Global Demand Model to Estimate Supersonic Commercial Services

Edwin R. Freire Burgos

Dissertation submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In

Civil Engineering

Antonio A. Trani, Chair
Susan Hotle
Gerardo W. Flintsch
Linbing Wang

September 24, 2021

Blacksburg, VA

Keywords: Supersonic Flights, Forecast Demand, Travel Time Savings, Low-Boom Aircraft, Value of Time,

Global Demand Model to Estimate Supersonic Commercial Services

Edwin R. Freire Burgos

ABSTRACT

Not too long ago, commercial supersonic aircraft flights were part of the air transportation system. In the 1970's we had the Russian-built Tupolev Tu-144 and the BAC/Aerospatiale Concorde, the latest being tin operation for 27 years. The work documented in this dissertation focused on the viability of bringing back supersonic aircraft as a transportation mode. Throughout three years, Virginia Tech and a team from NASA have been combining efforts to develop a model capable of predicting future air travel demand for supersonic vehicles. The model can predict future supersonic commercial services and allows aircraft designers from NASA to optimize aircraft performance and characteristics by maximizing the potential air travel demand.

The final product of this study was the development of the Low-Boom Supersonic Aircraft Model (LBSAM). The development progress took three years to be completed, and during each year, a version of the model with the preliminary predictions was made available to NASA. Each of the three versions of the model predicts future supersonic commercial services. What differentiates each version is the data, method, and aircraft type/design implemented; the latest version of the model is more realistic and provides a higher number of functionalities.

The first version of the model predicted the possible supersonic commercial service for three aircraft types: each with two variations. An 18-seat, 40-seat, and 60-seat low-boom and non-low-boom aircraft were analyzed. The second version of the model analyzed a 20-seat and 40-seat low-boom, non-low-boom aircraft with restrictions and non-low-boom aircraft without restrictions. The latest version of the model tries to estimate potential demand for a 43-seat and a 52-seat supersonic low-boom aircraft design. The low-boom concept refers to the implementation of technology that reduces the loudness of a sonic boom. A non-low-boom concept refers to an aircraft flying faster than Mach 1 with the technology's implementation that reduces the loudness of a sonic boom. The final results suggest that for a 52-seat LBSA, the potential worldwide demand is as follows.

- 33.4 million seats worldwide. Assuming an overland range of 3,200 nm., an overland Mach 1.7, and an overland fuel scale factor of 0.98.
- 772 aircraft needed worldwide. Assuming an overland range of 2,800 nm., an overland Mach 1.7, and an overland fuel scale factor of 0.90.
- 1,032 one-way OD pairs where LBSA can operate. Assuming an overland range of 2,800 nm., an overland Mach 1.7, and an overland fuel scale factor of 0.90.

The LBSAM is mainly driven by the cost per passenger mile values calculated for each one-way Origin-Destination (OD) pair. Additional uncertainties in the model include the market share and annual aircraft utilization. The market share refers to the percent of the demand that will switch from current subsonic commercial services to commercial supersonic services. During the three-year work, we considered a market share of 50% and 100%. Aircraft utilization refers to the number of hours that the airline will be able to use the aircraft. The majority of the projections were based on a 3,500-hour aircraft utilization.

Global Demand Model to Estimate Supersonic Commercial Services

Edwin R. Freire Burgos

GENERAL AUDIENCE ABSTRACT

Not too long ago, commercial supersonic aircraft flights were part of the air transportation system. An aircraft flying faster than the speed of sound is known as an aircraft flying at supersonic speed. Current commercial aircraft fly at subsonic speed. Subsonic speed refers to aircraft flying at a speed lower than the speed of sound. In the 1970's we had the Russian-built Tupolev Tu-144 and the BAC/Aerospatiale Concorde, the latest being tin operation for 27 years. The work documented in this dissertation focused on the viability of bringing back supersonic aircraft as a transportation mode. Throughout three years, Virginia Tech and a team from NASA have been combining efforts to develop a model capable of predicting future air travel demand for supersonic vehicles. The model can predict future supersonic commercial services and allows aircraft designers from NASA to optimize aircraft performance and characteristics by maximizing the potential air travel demand.

The purpose of this dissertation effort is to provide a better understanding of what could be the potential commercial demand for supersonic flight in the near future. We consider all the benefits and characteristics of supersonic flight and studied in detail what percentage of the travelers might be willing to migrate from the current subsonic market to the supersonic market. We estimated this ratio by studying the spending behavior of passengers in the current market. How much more are passengers willing to pay to save time? We can infer how much travelers value their time by comparing direct flights versus flights with an intermediate stop.

The results show that a demand of 33.4 million seats could be reached by the year 2040. The supersonic market would consist of more than one thousand one-way origin-destination pairs worldwide, and more than seven hundred supersonic aircraft are expected to satisfy the forecast demand.

Acknowledgments

First, I would like to express my sincere gratitude to Dr. Trani for his advice and mentorship throughout all these years. He has been an excellent professor and mentor. I thank you for all the help, advice, and knowledge you shared with me. Your expertise in the air transportation field has been a great motivation for me.

I want to thank Nicolas Hinze for teaching me how to improve the model's source code and make it more efficient. In addition, I would like to express my gratitude to my committee members Dr. Hotel, Dr. Flintsch, and Dr. Wang, for their guidance.

Special gratitude is given to Ty Vincent Marien and Sam Dollyhigh, who are part of the NASA Langley Research Center team, for their guidance and input throughout the project. I am grateful for working as part of this great team.

Finally, yet importantly, I would like to express my deepest thanks to my family, parents, and sister, who believed in me way before all this was possible. I am who I am, thanks to them.

Table of Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

The work presented in this dissertation is the recompilation of three years of work effort between Virginia Tech and a team from the National and Space Administration (NASA). At the beginning of this research project, the objective was to estimate the global demand for commercial supersonic services. After model development, NASA received an interactive demand forecast model. This tool helps aircraft design engineers optimize their designs by better understanding aircraft parameters' effect on potential supersonic commercial demand.

NASA received preliminary reports at the end of each of the three years [1] [2] [3]. In this document, those three reports are being labeled as Study 1, 2, and 3. These studies contain an analysis of supersonic aircraft commercial operations. The research presented quantifies the number of aircraft needed to fill the demand for supersonic air travel and the number of annual aircraft operations supported in a projected supersonic market. Several supersonic transport aircraft concepts are considered for introduction into the fleet. Even though all three studies have a final objective to estimate the potential supersonic future demand, aircraft concepts, databases, assumptions, and methodology in each study are different.

Table 1 presents a general description of two of the main parameters used in each of the studies. During Study 1, described in Chapter 2, the model analyzed three supersonic aircraft concepts, each with three different seating capacities. For Study 2, described in Chapter 3, three supersonic aircraft concepts were explored, each with two different seating capacities. Study three, presented in Chapter 4, analyzes one supersonic aircraft concept with two different seating capacities.

A high impact assumption in the analyses is the market share, representing what percentage of the demand will decide to choose supersonic over subsonic service. The concept of market share is discussed later in this document. For Study 1, we used a moderate, 50%, and a very optimistic, 100% market share. For Study 2 and Study 3, only a medium market share of 50% was implemented. The development of LBSAM is a group effort between Virginia Tech and NASA. Dr. Trani implemented the Life Cycle Cost model used in the LBSAM. Table 2 shows the assumptions used in each study to determine if a route was assigned with overland or overwater

characteristics. Table 3 presents a summarized description of the value of time analysis's significant differences for each study.

Table 1: Supersonic Aircraft Concepts Analyzed

Study	Aircraft Seating Capacity	Supersonic Aircraft Concept		
		Low-Boom	Non-Low-Boom	Non-Low-boom with Restriction
1	18, 40 & 60	✓	✓	✓
2	20 & 40	✓	✓	✓
3	43 & 52	✓		

Table 2: Travel Time Module – Overland and Overwater Route Characteristic Assumption by Study.

Study	Overland Characteristics	Overwater Characteristics
1	Assumed only for US Market	Assumed for US-Int. and International Markets
2	Assumed for all OD pairs with up to 2,500 nmi.	Assumed For all OD pairs over 2,500 nmi.
3	Flight trajectory – 25% or more overland	Flight trajectory – over 75% overwater

Table 3: Value of Time Analysis – Description of Main Parameters by Study

Study 1 – ARC 2012	Study 2 – ARC 2012	Study 3 – ARC 2016
• 56 OD pairs analyzed • Over 100,000 Premium records • One hour stopover for one-stop records • Average fares and travel times • Single VOT value depending on route distance.	• Over 2 million records • Economy Premium records • Premium Records • Organize the data in three markets: US, US-Int., and International. • 1-hour layover for one-stop records within the US market. • 2-hours layover for one-stop records for the US-Int. and International markets • VOT value calculated for all available OD pairs. • A VOT value per market was calculated using the	• Forty-seven million records were analyzed. • Premium records • Organize the data in three markets: US, US-Int., and International. • Implemented a network analysis using OAG 2016 reported travel times to account for layover times in one-stop records. • We calculated weighted average VOT values. • Weighted average for airport VOT value. • Weighted average close by airport VOT values (within 30 miles radius)

Study 1 – ARC 2012	Study 2 – ARC 2012	Study 3 – ARC 2016
	50th percentile VOT value of each market.	• Weighted average OD pair VOT value. • $400/hr. VOT limit

1.1 Background and Motivation

Today, all commercial flights are conducted at subsonic speeds. A subsonic aircraft speed refers to a speed below the speed of sound. The speed of sound is a term used to describe a physical constant for any given medium at a specific pressure and temperature [4]. Recently, there has been activity in developing aircraft concepts with supersonic speed capabilities. However, this is not the first time we have heard of supersonic flights. The first aircraft to fly at supersonic speeds was the Bell X-1 on October 14, 1947 [5]. The Bell X-1 did not take off from the ground. It was dropped from the belly of a Boeing B-29 mothership at an altitude of 23,000 ft. After launch, it reached an altitude of 43,000 ft. and a top speed of Mach 1.06. Subsequently, military aircraft were built with a typical speed limitation of Mach 2.5.

As for supersonic commercial flights, in December 1968, the Russian-built Tupolev Tu-144 had its first maiden flight. This aircraft was 215.6 ft. in length, with a wingspan of 94.5 with a typical cruising speed of Mach 2.2. The Tu-144 had its first supersonic flight in June 1969, and it was first publicly shown in Moscow in May 1970. This aircraft had a difficult start when the first production, Tu-144, crashed at the 1973 Paris Air Show. The aircraft was put into commercial service, flying mail in 1975 between Moscow and Alma Ata and then passengers in 1978. The aircraft was pulled from service after the second crash in 1978 [6]. It performed 102 commercial flights, of which only 55 carried passengers.

The second and better-known supersonic commercial aircraft was the BAC/Aerospatiale Concorde. The famous British-French turbojet-powered supersonic passenger airliner had its first maiden flight in March 1969. In November 1962, the United Kingdom and France agreed to work together to produce a supersonic transport (SST). The French firm Aerospatiale was responsible for the aircraft airframe. At the same time, Britain's Rolls-Royce and *"France Societe Nationale d'Etude et de Construction de Moteurs d'Aviation"* developed the aircraft engines. The Concorde had a maximum cruising speed of Mach 2.04. The first transatlantic flight was in September 1973,

with the first scheduled supersonic passenger service in January 1976. Even though the aircraft travel timesaving benefit was noticeable, its noise and operating cost greatly limited its service. The development cost could have never been recovered from operations, and it was not financially profitable. However, the European government's pride will keep them at the technical forefront of aerospace development. On July 25, 2000, a Concorde flight from Paris to New York suffered engine failure shortly after takeoff when debris from a burst tire caused a fuel tank rupture and a fire. The aircraft crashed into a small hotel and restaurant. All 109 persons on board, including 100 passengers and nine crew members, died, four people on the ground were also killed [7]. Twenty aircraft were built, including six prototypes and development aircraft, with British Airways and Air France being the only two operating airlines. After 27 years in service, the Concorde ceased operations in October 2003.

After a recent interest in bringing back commercial supersonic services, some work has been done regarding potential demand for this market. Literature review on similar research work where the objective is to estimate potential demand for supersonic resulted in the following. A group of researchers from Georgia Tech published the work done to evaluate the supersonic commercial market [8]. The results from this group estimate that there will be enough SST demand to account for 667 flights per day in the year 2050, with a market share of 100%. These flights would operate in 843 routes. The SST aircraft concept used was the 55-passengers Boom Overture. As a criterion, to identify a potential OD route for SST service, the minimum travel time saving must be at least 1.5 hours. There must also be enough premium (business and first-class) demand to support four round-trip flights per week. This would set the minimum demand to 11,440 annual seats. The database used in this analysis consisted of IATA and OAG. The demand forecast was based on IATA 2018 Market Intelligence Services and is flight operations and revenue passenger kilometer dependent.

Another research work found during the literature review was JAXA’s S4 Supersonic Low-Boom Airliner - A Collaborative Study on Aircraft Design, Sonic Boom Simulation, and Market Prospects [9]. The group analyzed a 50-seat and 36-seat SST aircraft concept. The results indicate that there could be 285 aircraft needed worldwide with 534 daily flights for the 50-seat aircraft concept. For the 36-seat aircraft concept, the group estimated 407 aircraft needed worldwide with 808 daily flights. The database used for the forecast demand in this analysis was Market

Intelligence by Sabre for the baseline year. From there, the group interpolates the premium passengers into future demand by using inter-regional passenger number growth prediction by the Boeing Commercial Market Outlook. The target year of this analysis is 2033, 18 years from their baseline case. The SST concept studied in this project assumes a range of 3,800 nm. at a supersonic speed of Mach 1.6 and 4,300 nm. at a subsonic speed of Mach 0.95. The aircraft utilization in this analysis was set at 10 hours per day. This is equivalent to an annual aircraft utilization of 3,650 hours. The subsonic travel times were calculated, assuming a constant speed of Mach 0.85. The study uses three different market shares: 10%, 25%, and 50%. The criteria established to identify a route as a potential SST candidate are the following. OD pairs with at least 20,000 minimum premium passengers per year and 1 flight per day. In addition, the researcher limited the study to those OD pairs with a maximum distance of 3,800 nm.

Additional research work found in the literature review shows a strong interest in the supersonic aircraft topic. Reference [10] proposes a model to predict routes, the number of operations, and the fleet-level impact of future commercial supersonic aircraft on routes touching the United States. The paper mention that one of the main assumptions in the model is that it considers that five percent of passengers on a route are the only potential supersonic passengers. The paper concludes that there are 205 possible routes for supersonic operations.

Reference [11] presents a design tool for a conceptual analysis of future commercial supersonic aircraft. The paper discusses the technical feasibility and economic viability of currently proposed commercial supersonic concepts. The research relies on publicly available Concorde data and focuses on supersonic aircraft concepts' performance and characteristics.

Reference [12] discuss the estimation of the market potential for supersonic airliners via analysis of the global premium ticket market. The paper concludes that there might not be enough demand to support large supersonic aircraft operations with seating capacity in the hundreds. However, there is a potential market for smaller supersonic aircraft with a seating capacity of around 20 seats. The research suggests that a small percentage of passengers willing to change to supersonic service – between 15% and 30%- could be sufficient for economic success.

Reference [13] proposes exploring small supersonic airliners' prospects using the Aerion AS2 jet as a business case study. The paper outline the economic and operational challenges of supersonic

airliners. The study concludes that the optimal path for modern supersonic aircraft to become a reality begins with small aircraft, such as the Aerion AS2 jet.

Reference [14] describe a perceived noise analysis for offset jets applied to commercial supersonic aircraft. The study focus on experimental jet noise data, engine performance codes, and aircraft noise predictions codes to assess takeoff noise levels and mission range for conceptual supersonic commercial aircraft. The study concludes that two types of engines could provide acceptable mission range performance for a conceptual 35 to 70 passenger commercial supersonic aircraft. A conventional mixed flow turbofan or a three-stream variable cycle engine.

Reference [15] discuss a supersonic vehicle system for the 2020 to 2035 timeframe. The paper proposes a concept aircraft called the Icon-II, which can meet or exceed specific aircraft performance and characteristics. The study focused on environmental and performance goals such as sonic boom, airport noise, cruise emissions, cruise Mach, range, payload, and fuel efficiency.

An extensive literature review can be found on commercial supersonic aircraft research. However, the available knowledge focus on the technical side of the aircraft or environmental issues with little to no information on passenger willingness to pay and demand. Based on the literature review, we determined the need for a tool to analyze the effect of multiple aircraft parameters on demand. The LBSAM is a tool that can help determine the best aircraft characteristic and performance to maximize potential market during the aircraft design process. The methodology, details, and factors considered making this model different from what others have already done.

1.2 Recent Supersonic Aircraft Concepts

Several supersonic aircraft concepts have been investigated in the last fifty years since the development of the British Aerospace/Aerospatiale Concorde and the larger Boeing Supersonic Transport 2707 SST concept in the sixties. The Concorde operated for nearly three decades in limited routes between North America (New York) and Europe (London and Paris). In the eighties, several corporate supersonic transport aircraft studies were carried out by Sukhoi (1988), Gulfstream/Sukhoi (1989), Lockheed Martin (2000), Aerion (2002), Dassault (2003), Tupolev (2004), Raytheon (2005), and more recently Aireon/Airbus (2014).

Previous studies focused on the business jet market because it was perceived that a high cost per mile could only be justified on the grounds of travel time savings for high-income earners. Such studies provide insight into various approaches taken by companies to develop supersonic technology. More recent efforts have shifted to the application of supersonic technology to either conversion of corporate conceived aircraft such as the Aerion2 or larger capacity SST designs such as the Boom and Boeing 765-107 concept studied for NASA [16] [17]. Table 4 shows some basics performance characteristics of four recent SST concepts.

Boom Technology Inc., better known as Boom Supersonic, is a US-based company targeting the supersonic commercial service market. Boom is one of the three companies seeking to bring back this type of service to the public. The other two companies are Aerion and Spike. While the latest are targeting the private business jet market, Boom is trying to capture the airlines' premium market. Boom is developing a 55-seat, three-engine SST [18]. This aircraft concept is known as the Boom Overture and is expected to be introduced in 2025. The Overture will have a cruising speed of Mach 2.2 at an altitude of 60,000 feet with a range of 4,500 nm. The company projects the need for 1,000 to 2,000 aircraft over the first ten years (2025-2035). Even with overland restrictions for supersonic flights due to noise pollution, the company claims that about 500 routes are economically viable. Boom claims that a ticket would cost about $5,000 for transoceanic flights between the U.S. and Europe [19]. For a comparative perspective with the work presented in this dissertation, the $5,000 ticket would translate to a $1.67/nm. for the New York to London (JFK-LHR) or a $1.05/nm. for the Los Angeles, CA to London (LAX-LHR). In Table 4, BFL refers to Balance Field Length; TOFL refers to Take-Off Field Length.

Table 4: Recently Proposed Supersonic Transport Aircraft

Vehicles	Aerion Aerospace AS2	Spike Aerospace S-512	Boom Overture	Boeing 765-107B
Cruise Speed	Mach 0.95 LRC Mach 1.1-1.2 Boomless Mach 1.4 LRC Mach 1.5 Max.	Mach 0.95 Mach 1.6 Typical Mach 1.8 Max.	Mach 0.95 LRC[1] Mach 2.2 LRC	Mach 1.8 Max.
Range	5,300 nm. at Mach 0.95 4,750 nm. at Mach1.4	6,200 nm. at Mach 1.6	4,500 nm.	5,600 nm. at Mach 1.6 5,350 nm. at Mach 1.8
Passengers	6-9 Executive 13-15 Business Class	12-18	50	50 Premium 90 Mixed 120 Economy
Dimensions	Length 148 ft. Wingspan 87 ft.	Length 134 ft. Wingspan 58 ft.	Length 170 ft. Wingspan 60 ft.	Length 241 ft. Wingspan 165 ft.
Others	BFL[2] 7,500	TOFL 6,000 ft.	TOFL 10,000 ft.	BFL 10,000 ft.

Besides the development, manufacture, and operation high cost of supersonic aircraft, noise is a critical barrier to supersonic aircraft operations. The path for supersonic aircraft will be challenging, but a lack of policies for supersonic flights will be a high wall to pass. These regulatory challenges are mainly related to noise and emissions of commercial supersonic operations. It is known that once the sound barrier is broken, meaning exceeding the speed of Mach 1, a sonic boom is generated. This sonic boom term refers to a thunder-like noise a ground observed hears when a supersonic aircraft flies overhead faster than the speed of sound. As objects traveling at supersonic speed through the air, molecules are pushed aside with great force, and this forms a shock wave, much like a boat creates a wake in water: the bigger and heavier the aircraft, the more air it displaces. The shock wave forms a “cone” of pressurized or built-up air molecules, which move outward and rearward in all directions and extend all the way to the ground. As this cone spreads across the landscape along the flight path, it creates a continuous sonic boom along the entire width of the cone's base. After the buildup by the shock wave, the quick release of pressures

[1] LRC - Long Range Cruise

[2] BFL – Balanced Field Length

is heard as the sonic boom [20]. Therefore, the U.S. and several other countries currently ban overland commercial supersonic flight [21].

1.3 Worldwide Commercial Airline Market

The first study (Study 1) assumes that future supersonic services are confined to route distances greater than 1,500 statute miles. That assumption was relaxed in Studies 2 and 3. On average, commercial airlines scheduled 93,602 daily flights worldwide, according to the Official Airline Guide 2016. Trips greater than 1,500 statute miles constitute 13% of all the commercial flights scheduled during that year [22]. Trips greater than 1,000 statute miles include 24% of all the commercial flights scheduled during that same year. Figure 1 shows a distribution of flight segments Worldwide. Figure 2 shows the number of seats provided by each distance segment. If the range of a supersonic aircraft is limited to segments greater than 1,000 statute miles and up to 5,000 statute miles, 30% of all seats offered worldwide fall within this segment category. If the range of a supersonic aircraft is limited to segments greater than 1,500 statute miles and up to 5,000 statute miles, 16% of all seats offered worldwide fall within this segment category. If the range of a supersonic aircraft is limited to all segments greater than 1,000 statute miles, 33% of all seats offered worldwide fall within this segment category. If the range of a supersonic aircraft is limited to all segments greater than 1,500 statute miles, 19% of all seats offered worldwide falls within this segment category. Thirty percent and thirty-three percent of the global seats equates to 1.5 and 1.6 billion seats in 2016, respectively.

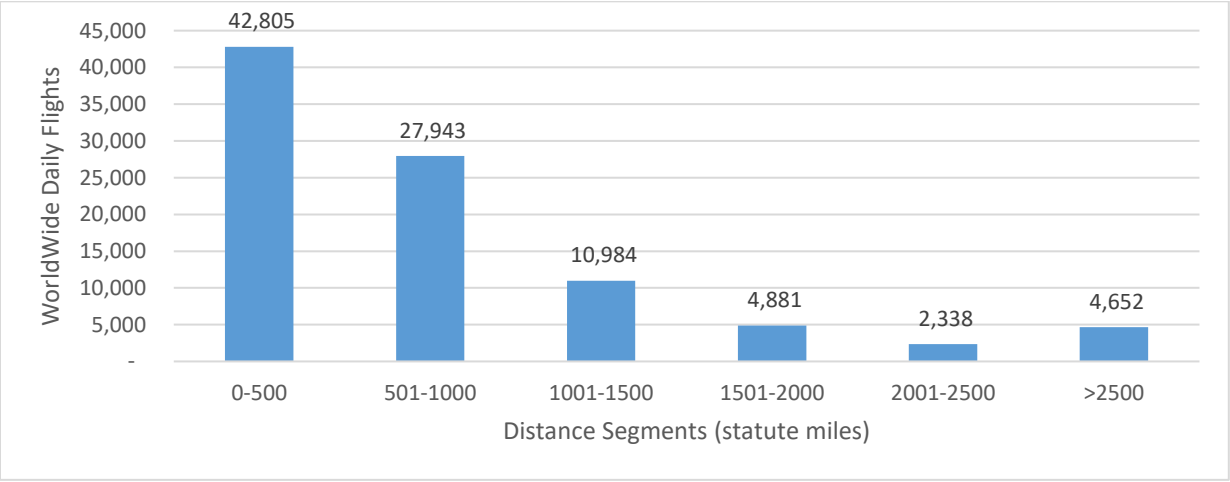


Figure 1: Distance Distribution of Daily Flights Worldwide. Source of data: OAG 2016.

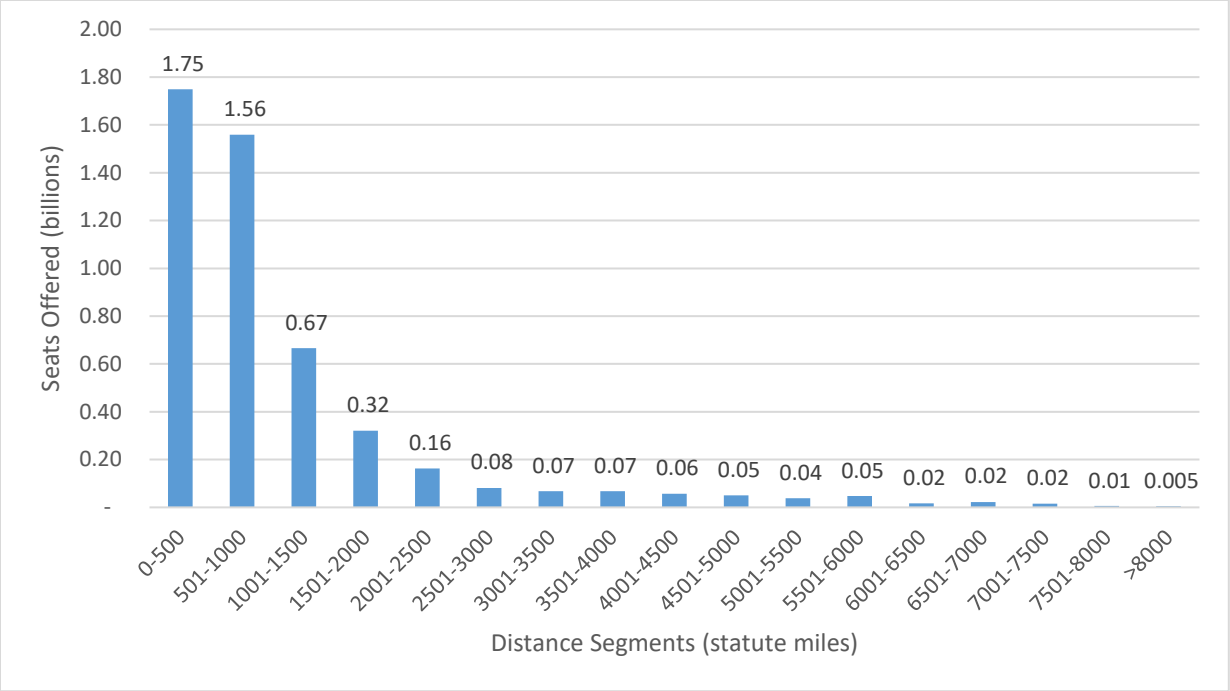


Figure 2: Distance Distribution of Seats Offered Worldwide. Source of data: OAG 2016.

Figure 3 shows the number of one-way origin-destination (OD) airport pairs with a significant number of seats offered. More than 4,000 one-way OD pairs worldwide provide more than 100,000 seats annually. Our survey of aircraft seating configurations shows that, in long-range flights, 3.6% of the seats in every flight are first-class seats, 15.6% of the seats are business class, and the remaining 80.8% of the seats are economy and premium economy seats. The numbers are significant because they quickly estimate the potential routes worldwide covered by supersonic aircraft. OAG 2016 data contains 58,597 one-way OD pairs, of which up to 4,425 could be the potential market for supersonic transport. Eight percent of the worldwide market accounts for one-way OD pairs with a distance greater or equal to 1,000 statute miles and at least 100,000 total seats offered per year. A detailed market demand analysis that considers the economics of the SST, travel time savings benefits, premium seats, and value of time, among other parameters, is described in the following sections.

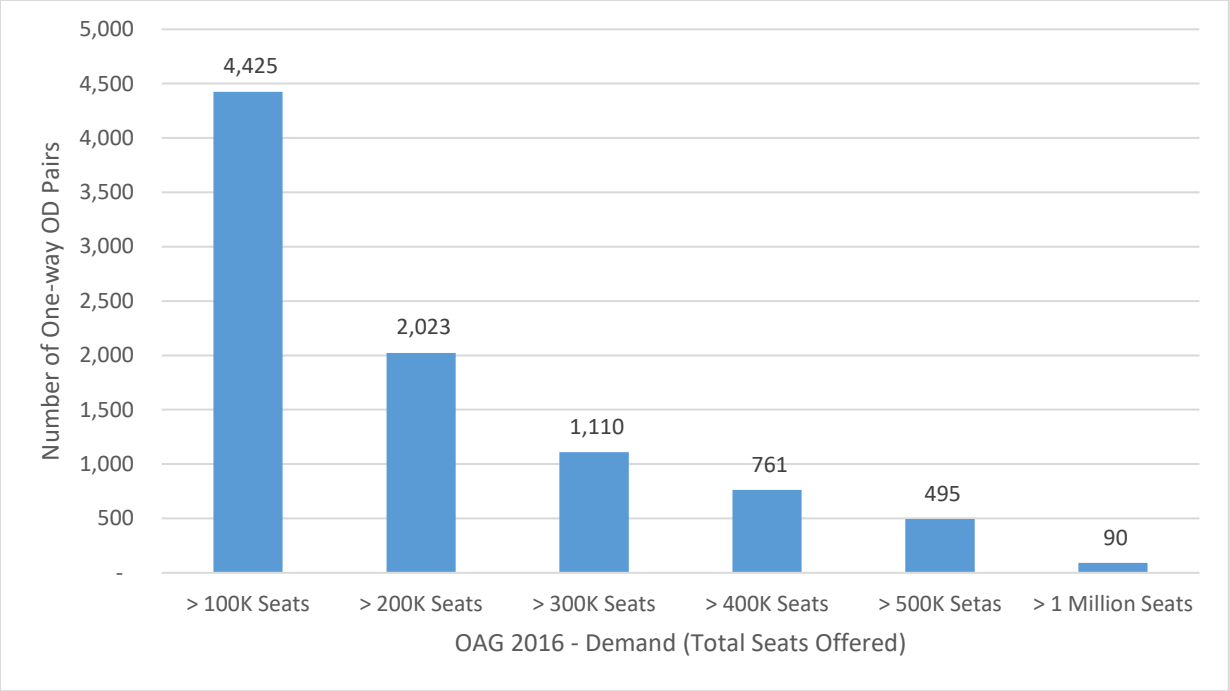


Figure 3: Number of Potential Worldwide Origin-Destination Markets vs. Number of Annual Seats Offered for Segments greater than 1,000 statute miles. Source of data: OAG 2016.

Table 5 shows the breakdown of OD pairs as a function of distance and annual premium seats offered. For example, there are 169 OD pairs between 3,001-3,500 statute miles that provide more than 100,000 premium seats. The figure shows that the number of OD pairs with a significant number of premium seats declines as route distance increases. Figure 4 shows the distance distribution of premium seats offered worldwide according to OAG 2016 data. This research project focuses on one-way OD pairs with distances greater than 1,000 statute miles, for Study 2 and Study 3, due to travel time-saving benefits. Study 1 was concentrated on one-way OD pairs with distances greater than 1,500 statute miles

Table 5: Breakdown of One-way OD Airport Pairs vs. Number of Premium Seats Offered per Distance Segment in Statute Miles. Source of data: OAG 2016.

Distance (st mi)	Supply of Seats											
	>100k		>200k		>300k		>400k		>500k		>1M	
	First	Business	First	Business	First	Business	First	Business	First	Business	First	Business
One-way OD Pairs	834		384		205		161		104		22	
1,500-2,000	5,490,213	10,587,305	4,563,435	8,107,085	3,789,313	6,189,469	3,384,415	5,582,655	2,813,798	4,003,204	985,408	1,591,184
One-way OD Pairs	431		172		112		72		54		8	
2,001-2,500	3,063,847	7,062,850	2,337,834	5,034,205	1,942,514	4,140,772	1,447,972	3,440,377	1,179,170	2,911,265	144,186	1,019,582
One-way OD Pairs	235		100		52		34		20		6	
2,501-3,000	1,443,815	4,800,082	1,109,753	3,438,038	821,507	2,601,302	677,269	2,168,498	584,515	1,459,564	299,352	525,819
One-way OD Pairs	169		81		50		38		33		6	
3,001-3,500	794,563	6,738,028	683,552	5,464,637	626,841	4,709,311	562,213	4,148,081	525,599	3,891,236	253,467	1,661,514
One-way OD Pairs	195		95		54		33		26		0	
3,501-4,000	728,513	6,404,603	537,287	4,991,348	405,083	3,780,775	337,189	2,873,576	297,763	2,433,446	-	-
One-way OD Pairs	179		68		46		28		16		0	
4,001-4,500	597,336	4,648,127	383,986	2,968,207	335,142	2,416,378	220,354	1,649,811	209,752	1,101,437	-	-
One-way OD Pairs	162		61		31		8		4		0	
4,501-5,000	286,308	3,778,106	179,124	2,167,669	120,472	1,410,397	43,300	477,009	1,428	235,002	-	-

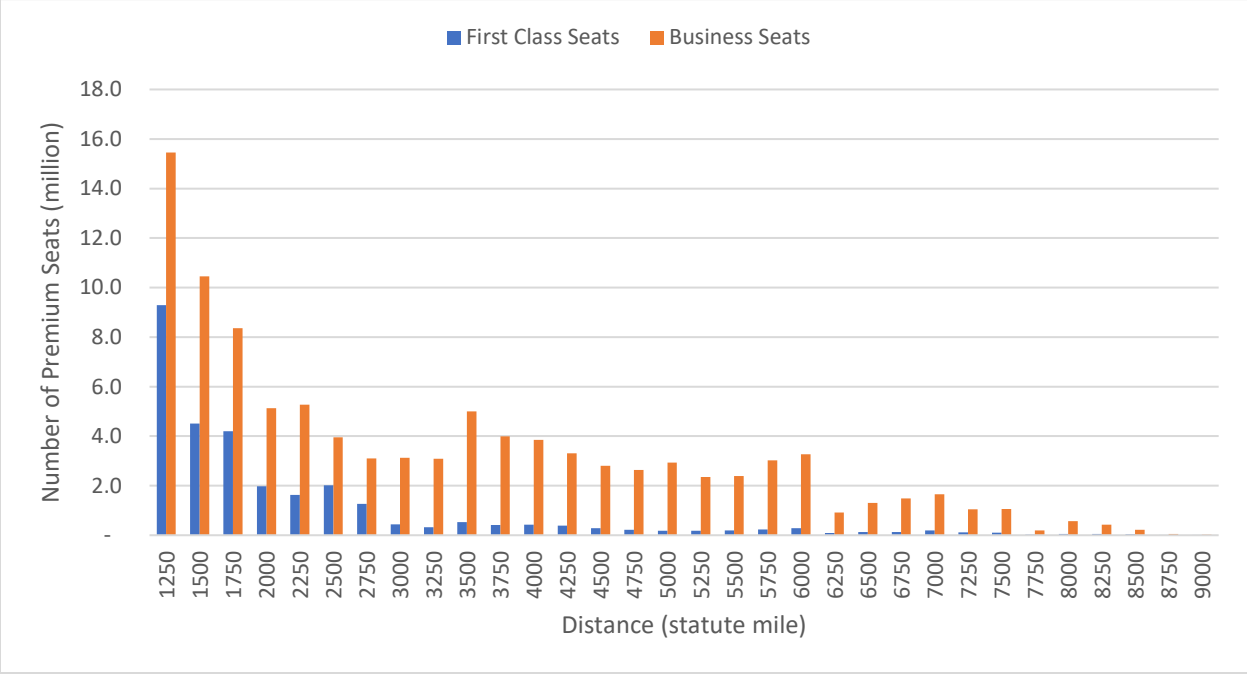


Figure 4: Distance Distribution of Premium Seats Offered Worldwide. Source of data: OAG 2016.

1.4 Databases

1.4.1 Airlines Reporting Corporation Data

Airline Reporting Corporation (ARC) data from two different years were used during the three years of work done for this dissertation. For Studies 1 and 2, ARC 2012 was analyzed. For Study 3, the Air Transportation System Laboratory (ATSL) had access to ARC 2016 data.

The ARC data is a large sample of passenger ticket records for tickets sold in the US. Domestic and international tickets are included in the data. Virginia Tech had access to more than 300 million records of ARC data for the year 2012. Most of these records are for the US market and the economy, economy premium, business, and first-class tickets are included. ARC data contains fares paid for flying different routes around the world. Using the ARC database information, we can estimate the percent of passengers who paid fares above a desired threshold (e.g., tickets whose fare exceeds $0.90 per seat-mile). Whereas the data is six years old, premium fares charges by airlines have not changed appreciably.

Similarly, when adjusted for inflation, United States domestic fares at 417 commercial airports have decreased by 10.3% between 2012 and 2016. Actual fare data helps us measure the price passengers (both business and first-class) are willing to pay for premium airline seats. Fares paid in premium services offer an attractive alternative to estimate the percentage of passengers willing to pay for future supersonic commercial services. The two-analysis done with this data is the Value of Time analysis and the Market Fare per Seat-Mile Analysis. These two analyses will be described in detail in their corresponding sections.

For Study one, we decided to use only ARC records from the premium market (business and first-class). Once again, the rationale behind this is that only the high-income earners would be able to afford supersonic services. After the results from Study 1, for Study 2, it was decided to include the ARC records from the economy-premium class market. Study 2 showed that a relatively small portion of demand could potentially come from the economy-premium market. The ARC 2012 was given to the ATSL for academic use, while the ARC 2016 was purchased. The data cost depended on the size of the data (total number of records to be delivered). For this reason and based on the findings of Study 1 and Study 2, for Study 3, it was decided to focus the analysis on premium class records and neglect the economy premium records.

1.4.2 Official Airline Guide 2016

The OAG data contains information about commercial schedule services. OAG provides origin-destination information, the frequency of flights, the number of seats offered (by class), and the equipment used to fly each origin-destination airport pair. OAG 2016 is used for the forecast baseline year to identify the number and percent of premium seats compared to the total number of seats supplied for each one-way OD pair and to identify sub-sonic gate-to-gate travel times. As mentioned before, we focused on premium demand for Study 1 and Study 3, which only includes business and first-class. For Study 2, besides business and first-class, the economy premium markets were also taken into consideration. While the premium seat to total seats offered ratio can be estimated from the OAG data, the economy premium percent is calculated using the Seat Guru website. Figure 5 shows an example of the seat-class configuration of a Boeing 777-300ER from China Airlines with 358 seats [23]. For this case, 17% of the seats were economy premium, while 11% were business class seats. After looking at multiple airlines' aircraft seat configurations for narrow and wide-body aircraft, the economy premium to total seat ratios is as follows. The continental US market's economy premium seat percent is 20% and 17% for both the US-International and International markets. The percentage of the economy premium (only for Study 2) and premium seats is used to estimate the number of these seats in the future by combining OAG 2016 and ICAO forecast. The ICAO forecast is explained in detail in one of the sections of this document. Potential growth patterns need to be estimated at the OD level because not all potential OD pairs in the network show historical growth.

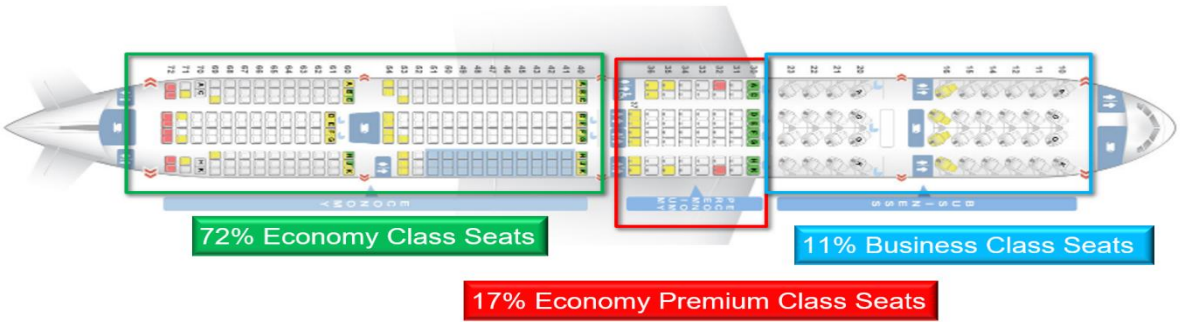


Figure 5: Example of a Boeing 777-300ER from China Airlines. Source of data: Seatguru.

1.5 Contribution to Knowledge in Commercial Supersonic Aircraft Research Area

Research work on commercial supersonic aircraft has increased over the past years. Literature review shows a strong interest in this type of service, which used to be part of the air transportation network two decades ago. The majority of the effort on supersonic aircraft research has been focused on aircraft characteristics and performance. Researchers are trying to find innovative methods to reduce the environmental impact that supersonic aircraft presents. Although excellent work is occurring to improve supersonic aircraft, little effort has been put towards realistically identify potential markets for commercial supersonic flights.

The potential demand for commercial supersonic services is assumed to come from the premium subsonic market. The premium market refers to current business class and first class travelers. The assumption is based on the expected high cost of operating supersonic aircraft. The literature review shows that effort to identify potential markets for commercial supersonic services is based on simply assuming that a fixed percentage of the premium subsonic market will shift to the supersonic market. Most research presents the potential demand if 5%, 10%, 25%, 50%, or even 100% of the premium subsonic demand moves to the supersonic market. Advanced research put additional effort into identifying potential routes by using supersonic flights' travel time saving benefits.

However, a gap in the literature review was exposed. There was a need to study in detail what could be the potential demand, why it would be generated, and how it could be identified. This dissertation work filled the gap found in the literature review. The LBSAM model is a tool that produces commercial supersonic demand forecast by analyzing, among several variables, air travelers' behavior by combining two main factors. The first factor is the value of time; how much travelers value their time based on historical data. The second factor is travel time savings offered by supersonic aircraft. By combining travel time benefits with how much travelers value their time, we estimate what fraction of the travelers could be willing to pay for supersonic flights. This exercise is done at the one-way origin-destination airport pair level for the worldwide market. The level of detail involved in the analysis adds up to the uniqueness of this dissertation.

The LBSAM is a unique tool that predicts future supersonic commercial services and allows aircraft designers from NASA to understand the causal links between aircraft performance and the potential air travel demand. The LBSAM allows technical personnel at NASA to assess the market

of possible designs quickly. To the best of our knowledge, no tool like this one exist in the public domain. The NASA Langley Research Center currently uses the LBSAM as part of their supersonic aircraft concept design projects.

1.6 Organization of the Dissertation

The research work done and the model developed for this dissertation are outlined using the following structure.

Chapter 2: Aviation Global Demand Forecast Model Development - Supersonic Aircraft Market: Study 1. First attempt to estimate potential commercial supersonic travel demand. For this study, we analyzed a low boom and non-low boom supersonic aircraft concept with a seating capacity of 20, 40, and 60 seats.

Chapter 3: Aviation Global Demand Forecast Model Development - Supersonic Aircraft Market: Study 2. Improvement of the worldwide commercial supersonic travel demand forecast. We analyzed a 20-seat and 40- seat low-boom supersonic aircraft concept and a 40-seats non-low boom supersonic aircraft concept during this study.

Chapter 4: Aviation Global Demand Forecast Model Development - Supersonic Aircraft Market: Study 3. During the third study, we created the Low Boom Supersonic Aircraft Model (LBSAM). Using the LBSAM, we analyzed over forty-eight scenarios by combining multiple aircraft parameters and performance characteristics.

Chapter 5: Conclusion and Recommendations. This chapter will conclude the work presented in this dissertation and recommendations for future research.

Reference.

Appendix A – Study 3: Additional analysis. This appendix presents parametric analysis performed using the LBSAM model.

Appendix B – Findings. This appendix presents a recompilation of the main findings of each of the three studies presented in this dissertation.

Appendix C – LBSAM Source Code

Chapter 2

2 Aviation Global Demand Forecast Model Development – Supersonic Aircraft Market: Study 1

2.1 NASA Supersonic Transport Design Concept

To develop a market study, we use several proposed NASA SST concepts. The essential characteristics of these aircraft are shown in Table 6: NASA Supersonic Transport Aircraft Concepts. Source: NASA Langley. The table shows a family of aircraft designs ranging in capacity from 18 premium seats (small no-low-boom concept) to a large 60-seat low-boom design with an overall length of 257 feet (seven feet longer than Boeing 747-8I - the longest commercial aircraft operating today). The right-hand side of Figure 6 shows the comparative size of other supersonic aircraft, including the BAC/Aerospatiale Concorde, Boom Overture, and Aerion2 supersonic business jet aircraft.

Table 6: NASA Supersonic Transport Aircraft Concepts. Source: NASA Langley.

Concept	TOGW[3]	Length, ft.	Wing Area, ft.	Thrust, lbf.	Total Fuel, lb.	Mach	L/D[4]	SFC[5]	Design Range, nm
18-Passenger NLB	Small non-low-boom.								
Overland	128,000	136.2	1,695	25,951	71,665	1.15	10.00	0.872	4,809
Overwater						1.6	9.00	.971	5,273
18-Passenger LB	Small Low-Boom. Max supersonic cruise weight over land is ≈ 112,000 lb.								
Overland	117,300	190.7	3,135	29,443	53,999	1.4	8.33	0.980	2,652
Overwater	138,000				74,699	1.6	9.31	0.968	5,216
40-Passenger NLB	Medium size non-low-boom.								
Overland	205,000	194.0	3,680	41,976	110,819	1.15	10.92	0.854	4,963
Overwater						1.6	9.37	0.940	5,243
40-Passenger LB	Medium size low-boom. Max supersonic cruise weight over land is ≈ 178,000 lb.								
Overland	178,200	240.3	3,530	39,529	79,200	1.4	8.46	0.940	2,777
Overwater	230,000				130,996	1.6	8.42	0.938	5,262
60-Passenger NLB	Large non-low-boom.								

[3] TOGW – Takeoff gross weight

[4] L/D – Lift-to-drag ratio.

[5] SFC – fuel efficiency of an engine design with respect to thrust.

Concept	TOGW[3]	Length, ft.	Wing Area, ft.	Thrust, lbf.	Total Fuel, lb.	Mach	L/D[4]	SFC[5]	Design Range, nm
Overland	240,000	225.0	4,905	42,000	120,191	1.15	10.92	0.851	4,152
Overwater						1.6	11.15	0.936	5,209
60-Passenger LB	Large low-boom. Max supersonic cruise weight over land is ≈ 203,000 lb.								
Overland	213,200	257.0	4,315	47,055	89,934	1.4	8.73	0.943	2,663
Overwater	260,000				136,734	1.6	8.29	0.969	5,255

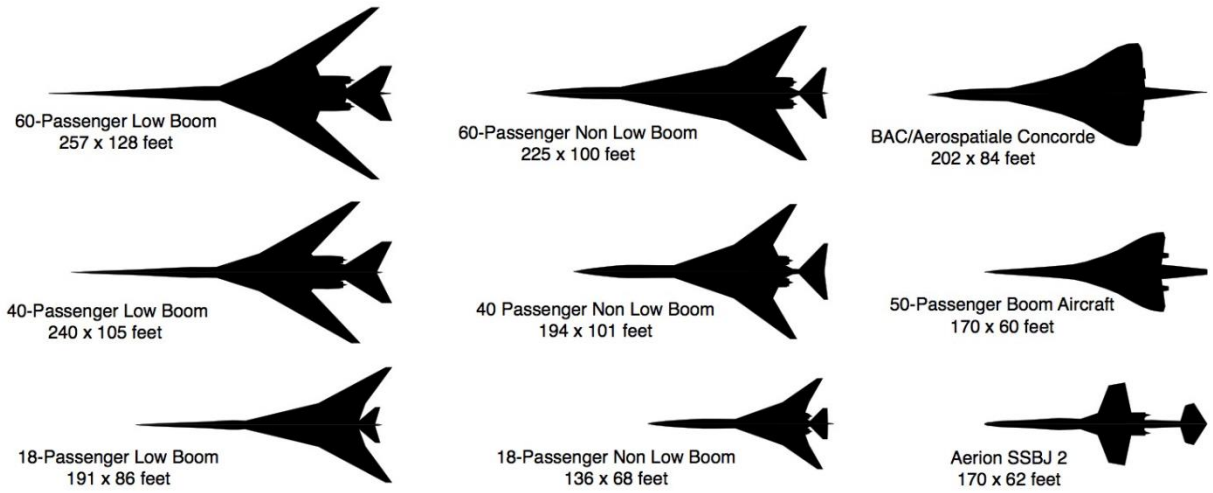


Figure 6: Illustration of Supersonic Aircraft Configurations Considered in the Study. Source of data: NASA LaRC (2018).

Additional information regarding supersonic aircraft airport compatibility and runway length requirements can be found in reference [3].

2.2 Life Cycle Cost Analysis

A series of Life Cycle Cost (LCC) models were developed to quantify each aircraft design cost per passenger mile. Each LCC model was developed in STELLA Author - a Systems Dynamics tool developed by High-Performance Systems (HPS, 2018). The LCC model has an interface created to facilitate making sensitivity runs of various operational factors. The Systems Dynamics LCC model tracks aircraft costs over the life cycle of operations to estimate an hourly operating cost. The model considers the following cost categories:

- Facilities cost (hangar, office space, and landing site)
- Recurring costs (engine, paint, refurbishing, avionics, mid-life inspection, etc.)
- Variable costs (fuel, oil, parts, miscellaneous, maintenance, etc.)
- Fixed costs (hull insurance, liability, maintenance software, property tax)
- Personnel costs (pilot salaries, benefits)
- Training costs (initial, maintenance, recurrent training, etc.)
- Capital and amortization costs (percent resale value, interest rate, purchase cost)
- Airline administrative cost

A detailed description of the life cycle cost analysis that complements the effort presented in this dissertation document is available in *"Aviation Global Demand Forecast Model Development and ISAAC Studies: Supersonic Aircraft Market Study"* (Freire, E. et al., 2018, 2019, 2020) [1] [2] [3].

2.3 Travel Time Analysis

Travel demand using supersonic aircraft is expected to be sensitive to travel time. The argument for faster travel speeds offered by supersonic aircraft is the travel time savings and increased productivity for the traveler. Section 2.4 presents travel demand estimates using the Airline Research Corporation database (ARC) using a value-of-time approach. This Section presents modifications made to a flight planner application developed by the Air Transportation Systems Laboratory for the Federal Aviation Administration (FAA). The flight planner was designed as part of a Global Oceanic Model (GO Model). We used the flight planner tool to estimate the travel time of supersonic aircraft both overland and overwater by using a wind-optimal route. Additional information on the flight planer tool can be found in "Flight Planner for the Global Oceanic Model" [24].

2.3.1 Additional Travel Times

Travelers make decisions on specific modes of transportation, considering published travel times between two airports. These published travel times include flight times, taxi-in, taxi-out, and additional travel times to account for recurrent congestion at airports. The flight planner travel times are supplemented with additional travel times to generate realistic travel times from the point of origin to the destination. The travel times estimated on the flight planner do not account for taxi times on airport terminals, climb, or descent times. Analysis of climb and descent profiles produced by FLOPS shows a typical supersonic aircraft climb profile to 49,000 feet takes 10.4 minutes (see Figure 7) and 125 nautical miles.

Similarly, a descent profile from cruise altitude to 10,000 feet requires 24 minutes and 135 nautical miles. The flight planner described in the previous section assumes the aircraft starts its flight at the origin airport on cruise. Similarly, the flight planner's optimal flight track ends at the destination airport at the same cruise altitude. We added 6 and 23 minutes to the climb and descent phases to correct the flight planner's coplanar travel time estimates to account for the additional climb and descent travel times. Figure 7 shows graphically the corrections made in the analysis to calculate gate-to-gate travel times.

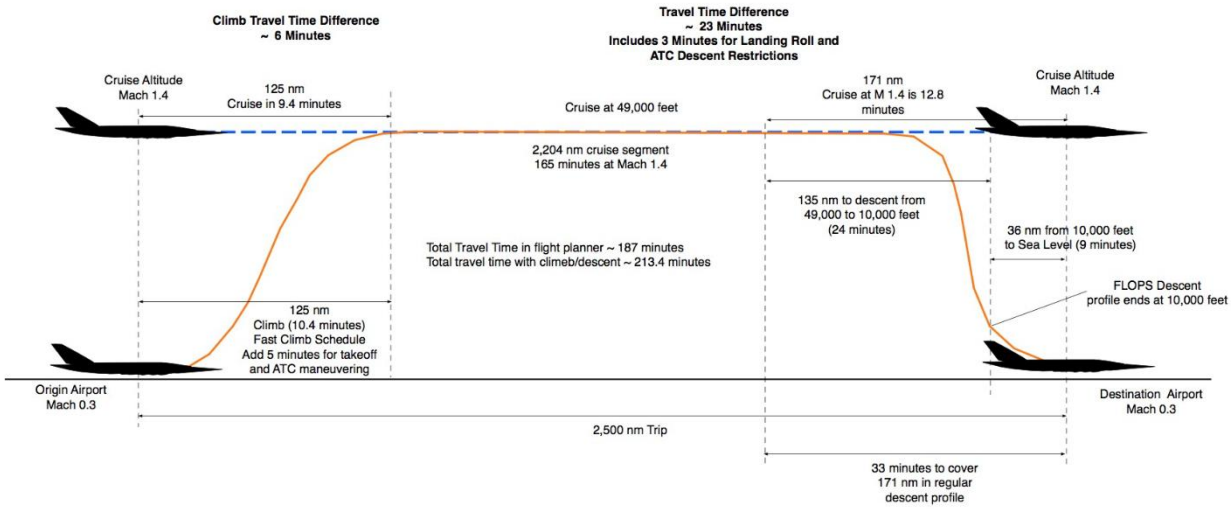


Figure 7: Climb and Descent Travel Times Calculations in Supersonic Aircraft Profiles [1].

To account for taxi times, two linear models are used to predict taxi-in and taxi-out times for supersonic aircraft. These linear models use Federal Aviation Administration Aviation Systems Performance Metrics (ASPM) data for 77 airports in the United States (ASPM, 2018). The details

of the models are explained in “Global Commercial Aircraft Fuel Burn and Emissions Forecast: 2016 to 2040” [25]. The models consider the number of arrivals to estimate taxi times. The number of annual operations at airports is obtained from OAG 2016 data. Figure 8 shows the linear regression model to estimate taxi-in times. Figure 9 shows the linear model to predict taxi-out times. For airports with no annual operations data in OAG, we assign minimum taxi-in and taxi-out times of 4 and 11 minutes, respectively. These are the taxi-in and taxi-out values predicted by the linear models for shallow annual operations.

For flights that exceed the supersonic aircraft's design range, we consider a stop-over time of one hour to account for refueling and cabin servicing. A stop-over also implies corrections from the flight planner travel time values to account for two taxi-out, two climbs, two descent, and two taxi-in phases.

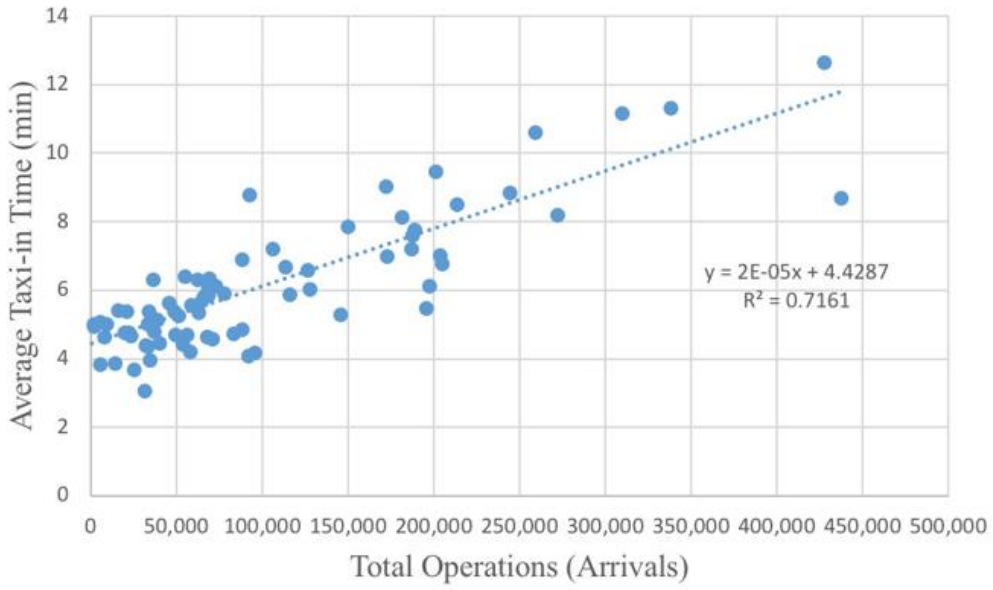


Figure 8: Linear Trend between Taxi-In Times and the Annual Number of Arrivals at ASMP 77 Airport for the Year 2015 [25].

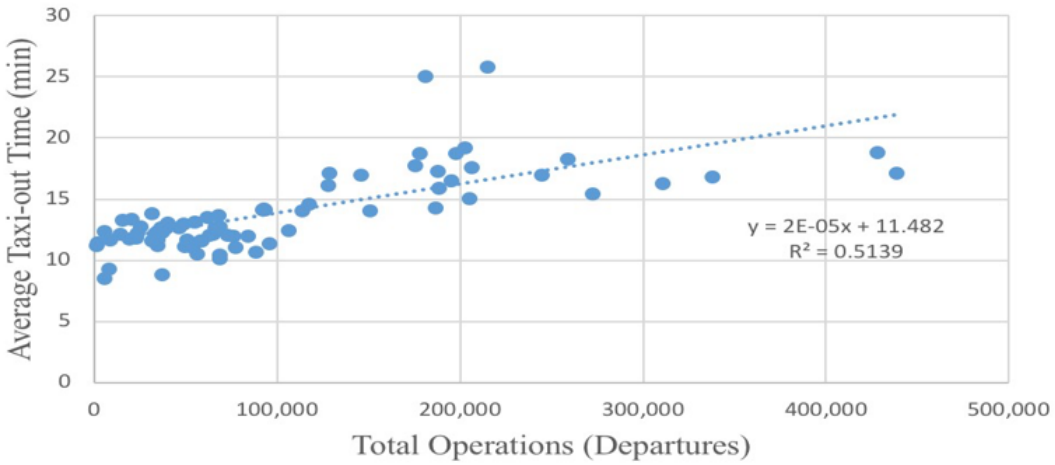


Figure 9: Linear Trend between Taxi-Out Times and the Annual Number of Arrivals at ASPM 77 Airports for the Year 2015 [25].

2.4 Value of Time & Market Fare per Seat-Mile Analysis

One of the main benefits of supersonic transport is the high speed of travel and hence the travel time savings associated with such travel. Quantifying the potential demand for supersonic services requires investigating the traveler's willingness to pay for additional speed (i.e., travel time savings). The Value of Time Analysis (VOT) concept estimates how travel demand responds to changes in travel time savings. This analysis estimates how much additional money a passenger is willing to pay per hour of travel time saved. The analysis employs an extensive fare database collected by the Airline Research Corporation in 2012 (ARC, 2012).

A total of fifty-six one-way Origin-Destination pairs with large numbers of premium seat ticket records were analyzed. ARC 2012 data provides information about passenger tickets and fares paid. ARC contains ticket records representing single flights (i.e., direct flights) and records with multiple leg flights (i.e., one or more stops). The logic behind the analysis can be explained with the following example. According to the data, a direct flight from Hong Kong International Airport to Singapore Changi Airport costs on average $1,640 for a premium seat. The same flight but with a stop in an intermediate airport cost on average $1,428. The direct flights take about 3.9 hours, while the flight with a one-stop takes about 6.8 hours to reach the final destination. In this example, the data indicate that passengers value their time at $73 per hour. (Eq. 1) shows how the Value of Time is calculated.

The value of time was studied for all 56 one-way OD pairs selected. Figure 10 presents over one hundred thousand records from ARC 2012 data for direct and one-stop flights showing fare paid versus travel time. The general trend is that direct flight records are more expensive and faster than one-stop flight records. A summary of the values of time derived from this analysis is presented in

Table 7. The table suggests that, on longer flights, passengers are willing to pay more for travel time savings than on shorter flights. Three segments of travel time are used in the remaining of the analysis.

There are four primary datasets and one ICAO report used in our methodology to estimate potential supersonic demand: a) the Airlines Reporting Corporation (ARC) data, b) the Official Airlines Guide (OAG), c) Gross domestic product (GDP) data, d) Official Airline Guide (OAG) 2016 data, and e) the ICAO Long Range Traffic Forecast, Passenger and Cargo, July 2016 [26].

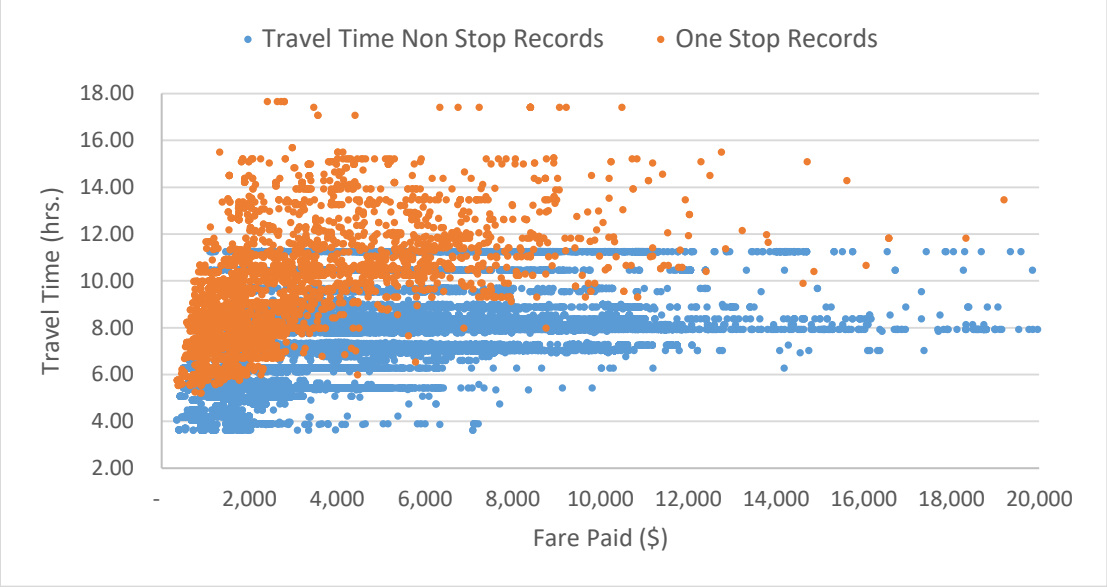


Figure 10: Direct Flights and One-Stop Flight Records from ARC 2012 Data. Fare Paid vs. Travel Time.

$$Value\ of\ Time = \frac{Nonstop\ Fare - OneStop\ Fare}{OneStop\ TravelTime - Nonstop\ TravelTime} \quad \text{(Eq. 1)}$$

Table 7: Value of Time Results.

Value of Time ($/hr.)	Distance Category (statute mile)
$73/hr.	1,500 – 2,500
$117/hr.	2,501 – 5,000
$139/hr.	Greater than 5,000

Figure 11 shows graphically the analysis performed with ARC 2012 data. The purpose of this analysis is to obtain the percent of passengers willing to pay fares at or above a given cost per passenger mile threshold. The analysis considers fares paid from ARC 2012, travel time savings by aircraft type, and value of time depending on travel distance to generate cumulative fare plots.

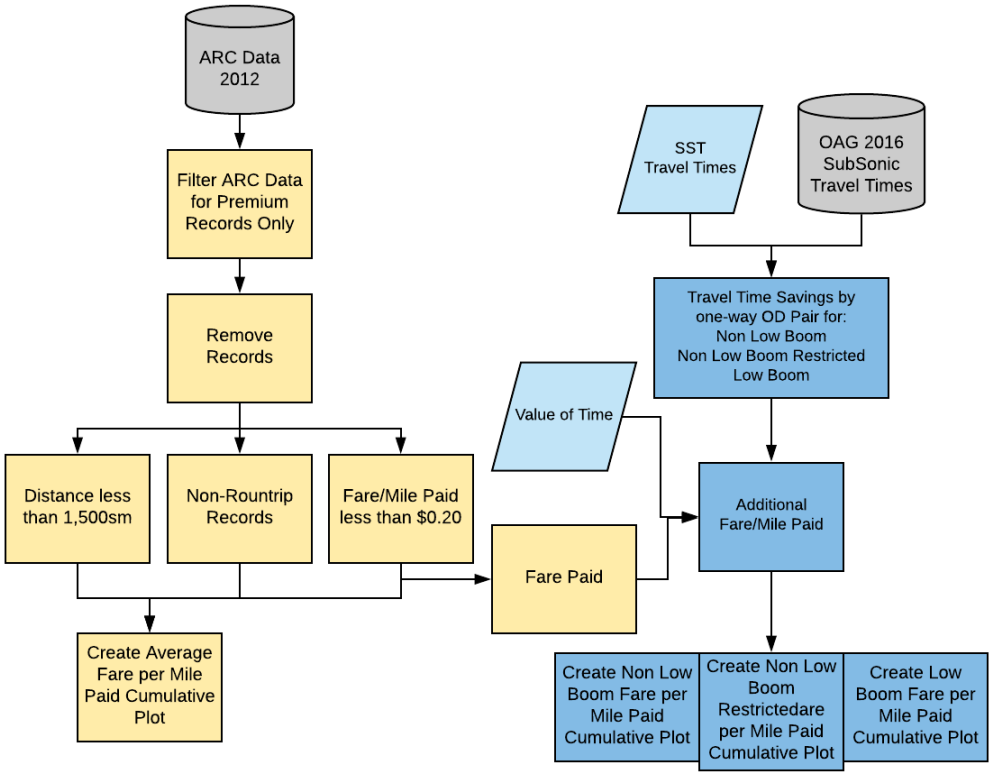


Figure 11: Flowchart of Value of Time and Market Fare per Seat-Mile Analysis.

The first step in the analysis was to read and parse the raw data and create a database. Next, we extracted premium records from more than 300 million records. Premium records include first-class and business class tickets. The analysis assumed that passengers paying for economy and premium economy tickets would not be candidates for the SST service. This assumption was validated with the life cycle cost model. More than 99% of the economy class records involve tickets with a fare per mile below $0.50, a price point below the best solution obtained with the life cycle cost model described in this report. Further analysis of premium records identified records that met three criteria: a) distance between origin and destination was more than 1,500 statute miles (sm.); b) records that involved round trips; and c) records with fare per mile value more than $0.20 per seat-mile. The application of these filters produced 2.2 million premium records from the original 300 million records in the ARC database.

The first criteria imply that in short trips (< 1500 statute miles), supersonic aircraft flights' travel time savings would make the shift from subsonic flights unfeasible. Trip distances less than 1,500 statute miles involve mainly overland flights. Using Mach 0.95 per current regulations for overland flights, the travel savings on a short trip are estimated to be 30 minutes or less. The second criteria (i.e., round trips) was applied to avoid errors when calculating the fare per mile paid by passengers. The third criterion used in this analysis was to remove all those records with a fare per mile paid less than $0.20. Records identified as premium in the database but having a fare per passenger mile less than $0.20 are considered upgrades and hence ignored.

The final step in the analysis is to create a generic cumulative fare per passenger mile plot to estimate the percent of passengers willing to pay fares at or above a given cost per passenger mile threshold level, as shown in Figure 12. For example, 44% of the ARC data's premium records involved fare per passenger mile below $0.50. Comparing passengers' average premium seat fares with life-cycle cost values provides an initial estimate of the potential premium supersonic aircraft services market.

For 18, 40, and 60-seat non-low-boom and non-low-boom restricted aircraft, 2%, 23%, and 40% of travelers using premium seats paid a fare at or above the $1.50, $0.75, and $0.61 passenger-mile value. For 18, 40, and 60-seat Low-boom aircraft, 1.7%, 18%, and 34% of travelers using

premium seats paid a fare at or above the $1.57, $0.81, and $0.65 passenger-mile value, respectively.

These estimates do not account for the passenger's willingness to pay a higher fare for faster services. This generic cumulative fare per passenger mile plot is used later in the analysis for one-way OD pairs containing ARC and OAG information. For example, a one-way OD pair that was not part of the 2012 (ARC 2012 data) network but is it in the OAG 2016 network will not have data to generate a cumulative plot.

We developed an algorithm to consider the passenger's willingness to pay higher fares for faster service using the ARC data. The model calculates the travel time savings by aircraft type (Non-Low-Boom, Non-Low-Boom restricted, and Low-Boom) and for each one-way OD pairs that contain enough records to generate a reasonable cumulative fare per mile plot. These cumulative plots are developed with and without consideration of the value of time. The percent of passengers willing to pay obtained from the cumulative plots is a function of cost per passenger-mile and travel time savings offered by the supersonic aircraft.

For example, using the JFK to LAX one-way OD pair and the operational economics of a 40-seat Non-low-boom aircraft ($0.75 per passenger mile), 41% of travelers using premium seats paid a fare at or above the $0.75 passenger-mile value. For the same 40-seat Non-low-boom aircraft restricted to fly overland at or below Mach 0.95, 38% of travelers using premium seats paid a fare at or above the $0.75 passenger-mile value. The analysis generates cumulative plots for 1,881 one-way OD pairs for the three aircraft types (Non-Low Boo, Non-Low-Boom restricted, and Low-Boom).

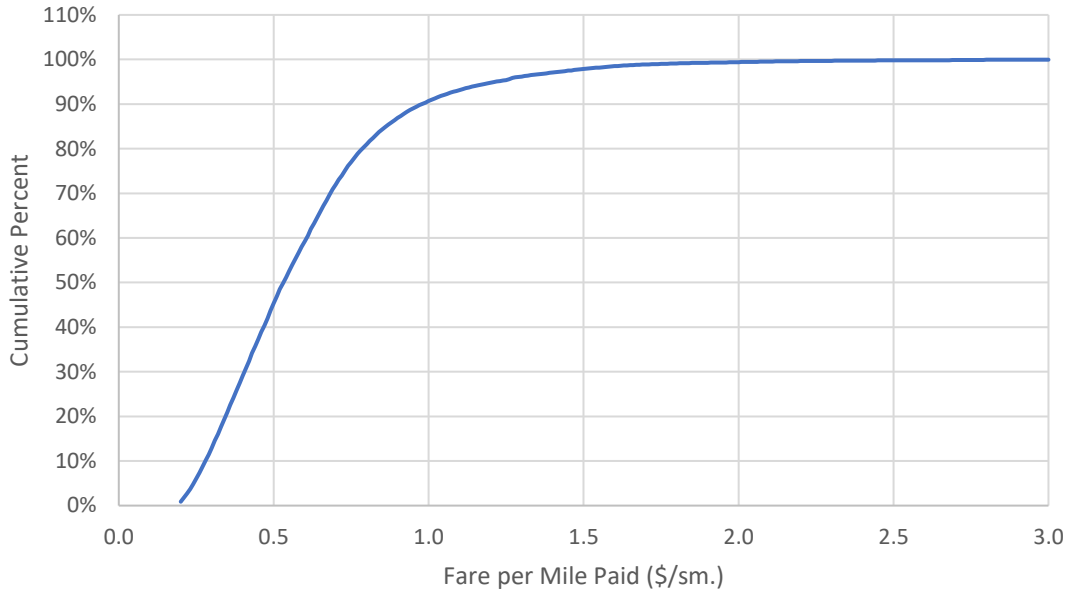


Figure 12: Cumulative Fare per Mile Paid Plot Using ARC 2012 Data.

2.5 Worldwide Commercial Air Travel Demand Forecast Model

OAG 2016 data has been used as the baseline year for this analysis. The methodology from the ICAO Long Range Traffic Forecast, Passenger and Cargo, July 2016 has been adopted to generate a forecast model that predicts future passenger demand for years 2030 to 2040. The ICAO document contains equations developed to forecast air traffic growth within and between eleven world regions. The eleven regions include North America, Central America and Caribbean, South America, Europe, North Africa, Sub Saharan Africa, Middle East, Central, and Southwest Asia, North Asia, and Pacific South East Asia.

We developed a regional mapping between the OAG and ICAO regions since the model uses OAG 2016 data as the baseline year. OAG data has seventeen world regions. In OAG, Africa, Asia, and Latin America are divided into four regions each. Europe is divided into two regions, and North America, the Middle East, and Southwest Pacific are a single region. The Compound Annual Rate of Growth (CARG) from the ICAO report was adopted in the analysis to map the region-to-region pair. Two hundred and one region-to-region were studied in the analysis.

The ICAO forecast model consists of several steps that are explained in the following steps.

- Identify the GDP values for the airport of origin and airport of destination for each one-way OD pair. If no GDP data is available for any given airport, then a value of zero is assigned.
- Calculate the Compound Annual Growth for the airport of origin (Dep_Airport) and the destination airport (Arr_Airport) using the GPD values assigned to each airport in the previous step.

$$Dep\ Airport\ CAGR = \frac{Dep\ GDP\ End\ Year}{Dep\ GDP\ Start\ Year}^{1/Years} - 1 \quad \textbf{(Eq. 2)}$$

$$Arr\ Airport\ CAGR = \frac{Arr\ GDP\ End\ Year}{Arr\ GDP\ Start\ Year}^{1/Years} - 1 \quad \textbf{(Eq. 3)}$$

- If any of the airport's GDP values is zero, then the average worldwide CARG value of 2.89% is used.
- Calculate the CAGR value for each one-way OD pair

$$Avg\ OD\ CAGR = \frac{Dep\ Airport\ CAGR\ +\ Arr\ Airport\ CAGR}{2} \quad \text{(Eq. 4)}$$

- Calculate the average region CAGR. This term is the average CAGR of all the airport CAGR that departs from the same region.
- Calculate forecast seats for each one-way OD pair from 2016 to 2040.

$$Total\ Seats = OAG_{2016}Seats * ICAO\ Region\ to\ Region\ CAGR^{Years} * \frac{Avg\ OD\ CAGR}{Avg\ region\ CAGR} \quad \textbf{(Eq. 5)}$$

Adopting the ICAO forecast methodology, a forecast of future seats was developed from 2030 to 2040. The results of the forecast are presented in. The number of seats forecast for commercial flights starts at 5 billion in 2017 and increases to 11.3 billion by the year 2040. The results show a compound annual growth rate of 3.48% worldwide.

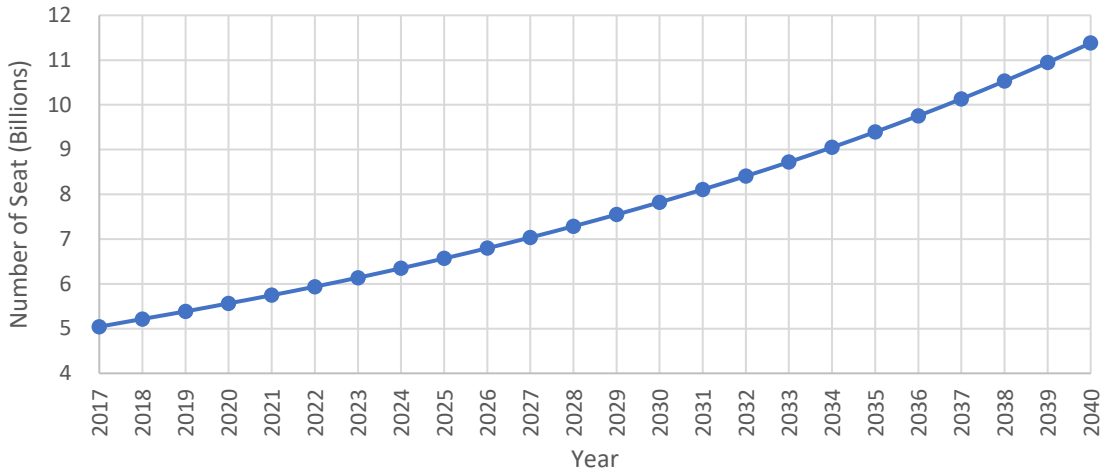


Figure 13: ICAO Worldwide Forecast Model, Number of Seats Over Time.

Our model predicts the number of seats that are expected to be offered in the future by airlines. Companies such as Boeing and Airbus also publish their market forecast. Instead of using seats, their metric is Revenue Passenger Kilometer (RPK). Such forecast can be revised by reading Boeing Commercial Market Outlook 2019-2038 [27] or Airbus Global Market Forecast 2019-2038 [28]. The model generates future projections of premium seats worldwide by using the ICAO forecast seats. As shown in Figure 13, the forecast estimates the number of seats that will be offered worldwide from the year 2017 to the year 2040. In consultation with NASA, we assume that SST aircraft will be introduced to the worldwide network in the year 2030. Hence, the ICAO forecast data spans from the year 2030 to 2040. Figure 14 describes the analysis performed using OAG 2016 data and the ICAO 2030-2040 forecast to predict potential demand for commercial supersonic services.

2.6 Worldwide Supersonic Air Travel Demand Forecast Model

Using 2016 OAG data as the base year, we extracted the total number of premium seats for each one-way OD pair. We then estimated the ratio of premium seats to total seats (i.e., indicates the percent of premium seats from the total seat count). We combined the ICAO forecast with the OAG data to match all the one-way OD pairs from both datasets. The ICAO forecast predicts the total number of seats in the future. The ICAO forecast does not predict the number of future premium seats. For this reason, the premium seat to total seat ratio is used to estimate the number of premium seats to be offered in each of the one-way OD pairs between 2030 and 2040.

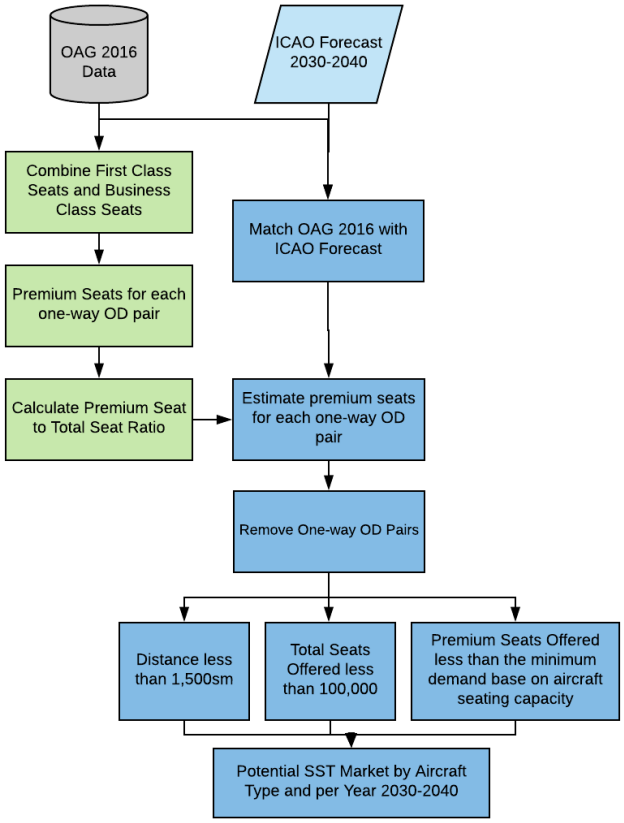


Figure 14: Flowchart of OAG 2016 and ICAO Forecast Analysis to Estimate Potential SST Commercial Market.

To find the potential market for SST flights over time, we employed three criteria to further narrow down the one-way OD pairs. The first criteria consider one-way OD pairs with a route distance greater than 1,500 sm. The second criteria identify one-way OD pairs with total seats offered greater than 100,000 annual seats. The rule was established to assume that one-way OD pairs with a small demand would not support reliable SST services (i.e., five times per week). The third criterion was to keep all one-way OD pairs with more than the minimum demand established based on aircraft seating capacity and annual operations. For the number of annual operations, the assumption is one flight per day and three hundred days a year as a minimum. The analysis considers three aircraft sizes with seating configurations for 18, 40, and 60 seats for the aircraft seating capacity. The minimum demand for premium seats for an 18-seat transport requires 5,400 premium seats per year—the 40 and 60-seat aircraft configurations a minimum annual demand of 12,000 and 18,000 premium seats, respectively.

The analysis also considers design tradeoffs using low-boom and non-low-boom aircraft and three cruise speed profiles. The three aircraft type combinations are labeled Non-Low-boom aircraft, Non-Low-boom restricted aircraft, and Low-Boom aircraft. The Non-Low-Boom aircraft flies at Mach 1.15 overland at Mach 1.6 overwater. The Non-Low-Boom restricted aircraft flies at Mach 0.95 overland and Mach 1.6 overwater. The Low-Boom aircraft flies at Mach 1.4 overland and Mach 1.6 overwater. Considering three seating capacity vehicles and three aircraft types, the model generates nine potential markets for SST commercial transports.

2.6.1 Number of Supersonic Aircraft Worldwide

Figure 15 presents a flowchart to estimate the number of supersonic aircraft worldwide using the ARC, OAG, and ICAO forecast analyses explained in previous sections of this report. The three-step procedure calculates the number of non-low-boom aircraft, non-low-boom restricted aircraft, and low-boom aircraft. First, premium seat demand is estimated for each one-way OD pair. Second, a filter is applied to select only one-way OD pairs with the potential for supersonic flights. Third, the number of aircraft needed to satisfy the forecast demand is calculated by considering; seat demand, travel times, and aircraft use limitations. The three-step process is explained in detail in the following paragraphs.

We use the OAG 2016 and ICAO forecast analysis to estimate each one-way OD pair (Figure 14) between 2030 and 2040. The number of premium seats for each one-way OD pair is multiplied by the corresponding percent of passengers willing to pay. The percent of passengers willing to pay is obtained from the cumulative plots generated considering the value of time, cost per passenger-mile, and travel time savings (Figure 11). The resulting number of premium seats is finally multiplied by a market share factor representing the percent of passengers in premium seats willing to pay equal or more than the projected cost per mile of supersonic concept adjusted for the value of time. We consider two possible scenarios with market share factors at 50% or 100% in the analysis presented. These market share scenarios are labeled as Scenario 1 (50% of passengers in

premium seats paying equal or more than the projected cost per mile of supersonic concept adjusted for the value of time) and Scenario 2 (100% of passengers in premium seats paying equal or more than the projected cost per mile of supersonic concept adjusted for the value of time). These scenarios represent a high level of uncertainty in demand for supersonic aircraft services for two reasons: a) we believe airlines will not be willing to eliminate premium seats from subsonic aircraft as this will affect the prices paid by passengers by economy class passengers; and b) it is expected that some passengers will not trade the premium seat comfort and amenities offered in subsonic aircraft for smaller business class type seats offered in some supersonic services. This last point may apply for supersonic flights above 4-5 hours.

The analysis considers a minimum SST premium demand filter to ensure that the one-way OD pairs at the end of the analysis can support at least one flight per day and three hundred days a year. The minimum SST premium seat demand thresholds for aircraft seating capacities of 18, 40, and 60 seats are 5,400, 12,000, and 18,000 seats, respectively. One-way OD pairs that do not meet such threshold are removed from the analysis.

The number of aircraft and flight hours needed to satisfy the premium seat demand is calculated considering travel times between one-way OD pairs and the number of hours each supersonic vehicle can be used. In this study, we assume aircraft are used 3,500 hours per year. The number of flight hours is divided by the allowed annual flight hours to calculate the number of aircraft needed for each one-way OD pair every year between 2030 and 2040. Once the number of aircraft required by the one-way OD pair is known, the values are added to estimate the total number of aircraft needed in the network. The last step is to divide the number of aircraft required by a network inefficiency factor (80% in this analysis). The reason to include a network inefficiency

factor is that airlines may have to reposition supersonic aircraft if the demands for one-way OD are small compared to the total supply of flight hours offered by one aircraft.

The number of aircraft considers a network load factor of 75%. This process is done for three aircraft types (non-low-boom, non-low-boom restricted, and low-boom), three seating capacities (18, 40, and 60 seats), and each year from 2030 to 2040. The equations below describe the computational procedure to estimate the number of aircraft worldwide.

$$SST\ Seats = [ICAO\ Forecast\ Seats] * [\%\ of\ Premium\ Seats] * [\%\ of\ Willingness\ to\ Pay] * [\%\ of\ Market\ Shift] \quad \textbf{(Eq. 6)}$$

Select one-way OD pair if.

$$SST\ Seats \geq [Seating\ Capacity] * [1\ Flight\ per\ Day] * [300\ days\ per\ Year] \quad \textbf{(Eq. 7)}$$

$$SST\ Aircraft = \frac{\frac{SST\ Seats}{Acft\ Seating\ Capacity} * Travel\ Time}{Allowed\ Annual\ Flights\ Hours} \over Network\ Inefficiency\ Factor \quad \textbf{(Eq. 8)}$$

$$SST\ Passenger = SST\ Seats * Load\ Factor \quad \textbf{(Eq. 9)}$$

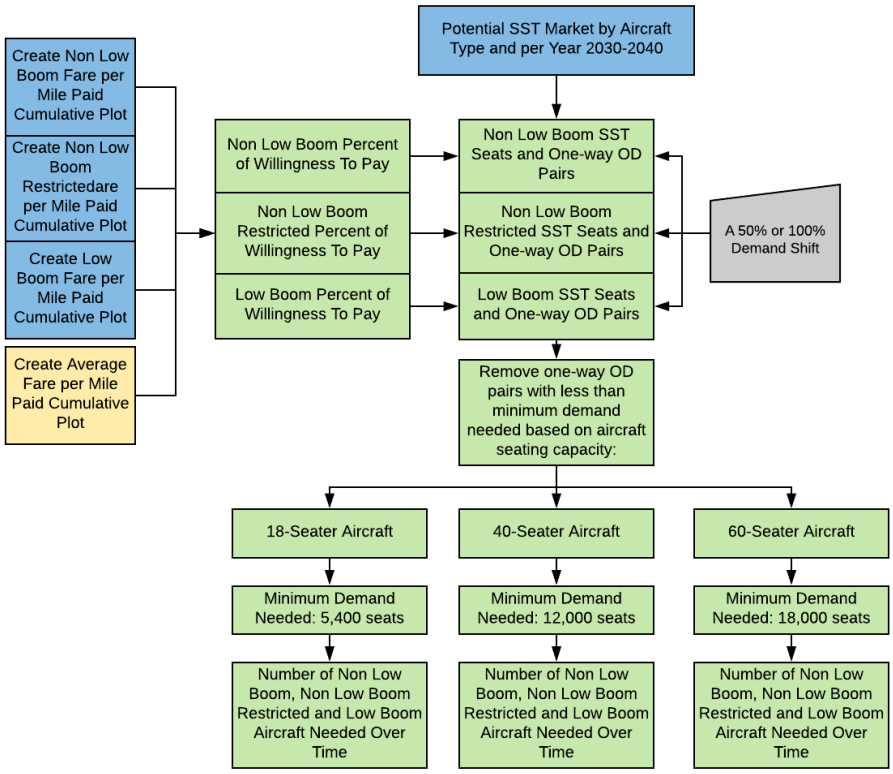


Figure 15: Flowchart to Calculate the Number of Aircraft Needed Worldwide Over Time.

2.7 Results

The previous sections' model was applied to three aircraft size configurations using the life cycle cost model results [2]. The results presented in this section include the following variations:

- 18-Seat Aircraft
  - Non-Low-Boom: $1.50/mi.
  - Non-Low-Boom restricted: $1.50/mi.
  - Low-Boom: $1.57/mi.
- 40-Seat Aircraft
  - Non-Low-Boom: $0.75/mi.
  - Non-Low-Boom restricted: $0.75/mi.
  - Low-Boom: $0.81/mi.
- 60-Seat Aircraft
  - Non-Low-Boom: $0.61/mi.
  - Non-Low-Boom restricted: $0.61/mi.
  - Low-Boom: $0.65/mi.

2.7.1 18-Seat Supersonic Transport Results with Moderate Market Share

The results presented in this section assume that 50% percent of passengers in premium seats pay equal to or more than the projected cost per mile of the 18-seat supersonic concept adjusted for the value of time.

Figure 16 presents the number of potential one-way OD pairs for non-low-boom, non-low-boom restricted, and low-boom aircraft over time. In 2030, we estimate 87, 84, and 69 potential one-way OD pairs for the non-low-boom, non-low-boom restricted, and low-boom aircraft. With an average of 3.4%, 3.6%, and 4.0% annual increase in the number of potential one-way OD pairs, by the year 2040, there could be 121, 119, and 102 potential one-way OD pairs for the non-low-boom, non-low-boom restricted, and low-boom aircraft, respectively.

Figure 17 presents the potential passenger demand for a non-low-boom, non-low-boom restricted, and low-boom aircraft over time. In 2030, we estimate 0.92, 0.90, and 0.74 million passengers for the non-low-boom, non-low-boom restricted, and low-boom market, respectively. With an average of 5.85%, 5.92%, and 6.22% annual increase in the number of passengers, by the year 2040, there could be 1.63, 1.61, and 1.35 million passengers using non-low-boom, non-low-boom restricted, and low-boom aircraft, respectively.

Figure 18 presents the number of non-low-boom, non-low-boom restricted, and low-boom aircraft needed to satisfy the demand worldwide over time. In 2030, we estimate the need for 86, 86, and 66 non-low-boom, non-low-boom restricted, and low-boom aircraft, respectively. With an average of 5.93%, 6.14%, and 6.08% annual increase in the number of passengers, by the year 2040, there would be a need for 153, 156, and 119 non-low-boom, non-low-boom restricted, and low-boom aircraft needed over time, respectively.

2.7.2 40-Seat Supersonic Transport Results with Moderate Market Share

The results presented in this section assume that 50% percent of passengers in premium seats pay equal to or more than the projected cost per mile of the 40-seat supersonic concept adjusted for the value of time.

Figure 19 presents the number of potential one-way OD pairs for the non-low-boom, non-low-boom restricted, and low-boom aircraft over time. In 2030, we estimate 393, 388, and 255 potential one-way OD pairs for the non-low-boom, non-low-boom restricted, and low-boom aircraft, respectively. With an average of 3.79%, 3.76%, and 4.71% annual increase in the number of potential one-way OD pairs, by the year 2040, there could be 570, 561, and 404 potential one-way OD pairs for the non-low-boom, non-low-boom restricted, and low-boom aircraft, respectively.

Figure 20 presents the potential passenger demand for a non-low-boom, non-low-boom restricted, and low-boom aircraft over time. In 2030, we estimate 8.6, 8.4, and 5.7 million passengers for the non-low-boom, non-low-boom restricted, and low-boom market, respectively. With an average of 5.54%, 5.54%, and 6.0% annual increase in the number of passengers, by the year 2040, there could be 14.8, 14.5, and 10.1 million passengers for the non-low-boom, non-low-boom restricted, and low-boom aircraft, respectively.

Figure 21 presents the number of non-low-boom, non-low-boom restricted, and low-boom aircraft needed to satisfy the demand worldwide over time. In 2030, we estimate 86, 86, and 66 non-low-boom, non-low-boom restricted, and low-boom aircraft required over time, respectively. With an average of 5.49%, 5.49%, and 5.95% annual increase in the number of passengers, by the year 2040, there could be 807, 817, and 501 non-low-boom, non-low-boom restricted, and low-boom aircraft needed over time, respectively.

2.7.3 60-Seat Supersonic Transport Results with Moderate Market Share

The results presented in this section assume that 50% percent of passengers in premium seats pay equal to or more than the projected cost per mile of the 60-seat supersonic concept adjusted for the value of time.

Figure 22 presents the number of potential one-way OD pairs for the non-low-boom, non-low-boom restricted, and low-boom aircraft over time. In 2030, we estimate 410, 401, and 356 potential one-way OD pairs for the non-low-boom, non-low-boom restricted, and low-boom market, respectively. With an average of 4.28%, 4.29%, and 4.34% annual increase in the number of potential one-way OD pairs by 2040, there could be 623, 610, and 544 potential one-way OD pairs for the non-low-boom, non-low-boom restricted, and low-boom market, respectively.

Figure 23 presents the potential passenger demand for a non-low-boom, non-low-boom restricted, and low-boom aircraft over time. In 2030, we estimate 12.2, 11.9, and 10.7 million passengers for the non-low-boom, non-low-boom restricted, and low-boom market, respectively. With an average of 5.87%, 5.90%, and 5.93% annual increase in the number of passengers, by the year 2040, there could be 21.5, 21.1, and 9.1 million passengers for the non-low-boom, non-low-boom restricted, and low-boom market, respectively.

Figure 24 presents the number of non-low-boom, non-low-boom restricted, and low-boom aircraft needed to satisfy the demand worldwide over time. In 2030, we estimate 447, 483, and 391 non-low-boom, non-low-boom restricted, and low-boom aircraft required over time, respectively. With an average of 5.70%, 5.73%, and 5.71% annual increase in the number of passengers, by the year 2040, there could be 830, 843, and 861 non-low-boom, non-low-boom restricted, and low-boom aircraft needed over time, respectively.

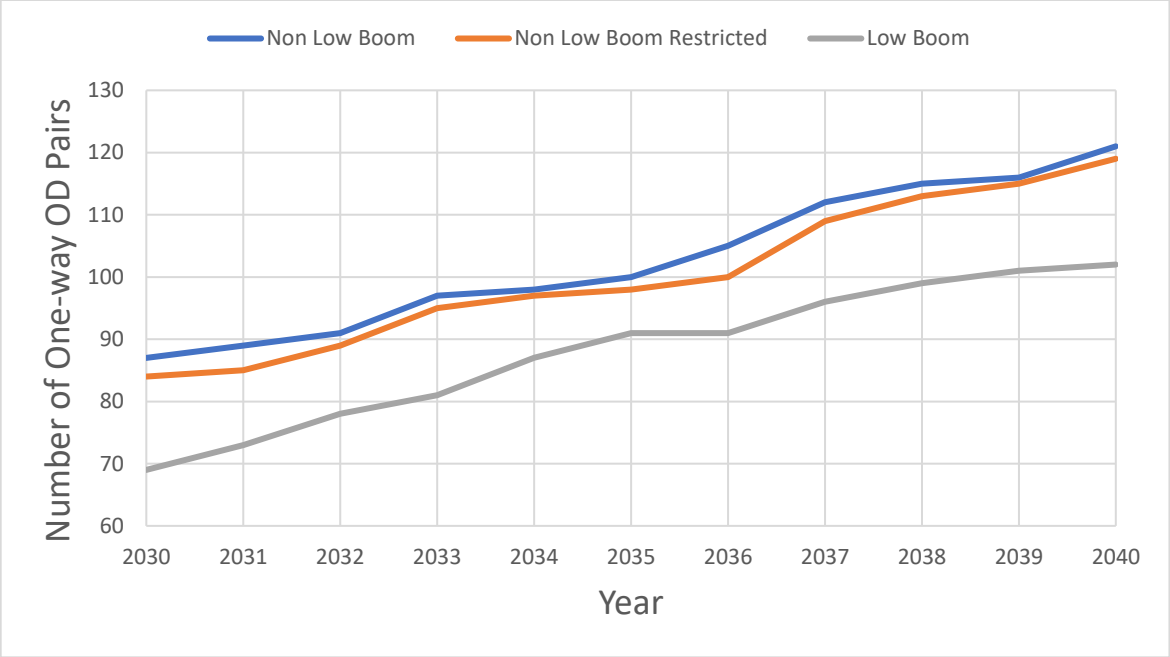


Figure 16: Number of Potential One-way OD Pairs Over Time. 50% of Passengers in Premium Seats Pay Equal or More than the Projected Cost per Mile of the 18-seat Supersonic Aircraft Adjusted for the Value of Time.

Figure 17: Estimated Demand Over Time. 50% of Passengers in Premium Seats Pay Equal or More than the Projected Cost per Mile of the 18-seat Supersonic Aircraft Adjusted for the Value of Time.

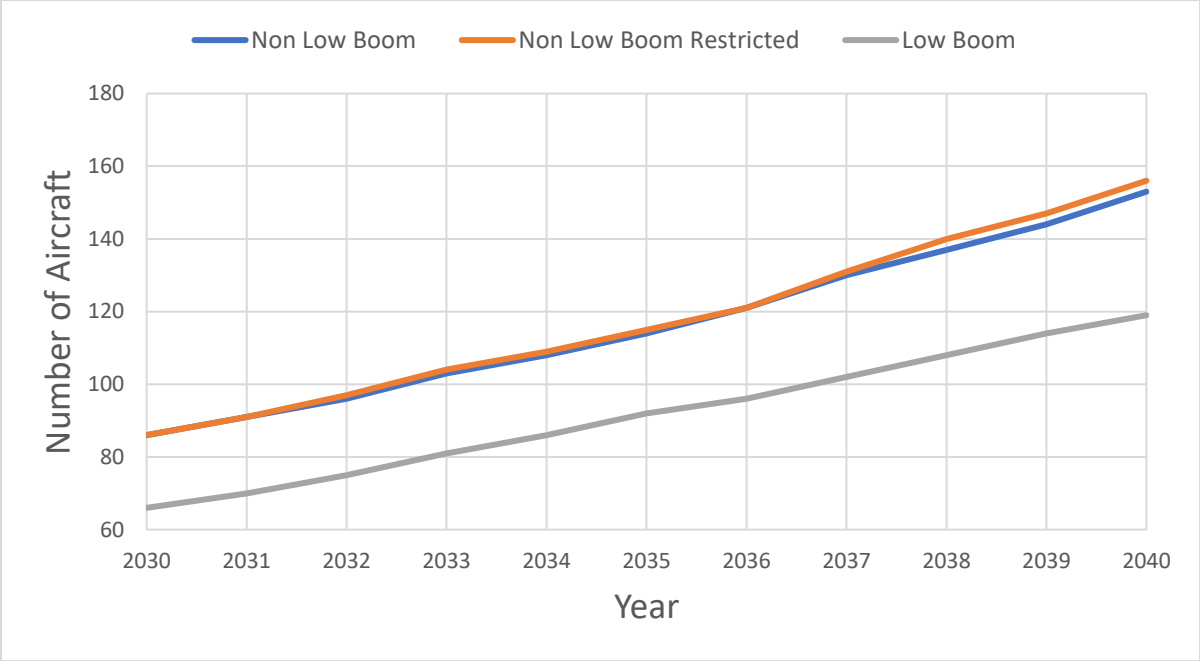


Figure 18: Number of Aircraft Needed Over Time. 50% of Passengers in Premium Seats Pay Equal or More than the Projected Cost per Mile of the 18-seat Supersonic Aircraft Adjusted for the Value of Time.

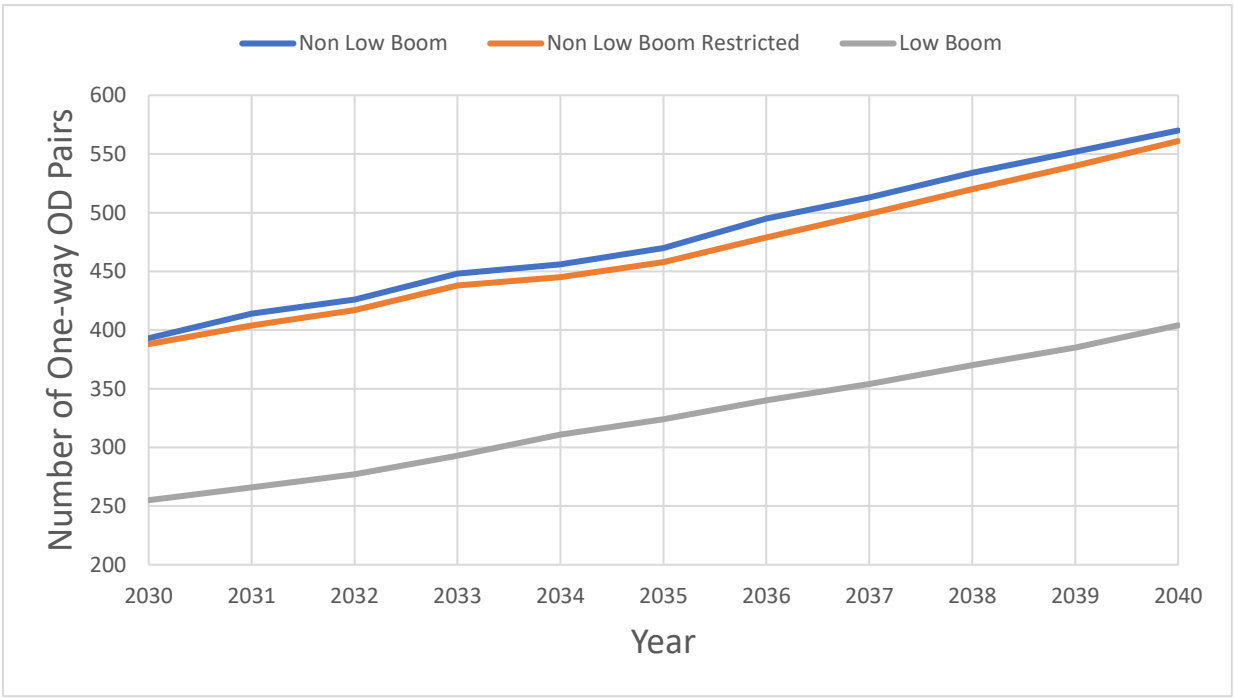


Figure 19: Number of Potential One-way OD Pairs Over Time. 50% of Passengers in Premium Seats Pay Equal or More than the Projected Cost per Mile of the 40-seat Supersonic Aircraft Adjusted for the Value of Time.

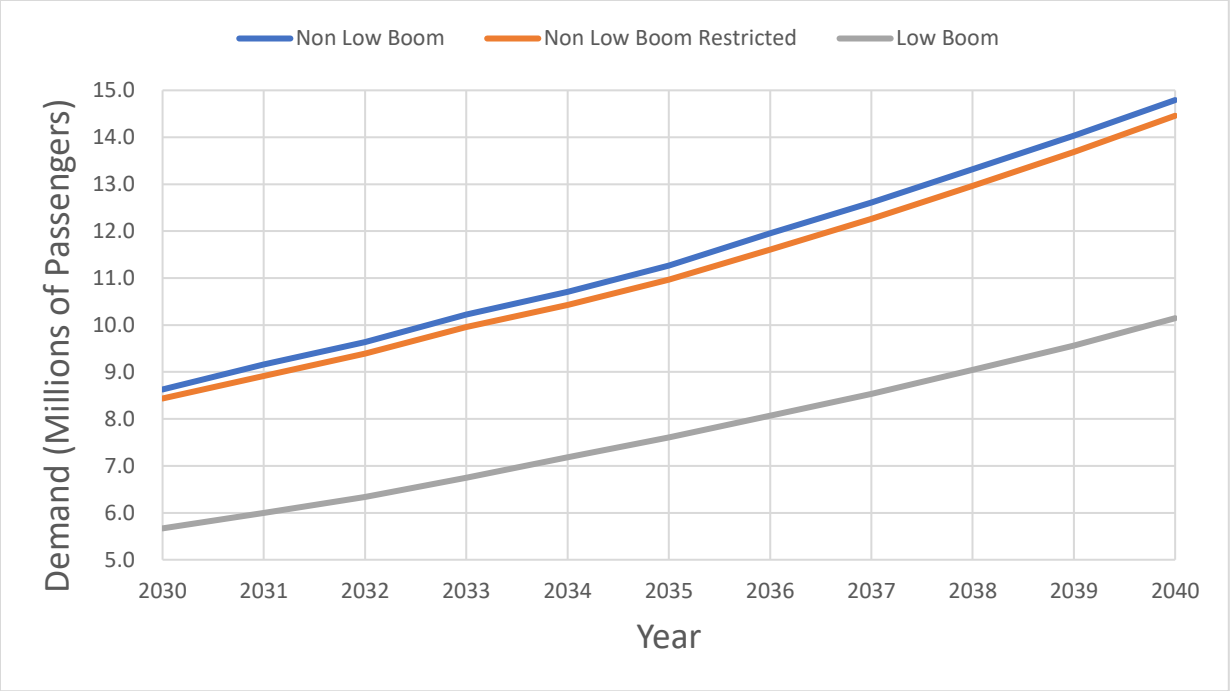


Figure 20: Estimated Demand Over Time. 40-Seat Aircraft. 50% of Passengers in Premium Seats Pay Equal or More than the Projected Cost per Mile of the 40-seat Supersonic Aircraft Adjusted for the Value of Time.

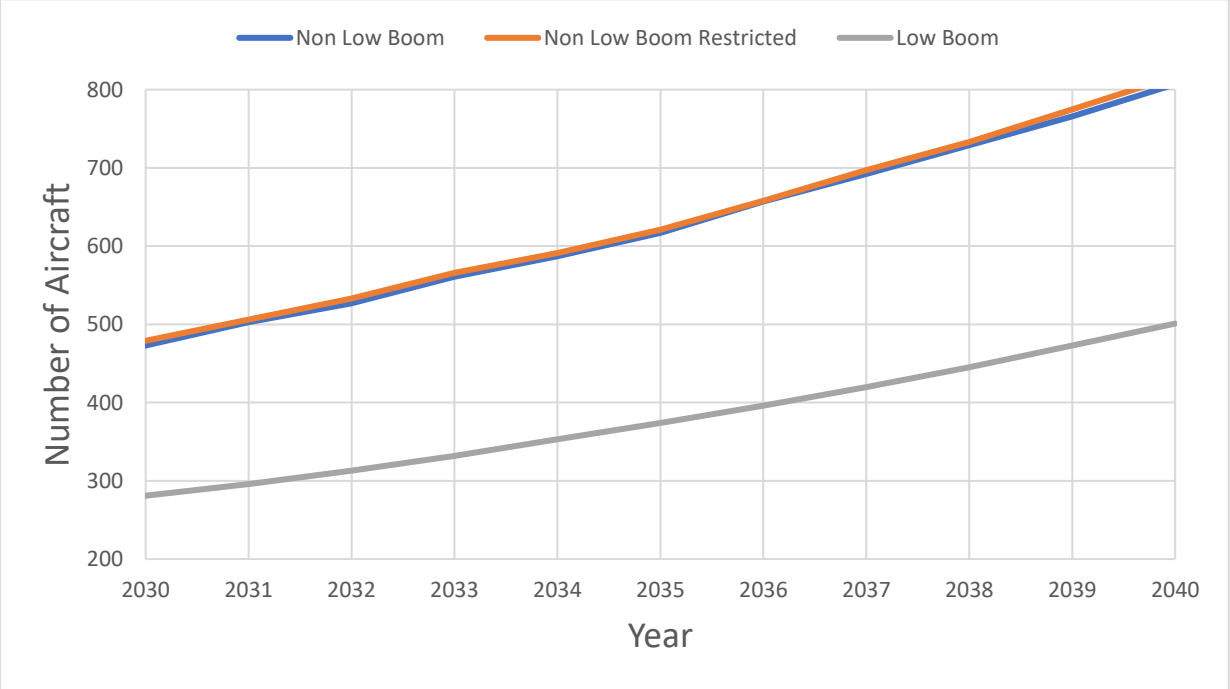


Figure 21: Number of Aircraft Needed Over Time. 50% of Passengers in Premium Seats Pay Equal or More than the Projected Cost per Mile of the 40-seat Supersonic Aircraft Adjusted for the Value of Time.

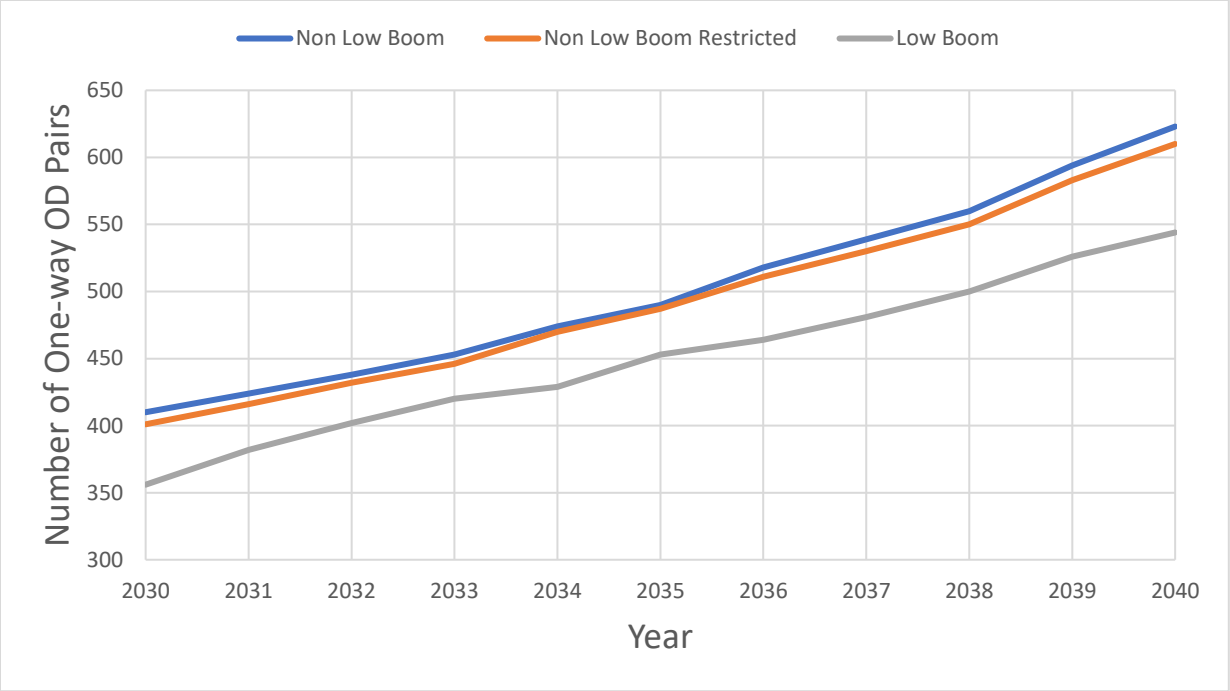


Figure 22: Number of Potential One-way OD Pairs Over Time. 50% of Passengers in Premium Seats Pay Equal or More than the Projected Cost per Mile of the 60-seat Supersonic Aircraft Adjusted for the Value of Time.

Figure 23: Estimated Demand Over Time. 60-Seat Aircraft. 50% of Passengers in Premium Seats Pay Equal or More than the Projected Cost per Mile of the 60-seat Supersonic Aircraft Adjusted for the Value of Time.

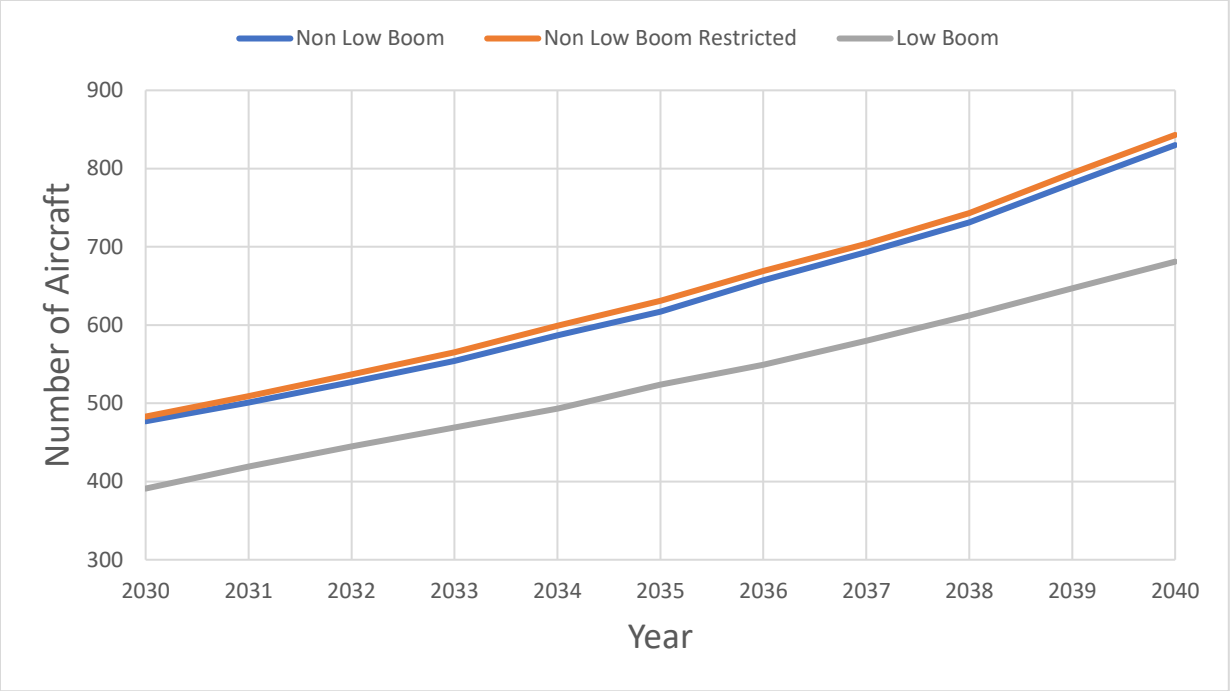


Figure 24: Number of Aircraft Needed Over Time. 50% of Passengers in Premium Seats Pay Equal or More than the Projected Cost per Mile of the 60-seat Supersonic Aircraft Adjusted for the Value of Time.

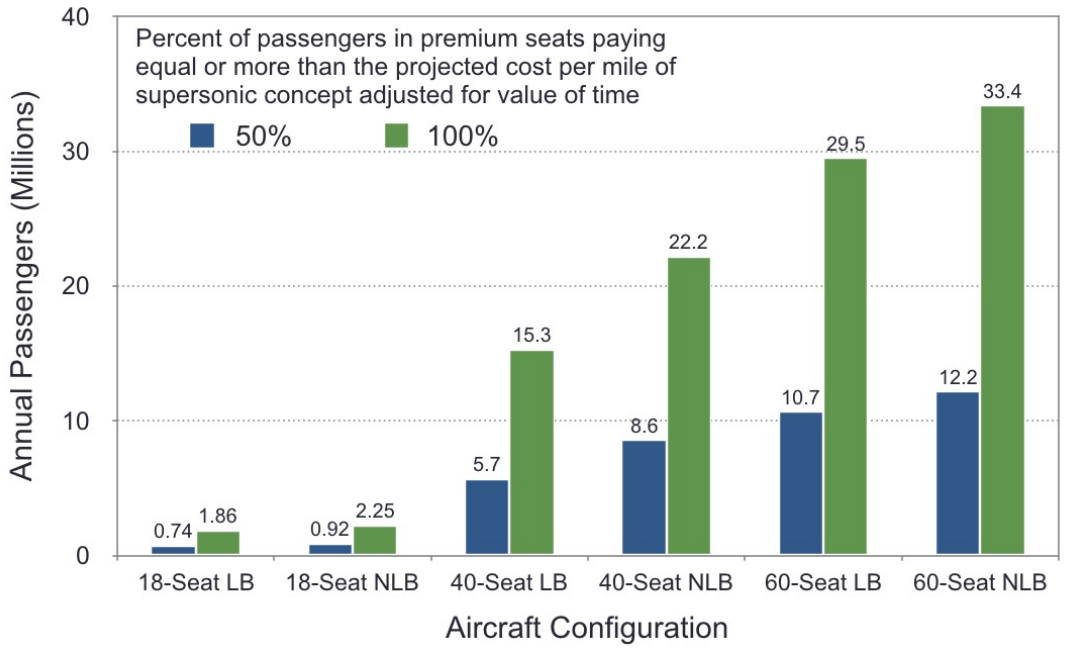


Figure 25: Potential Worldwide Supersonic Passenger Demand in the Year 2030 for Six Aircraft Configurations Studied.

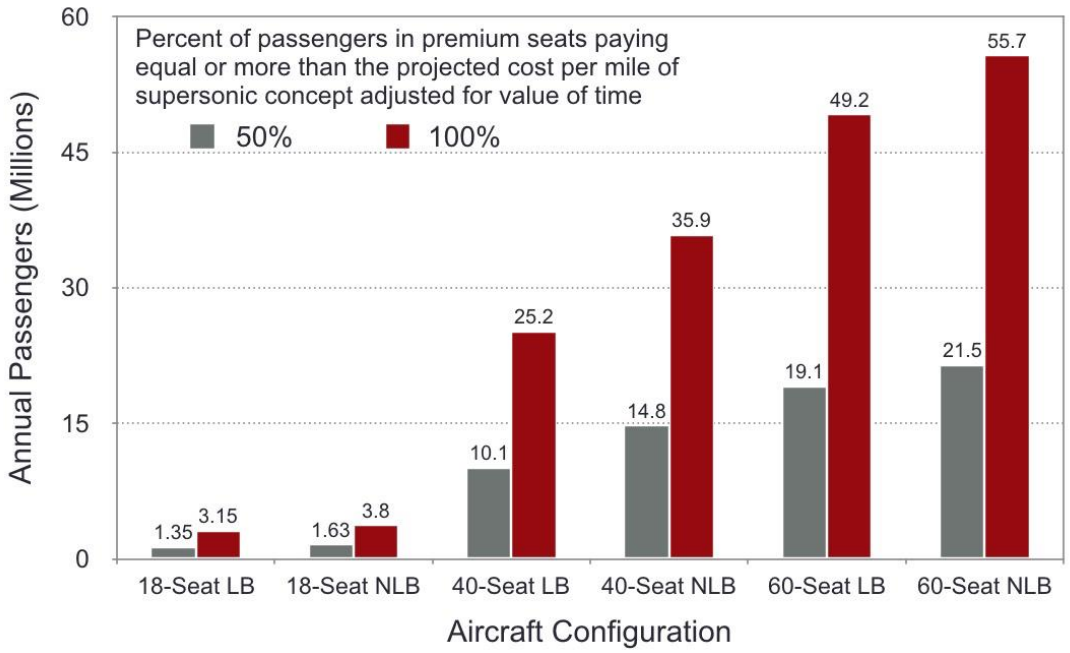


Figure 26: Potential Worldwide Supersonic Passenger Demand in the Year 2040 for Six Aircraft Configurations Studied.

Chapter 3

3 Aviation Global Demand Forecast Model Development – Supersonic Aircraft Market: Study 2

This chapter gathers the information and work done for the second study of the aviation global demand forecast model development for the supersonic aircraft market. Several changes and improvements were made compared to Study 1. Using the results from Study 1, NASA redirects its efforts towards developing a 20-seat and 40-seat. The second study opened the window to improve the previous model further. The two aircraft were analyzed based on low-boom, non-low-boom, and non-low-boom with restrictions performance. NASA's advanced aircraft design changes require a re-evaluation of demand results and a new methodology to obtain them. New assumptions, parameters, and procedures transitioned the model to generate more realistic results.

The analysis consists of three phases. The first phase encompasses the preliminary design and definition of the SST aircraft. SST characteristics and performance are evaluated to achieve an optimal design point using multi-objective criteria. NASA carried out this phase. The second phase estimates the aircraft's operational economics using a life cycle cost model developed and refined by Virginia Tech for this study. The life cycle cost model considers the aircraft performance generated in phase one. The third phase predicts the SST passenger and seat demand between the years 2030 and 2040. This section of the analysis identifies the potential SST market selecting one-way OD pairs that meet a minimum threshold of seats offered on selected routes. The potential number of seats on selected routes is driven by the travel time savings offered by SST services and passengers' potential inclination to pay SST fares estimated from the life cycle cost model. In phase two, the SST fares reflect the cost per passenger mile obtained from the life cycle cost module. The number of aircraft needed is estimated based on forecast demand, network characteristics, and aircraft performance.

3.1 NASA Supersonic Transport Design Concept

To develop a market study, we use three NASA SST aircraft concepts. Table 8 shows the essential characteristics of the aircraft. The table shows a family of aircraft designs ranging in capacity from 20 seats (low-boom concept) to a 40-seat Low-Boom design with an overall length of 242 feet (similar to a Boeing 777-300ER – one of the longest commercial aircraft operating today).

We quickly analyzed airport compatibility and determined that low-boom SST aircraft will have to use gates typically designed for FAA Aircraft Design Group V due to the long overall length. During this project, NASA refined the 40-seat, low-boom design to achieve substantial weight reductions and improved lift/drag ratios. The removal of a short field runway length constraint further enhanced the performance of the aircraft in cruise. Table 8 shows the performance characteristics of the 5th iteration of a low-boom design capable of flying 2,500 nm overland and 3,600 nm overwater. This aircraft configuration was refined over several months by NASA. Table 8 shows the 40-seat non-Low-Boom design's performance characteristics, which are not optimized and represent a different mission design range. For this reason, the potential demand for using low-boom and non-low-boom aircraft needs to be put in context.

Additional information regarding supersonic aircraft airport compatibility and runway length requirements can be found in reference [2].

Table 8: Nasa Supersonic Transport Aircraft Concepts, Study 2. Source: NASA LaRC.

Concept	TOGW	Length, ft.	Wing Area, ft.	Thrust, lbf	Total Fuel, lb	Mach	Design Range, nm
20-passenger Low-Boom							
Overland	121,000	203.2	2,040	26,200	58,782	1.6	2,500
Overwater	113,238	203.2	2,040	26,200	53,059	1.8	3,600
40-passenger Low-Boom							
Overland	159,576	242	3,032	39,000	81,133	1.6	2,500
Overwater	159,401	242	3,032	39,000	80,958	1.8	3,600
40-passenger Non-low-Boom							
Overland	199,000	190	3,682	49,270	105,443	1.15	4,660
Overland	199,000	190	3,682	49,270	105,443	0.95	5,989
Overwater	199,000	190	3,682	49,270	105,443	1.8	5,052

3.2 Travel Time Analysis

Travel demand using supersonic aircraft is expected to be sensitive to travel time. The argument for faster travel speeds offered by supersonic aircraft is the travel time savings and increased productivity for the traveler. Section 5 presents travel demand estimates using a value-of-time approach using the Airline Research Corporation database (ARC). This Section presents minor modifications made to a flight planner application developed by the Air Transportation Systems Laboratory for the Federal Aviation Administration (FAA). The flight planner was designed as part of a Global Oceanic Model (GO Model). There were no significant changes to this analysis when compared to the description presented in Section 2.3

3.3 Value of Time Analysis & Market Fare per Seat-Mile Analysis

For Study 2, a new Value of Time Analysis was implemented. One significant change to the market fare per seat-mile analysis was including economy premium seats from the ARC 2012 data. As described in a previous section, one of the main benefits of supersonic transport is the high travel speed and the travel time savings associated with such travel. Quantifying the potential demand for supersonic services requires investigating the traveler's willingness to pay for additional speed (i.e., travel time savings). The Value of Time Analysis (VOT) concept estimates how travel demand responds to changes in travel time savings. This analysis estimates how much additional money a passenger is willing to pay per hour of travel time saved. The analysis employs an extensive fare database collected by the Airline Research Corporation in 2012 (ARC, 2012).

The ARC 2012 data was used to derive the value of time for three types of markets worldwide. These markets include the US market, the US-International market, and the International only market. Economy premium and premium (first-class and business class) sub-markets were analyzed separately for each main market. Table 9 summarizes the number of airports and one-way origin-destination pairs analyzed for each market.

Table 9: Number of OD Pairs and Airports Analyzed from ARC 2012 Data to Estimate the Value of Time.

Market	Economy Premium		Premium	
	OD Pairs	Airports	OD Pairs	Airports
US	137	54	801	96
US-Int.	30	35	1,063	168
International	20	28	1,118	183

ARC 2012 data provides information about passenger tickets and fares paid. ARC contains ticket records representing single flights (i.e., direct flights) and records with multiple leg flights (i.e., one or more stops). The logic behind the analysis can be explained with the following example. According to the data, a direct flight from Boston Logan International Airport to Shanghai Pudong International Airport costs on average $3,478 for a premium seat. The same flight but with a stop in an intermediate airport cost on average $2,899. The direct flights take about 14.8 hours, while the flight with a one-stop takes about 18.8 hours to reach the final destination. In this example, the data indicate that passengers value their time at $144 per hour. The general trend is that direct flight records are more expensive and faster than one-stop flight records.

A value of time was derived for over 2,500 one-way OD pairs among the three markets. The value of time of each one-way OD pair was organized for each market, and the 50^{th} percentile value was selected as the market value of time to be used in the SST analysis. Figure 27 shows the organized values of time for the US-International premium market and the 50^{th} percentile value of $165/hr. A summary of the values of time derived from this analysis is presented in Table 10. As expected, lower values of time for the economy premium market than the premium market were identified. The analysis showed that the highest value of time is from the US to the International market, followed by the International market. The US market showed to have the lowest value of time, $53.33/hr. The low value could be related to a competitive market in the US and the 2009 US economic recession. Since the ARC data is from the year 2012, the economic recession could still affect the fares offered by the airlines or the passenger's willingness to pay behavior. For the SST analysis, a higher value of $61.20/hr. The Department of Transportation (DOT) recommended value for aviation passenger travel time for the US premium market was selected instead of the value of time derived from the ARC data.

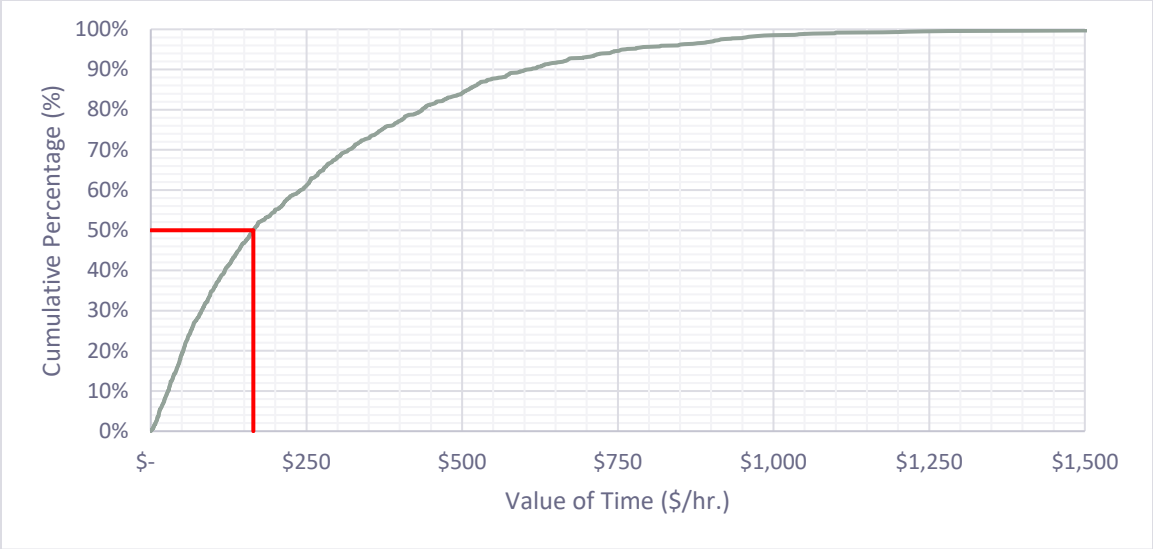


Figure 27: US-International Premium Market Cumulative Value of Time.

Table 10: Value of Time Results, Study 2.

Market	Economy Premium	Premium
US	$39.58/hr.	$61.20/hr.
US-International	$29.68/hr.	$165.59/hr.
International	$43.61/hr.	$140.19/hr.

Figure 28 shows graphically the analysis performed with ARC 2012 data. The purpose of this analysis is to obtain the percent of passengers willing to pay fares at or above a given cost per passenger mile threshold. The analysis considers fares paid from ARC 2012, travel time savings by aircraft type, and value of time depending on the market to generate cumulative fare plots.

The first step in the analysis was to read and parse the raw data and create a database. Next, we extracted economy premium and premium records from more than 300 million records. Premium records include first-class and business class tickets. We assumed that passengers paying for economy tickets would not be candidates for the SST service in the analysis. This assumption was validated with the life cycle cost model. More than 99% of the economy class records involve tickets with fares per mile below $0.50, a price point below the best solution obtained with the life cycle cost model described in this report. Further analysis of premium records identified records that met three criteria: a) distance between origin and destination was more than 1,000 statute miles (sm.); b) records that involved round trips; and c) records with fare per mile value more than $0.20

per seat-mile. These filters produced over 2 million records from the original 300 million records in the ARC database.

The first criteria imply that in short trips (< 1,000 statute miles), the travel time savings provided by supersonic aircraft flights would make the shift from subsonic flights unfeasible. Trip distances less than 1,000 statute miles involve mainly overland flights. Using Mach 0.95 per current regulations for overland flights, the travel savings on a short trip are estimated to be 30 minutes or less. The second criteria (i.e., round trips) was applied to avoid errors when calculating fare per mile paid by passengers. The third criterion used in this analysis was to remove all those records with fare per mile paid less than $0.20. Records identified as premium in the database but having fare per passenger mile less than $0.20 are considered upgrades and hence ignored.

The final step in the analysis is to create a cumulative fare per passenger mile plot for each one-way OD pair to estimate the percent of passengers willing to pay fares at or above a given cost per passenger mile threshold level. A minimum of 50 records is needed to generate each plot. A generic cumulative fare per passenger mile plot is generated for each of the three markets to be used when there is no available data for a given one-way OD pair. The generic plot is created by combining all the available records of each market.

Each cumulative fare per passenger mile plot is adjusted to account for the passenger's willingness to pay a higher fare for faster services. These fare per passenger mile plots are used later in the analysis for one-way OD pairs containing ARC and OAG data. There could be one-way OD pairs that do not have a cumulative fare per passenger mile plot because either there were less than 50 records or no records at all in the ARC 2012 data. The reason to have a one-way OD pair with no records is a route that showed up in the OAG 2016 data that did not exist in the year 2012 (ARC 2012 data). The generic plot is used and adjusted for travel time savings for those one-way OD pairs with no cumulative fare per passenger mile plot.

We developed an algorithm to consider the passenger's willingness to pay higher fares for faster service using the ARC data. The model calculates the travel time savings by aircraft type (Non-Low-Boom, Non-Low-Boom restricted, and Low-Boom) and for each one-way OD pairs that contain enough records to generate a reasonable cumulative fare per mile plot. These cumulative plots are generated with consideration of the value of time. The percent of passengers willing to

pay that is obtained from the cumulative plots is a function of cost per passenger-mile and travel time savings offered by the supersonic aircraft.

Figure 29 shows an example of the cumulative fare per passenger mile plot for three one-way OD pairs. From John F. Kennedy International Airport to Los Angeles International Airport (JFK-LAX) and Heathrow Airport (JKF-LHR), and from Dubai International Airport to Singapore Changi Airport (DXB-SIN). The curves are the ARC 2012 data results, the value of time analysis, and the travel time benefits offered by the supersonic transport. For this example, assuming a fare per passenger mile of $0.75, the willingness to pay would be 90%, 47%, and 22%, respectively.

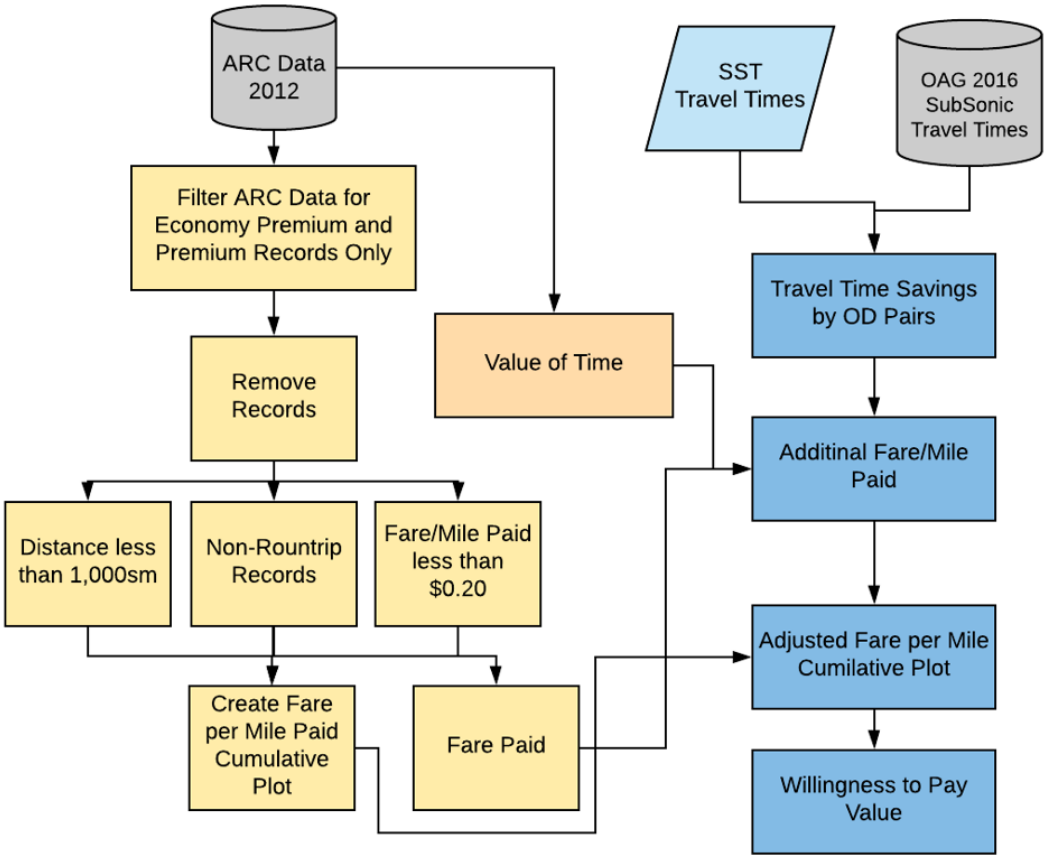


Figure 28: Flowchart of Airline Reporting Corporation 2012 Analysis, Study 2.

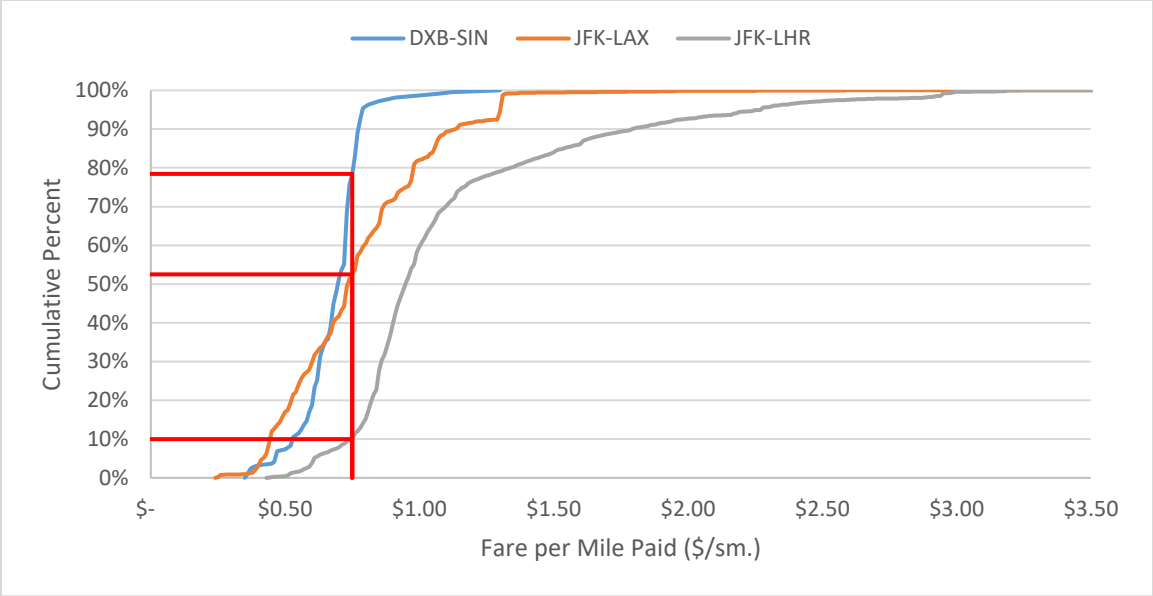


Figure 29: Cumulative Fare per Mile Paid Plot Using ARC 2012 Data.

3.4 Worldwide Commercial Air Travel Demand Forecast Model

OAG 2016 data has been used as the baseline year for this analysis. The methodology from the ICAO Long-range Traffic Forecast, Passenger and Cargo, July 2016 has been adopted to generate a forecast model that predicts future passenger demand for years 2030 to 2040. The ICAO document contains equations developed to forecast air traffic growth within and between eleven world regions. The eleven regions include North America, Central America and Caribbean, South America, Europe, North Africa, Sub Saharan Africa, Middle East, Central, and Southwest Asia, North Asia, and Pacific South East Asia. The same forecast model from Study 1, described in Section 2.5, was used for Study 2.

3.5 Worldwide Supersonic Air Travel Demand Forecast Model

The model generates future projections of premium seats worldwide by using the ICAO forecast seats. As shown in Figure 13: ICAO Worldwide Forecast Model, Number of Seats Over Time., the forecast estimates the number of seats that will be offered worldwide from the year 2017 to the year 2040. In consultation with NASA, we assume that SST aircraft will be introduced to the worldwide network in the year 2030. Hence the ICAO forecast data spans between the year 2030 and the year 2040. Figure 30 describes the analysis performed using OAG 2016 data and the ICAO 2030-2040 forecast to predict potential demand for commercial supersonic services. Using 2016

OAG data as the base year, we extracted the total number of premium seats and total seats for each one-way OD pair. We then estimated the ratio of premium seats to total seats (i.e., indicates the percent of premium seats from the total seat count). We combined the ICAO forecast with the OAG data to match all the one-way OD pairs from both datasets. The ICAO forecast predicts the total number of seats in the future. The ICAO forecast does not predict the number of future premium seats. For this reason, the economy premium and premium seat to total seat ratios are used to estimate the number of these types of seats to be offered in each of the one-way OD pairs for the years 2030 to 2040. A change in the analysis was the aggregation of economy premium records.

To find the potential market for SST flights over time, we employed three criteria to further narrow down the one-way OD pairs. The first criteria consider one-way OD pairs with a route distance greater than 1,000 sm. The second criteria identify one-way OD pairs with total seats offered greater than 100,00 annual seats. The rule was established based on the assumption that one-way OD pairs with a small demand would not be able to support reliable SST services (i.e., five times per week). The third criterion applied was to keep all one-way OD pairs with more than the minimum demand established based on aircraft seating capacity and annual operations. For the number of annual operations, the assumption is one flight per day and three hundred days a year as a minimum. For the aircraft seating capacity, the analysis considers a 40 seat configuration. The minimum demand for premium seats for a 40-seat transport requires 12,000 premium seats per year (combining economy premium and premium seat forecast).

The analysis also considers design tradeoffs using Low-Boom and non-Low-Boom aircraft and three cruise speed profiles. The three aircraft type combinations are labeled Non-Low-Boom aircraft, Non-Low-Boom restricted aircraft, and Low-Boom aircraft. The Non-Low-Boom aircraft flies at Mach 1.15 overland at Mach 1.8 overwater. The Non-Low-Boom restricted aircraft flies at Mach 0.95 overland and Mach 1.8 overwater. The Low-Boom aircraft flies at Mach 1.6 overland and Mach 1.8 overwater. The model generates potential markets for SST commercial transports for each of the three aircraft designs and sub-markets (US, US-International, and International only). The methodology to calculate the number of aircraft needed to satisfy the forecast demand remained the same as the one presented in Section 2.6.1 during Study 1.

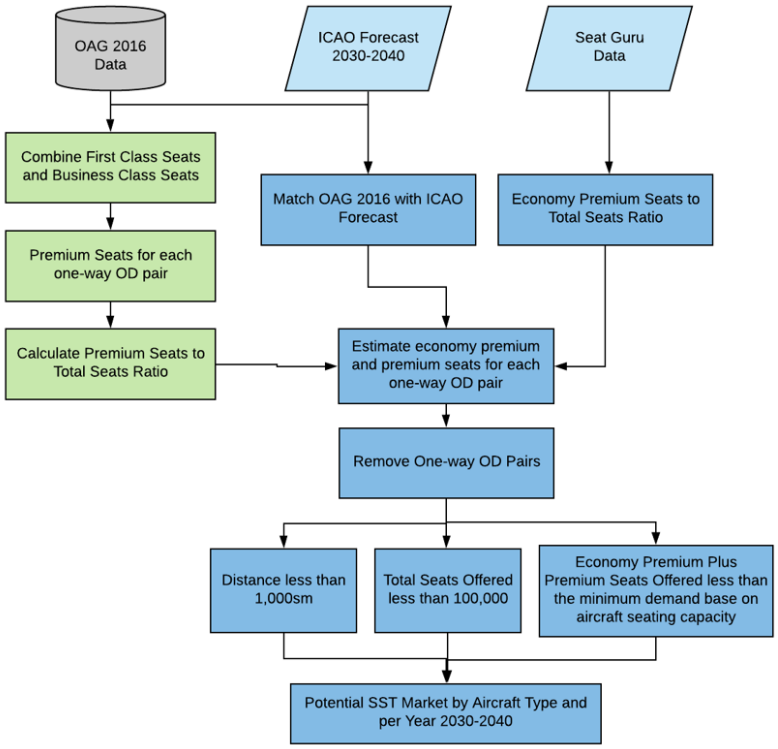


Figure 30: Flowchart of OAG 2016 and ICAO Forecast Analysis to Estimate Potential SST Market, Study 2.

3.6 Results

The model described in previous sections of the report was applied to three SST aircraft designs using the life cycle cost model described in reference [2]. The results presented in this section include the following variations:

- 40-Seat Aircraft
    - Non-Low-Boom SST Design
    - Non-Low-Boom restricted SST Design
    - Low-Boom SST Design

- 20-Seat Aircraft
    - Low-Boom SST Design
    - Non-Low-Boom restricted SST Design
    - Low-Boom SST Design

The results combined the SST projections for the US, US-International, and International only markets (worldwide market). The predictions on each graph for the 40-seat Low-Boom SST design should not be compared with the 40-seat non-low-boom, and 40-seat non-low-boom restricted SST designs. The aircraft performance and design optimization are not considered to be equal for them to be compared. Further improvement to the non-low-boom and non-low-boom restricted is needed to be compared with the Low-Boom SST design projections. Similarly, projections for the 20-seat Low-Boom SST aircraft should not be compared to the 20-seat Non-Low-Boom and Non-low-boom restricted.

3.6.1 40-Seat Supersonic Transport Results with Moderate Market Share - Worldwide Projections.

The results presented in this section assume that 50% percent of potential SST passengers worldwide pay equal to or more than the projected cost per mile of the 40-seat supersonic concept adjusted for the value of time.

Figure 31 presents the number of potential one-way OD pairs worldwide for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft over time. In 2030, we estimate 940, 876, and 1,052 potential one-way OD pairs for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft, respectively. With an average of 3.85%, 3.60%, and 3.99% annual increase in the

number of potential one-way OD pairs, by the year 2040, there could be 1,371, 1,247, and 1,555 potential one-way OD pairs for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft, respectively. Figure 32 shows the percent of the flight track that is flown overland vs. distance. The flight track of 34% of one-way OD pairs worldwide is at least 50% overland. The flight track of 8% of the one-way OD pairs worldwide is at least 100% overland. The flight track of 8% of the one-way OD pairs worldwide is at least 100% overwater.

Figure 33 presents the potential seat demand worldwide for non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft over time. In 2030, we estimate 23.9, 22.4, and 27.9 million seats for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom market, respectively. With an average of 5.27%, 5.13%, and 5.38% annual increase in the number of passengers, by the year 2040, there could be 40, 36.9, and 47.1 million seats for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft, respectively.

Figure 34 presents the number of non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft needed to satisfy the demand worldwide over time. In 2030, we estimate 935, 875, and 1,038 non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft required over time, respectively. With an average of 5.13%, 5.01%, and 5.15% annual increase in the number of aircraft by 2040, there could be 1,443, 1,525, and 1,715 non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft needed over time, respectively.

Figure 35 presents the potential passenger demand worldwide for non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft over time. In 2030, we estimate 19.5, 18.3, and 22.8 million passengers for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom market, respectively. With an average of 5.27%, 5.13%, and 5.38% annual increase in the number of passengers, by the year 2040, there could be 32.6, 30.1, and 38.4 million passengers for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft, respectively. The aircraft load factor for the US market used is 85%. An 81.5% aircraft load factor was used for both the US-International market and the International only market.

3.6.2 20-Seat Supersonic Transport Results with Moderate Market Share - Worldwide Projections

The results presented in this section assume that 50% percent of potential SST passengers worldwide pay equal to or more than the projected cost per mile of the 20-seat supersonic concept adjusted for the value of time.

Figure 36 presents the number of potential one-way OD pairs worldwide for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft over time. In 2030, we estimate 624, 576, and 564 potential one-way OD pairs for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft, respectively. With an average of 4.41%, 4.24%, and 4.34% annual increase in the number of potential one-way OD pairs by 2040, there could be 960, 872, and 862 potential one-way OD pairs for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft, respectively.

Figure 37 presents the potential seat demand worldwide for non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft over time. In 2030, we estimate 8.4, 7.7, and 7.9 million seats for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom market, respectively. With an average of 5.14%, 4.99%, and 5.17% annual increase in the number of passengers, by the year 2040, there could be 13.9, 12.9, and 12.8 million seats for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft, respectively.

Figure 38 presents the number of non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft needed to satisfy the demand worldwide over time. In 2030, we estimate 667, 703, and 608 non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft required over time, respectively. With an average of 5.12%, 5.04%, and 5.17% annual increase in the number of aircraft, by the year 2040, there could be 1,099, 1,149, and 1,006 non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft needed over time, respectively.

Figure 39 presents the potential passenger demand worldwide for non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft over time. In 2030, we estimate 6.9, 6.5, and 6.3 million passengers for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom market, respectively. With an average of 5.14%, 4.99%, and 5.17% annual increase in the number of passengers, by the year 2040, there could be 11.4, 10.5, and 10.4 million passengers for the non-

Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft, respectively. The aircraft load factor for the US market used is 85%. An 81.5% aircraft load factor was used for both the US-International market and the International only market.

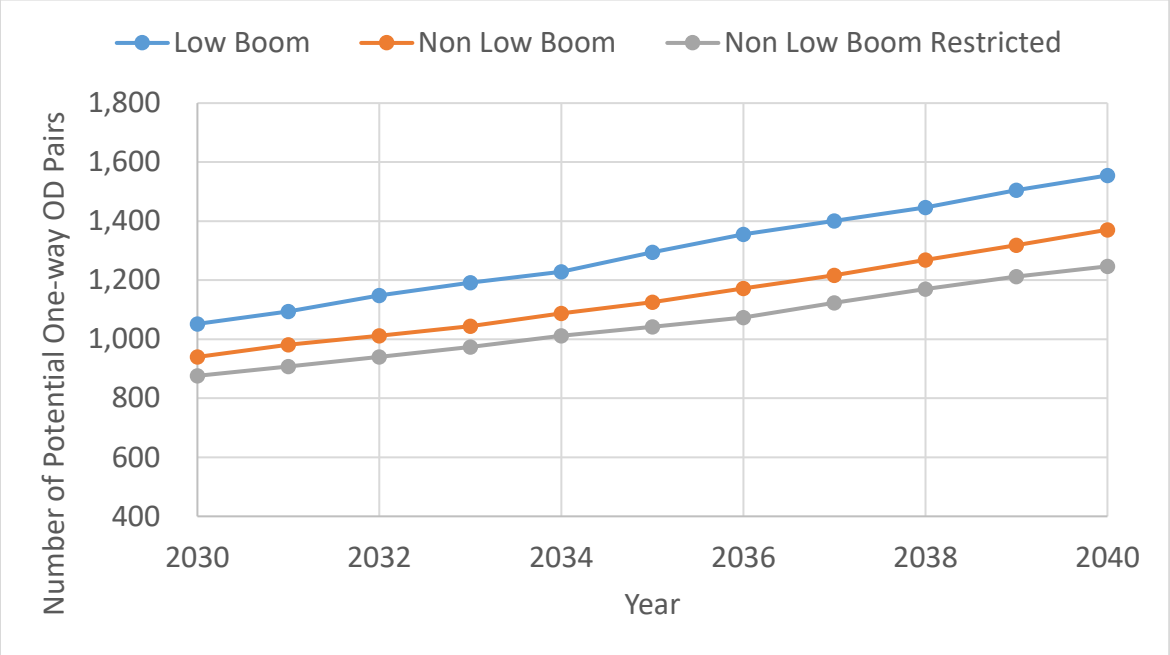


Figure 31: Number of Potential One-way OD Pairs Over Time. 50% of Potential SST Passengers Would Pay Equal or More than the Projected Cost per Mile of the 40-seat Supersonic Aircraft Adjusted for the Value of Time. Worldwide Projection.

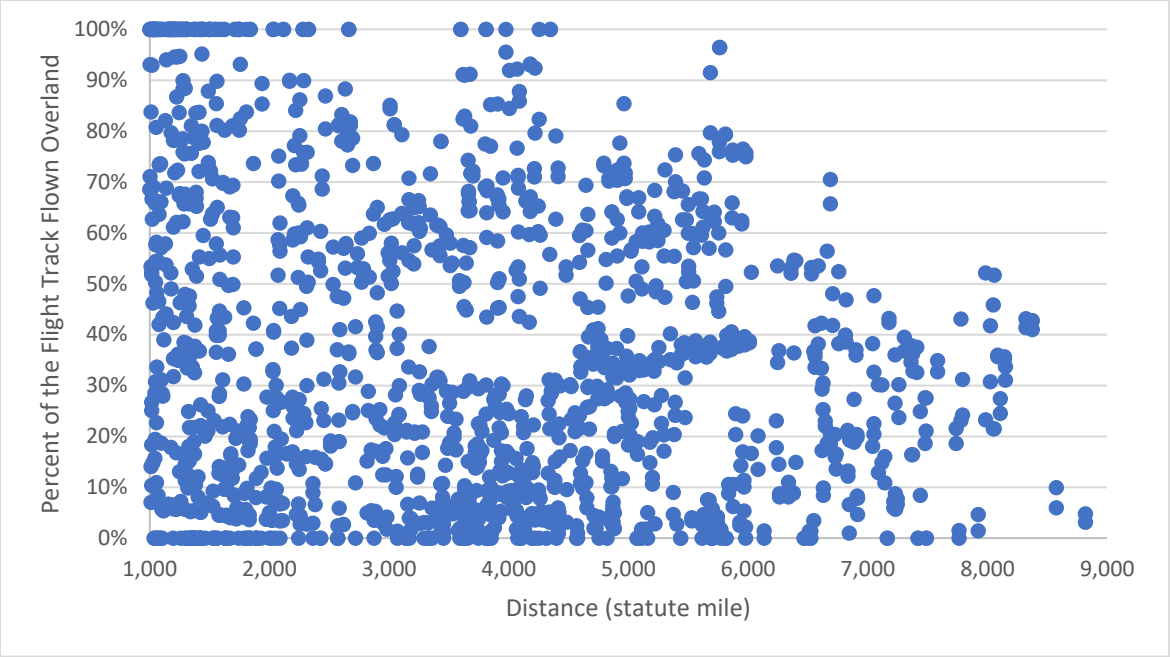


Figure 32: Percent of the Flight Track Flown Overland vs. Distance of all One-Way OD Pairs Forecast for the Low-Boom SST Design During the Year 2040 for the Worldwide Market.

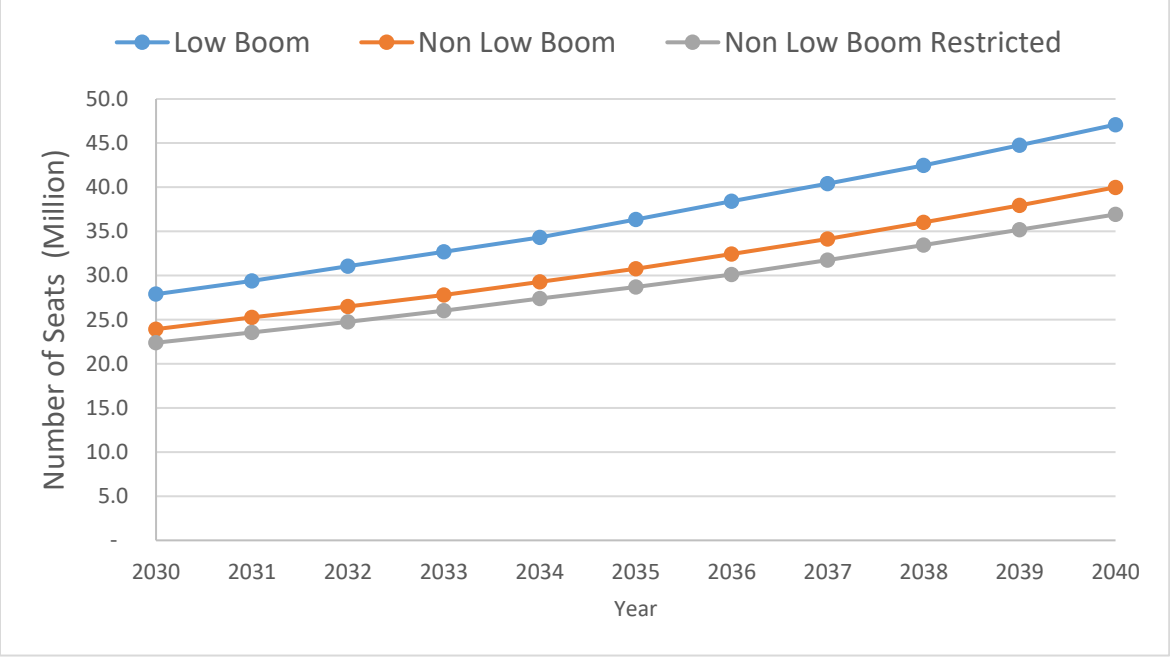


Figure 33: Estimated Seat-Demand Over Time. 50% of Potential SST Passengers Would Pay Equal or More than the Projected Cost per Mile of the 40-seat Supersonic Aircraft Adjusted for the Value of Time. Worldwide Projection.

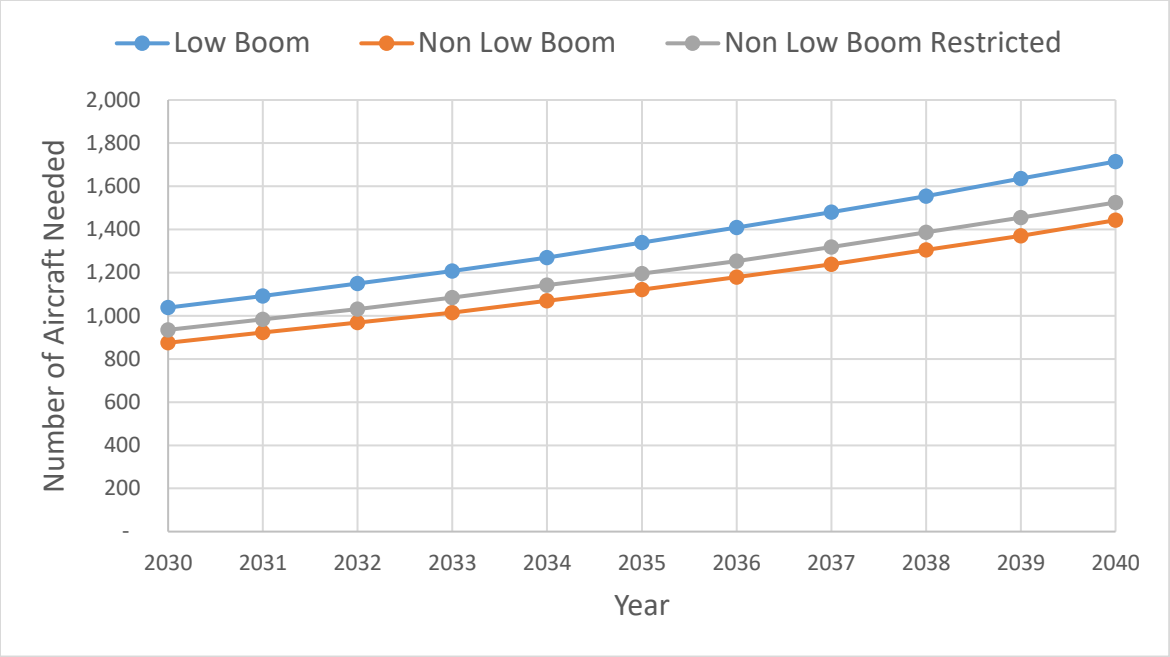


Figure 34: Number of Aircraft Needed Over Time. 50% of Potential SST Passengers Would Pay Equal or More than the Projected Cost per Mile of the 40-seat Supersonic Aircraft Adjusted for the Value of Time. Worldwide Projection.

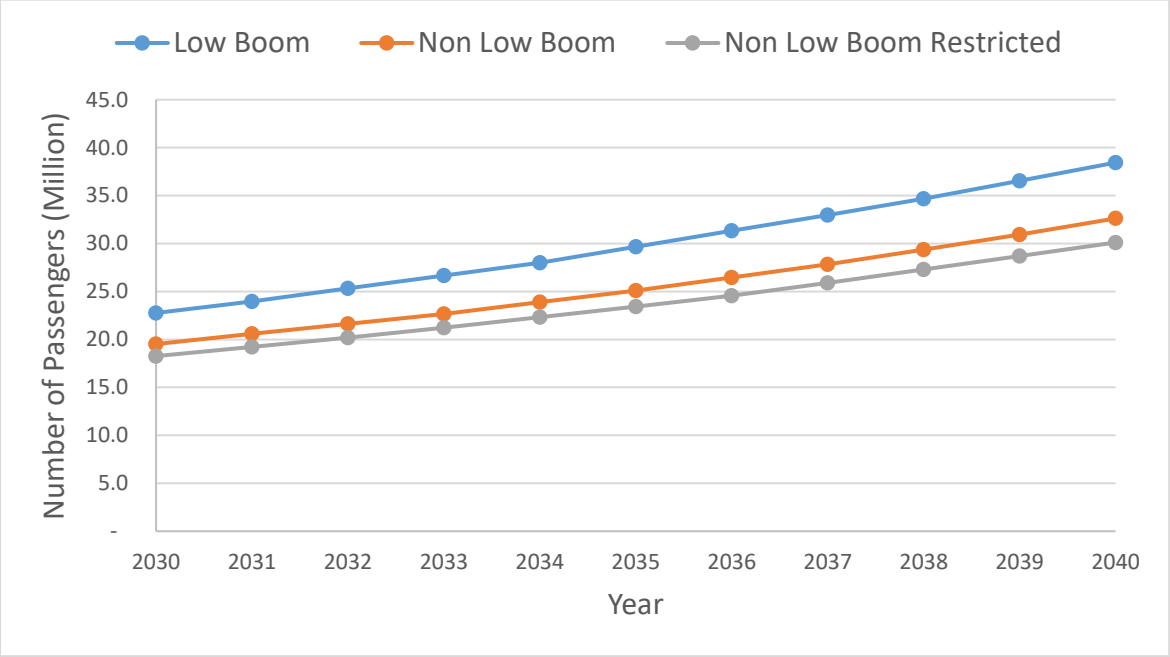


Figure 35: Estimated Passenger-Demand Over Time. 50% of Potential SST Passengers Would Pay Equal or More than the Projected Cost per Mile of the 40-seat Supersonic Aircraft Adjusted for the Value of Time. Worldwide Projection.

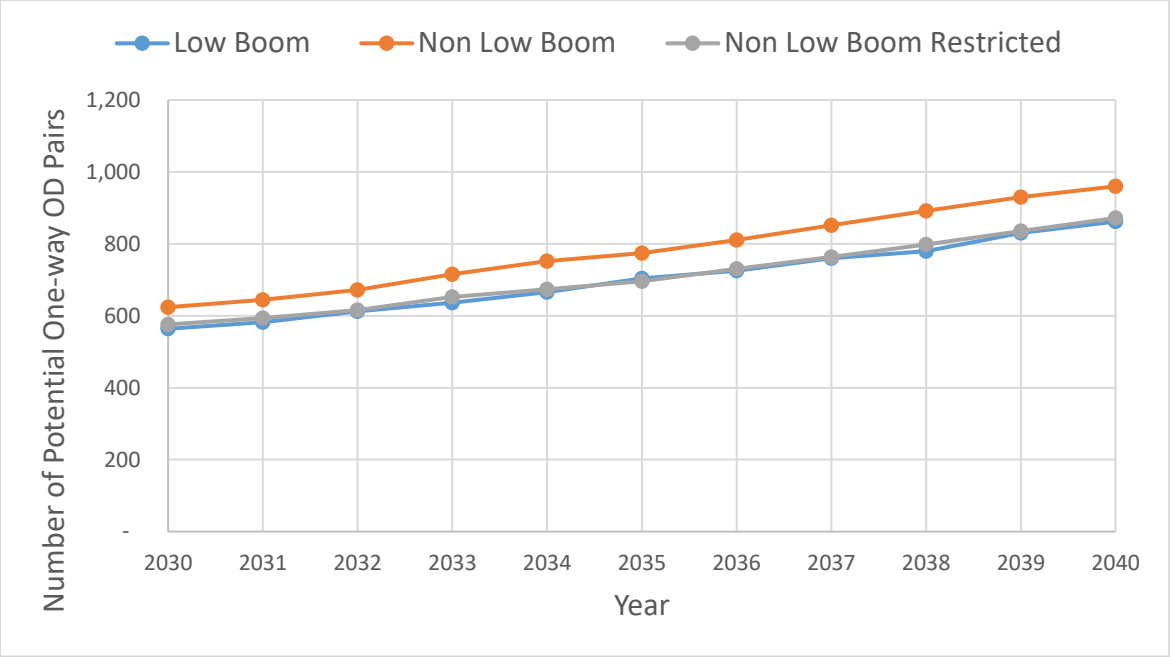


Figure 36: Number of Potential One-way OD Pairs Over Time. 50% of Potential SST Passengers Would Pay Equal or More than the Projected Cost per Mile of the 20-seat Supersonic Aircraft Adjusted for the Value of Time. Worldwide Projection.

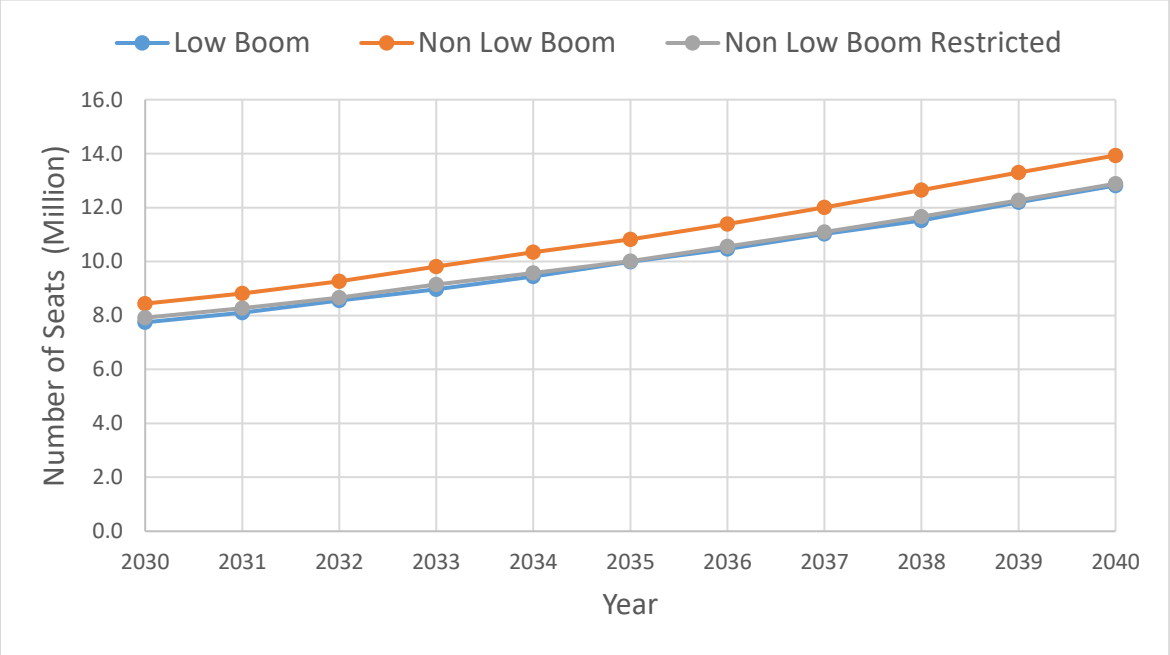


Figure 37: Estimated Seat-Demand Over Time. 50% of Potential SST Passengers Would Pay Equal or More than the Projected Cost per Mile of the 20-seat Supersonic Aircraft Adjusted for the Value of Time. Worldwide Projection.

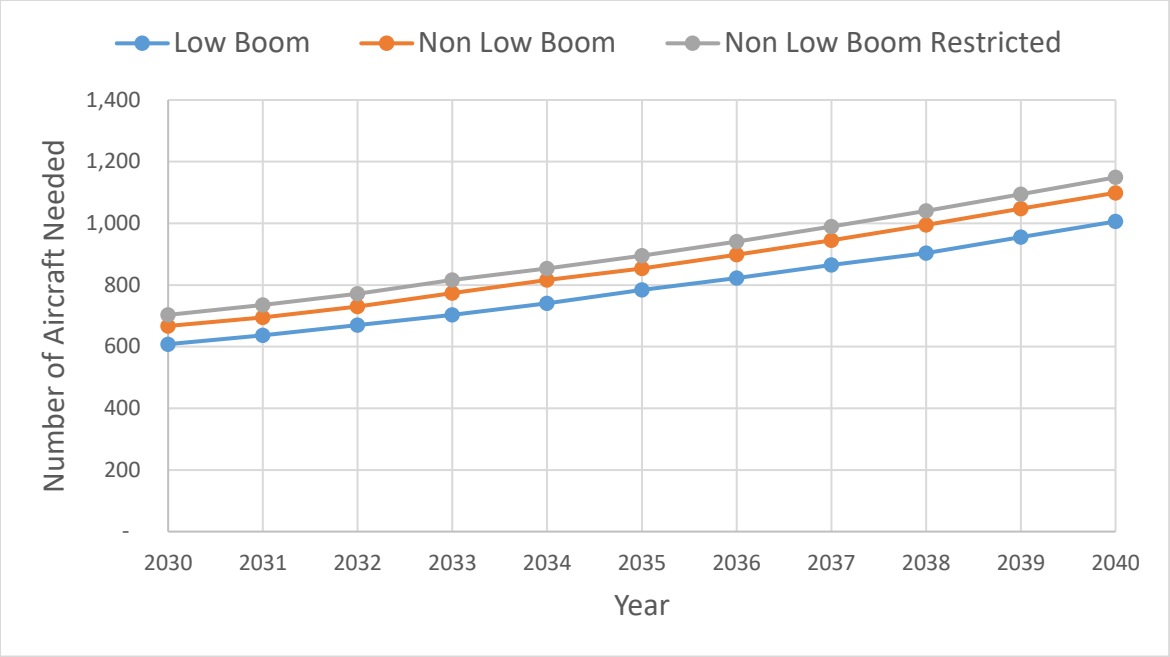


Figure 38: Number of Aircraft Needed Over Time. 50% of Potential SST Passengers Would Pay Equal or More than the Projected Cost per Mile of the 20-seat Supersonic Aircraft Adjusted for the Value of Time. Worldwide Projection.

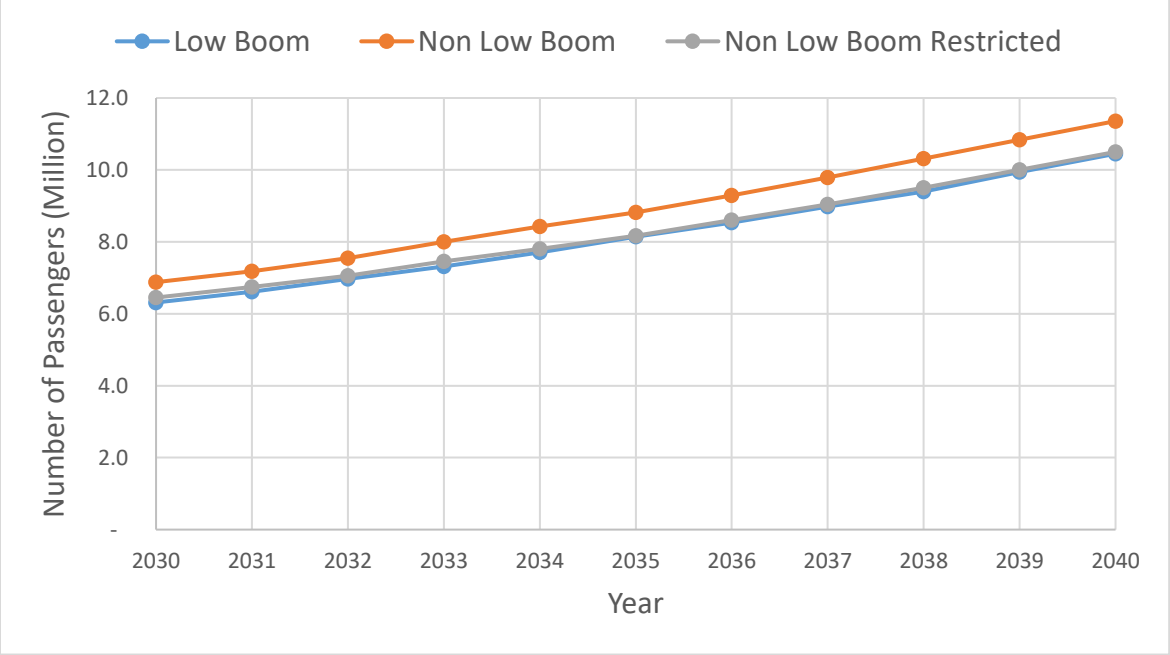


Figure 39: Estimated Passenger-Demand Over Time. 50% of Potential SST Passengers Would Pay Equal or More than the Projected Cost per Mile of the 20-seat Supersonic Aircraft Adjusted for the Value of Time. Worldwide Projection.

Chapter 4

4 Aviation Global Demand Forecast Model Development – Supersonic Aircraft Market: Study 3

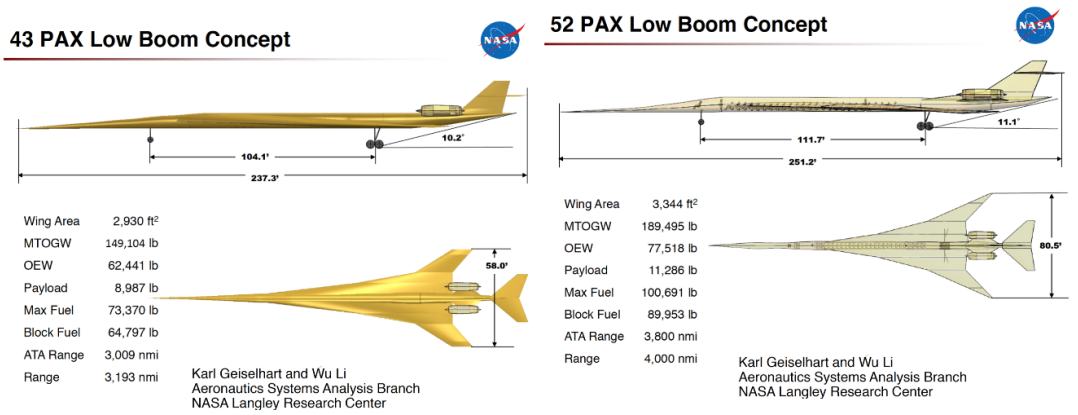
This report contains a preliminary analysis of low-boom supersonic aircraft commercial operations worldwide. The research presented estimates demand and the number of aircraft needed to fill the demand for supersonic air travel and annual aircraft operations worldwide. Multiple Low-Boom Supersonic Aircraft (LBSA) concepts are considered in this analysis. Specifically, we considered NASA optimized 43 and 52-seat LBSA aircraft with variations in overland range, aircraft utilization, and fuel consumption profiles. The figure shows six distinct modules comprising the methodology to estimate worldwide LBSA aircraft demand. The integrated model to estimate LBSA aircraft demand is called Low-Boom Supersonic Aircraft Demand Model (LBSAM).

The analysis uses the Official Airline Guide (OAG) data to estimate the number of premium seats offered worldwide across more than 56,000 origin-destination pairs. Similarly, we use the Airline Reporting Corporation (ARC) data to evaluate premium fares distribution across the same number of origin-destination pairs worldwide. NASA provided LBSA aircraft performance in the form of two detailed LBSA aircraft designs with 43 and 52-seats and surrogate models that approximate the performance of slight variations of the two optimized designs. Using the aircraft performance provided by NASA, we developed a simple aircraft development cost model to estimate each LBSA aircraft's unit cost given an estimated worldwide demand. A life-cycle cost model estimates the operational cost per hour and cost per passenger mile of each LBSA aircraft design using the unit cost estimated by the LBSA aircraft cost model. Finally, LBSA demand is calculated using the distribution of premium fares from ARC data and the cost per passenger mile estimated by the life cycle cost model. The integrated modeling process includes a critical feedback loop to predict the final LBSA aircraft demand. Initial LBSA aircraft unit cost estimates are refined each time the worldwide demand module runs. Higher worldwide demand requires more LBSA aircraft, and hence the unit cost of the aircraft changes in every iteration. This process is repeated until equilibrium is reached between the number of LBSA aircraft needed to satisfy worldwide demand and the number of aircraft produced.

4.1 NASA Supersonic Transport Design Concept

We use NASA's Low-Boom Supersonic Aircraft (LBSA) surrogate models to develop a market study based on two FLOPS optimized vehicle designs with 43 and 52 seats. Figure 40 shows two optimized LBSA aircraft designs ranging in capacity from 43 to 52 seats. Both aircraft are capable of flying 3,800 nautical miles over water. Both designs' striking feature is their overall length (237.3 and 251.2 feet, respectively), similar to a Boeing 747-8i – the longest commercial aircraft operating today. In a later section of the report, we quickly analyzed airport compatibility. We determined that LBSA aircraft will use gates typically designed for FAA Aircraft Design Groups V and VI due to the long overall length (251.2 feet for a 52-seat LBSA design). During this project, NASA refined multiple times two detailed LBSA aircraft designs with 43 and 52 seats. Similarly, NASA provided two LBSA aircraft surrogate models that predict the aircraft performance of variations of the two FLOPS optimized designs. The first surrogate model estimates

Additional information regarding NASA's supersonic aircraft parameters and characteristics, along with airport compatibility, can be found in reference [3].



Parameter	Low Boom 43-PAX	Low-Boom 52-PAX
Thrust (lbf)	35,050	42,993
Operating Empty Weight (lb)	62,441	77,518
Maximum Speed (knots)	1,056	1,056
Aircraft Unit Cost (Million $2020)	**145.6**	**169.5**

Figure 40: NASA FLOPS Optimized Low-Boom Aircraft Concepts Studied. Source: NASA LaRC.

4.2 Travel Time Analysis

Travelers make decisions on specific modes of transportation, considering published travel times between two airports. These published travel times include flight times, taxi-in, taxi-out, and additional travel times to account for recurrent congestion at airports. To generate realistic travel times from the point of origin to the point of destination, the flight planner travel times are supplemented with additional travel times to correct for the climb and descent phases of flight. The travel times estimated on the flight planner do not account for taxi, climb, or descent times. We added 6 and 23 minutes to the climb and descent phases to correct the flight planner's coplanar travel time estimates to account for the additional climb and descent travel times.

To account for taxi times, two linear models are used to predict taxi-in and taxi-out times for supersonic aircraft. These linear models use Federal Aviation Administration Aviation Systems Performance Metrics (ASPM) data for 77 airports in the United States (ASPM, 2018). The models consider the number of arrivals to estimate taxi times. The number of annual operations at airports is obtained from OAG 2016 data. Figure 8 shows the linear regression model to estimate taxi-in times. Figure 9 shows the linear model to predict taxi-out times. For airports with no annual operations data in OAG, we assign minimum taxi-in and taxi-out times of 4 and 11 minutes, respectively. These are the taxi-in and taxi-out values predicted by the linear models for very low annual operations.

The LBSA has different overland and overwater characteristics. During Study 1, the overland characteristic was assumed for the continental US market and overwater characteristics for the US-International and International markets. For Study 2, all the one-way OD pairs with a distance of 2,500 nm. or less were assumed to have overland, while the rest were considered to have overwater characteristics. This assumption overestimated the US-International and International LBSA demand. LBSA overwater characteristics means better aircraft performance when compared to LBSA overland characteristics. The better aircraft performance results in lower fare per mile values and lower travel times (higher travel time-saving benefit), increasing demand. For Study 3, the travel time analysis module was improved.

Figure 41 shows the flowchart of the travel time analysis module. The improvement made to this analysis in Study 3 was to assign aircraft characteristics base on route characteristics. The flight

planner application was improved to show the aircraft's speed, and the distance traveled at the specified speed. Using these two variables, we calculate the percent of the trajectory that the aircraft fly overland. LSBA overland characteristics to any given one-way OD pairs are made only if the flight trajectory is at least 25% overland. LSBA overwater characteristic is selected exclusively for those one-way OD pairs with a flight trajectory of 75% or more overwater. Once the right LBSA characteristic is determined, the cruise travel time obtained from the flight planner application is adjusted as described in (Eq. 10) to calculate the final LBSA travel time.

$$\begin{aligned} &Final\ LBSA\ TraveTime \\ &\quad = \ Cruise\ Travel\ Time \\ &\quad + \ (TaxiOut + DeltaClimb \ + DeltaDescent + TaxiIn) \\ &\quad * Roundup\left(\frac{route\ distance}{aircraft\ range}\right) + [(RefuelingTime \\ &\quad * Roundup\left(\frac{route\ distance}{aircraft\ range}\right) - 1)] \end{aligned} \qquad \textbf{(Eq. 10)}$$

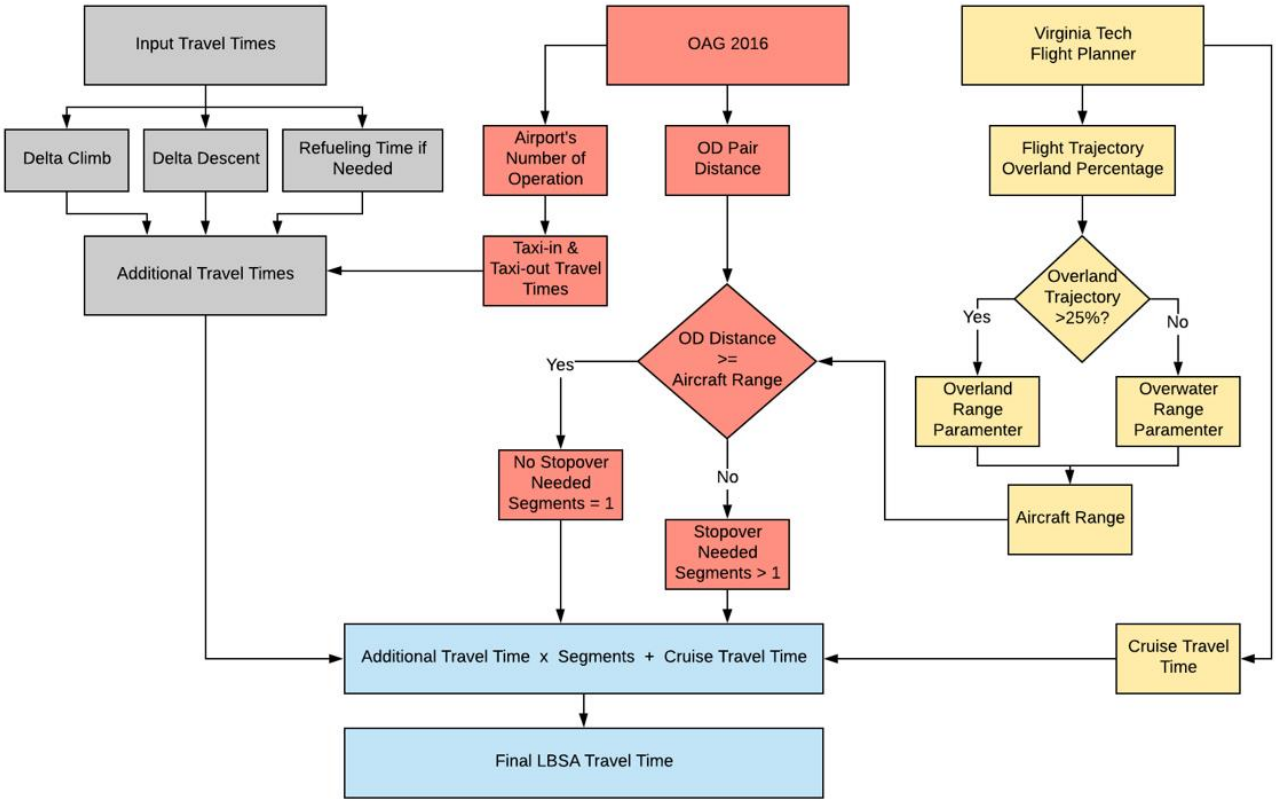


Figure 41: Flowchart of the Travel Time Analysis Module.

4.3 Value of Time Analysis & Market Fare per Seat-Mile Analysis

The general description of the value of time analysis and market fare seat-mile analysis is described in Section 2.4 and Section 3.3. However, for Study 3, several changes to these two modules were implemented. For this study, we used the ARC 2016 data. Once again, the data was used to derive the value of time for three types of markets worldwide. These markets include the US market, the US-International market, and the International only market. From Study 2, we learned that only a small fraction of the LSBA demand could come from the economy premium market. Since the OAG 2016 was purchased, and the price was dependent on the size of the data (total number of records to be delivered), we decided to only request records from the business and first-class markets. Table 11 summarizes the number of airports, one-way origin-destination pairs, records analyzed for each market. The records obtained from ARC 2016 were one-way non-stop, one-way with one-stop, and roundtrip records (with no stop). Since the economics of one-way vs. roundtrip purchases are different, to be consistent in the analysis, the value of time analysis only compares one-way non-stop records vs. one-way one-stop records.

Table 11: Number of Airports, OD Pairs, and Records Analyzed by Market from ARC 2016 Data.

Market	Airports	OD Pairs	Records
US	135	1,535	8.14 million
US-International	327	2,709	9.89 million
International	1,008	12,176	27.19 million

As shown in Figure 42, the model calculates a weighted average at the airport level after the value of time calculation at the OD pair level. Several changes were made to calculate the value of time at the OD pair level compared to the process implemented in Study 1 and Study 2. As mentioned before, only one-way records were used in the calculation. For the US market, a fare per mile paid the limit of $1.50/mi was implemented. For the US-International and International markets, we keep the $3/mi. limit. All records above these fare per mile threshold were eliminated from the analysis. Also, the $0.20/mi minimum criteria was eliminated from the analysis.

Another significant change was related to the travel times assumed for the one-way one-stop records. In the past, we used OAG 2016 travel times with a one-hour (US market) or two hours (US-International and International markets) layover time at the stopover airport. This assumption

can benefit or act as a disadvantage for some routes depending on accurate airline flight schedules. For routes with a low number of flights per day, assuming a short layover time would benefit the route. For example, a passenger traveling from Charlotte Douglas International Airport (CLT) to Hamad International Airport (DOH) in Doha, Qatar, with a stopover at John F. Kennedy International Airport (JFK), would experience a 4.8 hour layover time at JFK. Using a 2-hour layover instead of the actual 4.8 would increase the total gate-to-gate travel time for the CLT-JFK-DOH route. Looking at the value of time (Eq. 1), a higher travel time of the one-way with one-stop trips (records when using ARC data) would result in a lower travel time-saving benefit, which in return would overestimate the value of time.

On the contrary, using a two-hour layover time when actual flight schedules allow for shorter stopover times would underestimate the value of time. For example, a passenger flight from Hartsfield-Jackson Atlanta International Airport (ATL) to Madrid-Barajas Adolfo Suarez Airport (MAD) with a stopover at Amsterdam Airport Schiphol (AMS) would experience a 55-minute layover at AMS. The use of the two hours instead of the 55 minute layover time would increase the travel time-saving benefit, which would result in underestimating the value of time.

$$WAVOT = \frac{\sum_{j=1}^{n} VOT_{ij} * R_{ij}}{\sum_{j=1}^{n} R_{ij}} \quad \textbf{(Eq. 11)}$$

Where,
WAVOT = Weighted Average Value of Time
VOT_{ij} = Original VOT calculated at the one-way OD pair level from airport (i) to airport (j).
R_{ij} = Number of records from airport (i) to airport (j).

The overestimation or underestimation of the value of time has a high impact on the demand projection. In Study 3, a new approached to account for the layover time was implemented to avoid this issue. We used a network analysis using OAG 2016 flight schedules. For every one-way OD pair, we selected 100 itineraries and recorded the total travel time. This total travel time includes the gate-to-gate travel time from the airport of origin to the intermediate airport, layover time at the stopover airport, and the gate-to-gate travel time from the stopover airport to the airport of destination. We then calculated the median total travel time to be used in the value of time calculation for the one-way one-stop records. In this study, a value of time limit of $400/hr. was

implemented. For one-way OD pairs with a value of time higher than $400/hr. the calculated value was neglected, and the $400/hr. was assigned.

Figure 42 shows an additional step after the value of time calculation at the OD pair level. In Study 1, we used a generic value of time for each of the markets. The model assigned this value of time to all the one-way OD pairs within the same market. In Study 2, we enhance the calculations by using one-way OD pairs specific values of time. For Study 3, while brainstorming on improving this part of the model, we decided to change the logic behind the model. The passenger's value of time should not be route-dependent. A traveler flying from ATL to Los Angeles International Airport (LAX) should have the same value of time as if is flying from ATL to Dallas/Fort Worth International Airport (DFW).

For this reason, we used the values of time calculated at the one-way OD pair level and recalculated a weighted average value of time at the airport level. Knowing the number of records used at each one-way OD pair, we used (Eq. 11) to calculate the weighted average value of time at the airport level. For example, the initial value of time estimated for JFK varied from less than a dollar per hour to up to $249/hr. After the new methodology was implemented, JFK had a weighted average value of time of $165/hr. from 274,429 records used in the calculation process.

Furthermore, we are assigning the same value of time to nearby airports. A traveler flying from JFK to DFW should have the same value of time as if it were flying from a LaGuardia Airport (LGA), which is 11 miles from JFK. Once we calculate the value of time at the airport level, the model then looks for airports within a 30 miles radius. The value of time is then weighted average using the number of records used to calculate the airport value of time. For example, JFK, LGA, and Newark Liberty International Airport (EWR) had a weighted average airport value of time of $165/hr. (calculated from 274,429 records), $68/hr. (calculated from 11,530 records), and $164/hr. (calculated from 124,142 records), respectively. These three airports are between 11 miles to 21 miles apart. When we used (Eq. 11) for these three airports, we calculated a weighted average value of time of $165/hr. Finally, we further calculate the value for each route since a traveler flying from ATL to DFW should have the same value of time as if he is flying from DFW to ATL.

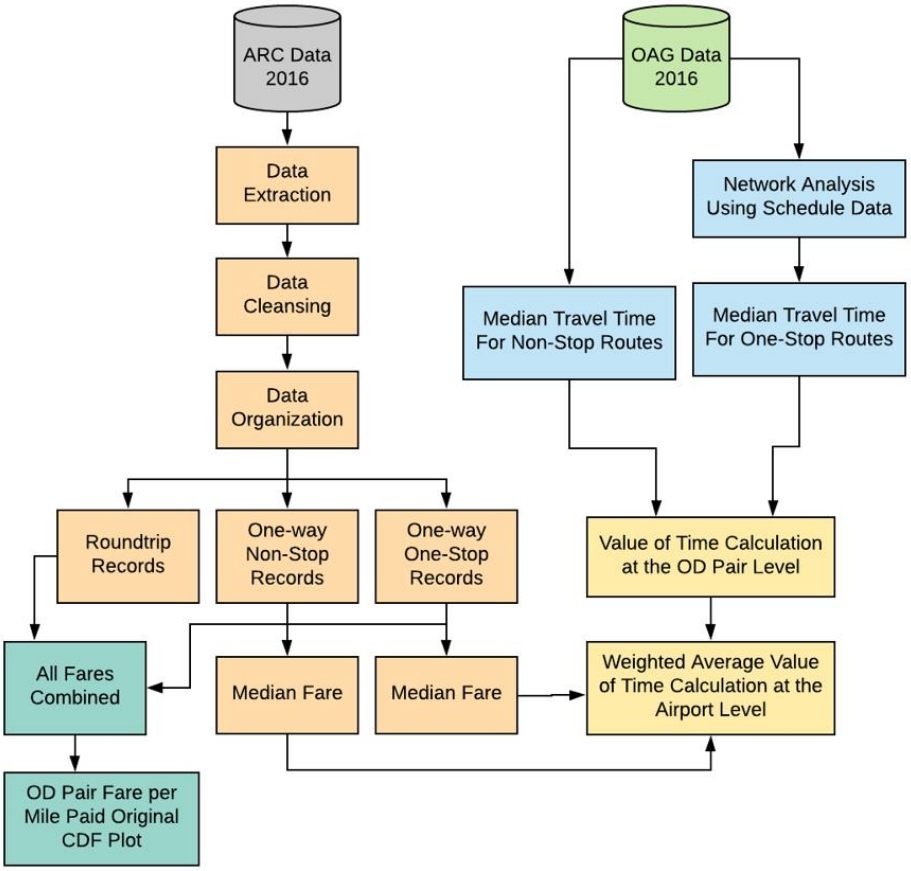


Figure 42: Flowchart of the Value of Time Analysis & Market Fare per Seat-Mile Analysis.

The process described above applies to all the one-way OD pairs with available data. For those one-way OD pairs with no available data, a market average is assigned. Figure 43 shows the VOT distribution for each of the three markets analyzed. The US, US-International, and International have a median VOT value of 103/hr., $203/hr., and $113/hr., respectively. According to the "2016 Revised Value of Time Guidance" from the U.S. Department of Transportation, the recommended hourly value of travel time savings (in 2015 U.S. $) for business category air travel is $75.80 [29]. This value translates to $88 in the 2021 U.S. dollar.

Table 1 summarizes how the value of time analysis evolved throughout the three studies and the main characteristics of each study.

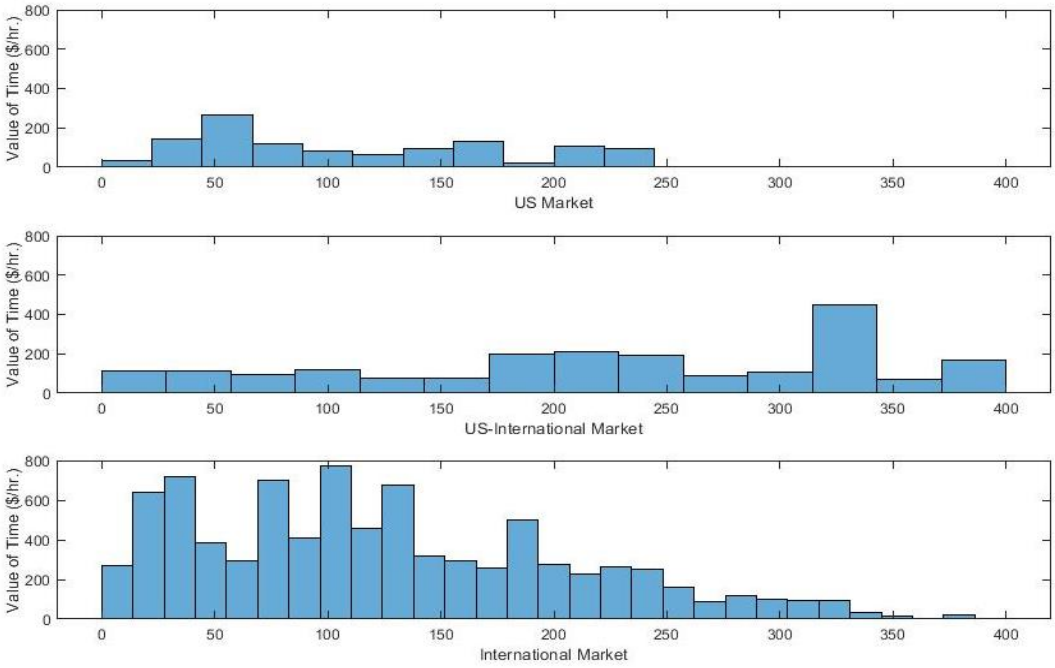


Figure 43: Value of Time Distribution by Market.

Table 12: Value of Time Analysis – Description of Main Parameters by Study.

Study 1 – ARC 2012	Study 2 – ARC 2012	Study 3 – ARC 2016
• 56 OD pairs analyzed • Over 100,000 Premium records • One hour stopover for one-stop records • Average fares and travel times • Single VOT value depending on route distance.	• Over 2 million records • Economy Premium records • Premium Records • Organize the data in three markets: US, US-Int., and International. • 1-hour layover for one-stop records within the US market. • 2-hours layover for one-stop records for the US-Int. and International markets • VOT value calculated for all available OD pairs. • A VOT value per market was calculated using the 50th percentile VOT value of each market.	• Forty-seven million records were analyzed. • Premium records • Organize the data in three markets: US, US-Int., and International. • Implemented a network analysis using OAG 2016 reported travel times to account for layover times in one-stop records. • We calculated weighted average VOT values. • Weighted average for airport VOT value. • Weighted average close by airport VOT values (within 30 miles radius) • Weighted average OD pair VOT value. • $400/hr. VOT limit

4.4 Worldwide Commercial Air Travel Demand Forecast Model

OAG 2016 data has been used as the baseline year for this analysis. The methodology from the ICAO Long-range Traffic Forecast, Passenger and Cargo, July 2016 has been adopted to generate a forecast model that predicts future passenger demand for years 2030 to 2040. The ICAO document contains equations developed to forecast air traffic growth within and between eleven world regions. The eleven regions include North America, Central America and Caribbean, South America, Europe, North Africa, Sub Saharan Africa, Middle East, Central, and Southwest Asia, North Asia, and Pacific Southeast Asia. The same forecast model from Study 1, described in Section 2.5, was used for Study 3.

4.5 Worldwide Supersonic Air Travel Demand Forecast Model

The model generates future projections of premium seats worldwide by using the ICAO forecast seats described in Section 3.5. The general process of estimating the LBSA demand was not modified when compared to Study 2. However, the significant changes made are associated with the integration of the model. Study 1 and Study 2 were focused on estimating the potential demand for specific aircraft designs provided by NASA. In this third modification, we created a tool (LBSAM) for NASA to optimize its design based on demand calculations. Instead of forecasting a low-boom supersonic aircraft with a certain speed and range, NASA can use LBSAM to run multiple cases by quickly changing different aircraft parameters.

Figure 30 shows the summarized process of going from the commercial aviation demand forecast to identifying potential routes with enough demand from the premium market (business and first-class) to support LBSA services. Beyond that point, the portion of passengers attracted to this type of service needs to be estimated. To estimate the LBSA demand, we follow the process described in Section 3.3 by combining the travel time analysis and value of time analysis (Section 4.5 and Section 4.6) and adjusting the original CDF plots (Section 4.6) to estimate the so-called willingness to pay percent. Figure 44 shows the complete process of the LBSAM application. Even though the general calculation of LBSA demand is the same as the one used before, the data used, the methodology, the assumption, among other parameters, are different.

The significant improvements made throughout the three studies allowed us or LBSAM users to run the model from a single Matlab main script using an Excel input file. From the input file, the user can change eight major aircraft parameters to the model. The LBSAM user can also change additional parameters from the Matlab main script. If the user wants to use a different value of time from the one calculated, the model has a bypass function. The user can specify a value of time at the market level, which would be assigned to all the one-way OD pairs within the market selected. The model can let the user decide if the airport runway length compatibility or the airport gate compatibility sections will be considered during the analysis for the airport compatibility module. The minimum runway length required is among the variable that can be changed from the main script. Currently, the takeoff runway length is set to 8,700 ft.

Additional parameters that can be changed from the Matlab main script are the start and end year of the demand forecast (currently 2030-2040), the value of time limit (now $400/hr.), minimum and the maximum distance of one-way OD pairs candidates for LBSA operations (currently 1,000 and 9,500 statute miles), minimum total demand (from commercial aviation forecast, Section 4.4) required from a one-way OD pair to be a candidate for LBSA operations (currently 100,000 seats), market share (now 50%), and the number of aircraft to be produced (used in the LCC module and currently set to 500 aircraft).

To enhance the LSBAM process, three modules were preprocessed to improve the model's computational time. These three modules are the travel time analysis (Section 4.2), value of time analysis (Section 4.3), and fare per seat-mile analysis (section 4.3). Suppose additional data is required from the travel time analysis. In that case, users can modify the following parameters: overland and overwater aircraft speed, overland and overwater aircraft range, minimum taxi-in and taxi-out times, delta climb additional time, and delta descent additional times.

For this Study 3, we provided NASA with guidance on how different LBSA parameters affect this type of service's potential demand. By analyzing a total of forty-eight cases. Table 13 shows the various parameters used in each case. The case matrix was applied to both LSBA (43-seat and 53-seat).

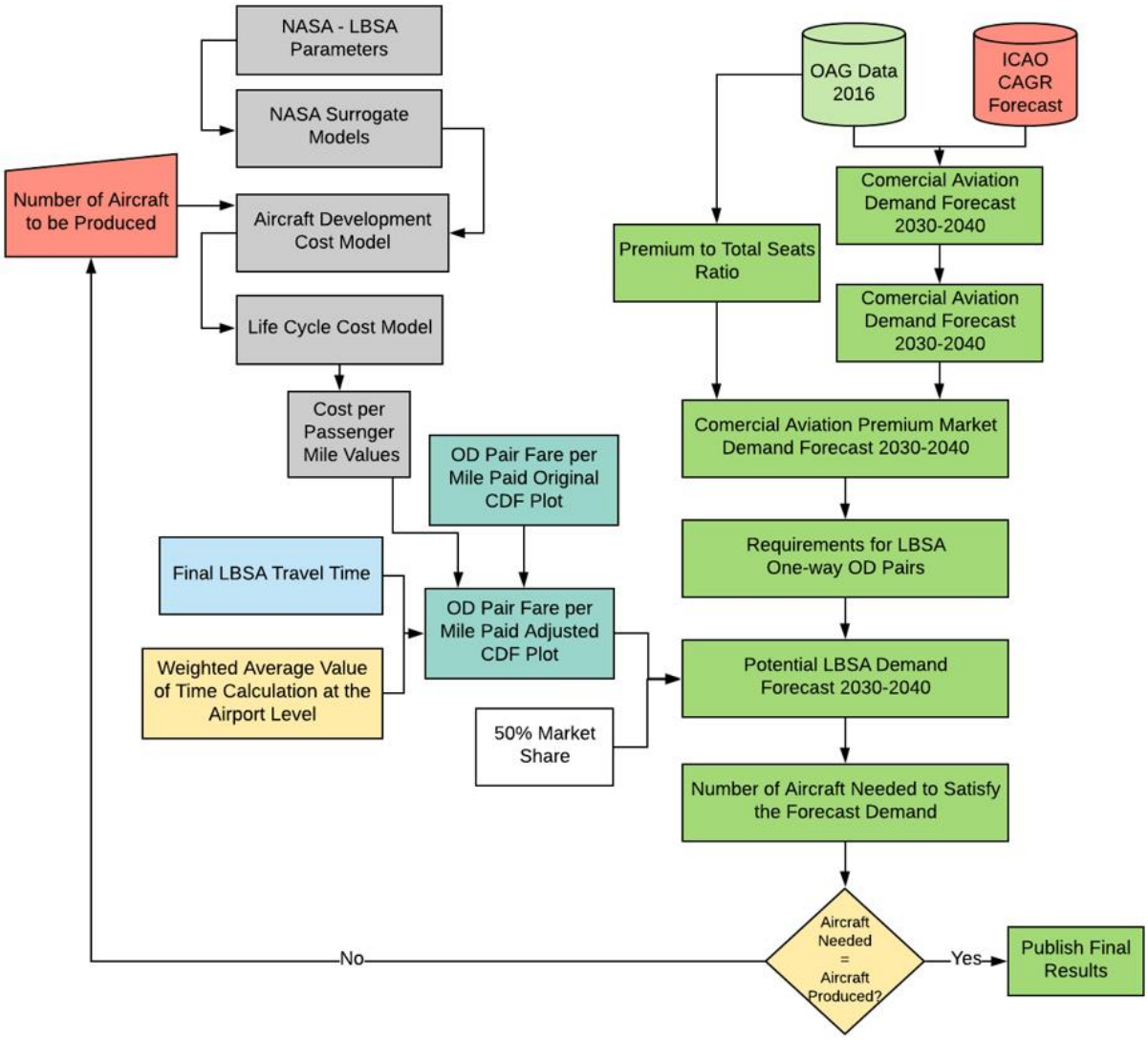


Figure 44: Flowchart of LBSAM Model.

Table 13: Low-Boom Supersonic Aircraft 24-Case Matrix.

Case	Parameters						
	Overland Range (nm.)	Overland Mach	Overland Fuel Scale Factor	Overwater Range (nm.)	Overwater Mach	Overwater Fuel Scale Factor	Aircraft Utilization (hrs.)
1	2,600	1.80	1.00	3,800	1.8	1.0	3500
2	2,600	1.80	0.96	3,800	1.8	1.0	3500
3	2,600	1.80	0.92	3,800	1.8	1.0	3500
4	2,800	1.80	1.00	3,800	1.8	1.0	3500
5	2,800	1.80	0.97	3,800	1.8	1.0	3500
6	2,800	1.80	0.94	3,800	1.8	1.0	3500
7	3,000	1.80	1.00	3,800	1.8	1.0	3500

Case	Parameters						
	Overland Range (nm.)	Overland Mach	Overland Fuel Scale Factor	Overwater Range (nm.)	Overwater Mach	Overwater Fuel Scale Factor	Aircraft Utilization (hrs.)
8	3,000	1.80	0.97	3,800	1.8	1.0	3500
9	3,200	1.80	1.00	3,800	1.8	1.0	3500
10	3,200	1.80	0.98	3,800	1.8	1.0	3500
11	2,800	1.70	1.00	3,800	1.8	1.0	3500
12	2,800	1.70	0.95	3,800	1.8	1.0	3500
13	2,800	1.70	0.90	3,800	1.8	1.0	3500
14	3,000	1.70	1.00	3,800	1.8	1.0	3500
15	3,000	1.70	0.95	3,800	1.8	1.0	3500
16	3,200	1.70	1.00	3,800	1.8	1.0	3500
17	3,200	1.70	0.98	3,800	1.8	1.0	3500
18	2,800	1.60	1.00	3,800	1.8	1.0	3500
19	2,800	1.60	0.94	3,800	1.8	1.0	3500
20	2,800	1.60	0.88	3,800	1.8	1.0	3500
21	3,000	1.60	1.00	3,800	1.8	1.0	3500
22	3,000	1.60	0.94	3,800	1.8	1.0	3500
23	3,200	1.60	1.00	3,800	1.8	1.0	3500
24	3,200	1.60	0.96	3,800	1.8	1.0	3500

4.6 Applying Linear Programing

In Section 2.6.1, (Eq. 8, we discussed calculating the number of aircraft needed to satisfy the demand. An alternate method to calculate the number of aircraft required. Using linear programming, we can obtain the desired number by identifying our decision variables and set the objective function subject to a set of constraints. For this example, we are using the demand results from case number 17. The analysis accounts for aircraft characteristics and performance such as aircraft cost, annual utilization, seating capacity, and travel times for each one-way origin-destination pair.

For this example, we used the linear programming solver from Matlab, which is part of the optimization toolbox. The solver finds the minimum value of the objective function specified by

$$\min f(x) such\ as\ that \begin{cases} A * x \leq b \\ Aeq * x = beq \\ lb \leq x \leq ub \end{cases} \quad \textbf{(Eq. 12)}$$

Where *f, x, b, beq, lb,* and *ub* are vectors, *A* and *Aeq* are matrices.

(f) are our coefficient vector, specified as a real vector. The coefficient vector represents the objective function f'*x. These are the coefficient of the decision variable in our objective function.

(A) are linear inequality constraints, specified as a real matrix. A is an M-by-N, where M is the number of inequalities, and N is the number of variables.

(Aeq) are linear equality constraints, specified as a real matrix. Aeq is an M-by-N, where M is the number of equalities, and N is the number of variables.

(b) are linear inequality constraints, specified as a real vector. (b) is an M-element vector related to the A matrix.

(beq) are linear equality constraints, specified as a real vector. (beq) is an M-element vector related to the A matrix.

(lb) are lower bound specified as a real vector.

(ub) are upper bound specified as a real vector.

The following format is used in a linear programing solver in Matlab:

[*x y*] = linprog(*f,A,b,Aeq,beq,lb,ub*)

Where x are the values of the decision variables and y is the value of the objective function.

Matlab implements a dual-simplex method in the solver programmed in the optimization toolbox.

To find the number of supersonic aircraft needed in the worldwide fleet, we define the problem as a mathematical programming minimization problem. The objective function is to minimize the number of flights and aircraft required to satisfy the demand while minimizing the purchase of the fleet. The 43-seats and 52-seat LBSA are assumed to cost 145.6 million and 169.5 million, respectively.

$$Minimize\ y = \sum_{k=1}^{m} \sum_{OD=1}^{n} S_k X_{OD} + \sum_{k=1}^{m} C_k A_k \qquad \textbf{(Eq. 13)}$$

Subject to:

$$\sum_{k=1}\sum_{OD=1}^{n} X_{ODk} + \sum_{k=2}\sum_{OD=1}^{n} X_{OkK} \geq \sum_{OD=1}^{n} D_{OD} \quad \textbf{(Eq. 14)}$$

$$\sum_{k=1}\sum_{OD=1}^{n} X_{ODk} + \sum_{k=2}\sum_{OD=1}^{n} X_{ODk} \geq 1 \quad \textbf{(Eq. 15)}$$

$$\sum_{k=1}\sum_{OD=1}^{n} T_{OD}X_{ODk} \leq \sum_{k=1} U_k A_k \quad \textbf{(Eq. 16)}$$

$$\sum_{k=2}\sum_{OD=1}^{n} T_{OD}X_{ODk} \leq \sum_{k=2} U_k A_k \quad \textbf{(Eq. 17)}$$

$$Non-negativity: X_{OD}, A_K \geq 0$$

Where,

k = type of aircraft. k=1, 43-seat LBSA; k=2, 52-seat LBSA

n = number of OD pairs, 166.

m = number of aircraft, 2.

OD = origin-destination airport pair. Includes the US and the US-International Markets.

S_k = Seating capacity for aircraft type k.

C_k = Cost of aircraft k.

X_{ODk} = Number of flights assigned to aircraft type K for the specified OD pair. (decision variable)

A_k = Number of aircraft of type k. (decision variable)

D_{OD} = Demand for OD pair.

T_{OD} = Travel time for OD pair.

U_k = Number of aircraft utilization hours per year for aircraft type k.

The equations were modified to be consistent with the format required by Matlab linear programming solver. The questions were modified as follows.

$$\sum_{k=1}\sum_{OD=1}^{n} -X_{ODk} - \sum_{k=2}\sum_{OD=1}^{n} -X_{OkK} \leq \sum_{OD=1}^{n} -D_{OD} \quad \textbf{(Eq. 18.1)}$$

$$\sum_{k=1}\sum_{OD=1}^{n} -X_{ODk} - \sum_{k=2}\sum_{OD=1}^{n} -X_{ODk} \leq -1 \quad \textbf{(Eq. 19.1)}$$

$$\sum_{k=1}\sum_{OD=1}^{n} T_{OD}X_{ODk} - \sum_{k=1} U_k A_k \leq 0 \quad \textbf{(Eq. 20.1)}$$

$$\sum_{k=2}\sum_{OD=1}^{n} T_{OD}X_{ODk} - \sum_{k=2} U_k A_k \leq 0 \quad \textbf{(Eq. 21.1)}$$

Table 14 presents the list of one hundred and sixty-six one-way origin-destination pairs specified by airport codes. The total number of constraints generated for the analysis is three hundred and thirty-four. A parametric analysis was completed by studying the effect of aircraft utilization in the results. The combination of the aircraft utilization for each scenario is presented in Table 15

Table 14: List of one-way origin-destination pairs. US and US-International Markets.

'ATL_LAX'	'AMS_ATL'	'CCS_MIA'	'EWR_FRA'	'IAD_CDG'	'JFK_ZRH'	'LHR_ORD'	'MXP_JFK'	'TLV_JFK'
'DTW_LAX'	'AMS_BOS'	'CDG_ATL'	'EWR_LHR'	'IAD_FRA'	'LAX_CDG'	'LHR_PHL'	'NRT_DFW'	'YVR_JFK'
'EWR_LAX'	'AMS_DTW'	'CDG_BOS'	'EWR_MUC'	'IAD_LHR'	'LAX_HKG'	'LHR_SEA'	'NRT_HNL'	'YVR_LAX'
'EWR_SFO'	'AMS_EWR'	'CDG_DTW'	'EWR_ZRH'	'IAH_LHR'	'LAX_LHR'	'LHR_SFO'	'ORD_FRA'	'ZRH_BOS'
'IAD_SFO'	'AMS_IAD'	'CDG_EWR'	'FRA_BOS'	'JFK_AMS'	'LAX_MEX'	'MAD_JFK'	'ORD_LHR'	'ZRH_EWR'
'JFK_LAX'	'AMS_JFK'	'CDG_IAD'	'FRA_EWR'	'JFK_BCN'	'LAX_PEK'	'MBJ_ATL'	'PEK_JFK'	'ZRH_JFK'
'JFK_SFO'	'AMS_MSP'	'CDG_JFK'	'FRA_IAD'	'JFK_CDG'	'LAX_PPT'	'MEX_JFK'	'PEK_LAX'	
'LAX_ATL'	'ATL_AMS'	'CDG_LAX'	'FRA_JFK'	'JFK_DXB'	'LAX_YVR'	'MEX_LAX'	'PEK_SFO'	
'LAX_DTW'	'ATL_CDG'	'CDG_SFO'	'FRA_ORD'	'JFK_FRA'	'LHR_ATL'	'MEX_MIA'	'PHL_LHR'	
'LAX_EWR'	'ATL_GRU'	'CLT_LHR'	'FRA_SFO'	'JFK_GRU'	'LHR_BOS'	'MIA_CCS'	'PPT_LAX'	
'LAX_JFK'	'ATL_LHR'	'DFW_GRU'	'GRU_ATL'	'JFK_GVA'	'LHR_CLT'	'MIA_GRU'	'PTY_MIA'	
'LAX_MIA'	'ATL_MBJ'	'DFW_LHR'	'GRU_DFW'	'JFK_HKG'	'LHR_DFW'	'MIA_LHR'	'SCL_MIA'	
'JFK_MIA'	'ATL_MTY'	'DFW_NRT'	'GRU_JFK'	'JFK_LHR'	'LHR_DTW'	'MIA_MEX'	'SEA_LHR'	
'JFK_MSP'	'BCN_JFK'	'DTW_AMS'	'GRU_MIA'	'JFK_MAD'	'LHR_EWR'	'MIA_PTY'	'SFO_CDG'	
'MIA_LAX'	'BOS_AMS'	'DTW_CDG'	'GVA_JFK'	'JFK_MEX'	'LHR_IAD'	'MIA_SCL'	'SFO_FRA'	
'MIA_JFK'	'BOS_CDG'	'DTW_LHR'	'HKG_JFK'	'JFK_MXP'	'LHR_IAH'	'MIA_SJO'	'SFO_HKG'	
'MSP_JFK'	'BOS_FRA'	'DXB_JFK'	'HKG_LAX'	'JFK_PEK'	'LHR_JFK'	'MSP_AMS'	'SFO_LHR'	
'SFO_EWR'	'BOS_LHR'	'EWR_AMS'	'HKG_SFO'	'JFK_SVO'	'LHR_LAX'	'MSP_LHR'	'SFO_PEK'	
'SFO_IAD'	'BOS_ZRH'	'EWR_BRU'	'HNL_NRT'	'JFK_TLV'	'LHR_MIA'	'MTY_ATL'	'SJO_MIA'	
'SFO_JFK'	'BRU_EWR'	'EWR_CDG'	'IAD_AMS'	'JFK_YVR'	'LHR_MSP'	'MUC_EWR'	'SVO_JFK'	

Table 15: List of Scenarios – Linear Programming Example.

Scenario	Aircraft Utilization, U_k (hours per year)	
	43-Seat LBSA	52-Seats LBSA
1	3,000	3,000
2	3,500	2,500
3	2,500	3,500

4.7 Result

The LSBA model described in Chapter 4 was applied to a 24-case matrix. The case matrix is used for both LBSA aircraft with 43 and 52-seats. Table 13 shows the parameters used in each of the cases. The results combined the LBSA projections for the US, US-International, and International only markets (worldwide). The LBSA aircraft market demand results are for the year 2040. The cases presented in Table 13 assume aircraft utilization of 3,500 hours per year.

4.7.1 52-Seat Low-Boom Supersonic Aircraft Worldwide Projections

Table 16 presents the results for the scenarios shown in Table 13.

- Case 17 shows the highest seat demand, with 33.4 million seats worldwide.
- Case 1 shows the lowest seat demand, with 26.8 million seats worldwide.
- Case 13 shows the highest number of aircraft needed, with 772 aircraft worldwide.
- Case 1 shows the lowest number of aircraft needed, with 635 aircraft worldwide.
- Case 13 shows 1,032 one-way OD pairs for the highest case in where LBSA can operate.
- Case 1 shows 824 one-way OD pairs for the lowest case in where LBSA can operate.

The effect of overland range in LBSA demand at constant overland speed is:

- At Mach 1.8 overland, projections increase 3.5% for every 200 nm increment in the overland range.
- At Mach 1.7 overland, projections increase 2.1% for every 200 nm increment in the overland range.
- At Mach 1.6 overland, projections increase 1.3% for every 200 nm increment in the overland range.

The effect of overland speed in LBSA demand at a constant overland range is:

- Projections increase by 2.2% when increasing the overland Mach from 1.6 to 1.7 at an overland range of 2,800 nm.
- Projections decrease by 9.9.% when increasing the overland Mach from 1.7 to 1.8 at an overland range of 2,800 nm.
- Projections increase by 2.2% when increasing the overland Mach from 1.6 to 1.7 at an overland range of 3,000 nm.
- Projections decrease by 10.1% when increasing the overland Mach from 1.7 to 1.8 at an overland range of 3,000 nm.
- Projections increase by 3.7% when increasing the overland Mach from 1.6 to 1.7 at an overland range of 3,200 nm.
- Projections decrease by 7.8% when increasing the overland Mach from 1.7 to 1.8 at an overland range of 3,200 nm.
- Case 17 generates 21.8% more demand than Case 1 based on the following differences:
    - An overland range difference of 600 nm.
    - A $0.01/statute mile difference, as shown in Figure 45 and Figure 46
    - An overland speed difference of Mach 0.1
    - An overland fuel scale factor difference of 2%

4.7.2 43-Seat Low-Boom Supersonic Aircraft Worldwide Predictions

Table 16 presents the results for the scenarios shown in Table 13.

- Case 17 shows the highest seat demand, with 29.8 million seats worldwide.
- Case 1 shows the lowest seat demand, with 23.9 million seats worldwide.
- Case 13 shows the highest number of aircraft needed, with 830 aircraft worldwide.
- Case 1 shows the lowest number of aircraft needed, with 696 aircraft worldwide.
- Case 13 shows 1,057 one-way OD pairs for the highest case in where LBSA can operate.
- Case 1 shows 882 one-way OD pairs for the lowest case in where LBSA can operate.

The effect of overland range in LBSA aircraft worldwide seat demand (at constant overland speed) are:

- At Mach 1.8 overland, projections increase 4.1% for every 200 nm increment in the overland range.
- At Mach 1.7 overland, projections increase 2.8% for every 200 nm increment in the overland range.
- At Mach 1.6 overland, projections increase 1.3% for every 200 nm increment in the overland range.

The effect of overland speed in LBSA aircraft worldwide seat demand (at a constant overland range) are:

- Projections increase by 7.0% when increasing the overland Mach from 1.6 to 1.7 at an overland range of 2,800 nm.
- Projections decrease by 7.3% when increasing the overland Mach from 1.7 to 1.8 at an overland range of 2,800 nm.
- Projections increase by 9.5% when increasing the overland Mach from 1.6 to 1.7 at an overland range of 3,000 nm.
- Projections decrease by 8.1% when increasing the overland Mach from 1.7 to 1.8 at an overland range of 3,000 nm.
- Projections increase by 10.0% when increasing the overland Mach from 1.6 to 1.7 at an overland range of 3,200 nm.
- Projections decrease by 4.9% when increasing the overland Mach from 1.7 to 1.8 at an overland range of 3,200 nm.
- Case 17 generates 20.2% more demand than Case 1 based on the following differences:
    - An overland range difference of 600 nm.
    - A $0.01/statute mile difference, as shown in Figure 47 and Figure 48.
    - An overland speed difference of Mach 0.1
    - An overland fuel scale factor difference of 2%

Table 16: Case Matrix Results for LBSA 43-Seat and LBSA 53-Seat. Projections for the Year 2040.

Parameters				Aircraft		Seats (Million)		OD's	
Case	Overland Range (nm.)	Overland Mach	Overland Fuel Scale Factor	43 Seats	52 Seats	43 Seats	52 Seats	43 Seats	52 Seats
1	2,600	1.80	1.00	696	635	23.9	26.8	882	824
2	2,600	1.80	0.96	735	670	25.1	28.1	910	858
3	2,600	1.80	0.92	778	701	26.5	29.3	954	888
4	2,800	1.80	1.00	721	656	25.0	28.0	919	861
5	2,800	1.80	0.97	753	681	26.0	28.9	943	888
6	2,800	1.80	0.94	786	702	27.0	29.8	979	910
7	3,000	1.80	1.00	729	651	26.1	28.8	949	881
8	3,000	1.80	0.97	764	674	27.1	29.6	977	904
9	3,200	1.80	1.00	767	678	27.6	30.3	995	931
10	3,200	1.80	0.98	790	696	28.3	30.9	1,017	950
11	2,800	1.70	1.00	765	714	27.3	31.1	997	974
12	2,800	1.70	0.95	795	737	28.2	31.9	1,019	992
13	2,800	1.70	0.90	830	772	29.3	33.2	1,057	1,032
14	3,000	1.70	1.00	792	723	28.5	32.0	1,029	984
15	3,000	1.70	0.95	818	745	29.3	32.8	1,048	999

Parameters				Aircraft		Seats (Million)		OD's	
Case	Overland Range (nm.)	Overland Mach	Overland Fuel Scale Factor	43 Seats	52 Seats	43 Seats	52 Seats	43 Seats	52 Seats
16	3,200	1.70	1.00	810	746	29.4	33.0	1,029	992
17	3,200	1.70	0.98	823	757	29.8	33.4	1,037	1,001
18	2,800	1.60	1.00	714	706	25.3	30.5	941	942
19	2,800	1.60	0.94	741	728	26.2	31.4	963	960
20	2,800	1.60	0.88	770	752	27.2	32.3	985	980
21	3,000	1.60	1.00	721	714	25.8	31.2	951	958
22	3,000	1.60	0.94	751	737	26.8	32.1	977	978
23	3,200	1.60	1.00	731	719	26.2	31.5	961	968
24	3,200	1.60	0.96	750	734	26.8	32.1	977	980

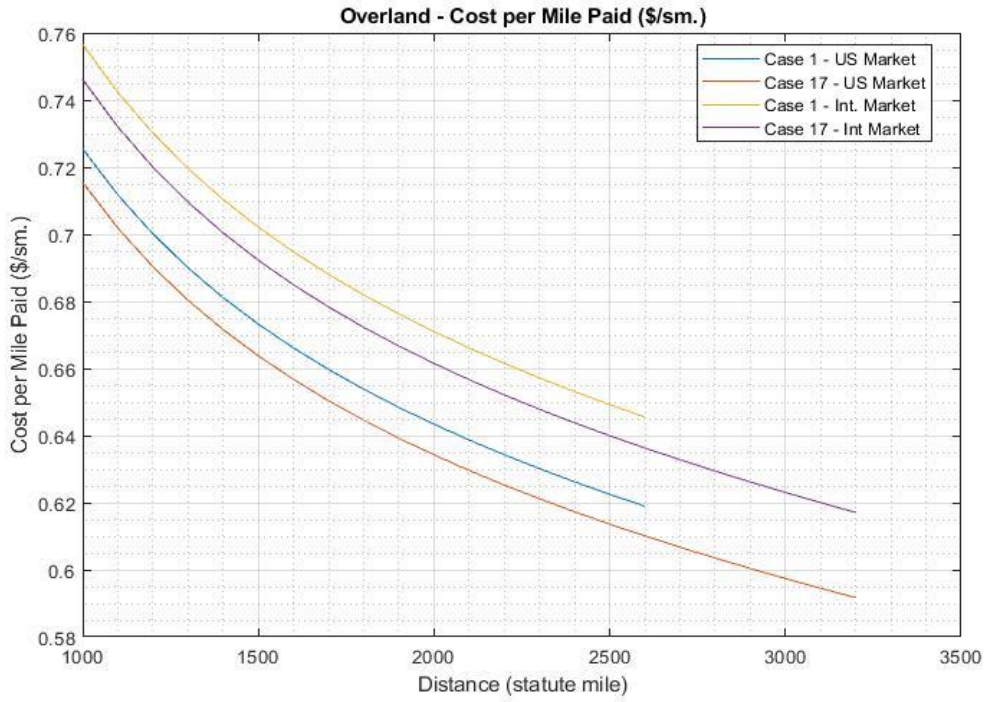


Figure 45: 52-Seat LBSA Overland Cost per Mile Paid ($/sm.) for Case 1 and Case 17 in Table 13.

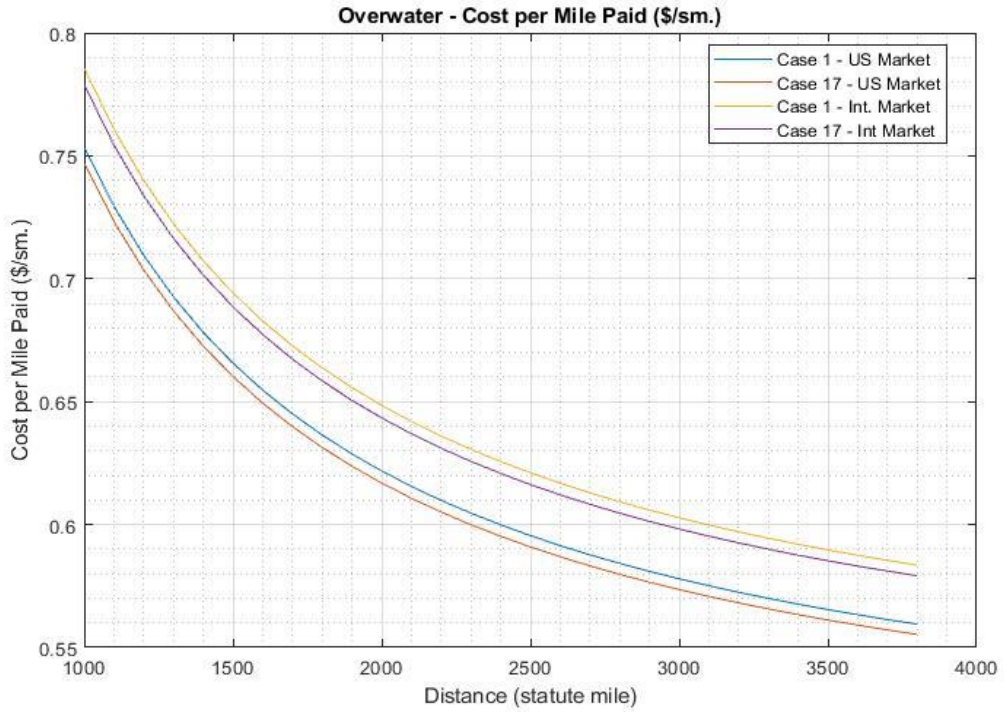


Figure 46: 52-Seats LBSA: Overwater Cost per Mile Paid ($/sm.) for Cases 1 and 17 in Table 13.

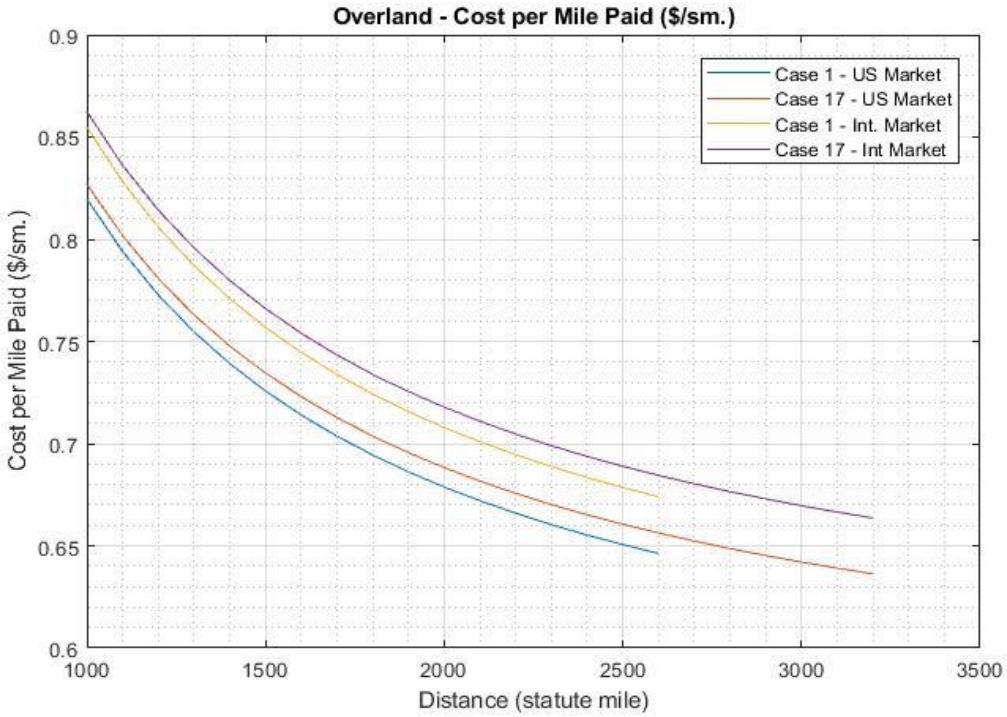


Figure 47: 43-Seats LBSA: Example of Overland Cost per Mile Paid ($/sm.) for Cases 1 and 17 in Table 13.

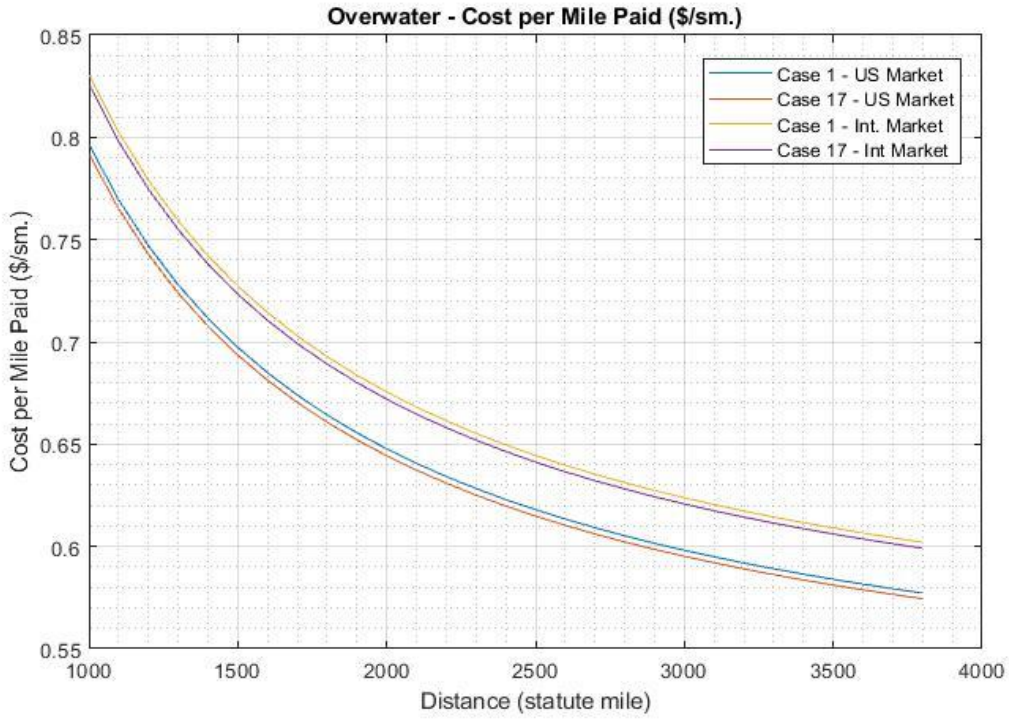


Figure 48: 43-Seats LBSA: Overwater Cost per Mile Paid ($/sm.) for Cases 1 and 17 in Table 13.

4.7.3 Linear Programming Parametric Analysis

This section presents the results of the linear programming problem. As described in Section 4.6, the objective function was set as a minimization problem. Table 17 shows the results for each of the three scenarios that were part of the parametric analysis.

Table 17: Results From Parametric Analysis – Linear Programming.

Scenario	43-Seats LBSA		52-LBSA	
	Flights	Aircraft	Flights	Aircraft
1	0	0	96,687	158
2	116,920	162	0	0
3	0	0	96,687	134

The conclusion indicates that both the cost of the aircraft and the aircraft's annual utilization directly impact the analysis results. If the annual utilization remains constant at three thousand hours per year, the optimal solution is purchasing one hundred and fifty-eight aircraft of type 52-Seat LBSA. In scenario two, assuming an aircraft utilization of three thousand and five hundred for the 43-Seat LBSA and utilization of two thousand and five hundred for the 52-Seat LBSA, the optimal solution is by purchasing one hundred and sixty-two aircraft of type 42-Seat LBSA. The

optimal solution is obtained in scenario three by purchasing one hundred and thirty-four aircraft of type 52-Seat LBSA.

Chapter 5

5 Summary of Contributions and Recommendation for Future Research

The Low Boom Supersonic Aircraft Model is a collaborative effort between Virginia Tech and the National Aeronautics and Space Administration (NASA). This chapter summarizes the contributions added to the field of Air Transportation systems, specifically within the Supersonic Aircraft Concept development research area. Additionally, Chapter 5 presents recommendations for future research work.

5.1 Summary of Contributions

The LBSAM is a tool that improves the knowledge in the supersonic aircraft concepts research area. This dissertation's main contribution to the Air Transportation field is the extensive analysis performed towards identifying potential commercial supersonic aircraft demand. The work presented in this dissertation explains in detail what could be the potential demand, why it would be generated, and how it could be identified the potential demand for supersonic transport services. This dissertation work filled the gap found in the literature review. The LBSAM model is an integrated tool that produces commercial supersonic demand forecast by considering, among several variables, air travelers' behavior by combining two main factors. The first factor is the value of time, and the second factor is the travel time benefit that supersonic flights provide. By combining these two factors, the LBSAM model estimates what percentage of travelers would be willing to pay for the high fares expected on supersonic flights. Travelers who currently are part of the premium subsonic market, business class and first class travelers.

At the conclusion of this analysis, it is possible to envision future supersonic transport services provided with improved aircraft designs, lower cost, and a revision of current government regulations. Results from the multiple scenarios analyzed with the LBSAM model suggest that there is potential demand in various markets. There will be markets that will attract more demand than others, depending on the final aircraft performances and characteristics and route characteristics.

However, there is still a long road before seeing commercial supersonic flights integrated into the air transportation network. Companies need to keep working on improving aircraft design,

performance, and characteristics. The negative environmental impact involved in supersonic aircraft needs to be improved. In addition, global regulations on supersonic flights need to be revised. Governments and aircraft manufacture need to work together to find a middle point where commercial supersonic flights can be a reality.

Companies working towards bringing back the commercial supersonic flight service cannot lose sight of who the target is; travelers. In the end, air travelers will be the one who decides whether supersonic flights becomes a reality. Faster speed resulting in shorter travel times cannot be the only benefit of traveling in a supersonic aircraft. This type of service needs to be attractive to the passengers spending almost six hours, gate-to-gate travel time on a 2,500 miles route, and the ones spending over ten or fifteen hours on an ultra-long route.

The current longest commercial route from Newark Liberty International Airport (EWR) to Singapore Changi Airport (SIN) lasts around 18 hours, gate-to-gate total travel time. Even if supersonic flights can reduce the travel time by half, passengers will still experience over eight hours in the air. Premium passengers are used to luxury amenities and services, including flatbed seats, suites, showers, and even personal butler. It is arguably questionable that a significant percentage of the premium travelers could be willing to change the luxury experience for faster travel time on a regular limited reclinable business seat offered on domestic routes. Having more spacious seating options on supersonic aircraft would reduce the total seating capacity of the aircraft. A lower number of seats on an aircraft would result in passengers experiencing higher fares.

Additionally, to reduce the loudness of the sonic boom, the aerodynamic of the aircraft call for narrow and elongated aircraft fuselage. Increasing the length of a supersonic aircraft would allow a higher number of seats, perhaps reducing the airfares. The narrow and long fuselage also allows faster speed while reducing the loudness of the sonic boom. However, increasing the length of the aircraft brings a limitation to commercial supersonic flight service. Supersonic aircraft design with a length similar to or above an Airbus 380, Boeing 777X, or alike would be limited to a small number of airports worldwide. Everything is connected, and identifying potential demand for this type of service is a highly complex problem. A problem that I am confident will be resolve in the coming years.

5.2 Recommendations for Future Research

In this section, recommendations are presented for future research. As mentioned before, there is a long road before we can see supersonic flights being integrated into the network. As a highly complex research topic, there will always be room for improvement.

5.2.1 Airport Compatibility

Improve the airport compatibility module of the LBSAM. Currently, it relies on identifying airport in which wide-body aircraft operates. It is assumed that if a wide-body aircraft can operate at a designated airport, it could be possible for a supersonic aircraft to operate; from a runway length requirement perspective. Additional effort should include an emphasis on identifying gate compatibility at the airport. Also, consider airport curfews and desired arrival and departure times in the LBSA analysis. Many airports impose nighttime curfews that could affect the demand for LBSA services. LBSA aircraft utilization in an airline network may be influenced by desired departure and arrival times to airports.

5.2.2 Impact of Supersonic Operations on the Subsonic Market

It is assumed that commercial supersonic demand will be generated from the existing subsonic commercial market. Shifting passengers of premium seats from subsonic aircraft to supersonic commercial aircraft would affect coach class tickets. The potential effect that the supersonic operation could have over the subsonic market should not be underestimated. The impact should be analyzed in a follow-up study. A quick analysis of some premium routes shows that the airline revenue attributed to premium seats ranges from 25-40% per flight. If a fraction of premium passengers shifts to supersonic services, airlines may have to increase their coach fares by 10-15% in such routes. Assuming a demand elasticity with a fare of -0.8 (for long-distance flights) could reduce passenger demand in coach services by 7-12%. This has real implications for the airline business.

5.2.3 Study Air Traveler's Behavior

Investigate the relationship between trip length and sub-mode choice for various LBSA cabin configurations. The current study was limited to standard seating configurations for all OD routes flown by LBSA aircraft. Longer routes may require "sleeper" seats that will reduce the number of

passengers to be carried in a given flight. This can have a substantial effect on ticket prices for such routes. Companies' marketing supersonic aircraft show at least two cabin configurations depending on the length of the flights. Additional investigation in this area would help improve one of the high uncertainty variables in the LMSAM model, market share.

5.2.4 Worldwide Commercial Supersonic Network

Investigate the effect of a worldwide network comprised of different airlines in the LBSA market demand. This addresses the need to investigate significant inefficiencies in aircraft utilization when airlines have limited LBSA aircraft numbers to fly a large number of OD pairs in their network. Additional investigation in this area would help improve one of the high uncertainty variables in the LMSAM model, annual aircraft utilization. We analyzed the effect of the yearly aircraft utilization using values that ranged from 2,500 to 3,500 hours per year. However, these values could be potentially optimistic compared to the 1,800 hours per year that the Concorde flew.

5.2.5 Additional Recommendations

Continue to optimize the aircraft designs being analyzed with the LBSAM model. Compare the advantages and disadvantages of integrating an aircraft fleet mix into the network.

Evaluate the design aspects of LBSA design in more detail to understand how maintenance and logistic support actions may play a role in the worldwide demand for LBSA aircraft services. Logistic support for dedicated LBSA flights may influence airline choices on where to deploy the vehicles. The demand estimated in this analysis does not consider the normal down selection of airlines' markets when deploying expensive assets.

Block fuel calculation for supersonic aircraft is not covered in this dissertation. However, it is part of the LBSAM model, and it is recommended to improve the modules that estimate block fuel as a function of distance for both overland and overwater missions. NASA provided surrogate models that provide accurate information for the vehicle design range. However, the method used to estimate block fuel for distances less than the design range seems to underestimate the actual block fuel for short to medium-distance flights. This reduces the cost of short and intermediate flights and increases LBSA demand in those routes. An alternative procedure to estimate block fuel over distance is to synchronize the NASA surrogate model that produces the aircraft design parameters

(thrust, OEW, ZFW, and TOGW) with the second surrogate model that produces block fuel as a function of distance.

References

[1] E. Freire Burgos, N. Hinze, S. Jungmin and A. Trani, "Aviation Global Demand Forecast Model Development and ISAAC Studies: Supersonic Aircraft Market Study," Air Transportation System Lab, Virginia Tech, Blacksburg, 2018.

[2] E. Freire Burgos, N. Hinze and A. Trani, "Aviation Global Demand Forecast Model Development and ISAAC Studies: Supersonic Aircraft Market Study," Air Transportation System Lab, Virginia Tech, Blacksburg, 2019.

[3] E. Freire Burgos, N. Hinze and A. Trani, "Aviation Global Demand Forecast Model Development and ISAAC Studies: Supersonic Aircraft Market Study," Air Transportation System Lab, Virginia Tech, Blacksburg, 2020.

[4] M. Rouse, "WhatIs," TechTarget, 01 09 2005. [Online]. Available: https://whatis.techtarget.com/definition/speed-of-sound. [Accessed 11 12 2020].

[5] E. Gregersen, "Britannica," Encyclopaedia Britanica, Inc., 21 07 2015. [Online]. Available: https://www.britannica.com/technology/supersonic-flight. [Accessed 11 12 2020].

[6] W. Hosch, "Britannica," Encyclopaedia Britannica, Inc., 13 11 2008. [Online]. Available: https://www.britannica.com/technology/Tupolev-Tu-144. [Accessed 15 11 2020].

[7] A. Tikkanen, "Britannica," Encyclopaedia Britannica, Inc., 26 9 2017. [Online]. Available: https://www.britannica.com/technology/Concorde. [Accessed 15 11 2020].

[8] C. Weit, J. Wen, A. Anand, M. Mayakonda, T. Zaidi and D. Mavris, "A Methodology for Supersonic Commercial Market," in *SMARTech*, Virtual Event, 2020.

[9] B. Liebhardta, K. Lütjens, A. Uenoc and I. Hiroaki, "JAXA's S4 Supersonic Low-Boom Airliner – A Collaborative Study on Aircraft Design, Sonic Boom Simulation, and Market Prospects," in *AIAA Aviation Forum*, Virtual Event, 2020.

[10] S. Jain, K. E. Ogunsina, H. Chao, W. A. Crossley and D. A. DeLaurentis, "Predicting Routes for, Number of Operations of, and Fleet-level Impacts of Future Commercial Supersonic Aircraft on Routes Touching the United States," *AIAA Aviation Forum*, 2020.

[11] M. Hassan, H. Pfaender and D. Mavris, "Design Tools for Conceptual Analysis of Future Commercial Supersonic Aircraft," *AIAA Aviation Forum*, 2020.

[12] B. Liebhardt, K. Luetjens and V. Gollnick, "Estimation of the Market Potential for Supersonic Airliners via Analysis of the Global Premium Ticket Market," *AIAA Aviation Technology*, 2011.

[13] B. Liebhardt, K. Lutjens, R. R. Tracy and A. O. Haas, "Exploring the Prospect of Small Supersonic Airliners - A Business Case Study Based on Aerion AS2 Jet," *AIAA Aviation Forum*, 2017.

[14] D. L. Huff, B. S. Henderson, J. J. Berton and J. A. Seidel, "Perceived Noise Analysis for Offset Jets Applied to Commercial Supersonic Aircraft," *AIAA SciTech Forum*, 2016.

[15] R. Welge, C. Nelson and J. Bonet, "Supersonic Vehicle Systems for the 2020 to 2035 Timeframe," *AIAA Applied Aerodynamics*, 2010.

[16] B. Liebhardt, K. Lutjens, R. Tracy and A. Hass, "Exploring the Prospect of Small Supersonic Airliners - A Case Study Based on the Aerion AS2 Jet," in *AIAA Aviation Forum* , Denver, 2017.

[17] R. Welge, C. Nelson and J. Bonet, "Supersonic Vehicle System for the 2020 to 2035 Timeframe," in *AIAA Applied Aerodynamics Conference*, Chicago, 2010.

[18] J. Bogaisky, "Forbes," Forbes Media, 19 01 2019. [Online]. Available: https://www.forbes.com/sites/jeremybogaisky/2019/01/04/boom-raises-100m-to-develop-supersonic-jet-its-going-to-need-a-lot-more/?sh=540b9f6f77e3. [Accessed 25 07 2020].

[19] J. Wynbrandt, "AIN Online," The Convention News Company, Inc., 15 06 2019. [Online]. Available: https://www.ainonline.com/aviation-news/aerospace/2019-06-15/boom-unveils-more-details-supersonic-airliner. [Accessed 16 07 2020].

[20] Y. Gibbs, "NASA," NASA, 15 08 2017. [Online]. Available: https://www.nasa.gov/centers/armstrong/news/FactSheets/FS-016-DFRC.html. [Accessed 16 17 2020].

[21] FAA, "4 CFR §91.817 - Civil Aircraft Sonic Boom," DOT, 1989.

[22] *Official Airline Guide*, 2016.

[23] "SeatGuru," TripAdvisor, 01 01 2016. [Online]. Available: https://www.seatguru.com/airlines/China_Airlines/China_Airlines_Boeing_777-300ER.php. [Accessed 03 04 2019].

[24] T. Spencer, A. Trani and N. Hinze, "Flight Planner for the Global Oceanic Model," Air Transportation System Laboratory, Virginia Tech, Blacksburg, 2017.

[25] R. Padalkar, "Global Commercial Aircraft Fuel Burn and Emissions Forecast: 2016 to 2040," Virginia Tech, Blacksburg, 2017.

[26] International Civil Aviation Organization, "ICAO Long-Term Traffic Forecasts," 2016.

[27] Boeing, "Commercial Market Outlook 2019-2038," Boeing, 2019.

[28] Airbus, "Global Market Forecast," Airbus, 2019.

[29] US DOT, "US DOT," 2015. [Online]. Available: https://www.transportation.gov/sites/dot.gov/files/docs/2016%20Revised%20Value%20of%20Travel%20Time%20Guidance.pdf. [Accessed 2021].

[30] Federal Aviation Administration, "Advisory Circular 150/5300-13A," U.S. DOT, 2014.

[31] P. Hart, N. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEE Transportation on Systems Science and Cybernetics* , vol. 4, pp. 100-107, 1968.

[32] L. Nicolai and G. Carichner, Fundamentals of Aircraft and Airship Design, American Institute of Aeronautics and Astronautics, 2010.

[33] Business & Commercial Aviation, "Purchase Planning Handbook," BCA, 2020.

APPENDIX A – STUDY 3: ADDITIONAL RESULTS

Additional results from Study 3 are presented in this section.

The worldwide LBSA demand forecast is the combination of the U.S., U.S.-International, and International markets. The model provides detailed projections by market from the year 2030 to the year 2040. Figure 49 to Figure 51 shows the individual LBSA 52-seats market projection by year for Case 14 from Table 13. The market distribution indicates that the US market could capture 9% of the worldwide LBSA demand, while the US-International and International markets could capture 30% and 61%, respectively.

Figure 49 shows the following results in the US market between the year 2030 and the year 2040.

- One-way OD pairs: 70 to 84.
- Seats: 1.8 million to 2.6 million.
- Aircraft needed: 29 to 41.
- Passengers: 1.5 million to 2.2 million.

In the US-International market, Figure 50 shows the following results between the year 2030 and the year 2040.

- One-way OD pairs: 194 to 304.
- Seats: 5.3 million to 8.8 million.
- Aircraft needed: 180 to 299.
- Passengers: 4.5 million to 7.5 million.

In the International market, Figure 51 shows the following results between the year 2030 and the year 2040.

- One-way OD pairs: 436 to 596.
- Seats: 11.8 million to 21.1 million.
- Aircraft needed: 220 to 396.
- Passengers: 11.4 million to 20.6 million.

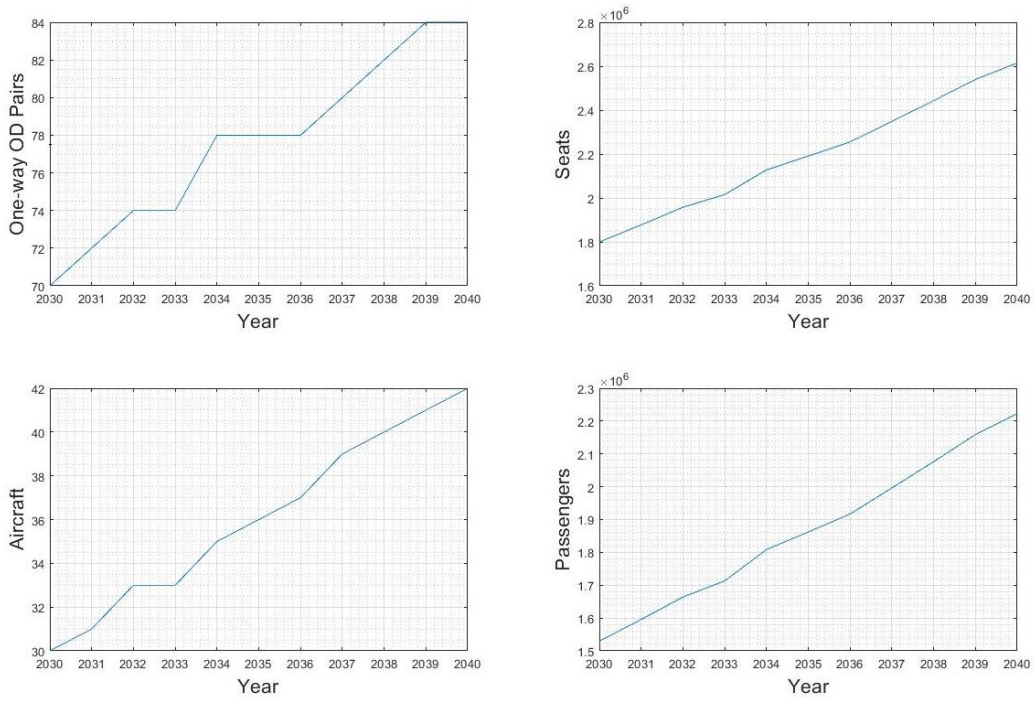


Figure 49: US Market Projections 52-Seats LBSA Aircraft: Case 14 Results.

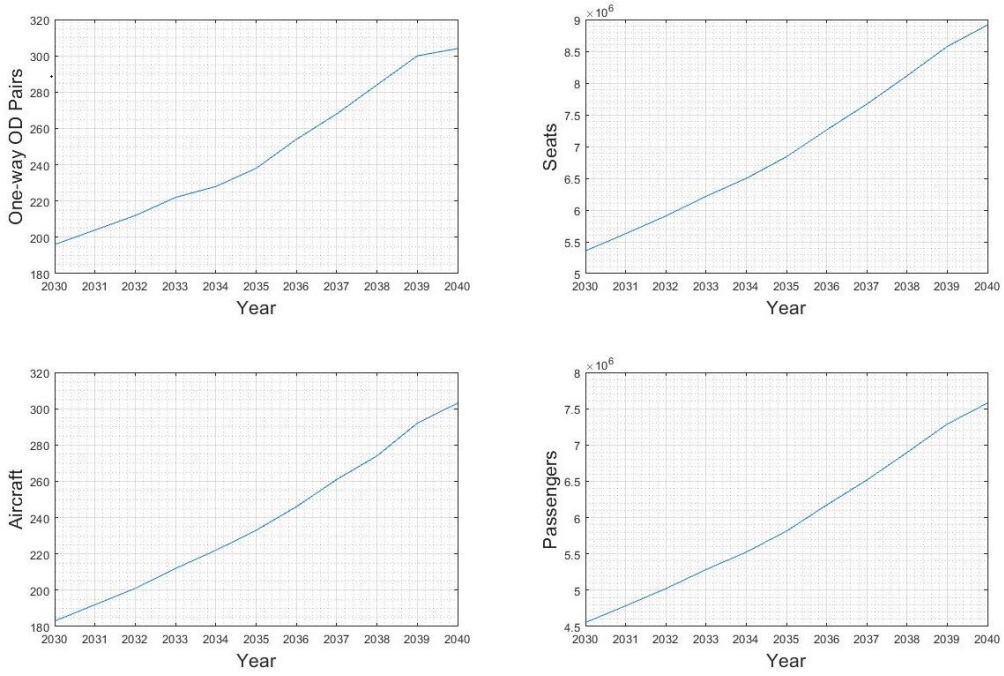


Figure 50: US-International Market Projection: 52-Seat LBSA Aircraft: Case 14 Results.

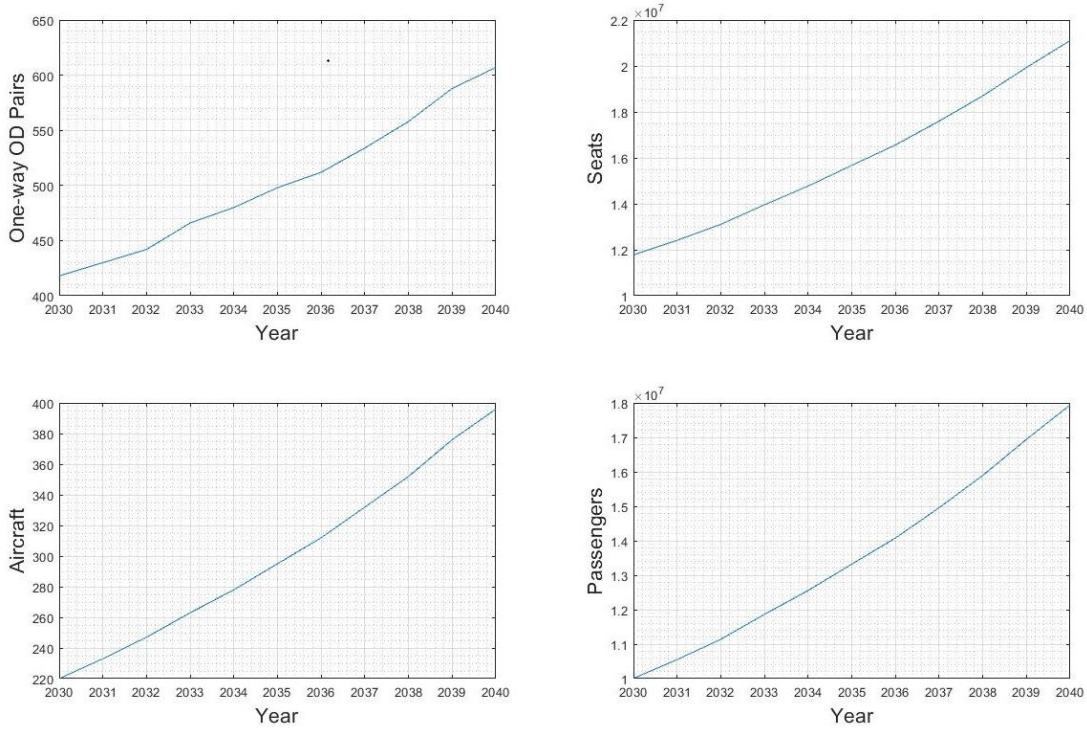


Figure 51: International Market Projection: 52-Seat LBSA Aircraft: Case 14 Results.

The results from a fuel scaling factor parametric analysis indicate the following. The effect of the fuel scaling factor on demand dissipates as the overland range increases. We observe a 9% increase in the LBSA seat demand projections when the overland fuel factor is decreased by 8% with a design overland range of 2,600 nm. and Mach 1.8 overland speed. LBSA aircraft demand projections increase 2% when the overland fuel factor is reduced by 2% with a design overland range of 3,200 nm. and Mach 1.8 overland speed.

This research work studies the potential LBSA demand for two vehicles. The effect that aircraft seating capacity has over the projections is the following. Increasing the aircraft seating capacity from 43-seats to 52-seats represents a 21% increase. The number of aircraft needed decrease by 6.7%, with a 21% increase in seating capacity. The seat-demand projection increases by 14.4% when the seating capacity increase by 21%. Finally, the number of potential one-way OD pairs decreases by 3.5%, with a 21% increase in LBSA seating capacity.

Aircraft Utilization, Size, and Range Analysis – A parametric case study

This Section illustrates how the range, aircraft size, and aircraft utilization affect LBSA demand. The cases presented in Table 13 assume aircraft utilization of 3,500 hours per year. Table 16 estimates LBSA market demand for multiple overland ranges and aircraft utilization varying from 1,800 to 3,500 annual hours. A total of 12 cases are presented for two LBSA aircraft configurations with 43 and 52 seats.

Table 18: LBSA 12 Case Matrix: Aircraft Utilization, Size, and Range Analysis.

	Parameters						
Case	Overland Range (nm.)	Overland Mach	Overland Fuel Scale Factor	Overwater Range (nm.)	Overwater Mach	Overwater Fuel Scale Factor	Aircraft Utilization(hrs.)
1	2,600	1.7	1.0	3,800	1.8	1.0	3,500
2	2,800	1.7	1.0	3,800	1.8	1.0	3,500
3	3,000	1.7	1.0	3,800	1.8	1.0	3,500
4	3,200	1.7	1.0	3,800	1.8	1.0	3,500
5	2,600	1.7	1.0	3,800	1.8	1.0	2,500
6	2,800	1.7	1.0	3,800	1.8	1.0	2,500
7	3,000	1.7	1.0	3,800	1.8	1.0	2,500
8	3,200	1.7	1.0	3,800	1.8	1.0	2,500
9	2,600	1.7	1.0	3,800	1.8	1.0	1,800
10	2,800	1.7	1.0	3,800	1.8	1.0	1,800
11	3,000	1.7	1.0	3,800	1.8	1.0	1,800
12	3,200	1.7	1.0	3,800	1.8	1.0	1,800

Table 19 presents the results for the scenarios presented in Table 18.

Increasing the aircraft seating capacity size from 43 to 52-seats (a 21% increase) has the following effect on LBSA demand. The results are for the year 2040.

- Increase seat demand by 13%
- Decrease the aircraft needed by 8%
- Decrease the number of one-way OD pairs by 3%

Increasing the overland range from 2,600 nm to 3,200 nm has the following effect on LBSA demand for the year 2040 (at 200 nm. increments in overland range while keeping the annual aircraft utilization constant at 3,500 hours).

- For the 52-seat LBSA aircraft
    - 2.4% increase in the number of aircraft needed.
    - 3.2% increase in seat demand.
    - 1.7% increase in the number of one-way OD pairs.
- For the 43-seat LBSA aircraft
    - 3.0% increase in the number of aircraft needed.
    - 3.7% increase in seat demand.
    - 2.0% increase in the number of one-way OD pairs.

Increasing the overland range from 2,600 nm to 3,200 nm has the following effect on LBSA demand for the year 2040 (at 200 nm increments in the overland range while keeping the annual aircraft utilization constant at 2,500 hours).

- For the 52-seat LBSA aircraft
    - 4.6% increase in the number of aircraft needed.
    - 5.1% increase in seat demand.
    - 4.2% increase in the number of one-way OD pairs.
- For the 43-seat LBSA aircraft
    - 5.3% increase in the number of aircraft needed.
    - 5.7% increase in seat demand.
    - 4.7% increase in the number of one-way OD pairs.

Increasing the overland range from 2,600 nm to 3,200 nm has the following effect on LBSA demand for the year 2040 (at 200 nm increments in the overland range while keeping the annual aircraft utilization constant at 1,800 hours).

- For the 52-seat LBSA aircraft
    - 9.6% increase in the number of aircraft needed.
    - 10.4% increase in seat demand.
    - 8.5% increase in the number of one-way OD pairs.
- For the 43-seat LBSA aircraft
    - 11.0% increase in the number of aircraft needed.
    - 11.8% increase in seat demand.
    - 11.7% increase in the number of one-way OD pairs.

At an overland range of 2,600, decreasing the annual aircraft utilization (AAU) affects LBSA demand for 2040.

- For the 52-seat LBSA aircraft
    - Reducing AAU from 3,500 hrs. to 2,500 hrs. yields:

        - 1.9% reduction in the number of aircraft needed.
        - 27.7% reduction in seat demand.
        - 25.6% reduction in the number of one-way OD pairs.
    - Reducing AAU from 2,500 hrs. to 1,800 hrs. yields:
        - 34.1% reduction in the number of aircraft needed.
        - 52.1% decrease in seat demand.
        - 48.2% decrease in the number of one-way OD pairs.
- For the 43-seat LBSA aircraft
    - Reducing AAU from 3,500 hrs. to 2,500 hrs. yields:
        - 4.0% reduction in the number of aircraft needed.
        - 30.3% reduction in seat demand.
        - 27.9% reduction in the number of one-way OD pairs.
    - Reducing AAU from 2,500 hrs. to 1,800 hrs. yields:
        - 29.3% reduction in the number of aircraft needed.
        - 48.9% reduction in seat demand.
        - 47.2% reduction in the number of one-way OD pairs.

At an overland range of 2,800, decreasing the annual aircraft utilization (AAU) affects LBSA demand for 2040.

- For the 52-seat LBSA aircraft
    - Reducing AAU from 3,500 hrs. to 2,500 hrs. yields:
        - 1.0% reduction in the number of aircraft needed.
        - 27.3% reduction in seat demand.
        - 24.3% reduction in the number of one-way OD pairs.
    - Reducing AAU from 2,500 hrs. to 1,800 hrs. yields:
        - 29.3% reduction in the number of aircraft needed.
        - 48.7% reduction in seat demand.
        - 46.6% reduction in the number of one-way OD pairs.
- For the 43-seat LBSA aircraft
    - Reducing AAU from 3,500 hrs. to 2,500 hrs. yields:
        - 4.8% reduction in the number of aircraft needed.
        - 30.4% reduction in seat demand.
        - 28.0% reduction in the number of one-way OD pairs.
    - Reducing AAU from 2,500 hrs. to 1,800 hrs.
        - 19.2% reduction in the number of aircraft needed.
        - 42.1% reduction in seat demand.
        - 37.3% reduction in the number of one-way OD pairs.
        -

At an overland range of 3,000, decreasing the annual aircraft utilization (AAU) affects LBSA demand for 2040.

- For the 52-seat LBSA aircraft
    - Reducing AAU from 3,500 hrs. to 2,500 hrs. yields:
        - 1.4% increase in the number of aircraft needed.
        - 25.9% decrease in seat demand.
        - 22.5% decrease in the number of one-way OD pairs.
    - Reducing AAU from 2,500 hrs. to 1,800 hrs. yields:
        - 27.3% decrease in the number of aircraft needed.
        - 46.4% decrease in seat demand.
        - 41.5% decrease in the number of one-way OD pairs.
- For the 43-seat LBSA aircraft
    - Reducing AAU from 3,500 hrs. to 2,500 hrs. yields:
        - 4.2% decrease in the number of aircraft needed.
        - 30.2% decrease in seat demand.
        - 27.1% decrease in the number of one-way OD pairs.
    - Reducing AAU from 2,500 hrs. to 1,800 hrs.
        - 19.8% decrease in the number of aircraft needed.
        - 41.7% decrease in seat demand.
        - 37.9% decrease in the number of one-way OD pairs.

At an overland range of 3,200, decreasing the annual aircraft utilization (AAU) affects LBSA demand for 2040.

- For the 52-seat LBSA aircraft
    - Reducing AAU from 3,500 hrs. to 2,500 hrs. yields:
        - 4.6% increase in the number of aircraft needed.
        - 23.6% reduction in seat demand.
        - 20.1% reduction in the number of one-way OD pairs.
    - Reducing AAU from 2,500 hrs. to 1,800 hrs. yields:
        - 24.2% reduction in the number of aircraft needed.
        - 44.4% reduction in seat demand.
        - 41.6% reduction in the number of one-way OD pairs.
- For the 43-seat LBSA aircraft
    - Reducing AAU from 3,500 hrs. to 2,500 hrs. yields:
        - 2.5% increase in the number of aircraft needed.
        - 26.2% reduction in seat demand.
        - 22.2% reduction in the number of one-way OD pairs.
    - Reducing AAU from 2,500 hrs. to 1,800 hrs. yields:
        - 17.3% reduction in the number of aircraft needed.
        - 39.6% reduction in seat demand.
        - 36.2% reduction in the number of one-way OD pairs.

Table 19: LBSA 12-Case Matrix Results: Aircraft Utilization, Size, and Range Analysis for 43 and 52-Seats LBSA Aircraft Demand Projections for the Year 2040.

Parameters			Aircraft		Seats (Million)		OD's	
Case	Overland Range (nm.)	Aircraft Utilization (hrs.)	43 Seats	52 Seats	43 Seats	52 Seats	43 Seats	52 Seats
1	2,600	3,500	741	694	26.4	30.0	969	942
2	2,800	3,500	765	714	27.3	31.1	997	974
3	3,000	3,500	792	723	28.5	32.0	1,029	984
4	3,200	3,500	810	746	29.4	33.0	1,029	992
5	2,600	2,500	711	681	18.4	21.7	699	701
6	2,800	2,500	728	707	19.0	22.6	718	737
7	3,000	2,500	759	733	19.9	23.7	750	763
8	3,200	2,500	830	780	21.7	25.2	801	793
9	2,600	1,800	503	449	9.4	10.4	369	363
10	2,800	1,800	588	500	11.0	11.6	450	438
11	3,000	1,800	609	533	11.6	12.7	466	446
12	3,200	1,800	686	591	13.1	14.0	511	463

Appendix B – Additional Findings and Recommendations

This section summarizes the findings of each of the three studies. In addition, the recommendations made for each study are described.

Findings for Study 1

The study focused on demand estimation for commercial supersonic services. The following findings are key to this study:

1. Analyzing various aircraft seating configurations for different international carriers shows that future supersonic aircraft must have at least equivalent comfort levels in premium economy seats today. Premium economy seats in long-range flights are 19 inches wide and 38-40 inches in pitch. For comparison, the BAC/Aerospatiale Concorde seats had 37 inches of pitch and 17.5 inches in width.
2. The results indicate that the number of one-way OD pairs that could support premium seat supersonic services in the year 2040 ranges from a low of 122 using 18-seat low boom aircraft to 624 for 60-seat non-low boom aircraft. These results assume 50% of passengers in premium seats paying equal to or more than the projected cost per mile of supersonic concept adjusted for the value of time.
3. The number of one-way OD pairs and demand is slightly higher for the non-low boom than the non-low boom restricted.
4. The results indicate that the number of supersonic aircraft needed to support commercial supersonic services in the year 2040 ranges from a low of 120 using 18-seat low boom aircraft to 681 for 60-seat non-low boom aircraft. These results assume 50% of passengers in premium seats paying equal to or more than the projected cost per mile of supersonic concept adjusted for the value of time.
5. Suppose 50% of passengers in premium seats paying equal to or more than the projected cost per mile of the supersonic concept adjusted for the value of time shift from subsonic to supersonic aircraft in the year 2040. In that case, the annual passenger demand in supersonic routes worldwide ranges from 1.35 million for the 18-seat supersonic aircraft to 19 million for a 60-seat low boom aircraft design. A 40-seat low boom design could attract close to eleven million passengers annually.

6. Based on the analysis presented, the future premium seat market may be fragmented into seats available in subsonic and supersonic aircraft. This fragmentation will occur naturally because not all premium passengers will be willing to pay for supersonic flights even when considering travel time savings.
7. The life cycle cost models developed for different aircraft types produced cost per passenger mile ranging from $1.57 for a low boom 18-seat supersonic aircraft to $0.61 for the 60-seat non-low boom supersonic aircraft.
8. 40-seat supersonic aircraft offer a reasonable tradeoff of size and economics of the configurations studied.
9. 60-seat supersonic aircraft could attract higher passenger demand than a 40-seat aircraft. However, the airport compatibility of 60-seat aircraft becomes an issue for many airports.
10. The results for the low boom design (number of one-way OD pairs, estimated demand, and number of aircraft needed over time) are lower than the results for the non-low boom and non-low boom restricted because of higher cost per passenger-mile.
11. In general, the non-low boom designs, if allowed to operate at transonic speed – Mach 1.15 – overland, have the highest demand because of better economics than low boom designs. Nevertheless, the potential market for 40 and 60-seat low-boom aircraft is close to the non-low boom aircraft.
12. The low boom aircraft designs use more fuel per flight than the non-low boom aircraft for a given distance profile (i.e., lower Specific Air Range). This is the result of higher aircraft weight and lower L/D with a net higher fuel consumption to meet a 2,500 nm design requirement flying overland at Mach 1.4. This fact requires further investigation.
13. The demand model used in the study employs historical fares paid by premium travelers from the Airline Research Corporation (ARC) database.
14. The percent of passengers willing to pay for faster travel speeds is driven by the cost per passenger-mile and travel time savings. Based on historical data of premium fares for direct and one-stop flights, travelers are willing to pay an average of $73 per hour (equivalent to $151,840 annual gross income) to $139 per hour (equal to $289,120 annual gross income). More investigation is needed on these numbers to understand if using averages underestimates the potential supersonic demand.

15. Parametric study of low boom design shows that decreasing the cost per passenger-mile of the 40-seat low boom configuration from $0.85/mi. to $0.75/mi. the low boom aircraft would attract an additional 0.9 million passengers worldwide per year compared to non-low boom aircraft.
16. Parametric analysis of the JFK to LAX one-way route concluded that the effect of travel time savings and the percent of passengers willing to pay is not substantial. The travel time savings produced by a speed increase of Mach 0.40 increase passenger willingness to pay by 4% (based on historical premium fares). However, increasing the cost per passenger-mile from $0.75/mi. to $0.85/mi. decreases the passenger willingness to pay by 20%.
17. Parametric analysis of the value of time indicates that a 28% increase in the value of time for the 18-Seat low boom aircraft can capture as much demand as the 18-seat non-low boom aircraft. For the 40 and 60-seat aircraft, the study indicates that with a 71% increase in the value of time above the baseline, the low boom design can capture as much demand as the non-low boom design.
18. An aspect of supersonic aircraft that has not been addressed in this study is the effect of high-altitude supersonic flight emissions in the upper layers of the atmosphere.
19. The low-boom designs are relatively long and may have airport compatibility problems operating from medium to large airports. For example, the 60-seat low-boom aircraft has an overall length of 257 feet - seven feet longer than Boeing 747-8I - the longest commercial aircraft operating today. Even a mid-size 43-seat low boom SST aircraft with an overall length of 243 feet will be five longer than an Airbus A380. Given the long wheelbase of supersonic aircraft, many airports will not be able to accommodate such aircraft without changes to the taxiway-apron infrastructure.

Recommendations for Study 1

1. 40 and 60-seat supersonic aircraft designs offer better economics to improve the chances of commercial feasibility.
2. Considering airport compatibility, the 40-seat aircraft designs have advantages over the 60-seat designs.

3. 40 and 60-seat low boom designs could be redesigned to make their airport compatibility easier. Relocating the nose gear in some configurations to a wheelbase of 80-90 feet would be desirable.
4. More analysis of the tradeoff between premium passenger fares paid and travel time savings should be carried out in a future study. The personal values of time estimated from actual premium fares offer a first-order alternative to understand how people value their time when switching from one-stop to direct flights. The estimated value of time ($117) for trips longer than 2,500 statute miles but less than 5,000 statute miles represents the top 1.7% of personal income earners in the United States (i.e., 167 million earners in 2017). This represents 2.7 million potential travelers in the US alone. Better estimates of the value of time could be obtained using passenger surveys or through focus group studies.
5. Further refinements to the performance of the low boom designs should be conducted. The differences in the fuel used for a given stage length between low boom and non-low boom designs do not favor the low boom design. This translates into higher operation costs (higher cost per passenger mile) and higher emissions per passenger.
6. Shifting passengers of premium seats from subsonic aircraft to supersonic commercial aircraft would affect coach class tickets. This impact is not trivial and should be analyzed in a follow-up study. A quick analysis of some premium routes shows that the airline revenue attributed to premium seats ranges from 25-40% per flight. If a fraction of premium passengers shifts to supersonic services, airlines may have to increase their coach fares by 10-15% in such routes. Assuming a demand elasticity with a fare of -0.8 (for long-distance flights) could reduce passenger demand in coach services by 7-12%. This has real implications for the airline business.

Findings for Study 2

The following points summarize the findings of Study 2, assuming SST operations in the year 2040. All findings assume that 50% of projected demand in the SST market are willing to pay equal or more than the projected cost per mile of a supersonic flight adjusted for the value of time. The Low-Boom design results are higher when compared to the non-Low-Boom and non-Low-Boom with subsonic speed restrictions. The Low-Boom SST design has a higher degree of

optimization and should not be compared directly to the non-Low-Boom SST designs. The load factor used to estimate the number of potential US passengers is 85%. The load factor for international routes is assumed to be 81.5%.

The aircraft speed assumptions are as follows:

- Low-Boom SST Design: Mach 1.6 overland and Mach 1.8 overwater
- Non-Low-Boom SST Design: Mach 1.15 overland and Mach 1.8 overwater
- Non-Low-Boom restricted SST Design: Mach 0.95 overland and Mach 1.8 overwater

The findings for a 40-Seat, Low-Boom SST aircraft are:

A. Worldwide SST Market
    1. Low-Boom SST Design
        - Up to 1,555 one-way OD pairs could support SST services.
        - The number of seats offered could reach 47.1 million.
        - The number of aircraft needed to satisfy the demand is 1,725.
        - The number of passengers could reach 38.4 million.
    2. Non-low-Boom SST Design
        - Up to 1,371 one-way OD pairs could support SST services.
        - The number of seats offered could reach 40 million.
        - The number of aircraft needed to satisfy the demand is 1,443.
        - The number of passengers could reach 32.6 million.
    3. Non-low-Boom restricted
        - Up to 1,247 one-way OD pairs could support SST services.
        - The number of seats offered could reach 36.9 million.
        - The number of aircraft needed to satisfy the demand is 1,525.
        - The number of passengers could reach 30.1 million.
B. US SST Market
    1. Low-Boom SST Design
        - Up to 62 one-way OD pairs could support SST services.

The findings for a 20-Seat, Low-Boom SST aircraft design are:

A. Worldwide SST Market

1. Low-Boom SST Design
    - Up to 862 one-way OD pairs could support SST services.
    - The number of seats offered could reach 12.8 million.
    - The number of aircraft needed to satisfy the demand is 1,006.
    - The number of passengers could reach 10.4 million.
2. Non-low-Boom SST Design
    - Up to 960 one-way OD pairs could support SST services.
    - The number of seats offered could reach 13.9 million.
    - The number of aircraft needed to satisfy the demand is 1,099.
    - The number of passengers could reach 11.4 million.
3. Non-low-Boom restricted
    - Up to 872 one-way OD pairs could support SST services.
    - The number of seats offered could reach 12.9 million.
    - The number of aircraft needed to satisfy the demand is 1,149.
    - The number of passengers could reach 10.5 million.

Additional findings of the study are:

1) In general, the International market contributes 66% of the worldwide SST market projections. The US-International and US markets contribute 31% and 3% of the worldwide SST market projections, respectively.

2) The ARC fare analysis results indicate that the values of time for the economy premium class are $39/hr., $29/hr., and $43/hr. for the US market, US-International market, and International markets, respectively.

3) The ARC fare analysis results indicate that the values of time for the premium class (first-class and business class) are $61/hr., $165/hr., and $140/hr. for the US market, US-International market, and International markets, respectively.

4) The value of time for the US premium market ($61/hr.) is considerably lower than the value of time for the US-International and International premium markets. This could be due to a highly competitive market in the US and the US economic recession a few years before 2012.

5) We estimate a loss of 0.6% in arrival runway capacity for every 1% in SST operations at the airport with future SST operations.

6) With future SST operations, we can expect a loss of 1.1% departure runway capacity for every 1% in SST departure operations at the airport.

7) Based on the analysis presented, the future premium seat market may be fragmented into seats available in subsonic and supersonic aircraft. This fragmentation will occur naturally because not all premium passengers will be willing to pay for supersonic flights even when considering travel time savings.

8) The life cycle cost model developed for a 40-seat Low-Boom SST aircraft design produced cost per passenger mile ranging from $0.93 to $0.61 depending on route characteristics (speed, distance, and market).

9) The life cycle cost model developed for a 20-seat Low-Boom SST aircraft design produced cost per passenger mile ranging from $1.46 to $0.96 depending on route characteristics (speed, distance, and market).

10) The life cycle cost model developed for a 40-seat non-Low-Boom and non-Low-Boom restricted SST aircraft design produced cost per passenger mile ranging from $1.05 to $0.61 depending on route characteristics (speed, distance, and market).

11) The life cycle cost model developed for a 20-seat non-Low-Boom and non-Low-Boom restricted SST aircraft design produced cost per passenger mile ranging from $1.67 to $0.87 depending on route characteristics (speed, distance, and market).

12) The demand model used in the study employs historical fares paid by premium travelers from the Airline Research Corporation (ARC) database (the year 2012).

13) The percent of passengers willing to pay for faster travel speeds is driven by the cost per passenger-mile and travel time savings. Based on historical data of economy premium and premium fares for direct and one-stop flights, a value of time from each class of service and market was derived.

14) Parametric study of Low-Boom design shows that when increasing the value of time for the US premium market from $61/hr. to $140/hr., the number of potential one-way OD pairs will increase from 62 to 114 by the year 2040.
15) Parametric study of Low-Boom design shows that when increasing the value of time for the US premium market from $61/hr. to $140/hr., the number of seats will increase from 1.45 to 2.73 million by the year 2040.
16) Parametric study of Low-Boom design shows that when increasing the value of time for the US premium market from $61/hr. to $140/hr. the number of aircraft needed to satisfy the projected demand increases from 29 to 53 by the year 2040.
17) Parametric study of Low-Boom design shows that when increasing the value of time for the US premium market from $61/hr. to $140/hr., the number of passengers increased from 1.23 to 2.32 million by the year 2040.

Recommendations for Study 2

1. Optimize the 40-seat non-Low-Boom aircraft design configurations to facilitate the comparisons with the 40-seat Low-Boom aircraft.
2. Considering airport compatibility, identify the longest runway of each potential airport to see if the SST has enough runway length to operate.
3. To re-evaluate the value of time for the US premium market (currently $61/hr.) with new ARC data from a recent year to investigate if a higher value can be used.
4. Optimize the cost per passenger mile used on each one-way OD pair based on the percent of the time the aircraft operates overland vs. overwater. Currently, suppose the distance between the airport of origin and the destination airport is less or equal to 2,500 nautical miles. In that case, the aircraft is limited to the overland speeds (Mach 0.95, Mach 1.15, or Mach 1.6 for the non-Low-Boom, non-Low-Boom restricted, and Low-Boom aircraft, respectively). Suppose the distance between the airport of origin and the destination airport is greater than 2,500 nautical miles. In that case, the aircraft is allowed to overwater speed (Mach 1.8 for all three aircraft designs).

Findings for Study 3

The following points summarize the preliminary findings for LBSA aircraft operations in the year 2040. All results assume that 50% of the LBSA aircraft market's projected demand is willing to pay equal or more than the projected cost per mile of a supersonic flight adjusted for the value of time. The load factor used to estimate the number of potential US passengers is 85%. The load factor for international routes is assumed to be 81.5%. A total of 24 cases are considered for two LBSA designs with 43 and 52 seats.

The findings for a 52-Seat, Low-Boom supersonic aircraft are:

- Case 17 shows the highest seat demand, with 33.4 million seats worldwide. This Case used an overland range of 3,200 nm., an overland Mach 1.7, and an overland fuel scale factor of 0.98.
- Case 1 shows the lowest seat demand, with 26.8 million seats worldwide. This Case used an overland range of 2,600 nm., an overland Mach 1.8, and an overland fuel scale factor of 1.0.
- Case 13 shows the highest number of aircraft needed, with 772 aircraft worldwide. This Case used an overland range of 2,800 nm., an overland Mach 1.7, and an overland fuel scale factor of 0.90.
- Case 1 shows the lowest number of aircraft needed, with 635 aircraft worldwide. This Case used an overland range of 2,600 nm., an overland Mach 1.8, and an overland fuel scale factor of 1.0.
- Case 13 shows 1,032 one-way OD pairs for the highest case in where LBSA can operate. This Case used an overland range of 2,800 nm., an overland Mach 1.7, and an overland fuel scale factor of 0.90.
- Case 1 shows 824 one-way OD pairs for the lowest case in where LBSA can operate. This Case used an overland range of 2,600 nm., an overland Mach 1.8, and an overland fuel scale factor of 1.0.

The effect of overland range on LBSA market demand at constant overland speed are:

- At Mach 1.8 overland, projections increase 3.5% by every 200 nm. increments in the overland range.
- At Mach 1.7 overland, projections increase 2.1% by every 200 nm. increments in the overland range.
- At Mach 1.6 overland, projections increase 1.3% by every 200 nm. increments in the overland range.

The effect of overland speed on LBSA market demand at a constant overland range are:

- Projections increase by 2.2% when increasing the overland Mach from 1.6 to 1.7 at an overland range of 2,800 nm.
- Projections decrease by 9.9% when increasing the overland Mach from 1.7 to 1.8 at an overland range of 2,800 nm.
- Projections increase by 2.2% when increasing the overland Mach from 1.6 to 1.7 at an overland range of 3,000 nm.
- Projections decrease by 10.1% when increasing the overland Mach from 1.7 to 1.8 at an overland range of 3,000 nm.
- Projections increase by 3.7% when increasing the overland Mach from 1.6 to 1.7 at an overland range of 3,200 nm.
- Projections decrease by 7.8% when increasing the overland Mach from 1.7 to 1.8 at an overland range of 3,200 nm.

The findings for a 43-Seat, Low-Boom supersonic aircraft design are:

- Case 17 shows the highest seat demand, with 29.8 million seats worldwide. This Case used an overland range of 3,200 nm., an overland Mach 1.7, and an overland fuel scale factor of 0.98.
- Case 1 shows the lowest seat demand, with 23.9 million seats worldwide. This Case used an overland range of 2,600 nm., an overland Mach 1.8, and an overland fuel scale factor of 1.0.
- Case 13 shows the highest number of aircraft needed, with 830 aircraft worldwide. This Case used an overland range of 2,800 nm., an overland Mach 1.7, and an overland fuel scale factor of 0.90.
- Case 1 shows the lowest number of aircraft needed, with 696 aircraft worldwide. This Case used an overland range of 2,600 nm., an overland Mach 1.8, and an overland fuel scale factor of 1.0.
- Case 13 shows 1,057 one-way OD pairs for the highest case in where LBSA can operate. This Case used an overland range of 2,800 nm., an overland Mach 1.7, and an overland fuel scale factor of 0.90.
- Case 1 shows 882 one-way OD pairs for the lowest case in where LBSA can operate. This Case used an overland range of 2,600 nm., an overland Mach 1.8, and an overland fuel scale factor of 1.0.

The effect of overland range on LBSA market demand at constant overland speed are:

- At Mach 1.8 overland, projections increase 4.1% for every 200 nm increment in the overland range.
- At Mach 1.7 overland, projections increase 2.8% for every 200 nm increment in the overland range.

- At Mach 1.6 overland, projections increase 1.3% for every 200 nm increment in the overland range.

The effect of overland speed on LBSA market demand at a constant overland range are:

- Projections increase by 7.0% when increasing the overland Mach from 1.6 to 1.7 at an overland range of 2,800 nm.
- Projections decrease by 7.3% when increasing the overland Mach from 1.7 to 1.8 at an overland range of 2,800 nm.
- Projections increase by 9.5% when increasing the overland Mach from 1.6 to 1.7 at an overland range of 3,000 nm.
- Projections decrease by 8.1% when increasing the overland Mach from 1.7 to 1.8 at an overland range of 3,000 nm.
- Projections increase by 10.0% when increasing the overland Mach from 1.6 to 1.7 at an overland range of 3,200 nm.
- Projections decrease by 4.9% when increasing the overland Mach from 1.7 to 1.8 at an overland range of 3,200 nm.

The results from a fuel scaling factor parametric analysis indicate the following. A 9% increase in the LBSA projections is expected when the overland fuel factor decreases by 8% with an overland range of 2,600 nm. and overland Mach 1.8. A 2% increase in the LBSA projections is expected when the overland fuel factor is decreased by 2% at an overland range of 3,200 nm. and overland Mach 1.8.

This research work studies the potential LBSA demand for two vehicles. The effect that aircraft seating capacity has over the projections is the following. Increasing the aircraft seating capacity from 43-seats to 52-seats represents a 21% increase in capacity. The number of aircraft needed to satisfy worldwide demand decreases by 6.7%, with a 21% increase in seating capacity. The seat-demand projection increases by 14.4% when the seating capacity increases by 21%. Finally, the number of potential one-way OD pairs decreases by 3.5%, with a 21% increase in LBSA seating capacity. The effect of LBSA aircraft annual utilization (AAU) is summarized below.

For the LBSA aircraft with 2800 nm overland range, we estimate the following trends:

- For a 52-seat LBSA aircraft
    - Reducing AAU from 3,500 hrs. to 2,500 hrs. yields:
        - 1.5% reduction in the number of aircraft needed.
        - 25.2% reduction in seat demand.

        - 22.2% reduction in the number of one-way OD pairs.
    - Reducing AAU from 2,500 hrs. to 1,800 hrs. yields:
        - 15.3% reduction in the number of aircraft needed.
        - 38.2% reduction in seat demand.
        - 35.2% d reduction in the number of one-way OD pairs.
- For a 43-seat LBSA aircraft
    - Reducing AAU from 3,500 hrs. to 2,500 hrs. yields:
        - 1.7% reduction in the number of aircraft needed.
        - 28.3% reduction in seat demand.
        - 24.6% reduction in the number of one-way OD pairs.
    - Reducing AAU from 2,500 hrs. to 1,800 hrs. yields:
        - 11.8% reduction in the number of aircraft needed.
        - 36.5% reduction in seat demand.
        - 33.2% reduction in the number of one-way OD pairs.

For the LBSA aircraft with 3000 nm overland range, we estimate the following trends:

- For a 52-seat LBSA aircraft
    - Reducing AAU from 3,500 hrs. to 2,500 hrs. yields:
        - 3.0% increase in the number of aircraft needed.
        - 24.5% reduction in seat demand.
        - 20.9% reduction in the number of one-way OD pairs.
    - Reducing AAU from 2,500 hrs. to 1,800 hrs. yields:
        - 15.1% reduction in the number of aircraft needed.
        - 37.8% reduction in seat demand.
        - 33.9% reduction in the number of one-way OD pairs.
- For a 43-seat LBSA aircraft
    - Reducing AAU from 3,500 hrs. to 2,500 hrs. yields:
        - 1.5% reduction in the number of aircraft needed.
        - 28.4% reduction in seat demand.
        - 24.4% reduction in the number of one-way OD pairs.
    - Reducing AAU from 2,500 hrs. to 1,800 hrs.
        - 11.0% reduction in the number of aircraft needed.
        - 35.7% reduction in seat demand.
        - 31.8% reduction in the number of one-way OD pairs.

Additional findings of the study are:

1) The international market contributes 61% of the worldwide LBSA demand. The US-International and US markets contribute 30% and 9% of the worldwide LBSA aircraft market projections.

2) The ARC fare analysis results indicate that the weighted average values of time for the premium class are $103/hr., $203/hr., and $113/hr. for the US market, US-International market, and International markets, respectively.
3) The demand model used in the study employs historical fares paid by premium travelers from the Airline Research Corporation (ARC) database (the year 2016).
4) The percent of passengers willing to pay for faster travel speeds is driven by the cost per passenger-mile and travel time savings. Based on historical data of premium fares for direct and one-stop flights, a value of time from each class of service and market was derived.

Recommendations for Study 3

1. Investigate the effect of a worldwide network comprised of different airlines in the LBSA market demand. This addresses the need to investigate significant inefficiencies in aircraft utilization when airlines have limited LBSA aircraft numbers to fly a large number of OD pairs in their network.
2. Evaluate the design aspects of LBSA design in more detail to understand how maintenance and logistic support actions may play a role in the worldwide demand for LBSA aircraft services. Logistic support for dedicated LBSA flights may influence airline choices on where to deploy the vehicles. The demand estimated in this analysis does not consider the normal down selection of airlines' markets when deploying expensive assets.
3. Investigate the relationship between trip length and sub-mode choice for various LBSA cabin configurations. The current study was limited to standard seating configurations for all OD routes flown by LBSA aircraft. Longer routes may require “sleeper” seats that will reduce the number of passengers to be carried in a given flight. This can have a substantial effect on ticket prices for such routes. Companies’ marketing supersonic aircraft show at least two cabin configurations depending on the length of the flights.
4. Improve the algorithms to estimate block fuel as a function of distance for both overland and overwater missions. NASA provided surrogate models that provide accurate information for the vehicle design range. However, the method used to estimate block fuel for distances less than the design range seems to underestimate the actual block fuel for short to medium-distance flights. This reduces the cost of short and intermediate flights and increases LBSA demand in

those routes. An alternative procedure to estimate block fuel over distance is to synchronize the NASA surrogate model that produces the aircraft design parameters (thrust, OEW, ZFW, and TOGW) with the second surrogate model that produces block fuel as a function of distance.

5. Investigate the fuel reserves built into FLOPS to assess the practical range of LBSA aircraft using typical airline reserves. Our experience with long oceanic flights is that airlines carry 16-17% of fuel above the actual fuel consumed.
6. Consider airport curfews and desired arrival and departure times in the LBSA analysis. Many airports impose nighttime curfews that could affect the demand for LBSA services. LBSA aircraft utilization in an airline network may be influenced by desired departure and arrival times to airports.

Appendix C – LBSAM Source Code

This section presents the source code of the Low Boom Supersonic Aircraft Model.

ARC_Main_Script

```
%% This script extract the ARC 2016 records from the original txt file.
% It also separates the records by trip type
% Oneway NonStop
% Oneway OneStop
% Roundtrip NonStop

clear
clc

%check add path function
local_disc = '';
main_delimiter = '\';
main_file_name = '.';
Input_Folder_Dir = ([local_disc,'..\..\SST_2020_Input']);
ARC_2016_Dir = ([local_disc,main_file_name,main_delimiter,'ARC_2016']);

addpath(genpath('.\'));
addpath(genpath(Input_Folder_Dir));

%Create Output directory for ARC 2016 main folder
if exist(([local_disc,main_file_name,main_delimiter,'ARC_2016\Output']),'dir') == 0

mkdir(([local_disc,main_file_name,main_delimiter,'ARC_2016\Output']))
mkdir(([local_disc,main_file_name,main_delimiter,'ARC_2016\Output\ARC_Data_Extracted']))
mkdir(([local_disc,main_file_name,main_delimiter,'ARC_2016\Output\ARC_Data_Filtered']))

end %if exist(([local_disc,main_file_name,delimeter,'ARC_2016\Output'])) == 0

addpath(local_disc,main_file_name,main_delimiter,'ARC_2016');

Import_ARC_2016(ARC_2016_Dir,Input_Folder_Dir,main_delimiter)

% clearvars -except local_disc main_delimeter main_file_name Input_Folder_Dir ARC_2016_Dir
Remove_records(ARC_2016_Dir,main_delimiter)

% clearvars -except local_disc main_delimeter main_file_name Input_Folder_Dir ARC_2016_Dir
Identify_Oneway_NonStop(ARC_2016_Dir,main_delimiter)

% clearvars -except local_disc main_delimeter main_file_name Input_Folder_Dir ARC_2016_Dir
Identify_Oneway_OneStop(ARC_2016_Dir,main_delimiter,Input_Folder_Dir)

% clearvars -except local_disc main_delimeter main_file_name Input_Folder_Dir ARC_2016_Dir
Identify_RoundTrip_NonStop(ARC_2016_Dir,main_delimiter)

rmpath(local_disc,main_file_name,main_delimiter,'ARC_2016');
```

```
function [] = Import_ARC_2016(ARC_2016_Dir,Input_Folder_Dir,main_delimiter)


save_dir = ([ARC_2016_Dir,main_delimiter,'Output\ARC_Data_Extracted']);
filename = ([Input_Folder_Dir,main_delimiter,'ARC_2016\Report_52_Weeks.txt']);
delimiter = '\t';

first_row = 2;
increment = 5000000;
length_of_data = 64779197;
number_of_runs = round(length_of_data/increment);
```

```
13
14 for run = 1:number_of_runs
15
16 disp(['Analyzing Data ',num2str(run),'/',num2str(number_of_runs)])
17 % Import data from text file.
18 % Script for importing data from the following text file:
19 %
20 % C:\Users\efreire\Desktop\ARC 2016\Report_52_Weeks.txt
21 %
22 % To extend the code to different selected data or a different text file,
23 % generate a function instead of a script.
24
25 % Auto-generated by MATLAB on 2020/02/19 15:35:41
26
27 %% Initialize variables.
28
29 if run == 1
30
31 startRow = first_row;
32 endRow = first_row + increment-1;
33
34 elseif run == 13
35
36 startRow = first_row + increment*(run-1);
37 endRow = length_of_data;
38
39 else
40
41 startRow = first_row + increment*(run-1);
42 endRow = (first_row-1) + increment * run;
43
44 end
45
46 %% Format for each line of text:
47 % column1: TicketIdentifier - int64 (%d64)
48 % column2: OriginAirport - text (%s)
49 % column3: DestinationAirport - text (%s)
50 % column4: SEGment - double (%f)
51 % column5: DOCUMENTAMOUNT - double (%f)
52 % column6: DOC_AMOUNT_USD - double (%f)
53 % column7: TRIP_TYPE_CD - text (%s)
54 % column8: CLASS_OF_SVC_CD - text (%s)
55 % column9: CABINCLASS - text (%s)
56 % column10: PRORATED_DOCUMENT_AMOUNT - double (%f)
57 % column11: SegmentMile - double (%f)
58 % column12: TotalMile - double (%f)
59 % For more information, see the TEXTSCAN documentation.
60 formatSpec = '%d64%s%s%f%f%f%s%s%s%f%f%f%[^\n\r]';
61
62 %% Open the text file.
63 fileID = fopen(filename,'r');
64
65 %% Read columns of data according to the format.
66 % This call is based on the structure of the file used to generate this
67 % code. If an error occurs for a different file, try regenerating the code
68 % from the Import Tool.
69 dataArray = textscan(fileID, formatSpec, endRow-startRow+1, 'Delimiter', delimiter, 'TextType', 'string', 'HeaderLines', startRow-1,
'ReturnOnError',
false, 'EndOfLine', '\r\n');
70
71 %% Close the text file.
72 fclose(fileID);
73
74 %% Post processing for unimportable data.
75 % No unimportable data rules were applied during the import, so no post
76 % processing code is included. To generate code which works for
```

```
77 % unimportable data, select unimportable cells in a file and regenerate the
78 % script.
79
80 %% Allocate imported array to column variable names
81 TicketIdentifier = dataArray{:, 1};
82 OriginAirport = cellstr(dataArray{:, 2});
83 DestinationAirport = cellstr(dataArray{:, 3});
84 SEGment = dataArray{:, 4};
85 DOCUMENTAMOUNT = dataArray{:, 5};
86 DOC_AMT_USD = dataArray{:, 6};
87 TRIP_TYPE_CD = cellstr(dataArray{:, 7});
88 CLASS_OF_SVC_CD = cellstr(dataArray{:, 8});
89 CABINCLASS = cellstr(dataArray{:, 9});
90 PRORATED_DOCUMENT_AMOUNT = dataArray{:, 10};
91 SegmentMile = dataArray{:, 11};
92 TotalMile = dataArray{:, 12};
93
94
95 %% Clear temporary variables
96 clearvars startRow endRow formatSpec fileID dataArray ans;
97
98 %%Save variables
99 TicketIdentifier_ser = getByteStreamFromArray(TicketIdentifier);
100 OriginAirport_ser = getByteStreamFromArray(OriginAirport);
101 DestinationAirport_ser = getByteStreamFromArray(DestinationAirport);
102 SEGment_ser = getByteStreamFromArray(SEGment);
103 DOCUMENTAMOUNT_ser = getByteStreamFromArray(DOCUMENTAMOUNT );
104 DOC_AMT_USD_ser = getByteStreamFromArray(DOC_AMT_USD);
105 TRIP_TYPE_CD_ser = getByteStreamFromArray(TRIP_TYPE_CD);
106 CLASS_OF_SVC_CD_ser = getByteStreamFromArray(CLASS_OF_SVC_CD);
107 CABINCLASS_ser = getByteStreamFromArray(CABINCLASS);
108 PRORATED_DOCUMENT_AMOUNT_ser = getByteStreamFromArray(PRORATED_DOCUMENT_AMOUNT);
109 SegmentMile_ser = getByteStreamFromArray(SegmentMile);
110 TotalMile_ser = getByteStreamFromArray(TotalMile);
111
112
113 save([save_dir,'\TicketIdentifier_ser_Run_',num2str(run),'.mat'],'TicketIdentifier_ser')
114 save([save_dir,'\OriginAirport_ser_Run_',num2str(run),'.mat'],'OriginAirport_ser')
115 save([save_dir,'\DestinationAirport_ser_Run_',num2str(run),'.mat'],'DestinationAirport_ser')
116 save([save_dir,'\SEGment_ser_Run_',num2str(run),'.mat'],'SEGment_ser')
117 save([save_dir,'\DOCUMENTAMOUNT_ser_Run_',num2str(run),'.mat'],'DOCUMENTAMOUNT_ser')
118 save([save_dir,'\DOC_AMT_USD_ser_Run_',num2str(run),'.mat'],'DOC_AMT_USD_ser')
119 save([save_dir,'\TRIP_TYPE_CD_ser_Run_',num2str(run),'.mat'],'TRIP_TYPE_CD_ser')
120 save([save_dir,'\CLASS_OF_SVC_CD_ser_Run_',num2str(run),'.mat'],'CLASS_OF_SVC_CD_ser')
121 save([save_dir,'\CABINCLASS_ser_Run_',num2str(run),'.mat'],'CABINCLASS_ser')
122 save([save_dir,'\PRORATED_DOCUMENT_AMOUNT_ser_Run_',num2str(run),'.mat'],'PRORATED_DOCUMENT_AMOUNT_ser')
123 save([save_dir,'\SegmentMile_ser_Run_',num2str(run),'.mat'],'SegmentMile_ser')
124 save([save_dir,'\TotalMile_ser_Run_',num2str(run),'.mat'],'TotalMile_ser')
125
126 disp(['Saved_Run_',num2str(run)])
127
128 clearvars -except save_dir first_row increment length_of_data number_of_runs filename delimiter
129 end
130
131 %%
132
133
134 for run = 1:number_of_runs
135
136 disp(['Analyzing Data ',num2str(run),'/',num2str(number_of_runs)])
137
138
139 %% Ticket Number
140 disp(['Loading TicketIdentifier_ser_Run_',num2str(run)])
141 load([save_dir,'\TicketIdentifier_ser_Run_',num2str(run),'.mat'],'TicketIdentifier_ser')
142
```

```
disp('Getting Array from Byte Stream')
TicketIdentifier = getArrayFromByteStream(TicketIdentifier_ser); clear TicketIdentifier_ser
if run == 1

TICKET_NBR = TicketIdentifier;
clear TicketIdentifier

else

TICKET_NBR = [TICKET_NBR;TicketIdentifier];
clear TicketIdentifier
end

%% Origin Airport

disp(['Loading OriginAirport_ser_Run_',num2str(run)])
load([save_dir,'\OriginAirport_ser_Run_',num2str(run),'.mat'],'OriginAirport_ser')

disp('Getting Array from Byte Stream')
OriginAirport = getArrayFromByteStream(OriginAirport_ser); clear OriginAirport_ser
if run == 1

ORIG_ARPT_CD = OriginAirport;
clear TicketIdentifier

else

ORIG_ARPT_CD = [ORIG_ARPT_CD;OriginAirport];
clear OriginAirport
end

%% Destination Airport

disp(['Loading DestinationAirport_ser_Run_',num2str(run)])
load([save_dir,'\DestinationAirport_ser_Run_',num2str(run),'.mat'],'DestinationAirport_ser')

disp('Getting Array from Byte Stream')
DestinationAirport = getArrayFromByteStream(DestinationAirport_ser); clear DestinationAirport_ser
if run == 1

DEST_ARPT_CD = DestinationAirport;
clear DestinationAirport

else

DEST_ARPT_CD = [DEST_ARPT_CD;DestinationAirport];
clear DestinationAirport
end

%% Segment ID

disp(['Loading SEGment_ser_Run_',num2str(run)])
load([save_dir,'\SEGment_ser_Run_',num2str(run),'.mat'],'SEGment_ser')

disp('Getting Array from Byte Stream')
SEGment = getArrayFromByteStream(SEGment_ser); clear SEGment_ser
if run == 1

SEG_ID = SEGment;
clear SEGment

else

SEG_ID = [SEG_ID;SEGment];
clear SEGment
end
```

```
209
210 %% DOCUMENT AMOUNT
211
212 disp(['Loading DOCUMENTAMOUNT_ser_Run_',num2str(run)])
213 load([save_dir,'\DOCUMENTAMOUNT_ser_Run_',num2str(run),'.mat'],'DOCUMENTAMOUNT_ser')
214
215 disp('Getting Array from Byte Stream')
216 DOCUMENTAMOUNT = getArrayFromByteStream(DOCUMENTAMOUNT_ser); clear DOCUMENTAMOUNT_ser
217 if run == 1
218
219 DOC_AMT = DOCUMENTAMOUNT;
220 clear DOCUMENTAMOUNT
221
222 else
223
224 DOC_AMT = [DOC_AMT;DOCUMENTAMOUNT];
225 clear DOCUMENTAMOUNT
226 end
227
228 %% DOCUMENT AMOUNT
229
230 disp(['Loading PRORATED_DOCUMENT_AMOUNT_ser_Run_',num2str(run)])
231 load([save_dir,'\PRORATED_DOCUMENT_AMOUNT_ser_Run_',num2str(run),'.mat'],'PRORATED_DOCUMENT_AMOUNT_ser')
232
233 disp('Getting Array from Byte Stream')
234 PRORATED_DOCUMENT_AMOUNT = getArrayFromByteStream(PRORATED_DOCUMENT_AMOUNT_ser); clear
PRORATED_DOCUMENT_AMOUNT_ser
235 if run == 1
236
237 SEG_DOC_AMT = PRORATED_DOCUMENT_AMOUNT;
238 clear PRORATED_DOCUMENT_AMOUNT
239
240 else
241
242 SEG_DOC_AMT = [SEG_DOC_AMT;PRORATED_DOCUMENT_AMOUNT];
243 clear PRORATED_DOCUMENT_AMOUNT
244 end
245
246
247 %% SEG_MILE
248
249 disp(['Loading SegmentMile_ser_Run_',num2str(run)])
250 load([save_dir,'\SegmentMile_ser_Run_',num2str(run),'.mat'],'SegmentMile_ser')
251
252 disp('Getting Array from Byte Stream')
253 SegmentMile = getArrayFromByteStream(SegmentMile_ser); clear SegmentMile_ser
254 if run == 1
255
256 SEG_MILE = SegmentMile;
257 clear SegmentMile
258
259 else
260
261 SEG_MILE = [SEG_MILE;SegmentMile];
262 clear SegmentMile
263 end
264
265
266 %% Total_MILE
267
268 disp(['Loading TotalMile_ser_Run_',num2str(run)])
269 load([save_dir,'\TotalMile_ser_Run_',num2str(run),'.mat'],'TotalMile_ser')
270
271 disp('Getting Array from Byte Stream')
272 TotalMile = getArrayFromByteStream(TotalMile_ser); clear TotalMile_ser
273 if run == 1
```

```
274
275 Total_Mile = TotalMile;
276 clear TotalMile
277
278 else
279
280 Total_Mile = [Total_Mile;TotalMile];
281 clear TotalMile
282 end
283
284 %% Trip Type
285
286 disp(['Loading TRIP_TYPE_CD_ser_Run_',num2str(run)])
287 load([save_dir,'\TRIP_TYPE_CD_ser_Run_',num2str(run),'.mat'],'TRIP_TYPE_CD_ser')
288
289 disp('Getting Array from Byte Stream')
290 TRIP_TYPE_CD = getArrayFromByteStream(TRIP_TYPE_CD_ser); clear TRIP_TYPE_CD_ser
291 if run == 1
292
293 TRIP_TYPE = TRIP_TYPE_CD;
294 clear TRIP_TYPE_CD
295
296 else
297
298 TRIP_TYPE = [TRIP_TYPE;TRIP_TYPE_CD];
299 clear TRIP_TYPE_CD
300 end
301
302 %%
303 %% CABINCLASS
304
305 disp(['Loading CABINCLASS_ser_Run_',num2str(run)])
306 load([save_dir,'\CABINCLASS_ser_Run_',num2str(run),'.mat'],'CABINCLASS_ser')
307
308 disp('Getting Array from Byte Stream')
309 CABINCLASS = getArrayFromByteStream(CABINCLASS_ser); clear CABINCLASS_ser
310 if run == 1
311
312 CABIN_CLASS = CABINCLASS;
313 clear CABINCLASS
314
315 else
316
317 CABIN_CLASS = [CABIN_CLASS;CABINCLASS];
318 clear CABINCLASS
319 end
320 %% SAVE
321
322 disp('Saving')
323 TICKET_NBR_ser = getByteStreamFromArray(TICKET_NBR);
324 ORIG_ARPT_CD_ser = getByteStreamFromArray(ORIG_ARPT_CD);
325 DEST_ARPT_CD_ser = getByteStreamFromArray(DEST_ARPT_CD);
326 SEG_ID_ser = getByteStreamFromArray(SEG_ID);
327 DOC_AMT_ser = getByteStreamFromArray(DOC_AMT);
328 SEG_DOC_AMT_ser = getByteStreamFromArray(SEG_DOC_AMT);
329 SEG_MILE_ser = getByteStreamFromArray(SEG_MILE);
330 Total_Mile_ser = getByteStreamFromArray(Total_Mile);
331 TRIP_TYPE_ser = getByteStreamFromArray(TRIP_TYPE);
332 CABIN_CLASS_ser = getByteStreamFromArray(CABIN_CLASS);
333
334 save([save_dir,'\TICKET_NBR_ser.mat'],'TICKET_NBR_ser','-v7.3')
335 save([save_dir,'\ORIG_ARPT_CD_ser.mat'],'ORIG_ARPT_CD_ser','-v7.3')
336 save([save_dir,'\DEST_ARPT_CD_ser.mat'],'DEST_ARPT_CD_ser','-v7.3')
337 save([save_dir,'\SEG_ID_ser.mat'],'SEG_ID_ser','-v7.3')
338 save([save_dir,'\DOC_AMT_ser.mat'],'DOC_AMT_ser','-v7.3')
339 save([save_dir,'\SEG_DOC_AMT_ser.mat'],'SEG_DOC_AMT_ser','-v7.3')
```

```
340 save([save_dir,'\SEG_MILE_ser.mat'],'SEG_MILE_ser','-v7.3')
341 save([save_dir,'\Total_Mile_ser.mat'],'Total_Mile_ser','-v7.3')
342 save([save_dir,'\TRIP_TYPE_ser.mat'],'TRIP_TYPE_ser','-v7.3')
343 save([save_dir,'\CABIN_CLASS_ser.mat'],'CABIN_CLASS_ser','-v7.3')
344
345
346 clear TICKET_NBR_ser ORIG_ARPT_CD_ser DEST_ARPT_CD_ser SEG_ID_ser...
347 DOC_AMT_ser SEG_DOC_AMT_ser SEG_MILE_ser Total_Mile_ser CABIN_CLASS_ser
348
349 end
350
351 return;
352
353
354
355
356
```

```
1 function [] = Remove_records(ARC_2016_Dir,main_delimiter)
2
3
4 %% Remove records with errors
5
6 load_dir = ([ARC_2016_Dir,main_delimiter,'Output',main_delimiter,'ARC_Data_Extracted']);
7 save_dir = ([ARC_2016_Dir,main_delimiter,'Output',main_delimiter,'ARC_Data_Filtered']);
8
9
10 %% Remove records of only "segment = 2" only
11 load([load_dir,'\TICKET_NBR_ser.mat'],'TICKET_NBR_ser')
12
13 load([load_dir,'\SEG_ID_ser.mat'],'SEG_ID_ser')
14
15 TICKET_NBR = getArrayFromByteStream(TICKET_NBR_ser); clear TICKET_NBR_ser
16 SEG_ID = getArrayFromByteStream(SEG_ID_ser); clear SEG_ID_ser
17
18 seg_2_index = SEG_ID == 2;
19 ticket_seg_2 = TICKET_NBR(seg_2_index);
20
21 [ticket_seg_2_index,~] = ismember(TICKET_NBR,ticket_seg_2);
22
23 tickets_related_to_seg_2 = TICKET_NBR(ticket_seg_2_index);
24
25 [unique_tickets_related_to_seg_2, ~, idx] = unique(tickets_related_to_seg_2);
26
27 count = histc(idx,unique(idx));
28
29 removed_index = count==1;
30
31 [remove_record_index,~] = ismember(TICKET_NBR,unique_tickets_related_to_seg_2(removed_index));
32
33 %% Remove records with distance discrepancy
34 load([load_dir,'\SEG_MILE_ser.mat'],'SEG_MILE_ser')
35 load([load_dir,'\Total_Mile_ser.mat'],'Total_Mile_ser')
36
37 SEG_MILE = getArrayFromByteStream(SEG_MILE_ser); clear SEG_MILE_ser
38 Total_Mile = getArrayFromByteStream(Total_Mile_ser); clear Total_Mile_ser
39
40
41 seg_1_index = SEG_ID == 1;
42
43 [remove_index,~] = ismember(seg_1_index,ticket_seg_2_index);
44
45 seg_1_index(remove_index) = [];
46
47 distance_discrepancy = Total_Mile(seg_1_index) - SEG_MILE(seg_1_index);
48
```

```
49 distance_discrepancy_index = distance_discrepancy ~= 0;
50
51 remove_records_due_to_distance_discrepancy = find(seg_1_index(distance_discrepancy_index));
52 remove_record_index = find(remove_record_index);
53 remove_final_index = [remove_record_index;remove_records_due_to_distance_discrepancy];
54 remove_final_index = unique(remove_final_index);
55
56 %% Load, reduce, and safe fields individually. Loading all files at once will reduce speed significantly due to file size.
57 %TICKET_NBR
58 TICKET_NBR(remove_final_index) = [];
59 TICKET_NBR_ser = getByteStreamFromArray(TICKET_NBR);
60 clear TICKET_NBR
61 save([save_dir,'\TICKET_NBR_ser.mat'],'TICKET_NBR_ser','-v7.3')
62 clear TICKET_NBR_ser
63
64 %SEG_ID
65 SEG_ID(remove_final_index) = [];
66 SEG_ID_ser = getByteStreamFromArray(SEG_ID);
67 clear SEG_ID
68 save([save_dir,'\SEG_ID_ser.mat'],'SEG_ID_ser','-v7.3')
69 clear SEG_ID_ser
70
71 %SEG_MILE
72 SEG_MILE(remove_final_index) = [];
73 SEG_MILE_ser = getByteStreamFromArray(SEG_MILE);
74 clear SEG_MILE
75 save([save_dir,'\SEG_MILE_ser.mat'],'SEG_MILE_ser','-v7.3')
76 clear SEG_MILE_ser
77
78 %Total_Mile
79 Total_Mile(remove_final_index) = [];
80 Total_Mile_ser = getByteStreamFromArray(Total_Mile);
81 clear Total_Mile
82 save([save_dir,'\Total_Mile_ser.mat'],'Total_Mile_ser','-v7.3')
83 clear Total_Mile_ser
84
85 %ORIG_ARPT_CD
86 load([load_dir,'\ORIG_ARPT_CD_ser.mat'],'ORIG_ARPT_CD_ser')
87 ORIG_ARPT_CD = getArrayFromByteStream(ORIG_ARPT_CD_ser);
88 clear ORIG_ARPT_CD_ser
89 ORIG_ARPT_CD(remove_final_index) = [];
90 ORIG_ARPT_CD_ser = getByteStreamFromArray(ORIG_ARPT_CD);
91 clear ORIG_ARPT_CD
92 save([save_dir,'\ORIG_ARPT_CD_ser.mat'],'ORIG_ARPT_CD_ser','-v7.3')
93 clear ORIG_ARPT_CD_ser
94
95 %DEST_ARPT_CD
96 load([load_dir,'\DEST_ARPT_CD_ser.mat'],'DEST_ARPT_CD_ser')
97 DEST_ARPT_CD = getArrayFromByteStream(DEST_ARPT_CD_ser);
98 clear DEST_ARPT_CD_ser
99 DEST_ARPT_CD(remove_final_index) = [];
100 DEST_ARPT_CD_ser = getByteStreamFromArray(DEST_ARPT_CD);
101 clear DEST_ARPT_CD
102 save([save_dir,'\DEST_ARPT_CD_ser.mat'],'DEST_ARPT_CD_ser','-v7.3')
103 clear DEST_ARPT_CD_ser
104
105 %DOC_AMT
106 load([load_dir,'\DOC_AMT_ser.mat'],'DOC_AMT_ser')
107 DOC_AMT = getArrayFromByteStream(DOC_AMT_ser);
108 clear DOC_AMT_ser
109 DOC_AMT(remove_final_index) = [];
110 DOC_AMT_ser = getByteStreamFromArray(DOC_AMT);
111 clear DOC_AMT
112 save([save_dir,'\DOC_AMT_ser.mat'],'DOC_AMT_ser','-v7.3')
113 clear DOC_AMT_ser
114
```

```
%SEG_DOC_AMT
load([load_dir,'\SEG_DOC_AMT_ser.mat'],'SEG_DOC_AMT_ser')
SEG_DOC_AMT = getArrayFromByteStream(SEG_DOC_AMT_ser);
clear SEG_DOC_AMT_ser
SEG_DOC_AMT(remove_final_index) = [];
SEG_DOC_AMT_ser = getByteStreamFromArray(SEG_DOC_AMT);
clear SEG_DOC_AMT
save([save_dir,'\SEG_DOC_AMT_ser.mat'],'SEG_DOC_AMT_ser','-v7.3')
clear SEG_DOC_AMT_ser

%TRIP_TYPE
load([load_dir,'\TRIP_TYPE_ser.mat'],'TRIP_TYPE_ser')
TRIP_TYPE = getArrayFromByteStream(TRIP_TYPE_ser);
clear TRIP_TYPE_ser
TRIP_TYPE(remove_final_index) = [];
TRIP_TYPE_ser = getByteStreamFromArray(TRIP_TYPE);
clear TRIP_TYPE
save([save_dir,'\TRIP_TYPE_ser.mat'],'TRIP_TYPE_ser','-v7.3')
clear TRIP_TYPE_ser

%Cabin_Class
load([load_dir,'\CABIN_CLASS_ser.mat'],'CABIN_CLASS_ser')
Cabin_Class = getArrayFromByteStream(CABIN_CLASS_ser);
clear CABIN_CLASS_ser
Cabin_Class(remove_final_index) = [];

[first_index,~] = ismember(Cabin_Class,'FIRST');
[business_index,~] = ismember(Cabin_Class,'BUSINESS');
CabinClass = zeros(length(Cabin_Class),1);
clear Cabin_Class
CabinClass(first_index) = 1; %1 = First
CabinClass(business_index) = 2; %2 = Business

CabinClass_ser = getByteStreamFromArray(CabinClass);
clear CabinClass
save([save_dir,'\CabinClass_ser.mat'],'CabinClass_ser','-v7.3')
clear CabinClass_ser
return;


```

```

function [] = Identify_Oneway_NonStop(ARC_2016_Dir,main_delimiter)
%% Separate One-way nonstop

load_dir = ([ARC_2016_Dir,main_delimiter,'Output',main_delimiter,'ARC_Data_Filtered']);
save_dir = ([ARC_2016_Dir,main_delimiter,'Output']);
load([load_dir,'\TICKET_NBR_ser.mat'],'TICKET_NBR_ser')
load([load_dir,'\SEG_ID_ser.mat'],'SEG_ID_ser')
load([load_dir,'\TRIP_TYPE_ser.mat'],'TRIP_TYPE_ser')

TICKET_NBR = getArrayFromByteStream(TICKET_NBR_ser); clear TICKET_NBR_ser
SEG_ID = getArrayFromByteStream(SEG_ID_ser); clear SEG_ID_ser
TRIP_TYPE = getArrayFromByteStream(TRIP_TYPE_ser); clear TRIP_TYPE_ser

%% OW Trip Type
%Index to all 'OW' trip type
OW_index = find(strcmp(TRIP_TYPE,'OW'));

%Index to OW_index with a SEG_ID = 2
OW_seg_2_index = SEG_ID(OW_index)==2;

%Index to all 'OW' trip type with SEG_ID = 2
oneway_onestop_index_seg_2 = OW_index(OW_seg_2_index);

%Index to all 'OW' one_stop trip
```

```
26 ticket_oneway_onestop_index = TICKET_NBR(oneway_onestop_index_seg_2);
27 unique_ticket_oneway_onestop_index = unique(ticket_oneway_onestop_index);
28
29 oneway_onestop_index = find(ismember(TICKET_NBR,unique_ticket_oneway_onestop_index));
30 remove = ismember(OW_index,oneway_onestop_index)==1;
31
32 %Remove all 'OW' indeces with SEG_ID = 2 related.
33 OW_index(remove) = [];
34 clear remove
35 %Index to all 'OW' trip type with SEG_ID = 1
36 oneway_nonstop_index = OW_index;
37
38 %%
39
40 Oneway_NonStop.TICKET_NBR = TICKET_NBR(oneway_nonstop_index);
41 Oneway_NonStop.SEG_ID = SEG_ID(oneway_nonstop_index);
42 Oneway_NonStop.TRIP_TYPE = TRIP_TYPE(oneway_nonstop_index);
43
44 clear TICKET_NBR SEG_ID TRIP_TYPE
45
46 %%
47
48 load([load_dir,'\SEG_MILE_ser.mat'],'SEG_MILE_ser')
49 SEG_MILE = getArrayFromByteStream(SEG_MILE_ser); clear SEG_MILE_ser
50
51 Oneway_NonStop.SEG_MILE = SEG_MILE(oneway_nonstop_index);
52 clear SEG_MILE
53 %%
54
55 load([load_dir,'\CabinClass_ser.mat'],'CabinClass_ser')
56
57 CabinClass = getArrayFromByteStream(CabinClass_ser); clear CabinClass_ser
58
59 Oneway_NonStop.CabinClass = CabinClass(oneway_nonstop_index);
60
61 clear CabinClass
62
63 %%
64
65 load([load_dir,'\Total_Mile_ser.mat'],'Total_Mile_ser')
66 Total_Mile = getArrayFromByteStream(Total_Mile_ser); clear Total_Mile_ser
67
68 Oneway_NonStop.Total_Mile = Total_Mile(oneway_nonstop_index);
69
70 clear Total_Mile
71
72 %%
73 load([load_dir,'\ORIG_ARPT_CD_ser.mat'],'ORIG_ARPT_CD_ser')
74 ORIG_ARPT_CD = getArrayFromByteStream(ORIG_ARPT_CD_ser); clear ORIG_ARPT_CD_ser
75
76 Oneway_NonStop.ORIG_ARPT_CD = ORIG_ARPT_CD(oneway_nonstop_index);
77
78 clear ORIG_ARPT_CD
79 %%
80 load([load_dir,'\DEST_ARPT_CD_ser.mat'],'DEST_ARPT_CD_ser')
81 DEST_ARPT_CD = getArrayFromByteStream(DEST_ARPT_CD_ser); clear DEST_ARPT_CD_ser
82
83 Oneway_NonStop.DEST_ARPT_CD = DEST_ARPT_CD(oneway_nonstop_index);
84
85 clear DEST_ARPT_CD
86 %%
87 load([load_dir,'\DOC_AMT_ser.mat'],'DOC_AMT_ser')
88 DOC_AMT = getArrayFromByteStream(DOC_AMT_ser); clear DOC_AMT_ser
89
90 Oneway_NonStop.DOC_AMT = DOC_AMT(oneway_nonstop_index);
91
```

```
92 clear DOC_AMT
93 %%
94 load([load_dir,'\SEG_DOC_AMT_ser.mat'],'SEG_DOC_AMT_ser')
95 SEG_DOC_AMT = getArrayFromByteStream(SEG_DOC_AMT_ser); clear SEG_DOC_AMT_ser
96
97 Oneway_NonStop.SEG_DOC_AMT = SEG_DOC_AMT(oneway_nonstop_index);
98
99 clear SEG_DOC_AMT
100
101 %% Fare per mile
102
103 %Oneway_nonstop
104 Oneway_NonStop.Fare_per_pax_mile = round((Oneway_NonStop.DOC_AMT ./ Oneway_NonStop.Total_Mile),2);
105
106 %%
107 %Oneway non stop
108 short_distance_index = Oneway_NonStop.SEG_MILE <1000;
109 ticekts_of_short_distance_records = Oneway_NonStop.TICKET_NBR(short_distance_index);
110 unique_ticekts_of_short_distance_records = unique(ticekts_of_short_distance_records);
111 [remove_record_index,~] = ismember(Oneway_NonStop.TICKET_NBR,unique_ticekts_of_short_distance_records);
112
113 Oneway_NonStop.TICKET_NBR(remove_record_index) = [];
114 Oneway_NonStop.SEG_ID(remove_record_index) = [];
115 Oneway_NonStop.SEG_MILE(remove_record_index) = [];
116 Oneway_NonStop.Total_Mile(remove_record_index) = [];
117 Oneway_NonStop.ORIG_ARPT_CD(remove_record_index) = [];
118 Oneway_NonStop.DEST_ARPT_CD(remove_record_index) = [];
119 Oneway_NonStop.DOC_AMT(remove_record_index) = [];
120 Oneway_NonStop.SEG_DOC_AMT(remove_record_index) = [];
121 Oneway_NonStop.Fare_per_pax_mile(remove_record_index) = [];
122
123 Oneway_Non_Stop = Oneway_NonStop;
124 clear Oneway_NonStop
125 %%
126 clearvars -except save_dir Oneway_Non_Stop
127 %% Remove fare per passenger mile greater than $3/hr.
128
129 high_fppm_index = Oneway_Non_Stop.Fare_per_pax_mile > 3;
130 tickets_high_fppm_index = Oneway_Non_Stop.TICKET_NBR(high_fppm_index);
131 unique_tickets_high_fppm_index = unique(tickets_high_fppm_index);
132 [remove_high_fppm_record_index,~] = ismember(Oneway_Non_Stop.TICKET_NBR,unique_tickets_high_fppm_index);
133
134 Oneway_Non_Stop.TICKET_NBR(remove_high_fppm_record_index) = [];
135 Oneway_Non_Stop.SEG_ID(remove_high_fppm_record_index) = [];
136 Oneway_Non_Stop.Total_Mile(remove_high_fppm_record_index) = [];
137 Oneway_Non_Stop.SEG_MILE(remove_high_fppm_record_index) = [];
138 Oneway_Non_Stop.ORIG_ARPT_CD(remove_high_fppm_record_index) = [];
139 Oneway_Non_Stop.DEST_ARPT_CD(remove_high_fppm_record_index) = [];
140 Oneway_Non_Stop.DOC_AMT(remove_high_fppm_record_index) = [];
141 Oneway_Non_Stop.SEG_DOC_AMT(remove_high_fppm_record_index) = [];
142 Oneway_Non_Stop.Fare_per_pax_mile(remove_high_fppm_record_index) = [];
143 Oneway_Non_Stop.DOC_AMT = round(Oneway_Non_Stop.DOC_AMT,2);
144 Oneway_Non_Stop.SEG_DOC_AMT = round(Oneway_Non_Stop.SEG_DOC_AMT,2);
145
146 %%
147 Oneway_Non_Stop_ser = getByteStreamFromArray(Oneway_Non_Stop);
148 save([save_dir,'\Oneway_Non_Stop_ser.mat'],'Oneway_Non_Stop_ser','-v7.3')
149
150 return;

1 function [] = Identify_Oneway_OneStop(ARC_2016_Dir,main_delimiter,Input_Folder_Dir)
2
3 load([Input_Folder_Dir,main_delimiter,'OD_Pairs_Year_2016.mat'],'OD_Pairs_Year_2016')
4
5 %% Separate One-way OneStop
6 load_dir = ([ARC_2016_Dir,main_delimiter,'Output',main_delimiter,'ARC_Data_Filtered']);
```

```
7 save_dir = ([ARC_2016_Dir,main_delimiter,'Output']);
8 load([load_dir,'\TICKET_NBR_ser.mat'],'TICKET_NBR_ser')
9 load([load_dir,'\SEG_ID_ser.mat'],'SEG_ID_ser')
10 load([load_dir,'\TRIP_TYPE_ser.mat'],'TRIP_TYPE_ser')
11
12 TICKET_NBR = getArrayFromByteStream(TICKET_NBR_ser); clear TICKET_NBR_ser
13 SEG_ID = getArrayFromByteStream(SEG_ID_ser); clear SEG_ID_ser
14 TRIP_TYPE = getArrayFromByteStream(TRIP_TYPE_ser); clear TRIP_TYPE_ser
15
16 %% OW Trip Type
17 %Index to all 'OW' trip type
18 OW_index = find(strcmp(TRIP_TYPE,'OW'));
19
20 %Index to OW_index with a SEG_ID = 2
21 OW_seg_2_index = SEG_ID(OW_index)==2;
22
23 %Index to all 'OW' trip type with SEG_ID = 2
24 oneway_onestop_index_seg_2 = OW_index(OW_seg_2_index);
25
26 %Index to all 'OW' one_stop trip
27 ticket_oneway_onestop_index = TICKET_NBR(oneway_onestop_index_seg_2);
28 unique_ticket_oneway_onestop_index = unique(ticket_oneway_onestop_index);
29 [oneway_onestop_index,~] = ismember(TICKET_NBR,unique_ticket_oneway_onestop_index);
30
31 %%
32
33 Oneway_OneStop.TICKET_NBR = TICKET_NBR(oneway_onestop_index);
34 Oneway_OneStop.SEG_ID = SEG_ID(oneway_onestop_index);
35
36 clear TICKET_NBR SEG_ID TRIP_TYPE
37
38 %%
39
40 load([load_dir,'\SEG_MILE_ser.mat'],'SEG_MILE_ser')
41 SEG_MILE = getArrayFromByteStream(SEG_MILE_ser); clear SEG_MILE_ser
42
43 Oneway_OneStop.SEG_MILE = SEG_MILE(oneway_onestop_index);
44
45 clear SEG_MILE
46 %%
47
48 load([load_dir,'\CabinClass_ser.mat'],'CabinClass_ser')
49
50 CabinClass = getArrayFromByteStream(CabinClass_ser); clear CabinClass_ser
51
52 Oneway_OneStop.CabinClass = CabinClass(oneway_onestop_index);
53
54 clear CabinClass
55 %%
56
57 load([load_dir,'\Total_Mile_ser.mat'],'Total_Mile_ser')
58 Total_Mile = getArrayFromByteStream(Total_Mile_ser); clear Total_Mile_ser
59
60 Oneway_OneStop.Total_Mile = Total_Mile(oneway_onestop_index);
61
62 clear Total_Mile
63 %%
64
65 load([load_dir,'\ORIG_ARPT_CD_ser.mat'],'ORIG_ARPT_CD_ser')
66 ORIG_ARPT_CD = getArrayFromByteStream(ORIG_ARPT_CD_ser); clear ORIG_ARPT_CD_ser
67
68 Oneway_OneStop.ORIG_ARPT_CD = ORIG_ARPT_CD(oneway_onestop_index);
69
70 clear ORIG_ARPT_CD
71 %%
72
```

```
73 load([load_dir,'\DEST_ARPT_CD_ser.mat'],'DEST_ARPT_CD_ser')
74 DEST_ARPT_CD = getArrayFromByteStream(DEST_ARPT_CD_ser); clear DEST_ARPT_CD_ser
75
76 Oneway_OneStop.DEST_ARPT_CD = DEST_ARPT_CD(oneway_onestop_index);
77
78 clear DEST_ARPT_CD
79 %%
80 load([load_dir,'\DOC_AMT_ser.mat'],'DOC_AMT_ser')
81 DOC_AMT = getArrayFromByteStream(DOC_AMT_ser); clear DOC_AMT_ser
82
83 Oneway_OneStop.DOC_AMT = DOC_AMT(oneway_onestop_index);
84
85 clear DOC_AMT
86 %%
87 load([load_dir,'\SEG_DOC_AMT_ser.mat'],'SEG_DOC_AMT_ser')
88 SEG_DOC_AMT = getArrayFromByteStream(SEG_DOC_AMT_ser); clear SEG_DOC_AMT_ser
89
90 Oneway_OneStop.SEG_DOC_AMT = SEG_DOC_AMT(oneway_onestop_index);
91
92 clear SEG_DOC_AMT
93 %% Remove false records
94
95 unique_tickets = unique(Oneway_OneStop.TICKET_NBR);
96 length_unique_tickets = length(unique_tickets);
97 check = zeros(length_unique_tickets,1);
98
99 for ticket = 1:length_unique_tickets
100
101 if mod(ticket, 1000) == 0
102 disp([num2str(ticket), '/', num2str(length_unique_tickets)]);
103
104 end
105
106
107 match_index = find(Oneway_OneStop.TICKET_NBR==unique_tickets(ticket));
108 seg_1 = Oneway_OneStop.SEG_ID(match_index) == 1;
109 seg_2 = Oneway_OneStop.SEG_ID(match_index) == 2;
110 check(ticket,1) =
ismember(Oneway_OneStop.DEST_ARPT_CD(match_index(seg_1)),Oneway_OneStop.ORIG_ARPT_CD(match_index(seg_2)));
111 end
112
113 false_record_index = check==0;
114 false_record_index_uniue_ticket = unique_tickets(false_record_index);
115 [remove_index,~] = ismember(Oneway_OneStop.TICKET_NBR,false_record_index_uniue_ticket);
116
117 Oneway_OneStop.TICKET_NBR(remove_index) = [];
118 Oneway_OneStop.SEG_ID(remove_index) = [];
119 Oneway_OneStop.SEG_MILE(remove_index) = [];
120 Oneway_OneStop.Total_Mile(remove_index) = [];
121 Oneway_OneStop.ORIG_ARPT_CD(remove_index) = [];
122 Oneway_OneStop.DEST_ARPT_CD(remove_index) = [];
123 Oneway_OneStop.DOC_AMT(remove_index) = [];
124 Oneway_OneStop.SEG_DOC_AMT(remove_index) = [];
125
126 %% Calculate fare per mile based on origin-destination distance instead of origin-stopover-destination distance
127 oag_od = strcat(OD_Pairs_Year_2016.DepAirport,'_',OD_Pairs_Year_2016.ArrAirport);
128 oag_dist = OD_Pairs_Year_2016.DistStMiles;
129 clear OD_Pairs_Year_2016
130
131 unique_tickets = unique(Oneway_OneStop.TICKET_NBR);
132 length_unique_tickets = length(unique_tickets);
133
134 for ticket = 1:length_unique_tickets
135
136 if mod(ticket, 1000) == 0
137 disp([num2str(ticket), '/', num2str(length_unique_tickets)]);
```

```
end
match_index = find(Oneway_OneStop.TICKET_NBR==unique_tickets(ticket));
seg_1 = Oneway_OneStop.SEG_ID(match_index) == 1;
seg_2 = Oneway_OneStop.SEG_ID(match_index) == 2;
arc_od = strcat(Oneway_OneStop.ORIG_ARPT_CD(match_index(seg_1)),'_',Oneway_OneStop.DEST_ARPT_CD(match_index(seg_2)));
[oag_idst_index,~] = ismember(oag_od,arc_od);
if sum(oag_idst_index) ~= 0
Oneway_OneStop.Fare_per_pax_mile(match_index,1) = round(Oneway_OneStop.DOC_AMT(match_index(seg_1)) ./
oag_dist(oag_idst_index),2);
else
Oneway_OneStop.Fare_per_pax_mile(match_index,1) = round((Oneway_OneStop.DOC_AMT(match_index) ./
Oneway_OneStop.Total_Mile
(match_index)),2);
end
end
%% Remove 0 mile distance records
short_distance_index = Oneway_OneStop.SEG_MILE==0;
ticekts_of_short_distance_records = Oneway_OneStop.TICKET_NBR(short_distance_index);
unique_ticekts_of_short_distance_records = unique(ticekts_of_short_distance_records);
[remove_record_index,~] = ismember(Oneway_OneStop.TICKET_NBR,unique_ticekts_of_short_distance_records);
Oneway_OneStop.TICKET_NBR(remove_record_index) = [];
Oneway_OneStop.SEG_ID(remove_record_index) = [];
Oneway_OneStop.SEG_MILE(remove_record_index) = [];
Oneway_OneStop.Total_Mile(remove_record_index) = [];
Oneway_OneStop.ORIG_ARPT_CD(remove_record_index) = [];
Oneway_OneStop.DEST_ARPT_CD(remove_record_index) = [];
Oneway_OneStop.DOC_AMT(remove_record_index) = [];
Oneway_OneStop.SEG_DOC_AMT(remove_record_index) = [];
Oneway_OneStop.Fare_per_pax_mile(remove_record_index) = [];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%%%%%%%%%%%%%%%%%
clearvars -except save_dir Oneway_OneStop
%% Remove fare per passenger mile greater than $3/hr.
remove_record_index = find(Oneway_OneStop.Fare_per_pax_mile > 3);
Oneway_OneStop.TICKET_NBR(remove_record_index) = [];
Oneway_OneStop.SEG_ID(remove_record_index) = [];
Oneway_OneStop.SEG_MILE(remove_record_index) = [];
Oneway_OneStop.Total_Mile(remove_record_index) = [];
Oneway_OneStop.ORIG_ARPT_CD(remove_record_index) = [];
Oneway_OneStop.DEST_ARPT_CD(remove_record_index) = [];
Oneway_OneStop.DOC_AMT(remove_record_index) = [];
Oneway_OneStop.SEG_DOC_AMT(remove_record_index) = [];
Oneway_OneStop.Fare_per_pax_mile(remove_record_index) = [];
clear remove_record_index
Oneway_OneStop_ser = getByteStreamFromArray(Oneway_OneStop);
save([save_dir,'\Oneway_OneStop_ser.mat'],'Oneway_OneStop_ser','-v7.3')
```

```
198 return;
199

1 function [] = Identify_RoundTrip_NonStop(ARC_2016_Dir,main_delimiter)
2
3 %% Separate One-way OneStop
4 load_dir = ([ARC_2016_Dir,main_delimiter,'Output',main_delimiter,'ARC_Data_Filtered']);
5 save_dir = ([ARC_2016_Dir,main_delimiter,'Output']);
6 load([load_dir,'\TICKET_NBR_ser.mat'],'TICKET_NBR_ser')
7 load([load_dir,'\SEG_ID_ser.mat'],'SEG_ID_ser')
8 load([load_dir,'\TRIP_TYPE_ser.mat'],'TRIP_TYPE_ser')
9
10 TICKET_NBR = getArrayFromByteStream(TICKET_NBR_ser); clear TICKET_NBR_ser
11 SEG_ID = getArrayFromByteStream(SEG_ID_ser); clear SEG_ID_ser
12 TRIP_TYPE = getArrayFromByteStream(TRIP_TYPE_ser); clear TRIP_TYPE_ser
13
14 %% RT Trip Type
15
16 %Index to all 'RT' trip type
17 round_trip_index = find(strcmp(TRIP_TYPE,'RT'));
18
19 %%
20
21 Roundtrip_NonStop.TICKET_NBR = TICKET_NBR(round_trip_index);
22 Roundtrip_NonStop.SEG_ID = SEG_ID(round_trip_index);
23 Roundtrip_NonStop.TRIP_TYPE = TRIP_TYPE(round_trip_index);
24 clear TICKET_NBR SEG_ID TRIP_TYPE
25
26 %%
27 load([load_dir,'\SEG_MILE_ser.mat]','SEG_MILE_ser')
28 SEG_MILE = getArrayFromByteStream(SEG_MILE_ser); clear SEG_MILE_ser
29
30 Roundtrip_NonStop.SEG_MILE = SEG_MILE(round_trip_index);
31 clear SEG_MILE
32 %%
33 load([load_dir,'\CabinClass_ser.mat'],'CabinClass_ser')
34
35 CabinClass = getArrayFromByteStream(CabinClass_ser); clear CabinClass_ser
36
37 Roundtrip_NonStop.CabinClass = CabinClass(round_trip_index);
38
39 clear CabinClass
40
41 %%
42 load([load_dir,'\Total_Mile_ser.mat'],'Total_Mile_ser')
43
44 Total_Mile = getArrayFromByteStream(Total_Mile_ser); clear Total_Mile_ser
45
46 Roundtrip_NonStop.Total_Mile = Total_Mile(round_trip_index);
47 clear Total_Mile
48
49 %%
50 load([load_dir,'\ORIG_ARPT_CD_ser.mat'],'ORIG_ARPT_CD_ser')
51
52 ORIG_ARPT_CD = getArrayFromByteStream(ORIG_ARPT_CD_ser); clear ORIG_ARPT_CD_ser
53
54 Roundtrip_NonStop.ORIG_ARPT_CD = ORIG_ARPT_CD(round_trip_index);
55 clear ORIG_ARPT_CD
56 %%
57
58 load([load_dir,'\DEST_ARPT_CD_ser.mat'],'DEST_ARPT_CD_ser')
59
60 DEST_ARPT_CD = getArrayFromByteStream(DEST_ARPT_CD_ser); clear DEST_ARPT_CD_ser
61
62 Roundtrip_NonStop.DEST_ARPT_CD = DEST_ARPT_CD(round_trip_index);
63 clear DEST_ARPT_CD
```

```
64 %%
65 load([load_dir,'\DOC_AMT_ser.mat'],'DOC_AMT_ser')
66 DOC_AMT = getArrayFromByteStream(DOC_AMT_ser); clear DOC_AMT_ser
67
68 Roundtrip_NonStop.DOC_AMT = DOC_AMT(round_trip_index);
69 clear DOC_AMT
70 %%
71 load([load_dir,'\SEG_DOC_AMT_ser.mat'],'SEG_DOC_AMT_ser')
72
73 SEG_DOC_AMT = getArrayFromByteStream(SEG_DOC_AMT_ser); clear SEG_DOC_AMT_ser
74
75 Roundtrip_NonStop.SEG_DOC_AMT = SEG_DOC_AMT(round_trip_index);
76
77 clear SEG_DOC_AMT
78
79 %% Fare per mile
80
81 Roundtrip_NonStop.Fare_per_pax_mile = round((Roundtrip_NonStop.DOC_AMT ./ Roundtrip_NonStop.Total_Mile),2);
82
83 %% Remove 0 mile distance records
84
85 %Roundtrip
86 short_distance_index = Roundtrip_NonStop.SEG_MILE <1000;
87 ticekts_of_short_distance_records = Roundtrip_NonStop.TICKET_NBR(short_distance_index);
88 unique_ticekts_of_short_distance_records = unique(ticekts_of_short_distance_records);
89 [remove_record_index,~] = ismember(Roundtrip_NonStop.TICKET_NBR,unique_ticekts_of_short_distance_records);
90
91 Roundtrip_NonStop.TICKET_NBR(remove_record_index) = [];
92 Roundtrip_NonStop.SEG_ID(remove_record_index) = [];
93 Roundtrip_NonStop.SEG_MILE(remove_record_index) = [];
94 Roundtrip_NonStop.Total_Mile(remove_record_index) = [];
95 Roundtrip_NonStop.ORIG_ARPT_CD(remove_record_index) = [];
96 Roundtrip_NonStop.DEST_ARPT_CD(remove_record_index) = [];
97 Roundtrip_NonStop.DOC_AMT(remove_record_index) = [];
98 Roundtrip_NonStop.SEG_DOC_AMT(remove_record_index) = [];
99 Roundtrip_NonStop.Fare_per_pax_mile(remove_record_index) = [];
100 clear short_distance_index ticekts_of_short_distance_records remove_record_index
101
102 %%
103 clearvars -except save_dir Roundtrip_NonStop main_delimiter
104 %% Remove fare per passenger mile greater than $3/hr.
105
106 high_fppm_index = Roundtrip_NonStop.Fare_per_pax_mile > 3;
107 tickets_high_fppm_index = Roundtrip_NonStop.TICKET_NBR(high_fppm_index);
108 unique_tickets_high_fppm_index = unique(tickets_high_fppm_index);
109 [remove_high_fppm_record_index,~] = ismember(Roundtrip_NonStop.TICKET_NBR,unique_tickets_high_fppm_index);
110
111 Roundtrip_NonStop.TICKET_NBR(remove_high_fppm_record_index) = [];
112 Roundtrip_NonStop.SEG_ID(remove_high_fppm_record_index) = [];
113 Roundtrip_NonStop.Total_Mile(remove_high_fppm_record_index) = [];
114 Roundtrip_NonStop.SEG_MILE(remove_high_fppm_record_index) = [];
115 Roundtrip_NonStop.ORIG_ARPT_CD(remove_high_fppm_record_index) = [];
116 Roundtrip_NonStop.DEST_ARPT_CD(remove_high_fppm_record_index) = [];
117 Roundtrip_NonStop.DOC_AMT(remove_high_fppm_record_index) = [];
118 Roundtrip_NonStop.SEG_DOC_AMT(remove_high_fppm_record_index) = [];
119 Roundtrip_NonStop.Fare_per_pax_mile(remove_high_fppm_record_index) = [];
120
121 Roundtrip_NonStop.DOC_AMT = round(Roundtrip_NonStop.DOC_AMT,2);
122 Roundtrip_NonStop.SEG_DOC_AMT = round(Roundtrip_NonStop.SEG_DOC_AMT,2);
123
124 %%
125
126 fare_delta = Roundtrip_NonStop.SEG_DOC_AMT .*2 - Roundtrip_NonStop.DOC_AMT;
127 fare_delta = round(fare_delta,2);
128 remove_index = fare_delta > 0.01;
129 ticket_remove_index = Roundtrip_NonStop.TICKET_NBR(remove_index);
```

```
130 unique_ticket_remove_index = unique(ticket_remove_index);
131 [remove_index_diff_delta_fare,~] = ismember(Roundtrip_NonStop.TICKET_NBR,unique_ticket_remove_index);
132
133 Roundtrip_NonStop.TICKET_NBR(remove_index_diff_delta_fare) = [];
134 Roundtrip_NonStop.SEG_ID(remove_index_diff_delta_fare) = [];
135 Roundtrip_NonStop.Total_Mile(remove_index_diff_delta_fare) = [];
136 Roundtrip_NonStop.SEG_MILE(remove_index_diff_delta_fare) = [];
137 Roundtrip_NonStop.ORIG_ARPT_CD(remove_index_diff_delta_fare) = [];
138
139 Roundtrip_NonStop.DEST_ARPT_CD(remove_index_diff_delta_fare) = [];
140 Roundtrip_NonStop.DOC_AMT(remove_index_diff_delta_fare) = [];
141 Roundtrip_NonStop.SEG_DOC_AMT(remove_index_diff_delta_fare) = [];
142 Roundtrip_NonStop.Fare_per_pax_mile(remove_index_diff_delta_fare) = [];
143
144 Roundtrip_NonStop_Part_1.TICKET_NBR = Roundtrip_NonStop.TICKET_NBR;
145 Roundtrip_NonStop_Part_1.SEG_ID = Roundtrip_NonStop.SEG_ID;
146 Roundtrip_NonStop_Part_1.Total_Mile = Roundtrip_NonStop.Total_Mile;
147 Roundtrip_NonStop_Part_1.SEG_MILE = Roundtrip_NonStop.SEG_MILE;
148 Roundtrip_NonStop_Part_1.ORIG_ARPT_CD = Roundtrip_NonStop.ORIG_ARPT_CD;
149
150 Roundtrip_NonStop_Part_2.DEST_ARPT_CD = Roundtrip_NonStop.DEST_ARPT_CD;
151 Roundtrip_NonStop_Part_2.DOC_AMT = Roundtrip_NonStop.DOC_AMT;
152 Roundtrip_NonStop_Part_2.SEG_DOC_AMT = Roundtrip_NonStop.SEG_DOC_AMT;
153 Roundtrip_NonStop_Part_2.Fare_per_pax_mile = Roundtrip_NonStop.Fare_per_pax_mile;
154
155 clear Roundtrip_NonStop
156 %%
157
158 Roundtrip_NonStop_Part_1_ser = getByteStreamFromArray(Roundtrip_NonStop_Part_1);
159 clear Roundtrip_NonStop_Part_1
160
161 save([save_dir,main_delimiter,'Roundtrip_NonStop_Part_1_ser.mat'],'Roundtrip_NonStop_Part_1_ser','-v7.3')
162
163 Roundtrip_NonStop_Part_2_ser = getByteStreamFromArray(Roundtrip_NonStop_Part_2);
164 clear Roundtrip_NonStop_Part_2
165
166 save([save_dir,main_delimiter,'Roundtrip_NonStop_Part_2_ser.mat'],'Roundtrip_NonStop_Part_2_ser','-v7.3')
167
168 return;
```

Pre_Processing_Main_Script

```
1 %% This script uses ARC 2016 parsed data and travel times for SST Pre-Processing
2
3 clear
4 clc
5
6 Run_Only_Travel_Time_Module = 1; %Yes=1, No=0
7
8 local_disc = '';
9 main_delimiter = '\';
10 main_file_name = '.';
11 Input_Folder_Dir = ([local_disc,'..\..\SST_2020_Input']);
12 SST_Pre_Processing_Dir = ([local_disc,main_file_name,main_delimiter,'SST_Pre_Processing']);
13
14 addpath(genpath('.\'));
15 addpath(genpath(Input_Folder_Dir));
16 acft_seating_capacity = 43;
17
18 SST_Travel_Times_Main_Function(SST_Pre_Processing_Dir) %open main function to change parameters
19
20 %%
21 if Run_Only_Travel_Time_Module == 0
22 %% VOT Pre-Processing
23 disp('US')
```

```
24 US_VOT_Main_Function(SST_Pre_Processing_Dir, Input_Folder_Dir)
25 disp('US_INT')
26 US_Int_VOT_Main_Function(SST_Pre_Processing_Dir, Input_Folder_Dir)
27 disp('INT')
28 Int_VOT_Main_Function(SST_Pre_Processing_Dir, Input_Folder_Dir)
29
30 %% CDF Pre-Processing
31
32 %Indicates the mach combination.
33 mach_overland = 1.7;
34 mach_overwater = 1.8;
35
36 Fare_per_Mile_CDF_Data_Main_Function_US(SST_Pre_Processing_Dir,mach_overland, mach_overwater,acft_seating_capacity)
37
38 Fare_per_Mile_CDF_Data_Main_Function_US_Int(SST_Pre_Processing_Dir,mach_overland, mach_overwater,acft_seating_capacity)
39
40 Fare_per_Mile_CDF_Data_Main_Function_Int(SST_Pre_Processing_Dir,mach_overland, mach_overwater,acft_seating_capacity)
41
42 end
```

```
1 function [] = SST_Travel_Times_Main_Function(SST_Pre_Processing_Dir)
2 %% This script uses the calculated travel times from the flight planner
3 % and adjust them to account for additional travel times such as taxi-in
4 % taxi-out, climb, descent, and layover.
5
6 local_disc = '';
7 main_delimiter = '\';
8 Input_Folder_Dir = ([local_disc,'..\..\SST_2020_Input']);
9 SST_Travel_Times_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'SST_Travel_Times']);
10 save_dir = ([SST_Travel_Times_Dir,main_delimiter,'Output']);
11
12 %Create Output directory for SST_Travel_Times folder
13 if exist((save_dir),'dir') == 0
14
15 mkdir(([SST_Travel_Times_Dir,main_delimiter,'Output']))
16
17 end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0
18
19 addpath(([SST_Travel_Times_Dir,main_delimiter,'Output']))
20
21 %% Parameters
22 %Mach speed to be analyzed.
23 overland_speed = [7;75;8]; %7=1.7, 75=1.75, 8=1.8
24 overwater_speed = [8;85;9]; %8=1.8, 85=1.85, 9=1.9
25
26 min_taxi_in_hrs = 0.0667; %4 minutes - if no data is available to estimate a specific taxi-in time; a minimum value is assigned
27 min_taxi_out_hrs = 0.1833; %11 minutes - if no data is available to estimate a specific taxi-out time; a minimum value is assigned
28
29 climb_time_hrs = 0.1; %6 minutes - adjustment for climb
30 descent_time_hrs = 0.3833; %23 minutes - adjustment for descent
31
32 layover_time_hrs = 1; %1hr. - layover time assigned for refuling if trip distance exceeds aircraft range
33 aircraft_range_overland_statute_mile = 3639.9; % 3,000nm. = 3,452sm %2,800nm = 3,222sm.; 3,200nm = 3682sm., %3,163nm. = 3,639.9sm
34 aircraft_range_overwater_statute_mile = 4142.8; %3,800nm = 4373sm. %3,600nm. = 4,142.8sm
35
36 %% Run Analysys
37
38 SST_Travel_Time_Adjustment(Input_Folder_Dir, main_delimiter, save_dir, overland_speed, overwater_speed, min_taxi_in_hrs,
min_taxi_out_hrs,
climb_time_hrs, descent_time_hrs, layover_time_hrs, aircraft_range_overland_statute_mile, aircraft_range_overwater_statute_mile)
39
40
41 return;
```

```
1
2 function [] = SST_Travel_Time_Adjustment(Input_Folder_Dir, main_delimiter, save_dir, overland_speed, overwater_speed, min_taxi_in_hrs,
```

```
min_taxi_out_hrs, climb_time_hrs, descent_time_hrs, layover_time_hrs, aircraft_range_overland_statute_mile,
aircraft_range_overwater_statute_mile)
3
4
5 load(([Input_Folder_Dir,main_delimiter,'OD_Pairs_Year_2016.mat']),'OD_Pairs_Year_2016')
6 OAG_Data = OD_Pairs_Year_2016;
7 clear OD_Pairs_Year_2016
8
9 % Generate airport list
10
11 airport_list = ([OAG_Data.DepAirport; OAG_Data.ArrAirport]);
12 unique_airport_list = unique(airport_list);
13
14 length_of_unique_airport_list = length(unique_airport_list);
15
16 airport_operations = zeros(length_of_unique_airport_list,1);
17
18 [~, airport_index] = ismember(OAG_Data.DepAirport, unique_airport_list);
19
20 for airport = 1:length_of_unique_airport_list
21 airport_operations(airport,1) = sum(OAG_Data.Total_Frequency(airport_index == airport));
22 end
23
24 for airport = 1:length_of_unique_airport_list
25
26 Taxi_Times.Airport(airport,1) = unique_airport_list(airport);
27
28 Taxi_Times.Taxi_Out(airport,1) = (0.000024256292906 * airport_operations(airport) + 11.481999999999999) /60;
29 Taxi_Times.Taxi_In(airport,1) = (0.000016906651376 * airport_operations(airport) + 4.428700000000000) / 60;
30
31 end
32
33 length_overland_speed = length(overland_speed);
34 length_overwater_speed = length(overwater_speed);
35
36 for overland_array_index = 1:length_overland_speed
37
38 for overwater_array_index = 1:length_overwater_speed
39
40 overland = overland_speed(overland_array_index);
41 overwater = overwater_speed(overwater_array_index);
42
43
load(([Input_Folder_Dir,main_delimiter,'Travel_Time_Data',main_delimiter,'routings_all_4000_1_',num2str(overland),'M_1_',num2str(overwat
er),'M.
mat']),'routings_all')
44 load(([Input_Folder_Dir,main_delimiter,'Travel_Time_Data',main_delimiter,'WindResults_all_4000_1_',num2str(overland),'M_1_',num2str
(overwater),'M.mat']),'WindResults_all')
45
46 %%
47
48 length_of_data = length(routings_all);
49
50 for od = 1:length_of_data
51
52 Travel_Times.DepAirport(od,1) = cellstr(routings_all(od).etms_departure_airport);
53
54 Travel_Times.ArrAirport(od,1) = cellstr(routings_all(od).etms_arrival_airport);
55
56 end
57
58 TT_Dep_Arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);
59
60 OAG_Dep_Arr_All = strcat(OAG_Data.DepAirport,'_',OAG_Data.ArrAirport);
61
62 [~,TT_OAG_Match_index] = ismember(TT_Dep_Arr,OAG_Dep_Arr_All);
```

```
63
64 Travel_Times.Distance = OAG_Data.DistStMiles(TT_OAG_Match_index);
65
66 for od = 1:length_of_data
67
68 length_of_time = length(WindResults_all(od).level(2).timeperiod.Time);
69
70 Travel_Times.Travel_Time(od,1) = WindResults_all(od).level(2).timeperiod.Time(length_of_time) ./ 3600;
71
72 end
73
74
75 %%
76 mach_overland = 1 + overland_speed(overland_array_index)/10;
77
78 [~, dep_airport_index] = ismember(Travel_Times.DepAirport,Taxi_Times.Airport);
79 [~, arr_airport_index] = ismember(Travel_Times.ArrAirport,Taxi_Times.Airport);
80
81 for route = 1:length_of_data
82
83 %%
84
85 length_od_points = length(WindResults_all(route).level(1).timeperiod.Mach);
86 dist = WindResults_all(route).level(1).timeperiod.Dist;
87
88 if length_od_points == 1
89
90 Travel_Times.Overland_Percent(route,1) = 1;
91
92 elseif overland ~= overwater
93
94 dist_between_pts = zeros(length_od_points-1,1);
95 for dist_point = 1:(length_od_points-1)
96
97 dist_between_pts(dist_point,1) = dist((dist_point+1)) - dist(dist_point);
98 end
99
100 mach = WindResults_all(route).level(1).timeperiod.Mach;
101 mach(length_od_points,1) = 0;
102
103 overland_index = mach == mach_overland;
104 overland_total_dist = sum(dist_between_pts(overland_index));
105
106
107 Travel_Times.Overland_Percent(route,1) = round((overland_total_dist ./ dist(length_od_points,1)),2);
108 clear dist_between_pts
109
110 else
111
112 dist_between_pts = zeros(length_od_points-1,1);
113
114 for dist_point = 1:(length_od_points-1)
115
116 dist_between_pts(dist_point,1) = dist((dist_point+1)) - dist(dist_point);
117 end
118
119 overland_indicator = WindResults_all(route).level(1).timeperiod.Overland;
120 overland_indicator(length_od_points,1) = 0;
121
122 overland_index = overland_indicator == 1;
123 overland_total_dist = sum(dist_between_pts(overland_index));
124
125
126 Travel_Times.Overland_Percent(route,1) = round((overland_total_dist ./ dist(length_od_points,1)),2);
127 clear dist_between_pts
128
```

```
129 end
130
131 %%
132 if Travel_Times.Overland_Percent(route,1) > 0.25
133
134 aircraft_range = aircraft_range_overland_statute_mile;
135
136 else
137
138 aircraft_range = aircraft_range_overwater_statute_mile;
139 end
140
141 if dep_airport_index(route) ~= 0
142
143 taxi_out_time_hrs = Taxi_Times.Taxi_Out(dep_airport_index(route));
144 else
145 taxi_out_time_hrs = min_taxi_out_hrs;
146 end
147
148
149 if arr_airport_index(route) ~= 0
150
151 taxi_in_time_hrs = Taxi_Times.Taxi_In(arr_airport_index(route));
152 else
153 taxi_in_time_hrs = min_taxi_in_hrs;
154 end
155
156
157 if Travel_Times.Distance(route)<= aircraft_range
158
159 Travel_Times.Travel_Time(route) = Travel_Times.Travel_Time(route) + taxi_out_time_hrs + climb_time_hrs + descent_time_hrs +
taxi_in_time_hrs;
160
161 else
162
163 Travel_Times.Travel_Time(route) = Travel_Times.Travel_Time(route) + (taxi_out_time_hrs + climb_time_hrs + descent_time_hrs +
taxi_in_time_hrs) * ceil(Travel_Times.Distance(route) / aircraft_range) + layover_time_hrs * (ceil(Travel_Times.Distance(route) /
aircraft_range) - 1);
164
165 end
166
167 end
168 overland_range_nm = ceil(aircraft_range_overland_statute_mile * 0.868976);
169 save(([save_dir,main_delimiter,'Travel_Times_with_Ground_Times_OverlandRange_',num2str(overland_range_nm),'_nm_1_',num2str
(overland),'M_1_',num2str(overwater),'M.mat']),'Travel_Times')
170
171 clearvars -except Input_Folder_Dir main_delimiter save_dir overland_speed overwater_speed min_taxi_in_hrs min_taxi_out_hrs
climb_time_hrs
descent_time_hrs layover_time_hrs aircraft_range_overland_statute_mile aircraft_range_overwater_statute_mile OAG_Data
overland_array_index
overwater_array_index Taxi_Times acft_seating_capacity length_overland_speed length_overwater_speed
172 end
173 end
174
175 return;
176
177
```

```
1 function [] = US_VOT_Main_Function(SST_Pre_Processing_Dir, Input_Folder_Dir)
2 %% This script calculates the value of time for the US market
3
4 main_delimiter = '\';
5 % Input_Folder_Dir = ([local_disc,'..\..\..\..\..\..\SST_2020_Input']);
6 SST_US_VOT_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'VOT\US_Market']);
7 save_dir = ([SST_US_VOT_Dir,main_delimiter,'Output']);
8 Upload_Dir = ('.\ARC_2016\Output');
```

```
%Create Output directory for SST_Travel_Times folder
if exist((save_dir),'dir') == 0

mkdir(([SST_US_VOT_Dir,main_delimiter,'Output']))

end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0

addpath(([SST_US_VOT_Dir,main_delimiter,'Output']))

load(([Input_Folder_Dir,main_delimiter,'airport_list.mat']),'airport_list')

Oneway_Non_Stop_Records_US(save_dir, Upload_Dir, main_delimiter,airport_list)

Rountrip_Non_Stop_Records_US(save_dir, Upload_Dir, main_delimiter,airport_list)

Oneway_Onestop_Records_US(save_dir, Upload_Dir, main_delimiter,airport_list)

Value_of_Time_Calculation_US(save_dir, main_delimiter,Input_Folder_Dir,airport_list)

return;

function [] = Oneway_Non_Stop_Records_US(save_dir, Upload_Dir, main_delimiter,airport_list)

load(([Upload_Dir,main_delimiter,'Oneway_Non_Stop_ser.mat']),'Oneway_Non_Stop_ser')
Oneway_Non_Stop = getArrayFromByteStream(Oneway_Non_Stop_ser);
clear Oneway_Non_Stop_ser

%%

%Remove the airports that are not part of both data set (ARC & OAG 2016).

missing_origin_airports_index = ismember(Oneway_Non_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs) == 0;

tickets = Oneway_Non_Stop.TICKET_NBR(missing_origin_airports_index);

unqiue_unique = unique(tickets);

remove = ismember(Oneway_Non_Stop.TICKET_NBR,unqiue_unique) ~= 0;

Oneway_Non_Stop.TICKET_NBR(remove) = [];
Oneway_Non_Stop.SEG_ID(remove) = [];
Oneway_Non_Stop.Total_Mile(remove) = [];
Oneway_Non_Stop.SEG_MILE(remove) = [];
Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
Oneway_Non_Stop.DOC_AMT(remove) = [];
Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
%
missing_dest_airports_index = ismember(Oneway_Non_Stop.DEST_ARPT_CD,airport_list.Airport_IDs) == 0;

tickets = Oneway_Non_Stop.TICKET_NBR(missing_dest_airports_index);

unqiue_unique = unique(tickets);

remove = ismember(Oneway_Non_Stop.TICKET_NBR,unqiue_unique) ~= 0;

Oneway_Non_Stop.TICKET_NBR(remove) = [];
Oneway_Non_Stop.SEG_ID(remove) = [];
Oneway_Non_Stop.Total_Mile(remove) = [];
Oneway_Non_Stop.SEG_MILE(remove) = [];
Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
Oneway_Non_Stop.DOC_AMT(remove) = [];
Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
```

```
45 Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
46
47 %%
48
49 %We keep only those records within the continental US.
50 [~,origin_airport_country_index] = ismember(Oneway_Non_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs);
51 [~,dest_airport_country_index] = ismember(Oneway_Non_Stop.DEST_ARPT_CD,airport_list.Airport_IDs);
52
53 ORIG_ARPT_country = airport_list.Country_IDs(origin_airport_country_index);
54 DEST_ARPT_country = airport_list.Country_IDs(dest_airport_country_index);
55
56 OD_ARPT_country = strcat(ORIG_ARPT_country,'_',DEST_ARPT_country);
57
58 non_US_OD_index = ismember(OD_ARPT_country,'US_US') == 0;
59
60 Oneway_Non_Stop.TICKET_NBR(non_US_OD_index) = [];
61 Oneway_Non_Stop.SEG_ID(non_US_OD_index) = [];
62 Oneway_Non_Stop.Total_Mile(non_US_OD_index) = [];
63 Oneway_Non_Stop.SEG_MILE(non_US_OD_index) = [];
64 Oneway_Non_Stop.ORIG_ARPT_CD(non_US_OD_index) = [];
65 Oneway_Non_Stop.DEST_ARPT_CD(non_US_OD_index) = [];
66 Oneway_Non_Stop.DOC_AMT(non_US_OD_index) = [];
67 Oneway_Non_Stop.SEG_DOC_AMT(non_US_OD_index) = [];
68 Oneway_Non_Stop.Fare_per_pax_mile(non_US_OD_index) = [];
69
70 %only for US, remove records with a fare per mile greater than
71 %$1.50/statute mile
72 remove_index = find(Oneway_Non_Stop.Fare_per_pax_mile > 1.5);
73
74 Oneway_Non_Stop.TICKET_NBR(remove_index) = [];
75 Oneway_Non_Stop.SEG_ID(remove_index) = [];
76 Oneway_Non_Stop.Total_Mile(remove_index) = [];
77 Oneway_Non_Stop.SEG_MILE(remove_index) = [];
78 Oneway_Non_Stop.ORIG_ARPT_CD(remove_index) = [];
79 Oneway_Non_Stop.DEST_ARPT_CD(remove_index) = [];
80 Oneway_Non_Stop.DOC_AMT(remove_index) = [];
81 Oneway_Non_Stop.SEG_DOC_AMT(remove_index) = [];
82 Oneway_Non_Stop.Fare_per_pax_mile(remove_index) = [];
83
84 %%
85
86 Oneway_Non_Stop_ser = getByteStreamFromArray(Oneway_Non_Stop);
87 clear Oneway_Non_Stop
88 save(([save_dir,main_delimiter,'Oneway_Non_Stop_US_ser.mat']),'Oneway_Non_Stop_ser','-v7.3')
89
90 return;
91
```

```
1 function [] = Rountrip_Non_Stop_Records_US(save_dir, Upload_Dir, main_delimiter,airport_list)
2
3 load(([Upload_Dir,main_delimiter,'Roundtrip_NonStop_Part_1_ser.mat']),'Roundtrip_NonStop_Part_1_ser')
4 Oneway_Non_Stop_Part_1 = getArrayFromByteStream(Roundtrip_NonStop_Part_1_ser);
5 clear Roundtrip_NonStop_Part_1_ser
6
7 load(([Upload_Dir,main_delimiter,'Roundtrip_NonStop_Part_2_ser.mat']),'Roundtrip_NonStop_Part_2_ser')
8 Oneway_Non_Stop_Part_2 = getArrayFromByteStream(Roundtrip_NonStop_Part_2_ser);
9 clear Roundtrip_NonStop_Part_2_ser
10 %%
11
12 Oneway_Non_Stop.TICKET_NBR = Oneway_Non_Stop_Part_1.TICKET_NBR;
13 Oneway_Non_Stop.SEG_ID = Oneway_Non_Stop_Part_1.SEG_ID;
14 Oneway_Non_Stop.Total_Mile = Oneway_Non_Stop_Part_1.Total_Mile;
15 Oneway_Non_Stop.SEG_MILE = Oneway_Non_Stop_Part_1.SEG_MILE;
16 Oneway_Non_Stop.ORIG_ARPT_CD = Oneway_Non_Stop_Part_1.ORIG_ARPT_CD;
17 Oneway_Non_Stop.DEST_ARPT_CD = Oneway_Non_Stop_Part_2.DEST_ARPT_CD;
18 Oneway_Non_Stop.DOC_AMT = Oneway_Non_Stop_Part_2.DOC_AMT;
```

```
Oneway_Non_Stop.SEG_DOC_AMT = Oneway_Non_Stop_Part_2.SEG_DOC_AMT;
Oneway_Non_Stop.Fare_per_pax_mile = Oneway_Non_Stop_Part_2.Fare_per_pax_mile;

clear Oneway_Non_Stop_Part_1 Oneway_Non_Stop_Part_2

%%
%Remove the airports that are not part of both data set (ARC & OAG 2016).

missing_origin_airports_index = ismember(Oneway_Non_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs) == 0;

tickets = Oneway_Non_Stop.TICKET_NBR(missing_origin_airports_index);

unqiue_unique = unique(tickets);

remove = ismember(Oneway_Non_Stop.TICKET_NBR,unqiue_unique) ~= 0;

Oneway_Non_Stop.TICKET_NBR(remove) = [];
Oneway_Non_Stop.SEG_ID(remove) = [];
Oneway_Non_Stop.Total_Mile(remove) = [];
Oneway_Non_Stop.SEG_MILE(remove) = [];
Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
Oneway_Non_Stop.DOC_AMT(remove) = [];
Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
%
missing_dest_airports_index = ismember(Oneway_Non_Stop.DEST_ARPT_CD,airport_list.Airport_IDs) == 0;

tickets = Oneway_Non_Stop.TICKET_NBR(missing_dest_airports_index);

unqiue_unique = unique(tickets);

remove = ismember(Oneway_Non_Stop.TICKET_NBR,unqiue_unique) ~= 0;

Oneway_Non_Stop.TICKET_NBR(remove) = [];
Oneway_Non_Stop.SEG_ID(remove) = [];
Oneway_Non_Stop.Total_Mile(remove) = [];
Oneway_Non_Stop.SEG_MILE(remove) = [];
Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
Oneway_Non_Stop.DOC_AMT(remove) = [];
Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];

%%

%We keep only those records within the continental US.
[~,origin_airport_country_index] = ismember(Oneway_Non_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs);
[~,dest_airport_country_index] = ismember(Oneway_Non_Stop.DEST_ARPT_CD,airport_list.Airport_IDs);

ORIG_ARPT_country = airport_list.Country_IDs(origin_airport_country_index);
DEST_ARPT_country = airport_list.Country_IDs(dest_airport_country_index);

OD_ARPT_country = strcat(ORIG_ARPT_country,'_',DEST_ARPT_country);

non_US_OD_index = ismember(OD_ARPT_country,'US_US') == 0;


Oneway_Non_Stop.TICKET_NBR(non_US_OD_index) = [];
Oneway_Non_Stop.SEG_ID(non_US_OD_index) = [];
Oneway_Non_Stop.Total_Mile(non_US_OD_index) = [];
Oneway_Non_Stop.SEG_MILE(non_US_OD_index) = [];
Oneway_Non_Stop.ORIG_ARPT_CD(non_US_OD_index) = [];
Oneway_Non_Stop.DEST_ARPT_CD(non_US_OD_index) = [];
Oneway_Non_Stop.DOC_AMT(non_US_OD_index) = [];
Oneway_Non_Stop.SEG_DOC_AMT(non_US_OD_index) = [];
```

```
85 Oneway_Non_Stop.Fare_per_pax_mile(non_US_OD_index) = [];
86
87 %only for US, remove records with a fare per mile greater than
88 %$1.50/statute mile
89 remove_index = find(Oneway_Non_Stop.Fare_per_pax_mile > 1.5);
90
91 Oneway_Non_Stop.TICKET_NBR(remove_index) = [];
92 Oneway_Non_Stop.SEG_ID(remove_index) = [];
93 Oneway_Non_Stop.Total_Mile(remove_index) = [];
94 Oneway_Non_Stop.SEG_MILE(remove_index) = [];
95 Oneway_Non_Stop.ORIG_ARPT_CD(remove_index) = [];
96 Oneway_Non_Stop.DEST_ARPT_CD(remove_index) = [];
97 Oneway_Non_Stop.DOC_AMT(remove_index) = [];
98 Oneway_Non_Stop.SEG_DOC_AMT(remove_index) = [];
99 Oneway_Non_Stop.Fare_per_pax_mile(remove_index) = [];
100
101 %%
102
103 Roundtrip_Non_Stop_ser = getByteStreamFromArray(Oneway_Non_Stop);
104 clear Oneway_Non_Stop
105 save(([save_dir,main_delimiter,'Roundtrip_Non_Stop_US_ser.mat']),'Roundtrip_Non_Stop_ser','-v7.3')
106
107 return;
108
```

```
1 function [] = Oneway_Onestop_Records_US(save_dir, Upload_Dir, main_delimiter,airport_list)
2
3 %Altohugh we are dealing with oneway_nonstop records. We want to keep does
4 %one-way OD pairs that appears in both files (onestop and nonstop). For
5 %that reason, we upload the oneway_non_stop output
6 load(([save_dir,main_delimiter,'Oneway_Non_Stop_US_ser.mat']),'Oneway_Non_Stop_ser')
7 Oneway_Non_Stop = getArrayFromByteStream(Oneway_Non_Stop_ser);
8 clear Oneway_Non_Stop_ser
9
10 load(([Upload_Dir,main_delimiter,'Oneway_OneStop_ser.mat']),'Oneway_OneStop_ser')
11 Oneway_OneStop = getArrayFromByteStream(Oneway_OneStop_ser);
12 clear Oneway_OneStop_ser
13 %%
14
15 %Remove the airports that are not part of both data set (ARC & OAG 2016).
16 missing_origin_airports_index = ismember(Oneway_OneStop.ORIG_ARPT_CD,airport_list.Airport_IDs) == 0;
17
18 tickets = Oneway_OneStop.TICKET_NBR(missing_origin_airports_index);
19 unqiue_unique = unique(tickets);
20 remove = ismember(Oneway_OneStop.TICKET_NBR,unqiue_unique) ~= 0;
21
22 Oneway_OneStop.TICKET_NBR(remove) = [];
23 Oneway_OneStop.SEG_ID(remove) = [];
24 Oneway_OneStop.SEG_MILE(remove) = [];
25 Oneway_OneStop.Total_Mile(remove) = [];
26 Oneway_OneStop.ORIG_ARPT_CD(remove) = [];
27 Oneway_OneStop.DEST_ARPT_CD(remove) = [];
28 Oneway_OneStop.DOC_AMT(remove) = [];
29 Oneway_OneStop.SEG_DOC_AMT(remove) = [];
30 Oneway_OneStop.Fare_per_pax_mile(remove) = [];
31
32 missing_dest_airports_index = ismember(Oneway_OneStop.DEST_ARPT_CD,airport_list.Airport_IDs) == 0;
33
34 tickets = Oneway_OneStop.TICKET_NBR(missing_dest_airports_index);
35 unqiue_unique = unique(tickets);
36 remove = ismember(Oneway_OneStop.TICKET_NBR,unqiue_unique) ~= 0;
37
38 Oneway_OneStop.TICKET_NBR(remove) = [];
39 Oneway_OneStop.SEG_ID(remove) = [];
40 Oneway_OneStop.SEG_MILE(remove) = [];
41 Oneway_OneStop.Total_Mile(remove) = [];
```

```
Oneway_OneStop.ORIG_ARPT_CD(remove) = [];
Oneway_OneStop.DEST_ARPT_CD(remove) = [];
Oneway_OneStop.DOC_AMT(remove) = [];
Oneway_OneStop.SEG_DOC_AMT(remove) = [];
Oneway_OneStop.Fare_per_pax_mile(remove) = [];


%We keep only those records within the continental US.
[~,origin_airport_country_index] = ismember(Oneway_OneStop.ORIG_ARPT_CD,airport_list.Airport_IDs);
[~,dest_airport_country_index] = ismember(Oneway_OneStop.DEST_ARPT_CD,airport_list.Airport_IDs);

ORIG_ARPT_country = airport_list.Country_IDs(origin_airport_country_index);
DEST_ARPT_country = airport_list.Country_IDs(dest_airport_country_index);

OD_ARPT_country = strcat(ORIG_ARPT_country,'_',DEST_ARPT_country);

non_US_OD_index = ismember(OD_ARPT_country,'US_US') == 0;

Oneway_OneStop.TICKET_NBR(non_US_OD_index) = [];
Oneway_OneStop.SEG_ID(non_US_OD_index) = [];
Oneway_OneStop.SEG_MILE(non_US_OD_index) = [];
Oneway_OneStop.Total_Mile(non_US_OD_index) = [];
Oneway_OneStop.ORIG_ARPT_CD(non_US_OD_index) = [];
Oneway_OneStop.DEST_ARPT_CD(non_US_OD_index) = [];
Oneway_OneStop.DOC_AMT(non_US_OD_index) = [];
Oneway_OneStop.SEG_DOC_AMT(non_US_OD_index) = [];
Oneway_OneStop.Fare_per_pax_mile(non_US_OD_index) = [];


%%
%reorganize onestop records
unique_ticket_number = unique(Oneway_OneStop.TICKET_NBR);

length_of_unique_ticket_number = length(unique_ticket_number);
row = 1;
for record = 1:length_of_unique_ticket_number

if mod(record, 5000) == 0
disp([num2str(record), '/', num2str(length_of_unique_ticket_number)]);

end

match_index = find(ismember(Oneway_OneStop.TICKET_NBR,unique_ticket_number(record)));
length_match = length(match_index);

if length_match == 2

seg_1_index = Oneway_OneStop.SEG_ID(match_index) == 1;
seg_2_index = Oneway_OneStop.SEG_ID(match_index) == 2;

One_Stop.TICKET_NBR(row,1) = unique_ticket_number(record);
One_Stop.ORIG_ARPT_CD(row,1) = Oneway_OneStop.ORIG_ARPT_CD(match_index(seg_1_index));
One_Stop.DEST_ARPT_CD(row,1) = Oneway_OneStop.DEST_ARPT_CD(match_index(seg_2_index));
One_Stop.STOP_ARPT(row,1) = Oneway_OneStop.DEST_ARPT_CD(match_index(seg_1_index));
One_Stop.Total_Distance(row,1) = Oneway_OneStop.Total_Mile(match_index(seg_1_index));
%SEG_MILE_one_stop(match_index(seg_1_index)) +
SEG_MILE_one_stop(match_index(seg_2_index));
One_Stop.DOC_AMT(row,1) = Oneway_OneStop.DOC_AMT(match_index(seg_1_index));
One_Stop.Fare_per_Mile(row,1) = Oneway_OneStop.Fare_per_pax_mile(match_index(seg_1_index));

row = row+1;

end
end

remove_index = find(One_Stop.Fare_per_Mile > 1.5);
```

```
106
107 One_Stop.TICKET_NBR(remove_index) = [];
108 One_Stop.ORIG_ARPT_CD(remove_index) = [];
109 One_Stop.DEST_ARPT_CD(remove_index) = [];
110 One_Stop.STOP_ARPT(remove_index) = [];
111 One_Stop.Total_Distance(remove_index) = [];
112 One_Stop.DOC_AMT(remove_index) = [];
113 One_Stop.Fare_per_Mile(remove_index) = [];
114
115
116 %%
117
118 OD_non_stop = strcat(Oneway_Non_Stop.ORIG_ARPT_CD,'_',Oneway_Non_Stop.DEST_ARPT_CD);
119
120 OD_one_stop = strcat(One_Stop.ORIG_ARPT_CD,'_',One_Stop.DEST_ARPT_CD);
121
122 [~,match_index] = ismember(OD_one_stop,OD_non_stop);
123
124 match_nonstop_onestop_index = find(match_index ~= 0);
125
126 %Match between nonstop and oneway
127 One_Stop_Match.TICKET_NBR = One_Stop.TICKET_NBR(match_nonstop_onestop_index);
128 One_Stop_Match.ORIG_ARPT_CD = One_Stop.ORIG_ARPT_CD(match_nonstop_onestop_index);
129 One_Stop_Match.DEST_ARPT_CD = One_Stop.DEST_ARPT_CD(match_nonstop_onestop_index);
130 One_Stop_Match.STOP_ARPT = One_Stop.STOP_ARPT(match_nonstop_onestop_index);
131 One_Stop_Match.Total_Distance = One_Stop.Total_Distance(match_nonstop_onestop_index);
132 One_Stop_Match.DOC_AMT = One_Stop.DOC_AMT(match_nonstop_onestop_index);
133 One_Stop_Match.Fare_per_Mile = One_Stop.Fare_per_Mile(match_nonstop_onestop_index);
134
135 save(([save_dir,main_delimiter,'One_Stop_Match_US.mat']),'One_Stop_Match')
136
137 return;
138
139
140
```

```
1
2 function [] = Value_of_Time_Calculation_US(save_dir, main_delimiter,Input_Folder_Dir,airport_list)
3
4 min_record_count = 10;
5 layover_time_hrs = 1; %hrs
6
7 load(([save_dir,main_delimiter,'Oneway_Non_Stop_US_ser.mat']),'Oneway_Non_Stop_ser')
8 Oneway_Non_Stop = getArrayFromByteStream(Oneway_Non_Stop_ser);
9 clear Oneway_Non_Stop_ser
10
11 load(([save_dir,main_delimiter,'One_Stop_Match_US.mat']),'One_Stop_Match')
12
13 load(([Input_Folder_Dir,'\VOT\CA_Network_2016_Domestic\Output_Files\A2A_CA_OTD_Network_NonStop_2016.
mat']),'A2A_CA_OTD_Network_NonStop_Ser')
14
15 A2A_CA_OTD_Network_NonStop = getArrayFromByteStream(A2A_CA_OTD_Network_NonStop_Ser);
16 clear A2A_CA_OTD_Network_NonStop_Ser
17
18
load(([Input_Folder_Dir,'\VOT\CA_Network_2016_Domestic\Output_Files\A2A_CA_TravelTime_2Legs_2016.mat']),'A2A_CA_TravelTime_2Legs
_Ser')
19 A2A_CA_TravelTime_2Legs = getArrayFromByteStream(A2A_CA_TravelTime_2Legs_Ser);
20 clear A2A_CA_TravelTime_2Legs_Ser
21
22 load(([Input_Folder_Dir,'\VOT\CA_Network_2016_Domestic\Input_Files\airportlist_oag_2016.mat']),'airportlist_oag')
23 airportlist_oag = cellstr(airportlist_oag);
24
25 load(([Input_Folder_Dir,main_delimiter,'OD_Pairs_Year_2016.mat']),'OD_Pairs_Year_2016')
26 %%
27
```

```
%Remove nonstop records for trips with no onestop records data
%available
OD_non_stop = strcat(Oneway_Non_Stop.ORIG_ARPT_CD,'_',Oneway_Non_Stop.DEST_ARPT_CD);

OD_one_stop = strcat(One_Stop_Match.ORIG_ARPT_CD,'_',One_Stop_Match.DEST_ARPT_CD);

[~,match_index] = ismember(OD_non_stop,OD_one_stop);
remove = find(match_index==0);

Oneway_Non_Stop.TICKET_NBR(remove) = [];
Oneway_Non_Stop.SEG_ID(remove) = [];
Oneway_Non_Stop.Total_Mile(remove) = [];
Oneway_Non_Stop.SEG_MILE(remove) = [];
Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
Oneway_Non_Stop.DOC_AMT(remove) = [];
Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];


%%
%Remove ARC records for one-way OD pairs that are not part of OAG2016
OD_non_stop = strcat(Oneway_Non_Stop.ORIG_ARPT_CD,'_',Oneway_Non_Stop.DEST_ARPT_CD);

OD_one_stop_1 = strcat(One_Stop_Match.ORIG_ARPT_CD,'_',One_Stop_Match.STOP_ARPT);

OD_one_stop_2 = strcat(One_Stop_Match.STOP_ARPT,'_',One_Stop_Match.DEST_ARPT_CD);


OD_OAG = strcat(OD_Pairs_Year_2016.DepAirport,'_',OD_Pairs_Year_2016.ArrAirport);


[~,OAG_non_stop_index] = ismember(OD_non_stop,OD_OAG);

[~,OAG_one_non_stop_1_index] = ismember(OD_one_stop_1,OD_OAG);

[~,OAG_one_non_stop_2_index] = ismember(OD_one_stop_2,OD_OAG);

delete_0 = find(OAG_non_stop_index == 0);
delete_1 = find(OAG_one_non_stop_1_index == 0);
delete_2 = find(OAG_one_non_stop_2_index == 0);
delete_3 = [delete_1 ; delete_2];

Oneway_Non_Stop.TICKET_NBR(delete_0) = [];
Oneway_Non_Stop.ORIG_ARPT_CD(delete_0) = [];
Oneway_Non_Stop.DEST_ARPT_CD(delete_0) = [];
Oneway_Non_Stop.DOC_AMT(delete_0) = [];
Oneway_Non_Stop.Fare_per_pax_mile(delete_0) = [];
Oneway_Non_Stop.SEG_ID(delete_0) = [];
Oneway_Non_Stop.Total_Mile(delete_0) = [];
Oneway_Non_Stop.SEG_MILE(delete_0) = [];
Oneway_Non_Stop.SEG_DOC_AMT(delete_0) = [];

OD_non_stop(delete_0) = [];

OD_one_stop_1(delete_3) = [];
OD_one_stop_2(delete_3) = [];
One_Stop_Match.TICKET_NBR(delete_3) = [];
One_Stop_Match.ORIG_ARPT_CD(delete_3) = [];
One_Stop_Match.DEST_ARPT_CD(delete_3) = [];
One_Stop_Match.STOP_ARPT(delete_3) = [];
One_Stop_Match.Total_Distance(delete_3) = [];
One_Stop_Match.DOC_AMT(delete_3) = [];
One_Stop_Match.Fare_per_Mile(delete_3) = [];

%Assign travel times to ARC records based on OAG elapsed travel times
```

```
94 [~,OAG_non_stop_index] = ismember(OD_non_stop,OD_OAG);
95
96 [~,OAG_one_non_stop_1_index] = ismember(OD_one_stop_1,OD_OAG);
97
98 [~,OAG_one_non_stop_2_index] = ismember(OD_one_stop_2,OD_OAG);
99
100 Travel_Time_non_stop_hrs = OD_Pairs_Year_2016.OD_AverageElapsedTime_hrs(OAG_non_stop_index);
101
102 One_Stop_Match.Travel_Time_hrs = OD_Pairs_Year_2016.OD_AverageElapsedTime_hrs(OAG_one_non_stop_1_index) +
OD_Pairs_Year_2016.
OD_AverageElapsedTime_hrs(OAG_one_non_stop_2_index) + layover_time_hrs; %1hrs of layover is added
103 %%
104 %Start of VOT Calculation
105 OD_one_stop = strcat(One_Stop_Match.ORIG_ARPT_CD,'_',One_Stop_Match.DEST_ARPT_CD);
106
107 OD_Combined = unique([OD_non_stop;OD_one_stop]);
108
109 length_of_OD_Combined = length(OD_Combined);
110 row = 1;
111 for OD = 1:length_of_OD_Combined
112
113 if mod(OD, 100) == 0
114 disp([num2str(OD), '/', num2str(length_of_OD_Combined)]);
115
116 end
117
118 if sum(ismember(OD_non_stop,OD_Combined(OD))) > 0 && sum(ismember(OD_one_stop,OD_Combined(OD))) > 0
119
120
121 [non_stop_index,~] = ismember(OD_non_stop,OD_Combined(OD));
122
123 [one_stop_index,~] = ismember(OD_one_stop,OD_Combined(OD));
124
125 non_stop_Fare = Oneway_Non_Stop.SEG_DOC_AMT(non_stop_index);
126 non_stop_travel_time_hrs = Travel_Time_non_stop_hrs(non_stop_index);
127
128 non_stop_Fare_50_percentile = prctile(non_stop_Fare,50);
129 non_stop_travel_time_hrs_mean = mean(non_stop_travel_time_hrs);
130
131 one_stop_Fare = One_Stop_Match.DOC_AMT(one_stop_index);
132 one_stop_travel_time_hrs = One_Stop_Match.Travel_Time_hrs(one_stop_index);
133
134
135
136 %%
137 origin_oag_list = char(OD_Combined(OD));
138 destination_oag_list = cellstr(origin_oag_list(1,5:7));
139 origin_oag_list = cellstr(origin_oag_list(1,1:3));
140
141 [origin_oag_list_index,~] = ismember(airportlist_oag,origin_oag_list);
142 [destination_oag_list_index,~] = ismember(airportlist_oag,destination_oag_list);
143
144 if sum(origin_oag_list_index) ~= 0
145
146 if sum(destination_oag_list_index) ~= 0
147
148 od_stop_index = A2A_CA_OTD_Network_NonStop(origin_oag_list_index,destination_oag_list_index);
149 if isempty(od_stop_index.legs) == 0
150 od_stop_travel_times = A2A_CA_TravelTime_2Legs(origin_oag_list_index,destination_oag_list_index);
151
152 remove = find(od_stop_index.legs == destination_oag_list_index);
153
154 od_stop_index.legs(remove) = [];
155 od_stop_travel_times.legs(remove) = [];
156 if isempty(od_stop_index.legs) == 0
157 stop_no_match_remove = ismember(airportlist_oag(od_stop_index.legs),One_Stop_Match.STOP_ARPT(one_stop_index))==0;
```

```
158
159 od_stop_index.legs(stop_no_match_remove) = [];
160 od_stop_travel_times.legs(stop_no_match_remove) = [];
161
162 if isempty(od_stop_travel_times.legs) == 1
163
164 od_stop_travel_times.legs = prctile(one_stop_travel_time_hrs,50);
165
166 end
167
168 end
169
170 end
171
172 end
173
174 end
175
176 if exist('od_stop_index','var') == 0
177
178 clear od_stop_travel_times
179 elseif exist('od_stop_index.legs','var') == 1
180 clear od_stop_travel_times
181
182 end
183
184 if (exist('od_stop_travel_times','var') == 1)
185
186 if isempty(one_stop_Fare) == 0
187
188 one_stop_Fare_50_percentile = prctile(one_stop_Fare,50);
189 one_stop_travel_time_hrs_mean = mean(od_stop_travel_times.legs);
190 clear od_stop_travel_times remove od_stop_index
191
192 if ((non_stop_Fare_50_percentile - one_stop_Fare_50_percentile) / (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean))
>= 0
193
194 if (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean) >= 1
195
196 if length(one_stop_Fare) >= min_record_count && length(non_stop_Fare) >= min_record_count
197
198 VOT.OD(row,1) = OD_Combined(OD);
199 VOT.Mean_Non_Stop_Fare(row,1) = non_stop_Fare_50_percentile;
200 VOT.Mean_Non_Stop_Travel_Time(row,1) = non_stop_travel_time_hrs_mean;
201 VOT.Mean_One_Stop_Fare(row,1) = one_stop_Fare_50_percentile;
202 VOT.Mean_One_Stop_Travel_Time(row,1) = one_stop_travel_time_hrs_mean;
203 VOT.Delta_Fare(row,1) = (non_stop_Fare_50_percentile - one_stop_Fare_50_percentile);
204 VOT.Delta_Stop_Travel_Time(row,1) = (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean);
205
206 VOT.Value_of_Time(row,1) = (non_stop_Fare_50_percentile - one_stop_Fare_50_percentile) ./ (one_stop_travel_time_hrs_mean -
non_stop_travel_time_hrs_mean);
207 VOT.Non_stop_count(row,1) = length(non_stop_Fare);
208 VOT.One_stop_count(row,1) = length(one_stop_Fare);
209 VOT.OAG_TT_Used(row,1) = 1;
210 row = row + 1;
211
212 clear od_stop_index od_stop_travel_times
213 end
214
215 end
216 end
217
218 end
219
220 else
221
```

```
222 %%
223 if isempty(one_stop_Fare) == 0
224
225 one_stop_Fare_50_percentile = prctile(one_stop_Fare,50);
226 one_stop_travel_time_hrs_mean = mean(one_stop_travel_time_hrs);
227
228 if ((non_stop_Fare_50_percentile - one_stop_Fare_50_percentile) / (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean))
>= 0
229
230 if (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean) >= 1
231
232 if length(one_stop_Fare) >= min_record_count && length(non_stop_Fare) >= min_record_count
233
234 VOT.OD(row,1) = OD_Combined(OD);
235 VOT.Mean_Non_Stop_Fare(row,1) = non_stop_Fare_50_percentile;
236 VOT.Mean_Non_Stop_Travel_Time(row,1) = non_stop_travel_time_hrs_mean;
237 VOT.Mean_One_Stop_Fare(row,1) = one_stop_Fare_50_percentile;
238 VOT.Delta_Fare(row,1) = (non_stop_Fare_50_percentile - one_stop_Fare_50_percentile);
239 VOT.Delta_Stop_Travel_Time(row,1) = (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean);
240
241 VOT.Value_of_Time(row,1) = (non_stop_Fare_50_percentile - one_stop_Fare_50_percentile) ./ (one_stop_travel_time_hrs_mean -
non_stop_travel_time_hrs_mean);
242 VOT.Non_stop_count(row,1) = length(non_stop_Fare);
243 VOT.One_stop_count(row,1) = length(one_stop_Fare);
244 VOT.OAG_TT_Used(row,1) = 2;
245 row = row + 1;
246
247
248 end
249
250 end
251 end
252 end
253 end
254
255 end
256 clearvars -except OD_non_stop Oneway_Non_Stop...
257 One_Stop_Match OD_one_stop OD_Combined...
258 length_of_OD_Combined row VOT LO load_dir OD Travel_Time_non_stop_hrs...
259 airportlist_oag A2A_CA_OTD_Network_NonStop A2A_CA_TravelTime_2Legs...
260 min_record_count od_stop_travel_times airport_list OD_Pairs_Year_2016...
261 save_dir main_delimiter Input_Folder_Dir airport_list;
262 end
263
264 %Save VOT value by one-way OD pair
265
266 VOT_by_OD = VOT;
267
268 save(([save_dir,main_delimiter,'VOT_by_OD_US.mat']),'VOT_by_OD')
269
270 clear VOT
271 %%
272 %VOT By Airport
273
274 length_of_od = length(VOT_by_OD.OD);
275 origin_airport = strings(length_of_od,1);
276 for od = 1:length_of_od
277
278 od_char = char(VOT_by_OD.OD(od,1));
279 origin_airport(od,1) = cellstr(od_char(1:3));
280
281 end
282
283 unqiue_origin_airport = unique(origin_airport);
284
285 length_of_airports = length(unqiue_origin_airport);
```

```
286
287 for airport = 1:length_of_airports
288
289 [airport_index,~] = ismember(origin_airport,unqiue_origin_airport(airport));
290
291 records_used = VOT_by_OD.Non_stop_count(airport_index) + VOT_by_OD.One_stop_count(airport_index);
292
293 VOT.Value_of_Time(airport,1) = round(sum(VOT_by_OD.Value_of_Time(airport_index) .* records_used) ./ sum(records_used),0);
294 VOT.Number_of_records(airport,1) = sum(records_used);
295 end
296
297 VOT.Airport = unqiue_origin_airport;
298
299 %Save VOT values by Airport
300 save(([save_dir,main_delimiter,'VOT_by_Aiport_US.mat']),'VOT')
301
302 %%
303
304 us_index = find(strcmp(OD_Pairs_Year_2016.DepIATACtry,'US'));
305 high_demand_index = OD_Pairs_Year_2016.Total_Seats(us_index)>=100000;
306 Value_of_Time.Airport = unique(OD_Pairs_Year_2016.DepAirport(us_index(high_demand_index)));
307
308 length_airport = length(VOT.Airport);
309
310 Value_of_Time.Value((1:length(Value_of_Time.Airport)),1) = round(mean(VOT.Value_of_Time),0);
311 Value_of_Time.Number_of_records((1:length(Value_of_Time.Airport)),1) = 1;
312 for airport = 1:length_airport
313
314 [airport_index,~] = ismember(Value_of_Time.Airport,VOT.Airport(airport));
315
316
317 Value_of_Time.Value(airport_index,1) = VOT.Value_of_Time(airport);
318 Value_of_Time.Number_of_records(airport_index,1) = VOT.Number_of_records(airport);
319 end
320
321 o_airport = Value_of_Time.Airport;
322 d_airport = Value_of_Time.Airport;
323
324 length_of_airport = length(o_airport);
325
326 origin_list = strings(length_of_airport,1);
327 destination_list = strings(length_of_airport,1);
328
329 m=1;
330 for origin = 1:length_of_airport
331
332 for destination = 1:length_of_airport
333
334 origin_list(m,1) = o_airport(origin);
335 destination_list(m,1) = d_airport(destination);
336 m = m+1;
337 end
338 end
339
340 od_from_vot_matrix = strcat(origin_list,'_',destination_list);
341 od_oag = strcat(OD_Pairs_Year_2016.DepAirport,'_',OD_Pairs_Year_2016.ArrAirport);
342 row = 0;
343 od_dist(1:length_of_airport,1:length_of_airport) = 999999;
344
345 for origin = 1:length_of_airport
346
347 for destination = 1:length_of_airport
348 row = row + 1;
349 if origin == destination
350
351 %do nothing
```

```
352
353 elseif sum(ismember(od_oag,od_from_vot_matrix(row))) ~= 0
354
355 [index,~] = ismember(od_oag,od_from_vot_matrix(row));
356 od_dist(origin,destination) = OD_Pairs_Year_2016.DistStMiles(index);
357 else
358
359 [o_match_index,~] = ismember(airport_list.Airport_IDs,o_airport(origin));
360 [d_match_index,~]= ismember(airport_list.Airport_IDs,d_airport(destination));
361
362 lat_o = airport_list.Apt_Lat(o_match_index);
363 lon_o = airport_list.Apt_Lon(o_match_index);
364
365 lat_d = airport_list.Apt_Lat(d_match_index);
366 lon_d = airport_list.Apt_Lon(d_match_index);
367
368 od_dist(origin,destination) = ceil(deg2sm(distance(lat_o,lon_o,lat_d,lon_d)));
369
370 end
371 end
372 end
373
374 close_by_airports(length_of_airport,1).index = zeros(length_of_airport,1);
375
376 for origin = 1:length_of_airport
377
378 close_by_airports(origin).index = find(od_dist(origin,:) <= 30);
379
380 end
381
382 for origin = 1:length_of_airport
383
384 if isempty(close_by_airports(origin).index) == 0
385
386 Value_of_Time.Value(close_by_airports(origin).index) = ceil(sum(Value_of_Time.Value(close_by_airports(origin).index) .* Value_of_Time.
Number_of_records(close_by_airports(origin).index)) ./ sum(Value_of_Time.Number_of_records(close_by_airports(origin).index)));
387
388 end
389 end
390
391 %Save Value of time (by weighted average)
392 save(([save_dir,main_delimiter,'Value_of_Time_US.mat']),'Value_of_Time')
393
394 return;
1 function [] = US_Int_VOT_Main_Function(SST_Pre_Processing_Dir, Input_Folder_Dir)
2 %% This script calculates the value of time for the US-Int. market
3
4 main_delimiter = '\';
5 % Input_Folder_Dir = ([local_disc,'..\..\..\..\..\..\SST_2020_Input']);
6 SST_US_Int_VOT_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'VOT\US_Int_Market']);
7 save_dir = ([SST_US_Int_VOT_Dir,main_delimiter,'Output']);
8 Upload_Dir = ('.\ARC_2016\Output');
9 %Create Output directory
10 if exist((save_dir),'dir') == 0
11
12 mkdir(([SST_US_Int_VOT_Dir,main_delimiter,'Output']))
13
14 end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0
15
16 addpath(([SST_US_Int_VOT_Dir,main_delimiter,'Output']))
17
18 load(([Input_Folder_Dir,main_delimiter,'airport_list.mat']),'airport_list')
19
20 Oneway_Non_Stop_Records_US_Int(save_dir, Upload_Dir, main_delimiter,airport_list)
21
22 Rountrip_Non_Stop_Records_US_Int(save_dir, Upload_Dir, main_delimiter,airport_list)
```

```
23
24 Oneway_Onestop_Records_US_Int(save_dir, Upload_Dir, main_delimiter,airport_list)
25
26 Value_of_Time_Calculation_US_Int(save_dir, main_delimiter,Input_Folder_Dir,airport_list)
27
28 return;

1 function [] = Oneway_Non_Stop_Records_US_Int(save_dir, Upload_Dir, main_delimiter,airport_list)
2
3 load(([Upload_Dir,main_delimiter,'Oneway_Non_Stop_ser.mat']),'Oneway_Non_Stop_ser')
4 Oneway_Non_Stop = getArrayFromByteStream(Oneway_Non_Stop_ser);
5 clear Oneway_Non_Stop_ser
6
7 %%
8
9 %Remove the airports that are not part of both data set (ARC & OAG 2016).
10
11 missing_origin_airports_index = ismember(Oneway_Non_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs) == 0;
12
13 tickets = Oneway_Non_Stop.TICKET_NBR(missing_origin_airports_index);
14
15 unqiue_unique = unique(tickets);
16
17 remove = ismember(Oneway_Non_Stop.TICKET_NBR,unqiue_unique) ~= 0;
18
19 Oneway_Non_Stop.TICKET_NBR(remove) = [];
20 Oneway_Non_Stop.SEG_ID(remove) = [];
21 Oneway_Non_Stop.Total_Mile(remove) = [];
22 Oneway_Non_Stop.SEG_MILE(remove) = [];
23 Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
24 Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
25 Oneway_Non_Stop.DOC_AMT(remove) = [];
26 Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
27 Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
28 %
29 missing_dest_airports_index = ismember(Oneway_Non_Stop.DEST_ARPT_CD,airport_list.Airport_IDs) == 0;
30
31 tickets = Oneway_Non_Stop.TICKET_NBR(missing_dest_airports_index);
32
33 unqiue_unique = unique(tickets);
34
35 remove = ismember(Oneway_Non_Stop.TICKET_NBR,unqiue_unique) ~= 0;
36
37 Oneway_Non_Stop.TICKET_NBR(remove) = [];
38 Oneway_Non_Stop.SEG_ID(remove) = [];
39 Oneway_Non_Stop.Total_Mile(remove) = [];
40 Oneway_Non_Stop.SEG_MILE(remove) = [];
41 Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
42 Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
43 Oneway_Non_Stop.DOC_AMT(remove) = [];
44 Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
45 Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
46
47 %%
48
49
50 %%We keep only those records between US and International market.
51
52 [~,origin_airport_country_index] = ismember(Oneway_Non_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs);
53 [~,dest_airport_country_index] = ismember(Oneway_Non_Stop.DEST_ARPT_CD,airport_list.Airport_IDs);
54
55 ORIG_ARPT_country = airport_list.Country_IDs(origin_airport_country_index);
56 DEST_ARPT_country = airport_list.Country_IDs(dest_airport_country_index);
57
58 [~ , ORGI_US] = ismember(ORIG_ARPT_country,'US');
59 [~ , DEST_us] = ismember(DEST_ARPT_country,'US');
```

```
60 OD_us = ORGI_US + DEST_us;
61 remove = find(OD_us ~= 1);
62
63 Oneway_Non_Stop.TICKET_NBR(remove) = [];
64 Oneway_Non_Stop.SEG_ID(remove) = [];
65 Oneway_Non_Stop.Total_Mile(remove) = [];
66 Oneway_Non_Stop.SEG_MILE(remove) = [];
67 Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
68 Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
69 Oneway_Non_Stop.DOC_AMT(remove) = [];
70 Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
71 Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
72
73 %%
74
75 Oneway_Non_Stop_ser = getByteStreamFromArray(Oneway_Non_Stop);
76 clear Oneway_Non_Stop
77 save(([save_dir,main_delimiter,'Oneway_Non_Stop_US_Int_ser.mat']),'Oneway_Non_Stop_ser','-v7.3')
78
79 return;
```

```
1 function [] = Rountrip_Non_Stop_Records_US_Int(save_dir, Upload_Dir, main_delimiter,airport_list)
2
3 load(([Upload_Dir,main_delimiter,'Roundtrip_NonStop_Part_1_ser.mat']),'Roundtrip_NonStop_Part_1_ser')
4 Oneway_Non_Stop_Part_1 = getArrayFromByteStream(Roundtrip_NonStop_Part_1_ser);
5 clear Roundtrip_NonStop_Part_1_ser
6
7 load(([Upload_Dir,main_delimiter,'Roundtrip_NonStop_Part_2_ser.mat']),'Roundtrip_NonStop_Part_2_ser')
8 Oneway_Non_Stop_Part_2 = getArrayFromByteStream(Roundtrip_NonStop_Part_2_ser);
9 clear Roundtrip_NonStop_Part_2_ser
10 %%
11
12 Oneway_Non_Stop.TICKET_NBR = Oneway_Non_Stop_Part_1.TICKET_NBR;
13 Oneway_Non_Stop.SEG_ID = Oneway_Non_Stop_Part_1.SEG_ID;
14 Oneway_Non_Stop.Total_Mile = Oneway_Non_Stop_Part_1.Total_Mile;
15 Oneway_Non_Stop.SEG_MILE = Oneway_Non_Stop_Part_1.SEG_MILE;
16 Oneway_Non_Stop.ORIG_ARPT_CD = Oneway_Non_Stop_Part_1.ORIG_ARPT_CD;
17 Oneway_Non_Stop.DEST_ARPT_CD = Oneway_Non_Stop_Part_2.DEST_ARPT_CD;
18 Oneway_Non_Stop.DOC_AMT = Oneway_Non_Stop_Part_2.DOC_AMT;
19 Oneway_Non_Stop.SEG_DOC_AMT = Oneway_Non_Stop_Part_2.SEG_DOC_AMT;
20 Oneway_Non_Stop.Fare_per_pax_mile = Oneway_Non_Stop_Part_2.Fare_per_pax_mile;
21
22 clear Oneway_Non_Stop_Part_1 Oneway_Non_Stop_Part_2
23
24 %%
25 %Remove the airports that are not part of both data set (ARC & OAG 2016).
26
27 missing_origin_airports_index = ismember(Oneway_Non_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs) == 0;
28
29 tickets = Oneway_Non_Stop.TICKET_NBR(missing_origin_airports_index);
30
31 unqiue_unique = unique(tickets);
32
33 remove = ismember(Oneway_Non_Stop.TICKET_NBR,unqiue_unique) ~= 0;
34
35 Oneway_Non_Stop.TICKET_NBR(remove) = [];
36 Oneway_Non_Stop.SEG_ID(remove) = [];
37 Oneway_Non_Stop.Total_Mile(remove) = [];
38 Oneway_Non_Stop.SEG_MILE(remove) = [];
39 Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
40 Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
41 Oneway_Non_Stop.DOC_AMT(remove) = [];
42 Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
43 Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
44 %
```

```
45 missing_dest_airports_index = ismember(Oneway_Non_Stop.DEST_ARPT_CD,airport_list.Airport_IDs) == 0;
46
47 tickets = Oneway_Non_Stop.TICKET_NBR(missing_dest_airports_index);
48
49 unqiue_unique = unique(tickets);
50
51 remove = ismember(Oneway_Non_Stop.TICKET_NBR,unqiue_unique) ~= 0;
52
53 Oneway_Non_Stop.TICKET_NBR(remove) = [];
54 Oneway_Non_Stop.SEG_ID(remove) = [];
55 Oneway_Non_Stop.Total_Mile(remove) = [];
56 Oneway_Non_Stop.SEG_MILE(remove) = [];
57 Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
58 Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
59 Oneway_Non_Stop.DOC_AMT(remove) = [];
60 Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
61 Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
62
63 %%
64
65 %We keep only those records between US and International.
66 [~,origin_airport_country_index] = ismember(Oneway_Non_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs);
67 [~,dest_airport_country_index] = ismember(Oneway_Non_Stop.DEST_ARPT_CD,airport_list.Airport_IDs);
68
69 ORIG_ARPT_country = airport_list.Country_IDs(origin_airport_country_index);
70 DEST_ARPT_country = airport_list.Country_IDs(dest_airport_country_index);
71
72 %%
73 [~ , ORGI_US] = ismember(ORIG_ARPT_country,'US');
74 [~ , DEST_us] = ismember(DEST_ARPT_country,'US');
75 OD_us = ORGI_US + DEST_us;
76 remove = find(OD_us ~= 1);
77
78 Oneway_Non_Stop.TICKET_NBR(remove) = [];
79 Oneway_Non_Stop.SEG_ID(remove) = [];
80 Oneway_Non_Stop.Total_Mile(remove) = [];
81 Oneway_Non_Stop.SEG_MILE(remove) = [];
82 Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
83 Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
84 Oneway_Non_Stop.DOC_AMT(remove) = [];
85 Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
86 Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
87
88 %%
89
90 Roundtrip_Non_Stop_ser = getByteStreamFromArray(Oneway_Non_Stop);
91 clear Oneway_Non_Stop
92 save(([save_dir,main_delimiter,'Roundtrip_Non_Stop_US_Int_ser.mat']),'Roundtrip_Non_Stop_ser','-v7.3')
93
94 return;
```

```
1 function [] = Oneway_Onestop_Records_US_Int(save_dir, Upload_Dir, main_delimiter,airport_list)
2
3 %Altohugh we are dealing with oneway_nonstop records. We want to keep does
4 %one-way OD pairs that appears in both files (onestop and nonstop). For
5 %that reason, we upload the oneway_non_stop output
6 load(([save_dir,main_delimiter,'Oneway_Non_Stop_US_Int_ser.mat']),'Oneway_Non_Stop_ser')
7 Oneway_Non_Stop = getArrayFromByteStream(Oneway_Non_Stop_ser);
8 clear Oneway_Non_Stop_ser
9
10 load(([Upload_Dir,main_delimiter,'Oneway_OneStop_ser.mat']),'Oneway_OneStop_ser')
11 Oneway_OneStop = getArrayFromByteStream(Oneway_OneStop_ser);
12 clear Oneway_OneStop_ser
13
14
15 %%
```

```
16
17 %Remove the airports that are not part of both data set (ARC & OAG 2016).
18 missing_origin_airports_index = ismember(Oneway_OneStop.ORIG_ARPT_CD,airport_list.Airport_IDs) == 0;
19
20 tickets = Oneway_OneStop.TICKET_NBR(missing_origin_airports_index);
21 unqiue_unique = unique(tickets);
22 remove = ismember(Oneway_OneStop.TICKET_NBR,unqiue_unique) ~= 0;
23
24 Oneway_OneStop.TICKET_NBR(remove) = [];
25 Oneway_OneStop.SEG_ID(remove) = [];
26 Oneway_OneStop.SEG_MILE(remove) = [];
27 Oneway_OneStop.Total_Mile(remove) = [];
28 Oneway_OneStop.ORIG_ARPT_CD(remove) = [];
29 Oneway_OneStop.DEST_ARPT_CD(remove) = [];
30 Oneway_OneStop.DOC_AMT(remove) = [];
31 Oneway_OneStop.SEG_DOC_AMT(remove) = [];
32 Oneway_OneStop.Fare_per_pax_mile(remove) = [];
33
34 missing_dest_airports_index = ismember(Oneway_OneStop.DEST_ARPT_CD,airport_list.Airport_IDs) == 0;
35
36 tickets = Oneway_OneStop.TICKET_NBR(missing_dest_airports_index);
37 unqiue_unique = unique(tickets);
38 remove = ismember(Oneway_OneStop.TICKET_NBR,unqiue_unique) ~= 0;
39
40 Oneway_OneStop.TICKET_NBR(remove) = [];
41 Oneway_OneStop.SEG_ID(remove) = [];
42 Oneway_OneStop.SEG_MILE(remove) = [];
43 Oneway_OneStop.Total_Mile(remove) = [];
44 Oneway_OneStop.ORIG_ARPT_CD(remove) = [];
45 Oneway_OneStop.DEST_ARPT_CD(remove) = [];
46 Oneway_OneStop.DOC_AMT(remove) = [];
47 Oneway_OneStop.SEG_DOC_AMT(remove) = [];
48 Oneway_OneStop.Fare_per_pax_mile(remove) = [];
49
50 %%
51 %reorganize onestop records
52 unique_ticket_number = unique(Oneway_OneStop.TICKET_NBR);
53
54 length_of_unique_ticket_number = length(unique_ticket_number);
55 row = 1;
56 for record = 1:length_of_unique_ticket_number
57
58 if mod(record, 1000) == 0
59 disp([num2str(record), '/', num2str(length_of_unique_ticket_number)]);
60
61 end
62
63
64 match_index = find(ismember(Oneway_OneStop.TICKET_NBR,unique_ticket_number(record)));
65 length_match = length(match_index);
66
67 if length_match == 2
68
69 seg_1_index = Oneway_OneStop.SEG_ID(match_index) == 1;
70 seg_2_index = Oneway_OneStop.SEG_ID(match_index) == 2;
71
72 One_Stop.TICKET_NBR(row,1) = unique_ticket_number(record);
73 One_Stop.ORIG_ARPT_CD(row,1) = Oneway_OneStop.ORIG_ARPT_CD(match_index(seg_1_index));
74 One_Stop.DEST_ARPT_CD(row,1) = Oneway_OneStop.DEST_ARPT_CD(match_index(seg_2_index));
75 One_Stop.STOP_ARPT(row,1) = Oneway_OneStop.DEST_ARPT_CD(match_index(seg_1_index));
76 One_Stop.Total_Distance(row,1) = Oneway_OneStop.Total_Mile(match_index(seg_1_index));
%SEG_MILE_one_stop(match_index(seg_1_index)) +
SEG_MILE_one_stop(match_index(seg_2_index));
77 One_Stop.DOC_AMT(row,1) = Oneway_OneStop.DOC_AMT(match_index(seg_1_index));
78 One_Stop.Fare_per_Mile(row,1) = Oneway_OneStop.Fare_per_pax_mile(match_index(seg_1_index));
79
```

```
80 row = row+1;
81
82 end
83 end
84
85 %Save the One_Stop variable so it can be used for the international market
86 %without having to generate it again
87 One_Stop_ser = getByteStreamFromArray(One_Stop);
88 save([save_dir,main_delimiter,'\One_Stop.mat'],'One_Stop_ser','-v7.3')
89
90
91 %%
92 %We keep only those records between US and International
93 %Remove records within the continental US only.
94 [~,origin_airport_country_index] = ismember(One_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs);
95 [~,stop_airport_country_index] = ismember(One_Stop.STOP_ARPT,airport_list.Airport_IDs);
96 [~,dest_airport_country_index] = ismember(One_Stop.DEST_ARPT_CD,airport_list.Airport_IDs);
97
98
99 ORIG_ARPT_country = airport_list.Country_IDs(origin_airport_country_index);
100 STOP_ARPT_country = airport_list.Country_IDs(stop_airport_country_index);
101 DEST_ARPT_country = airport_list.Country_IDs(dest_airport_country_index);
102
103 [~ , ORIG_ARPT_country_US] = ismember(ORIG_ARPT_country,'US');
104 [~ , STOP_ARPT_country_US] = ismember(STOP_ARPT_country,'US');
105 [~ , DEST_ARPT_country_US] = ismember(DEST_ARPT_country,'US');
106
107 sum_US_index = ORIG_ARPT_country_US + STOP_ARPT_country_US + DEST_ARPT_country_US;
108
109 delete_continental_us = find(sum_US_index == 3);
110
111 One_Stop.TICKET_NBR(delete_continental_us) = [];
112 One_Stop.ORIG_ARPT_CD(delete_continental_us) = [];
113 One_Stop.DEST_ARPT_CD(delete_continental_us) = [];
114 One_Stop.STOP_ARPT(delete_continental_us) = [];
115 One_Stop.Total_Distance(delete_continental_us) = [];
116 One_Stop.DOC_AMT(delete_continental_us) = [];
117 One_Stop.Fare_per_Mile(delete_continental_us) = [];
118 %%
119 %Remove international only records
120 [~,origin_airport_country_index] = ismember(One_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs);
121 [~,stop_airport_country_index] = ismember(One_Stop.STOP_ARPT,airport_list.Airport_IDs);
122 [~,dest_airport_country_index] = ismember(One_Stop.DEST_ARPT_CD,airport_list.Airport_IDs);
123
124 ORIG_ARPT_country = airport_list.Country_IDs(origin_airport_country_index);
125 STOP_ARPT_country = airport_list.Country_IDs(stop_airport_country_index);
126 DEST_ARPT_country = airport_list.Country_IDs(dest_airport_country_index);
127
128 [~ , ORIG_ARPT_country_US] = ismember(ORIG_ARPT_country,'US');
129 [~ , STOP_ARPT_country_US] = ismember(STOP_ARPT_country,'US');
130 [~ , DEST_ARPT_country_US] = ismember(DEST_ARPT_country,'US');
131
132 sum_US_index = ORIG_ARPT_country_US + STOP_ARPT_country_US + DEST_ARPT_country_US;
133
134 delete_international_only = find(sum_US_index == 0);
135
136 One_Stop.TICKET_NBR(delete_international_only) = [];
137 One_Stop.ORIG_ARPT_CD(delete_international_only) = [];
138 One_Stop.DEST_ARPT_CD(delete_international_only) = [];
139 One_Stop.STOP_ARPT(delete_international_only) = [];
140 One_Stop.Total_Distance(delete_international_only) = [];
141 One_Stop.DOC_AMT(delete_international_only) = [];
142 One_Stop.Fare_per_Mile(delete_international_only) = [];
143
144 %%
145 %Match between nonstop and oneway
```

```
146
147 OD_non_stop = strcat(Oneway_Non_Stop.ORIG_ARPT_CD,'_',Oneway_Non_Stop.DEST_ARPT_CD);
148
149 OD_one_stop = strcat(One_Stop.ORIG_ARPT_CD,'_',One_Stop.DEST_ARPT_CD);
150
151 match_nonstop_onestop_index = ismember(OD_one_stop,OD_non_stop) == 1;
152
153 One_Stop_Match.TICKET_NBR = One_Stop.TICKET_NBR(match_nonstop_onestop_index);
154 One_Stop_Match.ORIG_ARPT_CD = One_Stop.ORIG_ARPT_CD(match_nonstop_onestop_index);
155 One_Stop_Match.DEST_ARPT_CD = One_Stop.DEST_ARPT_CD(match_nonstop_onestop_index);
156 One_Stop_Match.STOP_ARPT = One_Stop.STOP_ARPT(match_nonstop_onestop_index);
157 One_Stop_Match.Total_Distance = One_Stop.Total_Distance(match_nonstop_onestop_index);
158 One_Stop_Match.DOC_AMT = One_Stop.DOC_AMT(match_nonstop_onestop_index);
159 One_Stop_Match.Fare_per_Mile = One_Stop.Fare_per_Mile(match_nonstop_onestop_index);
160
161
162 save(([save_dir,main_delimiter,'One_Stop_Match_US_Int.mat']),'One_Stop_Match')
163
164 return;
165
```

```
1 function [] = Value_of_Time_Calculation_US_Int(save_dir, main_delimiter,Input_Folder_Dir,airport_list)
2
3 min_record_count = 10;
4 layover_time_hrs = 2; %hrs
5
6 load(([save_dir,main_delimiter,'Oneway_Non_Stop_US_Int_ser.mat']),'Oneway_Non_Stop_ser')
7 Oneway_Non_Stop = getArrayFromByteStream(Oneway_Non_Stop_ser);
8 clear Oneway_Non_Stop_ser
9
10 load(([save_dir,main_delimiter,'One_Stop_Match_US_Int.mat']),'One_Stop_Match')
11
12 load(([Input_Folder_Dir,'\VOT\CA_Network_2016_Domestic\Output_Files\A2A_CA_OTD_Network_NonStop_2016.
mat']),'A2A_CA_OTD_Network_NonStop_Ser')
13
14 A2A_CA_OTD_Network_NonStop = getArrayFromByteStream(A2A_CA_OTD_Network_NonStop_Ser);
15 clear A2A_CA_OTD_Network_NonStop_Ser
16
17
load(([Input_Folder_Dir,'\VOT\CA_Network_2016_Domestic\Output_Files\A2A_CA_TravelTime_2Legs_2016.mat']),'A2A_CA_TravelTime_2Legs
_Ser')
18 A2A_CA_TravelTime_2Legs = getArrayFromByteStream(A2A_CA_TravelTime_2Legs_Ser);
19 clear A2A_CA_TravelTime_2Legs_Ser
20
21 load(([Input_Folder_Dir,'\VOT\CA_Network_2016_Domestic\Input_Files\airportlist_oag_2016.mat']),'airportlist_oag')
22 airportlist_oag = cellstr(airportlist_oag);
23
24 load(([Input_Folder_Dir,main_delimiter,'OD_Pairs_Year_2016.mat']),'OD_Pairs_Year_2016')
25 %%
26 %Remove nonstop records for trips with no onestop records data
27 %available
28 OD_non_stop = strcat(Oneway_Non_Stop.ORIG_ARPT_CD,'_',Oneway_Non_Stop.DEST_ARPT_CD);
29
30 OD_one_stop = strcat(One_Stop_Match.ORIG_ARPT_CD,'_',One_Stop_Match.DEST_ARPT_CD);
31
32 [~,match_index] = ismember(OD_non_stop,OD_one_stop);
33 remove = find(match_index==0);
34
35 Oneway_Non_Stop.TICKET_NBR(remove) = [];
36 Oneway_Non_Stop.SEG_ID(remove) = [];
37 Oneway_Non_Stop.Total_Mile(remove) = [];
38 Oneway_Non_Stop.SEG_MILE(remove) = [];
39 Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
40 Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
41 Oneway_Non_Stop.DOC_AMT(remove) = [];
```

```
42 Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
43 Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
44
45 %%
46 %Remove ARC records for one-way OD pairs that are not part of OAG2016
47 OD_non_stop = strcat(Oneway_Non_Stop.ORIG_ARPT_CD,'_',Oneway_Non_Stop.DEST_ARPT_CD);
48
49 OD_one_stop_1 = strcat(One_Stop_Match.ORIG_ARPT_CD,'_',One_Stop_Match.STOP_ARPT);
50
51 OD_one_stop_2 = strcat(One_Stop_Match.STOP_ARPT,'_',One_Stop_Match.DEST_ARPT_CD);
52
53 OD_OAG = strcat(OD_Pairs_Year_2016.DepAirport,'_',OD_Pairs_Year_2016.ArrAirport);
54
55 [~,OAG_non_stop_index] = ismember(OD_non_stop,OD_OAG);
56
57 [~,OAG_one_non_stop_1_index] = ismember(OD_one_stop_1,OD_OAG);
58
59 [~,OAG_one_non_stop_2_index] = ismember(OD_one_stop_2,OD_OAG);
60
61 delete_0 = find(OAG_non_stop_index == 0);
62 delete_1 = find(OAG_one_non_stop_1_index == 0);
63 delete_2 = find(OAG_one_non_stop_2_index == 0);
64 delete_3 = [delete_1 ; delete_2];
65
66 Oneway_Non_Stop.ORIG_ARPT_CD(delete_0) = [];
67 Oneway_Non_Stop.DEST_ARPT_CD(delete_0) = [];
68 Oneway_Non_Stop.DOC_AMT(delete_0) = [];
69 Oneway_Non_Stop.Fare_per_pax_mile(delete_0) = [];
70 Oneway_Non_Stop.SEG_ID(delete_0) = [];
71 Oneway_Non_Stop.Total_Mile(delete_0) = [];
72 Oneway_Non_Stop.SEG_MILE(delete_0) = [];
73 Oneway_Non_Stop.SEG_DOC_AMT(delete_0) = [];
74
75 OD_non_stop(delete_0) = [];
76
77 OD_one_stop_1(delete_3) = [];
78 OD_one_stop_2(delete_3) = [];
79 One_Stop_Match.TICKET_NBR(delete_3) = [];
80 One_Stop_Match.ORIG_ARPT_CD(delete_3) = [];
81 One_Stop_Match.DEST_ARPT_CD(delete_3) = [];
82 One_Stop_Match.STOP_ARPT(delete_3) = [];
83 One_Stop_Match.Total_Distance(delete_3) = [];
84 One_Stop_Match.DOC_AMT(delete_3) = [];
85 One_Stop_Match.Fare_per_Mile(delete_3) = [];
86
87 %Assign travel times to ARC records based on OAG elapsed travel times
88 [~,OAG_non_stop_index] = ismember(OD_non_stop,OD_OAG);
89
90 [~,OAG_one_non_stop_1_index] = ismember(OD_one_stop_1,OD_OAG);
91
92 [~,OAG_one_non_stop_2_index] = ismember(OD_one_stop_2,OD_OAG);
93
94 Travel_Time_non_stop_hrs = OD_Pairs_Year_2016.OD_AverageElapsedTime_hrs(OAG_non_stop_index);
95
96 One_Stop_Match.Travel_Time_hrs = OD_Pairs_Year_2016.OD_AverageElapsedTime_hrs(OAG_one_non_stop_1_index) +
OD_Pairs_Year_2016.
OD_AverageElapsedTime_hrs(OAG_one_non_stop_2_index) + layover_time_hrs;
97 %%
98 %Start of VOT Calculation
99 OD_one_stop = strcat(One_Stop_Match.ORIG_ARPT_CD,'_',One_Stop_Match.DEST_ARPT_CD);
100
101 OD_Combined = unique([OD_non_stop;OD_one_stop]);
102
103 length_of_OD_Combined = length(OD_Combined);
104 row = 1;
105 for OD = 1:length_of_OD_Combined
```

```
106
107 if mod(OD, 50) == 0
108 disp([num2str(OD), '/', num2str(length_of_OD_Combined)]);
109
110 end
111
112 if sum(ismember(OD_non_stop,OD_Combined(OD))) > 0
113
114 if sum(ismember(OD_one_stop,OD_Combined(OD))) > 0
115
116
117 [non_stop_index,~] = ismember(OD_non_stop,OD_Combined(OD));
118 [one_stop_index,~] = ismember(OD_one_stop,OD_Combined(OD));
119
120 non_stop_Fare = Oneway_Non_Stop.SEG_DOC_AMT(non_stop_index);
121 non_stop_travel_time_hrs = Travel_Time_non_stop_hrs(non_stop_index);
122
123 non_stop_Fare_50_percentile = prctile(non_stop_Fare,50);
124 non_stop_travel_time_hrs_mean = mean(non_stop_travel_time_hrs);
125
126 one_stop_Fare = One_Stop_Match.DOC_AMT(one_stop_index);
127 one_stop_travel_time_hrs = One_Stop_Match.Travel_Time_hrs(one_stop_index);
128
129 origin_oag_list = char(OD_Combined(OD));
130 destination_oag_list = cellstr(origin_oag_list(1,5:7));
131 origin_oag_list = cellstr(origin_oag_list(1,1:3));
132
133 [origin_oag_list_index,~] = ismember(airportlist_oag,origin_oag_list);
134 [destination_oag_list_index,~] = ismember(airportlist_oag,destination_oag_list);
135
136 if sum(origin_oag_list_index) ~= 0
137
138 if sum(destination_oag_list_index) ~= 0
139
140 od_stop_index = A2A_CA_OTD_Network_NonStop(origin_oag_list_index,destination_oag_list_index);
141 if isempty(od_stop_index.legs) == 0
142 od_stop_travel_times = A2A_CA_TravelTime_2Legs(origin_oag_list_index,destination_oag_list_index);
143
144 remove = find(od_stop_index.legs == destination_oag_list_index);
145
146 od_stop_index.legs(remove) = [];
147 od_stop_travel_times.legs(remove) = [];
148 if isempty(od_stop_index.legs) == 0
149 stop_no_match_remove = ismember(airportlist_oag(od_stop_index.legs),One_Stop_Match.STOP_ARPT(one_stop_index))==0;
150
151 od_stop_index.legs(stop_no_match_remove) = [];
152 od_stop_travel_times.legs(stop_no_match_remove) = [];
153
154 if isempty(od_stop_travel_times.legs) == 1
155
156 od_stop_travel_times.legs = prctile(one_stop_travel_time_hrs,50);
157
158 end
159
160 end
161
162 end
163
164 end
165
166 end
167
168 if exist('od_stop_index','var') == 0
169
170 clear od_stop_travel_times
171 elseif exist('od_stop_index.legs','var') == 1
```

```
clear od_stop_travel_times

end

if (exist('od_stop_travel_times','var') == 1)

if isempty(one_stop_Fare) == 0

one_stop_Fare_50_percentile = prctile(one_stop_Fare,50);
one_stop_travel_time_hrs_mean = mean(od_stop_travel_times.legs);
clear od_stop_travel_times remove od_stop_index

if ((non_stop_Fare_50_percentile - one_stop_Fare_50_percentile) / (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean))
>= 0

if (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean) >= 1

if length(one_stop_Fare) >= min_record_count && length(non_stop_Fare) >= min_record_count

VOT.OD(row,1) = OD_Combined(OD);
VOT.Mean_Non_Stop_Fare(row,1) = non_stop_Fare_50_percentile;
VOT.Mean_Non_Stop_Travel_Time(row,1) = non_stop_travel_time_hrs_mean;
VOT.Mean_One_Stop_Fare(row,1) = one_stop_Fare_50_percentile;
VOT.Mean_One_Stop_Travel_Time(row,1) = one_stop_travel_time_hrs_mean;
VOT.Delta_Fare(row,1) = (non_stop_Fare_50_percentile - one_stop_Fare_50_percentile);
VOT.Delta_Stop_Travel_Time(row,1) = (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean);

VOT.Value_of_Time(row,1) = (non_stop_Fare_50_percentile - one_stop_Fare_50_percentile) ./ (one_stop_travel_time_hrs_mean -
non_stop_travel_time_hrs_mean);
VOT.Non_stop_count(row,1) = length(non_stop_Fare);
VOT.One_stop_count(row,1) = length(one_stop_Fare);
VOT.OAG_TT_Used(row,1) = 1;
row = row + 1;

clear od_stop_index od_stop_travel_times
end

end
end

end
else

%%
if isempty(one_stop_Fare) == 0
one_stop_Fare_50_percentile = prctile(one_stop_Fare,50);
one_stop_travel_time_hrs_mean = mean(one_stop_travel_time_hrs);

if ((non_stop_Fare_50_percentile - one_stop_Fare_50_percentile) / (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean))
>= 0

if (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean) >= 1

if length(one_stop_Fare) >= min_record_count && length(non_stop_Fare) >= min_record_count

VOT.OD(row,1) = OD_Combined(OD);
VOT.Mean_Non_Stop_Fare(row,1) = non_stop_Fare_50_percentile;
VOT.Mean_Non_Stop_Travel_Time(row,1) = non_stop_travel_time_hrs_mean;
VOT.Mean_One_Stop_Fare(row,1) = one_stop_Fare_50_percentile;
VOT.Delta_Fare(row,1) = (non_stop_Fare_50_percentile - one_stop_Fare_50_percentile);
VOT.Delta_Stop_Travel_Time(row,1) = (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean);
VOT.Value_of_Time(row,1) = (non_stop_Fare_50_percentile - one_stop_Fare_50_percentile) ./ (one_stop_travel_time_hrs_mean -
non_stop_travel_time_hrs_mean);
VOT.Non_stop_count(row,1) = length(non_stop_Fare);
VOT.One_stop_count(row,1) = length(one_stop_Fare);
VOT.OAG_TT_Used(row,1) = 2;
```

```
row = row + 1;

end
end
end
end
end
end
end

clearvars -except OD_non_stop Oneway_Non_Stop...
One_Stop_Match OD_one_stop OD_Combined...
length_of_OD_Combined row VOT LO load_dir OD Travel_Time_non_stop_hrs...
airportlist_oag A2A_CA_OTD_Network_NonStop A2A_CA_TravelTime_2Legs...
min_record_count od_stop_travel_times airport_list OD_Pairs_Year_2016...
save_dir main_delimiter Input_Folder_Dir airport_list;
end


%Save VOT value by one-way OD pair

VOT_by_OD = VOT;

save(([save_dir,main_delimiter,'VOT_by_OD_US_Int.mat']),'VOT_by_OD')

clear VOT
%%
%VOT By Airport

length_of_od = length(VOT_by_OD.OD);

origin_airport = strings(length_of_od,1);

for od = 1:length_of_od

od_char = char(VOT_by_OD.OD(od,1));
origin_airport(od,1) = cellstr(od_char(1:3));

end

unqiue_origin_airport = unique(origin_airport);

length_of_airports = length(unqiue_origin_airport);

for airport = 1:length_of_airports

[airport_index,~] = ismember(origin_airport,unqiue_origin_airport(airport));

records_used = VOT_by_OD.Non_stop_count(airport_index) + VOT_by_OD.One_stop_count(airport_index);

VOT.Value_of_Time(airport,1) = round(sum(VOT_by_OD.Value_of_Time(airport_index) .* records_used) ./ sum(records_used),0);
VOT.Number_of_records(airport,1) = sum(records_used);

end

VOT.Airport = unqiue_origin_airport;

%Save VOT values by Airport
save(([save_dir,main_delimiter,'VOT_by_Aiport_US_Int']),'VOT')

%%
[dep_country,~] = ismember(OD_Pairs_Year_2016.DepIATACtry,'US');
[dest_country,~] = ismember(OD_Pairs_Year_2016.ArrIATACtry,'US');
dep_dest_country_index = find((dep_country + dest_country)==1);

high_demand_index = OD_Pairs_Year_2016.Total_Seats(dep_dest_country_index)>=100000;
```

```
300 Value_of_Time.Airport =
unique([OD_Pairs_Year_2016.DepAirport(dep_dest_country_index(high_demand_index));OD_Pairs_Year_2016.ArrAirport
(dep_dest_country_index(high_demand_index))]);
301
302 length_airport = length(VOT.Airport);
303
304 Value_of_Time.Value((1:length(Value_of_Time.Airport)),1) = round(mean(VOT.Value_of_Time),0);
305 Value_of_Time.Number_of_records((1:length(Value_of_Time.Airport)),1) = 1;
306
307 for airport = 1:length_airport
308
309 [airport_index,~] = ismember(Value_of_Time.Airport,VOT.Airport(airport));
310
311 Value_of_Time.Value(airport_index,1) = VOT.Value_of_Time(airport);
312 Value_of_Time.Number_of_records(airport_index,1) = VOT.Number_of_records(airport);
313 end
314
315 o_airport = Value_of_Time.Airport;
316 d_airport = Value_of_Time.Airport;
317
318 length_of_airport = length(o_airport);
319
320 origin_list = cell(length_of_airport,1);
321 destination_list = cell(length_of_airport,1);
322 m=1;
323 for origin = 1:length_of_airport
324
325 for destination = 1:length_of_airport
326
327 origin_list(m,1) = o_airport(origin);
328 destination_list(m,1) = d_airport(destination);
329 m = m+1;
330 end
331 end
332
333 od_from_vot_matrix = strcat(origin_list,'_',destination_list);
334 od_oag = strcat(OD_Pairs_Year_2016.DepAirport,'_',OD_Pairs_Year_2016.ArrAirport);
335 row = 0;
336 od_dist(1:length_of_airport,1:length_of_airport) = 999999;
337
338 for origin = 1:length_of_airport
339
340 for destination = 1:length_of_airport
341 row = row + 1;
342 if origin == destination
343
344 %do nothing
345
346 elseif sum(ismember(od_oag,od_from_vot_matrix(row))) ~= 0
347
348 [index,~] = ismember(od_oag,od_from_vot_matrix(row));
349 od_dist(origin,destination) = OD_Pairs_Year_2016.DistStMiles(index);
350 else
351
352 [o_match_index,~] = ismember(airport_list.Airport_IDs,o_airport(origin));
353 [d_match_index,~] = ismember(airport_list.Airport_IDs,d_airport(destination));
354
355 lat_o = airport_list.Apt_Lat(o_match_index);
356 lon_o = airport_list.Apt_Lon(o_match_index);
357
358 lat_d = airport_list.Apt_Lat(d_match_index);
359 lon_d = airport_list.Apt_Lon(d_match_index);
360
361 od_dist(origin,destination) = ceil(deg2sm(distance(lat_o,lon_o,lat_d,lon_d)));
362
363 end
```

```
364 end
365 end
366
367 close_by_airports(length_of_airport,1).index = zeros(length_of_airport,1);
368
369 for origin = 1:length_of_airport
370
371 close_by_airports(origin).index = find(od_dist(origin,:) <= 30);
372
373 end
374
375 for origin = 1:length_of_airport
376
377 if isempty(close_by_airports(origin).index) == 0
378
379 Value_of_Time.Value(close_by_airports(origin).index) = ceil(sum(Value_of_Time.Value(close_by_airports(origin).index) .* Value_of_Time.
Number_of_records(close_by_airports(origin).index)) ./ sum(Value_of_Time.Number_of_records(close_by_airports(origin).index)));
380
381 end
382 end
383
384
385 %Save Value of time (by weighted average)
386 save(([save_dir,main_delimiter,'Value_of_Time_US_Int.mat']),'Value_of_Time')
387
388 return;
```

```
1 function [] = Int_VOT_Main_Function(SST_Pre_Processing_Dir, Input_Folder_Dir)
2
3 %% This script calculates the value of time for the Int. market
4
5 main_delimiter = '\';
6 % Input_Folder_Dir = ([local_disc,'..\..\..\..\..\..\SST_2020_Input']);
7 SST_Int_VOT_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'VOT\Int_Market']);
8 save_dir = ([SST_Int_VOT_Dir,main_delimiter,'Output']);
9 Upload_Dir = ('.\ARC_2016\Output');
10 %Create Output directory
11 if exist((save_dir),'dir') == 0
12
13 mkdir(([SST_Int_VOT_Dir,main_delimiter,'Output']))
14
15 end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0
16
17 addpath(([SST_Int_VOT_Dir,main_delimiter,'Output']))
18
19 load(([Input_Folder_Dir,main_delimiter,'airport_list.mat']),'airport_list')
20
21 Oneway_Non_Stop_Records_Int(save_dir, Upload_Dir, main_delimiter,airport_list)
22
23 Rountrip_Non_Stop_Records_Int(save_dir, Upload_Dir, main_delimiter,airport_list)
24
25 Oneway_Onestop_Records_Int(save_dir, Upload_Dir, main_delimiter,airport_list)
26
27 Value_of_Time_Calculation_Int(save_dir, main_delimiter,Input_Folder_Dir,airport_list)
28
29 return;
```

```
1 function [] = Oneway_Non_Stop_Records_Int(save_dir, Upload_Dir, main_delimiter,airport_list)
2
3 load(([Upload_Dir,main_delimiter,'Oneway_Non_Stop_ser.mat']),'Oneway_Non_Stop_ser')
4 Oneway_Non_Stop = getArrayFromByteStream(Oneway_Non_Stop_ser);
5 clear Oneway_Non_Stop_ser
6
7 %%
8
9 %Remove the airports that are not part of both data set (ARC & OAG 2016).
```

```
10
11 missing_origin_airports_index = ismember(Oneway_Non_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs) == 0;
12
13 tickets = Oneway_Non_Stop.TICKET_NBR(missing_origin_airports_index);
14
15 unqiue_unique = unique(tickets);
16
17 remove = ismember(Oneway_Non_Stop.TICKET_NBR,unqiue_unique) ~= 0;
18
19 Oneway_Non_Stop.TICKET_NBR(remove) = [];
20 Oneway_Non_Stop.SEG_ID(remove) = [];
21 Oneway_Non_Stop.Total_Mile(remove) = [];
22 Oneway_Non_Stop.SEG_MILE(remove) = [];
23 Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
24 Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
25 Oneway_Non_Stop.DOC_AMT(remove) = [];
26 Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
27 Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
28 %
29 missing_dest_airports_index = ismember(Oneway_Non_Stop.DEST_ARPT_CD,airport_list.Airport_IDs) == 0;
30
31 tickets = Oneway_Non_Stop.TICKET_NBR(missing_dest_airports_index);
32
33 unqiue_unique = unique(tickets);
34
35 remove = ismember(Oneway_Non_Stop.TICKET_NBR,unqiue_unique) ~= 0;
36
37 Oneway_Non_Stop.TICKET_NBR(remove) = [];
38 Oneway_Non_Stop.SEG_ID(remove) = [];
39 Oneway_Non_Stop.Total_Mile(remove) = [];
40 Oneway_Non_Stop.SEG_MILE(remove) = [];
41 Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
42 Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
43 Oneway_Non_Stop.DOC_AMT(remove) = [];
44 Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
45 Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
46
47 %%
48
49 %Remove US related records.
50 [~,origin_airport_country_index] = ismember(Oneway_Non_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs);
51 [~,dest_airport_country_index] = ismember(Oneway_Non_Stop.DEST_ARPT_CD,airport_list.Airport_IDs);
52
53 ORIG_ARPT_country = airport_list.Country_IDs(origin_airport_country_index);
54 DEST_ARPT_country = airport_list.Country_IDs(dest_airport_country_index);
55
56 [~ , ORIG_ARPT_US] = ismember(ORIG_ARPT_country,'US');
57 [~ , DEST_ARPT_US] = ismember(DEST_ARPT_country,'US');
58
59 OD_US = ORIG_ARPT_US + DEST_ARPT_US;
60
61 delete_US_related = find(OD_US ~= 0);
62
63 Oneway_Non_Stop.TICKET_NBR(delete_US_related) = [];
64 Oneway_Non_Stop.SEG_ID(delete_US_related) = [];
65 Oneway_Non_Stop.Total_Mile(delete_US_related) = [];
66 Oneway_Non_Stop.SEG_MILE(delete_US_related) = [];
67 Oneway_Non_Stop.ORIG_ARPT_CD(delete_US_related) = [];
68 Oneway_Non_Stop.DEST_ARPT_CD(delete_US_related) = [];
69 Oneway_Non_Stop.DOC_AMT(delete_US_related) = [];
70 Oneway_Non_Stop.SEG_DOC_AMT(delete_US_related) = [];
71 Oneway_Non_Stop.Fare_per_pax_mile(delete_US_related) = [];
72 disp('saving')
73
74
75 Oneway_Non_Stop_ser = getByteStreamFromArray(Oneway_Non_Stop);
```

```
clear Oneway_Non_Stop
save(([save_dir,main_delimiter,'Oneway_Non_Stop_Int_ser.mat']),'Oneway_Non_Stop_ser','-v7.3')

return;

function [] = Rountrip_Non_Stop_Records_Int(save_dir, Upload_Dir, main_delimiter,airport_list)

load(([Upload_Dir,main_delimiter,'Roundtrip_NonStop_Part_1_ser.mat']),'Roundtrip_NonStop_Part_1_ser')
Oneway_Non_Stop_Part_1 = getArrayFromByteStream(Roundtrip_NonStop_Part_1_ser);
clear Roundtrip_NonStop_Part_1_ser

load(([Upload_Dir,main_delimiter,'Roundtrip_NonStop_Part_2_ser.mat']),'Roundtrip_NonStop_Part_2_ser')
Oneway_Non_Stop_Part_2 = getArrayFromByteStream(Roundtrip_NonStop_Part_2_ser);
clear Roundtrip_NonStop_Part_2_ser
%%

Oneway_Non_Stop.TICKET_NBR = Oneway_Non_Stop_Part_1.TICKET_NBR;
Oneway_Non_Stop.SEG_ID = Oneway_Non_Stop_Part_1.SEG_ID;
Oneway_Non_Stop.Total_Mile = Oneway_Non_Stop_Part_1.Total_Mile;
Oneway_Non_Stop.SEG_MILE = Oneway_Non_Stop_Part_1.SEG_MILE;
Oneway_Non_Stop.ORIG_ARPT_CD = Oneway_Non_Stop_Part_1.ORIG_ARPT_CD;
Oneway_Non_Stop.DEST_ARPT_CD = Oneway_Non_Stop_Part_2.DEST_ARPT_CD;
Oneway_Non_Stop.DOC_AMT = Oneway_Non_Stop_Part_2.DOC_AMT;
Oneway_Non_Stop.SEG_DOC_AMT = Oneway_Non_Stop_Part_2.SEG_DOC_AMT;
Oneway_Non_Stop.Fare_per_pax_mile = Oneway_Non_Stop_Part_2.Fare_per_pax_mile;

clear Oneway_Non_Stop_Part_1 Oneway_Non_Stop_Part_2

%%
%Remove the airports that are not part of both data set (ARC & OAG 2016).

missing_origin_airports_index = ismember(Oneway_Non_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs) == 0;

tickets = Oneway_Non_Stop.TICKET_NBR(missing_origin_airports_index);

unqiue_unique = unique(tickets);

remove = ismember(Oneway_Non_Stop.TICKET_NBR,unqiue_unique) ~= 0;

Oneway_Non_Stop.TICKET_NBR(remove) = [];
Oneway_Non_Stop.SEG_ID(remove) = [];
Oneway_Non_Stop.Total_Mile(remove) = [];
Oneway_Non_Stop.SEG_MILE(remove) = [];
Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
Oneway_Non_Stop.DOC_AMT(remove) = [];
Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
%
missing_dest_airports_index = ismember(Oneway_Non_Stop.DEST_ARPT_CD,airport_list.Airport_IDs) == 0;

tickets = Oneway_Non_Stop.TICKET_NBR(missing_dest_airports_index);

unqiue_unique = unique(tickets);

remove = ismember(Oneway_Non_Stop.TICKET_NBR,unqiue_unique) ~= 0;

Oneway_Non_Stop.TICKET_NBR(remove) = [];
Oneway_Non_Stop.SEG_ID(remove) = [];
Oneway_Non_Stop.Total_Mile(remove) = [];
Oneway_Non_Stop.SEG_MILE(remove) = [];
Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
Oneway_Non_Stop.DOC_AMT(remove) = [];
Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
```

```
62 %%
63
64 %Remove US related records.
65 [~,origin_airport_country_index] = ismember(Oneway_Non_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs);
66 [~,dest_airport_country_index] = ismember(Oneway_Non_Stop.DEST_ARPT_CD,airport_list.Airport_IDs);
67
68 ORIG_ARPT_country = airport_list.Country_IDs(origin_airport_country_index);
69 DEST_ARPT_country = airport_list.Country_IDs(dest_airport_country_index);
70
71 %%
72 [~ , ORGI_US] = ismember(ORIG_ARPT_country,'US');
73 [~ , DEST_us] = ismember(DEST_ARPT_country,'US');
74 OD_us = ORGI_US + DEST_us;
75 remove = find(OD_us ~= 0);
76
77 Oneway_Non_Stop.TICKET_NBR(remove) = [];
78 Oneway_Non_Stop.SEG_ID(remove) = [];
79 Oneway_Non_Stop.Total_Mile(remove) = [];
80 Oneway_Non_Stop.SEG_MILE(remove) = [];
81 Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
82 Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
83 Oneway_Non_Stop.DOC_AMT(remove) = [];
84 Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
85 Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
86
87 Roundtrip_Non_Stop_ser = getByteStreamFromArray(Oneway_Non_Stop);
88 clear Oneway_Non_Stop
89 save(([save_dir,main_delimiter,'Roundtrip_Non_Stop_Int_ser.mat']),'Roundtrip_Non_Stop_ser','-v7.3')
90
91 return;
```

```
1 function [] = Oneway_Onestop_Records_Int(save_dir, Upload_Dir, main_delimiter,airport_list)
2
3 %Altohugh we are dealing with oneway_nonstop records. We want to keep does
4 %one-way OD pairs that appears in both files (onestop and nonstop). For
5 %that reason, we upload the oneway_non_stop output
6 load(([save_dir,main_delimiter,'Oneway_Non_Stop_Int_ser.mat']),'Oneway_Non_Stop_ser')
7 Oneway_Non_Stop = getArrayFromByteStream(Oneway_Non_Stop_ser);
8 clear Oneway_Non_Stop_ser
9
10 load(([Upload_Dir,main_delimiter,'Oneway_OneStop_ser.mat']),'Oneway_OneStop_ser')
11 Oneway_OneStop = getArrayFromByteStream(Oneway_OneStop_ser);
12 clear Oneway_OneStop_ser
13
14 load(('.\SST_Pre_Processing\VOT\US_Int_Market\Output\One_Stop.mat'),'One_Stop_ser')
15 One_Stop = getArrayFromByteStream(One_Stop_ser);
16 clear One_Stop_ser
17
18 %%
19
20 %Remove the airports that are not part of both data set (ARC & OAG 2016).
21 missing_origin_airports_index = ismember(Oneway_OneStop.ORIG_ARPT_CD,airport_list.Airport_IDs) == 0;
22
23 tickets = Oneway_OneStop.TICKET_NBR(missing_origin_airports_index);
24 unqiue_unique = unique(tickets);
25 remove = ismember(Oneway_OneStop.TICKET_NBR,unqiue_unique) ~= 0;
26
27 Oneway_OneStop.TICKET_NBR(remove) = [];
28 Oneway_OneStop.SEG_ID(remove) = [];
29 Oneway_OneStop.SEG_MILE(remove) = [];
30 Oneway_OneStop.Total_Mile(remove) = [];
31 Oneway_OneStop.ORIG_ARPT_CD(remove) = [];
32 Oneway_OneStop.DEST_ARPT_CD(remove) = [];
33 Oneway_OneStop.DOC_AMT(remove) = [];
34 Oneway_OneStop.SEG_DOC_AMT(remove) = [];
35 Oneway_OneStop.Fare_per_pax_mile(remove) = [];
```

```
36
37 missing_dest_airports_index = ismember(Oneway_OneStop.DEST_ARPT_CD,airport_list.Airport_IDs) == 0;
38
39 tickets = Oneway_OneStop.TICKET_NBR(missing_dest_airports_index);
40 unqiue_unique = unique(tickets);
41 remove = ismember(Oneway_OneStop.TICKET_NBR,unqiue_unique) ~= 0;
42
43 Oneway_OneStop.TICKET_NBR(remove) = [];
44 Oneway_OneStop.SEG_ID(remove) = [];
45 Oneway_OneStop.SEG_MILE(remove) = [];
46 Oneway_OneStop.Total_Mile(remove) = [];
47 Oneway_OneStop.ORIG_ARPT_CD(remove) = [];
48 Oneway_OneStop.DEST_ARPT_CD(remove) = [];
49 Oneway_OneStop.DOC_AMT(remove) = [];
50 Oneway_OneStop.SEG_DOC_AMT(remove) = [];
51 Oneway_OneStop.Fare_per_pax_mile(remove) = [];
52
53 %%
54
55 [~,origin_airport_country_index] = ismember(One_Stop.ORIG_ARPT_CD,airport_list.Airport_IDs);
56 [~,stop_airport_country_index] = ismember(One_Stop.STOP_ARPT,airport_list.Airport_IDs);
57 [~,dest_airport_country_index] = ismember(One_Stop.DEST_ARPT_CD,airport_list.Airport_IDs);
58
59 ORIG_ARPT_country = airport_list.Country_IDs(origin_airport_country_index);
60 STOP_ARPT_country = airport_list.Country_IDs(stop_airport_country_index);
61 DEST_ARPT_country = airport_list.Country_IDs(dest_airport_country_index);
62
63 [~ , ORIG_ARPT_country_US] = ismember(ORIG_ARPT_country,'US');
64 [~ , STOP_ARPT_country_US] = ismember(STOP_ARPT_country,'US');
65 [~ , DEST_ARPT_country_US] = ismember(DEST_ARPT_country,'US');
66
67 sum_US_index = ORIG_ARPT_country_US + STOP_ARPT_country_US + DEST_ARPT_country_US;
68
69 delete_continental_us = find(sum_US_index ~= 0);
70
71 if isempty(delete_continental_us) == 0
72
73 One_Stop.TICKET_NBR(delete_continental_us) = [];
74 One_Stop.ORIG_ARPT_CD(delete_continental_us) = [];
75 One_Stop.DEST_ARPT_CD(delete_continental_us) = [];
76 One_Stop.STOP_ARPT(delete_continental_us) = [];
77 One_Stop.Total_Distance(delete_continental_us) = [];
78 One_Stop.DOC_AMT(delete_continental_us) = [];
79 One_Stop.Fare_per_Mile(delete_continental_us) = [];
80
81 end
82
83 %%
84 %Match between nonstop and oneway
85
86 OD_non_stop = strcat(Oneway_Non_Stop.ORIG_ARPT_CD,'_',Oneway_Non_Stop.DEST_ARPT_CD);
87
88 OD_one_stop = strcat(One_Stop.ORIG_ARPT_CD,'_',One_Stop.DEST_ARPT_CD);
89
90 match_nonstop_onestop_index = ismember(OD_one_stop,OD_non_stop) == 1;
91
92 One_Stop_Match.TICKET_NBR = One_Stop.TICKET_NBR(match_nonstop_onestop_index);
93 One_Stop_Match.ORIG_ARPT_CD = One_Stop.ORIG_ARPT_CD(match_nonstop_onestop_index);
94 One_Stop_Match.DEST_ARPT_CD = One_Stop.DEST_ARPT_CD(match_nonstop_onestop_index);
95 One_Stop_Match.STOP_ARPT = One_Stop.STOP_ARPT(match_nonstop_onestop_index);
96 One_Stop_Match.Total_Distance = One_Stop.Total_Distance(match_nonstop_onestop_index);
97 One_Stop_Match.DOC_AMT = One_Stop.DOC_AMT(match_nonstop_onestop_index);
98 One_Stop_Match.Fare_per_Mile = One_Stop.Fare_per_Mile(match_nonstop_onestop_index);
99
100 save(([save_dir,main_delimiter,'One_Stop_Match_Int.mat']),'One_Stop_Match')
101
```

```
102 return;
103
1 function [] = Value_of_Time_Calculation_Int(save_dir, main_delimiter,Input_Folder_Dir,airport_list)
2
3 min_record_count = 10;
4 layover_time_hrs = 2; %hrs
5
6 load(([save_dir,main_delimiter,'Oneway_Non_Stop_Int_ser.mat']),'Oneway_Non_Stop_ser')
7 Oneway_Non_Stop = getArrayFromByteStream(Oneway_Non_Stop_ser);
8 clear Oneway_Non_Stop_ser
9
10 load(([save_dir,main_delimiter,'One_Stop_Match_Int.mat']),'One_Stop_Match')
11
12 load(([Input_Folder_Dir,'\VOT\CA_Network_2016_Domestic\Output_Files\A2A_CA_OTD_Network_NonStop_2016.
mat']),'A2A_CA_OTD_Network_NonStop_Ser')
13
14 A2A_CA_OTD_Network_NonStop = getArrayFromByteStream(A2A_CA_OTD_Network_NonStop_Ser);
15 clear A2A_CA_OTD_Network_NonStop_Ser
16
17
load(([Input_Folder_Dir,'\VOT\CA_Network_2016_Domestic\Output_Files\A2A_CA_TravelTime_2Legs_2016.mat']),'A2A_CA_TravelTime_2Legs
_Ser')
18 A2A_CA_TravelTime_2Legs = getArrayFromByteStream(A2A_CA_TravelTime_2Legs_Ser);
19 clear A2A_CA_TravelTime_2Legs_Ser
20
21 load(([Input_Folder_Dir,'\VOT\CA_Network_2016_Domestic\Input_Files\airportlist_oag_2016.mat']),'airportlist_oag')
22 airportlist_oag = cellstr(airportlist_oag);
23
24 load(([Input_Folder_Dir,main_delimiter,'OD_Pairs_Year_2016.mat']),'OD_Pairs_Year_2016')
25 %%
26 %Remove nonstop records for trips with no onestop records data
27 %available
28 OD_non_stop = strcat(Oneway_Non_Stop.ORIG_ARPT_CD,'_',Oneway_Non_Stop.DEST_ARPT_CD);
29
30 OD_one_stop = strcat(One_Stop_Match.ORIG_ARPT_CD,'_',One_Stop_Match.DEST_ARPT_CD);
31
32 [~,match_index] = ismember(OD_non_stop,OD_one_stop);
33 remove = find(match_index==0);
34
35 Oneway_Non_Stop.TICKET_NBR(remove) = [];
36 Oneway_Non_Stop.SEG_ID(remove) = [];
37 Oneway_Non_Stop.Total_Mile(remove) = [];
38 Oneway_Non_Stop.SEG_MILE(remove) = [];
39 Oneway_Non_Stop.ORIG_ARPT_CD(remove) = [];
40 Oneway_Non_Stop.DEST_ARPT_CD(remove) = [];
41 Oneway_Non_Stop.DOC_AMT(remove) = [];
42 Oneway_Non_Stop.SEG_DOC_AMT(remove) = [];
43 Oneway_Non_Stop.Fare_per_pax_mile(remove) = [];
44
45 %%
46 %Remove ARC records for one-way OD pairs that are not part of OAG2016
47 OD_non_stop = strcat(Oneway_Non_Stop.ORIG_ARPT_CD,'_',Oneway_Non_Stop.DEST_ARPT_CD);
48
49 %%
50 OD_one_stop_1 = strcat(One_Stop_Match.ORIG_ARPT_CD,'_',One_Stop_Match.STOP_ARPT);
51
52 OD_one_stop_2 = strcat(One_Stop_Match.STOP_ARPT,'_',One_Stop_Match.DEST_ARPT_CD);
53
54 OD_OAG = strcat(OD_Pairs_Year_2016.DepAirport,'_',OD_Pairs_Year_2016.ArrAirport);
55
56 [~,OAG_non_stop_index] = ismember(OD_non_stop,OD_OAG);
57
58 [~,OAG_one_non_stop_1_index] = ismember(OD_one_stop_1,OD_OAG);
59
60 [~,OAG_one_non_stop_2_index] = ismember(OD_one_stop_2,OD_OAG);
61
```

```
62 delete_0 = find(OAG_non_stop_index == 0);
63 delete_1 = find(OAG_one_non_stop_1_index == 0);
64 delete_2 = find(OAG_one_non_stop_2_index == 0);
65 delete_3 = [delete_1 ; delete_2];
66
67 Oneway_Non_Stop.ORIG_ARPT_CD(delete_0) = [];
68 Oneway_Non_Stop.DEST_ARPT_CD(delete_0) = [];
69 Oneway_Non_Stop.DOC_AMT(delete_0) = [];
70 Oneway_Non_Stop.Fare_per_pax_mile(delete_0) = [];
71 Oneway_Non_Stop.SEG_ID(delete_0) = [];
72 Oneway_Non_Stop.Total_Mile(delete_0) = [];
73 Oneway_Non_Stop.SEG_MILE(delete_0) = [];
74 Oneway_Non_Stop.SEG_DOC_AMT(delete_0) = [];
75
76 OD_non_stop(delete_0) = [];
77
78 OD_one_stop_1(delete_3) = [];
79 OD_one_stop_2(delete_3) = [];
80 One_Stop_Match.TICKET_NBR(delete_3) = [];
81 One_Stop_Match.ORIG_ARPT_CD(delete_3) = [];
82 One_Stop_Match.DEST_ARPT_CD(delete_3) = [];
83 One_Stop_Match.STOP_ARPT(delete_3) = [];
84 One_Stop_Match.Total_Distance(delete_3) = [];
85 One_Stop_Match.DOC_AMT(delete_3) = [];
86 One_Stop_Match.Fare_per_Mile(delete_3) = [];
87
88 %Assign travel times to ARC records based on OAG elapsed travel times
89
90 [~,OAG_non_stop_index] = ismember(OD_non_stop,OD_OAG);
91
92 [~,OAG_one_non_stop_1_index] = ismember(OD_one_stop_1,OD_OAG);
93
94 [~,OAG_one_non_stop_2_index] = ismember(OD_one_stop_2,OD_OAG);
95
96 Travel_Time_non_stop_hrs = OD_Pairs_Year_2016.OD_AverageElapsedTime_hrs(OAG_non_stop_index);
97
98 One_Stop_Match.Travel_Time_hrs = OD_Pairs_Year_2016.OD_AverageElapsedTime_hrs(OAG_one_non_stop_1_index) +
OD_Pairs_Year_2016.
OD_AverageElapsedTime_hrs(OAG_one_non_stop_2_index) + layover_time_hrs; %1hrs of layover is added
99 %%
100 %Start of VOT Calculation
101
102 OD_one_stop = strcat(One_Stop_Match.ORIG_ARPT_CD,'_',One_Stop_Match.DEST_ARPT_CD);
103
104 OD_Combined = unique([OD_non_stop;OD_one_stop]);
105
106 length_of_OD_Combined = length(OD_Combined);
107 row = 1;
108 for OD = 1:length_of_OD_Combined
109
110
111 if mod(OD, 50) == 0
112 disp([num2str(OD), '/', num2str(length_of_OD_Combined)]);
113
114 end
115
116 if sum(ismember(OD_non_stop,OD_Combined(OD))) > 0
117
118 if sum(ismember(OD_one_stop,OD_Combined(OD))) > 0
119
120 [non_stop_index,~] = ismember(OD_non_stop,OD_Combined(OD));
121 [one_stop_index,~] = ismember(OD_one_stop,OD_Combined(OD));
122
123 non_stop_Fare = Oneway_Non_Stop.SEG_DOC_AMT(non_stop_index);
124 non_stop_travel_time_hrs = Travel_Time_non_stop_hrs(non_stop_index);
125
```

```
126 non_stop_Fare_50_percentile = prctile(non_stop_Fare,50);
127 non_stop_travel_time_hrs_mean = mean(non_stop_travel_time_hrs);
128
129 one_stop_Fare = One_Stop_Match.DOC_AMT(one_stop_index);
130 one_stop_travel_time_hrs = One_Stop_Match.Travel_Time_hrs(one_stop_index);
131
132 origin_oag_list = char(OD_Combined(OD));
133 destination_oag_list = cellstr(origin_oag_list(1,5:7));
134 origin_oag_list = cellstr(origin_oag_list(1,1:3));
135
136 [origin_oag_list_index,~] = ismember(airportlist_oag,origin_oag_list);
137 [destination_oag_list_index,~] = ismember(airportlist_oag,destination_oag_list);
138
139 if sum(origin_oag_list_index) ~= 0
140
141 if sum(destination_oag_list_index) ~= 0
142
143 od_stop_index = A2A_CA_OTD_Network_NonStop(origin_oag_list_index,destination_oag_list_index);
144 if isempty(od_stop_index.legs) == 0
145 od_stop_travel_times = A2A_CA_TravelTime_2Legs(origin_oag_list_index,destination_oag_list_index);
146
147 remove = find(od_stop_index.legs == destination_oag_list_index);
148
149 od_stop_index.legs(remove) = [];
150 od_stop_travel_times.legs(remove) = [];
151 if isempty(od_stop_index.legs) == 0
152 stop_no_match_remove = ismember(airportlist_oag(od_stop_index.legs),One_Stop_Match.STOP_ARPT(one_stop_index))==0;
153
154 od_stop_index.legs(stop_no_match_remove) = [];
155 od_stop_travel_times.legs(stop_no_match_remove) = [];
156
157 if isempty(od_stop_travel_times.legs) == 1
158
159 od_stop_travel_times.legs = prctile(one_stop_travel_time_hrs,50);
160
161 end
162
163 end
164
165 end
166 end
167
168 end
169
170 if exist('od_stop_index','var') == 0
171
172 clear od_stop_travel_times
173 elseif exist('od_stop_index.legs','var') == 1
174 clear od_stop_travel_times
175
176 end
177
178 if (exist('od_stop_travel_times','var') == 1)
179
180 %%
181
182 if isempty(one_stop_Fare) == 0
183
184 one_stop_Fare_50_percentile = prctile(one_stop_Fare,50);
185 one_stop_travel_time_hrs_mean = mean(od_stop_travel_times.legs);
186
187 clear od_stop_travel_times remove od_stop_index
188
189 if ((non_stop_Fare_50_percentile - one_stop_Fare_50_percentile) / (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean))
>= 0
190
```

```
191 if (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean) >= 1
192
193 if length(one_stop_Fare) >= min_record_count && length(non_stop_Fare) >= min_record_count
194
195 VOT.OD(row,1) = OD_Combined(OD);
196 VOT.Mean_Non_Stop_Fare(row,1) = non_stop_Fare_50_percentile;
197 VOT.Mean_Non_Stop_Travel_Time(row,1) = non_stop_travel_time_hrs_mean;
198 VOT.Mean_One_Stop_Fare(row,1) = one_stop_Fare_50_percentile;
199 VOT.Mean_One_Stop_Travel_Time(row,1) = one_stop_travel_time_hrs_mean;
200 VOT.Delta_Fare(row,1) = (non_stop_Fare_50_percentile - one_stop_Fare_50_percentile);
201 VOT.Delta_Stop_Travel_Time(row,1) = (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean);
202
203 VOT.Value_of_Time(row,1) = (non_stop_Fare_50_percentile - one_stop_Fare_50_percentile) ./ (one_stop_travel_time_hrs_mean -
non_stop_travel_time_hrs_mean);
204 VOT.Non_stop_count(row,1) = length(non_stop_Fare);
205 VOT.One_stop_count(row,1) = length(one_stop_Fare);
206 VOT.OAG_TT_Used(row,1) = 1;
207 row = row + 1;
208 clear od_stop_index od_stop_travel_times
209 end
210
211 end
212 end
213
214 end
215
216 else
217
218 if isempty(one_stop_Fare) == 0
219 one_stop_Fare_50_percentile = prctile(one_stop_Fare,50);
220 one_stop_travel_time_hrs_mean = mean(one_stop_travel_time_hrs);
221
222 if ((non_stop_Fare_50_percentile - one_stop_Fare_50_percentile) / (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean))
>= 0
223
224 if (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean) >= 1
225
226 if length(one_stop_Fare) >= min_record_count && length(non_stop_Fare) >= min_record_count
227
228 VOT.OD(row,1) = OD_Combined(OD);
229 VOT.Mean_Non_Stop_Fare(row,1) = non_stop_Fare_50_percentile;
230 VOT.Mean_Non_Stop_Travel_Time(row,1) = non_stop_travel_time_hrs_mean;
231 VOT.Mean_One_Stop_Fare(row,1) = one_stop_Fare_50_percentile;
232 VOT.Delta_Fare(row,1) = (non_stop_Fare_50_percentile - one_stop_Fare_50_percentile);
233 VOT.Delta_Stop_Travel_Time(row,1) = (one_stop_travel_time_hrs_mean - non_stop_travel_time_hrs_mean);
234
235 VOT.Value_of_Time(row,1) = (non_stop_Fare_50_percentile - one_stop_Fare_50_percentile) ./ (one_stop_travel_time_hrs_mean -
non_stop_travel_time_hrs_mean);
236 VOT.Non_stop_count(row,1) = length(non_stop_Fare);
237 VOT.One_stop_count(row,1) = length(one_stop_Fare);
238 VOT.OAG_TT_Used(row,1) = 2;
239
240 row = row + 1;
241
242
243 end
244 end
245 end
246 end
247 end
248 end
249 end
250
251 clearvars -except OD_non_stop Oneway_Non_Stop...
252 One_Stop_Match OD_one_stop OD_Combined...
253 length_of_OD_Combined row VOT LO load_dir OD Travel_Time_non_stop_hrs...
```

```
airportlist_oag A2A_CA_OTD_Network_NonStop A2A_CA_TravelTime_2Legs...
min_record_count od_stop_travel_times OD_Pairs_Year_2016...
save_dir main_delimiter Input_Folder_Dir airport_list;
end


%Save VOT value by one-way OD pair

VOT_by_OD = VOT;

save([save_dir,main_delimiter,'VOT_by_OD_Int.mat']),'VOT_by_OD')

clear VOT
%%
%VOT By Airport

length_of_od = length(VOT_by_OD.OD);

origin_airport = strings(length_of_od,1);

for od = 1:length_of_od

od_char = char(VOT_by_OD.OD(od,1));
origin_airport(od,1) = cellstr(od_char(1:3));

end

unqiue_origin_airport = unique(origin_airport);

length_of_airports = length(unqiue_origin_airport);

for airport = 1:length_of_airports

[airport_index,~] = ismember(origin_airport,unqiue_origin_airport(airport));

records_used = VOT_by_OD.Non_stop_count(airport_index) + VOT_by_OD.One_stop_count(airport_index);

VOT.Value_of_Time(airport,1) = round(sum(VOT_by_OD.Value_of_Time(airport_index) .* records_used) ./ sum(records_used),0);
VOT.Number_of_records(airport,1) = sum(records_used);

end

VOT.Airport = unqiue_origin_airport;

%Save VOT values by Airport
save([save_dir,main_delimiter,'VOT_by_Aiport_Int']),'VOT')

%%
[dep_country,~] = ismember(OD_Pairs_Year_2016.DepIATACtry,'US');
[dest_country,~] = ismember(OD_Pairs_Year_2016.ArrIATACtry,'US');
dep_dest_country_index = find((dep_country + dest_country)==0);

high_demand_index = OD_Pairs_Year_2016.Total_Seats(dep_dest_country_index)>=100000;
Value_of_Time.Airport =
unique([OD_Pairs_Year_2016.DepAirport(dep_dest_country_index(high_demand_index));OD_Pairs_Year_2016.ArrAirport
(dep_dest_country_index(high_demand_index))]);

length_airport = length(VOT.Airport);

Value_of_Time.Value((1:length(Value_of_Time.Airport)),1) = round(mean(VOT.Value_of_Time),0);
Value_of_Time.Number_of_records((1:length(Value_of_Time.Airport)),1) = 1;

for airport = 1:length_airport

[airport_index,~] = ismember(Value_of_Time.Airport,VOT.Airport(airport));

```

```
318
319 Value_of_Time.Value(airport_index,1) = VOT.Value_of_Time(airport);
320 Value_of_Time.Number_of_records(airport_index,1) = VOT.Number_of_records(airport);
321 end
322
323 o_airport = Value_of_Time.Airport;
324 d_airport = Value_of_Time.Airport;
325
326 length_of_airport = length(o_airport);
327
328 origin_list = cell(length_of_airport,1);
329 destination_list = cell(length_of_airport,1);
330 m=1;
331 for origin = 1:length_of_airport
332
333 for destination = 1:length_of_airport
334
335 origin_list(m,1) = o_airport(origin);
336 destination_list(m,1) = d_airport(destination);
337 m = m+1;
338 end
339 end
340
341 od_from_vot_matrix = strcat(origin_list,'_',destination_list);
342 od_oag = strcat(OD_Pairs_Year_2016.DepAirport,'_',OD_Pairs_Year_2016.ArrAirport);
343 row = 0;
344 od_dist(1:length_of_airport,1:length_of_airport) = 999999;
345
346 for origin = 1:length_of_airport
347
348 for destination = 1:length_of_airport
349 row = row + 1;
350 if origin == destination
351
352 %do nothing
353
354 elseif sum(ismember(od_oag,od_from_vot_matrix(row))) ~= 0
355
356 [index,~] = ismember(od_oag,od_from_vot_matrix(row));
357 od_dist(origin,destination) = OD_Pairs_Year_2016.DistStMiles(index);
358 else
359
360 [o_match_index,~] = ismember(airport_list.Airport_IDs,o_airport(origin));
361 [d_match_index,~] = ismember(airport_list.Airport_IDs,d_airport(destination));
362
363 lat_o = airport_list.Apt_Lat(o_match_index);
364 lon_o = airport_list.Apt_Lon(o_match_index);
365
366 lat_d = airport_list.Apt_Lat(d_match_index);
367 lon_d = airport_list.Apt_Lon(d_match_index);
368
369 od_dist(origin,destination) = ceil(deg2sm(distance(lat_o,lon_o,lat_d,lon_d)));
370
371 end
372 end
373 end
374
375 close_by_airports(length_of_airport,1).index = zeros(length_of_airport,1);
376
377 for origin = 1:length_of_airport
378
379 close_by_airports(origin).index = find(od_dist(origin,:) <= 30);
380
381 end
382
383
```

```
384 for origin = 1:length_of_airport
385
386 if isempty(close_by_airports(origin).index) == 0
387
388 Value_of_Time.Value(close_by_airports(origin).index) = ceil(sum(Value_of_Time.Value(close_by_airports(origin).index) .* Value_of_Time.
Number_of_records(close_by_airports(origin).index)) ./ sum(Value_of_Time.Number_of_records(close_by_airports(origin).index)));
389
390 end
391 end
392
393
394 %Save Value of time (by weighted average)
395 save(([save_dir,main_delimiter,'Value_of_Time_Int.mat']),'Value_of_Time')
396
397 return;
```

```
1 function [] = Fare_per_Mile_CDF_Data_Main_Function_US(SST_Pre_Processing_Dir,mach_overland, mach_overwater,acft_seating_capacity)
2 %% This script generates the fare per mile CDF data from ARC 2016 - US Market
3
4 main_delimiter = '\';
5 SST_US_CDF_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'CDF\US_Market']);
6 save_dir = ([SST_US_CDF_Dir,main_delimiter,'Output']);
7 Upload_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'VOT\US_Market\Output']);
8 TravelTime_Upload_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'SST_Travel_Times\Output']);
9 %Create Output directory
10 if exist((save_dir),'dir') == 0
11
12 mkdir(([SST_US_CDF_Dir,main_delimiter,'Output']))
13
14 end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0
15
16 addpath(([SST_US_CDF_Dir,main_delimiter,'Output']))
17
18 Fare_Per_Mile_Paid_General_US(Upload_Dir, main_delimiter, save_dir)
19
20 Fare_per_mile_paid_by_OD_Original_US(Upload_Dir, TravelTime_Upload_Dir, main_delimiter, save_dir, mach_overland, mach_overwater,
acft_seating_capacity)
21
22 return;
```

```
1
2 function [] = Fare_Per_Mile_Paid_General_US(Upload_Dir, main_delimiter, save_dir)
3
4 %General Fare per Mile Paid Curve
5
6 load(([Upload_Dir,main_delimiter,'One_Stop_Match_US.mat']),'One_Stop_Match')
7
8 Cost_per_Mile_Paid_one_stop = One_Stop_Match.Fare_per_Mile;
9
10 load(([Upload_Dir,main_delimiter,'Oneway_Non_Stop_US_ser.mat']),'Oneway_Non_Stop_ser')
11 Oneway_NonStop = getArrayFromByteStream(Oneway_Non_Stop_ser);
12 clear Oneway_Non_Stop_ser
13 %%
14
15 load(([Upload_Dir,main_delimiter,'Roundtrip_Non_Stop_US_ser.mat']),'Roundtrip_Non_Stop_ser')
16 Roundtrip_NonStop = getArrayFromByteStream(Roundtrip_Non_Stop_ser);
17 clear Oneway_Non_Stop_ser
18
19 %Combaine Rountrip nonstop and oneway nonstop records
20 Oneway_Non_Stop.TICKET_NBR = [Oneway_NonStop.TICKET_NBR; Roundtrip_NonStop.TICKET_NBR];
21 Oneway_Non_Stop.SEG_ID = [Oneway_NonStop.SEG_ID; Roundtrip_NonStop.SEG_ID];
22 Oneway_Non_Stop.Total_Mile = [Oneway_NonStop.Total_Mile; Roundtrip_NonStop.Total_Mile];
23 Oneway_Non_Stop.SEG_MILE = [Oneway_NonStop.SEG_MILE; Roundtrip_NonStop.SEG_MILE];
24 Oneway_Non_Stop.ORIG_ARPT_CD = [Oneway_NonStop.ORIG_ARPT_CD; Roundtrip_NonStop.ORIG_ARPT_CD];
25 Oneway_Non_Stop.DEST_ARPT_CD = [Oneway_NonStop.DEST_ARPT_CD; Roundtrip_NonStop.DEST_ARPT_CD];
26 Oneway_Non_Stop.DOC_AMT = [Oneway_NonStop.DOC_AMT; Roundtrip_NonStop.DOC_AMT];
```

```
27 Oneway_Non_Stop.SEG_DOC_AMT = [Oneway_NonStop.SEG_DOC_AMT; Roundtrip_NonStop.SEG_DOC_AMT];
28 Oneway_Non_Stop.Fare_per_pax_mile = [Oneway_NonStop.Fare_per_pax_mile; Roundtrip_NonStop.Fare_per_pax_mile];
29 clear Oneway_NonStop Roundtrip_NonStop
30 %%
31 Cost_per_Mile_Paid_non_stop = Oneway_Non_Stop.Fare_per_pax_mile;
32
33 Cost_per_Mile_Paid_Premium = round([Cost_per_Mile_Paid_one_stop;Cost_per_Mile_Paid_non_stop],2);
34
35 total_records = length(Cost_per_Mile_Paid_Premium);
36 unique_fare_per_mile_paid = unique(Cost_per_Mile_Paid_Premium);
37
38 length_of_unique_fare_per_mile_paid = length(unique_fare_per_mile_paid);
39
40 Fare_per_Mile_Paid_Percentage_Premium = zeros(length_of_unique_fare_per_mile_paid,1);
41
42 for record = 1:length_of_unique_fare_per_mile_paid
43
44 Fare_per_Mile_Paid_Percentage_Premium(record) = length(find(Cost_per_Mile_Paid_Premium == unique_fare_per_mile_paid(record))) ./
total_records ;
45
46 end
47 %%
48
49 for record = 1:length_of_unique_fare_per_mile_paid
50
51 Fare_per_Mile_Paid_Premium.Fare(record,1) = unique_fare_per_mile_paid(record);
52
53 Fare_per_Mile_Paid_Premium.Percentage(record,1) = length(find(Cost_per_Mile_Paid_Premium == unique_fare_per_mile_paid(record))) ./
total_records ;
54
55 if record == 1
56
57 Fare_per_Mile_Paid_Premium.Cumulative(record,1) = 0;
58
59 elseif record == 2
60
61 Fare_per_Mile_Paid_Premium.Cumulative(record,1) = Fare_per_Mile_Paid_Premium.Percentage(record) +
Fare_per_Mile_Paid_Premium.Percentage
(1);
62
63 else
64
65 Fare_per_Mile_Paid_Premium.Cumulative(record,1) = Fare_per_Mile_Paid_Premium.Percentage(record) +
Fare_per_Mile_Paid_Premium.Cumulative
(record - 1);
66
67 end
68
69 end
70 save(([save_dir,main_delimiter,'Fare_per_Mile_Paid_Premium_US.mat']),'Fare_per_Mile_Paid_Premium')
71
72 return;
1
2 function [] = Fare_per_mile_paid_by_OD_Original_US(Upload_Dir, TravelTime_Upload_Dir, main_delimiter, save_dir, mach_overland,
mach_overwater,
acft_seating_capacity)
3
4
5 load(([TravelTime_Upload_Dir,main_delimiter,'Travel_Times_with_Ground_Times_1_',num2str((mach_overland-
1)*10),'M_1_',num2str((mach_overwater-
1)*10),'M_',num2str(acft_seating_capacity),'_Seats.mat']),'Travel_Times')
6
7 load(([Upload_Dir,main_delimiter,'One_Stop_Match_US.mat']),'One_Stop_Match')
8
9 Cost_per_Mile_Paid_one_stop = One_Stop_Match.Fare_per_Mile;
10
```

```
load(([Upload_Dir,main_delimiter,'Oneway_Non_Stop_US_ser.mat']),'Oneway_Non_Stop_ser')
Oneway_NonStop = getArrayFromByteStream(Oneway_Non_Stop_ser);
clear Oneway_Non_Stop_ser
%%

load(([Upload_Dir,main_delimiter,'Roundtrip_Non_Stop_US_ser.mat']),'Roundtrip_Non_Stop_ser')
Roundtrip_NonStop = getArrayFromByteStream(Roundtrip_Non_Stop_ser);
clear Oneway_Non_Stop_ser

%Combaine Rountrip nonstop and oneway nonstop records
Oneway_Non_Stop.TICKET_NBR = [Oneway_NonStop.TICKET_NBR; Roundtrip_NonStop.TICKET_NBR];
Oneway_Non_Stop.SEG_ID = [Oneway_NonStop.SEG_ID; Roundtrip_NonStop.SEG_ID];
Oneway_Non_Stop.Total_Mile = [Oneway_NonStop.Total_Mile; Roundtrip_NonStop.Total_Mile];
Oneway_Non_Stop.SEG_MILE = [Oneway_NonStop.SEG_MILE; Roundtrip_NonStop.SEG_MILE];
Oneway_Non_Stop.ORIG_ARPT_CD = [Oneway_NonStop.ORIG_ARPT_CD; Roundtrip_NonStop.ORIG_ARPT_CD];
Oneway_Non_Stop.DEST_ARPT_CD = [Oneway_NonStop.DEST_ARPT_CD; Roundtrip_NonStop.DEST_ARPT_CD];
Oneway_Non_Stop.DOC_AMT = [Oneway_NonStop.DOC_AMT; Roundtrip_NonStop.DOC_AMT];
Oneway_Non_Stop.SEG_DOC_AMT = [Oneway_NonStop.SEG_DOC_AMT; Roundtrip_NonStop.SEG_DOC_AMT];
Oneway_Non_Stop.Fare_per_pax_mile = [Oneway_NonStop.Fare_per_pax_mile; Roundtrip_NonStop.Fare_per_pax_mile];
clear Oneway_NonStop Roundtrip_NonStop
%%

ORIG_ARPT_CD = [Oneway_Non_Stop.ORIG_ARPT_CD;One_Stop_Match.ORIG_ARPT_CD];

DEST_ARPT_CD = [Oneway_Non_Stop.DEST_ARPT_CD;One_Stop_Match.DEST_ARPT_CD];

Cost_per_Mile_Paid = round([Oneway_Non_Stop.Fare_per_pax_mile; Cost_per_Mile_Paid_one_stop],2);

clear fare_per_mile_non_stop One_Stop_Match ORIG_ARPT_CD_non_stop DEST_ARPT_CD_non_stop Oneway_Non_Stop

%%

TT_dep_arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);

Dep_Arr_All = strcat(ORIG_ARPT_CD,'_',DEST_ARPT_CD);

Dep_Arr_Unique = unique(Dep_Arr_All);

length_of_OD = length(Dep_Arr_Unique);

row = 1;
for OD = 1:length_of_OD


if mod(OD, 100) == 0
disp([num2str(OD), '/', num2str(length_of_OD)]);

end


[OD_index,~] = ismember(Dep_Arr_All,Dep_Arr_Unique(OD));
[TT_OD_index,~] = ismember(TT_dep_arr,Dep_Arr_Unique(OD));

if sum(OD_index) >= 50

if sum(TT_OD_index) ~= 0


OD_Fares = Cost_per_Mile_Paid(OD_index);

OD_Fares = round(OD_Fares,2);

else

OD_Fares = Cost_per_Mile_Paid(OD_index);
OD_Fares = round(OD_Fares,2);
```

```
77
78 end
79
80 %%
81
82 OD_Unique_Fares = unique(OD_Fares);
83
84 length_unique_fares = length(OD_Unique_Fares);
85
86 OD_count_fare = zeros(length_unique_fares,1);
87
88 for fare = 1:length_unique_fares
89
90 OD_count_fare(fare,1) = sum(ismember(OD_Fares,OD_Unique_Fares(fare)));
91
92 end
93
94 total_count = sum(OD_count_fare);
95
96 OD_cumm_fare = zeros(length_unique_fares,1);
97
98 for fare = 1:length_unique_fares
99
100 if fare == 1
101
102
103 OD_cumm_fare(fare,1) = 0 ;
104
105 elseif fare == 2
106
107 OD_cumm_fare(fare,1) = (OD_count_fare(fare,1) + OD_count_fare(1,1)) ;
108
109 else
110
111 OD_cumm_fare(fare,1) = (OD_count_fare(fare,1) + OD_cumm_fare((fare-1),1)) ;
112 end
113 end
114
115 OD_Fare_Per_Mile_Paid_Premium(row).OD = Dep_Arr_Unique(OD);
116 OD_Fare_Per_Mile_Paid_Premium(row).Unique_Fare = OD_Unique_Fares;
117 OD_Fare_Per_Mile_Paid_Premium(row).Cummulative_Count = OD_cumm_fare ./total_count ;
118
119 row = row + 1;
120 clear OD_count_fare OD_cumm_fare
121
122 end
123
124 end
125
126 save(([save_dir,main_delimiter,'OD_Fare_Per_Mile_Paid_Premium_Original_US.mat']),'OD_Fare_Per_Mile_Paid_Premium')
127
128 return;
129
1 function [] = Fare_per_Mile_CDF_Data_Main_Function_US_Int(SST_Pre_Processing_Dir,mach_overland,
mach_overwater,acft_seating_capacity)
2
3 %% This script generates the fare per mile CDF data from ARC 2016 - US Market
4
5 main_delimiter = '\';
6 SST_US_Int_CDF_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'CDF\US_Int_Market']);
7 save_dir = ([SST_US_Int_CDF_Dir,main_delimiter,'Output']);
8 Upload_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'VOT\US_Int_Market\Output']);
9 TravelTime_Upload_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'SST_Travel_Times\Output']);
10 %Create Output directory
11 if exist((save_dir),'dir') == 0
12
```

```
13 mkdir(([SST_US_Int_CDF_Dir,main_delimiter,'Output']))
14
15 end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0
16
17 addpath(([SST_US_Int_CDF_Dir,main_delimiter,'Output']))
18
19 Fare_Per_Mile_Paid_General_US_Int(Upload_Dir, main_delimiter, save_dir)
20
21 Fare_per_mile_paid_by_OD_Original_US_Int(Upload_Dir, TravelTime_Upload_Dir, main_delimiter, save_dir, mach_overland,
mach_overwater,
acft_seating_capacity)
22
23 return;

1
2 function [] = Fare_Per_Mile_Paid_General_US_Int(Upload_Dir, main_delimiter, save_dir)
3
4 %General Fare per Mile Paid Curve
5
6 load(([Upload_Dir,main_delimiter,'One_Stop_Match_US_Int.mat']),'One_Stop_Match')
7
8 Cost_per_Mile_Paid_one_stop = One_Stop_Match.Fare_per_Mile;
9
10 load(([Upload_Dir,main_delimiter,'Oneway_Non_Stop_US_Int_ser.mat']),'Oneway_Non_Stop_ser')
11 Oneway_NonStop = getArrayFromByteStream(Oneway_Non_Stop_ser);
12 clear Oneway_Non_Stop_ser
13 %%
14
15 load(([Upload_Dir,main_delimiter,'Roundtrip_Non_Stop_US_Int_ser.mat']),'Roundtrip_Non_Stop_ser')
16 Roundtrip_NonStop = getArrayFromByteStream(Roundtrip_Non_Stop_ser);
17 clear Oneway_Non_Stop_ser
18
19 %Combaine Rountrip nonstop and oneway nonstop records
20 Oneway_Non_Stop.TICKET_NBR = [Oneway_NonStop.TICKET_NBR; Roundtrip_NonStop.TICKET_NBR];
21 Oneway_Non_Stop.SEG_ID = [Oneway_NonStop.SEG_ID; Roundtrip_NonStop.SEG_ID];
22 Oneway_Non_Stop.Total_Mile = [Oneway_NonStop.Total_Mile; Roundtrip_NonStop.Total_Mile];
23 Oneway_Non_Stop.SEG_MILE = [Oneway_NonStop.SEG_MILE; Roundtrip_NonStop.SEG_MILE];
24 Oneway_Non_Stop.ORIG_ARPT_CD = [Oneway_NonStop.ORIG_ARPT_CD; Roundtrip_NonStop.ORIG_ARPT_CD];
25 Oneway_Non_Stop.DEST_ARPT_CD = [Oneway_NonStop.DEST_ARPT_CD; Roundtrip_NonStop.DEST_ARPT_CD];
26 Oneway_Non_Stop.DOC_AMT = [Oneway_NonStop.DOC_AMT; Roundtrip_NonStop.DOC_AMT];
27 Oneway_Non_Stop.SEG_DOC_AMT = [Oneway_NonStop.SEG_DOC_AMT; Roundtrip_NonStop.SEG_DOC_AMT];
28 Oneway_Non_Stop.Fare_per_pax_mile = [Oneway_NonStop.Fare_per_pax_mile; Roundtrip_NonStop.Fare_per_pax_mile];
29 clear Oneway_NonStop Roundtrip_NonStop
30 %%
31 Cost_per_Mile_Paid_non_stop = Oneway_Non_Stop.Fare_per_pax_mile;
32
33 Cost_per_Mile_Paid_Premium = round([Cost_per_Mile_Paid_one_stop;Cost_per_Mile_Paid_non_stop],2);
34
35 total_records = length(Cost_per_Mile_Paid_Premium);
36 unique_fare_per_mile_paid = unique(Cost_per_Mile_Paid_Premium);
37
38 length_of_unique_fare_per_mile_paid = length(unique_fare_per_mile_paid);
39
40 Fare_per_Mile_Paid_Percentage_Premium = zeros(length_of_unique_fare_per_mile_paid,1);
41
42 for record = 1:length_of_unique_fare_per_mile_paid
43
44 Fare_per_Mile_Paid_Percentage_Premium(record) = length(find(Cost_per_Mile_Paid_Premium == unique_fare_per_mile_paid(record))) ./
total_records ;
45
46 end
47 %%
48
49 for record = 1:length_of_unique_fare_per_mile_paid
50
51 Fare_per_Mile_Paid_Premium.Fare(record,1) = unique_fare_per_mile_paid(record);
```

```
52
53 Fare_per_Mile_Paid_Premium.Percentage(record,1) = length(find(Cost_per_Mile_Paid_Premium == unique_fare_per_mile_paid(record))) ./
total_records ;
54
55 if record == 1
56
57 Fare_per_Mile_Paid_Premium.Cumulative(record,1) = 0;
58
59 elseif record == 2
60
61 Fare_per_Mile_Paid_Premium.Cumulative(record,1) = Fare_per_Mile_Paid_Premium.Percentage(record) +
Fare_per_Mile_Paid_Premium.Percentage
(1);
62
63 else
64
65 Fare_per_Mile_Paid_Premium.Cumulative(record,1) = Fare_per_Mile_Paid_Premium.Percentage(record) +
Fare_per_Mile_Paid_Premium.Cumulative
(record - 1);
66
67 end
68
69 end
70
71 save(([save_dir,main_delimiter,'Fare_per_Mile_Paid_Premium_US_Int.mat']),'Fare_per_Mile_Paid_Premium')
72
73 return;
74
```

```
1
2 function [] = Fare_per_mile_paid_by_OD_Original_US_Int(Upload_Dir, TravelTime_Upload_Dir, main_delimiter, save_dir, mach_overland,
mach_overwater,acft_seating_capacity)
3
4
5 load(([TravelTime_Upload_Dir,main_delimiter,'Travel_Times_with_Ground_Times_1_',num2str((mach_overland-
1)*10),'M_1_',num2str((mach_overwater-
1)*10),'M_',num2str(acft_seating_capacity),'_Seats.mat']),'Travel_Times')
6
7 load(([Upload_Dir,main_delimiter,'One_Stop_Match_US_Int.mat']),'One_Stop_Match')
8
9 Cost_per_Mile_Paid_one_stop = One_Stop_Match.Fare_per_Mile;
10
11 load(([Upload_Dir,main_delimiter,'Oneway_Non_Stop_US_Int_ser.mat']),'Oneway_Non_Stop_ser')
12 Oneway_NonStop = getArrayFromByteStream(Oneway_Non_Stop_ser);
13 clear Oneway_Non_Stop_ser
14 %%
15
16 load(([Upload_Dir,main_delimiter,'Roundtrip_Non_Stop_US_Int_ser.mat']),'Roundtrip_Non_Stop_ser')
17 Roundtrip_NonStop = getArrayFromByteStream(Roundtrip_Non_Stop_ser);
18 clear Oneway_Non_Stop_ser
19
20 %Combaine Rountrip nonstop and oneway nonstop records
21 Oneway_Non_Stop.TICKET_NBR = [Oneway_NonStop.TICKET_NBR; Roundtrip_NonStop.TICKET_NBR];
22 Oneway_Non_Stop.SEG_ID = [Oneway_NonStop.SEG_ID; Roundtrip_NonStop.SEG_ID];
23 Oneway_Non_Stop.Total_Mile = [Oneway_NonStop.Total_Mile; Roundtrip_NonStop.Total_Mile];
24 Oneway_Non_Stop.SEG_MILE = [Oneway_NonStop.SEG_MILE; Roundtrip_NonStop.SEG_MILE];
25 Oneway_Non_Stop.ORIG_ARPT_CD = [Oneway_NonStop.ORIG_ARPT_CD; Roundtrip_NonStop.ORIG_ARPT_CD];
26 Oneway_Non_Stop.DEST_ARPT_CD = [Oneway_NonStop.DEST_ARPT_CD; Roundtrip_NonStop.DEST_ARPT_CD];
27 Oneway_Non_Stop.DOC_AMT = [Oneway_NonStop.DOC_AMT; Roundtrip_NonStop.DOC_AMT];
28 Oneway_Non_Stop.SEG_DOC_AMT = [Oneway_NonStop.SEG_DOC_AMT; Roundtrip_NonStop.SEG_DOC_AMT];
29 Oneway_Non_Stop.Fare_per_pax_mile = [Oneway_NonStop.Fare_per_pax_mile; Roundtrip_NonStop.Fare_per_pax_mile];
30 clear Oneway_NonStop Roundtrip_NonStop
31 %%
32
33 ORIG_ARPT_CD = [Oneway_Non_Stop.ORIG_ARPT_CD;One_Stop_Match.ORIG_ARPT_CD];
34
```

```
35 DEST_ARPT_CD = [Oneway_Non_Stop.DEST_ARPT_CD;One_Stop_Match.DEST_ARPT_CD];
36
37 Cost_per_Mile_Paid = round([Oneway_Non_Stop.Fare_per_pax_mile; Cost_per_Mile_Paid_one_stop],2);
38
39 clear fare_per_mile_non_stop One_Stop_Match ORIG_ARPT_CD_non_stop DEST_ARPT_CD_non_stop Oneway_Non_Stop
40
41 %%
42
43 TT_dep_arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);
44
45 Dep_Arr_All = strcat(ORIG_ARPT_CD,'_',DEST_ARPT_CD);
46
47 Dep_Arr_Unique = unique(Dep_Arr_All);
48
49 length_of_OD = length(Dep_Arr_Unique);
50
51 row = 1;
52
53 for OD = 1:length_of_OD
54
55 if mod(OD, 100) == 0
56 disp([num2str(OD), '/', num2str(length_of_OD)]);
57
58 end
59
60 [OD_index,~] = ismember(Dep_Arr_All,Dep_Arr_Unique(OD));
61 [TT_OD_index,~] = ismember(TT_dep_arr,Dep_Arr_Unique(OD));
62
63 if sum(OD_index) >= 50
64
65 if sum(TT_OD_index) ~= 0
66
67
68 OD_Fares = Cost_per_Mile_Paid(OD_index);
69
70 OD_Fares = round(OD_Fares,2);
71
72 else
73
74 OD_Fares = Cost_per_Mile_Paid(OD_index);
75 OD_Fares = round(OD_Fares,2);
76
77 end
78
79 %%
80
81 OD_Unique_Fares = unique(OD_Fares);
82
83 length_unique_fares = length(OD_Unique_Fares);
84
85 OD_count_fare = zeros(length_unique_fares,1);
86
87 for fare = 1:length_unique_fares
88
89 OD_count_fare(fare,1) = sum(ismember(OD_Fares,OD_Unique_Fares(fare)));
90
91 end
92
93 total_count = sum(OD_count_fare);
94
95 OD_cumm_fare = zeros(length_unique_fares,1);
96
97 for fare = 1:length_unique_fares
98
99 if fare == 1
100
```

```
101
102 OD_cumm_fare(fare,1) = 0 ;
103
104 elseif fare == 2
105
106 OD_cumm_fare(fare,1) = (OD_count_fare(fare,1) + OD_count_fare(1,1)) ;
107
108 else
109
110 OD_cumm_fare(fare,1) = (OD_count_fare(fare,1) + OD_cumm_fare((fare-1),1)) ;
111 end
112 end
113
114
115 OD_Fare_Per_Mile_Paid_Premium(row).OD = Dep_Arr_Unique(OD);
116 OD_Fare_Per_Mile_Paid_Premium(row).Unique_Fare = OD_Unique_Fares;
117 OD_Fare_Per_Mile_Paid_Premium(row).Cummulative_Count = OD_cumm_fare ./total_count ;
118
119 row = row + 1;
120 clear OD_count_fare OD_cumm_fare
121
122 end
123
124
125 end
126
127 save(([save_dir,main_delimiter,'OD_Fare_Per_Mile_Paid_Premium_Original_US_Int.mat']),'OD_Fare_Per_Mile_Paid_Premium')
128
129 return;
130
1 function [] = Fare_per_Mile_CDF_Data_Main_Function_Int(SST_Pre_Processing_Dir,mach_overland, mach_overwater,acft_seating_capacity)
2 %% This script generates the fare per mile CDF data from ARC 2016 - Int Market
3
4 main_delimiter = '\';
5 SST_Int_CDF_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'CDF\Int_Market']);
6 save_dir = ([SST_Int_CDF_Dir,main_delimiter,'Output']);
7 Upload_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'VOT\Int_Market\Output']);
8 TravelTime_Upload_Dir = ([SST_Pre_Processing_Dir,main_delimiter,'SST_Travel_Times\Output']);
9 %Create Output directory
10 if exist((save_dir),'dir') == 0
11
12 mkdir(([SST_Int_CDF_Dir,main_delimiter,'Output']))
13
14 end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0
15
16 addpath(([SST_Int_CDF_Dir,main_delimiter,'Output']))
17
18 Fare_Per_Mile_Paid_General_Int(Upload_Dir, main_delimiter, save_dir)
19
20 Fare_per_mile_paid_by_OD_Original_Int(Upload_Dir, TravelTime_Upload_Dir, main_delimiter, save_dir, mach_overland, mach_overwater,
acft_seating_capacity)
21
22 return;
1
2 function [] = Fare_Per_Mile_Paid_General_Int(Upload_Dir, main_delimiter, save_dir)
3
4 %General Fare per Mile Paid Curve
5
6 load(([Upload_Dir,main_delimiter,'One_Stop_Match_Int.mat']),'One_Stop_Match')
7
8 Cost_per_Mile_Paid_one_stop = One_Stop_Match.Fare_per_Mile;
9
10 load(([Upload_Dir,main_delimiter,'Oneway_Non_Stop_Int_ser.mat']),'Oneway_Non_Stop_ser')
11 Oneway_NonStop = getArrayFromByteStream(Oneway_Non_Stop_ser);
12 clear Oneway_Non_Stop_ser
13 %%
```

```
14
15 load(([Upload_Dir,main_delimiter,'Roundtrip_Non_Stop_Int_ser.mat']),'Roundtrip_Non_Stop_ser')
16 Roundtrip_NonStop = getArrayFromByteStream(Roundtrip_Non_Stop_ser);
17 clear Oneway_Non_Stop_ser
18
19 %Combaine Rountrip nonstop and oneway nonstop records
20 Oneway_Non_Stop.TICKET_NBR = [Oneway_NonStop.TICKET_NBR; Roundtrip_NonStop.TICKET_NBR];
21 Oneway_Non_Stop.SEG_ID = [Oneway_NonStop.SEG_ID; Roundtrip_NonStop.SEG_ID];
22 Oneway_Non_Stop.Total_Mile = [Oneway_NonStop.Total_Mile; Roundtrip_NonStop.Total_Mile];
23 Oneway_Non_Stop.SEG_MILE = [Oneway_NonStop.SEG_MILE; Roundtrip_NonStop.SEG_MILE];
24 Oneway_Non_Stop.ORIG_ARPT_CD = [Oneway_NonStop.ORIG_ARPT_CD; Roundtrip_NonStop.ORIG_ARPT_CD];
25 Oneway_Non_Stop.DEST_ARPT_CD = [Oneway_NonStop.DEST_ARPT_CD; Roundtrip_NonStop.DEST_ARPT_CD];
26 Oneway_Non_Stop.DOC_AMT = [Oneway_NonStop.DOC_AMT; Roundtrip_NonStop.DOC_AMT];
27 Oneway_Non_Stop.SEG_DOC_AMT = [Oneway_NonStop.SEG_DOC_AMT; Roundtrip_NonStop.SEG_DOC_AMT];
28 Oneway_Non_Stop.Fare_per_pax_mile = [Oneway_NonStop.Fare_per_pax_mile; Roundtrip_NonStop.Fare_per_pax_mile];
29 clear Oneway_NonStop Roundtrip_NonStop
30 %%
31 Cost_per_Mile_Paid_non_stop = Oneway_Non_Stop.Fare_per_pax_mile;
32
33 Cost_per_Mile_Paid_Premium = round([Cost_per_Mile_Paid_one_stop;Cost_per_Mile_Paid_non_stop],2);
34
35 total_records = length(Cost_per_Mile_Paid_Premium);
36 unique_fare_per_mile_paid = unique(Cost_per_Mile_Paid_Premium);
37
38 length_of_unique_fare_per_mile_paid = length(unique_fare_per_mile_paid);
39
40 Fare_per_Mile_Paid_Percentage_Premium = zeros(length_of_unique_fare_per_mile_paid,1);
41
42 for record = 1:length_of_unique_fare_per_mile_paid
43
44 Fare_per_Mile_Paid_Percentage_Premium(record) = length(find(Cost_per_Mile_Paid_Premium == unique_fare_per_mile_paid(record))) ./
total_records ;
45
46 end
47 %%
48
49 for record = 1:length_of_unique_fare_per_mile_paid
50
51 Fare_per_Mile_Paid_Premium.Fare(record,1) = unique_fare_per_mile_paid(record);
52
53 Fare_per_Mile_Paid_Premium.Percentage(record,1) = length(find(Cost_per_Mile_Paid_Premium == unique_fare_per_mile_paid(record))) ./
total_records ;
54
55 if record == 1
56
57 Fare_per_Mile_Paid_Premium.Cumulative(record,1) = 0;
58
59 elseif record == 2
60
61 Fare_per_Mile_Paid_Premium.Cumulative(record,1) = Fare_per_Mile_Paid_Premium.Percentage(record) +
Fare_per_Mile_Paid_Premium.Percentage
(1);
62
63 else
64
65 Fare_per_Mile_Paid_Premium.Cumulative(record,1) = Fare_per_Mile_Paid_Premium.Percentage(record) +
Fare_per_Mile_Paid_Premium.Cumulative
(record - 1);
66
67 end
68
69 end
70
71 save(([save_dir,main_delimiter,'Fare_per_Mile_Paid_Premium_Int.mat']),'Fare_per_Mile_Paid_Premium')
72
73 return;
```

```
74
1
2 function [] = Fare_per_mile_paid_by_OD_Original_Int(Upload_Dir, TravelTime_Upload_Dir, main_delimiter, save_dir, mach_overland,
mach_overwater,
acft_seating_capacity)
3
4
5 load(([TravelTime_Upload_Dir,main_delimiter,'Travel_Times_with_Ground_Times_1_',num2str((mach_overland-
1)*10),'M_1_',num2str((mach_overwater-
1)*10),'M_',num2str(acft_seating_capacity),'_Seats.mat']),'Travel_Times')
6
7 load(([Upload_Dir,main_delimiter,'One_Stop_Match_Int.mat']),'One_Stop_Match')
8
9 Cost_per_Mile_Paid_one_stop = One_Stop_Match.Fare_per_Mile;
10
11 load(([Upload_Dir,main_delimiter,'Oneway_Non_Stop_Int_ser.mat']),'Oneway_Non_Stop_ser')
12 Oneway_NonStop = getArrayFromByteStream(Oneway_Non_Stop_ser);
13 clear Oneway_Non_Stop_ser
14 %%
15
16 load(([Upload_Dir,main_delimiter,'Roundtrip_Non_Stop_Int_ser.mat']),'Roundtrip_Non_Stop_ser')
17 Roundtrip_NonStop = getArrayFromByteStream(Roundtrip_Non_Stop_ser);
18 clear Oneway_Non_Stop_ser
19
20 %Combaine Rountrip nonstop and oneway nonstop records
21 Oneway_Non_Stop.TICKET_NBR = [Oneway_NonStop.TICKET_NBR; Roundtrip_NonStop.TICKET_NBR];
22 Oneway_Non_Stop.SEG_ID = [Oneway_NonStop.SEG_ID; Roundtrip_NonStop.SEG_ID];
23 Oneway_Non_Stop.Total_Mile = [Oneway_NonStop.Total_Mile; Roundtrip_NonStop.Total_Mile];
24 Oneway_Non_Stop.SEG_MILE = [Oneway_NonStop.SEG_MILE; Roundtrip_NonStop.SEG_MILE];
25 Oneway_Non_Stop.ORIG_ARPT_CD = [Oneway_NonStop.ORIG_ARPT_CD; Roundtrip_NonStop.ORIG_ARPT_CD];
26 Oneway_Non_Stop.DEST_ARPT_CD = [Oneway_NonStop.DEST_ARPT_CD; Roundtrip_NonStop.DEST_ARPT_CD];
27 Oneway_Non_Stop.DOC_AMT = [Oneway_NonStop.DOC_AMT; Roundtrip_NonStop.DOC_AMT];
28 Oneway_Non_Stop.SEG_DOC_AMT = [Oneway_NonStop.SEG_DOC_AMT; Roundtrip_NonStop.SEG_DOC_AMT];
29 Oneway_Non_Stop.Fare_per_pax_mile = [Oneway_NonStop.Fare_per_pax_mile; Roundtrip_NonStop.Fare_per_pax_mile];
30 clear Oneway_NonStop Roundtrip_NonStop
31 %%
32
33 ORIG_ARPT_CD = [Oneway_Non_Stop.ORIG_ARPT_CD;One_Stop_Match.ORIG_ARPT_CD];
34
35 DEST_ARPT_CD = [Oneway_Non_Stop.DEST_ARPT_CD;One_Stop_Match.DEST_ARPT_CD];
36
37 Cost_per_Mile_Paid = round([Oneway_Non_Stop.Fare_per_pax_mile; Cost_per_Mile_Paid_one_stop],2);
38
39 clear fare_per_mile_non_stop One_Stop_Match ORIG_ARPT_CD_non_stop DEST_ARPT_CD_non_stop Oneway_Non_Stop
40
41 %%
42
43 TT_dep_arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);
44
45 Dep_Arr_All = strcat(ORIG_ARPT_CD,'_',DEST_ARPT_CD);
46
47 Dep_Arr_Unique = unique(Dep_Arr_All);
48
49 length_of_OD = length(Dep_Arr_Unique);
50
51 row = 1;
52
53 for OD = 1:length_of_OD
54
55 if mod(OD, 100) == 0
56 disp([num2str(OD), '/', num2str(length_of_OD)]);
57
58 end
59
60 [OD_index,~] = ismember(Dep_Arr_All,Dep_Arr_Unique(OD));
61 [TT_OD_index,~] = ismember(TT_dep_arr,Dep_Arr_Unique(OD));
```

```
62
63 if sum(OD_index) >= 50
64
65 if sum(TT_OD_index) ~= 0
66
67 OD_Fares = Cost_per_Mile_Paid(OD_index);
68
69 OD_Fares = round(OD_Fares,2);
70
71 else
72
73 OD_Fares = Cost_per_Mile_Paid(OD_index);
74 OD_Fares = round(OD_Fares,2);
75
76 end
77
78 %%
79 OD_Unique_Fares = unique(OD_Fares);
80
81 length_unique_fares = length(OD_Unique_Fares);
82
83 OD_count_fare = zeros(length_unique_fares,1);
84
85 for fare = 1:length_unique_fares
86
87 OD_count_fare(fare,1) = sum(ismember(OD_Fares,OD_Unique_Fares(fare)));
88
89 end
90
91 total_count = sum(OD_count_fare);
92
93 OD_cumm_fare = zeros(length_unique_fares,1);
94
95 for fare = 1:length_unique_fares
96
97 if fare == 1
98
99 OD_cumm_fare(fare,1) = 0 ;
100
101 elseif fare == 2
102
103 OD_cumm_fare(fare,1) = (OD_count_fare(fare,1) + OD_count_fare(1,1)) ;
104
105 else
106
107 OD_cumm_fare(fare,1) = (OD_count_fare(fare,1) + OD_cumm_fare((fare-1),1)) ;
108 end
109 end
110
111 OD_Fare_Per_Mile_Paid_Premium(row).OD = Dep_Arr_Unique(OD);
112 OD_Fare_Per_Mile_Paid_Premium(row).Unique_Fare = OD_Unique_Fares;
113 OD_Fare_Per_Mile_Paid_Premium(row).Cummulative_Count = OD_cumm_fare ./total_count ;
114
115 row = row + 1;
116 clear OD_count_fare OD_cumm_fare
117
118 end
119
120 end
121
122 save([save_dir,main_delimiter,'OD_Fare_Per_Mile_Paid_Premium_Original_Int.mat']),'OD_Fare_Per_Mile_Paid_Premium')
123
124 return;
125
```

SST_Market_Analysis_Main_Script

```
%% Run the SST projections for all three markets individually

clear
clc

% Run point design block fuel calculator using flops Excel file
% YES = 1
% NO = 0
PointDesign = 0;
Flops_Output_File_Name = 'PAX46M1.7R3163.xlsx';
Sheet_name = 'PAX46M1.7R3163';

local_disc = '';
main_delimiter = '\';
main_file_name = '.';
Input_Folder_Dir = ([local_disc,'..\..\SST_2020_Input']);
Load_Pre_Process_Dir = ([local_disc,main_file_name,main_delimiter,'SST_Pre_Processing']);
SST_Market_Analysis_Dir = ([local_disc,main_file_name,main_delimiter,'SST_Market_Analysis']);

load([Input_Folder_Dir,main_delimiter,'Airport_List_SST_Compatibility.mat'],'Airport_List_SST_Compatibility')
load([Input_Folder_Dir,main_delimiter,'Airport_Runway_Compatibility.mat'],'airport_runway_length_data')

load(([Input_Folder_Dir,main_delimiter,'airport_list.mat']),'airport_list')

addpath(genpath('.\'));
addpath(genpath(Input_Folder_Dir));
%% Read Parameters from Excel file
opts = spreadsheetImportOptions("NumVariables", 9);

% Specify sheet and range
opts.Sheet = "Sheet1";
opts.DataRange = "A3:I26";

% Specify column names and types
opts.VariableNames = ["VarName1", "Aircraft_Seating_Capacity", "OverlandRangenm", "OverlandMach", "OverlandFuleScaleFactor",
"OvewaterRangenm", "OverwaterMach", "OverwaterFuleScaleFactor", "AcftUtilization"];
opts.VariableTypes = ["string", "double", "double", "double", "double", "double", "double", "double", "double"];

% Specify variable properties
opts = setvaropts(opts, "VarName1", "WhitespaceRule", "preserve");
opts = setvaropts(opts, "VarName1", "EmptyFieldRule", "auto");

% Import the data
tbl = readtable("..\..\SST_2020_Input\SST_2020_Parameters.xlsx", opts, "UseExcel", false);

%% Convert to output type
Case_Parameters.CaseNumber = tbl.VarName1;
Case_Parameters.Aircraft_Seating_Capacity = tbl.Aircraft_Seating_Capacity;
Case_Parameters.OverlandRangenm = tbl.OverlandRangenm;
Case_Parameters.OverlandMach = tbl.OverlandMach;
Case_Parameters.OverlandFuleScaleFactor = tbl.OverlandFuleScaleFactor;
Case_Parameters.OvewaterRangenm = tbl.OvewaterRangenm;
Case_Parameters.OverwaterMach = tbl.OverwaterMach;
Case_Parameters.OverwaterFuleScaleFactor = tbl.OverwaterFuleScaleFactor;
Case_Parameters.Acft_Utilization_hrs = tbl.AcftUtilization;
%% Clear temporary variables
clear opts tbl
%%
day = '11';
month = '05';
yr = '2021';

if PointDesign == 1

nummber_of_cases = 1;
```

```
65
66 [flops_data,mach_overland, mach_overwater, number_of_seats, Fuel_Scaling_Parameter_OL,
Fuel_Scaling_Parameter_OW,hours_per_year,
aircraft_range_overland,aircraft_range_overwater] = Set_parameters_from_flops(Input_Folder_Dir,Flops_Output_File_Name,Sheet_name);
67
68 else
69
70 nummber_of_cases = length(Case_Parameters.CaseNumber);
71 flops_data = [];
72 end
73
74
75 for run_case = 7%1:nummber_of_cases
76
77 disp(['Analyzing Case: ',num2str(run_case),'/',num2str(nummber_of_cases)])
78 %% Parameters
79 run_of_day = char(Case_Parameters.CaseNumber(run_case));
80 % Value of time. $/hr.
81 US_VOT = 0;
82 US_Int_VOT = 0;
83 Int_VOT = 0;
84
85
86 if PointDesign == 0
87 %when adjusting the fare per mile paid CDF data for value of time and
88 %travel time savings. Indicates the match combination.
89 mach_overland = Case_Parameters.OverlandMach(run_case);
90 mach_overwater = Case_Parameters.OverwaterMach(run_case);
91
92 %aircraft seating capacity
93 number_of_seats = Case_Parameters.Aircraft_Seating_Capacity(run_case);
94
95 %FuelScalingFactor Percent - FSF = 100 means no adjustment
96 Fuel_Scaling_Parameter_OL = Case_Parameters.OverlandFuleScaleFactor(run_case); %To be use in the LCC module
97 Fuel_Scaling_Parameter_OW = Case_Parameters.OverwaterFuleScaleFactor(run_case); %To be use in the LCC module
98
99
100 %Operational hours per year per aircraft
101 hours_per_year = Case_Parameters.Acft_Utilization_hrs(run_case);
102
103 %Aircraft Range (nm.)
104 aircraft_range_overland = Case_Parameters.OverlandRangenm(run_case); % 3,000nm. = 3,542sm %2,800nm = 3,222sm.
105 aircraft_range_overwater = Case_Parameters.OvewaterRangenm(run_case);%3,800nm = 4373sm.
106
107 end
108
109 %Pax load factor
110 acft_pax_load_factor = 0.85;
111
112 %Start and End year for the forecast
113 SST_Year_Start = 2030;
114 SST_Year_End = 2040;
115
116 %Maximum VOT permitted in the analysis
117 vot_limit = 400; %$400/hr.
118
119
120 %Minimum OD distance
121 min_distance_statute_mile = 1000; %sm.
122
123 %Maximum OD distance
124 max_distance_statute_mile = 9500; %sm.
125
126 %Minimum total demand to consider an OD pair as a potential SST OD candidate
127 min_demand = 100000; %100,000 total seats (including economy and premium)
128
```

```
%Market share indicates the percent of travelers that at the end of the day
%could be willing to trade comfort for travel time savings
market_share = 0.50; %50percent

run_airport_gate_compatibility_section = 1; %Yes=1, No=0
run_airport_runway_compatibility_section = 1; %Yes=1, No=0
runway_length_min_ft = 8700;

% Number of aircraft to to be produced in the LCC
number_of_aircraft = 600;

%FuelScaling Factor (additional)
FSF = 100; %to be used only in the demand module. If a FSF was used in the LCC, don't use this one.

%Number of aircraft needed to satisfy the demand. Set to zero for iteration
%1 until the demand is calculated.
number_of_aircraft_needed = 0;

date = (['_',num2str(number_of_seats),'_Seats_',strcat([day,month,yr]),run_of_day]);

Iteration = 1;

while number_of_aircraft ~= number_of_aircraft_needed

if Iteration ~= 1

%After the first iteration, the number of aircraft needed to
%satisfy the demand becomes the number of aircraft to be used as
%production in the LCC until a steady stead is reach

number_of_aircraft = number_of_aircraft_needed;

end
%% LCC
disp('Running LCC Module')

Integrated_LowBoom_SurrogateModel(Input_Folder_Dir,aircraft_range_overland,aircraft_range_overwater,number_of_seats,mach_overland,
mach_overwater,hours_per_year,number_of_aircraft,Fuel_Scaling_Parameter_OL, Fuel_Scaling_Parameter_OW,date,PointDesign,flops_data)


%% US Market
disp('Running Demand Module')
disp('US Market')

Fare_per_Mile_CDF_Adjustment_Main_Function_US(SST_Market_Analysis_Dir,Load_Pre_Process_Dir,mach_overland,mach_overwater,vot_li
mit,
US_VOT,date,aircraft_range_overland,aircraft_range_overwater)

SST_US_Forecast_Main_Function(SST_Market_Analysis_Dir,Load_Pre_Process_Dir,Airport_List_SST_Compatibility,airport_list,mach_overland,
mach_overwater,number_of_seats,FSF,hours_per_year,acft_pax_load_factor,SST_Year_Start,SST_Year_End,vot_limit,aircraft_range_overland,
aircraft_range_overwater,min_distance_statute_mile,max_distance_statute_mile,min_demand,market_share,US_VOT,date,
run_airport_gate_compatibility_section, run_airport_runway_compatibility_section,runway_length_min_ft,airport_runway_length_data)

%% US_Int. Market
disp('US-International Market')

Fare_per_Mile_CDF_Adjustment_Main_Function_US_Int(SST_Market_Analysis_Dir,Load_Pre_Process_Dir,mach_overland,mach_overwater,vot
_limit,
US_Int_VOT,date,aircraft_range_overland,aircraft_range_overwater)

SST_US_Int_Forecast_Main_Function(SST_Market_Analysis_Dir,Load_Pre_Process_Dir,Airport_List_SST_Compatibility,airport_list,mach_overla
nd,
mach_overwater,number_of_seats,FSF,hours_per_year,acft_pax_load_factor,SST_Year_Start,SST_Year_End,vot_limit,aircraft_range_overland,
aircraft_range_overwater,min_distance_statute_mile,max_distance_statute_mile,min_demand,market_share,US_Int_VOT,date,
run_airport_gate_compatibility_section, run_airport_runway_compatibility_section,runway_length_min_ft,airport_runway_length_data)

```

```
%% Int. Market
disp('International Market')

Fare_per_Mile_CDF_Adjustment_Main_Function_Int(SST_Market_Analysis_Dir,Load_Pre_Process_Dir,mach_overland,mach_overwater,vot_limit,
Int_VOT,date,aircraft_range_overland,aircraft_range_overwater)

SST_Int_Forecast_Main_Function(SST_Market_Analysis_Dir,Load_Pre_Process_Dir,Airport_List_SST_Compatibility,airport_list,mach_overland,
mach_overwater,number_of_seats,FSF,hours_per_year,acft_pax_load_factor,SST_Year_Start,SST_Year_End,vot_limit,aircraft_range_overland,
aircraft_range_overwater,min_distance_statute_mile,max_distance_statute_mile,min_demand,market_share,Int_VOT,date,
run_airport_gate_compatibility_section, run_airport_runway_compatibility_section,runway_length_min_ft,airport_runway_length_data)

%% Merge Results

[number_of_aircraft_needed] =
Final_Results_Merge(SST_Market_Analysis_Dir,aircraft_range_overland,aircraft_range_overwater,mach_overland,
mach_overwater,date);

disp((['Iteration: ',num2str(Iteration)]))
if number_of_aircraft_needed ~= number_of_aircraft

disp((['Number of Aircraft Needed: ',num2str(number_of_aircraft_needed)]))
disp((['Number of Aircraft Produced: ',num2str(number_of_aircraft)]))
disp('Continue to Next Iteration')

else

disp((['Number of Aircraft Needed: ',num2str(number_of_aircraft_needed)]))
disp((['Number of Aircraft Produced: ',num2str(number_of_aircraft)]))
disp('Analysis Completed')



end

Iteration = Iteration + 1;
clearvars -except local_disc main_delimiter main_file_name...
Input_Folder_Dir Load_Pre_Process_Dir SST_Market_Analysis_Dir...
Airport_List_SST_Compatibility airport_runway_length_data day...
month yr run_of_day US_VOT US_Int_VOT Int_VOT mach_overland...
mach_overwater number_of_seats Fuel_Scaling_Parameter_OL FSF...
hours_per_year acft_pax_load_factor SST_Year_Start SST_Year_End...
vot_limit aircraft_range_overland aircraft_range_overwater...
min_distance_statute_mile max_distance_statute_mile min_demand...
market_share run_airport_gate_compatibility_section...
run_airport_runway_compatibility_section runway_length_min_ft...
number_of_aircraft number_of_aircraft_needed date Iteration airport_list...
Case_Parameters run_case nummber_of_cases Fuel_Scaling_Parameter_OW...
PointDesign Flops_Output_File_Name flops_data Sheet_name

end

%% Export final output to Excel files

if exist((([SST_Market_Analysis_Dir,main_delimiter,'Excel_Format_Output'])),'dir') == 0

mkdir(([SST_Market_Analysis_Dir,main_delimiter,'Excel_Format_Output']))
addpath(([SST_Market_Analysis_Dir,main_delimiter,'Excel_Format_Output']))

end

filename =
([SST_Market_Analysis_Dir,main_delimiter,'Excel_Format_Output',main_delimiter,'SST_Final_Output_OverlandRange_',num2str
(aircraft_range_overland),'nm_OverwaterRange_',num2str(aircraft_range_overwater),'nm_Mach_1_',num2str((mach_overland-1)
*10),'_Overland_&_Mach_1_',num2str((mach_overwater-1)*10),'_Overwater',date,'.xlsx']);

```

```
232 % % % %% Excel file for year 2040 Only
233 % % % %% US_Market
234 % % %
235 % % % load(([SST_Market_Analysis_Dir,'\US_Market\SST_Forecast\Output\SST_Final_Market_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_US',date,'.mat']),'SST_Final_Market')
236 % % % SST_Final_Market_US = SST_Final_Market;
237 % % % clear SST_Final_Market
238 % % %
239 % % % col_header1_us = {'US Market'};
240 % % % col_header2_us =
{'DepAirport_UniqueID','DepAirport','DepIATACtry','DepReg','ArrAirport_UniqueID','ArrAirport','ArrIATACtry','ArrReg','Total_Seats','Premium_S
eats','OD_Av
erageElapsedTime_hrs','DistStMiles','SST_Premium_Seats','SST_Total_Seats','Fare_per_mile_cost','generic_cdf','Aircraft_Needed','Number_of_
Passenger',};
241 % % % us_data_table =
table(([SST_Final_Market_US.(['Year_',num2str(SST_Year_End)]).DepAirport_UniqueID]),([SST_Final_Market_US.(['Year_',
num2str(SST_Year_End)]).DepAirport]),([SST_Final_Market_US.(['Year_',num2str(SST_Year_End)]).DepIATACtry]),([SST_Final_Market_US.(['Yea
r_',num2str
(SST_Year_End)]).DepReg]),([SST_Final_Market_US.(['Year_',num2str(SST_Year_End)]).ArrAirport_UniqueID]),([SST_Final_Market_US.(['Year_',n
um2str
(SST_Year_End)]).ArrAirport]),([SST_Final_Market_US.(['Year_',num2str(SST_Year_End)]).ArrIATACtry]),([SST_Final_Market_US.(['Year_',num2st
r
(SST_Year_End)]).ArrReg]),([SST_Final_Market_US.(['Year_',num2str(SST_Year_End)]).Total_Seats]),([SST_Final_Market_US.(['Year_',num2str(SS
T_Year_End)]).
Premium_Seats]),([SST_Final_Market_US.(['Year_',num2str(SST_Year_End)]).OD_AverageElapsedTime_hrs]),([SST_Final_Market_US.(['Year_',n
um2str
(SST_Year_End)]).DistStMiles]),([SST_Final_Market_US.(['Year_',num2str(SST_Year_End)]).SST_Premium_Seats]),([SST_Final_Market_US.(['Year
_',num2str
(SST_Year_End)]).SST_Total_Seats]),([SST_Final_Market_US.(['Year_',num2str(SST_Year_End)]).Fare_per_mile_cost]),([SST_Final_Market_US.(['
Year_',num2str
(SST_Year_End)]).generic_cdf]),([SST_Final_Market_US.(['Year_',num2str(SST_Year_End)]).Aircraft_Needed]),([SST_Final_Market_US.(['Year_',n
um2str
(SST_Year_End)]).Number_of_Passenger]));
242 % % %
243 % % % xlswrite(filename,col_header1_us,'Sheet1','A1');
244 % % % xlswrite(filename,col_header2_us,'Sheet1','A2');
245 % % % writetable(us_data_table,filename,'Sheet',1,'Range','A3','WriteVariableNames',false)
246 % % %
247 % % % %% US_Int_Market
248 % % %
249 % % % load(([SST_Market_Analysis_Dir,'\US_Int_Market\SST_Forecast\Output\SST_Final_Market_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_US_Int',date,'.mat']),'SST_Final_Market')
250 % % % SST_Final_Market_US_Int = SST_Final_Market;
251 % % % clear SST_Final_Market
252 % % % col_header1_us_int = {'US_Int Market'};
253 % % % col_header2_us_int =
{'DepAirport_UniqueID','DepAirport','DepIATACtry','DepReg','ArrAirport_UniqueID','ArrAirport','ArrIATACtry','ArrReg','Total_Seats','Premium_S
eats','OD_Av
erageElapsedTime_hrs','DistStMiles','SST_Premium_Seats','SST_Total_Seats','Fare_per_mile_cost','generic_cdf','Aircraft_Needed','Number_of_
Passenger',};
254 % % % us_int_data_table =
table(([SST_Final_Market_US_Int.(['Year_',num2str(SST_Year_End)]).DepAirport_UniqueID]),([SST_Final_Market_US_Int.
(['Year_',num2str(SST_Year_End)]).DepAirport]),([SST_Final_Market_US_Int.(['Year_',num2str(SST_Year_End)]).DepIATACtry]),([SST_Final_Mark
et_US_Int.
(['Year_',num2str(SST_Year_End)]).DepReg]),([SST_Final_Market_US_Int.(['Year_',num2str(SST_Year_End)]).ArrAirport_UniqueID]),([SST_Final_
Market_US_Int.
(['Year_',num2str(SST_Year_End)]).ArrAirport]),([SST_Final_Market_US_Int.(['Year_',num2str(SST_Year_End)]).ArrIATACtry]),([SST_Final_Marke
t_US_Int.(['Year_',
num2str(SST_Year_End)]).ArrReg]),([SST_Final_Market_US_Int.(['Year_',num2str(SST_Year_End)]).Total_Seats]),([SST_Final_Market_US_Int.(['Y
ear_',num2str
(SST_Year_End)]).Premium_Seats]),([SST_Final_Market_US_Int.(['Year_',num2str(SST_Year_End)]).OD_AverageElapsedTime_hrs]),([SST_Final_
Market_US_Int.
(['Year_',num2str(SST_Year_End)]).DistStMiles]),([SST_Final_Market_US_Int.(['Year_',num2str(SST_Year_End)]).SST_Premium_Seats]),
```

```
([SST_Final_Market_US_Int.(['Year_',num2str(SST_Year_End)]).SST_Total_Seats]),([SST_Final_Market_US_Int.(['Year_',num2str(SST_Year_End)])
.
Fare_per_mile_cost]),([SST_Final_Market_US_Int.(['Year_',num2str(SST_Year_End)]).generic_cdf]),([SST_Final_Market_US_Int.(['Year_',num2st
r(SST_Year_End)]).
Aircraft_Needed]),([SST_Final_Market_US_Int.(['Year_',num2str(SST_Year_End)]).Number_of_Passenger]));
255 % % %
256 % % % warning( 'off', 'MATLAB:xlswrite:AddSheet' ) ;
257 % % % xlswrite(filename,col_header1_us_int,'Sheet2','A1');
258 % % % xlswrite(filename,col_header2_us_int,'Sheet2','A2');
259 % % % writetable(us_int_data_table,filename,'Sheet',2,'Range','A3','WriteVariableNames',false)
260 % % %
261 % % % %% US_Int_Market
262 % % %
263 % % % load(([SST_Market_Analysis_Dir,'\Int_Market\SST_Forecast\Output\SST_Final_Market_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_Int',date,'.mat']),'SST_Final_Market')
264 % % % SST_Final_Market_Int = SST_Final_Market;
265 % % % clear SST_Final_Market
266 % % %
267 % % % col_header1_int = {'International Market'};
268 % % % col_header2_int =
{'DepAirport_UniqueID','DepAirport','DepIATACtry','DepReg','ArrAirport_UniqueID','ArrAirport','ArrIATACtry','ArrReg','Total_Seats','Premium_S
eats','OD_Av
erageElapsedTime_hrs','DistStMiles','SST_Premium_Seats','SST_Total_Seats','Fare_per_mile_cost','generic_cdf','Aircraft_Needed','Number_of_
Passenger',};
269 % % % int_data_table =
table(([SST_Final_Market_Int.(['Year_',num2str(SST_Year_End)]).DepAirport_UniqueID]),([SST_Final_Market_Int.(['Year_',
num2str(SST_Year_End)]).DepAirport]),([SST_Final_Market_Int.(['Year_',num2str(SST_Year_End)]).DepIATACtry]),([SST_Final_Market_Int.(['Yea
r_',num2str
(SST_Year_End)]).DepReg]),([SST_Final_Market_Int.(['Year_',num2str(SST_Year_End)]).ArrAirport_UniqueID]),([SST_Final_Market_Int.(['Year_',
num2str
(SST_Year_End)]).ArrAirport]),([SST_Final_Market_Int.(['Year_',num2str(SST_Year_End)]).ArrIATACtry]),([SST_Final_Market_Int.(['Year_',num2st
r
(SST_Year_End)]).ArrReg]),([SST_Final_Market_Int.(['Year_',num2str(SST_Year_End)]).Total_Seats]),([SST_Final_Market_Int.(['Year_',num2str(S
ST_Year_End)]).
Premium_Seats]),([SST_Final_Market_Int.(['Year_',num2str(SST_Year_End)]).OD_AverageElapsedTime_hrs]),([SST_Final_Market_Int.(['Year_',n
um2str
(SST_Year_End)]).DistStMiles]),([SST_Final_Market_Int.(['Year_',num2str(SST_Year_End)]).SST_Premium_Seats]),([SST_Final_Market_Int.(['Year
_',num2str
(SST_Year_End)]).SST_Total_Seats]),([SST_Final_Market_Int.(['Year_',num2str(SST_Year_End)]).Fare_per_mile_cost]),([SST_Final_Market_Int.(['
Year_',num2str
(SST_Year_End)]).generic_cdf]),([SST_Final_Market_Int.(['Year_',num2str(SST_Year_End)]).Aircraft_Needed]),([SST_Final_Market_Int.(['Year_',n
um2str
(SST_Year_End)]).Number_of_Passenger]));
270 % % %
271 % % % xlswrite(filename,col_header1_int,'Sheet3','A1');
272 % % % xlswrite(filename,col_header2_int,'Sheet3','A2');
273 % % % writetable(int_data_table,filename,'Sheet',3,'Range','A3','WriteVariableNames',false)
274 % % %
275
276
277 %% Excel file for years 2030 to 2040
278 %% US_Market
279
280 load(([SST_Market_Analysis_Dir,'\US_Market\SST_Forecast\Output\SST_Final_Market_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_US',date,'.mat']),'SST_Final_Market')
281 SST_Final_Market_US = SST_Final_Market;
282 clear SST_Final_Market
283
284 for year = SST_Year_Start:SST_Year_End
285
286 col_header1_us = {'US Market'};
287 col_header2_us =
{'DepAirport_UniqueID','DepAirport','DepIATACtry','DepReg','ArrAirport_UniqueID','ArrAirport','ArrIATACtry','ArrReg','Total_Seats','Premium_S
eats','OD_Av
```

```
erageElapsedTime_hrs','DistStMiles','SST_Premium_Seats','SST_Total_Seats','Fare_per_mile_cost','generic_cdf','Aircraft_Needed','Number_of_
Passenger',};
288 us_data_table =
table(([SST_Final_Market_US.(['Year_',num2str(year)]).DepAirport_UniqueID]),([SST_Final_Market_US.(['Year_',num2str(year)]).
DepAirport]),([SST_Final_Market_US.(['Year_',num2str(year)]).DepIATACtry]),([SST_Final_Market_US.(['Year_',num2str(year)]).DepReg]),
([SST_Final_Market_US.(['Year_',num2str(year)]).ArrAirport_UniqueID]),([SST_Final_Market_US.(['Year_',num2str(year)]).ArrAirport]),([SST_Fin
al_Market_US.
(['Year_',num2str(year)]).ArrIATACtry]),([SST_Final_Market_US.(['Year_',num2str(year)]).ArrReg]),([SST_Final_Market_US.(['Year_',num2str(yea
r)]).Total_Seats]),
([SST_Final_Market_US.(['Year_',num2str(year)]).Premium_Seats]),([SST_Final_Market_US.(['Year_',num2str(year)]).OD_AverageElapsedTime_
hrs]),
([SST_Final_Market_US.(['Year_',num2str(year)]).DistStMiles]),([SST_Final_Market_US.(['Year_',num2str(year)]).SST_Premium_Seats]),([SST_Fin
al_Market_US.
(['Year_',num2str(year)]).SST_Total_Seats]),([SST_Final_Market_US.(['Year_',num2str(year)]).Fare_per_mile_cost]),([SST_Final_Market_US.(['Ye
ar_',num2str
(year)]).generic_cdf]),([SST_Final_Market_US.(['Year_',num2str(year)]).Aircraft_Needed]),([SST_Final_Market_US.(['Year_',num2str(year)]).
Number_of_Passenger]));
289
290 xlswrite(filename,col_header1_us,(['US_Year_',num2str(year)]),'A1');
291 xlswrite(filename,col_header2_us,(['US_Year_',num2str(year)]),'A2');
292 writetable(us_data_table,filename,'Sheet',(['US_Year_',num2str(year)]),'Range','A3','WriteVariableNames',false)
293 end
294
295 %% US_Int_Market
296
297 load(([SST_Market_Analysis_Dir,'\US_Int_Market\SST_Forecast\Output\SST_Final_Market_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_US_Int',date,'.mat']),'SST_Final_Market')
298 SST_Final_Market_US_Int = SST_Final_Market;
299 clear SST_Final_Market
300 for year = SST_Year_Start:SST_Year_End
301
302 col_header1_us_int = {'US_Int Market'};
303 col_header2_us_int =
{'DepAirport_UniqueID','DepAirport','DepIATACtry','DepReg','ArrAirport_UniqueID','ArrAirport','ArrIATACtry','ArrReg','Total_Seats','Premium_S
eats','OD_Av
erageElapsedTime_hrs','DistStMiles','SST_Premium_Seats','SST_Total_Seats','Fare_per_mile_cost','generic_cdf','Aircraft_Needed','Number_of_
Passenger',};
304 us_int_data_table =
table(([SST_Final_Market_US_Int.(['Year_',num2str(year)]).DepAirport_UniqueID]),([SST_Final_Market_US_Int.(['Year_',num2str
(year)]).DepAirport]),([SST_Final_Market_US_Int.(['Year_',num2str(year)]).DepIATACtry]),([SST_Final_Market_US_Int.(['Year_',num2str(year)]).
DepReg]),
([SST_Final_Market_US_Int.(['Year_',num2str(year)]).ArrAirport_UniqueID]),([SST_Final_Market_US_Int.(['Year_',num2str(year)]).ArrAirport]),
([SST_Final_Market_US_Int.(['Year_',num2str(year)]).ArrIATACtry]),([SST_Final_Market_US_Int.(['Year_',num2str(year)]).ArrReg]),([SST_Final_M
arket_US_Int.
(['Year_',num2str(year)]).Total_Seats]),([SST_Final_Market_US_Int.(['Year_',num2str(year)]).Premium_Seats]),([SST_Final_Market_US_Int.(['Yea
r_',num2str
(year)]).OD_AverageElapsedTime_hrs]),([SST_Final_Market_US_Int.(['Year_',num2str(year)]).DistStMiles]),([SST_Final_Market_US_Int.(['Year_',
num2str(year)]).
SST_Premium_Seats]),([SST_Final_Market_US_Int.(['Year_',num2str(year)]).SST_Total_Seats]),([SST_Final_Market_US_Int.(['Year_',num2str(yea
r)]).
Fare_per_mile_cost]),([SST_Final_Market_US_Int.(['Year_',num2str(year)]).generic_cdf]),([SST_Final_Market_US_Int.(['Year_',num2str(year)]).A
ircraft_Needed]),
([SST_Final_Market_US_Int.(['Year_',num2str(year)]).Number_of_Passenger]));
305
306 warning( 'off', 'MATLAB:xlswrite:AddSheet' ) ;
307 xlswrite(filename,col_header1_us_int,(['US_Int_Year_',num2str(year)]),'A1');
308 xlswrite(filename,col_header2_us_int,(['US_Int_Year_',num2str(year)]),'A2');
309 writetable(us_int_data_table,filename,'Sheet',(['US_Int_Year_',num2str(year)]),'Range','A3','WriteVariableNames',false)
310 end
311 %% Int_Market
312
313 load(([SST_Market_Analysis_Dir,'\Int_Market\SST_Forecast\Output\SST_Final_Market_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_Int',date,'.mat']),'SST_Final_Market')
314 SST_Final_Market_Int = SST_Final_Market;
```

```
315 clear SST_Final_Market
316
317 for year = SST_Year_Start:year
318
319 col_header1_int = {'International Market'};
320 col_header2_int =
{'DepAirport_UniqueID','DepAirport','DepIATACtry','DepReg','ArrAirport_UniqueID','ArrAirport','ArrIATACtry','ArrReg','Total_Seats','Premium_S
eats','OD_Av
erageElapsedTime_hrs','DistStMiles','SST_Premium_Seats','SST_Total_Seats','Fare_per_mile_cost','generic_cdf','Aircraft_Needed','Number_of_
Passenger',};
321 int_data_table =
table(([SST_Final_Market_Int.(['Year_',num2str(year)]).DepAirport_UniqueID]),([SST_Final_Market_Int.(['Year_',num2str(year)]).
DepAirport]),([SST_Final_Market_Int.(['Year_',num2str(year)]).DepIATACtry]),([SST_Final_Market_Int.(['Year_',num2str(year)]).DepReg]),([SST_
Final_Market_Int.
(['Year_',num2str(year)]).ArrAirport_UniqueID]),([SST_Final_Market_Int.(['Year_',num2str(year)]).ArrAirport]),([SST_Final_Market_Int.(['Year_',
num2str(year)]).
ArrIATACtry]),([SST_Final_Market_Int.(['Year_',num2str(year)]).ArrReg]),([SST_Final_Market_Int.(['Year_',num2str(year)]).Total_Seats]),([SST_Fi
nal_Market_Int.
(['Year_',num2str(year)]).Premium_Seats]),([SST_Final_Market_Int.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs]),([SST_Final_Market
_Int.(['Year_',
num2str(year)]).DistStMiles]),([SST_Final_Market_Int.(['Year_',num2str(year)]).SST_Premium_Seats]),([SST_Final_Market_Int.(['Year_',num2str(
year)]).
SST_Total_Seats]),([SST_Final_Market_Int.(['Year_',num2str(year)]).Fare_per_mile_cost]),([SST_Final_Market_Int.(['Year_',num2str(year)]).gen
eric_cdf]),
([SST_Final_Market_Int.(['Year_',num2str(year)]).Aircraft_Needed]),([SST_Final_Market_Int.(['Year_',num2str(year)]).Number_of_Passenger]));
322
323 xlswrite(filename,col_header1_int,(['Int_Year_',num2str(year)]),'A1');
324 xlswrite(filename,col_header2_int,(['Int_Year_',num2str(year)]),'A2');
325 writetable(int_data_table,filename,'Sheet',(['Int_Year_',num2str(year)]),'Range','A3','WriteVariableNames',false)
326
327 end
328
329 end
330
1 function [flops_data,mach_overland, mach_overwater, number_of_seats, Fuel_Scaling_Parameter_OL,
Fuel_Scaling_Parameter_OW,hours_per_year,
aircraft_range_overland,aircraft_range_overwater] = Set_parameters_from_flops(Input_Folder_Dir,Flops_Output_File_Name,Sheet_name)
2
3 %% Import data from text file
4 %Read Excel file
5 TableData = xlsread(['C:\',Input_Folder_Dir,'\',Flops_Output_File_Name], Sheet_name);
6
7 %Identify the number of columns in the excel file
8 [~,NoOfColumn] = size(TableData);
9
10 % Create a flops_data struct file from the Excel file
11 flops_data.mission_weight_lb = TableData(1,1:NoOfColumn);
12 flops_data.overland_stage_length_nm = TableData(2,1:NoOfColumn);
13 flops_data.overwater_stage_length_nm = TableData(3,1:NoOfColumn);
14 flops_data.overland_block_time_hrs = TableData(4,1:NoOfColumn);
15 flops_data.overwater_block_time_hrs = TableData(5,1:NoOfColumn);
16 flops_data.overland_block_fuel_lbs = TableData(6,1:NoOfColumn);
17 flops_data.overwater_block_fuel_lbs = TableData(7,1:NoOfColumn);
18 flops_data.takeoff_weight_overland_mission_lbs = TableData(8,1:NoOfColumn);
19 flops_data.takeoff_weight_overwater_mission_lbs =TableData(9,1:NoOfColumn);
20 flops_data.overland_usable_range_nm = TableData(10,1:NoOfColumn);
21 flops_data.overwater_usable_range_nm = TableData(11,1:NoOfColumn);
22 flops_data.operating_empty_weight_lbs = TableData(12,1:NoOfColumn);
23 flops_data.overland_mach_number = TableData(13,1:NoOfColumn);
24 flops_data.overwater_mach_number = TableData(14,1:NoOfColumn);
25 flops_data.maximum_engine_thrust_lbf = TableData(15,1:NoOfColumn);
26 flops_data.mission_payload_lbs = TableData(16,1:NoOfColumn);
27 flops_data.overland_design_range_nm = TableData(17,1:NoOfColumn);
28 flops_data.overwater_design_range_nm = TableData(18,1:NoOfColumn);
29 clear TableData NoOfColumn
30
```

```
%Define additional parameters
mach_overland = flops_data.overland_mach_number(1);
mach_overwater = flops_data.overwater_mach_number(1);
number_of_seats = flops_data.mission_payload_lbs(1) ./ 209;
Fuel_Scaling_Parameter_OL = 1;
Fuel_Scaling_Parameter_OW = 1;
hours_per_year = 3500;
aircraft_range_overland = flops_data.overland_design_range_nm(1);%ceil(flops_data.overland_stage_length_nm(1)/100)*100;
aircraft_range_overwater = flops_data.overwater_design_range_nm(1);%ceil(flops_data.overwater_stage_length_nm(1)/100)*100;


return;


function [] = Integrated_LowBoom_SurrogateModel(Input_Folder_Dir,...
aircraft_range_overland,aircraft_range_overwater,number_of_seats,...
mach_overland,mach_overwater,hours_per_year,number_of_aircraft,...
Fuel_Scaling_Parameter_OL, Fuel_Scaling_Parameter_OW,date,PointDesign,...
flops_data)

Profile_Set = [1;2]; % 1 = overland, 2 = overwater
Load_Factor_Set = [0.85;0.815];
%Load_Factor_Set = [0.85;0.85];

length_of_profile = length(Profile_Set);
length_of_loadfactor = length(Load_Factor_Set);

for loadfactor = 1:length_of_loadfactor

for profile = 1:length_of_profile


% Script to test Karl's and Wu's surrogate models
%

% Inputs
% NLBR - Design Overwater Range, nm
% LBR - Design Overland Range, nm
% PAX - number of seats
% NLBMach = Maximum Mach (oberwater)
% LBMach - low boom profile Mach number (overland)
%

% Outputs

% MTOGW
% Payload
% Operating Empty Weight
% ZFW
% Thrust
% Over Land Range
% Over Water Range
% Over Land Fuel
% Over Water Fuel
% Over Land TOGW
% Over Water TOGW
% Over Land Range
% Over Water Range
% Over Land Time
% Over Water Time

%% Define the aircraft design parameters for Karl's surrogate model

NLBR = aircraft_range_overwater;
LBR = aircraft_range_overland;
PAX = number_of_seats;
```

```
53 NLBMach = mach_overwater;
54 LBMach = mach_overland;
55 PAX_weight = 209;
56 %Fuel_Scaling_Parameter = 1.0;
57
58 %% Define parameters for life cycle cost analysis
59 % Define parameters for aircraft development cost calculation
60
61 %Qproduction = 100:50:1250;
62 Qproduction = number_of_aircraft;
63 % Get maximum speed in knots to estimate the development cost
64 conv_Mach_to_knots = 573.57; % Conversion from...
65 %% Mach number to knots
66 SpeedAboveCruise = 24; % knots (or 0.04 Mach)
67 MaxSpeed_knots = conv_Mach_to_knots * NLBMach +...
68 SpeedAboveCruise;
69
70 % Here we define parameters to estimate the life cycle cost
71 %%of the aircraft
72
73 Flight_Hours_per_Year = hours_per_year;
74 Maintenance_Hours_per_Flight_Hour = 10;
75
76 aircraftProduced = number_of_aircraft;
77 % Number of aircraft produced to estimate the life cycle
78 % cost economics
79 OperationalProfile = Profile_Set(profile);
80 Load_Factor = Load_Factor_Set(loadfactor);
81
82
83 % Print all values to the Command Window
84
85 %clc
86 % % disp(['MTOGW (lb) ', num2str(MTOGW_lb) ])
87 % % disp(['Payload (lb) ', num2str(Payload_lb) ])
88 % % disp(['Operating_Empty_Weight (lb) ',
89 % % num2str(Operating_Empty_Weight_lb ) ])
90 % % disp(['Zero_Fuel_Weight (lb) ', num2str
91 % % (Zero_Fuel_Weight_lb) ])
92 % % disp(['Thrust (lb) ', num2str(Thrust_lb ) ])
93 % % disp(['Overland_Range1_nm ', num2str(Overland_Range1_nm) ])
94 % % disp(['Overwater_Range1_nm ', num2str(Overwater_Range1_nm) ])
95 % % disp(['Overland_Fuel_lb ', num2str(Overland_Fuel_lb) ])
96 % % disp(['Overwater_Fuel_lb ', num2str(Overwater_Fuel_lb) ])
97 % % disp(['Overland_TOGW_lb ', num2str(Overland_TOGW_lb) ])
98 % % disp(['Overwater_TOGW_lb ', num2str(Overwater_TOGW_lb ) ])
99 % % disp(['Overland_Range2_nm ', num2str(Overland_Range2_nm) ])
100 % % disp(['Overwater_Range2_nm ', num2str(Overwater_Range2_nm) ])
101 % % disp(['Overland_Time_nm ', num2str(Overland_Time_nm) ])
102 % % disp(['Overwater_Time_nm ', num2str(Overwater_Time_nm) ])
103 % % disp(' ')
104 % % disp(['Weight factor ', num2str(Weight) ])
105 % %
106 % Make a plot
107 %
108 % figure
109 % plot(PAX,Operating_Empty_Weight_lb,'o--r')
110 % xlabel('Passengers ','fontsize',24)
111 % ylabel('OEW (lb)','fontsize',24)
112 % grid
113
114 % Call Wu's surrogate model
115
116 % [fuelBurn_MLand,fuelBurn_MOverwater,aveSpeed_lb_MLand,aveSpeed_lb_MOverwater,Surrogate_RangeLB_overland, ...
117 % Surrogate_RangeLB_overwater] = surrogate_LB_model(LBR ,NLBR ,Zero_Fuel_Weight_lb,LBMach,NLBMach, MTOGW_lb);
118
```

```
% [fuelBurn_MLand,fuelBurn_MOverwater, aveSpeed_lb_MLand,aveSpeed_lb_MOverwater, Surrogate_RangeLB_overland,...
% Surrogate_RangeLB_overwater ] = blockFuelCalculator(Overland_Fuel_lb,Overwater_Fuel_lb, NLBR, LBR,NLBMach,LBMach );

if PointDesign == 1

[blockFuel_OL, blockFuel_OW, fuelBurn_overland_lb_hr,fuelBurn_overwater_lb_hr, blockTime_OL, blockTime_OW, ...
blockSpeed_OL, blockSpeed_OW, distanceOL,distanceOW, Operating_Empty_Weight_lb, Thrust_lb] = blockFuelCalculator_PointDesign
(flops_data);

else

% Call the function to estimate parameters using Karl's surrogate model

[Payload_lb,Operating_Empty_Weight_lb,Zero_Fuel_Weight_lb,Thrust_lb,Overland_Range1_nm, ...
Overwater_Range1_nm,Overland_Fuel_lb,Overwater_Fuel_lb,Overland_TOGW_lb,Overwater_TOGW_lb, ...
Overland_Range2_nm,Overwater_Range2_nm,Overland_Time_hrs,Overwater_Time_hrs,MTOGW_lb,Weight] = ...
lowBoom_SurrogateModel_weightedPAX_DOE_v5(NLBR,LBR,PAX,NLBMach,LBMach,PAX_weight);

[blockFuel_OL, blockFuel_OW, fuelBurn_overland_lb_hr,fuelBurn_overwater_lb_hr, blockTime_OL, blockTime_OW, ...
blockSpeed_OL, blockSpeed_OW, distanceOL,distanceOW ] = blockFuelCalculator_v2(PAX, MTOGW_lb, Overland_Fuel_lb, ...
Overwater_Fuel_lb, PAX_weight, NLBR, LBR,NLBMach,LBMach, Overland_Time_hrs, Overwater_Time_hrs, Zero_Fuel_Weight_lb, ...
Overland_Range1_nm, Overwater_Range1_nm);
end


%% Start this section to estimate the aircraft development costand the life cycle cost

% % Step 1 = Fuel burn vs distance and average block speed from Karl's design
% % Step 2 = Aircraft development cost over a number of aircraft produced
% % Step 3 = Cost per passenger mile for a fixed quantity produced
%
%% Calculate the cost per airframe over a production run with quantity Qproduction
% This section produces the cost per airframe for the aircraft
% configuration loaded above

[adjustedCostPer_Airframe_2020] = LifeCycleCostModel_QproductionOut(Qproduction,MaxSpeed_knots, ...
Operating_Empty_Weight_lb, Thrust_lb);

% Make a plot to show parametric analysis of developmnent cost of aircraft
% as a function of production run

% Here we calculate the cost for a given quantity produced
% (aircraftProduced). This fixes the quantity produced to a single value to
% estimate the cost per mile over a distance range

%aircraftCost = interp1(Qproduction,adjustedCostPer_Airframe_2020,aircraftProduced);
aircraftCost = adjustedCostPer_Airframe_2020;

% plot_DevelopmentCost(Qproduction,adjustedCostPer_Airframe_2020,aircraftProduced,aircraftCost)

%% Call the function to estimate the life-cycle cost of the aircraft using
% the outputs produced by the development cost and the specific
% performance of each aircraft produced by Karls' model

% [Adjusted_Cost_per_Passenger_Mile,Cost_Per_Mile, Mission_Stage_Lenght_nm] = LowBoom_LCC_Model_v1(aircraftCost, ...
% Flight_Hours_per_Year, Maintenance_Hours_per_Flight_Hour, aveSpeed_lb_MLand, ...
% aveSpeed_lb_MOverwater, fuelBurn_MLand, fuelBurn_MOverwater, OperationalProfile, Thrust_lb , ...
% Surrogate_RangeLB_overland, Surrogate_RangeLB_overwater, Load_Factor, PAX, Fuel_Scaling_Parameter_OL,
Fuel_Scaling_Parameter_OW);
%
[Adjusted_Cost_per_Passenger_Mile,Cost_Per_Mile, Mission_Stage_Lenght_nm] = LowBoom_LCC_Model_v1(aircraftCost, ...
Flight_Hours_per_Year, Maintenance_Hours_per_Flight_Hour, blockSpeed_OL, ...
blockSpeed_OW, fuelBurn_overland_lb_hr, fuelBurn_overwater_lb_hr, OperationalProfile, Thrust_lb, ...
distanceOL, distanceOW, Load_Factor, PAX, Fuel_Scaling_Parameter_OL, Fuel_Scaling_Parameter_OW);


```

```
% Make a plot to show parametric analysis of cost per passenger mile ($/nm)
% as a function of distance flown

plot_CostPerSeatMile(Mission_Stage_Lenght_nm,Adjusted_Cost_per_Passenger_Mile)

% Create an array with results for CPM and distance

results = [Mission_Stage_Lenght_nm' Adjusted_Cost_per_Passenger_Mile'];

%% Create Output directory

if exist(['..\',Input_Folder_Dir,'\Fare_data\',num2str(number_of_seats),'_Seats']),'dir') == 0

mkdir(['..\',Input_Folder_Dir,'\Fare_data\',num2str(number_of_seats),'_Seats']))

end %if exist(['..\',Input_Folder_Dir,'\Fare_data\',num2str(number_of_seats),'_Seats)']),'dir') == 0

addpath(['..\',Input_Folder_Dir,'\Fare_data\',num2str(number_of_seats),'_Seats']))

%%

if loadfactor == 1

if profile == 1

cost_per_passenger_mile.Over_land.distance = results(:,1);
cost_per_passenger_mile.Over_land.fare = results(:,2);
cost_per_passenger_mile.data.blockFuel_OL = blockFuel_OL;
cost_per_passenger_mile.data.blockFuel_OW = blockFuel_OW;
cost_per_passenger_mile.data.fuelBurn_overland_lb_hr = fuelBurn_overland_lb_hr;
cost_per_passenger_mile.data.fuelBurn_overwater_lb_hr = fuelBurn_overwater_lb_hr;
cost_per_passenger_mile.data.blockTime_OL = blockTime_OL;
cost_per_passenger_mile.data.blockTime_OW = blockTime_OW;
cost_per_passenger_mile.data.blockSpeed_OL = blockSpeed_OL;
cost_per_passenger_mile.data.blockSpeed_OW = blockSpeed_OW;
cost_per_passenger_mile.data.distanceOL = distanceOL;
cost_per_passenger_mile.data.distanceOW = distanceOW;
else

cost_per_passenger_mile.Over_water.distance = results(:,1);
cost_per_passenger_mile.Over_water.fare = results(:,2);
cost_per_passenger_mile.data.blockFuel_OL = blockFuel_OL;
cost_per_passenger_mile.data.blockFuel_OW = blockFuel_OW;
cost_per_passenger_mile.data.fuelBurn_overland_lb_hr = fuelBurn_overland_lb_hr;
cost_per_passenger_mile.data.fuelBurn_overwater_lb_hr = fuelBurn_overwater_lb_hr;
cost_per_passenger_mile.data.blockTime_OL = blockTime_OL;
cost_per_passenger_mile.data.blockTime_OW = blockTime_OW;
cost_per_passenger_mile.data.blockSpeed_OL = blockSpeed_OL;
cost_per_passenger_mile.data.blockSpeed_OW = blockSpeed_OW;
cost_per_passenger_mile.data.distanceOL = distanceOL;
cost_per_passenger_mile.data.distanceOW = distanceOW;
end
%% Save Results

save(['..\',Input_Folder_Dir,'\Fare_data\',num2str(number_of_seats),'_Seats\Fare_Low_Boom_US',date,'.mat']),'cost_per_passenger_mile')
save(['..\',Input_Folder_Dir,'\Fare_data\',num2str(number_of_seats),'_Seats\Fare_Low_Boom_US.mat']),'cost_per_passenger_mile')

else

if profile ==1

cost_per_passenger_mile.Over_land.distance = results(:,1);
cost_per_passenger_mile.Over_land.fare = results(:,2);
cost_per_passenger_mile.data.blockFuel_OL = blockFuel_OL;
cost_per_passenger_mile.data.blockFuel_OW = blockFuel_OW;
cost_per_passenger_mile.data.fuelBurn_overland_lb_hr = fuelBurn_overland_lb_hr;
```

```
248 cost_per_passenger_mile.data.fuelBurn_overwater_lb_hr = fuelBurn_overwater_lb_hr;
249 cost_per_passenger_mile.data.blockTime_OL = blockTime_OL;
250 cost_per_passenger_mile.data.blockTime_OW = blockTime_OW;
251 cost_per_passenger_mile.data.blockSpeed_OL = blockSpeed_OL;
252 cost_per_passenger_mile.data.blockSpeed_OW = blockSpeed_OW;
253 cost_per_passenger_mile.data.distanceOL = distanceOL;
254 cost_per_passenger_mile.data.distanceOW = distanceOW;
255 else
256
257 cost_per_passenger_mile.Over_water.distance = results(:,1);
258 cost_per_passenger_mile.Over_water.fare = results(:,2);
259 cost_per_passenger_mile.data.blockFuel_OL = blockFuel_OL;
260 cost_per_passenger_mile.data.blockFuel_OW = blockFuel_OW;
261 cost_per_passenger_mile.data.fuelBurn_overland_lb_hr = fuelBurn_overland_lb_hr;
262 cost_per_passenger_mile.data.fuelBurn_overwater_lb_hr = fuelBurn_overwater_lb_hr;
263 cost_per_passenger_mile.data.blockTime_OL = blockTime_OL;
264 cost_per_passenger_mile.data.blockTime_OW = blockTime_OW;
265 cost_per_passenger_mile.data.blockSpeed_OL = blockSpeed_OL;
266 cost_per_passenger_mile.data.blockSpeed_OW = blockSpeed_OW;
267 cost_per_passenger_mile.data.distanceOL = distanceOL;
268 cost_per_passenger_mile.data.distanceOW = distanceOW;
269 end
270
271
272 %% Save Results
273
274
save((['..\',Input_Folder_Dir,'\Fare_data\',num2str(number_of_seats),'_Seats\Fare_Low_Boom_Int',date,'.mat']),'cost_per_passenger_mile')
275 save((['..\',Input_Folder_Dir,'\Fare_data\',num2str(number_of_seats),'_Seats\Fare_Low_Boom_Int.mat']),'cost_per_passenger_mile')
276
277
278 end
279
280
281 close all
282
283 end
284 end
285
286 return;
287
1 function [blockFuel_OL, blockFuel_OW, fuelBurn_overland_lb_hr,fuelBurn_overwater_lb_hr, blockTime_OL, blockTime_OW, ...
2 blockSpeed_OL, blockSpeed_OW, distanceOL,distanceOW, Operating_Empty_Weight_lb, Thrust_lb] =
blockFuelCalculator_PointDesign(flops_data)
3
4
5 distanceOL = 1000:100:ceil(flops_data.overland_stage_length_nm(1)/100)*100;
6 distanceOW = 1000:100:ceil(flops_data.overwater_stage_length_nm(1)/100)*100;
7
8 %% NEW Regression equation for over land block fuel
9 coeffFuel_OL = polyfit(flops_data.overland_stage_length_nm,flops_data.overland_block_fuel_lbs,2); % Second-order polynomial model
10 blockFuel_OL = polyval(coeffFuel_OL,distanceOL);
11
12 %% NEW Regression equation for over water block fuel
13 coeffFuel_OW = polyfit(flops_data.overwater_stage_length_nm,flops_data.overwater_block_fuel_lbs,2); % Second-order polynomial model
14 blockFuel_OW = polyval(coeffFuel_OW,distanceOW);
15
16 %% NEW Regression equation for over land block times
17 coeffTime_OL = polyfit(flops_data.overland_stage_length_nm,flops_data.overland_block_time_hrs,2); % Linear model
18 blockTime_OL = polyval(coeffTime_OL,distanceOL);
19
20 blockSpeed_OL = distanceOL ./ blockTime_OL; % Block time overwater vs distanceOW vector
21
22 %% NEW Regression equation for over water block times
23 coeffTime_OW = polyfit(flops_data.overwater_stage_length_nm,flops_data.overwater_block_time_hrs,2); % Linear model
24 blockTime_OW = polyval(coeffTime_OW,distanceOW);
```

```
25
26 blockSpeed_OW = distanceOW ./ blockTime_OW;
27
28 % calculate fuel burns from block times
29
30 fuelBurn_overland_lb_hr = blockFuel_OL ./ blockTime_OL;
31 fuelBurn_overwater_lb_hr = blockFuel_OW ./ blockTime_OW;
32
33 Operating_Empty_Weight_lb = flops_data.operating_empty_weight_lbs(1);
34 Thrust_lb = flops_data.maximum_engine_thrust_lbf(1);
35
36 end
37
1 % Surrogate model from NASA Langley
2 % Conversion of Excel model equations
3 % developed by Karl
4
5 % Converted by A. Trani
6
7 function [Payload_lb,Operating_Empty_Weight_lb,Zero_Fuel_Weight_lb,Thrust_lb,Overland_Range1_nm, ...
8 Overwater_Range1_nm,Overland_Fuel_lb,Overwater_Fuel_lb,Overland_TOGW_lb,Overwater_TOGW_lb, ...
9 Overland_Range2_nm,Overwater_Range2_nm,Overland_Time_hrs,Overwater_Time_hrs,MTOGW_lb,Weight] = ...
10 lowBoom_SurrogateModel_weightedPAX_DOE_v5(NLBR,LBR,PAX,NLBMach,LBMach,PAX_weight)
11
12
13 % Inputs
14 % NLBR = Design Overwater Range, nm 3800
15 % LBR Design Overland Range, nm 3000
16 % PAX = 43
17 % NLBMach = Maximum Mach 1.8
18 % LBMach 1.7
19 %
20
21 % Outputs
22
23 % MTOGW
24 % Payload
25 % Operating Empty Weight
26 % ZFW
27 % Thrust
28 % Over Land Range
29 % Over Water Range
30 % Over Land Fuel
31 % Over Water Fuel
32 % Over Land TOGW
33 % Over Water TOGW
34 % Over Land Range
35 % Over Water Range
36 % Over Land Time
37 % Over Water Time
38
39 % Inputs apply to both 43 and 52 seat aircraft
40
41 Weight = (52-PAX)/9;
42
43 % fuelFraction_Overland = 0.98;
44 % fuelFraction_Overwater = 1.00;
45
46 % Start equations for 43 passenger aircraft
47
48 Payload_1 = PAX_weight .* PAX; % weight in pounds
49 Operating_Empty_Weight_1 = 42804.26 + -0.9968504*LBR + 2.245264*NLBR + 196.0699*PAX + 0.00037543*LBR.*LBR +
0.001294748*NLBR.*NLBR
+ 0.2067518*PAX.*PAX + 8612.945*LBMach.*LBMach + -0.0002413581*LBR.*NLBR + 0.05139336*NLBR.*PAX + -6.007268*NLBR.*LBMach + -
118.7405
*PAX.*LBMach;
```

```
50 Zero_Fuel_Weight_1 = 42804.26 + -0.9968504*LBR + 2.245264*NLBR + 405.0699*PAX + 0.00037543*LBR.*LBR + 0.001294748*NLBR.*NLBR
+ 0.2067518*PAX.*PAX + 8612.945*LBMach.*LBMach + -0.0002413581*LBR.*NLBR + 0.05139336*NLBR.*PAX + -6.007268*NLBR.*LBMach + -
118.7405
*PAX.*LBMach;
51 Thrust_1 = 15419.58 + -1.551947*LBR + 3.469119*NLBR + 131.0557*PAX + 0.0005835815*LBR.*LBR + 0.002030553*NLBR.*NLBR
+ 0.36345*PAX.*PAX + 12500*LBMach.*LBMach + -0.000374592*LBR.*NLBR + 0.08074544*NLBR.*PAX + -9.427754*NLBR.*LBMach + -
188.9189*PAX.
*LBMach;
52 Overland_Range1_1 = 16.93008 + 1.020703*LBR + 59.01307*LBMach + -0.0001736223*LBR.*LBMach;
53 Overwater_Range1_1 = 16.38548 + 1.02077*NLBR + 59.2079*LBMach + -9.759192E-09*NLBR*NLBR + -0.0001586204*NLBR*LBMach;
54 Overland_Fuel_1 = -5262.437 + 14.21307*LBR + 4.119441*NLBR + 397.7017*PAX + 0.003320731*NLBR.*NLBR + 27976.51*LBMach.
*LBMach + 0.00117847*LBR.*NLBR + 0.04035596*LBR.*PAX + 0.1330613*NLBR.*PAX + -21.16781*NLBR.*LBMach + -513.2906*PAX.*LBMach;
55 Overwater_Fuel_1 = 10889.26 + 12.82116*NLBR + 0.006320012*NLBR.*NLBR + 1.178589*PAX.*PAX + 41001.97*LBMach.*LBMach +
0.2797478*NLBR.*PAX + -31.86764*NLBR.*LBMach + -583.3308*PAX.*LBMach;
56 Overland_TOGW_1 = 45889.35 + 10.28922*LBR + 6.577221*NLBR + 869.562*PAX + 0.001067075*LBR.*LBR + 0.004934355*NLBR.*NLBR +
38815.43*LBMach.*LBMach + 0.0009692852*LBR.*NLBR + 0.03788696*LBR.*PAX + 0.1979802*NLBR.*PAX + -28.8521*NLBR.*LBMach + -
667.9332*PAX.
*LBMach;
57 Overwater_TOGW_1 = 56167.97 + 14.84903*NLBR + 559.8684*PAX + 0.00809869*NLBR.*NLBR + 1.387509*PAX.*PAX + 53337.94*LBMach.
*LBMach + 0.3513936*NLBR.*PAX + -40.08902*NLBR.*LBMach + -815.7946*PAX.*LBMach;
58 Overland_Range2_1 = LBR;
59 Overwater_Range2_1 = NLBR;
60 Overland_Time_1 = 2.972073 + 0.002106141*LBR + 4.081442E-05*NLBR + -2.598496*LBMach + -5.094999E-09*NLBR.*NLBR + 0.8094399
*LBMach.*LBMach + -4.620391E-09*LBR.*NLBR + -0.0006050385*LBR.*LBMach + -2.413128E-07*NLBR.*PAX;
61 Overwater_Time_1 = 5.29057 + 0.002039345*NLBR + -0.002831138*PAX + -5.02619*LBMach + -5.270265E-09*NLBR.*NLBR + 2.223648E-
05*PAX.*PAX + 1.483532*LBMach.*LBMach + -1.306243E-06*LBR.*LBMach + -0.0005660546*NLBR.*LBMach;
62 MTOGW_1 = 55668.58 + 14.19788*NLBR + 625.6466*PAX + 0.001404418*LBR.*LBR + 0.00877323*NLBR.*NLBR + 1.449349*PAX.
*PAX + 54012.51*LBMach.*LBMach + -0.001698949*LBR.*NLBR + 0.3481102*NLBR.*PAX + -40.34256*NLBR.*LBMach + -
847.2526*PAX.*LBMach;
63
64 % Define variables for 52 seater per Karl's nomenclature
65
66 vNLBR = NLBR;
67 vLBR = LBR;
68 vPAX = PAX;
69 vNLBMach = NLBMach;
70 vLBMach = LBMach;
71
72 % Populate the values for the 52 seater aircraft (second array element)
73
74 Payload_2 = PAX_weight .* PAX; % weight in pounds
75 Operating_Empty_Weight_2 = 121040.5 + -25.41987*vNLBR + -339.5284*vPAX + 0.003212665*vNLBR.*vNLBR + 1.434517*vPAX.*vPAX +
0.1015964*vNLBR.*vPAX;
76 Zero_Fuel_Weight_2 = 121040.5 + -25.41987*vNLBR + -130.5284*vPAX + 0.003212665*vNLBR.*vNLBR + 1.434517*vPAX.*vPAX + 0.1015964
*vNLBR.*vPAX;
77 Thrust_2 = 112450.7 + -35.56117*vNLBR + -634.0657*vPAX + 0.004490705*vNLBR.*vNLBR + 2.023374*vPAX.*vPAX + 0.142149
*vNLBR.*vPAX;
78 Overland_Range1_2 = 120.2812 + 1.018251*vLBR + 0.0000003825535*vLBR.*vLBR;
79 Overwater_Range1_2 = 118.1901 + 1.023058*vNLBR + -0.0000003565252*vNLBR.*vNLBR;
80 Overland_Fuel_2 = 326111.9 + 27.8378*vLBR + -147.0038*vNLBR + -2208.08*vPAX + 0.01674828*vNLBR.*vNLBR + 4.942013*vPAX.
*vPAX + -0.001250373*vLBR.*vNLBR + 0.4749506*vNLBR.*vPAX;
81 Overwater_Fuel_2 = 314300.5 + -110.1917*vNLBR + -2143.408*vPAX + 0.01411128*vNLBR.*vNLBR + 4.812647*vPAX.*vPAX + 0.449615
*vNLBR.*vPAX;
82 Overland_TOGW_2 = 603493 + 15.14462*vLBR + -223.3731*vNLBR + -3651.792*vPAX + 0.001559828*vLBR.*vLBR + 0.02491564*vNLBR.
*vNLBR + 12.39115*vPAX.*vPAX + 0.7645333*vNLBR.*vPAX;
83 Overwater_TOGW_2 = 478808.5 + -147.6084*vNLBR + -2804.985*vPAX + 0.01868412*vNLBR.*vNLBR + 9.53697*vPAX.*vPAX + 0.6113751
*vNLBR.*vPAX;
84 Overland_Range2_2 = -0.02173565 + 1.000002*vLBR + 0.004486606*vLBMach.*vLBMach;
85 Overwater_Range2_2 = 0.003711473 + 0.999999*vNLBR;
86 Overland_Time_2 = 0.4570523 + 0.001428535*vLBR + 0.01577379*vPAX + -0.00000005104635*vLBR.*vLBR + -0.00004206952*vPAX.
*vPAX + -0.000001478277*vLBR.*vPAX + -0.000002648109*vNLBR.*vPAX;
87 Overwater_Time_2 = -4.606022 + 0.003727151*vNLBR + 0.04239966*vPAX + -0.0000003226989*vNLBR.*vNLBR + -0.00009439455*vPAX.
*vPAX + -0.000009792375*vNLBR.*vPAX;
88 MTOGW_2 = max(Overland_TOGW_2,Overwater_TOGW_2);
89
90 % Calculate the averages
```

```
91
92 Payload_lb = mean(Payload_1,Payload_2); % weight in pounds
93 Operating_Empty_Weight_lb = Weight .*Operating_Empty_Weight_1 + Operating_Empty_Weight_2 .*(1-Weight);
94 Zero_Fuel_Weight_lb = Weight .*Zero_Fuel_Weight_1 + Zero_Fuel_Weight_2 .*(1-Weight);
95 Thrust_lb = Weight .*Thrust_1 + Thrust_2 .*(1-Weight);
96 Overland_Range1_nm = Weight .*Overland_Range1_1 + Overland_Range1_2 .*(1-Weight);
97 Overwater_Range1_nm = Weight .*Overwater_Range1_1 + Overwater_Range1_2 .*(1-Weight);
98 Overland_Fuel_lb = Weight .*Overland_Fuel_1 + Overland_Fuel_2 .*(1-Weight);
99 Overwater_Fuel_lb = Weight .*Overwater_Fuel_1 + Overwater_Fuel_2 .*(1-Weight);
100 Overland_TOGW_lb = Weight .*Overland_TOGW_1 + Overland_TOGW_2 .*(1-Weight);
101 Overwater_TOGW_lb = Weight .*Overwater_TOGW_1 + Overwater_TOGW_2 .*(1-Weight);
102 Overland_Range2_nm = Weight .*Overland_Range2_1 + Overland_Range2_2 .*(1-Weight);
103 Overwater_Range2_nm = Weight .*Overwater_Range2_1 + Overwater_Range2_2 .*(1-Weight);
104 Overland_Time_hrs = Weight .*Overland_Time_1 + Overland_Time_2 .*(1-Weight);
105 Overwater_Time_hrs = Weight .*Overwater_Time_1 + Overwater_Time_2 .*(1-Weight);
106 MTOGW_lb = max(Overland_TOGW_lb ,Overwater_TOGW_lb);
107
108 %
109 % % New section that deviates from NASA's surrogate model
110
111 % % Generate the Block fuel and Block time plots
112 %
113 % % Load the data for Mach Trade LB and Mach Trade Overwater
114 %
115 % % Data contains:
116 % %
117 % % R1 = Range (nm)
118 % % R2 = Block fuel (lb)
119 % % R3 = Block Time (hr)
120 %
121 % load MachTrade_OverwaterData.m
122 % load MachTrade_lowBoomData.m
123 %
124 % % Create new variables
125 %
126 % Reference_RangeLB_overland = MachTrade_lowBoomData(1,:);
127 % Reference_fuelLB_overland = MachTrade_lowBoomData(2,:);
128 % Reference_timeLB_overland = MachTrade_lowBoomData(3,:);
129 %
130 % Reference_RangeLB_overwater = MachTrade_OverwaterData(1,:);
131 % Reference_fuelLB_overwater = MachTrade_OverwaterData(2,:);
132 % Reference_timeLB_overwater = MachTrade_OverwaterData(3,:);
133 %
134 % % Generate new data for new design vehicle fuel (low boom overland)
135 % Call Wu's surrogate model here
136
137 % FuelSurrogate_LB_overland = MTOGW_lb+(Reference_RangeLB_overland-(MaximumRange/2500)*2662.385).*(MTOGW_lb-
(zero_fuel_weight_lb/76256).*(154481-25.812092*2662.385))./(2662.385.*(MaximumRange/2500)) ...
138 % - zero_fuel_weight_lb - 1.2473264*Reference_RangeLB_overland.*(2500./MaximumRange).*(MTOGW_lb/154481) - 5632.276*
(2*zero_fuel_weight_lb/76256-1);
139 % %
140 % BlockTimeSurrogate_LB_overland = 0.0011025*1.6*Reference_RangeLB_overland./MachNumber+0.817904*MachNumber/1.6;
141 % BlockSpeed_LB_overland = Reference_RangeLB_overland ./ BlockTimeSurrogate_LB_overland;
142 % %
143 % % % Generate new data for new design vehicle fuel (low boom overwater)
144 % %
145 %
146 % FuelSurrogate_LB_overwater = MTOGW_lb+(Reference_RangeLB_overwater-(MaxRange_overwater/3600)*3856.099).*(MTOGW_lb-
(zero_fuel_weight_lb/76256)*(154481-18.8791*3856.099))./(3856.099.*(MaxRange_overwater/3600)) ...
147 % - zero_fuel_weight_lb - 0.8799807*Reference_RangeLB_overwater.*(3600/MaxRange_overwater).*(MTOGW_lb/154481) - 5510.527*
(2*zero_fuel_weight_lb/76256-1);
148 % %
149 % BlockTimeSurrogate_LB_overwater =
0.0009687*1.8*Reference_RangeLB_overwater./Mach_overwater+0.909704*Mach_overwater/1.8;
150 % BlockSpeed_LB_overwater = Reference_RangeLB_overwater ./ BlockTimeSurrogate_LB_overwater;
151 % %
```

```
152 %
153 % % Plot the block fuel (lb) vs range (nm)
154 % figure
155 % plot(Reference_RangeLB_overland,FuelSurrogate_LB_overland,'o-r',Reference_RangeLB_overwater ,FuelSurrogate_LB_overwater,'^-b',
...
156 % Reference_RangeLB_overland,Reference_fuelLB_overland,'*-k',Reference_RangeLB_overwater,Reference_fuelLB_overwater,'+-g')
157 % xlabel('Range (nm)','fontsize',24)
158 % ylabel('Block Fuel (lb)','fontsize',24)
159 % legend('Surrogate Low-Boom Overland','Surrogate Low-Boom Overwater','FLOPS Low-Boom Overland', 'FLOPS Low-Boom Overwater')
160 % grid
161 %
162 % % Plot the block time (lb) vs range (nm)
163 % figure
164 % plot(Reference_RangeLB_overland,BlockTimeSurrogate_LB_overland,'o-
r',Reference_RangeLB_overwater,BlockTimeSurrogate_LB_overwater,'^-b', ...
165 % Reference_RangeLB_overland,Reference_timeLB_overland,'*-k',Reference_RangeLB_overwater,Reference_timeLB_overwater,'+-g');
166 % xlabel('Range (nm)','fontsize',24)
167 % ylabel('Block Time (hr)','fontsize',24)
168 % legend('Surrogate Low-Boom Overland','Surrogate Low-Boom Overwater','FLOPS Low-Boom Overland', 'FLOPS Low-Boom Overwater')
169 % grid
170 % %
171 % % Generate parameters for life cycle cost model
172 %
173 % fuelBurn_overland_lb_hr = FuelSurrogate_LB_overland ./ BlockTimeSurrogate_LB_overland;
174 % fuelBurn_overwater_lb_hr = FuelSurrogate_LB_overwater ./ BlockTimeSurrogate_LB_overwater;
175 % %
176 % Reference_fuelBurn_overland_lb_hr = Reference_fuelLB_overland ./ Reference_timeLB_overland;
177 % Reference_fuelBurn_overwater_lb_hr = Reference_fuelLB_overwater ./ Reference_timeLB_overwater;
178 % %
179 % % Plot the fuel burn (lb/hr) vs range (nm)
180 % figure
181 % plot(Reference_RangeLB_overland,fuelBurn_overland_lb_hr,'o-
r',Reference_RangeLB_overwater,Reference_fuelBurn_overwater_lb_hr,'^-b', ...
182 % Reference_RangeLB_overland,Reference_fuelBurn_overland_lb_hr,'*-
k',Reference_RangeLB_overwater,Reference_fuelBurn_overwater_lb_hr,'+-g');
183 % xlabel('Range (nm)','fontsize',24)
184 % ylabel('Fuel Burn (lb/hr)','fontsize',24)
185 % legend('Surrogate Low-Boom Overland','Surrogate Low-Boom Overwater','FLOPS Low-Boom Overland', 'FLOPS Low-Boom Overwater')
186 % grid
187 % %
188
189 end
190
```

```
1 function [blockFuel_OL, blockFuel_OW, fuelBurn_overland_lb_hr,fuelBurn_overwater_lb_hr, blockTime_OL, blockTime_OW, ...
2 blockSpeed_OL, blockSpeed_OW, distanceOL,distanceOW ] = blockFuelCalculator_v2(PAX, MTOGW_lb, Overland_Fuel_lb, ...
3 Overwater_Fuel_lb, PAX_weight, NLBR, LBR,NLBMach,LBMach, Overland_Time_hrs, Overwater_Time_hrs, Zero_Fuel_Weight_lb, ...
4 Overland_Range1_nm, Overwater_Range1_nm)
5
6 % Function to produce fuel burn and block time calculations over various
7 % distance values
8
9 % Function outputs:
10
11 % blockFuel_OL - Block fuel (in pounds) vector for over water operations. Vector
12 % that applies to distance values defined by variable distanceOL
13 % blockFuel_OW - Block fuel (in pounds) vector for over water operations. Vector
14 % that applies to distance values defined by variable distanceOW
15
16 % fuelBurn_overland_lb_hr = vector of fuel burn vs distance (over land) in
17 % fuelBurn_overwater_lb_hr = vector of fuel burn vs distance (over water)
18
19 % blockTime_OL - Block time (in hours) vector for over land operations. Vector
20 % that applies to distance values defined by variable distanceOL
21 % blockTime_OW - Block time (in hours) vector for over water operations. Vector
22 % that applies to distance values defined by variable distanceOW
```

```
23
24 % blockSpeed_OL - block speed vs distance vector (knots) over land
25 % mission
26 % blockSpeed_OW - block speed vs distance vector (knots) - over
27 % water mission
28 %
29
30 % distanceOL - distance vector for over land conditios. Nominally defined
31 % from 1000 to LBR (Low Boom Range)
32 % distanceOW - distance vector for over water conditios. Nominally defined
33 % from 1000 to LBR (Low Boom Range)
34
35 % Function inputs (all from Karl's surrogate model):
36
37 % PAX - number of passengers
38 % MTOGW_lb - max. takeoff gross weight
39 % Overland_Fuel_lb - from
40 % Overwater_Fuel_lb, PAX_weight, NLBR, LBR,NLBMach,LBMach, Overland_Time_hrs, Overwater_Time_hrs, Zero_Fuel_Weight_lb, ...
41 % Overland_Range1_nm, Overwater_Range1_nm
42
43 % Create a distance arrayPAX, MTOGW_lb, Overland_Fuel_lb, ...
44
45 distanceOL = 1000:100:LBR;
46 distanceOW = 1000:100:NLBR;
47
48 % New code to calculate the block fuel as a function of distance for
49 % overland operations
50
51 % Use Karl's surrogate model to find block fuel overland as a function of
52 % the distanceOL vector above
53
54 % Karls surrogate model;
55 % Call the function to estimate parameters using Karl's surrogate model
56 %
57 %% Here we execute the function " lowBoom_SurrogateModel_weightedPAX_DOE_v5"
58 % in vector form to obtain multiple values of:
59
60 % Overland_Range_Operational_nm
61 % Overland_Fuel_lb
62
63 % These values are obtained for each value of distanceOL - vector of
64 % distances from 1000 to LBR (low boom range)
65
66 [Payload_lb,Operating_Empty_Weight_lb,Zero_Fuel_Weight_lb,Thrust_lb,Overland_Range_Operational_nm, ...
67 Overwater_Range_Operational_nm,Overland_Fuel_lb,Overwater_Fuel_lb,Overland_TOGW_lb,Overwater_TOGW_lb, ...
68 Overland_Range2_nm,Overwater_Range2_nm,Overland_Time_hrs,Overwater_Time_hrs,MTOGW_lb,Weight] = ...
69 lowBoom_SurrogateModel_weightedPAX_DOE_v5(NLBR,distanceOL,PAX,NLBMach,LBMach,PAX_weight);
70
71 % Regression equation for over land block fuel
72 coeffFuel_OL = polyfit(Overland_Range_Operational_nm,Overland_Fuel_lb,2); % Second-order polynomial model
73 blockFuel_OL = polyval(coeffFuel_OL,distanceOL); % Block fuel overland vs distanceOL vector
74
75 % Regression equation for over land block times
76 coeffTime_OL = polyfit(Overland_Range_Operational_nm,Overland_Time_hrs,2); % Linear model
77 blockTime_OL = polyval(coeffTime_OL,distanceOL); % Block time overland vs distanceOL vector
78
79 blockSpeed_OL = distanceOL ./ blockTime_OL; % Block time overwater vs distanceOW vector
80
81 % Regression equation to predict the over water block fuel
82 % Use Wu's surrogate model equations to calculate the block time for a given distance
83 % over water
84
85 % Save only one value of MTOGW and ZFW from the previous calculation
86
87 MTOGW_OW_lb = MTOGW_lb(1); % MTOGW is the largest of over land TOGW and over water TOGW
88 Zero_Fuel_Weight_OW_lb = Zero_Fuel_Weight_lb(1); % ZFW is required in Wu's surrogate model
```

```
89
90 % Clear all unnecessary variables
91 clear Payload_lb Operating_Empty_Weight_lb Zero_Fuel_Weight_lb Thrust_lb Overland_Range_Operational_nm ...
92 Overwater_Range_Operational_nm Overland_Fuel_lb Overwater_Fuel_lb Overland_TOGW_lb Overwater_TOGW_lb ...
93 Overland_Range2_nm Overwater_Range2_nm Overland_Time_hrs Overwater_Time_hrs MTOGW_lb Weight
94
95 %% Estimate the block fuel over water and block time using Wu's surrogate model equations
96 %
97
98
99 blockFuel_OW = MTOGW_OW_lb+(distanceOW-(NLBR/3600)*3856.099).*(MTOGW_OW_lb-(Zero_Fuel_Weight_OW_lb/76256)*(154481-
18.8791*3856.
099))./(3856.099.*(NLBR/3600)) ...
100 - Zero_Fuel_Weight_OW_lb - 0.8799807*distanceOW.*(3600./NLBR).*(MTOGW_OW_lb/154481) -
5510.527*(2*Zero_Fuel_Weight_OW_lb/76256-1);
101
102 blockTime_OW= 0.0009687*1.8*distanceOW./NLBMach+0.909704*NLBMach/1.8;
103 blockSpeed_OW= distanceOW ./ blockTime_OW;
104
105 % calculate fuel burns from block times
106
107 fuelBurn_overland_lb_hr = blockFuel_OL ./ blockTime_OL;
108 fuelBurn_overwater_lb_hr = blockFuel_OW ./ blockTime_OW;
109
110 end
111
112
```

```
1 % Surrogate model from NASA Langley
2 % Conversion of Excel model equations
3 % developed by Karl
4
5 % Converted by A. Trani
6
7 function [Payload_lb,Operating_Empty_Weight_lb,Zero_Fuel_Weight_lb,Thrust_lb,Overland_Range1_nm, ...
8 Overwater_Range1_nm,Overland_Fuel_lb,Overwater_Fuel_lb,Overland_TOGW_lb,Overwater_TOGW_lb, ...
9 Overland_Range2_nm,Overwater_Range2_nm,Overland_Time_hrs,Overwater_Time_hrs,MTOGW_lb,Weight] = ...
10 lowBoom_SurrogateModel_weightedPAX_DOE_v5(NLBR,LBR,PAX,NLBMach,LBMach,PAX_weight)
11
12
13 % Inputs
14 % NLBR = Design Overwater Range, nm 3800
15 % LBR Design Overland Range, nm 3000
16 % PAX = 43
17 % NLBMach = Maximum Mach 1.8
18 % LBMach 1.7
19 %
20
21 % Outputs
22
23 % MTOGW
24 % Payload
25 % Operating Empty Weight
26 % ZFW
27 % Thrust
28 % Over Land Range
29 % Over Water Range
30 % Over Land Fuel
31 % Over Water Fuel
32 % Over Land TOGW
33 % Over Water TOGW
34 % Over Land Range
35 % Over Water Range
36 % Over Land Time
37 % Over Water Time
38
```

```
39 % Inputs apply to both 43 and 52 seat aircraft
40
41 Weight = (52-PAX)/9;
42
43 % fuelFraction_Overland = 0.98;
44 % fuelFraction_Overwater = 1.00;
45
46 % Start equations for 43 passenger aircraft
47
48 Payload_1 = PAX_weight .* PAX; % weight in pounds
49 Operating_Empty_Weight_1 = 42804.26 + -0.9968504*LBR + 2.245264*NLBR + 196.0699*PAX + 0.00037543*LBR.*LBR +
0.001294748*NLBR.*NLBR
+ 0.2067518*PAX.*PAX + 8612.945*LBMach.*LBMach + -0.0002413581*LBR.*NLBR + 0.05139336*NLBR.*PAX + -6.007268*NLBR.*LBMach + -
118.7405
*PAX.*LBMach;
50 Zero_Fuel_Weight_1 = 42804.26 + -0.9968504*LBR + 2.245264*NLBR + 405.0699*PAX + 0.00037543*LBR.*LBR + 0.001294748*NLBR.*NLBR
+ 0.2067518*PAX.*PAX + 8612.945*LBMach.*LBMach + -0.0002413581*LBR.*NLBR + 0.05139336*NLBR.*PAX + -6.007268*NLBR.*LBMach + -
118.7405
*PAX.*LBMach;
51 Thrust_1 = 15419.58 + -1.551947*LBR + 3.469119*NLBR + 131.0557*PAX + 0.0005835815*LBR.*LBR + 0.002030553*NLBR.*NLBR
+ 0.36345*PAX.*PAX + 12500*LBMach.*LBMach + -0.000374592*LBR.*NLBR + 0.08074544*NLBR.*PAX + -9.427754*NLBR.*LBMach + -
188.9189*PAX.
*LBMach;
52 Overland_Range1_1 = 16.93008 + 1.020703*LBR + 59.01307*LBMach + -0.0001736223*LBR.*LBMach;
53 Overwater_Range1_1 = 16.38548 + 1.02077*NLBR + 59.2079*LBMach + -9.759192E-09*NLBR*NLBR + -0.0001586204*NLBR*LBMach;
54 Overland_Fuel_1 = -5262.437 + 14.21307*LBR + 4.119441*NLBR + 397.7017*PAX + 0.003320731*NLBR.*NLBR + 27976.51*LBMach.
*LBMach + 0.00117847*LBR.*NLBR + 0.04035596*LBR.*PAX + 0.1330613*NLBR.*PAX + -21.16781*NLBR.*LBMach + -513.2906*PAX.*LBMach;
55 Overwater_Fuel_1 = 10889.26 + 12.82116*NLBR + 0.006320012*NLBR.*NLBR + 1.178589*PAX.*PAX + 41001.97*LBMach.*LBMach +
0.2797478*NLBR.*PAX + -31.86764*NLBR.*LBMach + -583.3308*PAX.*LBMach;
56 Overland_TOGW_1 = 45889.35 + 10.28922*LBR + 6.577221*NLBR + 869.562*PAX + 0.001067075*LBR.*LBR + 0.004934355*NLBR.*NLBR +
38815.43*LBMach.*LBMach + 0.0009692852*LBR.*NLBR + 0.03788696*LBR.*PAX + 0.1979802*NLBR.*PAX + -28.8521*NLBR.*LBMach + -
667.9332*PAX.
*LBMach;
57 Overwater_TOGW_1 = 56167.97 + 14.84903*NLBR + 559.8684*PAX + 0.00809869*NLBR.*NLBR + 1.387509*PAX.*PAX + 53337.94*LBMach.
*LBMach + 0.3513936*NLBR.*PAX + -40.08902*NLBR.*LBMach + -815.7946*PAX.*LBMach;
58 Overland_Range2_1 = LBR;
59 Overwater_Range2_1 = NLBR;
60 Overland_Time_1 = 2.972073 + 0.002106141*LBR + 4.081442E-05*NLBR + -2.598496*LBMach + -5.094999E-09*NLBR.*NLBR + 0.8094399
*LBMach.*LBMach + -4.620391E-09*LBR.*NLBR + -0.0006050385*LBR.*LBMach + -2.413128E-07*NLBR.*PAX;
61 Overwater_Time_1 = 5.29057 + 0.002039345*NLBR + -0.002831138*PAX + -5.02619*LBMach + -5.270265E-09*NLBR.*NLBR + 2.223648E-
05*PAX.*PAX + 1.483532*LBMach.*LBMach + -1.306243E-06*LBR.*LBMach + -0.0005660546*NLBR.*LBMach;
62 MTOGW_1 = 55668.58 + 14.19788*NLBR + 625.6466*PAX + 0.001404418*LBR.*LBR + 0.00877323*NLBR.*NLBR + 1.449349*PAX.
*PAX + 54012.51*LBMach.*LBMach + -0.001698949*LBR.*NLBR + 0.3481102*NLBR.*PAX + -40.34256*NLBR.*LBMach + -
847.2526*PAX.*LBMach;
63
64 % Define variables for 52 seater per Karl's nomenclature
65
66 vNLBR = NLBR;
67 vLBR = LBR;
68 vPAX = PAX;
69 vNLBMach = NLBMach;
70 vLBMach = LBMach;
71
72 % Populate the values for the 52 seater aircraft (second array element)
73
74 Payload_2 = PAX_weight .* PAX; % weight in pounds
75 Operating_Empty_Weight_2 = 121040.5 + -25.41987*vNLBR + -339.5284*vPAX + 0.003212665*vNLBR.*vNLBR + 1.434517*vPAX.*vPAX +
0.1015964*vNLBR.*vPAX;
76 Zero_Fuel_Weight_2 = 121040.5 + -25.41987*vNLBR + -130.5284*vPAX + 0.003212665*vNLBR.*vNLBR + 1.434517*vPAX.*vPAX + 0.1015964
*vNLBR.*vPAX;
77 Thrust_2 = 112450.7 + -35.56117*vNLBR + -634.0657*vPAX + 0.004490705*vNLBR.*vNLBR + 2.023374*vPAX.*vPAX + 0.142149
*vNLBR.*vPAX;
78 Overland_Range1_2 = 120.2812 + 1.018251*vLBR + 0.0000003825535*vLBR.*vLBR;
79 Overwater_Range1_2 = 118.1901 + 1.023058*vNLBR + -0.0000003565252*vNLBR.*vNLBR;
80 Overland_Fuel_2 = 326111.9 + 27.8378*vLBR + -147.0038*vNLBR + -2208.08*vPAX + 0.01674828*vNLBR.*vNLBR + 4.942013*vPAX.
*vPAX + -0.001250373*vLBR.*vNLBR + 0.4749506*vNLBR.*vPAX;
```

```
81 Overwater_Fuel_2 = 314300.5 + -110.1917*vNLBR + -2143.408*vPAX + 0.01411128*vNLBR.*vNLBR + 4.812647*vPAX.*vPAX + 0.449615
*vNLBR.*vPAX;
82 Overland_TOGW_2 = 603493 + 15.14462*vLBR + -223.3731*vNLBR + -3651.792*vPAX + 0.001559828*vLBR.*vLBR + 0.02491564*vNLBR.
*vNLBR + 12.39115*vPAX.*vPAX + 0.7645333*vNLBR.*vPAX;
83 Overwater_TOGW_2 = 478808.5 + -147.6084*vNLBR + -2804.985*vPAX + 0.01868412*vNLBR.*vNLBR + 9.53697*vPAX.*vPAX + 0.6113751
*vNLBR.*vPAX;
84 Overland_Range2_2 = -0.02173565 + 1.000002*vLBR + 0.004486606*vLBMach.*vLBMach;
85 Overwater_Range2_2 = 0.003711473 + 0.999999*vNLBR;
86 Overland_Time_2 = 0.4570523 + 0.001428535*vLBR + 0.01577379*vPAX + -0.00000005104635*vLBR.*vLBR + -0.00004206952*vPAX.
*vPAX + -0.000001478277*vLBR.*vPAX + -0.000002648109*vNLBR.*vPAX;
87 Overwater_Time_2 = -4.606022 + 0.003727151*vNLBR + 0.04239966*vPAX + -0.0000003226989*vNLBR.*vNLBR + -0.00009439455*vPAX.
*vPAX + -0.000009792375*vNLBR.*vPAX;
88 MTOGW_2 = max(Overland_TOGW_2,Overwater_TOGW_2);
89
90 % Calculate the averages
91
92 Payload_lb = mean(Payload_1,Payload_2); % weight in pounds
93 Operating_Empty_Weight_lb = Weight .*Operating_Empty_Weight_1 + Operating_Empty_Weight_2 .*(1-Weight);
94 Zero_Fuel_Weight_lb = Weight .*Zero_Fuel_Weight_1 + Zero_Fuel_Weight_2 .*(1-Weight);
95 Thrust_lb = Weight .*Thrust_1 + Thrust_2 .*(1-Weight);
96 Overland_Range1_nm = Weight .*Overland_Range1_1 + Overland_Range1_2 .*(1-Weight);
97 Overwater_Range1_nm = Weight .*Overwater_Range1_1 + Overwater_Range1_2 .*(1-Weight);
98 Overland_Fuel_lb = Weight .*Overland_Fuel_1 + Overland_Fuel_2 .*(1-Weight);
99 Overwater_Fuel_lb = Weight .*Overwater_Fuel_1 + Overwater_Fuel_2 .*(1-Weight);
100 Overland_TOGW_lb = Weight .*Overland_TOGW_1 + Overland_TOGW_2 .*(1-Weight);
101 Overwater_TOGW_lb = Weight .*Overwater_TOGW_1 + Overwater_TOGW_2 .*(1-Weight);
102 Overland_Range2_nm = Weight .*Overland_Range2_1 + Overland_Range2_2 .*(1-Weight);
103 Overwater_Range2_nm = Weight .*Overwater_Range2_1 + Overwater_Range2_2 .*(1-Weight);
104 Overland_Time_hrs = Weight .*Overland_Time_1 + Overland_Time_2 .*(1-Weight);
105 Overwater_Time_hrs = Weight .*Overwater_Time_1 + Overwater_Time_2 .*(1-Weight);
106 MTOGW_lb = max(Overland_TOGW_lb ,Overwater_TOGW_lb);
107
108 %
109 % % New section that deviates from NASA's surrogate model
110
111 % % Generate the Block fuel and Block time plots
112 %
113 % % Load the data for Mach Trade LB and Mach Trade Overwater
114 %
115 % % Data contains:
116 % %
117 % % R1 = Range (nm)
118 % % R2 = Block fuel (lb)
119 % % R3 = Block Time (hr)
120 %
121 % load MachTrade_OverwaterData.m
122 % load MachTrade_lowBoomData.m
123 %
124 % % Create new variables
125 %
126 % Reference_RangeLB_overland = MachTrade_lowBoomData(1,:);
127 % Reference_fuelLB_overland = MachTrade_lowBoomData(2,:);
128 % Reference_timeLB_overland = MachTrade_lowBoomData(3,:);
129 %
130 % Reference_RangeLB_overwater = MachTrade_OverwaterData(1,:);
131 % Reference_fuelLB_overwater = MachTrade_OverwaterData(2,:);
132 % Reference_timeLB_overwater = MachTrade_OverwaterData(3,:);
133 %
134 % % Generate new data for new design vehicle fuel (low boom overland)
135 % Call Wu's surrogate model here
136
137 % FuelSurrogate_LB_overland = MTOGW_lb+(Reference_RangeLB_overland-(MaximumRange/2500)*2662.385).*(MTOGW_lb-
(zero_fuel_weight_lb/76256).*(154481-25.812092*2662.385))./(2662.385.*(MaximumRange/2500)) ...
138 % - zero_fuel_weight_lb - 1.2473264*Reference_RangeLB_overland.*(2500./MaximumRange).*(MTOGW_lb/154481) - 5632.276*
(2*zero_fuel_weight_lb/76256-1);
139 % %
```

```
140 % BlockTimeSurrogate_LB_overland = 0.0011025*1.6*Reference_RangeLB_overland./MachNumber+0.817904*MachNumber/1.6;
141 % BlockSpeed_LB_overland = Reference_RangeLB_overland ./ BlockTimeSurrogate_LB_overland;
142 % %
143 % % % Generate new data for new design vehicle fuel (low boom overwater)
144 % %
145 %
146 % FuelSurrogate_LB_overwater = MTOGW_lb+(Reference_RangeLB_overwater-(MaxRange_overwater/3600)*3856.099).*(MTOGW_lb-
(zero_fuel_weight_lb/76256)*(154481-18.8791*3856.099))./(3856.099.*(MaxRange_overwater/3600)) ...
147 % - zero_fuel_weight_lb - 0.8799807*Reference_RangeLB_overwater.*(3600/MaxRange_overwater).*(MTOGW_lb/154481) - 5510.527*
(2*zero_fuel_weight_lb/76256-1);
148 % %
149 % BlockTimeSurrogate_LB_overwater =
0.0009687*1.8*Reference_RangeLB_overwater./Mach_overwater+0.909704*Mach_overwater/1.8;
150 % BlockSpeed_LB_overwater = Reference_RangeLB_overwater ./ BlockTimeSurrogate_LB_overwater;
151 % %
152 %
153 % % Plot the block fuel (lb) vs range (nm)
154 % figure
155 % plot(Reference_RangeLB_overland,FuelSurrogate_LB_overland,'o-r',Reference_RangeLB_overwater ,FuelSurrogate_LB_overwater,'^-b',
...
156 % Reference_RangeLB_overland,Reference_fuelLB_overland,'*-k',Reference_RangeLB_overwater,Reference_fuelLB_overwater,'+-g')
157 % xlabel('Range (nm)','fontsize',24)
158 % ylabel('Block Fuel (lb)','fontsize',24)
159 % legend('Surrogate Low-Boom Overland','Surrogate Low-Boom Overwater','FLOPS Low-Boom Overland', 'FLOPS Low-Boom Overwater')
160 % grid
161 %
162 % % Plot the block time (lb) vs range (nm)
163 % figure
164 % plot(Reference_RangeLB_overland,BlockTimeSurrogate_LB_overland,'o-
r',Reference_RangeLB_overwater,BlockTimeSurrogate_LB_overwater,'^-b', ...
165 % Reference_RangeLB_overland,Reference_timeLB_overland,'*-k',Reference_RangeLB_overwater,Reference_timeLB_overwater,'+-g');
166 % xlabel('Range (nm)','fontsize',24)
167 % ylabel('Block Time (hr)','fontsize',24)
168 % legend('Surrogate Low-Boom Overland','Surrogate Low-Boom Overwater','FLOPS Low-Boom Overland', 'FLOPS Low-Boom Overwater')
169 % grid
170 % %
171 % % Generate parameters for life cycle cost model
172 %
173 % fuelBurn_overland_lb_hr = FuelSurrogate_LB_overland ./ BlockTimeSurrogate_LB_overland;
174 % fuelBurn_overwater_lb_hr = FuelSurrogate_LB_overwater ./ BlockTimeSurrogate_LB_overwater;
175 % %
176 % Reference_fuelBurn_overland_lb_hr = Reference_fuelLB_overland ./ Reference_timeLB_overland;
177 % Reference_fuelBurn_overwater_lb_hr = Reference_fuelLB_overwater ./ Reference_timeLB_overwater;
178 % %
179 % % Plot the fuel burn (lb/hr) vs range (nm)
180 % figure
181 % plot(Reference_RangeLB_overland,fuelBurn_overland_lb_hr,'o-
r',Reference_RangeLB_overwater,Reference_fuelBurn_overwater_lb_hr,'^-b', ...
182 % Reference_RangeLB_overland,Reference_fuelBurn_overland_lb_hr,'*-
k',Reference_RangeLB_overwater,Reference_fuelBurn_overwater_lb_hr,'+-g');
183 % xlabel('Range (nm)','fontsize',24)
184 % ylabel('Fuel Burn (lb/hr)','fontsize',24)
185 % legend('Surrogate Low-Boom Overland','Surrogate Low-Boom Overwater','FLOPS Low-Boom Overland', 'FLOPS Low-Boom Overwater')
186 % grid
187 % %
188
189 end
190
```

```
1 function [adjustedCostPer_Airframe_2020] = LifeCycleCostModel_QproductionOut(Qproduction,MaxSpeed_knots, ...
2 Operating_Empty_Weight_lb, Thrust_lb)
3
4 % Function to estimate the life cycle cost of a low boom aircraft
5 % Method is adapted from L. Nicolai textbook
6 % Aircraft and Airship Design: Volume 1, AIAA 2012
7
```

```
8 % Nicolai's method employs equations originally developed at the
9 % RAND Corporation
10
11 % Script to estimate the development of cost of an aircaft
12 % Using Nicolai's method and modified DAPCA IV equations
13
14 % Global variables
15
16 profitMargin = 1.15; % Assumes 15% profit margin in the aircraft development program
17 yearOfAnalysis = 1998; % Year with cost values in Nicolai's equations
18 engineModel = 2; % 1 = Nicolai, 2 = Our model
19 noEnginesProduction = 2;
20 noEnginesDevelopment = 3;
21 amortizationMultiplier = 2;
22 avionicsInstallationMultiplier = 1.2; % 20% of the avionics cost to install during production
23 avionicsCost = 6e6; % $ in 1998 based on a survey done by Trani using B/C Aviation Avionics Data
24
25 % Input variables:
26
27 % Wempty_lb = Empty weight (lb)
28 % MaxSpeed_knots = Maximum speed in knots at best altitude
29 % Qproduction = Total number of aircraft produced
30
31 Qdevelopment = 7; % Development aircraft
32 yearOfAnalysis = 1998;
33 QC_to_Manufacturing_Hours_Ratio = 0.13; % Quality control to manufacturing hours ratio
34
35 % Aircraft data
36
37 % inletTemperature = 2660; % Degrees Rankine
38 noEngines = 2;
39 noEnginesLCC = 2; % ratio of no engines used in the life cycle
40
41 % Start the aircraft development cost equations
42
43 Qtotal = Qdevelopment + Qproduction;
44
45 % Airframe engineering cost
46
47 % Calculate the engineering hours
48
49 EHours_DTE = 4.86 * Operating_Empty_Weight_lb .^0.777 .* MaxSpeed_knots .^ 0.894 .* Qdevelopment .^ 0.163;
50 EHours_Total = 4.86 * Operating_Empty_Weight_lb .^0.777 .* MaxSpeed_knots .^ 0.894 .* Qtotal.^ 0.163;
51 EHours_Production = EHours_Total - EHours_DTE ;
52
53 % Estimate the hourly rates for all four activities (cost)
54
55 [hourlyRateTooling,hourlyRateEngineering,hourlyRateManufacturing,hourlyRateQC] = calculateHourlyRates(yearOfAnalysis);
56
57 EngineeringDTE_Cost = EHours_DTE * hourlyRateEngineering;
58 EngineeringProduction_Cost = EHours_Total * hourlyRateEngineering;
59 Engineering_DTE_Cost = EHours_Production + EngineeringProduction_Cost;
60
61 % Development support cost
62
63 DevelopmentCost = 66 * Operating_Empty_Weight_lb .^0.63 .* MaxSpeed_knots .^ 1.3; % in 1998 dollars
64
65 % Flight test and operations
66
67 FlightTestCost = 1852 * Operating_Empty_Weight_lb .^0.325 .* MaxSpeed_knots .^ 0.822 .* Qdevelopment .^ 1.21;
68
69 % Tooling cost
70
71 ToolingHours_DTE = 5.99 * Operating_Empty_Weight_lb .^0.777 .* MaxSpeed_knots .^ 0.696 .* Qdevelopment .^ 0.263;
72 ToolingHours_Total = 5.99 * Operating_Empty_Weight_lb .^0.777 .* MaxSpeed_knots .^ 0.696 .* Qtotal .^ 0.263;
73 ToolingHours_Production = ToolingHours_Total - ToolingHours_DTE;
```

```
74
75 ToolingDTE_Cost = ToolingHours_DTE * hourlyRateTooling;
76 ToolingProduction_Cost = ToolingHours_Production * hourlyRateTooling;
77 ToolingTotal_Cost = ToolingHours_Total * hourlyRateTooling;
78
79 % Manufacturing labor cost
80
81 ManLaborHours_DTE = 7.37 * Operating_Empty_Weight_lb .^0.82 .* MaxSpeed_knots .^ 0.484 .* Qdevelopment .^ 0.641;
82 ManLaborHours_Total = 7.37 * Operating_Empty_Weight_lb .^0.82 .* MaxSpeed_knots .^ 0.484 .* Qtotal .^ 0.641;
83 ManLaborHours_Production = ManLaborHours_Total - ManLaborHours_DTE;
84
85 ManLabor_DTE_Cost = ManLaborHours_DTE * hourlyRateManufacturing;
86 ManLabor_Production_Cost = ManLaborHours_Production * hourlyRateManufacturing;
87 ManLabor_Total_Cost = ManLaborHours_Total * hourlyRateManufacturing;
88
89 % Quality control cost
90
91 QualityControl_Hours_DTE = QC_to_Manufacturing_Hours_Ratio * ManLaborHours_DTE ;
92 QualityControl_Hours_Total = QC_to_Manufacturing_Hours_Ratio * ManLaborHours_Total;
93 QualityControl_Hours_Production = QC_to_Manufacturing_Hours_Ratio * ManLaborHours_Production;
94
95 QualityControl_DTE_Cost = QualityControl_Hours_DTE * hourlyRateQC;
96 QualityControl_Production_Cost = QualityControl_Hours_Total * hourlyRateQC;
97 QualityControl_Total_Cost = QualityControl_Hours_Production * hourlyRateQC;
98
99
100 % Manufacturing material cost (in 1998 dollars)
101
102 ManMaterial_DTE_Cost = 16.39 * Operating_Empty_Weight_lb .^0.921 .* MaxSpeed_knots .^ 0.621 .* Qdevelopment .^ 0.799;
103 ManMaterial_Production_Cost = 16.39 * Operating_Empty_Weight_lb .^0.921 .* MaxSpeed_knots .^ 0.621 .* Qproduction .^ 0.799;
104 ManMaterial_Total_Cost = ManMaterial_DTE_Cost + ManMaterial_Production_Cost;
105
106
107 % Engine cost
108
109 % Original equation to find the engine cost using the RAND equations
110 % EngineCost = 2306 * (0.043 .* thrust_surrogate_lb + 243.3 * MaxMach + 0.969 .* inletTemperature -2228);
111
112 if engineModel == 1
113 % Simplified expression in Nicolai
114 EngineCost = 520 * Thrust_lb .^0.8356;
115 elseif engineModel == 2
116 % Our own regression to find the engine cost based on a regression
117 % equation using modern engines (Trani 2020)
118 EngineCost = 1.0428e-08*Thrust_lb.^3 - 0.0028093*Thrust_lb.^2 + 329.55*Thrust_lb - 3.8218e+5; % dollars 1998
119 end
120
121 DTE_EngineCost = EngineCost * noEnginesDevelopment * Qdevelopment;
122 ProductionEngineCost = EngineCost * noEnginesProduction * Qproduction;
123 LCC_EngineCost = ProductionEngineCost + DTE_EngineCost;
124
125 % Avionics cost
126
127 AvionicsCost = avionicsCost * avionicsInstallationMultiplier; % dollars in 1998
128 LCC_AvionicsCost = AvionicsCost * Qtotal;
129 DTE_AvionicsCost = AvionicsCost * Qdevelopment;
130 Production_AvionicsCost = AvionicsCost * Qproduction;
131
132 % Summary
133
134 totalProgramCost = LCC_AvionicsCost + LCC_EngineCost + ManMaterial_Total_Cost + ...
135 QualityControl_Total_Cost + ManLabor_Total_Cost + ToolingTotal_Cost + FlightTestCost + ...
136 DevelopmentCost + Engineering_DTE_Cost;
137
138 totalDTECost = DTE_AvionicsCost + DTE_EngineCost + ManMaterial_DTE_Cost + ...
139 QualityControl_DTE_Cost + ManLabor_DTE_Cost + ToolingDTE_Cost + FlightTestCost + ...
```

```
DevelopmentCost + Engineering_DTE_Cost;

UnitCost_DTE = totalDTECost ./ (Qproduction/amortizationMultiplier);

totalProductionCost = Production_AvionicsCost + ProductionEngineCost + ManMaterial_Production_Cost + ...
QualityControl_Production_Cost + ManLabor_Production_Cost + ToolingProduction_Cost + ...
EngineeringProduction_Cost;

UnitCost_Production = totalProductionCost/ Qproduction;

costPerAirframe_1998 = UnitCost_DTE + UnitCost_Production;

% Use CPI index to convert 1998 costs into 2020 as needed

yearVector = [1998 2000 2010 2020 ];
CPI_index = [1.000 1.044 1.341 1.596];

cpi_value = interp1(yearVector,CPI_index,2020);

% costPerAirframe_1998 = totalProgramCost ./ Qtotal;
costPerAirframe_wProfit = costPerAirframe_1998 * profitMargin; % profit margin
adjustedCostPer_Airframe_2020 = costPerAirframe_wProfit * cpi_value;


% disp(['Engineering Hours ', num2str(EHours_Total) , 'hours'])
% disp(['Total Avionics Cost ', num2str(LCC_AvionicsCost) , '($)'])
% disp(['LCC_EngineCost ', num2str(LCC_EngineCost) , '($)'])
% disp(['Manufacturing Materials Cost ', num2str(ManMaterial_Total_Cost) , '($)'])
% disp(['Quality Control Cost ', num2str(QualityControl_Total_Cost) , '($)'])
% disp(['Man Labor Cost ', num2str(ManLabor_Total_Cost) , '($)'])
% disp(['Tooling Costs ', num2str(ToolingTotal_Cost) , '($)'])
% disp(['Flight Test Cost ', num2str(FlightTestCost) , '($)'])
% disp(['Development Cost ', num2str(DevelopmentCost) , '($)'])
% disp(['Engineering DTE Cost ', num2str(Engineering_DTE_Cost) , '($)'])

% Export the results

% costResults_1998 = [LCC_AvionicsCost ; LCC_EngineCost ; ManMaterial_Total_Cost ; ...
% QualityControl_Total_Cost ; ManLabor_Total_Cost ; ToolingTotal_Cost ; FlightTestCost ; ...
% DevelopmentCost ; Engineering_DTE_Cost]; % $ in 1998
%
% costResults_2020_millions = costResults_1998 * cpi_value / 1e6; % millions

% costLabels = {'Total Avionics Cost' ; 'Total Engine Cost' ; 'Manufacturing Materials Cost' ; ...
% 'Quality Control Cost'; 'Man Labor Cost' ; 'Tooling Costs' ; 'Flight Test Cost' ; 'Development Cost ' ; ...
% 'Engineering DTE Cost '};

end


function [hourlyRateTooling,hourlyRateEngineering,hourlyRateManufacturing,hourlyRateQC] = calculateHourlyRates(year)

% Calculation of hourly rate based on year of manufacturing
% Method accordiong to L. Nicolai
% US Department of Labor
% Hourly rates in $1998 dollars

hourlyRateTooling = 2.883*year-5666;
hourlyRateEngineering = 2.576 * year -5058;
hourlyRateManufacturing = 2.316 * year - 4552;
hourlyRateQC = 2.60 * year - 5112;

end


% Function to calculate LCC cost for a low-boom aircraft
```

```
2
3 % Inputs:
4
5 % Aircraft_Purchase_Price
6 % Flight_Hours_per_Year
7 % Fuel_Scaling_Parameter
8 % distance_lb_MLand
9 % distance_lb_MOverwater
10 % aveSpeed_lb_MLand
11 % aveSpeed_lb_MOverwater
12 % fuelBurn_MLand
13 % fuelBurn_MOverwater
14
15 % Outputs:
16
17 % Adjusted_Cost_per_Passenger_Mile ($)
18 % Cost_Per_Mile ($)
19
20 % function [Adjusted_Cost_per_Passenger_Mile,Cost_Per_Mile, Mission_Stage_Lenght_nm] = LowBoom_LCC_Model_v1(aircraftCost, ...
21 % Flight_Hours_per_Year, Maintenance_Hours_per_Flight_Hour, aveSpeed_lb_MLand, ...
22 % aveSpeed_lb_MOverwater, fuelBurn_MLand, fuelBurn_MOverwater, OperationalProfile, Thrust_lb, ...
23 % Surrogate_RangeLB_overland, Surrogate_RangeLB_overwater, Load_Factor, PAX, Fuel_Scaling_Parameter_OL,
Fuel_Scaling_Parameter_OW)
24
25
26 function [Adjusted_Cost_per_Passenger_Mile,Cost_Per_Mile, Mission_Stage_Lenght_nm] = LowBoom_LCC_Model_v1(aircraftCost, ...
27 Flight_Hours_per_Year, Maintenance_Hours_per_Flight_Hour, blockSpeed_OL, ...
28 blockSpeed_OW, fuelBurn_overland_lb_hr, fuelBurn_overwater_lb_hr, OperationalProfile, Thrust_lb, ...
29 distanceOL, distanceOW, Load_Factor, PAX, Fuel_Scaling_Parameter_OL, Fuel_Scaling_Parameter_OW)
30
31 % Define parameters
32
33 % Fuel_Scaling_Parameter = 1.0;
34 % Cruise_Mach_Number = 1.7;
35 % Mission_Stage_Lenght_nm = 2500;
36 Life_Cycle_Time = 15;
37 timeStep = 1; % Time step for LCC calculations (years)
38 InterestRate = 7.5; % interest rate in percent per year
39 Percent_Repositioning_Flights = 10; % percent
40 conv_lb2gal = 6.7; % converst to lbs from gallons
41
42 % Initialize the costs
43
44 Cumulative_Amortization_Cost = 0;
45 Cumulative_Costs = 0;
46 Cumulative_Fixed_Costs = 0;
47 Cumulative_Hangar_and_Office_Expenses = 0;
48 Cumulative_Periodic_Costs = 0;
49 Cumulative_Personnel_Cost = 0;
50
51 % Initialize the variables
52
53 Initial_Pilot_Training = 20000;
54 Crew_per_Flight = 2;
55 % Flight_Hours_per_Year = 3500;
56 Number_of_Pilots = ceil(Flight_Hours_per_Year/1000)*Crew_per_Flight;
57 Initial_Maintenance_Training = 15000;
58 Total_Initial_Training_Cost = Initial_Pilot_Training*Number_of_Pilots+Initial_Maintenance_Training;
59
60 % Calculate initial training cost
61 Cumulative_Training_Cost = Total_Initial_Training_Cost;
62 Cumulative_Variable_Cost = 0;
63
64 Total_Hours_Flown = 0;
65 Interest_Rate_perMonth = InterestRate/12/100; % Fraction per month
66 Payments = Life_Cycle_Time*12;
```

```
67 Percent_Resale_Value = .15;
68 Resale_Value = aircraftCost*Percent_Resale_Value;
69 Loan_Amount = aircraftCost-Resale_Value;
70
71 Monthly_Payment = Loan_Amount * Interest_Rate_perMonth * (1+Interest_Rate_perMonth) .^ Payments / (((1+ Interest_Rate_perMonth)
.^ Payments)
-1);
72
73 Annual_Amortization_Cost = Monthly_Payment*12;
74 Hull_Insurance = 6e5;
75 Liability_Insurance = 4e5;
76 Maintenance_Software_Programs = 12000;
77 Miscellaneous_Service = 20000;
78 Property_Tax = 130000;
79 Annual_Fixed_Costs = Hull_Insurance+Liability_Insurance+Maintenance_Software_Programs+Miscellaneous_Service+Property_Tax;
80 Hangar_and_Office_Lease_Space = 230000;
81 Miscellaneous_Office_Expense = 20000;
82 Annual_Hangar_and_Office_Expenses = Hangar_and_Office_Lease_Space+Miscellaneous_Office_Expense;
83
84 % Engine maintenance costs
85 % Engine_Overhaul_Cost = 2.43e6;
86
87 % Equation witrh scaling factor after 25,000 lbs of thrust
88 % Engine_Overhaul_Cost = 0.0001774 * thrust_surrogate_lb .^2 + 32.32 * thrust_surrogate_lb + 1.4399e6; % new equation developed on
July
20, 2020
89
90 EngineReserveCost_hr = 0.05337 * Thrust_lb + 307.1; % Hourly engine reserve to pay the overhaul
91 noEngines = 2;
92 Overhaul_Interval = 5000; % hours
93
94 % Annual_Engine_Overhaul_Cost = (Engine_Overhaul_Cost*noEngines)*Flight_Hours_per_Year/Overhaul_Interval;
95 Annual_Engine_Overhaul_Cost = EngineReserveCost_hr*Flight_Hours_per_Year;
96
97 % This section is old set of equations
98 % MidLife_INterval = 2500;
99 % MidLife_Hot_Section_Inspection = Engine_Overhaul_Cost/6;
100 % Annual_Midlife_Cost = Flight_Hours_per_Year/MidLife_INterval*(MidLife_Hot_Section_Inspection*noEngines);
101
102 Modernisation_and_Upgrades = 250000;
103 Modernisation_Time_INterval = 5000;
104 Annual_Modernisation_Costs = Modernisation_and_Upgrades*Flight_Hours_per_Year/Modernisation_Time_INterval;
105 Aircraft_Paint = 150000;
106 Paint_and_Refurbishing_Interval = 4000;
107 Annual_Paint_Cost = Aircraft_Paint*Flight_Hours_per_Year/Paint_and_Refurbishing_Interval;
108 Interior_Refurbishing = 2e5;
109 Annual_Refurbishing_Cost = Interior_Refurbishing*Flight_Hours_per_Year/Paint_and_Refurbishing_Interval;
110 % Annual_Periodic_Costs =
Annual_Engine_Overhaul_Cost+Annual_Midlife_Cost+Annual_Modernisation_Costs+Annual_Paint_Cost+Annual_Refurbishing_Cost;
111 Annual_Periodic_Costs = Annual_Engine_Overhaul_Cost+Annual_Modernisation_Costs+Annual_Paint_Cost+Annual_Refurbishing_Cost;
112 Recurrent_Maintenance_Training = 10000;
113 Recurrent_Pilot_Training = 15000;
114 Annual_Traning_Cost = Recurrent_Maintenance_Training+Recurrent_Pilot_Training*Number_of_Pilots;
115
116
117 % Calculate fuel consumption for two speed regimes
118 % This is a new function call to estimate the fuel burn as a function of
119 % distance flown
120
121 if OperationalProfile == 1 % Over land profile
122 Fuel_Consumption_lb_per_hr = fuelBurn_overland_lb_hr;
123 Mission_Stage_Lenght_nm = distanceOL;
124 Fuel_Consumption_Gal_per_Hour = Fuel_Scaling_Parameter_OL*Fuel_Consumption_lb_per_hr/conv_lb2gal ;
125 elseif OperationalProfile == 2 % Over water profile
126 Fuel_Consumption_lb_per_hr = fuelBurn_overwater_lb_hr;
127 Mission_Stage_Lenght_nm = distanceOW;
```

```
128 Fuel_Consumption_Gal_per_Hour = Fuel_Scaling_Parameter_OW*Fuel_Consumption_lb_per_hr/conv_lb2gal ;
129 end
130
131 Jet_Fuel_Cost_per_Gallon = 2.00;
132 Fuel_Expense_per_Hour = Fuel_Consumption_Gal_per_Hour*Jet_Fuel_Cost_per_Gallon;
133
134 Maintenance_Labor_Expense_per_Hour = 160;
135 Miscellaneous_Trip_Expenses = 140;
136 Schedule_Parts_Expense = 300;
137 Total_Variable_Costs_per_Hour =
Fuel_Expense_per_Hour+Maintenance_Hours_per_Flight_Hour*Maintenance_Labor_Expense_per_Hour+Miscellaneous_Trip_Expenses+Sched
ule_Parts_Exp
ense;
138 Annual_Variable_Cost = Flight_Hours_per_Year.*Total_Variable_Costs_per_Hour;
139 Annual_Pilot_Salary = 180000;
140 Percent_Salaries_to_Benefits = .25;
141 Personnel_Benefits = Annual_Pilot_Salary*Percent_Salaries_to_Benefits*Number_of_Pilots;
142 Pilots_Salaries = Annual_Pilot_Salary*Number_of_Pilots;
143 Cabin_Crew_Salary = 85000;
144 Cabin_Crew_per_Flight = 2;
145 Total_Cabin_Crew_per_Acft = ceil(Flight_Hours_per_Year/1000)*Cabin_Crew_per_Flight;
146 Total_Cabin_Crew_Salary = Cabin_Crew_Salary*Total_Cabin_Crew_per_Acft*(1+Percent_Salaries_to_Benefits/100);
147 Annual_Personnel_Costs = Personnel_Benefits+Pilots_Salaries+Total_Cabin_Crew_Salary;
148 Annual_Costs_of_Operation =
Annual_Amortization_Cost+Annual_Fixed_Costs+Annual_Hangar_and_Office_Expenses+Annual_Periodic_Costs+Annual_Traning_Cost+Annual
_Variable_C
ost+Annual_Personnel_Costs;
149 Annual_Hours = Flight_Hours_per_Year;
150 Profit_Margin = 10;
151 Deadhead_Hours = Flight_Hours_per_Year*Percent_Repositioning_Flights/100;
152 Revenue_Hours_per_Year = Flight_Hours_per_Year-Deadhead_Hours;
153
154 % Calculate the average block speed for a given stage length
155
156 if OperationalProfile == 1 % Over land profile
157 Average_Aircraft_Speed_knots = blockSpeed_OL;
158 elseif OperationalProfile == 2 % Over water profile
159 Average_Aircraft_Speed_knots = blockSpeed_OW;
160 end
161
162 % Insert the calculation of cumulative hours and costs
163
164 Cumulative_Amortization_Cost = 0;
165 Cumulative_Costs = 0;
166 Cumulative_Fixed_Costs = 0;
167 Cumulative_Hangar_and_Office_Expenses = 0;
168 Cumulative_Periodic_Costs = 0;
169 Cumulative_Personnel_Cost = 0;
170 Cumulative_Training_Cost = 0;
171 Cumulative_Variable_Cost = 0;
172 Total_Hours_Flown = 0;
173 Time = 0;
174
175 % Calculate the cumulative costs - use a simple loop
176
177 for i=1:timeStep:Life_Cycle_Time
178
179 Cumulative_Amortization_Cost = Cumulative_Amortization_Cost + Annual_Amortization_Cost * timeStep;
180 Cumulative_Costs = Cumulative_Costs + Annual_Costs_of_Operation * timeStep;
181 Cumulative_Fixed_Costs = Cumulative_Fixed_Costs + Annual_Fixed_Costs * timeStep;
182 Cumulative_Hangar_and_Office_Expenses = Cumulative_Hangar_and_Office_Expenses + Annual_Hangar_and_Office_Expenses * timeStep;
183 Cumulative_Periodic_Costs = Cumulative_Periodic_Costs + Annual_Periodic_Costs * timeStep;
184 Cumulative_Personnel_Cost = Cumulative_Personnel_Cost + Annual_Personnel_Costs * timeStep;
185 Cumulative_Training_Cost = Cumulative_Training_Cost + Annual_Traning_Cost * timeStep;
186 Cumulative_Variable_Cost = Cumulative_Variable_Cost + Annual_Variable_Cost * timeStep;
187 Total_Hours_Flown = Total_Hours_Flown + Annual_Hours * timeStep;
```

```
188 Time = Time + timeStep;
189 end
190
191 % Calculate the vehicle operational cost metrics
192
193 Revenue_Hours_LC = Total_Hours_Flown * Revenue_Hours_per_Year / Flight_Hours_per_Year;
194 Total_Cost_Per_Hour = Cumulative_Costs / (Revenue_Hours_LC);
195 Revenue_Required_Per_Hour = (1+Profit_Margin/100)*Total_Cost_Per_Hour;
196
197 Cost_Per_Mile = Revenue_Required_Per_Hour./Average_Aircraft_Speed_knots; % $/nm
198 Cost_per_Passenger_Mile = Cost_Per_Mile./(Load_Factor*PAX); % $/nm
199 Indirect_Cost_Multiplier = 1.1;
200 Adjusted_Cost_per_Passenger_Mile = Cost_per_Passenger_Mile.*Indirect_Cost_Multiplier; % $/nm
201 Cost_per_Available_Seat_mile = Cost_Per_Mile./PAX;
202 Cost_per_Trip = Mission_Stage_Lenght_nm./Average_Aircraft_Speed_knots.*Total_Cost_Per_Hour;
203 Total_Annual_Person_Trips = Cost_per_Passenger_Mile;
204 Flight_Time_hrs = Mission_Stage_Lenght_nm./Average_Aircraft_Speed_knots;
205 Revenue_Trips_per_Year = Revenue_Hours_per_Year./Flight_Time_hrs;
206 Passengers_per_Aircraft_per_Year = Revenue_Trips_per_Year.*Load_Factor*PAX;
207 Fleet_Size_for_Revenue_Trips = Total_Annual_Person_Trips./Passengers_per_Aircraft_per_Year;
208 Non_Revenue_Trips = Deadhead_Hours./Flight_Time_hrs;
209 Total_Fleet_for_All_Trips = Fleet_Size_for_Revenue_Trips.*(1+Percent_Repositioning_Flights./100);
210
211 end
212
213
1 function plot_CostPerSeatMile(Mission_Stage_Lenght_nm,Adjusted_Cost_per_Passenger_Mile)
2 % Function to just plot the aircraft development cost
3
4 % plot(Qproduction,adjustedCostPer_Airframe_2020,'--ok')
5 % xlabel('Production Run')
6 % ylabel('Cost per Airframe ($2020)')
7 % grid
8
9 % Create figure
10 figure1 = figure('Color',[1 1 1]);
11
12 % Create axes
13 axes1 = axes('Parent',figure1);
14 hold(axes1,'on');
15
16 % Create plot
17 plot(Mission_Stage_Lenght_nm,Adjusted_Cost_per_Passenger_Mile,'MarkerSize',10,'Marker','o','LineWidth',2.5,'LineStyle','--',...
18 'Color',[0 0 0]);
19
20 % Create labels
21 xlabel('Distance (nm)');
22 ylabel('Cost per Passenger Mile ($/nm)');
23
24 box(axes1,'on');
25 grid(axes1,'on');
26 hold(axes1,'off');
27 % Set the remaining axes properties
28 set(axes1,'FontSize',24);
29
30 end
31
32
1 function[] =
Fare_per_Mile_CDF_Adjustment_Main_Function_US(SST_Market_Analysis_Dir,Load_Pre_Process_Dir,mach_overland,mach_overwater,
vot_limit,US_VOT,date,aircraft_range_overland,aircraft_range_overwater)
2 %% This script generates the fare per mile CDF data from ARC 2016 - US Market
3
4 local_disc = '';
5 main_delimiter = '\';
6 Input_Folder_Dir = ([local_disc,'..\..\..\..\..\SST_2020_Input']);
```

```
7 SST_US_CDF_Dir = ([SST_Market_Analysis_Dir,main_delimiter,'US_Market\CDF']);
8 save_dir = ([SST_US_CDF_Dir,main_delimiter,'Output']);
9 Upload_Dir = ([Load_Pre_Process_Dir,'\VOT\US_Market\Output']);
10 TravelTime_Upload_Dir = ([Load_Pre_Process_Dir,main_delimiter,'SST_Travel_Times\Output']);
11 Upload_PreProc_CDF_Dir = ([Load_Pre_Process_Dir,main_delimiter,'CDF\US_Market\Output']);
12
13 %Create Output directory
14 if exist((save_dir),'dir') == 0
15
16 mkdir(([SST_US_CDF_Dir,main_delimiter,'Output']))
17
18 end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0
19
20 addpath(([SST_US_CDF_Dir,main_delimiter,'Output']))
21
22 Fare_per_mile_paid_by_OD_with_VOT_Adjustment_US(Upload_Dir, Upload_PreProc_CDF_Dir, Input_Folder_Dir,TravelTime_Upload_Dir,
main_delimiter,
save_dir, mach_overland, mach_overwater, vot_limit,US_VOT,date,aircraft_range_overland,aircraft_range_overwater)
23
24 return;
25
1
2 function [] = Fare_per_mile_paid_by_OD_with_VOT_Adjustment_US(Upload_Dir, Upload_PreProc_CDF_Dir,
Input_Folder_Dir,TravelTime_Upload_Dir,
main_delimiter, save_dir, mach_overland, mach_overwater, vot_limit,US_VOT,date,aircraft_range_overland,aircraft_range_overwater)
3
4
load(([TravelTime_Upload_Dir,main_delimiter,'Travel_Times_with_Ground_Times_OverlandRange_',num2str(aircraft_range_overland),'_nm_1
_',num2str
((mach_overland-1)*10),'M_1_',num2str((mach_overwater-1)*10),'M.mat']),'Travel_Times')
5
6 load(([Input_Folder_Dir,main_delimiter,'OD_Pairs_Year_2016.mat']),'OD_Pairs_Year_2016')
7
8 load(([Upload_Dir,main_delimiter,'Value_of_Time_US.mat']),'Value_of_Time')
9
10
load(([Upload_PreProc_CDF_Dir,main_delimiter,'OD_Fare_Per_Mile_Paid_Premium_Original_US.mat']),'OD_Fare_Per_Mile_Paid_Premium')
11
12 OAG_Data = OD_Pairs_Year_2016;
13 clear OD_Pairs_Year_2016
14 %%
15
16 if US_VOT ~= 0
17
18 Value_of_Time.Value(:) = US_VOT;
19
20 end
21
22 %%
23 %vot median value
24 index = Value_of_Time.Value <= vot_limit;
25 vot_median = round(prctile(Value_of_Time.Value(index),50),0);
26
27 %%
28 OAG_dep_arr = strcat(OAG_Data.DepAirport,'_',OAG_Data.ArrAirport);
29 TT_dep_arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);
30
31 [~,matched_index] = ismember(TT_dep_arr,OAG_dep_arr);
32
33 Travel_Times.SubSonic_Travel_Time = OAG_Data.OD_AverageElapsedTime_hrs(matched_index);
34
35 Travel_Times.Travel_Time_Saving = Travel_Times.SubSonic_Travel_Time - (Travel_Times.Travel_Time);
36
37 negative_saving = find(Travel_Times.Travel_Time_Saving < 0);
38
39 if isempty(negative_saving) == 0
```

```
40
41 Travel_Times.DepAirport(negative_saving) = [];
42 Travel_Times.ArrAirport(negative_saving) = [];
43 Travel_Times.Distance(negative_saving) = [];
44 Travel_Times.Travel_Time(negative_saving) = [];
45 Travel_Times.SubSonic_Travel_Time(negative_saving) = [];
46 Travel_Times.Travel_Time_Saving(negative_saving) = [];
47 end
48
49 TT_dep_arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);
50
51 length_of_OD = length(OD_Fare_Per_Mile_Paid_Premium);
52
53 for OD = 1:length_of_OD
54
55
56 % if mod(OD, 100) == 0
57 % disp([num2str(OD), '/', num2str(length_of_OD)]);
58 %
59 % end
60
61 [TT_OD_index,~] = ismember(TT_dep_arr,OD_Fare_Per_Mile_Paid_Premium(OD).OD);
62
63 if sum(TT_OD_index) ~= 0
64
65 od_char = char(OD_Fare_Per_Mile_Paid_Premium(OD).OD);
66 origin_airport = cellstr(od_char(1:3));
67 [vot_od_index,~] = ismember(Value_of_Time.Airport,origin_airport);
68 vot_od_count = 1;
69
70 if sum(vot_od_index) ~= 0
71 vot_od_count = Value_of_Time.Number_of_records(vot_od_index);
72
73 if Value_of_Time.Value(vot_od_index) > vot_limit
74
75 Value_of_Time_od = vot_limit;
76 Final_Value_of_Time_By_OD.Origin_Median_Used_Yes1_No_0(OD,1) = 0;
77 Final_Value_of_Time_By_OD.Origin_Limit_Used_Yes1_No_0(OD,1) = 1;
78
79 else
80
81 Value_of_Time_od = round(Value_of_Time.Value(vot_od_index),0);
82 Final_Value_of_Time_By_OD.Origin_Median_Used_Yes1_No_0(OD,1) = 0;
83 Final_Value_of_Time_By_OD.Origin_Limit_Used_Yes1_No_0(OD,1) = 0;
84 end
85
86 else
87
88 Value_of_Time_od = vot_median;
89 Final_Value_of_Time_By_OD.Origin_Median_Used_Yes1_No_0(OD,1) = 1;
90 Final_Value_of_Time_By_OD.Origin_Limit_Used_Yes1_No_0(OD,1) = 0;
91 end
92 %%
93
94 destination_airport = cellstr(od_char(5:7));
95
96 [vot_do_index,~] = ismember(Value_of_Time.Airport,destination_airport);
97 vot_do_count = 1;
98
99 if sum(vot_do_index) ~= 0
100 vot_do_count = Value_of_Time.Number_of_records(vot_do_index);
101 if Value_of_Time.Value(vot_do_index) > vot_limit
102
103 Value_of_Time_do = vot_limit;
104 Final_Value_of_Time_By_OD.Destination_Median_Used_Yes1_No_0(OD,1) = 0;
105 Final_Value_of_Time_By_OD.Destination_Limit_Used_Yes1_No_0(OD,1) = 1;
```

```
106 else
107
108 Value_of_Time_do = round(Value_of_Time.Value(vot_do_index),0);
109 Final_Value_of_Time_By_OD.Destination_Median_Used_Yes1_No_0(OD,1) = 0;
110 Final_Value_of_Time_By_OD.Destination_Limit_Used_Yes1_No_0(OD,1) = 0;
111 end
112
113 else
114
115 Value_of_Time_do = vot_median;
116 Final_Value_of_Time_By_OD.Destination_Median_Used_Yes1_No_0(OD,1) = 1;
117 Final_Value_of_Time_By_OD.Destination_Limit_Used_Yes1_No_0(OD,1) = 0;
118 end
119
120
121 VOT_value = round((Value_of_Time_od * vot_od_count + Value_of_Time_do * vot_do_count) / sum([vot_od_count;vot_do_count]),0);
122
123 Final_Value_of_Time_By_OD.Origin_Airport(OD,1)= origin_airport;
124 Final_Value_of_Time_By_OD.Origin_VOT(OD,1) = Value_of_Time_od;
125 Final_Value_of_Time_By_OD.Origin_Records(OD,1) = vot_od_count;
126
127 Final_Value_of_Time_By_OD.Destination_Airport(OD,1)= destination_airport;
128 Final_Value_of_Time_By_OD.Destination_VOT(OD,1) = Value_of_Time_do;
129 Final_Value_of_Time_By_OD.Destination_Records(OD,1) = vot_do_count;
130 Final_Value_of_Time_By_OD.Weighted_Avg_VOT(OD,1) = VOT_value;
131 Final_Value_of_Time_By_OD.Weighted_Avg_VOT_Records(OD,1) = vot_od_count + vot_do_count;
132
133 %%
134
135 additional_fare_per_mile = round((VOT_value * Travel_Times.Travel_Time_Saving(TT_OD_index)) ./
Travel_Times.Distance(TT_OD_index),2);
136
137 OD_Fare_Per_Mile_Paid_Premium(OD).Unique_Fare = OD_Fare_Per_Mile_Paid_Premium(OD).Unique_Fare + additional_fare_per_mile;
138
139 end
140
141 end
142
143 save(([save_dir,main_delimiter,'OD_Fare_Per_Mile_Paid_Premium_Adjusted_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_US',date,'.mat']),'OD_Fare_Per_Mile_Paid_Premium')
144
save(([save_dir,main_delimiter,'Final_Value_of_Time_By_OD_OverlandRange_',num2str(aircraft_range_overland),'_nm_OverwaterRange_',nu
m2str
(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-
1)*10),'_US',
date,'.mat']),'Final_Value_of_Time_By_OD')
145
146 return;
147
148
1 function [] =
SST_US_Forecast_Main_Function(SST_Market_Analysis_Dir,Load_Pre_Process_Dir,Airport_List_SST_Compatibility,airport_list,mach_overland,
mach_overwater,number_of_seats,FSF,hours_per_year,acft_pax_load_factor,SST_Year_Start,SST_Year_End,vot_limit,aircraft_range_overland,
aircraft_range_overwater,min_distance_statute_mile,max_distance_statute_mile,min_demand,market_share,US_VOT,date,
run_airport_gate_compatibility_section, run_airport_runway_compatibility_section,runway_length_min_ft,airport_runway_length_data)
2 %SST Market Script
3
4 local_disc = '';
5 main_delimiter = '\';
6 Input_Folder_Dir = ([local_disc,'..\..\..\..\..\..\SST_2020_Input']);
7 SST_US_Forecast_Dir = ([SST_Market_Analysis_Dir,main_delimiter,'US_Market\SST_Forecast']);
8 save_dir = ([SST_US_Forecast_Dir,main_delimiter,'Output']);
9 Upload_VOT_Dir = ([Load_Pre_Process_Dir,main_delimiter,'VOT\US_Market\Output']);
10 Upload_PreProc_CDF_Dir = ([Load_Pre_Process_Dir,main_delimiter,'CDF\US_Market\Output']);
11 Upload_CDF_Dir = ([SST_Market_Analysis_Dir,main_delimiter,'US_Market\CDF\Output']);
```

```
12
13 TravelTime_Upload_Dir = ([Load_Pre_Process_Dir,main_delimiter,'SST_Travel_Times\Output']);
14
15 %Create Output directory
16 if exist((save_dir),'dir') == 0
17
18 mkdir(([SST_US_Forecast_Dir,main_delimiter,'Output']))
19
20 end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0
21
22 addpath(([SST_US_Forecast_Dir,main_delimiter,'Output']))
23
24 %%
25
26 load(([Input_Folder_Dir,main_delimiter,'ICAO_Trip_Distribution_All.mat']),'Trip_Distribution_All')
27
28 load(([Input_Folder_Dir,main_delimiter,'OD_Pairs_Year_2016.mat']),'OD_Pairs_Year_2016')
29
30
load(([TravelTime_Upload_Dir,main_delimiter,'Travel_Times_with_Ground_Times_OverlandRange_',num2str(aircraft_range_overland),'_nm_1
_',num2str
((mach_overland-1)*10),'M_1_',num2str((mach_overwater-1)*10),'M.mat']),'Travel_Times')
31
32 load(([Upload_CDF_Dir,main_delimiter,'OD_Fare_Per_Mile_Paid_Premium_Adjusted_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_US',date,'.mat']),'OD_Fare_Per_Mile_Paid_Premium')
33
34 load(([Upload_PreProc_CDF_Dir,main_delimiter,'Fare_Per_Mile_Paid_Premium_US.mat']),'Fare_per_Mile_Paid_Premium')
35
36 load(([Upload_VOT_Dir,main_delimiter,'Value_of_Time_US.mat']),'Value_of_Time')
37
38
load(([Input_Folder_Dir,main_delimiter,'Fare_data\',num2str(number_of_seats),'_Seats\Fare_Low_Boom_US',date,'.mat']),'cost_per_passenge
r_mile')
39 %%
40
41 if US_VOT ~= 0
42
43 Value_of_Time.Value(:) = US_VOT;
44
45 end
46
47 %% Additional Parameters (calculated)
48 minimum_premium_seats = number_of_seats * 260; %1 flight per day, 260 days/year
49
50 index = Value_of_Time.Value <= vot_limit;
51 vot_median = round(prctile(Value_of_Time.Value(index),50),0);
52
53 %% FSF
54
55 fpm_reduction = 1-(100-FSF)/10 * 0.0346;
56 cost_per_passenger_mile.Over_land.fare = round(cost_per_passenger_mile.Over_land.fare .* fpm_reduction,3);
57 cost_per_passenger_mile.Over_water.fare = round(cost_per_passenger_mile.Over_water.fare .* fpm_reduction,3);
58
59
60 [OD_seats_percent_Premium,OD_Pairs_Year_2016,Trip_Distribution_All,generic_cdf,Fare_per_mile_cost] =
Fare_Percent_by_OD_Premium_US
(Fare_per_Mile_Paid_Premium,
OD_Fare_Per_Mile_Paid_Premium,OD_Pairs_Year_2016,Travel_Times,Trip_Distribution_All,SST_Year_Start,SST_Year_End,
cost_per_passenger_mile,vot_median,aircraft_range_overland,aircraft_range_overwater,min_distance_statute_mile);
61
62
63 %%
64 % disp('Analyzing: SST_Market')
65
66 SST_Market_Premium = SST_US_Market_Premium(Trip_Distribution_All, OD_Pairs_Year_2016,OD_seats_percent_Premium,generic_cdf,
```

```
Fare_per_mile_cost,SST_Year_Start,SST_Year_End,min_distance_statute_mile,max_distance_statute_mile,min_demand);
67
68 %%
69 % disp('Analyzing: SST_Market_Share_Premium')
70
71 [SST_Market_Share_Premium] =
SST_US_Market_Share_Premium(SST_Market_Premium,SST_Year_Start,SST_Year_End,market_share,save_dir,
main_delimiter,aircraft_range_overwater,mach_overland,mach_overwater,FSF,number_of_seats);
72
73 %%
74 % disp('Analyzing: Final_SST_US_Market')
75 Final_SST_US_Market(SST_Market_Share_Premium,Airport_List_SST_Compatibility,airport_list,minimum_premium_seats,hours_per_year,
number_of_seats,acft_pax_load_factor,SST_Year_Start,SST_Year_End,FSF,Travel_Times,aircraft_range_overwater,save_dir,main_delimiter,ma
ch_overland,
mach_overwater,date,run_airport_gate_compatibility_section,
run_airport_runway_compatibility_section,runway_length_min_ft,airport_runway_length_data,
aircraft_range_overland);
76
77 return;

1
2 function [OD_seats_percent_Premium,OD_Pairs_Year_2016,Trip_Distribution_All,generic_cdf,Fare_per_mile_cost] =
Fare_Percent_by_OD_Premium_US
(Fare_per_Mile_Paid_Premium,
OD_Fare_Per_Mile_Paid_Premium,OD_Pairs_Year_2016,Travel_Times,Trip_Distribution_All,SST_Year_Start,SST_Year_End,
cost_per_passenger_mile, vot_median,aircraft_range_overland,aircraft_range_overwater,min_distance_statute_mile)
3
4 delete_short_distance_od = find(OD_Pairs_Year_2016.DistStMiles < min_distance_statute_mile);
5
6 OD_Pairs_Year_2016.Carrier_Name(delete_short_distance_od) = [];
7 OD_Pairs_Year_2016.DepAirport_UniqueID(delete_short_distance_od) = [];
8 OD_Pairs_Year_2016.DepAirport(delete_short_distance_od) = [];
9 OD_Pairs_Year_2016.DepIATACtry(delete_short_distance_od) = [];
10 OD_Pairs_Year_2016.DepReg(delete_short_distance_od) = [];
11 OD_Pairs_Year_2016.ArrAirport_UniqueID(delete_short_distance_od) = [];
12 OD_Pairs_Year_2016.ArrAirport(delete_short_distance_od) = [];
13 OD_Pairs_Year_2016.ArrIATACtry(delete_short_distance_od) = [];
14 OD_Pairs_Year_2016.ArrReg(delete_short_distance_od) = [];
15 OD_Pairs_Year_2016.InternationalDomestic(delete_short_distance_od) = [];
16 OD_Pairs_Year_2016.Total_Seats(delete_short_distance_od) = [];
17 OD_Pairs_Year_2016.Total_FstSeats(delete_short_distance_od) = [];
18 OD_Pairs_Year_2016.Total_BusSeats(delete_short_distance_od) = [];
19 OD_Pairs_Year_2016.Total_EcoSeats(delete_short_distance_od) = [];
20 OD_Pairs_Year_2016.Total_Frequency(delete_short_distance_od) = [];
21 OD_Pairs_Year_2016.OD_AverageElapsedTime_hrs(delete_short_distance_od) = [];
22 OD_Pairs_Year_2016.DistStMiles(delete_short_distance_od) = [];
23 OD_Pairs_Year_2016.AcftData(delete_short_distance_od) = [];
24
25 for year = SST_Year_Start:SST_Year_End
26
27 Trip_Distribution_All.(['Year_',num2str(year)]).DepAirport_UniqueID(delete_short_distance_od) = [];
28 Trip_Distribution_All.(['Year_',num2str(year)]).ArrAirport_UniqueID(delete_short_distance_od) = [];
29 Trip_Distribution_All.(['Year_',num2str(year)]).Seats(delete_short_distance_od) = [];
30 end
31 %%
32 length_of_data = length(OD_Pairs_Year_2016.DepAirport);
33
34 Dep_Arr_All = strcat(OD_Pairs_Year_2016.DepAirport,'_',OD_Pairs_Year_2016.ArrAirport);
35
36 length_of_OD_fare = length(OD_Fare_Per_Mile_Paid_Premium);
37 Fare_Dep_Arr_All = cell(length_of_OD_fare,1);
38
39 for fare_od = 1:length_of_OD_fare
40
41 Fare_Dep_Arr_All(fare_od) = OD_Fare_Per_Mile_Paid_Premium(fare_od).OD;
42 end
```

```
43
44 Travel_Times_Dep_Arr_All = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);
45
46 OD_seats_percent_Premium = zeros(length_of_data,1);
47 Fare_per_mile_cost = zeros(length_of_data,1);
48 generic_cdf = zeros(length_of_data,1);
49
50 for OD = 1:length_of_data
51
52 clear aircraft_range
53
54 [od_index,~] = ismember(Fare_Dep_Arr_All,Dep_Arr_All(OD));
55
56
57 od_distance = round((OD_Pairs_Year_2016.DistStMiles(OD)) * 0.868976,0); %st to nm conversion
58
59 [travel_time_od_index,~] = ismember(Travel_Times_Dep_Arr_All,Dep_Arr_All(OD));
60
61 if sum(travel_time_od_index) ~= 0
62
63 if Travel_Times.Overland_Percent(travel_time_od_index,1) > 0.25
64
65 aircraft_range = aircraft_range_overland;
66
67 else
68
69 aircraft_range = aircraft_range_overwater;
70 end
71
72 if od_distance > aircraft_range
73
74 while od_distance > aircraft_range
75
76 od_distance = ceil(od_distance/2);
77 end
78 end
79
80 %%
81
82 if Travel_Times.Overland_Percent(travel_time_od_index) > 0.25
83
84 %overland
85
86 if od_distance >= max(cost_per_passenger_mile.Over_land.distance)
87
88 Fare_per_Mile_Paid_Threshold_Premium = min(cost_per_passenger_mile.Over_land.fare);
89
90 elseif od_distance <= min(cost_per_passenger_mile.Over_land.distance)
91
92 Fare_per_Mile_Paid_Threshold_Premium = max(cost_per_passenger_mile.Over_land.fare);
93
94 else
95
96 %interpolate bewteen values
97
98 difference_in_value = od_distance - cost_per_passenger_mile.Over_land.distance;
99
100 lower_bound_index = max(find(difference_in_value>0));
101
102 upper_bound_index = lower_bound_index + 1;
103
104 Fare_per_Mile_Paid_Threshold_Premium = cost_per_passenger_mile.Over_land.fare(lower_bound_index) - ((cost_per_passenger_mile.
Over_land.distance(lower_bound_index) - od_distance) / (cost_per_passenger_mile.Over_land.distance(lower_bound_index) -
cost_per_passenger_mile.
Over_land.distance(upper_bound_index))) * (cost_per_passenger_mile.Over_land.fare(lower_bound_index) -
cost_per_passenger_mile.Over_land.fare
```

```
(upper_bound_index));
105
106 end
107
108 else
109
110 %overwater
111
112
113 if od_distance >= max(cost_per_passenger_mile.Over_water.distance)
114
115 Fare_per_Mile_Paid_Threshold_Premium = min(cost_per_passenger_mile.Over_water.fare);
116
117 elseif od_distance <= min(cost_per_passenger_mile.Over_water.distance)
118
119 Fare_per_Mile_Paid_Threshold_Premium = max(cost_per_passenger_mile.Over_water.fare);
120
121 else
122
123 %interpolate bewteen values
124
125 difference_in_value = od_distance - cost_per_passenger_mile.Over_water.distance;
126
127 lower_bound_index = max(find(difference_in_value>0));
128
129 upper_bound_index = lower_bound_index + 1;
130
131 Fare_per_Mile_Paid_Threshold_Premium = cost_per_passenger_mile.Over_water.fare(lower_bound_index) - ((cost_per_passenger_mile.
Over_water.distance(lower_bound_index) - od_distance) / (cost_per_passenger_mile.Over_water.distance(lower_bound_index) -
cost_per_passenger_mile.
Over_water.distance(upper_bound_index))) * (cost_per_passenger_mile.Over_water.fare(lower_bound_index) -
cost_per_passenger_mile.Over_water.fare
(upper_bound_index));
132
133 end
134
135 end
136
137 Fare_per_Mile_Paid_Threshold_Premium = Fare_per_Mile_Paid_Threshold_Premium * 0.868976; %convert $/nm. to $/sm.
138
139 if sum(od_index) ~= 0
140
141
142 if min(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare) > Fare_per_Mile_Paid_Threshold_Premium
143
144 OD_seats_percent_Premium(OD) = 1.0;
145
146 else
147
148 if max(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare) >= Fare_per_Mile_Paid_Threshold_Premium
149
150 if sum(ismember(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare,Fare_per_Mile_Paid_Threshold_Premium)) == 1
151
152 [fare_index,~] = ismember(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare,Fare_per_Mile_Paid_Threshold_Premium);
153
154 OD_seats_percent_Premium(OD) = 1 - OD_Fare_Per_Mile_Paid_Premium(od_index).Cummulative_Count(fare_index);
155
156 else
157
158 %interpolate bewteen values
159
160 difference_in_value = Fare_per_Mile_Paid_Threshold_Premium - OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare;
161
162 lower_bound_index = max(find(difference_in_value>0));
163
164 upper_bound_index = lower_bound_index + 1;
```

```
165
166 OD_seats_percent_Premium(OD) = 1 - (OD_Fare_Per_Mile_Paid_Premium(od_index).Cummulative_Count(lower_bound_index) -
((OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare(lower_bound_index) - Fare_per_Mile_Paid_Threshold_Premium) ./
(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare(lower_bound_index) - OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare
(upper_bound_index))) .* (OD_Fare_Per_Mile_Paid_Premium(od_index).Cummulative_Count(lower_bound_index) -
OD_Fare_Per_Mile_Paid_Premium
(od_index).Cummulative_Count(upper_bound_index)));
167 end
168
169 else
170
171 OD_seats_percent_Premium(OD) = 0;
172
173 end
174
175 end
176
177
178 else
179
180 generic_cdf(OD,1) = 1;
181
182
183 [od_index_2,~] = ismember(Travel_Times_Dep_Arr_All,Dep_Arr_All(OD));
184
185 if sum(od_index_2) ~= 0
186
187 travel_time_saving = OD_Pairs_Year_2016.OD_AverageElapsedTime_hrs(OD) - Travel_Times.Travel_Time(od_index_2);
188
189 if travel_time_saving > 0
190
191
192 additional_fare = (vot_median * travel_time_saving) ./ Travel_Times.Distance(od_index_2);
193
194 Fare_per_Mile_Paid_Premium_adjusted.Fare = round((Fare_per_Mile_Paid_Premium.Fare + additional_fare),2);
195 Fare_per_Mile_Paid_Premium_adjusted.Cumulative = Fare_per_Mile_Paid_Premium.Cumulative;
196 %
197
198 %%
199 if min(Fare_per_Mile_Paid_Premium_adjusted.Fare) > Fare_per_Mile_Paid_Threshold_Premium
200
201 OD_seats_percent_Premium(OD) = 1.0;
202
203 else
204
205 if max(Fare_per_Mile_Paid_Premium_adjusted.Fare) >= Fare_per_Mile_Paid_Threshold_Premium
206
207 if sum(ismember(Fare_per_Mile_Paid_Premium_adjusted.Fare,Fare_per_Mile_Paid_Threshold_Premium)) == 1
208
209 [fare_index,~] = ismember(Fare_per_Mile_Paid_Premium_adjusted.Fare,Fare_per_Mile_Paid_Threshold_Premium);
210
211 OD_seats_percent_Premium(OD) = 1 - Fare_per_Mile_Paid_Premium_adjusted.Cumulative(fare_index);
212
213 else
214
215 %interpolate bewteen values
216
217 difference_in_value = Fare_per_Mile_Paid_Threshold_Premium - Fare_per_Mile_Paid_Premium_adjusted.Fare;
218
219 lower_bound_index = max(find(difference_in_value>0));
220
221 upper_bound_index = lower_bound_index + 1;
222
223 OD_seats_percent_Premium(OD) = 1 - (Fare_per_Mile_Paid_Premium_adjusted.Cumulative(lower_bound_index) -
((Fare_per_Mile_Paid_Premium_adjusted.Fare(lower_bound_index) - Fare_per_Mile_Paid_Threshold_Premium) ./
(Fare_per_Mile_Paid_Premium_adjusted.
```

```
Fare(lower_bound_index) - Fare_per_Mile_Paid_Premium_adjusted.Fare(upper_bound_index))) .*
(Fare_per_Mile_Paid_Premium_adjusted.Cumulative
(lower_bound_index) - Fare_per_Mile_Paid_Premium_adjusted.Cumulative(upper_bound_index)));
224 end
225
226 else
227
228 OD_seats_percent_Premium(OD) = 0;
229
230 end
231
232 end
233 %%
234
235 else
236
237 [index_Premium,~] = ismember(Fare_per_Mile_Paid_Premium.Fare,Fare_per_Mile_Paid_Threshold_Premium);
238
239 if sum(index_Premium) ~= 0
240
241 Premium_seats_percent = 1 - Fare_per_Mile_Paid_Premium.Cumulative(index_Premium);
242 else
243 difference_in_value = Fare_per_Mile_Paid_Threshold_Premium - Fare_per_Mile_Paid_Premium.Fare;
244
245 lower_bound_index = max(find(difference_in_value>0));
246
247 upper_bound_index = lower_bound_index + 1;
248
249 Premium_seats_percent = 1 - (Fare_per_Mile_Paid_Premium.Cumulative(lower_bound_index) - ((Fare_per_Mile_Paid_Premium.Fare
(lower_bound_index) - Fare_per_Mile_Paid_Threshold_Premium) ./ (Fare_per_Mile_Paid_Premium.Fare(lower_bound_index) -
Fare_per_Mile_Paid_Premium.
Fare(upper_bound_index))) .* (Fare_per_Mile_Paid_Premium.Cumulative(lower_bound_index) - Fare_per_Mile_Paid_Premium.Cumulative
(upper_bound_index)));
250 end
251
252 OD_seats_percent_Premium(OD) = Premium_seats_percent;
253
254
255 end
256
257 else
258
259 [index_Premium,~] = ismember(Fare_per_Mile_Paid_Premium.Fare,Fare_per_Mile_Paid_Threshold_Premium);
260
261 if sum(index_Premium) ~= 0
262
263 Premium_seats_percent = 1 - Fare_per_Mile_Paid_Premium.Cumulative(index_Premium);
264 else
265 difference_in_value = Fare_per_Mile_Paid_Threshold_Premium - Fare_per_Mile_Paid_Premium.Fare;
266
267 lower_bound_index = max(find(difference_in_value>0));
268
269 upper_bound_index = lower_bound_index + 1;
270
271 Premium_seats_percent = 1 - (Fare_per_Mile_Paid_Premium.Cumulative(lower_bound_index) - ((Fare_per_Mile_Paid_Premium.Fare
(lower_bound_index) - Fare_per_Mile_Paid_Threshold_Premium) ./ (Fare_per_Mile_Paid_Premium.Fare(lower_bound_index) -
Fare_per_Mile_Paid_Premium.
Fare(upper_bound_index))) .* (Fare_per_Mile_Paid_Premium.Cumulative(lower_bound_index) - Fare_per_Mile_Paid_Premium.Cumulative
(upper_bound_index)));
272 end
273
274 OD_seats_percent_Premium(OD) = Premium_seats_percent;
275
276 end
277
278 end
```

```
279
280 Fare_per_mile_cost(OD,1) = Fare_per_Mile_Paid_Threshold_Premium;
281
282
283 end
284 end
285
286
287 return;
288
```

```
1 function SST_Market_Premium = SST_US_Market_Premium(Trip_Distribution_All,
OD_Pairs_Year_2016,OD_seats_percent_Premium,generic_cdf,
Fare_per_mile_cost,SST_Year_Start,SST_Year_End,min_distance_statute_mile,max_distance_statute_mile,min_demand)
2
3 OAG2016 = OD_Pairs_Year_2016;
4 OAG2016.Premium_Seats = OAG2016.Total_FstSeats + OAG2016.Total_BusSeats;
5 OAG2016.OD_seats_percent_Premium = OD_seats_percent_Premium;
6 OAG2016.Fare_per_mile_cost = Fare_per_mile_cost;
7
8 clear OD_Pairs_Year_2016
9
10 All_OD_Pair_Data = strcat(OAG2016.DepAirport,'_',OAG2016.ArrAirport);
11
12 Dep = char(Trip_Distribution_All.Year_2017.DepAirport_UniqueID);
13 Dep = Dep(:,1:3);
14 Dep = cellstr(Dep);
15
16 Arr = char(Trip_Distribution_All.Year_2017.ArrAirport_UniqueID);
17 Arr = Arr(:,1:3);
18 Arr = cellstr(Arr);
19
20 All_Trip_Distribution = strcat(Dep,'_',Arr);
21
22 [~ , OD_Pair_Index] = ismember(All_OD_Pair_Data,All_Trip_Distribution);
23
24 delete_record = OD_Pair_Index == 0;
25
26 OAG2016.Carrier_Name(delete_record) = [];
27 OAG2016.DepAirport_UniqueID(delete_record) = [];
28 OAG2016.DepAirport(delete_record) = [];
29 OAG2016.DepIATACtry(delete_record) = [];
30 OAG2016.DepReg(delete_record) = [];
31 OAG2016.ArrAirport_UniqueID(delete_record) = [];
32 OAG2016.ArrAirport(delete_record) = [];
33 OAG2016.ArrIATACtry(delete_record) = [];
34 OAG2016.ArrReg(delete_record) = [];
35 OAG2016.InternationalDomestic(delete_record) = [];
36 OAG2016.Total_Seats(delete_record) = [];
37 OAG2016.Premium_Seats(delete_record) = [];
38 OAG2016.Total_FstSeats(delete_record) = [];
39 OAG2016.Total_BusSeats(delete_record) = [];
40 OAG2016.Total_EcoSeats(delete_record) = [];
41 OAG2016.Total_Frequency(delete_record) = [];
42 OAG2016.OD_AverageElapsedTime_hrs(delete_record) = [];
43 OAG2016.DistStMiles(delete_record) = [];
44 OAG2016.AcftData(delete_record) = [];
45 OAG2016.OD_seats_percent_Premium(delete_record) = [];
46 OAG2016.Fare_per_mile_cost(delete_record) = [];
47 OAG2016.Percent = OAG2016.Premium_Seats ./ OAG2016.Total_Seats;
48
49 %%
50
51 for year = SST_Year_Start:SST_Year_End
52
53 SST.(['Year_',num2str(year)]).DepAirport_UniqueID = OAG2016.DepAirport_UniqueID;
```

```
SST.(['Year_',num2str(year)]).DepAirport = OAG2016.DepAirport;
SST.(['Year_',num2str(year)]).DepIATACtry = OAG2016.DepIATACtry;
SST.(['Year_',num2str(year)]).DepReg = OAG2016.DepReg;
SST.(['Year_',num2str(year)]).ArrAirport_UniqueID = OAG2016.ArrAirport_UniqueID;
SST.(['Year_',num2str(year)]).ArrAirport = OAG2016.ArrAirport;
SST.(['Year_',num2str(year)]).ArrIATACtry = OAG2016.ArrIATACtry;
SST.(['Year_',num2str(year)]).ArrReg = OAG2016.ArrReg;
SST.(['Year_',num2str(year)]).InternationalDomestic = OAG2016.InternationalDomestic;

SST.(['Year_',num2str(year)]).Total_Seats = Trip_Distribution_All.(['Year_',num2str(year)]).Seats;
SST.(['Year_',num2str(year)]).Premium_Seats = ceil(Trip_Distribution_All.(['Year_',num2str(year)]).Seats .* OAG2016.Percent);

SST.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs = OAG2016.OD_AverageElapsedTime_hrs;
SST.(['Year_',num2str(year)]).DistStMiles = OAG2016.DistStMiles;
SST.(['Year_',num2str(year)]).OD_premium_seats_percent = OAG2016.OD_seats_percent_Premium;
SST.(['Year_',num2str(year)]).Fare_per_mile_cost = OAG2016.Fare_per_mile_cost;
SST.(['Year_',num2str(year)]).generic_cdf = generic_cdf;

end

%%

SST_Market_Premium = SST;

for year = SST_Year_Start:SST_Year_End

max_distance_index = find(SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles > max_distance_statute_mile);

if isempty(max_distance_index) == 0

SST_Market_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepAirport(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepIATACtry(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepReg(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrIATACtry(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrReg(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).InternationalDomestic(max_distance_index) = [];

SST_Market_Premium.(['Year_',num2str(year)]).Total_Seats(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).Premium_Seats(max_distance_index) = [];

SST_Market_Premium.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).generic_cdf(max_distance_index) = [];

end

clear max_distance_index

min_distance_index = find(SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles < min_distance_statute_mile);

if isempty(min_distance_index) == 0

SST_Market_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepAirport(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepIATACtry(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepReg(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrIATACtry(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrReg(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).InternationalDomestic(min_distance_index) = [];
```

```
120
121 SST_Market_Premium.(['Year_',num2str(year)]).Total_Seats(min_distance_index) = [];
122 SST_Market_Premium.(['Year_',num2str(year)]).Premium_Seats(min_distance_index) = [];
123
124 SST_Market_Premium.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(min_distance_index) = [];
125 SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles(min_distance_index) = [];
126 SST_Market_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent(min_distance_index) = [];
127 SST_Market_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost(min_distance_index) = [];
128 SST_Market_Premium.(['Year_',num2str(year)]).generic_cdf(min_distance_index) = [];
129
130 end
131
132 clear min_distance_index
133
134 min_demnad_index = find(SST_Market_Premium.(['Year_',num2str(year)]).Total_Seats < min_demand);
135
136 if isempty(min_demnad_index) == 0
137
138 SST_Market_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID(min_demnad_index) = [];
139 SST_Market_Premium.(['Year_',num2str(year)]).DepAirport(min_demnad_index) = [];
140 SST_Market_Premium.(['Year_',num2str(year)]).DepIATACtry(min_demnad_index) = [];
141 SST_Market_Premium.(['Year_',num2str(year)]).DepReg(min_demnad_index) = [];
142 SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID(min_demnad_index) = [];
143 SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport(min_demnad_index) = [];
144 SST_Market_Premium.(['Year_',num2str(year)]).ArrIATACtry(min_demnad_index) = [];
145 SST_Market_Premium.(['Year_',num2str(year)]).ArrReg(min_demnad_index) = [];
146 SST_Market_Premium.(['Year_',num2str(year)]).InternationalDomestic(min_demnad_index) = [];
147
148 SST_Market_Premium.(['Year_',num2str(year)]).Total_Seats(min_demnad_index) = [];
149 SST_Market_Premium.(['Year_',num2str(year)]).Premium_Seats(min_demnad_index) = [];
150
151 SST_Market_Premium.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(min_demnad_index) = [];
152 SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles(min_demnad_index) = [];
153 SST_Market_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent(min_demnad_index) = [];
154 SST_Market_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost(min_demnad_index) = [];
155 SST_Market_Premium.(['Year_',num2str(year)]).generic_cdf(min_demnad_index) = [];
156
157 end
158
159 clear min_demnad_index
160 end
161
162
163 return;
164
1
2 function [] =
Final_SST_US_Market(SST_Market_Share_Premium,Airport_List_SST_Compatibility,airport_list,minimum_premium_seats,hours_per_year,
number_of_seats,acft_pax_load_factor,SST_Year_Start,SST_Year_End,FSF,Travel_Times,aircraft_range_overwater,save_dir,main_delimiter,ma
ch_overland,
mach_overwater,date,run_airport_gate_compatibility_section,
run_airport_runway_compatibility_section,runway_length_min_ft,airport_runway_length_data,
aircraft_range_overland)
3
4 acft_seat_capacity = number_of_seats;
5
6 for year = SST_Year_Start:SST_Year_End
7
8 US_US =
strcat(SST_Market_Share_Premium.(['Year_',num2str(year)]).DepIATACtry,'_',SST_Market_Share_Premium.(['Year_',num2str(year)]).
ArrIATACtry);
9
10 non_us_index = ismember(US_US,'US_US') == 0;
11
12 SST_Market_Share_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID(non_us_index) = [];
13 SST_Market_Share_Premium.(['Year_',num2str(year)]).DepAirport(non_us_index) = [];
```

```
14 SST_Market_Share_Premium.(['Year_',num2str(year)]).DepIATACtry(non_us_index) = [];
15 SST_Market_Share_Premium.(['Year_',num2str(year)]).DepReg(non_us_index) = [];
16 SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID(non_us_index) = [];
17 SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrAirport(non_us_index) = [];
18 SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrIATACtry(non_us_index) = [];
19 SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrReg(non_us_index) = [];
20 SST_Market_Share_Premium.(['Year_',num2str(year)]).InternationalDomestic(non_us_index) = [];
21 SST_Market_Share_Premium.(['Year_',num2str(year)]).Total_Seats(non_us_index) = [];
22 SST_Market_Share_Premium.(['Year_',num2str(year)]).Premium_Seats(non_us_index) = [];
23 SST_Market_Share_Premium.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(non_us_index) = [];
24 SST_Market_Share_Premium.(['Year_',num2str(year)]).DistStMiles(non_us_index) = [];
25 SST_Market_Share_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent(non_us_index) = [];
26 SST_Market_Share_Premium.(['Year_',num2str(year)]).SST_Premium_Seats(non_us_index) = [];
27 SST_Market_Share_Premium.(['Year_',num2str(year)]).generic_cdf(non_us_index) = [];
28 SST_Market_Share_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost(non_us_index) = [];
29 end
30
31 %%
32
33 for year = SST_Year_Start:SST_Year_End
34
35 SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID =
SST_Market_Share_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID;
36 SST_Final_Market.(['Year_',num2str(year)]).DepAirport = SST_Market_Share_Premium.(['Year_',num2str(year)]).DepAirport;
37 SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry = SST_Market_Share_Premium.(['Year_',num2str(year)]).DepIATACtry;
38 SST_Final_Market.(['Year_',num2str(year)]).DepReg = SST_Market_Share_Premium.(['Year_',num2str(year)]).DepReg;
39
40 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID =
SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID;
41 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport = SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrAirport;
42 SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry = SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrIATACtry;
43 SST_Final_Market.(['Year_',num2str(year)]).ArrReg = SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrReg;
44
45 SST_Final_Market.(['Year_',num2str(year)]).Total_Seats = SST_Market_Share_Premium.(['Year_',num2str(year)]).Total_Seats;
46
47 SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats = SST_Market_Share_Premium.(['Year_',num2str(year)]).Premium_Seats;
48
49 SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs = SST_Market_Share_Premium.(['Year_',num2str(year)]).
OD_AverageElapsedTime_hrs;
50 SST_Final_Market.(['Year_',num2str(year)]).DistStMiles = SST_Market_Share_Premium.(['Year_',num2str(year)]).DistStMiles;
51
52 SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats =
SST_Market_Share_Premium.(['Year_',num2str(year)]).SST_Premium_Seats;
53
54 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats = SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats;
55 SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost =
SST_Market_Share_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost;
56 SST_Final_Market.(['Year_',num2str(year)]).generic_cdf = SST_Market_Share_Premium.(['Year_',num2str(year)]).generic_cdf;
57
58 end
59
60
61 for year = SST_Year_Start:SST_Year_End
62
63
64 od_with_min_demand_index = find(SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats >= minimum_premium_seats);
65
66 o_d =
strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(od_with_min_demand_index),'_',SST_Final_Market.(['Year_',num2str(year)]).
ArrAirport(od_with_min_demand_index));
67 d_o =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(od_with_min_demand_index),'_',SST_Final_Market.(['Year_',num2str(year)]).
DepAirport(od_with_min_demand_index));
68 od_do = [o_d;d_o];
69 od_do_unique = unique(od_do);
70
```

```
71 SST_Final_Market_OD =
strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);
72
73 delete_index = ismember(SST_Final_Market_OD,od_do_unique) == 0;
74
75
76 SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID(delete_index) = [];
77 SST_Final_Market.(['Year_',num2str(year)]).DepAirport(delete_index) = [];
78 SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry(delete_index) = [];
79 SST_Final_Market.(['Year_',num2str(year)]).DepReg(delete_index) = [];
80 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID(delete_index) = [];
81 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(delete_index) = [];
82 SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry(delete_index) = [];
83 SST_Final_Market.(['Year_',num2str(year)]).ArrReg(delete_index) = [];
84 SST_Final_Market.(['Year_',num2str(year)]).Total_Seats(delete_index) = [];
85 SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats(delete_index) = [];
86
87 SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(delete_index) = [];
88 SST_Final_Market.(['Year_',num2str(year)]).DistStMiles(delete_index) = [];
89 SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats(delete_index) = [];
90
91 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(delete_index) = [];
92 SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost (delete_index) = [];
93 SST_Final_Market.(['Year_',num2str(year)]).generic_cdf(delete_index) = [];
94
95 end
96
97 %% Airport Gate Compatibility
98
99 if run_airport_gate_compatibility_section == 1
100
101 for year = SST_Year_Start:SST_Year_End
102 clear od_dist
103
104 airport_sst_compatibility_index = ismember(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,Airport_List_SST_Compatibility)==0;
105
106 sst_od = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);
107
108 while sum(airport_sst_compatibility_index) ~= 0
109
110 airport_sst_compatibility_index = find(airport_sst_compatibility_index);
111
112 no_comp_do =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)),'_',SST_Final_Market.(['Year_',
num2str(year)]).DepAirport(airport_sst_compatibility_index(1)));
113 [no_comp_do_index,~] = ismember(sst_od,no_comp_do);
114 length_of_od = length(SST_Final_Market.(['Year_',num2str(year)]).DepAirport);
115
116 [o_match_index] =
ismember(airport_list.Airport_IDs,SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)));
117 lat_o = airport_list.Apt_Lat(o_match_index);
118 lon_o = airport_list.Apt_Lon(o_match_index);
119
120 od_dist = zeros(length_of_od,1);
121 for airport = 1:length_of_od
122
123 [d_match_index] = ismember(airport_list.Airport_IDs,SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport));
124
125 lat_d = airport_list.Apt_Lat(d_match_index);
126 lon_d = airport_list.Apt_Lon(d_match_index);
127
128 od_dist(airport,1) = ceil(deg2sm(distance(lat_o,lon_o,lat_d,lon_d)));
129 end
130
131 zero_dist = od_dist == 0;
132
```

```
od_dist(zero_dist) = 999999;

min_dist = min(od_dist);

if min_dist <= 30

redirect_airport_index = find(od_dist==min_dist);

od_redirect_verification =
strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)),'_',SST_Final_Market.(['Year_',
num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)));
do_redirect_verification =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)),'_',SST_Final_Market.
(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)));


od_redirect_exist = ismember(sst_od,od_redirect_verification);
do_redirect_exist = ismember(sst_od,do_redirect_verification);

if sum(od_redirect_exist) == 0
% disp(year)
% disp('change')
% disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)))
% disp('to')
% disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)))
%
SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)) = SST_Final_Market.(['Year_',num2str(year)]).
DepAirport(redirect_airport_index(1));
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(no_comp_do_index) = SST_Final_Market.(['Year_',num2str(year)]).DepAirport
(redirect_airport_index(1));

else
% disp('add from')
% disp(strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]),'_',
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)])))
% disp('to')
%
disp(strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(od_redirect_exist);find(do_redirect_exist)]),'_',SST_Final_Market.
(['Year_',num2str(year)]).ArrAirport([find(od_redirect_exist);find(do_redirect_exist)])))
%
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(od_redirect_exist) =
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats
(od_redirect_exist) + SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(airport_sst_compatibility_index(1));
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(do_redirect_exist) =
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats
(do_redirect_exist) + SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(no_comp_do_index);


SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs([find(no_comp_do_index);airport_sst_compatibility_index(1)]) =
[];
SST_Final_Market.(['Year_',num2str(year)]).DistStMiles([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).generic_cdf([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];

```

```
185 end
186
187
188 else
189
190 % disp(year)
191 % disp('delete')
192
193 % disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]))
194 %
195 % disp(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]))
196 %
197 SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
198 SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
199 SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
200 SST_Final_Market.(['Year_',num2str(year)]).DepReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
201 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
202 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
203 SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
204 SST_Final_Market.(['Year_',num2str(year)]).ArrReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
205 SST_Final_Market.(['Year_',num2str(year)]).Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
206 SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
207 SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs([find(no_comp_do_index);airport_sst_compatibility_index(1)]) =
[];
208 SST_Final_Market.(['Year_',num2str(year)]).DistStMiles([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
209 SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
210 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
211 SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
212 SST_Final_Market.(['Year_',num2str(year)]).generic_cdf([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
213
214
215 end
216 airport_sst_compatibility_index = ismember(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,Airport_List_SST_Compatibility)==0;
217
218 sst_od = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);
219
220 end
221
222 end
223 end
224
225 %% Runway Length Compatibility
226 if run_airport_runway_compatibility_section == 1
227
228 runway_length_index = airport_runway_length_data.runway_ft >= runway_length_min_ft;
229 Airport_List_SST_Compatibility = airport_runway_length_data.airport(runway_length_index);
230
231 for year = SST_Year_Start:SST_Year_End
232 clear od_dist
233
234 airport_sst_compatibility_index = ismember(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,Airport_List_SST_Compatibility)==0;
235
236 sst_od = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);
237
238 while sum(airport_sst_compatibility_index) ~= 0
239
240 airport_sst_compatibility_index = find(airport_sst_compatibility_index);
241
242 no_comp_do =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)),'_',SST_Final_Market.(['Year_',
num2str(year)]).DepAirport(airport_sst_compatibility_index(1)));
243 [no_comp_do_index,~] = ismember(sst_od,no_comp_do);
244 length_of_od = length(SST_Final_Market.(['Year_',num2str(year)]).DepAirport);
245
246 [o_match_index] =
ismember(airport_list.Airport_IDs,SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)));
```

```
247 lat_o = airport_list.Apt_Lat(o_match_index);
248 lon_o = airport_list.Apt_Lon(o_match_index);
249
250 od_dist = zeros(length_of_od,1);
251 for airport = 1:length_of_od
252
253 [d_match_index] = ismember(airport_list.Airport_IDs,SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport));
254
255 lat_d = airport_list.Apt_Lat(d_match_index);
256 lon_d = airport_list.Apt_Lon(d_match_index);
257
258 od_dist(airport,1) = ceil(deg2sm(distance(lat_o,lon_o,lat_d,lon_d)));
259 end
260
261 zero_dist = od_dist == 0;
262
263 od_dist(zero_dist) = 999999;
264
265 min_dist = min(od_dist);
266
267 if min_dist <= 30
268
269 redirect_airport_index = find(od_dist==min_dist);
270
271 od_redirect_verification =
strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)),'_',SST_Final_Market.(['Year_',
num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)));
272 do_redirect_verification =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)),'_',SST_Final_Market.
(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)));
273
274
275 od_redirect_exist = ismember(sst_od,od_redirect_verification);
276 do_redirect_exist = ismember(sst_od,do_redirect_verification);
277
278 if sum(od_redirect_exist) == 0
279 % disp(year)
280 % disp('line149')
281 % disp('change')
282 % disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)))
283 % disp('to')
284 % disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)))
285 %
286 SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)) = SST_Final_Market.(['Year_',num2str(year)]).
DepAirport(redirect_airport_index(1));
287 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(no_comp_do_index) = SST_Final_Market.(['Year_',num2str(year)]).DepAirport
(redirect_airport_index(1));
288
289 else
290
291 % disp('add from')
292 % disp(strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]),'_',
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)])))
293 % disp('to')
294 %
disp(strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(od_redirect_exist);find(do_redirect_exist)]),'_',SST_Final_Market.
(['Year_',num2str(year)]).ArrAirport([find(od_redirect_exist);find(do_redirect_exist)])))
295
296 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(od_redirect_exist) =
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats
(od_redirect_exist) + SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(airport_sst_compatibility_index(1));
297 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(do_redirect_exist) =
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats
(do_redirect_exist) + SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(no_comp_do_index);
298
299
```

```
SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs([find(no_comp_do_index);airport_sst_compatibility_index(1)]) =
[];
SST_Final_Market.(['Year_',num2str(year)]).DistStMiles([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).generic_cdf([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];

end


else

% disp(year)
%
% disp('delete')
%
% disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]))
%
% disp(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]))

SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs([find(no_comp_do_index);airport_sst_compatibility_index(1)]) =
[];
SST_Final_Market.(['Year_',num2str(year)]).DistStMiles([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).generic_cdf([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];


end
airport_sst_compatibility_index = ismember(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,Airport_List_SST_Compatibility)==0;

sst_od = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);

end

end
end
%%

TT_Dep_Arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);

for year = SST_Year_Start:SST_Year_End

number_of_OD = length(SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID);
```

```
364
365 SST_Dep_Arr = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);
366
367 for OD = 1:number_of_OD
368
369
370 [od_travel_time_index,~] = ismember(TT_Dep_Arr,SST_Dep_Arr(OD));
371
372 if isempty(Travel_Times.Travel_Time(od_travel_time_index)) == 0
373
374 seats = SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(OD);
375
376 flights = ceil(seats ./ acft_seat_capacity);
377
378 total_flight_hours = flights .* Travel_Times.Travel_Time(od_travel_time_index);
379
380 SST_Final_Market.(['Year_',num2str(year)]).Aircraft_Needed(OD,1) = (total_flight_hours ./ hours_per_year);
381
382 clear seats flights total_flight_hours
383
384 else
385 SST_Final_Market.(['Year_',num2str(year)]).Aircraft_Needed(OD,1) = 0;
386
387 clear seats flights total_flight_hours
388 end
389
390 end
391
392 SST_Final_Market.(['Year_',num2str(year)]).Number_of_Passenger = SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats .*
acft_pax_load_factor;
393
394 end
395
396 save(([save_dir,main_delimiter,'SST_Final_Market_OverlandRange_',num2str(aircraft_range_overland),'_nm_OverwaterRange_',num2str
(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-
1)*10),'_US',
date,'.mat']),'SST_Final_Market')
397
398 return;
399

1 function [] =
Fare_per_Mile_CDF_Adjustment_Main_Function_US_Int(SST_Market_Analysis_Dir,Load_Pre_Process_Dir,mach_overland,mach_overwater,
vot_limit,US_Int_VOT,date,aircraft_range_overland,aircraft_range_overwater)
2 %% This script generates the fare per mile CDF data from ARC 2016 - US Market
3
4
5 local_disc = '';
6 main_delimiter = '\';
7 Input_Folder_Dir = ([local_disc,'..\..\..\..\..\SST_2020_Input']);
8 SST_US_Int_CDF_Dir = ([SST_Market_Analysis_Dir,main_delimiter,'US_Int_Market\CDF']);
9 save_dir = ([SST_US_Int_CDF_Dir,main_delimiter,'Output']);
10 Upload_Dir = ([Load_Pre_Process_Dir,'\VOT\US_Int_Market\Output']);
11 TravelTime_Upload_Dir = ([Load_Pre_Process_Dir,main_delimiter,'SST_Travel_Times\Output']);
12 Upload_PreProc_CDF_Dir = ([Load_Pre_Process_Dir,main_delimiter,'CDF\US_Int_Market\Output']);
13
14 %Create Output directory
15 if exist((save_dir),'dir') == 0
16
17 mkdir(([SST_US_Int_CDF_Dir,main_delimiter,'Output']))
18
19 end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0
20
21 addpath(([SST_US_Int_CDF_Dir,main_delimiter,'Output']))
22
```

```
23 Fare_per_mile_paid_by_OD_with_VOT_Adjustment_US_Int(Upload_Dir, Upload_PreProc_CDF_Dir,
Input_Folder_Dir,TravelTime_Upload_Dir,
main_delimiter, save_dir, mach_overland, mach_overwater, vot_limit,US_Int_VOT,date,aircraft_range_overland,aircraft_range_overwater)
24
25 return;
26
1
2 function [] = Fare_per_mile_paid_by_OD_with_VOT_Adjustment_US_Int(Upload_Dir, Upload_PreProc_CDF_Dir,
Input_Folder_Dir,TravelTime_Upload_Dir,
main_delimiter, save_dir, mach_overland, mach_overwater, vot_limit,US_Int_VOT,date,aircraft_range_overland,aircraft_range_overwater)
3
4
load(([TravelTime_Upload_Dir,main_delimiter,'Travel_Times_with_Ground_Times_OverlandRange_',num2str(aircraft_range_overland),'_nm_1
_',num2str
((mach_overland-1)*10),'M_1_',num2str((mach_overwater-1)*10),'M.mat']),'Travel_Times')
5
6 load(([Input_Folder_Dir,main_delimiter,'OD_Pairs_Year_2016.mat']),'OD_Pairs_Year_2016')
7
8 load(([Upload_Dir,main_delimiter,'Value_of_Time_US_Int.mat']),'Value_of_Time')
9
10
load(([Upload_PreProc_CDF_Dir,main_delimiter,'OD_Fare_Per_Mile_Paid_Premium_Original_US_Int.mat']),'OD_Fare_Per_Mile_Paid_Premium
')
11
12 OAG_Data = OD_Pairs_Year_2016;
13 clear OD_Pairs_Year_2016
14 %%
15
16 if US_Int_VOT ~= 0
17
18 Value_of_Time.Value(:) = US_Int_VOT;
19
20 end
21
22 %%
23 %vot median value
24 index = Value_of_Time.Value <= vot_limit;
25 vot_median = round(prctile(Value_of_Time.Value(index),50),0);
26
27 %%
28
29 OAG_dep_arr = strcat(OAG_Data.DepAirport,'_',OAG_Data.ArrAirport);
30 TT_dep_arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);
31
32 [~,matched_index] = ismember(TT_dep_arr,OAG_dep_arr);
33
34 Travel_Times.SubSonic_Travel_Time = OAG_Data.OD_AverageElapsedTime_hrs(matched_index);
35
36 Travel_Times.Travel_Time_Saving = Travel_Times.SubSonic_Travel_Time - (Travel_Times.Travel_Time);
37
38 negative_saving = find(Travel_Times.Travel_Time_Saving < 0);
39
40 if isempty(negative_saving) == 0
41
42 Travel_Times.DepAirport(negative_saving) = [];
43 Travel_Times.ArrAirport(negative_saving) = [];
44 Travel_Times.Distance(negative_saving) = [];
45 Travel_Times.Travel_Time(negative_saving) = [];
46 Travel_Times.SubSonic_Travel_Time(negative_saving) = [];
47 end
48
49 TT_dep_arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);
50
51 length_of_OD = length(OD_Fare_Per_Mile_Paid_Premium);
52
53 for OD = 1:length_of_OD
```

```
54
55
56 % if mod(OD, 100) == 0
57 % disp([num2str(OD), '/', num2str(length_of_OD)]);
58 %
59 % end
60
61 [TT_OD_index,~] = ismember(TT_dep_arr,OD_Fare_Per_Mile_Paid_Premium(OD).OD);
62
63 if sum(TT_OD_index) ~= 0
64
65 %%
66 od_char = char(OD_Fare_Per_Mile_Paid_Premium(OD).OD);
67 origin_airport = cellstr(od_char(1:3));
68 [vot_od_index,~] = ismember(Value_of_Time.Airport,origin_airport);
69 vot_od_count = 1;
70
71 if sum(vot_od_index) ~= 0
72 vot_od_count = Value_of_Time.Number_of_records(vot_od_index);
73 if Value_of_Time.Value(vot_od_index) > vot_limit
74
75 Value_of_Time_od = vot_limit;
76 Final_Value_of_Time_By_OD.Origin_Median_Used_Yes1_No_0(OD,1) = 0;
77 Final_Value_of_Time_By_OD.Origin_Limit_Used_Yes1_No_0(OD,1) = 1;
78
79 else
80
81 Value_of_Time_od = round(Value_of_Time.Value(vot_od_index),0);
82 Final_Value_of_Time_By_OD.Origin_Median_Used_Yes1_No_0(OD,1) = 0;
83 Final_Value_of_Time_By_OD.Origin_Limit_Used_Yes1_No_0(OD,1) = 0;
84 end
85
86 else
87
88 Value_of_Time_od = vot_median;
89 Final_Value_of_Time_By_OD.Origin_Median_Used_Yes1_No_0(OD,1) = 1;
90 Final_Value_of_Time_By_OD.Origin_Limit_Used_Yes1_No_0(OD,1) = 0;
91 end
92 %%
93
94 destination_airport = cellstr(od_char(5:7));
95
96 [vot_do_index,~] = ismember(Value_of_Time.Airport,destination_airport);
97 vot_do_count = 1;
98
99 if sum(vot_do_index) ~= 0
100 vot_do_count = Value_of_Time.Number_of_records(vot_do_index);
101 if Value_of_Time.Value(vot_do_index) > vot_limit
102
103 Value_of_Time_do = vot_limit;
104 Final_Value_of_Time_By_OD.Destination_Median_Used_Yes1_No_0(OD,1) = 0;
105 Final_Value_of_Time_By_OD.Destination_Limit_Used_Yes1_No_0(OD,1) = 1;
106 else
107
108 Value_of_Time_do = round(Value_of_Time.Value(vot_do_index),0);
109 Final_Value_of_Time_By_OD.Destination_Median_Used_Yes1_No_0(OD,1) = 0;
110 Final_Value_of_Time_By_OD.Destination_Limit_Used_Yes1_No_0(OD,1) = 0;
111 end
112
113 else
114
115 Value_of_Time_do = vot_median;
116 Final_Value_of_Time_By_OD.Destination_Median_Used_Yes1_No_0(OD,1) = 1;
117 Final_Value_of_Time_By_OD.Destination_Limit_Used_Yes1_No_0(OD,1) = 0;
118 end
119
```

```
120 VOT_value = round((Value_of_Time_od * vot_od_count + Value_of_Time_do * vot_do_count) / sum([vot_od_count;vot_do_count]),0);
121
122 Final_Value_of_Time_By_OD.Origin_Airport(OD,1)= origin_airport;
123 Final_Value_of_Time_By_OD.Origin_VOT(OD,1) = Value_of_Time_od;
124 Final_Value_of_Time_By_OD.Origin_Records(OD,1) = vot_od_count;
125
126 Final_Value_of_Time_By_OD.Destination_Airport(OD,1)= destination_airport;
127 Final_Value_of_Time_By_OD.Destination_VOT(OD,1) = Value_of_Time_do;
128 Final_Value_of_Time_By_OD.Destination_Records(OD,1) = vot_do_count;
129 Final_Value_of_Time_By_OD.Weighted_Avg_VOT(OD,1) = VOT_value;
130 Final_Value_of_Time_By_OD.Weighted_Avg_VOT_Records(OD,1) = vot_od_count + vot_do_count;
131
132 %%
133
134 additional_fare_per_mile = round((VOT_value * Travel_Times.Travel_Time_Saving(TT_OD_index)) ./
Travel_Times.Distance(TT_OD_index),2);
135
136 OD_Fare_Per_Mile_Paid_Premium(OD).Unique_Fare = OD_Fare_Per_Mile_Paid_Premium(OD).Unique_Fare + additional_fare_per_mile;
137
138 end
139
140 end
141
142 save(([save_dir,main_delimiter,'OD_Fare_Per_Mile_Paid_Premium_Adjusted_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_US_Int',date,'.mat']),'OD_Fare_Per_Mile_Paid_Premium')
143
save(([save_dir,main_delimiter,'Final_Value_of_Time_By_OD_OverlandRange_',num2str(aircraft_range_overland),'_nm_OverwaterRange_',nu
m2str
(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-
1)*10),'_US_Int',
date,'.mat']),'Final_Value_of_Time_By_OD')
144
145 return;

1 function [] =
SST_US_Int_Forecast_Main_Function(SST_Market_Analysis_Dir,Load_Pre_Process_Dir,Airport_List_SST_Compatibility,airport_list,
mach_overland,mach_overwater,number_of_seats,FSF,hours_per_year,acft_pax_load_factor,SST_Year_Start,SST_Year_End,vot_limit,aircraft_r
ange_overland,
aircraft_range_overwater,min_distance_statute_mile,max_distance_statute_mile,min_demand,market_share,US_Int_VOT,date,
run_airport_gate_compatibility_section, run_airport_runway_compatibility_section,runway_length_min_ft,airport_runway_length_data)
2 %SST Market Script
3
4 local_disc = '';
5 main_delimiter = '\';
6 Input_Folder_Dir = ([local_disc,'..\..\..\..\..\..\SST_2020_Input']);
7 SST_US_Int_Forecast_Dir = ([SST_Market_Analysis_Dir,main_delimiter,'US_Int_Market\SST_Forecast']);
8 save_dir = ([SST_US_Int_Forecast_Dir,main_delimiter,'Output']);
9 Upload_VOT_Dir = ([Load_Pre_Process_Dir,main_delimiter,'VOT\US_Int_Market\Output']);
10 Upload_PreProc_CDF_Dir = ([Load_Pre_Process_Dir,main_delimiter,'CDF\US_Int_Market\Output']);
11 Upload_CDF_Dir = ([SST_Market_Analysis_Dir,main_delimiter,'US_Int_Market\CDF\Output']);
12 TravelTime_Upload_Dir = ([Load_Pre_Process_Dir,main_delimiter,'SST_Travel_Times\Output']);
13
14 %Create Output directory
15 if exist((save_dir),'dir') == 0
16
17 mkdir(([SST_US_Int_Forecast_Dir,main_delimiter,'Output']))
18
19 end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0
20
21 addpath(([SST_US_Int_Forecast_Dir,main_delimiter,'Output']))
22
23 %%
24
25 load(([Input_Folder_Dir,main_delimiter,'ICAO_Trip_Distribution_All.mat']),'Trip_Distribution_All')
26
```

```
27 load(([Input_Folder_Dir,main_delimiter,'OD_Pairs_Year_2016.mat']),'OD_Pairs_Year_2016')
28
29
load(([TravelTime_Upload_Dir,main_delimiter,'Travel_Times_with_Ground_Times_OverlandRange_',num2str(aircraft_range_overland),'_nm_1
_',num2str
((mach_overland-1)*10),'M_1_',num2str((mach_overwater-1)*10),'M.mat']),'Travel_Times')
30
31 load(([Upload_CDF_Dir,main_delimiter,'OD_Fare_Per_Mile_Paid_Premium_Adjusted_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_US_Int',date,'.mat']),'OD_Fare_Per_Mile_Paid_Premium')
32
33 load(([Upload_PreProc_CDF_Dir,main_delimiter,'Fare_Per_Mile_Paid_Premium_US_Int.mat']),'Fare_per_Mile_Paid_Premium')
34
35 load(([Upload_VOT_Dir,main_delimiter,'Value_of_Time_US_Int.mat']),'Value_of_Time')
36
37
load(([Input_Folder_Dir,main_delimiter,'Fare_data\',num2str(number_of_seats),'_Seats\Fare_Low_Boom_Int',date,'.mat']),'cost_per_passenge
r_mile')
38 %%
39
40 if US_Int_VOT ~= 0
41
42 Value_of_Time.Value(:) = US_Int_VOT;
43
44 end
45
46 %% Additional Parameters (calculated)
47 minimum_premium_seats = number_of_seats * 260; %1 flight per day, 260 days/year
48
49 index = Value_of_Time.Value <= vot_limit;
50 vot_median = round(prctile(Value_of_Time.Value(index),50),0);
51
52 %% FSF
53
54 fpm_reduction = 1-(100-FSF)/10 * 0.0346;
55 cost_per_passenger_mile.Over_land.fare = round(cost_per_passenger_mile.Over_land.fare .* fpm_reduction,3);
56 cost_per_passenger_mile.Over_water.fare = round(cost_per_passenger_mile.Over_water.fare .* fpm_reduction,3);
57
58 [OD_seats_percent_Premium,OD_Pairs_Year_2016,Trip_Distribution_All,generic_cdf,Fare_per_mile_cost] =
Fare_Percent_by_OD_Premium_US_Int
(Fare_per_Mile_Paid_Premium,
OD_Fare_Per_Mile_Paid_Premium,OD_Pairs_Year_2016,Travel_Times,Trip_Distribution_All,SST_Year_Start,SST_Year_End,
cost_per_passenger_mile, vot_median,aircraft_range_overland,aircraft_range_overwater,min_distance_statute_mile);
59
60 %%
61
62 SST_Market_Premium = SST_US_Int_Market_Premium(Trip_Distribution_All,
OD_Pairs_Year_2016,OD_seats_percent_Premium,generic_cdf,
Fare_per_mile_cost,SST_Year_Start,SST_Year_End,min_distance_statute_mile,max_distance_statute_mile,min_demand);
63
64 %%
65
66 [SST_Market_Share_Premium] =
SST_US_Int_Market_Share_Premium(SST_Market_Premium,SST_Year_Start,SST_Year_End,market_share,save_dir,
main_delimiter,aircraft_range_overwater,mach_overland,mach_overwater,FSF,number_of_seats);
67
68 %%
69
70
Final_SST_US_Int_Market(SST_Market_Share_Premium,Airport_List_SST_Compatibility,airport_list,minimum_premium_seats,hours_per_year,
number_of_seats,acft_pax_load_factor,SST_Year_Start,SST_Year_End,FSF,Travel_Times,aircraft_range_overwater,save_dir,main_delimiter,ma
ch_overland,
mach_overwater,date,run_airport_gate_compatibility_section,
run_airport_runway_compatibility_section,runway_length_min_ft,airport_runway_length_data,
aircraft_range_overland);
71
```

```
72 return;

1
2 function [OD_seats_percent_Premium,OD_Pairs_Year_2016,Trip_Distribution_All,generic_cdf,Fare_per_mile_cost] =
Fare_Percent_by_OD_Premium_US_Int(Fare_per_Mile_Paid_Premium, OD_Fare_Per_Mile_Paid_Premium,OD_Pairs_Year_2016,Travel_Times,
Trip_Distribution_All,SST_Year_Start,SST_Year_End, cost_per_passenger_mile, vot_median,aircraft_range_overland,aircraft_range_overwater,
min_distance_statute_mile)
3 delete_short_distance_od = find(OD_Pairs_Year_2016.DistStMiles < min_distance_statute_mile);
4
5 OD_Pairs_Year_2016.Carrier_Name(delete_short_distance_od) = [];
6 OD_Pairs_Year_2016.DepAirport_UniqueID(delete_short_distance_od) = [];
7 OD_Pairs_Year_2016.DepAirport(delete_short_distance_od) = [];
8 OD_Pairs_Year_2016.DepIATACtry(delete_short_distance_od) = [];
9 OD_Pairs_Year_2016.DepReg(delete_short_distance_od) = [];
10 OD_Pairs_Year_2016.ArrAirport_UniqueID(delete_short_distance_od) = [];
11 OD_Pairs_Year_2016.ArrAirport(delete_short_distance_od) = [];
12 OD_Pairs_Year_2016.ArrIATACtry(delete_short_distance_od) = [];
13 OD_Pairs_Year_2016.ArrReg(delete_short_distance_od) = [];
14 OD_Pairs_Year_2016.InternationalDomestic(delete_short_distance_od) = [];
15 OD_Pairs_Year_2016.Total_Seats(delete_short_distance_od) = [];
16 OD_Pairs_Year_2016.Total_FstSeats(delete_short_distance_od) = [];
17 OD_Pairs_Year_2016.Total_BusSeats(delete_short_distance_od) = [];
18 OD_Pairs_Year_2016.Total_EcoSeats(delete_short_distance_od) = [];
19 OD_Pairs_Year_2016.Total_Frequency(delete_short_distance_od) = [];
20 OD_Pairs_Year_2016.OD_AverageElapsedTime_hrs(delete_short_distance_od) = [];
21 OD_Pairs_Year_2016.DistStMiles(delete_short_distance_od) = [];
22 OD_Pairs_Year_2016.AcftData(delete_short_distance_od) = [];
23
24 for year = SST_Year_Start:SST_Year_End
25
26 Trip_Distribution_All.(['Year_',num2str(year)]).DepAirport_UniqueID(delete_short_distance_od) = [];
27 Trip_Distribution_All.(['Year_',num2str(year)]).ArrAirport_UniqueID(delete_short_distance_od) = [];
28 Trip_Distribution_All.(['Year_',num2str(year)]).Seats(delete_short_distance_od) = [];
29 end
30 %%
31 length_of_data = length(OD_Pairs_Year_2016.DepAirport);
32
33 Dep_Arr_All = strcat(OD_Pairs_Year_2016.DepAirport,'_',OD_Pairs_Year_2016.ArrAirport);
34
35 length_of_OD_fare = length(OD_Fare_Per_Mile_Paid_Premium);
36 Fare_Dep_Arr_All = cell(length_of_OD_fare,1);
37
38 for fare_od = 1:length_of_OD_fare
39
40 Fare_Dep_Arr_All(fare_od) = OD_Fare_Per_Mile_Paid_Premium(fare_od).OD;
41 end
42
43 Travel_Times_Dep_Arr_All = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);
44
45 OD_seats_percent_Premium = zeros(length_of_data,1);
46 Fare_per_mile_cost = zeros(length_of_data,1);
47 generic_cdf = zeros(length_of_data,1);
48
49 for OD = 1:length_of_data
50
51 clear aircraft_range
52
53 [od_index,~] = ismember(Fare_Dep_Arr_All,Dep_Arr_All(OD));
54
55
56 od_distance = round((OD_Pairs_Year_2016.DistStMiles(OD)) * 0.868976,0); %st to nm conversion
57
58 [travel_time_od_index,~] = ismember(Travel_Times_Dep_Arr_All,Dep_Arr_All(OD));
59
60 if sum(travel_time_od_index) ~= 0
61
```

```
62 if Travel_Times.Overland_Percent(travel_time_od_index,1) > 0.25
63
64 aircraft_range = aircraft_range_overland;
65
66 else
67
68 aircraft_range = aircraft_range_overwater;
69 end
70
71 if od_distance > aircraft_range
72
73 while od_distance > aircraft_range
74
75 od_distance = ceil(od_distance/2);
76 end
77 end
78
79 %%
80
81 if Travel_Times.Overland_Percent(travel_time_od_index) > 0.25
82
83 %overland
84
85 if od_distance >= max(cost_per_passenger_mile.Over_land.distance)
86
87 Fare_per_Mile_Paid_Threshold_Premium = min(cost_per_passenger_mile.Over_land.fare);
88
89 elseif od_distance <= min(cost_per_passenger_mile.Over_land.distance)
90
91 Fare_per_Mile_Paid_Threshold_Premium = max(cost_per_passenger_mile.Over_land.fare);
92
93 else
94
95 %interpolate bewteen values
96
97 difference_in_value = od_distance - cost_per_passenger_mile.Over_land.distance;
98
99 lower_bound_index = max(find(difference_in_value>0));
100
101 upper_bound_index = lower_bound_index + 1;
102
103 Fare_per_Mile_Paid_Threshold_Premium = cost_per_passenger_mile.Over_land.fare(lower_bound_index) - ((cost_per_passenger_mile.
Over_land.distance(lower_bound_index) - od_distance) / (cost_per_passenger_mile.Over_land.distance(lower_bound_index) -
cost_per_passenger_mile.
Over_land.distance(upper_bound_index))) * (cost_per_passenger_mile.Over_land.fare(lower_bound_index) -
cost_per_passenger_mile.Over_land.fare
(upper_bound_index));
104
105 end
106
107 else
108
109 %overwater
110
111
112 if od_distance >= max(cost_per_passenger_mile.Over_water.distance)
113
114 Fare_per_Mile_Paid_Threshold_Premium = min(cost_per_passenger_mile.Over_water.fare);
115
116 elseif od_distance <= min(cost_per_passenger_mile.Over_water.distance)
117
118 Fare_per_Mile_Paid_Threshold_Premium = max(cost_per_passenger_mile.Over_water.fare);
119
120 else
121
122 %interpolate bewteen values
```

```
123
124 difference_in_value = od_distance - cost_per_passenger_mile.Over_water.distance;
125
126 lower_bound_index = max(find(difference_in_value>0));
127
128 upper_bound_index = lower_bound_index + 1;
129
130 Fare_per_Mile_Paid_Threshold_Premium = cost_per_passenger_mile.Over_water.fare(lower_bound_index) - ((cost_per_passenger_mile.
Over_water.distance(lower_bound_index) - od_distance) / (cost_per_passenger_mile.Over_water.distance(lower_bound_index) -
cost_per_passenger_mile.
Over_water.distance(upper_bound_index))) * (cost_per_passenger_mile.Over_water.fare(lower_bound_index) -
cost_per_passenger_mile.Over_water.fare
(upper_bound_index));
131
132 end
133
134 end
135
136 Fare_per_Mile_Paid_Threshold_Premium = Fare_per_Mile_Paid_Threshold_Premium * 0.868976; %convert $/nm. to $/sm.
137
138 if sum(od_index) ~= 0
139
140 if min(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare) > Fare_per_Mile_Paid_Threshold_Premium
141
142 OD_seats_percent_Premium(OD) = 1.0;
143
144 else
145
146 if max(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare) >= Fare_per_Mile_Paid_Threshold_Premium
147
148 if sum(ismember(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare,Fare_per_Mile_Paid_Threshold_Premium)) == 1
149
150 [fare_index,~] = ismember(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare,Fare_per_Mile_Paid_Threshold_Premium);
151
152 OD_seats_percent_Premium(OD) = 1 - OD_Fare_Per_Mile_Paid_Premium(od_index).Cummulative_Count(fare_index);
153
154 else
155
156 %interpolate bewteen values
157
158 difference_in_value = Fare_per_Mile_Paid_Threshold_Premium - OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare;
159
160 lower_bound_index = max(find(difference_in_value>0));
161
162 upper_bound_index = lower_bound_index + 1;
163
164 OD_seats_percent_Premium(OD) = 1 - (OD_Fare_Per_Mile_Paid_Premium(od_index).Cummulative_Count(lower_bound_index) -
((OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare(lower_bound_index) - Fare_per_Mile_Paid_Threshold_Premium) ./
(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare(lower_bound_index) - OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare
(upper_bound_index))) .* (OD_Fare_Per_Mile_Paid_Premium(od_index).Cummulative_Count(lower_bound_index) -
OD_Fare_Per_Mile_Paid_Premium
(od_index).Cummulative_Count(upper_bound_index)));
165 end
166
167 else
168
169 OD_seats_percent_Premium(OD) = 0;
170
171 end
172
173 end
174
175 else
176
177 generic_cdf(OD,1) = 1;
178
```

```
179 [od_index_2,~] = ismember(Travel_Times_Dep_Arr_All,Dep_Arr_All(OD));
180
181 if sum(od_index_2) ~= 0
182
183 travel_time_saving = OD_Pairs_Year_2016.OD_AverageElapsedTime_hrs(OD) - Travel_Times.Travel_Time(od_index_2);
184
185 if travel_time_saving > 0
186
187 additional_fare = (vot_median * travel_time_saving) ./ Travel_Times.Distance(od_index_2);
188
189 Fare_per_Mile_Paid_Premium_adjusted.Fare = round((Fare_per_Mile_Paid_Premium.Fare + additional_fare),2);
190 Fare_per_Mile_Paid_Premium_adjusted.Cumulative = Fare_per_Mile_Paid_Premium.Cumulative;
191
192 %%
193 if min(Fare_per_Mile_Paid_Premium_adjusted.Fare) > Fare_per_Mile_Paid_Threshold_Premium
194
195 OD_seats_percent_Premium(OD) = 1.0;
196
197 else
198
199 if max(Fare_per_Mile_Paid_Premium_adjusted.Fare) >= Fare_per_Mile_Paid_Threshold_Premium
200
201 if sum(ismember(Fare_per_Mile_Paid_Premium_adjusted.Fare,Fare_per_Mile_Paid_Threshold_Premium)) == 1
202
203 [fare_index,~] = ismember(Fare_per_Mile_Paid_Premium_adjusted.Fare,Fare_per_Mile_Paid_Threshold_Premium);
204
205 OD_seats_percent_Premium(OD) = 1 - Fare_per_Mile_Paid_Premium_adjusted.Cumulative(fare_index);
206
207 else
208
209 %interpolate bewteen values
210
211 difference_in_value = Fare_per_Mile_Paid_Threshold_Premium - Fare_per_Mile_Paid_Premium_adjusted.Fare;
212
213 lower_bound_index = max(find(difference_in_value>0));
214
215 upper_bound_index = lower_bound_index + 1;
216
217 OD_seats_percent_Premium(OD) = 1 - (Fare_per_Mile_Paid_Premium_adjusted.Cumulative(lower_bound_index) -
((Fare_per_Mile_Paid_Premium_adjusted.Fare(lower_bound_index) - Fare_per_Mile_Paid_Threshold_Premium) ./
(Fare_per_Mile_Paid_Premium_adjusted.
Fare(lower_bound_index) - Fare_per_Mile_Paid_Premium_adjusted.Fare(upper_bound_index))) .*
(Fare_per_Mile_Paid_Premium_adjusted.Cumulative
(lower_bound_index) - Fare_per_Mile_Paid_Premium_adjusted.Cumulative(upper_bound_index)));
218 end
219
220 else
221
222 OD_seats_percent_Premium(OD) = 0;
223
224 end
225
226 end
227 %%
228
229 else
230
231 [index_Premium,~] = ismember(Fare_per_Mile_Paid_Premium.Fare,Fare_per_Mile_Paid_Threshold_Premium);
232
233 if sum(index_Premium) ~= 0
234
235 Premium_seats_percent = 1 - Fare_per_Mile_Paid_Premium.Cumulative(index_Premium);
236 else
237 difference_in_value = Fare_per_Mile_Paid_Threshold_Premium - Fare_per_Mile_Paid_Premium.Fare;
238
239 lower_bound_index = max(find(difference_in_value>0));
```

```
240
241 upper_bound_index = lower_bound_index + 1;
242
243 Premium_seats_percent = 1 - (Fare_per_Mile_Paid_Premium.Cumulative(lower_bound_index) - ((Fare_per_Mile_Paid_Premium.Fare
(lower_bound_index) - Fare_per_Mile_Paid_Threshold_Premium) ./ (Fare_per_Mile_Paid_Premium.Fare(lower_bound_index) -
Fare_per_Mile_Paid_Premium.
Fare(upper_bound_index))) .* (Fare_per_Mile_Paid_Premium.Cumulative(lower_bound_index) - Fare_per_Mile_Paid_Premium.Cumulative
(upper_bound_index)));
244 end
245
246 OD_seats_percent_Premium(OD) = Premium_seats_percent;
247
248 end
249
250 else
251
252 [index_Premium,~] = ismember(Fare_per_Mile_Paid_Premium.Fare,Fare_per_Mile_Paid_Threshold_Premium);
253 if sum(index_Premium) ~= 0
254
255 Premium_seats_percent = 1 - Fare_per_Mile_Paid_Premium.Cumulative(index_Premium);
256 else
257 difference_in_value = Fare_per_Mile_Paid_Threshold_Premium - Fare_per_Mile_Paid_Premium.Fare;
258
259 lower_bound_index = max(find(difference_in_value>0));
260
261 upper_bound_index = lower_bound_index + 1;
262
263 Premium_seats_percent = 1 - (Fare_per_Mile_Paid_Premium.Cumulative(lower_bound_index) - ((Fare_per_Mile_Paid_Premium.Fare
(lower_bound_index) - Fare_per_Mile_Paid_Threshold_Premium) ./ (Fare_per_Mile_Paid_Premium.Fare(lower_bound_index) -
Fare_per_Mile_Paid_Premium.
Fare(upper_bound_index))) .* (Fare_per_Mile_Paid_Premium.Cumulative(lower_bound_index) - Fare_per_Mile_Paid_Premium.Cumulative
(upper_bound_index)));
264 end
265
266 OD_seats_percent_Premium(OD) = Premium_seats_percent;
267
268 end
269
270 end
271
272 Fare_per_mile_cost(OD,1) = Fare_per_Mile_Paid_Threshold_Premium;
273 end
274
275 end
276
277 return;
278
```

```
1 function SST_Market_Premium = SST_US_Int_Market_Premium(Trip_Distribution_All,
OD_Pairs_Year_2016,OD_seats_percent_Premium,generic_cdf,
Fare_per_mile_cost,SST_Year_Start,SST_Year_End,min_distance_statute_mile,max_distance_statute_mile,min_demand)
2
3 OAG2016 = OD_Pairs_Year_2016;
4 OAG2016.Premium_Seats = OAG2016.Total_FstSeats + OAG2016.Total_BusSeats;
5 OAG2016.OD_seats_percent_Premium = OD_seats_percent_Premium;
6 OAG2016.Fare_per_mile_cost = Fare_per_mile_cost;
7
8 clear OD_Pairs_Year_2016
9 All_OD_Pair_Data = strcat(OAG2016.DepAirport,'_',OAG2016.ArrAirport);
10
11 Dep = char(Trip_Distribution_All.Year_2017.DepAirport_UniqueID);
12 Dep = Dep(:,1:3);
13 Dep = cellstr(Dep);
14
15 Arr = char(Trip_Distribution_All.Year_2017.ArrAirport_UniqueID);
16 Arr = Arr(:,1:3);
```

```
Arr = cellstr(Arr);

All_Trip_Distribution = strcat(Dep,'_',Arr);

[~ , OD_Pair_Index] = ismember(All_OD_Pair_Data,All_Trip_Distribution);

delete_record = find(OD_Pair_Index == 0);
OAG2016.Carrier_Name(delete_record) = [];
OAG2016.DepAirport_UniqueID(delete_record) = [];
OAG2016.DepAirport(delete_record) = [];
OAG2016.DepIATACtry(delete_record) = [];
OAG2016.DepReg(delete_record) = [];
OAG2016.ArrAirport_UniqueID(delete_record) = [];
OAG2016.ArrAirport(delete_record) = [];
OAG2016.ArrIATACtry(delete_record) = [];
OAG2016.ArrReg(delete_record) = [];
OAG2016.InternationalDomestic(delete_record) = [];
OAG2016.Total_Seats(delete_record) = [];
OAG2016.Premium_Seats(delete_record) = [];
OAG2016.Total_FstSeats(delete_record) = [];
OAG2016.Total_BusSeats(delete_record) = [];
OAG2016.Total_EcoSeats(delete_record) = [];
OAG2016.Total_Frequency(delete_record) = [];
OAG2016.OD_AverageElapsedTime_hrs(delete_record) = [];
OAG2016.DistStMiles(delete_record) = [];
OAG2016.AcftData(delete_record) = [];

OAG2016.OD_seats_percent_Premium(delete_record) = [];
OAG2016.Fare_per_mile_cost(delete_record) = [];
OAG2016.Percent = OAG2016.Premium_Seats ./ OAG2016.Total_Seats;

%%

for year = SST_Year_Start:SST_Year_End

SST.(['Year_',num2str(year)]).DepAirport_UniqueID = OAG2016.DepAirport_UniqueID;
SST.(['Year_',num2str(year)]).DepAirport = OAG2016.DepAirport;
SST.(['Year_',num2str(year)]).DepIATACtry = OAG2016.DepIATACtry;
SST.(['Year_',num2str(year)]).DepReg = OAG2016.DepReg;
SST.(['Year_',num2str(year)]).ArrAirport_UniqueID = OAG2016.ArrAirport_UniqueID;
SST.(['Year_',num2str(year)]).ArrAirport = OAG2016.ArrAirport;
SST.(['Year_',num2str(year)]).ArrIATACtry = OAG2016.ArrIATACtry;
SST.(['Year_',num2str(year)]).ArrReg = OAG2016.ArrReg;
SST.(['Year_',num2str(year)]).InternationalDomestic = OAG2016.InternationalDomestic;

SST.(['Year_',num2str(year)]).Total_Seats = Trip_Distribution_All.(['Year_',num2str(year)]).Seats;
SST.(['Year_',num2str(year)]).Premium_Seats = ceil(Trip_Distribution_All.(['Year_',num2str(year)]).Seats .* OAG2016.Percent);

SST.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs = OAG2016.OD_AverageElapsedTime_hrs;
SST.(['Year_',num2str(year)]).DistStMiles = OAG2016.DistStMiles;
SST.(['Year_',num2str(year)]).OD_premium_seats_percent = OAG2016.OD_seats_percent_Premium;
SST.(['Year_',num2str(year)]).Fare_per_mile_cost = OAG2016.Fare_per_mile_cost;
SST.(['Year_',num2str(year)]).generic_cdf = generic_cdf;

end

%%

SST_Market_Premium = SST;

for year = SST_Year_Start:SST_Year_End

max_distance_index = find(SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles > max_distance_statute_mile);

if isempty(max_distance_index) == 0

```

```
SST_Market_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepAirport(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepIATACtry(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepReg(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrIATACtry(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrReg(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).InternationalDomestic(max_distance_index) = [];

SST_Market_Premium.(['Year_',num2str(year)]).Total_Seats(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).Premium_Seats(max_distance_index) = [];

SST_Market_Premium.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost(max_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).generic_cdf(max_distance_index) = [];

end

clear max_distance_index

min_distance_index = find(SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles < min_distance_statute_mile);

if isempty(min_distance_index) == 0

SST_Market_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepAirport(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepIATACtry(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepReg(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrIATACtry(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrReg(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).InternationalDomestic(min_distance_index) = [];

SST_Market_Premium.(['Year_',num2str(year)]).Total_Seats(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).Premium_Seats(min_distance_index) = [];

SST_Market_Premium.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost(min_distance_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).generic_cdf(min_distance_index) = [];

end

clear min_distance_index

min_demnad_index = find(SST_Market_Premium.(['Year_',num2str(year)]).Total_Seats < min_demand);

if isempty(min_demnad_index) == 0

SST_Market_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID(min_demnad_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepAirport(min_demnad_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepIATACtry(min_demnad_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).DepReg(min_demnad_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID(min_demnad_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport(min_demnad_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrIATACtry(min_demnad_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).ArrReg(min_demnad_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).InternationalDomestic(min_demnad_index) = [];

SST_Market_Premium.(['Year_',num2str(year)]).Total_Seats(min_demnad_index) = [];
SST_Market_Premium.(['Year_',num2str(year)]).Premium_Seats(min_demnad_index) = [];
```

```
149
150 SST_Market_Premium.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(min_demnad_index) = [];
151 SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles(min_demnad_index) = [];
152 SST_Market_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent(min_demnad_index) = [];
153 SST_Market_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost(min_demnad_index) = [];
154 SST_Market_Premium.(['Year_',num2str(year)]).generic_cdf(min_demnad_index) = [];
155 end
156
157 clear min_demnad_index
158 end
159
160
161 return;
162
```

```
1
2 function [SST_Market_Share_Premium] =
SST_US_Int_Market_Share_Premium(SST_Market_Premium,SST_Year_Start,SST_Year_End,market_share,save_dir,
main_delimiter,aircraft_range_overwater,mach_overland,mach_overwater,FSF,number_of_seats)
3
4 SST_Market_Share_Premium = SST_Market_Premium;
5
6 for year = SST_Year_Start:SST_Year_End
7
8 SST_Market_Share_Premium.(['Year_',num2str(year)]).SST_Premium_Seats = ceil(((SST_Market_Share_Premium.(['Year_',num2str(year)]).
Premium_Seats .* SST_Market_Share_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent) .* market_share));
9
10 end
11
12
%save(([save_dir,main_delimiter,'ICAO_SST_Market_Share_Premium_',num2str(aircraft_range_overwater),'nm_1_',num2str((mach_overland-
1)
*10),'M_1_',num2str((mach_overwater-
1)*10),'M_',num2str(number_of_seats),'Seater_FSF_',num2str(FSF),'_US_Int']),'SST_Market_Share_Premium')
13
14 return;
15
```

```
1
2 function [] =
Final_SST_US_Int_Market(SST_Market_Share_Premium,Airport_List_SST_Compatibility,airport_list,minimum_premium_seats,hours_per_year,
number_of_seats,acft_pax_load_factor,SST_Year_Start,SST_Year_End,FSF,Travel_Times,aircraft_range_overwater,save_dir,main_delimiter,ma
ch_overland,
mach_overwater,date,run_airport_gate_compatibility_section,
run_airport_runway_compatibility_section,runway_length_min_ft,airport_runway_length_data,
aircraft_range_overland)
3
4 acft_seat_capacity = number_of_seats;
5
6 for year = SST_Year_Start:SST_Year_End
7
8 US_dep = ismember(SST_Market_Share_Premium.(['Year_',num2str(year)]).DepIATACtry,'US');
9 US_arr = ismember(SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrIATACtry,'US');
10 US_dep_US_arr = US_dep + US_arr;
11 non_int_us_index = find(US_dep_US_arr ~= 1);
12
13 SST_Market_Share_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID(non_int_us_index) = [];
14 SST_Market_Share_Premium.(['Year_',num2str(year)]).DepAirport(non_int_us_index) = [];
15 SST_Market_Share_Premium.(['Year_',num2str(year)]).DepIATACtry(non_int_us_index) = [];
16 SST_Market_Share_Premium.(['Year_',num2str(year)]).DepReg(non_int_us_index) = [];
17 SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID(non_int_us_index) = [];
18 SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrAirport(non_int_us_index) = [];
19 SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrIATACtry(non_int_us_index) = [];
20 SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrReg(non_int_us_index) = [];
21 SST_Market_Share_Premium.(['Year_',num2str(year)]).InternationalDomestic(non_int_us_index) = [];
22 SST_Market_Share_Premium.(['Year_',num2str(year)]).Total_Seats(non_int_us_index) = [];
```

```
SST_Market_Share_Premium.(['Year_',num2str(year)]).Premium_Seats(non_int_us_index) = [];
SST_Market_Share_Premium.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(non_int_us_index) = [];
SST_Market_Share_Premium.(['Year_',num2str(year)]).DistStMiles(non_int_us_index) = [];
SST_Market_Share_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent(non_int_us_index) = [];
SST_Market_Share_Premium.(['Year_',num2str(year)]).SST_Premium_Seats(non_int_us_index) = [];
SST_Market_Share_Premium.(['Year_',num2str(year)]).generic_cdf(non_int_us_index) = [];
SST_Market_Share_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost(non_int_us_index) = [];

end

%%

for year = SST_Year_Start:SST_Year_End

SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID =
SST_Market_Share_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID;
SST_Final_Market.(['Year_',num2str(year)]).DepAirport = SST_Market_Share_Premium.(['Year_',num2str(year)]).DepAirport;
SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry = SST_Market_Share_Premium.(['Year_',num2str(year)]).DepIATACtry;
SST_Final_Market.(['Year_',num2str(year)]).DepReg = SST_Market_Share_Premium.(['Year_',num2str(year)]).DepReg;

SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID =
SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID;
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport = SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrAirport;
SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry = SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrIATACtry;
SST_Final_Market.(['Year_',num2str(year)]).ArrReg = SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrReg;

SST_Final_Market.(['Year_',num2str(year)]).Total_Seats = SST_Market_Share_Premium.(['Year_',num2str(year)]).Total_Seats;

SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats = SST_Market_Share_Premium.(['Year_',num2str(year)]).Premium_Seats;

SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs = SST_Market_Share_Premium.(['Year_',num2str(year)]).
OD_AverageElapsedTime_hrs;
SST_Final_Market.(['Year_',num2str(year)]).DistStMiles = SST_Market_Share_Premium.(['Year_',num2str(year)]).DistStMiles;

SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats =
SST_Market_Share_Premium.(['Year_',num2str(year)]).SST_Premium_Seats;
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats = SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats;% +
SST_Final_Market.
(['Year_',num2str(year)]).SST_Economy_Premium_Seats;
SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost =
SST_Market_Share_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost;
SST_Final_Market.(['Year_',num2str(year)]).generic_cdf = SST_Market_Share_Premium.(['Year_',num2str(year)]).generic_cdf;

end


for year = SST_Year_Start:SST_Year_End

od_with_min_demand_index = find(SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats >= minimum_premium_seats);

o_d =
strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(od_with_min_demand_index),'_',SST_Final_Market.(['Year_',num2str(year)]).
ArrAirport(od_with_min_demand_index));
d_o =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(od_with_min_demand_index),'_',SST_Final_Market.(['Year_',num2str(year)]).
DepAirport(od_with_min_demand_index));
od_do = [o_d;d_o];
od_do_unique = unique(od_do);

SST_Final_Market_OD =
strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);

delete_index = find(ismember(SST_Final_Market_OD,od_do_unique) == 0);

SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID(delete_index) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepAirport(delete_index) = [];
```

```
77 SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry(delete_index) = [];
78 SST_Final_Market.(['Year_',num2str(year)]).DepReg(delete_index) = [];
79 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID(delete_index) = [];
80 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(delete_index) = [];
81 SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry(delete_index) = [];
82 SST_Final_Market.(['Year_',num2str(year)]).ArrReg(delete_index) = [];
83 SST_Final_Market.(['Year_',num2str(year)]).Total_Seats(delete_index) = [];
84 SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats(delete_index) = [];
85 SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(delete_index) = [];
86 SST_Final_Market.(['Year_',num2str(year)]).DistStMiles(delete_index) = [];
87 SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats(delete_index) = [];
88 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(delete_index) = [];
89 SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost (delete_index) = [];
90 SST_Final_Market.(['Year_',num2str(year)]).generic_cdf(delete_index) = [];
91
92 end
93 %% Airport Gate Compatibility
94
95 if run_airport_gate_compatibility_section == 1
96
97 for year = SST_Year_Start:SST_Year_End
98 clear od_dist
99
100 airport_sst_compatibility_index = ismember(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,Airport_List_SST_Compatibility)==0;
101
102 sst_od = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);
103
104 while sum(airport_sst_compatibility_index) ~= 0
105
106 airport_sst_compatibility_index = find(airport_sst_compatibility_index);
107
108 no_comp_do =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)),'_',SST_Final_Market.(['Year_',
num2str(year)]).DepAirport(airport_sst_compatibility_index(1)));
109 [no_comp_do_index,~] = ismember(sst_od,no_comp_do);
110 length_of_od = length(SST_Final_Market.(['Year_',num2str(year)]).DepAirport);
111
112 [o_match_index] =
ismember(airport_list.Airport_IDs,SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)));
113 lat_o = airport_list.Apt_Lat(o_match_index);
114 lon_o = airport_list.Apt_Lon(o_match_index);
115
116 od_dist = zeros(length_of_od,1);
117 for airport = 1:length_of_od
118
119 [d_match_index] = ismember(airport_list.Airport_IDs,SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport));
120
121 lat_d = airport_list.Apt_Lat(d_match_index);
122 lon_d = airport_list.Apt_Lon(d_match_index);
123
124 od_dist(airport,1) = ceil(deg2sm(distance(lat_o,lon_o,lat_d,lon_d)));
125 end
126
127 zero_dist = od_dist == 0;
128
129 od_dist(zero_dist) = 999999;
130
131 min_dist = min(od_dist);
132
133 if min_dist <= 30
134
135 redirect_airport_index = find(od_dist==min_dist);
136
137 od_redirect_verification =
strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)),'_',SST_Final_Market.(['Year_',
num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)));
```

```
138 do_redirect_verification =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)),'_',SST_Final_Market.
(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)));
139
140
141 od_redirect_exist = ismember(sst_od,od_redirect_verification);
142 do_redirect_exist = ismember(sst_od,do_redirect_verification);
143
144 if sum(od_redirect_exist) == 0
145 %disp(year)
146 %disp('change')
147 %disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)))
148 %disp('to')
149 %disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)))
150
151 SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)) = SST_Final_Market.(['Year_',num2str(year)]).
DepAirport(redirect_airport_index(1));
152 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(no_comp_do_index) = SST_Final_Market.(['Year_',num2str(year)]).DepAirport
(redirect_airport_index(1));
153
154 else
155 %disp('add from')
156 %disp(strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]),'_',
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)])))
157 %disp('to')
158
%disp(strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(od_redirect_exist);find(do_redirect_exist)]),'_',SST_Final_Market.
(['Year_',num2str(year)]).ArrAirport([find(od_redirect_exist);find(do_redirect_exist)])))
159
160 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(od_redirect_exist) =
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats
(od_redirect_exist) + SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(airport_sst_compatibility_index(1));
161 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(do_redirect_exist) =
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats
(do_redirect_exist) + SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(no_comp_do_index);
162
163
164 SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
165 SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
166 SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
167 SST_Final_Market.(['Year_',num2str(year)]).DepReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
168 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
169 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
170 SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
171 SST_Final_Market.(['Year_',num2str(year)]).ArrReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
172 SST_Final_Market.(['Year_',num2str(year)]).Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
173 SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
174 SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs([find(no_comp_do_index);airport_sst_compatibility_index(1)]) =
[];
175 SST_Final_Market.(['Year_',num2str(year)]).DistStMiles([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
176 SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
177 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
178 SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
179 SST_Final_Market.(['Year_',num2str(year)]).generic_cdf([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
180
181 end
182
183
184 else
185
186 %disp(year)
187 %disp('delete')
188
189 %disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]))
190
191 %disp(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]))
```

```
192
193 SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
194 SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
195 SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
196 SST_Final_Market.(['Year_',num2str(year)]).DepReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
197 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
198 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
199 SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
200 SST_Final_Market.(['Year_',num2str(year)]).ArrReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
201 SST_Final_Market.(['Year_',num2str(year)]).Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
202 SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
203 SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs([find(no_comp_do_index);airport_sst_compatibility_index(1)]) =
[];
204 SST_Final_Market.(['Year_',num2str(year)]).DistStMiles([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
205 SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
206 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
207 SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
208 SST_Final_Market.(['Year_',num2str(year)]).generic_cdf([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
209
210
211 end
212 airport_sst_compatibility_index = ismember(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,Airport_List_SST_Compatibility)==0;
213
214 sst_od = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);
215
216 end
217
218 end
219 end
220
221 %% Runway Length Compatibility
222 if run_airport_runway_compatibility_section == 1
223
224 runway_length_index = airport_runway_length_data.runway_ft >= runway_length_min_ft;
225 Airport_List_SST_Compatibility = airport_runway_length_data.airport(runway_length_index);
226
227 for year = SST_Year_Start:SST_Year_End
228 clear od_dist
229
230 airport_sst_compatibility_index = ismember(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,Airport_List_SST_Compatibility)==0;
231
232 sst_od = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);
233
234 while sum(airport_sst_compatibility_index) ~= 0
235
236 airport_sst_compatibility_index = find(airport_sst_compatibility_index);
237
238 no_comp_do =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)),'_',SST_Final_Market.(['Year_',
num2str(year)]).DepAirport(airport_sst_compatibility_index(1)));
239 [no_comp_do_index,~] = ismember(sst_od,no_comp_do);
240 length_of_od = length(SST_Final_Market.(['Year_',num2str(year)]).DepAirport);
241
242 [o_match_index] =
ismember(airport_list.Airport_IDs,SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)));
243 lat_o = airport_list.Apt_Lat(o_match_index);
244 lon_o = airport_list.Apt_Lon(o_match_index);
245
246 od_dist = zeros(length_of_od,1);
247 for airport = 1:length_of_od
248
249 [d_match_index] = ismember(airport_list.Airport_IDs,SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport));
250
251 lat_d = airport_list.Apt_Lat(d_match_index);
252 lon_d = airport_list.Apt_Lon(d_match_index);
253
```

```
od_dist(airport,1) = ceil(deg2sm(distance(lat_o,lon_o,lat_d,lon_d)));
end

zero_dist = od_dist == 0;

od_dist(zero_dist) = 999999;

min_dist = min(od_dist);

if min_dist <= 30

redirect_airport_index = find(od_dist==min_dist);

od_redirect_verification =
strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)),'_',SST_Final_Market.(['Year_',
num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)));
do_redirect_verification =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)),'_',SST_Final_Market.
(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)));


od_redirect_exist = ismember(sst_od,od_redirect_verification);
do_redirect_exist = ismember(sst_od,do_redirect_verification);

if sum(od_redirect_exist) == 0
%disp(year)
%disp('line149')
%disp('change')
%disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)))
%disp('to')
%disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)))

SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)) = SST_Final_Market.(['Year_',num2str(year)]).
DepAirport(redirect_airport_index(1));
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(no_comp_do_index) = SST_Final_Market.(['Year_',num2str(year)]).DepAirport
(redirect_airport_index(1));

else

%disp('add from')
%disp(strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]),'_',
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)])))
%disp('to')

%disp(strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(od_redirect_exist);find(do_redirect_exist)]),'_',SST_Final_Market.
(['Year_',num2str(year)]).ArrAirport([find(od_redirect_exist);find(do_redirect_exist)])))

SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(od_redirect_exist) =
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats
(od_redirect_exist) + SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(airport_sst_compatibility_index(1));
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(do_redirect_exist) =
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats
(do_redirect_exist) + SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(no_comp_do_index);


SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
```

```
SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs([find(no_comp_do_index);airport_sst_compatibility_index(1)]) =
[];
SST_Final_Market.(['Year_',num2str(year)]).DistStMiles([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).generic_cdf([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];

end


else

%disp(year)

%disp('delete')

%disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]))

%disp(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]))

SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs([find(no_comp_do_index);airport_sst_compatibility_index(1)]) =
[];
SST_Final_Market.(['Year_',num2str(year)]).DistStMiles([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).generic_cdf([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];


end
airport_sst_compatibility_index = ismember(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,Airport_List_SST_Compatibility)==0;

sst_od = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);

end

end
end

%%
TT_Dep_Arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);

for year = SST_Year_Start:SST_Year_End

number_of_OD = length(SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID);

SST_Dep_Arr = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);

for OD = 1:number_of_OD

[od_travel_time_index,~] = ismember(TT_Dep_Arr,SST_Dep_Arr(OD));

seats = SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(OD);

flights = ceil(seats ./ acft_seat_capacity);
```

```
370
371 total_flight_hours = flights .* Travel_Times.Travel_Time(od_travel_time_index);
372
373 SST_Final_Market.(['Year_',num2str(year)]).Aircraft_Needed(OD,1) = (total_flight_hours ./ hours_per_year);
374
375 clear seats flights total_flight_hours
376 end
377
378 SST_Final_Market.(['Year_',num2str(year)]).Number_of_Passenger = SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats .*
acft_pax_load_factor;
379
380 end
381
382
383 save(([save_dir,main_delimiter,'SST_Final_Market_OverlandRange_',num2str(aircraft_range_overland),'_nm_OverwaterRange_',num2str
(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-
1)*10),'_US_Int',
date,'.mat']),'SST_Final_Market')
384 return;
385
```

```
1 function [] =
Fare_per_Mile_CDF_Adjustment_Main_Function_Int(SST_Market_Analysis_Dir,Load_Pre_Process_Dir,mach_overland,mach_overwater,
vot_limit,Int_VOT,date,aircraft_range_overland,aircraft_range_overwater)
2 %% This script generates the fare per mile CDF data from ARC 2016 - Int Market
3
4 local_disc = '';
5 main_delimiter = '\';
6 Input_Folder_Dir = ([local_disc,'..\..\..\..\..\..\SST_2020_Input']);
7 SST_Int_CDF_Dir = ([SST_Market_Analysis_Dir,main_delimiter,'Int_Market\CDF']);
8 save_dir = ([SST_Int_CDF_Dir,main_delimiter,'Output']);
9 Upload_Dir = ([Load_Pre_Process_Dir,'\VOT\Int_Market\Output']);
10 TravelTime_Upload_Dir = ([Load_Pre_Process_Dir,main_delimiter,'SST_Travel_Times\Output']);
11 Upload_PreProc_CDF_Dir = ([Load_Pre_Process_Dir,main_delimiter,'CDF\Int_Market\Output']);
12
13 %Create Output directory
14 if exist((save_dir),'dir') == 0
15
16 mkdir(([SST_Int_CDF_Dir,main_delimiter,'Output']))
17
18 end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0
19
20 addpath(([SST_Int_CDF_Dir,main_delimiter,'Output']))
21
22 Fare_per_mile_paid_by_OD_with_VOT_Adjustment_Int(Upload_Dir, Upload_PreProc_CDF_Dir, Input_Folder_Dir, TravelTime_Upload_Dir,
main_delimiter,
save_dir, mach_overland, mach_overwater, vot_limit,Int_VOT,date,aircraft_range_overland,aircraft_range_overwater)
23
24 return;
```

```
1
2 function [] = Fare_per_mile_paid_by_OD_with_VOT_Adjustment_Int(Upload_Dir, Upload_PreProc_CDF_Dir,
Input_Folder_Dir,TravelTime_Upload_Dir,
main_delimiter, save_dir, mach_overland, mach_overwater, vot_limit,Int_VOT,date,aircraft_range_overland,aircraft_range_overwater)
3
4
load(([TravelTime_Upload_Dir,main_delimiter,'Travel_Times_with_Ground_Times_OverlandRange_',num2str(aircraft_range_overland),'_nm_1
_',num2str
((mach_overland-1)*10),'M_1_',num2str((mach_overwater-1)*10),'M.mat']),'Travel_Times')
5
6 load(([Input_Folder_Dir,main_delimiter,'OD_Pairs_Year_2016.mat']),'OD_Pairs_Year_2016')
7
8 load(([Upload_Dir,main_delimiter,'Value_of_Time_Int.mat']),'Value_of_Time')
9
10
load(([Upload_PreProc_CDF_Dir,main_delimiter,'OD_Fare_Per_Mile_Paid_Premium_Original_Int.mat']),'OD_Fare_Per_Mile_Paid_Premium')
```

```
11
12 OAG_Data = OD_Pairs_Year_2016;
13 clear OD_Pairs_Year_2016
14 %%
15
16 if Int_VOT ~= 0
17
18 Value_of_Time.Value(:) = Int_VOT;
19
20 end
21
22 %%
23 %vot median value
24 index = Value_of_Time.Value <= vot_limit;
25 vot_median = round(prctile(Value_of_Time.Value(index),50),0);
26
27 %%
28 OAG_dep_arr = strcat(OAG_Data.DepAirport,'_',OAG_Data.ArrAirport);
29 TT_dep_arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);
30
31 [~,matched_index] = ismember(TT_dep_arr,OAG_dep_arr);
32
33 Travel_Times.SubSonic_Travel_Time = OAG_Data.OD_AverageElapsedTime_hrs(matched_index);
34
35 Travel_Times.Travel_Time_Saving = Travel_Times.SubSonic_Travel_Time - (Travel_Times.Travel_Time);
36
37 negative_saving = find(Travel_Times.Travel_Time_Saving < 0);
38
39 if isempty(negative_saving) == 0
40
41 Travel_Times.DepAirport(negative_saving) = [];
42 Travel_Times.ArrAirport(negative_saving) = [];
43 Travel_Times.Distance(negative_saving) = [];
44 Travel_Times.Travel_Time(negative_saving) = [];
45 Travel_Times.SubSonic_Travel_Time(negative_saving) = [];
46 end
47
48 TT_dep_arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);
49
50 length_of_OD = length(OD_Fare_Per_Mile_Paid_Premium);
51
52 for OD = 1:length_of_OD
53
54
55 % if mod(OD, 100) == 0
56 % disp([num2str(OD), '/', num2str(length_of_OD)]);
57 %
58 % end
59
60 [TT_OD_index,~] = ismember(TT_dep_arr,OD_Fare_Per_Mile_Paid_Premium(OD).OD);
61
62 if sum(TT_OD_index) ~= 0
63
64 %%
65 od_char = char(OD_Fare_Per_Mile_Paid_Premium(OD).OD);
66 origin_airport = cellstr(od_char(1:3));
67 [vot_od_index,~] = ismember(Value_of_Time.Airport,origin_airport);
68 vot_od_count = 1;
69 if sum(vot_od_index) ~= 0
70 vot_od_count = Value_of_Time.Number_of_records(vot_od_index);
71 if Value_of_Time.Value(vot_od_index) > vot_limit
72
73
74
75 Value_of_Time_od = vot_limit;
76 Final_Value_of_Time_By_OD.Origin_Median_Used_Yes1_No_0(OD,1) = 0;
```

```
77 Final_Value_of_Time_By_OD.Origin_Limit_Used_Yes1_No_0(OD,1) = 1;
78
79 else
80
81 Value_of_Time_od = round(Value_of_Time.Value(vot_od_index),0);
82 Final_Value_of_Time_By_OD.Origin_Median_Used_Yes1_No_0(OD,1) = 0;
83 Final_Value_of_Time_By_OD.Origin_Limit_Used_Yes1_No_0(OD,1) = 0;
84 end
85
86 else
87
88 Value_of_Time_od = vot_median;
89 Final_Value_of_Time_By_OD.Origin_Median_Used_Yes1_No_0(OD,1) = 1;
90 Final_Value_of_Time_By_OD.Origin_Limit_Used_Yes1_No_0(OD,1) = 0;
91 end
92 %%
93
94 destination_airport = cellstr(od_char(5:7));
95
96 [vot_do_index,~] = ismember(Value_of_Time.Airport,destination_airport);
97 vot_do_count = 1;
98 if sum(vot_do_index) ~= 0
99 vot_do_count = Value_of_Time.Number_of_records(vot_do_index);
100 if Value_of_Time.Value(vot_do_index) > vot_limit
101
102 Value_of_Time_do = vot_limit;
103 Final_Value_of_Time_By_OD.Destination_Median_Used_Yes1_No_0(OD,1) = 0;
104 Final_Value_of_Time_By_OD.Destination_Limit_Used_Yes1_No_0(OD,1) = 1;
105 else
106
107 Value_of_Time_do = round(Value_of_Time.Value(vot_do_index),0);
108 Final_Value_of_Time_By_OD.Destination_Median_Used_Yes1_No_0(OD,1) = 0;
109 Final_Value_of_Time_By_OD.Destination_Limit_Used_Yes1_No_0(OD,1) = 0;
110 end
111
112 else
113
114 Value_of_Time_do = vot_median;
115 Final_Value_of_Time_By_OD.Destination_Median_Used_Yes1_No_0(OD,1) = 1;
116 Final_Value_of_Time_By_OD.Destination_Limit_Used_Yes1_No_0(OD,1) = 0;
117 end
118
119 VOT_value = round((Value_of_Time_od * vot_od_count + Value_of_Time_do * vot_do_count) / sum([vot_od_count;vot_do_count]),0);
120
121 Final_Value_of_Time_By_OD.Origin_Airport(OD,1)= origin_airport;
122 Final_Value_of_Time_By_OD.Origin_VOT(OD,1) = Value_of_Time_od;
123 Final_Value_of_Time_By_OD.Origin_Records(OD,1) = vot_od_count;
124
125 Final_Value_of_Time_By_OD.Destination_Airport(OD,1)= destination_airport;
126 Final_Value_of_Time_By_OD.Destination_VOT(OD,1) = Value_of_Time_do;
127 Final_Value_of_Time_By_OD.Destination_Records(OD,1) = vot_do_count;
128 Final_Value_of_Time_By_OD.Weighted_Avg_VOT(OD,1) = VOT_value;
129 Final_Value_of_Time_By_OD.Weighted_Avg_VOT_Records(OD,1) = vot_od_count + vot_do_count;
130
131 %%
132
133 additional_fare_per_mile = round((VOT_value * Travel_Times.Travel_Time_Saving(TT_OD_index)) ./
Travel_Times.Distance(TT_OD_index),2);
134
135 OD_Fare_Per_Mile_Paid_Premium(OD).Unique_Fare = OD_Fare_Per_Mile_Paid_Premium(OD).Unique_Fare + additional_fare_per_mile;
136
137 end
138
139 end
140
141 save(([save_dir,main_delimiter,'OD_Fare_Per_Mile_Paid_Premium_Adjusted_OverlandRange_',num2str
```

```
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_Int',date,'.mat']),'OD_Fare_Per_Mile_Paid_Premium')
142
save(([save_dir,main_delimiter,'Final_Value_of_Time_By_OD_OverlandRange_',num2str(aircraft_range_overland),'_nm_OverwaterRange_',nu
m2str
(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-
1)*10),'_Int',date,'.
mat']),'Final_Value_of_Time_By_OD')
143
144 return;

1 function [] =
SST_Int_Forecast_Main_Function(SST_Market_Analysis_Dir,Load_Pre_Process_Dir,Airport_List_SST_Compatibility,airport_list,mach_overland,
mach_overwater,number_of_seats,FSF,hours_per_year,acft_pax_load_factor,SST_Year_Start,SST_Year_End,vot_limit,aircraft_range_overland,
aircraft_range_overwater,min_distance_statute_mile,max_distance_statute_mile,min_demand,market_share,Int_VOT,date,
run_airport_gate_compatibility_section, run_airport_runway_compatibility_section,runway_length_min_ft,airport_runway_length_data)
2 %SST Market Script
3
4 local_disc = '';
5 main_delimiter = '\';
6 Input_Folder_Dir = ([local_disc,'..\..\..\..\..\SST_2020_Input']);
7 SST_Int_Forecast_Dir = ([SST_Market_Analysis_Dir,main_delimiter,'Int_Market\SST_Forecast']);
8 save_dir = ([SST_Int_Forecast_Dir,main_delimiter,'Output']);
9 Upload_VOT_Dir = ([Load_Pre_Process_Dir,main_delimiter,'VOT\Int_Market\Output']);
10 Upload_PreProc_CDF_Dir = ([Load_Pre_Process_Dir,main_delimiter,'CDF\Int_Market\Output']);
11 Upload_CDF_Dir = ([SST_Market_Analysis_Dir,main_delimiter,'Int_Market\CDF\Output']);
12 TravelTime_Upload_Dir = ([Load_Pre_Process_Dir,main_delimiter,'SST_Travel_Times\Output']);
13
14 %Create Output directory
15 if exist((save_dir),'dir') == 0
16
17 mkdir(([SST_Int_Forecast_Dir,main_delimiter,'Output']))
18
19 end %if exist(([SST_Travel_Times_Dir,main_delimeter,'Output']),'dir') == 0
20
21 addpath(([SST_Int_Forecast_Dir,main_delimiter,'Output']))
22
23 %%
24
25 load(([Input_Folder_Dir,main_delimiter,'ICAO_Trip_Distribution_All.mat']),'Trip_Distribution_All')
26
27 load(([Input_Folder_Dir,main_delimiter,'OD_Pairs_Year_2016.mat']),'OD_Pairs_Year_2016')
28
29
load(([TravelTime_Upload_Dir,main_delimiter,'Travel_Times_with_Ground_Times_OverlandRange_',num2str(aircraft_range_overland),'_nm_1
_',num2str
((mach_overland-1)*10),'M_1_',num2str((mach_overwater-1)*10),'M.mat']),'Travel_Times')
30
31 load(([Upload_CDF_Dir,main_delimiter,'OD_Fare_Per_Mile_Paid_Premium_Adjusted_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_Int',date,'.mat']),'OD_Fare_Per_Mile_Paid_Premium')
32
33 load(([Upload_PreProc_CDF_Dir,main_delimiter,'Fare_Per_Mile_Paid_Premium_Int.mat']),'Fare_per_Mile_Paid_Premium')
34
35 load(([Upload_VOT_Dir,main_delimiter,'Value_of_Time_Int.mat']),'Value_of_Time')
36
37
load(([Input_Folder_Dir,main_delimiter,'Fare_data\',num2str(number_of_seats),'_Seats\Fare_Low_Boom_Int',date,'.mat']),'cost_per_passenge
r_mile')
38 %%
39
40 if Int_VOT ~= 0
41
42 Value_of_Time.Value(:) = Int_VOT;
43
44 end
```

```
45
46 %% Additional Parameters (calculated)
47 minimum_premium_seats = number_of_seats * 260; %1 flight per day, 260 days/year
48
49 index = Value_of_Time.Value <= vot_limit;
50 vot_median = round(prctile(Value_of_Time.Value(index),50),0);
51
52 %% FSF
53
54 fpm_reduction = 1-(100-FSF)/10 * 0.0346;
55 cost_per_passenger_mile.Over_land.fare = round(cost_per_passenger_mile.Over_land.fare .* fpm_reduction,3);
56 cost_per_passenger_mile.Over_water.fare = round(cost_per_passenger_mile.Over_water.fare .* fpm_reduction,3);
57
58 [OD_seats_percent_Premium,OD_Pairs_Year_2016,Trip_Distribution_All,generic_cdf,Fare_per_mile_cost] =
Fare_Percent_by_OD_Premium_Int
(Fare_per_Mile_Paid_Premium,
OD_Fare_Per_Mile_Paid_Premium,OD_Pairs_Year_2016,Travel_Times,Trip_Distribution_All,SST_Year_Start,SST_Year_End,
cost_per_passenger_mile,vot_median,aircraft_range_overland,aircraft_range_overwater,min_distance_statute_mile);
59
60 %%
61
62 SST_Market_Premium = SST_Int_Market_Premium(Trip_Distribution_All, OD_Pairs_Year_2016,OD_seats_percent_Premium,generic_cdf,
Fare_per_mile_cost,SST_Year_Start,SST_Year_End,min_distance_statute_mile,max_distance_statute_mile,min_demand);
63
64 %%
65
66 [SST_Market_Share_Premium] =
SST_Int_Market_Share_Premium(SST_Market_Premium,SST_Year_Start,SST_Year_End,market_share,save_dir,
main_delimiter,aircraft_range_overwater,mach_overland,mach_overwater,FSF,number_of_seats);
67
68 %%
69 Final_SST_Int_Market(SST_Market_Share_Premium,Airport_List_SST_Compatibility,airport_list, minimum_premium_seats,hours_per_year,
number_of_seats,acft_pax_load_factor,SST_Year_Start,SST_Year_End,FSF,Travel_Times,aircraft_range_overwater,save_dir,main_delimiter,ma
ch_overland,
mach_overwater,date,run_airport_gate_compatibility_section,
run_airport_runway_compatibility_section,runway_length_min_ft,airport_runway_length_data,
aircraft_range_overland);
70
71 return;

1
2 function [OD_seats_percent_Premium,OD_Pairs_Year_2016,Trip_Distribution_All,generic_cdf,Fare_per_mile_cost] =
Fare_Percent_by_OD_Premium_Int
(Fare_per_Mile_Paid_Premium,
OD_Fare_Per_Mile_Paid_Premium,OD_Pairs_Year_2016,Travel_Times,Trip_Distribution_All,SST_Year_Start,SST_Year_End,
cost_per_passenger_mile,vot_median,aircraft_range_overland,aircraft_range_overwater,min_distance_statute_mile)
3
4 delete_short_distance_od = find(OD_Pairs_Year_2016.DistStMiles < min_distance_statute_mile);%sm.
5
6 OD_Pairs_Year_2016.Carrier_Name(delete_short_distance_od) = [];
7 OD_Pairs_Year_2016.DepAirport_UniqueID(delete_short_distance_od) = [];
8 OD_Pairs_Year_2016.DepAirport(delete_short_distance_od) = [];
9 OD_Pairs_Year_2016.DepIATACtry(delete_short_distance_od) = [];
10 OD_Pairs_Year_2016.DepReg(delete_short_distance_od) = [];
11 OD_Pairs_Year_2016.ArrAirport_UniqueID(delete_short_distance_od) = [];
12 OD_Pairs_Year_2016.ArrAirport(delete_short_distance_od) = [];
13 OD_Pairs_Year_2016.ArrIATACtry(delete_short_distance_od) = [];
14 OD_Pairs_Year_2016.ArrReg(delete_short_distance_od) = [];
15 OD_Pairs_Year_2016.InternationalDomestic(delete_short_distance_od) = [];
16 OD_Pairs_Year_2016.Total_Seats(delete_short_distance_od) = [];
17 OD_Pairs_Year_2016.Total_FstSeats(delete_short_distance_od) = [];
18 OD_Pairs_Year_2016.Total_BusSeats(delete_short_distance_od) = [];
19 OD_Pairs_Year_2016.Total_EcoSeats(delete_short_distance_od) = [];
20 OD_Pairs_Year_2016.Total_Frequency(delete_short_distance_od) = [];
21 OD_Pairs_Year_2016.OD_AverageElapsedTime_hrs(delete_short_distance_od) = [];
22 OD_Pairs_Year_2016.DistStMiles(delete_short_distance_od) = [];
```

```
23 OD_Pairs_Year_2016.AcftData(delete_short_distance_od) = [];
24
25 for year = SST_Year_Start:SST_Year_End
26
27 Trip_Distribution_All.(['Year_',num2str(year)]).DepAirport_UniqueID(delete_short_distance_od) = [];
28 Trip_Distribution_All.(['Year_',num2str(year)]).ArrAirport_UniqueID(delete_short_distance_od) = [];
29 Trip_Distribution_All.(['Year_',num2str(year)]).Seats(delete_short_distance_od) = [];
30 end
31 %%
32 length_of_data = length(OD_Pairs_Year_2016.DepAirport);
33
34 Dep_Arr_All = strcat(OD_Pairs_Year_2016.DepAirport,'_',OD_Pairs_Year_2016.ArrAirport);
35
36 length_of_OD_fare = length(OD_Fare_Per_Mile_Paid_Premium);
37 Fare_Dep_Arr_All = cell(length_of_OD_fare,1);
38
39 for fare_od = 1:length_of_OD_fare
40
41 Fare_Dep_Arr_All(fare_od) = OD_Fare_Per_Mile_Paid_Premium(fare_od).OD;
42 end
43
44 Travel_Times_Dep_Arr_All = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);
45 OD_seats_percent_Premium = zeros(length_of_data,1);
46 generic_cdf = zeros(length_of_data,1);
47
48 for OD = 1:length_of_data
49
50 clear aircraft_range
51
52 [od_index,~] = ismember(Fare_Dep_Arr_All,Dep_Arr_All(OD));
53
54 od_distance = round((OD_Pairs_Year_2016.DistStMiles(OD)) * 0.868976,0); %st to nm conversion
55
56 [travel_time_od_index,~] = ismember(Travel_Times_Dep_Arr_All,Dep_Arr_All(OD));
57
58 if sum(travel_time_od_index) ~= 0
59
60 if Travel_Times.Overland_Percent(travel_time_od_index,1) > 0.25
61
62 aircraft_range = aircraft_range_overland;
63
64 else
65
66 aircraft_range = aircraft_range_overwater;
67 end
68
69 if od_distance > aircraft_range
70
71 while od_distance > aircraft_range
72
73 od_distance = ceil(od_distance/2);
74 end
75 end
76
77 if Travel_Times.Overland_Percent(travel_time_od_index) > 0.25
78
79 %overland
80
81 if od_distance >= max(cost_per_passenger_mile.Over_land.distance)
82
83 Fare_per_Mile_Paid_Threshold_Premium = min(cost_per_passenger_mile.Over_land.fare);
84
85 elseif od_distance <= min(cost_per_passenger_mile.Over_land.distance)
86
87 Fare_per_Mile_Paid_Threshold_Premium = max(cost_per_passenger_mile.Over_land.fare);
88
```

```
89 else
90
91 %interpolate bewteen values
92
93 difference_in_value = od_distance - cost_per_passenger_mile.Over_land.distance;
94
95 lower_bound_index = max(find(difference_in_value>0));
96
97 upper_bound_index = lower_bound_index + 1;
98
99 Fare_per_Mile_Paid_Threshold_Premium = cost_per_passenger_mile.Over_land.fare(lower_bound_index) - ((cost_per_passenger_mile.
Over_land.distance(lower_bound_index) - od_distance) / (cost_per_passenger_mile.Over_land.distance(lower_bound_index) -
cost_per_passenger_mile.
Over_land.distance(upper_bound_index))) * (cost_per_passenger_mile.Over_land.fare(lower_bound_index) -
cost_per_passenger_mile.Over_land.fare
(upper_bound_index));
100
101 end
102
103 else
104
105 %overwater
106
107
108 if od_distance >= max(cost_per_passenger_mile.Over_water.distance)
109
110 Fare_per_Mile_Paid_Threshold_Premium = min(cost_per_passenger_mile.Over_water.fare);
111
112 elseif od_distance <= min(cost_per_passenger_mile.Over_water.distance)
113
114 Fare_per_Mile_Paid_Threshold_Premium = max(cost_per_passenger_mile.Over_water.fare);
115
116 else
117
118 %interpolate bewteen values
119
120 difference_in_value = od_distance - cost_per_passenger_mile.Over_water.distance;
121
122 lower_bound_index = max(find(difference_in_value>0));
123
124 upper_bound_index = lower_bound_index + 1;
125
126 Fare_per_Mile_Paid_Threshold_Premium = cost_per_passenger_mile.Over_water.fare(lower_bound_index) - ((cost_per_passenger_mile.
Over_water.distance(lower_bound_index) - od_distance) / (cost_per_passenger_mile.Over_water.distance(lower_bound_index) -
cost_per_passenger_mile.
Over_water.distance(upper_bound_index))) * (cost_per_passenger_mile.Over_water.fare(lower_bound_index) -
cost_per_passenger_mile.Over_water.fare
(upper_bound_index));
127
128 end
129
130 end
131
132 Fare_per_Mile_Paid_Threshold_Premium = Fare_per_Mile_Paid_Threshold_Premium * 0.868976; %convert $/nm. to $/sm.
133
134 if sum(od_index) ~= 0
135
136 if min(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare) > Fare_per_Mile_Paid_Threshold_Premium
137
138 OD_seats_percent_Premium(OD) = 1.0;
139
140 else
141
142 if max(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare) >= Fare_per_Mile_Paid_Threshold_Premium
143
144 if sum(ismember(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare,Fare_per_Mile_Paid_Threshold_Premium)) == 1
```

```
145
146 [fare_index,~] = ismember(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare,Fare_per_Mile_Paid_Threshold_Premium);
147
148 OD_seats_percent_Premium(OD) = 1 - OD_Fare_Per_Mile_Paid_Premium(od_index).Cummulative_Count(fare_index);
149
150 else
151
152 %interpolate bewteen values
153
154 difference_in_value = Fare_per_Mile_Paid_Threshold_Premium - OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare;
155
156 lower_bound_index = max(find(difference_in_value>0));
157
158 upper_bound_index = lower_bound_index + 1;
159
160 OD_seats_percent_Premium(OD) = 1 - (OD_Fare_Per_Mile_Paid_Premium(od_index).Cummulative_Count(lower_bound_index) -
((OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare(lower_bound_index) - Fare_per_Mile_Paid_Threshold_Premium) ./
(OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare(lower_bound_index) - OD_Fare_Per_Mile_Paid_Premium(od_index).Unique_Fare
(upper_bound_index))) .* (OD_Fare_Per_Mile_Paid_Premium(od_index).Cummulative_Count(lower_bound_index) -
OD_Fare_Per_Mile_Paid_Premium
(od_index).Cummulative_Count(upper_bound_index)));
161 end
162
163 else
164
165 OD_seats_percent_Premium(OD) = 0;
166
167 end
168
169 end
170
171 else
172
173 generic_cdf(OD,1) = 1;
174
175 [od_index_2,~] = ismember(Travel_Times_Dep_Arr_All,Dep_Arr_All(OD));
176
177 if sum(od_index_2) ~= 0
178
179 travel_time_saving = OD_Pairs_Year_2016.OD_AverageElapsedTime_hrs(OD) - Travel_Times.Travel_Time(od_index_2);
180
181 if travel_time_saving > 0
182
183 additional_fare = (vot_median * travel_time_saving) ./ Travel_Times.Distance(od_index_2);
184
185 Fare_per_Mile_Paid_Premium_adjusted.Fare = round((Fare_per_Mile_Paid_Premium.Fare + additional_fare),2);
186 Fare_per_Mile_Paid_Premium_adjusted.Cumulative = Fare_per_Mile_Paid_Premium.Cumulative;
187 %
188
189 %%
190 if min(Fare_per_Mile_Paid_Premium_adjusted.Fare) > Fare_per_Mile_Paid_Threshold_Premium
191
192 OD_seats_percent_Premium(OD) = 1.0;
193
194 else
195
196 if max(Fare_per_Mile_Paid_Premium_adjusted.Fare) >= Fare_per_Mile_Paid_Threshold_Premium
197
198 if sum(ismember(Fare_per_Mile_Paid_Premium_adjusted.Fare,Fare_per_Mile_Paid_Threshold_Premium)) == 1
199
200 [fare_index,~] = ismember(Fare_per_Mile_Paid_Premium_adjusted.Fare,Fare_per_Mile_Paid_Threshold_Premium);
201
202 OD_seats_percent_Premium(OD) = 1 - Fare_per_Mile_Paid_Premium_adjusted.Cumulative(fare_index);
203
204 else
205
```

```
206 %interpolate bewteen values
207
208 difference_in_value = Fare_per_Mile_Paid_Threshold_Premium - Fare_per_Mile_Paid_Premium_adjusted.Fare;
209
210 lower_bound_index = max(find(difference_in_value>0));
211
212 upper_bound_index = lower_bound_index + 1;
213
214 OD_seats_percent_Premium(OD) = 1 - (Fare_per_Mile_Paid_Premium_adjusted.Cumulative(lower_bound_index) -
((Fare_per_Mile_Paid_Premium_adjusted.Fare(lower_bound_index) - Fare_per_Mile_Paid_Threshold_Premium) ./
(Fare_per_Mile_Paid_Premium_adjusted.
Fare(lower_bound_index) - Fare_per_Mile_Paid_Premium_adjusted.Fare(upper_bound_index))) .*
(Fare_per_Mile_Paid_Premium_adjusted.Cumulative
(lower_bound_index) - Fare_per_Mile_Paid_Premium_adjusted.Cumulative(upper_bound_index)));
215 end
216
217 else
218
219 OD_seats_percent_Premium(OD) = 0;
220
221 end
222
223 end
224 %%
225
226 else
227
228 [index_Premium,~] = ismember(Fare_per_Mile_Paid_Premium.Fare,Fare_per_Mile_Paid_Threshold_Premium);
229
230 if sum(index_Premium) ~= 0
231
232 Premium_seats_percent = 1 - Fare_per_Mile_Paid_Premium.Cumulative(index_Premium);
233 else
234 difference_in_value = Fare_per_Mile_Paid_Threshold_Premium - Fare_per_Mile_Paid_Premium.Fare;
235
236 lower_bound_index = max(find(difference_in_value>0));
237
238 upper_bound_index = lower_bound_index + 1;
239
240 Premium_seats_percent = 1 - (Fare_per_Mile_Paid_Premium.Cumulative(lower_bound_index) - ((Fare_per_Mile_Paid_Premium.Fare
(lower_bound_index) - Fare_per_Mile_Paid_Threshold_Premium) ./ (Fare_per_Mile_Paid_Premium.Fare(lower_bound_index) -
Fare_per_Mile_Paid_Premium.
Fare(upper_bound_index))) .* (Fare_per_Mile_Paid_Premium.Cumulative(lower_bound_index) - Fare_per_Mile_Paid_Premium.Cumulative
(upper_bound_index)));
241 end
242 % OD_premium_seats_percent_NonLowBoom(OD) = premium_seats_percent_NonLowBoom;
243 OD_seats_percent_Premium(OD) = Premium_seats_percent;
244
245 end
246
247 else
248
249 [index_Premium,~] = ismember(Fare_per_Mile_Paid_Premium.Fare,Fare_per_Mile_Paid_Threshold_Premium);
250
251 if sum(index_Premium) ~= 0
252
253 Premium_seats_percent = 1 - Fare_per_Mile_Paid_Premium.Cumulative(index_Premium);
254 else
255 difference_in_value = Fare_per_Mile_Paid_Threshold_Premium - Fare_per_Mile_Paid_Premium.Fare;
256
257 lower_bound_index = max(find(difference_in_value>0));
258
259 upper_bound_index = lower_bound_index + 1;
260
261 Premium_seats_percent = 1 - (Fare_per_Mile_Paid_Premium.Cumulative(lower_bound_index) - ((Fare_per_Mile_Paid_Premium.Fare
```

```
(lower_bound_index) - Fare_per_Mile_Paid_Threshold_Premium) ./ (Fare_per_Mile_Paid_Premium.Fare(lower_bound_index) -
Fare_per_Mile_Paid_Premium.
Fare(upper_bound_index))) .* (Fare_per_Mile_Paid_Premium.Cumulative(lower_bound_index) - Fare_per_Mile_Paid_Premium.Cumulative
(upper_bound_index)));
262 end
263
264 OD_seats_percent_Premium(OD) = Premium_seats_percent;
265
266 end
267
268 end
269
270 Fare_per_mile_cost(OD,1) = Fare_per_Mile_Paid_Threshold_Premium;
271
272 end
273 end
274
275 return
276
277
```

```
1 function SST_Market_Premium = SST_Int_Market_Premium(Trip_Distribution_All,
OD_Pairs_Year_2016,OD_seats_percent_Premium,generic_cdf,
Fare_per_mile_cost,SST_Year_Start,SST_Year_End,min_distance_statute_mile,max_distance_statute_mile,min_demand)
2
3 OAG2016 = OD_Pairs_Year_2016;
4 OAG2016.Premium_Seats = OAG2016.Total_FstSeats + OAG2016.Total_BusSeats;
5 OAG2016.OD_seats_percent_Premium = OD_seats_percent_Premium;
6 OAG2016.Fare_per_mile_cost = Fare_per_mile_cost;
7
8 clear OD_Pairs_Year_2016
9 All_OD_Pair_Data = strcat(OAG2016.DepAirport,'_',OAG2016.ArrAirport);
10
11 Dep = char(Trip_Distribution_All.Year_2017.DepAirport_UniqueID);
12 Dep = Dep(:,1:3);
13 Dep = cellstr(Dep);
14
15 Arr = char(Trip_Distribution_All.Year_2017.ArrAirport_UniqueID);
16 Arr = Arr(:,1:3);
17 Arr = cellstr(Arr);
18
19 All_Trip_Distribution = strcat(Dep,'_',Arr);
20
21 OD_Pair_Index = ismember(All_OD_Pair_Data,All_Trip_Distribution);
22
23 delete_record = find(OD_Pair_Index == 0);
24 OAG2016.Carrier_Name(delete_record) = [];
25 OAG2016.DepAirport_UniqueID(delete_record) = [];
26 OAG2016.DepAirport(delete_record) = [];
27 OAG2016.DepIATACtry(delete_record) = [];
28 OAG2016.DepReg(delete_record) = [];
29 OAG2016.ArrAirport_UniqueID(delete_record) = [];
30 OAG2016.ArrAirport(delete_record) = [];
31 OAG2016.ArrIATACtry(delete_record) = [];
32 OAG2016.ArrReg(delete_record) = [];
33 OAG2016.InternationalDomestic(delete_record) = [];
34 OAG2016.Total_Seats(delete_record) = [];
35 OAG2016.Premium_Seats(delete_record) = [];
36 OAG2016.Total_FstSeats(delete_record) = [];
37 OAG2016.Total_BusSeats(delete_record) = [];
38 OAG2016.Total_EcoSeats(delete_record) = [];
39 OAG2016.Total_Frequency(delete_record) = [];
40 OAG2016.OD_AverageElapsedTime_hrs(delete_record) = [];
41 OAG2016.DistStMiles(delete_record) = [];
42 OAG2016.AcftData(delete_record) = [];
43
```

```
44 OAG2016.OD_seats_percent_Premium(delete_record) = [];
45 OAG2016.Fare_per_mile_cost(delete_record) = [];
46 OAG2016.Percent = OAG2016.Premium_Seats ./ OAG2016.Total_Seats;
47
48 %%
49
50 for year = SST_Year_Start:SST_Year_End
51
52 SST.(['Year_',num2str(year)]).DepAirport_UniqueID = OAG2016.DepAirport_UniqueID;
53 SST.(['Year_',num2str(year)]).DepAirport = OAG2016.DepAirport;
54 SST.(['Year_',num2str(year)]).DepIATACtry = OAG2016.DepIATACtry;
55 SST.(['Year_',num2str(year)]).DepReg = OAG2016.DepReg;
56 SST.(['Year_',num2str(year)]).ArrAirport_UniqueID = OAG2016.ArrAirport_UniqueID;
57 SST.(['Year_',num2str(year)]).ArrAirport = OAG2016.ArrAirport;
58 SST.(['Year_',num2str(year)]).ArrIATACtry = OAG2016.ArrIATACtry;
59 SST.(['Year_',num2str(year)]).ArrReg = OAG2016.ArrReg;
60 SST.(['Year_',num2str(year)]).InternationalDomestic = OAG2016.InternationalDomestic;
61
62 SST.(['Year_',num2str(year)]).Total_Seats = Trip_Distribution_All.(['Year_',num2str(year)]).Seats;
63 SST.(['Year_',num2str(year)]).Premium_Seats = ceil(Trip_Distribution_All.(['Year_',num2str(year)]).Seats .* OAG2016.Percent);
64
65 SST.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs = OAG2016.OD_AverageElapsedTime_hrs;
66 SST.(['Year_',num2str(year)]).DistStMiles = OAG2016.DistStMiles;
67 SST.(['Year_',num2str(year)]).OD_premium_seats_percent = OAG2016.OD_seats_percent_Premium;
68 SST.(['Year_',num2str(year)]).Fare_per_mile_cost = OAG2016.Fare_per_mile_cost;
69 SST.(['Year_',num2str(year)]).generic_cdf = generic_cdf;
70
71 end
72
73 %%
74
75 SST_Market_Premium = SST;
76
77 for year = SST_Year_Start:SST_Year_End
78
79 max_distance_index = find(SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles > max_distance_statute_mile);
80
81 if isempty(max_distance_index) == 0
82
83 SST_Market_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID(max_distance_index) = [];
84 SST_Market_Premium.(['Year_',num2str(year)]).DepAirport(max_distance_index) = [];
85 SST_Market_Premium.(['Year_',num2str(year)]).DepIATACtry(max_distance_index) = [];
86 SST_Market_Premium.(['Year_',num2str(year)]).DepReg(max_distance_index) = [];
87 SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID(max_distance_index) = [];
88 SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport(max_distance_index) = [];
89 SST_Market_Premium.(['Year_',num2str(year)]).ArrIATACtry(max_distance_index) = [];
90 SST_Market_Premium.(['Year_',num2str(year)]).ArrReg(max_distance_index) = [];
91 SST_Market_Premium.(['Year_',num2str(year)]).InternationalDomestic(max_distance_index) = [];
92
93 SST_Market_Premium.(['Year_',num2str(year)]).Total_Seats(max_distance_index) = [];
94 SST_Market_Premium.(['Year_',num2str(year)]).Premium_Seats(max_distance_index) = [];
95
96 SST_Market_Premium.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(max_distance_index) = [];
97 SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles(max_distance_index) = [];
98 SST_Market_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent(max_distance_index) = [];
99 SST_Market_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost(max_distance_index) = [];
100 SST_Market_Premium.(['Year_',num2str(year)]).generic_cdf(max_distance_index) = [];
101
102 end
103
104 clear max_distance_index
105
106 min_distance_index = find(SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles < min_distance_statute_mile);
107
108 if isempty(min_distance_index) == 0
109
```

```
110 SST_Market_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID(min_distance_index) = [];
111 SST_Market_Premium.(['Year_',num2str(year)]).DepAirport(min_distance_index) = [];
112 SST_Market_Premium.(['Year_',num2str(year)]).DepIATACtry(min_distance_index) = [];
113 SST_Market_Premium.(['Year_',num2str(year)]).DepReg(min_distance_index) = [];
114 SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID(min_distance_index) = [];
115 SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport(min_distance_index) = [];
116 SST_Market_Premium.(['Year_',num2str(year)]).ArrIATACtry(min_distance_index) = [];
117 SST_Market_Premium.(['Year_',num2str(year)]).ArrReg(min_distance_index) = [];
118 SST_Market_Premium.(['Year_',num2str(year)]).InternationalDomestic(min_distance_index) = [];
119
120 SST_Market_Premium.(['Year_',num2str(year)]).Total_Seats(min_distance_index) = [];
121 SST_Market_Premium.(['Year_',num2str(year)]).Premium_Seats(min_distance_index) = [];
122
123 SST_Market_Premium.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(min_distance_index) = [];
124 SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles(min_distance_index) = [];
125 SST_Market_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent(min_distance_index) = [];
126 SST_Market_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost(min_distance_index) = [];
127 SST_Market_Premium.(['Year_',num2str(year)]).generic_cdf(min_distance_index) = [];
128
129 end
130
131 clear min_distance_index
132
133 min_demnad_index = find(SST_Market_Premium.(['Year_',num2str(year)]).Total_Seats < min_demand);
134
135 if isempty(min_demnad_index) == 0
136
137 SST_Market_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID(min_demnad_index) = [];
138 SST_Market_Premium.(['Year_',num2str(year)]).DepAirport(min_demnad_index) = [];
139 SST_Market_Premium.(['Year_',num2str(year)]).DepIATACtry(min_demnad_index) = [];
140 SST_Market_Premium.(['Year_',num2str(year)]).DepReg(min_demnad_index) = [];
141 SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID(min_demnad_index) = [];
142 SST_Market_Premium.(['Year_',num2str(year)]).ArrAirport(min_demnad_index) = [];
143 SST_Market_Premium.(['Year_',num2str(year)]).ArrIATACtry(min_demnad_index) = [];
144 SST_Market_Premium.(['Year_',num2str(year)]).ArrReg(min_demnad_index) = [];
145 SST_Market_Premium.(['Year_',num2str(year)]).InternationalDomestic(min_demnad_index) = [];
146
147 SST_Market_Premium.(['Year_',num2str(year)]).Total_Seats(min_demnad_index) = [];
148 SST_Market_Premium.(['Year_',num2str(year)]).Premium_Seats(min_demnad_index) = [];
149
150 SST_Market_Premium.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(min_demnad_index) = [];
151 SST_Market_Premium.(['Year_',num2str(year)]).DistStMiles(min_demnad_index) = [];
152 SST_Market_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent(min_demnad_index) = [];
153 SST_Market_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost(min_demnad_index) = [];
154 SST_Market_Premium.(['Year_',num2str(year)]).generic_cdf(min_demnad_index) = [];
155 end
156
157 clear min_demnad_index
158 end
159
160
161 return;
162

1
2 function [SST_Market_Share_Premium] =
SST_Int_Market_Share_Premium(SST_Market_Premium,SST_Year_Start,SST_Year_End,market_share,save_dir,
main_delimiter,aircraft_range_overwater,mach_overland,mach_overwater,FSF,number_of_seats)
3
4 SST_Market_Share_Premium = SST_Market_Premium;
5
6 for year = SST_Year_Start:SST_Year_End
7
8 SST_Market_Share_Premium.(['Year_',num2str(year)]).SST_Premium_Seats = ceil(((SST_Market_Share_Premium.(['Year_',num2str(year)]).
Premium_Seats .* SST_Market_Share_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent) .* market_share));
9
```

```
10 end
11
12
%save(([save_dir,main_delimiter,'ICAO_SST_Market_Share_Premium_',num2str(aircraft_range_overwater),'nm_1_',num2str((mach_overland-
1)
*10),'M_1_',num2str((mach_overwater-
1)*10),'M_',num2str(number_of_seats),'Seater_FSF_',num2str(FSF),'_Int']),'SST_Market_Share_Premium')
13
14 return;
15

1
2 function [] =
Final_SST_Int_Market(SST_Market_Share_Premium,Airport_List_SST_Compatibility,airport_list,minimum_premium_seats,hours_per_year,
number_of_seats,acft_pax_load_factor,SST_Year_Start,SST_Year_End,FSF,Travel_Times,aircraft_range_overwater,save_dir,main_delimiter,ma
ch_overland,
mach_overwater,date,run_airport_gate_compatibility_section,
run_airport_runway_compatibility_section,runway_length_min_ft,airport_runway_length_data,
aircraft_range_overland)
3
4 acft_seat_capacity = number_of_seats;
5
6 for year = SST_Year_Start:SST_Year_End
7
8 [~, US_dep] = ismember(SST_Market_Share_Premium.(['Year_',num2str(year)]).DepIATACtry,'US');
9 [~, US_arr] = ismember(SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrIATACtry,'US');
10 US_dep_US_arr = US_dep + US_arr;
11 non_int_us_index = find(US_dep_US_arr ~= 0);
12
13 SST_Market_Share_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID(non_int_us_index) = [];
14 SST_Market_Share_Premium.(['Year_',num2str(year)]).DepAirport(non_int_us_index) = [];
15 SST_Market_Share_Premium.(['Year_',num2str(year)]).DepIATACtry(non_int_us_index) = [];
16 SST_Market_Share_Premium.(['Year_',num2str(year)]).DepReg(non_int_us_index) = [];
17 SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID(non_int_us_index) = [];
18 SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrAirport(non_int_us_index) = [];
19 SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrIATACtry(non_int_us_index) = [];
20 SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrReg(non_int_us_index) = [];
21 SST_Market_Share_Premium.(['Year_',num2str(year)]).InternationalDomestic(non_int_us_index) = [];
22 SST_Market_Share_Premium.(['Year_',num2str(year)]).Total_Seats(non_int_us_index) = [];
23 SST_Market_Share_Premium.(['Year_',num2str(year)]).Premium_Seats(non_int_us_index) = [];
24 SST_Market_Share_Premium.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(non_int_us_index) = [];
25 SST_Market_Share_Premium.(['Year_',num2str(year)]).DistStMiles(non_int_us_index) = [];
26 SST_Market_Share_Premium.(['Year_',num2str(year)]).OD_premium_seats_percent(non_int_us_index) = [];
27 SST_Market_Share_Premium.(['Year_',num2str(year)]).SST_Premium_Seats(non_int_us_index) = [];
28 SST_Market_Share_Premium.(['Year_',num2str(year)]).generic_cdf(non_int_us_index) = [];
29 SST_Market_Share_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost(non_int_us_index) = [];
30
31 end
32
33 %%
34
35 for year = SST_Year_Start:SST_Year_End
36
37 SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID =
SST_Market_Share_Premium.(['Year_',num2str(year)]).DepAirport_UniqueID;
38 SST_Final_Market.(['Year_',num2str(year)]).DepAirport = SST_Market_Share_Premium.(['Year_',num2str(year)]).DepAirport;
39 SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry = SST_Market_Share_Premium.(['Year_',num2str(year)]).DepIATACtry;
40 SST_Final_Market.(['Year_',num2str(year)]).DepReg = SST_Market_Share_Premium.(['Year_',num2str(year)]).DepReg;
41
42 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID =
SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrAirport_UniqueID;
43 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport = SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrAirport;
44 SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry = SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrIATACtry;
45 SST_Final_Market.(['Year_',num2str(year)]).ArrReg = SST_Market_Share_Premium.(['Year_',num2str(year)]).ArrReg;
46
47 SST_Final_Market.(['Year_',num2str(year)]).Total_Seats = SST_Market_Share_Premium.(['Year_',num2str(year)]).Total_Seats;
```

```
48
49 SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats = SST_Market_Share_Premium.(['Year_',num2str(year)]).Premium_Seats;
50
51 SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs = SST_Market_Share_Premium.(['Year_',num2str(year)]).
OD_AverageElapsedTime_hrs;
52 SST_Final_Market.(['Year_',num2str(year)]).DistStMiles = SST_Market_Share_Premium.(['Year_',num2str(year)]).DistStMiles;
53
54 SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats =
SST_Market_Share_Premium.(['Year_',num2str(year)]).SST_Premium_Seats;
55 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats = SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats; %+
SST_Final_Market.
(['Year_',num2str(year)]).SST_Economy_Premium_Seats;
56 SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost =
SST_Market_Share_Premium.(['Year_',num2str(year)]).Fare_per_mile_cost;
57 SST_Final_Market.(['Year_',num2str(year)]).generic_cdf = SST_Market_Share_Premium.(['Year_',num2str(year)]).generic_cdf;
58
59
60 end
61
62 %%
63
64 for year = SST_Year_Start:SST_Year_End
65
66 od_with_min_demand_index = find(SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats >= minimum_premium_seats);
67
68 o_d =
strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(od_with_min_demand_index),'_',SST_Final_Market.(['Year_',num2str(year)]).
ArrAirport(od_with_min_demand_index));
69 d_o =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(od_with_min_demand_index),'_',SST_Final_Market.(['Year_',num2str(year)]).
DepAirport(od_with_min_demand_index));
70 od_do = [o_d;d_o];
71 od_do_unique = unique(od_do);
72
73 SST_Final_Market_OD =
strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);
74
75 delete_index = ismember(SST_Final_Market_OD,od_do_unique) == 0;
76
77 SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID(delete_index) = [];
78 SST_Final_Market.(['Year_',num2str(year)]).DepAirport(delete_index) = [];
79 SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry(delete_index) = [];
80 SST_Final_Market.(['Year_',num2str(year)]).DepReg(delete_index) = [];
81 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID(delete_index) = [];
82 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(delete_index) = [];
83 SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry(delete_index) = [];
84 SST_Final_Market.(['Year_',num2str(year)]).ArrReg(delete_index) = [];
85 SST_Final_Market.(['Year_',num2str(year)]).Total_Seats(delete_index) = [];
86 SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats(delete_index) = [];
87 SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(delete_index) = [];
88 SST_Final_Market.(['Year_',num2str(year)]).DistStMiles(delete_index) = [];
89 SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats(delete_index) = [];
90 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(delete_index) = [];
91 SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost (delete_index) = [];
92 SST_Final_Market.(['Year_',num2str(year)]).generic_cdf(delete_index) = [];
93
94 end
95
96 %% Airport Gate Compatibility
97
98 if run_airport_gate_compatibility_section == 1
99
100 for year = SST_Year_Start:SST_Year_End
101 clear od_dist
102
103 airport_sst_compatibility_index = ismember(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,Airport_List_SST_Compatibility)==0;
```

```
104
105 sst_od = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);
106
107 while sum(airport_sst_compatibility_index) ~= 0
108
109 airport_sst_compatibility_index = find(airport_sst_compatibility_index);
110
111 no_comp_do =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)),'_',SST_Final_Market.(['Year_',
num2str(year)]).DepAirport(airport_sst_compatibility_index(1)));
112 [no_comp_do_index,~] = ismember(sst_od,no_comp_do);
113 length_of_od = length(SST_Final_Market.(['Year_',num2str(year)]).DepAirport);
114
115 [o_match_index] =
ismember(airport_list.Airport_IDs,SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)));
116 lat_o = airport_list.Apt_Lat(o_match_index);
117 lon_o = airport_list.Apt_Lon(o_match_index);
118
119 od_dist = zeros(length_of_od,1);
120 for airport = 1:length_of_od
121
122 [d_match_index] = ismember(airport_list.Airport_IDs,SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport));
123
124 lat_d = airport_list.Apt_Lat(d_match_index);
125 lon_d = airport_list.Apt_Lon(d_match_index);
126
127 od_dist(airport,1) = ceil(deg2sm(distance(lat_o,lon_o,lat_d,lon_d)));
128 end
129
130 zero_dist = od_dist == 0;
131
132 od_dist(zero_dist) = 999999;
133
134 min_dist = min(od_dist);
135
136 if min_dist <= 30
137
138 redirect_airport_index = find(od_dist==min_dist);
139
140 od_redirect_verification =
strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)),'_',SST_Final_Market.(['Year_',
num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)));
141 do_redirect_verification =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)),'_',SST_Final_Market.
(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)));
142
143
144 od_redirect_exist = ismember(sst_od,od_redirect_verification);
145 do_redirect_exist = ismember(sst_od,do_redirect_verification);
146
147 if sum(od_redirect_exist) == 0
148 %disp(year)
149 %disp('change')
150 %disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)))
151 %disp('to')
152 %disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)))
153
154 SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)) = SST_Final_Market.(['Year_',num2str(year)]).
DepAirport(redirect_airport_index(1));
155 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(no_comp_do_index) = SST_Final_Market.(['Year_',num2str(year)]).DepAirport
(redirect_airport_index(1));
156
157 else
158 %disp('add from')
159 %disp(strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]),'_',
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)])))
```

```
%disp('to')

%disp(strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(od_redirect_exist);find(do_redirect_exist)]),'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(od_redirect_exist);find(do_redirect_exist)])))

SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(od_redirect_exist) = SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(od_redirect_exist) + SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(airport_sst_compatibility_index(1));
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(do_redirect_exist) = SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(do_redirect_exist) + SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(no_comp_do_index);


SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DistStMiles([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).generic_cdf([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];

end


else

%disp(year)
%disp('delete')

%disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]))

%disp(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]))

SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DepReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).DistStMiles([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).generic_cdf([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];


end
airport_sst_compatibility_index = ismember(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,Airport_List_SST_Compatibility)==0;

sst_od = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);
```

```
218
219 end
220
221 end
222 end
223
224 %% Runway Length Compatibility
225 if run_airport_runway_compatibility_section == 1
226
227 runway_length_index = airport_runway_length_data.runway_ft >= runway_length_min_ft;
228 Airport_List_SST_Compatibility = airport_runway_length_data.airport(runway_length_index);
229
230 for year = SST_Year_Start:SST_Year_End
231 clear od_dist
232
233 airport_sst_compatibility_index = ismember(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,Airport_List_SST_Compatibility)==0;
234
235 sst_od = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);
236
237 while sum(airport_sst_compatibility_index) ~= 0
238
239 airport_sst_compatibility_index = find(airport_sst_compatibility_index);
240
241 no_comp_do =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)),'_',SST_Final_Market.(['Year_',
num2str(year)]).DepAirport(airport_sst_compatibility_index(1)));
242 [no_comp_do_index,~] = ismember(sst_od,no_comp_do);
243 length_of_od = length(SST_Final_Market.(['Year_',num2str(year)]).DepAirport);
244
245 [o_match_index] =
ismember(airport_list.Airport_IDs,SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)));
246 lat_o = airport_list.Apt_Lat(o_match_index);
247 lon_o = airport_list.Apt_Lon(o_match_index);
248
249 od_dist = zeros(length_of_od,1);
250 for airport = 1:length_of_od
251
252 [d_match_index] = ismember(airport_list.Airport_IDs,SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport));
253
254 lat_d = airport_list.Apt_Lat(d_match_index);
255 lon_d = airport_list.Apt_Lon(d_match_index);
256
257 od_dist(airport,1) = ceil(deg2sm(distance(lat_o,lon_o,lat_d,lon_d)));
258 end
259
260 zero_dist = od_dist == 0;
261
262 od_dist(zero_dist) = 999999;
263
264 min_dist = min(od_dist);
265
266 if min_dist <= 30
267
268 redirect_airport_index = find(od_dist==min_dist);
269
270 od_redirect_verification =
strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)),'_',SST_Final_Market.(['Year_',
num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)));
271 do_redirect_verification =
strcat(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(airport_sst_compatibility_index(1)),'_',SST_Final_Market.
(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)));
272
273
274 od_redirect_exist = ismember(sst_od,od_redirect_verification);
275 do_redirect_exist = ismember(sst_od,do_redirect_verification);
276
```

```
277 if sum(od_redirect_exist) == 0
278 %disp(year)
279 %disp('line149')
280 %disp('change')
281 %disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)))
282 %disp('to')
283 %disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport(redirect_airport_index(1)))
284
285 SST_Final_Market.(['Year_',num2str(year)]).DepAirport(airport_sst_compatibility_index(1)) = SST_Final_Market.(['Year_',num2str(year)]).
DepAirport(redirect_airport_index(1));
286 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(no_comp_do_index) = SST_Final_Market.(['Year_',num2str(year)]).DepAirport
(redirect_airport_index(1));
287
288 else
289
290 %disp('add from')
291 %disp(strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]),'_',
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)])))
292 %disp('to')
293
%disp(strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(od_redirect_exist);find(do_redirect_exist)]),'_',SST_Final_Market.
(['Year_',num2str(year)]).ArrAirport([find(od_redirect_exist);find(do_redirect_exist)])))
294
295 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(od_redirect_exist) =
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats
(od_redirect_exist) + SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(airport_sst_compatibility_index(1));
296 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(do_redirect_exist) =
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats
(do_redirect_exist) + SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(no_comp_do_index);
297
298
299 SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
300 SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
301 SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
302 SST_Final_Market.(['Year_',num2str(year)]).DepReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
303 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
304 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
305 SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
306 SST_Final_Market.(['Year_',num2str(year)]).ArrReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
307 SST_Final_Market.(['Year_',num2str(year)]).Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
308 SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
309 SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs([find(no_comp_do_index);airport_sst_compatibility_index(1)]) =
[];
310 SST_Final_Market.(['Year_',num2str(year)]).DistStMiles([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
311 SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
312 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
313 SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
314 SST_Final_Market.(['Year_',num2str(year)]).generic_cdf([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
315
316 end
317
318
319 else
320
321 %disp(year)
322
323 %disp('delete')
324
325 %disp(SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]))
326
327 %disp(SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]))
328
329 SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
330 SST_Final_Market.(['Year_',num2str(year)]).DepAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
331 SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
332 SST_Final_Market.(['Year_',num2str(year)]).DepReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
```

```
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrAirport([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).ArrReg([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs([find(no_comp_do_index);airport_sst_compatibility_index(1)]) =
[];
SST_Final_Market.(['Year_',num2str(year)]).DistStMiles([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];
SST_Final_Market.(['Year_',num2str(year)]).generic_cdf([find(no_comp_do_index);airport_sst_compatibility_index(1)]) = [];


end
airport_sst_compatibility_index = ismember(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,Airport_List_SST_Compatibility)==0;

sst_od = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);

end

end
end

%%

TT_Dep_Arr = strcat(Travel_Times.DepAirport,'_',Travel_Times.ArrAirport);

for year = SST_Year_Start:SST_Year_End

number_of_OD = length(SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID);

SST_Dep_Arr = strcat(SST_Final_Market.(['Year_',num2str(year)]).DepAirport,'_',SST_Final_Market.(['Year_',num2str(year)]).ArrAirport);

for OD = 1:number_of_OD

[od_travel_time_index,~] = ismember(TT_Dep_Arr,SST_Dep_Arr(OD));

if sum(od_travel_time_index) ~= 1

od_travel_time_index = 0;
end

seats = SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(OD);

flights = ceil(seats ./ acft_seat_capacity);

if od_travel_time_index == 0
%disp(1)
total_flight_hours = 0;
else

total_flight_hours = flights .* Travel_Times.Travel_Time(od_travel_time_index);

end

SST_Final_Market.(['Year_',num2str(year)]).Aircraft_Needed(OD,1) = (total_flight_hours ./ hours_per_year);

clear seats flights total_flight_hours
end

SST_Final_Market.(['Year_',num2str(year)]).Number_of_Passenger = SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats .*
acft_pax_load_factor;

end
```

```
397
398
399
400
401 for year = SST_Year_Start:SST_Year_End
402
403 delete = find(SST_Final_Market.(['Year_',num2str(year)]).Aircraft_Needed == 0);
404
405 if isempty(delete) == 0
406
407 SST_Final_Market.(['Year_',num2str(year)]).DepAirport_UniqueID(delete) = [];
408 SST_Final_Market.(['Year_',num2str(year)]).DepAirport(delete) = [];
409 SST_Final_Market.(['Year_',num2str(year)]).DepIATACtry(delete) = [];
410 SST_Final_Market.(['Year_',num2str(year)]).DepReg(delete) = [];
411 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport_UniqueID(delete) = [];
412 SST_Final_Market.(['Year_',num2str(year)]).ArrAirport(delete) = [];
413 SST_Final_Market.(['Year_',num2str(year)]).ArrIATACtry(delete) = [];
414 SST_Final_Market.(['Year_',num2str(year)]).ArrReg(delete) = [];
415 SST_Final_Market.(['Year_',num2str(year)]).Total_Seats(delete) = [];
416 SST_Final_Market.(['Year_',num2str(year)]).Premium_Seats(delete) = [];
417 SST_Final_Market.(['Year_',num2str(year)]).OD_AverageElapsedTime_hrs(delete) = [];
418 SST_Final_Market.(['Year_',num2str(year)]).DistStMiles(delete) = [];
419 SST_Final_Market.(['Year_',num2str(year)]).SST_Premium_Seats(delete) = [];
420 SST_Final_Market.(['Year_',num2str(year)]).SST_Total_Seats(delete) = [];
421 SST_Final_Market.(['Year_',num2str(year)]).Aircraft_Needed(delete) = [];
422 SST_Final_Market.(['Year_',num2str(year)]).Number_of_Passenger(delete) = [];
423 SST_Final_Market.(['Year_',num2str(year)]).Fare_per_mile_cost (delete) = [];
424 SST_Final_Market.(['Year_',num2str(year)]).generic_cdf(delete) = [];
425
426 end
427 end
428
429
430
431 save(([save_dir,main_delimiter,'SST_Final_Market_OverlandRange_',num2str(aircraft_range_overland),'_nm_OverwaterRange_',num2str
(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-
1)*10),'_Int',date,'.
mat']),'SST_Final_Market')
432 return;
433
```

```
1 function [number_of_aircraft_needed] =
Final_Results_Merge(SST_Market_Analysis_Dir,aircraft_range_overland,aircraft_range_overwater,mach_overland,
mach_overwater,date)
2
3
4 load(([SST_Market_Analysis_Dir,'\US_Market\SST_Forecast\Output\SST_Final_Market_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_US',date,'.mat']),'SST_Final_Market')
5 SST_Final_Market_US = SST_Final_Market;
6 clear SST_Final_Market
7
8 load(([SST_Market_Analysis_Dir,'\US_Int_Market\SST_Forecast\Output\SST_Final_Market_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_US_Int',date,'.mat']),'SST_Final_Market')
9 SST_Final_Market_US_Int = SST_Final_Market;
10 clear SST_Final_Market
11
12 load(([SST_Market_Analysis_Dir,'\Int_Market\SST_Forecast\Output\SST_Final_Market_OverlandRange_',num2str
(aircraft_range_overland),'_nm_OverwaterRange_',num2str(aircraft_range_overwater),'_nm_Mach_Overland_1_',num2str((mach_overland-1)
*10),'_&_Mach_Overwater_1_',num2str((mach_overwater-1)*10),'_Int',date,'.mat']),'SST_Final_Market')
13 SST_Final_Market_Int = SST_Final_Market;
14 clear SST_Final_Market
15
16
17 for year = 2030:2040
```

```
18
19 if isempty(SST_Final_Market_Int.(['Year_',num2str(year)]).DepAirport_UniqueID) == 1
20
21 int_od = 0;
22 int_seat = 0;
23 int_acft = 0;
24 int_pax = 0;
25
26 else
27
28 int_od = length(SST_Final_Market_Int.(['Year_',num2str(year)]).DepAirport_UniqueID);
29 int_seat = sum(SST_Final_Market_Int.(['Year_',num2str(year)]).SST_Total_Seats);
30 int_acft = sum(SST_Final_Market_Int.(['Year_',num2str(year)]).Aircraft_Needed);
31 int_pax = sum(SST_Final_Market_Int.(['Year_',num2str(year)]).Number_of_Passenger);
32
33 end
34
35 if isempty(SST_Final_Market_US.(['Year_',num2str(year)]).DepAirport_UniqueID) == 1
36
37 us_od = 0;
38 us_seat = 0;
39 us_acft = 0;
40 us_pax = 0;
41
42 else
43
44 us_od = length(SST_Final_Market_US.(['Year_',num2str(year)]).DepAirport_UniqueID);
45 us_seat = sum(SST_Final_Market_US.(['Year_',num2str(year)]).SST_Total_Seats);
46 us_acft = sum(SST_Final_Market_US.(['Year_',num2str(year)]).Aircraft_Needed);
47 us_pax = sum(SST_Final_Market_US.(['Year_',num2str(year)]).Number_of_Passenger);
48
49 end
50
51 if isempty(SST_Final_Market_US_Int.(['Year_',num2str(year)]).DepAirport_UniqueID) == 1
52
53 us_int_od = 0;
54 us_int_seat = 0;
55 us_int_acft = 0;
56 us_int_pax = 0;
57
58 else
59
60 us_int_od = length(SST_Final_Market_US_Int.(['Year_',num2str(year)]).DepAirport_UniqueID);
61 us_int_seat = sum(SST_Final_Market_US_Int.(['Year_',num2str(year)]).SST_Total_Seats);
62 us_int_acft = sum(SST_Final_Market_US_Int.(['Year_',num2str(year)]).Aircraft_Needed);
63 us_int_pax = sum(SST_Final_Market_US_Int.(['Year_',num2str(year)]).Number_of_Passenger);
64
65 end
66
67
68 final_merge_all.od((year-2029),1) = us_od + int_od + us_int_od;
69
70
71 final_merge_all.seats((year-2029),1) = us_seat + int_seat + us_int_seat;
72
73
74 final_merge_all.acft((year-2029),1) = ceil((us_acft + int_acft + us_int_acft) ./ 0.90);
75
76
77 final_merge_all.pax((year-2029),1) = us_pax +int_pax + us_int_pax;
78
79
80 end
81
82 number_of_aircraft_needed = final_merge_all.acft((year-2029),1);
83return;
```