* (Taxi_SOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.CO2_kg(aIndex) =
Taxi_CO2_kg - (NASAaircraft(1).Taxi_CO2_kg_factor* (Taxi_CO2_kg));
                elseif   strcmp(flight(od).originalAircraft(aIndex),'TW160__') == 1
                    flight(od).LTO_CO_Emission_kg(aIndex)  = LTO_CO_kg -
(NASAaircraft(2).LTO_CO_kg_factor*(LTO_CO_kg)) ;
                    flight(od).LTO_HC_Emission_kg(aIndex)  = LTO_HC_kg -
(NASAaircraft(2).LTO_HC_kg_factor*(LTO_HC_kg));
                    flight(od).LTO_NOx_Emission_kg(aIndex) = LTO_NOx_kg -
(NASAaircraft(2).LTO_NOx_kg_factor*(LTO_NOx_kg));
                    flight(od).LTO_SOx_Emission_kg(aIndex) = LTO_SOx_kg  -
(NASAaircraft(2).LTO_SOx_kg_factor*(LTO_SOx_kg));
                    flight(od).LTO_CO2_Emission_kg(aIndex) = LTO_CO2_kg -
(NASAaircraft(2).LTO_CO2_kg_factor*(LTO_CO2_kg));
```

```
                    flight(od).Takeoff_Emissions_Single_Flight.CO_kg(aIndex)  =
Takeoff_CO_kg -(NASAaircraft(2).Takeoff_CO_kg_factor*(Takeoff_CO_kg)) ;
                    flight(od).Takeoff_Emissions_Single_Flight.HC_kg(aIndex)  =
Takeoff_HC_kg - (NASAaircraft(2).Takeoff_HC_kg_factor*(Takeoff_HC_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.NOx_kg(aIndex) =
Takeoff_NOx_kg - (NASAaircraft(2).Takeoff_HC_kg_factor*(Takeoff_NOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.SOx_kg(aIndex) =
Takeoff_SOx_kg - (NASAaircraft(2).Takeoff_SOx_kg_factor*(Takeoff_SOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.CO2_kg(aIndex) =
Takeoff_CO2_kg - (NASAaircraft(2).Takeoff_CO2_kg_factor*(Takeoff_CO2_kg));

                    flight(od).Climb_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_ClimbOut_CO_kg -(NASAaircraft(2).LTO_ClimbOut_CO_kg_factor*(LTO_ClimbOut_CO_kg));
                    flight(od).Climb_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_ClimbOut_HC_kg -(NASAaircraft(2).LTO_ClimbOut_HC_kg_factor*(LTO_ClimbOut_HC_kg));
                    flight(od).Climb_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_ClimbOut_NOx_kg -
(NASAaircraft(2).LTO_ClimbOut_NOx_kg_factor*(LTO_ClimbOut_NOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_ClimbOut_SOx_kg -
(NASAaircraft(2).LTO_ClimbOut_SOx_kg_factor*(LTO_ClimbOut_SOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_ClimbOut_CO2_kg -
(NASAaircraft(2).LTO_ClimbOut_CO2_kg_factor*(LTO_ClimbOut_CO2_kg));

                    flight(od).Descent_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_Descent_CO_kg -(NASAaircraft(2).LTO_Descent_CO_kg_factor*(LTO_Descent_CO_kg));
                    flight(od).Descent_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_Descent_HC_kg -(NASAaircraft(2).LTO_Descent_HC_kg_factor*(LTO_Descent_HC_kg));
                    flight(od).Descent_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_Descent_NOx_kg -(NASAaircraft(2).LTO_Descent_NOx_kg_factor*(LTO_Descent_NOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_Descent_SOx_kg -(NASAaircraft(2).LTO_Descent_SOx_kg_factor*(LTO_Descent_SOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_Descent_CO2_kg - (NASAaircraft(2).LTO_Descent_CO2_kg_factor*(LTO_Descent_CO2_kg));

                    flight(od).Taxi_Emissions_Single_Flight.CO_kg(aIndex)  =
Taxi_CO_kg - (NASAaircraft(2).Taxi_CO_kg_factor* (Taxi_CO_kg));
                    flight(od).Taxi_Emissions_Single_Flight.HC_kg(aIndex)  =
Taxi_HC_kg - (NASAaircraft(2).Taxi_HC_kg_factor* (Taxi_HC_kg));
```

```
                    flight(od).Taxi_Emissions_Single_Flight.NOx_kg(aIndex) =
Taxi_NOx_kg - (NASAaircraft(2).Taxi_NOx_kg_factor* (Taxi_NOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.SOx_kg(aIndex) =
Taxi_SOx_kg - (NASAaircraft(2).Taxi_SOx_kg_factor* (Taxi_SOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.CO2_kg(aIndex) =
Taxi_CO2_kg - (NASAaircraft(2).Taxi_CO2_kg_factor* (Taxi_CO2_kg));
                elseif   strcmp(flight(od).originalAircraft(aIndex),'TW216__') == 1
                    flight(od).LTO_CO_Emission_kg(aIndex)  = LTO_CO_kg -
(NASAaircraft(3).LTO_CO_kg_factor*(LTO_CO_kg)) ;
                    flight(od).LTO_HC_Emission_kg(aIndex)  = LTO_HC_kg -
(NASAaircraft(3).LTO_HC_kg_factor*(LTO_HC_kg));
                    flight(od).LTO_NOx_Emission_kg(aIndex) = LTO_NOx_kg -
(NASAaircraft(3).LTO_NOx_kg_factor*(LTO_NOx_kg));
                    flight(od).LTO_SOx_Emission_kg(aIndex) = LTO_SOx_kg  -
(NASAaircraft(3).LTO_SOx_kg_factor*(LTO_SOx_kg));
                    flight(od).LTO_CO2_Emission_kg(aIndex) = LTO_CO2_kg -
(NASAaircraft(3).LTO_CO2_kg_factor*(LTO_CO2_kg));

                    flight(od).Takeoff_Emissions_Single_Flight.CO_kg(aIndex)  =
Takeoff_CO_kg -(NASAaircraft(3).Takeoff_CO_kg_factor*(Takeoff_CO_kg)) ;
                    flight(od).Takeoff_Emissions_Single_Flight.HC_kg(aIndex)  =
Takeoff_HC_kg - (NASAaircraft(3).Takeoff_HC_kg_factor*(Takeoff_HC_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.NOx_kg(aIndex) =
Takeoff_NOx_kg - (NASAaircraft(3).Takeoff_NOx_kg_factor*(Takeoff_NOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.SOx_kg(aIndex) =
Takeoff_SOx_kg - (NASAaircraft(3).Takeoff_SOx_kg_factor*(Takeoff_SOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.CO2_kg(aIndex) =
Takeoff_CO2_kg - (NASAaircraft(3).Takeoff_CO2_kg_factor*(Takeoff_CO2_kg));

                    flight(od).Climb_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_ClimbOut_CO_kg -(NASAaircraft(3).LTO_ClimbOut_CO_kg_factor*(LTO_ClimbOut_CO_kg));
                    flight(od).Climb_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_ClimbOut_HC_kg -(NASAaircraft(3).LTO_ClimbOut_HC_kg_factor*(LTO_ClimbOut_HC_kg));
                    flight(od).Climb_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_ClimbOut_NOx_kg -
(NASAaircraft(3).LTO_ClimbOut_NOx_kg_factor*(LTO_ClimbOut_NOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_ClimbOut_SOx_kg -
(NASAaircraft(3).LTO_ClimbOut_SOx_kg_factor*(LTO_ClimbOut_SOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_ClimbOut_CO2_kg -
(NASAaircraft(3).LTO_ClimbOut_CO2_kg_factor*(LTO_ClimbOut_CO2_kg));
```

```
                    flight(od).Descent_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_Descent_CO_kg -(NASAaircraft(3).LTO_Descent_CO_kg_factor*(LTO_Descent_CO_kg));
                    flight(od).Descent_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_Descent_HC_kg -(NASAaircraft(3).LTO_Descent_HC_kg_factor*(LTO_Descent_HC_kg));
                    flight(od).Descent_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_Descent_NOx_kg -(NASAaircraft(3).LTO_Descent_NOx_kg_factor*(LTO_Descent_NOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_Descent_SOx_kg -(NASAaircraft(3).LTO_Descent_SOx_kg_factor*(LTO_Descent_SOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_Descent_CO2_kg - (NASAaircraft(3).LTO_Descent_CO2_kg_factor*(LTO_Descent_CO2_kg));

                    flight(od).Taxi_Emissions_Single_Flight.CO_kg(aIndex)  =
Taxi_CO_kg - (NASAaircraft(3).Taxi_CO_kg_factor* (Taxi_CO_kg));
                    flight(od).Taxi_Emissions_Single_Flight.HC_kg(aIndex)  =
Taxi_HC_kg - (NASAaircraft(3).Taxi_HC_kg_factor* (Taxi_HC_kg));
                    flight(od).Taxi_Emissions_Single_Flight.NOx_kg(aIndex) =
Taxi_NOx_kg - (NASAaircraft(3).Taxi_NOx_kg_factor* (Taxi_NOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.SOx_kg(aIndex) =
Taxi_SOx_kg - (NASAaircraft(3).Taxi_SOx_kg_factor* (Taxi_SOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.CO2_kg(aIndex) =
Taxi_CO2_kg - (NASAaircraft(3).Taxi_CO2_kg_factor* (Taxi_CO2_kg));
                elseif   strcmp(flight(od).originalAircraft(aIndex),'TW301__') == 1
                    flight(od).LTO_CO_Emission_kg(aIndex)  = LTO_CO_kg -
(NASAaircraft(4).LTO_CO_kg_factor*(LTO_CO_kg)) ;
                    flight(od).LTO_HC_Emission_kg(aIndex)  = LTO_HC_kg -
(NASAaircraft(4).LTO_HC_kg_factor*(LTO_HC_kg));
                    flight(od).LTO_NOx_Emission_kg(aIndex) = LTO_NOx_kg -
(NASAaircraft(4).LTO_NOx_kg_factor*(LTO_NOx_kg));
                    flight(od).LTO_SOx_Emission_kg(aIndex) = LTO_SOx_kg  -
(NASAaircraft(4).LTO_SOx_kg_factor*(LTO_SOx_kg));
                    flight(od).LTO_CO2_Emission_kg(aIndex) = LTO_CO2_kg -
(NASAaircraft(4).LTO_CO2_kg_factor*(LTO_CO2_kg));

                    flight(od).Takeoff_Emissions_Single_Flight.CO_kg(aIndex)  =
Takeoff_CO_kg -(NASAaircraft(4).Takeoff_CO_kg_factor*(Takeoff_CO_kg)) ;
                    flight(od).Takeoff_Emissions_Single_Flight.HC_kg(aIndex)  =
Takeoff_HC_kg - (NASAaircraft(4).Takeoff_HC_kg_factor*(Takeoff_HC_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.NOx_kg(aIndex) =
Takeoff_NOx_kg - (NASAaircraft(4).Takeoff_NOx_kg_factor*(Takeoff_NOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.SOx_kg(aIndex) =
Takeoff_SOx_kg - (NASAaircraft(4).Takeoff_SOx_kg_factor*(Takeoff_SOx_kg));
```

```
                    flight(od).Takeoff_Emissions_Single_Flight.CO2_kg(aIndex) =
Takeoff_CO2_kg - (NASAaircraft(4).Takeoff_CO2_kg_factor*(Takeoff_CO2_kg));

                    flight(od).Climb_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_ClimbOut_CO_kg -(NASAaircraft(4).LTO_ClimbOut_CO_kg_factor*(LTO_ClimbOut_CO_kg));
                    flight(od).Climb_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_ClimbOut_HC_kg -(NASAaircraft(4).LTO_ClimbOut_HC_kg_factor*(LTO_ClimbOut_HC_kg));
                    flight(od).Climb_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_ClimbOut_NOx_kg -
(NASAaircraft(4).LTO_ClimbOut_NOx_kg_factor*(LTO_ClimbOut_NOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_ClimbOut_SOx_kg -
(NASAaircraft(4).LTO_ClimbOut_SOx_kg_factor*(LTO_ClimbOut_SOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_ClimbOut_CO2_kg -
(NASAaircraft(4).LTO_ClimbOut_CO2_kg_factor*(LTO_ClimbOut_CO2_kg));

                    flight(od).Descent_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_Descent_CO_kg -(NASAaircraft(4).LTO_Descent_CO_kg_factor*(LTO_Descent_CO_kg));
                    flight(od).Descent_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_Descent_HC_kg -(NASAaircraft(4).LTO_Descent_HC_kg_factor*(LTO_Descent_HC_kg));
                    flight(od).Descent_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_Descent_NOx_kg -(NASAaircraft(4).LTO_Descent_NOx_kg_factor*(LTO_Descent_NOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_Descent_SOx_kg -(NASAaircraft(4).LTO_Descent_SOx_kg_factor*(LTO_Descent_SOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_Descent_CO2_kg - (NASAaircraft(4).LTO_Descent_CO2_kg_factor*(LTO_Descent_CO2_kg));

                    flight(od).Taxi_Emissions_Single_Flight.CO_kg(aIndex)  =
Taxi_CO_kg - (NASAaircraft(4).Taxi_CO_kg_factor* (Taxi_CO_kg));
                    flight(od).Taxi_Emissions_Single_Flight.HC_kg(aIndex)  =
Taxi_HC_kg - (NASAaircraft(4).Taxi_HC_kg_factor* (Taxi_HC_kg));
                    flight(od).Taxi_Emissions_Single_Flight.NOx_kg(aIndex) =
Taxi_NOx_kg - (NASAaircraft(4).Taxi_NOx_kg_factor* (Taxi_NOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.SOx_kg(aIndex) =
Taxi_SOx_kg - (NASAaircraft(4).Taxi_SOx_kg_factor* (Taxi_SOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.CO2_kg(aIndex) =
Taxi_CO2_kg - (NASAaircraft(4).Taxi_CO2_kg_factor* (Taxi_CO2_kg));
                elseif   strcmp(flight(od).originalAircraft(aIndex),'TW400__') == 1

                    flight(od).LTO_CO_Emission_kg(aIndex)  = LTO_CO_kg -
(NASAaircraft(5).LTO_CO_kg_factor*(LTO_CO_kg)) ;
```

```
                    flight(od).LTO_HC_Emission_kg(aIndex)  = LTO_HC_kg -
(NASAaircraft(5).LTO_HC_kg_factor*(LTO_HC_kg));
                    flight(od).LTO_NOx_Emission_kg(aIndex) = LTO_NOx_kg -
(NASAaircraft(5).LTO_NOx_kg_factor*(LTO_NOx_kg));
                    flight(od).LTO_SOx_Emission_kg(aIndex) = LTO_SOx_kg  -
(NASAaircraft(5).LTO_SOx_kg_factor*(LTO_SOx_kg));
                    flight(od).LTO_CO2_Emission_kg(aIndex) = LTO_CO2_kg -
(NASAaircraft(5).LTO_CO2_kg_factor*(LTO_CO2_kg));

                    flight(od).Takeoff_Emissions_Single_Flight.CO_kg(aIndex)  =
Takeoff_CO_kg -(NASAaircraft(5).Takeoff_CO_kg_factor*(Takeoff_CO_kg)) ;
                    flight(od).Takeoff_Emissions_Single_Flight.HC_kg(aIndex)  =
Takeoff_HC_kg - (NASAaircraft(5).Takeoff_HC_kg_factor*(Takeoff_HC_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.NOx_kg(aIndex) =
Takeoff_NOx_kg - (NASAaircraft(5).Takeoff_NOx_kg_factor*(Takeoff_NOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.SOx_kg(aIndex) =
Takeoff_SOx_kg - (NASAaircraft(5).Takeoff_SOx_kg_factor*(Takeoff_SOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.CO2_kg(aIndex) =
Takeoff_CO2_kg - (NASAaircraft(5).Takeoff_CO2_kg_factor*(Takeoff_CO2_kg));

                    flight(od).Climb_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_ClimbOut_CO_kg -(NASAaircraft(5).LTO_ClimbOut_CO_kg_factor*(LTO_ClimbOut_CO_kg));
                    flight(od).Climb_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_ClimbOut_HC_kg -(NASAaircraft(5).LTO_ClimbOut_HC_kg_factor*(LTO_ClimbOut_HC_kg));
                    flight(od).Climb_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_ClimbOut_NOx_kg -
(NASAaircraft(5).LTO_ClimbOut_NOx_kg_factor*(LTO_ClimbOut_NOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_ClimbOut_SOx_kg -
(NASAaircraft(5).LTO_ClimbOut_SOx_kg_factor*(LTO_ClimbOut_SOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_ClimbOut_CO2_kg -
(NASAaircraft(5).LTO_ClimbOut_CO2_kg_factor*(LTO_ClimbOut_CO2_kg));

                    flight(od).Descent_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_Descent_CO_kg -(NASAaircraft(5).LTO_Descent_CO_kg_factor*(LTO_Descent_CO_kg));
                    flight(od).Descent_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_Descent_HC_kg -(NASAaircraft(5).LTO_Descent_HC_kg_factor*(LTO_Descent_HC_kg));
                    flight(od).Descent_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_Descent_NOx_kg -(NASAaircraft(5).LTO_Descent_NOx_kg_factor*(LTO_Descent_NOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_Descent_SOx_kg -(NASAaircraft(5).LTO_Descent_SOx_kg_factor*(LTO_Descent_SOx_kg));
```

```
                    flight(od).Descent_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_Descent_CO2_kg - (NASAaircraft(5).LTO_Descent_CO2_kg_factor*(LTO_Descent_CO2_kg));
                elseif   strcmp(flight(od).originalAircraft(aIndex),'CS100__') == 1 ||
strcmp(flight(od).originalAircraft(aIndex),'CS300__') == 1
||strcmp(flight(od).originalAircraft(aIndex),'B777-7x__') == 1
||strcmp(flight(od).originalAircraft(aIndex),'B777-8X__') == 1
||strcmp(flight(od).originalAircraft(aIndex),'B777-9X__') == 1
||strcmp(flight(od).originalAircraft(aIndex),'A319neo__') == 1
||strcmp(flight(od).originalAircraft(aIndex),'A320neo__') == 1
||strcmp(flight(od).originalAircraft(aIndex),'A321neo__') == 1

                    flight(od).LTO_CO_Emission_kg(aIndex)  = LTO_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_CO_kg)) ;
                    flight(od).LTO_HC_Emission_kg(aIndex)  = LTO_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_HC_kg));
                    flight(od).LTO_NOx_Emission_kg(aIndex) = LTO_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_NOx_kg));
                    flight(od).LTO_SOx_Emission_kg(aIndex) = LTO_SOx_kg  -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_SOx_kg));
                    flight(od).LTO_CO2_Emission_kg(aIndex) = LTO_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_CO2_kg));

                    flight(od).Takeoff_Emissions_Single_Flight.CO_kg(aIndex)  =
Takeoff_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(Takeoff_CO_kg)) ;
                    flight(od).Takeoff_Emissions_Single_Flight.HC_kg(aIndex)  =
Takeoff_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(Takeoff_HC_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.NOx_kg(aIndex) =
Takeoff_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(Takeoff_NOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.SOx_kg(aIndex) =
Takeoff_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(Takeoff_SOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.CO2_kg(aIndex) =
Takeoff_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(Takeoff_CO2_kg));

                    flight(od).Climb_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_ClimbOut_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_ClimbOut_CO_kg));
```

```
                    flight(od).Climb_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_ClimbOut_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_ClimbOut_HC_kg));
                    flight(od).Climb_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_ClimbOut_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_ClimbOut_NOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_ClimbOut_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_ClimbOut_SOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_ClimbOut_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_ClimbOut_CO2_kg));

                    flight(od).Descent_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_Descent_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_Descent_CO_kg));
                    flight(od).Descent_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_Descent_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_Descent_HC_kg));
                    flight(od).Descent_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_Descent_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_Descent_NOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_Descent_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_Descent_SOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_Descent_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*(LTO_Descent_CO2_kg));

                    flight(od).Taxi_Emissions_Single_Flight.CO_kg(aIndex)  =
Taxi_CO_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)* (Taxi_CO_kg));
                    flight(od).Taxi_Emissions_Single_Flight.HC_kg(aIndex)  =
Taxi_HC_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)* (Taxi_HC_kg));
                    flight(od).Taxi_Emissions_Single_Flight.NOx_kg(aIndex) =
Taxi_NOx_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*
(Taxi_NOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.SOx_kg(aIndex) =
Taxi_SOx_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*
(Taxi_SOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.CO2_kg(aIndex) =
Taxi_CO2_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(1)*
(Taxi_CO2_kg));
```

```
                elseif   strcmp(flight(od).originalAircraft(aIndex),'B737Max__') == 1
|| strcmp(flight(od).originalAircraft(aIndex),'B738Max__') == 1
||strcmp(flight(od).originalAircraft(aIndex),'B739Max__') == 1
||strcmp(flight(od).originalAircraft(aIndex),'E190E2__') == 1

                    flight(od).LTO_CO_Emission_kg(aIndex)  = LTO_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_CO_kg)) ;
                    flight(od).LTO_HC_Emission_kg(aIndex)  = LTO_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_HC_kg));
                    flight(od).LTO_NOx_Emission_kg(aIndex) = LTO_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_NOx_kg));
                    flight(od).LTO_SOx_Emission_kg(aIndex) = LTO_SOx_kg  -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_SOx_kg));
                    flight(od).LTO_CO2_Emission_kg(aIndex) = LTO_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_CO2_kg));

                    flight(od).Takeoff_Emissions_Single_Flight.CO_kg(aIndex)  =
Takeoff_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(Takeoff_CO_kg)) ;
                    flight(od).Takeoff_Emissions_Single_Flight.HC_kg(aIndex)  =
Takeoff_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(Takeoff_HC_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.NOx_kg(aIndex) =
Takeoff_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(Takeoff_NOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.SOx_kg(aIndex) =
Takeoff_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(Takeoff_SOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.CO2_kg(aIndex) =
Takeoff_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(Takeoff_CO2_kg));

                    flight(od).Climb_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_ClimbOut_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_ClimbOut_CO_kg));
                    flight(od).Climb_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_ClimbOut_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_ClimbOut_HC_kg));
                    flight(od).Climb_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_ClimbOut_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_ClimbOut_NOx_kg));
```

```
                    flight(od).Climb_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_ClimbOut_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_ClimbOut_SOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_ClimbOut_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_ClimbOut_CO2_kg));

                    flight(od).Descent_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_Descent_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_Descent_CO_kg));
                    flight(od).Descent_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_Descent_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_Descent_HC_kg));
                    flight(od).Descent_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_Descent_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_Descent_NOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_Descent_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_Descent_SOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_Descent_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*(LTO_Descent_CO2_kg));

                    flight(od).Taxi_Emissions_Single_Flight.CO_kg(aIndex)  =
Taxi_CO_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)* (Taxi_CO_kg));
                    flight(od).Taxi_Emissions_Single_Flight.HC_kg(aIndex)  =
Taxi_HC_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)* (Taxi_HC_kg));
                    flight(od).Taxi_Emissions_Single_Flight.NOx_kg(aIndex) =
Taxi_NOx_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*
(Taxi_NOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.SOx_kg(aIndex) =
Taxi_SOx_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*
(Taxi_SOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.CO2_kg(aIndex) =
Taxi_CO2_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(2)*
(Taxi_CO2_kg));

                elseif   strcmp(flight(od).originalAircraft(aIndex),'A330neo__') == 1

                    flight(od).LTO_CO_Emission_kg(aIndex)  = LTO_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_CO_kg)) ;
```

```
                    flight(od).LTO_HC_Emission_kg(aIndex)  = LTO_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_HC_kg));
                    flight(od).LTO_NOx_Emission_kg(aIndex) = LTO_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_NOx_kg));
                    flight(od).LTO_SOx_Emission_kg(aIndex) = LTO_SOx_kg  -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_SOx_kg));
                    flight(od).LTO_CO2_Emission_kg(aIndex) = LTO_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_CO2_kg));

                    flight(od).Takeoff_Emissions_Single_Flight.CO_kg(aIndex)  =
Takeoff_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(Takeoff_CO_kg)) ;
                    flight(od).Takeoff_Emissions_Single_Flight.HC_kg(aIndex)  =
Takeoff_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(Takeoff_HC_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.NOx_kg(aIndex) =
Takeoff_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(Takeoff_NOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.SOx_kg(aIndex) =
Takeoff_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(Takeoff_SOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.CO2_kg(aIndex) =
Takeoff_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(Takeoff_CO2_kg));

                    flight(od).Climb_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_ClimbOut_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_ClimbOut_CO_kg));
                    flight(od).Climb_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_ClimbOut_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_ClimbOut_HC_kg));
                    flight(od).Climb_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_ClimbOut_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_ClimbOut_NOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_ClimbOut_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_ClimbOut_SOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_ClimbOut_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_ClimbOut_CO2_kg));
```

```
                    flight(od).Descent_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_Descent_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_Descent_CO_kg));
                    flight(od).Descent_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_Descent_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_Descent_HC_kg));
                    flight(od).Descent_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_Descent_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_Descent_NOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_Descent_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_Descent_SOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_Descent_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*(LTO_Descent_CO2_kg));

                    flight(od).Taxi_Emissions_Single_Flight.CO_kg(aIndex)  =
Taxi_CO_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)* (Taxi_CO_kg));
                    flight(od).Taxi_Emissions_Single_Flight.HC_kg(aIndex)  =
Taxi_HC_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)* (Taxi_HC_kg));
                    flight(od).Taxi_Emissions_Single_Flight.NOx_kg(aIndex) =
Taxi_NOx_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*
(Taxi_NOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.SOx_kg(aIndex) =
Taxi_SOx_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*
(Taxi_SOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.CO2_kg(aIndex) =
Taxi_CO2_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(3)*
(Taxi_CO2_kg));

                elseif   strcmp(flight(od).originalAircraft(aIndex),'E195E2__') == 1

                    flight(od).LTO_CO_Emission_kg(aIndex)  = LTO_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_CO_kg)) ;
                    flight(od).LTO_HC_Emission_kg(aIndex)  = LTO_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_HC_kg));
                    flight(od).LTO_NOx_Emission_kg(aIndex) = LTO_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_NOx_kg));
                    flight(od).LTO_SOx_Emission_kg(aIndex) = LTO_SOx_kg  -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_SOx_kg));
                    flight(od).LTO_CO2_Emission_kg(aIndex) = LTO_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_CO2_kg));
```

```
                    flight(od).Takeoff_Emissions_Single_Flight.CO_kg(aIndex)  =
Takeoff_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(Takeoff_CO_kg)) ;
                    flight(od).Takeoff_Emissions_Single_Flight.HC_kg(aIndex)  =
Takeoff_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(Takeoff_HC_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.NOx_kg(aIndex) =
Takeoff_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(Takeoff_NOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.SOx_kg(aIndex) =
Takeoff_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(Takeoff_SOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.CO2_kg(aIndex) =
Takeoff_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(Takeoff_CO2_kg));

                    flight(od).Climb_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_ClimbOut_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_ClimbOut_CO_kg));
                    flight(od).Climb_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_ClimbOut_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_ClimbOut_HC_kg));
                    flight(od).Climb_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_ClimbOut_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_ClimbOut_NOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_ClimbOut_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_ClimbOut_SOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_ClimbOut_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_ClimbOut_CO2_kg));

                    flight(od).Descent_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_Descent_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_Descent_CO_kg));
                    flight(od).Descent_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_Descent_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_Descent_HC_kg));
                    flight(od).Descent_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_Descent_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_Descent_NOx_kg));
```

```
                    flight(od).Descent_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_Descent_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_Descent_SOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_Descent_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*(LTO_Descent_CO2_kg));

                    flight(od).Taxi_Emissions_Single_Flight.CO_kg(aIndex)  =
Taxi_CO_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)* (Taxi_CO_kg));
                    flight(od).Taxi_Emissions_Single_Flight.HC_kg(aIndex)  =
Taxi_HC_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)* (Taxi_HC_kg));
                    flight(od).Taxi_Emissions_Single_Flight.NOx_kg(aIndex) =
Taxi_NOx_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*
(Taxi_NOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.SOx_kg(aIndex) =
Taxi_SOx_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*
(Taxi_SOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.CO2_kg(aIndex) =
Taxi_CO2_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(4)*
(Taxi_CO2_kg));

                elseif   strcmp(flight(od).originalAircraft(aIndex),'A350-1000__') ==
1

                    flight(od).LTO_CO_Emission_kg(aIndex)  = LTO_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_CO_kg)) ;
                    flight(od).LTO_HC_Emission_kg(aIndex)  = LTO_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_HC_kg));
                    flight(od).LTO_NOx_Emission_kg(aIndex) = LTO_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_NOx_kg));
                    flight(od).LTO_SOx_Emission_kg(aIndex) = LTO_SOx_kg  -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_SOx_kg));
                    flight(od).LTO_CO2_Emission_kg(aIndex) = LTO_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_CO2_kg));

                    flight(od).Takeoff_Emissions_Single_Flight.CO_kg(aIndex)  =
Takeoff_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(Takeoff_CO_kg)) ;
                    flight(od).Takeoff_Emissions_Single_Flight.HC_kg(aIndex)  =
Takeoff_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(Takeoff_HC_kg));
```

```
                    flight(od).Takeoff_Emissions_Single_Flight.NOx_kg(aIndex) =
Takeoff_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(Takeoff_NOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.SOx_kg(aIndex) =
Takeoff_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(Takeoff_SOx_kg));
                    flight(od).Takeoff_Emissions_Single_Flight.CO2_kg(aIndex) =
Takeoff_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(Takeoff_CO2_kg));

                    flight(od).Climb_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_ClimbOut_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_ClimbOut_CO_kg));
                    flight(od).Climb_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_ClimbOut_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_ClimbOut_HC_kg));
                    flight(od).Climb_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_ClimbOut_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_ClimbOut_NOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_ClimbOut_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_ClimbOut_SOx_kg));
                    flight(od).Climb_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_ClimbOut_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_ClimbOut_CO2_kg));

                    flight(od).Descent_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_Descent_CO_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_Descent_CO_kg));
                    flight(od).Descent_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_Descent_HC_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_Descent_HC_kg));
                    flight(od).Descent_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_Descent_NOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_Descent_NOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_Descent_SOx_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_Descent_SOx_kg));
                    flight(od).Descent_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_Descent_CO2_kg -
(nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*(LTO_Descent_CO2_kg));
```

```
                    flight(od).Taxi_Emissions_Single_Flight.CO_kg(aIndex)  =
Taxi_CO_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)* (Taxi_CO_kg));
                    flight(od).Taxi_Emissions_Single_Flight.HC_kg(aIndex)  =
Taxi_HC_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)* (Taxi_HC_kg));
                    flight(od).Taxi_Emissions_Single_Flight.NOx_kg(aIndex) =
Taxi_NOx_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*
(Taxi_NOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.SOx_kg(aIndex) =
Taxi_SOx_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*
(Taxi_SOx_kg));
                    flight(od).Taxi_Emissions_Single_Flight.CO2_kg(aIndex) =
Taxi_CO2_kg - (nextgenacftFuelburnSavingFactors.Fuel_burn_efficiency(5)*
(Taxi_CO2_kg));

                else
                    flight(od).LTO_CO_Emission_kg(aIndex)  = LTO_CO_kg;
                    flight(od).LTO_HC_Emission_kg(aIndex)  = LTO_HC_kg;
                    flight(od).LTO_NOx_Emission_kg(aIndex) = LTO_NOx_kg;
                    flight(od).LTO_SOx_Emission_kg(aIndex) = LTO_SOx_kg;
                    flight(od).LTO_CO2_Emission_kg(aIndex) = LTO_CO2_kg;

                    flight(od).Takeoff_Emissions_Single_Flight.CO_kg(aIndex)  =
Takeoff_CO_kg;
                    flight(od).Takeoff_Emissions_Single_Flight.HC_kg(aIndex)  =
Takeoff_HC_kg;
                    flight(od).Takeoff_Emissions_Single_Flight.NOx_kg(aIndex) =
Takeoff_NOx_kg;
                    flight(od).Takeoff_Emissions_Single_Flight.SOx_kg(aIndex) =
Takeoff_SOx_kg;
                    flight(od).Takeoff_Emissions_Single_Flight.CO2_kg(aIndex) =
Takeoff_CO2_kg;

                    flight(od).Climb_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_ClimbOut_CO_kg;
                    flight(od).Climb_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_ClimbOut_HC_kg;
                    flight(od).Climb_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_ClimbOut_NOx_kg;
                    flight(od).Climb_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_ClimbOut_SOx_kg;
                    flight(od).Climb_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_ClimbOut_CO2_kg;
```

```
                    flight(od).Descent_Emissions_Single_Flight.CO_kg(aIndex)  =
LTO_Descent_CO_kg;
                    flight(od).Descent_Emissions_Single_Flight.HC_kg(aIndex)  =
LTO_Descent_HC_kg;
                    flight(od).Descent_Emissions_Single_Flight.NOx_kg(aIndex) =
LTO_Descent_NOx_kg;
                    flight(od).Descent_Emissions_Single_Flight.SOx_kg(aIndex) =
LTO_Descent_SOx_kg;
                    flight(od).Descent_Emissions_Single_Flight.CO2_kg(aIndex) =
LTO_Descent_CO2_kg;

                    flight(od).Taxi_Emissions_Single_Flight.CO_kg(aIndex)  =
Taxi_CO_kg ;
                    flight(od).Taxi_Emissions_Single_Flight.HC_kg(aIndex)  =
Taxi_HC_kg ;
                    flight(od).Taxi_Emissions_Single_Flight.NOx_kg(aIndex) =
Taxi_NOx_kg ;
                    flight(od).Taxi_Emissions_Single_Flight.SOx_kg(aIndex) =
Taxi_SOx_kg ;
                    flight(od).Taxi_Emissions_Single_Flight.CO2_kg(aIndex) =
Taxi_CO2_kg ;
                end

            end % Frequency ==0
        end % OD_Time(od).Incorrect_Flight(1, aIndex) < 0
    end % aIndex = 1: length(flight(od).Aircraft_Index_in_BADA)
end % OD_Time(od).Incorrect_Flight < 0
end% for od = 1 : flight_Size

%% Save Output
disp(' ')
disp('Saving Emissions Calculations.. ')
save ([Save_Dir, '\flight_4_', Case_Year_Str, '.mat'], 'flight', '-v7.3');

disp('Saved.. ')
disp(' ')
return;
```