5-2012

Design of Orbital Maneuvers with Aeroassisted Cubesatellites

Stephanie Clark
University of Arkansas, Fayetteville

DESIGN OF ORBITAL MANEUVERS WITH AEROASSISTED CUBESATELLITES

DESIGN OF ORBITAL MANEUVERS WITH AEROASSISTED CUBESATELLITES

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Mechanical Engineering

By

Stephanie Joy Clark
University of Arkansas
Bachelor of Science in Mechanical Engineering 2010

May 2012
University of Arkansas

ABSTRACT

Recent advances within the field of cube satellite technology has allowed for the possible development of a maneuver that utilizes a satellite's Low Earth Orbit (LEO) and increased atmospheric density to effectively use lift and drag to implement a noncoplanar orbital maneuver. Noncoplanar maneuvers typically require large quantities of propellant due to the large $\Delta\upsilon$ that is required. However, similar maneuvers using perturbing forces require little or no propellant to create the $\Delta\upsilon$ required. This research reported here studied on the effects of lift on orbital changes, those of noncoplanar types in particular, for small satellites without orbital maneuvering thrusters.

In order to test the effect of perturbing forces, a Runge-Kutta-based computer model was developed using MATLAB. Fundamental differential equations of satellite motion were used to propagate the time, position, and velocity for each experimental orbit. This model requires the input of initial position and velocity (defined by the Earth-Centered Cartesian Coordinate system), constants, and the number of intervals. In addition, the model used an upper atmosphere model created from the Atmospheric Standard of 1976 to approximate density at altitudes to 1000 kilometers, covering well beyond the interested altitude on the order of 100's of kilometers. Utilizing the orbital model, a lift comparison was completed between a constant angle of attack and scheduled change of the angle of attack; both comparisons were done with varying accommodation coefficients. In addition, a lift comparison was also completed based on scaling laws to analyze any differing effects caused by increase mass and surface area.

This thesis is approved for recommendation
to the Graduate Council.

Thesis Director:

____

Dr. Adam Huang

Thesis Committee:

____

Dr. Rick Couvillion

____

Dr. Larry Roe

THESIS DUPLICATION RELEASE

I hereby authorize the University of Arkansas Libraries to duplicate this thesis when needed for research and/or scholarship.

Agreed _______________________________________
Stephanie Joy Clark

Refused ______________________________________
Stephanie Joy Clark

ACKNOWLEDGMENTS

This thesis would not have been possible if it was not for the help and support of professors, staff, friends and family. I am truly indebted and thankful for the entire Mechanical Engineering facility that through my undergraduate and graduate education has taught me so much. A special thanks to Debbie Haynes; without her watching out for me over past seven years I am almost certain I would have missed several deadlines and not made half the corporate contacts that I have today. It is with great pleasure that I thank Dr. Roe and Dr. Couvillion. They both have taught me so much in undergraduate and graduate coursework. In everything that I accomplished for them they had me striving to do my best. It was my honor to have them as my thesis committee board. The guidance, experience, and knowledge that Dr. Huang has given me through my graduate research are immeasurable and I especially thank him for the time he has done spent me over the past two years.

I would like to thank all my family and friends for all their prayers, encouragement, and strength that has been given me throughout the years. I know that without all these things I would not be where I am today.

DEDICATIONS

This thesis is dedicated to my father, who spent so many years allowing me to tinker as a child and always encouraged my creativity. It is also dedicated to my mother, who has spent many years helping me to understand concepts that were just beyond my reach and for being my strength during the stressful times. Lastly, this thesis is dedicated to my Lord, who has given me everything I need, including the strength to finish this thesis.

TABLE OF CONTENTS

FIGURES AND TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

Symbols

Variable	Description	Units
F	Force	N
F_{Total}	Total Force experienced by orbiting object	N
F_{other}	Perturbing Forces	N
F_{drag}	Drag Force experienced by orbiting object	N
$F_{Solar\ Pressure}$	Solar Pressure Force experienced by orbiting object	N
F_{Thrust}	Thrust Force experienced by orbiting object	N
F_{Lift}	Lift Force experienced by orbiting object	N
F_g	Gravitational Force	N
$\boldsymbol{a}$	Acceleration vector in Cartesian Coordinates	km/s^2
a	Magnitude of acceleration vector	km/ s^2
$\ddot{\boldsymbol{r}}_{\boldsymbol{\imath}}$	Acceleration experienced by orbiting object	km/s^2
$\dot{\boldsymbol{r}}_{\boldsymbol{\imath}}$	Velocity Vector of an orbiting object	km/s
$\boldsymbol{r}$	Radius Position Vector of an orbiting object	km
r	Magnitude of radius vector	km
$\boldsymbol{V_{rel}}$	Relative velocity vector	km/s
v_{rel}	Magnitude of velocity relative to rotating Earth	km/s
$\dot{m}_\imath$	Change in mass per unit time	kg/s
G	Universal Gravitational Constant	km^3/kg-s^2
M	Mass of the Earth	kg
m	Mass of the orbiting object	kg
C_D	Coefficient of Drag	N/A
A	Surface Area of Cube Satellite	km^2
ρ	Density of Atmosphere varies with altitude	kg/m^3
ω_e	Earth Angular Velocity Vector	rad/s
$\Delta\upsilon$	Applied change in velocity	km/s

Abbreviations

LEO	Low Earth Orbit
R.K.	Runge-Kutta
Cubesat	Cube Satellite
FPA	Flight Path Angle
AOA	Angle of Attack

I. INTRODUCTION

This thesis was designed and written to assist in the understanding of the reasoning and production of a computer-based orbital model. The first chapter of this thesis will introduce topics that contribute to this research as a whole. The second chapter will discuss the theoretical concepts that were necessary for the construction of the computational model. The third chapter outlines the specifics that were needed to apply the theoretical concepts to a computational system. The fourth chapter illustrates the construction of each model that ended with the gravity, drag and lift model, and the results to the study of the effect of aeroassisted maneuvers on inclination. The fifth chapter specifies what is still to be done to increase the accuracy of the constructed model and the future areas of research that have become available because of this research. The sixth chapter analyzes the results form chapter four. The MATLAB codes that were constructed and used in this research can be found in Appendix A.

a. Objectives

This research will develop a computer model that allows for the investigation of using perturbing forces from aerodynamic drag/lift maneuvers to demonstrate useful orbital maneuvers. In addition, this research will provide insight to noncoplanar aeroassisted maneuvers based on the scaling effects of size.

b. Satellites

Minisatellies (any conventional spacecraft that can be launched from a Space Launch Vehicle (SLV)) were made useful by the continual decrease of size and mass, and according to Helvajian, if this trend continued more applications would become possible [1]. Consequently, as this trend continued different classifications were assigned to satellites based on their reduced mass. Table 1 shows the classifications that have been created based on overall weight.

Table 1: Small Satellite Classification by Mass [2]

Spacecraft Classification	Mass Range
Microsatellite	10-1000 kg
Nanosatellite	1-10 kg
Picosatellite	0.1-1 kg
Femtosatellite	0.01-0.1kg
Attosatellites	0.001-0.01 kg
Xenosatellites	1-10 mg

The Cube-Satellite (cubesat) program was started by California Polytechnic State University and Stanford University in 1999. From this program, a standard for cubesats has been put into place to regulate any future research or development within the field. In order to meet the standard, a cubesat should have the dimensions of 100mm × 100mm × 113.5mm and weigh no more than 1.33 kilograms [2]. Overall, the foremost advantages of the cubesat are the structural standardization created by the standard and the cost of launch [3]. Cubesats, the main focus application of this research, are considered nanosatellites (nanosats) by weight. However, associated cost comparison between nanosats and cubesats has been reduced to allow for availability to research groups of university students [3]. Cubesats now have the possibility to launch as secondary payloads and only pay a "piggyback" fee of $40,000 [4].

Despite the availability and cost advantages that come with cubesats, one disadvantage is the absence of propulsion. Propulsion system (currently non-existent in cubesats) propellant tanks are generally expected to weigh approximately 30 percent of the entire mass of a nanosat. Furthermore, propulsions systems do not possess the ability to be easily downsized [5]. Thus, cubesats are currently restricted to a coasting flight in Low Earth Orbit (LEO) due to a lack of propulsive or orbital maneuvering resources [6].

c. Classical Orbital Elements

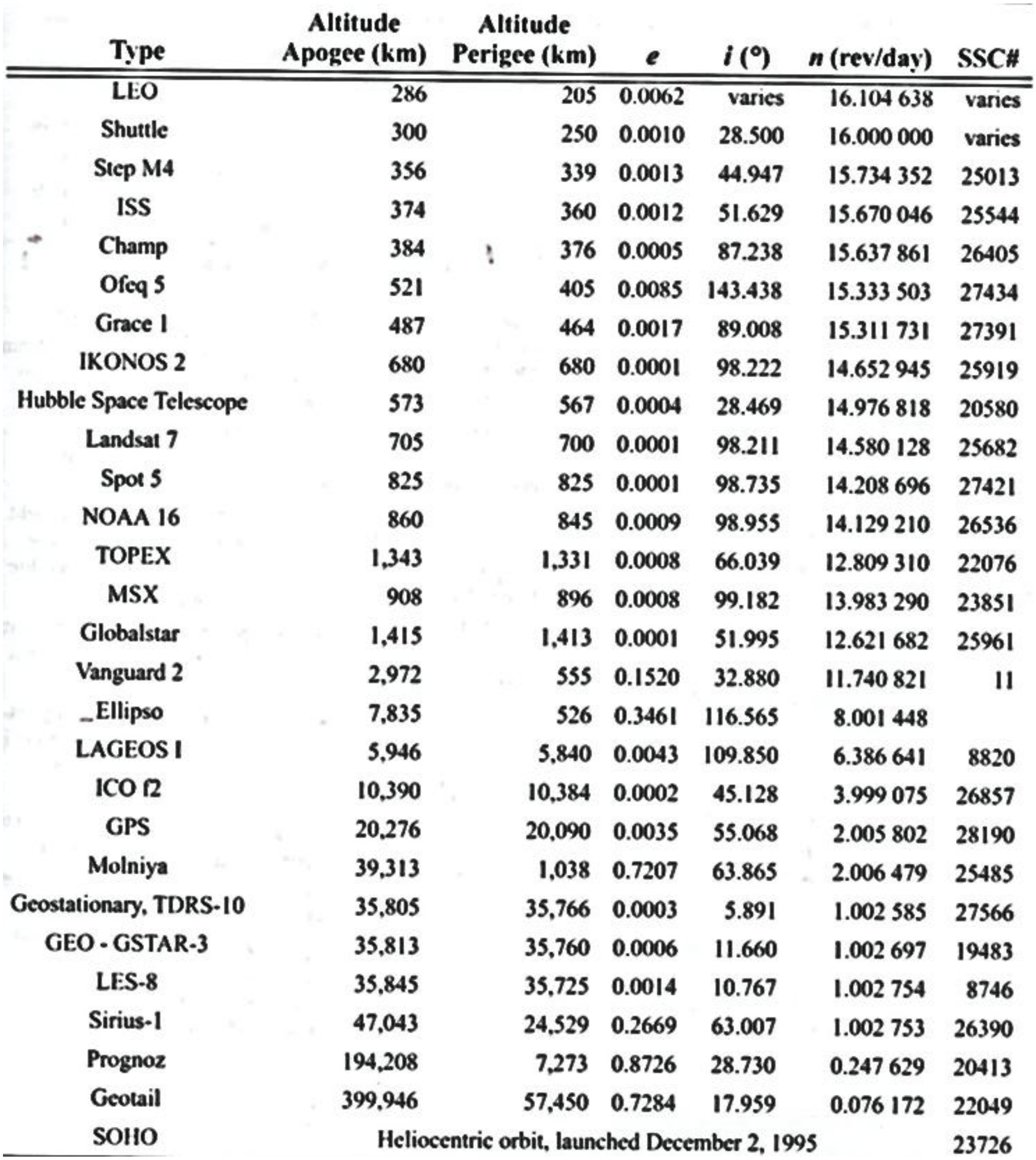
Classical Orbital Elements, also known as Keplerian elements, assist in describing the shape and size of an orbit [7]. There are six components that represent the classical orbital elements: semimajor axis (*a*), eccentricity (*e*), inclination (*i*), right ascension of the ascending node *(Ω)*, argument of periapsis *(ω)*, and true anomaly *(θ)*. [8] The semimajor axis is the mean value of the minimum (perigee) and maximum (apogee) distance. Eccentricity is a measure of the orbits difference from a circle. Orbit shapes are described by this value: Circle (e = 0), Ellipse (0 < e < 1), Parabola (e = 1), Hyperbola (1< e). The true anomaly is the angle between the reference direction and the current position vector [9]. The inclination is the angle of intersection between the plane of orbit and the equatorial plane. The right ascension of the ascending node is the angle between the vernal equinox and the location in which the satellite passes the equator from south to north during its orbit. The vernal equinox describes the direction in which the Sun is seen from the Earth. The argument of perigee is the angle between the direction of the ascending node and the direction of perigee [9].

These six elements are often used algebraically to solve limited but useful cases of orbital parameters instead of the equation of motion (EOM). However, the widespread availability of computers today makes it practical to solve the EOM differential equations [8]. Thus, for this research, the position and velocity are used over the Classical Orbital Elements. This was preferable over the six elements because at times the elements can become awkward due to their scalar representation. For example, the semimajor axis is undefined for parabolic orbits, the argument of periapsis is "ill defined" for almost circular orbits, and the right ascension of ascending node is also ill defined for orbits with zero inclination [8]. In addition, no ephemeral data is used for the research described here since the focus is on potential benefits of aeroassisted cubesats rather than specific mission parameters (i.e., launch dates and ground coverage).

d. Orbit Types

When referring to the orbit of an object around the Earth, several types of orbits in which the object can be launched into exist and are commonly categorized by the period (the amount of time required to make one orbit around the Earth). The five main types of orbit categories are geosynchronous, geostationary, semi-synchronous, super-synchronous, and deep space. The geosynchronous orbit has a period equal to the rotation of the Earth (approximate altitude of 35,780 km); geostationary remains over location above the Earth; semi-synchronous have an orbit of 12 hours; super-synchronous orbits have a period greater than 24 hours; and deep space orbits are near semi-synchronous altitudes. Categories that reference orbit types by altitude include LEO, which refers to an orbit less than 800 km and mid-Earth orbits (MEO), which is the altitude range of 800 km to 30,000 km [7]. There are many more orbit classifications that exist, oftentimes associated with a particular historic or currently in use missions, and are referenced in Table 2.

Table 2: Satellite Orbit Classifications [7]



Type	Altitude Apogee (km)	Altitude Perigee (km)	*e*	*i* (°)	*n* (rev/day)	SSC#
LEO	286	205	0.0062	varies	16.104 638	varies
Shuttle	300	250	0.0010	28.500	16.000 000	varies
Step M4	356	339	0.0013	44.947	15.734 352	25013
ISS	374	360	0.0012	51.629	15.670 046	25544
Champ	384	376	0.0005	87.238	15.637 861	26405
Ofeq 5	521	405	0.0085	143.438	15.333 503	27434
Grace 1	487	464	0.0017	89.008	15.311 731	27391
IKONOS 2	680	680	0.0001	98.222	14.652 945	25919
Hubble Space Telescope	573	567	0.0004	28.469	14.976 818	20580
Landsat 7	705	700	0.0001	98.211	14.580 128	25682
Spot 5	825	825	0.0001	98.735	14.208 696	27421
NOAA 16	860	845	0.0009	98.955	14.129 210	26536
TOPEX	1,343	1,331	0.0008	66.039	12.809 310	22076
MSX	908	896	0.0008	99.182	13.983 290	23851
Globalstar	1,415	1,413	0.0001	51.995	12.621 682	25961
Vanguard 2	2,972	555	0.1520	32.880	11.740 821	11
_Ellipso	7,835	526	0.3461	116.565	8.001 448	
LAGEOS I	5,946	5,840	0.0043	109.850	6.386 641	8820
ICO f2	10,390	10,384	0.0002	45.128	3.999 075	26857
GPS	20,276	20,090	0.0035	55.068	2.005 802	28190
Molniya	39,313	1,038	0.7207	63.865	2.006 479	25485
Geostationary, TDRS-10	35,805	35,766	0.0003	5.891	1.002 585	27566
GEO - GSTAR-3	35,813	35,760	0.0006	11.660	1.002 697	19483
LES-8	35,845	35,725	0.0014	10.767	1.002 754	8746
Sirius-1	47,043	24,529	0.2669	63.007	1.002 753	26390
Prognoz	194,208	7,273	0.8726	28.730	0.247 629	20413
Geotail	399,946	57,450	0.7284	17.959	0.076 172	22049
SOHO	Heliocentric orbit, launched December 2, 1995					23726

e. Maneuver Concept

Despite the increase in the desire to have nanosats perform high Δυ maneuvers, smaller satellites, historically, have to overcome insufficient or no propulsion systems due to the need to keep development costs low [10]. Large Δυ (noncoplanar maneuvers) are expected to be “high cost” in comparison with coplanar maneuvers [11]; therefore, this research will be focused mainly on the research of noncoplanar maneuvers. Noncoplanar maneuvers affect two aspects of an orbit: “inclination, i, and the right ascension of the ascending node, Ω” [7]. Currently, a change in orbital inclination is performed by firing propulsion thrusters normal to the velocity vector and therefore increasing the inclination (see Figure 1).

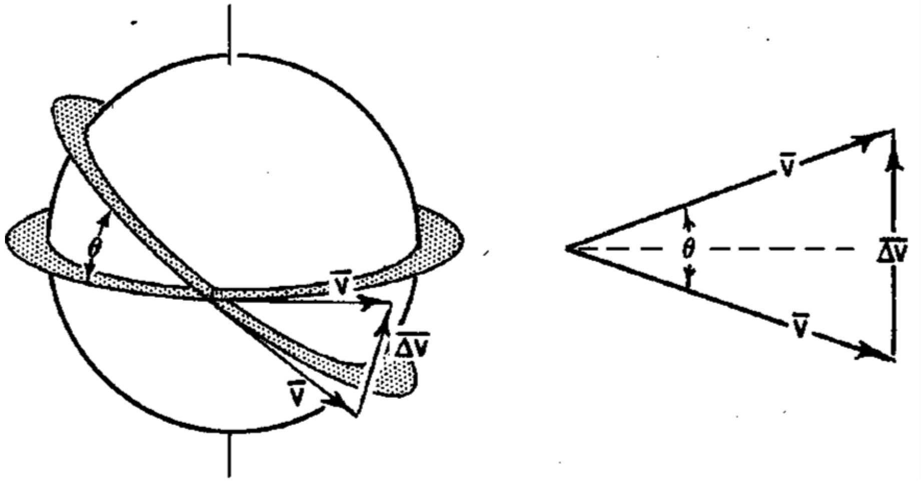


Figure 1: Simple Noncoplanar maneuver [12]

In general, noncoplanar maneuvers require larger amounts of propellant compared to planar maneuvers because the Δυ required is on the magnitude of the satellite velocity (approx. 7 to 8 $\frac{km}{s}$ for earth-based orbits). Even for large satellite missions, when inclination changes approach 60 degrees, they begin to become prohibitively expensive [12]. Noncoplanar orbital maneuvers are desirable for three reasons. First, the site from which the satellite was launched restricts the range of inclination of the orbit. Second, specific orientations can be preferable in order to reduce the costs in orbit transfer propulsion from the launch vehicle upper stages. Finally, timing constraints may dictate a specific launch window which is not desirable. All of the above become even more restricting for cubesats due to their “piggy backing” nature of launch operations.

The orbital maneuver researched here is used to adjust only the orbital inclination using lift forces caused by atmospheric density and satellite design. A side effect of this maneuver is the reduction of the planar orbit due to the drag force experience from the satellite design. Thus, from the perspective of orbital (mission) life-times, the key is to operate under maximized lift-to-drag ratio conditions much like typical aero-vehicles.

f. Perturbing Forces

Forces that cause “deviations from a normal, idealized, or undisturbed motion” are called perturbing forces. Three techniques exist when solving for the effect of perturbing forces: analytical, numerical, and semi-analytical; out of these, numerical solutions are the most accurate and flexible. The typical perturbing forces considered with orbit modeling are “asphericity of the central body, atmospheric drag and lift, third body effects, solar-radiation pressure, thrust, magnetic fields, solid-Earth and ocean tides, Earth re-radiation (albedo), and relativistic effects.” These forces are not always small, but can be comparable to attracting (gravity) forces [7]. The two main forces focused on during the development of this model are drag and lift. Drag Force affects the semi-major axis and the eccentricity characteristics of an orbit, but the main affect is the reduction in orbit (as seen in Figure 2) and eventual re-entry or impact with the Earth [7].

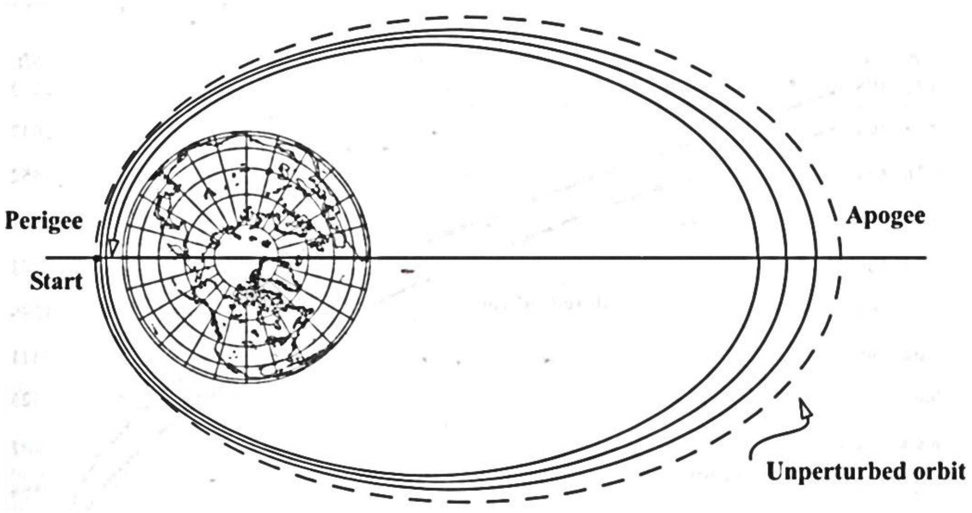


Figure 2: Change in Orbital Altitude due to Drag [7]

When modeling a non-spherical object effects due to lift and side forces must also be considered [7]. Noncoplanar maneuvers cannot be controlled by drag forces but lift affects the orbit in a direction normal to the drag vector. It is apparent that lift forces are considerable and should be useful to create small orbital adjustments within LEO [13].

g. Satellite Stabilization

Cubesats have begun to approach attitude control capabilities. There are two types of attitude controls: passive and active techniques. Passive attitude control methods include "magnets, gravity gradients, and spin stabilization." Magnets are used in near equatorial orbits and spin stabilization is caused by gyroscopic stiffness to resist torque on two axes. Active attitude control methods include "thrusters, magnetic torquers, and reaction wheels." However, thrusters require propulsions systems which are not easily downsized, and reaction wheels are relatively large and often require too much power for continual operations on cubesats [4]. Under aeroassisted conditions, it is important to assume that the satellite has reached a stabilized condition because a spinning satellite will cancel out any effects due to lift forces due to averaging [13]

h. Models

Determining the orbit of small objects accurately is fundamental today “because objects as small as flecks of paint have damaged the Space Shuttle’s windows.” Solving orbit motion is reliant on estimation, which is connected to initial orbit determination, prediction, and uncertainty estimates [7]. One main source of error (when using computational models) is the imperfections that exist within gravity and atmospheric models. In addition, lacking the ability to model the ballistic coefficient when in LEO can cause large errors during modeling [7]. These concerns were by-passed in this research due to the focus on clarifying the effects of aeroassist on cubesats rather than providing for mission planning with the resulting data.

i. Atmospheric Models

Several different atmospheric models have been created to determine the density of the atmosphere at changing altitudes. These models are typically developed in one of two ways: first, “by combining conservation laws and atmospheric-constituent models into a physical model” and second by simplifying physical theories developed from measurements and satellite tracking [7]. Since most cubesats orbit earth at a LEO their lifespans are restricted due to atmospheric drag [6]. Therefore, in order to model cubesat orbits accurately, an accurate atmospheric model must be used [7]. However, not even the best theories can model the atmosphere exactly [7]. Figure 3 (on the next page) shows the complexity of several atmospheric model evolutions.

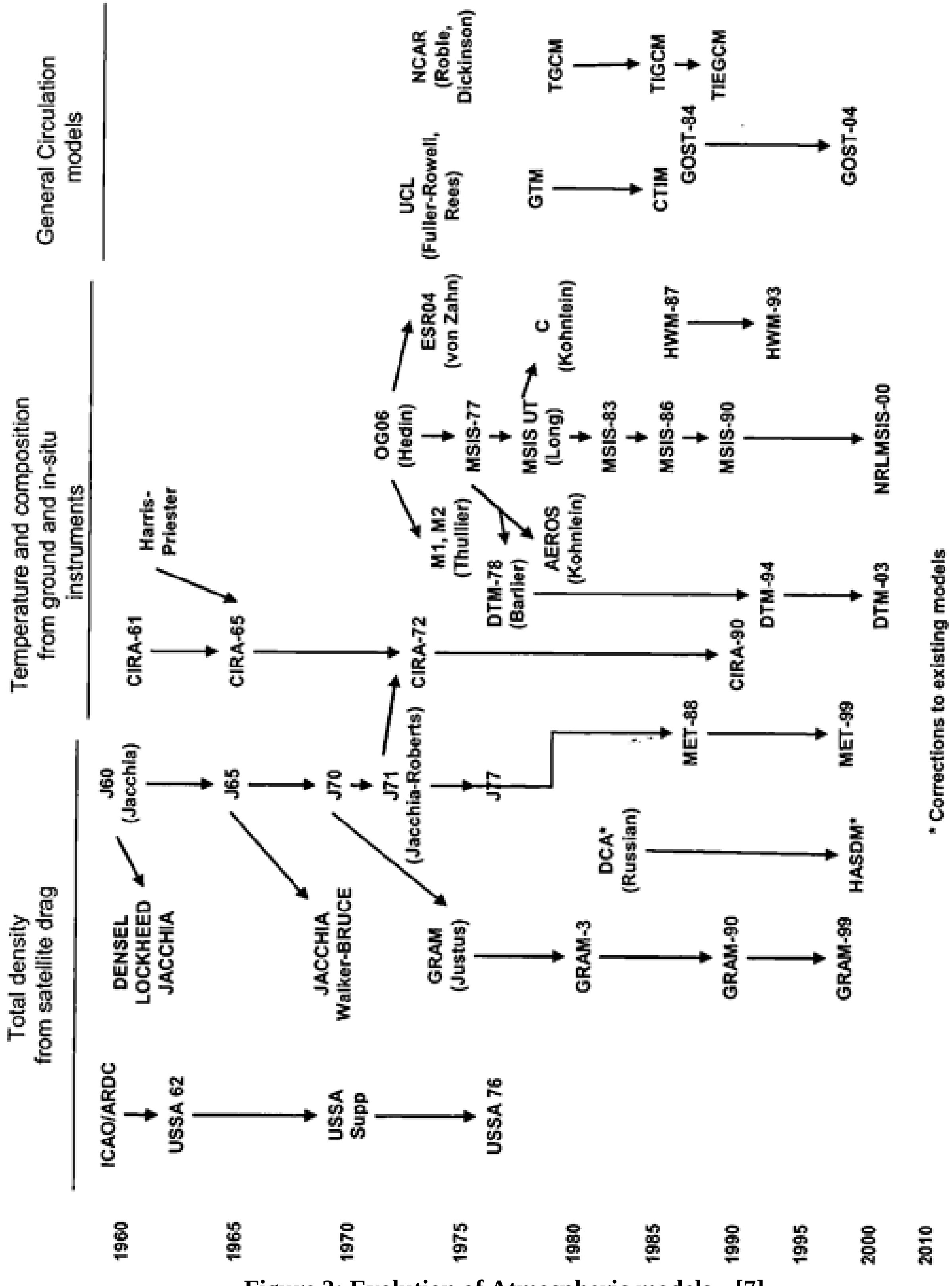
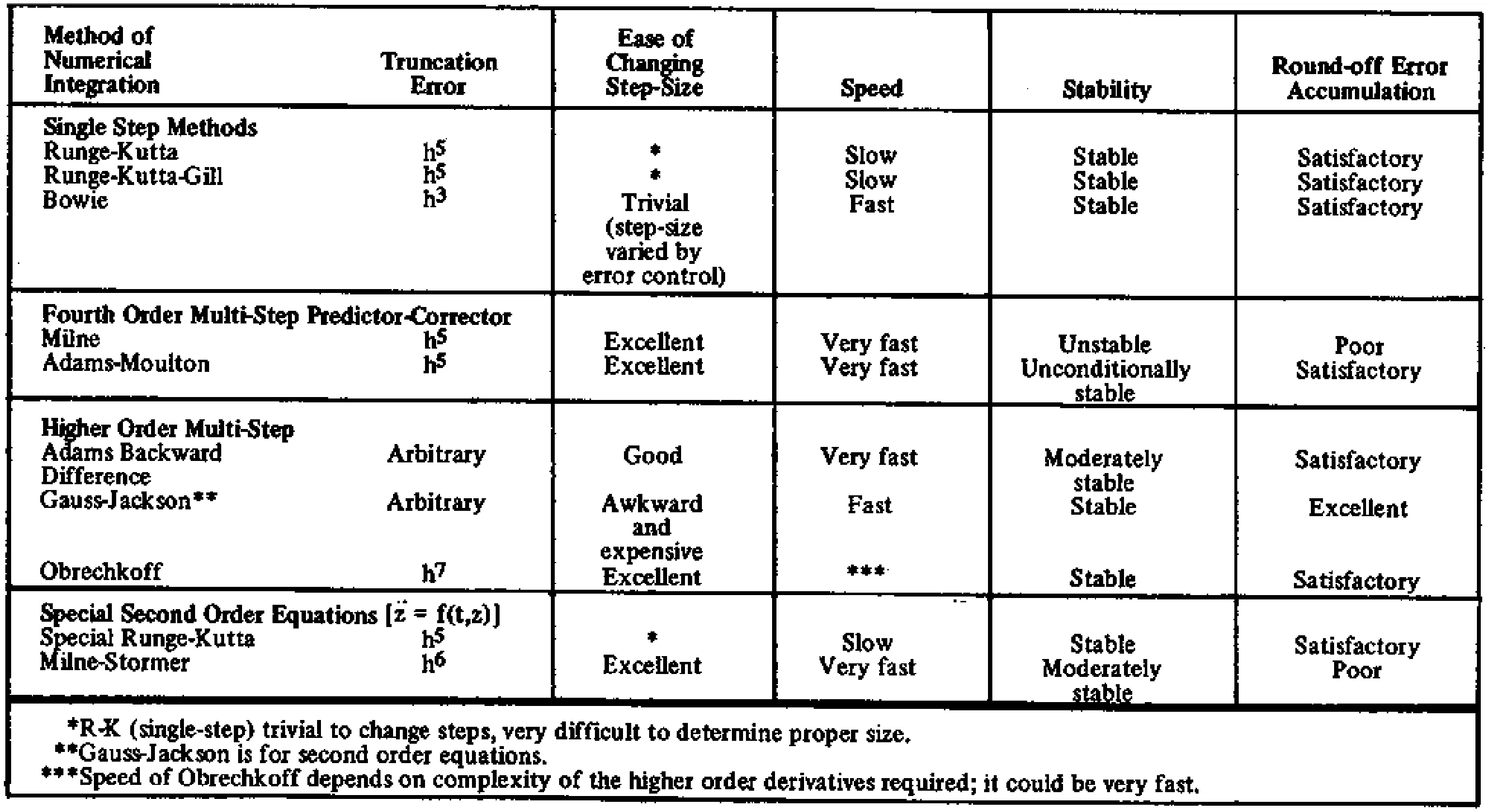


Figure 3: Evolution of Atmospheric models [7]

j. Numerical Integration Methods

When it comes to selecting the right method to numerically integrate differential equations, a variety of methods have to be researched. A few factors to be taken into consideration are speed, accuracy, storage, and complexity. A quality numerical solver should possess the following characteristics: largest step-size possible, quick and simple provisions for variable step size, low computational costs, stable, avoids exponential error growth, minimal rounding errors, and a bound on truncation error. However, not one method can possess all of the above qualities but they can be used in making a decision regarding which method to use [12]. Table 3 (on the next page) compares several methods with respect to the consideration factors discussed above.

Table 3: Numerical Integration Type Comparison [12]



Method of Numerical Integration	Truncation Error	Ease of Changing Step-Size	Speed	Stability	Round-off Error Accumulation
Single Step Methods					
Runge-Kutta	h^5	*	Slow	Stable	Satisfactory
Runge-Kutta-Gill	h^5	*	Slow	Stable	Satisfactory
Bowie	h^3	Trivial (step-size varied by error control)	Fast	Stable	Satisfactory
Fourth Order Multi-Step Predictor-Corrector					
Milne	h^5	Excellent	Very fast	Unstable	Poor
Adams-Moulton	h^5	Excellent	Very fast	Unconditionally stable	Satisfactory
Higher Order Multi-Step					
Adams Backward Difference	Arbitrary	Good	Very fast	Moderately stable	Satisfactory
Gauss-Jackson**	Arbitrary	Awkward and expensive	Fast	Stable	Excellent
Obrechkoff	h^7	Excellent	***	Stable	Satisfactory
Special Second Order Equations [$\ddot{z} = f(t,z)$]					
Special Runge-Kutta	h^5	*	Slow	Stable	Satisfactory
Milne-Stormer	h^6	Excellent	Very fast	Moderately stable	Poor

*R-K (single-step) trivial to change steps, very difficult to determine proper size.

**Gauss-Jackson is for second order equations.

***Speed of Obrechkoff depends on complexity of the higher order derivatives required; it could be very fast.

There are two main types of integration methods when it comes to iteration steps: "*single-step* and *multi-step*". Single-step methods allow the determination of a state at ensuing times $(t_o + h)$. A common method used for single-step integration is the fourth-order Runge-Kutta method (RK4). Runge-Kutta (R.K.) solvers are superior because these methods do not involve a "sequence of back values to start the integrator" and keep computation costs low; furthermore, R.K. methods solve for the next state by evaluating several slopes at several "next states" based on a differing values for $t_o + h$, and then calculating a weighted average of those values. Figure 4 helps to illustrate how RK methods solve the next iteration [7].

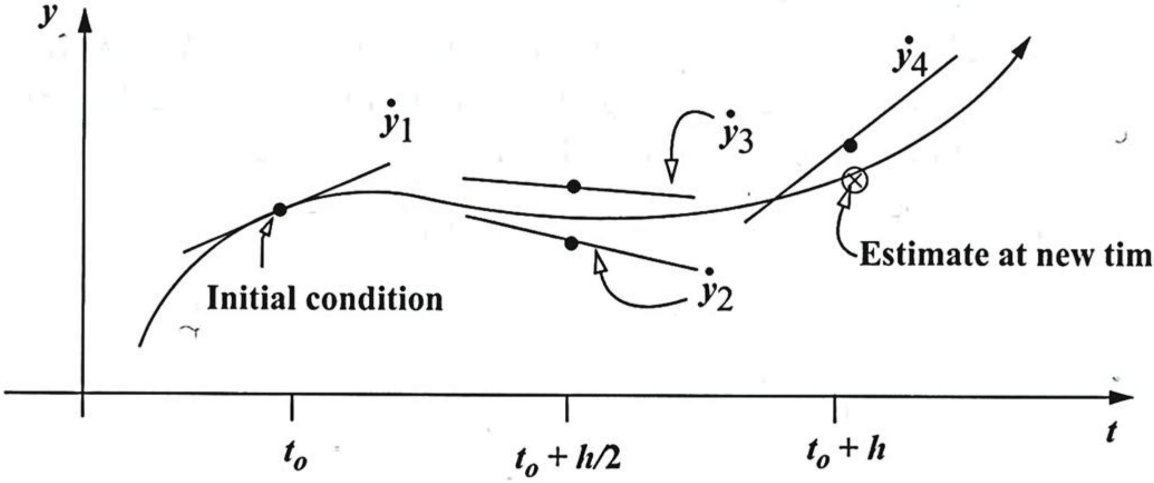


Figure 4: Illustration of next state calculation using RK4 [7]

Despite the advantages that R.K. methods offer, a major downfall to R.K. methods derives from the fact that they only use each state once, bringing the use of multi-step methods into use. There are two types of multi-step methods, explicit and implicit. Explicit methods solve for the next state using only previously known or solved states. An example of an explicit method is the family of Adams-Bashforth methods. Implicit methods solve for the next state using previously states and next state values. An example of an implicit method is the family of Adams-Moulton methods. Derived from the combination of an explicit and implicit method are the predictor-corrector methods [14]. These methods use an “initial estimate using previous estimate of the functions rate of change, and a second series…to further refine the result” [7].

While multi-step methods are typically more accurate, they require the storage of back values and are not self-starting [7]. A common practice that is employed in order to utilize these multi-step methods is numerical integration using a single step method to start the integration process and then switching to a multi-step method with the back values in tow. However, the order or error of the single-step method and the multi-step method must match. Therefore, if a high order RK10 method is being used then a 7th order Adams-Moulton or a 8th order Adams-Bashforth method must be utilized [14]. Another method (not discussed in depth) is Gauss-Jackson, which is one of the most used methods involved with trajectory problems [12].

II. CONCEPT

a. Relative Motion

The acceleration experienced by an orbiting object can be found by analyzing Newton's second law of motion, $F = ma$; the acceleration experienced by a satellite will be denoted as $\ddot{r}$ [12]. Rectangular or Cartesian coordinates are typical in the field of astrodynamics so all position, velocity, and acceleration vectors will be expressed as such [12].

The total force acting on an orbiting object is the sum of gravitation and perturbing forces.

$$F_{Total} = F_g + F_{other} \qquad \text{Eq. 1}$$

$$\begin{aligned} \text{where } F_{Total} &= Total\ Force \\ F_g &= Gravitation\ Force \\ F_{other} &= Perturbing\ Force \end{aligned}$$

The perturbing force is equal to the sum of the forces created by drag, lift, thrust, solar pressure, etc. [12]. Normally, these forces are not modeled due to their relatively small size compared to gravity [15]. Therefore, for the moment, the acceleration will be discussed without perturbing forces, but brought back into the discussion later in this section.

Newton's second law of motion can be written as a time derivative of mass and velocity from $F = ma$ and is expressed in Eq. 2.

$$\frac{d(mv)}{dt} = F_{Total} \qquad \text{Eq. 2}$$

$$\begin{aligned} \text{where } \tfrac{d}{dt} &= time\ derivative \\ m &= mass\ of\ the\ object \\ v &= velocity\ of\ the\ object \end{aligned}$$

If this time derivative were rearranged algebraically, the acceleration experienced would appear as Eq. 3.

$$\ddot{r}_\imath = \frac{F_{Total}}{m_i} r - \dot{r}_\imath \frac{\dot{m}_\imath}{m_i} \qquad \text{Eq. 3}$$

where $\ddot{r}_\imath = accleration\ of\ mass$

$m_i = mass$

$r = radius\ position\ of\ mass$

$\dot{r}_\imath = velocity\ of\ mass$

$\dot{m}_\imath = change\ in\ mass\ with\ respect\ to\ time$

However, the change in mass with respect to time,$\dot{m}_\imath$, is assumed to be zero because for propellant is not expected in this research. From Eq. 1, the gravitational force is Newton’s law of universal gravitation (which is represented by Eq. 4).

$$\boldsymbol{F_g} = \frac{GMm}{r^2}\boldsymbol{\hat{r}} \qquad \text{Eq. 4}$$

where $G = Universal\ Gravitation\ constant$

$M = Mass\ of\ the\ Earth$

$m = mass\ of\ the\ satellite$

$r = magnitude\ of\ the\ radius\ vector$

$\boldsymbol{\hat{r}} = unit\ radius\ vector$

In Eq. 4, GM is also known as μ (the gravitation parameter) which for Earth is the constant value of 398,600.4 $\frac{km^3}{s^2}$ and $\boldsymbol{\hat{r}}$ is equal to $\frac{\hat{r}}{r}$ [16]. All other gravitation effects from the other surrounding bodies are considered negligible compared to the gravitational effect from the body the cubesat is orbiting around and is not modeled. Eq. 3 represents the acceleration due to multiple bodies, but since all other effects are considered negligible in the present model the equation describing the acceleration experienced by a cubesat is thus the following Eq. 5 [12]:

$$\boldsymbol{\ddot{r}} = \frac{\mu}{r^3}\boldsymbol{r} \qquad \text{Eq. 5}$$

where $\boldsymbol{\ddot{r}} = acceleration\ vector$

$\mu = Gravitational\ Parameter$

However, Eq.5 only models gravity and does not take into account the effects of atmospheric density and perturbing forces. Without accurately modeling perturbing forces an accurate model of satellite motion could not be created.

As an orbiting object begins to reach lower altitudes density increases and drag and lift are no longer negligible. Therefore, these forces must be taken into account and modeled. The force due to drag is generically represented by Eq. 6.

$$\boldsymbol{F_{drag}} = \frac{C_D A \rho {V_{rel}}^2}{2} \boldsymbol{e_v} \qquad \text{Eq. 6}$$

where $\boldsymbol{F_{drag}} = Drag\ force$

$C_D = Coefficient\ of\ Drag$

$A = surface\ area\ of\ satellite$

$\rho = Density$

$V_{rel} = Relative\ Velocity$

$\boldsymbol{e_v} = Drag\ Force\ Vector$

Combining drag and gravity, the acceleration effects in Eq. 5 is then represented by Eq. 7 [17].

$$\ddot{\boldsymbol{r}} = \frac{GM}{r^3}\boldsymbol{r} + \frac{C_D A \rho {v_{rel}}^2}{2m} \boldsymbol{e_v} \qquad \text{Eq. 7}$$

In order to take into account the effects of lift, the equation for lift forces is represented by Eq. 8.

$$\boldsymbol{F_{lift}} = \frac{C_L A \rho {V_{rel}}^2}{2} \boldsymbol{v_{Lift}} \qquad \text{Eq. 8}$$

where $\boldsymbol{F_{lift}} = Lift\ Force$

$C_L = Coefficient\ of\ Lift$

$\boldsymbol{v_{Lift}} = Lift\ Vector$

In Eq. 8, the vector $\boldsymbol{v_{lift}}$ can be found by taking the vector for drag force and scaling it by its magnitude, $\frac{\boldsymbol{F_{Lift}}}{F_{Lift}}$. It is then observed that $\boldsymbol{v_{lift}} = (\boldsymbol{v}_{rel} \times \boldsymbol{n}) \times \boldsymbol{v}_{rel}$ where **n** is the vector describing the orientation of the satellite and $\boldsymbol{v}_{rel}$ is the relative velocity vector [13]. Therefore, Eq. 9 models relative motion taking into account gravity, drag, and lift.

$$\ddot{\boldsymbol{r}} = \frac{GM}{r^3}\boldsymbol{r} + \frac{C_D A \rho {v_{rel}}^2}{2m}\boldsymbol{e}_v + \frac{C_L A \rho {v_{rel}}^2}{2m}\boldsymbol{v_{Lift}} \qquad \text{Eq. 9}$$

b. Coordinate Frames

i. Earth, Body, and Wind Axis

When modeling an orbiting object with respect to the Earth, three coordinate frames and six angles of rotation that must be defined. The first coordinate system is the inertial or basis frame. For an object orbiting the Earth this frame is the Earth-Centered coordinate frame and is depicted in Figure 5.

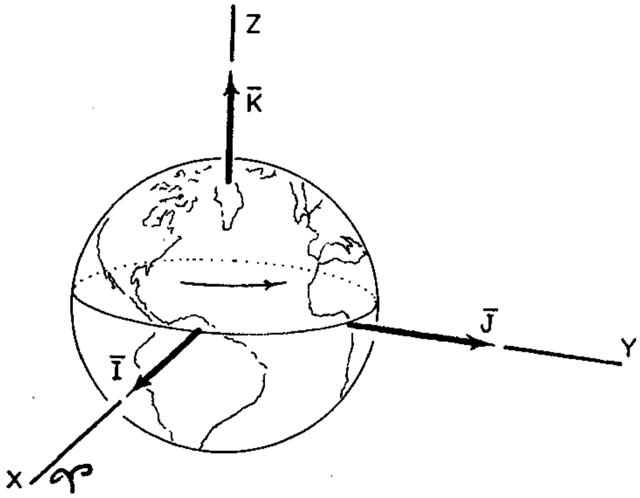


Figure 5: Earth-Centered Coordinate Frame [12]

The second coordinate system is referred to as the Body-Axis coordinate frame. This coordinate frame takes into account rotation of the satellite with respect to the Earth-Centered coordinate frame. The angle θ represents the rotation about the x–axis which rotates the yz–plane. Likewise, the angle γ represents the rotation about the y–axis which rotates the xz–plane. The angle η represents the rotation about the z–axis which rotates the xy–plane.

The third coordinate system is referred to as the Wind–Axis coordinate frame. This coordinate frame takes into account the angles at which the air and the satellite interact. Within this frame, the angle α represents the rotation of the about the x–axis, which rotates the yz–plane, while the angle β represents the rotation about the y–axis, which rotates the xz–plane. The angle δ represents the rotation about the z–axis, which rotates the xy–plane.

ii. Coordinate Transformation

Multiple reference coordinate frames must be used when creating a model of an orbiting object because drag and lift forces must be calculated in reference to the wind, while gravity is calculated in reference to the Earth. Therefore, both lift and drag forces require calculation in the Wind Axis before being transformed back to Earth-Centered Axis. Once back in the Earth-Centered frame the acceleration due to drag and lift is added to the gravity acceleration that is calculated [15].

When comparing vectors before and after coordinate transformation "in any coordinate frame," both vectors will still possess the same magnitude and direction [12]. There are two types of coordinate transformations: simple and complex. Simple coordinate transformations rotate about one of the primary axes (x, y, or z) of the original frame and results in a new coordinate system denoted as $x'y'z'$. Complex coordinate transformations rotate around two or more primary axes in order to define the new coordinate system. Table 4 shows the transformation matrices that are used to describe the rotation about any of the three axes.

Table 4: Matrix Transformations about one axis

x-axis rotation	y-axis rotation	z-axis rotation
$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & sin\alpha \\ 0 & -sin\alpha & cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$	$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$	$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} cos\gamma & sin\gamma & 0 \\ -sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

When completing complex transformations, it is important to realize that matrix multiplication is not commutative, and, therefore, the correct order or matrices must be observed. When transformations are being completed, keeping track of the transformation process can be done in two ways. First, as each axis rotation is completed, a transitional vector can be formed and used to complete the next transformation as a "new" xyz, and both the transitional and transformation vectors are tracked. In order to illustrate more clearly, Eq. 10 demonstrates the process.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & sin\alpha \\ 0 & -sin\alpha & cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\alpha \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \qquad \text{Eq. 10}$$

In Eq. 10, the transitional matrix is $x'y'z'$ and $x''y''z''$ is the frame that encapsulates all the rotations. In this case the basis, transitional and final transformation matrix are tracked with time. Second, the transformation can be done in one calculation using several transformation matrices and the basis. In order to illustrate this more clearly, Eq. 11 demonstrates the process [12].

$$\begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\alpha \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & sin\alpha \\ 0 & -sin\alpha & cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \text{Eq. 11}$$

In Eq. 11, xyz represents the basis and $x''y''z''$ represents the final transformation vector. In this case, only the basis and the final transformation vector are tracked with time.

c. Coefficients

i. Ballistic Coefficient

The ballistic coefficient is defined as $\frac{m}{C_d A}$, and it is derived from the equation for the acceleration due to drag, $a_D = \frac{\rho A {v_{rel}}^2 C_d}{2m}$. This coefficient measures the tendency of an object to be affected by drag. A low coefficient means that an object will be highly affected by drag, while in contrast otherwise a high coefficient means the object will only be slightly affected by drag [7].

ii. Drag and Lift Coefficient

The coefficient of drag, C_d, is, normally, widely accepted as 2.2 for satellites orbiting in the upper atmosphere (sphere are 2.0 to 2.1). This coefficient is “satellite configuration-specific” and is rarely approximated to more than three significant figures [7]. In contrast to the coefficient of drag, the coefficient of lift, C_L, is complicated to model and measure from orbital altitudes. This coefficient is usually estimated using physical models and modeling assumptions. The coefficient of drag and lift values used for this research are expressed by Eq. 12 [13].

$$\begin{gathered} C_d = 2\left[1 + \frac{2}{3}\cos(\theta)\sqrt{1 + \alpha\left(\frac{T_w}{T_a} - 1\right)}\right] \\ C_l = \frac{4}{3}\sin(\theta)\sqrt{1 + \alpha\left(\frac{T_w}{T_a} - 1\right)} \end{gathered} \qquad \text{Eq. 12}$$

These equations represent the coefficient of drag and lift for a flat plate with an incident angle [18]. A cube then can be modeled as flat plates and some modifications to the equation will be discussed later in this section.

In Eq. 12, the ratio $\frac{T_w}{T_a}$ represents that ratio of the satellite surface temperature, T_w, and the temperature of the local atmosphere, T_w [13]. However, this ratio is assumed to be unity in this research which simplifies the equation to $\sqrt{1-\alpha}$ [18].

The above equation, like stated above, represents the drag and lift coefficients for a flat plate. A few modifications were needed to more accurately represent the drag and lift for a cube, which in essence is a culmination of several flat plates. First, the orientation and rotation of a cubesat is represented in Figure 6, and the arrow represents the flight path direction.

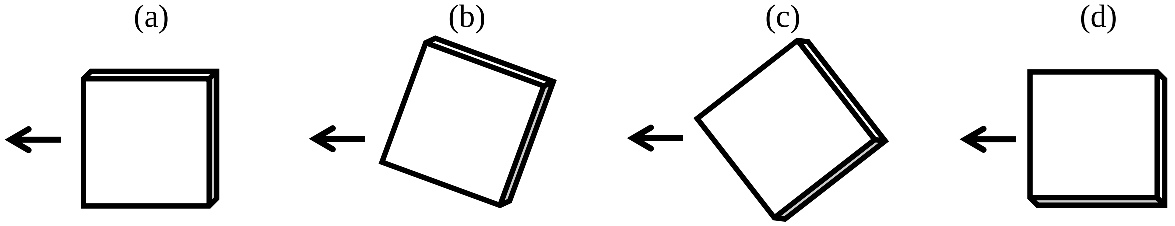


Figure 6: Orientation and Rotation of a Cubesat
(a) 0 degree (b) >45 degree (c) <45 degree (d) 90 degree incident angle

Additionally, the image (a) is the original orientation of the modeled cubesat; images (b) through (d) illustrate what is observed as a full rotation (90 degrees) is completed. It is observed that there is only drag initially and after 90 degrees of rotation. However, based on Eq. 12, at a 90 degree incident angle there exists a coefficient of lift value of $\frac{4}{3}$. Second, since there is only drag at an incidence angle of zero and 90 the coefficient of drag should be identical at either angle. However, at an incident angle of zero Eq. 12 produces a coefficient of drag of 2.13, whereas at an incident angle of zero the coefficient of drag is 2. Therefore, both equations for the coefficients of drag and lift had to be modified to fit a cube (or multiple plates). This was done with the concept of frequency. The frequency is defined as the number of profiles passing per second [19]. In our case, it is the number of complete cycles per revolution (2π). Adjustments were made to create a lift and drag equation that would fit basic assumptions of drag and lift of a cube using the frequency. The resulting expressions are illustrated by Eq. 13.

$$\begin{gathered} C_d = 2\left[1 + \frac{2}{3}\cos(4\theta)\sqrt{1-\alpha}\right] \\ C_l = \frac{4}{3}\sin(2\theta)\sqrt{1-\alpha} \end{gathered} \qquad \text{Eq. 13}$$

From Eq. 13, it can be seen that the frequency of both the drag and lift coefficients where changed. The cosine and sine wave frequency was adjusted to allow the wave to return to the original value by the completion of a 90 degree rotation.

It should be noted that the lift and drag caused by the backside of the satellite was considered negligible for these models.

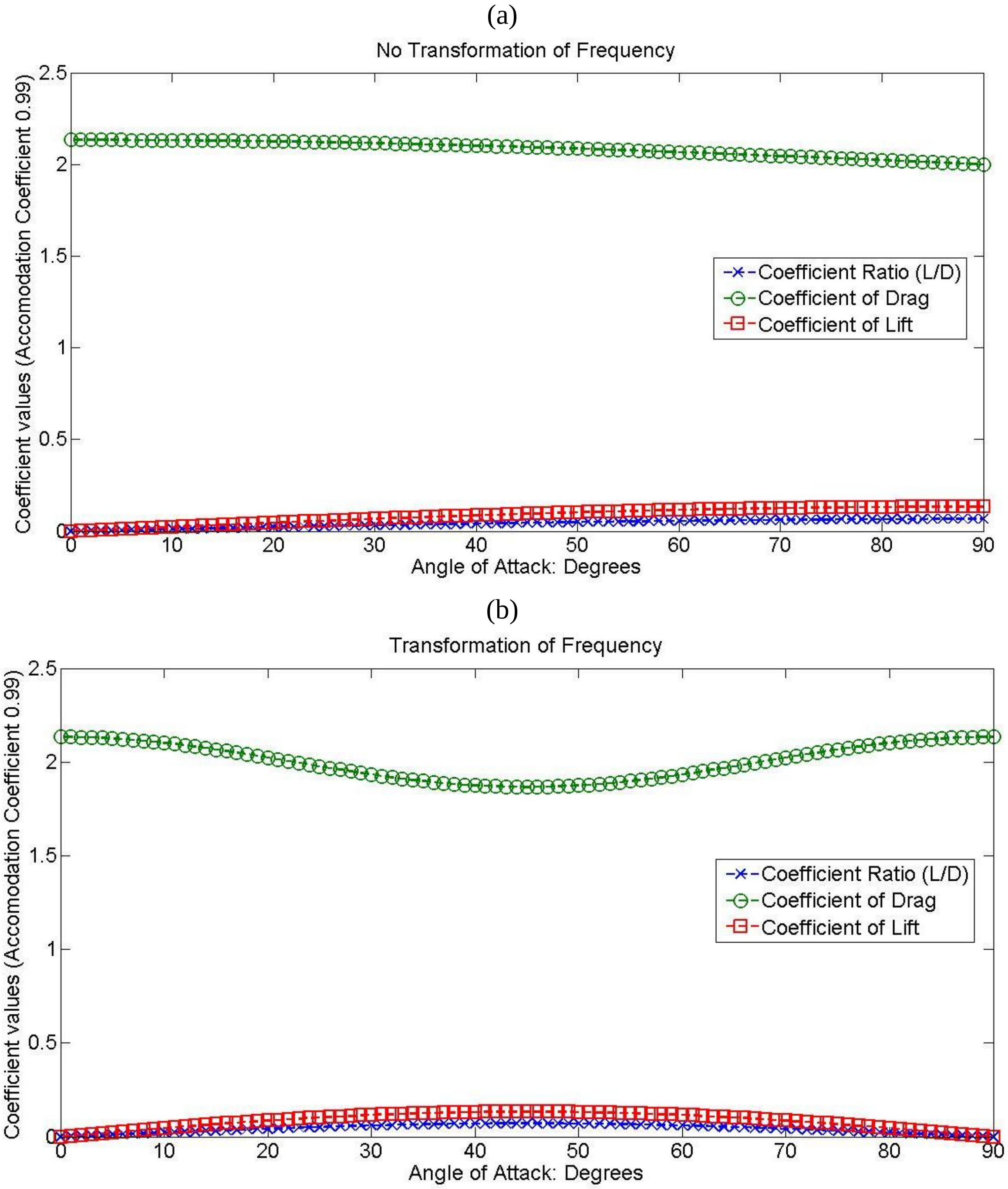


Figure 7: Coefficients with Varying Angle of Attack and Accommodation Coefficient (a) Flat Plate Equation (b) Transformed Flat Plate Equation

The ideal angle in which to perform aerodynamic lift maneuvers is at 45 degrees (observable in Figure 7) because the ratio of lift-to-drag is at peak value. However, this may not be true if mission life time is taken into consideration

i. *Accommodation Coefficient*

$$\alpha = \frac{E_{inc} - E_r}{E_{inc} - E_s} \qquad \text{Eq. 14}$$

where $\alpha = Accomodation\ Coefficient$

$E_{inc} = kinetic\ energy\ of\ a\ incident\ molecule$

$E_r = kinetic\ energy\ of\ a\ reflected\ molecule$

$E_s =$ kinetic energy that particles would have if emitted with an energy due to the satellite surface temperature.

This coefficient measures how much “memory the gas molecule retains of its initial velocity after its interaction with the satellite surface” [13]. Complete accommodation ($\alpha = 1$) denotes that the “emitted particle has a kinetic energy that has completely adjusted to the thermal energy of the surface” [13].

The accommodation coefficient affects the coefficients of both drag and lift. It can be seen in Figure 8 and Figure 9 the difference in values is significant between accommodation coefficient values of one and zero. Figure 8 shows the difference in the coefficient of drag and lift undergoing varying values of the accomodation coefficient over one rotation (0 to 90 degrees). Figure 8 also shows that the lift for a satellite becomes predominate with extremely low accommodation coefficients.

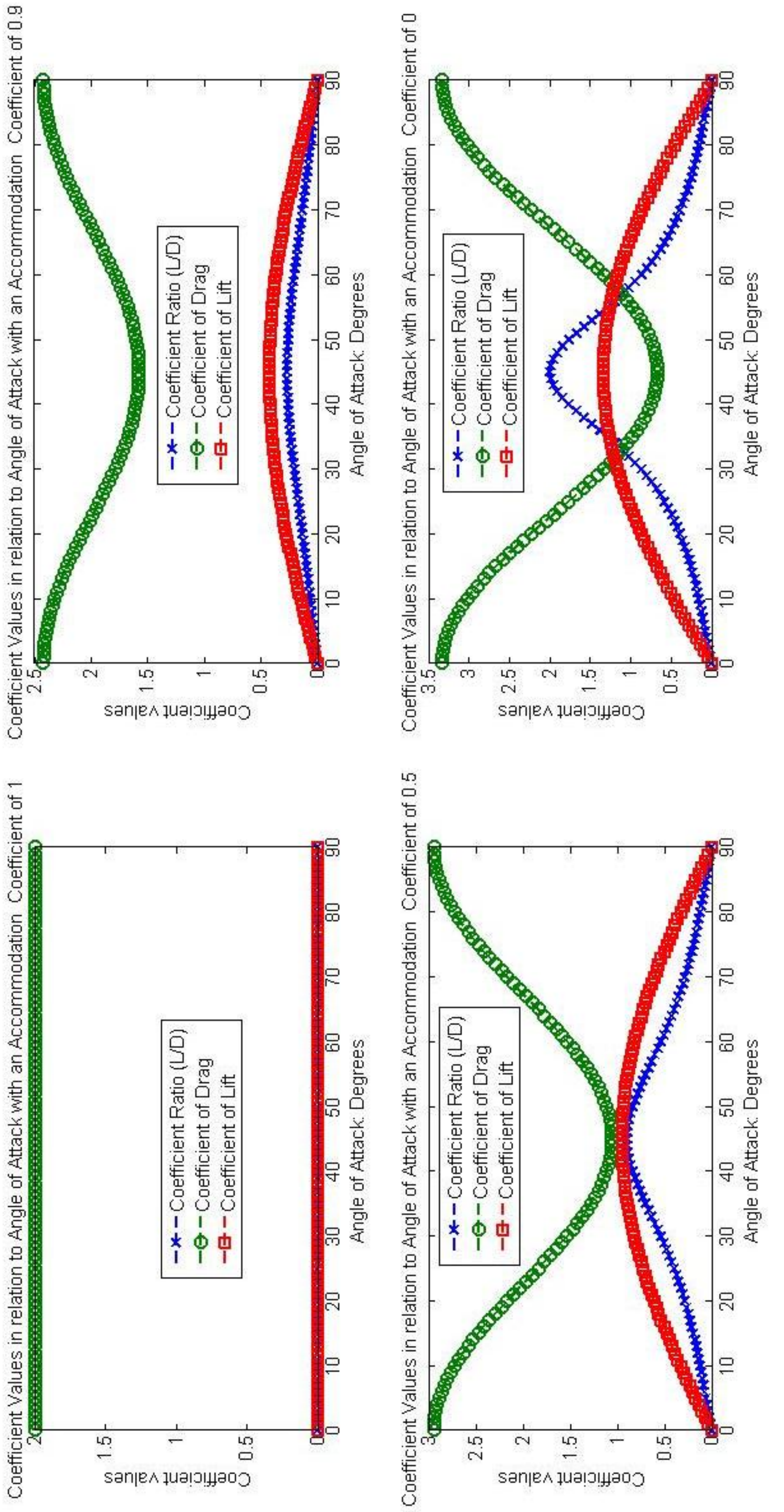


Figure 8: The effect of changes in accommodation coefficient of Drag and Lift Coefficients

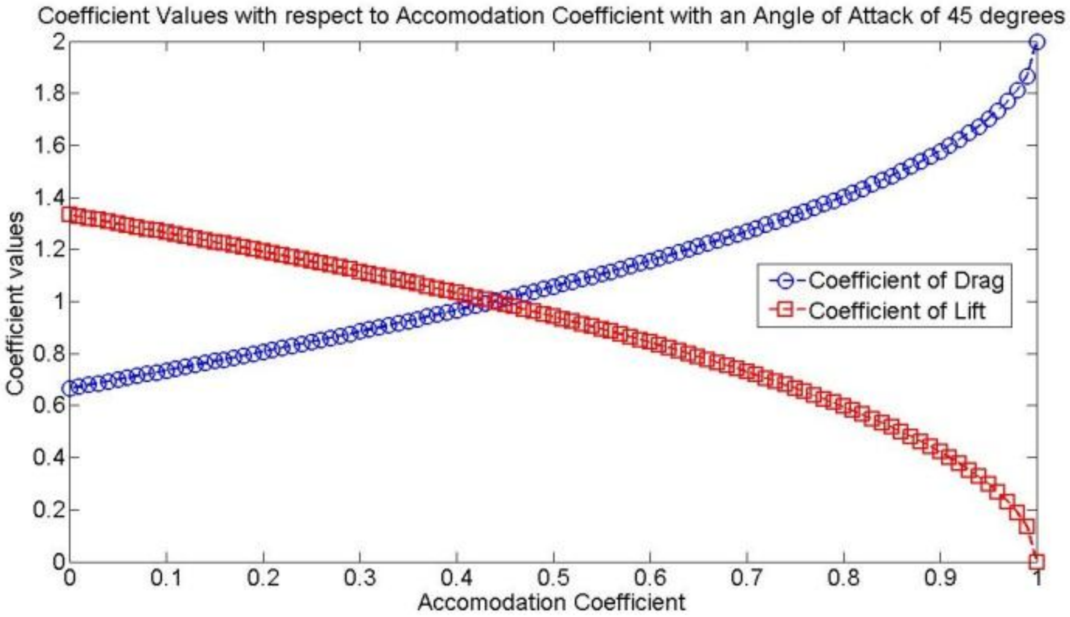


Figure 9: Drag and Lift Coefficient values at 45 with differing accommodation coefficients

The accommodation coefficient is not well defined at orbital altitudes but the accepted values range from one to 0.90 [13], [20]. It is observed from Figure 9 that as the accommodation coefficient value lowers, the coefficient of drag decreases, while simultaneously the coefficient of lift increases. Therefore, all trials to compare lift effects will be done at accommodation coefficient values of one and 0.90, and any results that would have been produced by an accommodation coefficient between one and 0.90 would fall in between that two trial results.

III. COMPUTER MODEL

By definition, a computer model involves one or several differential equations "whose unknown function is a quantity of interest" [21]. Therefore, a model was built using MATLAB's software capabilities and Newton's differential equations of motion [22]. A mathematical model allows a list to be created that describes the effects that are being considered and disregard the effects in which are found to be negligible or unimportant [21].

a. Differential Equation

The previously derived motion equations (found in chapter II) are a second order differential equation. However, MATLAB provides powerful built-in subroutines readily available to solve first order differential equations. Therefore, second order differential equations must be solved by expressing the equations as two first order differential equations. The two coupled first order differential equations used for motion on an orbiting object can be expressed in general form shown in Eq. 15 [7].

$$f(t,y) = \begin{bmatrix} \dot{r} \\ \ddot{r} \end{bmatrix} = \begin{bmatrix} v \\ a \end{bmatrix} \qquad y(t_0) = \begin{bmatrix} r_0 \\ v_o \end{bmatrix} \qquad \text{Eq. 15}$$

where $r_0 = inital\ radius\ vector$
$v_o = initial\ velocity\ vector$
$v = velocity\ vector$
$a = acceleration\ vector$

The initial condition vectors, $\boldsymbol{r_0}$ and $\boldsymbol{v_o}$, are expressed in Cartesian Coordinates, therefore, the acceleration of the cubesat must be expressed in Cartesian Coordinates as well. The acceleration experienced by an orbiting object is expressed by Eq. 16.

$$\ddot{\boldsymbol{r}} = -\frac{GM}{r^2}\frac{\boldsymbol{r}}{r} - \frac{C_d A \rho {v_{rel}}^2}{2m}\frac{\boldsymbol{v_{rel}}}{v_{rel}} - \frac{C_d A \rho {v_r}^2}{2m}\frac{\boldsymbol{v_{lift}}}{v_{rel}} \qquad \text{Eq. 16}$$

In Eq. 16, $\boldsymbol{v_{rel}} = \boldsymbol{v} - \boldsymbol{\omega} \times \boldsymbol{r}$ and $\boldsymbol{v_{lift}} = (\boldsymbol{v_{rel}} \times \boldsymbol{n}) \times \ \boldsymbol{v_{rel}}$. The vector, $\boldsymbol{\omega}$, is defined as $[0\ 0\ \omega_e]$ where the constant ω_e expresses the Earth angular velocity vector and has the value of $7.292115486 \times 10^{-5} \frac{rad}{s}$ [17]. The seconder order differential equation expressed by Eq. 16 then becomes Eq. 17 after incorporating all cross products.

$$\ddot{\boldsymbol{r}} = \boldsymbol{a}_g + \boldsymbol{a}_D + \boldsymbol{a}_L \qquad \text{Eq. 17}$$

$$\boldsymbol{a_g} = -\frac{GM}{r^3}[x\ y\ z]^T$$

$$\boldsymbol{a_D} = -\frac{C_d A \rho {v_r}^2}{2m} \frac{[v_{wind,x} \quad v_{wind,y} \quad v_{zwind,}] - [0\ 0\ \omega_e]^T \times [x\ y\ z]^T}{v_r}$$

$$\boldsymbol{a_L} = -\frac{C_d A \rho {v_r}^2}{2m} \frac{([v_{Wind,x} \quad v_{Wind,y} \quad v_{wind,z}] \times [n_x \quad n_y \quad n_z]^T) \times [v_{wind,x} \quad v_{wind,y} \quad v_{zwind,}]^T}{v_r}$$

The cross product within the relative velocity vector is expressed as below in Cartesian Coordinates.

$$[0\ 0\ \omega_e]^T \times [x\ y\ z]^T \rightarrow \begin{bmatrix} \boldsymbol{x} & \boldsymbol{y} & \boldsymbol{z} \\ 0 & 0 & \omega_e \\ x & y & z \end{bmatrix} \rightarrow \begin{vmatrix} 0 & \omega_e \\ y & z \end{vmatrix} \boldsymbol{x} - \begin{vmatrix} 0 & \omega_e \\ x & z \end{vmatrix} \boldsymbol{y} + \begin{vmatrix} 0 & 0 \\ x & y \end{vmatrix} \boldsymbol{z}$$

$$= (z * 0 - y * \omega_e)\boldsymbol{x} - (z * 0 - x * \omega_e)\boldsymbol{y} + (y * 0 - x * 0)\boldsymbol{z}$$

$$= \begin{bmatrix} -\omega_e y \boldsymbol{x} \\ \omega_e x \boldsymbol{y} \\ 0\boldsymbol{z} \end{bmatrix} \qquad \text{Eq. 18}$$

The cross products within the lift force vector are expressed in Cartesian Coordinates as follows. It is assumed that all lift forces are experienced in the upward direction (in this case along the z-axis) since both lift and drag forces are calculated in the wind axis.

$$([v_{Wind,x} \quad v_{Wind,y} \quad v_{wind,z}\ ] \times [n_x \quad n_y \quad n_z]) \times [v_{wind,x} \quad v_{wind,y} \quad v_{zwind,}]$$

$$= \begin{bmatrix} (v_x n_z v_z)\boldsymbol{x} \\ (v_y n_z v_z)\boldsymbol{y} \\ ({v_y}^2 n_z - n_z {v_x}^2)\boldsymbol{z} \end{bmatrix}^T \qquad \text{Eq. 19}$$

The relative velocity should be calculated with the velocity and radius vectors in the Earth-Centered reference frame [15]. The relative velocity was calculated in this frame and then transformed into the Wind Axis frame to calculate drag and lift. Therefore the acceleration from Eq. 16 is expressed as Eq. 20 after incorporating the cross product and placing it in a Cartesian coordinate matrix.

$$\ddot{r} = \begin{bmatrix} \left(-\frac{GM}{r^3}x - \frac{C_d A\rho v_r}{2m}(v_x + \omega_e y) - \frac{C_l A\rho v_r}{2m}(v_x n_z v_z)\right)\boldsymbol{x} \\ \left(-\frac{GM}{r^3}y - \frac{C_d A\rho v_r}{2m}(v_y - \omega_e x) - \frac{C_l A\rho v_r}{2m}(v_y n_z v_z)\right)\boldsymbol{y} \\ \left(-\frac{GM}{r^3}z - \frac{C_l A\rho v_r}{2m}({v_y}^2 n_z - n_z {v_x}^2)\right)\boldsymbol{z} \end{bmatrix} \qquad \text{Eq. 20}$$

b. Model Inputs

A computer model requires initial values to be entered by the user before running the program. For example, ordinary differential equation (ode) solvers developed by MATLAB require the time span, differential equation function, and the initial state to propagate over time. The proposed model requires several more inputs. First, the time span is broken up into two separate initial inputs. The normal required time span is the 2×1matrix that contains zero and the period of one complete orbit, $[0 \quad T]$. In addition to this, an input was created to allow the user to modify the approximate number of orbits that the cubesat will make without changing the initial time span matrix. Once within the program the time span and number of intervals is adjusted based on the number of orbits to be completed. It should be noted that this is an approximate number of orbits due to the fact that the orbit period reduces over time as drag reduces the satellite orbit and it begins to approach Earth's surface. The number of intervals desired must be an input form the user. This value is most easily determined by expressing the number of intervals to be used as the period of the orbit, *T*, divided by the desired time step, *h*.

There are six quantities that together make up the state of a satellite, which usually is connected with position and velocity [7]. Therefore, for this model the initial state input is expressed by a 6×1 matrix, in which, the first three elements are the initial radial position vector expressed in Cartesian coordinates. The latter three elements are the initial velocity vector also expressed in Cartesian coordinates. The input vector that describes the initial position and velocity of the body should represent the position and velocity at perigee. A 6×1 matrix must be an input created by the user that describes the rotation of the transformation matrices. The first three elements are the x,y, and z axis rotations into the Body Axis, while the following three elements describe the rotation into the Wind Axis. A 6×1 matrix is input that defines the constants that are to be used in calculating gravity, drag, and lift. This matrix includes the Universal Gravitation constant, Earth's mass, surface area of the satellite, the mass of the satellite, and Earth's rotational constant.

c. Coordinate System

As discussed above when transforming vectors a computer program can be set up to create intermediate vectors or the transformation can be done in one equation with multiple matrix multiplications. The same is with multiple coordinate systems as long as they build off each other. When transforming from the Earth-Centered to Wind axis the Body axis is comparable as the transitional frame. The Body axis is not used in calculations, however, it has been decided that an intermediate frame will be created to monitor coordinate transformations. Therefore, for each iteration value for the Earth-Centered, Body, and Wind axis will be calculated and stored.

When transforming coordinate systems, both the radius and velocity vectors must be transformed using the same transformation matrices [12]. The model requires the input of a

1×6 matrix which includes the position and velocity vector. This single input matrix is required in order to more easily run the differential equation solver which outputs one matrix capturing velocity and acceleration. However, in order to transform position and velocity, they initial matrix is split into two corresponding matrices (r and v) and transformed separately.

As mentioned above, whenever a vector is transformed, it will always have the same magnitude and direction. Figure 10 (see below) shows that the position magnitude per iteration is equal in all three coordinate frames after a 30 degree rotation into the Body frame and a 60 degree rotation into the Wind frame.

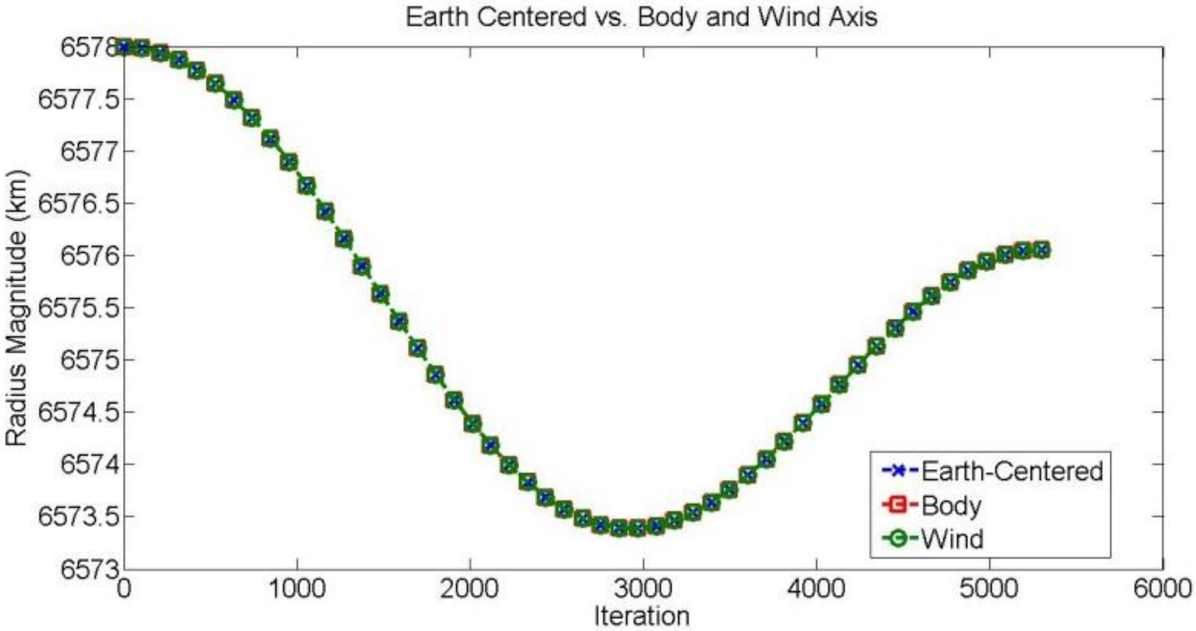


Figure 10: Coordinate System Transformation with unchanged magnitude

For simplicity, it is assumed that there exists a cubesat design with the required attitude control system to stabilize its angle of attack. Thus, there is only rotation about the x–axis between the Earth-Centered and Body axes while there is not rotation between the Body and Wind axes.

d. Atmospheric Model

The chosen atmospheric model for this research is the United States Standard Atmosphere of 1976 (USSA-76). The code that has been implemented was found using MATLAB's Mathworks File Exchange website [23]. This program was developed by Brent Lewis at the University of Colorado-Boulder in 2007 and was revised by Lewis the same year with the assistance of Rich Rieber. Since this MATLAB source code was designed for the purpose of a numerical analysis project, an error analysis was completed to verify the accuracy when compared to the USSA-76. The below table and figures compare the temperature, density, and relative error associated between the USSA-76 and the MATLAB source code [24].

Table 5: Temperature and Density of US Standard and MATLAB Source Code

Altitude (km)	Temperature (K)	Density (kg/m^3)	Temperature MATLAB Model (K)	Density MATLAB Model (kg/m^3)
0	288.150	1.2250	288.15	1.224999156
25	221.552	4.0084×10^{-02}	220.554734	0.046938174
50	270.650	1.0269×10^{-03}	270.65	0.001162706
75	208.399	3.9921×10^{-05}	210.3616998	4.6383×10^{-05}
100	195.080	5.6040×10^{-07}	193.2773459	6.7127×10^{-07}
125	417.230	1.2910×10^{-08}	406.2166827	1.4267×10^{-08}
150	634.390	2.7060×10^{-09}	627.5354045	2.1957×10^{-09}
175	769.810	6.3390×10^{-10}	765.527083	6.6032×10^{-10}
200	854.560	2.5410×10^{-10}	851.87331	2.6265×10^{-10}
225	907.780	1.1840×10^{-10}	906.0948474	1.2182×10^{-10}
250	941.330	6.0730×10^{-11}	940.2629471	6.2295×10^{-11}
275	962.540	3.3290×10^{-11}	961.868887	3.4077×10^{-11}
300	976.010	1.9160×10^{-11}	975.5780803	1.9578×10^{-11}
350	990.060	7.0140×10^{-12}	989.8816712	7.1521×10^{-12}
400	995.83	2.8×10^{-12}	995.7528056	2.8543×10^{-12}
450	998.22	1.2×10^{-12}	998.194344	1.2051×10^{-12}
500	999.24	5.22×10^{-13}	999.2227044	5.3025×10^{-13}

Table 5 lists the temperature and density of the atmosphere up to 500 km. If a cubesat began an orbit at 500 km the density is low enough to require several thousand orbits to end its orbit life. Therefore, we will not be analyzing satellites that orbit at high LEOs.

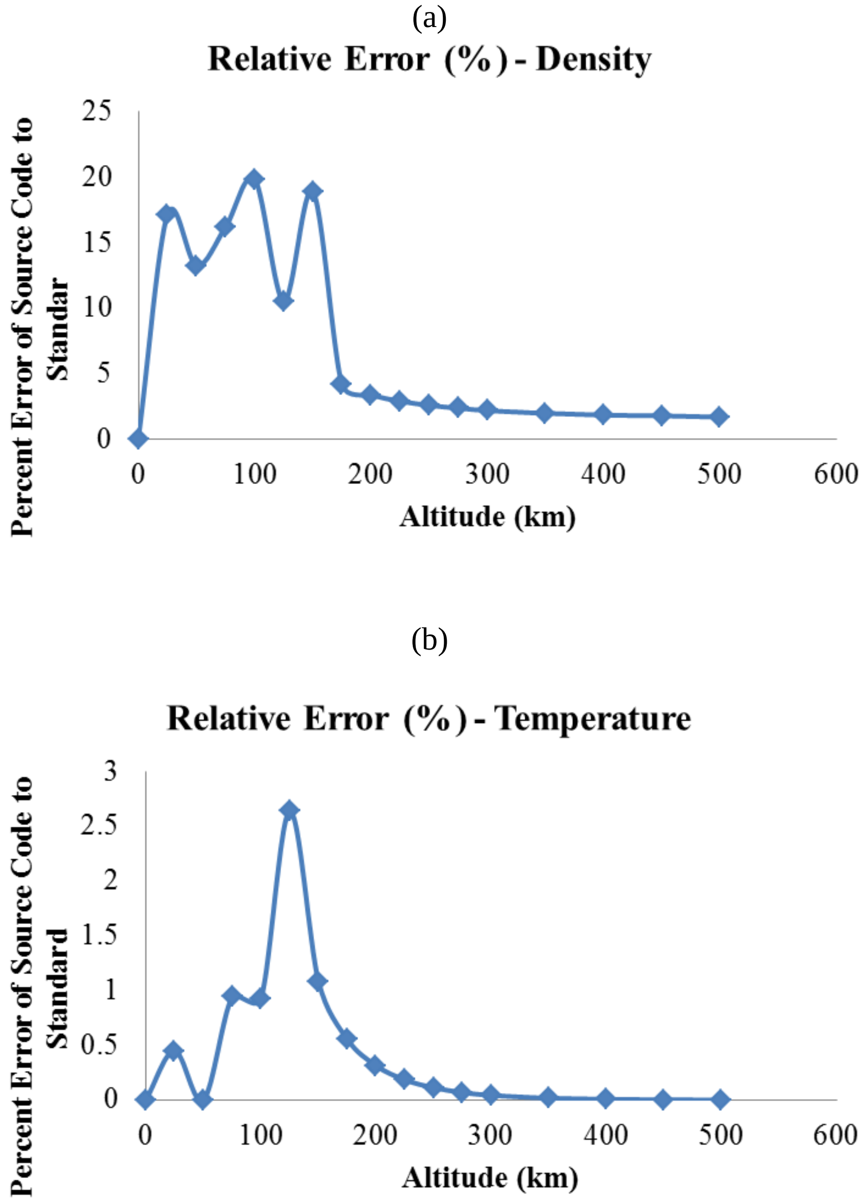


Figure 11: Relative Error (a) Density (b) Temperature

Figure 11 shows the percent difference between the USSA-76 and the MATLAB source code. It can be seen in Figure 11(a) that below 175 km the percent error varies between ten and

20 percent. This error is too large to use in general engineering calculations; however, the length of orbit at that altitude will last only approximately 3.5 orbits at 175 km and 1 orbit at 150 km (as seen in Figure 12) rendering any analysis of cubesats at this altitude not beneficial to understanding the effects of lift. The range that will be most analyzed will be 200 km to 500 km where the percent error is less than five.

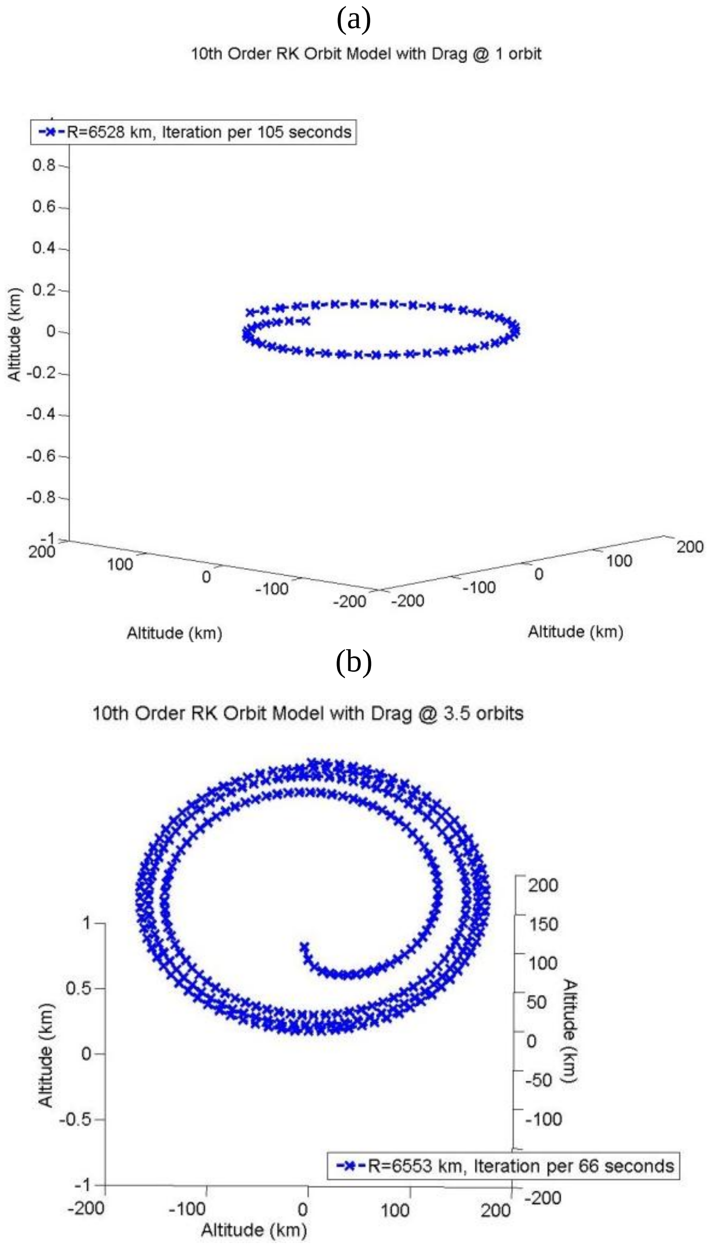
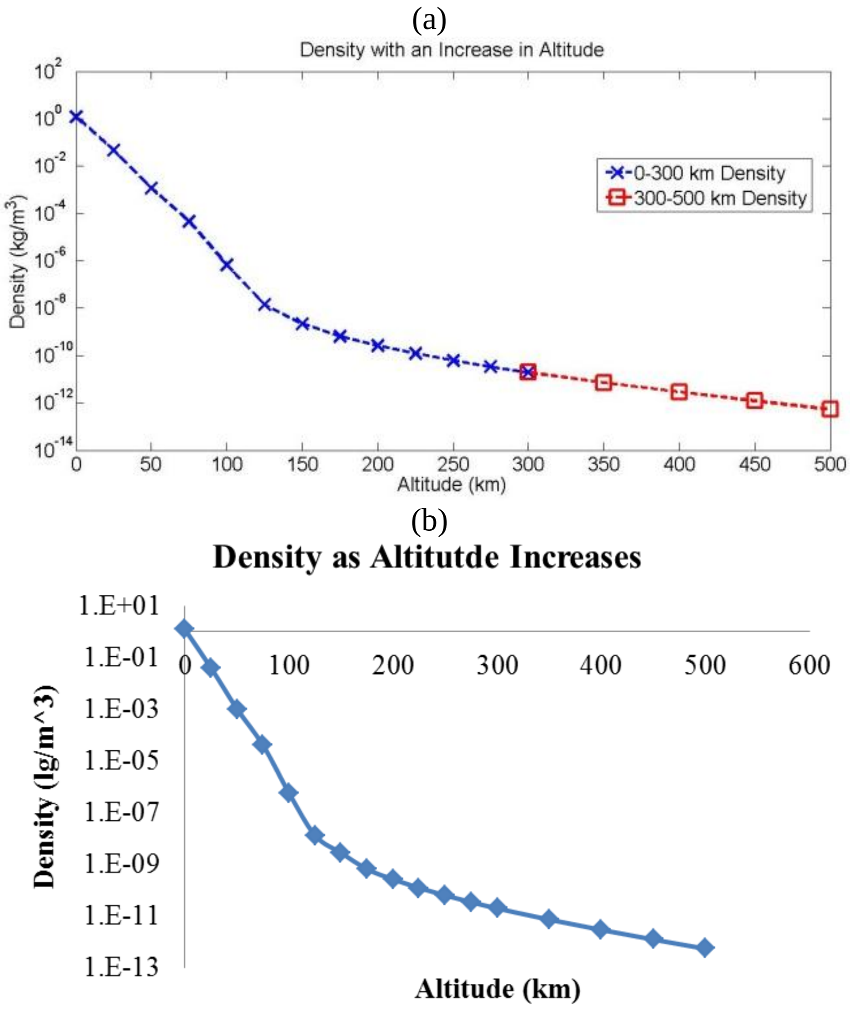


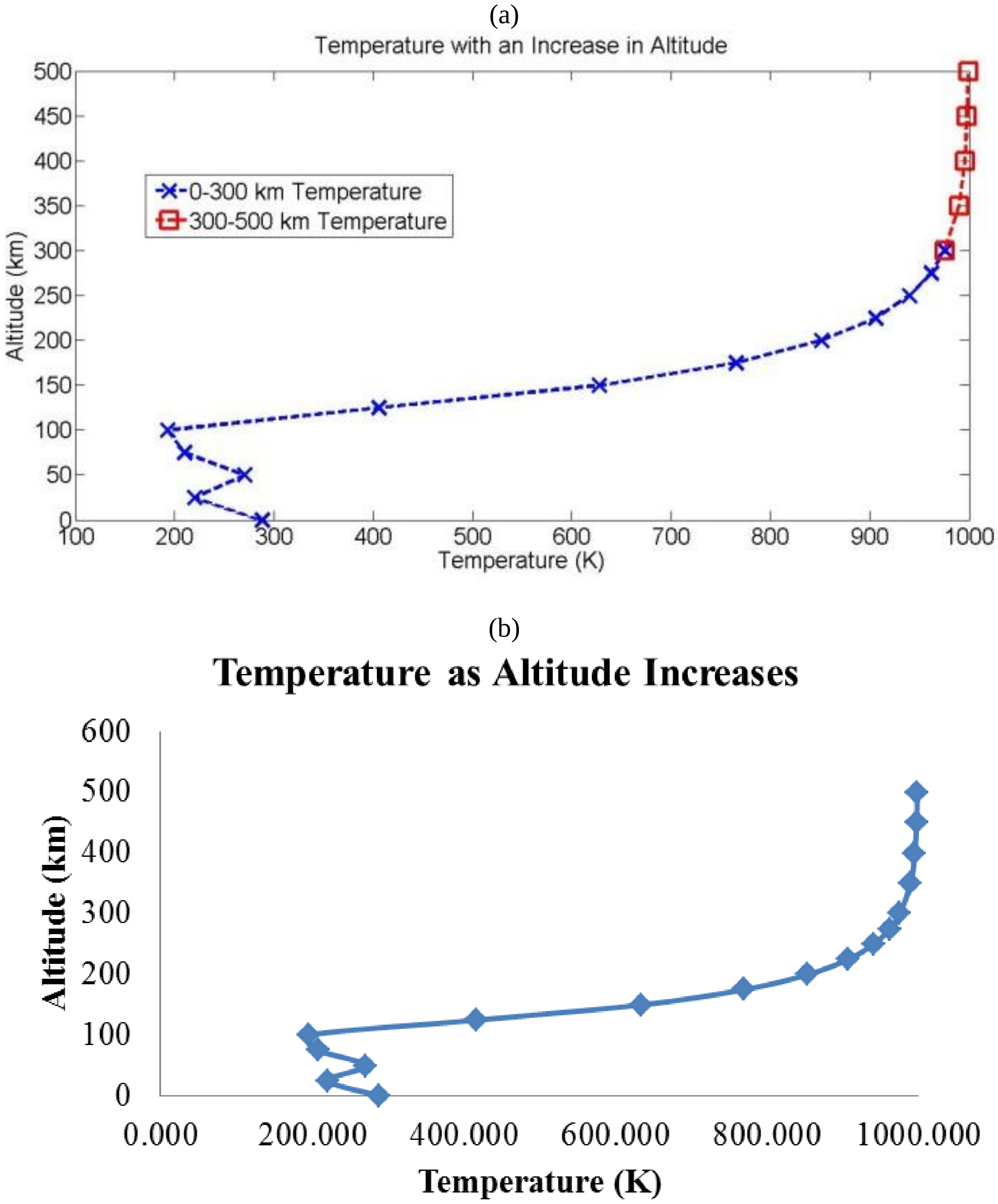
Figure 12: Drag Orbital Model (a) 150km (b) 175 km

The below figures compares the trends of temperature, density, and gravity created from the MATALB source code, USSA-76, and Newton’s Law of Gravitation. Figure 13 plots density as numerically found using the MATLAB source and the USSA-76. Figure 14 plots these temperature variations as numerically found using the MATLAB source and the USSA-76. Figure 15 plots gravity as numerically found using the MATLAB source and Newton’s Law of Gravitation.


(a)

(b)

Figure 13: Density of (a) MATLAB source code (b) USSA – 76

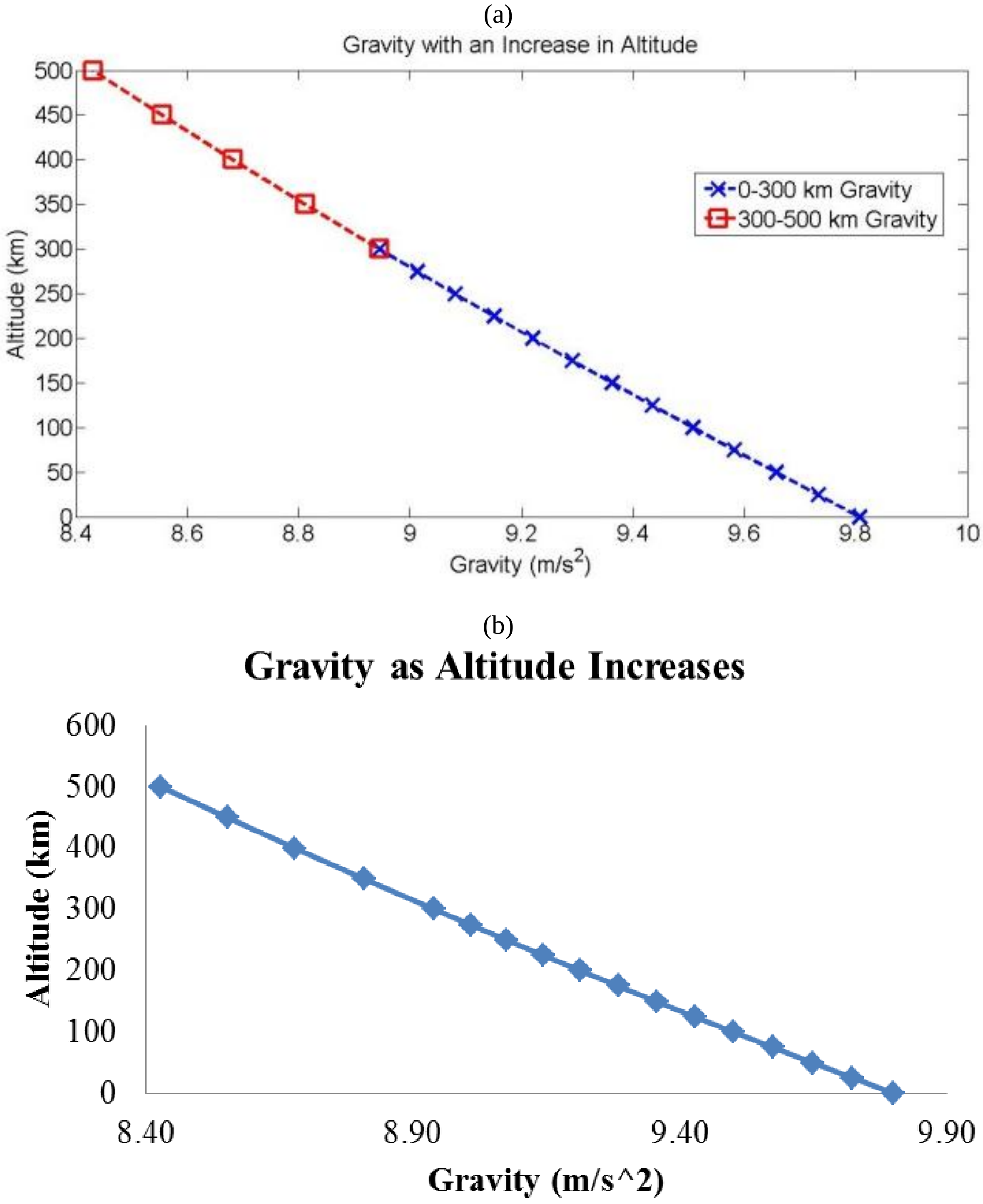

(a)

Temperature with an Increase in Altitude

(b)

Temperature as Altitude Increases

Figure 14: Atmospheric Temperature (a) using MATLAB source code (b) using USSA-76



(a)

(b)

Figure 15: Gravity (a) using MATLAB source code (b) Newton’s Law of Gravitation

The USSA-76 model meets the basic accuracy requirements of this research; however, this it should be replaced by the NRLMSIS-00 model for higher accuracy solutions. The USSA was released in 1962 and 1976 in an effort to create a "standard reference model". The Standard Atmosphere attempts to fix the errors associated with exponential models by choosing bands that increase accuracy. However, the NRSMSIS-00 takes into account that at the radius of perigee a satellite can cross several bands in a single iteration [7]. Therefore, the atmospheric model using the NRLMSIS-00 source code should be used to replace the USSA-76 atmospheric model that is currently being used; this is not implemented due the need to convert the NRLMSIS-00 code from FORTRAN 77 to MATLAB.

e. Numerical Integration Methods

i. Computational Costs

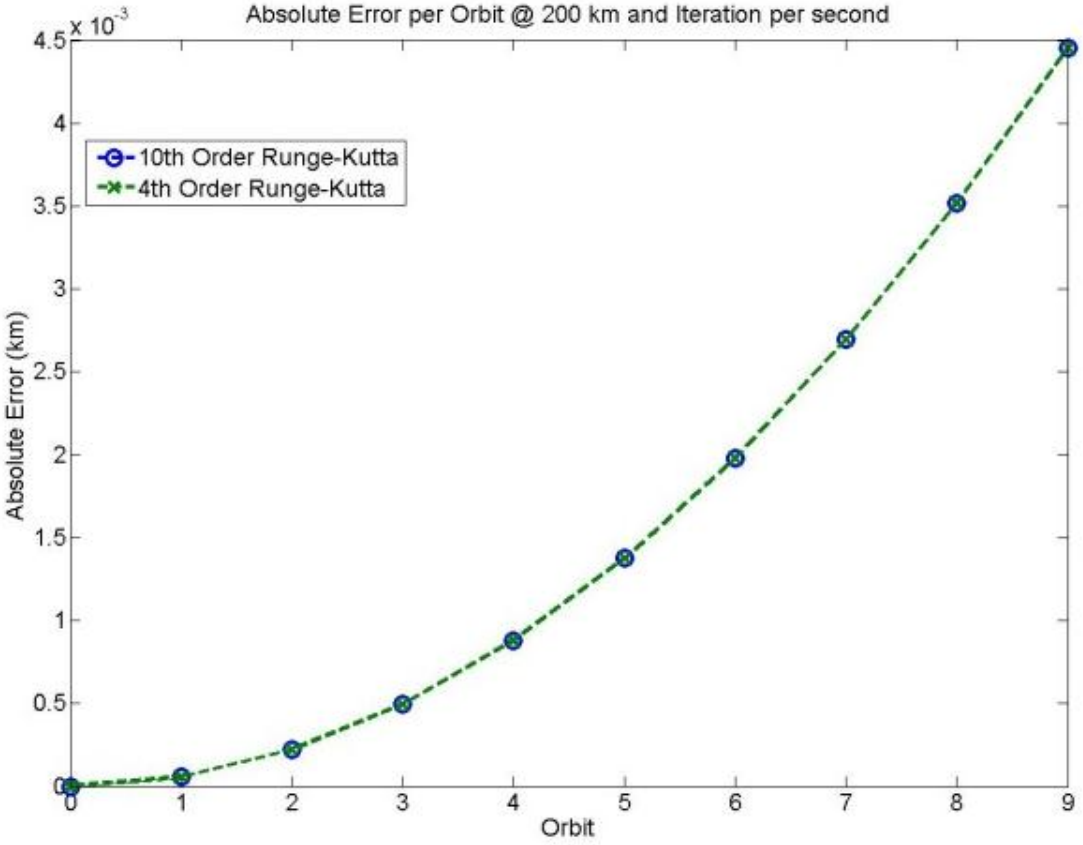
When this model was originally designed it used a simple 4th order RK method (RK4); however, this method only maintains a local truncation error of $O(h^4)$ [14]. The term *truncation error* is defined as "the error involved in using a truncated, or finite, summation to approximate the sum of an infinite series" [14]. In order to guarantee accuracy a higher order RK method was desirable. Therefore, a new RK numerical differential equation solver was built using the 10th order RK method (RK10) which has a local truncation error of $O(h^8)$ [25]. However, several analytical comparisons of the two methods were completed to verify which method was to be utilized in the model.

First, FLOPS were approximated for both RK methods and compared. FLOP stand for *floating point operation* and is defined as a "theoretical and crude measure of efficiency." The flop count is basically the number of floating point operations involved in an algorithm. A floating point operation is any mathematical operation ±, ×, ÷, etc. [26]. When integrating by

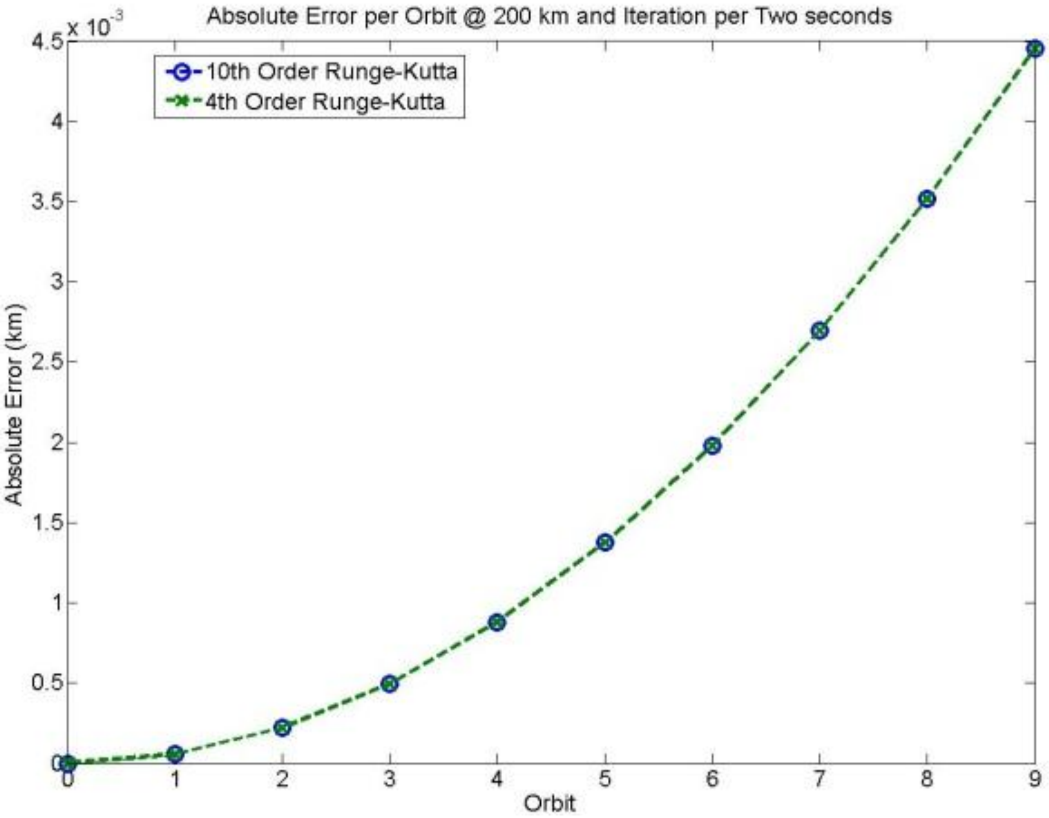
N (amount of intervals) the total FLOP count for the RK4 method is $(22 \times N)$ whereas the FLOP count for the RK10 method is $(239 \times N)$.

Second, the errors associated with differing step sizes were calculated. More will be discussed later (in this chapter) on how step sizes were solved for, but for now a comparison was made between numerical solutions to an orbit at 200 km with step sizes (h) equal to 1,2,3,10, and 106 seconds. From Figure 16 it can be seen that the max step size that can be utilized with RK4 without a drastic increase in error is 10 seconds; whereas the RK10 can reach 106 seconds.

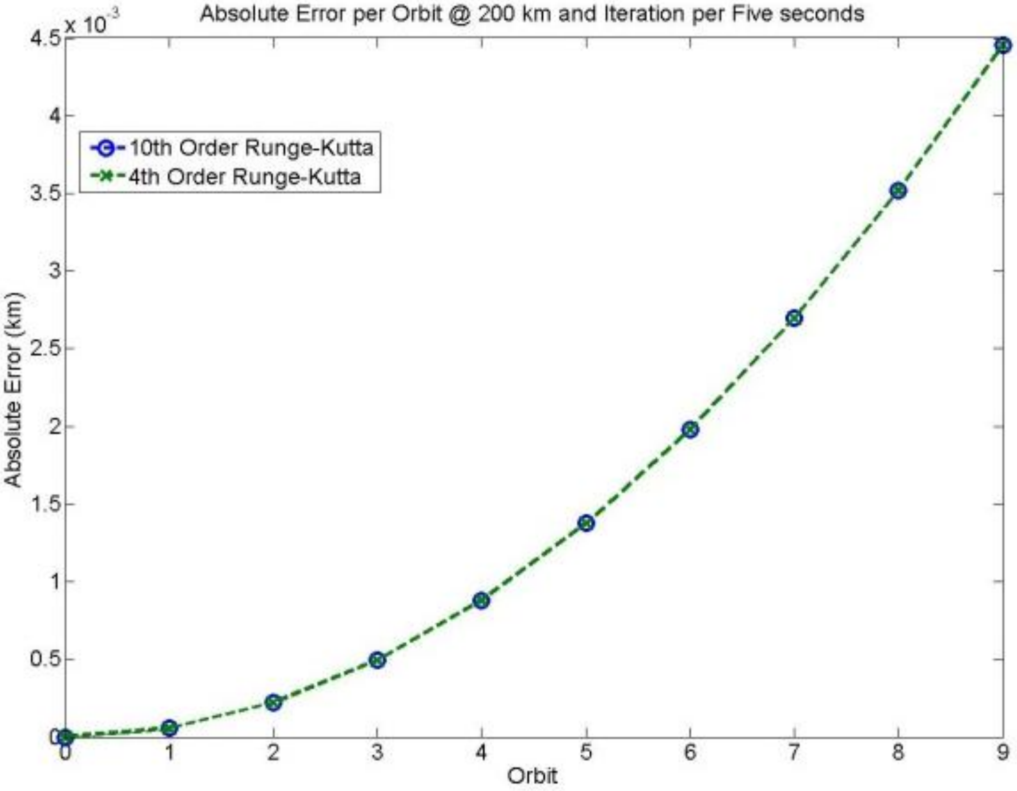
(a)



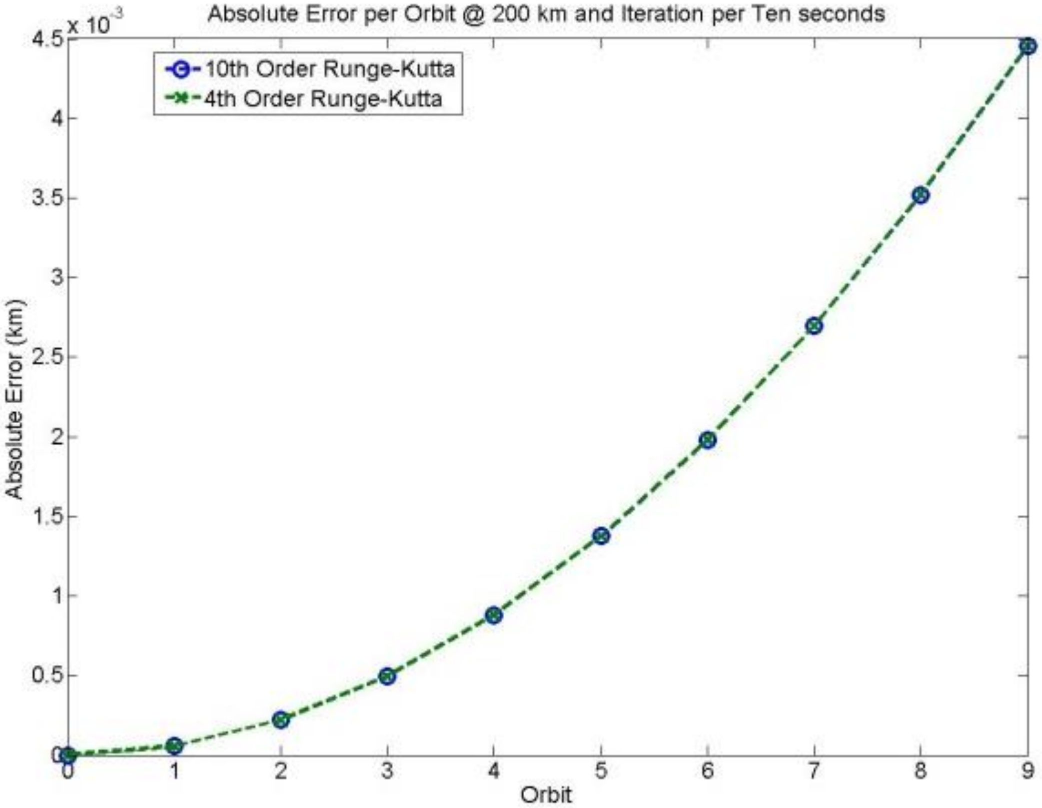
(b)


Absolute Error per Orbit @ 200 km and Iteration per Two seconds
x 10^-3
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Absolute Error (km)
10th Order Runge-Kutta
4th Order Runge-Kutta
0
1
2
3
4
5
6
7
8
9
Orbit

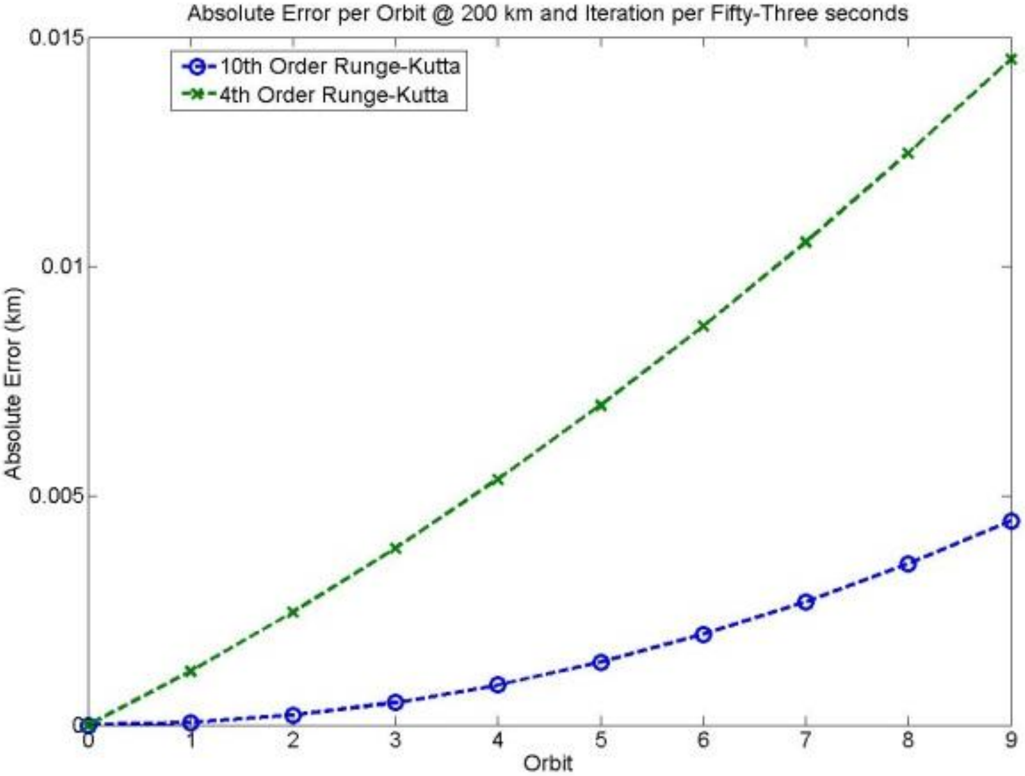
(c)


Absolute Error per Orbit @ 200 km and Iteration per Five seconds
x 10^-3
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Absolute Error (km)
10th Order Runge-Kutta
4th Order Runge-Kutta
0
1
2
3
4
5
6
7
8
9
Orbit

(d)


Absolute Error per Orbit @ 200 km and Iteration per Ten seconds
x 10^-3
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Absolute Error (km)
0
1
2
3
4
5
6
7
8
9
Orbit
10th Order Runge-Kutta
4th Order Runge-Kutta

(e)


Absolute Error per Orbit @ 200 km and Iteration per Fifty-Three seconds
0.015
0.01
0.005
0
Absolute Error (km)
0
1
2
3
4
5
6
7
8
9
Orbit
10th Order Runge-Kutta
4th Order Runge-Kutta

(f)

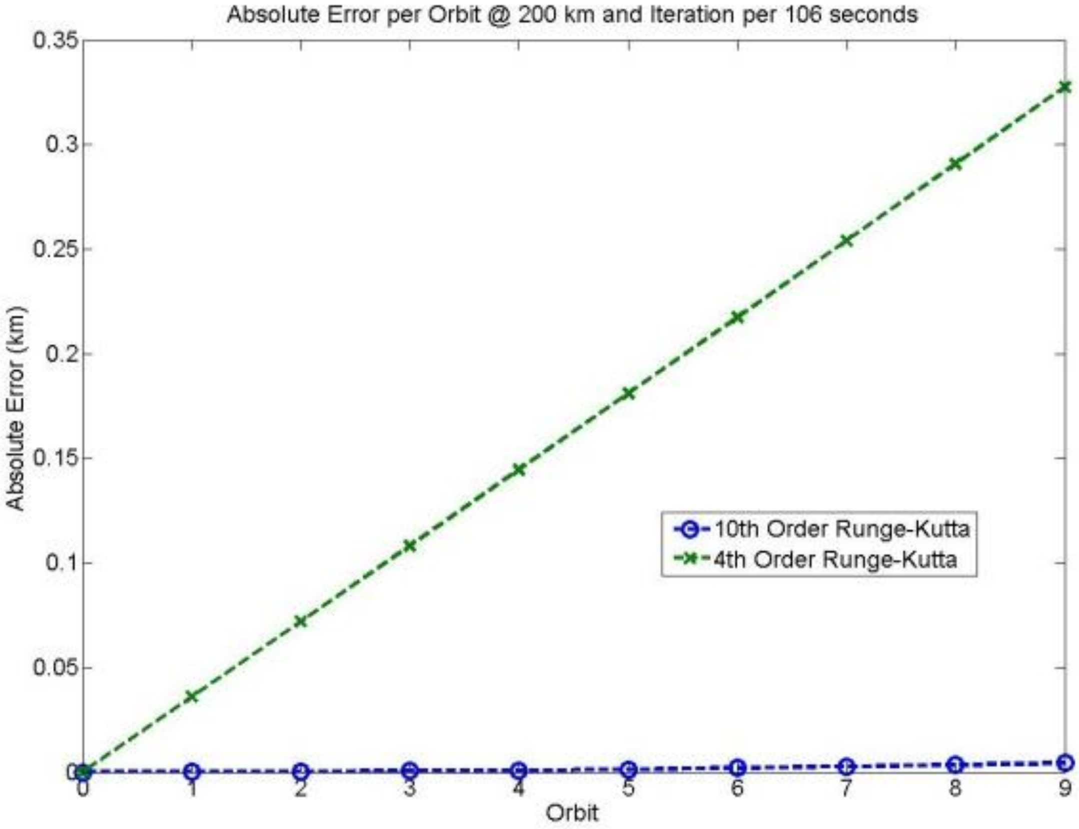


Figure 16: Error Associated with Step Size over 10 Orbits
(a) h=1 sec (b) h= 2 sec (c) h=5 sec (d) h= 10 sec (e) h =53 sec (f) h= 106 sec

Third, the two R.K. methods tend to follow the same increase in error as long as the step size does not get too large. Therefore, an analysis was done to see benefits provided by the RK10 and to measure the additional computation costs. If a visual comparison of the RK4 and RK10 is done it appears that the RK10 will take more computation time based on the amount of equations to solve. The RK4 uses 12 constants and 5 equations to solve for the next time state, whereas the RK10 uses 187 constants and 17 equations. This visual comparison is correct if the same step size is used. MATLAB's "*tictoc"* feature was used to measure the computational time of both methods at an orbit of 200 km and a time step of 10 seconds. The RK4 method would take 0.59788 seconds while the RK10 method would take 3.58104 seconds. However, this visual comparison is not accurate if the methods used differing time steps that still give accurate results. As can be seen in Figure 16, at a step size of 10 seconds the RK4 method had an absolute error of 4.5×10^{-3} and after 10 orbits. The RK10 (if plotted separately) has an absolute error of 4.5×10^{-3} while utilizing a step size of 106 seconds. These two step sizes were used to approximate computation time if differing step sizes were used. In order to create a model of an orbiting object at 200 km the RK4 takes 0.55734 seconds while the RK10 takes 0.50405 seconds.

Fourth, a comparison was made between saved computation time and a reduction in absolute error. The time to calculate a model over 10 and 100 orbits at an altitude of 200 km was measured at a time step of 10 seconds. The RK4 method contributed an absolute error of 0.004456 km after 10 orbits, whereas the RK10 contributed an absolute error of 0.004454 km. The increase computational time was 2.71782 seconds. The RK4 method contributed an absolute error of 0.513009 km after 100 orbits, whereas the RK10 contributed an absolute error of 0.512984 km. The increase computational time was 2.35552 seconds. The computational time was calculated on a Dell OptiPlex 745 with an Intel Core TM2 6400 processor at 2.13 GHz, a 2 GB RAM, and a 250 GB hard drive.

Based on the factors discussed above, in the introduction to numerical integration methods, the 10th order Runge-Kutta method was chosen to numerically solve time iterations. This method was chosen for several reasons. The single step method was chosen to avoid the creation of multiple high order of error methods required to start multi-step methods. The largest advantage of the RK10 method is the allowance to adjust the step size from a small step with relatively small error to larger step sizes that size produces accurate models. The small step sizes are sufficient when creating models that run for a low number of orbits (approximately 20).

However, a small step size when modeling the effects of lift at 450 km take several thousand iterations, and in order to keep the computation costs low a larger step size is required.

ii. Step Size

The step size must be calculated beforehand in order to insure that the program will run successfully. MATLAB runs by creating matrices for all variables and, therefore, the program must run with intervals that divide out evenly into integer specific matrix locations. Number trees were used to find step sizes that would allow MATLAB to step at integers during an orbit. Number trees assist in finding all the multiples of a specific number. If any two numbers are on the same row and both are connected to a single number a row above them, then the number in the top row can be found by multiplying the two below it. From Figure 17 you can see that the only available step sizes for an time period of 5300 seconds is 530, 106, 53, 10, 5, 2, or 1 second. However, step sizes as large as 530 can become unreasonable due to the large lapsed time in measured perturbing forces.

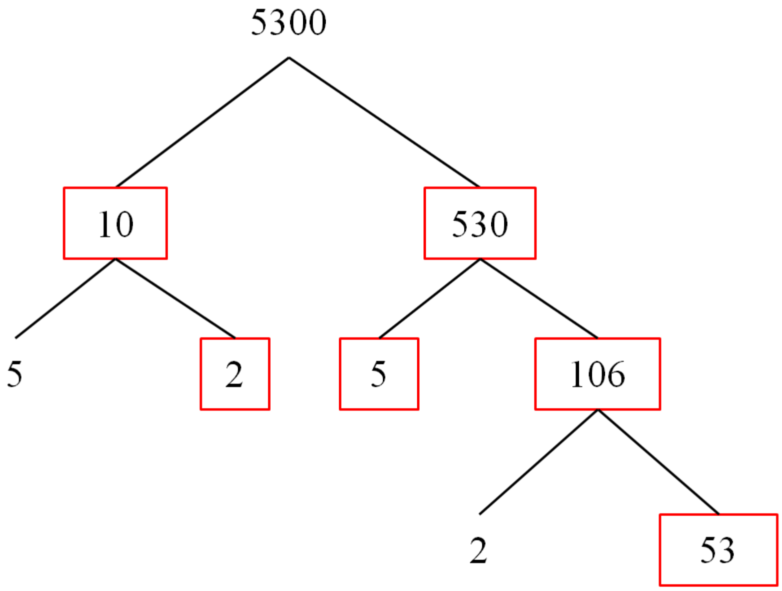


Figure 17: Number Tree that provides the possible step sizes

Currently the step sizes are based off of the period in which the object orbits. For example (referring to above) at a 200 km circular orbit there are the specified time steps given in Figure 17. However, if the model were created at 250 km the time steps would be different and would require another number tree. This method can become tedious. If a general time step could be found for the RK10 method than the same step could be used in every model and the user could modify how long the desired time length. However, it is convenient to us the period of an orbit because it allows for models run for any given number of approximate orbits.

IV. EXPERIMENTATION AND RESULTS

a. Computer Generated Models

In order to develop a computer model that would accurately calculate the position and velocity over time, the model was built through-out several small steps instead of immediately attempting to build a model that involves an atmosphere, drag, lift, and a high order of error. First, a simple projectile motion model was developed using the accepted values for Two-Dimensional acceleration. Second, a computer model was created to account only for gravity. Third, a model was created that accounted for only gravity and drag. Finally, a model was created to account for gravity, lift, and drag. In the future it is possible to modify the model if forces such as solar pressure or thrust were desired to be added.

In order to build a model that will calculate the position of a cubesat experiencing gravity, drag, and lift the problem was broken down into four small models. As each model was built the next one built on top of the existing model. To begin a model was creating that propagated a projectile motion problem. Second, a gravity only model was created; third, a gravity, drag model; And lastly, a gravity, drag and lift model.

i. Projectile Motion

The projectile motion of a simple object being fired at a given angle was plotted using the RK solver. The differential vectors still include the velocity and acceleration.

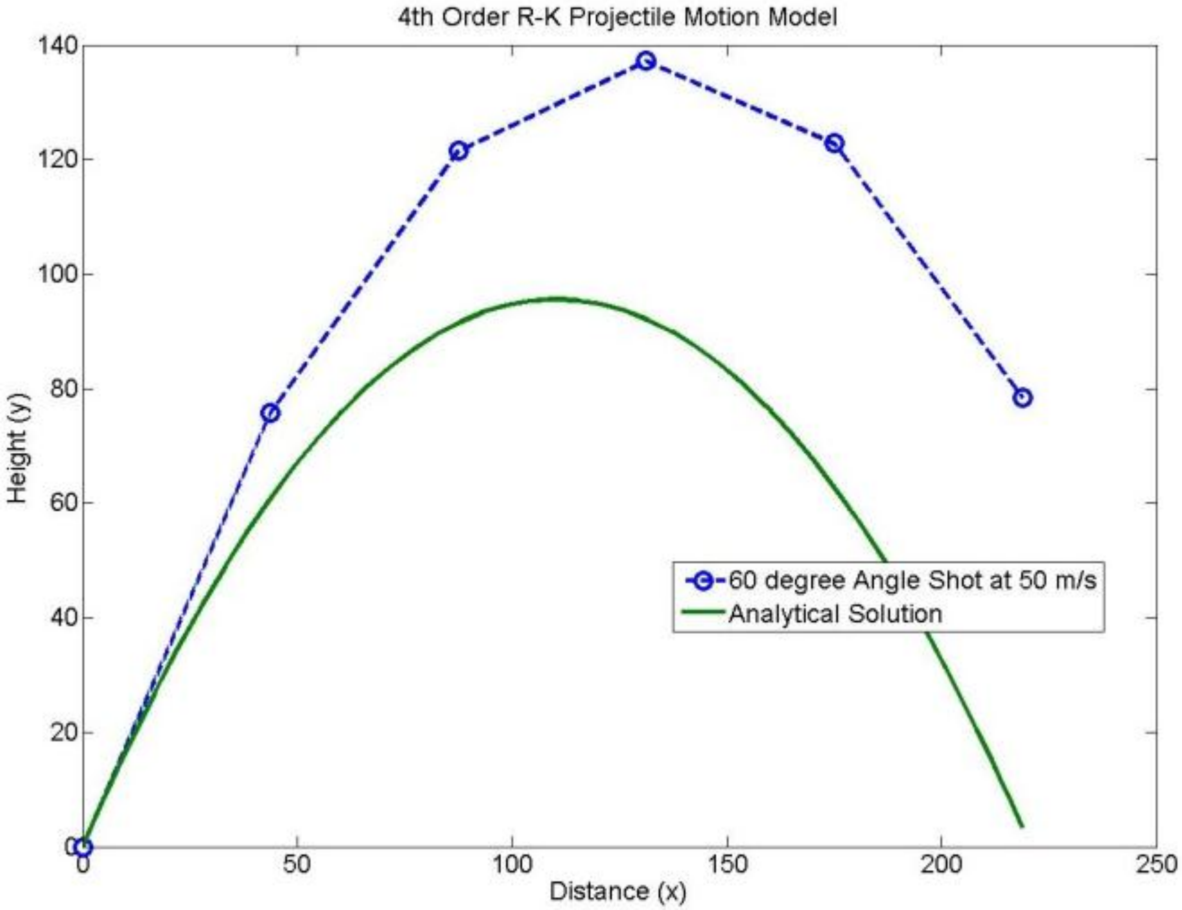
$$f(t,y) = \begin{bmatrix} \dot{r} \\ \ddot{r} \end{bmatrix} = \begin{bmatrix} v \\ a \end{bmatrix} \qquad y(t_0) = \begin{bmatrix} r_0 \\ v_o \end{bmatrix}$$

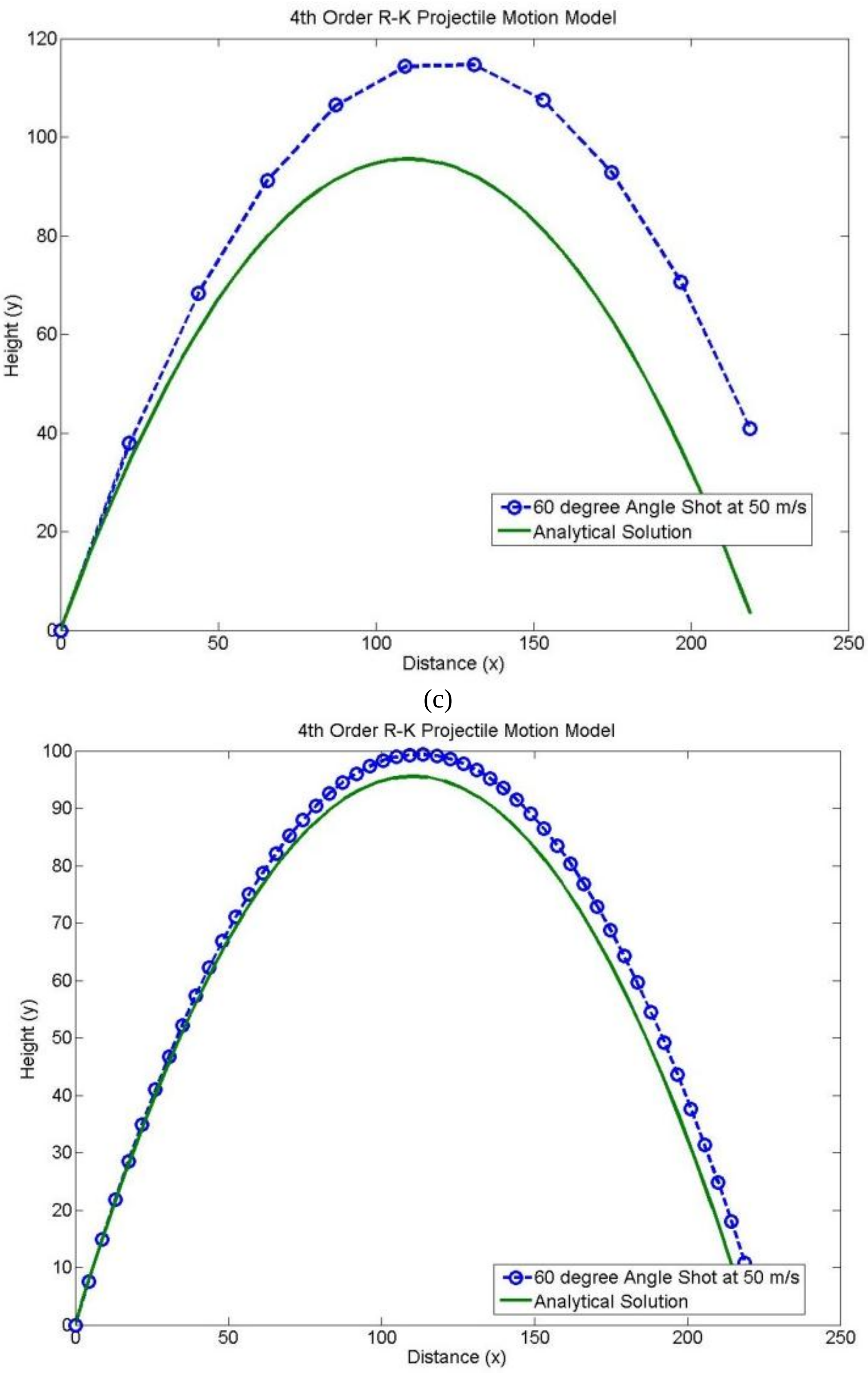
The velocity was broken down into its x and y components and was substituted into the form above to form Eq. 21.

$$\begin{bmatrix} \dot{r} \\ \ddot{r} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ a_x \\ a_y \end{bmatrix} \qquad \text{Eq. 21}$$

The acceleration in the x–direction is zero due to the assumption of no drag and -9.81 $\frac{m}{s^2}$ (gravity) in the y–direction using the assumption that gravity is constant [27]. Figure 18 shows the results from plotting an object being fired at 50 $\frac{m}{s}$ at a 60 degree angle. The propagated model is compared with an analytical solution in order to help in illustrating how the model is reacting to changes. **NOTE:** All values are in meters.

(a)





(b)

4th Order R-K Projectile Motion Model
Height (y)
Distance (x)
0
20
40
60
80
100
120
0
50
100
150
200
250
60 degree Angle Shot at 50 m/s
Analytical Solution

(c)

4th Order R-K Projectile Motion Model
Height (y)
Distance (x)
0
10
20
30
40
50
60
70
80
90
100
0
50
100
150
200
250
60 degree Angle Shot at 50 m/s
Analytical Solution

(d)

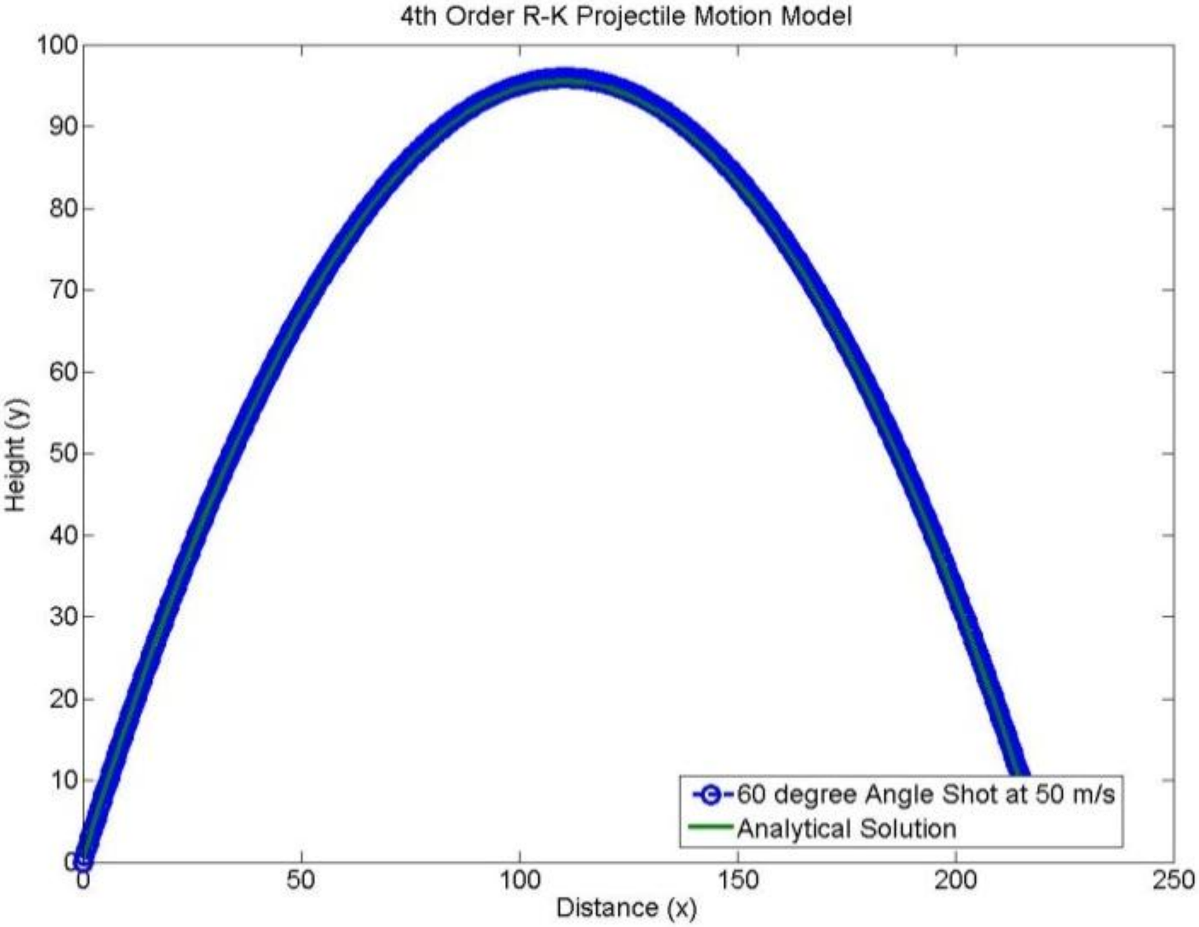


Figure 18: Projectile motion with increasing intervals of calculation (a) 5 Intervals (b) 10 Intervals (c) 50 Intervals (d) 500 Intervals

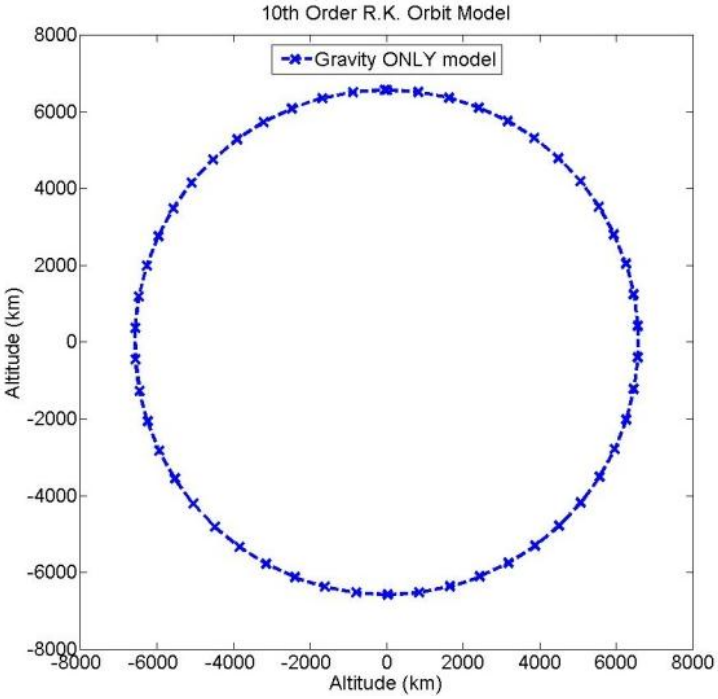
An important observation to make from Figure 18 is the increased accuracy of the projectile motion model with the increased number of intervals used. It is comparable to the concepts of analytical integration in calculus. The basic understanding of integrating involves breaking up a function into an infinite number of intervals and evaluating the area in that region by adding the infinitely small intervals together. Therefore, in theory, if the number of intervals were forever increased, the more accurate the model should always be. However, this is not the case because of two reasons. First, evaluating that many intervals would take too much computation cost. Second, there is a bound on the error that will always exist when numerically computing numbers due to rounding and truncation errors. Therefore, an appropriate number of intervals is found and used with an understanding of the error associated with the model.

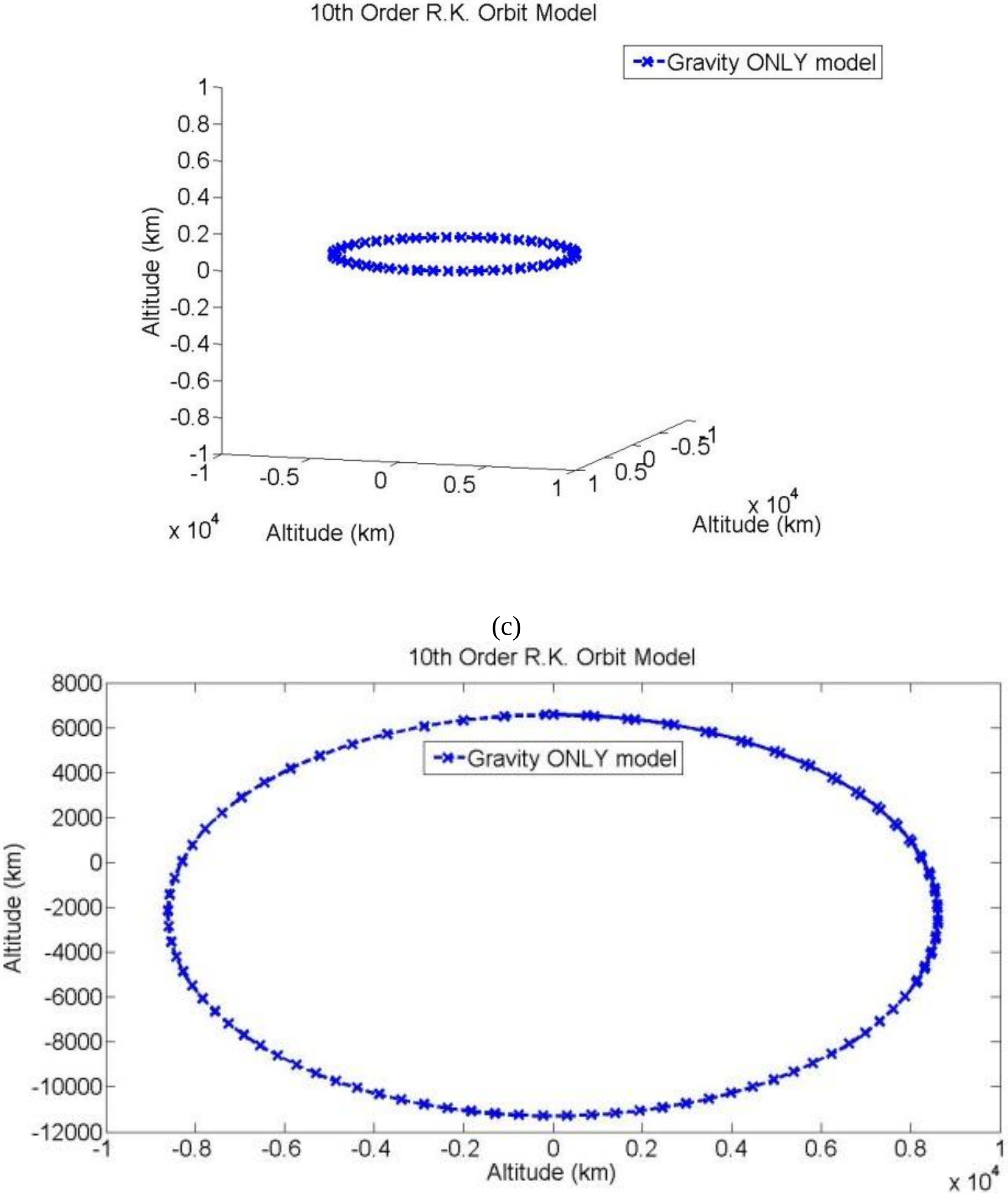
ii. Gravity Only

Once the projectile motion model was built, minor changes were required into order to model an orbiting object instead of a simple motion in flight. First, the model of an orbiting satellite is done in a three-dimensional system. Second, it is now assumed that gravity does not remain constant. As seen Figure 15, gravity decreases with an increase in altitude and is defined by Newton's Law of Gravitation. Since orbits with altitudes between 200 km and 400 km will be analyzed, varying gravity must be taken into account. An orbiting object will breakdown to have components of acceleration in each coordinate direction, but the z–direction will not experience any unless there is an inclination involved. Therefore, Eq. 21 is transformed into Eq. 22 and is modeled in Figure 19.

$$\begin{bmatrix}\dot{r}\\ \ddot{r}\end{bmatrix} = \begin{bmatrix} v_x & v_y & v_z & -\frac{GM}{r^3}x & -\frac{GM}{r^3}y & -\frac{GM}{r^3}z \end{bmatrix}^T \qquad \text{Eq. 22}$$

(a)




(b)

(c)

10th Order R.K. Orbit Model
Gravity ONLY model
Altitude (km)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.5
0
0.5
1
1
0.5
0
-0.5
-1
x 10^4
Altitude (km)
x 10^4
Altitude (km)
10th Order R.K. Orbit Model
Gravity ONLY model
8000
6000
4000
2000
0
-2000
-4000
-6000
-8000
-10000
-12000
Altitude (km)
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
Altitude (km)
x 10^4

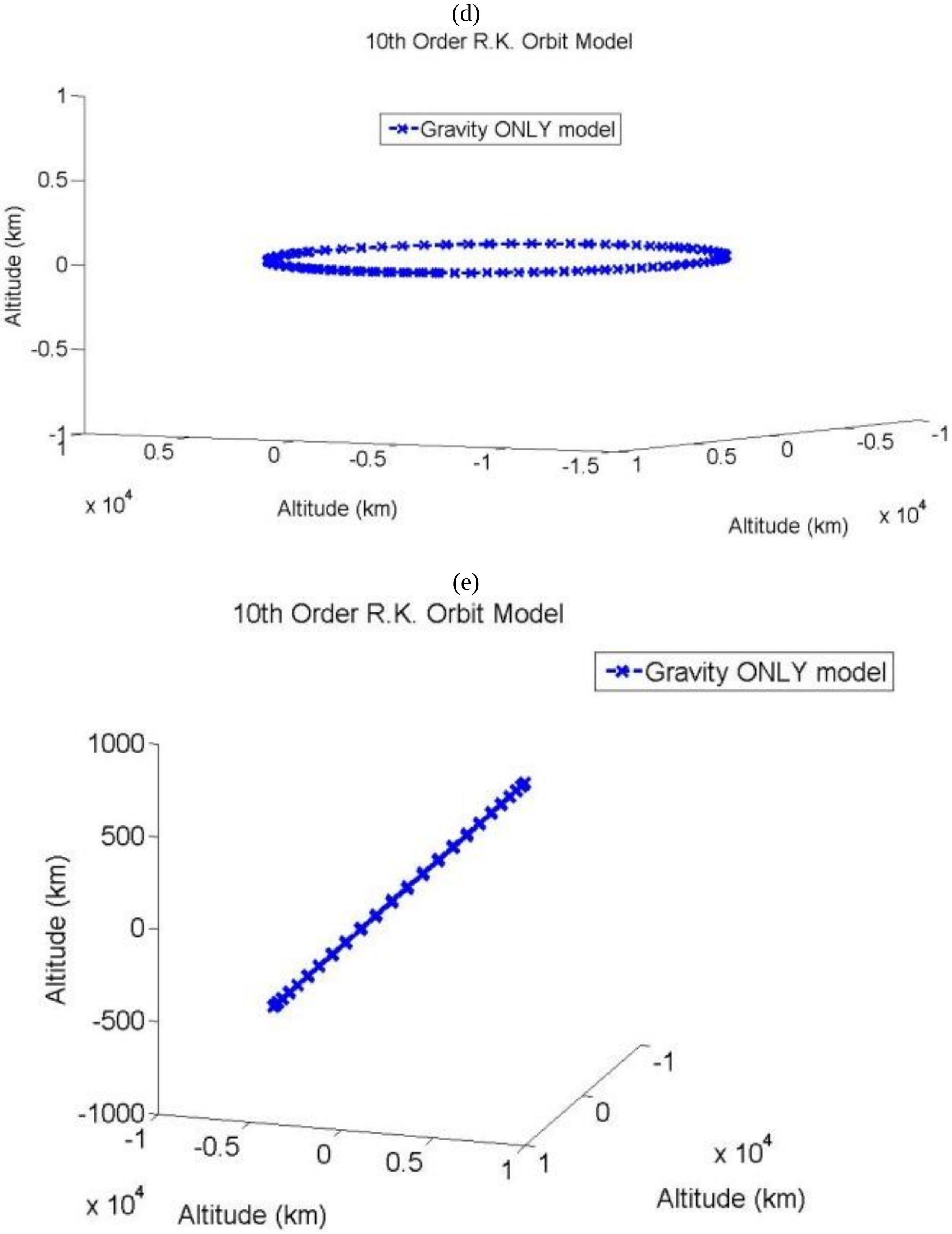


Figure 19: Gravity only Orbit Models (a-b) Circular (c-d) Elliptic (e) Circular with inclination

The different orbit shapes were created by adjusting the initial state vector which includes initial position and velocity vectors. The difference between an elliptic and a circular orbit is a slight adjustment in the velocity. This analytically corresponds with the velocity show in Eq. 23.

$$V = \sqrt{\frac{2\mu}{r} - \frac{\mu}{a}} \qquad \text{Eq. 23}$$

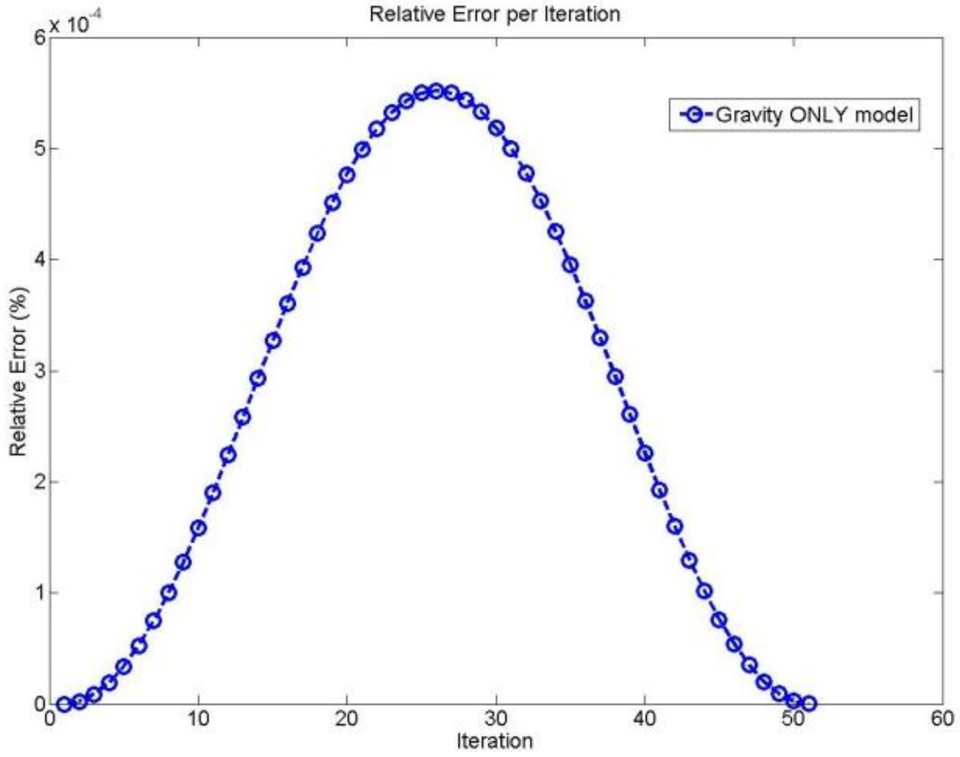
If the orbiting object follows a circular path, then $a = r$ and $V = \sqrt{\frac{\mu}{r}}$. Therefore, if the velocity is adjusted up by 25 percent then $a = 2.28r$ (found using Eq. 24) [16].

$$1.25\sqrt{\frac{\mu}{r}} = \sqrt{\frac{2\mu}{r} - \frac{\mu}{a}} \qquad \text{Eq. 24}$$

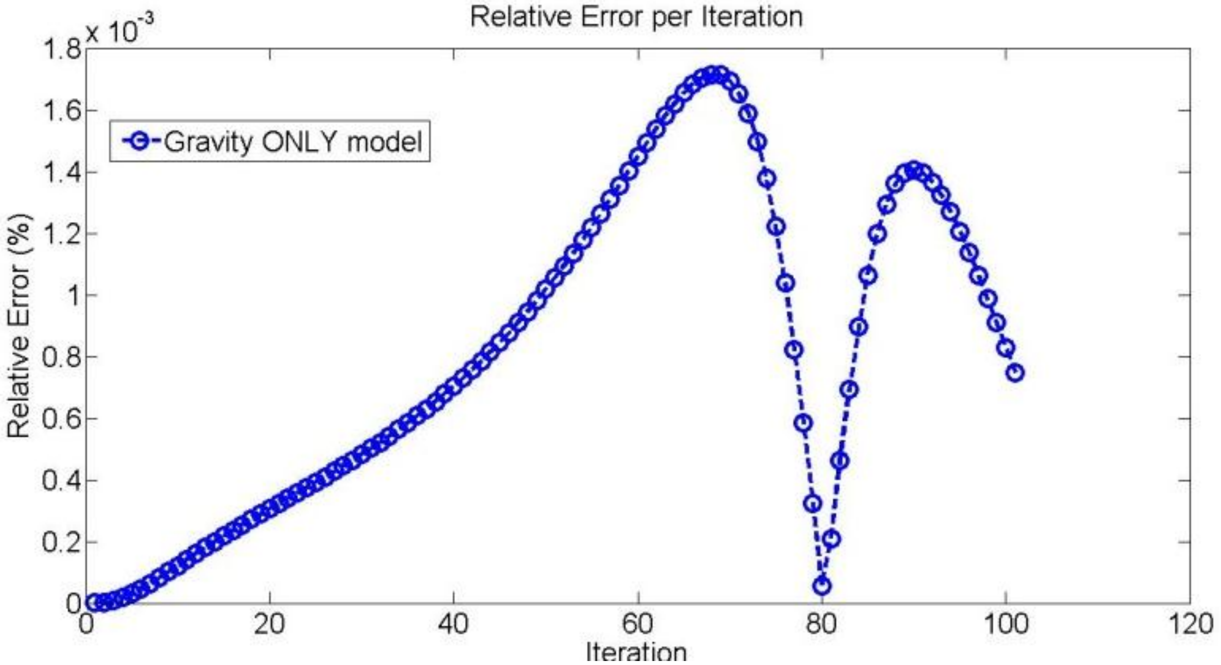
An orbit with inclination is created by giving a velocity component into the z–direction. Likewise, the magnitude of the velocity for an inclination orbit still determines the shape of the orbit.

The error associated with the propagation of an orbit is sinusoidal. This is identical to the sinusoidal calculation of the orbit. Every time that an orbit is modeled, there is an automatic error offset occurs because of the initial input errors. The relative error for each model is given below in Figure 20.

(a)


Relative Error per Iteration
x 10^-4
6
5
4
3
2
1
0
Relative Error (%)
0
10
20
30
40
50
60
Iteration
Gravity ONLY model

(b)


Relative Error per Iteration
x 10^-3
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Relative Error (%)
0
20
40
60
80
100
120
Iteration
Gravity ONLY model

(c)

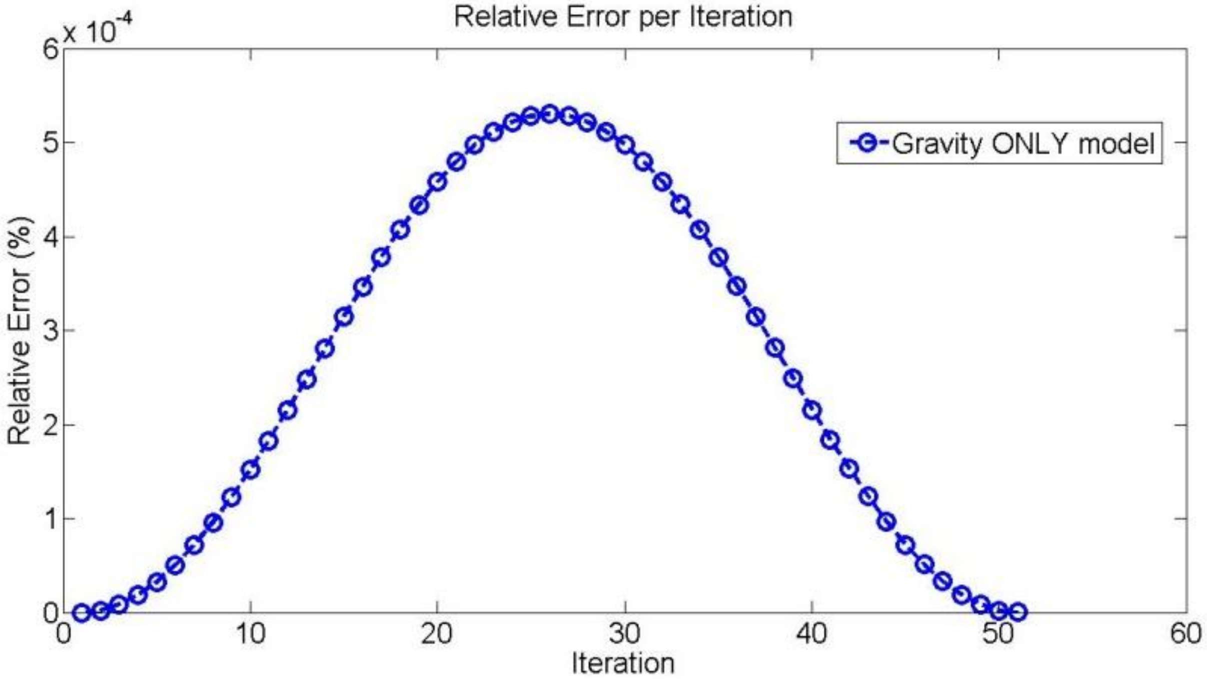


Figure 20: Relative Error of Gravity Models over One Orbit
(a) Circular Orbit (b) Elliptical Orbit (c) Circular Orbit with Inclination

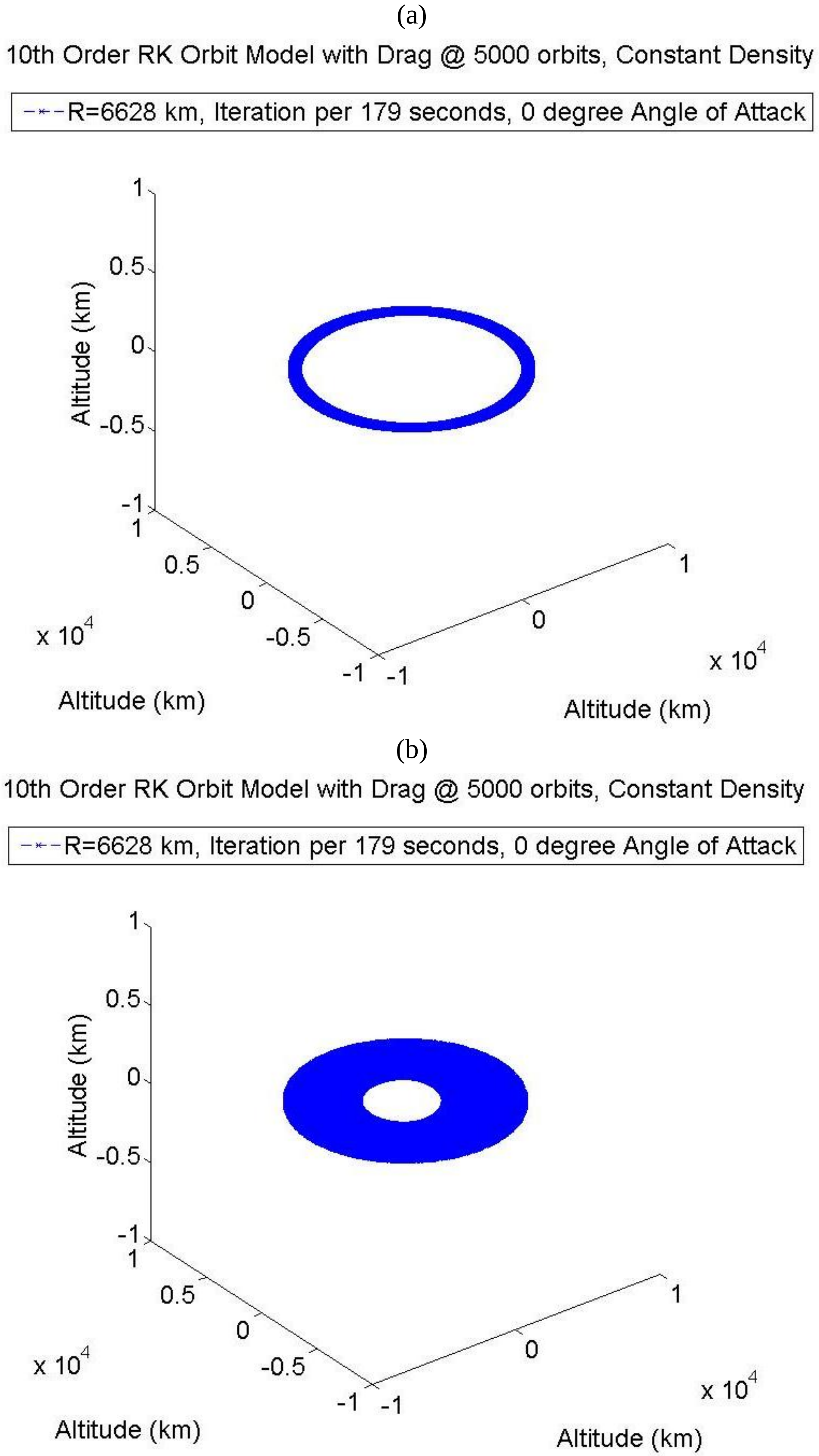
Figure 20 (b) has a slight abnormality compared with the other errors which form a sinusoidal error. This is possibly due to the elliptical orbit shape while the others where circular, but no in depth research was done to explain this.

iii. Gravity and Drag

Once the gravity model was built, additional changes were required in order to model an object that experiences drag. Before the creation of this model that represented gravity and drag effects on an orbiting object the coefficient of drag was assumed to be the accepted value of 2.2; however, once this model was built the coefficient varied with the angle of attack.

$$\begin{bmatrix}\dot{r}\\ \ddot{r}\end{bmatrix} = \begin{bmatrix} v_x\boldsymbol{x} \\ v_y\boldsymbol{y} \\ v_z\boldsymbol{z} \\ \left(-\frac{GM}{r^3}x - \frac{C_dA\rho v_{rel}}{2m}(v_x + \omega_e y)\right)\boldsymbol{x} \\ \left(-\frac{GM}{r^3}y - \frac{C_dA\rho v_{rel}}{2m}(v_y - \omega_e x)\right)\boldsymbol{y} \\ \left(-\frac{GM}{r^3}z\right)\boldsymbol{z} \end{bmatrix} \quad where\ v_{rel} = |\boldsymbol{v_{rel}}| \qquad \text{Eq. 25}$$

First, it is assumed that the atmosphere is modeled with a constant atmosphere. Figure 21 illustrates the effects of a constant atmosphere on a satellite. Figure 21(a) depicts an atmospheric model that has a low constant density (valued at 300 km) the value for density in this model is $1.9578\times 10^{-11}\frac{kg}{m^3}$; whereas (b) illustrates a model that has a higher constant density (valued at 200 km) the value for density in this model is $2.6265\times 10^{-10}\frac{km}{m^3}$. It can be observed that the density model formulated from the higher altitude created less force than the lower altitude density model. This is accurate with basic assumptions that the satellite's orbit will degrade faster as the satellite enters further into the atmosphere.

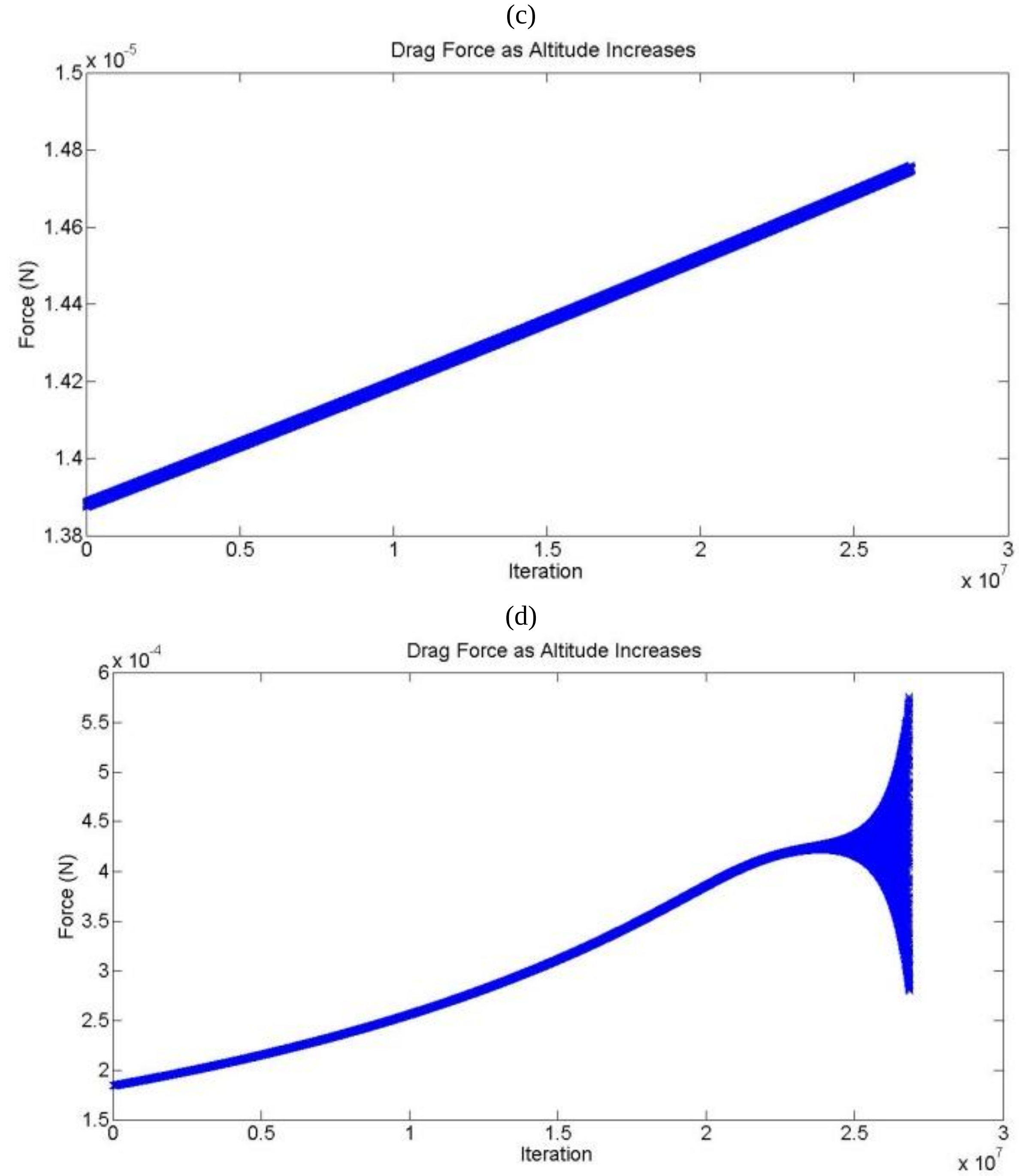


(a)

10th Order RK Orbit Model with Drag @ 5000 orbits, Constant Density
R=6628 km, Iteration per 179 seconds, 0 degree Angle of Attack
Altitude (km)
1
0.5
0
-0.5
-1
1
0.5
0
-0.5
-1
x 10^4
Altitude (km)
-1
0
1
x 10^4
Altitude (km)

(b)

10th Order RK Orbit Model with Drag @ 5000 orbits, Constant Density
R=6628 km, Iteration per 179 seconds, 0 degree Angle of Attack
Altitude (km)
1
0.5
0
-0.5
-1
1
0.5
0
-0.5
-1
x 10^4
Altitude (km)
-1
0
1
x 10^4
Altitude (km)



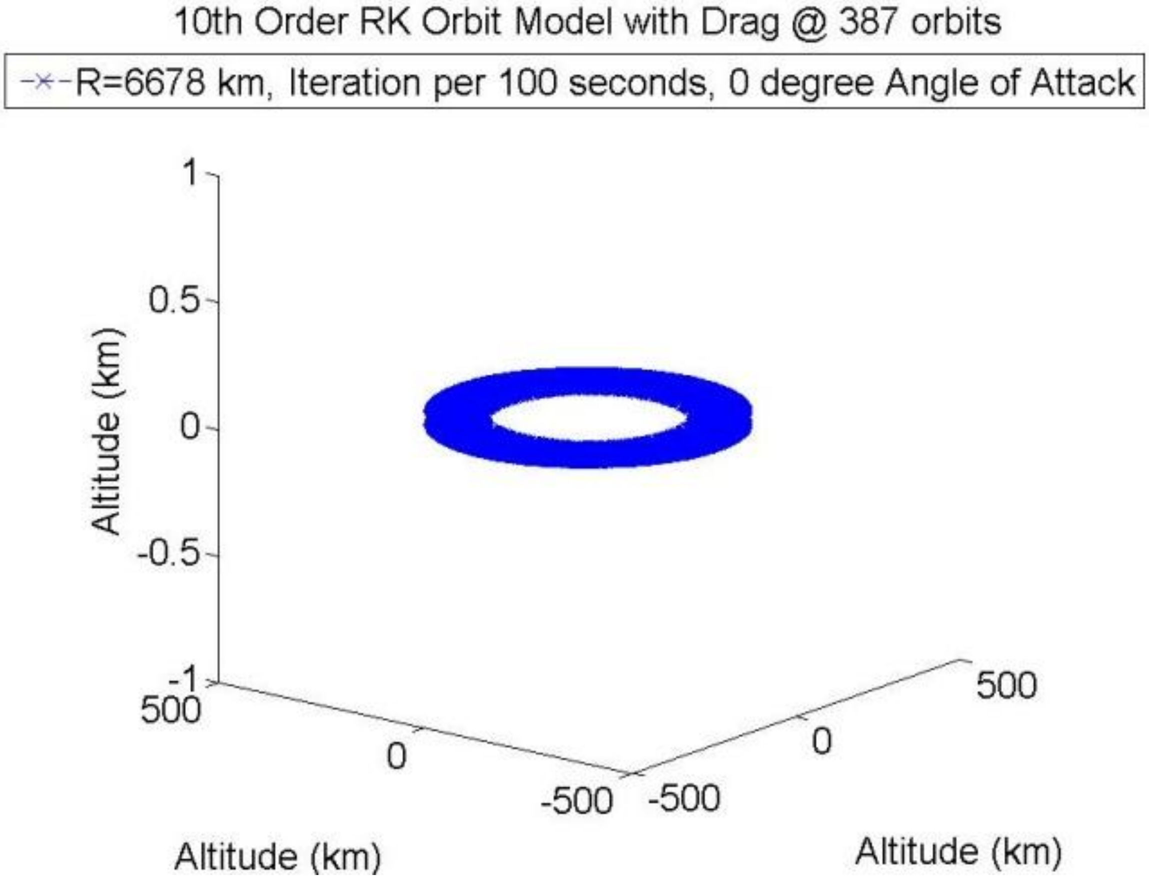
(c)

(d)

Figure 21: Orbit Model with a constant atmosphere
(a) Constant density from 300 km (b) Constant density from 200 km
(c) Drag with Constant density at 300 km (d) Drag with Constant density at 200 km

Second, orbit models with initial altitudes between 200 km and 300 km were created with an atmospheric model created from the USSA-76. Figure 22 models the motion of an orbiting object with a varying density model. Figure 22 (a) depicts a model that has a starting altitude of 300 km, whereas (b) illustrates model that has an initial altitude of 200 km. It can be observed that the lower altitude model orbits significantly fewer orbits than the model beginning with the initial higher altitude. This is significant because it demonstrates the atmospheric model plays a substantial role in the modeling of an orbiting object. If the assumption what made that there is constant density atmosphere there would be two consequences. Depending on the assumed density value the amount of time orbiting the Earth would be significantly different and the forces experienced by both drag and lift would be inaccurate.

(a)



(b)

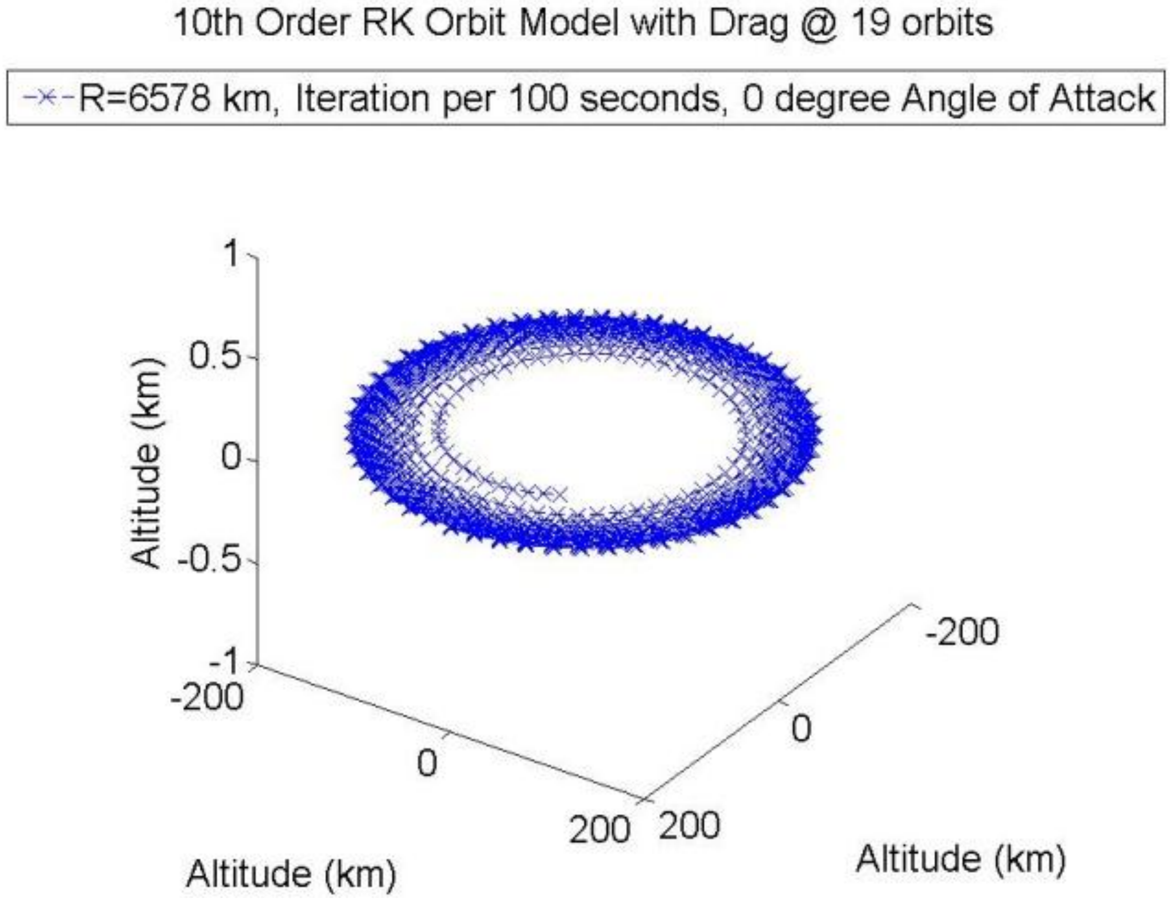


Figure 22: Orbit Model by varying altitude

The accuracy of drag and gravity models were analyzed using the definition of work and conservation of energy.

$$W = \int_1^2 F dx \qquad \text{Eq. 26}$$

where $W = Work$

$F = Force$

The total value for work can be broken down into the sum of two integrals, x and y components.

$$W = W_x + W_y = \int_1^2 F_x dx + \int_1^2 F_y dy \qquad \text{Eq. 27}$$

where $W_x = Work\ in\ x\ direction$

$W_y = Work\ in\ y\ direction$

$F_x = Force\ in\ x\ direction$

$F_y = Force\ in\ y\ direction$

The force for gravity and drag can be found from the acceleration components in Eq. 25 and the mass of the satellite. After completing the integral the final expression for work is represented by Eq. 28.

$$\left[-\frac{GMm}{2r^3}x^2 - \frac{C_d S\rho v_{rel}}{2}(v_x + \omega y)x\right]_{x_1}^{x_2} + \left[-\frac{GMm}{2r^3}y^2 - \frac{C_d S\rho v_{rel}}{2}(v_y - \omega x)y\right]_{y_1}^{y_2} \qquad \text{Eq. 28}$$

The work that is done on the satellite by the atmosphere should equal the amount of energy change of the satellite. The energy equation to quantify that loose is expressed by Eq. 29.

$$\Delta E = \left(\left[\frac{v_f^2}{2} - \frac{\mu}{r_f}\right] - \left[\frac{v_i^2}{2} - \frac{\mu}{r_i}\right]\right)m \qquad \text{Eq. 29}$$

where $\Delta E = change\ in\ energy\ between\ states$

$v_f, v_i = final/initial\ \ velocity$

$r_f, r_i = final/initial\ \ radius$

If the values between the both the work and energy equation are approximately equal then the model has an acceptable accuracy. Table 6 compares the energy and work of a cubesat after a finite number of orbits.

Table 6: Work versus Energy values for 50 km intervals

R	Orbits	Energy (KJ)	Work (KJ)
6578	10	0.0207	-0.0203
	40	0.0984	-0.0956
	75	0.2517	-0.2398
	90	0.3636	-0.3506
	95	0.4190	-0.4076
	100	0.5044	-0.4948
	105	-8.3005	-30.6728

The difference in sign between work and energy are not significant. The difference in sign comes from the reference frame for energy. The zero point reference from for the energy equation is infinity. The last energy and work calculation possible comes from the fact that the satellite is at the end of its orbital life.

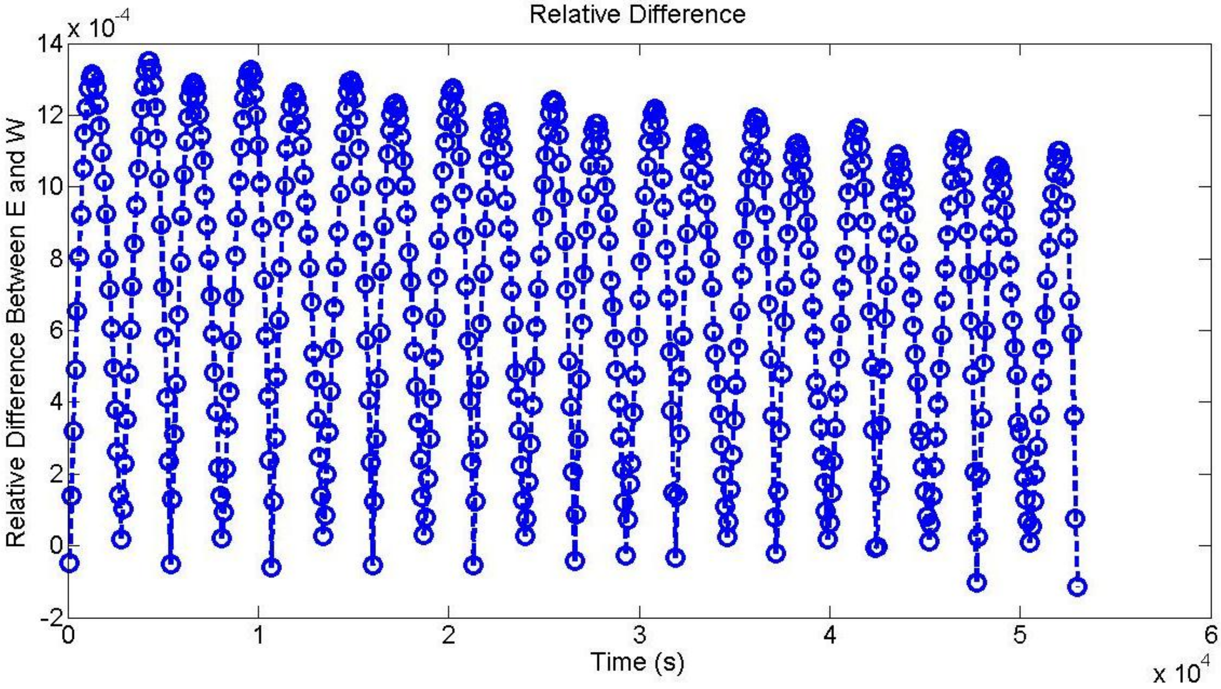


Figure 23: Relative Error between Work and Energy Values

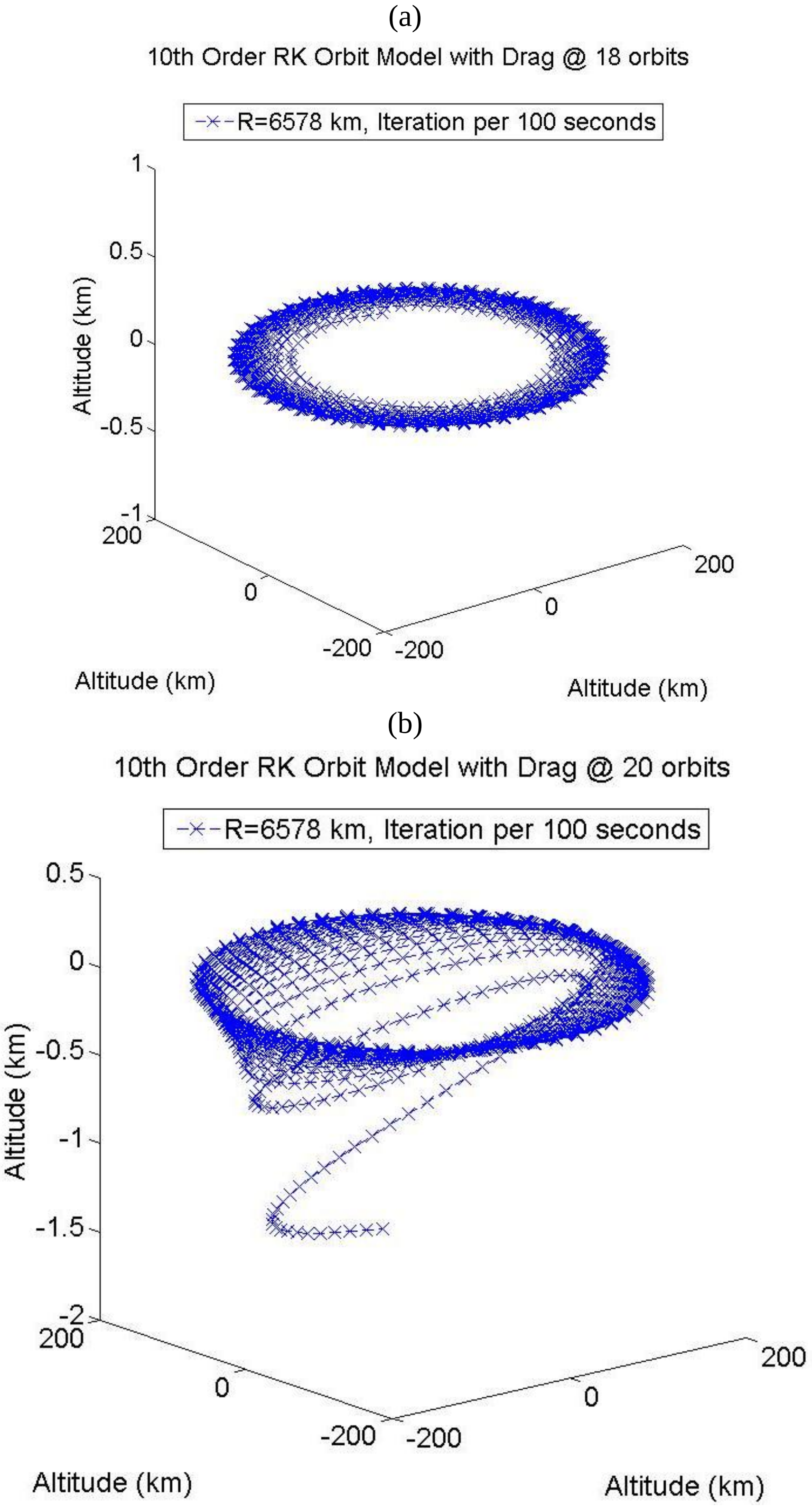
Figure 23 is the measurement of error associated with each iteration's energy and work calculation as the satellite deorbits.

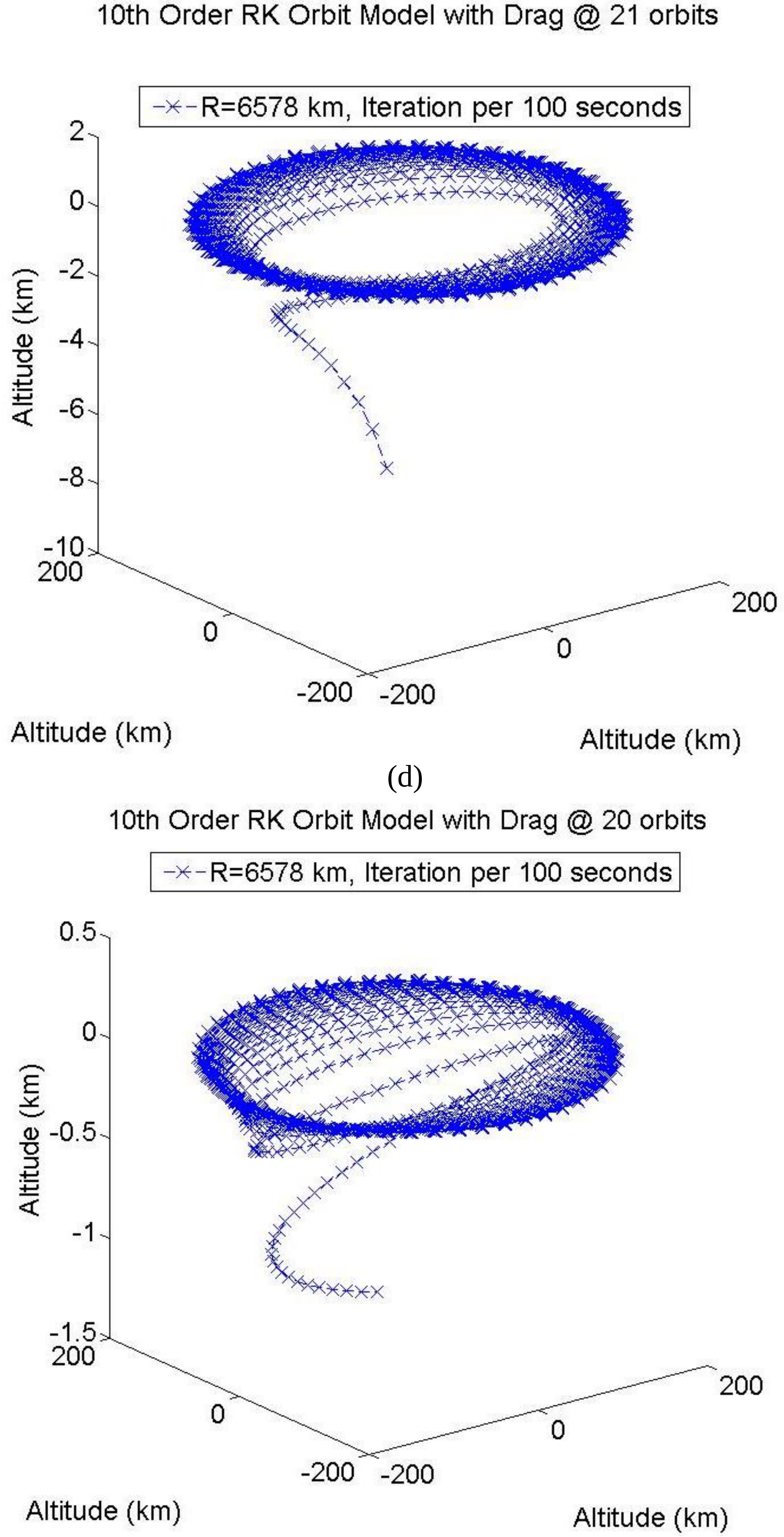
iv. Gravity, Drag, and Lift

Final changes were required in order to model an orbiting object that is perturbed by atmospheric density in the form of lift and drag. Therefore, Eq. 25 is transformed into Eq. 30. The following models were created with the assumptions that the coefficients of drag and lift varied with the angle of attack and that the accommodation coefficient was 0.99.

$$\begin{bmatrix} \dot{r} \\ \ddot{r} \end{bmatrix} = \begin{bmatrix} v_x \boldsymbol{x} \\ v_y \boldsymbol{y} \\ v_z \boldsymbol{z} \\ \left(-\frac{GM}{r^3}x - \frac{C_d A \rho v_{rel}}{2m}(v_x + \omega_e y) - \frac{C_l A \rho v_{rel}}{2m}(v_x n_z v_z)\right)\boldsymbol{x} \\ \left(-\frac{GM}{r^3}y - \frac{C_d A \rho v_{rel}}{2m}(v_y - \omega_e x) - \frac{C_l A \rho v_{rel}}{2m}(v_y n_z v_z)\right)\boldsymbol{y} \\ \left(-\frac{GM}{r^3}z - \frac{C_l A \rho v_{rel}}{2m}({v_y}^2 n_z - {v_x}^2 n_z)\right)\boldsymbol{z} \end{bmatrix} \qquad \text{Eq. 30}$$

The models that were created were analyzed for accuracy by visible inspection. In essence, if the angle of rotation was zero or 90 then there should be no lift and mimic the models created by the gravity and drag models, and the change in inclination should appear to be as a maximum at the 45 degree rotation.


(a)
(b)
10th Order RK Orbit Model with Drag @ 18 orbits
R=6578 km, Iteration per 100 seconds
Altitude (km)
1
0.5
0
-0.5
-1
200
0
-200
-200
0
200
Altitude (km)
Altitude (km)
10th Order RK Orbit Model with Drag @ 20 orbits
R=6578 km, Iteration per 100 seconds
Altitude (km)
0.5
0
-0.5
-1
-1.5
-2
200
0
-200
-200
0
200
Altitude (km)
Altitude (km)



(c)

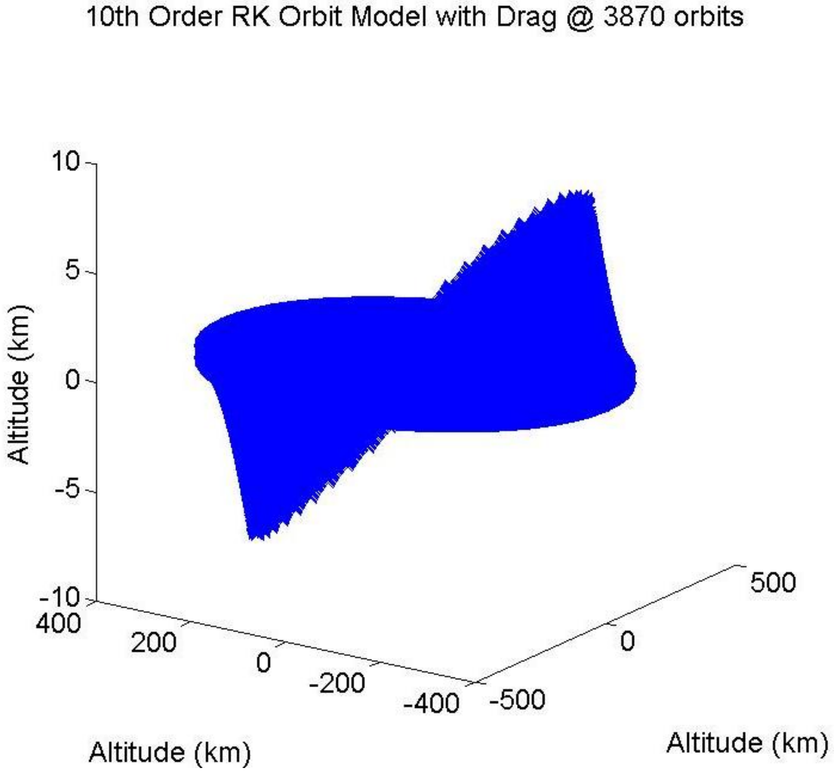
(d)

Figure 24: Orbit Models with Differing Angle of Attacks (a) 0 degrees (b) 30 degrees (c) 45 degrees (d) 60 degrees

Figure 24 shows the slight inclination change due to lift and the effects of drag at LEO satellites. At LEO there is more effect from drag than lift so the satellite still deorbits and re-enters. The rapidness of the deorbit restricts the maneuver to small inclination changes.

Figure 25 illustrates that at higher altitudes the orbiting object is able to stay in orbit longer allowing for larger inclination changes.

(a)



(b)

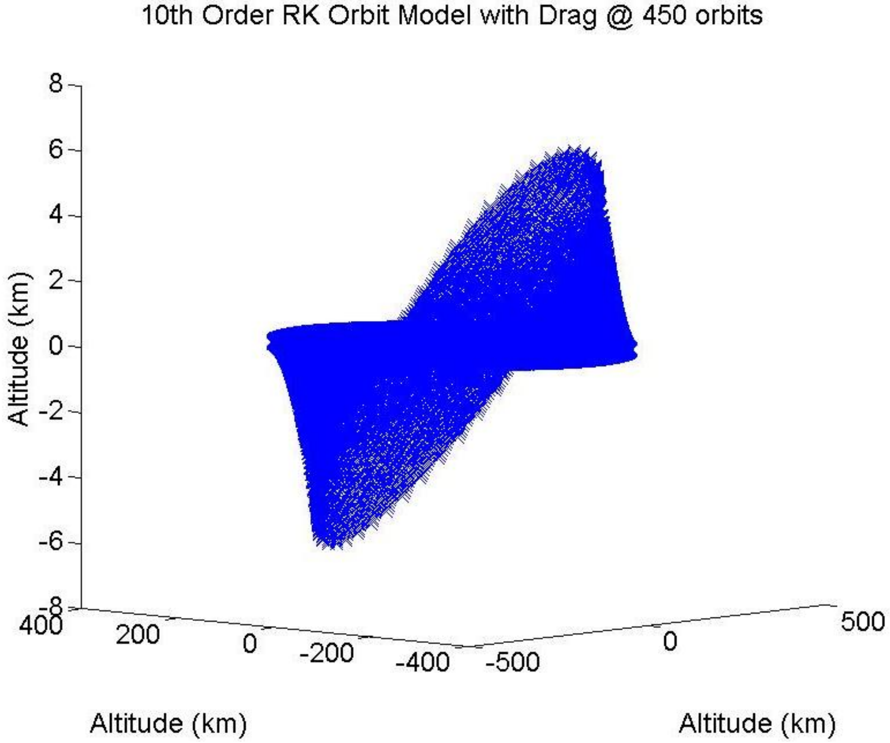


Figure 25: Orbits with prolonged exposer to Lift Forces
(a) Altitude of 400 km for 3870 Orbits (b) Altitude of 300 km for 1371 Orbits (α=0.9)

Figure 26 elaborates on the increase in orbital lifetime of an orbiting cubesat as the initial altitude increases.

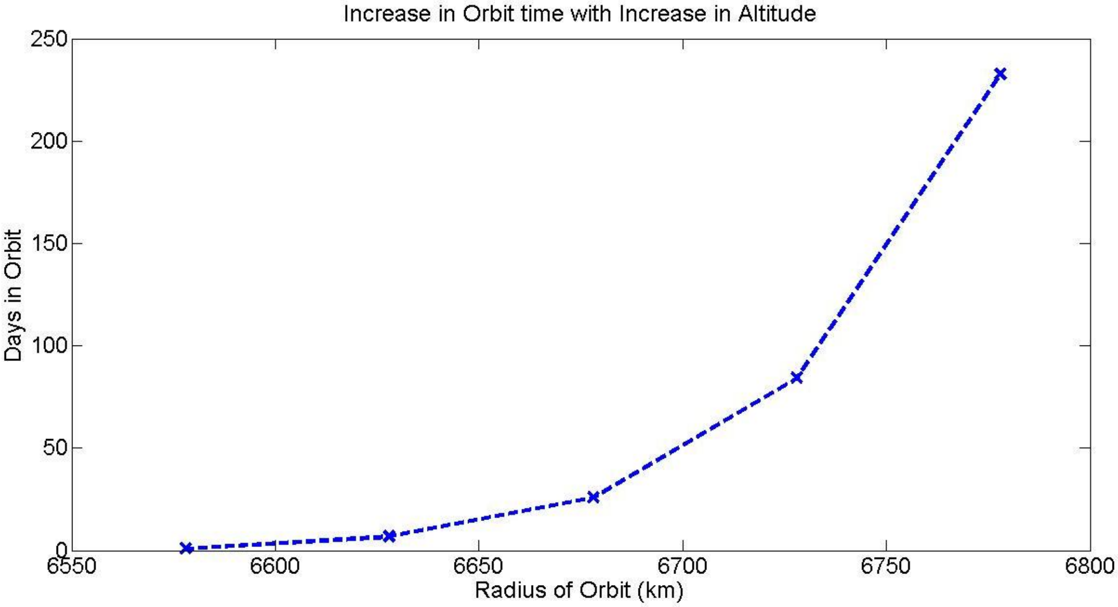


Figure 26: Orbital Time as Altitude Increases (α=0.9)

b. Cubesat Inclination Analysis: Constant Angle of Attack

Once the model was created and verified, it was then used to analyze the difference between the requirements to change inclination with propulsive maneuvers versus aerodynamic assistance. Traditionally when noncoplanar maneuvers are done they required a Δυ which is calculated from Eq. 31.

$$\Delta v = 2vsin\left(\frac{\theta}{2}\right) \qquad \text{Eq. 31}$$

where $\Delta v = normal\ velocity\ applied\ to\ change\ planes$
$v = velocity\ at\ point\ in\ orbit$
$\theta = angle\ of\ inclination\ change$

As inclination requirements get larger the magnitude of the Δυ required approaches and exceeds the magnitude of the orbital velocity (approximately 7 $\frac{km}{s}$) as seen by Figure 27 [12]. Figure 27 was created assuming that the orbiting position and velocity of the cubesat was 6778 km and 7.6686 $\frac{km}{s}$.

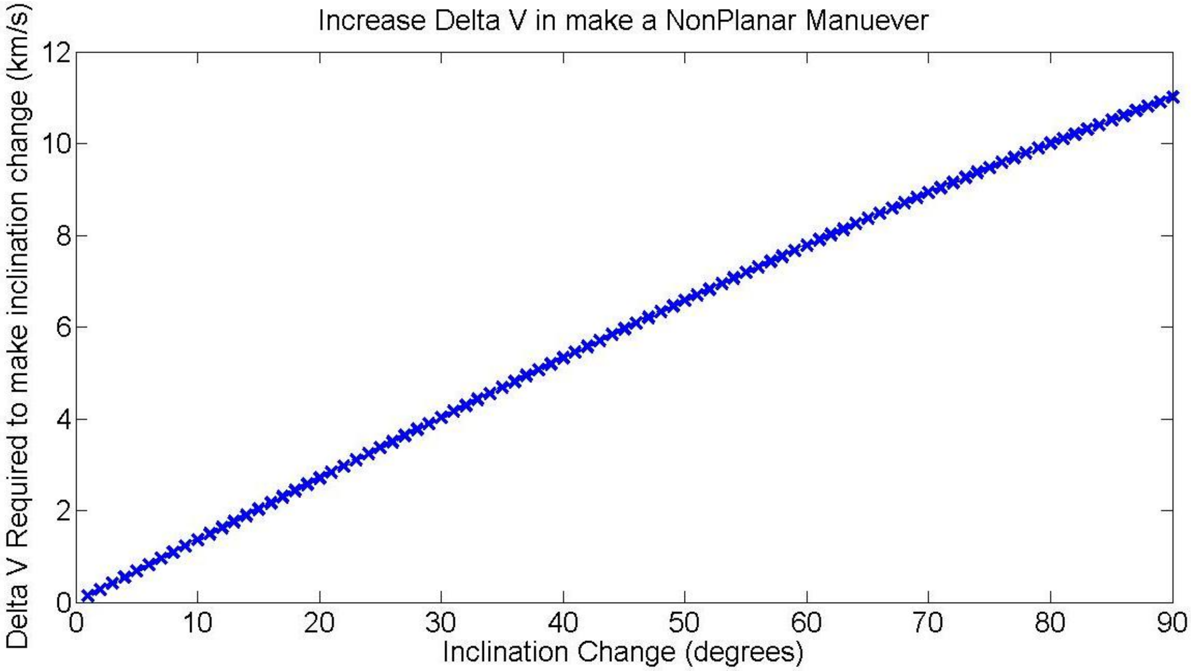


Figure 27: Δυ Required for Inclination Changes

Using the geometry of the spherical coordinate system, which is observed in Figure 28, the angle between the z-axis and the radius vector is found using Eq. 32 [28].

$$\phi = cos^{-1}\left(\frac{z}{r}\right) \qquad \text{Eq. 32}$$

where $\phi = angle\ between\ z\ axis\ and\ vector$
$z = magnitude\ in\ z - direction$
$r = length\ of\ radius\ vector$

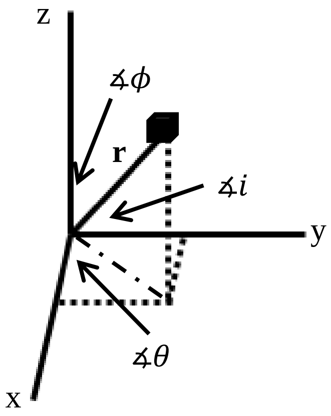


Figure 28: Spherical Coordinate System Geometry

The inclination of the orbit, caused by the aerodynamic effects, is then solved for using the angles from the spherical coordinate system.

$$i = 90 - \phi \qquad \text{Eq. 33}$$

where $i = angle\ of\ inclination$

All models created for the duration of the paper will hold the assumptions that the coefficients of drag and lift varies with the attack angle and accommodation coefficient.

The ending orbital inclination of the model was measured at based on the final position of five orbital models. For these models the angle of attack was set at a constant 45 degrees, the accommodation coefficient was varied. Figure 29 represents the final inclination, after each orbit life, with an accommodation coefficient of one.

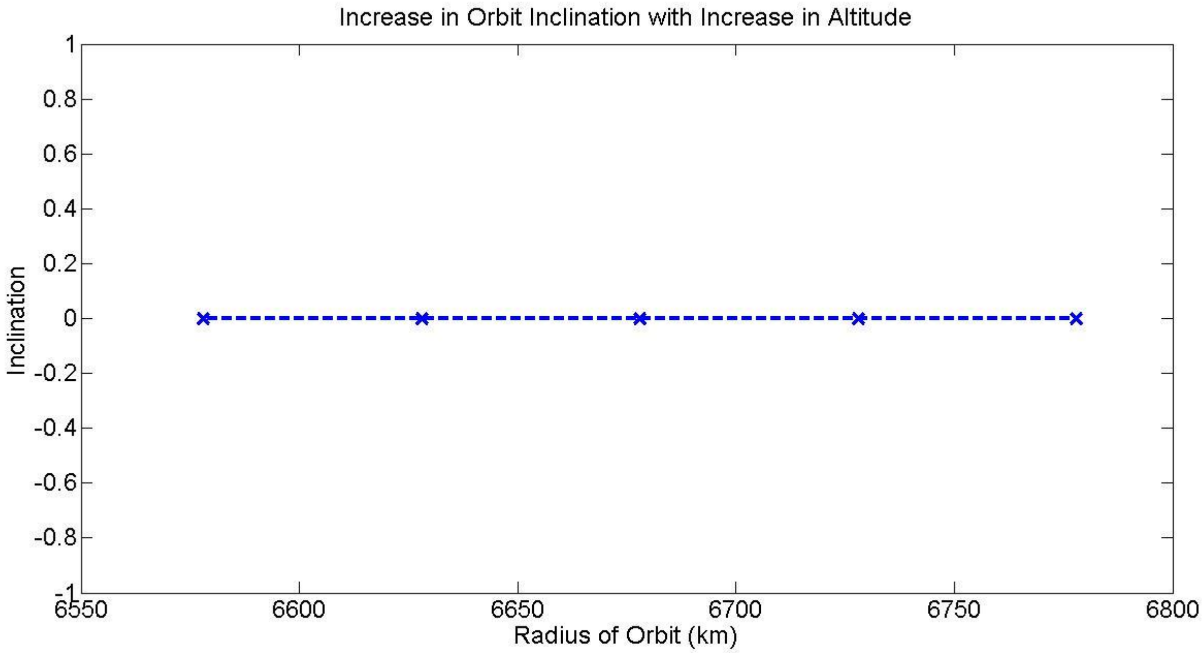


Figure 29: Final Inclination of Orbit Model based on Initial Orbital Radius (α=1)

Figure 30 represents the final inclination of each orbit based on maximum z altitude over the entirety of the satellites life. For these models the accommodation coefficient was set at 0.90.

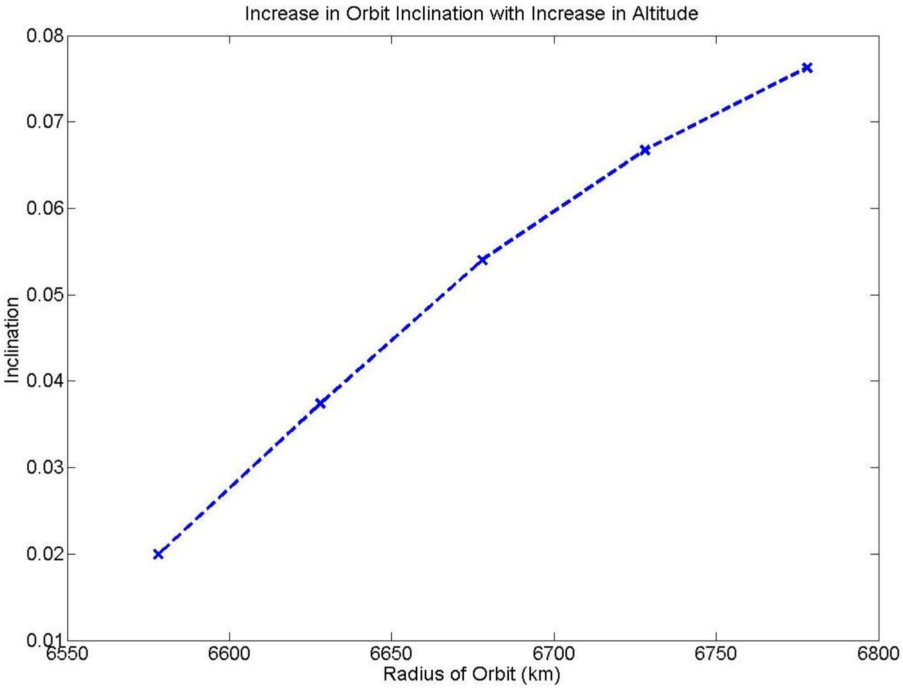


Figure 30: Final Inclination of Orbit Model based on Initial Orbital Radius (α=0.9)

Figure 31 illustrates the lift and drag forces experience by each of the five orbiting altitude scenarios.

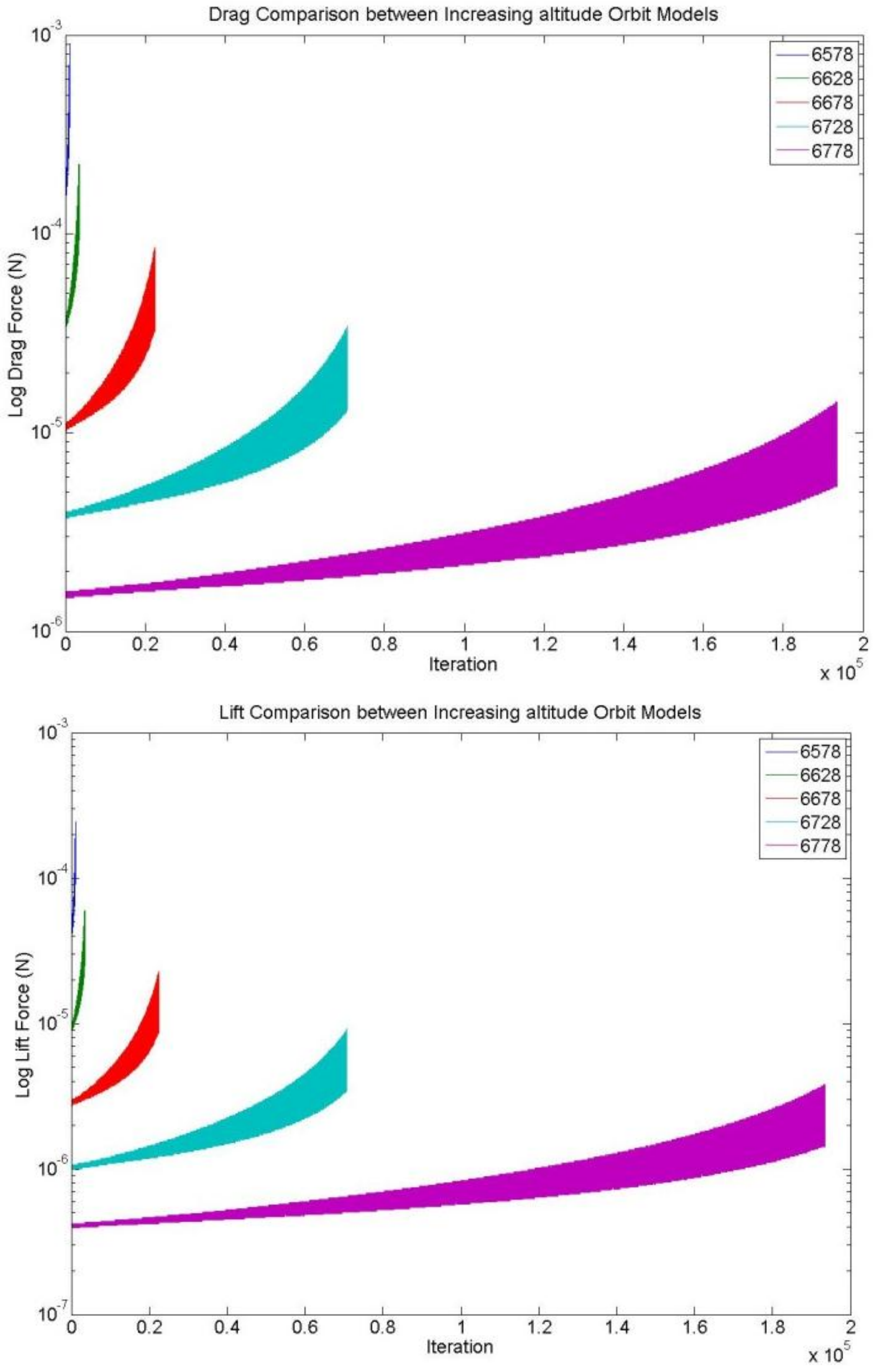


Figure 31: Drag and Lift Force Experienced During Inclination Change Model

Upon analyzing the trend seen in Figure 9 and the support of an increased inclination between Figure 29 and Figure 30 it was decided to continue analyzing the trend of the effect of the accommodation coefficient on lift and inclination change. Figure 32 and Figure 33 elaborates on the effect of lift on inclination with an accommodation coefficient of 0.5.

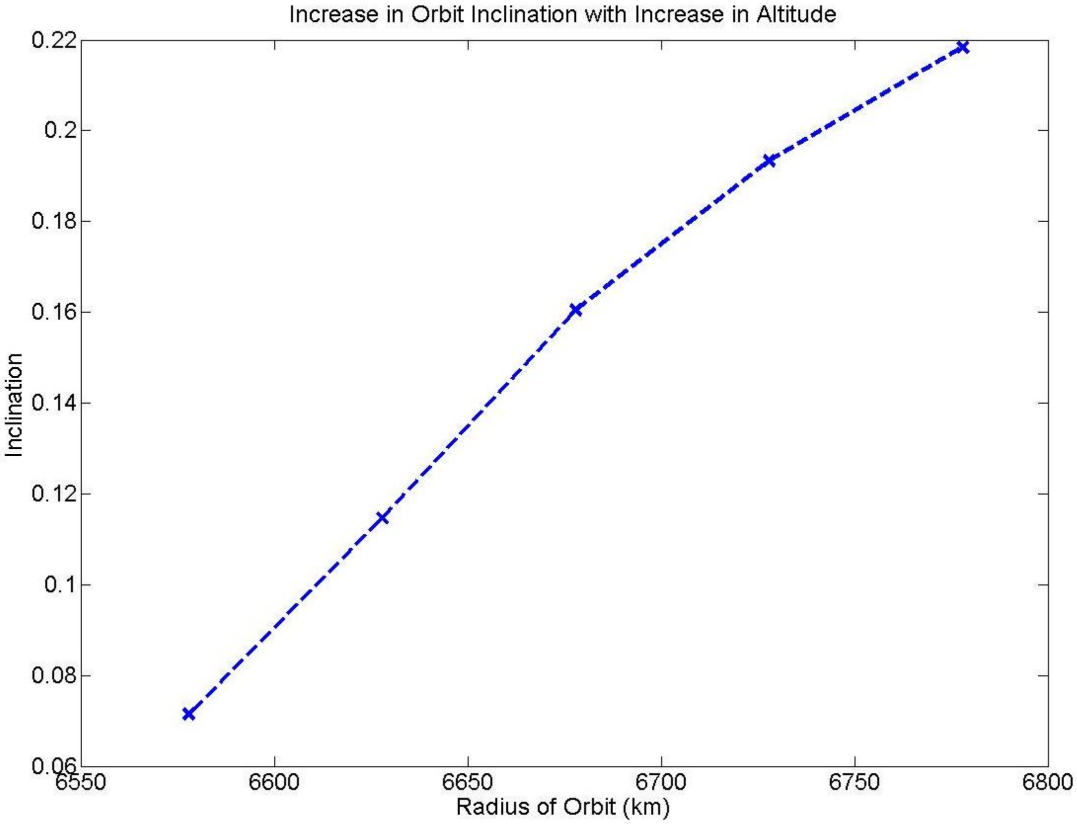


Figure 32: Orbital Inclination Change varying with initial Radius (α=0.5)

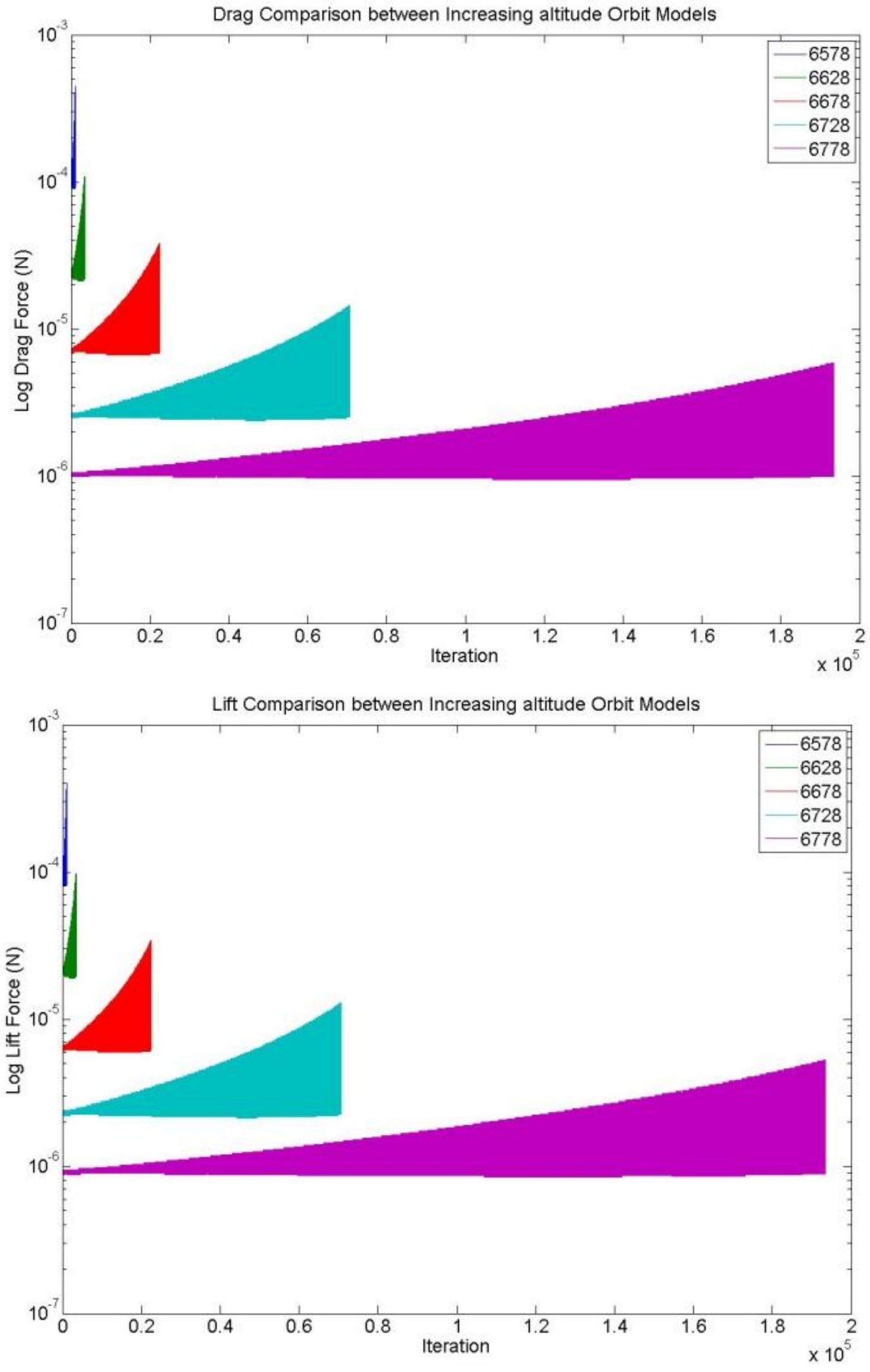


Figure 33: Comparison of Drag and Lift Force versus starting radial altitude (α=0.5)

Figure 34 and Figure 35 illustrates the effect of lift on inclination with an accommodation coefficient of zero.

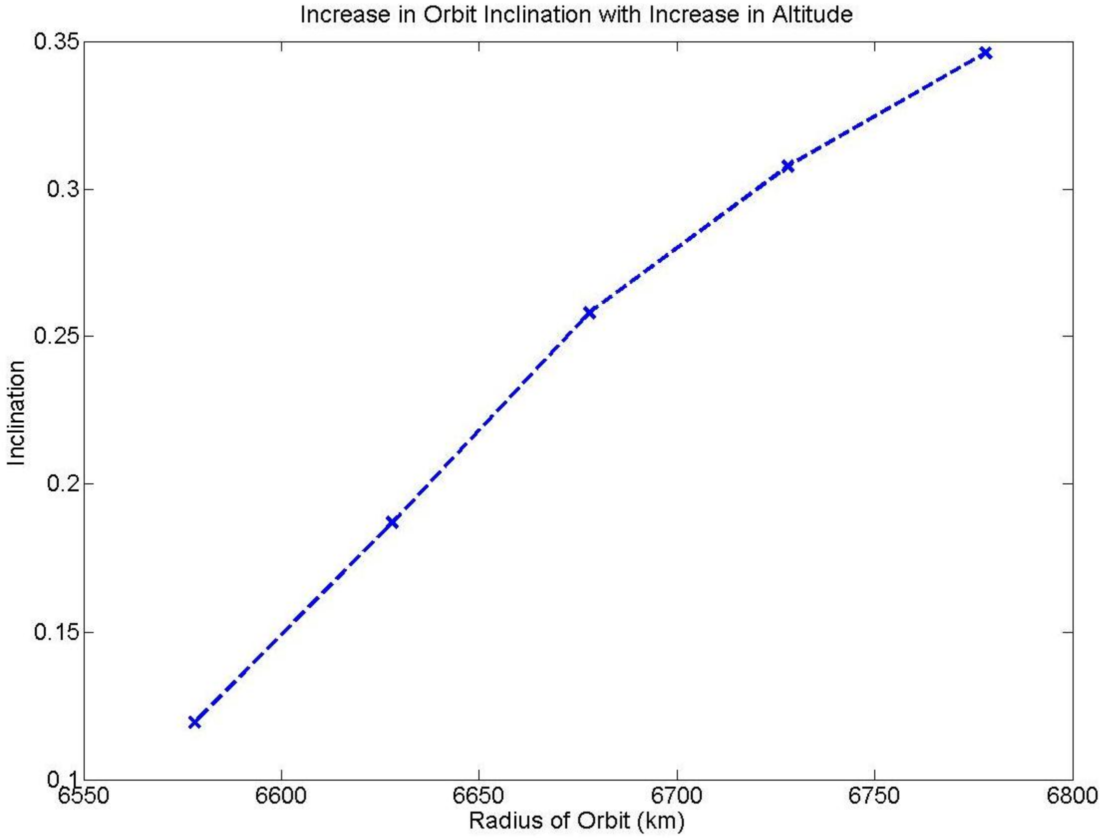


Figure 34: Orbital Inclination Change varying with initial Radius (α=0)

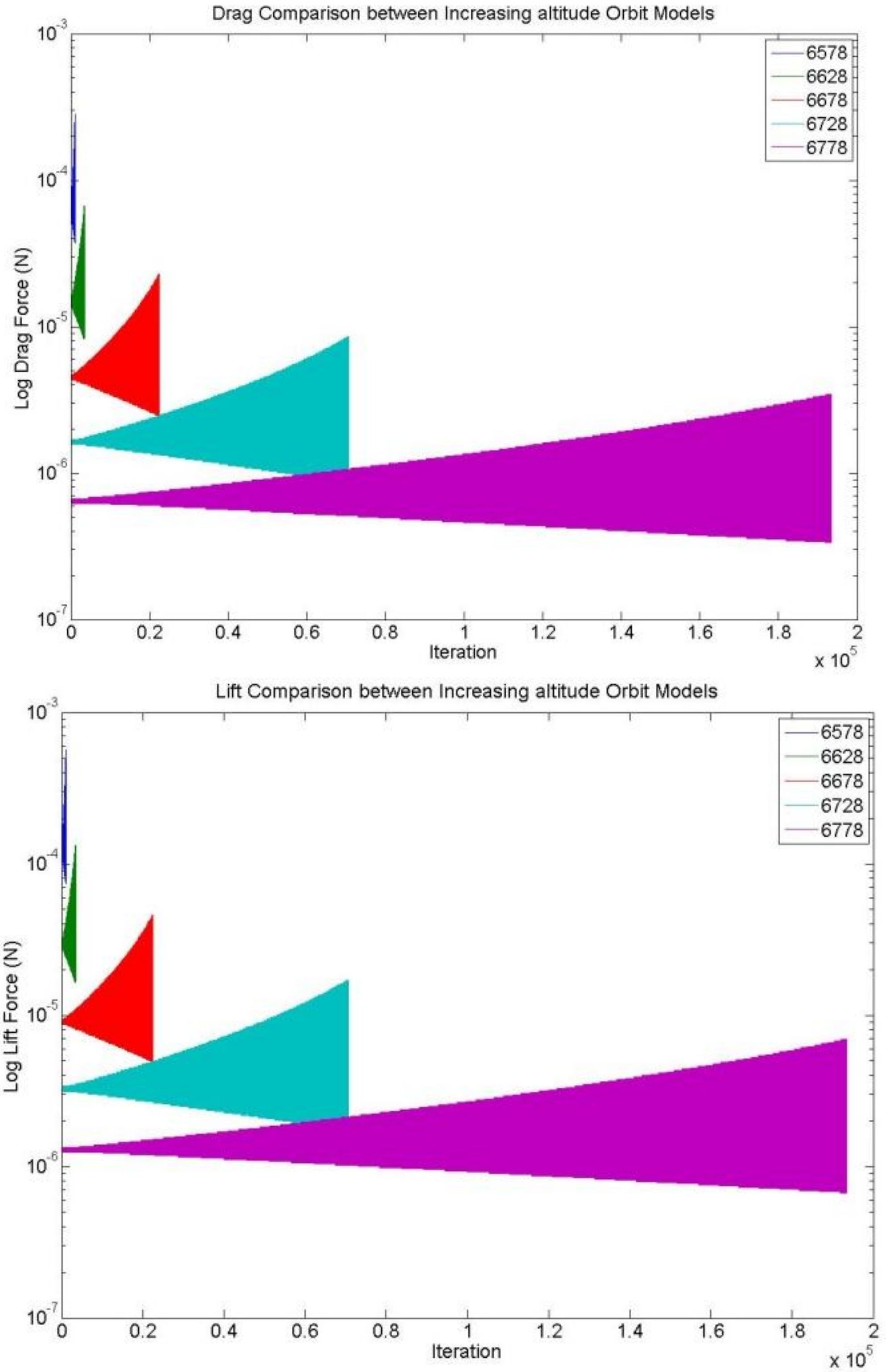


Figure 35: Comparison of Drag and Lift Force versus starting radial altitude (α=0)

Once, the difference in lift force and inclination change (due to the accommodation coefficient) was observed a final test was done to measure the extent of the atmosphere property effects on the inclination change. The accommodation coefficient was zero, and the total time of measurement was 20 orbits. Despite an accommodation coefficient of this value being unlikely it served as the best possible change in inclination so it was chosen as the value to do a basis comparison.

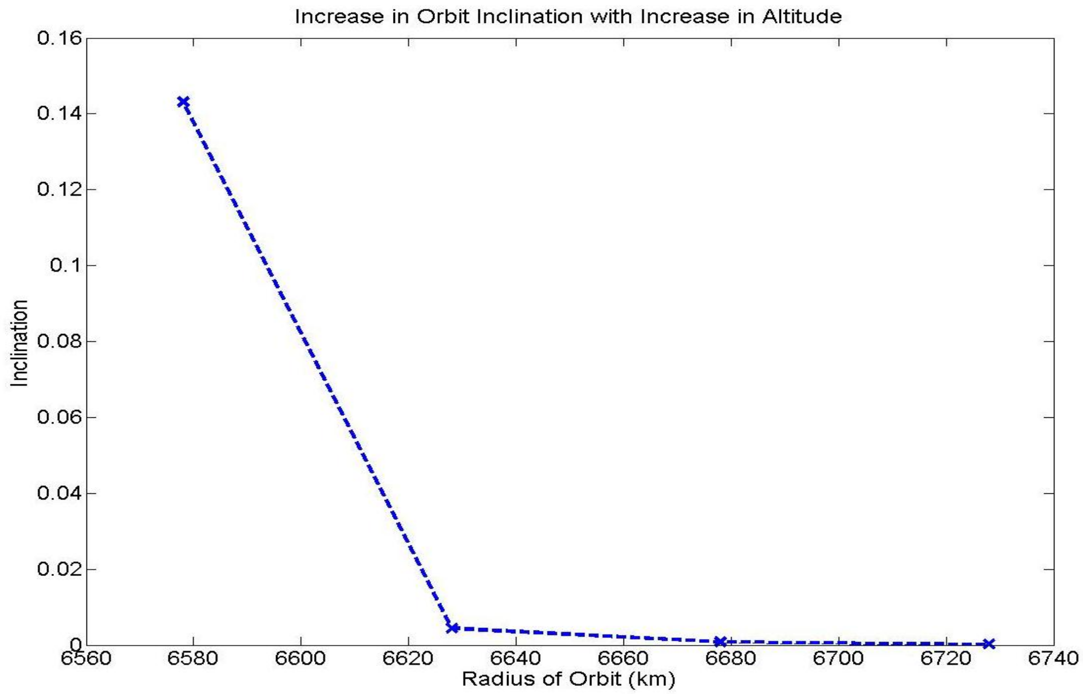


Figure 36: Final Inclination of Orbital Models after 20 Orbits (α=0)

From Figure 36, it is observed that it creates inclination changes at low altitudes, which matches theory due to the increased density of the atmosphere.

c. Inclination Analysis: Scheduled Angle of Attack

For these models the angle of attack was scheduled to rotate between 45, 0, and negative and varied with accommodation coefficient values. The angle of attack rotation was assumed to be instantaneous.

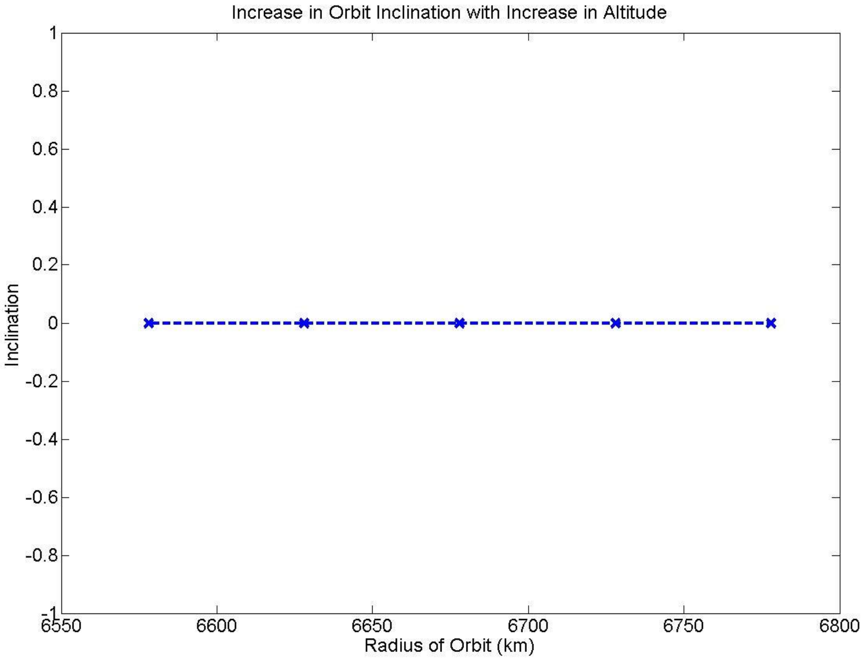


Figure 37: Final Inclination of Orbit Model based on Initial Orbital Radius (α=1)

The final inclination of each orbit is based on maximum z-dimension altitude over the entirety of the satellites life. Figure 38 and Figure 39 show the inclination and Lift force for models created with an accommodation coefficient was set at 0.90.

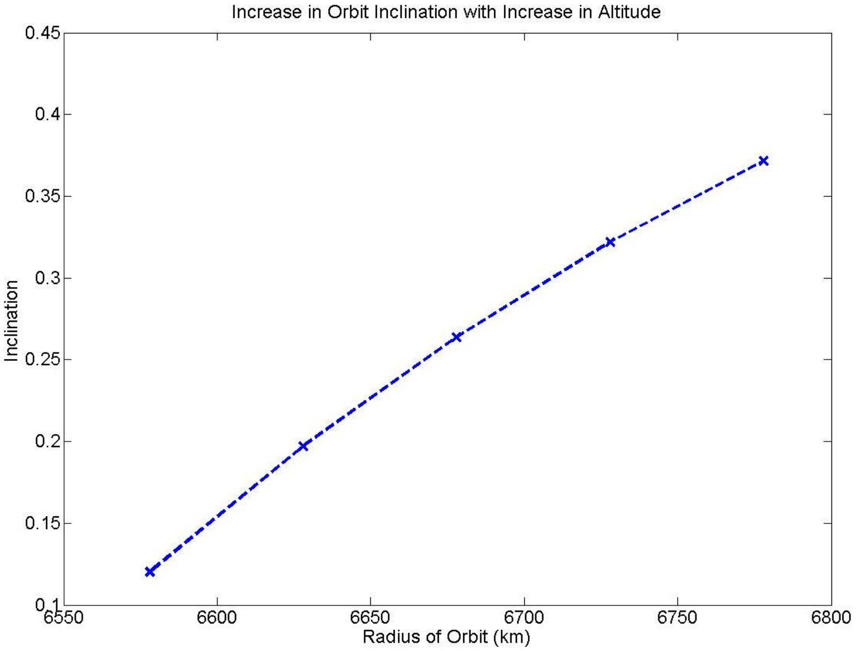


Figure 38: Final Inclination of Orbit Model based on Initial Orbital Radius (α=0.9)

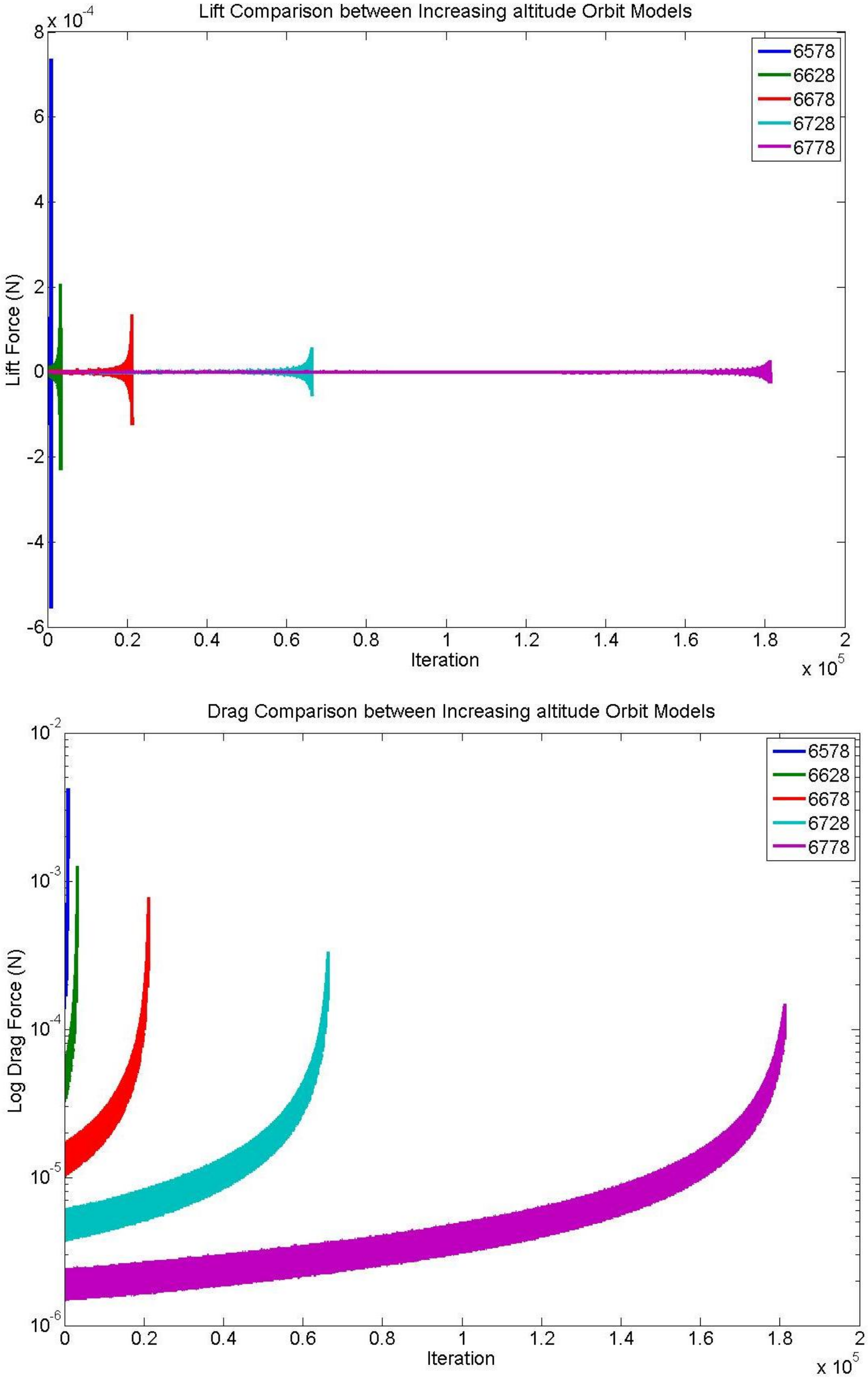


Figure 39: Drag and Lift Force Experienced During Inclination Change Model

Figure 40 and Figure 41 elaborates on the effect of lift on inclination with an accommodation coefficient of 0.5.

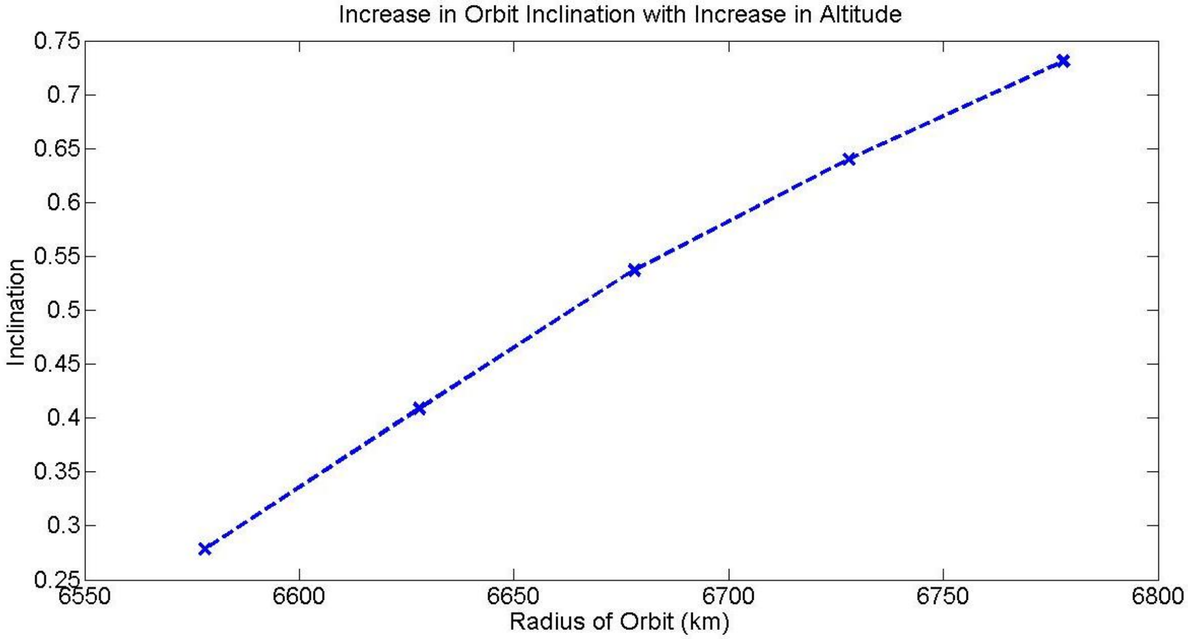
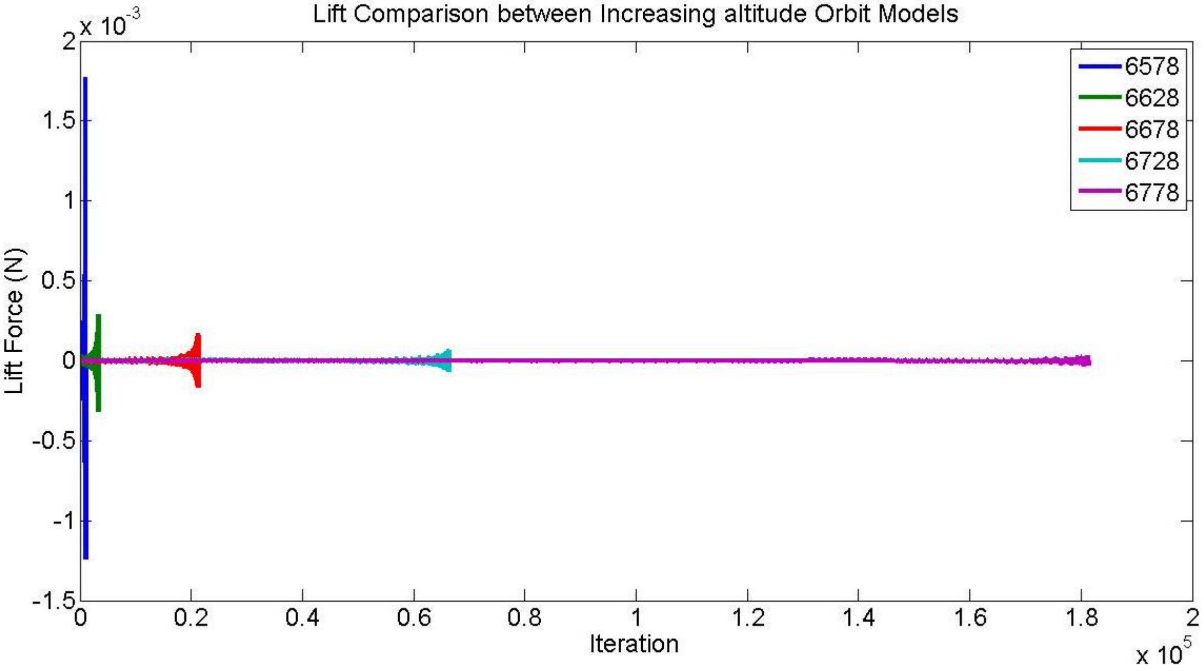


Figure 40: Orbital Inclination Change varying with initial Radius (α=0.5)



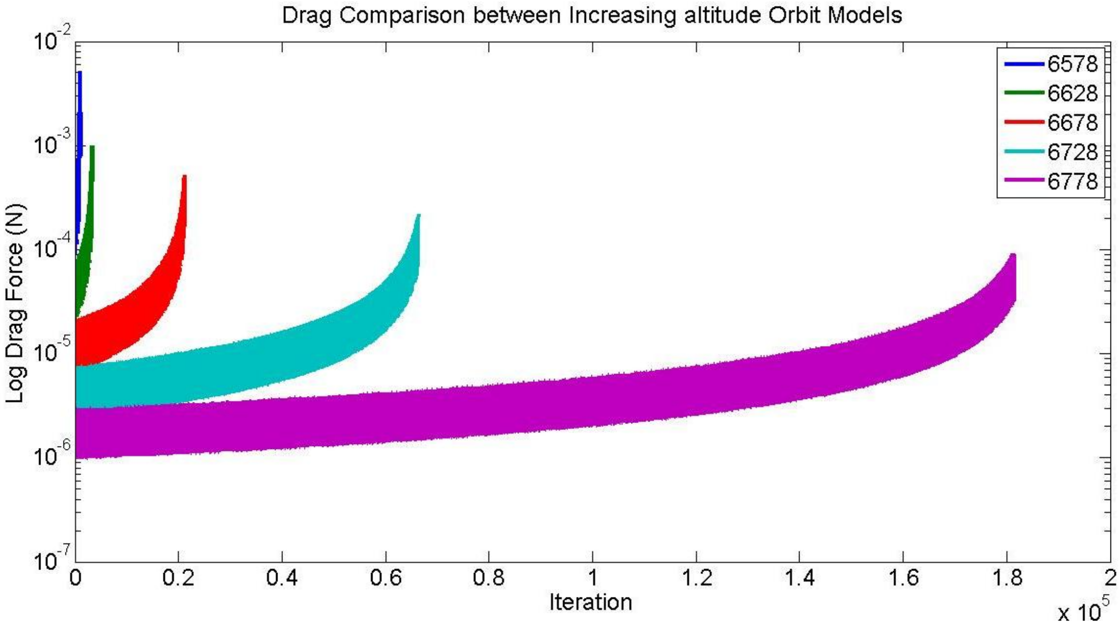


Figure 41: Comparison of Drag and Lift Force versus starting radial altitude (α=0.5)

Figure 42 and Figure 43 illustrates the effect of lift on inclination with an accommodation coefficient of zero.

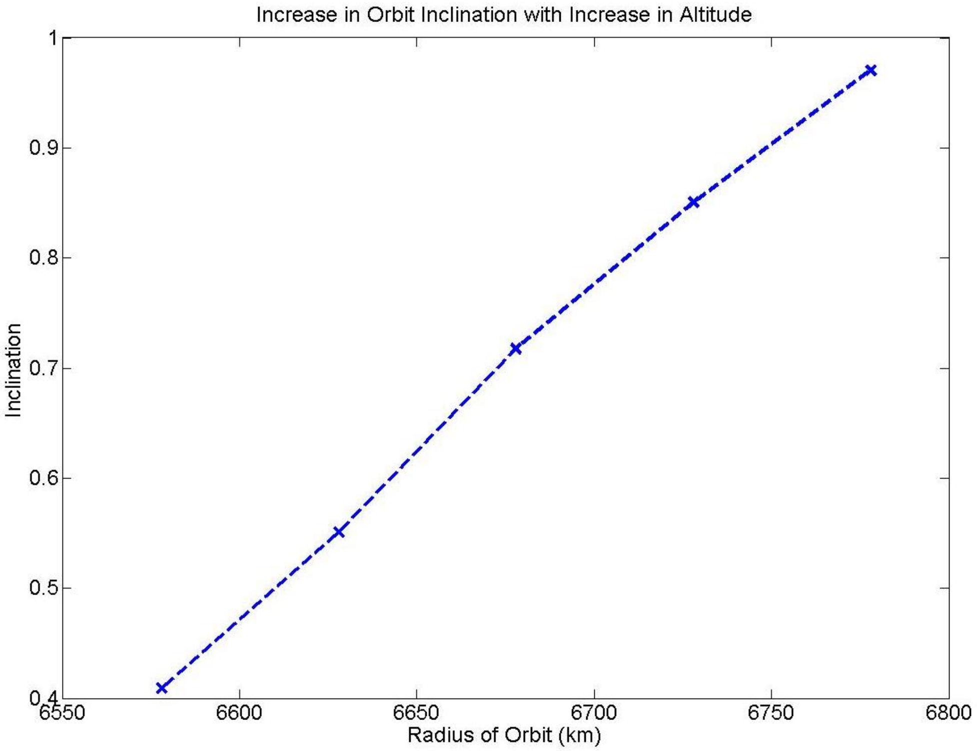


Figure 42: Orbital Inclination Change varying with initial Radius (α=0)

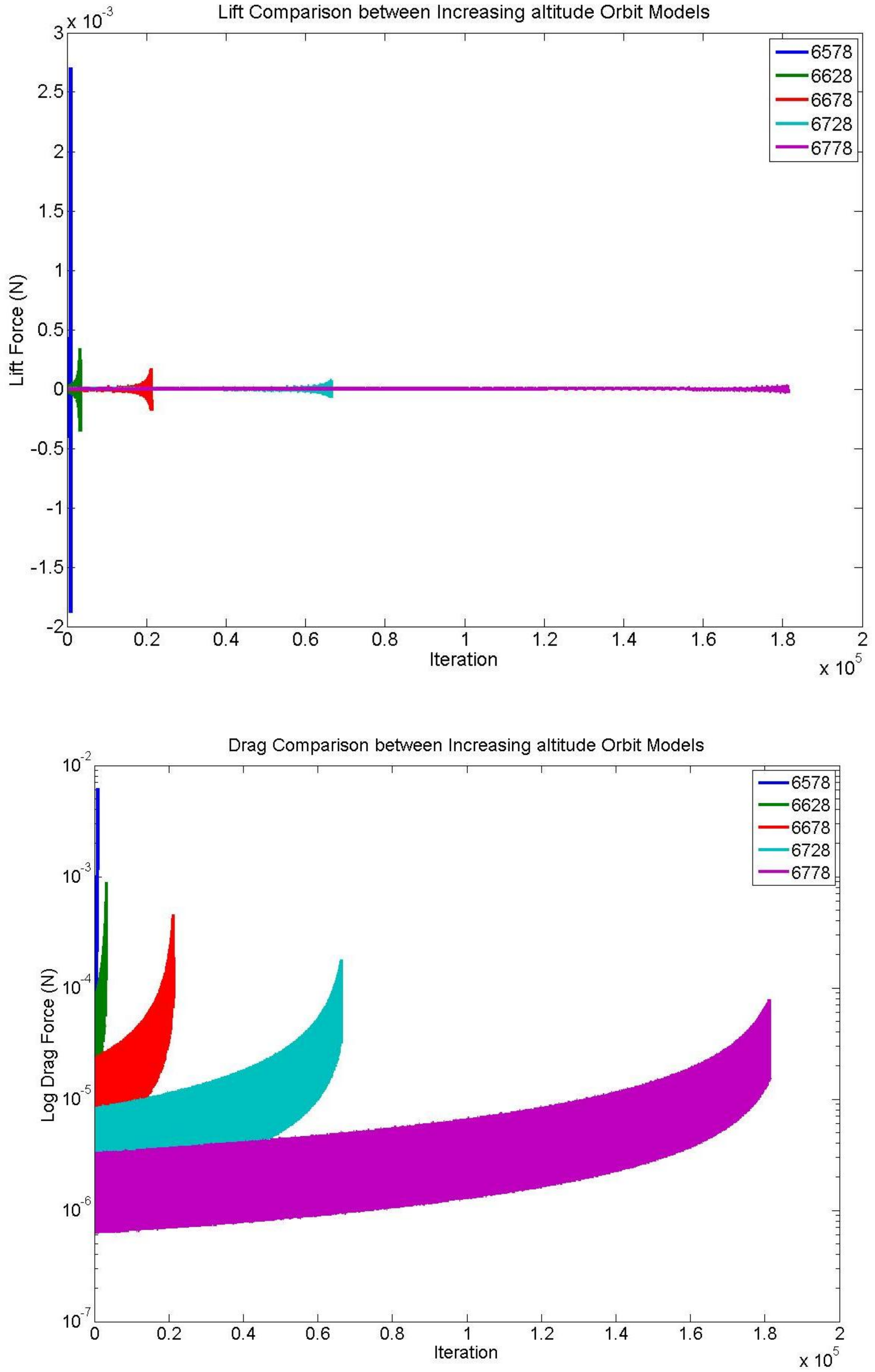


Figure 43: Comparison of Drag and Lift Force versus starting radial altitude (α=0)

Once, the difference in lift and inclination change (due to the accommodation coefficient) was observed a final test was done to measure the extent of the atmosphere property effects on the inclination change. The accommodation coefficient was zero, and the total time of measurement was 19 orbits.

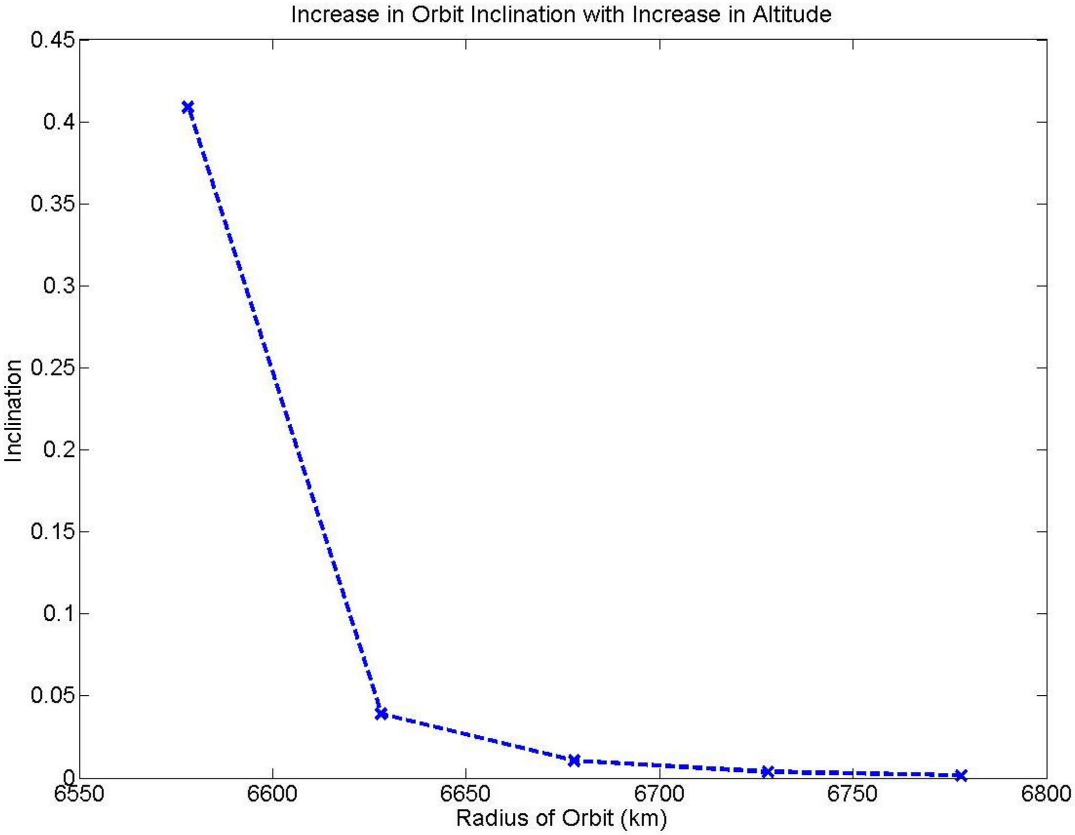


Figure 44: Final Inclination of Orbital Models after 20 Orbits (α=0)

d. Scaling Laws

Scaling laws approximate mass, surface area, and volume based on the measurement of ℓ. The surface area and mass of a cubic satellite were scaled up and the inclination change was compared. The surface area and mass are found using Eq. 34.

$$\forall = \frac{m}{\rho} \propto \ell^3 \; where \; \rho = 1330 \; kg/m^3 \qquad \text{Eq. 34}$$

$$SA \propto \; \ell^2$$

Table 7 shows the surface area and mass values and Figure 45 shows the increasing ballistic coefficient when scaling from 0.1 to 1 ℓ.

Table 7: Increasing Surface Value and Mass based on Scaling laws

ℓ	meters	Surface Area (m^2)	Surface Area (km^2)	Mass (kg)
1	0.1	1.00E-02	1.00E-08	1.33
2	0.2	4.00E-02	4.00E-08	10.64
4	0.4	1.60E-01	1.60E-07	85.12
6	0.6	3.60E-01	3.60E-07	287.28
8	0.8	6.40E-01	6.40E-07	680.96
10	1	1.00E+00	1.00E-06	1330

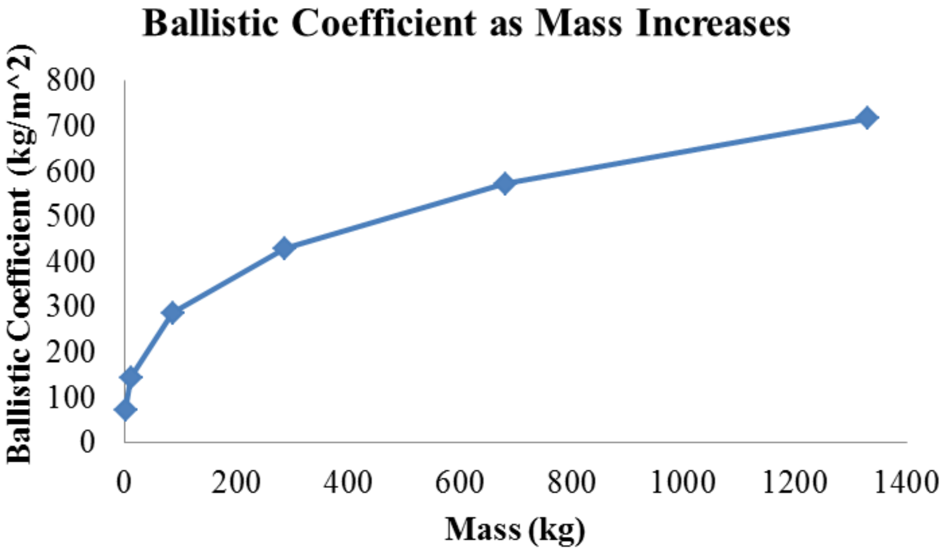


Figure 45: Increasing Ballistic Coefficient due to Scaling Laws

It can be seen in Figure 46 and Figure 47 that the inclination change experienced by a satellite with greater mass and surface area is the same as one with smaller mass and surface area is the same. However, due to the high ballistic coefficient the inclination change for the larger satellite take longer to complete. The change in inclination angle over time was found per iteration by finding the angle between the actual vector and the vector that would exist in the equatorial plane.

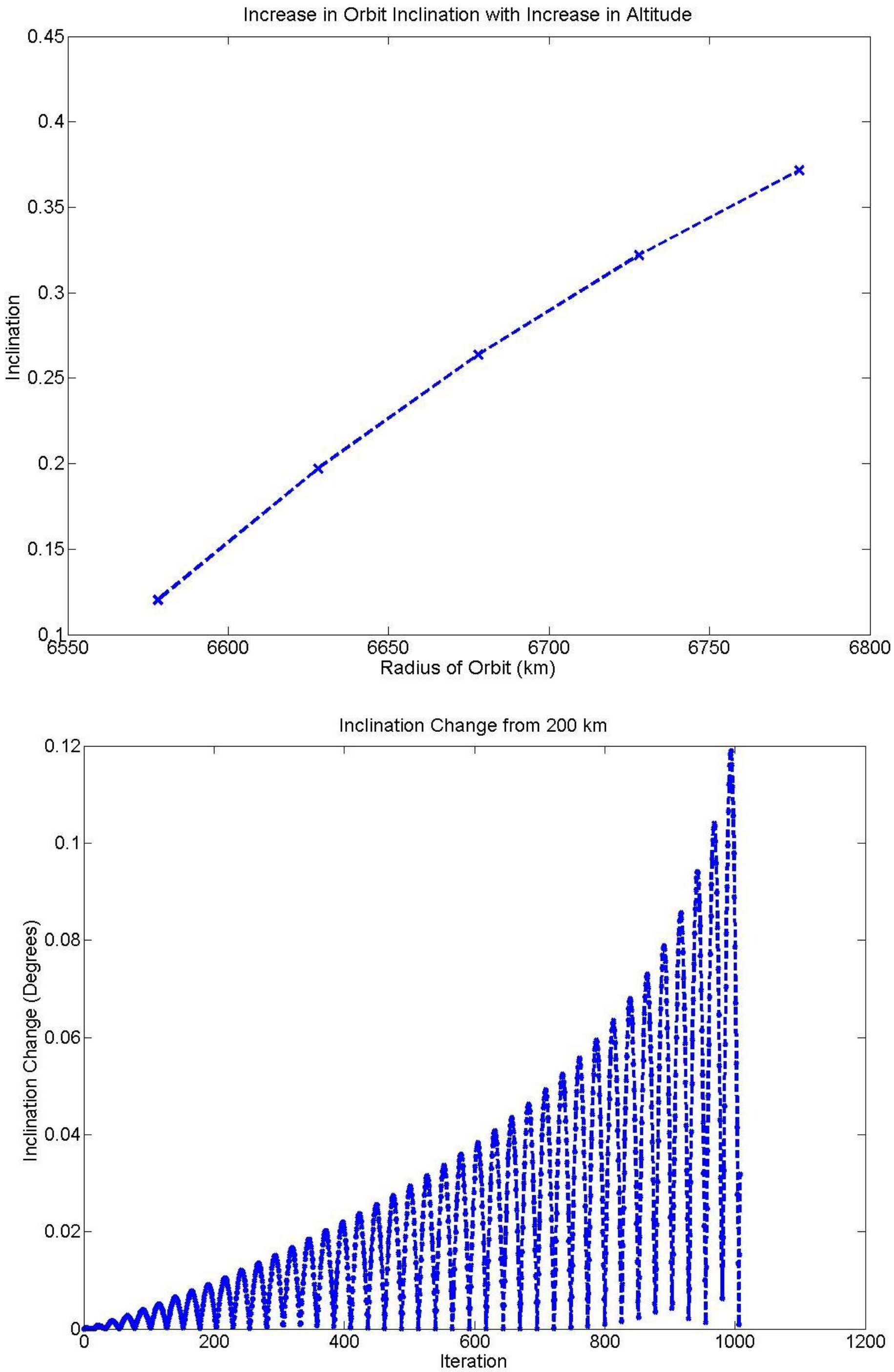
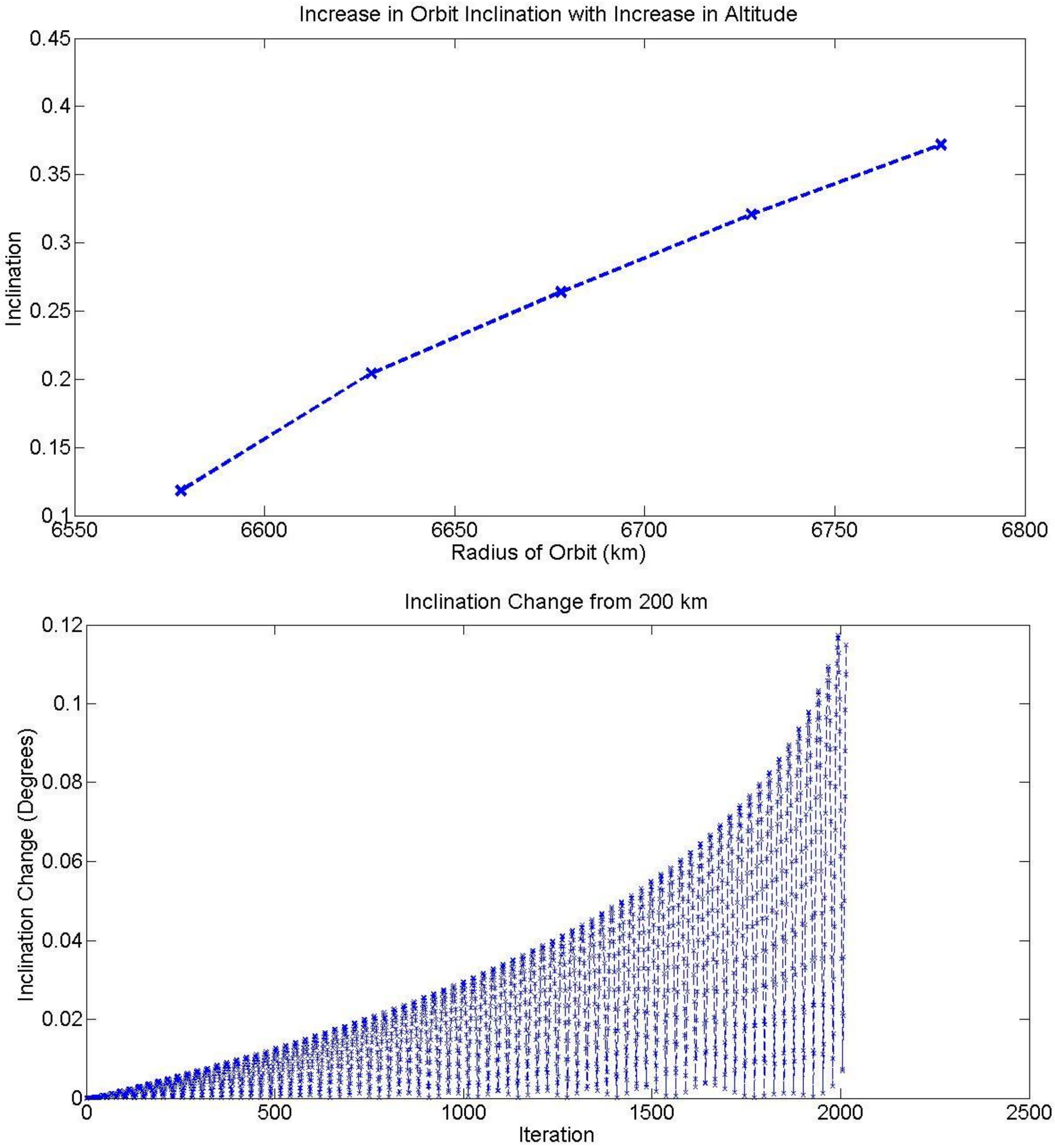


Figure 46: Inclination Change by altitude and per iteration for 1.33 kg satellite


Increase in Orbit Inclination with Increase in Altitude

Inclination Change from 200 km

Figure 47: Inclination Change by altitude and per iteration for 10.64 kg satellite

V. FUTURE WORK

Several changes and implementations still need to be applied to make this model even more accurate. First, the exchange of the NRLMSIS-00 for the USSA-76 needs to be made. Due to the modularity of the overall program, this modification should be relatively straight forward. A density function based on USSA-76 has been built to utilize the MATLAB source code and solve for the atmospheric density up to 1000 km. This function is one line inside the orbital model and can be replaced with a similar code created with the NRLMSIS-00 open source FORTRAN code.

Second, further consideration needs to be given towards the implementation of other perturbing forces, such as Earth oblateness, solar pressure, and thrusting vectors. Along with oblateness, drag is one of the strongest influences on a satellite in LEO [7]. Any additions were designed to be easily administered by using the differential equations that describe their behavior and breaking it apart into Cartesian coordinates. Ignoring the effects of all the perturbation forces can restrict the accuracy of the potion and velocity over long time periods [7]. In addition, adding the governing differential equations for thrusters allows for modeled orbital maneuvers paired with aeroassistance.

Besides a few implementations to the model to increase accuracy, the creation of this model will allow for more study in the field of cubesat aeroassistance. For example, there is a lack of literature on detailed aerodynamic characterization of various 3D shapes in LEO flight; with most basic research done decades ago. Another aspect is to complete a more thorough analysis of the accommodation coefficient effects on different surfaces relevant to cubesats and the effects at altitude.

VI. CONCLUSION

Overall, the testing that was done with the developed orbital trajectory model observed that if complete accommodation was assumed, no inclination change occurred by the lift forces with the model. However, as the accommodation coefficient was reduced inclination changes due to lift began to occur. The greatest inclination change observed by the constant angle of attack (constant positive z-directional lift) was 0.6 degrees with an accommodation coefficient of zero and a 0.07 degree change with an accommodation coefficient of 0.9. The greatest inclination observed by scheduled angle of attack (alternating the lift direction to positive and negative z direction) was close to one degree with an accommodation coefficient of zero and a 0.4 degree change with an accommodation coefficient of 0.9. Therefore, it is possible to make inclination changes with aerodynamic maneuvers but it is unlikely due to the requirement of having an accommodation coefficient of zero. It should be articulated that an accommodation coefficient of zero is not realistic, but these tests were performed to find where the maximum lift occurred. The 0.07 degree and 0.4 degree inclination changes were made with an accommodation coefficient of 0.90 and at a starting altitude of 400 km and an initial orbiting velocity of 7.6686 $\frac{km}{s}$. The equivalent Δυ for these inclination changes (if drag compensation was assumed) are 0.0094 $\frac{km}{s}$ and 0.0535 $\frac{km}{s}$.

When scaling laws where applied, inclination change comparisons where made between cube satellites of different surfaces areas and masses. The surface area and mass comparisons were made between satellites weighing 1.33 kg and 10.64 kg. It was assumed that a trend would follow for further increasing size. It was observed that the increase in size for the nominal cubic shape does not affect the final inclination change that is experienced. This is due to the balanced trade-off between surface area and mass, where larger satellites experienced less aerodynamic forces (less surface area) but gained more orbital life (increased mass) when compared to smaller satellites.

BIBLIOGRAPHY

[1] H. Helvajian and E. Robinson, Micro- and Nanotechnology for Space Systems, Los Angeles, CA: American Institute of Aeronautics and Astronautics, 1997.

[2] T. C. Program, "CubeSat Design Specifiction Rev. 12," California Polytechnic University, Saint Luis Obisbo.

[3] H. Helvajian and S. Janson, Small Satellites: Past, Present, and Future, Aerospace Press, 2008.

[4] P. Thakker, Emergence of pico- and nanosatellites for atmospheric research and technology testing, Reston, Virgina: American Institute of Aeronautics and Astronautics, 2010.

[5] J. Cen and J. Xu, "A new prototype of self-pressurizing fuel tank for micro and nano-satellites," *Acta Astronautica,* pp. 410-415, 2009.

[6] R. A. Zeledon and M. A. Peck, "Electrolysis Propulsion for CubeSat-Scale Spacecraft," in *AIAA SPACE 2011 Conference & Exposition*, Long Beach, CA, 2011.

[7] D. A. Vallado and D. M. Wayne, Fundamentals of Astrodynamics and Applications, New York: Springer, 2007.

[8] M. D. Griffin and J. R. French, Space Vehicle Design, Reston, VA: American Institute of Aeronautics and Astronautics, Inc., 2004.

[9] O. Montenbruck and E. Gill, Satellite Orbits: Models, Methods, and Applications, Berlin, Germany: Springer, 2005.

[10] A. Baker, A. Curiel, J. Shcaffner and M. Sweeting, "You can get there from here: Advanced low cost propulsion concepts for small satellites beyond LEO," *Acta Astronautica,* pp. 288-301, 2005.

[11] F. A. El-Salam, "Optimization out-of-orbit plane changes using aeroassisted maneuvers," *Applied Mathematics and Computation*, pp. 1303-1313, 2005.

[12] R. R. Bate, D. D. Mueller and J. E. White, Fundamentals of Astrodynamics, New York: Dover Publications, 1971.

[13] M. Horsley, "An investigation into using differential drag for controlling a formation of CubeSats," in *Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference*, Maui, Hawaii, 2011.

[14] R. L. Burder and J. D. Faires, Numerical Analysis, Pacific Grove, CA: Brooks/Cole, 2001.

[15] T.-S. Bae, D. Grejner-Brzezinska and J. H. Kwon, "Efficient LEO Dynamic Orbit Determination with Triple Differenced GPS Carrier Phases," *Journal of Navigation*, pp. 217-232, 2007.

[16] C. D. Brown, Elements of Spacecraft Design, Reston, VA: American Institute of Aeronautics and Astronautics, Inc., 2002.

[17] T.-S. Bae, "LEO Dynamic Orbit Enhancement Using Atmospheric and Empirical Force Modeling," in *Proceedings of the 18th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2005)*, Long Beach, CA, 2005.

[18] G. Cook, "Satellite Drag Coefficients," *Planetary Space Science*, pp. 929-946, 1965.

[19] E. Hecht, Physics: Algebra/Trig, Pacific Grove, CA: Brooks/Cole-Thomson Learning, 2003.

[20] D. King-Hele, Satellite orbits in an atmosphere: theory and applications, Bishopbriggs, Glasgow: Blackie and Son Ltd, 1987.

[21] P. W. Davis, Differential equations: modeling with MATLAB, Prentice Hall, 1999.

[22] Mathworks Inc., "MATLAB, Mathworks Inc. Software Package," Natick, 2011.

[23] B. Lewis, "Complete 1976 Standard Atmosphere," 11 January 2007. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/13635-complete-1976-standard-atmosphere. [Accessed 2011].

[24] U.S. Standard Atmosphere, 1976, Washington D.C.: U.S. Government Printing Office, 1976.

[25] T. Feagin, "A Tenth-Order Runge-Kutta Method with Error Estimate," 24 December 2006. [Online]. Available: http://sce.uhcl.edu/feagin/courses/rk10.pdf. [Accessed 30 January 2012].

[26] B. N. Datta, Numerical Linear Algebra and applications, Philadelphia, PA: Society for Industrial and Applied Mathematics, 2010.

[27] E. Mazur, Physics Principles and Practice, Upper Saddle River, New Jersey: Prentice Hall, 2010.

[28] M. R. Lindeburg, Mechanical Engineering Reference Manual for the PE Exam, Blemont, CA: Professional Publications, 2001.

[29] M. W. Smith, S. Seager, C. M. Pong and J. S. Villasenor, "ExoplanetSat: Detecting transiting exoplanets using a low-cost CubeSat platform," in *ExoplanetSat: detecting transiting exoplanets using a low-cost CubeSat platform.*, San Diego, California, 2010.

[30] H. D. Curtis, Orbital Mechanics for Engineering Students, Burlington, MA: Elsevier Ltd, 2005.

APPENDIX A – MATLAB Codes

a. Orbital Propagation Model

```
function [tgrid,ansxy, F]=Orbiting_Model_with_Drag_And_Lift(span, initial, N, Orbits, Angles,
Constants, AC)
%This function models a satellite in orbit with drag and lift. This program
%uses a 10th order RK method as the numerical differential equations solver
%
%Inputs:
%       Span = [0 Period]
%       Initial = [x0 y0 z0 vx0 vy0 vz0]
%       N = Period/Step size desired
%       Orbits = Number of orbits to go while testing
%       Angles = [theta gamma delta alpha beta eta] in degrees
%           theta = (rotation around x-axis from Earth Centered to Body Axis)
%           gamma = (rotation around y-axis from Earth Centered to Body Axis)
%           eta = (rotation around z-axis from Earth Centered to Body Axis
%           alpha = (rotation around x-axis from Body Axis to Wind Axis)
%           beta = (rotation around y-axis from Body Axis to Wind Axis)
%           delta = (rotation around z-axis from Body Axis to Wind Axis
%       Constants = [G (km^3/kg-s^2) M (kg) SA m omega_e]
%           G = Universal Gravitational Constant
%           M = Mass of the Earth
%           SA = Satellite Surface area
%           m = Satellite mass
%           omega_e = rotational velocity of the Earth
%         AC = Accommodation Coefficient
%Outputs:
%       tgrid = [0:h:Peroid*Orbits]
%       ansxy = Earth centered axis, [x y z vx vy vz]
%       F = Gravity, Drag, and Lift Forces Experienced [G D L]
format long

span=span*Orbits;
N=N*Orbits;

tstart_10th=tic;
[tgrid, ansxy, F , ~, ~]=tenthorderRK022912('fn040312', span, initial, N, Angles, Constants,
AC);
telapsed_10th=toc(tstart_10th);

plot_force(tgrid, F);
Three_D_Orbit(tgrid, ansxy);

end
```

b. 10th Order Runge-Kutta Numerical Integrator – Constant Angle of Attack

```
function [tgrid, ansXY, F, xyz_prime, xyz_doubleprime]=tenthorderRK022912(fn, span, initial,
N, angles, constants, AC)
%This function is the Numerical Integration solver 10th order Runge-Kutta
%
%The inputs of this program are identical to the initial inputs expect span
%and the number of intervals have been adjusts by the number of orbits

format long

ainitial=span(1); binitial=span(2);     % Beginning and Ending Points
h=(binitial-ainitial)/N;                % Step Size
tgrid = [ainitial:h:binitial]';         % Column list of time values

%Creating the needed matrices and assigning length to save memory
ansXY=zeros(length(tgrid), length(initial));
accel_d=zeros(length(tgrid)-1,1); accel_g=zeros(length(tgrid)-1,1); accel_l=zeros(length(tgrid)-
1,1);
xyz_prime=zeros(length(tgrid), 3); xyz_doubleprime=zeros(length(tgrid), 3);
Vxyz_prime=zeros(length(tgrid), 3); Vxyz_doubleprime=zeros(length(tgrid), 3);


t_initial = ainitial;                 % Starting point
yn=initial;                          % Row list of initial values
ansXY(1,:) = initial;                   % First row has initial values
rho_set=[]; R0=[];
Body=[angles(1) angles(2) angles(3)]; Wind = [angles(4) angles(5) angles(6)];

%Radius Vector Coordinate Transformation
xyz_prime(1,:)=Earth_Axis_To_Body_Axis(Body, [yn(1) yn(2) yn(3)]);
xyz_doubleprime(1,:)=Body_Axis_To_Wind_Axis(Wind, xyz_prime(1,:));

%Radius Vector Coordinate Transformation
Vxyz_prime(1,:)=Earth_Axis_To_Body_Axis(Body, [yn(4) yn(5) yn(6)]);
Vxyz_doubleprime(1,:)=Body_Axis_To_Wind_Axis(Wind, Vxyz_prime(1,:));

%Define Constants for RK Method
[c a b]=RKconstants;

%Calculate Drag and Lift Coefficients
[C_d C_l]=drag_lift_coeffecients(Body, Wind, AC);
constants(6)=C_d; constants(7)=C_l;

%Beginning of RK10 solver loop
for i = 2:N+1
```

```
   [R0 rho rho_set]=density122211(yn, i, R0, rho_set); % Producing atmospheric model

   %The 17 equations required to solve for the next state
   k0=h*feval(fn, t_initial+a(1)*h, yn, rho, Body, Wind, constants);

   yn1 = yn + b(1,1)*k0;
   tn1 = t_initial+a(2)*h;
   k1 = h*feval(fn,tn1,yn1, rho, Body, Wind, constants);

   yn2 = yn + b(2,1)*k0+b(2,2)*k1;
   tn2 = t_initial+a(3)*h;
   k2 = h*feval(fn, tn2, yn2, rho, Body, Wind, constants);

   yn3 = yn + b(3,1)*k0+b(3,2)*k1+b(3,3)*k2;
   tn3 = t_initial+a(4)*h;
   k3 = h*feval(fn, tn3, yn3, rho, Body, Wind, constants);

   yn4 = yn + b(4,1)*k0+b(4,2)*k1+b(4,3)*k2+b(4,4)*k3;
   tn4 = t_initial+a(5)*h;
   k4 =h*feval(fn, tn4, yn4, rho, Body, Wind, constants);

   yn5 = yn + b(5,1)*k0+b(5,2)*k1+b(5,3)*k2+b(5,4)*k3+b(5,5)*k4;
   tn5 = t_initial+a(6)*h;
   k5 = h*feval(fn, tn5, yn5, rho, Body, Wind, constants);

   yn6 = yn + b(6,1)*k0+b(6,2)*k1+b(6,3)*k2+b(6,4)*k3+b(6,5)*k4+b(6,6)*k5;
   tn6 = t_initial+a(7)*h;
   k6 = h*feval(fn, tn6, yn6, rho, Body, Wind, constants);

   yn7 = yn + b(7,1)*k0+b(7,2)*k1+b(7,3)*k2+b(7,4)*k3+b(7,5)*k4+b(7,6)*k5+b(7,7)*k6;
   tn7 = t_initial+a(8)*h;
   k7 = h*feval(fn, tn7, yn7, rho, Body, Wind, constants);

   yn8 = yn +
b(8,1)*k0+b(8,2)*k1+b(8,3)*k2+b(8,4)*k3+b(8,5)*k4+b(8,6)*k5+b(8,7)*k6+b(8,8)*k7;
   tn8 = t_initial+a(9)*h;
   k8 = h*feval(fn, tn8, yn8, rho, Body, Wind, constants);

   yn9 = yn +
b(9,1)*k0+b(9,2)*k1+b(9,3)*k2+b(9,4)*k3+b(9,5)*k4+b(9,6)*k5+b(9,7)*k6+b(9,8)*k7+b(9,9)*
k8;
   tn9 = t_initial+a(10)*h;
   k9 = h*feval(fn, tn9, yn9, rho, Body, Wind, constants);
   yn10 = yn +
b(10,1)*k0+b(10,2)*k1+b(10,3)*k2+b(10,4)*k3+b(10,5)*k4+b(10,6)*k5+b(10,7)*k6+b(10,8)*k
7+b(10,9)*k8+b(10,10)*k9;
   tn10 = t_initial+a(11)*h;
   k10 = h*feval(fn, tn10, yn10, rho, Body, Wind, constants);

   yn11 = yn +
b(11,1)*k0+b(11,2)*k1+b(11,3)*k2+b(11,4)*k3+b(11,5)*k4+b(11,6)*k5+b(11,7)*k6+b(11,8)*k
7+b(11,9)*k8+b(11,10)*k9+b(11,11)*k10;
   tn11 = t_initial+a(12)*h;
   k11 = h*feval(fn, tn11, yn11, rho, Body, Wind, constants);

   yn12 = yn +
b(12,1)*k0+b(12,2)*k1+b(12,3)*k2+b(12,4)*k3+b(12,5)*k4+b(12,6)*k5+b(12,7)*k6+b(12,8)*k
7+b(12,9)*k8+b(12,10)*k9+b(12,11)*k10+b(12,12)*k11;
   tn12 = t_initial+a(13)*h;
   k12 = h*feval(fn, tn12, yn12, rho, Body, Wind, constants);

   yn13 = yn +
b(13,1)*k0+b(13,2)*k1+b(13,3)*k2+b(13,4)*k3+b(13,5)*k4+b(13,6)*k5+b(13,7)*k6+b(13,8)*k
7+b(13,9)*k8+b(13,10)*k9+b(13,11)*k10+b(13,12)*k11+b(13,13)*k12;
   tn13 = t_initial+a(14)*h;
   k13 = h*feval(fn, tn13, yn13, rho, Body, Wind, constants);

   yn14 = yn +
b(14,1)*k0+b(14,2)*k1+b(14,3)*k2+b(14,4)*k3+b(14,5)*k4+b(14,6)*k5+b(14,7)*k6+b(14,8)*k
7+b(14,9)*k8+b(14,10)*k9+b(14,11)*k10+b(14,12)*k11+b(14,13)*k12+b(14,14)*k13;
   tn14 = t_initial+a(15)*h;
   k14 = h*feval(fn, tn14, yn14, rho, Body, Wind, constants);

   yn15 = yn +
b(15,1)*k0+b(15,2)*k1+b(15,3)*k2+b(15,4)*k3+b(15,5)*k4+b(15,6)*k5+b(15,7)*k6+b(15,8)*k
7+b(15,9)*k8+b(15,10)*k9+b(15,11)*k10+b(15,12)*k11+b(15,13)*k12+b(15,14)*k13+b(15,15)
*k14;
   tn15 = t_initial+a(16)*h;
   k15 = h*feval(fn, tn15, yn15, rho, Body, Wind, constants);

   yn16 = yn +
b(16,1)*k0+b(16,2)*k1+b(16,3)*k2+b(16,4)*k3+b(16,5)*k4+b(16,6)*k5+b(16,7)*k6+b(16,8)*k
7+b(16,9)*k8+b(16,10)*k9+b(16,11)*k10+b(16,12)*k11+b(16,13)*k12+b(16,14)*k13+b(16,15)
*k14+b(16,16)*k15;
   tn16 = t_initial+a(17)*h;
   k16 = h*feval(fn, tn16, yn16, rho, Body, Wind, constants);
```

```
Kh=c(1)*k0+c(2)*k1+c(3)*k2+c(4)*k3+c(5)*k4+c(6)*k5+c(7)*k6+c(8)*k7+c(9)*k8+c(10)*k9
+c(11)*k10+c(12)*k11+c(13)*k12+c(14)*k13+c(15)*k14+ c(16)*k15+c(17)*k16;

   % Next State
   ynext = yn + Kh;

   %Calculates the Gravity, Drag, and Lift Forces experienced during this
   %iteration
   [d g l]=gettotalaccel(yn, rho, Body, Wind, constants);
   accel_d(i-1)=d;
   accel_g(i-1)=g;
   accel_l(i-1)=l;

   %Stores and resets for next iteration
   ansXY(i,:)= ynext;
   yn=ynext;
   t_initial = t_initial+h;

   %Keeping track of Coordinates systems
   prime=Earth_Axis_To_Body_Axis(Body, [yn(1) yn(2) yn(3)]); prime=prime';
xyz_prime(i,:)=prime;
   double=Body_Axis_To_Wind_Axis(Wind, prime); double=double';
xyz_doubleprime(i,:)=double;

   prime=Earth_Axis_To_Body_Axis(Body,[yn(4) yn(5) yn(6)]); prime=prime';
Vxyz_prime(i,:)=prime;
   double=Body_Axis_To_Wind_Axis(Wind, prime); double=double';
Vxyz_doubleprime(i,:)=double;

end

  %Putting the forces into one matrix
  F=[accel_g, accel_d, accel_l];

 end
```

c. Coefficients of Drag and Lift – Constant Angle of Attack

```
function [Cd Cl]=drag_lift_coeffecients(Body, Wind, AC)
%This function define the Coefficients of Drag and Lift from the Angle of Attack
%
% Input: Body = angle rotations from Earth Axis to Body Axis
%        Wind = angle rotations from Body Axis to Wind Axis
%        AC = Accommodation Coefficient
```

```
bx=Body(1); by=Body(2); bz=Body(3);
wx=Wind(1); wx=Wind(2); wz=Wind(3);

%Coefficients
Cd=2+(4/3)*sqrt(1+AC)*cosd(4*bx);
Cl=(4/3)*sqrt(1+AC)*sind(2*bx);

end
```

d. 10th Order Runge-Kutta Numerical Integrator – Scheduled Angle of Attack

```
function [tgrid, ansXY, F, xyz_prime, xyz_doubleprime]=tenthorderRK022912(fn, span, initial,
N, angles, constants, AC)
format long

ainitial=span(1); binitial=span(2);
h=(binitial-ainitial)/N;
tgrid = [ainitial:h:binitial]';        % Column list of time values

ansXY=zeros(length(tgrid), length(initial));
accel_d=zeros(length(tgrid)-1,1); accel_g=zeros(length(tgrid)-1,1); accel_l=zeros(length(tgrid)-
1,1);
xyz_prime=zeros(length(tgrid), 3); xyz_doubleprime=zeros(length(tgrid), 3);
Vxyz_prime=zeros(length(tgrid), 3); Vxyz_doubleprime=zeros(length(tgrid), 3);


t_initial = ainitial;                  % Starting point
yn=initial;                          % Row list of initial values
ansXY(1,:) = initial;                   % First row has initial values
rho_set=[]; R0=[];
Body=[angles(1) angles(2) angles(3)]; Wind = [angles(4) angles(5) angles(6)];

%Radius Vector Coordinate Transformation
xyz_prime(1,:)=Earth_Axis_To_Body_Axis(Body, [yn(1) yn(2) yn(3)]);
xyz_doubleprime(1,:)=Body_Axis_To_Wind_Axis(Wind, xyz_prime(1,:));

%Radius Vector Coordinate Transformation
Vxyz_prime(1,:)=Earth_Axis_To_Body_Axis(Body, [yn(4) yn(5) yn(6)]);
Vxyz_doubleprime(1,:)=Body_Axis_To_Wind_Axis(Wind, Vxyz_prime(1,:));

[c a b]=RKconstants;
OneAndTwo=zeros(2, length(initial));
% Recall: Numerical theory index starts at zero, MATLAB index starts at 1
 for i = 2:N+1

   [C_d C_l]=drag_lift_coeffecients(i, Body, Wind, AC, OneAndTwo);
   constants(6)=C_d; constants(7)=C_l; DragCoeffecient(i-1)=C_d;

   [R0 rho rho_set]=density122211(yn, i, R0, rho_set);

   k0=h*feval(fn, t_initial+a(1)*h, yn, rho, Body, Wind, constants);

   yn1 = yn + b(1,1)*k0;
   tn1 = t_initial+a(2)*h;
   k1 = h*feval(fn,tn1,yn1, rho, Body, Wind, constants);

   yn2 = yn + b(2,1)*k0+b(2,2)*k1;
   tn2 = t_initial+a(3)*h;
   k2 = h*feval(fn, tn2, yn2, rho, Body, Wind, constants);

   yn3 = yn + b(3,1)*k0+b(3,2)*k1+b(3,3)*k2;
   tn3 = t_initial+a(4)*h;
   k3 = h*feval(fn, tn3, yn3, rho, Body, Wind, constants);

   yn4 = yn + b(4,1)*k0+b(4,2)*k1+b(4,3)*k2+b(4,4)*k3;
   tn4 = t_initial+a(5)*h;
   k4 =h*feval(fn, tn4, yn4, rho, Body, Wind, constants);

   yn5 = yn + b(5,1)*k0+b(5,2)*k1+b(5,3)*k2+b(5,4)*k3+b(5,5)*k4;
   tn5 = t_initial+a(6)*h;
   k5 = h*feval(fn, tn5, yn5, rho, Body, Wind, constants);

   yn6 = yn + b(6,1)*k0+b(6,2)*k1+b(6,3)*k2+b(6,4)*k3+b(6,5)*k4+b(6,6)*k5;
   tn6 = t_initial+a(7)*h;
   k6 = h*feval(fn, tn6, yn6, rho, Body, Wind, constants);

   yn7 = yn + b(7,1)*k0+b(7,2)*k1+b(7,3)*k2+b(7,4)*k3+b(7,5)*k4+b(7,6)*k5+b(7,7)*k6;
   tn7 = t_initial+a(8)*h;
   k7 = h*feval(fn, tn7, yn7, rho, Body, Wind, constants);

   yn8 = yn +
b(8,1)*k0+b(8,2)*k1+b(8,3)*k2+b(8,4)*k3+b(8,5)*k4+b(8,6)*k5+b(8,7)*k6+b(8,8)*k7;
   tn8 = t_initial+a(9)*h;
   k8 = h*feval(fn, tn8, yn8, rho, Body, Wind, constants);

   yn9 = yn +
b(9,1)*k0+b(9,2)*k1+b(9,3)*k2+b(9,4)*k3+b(9,5)*k4+b(9,6)*k5+b(9,7)*k6+b(9,8)*k7+b(9,9)*
k8;
   tn9 = t_initial+a(10)*h;
   k9 = h*feval(fn, tn9, yn9, rho, Body, Wind, constants);
```

```
    yn10 = yn + 
b(10,1)*k0+b(10,2)*k1+b(10,3)*k2+b(10,4)*k3+b(10,5)*k4+b(10,6)*k5+b(10,7)*k6+b(10,8)*k
7+b(10,9)*k8+b(10,10)*k9;
    tn10 = t_initial+a(11)*h;
    k10 = h*feval(fn, tn10, yn10, rho, Body, Wind, constants);

    yn11 = yn + 
b(11,1)*k0+b(11,2)*k1+b(11,3)*k2+b(11,4)*k3+b(11,5)*k4+b(11,6)*k5+b(11,7)*k6+b(11,8)*k
7+b(11,9)*k8+b(11,10)*k9+b(11,11)*k10;
    tn11 = t_initial+a(12)*h;
    k11 = h*feval(fn, tn11, yn11, rho, Body, Wind, constants);

    yn12 = yn + 
b(12,1)*k0+b(12,2)*k1+b(12,3)*k2+b(12,4)*k3+b(12,5)*k4+b(12,6)*k5+b(12,7)*k6+b(12,8)*k
7+b(12,9)*k8+b(12,10)*k9+b(12,11)*k10+b(12,12)*k11;
    tn12 = t_initial+a(13)*h;
    k12 = h*feval(fn, tn12, yn12, rho, Body, Wind, constants);

    yn13 = yn + 
b(13,1)*k0+b(13,2)*k1+b(13,3)*k2+b(13,4)*k3+b(13,5)*k4+b(13,6)*k5+b(13,7)*k6+b(13,8)*k
7+b(13,9)*k8+b(13,10)*k9+b(13,11)*k10+b(13,12)*k11+b(13,13)*k12;
    tn13 = t_initial+a(14)*h;
    k13 = h*feval(fn, tn13, yn13, rho, Body, Wind, constants);

    yn14 = yn + 
b(14,1)*k0+b(14,2)*k1+b(14,3)*k2+b(14,4)*k3+b(14,5)*k4+b(14,6)*k5+b(14,7)*k6+b(14,8)*k
7+b(14,9)*k8+b(14,10)*k9+b(14,11)*k10+b(14,12)*k11+b(14,13)*k12+b(14,14)*k13;
    tn14 = t_initial+a(15)*h;
    k14 = h*feval(fn, tn14, yn14, rho, Body, Wind, constants);

    yn15 = yn + 
b(15,1)*k0+b(15,2)*k1+b(15,3)*k2+b(15,4)*k3+b(15,5)*k4+b(15,6)*k5+b(15,7)*k6+b(15,8)*k
7+b(15,9)*k8+b(15,10)*k9+b(15,11)*k10+b(15,12)*k11+b(15,13)*k12+b(15,14)*k13+b(15,15)
*k14;
    tn15 = t_initial+a(16)*h;
    k15 = h*feval(fn, tn15, yn15, rho, Body, Wind, constants);

    yn16 = yn + 
b(16,1)*k0+b(16,2)*k1+b(16,3)*k2+b(16,4)*k3+b(16,5)*k4+b(16,6)*k5+b(16,7)*k6+b(16,8)*k
7+b(16,9)*k8+b(16,10)*k9+b(16,11)*k10+b(16,12)*k11+b(16,13)*k12+b(16,14)*k13+b(16,15)
*k14+b(16,16)*k15;
    tn16 = t_initial+a(17)*h;
    k16 = h*feval(fn, tn16, yn16, rho, Body, Wind, constants);
```

```
Kh=c(1)*k0+c(2)*k1+c(3)*k2+c(4)*k3+c(5)*k4+c(6)*k5+c(7)*k6+c(8)*k7+c(9)*k8+c(10)*k9
+c(11)*k10+c(12)*k11+c(13)*k12+c(14)*k13+c(15)*k14+ c(16)*k15+c(17)*k16;
    ynext = yn + Kh; %ynext is in wind axis coordinates

    [d g l]=gettotalaccel(yn, rho, Body, Wind, constants);
    accel_d(i-1)=d;
    accel_g(i-1)=g;
    accel_l(i-1)=l;

    OneAndTwo(1,:)=ynext; OneAndTwo(2,:)=yn;

    ansXY(i,:)= ynext;
    yn=ynext;
    t_initial = t_initial+h;

    %Keeping track of Coordinates systems
    prime=Earth_Axis_To_Body_Axis(Body, [yn(1) yn(2) yn(3)]); prime=prime';
xyz_prime(i,:)=prime;
    double=Body_Axis_To_Wind_Axis(Wind, prime); double=double';
xyz_doubleprime(i,:)=double;

    prime=Earth_Axis_To_Body_Axis(Body,[yn(4) yn(5) yn(6)]); prime=prime';
Vxyz_prime(i,:)=prime;
    double=Body_Axis_To_Wind_Axis(Wind, prime); double=double';
Vxyz_doubleprime(i,:)=double;

    i
  end
  DragCoeffecient=DragCoeffecient';
  F=[accel_g, accel_d, accel_l];

 end
```

e. Coefficients of Drag and Lift – Scheduled Angle of Attack

```
function [Cd Cl]=drag_lift_coeffecients(i,Body,Wind,AC,OneAndTwo)

     if i==2
         bx=Body(1); by=Body(2); bz=Body(3);
         wx=Wind(1); wy=Wind(2); wz=Wind(3);
     else
         bx=quadrant_test(OneAndTwo);
     end

Cd=2+(4/3)*sqrt(1-AC)*cosd(4*bx);
```

```
Cl=(4/3)*sqrt(1-AC)*sind(2*bx);

 end

function bx=quadrant_test(OneAndTwo)

 z1=OneAndTwo(2,3); z0=OneAndTwo(1,3);

 if z1-z0>=0
     if z1>0
         bx=45;
     else
         bx=0;
     end
 elseif z1-z0<0
     if z1>0
         bx=0;
     else
         bx=-45;
     end
 end

 end
 %%
function derive=fn040312(h, initial,rho, Body, Wind,  constants)
format long

G=constants(1); M=constants(2);

x=initial(1);y=initial(2); z=initial(3);
vx=initial(4); vy=initial(5); vz=initial(6);

%Vectors of R, V, and Relative Velocity
R=[x y z];
V=[vx vy vz];

%Vector Norms
r=norm(R);

[D L]=Drag_And_Lift(R,V,rho,Body, Wind, constants);

derive=[vx
        vy
        vz
        -(G*M/r^3)*x+D(1)+L(1)
        -(G*M/r^3)*y+D(2)+L(2)
        -(G*M/r^3)*z+D(3)+L(3)]';

end %End of function
```

f. Differential Equations

```
function derive=fn040312(h, initial,rho,Body, Wind,constants)
% This function is the function that contains the differential equations
% that describe the motion of the orbiting satellite
%
%Input:h = Time Step
%      initial = state variable [x y z vx vy vz]
%      rho = Density at current altitude
%      Body = angle rotations from Earth Axis to Body Axis
%      Wind = angle rotations from Body Axis to Wind Axis
%      Constants = The constants that are required to solve for Gravity and
%              Drag [G (km^3/kg-s^2) M (kg) SA m omega_e]
%              G = Universal Gravitational Constant
%              M = Mass of the Earth
%              SA = Satellite Surface area
%              m = Satellite mass
%              omega_e = rotational velocity of the Earth
%Outputs:derive = the velocity and acceleration output vector
%             [vx vy vz ax ay az]

format long

G=constants(1); M=constants(2);

%Position and velocity assigned to a variable
x=initial(1);y=initial(2); z=initial(3);
vx=initial(4); vy=initial(5); vz=initial(6);

%Vectors of R, V, and Relative Velocity
R=[x y z];
V=[vx vy vz];

%Vector Norms
r=norm(R);

%Calculate Drag and Lift Acceleration
[D L]=Drag_And_Lift(R,V,rho,Body,Wind,constants);

derive=[vx
```

```
    vy
    vz
    -(G*M/r^3)*x+D(1)+L(1)
    -(G*M/r^3)*y+D(2)+L(2)
    -(G*M/r^3)*z+D(3)+L(3)]';

end
```

g. Lift and Drag Force

```
 function [D L]=Drag_And_Lift(R,V, rho, Body, Wind, constants)
 %This function solves for the Lift and Drag acceleration experienced by
 %the orbiting object at a given state
 %
 %Input:R = Position Vector [x y z]
 %     V = Velocity Vector [vx vy vz]
 %     rho = The density of the atmosphere at the current altitude
 %     Body = angle rotations from Earth Axis to Body Axis
 %     Wind = angle rotations from Body Axis to Wind Axis
 %     Constants = The constants that are required to solve for Gravity and
 %             Drag [G (km^3/kg-s^2) M (kg) SA m omega_e]
 %             G = Universal Gravitational Constant
 %             M = Mass of the Earth
 %             SA = Satellite Surface area
 %             m = Satellite mass
 %             omega_e = rotational velocity of the Earth
 %Outputs:D = Acceleration due to drag
 %       L = Acceleration due to lift

 %Assigning costs to variable
SA=constants(3); m=constants(4); C_d=constants(6); C_l=constants(7);
omega_e=constants(5); omega=[0 0 omega_e];

%Relative Velocity Vectors
V_rel=V-cross(omega,R); v_rel=norm(V_rel);

%Transform Relative Velocity vector to calculate Drag and Lift in Wind Axis
prime=Earth_Axis_To_Body_Axis(Body,V_rel);
double=Body_Axis_To_Wind_Axis(Wind,prime');
V_rel=double'; v_rel=norm(V_rel);

%Lift Vector
N=[0 0 1];
Lift_vector=cross(cross(V_rel,N),V_rel);

%Acceleration due to Drag and Lift
```

```
D=-((C_d*SA*rho*v_rel)/(2*m)).*V_rel;
L=-((C_l*SA*rho*v_rel)/(2*m)).*Lift_vector;

%Transforming acceleration back to Earth-Centered Axis
D=Wind_Axis_To_Body_Axis(Wind, D);
D=Body_Axis_To_Earth_Axis(Body, D');

L=Wind_Axis_To_Body_Axis(Wind, L);
L=Body_Axis_To_Earth_Axis(Body, L');

 end
```

h. Coordinate Transformations

```
function xyz_prime=Earth_Axis_To_Body_Axis(angles, xyz)
%This function transforms a vector form the Earth-Centered Axis to the Body
%Axis
%
%Input: angles = angle rotations from Earth Axis to Body Axis
%       xyz = The vector currently expressed in the Earth-Centered Axis
%Output: xyz_prime =  The vector now expressed in the Body Axis
%
%

x=angles(1); y=angles(2); z=angles(3);

transformy=[cosd(y) 0 -sind(y); 0 1 0; sind(y) 0 cosd(y)];
transformx=[1 0 0; 0 cosd(x) sind(x); 0 -sind(x) cosd(x)];
transformz=[cosd(z) sind(z) 0; -sind(z) cosd(z) 0; 0 0 1];
xyz_prime=transformz*transformy*transformx*xyz';

end
%%
function xyz_doubleprime=Body_Axis_To_Wind_Axis(angles, xyz_prime)
%This function transforms a vector form the Body Axis to the Wind
%Axis
%
%Input: angles = angle rotations from Body Axis to Wind Axis
%       xyz = The vector currently expressed in the Body Axis
%Ouput: xyz_doubleprime =  The vector now expressed in the Wind Axis
%
%

x=angles(1); y=angles(2); z=angles(3);
```

```
transformy=[cosd(y) 0 -sind(y); 0 1 0; sind(y) 0 cosd(y)];
transformx=[1 0 0; 0 cosd(x) sind(x); 0 -sind(x) cosd(x)];
transformz=[cosd(z) sind(z) 0; -sind(z) cosd(z) 0; 0 0 1];
xyz_doubleprime=transformz*transformy*transformx*xyz_prime';

end
%%
function xyz=Body_Axis_To_Earth_Axis(angles, xyz_prime)
%This function transforms a vector form the Body Axis to the Earth-Centered
%Axis
%
%Input: angles = angle rotations from Earth-Centered Axis to Body Axis
%       xyz_prime = The vector currently expressed in the Body Axis
%Ouput: xyz =  The vector now expressed in the Earth-Centered Axis

x=angles(1); y=angles(2); z=angles(3);

transformy=inv([cosd(y) 0 -sind(y); 0 1 0; sind(y) 0 cosd(y)]);
transformx=inv([1 0 0; 0 cosd(x) sind(x); 0 -sind(x) cosd(x)]);
transformz=inv([cosd(z) sind(z) 0; -sind(z) cosd(z) 0; 0 0 1]);
xyz=transformz*transformy*transformx*xyz_prime';

end
%%
function xyz_prime=Wind_Axis_To_Body_Axis(angles, xyz_doubleprime)
%This function transforms a vector form the Body Axis to the Earth-Centered
%Axis
%
%Input: angles = angle rotations from Body Axis to Wind Axis
%       xyz_doubleprime = The vector currently expressed in the Wind Axis
%Ouput: xyz_prime =  The vector now expressed in the Body Axis

x=angles(1); y=angles(2); z=angles(3);

transformy=inv([cosd(y) 0 -sind(y); 0 1 0; sind(y) 0 cosd(y)]);
transformx=inv([1 0 0; 0 cosd(x) sind(x); 0 -sind(x) cosd(x)]);
transformz=inv([cosd(z) sind(z) 0; -sind(z) cosd(z) 0; 0 0 1]);
xyz_prime=transformz*transformy*transformx*xyz_doubleprime';

end
```

i. Runge-Kutta Constants

```
function [c alpha beta]=RKconstants
%This function contains all of the constants used in the 10th order
```

```
%Runge-Kutta method
%
%c = the constants used in weighted average to calculate the next state
%alpha = the constants used to calculate the next time step before doing
%       the weighted average
%Beta = the constants used in calculating the next y variable to use in the
%next calculation that will lead in to the weighted average

c=[0.0333333333333333333333333333333333333333333333333333333333333
   0.0250000000000000000000000000000000000000000000000000000000000
   0.0333333333333333333333333333333333333333333333333333333333333
   0.000000000000000000000000000000000000000000000000000000000000
   0.0500000000000000000000000000000000000000000000000000000000000
   0.000000000000000000000000000000000000000000000000000000000000
   0.0400000000000000000000000000000000000000000000000000000000000
   0.000000000000000000000000000000000000000000000000000000000000
   0.189237478148923490158306404106012326238162346948625830327194
   0.277429188517743176508360262560654340428504319718040836339472
   0.277429188517743176508360262560654340428504319718040836339472
   0.189237478148923490158306404106012326238162346948625830327194
   -0.0400000000000000000000000000000000000000000000000000000000000
   -0.0500000000000000000000000000000000000000000000000000000000000
   -0.0333333333333333333333333333333333333333333333333333333333333
   -0.0250000000000000000000000000000000000000000000000000000000000
   0.0333333333333333333333333333333333333333333333333333333333333]';

alpha=[0.000000000000000000000000000000000000000000000000000000000000
       0.100000000000000000000000000000000000000000000000000000000000
       0.539357840802981787532485197881302436857273449701009015505500
       0.809036761204472681298727796821953655285910174551513523258250
       0.309036761204472681298727796821953655285910174551513523258250
       0.981074190219795268254879548310562080489056746118724882027805
       0.833333333333333333333333333333333333333333333333333333333333
       0.354017365856802376329264185948796742115824053807373968324184
       0.882527661964732346425501486979669075182867844268052119663791
       0.642615758240322548157075497020439535959501736363212695909875
       0.357384241759677451842924502979560464040498263636787304090125
       0.117472338035267653574498513020330924817132155731947880336209
       0.833333333333333333333333333333333333333333333333333333333333
       0.309036761204472681298727796821953655285910174551513523258250
       0.539357840802981787532485197881302436857273449701009015505500
       0.100000000000000000000000000000000000000000000000000000000000
       1.000000000000000000000000000000000000000000000000000000000000]';

beta=zeros(16,16);
```

```
beta(1,1)=0.100000000000000000000000000000000000000000000000000000000000;

beta(2,1)=-0.91517656137529144052001501927534215431895138766436972056466 0;
beta(2,2)=1.45453440217827322805250021715664459117622483736537873607016;

beta(3,1)=0.202259190301118170324681949205488413821477543637878380814562;
beta(3,3)=0.606777570903354510974045847616465241464432630913635142443687;

beta(4,1)=0.184024714708643575149100693471120664216774047979591417844635;
beta(4,3)=0.197966831227192369068141770510388793370637287463360401555746;
beta(4,4)=-0.0729547847313632629185146671595558023015011608914382961421311;

beta(5,1)=0.0879007340206681337319777094132125475918886824944548534041378;
beta(5,4)=0.410459702520260645318174895920453426088035325902848695210406;
beta(5,5)=0.482713753678866489204726942976896106809132737721421333413261;

beta(6,1)=0.0859700504902460302188480225945808401411132615636600222593880;
beta(6,4)=0.330885963040722183948884057658753173648240154838402033448632;
beta(6,5)=0.489662957309450192844507011135898201178015478433790097210790;
beta(6,6)=-0.0731856375070850736789057580558988816340355615025188195854775;

beta(7,1)=0.120930449125333720660378854927668953958938996999703678812621;
beta(7,5)=0.260124675758295622809007617838335174368108756484693361887839;
beta(7,6)=0.0325402621549091330158899334391231259332716675992700000776101;
beta(7,7)=-0.0595780211817361001560122202563305121444953672762930724538856;

beta(8,1)=0.110854379580391483508936171010218441909425780168656559807038;
beta(8,6)=-0.0605761488255005587620924953655516875526344415354339234619466;
beta(8,7)=0.321763705601778390100898799049878904081404368603077129251110;
beta(8,8)=0.510485725608063031577759012285123416744672137031752354067590;

beta(9,1)=0.112054414752879004829715002761802363003717611158172229329393;
beta(9,6)=-0.144942775902865915672349828340980777181668499748506838876185;
beta(9,7)=-0.333269719096256706589705211415746871709467423992115497968724;
beta(9,8)=0.499269229556880061353316843969978567860276816592673201240332;
beta(9,9)=0.509504608929686104236098690045386253986643232352989602185060;

beta(10,1)=0.113976783964185986138004186736901163890724752541486831640341;
beta(10,6)=-0.0768813364203356938586214289120895270821349023390922987406384;
beta(10,7)=0.239527360324390649107711455271882373019741311201004119339563;
beta(10,8)=0.397774662368094639047830462488952104564716416343454639902613;
beta(10,9)=0.0107558956873607455550609147441477450257136782823280838547024;
beta(10,10)=-0.327769124164018874147061087350233395378262992392394071906457;

beta(11,1)=0.0798314528280196046351426864486400322758737630423413945356284;
beta(11,6)=-0.0520329686800603076514949887612959068721311443881683526937298;
beta(11,7)=-0.0576954146168548881732784355283433509066159287152968723021864;
beta(11,8)=0.194781915712104164976306262147382871156142921354409364738090;
beta(11,9)=0.145384923188325069727524825977071194859203467568236523866582;
beta(11,10)=-0.0782942710351670777553986729725692447252077047239160551335016;
beta(11,11)=-0.114503299361098912184303164290554670970133218405658122674674;

beta(12,1)=0.985115610164857280120041500306517278413646677314195559520529;
beta(12,4)=0.330885963040722183948884057658753173648240154838402033448632;
beta(12,5)=0.489662957309450192844507011135898201178015478433790097210790;
beta(12,6)=-1.37896486574843567582112720930751902353904327148559471526397;
beta(12,7)=-0.861164195027635666673916999665534573351026060987427093314412;
beta(12,8)=5.78428813637537220022999785486578436006872789689499172601856;
beta(12,9)=3.28807761985103566890460615937314805477268252903342356581925;
beta(12,10)=-2.38633905093136384013422325215527866148401465975954104585807;
beta(12,11)=-3.25479342483643918654589367587788726747711504674780680269911;
beta(12,12)=-2.16343541686422982353954211300054820889678036420109999154887;

beta(13,1)=0.895080295771632891049613132336585138148156279241561345991710;
beta(13,3)=0.197966831227192369068141770510388793370637287463360401555746;
beta(13,4)=-0.0729547847313632629185146671595558023015011608914382961421311;
beta(13,6)=-0.851236239662007619739049371445966793289359722875702227166105;
beta(13,7)=0.398320112318533301719718614174373643336480918103773904231856;
beta(13,8)=3.63937263181035606029412920047090044132027387893977804176229;
beta(13,9)=1.54822877039830322365301663075174564919981736348973496313065;
beta(13,10)=-2.12221714704053716026062427460427261025318461146260124401561;
beta(13,11)=-1.58350398545326172713384349625753212757269188934434237975291;
beta(13,12)=-1.71561608285936264922031819751349098912615880827551992973034;
beta(13,13)=-0.0244036405750127452135415444412216875465593598370910566069132;

beta(14,1)=-0.915176561375291440520015019275342154318951387664369720564660;
beta(14,2)=1.45453440217827322805250021715664459117622483736537873607016;
beta(14,5)=-0.777333643644968233538931228575302137803351053629547286334469;
beta(14,7)=-0.0910895662155176069593203555807484200111889091770101799647985;
beta(14,13)=0.0910895662155176069593203555807484200111889091770101799647985;
beta(14,14)=0.777333643644968233538931228575302137803351053629547286334469;

beta(15,1)=0.100000000000000000000000000000000000000000000000000000000000;
beta(15,3)=-0.157178665799771163367058998273128921867183754126709419409654;
beta(15,15)=0.157178665799771163367058998273128921867183754126709419409654;

beta(16,1)=0.181781300700095283888472062582262379650443831463199521664945;
beta(16,2)=0.675000000000000000000000000000000000000000000000000000000000;
beta(16,3)=0.342758159847189839942220553413850871742338734703958919937260;
beta(16,4)=0.000000000000000000000000000000000000000000000000000000000000;
beta(16,5)=0.259111214548322744512977076191767379267783684543182428778156;
beta(16,6)=-0.358278966717952089048961276721979397739750634673268802484271;
beta(16,7)=-1.04594895940883306095050068756409905131588123172378489286080;
beta(16,8)=0.930327845415626983292300564432428777137601651182965794680397;
beta(16,9)=1.77950959431708102446142106794824453926275743243327790536000;
beta(16,10)=0.100000000000000000000000000000000000000000000000000000000000;
beta(16,11)=-0.282547569539044081612477785222287276408489375976211189952877;
beta(16,12)=-0.159327350119972549169261984373485859278031542127551931461821;
beta(16,13)=-0.145515894647001510860991961081084111308650130578626404945571;
beta(16,14)=-0.259111214548322744512977076191767379267783684543182428778156;
beta(16,15)=-0.342758159847189839942220553413850871742338734703958919937260;
beta(16,16)=-0.675000000000000000000000000000000000000000000000000000000000;

end
```

j. Force calculations

```
function [d g l]=gettotalaccel(K, rho, constants)
%This function calculates the Drag, Gravity , and Lift forces experienced
%by the satellite at each iteration
%
%Input: K = position and velocity vector of the state
%       rho = density at the current altitude
%       Constants = The constants that are required to solve for Gravity and
%               Drag [G (km^3/kg-s^2) M (kg) SA m omega_e]
%               G = Universal Gravitational Constant
%               M = Mass of the Earth
%               SA = Satellite Surface area
%               m = Satellite mass
%               omega_e = rotational velocity of the Earth

% constants assigned to variables
G=constants(1); M=constants(2); C_d=constants(6); C_l=constants(7);
SA=constants(3); m=constants(4); omega_e=constants(5); omega=[0 0 omega_e];

%Position and Velocity assigned to variable
x=K(:,1); vx=K(:,4);
y=K(:,2); vy=K(:,5);
z=K(:,3); vz=K(:,6);

%Vectors formed
R=[x y z]; V=[vx vy vz]; V_rel=V-cross(omega, R);

%Transform the Relative Velocity Vector
V_rel=Earth_Axis_To_Body_Axis(Body, V_rel);
V_rel=Body_Axis_To_Wind_Axis(Wind, V_rel');
r=norm(R);
```

```
v=norm(V_rel);

%calculate the force from gravity, drag, and lift
d=(-.5*rho*v^2*SA*C_d)*1000;
l=(-.5*rho*v^2*SA*C_l)*1000;
g=(-G*M*m/r^2)*1000;

end
```

k. Atmospheric Model

```
function [R0 rho_i rho_set]=density122211(yn, i, R0, rho_set)
%This function creates the atmospheric Density profile from the USSA-76
%model
%
%Inputs:yn = state variable [x y z vx vy vz]
%       i = iteration step
%       R0 = Initial Radius
%       rho_set = atmospheric density profile
%Output:R0 = initial Radius
%       rho_i = the density at the given altitude
%       rho_set = the atmospheric density profile

format long

%State vectors to variables
initial=yn;
x0=initial(1); y0=initial(2); z0=initial(3);

%The altitude currently located
r0=sqrt(x0^2+y0^2+z0^2);
alt=r0-6378;
alt=round(alt);

%If this is the first iteration the atmospheric profile is created for 0 to
%1000 km.  The entire profile is made so that it will not have to be
%created again.
if i==2
   R0=r0;
   [rho]=atmodensity(1000);
   rho_set=rho;
end

%For every other iteration the density at the altitude is assigned properly
rho_i=rho_set(alt+1);
```

```
end
%%
function [rho]=atmodensity(alt)
%This function is a supplement to the density profile created in 122211
%funtion.  It initialized the USSA-76 generator.
%
%Input: alt = the altitude at which to create the profile to
%Output: rho = Density profile

format long
[rho]=atmo(alt,1,1);

rho=rho*1000^3;

end
%%
function [rho] = atmo(alt,division,units)
format long
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Program:    1976 Standard Atmosphere Calculator[0-1000 km]
%   Author:     Brent Lewis(RocketLion@gmail.com)
%               University of Colorado-Boulder
%   History:    Original-1/10/2007
%               Revision-1/12/2007-Corrected for changes in Matlab versions
%               for backward compatibility-Many thanks to Rich
%               Rieber(rrieber@gmail.com)
%   Input:      alt:        Final Geometric Altitude[km]
%               division:   Reporting points for output arrays[km]
%                           (.01 km & Divisible by .01 km)
%               units:      1-[Metric]
%                           2-{English}
%   Default:    Values used if no input
%               alt:        1000 km
%               division:   1 km
%               units:      Metric
%   Output:     Each value has a specific region that it is valid in with this model
%               and is only printed out in that region
%               [Z, Z_L, Z_U, T, P, rho, c, g, mu, nu, k, n, n_sum]
%               Z:          Total Reporting Altitudes[0<=alt<=1000 km][km]{ft}
%               Z_L:        Lower Atmosphere Reporting Altitudes[0<=alt<=86 km][km]{ft}
%               Z_U:        Upper Atmosphere Reporting Altitudes[86<=alt<=1000 km][km]{ft}
%               T:          Temperature array[0<=alt<=1000 km][K]{R}
%               P:          Pressure array[0<=alt<=1000 km][Pa]{in_Hg}
%               rho:        Density array[0<=alt<=1000 km][kg/m^3]{lb/ft^3}
%               c:          Speed of sound array[0<=alt<=86 km][m/s]{ft/s}
%          g:       Gravity array[0<=alt<=1000 km][m/s^2]{ft/s^2}
%          mu:       Dynamic Viscosity array[0<=alt<=86 km][N*s/m^2]{lb/(ft*s)}
%          nu:      Kinematic Viscosity array[0<=alt<=86 km][m^2/s]{ft^2/s}
%          k:       Coefficient of Thermal Conductivity
%                   array[0<=alt<=86 km][W/(m*K)]{BTU/(ft*s*R)}
%          n:       Number Density of individual gases
%                   (N2 O O2 Ar He H)[86km<=alt<=1000km][1/m^3]{1/ft^3}
%          n_sum:    Number Density of total gases
%                   [86km<=alt<=1000km][1/m^3]{1/ft^3}
%   Acknowledgements:     1976 U.S. Standard Atmosphere
%                   Prof. Adam Norris-Numerical Analysis Class
%                   Steven S. Pietrobon USSA1976 Program
%   Notes:              Program uses a 5-point Simpson's Rule in 10
%                   meter increments.  Results DO vary by less 1%
%                   compared to tabulated values and is probably
%                   caused by different integration techniques
%   Examples:             atmo() will compute the full atmosphere in 1 km
%                   increments and output in Metric Units
%                   atmo(10) will compute the atmosphere between 0
%                   and 10 km in 1 km increments and output in
%                   Metric Units
%                   atmo(20,.1,2) will compute the atmosphere
%                   between 0 and 20 km in 100 m increments and
%                   output in English Units
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if nargin == 0
   alt = 1000;
   division = 1;
   units = 1;
elseif nargin == 1
   division = 1;
   units = 1;
elseif nargin == 2
   units = 1;
end

%   Error Reporting
if nargin > 3
   error('Too many inputs')
elseif mod(division,.01) ~= 0
   error('Divisions must be multiples of .01 km')
elseif units ~= 1 && units ~= 2
   error('Units Choice Invalid[1-Metric,2-English]')
elseif alt<0 || alt>1000
   error('Program only valid for 0<altitudes<1000 km')
```

```
end

%   Matrix Pre-allocation
if alt <= 86
    Z_L = (0:division:alt)';
    Z_U = [];
    n = [];
else
    Z_L = (0:division:86)';
    Z_U = (86:division:alt)';
    if mod(86,division) ~= 0
        Z_L = [Z_L; 86];
    end
    if mod(alt-86,division) ~= 0
        Z_U = [Z_U; alt];
    end
end
T_L = zeros(size(Z_L));
T_M_L = T_L;
T_U = zeros(size(Z_U));

%   Conversion Factor Used in 80<alt<86 km
Z_M = 80:.5:86;
M_M_0 = [1 .999996 .999989 .999971 .999941 .999909 ...
    .999870 .999829 .999786 .999741 .999694 .999641 .999579];

%   Constants
M_0 = 28.9644;
M_i = [28.0134; 15.9994; 31.9988; 39.948; 4.0026; 1.00797];
beta = 1.458e-6;
gamma = 1.4;
g_0 = 9.80665;
R = 8.31432e3;
r_E = 6.356766e3;
S = 110.4;
N_A = 6.022169e26;

%   Temperature
for i = 1 : length(Z_L)
    T_L(i,1) = atmo_temp(Z_L(i));
    T_M_L(i,1) = T_L(i,1);
    if Z_L(i) > 80 && Z_L(i) < 86
        T_L(i,1) = T_L(i)*interp1(Z_M,M_M_0,Z_L(i));
    end
end
for i = 1 : length(Z_U)
```

```
    T_U(i,1) = atmo_temp(Z_U(i));
end

%   Number Density
if alt > 86
    n = atmo_compo(alt,division);
    n_sum = sum(n,2);
else
    n = [];
    n_sum = [];
end

%   Pressure
P_L = atmo_p(Z_L);
P_U = atmo_p(Z_U,T_U,n_sum);

%   Density
rho_L = M_0*P_L./(R*T_M_L);
if ~isempty(P_U)
    rho_U = n*M_i/N_A;
else
    rho_U = [];
end

%   Speed of Sound
c = sqrt(gamma*R*T_M_L/M_0);
%   Dynamic Viscosity
mu = beta*T_L.^1.5./(T_L+S);
%   Kinematic Viscosity
nu = mu./rho_L;
%   Thermal Conductivity Coefficient
k = 2.64638e-3*T_L.^1.5./(T_L+245*10.^(-12./T_L));

%   Combine Models
T = [T_L(1:end-1*double(~isempty(T_U)));T_U];
P = [P_L(1:end-1*double(~isempty(T_U)));P_U];
rho = [rho_L(1:end-1*double(~isempty(T_U)));rho_U];
Z = [Z_L(1:end-1*double(~isempty(T_U)));Z_U];

%   Gravity
g = g_0*(r_E./(r_E+Z)).^2;

if units == 2
    unit_c = [3.048e-1 3.048e-1 3.048e-1 5/9 0.0001450377 1.6018463e1...
        3.048e-1 3.048e-1 1.488163944 9.290304e-2 6.226477504e-3...
        3.531466672e2 3.531466672e2];
```

```
   Z = Z/unit_c(1);
   Z_L = Z_L/unit_c(2);
   Z_U = Z_U/unit_c(3);
   T = T/unit_c(4);
   P = P/unit_c(5);
   rho = rho/unit_c(6);
   c = c/unit_c(7);
   g = g/unit_c(8);
   mu = mu/unit_c(9);
   nu = nu/unit_c(10);
   k = n/unit_c(11);
   n_sum = n_sum/unit_c(12);
end
end
%%
function Temp = atmo_temp(alt)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Program:    Atmospheric Temperature Calculation
%   Author:     Brent Lewis(RocketLion@gmail.com)
%               University of Colorado-Boulder
%   History:    Original-1/09/2007
%   Input:      alt:    Geometric Altitude of desired altitude[scalar][km]
%   Output:     Temp:   Temperature at desired altitude using values from
%                       1976 Standard Atmosphere[K]
%   Note:       Only Valid Between 0 km and 1000 km
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%   Constants
r_E = 6.356766e3;
epsilon = 1e5*eps;

%   Defined temperatures at each layer
T = [288.15 216.65 216.65 228.65 270.65 270.65 ...
   214.65 186.95 186.8673 240 360 1000];
L = [-6.5 0 1 2.8 0 -2.8 -2 0 0 12 0];

%   Geopotential/Geometric Altitudes used for Geometric Altitudes < 86 km
H = [0 11 20 32 47 51 71];
Z = r_E*H./(r_E-H);
%   Geometric Altitudes used for Altitudes >86 km
Z(8:12) = [86 91 110 120 1000];

if alt < Z(1) || alt > (Z(12)+epsilon)
   error('Altitude must be 0-1000 km')
```

```
end

%   Temperature Calculation with Molecular Temperature below 86 km and
%   Kinetic Temperature above
if alt >= Z(1) && alt <= Z(8)
   Temp = interp1(Z,T,alt);
elseif alt > Z(8) && alt <= Z(9)
   Temp = T(9);
elseif alt > Z(9) && alt <= Z(10)
   a = 19.9429;
   A = -76.3232;
   T_c = 263.1905;
   Temp = T_c+A*sqrt(1-((alt-Z(9))/a)^2);
elseif  alt > Z(10) && alt <= Z(11)
   Temp = interp1(Z,T,alt);
elseif alt > Z(11)
   lambda = L(10)/(T(12)-T(11));
   xi = (alt-Z(11))*(r_E+Z(11))/(r_E+alt);
   Temp = T(12)-(T(12)-T(11))*exp(-lambda*xi);
end
end
%%
function n_i_array = atmo_compo(alt,division)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Program:    High Altitude Atmospheric Composition Calculation
%   Author:     Brent Lewis(RocketLion@gmail.com)
%               University of Colorado-Boulder
%   History:    Original-1/10/2007
%               Revision-1/12/2007-Corrected for changes in Matlab versions
%               for backward compatibility-Many thanks to Rich
%               Rieber(rrieber@gmail.com)
%   Input:      alt:        Geometric Altitude of desired altitude[scalar][km]
%               division:   Desired output altitudes
%   Output:     n_i_array:  Array of compositions of [N2 O O2 Ar He H] at
%                           desired reporting altitudes using equations
%                           from 1976 Standard Atmosphere
%   Note:       Only Valid Between 86 km and 1000 km
%               Division must be a multiple of 10 m;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Z_i = [86 91 95 97 100 110 115 120 150 500 1000];
step = .01;

if alt < Z_i(1) || alt>Z_i(length(Z_i))
```

```
    n_i_array = [];
    return;
end

%   Gas coefficients
alpha_i = [0; 0; 0; 0; -.4; -.25];
a_i = [0; 6.986e20; 4.863e20; 4.487e20; 1.7e21; 3.305e21];
b_i = [0; .75; .75; .87; .691; .5];
Q_i = [0; -5.809644e-4; 1.366212e-4; 9.434079e-5; -2.457369e-4];
q_i = [0; -3.416248e-3; 0; 0; 0];
U_i = [0; 56.90311; 86; 86; 86];
u_i = [0; 97; 0; 0; 0];
W_i = [0; 2.70624e-5; 8.333333e-5; 8.333333e-5; 6.666667e-4];
w_i = [0; 5.008765e-4; 0; 0; 0];

%   Gas Data
R = 8.31432e3;
phi = 7.2e11;
T_7 = 186.8673;
T_11 = 999.2356;
%   Molecular Weight & Number Density based on values at 86 km & 500 km for
%   Hydrogen
n_i_86 = [1.129794e20; 8.6e16; 3.030898e19; 1.3514e18; 7.5817e14; 8e10];
n_i_alt = n_i_86;
sum_n = [ones(3,1)*n_i_86(1);ones(2,1)*sum(n_i_86(1:3));sum(n_i_86(1:5))];
M_i = [28.0134; 15.9994; 31.9988; 39.948; 4.0026; 1.00797];
M_0 = 28.9644;

n_int = zeros(size(n_i_86));
j = 1;
n_i_array = zeros(floor((alt-86)/division)+1,6);
for i = 1 : length(Z_i)-1
    if alt > Z_i(i)
        Z_start = Z_i(i);
        if alt > Z_i(i+1)
            Z_end = Z_i(i+1);
        else
            Z_end = alt;
        end
        for Z_0 = Z_start:step:Z_end-step
            Z_1 = Z_0+step;
            if Z_1 <= Z_i(5)
                M = ones(size(M_i))*M_0;
            else
                M = [(n_i_alt(1)*M_i(1))./sum_n(1:3);...
                    sum((n_i_alt(1:3).*M_i(1:3)))./sum_n(4:5);...
                    sum((n_i_alt(1:5).*M_i(1:5)))./sum_n(6)];
            end
            sum_n = [ones(3,1)*n_i_alt(1);ones(2,1)*sum(n_i_alt(1:3));sum(n_i_alt(1:5))];
            n_int = f_n(a_i,alpha_i,b_i,M,M_i,n_int,phi,...
                Q_i,q_i,R,sum_n,U_i,u_i,W_i,w_i,Z_i,Z_0,Z_1);
            n_i_alt(1:5) = n_i_86(1:5)*T_7/atmo_temp(Z_1).*exp(-n_int(1:5));
            if Z_1 < Z_i(9)
                n_i_alt(6) = 0;
            else
                tau = int_tau(alt);
                n_i_alt(6) = (T_11/atmo_temp(Z_1))^(1+alpha_i(6))*...
                    (n_i_86(6)*exp(-tau)-n_int(6));
            end
            if mod(Z_0,division) == 0
                n_i_array(j,:) = n_i_alt';
                j = j+1;
            end

        end
    end
end
n_i_end(1:5) = n_i_86(1:5)*T_7/atmo_temp(alt).*exp(-n_int(1:5));
if alt < Z_i(9)
    n_i_end(6) = 0;
else
    tau = int_tau(alt);
    n_i_end(6) = (T_11/atmo_temp(Z_1))^(1+alpha_i(6))*...
        (n_i_86(6)*exp(-tau)-n_int(6));
end
n_i_array(j,:) = n_i_end;
end
%%
function P = atmo_p(alt, T, sum_n)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Program:    Atmospheric Pressure Calculation
%   Author:     Brent Lewis(RocketLion@gmail.com)
%               University of Colorado-Boulder
%   History:    Original-1/10/2007
%               Revision-1/12/2007-Corrected for changes in Matlab versions
%               for backward compatability-Many thanks to Rich
%               Rieber(rrieber@gmail.com)
%   Input:      alt:    Geometric altitude vector of desired pressure data[km]
%               T:      Temperature vector at given altitude points
%                       Required only for altitudes greater than 86 km[K]
%               sum_n:  Total number density of atmospheric gases[1/m^3]
%   Output:     P:      Pressure vector[Pa]
%   Note:       Must compute altitudes below 86 km and above 86 km on two
%               different runs to allow line up of altitudes and
%               temperatures
%   Examples:   atmo_p(0) = 101325 Pa
%               atmo_p(0:10) = Pressures between 0-10 km at 1 km increments
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if nargin == 1
   T = [];
   sum_n = [];
end

%   Constants
N_A = 6.022169e26;
g_0 = 9.80665;
M_0 = 28.9644;
R = 8.31432e3;
r_E = 6.356766e3;
%   Geopotential/Geometric Altitudes used for Geometric Altitudes < 86 km
H = [0 11 20 32 47 51 71 84.852];
Z = r_E*H./(r_E-H);
Z(8) = 86;

%   Defined temperatures/lapse rates/pressures/density at each layer
T_M_B = [288.15 216.65 216.65 228.65 270.65 270.65 214.65];
L = [-6.5 0 1 2.8 0 -2.8 -2]/1e3;
P_ref = [1.01325e5 2.2632e4 5.4748e3 8.6801e2 1.1090e2 6.6938e1 3.9564];

%   Preallocation of Memory
P = zeros(size(alt));

for i = 1 : length(alt)
   Z_i = alt(i);
   if isempty(sum_n)
      index = find(Z>=Z_i)-1+double(Z_i==0);
      index = index(1);
      Z_H = r_E*Z_i/(r_E+Z_i);
      if L(index) == 0
         P(i) = P_ref(index)*exp(-g_0*M_0*(Z_H-H(index))*1e3/(R*T_M_B(index)));
      else
         P(i) = P_ref(index)*(T_M_B(index)/...
            (T_M_B(index)+L(index)*(Z_H-H(index))*1e3))^...
            (g_0*M_0/(R*L(index)));
      end
   else
```

```
        P(i) = sum_n(i)*R*T(i)/N_A;
    end
end
end
%%
function n_int = f_n(a_i,alpha_i,b_i,M,M_i,n_int,phi,...
    Q_i,q_i,R,sum_n,U_i,u_i,W_i,w_i,Z_i,Z_0,Z_1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Program:    Gas Integral Program
%   Author:     Brent Lewis(RocketLion@gmail.com)
%               University of Colorado-Boulder
%   History:    Original-1/10/2007
%   Input:      As defined in 1976 Standard Atmosphere
%   Output:     n_int:  Integral values computed using 5-point Simpsons
%                       Rule
%   Note:       Created for running in Atmospheric Program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%   Constants
g_0 = 9.80665;
r_E = 6.356766e3;
T_c = 263.1905;
A_8 = -76.3232;
a_8 = 19.9429;
L_K_9 = 12;
T_7 = 186.8673;
T_9 = 240;
T_10 = 360;
T_11 = 999.2356;
T_inf = 1000;

%   Molecular Diffusion Coefficients
K_7 = 1.2e2;

alt_j = linspace(Z_0,Z_1,5);
n_i = zeros(6,length(alt_j));
for j = 1 : length(alt_j)
    Z = alt_j(j);
    %    Temperature Values
    if Z < Z_i(2)
        T = T_7;
        dT_dZ = 0;
    elseif Z < Z_i(6)
        T = T_c+A_8*sqrt(1-((Z-Z_i(2))/a_8)^2);
        dT_dZ = -A_8/a_8^2*(Z-Z_i(2))*(1-((Z-Z_i(2))/a_8)^2)^-.5;
    elseif Z < Z_i(8)
        T = T_9+L_K_9*(Z-Z_i(6));
        dT_dZ = L_K_9;
    elseif Z >= Z_i(8)
        lambda = L_K_9/(T_inf-T_10);
        xi = (Z-Z_i(8))*(r_E+Z_i(8))/(r_E+Z);
        T = T_inf-(T_inf-T_10)*exp(-lambda*xi);
        dT_dZ = lambda*(T_inf-T_10)*((r_E+Z_i(8))/(r_E+Z))^2*exp(-lambda*xi);
    end
    %     K Values
    if Z < Z_i(3)
        K = K_7;
    elseif Z < Z_i(7)
        K = K_7*exp(1-400/(400-(Z-95)^2));
    elseif Z >= Z_i(7)
        K = 0;
    end
    %     Gravity
    g = g_0*(r_E/(r_E+Z))^2;

    %     N
    if Z <= Z_i(5)
        M_N2 = M(1);
    else
        M_N2 = M_i(1);
    end
    n_i(1,j) = M_N2*g/(T*R)*1e3;
    %     O O2 Ar He
    D = a_i(2:5).*exp(b_i(2:5).*log(T/273.15))./sum_n(2:5);
    if K ~= 0
        f_Z = (g/(R*T)*D./(D+K).*(M_i(2:5)+M(2:5)*K./D+...
            alpha_i(2:5)*R/g*dT_dZ/1e3))*1e3;
    else
        f_Z = (g/(R*T)*(M_i(2:5)+alpha_i(2:5)*R/g*dT_dZ/1e3))*1e3;
    end
    if Z <= Z_i(4)
        vdk = Q_i(2:5).*([Z;Z;Z;Z]-U_i(2:5)).^2.*exp(-W_i(2:5).*...
            ([Z;Z;Z;Z]-U_i(2:5)).^3)+q_i(2:5).*(u_i(2:5)-[Z;Z;Z;Z]).^2.*...
            exp(-w_i(2:5).*(u_i(2:5)-[Z;Z;Z;Z]).^3);
    else
        vdk = Q_i(2:5).*([Z;Z;Z;Z]-U_i(2:5)).^2.*exp(-W_i(2:5).*...
            ([Z;Z;Z;Z]-U_i(2:5)).^3);
    end
    n_i(2:5,j) = f_Z+vdk;
    %     H
```

```
    if Z_1 < 150 || Z_1 > 500
        n_i(6,j) = 0;
    else
        D_H = a_i(6)*exp(b_i(6)*log(T/273.15))/sum_n(6);
        n_i(6,j) = phi/D_H*(T/T_11)^(1+alpha_i(6));
    end

end
h = alt_j(2)-alt_j(1);
n_int = n_int+(2*h/45*(7*n_i(:,1)+32*n_i(:,2)+12*n_i(:,3)+32*n_i(:,4)+7*n_i(:,5)));
end
%%
function TAU = int_tau(Z)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Program:    Tau Integral Computation for Hydrogen Composition Program
%               in Atmospheric Model
%   Author:     Brent Lewis(RocketLion@gmail.com)
%               University of Colorado-Boulder
%   History:    Original-1/10/2007
%   Input:      Z:      Altitude value
%   Output:     TAU:    Integral Value
%   Note:       This program computes the value of Tau directly with the
%               integral done by hand and only the second integration limit
%               needing to be inputted
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%   Constants
L_K_9 = 12;
T_10 = 360;
T_inf = 1000;
Z_10 = 120;
g_0 = 9.80665;
r_E = 6.356766e3;
R = 8.31432e3;
lambda = L_K_9/(T_inf-T_10);
M_H = 1.00797;

%   Value of Integration limit computed previously
tau_11 = 8.329503912749350e-004;

tau_Z = M_H*g_0*r_E^2/R*...
    log((exp(lambda*(Z-Z_10)*(r_E+Z_10)/(r_E+Z))-1)*T_inf+T_10)/...
    (lambda*T_inf*(r_E+Z_10)^2);
```

```
TAU = tau_Z-tau_11;
end
```

l. Plots

```
function Three_D_Orbit(tgrid, ansxy)
%This function create a 3-D orbit from the propagated states
%

x=ansxy(:,1); y=ansxy(:,2); z=ansxy(:,3);
x0=x; y0=y; z0=z;

for a=1:length(x0)
   w(a)=y0(a)/x0(a);
end
theta=atan(w); theta=theta';

for b=1:length(theta)
   thetab=theta(b); xb=x0(b); yb=y0(b);

   if xb>=0
      xx=xb-6378*cos(thetab);
      yy=yb-6378*sin(thetab);
   elseif xb<0
      xx=xb+6378*cos(thetab);
      yy=yb+6378*sin(thetab);
   end

   x0(b)=xx; y0(b)=yy;
end

figure
plot3(x0,y0,z0, '--x', 'MarkerSize',12);
axis square
xlabel('Altitude (km)')
ylabel('Altitude (km)')
zlabel('Altitude (km)')
title('10th Order RK Orbit Model with Drag @ 3.5 orbits')
hleg1=legend('R=6553 km, Iteration per 66 seconds');
set(hleg1,'Location','Best')

end
%%
function plot_force(tgrid, F)
%This function plots the forces experienced by the satellite over the
%duration of the model
```

```
t=tgrid; t(1)=[];

figure
plot(t, F(:,1),'--x', 'MarkerSize', 10);
xlabel('Iteration')
ylabel('Force (N)')
title('Gravity Force as Altitude Increases')

figure
plot(t, F(:,2), '--x', 'MarkerSize', 10);
xlabel('Iteration')
ylabel('Force (N)')
title('Drag Force as Altitude Increases')

figure
plot(t, F(:,3), '--x', 'MarkerSize', 10);
xlabel('Iteration')
ylabel('Lift (N)')
title('Lift Force as Altitude Increases')

end
```