3-14-2014

Ground Target Overflight and Orbital Maneuvering via Atmospheric Maneuvering

Devin K. Dalton

GROUND TARGET OVERFLIGHT AND ORBITAL MANEUVERING VIA AEROASSISTED MANEUVERS

THESIS

Devin K. Dalton, Captain, USAF
AFIT-ENY-14-M-12

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENY-14-M-12

GROUND TARGET OVERFLIGHT AND ORBITAL MANEUVERING VIA AEROASSISTED MANEUVERS

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Aeronautical Engineering

Devin K. Dalton, BS

Captain, USAF

March 2014

AFIT-ENY-14-M-12

GROUND TARGET OVERFLIGHT AND ORBITAL MANEUVERING VIA AEROASSISTED MANEUVERS

Devin K. Dalton, BS
Captain, USAF

Approved:

_____//signed//_____	10 March 2014
Ronald J. Simmons, Lt Col, USAF, PhD (Chair)	Date
_____//signed//_____	10 March 2014
Jonathan T. Black, PhD	Date
_____//signed//_____	10 March 2014
Kerry D. Hicks, PhD	Date

Abstract

A satellite with lifting capabilities may enter into a skip trajectory wherein perigee of the orbit is within the sensible atmosphere of the Earth. During flight through the atmosphere, aerodynamic forces may be utilized to modify the orbital trajectory rather than using onboard fuel. This aeroassisted maneuver has the potential to decrease fuel costs of re-tasking satellites to overfly ground locations, the cost of orbit rendezvous, and initial orbit insertion from launch. The trajectory dynamics of purely propulsive in-plane and out-of-plane maneuvers, along with aeroassisted maneuvers, are simulated in order to determine the time of arrival and ΔV associated with each maneuver required to overfly a sample ground target. Results indicate that aeroassisted maneuvers offer more overflight solutions per day than planar maneuvers while requiring less ΔV than exo-atmospheric plane change maneuvers. The time of arrival and ΔV associated with each maneuver required to overfly a ground target is found for multiple ground target locations and starting orbits in order to determine analytical trends. From these trends, closed-form estimations of the ΔV and arrival time are generated for each maneuver type. Initial closed-form estimations show reasonable accuracy. Furthermore, the ability of the aeroassisted maneuver to modify an initial orbital trajectory is quantified by measuring the change in inclination and right ascension of the ascending node (RAAN) as perigee is lowered into the atmosphere. Results show a 75% decrease in ΔV over traditional exo-atmospheric maneuvers with a single skip enabling a satellite to change the orbital inclination and RAAN up to 45° and 90° respectively.

To the love of my life, who supports, enables, and encourages me, always.

Acknowledgments

First, I would like to express my deep appreciation to my wife for her love and support. She encourages and uplifts me and without her I could not have accomplished what I have. I would also like to express my sincere appreciation to my advisor Lt. Col Ronald Simmons who's knowledge has been tested by having me as a research student. His belief in me helped me to believe in myself. I am grateful as well for the support of the many other faculty members who were always willing to drop what they were doing in order to help me, including those on my thesis committee, Dr Black, and Dr Hicks. Lastly, I would like to thank Bob Bettinger for the countless hours he helped me to understand and define the problem as well as for his help in attempting to communicate the results of my research in this thesis document.

Devin K. Dalton

Table of Contents

Page

List of Figures

Page

List of Tables

Page

List of Symbols

Latin Symbol	*Definition*	*Base Unit of Measure*
a	Orbital semi-major axis	m
a_{decel}	Total deceleration	$\mathrm{m} \cdot \mathrm{s}^{-2}$
g	Gravitational acceleration	$\mathrm{m} \cdot \mathrm{s}^{-2}$
h	Altitude	m
i	Inclination angle	rad
m	Vehicle mass	kg
r	Geocentric radial distance from planet center of mass to vehicle	m
t	General time	s
C_D	Coefficient of drag	unitless
C_L	Coefficient of lift	unitless
D	Drag force	$\mathrm{kg} \cdot \mathrm{m} \cdot \mathrm{s}^{-1}$
L	Lift force	$\mathrm{kg} \cdot \mathrm{m} \cdot \mathrm{s}^{-1}$
N_{orbits}	Number of Orbits	variable
P	Keplerian orbital period	s
$\dot{Q}$	Stagnation Heat Flux	$\mathrm{BTU}(\mathrm{ft}^2\ \mathrm{s})^{-1}$,
RMS	Root mean square	unitless
S	Planform area	m^2
V	Velocity	$\mathrm{m} \cdot \mathrm{s}^{-1}$

Greek Symbol	*Definition*	*Base Unit of Measure*
β	Atmospheric scale height	m^{-1}
γ	Flight-path angle	rad
θ	Longitude	rad
μ	Gravitational parameter	$\mathrm{m}^3 \cdot \mathrm{s}^{-2}$
ν	True Anamoly	rad
ρ	Atmospheric density	$\mathrm{kg} \cdot \mathrm{m}^{-3}$
σ	Bank angle	rad
φ	Latitude	rad
ψ	Heading angle	rad
ω	Argument of Perigee	rad
$\omega_{\oplus}$	Planetary rotation rate	$\mathrm{rad} \cdot \mathrm{s}^{-2}$
Ω	Right Ascension of the Ascending Node	rad
Δ	Change in value, i.e. ΔV	$(\cdot)$

GROUND TARGET OVERFLIGHT AND ORBITAL MANEUVERING VIA AEROASSISTED MANEUVERS

I. Introduction

General Issue

The development of Trans-Atmospheric Vehicles (TAVs) enable the utilization of skip or atmospheric maneuvers wherein the aerodynamic forces can be exploited with lifting surfaces to maneuver the vehicle to a desired orbit change without the use of onboard propellant. The purpose of this research is to determine under what conditions an atmospheric skip maneuver would be preferable to more traditional methods, such as a) phasing maneuvers (change only the semi-major axis or size of the orbit); b) simple plane changes (change only the inclination of the orbit); and c) Combined plane changes (change both the inclination and the right ascension of the ascending node (RAAN) of the orbit, where RAAN is the angle between the right ascension and the vernal equinox). Another purpose of this study is to identify and exploit trends in the velocity change cost (ΔV) of the aeroassisted maneuver given various initial orbits and mission parameters in order to enable vehicle users, and researchers a preliminary estimate of cost for a desired orbital change.

Problem Statement

The capability of having a satellite over any given spot on the Earth at any given time is extremely valuable for a variety of reasons, to include, reconnaissance, surveillance, and weather tracking. While infeasible to have satellites everywhere at once, this capability is approximated using constellations of satellites; each enabled to be

re-tasked in order to overfly a desired point on the Earth at a desired time. These re-taskings are extremely expensive in terms of fuel cost, which for an orbiting satellite equates to system lifespan and mission availability. Overall, the user's need and associated benefit of re-tasking a satellite must be evaluated against the degradation of system lifespan in terms of months or even years.

TAV's have the potential of offsetting some of these fuel costs, for certain types of maneuvers. The capabilities of the TAV to offset fuel costs remains largely undefined and, therefore, it is the focus of this research to explore the capabilities of a notional TAV defined in Table 1.1 to change any given Low Earth Orbit (LEO) to a new desired LEO orbit so as to overfly a specified ground location, rendezvous with another satellite or be placed in its mission orbit.

Table 1.1. Trans-Atmospheric Vehicle (TAV) Parameters (Bettinger, 2014: 4)

Total Wet Mass	5000 kg
Planform Area (S)	10 m^2
Coefficient of Drag (C_D)	0.5
Coefficient of Lift (C_L)	3.0

Research Objectives, Questions and Hypotheses

It is the intent of this research to complete the following research objectives:

- Given any circular reference orbit determine the ΔV and associated time of arrival (TOA) required to overfly any given ground target location on the Earth, within some tolerance, using purely planar phasing maneuvers, simple plane changes, and atmospheric skip maneuvers.

- Extrapolate data from multiple reference orbits, target locations, and bank angles to create a general ΔV and TOA calculator for any target location and reference orbit.
- Identify trends in both ΔV cost and TOA with target latitude versus orbit inclination and target longitude versus orbit RAAN.
- Given any circular reference orbit, determine the reachability domain of the TAV to enter other orbital trajectories via atmospheric maneuvering.

Methodology

The dynamics model used throughout this research was developed by Dr. Kerry Hicks in his book *An Introduction to Astrodynamic Re-Entry*. The equations of motion are defined in Eqs (1.1) – (1.6) and are valid for both exo-atmospheric and atmospheric maneuvers without thrust (Hicks, 2009:42, 52).

$$\dot{r} = {}^{R}V \sin\gamma \tag{1.1}$$

$$\dot{\theta} = \frac{{}^{R}V \cos\gamma \cos\psi}{r \cos\phi} \tag{1.2}$$

$$\dot{\phi} = \frac{{}^{R}V \cos\gamma \sin\psi}{r} \tag{1.3}$$

$${}^{R}\dot{V} = -\frac{D}{m} - g \sin\gamma + r\omega_{\oplus}^{2} \cos\phi \,(\cos\phi \sin\gamma - \sin\phi \sin\psi \cos\gamma) \tag{1.4}$$

$${}^{R}V\dot{\gamma} = \frac{L}{m}\cos\sigma - g\cos\gamma + \frac{{}^{R}V^{2}}{r}\cos\gamma + 2\,{}^{R}V\omega_{\oplus}\cos\phi\cos\psi + r\omega_{\oplus}^{2}\cos\phi\,(\cos\phi\cos\gamma - \sin\phi\sin\psi\sin\gamma) \tag{1.5}$$

$$ {}^{R}V\dot{\psi} = \frac{L\sin\sigma}{m\cos\gamma} - \frac{{}^{R}V^{2}}{r}\cos\gamma\cos\psi\tan\phi + 2\,{}^{R}V\omega_{\oplus}(\sin\psi\cos\phi\tan\gamma - \sin\phi) - \frac{r\omega_{\oplus}^{2}}{\cos\gamma}\sin\phi\cos\phi\cos\psi \tag{1.6} $$

Where all quantities are defined in the nomenclature index of Appendix A.

The majority of the research done was through a dynamics simulation model which allows the user to define a target location on the Earth, an initial circular LEO orbit, TAV parameters, a simulation run time, and a start date and time. The simulation model can also sweep through any input parameter while holding the others constant. Given the inputs the simulation uses the equations of motion defined above to propagate the orbit forward in time for the user specified run time. The simulation will then find all target overflight solutions for three types of maneuvers; completely in-plane phasing maneuvers, simple plane change maneuvers, and atmospheric skip maneuvers (defined below). Multiple parameters are recorded for each overflight such as: time of arrival, needed velocity change, inclination change, acceleration, and perigee altitude. The overflight data was recorded for multiple target locations and initial orbits in order to map broader trends and determine under what conditions an atmospheric skip maneuver is profitable. Once data from multiple scenarios was collected the data was analyzed for relationships and the data was fit to curves using various methods. Models were then developed which give reasonable estimations of the cost of aeroassisted maneuvers which can be used by system users to determine the worth of doing such a maneuver.

Phasing Maneuvers.

A phasing maneuver is performed by either increasing or decreasing the semi-major axis of the original reference orbit. An instantaneous increase in velocity in the direction of satellite motion will create a lofting eccentric orbit with perigee at the original reference orbit maneuver altitude (Fig. 1.1). This increase in semi-major axis will increase the period of the orbit, thus allowing the Earth more time to rotate before the satellite passes over a given location and effectively move the ground track westward (Fig. 1.2).

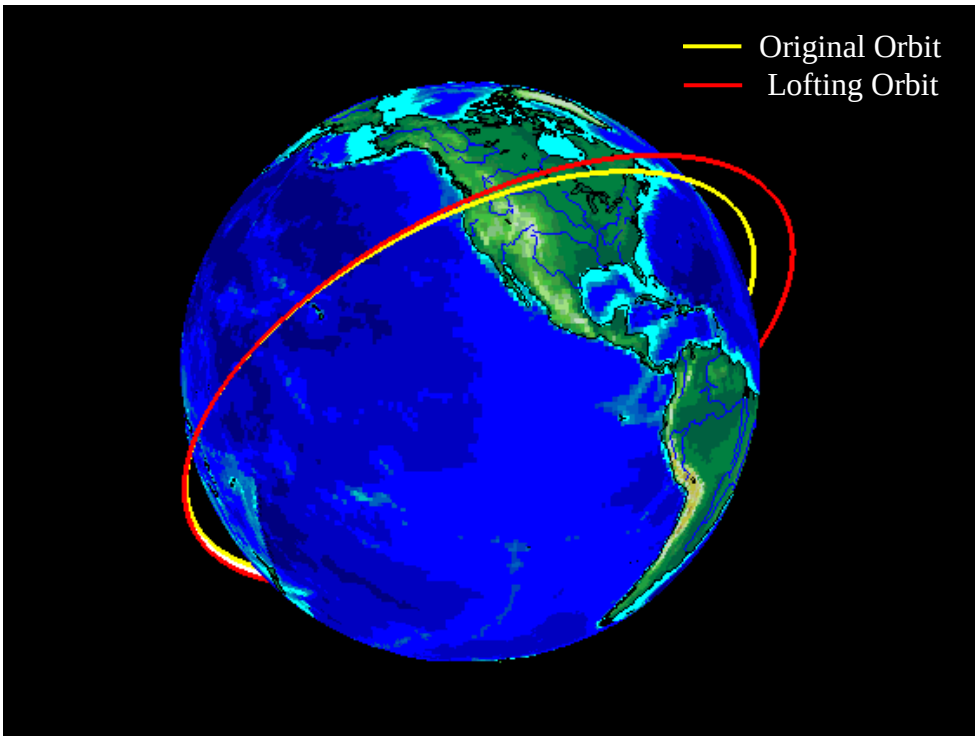

Figure 1.1. Lofting in-Plane Phasing Maneuver

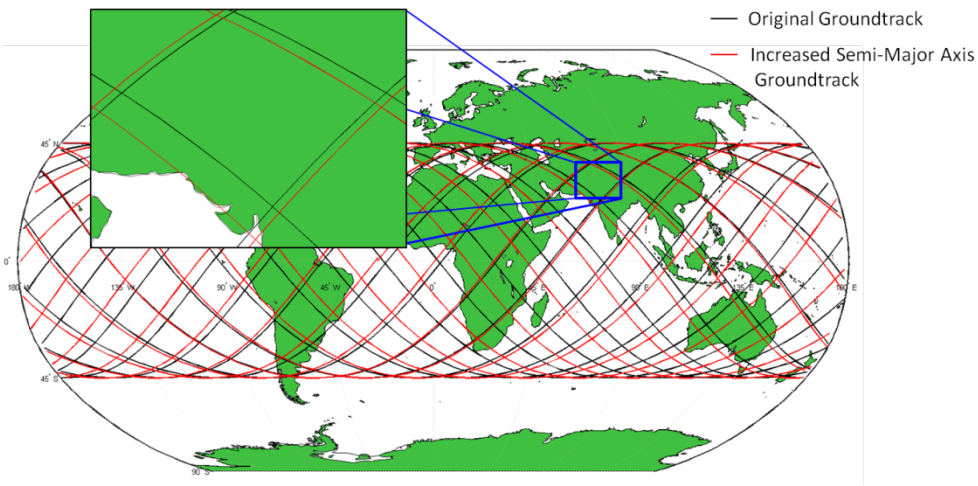

Figure 1.2. Increased Semi-Major Axis Resulting in a Westward Shift of the Groundtrack

Conversely, an instantaneous decrease in velocity in the direction of satellite motion will create a descending eccentric orbit with apogee at the original reference orbit maneuver altitude. This decrease in semi-major axis will decrease the period of the orbit and therefore allow the Earth less time to rotate before the satellite passes over a given point, effectively moving the ground track eastward (Fig. 1.3). A descending phasing maneuver is limited by the perigee altitude of the descending eccentric orbit; once the orbit perigee is below the sensible atmosphere friction from the atmosphere will reduce the kinetic energy of the TAV. Such a reduction in energy will prevent the TAV from reaching the initial, pre-maneuver orbit altitude without an additional orbit-raising maneuver. Figure 1.4 shows the variation in longitude and altitude for both lofting and descending phasing maneuvers.

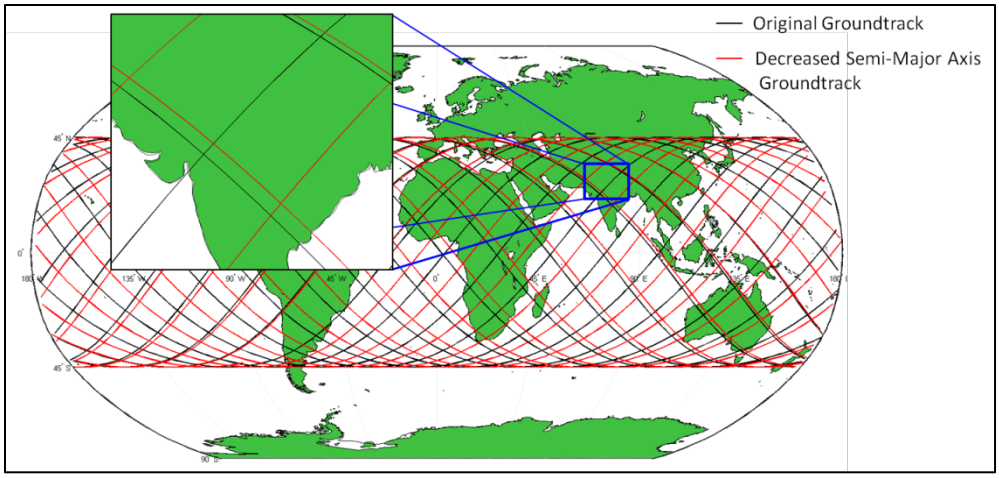

Figure 1.3. Decreased Semi-Major Axis Resulting in an Eastward Shift of the Groundtrack

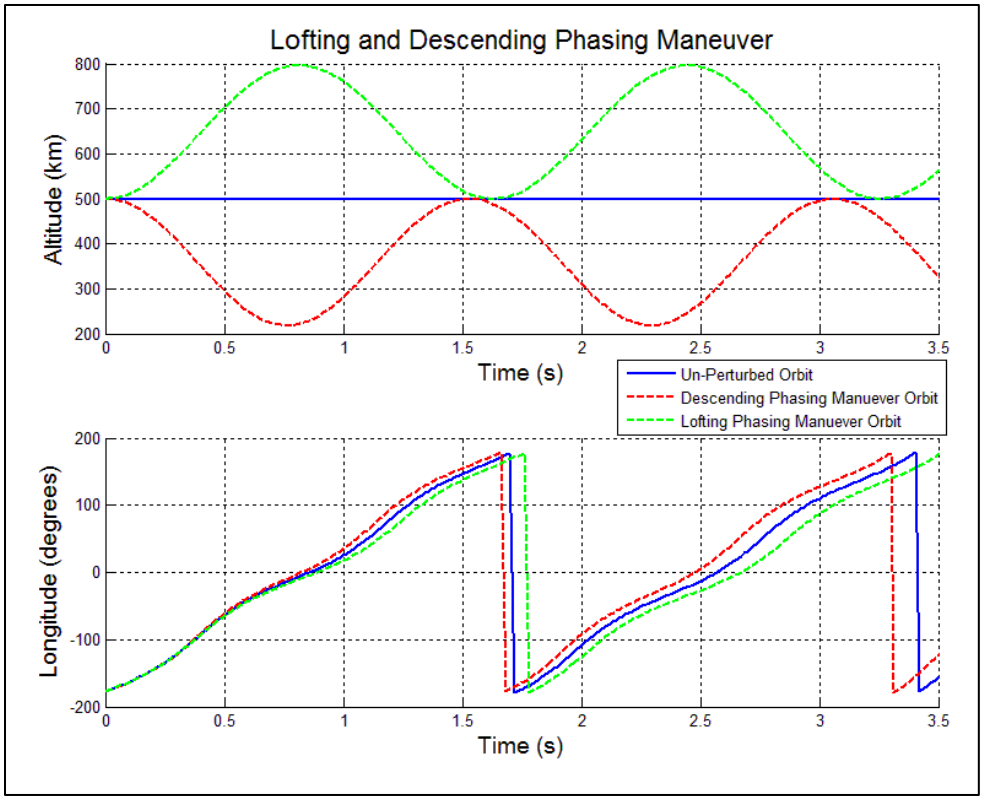

Figure 1.4. Increasing/Decreasing the Orbital Period via a Lofting/Descending Phasing Maneuver

Plane Change Maneuvers.

For the purposes of this research, a plane change maneuver is defined as a change only in orbit inclination (Fig 1.5). The ΔV for this maneuver is given by Eq. (1.7), which was obtained from William E. Wiesel in his book *Spaceflight Dynamics* (110). The direction that the ground track is shifted for both an increase and decrease in inclination depends on the direction the satellite is moving (north or south) as well as the hemisphere in which the maneuver is performed. These effects are detailed in Tables 1.2 and 1.3, and are shown in Fig. 1.6.

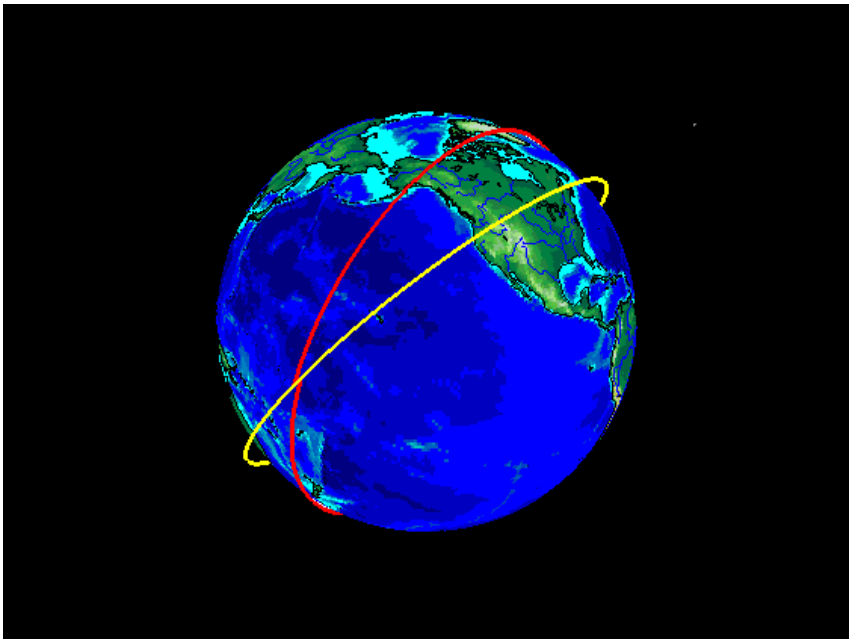

Figure 1.5. Simple Plane Change Showing a 25° Inclination Change

$$\Delta V = 2V_{initial} \sin(\Delta i) \tag{1.7}$$

Where

ΔV = Velocity change needed for the inclination change

$V_{initial}$ = Satellite velocity at time of maneuver

Δi = Inclination change

Table 1.2. Effect on Pro-Grade Ground Track Due to Inclination Change

Inclination Change	Hemisphere	Current Heading	Ground Track Effect
(+)	North	North	North-West
		South	North-East
	South	North	South-East
		South	South-West
(-)	North	North	South-East
		South	South-West
	South	North	North-West
		South	North-East
(+) or (-)	Equator	North or South	No Change

Table 1.3. Effect on Retro-Grade Ground Track Due to Inclination Change

Inclination Change	Hemisphere	Direction	Ground Track Effect
(+)	North	North	North-East
		South	North-West
	South	North	South-West
		South	South-East
(-)	North	North	South-West
		South	South-East
	South	North	North-East
		South	North-West
(+) or (-)	Equator	North or South	No Change

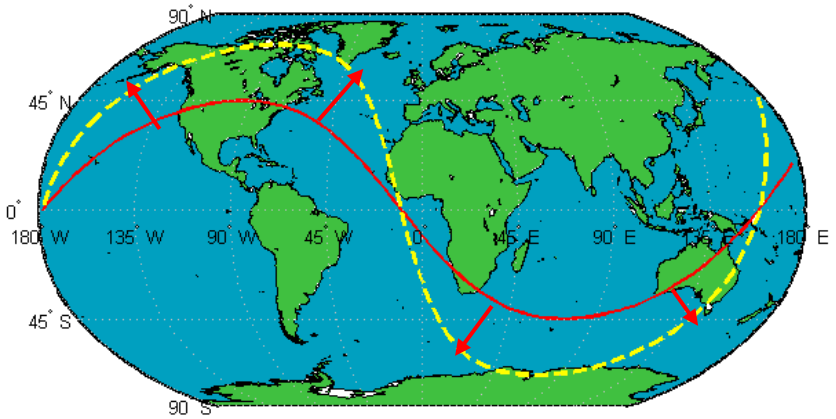

Fig. 1.6. Effect on Groundtrack at Various Stages of Orbit
For a Positive Inclination Change

Aeroassisted Maneuvers

A skip or aeroassisted maneuver is similar to a descending phasing maneuver with the exception that the TAV descends into the sensible atmosphere and then banks either left (-) or right (+) in order to change both inclination and RAAN. Fig. 1.7 illustrates the three phases of an aeroassisted maneuver: exo-atmospheric deorbit, atmospheric flight, and exo-astmospheric re-orbit. The effect on the inclination and RAAN for a given positive or negative bank depends on the flight direction and current hemisphere location (Table 1.4). Once the TAV reaches its decayed apogee altitude following atmospheric passage the vehicle can either re-circularize its orbit at the decayed apogee altitude or initiate a Hohmann transfer and re-circularize at the higher original altitude.

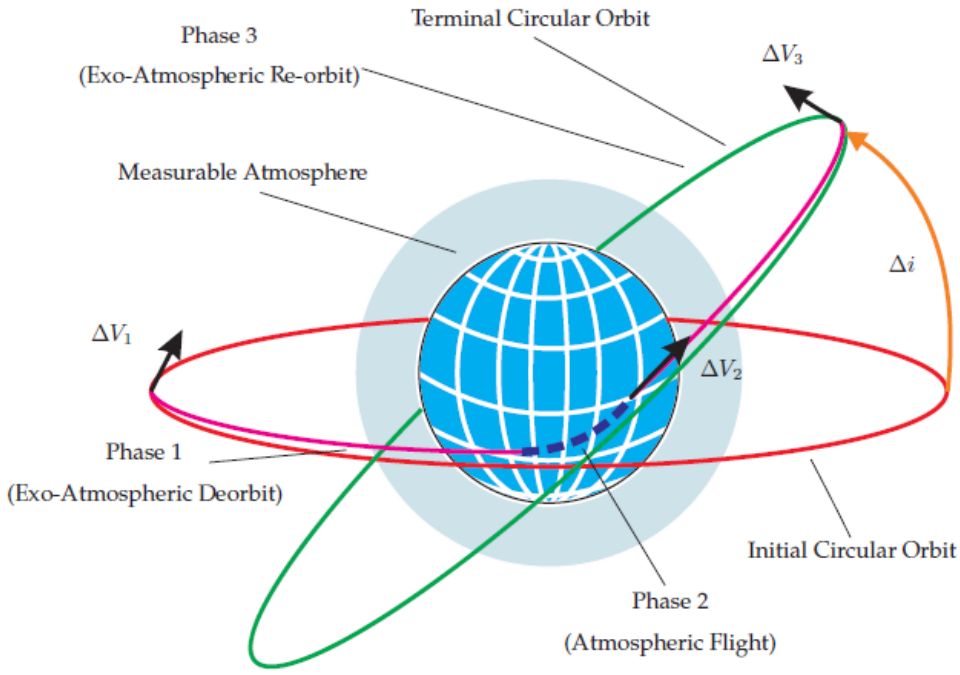

Figure 1.7. Schematic of Trajectory for an Aeroassisted Orbital Transfer (Rao, et al., 2008: 4)

Table 1.4. Effect on a Pro-Grade Orbit Inclination and RAAN
Due to Banking within Atmosphere

Bank	Hemisphere	Direction	Effect on Inclination	Effect on RAAN
(+)	North	North	(-)	(-)
		South	(-)	(+)
	South	North	(+)	(-)
		South	(+)	(+)
(-)	North	North	(+)	(+)
		South	(+)	(-)
	South	North	(-)	(+)
		South	(-)	(-)

Table 1.5. Effect on a Retro-Grade Orbit Inclination and RAAN
Due to Banking within Atmosphere

Bank	Hemisphere	Direction	Effect on Inclination	Effect on RAAN
(+)	North	North	(-)	(+)
		South	(-)	(-)
	South	North	(+)	(+)
		South	(+)	(-)
(-)	North	North	(+)	(-)
		South	(+)	(+)
	South	North	(-)	(-)
		South	(-)	(+)

Assumptions and Limitations

The scope of this research is limited to circular LEO reference orbits. The limitation is reasonable as most satellites with the capability of performing an aeroassisted maneuver reside in LEO. Other assumptions include: two-body dynamics; spherical Earth; constant mass, lift coefficient and drag coefficient; lift always perpendicular to the velocity vector and drag always parallel to the velocity vector. Two-body dynamics are very accurate for small time scales. The majority of the research

presented is limited to 24 hrs and therefore multiple body dynamics have little effect on the TAV trajectory. Oblate Earth effects are also minimal and were neglected as the overall effort of this study is to determine trends and patterns for the timing and ΔV of overflight solutions and orbital parameter change. Oblate Earth effects have no effect on the trends and little effect on the timing and ΔV of solutions. The study is limited to non-thrusting TAVs during the aeroassisted maneuver and so mass stays constant other than for oblation which is small enough to be ignored. Constant lift and drag coefficients are a simplifying case but the findings in this study can be modified for vehicles with variable lift and drag.

Preview

With a general outline of the research objectives and methodology given in Chapter 1, Chapter II will comprise a review of relevant literature on the topic of skip entry and aeroassisted maneuvers. Studies have shown that orbit modification can be done using aeroassisted maneuver using less ΔV than traditional methods. However, research has been more specific in scope with regards to usability of the maneuver to perform a valuable mission from multiple starting orbits. Chapter III will provide a more in-depth analysis of the methodology employed to obtain the results given in Chapter IV. Chapter III will also include an analysis of the validity of the preceding assumptions and limitations. In addition to simulation results Chapter IV will provide a closed-form analytic equation capable of being used to determine the cost and timing of ground target overflight solutions using phasing, simple plane change, and aeroassisted maneuvers. Chapter V will discuss the significance of the present research as well as provide an

assessment of further potential research on the topic of skip entry maneuvers. The appendix material will include: nomenclature, test matrices, and MATLAB programs/functions used in the simulation model.

II. Literature Review

Chapter Overview

The literature review presented in this chapter provides a summary of relevant research pertaining to aeroassisted satellite maneuvers so as to show uniqueness in the research presented by this thesis as well as provide an understanding of current technology and related research efforts in the field of aeroassisted maneuvers.

Relevant Research

Multiple studies beginning in the early 1990's have shown aeroassisted maneuvers to have many potential benefits. Researchers have demonstrated significant fuel savings over purely propulsive maneuvers for both out-of-plane and in-plane maneuvers. In Paul G. Gonsalves, Scott M. Allen, and Alper K Caglayan's technical report, "An Alternative Concept for Aeroassist Orbit Transfers," two types of maneuvers are identified: 1) co-planar orbital transfers from high Earth orbit (HEO) to LEO; and 2) an orbital plan change with or without a decrease in orbital altitude. They further categorize these maneuvers into 'hard' and 'soft' maneuvers. A 'soft' maneuver is defined as "a multiple pass aeroassisted orbital transfer which involves allowing the vehicle to stay at higher altitudes and use available aerodynamic forces during each perigee pass to perform and orbital transfer," while a 'hard' maneuver entails "forcing the vehicle to pull down into the atmosphere and performing the aeroassisted orbital transfer with a single pass" (Gonsalves, et al., 1991:9-10). 'Soft' maneuvers were determined to be beneficial for stationkeeping type operations as they don't require a deboost/re-boost stage and the heating and acceleration are kept minimal. Gonsalves et

al. determined that a 'hard' maneuver may be beneficial when "a political or military crisis necessitates a spacecraft performing a hard maneuver to position itself over a specific geographic region for a short period of time" (Gonsalves, et al., 1991:10). A hard type of maneuver allows for a much faster orbital transfer but comes at a higher fuel cost and increased heating and acceleration loads.

In a study of minimum-fuel LEO aeroassisted orbital transfer, Christopher Darby and Anil V. Rao identify aerobrake, aerocapture, and aerogravity assist maneuvers. Aerobrake is a planar maneuver wherein the perigee of an orbit is placed within the upper atmosphere of a planet in order to allow aerodynamic drag to decrease both the semi-major axis and eccentricity of an orbit. Similar to aerobrake is aerocapture which utilizes the aerodynamic forces of the upper atmosphere to decrease the orbital energy of a hyperbolic orbit to that of an elliptical orbit. Lastly, aerogravity assist "combines the atmosphere with propulsion and gravity" in order to produce a desired change in an orbit on a hyperbolic trajectory (Darby and Rao, 2010:3).

Further definitions of various types of aeroassisted maneuvers are found in Richard E. Johnsons' thesis entitled, "Effects of Thrust Vector Control on the Performance of the Aerobang Orbital Plane Change Maneuver." Three categories of maneuvers are identified as being synergistic in nature: aerobang, aerocruise, and aeroglide. All three maneuvers are similar and differ only in the use of thrust during the atmospheric phase of the maneuver. Propulsive forces are used by all three types to lower the vehicle into the atmosphere wherein the aerodynamic forces are used to modify the vehicle's orbit. For an aerobang maneuver, the vehicle continuously thrusts at the maximum throttle setting. Continuously thrusting allows the vehicle to spend a shorter

amount of time within the atmosphere therefore reducing the heating effects. The increased speed through the atmosphere also increases the aerodynamic forces, thus allowing for a greater change in orbital parameters and a smaller amount of fuel needed for re-circularization after the maneuver. An aerocruise maneuver also continuously thrusts through the atmosphere, but only thrusts enough to counteract drag. Lastly, aeroglide uses no thrust while passing through the atmosphere and requires a greater amount of fuel to re-circularize after the atmospheric maneuver (Johnson, 1993:3-4).

Of these three types of maneuvers, John C. Nicholson shows that the aeroglide is the least expensive in terms of fuel use in his "Numerical Optimization of Synergistic Maneuvers." He illustrates that for a 20% expenditure of available fuel, an inclination change of 6°, 5°, and 7° is possible for the aerobang, aerocruise, and aeroglide maneuvers respectively. Furthermore, the aeroglide maneuver is capable of delivering an 18° change in inclination for a 40% expenditure of fuel, while the aerobang and aerocruise are only capable of delivering 16° and 14° respectively. The lesser performance of the aerobang and aerocruise maneuvers is due in part to larger heating constraints. Nicholson continues with a comparison of the aeroassisted maneuvers with purely propulsive maneuvers conducted outside the atmosphere. He concludes that all aeroassisted maneuvers outperform exo-atmospheric inclination change. For the 20% and 40% fuel expenditure cases, the purely propulsive plane change can provide a 5° and 11° inclination change, respectively (Nicholson 1994:68).

Nicholson's findings are confirmed by Michael S. Parish in his thesis, "Optimality of Aeroassisted Orbital Plane Changes." Parish concludes that aeroassisted maneuvers in general require less propellant than an exo-atmopsheric maneuver to

produce a desired change in inclination angle (Parish, 1995:53, 55). Darby and Rao also find that aeroassisted maneuvers require less ΔV for a given inclination change than do exo-atmospheric maneuvers. For example, their research shows that an inclination change of 20° requires 1.5 km/s of ΔV for an aeroassisted maneuver, while the exo-atmospheric maneuver required 2.8 km/s. The difference is even starker for an inclination change of 40°: the aeroassisted maneuver required 2.0 km/s, while the exo-atmospheric maneuver required 5.5 km/s of ΔV (Darby and Rao, 2010:21).

In their paper "Numerical Optimization Study of Multiple-Pass Aeroassisted Orbital Transfer," Anil V. Rao, Sean Tang, and Wayne P. Hallman examined the problem of a minimum-impulse multiple-pass aeroassisted orbital transfer from geostationary orbit (GEO) to LEO with constraints on heating rate, angle-of-attack, and transfer time. Rao et al. determined that "When the heating rate is unconstrained the multiple-pass transfer offers only a small savings in DV over the single-pass transfer whereas when the heating rate is constrained the multiple-pass maneuver offers significant savings in DV over the single-pass transfer." (Rao, et al., 2002: 228-227). Furthermore, the total atmospheric inclination change approached the limit of approximately $36.2°$ as the number of atmospheric passes increased. This inclination change limit varies with TAV lift to drag ratio. In all test cases, the aeroassisted orbital transfer offered "substantial savings" in ΔV when compared with non-coplanar two-impulse (Hohmann-type) and three-impulse (bi-elliptic), all-propulsive transfers conducted in the vacuum environment (Rao, et al., 2002:228-230).

Prior to the commencement of any research into aeroassisted maneuvers, a firm foundation in the understanding of atmospheric re-entry is required. In *Space Vehicle*

Design, Michael Griffin and James French discuss reference systems needed to understand atmospheric re-entry. They define the entry interface altitude commonly taken by convention as 122 km (Griffin and French, 2004:277). In order to obtain a desired lateral maneuver, Griffen and French state that the "lifting entry vehicle must bank to obtain a turning force normal to the initial plane and upon attaining the desired heading angle change, reduce the bank angle again to zero". They also state that the propulsive requirements for both interplanetary and orbital operations can be significantly reduced with maneuvers that utilize the atmosphere for aero braking or aeroassisted plane change (Griffin and French, 2004:317-318).

Summary

Multiple studies have shown that aeroassisted maneuvers offer a unique and more affordable option to exo-atmospheric orbital transfers. It now becomes important to quantify the capabilities of the aeroassisted maneuver. This quantification includes constraints on the mission, timing, and initial orbit that are required to justify the use of an aeroassisted maneuver. It will also be important to understand the orbital parameter changes possible given any starting orbit. This study will differ from other studies in two ways: first, the aeroassisted maneuver will be analyzed for its use in overflying ground targets as opposed to simply orbit change; second, the goal of this study is to find analytical trends for multiple mission types while starting from multiple initial orbits as opposed to simply studying the aerodynamics of the aeroassisted maneuver.

III. Methodology

Chapter Overview

The purpose of this chapter is to provide a description of the model used to obtain the results throughout this thesis. This chapter will also provide an explanation of the assumptions, limitations, and algorithms used in the dynamics model for both exo-atmospheric and atmospheric skip maneuvers to include the atmospheric model and the deceleration calculations. Furthermore, the method of gathering and analyzing data will be discussed for both the ground target overflight and the orbit reachability via aeroassisted maneuver sections.

Overflight Simulation Dynamics Model

The ΔV and TOA needed to overfly a ground target from an initial orbit using in-plane phasing maneuvers, simple plane change maneuvers, and aeroassisted maneuvers were determined using an overflight simulation dynamics model. The model accepts initial orbit and target location inputs, then autonomously finds the ΔV and TOA for each maneuver type. A flow diagram of the model is shown in Fig. 3.1. The algorithm for each maneuver will be given in their respective sections below.

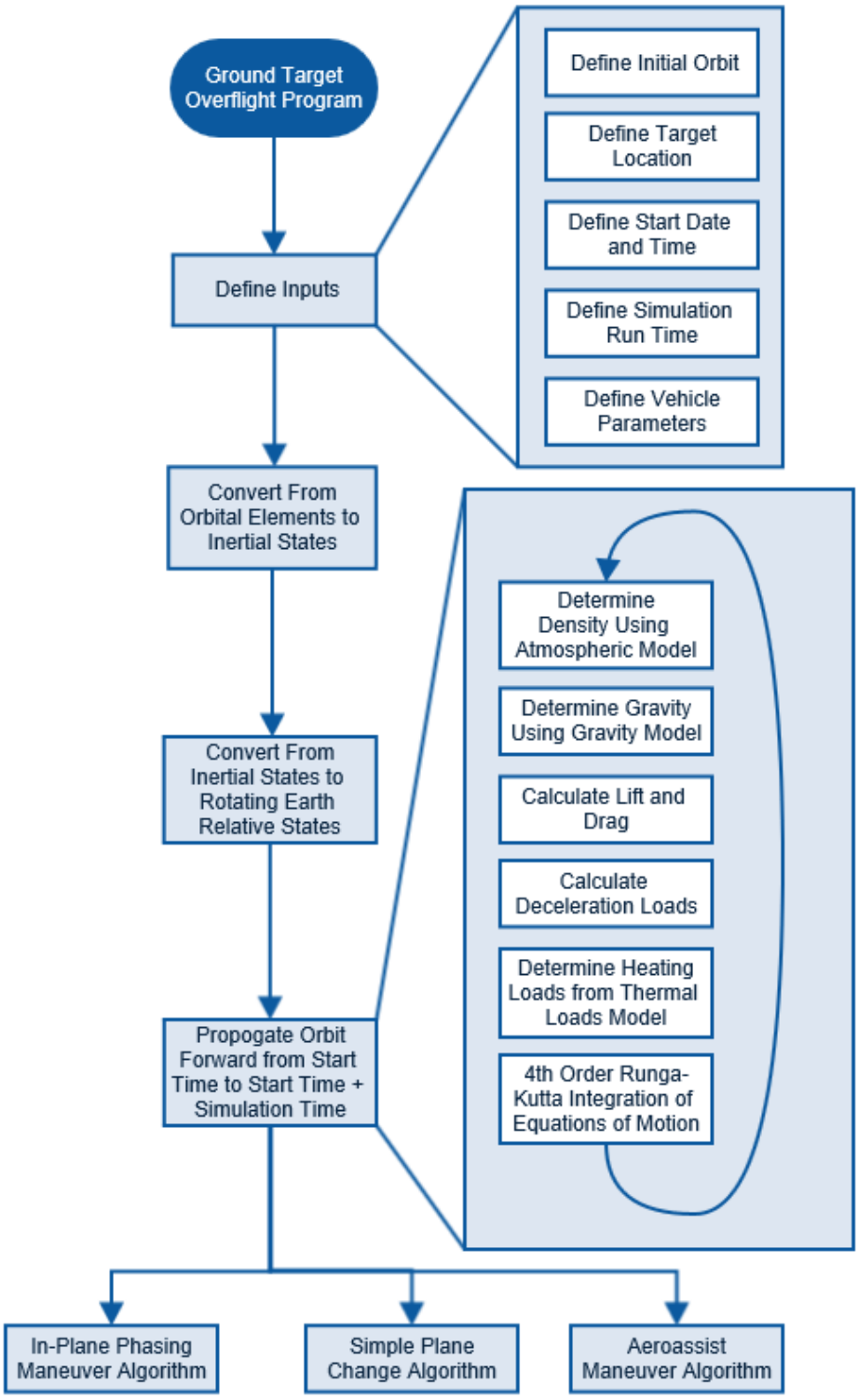

Figure 3.1. Overflight Simulation Dynamics Model Flow Diagram

Model Inputs

The user can modify any of the input parameters which include: initial orbit, target location, simulation start date and time, simulation run time, and vehicle parameters. The initial orbit is defined in terms of the six classical orbital elements defined by Wiesel in his book *Space Flight Dynamics* (Fig. 3.2).

1. Semi-Major-Axis, a: Half the major diameter of the orbital ellipse (since a spherical Earth and circular orbit are both assumed this value will be equal to the summation of the satellite altitude and radius of the Earth).
2. Eccentricity, e: Specifies the relative shape of the ellipse (e = 0 for a circular orbit)
3. Inclination, i: The angle between the equatorial plane and the orbital plane measured at the ascending node (the point where the orbit passes the equator while traveling north). It is also the angle between a line normal to the orbital plane and the line from the center of the Earth through the North Pole; the inclination range is $0° \leq i \leq 180°$.
4. Right Ascension of the Ascending Node (Ω): The angle between the vernal equinox and the ascending node, similar to longitude; the RAAN range is $0° \leq \Omega \leq 360°$.
5. Argument of Perigee, ω: The angle between the line of nodes and the perigee point. (for circular orbits perigee is not defined and so ω becomes unnecessary)
6. True Anamoly, ν: Gives the current location of the satellite along the orbital trajectory. The angle between the line of perigee (for circular

orbits and ω = 0, the point of perigee is the line of nodes) and the satellite. The ν range is 0° ≤ ν ≤ 360°.

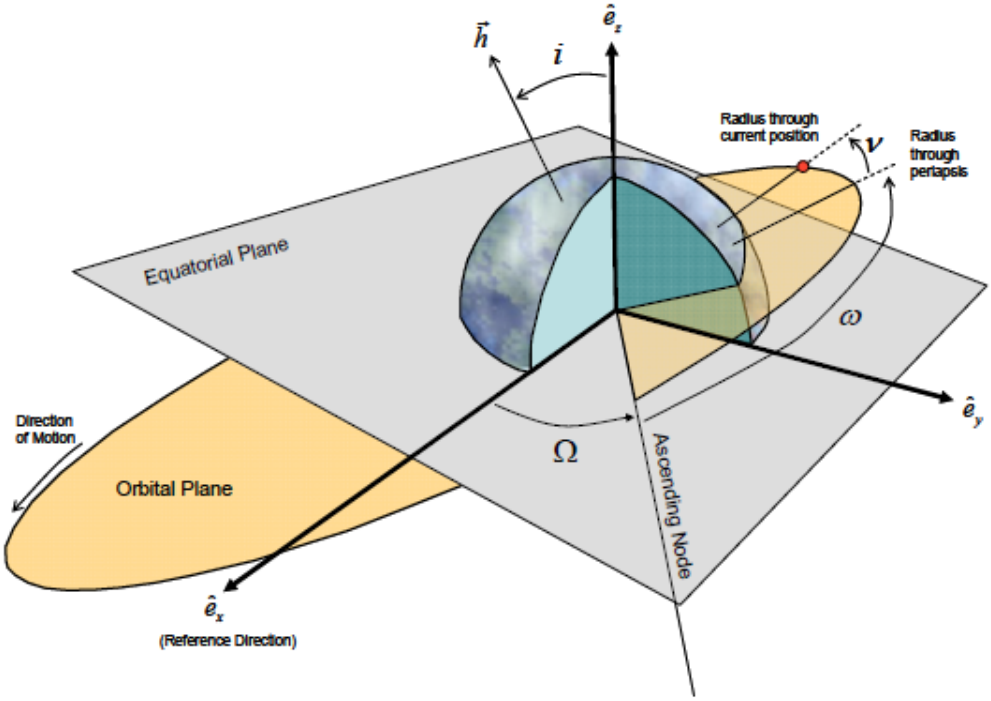

Figure 3.2. Angle Visualization of Classical Orbital Elements (Hicks, 2009:11)

Orbital Elements to Inertial Planet Fixed-States

In order to use the equations of motion developed by Hicks, the position and velocity of the satellite must be defined using six state variables defined below and shown in Fig 3.3 (Hicks, 2009:29-31). These state variables should be defined in relation to a planet fixed coordinate system where $\hat{e}_{x1}$ rotates with the Earth and points through the equator at a longitude of 0° (the prime meridian), $\hat{e}_{z1}$ points through the north pole and $\hat{e}_{y1}$ points out the equator at a longitude of 90°. The six state variables are defined as:

1. Radial Position, r: Distance from the center of the Earth.

2. Longitude, θ: Angular distance east (+) or west (-) from the prime meridian (-180° ≤ θ ≤ 180°).
3. Latitude, ϕ: Angular distance north (+) or south (-) of the equator (-90° ≤ φ ≤ 90°).
4. Inertial Velocity, ^{I}V: Magnitude of velocity relative to inertial space.
5. Flight Path Angle, γ: The angle between the local horizontal plane (the plane passing through the vehicle and orthogonal to the position vector) and the velocity vector. The flight path angle is negative for a satellite heading towards the Earth and positive for a satellite moving away from the Earth, this is similar to pitching angle (-90° ≤ γ ≤ 90°). .
6. Heading Angle, ψ: The angle between the local parallel of latitude and the projection of the velocity vector on the local horizontal plane. ψ defines whether the satellite is moving north (+) or south (-); (-180° ≤ ψ ≤ 180°).

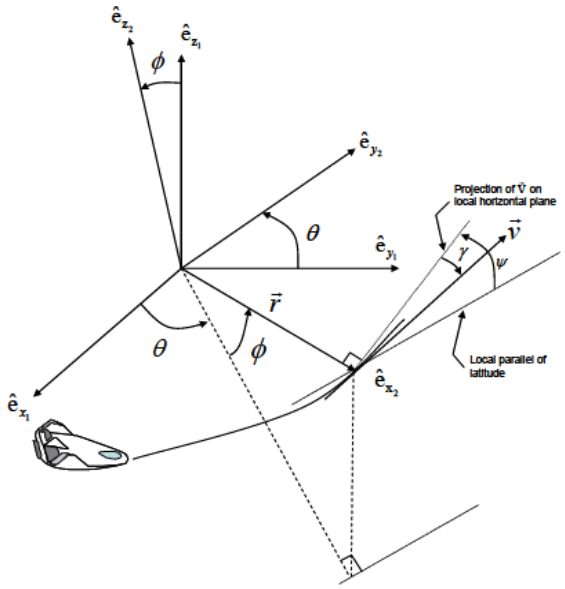

Figure 3.3. Illustration of the Six State Variables in Relation to Planet-Fixed ($\hat{e}_{x1}$, $\hat{e}_{y1}$, $\hat{e}_{z1}$) and Vehicle Pointing ($\hat{e}_{x2}$, $\hat{e}_{y2}$, $\hat{e}_{z2}$) Coordinate Systems (Hicks, 2009:31)

The classical orbital elements must now be transformed into the six state variables. The angle between the prime meridian and the vernal equinox must be determined because the state position variables are defined relative to a rotating Earth. This is done by first calculating the time difference between the simulation start time and a known time when the prime meridian was aligned with the vernal equinox, then multiplying that time by the rotation rate of the Earth, $\omega_{\oplus}$. The position and velocity vectors ($\vec{r}, \vec{v}$) can be calculated and expressed in the perifocal coordinate system (Fig. 3.4) using Eqs. (3.1) – (3.4) developed by Hicks (Hicks, 2009:22-23).

$$r = \frac{a(1-e^2)}{1+e\cos(\nu)} \tag{3.1}$$

$$\vec{r} = r\cos(\nu)\,\hat{e}_P + r\sin(\nu)\hat{e}_Q \tag{3.2}$$

$$p = a(1-e^2) \tag{3.3}$$

$${}^{I}\vec{V} = \sqrt{\frac{\mu}{p}}\left[-\sin(\nu)\,\hat{e}_P + (e+\cos(\nu))\hat{e}_Q\right] \tag{3.4}$$

Where

$\mu = Gm_{\oplus} = 3.986 \times 10^5$ km^3/s^2

G = universal gravitational constant = 6.67×10^{-11} Nm^2/kg^2

$m_{\oplus}$ = mass of the Earth = 5.98×10^{24} kg

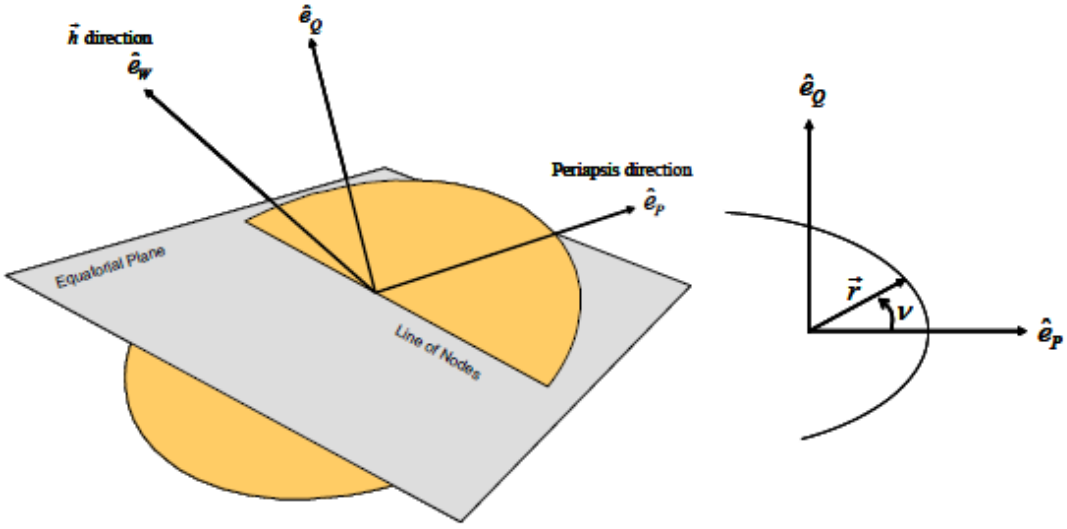

Fig. 3.4. Perifocal Coordinate System (Hicks, 2009:21)

The position and velocity vectors can then be transformed into planet fixed coordinates (Fig 3.3) via the following transformation matrices (Hicks, 2009:29-30):

$$[\hat{e}] = \begin{bmatrix} \cos(\Omega)\cos(\omega) - \sin(\Omega)\sin(\omega)\cos\,(\mathrm{i}) & -\cos(\Omega)\sin(\omega) - \sin(\Omega)\cos(\omega)\cos\,(i) & \sin(\Omega)\sin\,(i) \\ \sin(\Omega)\cos(\omega) + \cos(\Omega)\sin(\omega)\cos\,(i) & -\sin(\Omega)\sin(\omega) + \cos(\Omega)\cos(\omega)\cos\,(i) & -\cos(\Omega)\sin\,(i) \\ \sin(\omega)\sin\,(i) & \cos(\omega)\sin\,(i) & \cos\,(i) \end{bmatrix} \begin{bmatrix} \hat{e}_p \\ \hat{e}_Q \\ \hat{e}_w \end{bmatrix} \tag{3.5}$$

$$[\hat{e}_1] = \begin{bmatrix} \cos(\omega_\oplus \Delta t) & \sin(\omega_\oplus \Delta t) & 0 \\ -\sin(\omega_\oplus \Delta t) & \cos(\omega_\oplus \Delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} [\hat{e}] \tag{3.6}$$

Once the vectors are defined in the correct coordinate frame, the angle states can then be determined using Eqs. (3.7) – (3.10). In Eq. (3.10) the positive arc-cosine term is used for prograde orbits while the negative term is used for retrograde orbits. The r(1), r(2), and r(3) terms rerpresent the 1st, 2nd, and 3rd terms of the r vector.

$$\theta = \text{atan}\,\frac{r(2)}{r(1)} \tag{3.7}$$

$$\phi = \frac{\pi}{2} - \text{acos}\ \frac{r(3)}{|\vec{r}|} \tag{3.8}$$

$$\gamma = \frac{\pi}{2} - acos\left[\frac{\vec{r} \cdot {}^{I}\vec{V}}{|\vec{r}|\,\left|{}^{I}\vec{V}\right|}\right] \tag{3.9}$$

$$\psi = \frac{\pi}{2} \pm acos\left[\frac{\hat{e}_{1z} \cdot {}^{I}\vec{V}}{\left|{}^{I}\vec{V}\right|}\right] \tag{3.10}$$

Inertial States to Relative States

The velocity, flight path angle, and heading angle found above are defined relative to inertial space and need to be found relative to the rotating Earth. In order to do this the position and velocity as vectors must be expressed in the planet-fixed coordinate system. This is again accomplished through the use of the transformation matrices found in Chapter 3 of Hicks' text. A vehicle pointing reference frame is shown in Fig. 3.3. The position state, 'r' can be written in vector form in the vehicle-pointing reference frame as:

$$\vec{r} = \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} \hat{e}_2 \tag{3.11}$$

Similarly, the velocity can be written in vector form using a velocity reference coordinate frame $\hat{e}''$ where the $\hat{e}_y''$ axis is defined as the direction of the vehicle velocity vector.

$$\vec{V} = \begin{bmatrix} 0 \\ V \\ 0 \end{bmatrix} \hat{e}^{''} \tag{3.12}$$

These vectors can then be transformed into the Earth-fixed coordinate frame ($\hat{e}_1$) using the two transformation matrices shown below (Hicks, 2009:30-35)

$$\hat{e}_2 = \begin{bmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma)\cos(\psi) & \cos(\gamma)\cos(\psi) & -\sin(\psi) \\ -\sin(\gamma)\sin(\psi) & \cos(\gamma)\sin(\psi) & \cos(\psi) \end{bmatrix} \hat{e}'' \tag{3.13}$$

$$\hat{e}_1 = \begin{bmatrix} \cos(\phi)\cos(\theta) & -\sin(\theta) & -\sin(\phi)\cos(\theta) \\ \cos(\phi)\sin(\theta) & \cos(\theta) & -\sin(\phi)\sin(\theta) \\ \sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \hat{e}_2 \tag{3.14}$$

With the position and velocity vectors written in an Earth-fixed coordinate systems and by assuming that the atmosphere velocity doesn't vary with altitude, the relative velocity can be calculated by subtracting the cross product of the Earth's rotation and the vehicle position vector from the vehicle velocity vector (Eq. 3.15) (Hicks, 2009:393).

$$^{R}\vec{V} = {}^{I}\vec{V} - \vec{\omega}_{\oplus} \times \vec{r} \tag{3.15}$$

Where $$\omega_{\oplus} = \begin{bmatrix} 0 \\ 0 \\ 7.2921158 \times 10^{-5} \end{bmatrix}$$

Finally, the flight-path angle and heading angles relative to the rotating Earth are determined in the same manner as the flight-path angle and heading angles relative to inertial space were found but substituting the rotating Earth relative velocity for the Inertial velocity as shown in Eqs. (3.16) – (3.17).

$$\gamma = \frac{\pi}{2} - acos\left[\frac{\vec{r} \cdot {}^{R}\vec{V}}{|\vec{r}|\left|{}^{R}\vec{V}\right|}\right] \tag{3.16}$$

$$\psi = \frac{\pi}{2} \pm acos\left[\frac{\hat{e}_{1z} \cdot {}^{R}\vec{V}}{\left|{}^{R}\vec{V}\right|}\right] \tag{3.17}$$

Numerical Integration of the Equations of Motion

With the initial states defined relative to the rotating Earth. Hicks' equations of motion Eqs. (1.1) – (1.6) can be numerically integrated from t = 0 to a user specified simulation time (default is 24 hours) to determine the trajectory of the unperturbed orbit. The derivation of said equations of motion is done in Hicks' text and will not be shown here. However, the assumptions made in the said derivation are (Hicks, 2009:55):

1) Lift and drag are perpendicular and parallel to velocity, respectively.
2) Gravity acts along the r vector line
3) Mass is Constant
4) The vehicle is treated as a point mass
5) Planet rotation is constant about the north pole
6) No thrust

The integration of the equations of motion is completed using a fourth-order Runge-Kutta solver. The model was validated against actual flight data of the Apollo 10 re-entry. Figure 3.5 shows a comparison of the re-entry flight data for altitude as a function of time with the results of the 4^{th} order Runge-Kutta integration of the equations of motion (Eq (1.1) – (1.6)). During integration the deceleration and heating are calculated at each time step along the trajectory. Assumptions related to deceleration and heating are describe in the subsequent sections.

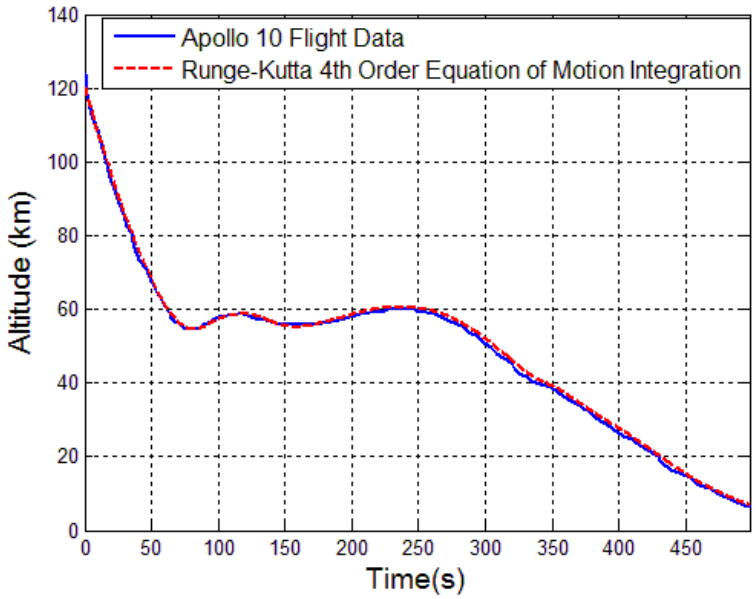

Figure 3.5. Comparison of Apollo 10 Flight Data with Numerical Integration for Model Verification Purposes

Atmospheric Model

The values for lift and drag are calculated by the following:

$$L = \frac{\rho C_L S V^2}{2} \tag{3.18}$$

$$D = \frac{\rho C_D S V^2}{2} \tag{3.19}$$

Where C_L, C_D, and S are assumed constant and defined in Table 1.1. V is defined as the magnitude of the relative velocity, and ρ is the atmospheric density. The density is calculated using a combined piecewise density model developed by Robert Bettinger. In the development of his model, Bettinger compared two different models with experimental atmospheric data derive a combined atmospheric model. First, an exponential atmospheric model was examined and is given by:

$$\rho(h) = \rho_{\oplus} e^{-\beta(h)} \tag{3.20}$$

where $\rho_{\oplus}$ is the density at sea level (1.225 kg/m^3), β is the atmospheric scale height and a constant for any given planet (for Earth β = 0.14 km^{-1}), and *h* is the altitude. The Earth is assumed to be a uniform sphere and therefore $h = r - R_{\oplus}$.

Second, the single variation model provided by Nguyen X. Vinh, Adolf Busemann, and Robert D. Culp in their book *Hypersonic and Planetrary Entry Flight Mechanics* was examined (Vinh et al., 1980:9).

$$\rho(h_{gd}) = \rho_i \left[\left(1 + \delta_H \left(\frac{h_{gd} - h_i}{R_{\oplus}} \right) \right)^{-1} \right]^{\frac{1+a}{a}} \tag{3.21}$$

where the index *i* denotes seven different sections of the atmosphere.

With these models Bettinger then compared the exponential and single variation models with the density results from the MSIS-E-90 density model within an altitude range of $0 \leq h \leq 1000$ km for the sample dates of 01 January 2000 – 2012, and the atmospheric density profile model obtained from the AGI analysis module 'Astrogator' within Satellite Toolkit. The models are plotted against each other and shown in Fig. 3.6.

Bettinger concludes that the exponential model maintains the least deviation from the MSIS and STK data up to approximately 84 km where the solution diverges and the single variation model becomes more accurate. The single variation model is only applicable up to 300 km and therefore another model is needed to model the upper reaches of the rarefied atmosphere (300 km $\leq$ h $\leq$ 1000 km). With these prediction accuracy and limitations identified, Bettinger then developed a piecewise-continuous function or 'Combined Model' described by Eq (3.22) (Bettinger, 2014: 25).

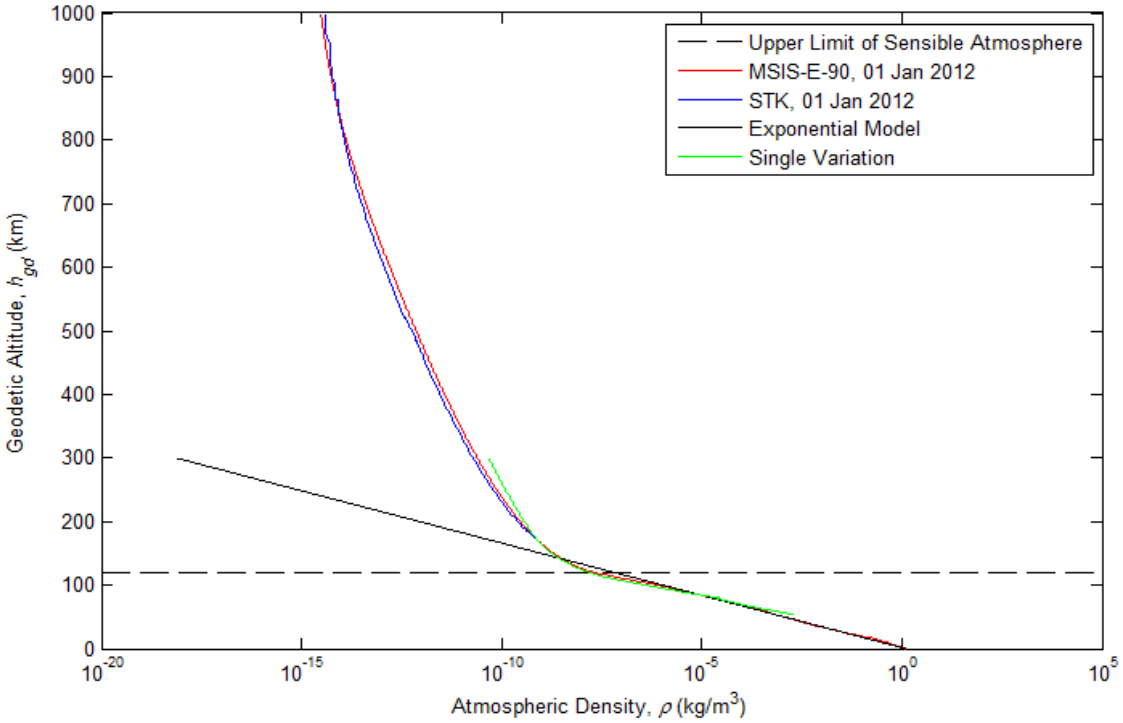

Figure 3.6. Comparison of Atmospheric Density Models with MSIS-E-90 and STK® Density Data for 01 January 2012 (Bettinger, 2014:28)

$$\rho(h_{gd}) = \begin{cases} \rho_{\oplus} e^{-\beta(h)} & , \ h < 84 \text{ km} \\ \rho_i \left[\left(1 + \delta_H \left(\dfrac{h - h_i}{R_{\oplus}} \right) \right)^{-1} \right]^{\frac{1+a}{a}} & , \ 84 \leq h \leq 120 \text{ km} \\ (4.50847623 \text{ x } 10^7) \cdot (h)^{-7.44605852} & , 120 < h \leq 1000 \text{ km} \end{cases} \tag{3.22}$$

For all altitudes above the 1000 km threshold, the density is assumed to be 0.0 kg/m^3. Parameters given in the single variation segment of Eq. (3.22) are listed in the table below:

Table 3.1. Atmospheric Density Model Parameters (Bettinger, 2014:29)

Altitude Section	h_i (km)	ρ_i (kg/m^3)	a	δ_H
$84 \leq h_{gd} \leq 90$ km	85	$7.726 \text{ x } 10^{-6}$	0.1545455	197.9740
$91 \leq h_{gd} \leq 106$ km	99	$4.504 \text{ x } 10^{-7}$	0.1189286	128.4577
$107 \leq h_{gd} \leq 120$ km	110	$5.930 \text{ x } 10^{-8}$	0.5925240	432.8484

Deceleration Calculation

The limiting conditions under which all re-entering vehicles must operate are the deceleration and atmospheric heating loads. Therefore, the calculation of these loads throughout the skip maneuver is essential in determining the validity of any given maneuver. Hicks' breaks down the deceleration into tangential (along the velocity vector) and normal (along the lift vector) (Hicks, 2009:66). These are given as:

$$(a_{decel})_v = \frac{D}{m} + gsin(\gamma) \tag{3.23}$$

$$(a_{decel})_L = \frac{-L}{m} - \left[\frac{V^2}{r} - gcos(\gamma)\right] \tag{3.24}$$

By assuming a spherical Earth we can assume that gravity is only a function of the radius, therefore a spherical Earth gravity model (Eq. (3.25)) can be used in Eqs. (3.23) – (3.24).

$$g = g_{\oplus}\left(\frac{R_{\oplus}}{r}\right)^2 \tag{3.25}$$

Where $g_{\oplus}$ is the sea-level gravitational acceleration (9.7983 m/s^2) and $R_{\oplus}$ is the radius of the Earth (6378.137 km) (Hicks, 2009:64). The overall normalized deceleration in terms of number of g's can then be calculated by:

$$(a_{decel}) = \frac{\sqrt{(a_{decel})_v + (a_{decel})_L}}{g} \tag{3.26}$$

Heating Calculation

An equation for calculating the stagnation heat flux is given by Anil V. Rao, Sean Tang, and Wayne P. Hallman in their paper, "Numerical Optimization Study of Multiple-Pass Aeroassisted Orbital Transfer."

$$\dot{Q} = 17600\left(\frac{\rho}{\rho_{\oplus}}\right)^{0.5}\left(\frac{V}{V_e}\right)^{3.15} \tag{3.27}$$

Where $\dot{Q}$ is the stagnation point heating rate in BTU/(ft^2 s), ρ is the atmospheric density at the satellites current altitude, $\rho_{\oplus}$is the atmospheric density at sea level, V is the satellites current relative velocity, and V_e is the circular orbit speed at the surface of the Earth (Rao, et al., 2002:219).

Latitude and Longitude Passes

With a method established for determining the states and loads for any given starting conditions, an algorithm is now required to determine the time of arrival and cost of overflying a specified ground target with the three maneuver types. In order to determine the needed change in ground track for these various maneuver types, the groundtrack position relative to the target position was quantified by measuring both the relative latitude difference of each target longitudinal crossing, and the relative longitude difference of each target latitudinal crossing as shown in Fig. 3.7 and 3.8. The times of these crossing relative to the start time and the direction (north or south) are also noted. Since the numerical integration provides only discrete points and not continuous values for the position of the satellite as a function of time, the values for each longitudinal and latitudinal crossing were calculated by a method of linear interpolation. Specifically, the interpolation utilized the two points directly north and south for the longitudinal crossing and the two data points directly east and west of the latitudinal crossing. These values are also used in the targeting algorithm to determine the proximity of the satellite to the target location.

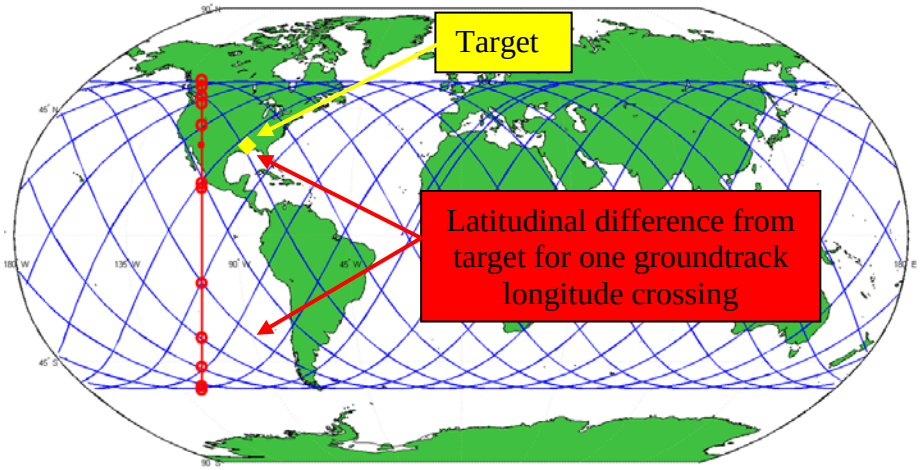

Figure 3.7. Relative Latitude Difference Points of Each Groundtrack Target Longitudinal Crossing

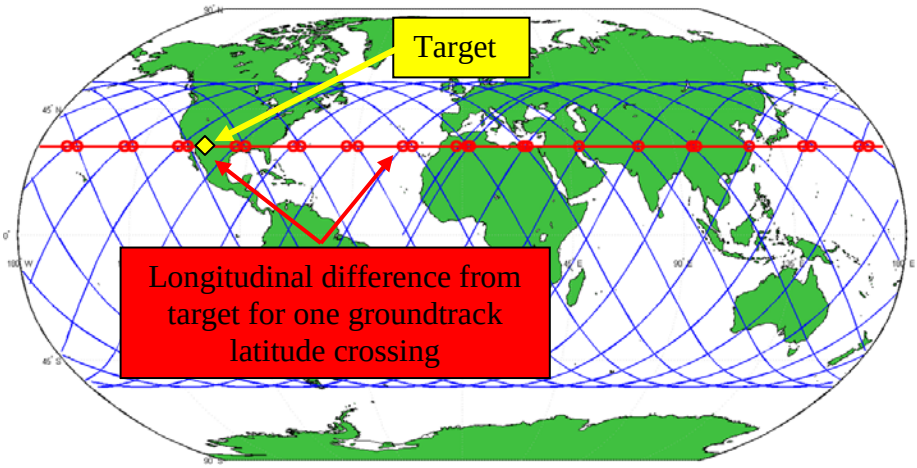

Figure 3.8. Relative Longitude Difference Points of Each Groundtrack Target Latitudinal Crossing

Phasing Manuever Algorithm

A completely planar solution to overflying the target is, in general, the least expensive type of maneuver in terms of ΔV. Due to the conservation of angular momentum, the velocity change required to either increase or decrease the semi-major axis within the plane of an orbit is significantly less than the velocity change required to

change the orbital plane. While the phasing solution is the least expensive maneuver, it is also the most solution. The solution space is limited in two very significant ways because the orbital plane is fixed in inertial space: first, overflights are only possible for targets with latitudes (north or south) less than the inclination of the orbit; second, any point on the Earth with a latitude less than the orbit's inclination will only pass through the orbital plane twice per revolution of the Earth, ie: twice per day. Therefore, no matter how the semi-major axis of an orbit is changed the time of arrival of any overflight is constrained to two distinct times for target with latitudes less than orbit inclination.

Since the target location on the Earth will only pass through the orbital plane twice per day, a phasing maneuver overflight can be achieved by either increasing or decreasing the orbital period so that the satellite overflies the target latitude at the correct time. By increasing the semi-major axis of an orbit the period is also increased thus allowing the Earth more time to rotate per revolution of the satellite. This increase in orbit period effectively moves the groundtrack westward (Fig. 3.9). Similarly, decreasing the semi-major axis will decrease the orbital period, thereby allowing less time for the Earth to rotate per revolution of the satellite, effectively shifting the groundtrack eastward (Fig. 3.10).

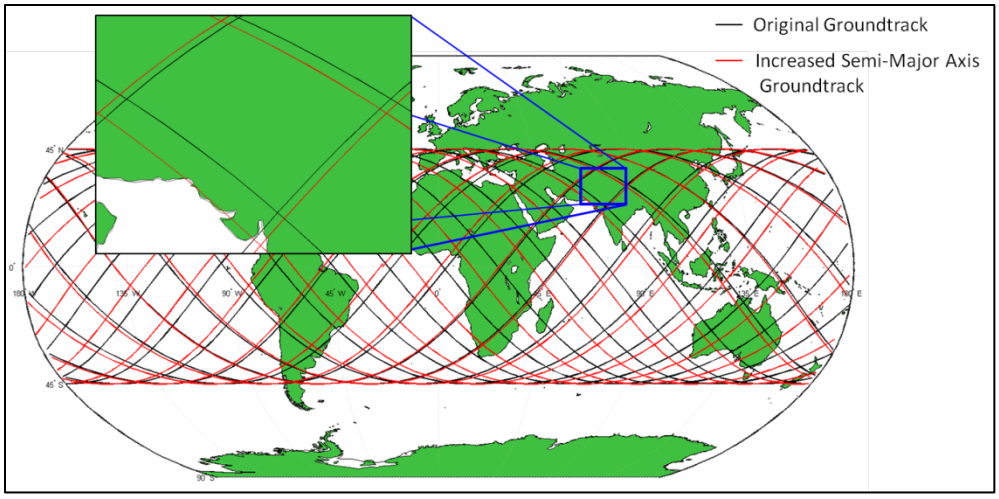

Figure 3.9. Increased Semi-Major Axis Resulting In a Westward Shift of the Groundtrack

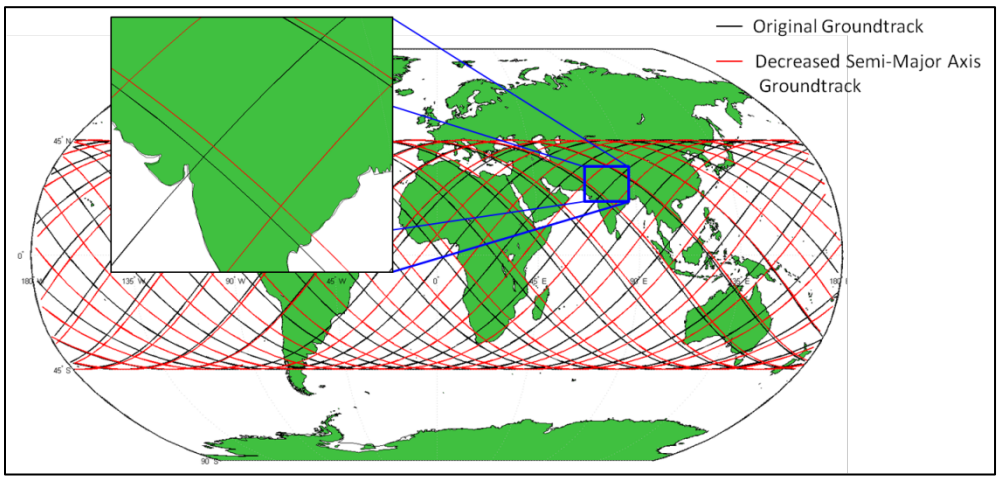

Figure 3.10. Decreased Semi-Major Axis Resulting In an Eastward Shift of the Groundtrack

The arrival time and ΔV to overfly a ground target were determined using the overflight simulation dynamics model. Fig. 3.11 shows the algorithm used within the model to determine the arrival time and ΔV.

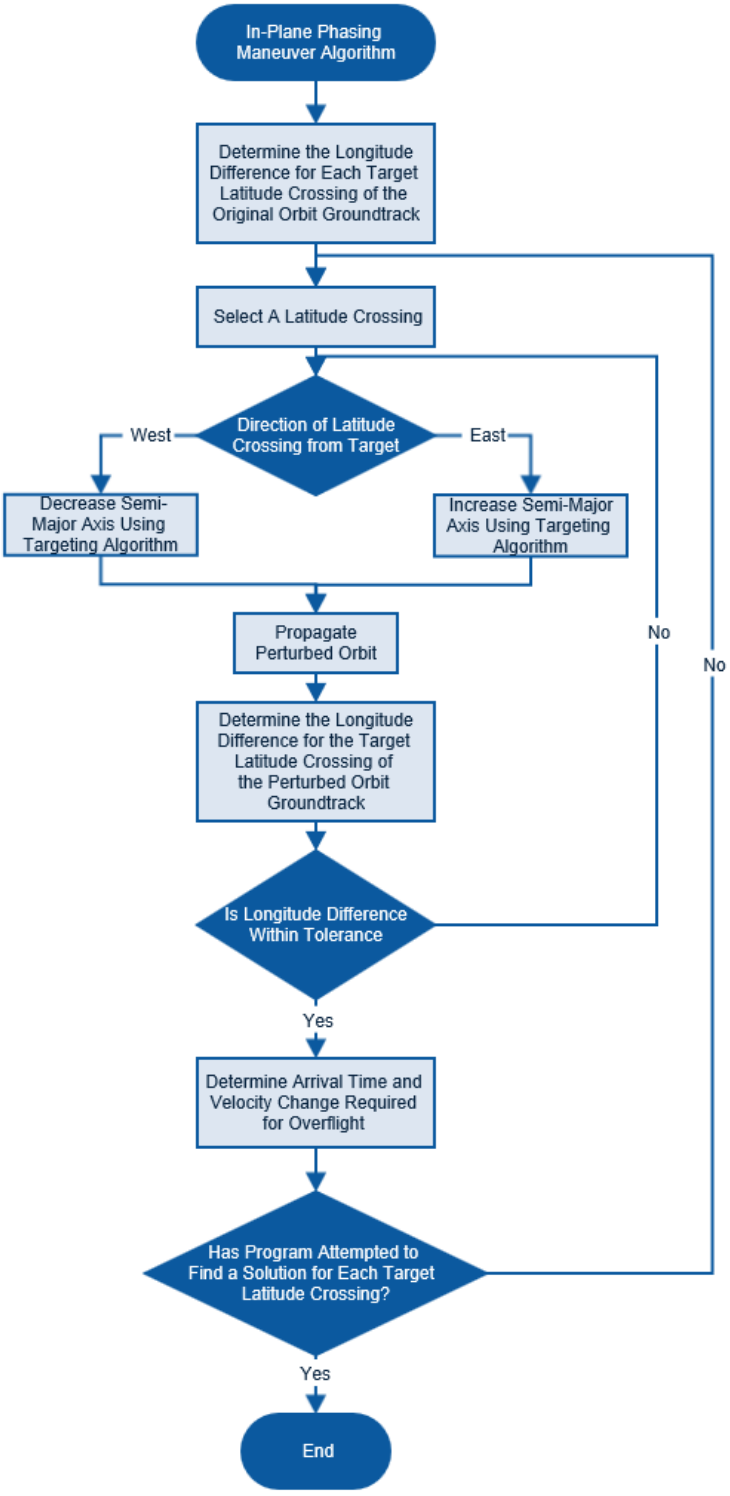

Figure 3.11. ΔV and Arrival Time Algorithm for Overflying a Ground Target Using an In-Plane Phasing Maneuver Flow Diagram

A closed from solution for the arrival time and ΔV would be beneficial both for ease of computing and validating the simulation model. An orginal analytical solution id derived below. The two times where the target location will be within the orbital plane, measured from the simulation start time until the Earth will pass under the orbital plane can be determined by first determining the longitude of the two points where the orbital plane intersects the target latitude. Finding these two longitude points can be accomplished by finding the two true anomaly values (amount of rotation since crossing the equator while heading north) which correspond to the target latitude and then using those values to determine the longitude. By expanding and simplifying Eqs. (3.1) and (3.5) for a circular orbit, the position can be written in the Geocentric Equatorial Coordinate Frame using the user input orbital elements as:

$$\vec{r} = \begin{bmatrix} \cos(\nu)\cos(\Omega) - \sin(\nu)\sin(\Omega)\cos(\mathrm{i}) \\ \cos(\nu)\sin(\Omega) + \sin(\nu)\cos(\Omega)\cos(\mathrm{i}) \\ \sin(\nu)\sin(\mathrm{i}) \end{bmatrix} \tag{3.28}$$

Substituting this value into Eq. (3.8) gives:

$$\varphi = \frac{\pi}{2} - \operatorname{acos}\left[\sin(\nu)\sin(\mathrm{i})\right] \tag{3.29}$$

Solving for true anomaly gives the first true anomaly point corresponding with the target latitude:

$$\nu_1 = \operatorname{asin}\left[\frac{\sin(\phi_{target})}{\sin(i)}\right] \tag{3.30}$$

Due to the fact that the arcsine function only returns values from $-\pi/2$ to $\pi/2$ the second true anomaly point corresponding with the target latitude is simply:

$$\nu_2 = \frac{\pi}{2} - \nu_1 \tag{3.31}$$

Using Eq.(3.7), the two uncorrected longitude points can be found as follows:

$$\theta_{\text{uncorrected }1,2} = \text{atan}\left[\frac{\cos(\nu_{1,2})\sin(\Omega) + \sin(\nu_{1,2})\cos(\Omega)\cos(\text{i})}{\cos(\nu_{1,2})\cos(\Omega) - \sin(\nu_{1,2})\sin(\Omega)\cos(\text{i})}\right] \tag{3.32}$$

These uncorrected longitude points correspond with degrees of longitude measured from the inertial x-axis (vernal equinox) and require transformation into longitude measured from the Earth prime meridian. This is done by subtracting the amount the Earth has rotated since the prime meridian and inertial x-axis were aligned from the uncorrected longitude points:

$$\theta_{1,2} = \theta_{\text{uncorrected }1,2} - \omega_{\oplus}\Delta t_{aligned} \tag{3.33}$$

Where Δt is the time measured in seconds from some known time where the inertial x-axis and the prime meridian were aligned and the simulation start time. Finally, the times corresponding to the Earth ground target being within the orbital plane can be found by equating the target longitude distance to a time difference via the rotation rate of the Earth.

$$t_{1,2} = \frac{\theta_{1,2} - \theta_{\text{target}}}{\omega_{\oplus}} \tag{3.34}$$

Since the variable, Δθ is defined from 0 to 360, 2π should be if $(\theta_{1,2} - \theta_{\text{target}}) < 0$ to make Δθ in the correct range. Combining Eqs. (3.30) – (3.34) and simplifying gives the time of arrival for all in plane maneuvers given any initial orbit as a function of target latitude and longitude (Eq.(3.35) and (3.36)). Again, *t* is defined from 0 to 24 hours, therefore if t_1 or t_2 is less than 0, 24 hours should be added to *t*.

$$t_1 = \frac{1}{\omega_\oplus}\left\{\text{atan}\left[\frac{\sqrt{1-\left(\frac{\sin(\phi)}{\sin(i)}\right)^2}\sin(\Omega)\tan(i)+\sin(\phi)\cos(\Omega)}{\sqrt{1-\left(\frac{\sin(\phi)}{\sin(i)}\right)^2}\cos(\Omega)\tan(i)-\sin(\phi)\sin(\Omega)}\right]-\omega_\oplus\Delta t_{aligned}-\theta\right\} \quad (3.35)$$

$$t_2 = \frac{1}{\omega_\oplus}\left\{\text{atan}\left[\frac{-\sqrt{1-\left(\frac{\sin(\varphi)}{\sin(i)}\right)^2}\sin(\Omega)\tan(i)+\sin(\varphi)\cos(\Omega)}{-\sqrt{1-\left(\frac{\sin(\varphi)}{\sin(i)}\right)^2}\cos(\Omega)\tan(i)-\sin(\varphi)\sin(\Omega)}\right]-\omega_\oplus\Delta t_{aligned}-\theta\right\} \quad (3.36)$$

The minimum amount of velocity change required is related to the northerly and southerly heading target latitude crossing with the minimum longitude difference divided by the amount of time for the latitude crossing to occur since the beginning of the simulation. If the longitudinal difference of each target latitude crossing, along with the time measured from simulation start time of each latitude crossing is known, then the amount of velocity change required can be determined analytically. First, the number of orbits from the start of the simulation to each particular latitude crossing needs to be calculated. This value is both the integer number of complete orbits until the latitude crossing as well as the percentage of the final orbit until the latitude crossing. Since the initial orbit is assumed to be circular with constant velocity the satellite completes the same percentage of its orbit for every unit of time:

$$N_{orbits} = \frac{t_{\varphi crossing}}{P_i} \quad (3.37)$$

Where P is the orbital period (Sellers, 2004:142):

$$P_i = 2\pi\sqrt{\frac{{a_i}^3}{\mu}} \quad (3.38)$$

Where

a_i = Un-perturbed orbit semi-major axis

μ = Earth's gravitational parameter = 3.98600442 x 10^5 km^3/s^2

The longitude difference of each target latitude crossing corresponds to a required shift in the orbital period, while N_{orbits} corresponds to the amount of time allotted for the shift to occur. Thus, the change in orbital period can be calculated as:

$$\Delta P = \frac{\Delta\theta_{\varphi crossing}}{\omega_{\oplus} N_{orbits}} \tag{3.39}$$

Now the period of the perturbed orbit can be calculated by the following:

$$P_p = P_i + \Delta P \tag{3.40}$$

Which in turn allows for the calculation of the semi-major axis of the perturbed orbit.

$$a_p = \sqrt[3]{\mu \left(\frac{P_p}{2\pi}\right)^2} \tag{3.41}$$

The velocity of the perturbed orbit can then be found by (Sellers, 2004:141-142):

$${}^{I}V_p = \sqrt{\frac{2\mu}{R_i} - \frac{\mu}{a_p}} \tag{3.42}$$

where R_i = *Initial Orbit Radius* (R_{Earth} + altitude). Since Eq. (3.42) gives the inertial velocity of the orbit, the required change in velocity for the phasing maneuver can be determined by subtracting the initial inertial velocity of the unperturbed orbit from the perturbed orbit velocity:

$$\Delta V_{phase} = {}^{I}V_p - {}^{I}V_i \tag{3.43}$$

Combining Eqs. (3.37) – (3.43) and simplifying gives the velocity change required for a ground target overflight using a phasing maneuver for any circular orbit given: the initial

altitude, initial velocity, the longitudinal distance from the target a target latitude crossing, and the time that crossing occurs measured from the maneuver start time (Eq. 3.44).

$$\Delta V_{phase} = \sqrt{\frac{\mu}{R_{\oplus} - h}\left(2 - \left(\frac{\omega_{\oplus} t_{\varphi crossing}}{\omega_{\oplus} t_{\varphi crossing} - \Delta\theta_{\varphi crossing}}\right)^{\frac{2}{3}}\right)} - {}^{I}V_i \tag{3.44}$$

Simple Plane Change Algorithm

On the opposite end of the ΔV cost solution spectrum is the "Simple Plane Change" solution. For this type of solution, multiple time of arrival solutions exist as opposed to only two time of arrival solutions for the phasing maneuver. As a consequence however, the cost in ΔV for the simple plane change can be enormous. As the cost is generally high using a simple plane change would only be done in the most extreme of circumstances and is included in the analysis for comparison purposes. The velocity change associated with a simple plane change is given in Eq. (3.45) (Bate, et al., 1971:169). Figure 3.12 shows the fuel cost for a given inclination change.

$$\Delta \mathrm{V} = 2V \sin\left(\frac{\Delta i}{2}\right) \tag{3.45}$$

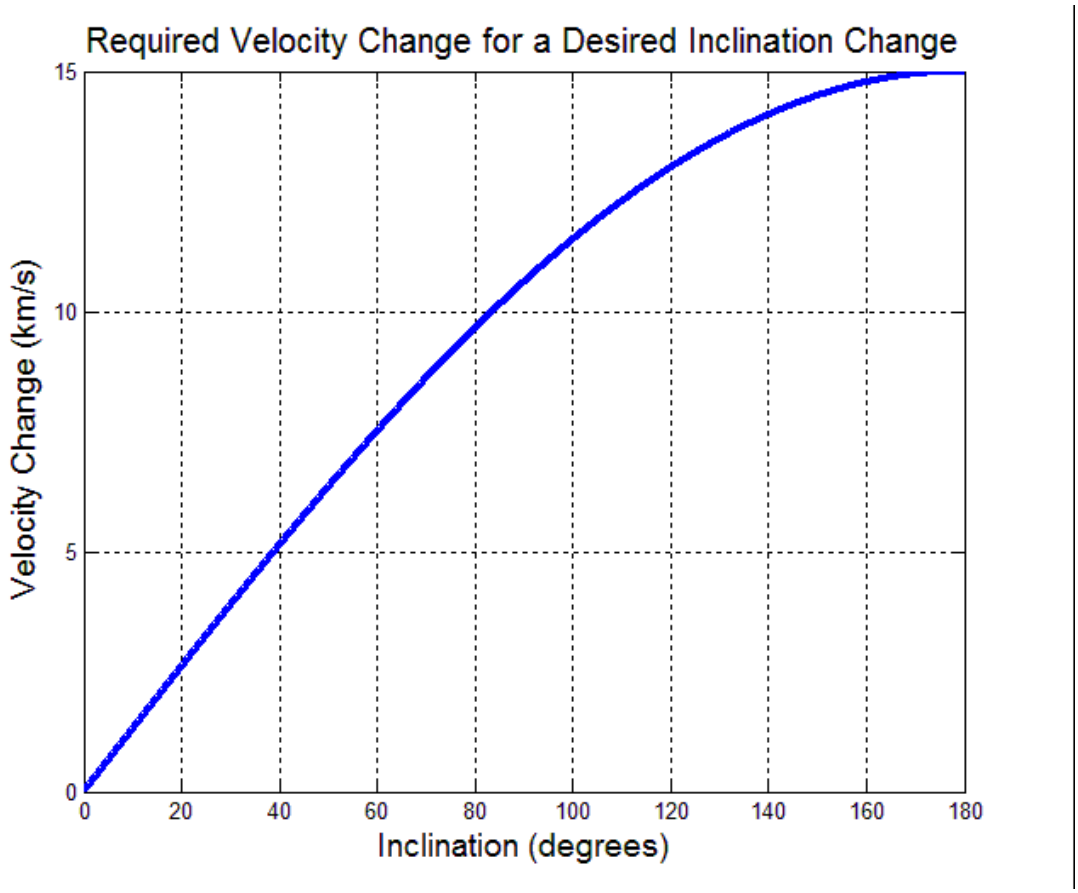

Figure 3.12. Required Velocity Change for a Simple Plane Change.

With RAAN remaining constant, the points where the groundtrack overflies the equator remain at the same longitude. Therefore, an overflight can be achieved by changing the inclination of an equatorial overflight such that the ground track is shifted to overfly the target directly. Therefore one overflight is possible per orbit of the satellite, for LEO satellites this equates to approximately one overflight every 90 minutes.

In order to determine the inclination change needed to overly a ground target, an algorithm for the change in inclination (increase or decrease) based on the groundtrack geometry is required. First, it is noted that for a given orbit inclination grade (prograde or retrograde) only target longitude crossings which occur in the same hemisphere as the

target are possible to overfly without changing the inclination grade of the orbit. Therefore, it is necessary to find all of the overflights of the orbit constraining inclination to stay in the same grade as the original orbit, and then to change the grade of the original orbit and find all of the overflights with the grade change. For each of these cases, the needed inclination direction change depends on the orbit's inclination grade, the hemisphere location of the target, and the latitude difference of the target longitudinal crossings as shown in Table 3.2.

Table 3.2. Needed Inclination Change Based on Orbit and Groundtrack Geometry

Inclination Grade	**Hemisphere**	**Relative Latitude Difference of the Groundtrack from the Target**	**Needed Inclination Change**
Prograde	North	North	(+)
		South	(-)
	South	North	(-)
		South	(+)
Retrograde	North	North	(-)
		South	(+)
	South	North	(+)
		South	(-)

Based on the preceding heuristics the amount of inclination change can be determined iteratively by varying inclination until the relative latitude difference of the groundtrack from the target for each target longitude crossing is within a set tolerance. The amount of velocity change required for the inclination change can then be determined via Eq. (3.45). Figure 3.13 shows the flow diagram for the simple plane change algorithm within the overflight simulation dynamics model.

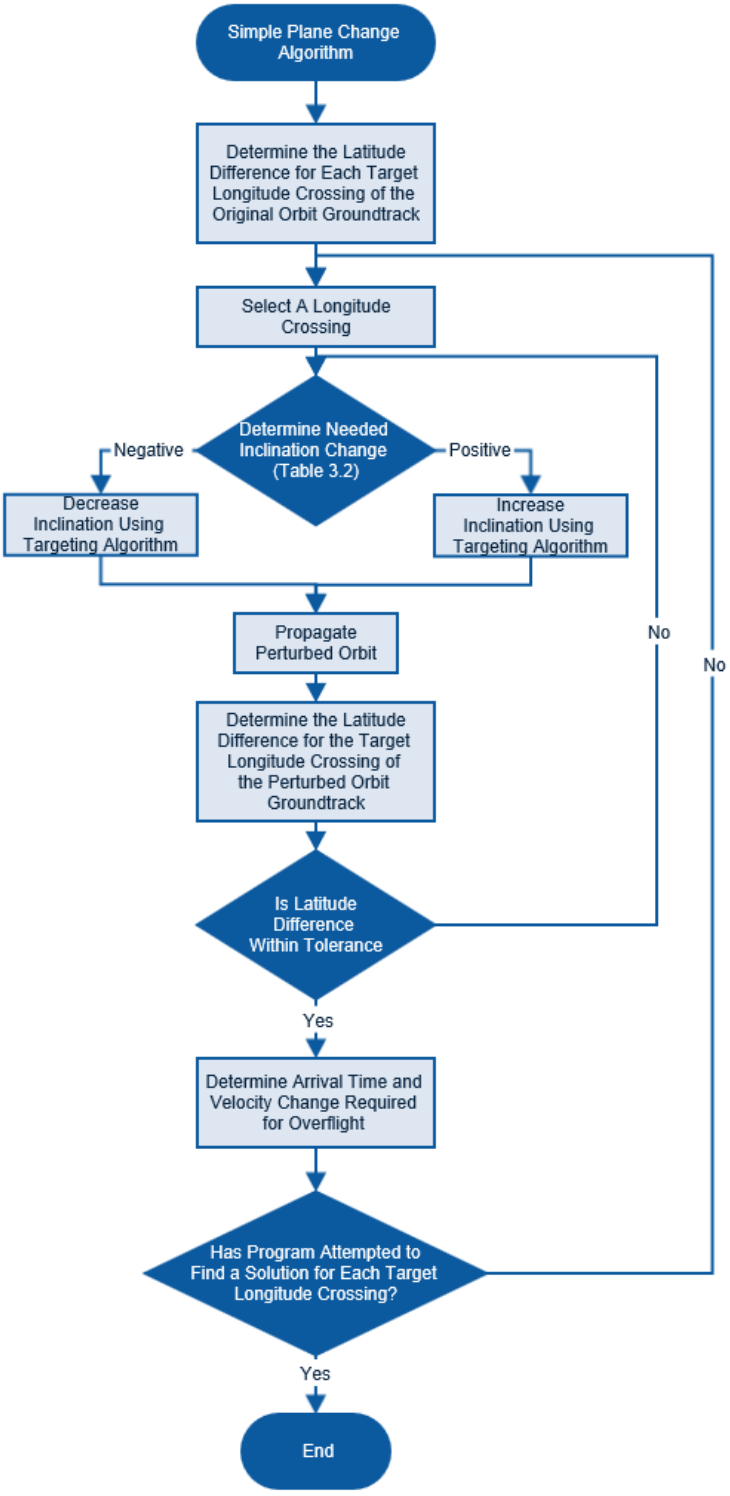

Figure 3.13. ΔV and Arrival Time Algorithm for Overflying a Ground Target Using an In-Plane Phasing Maneuver Flow Diagram

Aeroassisted Maneuver Algorithm

Both phasing maneuvers and simple plane change maneuvers have been used extensively throughout space flight and are reasonably well understood. The purpose of this study is to determine the value of atmospheric skip maneuvers and the conditions under which they are a viable option. It is therefore necessary to compare the atmospheric skip maneuver directly with the phasing maneuver and simple plane change maneuvers. The atmospheric skip maneuver or aeroassisted maneuver shares many similar advantages and disadvantages with both the phasing and the simple plane change maneuvers. For most orbits and target locations, the aeroassisted maneuver has more than two time-of-arrival solutions and in general costs less in terms of ΔV than does the simple plane change.

The aeroassisted maneuver is completed by first conducting an initial deboost burn which decreases the semi-major axis of the orbit so that perigee is within the sensible atmosphere. While within the atmosphere, the TAV banks and uses aerodynamic forces to modify its orbit. Once the vehicle reaches trajectory apogee, it is commanded to either re-circularize at the apogee altitude, which will be less than the original circular altitude, or it is commanded to re-circularize at the original altitude via a Hohmann transfer. The total velocity change required for the maneuver includes the velocity needed for the de-orbit burn as well as the velocity needed to re-circularize at either the decayed apogee altitude the original altitude.

The total velocity change is related to the velocity needed for the initial deorbit burn. The larger the decrease in velocity, the more atmosphere the TAV will penetrate allowing it to more dramatically change its orbit. The deeper penetration of the

atmosphere will also cause the vehicle to lose more of its kinetic energy within the atmosphere thereby requiring a greater amount of ΔV to re-circularize. The amount of atmosphere penetration required is a function of how much groundtrack change is needed to overfly the target.

The effect of the aeroassisted maneuver on the orbit depends again on the geometry of the ground track when the maneuver is performed and the direction of bank. The effect on the groundtrack for maneuvering at various times through the orbit is shown in Fig. 3.14 and Table 3.3. By understanding both the effect of banking at various times throughout the orbit, and the relative latitudinal difference of each target longitude crossing, a heuristic-based algorithm can be created to determine at what maneuver section to bank and in what direction to bank. This algorithm is shown in Table 3.4.

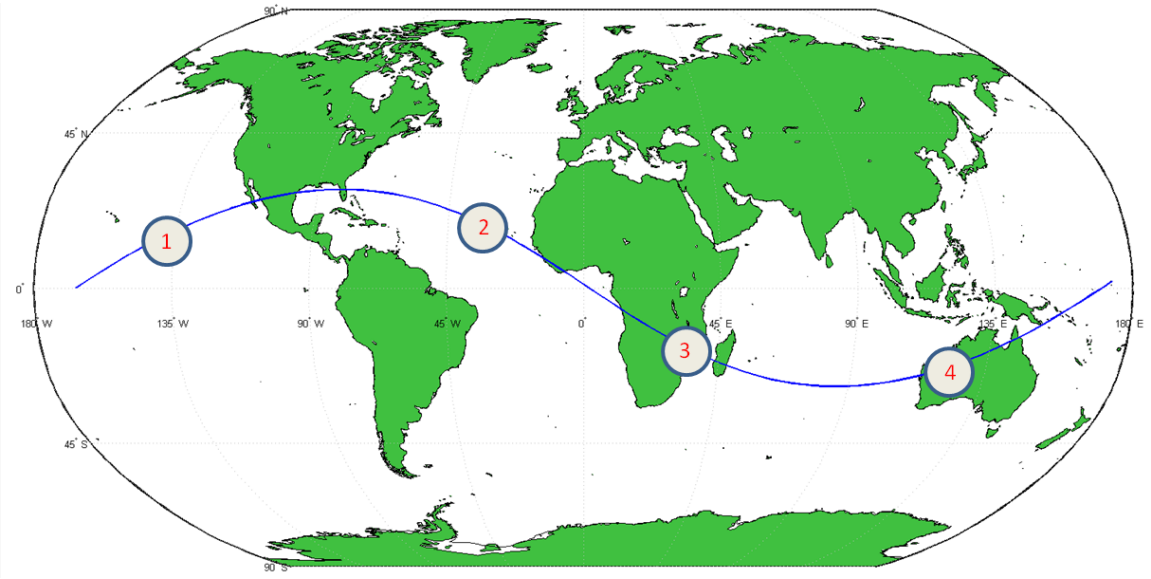

Figure 3.14. Maneuver Sections Used in Determining the Effect of Banking Within Atmosphere at Varying Times Throughout Orbit.

Table 3.3. Effect of Banking at Various Maneuver Sections on Orbit Inclination and RAAN.

Maneuver Section	Bank	Effect on Inclination	Effect on RAAN
1	(+)	(-)	(-)
	(-)	(+)	(+)
2	(+)	(-)	(+)
	(-)	(+)	(-)
3	(+)	(+)	(+)
	(-)	(-)	(-)
4	(+)	(+)	(-)
	(-)	(-)	(+)

With the maneuver section and bank direction set, it now becomes possible to iterate the initial deorbit boost until the latitude difference of each target longitude pass is within pre-determined tolerances. This process, however, is more complicated than it was for the phasing and simple plane change maneuver calculations. First, the maneuver section is known, but the time required to enter the atmosphere given an initial decrease in orbital velocity is not. To determine this time, the velocity is decreased at t = 0 and the equations of motion are then integrated forward in time. The time until the TAV enters the sensible atmosphere (122 km) is then measured and defined as 'maneuver time.' The maneuver start time (the time in the orbit when the velocity decrease should occur in order to enter the atmosphere at the desired maneuver section) can then be determined by subtracting the maneuver time from the time associated with the maneuver section (Eq. 3.46).

$$t_{start} = t_{section} - t_{maneuver} \tag{3.46}$$

Table 3.4. Aeroassisted Heuristics in Determining Maneuver Section and Bank Direction

Inclination Grade	Target and Groundtrack in the Same Hemisphere?	Hemisphere of the Groundtrack Target Longitude Crossing	Heading of the Groundtrack Target Longitude Crossing	Relative Latitude Difference of the Groundtrack from the Target	Maneuver Section for (+) Bank	Maneuver Section for (-) Bank
Prograde	Yes	North	North	North	2	4
				South	4	2
			South	North	3	1
				South	1	3
		South	North	North	1	3
				South	3	1
			South	North	4	2
				South	2	4
	No	North	North	North	3	1
				South	3	1
			South	North	2	4
				South	2	4
		South	North	North	2	4
				South	2	4
			South	North	3	1
				South	3	1
Retorgrade	Yes	North	North	North	3	1
				South	1	3
			South	North	2	4
				South	4	2
		South	North	North	4	2
				South	2	4
			South	North	1	3
				South	3	1
	No	North	North	North	2	4
				South	2	4
			South	North	3	1
				South	3	1
		South	North	North	3	1
				South	3	1
			South	North	2	4
				South	2	4

Next, the equations of motion are integrated again but with the velocity decrease occurring at t_{start} rather than t = 0. This integration however does not include the re-circularization at apogee following the skip. If left without re-circularization, the TAV continues to re-enter the atmosphere, losing energy each pass, thereby lowering the semi-major axis until the TAV impacts the Earth. Therefore, all six state variables need to be determined at the first apogee after skip. These states are calculated by interpolating between the two points where the heading angle transitions from positive to negative in the trajectory data arising from the numerical integration. The velocity as well as the heading angle needed to re-circularize the orbit at this point need to be determined. The circular velocity is determined by setting the change in flight path angle to 0 in Eq. (3.47) with thrust equal to 0, then solving the resulting quadratic equation for the velocity (Hicks, 2009:52)

$$
\begin{aligned}
{}^{R}V\dot{\gamma} = \frac{L}{m}\cos\sigma - g\cos\gamma + \frac{{}^{R}V^{2}}{r}\cos\gamma + 2\,{}^{R}V\omega_{\oplus}\cos\phi\cos\psi + \\
r\omega_{\oplus}^{2}\cos\phi\,(\cos\phi\cos\gamma - \sin\phi\sin\psi\sin\gamma) = 0
\end{aligned}
\tag{3.47}
$$

The heading angle is calculated by iterating the initial heading angle until the maximum latitude reached by the circularized orbit is the same as the maximum latitude reached by the un-circularized orbit. The post-circularized orbit trajectory data is combined with the pre-circularized data at the apogee point to create one set of correct trajectory data. The latitude difference of each target longitude crossing of the skip maneuver trajectory can then be calculated. The initial decrease in velocity is either increased or decreased according to the heuristics in Table 3.4 and the entire process is repeated again (Fig 3.15).

Algorithm iteration continues until either the latitude difference for each target longitude crossing is within an accepted tolerance or the maneuver exceeds pre-determined limits imposed on deceleration and heating.

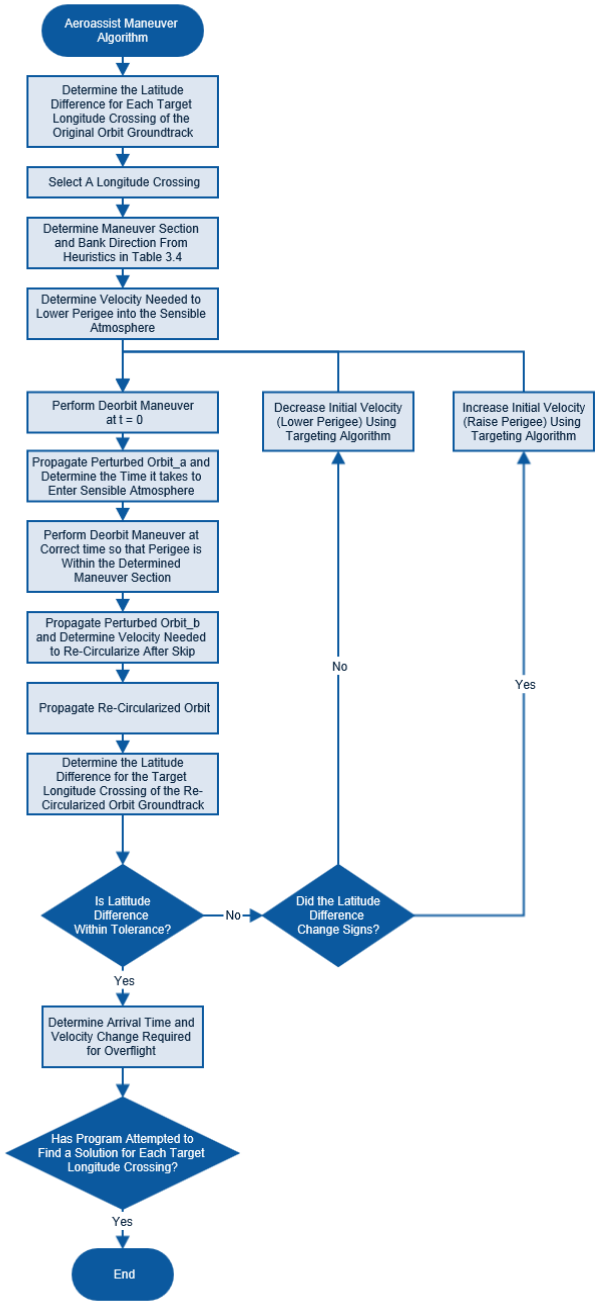

Figure 3.15. ΔV and Arrival Time Algorithm for Overflying a Ground Target Using an Aeroassisted Maneuver Flow Diagram

Mapping Aeroassisted Trade Space

Given any initial circular orbit, target location, and simulation time, the time of arrival and velocity change required for every possible overflight using the three types of maneuvers can be determined. In order to determine the conditions which would encourage the use of the aeroassisted maneuver, multiple different target locations and initial orbits were simulated and subsequently analyzed to ascertain whether data trends exist that could yield closed-form functions enabling and approximation of the available options via the three types of maneuvers. The search space is quite large and includes varying initial semi-major axis, initial orbit inclination and RAAN, as well as target longitude and target latitude. A test matrix was developed and is included in Appendix B.

Reachability

The uses of aeroassisted maneuvers are varied and include more than modifying an orbit in order to overly a target on the Earth. The maneuver can also be used to change a satellite's orbit so as to rendezvous with another satellite. Another use of the aeroassisted maneuver is for initial orbit insertion. The lowest inclination any satellite can launch to is the latitude of its launch site because a satellite must overfly its launch site. Therefore, if a satellite mission requirement is for the satellite to be at a lower inclination than the launch site latitude, a simple plane change is required once in orbit to lower the orbit's inclination. Such a maneuver can be tremendously expensive as shown in Fig. 3.12. Performing an aeroassisted maneuver instead of the traditional simple plane change has the potential of dramatically offsetting costs for initial orbit insertion.

With these motivations in mind, a study of orbit reachability using an aeroassisted maneuver was completed. For the purposes of this research reachability is defined as the

inclinations and RAANs reachable when using an aeroassisted maneuver from a given initial orbit. As stated earlier, the effects on inclination and RAAN using an aeroassisted maneuver depend on the geometry of the orbit during the maneuver. Therefore, the skip maneuver algorithm described above was modified to measure the inclination and RAAN change of an orbit for 12 different start times (Fig. 3.16), as well as multiple initial velocity decreases. The initial velocity ranged from the velocity needed to enter the sensible atmosphere, to the velocity that would overload the TAV structurally (10 g's). The reachability test matrix is also included in Appendix B.

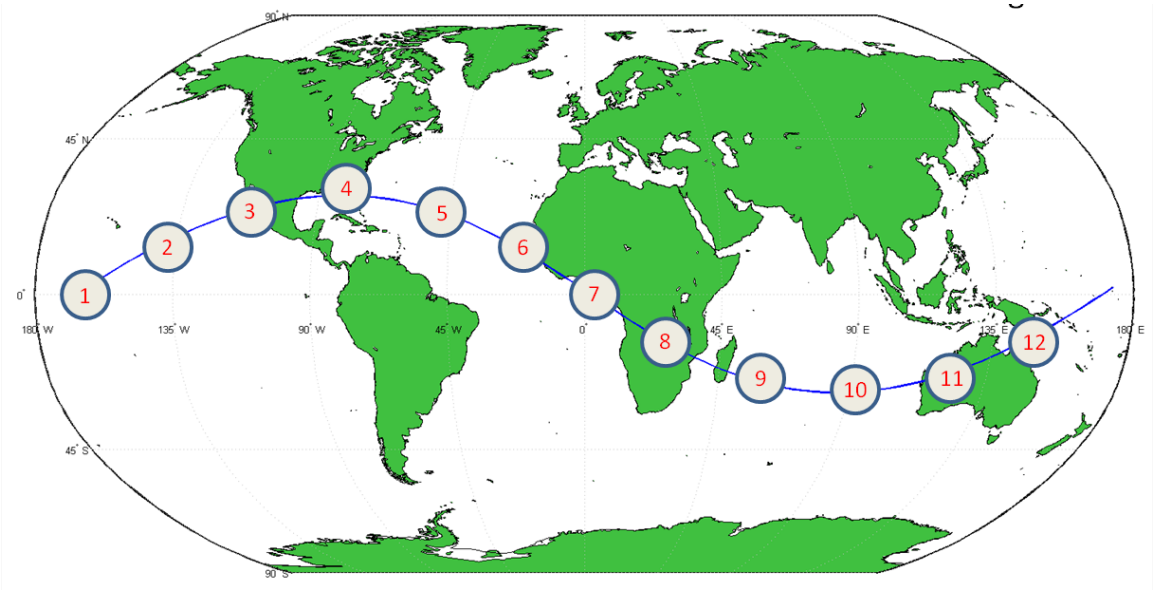

Figure 3.16. Maneuver Start Times Defined for Inclination and RAAN Reachability Study

Summary

The method for obtaining the results discussed in Chapter 4 have been detailed. Given an initial state, the equations of motion are numerically integrated using a 4^{th} order Runga-Kutta method to produce the orbital trajectory. An equation for calculating the time of arrival of any phasing maneuver solution was derived as well as a method for

determining the associated change in velocity. The change in velocity and time or arrival was determined iteratively for target overflights via a simple inclination plane change as well as an aeroassisted maneuver. Finally a method for determining the reachability of any given orbital plane through the use of aeroassisted maneuvers was determined and detailed.

IV. Analysis and Results

Chapter Overview

This chapter will present the results of the dynamics overflight model including the ΔV costs and time of overflights using all three maneuver types. The data for multiple initial orbits as well as target locations on the Earth will be analyzed and the resulting trends will be presented. Methods for determining preliminary estimates of the cost and time functions without extensive computing will be discussed and verified against data gained from the dynamics model. The results of the reachability study will also be outlined and analyzed, with these results being compared to traditional methods to determine overall ΔV cost savings via aeroassisted maneuvers.

Results of Ground Target Overflight Simulations

In-Plane Phasing Maneuver

The timing of any in-plane phasing maneuver overflight solution will have significant impact on the ΔV cost and timing of any simple inclination plane change or aeroassisted maneuver. Equations (3.35) – (3.36) give the timing of any in-plane solution for a given starting orbit, start time, and target location. Figures (4.1) – (4.3) show the time of arrival of any in-plane solution as a function of target latitude. The target longitude and the initial orbit inclination and RAAN change for each figure and are noted in the figure heading.

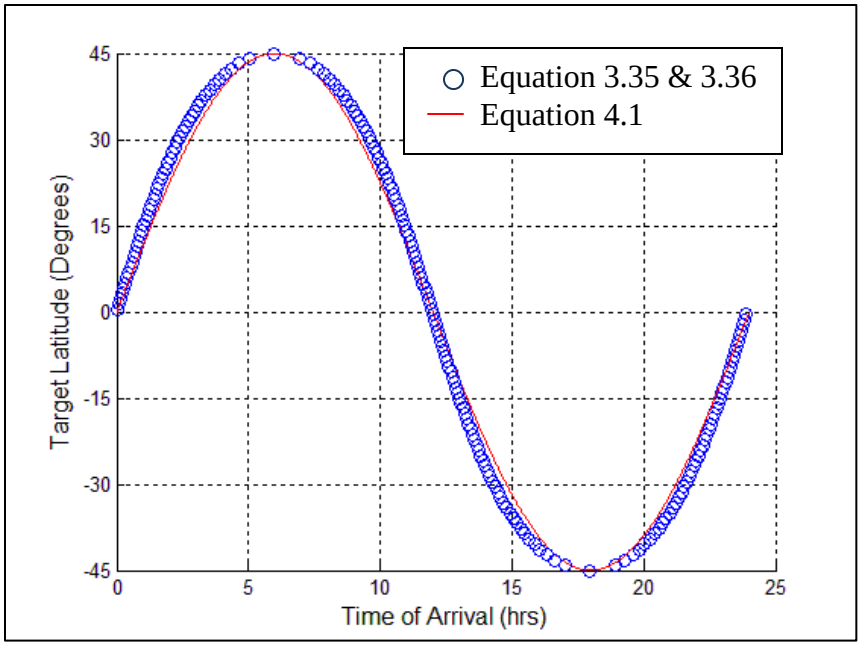

Figure 4.1. Time of Arrival Using and In-Plane Solution with Varied Target Latitude with Inclination = 45°, Target Longitude = 0°, RAAN = 0°, and Time from Inertial/Planet Fixed Coordinate Systems Alignment = 0 hours

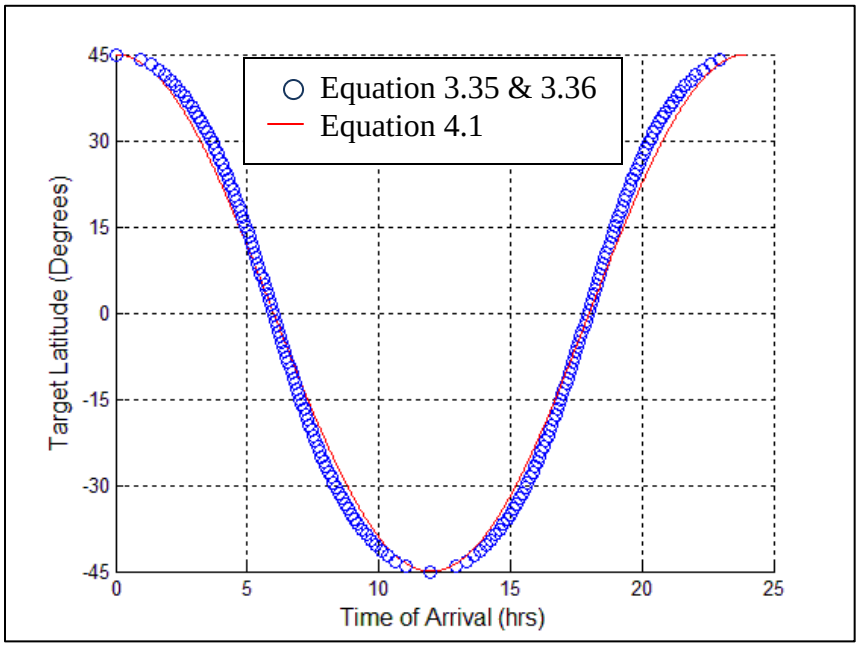

Figure 4.2. Time of Arrival Using and In-Plane Solution with Varied Target Latitude with Inclination = 45°, Target Longitude = 90°, RAAN = 0°, and Time from Inertial/Planet Fixed Coordinate Systems Alignment = 0 hours

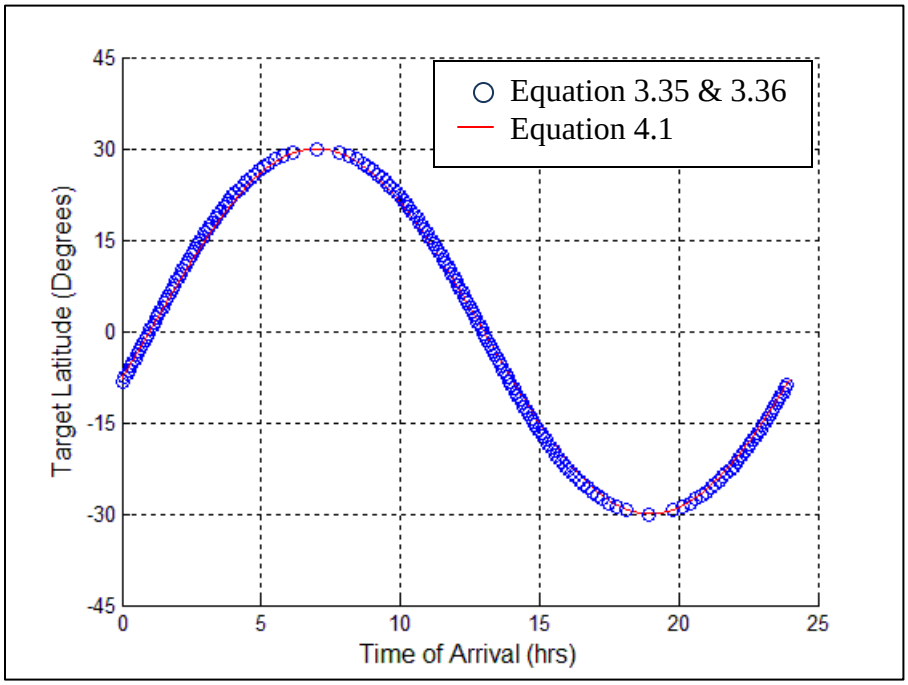

Figure 4.3. Time of Arrival Using and In-Plane Solution with Varied Target Latitude with Inclination = 30°, Target Longitude = 0°, RAAN = 40°, and Time from Inertial/Planet Fixed Coordinate Systems Alignment = 2 hours

Observing the effect of target latitude, target longitude, RAAN, and inclination on the shape of time of arrival indicates that the latitude of a given time very nearly approaches Eq. (4.1) below:

$$\phi = i \cdot \sin\left(\frac{\pi t}{12} + \theta - \Omega - \omega_{\oplus}\Delta t_{aligned}\right) \tag{4.1}$$

Equation (4.1) can be solved for the time of arrival as a function of target location, orbit, and time from inertial/planet-fixed coordinate system alignment, yielding Eq (4.2).

$$t = \sin^{-1}\left(\frac{\phi}{i}\right) + \Omega + \omega_{\oplus}\Delta t_{aligned} - \theta - \frac{\pi t}{12} \tag{4.2}$$

The results of the dynamics model simulation in finding both the cost and time of arrival using an in-plane phasing maneuver solution agree with the time of arrival solutions shown above. Figure 4.4 shows the results of the in-plane phasing maneuver dynamics model simulation for an initial orbit and target location.

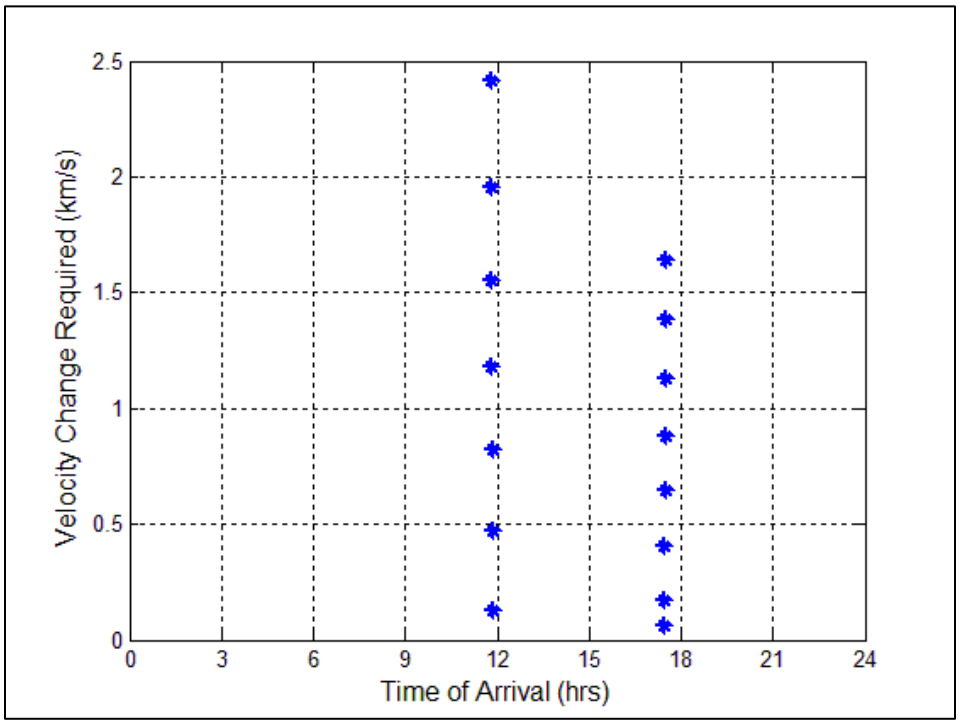

Figure 4.4. Velocity Change Cost and Time of Arrival to Overfly Tehran, Iran Using an In-Plane Phasing Maneuver from an initial orbit with initial parameters: Alt = 500 km, Inclination = 45°, Longitude = 0°, Latitude = 0°.

As shown in Fig. 4.4, all solutions are divided into one of two arrival time-bins and from these bins the only solutions of interest are the two minimum ΔV solutions. The two minimum velocity changes required for the case above are 0.129 km/s for the near 12 hr-solution and 0.065 km/s for the near 18-hr solution. The higher ΔV solutions represent a large requisite change in longitude with a short amount of time for the groundtrack change (ie, performing the longitude change in 1 orbit as opposed to 5 orbits).

The analytic solution for the velocity change required for a phasing maneuver described by Eq. (3.44), along with the analytic solution for the time of arrival solutions to the phasing maneuver (Eqs. (3.35) and (3.36)) were plotted against the simulation model data as shown in Fig. 4.5. As shown, The two solutions closely approximate each other with the percent error of the minimum ΔV solutions being 1.2% for the near-12 hour solution and 1.8% for the near 18-hr solution.

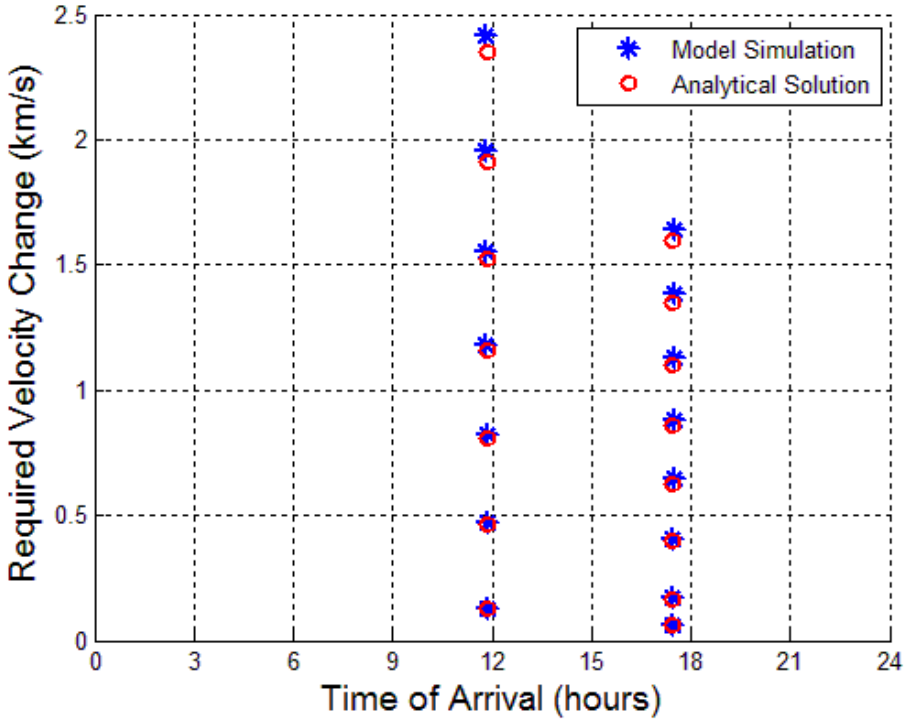

Figure 4.5. Comparison of the Dynamics Simulation Solution to the Analytical Solution for a Phasing Maneuver Ground Target Overflight of Tehran, Iran; Given the Initial Circular Orbit with: Alt = 500 km, Inclination = 45°, Longitude = 0°, and Latitude = 0°

Simple Plane Change Maneuver Results

As a ground target overflight solution exists via a simple plane change every orbit, and the semi-major axis for this type of maneuver remains constant, the time of arrival of all solutions will be a multiple of the orbital period plus/minus some function of the inclination change. If it is assumed that the change in time of arrival due to the inclination change is small then we can approximate the time of arrival as simply a multiple of the orbital period. This can be seen in the solution produced by the dynamics model simulation for overflying Tehran via an inclination plane change (Figure 4.6). The minimum ΔV solutions occur near the same time as the two in-plane phasing maneuver solutions. The time of arrival of the first overflight is approximately:

$$TOA_1 = \frac{\Delta\nu}{2\pi} P_i \tag{4.3}$$

where Δν is the change in true anomaly from the maneuver start time to the first overflight of the target latitude. The time of arrival can then be approximated as:

$$TOA_N = \frac{\Delta\nu}{2\pi} P_i + (N-1)P_i \tag{4.4}$$

where N is an integer corresponding with the N^{th} overflight.

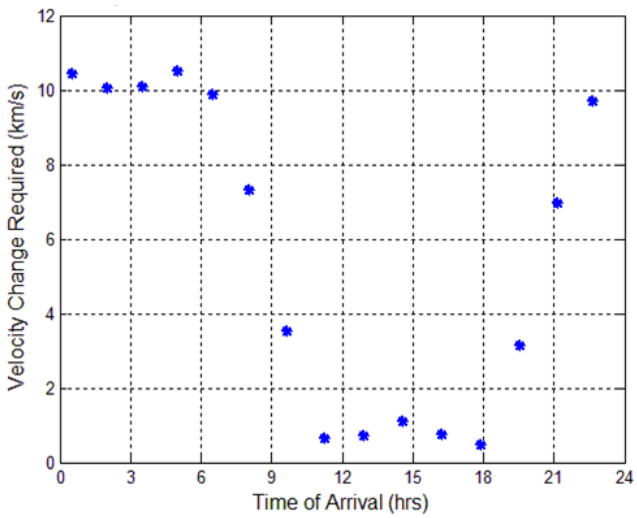

Figure 4.6. Simple Plane Change Maneuver Ground Target Overflight of Tehran, Iran; Given the Initial Circular Orbit of: Alt = 500 km, Inclination = 45°, Longitude = 0°, and Latitude = 0°

The velocity cost and timing of ground target overflights via a simple plane change were calculated using the dynamics model simulation over a wide range of initial orbits and target locations in order to determine the velocity cost as a function of initial orbit and target location. The resulting data shows a similar shape trend in the dv vs. toa plot for any input. Although a successful overflight is not achievable for any given time of arrival, it will be useful to map the discrete data points to a continuous function of time of arrival. Once this function is created the discrete points can be found again by using only the arrival times available from Eq. (4.4).

The data shows that the velocity change as a function of time of arrival is sinusoidal for target latitudes greater than the initial orbit inclination. For target latitudes less than the orbit inclination the function acquires two 'bumps' as can be seen in Fig. 4.6 from 0 to 5 hours and from 12 to 18 hours. The end of the lower bump correspond with the timing solutions of the phasing maneuver (Fig. 4.5). In order to better visualize this

trend with change in target latitude, Fig. 4.7 shows a three-dimensional surface of the velocity change cost vs. time of arrival vs. target latitude. The surface can be described target latitude varying on the axis in and out of the page of Fig. 4.6, while all other input parameters remain constant. While it is useful to plot the velocity change as a continuous function of arrival time it is important to note that in actuality only discrete time of arrivals are possible. The black dots in Fig. 4.7 are the actual possible discrete time of arrival overflights while the rest of the surface is for visualization of the trends only. The highest cost is for targets close to the equator while the lowest costs are for targets close to the initial orbit inclination. It can also be observed that trends in the southern hemisphere are exactly the inverse mirror of the trends in the northern hemisphere. Changes in target longitude or initial orbit RAAN while keeping other inputs constant corresponds with a horizontal shift of the ΔV vs. toa plot (e.g. Fig. 4.6).

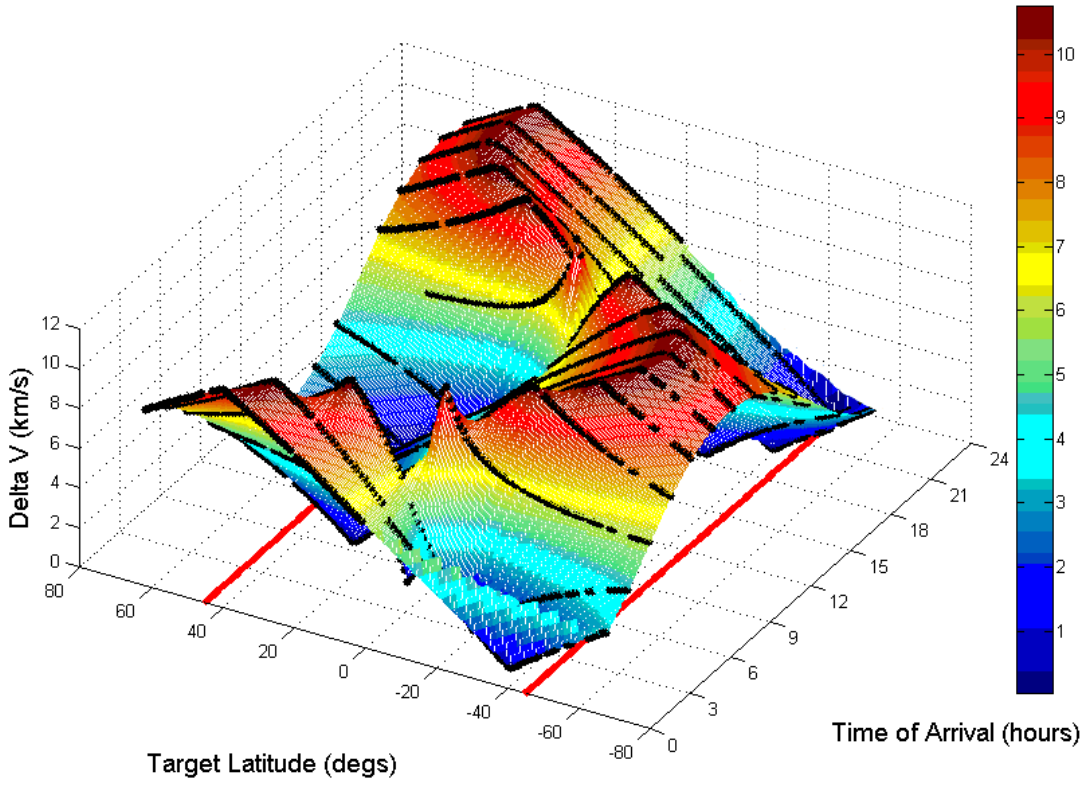

Figure 4.7. Velocity Change Cost of Overflying Ground Target for a Desired Arrival Time and Target Latitude Using a Simple Plane Change Maneuver From Initial Orbit: Alt = 500km, Inclination = 45°, Longitude = 0°, and Latitude = 0°

With these observations made it is possible to define one function that fits all data for the inclination plane change maneuver. It can be shown that a function of the type shown in Eq. (4.5) matches well with all data collected:

$$\Delta V_{plane} = A\left|\sin\left(\frac{t_{arrival}2\pi}{24} + B\right) + \frac{C}{A}\right| \tag{4.5}$$

with the correction:

If: $\Delta V_{plane} > 2V_{initial}\sin\left(\frac{\pi}{4}\right)$

Then: $\left(\Delta V_{plane}\right)_{new} = 4V_{initial} sin\left(\frac{\pi}{4}\right) - \left(\Delta V_{plane}\right)_{old}$

where A and C are functions of target latitude, and orbit inclination and B is a function of target longitude and orbit RAAN. The variables A, B, and C were found iteratively using a genetic algorithm function for many different target latitude/longitudes and initial orbits. The genetic algorithm attempts to find the global minima of the error function shown in Eq. (4.6) for each case of input parameters by guessing an initial A, B and C and then 'evolving' them toward an optimal solution. The optimal A, B, and C where found for multiple different initial input cases in order to find trends in A, B, and C with input parameters. Figure 4.8 shows the A, B, and C found which minimize the error function for multiple different target latitudes.

$$error = \left\| \left(\Delta V_{plane}\right)_{data} - A \left| \sin\left(\frac{t_{arrival} 2\pi}{24} + B\right) + \frac{C}{A} \right| \right\| \tag{4.6}$$

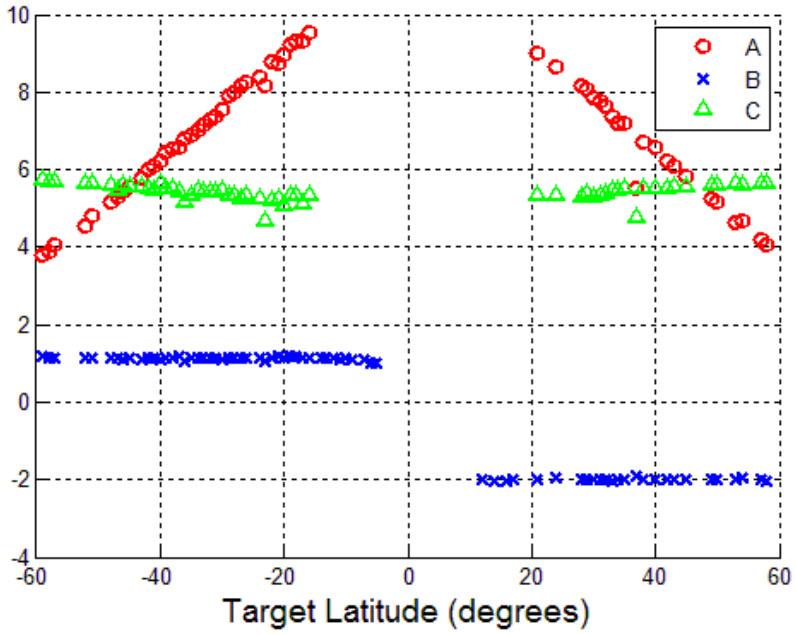

Figure 4.8. A, B, and C Which Minimize the Error Function (Eq 4.14) for Various Target Latitudes

A least squares fit of the data in Fig. 4.8 gives:

$$A(\phi) = -0.1348\left|\phi_{target}\right| + 11.7 \tag{4.7}$$

$$B = \begin{cases} 1.15 & ,\phi_{target} < 0 \\ 1.15 - \pi & ,\phi_{target} > 0 \end{cases} \tag{4.8}$$

$$C = 5.634 = A(i) \tag{4.9}$$

By using similar methods while varying different input parameters, it can be shown that the velocity cost for a ground target overflight via an inclination plane change as a function of arrival time is:

$$\Delta V_{plane} = \left(-0.1348\left|\phi_{target}\right| + 11.7\right)\left|\sin\left(15t_{arrival} + \theta_{target} - \Omega + 180k\right) + \left(\frac{-0.1348i + 11.7}{-0.1348\left|\varphi_{target}\right| + 11.7}\right)\right| \tag{4.10}$$

where all angular values are in degrees, $t_{arrival}$ is in hours and k = 1 for targets in the northern hemisphere and 0 for targets in the southern hemisphere. Note: $t_{arrival}$ is not continuous but is a discrete set of values determined from Eq. (4.4). Equations (4.4) and (4.10) were used to calculate the time of arrival and velocity change cost for the same scenario as found in Fig. 4.6 and plotted against the data from the simulation model (Fig. 4.9). Equation (4.10) gives accurate results for the velocity change cost for a given time of arrival for the lower range of velocity cost. The accuracy deteriorates for the upper region of the sine curve as the actual curve more closely approximates a square wave than a sine wave, however, the region of greatest interest is the lower ΔV region where

model accuracy improves. The RMS error is 0.61 km/s for the ΔV solution and 0.19 hours for the timing solution. The greatest inaccuracies stem from the inprecise time of arrival estimation (Eq. 4.4). An improvement in the time of arrival model would significantly improve the accuracy of the velocity calculation. With accurate timing solutions the RMS error for the ΔV reduces to 0.543 km/s.

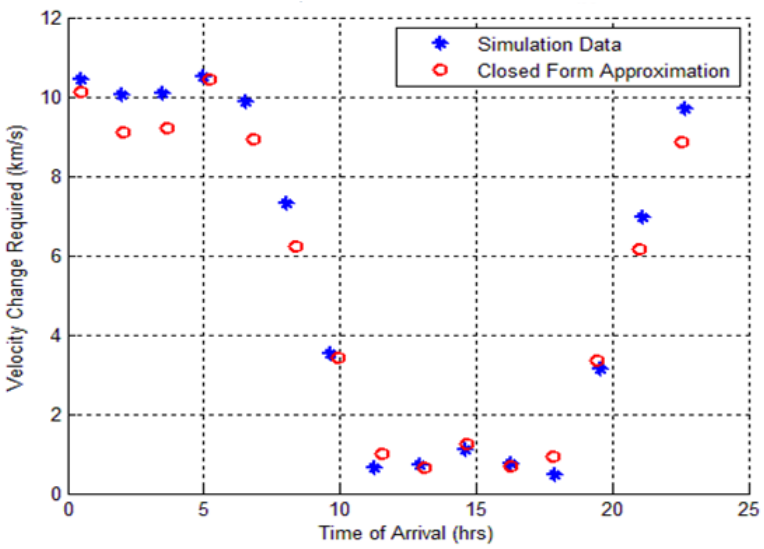

Figure 4.9. Comparison of Eqs. (4.4) & (4.10) to Simulation Data for Tehran, Iran Overflight Solutions from an Initial Orbit of: Alt = 500 km, Inclination = 45°, Starting Longitude = 178° W, Starting Latitude = 0°

Aeroassisted Maneuver Results

The time of arrival model for the aeroassisted maneuver is similar to that of the plane change maneuver, with the exception that for certain arrival times, the corresponding velocity change would cause a violation of deceleration loads. As a result, the time of arrival points can be calculated by first using Eq. 4.11 to calculate all possible arrival times, then calculate the corresponding velocity changes using the model

developed below, and finally eliminating those arrival times which require a velocity change greater than the allowable change that would exceed 10 g's deceleration.

$$TOA_N = \frac{\Delta v}{2\pi} P_i + (N-1)P_i \tag{4.11}$$

The simulation results for the aeroassisted maneuver are plotted against the results for the phasing and plane change maneuvers in Fig. 4.10. It is readily apparent that the aeroassisted maneuver offers unique advantages over the other two types of maneuvers. More arrival times are possible than there are for the phasing maneuver, while the cost of those maneuver is significantly less than the cost for overflying at the same time using an inclination plane change maneuver. It is also of note that the shape of the aeroassisted maneuver is similar to that of the inclination plane change maneuver but scaled down. The aeroassisted maneuver data is plotted alone for better visibility in Fig. 4.11. Re-circularization at the original altitude increases the velocity change cost by anywhere between .1 and .45 km/s and is a function of the velocity change cost required for re-circularization at the decayed altitude.

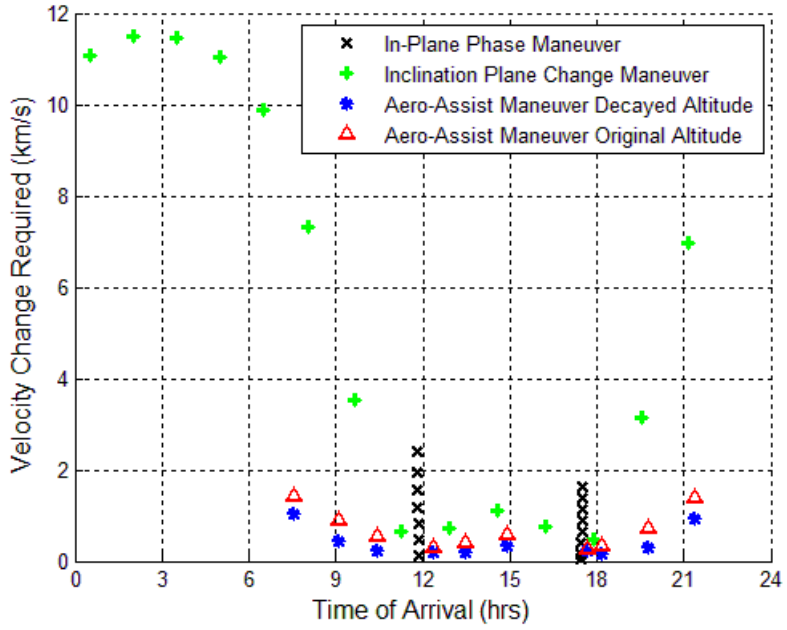

Figure 4.10. Comparison of ΔV and TOA Necessary for Overlying Tehran, Iran using Various Maneuver Types for an Initial Orbit of: Alt = 500 km, Inclination = 45°, Starting Longitude = 178° W, Starting Latitude = 0°

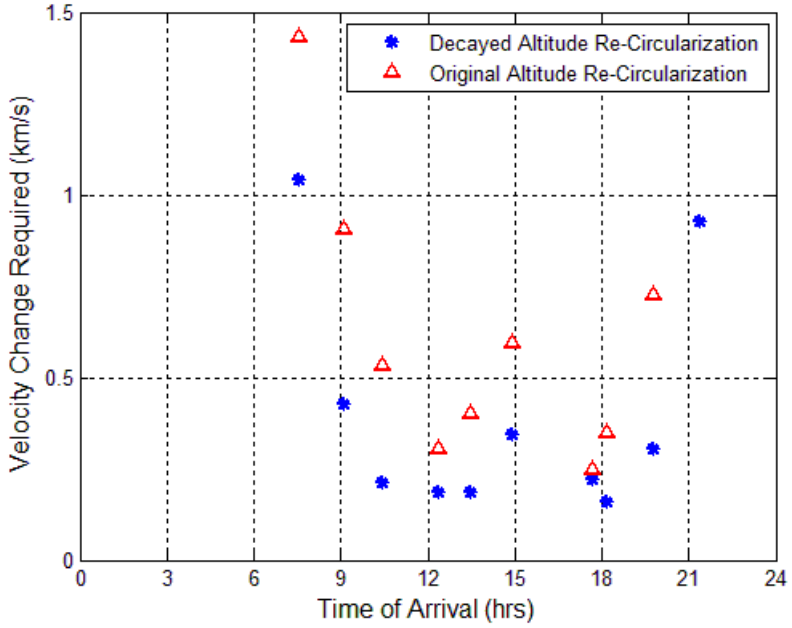

Figure 4.11. ΔV and TOA Necessary for Overlying Tehran, Iran using an Aeroassisted Maneuver from an Initial Orbit of: Alt = 500 km, Inclination = 45°, Starting Longitude = 178° W, Starting Latitude = 0°

The velocity change and timing of all possible solutions using the aeroassisted maneuver were calculated using the dynamics model simulation for multiple different target locations and initial orbits with the intent of identifying trends in order to develop a velocity change model. The data shows that the velocity cost trends for the aeroassisted maneuver mimic the inclination plane change maneuver cost with the exception that the higher ΔV solutions are not possible. Figure 4.12 shows a three-dimensional surface plot similar to Fig. 4.7 but the velocity change is for an aeroassisted maneuver. Again, the actual velocity change is not a continuous function of arrival time, only the discrete (black) points are actually possible, but the continuous function allows for trend visualization.

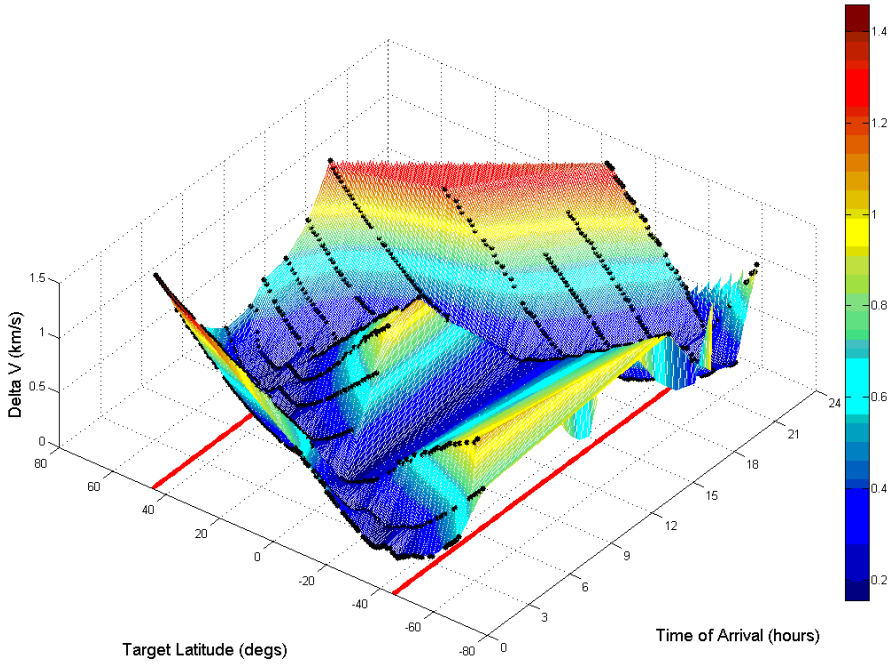

Figure 4.12a. Surface Plot of Original Altitude Aeroassisted Maneuver Velocity Change Cost as a Function of Both Arrival Time and Target Latitude

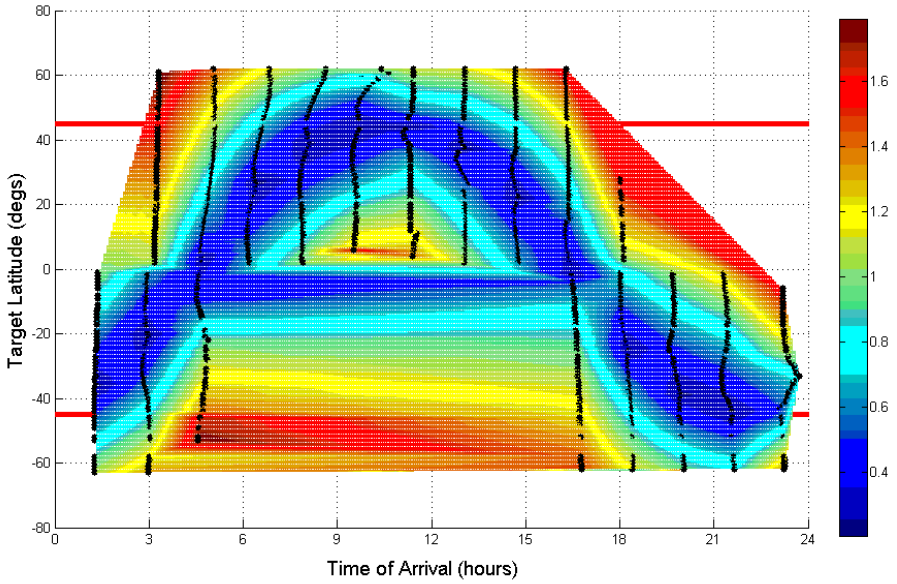

Figure 4.12b. Contour Plot of Original Altitude Aeroassisted Maneuver Velocity Change Cost as a Function of Both Arrival Time and Target Latitude

The trend is very similar to the trend seen in the plane change maneuver data, but without the higher velocity points. This similarity can be seen by re-plotting the data in Fig. 4.7 but without the higher velocity points (Fig. 4.13). As the purpose of these plots is not to determine the cost of a maneuver, but rather to identify and visualize trends within the data. The actual cost of the velocity change as shown in the color bars is not as important as the relative cost is between the maximum and minimum. Therefore, the color bars are not equal in the two plots of Fig. 4.13.

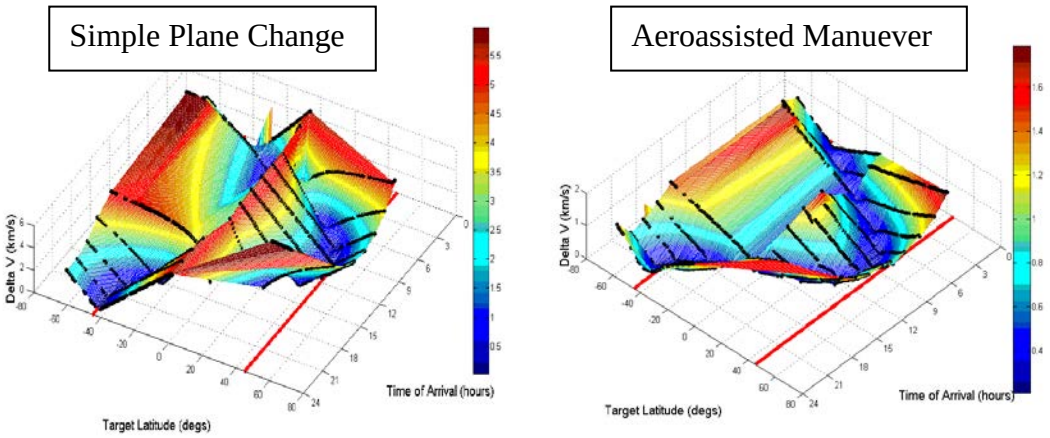

Figure 4.13. Visualization of Similar Trends within Inclination Plane Change and Aero-Assist Velocity Cost Data

With similar trends in velocity cost, a simplified estimation of the aeroassisted velocity change is:

$$\Delta V_{skip} = C_{scale}\Delta V_{plane} + \Delta V_{min} \tag{4.12}$$

Where C_{scale} is a scaling factor to be determined and ΔV_{min} is the minimum velocity change required to enter the atmosphere and re-circularize the orbit. The scaling factor is determined in a similar manner to the A, B, and C functions for the plane change maneuver. A minimizing function was used to find the scaling factor which minimizes the error function below. This scaling factor was calculated for multiple different target locations and initial orbits resulting in the data shown in Fig. 4.14.

$$error = \left\| \left(\Delta V_{skip}\right)_{data} - \left(C_{scale}\Delta V_{plane} + \Delta V_{min}\right) \right\| \tag{4.13}$$

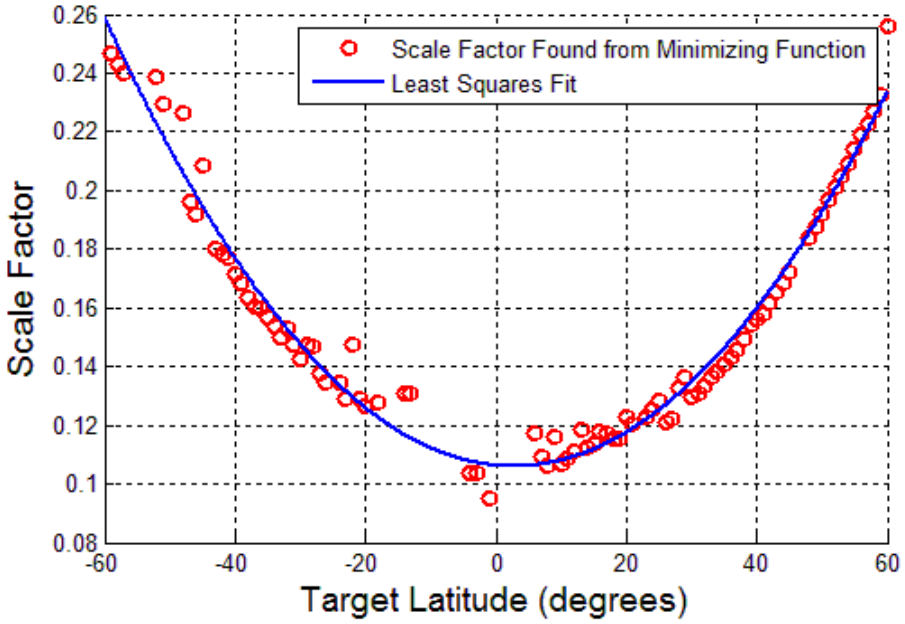

Figure 4.14. Scaling Factor for Minimizing Error Function (Eq. (4.13)) plotted against Target Latitude

A least squares fit of the data shows C_{scale} to be:

$$C_{scale} = 3.89 \times 10^{-5}\left(\phi_{target}\right)^2 - 2.41 \times 10^{-3}\left(\phi_{target}\right) + 0.10624 \tag{4.14}$$

The minimum velocity needed to enter the atmosphere and re-circularize again is a function of the semi-major axis. For the case of a 500 km altitude initial orbit the minimum velocity change is approximately 0.2 km/s. Using this value along with the scaling factor found from Eq. 4.14. The velocity change as a function of arrival time for a skip maneuver with re-circularization at the original altitude can be written as:

$$\Delta V_{skip} = 0.2 + \left(3.89 \times 10^{-5}\left(\phi_{target}\right)^2 - 2.41 \times 10^{-3}\left(\phi_{target}\right) + 0.10624\right)\Delta V_{plane} \tag{4.15}$$

Equation (4.15) gives reasonably accurate results for calculating the velocity change with larger inaccuracies occurring for targets close to the equator. The model

needs improvement but can suffice an approximation to be used in giving system users or researches a quick cost function of the velocity and arrival time needed for an aeroassisted maneuver. With the arrival times calculated using Eq. (4.11) the ΔV can be calculated using Eq. (4.15). All ΔV's greater than 1.8 km/s are then discarded as these velocities would overload the TAV (See Fig. 4.25) . Figure 4.15 shows the results of the model for one target location and initial orbit case. The RMS error for the ΔV is 0.093 km/s while the RMS error of the timing solution is 0.180 hours or 10 minutes. The error is smaller for target latitudes greater than the initial orbit inclination while the error increases as the target latitude approaches the equator. The majority of the error is found in the timing solution and this error carries over into the ΔV solution and therefore a more accurate timing solution is required to increase the overall model accuracy. The RMS error of the ΔV using accurate arrival times for the case shown in Fig. 4.15 is 0.072 km/s.

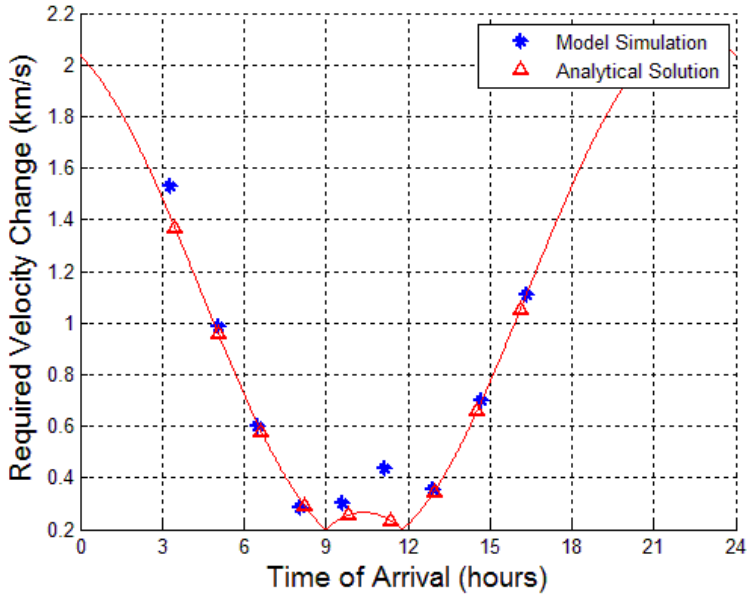

Figure 4.15. Comparison of Model (Eqs. 4.11, 4.13) to Dynamic Simulation Model for the Aeroassisted Maneuver with Re-Circularization at the Original Altitude.

Finding the velocity change required using the aeroassisted maneuver with re-circularization occurring at the decayed altitude is simply a matter of subtracting the correct amount from the original altitude re-circularization velocity change. This amount is a function of the velocity change required for re-circularization at the original altitude and can be shown to be:

$$\left(\Delta V_{skip}\right)_{decayed} = \left(\Delta V_{skip}\right)_{original} - \left[0.4\left(\Delta V_{skip}\right)_{original} + 0.1\right] \tag{4.16}$$

Equation (4.16) in conjunction with Eq. (4.15) gives reasonably accurate results for the velocity change using an aeroassisted maneuver and re-circularizing at the decayed altitude in order to overfly a ground target.

Target Overlfight Calculator

The ground target overlfight simulation along with the approximations determined above were used to create a target overlfight calculator. This calculator is a graphical user interface wherin the user can input the initial orbit, target location, start date and time, simulation run time, and vehicle properties. The user can then select to use either the numerical integration or the approximations determined above in order to determine the time of arrival of possible overflights and their associated ΔV. The numerical integration will give more accurate results but will take approximately 20 minutes for the calculations to be made. The approximations, while less accurate, will give results instantly. Figure 4.16 shows the graphical user interface. The outputs of the interface are: firstly, a plot similar to figure 4.10 showing the arrival time on the horizontal axis and the ΔV on the vertical axis with the various maneuver types being

distinguished by different symbols and colors; second, a table showing the time of arrival, ΔV, and maneuver type of all overflight solutions.

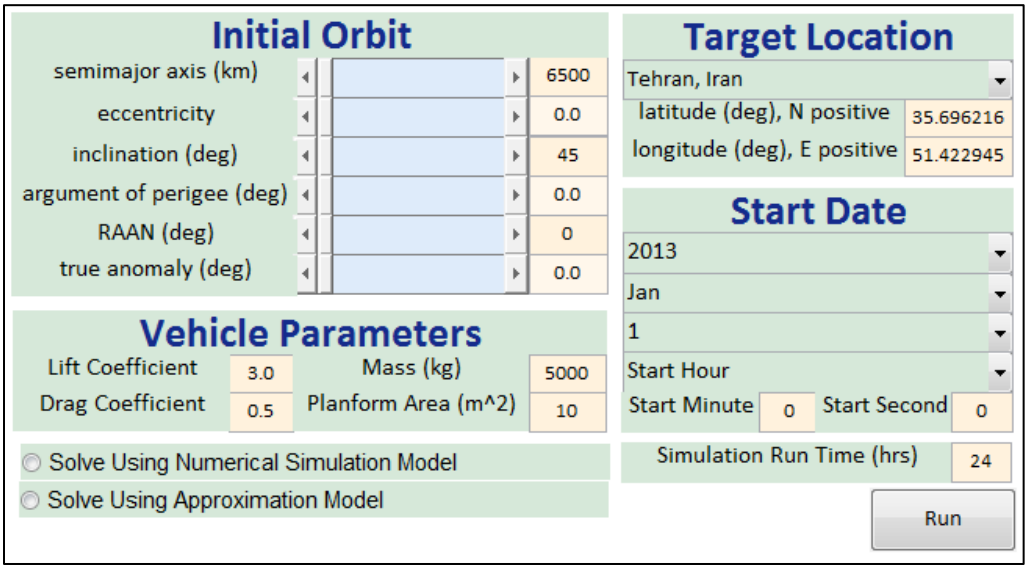

Figure 4.16. Graphical User Interface Used for the Target Overflight Calculator

Results of Aero-Assist Maneuver Reachability Simulations

The results of the study into the ability for an aeroassisted maneuver to modify orbital elements demonstrate that a TAV with an aeroassisted capability can dramatically offset cost in rendezvous maneuvers as well as desired orbit insertion maneuvers. The effect on orbit inclination and RAAN using an aeroassisted maneuver depend on the position of the TAV relative to the Earth when the maneuver is performed. To study this effect the change in inclination and RAAN were measured for multiple initial velocity deboost amounts as well as maneuver start times. The initial deboost velocity was varied from the minimum amount of velocity needed to enter the sensible atmosphere to the amount of deboost velocity that would over-g the TAV. The maneuver was performed at 12 different locations or starting times throughout one TAV orbit as shown in Fig. 4.17.

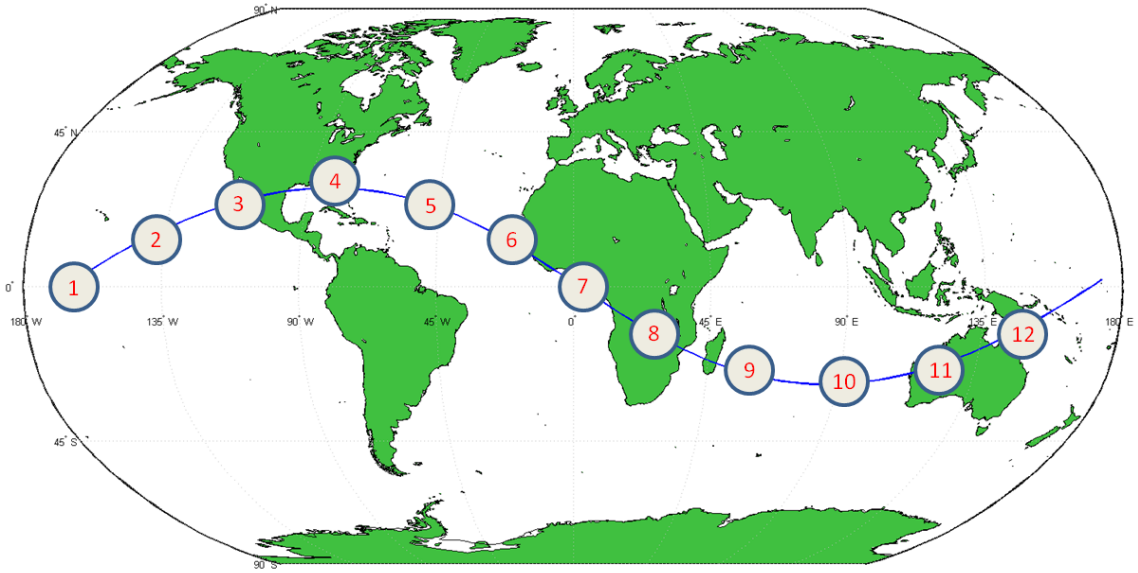

Figure 4.17. Maneuver Start Locations for Aeroassisted Reachability Study

The resulting change in inclination and RAAN were plotted with their associated velocity cost in three-dimensional space. Figure 4.18 shows the results for a circular orbit at an altitude of 500 km with an inclination of 28.52°. The inclination of 28.52° was chosen because it is the latitude of Cape Canaveral and therefore the lowest inclination achievable directly from a launch from that location. Figure 4.19 is a three-dimensional surface plot created by interpolating the data shown in Fig. 4.18.

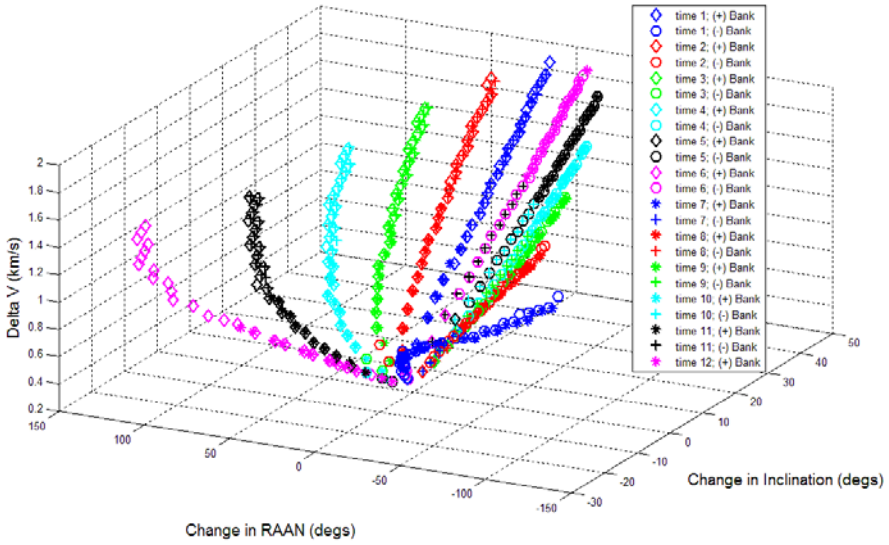

Figure 4.18. Simulation Data Point Cost of Inclination and RAAN change via an Aeroassisted Maneuver Performed from a 500 km Circular Orbit with 28.52° Inclination

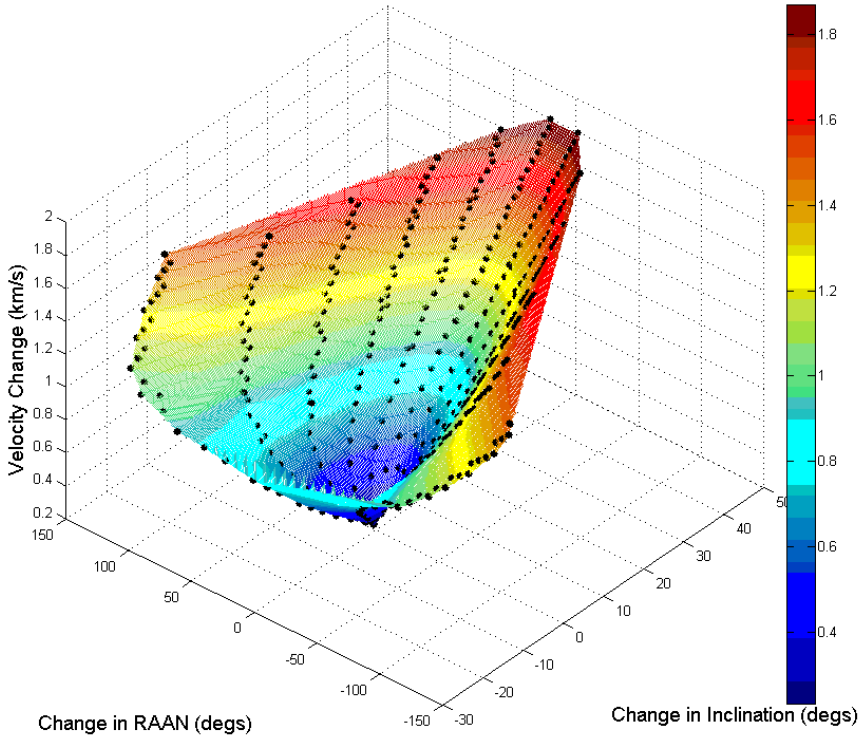

Figure 4.19. 3-Dimensional Surface of Velocity Cost for an Inclination and RAAN change via an Aeroassisted Maneuver Performed from a 500 km Circular Orbit with 28.52° Inclination

The surface is approximately a bowl with the exception that the inclination cannot change in the negative direction more than the original orbit inclination (28.52°) as this would produce a negative inclination. Inclination is always positive and is defined from $0 < i < 180°$. Any inclination less than zero is actually a positive inclination with the ascending node becoming the descending node. Many interesting facts can be gleamed from an analysis of Fig. 4.19. For a given orbit there exists a singular velocity and maneuver start time that correspond with a given inclination, RAAN change pair. In other words, if an inclination change of 10° and a RAAN change of -20° is desired, the maneuver must be done at a certain time, and has a particular velocity change cost. The range of orbital parameter change is quite extensive for a single skip maneuver. This range can be better seen in a top down view of Fig. 4.19 as seen in Fig. 4.20 and is shown in Table 4.1.

Table 4.1: Reachability from 500 km Circular 28.52° Inclined Orbit

	Maximum Change (°)	Decayed Altitude Re-Circularization ΔV (km/s)	Original Altitude Re-Circularization ΔV (km/s)	Max Deceleration (g's)	Perigee Altitude (km)
(+) Inclination	46.7	1.45	1.84	10.3	53.5
(-) Inclination	-25.2	0.67	0.93	4.1	60.5
(+) RAAN	139.6	1.26	1.5	10.6	53.3
(-) RAAN	-139	1.23	1.47	10.2	53.6

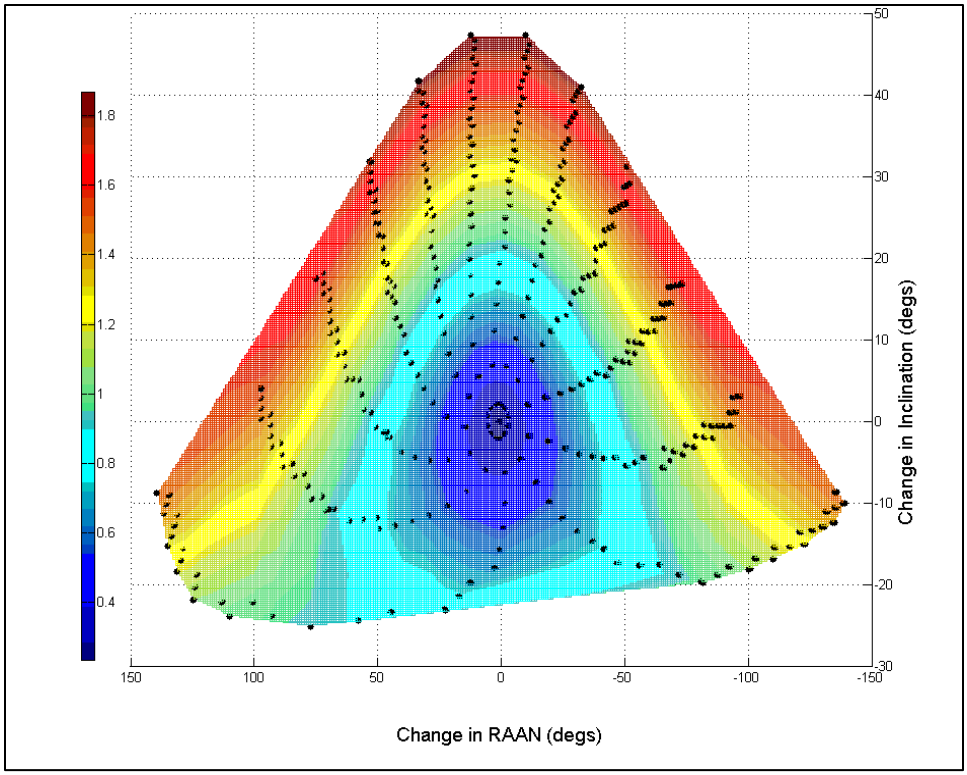

Figure 4.20. Top-Down Contour View of Velocity Cost for Inclination and RAAN Change using an Aeroassisted Maneuver

Aeroassisted maneuvers have the potential to dramatically offset costs of achieving a desired orbit. To demonstrate this cost savings the velocity change of inclination and RAAN change were calculated using traditional plane change maneuvers. A process for calculating the velocity change via a combined plane change maneuver is outlined in *Fundamentals of Astrodynamics and Applications*, by David Vallado. The process is simplified if the initial orbit is circular as the location of the maneuver will have no effect on overall cost because the starting velocity is a constant. First the angle, θ is found using Eq. (4.17), and the maneuver velocity cost is found using Eq. (4.18).

$$\cos(\theta) = -\cos(i_1)\cos(\pi - i_2) + \sin(i_1)\sin(\pi - i_2)\cos(\Delta\Omega) \tag{4.17}$$

$$\Delta V = 2V_1 sin\left(\frac{\theta}{2}\right) \tag{4.18}$$

This traditional method velocity cost is plotted along with the velocity cost for the aeroassisted maneuver in Fig. 4.21 using the same color legend in order to better compare the two. It is clear that an aeroassisted maneuver can significantly reduce the cost of orbit insertion. For example, if an equatorial orbit were desired, a satellite launched from Cape Canaveral would have to perform a plane change, which at 500 km would cost 3.75 km/s of velocity change using a simple plane change. The same change in inclination via an aeroassisted maneuver would cost roughly 0.94 km/s, representing a 75% decrease in velocity cost.

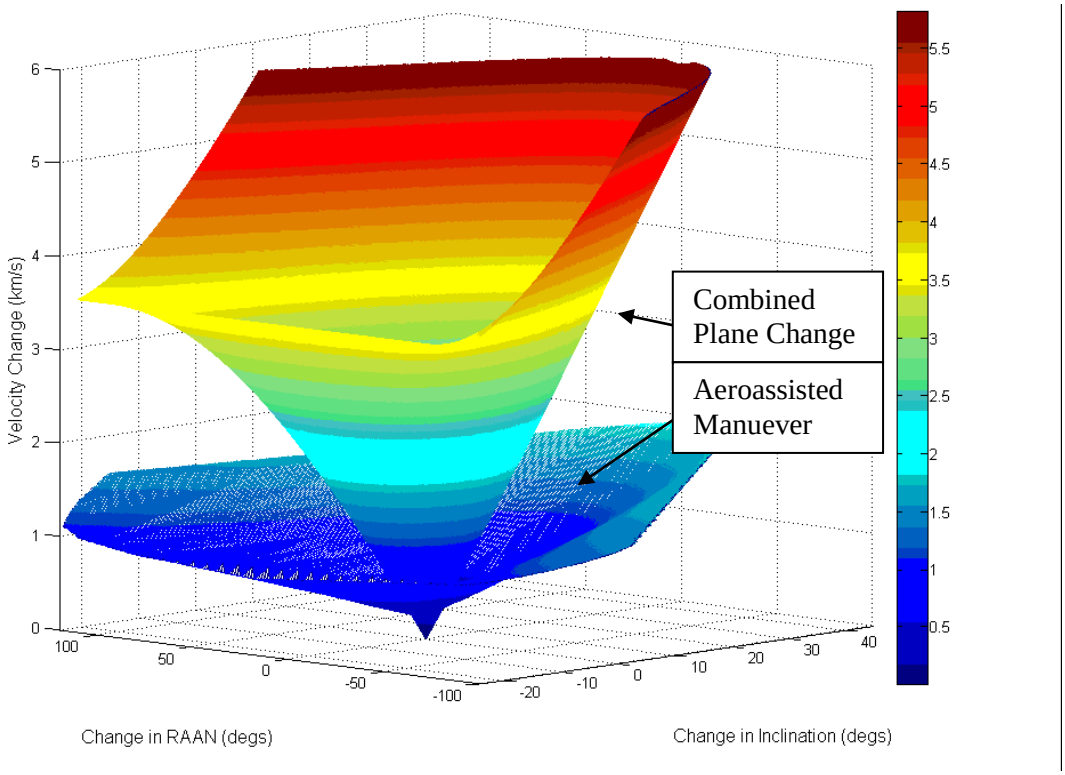

Figure 4.21. Comparison of Cost for Orbit Change using an Aeroassisted Maneuver with Traditional Methods

The velocity cost of changing orbital parameters using an aeroassisted maneuver varies with initial orbit inclination. The variation of reachability for a change in initial orbit inclination is shown in Fig. 4.22. As the inclination of the initial orbit increases the surface becomes more conical because the negative inclination change doesn't run into the positive inclination restraint. The cost for a change in orbit inclination is approximately the same for all initial inclination cases. The non-symettry of ΔV cost with north to south inclination change is due to the larger rotation speed of the earth near the equator.

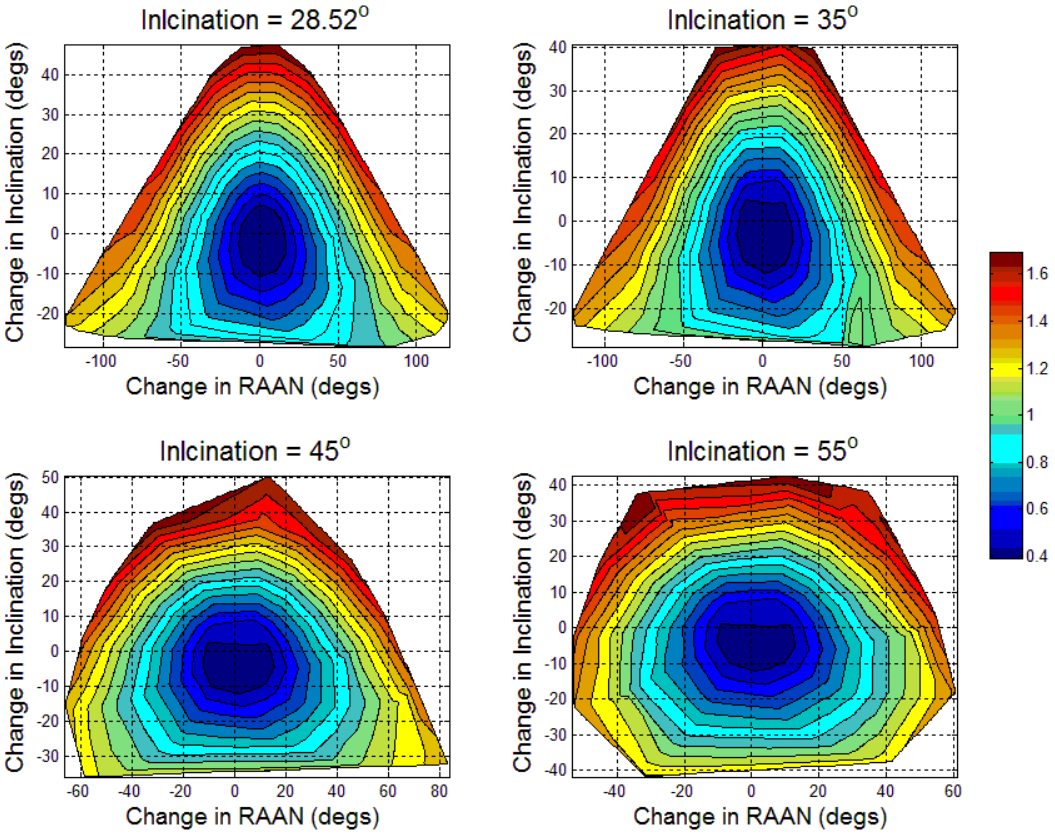

Fig. 4.22. ΔV Cost for a Change in Inclination and RAAN for Varied Initial Orbit Inclination; Initial Alt = 500 km, Bank Angle = 80°

Perigee Altitude

It is beneficial to see the change in inclination and RAAN as a function of perigee altitude rather than ΔV since it permits mission planners to know the perigee altitude of the decayed orbit in order to achieve a certain orbital change. Figure 4.23 shows the trend for an inclination of 55° with a 80° bank and starting altitude of 500km. The lowest perigee altitude without over g-loading the TAV is approximately 54 km.

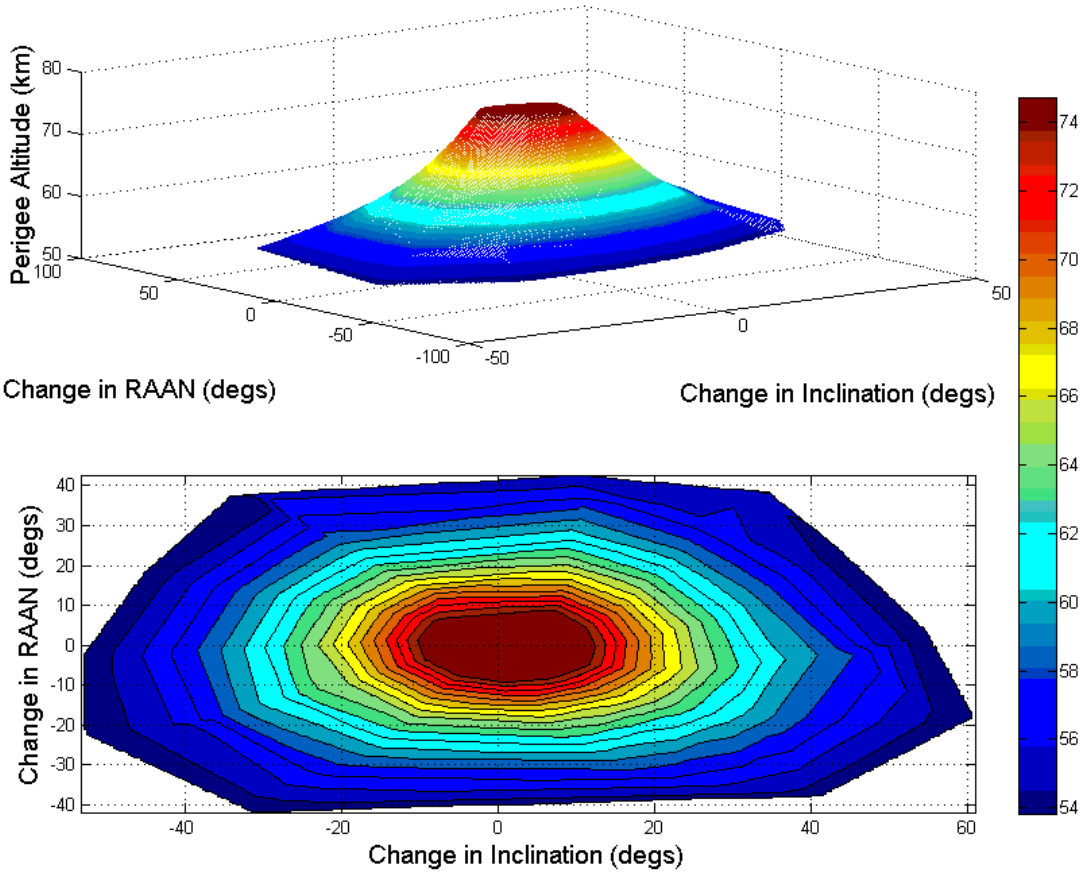

Figure 4.23. Pergiee Altitude for Desired Orbital Change Using an Aeroassisted Maneuver from an Initial Orbit of: Alt = 500 km, Inclination = 55°, Bank Angle = 80°

Deceleration

The amount of deceleration on the TAV is a linear function of the initial deboost velocity but changes with overall velocity based on where the maneuver was performed

and what direction of bank (Fig 4.24). For the worst case scenario, a final original re-circularization velocity under 1.8 km/s will always keep the g-loading under 10 g's. The g-loading limit drives the limiting condition for Eq. (4.11) and (4.13) where all solutions with a calculated $\Delta V > 1.8$ km/s are thrown out on the basis of over g-loading the TAV.

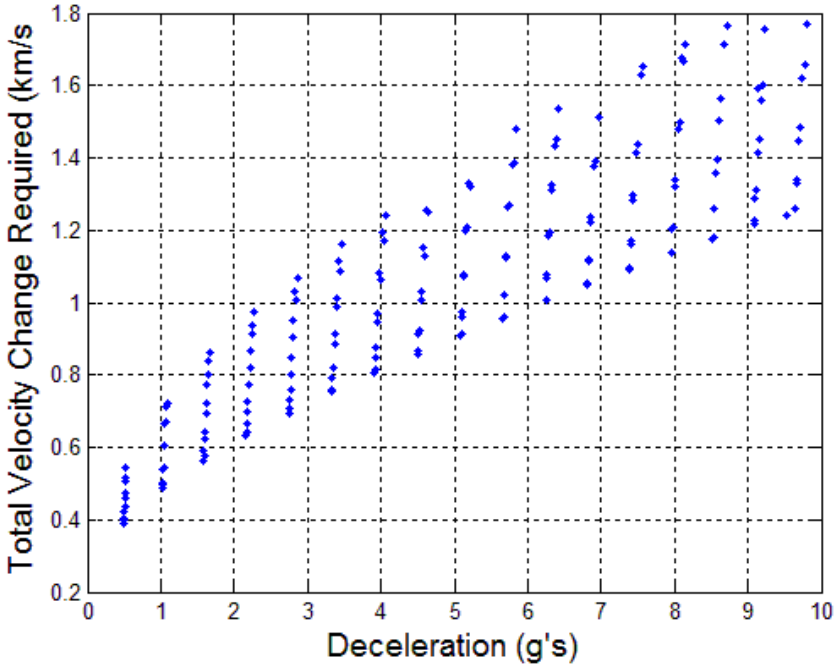

Figure 4.24. Velocity Change Required to Re-Circularize at the Original Orbit Altitude with the Associated Deceleration

Investigative Questions Answered/ Summary

An aeroassisted maneuver can be used to overfly a ground target when the arrival time of the phasing solution is not sufficient and the cost of the aeroassisted maneuver is acceptable. The cost and timing of the said maneuver can be calculated using Eqs. 4.11 and 4.13. Many orbits are reachable from a single orbit using an aeroassisted maneuver and can be used to replace traditional methods with significant cost savings.

V. Conclusions and Recommendations

Chapter Overview

The following chapter makes an effort to provide a clear and concise overview of the conclusions formulated throughout the study, and its significance. Recommendations for future research are thoughtfully given along with supporting rationale.

Conclusions of Research

The overflight simulation program successfully found both the time of arrival and velocity cost for in-plane phasing maneuvers, inclination plane change maneuvers, and aeroassisted maneuvers. Data gathered using the simulation program was used to develop a model for calculating the timing and velocity cost to successfully overfly any ground target from multiple starting circular orbits using all three type of maneuvers mentioned above. These models were used in the creation of a 'Target Overflight Calculator' which can be used by system users and researchers to gain a rapid estimation of all possible overflights, with their associated maneuver, arrival time, and ΔV cost. The reachability simulation program successfully predicted the ability of the aeroassisted maneuver to change the orbital parameters of any given starting orbit. The change in orbital parameters is a function of both the location where the aeroassisted maneuver is performed and the initial velocity descent into the atmosphere.

Significance of Research

Aeroassisted maneuvers offer a promising ΔV cost saving alternative to traditional methods for both overflying a ground target as well as changing a given orbit to a different desired orbit, including initial orbit insertion. Aeroassisted maneuvers

offer 5 to 8 more overflight solutions per day than do planar maneuvers while requiring 75%-90% less ΔV than exo-atmospheric plane change maneuvers. Results of the reachability study show a 75% decrease in ΔV over traditional exo-atmospheric maneuver with a single skip maneuver enabling a satellite to change its orbits inclination and RAAN up to ;The results of the study demonstrate the need for continued research in the development of trans-atmospheric vehicles.

Recommendations for Future Research

Although fulfilling the research objectives, the preceding research represents a fraction of the potential analysis in the study of aeroassisted maneuvers. Consequently, the following provides and outline of recommended future research.

- ***Effect of Bank Angle on Successful Overflight Timing and Cost:*** Determine aeroassisted maneuver reachability by varying the bank angle and identifying the resulting change it timing and ΔV costs.
- ***Vary Altitude of Initial Orbit:*** Repeat study with various starting initial orbit altitudes in order to determine the approximation equations as function of initial orbital altitude.
- ***Improve the Simple Plane Change and Aeroassisted Timing Solution:*** The main source of error in the developed model for timing and cost of successful overflights arises from error in the timing model. This model should be improved by determining an offset for the timing solution of each overflight based on the location of the original groundtrack in relation to the target location.

- ***Improve the velocity cost model for the aeroassisted maneuver:*** A Fourier series with multiple coefficients can be used to find a function that more accurately reflects that data found from the simulation program, particularly for targets close to the equator.
- ***Study multiple skip maneuvers.*** Multiple skips maneuvers are possible and have the potential of decreasing the cost of orbit modification. The structure of the simulation program can be easily modified to analyze multiple skip maneuvers, thus enabling the determination of the optimal number of skips for a desired change in orbit.

Appendix A: Nomenclature, Notation, and Units of Measure

The following list of symbols is alphabetical: Lowercase, then uppercase; Latin, then Greek. Due to the magnitude of distances associated with astrodynamics and re-entry analysis, all of the following symbols containing the base unit of measure of meters (m) are converted to kilometers (km). For the symbols related to planetary equatorial radius and rotation rate, the notation subscript $(\cdot)$ indicates an unspecified planet, while for χ and Δ the same notation indicates an unspecified base unit of measure.

Latin Symbol	*Definition*	*Base Unit of Measure*
a	Orbital semi-major axis	m
a_{decel}	Total deceleration	$\mathrm{m} \cdot \mathrm{s}^{-2}$
g	Gravitational acceleration	$\mathrm{m} \cdot \mathrm{s}^{-2}$
h	Altitude	m
i	Inclination angle	rad
m	Vehicle mass	kg
r	Geocentric radial distance from planet center of mass to vehicle	m
t	General time	s
C_D	Coefficient of drag	unitless
C_L	Coefficient of lift	unitless
D	Drag force	$\mathrm{kg} \cdot \mathrm{m} \cdot \mathrm{s}^{-1}$
L	Lift force	$\mathrm{kg} \cdot \mathrm{m} \cdot \mathrm{s}^{-1}$
N_{orbits}	Number of Orbits	variable
P	Keplerian orbital period	s
$\dot{Q}$	Stagnation Heat Flux	$\mathrm{BTU}(\mathrm{ft}^2\ \mathrm{s})^{-1}$,
RMS	Root mean square	unitless
S	Planform area	m^2
V	Velocity	$\mathrm{m} \cdot \mathrm{s}^{-1}$

Greek Symbol	*Definition*	*Base Unit of Measure*
β	Atmospheric scale height	m^{-1}
γ	Flight-path angle	rad
θ	Longitude	rad
μ	Gravitational parameter	$\mathrm{m}^3 \cdot \mathrm{s}^{-2}$
ν	True Anamoly	rad

ρ	Atmospheric density	$\mathrm{kg} \cdot \mathrm{m}^{-3}$
σ	Bank angle	rad
φ	Latitude	rad
ψ	Heading angle	rad
ω	Argument of Perigee	rad
$\omega_{\oplus}$	Planetary rotation rate	$\mathrm{rad} \cdot \mathrm{s}^{-2}$
Ω	Right Ascension of the Ascending Node	rad
Δ	Change in value, i.e. ΔV	$(\cdot)$

Symbol Scripting	*Definition*
$(\ )_{gd}$	Geodetic value
$(\ )_i$	Initial conditions
$(\ )_r$	Component in radial direction
$(\ )_s$	Stagnation value
$(\ )_v$	Component in velocity direction
$(\ )_w$	Conditions at vehicle surface
$(\ )_L$	Component in lift direction
$(\ )_\phi$	Component in transverse direction
${}^{I}(\ )$	Measured with respect to an inertial frame
${}^{R}(\ )$	Measured with respect to a rotating frame
$(\ )_{\oplus}$	Conditions for the Earth

Appendix B: Test Matrices

Table B.1 Overflight Simulation Test Matrix

				Initial Orbit						
Test #	**Run #**	**tgtlat**	**tgtlon**	**a**	**e**	**i**	**ω**	**Ω**	**v**	**Bank**
1	1	0.5	-90	6878	0	45	0	-24	0	80
	2	0.5	-90	6878	0	45	0	-21.6	0	80
	3	0.5	-90	6878	0	45	0	-19.2	0	80
	4	0.5	-90	6878	0	45	0	-16.8	0	80
	5	0.5	-90	6878	0	45	0	-14.4	0	80
	6	0.5	-90	6878	0	45	0	-12	0	80
	7	0.5	-90	6878	0	45	0	-9.6	0	80
	8	0.5	-90	6878	0	45	0	-7.2	0	80
	9	0.5	-90	6878	0	45	0	-4.8	0	80
	10	0.5	-90	6878	0	45	0	-2.4	0	80
	11	0.5	-90	6878	0	45	0	0	0	80
	12	0.5	-90	6878	0	45	0	2.4	0	80
	13	0.5	-90	6878	0	45	0	4.8	0	80
	14	0.5	-90	6878	0	45	0	7.2	0	80
	15	0.5	-90	6878	0	45	0	9.6	0	80
	16	0.5	-90	6878	0	45	0	12	0	80
	17	0.5	-90	6878	0	45	0	14.4	0	80
	18	0.5	-90	6878	0	45	0	16.8	0	80
	19	0.5	-90	6878	0	45	0	19.2	0	80
	20	0.5	-90	6878	0	45	0	21.6	0	80
	21	0.5	-90	6878	0	45	0	24	0	80

				Initial Orbit						
Test #	**Run #**	**tgtlat**	**tgtlon**	**a**	**e**	**i**	**ω**	**Ω**	**v**	**Bank**
2	1	15	-90	6878	0	45	0	-24	0	80
	2	15	-90	6878	0	45	0	-21.6	0	80
	3	15	-90	6878	0	45	0	-19.2	0	80
	4	15	-90	6878	0	45	0	-16.8	0	80
	5	15	-90	6878	0	45	0	-14.4	0	80
	6	15	-90	6878	0	45	0	-12	0	80
	7	15	-90	6878	0	45	0	-9.6	0	80
	8	15	-90	6878	0	45	0	-7.2	0	80
	9	15	-90	6878	0	45	0	-4.8	0	80
	10	15	-90	6878	0	45	0	-2.4	0	80
	11	15	-90	6878	0	45	0	0	0	80
	12	15	-90	6878	0	45	0	2.4	0	80
	13	15	-90	6878	0	45	0	4.8	0	80
	14	15	-90	6878	0	45	0	7.2	0	80
	15	15	-90	6878	0	45	0	9.6	0	80
	16	15	-90	6878	0	45	0	12	0	80
	17	15	-90	6878	0	45	0	14.4	0	80
	18	15	-90	6878	0	45	0	16.8	0	80
	19	15	-90	6878	0	45	0	19.2	0	80
	20	15	-90	6878	0	45	0	21.6	0	80
	21	15	-90	6878	0	45	0	24	0	80
	1	15	-90	6878	0	45	0	-24	0	80

				Initial Orbit						
Test #	Run #	tgtlat	tgtlon	a	e	i	ω	Ω	v	Bank
3	1	30	-90	6878	0	45	0	-24	0	80
	2	30	-90	6878	0	45	0	-21.6	0	80
	3	30	-90	6878	0	45	0	-19.2	0	80
	4	30	-90	6878	0	45	0	-16.8	0	80
	5	30	-90	6878	0	45	0	-14.4	0	80
	6	30	-90	6878	0	45	0	-12	0	80
	7	30	-90	6878	0	45	0	-9.6	0	80
	8	30	-90	6878	0	45	0	-7.2	0	80
	9	30	-90	6878	0	45	0	-4.8	0	80
	10	30	-90	6878	0	45	0	-2.4	0	80
	11	30	-90	6878	0	45	0	0	0	80
	12	30	-90	6878	0	45	0	2.4	0	80
	13	30	-90	6878	0	45	0	4.8	0	80
	14	30	-90	6878	0	45	0	7.2	0	80
	15	30	-90	6878	0	45	0	9.6	0	80
	16	30	-90	6878	0	45	0	12	0	80
	17	30	-90	6878	0	45	0	14.4	0	80
	18	30	-90	6878	0	45	0	16.8	0	80
	19	30	-90	6878	0	45	0	19.2	0	80
	20	30	-90	6878	0	45	0	21.6	0	80
	21	30	-90	6878	0	45	0	24	0	80
	1	30	-90	6878	0	45	0	-24	0	80

				Initial Orbit						
Test #	**Run #**	**tgtlat**	**tgtlon**	**a**	**e**	**i**	**ω**	**Ω**	**v**	**Bank**
4	1	45	-90	6878	0	45	0	-24	0	80
	2	45	-90	6878	0	45	0	-21.6	0	80
	3	45	-90	6878	0	45	0	-19.2	0	80
	4	45	-90	6878	0	45	0	-16.8	0	80
	5	45	-90	6878	0	45	0	-14.4	0	80
	6	45	-90	6878	0	45	0	-12	0	80
	7	45	-90	6878	0	45	0	-9.6	0	80
	8	45	-90	6878	0	45	0	-7.2	0	80
	9	45	-90	6878	0	45	0	-4.8	0	80
	10	45	-90	6878	0	45	0	-2.4	0	80
	11	45	-90	6878	0	45	0	0	0	80
	12	45	-90	6878	0	45	0	2.4	0	80
	13	45	-90	6878	0	45	0	4.8	0	80
	14	45	-90	6878	0	45	0	7.2	0	80
	15	45	-90	6878	0	45	0	9.6	0	80
	16	45	-90	6878	0	45	0	12	0	80
	17	45	-90	6878	0	45	0	14.4	0	80
	18	45	-90	6878	0	45	0	16.8	0	80
	19	45	-90	6878	0	45	0	19.2	0	80
	20	45	-90	6878	0	45	0	21.6	0	80
	21	45	-90	6878	0	45	0	24	0	80
	1	45	-90	6878	0	45	0	-24	0	80

				Initial Orbit						
Test #	Run #	tgtlat	tgtlon	a	e	i	ω	Ω	v	Bank
5	1	-60	40	6878	0	45	0	0	0	80
	2	-59	40	6878	0	45	0	0	0	80
	3	-58	40	6878	0	45	0	0	0	80
	4	-57	40	6878	0	45	0	0	0	80
	5	-56	40	6878	0	45	0	0	0	80
	6	-55	40	6878	0	45	0	0	0	80
	7	-54	40	6878	0	45	0	0	0	80
	8	-53	40	6878	0	45	0	0	0	80
	9	-52	40	6878	0	45	0	0	0	80
	10	-51	40	6878	0	45	0	0	0	80
	11	-50	40	6878	0	45	0	0	0	80
	12	-49	40	6878	0	45	0	0	0	80
	13	-48	40	6878	0	45	0	0	0	80
	14	-47	40	6878	0	45	0	0	0	80
	15	-46	40	6878	0	45	0	0	0	80
	16	-45	40	6878	0	45	0	0	0	80
	17	-44	40	6878	0	45	0	0	0	80
	18	-43	40	6878	0	45	0	0	0	80
	19	-42	40	6878	0	45	0	0	0	80
	20	-41	40	6878	0	45	0	0	0	80
	21	-40	40	6878	0	45	0	0	0	80
	22	-39	40	6878	0	45	0	0	0	80

	23	-38	40	6878	0	45	0	0	0	80
	24	-37	40	6878	0	45	0	0	0	80
	25	-36	40	6878	0	45	0	0	0	80
	26	-35	40	6878	0	45	0	0	0	80
	27	-34	40	6878	0	45	0	0	0	80
	28	-33	40	6878	0	45	0	0	0	80
	29	-32	40	6878	0	45	0	0	0	80
	30	-31	40	6878	0	45	0	0	0	80
	31	-30	40	6878	0	45	0	0	0	80
	32	-29	40	6878	0	45	0	0	0	80
	33	-28	40	6878	0	45	0	0	0	80
	34	-27	40	6878	0	45	0	0	0	80
	35	-26	40	6878	0	45	0	0	0	80
	36	-25	40	6878	0	45	0	0	0	80
	37	-24	40	6878	0	45	0	0	0	80
	38	-23	40	6878	0	45	0	0	0	80
	39	-22	40	6878	0	45	0	0	0	80
	40	-21	40	6878	0	45	0	0	0	80
	41	-20	40	6878	0	45	0	0	0	80
	42	-19	40	6878	0	45	0	0	0	80
	43	-18	40	6878	0	45	0	0	0	80
	44	-17	40	6878	0	45	0	0	0	80
	45	-16	40	6878	0	45	0	0	0	80
	46	-15	40	6878	0	45	0	0	0	80
	47	-14	40	6878	0	45	0	0	0	80

	48	-13	40	6878	0	45	0	0	0	80
	49	-12	40	6878	0	45	0	0	0	80
	50	-11	40	6878	0	45	0	0	0	80
	51	-10	40	6878	0	45	0	0	0	80
	52	-9	40	6878	0	45	0	0	0	80
	53	-8	40	6878	0	45	0	0	0	80
	54	-7	40	6878	0	45	0	0	0	80
	55	-6	40	6878	0	45	0	0	0	80
	56	-5	40	6878	0	45	0	0	0	80
	57	-4	40	6878	0	45	0	0	0	80
	58	-3	40	6878	0	45	0	0	0	80
	59	-2	40	6878	0	45	0	0	0	80
	60	-1	40	6878	0	45	0	0	0	80
	61	0	40	6878	0	45	0	0	0	80
	62	1	40	6878	0	45	0	0	0	80
	63	2	40	6878	0	45	0	0	0	80
	64	3	40	6878	0	45	0	0	0	80
	65	4	40	6878	0	45	0	0	0	80
	66	5	40	6878	0	45	0	0	0	80
	67	6	40	6878	0	45	0	0	0	80
	68	7	40	6878	0	45	0	0	0	80
	69	8	40	6878	0	45	0	0	0	80
	70	9	40	6878	0	45	0	0	0	80
	71	10	40	6878	0	45	0	0	0	80
	72	11	40	6878	0	45	0	0	0	80

	73	12	40	6878	0	45	0	0	0	80
	74	13	40	6878	0	45	0	0	0	80
	75	14	40	6878	0	45	0	0	0	80
	76	15	40	6878	0	45	0	0	0	80
	77	16	40	6878	0	45	0	0	0	80
	78	17	40	6878	0	45	0	0	0	80
	79	18	40	6878	0	45	0	0	0	80
	80	19	40	6878	0	45	0	0	0	80
	81	20	40	6878	0	45	0	0	0	80
	82	21	40	6878	0	45	0	0	0	80
	83	22	40	6878	0	45	0	0	0	80
	84	23	40	6878	0	45	0	0	0	80
	85	24	40	6878	0	45	0	0	0	80
	86	25	40	6878	0	45	0	0	0	80
	87	26	40	6878	0	45	0	0	0	80
	88	27	40	6878	0	45	0	0	0	80
	89	28	40	6878	0	45	0	0	0	80
	90	29	40	6878	0	45	0	0	0	80
	91	30	40	6878	0	45	0	0	0	80
	92	31	40	6878	0	45	0	0	0	80
	93	32	40	6878	0	45	0	0	0	80
	94	33	40	6878	0	45	0	0	0	80
	95	34	40	6878	0	45	0	0	0	80
	96	35	40	6878	0	45	0	0	0	80
	97	36	40	6878	0	45	0	0	0	80

	98	37	40	6878	0	45	0	0	0	80
	99	38	40	6878	0	45	0	0	0	80
	100	39	40	6878	0	45	0	0	0	80
	101	40	40	6878	0	45	0	0	0	80
	102	41	40	6878	0	45	0	0	0	80
	103	42	40	6878	0	45	0	0	0	80
	104	43	40	6878	0	45	0	0	0	80
	105	44	40	6878	0	45	0	0	0	80
	106	45	40	6878	0	45	0	0	0	80
	107	46	40	6878	0	45	0	0	0	80
	108	47	40	6878	0	45	0	0	0	80
	109	48	40	6878	0	45	0	0	0	80
	110	49	40	6878	0	45	0	0	0	80
	111	50	40	6878	0	45	0	0	0	80
	112	51	40	6878	0	45	0	0	0	80
	113	52	40	6878	0	45	0	0	0	80
	114	53	40	6878	0	45	0	0	0	80
	115	54	40	6878	0	45	0	0	0	80
	116	55	40	6878	0	45	0	0	0	80
	117	56	40	6878	0	45	0	0	0	80
	118	57	40	6878	0	45	0	0	0	80
	119	58	40	6878	0	45	0	0	0	80
	120	59	40	6878	0	45	0	0	0	80
	121	60	40	6878	0	45	0	0	0	80

				Initial Orbit						
Test #	**Run #**	**tgtlat**	**tgtlon**	**a**	**e**	**i**	**ω**	**Ω**	**v**	**Bank**
6	1	-60	-90	6878	0	45	0	0	0	80
	2	-59	-90	6878	0	45	0	0	0	80
	3	-58	-90	6878	0	45	0	0	0	80
	4	-57	-90	6878	0	45	0	0	0	80
	5	-56	-90	6878	0	45	0	0	0	80
	6	-55	-90	6878	0	45	0	0	0	80
	7	-54	-90	6878	0	45	0	0	0	80
	8	-53	-90	6878	0	45	0	0	0	80
	9	-52	-90	6878	0	45	0	0	0	80
	10	-51	-90	6878	0	45	0	0	0	80
	11	-50	-90	6878	0	45	0	0	0	80
	12	-49	-90	6878	0	45	0	0	0	80
	13	-48	-90	6878	0	45	0	0	0	80
	14	-47	-90	6878	0	45	0	0	0	80
	15	-46	-90	6878	0	45	0	0	0	80
	16	-45	-90	6878	0	45	0	0	0	80
	17	-44	-90	6878	0	45	0	0	0	80
	18	-43	-90	6878	0	45	0	0	0	80
	19	-42	-90	6878	0	45	0	0	0	80
	20	-41	-90	6878	0	45	0	0	0	80
	21	-40	-90	6878	0	45	0	0	0	80
	22	-39	-90	6878	0	45	0	0	0	80
	23	-38	-90	6878	0	45	0	0	0	80

	24	-37	-90	6878	0	45	0	0	0	80
	25	-36	-90	6878	0	45	0	0	0	80
	26	-35	-90	6878	0	45	0	0	0	80
	27	-34	-90	6878	0	45	0	0	0	80
	28	-33	-90	6878	0	45	0	0	0	80
	29	-32	-90	6878	0	45	0	0	0	80
	30	-31	-90	6878	0	45	0	0	0	80
	31	-30	-90	6878	0	45	0	0	0	80
	32	-29	-90	6878	0	45	0	0	0	80
	33	-28	-90	6878	0	45	0	0	0	80
	34	-27	-90	6878	0	45	0	0	0	80
	35	-26	-90	6878	0	45	0	0	0	80
	36	-25	-90	6878	0	45	0	0	0	80
	37	-24	-90	6878	0	45	0	0	0	80
	38	-23	-90	6878	0	45	0	0	0	80
	39	-22	-90	6878	0	45	0	0	0	80
	40	-21	-90	6878	0	45	0	0	0	80
	41	-20	-90	6878	0	45	0	0	0	80
	42	-19	-90	6878	0	45	0	0	0	80
	43	-18	-90	6878	0	45	0	0	0	80
	44	-17	-90	6878	0	45	0	0	0	80
	45	-16	-90	6878	0	45	0	0	0	80
	46	-15	-90	6878	0	45	0	0	0	80
	47	-14	-90	6878	0	45	0	0	0	80
	48	-13	-90	6878	0	45	0	0	0	80

	49	-12	-90	6878	0	45	0	0	0	80
	50	-11	-90	6878	0	45	0	0	0	80
	51	-10	-90	6878	0	45	0	0	0	80
	52	-9	-90	6878	0	45	0	0	0	80
	53	-8	-90	6878	0	45	0	0	0	80
	54	-7	-90	6878	0	45	0	0	0	80
	55	-6	-90	6878	0	45	0	0	0	80
	56	-5	-90	6878	0	45	0	0	0	80
	57	-4	-90	6878	0	45	0	0	0	80
	58	-3	-90	6878	0	45	0	0	0	80
	59	-2	-90	6878	0	45	0	0	0	80
	60	-1	-90	6878	0	45	0	0	0	80
	61	0	-90	6878	0	45	0	0	0	80
	62	1	-90	6878	0	45	0	0	0	80
	63	2	-90	6878	0	45	0	0	0	80
	64	3	-90	6878	0	45	0	0	0	80
	65	4	-90	6878	0	45	0	0	0	80
	66	5	-90	6878	0	45	0	0	0	80
	67	6	-90	6878	0	45	0	0	0	80
	68	7	-90	6878	0	45	0	0	0	80
	69	8	-90	6878	0	45	0	0	0	80
	70	9	-90	6878	0	45	0	0	0	80
	71	10	-90	6878	0	45	0	0	0	80
	72	11	-90	6878	0	45	0	0	0	80
	73	12	-90	6878	0	45	0	0	0	80

	74	13	-90	6878	0	45	0	0	0	80
	75	14	-90	6878	0	45	0	0	0	80
	76	15	-90	6878	0	45	0	0	0	80
	77	16	-90	6878	0	45	0	0	0	80
	78	17	-90	6878	0	45	0	0	0	80
	79	18	-90	6878	0	45	0	0	0	80
	80	19	-90	6878	0	45	0	0	0	80
	81	20	-90	6878	0	45	0	0	0	80
	82	21	-90	6878	0	45	0	0	0	80
	83	22	-90	6878	0	45	0	0	0	80
	84	23	-90	6878	0	45	0	0	0	80
	85	24	-90	6878	0	45	0	0	0	80
	86	25	-90	6878	0	45	0	0	0	80
	87	26	-90	6878	0	45	0	0	0	80
	88	27	-90	6878	0	45	0	0	0	80
	89	28	-90	6878	0	45	0	0	0	80
	90	29	-90	6878	0	45	0	0	0	80
	91	30	-90	6878	0	45	0	0	0	80
	92	31	-90	6878	0	45	0	0	0	80
	93	32	-90	6878	0	45	0	0	0	80
	94	33	-90	6878	0	45	0	0	0	80
	95	34	-90	6878	0	45	0	0	0	80
	96	35	-90	6878	0	45	0	0	0	80
	97	36	-90	6878	0	45	0	0	0	80
	98	37	-90	6878	0	45	0	0	0	80

	99	38	-90	6878	0	45	0	0	0	80
	100	39	-90	6878	0	45	0	0	0	80
	101	40	-90	6878	0	45	0	0	0	80
	102	41	-90	6878	0	45	0	0	0	80
	103	42	-90	6878	0	45	0	0	0	80
	104	43	-90	6878	0	45	0	0	0	80
	105	44	-90	6878	0	45	0	0	0	80
	106	45	-90	6878	0	45	0	0	0	80
	107	46	-90	6878	0	45	0	0	0	80
	108	47	-90	6878	0	45	0	0	0	80
	109	48	-90	6878	0	45	0	0	0	80
	110	49	-90	6878	0	45	0	0	0	80
	111	50	-90	6878	0	45	0	0	0	80
	112	51	-90	6878	0	45	0	0	0	80
	113	52	-90	6878	0	45	0	0	0	80
	114	53	-90	6878	0	45	0	0	0	80
	115	54	-90	6878	0	45	0	0	0	80
	116	55	-90	6878	0	45	0	0	0	80
	117	56	-90	6878	0	45	0	0	0	80
	118	57	-90	6878	0	45	0	0	0	80
	119	58	-90	6878	0	45	0	0	0	80
	120	59	-90	6878	0	45	0	0	0	80
	121	60	-90	6878	0	45	0	0	0	80

				Initial Orbit						
Test #	Run #	tgtlat	tgtlon	a	e	i	ω	Ω	v	Bank
7	1	45	-90	6878	0	0	0	0	0	80
	2	45	-90	6878	0	1	0	0	0	80
	3	45	-90	6878	0	2	0	0	0	80
	4	45	-90	6878	0	3	0	0	0	80
	5	45	-90	6878	0	4	0	0	0	80
	6	45	-90	6878	0	5	0	0	0	80
	7	45	-90	6878	0	6	0	0	0	80
	8	45	-90	6878	0	7	0	0	0	80
	9	45	-90	6878	0	8	0	0	0	80
	10	45	-90	6878	0	9	0	0	0	80
	11	45	-90	6878	0	10	0	0	0	80
	12	45	-90	6878	0	11	0	0	0	80
	13	45	-90	6878	0	12	0	0	0	80
	14	45	-90	6878	0	13	0	0	0	80
	15	45	-90	6878	0	14	0	0	0	80
	16	45	-90	6878	0	15	0	0	0	80
	17	45	-90	6878	0	16	0	0	0	80
	18	45	-90	6878	0	17	0	0	0	80
	19	45	-90	6878	0	18	0	0	0	80
	20	45	-90	6878	0	19	0	0	0	80
	21	45	-90	6878	0	20	0	0	0	80
	22	45	-90	6878	0	21	0	0	0	80

	23	45	-90	6878	0	22	0	0	0	80
	24	45	-90	6878	0	23	0	0	0	80
	25	45	-90	6878	0	24	0	0	0	80
	26	45	-90	6878	0	25	0	0	0	80
	27	45	-90	6878	0	26	0	0	0	80
	28	45	-90	6878	0	27	0	0	0	80
	29	45	-90	6878	0	28	0	0	0	80
	30	45	-90	6878	0	29	0	0	0	80
	31	45	-90	6878	0	30	0	0	0	80
	32	45	-90	6878	0	31	0	0	0	80
	33	45	-90	6878	0	32	0	0	0	80
	34	45	-90	6878	0	33	0	0	0	80
	35	45	-90	6878	0	34	0	0	0	80
	36	45	-90	6878	0	35	0	0	0	80
	37	45	-90	6878	0	36	0	0	0	80
	38	45	-90	6878	0	37	0	0	0	80
	39	45	-90	6878	0	38	0	0	0	80
	40	45	-90	6878	0	39	0	0	0	80
	41	45	-90	6878	0	40	0	0	0	80
	42	45	-90	6878	0	41	0	0	0	80
	43	45	-90	6878	0	42	0	0	0	80
	44	45	-90	6878	0	43	0	0	0	80
	45	45	-90	6878	0	44	0	0	0	80
	46	45	-90	6878	0	45	0	0	0	80
	47	45	-90	6878	0	46	0	0	0	80

	48	45	-90	6878	0	47	0	0	0	80
	49	45	-90	6878	0	48	0	0	0	80
	50	45	-90	6878	0	49	0	0	0	80
	51	45	-90	6878	0	50	0	0	0	80
	52	45	-90	6878	0	51	0	0	0	80
	53	45	-90	6878	0	52	0	0	0	80
	54	45	-90	6878	0	53	0	0	0	80
	55	45	-90	6878	0	54	0	0	0	80
	56	45	-90	6878	0	55	0	0	0	80
	57	45	-90	6878	0	56	0	0	0	80
	58	45	-90	6878	0	57	0	0	0	80
	59	45	-90	6878	0	58	0	0	0	80
	60	45	-90	6878	0	59	0	0	0	80
	61	45	-90	6878	0	60	0	0	0	80
	62	45	-90	6878	0	61	0	0	0	80
	63	45	-90	6878	0	62	0	0	0	80
	64	45	-90	6878	0	63	0	0	0	80
	65	45	-90	6878	0	64	0	0	0	80
	66	45	-90	6878	0	65	0	0	0	80
	67	45	-90	6878	0	66	0	0	0	80
	68	45	-90	6878	0	67	0	0	0	80
	69	45	-90	6878	0	68	0	0	0	80
	70	45	-90	6878	0	69	0	0	0	80
	71	45	-90	6878	0	70	0	0	0	80
	72	45	-90	6878	0	71	0	0	0	80

	73	45	-90	6878	0	72	0	0	0	80
	74	45	-90	6878	0	73	0	0	0	80
	75	45	-90	6878	0	74	0	0	0	80
	76	45	-90	6878	0	75	0	0	0	80

				Initial Orbit						
Test #	**Run #**	**tgtlat**	**tgtlon**	**a**	**e**	**i**	**ω**	**Ω**	**v**	**Bank**
8	1	35	-180	6878	0	45	0	0	0	80
	2	35	-175	6878	0	1	0	0	0	80
	3	35	-170	6878	0	2	0	0	0	80
	4	35	-165	6878	0	3	0	0	0	80
	5	35	-160	6878	0	4	0	0	0	80
	6	35	-155	6878	0	5	0	0	0	80
	7	35	-150	6878	0	6	0	0	0	80
	8	35	-145	6878	0	7	0	0	0	80
	9	35	-140	6878	0	8	0	0	0	80
	10	35	-135	6878	0	9	0	0	0	80
	11	35	-130	6878	0	10	0	0	0	80
	12	35	-125	6878	0	11	0	0	0	80
	13	35	-120	6878	0	12	0	0	0	80
	14	35	-115	6878	0	13	0	0	0	80
	15	35	-110	6878	0	14	0	0	0	80
	16	35	-105	6878	0	15	0	0	0	80
	17	35	-100	6878	0	16	0	0	0	80
	18	35	-95	6878	0	17	0	0	0	80

	19	35	-90	6878	0	18	0	0	0	80
	20	35	-85	6878	0	19	0	0	0	80
	21	35	-80	6878	0	20	0	0	0	80
	22	35	-75	6878	0	21	0	0	0	80
	23	35	-70	6878	0	22	0	0	0	80
	24	35	-65	6878	0	23	0	0	0	80
	25	35	-60	6878	0	24	0	0	0	80
	26	35	-55	6878	0	25	0	0	0	80
	27	35	-50	6878	0	26	0	0	0	80
	28	35	-45	6878	0	27	0	0	0	80
	29	35	-40	6878	0	28	0	0	0	80
	30	35	-35	6878	0	29	0	0	0	80
	31	35	-30	6878	0	30	0	0	0	80
	32	35	-25	6878	0	31	0	0	0	80
	33	35	-20	6878	0	32	0	0	0	80
	34	35	-15	6878	0	33	0	0	0	80
	35	35	-10	6878	0	34	0	0	0	80
	36	35	-5	6878	0	35	0	0	0	80
	37	35	0	6878	0	36	0	0	0	80
	38	35	5	6878	0	37	0	0	0	80
	39	35	10	6878	0	38	0	0	0	80
	40	35	15	6878	0	39	0	0	0	80
	41	35	20	6878	0	40	0	0	0	80
	42	35	25	6878	0	41	0	0	0	80
	43	35	30	6878	0	42	0	0	0	80

	44	35	35	6878	0	43	0	0	0	80
	45	35	40	6878	0	44	0	0	0	80
	46	35	45	6878	0	45	0	0	0	80
	47	35	50	6878	0	46	0	0	0	80
	48	35	55	6878	0	47	0	0	0	80
	49	35	60	6878	0	48	0	0	0	80
	50	35	65	6878	0	49	0	0	0	80
	51	35	70	6878	0	50	0	0	0	80
	52	35	75	6878	0	51	0	0	0	80
	53	35	80	6878	0	52	0	0	0	80
	54	35	85	6878	0	53	0	0	0	80
	55	35	90	6878	0	54	0	0	0	80
	56	35	95	6878	0	55	0	0	0	80
	57	35	100	6878	0	56	0	0	0	80
	58	35	105	6878	0	57	0	0	0	80
	59	35	110	6878	0	58	0	0	0	80
	60	35	115	6878	0	59	0	0	0	80
	61	35	120	6878	0	60	0	0	0	80
	62	35	125	6878	0	61	0	0	0	80
	63	35	130	6878	0	62	0	0	0	80
	64	35	135	6878	0	63	0	0	0	80
	65	35	140	6878	0	64	0	0	0	80
	66	35	145	6878	0	65	0	0	0	80
	67	35	150	6878	0	66	0	0	0	80
	68	35	155	6878	0	67	0	0	0	80

	69	35	160	6878	0	68	0	0	0	80
	70	35	165	6878	0	69	0	0	0	80
	71	35	170	6878	0	70	0	0	0	80
	72	35	175	6878	0	71	0	0	0	80
	73	35	180	6878	0	72	0	0	0	80

Table B.2. Reachability Test Matrix

Test #	Inclination	Altitude (km)	Bank Angle
1	45	500	80
2	28.72	500	80
3	35	500	80
4	55	500	80
5	45	250	80
6	45	750	80
7	45	1000	80
8	45	500	75
9	45	500	85
10	45	500	90

Appendix C: MATLAB® Code for Overflight Simulation Dynamics Model & Reachability Model

Table C.1 Description of MATLAB Functions for Overflight Simulation

File Name	File Description
Optimal_Overfly_Manuevering.m	Main Overflight Simulation Dynamics Model Program
Optimal_Overfly_Inputs.m	Input File for Main Overflight Program
OrbEl_toPlanetFix.m	Converts Orbital Element Input to Inertial States
Inertial2Rel_States.m	Converts Inertial States to Relative States
delta_longitude.m	Calculate the longitude between to points on the Earth
d_lat_at_lon_func.m	Finds the latitude difference from the target at each target longitude crossing
d_lon_at_lat_func.m	Finds the longitude difference from the target at each target latitude crossing
equations_of_motion_ODE45.m	Equation of Motion Function to be used in Conjuction with MATLAB ODE 45 Function
AtmosModel.m	Atmospheric Model; finds the density for a given altitude
bank_section.m	Heuristics for determing the needed bank direction and maneuver section
states_at_lonpass.m	Finds the time and states at the target longitude crossings
states_at_latpass.m	Finds the time and states at the target latitude crossings
WGS84Constants	Defines WGS 84 Constant Values
Overfly_gui.m	Graphical User Interface for dV, TOA Calculator
globe_plot2.m	Plots Groundtrack of Original and Perturbed Orbits
plotstates.m	Plots all 6 states vs time for a given orbit
TOA_vs_dV_all.m	Plots the Time of Arrival vs. the dV for all maneuver types

Table C.2. Description of MATLAB Function for Reachability Simulation

File Name	File Description
Reachability.m	Reachability Main Function
Raan__Inc_Surface.m	Plots Reachbility Plot

Table C.3 Description of MATLAB Functions Used in Curve Fitting Process

File Name	File Description
fitness.m	Minimize Error function with Multiple Variables
data_sort.m	Sorts Data Created By Simulation Model
curve_fit_main.m	Finds Curve Fits for A, B, C Constants

Optimal_Overfly_Manuevering.m

```
%% --- Optimal Space Maneuvering for Ground Target Overfly Version 1.5
---
%                                                                  Devin
Dalton
%                                                                    7 Aug
2013
%
% This code calculates the optimal satellite manuever from a fuel
% consumption and delta-V perspective for a given target, time of
arrival,
% and initial orbit.
%
%
% Governing Equations:
%   - Three dimentional entry equations 3.35-3.37 & 3.65-3.67 on
%     p. 42 & 52 of Hicks text
%   - Variables are integrated in dimensional form
%
% Code assumes:
%   - Rotating planet
%   - Non-thrusting entry
%   - Non-planar entry (for planar, sigma set to 0 or 180 deg)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
```

```
clear all; clc; close all;


% Initialize Variables for each RAAN run
```

```
totalcount = 0;
test = 0;
directory = [];


% Sweep Through Desired Variable (ie tgtlat, inclination, RAAN)
for run = 1:28


% Save data from previous run before clearing data for new sweep
save('directory.mat','directory')
save('run.mat','run')
save('test.mat','test')
save('totalcount.mat','totalcount')
clear all;


% Declare Global Variables
global tgtlat tgtlon initialorbit startdate simtime vehicle inc_sign
global mu g0 Rearth wearth J2 J3 J4 J6 FlatE EccE Beta rho0 BR flagger2
global StefBoltz Boltz bank flagger1 index bankplot timer deltaV tstart
global flagger acc


% Bring in constants and define saved variables that were cleared above
bank = 0;
Optimal_Overfly_Inputs;       % Loads User Defined Inputs
WGS84Constants;               % Loads Earth WGS 84 Constants
test_num = 232;
lambda= 1e-6;
D = 1.15;
load('directory.mat','directory')
load('test.mat')
load('run.mat')
load('totalcount.mat')
tgtlat = tgtlat + run;
%initialorbit.i = initialorbit.i + run;
%initialorbit.RAAN = initialorbit.RAAN - 26.4 +24/10*run;




% Convert Initial Orbit to inertial R(km/s), Lat(rad), Long(rad),
V(km/s),
% fpa(rad), and heading angle(rad)
initial_states = OrbEl_to_PlanetFix(initialorbit, startdate);
Vp = sqrt(2*mu/initial_states(1));


% Error message if initial orbit is below the earth
starterror = [];
if initial_states(1) < Rearth
```

```
    display('Initial state is below the surface of the earth.')
    starterror = 1;
end


% Convert initial parameters from inertial to planet relative
rel_initial_states = Inertial2Rel_States(initial_states);
p_states = rel_initial_states;
p_states(4) = Vp;
p_rel_states = Inertial2Rel_States(p_states);
Vp_rel = p_rel_states(4);




 %%           -- Propagate initial orbit forward in time --


if isempty(starterror)     % Only perform if initial orbit is a valid
orbit
tic
options = odeset('RelTol',1e-6,'AbsTol',1e-6,'Events',@breakevent);
inc_sign = 0;
index = 1;
timer = [];
bankplot = [];
rel_initial_states(4) = rel_initial_states(4);
[time,rel_states] = ode45(@equations_of_motion_ODE45, ...
    [0 simtime],rel_initial_states,options);
rel_bankplot = bankplot;
rel_gs = acc;
reltimer = timer;


% Correct Longitude to be between -180 and 180 degrees
datapoints = length(time);
for i = 1:datapoints
    if rel_states(i,2) > pi
        rel_states(i:end,2) = -(pi - (rel_states(i:end,2) - pi));
    elseif rel_states(i:end,2) < -pi
        rel_states(i:end,2) = (pi + (rel_states(i:end,2) + pi));
    end
end


% Earth Intersection Warning
if time(end) ~= simtime
    display('Orbit intersects the earth.')
end
```

```
% Find the longitudinal and latitudinal difference at each point
for i = 1:datapoints
    dlon(i) = delta_longitude(rad2deg(rel_states(i,2)),tgtlon);
end
dlat = rad2deg(rel_states(:,3)) - tgtlat;
lat = rel_states(:,3);
lon = rel_states(:,2);
distance = acos(sin(lat).*sind(tgtlat) + ...
           cos(lat).*cosd(tgtlat).*cosd(transpose(dlon)))*Rearth;
ddist_dt = diff(distance);


% Find the lat/long difference for the closest point of each pass
count = 1;
for i = 2:datapoints-1
    if ddist_dt(i-1) < 0 && ddist_dt(i) > 0
        mindist_index(count) = i;
        count = count + 1;
    end
end
dlon_miss = dlon(mindist_index);
dlat_miss = dlat(mindist_index);


for i = 1:count-1
    y1 = rad2deg(lat(mindist_index(i) - 1));
    y2 = rad2deg(lat(mindist_index(i)));
    y3 = rad2deg(lat(mindist_index(i) + 1));
    x1 = rad2deg(lon(mindist_index(i) - 1));
    x2 = rad2deg(lon(mindist_index(i)));
    x3 = rad2deg(lon(mindist_index(i) + 1));
    m1 = (y2 - y1)/(x2 - x1);
    m2 = (y3 - y2)/(x3 - x2);
    xmin1 = (m1*(tgtlat -y2 + m1*x2) + tgtlon)/(1 + m1^2);
    xmin2 = (m2*(tgtlat -y3 + m2*x3) + tgtlon)/(1 + m2^2);
    ymin1 = y2 + m1*(xmin1 - x2);
    ymin2 = y3 + m2*(xmin2 - x2);
    min_dist1 = sqrt((tgtlon - xmin1)^2 + (tgtlat - ymin1)^2);
    min_dist2 = sqrt((tgtlon - xmin2)^2 + (tgtlat - ymin2)^2);
    if min_dist1 < min_dist2
        min_dist(i) = min_dist1;
    else
        min_dist(i) = min_dist2;
    end
end
phase_min_dist = min_dist;
plane_min_dist = min_dist;
skip_min_dist = min_dist;
overflycount = 0;
skip_overflycount = 0;


% Find the d-longitude for each target latitude passing
if abs(tgtlat) <= initialorbit.i
```

```
    [dlon_lat, skiptheta] = d_lon_at_lat_func(lon, lat, tgtlon,
tgtlat);
    [dlon_pass_states, dlon_pass_time] = states_at_pass(rel_states,
time, tgtlat);
end
[dlat_lon, phi] = d_lat_at_lon_func(lon, lat, tgtlon, tgtlat);
[pass_states, pass_time] = states_at_lonpass(rel_states, time, tgtlon);
Vp_rel = 11;


% Find the closest northern, easter, southern, and western points if
% tgtlat < inclination otherwise find just the north or souther points
if abs(tgtlat) < initialorbit.i
si = 0;
ni = 0;
for i = 1:length(dlat_lon)
    if dlat_lon(i) < 0
        si = si + 1;
        s_points(si) = -dlat_lon(i);
    else
        ni = ni + 1;
        n_points(ni) = dlat_lon(i);
    end
end
n_point_deg = min(n_points);
s_point_deg = min(s_points);
n_point = acos(sind(tgtlat + n_point_deg)*sind(tgtlat) + ...
          cosd(tgtlat + n_point_deg)*cosd(tgtlat))*Rearth;
s_point = acos(sind(tgtlat - s_point_deg)*sind(tgtlat) + ...
          cosd(tgtlat - s_point_deg)*cosd(tgtlat))*Rearth;
n_index = find(dlat_lon == n_point_deg);
s_index = find(-dlat_lon == s_point_deg);
n_time = pass_time(n_index);
s_time = pass_time(s_index);
if abs(tgtlat) <= initialorbit.i
    ei = 0;
    wi = 0;
    for i = 1:length(dlon_lat)
        if dlon_lat(i) < 0
            wi = wi + 1;
            w_points(wi) = -dlon_lat(i);
        else
            ei = ei + 1;
            e_points(ei) = dlon_lat(i);
        end
    end
    e_point_deg = min(e_points);
    w_point_deg = min(w_points);
    e_point = acos(sind(tgtlat)*sind(tgtlat) + ...
          cosd(tgtlat)*cosd(tgtlat)*cosd(e_point_deg))*Rearth;
    w_point = acos(sind(tgtlat)*sind(tgtlat) + ...
          cosd(tgtlat)*cosd(tgtlat)*cosd(-w_point_deg))*Rearth;
    e_index = find(dlon_lat == e_point_deg);
    w_index = find(-dlon_lat == w_point_deg);
    e_time = dlon_pass_time(e_index);
```

```
    w_time = dlon_pass_time(w_index);
else
    e_point = 0;
    w_point = 0;
    e_time = 0;
    w_time = 0;
end
elseif tgtlat > 0
    n_point = 0;
    e_point = 0;
    w_point = 0;
    n_time = 0;
    e_time = 0;
    w_time = 0;
    si = 0;
    for i = 1:length(dlat_lon)
        if dlat_lon(i) < 0
            si = si + 1;
            s_points(si) = -dlat_lon(i);
        end
    end
    s_point_deg = min(s_points);
    s_point = acos(sind(tgtlat - s_point_deg)*sind(tgtlat) + ...
              cosd(tgtlat - s_point_deg)*cosd(tgtlat))*Rearth;
    s_index = find(-dlat_lon == s_point_deg);
    s_time = pass_time(s_index);
elseif tgtlat < 0
    s_point = 0;
    e_point = 0;
    w_point = 0;
    s_time = 0;
    e_time = 0;
    w_time = 0;
    ni = 0;
    for i = 1:length(dlat_lon)
        if dlat_lon(i) > 0
            ni = ni + 1;
            n_points(ni) = dlat_lon(i);
        end
    end
    n_point_deg = min(n_points);
    n_point = acos(sind(tgtlat + n_point_deg)*sind(tgtlat) + ...
              cosd(tgtlat + n_point_deg)*cosd(tgtlat))*Rearth;
    n_index = find(dlat_lon == n_point_deg);
    n_time = pass_time(n_index);
end


% Find the 4 section times to be used to correspond bank with the
correct
% direction in ground track shift
t1count = 0;
t2count = 0;
t3count = 0;
t4count = 0;
```

```
for i = 2:length(time)
    if rel_states(i-1,3) < 0 && rel_states(i,3) > 0
        t1count = t1count + 1;
        t1(t1count,1) = time(i);
        t1(t1count,2) = 1;
    elseif rel_states(i-1,6) > 0 && rel_states(i,6) < 0
        t2count = t2count + 1;
        t2(t2count,1) = time(i);
        t2(t2count,2) = 2;
    elseif rel_states(i-1,3) > 0 && rel_states(i,3) < 0
        t3count = t3count + 1;
        t3(t3count,1) = time(i);
        t3(t3count,2) = 3;
    elseif rel_states(i-1,6) < 0 && rel_states(i,6) > 0
        t4count = t4count + 1;
        t4(t4count,1) = time(i);
        t4(t4count,2) = 4;
    end
end
tsection1 = [t1 ; t2; t3; t4];
[tsection2 tindex] = sort(tsection1(:,1));
for i = 1:length(tsection1)
    tsection_start(i,1) = tsection2(i,1);
    tsection_start(i,2) = tsection1(tindex(i),2);
end
globe_plot(rel_states)

 %% ------------------------ PHASING MANEUVER -------------------------
----


if abs(tgtlat) <= initialorbit.i
for i = 1:length(dlon_lat)


    % Initialize loop variables
    miss_tol = .1;                          % distance tolerance in km
    phase_dlon_lat = dlon_lat(i);
    phase_distance(1:length(distance),i) = distance;
    phase_initial_states(i,:) = rel_initial_states;
    phase_dv_i = 0.25;
    k=0;


    while abs(phase_dlon_lat) > miss_tol %-------------------------


        % Algorithm to change the step size of the dv correction
        k = k+1;
        if phase_dlon_lat > 0
            zeroin(k) = 1;
        else
            zeroin(k) = -1;
```

```
        end
        if k > 1
            if zeroin(k) == -zeroin(k-1)
                phase_dv_i = phase_dv_i/2;
            end
        end
        if k > 2
            if zeroin(k) == zeroin(k-1) && zeroin(k) == -zeroin(k-2)
                phase_dv_i = phase_dv_i/2;
            end
        end


        % Change initial velocity in order to produce desired
groundtrack
        if phase_dlon_lat > 0;
            phase_initial_states(i,4) = phase_initial_states(i,4) ...
                + phase_dv_i;
        else
            phase_initial_states(i,4) = phase_initial_states(i,4) ...
                - phase_dv_i;
        end
        phase_V = phase_initial_states(i,4);


        % Calculate the flight profile using calculated initial
conditions
        index = 1;
        timer = [];
        bankplot = [];
        acc = [];
        [phase_time_i,phase_states_i] =
ode45(@equations_of_motion_ODE45,...
                [0 simtime],phase_initial_states(i,:),options);
        phase_points = length(phase_time_i);


       % Correct Longitude to be from -180 to 180
        for ii = 1:phase_points
            if phase_states_i(ii,2) > pi
                phase_states_i(ii:end,2) = -(pi -
(phase_states_i(ii:end,2) - pi));
            elseif phase_states_i(ii:end,2) < -pi
                phase_states_i(ii:end,2) = (pi +
(phase_states_i(ii:end,2) + pi));
            end
        end
        phase_lat_i = phase_states_i(:,3);
        phase_lon_i = phase_states_i(:,2);
        phase_lat_i(phase_points+1:end) = [];


        % Calculate the lat/long difference and distance from the
groundtrack
```

```
        for ii = 1:phase_points
            phase_dlon(ii) =
delta_longitude(rad2deg(phase_states_i(ii,2)),tgtlon);
        end
        phase_dlat = rad2deg(phase_states_i(:,3)) - tgtlat;
        phase_dlon(phase_points+1:end) = [];
        phase_dlat(phase_points+1:end) = [];
        phase_distance(1:phase_points,i) =
acos(sin(phase_lat_i).*sind(tgtlat) + ...
                  cos(phase_lat_i).*cosd(tgtlat).* ...
                  cosd(transpose(phase_dlon)))*Rearth;
        phase_ddist_dt = diff(phase_distance(:,i));
        phase_mindist_index = [];

        %globe_plot2(rel_states,phase_states_i(:,:))
        % Calculate the longitudinal difference from each target
latitude
        % crossing, then check for hyperbolic, re-entry, and less than
        % simulation time errors.
        [phase_dlon_lat_i,phase_theta] = d_lon_at_lat_func(phase_lon_i,
phase_lat_i, ...
                        tgtlon, tgtlat);
        if phase_time_i(end) ~= simtime
            last_distance = acos(sin(phase_lat_i(end)).*sind(tgtlat) +
...
                           cos(phase_lat_i(end)).*cosd(tgtlat).* ...
                           cosd(phase_dlon(end)))*Rearth;
            if phase_dv_i < 0.01 && last_distance > 10
                error_message = ['Pass ' num2str(i) ' cannot be ' ...
                    ,'reached via a phase maneuver without ' ...
                    ,'intersecting the earth'];
                disp(error_message);
                unable = 1;
                break
            else
                phase_dlon_lat = 5;
                phase_theta(i) = tgtlon + 5;
            end
        elseif length(phase_dlon_lat_i) < i && phase_V < Vp_rel
            last_distance = acos(sin(phase_lat_i(end)).*sind(tgtlat) +
...
                cos(phase_lat_i(end)).*cosd(tgtlat).* ...
                cosd(phase_dlon(end)))*Rearth;
            if phase_dv_i < 0.01 && last_distance > 10
                error_message = ['Pass ' num2str(i) ' cannot be ' ...
                    ,'reached via a phase maneuver in allotted time'];
                disp(error_message);
                unable = 1;
                break
            else
                phase_dlon_lat = -5;
                phase_theta(i) = tgtlon - 5;
            end
        elseif phase_V > Vp_rel
            if phase_dv_i < 0.001
```

```
                error_message = ['Pass ' num2str(i) ' cannot be ' ...
                    ,'reached via a phase maneuver because the delta '
...
                    ,'would cause a hyperbolic orbit'];
                disp(error_message);
                unable = 1;
                break
            else
                phase_dlon_lat = -5;
                phase_theta(i) = tgtlon - 5;
            end
        else
            phase_dlon_lat = phase_dlon_lat_i(i);
            unable = [];
        end

        if k > 40
            disp('not sure why I coudn''t get this one')
            unable = 1;
            break
        end


    end   %-------------- End of while loop ----------------------


    if k == 0
    phase_gs_i = rel_gs;
    phase_states_i = rel_states;
    phase_time_i = time;
    overflycount = overflycount + 1;


    % Interpolate other data to find data at actual pass
    [phase_pass_states_i, phase_pass_time_i] = ...
        states_at_lonpass(phase_states_i, phase_time_i, tgtlon);


    % Resize array size for saving wanted data each loop
    phase_points = length(time);
    num1(i) = phase_points;
    if i > 1
        if phase_points < max(num1(1:i-1))
            phase_states_i(phase_points+1:max(num1(1:i-1)),:) = ...
                zeros(max(num1(1:i-1)) - phase_points,6);
            phase_time_i(phase_points+1:max(num1(1:i-1))) = ...
                zeros(max(num1(1:i-1)) - phase_points,1);
        elseif phase_points > max(num1(1:i-1))
            phase_states(max(num1(1:i-1))+1:phase_points,:,1:i-1) = ...
                zeros(phase_points - max(num1(1:i-1)),6,i-1);
            phase_time(max(num1(1:i-1))+1:phase_points,1:i-1) = ...
                zeros(phase_points - max(num1(1:i-1)),i-1);
        end
```

```
    end
    phase_states(:,:,overflycount) = phase_states_i(:,:);
    phase_dv(overflycount) = abs(phase_initial_states(i,4) - ...
                            rel_initial_states(4));
    phase_time(:,overflycount) = phase_time_i;
    phase_pass_states(overflycount,:) = phase_pass_states_i(i,:);
    phase_pass_time(overflycount) = phase_pass_time_i(i);


    % Save all wanted data into "test" variable
    phasecount = totalcount + overflycount;
    test(phasecount,1) = test_num;
    test(phasecount,2) = run;
    test(phasecount,3) = tgtlat;
    test(phasecount,4) = tgtlon;
    test(phasecount,5) = bank;
    test(phasecount,6) = initialorbit.a;
    test(phasecount,7) = initialorbit.e;
    test(phasecount,8) = initialorbit.i;
    test(phasecount,9) = initialorbit.RAAN;
    test(phasecount,10) = initialorbit.AoP;
    test(phasecount,11) = initialorbit.nu;
    test(phasecount,12) = initial_states(4);
    test(phasecount,13) = rad2deg(initial_states(3));
    test(phasecount,14) = rad2deg(initial_states(2));
    test(phasecount,15) = n_point;
    test(phasecount,16) = e_point;
    test(phasecount,17) = w_point;
    test(phasecount,18) = s_point;
    test(phasecount,19) = n_time;
    test(phasecount,20) = e_time;
    test(phasecount,21) = w_time;
    test(phasecount,22) = s_time;
    test(phasecount,23) = 2;
    test(phasecount,24) = i;
    test(phasecount,25) = 0;
    test(phasecount,26) = phase_pass_time(overflycount);
    test(phasecount,27) = 0;
    test(phasecount,28) = 0;
    test(phasecount,29) = 0;
    test(phasecount,30) = 0;
    test(phasecount,31) = 0;
    test(phasecount,32) = 0;
    test(phasecount,33) = 0;
    test(phasecount,34) = rel_initial_states(1);
    test(phasecount,35) = rel_initial_states(1);
    test(phasecount,36) = rel_initial_states(1);
    test(phasecount,37) = rel_initial_states(1) - Rearth;
    test(phasecount,38) = rel_initial_states(1) - Rearth;
    test(phasecount,39) = rel_initial_states(1) - Rearth;
    test(phasecount,40) = max(phase_gs_i);
    test(phasecount,41) = 0;
    test(phasecount,42) = 2.44*(phase_pass_states(overflycount,1)-
Rearth)*lambda/D;
```

```
else

    if isempty(unable)  % Only perform if successful overflight
achieved


        overflycount = overflycount + 1;
        globe_plot2(rel_states,phase_states_i(:,:))
%         plotm([tgtlat tgtlat],[tgtlon phase_theta(i)],'r')
%         plotstates(phase_states_i,phase_time_i)



        % Interpolate other data to find data at actual pass
        [phase_pass_states_i, phase_pass_time_i] = ...
            states_at_pass(phase_states_i, phase_time_i, tgtlat);


        % Resize array size for saving wanted data each loop
        num1(i) = phase_points;
        if i > 1
            if phase_points < max(num1(1:i-1))
                phase_states_i(phase_points+1:max(num1(1:i-1)),:) = ...
                    zeros(max(num1(1:i-1)) - phase_points,6);
                phase_time_i(phase_points+1:max(num1(1:i-1))) = ...
                    zeros(max(num1(1:i-1)) - phase_points,1);
            elseif phase_points > max(num1(1:i-1))
                phase_states(max(num1(1:i-1))+1:phase_points,:,1:i-1) =
...
                    zeros(phase_points - max(num1(1:i-1)),6,i-1);
                phase_time(max(num1(1:i-1))+1:phase_points,1:i-1) = ...
                    zeros(phase_points - max(num1(1:i-1)),i-1);
            end
        end
        phase_states(:,:,overflycount) = phase_states_i(:,:);
        phase_dv(overflycount) = abs(phase_initial_states(i,4) - ...
                                 rel_initial_states(4));
        phase_time(:,overflycount) = phase_time_i;
        phase_pass_states(overflycount,:) = phase_pass_states_i(i,:);
        phase_pass_time(overflycount) = phase_pass_time_i(i);


        % Save all wanted data into "test" variable
        phasecount = overflycount + totalcount;
        test(phasecount,1) = test_num;
        test(phasecount,2) = run;
        test(phasecount,3) = tgtlat;
        test(phasecount,4) = tgtlon;
        test(phasecount,5) = 0;
        test(phasecount,6) = initialorbit.a;
        test(phasecount,7) = initialorbit.e;
        test(phasecount,8) = initialorbit.i;
        test(phasecount,9) = initialorbit.RAAN;
```

```
        test(phasecount,10) = initialorbit.AoP;
        test(phasecount,11) = initialorbit.nu;
        test(phasecount,12) = initial_states(4);
        test(phasecount,13) = rad2deg(initial_states(3));
        test(phasecount,14) = rad2deg(initial_states(2));
        test(phasecount,15) = n_point;
        test(phasecount,16) = e_point;
        test(phasecount,17) = w_point;
        test(phasecount,18) = s_point;
        test(phasecount,19) = n_time;
        test(phasecount,20) = e_time;
        test(phasecount,21) = w_time;
        test(phasecount,22) = s_time;
        test(phasecount,23) = 1;
        test(phasecount,24) = i;
        test(phasecount,25) = 0;
        test(phasecount,26) = phase_pass_time(overflycount);
        test(phasecount,27) = phase_dv(overflycount);
        test(phasecount,28) = 0;
        test(phasecount,29) = 0;
        test(phasecount,30) = 0;
        test(phasecount,31) = phase_dv(overflycount);
        test(phasecount,32) = phase_dv(overflycount);
        test(phasecount,33) = 0;
        test(phasecount,34) = min(phase_states_i(:,1));
        test(phasecount,35) = max(phase_states_i(:,1));
        test(phasecount,36) = phase_pass_states(overflycount,1);
        test(phasecount,37) = min(phase_states_i(:,1)) - Rearth;
        test(phasecount,38) = max(phase_states_i(:,1)) - Rearth;
        test(phasecount,39) = phase_pass_states(overflycount,1) -
Rearth;
        test(phasecount,40) = 0;
        test(phasecount,41) = 0;
        test(phasecount,42) = 2.44*(phase_pass_states(overflycount,1)-
Rearth)*lambda/D;



    end

    end
end
else
    phasecount = totalcount;
end


% %   ---------------------- SIMPLE PLANE CHANGE ----------------------
----


% Initialize outerloop variables
counter1 = 0;
miss_tol = .1;
```

```
plane_overflycount = 0;


% Overflights exist using a simple plane change for longitude crossings
% that lie in the same hemisphere as the target latitude
for i = 1:length(dlat_lon)
    if phi(i) > 0 && tgtlat >= 0
        counter1 = counter1 + 1;
        plane_index(counter1) = i;
    elseif phi(i) < 0 && tgtlat <= 0
        counter1 = counter1 + 1;
        plane_index(counter1) = i;
    end
end


% Loop through the number of possible overflights
for i = 1:counter1


    % Initialize innerloop variables
    plane_dlat_lon = dlat_lon(plane_index(i));
    plane_initial_states(i,:) = rel_initial_states;
    plane_initialorbit = initialorbit;
    plane_inc = initialorbit.i;
    plane_di = 5;
    k=0;
    less_inc = 0;


    % Iterate orbit inclination until overflight is within tolerance
    while abs(plane_dlat_lon) > miss_tol


        % Algorithm to change the step size of the d-inclination
correction
        k = k+1;
        if plane_dlat_lon > 0
            zeroin(k) = 1;
        elseif less_inc == 1
            zeroin(k) = -zeroin(k-1);
        else
            zeroin(k) = -1;
        end
        if k > 1
            if zeroin(k) == -zeroin(k-1)
                plane_di = plane_di/2;
            end
        end
        if k > 2
            if zeroin(k) == zeroin(k-1) && zeroin(k) == -zeroin(k-2)
                plane_di = plane_di/2;
            end
```

```
        end


        % Change iinclination in order to produce desired groundtrack
        wrong_grade = [];
        if tgtlat == 0 && plane_inc <= 90;
            plane_inc = 0;
        elseif tgtlat == 0 && plane_inc > 90;
            plane_inc = 180;
        elseif less_inc == 1
            plane_inc = plane_inc - plane_di;
        elseif plane_dlat_lon > 0 && tgtlat > 0 && plane_inc <= 90;
            plane_inc = plane_inc - plane_di;
        elseif plane_dlat_lon < 0 && tgtlat > 0 && plane_inc <= 90;
            plane_inc = plane_inc + plane_di;
        elseif plane_dlat_lon > 0 && tgtlat < 0 && plane_inc <= 90;
            plane_inc = plane_inc + plane_di;
        elseif plane_dlat_lon < 0 && tgtlat < 0 && plane_inc <= 90;
            plane_inc = plane_inc - plane_di;
        elseif plane_dlat_lon > 0 && tgtlat > 0 && plane_inc > 90;
            plane_inc = plane_inc + plane_di;
        elseif plane_dlat_lon < 0 && tgtlat > 0 && plane_inc > 90;
            plane_inc = plane_inc - plane_di;
        elseif plane_dlat_lon > 0 && tgtlat < 0 && plane_inc > 90;
            plane_inc = plane_inc - plane_di;
        elseif plane_dlat_lon < 0 && tgtlat < 0 && plane_inc > 90;
            plane_inc = plane_inc + plane_di;
        end
        if initialorbit.i > 90 && plane_inc <= 90
            plane_inc = plane_inc + plane_di;
            plane_di = plane_di/2;
            wrong_grade = 1;
        elseif initialorbit.i < 90 && plane_inc >= 90
            plane_inc =  plane_inc - plane_di;
            plane_di = plane_di/2;
            wrong_grade = 1;
        end
        if isempty(wrong_grade)
        plane_initialorbit.i = plane_inc;


        % Convert from orbital parameters to earth relative states
        plane_inert_initial_states = ...
            OrbEl_to_PlanetFix(plane_initialorbit, startdate);
        plane_initial_states = ...
            Inertial2Rel_States(plane_inert_initial_states);
        plane_initial_states(4) = plane_initial_states(4);


        % Calculate the flight profile using calculated initial
conditions
        index = 1;
        timer = [];
        bankplot = [];
```

```
        acc = [];
        [plane_time_i,plane_states_i] = ...
            ode45(@equations_of_motion_ODE45,...
                [0 simtime],plane_initial_states,options);
        plane_points = length(plane_time_i);


       % Correct Longitude to be from -180 to 180
        for ii = 1:plane_points
            if plane_states_i(ii,2) > pi
                plane_states_i(ii:end,2) = ...
                    -(pi - (plane_states_i(ii:end,2) - pi));
            elseif plane_states_i(ii:end,2) < -pi
                plane_states_i(ii:end,2) = ...
                    (pi + (plane_states_i(ii:end,2) + pi));
            end
        end
        plane_lat_i = plane_states_i(:,3);
        plane_lon_i = plane_states_i(:,2);
        plane_lat_i(plane_points+1:end) = [];


        % Calculate the lat/long difference & distance from the
groundtrack
        for ii = 1:plane_points
            plane_dlon(ii) = ...
                delta_longitude(rad2deg(plane_states_i(ii,2)),tgtlon);
        end
        plane_dlat = rad2deg(plane_states_i(:,3)) - tgtlat;
        plane_dlon(plane_points+1:end) = [];
        plane_dlat(plane_points+1:end) = [];


        % Calculate the latitudinal difference from each target
longitude
        % crossing
        [plane_dlat_lon_i,plane_phi_i] = ...
            d_lat_at_lon_func(plane_lon_i, plane_lat_i,tgtlon, tgtlat);
        if plane_index(i) <= length(plane_dlat_lon_i)
            plane_dlat_lon = plane_dlat_lon_i(plane_index(i));
            plane_phi = plane_phi_i(plane_index(i));
            less_inc = 0;
        else
            less_inc = 1;
        end
        end


        if k > 30
            error_message = ['Pass ' num2str(i) ' can''t be reached '
...
                ,'within 30 iterations'];
            disp(error_message);
            unable = 1;
```

```
            break
        else
            unable = [];
        end
%         globe_plot(plane_states_i)
%         plotm([tgtlat plane_phi],[tgtlon tgtlon],'r')



    end  %------------------ End of while loop ------------------------
--%


if k == 0
    plane_gs_i = rel_gs;
    plane_states_i = rel_states;
    plane_time_i = time;
    plane_overflycount = plane_overflycount + 1;


    % Interpolate other data to find data at actual pass
    [plane_pass_states_i, plane_pass_time_i] = ...
        states_at_lonpass(plane_states_i, plane_time_i, tgtlon);


    % Resize array size for saving wanted data each loop
    plane_points = length(time);
    num1(i) = plane_points;
    if i > 1
        if plane_points < max(num1(1:i-1))
            plane_states_i(plane_points+1:max(num1(1:i-1)),:) = ...
                zeros(max(num1(1:i-1)) - plane_points,6);
            plane_time_i(plane_points+1:max(num1(1:i-1))) = ...
                zeros(max(num1(1:i-1)) - plane_points,1);
        elseif plane_points > max(num1(1:i-1))
            plane_states(max(num1(1:i-1))+1:plane_points,:,1:i-1) = ...
                zeros(plane_points - max(num1(1:i-1)),6,i-1);
            plane_time(max(num1(1:i-1))+1:plane_points,1:i-1) = ...
                zeros(plane_points - max(num1(1:i-1)),i-1);
        end
    end
    plane_states(:,:,plane_overflycount) = plane_states_i(:,:);
    plane_dv(plane_overflycount) = 2*plane_inert_initial_states(4)* ...
        sind(abs(plane_initialorbit.i - initialorbit.i)/2);
    plane_time(:,plane_overflycount) = plane_time_i;
    plane_pass_states(plane_overflycount,:) =
plane_pass_states_i(plane_index(i),:);
    plane_pass_time(plane_overflycount) =
plane_pass_time_i(plane_index(i));

    % Save all wanted data into "test" variable
    planecount = phasecount + plane_overflycount;
    test(planecount,1) = test_num;
```

```
    test(planecount,2) = run;
    test(planecount,3) = tgtlat;
    test(planecount,4) = tgtlon;
    test(planecount,5) = bank;
    test(planecount,6) = initialorbit.a;
    test(planecount,7) = initialorbit.e;
    test(planecount,8) = initialorbit.i;
    test(planecount,9) = initialorbit.RAAN;
    test(planecount,10) = initialorbit.AoP;
    test(planecount,11) = initialorbit.nu;
    test(planecount,12) = initial_states(4);
    test(planecount,13) = rad2deg(initial_states(3));
    test(planecount,14) = rad2deg(initial_states(2));
    test(planecount,15) = n_point;
    test(planecount,16) = e_point;
    test(planecount,17) = w_point;
    test(planecount,18) = s_point;
    test(planecount,19) = n_time;
    test(planecount,20) = e_time;
    test(planecount,21) = w_time;
    test(planecount,22) = s_time;
    test(planecount,23) = 2;
    test(planecount,24) = i;
    test(planecount,25) = 0;
    test(planecount,26) = plane_pass_time(plane_overflycount);
    test(planecount,27) = 0;
    test(planecount,28) = 0;
    test(planecount,29) = 0;
    test(planecount,30) = 0;
    test(planecount,31) = 0;
    test(planecount,32) = 0;
    test(planecount,33) = 0;
    test(planecount,34) = rel_initial_states(1);
    test(planecount,35) = rel_initial_states(1);
    test(planecount,36) = rel_initial_states(1);
    test(planecount,37) = rel_initial_states(1) - Rearth;
    test(planecount,38) = rel_initial_states(1) - Rearth;
    test(planecount,39) = rel_initial_states(1) - Rearth;
    test(planecount,40) = max(plane_gs_i);
    test(planecount,41) = 0;
    test(planecount,42) =
2.44*(plane_pass_states(plane_overflycount,1)-Rearth)*lambda/D;

else

if isempty(unable)

    plane_overflycount = plane_overflycount + 1;


    % Interpolate other data to find data at actual pass
    [plane_pass_states_i, plane_pass_time_i] = ...
        states_at_lonpass(plane_states_i, plane_time_i, tgtlon);
```

```
    % Resize array size for saving wanted data each loop
    num1(i) = plane_points;
    if i > 1
        if plane_points < max(num1(1:i-1))
            plane_states_i(plane_points+1:max(num1(1:i-1)),:) = ...
                zeros(max(num1(1:i-1)) - plane_points,6);
            plane_time_i(plane_points+1:max(num1(1:i-1))) = ...
                zeros(max(num1(1:i-1)) - plane_points,1);
        elseif plane_points > max(num1(1:i-1))
            plane_states(max(num1(1:i-1))+1:plane_points,:,1:i-1) = ...
                zeros(plane_points - max(num1(1:i-1)),6,i-1);
            plane_time(max(num1(1:i-1))+1:plane_points,1:i-1) = ...
                zeros(plane_points - max(num1(1:i-1)),i-1);
        end
    end
    plane_states(:,:,plane_overflycount) = plane_states_i(:,:);
    plane_dv(plane_overflycount) = 2*plane_inert_initial_states(4)* ...
        sind(abs(plane_initialorbit.i - initialorbit.i)/2);
    plane_time(:,plane_overflycount) = plane_time_i;
    plane_pass_states(plane_overflycount,:) =
plane_pass_states_i(plane_index(i),:);
    plane_pass_time(plane_overflycount) =
plane_pass_time_i(plane_index(i));


    % Save all wanted data into "test" variable
    planecount = phasecount + plane_overflycount;
    test(planecount,1) = test_num;
    test(planecount,2) = run;
    test(planecount,3) = tgtlat;
    test(planecount,4) = tgtlon;
    test(planecount,5) = 0;
    test(planecount,6) = initialorbit.a;
    test(planecount,7) = initialorbit.e;
    test(planecount,8) = initialorbit.i;
    test(planecount,9) = initialorbit.RAAN;
    test(planecount,10) = initialorbit.AoP;
    test(planecount,11) = initialorbit.nu;
    test(planecount,12) = initial_states(4);
    test(planecount,13) = rad2deg(initial_states(3));
    test(planecount,14) = rad2deg(initial_states(2));
    test(planecount,15) = n_point;
    test(planecount,16) = e_point;
    test(planecount,17) = w_point;
    test(planecount,18) = s_point;
    test(planecount,19) = n_time;
    test(planecount,20) = e_time;
    test(planecount,21) = w_time;
    test(planecount,22) = s_time;
    test(planecount,23) = 2;
    test(planecount,24) = i;
    test(planecount,25) = 0;
    test(planecount,26) = plane_pass_time(plane_overflycount);
    test(planecount,27) = plane_dv(plane_overflycount);
```

```
    test(planecount,28) = 0;
    test(planecount,29) = 0;
    test(planecount,30) = 0;
    test(planecount,31) = plane_dv(plane_overflycount);
    test(planecount,32) = 0;
    test(planecount,33) = plane_initialorbit.i - initialorbit.i;
    test(planecount,34) = min(plane_states_i(:,1));
    test(planecount,35) = max(plane_states_i(:,1));
    test(planecount,36) = plane_pass_states(plane_overflycount,1);
    test(planecount,37) = min(plane_states_i(:,1)) - Rearth;
    test(planecount,38) = max(plane_states_i(:,1)) - Rearth;
    test(planecount,39) = plane_pass_states(plane_overflycount,1) -
Rearth;
    test(planecount,40) = 0;
    test(planecount,41) = 0;
    test(planecount,42) =
2.44*(plane_pass_states(plane_overflycount,1)-Rearth)*lambda/D;



 %globe_plot2(rel_states,plane_states_i)
 %plotm([tgtlat plane_phi],[tgtlon tgtlon],'r')
end
end
end


%--------------- Repeat above for an opposite grade orbit -------------
---%


initialorbit2 = initialorbit;
initialorbit2.i = 180 - initialorbit.i;
initial_states2 = OrbEl_to_PlanetFix(initialorbit2, startdate);
rel_initial_states2 = Inertial2Rel_States(initial_states2);
rel_initial_states2(4) = rel_initial_states2(4);
index = 1;
timer = [];
bankplot = [];
[time2,rel_states2] = ode45 ...
    (@equations_of_motion_ODE45,[0
simtime],rel_initial_states2,options);


% Correct Longitude to be between -180 and 180 degrees
datapoints2 = length(time2);
for i = 1:datapoints2
    if rel_states2(i,2) > pi
        rel_states2(i:end,2) = -(pi - (rel_states2(i:end,2) - pi));
    elseif rel_states2(i:end,2) < -pi
        rel_states2(i:end,2) = (pi + (rel_states2(i:end,2) + pi));
    end
end
```

```
% Find the longitudinal and latitudinal difference at each point
for i = 1:datapoints2
    dlon2(i) = delta_longitude(rad2deg(rel_states2(i,2)),tgtlon);
end
dlat2 = rad2deg(rel_states2(:,3)) - tgtlat;
lat2 = rel_states2(:,3);
lon2 = rel_states2(:,2);


% Find the d-longitude for each target latitude passing
[dlat_lon2, phi2] = d_lat_at_lon_func(lon2, lat2, tgtlon, tgtlat);


% Overflights exist using a simple plane change for longitude crossings
% that lie in the same hemisphere as the target latitude
counter2 = counter1;
for i = 1:length(dlat_lon2)
    if phi2(i) > 0 && tgtlat >= 0
        counter2 = counter2 + 1;
        plane_index(counter2) = i;
    elseif phi2(i) < 0 && tgtlat <= 0
        counter2 = counter2 + 1;
        plane_index(counter2) = i;
    end
end


% Loop through the number of possible overflights
for i = counter1+1:counter2


    % Initialize innerloop variables
    plane_dlat_lon = dlat_lon2(plane_index(i));
    plane_initial_states(i,:) = rel_initial_states2;
    plane_initialorbit = initialorbit2;
    plane_inc = initialorbit2.i;
    plane_di = 5;
    k=0;


    % Iterate orbit inclination until overflight is within tolerance
    while abs(plane_dlat_lon) > miss_tol %-------------------------


        % Algorithm to change the step size of the d-inclination
correction
        k = k+1;
        if plane_dlat_lon > 0
            zeroin(k) = 1;
        else
```

```
    zeroin(k) = -1;
end
if k > 1
    if zeroin(k) == -zeroin(k-1)
        plane_di = plane_di/2;
    end
end
if k > 2
    if zeroin(k) == zeroin(k-1) && zeroin(k) == -zeroin(k-2)
        plane_di = plane_di/2;
    end
end


% Change iinclination in order to produce desired groundtrack
wrong_grade = [];
if tgtlat == 0 && plane_inc <= 90;
    plane_inc = 0;
elseif tgtlat == 0 && plane_inc > 90;
    plane_inc = 180;
elseif plane_dlat_lon > 0 && tgtlat > 0 && plane_inc <= 90;
    plane_inc = plane_inc - plane_di;
elseif plane_dlat_lon < 0 && tgtlat > 0 && plane_inc <= 90;
    plane_inc = plane_inc + plane_di;
elseif plane_dlat_lon > 0 && tgtlat < 0 && plane_inc <= 90;
    plane_inc = plane_inc + plane_di;
elseif plane_dlat_lon < 0 && tgtlat < 0 && plane_inc <= 90;
    plane_inc = plane_inc - plane_di;
elseif plane_dlat_lon > 0 && tgtlat > 0 && plane_inc > 90;
    plane_inc = plane_inc + plane_di;
elseif plane_dlat_lon < 0 && tgtlat > 0 && plane_inc > 90;
    plane_inc = plane_inc - plane_di;
elseif plane_dlat_lon > 0 && tgtlat < 0 && plane_inc > 90;
    plane_inc = plane_inc - plane_di;
elseif plane_dlat_lon < 0 && tgtlat < 0 && plane_inc > 90;
    plane_inc = plane_inc + plane_di;
end
if initialorbit2.i > 90 && plane_inc <= 90
    plane_inc = plane_inc + plane_di;
    plane_di = plane_di/2;
    wrong_grade = 1;
elseif initialorbit2.i < 90 && plane_inc >= 90
    plane_inc =  plane_inc - plane_di;
    plane_di = plane_di/2;
    wrong_grade = 1;
end
if isempty(wrong_grade)
plane_initialorbit.i = plane_inc;


% Convert from orbital parameters to earth relative states
plane_inert_initial_states = ...
    OrbEl_to_PlanetFix(plane_initialorbit, startdate);
plane_initial_states = ...
```

```
            Inertial2Rel_States(plane_inert_initial_states);


        % Calculate the flight profile using calculated initial
conditions
        index = 1;
        timer = [];
        bankplot = [];
        acc = [];
        [plane_time_i,plane_states_i]
=ode45(@equations_of_motion_ODE45,...
                [0 simtime],plane_initial_states,options);
        plane_points = length(plane_time_i);


       % Correct Longitude to be from -180 to 180
        for ii = 1:plane_points
            if plane_states_i(ii,2) > pi
                plane_states_i(ii:end,2) = ...
                    -(pi - (plane_states_i(ii:end,2) - pi));
            elseif plane_states_i(ii:end,2) < -pi
                plane_states_i(ii:end,2) = ...
                    (pi + (plane_states_i(ii:end,2) + pi));
            end
        end
        plane_lat_i = plane_states_i(:,3);
        plane_lon_i = plane_states_i(:,2);
        plane_lat_i(plane_points+1:end) = [];


        % Calculate the lat/long difference & distance from the
groundtrack
        for ii = 1:plane_points
            plane_dlon(ii) = ...
                delta_longitude(rad2deg(plane_states_i(ii,2)),tgtlon);
        end
        plane_dlat = rad2deg(plane_states_i(:,3)) - tgtlat;
        plane_dlon(plane_points+1:end) = [];
        plane_dlat(plane_points+1:end) = [];


        % Calculate the latitudinal difference from each target
longitude
        % crossing
        [plane_dlat_lon_i,plane_phi_i] = ...
            d_lat_at_lon_func(plane_lon_i, plane_lat_i, tgtlon,
tgtlat);
        plane_dlat_lon = plane_dlat_lon_i(plane_index(i));
        plane_phi = plane_phi_i(plane_index(i));


        if k > 30
```

```
            error_message = ['Pass ' num2str(i) ' can''t be reached '
...
                ,'within 30 iterations'];
            disp(error_message);
            unable = 1;
            break
        else
            unable = [];
        end
        end


    end  %------------------ End of while loop -----------------------
---


if k == 0
    plane_gs_i = rel_gs;
    plane_states_i = rel_states;
    plane_time_i = time;
    plane_overflycount = plane_overflycount + 1;

    % Interpolate other data to find data at actual pass
    [plane_pass_states_i, plane_pass_time_i] = ...
        states_at_lonpass(plane_states_i, plane_time_i, tgtlon);


    % Resize array size for saving wanted data each loop
    plane_points = length(time);
    num1(i) = plane_points;
    if i > 1
        if plane_points < max(num1(1:i-1))
            plane_states_i(plane_points+1:max(num1(1:i-1)),:) = ...
                zeros(max(num1(1:i-1)) - plane_points,6);
            plane_time_i(plane_points+1:max(num1(1:i-1))) = ...
                zeros(max(num1(1:i-1)) - plane_points,1);
        elseif plane_points > max(num1(1:i-1))
            plane_states(max(num1(1:i-1))+1:plane_points,:,1:i-1) = ...
                zeros(plane_points - max(num1(1:i-1)),6,i-1);
            plane_time(max(num1(1:i-1))+1:plane_points,1:i-1) = ...
                zeros(plane_points - max(num1(1:i-1)),i-1);
        end
    end
    plane_states(:,:,plane_overflycount) = plane_states_i(:,:);
    plane_dv(plane_overflycount) = 2*plane_inert_initial_states(4)* ...
        sind(abs(plane_initialorbit.i - initialorbit.i)/2);
    plane_time(:,plane_overflycount) = plane_time_i;
    plane_pass_states(plane_overflycount,:) = plane_pass_states_i(i,:);
    plane_pass_time(plane_overflycount) = plane_pass_time_i(i);


    % Save all wanted data into "test" variable
    planecount = phasecount + plane_overflycount;
```

```
    test(planecount,1) = test_num;
    test(planecount,2) = run;
    test(planecount,3) = tgtlat;
    test(planecount,4) = tgtlon;
    test(planecount,5) = bank;
    test(planecount,6) = initialorbit.a;
    test(planecount,7) = initialorbit.e;
    test(planecount,8) = initialorbit.i;
    test(planecount,9) = initialorbit.RAAN;
    test(planecount,10) = initialorbit.AoP;
    test(planecount,11) = initialorbit.nu;
    test(planecount,12) = initial_states(4);
    test(planecount,13) = rad2deg(initial_states(3));
    test(planecount,14) = rad2deg(initial_states(2));
    test(planecount,15) = n_point;
    test(planecount,16) = e_point;
    test(planecount,17) = w_point;
    test(planecount,18) = s_point;
    test(planecount,19) = n_time;
    test(planecount,20) = e_time;
    test(planecount,21) = w_time;
    test(planecount,22) = s_time;
    test(planecount,23) = 2;
    test(planecount,24) = i;
    test(planecount,25) = 0;
    test(planecount,26) = plane_pass_time(plane_overflycount);
    test(planecount,27) = 0;
    test(planecount,28) = 0;
    test(planecount,29) = 0;
    test(planecount,30) = 0;
    test(planecount,31) = 0;
    test(planecount,32) = 0;
    test(planecount,33) = 0;
    test(planecount,34) = rel_initial_states(1);
    test(planecount,35) = rel_initial_states(1);
    test(planecount,36) = rel_initial_states(1);
    test(planecount,37) = rel_initial_states(1) - Rearth;
    test(planecount,38) = rel_initial_states(1) - Rearth;
    test(planecount,39) = rel_initial_states(1) - Rearth;
    test(planecount,40) = max(plane_gs_i);
    test(planecount,41) = 0;
    test(planecount,42) =
2.44*(plane_pass_states(plane_overflycount,1)-Rearth)*lambda/D;

else

if isempty(unable)

    plane_overflycount = plane_overflycount + 1;


    % Interpolate other data to find data at actual pass
    [plane_pass_states_i, plane_pass_time_i] = ...
        states_at_lonpass(plane_states_i, plane_time_i, tgtlon);
```

```
    % Resize array size for saving wanted data each loop
    num1(i) = plane_points;
    if i > 1
        if plane_points < max(num1(1:i-1))
            plane_states_i(plane_points+1:max(num1(1:i-1)),:) = ...
                zeros(max(num1(1:i-1)) - plane_points,6);
            plane_time_i(plane_points+1:max(num1(1:i-1))) = ...
                zeros(max(num1(1:i-1)) - plane_points,1);
        elseif plane_points > max(num1(1:i-1))
            plane_states(max(num1(1:i-1))+1:plane_points,:,1:i-1) = ...
                zeros(plane_points - max(num1(1:i-1)),6,i-1);
            plane_time(max(num1(1:i-1))+1:plane_points,1:i-1) = ...
                zeros(plane_points - max(num1(1:i-1)),i-1);
        end
    end
    plane_states(:,:,plane_overflycount) = plane_states_i(:,:);
    plane_dv(plane_overflycount) = 2*plane_inert_initial_states(4)* ...
        sind(abs(plane_initialorbit.i - initialorbit.i)/2);
    plane_time(:,plane_overflycount) = plane_time_i;
    plane_pass_states(plane_overflycount,:) =
plane_pass_states_i(plane_index(i),:);
    plane_pass_time(plane_overflycount) =
plane_pass_time_i(plane_index(i));


    % Save all wanted data into "test" variable
    planecount = phasecount + plane_overflycount;
    test(planecount,1) = test_num;
    test(planecount,2) = run;
    test(planecount,3) = tgtlat;
    test(planecount,4) = tgtlon;
    test(planecount,5) = 0;
    test(planecount,6) = initialorbit.a;
    test(planecount,7) = initialorbit.e;
    test(planecount,8) = initialorbit.i;
    test(planecount,9) = initialorbit.RAAN;
    test(planecount,10) = initialorbit.AoP;
    test(planecount,11) = initialorbit.nu;
    test(planecount,12) = initial_states(4);
    test(planecount,13) = rad2deg(initial_states(3));
    test(planecount,14) = rad2deg(initial_states(2));
    test(planecount,15) = n_point;
    test(planecount,16) = e_point;
    test(planecount,17) = w_point;
    test(planecount,18) = s_point;
    test(planecount,19) = n_time;
    test(planecount,20) = e_time;
    test(planecount,21) = w_time;
    test(planecount,22) = s_time;
    test(planecount,23) = 2;
    test(planecount,24) = i;
    test(planecount,25) = 0;
```

```
    test(planecount,26) = plane_pass_time(plane_overflycount);
    test(planecount,27) = plane_dv(plane_overflycount);
    test(planecount,28) = 0;
    test(planecount,29) = 0;
    test(planecount,30) = 0;
    test(planecount,31) = plane_dv(plane_overflycount);
    test(planecount,32) = 0;
    test(planecount,33) = plane_initialorbit.i - initialorbit.i;
    test(planecount,34) = min(plane_states_i(:,1));
    test(planecount,35) = max(plane_states_i(:,1));
    test(planecount,36) = plane_pass_states(plane_overflycount,1);
    test(planecount,37) = min(plane_states_i(:,1)) - Rearth;
    test(planecount,38) = max(plane_states_i(:,1)) - Rearth;
    test(planecount,39) = plane_pass_states(plane_overflycount,1) -
Rearth;
    test(planecount,40) = 0;
    test(planecount,41) = 0;
    test(planecount,42) =
2.44*(plane_pass_states(plane_overflycount,1)-Rearth)*lambda/D;


 %globe_plot2(rel_states,plane_states_i)
 %plotm([tgtlat plane_phi],[tgtlon tgtlon],'r')
% plotstates(plane_states_i,plane_time_i)
end
end
end



%% ------------------------ SKIP MANEUVER ----------------------------
---


% Initialize outer variables
options = odeset('RelTol',1e-7,'AbsTol',1e-7,'Events',@breakevent);
skip_inc = initialorbit.i;
miss_tol = 0.1;
if initialorbit.i <= 90
    orig_proretro = 1;
else
    orig_proretro = -1;
end
tsection = tsection_start;


% Find the velocity change needed to overfly each target longitudinal
% crossing
for i = 1:length(dlat_lon)


    % Initialize outer loop variables
    skip_dlat_lon = dlat_lon(i);
```

```
    skip_lat_sign = sign(phi(i));
    lat_sign = sign(phi(i));
    skip_initial_states = rel_initial_states;
    skip_v = rel_initial_states(4);
    skip_dv_i = 0.14;
    k=0;
    proretro = orig_proretro;
    skip_error = [];
    reentry = [];
    bankflag = [];
    getlower = 0;
    gethigher = 0;
    getlower_count = 0;
    stayV = 0;
    kcount = 0;
    startlate = 0;
    bank_angle = 80;
    skipstartflag = 0;


    % Determine which section of flight and which direction of bank
will
    % produce the optimum change in the overflight groundtrack
    [banksign section] = ...
        bank_section(pass_states(i,6),phi(i),dlat_lon(i),tgtlat);


    % Determine if original ground track needs (+) or (-) inclination
    % change
    if skip_dlat_lon > 0 && tgtlat > 0 && skip_inc <= 90;
        inc_sign = -1;
    elseif skip_dlat_lon < 0 && tgtlat > 0 && skip_inc <= 90;
        inc_sign = 1;
    elseif skip_dlat_lon > 0 && tgtlat < 0 && skip_inc <= 90;
        inc_sign = 1;
    elseif skip_dlat_lon < 0 && tgtlat < 0 && skip_inc <= 90;
        inc_sign = -1;
    elseif skip_dlat_lon > 0 && tgtlat > 0 && skip_inc > 90;
        inc_sign = 1;
    elseif skip_dlat_lon < 0 && tgtlat > 0 && skip_inc > 90;
        inc_sign = -1;
    elseif skip_dlat_lon > 0 && tgtlat < 0 && skip_inc > 90;
        inc_sign = -1;
    elseif skip_dlat_lon < 0 && tgtlat < 0 && skip_inc > 90;
        inc_sign = 1;
    end


    % Change velocity until minimum distance is within tolerances or
until
    % determined that overflight for particular longitudinal crossing
is
    % not possible
    while abs(skip_dlat_lon) > miss_tol
```

```
        % Initialize inner loop variables
        stop = 0;
        ii = 0;
        Rsensible = 6515;


        % If entering sensible atm puts groundtrack on opposite
latitude side
        % of the target switch the bank angle
        if getlower_count > 4 && isempty(bankflag)
            %bank_angle = -bank_angle;
            dlat_lon(i) = -dlat_lon(i);
            bankflag = 1;
        elseif getlower_count > 4
            bank_angle_change = 1;
        else
            bank_angle_change = 0;
        end


        % Algorithm to change the step size of the d-v correction
        k = k+1;
        if k > 1 && skipstartflag == 0
            skip_dv_i = skip_dv_i/2;
            skipstartflag = 1;
        end
        if stayV == 1
            zeroin(k) = zeroin(k-1);
        elseif getlower == 1
            zeroin(k) = 1;
        elseif gethigher == 1
            zeroin(k) = -1;
        elseif sign(skip_dlat_lon) == sign(dlat_lon(i)) && ...
                (skip_lat_sign == lat_sign)
            zeroin(k) = 1;
        elseif sign(skip_dlat_lon) ~= sign(dlat_lon(i)) && ...
                (skip_lat_sign == lat_sign)
            zeroin(k) = -1;
        elseif sign(skip_dlat_lon) == sign(dlat_lon(i)) && ...
                (skip_lat_sign ~= lat_sign) && (proretro ~=
orig_proretro)
            zeroin(k) = -1;
        elseif sign(skip_dlat_lon) ~= sign(dlat_lon(i)) && ...
                (skip_lat_sign ~= lat_sign) && (proretro ~=
orig_proretro)
            zeroin(k) = 1;
        elseif sign(skip_dlat_lon) == sign(dlat_lon(i)) && ...
                (skip_lat_sign ~= lat_sign)
            zeroin(k) = 1;
        elseif sign(skip_dlat_lon) ~= sign(dlat_lon(i)) && ...
                (skip_lat_sign ~= lat_sign)
            zeroin(k) = -1;
```

```
        end
        if k > 1
            if zeroin(k) == -zeroin(k-1)
                skip_dv_i = skip_dv_i/2;
            end
        end
        if k > 2
            if zeroin(k) == zeroin(k-1) && zeroin(k) == -zeroin(k-2)
                skip_dv_i = skip_dv_i/2;
            end
        end


        % Change initial velocity in order to produce desired
groundtrack
        if stayV == 1;
            skip_v = skip_v + 0;
        elseif getlower == 1
            skip_v = skip_v - skip_dv_i;
        elseif reentry == 1
            skip_v = skip_v + skip_dv_i;
            %skip_dv_i = skip_dv_i/2;
        elseif gethigher == 1;
            skip_v = skip_v + skip_dv_i;
        elseif sign(skip_dlat_lon) == sign(dlat_lon(i)) && ...
                (skip_lat_sign == lat_sign)
            skip_v = skip_v - skip_dv_i;
        elseif sign(skip_dlat_lon) ~= sign(dlat_lon(i)) && ...
                (skip_lat_sign == lat_sign)
            skip_v = skip_v + skip_dv_i;
        elseif sign(skip_dlat_lon) == sign(dlat_lon(i)) && ...
                (skip_lat_sign ~= lat_sign) && (proretro ~=
orig_proretro)
            skip_v = skip_v + skip_dv_i;
        elseif sign(skip_dlat_lon) ~= sign(dlat_lon(i)) && ...
                (skip_lat_sign ~= lat_sign) && (proretro ~=
orig_proretro)
            skip_v = skip_v - skip_dv_i;
        elseif sign(skip_dlat_lon) == sign(dlat_lon(i)) && ...
                (skip_lat_sign ~= lat_sign)
            skip_v = skip_v - skip_dv_i;
        elseif sign(skip_dlat_lon) ~= sign(dlat_lon(i)) && ...
                (skip_lat_sign ~= lat_sign)
            skip_v = skip_v + skip_dv_i;
        end
        skip_initial_states(4) = skip_v;
        skip_v;


        % Initialize variables needed to determine the needed bank
angle
        % for various phases of flight trajectory if unable to perform
at
        % the optimal time.
```

```
        bank = bank_angle;
        if inc_sign > 0 && skip_initial_states(6) > 0
            flagger1 = 1;
            flagger2 = 1;
        elseif inc_sign < 0 && skip_initial_states(6) > 0
            flagger1 = 2;
            flagger2 = 2;
        elseif inc_sign > 0 && skip_initial_states(6) < 0
            flagger1 = 3;
            flagger2 = 3;
        elseif inc_sign < 0 && skip_initial_states(6) < 0
            flagger1 = 4;
            flagger2 = 4;
        else
            flagger1 = 5;
            flagger2 = 5;
        end


        % Find the flight profile with the change in velocity occureing
at
        % time 0.
        index = 1;
        timer = [];
        bankplot = [];
        acc = [];
        [skip_time_11,skip_states_11] =
ode45(@equations_of_motion_ODE45_bank,...
                [0 simtime],skip_initial_states,options);
        skip_points = length(skip_time_11);
        skip_bankplot_11 = bankplot;
        skip_gs_11 = acc;
        skip_timer_11 = timer;


        % Find the time it takes to enter the sensible atm and restart
loop
        % if sensible atm is not reached
        if min(skip_states_11(:,1)) < Rsensible
            while stop == 0
                ii = ii + 1;
                if skip_states_11(ii,1) < Rsensible
                    stop = 1;
                    tmaneuver = skip_time_11(ii);
                end
            end


            % Determine the maneuver start time and the direction of
bank
            ii=0;
            if k == 1
                while tsection(1,1) - tmaneuver< 0
                    ii = ii+1;
```

```
                    tsection(1:end-1,:) = tsection(2:end,:);
                    tsection(end,:) = [];
                    if length(tsection) < 4;
                        tsection(4,:) = tsection_reserve(ii+4,:);
                    end
                end
                if length(tsection) > 4
                    tsection_reserve = tsection;
                end
                tsection(5:end,:) = [];
            end
            periodtime = (tsection(3,1) - tsection(2,1))*4;
            if startlate == 1
                tsection(:,1) = tsection(:,1) + periodtime;
            end
            quadtime = (tsection(2,1) - tsection(1,1))/4;
            postime1 = tsection(tsection(:,2) == section(1),1) +
quadtime;
            postime2 = tsection(tsection(:,2) == section(2),1) +
quadtime;
            if postime1 < postime2
                tstart = postime1 - tmaneuver;
                if bank_angle_change == 1
                    bank = -bank*sign(banksign(1));
                else
                    bank = bank*sign(banksign(1));
                end
            else
                tstart = postime2 - tmaneuver;
                if bank_angle_change == 1
                    bank = -bank*sign(banksign(2));
                else
                    bank = bank*sign(banksign(2));
                end
            end


            % Re-start loop if overflight time is less than the time it
            % takes to perform maneuver.
            if pass_time(i) < tmaneuver
                if abs(skip_dlat_lon) < miss_tol*3
                    error_message = ['Pass ' num2str(i) ' can be
reached ' ...
                    ,'within 3*miss toleranace or 30 miles'];
                    disp(error_message);
                    skip_error = 1;
                    break
                else
                skip_error = 1;
                break
                end
            end
```

```
            % Find state variables at maneuver start time and the
change in
            % velocity for the first phase of the trajectory
            if pass_time(i) > tstart
                stop = 0;
                ii = 0;
                while stop == 0;
                    ii = ii + 1;
                    if time(ii) > tstart
                        time_index = ii - 1;
                        stop = 1;
                    end
                end
                ii = 0;
                stop = 0;
                while stop == 0;
                    ii = ii + 1;
                    if reltimer(ii) > tstart
                        timer_index = ii - 1;
                        stop = 1;
                    end
                end
                skiptime_initial_states = rel_states(time_index,:);
                skiptime_initial_states(4) = skip_v;
                skip_dv1 = rel_states(time_index,4) - ...
                    skiptime_initial_states(4);


                % Calculate the flight profile with the bank maneuver
done
                % a the correct time
                index = 1;
                timer = [];
                bankplot = [];
                acc = [];
                [skip_time_11,skip_states_11] = ...
                    ode45(@equations_of_motion_ODE45,...
                        [0 simtime-
tstart],skiptime_initial_states,options);
                 skip_bankplot_11 = bankplot;
                 skip_gs_11 = acc;
                 skip_timer_11 = timer;


                % Concatenate the pre-skip data with the skip data
                skip_time_11 = skip_time_11 + tstart;
                skip_timer_11 = skip_timer_11 + tstart;
                skip_time_1 = vertcat(time(1:time_index-
1),skip_time_11);
                skip_states_1 = vertcat(rel_states(1:time_index-
1,:),...
                    skip_states_11);
                skip_bankplot_1 = horzcat(rel_bankplot(1:timer_index-
1),...
```

```
                    skip_bankplot_11);
                skip_gs_1 = horzcat(rel_gs(1:timer_index-
1),skip_gs_11);
                skip_timer_1 = horzcat(reltimer(1:timer_index-1),...
                    skip_timer_11);
                skip_points = length(skip_states_1);
                startime = tstart;

            else


                % If skip was done at time 0 no need to concatenate
                skip_time_1 = skip_time_11;
                skip_states_1 = skip_states_11;
                skip_bankplot_1 = skip_bankplot_11;
                skip_gs_1 = skip_gs_11;
                skip_timer_1 = skip_timer_11;
                skip_points = length(skip_states_1);
                skip_dv1 = rel_initial_states(4) - skip_v;
                startime = 0;


            end


            % Find the states at the highest and the lowest point of
orbit
            % after 1st skip
            for ii = 2:skip_points
                if pass_time(i) > tstart
                    if skip_states_1(ii,5) < 0 && ...
                            skip_states_1(ii-1,5) > 0 && ...
                            skip_time_1(ii) > (tstart + tmaneuver)
                        skip_index1 = ii-1;
                        reentry = [];
                        break
                    else
                        reentry = 1;
                        gethigher = 1;
                    end
                else
                    if skip_states_1(ii,5) < 0 && skip_states_1(ii-1,5)
> 0
                        skip_index1 = ii-1;
                        reentry = [];
                        break
                    else
                        reentry = 1;
                        gethigher = 1;
                    end
                end
            end
            if isempty(reentry)
```

```
            time_index1 = find(abs(skip_timer_1 -
skip_time_1(skip_index1))...
                == min(abs(skip_timer_1 - skip_time_1(skip_index1))));
            skip_rp = min(skip_states_1(1:skip_index1,1));
            skip_ra = skip_states_1(skip_index1,1);
            skip_rel_initial_states = skip_states_1(skip_index1,:);
            skip_dv2_1 = skip_rel_initial_states(4);


            % Propagate orbit foward from the highest point of the
orbit
            % without any bank to find the max latitude in order to
later
            % determine the correct heading angle after re-
circularization
            bank = 0;
            index = 1;
            timer = [];
            bankplot = [];
            acc = [];
            Period = 2*pi*sqrt(skip_rel_initial_states(1)^3/mu);
            [skip_time_2, skip_states_2] =
ode45(@equations_of_motion_ODE45,...
                    [0 2*Period],skip_rel_initial_states,options);
            maxlat = max(skip_states_2(:,3));


            % Find the circular velocity(relative to the rotating
planet) by
            % setting gamma dot = 0 and solving equation 3.66 for
relative
            % velocity.  For derived circular velocity iterate to find
the
            % corresponding heading angle given the max lat calculated
above



            % Initialize loop variables
            kk = 0;
            psides = skip_rel_initial_states(6);
            psi_orig = psides;
            dhead = 0.01;
            loop = 1;


            while loop == 1


                % Define variable used in Eq 3.66
                skipR        = skip_rel_initial_states(1);
                skiptheta    = skip_rel_initial_states(2);
                skipphi      = skip_rel_initial_states(3);
```

```
                skipGamma    = 0.0;
                skippsi      = skip_rel_initial_states(6);
                skipSigma    = deg2rad(bank);
                skipg        = g0 *(Rearth/skipR)^2.0;
                skiprho      = AtmosModel(skipR-Rearth,2);
                skipCl       = vehicle.Cl;
                skipS        = vehicle.S;
                skipm        = vehicle.m;


                % Solve for velocity in equation 3.66
                A        =
(1/skipR)+((skiprho*skipCl*skipS)/(2*skipm))*cos(skipSigma);
                B        = 2*wearth*cos(skipphi)*cos(skippsi);
                C        = -(skipg*cos(skipGamma))+
skipR*wearth^2*cos(skipphi)*(cos(skipphi)*cos(skipGamma)+sin(skipphi)*s
in(skippsi)*sin(skipGamma));
                Vcirc2   = (-B+(B^2-4*A*C)^0.5)/(2*A);


                % Find the corresponding Vcirc and heading angle
                psinew   = skippsi;
                done     = 0;
                cnt      = 0;
                skippsi      = psides;
                while (done == 0)
                    cnt    = cnt+1;
                    done   = 1;
                    A      =
(1/skipR)+((skiprho*skipCl*skipS)/(2*skipm))*cos(skipSigma);
                    B      = 2*wearth*cos(skipphi)*cos(skippsi);
                    C      = -(skipg*cos(skipGamma))+
skipR*wearth^2*cos(skipphi)*(cos(skipphi)*cos(skipGamma)+sin(skipphi)*s
in(skippsi)*sin(skipGamma));
                    Vcirc2 = (-B+(B^2-4*A*C)^0.5)/(2*A);

                    psinew = psides +
asin(2*pi*skipR*sin(psides)/(86400*Vcirc2));
                    if (abs(skippsi-psinew)>0.0000001)
                        done = 0;
                        skippsi  = psinew;
                    end
                end
                skip_rel_initial_states(6) = psinew;
                skip_rel_initial_states(4) = Vcirc2;
                skip_rel_initial_states(5) = 0;
                skip_dv2 = Vcirc2 - skip_dv2_1;


                % Propogate Circular Orbit forward in time for 2
periods
                index = 1;
                timer = [];
                bankplot = [];
```

```
                acc = [];
                starttime = skip_time_1(skip_index1);
                [skip_time_2,skip_states_2] =
ode45(@equations_of_motion_ODE45,[0
2*Period],skip_rel_initial_states,options);
                skip_maxlat = max(skip_states_2(:,3));


                % Compare the max latitude of new circular orbit and
compare to
                % known desired max latitude of non-circular orbit.
Break out
                % of loop once the initial heading angle gives the
correct max
                % latitude
                if skip_maxlat > maxlat - 0.0001 && skip_maxlat <
maxlat + 0.0001;
                    break;
                end


                % Algorithm to change the step size of the d-heading
correction
                kk = kk+1;
                signchange = sign(skip_rel_initial_states(6));
                if skip_maxlat > maxlat && sign(psi_orig) == signchange
                    zeroin2(kk) = 1;
                elseif skip_maxlat > maxlat && sign(psi_orig) ~=
signchange
                    zeroin2(kk) = -1;
                elseif skip_maxlat < maxlat && sign(psi_orig) ==
signchange
                    zeroin2(kk) = -1;
                elseif skip_maxlat < maxlat && sign(psi_orig) ~=
signchange
                    zeroin2(kk) = 1;
                end
                if kk > 1
                    if zeroin2(kk) == -zeroin2(kk-1)
                        dhead = dhead/2;
                    end
                end
                if kk > 2
                    if zeroin2(kk) == zeroin2(kk-1) && ...
                            zeroin2(kk) == -zeroin2(kk-2)
                        dhead = dhead/2;
                    end
                end


                % Correct Psi Desired to reach the correct maximum
latitude
                psi1 = skip_rel_initial_states(6);
                if skip_maxlat > maxlat && psi1 > 0 && psi1 <= pi/2
```

```
                    psides = psides - dhead;
                elseif skip_maxlat < maxlat && psi1 > 0 && psi1 <= pi/2
                    psides = psides + dhead;
                elseif skip_maxlat > maxlat && psi1 < 0 && psi1 <= pi/2
                    psides = psides + dhead;
                elseif skip_maxlat < maxlat && psi1 < 0 && psi1 <= pi/2
                    psides = psides - dhead;
                elseif skip_maxlat > maxlat && psi1 > 0 && psi1 > pi/2
                    psides = psides + dhead;
                elseif skip_maxlat < maxlat && psi1 > 0 && psi1 > pi/2
                    psides = psides - dhead;
                elseif skip_maxlat > maxlat && psi1 < 0 && psi1 > pi/2
                    psides = psides - dhead;
                elseif skip_maxlat < maxlat && psi1 < 0 && psi1 > pi/2
                    psides = psides + dhead;
                end

                if kk > 25
                    break
                end

            end

            % Calculate the change in inclination
            if psi1 > pi/2
                skip_inclination = 180 - rad2deg(maxlat);
            else
                skip_inclination = rad2deg(maxlat);
            end
            skip_delta_i = skip_inclination - initialorbit.i;


            % Determine if new orbit changes from prograde to
retrograde or
            % vice versa
            if abs(skip_rel_initial_states(6)) <= pi/2
                proretro = 1;
            else
                proretro = -1;
            end


            % Propogate Circular Orbit forward in time for the rest of
simtime
            index = 1;
            timer = [];
            bankplot = [];
            acc = [];
            starttime = skip_time_1(skip_index1);
            [skip_time_2,skip_states_2] =
ode45(@equations_of_motion_ODE45,...
                    [0 simtime -
starttime],skip_rel_initial_states,options);
            skip_maxlat = max(skip_states_2(:,3));
```

```
            skip_bankplot_2 = bankplot;
            skip_gs_2 = acc;
            skip_timer_2 = timer;


            % Concatenate the first preskip data with post
circularization data
            skip_time_2 = skip_time_2 +  starttime;
            skip_timer_2 = skip_timer_2 + starttime;
            skip_time_i = vertcat(skip_time_1(1:skip_index1-
1),skip_time_2);
            skip_states_i = vertcat(skip_states_1(1:skip_index1-
1,:),...
                                    skip_states_2);
            skip_bankplot_i = horzcat(skip_bankplot_1(1:time_index1-
1),...
                                      skip_bankplot_2);
            skip_timer_i = horzcat(skip_timer_1(1:time_index1-1),...
                                   skip_timer_2);
            skip_gs_i = horzcat(skip_gs_1(1:time_index1-1),skip_gs_2);
            skip_points = length(skip_time_i);


            % Correct Longitude to be from -180 to 180
            for ii = 1:skip_points
                if skip_states_i(ii,2) > pi
                    skip_states_i(ii:end,2) = -(pi - ...
                        (skip_states_i(ii:end,2) - pi));
                elseif skip_states_i(ii:end,2) < -pi
                    skip_states_i(ii:end,2) = (pi + ...
                        (skip_states_i(ii:end,2) + pi));
                end
            end
            skip_lat_i = skip_states_i(:,3);
            skip_lon_i = skip_states_i(:,2);


            % Calculate the lat/long difference and distance from the
groundtrack
            for ii = 1:skip_points
                skip_dlon(ii) =
delta_longitude(rad2deg(skip_states_i(ii,2)),tgtlon);
            end
            skip_dlat = rad2deg(skip_states_i(:,3)) - tgtlat;
            skip_dlon(skip_points+1:end) = [];
            skip_dlat(skip_points+1:end) = [];


            % Calculate the latitudinal difference from each target
longitude
            % crossing or display errors if new groundtrack doesn't get
far
            % enough
            [skip_dlat_lon_i,skip_phi_i] = ...
```

```
                d_lat_at_lon_func(skip_lon_i, skip_lat_i, tgtlon,
tgtlat);
            if length(skip_dlat_lon_i) < i
                last_distance = acos(sin(skip_lat_i(end)).*sind(tgtlat)
+ ...
                    cos(skip_lat_i(end)).*cosd(tgtlat).* ...
                    cosd(skip_dlon(end)))*Rearth;
                if tsection(4,1) > simtime
                    if skip_dv_i < 0.001 && last_distance > 10
                        error_message = ['Pass ' num2str(i) ' cannot be
' ...
                            ,'reached via a skip maneuver in allotted
time'];
                        disp(error_message);
                        unable = 1;
                        break
                    else
                        skip_dlat_lon = dlat_lon(i);
                        skip_phi = phi(i);
                        gethigher = 1;
                    end
                else
                    startlate = 1;
                    stayV = 1;
                    kcount = kcount + 1;
                    skip_dlat_lon = dlat_lon(i);
                    skip_phi = phi(i);
                end
            elseif k > 21 && abs(skip_dlat_lon) < miss_tol*3
                error_message = ['Pass ' num2str(i) ' can be reached '
...
                    ,'within 3*miss toleranace or 30 miles'];
                disp(error_message);
                break
            elseif k > 21 + kcount
                error_message = ['Pass ' num2str(i) ' cannot be ' ...
                    ,'reached with k < 20'];
                disp(error_message);
                unable = 1;
                break
            elseif max(skip_gs_i) > 15 && skip_dlat_lon > 1
                error_message = ['Pass' num2str(i) ' cannot be ' ...
                    ,'reached within g-limits'];
                disp(error_message)
                unable = 1;
                break
            else
                startlate = 0;
                stayV = 0;
                if orig_proretro == 1
                    if skip_rel_initial_states(2) > deg2rad(tgtlon)
                        if (skip_rel_initial_states(2) -
deg2rad(tgtlon)) < deg2rad(22.5)
                            dangerzone = 1;
                        else
```

```
                            dangerzone = 0;
                        end
                    else
                        dangerzone = 0;
                    end
                else
                    if skip_rel_initial_states(2) < deg2rad(tgtlon)
                        if deg2rad(tgtlon) - skip_rel_initial_states(2)
< deg2rad(22.5)
                            dangerzone = 1;
                        else
                            dangerzone = 0;
                        end
                    else
                        dangerzone = 0;
                    end
                end
                if proretro ~= orig_proretro && dangerzone == 1 &&
abs(skip_rel_initial_states(6)) > deg2rad(91)
                    skip_dlat_lon = skip_dlat_lon_i(i+1);
                    skip_phi = skip_phi_i(i+1);
                else
                    skip_dlat_lon = skip_dlat_lon_i(i);
                    skip_phi = skip_phi_i(i);
                end
                skip_lat_sign = sign(skip_phi);
                unable = [];
                gethigher = 0;
            end

              %globe_plot2(rel_states,skip_states_i)
              %plotm([tgtlat skip_phi],[tgtlon tgtlon],'r')
%              figure
%              plot(skip_timer_i,skip_bankplot_i)

            end         % End if skip after re-entry section


        % End of if velocity change doesn't get vehicle into sensible
        % atmosphere
            getlower = 0;
        else
            getlower = 1;
            getlower_count = getlower_count + 1;
        end



    end %------------------ End of while loop ------------------------
-%

if k == 0
    skip_gs_i = rel_gs;
    skip_states_i = rel_states;
```

```
    skip_time_i = time;
    skip_overflycount = skip_overflycount + 1;

    % Interpolate other data to find data at actual pass
    [skip_pass_states_i, skip_pass_time_i] = ...
        states_at_lonpass(skip_states_i, skip_time_i, tgtlon);


    % Resize array size for saving wanted data each loop
        skip_points = length(time);
        num1(i) = skip_points;
        if i > 1
            if skip_points < max(num1(1:i-1))
                skip_states_i(skip_points+1:max(num1(1:i-1)),:) = ...
                    zeros(max(num1(1:i-1)) - skip_points,6);
                skip_time_i(skip_points+1:max(num1(1:i-1))) = ...
                    zeros(max(num1(1:i-1)) - skip_points,1);
            elseif skip_points > max(num1(1:i-1))
                skip_states(max(num1(1:i-1))+1:skip_points,:,1:i-1) =
...
                    zeros(skip_points - max(num1(1:i-1)),6,i-1);
                skip_time(max(num1(1:i-1))+1:skip_points,1:i-1) = ...
                    zeros(skip_points - max(num1(1:i-1)),i-1);
            end
        end
        skip_states(:,:,skip_overflycount) = skip_states_i(:,:);
        skip_dv(skip_overflycount) = 0;
        transfer_dv(skip_overflycount) = 0;
        skip_time(:,skip_overflycount) = skip_time_i;
        skip_pass_states(skip_overflycount,:) =
skip_pass_states_i(i,:);
        skip_pass_time(skip_overflycount) = skip_pass_time_i(i);


    % Save all wanted data into "test" variable
    skipcount = planecount + skip_overflycount;
    test(skipcount,1) = test_num;
    test(skipcount,2) = run;
    test(skipcount,3) = tgtlat;
    test(skipcount,4) = tgtlon;
    test(skipcount,5) = bank;
    test(skipcount,6) = initialorbit.a;
    test(skipcount,7) = initialorbit.e;
    test(skipcount,8) = initialorbit.i;
    test(skipcount,9) = initialorbit.RAAN;
    test(skipcount,10) = initialorbit.AoP;
    test(skipcount,11) = initialorbit.nu;
    test(skipcount,12) = initial_states(4);
    test(skipcount,13) = rad2deg(initial_states(3));
    test(skipcount,14) = rad2deg(initial_states(2));
    test(skipcount,15) = n_point;
    test(skipcount,16) = e_point;
    test(skipcount,17) = w_point;
    test(skipcount,18) = s_point;
```

```
    test(skipcount,19) = n_time;
    test(skipcount,20) = e_time;
    test(skipcount,21) = w_time;
    test(skipcount,22) = s_time;
    test(skipcount,23) = 3;
    test(skipcount,24) = i;
    test(skipcount,25) = 0;
    test(skipcount,26) = skip_pass_time(skip_overflycount);
    test(skipcount,27) = 0;
    test(skipcount,28) = 0;
    test(skipcount,29) = 0;
    test(skipcount,30) = 0;
    test(skipcount,31) = 0;
    test(skipcount,32) = 0;
    test(skipcount,33) = 0;
    test(skipcount,34) = rel_initial_states(1);
    test(skipcount,35) = rel_initial_states(1);
    test(skipcount,36) = rel_initial_states(1);
    test(skipcount,37) = rel_initial_states(1) - Rearth;
    test(skipcount,38) = rel_initial_states(1) - Rearth;
    test(skipcount,39) = rel_initial_states(1) - Rearth;
    test(skipcount,40) = max(skip_gs_i);
    test(skipcount,41) = 0;
    test(skipcount,42) = 2.44*(skip_pass_states(skip_overflycount,1)-
Rearth)*lambda/D;

else

if isempty(skip_error)


    % Dipslay error if calculate overflight breaks g-limitation
    if max(skip_gs_i) > 10
        error_message = ['WARNING! Pass ' num2str(i) ' cannot be ' ...
            ,'reached within g-limits'];
        disp(error_message);
    end


    if isempty(unable)  % Only perform if successful overflight
achieved


        skip_overflycount = skip_overflycount + 1;

        % Find the velocity needed to re-circularize at the original
orbit
        r0 = initial_states(1);
        skip_transfer_dv_guess = sqrt(mu/skip_ra - mu/(skip_ra + r0))
...
            - sqrt(mu/skip_ra - mu/(skip_ra+skip_rp));
        skip_ra2 = skip_ra;
        skip_r_tol = 0.1;
```

```
        jj = 0;
        transfer_initial_states = skip_rel_initial_states;
        transfer_initial_states(4) = skip_rel_initial_states(4) + ...
                    skip_transfer_dv_guess;
        skip_transfer_dv = 0.03;
        while skip_ra2 > r0 + skip_r_tol || skip_ra2 < r0 - skip_r_tol


            index = 1;
            timer = [];
            bankplot = [];
            acc = [];
            [skip_time_t,skip_states_t] =
ode45(@equations_of_motion_ODE45,...
                    [0 simtime -
starttime],transfer_initial_states,options);
            skip_ra2 = max(skip_states_t(:,1));
                    % Algorithm to change the step size of the dv
correction
            jj = jj+1;
            if skip_ra2 > r0
                zeroin(jj) = 1;
            else
                zeroin(jj) = -1;
            end
            if jj > 1
                if zeroin(jj) == -zeroin(jj-1)
                    skip_transfer_dv = skip_transfer_dv/2;
                end
            end
            if jj > 2
                if zeroin(jj) == zeroin(jj-1) && zeroin(jj) == -
zeroin(jj-2)
                    skip_transfer_dv = skip_transfer_dv/2;
                end
            end


            if skip_ra2 < r0
                transfer_initial_states(4) = transfer_initial_states(4)
+ ...
                    skip_transfer_dv;
            else
                transfer_initial_states(4) = transfer_initial_states(4)
- ...
                    skip_transfer_dv;
            end
        end
        transfer_states_index = find(skip_states_t == skip_ra2);
        transfer_states = skip_states_t(transfer_states_index,:);


         % Define variable used in Eq 3.66
        transfer_R        = transfer_states(1);
```

```
        transfer_theta    = transfer_states(2);
        transfer_phi      = transfer_states(3);
        transfer_Gamma    = 0.0;
        transfer_psi      = transfer_states(6);
        transfer_Sigma    = deg2rad(bank);
        transfer_g        = g0 *(Rearth/transfer_R)^2.0;
        transfer_rho      = AtmosModel(transfer_R-Rearth,2);
        transfer_Cl       = vehicle.Cl;
        transfer_S        = vehicle.S;
        transfer_m        = vehicle.m;


        % Solve for velocity in equation 3.66
        A        =
(1/transfer_R)+((transfer_rho*transfer_Cl*transfer_S)/(2*transfer_m))*c
os(transfer_Sigma);
        B        = 2*wearth*cos(transfer_phi)*cos(transfer_psi);
        C        = -(transfer_g*cos(transfer_Gamma))+
transfer_R*wearth^2*cos(transfer_phi)*(cos(transfer_phi)*cos(transfer_G
amma)+sin(transfer_phi)*sin(transfer_psi)*sin(transfer_Gamma));
        transfer_Vcirc   = (-B+(B^2-4*A*C)^0.5)/(2*A);


        % Interpolate other data to find data at actual pass
        [skip_pass_states_i, skip_pass_time_i] = ...
            states_at_lonpass(skip_states_i, skip_time_i, tgtlon);


        % Resize array size for saving wanted data each loop
        num1(i) = skip_points;
        if i > 1
            if skip_points < max(num1(1:i-1))
                skip_states_i(skip_points+1:max(num1(1:i-1)),:) = ...
                    zeros(max(num1(1:i-1)) - skip_points,6);
                skip_time_i(skip_points+1:max(num1(1:i-1))) = ...
                    zeros(max(num1(1:i-1)) - skip_points,1);
            elseif skip_points > max(num1(1:i-1))
                skip_states(max(num1(1:i-1))+1:skip_points,:,1:i-1) =
...
                    zeros(skip_points - max(num1(1:i-1)),6,i-1);
                skip_time(max(num1(1:i-1))+1:skip_points,1:i-1) = ...
                    zeros(skip_points - max(num1(1:i-1)),i-1);
            end
        end
        skip_states(:,:,skip_overflycount) = skip_states_i(:,:);
        skip_dv(skip_overflycount) = skip_dv1 + skip_dv2;
        transfer_dv(skip_overflycount) = (transfer_Vcirc - ...
            transfer_states(4)) + (transfer_initial_states(4) - ...
            skip_dv2_1) + skip_dv1;
        skip_time(:,skip_overflycount) = skip_time_i;
        skip_pass_states(skip_overflycount,:) =
skip_pass_states_i(i,:);
        skip_pass_time(skip_overflycount) = skip_pass_time_i(i);
```

```
        % Save all wanted data into "test" variable
        skipcount = planecount + skip_overflycount;
        test(skipcount,1) = test_num;
        test(skipcount,2) = run;
        test(skipcount,3) = tgtlat;
        test(skipcount,4) = tgtlon;
        test(skipcount,5) = bank;
        test(skipcount,6) = initialorbit.a;
        test(skipcount,7) = initialorbit.e;
        test(skipcount,8) = initialorbit.i;
        test(skipcount,9) = initialorbit.RAAN;
        test(skipcount,10) = initialorbit.AoP;
        test(skipcount,11) = initialorbit.nu;
        test(skipcount,12) = initial_states(4);
        test(skipcount,13) = rad2deg(initial_states(3));
        test(skipcount,14) = rad2deg(initial_states(2));
        test(skipcount,15) = n_point;
        test(skipcount,16) = e_point;
        test(skipcount,17) = w_point;
        test(skipcount,18) = s_point;
        test(skipcount,19) = n_time;
        test(skipcount,20) = e_time;
        test(skipcount,21) = w_time;
        test(skipcount,22) = s_time;
        test(skipcount,23) = 3;
        test(skipcount,24) = i;
        test(skipcount,25) = startime;
        test(skipcount,26) = skip_pass_time(skip_overflycount);
        test(skipcount,27) = skip_dv1;
        test(skipcount,28) = skip_dv2;
        test(skipcount,29) = transfer_initial_states(4) - skip_dv2_1;
        test(skipcount,30) = transfer_Vcirc - transfer_states(4);
        test(skipcount,31) = skip_dv(skip_overflycount);
        test(skipcount,32) = transfer_dv(skip_overflycount);
        test(skipcount,33) = skip_delta_i;
        test(skipcount,34) = skip_rp;
        test(skipcount,35) = skip_ra;
        test(skipcount,36) = skip_pass_states(skip_overflycount,1);
        test(skipcount,37) = skip_rp - Rearth;
        test(skipcount,38) = skip_ra - Rearth;
        test(skipcount,39) = skip_pass_states(skip_overflycount,1) -
Rearth;
        test(skipcount,40) = max(skip_gs_i);
        test(skipcount,41) = 0;
        test(skipcount,42) =
2.44*(skip_pass_states(skip_overflycount,1)-Rearth)*lambda/D;

        %globe_plot2(rel_states,skip_states_i)
        %figure
        %plot(skip_timer_i,skip_gs_i)
    end
end
end
```

```
end

%%                          Display Results



if abs(tgtlat) <= initialorbit.i

TOA_vs_dV_all(phase_pass_time,phase_dv,plane_pass_time,plane_dv,skip_pa
ss_time,skip_dv,skip_pass_time,transfer_dv,bank_angle)
else

TOA_vs_dV_2(plane_pass_time,plane_dv,skip_pass_time,skip_dv,skip_pass_t
ime,transfer_dv,bank_angle)
end
toc



testnum = strcat('Test_',num2str(test_num));
runnum = strcat('Run_',num2str(run));
%if isempty(directory)
    folder1 = ['C:\Users\user\Documents\MATLAB\Thesis\Thesis
Figures\',testnum];
    mkdir('C:\Users\user\Documents\MATLAB\Thesis\Thesis
Figures',testnum)
    mkdir(folder1,'TOAvsDV')
    mkdir(folder1,'Original_GT')
    Toa_folder = [folder1,'\TOAvsDV'];
    org_folder = [folder1,'\Original_GT'];
    directory = 1;
%end
fig1 = (run)*2-1;
fig2 = (run)*2;
TOAfilename = [Toa_folder ,'\' runnum, '_TOAvsDV.jpg'];
orgfilename = [org_folder ,'\' runnum, '_Orig_GT.jpg'];
saveas(fig1,orgfilename)
saveas(fig2,TOAfilename)

end

totalcount = skipcount;

end



% Save the test data in both excel and matlab formats
testnum = strcat('Test_',num2str(test_num));
testheader = {'Test #', 'Run #', 'Target Lat', 'Target Long', 'Bank
Angle (deg)', ...
    'a', 'e', 'i', 'RAAN', 'AoP', 'True Anamoly', 'Initial V', ...
    'Initial Latitude', 'Initial Longitude', 'N dist', 'E dist', 'W
dist', ...
```

```
    'S dist', 'N time', 'E time', 'W time', 'S Time', 'type', 'Pass #',
...
    'Start Time', 'ToA', 'dv1', 'dv2', 'dv3', 'dv4', 'dvtotal 1',
'dvtotal 2', ...
    'Inc Change', 'Perigee R', 'Apogee R', 'Overflight R', 'Perigee
Alt', ...
    'Apogee Alt', 'Overglight Alt', 'Max G''s', 'Heat Flux', 'Ground
Res'};
testdata = strcat('\', testnum,'data','.xls');
testdata2 = strcat('\', testnum,'data','.mat');
folder2 = [folder1, testdata];
folder3 = [folder1, testdata2];
xlswrite(folder2,testheader)
xlswrite(folder2,test,1,'A2')
save(folder3,'test')
```

Optimal_Overfly_Inputs.m

```
 function Optimal_Overfly_Inputs

global tgtlat tgtlon initialorbit startdate simtime vehicle

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

% Target Location

%     tgtlat = 35.696216;
%     tgtlon = 51.422945;
tgtlat = 42;    % Target Latitude (deg), N positive
tgtlon = -90;    % Target Longitude (deg), E positive

% Initial Orbit Classical Elements
initialorbit.a = 6878;      % Semi-Major Axis (km)
initialorbit.e = 0;       % Eccentricity (0-1)
initialorbit.i = 45;        % Inclination (deg)
initialorbit.RAAN = -24;   % Right Ascension of the Ascending Node
(deg)
initialorbit.AoP = 0;       % Argument of Perigee (deg)
initialorbit.nu = 0;        % True Anamoly (deg)

% Simulation Start Date and Time
startdate.year = 2017;      % Simulation start year
startdate.month = 4;        % Simulation start month (1-12)
startdate.day = 11;         % Simulation start day (1-31)
startdate.hour = 16;        % Simulation start hour (1-23)
startdate.min = 0;          % Simulation start minute (1-59)
startdate.sec = 0;          % Simulation start second (1-59)

% Simulation Run Time
sim_time.day = 0;           % # of whole days
sim_time.hour = 24;          % # of whole hours (1-23)
```

```
sim_time.min = 0;           % # of whole minutes (1-59)
sim_time.sec = 0;           % # of whole seconds (1-59)

% Vehicle Parameters
vehicle.m = 5000;           % Vehicle mass (kg)
vehicle.S = 1e-5;           % Vehicle planform area (km)
vehicle.Cl = 3;             % Vehicle max lift coefficient
vehicle.Cd = 0.5;             % Vehicle drag coefficient

simtime = sim_time.day*86400 +  sim_time.hour * 3600 + sim_time.min*60
...
    +sim_time.sec;
```

delta_longitude.m

```
function [ dlon ] = delta_longitude( lon1, lon2 )
% This function finds the difference between two longitudinal points
% taking into account the international date line.

% Inputs: lon1 -- longitude of point 1 (degrees)
%         lon2 -- longitude of point 1 degrees)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Outputs: d_lon -- the longitudinal difference, lon1 east of lon2 is
%                   defined as positive (degrees)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%



if lon1 > 0 && lon2 < 0
    dlon = -360 + (lon1 - lon2);
elseif lon1 < 0 && lon2 > 0
    dlon = 360 + (lon1 - lon2);
else
    dlon = lon1 - lon2;
end
if dlon < -180
    dlon = dlon + 360;
elseif dlon > 180
    dlon = dlon - 360;
end



end
```

d_lat_at_lon.m

```
function [ d_lat, phi ] = d_lat_at_lon_func( lon, lat, tgtlon, tgtlat )
% This function finds the latitudinal difference of each target
longitude
% pass at the point where the profile overflight path is the same as
the
% target longitude

% Inputs: lon (profile longitude in radians)
%         lat (profile latitude in radians)
%         tgtlat (target latitude in degrees)
%         tgtlon (target longitude in degrees)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Outputs: d_lat (the latitudinal difference in degrees of each
longitude
%                 passing.)
%          phi (the latitude of the current pass at the target
longitude
%                 used for diagnostic purposes)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%



lat = rad2deg(lat);
lon = rad2deg(lon);


% Find longitude difference of each data point
for i = 1:length(lon)
    dlon(i) = delta_longitude(lon(i),tgtlon);
end


% Find all target longitude crossing data points
loncount = 0;
for i = 2:length(dlon)
    if dlon(i-1) < 0 && dlon(i) > 0 && abs(dlon(i)) < 90;
        loncount = loncount + 1;
        lonpass_index(loncount) = i;
    elseif dlon(i-1) > 0 && dlon(i) < 0 && abs(dlon(i)) < 90;
        loncount = loncount + 1;
        lonpass_index(loncount) = i;
    end
end


% Use a linear interpolation to find the latitudinal difference at each
```

```
% longitude pass
for i = 1:loncount
    m = (lon(lonpass_index(i)) - lon(lonpass_index(i) - 1))/ ...
        (lat(lonpass_index(i)) - lat(lonpass_index(i) - 1));
    phi(i) = lat(lonpass_index(i)) + (tgtlon -
lon(lonpass_index(i)))/m;
    d_lat(i) = phi(i) - tgtlat;
end


end
```

d_lon_at_lat.m

```
function [ d_lon, theta ] = d_lon_at_lat_func( lon, lat, tgtlon, tgtlat
)
% This function finds the longitudinal difference of each target
latitude
% pass at the point where the profile overflight path is the same as
the
% target latitude

% Inputs: lon (profile longitude in radians)
%         lat (profile latitude in radians)
%         tgtlat (target latitude in degrees)
%         tgtlon (target longitude in degrees)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Outputs: d_lon (the longitudinal difference in degrees of each
latitude
%                 passing.)
%          theta (the longitude of the current pass at the target
latitude
%                 used for diagnostic purposes)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%


lat = rad2deg(lat);
lon = rad2deg(lon);


% Find latitude difference of each data point
dlat = lat - tgtlat;


% Find all target latitude crossing data points
```

```
latcount = 0;
for i = 2:length(dlat)
    if dlat(i-1) < 0 && dlat(i) > 0
        latcount = latcount + 1;
        latpass_index(latcount) = i;
    elseif dlat(i-1) > 0 && dlat(i) < 0
        latcount = latcount + 1;
        latpass_index(latcount) = i;
    end
end


% Use a linear interpolation to find the longitudinal difference at
each
% latitude pass
for i = 1:latcount
    m = (lat(latpass_index(i)) - lat(latpass_index(i) - 1))/ ...
        (lon(latpass_index(i)) - lon(latpass_index(i) - 1));
    theta(i) = lon(latpass_index(i)) + (tgtlat -
lat(latpass_index(i)))/m;
    if theta(i) > 0 && tgtlon < 0
        d_lon(i) = -360 + (theta(i) - tgtlon);
    elseif theta(i) < 0 && tgtlon > 0
        d_lon(i) = 360 + (theta(i) - tgtlon);
    else
        d_lon(i) = theta(i) - tgtlon;
    end
    if d_lon(i) < -180
        d_lon(i) = d_lon(i) + 360;
    elseif d_lon(i) > 180
        d_lon(i) = d_lon(i) - 360;
    end
end


end
```

equations_of_motionODE45.m

```
function [ rel_states ] = equations_of_motion_ODE45( t,F0 )
% Equtions of Motion for use with ODE 45.  Using a initial state, (F0),
and
% specified time, t ODE 45 propogates the states forward for the
specified
% time using the the rotating earth equations of motion found in Hick's
% Eqs 12.1- 12.3 and 12.16 - 12.18
%
% Inputs: F0(1-6):
%                   1. R (km)
%                   2. Longitude (rads), E positive
%                   3. Latitude (rads), N positive
```

```
%                   4. Velocity (km/s)
%                   5. Flight Path Angle (rads)
%                   6. Heading Angle (rads)
%          t: time range (sec)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Outputs: rel_states(1-6)
%                   1. R (km)
%                   2. Longitude (rads), E positive
%                   3. Latitude (rads), N positive
%                   4. Velocity (km/s)
%                   5. Flight Path Angle (rads)
%                   6. Heading Angle (rads)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

% Assumptions:
%   - 2 Body Dynamics
%   - Lift perpendicular to velocity
%   - Drag parallel to velocity
%   - Mass is constant
%   - CL/CD is constant

global g0 rho0 Beta Rearth wearth vehicle bank inc_sign flagger1
flagger2
global bank_ode index bankplot timer switchtime acc

timer(index) = t;
m = vehicle.m;              % mass (kg)
S = vehicle.S;              % planform area (km^2)
Cl = vehicle.Cl;            % lift coefficient
Cd = vehicle.Cd;            % drag coefficient

% Re-write Initial States for ease of use in equations
R = F0(1);
lon = F0(2);
lat = F0(3);
V = F0(4);
fpa = F0(5);
heading = F0(6);


% Calculate conditions at current altitude location
h = R - Rearth;
rho = AtmosModel(h,2);
D = rho*Cd*S*V^2/2;
L = rho*Cl*S*V^2/2;
g = g0*(Rearth/R)^2;
time = t;
bank_ode = deg2rad(bank);
```

```
% Select Bank angle
% if flagger1 == flagger2
%     if lat > deg2rad(25)
%         bank_ode = -deg2rad(bank);
%         flagger2 = 1;
%         switchtime = t;
%     elseif lat <= deg2rad(25)
%         bank_ode = deg2rad(bank);
%         flagger2 = 2;
%         switchtime = t;
%     elseif inc_sign > 0 && heading < 0
%         bank_ode = -deg2rad(bank);
%         flagger2 = 3;
%         switchtime = t;
%     elseif inc_sign < 0 && heading < 0
%         bank_ode = deg2rad(bank);
%         flagger2 = 4;
%         switchtime = t;
%     end
% elseif isempty(flagger1)
%     bank_ode = 0;
% elseif timer(index) > switchtime + 200
%     flagger1 = flagger2;
% end
% if heading < .15 && heading > -0.15
%     bank_ode = 0;%-deg2rad(bank);
% end
bankplot(index) = bank_ode;


% Find g-loading
av = D/m + g*sin(fpa);                    % velocity acc
al = -L/m - (V^2/R - g)*cos(fpa);         % lift acc
acc(index) = sqrt(av^2 + al^2)/g;         % total decelration


% Write Equations of Motion (Eqs 12.1 - 12.6)
dR = V*sin(fpa);
dlon = V*cos(fpa)*cos(heading)/(R*cos(lat));
dlat = V*cos(fpa)*sin(heading)/R;
dV = -D/m - g*sin(fpa) + R*wearth^2*cos(lat)*(cos(lat)*sin(fpa)...
    - sin(lat)*sin(heading)*cos(fpa));
dfpa = (1/V)*(L/m*cos(bank_ode) - g*cos(fpa) + V^2/R*cos(fpa) +  ...
    2*V*wearth*cos(lat)*cos(heading) + R*wearth^2*cos(lat)*(cos(lat)...
    *cos(fpa) + sin(lat)*sin(heading)*sin(fpa)));
dheading = (1/V)*(L*sin(bank_ode)/(m*cos(fpa)) -
V^2/R*cos(fpa)*cos(heading)...
    *tan(lat) + 2*V*wearth*(sin(heading)*cos(lat)*tan(fpa) - sin(lat))
...
    - R*wearth^2/cos(fpa)*sin(lat)*cos(lat)*cos(heading));


index = index + 1;
```

```
rel_states=[dR;dlon;dlat;dV;dfpa;dheading];
return
```

AtmosModel.m

```
function [Rho] = AtmosModel(h_gd,AtmosModel_Choice)

global Rearth Beta rho0

WGS84Constants; %Loads global constants from external m-file

%Note: AtmosModel_Choice
%1 = Exponential density model
%2 = Combined density model (similar to MSIS model)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%% Exponential Density Model (kg/km^3)
if     AtmosModel_Choice == 1
    Rho = rho0.*exp(-Beta.*h_gd);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%% Combined Density Model    (kg/km^3)
%Note: Exponential Model:                 h < 84 km
%      Scale Height (v1) Variation Model: 84  <= h <= 120  km
%      Power Model:                       121 <= h <= 1000 km

elseif AtmosModel_Choice == 2
%Reference altitude (km)
h_i        = [67; 85; 99; 110];

%Reference density (kg/km^3)
Rho_i      = [1.4975e-4; 7.726e-6; 4.504e-7; 5.930e-8] * (1000)^3;

%Reference scale height (km)
Hi         = [6.6597; 4.979; 5.905; 8.731];

%Reference molecular scale temperature (K)
TMi        = [222.8; 165.7; 195.6; 288.2];

%Atmospheric constant (K/km)
Constant_A = [0.1296385; 0.1545455; 0.1189286; 0.5925240];

%Atmospheric constant (K/km)
Constant_B = [4.044231; 0.0; 3.878571; 19.17964];

%Dimensionless parameters
deltaH     = (Constant_A.*Rearth)./Hi;
```

```
deltaTM    = (Constant_B.*Rearth)./TMi;

%Altitude Sections
if     h_gd <= 84                    %Section 1: Exponential model
    Rho = rho0.*exp(-Beta.*h_gd);

elseif h_gd >  84 && h_gd <= 90   %Section 2: Single Variation
    Rho = Rho_i(2).*((1./(1 + deltaH(2).*((h_gd - h_i(2))./Rearth))).^
...
        ((1 + Constant_A(2))./Constant_A(2)));

elseif h_gd >  90  && h_gd <= 106  %Section 3: Single Variation
    Rho = Rho_i(3).*((1./(1 + deltaH(3).*((h_gd - h_i(3))./Rearth))).^
...
        ((1 + Constant_A(3))./Constant_A(3)));

elseif h_gd >  106 && h_gd <= 120  %Section 4: Single Variation
    Rho = Rho_i(4).*((1./(1 + deltaH(4).*((h_gd - h_i(4))./Rearth))).^
...
        ((1 + Constant_A(4))./Constant_A(4)));

elseif h_gd >  120 && h_gd <= 1000 %Section 5: Power Model
    Rho = ((4.50847623E7).*((h_gd).^(-7.44605852))).*((1000)^3);

    %Note: 'Power Model' formulated with altitude in units of (km) and
    %      the output density in (kg/m^3)

else %if h_gd > 1000;
    Rho = 0;
end

end
```

bank_section.m

```
function [ banksign, section ] = bank_section(psi, lat, dlat, tgtlat)
% This function determines the optimal portion of an orbit to perform a
% bank maneuever in order to shift the orignal ground track in the
dedired
% location


% Inputs: psi -- heading angle of the current ground track at the same
%                longitude as the target
%         lat -- latitude of the current ground track at the same
longitude
%                as the target
%        dlat -- latitude difference of the ground track to the target
%                 (positive is defined as ground track north of target)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
```

```
% Outputs: banksign -- (+) for right bank, (-) for left bank
%          section  -- the optimal ground track section to perfrom bank
%                      maneuver
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%


dlat = -dlat;
if psi <= 90
    if sign(lat) == sign(tgtlat)
        if psi > 0 && lat > 0 && dlat > 0
            section = [2 4];
        elseif psi > 0 && lat > 0 && dlat < 0
            section = [4 2];
        elseif psi < 0 && lat > 0 && dlat > 0
            section = [3 1];
        elseif psi < 0 && lat > 0 && dlat < 0
            section = [1 3];
        elseif psi < 0 && lat < 0 && dlat > 0
            section = [4 2];
        elseif psi < 0 && lat < 0 && dlat < 0
            section = [2 4];
        elseif psi > 0 && lat < 0 && dlat > 0
            section = [1 3];
        elseif psi > 0 && lat < 0 && dlat < 0
            section = [3 1];
        end
    elseif sign(lat) ~= sign(tgtlat)
        if psi > 0 && lat > 0
            section = [3 1];
        elseif psi < 0 && lat > 0
            section = [2 4];
        elseif psi < 0 && lat < 0
            section = [3 1];
        elseif psi > 0 && lat < 0
            section = [2 4];
        end
    end
elseif psi > 90
    if sign(lat) == sign(tgtlat)
        if psi > 0 && lat > 0 && dlat > 0
            section = [3 1];
        elseif psi > 0 && lat > 0 && dlat < 0
            section = [1 3];
        elseif psi < 0 && lat > 0 && dlat > 0
            section = [2 4];
        elseif psi < 0 && lat > 0 && dlat < 0
            section = [4 2];
        elseif psi < 0 && lat < 0 && dlat > 0
            section = [1 3];
        elseif psi < 0 && lat < 0 && dlat < 0
            section = [3 1];
        elseif psi > 0 && lat < 0 && dlat > 0
```

```
            section = [4 2];
        elseif psi > 0 && lat < 0 && dlat < 0
            section = [2 4];
        end
    elseif sign(lat ~= sign(tgtlat))
        if psi > 0 && lat > 0
            section = [2 4];
        elseif psi < 0 && lat > 0
            section = [3 1];
        elseif psi < 0 && lat < 0
            section = [2 4];
        elseif psi > 0 && lat < 0
            section = [3 1];
        end
    end
end


banksign = [-1 1];
```

states_at_lonpass.m

```
function [ pass_states, pass_time ] = states_at_lonpass( states, time,
tgtlon )
% This function uses a linear interpolation to find the values of input
% states data at the target longitude location

% Inputs: states(:,1-6):
%                   1. R (km)
%                   2. longitude (rads), E positive
%                   3. latitude (rads), N positive
%                   4. Velocity (km/s)
%                   5. Flight Path Angle (rads)
%                   6. Heading Angle (rads)
%         time -- time corresponding with state values (seconds)
%         tgtlat -- Target latitude (degrees)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Outputs: pass_states(1-6)
%                   1. R (km)
%                   2. longitude (rads), E positive
%                   3. latitude (rads), N positive
%                   4. Velocity (km/s)
%                   5. Flight Path Angle (rads)
%                   6. Heading Angle (rads)
%           pass_time -- time of overflight (seconds)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
```

```
lon = rad2deg(states(:,2));


% Find latitude difference of each data point
for i = 1:length(lon)
    dlon(i) = delta_longitude(lon(i),tgtlon);
end
dlon = deg2rad(dlon);


% Find all target longitude crossing data points
loncount = 0;
for i = 2:length(dlon)
    if dlon(i-1) < 0 && dlon(i) > 0 && abs(dlon(i)) < pi/2;
        loncount = loncount + 1;
        lonpass_index(loncount) = i;
    elseif dlon(i-1) > 0 && dlon(i) < 0 && abs(dlon(i)) < pi/2;
        loncount = loncount + 1;
        lonpass_index(loncount) = i;
    end
end


% Use a linear interpolation to find the latitudinal difference at each
% longitude pass
for i = 1:loncount
    m = (lon(lonpass_index(i)) - lon(lonpass_index(i) - 1))./ ...
        (states(lonpass_index(i),:) - states(lonpass_index(i) - 1,:));
    pass_states(i,:) = states(lonpass_index(i),:) + (tgtlon -
lon(lonpass_index(i)))./m;
    m2 = (lon(lonpass_index(i)) - lon(lonpass_index(i) - 1))/ ...
        (time(lonpass_index(i)) - time(lonpass_index(i) - 1));
    pass_time(i) = time(lonpass_index(i)) + (tgtlon -
lon(lonpass_index(i)))/m2;
end


end
```

states_at_latpass.m

```
function [ pass_states, pass_time ] = states_at_pass( states, time,
tgtlat )
% This function uses a linear interpolation to find the values of input
% states data at the target latitude location

% Inputs: states(:,1-6):
%                   1. R (km)
%                   2. longitude (rads), E positive
%                   3. latitude (rads), N positive
```

```
%                   4. Velocity (km/s)
%                   5. Flight Path Angle (rads)
%                   6. Heading Angle (rads)
%          tgtlat -- Target latitude (degrees)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Outputs: pass_states(1-6)
%                   1. R (km)
%                   2. longitude (rads), E positive
%                   3. latitude (rads), N positive
%                   4. Velocity (km/s)
%                   5. Flight Path Angle (rads)
%                   6. Heading Angle (rads)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

lat = rad2deg(states(:,3));


% Find latitude difference of each data point
dlat = lat - tgtlat;


% Find all target latitude crossing data points
latcount = 0;
for i = 2:length(dlat)
    if dlat(i-1) < 0 && dlat(i) > 0
        latcount = latcount + 1;
        latpass_index(latcount) = i;
    elseif dlat(i-1) > 0 && dlat(i) < 0
        latcount = latcount + 1;
        latpass_index(latcount) = i;
    end
end


% Use a linear interpolation to find the longitudinal difference at
each
% latitude pass
for i = 1:latcount
    m = (lat(latpass_index(i)) - lat(latpass_index(i) - 1))./ ...
        (states(latpass_index(i),:) - states(latpass_index(i) - 1,:));
    pass_states(i,:) = states(latpass_index(i),:) + (tgtlat -
lat(latpass_index(i)))./m;
    m2 = (lat(latpass_index(i)) - lat(latpass_index(i) - 1))/ ...
        (time(latpass_index(i)) - time(latpass_index(i) - 1));
    pass_time(i) = time(latpass_index(i)) + (tgtlat -
lat(latpass_index(i)))/m2;
end
```

```
end
```

WGS84Constants.m

```
function WGS84Constants

global mu g0 Rearth wearth J2 J3 J4 J6 FlatE EccE Beta rho0 BR
StefBoltz Boltz

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%% Earth Planetary Constants
mu     = 398600.442;       % Gravitational parameter (km^3/s^2)
Rearth = 6378.137;         % Planetary radius (km)
g0     = mu/(Rearth^2);    % Sea-level gravitational acceleration
(km/s^2)
wearth = 7.2921158e-5;     % Planetary rotational velocity (rad/s)

%Jeffery's Constants
J2     =  0.0010826269;
J3     = -0.0000025323;
J4     = -0.0000016204;
J6     = -0.0000021;

%Planetary Eccentricity Calculation
FlatE  = 1.0/298.257;         %Flattening parameter (f)
EccE2  = (2.0 - FlatE)*FlatE; %Square of planetary eccentricity
EccE   = sqrt(EccE2);         %Planetary eccentricity

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%% Earth Atmospheric Constants
Beta   = 0.14;             %Atmospheric scale height (km^-1)
rho0   = 1.225 * (1000)^3; %Atmospheric density @ planetary surface
(kg/km^3)
BR     = 900;              %Average atmos. parameter for universal
formulation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%% Physical Constants
StefBoltz = 5.67E-8;       %Stefan-Boltzmann constant (W.m^-2.K^-4)
Boltz     = 1.380658E-23;  %Boltzmann constant (J/K)
```

Overfly_GUI.m

```
function varargout = gui_learning(varargin)
% GUI_LEARNING MATLAB code for gui_learning.fig
```

```
%      GUI_LEARNING, by itself, creates a new GUI_LEARNING or raises
the existing
%      singleton*.
%
%      H = GUI_LEARNING returns the handle to a new GUI_LEARNING or the
handle to
%      the existing singleton*.
%
%      GUI_LEARNING('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in GUI_LEARNING.M with the given input
arguments.
%
%      GUI_LEARNING('Property','Value',...) creates a new GUI_LEARNING
or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before gui_learning_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to gui_learning_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help gui_learning

% Last Modified by GUIDE v2.5 06-Sep-2013 16:07:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui_learning_OpeningFcn, ...
                   'gui_OutputFcn',  @gui_learning_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before gui_learning is made visible.
function gui_learning_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui_learning (see VARARGIN)

% Choose default command line output for gui_learning
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes gui_learning wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = gui_learning_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --------------- EXECUTES ON RUN BUTTON PRESS ------------------------
----


function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
clc

global tgtlat tgtlon vehicle
global mu g0 Rearth wearth J2 J3 J4 J6 FlatE EccE Beta Rho0 BR
global StefBoltz Boltz



%%            ------ GET INPUTS FROM USER INTERFACE ------

initial_orbit.a = str2num(get(handles.a, 'string'));
initial_orbit.e = str2num(get(handles.e, 'string'));
initial_orbit.i = str2num(get(handles.i, 'string'));
initial_orbit.AoP = str2num(get(handles.w, 'string'));
```

```
initial_orbit.RAAN = str2num(get(handles.O, 'string'));
initial_orbit.nu = str2num(get(handles.v, 'string'));

vehicle.m = str2num(get(handles.m, 'string'));
vehicle.S = str2num(get(handles.S, 'string'))*1e-6;
vehicle.Cl = str2num(get(handles.Cl, 'string'));
vehicle.Cd = str2num(get(handles.Cd, 'string'));

years = get(handles.years, 'string');
months = get(handles.months, 'string');
days = get(handles.days, 'string');
hours = get(handles.hours, 'string');
month = months(get(handles.months, 'value'));
if strcmp(month,'Jan')
    startdate.month = 1;
elseif strcmp(month,'Feb')
    startdate.month = 2;
elseif strcmp(month,'Mar')
    startdate.month = 3;
elseif strcmp(month,'Apr')
    startdate.month = 4;
elseif strcmp(month,'May')
    startdate.month = 5;
elseif strcmp(month,'Jun')
    startdate.month = 6;
elseif strcmp(month,'July')
    startdate.month = 7;
elseif strcmp(month,'Aug')
    startdate.month = 8;
elseif strcmp(month,'Sep')
    startdate.month = 9;
elseif strcmp(month,'Oct')
    startdate.month = 10;
elseif strcmp(month,'Nov')
    startdate.month = 11;
elseif strcmp(month,'Dec')
    startdate.month = 12;
end
startdate.year = str2double(years(get(handles.years, 'value')));
startdate.day = str2double(days(get(handles.days, 'value')));
startdate.hour = hours(get(handles.hours, 'value'));
if strcmp(startdate.hour,'Start Hour')
    startdate.hour = 0;
else
    startdate.hour = str2double(startdate.hour);
end
startdate.min = str2double(get(handles.minute, 'string'));
startdate.sec = str2double(get(handles.second, 'string'));

simtime = str2num(get(handles.simtime, 'string'))*3600;

tgtlat = str2num(get(handles.tgtlat, 'string'));
tgtlon = str2num(get(handles.tgtlon, 'string'));
```

```
% Update user interface input values
set(handles.slider_a, 'value', initial_orbit.a);
set(handles.slider_e, 'value', initial_orbit.e);
set(handles.slider_i, 'value', initial_orbit.i);
set(handles.slider_w, 'value', initial_orbit.AoP);
set(handles.slider_O, 'value', initial_orbit.RAAN);
set(handles.slider_v, 'value', initial_orbit.nu);

initial_orbit
vehicle
tgtlon
tgtlat
startdate
simtime

%% Load Constants and Convert initial orbit to relative planet fixed
parameters

%Optimal_Overfly_Inputs;       % Loads User Defined Inputs
WGS84Constants;              % Loads Earth WGS 84 Constants

% Convert Initial Orbit to inertial R(km/s), Lat(rad), Long(rad),
V(km/s),
% fpa(rad), and heading angle(rad)
initial_states = OrbEl_to_PlanetFix(initial_orbit, startdate);

% Error message if initial orbit is below the earth
starterror = [];
if initial_states(1) < Rearth
    display('Initial state is below the surface of the earth.')
    starterror = 1;
end

% Convert initial parameters from inertial to planet relative
rel_initial_states = Inertial2Rel_States(initial_states);



%% Propagate initial orbit forward in time for the user specified
% simulation time
if isempty(starterror)
options = odeset('RelTol',1e-7,'AbsTol',1e-7,'Events',@breakevent);
dv = 0;
rel_initial_states(4) = rel_initial_states(4) + dv;
[time,rel_states] = ode45(@equations_of_motion_ODE45,[0
simtime],rel_initial_states,options);
datapoints = length(time);
for i = 1:datapoints
    if rel_states(i,2) > pi
        rel_states(i:end,2) = -(pi - (rel_states(i:end,2) - pi));
    elseif rel_states(i:end,2) < -pi
        rel_states(i:end,2) = (pi + (rel_states(i:end,2) + pi));
```

```
    end
end
if time(end) ~= simtime
    display('Orbit intersects the earth.')
end

% Find the longitudinal and latitudinal difference at each point
for i = 1:datapoints
    if rel_states(i,2) > 0 && tgtlon < 0
        dlon(i) = -360+(rad2deg(rel_states(i,2)) - tgtlon);
    elseif rel_states(i,2) < 0 && tgtlon > 0
        dlon(i) = 360 + (rad2deg(rel_states(i,2)) - tgtlon);
    else
        dlon(i) = rad2deg(rel_states(i,2)) - tgtlon;
    end
    if dlon(i) > 180
        dlon(i:end) = -(180 - (dlon(i:end) - 180));
    elseif dlon(i:end) < -180
        dlon(i:end) = (180 + (dlon(i:end) + 180));
    end
end
dlat = rad2deg(rel_states(:,3)) - tgtlat;
lat = rel_states(:,3);
lon = rel_states(:,2);
distance = acos(sin(lat).*sind(tgtlat) + ...
           cos(lat).*cosd(tgtlat).*cosd(transpose(dlon)))*Rearth;
ddist_dt = diff(distance);

% Find the lat/long difference for the closest point of each pass
count = 1;
for i = 2:datapoints-1
    if ddist_dt(i-1) < 0 && ddist_dt(i) > 0
        mindist_index(count) = i;
        count = count + 1;
    end
end
dlon_miss = dlon(mindist_index);
dlat_miss = dlat(mindist_index);

miss_tol = 1;
initial_phase_states = rel_initial_states;
phase_dlon_miss = dlon_miss;
phase_dv = 0.1;
k=1;
for i = 2:2%count-1
    distance(mindist_index(i))
    while distance(mindist_index(i)) > miss_tol

        if phase_dlon_miss(i) > 0
            zeroin(k) = 1;
        else
            zeroin(k) = -1;
        end
        if k > 1
```

```
            if zeroin(k) == -zeroin(k-1)
                phase_dv = phase_dv/2;
            end
        end
        if k > 2
            if zeroin(k) == zeroin(k-1) && zeroin(k) == -zeroin(k-2)
                phase_dv = phase_dv/2;
            end
        end
        k = k+1
        phase_dv


        if phase_dlon_miss(i) > 0;
            initial_phase_states(4) = initial_phase_states(4) -
phase_dv;
        else
            initial_phase_states(4) = initial_phase_states(4) +
phase_dv;
        end

        [phase_time,phase_states] =
ode45(@equations_of_motion_ODE45,...
                [0 simtime],initial_phase_states,options);


        phase_points = length(phase_time)
        for ii = 1:phase_points
            if phase_states(ii,2) > pi
                phase_states(ii:end,2) = -(pi - (phase_states(ii:end,2)
- pi));
            elseif rel_states(ii:end,2) < -pi
                phase_states(ii:end,2) = (pi + (phase_states(ii:end,2)
+ pi));
            end
        end


        for ii = 1:phase_points
            if phase_states(ii,2) > 0 && tgtlon < 0
                phase_dlon(ii) = -360+(rad2deg(phase_states(ii,2)) -
tgtlon);
            elseif phase_states(ii,2) < 0 && tgtlon > 0
                phase_dlon(ii) = 360 + (rad2deg(phase_states(ii,2)) -
tgtlon);
            else
                phase_dlon(ii) = rad2deg(phase_states(ii,2)) - tgtlon;
            end
            if phase_dlon(ii) > 180
                phase_dlon(ii:end) = -(180 - (phase_dlon(ii:end) -
180));
            elseif dlon(i:end) < -180
                phase_dlon(ii:end) = (180 + (phase_dlon(ii:end) +
180));
```

```
            end
        end
        phase_dlat = rad2deg(phase_states(:,3)) - tgtlat;
        phase_lat = phase_states(:,3);
        phase_lon = phase_states(:,2);
        phase_lat(phase_points+1:end) = [];
        phase_dlon(phase_points+1:end) = [];
        phase_dlat(phase_points+1:end) = [];
        phase_distance = acos(sin(phase_lat).*sind(tgtlat) + ...

cos(phase_lat).*cosd(tgtlat).*cosd(transpose(phase_dlon)))*Rearth;
        phase_ddist_dt = diff(phase_distance);
        phase_count = 1;
        for ii = 2:phase_points-1
            if phase_ddist_dt(ii-1) < 0 && phase_ddist_dt(ii) > 0
                phase_mindist_index(phase_count) = ii;
                phase_count = phase_count + 1;
            end
        end
        if phase_count == 1
            phase_mindist_index = [];
            phase_dlon_miss = -1;
        else
            phase_dlon_miss = phase_dlon(phase_mindist_index);
            phase_dlat_miss = phase_dlat(phase_mindist_index);
        end



        %globe_plot(phase_states)
        if k == 20
            break
        end

    end
end



% -- Save wanted data --
% save('initial_orbit.mat','initial_orbit')
% save('rel_initial_states.mat','rel_initial_states')
% save('rel_states.mat','rel_states')
% save('distance.mat','distance')
% save('phase_states.mat','phase_states')

% -- Plot wanted data --
% globe_plot(rel_states);
% plotstates(rel_states,time);
% groundtrack(rel_states);
% figure
% plot(time,distance)
% figure
% plot(time(1:datapoints-1),ddist_dt)
```

```
end



% --- Executes on slider movement.
function slider_a_Callback(hObject, eventdata, handles)
% hObject    handle to slider_a (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
set(handles.a, 'string', get(handles.slider_a, 'Value'));



% --- Executes during object creation, after setting all properties.
function slider_a_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_a (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function a_Callback(hObject, eventdata, handles)
% hObject    handle to a (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of a as text
%        str2double(get(hObject,'String')) returns contents of a as a
double



% --- Executes during object creation, after setting all properties.
function a_CreateFcn(hObject, eventdata, handles)
% hObject    handle to a (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function e_Callback(hObject, eventdata, handles)
% hObject    handle to e (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of e as text
%        str2double(get(hObject,'String')) returns contents of e as a
double


% --- Executes during object creation, after setting all properties.
function e_CreateFcn(hObject, eventdata, handles)
% hObject    handle to e (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on slider movement.
function slider_e_Callback(hObject, eventdata, handles)
% hObject    handle to slider_e (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
set(handles.e, 'string', get(handles.slider_e, 'Value'));



% --- Executes during object creation, after setting all properties.
function slider_e_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_e (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
```

```
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider_i_Callback(hObject, eventdata, handles)
% hObject    handle to slider_i (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
set(handles.i, 'string', get(handles.slider_i, 'Value'));

% --- Executes during object creation, after setting all properties.
function slider_i_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_i (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end




function i_Callback(hObject, eventdata, handles)
% hObject    handle to i (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of i as text
%        str2double(get(hObject,'String')) returns contents of i as a
double


% --- Executes during object creation, after setting all properties.
function i_CreateFcn(hObject, eventdata, handles)
% hObject    handle to i (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function slider_w_Callback(hObject, eventdata, handles)
% hObject    handle to slider_w (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
set(handles.w, 'string', get(handles.slider_w, 'Value'));

% --- Executes during object creation, after setting all properties.
function slider_w_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_w (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function w_Callback(hObject, eventdata, handles)
% hObject    handle to w (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of w as text
%        str2double(get(hObject,'String')) returns contents of w as a
double


% --- Executes during object creation, after setting all properties.
function w_CreateFcn(hObject, eventdata, handles)
% hObject    handle to w (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function slider_O_Callback(hObject, eventdata, handles)
% hObject    handle to slider_O (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
set(handles.O, 'string', get(handles.slider_O, 'Value'));

% --- Executes during object creation, after setting all properties.
function slider_O_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_O (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function O_Callback(hObject, eventdata, handles)
% hObject    handle to O (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of O as text
%        str2double(get(hObject,'String')) returns contents of O as a
double


% --- Executes during object creation, after setting all properties.
function O_CreateFcn(hObject, eventdata, handles)
% hObject    handle to O (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on slider movement.
function slider_v_Callback(hObject, eventdata, handles)
% hObject    handle to slider_v (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
set(handles.v, 'string', get(handles.slider_v, 'Value'));


% --- Executes during object creation, after setting all properties.
function slider_v_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_v (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function v_Callback(hObject, eventdata, handles)
% hObject    handle to v (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of v as text
%        str2double(get(hObject,'String')) returns contents of v as a
double


% --- Executes during object creation, after setting all properties.
function v_CreateFcn(hObject, eventdata, handles)
% hObject    handle to v (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function tgtlat_Callback(hObject, eventdata, handles)
% hObject    handle to tgtlat (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of tgtlat as text
%        str2double(get(hObject,'String')) returns contents of tgtlat
as a double


% --- Executes during object creation, after setting all properties.
function tgtlat_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tgtlat (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function tgtlon_Callback(hObject, eventdata, handles)
% hObject    handle to tgtlon (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of tgtlon as text
%        str2double(get(hObject,'String')) returns contents of tgtlon
as a double


% --- Executes during object creation, after setting all properties.
function tgtlon_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tgtlon (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on selection change in targetmenu.
function targetmenu_Callback(hObject, eventdata, handles)
% hObject    handle to targetmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns targetmenu
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
targetmenu
targets = get(handles.targetmenu, 'string');
tgtlocation = targets(get(handles.targetmenu, 'value'));
if strcmp(tgtlocation,'Dayton, Oh')
    tgtlat = 39.758944;
    tgtlon = -84.191629;
elseif strcmp(tgtlocation,'Moscow, Russia')
    tgtlat = 55.751244;
    tgtlon = 37.618423;
elseif strcmp(tgtlocation,'Tehran, Iran')
    tgtlat = 35.696216;
    tgtlon = 51.422945;
else
    tgtlat = str2num(get(handles.tgtlat, 'string'));
    tgtlon = str2num(get(handles.tgtlon, 'string'));
end
set(handles.tgtlat, 'string', tgtlat);
set(handles.tgtlon, 'string', tgtlon);


% --- Executes during object creation, after setting all properties.
function targetmenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to targetmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Cl_Callback(hObject, eventdata, handles)
% hObject    handle to Cl (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Cl as text
```

```
%        str2double(get(hObject,'String')) returns contents of Cl as a
double


% --- Executes during object creation, after setting all properties.
function Cl_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Cl (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Cd_Callback(hObject, eventdata, handles)
% hObject    handle to Cd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Cd as text
%        str2double(get(hObject,'String')) returns contents of Cd as a
double


% --- Executes during object creation, after setting all properties.
function Cd_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Cd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function m_Callback(hObject, eventdata, handles)
% hObject    handle to m (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of m as text
```

```
%        str2double(get(hObject,'String')) returns contents of m as a
double


% --- Executes during object creation, after setting all properties.
function m_CreateFcn(hObject, eventdata, handles)
% hObject    handle to m (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function S_Callback(hObject, eventdata, handles)
% hObject    handle to S (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of S as text
%        str2double(get(hObject,'String')) returns contents of S as a
double


% --- Executes during object creation, after setting all properties.
function S_CreateFcn(hObject, eventdata, handles)
% hObject    handle to S (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --------------------------------------------------------------------
function Untitled_1_Callback(hObject, eventdata, handles)
% hObject    handle to Untitled_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on selection change in years.
function years_Callback(hObject, eventdata, handles)
% hObject    handle to years (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns years
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
years


% --- Executes during object creation, after setting all properties.
function years_CreateFcn(hObject, eventdata, handles)
% hObject    handle to years (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in months.
function months_Callback(hObject, eventdata, handles)
% hObject    handle to months (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns months
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
months


% --- Executes during object creation, after setting all properties.
function months_CreateFcn(hObject, eventdata, handles)
% hObject    handle to months (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on selection change in days.
function days_Callback(hObject, eventdata, handles)
% hObject    handle to days (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns days
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from days


% --- Executes during object creation, after setting all properties.
function days_CreateFcn(hObject, eventdata, handles)
% hObject    handle to days (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in hours.
function hours_Callback(hObject, eventdata, handles)
% hObject    handle to hours (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns hours
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
hours


% --- Executes during object creation, after setting all properties.
function hours_CreateFcn(hObject, eventdata, handles)
% hObject    handle to hours (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function minute_Callback(hObject, eventdata, handles)
% hObject    handle to minute (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of minute as text
%        str2double(get(hObject,'String')) returns contents of minute
as a double


% --- Executes during object creation, after setting all properties.
function minute_CreateFcn(hObject, eventdata, handles)
% hObject    handle to minute (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function second_Callback(hObject, eventdata, handles)
% hObject    handle to second (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of second as text
%        str2double(get(hObject,'String')) returns contents of second
as a double


% --- Executes during object creation, after setting all properties.
function second_CreateFcn(hObject, eventdata, handles)
% hObject    handle to second (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function simtime_Callback(hObject, eventdata, handles)
% hObject    handle to simtime (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of simtime as text
%        str2double(get(hObject,'String')) returns contents of simtime
as a double


% --- Executes during object creation, after setting all properties.
function simtime_CreateFcn(hObject, eventdata, handles)
% hObject    handle to simtime (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

globe_plot2.m

```
function globe_plot2(states, states2)
% Plots six position and velocity states as a function of time

global tgtlon tgtlat
states(:,2:3) = rad2deg(states(:,2:3));
states(:,5:6) = rad2deg(states(:,5:6));
states2(:,2:3) = rad2deg(states2(:,2:3));
states2(:,5:6) = rad2deg(states2(:,5:6));
figure
worldmap('World')
load coast
plotm(lat, long,'k')
geoshow('landareas.shp', 'FaceColor', [0.3 0.7 0.15])
plotm(states(:,3),states(:,2),'r','linewidth',2.2)
plotm(states2(:,3),states2(:,2),'y--','linewidth',2.5)
%plotm(tgtlat,tgtlon,'b.','markersize',20)
hold on



end
```

plot_states.m

```
function plotstates(states, time)
global Rearth
% Plots six position and velocity states as a function of time

% Inputs: initial_states():
%                   1 - Radius (km)
%                   2 - Longitude (rads)
%                   3 - Latitude (rads)
%                   4 - Velocity (km/s)
%                   5 - Flight Path Angle (rads)
%                   6 - Heading Angle (rads)
%          time - time (seconds)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Outputs: Subplot of states vs time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

states(:,2:3) = rad2deg(states(:,2:3));
states(:,5:6) = rad2deg(states(:,5:6));
states(:,1) = states(:,1) - Rearth;
figure
subplot(3,2,1)
plot(time,states(:,1),'linewidth',1.75)
grid on
%ylim([0 1000])
xlabel('Time (s)')
ylabel('Altitude (km)')

subplot(3,2,2)
plot(time,states(:,2),'linewidth',1.75)
grid on
xlabel('Time (s)')
ylabel('Longitude (deg)')

subplot(3,2,4)
plot(time,states(:,3),'linewidth',1.75)
grid on
xlabel('Time (s)')
ylabel('Latitude (deg)')

subplot(3,2,3)
plot(time,states(:,4),'linewidth',1.75)
grid on
xlabel('Time (s)')
ylabel('Velocity (km/s)')

subplot(3,2,5)
plot(time,states(:,5),'linewidth',1.75)
grid on
```

```
xlabel('Time (s)')
ylabel('Flight Path Angle (deg)')

subplot(3,2,6)
plot(time,states(:,6),'linewidth',1.75)
grid on
xlabel('Time (s)')
ylabel('Heading Angle (deg)')
```

TOA_vs_dV_All.m

```
function TOA_vs_dV_all(time1, dv1, time2, dv2, time3, dv3, time4, dv4,
bank_angle)
global Rearth
% Plots Required delta V in km/s vs. time of arrival in minutes

% Inputs: time1 -- phase time of arrival (seconds)
%         dv1   -- phase change in velocity (km/s)
%         time2 -- plane time of arrival (seconds)
%         dv2   -- plane change in velocity (km/s)
%         time3 -- skip time of arrival (seconds)
%         dv3   -- skip change in velocity (km/s)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Outputs: dv vs toa plot
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

global tgtlat tgtlon initialorbit


time1 = time1/3600;
time2 = time2/3600;
time3 = time3/3600;
time4 = time4/3600;

%figure('Position',get(0,'ScreenSize'))
figure
plot(time1,dv1,'b*','linewidth',3)
hold on
plot(time2,dv2,'g*','linewidth',3)
plot(time3,dv3,'r*','linewidth',3)
plot(time4,dv4,'k*','linewidth',3)
grid on
xlabel('Time of Arrival (hours)')
ylabel('Change in Velocity (km/s)')
title1 = ['  Target Lat = ' num2str(tgtlat)];
title2 = ['  Target Lon = ' num2str(tgtlon)];
```

```
title3 = ['        Initial Orbit a = ' num2str(initialorbit.a)];
title4 = ['                        e = ' num2str(initialorbit.e)];
title5 = ['                           i = ' num2str(initialorbit.i)];
title6 = ['            RAAN = ' num2str(initialorbit.RAAN)];
title7 = ['                        w = ' num2str(initialorbit.AoP)];
title8 = ['True Anamoly = ' num2str(initialorbit.nu)];
title9 = [       'Bank = ' num2str(bank_angle)];
title1 = [title1 title2];
title({title1,title3,title4,title5,title6,title7,title8,title9},'fontsi
ze',12)
legend('Phasing Maneuver','Simple Plane Change Maneuver','Atmospheric
Skip Maneuver','Re-Circularizing Atmospheric
Maneuver','Location','NorthOutside')
```

Reachability.m

```
%% --- Optimal Space Maneuvering for Ground Target Overfly Version 1.5
---
%                                                                  Devin
Dalton
%                                                                    7 Aug
2013
%
% This code calculates the optimal satellite manuever from a fuel
% consumption and delta-V perspective for a given target, time of
arrival,
% and initial orbit.
%
%
% Governing Equations:
%   - Three dimentional entry equations 3.35-3.37 & 3.65-3.67 on
%     p. 42 & 52 of Hicks text
%   - Variables are integrated in dimensional form
%
% Code assumes:
%   - Rotating planet
%   - Non-thrusting entry
%   - Non-planar entry (for planar, sigma set to 0 or 180 deg)
%
% CHANGE FROM LAST VERSION:
%                        1. Bank Angle Sweep
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%


for devin = 5:5
    clear test;
    if devin == 1
        aorbit = 500;
        iorbit = 28.52;
        bankers = 80;
```

```
        testnum = 24;
    elseif devin == 2
        aorbit = 500;
        iorbit = 45;
        bankers = 75;
        testnum = 25;
     elseif devin == 3
        aorbit = 500;
        iorbit = 45;
        bankers = 85;
        testnum = 26;
    elseif devin == 4
        aorbit = 500;
        iorbit = 45;
        bankers = 90;
        testnum = 27;
    elseif devin == 5
        aorbit = 250;
        iorbit = 45;
        bankers = 80;
        testnum = 28;
    elseif devin == 6
        aorbit = 750;
        iorbit = 45;
        bankers = 80;
        testnum = 29;
    elseif devin == 7
        aorbit = 500;
        iorbit = 45;
        bankers = 80;
        testnum = 21;
    end
    devin



% Declare Global Variables
global tgtlat tgtlon initialorbit startdate simtime vehicle inc_sign
global mu g0 Rearth wearth J2 J3 J4 J6 FlatE EccE Beta rho0 BR flagger2
global StefBoltz Boltz bank flagger1 index bankplot timer deltaV tstart
global flagger acc


bankangle = 80;
% testnum = 24;

% Bring in constants and define saved variables that were cleared above
Optimal_Overfly_Inputs;       % Loads User Defined Inputs
WGS84Constants;               % Loads Earth WGS 84 Constants
initialorbit.a = aorbit + Rearth;
initialorbit.i = iorbit;
bankangle = bankers;
% test_num = 231;
lambda= 1e-6;
```

```
D = 1.15;
bank = 80;
load('directory.mat','directory')
%load('test.mat')
load('run.mat')
load('totalcount.mat')
%bank_angle = 84 + run;
%tgtlat = tgtlat + run;
%initialorbit.i = initialorbit.i + run;
%initialorbit.RAAN = initialorbit.RAAN - 26.4 +24/10*run;



% Convert Initial Orbit to inertial R(km/s), Lat(rad), Long(rad),
V(km/s),
% fpa(rad), and heading angle(rad)
initial_states = OrbEl_to_PlanetFix(initialorbit, startdate);
Vp = sqrt(2*mu/initial_states(1));


% Error message if initial orbit is below the earth
starterror = [];
if initial_states(1) < Rearth
    display('Initial state is below the surface of the earth.')
    starterror = 1;
end


% Convert initial parameters from inertial to planet relative
rel_initial_states = Inertial2Rel_States(initial_states);
p_states = rel_initial_states;
p_states(4) = Vp;
p_rel_states = Inertial2Rel_States(p_states);
Vp_rel = p_rel_states(4);




 %%           -- Propagate initial orbit forward in time --


if isempty(starterror)     % Only perform if initial orbit is a valid
orbit
tic
options = odeset('RelTol',1e-6,'AbsTol',1e-6,'Events',@breakevent);
inc_sign = 0;
index = 1;
timer = [];
bankplot = [];
rel_initial_states(4) = rel_initial_states(4);
```

```
[time,rel_states] = ode45(@equations_of_motion_ODE45, ...
    [0 simtime],rel_initial_states,options);
rel_bankplot = bankplot;
rel_gs = acc;
reltimer = timer;


% Correct Longitude to be between -180 and 180 degrees
datapoints = length(time);
for i = 1:datapoints
    if rel_states(i,2) > pi
        rel_states(i:end,2) = -(pi - (rel_states(i:end,2) - pi));
    elseif rel_states(i:end,2) < -pi
        rel_states(i:end,2) = (pi + (rel_states(i:end,2) + pi));
    end
end


% Earth Intersection Warning
if time(end) ~= simtime
    display('Orbit intersects the earth.')
end


% Find the longitudinal and latitudinal difference at each point
for i = 1:datapoints
    dlon(i) = delta_longitude(rad2deg(rel_states(i,2)),tgtlon);
end
dlat = rad2deg(rel_states(:,3)) - tgtlat;
lat = rel_states(:,3);
lon = rel_states(:,2);
distance = acos(sin(lat).*sind(tgtlat) + ...
           cos(lat).*cosd(tgtlat).*cosd(transpose(dlon)))*Rearth;
ddist_dt = diff(distance);


% Find the lat/long difference for the closest point of each pass
count = 1;
for i = 2:datapoints-1
    if ddist_dt(i-1) < 0 && ddist_dt(i) > 0
        mindist_index(count) = i;
        count = count + 1;
    end
end
dlon_miss = dlon(mindist_index);
dlat_miss = dlat(mindist_index);


for i = 1:count-1
    y1 = rad2deg(lat(mindist_index(i) - 1));
    y2 = rad2deg(lat(mindist_index(i)));
    y3 = rad2deg(lat(mindist_index(i) + 1));
    x1 = rad2deg(lon(mindist_index(i) - 1));
```

```
    x2 = rad2deg(lon(mindist_index(i)));
    x3 = rad2deg(lon(mindist_index(i) + 1));
    m1 = (y2 - y1)/(x2 - x1);
    m2 = (y3 - y2)/(x3 - x2);
    xmin1 = (m1*(tgtlat -y2 + m1*x2) + tgtlon)/(1 + m1^2);
    xmin2 = (m2*(tgtlat -y3 + m2*x3) + tgtlon)/(1 + m2^2);
    ymin1 = y2 + m1*(xmin1 - x2);
    ymin2 = y3 + m2*(xmin2 - x2);
    min_dist1 = sqrt((tgtlon - xmin1)^2 + (tgtlat - ymin1)^2);
    min_dist2 = sqrt((tgtlon - xmin2)^2 + (tgtlat - ymin2)^2);
    if min_dist1 < min_dist2
        min_dist(i) = min_dist1;
    else
        min_dist(i) = min_dist2;
    end
end
phase_min_dist = min_dist;
plane_min_dist = min_dist;
skip_min_dist = min_dist;
overflycount = 0;
skip_overflycount = 0;


% Find the d-longitude for each target latitude passing
if abs(tgtlat) <= initialorbit.i
    [dlon_lat, skiptheta] = d_lon_at_lat_func(lon, lat, tgtlon,
tgtlat);
    [dlon_pass_states, dlon_pass_time] = states_at_pass(rel_states,
time, tgtlat);
end
[dlat_lon, phi] = d_lat_at_lon_func(lon, lat, tgtlon, tgtlat);
[pass_states, pass_time] = states_at_lonpass(rel_states, time, tgtlon);
Vp_rel = 11;


% Find the closest northern, easter, southern, and western points if
% tgtlat < inclination otherwise find just the north or souther points
if abs(tgtlat) < initialorbit.i
si = 0;
ni = 0;
for i = 1:length(dlat_lon)
    if dlat_lon(i) < 0
        si = si + 1;
        s_points(si) = -dlat_lon(i);
    else
        ni = ni + 1;
        n_points(ni) = dlat_lon(i);
    end
end
n_point_deg = min(n_points);
s_point_deg = min(s_points);
n_point = acos(sind(tgtlat + n_point_deg)*sind(tgtlat) + ...
          cosd(tgtlat + n_point_deg)*cosd(tgtlat))*Rearth;
s_point = acos(sind(tgtlat - s_point_deg)*sind(tgtlat) + ...
```

```
          cosd(tgtlat - s_point_deg)*cosd(tgtlat))*Rearth;
n_index = find(dlat_lon == n_point_deg);
s_index = find(-dlat_lon == s_point_deg);
n_time = pass_time(n_index);
s_time = pass_time(s_index);
if abs(tgtlat) <= initialorbit.i
    ei = 0;
    wi = 0;
    for i = 1:length(dlon_lat)
        if dlon_lat(i) < 0
            wi = wi + 1;
            w_points(wi) = -dlon_lat(i);
        else
            ei = ei + 1;
            e_points(ei) = dlon_lat(i);
        end
    end
    e_point_deg = min(e_points);
    w_point_deg = min(w_points);
    e_point = acos(sind(tgtlat)*sind(tgtlat) + ...
          cosd(tgtlat)*cosd(tgtlat)*cosd(e_point_deg))*Rearth;
    w_point = acos(sind(tgtlat)*sind(tgtlat) + ...
          cosd(tgtlat)*cosd(tgtlat)*cosd(-w_point_deg))*Rearth;
    e_index = find(dlon_lat == e_point_deg);
    w_index = find(-dlon_lat == w_point_deg);
    e_time = dlon_pass_time(e_index);
    w_time = dlon_pass_time(w_index);
else
    e_point = 0;
    w_point = 0;
    e_time = 0;
    w_time = 0;
end
elseif tgtlat > 0
    n_point = 0;
    e_point = 0;
    w_point = 0;
    n_time = 0;
    e_time = 0;
    w_time = 0;
    si = 0;
    for i = 1:length(dlat_lon)
        if dlat_lon(i) < 0
            si = si + 1;
            s_points(si) = -dlat_lon(i);
        end
    end
    s_point_deg = min(s_points);
    s_point = acos(sind(tgtlat - s_point_deg)*sind(tgtlat) + ...
              cosd(tgtlat - s_point_deg)*cosd(tgtlat))*Rearth;
    s_index = find(-dlat_lon == s_point_deg);
    s_time = pass_time(s_index);
elseif tgtlat < 0
    s_point = 0;
    e_point = 0;
```

```
    w_point = 0;
    s_time = 0;
    e_time = 0;
    w_time = 0;
    ni = 0;
    for i = 1:length(dlat_lon)
        if dlat_lon(i) > 0
            ni = ni + 1;
            n_points(ni) = dlat_lon(i);
        end
    end
    n_point_deg = min(n_points);
    n_point = acos(sind(tgtlat + n_point_deg)*sind(tgtlat) + ...
              cosd(tgtlat + n_point_deg)*cosd(tgtlat))*Rearth;
    n_index = find(dlat_lon == n_point_deg);
    n_time = pass_time(n_index);
end



% Find the 4 section times to be used to correspond bank with the
correct
% direction in ground track shift
t1count = 0;
t2count = 0;
t3count = 0;
t4count = 0;
for i = 2:length(time)
    if rel_states(i-1,3) < 0 && rel_states(i,3) > 0
        t1count = t1count + 1;
        t1(t1count,1) = time(i);
        t1(t1count,2) = 1;
    elseif rel_states(i-1,6) > 0 && rel_states(i,6) < 0
        t2count = t2count + 1;
        t2(t2count,1) = time(i);
        t2(t2count,2) = 2;
    elseif rel_states(i-1,3) > 0 && rel_states(i,3) < 0
        t3count = t3count + 1;
        t3(t3count,1) = time(i);
        t3(t3count,2) = 3;
    elseif rel_states(i-1,6) < 0 && rel_states(i,6) > 0
        t4count = t4count + 1;
        t4(t4count,1) = time(i);
        t4(t4count,2) = 4;
    end
end
tsection1 = [t1 ; t2; t3; t4];
[tsection2 tindex] = sort(tsection1(:,1));
for i = 1:length(tsection1)
    tsection_start(i,1) = tsection2(i,1);
    tsection_start(i,2) = tsection1(tindex(i),2);
end
%globe_plot(rel_states)
```

```
%% -----------------Determine Min Dv to Enter Atmosphere --------------
---

min_dv_step = 0.0025;
findmin_states = rel_initial_states;
Rmin = min(rel_states(:,1));
Rsensible = 6515;

counter = 0;
while Rmin > Rsensible
    findmin_states(4) = findmin_states(4) - min_dv_step;
    [min_time,min_states] = ode45(@equations_of_motion_ODE45, ...
        [0 11500],findmin_states,options);
    Rmin = min(min_states(:,1));
    counter = counter+1;
end
min_dv = findmin_states(4);
delta_min_dv = rel_initial_states(4) - min_dv;
%% ------------------------ SKIP MANEUVER ----------------------------
---


% Initialize outer variables
options = odeset('RelTol',1e-7,'AbsTol',1e-7,'Events',@breakevent);
skip_inc = initialorbit.i;
if initialorbit.i <= 90
    orig_proretro = 1;
else
    orig_proretro = -1;
end
tsection = tsection_start;




% Initialize outer loop variables
skip_initial_states = rel_initial_states;
skip_v = rel_initial_states(4);
skip_dv_i = 0.01;
k=0;
proretro = orig_proretro;
skip_error = [];
reentry = [];
bankflag = [];
getlower = 0;
gethigher = 0;
getlower_count = 0;
stayV = 0;
kcount = 0;
startlate = 0;
skipstartflag = 0;
```

```
% Change velocity until minimum distance is within tolerances or until
% determined that overflight for particular longitudinal crossing is
% not possible
counterlog = 0;
for ti = 10:12          % Sweep Through Maneuver Start Time
    ti
for bi = 1:2           % Switch Between positive and negative Bank
skip_dv = .0;
counterlog = counterlog+1;
for i = 1:50
    if bi == 1
        banksign = 1;
    else
        banksign = -1;
    end
    % Sweep Through Initial Velocity
    skip_dv = skip_dv + skip_dv_i;
    skip_initial_states(4) = min_dv - skip_dv;


    % Find the flight profile with the change in velocity occurring at
    % time 0.
    index = 1;
    timer = [];
    bankplot = [];
    acc = [];
    bank = bankangle;
    stop = 0;
    [skip_time_11,skip_states_11] =
ode45(@equations_of_motion_ODE45_bank2,...
            [0 simtime],skip_initial_states,options);
    skip_points = length(skip_time_11);
    skip_bankplot_11 = bankplot;
    skip_gs_11 = acc;
    skip_timer_11 = timer;
    %plotstates(skip_states_11,skip_time_11)
    %globe_plot2(rel_states,skip_states_11)

    % Find the time it takes to enter the sensible atm and restart loop
    % if sensible atm is not reached
    ii = 0;
    while stop == 0
        ii = ii + 1;
        if skip_states_11(ii,1) < Rsensible
            stop = 1;
            tmaneuver = skip_time_11(ii);
        end
    end
```

```
    tstart = tsection(13,1) +(tsection(16,1) -
tsection(12,1))/(12)*(ti-1) - tmaneuver;



    % Find state variables at maneuver start time and the change in
    % velocity for the first phase of the trajectory
    stop = 0;
    ii = 0;
    while stop == 0;
        ii = ii + 1;
        if time(ii) > tstart
            time_index = ii - 1;
            stop = 1;
        end
    end
    ii = 0;
    stop = 0;
    while stop == 0;
        ii = ii + 1;
        if reltimer(ii) > tstart
            timer_index = ii - 1;
            stop = 1;
        end
    end
    skiptime_initial_states = rel_states(time_index,:);
    skiptime_initial_states(4) = skiptime_initial_states(4) -
delta_min_dv - skip_dv;
    skip_dv1 = rel_states(time_index,4) - ...
        skiptime_initial_states(4);


    % Calculate the flight profile with the bank maneuver done
    % a the correct time
    index = 1;
    timer = [];
    bankplot = [];
    acc = [];
    [skip_time_11,skip_states_11] = ...
        ode45(@equations_of_motion_ODE45,...
            [0 simtime-tstart],skiptime_initial_states,options);
     skip_bankplot_11 = bankplot;
     skip_gs_11 = acc;
     skip_timer_11 = timer;


    % Concatenate the pre-skip data with the skip data
    skip_time_11 = skip_time_11 + tstart;
    skip_timer_11 = skip_timer_11 + tstart;
    skip_time_1 = vertcat(time(1:time_index-1),skip_time_11);
    skip_states_1 = vertcat(rel_states(1:time_index-1,:),...
        skip_states_11);
    skip_bankplot_1 = horzcat(rel_bankplot(1:timer_index-1),...
```

```
        skip_bankplot_11);
    skip_gs_1 = horzcat(rel_gs(1:timer_index-1),skip_gs_11);
    skip_timer_1 = horzcat(reltimer(1:timer_index-1),...
        skip_timer_11);
    skip_points = length(skip_states_1);
    startime = tstart;


    % Find the states at the highest and the lowest point of orbit
    % after 1st skip
    for ii = 2:skip_points
%         if pass_time(i) > tstart
          if skip_states_1(ii,5) < 0 && ...
                  skip_states_1(ii-1,5) > 0 && ...
                   skip_time_1(ii) > (tstart + 100)
              skip_index1 = ii-1;
              reentry = [];
              break
          else
              reentry = 1;
              gethigher = 1;
          end
%         else
%             if skip_states_1(ii,5) < 0 && skip_states_1(ii-1,5) > 0
%                 skip_index1 = ii-1;
%                 reentry = [];
%                 break
%             else
%                 reentry = 1;
%                 gethigher = 1;
%             end
%         end
    end
    time_index1 = find(abs(skip_timer_1 - skip_time_1(skip_index1))...
        == min(abs(skip_timer_1 - skip_time_1(skip_index1))));
    skip_rp = min(skip_states_1(1:skip_index1,1));
    skip_ra = skip_states_1(skip_index1,1);
    skip_rel_initial_states = skip_states_1(skip_index1,:);
    skip_dv2_1 = skip_rel_initial_states(4);


    % Propagate orbit foward from the highest point of the orbit
    % without any bank to find the max latitude in order to later
    % determine the correct heading angle after re-circularization
    bank = 0;
    index = 1;
    timer = [];
    bankplot = [];
    acc = [];
    Period = 2*pi*sqrt(skip_rel_initial_states(1)^3/mu);
    [skip_time_2, skip_states_2] = ode45(@equations_of_motion_ODE45,...
            [0 2*Period],skip_rel_initial_states,options);
    maxlat = max(abs(skip_states_2(:,3)));
```

```
    % Find the circular velocity(relative to the rotating planet) by
    % setting gamma dot = 0 and solving equation 3.66 for relative
    % velocity.  For derived circular velocity iterate to find the
    % corresponding heading angle given the max lat calculated above



    % Initialize loop variables
    kk = 0;
    psides = skip_rel_initial_states(6);
    psi_orig = psides;
    dhead = 0.01;
    loop = 1;


    while loop == 1


        % Define variable used in Eq 3.66
        skipR        = skip_rel_initial_states(1);
        skiptheta    = skip_rel_initial_states(2);
        skipphi      = skip_rel_initial_states(3);
        skipGamma    = 0.0;
        skippsi      = skip_rel_initial_states(6);
        skipSigma    = deg2rad(bank);
        skipg        = g0 *(Rearth/skipR)^2.0;
        skiprho      = AtmosModel(skipR-Rearth,2);
        skipCl       = vehicle.Cl;
        skipS        = vehicle.S;
        skipm        = vehicle.m;


        % Solve for velocity in equation 3.66
        A        =
(1/skipR)+((skiprho*skipCl*skipS)/(2*skipm))*cos(skipSigma);
        B        = 2*wearth*cos(skipphi)*cos(skippsi);
        C        = -(skipg*cos(skipGamma))+
skipR*wearth^2*cos(skipphi)*(cos(skipphi)*cos(skipGamma)+sin(skipphi)*s
in(skippsi)*sin(skipGamma));
        Vcirc2   = (-B+(B^2-4*A*C)^0.5)/(2*A);


        % Find the corresponding Vcirc and heading angle
        psinew   = skippsi;
        done     = 0;
        cnt      = 0;
        skippsi      = psides;
        while (done == 0)
            cnt    = cnt+1;
            done   = 1;
            A      =
(1/skipR)+((skiprho*skipCl*skipS)/(2*skipm))*cos(skipSigma);
```

```
            B      = 2*wearth*cos(skipphi)*cos(skippsi);
            C      = -(skipg*cos(skipGamma))+
skipR*wearth^2*cos(skipphi)*(cos(skipphi)*cos(skipGamma)+sin(skipphi)*s
in(skippsi)*sin(skipGamma));
            Vcirc2 = (-B+(B^2-4*A*C)^0.5)/(2*A);

            psinew = psides +
asin(2*pi*skipR*sin(psides)/(86400*Vcirc2));
            if (abs(skippsi-psinew)>0.0000001)
                done = 0;
                skippsi  = psinew;
            end
        end
        skip_rel_initial_states(6) = psinew;
        skip_rel_initial_states(4) = Vcirc2;
        skip_rel_initial_states(5) = 0;
        skip_dv2 = Vcirc2 - skip_dv2_1;


        % Propogate Circular Orbit forward in time for 2 periods
        index = 1;
        timer = [];
        bankplot = [];
        acc = [];
        starttime = skip_time_1(skip_index1);
        [skip_time_2,skip_states_2] =
ode45(@equations_of_motion_ODE45,[0
2*Period],skip_rel_initial_states,options);
        skip_maxlat = max(skip_states_2(:,3));


        % Compare the max latitude of new circular orbit and compare to
        % known desired max latitude of non-circular orbit.  Break out
        % of loop once the initial heading angle gives the correct max
        % latitude
        if skip_maxlat > maxlat - 0.0001 && skip_maxlat < maxlat +
0.0001;
            break;
        end


        % Algorithm to change the step size of the d-heading correction
        kk = kk+1;
        signchange = sign(skip_rel_initial_states(6));
        if skip_maxlat > maxlat && sign(psi_orig) == signchange
            zeroin2(kk) = 1;
        elseif skip_maxlat > maxlat && sign(psi_orig) ~= signchange
            zeroin2(kk) = -1;
        elseif skip_maxlat < maxlat && sign(psi_orig) == signchange
            zeroin2(kk) = -1;
        elseif skip_maxlat < maxlat && sign(psi_orig) ~= signchange
            zeroin2(kk) = 1;
        end
        if kk > 1
```

```
        if zeroin2(kk) == -zeroin2(kk-1)
            dhead = dhead/2;
        end
    end
    if kk > 2
        if zeroin2(kk) == zeroin2(kk-1) && ...
                zeroin2(kk) == -zeroin2(kk-2)
            dhead = dhead/2;
        end
    end


    % Correct Psi Desired to reach the correct maximum latitude
    psi1 = skip_rel_initial_states(6);
    if skip_maxlat > maxlat && psi1 > 0 && psi1 <= pi/2
        psides = psides - dhead;
    elseif skip_maxlat < maxlat && psi1 > 0 && psi1 <= pi/2
        psides = psides + dhead;
    elseif skip_maxlat > maxlat && psi1 < 0 && psi1 <= pi/2
        psides = psides + dhead;
    elseif skip_maxlat < maxlat && psi1 < 0 && psi1 <= pi/2
        psides = psides - dhead;
    elseif skip_maxlat > maxlat && psi1 > 0 && psi1 > pi/2
        psides = psides + dhead;
    elseif skip_maxlat < maxlat && psi1 > 0 && psi1 > pi/2
        psides = psides - dhead;
    elseif skip_maxlat > maxlat && psi1 < 0 && psi1 > pi/2
        psides = psides - dhead;
    elseif skip_maxlat < maxlat && psi1 < 0 && psi1 > pi/2
        psides = psides + dhead;
    end

    if kk > 25
        break
    end

end

% Calculate the change in inclination
if psi1 > pi/2
    skip_inclination = 180 - rad2deg(maxlat);
else
    skip_inclination = rad2deg(maxlat);
end
skip_delta_i = skip_inclination - initialorbit.i;


% Determine if new orbit changes from prograde to retrograde or
% vice versa
if abs(skip_rel_initial_states(6)) <= pi/2
    proretro = 1;
else
    proretro = -1;
end
```

```
% Propogate Circular Orbit forward in time for the rest of simtime
index = 1;
timer = [];
bankplot = [];
acc = [];
starttime = skip_time_1(skip_index1);
[skip_time_2,skip_states_2] = ode45(@equations_of_motion_ODE45,...
        [0 simtime - starttime],skip_rel_initial_states,options);
skip_maxlat = max(skip_states_2(:,3));
skip_bankplot_2 = bankplot;
skip_gs_2 = acc;
skip_timer_2 = timer;


% Concatenate the first preskip data with post circularization data
skip_time_2 = skip_time_2 +  starttime;
skip_timer_2 = skip_timer_2 + starttime;
skip_time_i = vertcat(skip_time_1(1:skip_index1-1),skip_time_2);
skip_states_i = vertcat(skip_states_1(1:skip_index1-1,:),...
                skip_states_2);
skip_bankplot_i = horzcat(skip_bankplot_1(1:time_index1-1),...
                  skip_bankplot_2);
skip_timer_i = horzcat(skip_timer_1(1:time_index1-1),...
               skip_timer_2);
skip_gs_i = horzcat(skip_gs_1(1:time_index1-1),skip_gs_2);
skip_points = length(skip_time_i);


% Correct Longitude to be from -180 to 180
for ii = 1:skip_points
    if skip_states_i(ii,2) > pi
        skip_states_i(ii:end,2) = -(pi - ...
            (skip_states_i(ii:end,2) - pi));
    elseif skip_states_i(ii:end,2) < -pi
        skip_states_i(ii:end,2) = (pi + ...
            (skip_states_i(ii:end,2) + pi));
    end
end
skip_lat_i = skip_states_i(:,3);
skip_lon_i = skip_states_i(:,2);


% Correct Longitude to be from -180 to 180 for skip_2 data
for ii = 1:length(skip_states_2)
    if skip_states_2(ii,2) > pi
        skip_states_2(ii:end,2) = -(pi - ...
            (skip_states_2(ii:end,2) - pi));
    elseif skip_states_2(ii:end,2) < -pi
        skip_states_2(ii:end,2) = (pi + ...
            (skip_states_2(ii:end,2) + pi));
    end
end
```

```
    skip_lat_2 = skip_states_2(:,3);
    skip_lon_2 = skip_states_2(:,2);


    % Find the change in RAAN
    % First find the longitude of the equatorial crossing after
maneuver
    if ti < 10
        for ii = 2:length(skip_states_2)
            if skip_states_2(ii-1,3) < 0 && skip_states_2(ii,3) > 0
                break
            end
        end
    else
        for ii = 2:length(skip_states_2)
            if skip_states_2(ii-1,3) > 0 && skip_states_2(ii,3) < 0
                break
            end
        end
    end
    lat2 = skip_states_2(ii,3);
    lat1 = skip_states_2(ii-1,3);
    latx = 0;
    lon2 = skip_states_2(ii,2);
    lon1 = skip_states_2(ii-1,2);
    maneuver_lon = (latx - lat1)/(lat2 - lat1)*(lon2 - lon1) + lon1;


    % Now find the Longitude of the same equatorial crossing of no-skip
    % data
    if ti < 10
        ti_index = find(time == tsection(16,1));
    else
        ti_index = find(time == tsection(18,1));
    end
    lat2 = rel_states(ti_index,3);
    lat1 = rel_states(ti_index-1,3);
    latx = 0;
    lon2 = rel_states(ti_index,2);
    lon1 = rel_states(ti_index-1,2);
    orig_lon = (latx - lat1)/(lat2 - lat1)*(lon2 - lon1) + lon1;
    RAAN_shift = rad2deg(maneuver_lon - orig_lon);


    % globe_plot2(rel_states,skip_states_i)
    % plotstates(skip_states_i,skip_time_i)
%    plotm([tgtlat skip_phi],[tgtlon tgtlon],'r')
%             figure
%             plot(skip_timer_i,skip_bankplot_i)


 % Find the velocity needed to re-circularize at the original orbit
    r0 = initial_states(1);
    skip_transfer_dv_guess = sqrt(mu/skip_ra - mu/(skip_ra + r0)) ...
```

```
        - sqrt(mu/skip_ra - mu/(skip_ra+skip_rp));
    skip_ra2 = skip_ra;
    skip_r_tol = 0.1;
    jj = 0;
    transfer_initial_states = skip_rel_initial_states;
    transfer_initial_states(4) = skip_rel_initial_states(4) + ...
                skip_transfer_dv_guess;
    skip_transfer_dv = 0.03;
    while skip_ra2 > r0 + skip_r_tol || skip_ra2 < r0 - skip_r_tol


        index = 1;
        timer = [];
        bankplot = [];
        acc = [];
        [skip_time_t,skip_states_t] =
ode45(@equations_of_motion_ODE45,...
                [0 simtime -
starttime],transfer_initial_states,options);
        skip_ra2 = max(skip_states_t(:,1));
                % Algorithm to change the step size of the dv
correction
        jj = jj+1;
        if skip_ra2 > r0
            zeroin(jj) = 1;
        else
            zeroin(jj) = -1;
        end
        if jj > 1
            if zeroin(jj) == -zeroin(jj-1)
                skip_transfer_dv = skip_transfer_dv/2;
            end
        end
        if jj > 2
            if zeroin(jj) == zeroin(jj-1) && zeroin(jj) == -zeroin(jj-
2)
                skip_transfer_dv = skip_transfer_dv/2;
            end
        end


        if skip_ra2 < r0
            transfer_initial_states(4) = transfer_initial_states(4) +
...
                skip_transfer_dv;
        else
            transfer_initial_states(4) = transfer_initial_states(4) -
...
                skip_transfer_dv;
        end
    end
    transfer_states_index = find(skip_states_t == skip_ra2);
    transfer_states = skip_states_t(transfer_states_index,:);
```

```
     % Define variable used in Eq 3.66
    transfer_R        = transfer_states(1);
    transfer_theta    = transfer_states(2);
    transfer_phi      = transfer_states(3);
    transfer_Gamma    = 0.0;
    transfer_psi      = transfer_states(6);
    transfer_Sigma    = deg2rad(bank);
    transfer_g        = g0 *(Rearth/transfer_R)^2.0;
    transfer_rho      = AtmosModel(transfer_R-Rearth,2);
    transfer_Cl       = vehicle.Cl;
    transfer_S        = vehicle.S;
    transfer_m        = vehicle.m;


    % Solve for velocity in equation 3.66
    A        =
(1/transfer_R)+((transfer_rho*transfer_Cl*transfer_S)/(2*transfer_m))*c
os(transfer_Sigma);
    B        = 2*wearth*cos(transfer_phi)*cos(transfer_psi);
    C        = -(transfer_g*cos(transfer_Gamma))+
transfer_R*wearth^2*cos(transfer_phi)*(cos(transfer_phi)*cos(transfer_G
amma)+sin(transfer_phi)*sin(transfer_psi)*sin(transfer_Gamma));
    transfer_Vcirc   = (-B+(B^2-4*A*C)^0.5)/(2*A);


    % Interpolate other data to find data at actual pass
    [skip_pass_states_i, skip_pass_time_i] = ...
        states_at_lonpass(skip_states_i, skip_time_i, tgtlon);

    skipcount = i + ((bi-1)*27) + ((ti-1)*54);

    test(skipcount,1) = skip_dv1;
    test(skipcount,2) = skip_dv2;
    test(skipcount,3) = transfer_initial_states(4) - skip_dv2_1;
    test(skipcount,4) = transfer_Vcirc - transfer_states(4);
    test(skipcount,5) = skip_dv1 + skip_dv2;
    test(skipcount,6) = test(skipcount,3) + test(skipcount,4)+skip_dv1;
    test(skipcount,7) = skip_delta_i;
    test(skipcount,8) = RAAN_shift;
    test(skipcount,9) = max(skip_gs_i);
    test(skipcount,10) = skip_rp - Rearth;
    test(skipcount,11) = tstart;
    test(skipcount,12) = tstart + tmaneuver;
    test(skipcount,13) = (bankangle)*banksign;
    test(skipcount,14) = counterlog;
    test(skipcount,15) = initialorbit.i;
    test(skipcount,16) = initialorbit.a - Rearth;
%     if max(skip_gs_i) > 10
%         break
%     end
```

```
end %------------------ End of while loop -------------------------%

end
end
end


 filewrite = ['RAAN and Inc for dv sweep', num2str(testnum)];
xlswrite(filewrite,test)
end
```

Raan_Inc_Surface.m

```
clear all
close all
clc

data = xlsread('RAAN and Inc for dv sweep2.xls');


for i = 1:length(data(:,8))
    if data(i,8) > 180
        data(i,8) = data(i,8) - 360;
    elseif data(i,8) < -180
        data(i,8) = data(i,8) + 360;
    end
end


x = data(:,7);
y = data(:,8);
z = data(:,6);
F = TriScatteredInterp(x,y,z);


xi = linspace(min(x),max(x),300);
yi = linspace(min(y),max(y),300);
[qx,qy] = meshgrid(xi,yi);
qz = F(qx,qy);
% qz(end,end) = 5;
% qz(1,1) = 0;

figure
subplot(2,1,1)
mesh(qx,qy,qz);
hold on
% plot3(x,y,z,'.k','markersize',14)
grid on
xlabel('Change in Inclination (degs)','fontsize',14)
ylabel('Change in RAAN (degs) ','fontsize',14)
zlabel('Velocity Change (km/s)','fontsize',14)
% colorbar
```

```
% title({'Velocity Change Cost for a Desired Inclination and RAAN
Change',...
%    'Using an Atmospheric Skip Manuever w/ Re-Circularization at
Original Altitude','Bank = 80 degs','Initial Orbit:','a = 6878 km',...
%    'e = 0.0','i = 35 degs'},'fontsize',16)


subplot(2,1,2)
contourf(qy,qx,qz,18);
hold on
% plot3(x,y,z,'.k','markersize',14)
grid on
xlabel('Change in Inclination (degs)','fontsize',14)
ylabel('Change in RAAN (degs) ','fontsize',14)
zlabel('Velocity Change (km/s)','fontsize',14)
colorbar
% title({'Velocity Change Cost for a Desired Inclination and RAAN
Change',...
%    'Using an Atmospheric Skip Manuever w/ Re-Circularization at
Original Altitude','Bank = 80 degs','Initial Orbit:','a = 6878 km',...
%    'e = 0.0','i = 35 degs'},'fontsize',16)



figure
hold on
grid on
for i = 1:24
    if i == 1
        color = 'db';
    elseif i == 2
        color = 'ob';
    elseif i == 3
        color = 'dr';
    elseif i == 4
        color = 'or';
    elseif i == 5
        color = 'dg';
    elseif i == 6
        color = 'og';
    elseif i == 7
        color = 'dc';
    elseif i == 8
        color = 'oc';
    elseif i == 9
        color = 'dk';
    elseif i == 10
        color = 'ok';
    elseif i == 11
        color = 'dm';
    elseif i == 12
        color = 'om';
    elseif i == 13
        color = '*b';
```

```
    elseif i == 14
        color = '+b';
    elseif i == 15
        color = '*r';
    elseif i == 16
        color = '+r';
    elseif i == 17
        color = '*g';
    elseif i == 18
        color = '+g';
    elseif i == 19
        color = '*c';
    elseif i == 20
        color = '+c';
    elseif i == 21
        color = '*k';
    elseif i == 22
        color = '+k';
    elseif i == 23
        color = '*m';
    elseif i == 24
        color = '+m';
    end
    lowlimit = (i-1)*30+1;
    uplimit = i*30;

plot3(x(lowlimit:uplimit,1),y(lowlimit:uplimit,1),z(lowlimit:uplimit,1)
,color,'markersize',10,'linewidth',2)
    xlabel('Change in Inclination (degs)','fontsize',14)
    ylabel('Change in RAAN (degs) ','fontsize',14)
    zlabel('Delta V (km/s)','fontsize',14)
end
title({'Velocity Change Cost for a Desired Inclination and RAAN
Change',...
   'Using an Atmospheric Skip Manuever w/ Re-Circularization at
Original Altitude','Bank = 80 degs','Initial Orbit:','a = 6878 km',...
   'e = 0.0','i = 28.52 degs (Cape Canaveral Latitude)'},'fontsize',16)
legend({'time 1; (+) Bank','time 1; (-) Bank','time 2; (+) Bank','time
2; (-) Bank','time 3; (+) Bank','time 3; (-) Bank',...
    'time 4; (+) Bank','time 4; (-) Bank','time 5; (+) Bank','time 5;
(-) Bank','time 6; (+) Bank','time 6; (-) Bank','time 7; (+) Bank',...
    'time 7; (-) Bank','time 8; (+) Bank','time 8; (-) Bank','time 9;
(+) Bank','time 9; (-) Bank','time 10; (+) Bank',...
    'time 10; (-) Bank','time 11; (+) Bank','time 11; (-) Bank','time
12; (+) Bank','time 12; (-) Bank'})

figure
plot(data(:,9),data(:,6),'.');
grid on
xlabel('Deceleration (g''s)','fontsize',14)
ylabel('Total Velocity Change Required (km/s','fontsize',14)
xlim([0 10])
```

fitness.m

```
function [ Z ] = fitness( A )


global toa_fit dv_fit lat_fit


% n = max(size(A));


% for i = 1:n
%    Z(i) = norm(dv - A(i)*abs(sin(toa.*2*pi/24-B(i)) + C(i)/A(i)));
% end
if lat_fit < 0
    B= 1.15;
else
    B= 1.15 - pi;
end
Z = norm(dv_fit - A(1)*abs(sin(toa_fit.*2*pi/24-B) + A(2)/A(1)));


end
```

data_sort.m

```
%%                          Data Sort
%%

clear all
close all
clc


% Define the input filename to be sorted and the column to sort by
filename = 'tgtlat v 35i';
sort_on_column = 3;


% Bring in data and sort all of it by defined column
data_all = xlsread(filename);
datapoints = length(data_all(:,sort_on_column));
[sorted_column sort_index] = sort(data_all(:,sort_on_column));
for i = 1:datapoints
    sorted_data_all(i,:) = data_all(sort_index(i),:);
end


% Separate by type
phasecount = 0;
planecount = 0;
```

```
skipcount = 0;
for i = 1:datapoints
    if sorted_data_all(i,23) == 1
        phasecount = phasecount + 1;
        phase_data(phasecount,:) = sorted_data_all(i,:);
    elseif sorted_data_all(i,23) == 2
        planecount = planecount + 1;
        plane_data(planecount,:) = sorted_data_all(i,:);
    elseif sorted_data_all(i,23) ==3
        skipcount = skipcount + 1;
        skip_data(skipcount,:) = sorted_data_all(i,:);
    end
end


% Save data
filename1 = 'tgtlat v 35i';
filename2 = [filename,' sorted'];
filename3 = [filename1, ' phase'];
filename4 = [filename1, ' plane'];
filename5 = [filename1, ' skip'];
testheader = {'Test #', 'Run #', 'Target Lat', 'Target Long', 'Bank
Angle (deg)', ...
    'a', 'e', 'i', 'RAAN', 'AoP', 'True Anamoly', 'Initial V', ...
    'Initial Latitude', 'Initial Longitude', 'N dist', 'E dist', 'W
dist', ...
    'S dist', 'N time', 'E time', 'W time', 'S Time', 'type', 'Pass #',
...
    'Start Time', 'ToA', 'dv1', 'dv2', 'dv3', 'dv4', 'dvtotal 1',
'dvtotal 2', ...
    'Inc Change', 'Perigee R', 'Apogee R', 'Overflight R', 'Perigee
Alt', ...
    'Apogee Alt', 'Overglight Alt', 'Max G''s', 'Heat Flux', 'Ground
Res'};
xlswrite(filename2,testheader);
xlswrite(filename3,testheader);
xlswrite(filename4,testheader);
xlswrite(filename5,testheader);
xlswrite(filename2,sorted_data_all,1,'A2');
xlswrite(filename3,phase_data,1,'A2');
xlswrite(filename4,plane_data,1,'A2');
xlswrite(filename5,skip_data,1,'A2');


% Re-Write data for use in plots
% tgtlat_skip_data(:,:) =
[skip_data(:,26)/3600,skip_data(:,3),skip_data(:,31)];
```

curve_fit_main.m

```
clear all
close all
clc


data = xlsread('tgtlat - 65 to 65 skip.xls');
data2 = xlsread('tgtlat - 65 to 65 plane.xls');
points = length(data);
points2 = length(data2);

% Sort each run into its own variable
lat = -60;
latstr = num2str(abs(lat));
filename = ['data_n',latstr];
filename2 = ['data2_n',latstr];
j = 0;
jj = 0;
for k = 1:40
    for i = 1:points
        if data(i,3) == lat
            j = j + 1;
            eval([filename,'(j,:) = data(i,:);'])
            data_temp(j,:) = data(i,:);
        end
    end
    for i = 1:points2
        if data2(i,3) == lat
            jj = jj + 1;
            eval([filename2,'(jj,:) = data2(i,:);'])
            data2_temp(jj,:) = data2(i,:);
        end
    end
    lat = lat + 1;
    latstr = num2str(abs(lat));
    if lat > 0
        filename = ['data_p',latstr];
        filename2 = ['data2_p',latstr];
    else
        filename = ['data_n',latstr];
        filename2 = ['data2_n',latstr];
    end


    % Create Least Square Matrices
    if j ~= 0
        %for jj = 1
        toa = data_temp(:,25)/3600;
        toa2 = data2_temp(:,26)/3600;
        scale = 2*pi/24;
        scale2 = 2*pi/24;
        toa = toa*scale;
        toa2 = toa2*scale2;
```

```
        dv = data_temp(:,31);
        dv2 = data2_temp(:,31);
        A = [sin(toa) cos(toa) ones(length(toa),1)];
        A2 = [sin(toa2) cos(toa2) ones(length(toa2),1)];
        %A = [toa.^5 toa.^4 toa.^3  toa.^2  toa ones(length(toa),1)];
        b = dv;
        b2 = dv2;
        c = inv(transpose(A)*A)*transpose(A)*b;
        c2 = inv(transpose(A2)*A2)*transpose(A2)*b2;
        x = linspace(0,scale*24,200);
        y = c(1)*sin(x) + c(2)*cos(x) + c(3);
        y2 = c2(1)*sin(x) + c2(2)*cos(x) + c2(3);
        %y = c(1)*x.^5 + c(2)*x.^4 + c(3)*x.^3 +c(4)*x.^2 + c(5)*x
+c(6);
        figure
        plot(toa,dv,'.')
        hold on
        grid on
        ylim([0 13])
        plot(x,y,'r')
        plot(toa2,dv2,'.g')
        plot(x,y2,'k')
        %end

    end

    j = 0;
    jj = 0;
    clear data_temp toa adjust dv A b c x y data2_temp toa2 adjust2 dv2
A2 b2 c2 y2
end

%
% A = [data(:,1).^2 data(:,1) data(:,3)];
% b = data(:,4);
% c = inv(transpose(A)*A)*transpose(A)*b;
% x = linspace(0,30,200);
% %y = c(1)*sqrt(x)+c(2);
% y = c(1)*x.^2 + c(2)*x + c(3)
%
% plot(data(:,1),b,'.')
% hold on
% plot(x,y,'r')
%
%
%
% data = xlsread('least squares dv vs di.xlsx');
% A = [-data(:,1).^2 -data(:,1) data(:,3)];
% b = data(:,4);
% c = inv(transpose(A)*A)*transpose(A)*b;
% x = linspace(0,30,200);
% %y = c(1)*sqrt(x)+c(2);
% y = c(1)*x.^2 + c(2)*x + c(3)
%
% figure
```

```
% plot(data(:,1),b,'.')
% hold on
% plot(x,y,'r')
```

Bibliography

Bate, Roger R., Donald D. Mueller, and Jerry E. White. *Fundamentals of Astrodynamics*. New York, NY: Dover Publications, Inc., 1971.

Bettinger, Robert A. "The Prospect of Responsive Satellites Utilizing Atmospheric Skip Entry Maneuvers." Ph.D Dissertation (DRAFT). School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, February 2014.

Co, Thomas C. *Operationally Responsive Spacecraft Using Electric Propulsion*. Ph.D Dissertation, AFIT/DS/ENY/12-01. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, September 2012 (ADA564646).

Darby, Christopher L. and Anil V. Rao. "Minimum-Fuel Low-Earth Orbit Aeroassisted Orbital Transfer of Small Spacecraft." Paper presented at the 20th AAS/AIAA Space Flight Mechanics Meeting, San Diego, California, 14-17 February 2010

Gonsalves, Paul G., Scott M. Allen, and Alper K. Caglayan. "An Alternative Concept for Aeroassisted Orbit Transfers." *DARPA TR R90222*. Cambridge, MA: Charles River Analytics Inc., 1991.

Griffen, Michael D. and James R. French. *Space Vehicle Design*, Second Edition. Reston, VA: American Institute of Aeronautics and Astronautics, Inc., 2004.

Hicks, Kerry D. *Introduction to Astrodynamic Re-entry*. Wright-Patterson AFB, OH: Air Force Institute of Technology, 2009. (AFIT TR-09-03)

Nicholson, John C. *Numerical Optimization of Synergistic Maneuvers*. MS thesis; Department of Aeronautical and Astronautical Engineering, Naval Postgraduate School, Monterey, CA, June 1994 (ADA283398).

Parish II, Michael S. *Optimality of Aeroassisted Orbital Plane Changes*. MS thesis; Department of Aeronautical and Astronautical Engineering, Naval Postgraduate School, Monterey, CA, December 1995 (ADA306573).

Rao, Anil V. and Arthur E. Scherich. "A Concept for Operationally Responsive Space Mission Planning Using Aeroassisted Orbital Transfer." Paper presented at the 6th Responsive Space Conference, Los Angeles, California, 28 April – 1 May 2008.

Rao, Anil V., Sean Tang, and Wayne P. Hallman. “Numerical Optimization Study of Multiple-Pass Aeroassisted Orbital Transfer.” *Optimal Control Applications and Methods*, 23: 215-238 (2002).

Sellers, Jerry Jon. *Understanding Space; An Introduction to Astronautics*, Third Edition. New York, NY: The McGraw-Hill Companies, Inc., 2004.

Weisel, William E. *Spaceflight Dynamics.* Third Edition. CreateSpace, 2010.

Vallado, David. *Fundamentals of Astrodynamics and Applications*, Fourth Edition. Hawthorne, CA: Microcosm Press Year, 2013.

Vinh, Nguyen X., Adolf Busemann, and Robert D. Culp. *Hypersonic and Planetary Entry Flight Mechanics*. Ann Arbor: The University of Michigan Press, 1980.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
27-03-2014	Master's Thesis	August 2012 – March 2014

TITLE AND SUBTITLE

Ground Target Overflight and Orbital Maneuvering via Atmospheric Maneuvering

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Dalton, Devin K., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENY)
2950 Hobson Way, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENY-14-M-12

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RHIQ (example)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRUBTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
An aeroassisted (skip) maneuver has the potential to decrease fuel costs of re-tasking satellites to overfly ground locations, orbital rendezvous, and initial orbit insertion. The trajectory dynamics of purely propulsive in-plane and out-of-plane maneuvers, along with aeroassisted maneuvers, are simulated in order to determine the time of arrival (TOA) and ΔV associated with each maneuver required to overfly a ground target. Results indicate that aeroassisted maneuvers offer more overflight solutions per day than planar maneuvers while requiring less ΔV than exo-atmospheric plane change maneuvers. The TOA and ΔV associated with each maneuver required to overfly a ground target is found for multiple ground target locations and starting orbits in order to determine analytical trends. From these trends, closed-form estimations of the ΔV and TOA are generated for each maneuver type. Initial closed-form estimations show reasonable accuracy. The ability of the aeroassisted maneuver to modify an initial orbital trajectory is quantified by measuring the change in inclination and right ascension of the ascending node (RAAN) as perigee is lowered. Results show a 75% decrease in ΔV over traditional exo-atmospheric maneuvers with a single skip enabling a satellite to change the orbital inclination and RAAN up to 45° and 90° respectively.

15. SUBJECT TERMS
Skip, Aeroassisted, Orbital Maneuvering, Ground Target, Overflight

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Ronald J. Simmons, Lt Col, USAF,PhD AFIT/ENY
U	**U**	**U**	**UU**	239	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636, ext 4723 (Ronald.simmons@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18