Spring 2008

Analysis of access-to-space missions utilizing on-board energy management and entropic analysis

Tyler Winter

Department:

ANALYSIS OF ACCESS-TO-SPACE MISSIONS UTILIZING ON-BOARD ENERGY MANAGEMENT AND ENTROPIC ANALYSIS

by

TYLER FORREST WINTER

A THESIS

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN AEROSPACE ENGINEERING

2008

Approved by

Dr. David W. Riggins, Advisor
Dr. Henry J. Pernicka
Col. Thomas D. Akers, USAF, Ret.

ABSTRACT

The overall objective of this study is the thermodynamically consistent quantification of second-law and performance losses for vehicle configurations implemented or suggested for access-to-space missions and the development of the relationships between the mission entropy generation and overall mission performance. The detailed theory which allows thermodynamically rigorous loss accounting (entropy generation and direct relationship to vehicle performance) is developed and discussed for aerospace vehicles. A full vehicle trajectory code utilizing simplified models for multi-stage rockets and air-breathing propulsion systems is developed and validated specifically for use as a testbed for second law-based theory and concepts. Results are shown in which a multi-stage rocket-powered vehicle and a rocket/air-breathing (combined cycle) vehicle are compared in detail, both in terms of conventional information as well as loss and energy utilization analysis incorporating the second law of thermodynamics. The analysis code is highly modular and improvements can and will be incorporated in terms of aerodynamics, propulsion, and sub-systems weights modeling. This work represents the first complete loss analysis as obtained from basic thermodynamic principles of general access-to-space vehicles and missions.

ACKNOWLEDGMENTS

I would like to express my utmost regard and appreciation for my advisor, Dr. David Riggins. His intellectual insight and guidance has proven invaluable to my individual, educational, and professional development. I would also like to thank my committee; Dr. Henry Pernicka and Col. Tom Akers for their time and assistance instructing me and reviewing this thesis.

Finally, I would like to thank my parents for their continual love, encouragement, and support throughout this endeavor and my entire life.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Page

LIST OF TABLES

Page

NOMENCLATURE

Symbol	Description
T	Thrust Force
D	Drag Force
L	Lift Force
m_v	Mass of Vehicle
g	Gravitational Acceleration
θ	Vehicle Body Angle
φ	Global Vehicle Rotation Angle
$dV_{x'}$	Differential Change in Velocity in the x'-direction
$dV_{y'}$	Differential Change in Velocity in the y'-direction
dt	Differential Time Step
V_{new}	Velocity at New Time Step
V_{old}	Velocity at Previous Time Step
θ_{new}	Vehicle Body Angle at New Time Step
θ_{old}	Vehicle Body Angle at Previous Time Step
$d\theta$	Differential Change in Vehicle Body Angle
x_{new}	x-coordinate at New Time Step
y_{new}	y-coordinate at New Time Step
x_{old}	x-coordinate at Previous Time Step
y_{old}	y-coordinate at Previous Time Step
h	Altitude
R_e	Radius of Earth
x	Global x-coordinate
y	Global y-coordinate
g_0	Gravitational Acceleration at Sea-level
T(h)	Temperature as a Function of Altitude
P(h)	Pressure as a Function of Altitude
ρ(h)	Density as a Function of Altitude
R_{gas}	Gas Constant for Air

a_0	Slope of Altitude-Temperature Gradient
T_r	Thrust of Rocket
$\dot{m}_{propellant}$	Mass Flow Rate of Propellant
C	Effective Exhaust Velocity
T_{ab}	Thrust of Air-Breather
$\dot{m}_{air}$	Mass Flow Rate of Air
C_p	Specific Heat due to Constant Pressure
T_{t4}	Total Temperature in Scramjet Burner
V_∞	Free-stream Velocity
M_∞	Free-stream Mach Number
C_{lw}	Coefficient of Lift due to the Wing
α	Angle of Attack
C_{dw}	Coefficient of Drag due to the Wing
$C_{db,r}$	Coefficient of Drag due to the Rocket Body
β	Shockwave Angle
γ	Ratio of Specific Heats
$C_{db,ab}$	Coefficient of Drag due to the Air-Breather Body
F_{ab}	Air-Breather Mass Capture Factor
H_{fuel} (H_{prop})	Heating Value of Fuel
$dS_{irr(total)}$	Differential Change in Entropy Associated with all Irreversibilities
dh	Differential Change in Altitude
$\Delta\vec{V}$	Required Change in Velocity Vector to Circularize Orbit
$\vec{V}_{circular}$	Circular Orbit Velocity Vector
λ	Angle Between Vehicle Velocity Vector and Orbit Velocity Vector
m_0	Initial Vehicle Mass
RABCC	Rocket/Air-Breathing Combined Cycle
λ_P	Vehicle Propellant Mass Fraction

1. INTRODUCTION

1.1. OVERVIEW: SPACE ACCESS MISSIONS

The necessity of efficient vehicle design and operation in order to maximize final payload-to-orbit is paramount when considering access-to-space applications. Access-to-space missions demand the careful consideration of a variety of complexities intrinsic to optimal vehicle design and overall mission-based performance. These issues can range from large vehicle accelerations to excess vehicle heating depending on the flight path traversed. Other concerns include proper methodology/techniques for vehicle body angle control (i.e. aerodynamic, propulsive, etc.) as well as general trajectory selection. Typically, vehicle selection dictates the ideal trajectory path into space. For example, a multi-stage rocket will generally maintain a trajectory that will result in its escape from the atmosphere relatively quickly, hence reducing the amount of time the vehicle is exposed to aerodynamic heating and drag. On the other hand, an air-breathing configuration will inevitably fly within the atmosphere much longer in order to increase (as much as practically possible) the vehicle velocity where atmospheric oxygen can be effectively inducted, before reaching space. To mitigate heat loads experienced, an air-breathing vehicle is also more likely to fly in such a way as to maintain a constant dynamic pressure for sustained times. Regardless of the vehicle and trajectory, there exists a need to evaluate and understand the capabilities, performance, and particularly the losses associated with an aerospace vehicle access-to-space mission

Analysis from a second law standpoint allows for both preliminary and detailed insight into the feasibility and characteristics of access-to-space missions when comparing candidate vehicles. By determining and evaluating in detail the total entropy generation of a given aerospace vehicle throughout a mission, proper tallying of losses and loss impact on vehicle and mission performance can be conducted. This thesis develops the theoretical basis for such an investigation as well as a complete system-integrated access-to-space trajectory solver routine with necessary models for aerodynamics, propulsion, and loss analysis. It then applies this trajectory model to known candidate access-to-space missions, including the initial stages of the Apollo configuration (to orbit) and then candidate rocket and combined cycle (rocket/air-

breathing) single stage to orbit configurations. Loss accounting in terms of both entropy generation and necessary propellant mass fractions necessary are fully analyzed.

1.2. TRAJECTORY SOLVER

The initial task of this research was to create a trajectory code that would allow various aerospace vehicles to be assessed in terms of both vehicle performance and the associated loss analysis of particular interest here. Emphasis for this work in terms of main purpose and scope was on access-to-space missions, although theory, trajectory/vehicle routine and some examples studied are quite general in nature, i.e. can apply to aerospace vehicles and missions of any type. It was considered critical to develop a fast and reliable trajectory solver that would provide useful and quantitative results (within the necessary constraint of relatively simple approximations and assumptions regarding aerodynamics, propulsion, control, and weights modeling) as well as to allow reasonable time frames in terms of computational time for assessing trajectory/loss histories for specific vehicle configurations. This entailed the construction of various models used to describe the necessary forces (i.e. thrust, lift, drag, etc.) and the environment encountered by an aerospace vehicle used for access-to-space. Specifically, three vehicle models were constructed. The models corresponded to a single stage rocket, a rocket/air-breathing combined cycle vehicle, and a multi-stage rocket. Furthermore, atmospheric property and vehicle control models were also developed. Lastly, the second law and vehicle equations of motion were combined to form a relationship in which the associated vehicle losses could be calculated differentially and summed across a mission and directly related to the theoretical analysis discussed below. This analysis ultimately enabled vehicle mass fractions (in terms of propellant required to overcome losses and to facilitate kinetic/potential energy changes) to be computed and compared for any simulated mission.

1.3. ENTROPIC ANALYSIS

The design and analysis of many different engineering systems have significantly benefited from determining and then maximizing the work potential (i.e. minimize work potential losses) across all stages of operation of the system. The maximization of work

potential within given thermodynamic constraints is equivalent to minimization of entropy generation due to irreversibilities and can be viewed as necessary to proper energy management for any system. For aerospace systems this necessitates the complete understanding and interplay of force interactions which determine vehicle performance and the entropy generation which results from the force interactions. However, with the ability to quantify various losses in a given system in terms of entropy generation as well as the relationship of the entropy generation to vehicle force-based performance, a system-level approach can be taken to optimize overall mission performance. Specifically, in this work, the relationship between force-based vehicle performance and entropy generation is derived in detail and results will be studied analytically. In addition, the method will be applied to representative access-to-space missions utilizing a differentially based trajectory routine (as noted above).

1.4. THESIS OUTLINE

This document is composed of four different sections. The first main section is a literature survey of representative work associated with trajectory analysis, methodologies, and techniques, especially for access-to-space mission vehicles. This section also includes a summary of some previous work related to the entropic analysis of aerospace vehicles. The second main section develops in detail the second law theory and equations relevant to a vehicle in atmospheric flight both at an instant and as integrated across a mission. This culminates in the description of the vehicle mass fractions in terms of propellant fractions necessary for kinetic and potential energy changes and the propellant necessary to overcome all losses due to irreversibility.

The third section of the thesis details the basic assumptions and modeling incorporated into the trajectory solver which was developed in this work. Specifically, the two-dimensional equations of motion are formulated and manipulated, and the modeling of the Earth in the trajectory model is discussed. The model used for the calculation of atmospheric properties at all altitudes is discussed. The propulsion system and external aerodynamics models as well as energy management issues relevant to access-to-space vehicles and missions are described. This third section closes with a

summary of the relatively complex methodology (even with simplified models) necessary for meaningful simulation of access-to-space missions.

The final (fourth) main section describes the results obtained using the trajectory solver and associated loss analysis for two specific missions. The first mission involves validating the trajectory solver simulated results against actual data from the famous Apollo 11 flight. The second mission shows a comparison between a multi-stage rocket and an air-breathing combined cycle vehicle. Both of these mission analyses culminate in the quantification of a corresponding on-board energy usage and provide detailed and comparative vehicle mass fraction analysis.

2. LITERATURE SURVEY

The following review of literature separately considers three main subject areas relevant to the current study. The first area involves trajectory modeling and simulation for aerospace vehicles; of interest is the variety of methodologies and techniques previously and currently used to predict or simulate the trajectory of an aerospace vehicle. The second topic is focused on previous work regarding different space access missions and vehicles of specific interest to the current investigation. The third area involves a summary of selected previous work in the area of entropic analysis as applied to aerospace systems.

2.1. METHODS OF TRAJECTORY DEVELOPMENT

There are two main industry-standard trajectory optimization programs; these are the Program to Optimize Simulated Trajectories (POST) [1] and the Optimal Trajectories by Implicit Simulation (OTIS) [2]. POST and OTIS were developed by Lockheed Martin Astronautics/NASA Langley Research Center and the Boeing Corporation/NASA Glenn Research Center, respectively. Each of these sophisticated optimization tools implement different methods used in solving for a trajectory. Actually, OTIS provides two modes in which a trajectory can be determined. Both POST and OTIS are able to perform a time integration in which a direct shooting method is used to determine the optimal trajectory. Additionally, however, OTIS has the option to use a collocation method to implicitly determine a trajectory solution. More recently, Windhorst [3] discusses the development of an alternative trajectory tool known as *Mission*. This tool incorporates the conventional gradient-based optimization similar to POST and OTIS but also is capable of implementing a genetic algorithm. It is important to note that regardless of the program used each inevitably must satisfy the basic governing physics and equations of motion associated with an aerospace vehicle.

While both OTIS and POST are proven trajectory optimizers, an alternative trajectory solver was chosen to be developed since loss analysis quantification associated with space access vehicles was the primary objective not trajectory optimization. Not only does individual development of a trajectory solver provide challenging instructive

opportunities, it also allows familiarity with the source code which enables a user to have complete customization of any additional analysis desired.

2.2. ACCESS-TO-SPACE MISSIONS

Several space access missions were reviewed with specific focus on vehicle configurations. Orloff [4] conducted a classical comparison of a single-stage-to-orbit (SSTO) rocket and air-breathing vehicles. His analysis utilized HySIDE, a code developed by the Astrox Corporation. He concluded that air-breathing systems prevail over rockets in the category of SSTO configurations. Nevertheless, he suggested that further research into the uncertainties associated with air-breathing vehicles is necessary. Dissel, Kothari, and Lewis [5] also used the HySIDE code to investigate two-stage-to-orbit air-breathing configurations. They developed and analyzed six different systems which were constructed from three configurations such as a horizontal-takeoff/horizontal-landing (HTHL) hypersonic air-breathing booster with upper-stage reusable rocket, a HTHL turbojet booster with upper-stage hypersonic air-breather, and a vertical-takeoff/horizontal-landing (VTHL) rocket booster with upper-stage hypersonic air-breather. Overall, their results indicated several capable air-breathing vehicle system designs which warrant additional detailed analysis.

Spaceworks Engineering, Inc. and the Air Force Research Laboratory at Wright-Patterson Air Force Base designed and rigorously analyzed a two-stage-to-orbit launch vehicle that took advantage of air-breathing propulsion known as *Quicksat* [6]. Specifically, a strike mission, cargo delivery, and space-access configuration were considered. By using a plethora of complex design tools they were able to perform a preliminary concept analysis of the vehicle configurations. Trefny [7] explored the feasibility of a single-stage-to-orbit air-breathing vehicle, known as *Trailblazer*, by making use of OTIS. Olds [8] directed a team of undergraduate students at Georgia Tech in developing and analyzing a SSTO air-breathing hypersonic vehicle named *StarRunner*. Ultimately, by surveying these various documented missions, a fundamental realization of many different vehicle systems was obtained. Also, the data contained within these reports could be used to create similar representative aerospace vehicles for testing in the trajectory code.

2.3. ENTROPIC ANALYSIS OF AEROSPACE VEHICLES

Application of entropic analysis to aerospace vehicle systems can be traced back to a textbook by Oswatitsch [9] in which he appropriately included the effect of the second law on the wake process of a base aerodynamic shape. Previous work was also done by Foa [10] who discussed the quantification of losses as gains in entropy. Curran and Craig [11] described the concept of thrust or thrust-work potential (also called stream-thrust based methods) for the performance characterization of high-speed ramjet and scramjet engines. Riggins [12, 13] has performed additional work which stems directly from their previous efforts. The use of these methods has enabled the complete characterization of the loss in scramjet engine thrust due to irreversibility and has allowed the assessment of engine thrust losses in terms of irreversible loss mechanism and location.

In a closely related development, the general concept of work availability as applied to aerospace jet engines (turbojets and turbofans) has been developed and utilized by Roth [14, 15] who has suggested the use of work availability as a 'common currency' for engine design, evaluation, and optimization, generally without explicit consideration of entropy (second-law considerations) necessary. In addition, a significant amount of work has also been done in the area of applying conventional exergy (or availability) to the problem of aerospace vehicle design and evaluation (see, for example Clarke and Horlock [16] and Czysz and Murthy [17]). Availability is based on the assessment of the maximum reversible work as measured from a dead state and is attractive as a 'single currency' candidate; i.e. it is well-established and has an excellent track record for cyclic ground-based systems such as power plants. However, Riggins [18, 19] has shown problems with conventional availability when directly applied to very simple jet engine optimization problems and has suggested a modification of exergy (called engine-based exergy) which essentially unifies it with the stream thrust concepts discussed above. Moorhouse [20] articulated the need and vision for a 'common currency' which applies to all sub-systems of an aerospace vehicle and can be used in design, analysis, and optimization; this work provides a solid technical demonstration of that 'common currency'. Additionally, Moorhouse [21] examined incorporating the concept of entropy and thermo-economics into high-speed vehicle evaluation using availability techniques.

3. THEORY: SECOND-LAW/ENTROPIC ANALYSIS

This section develops the underlying theory and analytical methodologies implemented in the formulation and quantification of the second law performance of an aerospace vehicle in atmospheric flight. Next, the associated vehicle mass fraction analysis methodology (in terms of losses) is developed. A simplified single-stage-to-orbit (SSTO) example is then presented to outline and exemplify the completely general concepts contained within this chapter. This methodology is shown to provide the minimum thermodynamically permissible propellant mass fraction for any system.

3.1. COMBINATION OF SECOND-LAW ANALYSIS AND VEHICLE EQUATIONS OF MOTION

Consider the conventional forces intrinsic to an aerospace vehicle in flight in the atmosphere as shown in Figure 3.1.

Figure 3.1. Vehicle Configuration for Equations of Motion Analysis

For simplicity, let the thrust be aligned with the flight direction (i.e. the relative wind which is here designated as along the local x axis) such that $\alpha_\tau = 0$. However, the final results can be shown to be valid for any arbitrary α_τ. The instantaneous equation of motion in the flight direction for the vehicle is as follows:

$$T - D - m_{veh} g \cos\theta = m_{veh} \frac{dV}{dt} \tag{1}$$

The overall net fluid dynamic force experienced by the vehicle in the flight direction is F_x where

$$F_x = T - D \tag{2}$$

Hence,

$$F_x = m_{veh} g \cos\theta + m_{veh} \frac{dV}{dt} \tag{3}$$

Write the instantaneous power associated with the work done by F_x as:

$$F_x \bullet V = m_{veh} V \frac{dV}{dt} + m_{veh} V g \cos\theta \tag{4}$$

Now, write the force-entropy relationship for a nominally thermally balanced aerospace vehicle (see also [22]):

$$F_x \bullet V = \dot{m}_p \left[\frac{V^2}{2} + H_{prop} \right] - T_i \dot{S}_{irr(total)} \tag{5}$$

Here H_{prop} is the heating value of the fuel and $\dot{m}_p$ is the mass flow rate of propellant. $\dot{S}_{irr(total)}$ is the total entropy production rate associated with irreversibilities including the wake equilibrium process, i.e.

$$\dot{S}_{irr(total)} = \dot{S}_{irr(veh)} + \dot{S}_{wake} . \tag{6}$$

Equate expressions (4) and (5) to obtain:

$$\dot{m}_p\left[\frac{V^2}{2}+H_{prop}\right]-T_i\dot{S}_{irr(total)}=m_{veh}V\frac{dV}{dt}+m_{veh}Vg\cos\theta \tag{7}$$

Now $\dot{m}_p=-dm_{veh}/dt$ by definition so this expression can be written as

$$-\frac{dm_{veh}}{dt}\left[\frac{V^2}{2}+H_{prop}\right]-T_i\dot{S}_{irr(total)}=m_{veh}V\frac{dV}{dt}+m_{veh}Vg\cos\theta \tag{8}$$

Furthermore, $V\cos\theta=dh/dt$ where dh is the differential change in altitude of the vehicle and $\dot{S}_{irr(total)}=dS_{irr(total)}/dt$ where $dS_{irr(total)}$ is the entropy generation due to irreversibilities across both the vehicle control volume and the wake equilibration process during time dt. Therefore, the following is written:

$$-dm_{veh}\left[\frac{V^2}{2}+H_{prop}\right]-T_i dS_{irr(total)}=m_{veh}VdV+(m_{veh}g)dh \tag{9}$$

By definition, $VdV=d(V^2/2)$. Therefore

$$-T_i dS_{irr(total)}=dm_{veh}\left[\frac{V^2}{2}+H_{prop}\right]+m_{veh}d\left[\frac{V^2}{2}\right]+(m_{veh}g)dh \tag{10}$$

Now note that since the heating value of the propellant, H_{prop}, is fixed:

$$d\left[\frac{V^2}{2}\right]=d\left[\frac{V^2}{2}+H_{prop}\right]. \tag{11}$$

Consequently,

$$-T_i dS_{irr(total)} = dm_{veh}\left[\frac{V^2}{2} + H_{prop}\right] + m_{veh} d\left[\frac{V^2}{2} + H_{prop}\right] + (m_{veh} g)dh \tag{12}$$

This can be then written using the chain rule of differentiation as

$$-T_i dS_{irr(total)} = d\left[m_{veh}\frac{V^2}{2} + m_{veh} H_{prop}\right] + (m_{veh} g)dh \tag{13}$$

This can finally be rearranged and integrated across a mission as follows

$$H_{fuel}\Delta(\textit{propellant mass used}) - \int_{mission} T_i dS_{irr(total)} = \Delta(m_{veh}\frac{V^2}{2}) + \int_{mission} m_{veh} g dh \tag{14}$$

In this important balance equation:

$H_{prop}\Delta(\textit{propellant mass used})$ is the energy content associated with the expended propellant across the mission.

$\int_{mission} T_i dS_{irr(total)}$ (= $T_i \Delta S_{irr(total)}$ for assumption of constant T_i) is the cumulative lost work associated with all irreversibilities occurring during the mission including in the wake of the vehicle.

$\Delta(m_{veh}\frac{V^2}{2})$ is the change in vehicle kinetic energy across the mission.

$\int_{mission} m_{veh} g dh$ is the change in vehicle gravitational potential energy across the mission.

A generic sketch showing the fractions of overall on-board energy used in a mission is given in Figure 3.2.

Figure 3.2. Fractions of Overall On-board Energy Used in a Typical Mission

The overall quantity of energy used in the mission is shown to be discretely subdivided using equation 14 into three main categories corresponding to vehicle kinetic energy change, vehicle altitude change, and loss (irreversibility) recovery. This relationship is then used to define the following vehicle-based availability parameter which essentially provides the common currency (through the $S_{irr(total)}$) for vehicle loss assessment and optimization across a mission

$$Ex_{veh,mission} = H_{prop}\Delta(\textit{propellant mass used}) - \int_{mission} T_i dS_{irr(total)} \,. \tag{15}$$

A mission-based vehicle effectiveness is also directly defined as

$$\eta_{mission} = \frac{H_{prop}\Delta(\textit{propellant mass used}) - \int_{mission} T_i dS_{irr(total)}}{H_{prop}\Delta(\textit{propellant mass used})} \tag{16}$$

or

$$\eta_{mission} = 1 - \frac{\int_{mission} T_i dS_{irr(total)}}{H_{prop}\Delta(\textit{propellant mass used})} \tag{17}$$

One can also define an 'instantaneous' measure of second-law vehicle effectiveness by defining

$$\eta = 1 - \frac{T_i \dot{S}_{irr(total)}}{H_{prop} \dot{m}_p} \tag{18}$$

where $\dot{m}_p$ is the instantaneous mass flow rate of propellant expended by the vehicle. This implies that one always wants to minimize the entropy generated due to irreversibility per kilogram of propellant expended. Note that the limiting case of a vehicle in cruise simply implies that

$$(H_{prop} + \frac{V^2}{2})\Delta(\textit{propellant mass used}) = \int_{mission} T_i dS_{irr(total)} \tag{19}$$

Therefore, for a vehicle in cruise, minimum propellant usage simply corresponds to minimum overall (vehicle and wake) entropy production. For the limiting case of a vehicle in glide (with no propellant usage), the goal would also correspond to the minimization of overall (vehicle and wake) entropy production. In general, however, the parameter which should be maximized is

$$Ex_{veh,mission} = H_{propl}\Delta(\textit{propellant mass used}) - \int_{mission} T_i dS_{irr(total)} \ . \tag{20}$$

It is critical to realize that the entropy generation due to irreversibility term here must include the wake entropy generation and furthermore that the allocation or separation of losses due to irreversibilities must incorporate the coupling between vehicle irreversibilities and their impact on wake irreversibility. This must be done using the lost force (loss-stripping) methodology discussed in other references such as [23].

3.2. VEHICLE MASS FRACTION ANALYSIS

The following section examines the vehicle mass fraction characteristics which result from examination and analysis of equation 14. The vehicle (overall) propellant mass fraction is defined as follows:

$$\lambda_P = \frac{\Delta(\textit{ propellant mass used})}{m_{veh(i)}} \tag{21}$$

This quantity can be defined using (14) as follows:

$$\lambda_P = \frac{\Delta(m_{veh}\frac{V^2}{2})}{H_{prop}m_{veh(i)}} + \frac{\int_{mission} m_{veh}gdh}{H_{prop}m_{veh(i)}} + \frac{\int_{mission} T_i dS_{irr(total)}}{H_{prop}m_{veh(i)}} \tag{22}$$

or

$$\lambda_P = \lambda_{P,KE} + \lambda_{P,PE} + \lambda_{P,LOSS} \tag{23}$$

Here

$$\lambda_{P,KE} = \frac{\Delta(m_{veh}\frac{V^2}{2})}{H_{prop}m_{veh(i)}} \tag{24}$$

$$\lambda_{P,PE} = \frac{\int_{mission} m_{veh}gdh}{H_{prop}m_{veh(i)}} \tag{25}$$

$$\lambda_{P,LOSS} = \frac{\int_{mission} T_i dS_{irr(total)}}{H_{prop}m_{veh(i)}} \tag{26}$$

The first term on the right-hand side of this expression, $\lambda_{P,KE}$ represents the propellant mass fraction associated with the kinetic energy change of the vehicle across the mission. The second term on the right-hand side, $\lambda_{P,PE}$ represents the propellant mass

fraction associated with the potential energy change across the mission. The last term, $\lambda_{P,LOSS}$ represents the propellant mass fraction available and necessary for overcoming losses (irreversibilities) of all types during the mission. It is important to point out that the change in kinetic energy in equation 24 (and even the potential energy) can be negative over a time step or even an entire mission. To see this, consider the following example. If a vehicle is traveling at constant altitude and throttles the engines such that the mass of the vehicle remains unchanged, then the kinetic energy term in equation 22 will thus be negative due to the reduction in velocity. This situation assumes that the vehicle lifting surfaces are rotating in such a manner to maintain level flight. The left-hand side of equation 22 as well as the second term on the right-hand side will obviously be zero for this scenario. Therefore, the overall loss term will be positive and equal to the value of the first term in equation 22. The three terms on the right-hand side of equation 22 are quantified for specific missions and presented in a pie chart in the results section. This method of displaying the three quantities would not be useful for the constant altitude example previously described.

The altitude integral in equation 14 and subsequent development can be approximately modeled (assuming constant $\dot{m}_p$ and negligible change in g) as follows:

$$\int_{h_i}^{h_f} m_{veh} g dh = \left[m_{veh(i)} - \frac{\Delta(\textit{ propellant mass used})}{2} \right] g(h_f - h_i) \tag{27}$$

This integral is typically relatively small compared to the other terms in any event. It is particularly suited for many rockets (across the burn of a given stage); less suitable for air-breathing configurations. For illustration purposes, the modeling used in equation 27 is used for the rest of the analysis.

Using this approximation and defining an access-to-space mission in which V_i and h_i are both equal to zero, the following expression for the overall vehicle propellant mass fraction is written:

$$\lambda_P = \frac{(1-\lambda_P)\frac{V_f^{\,2}}{2} \; + \; gh_f \; + \; \frac{\int_{mission} T_i dS_{irr\,(total)}}{m_{veh(i)}}}{H_{prop} + \frac{1}{2} gh_f} \tag{28}$$

or

$$\lambda_P = \lambda_{P,KE} + \lambda_{P,PE} + \lambda_{P,LOSS} \tag{29}$$

where

$$\lambda_{P,KE} = \frac{(1-\lambda_P)\frac{V_f^{\,2}}{2}}{H_{prop} + \frac{1}{2} gh_f} \quad \text{and} \quad \lambda_{P,PE} = \frac{gh_f}{H_{prop} + \frac{1}{2} gh_f} \tag{30}$$

This indicates then that the vehicle propellant mass fraction can be subdivided into three contributions as demonstrated in (23) and (29): 1) propellant fraction necessary to effect the kinetic energy change across the mission, 2) propellant fraction necessary to effect altitude change from initial to final altitude (i.e. potential energy change across the mission) and 3) propellant fraction associated with (available for) overcoming all irreversibilities both in and over the aerospace vehicle and in the vehicle wake. Furthermore, the overall mass of the vehicle can be considered to be the sum of the propellant mass, the payload mass, and the structural/system mass (sometimes termed the 'dead weight' mass), i.e.

$$1 = \lambda_p + \lambda_d + \lambda_l = \lambda_p + \frac{structure\,/\,system\;mass}{m_{veh(i)}} + \frac{payload\;mass}{m_{veh(i)}} \tag{31}$$

3.3. VEHICLE PROPELLANT MASS FRACTION ANALYSIS (VELOCITY CHANGE ANALYSIS)

As an alternative formulation which can also be useful, equation 24 can also be easily rearranged to yield the following expression in which the subscripts i and f indicate initial and final conditions across the mission (or mission leg) of interest:

$$(H_{prop}+\frac{V_f^{\,2}}{2})\Delta(\textit{propellant mass used})=m_{veh(i)}(\frac{V_f^{\,2}-V_i^{2}}{2})+\int_{h_i}^{h_f} m_{veh}gdh+\int_{mission} T_i dS_{irr(total)} \tag{32}$$

Hence, for a vehicle which initiates from and returns to rest at the same altitude such that $V_i = V_f = 0$ and $h_i = h_f$, the overall propellant mass used during the mission is then

$$\Delta(\textit{propellant mass used})=\frac{\int_{h_i}^{h_i} m_{veh}gdh+\int_{mission} T_i ds_{irr(total)}}{H_{prop}} \tag{33}$$

In this relationship, the altitude integral remains because m_{veh} is continually decreasing across the mission.

The following alternative breakdown for the vehicle propellant mass fraction can then be written, again using the previously described potential energy term approximation:

$$\lambda_P = \frac{\dfrac{\frac{V_f^{\,2}}{2}-\frac{V_i^{2}}{2}}{H_{prop}+\frac{V_f^{\,2}}{2}}+\dfrac{g(h_f-h_i)}{H_{prop}+\frac{V_f^{\,2}}{2}}+\dfrac{\int_{mission} T_i ds_{irr(total)}}{m_{veh(i)}(H_{prop}+\frac{V_f^{\,2}}{2})}}{\left[1+\dfrac{g(h_f-h_i)}{2(H_{prop}+\frac{V_f^{\,2}}{2})}\right]} \tag{34}$$

This indicates then that the vehicle propellant mass fraction can be (if desired) subdivided into three contributions as demonstrated in (34): 1) propellant fraction necessary to effect a change in $V^2/2$, 2) scaled propellant fraction necessary for altitude change from initial to final altitude and 3) scaled propellant fraction associated with overcoming all irreversibilities both in and over the aerospace vehicle and in the vehicle wake. Note the inevitable 'rebalancing' of the propellant mass fractions here, although by definition the proportionality between potential and loss fractions will not change. This formulation can be of interest when analyzing raw velocity change increments.

3.4. EXAMPLE: SSTO COMPARISON BASED ON SECOND-LAW ANALYSIS

As a simple example, consider the take-off mass fraction breakdown for two SSTO vehicles, the first propelled by a H_2-O_2 rocket and the second by an air-breathing H_2-fueled propulsion system or systems. Let the required altitude change be 300,000 m at a required final velocity of 7900 m/s. The approximate heating value of stoichiometric H_2-O_2 combustion is taken as 1.34×10^7 J/kg (propellant) and the approximate heating value of H_2 in air is taken as 1.2×10^8 J/kg (fuel).

By utilizing equations 28 and 30, one can create a mass fraction breakdown. Figure 3.3 provides a plot of mass fraction 'breakdown' versus overall propellant mass fraction for the SSTO mission for a H_2-O_2 rocket as defined above. Specifically, the overall propellant mass fraction is the sum of the propellant mass fractions due to kinetic energy change, potential energy change, and losses. Note that (for instance) for an overall propellant mass fraction of 0.8, the dead weight and payload mass fractions as defined above would be 0.2, by definition. The most interesting result displayed on this figure is the distribution of the propellant mass fraction available to overcome losses denoted by the green line. Note that it rapidly approaches zero for decreasing overall propellant mass fraction such that there is no propellant mass fraction available for overcoming ANY losses for an overall propellant mass fraction less than approximately 0.75. In other words, any SSTO rocket configuration with smaller propellant mass fraction than 0.75 is thermodynamically impossible (representing a violation of the second law). Furthermore, a real rocket system will of course require a considerable fraction of its propellant for overcoming losses both internal and external (i.e. the second

law limit illustrated on this figure is the absolute limit, not the practical/feasible limit); it is thus not encouraging to note the rapidly constricting envelope of available payload/dead weight mass for realistic loss amounts.

Figure 3.3. Propellant Mass Fraction Breakdown for SSTO H_2-O_2 Rocket Mission

Figure 3.4 shows a similar breakdown for a SSTO H_2-air (air-breathing) mission. Note that there is much more propellant mass fraction available for overcoming losses as compared to the SSTO rocket, such that the range of thermodynamically possible configurations is much expanded over the rocket configuration (for the air-breathing SSTO vehicle there is no possible configuration for any propellant mass fraction less than approximately 0.25). Due to the expected higher irreversibility in general of air-breathing SSTO (due to long duration accelerating trans-atmospheric flight), it would of

course ‘need’ this margin and *perhaps* a good deal more! Note also that the modeling of the potential energy term used in the analysis is not very applicable to the air-breathing case, although these results are based on that crude modeling. (Again, however, the propellant mass fraction associated with altitude change is very small compared to the propellant mass fractions associated with kinetic energy change and losses.)

Figure 3.4. Propellant Mass Fraction Breakdown for SSTO H_2-air Air-breathing Mission

4. TRAJECTORY SOLVER CODING/METHODOLOGY

This section outlines the development of the vehicle trajectory code including model descriptions as well as the methodology implemented to interpret the results for a given mission.

4.1. DEVELOPMENT OF VEHICLE TRAJECTORY/MISSION ANALYSIS

In order to allow the equations of motion to be solved in a rapid fashion, several fundamental assumptions were made about the aerospace vehicle in motion through the atmosphere. The vehicle trajectory was assumed to be represented in a two-dimensional plane (i.e. no out of plane translation is allowed) and hence traveled along a two-dimensional flight path within that plane. In terms of action of all forces, the vehicle was assumed to be a point mass such that the forces lie through the vehicle center-of-gravity, i.e. no moments were considered which by extension means trim was not considered in the present study. In addition, the thrust and drag vectors were assumed to remain aligned at all times with the relative wind (free-stream velocity vector) as indicated in Figure 4.1. This means that the vehicle was continually thrusting in the direction of its instantaneous motion which aligns with its longitudinal axis. These assumptions are relatively straight-forward although restrictive but facilitate a broad study such as undertaken in this investigation.

Figure 4.1. Free-body Diagram of Two-dimensional Vehicle

Figure 4.1 shows the force diagram for the vehicle utilizing these assumptions under the constraint of a 'flat-Earth' model, i.e. where the y axis is aligned with the weight vector of the vehicle as shown. Note that the x′ and y′ axis are vehicle-fixed axes, i.e. the x′ axis is collinear with the vehicle longitudinal axis (and hence with the thrust, drag, and free-stream vector) whereas the y′ axis is perpendicular to the free-stream vector (and hence collinear with the lift vector). As the vehicle translates and changes orientation (angle) with respect to Earth-fixed axes (x and y), the vehicle-fixed axes are obviously variable with the Earth-fixed axes. It is very convenient to write the instantaneous equations of motion in the x′ and y′ directions for the vehicle as follows:

$$x' : T - D - m_v g \cos(\theta) = m_v \frac{dV_{x'}}{dt} \tag{35}$$

$$y' : m_v g \sin(\theta) - L = m_v \frac{dV_{y'}}{dt} \tag{36}$$

where T, D, and L represent the magnitudes of the thrust, drag and lift vectors respectively. The mass of the vehicle is designated as m_v and the standard gravitational acceleration is labeled as g. The vehicle body angle as shown in Figure 4.1 is θ. Solving equations 35 and 36 for $dV_{x'}$ and $dV_{y'}$, respectively, yields:

$$dV_{x'} = \left(T - D - m_v g \cos(\theta)\right)\frac{dt}{m_v} \tag{37}$$

$$dV_{y'} = \left(m_v g \sin(\theta) - L\right)\frac{dt}{m_v} \tag{38}$$

With the equations in this form and with appropriate models for instantaneous lift, drag, thrust and vehicle orientation, the differential change in vehicle velocity can thus be solved at an instant within the vehicle mission.

4.2. MODELING DESCRIPTIONS WITHIN VEHICLE TRAJECTORY CODE

The trajectory solver routine which was developed and used in this work was programmed utilizing the FORTRAN 90 language and can be found in Appendix A. In order to build this routine and apply it to the problems of interest in this investigation, it was necessary to build baseline models (initially simple) to calculate the lift, drag, and thrust forces on the vehicles of interest as well as to develop methodologies for advancing the vehicle trajectory across the entire atmosphere (ground launch to low Earth orbit). The following subsections describe details of the baseline and final models that were developed for trajectory prediction, Earth modeling, aerodynamics, propulsion, etc.

4.2.1. Trajectory Prediction and Earth Modeling. To determine the actual trajectory of the vehicle at each time step, one must develop a relation to calculate the change in vehicle velocity as well as the change in vehicle body angle, θ. To develop these relations it is helpful to consider Figure 4.2.

Figure 4.2. Differential Change in Velocity Vector Diagram

From Figure 4.2 one can develop the following equations to determine the magnitude of the new vehicle velocity and the change in vehicle body angle:

$$V_{new} = \sqrt{\left(V_{old} + dV_{x'}\right)^2 + dV_{y'}^2} \tag{39}$$

$$\theta_{new} = \theta_{old} + d\theta \quad where \quad d\theta = \arctan\left(\frac{dV_{y'}}{V_{old} + dV_{x'}}\right) \tag{40}$$

A round-Earth assumption was used in developing the Earth model as shown in Figure 4.3 in which the x-y frame is simply superimposed on the circular Earth as shown.

Figure 4.3. Round-Earth Diagram

This allows the development of a modified instantaneous equation of motion set for the vehicle in terms of the angle φ which denotes the angle the vehicle rotated about the center of Earth from the initial launch point. These modified equations are as follows:

$$dV_{x'} = \left(T - D - m_v g \cos(\theta - \varphi)\right)\frac{dt}{m_v} \tag{41}$$

$$dV_{y'} = \left(m_v g \sin(\theta - \phi) - L\right)\frac{dt}{m_v} \tag{42}$$

The horizontal and vertical positions at each time step can be determined from the following:

$$x_{new} = x_{old} + \left(\frac{V_{old} + V_{new}}{2}\right)\sin\left(\frac{\theta_{old} + \theta_{new} + 2\phi}{2}\right)dt \tag{43}$$

$$y_{new} = y_{old} + \left(\frac{V_{old} + V_{new}}{2}\right)\cos\left(\frac{\theta_{old} + \theta_{new} + 2\phi}{2}\right)dt \tag{44}$$

From observing Figure 4.3, the altitude can be determined by the following:

$$h = \sqrt{\left(R_e + y\right)^2 + x^2} - R_e \tag{45}$$

As altitude increases, the decrease in the local gravitational acceleration will also become important. To account for this effect, the following relationship was used:

$$g = g_0 \left(\frac{R_e}{R_e + h} \right)^2 \tag{46}$$

4.2.2. Atmospheric Model. The atmospheric properties at all points in the flight envelope (i.e. ambient pressure, temperature, and density) were modeled using standard techniques resulting in a typical Standard Atmosphere Model. This model defines the temperature as a function of altitude by separating the temperature versus altitude characteristics into 7 distinct regions as shown in Figure 4.4.

Figure 4.4. Standard Atmosphere Temperature Variation

The basic hydrostatic force equation was used to derive the following relationships for the pressure and density by region:

Constant Temperature: $$T(h) = T_{alt} \tag{47}$$

$$P(h) = P_{alt} e^{\left(\frac{-g_0}{R_{gas} T_{alt}}(h - h_{alt})\right)} \tag{48}$$

$$\rho(h) = \rho_{alt} e^{\left(\frac{-g_0}{R_{gas} T_{alt}}(h - h_{alt})\right)} \tag{49}$$

Gradient Temperature: $$T(h) = T_0 + a_0 \left(h - h_0\right) \tag{50}$$

$$P(h) = P_0 \left(\frac{T(h)}{T_0}\right)^{\left(\frac{-g_0}{a_0 R_{gas}}\right)} \tag{51}$$

$$\rho(h) = \rho_0 \left(\frac{T(h)}{T_0}\right)^{\left(\frac{-g_0}{a_0 R_{gas}} - 1\right)} \tag{52}$$

Utilizing these relationships, a consistent altitude model can then be readily built as part of the vehicle trajectory solver.

4.2.3. Vehicle Models. Three specific vehicle models were selected and then developed within the code in order to provide a way to compare the performance of different vehicle systems for a given mission. The initial vehicle system modeled was a basic single stage rocket. The next vehicle developed was the rocket/air-breathing combined cycle (RABCC). This model consists of three stages which utilized the rocket model for the initial and final leg and the air-breathing model for the second leg. And lastly, in an effort to allow for comparison between simulated data from the trajectory solver code and actual data from existent missions, the multi-stage rocket vehicle was constructed. It is important to note that the single/multi-stage rocket and the RABCC vehicles traversed different trajectories; however, this will be discussed later.

The vehicle configurations were distinguished by combining specific models for thrust, lift, drag, and trajectory development. Specific numerical information regarding vehicle characteristics is acquired from the user input deck. Additionally, for the multi-stage rocket, data detailing the individual stages are collected during runtime.

4.2.4. Thrust Models. The thrust for the single/multi-stage rockets was determined by specifying a mass flow rate of propellant $\dot{m}_{propellant}$ and an effective exhaust velocity *C* as shown below:

$$T_r = \dot{m}_{propellant} C \tag{53}$$

It is important to note that this relationship does not account for altitude effects on the nozzle (i.e. back-pressure effects). The thrust for the rocket/air-breathing combined cycle was calculated as shown below:

$$T_{ab} = \dot{m}_{air} \left[\sqrt{2 C_P T_{t4}} - V_\infty \right] \tag{54}$$

where $\dot{m}_{air}$, C_p, T_{t4}, and V_∞ represent the mass flow rate of air, specific heat at constant pressure, the maximum total temperature in the scramjet burner, and the free-stream velocity, respectively. A 'maximum possible' exit velocity condition is implied when thrust is calculated in this manner.

4.2.5. External Aerodynamics: Lift and Drag Models. Since the vehicle was modeled as a point-mass, when developing the equations of motion, lift and drag contributing devices must be specified in order to construct the lift and drag terms. The vehicle was modeled as consisting of a flat plate wing and a fuselage which contains the propulsion system as shown in Figure 4.5.

It is important to note that the wing is the only lifting surface on the vehicle whereas both the wing and the fuselage contribute drag. This also means that the wing is the only body angle control (other than gravity turns) for generating changes in vehicle orientation. However, to be discussed later, additional thrust vectoring can be programmed into the solver to provide another viable method of control. The three terms which define the vehicle characteristics are the planform area, S_w, the angle of attack, α, and the cross-sectional area of the fuselage, A_c.

Figure 4.5. Vehicle External Aerodynamic Contributions

The following highly approximate models for the coefficient of lift, C_{lw}, with varying Mach number are shown below:

$$M_\infty < 0.3 : C_{lw} = 0.11\alpha \frac{\pi}{180} = C_{lw,0} \tag{55}$$

$$0.3 < M_\infty < 0.7 : C_{lw} = \frac{C_{lw,0}}{\sqrt{1 - M_\infty^2}} \tag{56}$$

$$M_\infty = 1.0 : C_{lw} = 0.0 \tag{57}$$

$$M_\infty \geq 1.3 : C_{lw} = \frac{4\alpha}{\sqrt{M_\infty^2 - 1}} \tag{58}$$

Similar approximate models for the coefficient of drag due to the wing, C_{dw}, were used as shown below:

$$M_\infty < 0.3 : C_{dw} = 0.008\alpha \frac{\pi}{180} = C_{dw,0} \tag{59}$$

$$0.3 < M_\infty < 0.7 : C_{dw} = C_{dw,0} \tag{60}$$

$$M_\infty = 1.0 : C_{dw} = 0.2 \tag{61}$$

$$M_\infty \geq 1.3 : C_{dw} = \frac{4\alpha^2}{\sqrt{M_\infty^2 - 1}} \tag{62}$$

Again, the models for the drag coefficient are highly approximate and developed from linearized theory. Equations 58 and 62 represent the small-angle approximation equations that were used for supersonic and hypersonic flight regimes. The coefficient of lift and drag models due to the wing were linearly interpolated throughout the transonic Mach regime. The coefficient of body drag for the single/multi-stage rocket was calculated as shown below:

$$M_\infty < 4 : C_{db,r} = 0.34 \tag{63}$$

$$M_\infty \geq 4 : C_{db,r} = \frac{2}{\gamma M_\infty^2}\left[1 + \frac{2\gamma}{\gamma + 1}\left(M_\infty^2 \sin^2 \beta - 1\right)\right] \quad where\ \beta = \left(10.5 + \frac{274}{M_\infty^2}\right)\frac{\pi}{180} \tag{64}$$

Equation 64 is an approximation determined from modeling the pressure drag from oblique shocks present on a two-dimensional wedge shape, with a curve-fit for the shock angle, β, for varying Mach numbers. The coefficient of body drag for the rocket/air-breathing combined cycle was calculated in essentially the same fashion as for the single/multi-stage rocket. However, to account for the area of mass capture associated with the air-breathing engine embedded within the fuselage, an adjustable factor is added as shown below:

$$C_{db.ab} = F_{ab} C_{db,r} \quad where\ \ 0 \leq F_{ab} \leq 1.0 \tag{65}$$

Typically, this factor is less than 0.5 for current air-breathing configurations.

Additional aerodynamic modeling improvements were made for the supersonic flight regime. Specifically, oblique shock and expansion waves were included in the computation of lift and drag on the wing. Figure 4.6 displays a flat plate type wing in supersonic flow. In order to determine the lift and drag generated by the wing, one must determine the static pressure ratios of region 2 to region 1 and region 3 to region 1 (shown in Figure 4.6). The following procedure outlines the methodology used to compute the lift and drag for this scenario at each time step.

Figure 4.6. Supersonic Flow Over a Wing

First, the Prandtl-Meyer function is calculated for region 1 from the following well-known expression:

$$\nu_1 = \nu(M_1) = \sqrt{\frac{\gamma+1}{\gamma-1}}\tan^{-1}\left(\sqrt{\frac{\gamma-1}{\gamma+1}\left(M_1^2-1\right)}\right) - \tan^{-1}\left(\sqrt{\left(M_1^2-1\right)}\right) \tag{66}$$

where γ is the ratio of specific heats and M_1 is the Mach number from region 1. Then, the Prandtl-Meyer function for region 2 can be determined simply from:

$$\nu(M_2) = \alpha + \nu(M_1) \tag{67}$$

Now, with the value of the Prandtl-Meyer function obtained from equation 67, one can inversely solve equation 66 to acquire, M_2, the Mach number from region 2. The static pressure ratio between regions 2 and 1 can be calculated from the following relationships between total and static conditions:

$$\frac{P_2}{P_1} = \frac{P_2}{P_{O2}}\frac{P_{O2}}{P_{O1}}\frac{P_{O1}}{P_1} = \left(1+\frac{\gamma-1}{2}M_2^2\right)^{-\frac{\gamma}{\gamma-1}}(1)\left(1+\frac{\gamma-1}{2}M_1^2\right)^{\frac{\gamma}{\gamma-1}} \tag{68}$$

Since the expansion fan is an isentropic process, the stagnation pressures from region 1 to region 2 remain constant and hence their ratio is unity. Next, to determine the properties across a shockwave, one must first determine the shockwave angle β. This can be computed from the following well-known relation:

$$\tan(\alpha) = 2\cot(\beta)\left[\frac{M_1^2 \sin^2(\beta) - 1}{M_1^2(\gamma + \cos(\beta)) + 2}\right] \tag{69}$$

Solving equation 69 for β usually involves interpreting a contour plot. Therefore, an iterative method was used to determine the shock angle for a given Mach number and flow deflection angle. The equivalent normal Mach number for region 1 corresponding to the shock angle β can be computed from:

$$M_{n,1} = M_1 \sin(\beta) \tag{70}$$

Consequently, the static pressure ratio between region 3 and region 1 can now be calculated from the following normal shock relation:

$$\frac{P_3}{P_1} = 1 + \frac{2\gamma}{\gamma + 1}\left(M_{n,1}^2 - 1\right) \tag{71}$$

Finally, the lift and drag can be determined simply from:

$$L = (P_3 - P_2) S_w \cos(\alpha) \tag{72}$$

$$D = (P_3 - P_2) S_w \sin(\alpha) \tag{73}$$

And hence, the coefficients of lift and drag for the wing can be calculated from:

$$C_{L_w} = 2\left(\frac{P_3}{P_1} - \frac{P_2}{P_1}\right)\frac{\cos(\alpha)}{\gamma M_1^2} \tag{74}$$

$$C_{D_w} = 2\left(\frac{P_3}{P_1} - \frac{P_2}{P_1}\right)\frac{\sin(\alpha)}{\gamma M_1^2} \tag{75}$$

These wing contributions were added to the overall lift and drag calculated on the vehicle in supersonic flow across each time step.

Obviously, the Prandtl-Meyer function is a transcendental expression and thus cannot be solved analytically for the Mach number. This calculation is traditionally performed as a table lookup or by iteratively solving equation 66. Initially, an iterative method was programmed; however, this resulted in large computation times. An attempt was made to research various inverse solutions of the Prandtl-Meyer function [24]. However, the errors associated with the approximations presented in [24] were deemed unacceptable. Instead, the author constructed piece-wise exponential and logarithmic curve-fits for the inverse solution of the Prandtl-Meyer function. The errors associated with these curve-fits were less than 1%. The corresponding time of computation was greatly expedited by having an approximate inverse solution for the Mach number.

Lastly, an effort was made to monitor the shock detachment angle for a given Mach number and angle of attack. If shock detachment is repeatedly detected, useful suggestions are output to the user such as increasing the planform area of the wing. The analytical procedure used to implement this check was outlined in a separate document attached in Appendix B.

4.2.6. Wing Angle Control Routine. Developing the method of vehicle body angle control utilizing this level of modeling (as well as the limitations of reasonable performance using the model of the vehicle wing) requires great attention to the definition of physical limitations on vehicle orientation, operation, run-time execution changes and error handling. As an example of the user inputs which have been implemented, maximum and minimum vehicle body angle limitations must be specified by the user. If the maximum or minimum vehicle body angles are exceeded during a simulation, the wing angle is automatically adjusted accordingly such that the vehicle will maintain proper orientation. Another important input is the maximum allowed wing angle of attack. If this limit is exceeded, the code aborts and helpful suggestions are made to the user concerning various inputs that can be changed to reduce the required

angle of attack in order to complete the mission (i.e. increase wing area). Also, since small-angle approximations are used in calculating expressions for the lift and drag, care must be taken in selecting a reasonable wing angle deflection limit.

Transitioning between the rocket-powered and air-breathing legs for the rocket/air-breathing combined cycle configuration warranted a need for a progressive adjustment of the wing angle as opposed to an instantaneous correction which requires the vehicle to experience large accelerations. Figure 4.7 depicts the instantaneous wing angle correction method. As can be seen, this method incurred generally unacceptable large accelerations on the vehicle making it (at the very least) an impractical option for human passengers. To account for this, Figure 4.8 shows the gradual wing angle correction which, in fact, significantly reduced the accelerations required.

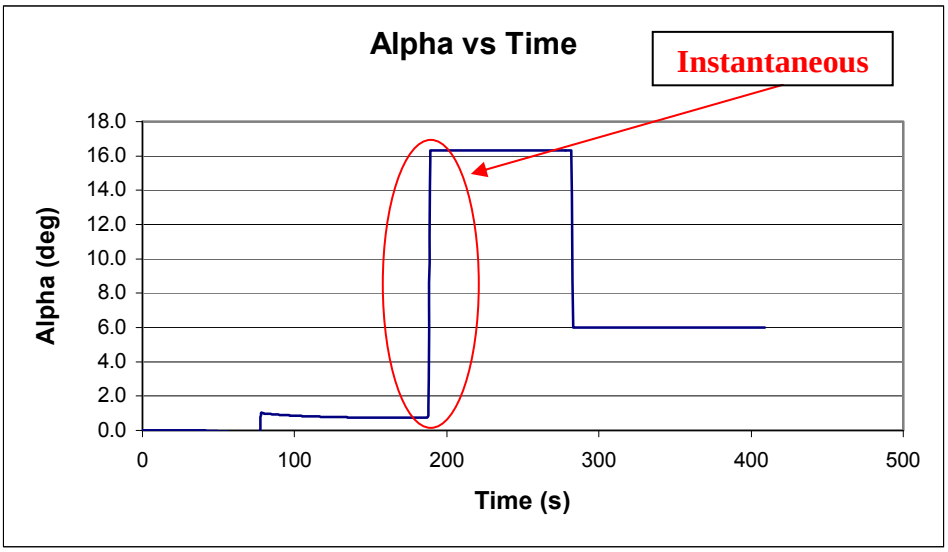


Figure 4.7. Instantaneous Wing Angle Correction

One important observation must be made regarding the control of the vehicle body angle. When the vehicle escapes the modeled atmosphere (i.e. altitude greater than 105 km), control by wing angle is impossible since lift and drag can no longer be generated. Therefore, if additional control inside or outside of the modeled atmosphere is necessary, then one can utilize thrust vectoring. This method enables gimbaling of the thrusting nozzle and hence impacts the vehicle body angle.

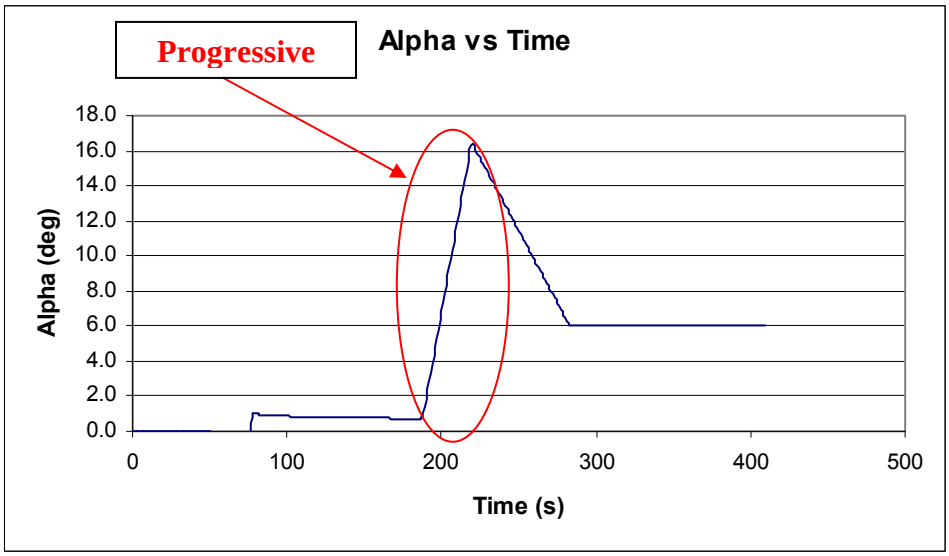


Figure 4.8. Progressive Wing Angle Adjustment

This specific capability was not programmed into the solver in a general sense; however, the implementation is straightforward. In fact, thrust vectoring was indeed programmed in for a specific mission to be discussed later.

4.2.7. Energy Utilization Calculations. As discussed earlier, the governing balance in terms of energy/power utilization across a vehicle mission which results when the equations of motion and the second law are combined as follows:

$$H_{fuel}\Delta\left(propellant\ mass\right) - \int_{mission} T_i dS_{irr(total)} = \Delta\left(m_v \frac{V^2}{2}\right) + \int_{mission} m_v g dh \qquad (76)$$

The first term in equation 76 represents the energy content associated with the expended propellant across the mission. The second term represents the cumulative lost work associated with all irreversibilities throughout the mission. The last two terms account for the change in vehicle kinetic and potential energy, respectively. The cumulative lost work term can be calculated at each differential time step and then summed across the entire mission.

4.2.8. Integration along a Vehicle Path (Trajectory Determination). Each of the vehicle configurations considered traverse distinct trajectories throughout the atmosphere to achieve space access. The primary objective of each vehicle is to achieve

a low-Earth orbit with the maximum possible final vehicle mass. Obviously, this corresponds to a maximum of final payload mass into orbit. The distinct trajectories are generated from solving the equations of motion for a given set of initial conditions as well as specified vehicle parameters such as wing area, total vehicle mass and cross-sectional area, etc. The simulation of a single/multi-stage rocket-powered vehicle consists of beginning at a specified launch angle and firing until the fuel is depleted or a burnout altitude is achieved. When this altitude is attained the rocket-powered vehicle essentially switches off and enters a coasting phase until the vehicle reaches a specified altitude corresponding to low-Earth orbit. Obviously, for a multi-stage configuration, several stage transitions can take place before the mission is complete. Finally, an impulse is executed producing the required delta V to place the vehicle in a circular low-Earth orbit.

Similarly, the rocket/air-breathing vehicle simulation initialized the vehicle at a starting velocity. First, a rocket was fired until the air-breathing acceleration altitude was achieved. At this stage in the trajectory, the rocket was disengaged and the vehicle coasted until it had sufficiently nosed over where it then commenced the air-breathing propulsion system. The vehicle was accelerated at a nearly constant altitude to a specified Mach number. In order to maintain altitude while accelerating, the necessary wing angle to balance lift and weight was calculated at each time step during the mission leg. Once the transition Mach number was attained, the air-breathing system ceased operation and the final rocket ignited. During this transition, the vehicle rotated upward in an orientation conducive to gaining altitude. The final rocket was fired until a precise burnout altitude was attained by which the vehicle coasted to an altitude in low-Earth orbit. Again, a final impulse was fired to set the vehicle into a circular orbit.

After several simulations were conducted, an additional trajectory was developed for the RABCC vehicle. This new trajectory would allow the vehicle to simultaneously climb and accelerate. A structural mass drop corresponding to the superfluous first stage components of the RABCC was also incorporated. This loss of mass improved overall vehicle performance for a given mission. It is important to note that a constant dynamic pressure trajectory is usually implemented for optimization of air-breathing vehicle systems. This method ensures minimal heating since the vehicle remains in the

atmosphere for a long duration. However, since complete trajectory optimization was not the main objective in this study, that trajectory algorithm was not coded.

Generally, for a given mission, several iterations on the chosen initial conditions were performed in order to attempt to locate the optimal trajectory for maximizing the final mass into orbit. This involved modifying parameters such as the launch angle, burnout altitude, etc., in such a way that the resulting trajectory provided space access with the final payload mass maximized. For example, an increase in launch angle significantly impacts the orientation of the vehicle and hence the vertical component of the velocity will be larger. It is important to note that these were the only efforts made to "optimize" the trajectory (i.e. no specific trajectory optimizers were implemented).

In order to compute the final vehicle masses, the propellant required to position the vehicle in orbit must be determined. Figure 4.9 shows a schematic of how the magnitude of the delta V is calculated.

Figure 4.9. Delta V Diagram

From observing Figure 4.9, the following equations can be written:

$$\Delta\vec{V} = \vec{V}_{circular} - \vec{V} \tag{77}$$

$$\left|\Delta\vec{V}\right| = \sqrt{\vec{V}_{circular}\cdot\vec{V}_{circular} + \vec{V}\cdot\vec{V} - 2\left|\vec{V}_{circular}\right|\left|\vec{V}\right|\cos\lambda} \tag{78}$$

where λ is the angle between the current and circular velocity. Once the magnitude of the delta V is computed from equation 78, the following well-known relationship can be used to determine the required propellant mass:

$$m_{propellant} = m_0 \left(1 - e^{-\frac{|\Delta \vec{V}|}{C}} \right) \tag{79}$$

4.2.9. Interpreting Mission Data. After a simulation was successful, a large data file containing mission critical information about the time histories and performance of the vehicle was created. Since many missions were simulated, a quick and efficient way to visualize the enormous amount of data was essential. Therefore, a formatted Excel spreadsheet was created in which the data file could be imported. Once the data was imported, various plots were generated providing instant visualization. Figure 4.10 shows a screenshot of this template.

Having this ability greatly assisted in understanding a simulated mission thoroughly and also in the development of the code. The procedure involving operation of the trajectory code as well as importing the data for visualization is described in the document attached in Appendix C.

Figure 4.10. Formatted Excel Template Screenshot

5. RESULTS

A mission corresponding to the historic Apollo 11 flight to the Moon was initially simulated to serve as a validation of the trajectory solver. This particular case was chosen because of the availability of mission data. Next, a comparison mission between the common Delta II multistage rocket and a hypersonic air-breathing/rocket combined cycle vehicle was performed. These mission analyses provide for the first time a detailed examination from a second law perspective of the performance and losses incurred by such aerospace vehicle systems.

5.1. APOLLO 11 VALIDATION MISSION

The Apollo 11 mission required the launching of a giant three-stage rocket known as the Saturn V which can be seen in Figure 5.1.

Figure 5.1. Saturn V Launch Vehicle

5.1.1. Mission Events and Parameters. The first stage of the Saturn V rocket utilized five Rocketdyne F-1 engines burning RP-1/LOX propellant and producing a thrust of approximately 35 MN. The second stage contained 5 Rocketdyne J-2 engines burning LH_2/LOX propellant and generating a thrust of 4.45 MN. The first and second stage inter-stage unit masses were 4583 kg and 3665 kg, respectively. The third stage included 1 Rocketdyne J-2 engine which also burnt LH_2/LOX propellant and produced a thrust of 890000 N. The instrument unit mass located above the third stage was 1953 kg. Table 5.1 summarizes the rocket stage parameters obtained from [25].

Table 5.1. Saturn V Rocket Stage Parameters

Parameter	Stage 1	Stage 2	Stage 3 (initial)
Fueled Mass (kg)	2278247	480432	118171
Empty Mass (kg)	130975	36250	11340
Propellant	RP-1/LOX	LH_2/LOX	LH_2/LOX
Isp (s)	303	453	453
Time of Burn (s)	170	395	165
Burnout Altitude (m)	61000	184000	185000
Burnout Speed (m/s)	2682	6838	6838
Thrust (MN)	35.155	4.45	0.89

The first and second stages of the rocket would burn until depletion at which point the emptied stage mass separated from the accelerating vehicle. The third stage was fired once to insert the spacecraft into a circular parking orbit allowing two orbits of Earth and then again to inject the vehicle on a trans-lunar trajectory. Since this mission represented a validation case for the developed trajectory solver, only the section containing the initial launch through the insertion to a circular parking orbit was of interest. Table 5.2 summarizes the Apollo 11 mission events of interest acquired from [26]. The velocity shown in Table 5.2 corresponds to an inertial velocity which incorporated the additional component of velocity due to the rotation of the Earth. The value of the component added can be approximated by simply calculating the product of the linear velocity of the surface of the Earth at the equator and the cosine of the latitude of the launch location

(i.e. 28.57° for Cape Canaveral). It is important to note that this component was only implemented as a uniform addition to every data point throughout this mission and did not at all impact the vehicle's velocity modeling throughout the atmosphere nor the modeling and simulation of the trajectory.

Table 5.2. Apollo 11 Mission Events

Event	Time (s)	Altitude (m)	Range (m)	Velocity (m/s)
First Motion	0	56	0	409
Maximum Dynamic Pressure	81	13218	5000	804
S-IC Center Engine Cutoff	135	44379	46115	1983
S-IC Outboard Engines Cutoff	160.8	66341	91859	2753
S-IC/S-II Separation	161.6	67051	92970	2763
S-II Ignition	163.2	67629	95007	2761
S-II Aft Inter-stage Jettison	191.5	91826	161124	2884
LET Jettison	197.2	96012	174645	2980
S-II Center Engine Cutoff	459.8	179269	1111200	5719
S-II Outboard Engines Cutoff	551.4	185855	1639020	6933
S-II/S-IVB Separation	552.3	185923	1644391	6936
S-IVB Ignition	555.4	185932	1645354	6936
S-IVB First Cutoff	700.1	188353	2639470	7791
Parking Orbit Insertion	710.1	188285	2711143	7793

5.1.2. Mission Testing. After acquiring all of the necessary data from the above sources, the corresponding input deck was created. Initially, the simulation was executed with a specified launch angle perpendicular to the ground. Since no external forces affecting the tipping of the rocket were present, this first test was equivalent to a sounding rocket trajectory not representative of the actual mission. In order to obtain a more accurate trajectory with appropriate vehicle tipping, the initial launch angle was slightly modified. However, this adjustment could not alone ensure a better trajectory. A need for effective control of the rocket was also present. As previously mentioned in the vehicle modeling section, modification of the wing angle of attack was the only method

of vehicle control. Preliminary results suggested that, in addition to tipping, further control be implemented at an altitude above 105000 m. Unfortunately, control by wing angle was not possible since the target altitude was above the edge of the modeled atmosphere (i.e. lift and drag were negligible). The only other viable method of control realized was thrust vectoring of the rocket nozzle. A simple control scheme was programmed in which the vehicle targeted an ideal altitude. If this altitude was above or below the vehicle's current altitude, then the thrusting angle would be decreased or increased, respectively. Furthermore, a thrust vectored accelerated climb was also implemented at the end of the mission. The final refined input deck as well as additional mission plots can be seen in Appendix D.

5.1.3. Mission Results. Various charts generated from the simulated data were overlaid with the actual reported Apollo 11 mission data to provide a visual correlation between the data. The trajectory for this mission can be seen in Figure 5.2.

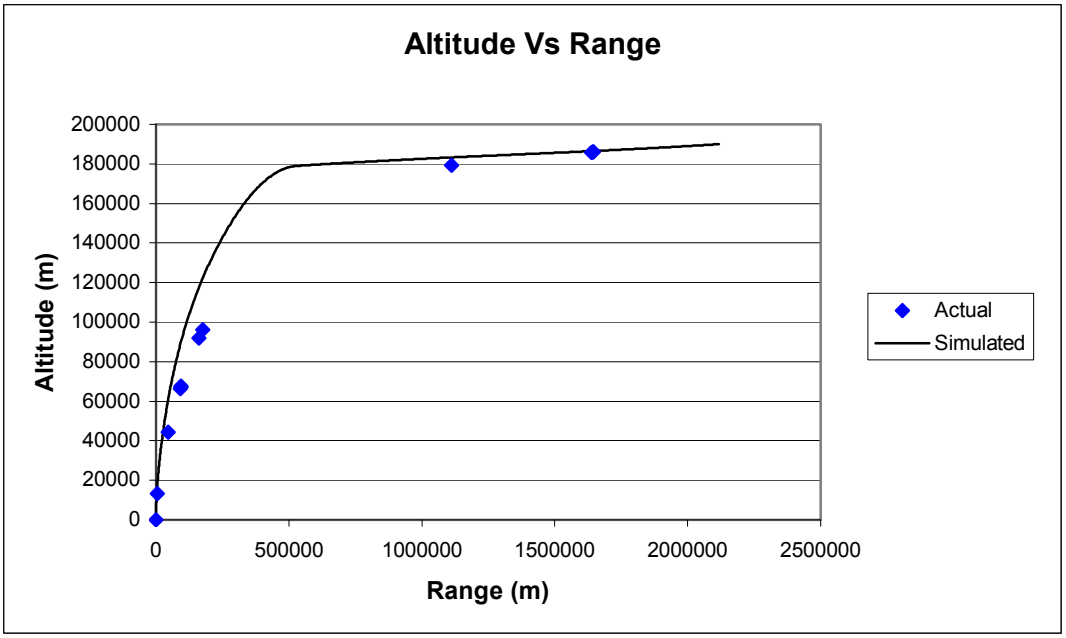


Figure 5.2. Actual and Simulated Apollo 11 Mission Trajectory

This steep trajectory is very typical among multistage rockets. The idea behind this style of trajectory was to spend the least amount of time possible in the atmosphere to

minimize the drag losses and heating of the vehicle. A comparison of the actual and simulated altitude time histories can be seen in Figure 5.3.

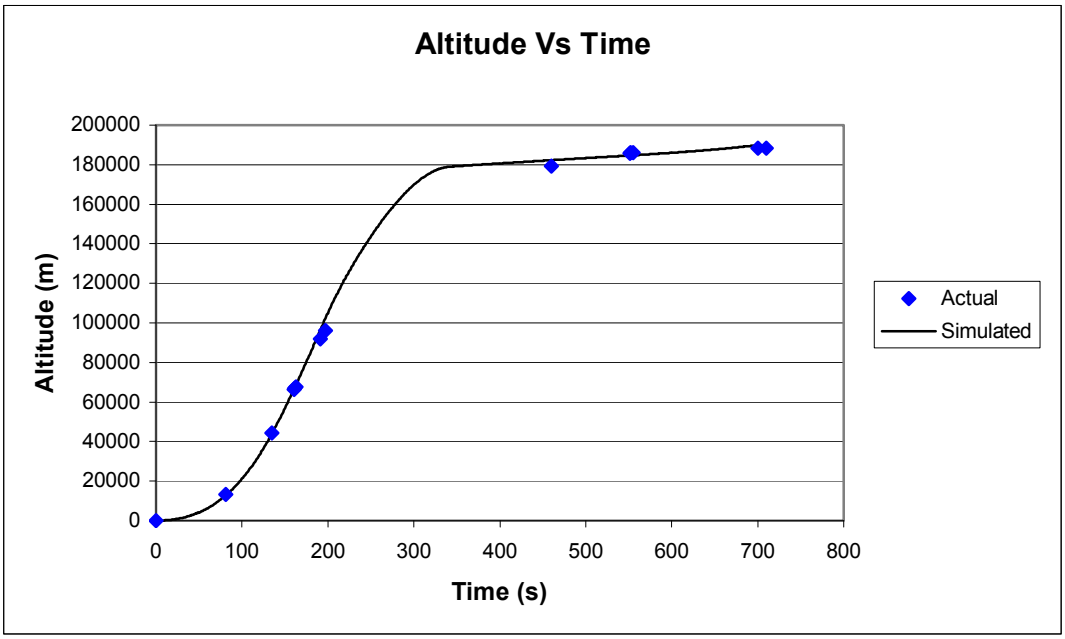


Figure 5.3. Actual and Simulated Apollo 11 Mission Altitude Time History

As one can see from observing Figure 5.3, the simulated data correspond very well with the actual mission data. Likewise, Figure 5.4 shows the similarity between the velocity time history data. From observing Figure 5.4, one can notice that the transitioning between the first two stages occurred slightly later in the simulated data. The vertical spike observed at the end of the mission corresponds to the change in velocity required to maintain a circular orbit of 189000 m. Furthermore, a series of charts containing second law performance information was generated. Figure 5.5 displays a time history of the on-board energy usage for the mission.

The incremental and accumulated changes in kinetic energy, potential energy, and losses due to irreversibilities can be seen for the entire mission. At any given time during the mission, the sum of these three values should be equivalent to the energy content of the spent fuel. Figure 5.5 basically represents a visual depiction of each term in equation 14 for the simulated mission at every time step. Figure 5.6 shows a sectional breakdown

of the on-board energy usage. Note that, nearly 70 percent of the energy content associated with the spent propellant can be attributed to losses due to irreversibilities.

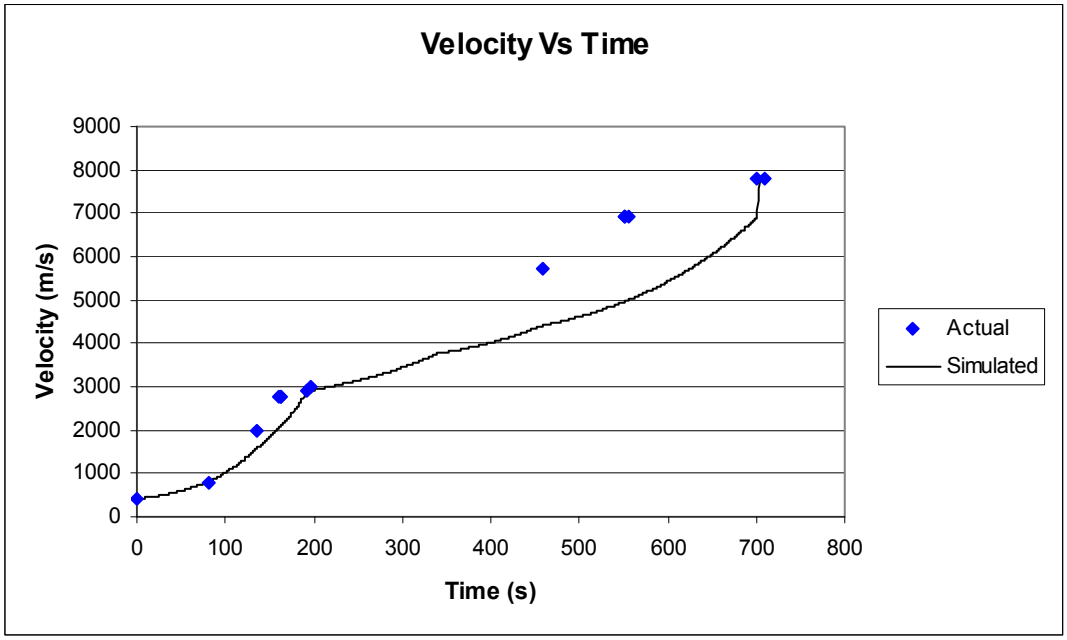


Figure 5.4. Actual and Simulated Apollo 11 Mission Velocity Time History

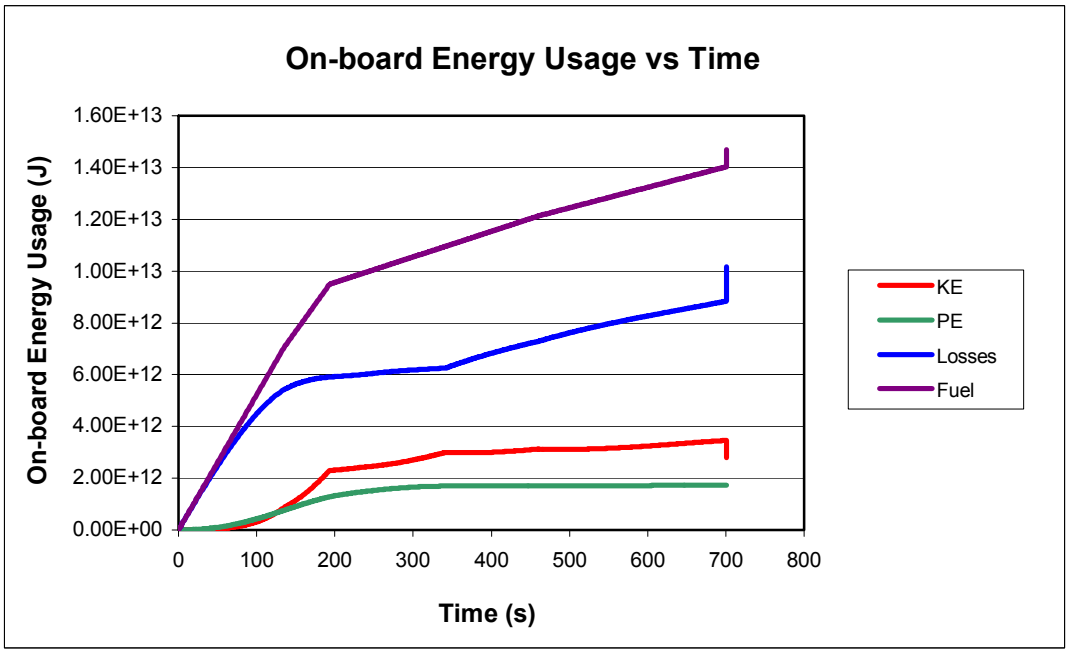


Figure 5.5. Apollo 11 On-board Energy Usage Time History

It is important to note that the second law analysis described here conceptually allows one to obtain the detailed breakdown of the losses in terms of loss mechanism, location, and time of loss with the level of detail depending upon the level of modeling and simulation. Here, as this represents the first application of this powerful methodology, the breakdown is relatively simple; however, it can be applied with detailed results to complex analyses involving CFD, etc.

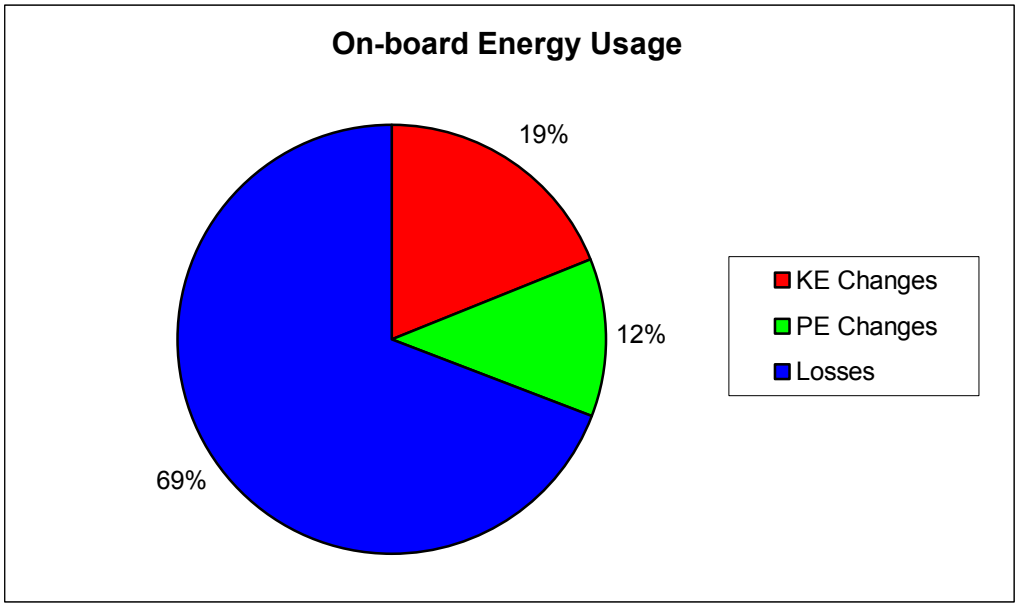


Figure 5.6. Apollo 11 On-board Energy Usage

Figure 5.7 displays the complete various vehicle mass fractions for this case, which of course show that the propellant expended is directly proportional to the on-board energy utilization. Six main categories were chosen to describe the mass fractions shown in this figure. Lambda P (KE) represents the amount of propellant needed to achieve the change in kinetic energy for the mission as a fraction of the total vehicle mass. Likewise, Lambda P (PE) and Lambda P (Losses) indicate the propellant fraction necessary to overcome the change in potential energy and attributed to losses due to irreversibilities, respectively. Lambda UP, D, and PL signify the mass fraction associated with the unused propellant, structural components, and payload, correspondingly. Figure 5.8 displays the vehicle mass fractions determined from the alternative "velocity change

analysis" method. It is important to point out that the three propellant mass fractions are scaled as mentioned previously.

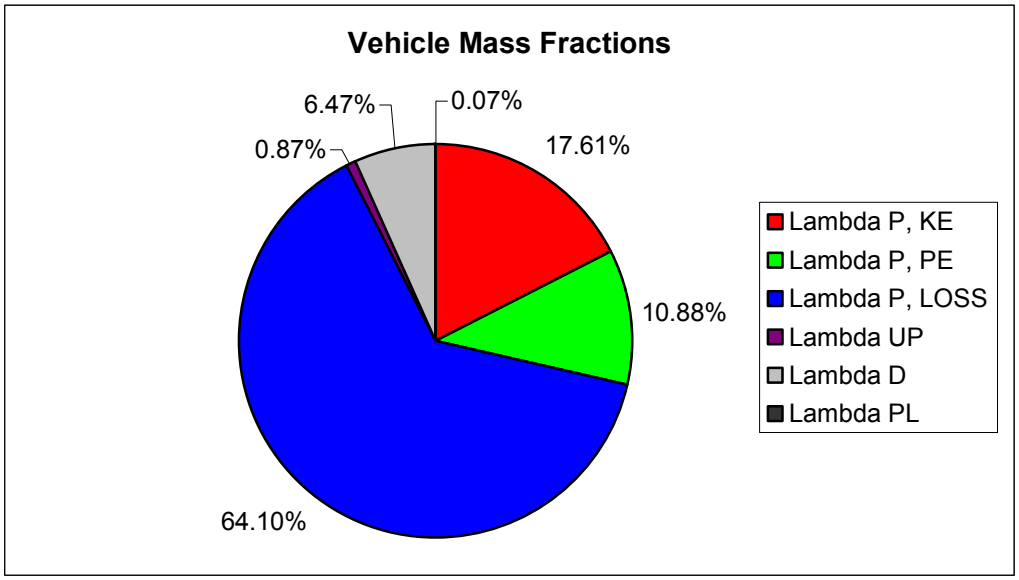


Figure 5.7. Apollo 11 Vehicle Mass Fractions

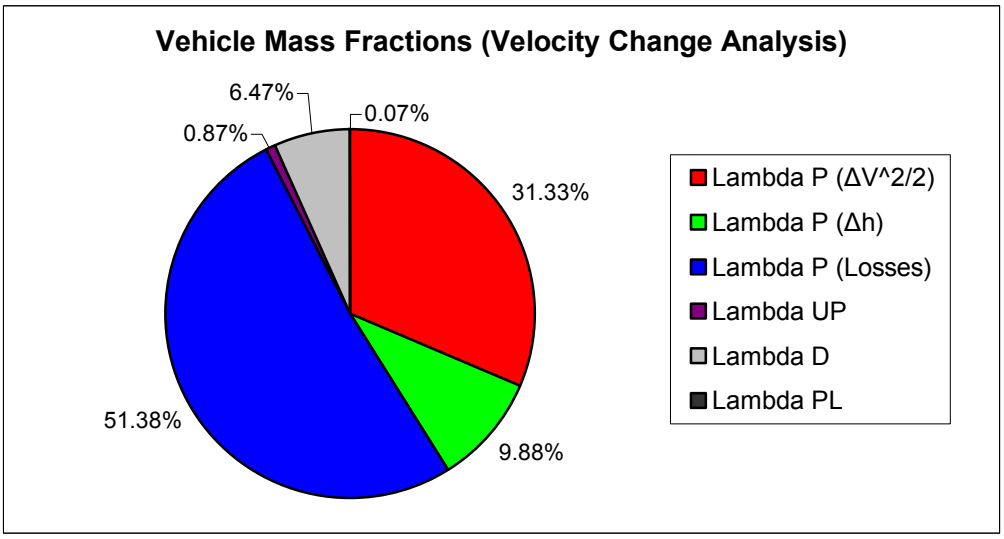


Figure 5.8. Apollo 11 Vehicle Mass Fractions (Velocity Change Analysis)

The lost work due to irreversibilities computed by the trajectory solver for each stage of the mission can be compared to similar analytical calculations. As mentioned

previously, the trajectory solver determines the total lost work by differentially summing the quantities in the following rearrangement of equation 14.

$$\int_{mission} T_i dS_{irr} = (H_{prop} + \frac{V_f^{\,2}}{2})\Delta(\textit{propellant mass used}) - m_{veh(i)}(\frac{V_f^{\,2} - V_i^2}{2}) - \int_{h_i}^{h_f} m_{veh} g dh \quad (80)$$

As explained earlier, by assuming negligible changes in the gravitational acceleration constant and a constant mass flow rate of propellant, the altitude integral in equation 80 can be approximated resulting in the following expression.

$$\begin{aligned}\int_{mission} T_i dS_{irr(total)} = (H_{prop} + \frac{V_f^{\,2}}{2})\Delta(\textit{propellant mass used}) - m_{veh(i)}(\frac{V_f^{\,2} - V_i^2}{2}) \\ - \left[m_{veh(i)} - \frac{\Delta(\textit{propellant mass used})}{2} \right] g(h_f - h_i)\end{aligned} \quad (81)$$

Differences in the values of data reported from references [25] and [26] were noticed, hence, two comparisons will be made to demonstrate the sensitivities associated with the lost work calculations. Tables 5.3 and 5.4 contain the relevant mission stage data corresponding to references [25] and [26], respectively, whereas, Table 5.5 contains a summary of the simulated data.

Table 5.3. Apollo 11 Mission Stage Data from Reference [25]

	Stage 1	Stage 2	Stage 3	Total
H_{prop} (J/kg)	4417670.1	9874256.9	9874256.9	-
V_i (m/s)	0	2681.6667	6838.0556	-
V_f (m/s)	2681.6667	6838.0556	6838.0556	-
$m_{used\ prop}$ (kg)	2147272	444182	33045.078	-
$m_{veh(i)}$ (kg)	2887051	604221	120124	-
h_i (m)	0	61000	184000	-
h_f (m)	61000	184000	185000	-
Lost work (J)	5.741E+12	2.356E+12	1.098E+12	9.194E+12

Table 5.4. Apollo 11 Mission Stage Data from Reference [26]

	Stage 1	Stage 2	Stage 3	Total
H_{prop} (J/kg)	4417670.1	9874256.9	9874256.9	-
V_i (m/s)	0	2761.21	6936.24	-
V_f (m/s)	2761.21	6936.24	7791.42	-
$m_{used\ prop}$ (kg)	2147272	444182	33045.078	-
$m_{veh(i)}$ (kg)	2887051	604221	120124	-
h_i (m)	0	67629	185932	-
h_f (m)	67629	185932	188353	-
Lost work (J)	5.463E+12	2.396E+12	5.704E+11	8.429E+12

Figures 5.9 and 5.10 were constructed from the parameters shown in the above tables. By examining Figure 5.9, one can see that the lost work calculations correspond very well for each of the stages of the mission. When viewing Figure 5.10, one can observe a larger difference in irreversibilities computed for stage 3. The press kit data [26] predicted a value of approximately one half the value determined by the trajectory solver. This occurrence is a result of the large change in velocity of stage 3 as shown numerically in Table 5.4.

Table 5.5. Apollo 11 Simulated Mission Stage Data

	Stage 1	Stage 2	Stage 3	Total
H_{prop} (J/kg)	4417670.1	9874256.9	9874256.9	-
V_i (m/s)	0	2894.61	6501.71	-
V_f (m/s)	2894.61	6501.71	7786.2612	-
$m_{used\ prop}$ (kg)	2145517	459504.78	65634.131	-
$m_{veh(i)}$ (kg)	2887051	603367.46	143862.68	-
h_i (m)	0	98176.71	189994.39	-
h_f (m)	98176.71	189994.39	190000.27	-
Lost work (J)	5.907E+12	2.937E+12	1.318E+12	1.016E+13

Figure 5.9. Apollo 11 Comparison of Lost Work between Reference [25] and Simulated Data

It is important to note that a large change in kinetic energy also occurs in the simulated data; however, the change in potential energy was rather small since the third stage corresponds to the nearly instantaneous orbital positioning maneuver.

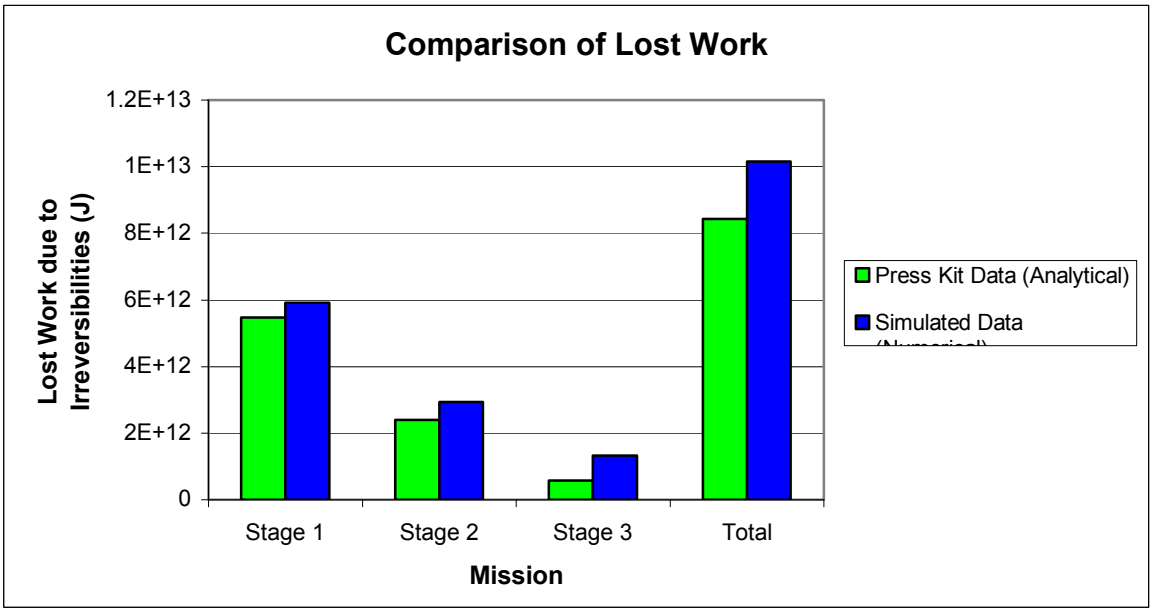


Figure 5.10. Apollo 11 Comparison of Lost Work between Reference [26] and Simulated Data

5.2. DELTA II/ROCKET AIR-BREATHING COMBINED CYCLE COMPARISON MISSIONS

A nominal Delta II mission was chosen to be simulated because of the high frequency of vehicle launches and availability of the corresponding data [27]. Figure 5.11 is a depiction of the Delta II rocket.

The mission consisted of delivering a 2032 kg payload, similar to the mass of a global positioning system (GPS) satellite, to an orbital altitude of 189000 m. The rocket air-breathing combined cycle vehicle was modeled approximately after Quicksat [6] and expected to accomplish the same Delta II mission. Figure 5.12 displays an image of the Quicksat vehicle.

Figure 5.11. Delta II Launch Vehicle

Figure 5.12. Quicksat Rocket/Air-Breathing Combined Cycle Vehicle Configuration [6]

5.2.1. Delta II Mission Overview and Parameters. The Delta II rocket was composed of 3 separate stages. The first stage contained 9 graphite-epoxy motor booster rockets fueled on Hydroxyl-terminated polybutadiene (HTPB) and a RS-27A engine burning LH_2/LOX propellant. Initially, 6 boosters were fired for approximately 60 seconds and ejected upon burnout. Next, the 3 remaining booster rockets were burnt until completion and then discarded. The RS-27A engine augmented the booster rockets by providing complimentary thrust throughout the entire first stage. Stage 2 was comprised of an AJ10-118K engine utilizing N_2O_4/Aerozine-50 as a propellant. The final stage employed a Star-48B engine which contained a solid mixture of NH_4ClO_4/Al/HTPB Binder (71%/18%/11%) as fuel. Table 5.6 summarizes the Delta II stage information for this mission. It is important to note that the information listed for the booster rockets corresponds to the parameters for one booster. The complete input deck and graphs related to this mission can be found in Appendix D.

Table 5.6. Delta II Rocket Stage Parameters

Parameter	Booster (9)	Stage 1	Stage 2	Stage 3
Fueled Mass (kg)	13080	101900	6953	2141
Empty Mass (kg)	1314	6100	949	132
Propellant	HTPB	RP-1/LOX	N_2O_4/A-50	NH_4ClO_4/Al/HTPB
Isp (s)	273.8	301.7	319.2	292.2
Thrust (N)	499100	889600	43580	66367

5.2.2. Rocket/Air-Breathing Combined Cycle Mission Overview and Parameters. The vehicle consists of an initial and final rocket stage with an intermediate air-breathing ramjet/scramjet stage. This differs from the Quicksat configuration which used turbines and rockets in the first stage. The initial stage was outfitted with 8 liquid-fueled (JP-7/H_2O_2) rocket engines. Taking advantage of external atmospheric air, the air-breathing system burnt JP-7 with this "free" oxidizer. The upper stage rocket also utilized JP-7/H_2O_2 as a propellant. Initially, the first stage rocket would accelerate and carry the vehicle to a specified altitude. Next, the first stage would transition to the air-breathing acceleration stage. At this point, the vehicle would accelerate up to Mach 10 while slightly climbing. Upon attaining Mach 10, the air-breathing portion of the vehicle would separate from the upper stage rocket and return for landing. Once isolated, the upper stage rocket would rotate and climb to the destination altitude. Table 5.7 summarizes the stage information for the rocket/air-breathing combined cycle vehicle. The data listed in Table 5.7 for stage 1 corresponds to the combination of all 8 rockets. The associated input deck and mission plots can be seen in Appendix D.

Table 5.7. Rocket/Air-Breathing Combined Cycle Stage Parameters

Parameter	Stage 1	Air-Breathing Stage	Final Stage
Fueled Mass (kg)	122574	172921	34665
Empty Mass (kg)	4311	125811	2764
Propellant	JP-7/H_2O_2	JP-7/Air	JP-7/H_2O_2
Isp (s)	329.9	-	336.1
Thrust (MN)	3.82	1.7 – 9	457900

5.2.3. Comparison of Results. Both vehicle configurations were able to deliver the 2032 kg payload successfully to a circular orbit of 189000 m. Figure 5.13 shows a comparison of the altitude time histories for each mission as well as the three distinct stages of the rocket/air-breathing combined cycle vehicle.

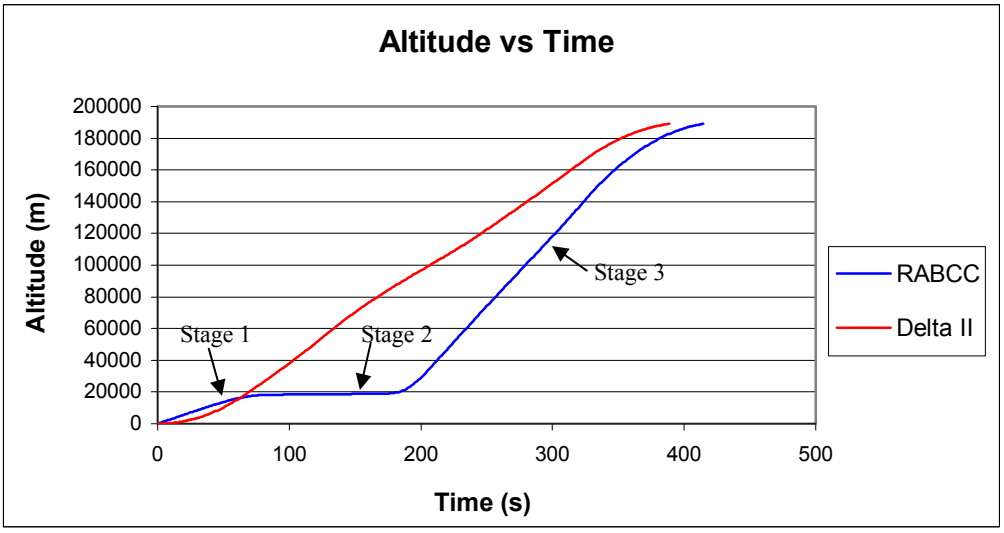


Figure 5.13. RABCC/Delta II Comparison of Altitude Time Histories

As expected, the Delta II rocket escaped the atmosphere much earlier than the rocket/air-breathing combined cycle vehicle. Figures 5.14 and 5.15 show the on-board energy usage for the Delta II and air-breathing mission, respectively.

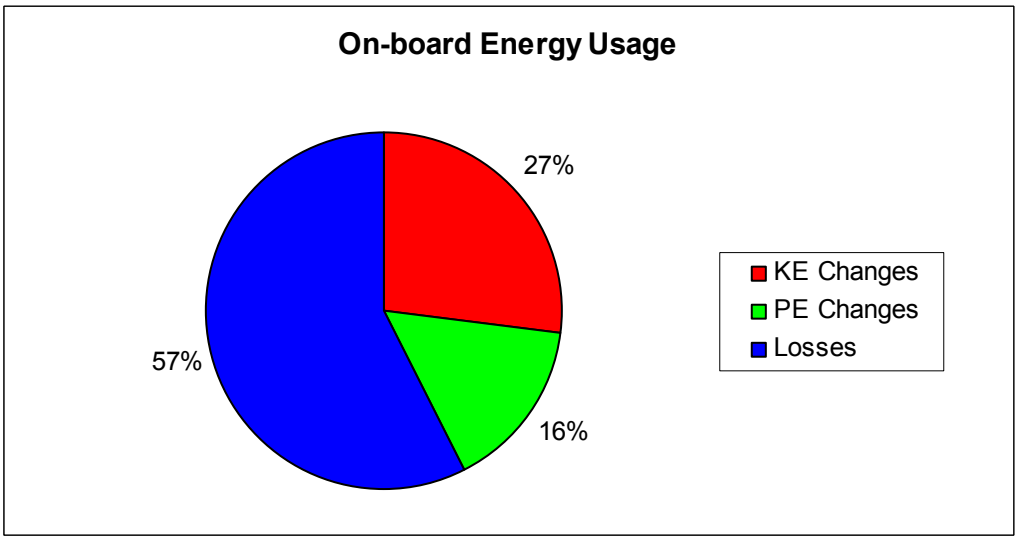


Figure 5.14. On-board Energy Usage for Delta II Mission

It is important to point out that Figures 5.14 and 5.15 represent the relative raw values of on-board energy used throughout the missions. Figure 5.15 displays a larger percentage of on-board energy used corresponding to changes in kinetic energy.

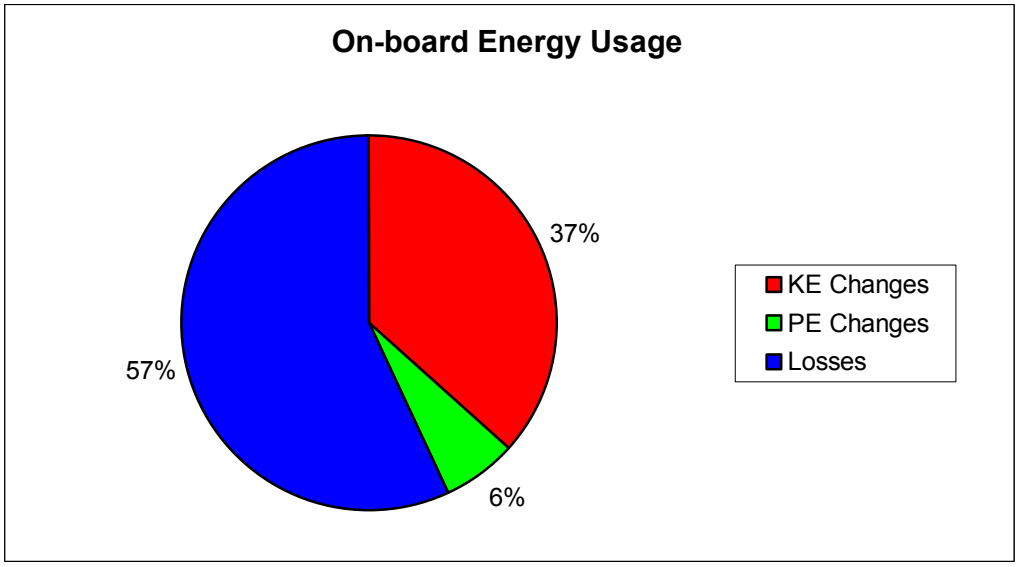


Figure 5.15. On-board Energy Usage for RABCC Mission

This makes sense because the air-breathing vehicle was accelerated more rapidly while containing more mass than the Delta II rocket. To further investigate these data and obtain a better understanding of what it represents, one must observe Figures 5.16 and 5.17, which show the vehicle mass fractions for the two missions.

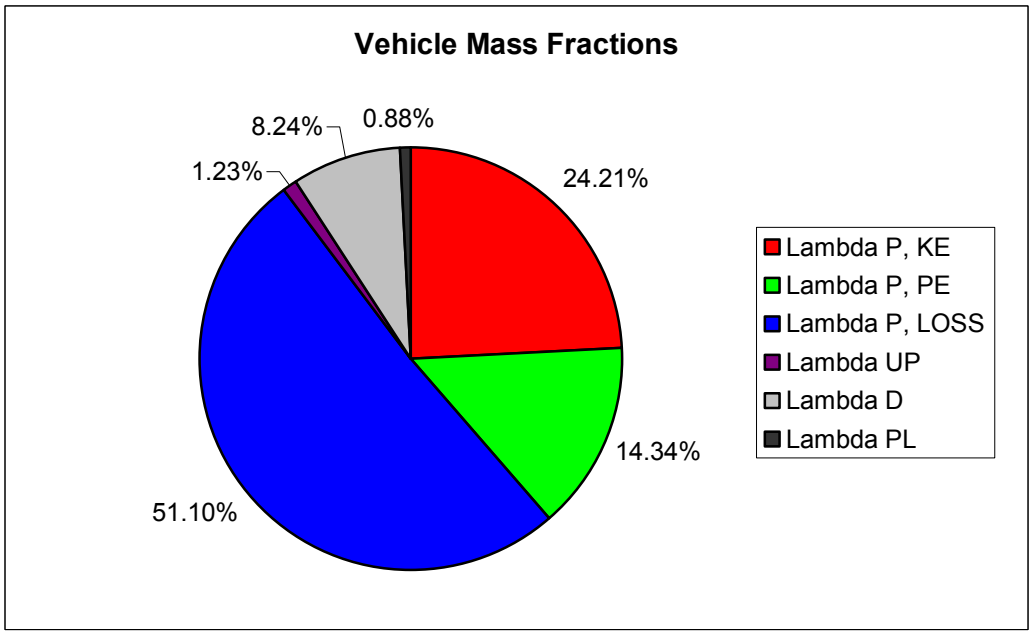


Figure 5.16. Vehicle Mass Fractions for Delta II Mission

By examining Figures 5.16 and 5.17, one can gain insight as to how the propellant mass was utilized and how the relative masses of the vehicle components compare with one another for the mission. For example, one typical difference between the two charts can be seen in the relative structural and propellant mass fractions. Figure 5.16 shows that only 8.2% of the mass of the Delta vehicle is structural whereas Figure 5.17 indicates that 40% of the combined cycle/air-breather's mass is structural. This can be attributed to the heavy air-breathing engine and structural materials. Like most multi-stage rockets, Figure 5.16 illustrates that the overall vehicle's mass is largely comprised of propellant (nearly 91%).

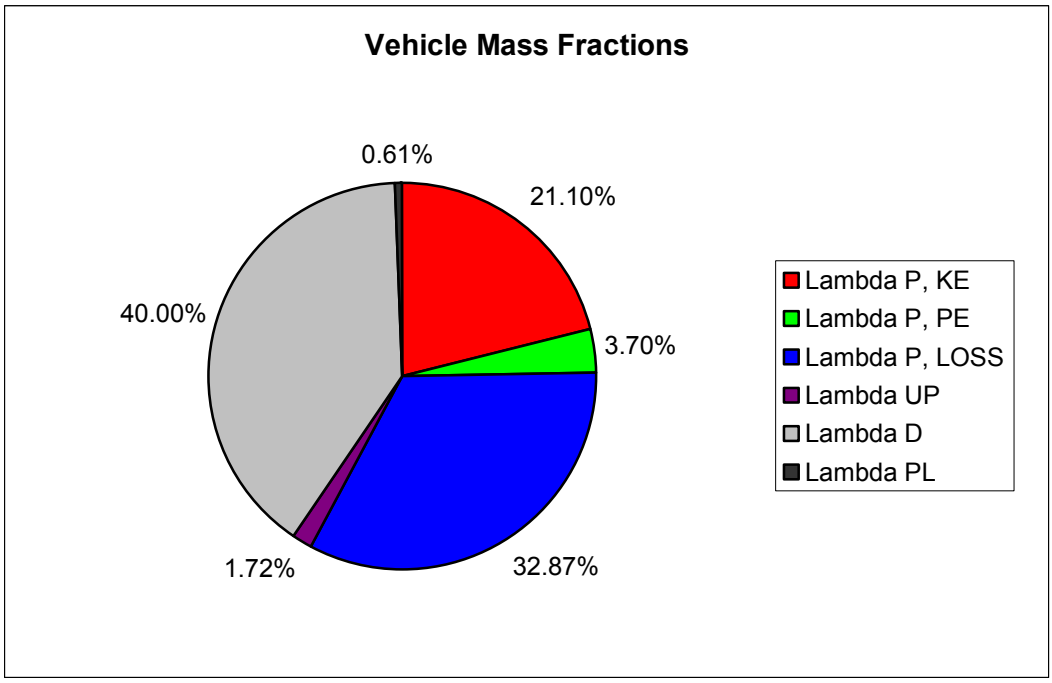


Figure 5.17. Vehicle Mass Fractions for RABCC Mission

On the other hand, since no oxidizer is carried on-board for the air-breathing leg, Figure 5.17 demonstrates the trade between propellant (nearly 60%) and structural mass for the RABCC mission. One can also observe that the percentage of propellant mass used to overcome losses due to irreversibilities as well as changes in potential energy is less for the air-breathing mission. Figures 5.18 and 5.19 represent the alternative

viewpoints of the vehicle mass fractions obtained by implementing the velocity change analysis.

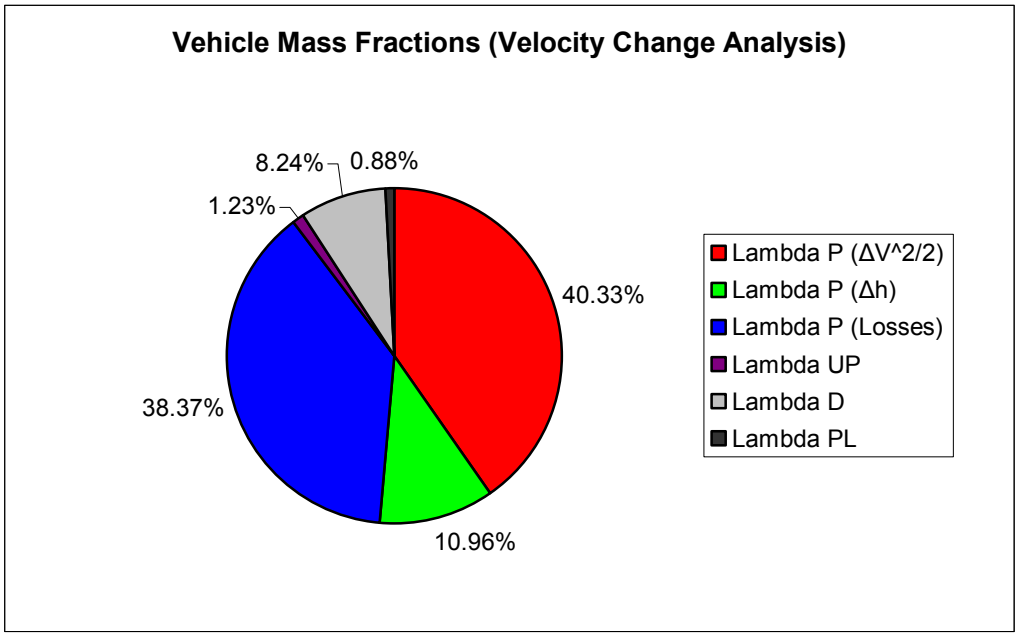


Figure 5.18. Vehicle Mass Fractions for Delta II Mission (Velocity Change Analysis)

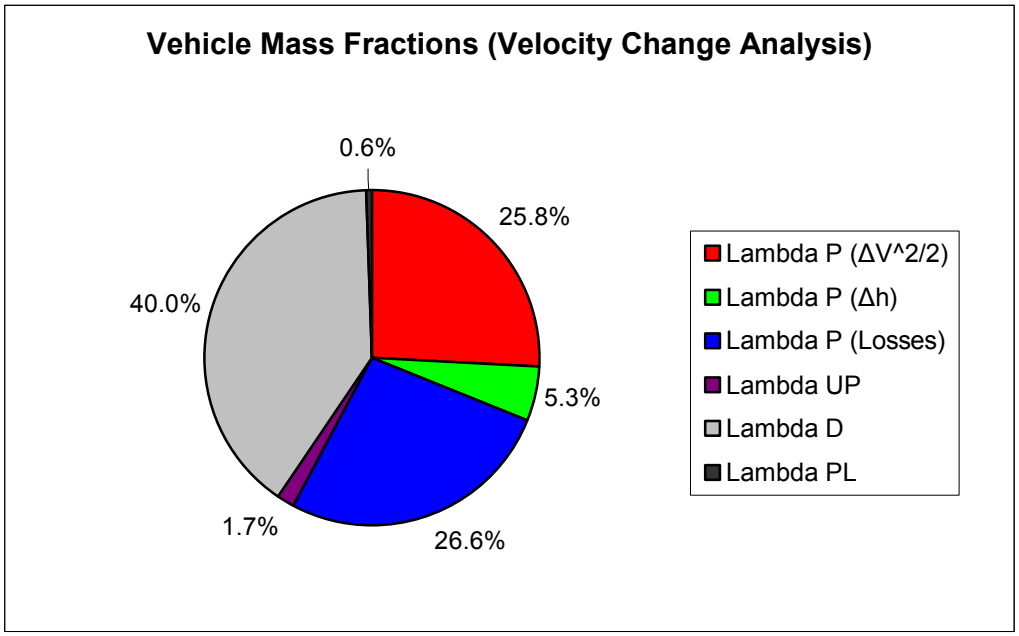


Figure 5.19. Vehicle Mass Fractions for RABCC Mission (Velocity Change Analysis)

Figures 5.20 and 5.21 depict the on-board energy usage time histories for the Delta II mission and RABCC mission. As mentioned previously, in both figures, one can notice periods in which the incremental changes in kinetic energy are negative.

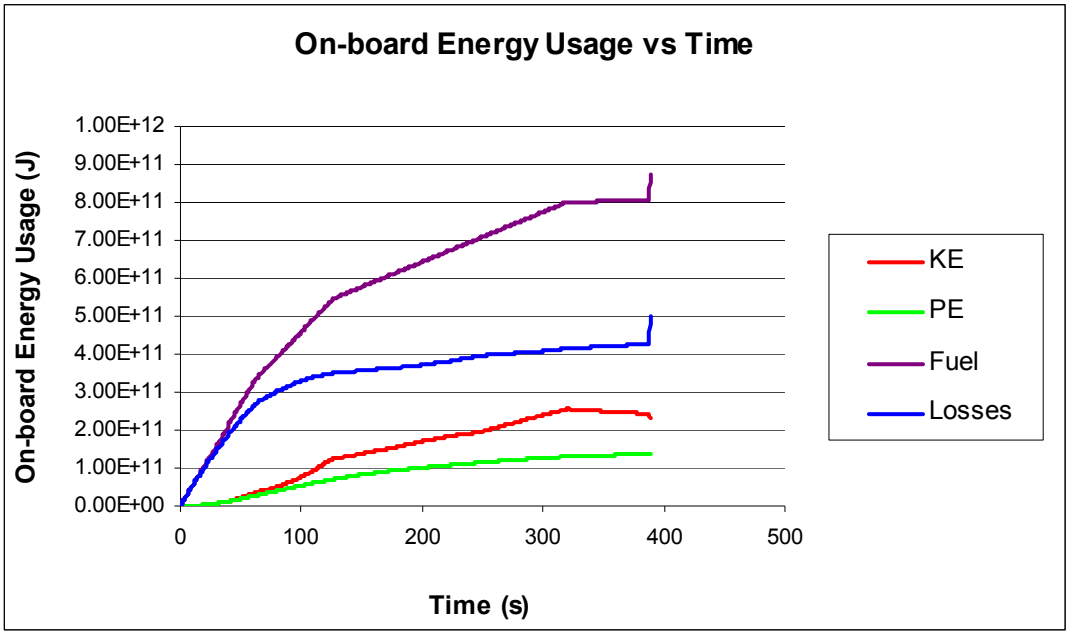


Figure 5.20. On-board Energy Usage Time History for Delta II Mission

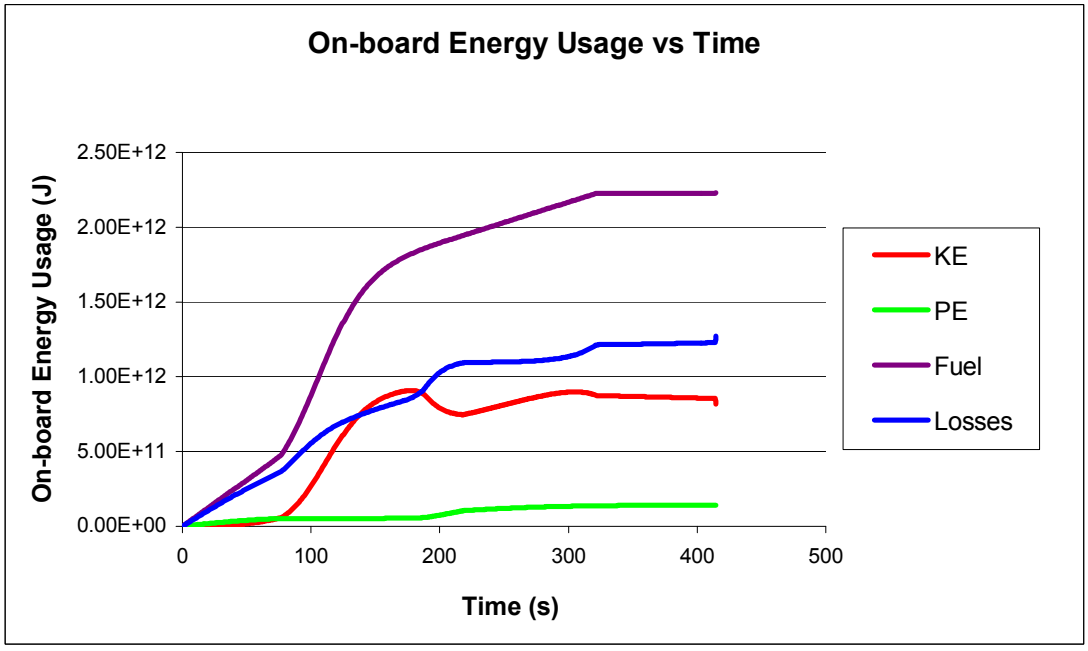


Figure 5.21. On-board Energy Usage Time History for RABCC Mission

5.3. DISCUSSION OF RESULTS

The Apollo 11 mission comparison between the simulated and actual data served as a reasonable validation of the trajectory solver. Specifically, the altitude time histories and trajectory plots were to shown to correspond very well. The implementation of the second law performance calculations allowed for the first time the thermodynamically consistent quantification of losses for the historic mission. The analytical method for computing loss information was shown to closely represent the results obtained by the integration of losses through the trajectory using numerical analysis. The sensitivities of this method were also shown.

The Delta II multi-stage rocket and rocket/air-breathing combined cycle comparison missions were successfully analyzed. The on-board energy usage for each mission was analyzed and discussed. A breakdown of the vehicle mass fractions from the standpoint of losses was generated and discussed for each configuration.

6. CONCLUSIONS

The second law theoretical framework for aerospace vehicles used in access-to-space missions has been developed from first principles. A trajectory code incorporating aerodynamics, propulsion, controls, and weight modeling was created to provide a tool with the ability to analyze missions of interest with emphasis on access-to-space applications. A code validation mission utilizing the Apollo 11 (to Earth orbit) configuration as well as a comparative study between more recent rocket and combined cycle (rocket/air-breathing) configurations was performed.

6.1. TRAJECTORY/ENTROPIC ANALYSIS OF ACCESS-TO-SPACE MISSIONS

Entropic-based analysis is a powerful and essential tool that should be used in the evaluation of access-to-space missions. This analysis can be invoked to obtain any level of detail regarding specific loss descriptions for a given mission. Sophisticated analysis tools and advanced models will provide more details regarding loss information. With this information, one can then make effective decisions associated with improving overall vehicle and mission performance.

The development of a trajectory solver code which incorporated the important second law impact measured by entropic gains associated with aerospace vehicle irreversibilities across space access missions was presented. Specifically, documented data corresponding to the Apollo 11 flight were used to validate this tool with great success. Finally, a comparison mission outlining the loss quantification between a Delta II rocket and a rocket/air-breathing combined cycle vehicle was shown. The air-breathing configuration was found to require less vehicle mass in the form of propellant to overcome irreversibilities as opposed to the Delta II rocket. However, this reduction in propellant mass is offset by increased structural considerations.

6.2. FUTURE WORK

Several model enhancements and additions remain as future work in the development of this trajectory solver. The original two-dimensional trajectory code could be expanded into a full three-dimensional solver. Vehicle modeling could be

advanced to that of a rigid body as opposed to the current point-mass model. More realistic aerodynamic coefficient modeling should be determined for the various vehicle configurations. For example, computational fluid dynamic (CFD) analysis could be used to determine representative lift and drag values across various Mach and altitude ranges. These values could be curve-fit and then actively implemented into the trajectory solver. The modeling of the rocket/air-breathing combined cycle could be improved by constructing individual turbojet and ram/scramjet models with component efficiencies. A constant dynamic pressure trajectory could also be implemented for the RABCC.

From a more innovative standpoint, effective on-board energy usage could be employed, in a variety of ways, to determine more efficient flight trajectories. Some of these ways are shown in Figure 6.1.

Figure 6.1. Various Usages of On-board Energy

As the technology associated with the assorted exploitations of on-board energy shown in Figure 6.1 become more realistic, the implementation of an air-breathing configuration may be deemed practical. Finally, with all of the above enhancements included, a reexamination of the entropic and trajectory analysis could be conducted.

APPENDIX A
TRAJECTORY SOLVER CODE

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!       Name:  Tyler Winter
!       Date of Last Revision:  1/22/2008
!       Description:  This program is a full 2-d trajectory solver for 3 different
!               aerospace vehicles (SSR, RABCC, MSR).  The program simulates the entire
!               mission from ground launch until a low-Earth circular orbit is established.
!               This program also contains many subroutines that perform the necessary
!               calculations to determine important parameters (Lift, Thrust, etc.).
!       Subroutines:
!               AA_Model(constaoa, thetaaoa, aoa)
!               Atmospheric_Model(alt, amodel, Talt, Palt, Rhoalt)
!               Btbm(thetatbm, Mtbm, valuebeta)
!               Drag_Model(FSRhod, FSTd, FSVd, wingAd, crossAd, alphad, dmodel, drag)
!               Gravity_Model(alt, gmodel, galt)
!               Lift_Model(FSRhol, FSTl, FSVl, wingAl, alphal, lmodel, lift)
!               Lost_Work(deltape, deltake, deltafe, deltalw)
!               MPMF(numpmf, newvaluempmf)
!               PMF(pmfM, valuepmf)
!               Thrust_Model(FSRhot, FSTt, FSPt, FSVt, Aratio, crossA, Tott4, hvf,
!                       omass, vmodel, nmass, thrust)
!               Weight_Model(xpos, ypos, vmass, thetaw, wmodel, xweight, yweight)
!               Write_Data(xwd, ywd, mvwd, thetawd, alphawd, twd, lwd, dwd, wxwd, wywd,
!                       timewd, vwd, tempwd, preswd, denswd, altwd, qinfwd, accelwd, dpewd,
!                       dkewd, dfewd, dlwwd, phiwd, dVxwd, dVywd)
!       Definition of Variables:
!       -Real Parameters-
!               G               -       Gravitational Constant (m^3/kg-s^2)
!               g0              -       Acceleration due to Gravity at Sea-level (m/s^2)
!               Me              -       Mass of Earth (kg)
!               pi              -       Ratio of a Circle's Circumference to its Diameter
!               Psl             -       Pressure at Sea-level (N/m^2)
!               Re              -       Radius of Earth (m)
!               Rhosl           -       Density at Sea-level (kg/m^3)
!               Tsl             -       Temperature at Sea-level (K)
!       -Real Arrays-
!               CMSR            -       Effective Exhaust Velocities for each stage of the MSR (m/s)
!               MFMSR           -       Propellant Flow Rate for each stage of the MSR (kg/s)
!               MMSR            -       Total Mass for each stage of the MSR (kg)
!               MSTRMSR         -       Structural Mass for each stage of the MSR (kg)
!       -Characters-
!               FileName        -       File name of the input file to be read
!       -Integers-
!               abstage         -       Current Propulsion Stage for RABCC
!               a_model         -       Atmosphere Model (0 - Exponential, 1 - 7-part)
!               d_model         -       Drag Model (0 - Basic, 1 - Improved)
!               e_model         -       Earth Model (0 - Flat Earth, 1 - Round Earth)
!               g_model         -       Gravity Model for g (0 - Sea-level, 1 - g(alt))
!               i               -       Multi-purpose counter
!               InputStatus     -       Input flag for reading input file
!               j               -       Multi-purpose counter
!               l_model         -       Lift Model (0 - Small Angle, 1 - Oblique S/E)
!               NRS             -       Number of rocket stages
!               num_lines       -       Number of lines to skip between each written line of data
!               OpenStatus      -       Flag for opening a file
!               rstage          -       Current rocket stage
!               tr_model        -       Trajectory Model (0 - CAA, 1 - SA CAA, 2 - SA AC)
!               v_model         -       Vehicle Model (0 - SSR, 1 - ARCC, 2 - MSR)
!               writeflag       -       Flag to indicate whether to write or not to a file
!       -Reals-
!               abalpha         -       Constant AB Wing Angle (deg)
!               ABbodyfactor    -       AB Body Factor (multiplied by Cdb)
!               abrtransalt     -       AB-R Transition Altitude (m)
!               abrtransM       -       AB-R Transition Mach Number (6-10)
!               absmwasf        -       Absolute Maximum Wing Angle Scale Factor
!                                       Wing alpha increases to this limit then snaps back to
!                                       maxrwalpha for coasting
!               Ac              -       Cross-section Area of Vehicle (m^2)
!               AcapoAc         -       Ratio of Capture Area to Cross-sectional Area
!               accel           -       Instantaneous Acceleration (m/s^2)
!               alpha           -       Wing Angle of Attack (rad)
```

```
!   alpharr1      -   R-AB Rotational Rate of Wing (deg/s)
!   alpharr2      -   AB-R Rotational Rate of Wing (deg/s)
!   alti          -   Initial Altitude (m)
!   altitude      -   Current Altitude (m)
!   boalt         -   Burnout Altitude (m)
!   calpha        -   Initial Wing Angle of Attack (deg)
!   Cdbody        -   Coefficient of Drag for Rocket Body
!   CF            -   Final Rocket Effective Exhaust Velocity (m/s)
!   C1            -   Main Rocket Effective Exhaust Velocity (m/s)
!   Cp            -   Specific Heat at Constant Pressure (J/kg-K)
!   D             -   Current Drag (N)
!   dac           -   Delta Alpha Correction (rad)
!   dalphalim     -   Max Change in Wing Angle Limit (deg)
!   deltaV        -   Change in Velocity required to attain orbit (m/s)
!   density       -   Density at current altitude (kg/m^3)
!   difffe        -   Running sum of energy content of expended fuel (J)
!   diffke        -   Running sum of kinetic energy (J)
!   difflw        -   Running sum of lost work (J)
!   diffpe        -   Running sum of potential energy (J)
!   dmkesum       -   Running sum of diff. prop. mass for kinetic energy (kg)
!   dmlwsum       -   Running sum of diff. prop. mass for lost work (kg)
!   dmpesum       -   Running sum of diff. prop. mass for potential energy (kg)
!   dt            -   Time Step (s)
!   dtheta        -   Differential change in vehicle angle to the vertical (rad)
!   dVx           -   Differential change in x-component of vehicle velocity (m/s)
!   dVy           -   Differential change in y-component of vehicle velocity (m/s)
!   fmass         -   Final Vehicle Mass (kg)
!   fuelspent     -   Total Mass of Fuel Spent (kg)
!   gamma         -   Ratio of Specific Heats
!   h             -   Heating Value of Fuel (J/kg)
!   imaxalt       -   Instantaneous Max Altitude (m)
!   ismootha1     -   R-AB Initial Smoothing Alpha Limit (deg)
!   L             -   Current Lift (N)
!   LA            -   Launch Angle measured from horizon (<= 90 deg)
!   LAmax         -   Max Vehicle Angle to Horizon (<= 90 deg)
!   lamd          -   Structural Vehicle Mass Fraction
!   LAmin         -   Min Vehicle Angle to Horizon (<= 0 deg)
!   laml          -   Payload Vehicle Mass Fraction
!   mABir         -   AB Initial Rocket Mass Drop (structural) (kg)
!   mabthetah     -   AB-R Transition Max Horizon Vehicle Angle (deg)
!   maxalpha      -   Max Wing Angle (deg)
!   mdotpf        -   Mass Flow Rate of Final Rocket Propellant (kg/s)
!   mdotp1        -   Mass Flow Rate of Main Rocket Propellant (kg/s)
!   mpayload      -   Payload Mass (kg)
!   mprop         -   Mass of Propellant (kg)
!   mrwalpha      -   R-Ballastic Transition Max Wing Angle (deg)
!   mstr          -   Structural (System) Mass (kg)
!   mv0           -   Initial Total Vehicle Mass (including propellant) (kg)
!   newmass       -   Current Total Vehicle Mass (kg)
!   oldalt        -   Altitude from previous time step (m)
!   oldmass       -   Total Vehicle Mass from previous time step (kg)
!   orbalt        -   Orbital Altitude (m)
!   phi           -   Global Vehicle Rotation Angle (rad)
!   pressure      -   Pressure at current altitude (N/m^2)
!   qinf          -   Dynamic Pressure (N/m^2)
!   rabtransalt   -   R-AB Transition Altitude (m)
!   rae           -   Rocket Accelerator Efficiency
!   remainmstr    -   Remaining Structural Mass (kg)
!   Rgas          -   Gas Constant (J/kg-K)
!   smootha1      -   R-AB Smoothing Alpha Limit (deg)
!                     1st leg transition angle value at which below this
!                     value no additional smoothing is needed, if alpharr1
!                     is greater than this value, smoothing is needed
!   smootha2      -   AB-R Smoothing Alpha Limit (deg)
!                     2nd leg transition angle value at which below this
!                     value no additional smoothing is needed, if alpharr2
!                     is greater than this value, smoothing is needed
!   SumMRS        -   Sum of the mass of the rocket stages (kg)
!   SummstrMRS    -   Sum of the structural mass of the rocket stages (kg)
!   Sw            -   Planform Area of Wings/Lifting Body (m^2)
```

```
!                    T                   -          Current Thrust (N)
!                    temperature         -          Temperature at current altitude (K)
!                    thetamax            -          Max Vehicle Angle measured from the vertical (rad)
!                    thetamin            -          Min Vehicle Angle measured from the veritcal (rad)
!                    thetanew            -          Current Vehicle Angle measured from the vertical (rad)
!                    thetaold            -          Veh. Angle measured from vert. from prev. time step (rad)
!                    time                -          Current Time (s)
!                    totaltime           -          Total Time allowed for mission (s)
!                    Tt0                 -          Total Temperature at Engine Inlet (K)
!                    Tt4                 -          Total Temperature at Combustor Exit (K)
!                    Vcirc               -          Velocity of vehicle in circular orbit (m/s)
!                    Vi                  -          Initial Velocity (m/s)
!                    Vnew                -          Current Vehicle Velocity (m/s)
!                    Vold                -          Vehicle Velocity from previous time step (m/s)
!                    Wx                  -          X-component of Weight in Earth-fixed frame (N)
!                    Wy                  -          Y-component of Weight in Earth-fixed frame (N)
!                    xnew                -          Current x-comp of altitude in Earth-fixed frame (m)
!                    xold                -          X-comp of alt. in Earth-fixed frame from prev. time step (m)
!                    ynew                -          Current y-comp of altitude in Earth-fixed frame (m)
!                    yold                -          Y-comp of alt. in Earth-fixed frame from prev. time step (m)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Program Trajectory_Solver

Implicit None

!        Declaration of Variables
Real(kind = 8), Parameter :: pi = 3.1415926535897932384, g0 = 9.81, Psl = 101325.0
Real(kind = 8), Parameter :: Tsl = 288.16, Rhosl = 1.225, Re = 6378137.0
Real(kind = 8), Parameter :: G = 6.67e-11, Me = 5.97e24
Real(kind = 8), Dimension(:), Allocatable :: CMSR, MMSR, MFMSR, MSTRMSR
Character(20) :: FileName
Integer :: OpenStatus, InputStatus, a_model, d_model, g_model, l_model, v_model
Integer :: e_model, tr_model, num_lines, writeflag, NRS, i, j, rstage = 1
Integer :: abstage
Real(kind = 8) :: dt, mv0, Sw, Ac, C1, CF, mdotp1, h, Tt4, AcapoAc, gamma, Cp, Rgas
Real(kind = 8) :: altitude, temperature, pressure, density, alpha, Vi, alti
Real(kind = 8) :: T, L, D, Wx, Wy, dtheta, time = 0, Vnew, Vold, newmass, mdotpf
Real(kind = 8) :: oldmass, LA, dVx, dVy, Tt0, xold, xnew, yold, ynew, thetanew
Real(kind = 8) :: thetaold, totaltime, LAmax, LAmin, thetamax, thetamin, boalt
Real(kind = 8) :: fmass, Vcirc, deltaV, accel = 0, mprop, phi, rabtransalt, abrtransM
Real(kind = 8) :: mrwalpha, qinf, alpharr1, alpharr2, mabthetah, calpha = 0
Real(kind = 8) :: ismootha1, smootha1, smootha2, absmwasf, oldalt = 0, diffpe = 0
Real(kind = 8) :: diffke = 0, difffe = 0, difflw = 0, orbalt, Cdbody, SummstrMRS = 0
Real(kind = 8) :: ABbodyfactor, dalphalim, imaxalt, maxalpha, mstr, SumMRS = 0
Real(kind = 8) :: dmkesum = 0, dmpesum = 0, dmlwsum = 0, remainmstr, mABir, rae
Real(kind = 8) :: lamd, laml, abalpha, abrtransalt, mpayload, fuelspent = 0, dac

Print *
Write(*, '(1X, A)', Advance = "NO") "Enter the input file name: "
Read *, FileName
Print *

!        Open and begin reading input file
Open(Unit = 10, File = FileName, Status = "OLD", Action = "READ", &
         & Position = "REWIND", Iostat = OpenStatus)
If(OpenStatus > 0) STOP "**** CANNOT OPEN FILE ****"

Read(10, *, Iostat = InputStatus)
Read(10, *, Iostat = InputStatus) a_model
Read(10, *, Iostat = InputStatus) d_model
Read(10, *, Iostat = InputStatus) e_model
Read(10, *, Iostat = InputStatus) g_model
Read(10, *, Iostat = InputStatus) l_model
Read(10, *, Iostat = InputStatus) v_model
Read(10, *, Iostat = InputStatus) tr_model
Read(10, *, Iostat = InputStatus)
Read(10, *, Iostat = InputStatus) writeflag
Read(10, *, Iostat = InputStatus) num_lines
```

```
Read(10, *, Iostat = InputStatus) dt
Read(10, *, Iostat = InputStatus) totaltime
Read(10, *, Iostat = InputStatus) alti
Read(10, *, Iostat = InputStatus) orbalt
Read(10, *, Iostat = InputStatus) boalt
Read(10, *, Iostat = InputStatus) gamma
Read(10, *, Iostat = InputStatus) Rgas
Read(10, *, Iostat = InputStatus)
Read(10, *, Iostat = InputStatus) Vi
Read(10, *, Iostat = InputStatus) calpha
Read(10, *, Iostat = InputStatus) LA
Read(10, *, Iostat = InputStatus) dac
Read(10, *, Iostat = InputStatus) dalphalim
Read(10, *, Iostat = InputStatus) maxalpha
Read(10, *, Iostat = InputStatus) LAmax
Read(10, *, Iostat = InputStatus) LAmin
Read(10, *, Iostat = InputStatus) Sw
Read(10, *, Iostat = InputStatus) Ac
Read(10, *, Iostat = InputStatus) Cdbody
Read(10, *, Iostat = InputStatus)
Read(10, *, Iostat = InputStatus) abalpha
Read(10, *, Iostat = InputStatus) rabtransalt
Read(10, *, Iostat = InputStatus) alpharr1
Read(10, *, Iostat = InputStatus) ismootha1
Read(10, *, Iostat = InputStatus) smootha1
Read(10, *, Iostat = InputStatus) alpharr2
Read(10, *, Iostat = InputStatus) smootha2
Read(10, *, Iostat = InputStatus) abrtransM
Read(10, *, Iostat = InputStatus) abrtransalt
Read(10, *, Iostat = InputStatus) mabthetah
Read(10, *, Iostat = InputStatus) mrwalpha
Read(10, *, Iostat = InputStatus) absmwasf
Read(10, *, Iostat = InputStatus) h
Read(10, *, Iostat = InputStatus) Tt4
Read(10, *, Iostat = InputStatus) Cp
Read(10, *, Iostat = InputStatus) AcapoAc
Read(10, *, Iostat = InputStatus) ABbodyfactor
Read(10, *, Iostat = InputStatus)
Read(10, *, Iostat = InputStatus) mdotp1
Read(10, *, Iostat = InputStatus) mdotpf
Read(10, *, Iostat = InputStatus) C1
Read(10, *, Iostat = InputStatus) CF
Read(10, *, Iostat = InputStatus) rae
Read(10, *, Iostat = InputStatus) NRS
Read(10, *, Iostat = InputStatus)
Read(10, *, Iostat = InputStatus) mv0
Read(10, *, Iostat = InputStatus) mstr
Read(10, *, Iostat = InputStatus) mpayload
Read(10, *, Iostat = InputStatus) mABir

!               Attempt to mitigate input errors
If(InputStatus > 0) STOP "**** INPUT ERROR ****"
If(Rgas <= 0) STOP "NEGATIVE OR ZERO GAS CONSTANT!"
If((a_model /= 0) .and. (a_model /= 1)) Then
                STOP "ATMOSPHERE MODEL SELECTION INVALID!"
Else If((d_model /= 0) .and. (d_model /= 1)) Then
                STOP "DRAG MODEL SELECTION INVALID!"
Else If((e_model /= 0) .and. (e_model /= 1)) Then
                STOP "EARTH MODEL SELECTION INVALID!"
Else If((g_model /= 0) .and. (g_model /= 1)) Then
                STOP "GRAVITY MODEL SELECTION INVALID!"
Else If((l_model /= 0) .and. (l_model /= 1)) Then
                STOP "LIFT MODEL SELECTION INVALID!"
Else If((v_model /= 0) .and. (v_model /= 1) .and. (v_model /= 2)) Then
                STOP "VEHICLE MODEL SELECTION INVALID!"
Else If((tr_model /= 0) .and. (tr_model /= 1) .and. (tr_model /= 2)) Then
                STOP "TRAJECTORY MODEL SELECTION INVALID!"
Else If((writeflag /= 0) .and. (writeflag /= 1)) Then
                STOP "WRITEFLAG SELECTION INVALID!"
End If
```

```
If(dalphalim <= dac) Then
        STOP "MAX CHANGE IN WING ALPHA LESS THAN OR EQUAL TO DELTA ALPHA CORRECTION!"
Else If(dalphalim <= alpharr1) Then
        STOP "MAX CHANGE IN WING ALPHA LESS THAN OR EQUAL TO R-AB ROTATIONAL RATE!"
Else If(dalphalim <= ismootha1) Then
        STOP "MAX CHANGE IN WING ALPHA LESS THAN OR EQUAL TO INITIAL SMOOTHING ALPHA!"
Else If(dalphalim <= smootha1) Then
        STOP "MAX CHANGE IN WING ALPHA LESS THAN OR EQUAL TO SMOOTHING ALPHA 1!"
Else If(dalphalim <= alpharr2) Then
        STOP "MAX CHANGE IN WING ALPHA LESS THAN OR EQUAL TO AB-R ROTATIONAL RATE!"
Else If(dalphalim <= smootha2) Then
        STOP "MAX CHANGE IN WING ALPHA LESS THAN OR EQUAL TO SMOOTHING ALPHA 2!"
End If
If(maxalpha <= calpha) Then
        STOP "MAX WING ALPHA LESS THAN OR EQUAL TO INITIAL WING ALPHA!"
Else If(maxalpha <= dalphalim) Then
        STOP "MAX WING ALPHA LESS THAN OR EQUAL TO MAX CHANGE IN WING ALPHA!"
Else If(maxalpha <= alpharr1) Then
        STOP "MAX WING ALPHA LESS THAN OR EQUAL TO R-AB ROTATIONAL RATE!"
Else If(maxalpha <= ismootha1) Then
        STOP "MAX WING ALPHA LESS THAN OR EQUAL TO INITIAL SMOOTHING ALPHA!"
Else If(maxalpha <= smootha1) Then
        STOP "MAX WING ALPHA LESS THAN OR EQUAL TO SMOOTHING ALPHA 1!"
Else If(maxalpha <= alpharr2) Then
        STOP "MAX WING ALPHA LESS THAN OR EQUAL TO AB-R ROTATIONAL RATE!"
Else If(maxalpha <= smootha2) Then
        STOP "MAX WING ALPHA LESS THAN OR EQUAL TO SMOOTHING ALPHA 2!"
Else If(maxalpha <= abalpha) Then
        STOP "MAX WING ALPHA LESS THAN OR EQUAL TO AB ALPHA!"
End If
If((rae < 0) .or. (rae > 1.0)) STOP "ROCKET ACCELERATOR EFFICIENCY MUST BE BETWEEN 0 AND 1!"
If(abalpha < 0) STOP "NEGATIVE AB ALPHA VALUE!"
If(AcapoAc <= 0) STOP "NEGATIVE OR ZERO RATIO OF CAPTURE TO CROSS-SECTIONAL AREA!"
If(ABbodyfactor <= 0) STOP "NEGATIVE OR ZERO ABBODYFACTOR VALUE!"
If(h <= 0) STOP "NEGATIVE OR ZERO HEATING VALUE OF FUEL!"
If(alti < 0) STOP "NEGATIVE INITIAL ALTITUDE!"
If(dac <= 0) STOP "NEGATIVE OR ZERO DELTA ALPHA CORRECTION!"
If(orbalt < 105000) Print*, "WARNING!  ORBITAL ALTITUDE LIES WITHIN ATMOSPHERE!"
If(Cdbody < 0) STOP "NEGATIVE COEFFICIENT OF BODY DRAG VALUE!"
If(calpha < 0) STOP "NEGATIVE WING ANGLE!"
If(ismootha1 <= 0) STOP "NEGATIVE OR ZERO INITIAL SMOOTHING RATE!"
If((smootha1 <= 0) .or. (smootha2 <= 0)) STOP "NEGATIVE OR ZERO SMOOTHING RATE!"
If((alpharr1 <= 0) .or. (alpharr2 <= 0)) STOP "NEGATIVE OR ZERO ROTATIONAL RATE OF WING!"
If(dt <= 0) STOP "NEGATIVE OR ZERO TIME STEP!"
If(totaltime <= 0) STOP "NEGATIVE OR ZERO TOTAL TIME!"
If(dt >= totaltime) STOP "TIME STEP GREATER THAN OR EQUAL TO TOTAL TIME!"
If(num_lines <= 0) STOP "NEGATIVE OR ZERO NUMBER OF LINES TO PRINT!"
If(Vi <= 0) STOP "NEGATIVE OR ZERO INITIAL VELOCITY!"
If(rabtransalt < 0) STOP "NEGATIVE ROCKET-TO-AB TRANSITION ALTITUDE!"
If(rabtransalt <= alti) STOP "TRANISITION ALTITUDE IS LESS THAN OR EQUAL TO INITIAL ALTITUDE!"
If(abrtransalt < rabtransalt) STOP "AB-TO-ROCKET LESS THAN ROCKET-TO-AB TRANSITION ALTITUDE!"
If(abrtransalt <= alti) STOP "TRANISITION ALTITUDE IS LESS THAN OR EQUAL TO INITIAL ALTITUDE!"
If(boalt <= alti) STOP "BURNOUT ALTITUDE IS LESS THAN OR EQUAL TO INITIAL ALTITUDE!"
If(boalt <= rabtransalt) STOP "BURNOUT ALTITUDE IS LESS THAN OR EQUAL TO TRANSITION ALTITUDE!"
If(boalt <= abrtransalt) STOP "BURNOUT ALTITUDE IS LESS THAN OR EQUAL TO TRANSITION ALTITUDE!"
If(absmwasf < 1.0) STOP "ABSOLUTE MAXIMUM WING ANGLE SCALE FACTOR LESS THAN ONE!"
If((abrtransM < 4.0) .or. (abrtransM > 15.0)) STOP "TRANSITION MACH NUMBER OUTSIDE RANGE!"
If(mrwalpha <= 0) Print *, "WARNING!!  NEGATIVE OR ZERO TRANSITION ROCKET WING ANGLE!"
If((mabthetah < LAmin) .or. (mabthetah > LAmax)) STOP "MAX AB ANGLE TO HORIZON OUTSIDE RANGE!"
If(mABir < 0) STOP "AB INITIAL ROCKET MASS DROP NEGATIVE!"
If(mABir >= mstr) STOP "AB INITIAL ROCKET MASS DROP GREATER THAN OR EQUAL TO MSTR!"
If(mv0 <= 0) STOP "NEGATIVE OR ZERO INITIAL MASS!"
If(mpayload < 0) STOP "NEGATIVE PAYLOAD MASS!"
If(mv0-mstr-mpayload <= 0) STOP "NEGATIVE OR ZERO PROPELLANT MASS!"
If(mv0-mstr-mpayload >= mv0) STOP "PROPELLANT MASS IS GREATER THAN OR EQUAL TO TOTAL MASS!"
If(Sw <= 0) STOP "NEGATIVE OR ZERO WING AREA!"
If(Ac <= 0) STOP "NEGATIVE OR ZERO CROSS-SECTIONAL AREA!"
If(C1 <= 0) STOP "NEGATIVE OR ZERO EFFECTIVE EXHAUST VELOCITY!"
If(CF <= 0) STOP "NEGATIVE OR ZERO EFFECTIVE EXHAUST VELOCITY!"
If(mdotp1 <= 0) STOP "NEGATIVE OR ZERO MASS FLOW RATE OF PROPELLANT!"
```

```
If(mdotpf <= 0) STOP "NEGATIVE OR ZERO MASS FLOW RATE OF PROPELLANT!"
If(Tt4 <= 0) STOP "NEGATIVE OR ZERO COMBUSTOR EXIT TOTAL TEMPERATURE!"
If(LA < 0) STOP "NEGATIVE LAUNCH ANGLE!"
If(LA > LAmax) STOP "LAUNCH ANGLE IS GREATER THAN MAX ANGLE TO HORIZON!"
If(LA < LAmin) STOP "LAUNCH ANGLE IS SMALLER THAN MIN ANGLE TO HORIZON!"
If(LAmax <= 0) STOP "NEGATIVE OR ZERO MAX ANGLE TO HORIZON!"
If(LAmax > 90) STOP "MAX ANGLE TO HORIZON IS TOO LARGE!"
If((calpha > 0) .and. (LA == 90)) STOP "POSITIVE WING ANGLE WITH VERTICAL LAUNCH!"
If(LAmin < 0) Print*, "WARNING!!  NEGATIVE MIN ANGLE TO HORIZON!"
If(LAmax <= LAmin) STOP "MAX LESS THAN OR EQUAL TO MIN ANGLE TO HORIZON!"
If(gamma <= 0) STOP "NEGATIVE OR ZERO RATIO OF SPECIFIC HEATS!"
If(Cp <= 0) STOP "NEGATIVE OR ZERO SPECIFIC HEAT AT CONSTANT PRESSURE!"
If(NRS < 2) STOP "PLEASE USE SSR FOR VEHICLE MODEL OR INCREASE NUMBER OF ROCKET STAGES!"

!       Initializing structural mass values
If(v_model == 0) Then
        remainmstr = mstr
Else If(v_model == 1) Then
        remainmstr = mstr

!       Collecting various information regarding the multi-stage rocket as well as error
!       trapping for inputs made by user
Else If(v_model == 2) Then
        Allocate(CMSR(NRS), MMSR(NRS), MFMSR(NRS), MSTRMSR(NRS))
        Print*, "The number of stages for the multi-staged rocket was designated as: ", NRS
        Print*, "Input the effective exhaust velocity (m/s) and necessary masses (kg)."
        Print*, "Also enter the mass flow rate of propellant (kg/s) of each stage."
        Do i = 1, NRS
                Do
                        Print*, "C for stage", i, ": "
                        Read *, CMSR(i)
                        If(CMSR(i) > 0) Then
                                Exit
                        Else
                                Print*, "Please enter a positive effective exhaust velocity!"
                        End If
                End Do
                Do
                        Print*, "Total mass for stage", i, ": "
                        Read *, MMSR(i)
                        Print*, "Structural mass for stage", i, ": "
                        Read *, MSTRMSR(i)
                        Do j = 1, i
                                SumMRS = SumMRS + MMSR(j)
                                SummstrMRS = SummstrMRS + MSTRMSR(j)
                        End Do
                        If((MMSR(i) > 0) .and. (SumMRS <= mv0-mpayload) .and. (SummstrMRS <= mstr)) Then
                                SumMRS = 0
                                SummstrMRS = 0
                                Exit
                        Else If(MMSR(i) <= 0) Then
                                Print*, "Please enter a positive mass!"
                                SumMRS = 0
                                SummstrMRS = 0
                        Else If(MSTRMSR(i) <= 0) Then
                                Print*, "Please enter a positive mass!"
                                SumMRS = 0
                                SummstrMRS = 0
                        Else If(SumMRS > mv0-mpayload) Then
                                Print*, "Sum of mass for stages exceeds dropped and propellant mass allowed value!"
                                Stop
                        Else If(SummstrMRS > mstr) Then
                                Print*, "Sum of individual structural mass exceeds specifed total structural mass!"
                                Stop
                        End If
                End Do
                Do
                        Print*, "Mass flow rate of propellant for stage", i, ": "
                        Read *, MFMSR(i)
                        If(MFMSR(i) > 0) Then
```

```
                                        Exit
                                Else
                                        Print*, "Please enter a positive mass flow rate of propellant!"
                                End If
                        End Do
                End Do
                remainmstr = mstr
End If

!               Initializing several variables that are to be used in the initial calling of each
!               subroutine before the mission is begun
xold = 0.0
yold = alti
dVx = 0.0
dVy = 0.0
LA = LA*(pi/180.)
LAmax = LAmax*(pi/180.)
LAmin = LAmin*(pi/180.)
thetamax = (pi/2.) - LAmin
thetamin = (pi/2.) - LAmax
calpha = calpha*(pi/180.)
If(tr_model == 0) Then
                thetanew = pi/2.
Else
                thetanew = (pi/2.) - LA
End If
Vnew = Vi
newmass = mv0
mprop = mv0 - mstr - mpayload
imaxalt = alti
altitude = alti
phi = 0

!               Initial calling of subroutines which set up mission parameters for simulation
Call Atmospheric_Model(alti, a_model, temperature, pressure, density)
qinf = 0.5*density*Vnew**2
Call AA_Model(calpha, thetanew, alpha)
Call Lift_Model(density, temperature, Vi, Sw, alpha, l_model, L)
Call Drag_Model(density, temperature, Vi, Sw, Ac, alpha, d_model, D)
Call Weight_Model(xold, yold, mv0, thetanew, Wx, Wy)
Call Thrust_Model(density, temperature, pressure, Vi, AcapoAc, Ac, Tt4, h, mv0, &
                &  v_model, newmass, T)
If(writeflag == 1)Then
                Call Write_Data(xold, yold, mv0, thetanew, alpha, T, L, D, Wx, Wy, time, &
                        & Vnew, temperature, pressure, density, altitude, qinf, accel, diffpe, &
                        & diffke, difffe, difflw, phi, dVx, dVy)
End If
Vold = Vi
oldmass = newmass
oldalt = altitude

!               Main Do-loop that executes mission from ground launch until the orbital altitude
!               is attained or until the total time specified in the input file is met
Do time = dt, totaltime, dt

!               Since the initial subroutines have been called, the differential parameters that
!               involve important variables such as Thrust, Drag, etc. are calculated
                dVx = (T - D - Wx)*(dt/newmass)
                dVy = (Wy - L)*(dt/newmass)
                accel = sqrt(dVx**2 + dVy**2)/dt
                dtheta = atan(dVy/(Vold + dVx))
                thetaold = thetanew
                thetanew = thetaold + dtheta

!               Checking to make sure the vehicle does not tip 20 degrees below the horizon after
!               each new calculation of theta
                If((thetanew-phi < (-110*pi/180.)) .or. (thetanew-phi > (110*pi/180.))) Then
                        Print *, "VEHICLE HAS TIPPED 20 DEGREES BELOW HORIZON!"
                        Print *, "Theta New Value =", thetanew
                        Print *, "Theta Old Value =", thetaold
```

```
                STOP
        End If

!       Calculation of velocity magnitude and altitude components (which are measured
!       from an Earth-fixed frame)
        Vnew = sqrt((Vold + dVx)**2 + dVy**2)
        xnew = xold + ((Vnew + Vold)/2)*(sin((thetaold + thetanew + 2*phi)/2))*dt
        ynew = yold + ((Vnew + Vold)/2)*(cos((thetaold + thetanew + 2*phi)/2))*dt
        xold = xnew
        yold = ynew

!       Altitude determination based on type of Earth-model and trajectory model used
!       Also calculation of the global vehicle angle phi
        If(e_model == 0) Then
                If(tr_model == 0) Then
                        altitude = ynew
                Else If(tr_model > 0) Then
                        altitude = sqrt(ynew**2 + xnew**2)
                End If
                phi = 0
        Else If(e_model == 1) Then
                If(tr_model == 0) Then
                        altitude = sqrt((Re + ynew)**2 + xnew**2) - Re
                Else If(tr_model > 0) Then
                        altitude = sqrt((Re + ynew)**2 + xnew**2) - Re
                End If
                If((xold > 0) .and. (Re + yold < 0))Then
                        phi = pi + atan(xold/(Re + yold))
                Else If((xold < 0) .and. (Re + yold < 0))Then
                        phi =pi + atan(xold/(Re + yold))
                Else If((xold < 0) .and. (Re + yold > 0))Then
                        phi = 2*pi+atan(xold/(Re + yold))
                Else
                        phi = atan(xold/(Re + yold))
                End If
        End If

!       Setting instantaneous max altitude and also checking to see if the vehicle
!       has dropped below 90% of the highest altitude attained
        If(altitude >= imaxalt) Then
                imaxalt = altitude
        Else If(altitude < 0.9*imaxalt) Then
                Print*, "DROPPED BELOW 90% OF HIGHEST ATTAINED ALTITUDE!"
                Print*, "CHECK OUTPUTS!"
                Print*, "ANGLE OF ATTACK MAY NOT BE LARGE ENOUGH!"
                Print*, "LAUNCH ANGLE MAY BE TOO SMALL!"
                Print*, "BURNOUT ALTITUDE MAY BE SET TOO LOW!"
                STOP
        End If

!       The following order of the subroutines was determined in order to propagate
!       the differential changes in the variables calculated at the beginning of the
!       Do-loop
        Call AA_model(calpha, thetanew, alpha)
        Call Atmospheric_Model(altitude, a_model, temperature, pressure, density)
        qinf = 0.5*density*Vnew**2
        Call Lift_Model(density, temperature, Vnew, Sw, alpha, l_model, L)
        Call Drag_Model(density, temperature, Vnew, Sw, Ac, alpha, d_model, D)
        oldmass = newmass
        Call Weight_Model(xnew, ynew, oldmass, thetanew, Wx, Wy)
        Call Thrust_Model(density, temperature, pressure, Vnew, AcapoAc, Ac, Tt4, &
                & h, oldmass, v_model, newmass, T)
        Call Lost_Work(diffpe, diffke, difffe, difflw)
        oldalt = altitude
        Vold = Vnew
        If(writeflag == 1) Then
                Call Write_Data(xnew, yold, newmass, thetanew, alpha, T, L, D, Wx, Wy, &
                        & time, Vnew, temperature, pressure, density, altitude, qinf, accel, &
                        & diffpe, diffke, difffe, difflw, phi, dVx, dVy)
        End If
```

```
!       Checking to determine whether or not the orbital altitude has been attained and if
!       so a sequence of calculations is done to determine orbital positioning criteria
        If(altitude > orbalt) Then
                Print *, "Congratulations, the", nint(orbalt), "m orbit altitude was reached!"
                Vcirc = sqrt(G*Me/(altitude + Re))
                deltaV = sqrt(Vnew**2 + Vcirc**2 - 2*Vnew*Vcirc*cos((pi/2.) - (thetanew-phi)))
                fmass = newmass*exp(-(abs(deltaV))/CF)
                fuelspent = fuelspent+newmass*(1-exp(-deltaV/CF))

!       Checking if enough mass was available to execute the necessary positioning delta V
                If(fmass < mpayload + remainmstr) Then
                        Print*, "However, orbit could not be circularized because the required propellant"
                        Print*, "to execute the delta V was greater than the remaining onboard propellant!"
                        Print*, "Propellant Deficit :", mpayload+remainmstr-fmass, "kg"
                Else
                        Print *, "Final Vehicle Mass (Payload + Unused Propellant + Retained Structural) =", fmass, "kg"
                End If

!       If writeflag == 1 then all of the mass data will be written to a file
                If(writeflag == 1) Then
                        If(fmass < mpayload + remainmstr) Then
                                Write(11, *) "Final Mass (using payload and remaining structure mass as propellant) =",
fmass, "kg"
                                Write(11, *) "Delta V Needed =", deltaV, "m/s"
                                Write(11, *) "Final Velocity =", Vcirc, "m/s"
                                Write(11, *) "Propellant Deficit :", mpayload+remainmstr-fmass, "kg"

!       Writes all of the multi-stage rocket mass information to a file
                                If(v_model == 2) Then
                                        Do i = 1, NRS
                                                Write(11, *) "Effective Exhaust Velocity (m/s) of Stage", i, ":",
CMSR(i)
                                                Write(11, *) "Total Mass (kg) of Stage", i, ":", MMSR(i)
                                                Write(11, *) "Structural Mass (kg) of Stage", i, ":", MSTRMSR(i)
                                                Write(11, *) "Propellant Mass (kg) of Stage", i, ":", MMSR(i) -
MSTRMSR(i)
                                                Write(11, *) "Mass Flow Rate of Propellant (kg/s) of Stage", i, ":",
MFMSR(i)
                                        End Do
                                        Deallocate(CMSR, MMSR, MFMSR, MSTRMSR)
                                End If
                                Write(11, *) "NO MASS FRACTION RESULTS SINCE CIRCULAR ORBIT WAS
NOT ESTABLISHED!"
                                Close(10)
                                Close(11)
                                STOP
                        End If
                        Write(11, *) "Final Mass (Payload + Unused Propellant + Retained Structural) =", fmass, "kg"
                        Write(11, *) "Delta V =", deltaV, "m/s"
                        Write(11, *) "Final Velocity =", Vcirc, "m/s"
                        Write(11, *) "Retained Structural Mass =", remainmstr, "kg"
                        Write(11, *) "Used Propellant Mass =", fuelspent
                        Write(11, *) "Unused Propellant Mass =", fmass-mpayload-remainmstr, "kg"
                        Write(11, *) "Payload Mass =", mpayload, "kg"
                        If(v_model == 2) Then
                                Do i = 1, NRS
                                        Write(11, *) "Effective Exhaust Velocity (m/s) of Stage", i, ":", CMSR(i)
                                        Write(11, *) "Total Mass (kg) of Stage", i, ":", MMSR(i)
                                        Write(11, *) "Structural Mass (kg) of Stage", i, ":", MSTRMSR(i)
                                        Write(11, *) "Propellant Mass (kg) of Stage", i, ":", MMSR(i) - MSTRMSR(i)
                                        Write(11, *) "Mass Flow Rate of Propellant (kg/s) of Stage", i, ":", MFMSR(i)
                                End Do
                                Deallocate(CMSR, MMSR, MFMSR, MSTRMSR)
                        End If

!       Vehicle mass fraction data is written to file
                        Write(11, *) "Lambda_P_(dV) =", dmkesum/mv0
                        Write(11, *) "Lambda_P_(dh) =", dmpesum/mv0
                        Write(11, *) "Lambda_P_(ds) =", dmlwsum/mv0
```

```
                        Write(11, *) "Lambda_P_Total_(=dV+dh+ds) =", fuelspent/mv0
                        Write(11, *) "Lambda_UP =", (fmass-mpayload-remainmstr)/mv0
                        Write(11, *) "Lambda_D =", lamd
                        Write(11, *) "Lambda_PL =", laml
                        Write(11, *) "SUM_(=1) =", fuelspent/mv0+lamd+laml+(fmass-mpayload-remainmstr)/mv0
                        Close(11)
                End If
                Close(10)
                STOP
        End If
End Do
Close(10)

!       If the total time runs out before the orbital altitude is attained all of the mass data
!       will be written to a file
If(v_model == 2) Then
        If(writeflag == 1) Then
                Do i = 1, NRS
                        Write(11, *) "Effective Exhaust Velocity (m/s) of Stage", i, ":", CMSR(i)
                        Write(11, *) "Total Mass (kg) of Stage", i, ":", MMSR(i)
                        Write(11, *) "Structural Mass (kg) of Stage", i, ":", MSTRMSR(i)
                        Write(11, *) "Propellant Mass (kg) of Stage", i, ":", MMSR(i) - MSTRMSR(i)
                        Write(11, *) "Mass Flow Rate of Propellant (kg/s) of Stage", i, ":", MFMSR(i)
                End Do
        End If
        Deallocate(CMSR, MMSR, MFMSR, MSTRMSR)
End If
Write(11, *) "Retained Structural Mass =", remainmstr, "kg"
Write(11, *) "Used Propellant Mass =", fuelspent
Write(11, *) "Unused Propellant =", newmass-mpayload-remainmstr, "kg"
Write(11, *) "Payload Mass =", mpayload, "kg"
If(writeflag == 1) Then
        Write(11, *) "Lambda_P_(dV) =", dmkesum/mv0
        Write(11, *) "Lambda_P_(dh) =", dmpesum/mv0
        Write(11, *) "Lambda_P_(ds) =", dmlwsum/mv0
        Write(11, *) "Lambda_P_Total_(=dV+dh+ds) =", fuelspent/mv0
        Write(11, *) "Lambda_UP =", (fmass-mpayload-remainmstr)/mv0
        Write(11, *) "Lambda_D =", lamd
        Write(11, *) "Lambda_PL =", laml
        Write(11, *) "SUM_(=1) =", fuelspent/mv0+lamd+laml+(fmass-mpayload-remainmstr)/mv0
        Close(11)
End If

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Contains

Subroutine AA_Model(constaoa, thetaaoa, aoa)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!       Name:  Tyler Winter
!       Description:  Routine to model the angle of attack variation during flight.
!       Subroutines Called:
!               Btbm(thetatbm, Mtbm, valuebeta)
!               Gravity_Model(alt, gmodel, galt)
!               MPMF(numpmf, newvaluempmf)
!       Inputs:
!               constaoa        -       Constant Angle of Attack Value (rad)
!               thetaaoa        -       Angle of Vehicle Horizon measured to the Vertical (rad)
!       Outputs:
!               aoa             -       Angle of Attack (rad)
!       External Parameters Used:
!               abalpha         -       Constant AB Wing Angle (deg)
!               abrtransalt     -       AB-R Transition Altitude (m)
!               abrtransM       -       AB-R Transition Mach Number (6-10)
!               absmwasf        -       Absolute Maximum Wing Angle Scale Factor
!                                       Wing alpha increases to this limit then snaps back to
!                                       maxrwalpha for coasting
!               alpha           -       Wing Angle of Attack (rad)
!               alpharr1        -       R-AB Rotational Rate of Wing (deg/s)
!               alpharr2        -       AB-R Rotational Rate of Wing (deg/s)
!               altitude        -       Altitude (m)
```

```
!           boalt            -     Burnout Altitude (m)
!           dac              -     Delta Alpha Correction (rad)
!           density          -     Free-stream Density (kg/m^3)
!           dt               -     Time Step (s)
!           gamma            -     Ratio of Specific Heats
!           g_model          -     Gravity Model (0 - sea level, 1 - g(alt))
!           ismootha1        -     R-AB Initial Smoothing Alpha Limit (deg)
!           l_model          -     Lift Model (0 - Small Angle, 1 - Oblique S/E)
!           maxalpha         -     Max Wing Angle (deg)
!           mrwalpha         -     R-Ballastic Transition Max Wing Angle (deg)
!           newmass          -     Mass of vehicle (kg)
!           phi              -     Global Vehicle Rotation Angle (rad)
!           pi               -     Ratio of a Circle's Circumference to its Diameter
!           rabtransalt      -     R-AB Transition Altitude (m)
!           Rgas             -     Gas Constant (J/kg-K)
!           smootha1         -     R-AB Smoothing Alpha Limit (deg)
!                                  1st leg transition angle value at which below this
!                                  value no additional smoothing is needed, if alpharr1
!                                  is greater than this value, smoothing is needed
!           smootha2         -     AB-R Smoothing Alpha Limit (deg)
!                                  2nd leg transition angle value at which below this
!                                  value no additional smoothing is needed, if alpharr2
!                                  is greater than this value, smoothing is needed
!           Sw               -     Wing Area (m^2)
!           temperature      -     Free-stream Temperature (K)
!           thetamax         -     Minimum Angle to Horizon (radians)
!           thetamin         -     Maximum Angle to Horizon (radians)
!           tr_model         -     Trajectory Model (0 - Const. Alt., 1 - Space Access)
!           Vnew             -     Free-stream Velocity (m/s)
!           v_model          -     Vehicle Model (0 - SSR, 1 - ARCC, 2 - MSR)
!     Internal Parameters Used:
!           aa               -     Counter for range of angles (deg)
!           alphaold         -     Wing Angle of Attack from previous time step (rad)
!           betaaoa          -     Oblique Shock Wave angle (rad)
!           countaoa         -     Flag variable to assist in proper flow of code
!           gaoa             -     Acceleration due to Gravity (m/s^2)
!           Laoa             -     Lift (N)
!           Minfaoa          -     Free-stream Mach Number
!           M2aoa            -     Downstream Mach Number
!           nu1aoa           -     Nu Value upstream of expansion (rad)
!           nu2aoa           -     Nu Value downstream of expansion (rad)
!           pflag            -     Flag that indicates what stage the vehicle is in
!                                  pflag = -1: initially, until the vehicle tips about
!                                              horizontal (89.5 < theta < 91.0)
!                                  pflag = 0: horizontal until vehicle reaches transition Mach
!                                  pflag = 1: rotation of wings/final rocket/coast phase
!           P2oP1aoa         -     Pressure Ratio of Upstream disturbance to top of wing
!           P3oP1aoa         -     Pressure Ratio of Upstream disturbance to bottom of wing
!           qinfaoa          -     Free-stream Dynamic Pressure (N/m^2)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Real(kind = 8), Intent(IN) :: constaoa, thetaaoa
Real(kind = 8), Intent(OUT) :: aoa

Real(kind = 8) :: Minfaoa, qinfaoa, gaoa, alphaold, aflag2 = 0, nu1aoa, nu2aoa
Real(kind = 8) :: Laoa, M2aoa, P2oP1aoa, P3oP1aoa, aa, betaaoa
Integer :: countaoa = 0, pflag = -1, aflag1 = 0

!     Each wing angle routine is separated first by trajectory model and then by
!     vehicle model.  For tr_model = 0, separation by vehicle model is not necessary
!     since it is constant altitude flight.  Instead, categorization by lift model is
!     more important since, as part of the routine the correct angle which propagates
!     level flight must be determined.
If(tr_model == 0) Then
      qinfaoa = 0.5*density*Vnew**2
      Minfaoa = Vnew/sqrt(gamma*Rgas*temperature)
      Call Gravity_Model(altitude, g_model, gaoa)

!     For l_model = 0, the appropriate wing angle is separated by ranges of Mach
```

```
!       numbers.  The angle is then determined by solving for an expression involving
!       level flight which enforces that lift = 0
        If(l_model == 0) Then
                If(Minfaoa < 0) Then
                        STOP "NEGATIVE MACH NUMBER!"
                Else If(Minfaoa < 0.3) Then
                        If(qinfaoa == 0) Then
                                aoa = 0.0
                        Else
                                aoa = (newmass*gaoa/(0.11*qinfaoa*Sw))*(pi/180.)
                        End If
                Else If((Minfaoa >= 0.3) .and. (Minfaoa < 0.7)) Then
                        If(qinfaoa == 0) Then
                                aoa = 0.0
                        Else
                                aoa = (newmass*gaoa*sqrt(1 - Minfaoa**2)/(0.11*qinfaoa*Sw))*(pi/180.)
                        End If
                Else If((Minfaoa >= 0.7) .and. (Minfaoa < 0.9999)) Then
!                       Print *, "Warning: Transonic Mach!"
                        If(qinfaoa == 0) Then
                                aoa = 0.0
                        Else
                                aoa = ((newmass*gaoa*sqrt(1 - Minfaoa**2)/(0.11*qinfaoa*Sw)) &
                                        & *(pi/180.)+newmass*gaoa*sqrt(abs(Minfaoa**2 - 1))/(4 &
                                        & *qinfaoa*Sw))/2.0
                        End If
                Else If((Minfaoa >= 0.9999) .and. (Minfaoa <= 1.0001)) Then
!                       Print *, "Warning: Sonic Mach!"
                        aoa = 5*(pi/180.)
                Else If((Minfaoa > 1.0001) .and. (Minfaoa < 1.3)) Then
!                       Print *, "Warning: Transonic Mach!"
                        If(qinfaoa == 0) Then
                                aoa = 0.0
                        Else
                                aoa = ((newmass*gaoa*sqrt(abs(1 - Minfaoa**2))/(0.11*qinfaoa &
                                        & *Sw))*(pi/180.) + newmass*gaoa*sqrt(Minfaoa**2 - 1)/(4 &
                                        & *qinfaoa*Sw))/2.0
                        End If
                Else If(Minfaoa >= 1.3) Then
                        If(qinfaoa == 0) Then
                                aoa = 0.0
                        Else
                                aoa = newmass*gaoa*sqrt(Minfaoa**2 - 1)/(4*qinfaoa*Sw)
                        End If
                End If
                If(aoa >= (maxalpha*(pi/180))) Then
                        Print*, "WING ANGLE TOO LARGE!  CONSIDER INCREASING AREA OF WING!"
                        STOP
                End If

!       For l_model = 1, this is similar to l_model = 0, except now when the Mach
!       number is greater than one, the angle is determined by solving an expression
!       that accounts for oblique shock and expansion waves while still enforcing that
!       lift = 0.
        Else If(l_model == 1) Then
                If(Minfaoa < 0) Then
                        STOP "NEGATIVE MACH NUMBER!"
                Else If(Minfaoa < 0.3) Then
                        If(qinfaoa == 0) Then
                                aoa = 0.0
                        Else
                                aoa = (newmass*gaoa/(0.11*qinfaoa*Sw))*(pi/180.)
                        End If
                Else If((Minfaoa >= 0.3) .and. (Minfaoa < 0.7)) Then
                        If(qinfaoa == 0) Then
                                aoa = 0.0
                        Else
                                aoa = (newmass*gaoa*sqrt(1 - Minfaoa**2)/(0.11*qinfaoa*Sw))*(pi/180.)
                        End If
                Else If((Minfaoa >= 0.7) .and. (Minfaoa < 0.9999)) Then
```

```
!                                       Print *, "Warning: Transonic Mach!"
                                    If(qinfaoa == 0) Then
                                        aoa = 0.0
                                    Else
                                        aoa = ((newmass*gaoa*sqrt(1 - Minfaoa**2)/(0.11*qinfaoa*Sw)) &
                                            & *(pi/180.)+newmass*gaoa*sqrt(abs(Minfaoa**2 - 1))/(4 &
                                            & *qinfaoa*Sw))/2.0
                                    End If
                                Else If((Minfaoa >= 0.9999) .and. (Minfaoa <= 1.0001)) Then
!                                       Print *, "Warning: Sonic Mach!"
                                    aoa = 5*(pi/180.)
                                Else If(Minfaoa > 1.0001) Then
                                    If(qinfaoa == 0) Then
                                        aoa = 0.0
                                    Else
                                        Do aa = 0.01, 45, 0.01
                                            aoa = aa*(pi/180.)
                                            nu1aoa = sqrt((gamma+1)/(gamma-1))*atan(sqrt(((gamma-1)/(gamma+1))* &
                                                & (Minfaoa**2-1)))-atan(sqrt(Minfaoa**2-1))
                                            nu2aoa = nu1aoa + aoa
                                            Call MPMF(nu2aoa, M2aoa)
                                            P2oP1aoa = (1./((1+(gamma-1)/2.*M2aoa**2)**(gamma/(gamma-1))))* &
                                                & (1+(gamma-1)/2.*Minfaoa**2)**(gamma/(gamma-1))
                                            Call Btbm(aoa, Minfaoa, betaaoa)
                                            P3oP1aoa = 1+(2*gamma/(gamma+1))*(Minfaoa**2*sin(betaaoa)**2 - 1)
                                            Laoa = pressure*(P3oP1aoa-P2oP1aoa)*Sw*cos(aoa)
                                            If(newmass*gaoa-Laoa < 0) Then
                                                Exit
                                            End If
                                        End Do
                                    End If
                                End If
                            End If

!                           For tr_model = 1, each angle routine is separated by the specific vehicle model.
                        Else If(tr_model == 1) Then

!                           For v_model = 0, the angle is determined for the rocket by essentially remaining
!                           constant (initially set value by user) until the vehicle noses up or down too
!                           much.  If this limit (i.e. thetamin, which is determined by LAmax) is violated,
!                           then the angle is adjusted up or down by the dac (delta angle correction) factor
!                           which is specified in the input file.
                            If(v_model == 0) Then
                                If(countaoa == 0) Then
                                    aoa = constaoa
                                    countaoa = countaoa + 1
                                Else If(countaoa /= 0) Then
                                    If(countaoa == 2) Then
                                        If(altitude < boalt) Then
                                            STOP "DROPPED BELOW BURNOUT ALTITUDE!"
                                        End If
                                    End If
                                    If(thetaaoa-phi < thetamin) Then
                                        aoa = alpha - dac
!                                           Print *, "MAX ANGLE TO HORIZON ACHIEVED, ADJUSTING WING
ANGLE!"
!                                           Print *, "NEW ALPHA = ", alpha*0.01
                                    Else If(thetaaoa-phi > thetamax) Then
                                        aoa = alpha + dac
!                                           Print *, "MIN ANGLE TO HORIZON ACHIEVED, ADJUSTING WING
ANGLE!"
!                                           Print *, "NEW ALPHA = ", aoa
                                    Else
                                        aoa = alpha
                                    End If
                                    If(altitude > boalt) Then
                                        If(countaoa == 1) Then
                                            countaoa = countaoa + 1
                                        End If
                                        aoa = 0.0
```

```
                        End If
                End If

!       For v_model = 1, the angle calculations for the RABCC still include the limiters
!       (thetamin/thetamax) that the rocket does but also a few other models.  The first
!       addition is that if the rocket-to-AB transition altitude has been reached and the
!       vehicle's nose is tipped up greater than 0.1 degrees from the horizon then a basic
!       smoothing method is used to gradually tip the vehicle's nose down to the horizon.
        Else If(v_model == 1) Then
                If(countaoa == 0) Then
                        aoa = constaoa
                        countaoa = countaoa + 1
                Else If(countaoa /= 0) Then
                        If(altitude <= rabtransalt) Then
                                If(countaoa == 3) Then
                                        Print *, "DROPPED BELOW TRANSITION ALTITUDE!"
                                        STOP
                                End If
                                If(countaoa > 1) Then
                                        If(thetaaoa-phi < thetamin) Then
                                                aoa = alpha - dac
!                                                       Print *, "MAX ANGLE TO HORIZON ACHIEVED,
ADJUSTING WING ANGLE!"
!                                                       Print *, "NEW ALPHA = ", alpha*0.01
                                        Else If(thetaaoa-phi > thetamax) Then
                                                aoa = alpha + dac
!                                                       Print *, "MIN ANGLE TO HORIZON ACHIEVED,
ADJUSTING WING ANGLE!"
!                                                       Print *, "NEW ALPHA = ", aoa
                                        Else
                                                aoa = alpha
                                        End If
                                End If
                                If(countaoa == 1) Then
                                        countaoa = countaoa + 1
                                End If
                        Else If((altitude > rabtransalt) .or. (countaoa >= 3)) Then
                                If(countaoa == 2) Then
                                        countaoa = countaoa + 1
                                End If
                                If((countaoa == 3) .and. (thetaaoa-phi <= 89.9*(pi/180))) Then
                                        If(aflag1 == 0) Then
                                                If(alpharr1 > ismootha1) Then
                                                        aoa = alpha + (-alpharr1*pi/180.)/100.
                                                        If(alpha >= alpharr1*(pi/180.)) Then
                                                                aflag1 = 1
                                                        End If
                                                Else If(alpharr1 <= ismootha1) Then
                                                        aoa = (-alpharr1)*(pi/180.)
                                                        aflag1 = 1
                                                End If
                                        Else
                                                aoa = alpha + (-alpharr1)*(pi/180.)
                                        End If
                                End If
                                qinfaoa = 0.5*density*Vnew**2
                                Minfaoa = Vnew/sqrt(gamma*Rgas*temperature)
                                Call Gravity_Model(altitude, g_model, gaoa)
                                If(pflag == 0) Then

!       Again, to account for the various permutations of trajectory, lift, and vehicle models
!       the following code was included.  This is basically the same as above with the addition
!       of a smoothing method introduced at the bottom of this main if-statement.
                                        If(l_model == 0) Then
                                                If(Minfaoa < 0) Then
                                                        STOP "NEGATIVE MACH NUMBER!"
                                                Else If(Minfaoa < 0.3) Then
                                                        If(qinfaoa == 0) Then
                                                                aoa = 0.0
                                                        Else
```

```
                                        aoa =
(newmass*gaoa/(0.11*qinfaoa*Sw))*(pi/180.)
                                    End If
                                Else If((Minfaoa >= 0.3) .and. (Minfaoa < 0.7)) Then
                                    If(qinfaoa == 0) Then
                                        aoa = 0.0
                                    Else
                                        aoa = (newmass*gaoa*sqrt(1 -
Minfaoa**2)/(0.11*qinfaoa*Sw))*(pi/180.)
                                    End If
                                Else If((Minfaoa >= 0.7) .and. (Minfaoa < 0.9999)) Then
!                                   Print *, "Warning: Transonic Mach!"
                                    If(qinfaoa == 0) Then
                                        aoa = 0.0
                                    Else
                                        aoa = ((newmass*gaoa*sqrt(1 -
Minfaoa**2)/(0.11*qinfaoa*Sw))*(pi/180.) + &
                                                &
newmass*gaoa*sqrt(abs(Minfaoa**2 - 1))/(4*qinfaoa*Sw))/2.0
                                    End If
                                Else If((Minfaoa >= 0.9999) .and. (Minfaoa <= 1.0001)) Then
!                                   Print *, "Warning: Sonic Mach!"
                                    aoa = 5*(pi/180.)
                                Else If((Minfaoa > 1.0001) .and. (Minfaoa < 1.3)) Then
!                                   Print *, "Warning: Transonic Mach!"
                                    If(qinfaoa == 0) Then
                                        aoa = 0.0
                                    Else
                                        aoa = ((newmass*gaoa*sqrt(abs(1 -
Minfaoa**2))/(0.11*qinfaoa*Sw))* &
                                                & (pi/180.) +
newmass*gaoa*sqrt(Minfaoa**2 - 1)/(4*qinfaoa*Sw))/2.0
                                    End If
                                Else If(Minfaoa >= 1.3) Then
                                    If(qinfaoa == 0) Then
                                        aoa = 0.0
                                    Else
                                        aoa = newmass*gaoa*sqrt(Minfaoa**2 -
1)/(4*qinfaoa*Sw)
                                    End If
                                End If
                                If((smootha1 < abs(aoa-alphaold)*(180./pi)) .and. (thetaaoa-phi <=
89.9* &
                                & (pi/180.))) Then
                                    aoa = alphaold + smootha1*(pi/180.)
                                End If
                                If(Minfaoa >= abrtransM) Then
                                    pflag = 1
                                End If
                                If(aoa >= (maxalpha*(pi/180))) Then
                                    Print*, "WING ANGLE TOO LARGE!  CONSIDER
INCREASING AREA OF WING!"
                                    STOP
                                End If

!       This next section is again similar to the code above, now for the AB.  It attempts
!       to enforce level flight for the acceleration leg of the mission, however, once the
!       AB-to-rocket transition Mach number is reached, the baseline limiters are reinforced
!       with an additional smoothing method used to gradually rotate the AB nose up until
!       the absmwasf*mrwalpha is reached, then it rotates down to just the mrwalpha so that
!       transitioning between the propulsion stages takes place smoothly (i.e. low g's).
                            Else If(l_model == 1) Then
                                If(Minfaoa < 0) Then
                                    STOP "NEGATIVE MACH NUMBER!"
                                Else If(Minfaoa < 0.3) Then
                                    If(qinfaoa == 0) Then
                                        aoa = 0.0
                                    Else
                                        aoa =
(newmass*gaoa/(0.11*qinfaoa*Sw))*(pi/180.)
```

```
                                                End If
                                        Else If((Minfaoa >= 0.3) .and. (Minfaoa < 0.7)) Then
                                                If(qinfaoa == 0) Then
                                                        aoa = 0.0
                                                Else
                                                        aoa = (newmass*gaoa*sqrt(1 -
Minfaoa**2)/(0.11*qinfaoa*Sw))*(pi/180.)
                                                End If
                                        Else If((Minfaoa >= 0.7) .and. (Minfaoa < 0.9999)) Then
!                                               Print *, "Warning: Transonic Mach!"
                                                If(qinfaoa == 0) Then
                                                        aoa = 0.0
                                                Else
                                                        aoa = ((newmass*gaoa*sqrt(1 -
                                                                &
Minfaoa**2)/(0.11*qinfaoa*Sw))*(pi/180.) + &
newmass*gaoa*sqrt(abs(Minfaoa**2 - 1))/(4*qinfaoa*Sw))/2.0
                                                End If
                                        Else If((Minfaoa >= 0.9999) .and. (Minfaoa <= 1.0001)) Then
!                                               Print *, "Warning: Sonic Mach!"
                                                aoa = 5*(pi/180.)
                                        Else If(Minfaoa > 1.0001) Then
                                                If(qinfaoa == 0) Then
                                                        aoa = 0.0
                                                Else
                                                        Do aa = 0.01, 45, 0.01
                                                                aoa = aa*(pi/180.)
                                                                nu1aoa =
sqrt((gamma+1)/(gamma-1))*atan(sqrt(((gamma-1)/(gamma+1))* &
                                                                        & (Minfaoa**2-1)))-
atan(sqrt(Minfaoa**2-1))
                                                                nu2aoa = nu1aoa + aoa
                                                                Call MPMF(nu2aoa, M2aoa)
                                                                P2oP1aoa = (1./((1+(gamma-
1)/2.*M2aoa**2)**(gamma/(gamma-1))))* &
                                                                        & (1+(gamma-
1)/2.*Minfaoa**2)**(gamma/(gamma-1))
                                                                Call Btbm(aoa, Minfaoa,
betaaoa)
                                                                P3oP1aoa =
1+(2*gamma/(gamma+1))*(Minfaoa**2*sin(betaaoa)**2 - 1)
                                                                Laoa =pressure*(P3oP1aoa-
P2oP1aoa)*Sw*cos(aoa)
                                                                If(newmass*gaoa-Laoa < 0)
Then
                                                                        Exit
                                                                End If
                                                        End Do
                                                End If
                                        End If
                                        If(Minfaoa >= abrtransM) Then
                                                pflag = 1
                                        End If
                                End If
                        Else If(pflag == 1) Then
                                If(thetaaoa-phi < thetamin) Then
                                        aoa = alpha - dac
!                                               Print *, "MAX ANGLE TO HORIZON ACHIEVED,
ADJUSTING WING ANGLE!"
!                                               Print *, "NEW ALPHA = ", alpha*0.01
                                Else If(thetaaoa-phi > thetamax) Then
                                        aoa = alpha + dac
!                                               Print *, "MIN ANGLE TO HORIZON ACHIEVED,
ADJUSTING WING ANGLE!"
!                                               Print *, "NEW ALPHA = ", aoa
                                Else
                                        If(aflag2 == 0) Then
                                                aoa = alpharr2*(pi/180.)
                                                If(smootha2 < abs(aoa-alphaold)*(180./pi)) Then
                                                        aoa = alphaold + smootha2*(pi/180.)
```

```
                                                                Else If(alpha > alpharr2*(pi/180.)) Then
                                                                        aflag2 = 1
                                                                End If
                                                                If(alpha >= absmwasf*mrwalpha*(pi/180.)) Then
                                                                        aflag2 = 1
                                                                End If
                                                        Else If(alpha < mrwalpha*(pi/180.)) Then
                                                                aoa = alpha + alpharr2*(pi/180.)
                                                        Else
                                                                aoa = mrwalpha*(pi/180.)
                                                        End If
                                                End If
                                                If(aflag2 >= 1) Then
                                                        If(aflag2 < 100./dt) Then
                                                                aoa = (1-
aflag2/(100./dt))*absmwasf*mrwalpha*pi/180.
                                                                If(aoa < mrwalpha*pi/180.) Then
                                                                        aoa = mrwalpha*pi/180.
                                                                End If
                                                                aflag2 = aflag2 + 1
                                                        Else
                                                                aoa = mrwalpha*pi/180.
                                                        End If
                                                End If
                                        End If
                                        If((pflag == -1) .and. ((thetaaoa-phi >= 89.5*(pi/180.)) .and. &
                                                & (thetaaoa-phi <= 91.0*(pi/180.)))) Then
                                                pflag = 0
                                                countaoa = countaoa + 1
                                        End If
                                End If
                        End If

!               For v_model = 2, this angle determination is identical to the SSR, however
!               it was kept separate in case future adjustments were to be made.
                Else If(v_model == 2) Then
                        If(countaoa == 0) Then
                                aoa = constaoa
                                countaoa = countaoa + 1
                        Else If(countaoa /= 0) Then
                                If(countaoa == 2) Then
                                        If(altitude < boalt) Then
                                                STOP "DROPPED BELOW BURNOUT ALTITUDE!"
                                        End If
                                End If
                                If(thetaaoa-phi < thetamin) Then
                                        aoa = alpha - dac
!                                               Print *, "MAX ANGLE TO HORIZON ACHIEVED, ADJUSTING WING
ANGLE!"
!                                               Print *, "NEW ALPHA = ", alpha*0.01
                                Else If(thetaaoa-phi > thetamax) Then
                                        aoa = alpha + dac
!                                               Print *, "MIN ANGLE TO HORIZON ACHIEVED, ADJUSTING WING
ANGLE!"
!                                               Print *, "NEW ALPHA = ", aoa
                                Else
                                        aoa = alpha
                                End If
                                If(altitude > boalt) Then
                                        If(countaoa == 1) Then
                                                countaoa = countaoa + 1
                                        End If
                                        aoa = 0.0
                                End If
                        End If
                End If

!               For tr_model = 2, the angle determination methods were kept the same for the
!               SSR and the MSR, however, now that the RABCC will be accelerating and climbing
!               the code for v_model = 1 was modified.
```

```
Else If(tr_model == 2) Then
        qinfaoa = 0.5*density*Vnew**2
        Minfaoa = Vnew/sqrt(gamma*Rgas*temperature)
        If(v_model == 0) Then
                If(countaoa == 0) Then
                        aoa = constaoa
                        countaoa = countaoa + 1
                Else If(countaoa /= 0) Then
                        If(countaoa == 2) Then
                                If(altitude < boalt) Then
                                        STOP "DROPPED BELOW BURNOUT ALTITUDE!"
                                End If
                        End If
                        If(thetaaoa-phi < thetamin) Then
                                aoa = alpha - dac
!                                       Print *, "MAX ANGLE TO HORIZON ACHIEVED, ADJUSTING WING
ANGLE!"
!                                       Print *, "NEW ALPHA = ", alpha*0.01
                        Else If(thetaaoa-phi > thetamax) Then
                                aoa = alpha + dac
!                                       Print *, "MIN ANGLE TO HORIZON ACHIEVED, ADJUSTING WING
ANGLE!"
!                                       Print *, "NEW ALPHA = ", aoa
                        Else
                                aoa = alpha
                        End If
                        If(altitude > boalt) Then
                                If(countaoa == 1) Then
                                        countaoa = countaoa + 1
                                End If
                                aoa = 0.0
                        End If
                End If

!       Much of the same basic angle models were maintained with the addition that
!       now the rocket-to-AB portion of the mission includes a rotation of the wing
!       angle until the abalpha is reached at which point this angle is fixed until
!       the accelerated climb ends.  When triggered by either the AB-to-rocket
!       transition Mach number or altitude being attained, a smoothing method is
!       again implemented.
        Else If(v_model == 1) Then
                If(countaoa == 0) Then
                        aoa = constaoa
                        countaoa = countaoa + 1
                Else If(countaoa /= 0) Then
                        If(altitude <= rabtransalt) Then
                                If(countaoa == 3) Then
                                        Print *, "DROPPED BELOW TRANSITION ALTITUDE!"
                                        STOP
                                End If
                                If(countaoa > 1) Then
                                        If(thetaaoa-phi < thetamin) Then
                                                aoa = alpha - dac
!                                                       Print *, "MAX ANGLE TO HORIZON ACHIEVED,
ADJUSTING WING ANGLE!"
!                                                       Print *, "NEW ALPHA = ", alpha*0.01
                                        Else If(thetaaoa-phi > thetamax) Then
                                                aoa = alpha + dac
!                                                       Print *, "MIN ANGLE TO HORIZON ACHIEVED,
ADJUSTING WING ANGLE!"
!                                                       Print *, "NEW ALPHA = ", aoa
                                        Else
                                                aoa = alpha
                                        End If
                                End If
                                If(countaoa == 1) Then
                                        countaoa = countaoa + 1
                                End If
                        Else If(((altitude > rabtransalt) .or. (countaoa >= 3)) .and. (altitude < abrtransalt)) Then
                                If(countaoa == 2) Then
```

```
                                        countaoa = countaoa + 1
                                    End If
                                    If(aoa < abalpha*(pi/180)) Then
                                        aoa = aoa + alpharr1*(pi/180)
                                    Else
                                        aoa = abalpha*(pi/180)
                                    End If
                                Else If((Minfaoa > abrtransM) .or. (altitude > abrtransalt)) Then
                                    If(thetaaoa-phi < thetamin) Then
                                        aoa = alpha - dac
!                                                   Print *, "MAX ANGLE TO HORIZON ACHIEVED,
ADJUSTING WING ANGLE!"
!                                                   Print *, "NEW ALPHA = ", alpha*0.01
                                    Else If(thetaaoa-phi > thetamax) Then
                                        aoa = alpha + dac
!                                                   Print *, "MIN ANGLE TO HORIZON ACHIEVED,
ADJUSTING WING ANGLE!"
!                                                   Print *, "NEW ALPHA = ", aoa
                                    Else
                                        If(aflag2 == 0) Then
                                            aoa = alpharr2*(pi/180.)
                                            If(smootha2 < abs(aoa-alphaold)*(180./pi)) Then
                                                aoa = alphaold + smootha2*(pi/180.)
                                            Else If(alpha > alpharr2*(pi/180.)) Then
                                                aflag2 = 1
                                            End If
                                            If(alpha >= absmwasf*mrwalpha*(pi/180.)) Then
                                                aflag2 = 1
                                            End If
                                        Else If(alpha < mrwalpha*(pi/180.)) Then
                                            aoa = alpha + alpharr2*(pi/180.)
                                        Else
                                            aoa = mrwalpha*(pi/180.)
                                        End If
                                    End If
                                    If(aflag2 >= 1) Then
                                        If(aflag2 < 100./dt) Then
                                            aoa = (1-aflag2/(100./dt))*absmwasf*mrwalpha*pi/180.
                                            If(aoa < mrwalpha*pi/180.) Then
                                                aoa = mrwalpha*pi/180.
                                            End If
                                            aflag2 = aflag2 + 1
                                        Else
                                            aoa = mrwalpha*pi/180.
                                        End If
                                    End If
                                End If
                            End If
                        Else If(v_model == 2) Then
                            If(countaoa == 0) Then
                                aoa = constaoa
                                countaoa = countaoa + 1
                            Else If(countaoa /= 0) Then
                                If(countaoa == 2) Then
                                    If(altitude < boalt) Then
                                        STOP "DROPPED BELOW BURNOUT ALTITUDE!"
                                    End If
                                End If
                                If(thetaaoa-phi < thetamin) Then
                                    aoa = alpha - dac
!                                               Print *, "MAX ANGLE TO HORIZON ACHIEVED, ADJUSTING WING
ANGLE!"
!                                               Print *, "NEW ALPHA = ", alpha*0.01
                                Else If(thetaaoa-phi > thetamax) Then
                                    aoa = alpha + dac
!                                               Print *, "MIN ANGLE TO HORIZON ACHIEVED, ADJUSTING WING
ANGLE!"
!                                               Print *, "NEW ALPHA = ", aoa
                                Else
                                    aoa = alpha
```

```
					End If
					If(altitude > boalt) Then
						If(countaoa == 1) Then
							countaoa = countaoa + 1
						End If
						aoa = 0.0
					End If
				End If
			End If
End If
If(aoa >= (maxalpha*(pi/180))) Then
	Print*, "WING ANGLE TOO LARGE!  CONSIDER INCREASING AREA OF WING!"
	STOP
End If
alphaold = aoa
End Subroutine AA_Model




Subroutine Atmospheric_Model(alt, amodel, Talt, Palt, Rhoalt)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!	Name:  Tyler Winter
!	Description:  Routine to determine atmospheric properties at a given altitude.
!	Subroutines Called:  None
!	Inputs:
!		alt		-	Altitude (m)
!		amodel		-	Atmosphere Model (0 - exponential, 1 - 7-part)
!	Outputs:
!		Palt		-	Pressure at Altitude (N/m^2)
!		Rhoalt		-	Density at Altitude (kg/m^3)
!		Talt		-	Temperature at Altitude (K)
!	External Parameters Used:
!		g0		-	Acceleration due to Gravity at Sea-level (m/s^2)
!		Psl		-	Pressure at Sea-level (N/m^2)
!		Rgas		-	Gas Constant (J/kg-K)
!		Rhosl		-	Density at Sea-level (kg/m^3)
!		Tsl		-	Temperature at Sea-level (K)
!	Internal Parameters Used:
!		alt1		-	Base Altitude Initialized for each Region (m)
!		a1		-	Slope of Temperature Gradient through Atmosphere (0-11km) (K/m)
!		a2		-	Slope of Temperature Gradient through Atmosphere (25-47km) (K/m)
!		a3		-	Slope of Temperature Gradient through Atmosphere (53-79km) (K/m)
!		a4		-	Slope of Temperature Gradient through Atmosphere (90-105km) (K/m)
!		P1		-	Base Pressure Initialized for each Region (N/m^2)
!		Rho1		-	Base Density Initialized for each Region (kg/m^3)
!		T1		-	Base Temperature Initialized for each Region (K)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Real(kind = 8), Intent(IN) :: alt
Integer, Intent(IN) :: amodel
Real(kind = 8), Intent(OUT) :: Talt, Palt, Rhoalt

Real(kind = 8), Parameter :: a1 = -6.5e-3, a2 = 3e-3, a3 = -4.5e-3, a4 = 4e-3
Real(kind = 8) :: P1, T1, Rho1, alt1

If(alt < 0) STOP "NEGATIVE ALTITUDE VALUE!"

!	For amodel = 0, the simple exponential modeling of the atmosphere is used.
If(amodel == 0) Then
	Palt = Psl*exp(-alt/7000)
	Talt = 230
	Rhoalt = Palt/(Rgas*Talt)

!	For amodel = 1, the 7-part atmospheric modeling is used which takes advantage
!	of the 4 temperature gradient regions separated by altitude.
Else If(amodel == 1) Then
	If(alt <= 11000) Then
		alt1 = 0
```

```
                T1 = Tsl
                P1 = Psl
                Rho1 = Rhosl
                Talt = T1 + a1*(alt - alt1)
                Palt = P1*(Talt/T1)**(-g0/(a1*Rgas))
                Rhoalt = Rho1*(Talt/T1)**((-g0/(a1*Rgas))-1.)
        Else If(alt <= 25000) Then
                alt1 = 11000
                Talt = 216.66
                P1 = 22700
                Rho1 = 0.3648
                Palt = P1*exp((-g0/(Rgas*Talt))*(alt - alt1))
                Rhoalt = Rho1*exp((-g0/(Rgas*Talt))*(alt - alt1))
        Else If(alt <= 47000) Then
                alt1 = 25000
                T1 = 216.66
                P1 = 2527.3
                Rho1 = 0.040639
                Talt = T1 + a2*(alt - alt1)
                Palt = P1*(Talt/T1)**(-g0/(a2*Rgas))
                Rhoalt = Rho1*(Talt/T1)**((-g0/(a2*Rgas))-1.)
        Else If(alt <= 53000) Then
                alt1 = 47000
                P1 = 125.58
                Rho1 = 0.0015535
                Talt = 282.66
                Palt = P1*exp((-g0/(Rgas*Talt))*(alt - alt1))
                Rhoalt = Rho1*exp((-g0/(Rgas*Talt))*(alt - alt1))
        Else If(alt <= 79000) Then
                alt1 = 53000
                T1 = 282.66
                P1 = 61.493
                Rho1 = 0.00075791
                Talt = T1 + a3*(alt - alt1)
                Palt = P1*(Talt/T1)**(-g0/(a3*Rgas))
                Rhoalt = Rho1*(Talt/T1)**((-g0/(a3*Rgas))-1.)
        Else If(alt <= 90000) Then
                alt1 = 79000
                P1 = 1.0623
                Rho1 = 0.0000223398
                Talt = 165.66
                Palt = P1*exp((-g0/(Rgas*Talt))*(alt - alt1))
                Rhoalt = Rho1*exp((-g0/(Rgas*Talt))*(alt - alt1))
        Else If(alt <= 105000) Then
                alt1 = 90000
                T1 = 165.66
                P1 = 0.109784
                Rho1 = 0.000002308744
                Talt = T1 + a4*(alt - alt1)
                Palt = P1*(Talt/T1)**(-g0/(a4*Rgas))
                Rhoalt = Rho1*(Talt/T1)**((-g0/(a4*Rgas))-1.)
        Else If(alt > 105000) Then
                Rhoalt = 0
                Palt = 0
        End If
End If

End Subroutine Atmospheric_Model



Subroutine Btbm(thetatbm, Mtbm, valuebeta)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!       Name:  Tyler Winter
!       Description:  Routine to determine the value of the shockwave angle, Beta, from
!               the Theta-Beta-Mach and Beta-Theta-Mach relations.
!       Subroutines Called:  None
!       Inputs:
!               Mtbm            -       Mach Number
```

```
!                       thetatbm                -               Theta Value (rad)
!               Outputs:
!                       valuebeta               -               Beta Value for given Mach Number and Theta (rad)
!               External Parameters Used:
!                       gamma                   -               Ratio of Specific Heats
!               Internal Parameters Used:
!                       betamaxtbm              -               Max Beta Value corresponding to Theta-Max Value (rad)
!                       betatbm                 -               Beta Value determined from Analytic Method (rad)
!                       chitbm                  -               Chi Value used for Analytic Method
!                       coefb                   -               B Coefficient for Theta-Max Quadratic
!                       coefc                   -               C Coefficient for Theta-Max Quadratic
!                       deltatbm                -               Delta Value for Analytic Method (0-strong, 1-weak solution)
!                       guesstbm                -               Initial Guess of Beta Value (30 degs for weak solution) (rad)
!                       lambdatbm               -               Lambda Value used for Analytic Method
!                       newvaluetbm             -               Beta Value determined from Iterative Method (rad)
!                       thetamaxtbm             -               Theta-Max Value determined Analytically (rad)
!                       x1sol                   -               1st Solution of Theta-Max Quadratic
!                       x2sol                   -               2nd Solution of Theta-Max Quadratic
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Real(kind = 8), Intent(IN) :: thetatbm, Mtbm
Real(kind = 8), Intent(OUT) :: valuebeta

Real(kind = 8) :: newvaluetbm, guesstbm, betatbm, thetamaxtbm, lambdatbm
Real(kind = 8) :: chitbm, coefb, coefc, x1sol, x2sol, betamaxtbm
Integer :: deltatbm

!               Initializing guess for iterative method and solving for thetamax and betamax
guesstbm = 0.523598783333333333333333333333333
coefb = -0.5 - 1./(2*gamma) + 2./(Mtbm**2*gamma)
coefc = -1./(2*Mtbm**2) - 1./(2*Mtbm**2*gamma) - 1./(Mtbm**4*gamma)
x1sol = (-coefb + sqrt(coefb**2 - 4*coefc))/2.
x2sol = (-coefb - sqrt(coefb**2 - 4*coefc))/2.
If(x1sol < 0) Then
        If(x2sol < 0) Then
                STOP "TWO NEGATIVE SOLUTIONS TO THETA-MAX QUADRATIC!"
        End If
        betamaxtbm = asin(sqrt(x2sol))
Else If(x2sol < 0) Then
        betamaxtbm = asin(sqrt(x1sol))
End If
thetamaxtbm = atan(2/tan(betamaxtbm)*((Mtbm**2*sin(betamaxtbm)**2-1)/(Mtbm**2* &
        & (gamma+cos(2*betamaxtbm))+2)))

!               Checks to see whether or not the current theta value is less than thetamax.
!               If it is, than an iterative method to determine beta is compared to an
!               analytical method.  If it is not, then the shock is detached and the beta
!               value is set to betamaxtbm.
If((thetatbm < thetamaxtbm) .and. (thetatbm > 0)) Then
        Do
                newvaluetbm = guesstbm - ((2*(1/tan(guesstbm))*((Mtbm**2*sin(guesstbm)**2 - 1) &
                        & /(Mtbm**2*(gamma + cos(2*guesstbm))+2)) - tan(thetatbm))/(((-2*(1/sin &
                        & (guesstbm)**2)*(Mtbm**2*sin(guesstbm)**2 - 1))/(Mtbm**2*(gamma+cos(2* &
                        & guesstbm))+2)) + (2/tan(guesstbm))*(((Mtbm**2*(gamma+cos(2*guesstbm)+ &
                        & 2))*Mtbm**2*sin(2*guesstbm)+2*Mtbm**2*sin(2*guesstbm)*(Mtbm**2*sin &
                        & (guesstbm)**2-1))/(Mtbm**2*(gamma+cos(2*guesstbm))+2)**2)))
                If(abs(newvaluetbm - guesstbm) < 0.00000000000001) Exit
                guesstbm = newvaluetbm
        End Do
        deltatbm = 1
        lambdatbm = sqrt((Mtbm**2-1)**2 - 3*(1+(gamma-1)/2*Mtbm**2)*(1+(gamma+1)/2* &
                & Mtbm**2)*tan(thetatbm)**2)
        chitbm = ((Mtbm**2-1)**3 - 9*(1+(gamma-1)/2*Mtbm**2)*(1+(gamma-1)/2*Mtbm**2 &
                & + (gamma+1)/4*Mtbm**4)*tan(thetatbm)**2)/(lambdatbm**3)
        betatbm = atan((Mtbm**2-1+2*lambdatbm*cos((4*pi*deltatbm+acos(chitbm))/3))/(3*(1+ &
                & (gamma-1)/2*Mtbm**2)*tan(thetatbm)))
        If(abs(newvaluetbm - betatbm)*(180./pi) < 0.1) Then
                valuebeta = (newvaluetbm + betatbm)/2.
        Else If(newvaluetbm < betatbm) Then
```

```
            valuebeta = newvaluetbm
        Else
            valuebeta = betatbm
        End If
Else If(thetatbm >= thetamaxtbm) Then
        Print*, "Detached Shock!", thetatbm*180./pi, thetamaxtbm*180./pi
        valuebeta = betamaxtbm
End If

End Subroutine Btbm



Subroutine Drag_Model(FSRhod, FSTd, FSVd, wingAd, crossAd, alphad, dmodel, drag)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!       Name:  Tyler Winter
!       Description:  Routine to determine drag generated for various vehicles.
!       Subroutines Called:
!               Btbm(thetatbm, Mtbm, valuebeta)
!               MPMF(numpmf, newvaluempmf)
!               PMF(pmfM, valuepmf)
!       Inputs:
!               alphad          -       Angle of Attack (radians)
!               crossAd         -       Cross-section Area of Body (m^2)
!               dmodel          -       Drag Model (0 - Basic, 1 - Improved)
!               FSRhod          -       Free-stream Density (kg/m^3)
!               FSTd            -       Free-stream Temperature (K)
!               FSVd            -       Free-stream Velocity (m/s)
!               wingAd          -       Area of Lifting Surface (m^2)
!       Outputs:
!               drag            -       Drag Value (N)
!       External Parameters Used:
!               ABbodyfactor    -       AB Body Factor (multiplied by Cdb)
!               altitude        -       Current Altitude (m)
!               Cdbody          -       Coefficient of Drag for Rocket Body
!               gamma           -       Ratio of Specific Heats
!               pi              -       Ratio of a Circle's Circumference to its Diameter
!               Rgas            -       Gas Constant (J/kg-K)
!               v_model         -       Vehicle Model (0 - SSR, 1 - ARCC, 2 - MSR)
!       Internal Parameters Used:
!               beta            -       Oblique Shock Wave Angle (rad)
!               betad           -       Oblique Shock Wave Angle (rad)
!               Cdb             -       Coefficient of Drag for Body
!               Cdw             -       Coefficient of Drag for Lifting Surface
!               Minfd           -       Free-stream Mach Number
!               M2d             -       Downstream Mach Number
!               nu1d            -       Nu Value upstream of expansion (rad)
!               nu2d            -       Nu Value downstream of expansion (rad)
!               P01oP1          -       Upstream Total Pressure to Static Pressure ratio
!               P02oP2          -       Top Total Pressure to Static Pressure ratio
!               P2oP1           -       Top Static Pressure to Upstream Static Pressure ratio
!               P3oP1           -       Bottom Static Pressure to Upstream Static Pressure ratio
!               qinfd           -       Free-stream Dynamic Pressure (N/m^2)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Real(kind = 8), Intent(IN) :: FSRhod, FSTd, FSVd, wingAd, crossAd, alphad
Integer, Intent(IN) :: dmodel
Real(kind = 8), Intent(OUT) :: drag

Real(kind = 8) :: Minfd, Cdw, Cdb, qinfd, beta, nu1d, nu2d, betad, M2d
Real(kind = 8) :: P01oP1, P02oP2, P2oP1, P3oP1

!       For dmodel == 0, the coefficients of drag due to the wing and the body are each
!       calculated (or set by user).  The coefficient of drag due to the wing utitlizes
!       simple models based on Mach number for supersonic flow.
If(dmodel == 0) Then
        If(altitude <= 105000) Then
            Minfd = FSVd/sqrt(gamma*Rgas*FSTd)
```

```
qinfd = 0.5*FSRhod*FSVd**2
If(Minfd < 0) Then
	STOP "NEGATIVE MACH NUMBER!"
Else If(Minfd < 0.3) Then
	Cdw = 0.008*alphad*(180./pi)
	If(v_model == 0) Then
		Cdb = Cdbody
	Else If(v_model == 1) Then
		If(abstage == 1) Then
			Cdb = Cdbody
		Else If(abstage == 2) Then
			Cdb = ABbodyfactor*Cdbody
		Else If(abstage == 3) Then
			Cdb = Cdbody
		End If
	Else If(v_model == 2) Then
		Cdb = Cdbody
	End If
Else If((Minfd >= 0.3) .and. (Minfd < 0.7)) Then
	Cdw = 0.008*alphad*(180./pi)
	If(v_model == 0) Then
		Cdb = Cdbody
	Else If(v_model == 1) Then
		If(abstage == 1) Then
			Cdb = Cdbody
		Else If(abstage == 2) Then
			Cdb = ABbodyfactor*Cdbody
		Else If(abstage == 3) Then
			Cdb = Cdbody
		End If
	Else If(v_model == 2) Then
		Cdb = Cdbody
	End If
Else If((Minfd >= 0.7) .and. (Minfd < 0.9999)) Then
!	Print *, "Warning: Transonic Mach!  Cdw = 0.2!  Minfd = ", Minfd
	Cdw = 0.2
	If(v_model == 0) Then
		Cdb = Cdbody
	Else If(v_model == 1) Then
		If(abstage == 1) Then
			Cdb = Cdbody
		Else If(abstage == 2) Then
			Cdb = ABbodyfactor*Cdbody
		Else If(abstage == 3) Then
			Cdb = Cdbody
		End If
	Else If(v_model == 2) Then
		Cdb = Cdbody
	End If
Else If((Minfd >= 0.9999) .and. (Minfd <= 1.0001)) Then
!	Print *, "Warning: Sonic Mach!  Cdw = 0.2!"
	Cdw = 0.2
	If(v_model == 0) Then
		Cdb = Cdbody
	Else If(v_model == 1) Then
		If(abstage == 1) Then
			Cdb = Cdbody
		Else If(abstage == 2) Then
			Cdb = ABbodyfactor*Cdbody
		Else If(abstage == 3) Then
			Cdb = Cdbody
		End If
	Else If(v_model == 2) Then
		Cdb = Cdbody
	End If
Else If((Minfd > 1.0001) .and. (Minfd < 1.3)) Then
!	Print *, "Warning: Transonic Mach!  Minfd = ", Minfd
	Cdw = 4*alphad**2/sqrt(Minfd**2 - 1)
	If(v_model == 0) Then
		Cdb = Cdbody
```

```
                                Else If(v_model == 1) Then
                                        If(abstage == 1) Then
                                                Cdb = Cdbody
                                        Else If(abstage == 2) Then
                                                Cdb = ABbodyfactor*Cdbody
                                        Else If(abstage == 3) Then
                                                Cdb = Cdbody
                                        End If
                                Else If(v_model == 2) Then
                                        Cdb = Cdbody
                                End If
                        Else If(Minfd >= 1.3) Then
                                Cdw = 4*alphad**2/sqrt(Minfd**2 - 1)
                                If(v_model == 0) Then
                                        Cdb = Cdbody
                                Else If(v_model == 1) Then
                                        If(abstage == 1) Then
                                                Cdb = Cdbody
                                        Else If(abstage == 2) Then
                                                Cdb = ABbodyfactor*Cdbody
                                        Else If(abstage == 3) Then
                                                Cdb = Cdbody
                                        End If
                                Else If(v_model == 2) Then
                                        Cdb = Cdbody
                                End If

!       For Mach greater than 4, the equation for Cdb is an approximation determined from
!       modeling the pressure drag from oblique shocks present on a two-dimensional wedge
!       shape, with a curve-fit for the shock angle, beta, for varying Mach numbers.
                                If(Minfd >= 4.0) Then
                                        beta = (10.5 + 274/(Minfd**2))*(pi/180)
                                        Cdb = (2/(gamma*Minfd**2))*(1+(2*gamma/(gamma+1))*((Minfd*sin(beta))**2-1))
                                        If(v_model == 1) Then
                                                If(abstage == 1) Then
                                                        Cdb = Cdbody
                                                Else If(abstage == 2) Then
                                                        Cdb = ABbodyfactor*Cdbody
                                                Else If(abstage == 3) Then
                                                        Cdb = Cdbody
                                                End If
                                        End If
                                End If
                        End If
                        If((alphad <= 0.00001) .and. (alphad >= -0.00001)) Cdw = 0.0
                        drag = qinfd*Cdw*wingAd + qinfd*Cdb*crossAd
                Else
                        drag = 0
                End If

!       For dmodel = 1, this model is similar to the above model for Mach less than 1.
!       However, when shock and expansion waves are present, they are taken into account
!       and the drag on the wing is calculated by determining the pressure ratios.
        Else If(dmodel == 1) Then
                If(altitude <= 105000) Then
                        Minfd = FSVd/sqrt(gamma*Rgas*FSTd)
                        qinfd = 0.5*FSRhod*FSVd**2
                        If(Minfd < 0) Then
                                STOP "NEGATIVE MACH NUMBER!"
                        Else If(Minfd < 0.3) Then
                                Cdw = 0.008*alphad*(180./pi)
                                If(v_model == 0) Then
                                        Cdb = Cdbody
                                Else If(v_model == 1) Then
                                        If(abstage == 1) Then
                                                Cdb = Cdbody
                                        Else If(abstage == 2) Then
                                                Cdb = ABbodyfactor*Cdbody
                                        Else If(abstage == 3) Then
                                                Cdb = Cdbody
```

```
                End If
            Else If(v_model == 2) Then
                Cdb = Cdbody
            End If
        Else If((Minfd >= 0.3) .and. (Minfd < 0.7)) Then
            Cdw = 0.008*alphad*(180./pi)
            If(v_model == 0) Then
                Cdb = Cdbody
            Else If(v_model == 1) Then
                If(abstage == 1) Then
                    Cdb = Cdbody
                Else If(abstage == 2) Then
                    Cdb = ABbodyfactor*Cdbody
                Else If(abstage == 3) Then
                    Cdb = Cdbody
                End If
            Else If(v_model == 2) Then
                Cdb = Cdbody
            End If
        Else If((Minfd >= 0.7) .and. (Minfd < 0.9999)) Then
!           Print *, "Warning: Transonic Mach!  Cdw = 0.2!  Minfd = ", Minfd
            Cdw = 0.2
            If(v_model == 0) Then
                Cdb = Cdbody
            Else If(v_model == 1) Then
                If(abstage == 1) Then
                    Cdb = Cdbody
                Else If(abstage == 2) Then
                    Cdb = ABbodyfactor*Cdbody
                Else If(abstage == 3) Then
                    Cdb = Cdbody
                End If
            Else If(v_model == 2) Then
                Cdb = Cdbody
            End If
        Else If((Minfd >= 0.9999) .and. (Minfd <= 1.1)) Then
!           Print *, "Warning: Sonic Mach!  Cdw = 0.2!"
            Cdw = 0.2
            If(v_model == 0) Then
                Cdb = Cdbody
            Else If(v_model == 1) Then
                If(abstage == 1) Then
                    Cdb = Cdbody
                Else If(abstage == 2) Then
                    Cdb = ABbodyfactor*Cdbody
                Else If(abstage == 3) Then
                    Cdb = Cdbody
                End If
            Else If(v_model == 2) Then
                Cdb = Cdbody
            End If
        Else If(Minfd > 1.1) Then
            If(v_model == 0) Then
                Cdb = Cdbody
            Else If(v_model == 1) Then
                If(abstage == 1) Then
                    Cdb = Cdbody
                Else If(abstage == 2) Then
                    Cdb = ABbodyfactor*Cdbody
                Else If(abstage == 3) Then
                    Cdb = Cdbody
                End If
            Else If(v_model == 2) Then
                Cdb = Cdbody
            End If
            If(alphad*(180./pi) > 0.00001) Then
                Call PMF(Minfd, nu1d)
                nu2d = alphad + nu1d
                Call MPMF(nu2d, M2d)
                P01oP1 = (1 + (gamma-1)/2*Minfd**2)**(gamma/(gamma-1))
```

```
                        P02oP2 = (1 + (gamma-1)/2*M2d**2)**(gamma/(gamma-1))
                        P2oP1 = (1/P02oP2)*(P01oP1)
                        Call Btbm(alphad, Minfd, betad)
                        P3oP1 = 1+(2*gamma/(gamma+1))*(Minfd**2*sin(betad)**2-1)
                        Cdw = (P3oP1 - P2oP1)*sin(alphad)/((gamma/2.)*Minfd**2)
                Else If(alphad*(180./pi) < -0.00001) Then
                        Call PMF(Minfd, nu1d)
                        nu2d = abs(alphad) + nu1d
                        Call MPMF(nu2d, M2d)
                        P01oP1 = (1 + (gamma-1)/2*Minfd**2)**(gamma/(gamma-1))
                        P02oP2 = (1 + (gamma-1)/2*M2d**2)**(gamma/(gamma-1))
                        P2oP1 = (1/P02oP2)*(P01oP1)
                        Call Btbm(abs(alphad), Minfd, betad)
                        P3oP1 = 1+(2*gamma/(gamma+1))*(Minfd**2*sin(betad)**2-1)
                        Cdw = (P3oP1 - P2oP1)*sin(abs(alphad))/((gamma/2.)*Minfd**2)
                End If
        End If
        If((alphad*(180./pi) <= 0.00001) .and. (alphad*(180./pi) >= -0.00001)) Cdw = 0.0
        drag = qinfd*Cdw*wingAd + qinfd*Cdb*crossAd
    Else
        drag = 0
    End If
End If

End Subroutine Drag_Model



Subroutine Gravity_Model(alt, gmodel, galt)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!       Name:  Tyler Winter
!       Description:  Routine to model the acceleration due to gravity.
!       Subroutines Called:  None
!       Inputs:
!               alt         -       Current Altitude (m)
!               gmodel      -       Gravity Model (0 - sea level, 1 - g(alt))
!       Outputs:
!               galt        -       Acceleration due to Gravity at Altitude (m/s^2)
!       External Parameters Used:
!               g0          -       Acceleration due to Gravity at Sea-level (m/s^2)
!               Re          -       Radius of Earth (m)
!       Internal Parameters Used:  None
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Real(kind = 8), Intent(IN) :: alt
Integer, Intent(IN) :: gmodel
Real(kind = 8), Intent(OUT) :: galt

If(alt < 0) STOP "NEGATIVE ALTITUDE VALUE!"

!       This model keeps the acceleration due to gravity fixed at the sea-level value.
If(gmodel == 0) Then
        galt = g0

!       This model takes into account the variation with altitude.
Else If(gmodel == 1) Then
        galt = g0*(Re/(Re + alt))**2
End If

End Subroutine Gravity_Model



Subroutine Lift_Model(FSRhol, FSTl, FSVl, wingAl, alphal, lmodel, lift)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!       Name:  Tyler Winter
!       Description:  Routine to determine the lift generated for various vehicles.
```

```
!		Subroutines Called:
!			Btbm(thetatbm, Mtbm, valuebeta)
!			MPMF(numpmf, newvaluempmf)
!			PMF(pmfM, valuepmf)
!		Inputs:
!			FSRhol	-		Free-stream Density (kg/m^3)
!			FSTl	-		Free-stream Temperature (K)
!			FSVl	-		Free-stream Velocity (m/s)
!			wingAl	-		Area of Lifting Surface (m^2)
!			alphal	-		Angle of Attack (radians)
!			lmodel	-		Lift Model (0 - Small Angle, 1 - Oblique S/E)
!		Outputs:
!			lift	-		Lift Value (N)
!		External Parameters Used:
!			altitude-	Current Altitude (m)
!			gamma	-		Ratio of Specific Heats
!			Rgas	-		Gas Constant (J/kg-K)
!			pi	-		Ratio of a Circle's Circumference to its Diameter
!		Internal Parameters Used:
!			betal	-		Oblique Shock Wave Angle (rad)
!			Clw	-		Coefficient of Lift for Lifting Surface
!			Minfl	-		Free-stream Mach Number
!			M2l	-		Downstream Mach Number
!			nu1l	-		Nu Value upstream of expansion (rad)
!			nu2l	-		Nu Value downstream of expansion (rad)
!			P01oP1	-		Upstream Total Pressure to Static Pressure ratio
!			P02oP2	-		Top Total Pressure to Static Pressure ratio
!			P2oP1	-		Top Static Pressure to Upstream Static Pressure ratio
!			P3oP1	-		Bottom Static Pressure to Upstream Static Pressure ratio
!			qinfl	-		Free-stream Dynamic Pressure (N/m^2)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Real(kind = 8), Intent(IN) :: FSRhol, FSTl, FSVl, wingAl, alphal
Integer, Intent(IN) :: lmodel
Real(kind = 8), Intent(OUT) :: lift

Real(kind = 8) :: Minfl, Clw, qinfl, nu1l, nu2l, betal, P2oP1, P01oP1, P3oP1
Real(kind = 8) :: P02oP2, M2l

!		For lmodel = 0, a basic approximate model from linearized theory is used to
!		determine the coefficient of lift due to the wing.
If(lmodel == 0) Then
		If(altitude <= 105000) Then
			Minfl = FSVl/sqrt(gamma*Rgas*FSTl)
			qinfl = 0.5*FSRhol*FSVl**2
			If(Minfl < 0) Then
				STOP "NEGATIVE MACH NUMBER!"
			Else If(Minfl < 0.3) Then
				Clw = 0.11*alphal*(180./pi)
			Else If((Minfl >= 0.3) .and. (Minfl < 0.7)) Then
				Clw = 0.11*alphal*(180./pi)/sqrt(1 - Minfl**2)
			Else If((Minfl >= 0.7) .and. (Minfl < 0.9999)) Then
!				Print *, "Warning: Transonic Mach!"
				Clw = 0.11*alphal*(180./pi)/sqrt(1 - Minfl**2)
			Else If((Minfl >= 0.9999) .and. (Minfl <= 1.0001)) Then
!				Print *, "Warning: Sonic Mach!  Clw = 0.0!"
				Clw = 0.0
			Else If((Minfl > 1.0001) .and. (Minfl < 1.3)) Then
!				Print *, "Warning: Transonic Mach!"
				Clw = 0.11*alphal*(180./pi)/sqrt(Minfl**2 - 1)
			Else If(Minfl >= 1.3) Then
				Clw = 4*alphal/sqrt(Minfl**2 - 1)
			End If
			lift = qinfl*Clw*wingAl
		Else
			lift = 0
		End If

!		For lmodel = 1, the same models are used here as well for Mach numbers less
```

```
!		than 1.  However, when shock and expansion waves are present, they are
!		taken into account and the lift on the wing is calculated by determining
!		the pressure ratios.
Else If(lmodel == 1) Then
		If(altitude <= 105000) Then
				Minfl = FSVl/sqrt(gamma*Rgas*FSTl)
				qinfl = 0.5*FSRhol*FSVl**2
				If(Minfl < 0) Then
						STOP "NEGATIVE MACH NUMBER!"
				Else If(Minfl < 0.3) Then
						Clw = 0.11*alphal*(180./pi)
				Else If((Minfl >= 0.3) .and. (Minfl < 0.7)) Then
						Clw = 0.11*alphal*(180./pi)/sqrt(1 - Minfl**2)
				Else If((Minfl >= 0.7) .and. (Minfl < 0.9999)) Then
!						Print *, "Warning: Transonic Mach!"
						Clw = 0.11*alphal*(180./pi)/sqrt(1 - Minfl**2)
				Else If((Minfl >= 0.9999) .and. (Minfl <= 1.1)) Then
!						Print *, "Warning: Sonic Mach!  Clw = 0.0!"
						Clw = 0.0
				Else If(Minfl > 1.1) Then
						If(alphal*(180./pi) > 0.00001) Then
								Call PMF(Minfl, nu1l)
								nu2l = alphal + nu1l
								Call MPMF(nu2l, M2l)
								P01oP1 = (1 + (gamma-1)/2*Minfl**2)**(gamma/(gamma-1))
								P02oP2 = (1 + (gamma-1)/2*M2l**2)**(gamma/(gamma-1))
								P2oP1 = (1/P02oP2)*(P01oP1)
								Call Btbm(alphal, Minfl, betal)
								P3oP1 = 1+(2*gamma/(gamma+1))*(Minfl**2*sin(betal)**2-1)
								Clw = (P3oP1 - P2oP1)*cos(alphal)/((gamma/2.)*Minfl**2)
						Else If(alphal*(180./pi) < -0.00001) Then
								Call PMF(Minfl, nu1l)
								nu2l = abs(alphal) + nu1l
								Call MPMF(nu2l, M2l)
								P01oP1 = (1 + (gamma-1)/2*Minfl**2)**(gamma/(gamma-1))
								P02oP2 = (1 + (gamma-1)/2*M2l**2)**(gamma/(gamma-1))
								P2oP1 = (1/P02oP2)*(P01oP1)
								Call Btbm(abs(alphal), Minfl, betal)
								P3oP1 = 1+(2*gamma/(gamma+1))*(Minfl**2*sin(betal)**2-1)
								Clw = (P2oP1 - P3oP1)*cos(abs(alphal))/((gamma/2.)*Minfl**2)
						End If
				End If
				If((alphal*(180./pi) <= 0.00001) .and. (alphal*(180./pi) >= -0.00001)) Clw = 0.0
				lift = qinfl*Clw*wingAl
		Else
				lift = 0
		End If
End If

End Subroutine Lift_Model



Subroutine Lost_Work(deltape, deltake, deltafe, deltalw)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!		Name:  Tyler Winter
!		Description:  Routine to determine the cumulative lost work associated with all
!				irreversibilities occurring during the mission including the vehicle wake.
!				This routine also calculates the various vehicle mass fractions to be used
!				for analysis.
!		Subroutines Called:
!				Gravity_Model(alt, gmodel, galt)
!		Inputs:  None
!		Outputs:
!				deltafe		-		Running sum of energy content of expended fuel (J)
!				deltake		-		Running sum of kinetic energy (J)
!				deltalw		-		Running sum of lost work (J)
!				deltape		-		Running sum of potential energy (J)
!		External Parameters Used:
```

```
!               abstage         -       Current Propulsion Stage for RABCC
!               altitude        -       Current Altitude (m)
!               CF              -       Final Rocket Effective Exhaust Velocity (m/s)
!               CMSR            -       Effective Exhaust Velocities for each stage of the MSR (m/s)
!               C1              -       Main Rocket Effective Exhaust Velocity (m/s)
!               dmkesum         -       Running sum of diff. prop. mass for kinetic energy (kg)
!               dmlwsum         -       Running sum of diff. prop. mass for lost work (kg)
!               dmpesum         -       Running sum of diff. prop. mass for potential energy (kg)
!               G               -       Gravitational Constant (m^3/kg-s^2)
!               g_model         -       Gravity Model for g (0 - Sea-level, 1 - g(alt))
!               h               -       Heating Value of Fuel (J/kg)
!               lamd            -       Structural Vehicle Mass Fraction
!               laml            -       Payload Vehicle Mass Fraction
!               Me              -       Mass of Earth (kg)
!               mpayload        -       Payload Mass (kg)
!               mstr            -       Structural (System) Mass (kg)
!               mv0             -       Initial Total Vehicle Mass (including propellant) (kg)
!               newmass         -       Current Total Vehicle Mass (kg)
!               NRS             -       Number of rocket stages
!               oldalt          -       Altitude from previous time step (m)
!               oldmass         -       Total Vehicle Mass from previous time step (kg)
!               orbalt          -       Orbital Altitude (m)
!               phi             -       Global Vehicle Rotation Angle (rad)
!               pi              -       Ratio of a Circle's Circumference to its Diameter
!               rae             -       Rocket Accelerator Efficiency
!               Re              -       Radius of Earth (m)
!               rstage          -       Current rocket stage
!               thetanew        -       Current Vehicle Angle measured from the vertical (rad)
!               Vnew            -       Current Vehicle Velocity (m/s)
!               Vold            -       Vehicle Velocity from previous time step (m/s)
!               v_model         -       Vehicle Model (0 - SSR, 1 - ARCC, 2 - MSR)
!       Internal Parameters Used:
!               delV            -       Change in Velocity required to attain orbit (m/s)
!               dfe             -       Differential Change in energy content of expended fuel (J)
!               dfesum          -       Running sum of energy content of expended fuel (J)
!               dke             -       Differential Change in kinetic energy (J)
!               dkesum          -       Running sum of kinetic energy (J)
!               dmke            -       Diff. change in prop. mass for kinetic energy (kg)
!               dmlw            -       Diff. change in prop. mass for lost work (kg)
!               dmpe            -       Diff. change in prop. mass for potential energy (kg)
!               dmpu            -       Diff. change in prop. mass (kg)
!               dpe             -       Differential Change in potential energy (J)
!               dpesum          -       Running sum of potential energy (J)
!               glw             -       Acceleration due to Gravity at Altitude (m/s^2)
!               oldstage        -       Rocket stage from last time step
!               Vc              -       Velocity of vehicle in circular orbit (m/s)
!               stagechange     -       Determines if rocket stage transitioned
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Real(kind = 8), Intent(OUT) :: deltape, deltake, deltafe, deltalw

Real(kind = 8) :: dpesum = 0, dkesum = 0, dfesum = 0, glw, dpe, dke, dfe, Vc, delV
Real(kind = 8) :: dmke, dmpe, dmlw, dmpu
Integer :: oldstage = 1, stagechange

!       For v_model = 0, the differentials are calculated for the rocket.  One half the
!       effective exhaust velocity squared is used for the h-values for determining dfe.
!       Notice that the initial rocket's effective exhaust velocity is C1 and the final
!       rocket's effective exhaust velocity (for orbit positioning) is CF.
If(v_model == 0) Then
        Call Gravity_Model(altitude, g_model, glw)
        dpe = newmass*glw*(altitude - oldalt)
        dke = (newmass*Vnew**2-oldmass*Vold**2)/2
        dfe = rae*(C1**2/2)*(oldmass-newmass)
        dmke = oldmass*(Vnew**2-Vold**2)/(C1**2+Vnew**2)
        dmpe = 2*dpe/(C1**2+Vnew**2)
        dmpu = oldmass-newmass
        If(altitude > orbalt) Then
                Vc = sqrt(G*Me/(altitude + Re))
```

```
                    delV = sqrt(Vnew**2 + Vc**2 - 2*Vnew*Vc*cos((pi/2.) - (thetanew-phi)))
                    dke = dke + (newmass*exp(-(abs(delV))/CF)*Vc**2 - newmass*Vnew**2)/2
                    dfe = dfe + rae*(CF**2/2)*(newmass - newmass*exp(-(abs(delV))/CF))
                    dmke = dmke + newmass*(Vc**2-Vnew**2)/(CF**2+Vc**2)
                    dmpu = dmpu + newmass*(1-exp(-delV/CF))
                    laml = mpayload/mv0
                    lamd = mstr/mv0
            End If
            dmlw = dmpu-dmpe-dmke
            dpesum = dpesum + dpe
            dkesum = dkesum + dke
            dfesum = dfesum + dfe
            deltape = dpesum
            deltake = dkesum
            deltafe = dfesum
            deltalw = deltafe - deltake - deltape
            dmkesum = dmkesum + dmke
            dmpesum = dmpesum + dmpe
            dmlwsum = dmlwsum + dmlw

!           For v_model = 1, the differentials are calculated for the RABCC.  One half the
!           effective exhaust velocity squared is used for the h-values for determining dfe
!           for the initial and final rocket segments.  Again, notice that the initial
!           rocket's effective exhaust velocity is C1 and the final rocket's effective
!           exhaust velocity (for orbit positioning) is CF.
Else If(v_model == 1) Then
            Call Gravity_Model(altitude, g_model, glw)
            dpe = newmass*glw*(altitude - oldalt)
            dke = (newmass*Vnew**2 - oldmass*Vold**2)/2
            dmpu = oldmass-newmass
            If(abstage == 1) Then
                    dfe = rae*(C1**2/2)*(oldmass - newmass)
                    dmke = oldmass*(Vnew**2-Vold**2)/(C1**2+Vnew**2)
                    dmpe = 2*dpe/(C1**2+Vnew**2)
            Else If(abstage == 2) Then
                    dfe = h*(oldmass - newmass)
                    dmke = oldmass*(Vnew**2-Vold**2)/(h+Vnew**2)
                    dmpe = 2*dpe/(h+Vnew**2)
            Else If(abstage == 3) Then
                    dfe = rae*(CF**2/2)*(oldmass - newmass)
                    dmke = oldmass*(Vnew**2-Vold**2)/(CF**2+Vnew**2)
                    dmpe = 2*dpe/(CF**2+Vnew**2)
            End If
            If(altitude > orbalt) Then
                    Vc = sqrt(G*Me/(altitude + Re))
                    delV = sqrt(Vnew**2 + Vc**2 - 2*Vnew*Vc*cos((pi/2.) - (thetanew-phi)))
                    dke = dke + (newmass*exp(-(abs(delV))/CF)*Vc**2 - newmass*Vnew**2)/2
                    dfe = dfe + rae*(CF**2/2)*(newmass - newmass*exp(-(abs(delV))/CF))
                    dmke = dmke + newmass*(Vc**2-Vnew**2)/(CF**2+Vc**2)
                    dmpu = dmpu + newmass*(1-exp(-delV/CF))
                    laml = mpayload/mv0
                    lamd = mstr/mv0
            End If
            dmlw = dmpu-dmpe-dmke
            dpesum = dpesum + dpe
            dkesum = dkesum + dke
            dfesum = dfesum + dfe
            deltape = dpesum
            deltake = dkesum
            deltafe = dfesum
            deltalw = deltafe - deltake - deltape
            dmkesum = dmkesum + dmke
            dmpesum = dmpesum + dmpe
            dmlwsum = dmlwsum + dmlw

!           For v_model = 2, the differentials are calculated for the MSR.  One half the
!           effective exhaust velocity squared is used for the h-values for determining dfe.
!           Notice that now the rocket's effective exhaust velocites are determined by the
!           user inputs (CMSR values).  Also, the final  rocket's effective exhaust velocity
!           (for orbit positioning) is CF.
```

```
Else If(v_model == 2) Then
        stagechange = rstage - oldstage
        Call Gravity_Model(altitude, g_model, glw)
        dpe = newmass*glw*(altitude - oldalt)
        dke = (newmass*Vnew**2-oldmass*Vold**2)/2
        dmpu = oldmass-newmass

!       This conditional statement is to account for the structural stage mass drop
        If(stagechange > 0) Then
                dmpu = dmpu - MSTRMSR(oldstage)
                dke = ((newmass+MSTRMSR(oldstage))*Vnew**2-oldmass*Vold**2)/2
        End If
        If(rstage <= NRS) Then
                dfe = rae*(CMSR(rstage)**2/2)*(oldmass-newmass)
                dmke = oldmass*(Vnew**2-Vold**2)/(CMSR(rstage)**2+Vnew**2)
                dmpe = 2*dpe/(CMSR(rstage)**2+Vnew**2)
        End If
        If((rstage > NRS) .and. (altitude < boalt)) Then
                dfe = 0
                dmke = 0
                dmpe = 0
        End If
        If(altitude > orbalt) Then
                Vc = sqrt(G*Me/(altitude + Re))
                delV = sqrt(Vnew**2 + Vc**2 - 2*Vnew*Vc*cos((pi/2.) - (thetanew-phi)))
                dke = dke + (newmass*exp(-(abs(delV))/CF)*Vc**2 - newmass*Vnew**2)/2
                dfe = dfe + rae*(CF**2/2)*(newmass - newmass*exp(-(abs(delV))/CF))
                dmke = dmke + newmass*(Vc**2-Vnew**2)/(CF**2+Vc**2)
                dmpu = dmpu + newmass*(1-exp(-delV/CF))
                laml = mpayload/mv0
                lamd = mstr/mv0
        End If
        dmlw = dmpu-dmpe-dmke
        dpesum = dpesum + dpe
        dkesum = dkesum + dke
        dfesum = dfesum + dfe
        deltape = dpesum
        deltake = dkesum
        deltafe = dfesum
        deltalw = deltafe - deltake - deltape
        dmkesum = dmkesum + dmke
        dmpesum = dmpesum + dmpe
        dmlwsum = dmlwsum + dmlw
End If

oldstage = rstage
End Subroutine Lost_Work




Subroutine MPMF(numpmf, newvaluempmf)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!       Name:  Tyler Winter
!       Description:  Routine to determine the Mach number using exponential and loga-
!               rithmic curve fits of the Prandtl-Meyer function with gamma = 1.4.
!       Subroutines Called:  None
!       Inputs:
!               numpmf          -       Nu Value for Mach Number being determined (rad)
!       Outputs:
!               newvaluempmf    -       Mach Number for given Nu Value
!       External Parameters Used:
!               pi              -       Ratio of a Circle's Circumference to its Diameter
!       Internal Parameters Used:
!               nud             -       Nu Value in Degrees
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Real(kind = 8), Intent(IN) :: numpmf
Real(kind = 8), Intent(OUT) :: newvaluempmf
```

```
Real(kind = 8) :: nud

nud = numpmf*(180./pi)

If((nud < 0) .or. (nud > 130.45)) Then
        Print*, "NU VALUE NEGATIVE OR GREATER THAN 130.45 deg!!"
        STOP
End If

!       Determines value of Mach number from Prandtl-Meyer function with gamma = 1.4
If((nud > 0) .and. (nud <= 76.9202)) Then
        newvaluempmf = 1.169*exp(0.0189*nud)
Else If((nud > 76.9202) .and. (nud < 102.3162)) Then
        newvaluempmf = 0.5756*exp(0.0276*nud)
Else If((nud >= 102.3162) .and. (nud <= 111.5090)) Then
        newvaluempmf = 0.1049*exp(0.0444*nud)
Else If((nud > 111.5090) .and. (nud < 116.1952)) Then
        newvaluempmf = 0.0154*exp(0.0616*nud)
Else If((nud >= 116.1952) .and. (nud < 119.0283)) Then
        newvaluempmf = 0.0021*exp(0.079*nud)
Else If((nud >= 119.0283) .and. (nud < 120.9242)) Then
        newvaluempmf = 317.4*log(nud) - 1492.1
Else If(nud >= 120.9242) Then
        Print*, "MACH 30 OR GREATER!!"
        newvaluempmf = 317.4*log(nud) - 1492.1
        Print*, nud, newvaluempmf
End If

End Subroutine MPMF




Subroutine PMF(pmfM, valuepmf)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!       Name:  Tyler Winter
!       Description:  Routine to determine the value of the Prandtl-Meyer function.
!       Subroutines Called:  None
!       Inputs:
!               pmfM            -       Free-stream Mach Number
!       Outputs:
!               valuepmf        -       Nu Value for given Mach Number (rad)
!       External Parameters Used:
!               gamma           -       Ratio of Specific Heats
!       Internal Parameters Used:  None
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Real(kind = 8), Intent(IN) :: pmfM
Real(kind = 8), Intent(OUT) :: valuepmf

valuepmf = sqrt((gamma+1)/(gamma-1))*atan(sqrt(((gamma-1)/(gamma+1))*(pmfM**2-1))) &
        & -atan(sqrt(pmfM**2-1))

End Subroutine PMF




Subroutine Thrust_Model(FSRhot, FSTt, FSPt, FSVt, Aratio, crossA, Tott4, hvf, &
        & omass, vmodel, nmass, thrust)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!       Name:  Tyler Winter
!       Description:  Routine to determine the thrust generated by various vehicles.
!               Mass accounting also takes place in this routine.
!       Subroutines Called:  None
!       Inputs:
!               Aratio          -       Ratio of Capture Area to Cross-sectional Area
!               crossA          -       Cross-sectional Area of Vehicle (m^2)
```

```
!                       FSRhot                  -               Free-stream Density (kg/m^3)
!                       FSTt                    -               Free-stream Temperature (K)
!                       FSPt                    -               Free-stream Pressure (N/m^2)
!                       FSVt                    -               Free-stream Velocity (m/s)
!                       hvf                     -               Heating Value of Fuel (J/kg)
!                       omass                   -               Old Total Vehicle Mass (kg)
!                       Tott4                   -               Total Temperature at Combustor Exit (K)
!                       vmodel                  -               Vehicle Model (0 - SSR, 1 - ARCC, 2 - MSR)
!               Outputs:
!                       nmass                   -               New Total Vehicle Mass (kg)
!                       thrust                  -               Thrust value (N)
!               External Parameters Used:
!                       abrtransalt             -               AB-R Transition Altitude (m)
!                       abrtransM               -               AB-R Transition Mach Number (6-10)
!                       abstage                 -               Current Propulsion Stage for RABCC
!                       altitude                -               Current Altitude (m)
!                       boalt                   -               Burnout Altitude (m)
!                       CF                      -               Final Rocket Effective Exhaust Velocity (m/s)
!                       Cp                      -               Specific Heat at Constant Pressure (J/kg-K)
!                       CMSR                    -               Effective Exhaust Velocities for each stage of the MSR (m/s)
!                       C1                      -               Main Rocket Effective Exhaust Velocity (m/s)
!                       dt                      -               Time Step (s)
!                       fuelspent               -               Total Mass of Fuel Spent (kg)
!                       gamma                   -               Ratio of Specific Heats
!                       mABir                   -               AB Initial Rocket Mass Drop (structural) (kg)
!                       mabthetah               -               AB-R Transition Max Horizon Vehicle Angle (deg)
!                       mdotpf                  -               Mass Flow Rate of Final Rocket Propellant (kg/s)
!                       mdotp1                  -               Mass Flow Rate of Main Rocket Propellant (kg/s)
!                       MFMSR                   -               Propellant Flow Rate for each stage of the MSR (kg/s)
!                       MMSR                    -               Total Mass for each stage of the MSR (kg)
!                       mprop                   -               Mass of Propellant (kg)
!                       MSTRMSR                 -               Structural Mass for each stage of the MSR (kg)
!                       NRS                     -               Number of rocket stages
!                       rabtransalt             -               R-AB Transition Altitude (m)
!                       remainmstr              -               Remaining Structural Mass (kg)
!                       Rgas                    -               Gas Constant (J/kg-K)
!                       thetanew                -               Current Vehicle Angle measured from the vertical (rad)
!                       time                    -               Current Time (s)
!                       rstage                  -               Current rocket stage
!                       tr_model                -               Trajectory Model (0 - CAA, 1 - SA CAA, 2 - SA AC)
!               Internal Parameters Used:
!                       counterv                -               Flag to insure proper flow of thrust models
!                       ma                      -               Mass Flow Rate of Air (kg/s)
!                       mf                      -               Mass Flow Rate of Fuel (kg/s)
!                       Minft                   -               Free-stream Mach Number
!                       scount                  -               Counter for rocket stages
!                       Sumsmass                -               Running sum of total mass for current stages (kg)
!                       Sumsmassstr             -               Running sum of total structural mass for current stages (kg)
!                       Tott0                   -               Total Temperature at Engine Inlet (K)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Real(kind = 8), Intent(IN) :: FSRhot, FSTt, FSPt, FSVt, Aratio, crossA
Real(kind = 8), Intent(IN) :: Tott4, hvf, omass
Integer, Intent(IN) :: vmodel
Real(kind = 8), Intent(OUT) :: nmass, thrust

Real(kind = 8) :: ma, mf, Minft, Tott0, Sumsmass = 0, Sumsmassstr = 0
Integer :: counterv = 0, scount

!               For vmodel = 0, the rocket's thrust is simply calculated by the product of the
!               mass flow rate of propellant and the effective exhaust velocity.  Mass
!               expenditures are also accounted for.
If(vmodel == 0) Then
                If((altitude > boalt) .and. (counterv == 0)) Then
                        counterv = counterv + 1
                End If
                If((altitude < boalt) .and. (counterv == 1)) Then
                        Print *, "DROPPED BELOW BURNOUT ALITUDE!"
                        STOP
```

```
                End If
                If(counterv == 0) Then
                        thrust = mdotp1*C1
                        nmass = omass - mdotp1*dt
                        fuelspent = fuelspent + mdotp1*dt
                Else If(counterv == 1) Then
                        thrust = 0
                End If
                If(fuelspent >= mprop) STOP "OUT OF FUEL!"

!               For vmodel = 1, the calculations were separated by trajectory model.  For
!               tr_model = 0, the AB does not contain any rockets so only calculations for the
!               specified AB thrust will be performed.  For tr_model = 1, since this model
!               consists of an initial rocket, the AB leg, and the final rocket, the thrust
!               model was separated by some indication of leg transitioning (i.e. rabtransalt).
!               For tr_model = 2, since this model is an accelerated climb, the condition for
!               transitioning to the final rocket is whether the abrtransalt or abrtransM is
!               attained as opposed to having vehicle heading criteria (for tr_model = 1).
!               This is because when a level acceleration takes place the vehicle must be
!               rotated       before the rocket leg begins.  Since tr_model = 2 is an accelerated
!               climb, the vehicle's heading is already oriented in the "right" direction and
!               the only criteria for transitioning is the altitude and/or Mach number.
Else If(vmodel == 1) Then
                If(tr_model == 0) Then
                        ma = FSRhot*FSVt*Aratio*crossA
                        thrust = ma*(sqrt(2*Cp*Tott4) - FSVt)
                        Tott0 = FSTt*(1 + ((gamma - 1)/2.)*(Minft**2))
                        mf = ma*Cp*(Tott4 - Tott0)/hvf
                        nmass = omass - mf*dt
                        fuelspent = fuelspent + mf*dt
                Else If(tr_model == 1) Then
                        Minft = FSVt/sqrt(gamma*Rgas*FSTt)
                        If(altitude <= rabtransalt) Then
                                If(counterv == 0) Then
                                        Print *, "ROCKET STAGE!"
                                        abstage = 1
                                        counterv = counterv + 1
                                End If
                                If((counterv == 2) .or. (counterv == 3)) Then
                                        Print *, "DROPPED BELOW TRANSITION ALTITUDE!"
                                        STOP
                                End If
                                thrust = mdotp1*C1
                                nmass = omass - mdotp1*dt
                                fuelspent = fuelspent + mdotp1*dt
                        Else If((altitude > rabtransalt) .or. (counterv == 3)) Then
                                If(counterv == 1) Then
                                        Print *, "RAM/SCRAM STAGE!"
                                        nmass = nmass - mABir
                                        remainmstr = remainmstr - mABir
                                        abstage = 2
                                        counterv = counterv + 1
                                End If
                                If((Minft < abrtransM) .and. (counterv <= 2)) Then
                                        ma = FSRhot*FSVt*Aratio*crossA
                                        thrust = ma*(sqrt(2*Cp*Tott4) - FSVt)
                                        Tott0 = FSTt*(1 + ((gamma - 1)/2.)*(Minft**2))
                                        mf = ma*Cp*(Tott4 - Tott0)/hvf
                                        nmass = omass - mf*dt
                                        fuelspent = fuelspent + mf*dt
                                Else If((Minft >= abrtransM) .or. (counterv >= 3)) Then
                                        If((thetanew-phi > (pi/2. - (mabthetah*pi/180.))) .and. (counterv < 4)) Then
                                                ma = FSRhot*FSVt*Aratio*crossA
                                                thrust = ma*(sqrt(2*Cp*Tott4) - FSVt)
                                                Tott0 = FSTt*(1 + ((gamma - 1)/2.)*(Minft**2))
                                                mf = ma*Cp*(Tott4 - Tott0)/hvf
                                                nmass = omass - mf*dt
                                                fuelspent = fuelspent + mf*dt
                                                If(counterv == 2) Then
                                                        counterv = counterv + 1
```

```
                                End If
                        Else If((thetanew-phi <= (pi/2. - (mabthetah*pi/180.))) .or. (counterv >= 4)) Then
                                If(counterv == 3) Then
                                        Print *, "FINAL ROCKET STAGE!"
                                        abstage = 3
                                        counterv = counterv + 1
                                End If
                                If((altitude > boalt) .and. (counterv == 4)) Then
                                        counterv = counterv + 1
                                End If
                                If((counterv == 5) .and. (altitude < boalt)) Then
                                        Print *, "DROPPED BELOW BURNOUT ALTITUDE!"
                                        STOP
                                End If
                                If(counterv == 4) Then
                                        thrust = mdotpf*CF
                                        nmass = omass - mdotpf*dt
                                        fuelspent = fuelspent + mdotpf*dt
                                Else If(counterv == 5) Then
                                        thrust = 0
                                End If
                        End If
                End If
        End If
Else If(tr_model == 2) Then
        Minft = FSVt/sqrt(gamma*Rgas*FSTt)
        If(altitude <= rabtransalt) Then
                If(counterv == 0) Then
                        Print *, "ROCKET STAGE!"
                        abstage = 1
                        counterv = counterv + 1
                End If
                If((counterv == 2) .or. (counterv == 3)) Then
                        Print *, "DROPPED BELOW TRANSITION ALTITUDE!"
                        STOP
                End If
                thrust = mdotp1*C1
                nmass = omass - mdotp1*dt
                fuelspent = fuelspent + mdotp1*dt
        Else If((altitude > rabtransalt) .or. (counterv == 3)) Then
                If(counterv == 1) Then
                        Print *, "RAM/SCRAM STAGE!"
                        nmass = nmass - mABir
                        remainmstr = remainmstr - mABir
                        abstage = 2
                        counterv = counterv + 1
                End If
                If((abstage == 2) .and. (counterv <= 2)) Then
                        ma = FSRhot*FSVt*Aratio*crossA
                        thrust = ma*(sqrt(2*Cp*Tott4) - FSVt)
                        Tott0 = FSTt*(1 + ((gamma - 1)/2.)*(Minft**2))
                        mf = ma*Cp*(Tott4 - Tott0)/hvf
                        nmass = omass - mf*dt
                        fuelspent = fuelspent + mf*dt
                        If((counterv == 2) .and. (altitude > abrtransalt)) Then
                                counterv = counterv + 1
                        End If
                Else If((altitude > abrtransalt) .or. (Minft >= abrtransM) .or. (counterv >= 3)) Then
                        If(counterv == 3) Then
                                Print *, "FINAL ROCKET STAGE!"
                                abstage = 3
                                counterv = counterv + 1
                        End If
                        If((altitude > boalt) .and. (counterv == 4)) Then
                                counterv = counterv + 1
                        End If
                        If((counterv == 5) .and. (altitude < boalt)) Then
                                Print *, "DROPPED BELOW BURNOUT ALTITUDE!"
                                STOP
                        End If
```

```
                              If(counterv == 4) Then
                                        thrust = mdotpf*CF
                                        nmass = omass - mdotpf*dt
                                        fuelspent = fuelspent + mdotpf*dt
                              Else If(counterv == 5) Then
                                        thrust = 0
                              End If
                    End If
          End If
     End If
     If(fuelspent >= mprop) STOP "OUT OF FUEL!"

!    For vmodel = 2, the MSR thrust calculations are similar to the SSR calculations.
!    However, now, due to the possibility of varying effective exhaust velocities,
!    propellant mass consumption rates, and structural masses, accounting for all of
!    these mass issues needed to be addressed.
Else If(vmodel == 2) Then
     If((altitude > boalt) .and. (counterv == 0)) Then
          counterv = counterv + 1
     End If
     If((altitude < boalt) .and. (counterv == 1)) Then
          Print *, "DROPPED BELOW BURNOUT ALITUDE!"
          STOP
     End If
     If(counterv == 0) Then
          If(time == dt)Then
               Print*, "Stage 1"
          End If
          If(rstage <= NRS) Then
               thrust = MFMSR(rstage)*CMSR(rstage)
               nmass = omass - MFMSR(rstage)*dt
               fuelspent = fuelspent + MFMSR(rstage)*dt
               Do scount = 1, rstage
                    Sumsmass = Sumsmass + MMSR(scount)
                    Sumsmassstr = Sumsmassstr + MSTRMSR(rstage)
               End Do
               If(fuelspent >= Sumsmass - Sumsmassstr) Then
                    nmass = nmass - MSTRMSR(rstage)
                    remainmstr = remainmstr - MSTRMSR(rstage)
                    rstage = rstage + 1
                    If(rstage <= NRS)Then
                         Print*, "Stage", rstage
                    End If
               End If
               Sumsmass = 0
               Sumsmassstr = 0
          End If
          If((rstage > NRS) .and. (altitude < boalt)) Then
               Print*, "Final Rocket Stage has ended before burnout altitude was acheived!"
               thrust = 0
          End If
     Else If(counterv == 1) Then
          thrust = 0
     End If
     If(fuelspent >= mprop) STOP "OUT OF FUEL!"
End If

If(thrust < 0) Then
     Print*, "WARNING: NEGATIVE THRUST!!!!"
End If

End Subroutine Thrust_Model



Subroutine Weight_Model(xpos, ypos, vmass, thetaw, xweight, yweight)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!    Name:  Tyler Winter
!    Description:  Routine to model weight for various trajectories.
```

```
!		Subroutines Called:
!			Gravity_Model(alt, gmodel, galt)
!		Inputs:
!			thetaw	-	Angle between Earth-fixed Frame and Vehicle Frame (rad)
!			vmass	-	Vehicle Mass (kg)
!			xpos	-	X-position of Vehicle in Earth-fixed Frame (m)
!			ypos	-	Y-position of Vehicle in Earth-fixed Frame (m)
!		Outputs:
!			xweight	-	X-component of Weight Value (N)
!			yweight	-	Y-component of Weight Value (N)
!		External Parameters Used:
!			e_model	-	Earth Model (0 - Flat Earth, 1 - Round Earth)
!			g_model	-	Gravity Model (0 - Sea-level, 1 - g(alt))
!			pi	-	Ratio of a Circle's Circumference to its Diameter
!			Re	-	Radius of Earth (m)
!			tr_model	-	Trajectory Model (0 - Const. Alt., 1 - Space Access)
!		Internal Parameters Used:
!			accelg	-	Gravitational Acceleration (m/s^2)
!			altw	-	Current Altitude (m)
!			phiw	-	Angle between Earth-fixed Y-axis and Radius Vector (rad)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Real(kind = 8), Intent(IN) :: xpos, ypos, vmass, thetaw
Real(kind = 8), Intent(OUT) :: xweight, yweight

Real(kind = 8) :: phiw, accelg, altw

!		For e_model = 0, each trajectory model is addressed and the altitude is
!		calculated to determine weight components.  Global vehicle rotation is accounted
!		for by phiw, since this is the flat Earth model, phiw = 0 (i.e. no rotation).
If(e_model == 0) Then
		If(tr_model == 0) Then
			altw = ypos
		Else If(tr_model > 0) Then
			altw = sqrt(ypos**2 + xpos**2)
		End If
		phiw = 0.0

!		For e_model = 1, now rotation must be addressed by calculating the angle phi.
!		Again, altitude is also determined.
Else If(e_model == 1) Then
		If(tr_model == 0) Then
			altw = sqrt((Re + ypos)**2 + xpos**2) - Re
		Else If(tr_model > 0) Then
			altw = sqrt((Re + ypos)**2 + xpos**2) - Re
		End If
		If((xpos > 0) .and. (Re + ypos < 0))Then
			phiw = pi + atan(xpos/(Re + ypos))
		Else If((xpos < 0) .and. (Re + ypos < 0))Then
			phiw = pi + atan(xpos/(Re + ypos))
		Else If((xpos < 0) .and. (Re + ypos > 0))Then
			phiw = 2*pi + atan(xpos/(Re + ypos))
		Else
			phiw = atan(xpos/(Re + ypos))
		End If
End If
If(altw < 0) STOP "NEGATIVE ALTITUDE VALUE!"


!		Deteriming acceleration due to gravity from altitude determined above
Call Gravity_Model(altw, g_model, accelg)

!		Calculation of weight components
xweight	= vmass*accelg*cos(thetaw - phiw)
yweight = vmass*accelg*sin(thetaw - phiw)

End Subroutine Weight_Model
```

```
Subroutine Write_Data(xwd, ywd, mvwd, thetawd, alphawd, twd, lwd, dwd, wxwd, wywd, &
		& timewd, vwd, tempwd, preswd, denswd, altwd, qinfwd, accelwd, dpewd, dkewd,  &
		& dfewd, dlwwd, phiwd, dVxwd, dVywd)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!		Name:  Tyler Winter
!		Description:  Routine to write data to a file.
!		Subroutines Called:  None
!		Inputs:
!			accelwd		-	Instantaneous Acceleration (m/s^2)
!			alphawd		-	Wing Angle (rad)
!			altwd		-	Altitude (m)
!			denswd		-	Density (kg/m^3)
!			dfewd		-	Running sum of energy content of expended fuel (J)
!			dkewd		-	Running sum of kinetic energy (J)
!			dlwwd		-	Running sum of lost work (J)
!			dpewd		-	Running sum of potential energy (J)
!			dVxwd		-	Differential change in x-component of vehicle velocity (m/s)
!			dVywd		-	Differential change in y-component of vehicle velocity (m/s)
!			dwd		-	Drag (N)
!			lwd		-	Lift (N)
!			mvwd		-	Vehicle Mass (kg)
!			phiwd		-	Angle between Earth-fixed Y-axis and Radius Vector (rad)
!			preswd		-	Pressure (N/m^2)
!			qinfwd		-	Free-stream Dynamic Pressure (N/m^2)
!			tempwd		-	Temperature (K)
!			thetawd		-	Angle between Earth-fixed Frame and Vehicle Frame (rad)
!			timewd		-	Current time (s)
!			twd		-	Thrust (N)
!			vwd		-	Velocity (m/s)
!			wxwd		-	X-component of Weight Value (N)
!			wywd		-	Y-component of Weight Value (N)
!			xwd		-	X-position of Vehicle in Earth-fixed Frame (m)
!			ywd		-	Y-position of Vehicle in Earth-fixed Frame (m)
!		Outputs:  None
!		External Parameters Used:
!			altitude	-	Current Altitude (m)
!			dalphalim	-	Max Change in Wing Angle Limit (deg)
!			num_lines	-	Indicates How Often to Write to File
!			orbalt		-	Orbital Altitude (m)
!			pi		-	Ratio of a Circle's Circumference to its Diameter
!		Internal Parameters Used:
!			alphaoldwd	-	Wing Angle of Attack from previous time step (rad)
!			counterwd	-	Counter to Keep Track of Iterations
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Real(kind = 8), Intent(IN) :: xwd, ywd, mvwd, thetawd, alphawd, twd, lwd, dwd, phiwd
Real(kind = 8), Intent(IN) :: wxwd, wywd, timewd, vwd, tempwd, preswd, denswd
Real(kind = 8), Intent(IN) :: altwd, qinfwd, accelwd, dpewd, dkewd, dfewd, dlwwd
Real(kind = 8), Intent(IN) :: dVxwd, dVywd

Real(kind = 8) :: alphaoldwd
Integer :: counterwd = 0

!		Opens files for writing and initializes parameter headings
If(counterwd == 0) Then
		Open(unit = 11, file = 'Outputs.out', status = 'new', action = 'write', &
			& position = 'rewind')
!		Write(11, *) "			Time (s)				X (m)				", &
!		& "			Y (m)						Mass (kg)
		Theta (deg)", &
!		& "			Alpha (deg)						T (N)
L (N) ", &
!		& "			D (N)						Wx (N)
Wy (N)				", &
!		& "		Vnew (m/s)						Temperature (K)
Pressure ", &
!		& "(Pa)				Density (kg/m^3)				Altitude (m)   ", &
```

```
!           & "    Mach                                              Qinf (Pa)                     Instantaneous
Accele", &
!           & "ration (m/s^2)                              KE Changes (J)                          PE Changes (J)
            ", &
!           & "          Energy Content of Expended Fuel (J)                                  Total Lost W", &
!           & "ork (J) "
            Write(11, *) "          Time (s)            X (m)                        Y (m)          ", &
                  & "    dVx (m/s)          dVy (m/s)         Mass (kg)           ", &
                  & "Theta(deg)      Alpha (deg)      Phi (deg)           T (N)             ", &
                  & "   L (N)                D (N)                       Wx (N)          Wy (N)                  Vnew (m", &
                  & "/s)     Temperature (K)    Pressure (Pa)    Density (kg/m^3)     ", &
                  & " Altitude (m)          Mach               Qinf (Pa)     I. Accel. (m", &
                  & "/s^2)     KE Changes (J)       PE Changes (J)      E. C. Exp. Fue", &
                  & "l (J)   Total Lost Work (J)"
End If

!           Writes to a file only when the specified number of lines is skipped or when
!           the orbital altitude is attained
If ((mod(counterwd, num_lines) == 0) .or. (altitude > orbalt)) Then
            Write(11, *) timewd, xwd, ywd, dVxwd, dVywd, mvwd, thetawd*(180./pi), alphawd*(180./pi), &
            & phiwd*(180./pi), twd, lwd, dwd, wxwd, wywd, vwd, tempwd, preswd, denswd, altwd, &
            & Vnew/sqrt(gamma*Rgas*temperature), qinfwd, accelwd, dkewd, dpewd, &
            & dfewd, dlwwd
End If
If(counterwd > 5) Then
            If(abs(alphaoldwd - alphawd) > dalphalim*(pi/180.)) Then
                  Print*, "MAX CHANGE IN ALPHA LIMIT VIOLATED!  CHECK OUTPUTS!"
                  Print*, "ATTEMPTED CHANGE IN ALPHA:  ", abs(alphaoldwd - alphawd)*(180./pi)
                  STOP
            End If
End If

counterwd = counterwd + 1
alphaoldwd = alphawd

End Subroutine Write_Data
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

End Program Trajectory_Solver
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

APPENDIX B

ANALYTICAL PROCEDURE TO DETERMINE THE SHOCK DETACHMENT ANGLE

Analytical Procedure to Determine the Shock Detachment Angle (θ_{max})

Introduction

In order to expedite computation time, an analytical solution is always preferred to an iterative solution. Having a closed-form solution for a parameter enables efficient calculations to be performed computationally. The variable of interest in this document is the maximum deflection angle, θ_{max}, at which a shock wave detaches from the surface of a body. This angle is a function of the shock wave angle and Mach number. The following is a procedure to determine an analytical solution for θ_{max} for a given shock wave angle and Mach number.

Nomenclature

θ_{max} - Maximum Deflection Angle

θ - Flow-Deflection Angle

β - Oblique Shock Wave Angle

M - Upstream Mach Number

γ - Ratio of Specific Heats

A, B, C - Coefficients for Biquadratic Equation

Procedure

Start by writing the well-known θ-β-M relation [1]:

$$f = \tan(\theta) = 2\cot(\beta)\left[\frac{M^2\sin^2(\beta)-1}{M^2(\gamma+\cos(2\beta))+2}\right]$$

Where θ is the flow-deflection angle, β is the oblique shock wave angle, M is the Mach number upstream of the shock wave, and γ is the ratio of specific heats. Since the upstream Mach number will be known and γ will be fixed, it will be desirable to determine the value of β which maximizes the function stated above. It is important to note that the maximum of f occurs when θ is a maximum and hence $\tan(\theta)$ is a maximum. By taking the derivative with respect to β of the right-hand side and setting it equal to zero the following can be shown (Note: Several trigonometric identities that are not shown below were used to obtain the final result.):

$$\frac{\partial f}{\partial \beta} = \frac{\left(-2\csc^2(\beta)\right)\left(M^2\sin^2(\beta)-1\right)}{M^2\left(\gamma+\cos(2\beta)\right)+2} + \left(2\cot(\beta)\right)\left[M^2\sin(2\beta)\right]\left[\frac{\left[M^2\left(\gamma+\cos(2\beta)\right)+2\right]+2\left[M^2\sin^2(\beta)-1\right]}{\left[M^2\left(\gamma+\cos(2\beta)\right)+2\right]^2}\right] = 0$$

Rearranging yields:

$$\frac{\left(M^2\sin^2(\beta)-1\right)}{\sin(\beta)\cos(\beta)} = \left[2M^2\sin(2\beta)\right]\left[\frac{1}{2}+\frac{\left[M^2\sin^2(\beta)-1\right]}{\left[M^2\left(\gamma+\cos(2\beta)\right)+2\right]}\right] \Rightarrow$$

$$\frac{\left(M^2\sin^2(\beta)-1\right)}{M^2\sin^2(2\beta)} = \left[\frac{1}{2}+\frac{\left[M^2\sin^2(\beta)-1\right]}{\left[M^2\left(\gamma+\cos(2\beta)\right)+2\right]}\right] \Rightarrow$$

$$\left(M^2\sin^2(\beta)-1\right)\left[\frac{1}{M^2\sin^2(2\beta)}-\frac{1}{\left[M^2\left(\gamma+\cos(2\beta)\right)+2\right]}\right] = \frac{1}{2}$$

Finding a common denominator yields:

$$\left(M^2\sin^2(\beta)-1\right)\left[\frac{M^2\gamma+M^2\cos(2\beta)+2-M^2\sin^2(2\beta)}{M^4\gamma\sin^2(2\beta)+M^4\sin^2(2\beta)\cos(2\beta)+2M^2\sin^2(2\beta)}\right]=\frac{1}{2}\Rightarrow$$

$$M^4\gamma\sin^2(\beta)+M^4\sin^2(\beta)\cos(2\beta)+2M^2\sin^2(\beta)-M^4\sin^2(2\beta)\sin^2(\beta)-M^2\gamma-$$
$$M^2\cos(2\beta)-2=2M^4\gamma\sin^2(\beta)\cos^2(\beta)+M^4\sin(\beta)\cos(\beta)\sin(2\beta)\cos(2\beta)$$

Writing all of the trigonometric quantities in terms of $\sin(\beta)$ and simplifying yields:

$$\sin^4(\beta)+\left(-\frac{1}{2}-\frac{1}{2\gamma}+\frac{2}{M^2\gamma}\right)\sin^2(\beta)+\left(-\frac{1}{2M^2}-\frac{1}{2M^2\gamma}-\frac{1}{M^4\gamma}\right)=0$$

By making the following substitutions, the biquadratic equation above can be simplified:

$$x=\sin^2(\beta),\ A=1,\ B=\frac{-M^2\gamma-M^2+4}{2M^2\gamma},\text{ and }C=\frac{-M^2\gamma-M^2-2}{2M^4\gamma}$$

$Ax^2+Bx+C=0$ with the familiar solutions: $x_1=\frac{-B+\sqrt{B^2-4AC}}{2A}$ and $x_2=\frac{-B-\sqrt{B^2-4AC}}{2A}$

So we have four resulting solutions for β:

$$\beta_1=\sin^{-1}\left(\sqrt{\frac{-B+\sqrt{B^2-4AC}}{2A}}\right),\beta_2=\sin^{-1}\left(-\sqrt{\frac{-B+\sqrt{B^2-4AC}}{2A}}\right)$$

$$\beta_3=\sin^{-1}\left(\sqrt{\frac{-B-\sqrt{B^2-4AC}}{2A}}\right),\text{ and }\beta_4=\sin^{-1}\left(-\sqrt{\frac{-B-\sqrt{B^2-4AC}}{2A}}\right)$$

To determine which β is the correct solution corresponding to the natural phenomena of shock detachment, one must find a real, positive solution for β which maximizes f. Since β_2 and β_4 are negative they can be immediately discarded as shown below:

$$\beta_2 = \sin^{-1}\left(-\sqrt{\frac{-B+\sqrt{B^2-4AC}}{2A}}\right) = -\sin^{-1}\left(\sqrt{\frac{-B+\sqrt{B^2-4AC}}{2A}}\right) < 0$$

$$\beta_4 = \sin^{-1}\left(-\sqrt{\frac{-B-\sqrt{B^2-4AC}}{2A}}\right) = -\sin^{-1}\left(\sqrt{\frac{-B-\sqrt{B^2-4AC}}{2A}}\right) < 0$$

To show that β_1 is the correct solution, consider the following. In general, $M > 1$ and $\gamma > 0$ (ideally, $\gamma = 1.4$), and from this observation the coefficient C from the quadratic equation above will always be negative. Since $A = 1$, the solutions for β_1 and β_3 reduce to:

$$\beta_1 = \sin^{-1}\left(\sqrt{\frac{-B+\sqrt{B^2-4C}}{2}}\right) \text{ and } \beta_3 = \sin^{-1}\left(\sqrt{\frac{-B-\sqrt{B^2-4C}}{2}}\right)$$

Since $C < 0$: $\sqrt{B^2-4C} > B$ for all physically possible values of B. Since this is true, the radicand in the solution for β_3 will always be negative, hence, β_3 is a non-real (non-physical) solution. By elimination, β_1 is the resulting solution that determines θ_{max}. However, it is important to show that:

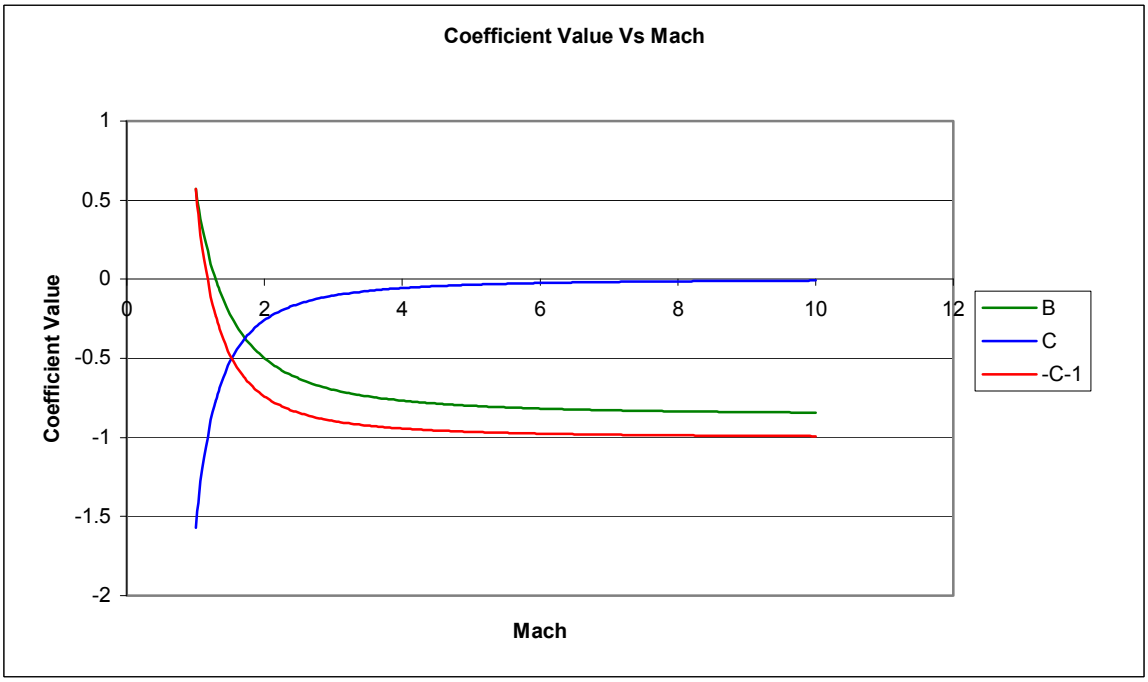
$$\sin(\beta_1) = \sqrt{\frac{-B+\sqrt{B^2-4C}}{2}} \le 1$$

(Note: It has already been shown above that $\sqrt{\frac{-B+\sqrt{B^2-4C}}{2}} \geq 0$ since we demand a positive solution.)

So it remains to show that:

$$\frac{-B+\sqrt{B^2-4C}}{2} \leq 1 \Rightarrow \sqrt{B^2-4C} \leq B+2 \Rightarrow B \geq -C-1$$

It is very helpful and easy to show that $B \geq -C-1$ for all physically possible values of B and C graphically. Consider the following graph of the coefficients as a function of Mach (with $\gamma = 1.4$):



Obviously, one can see from the above figure that $B > -C-1$ for all values of Mach except when $M = 1$ in which that case, $B = -C-1$. This, in fact, agrees with the conclusion of β_1 as the correct solution. By replacing the coefficients B and C in the

β_1 expression and substituting this into f and rearranging, one can obtain the following closed-form analytical expression, as given by Shapiro [2] in a problem, for θ_{max} :

$$\theta_{max} = \tan^{-1}\left[2\cot(\beta_1)\left[\frac{M^2\sin^2(\beta_1)-1}{M^2(\gamma+\cos(2\beta_1))+2}\right]\right]$$

Where $\beta_1 = \sin^{-1}\left[\sqrt{\frac{M^2(\gamma+1)-4+\sqrt{\left[M^4(\gamma+1)+8(\gamma-1)M^2+16\right](\gamma+1)}}{4M^2\gamma}}\right]$

References

[1] John D. Anderson Jr., "Modern Compressible Flow with Historical Perspective." 3rd Ed. Pg. 136. New York, NY. 2003.

[2] Ascher H. Shapiro, "The Dynamics and Thermodynamics of Compressible Fluid Flow." Volume 1. Pg. 586. New York, NY. 1953.

APPENDIX C

TRAJECTORY CODE OPERATION INSTRUCTIONS

Trajectory Code Operation Instructions

Before execution of the program, appropriately specify the data contained within the input file. Once the input file is updated, saved, and ready to be used, execute the program. Upon execution of the code, the program will inform the success or failure of the mission. If the mission failed, the user must make an intelligent judgment as to what caused the failure. First, one important recommendation is to view the corresponding output file to ascertain what actually caused the mission to fail. This could be done by manually looking at the numerical results in the file named "Outputs.out" or by importing that same file into Excel and visualizing the incomplete (failed) mission data. There could be a number of possibilities as to why the mission failed such as insufficient propellant, velocity, or wing angle control. Once the mission is successful, the user can follow the steps outlined here to visualize the data in a convenient Excel spreadsheet.

Importing and visualizing the data:

1. Open the "Template.xls" file and save it with a new sensible name.
2. Click the "Mission Data" tab at the bottom of the spreadsheet to view the numerical results.
3. Right click on the "A" column, such that every cell in that column is highlighted, and select "Insert."
4. Now right click on the newly named "B" column and select "Delete." (Note: To import new data into Excel, the column that was last used to import data, in this case "B", must be deleted before new data can be imported "over" the old data.)
5. Click the A2 cell so that it becomes highlighted.
6. From the menus above, select "Data", then "Import External Data", then "Import Data."
7. The output file must now be located and selected for importing. Navigate to the same folder in which the program was executed, and then select the "Outputs.out" file. (Note: This may require selecting "All Files" under the "Files of type:" option.)
8. Once you click "Open" the following next steps must be followed very carefully to insure that the data is kept in correct form and imported over the old data properly.
9. Make sure the "Delimited" radio button is selected and that the "Start import at row" option is increased to 2. Then click "Next."
10. Under the "Delimiters" section, uncheck "Tab" and check "Space." (Note: Also make sure the "Treat consecutive delimiters as one" option is checked.) Then click "Finish."
11. Click the "Properties" button and under the "Data formatting and layout" section, uncheck "Adjust column width" and make sure "Preserve cell formatting" is

checked. Next, under the "If the number of rows in the data range changes upon refresh" option, select "Overwrite existing cells with new data, clear unused cells." Then click "OK." Then click "OK" one last time.

12. The last step causes the new data to be imported over the previous data in the file. However, since the data could possibly contain more (or less) rows than the previous data, the next steps (listed here in step 12) are suggestions to assist with formatting. First, if some data appears to contain "###############" symbols, then the number of decimal places for those cells must be reduced. This can be done by clicking the "Decrease Decimal" button from the "Formatting" toolbar until the number appears in proper form. Next, it is also possible for the data corresponding to the vehicle mass fraction analysis to only appear as "1" or "0." This is basically the exact opposite of what is happening in the "###############" situation. So to fix this, select the appropriate cell(s) and then click the "Increase Decimal" button from the "Formatting" toolbar until the number appears in proper form. (Note: These two formatting operations can be done to multiple cells at once to expedite the process.)
13. Next, click the "Mission Plots" tab at the bottom of the spreadsheet to view the visual plots created from the new data. There are exactly 18 plots (two of which are pie charts). At this point, the user may be viewing incomplete (or complete) charts describing the mission results. A quick way to determine if the charts are incomplete is to view any one of the time plots and relate this to the total mission time. If it is less, then the user is viewing incomplete data, whereas, if it is greater, then the user may be viewing inaccurate information beyond the last time segment in the current mission of interest. The next steps will describe how to properly select the data sources for each of the charts.
14. Right click the gray background (not the gridlines or the actual data points) of the Altitude vs. Time plot and select "Source Data…" Click the "Series" tab at the top and then click the button attached to the "X Values:" textbox which allows the user to view where the data is selected within the spreadsheet. Upon clicking, the "Mission Data" spreadsheet should appear with some dataset highlighted by a moving dotted line box. Scroll down to make sure all of the "Time" data points are selected, if they are not the user can simply click and drag over all the time data (excluding the title "Time (s)" cell) or by adjusting the last number in the "Source Data – X Values:" box to the appropriate number corresponding to the last cell in which the final time data point is located (i.e. changing "B252" to "B274" assumes that the final time data point is found in cell B274). Once this new selection or entry is made, click the button attached to the textbox to return to the "Source Data" window. This procedure can be repeated for the "Y Values:" and the entire subsequent plot "X Values:" and "Y Values:" data sources. In fact, the numerical value (274 for the example above) determined is the same ending for all of the plots (except the mass plot and two pie charts which will be handled below). The user should take advantage of this fact to greatly expedite this process for the rest of the plots. (Note: For the On-board Energy vs. Time plot, this step occurs four times! Once for each data series.)
15. Since the mission end time and final orbital positioning maneuver are assumed to coincide at the exact same moment, the final mass of the vehicle is actually placed

in an adjacent cell below the last numerical value (i.e. 274). To add this data point, the user must reopen the "Source Data" window and click the button attached to the "Y Values:" textbox. Now, click and drag over the mass data (usually starting in cell G2 and ending at say G274), let go of the mouse button and then hold down the Ctrl button and simultaneously click the final mass value (usually in cell say L275). Then press enter to return to the "Source Data" window. (Note: If an error message is obtained, then an error in the selecting has occurred. Simply click "OK" to the message and then delete all of the text within the textbox and start the click and drag process over again remembering to select the additional final mass data point.)

16. This step will assist in updating the information in the two pie charts. Right click the On-board Energy Usage pie and select "Source Data…" Select the "Series" tab at the top of the window. Delete all of the data in the "Values:" textbox and then click the button attached to the textbox. In the "Mission Data" spreadsheet, the user must simultaneously Ctrl click the three boxes corresponding to the last data entry cell in the "KE Changes (J)", "PE Changes (J)", and "Losses (J)" columns (which should correspond to say cells X274, Y274, and AA274, respectively). Then press "Enter" and select "OK." The On-board Energy Usage pie should now contain the correct data. Now the user must repeat the process for the next pie chart. Once the "Values:" textbox is cleared, click the button attached to it. Now, simultaneously Ctrl click the cells (all six of which are located in column D below the main dataset) which correspond to the "Lambda _P_(dV)", "Lambda_P_(dh)", "Lambda_P_(ds)", "Lambda_UP", "Lambda_D", and finally "Lambda_PL" values. Press "Enter" to return to the "Source Data" window. Now the "Category Labels:" must be updated. Clear the textbox and then click the button attached to it. Then simultaneously Ctrl click the names in the cells (all six of which are located in Column B below the main dataset) corresponding to "Lambda _P_(dV)", "Lambda_P_(dh)", "Lambda_P_(ds)", "Lambda_UP", "Lambda_D", and finally "Lambda_PL." Finally, press "Enter" and click "OK."

This concludes the steps for importing and visualizing the numerical data from the mission output file. At first glance, these steps may seem overwhelming to the user. However, after this process is repeated a few times, the user will be surprised at how quickly the data can be transformed from numbers into extremely useful and descriptive plots.

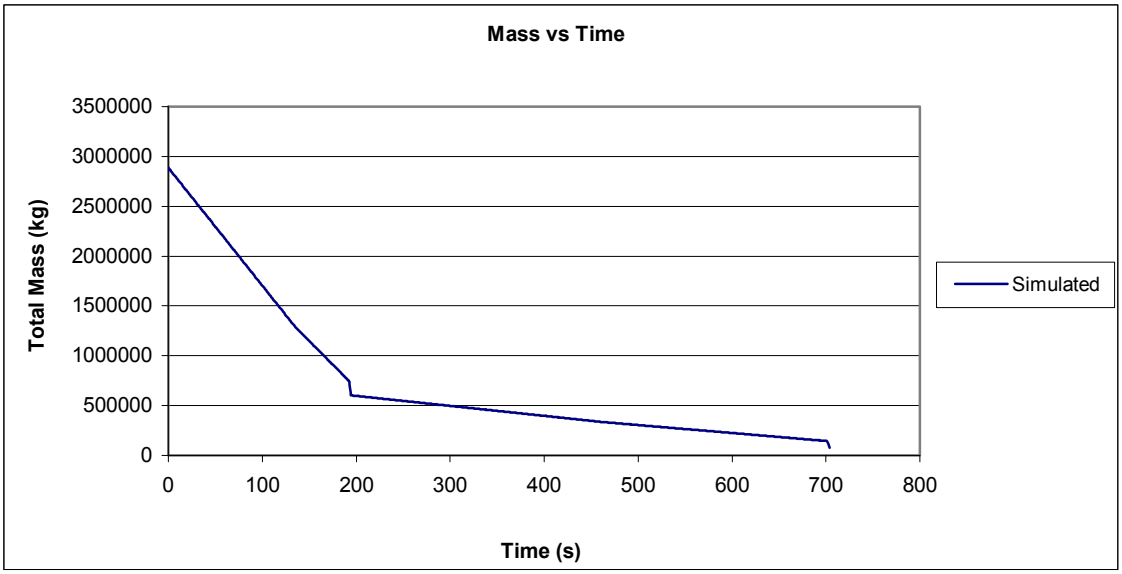
APPENDIX D

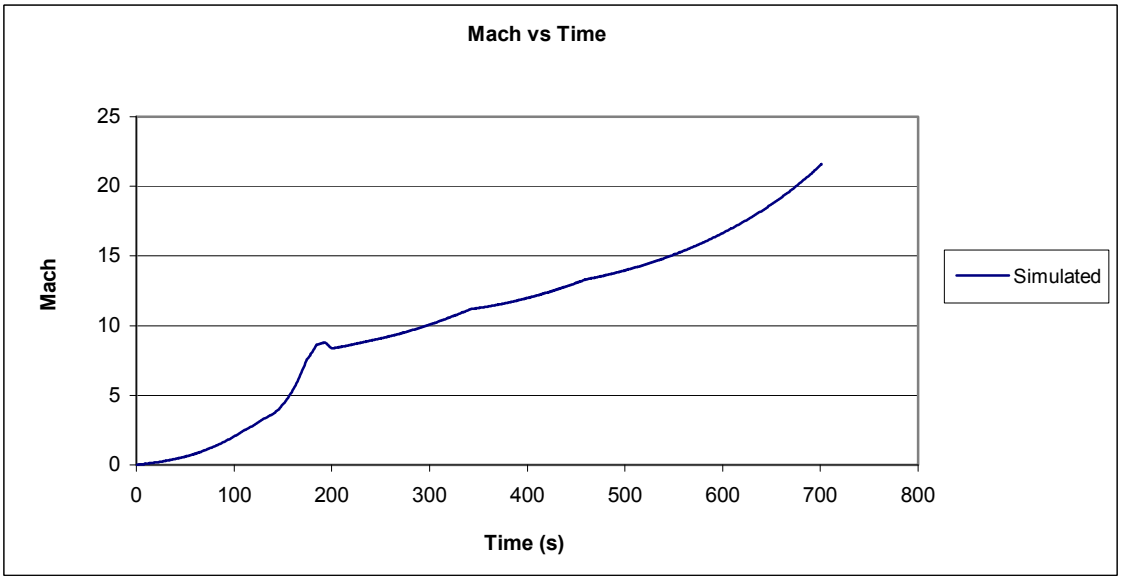
EXAMPLE INPUT DECKS AND MISSION PLOTS

APOLLO 11 MISSION INPUT DECK

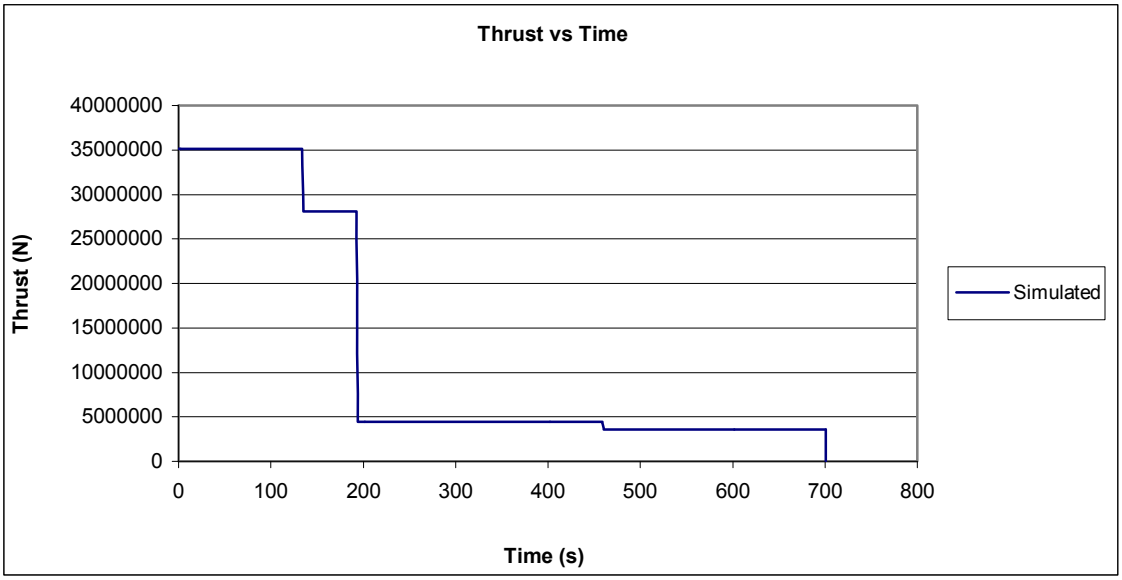
```
----------------------------------------------MODEL SPECIFICATIONS----------------------------------------------
1                                   ! Atmosphere Model (0 - Exponential, 1 - 7-part)
1                                   ! Drag Model (0 - Basic, 1 - Oblique S/E)
1                                   ! Earth Model (0 - Flat Earth, 1 - Round Earth)
1                                   ! Gravity Model for g (0 - Sea-level, 1 - g(alt))
1                                   ! Lift Model (0 - Small Angle, 1 - Oblique S/E)
2                                   ! Vehicle Model (0 - SSR, 1 - RABCC, 2 - MSR)
2                                   ! Trajectory Model (0 - CAA, 1 - SA CAA, 2 - SA AC)
----------------------------------------------MISSION SPECIFICATIONS----------------------------------------------
1                                   ! Output File Writing (0 - Off, 1 - On)
100                                 ! Print every # iterations
0.01                                ! Time Step (s)
10000.0                             ! Total Time (s)
0.0                                 ! Initial Altitude (m)
190000.0                            ! Orbital Altitude (m)
190000.0                            ! Burnout Altitude (m)
1.4                                 ! Ratio of Specific Heats
287.0                               ! Gas Constant (J/kg-K)
-----------------------------------GENERAL VEHICLE SPECIFICATIONS-----------------------------------
1.0                                 ! Initial Velocity (m/s)
0.0                                 ! Initial Wing Angle of Attack (deg)
89.9999998756                       ! Launch Angle (<= 90 deg)
0.001                               ! Delta Alpha Correction (rad)
10.0                                ! Max Change in Wing Angle Limit (deg)
30.0                                ! Max Wing Angle (deg)
90.0                                ! Max Vehicle Angle to Horizon (<= 90 deg)
0.0                                 ! Min Vehicle Angle to Horizon (<= 0 deg)
907.9334                            ! Planform Area of Wings/Lifting Body (m^2)
79.48512                            ! Cross-section Area of Vehicle (m^2)
0.34                                ! Coefficient of Drag for Body (Cdb)
---------------------------------AIR-BREATHING VEHICLE SPECIFICATIONS---------------------------------
1.0                                 ! Constant AB Wing Angle (deg)
20000.0                             ! R-AB Transition Altitude (m)
0.0005                              ! R-AB Rotational Rate of Wing (degs/sec)
0.0005                              ! R-AB Initial Smoothing Alpha Limit (deg)
0.0004                              ! R-AB Smoothing Alpha Limit (deg)
0.008                               ! AB-R Rotational Rate of Wing (degs/sec)
0.005                               ! AB-R Smoothing Alpha Limit (deg)
10.0                                ! AB-R Transition Mach Number (4-15)
80000                               ! AB-R Transition Altitude (m)
10.0                                ! AB-R Transition Max Horizon Vehicle Angle (deg)
6.0                                 ! R-Ballastic Transition Max Wing Angle (deg)
2.75                                ! Absolute Maximum Wing Angle Scale Factor
1.32e8                              ! Heating Value of Fuel (J/kg)
5000.0                              ! Total Temperature at Combustor Exit (K)
1005.0                              ! Specific Heat at Constant Pressure (J/kg-K)
10.0                                ! Ratio of Capture Area to Cross-sectional Area
0.25                                ! AB Body Factor (multiplied by Cdb)
-----------------------------------ROCKET VEHICLE SPECIFICATIONS-----------------------------------
11827.0237                          ! Mass Flow Rate of Main Rocket Propellant (kg/s)
200.2732                            ! Mass Flow Rate of Final Rocket Propellant (kg/s)
2972.43                             ! Main Rocket Effective Exhaust Velocity (m/s)
4443.93                             ! Final Rocket Effective Exhaust Velocity (m/s)
3                                   ! Number of Rocket Stages for MSR
----------------------------------------------MASS SPECIFICATIONS----------------------------------------------
2887051.0                           ! Total Vehicle Mass (including propellant) (kg)
186813.0                            ! Structural (System) Mass (kg)
1953.0                              ! Payload Mass (kg)
0.0                                 ! AB Initial Rocket Mass Drop (structural) (kg)
```

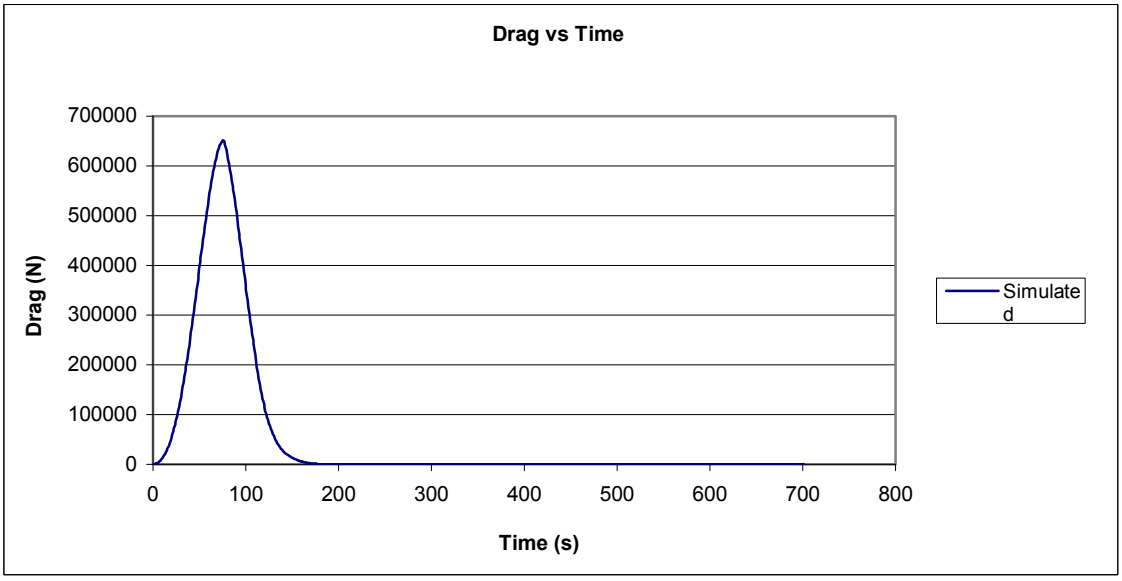
APOLLO 11 MISSION PLOTS (1/3)



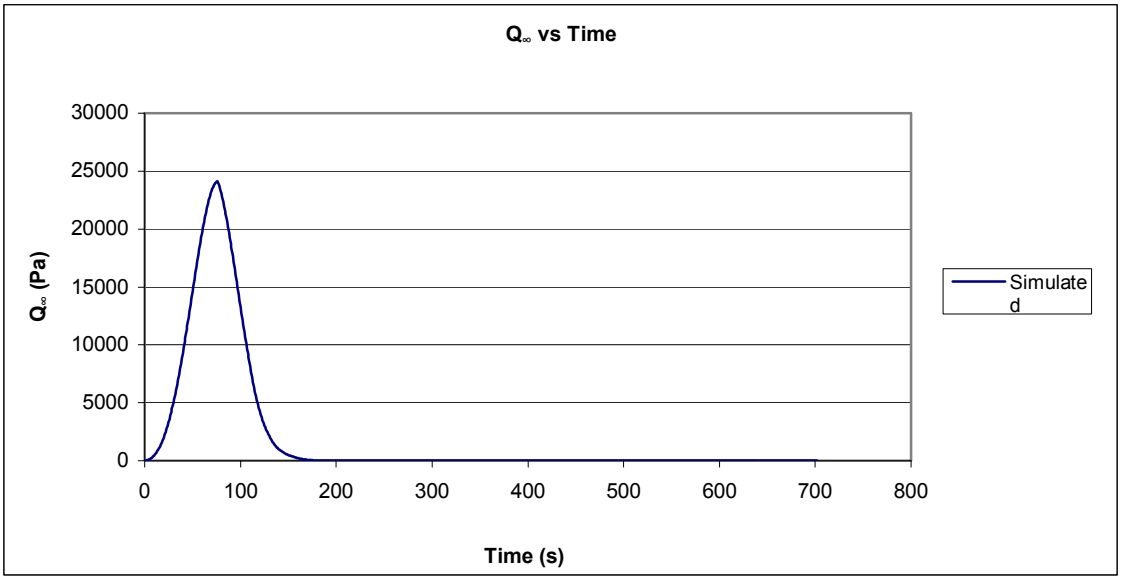


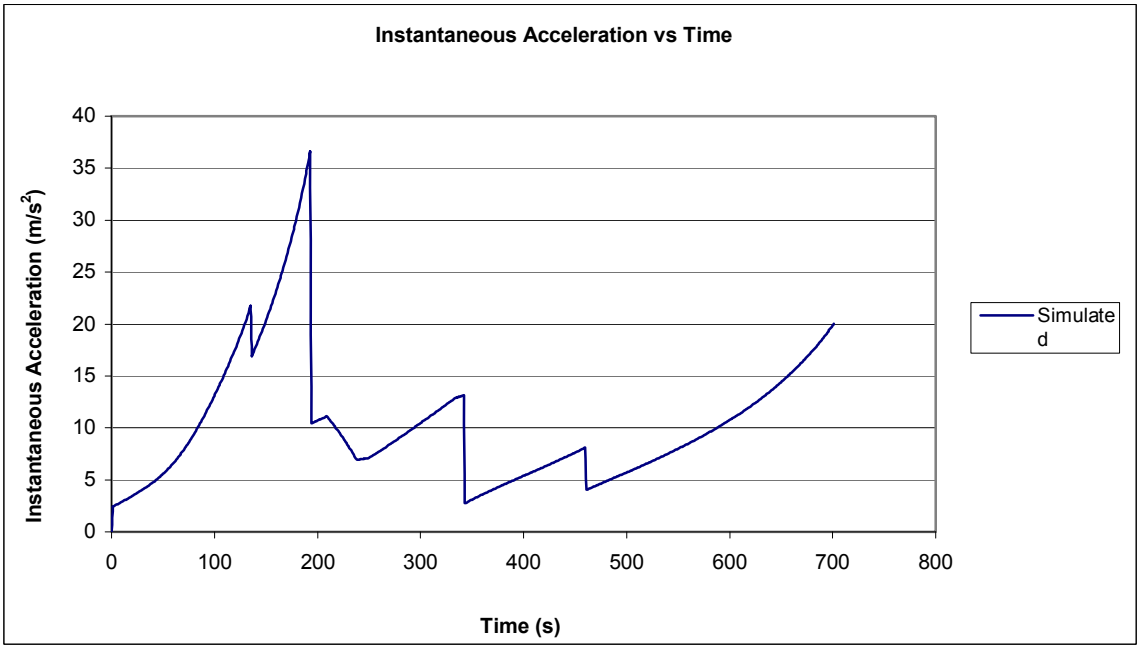
APOLLO 11 MISSION PLOTS (2/3)





APOLLO 11 MISSION PLOTS (3/3)

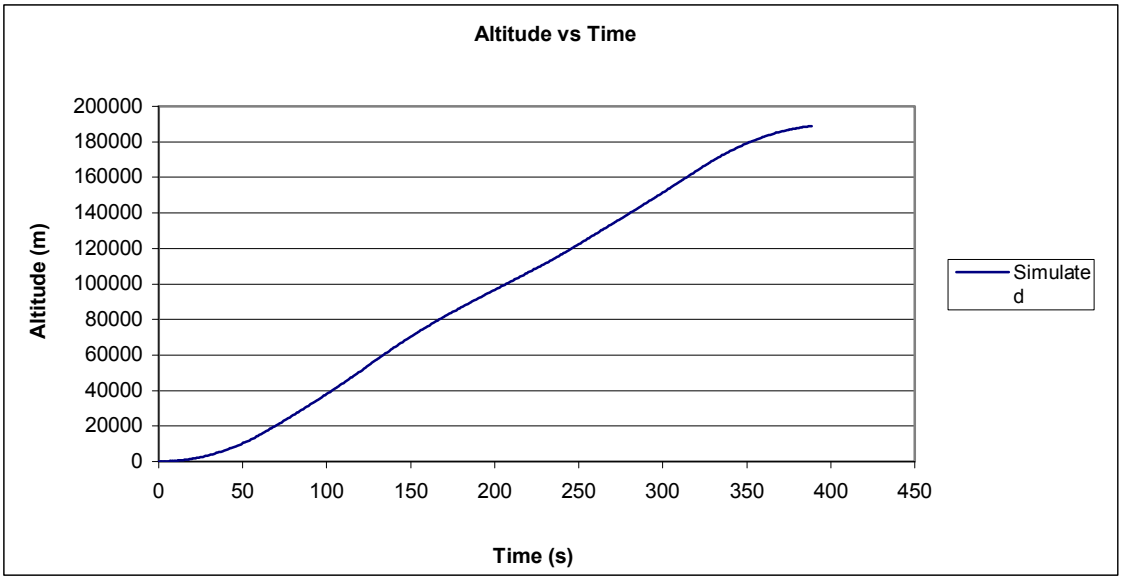


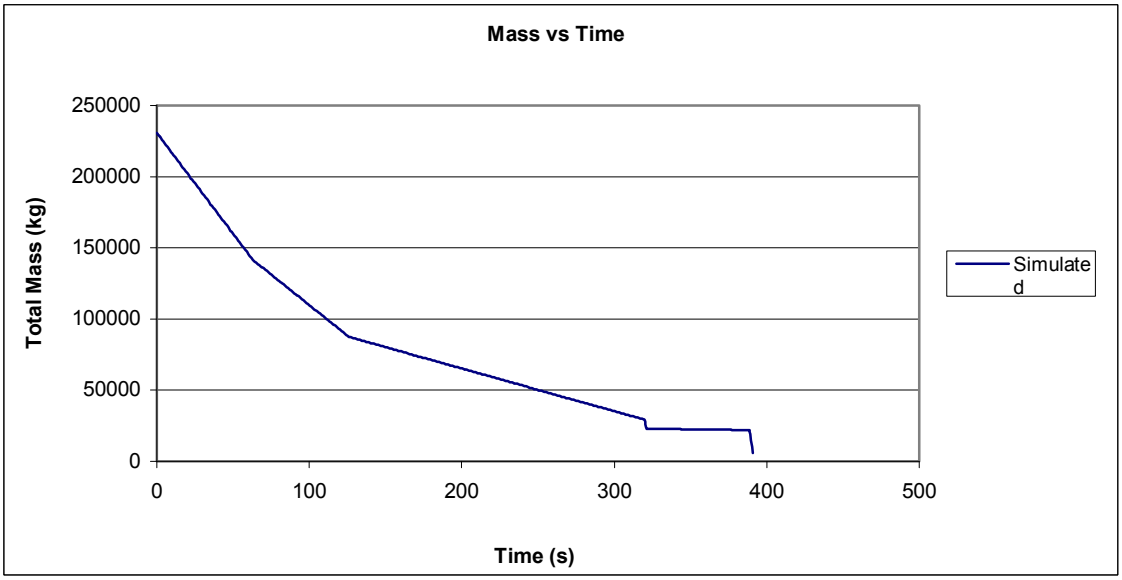


DELTA II MISSION INPUT DECK

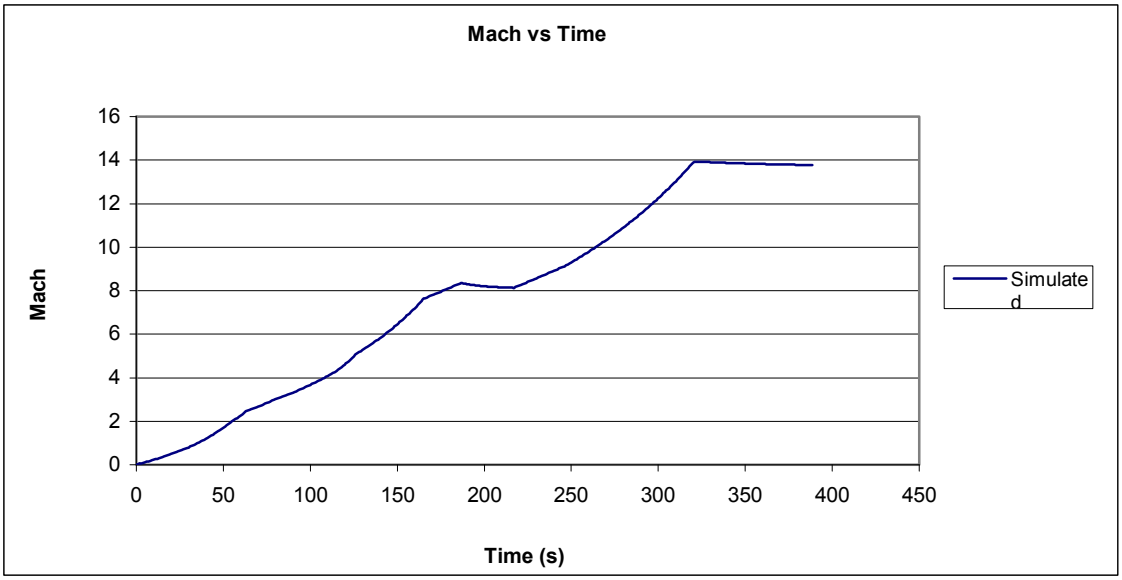
```
----------------------------------------------MODEL SPECIFICATIONS-----------------------------------------
1                                   ! Atmosphere Model (0 - Exponential, 1 - 7-part)
1                                   ! Drag Model (0 - Basic, 1 - Oblique S/E)
1                                   ! Earth Model (0 - Flat Earth, 1 - Round Earth)
1                                   ! Gravity Model for g (0 - Sea-level, 1 - g(alt))
1                                   ! Lift Model (0 - Small Angle, 1 - Oblique S/E)
2                                   ! Vehicle Model (0 - SSR, 1 - RABCC, 2 - MSR)
2                                   ! Trajectory Model (0 - CAA, 1 - SA CAA, 2 - SA AC)
----------------------------------------------MISSION SPECIFICATIONS---------------------------------------
1                                   ! Output File Writing (0 - Off, 1 - On)
100                                 ! Print every # iterations
0.01                                ! Time Step (s)
10000.0                             ! Total Time (s)
0.0                                 ! Initial Altitude (m)
189000.0                            ! Orbital Altitude (m)
189000.0                            ! Burnout Altitude (m)
1.4                                 ! Ratio of Specific Heats
287.0                               ! Gas Constant (J/kg-K)
-------------------------------------GENERAL VEHICLE SPECIFICATIONS-------------------------------------
1.0                                 ! Initial Velocity (m/s)
0.0                                 ! Initial Wing Angle of Attack (deg)
89.983                              ! Launch Angle (<= 90 deg)
0.001                               ! Delta Alpha Correction (rad)
10.0                                ! Max Change in Wing Angle Limit (deg)
30.0                                ! Max Wing Angle (deg)
90.0                                ! Max Vehicle Angle to Horizon (<= 90 deg)
0.0                                 ! Min Vehicle Angle to Horizon (<= 0 deg)
92.64                               ! Planform Area of Wings/Lifting Body (m^2)
15.2053                             ! Cross-section Area of Vehicle (m^2)
0.34                                ! Coefficient of Drag for Body (Cdb)
---------------------------------AIR-BREATHING VEHICLE SPECIFICATIONS---------------------------------
1.0                                 ! Constant AB Wing Angle (deg)
20000.0                             ! R-AB Transition Altitude (m)
0.0005                              ! R-AB Rotational Rate of Wing (degs/sec)
0.0005                              ! R-AB Initial Smoothing Alpha Limit (deg)
0.0004                              ! R-AB Smoothing Alpha Limit (deg)
0.008                               ! AB-R Rotational Rate of Wing (degs/sec)
0.005                               ! AB-R Smoothing Alpha Limit (deg)
10.0                                ! AB-R Transition Mach Number (4-15)
80000                               ! AB-R Transition Altitude (m)
10.0                                ! AB-R Transition Max Horizon Vehicle Angle (deg)
6.0                                 ! R-Ballastic Transition Max Wing Angle (deg)
2.75                                ! Absolute Maximum Wing Angle Scale Factor
1.32e8                              ! Heating Value of Fuel (J/kg)
5000.0                              ! Total Temperature at Combustor Exit (K)
1005.0                              ! Specific Heat at Constant Pressure (J/kg-K)
10.0                                ! Ratio of Capture Area to Cross-sectional Area
0.25                                ! AB Body Factor (multiplied by Cdb)
-------------------------------------ROCKET VEHICLE SPECIFICATIONS-------------------------------------
1415.475                            ! Mass Flow Rate of Main Rocket Propellant (kg/s)
23.153                              ! Mass Flow Rate of Final Rocket Propellant (kg/s)
2744.0965                           ! Main Rocket Effective Exhaust Velocity (m/s)
2866.482                            ! Final Rocket Effective Exhaust Velocity (m/s)
1.0                                 ! Rocket Accelerator Efficiency (>0, <1)
3                                   ! Number of Rocket Stages for MSR
-----------------------------------------------MASS SPECIFICATIONS-----------------------------------------
230746.1                            ! Total Vehicle Mass (including propellant) (kg)
19007.0                             ! Structural (System) Mass (kg)
2032.1                              ! Payload Mass (kg)
0.0                                 ! AB Initial Rocket Mass Drop (structural) (kg)
```

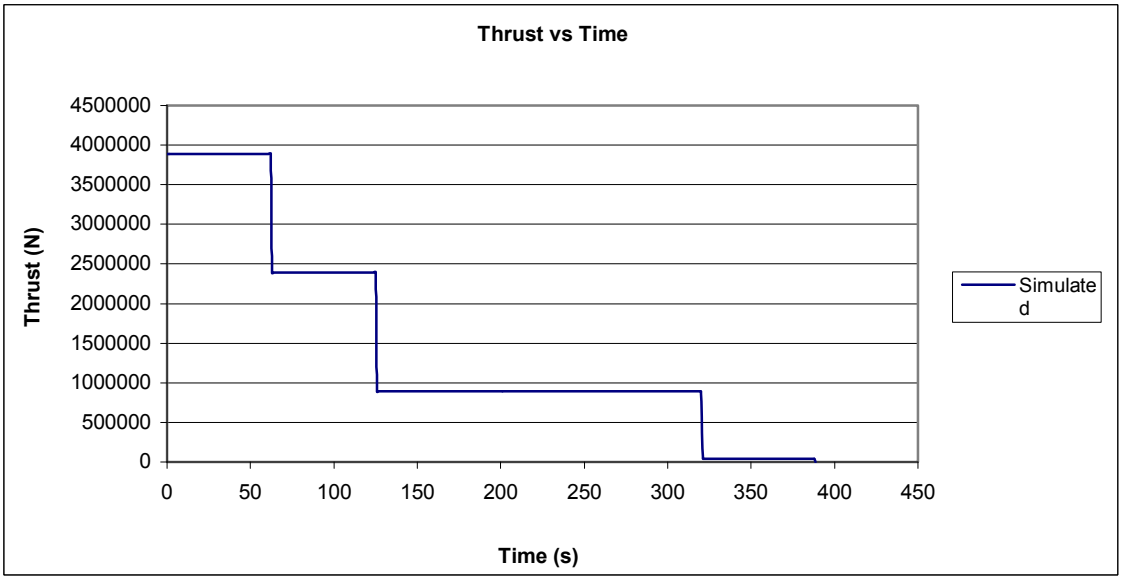
DELTA II MISSION PLOTS (1/3)



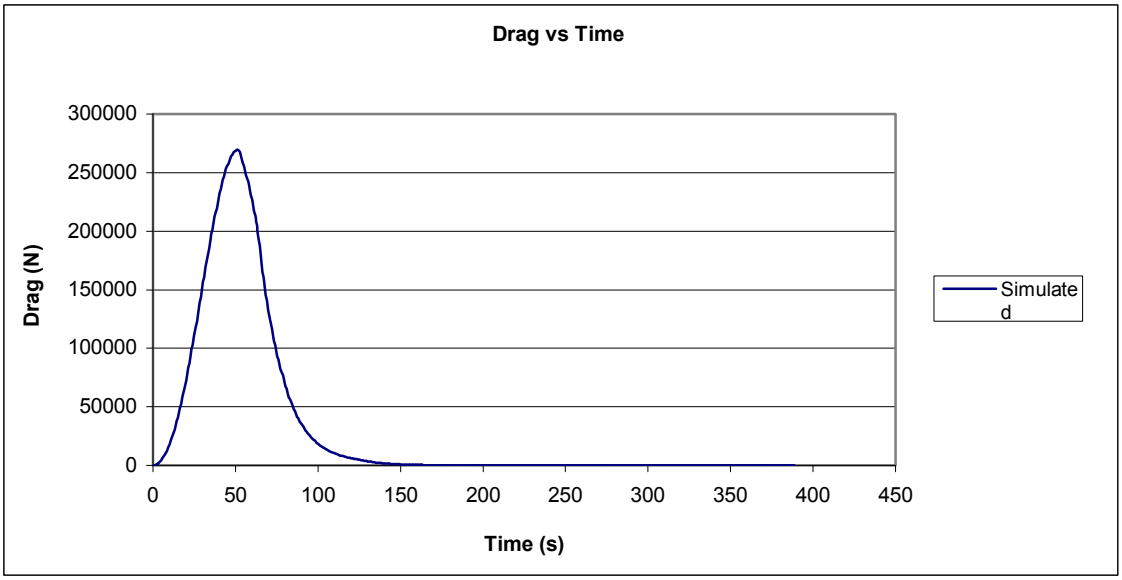


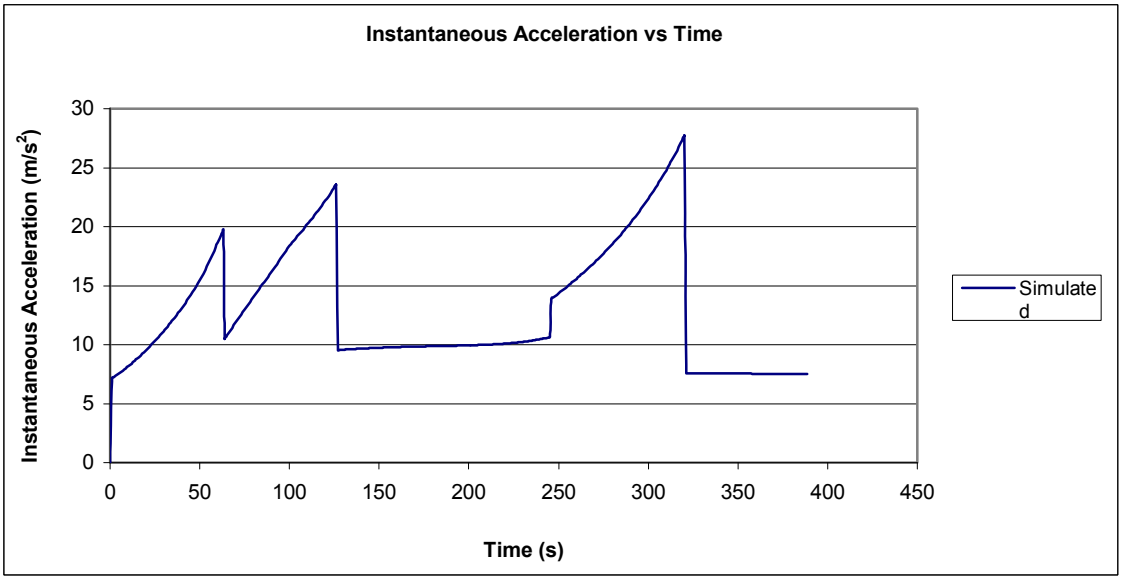
DELTA II MISSION PLOTS (2/3)





DELTA II MISSION PLOTS (3/3)

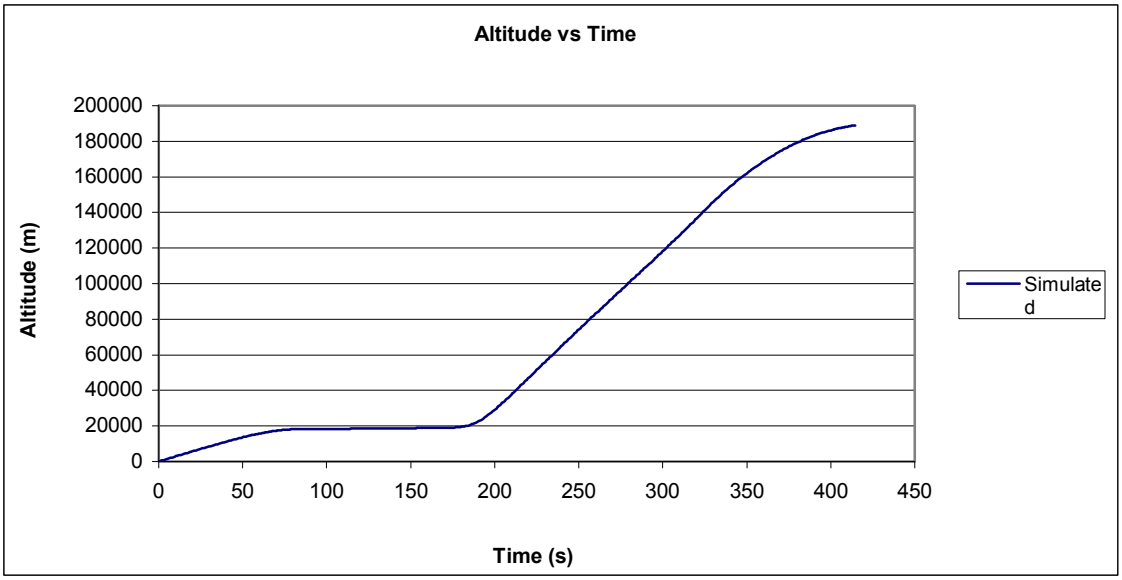


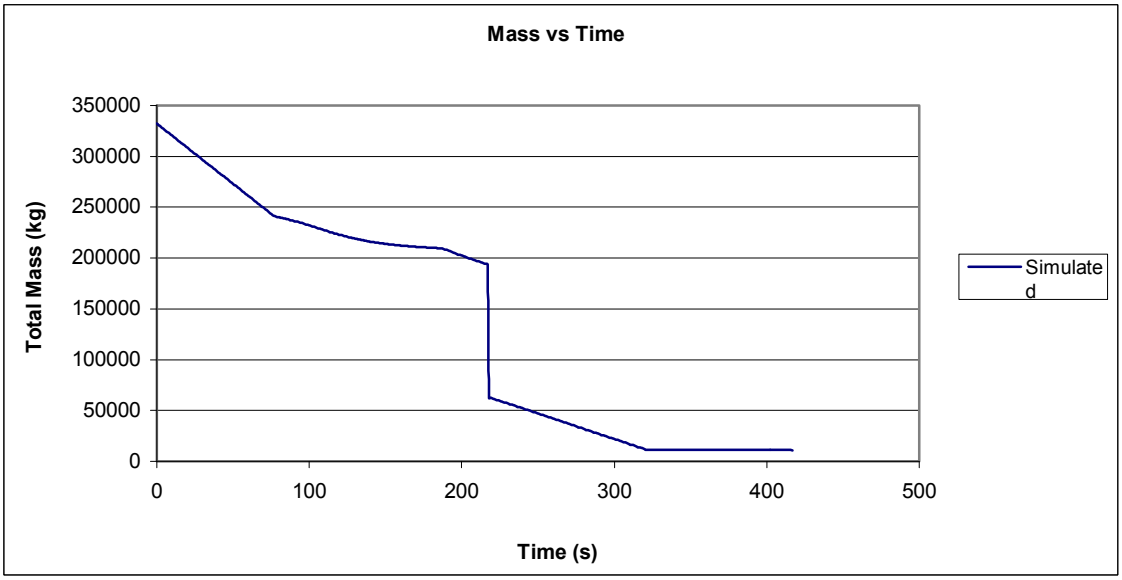


RABCC MISSION INPUT DECK

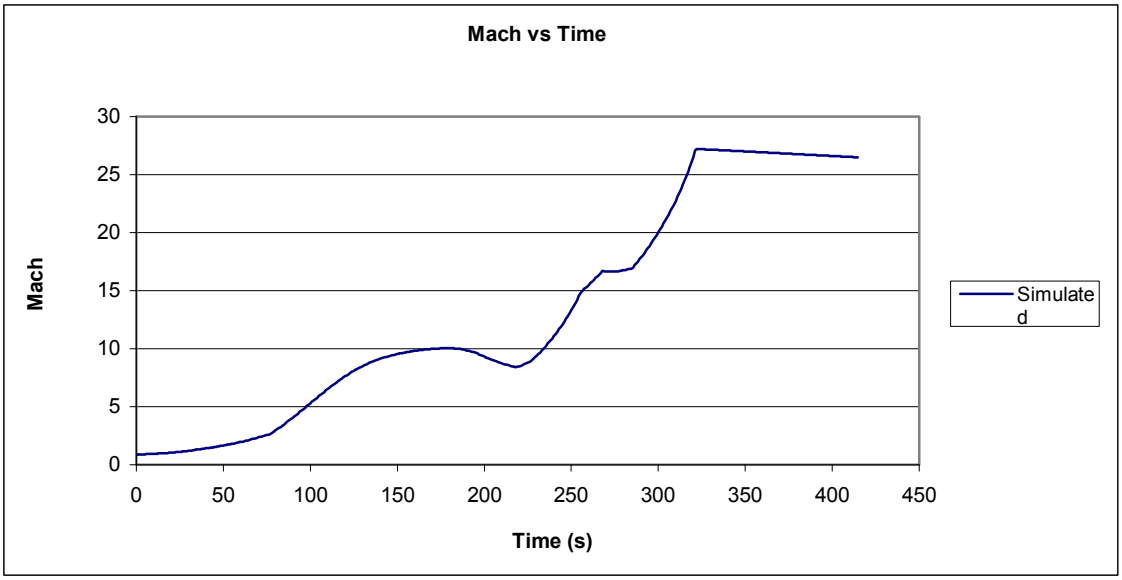
```
----------------------------------------------MODEL SPECIFICATIONS-----------------------------------------
1                                   ! Atmosphere Model (0 - Exponential, 1 - 7-part)
1                                   ! Drag Model (0 - Basic, 1 - Oblique S/E)
1                                   ! Earth Model (0 - Flat Earth, 1 - Round Earth)
1                                   ! Gravity Model for g (0 - Sea-level, 1 - g(alt))
1                                   ! Lift Model (0 - Small Angle, 1 - Oblique S/E)
1                                   ! Vehicle Model (0 - SSR, 1 - RABCC, 2 - MSR)
1                                   ! Trajectory Model (0 - CAA, 1 - SA CAA, 2 - SA AC)
----------------------------------------------MISSION SPECIFICATIONS---------------------------------------
1                                   ! Output File Writing (0 - Off, 1 - On)
100                                 ! Print every # iterations
0.01                                ! Time Step (s)
10000.0                             ! Total Time (s)
0.0                                 ! Initial Altitude (m)
189000.0                            ! Orbital Altitude (m)
138000.0                            ! Burnout Altitude (m)
1.4                                 ! Ratio of Specific Heats
287.0                               ! Gas Constant (J/kg-K)
-------------------------------------GENERAL VEHICLE SPECIFICATIONS-------------------------------------
300.0                               ! Initial Velocity (m/s)
0.0                                 ! Initial Wing Angle of Attack (deg)
73.0                                ! Launch Angle (<= 90 deg)
0.001                               ! Delta Alpha Correction (rad)
10.0                                ! Max Change in Wing Angle Limit (deg)
30.0                                ! Max Wing Angle (deg)
90.0                                ! Max Vehicle Angle to Horizon (<= 90 deg)
0.0                                 ! Min Vehicle Angle to Horizon (<= 0 deg)
368.6                               ! Planform Area of Wings/Lifting Body (m^2)
31.9094                             ! Cross-section Area of Vehicle (m^2)
0.34                                ! Coefficient of Drag for Body (Cdb)
---------------------------------AIR-BREATHING VEHICLE SPECIFICATIONS---------------------------------
4.0                                 ! Constant AB Wing Angle (deg)
18000.0                             ! R-AB Transition Altitude (m)
0.005                               ! R-AB Rotational Rate of Wing (degs/sec)
0.0005                              ! R-AB Initial Smoothing Alpha Limit (deg)
0.0004                              ! R-AB Smoothing Alpha Limit (deg)
0.008                               ! AB-R Rotational Rate of Wing (degs/sec)
0.005                               ! AB-R Smoothing Alpha Limit (deg)
10.0                                ! AB-R Transition Mach Number (4-15)
45000.0                             ! AB-R Transition Altitude (m)
10.0                                ! AB-R Transition Max Horizon Vehicle Angle (deg)
8.0                                 ! R-Ballastic Transition Max Wing Angle (deg)
2.75                                ! Absolute Maximum Wing Angle Scale Factor
4.35e7                              ! Heating Value of Fuel (J/kg)
5000.0                              ! Total Temperature at Combustor Exit (K)
1005.0                              ! Specific Heat at Constant Pressure (J/kg-K)
1.0                                 ! Ratio of Capture Area to Cross-sectional Area
0.25                                ! AB Body Factor (multiplied by Cdb)
-------------------------------------ROCKET VEHICLE SPECIFICATIONS-------------------------------------
1182.63                             ! Mass Flow Rate of Main Rocket Propellant (kg/s)
500.0                               ! Mass Flow Rate of Final Rocket Propellant (kg/s)
3236.42                             ! Main Rocket Effective Exhaust Velocity (m/s)
3297.141                            ! Final Rocket Effective Exhaust Velocity (m/s)
1.0                                 ! Rocket Accelerator Efficiency (>0, <1)
2                                   ! Number of Rocket Stages for MSR
----------------------------------------------MASS SPECIFICATIONS-----------------------------------------
332193.1                            ! Total Vehicle Mass (including propellant) (kg)
132885.0                            ! Structural (System) Mass (kg)
2032.1                              ! Payload Mass (kg)
0.0                                 ! AB Initial Rocket Mass Drop (structural) (kg)
```

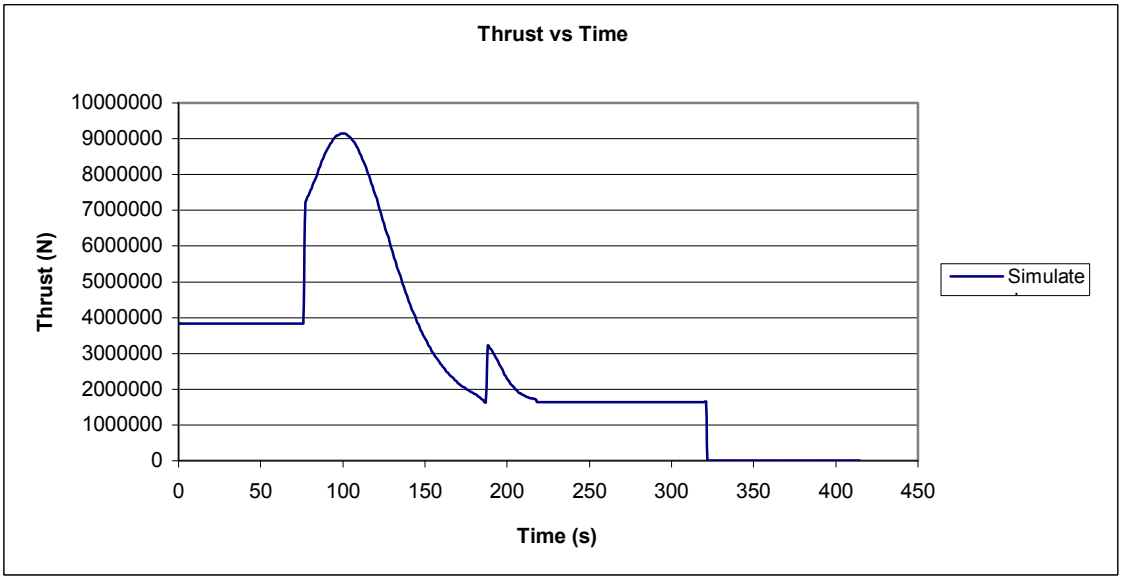
RABCC MISSION PLOTS (1/4)



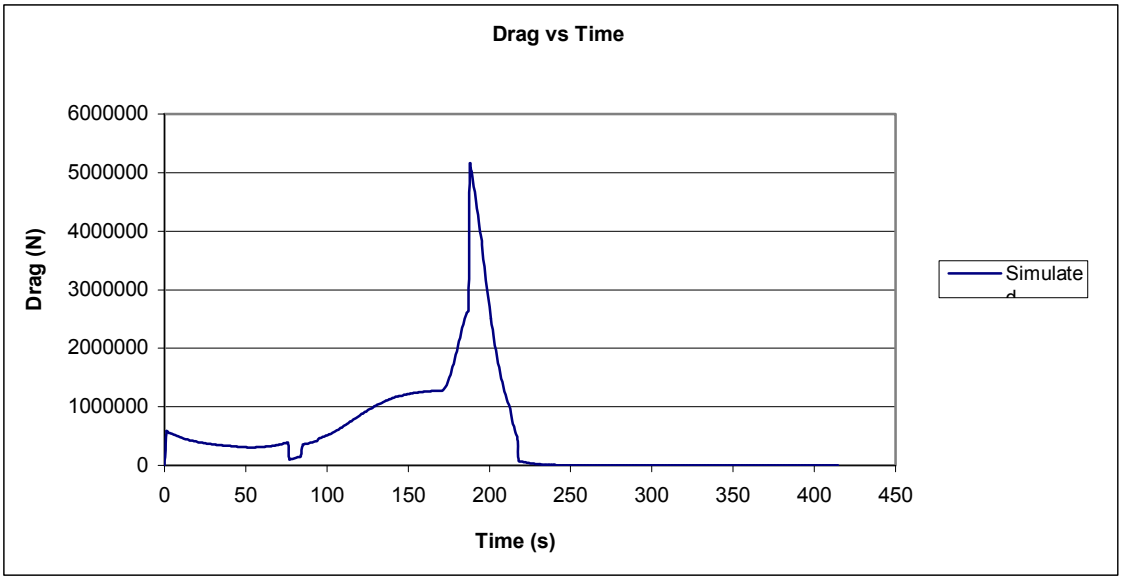


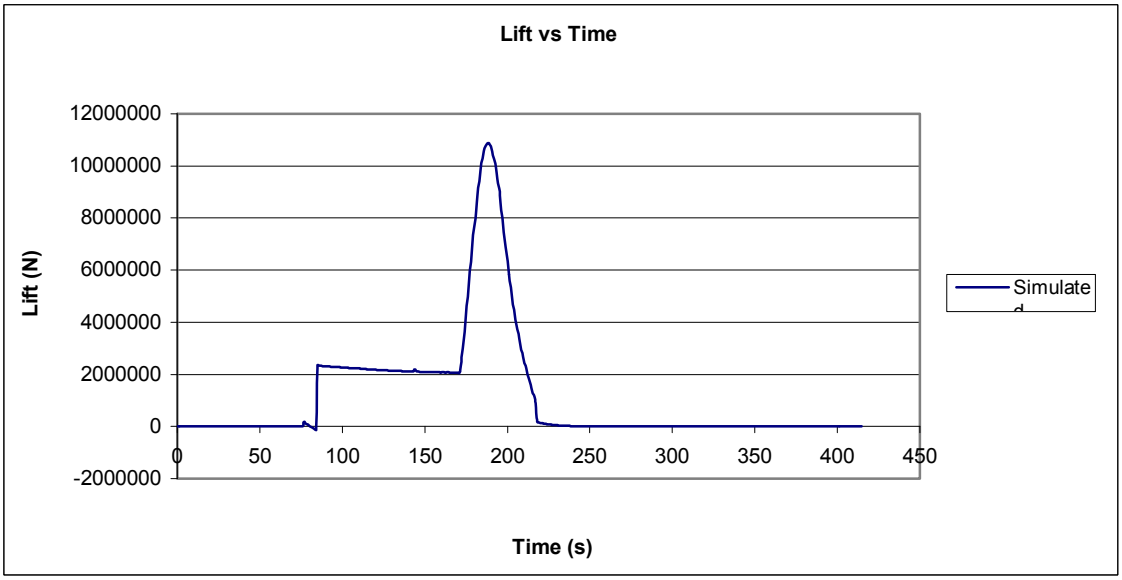
RABCC MISSION PLOTS (2/4)



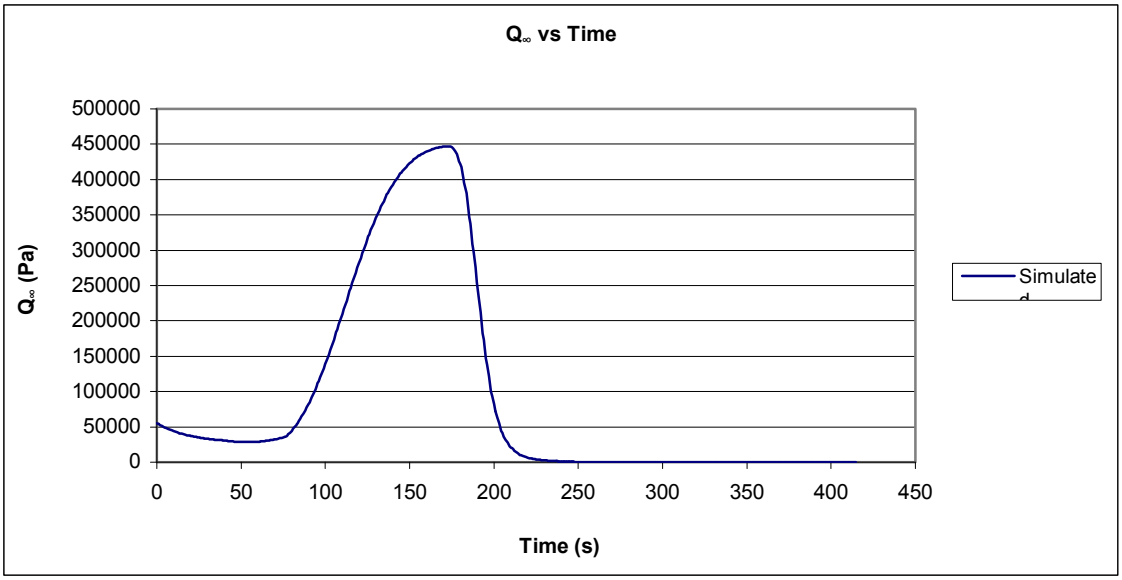


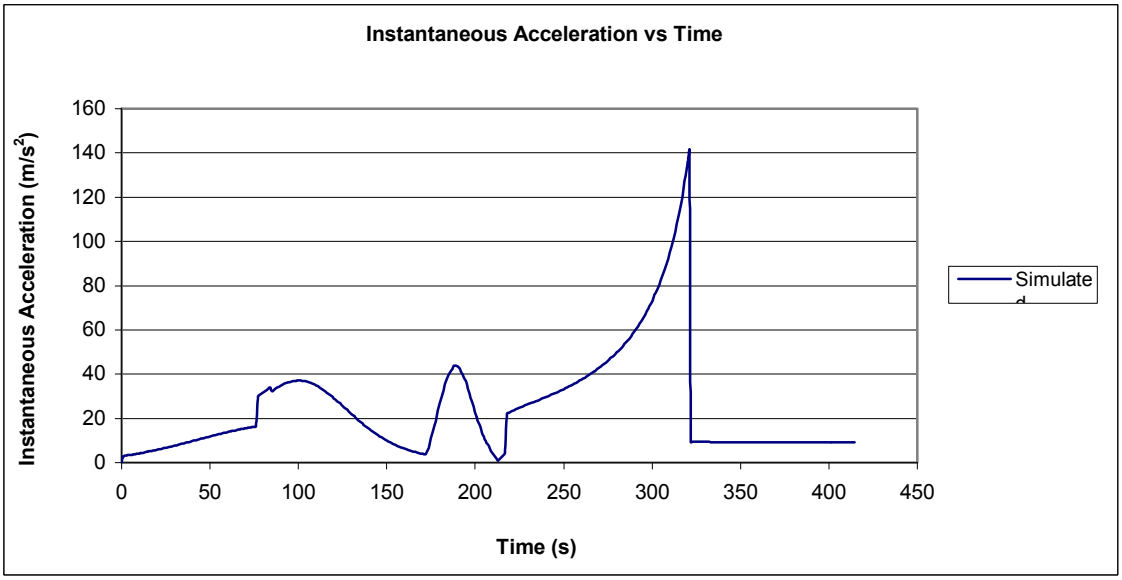
RABCC MISSION PLOTS (3/4)





RABCC MISSION PLOTS (4/4)





BIBLIOGRAPHY

[1] Brauer, G.L., Cornick, D.E., and Stevenson, R., "Capabilities and Applications of the Program to Optimize Simulated Trajectories." NASA CR-2770, Feb. 1977.

[2] Hargraves, C.R., Paris, S.W., and Vlases, W.G., "OTIS Past, Present, and Future," AIAA 92-4530. 1992.

[3] Windhorst, R., Galloway, E., Lau, E., Saunders, D., and Gage, P., "Aerospace Vehicle Trajectory Design and Optimization Within a Multi-Disciplinary Environment," AIAA Paper 2004-704, January 2004.

[4] Orloff, B. S., "A Comparative Analysis of Single-Stage-to-Orbit Rocket and Air-Breathing Vehicles," Master's Thesis, Air Force Institute of Technology, WPAFB, Ohio, June 2006.

[5] Dissel, A. F., Kothari, A. P., and Lewis, M. J., "Investigation of Two-Stage-to-Orbit Airbreathing Launch-Vehicle Configurations," *Journal of Spacecraft and Rockets*, Vol. 43, No. 3, May-June 2006, pp. 568-574.

[6] J.E. Bradford, J.G. Wallace, A.C. Charania, D.R. Eklund, "Quicksat: A Two-Stage to Orbit Reusable Launch Vehicle Utilizing Air-Breathing Propulsion for Responsive Space Access," Presented at SPACE 2004 Conference and Exposition, September 2004.

[7] Trefny, C. J., "An Air-Breathing Launch Vehicle Concept for Single-Stage-to-Orbit," NASA/TM-1999-209089, AIAA-99-2730, May 1999.

[8] Olds, J. and Biltgen, P., "StarRunner: A Single-Stage-to-Orbit, Airbreathing, Hypersonic Propulsion System," AIAA Publication, 2004.

[9] Oswatitsch, K., *Gas Dynamics*, Academic Press Inc., New York, 1956, Chap. 4.

[10] Foa, J., *Elements of Flight Propulsion*, Wiley and Sons, New York, 1960, Chapter 13.

[11] Curran, T. and Craig, R., "The Use of Stream Thrust Concepts for the Approximate Evaluation of Hypersonic Ramjet Engine Performance," USAF Aero-Propulsion Laboratory TR-73-38, 1978.

[12] Riggins, D., McClinton, C.R., and Vitt, P., "Thrust Losses in Hypersonic Engines Part 1: Methodology," *AIAA Journal of Propulsion and Power*, Vol. 13, No. 2, 1997, pp. 281-287.

[13] Riggins, D., "Thrust Losses in Hypersonic Engines Part 2: Applications," *AIAA Journal of Propulsion and Power*, Vol. 13, No. 2, 1997, pp. 288-295.

[14] Roth, B., "Comparison of Thermodynamic Loss Models Suitable for Gas Turbine Propulsion," *AIAA Journal of Propulsion and Power*, Vol. 17, No. 2, 2001, pp. 324-332.

[15] Roth, B., "A Work Potential Perspective of Engine Component Performance," *AIAA Journal of Propulsion and Power*, Vol. 18, No. 6, 2002, pp. 1183-1190.

[16] Clarke, J., and Horlock, J., "Availability and Propulsion," *Journal of Mechanical Engineering Science*, Vol. 17, No. 4, 1975, pp. 223-232.

[17] Czysz, P. and Murthy, S., "Energy Analysis of High-Speed Flight Systems," *High-Speed Flight Propulsion Systems*, edited by T. Curran and S. Murthy, Progress in Astronautics and Aeronauticss, AIAA, Washington, D.C., 1991, pp. 143-236.

[18] Riggins, D., "Evaluation of Performance Loss Methods for High-Speed Engines and Engine Components," *AIAA Journal of Propulsion and Power*, Vol. 13, No. 2, 1997, pp. 296-304.

[19] Riggins, D., "High-Speed Engine/Component Performance Assessment Using Exergy and Thrust-Based Methods," NASA Contractor Report - 198271, 1996.

[20] Moorhouse, D., "A Proposed System-Level Multidisciplinary Analysis Technique Based on Exergy Methods," AIAA Journal of Aircraft, Vol. 40, No. 1, 2003, pp. 10-15.

[21] Moorhouse, D., Hoke, C., and Prendergast, J., "Thermal Analysis of Hypersonic Inlet Flow with Exergy-Based Design Methods," Journal/Date?

[22] Riggins, D., Moorhouse, D., and Taylor, T., "Methodology for Performance Analysis of Aerospace Vehicles using the Laws of Thermodynamics," *AIAA Journal of Aircraft*, Vol. 43, No. 4, 2006, pp. 953-963.

[23] Riggins, D., Camberos, J., and Baker, M., "The Analysis of Power Losses and Wake Entropy Production for Hypersonic Flight Vehicles," AIAA Paper 2006-7904, November 2006.

[24] Ozcan, O., Edis, F. O., and Aslan, A. R., "Inverse Solutions of the Prandtl-Meyer Function," *AIAA Journal of Aircraft*, Vol. 31, No. 6: Engineering Notes, 1994, pp. 1422-1424.

[25] J. Allday, "Apollo in Perspective: Spaceflight Then and Now," Taylor and Francis, pp.44-45, March 2000.

[26] "Apollo 11 Lunar Landing Mission Press Kit," http://www.hq.nasa.gov/alsj/a11/a11prskit.html, accessed January 28, 2008.

[27] Boeing Company, "Boeing Delta II Payload User's Guide," data taken from http://www.spaceandtech.com/spacedata/elvs/delta2_specs.shtml, accessed February 6, 2008.

[28] Riggins, D.W., "The Thermodynamic Continuum of Jet Engine Performance; The Principle of Lost Work due to Irreversibility in Aerospace Systems," *International Journal of Thermodynamics*, Vol. 6, No. 3, 2003, pp. 107-120.

[29] Liepmann, H. W., Roshko, A., *Elements of Gasdynamics*, John Wiley and Sons, Inc., New York, 1957.

[30] Anderson, J. D., *Modern Compressible Flow*, McGraw-Hill Inc., New York, 2003.

[31] Anderson, J. D., *Fundamentals of Aerodynamics*, McGraw-Hill Inc., New York, 2001.

[32] Anderson, J. D., *Hypersonic and High Temperature Gas Dynamics*, American Institute of Aeronautics and Astronautics, Inc., Virginia, 2000.

[33] Mattingly, J. D., *Elements of Propulsion: Gas Turbines and Rockets*, American Institute of Aeronautics and Astronautics, Inc., Virginia, 2006.

[34] Oates, G. C., *Aerothermodynamics of Gas Turbine and Rocket Propulsion*, American Institute of Aeronautics and Astronautics, Inc., Virginia, 1997.

VITA

Tyler Forrest Winter was born in Dubuque, Iowa on April 15, 1985. He lived in Iowa, Illinois, Florida, Tennessee, and Missouri, but spent most of his life in the St. Louis area. After graduating from Waterloo High School in 2003, he attended the University of Missouri-Rolla. During his undergraduate career, he was an active member of Tau Beta Pi, Sigma Gamma Tau, the Mathematical Association of America at UMR, the Student Union Board and held the position of Chair for the American Institute of Aeronautics and Astronautics chapter. He also served as a peer learning assistant in the Learning Enhancement Across Disciplines program. He began conducting research under Dr. Riggins in 2004 and continued throughout his graduate career. He graduated Summa Cum Laude with a Bachelor of Science in Aerospace Engineering with a minor in Mathematics in May of 2007. He received his Master of Science Degree in Aerospace Engineering at MS&T in May of 2008.