Ryerson University
Faculty of Engineering and Architectural Science
Department of Aerospace Engineering

A Mathematical Method for Predicting the Design Performance of Single and Multi-stage Rockets

Alexander Chang

AER870
Aerospace Engineering Thesis
Faculty Advisor - Dr. Jeffrey Yokota
13/04/2020

Acknowledgements

The Author would like to thank Dr. Yokota for his consultation and support on this project. The feedback given was invaluable which helped push this project to achieve its desired results.

The Author would also like to thank Ryerson University and the Department of Aerospace Engineering for providing this opportunity to work on an Undergraduate Thesis project. Furthermore, the Author would like to thank Ryerson Rocketry Club for providing the flight data.

Abstract

Methods for predicting the performance of rockets are not new, however they often exist only within private organizations and in order to ensure competitive advantage, organizations tend to not share any details about their inner performance models. This open-source method gives students, design-teams and hobbyists a method to obtain baseline approximations for the performance of both single and multi-stage tandem rockets and provides a method which can easily be modified to meet the end-user's requirements. The method solves for the mass, flight-path angle, velocity, altitude, and down-range distance using a numerical integrator to solve a set of nonlinear ordinary differential equations.

Contents

List of Figures

List of Tables

Nomenclature

C_D	Coefficient of drag, dimensionless parameter
$Eq.$	Shorthand for Equation
$Fig.$	Shorthand for Figure
I_{sp}	Specific Impulse, s
IVP	Initial Value Problem
ODE	Ordinary Differential Equation
R	Ideal gas constant, 287 J/kg K
R_E	Radius of the Earth
ROC	Rate of Change
$SATP$	Standard Ambient Temperature and Pressure

1 Preface

The purpose of this project is to design an open-source method to compute a baseline approximation for the performance of both single and multi-stage tandem rockets up to and including orbit. However, the method is only capable of calculating gravity turns, and any type of thrust-vectoring maneuvers will require the source code to be modified. Furthermore, as it is a flight-prediction method for rockets, once the rocket reaches orbit, any delta-V maneuvers desired by the user and determination of orbital parameters will have to be done by either manual modification to the code or with a different method altogether.

This document covers the proper use of the code provided, the equations and methods used to carry out the calculations, and an analysis on the precision and accuracy of the results produced. The method is designed to output five key parameters, namely mass, velocity, altitude, flight-path angle, and down-range distance.

2 Literature Review

2.1 Similar Existing Methods

The development of flight prediction methods are not new, and exist within many private organizations. Additionally, design tools such as open-rocket are easily accessible to the public for model rocketry, however its accuracy comes into question as it is an idealized case and disregards many variables which have an effect on the rocket's overall performance. Furthermore, many open-source methods are only sufficient for single-stage rockets and do not provide accurate predictions rockets reach space.

2.2 Numerical Integrators

Various methods for numerical integration exist, however the methods being examined are those that are valid for solving IVP's. The numerical methods being used are for forward integration, thereby using data at step i and $i{+}1$ to estimate the derivative. The solution at $i{+}1$ is then taken as the new 'starting point' which will then be used to calculate the solution until a stopping condition is reached [1, p.110-111].

2.2.1 Fixed Step-Size Integrators

One benefit of fixed step-size integrators is that they are generally easy to implement when compared to variable step-size integrators. By using higher-order integrators or integrators with more stages, this often reduces the truncation/numerical error [1, p.553-579]. With regards to computational time, consider the Forward Euler method and the Runge-Kutta 4th (RKF4) Order methods of integration:

Forward Euler Method

$$y = y + hk_1; \tag{2.1}$$

$$k_1 = f(t, y); \tag{2.2}$$

Runge-Kutta 4th Order Method

$$y = y + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.3}$$

$$k_1 = f(t, y) \tag{2.4}$$

$$k_2 = f(t + h/2, y + k_1/2) \tag{2.5}$$

$$k_3 = f(t + h/2, y + k_2/2) \tag{2.6}$$

$$k_4 = f(t + h, y + k_3) \tag{2.7}$$

Looking at both the Forward Euler method and the Runge-Kutta 4th Order method, one can easily deduce that the number of computations required for the Forward Euler method is significantly less than the RKF4 method. Where the Forward Euler method can finish computing the system of ODE's with 2 equations, the RKF4 method requires 5. When computed over a small step-size or for flights with a long durations, the RKF4 method will require significantly more time to finish computing, whereas the Forward Euler method will be relatively fast. However as stated previously, using lower-order integrators or larger step-sizes may introduce larger amounts of numerical error.

Several other numerical integrators that were used for testing include the Explicit Midpoint method, 3 stage method, and Butcher's 5th Order Runge-Kutta method. These additional methods are included in the code for the end-user to easily switch between.

Explicit Midpoint Method

$$y = y + hk_2 \tag{2.8}$$

$$k_1 = f(t, y) \tag{2.9}$$

$$k_2 = f(t + \frac{h}{2}, y + \frac{k_1}{2}) \tag{2.10}$$

3-Stage Method

$$y = y + h(\frac{k_1}{6} + k_2\frac{2}{3} + \frac{k_3}{6}) \tag{2.11}$$

$$k_1 = f(t, y) \tag{2.12}$$

$$k_2 = f(t + \frac{h}{2}, y + \frac{k_1}{2}) \tag{2.13}$$

$$k_3 = f(t + h, y - k_1 + 2k_2) \tag{2.14}$$

5th Order Runge-Kutta Method

$$y = y + \frac{h}{90}(7k_1 + 32k_3 + 12k_4 + 32k_5 + 7k_6) \tag{2.15}$$

$$k_1 = f(t, y) \tag{2.16}$$

$$k_2 = f(t + \frac{h}{4}, y + h\frac{k_1}{4}) \tag{2.17}$$

$$k_3 = f(t + \frac{h}{4}, y + h\frac{k_1}{8} + h\frac{k_2}{8}) \tag{2.18}$$

$$k_4 = f(t + \frac{h}{2}, y - h\frac{k_2}{2} + hk_3) \tag{2.19}$$

$$k_5 = f(t + h\frac{3}{4}, y + k_1 h\frac{3}{16} + k_4 h\frac{9}{16}) \tag{2.20}$$

$$k_6 = f(t + h, y - k_1 h\frac{3}{7} + k_3 h\frac{2}{7} - k_4 h\frac{12}{7} + k_5 h\frac{8}{7}) \tag{2.21}$$

2.2.2 Variable Step-Size Integrators

Variable step-size integrators work in a similar fashion to fixed step-size integrators, however, the step-size changes with each step depending on the total error between steps specified by the user. This can result in significant time savings when conducting calculations over a long duration (i.e. calculations over many orbital periods). Unlike the fixed step-size integrators, variable step-size integrators are significantly harder to implement due to its added complexity.

2.3 Publicly Available Flight Data and Rocket Parameters

Basic information on the geometry (such as diameter, height, I_{sp} etc.) of many launch vehicles are easily accessible, however these values only give a rough estimate to the actual performance of the rocket. Information regarding the drag profile, thrust curve, and other data regarding to actual flights are often confidential and thus unattainable. This creates a major hurdle for this project when it comes to verifying the results due to the small sample size of data available. This issue will be discussed further in 5.3.

3 Flight Model Design

Section 3 outlines the design of the rocket's flight model and the equations used to carry out the calculations. The back-end logic behind the implementation of these equations and the numerical solution can be found in [2].

3.1 Rocket Vehicle Dynamics

3.1.1 Equations of Motion

Section 3.1.1 will refrain from deriving all the equations of motion as this has already been done in numerous texts before. However, those seeking the derivations can find them in [2, p.706-711]. Before the equations can be derived, a reference plane must first be defined.

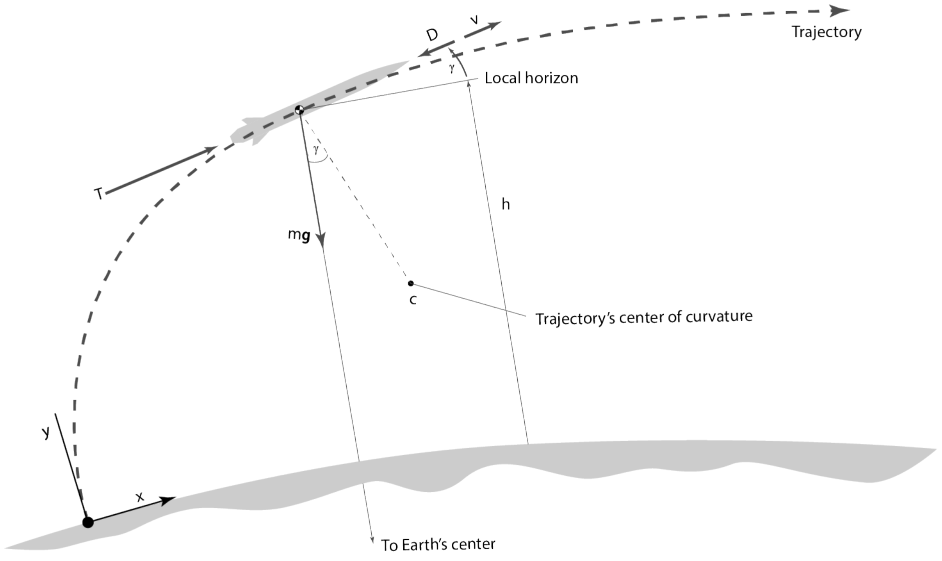


Figure 3.1: Reference Plane of the Rocket

With the reference plane defined, the following equations are listed in the order of which they are to be solved.

$$a_n = \frac{dv}{dt} = \frac{T - D}{m} - g \sin \gamma \tag{3.1}$$

Eq. 3.1 is the normal component of the acceleration. Alternatively, it can be written as $\frac{dv}{dt}$; the change in velocity over the change in time, where T is the thrust produced by the rocket's motor/engine in N, D is the drag force in N on the rocket which acts in opposing normal direction of the vehicle's flight path, m is the mass of the rocket in kg, g is the gravitational constant of the Earth in m/s^2, and γ is the flight path angle in radians.

$$\dot{\gamma} = \frac{d\gamma}{dt} = -\frac{1}{v}(g - \frac{v^2}{R_E + h}) \cos\gamma \tag{3.2}$$

Eq. 3.2 is the change in the flight path angle over the change in time, where v is the velocity of the rocket in m/s, g is the gravitational constant of the Earth in m/s^2, R_E is the radius of the Earth in m, h is the altitude of the rocket in m, and γ is the flight path angle in radians.

$$\dot{h} = \frac{dh}{dt} = v \sin\gamma \tag{3.3}$$

Eq. 3.3 is the change of the rocket's altitude over the change in time, where v is the velocity of the rocket in m/s and γ is the flight path angle in radians.

$$\dot{x} = \frac{dx}{dt} = \frac{R_E}{R_E + h} - v \cos\gamma \tag{3.4}$$

Eq. 3.4 is the change of the rocket's downrange distance over the change in time, where R_E is the radius of the Earth in m, h is the altitude of the rocket in m, v is the velocity of the rocket in m/s, and γ is the flight path angle in radians.

In the equations listed above, most of the parameters are not constant and change over the course of the rocket's flight. Due to these parameters constantly changing, there is a need for an iterative method to accurately compute the parameters over the duration of flight.

3.1.2 C_D as a Function of Velocity and Geometry

Another major factor on rocket performance is drag which is defined as:

$$\frac{1}{2}\rho v^2 A C_D \tag{3.5}$$

Where ρ is the density of the air in kg/m^3, v is the velocity of the rocket in m/s, A is the cross-sectional reference area of the rocket in m^2, and C_D is the drag coefficient which is a dimensionless term. Due to the drag coefficient being dependent on both the external geometry and free-stream velocity of the rocket, the drag coefficient can only be found either through wind tunnel testing or a CFD analysis.

3.1.3 Thrust and Mass Flow

Rocket motors/engines which are commercially available often have thrust curves associated with it outlining the net thrust over the burn duration. If a thrust curve is given, then determining the thrust at time t during a rocket's flight is trivial. However, if the thrust curve is unavailable, the thrust can be approximated by using a rocket's thrust-to-weight ratio.

$$T = \frac{T}{W} mg \tag{3.6}$$

Eq. 3.6 is the thrust of the rocket in N where $\frac{T}{W}$ is the thrust-to-weight ratio which is dimensionless, m is the mass of the rocket in kg and g is the gravitational acceleration in m/s^2.

$$\dot{m} = \frac{-T}{I_{sp} g} \tag{3.7}$$

Eq. 3.7 is the mass flow rate of the rocket's propellant, where T is the total thrust in N, I_{sp} is the specific impulse of the rocket engine in s, and g is the gravitational acceleration in m/s^2.

3.2 Earth's Gravitational Field

To further improve the accuracy of the model, Earth's gravitational field cannot be assumed to be constant for rockets reaching high-altitudes. However, the assumption is made that the Earth is perfectly spherical, thus the gravitational constant for any given altitude is the same throughout the Earth.

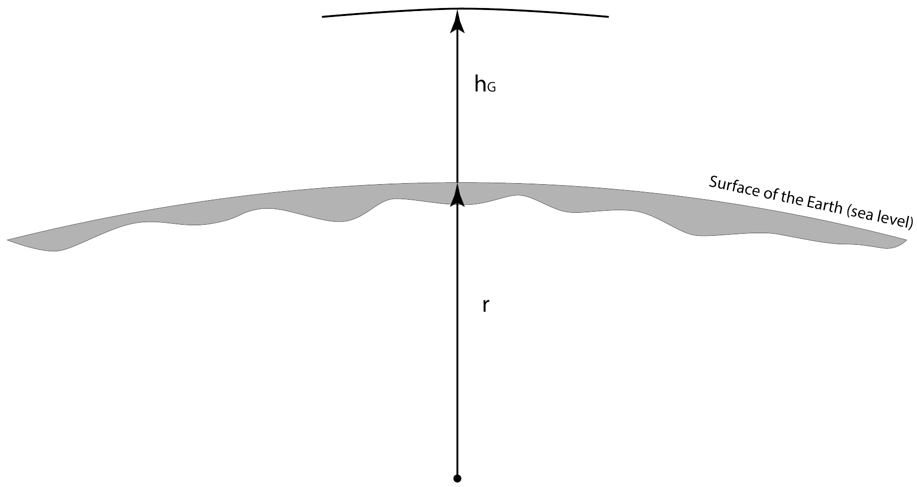


Figure 3.2: Defining the Parameters of Earth's Gravitational Model

$$g = \frac{g_0}{1 + (\frac{h_g}{r})^2} \tag{3.8}$$

In Eq. 3.8 g is the gravitational constant for a given altitude in m/s^2, h_g is the altitude in m, and r is the radius of the Earth in m. The full derivation of Eq. 3.8 can be found in [3, p.113].

3.3 The Atmospheric Model

The calculations conducted for determining the atmospheric temperature and density are done so with the geopotential altitude, which accounts for the variation of gravity due to the latitude and altitude at sea level. Additionally, the atmospheric model used for this simulation takes the conditions at the launch site (namely the temperature, T and air density ρ) into account. Those interested in seeing the full derivation of the equation used in Section 3.3 can find it in [3, p.110-121].

3.3.1 Atmospheric Temperature

The temperature of the atmosphere can be classified into two distinct categories; isothermal layers where the temperature is constant, and gradient layers where temperature changes linearly. The isothermal layers lies between 11 to 25 km, 47 to 53 km and 79 to 90 km, with the temperatures being 216.66 K, 282.66 K, and 165.66 K respectively. Fig. 3.3 outlines where the isothermal and gradient layers lay within the atmosphere.

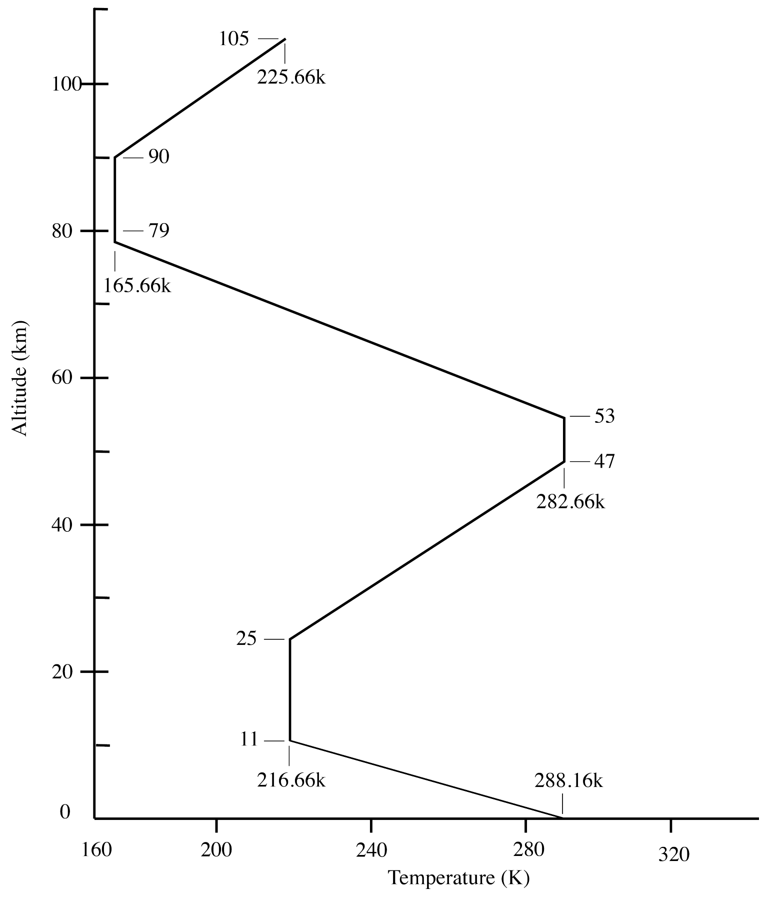


Figure 3.3: Standard Atmospheric Model

Within the gradient layers, the lapse-rate can be represented with the following equation.

$$a_1 = \frac{T_2 - T_1}{h_1}; \tag{3.9}$$

Where a is known as the lapse rate and is dimensionless, T are the temperatures at the beginning and end of the region respectively in K, and h is the altitude in m. With this relationship, the lapse rate in region 1 (a_1) can then be changed accordingly for the launch site ambient temperatures instead of using the standard temperature. To help visualise this, refer to Fig. 3.4 below:

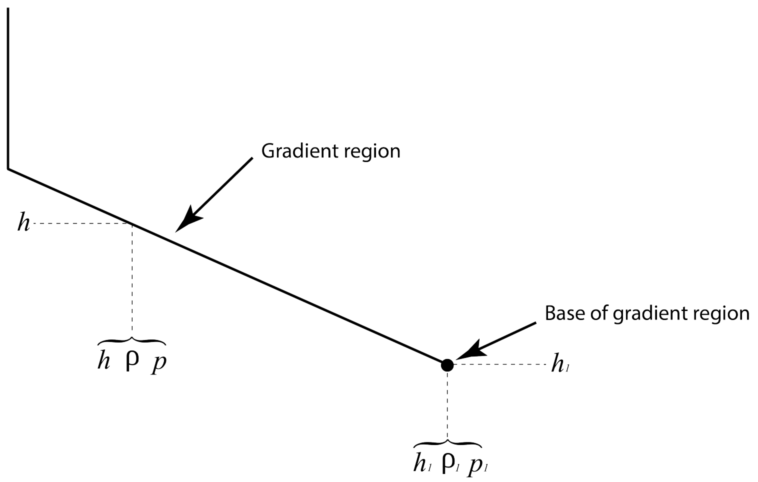


Figure 3.4: Gradient Layer Parameters

The temperature at within any region can be found with the following relationship, where T_1 is the temperature at the beginning of region 1 in K, a is the lapse rate of the region, and h is the current altitude of the rocket in m.

$$T = T_1 + a_1 h; \tag{3.10}$$

3.3.2 Atmospheric Density

Once the temperature and the lapse rate is known, the atmospheric density can then be approximated with the following equation.

$$\rho = \rho_1 (\frac{T}{T_1})^{-\frac{g_0}{a_1 R}+1} \tag{3.11}$$

Where ρ is the density at a given altitude and ρ_1 is the density at the beginning of region 1 in kg/m^3, T_1 is the temperature at the beginning of region 1 in K, g_0 is the gravitational constant at sea level in m/s^2, a_1 is the lapse rate which is dimensionless, and R is the gas constant.

To calculate the pressure and temperatures at each subsequent region, the end conditions from the previous region are used as the new initial conditions. For example, if region 1 is 0 to 11 km, the temperature and pressure values at 11 km are then used to calculate values within region 2 (11 to 25 km).

4 Integrating Equations into MATLAB

With all of the governing equations and parameters defined in Section 3, Section 4 will outline the implementation of these equations and parameters into the program itself. Several flow charts are included to help illustrate the processes and sequences of events within the code.

4.1 Logic Flow Charts

Fig. 4.1 depicts a high-level overview of key processes of the method while 4.2 outlines the steps inside the integrator. Starting with 4.1, once the initial conditions and constraints are loaded and a numerical integrator is chosen, the method checks to see if there is still available propellant. If propellant still remains (i.e motor burn is still in progress), it checks for a thrust curve and determines the instantaneous thrust available. If there is no more propellant, the program then checks to see if any additional rocket stages remain. If so, the structural mass of the current stage is subtracted from the total mass followed by the 'ignition' of the next stage. Next, the mass flow rate of the propellant is found and the new mass is calculated. With the mass found, the integrator is then run to solve the system of ODE's. If additional stages in the integrator exists, it will then compute all additional stages to completion. The rocket's new parameters are then calculated at the end of the final stage of the integrator, followed by a check to see if the program has reached the stopping conditions.

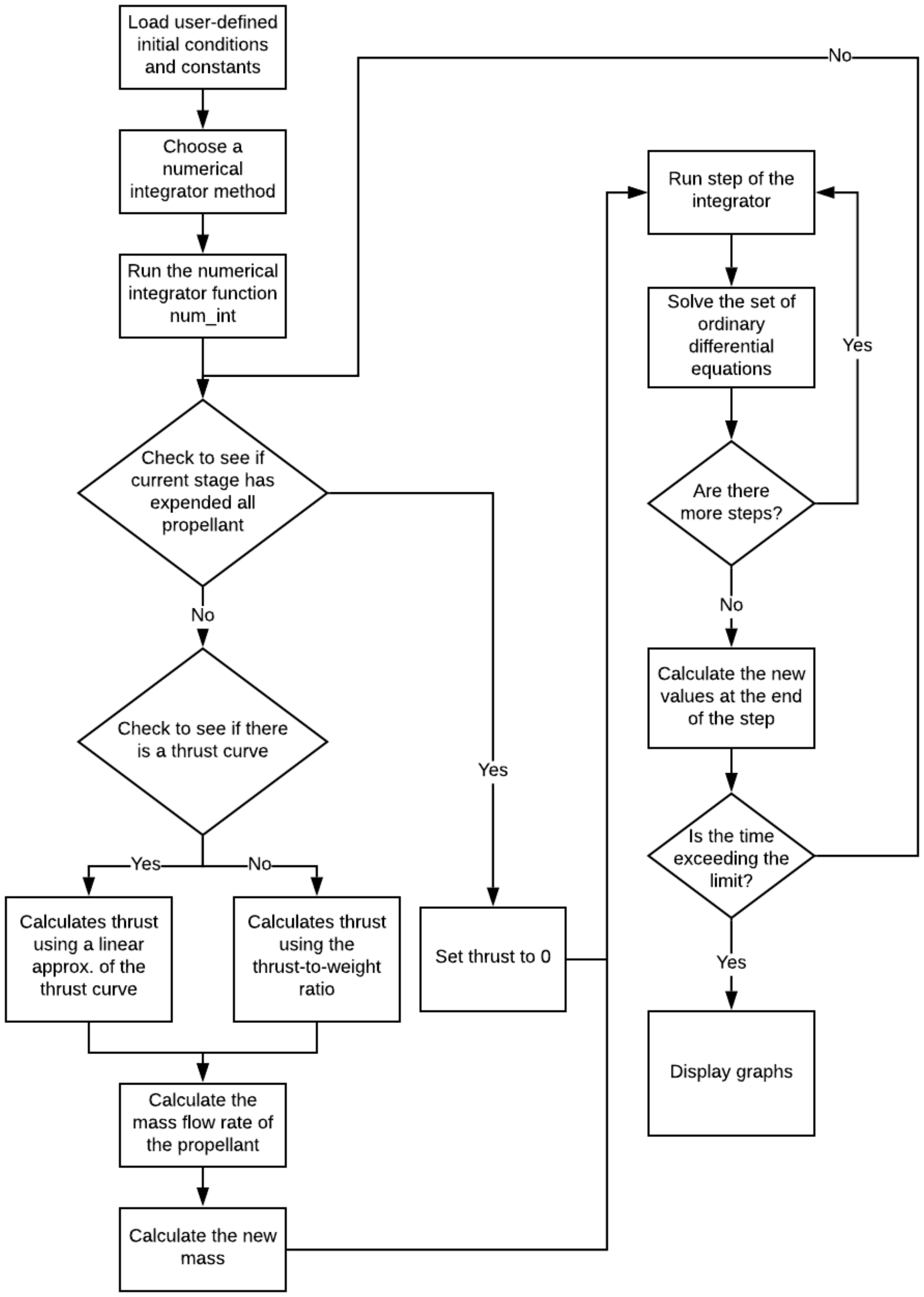

Load user-defined initial conditions and constants
Choose a numerical integrator method
Run the numerical integrator function num_int
Check to see if current stage has expended all propellant
No
Yes
Check to see if there is a thrust curve
Yes
No
Calculates thrust using a linear approx. of the thrust curve
Calculates thrust using the thrust-to-weight ratio
Set thrust to 0
Calculate the mass flow rate of the propellant
Calculate the new mass
Run step of the integrator
Solve the set of ordinary differential equations
Are there more steps?
Yes
No
Calculate the new values at the end of the step
Is the time exceeding the limit?
No
Yes
Display graphs

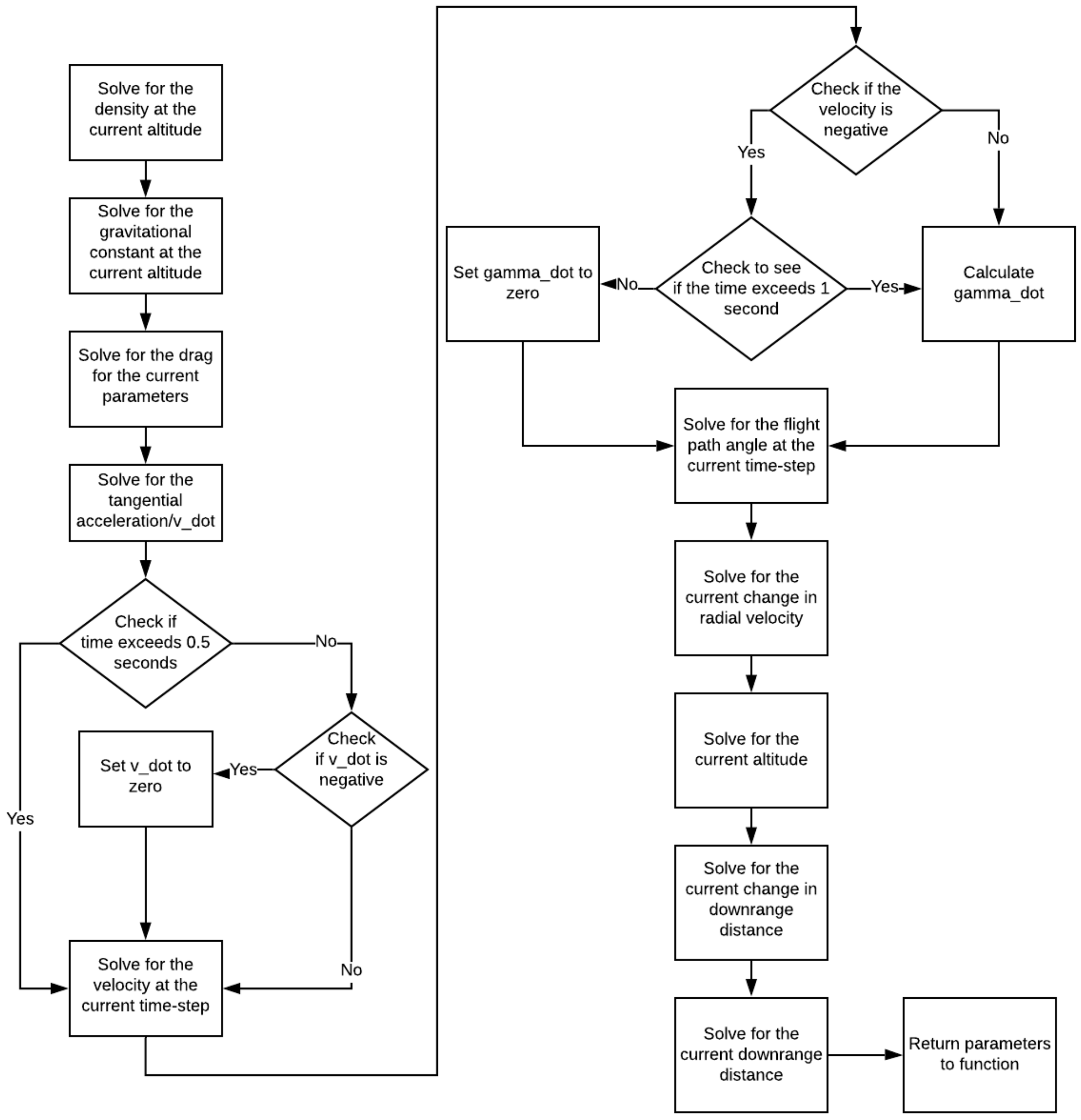
Figure 4.1: High-Level Overview of Program Logic

Fig. 4.2 illustrates the order of operations within the *rates.m* file. As the figure shows, there two Boolean operators after the calculations of $\dot{v}$ and $\dot{\gamma}$. This is to ensure that the program is stable and to prevent the rocket from crashing once the program starts running. Taking the first Boolean operator just after the calculation of $\dot{v}$, for a given step-size, the gravitational force may exceed the rocket's instantaneous thrust term provided by the motor. However, as it is still on the ground, an opposing normal force is needed to prevent the rocket from essentially 'falling through the ground'. A simple statics equation can help visualize this:

$$F_{net} = T + G + N \tag{4.1}$$

While on the ground, the net force on the rocket is a summation of thrust T, gravity G, and the normal force N. There is no drag as the velocity is 0. If the gravitational acceleration term exceeds the instantaneous acceleration provided by the motor, the acceleration ($\dot{v}$) is set to zero as it has not yet taken off. The Boolean check for time exceeding 0.5 seconds is to allow for a negative $\dot{v}$ term later on during flight if the rocket descends.

The second Boolean check just after the calculation of $\dot{\gamma}$ is to prevent the rocket from immediately 'tipping over' due to an imbalance of forces, or if the rocket is starting on a steep launch angle. The remaining blocks in the diagram outline which parameters are being solved using the equations from Chapter 3.



Solve for the density at the current altitude
Solve for the gravitational constant at the current altitude
Solve for the drag for the current parameters
Solve for the tangential acceleration/v_dot
Check if time exceeds 0.5 seconds
No
Yes
Check if v_dot is negative
Yes
Set v_dot to zero
No
Solve for the velocity at the current time-step
Check if the velocity is negative
Yes
No
Check to see if the time exceeds 1 second
No
Set gamma_dot to zero
Yes
Calculate gamma_dot
Solve for the flight path angle at the current time-step
Solve for the current change in radial velocity
Solve for the current altitude
Solve for the current change in downrange distance
Solve for the current downrange distance
Return parameters to function

Figure 4.2: Order of Computations Within the Integrator

4.2 Modifying the method for Tandem Staging

Modelling for the booster separation was done under the assumption that the separation between the stages is instantaneous, with the following stage igniting immediately after. One option considered for modelling the booster separation was to split the problem into two. The first calculation would solve the parameters from launch until burnout, with the second calculation computing the ignition of the second stage until the end of its flight. However, splitting the problem into two and using the final values of the first calculation as the initial values for the second may lead to variations in the solution. This is because there may be interactions between the equations and parameters during the separation that might not be represented properly. To avoid this possibility completely, the entire problem must be as one.

Several test cases were run with multiple stages. The tests done were with the exact same parameters for all stages (i.e. the same motor, C_D, mass etc.).

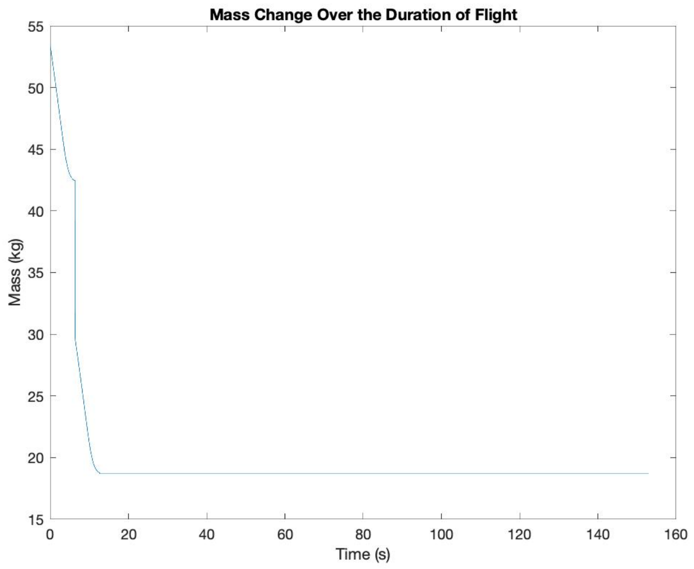


Figure 4.3: Mass Change for a 2-Stage Rocket Test Case

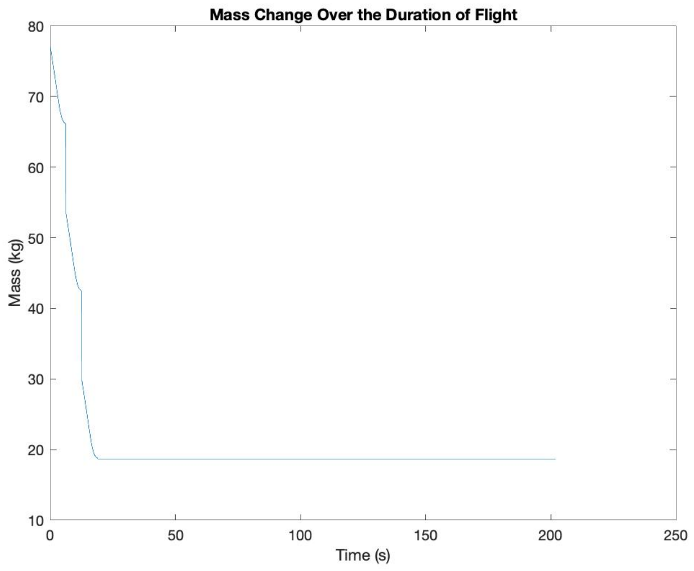


Figure 4.4: Mass Change for a 3-Stage Rocket Test Case

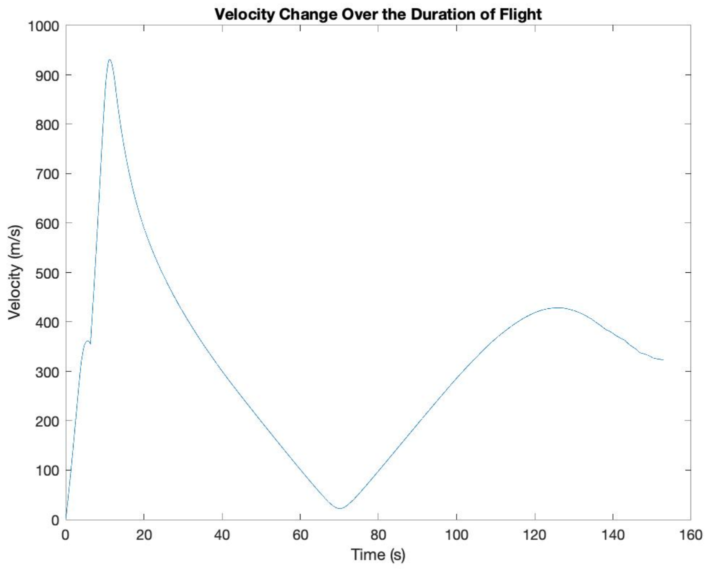


Figure 4.5: Velocity Change for a 2-Stage Rocket Test Case

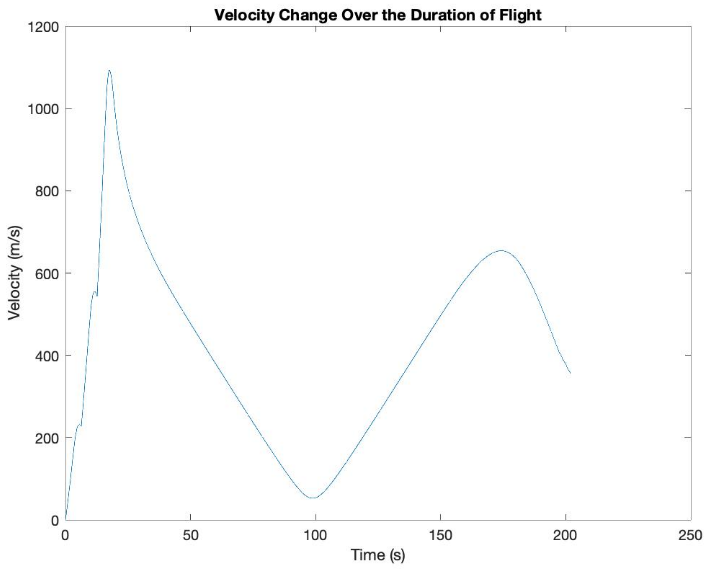


Figure 4.6: Velocity Change for a 3-Stage Rocket Test Case

Fig. 4.3 and 4.4 for both the 2 and 3 stage rockets respectively shows the sudden change in the mass, signifying the instance just after burnout where the first stage is discarded, followed by the immediate ignition of the next stage. Once burnout is reached on the final stage, the mass becomes equivalent to the summation of the structural and payload masses.

Fig. 4.5 and 4.6 shows a significant change in velocity immediately after booster separation which is expected. By taking a look at Newton's Second Law, $F = ma$, as mass decreases then the acceleration term must increase to remain at equilibrium. By extension, velocity must then also increase as velocity is the integral of acceleration.

5 Accuracy and Precision of the Method

Within this section, the accuracy and precision of the method will be critiqued. The accuracy of the method itself can be split into two categories; numerical accuracy and the accuracy between the computed results and test data. Numerical accuracy can be improved by using higher-order integrators or integrators with more stages. Going back to the comparison in 2.2.1, the RKF4 method has a global truncation error on the order of $O(h^4)$ while the Euler method has a global truncation error on the order of $O(h^2)$. However, the accuracy of the results to actual flight data depends on how the flight model is built and how closely it resembles to the behaviour of the rocket through its flight. Furthermore, there may be small nuances between rockets, thus the parameters within the method would need to be tweaked and adjusted. The precision of the method can be measured by seeing how close the results are to each other by tweaking parameters such as the step-size, integrator type, etc.

5.1 Comparison with Hand Calculations

Hand calculations for these iterative solutions produce a reasonable ballpark estimate and can indicate whether the method is providing a reasonable solution. By including a drag term into the calculations, this can help improve the overall results. With hand calculations, a different set of equations must be used as the equations in Chapter 2 are for an iterative method.

5.1.1 Equations for Single Stage Comparison

For the single-stage comparison, a modified version of the equations in [2, p.710-711]the following set of equations were evaluated.

$$v = c\ln\frac{m_o}{m_f} - g_o t - \frac{D}{m_f}t \tag{5.1}$$

$$t = \frac{c\ln\frac{m_o}{m_f}}{g_o + \frac{D}{m_f}} \tag{5.2}$$

Eq. 5.1 yields an approximate value for the velocity given the initial mass m_o in kg, the final mass m_f in kg, the Earth's gravitational acceleration g_o, and drag D in N. The c term is the product of $I_{sp}g_0$. Setting the velocity to zero in Eq. 5.1, it can then be rearranged for time in s to apogee. By taking the integral of Eq. 5.1 with respect to time, the equation can then be solved for apogee by substituting the time to apogee into Eq. 5.6.

$$h = \frac{c}{\dot{m}_e}(m_o\ln\frac{m_f}{m_o} + m_o + m_f) + ct\ln\frac{m_o}{m_f} - \frac{1}{2}g_o t^2 - \frac{1}{2}\frac{D}{m_f}t^2 \tag{5.3}$$

5.1.2 Equations for n-Stage Comparison

For the multi-stage calculations, the final mass m_f of each stage is the sum of all masses above it as illustrated in Fig. 5.1. m_p is the mass of the propellant, m_o is the empty mass, m_e is the structural and electrical mass of the stage, and m_PL is the mass of the payload.

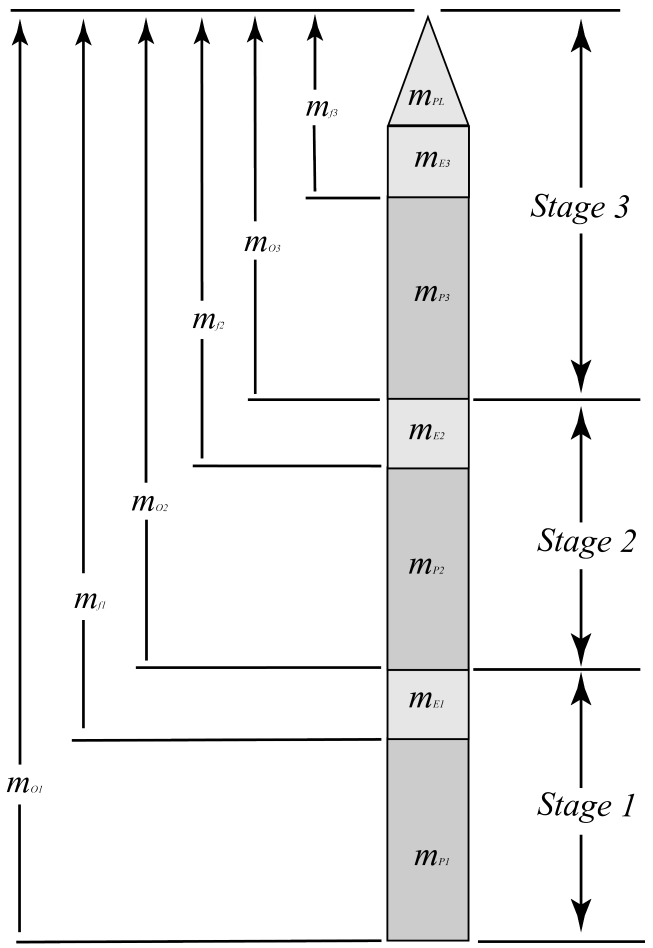


Figure 5.1: Mass Ratios of a 3 Stage Rocket

Before carrying out the following calculations, an assumption is made that the n+1 stage ignites immediately after n stage burns out. With this assumption, the burnout velocity can then be written as the sum of the burnout velocities minus the losses from gravity and drag during the rocket's cruise to apogee.

$$v = \sum_{i=1}^{n} v_{bo,l_i} - \sum_{i=1}^{n} \frac{D_i}{m_f} t_i - g_o t_a \tag{5.4}$$

Note the two different times. t_i denotes the burn time of each individual stage while t_a is time to apogee (both in s). Assuming that the velocity of the rocket is 0 at apogee, the equation can then be rearranged for t_a. With the time to apogee known, the apogee itself can be found taking the integral of Eq. 5.5.

$$h = \int_0^t v = \int_0^t \sum_{i=1}^{n} v_{bo_i,l_i} - \sum_{i=1}^{n} \int_0^t v_D t_i - \int_0^t g_o t_a \tag{5.5}$$

$$h = \sum_{i=1}^{n} \left(\frac{c_i}{\dot{m}_{ei}} (m_{oi} \ln \frac{m_{fi}}{m_{oi}} + m_{oi} + m_{fi}) + c_i t_i \ln \frac{m_{oi}}{m_{fi}} \right) - \frac{1}{2} \sum_{i=1}^{n} \frac{D_i}{m_{fi}} t_i^2 - \frac{1}{2} g_o t_a^2 \tag{5.6}$$

The variable notation is the same as the single stage case with the addition of subscript i. This denotes each stage with n being the final stage of the rocket.

5.1.3 Results from Hand Calculations

For the hand calculations carried out, the velocity used to approximate the drag was the average velocity found from the iterative solution. Both sets of calculations were done with the same parameters and are listed in the following tables along with the corresponding results. To simplify the multi-stage calculations, each stage is identical (i.e. same set of parameters are used).

Diameter	Ref. Area	I_{sp}	m_o	m_f	v_{avg}	ρ	C_D
6.11 in	0.0189 m^2	197 s	34.6 kg	24 kg	400 m/s	0.7 kg/m^3	0.3

Table 1: Test Parameters Used for Calculations

Number of Stages	Time to Apogee (s)	Apogee (m)
1 Stage (Iterative)	37.6 s	7971 m
1 Stage (Hand Calc.)	37.21 s	8707 m
Error (%)	1.05%	9.33%
3 Stage (Iterative)	64.235 s	18800 m
3 Stage (Hand Calc.)	55.235 s	20730 m
Error (%)	14%	10.27%

Table 2: Tabulated Results Between Hand Calculations and Simulation

The difference between the hand calculations and iterative method can be chalked up to how the equations are calculated and the assumptions made. The hand calculations do not take any of the atmospheric conditions into account (except for density, which is used for drag losses). Furthermore, the hand calculations are solved with the assumption that various parameters are fixed whereas the iterative method does not. With the differences taken into account, it can be concluded that the iterative method does provide a solution within a reasonable ballpark. The full set of calculations carried out can be found in Appendix A.8.

5.2 Changes in Precision with Varying Step-Sizes

One parameter which tends to have a large effect on the precision of the result is the step-size (or time-step). To test this, a one-stage rocket case was used where the step-sizes were made smaller after each iteration by $\frac{h}{2}$, where h is the step-size starting at 1. After integrating with the Runge-Kutta 4th Order method, the following graphs were generated and illustrates some interesting behaviours:

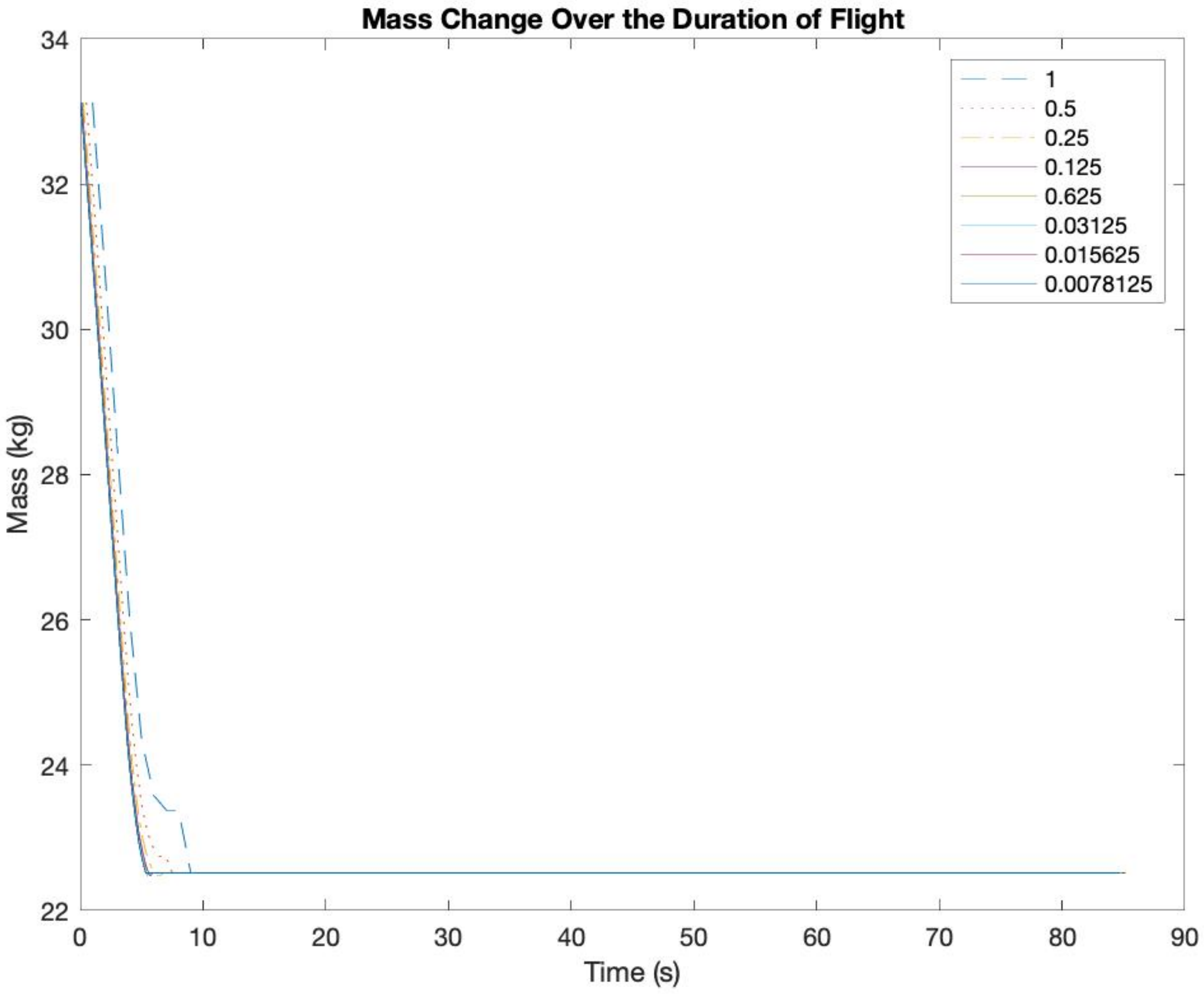


Figure 5.2: Comparison of Mass with Varying Step-Sizes

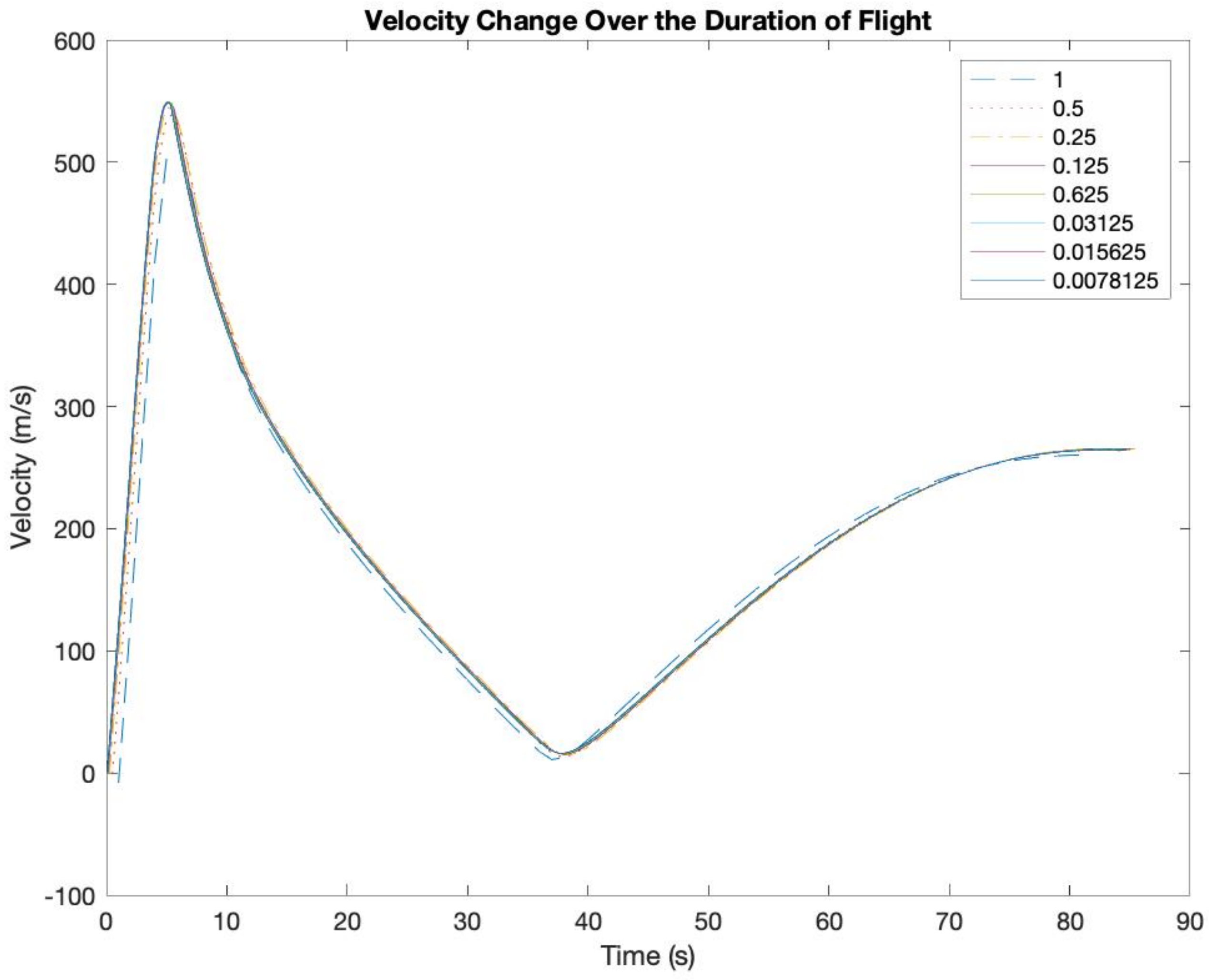


Figure 5.3: Comparison of Velocity with Varying Step-Sizes

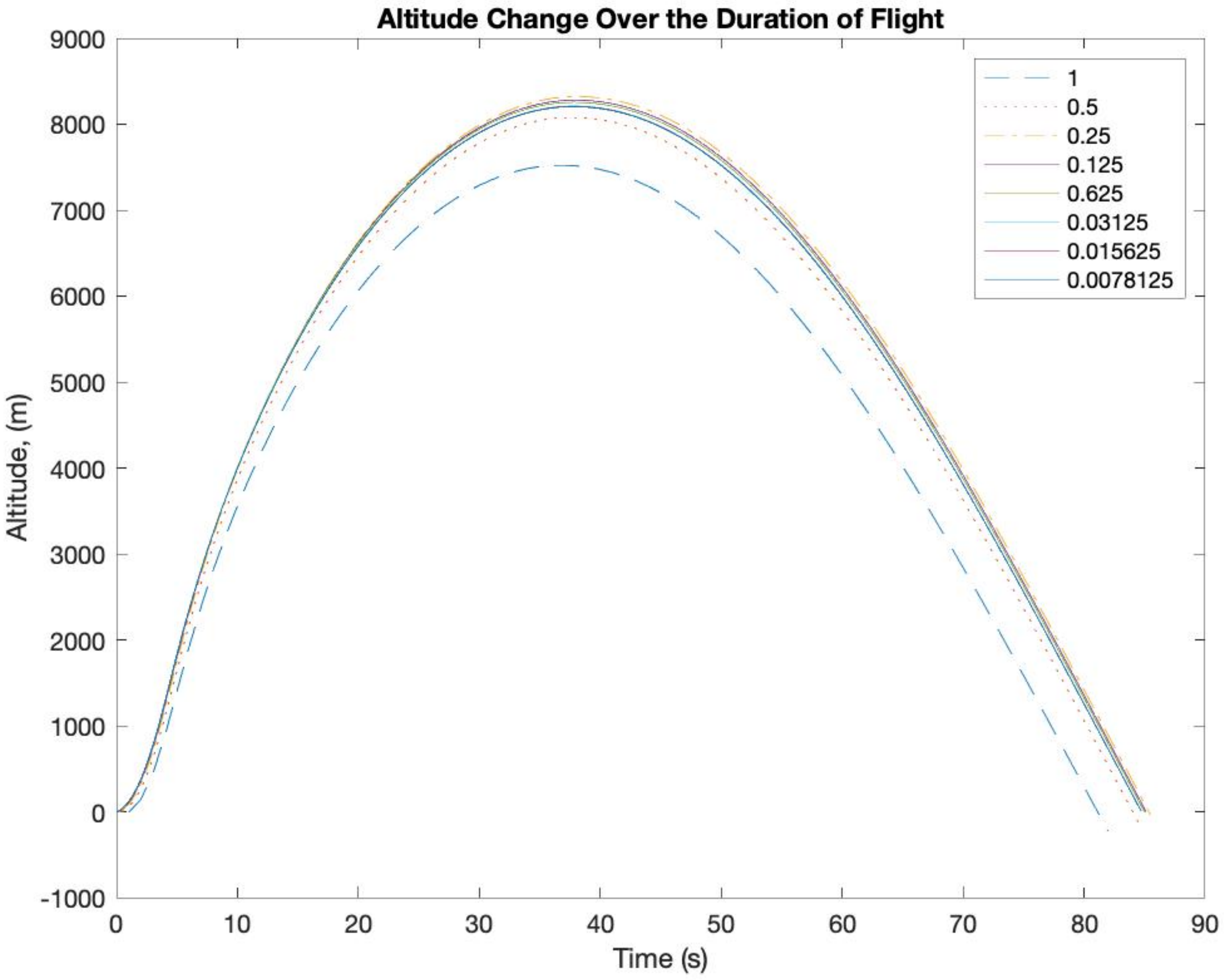


Figure 5.4: Comparison of Altitude with Varying Step-Sizes

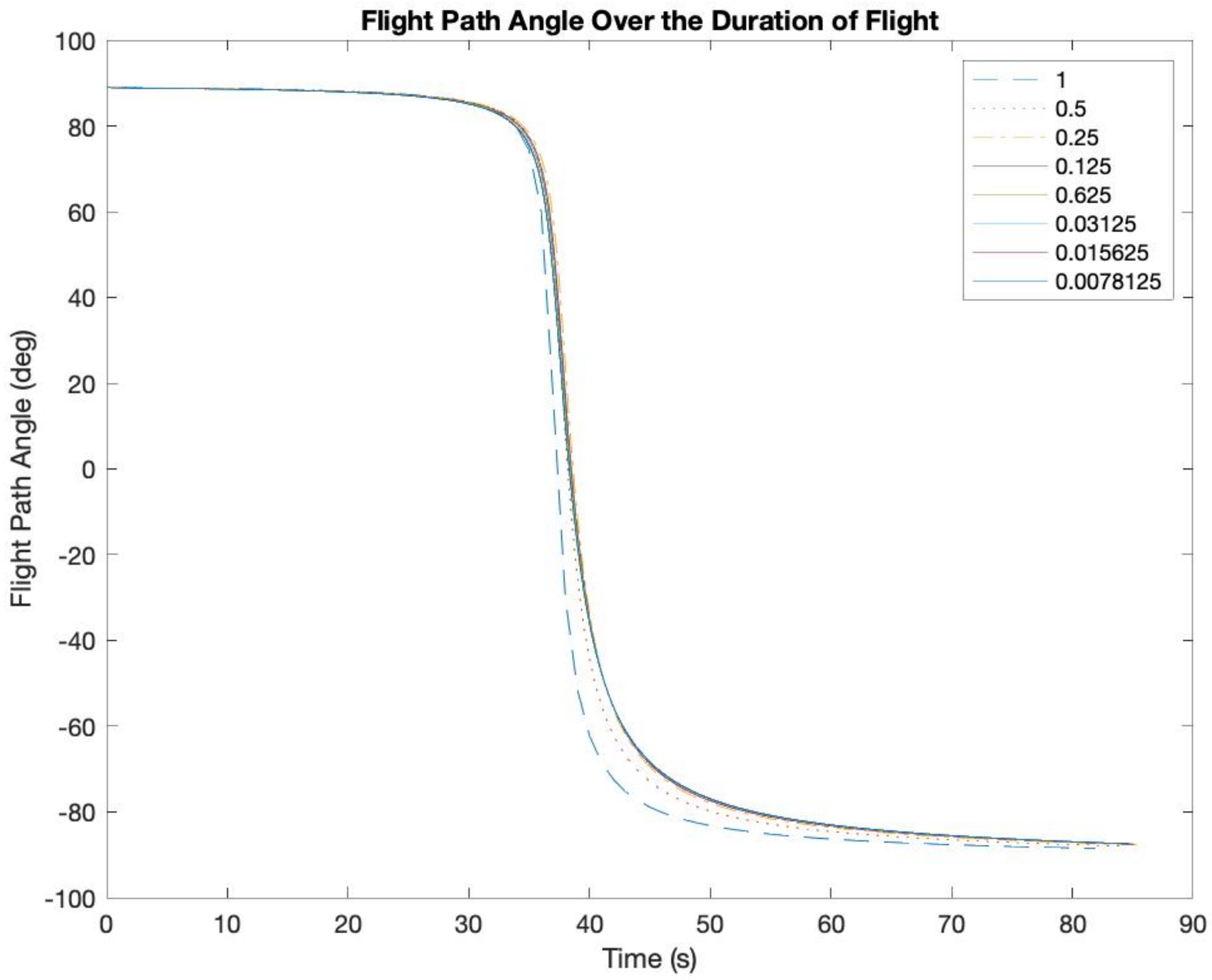


Figure 5.5: Comparison of Flight Path Angle with Varying Step-Sizes

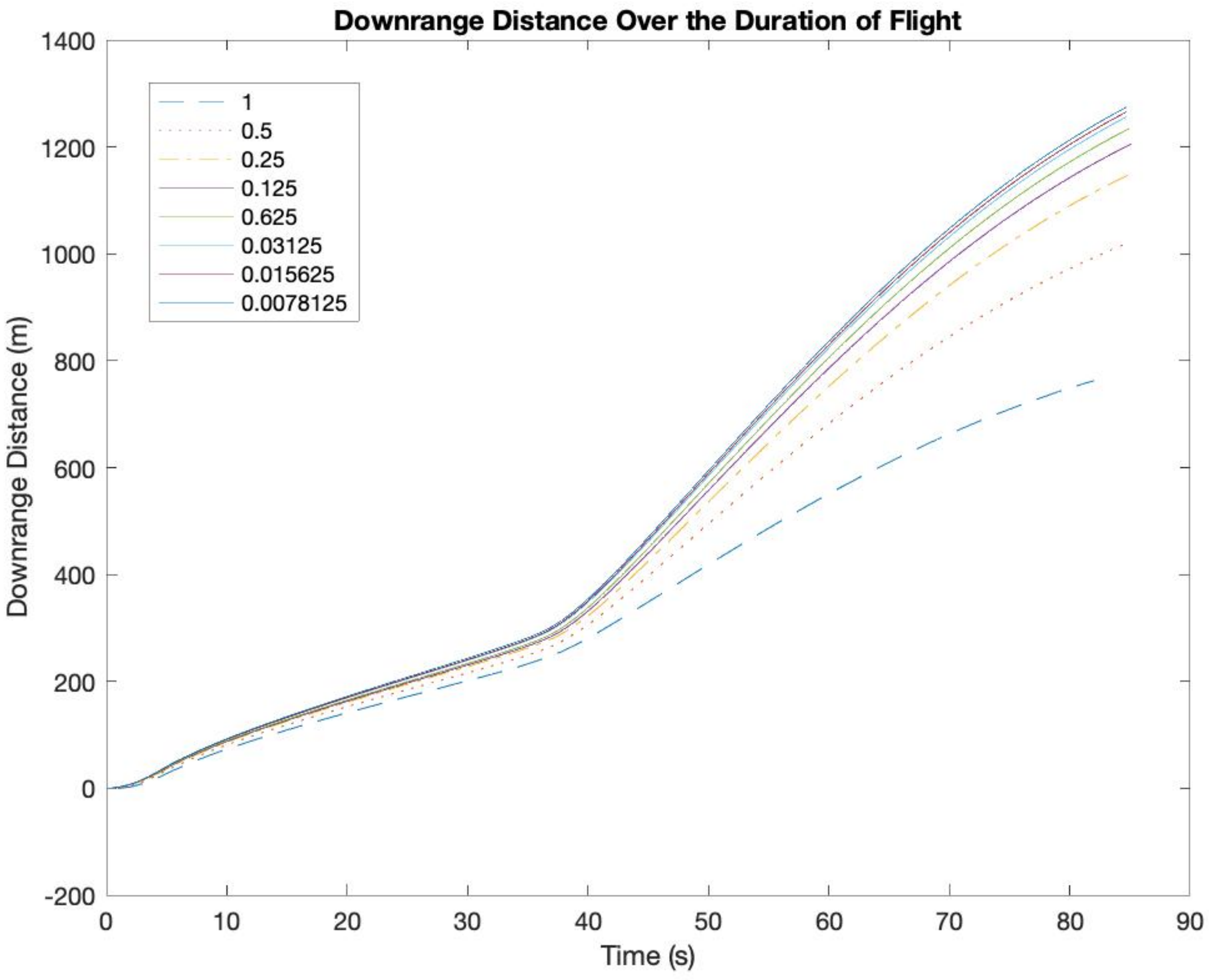


Figure 5.6: Comparison of Downrange Distance with Varying Step-Sizes

From the Fig. 5.2 to 5.6 it shows that with a smaller step size, the solution starts to converge towards an exact solution. This behaviour is as expected as using a smaller step size helps reduces the total truncation error. However, it is possible to increase the roundoff error in calculations due to subtractive cancellation if the step-size is made too small. The step-size where this occurs is dependent on a case by case basis, thus the user will have to manually test for this.

5.3 Comparison with Flight Data

The flight data used for comparison is provided by Ryerson Rocketry Club from a launch in 2018 with the data recorded on a commercially available flight computer. However, the data provided did not include the launch site conditions such as the launch site altitude ASL, air density, or the ambient temperature on the day of the launch. Thus, the launch conditions used in the simulation are approximated for 06/21/2018 around midday to coincide with the approximate launch time.

ASL	1400.556m
$T_{amb}(approx.)$	37^oC
$\rho_{amb}(approx.)$	1.1084 kg/m^3
Dry Mass	28.93 kg
Fuel Mass	10.6 kg
Launch Angle	87^o

Table 3: Launch Conditions at Spaceport America for 06/21/2018

Plotting the simulation results over the flight data, one can see the similarity between the two from launch until apogee in Fig. 5.7. After reaching apogee, the two values start to differ as the rocket from Ryerson Rocketry Club deploys a parachute while in the simulation it continues on a ballistic trajectory.

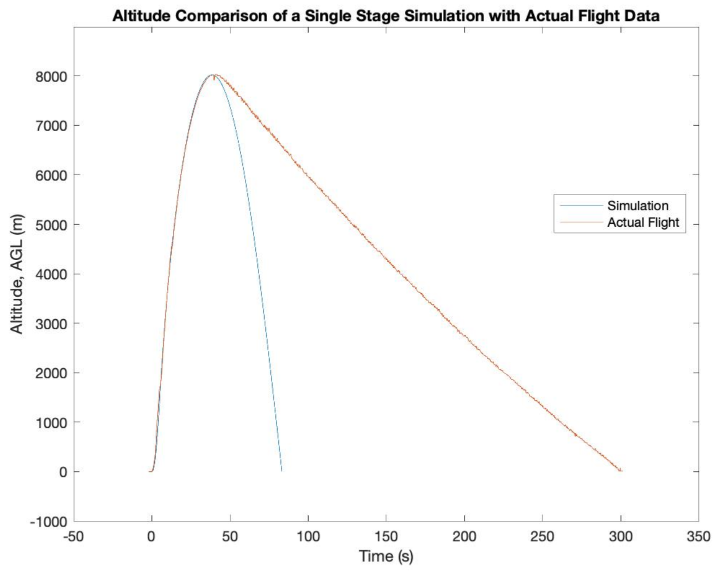


Figure 5.7: Comparison of the Predicted Altitude for a Single Stage Rocket

The simulation calculated an apogee of 8016.8 m while the actual apogee was 8019.2 m, yielding an error of 0.0274%. This error should be taken with caution as it could be greater or less due to the assumptions made for the atmospheric conditions and an approximate mass value.

5.3.1 Sources of Error

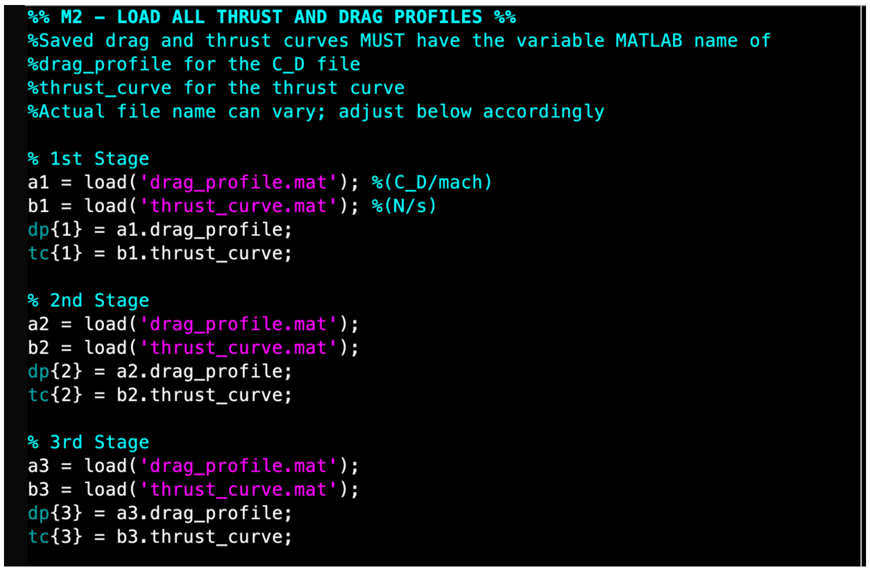
As stated before, numerical integrators all have an inherent amount of truncation error. This error cannot be removed completely, but it can be reduced by using higher-order integrators. Additional numerical errors can also result from the rounding of digits after each calculation. Furthermore, there is also the error which comes from the approximations made in Chapter 3.

6 General Use and Operations of the Code

The method is designed to ideally be used with user-specified presets. However, manually inputting parameters using the command line is also possible, albeit time-consuming if the user wishes to test numerous cases. Furthermore, this program assumes that the user has a basic understanding of the MATLAB environment.

6.1 Loading Custom Thrust and Drag Profiles

If the user has a thrust curve for the rocket's motor/engine or the drag profile of the launch vehicle, it can be loaded into the program by modifying the section named *M2 - Load All Thrust and Drag Profiles*. If no drag profile or thrust curve exists, comment out the following section:

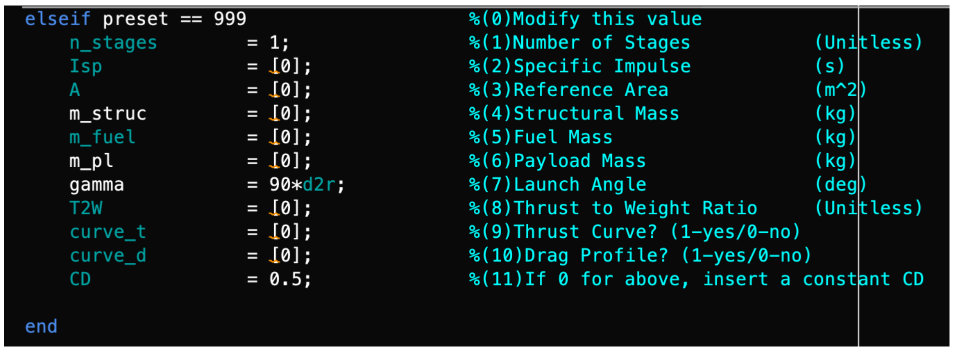


```
%% M2 - LOAD ALL THRUST AND DRAG PROFILES %%
%Saved drag and thrust curves MUST have the variable MATLAB name of
%drag_profile for the C_D file
%thrust_curve for the thrust curve
%Actual file name can vary; adjust below accordingly

% 1st Stage
a1 = load('drag_profile.mat'); %(C_D/mach)
b1 = load('thrust_curve.mat'); %(N/s)
dp{1} = a1.drag_profile;
tc{1} = b1.thrust_curve;

% 2nd Stage
a2 = load('drag_profile.mat');
b2 = load('thrust_curve.mat');
dp{2} = a2.drag_profile;
tc{2} = b2.thrust_curve;

% 3rd Stage
a3 = load('drag_profile.mat');
b3 = load('thrust_curve.mat');
dp{3} = a3.drag_profile;
tc{3} = b3.thrust_curve;
```

Figure 6.1: Loading Thrust and Drag Profiles

6.2 Creating a Preset

To create a preset, values within *M3 - Launch Vehicle Parameters* section in the *main.m* file will need to be modified. To do so, make a copy of the template and modify the 12 parameters accordingly. An example of the preset template is shown in Fig. 6.2.



```
elseif preset == 999                          %(0)Modify this value
    n_stages        = 1;                      %(1)Number of Stages          (Unitless)
    Isp             = [0];                    %(2)Specific Impulse          (s)
    A               = [0];                    %(3)Reference Area            (m^2)
    m_struc         = [0];                    %(4)Structural Mass           (kg)
    m_fuel          = [0];                    %(5)Fuel Mass                 (kg)
    m_pl            = [0];                    %(6)Payload Mass              (kg)
    gamma           = 90*d2r;                 %(7)Launch Angle              (deg)
    T2W             = [0];                    %(8)Thrust to Weight Ratio    (Unitless)
    curve_t         = [0];                    %(9)Thrust Curve? (1-yes/0-no)
    curve_d         = [0];                    %(10)Drag Profile? (1-yes/0-no)
    CD              = 0.5;                    %(11)If 0 for above, insert a constant CD

end
```

Figure 6.2: Preset Template

A brief description of each parameter is listed below:

(0) Preset index. Change this to a value that is not in use.

(1) Change this entry to the total number of stages of the rocket.

(2) I_{sp} (in s) of each stage in an array, starting with the first stage.

(3) Cross-sectional reference area for each stage of the rocket in an array, starting with the first stage.

(4) Total structural mass (including avionics, engine etc.) of each stage in an array, starting with the first stage.

(5) Total propellant mass of each stage in an array, starting with the first stage.

(6) Total mass of actual payload (not the combined sums of n-stages above) of each stage in an array, starting with the first stage.

(7) Launch angle of the rocket.

(8) Thrust-to-weight ratio of each stage in an array, starting with the first stage.

(9) Signifies whether a thrust curve exists for each stage in an array, starting with the first stage. (1 - yes, 0 - no)

(10) Signifies whether a drag profile exists exists for each stage in an array, starting with the first stage. (1 - yes, 0 - no)

(11) If no drag profile exists (i.e. 0 for entry (10)) enter a fixed C_D value.

6.3 Launch Site Conditions

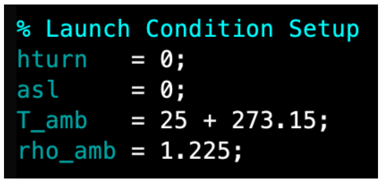
By default, the program is set to the SATP conditions. This can be found under the launch condition setup portion within the *main.m* file. The only values of interest is the altitude ASL in m, the ambient temperature in Celsius, and the ambient pressure kg/m^2. These values are tabulated below.

ASL	0
T_{amb}	25
ρ_{amb}	1.225

Table 4: SATP Settings

6.3.1 Modifying Launch Site Conditions

Modifying the launch site conditions can be done through the launch condition setup section of the code and changing the following values:



```
% Launch Condition Setup
hturn   = 0;
asl     = 0;
T_amb   = 25 + 273.15;
rho_amb = 1.225;
```

Figure 6.3: Launch Conditions within the Code

hturn is the altitude where the pitchover begins (i.e. the launch angle is no longer fixed to the initial value). This usually occurs once the rocket has cleared the launch rail/launch pad or early on during its flight. The remaining values should be changed to the launch site's altitude above sea level in m, ambient temperature in K, and the ambient air density in kg/m^2.

6.4 Limitations of the Method

Due to the constrained duration of this project, only a single set of data was readily available for comparison, and as such, the results from the use of the method should be taken with caution. The method can produce a reasonable estimate towards the performance of the rocket, however, further calibration will need to be done to achieve more accurate results.

7 Next Steps

The next steps for this design method is to obtain additional sets of flight data for both single and multi-stage rockets to compare with the results of the method. This in turn will allow for adjustments to be made to further improve the accuracy of the method and possibly validating the results obtained from this method. Additionally, future improvements can be made to allow for additional parameters to be handled such as delays before stage ignition, mid-flight course adjustments, etc.

8 Summary of Results

Over the course of the project, a method for calculating a rocket's mass, flight-path angle, velocity, altitude, and down-range distance was designed. The method utilises a numerical integrator to solve the system of ODE's for a solution. Hand calculations were then compared with the results from several test cases to determine whether the solution from the method was in an appropriate 'ballpark' range. The errors between the hand-calculations and the method for both the single and 3-stage cases were all under 15%. This difference can be attributed to the drag being a constant fixed value and the atmospheric conditions not taken into account in the hand calculations. A second test was done to test the precision of the tool by changing the step-size, where it was found that by reducing the step-size, the method does converge towards a single solution. When compared to actual flight data from a launch in 2018, there was an error of 0.0274% between the actual apogee and the apogee provided from the simulation. This error can be attributed to the exact launch conditions not being known and the assumptions made inside the method itself. The overall accuracy of the method should be taken with some criticism, as there was only one test case available for comparison.

9 References

[1] S. C. Chapra, *Applied Numerical Methods.* 1221 Avenue of the Americas, New York, NY 10020: McGraw-Hill, 3 ed., 2012.

[2] H. D. Curtis, *Orbital Mechanics for Engineering Students.* The Boulevard, Langford Lane, Kidlington, Oxford, OX5 1GB, UK 225 Wyman Street, Waltham, 02451, USA: Butterworth-Heinemann, 2019.

[3] J. D. A. Jr., *Introduction to Flight.* 1221 Avenue of the Americas, New York, NY 10020: McGraw-Hill, 2012.

A Source Code

A.1 main.m

```
%% M1 - INITIALIZATION %%
% ALL UNITS ARE IN SI %
clc;
clear;
clear global;

% Rocket Parameters %
global...
    Isp...      % Specific impulse                       (s)
    m_t...      % Total mass of stage                    (kg)
    m_fuel...   % Total fuel mass of stage               (kg)
    t_bo...     % Burnout time of stage                  (s)
    T2W...      % Thrust to Weight ratio of stage        (Unitless)
    y...        % column vector for paramters            (Refer ...
        below)
    A...        % Reference area for drag computations   (m^2)
    hturn...    % Pitchover altitude                     (m)
    CD          % Drag Coefficient                       (Unitless)

% Launch Site Conditions %
global...
    asl...      % Launch site alt (above sea level)      (m)
    T_amb...    % Ambient temperature at launch site     (K)
    rho_amb     % Ambient air density                    (kg/m^3)

% Counters %
global...
    stage...    % Indicates the current stage of the vehicle
    n_stages... % Total number of stages in launch vehicle
    burn...     % Indicates whether the motor/engine is ...
        currently in use
    bypass      % Indicator of whether to bypass thrust ...
        calculations after burnout

% Thrust and Drag curve_ts
global...
    tc...       % Thrust curve of each motor/engine
    dp...       % Drag profile of the launch vehicle at each stage
    curve_d...  % States wehether a drag profile exists
    curve_t     % States whether a thrust curve is available, if ...
        not,
                % an alternate method will be used to compute thrust

% Constants
global...
    Re...       % Radius of the Earth                    (m)
    r2d...      % Conversion; Radians to Degrees         (Unitless)
    d2r         % Conversion; Degrees to Radians         (Unitless)

    Re      = 6378100;
    r2d     = 180/pi;
    d2r     = pi/180;

%% M2 - LOAD ALL THRUST AND DRAG PROFILES %%
%Saved drag and thrust curves MUST have the variable MATLAB name of
%drag_profile for the C_D file
%thrust_curve for the thrust curve
%Actual file name can vary; adjust below accordingly

% 1st Stage
a1 = load('drag_profile.mat'); %(C_D/mach)
b1 = load('thrust_curve.mat'); %(N/s)
dp{1} = a1.drag_profile;
tc{1} = b1.thrust_curve;

% 2nd Stage
a2 = load('drag_profile.mat');
b2 = load('thrust_curve.mat');
dp{2} = a2.drag_profile;
tc{2} = b2.thrust_curve;

% 3rd Stage
a3 = load('drag_profile.mat');
b3 = load('thrust_curve.mat');
dp{3} = a3.drag_profile;
tc{3} = b3.thrust_curve;

%% M3 - LAUNCH VEHICLE PARAMETERS %%
preset = input('Enter a saved profile \n');
if preset == 0
    %% NO PRESET
    % Specify number of booster stages
    n_stages = input('Number of Stages \n');

    for i = 1:n_stages
        Isp(i)              = input(['Specific Impulse of Stage ...
            ' num2str(i) ' in seconds \n']);
        A(i)                = input(['Cross-Sectional Reference ...
            Area of Stage ' num2str(i) ' in m^2\n']);
        m_struc(i)          = input(['Structural Mass of Stage ' ...
            num2str(i) ' in kg\n']);
        m_fuel(i)           = input(['Fuel Mass of Stage ' ...
            num2str(i) ' in kg\n']);
        no_thrust_curve     = input('Is a thrust curve ...
            available? (1 for yes, 0 for no)');
        if no_thrust_curve == 0
            T2W(i) = input(['Thrust to Weight Ratio of Stage ' ...
                num2str(i)]);
        end
        if i == n_stages
            m_pl(i) = input(['Payload Mass of Stage ' num2str(i) ...
                ' in kg\n']);
        end
    end

    % Specify an initial launch angle
    gamma = input('Initial Launch Angle in degrees \n');
    gamma = gamma*d2r;
```

```

elseif preset == 1
    %% PRESET 1; RRC Single Stage Sounding Rocket (Spaceport ...
        America Cup 2018)
    n_stages        = 1;                 %Number of Stages ...
                  (Unitless)
    Isp             = 197;               %Specific Impulse ...
                  (s)
    diam            = 0.155194;          %Cross-Sectional ...
        Diameter    (m)
    A               = pi*(diam/2)^2;     %Reference Area ...
                    (m^2)
    m_struc(1)      = 55*0.453592;       %Structural Mass ...
                   (kg)
    m_fuel(1)       = 23.369*0.453592;   %Fuel Mass ...
                         (kg)
    m_pl(1)         = 8.8*0.453592;      %Payload Mass ...
                      (kg)
    curve_t(1)      = 1;
    curve_d(1)      = 1;
    gamma           = 87*d2r;
    T2W(1)          = 17;
    CD              = 0.5;

elseif preset == 2
    %% PRESET 2; 2-Stage Sounding Rocket
    n_stages        = 2;                 %Number of Stages ...
                  (Unitless)
    Isp             = [197, 197];        %Specific Impulse ...
                  (s)
    diam            = 0.127;             %Cross-Sectional ...
        Diameter    (m)
    area            = (pi*(diam/2)^2);   %Reference Area ...
                    (m^2)
    A               = [area,area];       %Reference Area ...
                    (m^2)
    m_struc         = [12.68,16.68];     %Structural Mass ...
                   (kg)
    m_fuel          = [11,11];           %Fuel Mass ...
                         (kg)
    m_pl            = [0,2];             %Payload Mass ...
                      (kg)
    gamma           = 90*d2r;
    % T2W           = [17,17];
    curve_t         = [1,1];
    curve_d         = [1,1];
    CD              = 0.5;

elseif preset == 3
    %% PRESET 3; 3-Stage Rocket (Hand Calculation Comparison)
    n_stages        = 3;                     %Number of Stages ...
                  (N/A)
    Isp             = [197, 197, 197];       %Specific Impulse ...
                  (s)
    diam            = 0.155194;              %Cross-Sectional ...
        Diameter    (m)
    area             = (pi*(diam/2)^2);      %Reference Area ...
                        (m^2)
    A                = [area, area, area];   %Reference Area ...
                        (m^2)
    m_struc          = [24,24,24];           %Structural Mass ...
                       (kg)
    m_fuel           = [10.6,10.6,10.6];     %Fuel Mass ...
                             (kg)
    m_pl             = [0,0,0];              %Payload Mass ...
                          (kg)
    gamma            = 89*d2r;
    % T2W            = [17,17,17];
    curve_t          = [1,1,1];
    curve_d          = [1,1,1];


elseif preset == 4
        %% PRESET 4; 1-Stage Rocket (Hand Calculation Comparison)
    n_stages        = 1;                 %Number of Stages ...
                  (Unitless)
    Isp              = 197;              %Specific Impulse ...
                  (s)
    diam             = 0.155194;         %Cross-Sectional ...
        Diameter   (m)
    A                = pi*(diam/2)^2;    %Reference Area ...
                        (m^2)
    m_struc(1)       = 24;               %Structural Mass ...
                       (kg)
    m_fuel(1)        = 10.6;             %Fuel Mass ...
                             (kg)
    m_pl(1)          = 0;                %Payload Mass ...
                          (kg)
    curve_t(1)       = 1;
    curve_d(1)       = 1;
    gamma            = 89*d2r;
    T2W(1)           = 17;
    CD               = 0.5;


elseif preset == 999                     %(0)Modify this value
    n_stages         = 1;                %(1)Number of Stages ...
                  (Unitless)
    Isp              = [0];              %(2)Specific Impulse ...
                  (s)
    A                = [0];              %(3)Reference Area ...
                        (m^2)
    m_struc          = [0];              %(4)Structural Mass ...
                       (kg)
    m_fuel           = [0];              %(5)Fuel Mass ...
                             (kg)
    m_pl             = [0];              %(6)Payload Mass ...
                          (kg)
    gamma            = 90*d2r;           %(7)Launch Angle ...
                          (deg)
    T2W              = [0];              %(8)Thrust to Weight ...
        Ratio     (Unitless)
    curve_t         = [0];              %(9)Thrust Curve? ...
        (1-yes/0-no)
    curve_d         = [0];              %(10)Drag Profile? ...
        (1-yes/0-no)
    CD              = 0.5;              %(11)If 0 for above, ...
        insert a constant CD

end

%% MASS RATIOS %%
% Iterative loop to determine the total masses of each stage, ...
    where the
% payload mass of n-stage is the mass of all stages above
j = 0;
k = 1;

for i = n_stages:-1:1
    if i > 1
        m_t(n_stages-j)  = m_struc(n_stages-j) + ...
            m_fuel(n_stages-j) + m_pl(n_stages-j);
        m_pl(n_stages-k) = m_t(n_stages-j);

        j = j + 1;
        k = k + 1;

    elseif i == 1
        m_t(1) = m_struc(1) + m_fuel(1) + m_pl(1);
    end
end

%% M4 - SETTING LAUNCH CONDITIONS %%
% Launch Condition Setup
hturn   = 0;
asl     = 0;                %1400.556;  (Swap for Spaceport cond.)
T_amb   = 25 + 273.15;      %37;        (Swap for Spaceport cond.)
rho_amb = 1.225;            %1.1084;    (Swap for Spaceport cond.)

% Initialize Counters
stage   = 1;
t_bo    = 0;
burn    = 1;
bypass  = 0;

% Initialize IVC
v       = 0;                % Initial velocity              (m/s)
h       = 0;                % Initial Altitude              (m)
x       = 0;                % Initial downrange distance    (m)
y       = [v,h,gamma,x];    % Store IVC for import          ...
    (Numerous) [m_t(1),v,h,gamma,x];
ti      = 0;                % Starting time of integration  (s)
tf      = 8000;             % Maximum time for integration  (s)
step    = 0.001;              % Step-size of the integration  (s)
method  = 4;                % Chosen Integrator

% METHODS %
% 1 - Forward Euler Method
% 2 - Explicit Midpoint Method
```

```matlab
% 3 - 3rd Order Method
% 4 - 4th Order Method
% 5 - Ralston's 4th-order method

[t1,y1,s1] = num_int(@rates, ti, tf, y, step, method);

%% M5 - VISUALIZATION %%
%Plotting of the results from the integration
figure
plot(s1(:,1),s1(:,2))
xlabel('Time (s)')
ylabel('Mass (kg)')
title('Mass Change Over the Duration of Flight')

figure(2)
plot(s1(:,1),s1(:,3))
xlabel('Time (s)')
ylabel('Velocity (m/s)')
title('Velocity Change Over the Duration of Flight')

figure(3)
plot(s1(:,1),s1(:,4))
xlabel('Time (s)')
ylabel('Altitude, (m)')
title('Altitude Change Over the Duration of Flight')

figure
plot(s1(:,1),s1(:,5),'--')
xlabel('Time (s)')
ylabel('Flight Path Angle (deg)')
title('Flight Path Angle Over the Duration of Flight')

figure
plot(s1(:,1),s1(:,6),'--')
xlabel('Time (s)')
ylabel('Downrange Distance (m)')
title('Downrange Distance Over the Duration of Flight')
```

A.2 num_int.m

```
function [t,y,s] = num_int(f, ti, tf, y, h,method)
global...
    burn...
    m_t...
    stage...
    n_stages...
    r2d...
    bypass...
    tc...
    m...
    m_fuel...
    curve_t...
    t_bo...
    Isp...
    T...
    T2W

t = ti;
i = 0;
g0 = gravity(0);
m = m_t(stage);

while 1
    if t < tf
        %% MASS FLOW (m_dot) %%
        % Check to see if remaining mass is equal to empty mass
        % m_empty = m_t(stage) - m_fuel(stage);
        % g0 = gravity(0);
        % (1) First  check is to see if the total mass is less ...
            than or equal
        % to the dry-mass of the launch vehicle (2) Second check ...
            is to see
        % if the booster burn status has been completed (0 for ...
            complete, 1
        % for in process, 2 for transition to next stage) The ...
            burn variable
        % also acts as a marker for transitioning between stages

        if bypass == 1 && stage < n_stages
            burn = 1;
            bypass = 0;
            stage   = stage + 1;
            m = m_t(stage);
        elseif bypass == 1 && stage == n_stages
            burn = 0;
            m = m_t(stage) - m_fuel(stage);
        end

        m_empty = m_t(stage) - m_fuel(stage);

        if bypass == 0
            if m ≤ m_empty
                T       = 0;
                burn    = 0;

            else
                %Checks to see if a thrust curve is available; ...
                    if not, the thrust to
                %weight ratio will be used to determine the thrust
                if curve_t(stage) == 0
                    T = T2W(stage)*m_t(stage)*g0;
                else
                    [burn,T] = thrust(t-t_bo,tc{stage});
                end
            end

            if burn == 0
                t_bo    = t;
                bypass  = 1;
            end
        else
            T = 0;
        end
        %Calculating the mass flow rate of the fuel mass
        m_dot = -T/Isp(stage)/g0;

        %Resolving new mass
        m = m + m_dot*h;

        if method == 1
            %% Forward Euler Method
            k_1 = f(t,y,h);
            y   = y + h*k_1;

        elseif method == 2
            %% Explicit Midpoint Method
            k_1 = f(t, y, h);
            k_2 = f(t + h/2, y + k_1/2, h/2);
            y   = y+ h*k_2;

        elseif method == 3
            %% 3-Stage Method
            k_1 = f(t, y, h);
            k_2 = f(t + h/2, y + k_1/2, h/2);
            k_3 = f(t + h, y - k_1 + 2*k_2, h);
            y   = y + h*(k_1/6 + (2/3)*k_2 + k_3/6);

        elseif method == 4
            %% 4-Stage Method
            k_1 = f(t, y, h);
            k_2 = f(t + h/2, y + k_1/2, h/2);
            k_3 = f(t + h/2, y + k_2/2, h/2);
            k_4 = f(t + h, y + k_3, h);
            y   = y + h*((1/6)*(k_1 + 2*k_2 + 2*k_3 + k_4));

        elseif method == 5
            %% Ralston's 4th Order method
            k_1 = f(t, y, h);
            k_2 = f(t + 0.4*h, y + 0.4*k_1, 0.4*h);
            k_3 = f(t + 0.45573725*h, y + .29697761*k_1 + ...
                .15875964*k_2, 0.45573725*h);
            k_4 = f(t + h, y + .21810040*k_1 -3.05096516*k_2 + ...
                3.83286476*k_3, h);
            y   = y + h*(.17476028*k_1 -.55148066*k_2 + ...
                +1.20553560*k_3 + .17118478*k_4);

        elseif method == 6
            %% Butcher's 5th Order Runge-Kutta Method
            k_1 = f(t, y, h);
            k_2 = f(t + h/4, y + (k_1/4)*h, h/4);
            k_3 = f(t + h/4, y + (k_1/8)*h + (k_2/8)*h, h/4);
            k_4 = f(t + h/2, y - (k_2/2)*h + k_3*h, h/2);
            k_5 = f(t + h*(3/4), y + (3/16)*k_1*h + ...
                (9/16)*k_4*h, h*(3/4));
            k_6 = f(t + h, y - (3/7)*k_1*h + (2/7)*k_3*h - ...
                (12/7)*k_4*h + (8/7)*k_5*h, h);
            y   = y + (1/90)*(7*k_1 + 32*k_3 + 12*k_4 + 32*k_5 + ...
                7*k_6)*h;

        end

        %% DISPLAY DATA %%

        %         fprintf('\n\n -------------Step ...
            %d---------------\n',i)
        %         fprintf('\n\n ...
            -----------------------------------\n')
        fprintf('\n Time                      = %10g s   ',t)
        %         fprintf('\n Mass                      = %10g ...
            kg  ',y(1))
        %         fprintf('\n Velocity                  = %10g ...
            m/s   ',y(2))
        %         fprintf('\n Altitude                  = %10g m ...
            ',y(3))
        %         fprintf('\n Flight Path Angle         = %10g ...
            deg',y(4)*r2d)
        %         fprintf('\n Downrange Distance        = %10g m ...
             ',y(5))
        %         fprintf('\n\n ...
            -----------------------------------\n')

        t = t + h;
        i = i+1;

        % Stores data in a matrix for graphing purposes
        s(i,1) = t;
        s(i,2) = m;
        s(i,3) = y(1);
        s(i,4) = y(2);
        s(i,5) = y(3)*r2d;
        s(i,6) = y(4);

        if y(2) < 0 && t > 1
            return
        end
    else
        return
    end
```

```
end
end
```

A.3 rates.m

```
function dydt = rates(t,y,step)
global...
    stage...
    dp...
    A...
    Re...
    m...
    T...
    hturn...
    curve_d...
    CD
v            = y(1);
h            = y(2);
gamma        = y(3);
x            = y(4);

%% VELOCITY DETERMINATION %%
[temp,r]     = rho(h);
g            = gravity(h);

if curve_d(stage) == 1
D = drag(r,v,A(stage),temp,dp{stage});
else
    D = 0.5*r*v^2*A(stage)*CD;
end

v_dot = (T-D)/m - g*sin(gamma);

if t < 0.5 && v_dot < 0
    v_dot = 0;
end

v = v + v_dot*step;

%% FLIGHT PATH ANGLE %%

%Holds gamma_dot = 0 b/c if v_dot = 0, then there can be no
%gamma_dot
if h < hturn
    gamma_dot = 0;
elseif v == 0 || v < 1 && t < 1
    gamma_dot = 0;
else
    gamma_dot = (-1/v)*(g - ((v^2)/(Re+h)))*cos(gamma);
end
% solve for the new flight path angle
gamma    = gamma + gamma_dot*step;

%% ALTITUDE %%
h_dot    = v*sin(gamma);
h        = h + h_dot*step;

%% DOWNRANGE DISTANCE %%
%solve for the rate of change downrange
```

```
x_dot    = (Re/(Re + h))*v*cos(gamma);

%solve for the new downrange distance
x        = x + x_dot*t;

dydt(1) = v_dot;
dydt(2) = h_dot;
dydt(3) = gamma_dot;
dydt(4) = x_dot;
end
```

A.4 thrust.m

```
%% Thrust Determination %%
%The function drag returns the drag force at a specific velocity ...
    by using a
%linear approximation method for finding the C_D values in ...
    intermittent
%ranges

%Requires inputs of density (kg/m^3), velocity (m/s^2), ...
    cross-sectional
%area (m^2) and the drag profile (matrix with 2 columns, 1st being
%velocities, 2nd being the correlating C_D values)
function [burn,T] = thrust(t,thrust_curve)

    %Check if the time is exceeding max burn time
    if t < thrust_curve(length(thrust_curve(:,1)),1)
        for i = 1:length(thrust_curve)
            if t ≤ thrust_curve(i+1,1) && t ≥ thrust_curve(i,1)
                thrust_2    = thrust_curve(i+1,2);
                thrust_1    = thrust_curve(i,2);
                t_2         = thrust_curve(i+1,1);
                t_1         = thrust_curve(i,1);

                %Linear approximation
                T       = ((thrust_2 - thrust_1)/(t_2 - t_1)) * ...
                    (t-t_1) + thrust_1;   %(N)
                burn    = 1;
                break
            end
        end
    else
        T       = 0;
        burn    = 0;
    end
end
```

A.5 drag.m

```
%% Drag Determination %%
%The function drag returns the drag force at a specific velocity ...
    by using a
%linear approximation method for finding the C_D values in ...
    intermittent
%ranges

%Requires inputs of density (kg/m^3), velocity (m/s^2), ...
    cross-sectional
%area (m^2) and the drag profile (matrix with 2 columns, 1st being
%velocities, 2nd being the correlating C_D values)

function D = drag(rho,v,A,T,drag_profile)
    gamma   = 1.4;
    R       = 287.058;
    v_sound = sqrt(gamma*R*T);
    mach    = abs(v/v_sound);

l = length(drag_profile); %length of the drag profile matrix
for i = 1:(l-1)
    if mach ≥ drag_profile(i,1) && mach < drag_profile(i+1,1)
        C_D_2   = drag_profile(i+1,2);
        C_D_1   = drag_profile(i,2);
        mach_2  = drag_profile(i,1);
        mach_1  = drag_profile(i+1,1);

        %Linear approximation
        C_D = ((C_D_2 - C_D_1)/(mach_2 - mach_1)) * ...
            (mach-mach_1) + C_D_1;
        break

    elseif mach ≥ drag_profile(l,1)
        C_D     = drag_profile(l,2);
        break

    elseif mach ≤ 0
            disp('END');
        return
    end
end
    D = 0.5*rho*(v^2)*A*C_D;
end
```

A.6 rho.m

```
%% Density Determination %%
%The function rho returns the density in kg/m^3

%Requires an input of an altitude above sea level
%Calculations are done based on the standard atmosphere

function [T,r]  = rho(h_g)
global  T_amb asl Re rho_amb
%d2k = 273.15;
g0 = gravity(0);
h_fix   = h_g + asl;
R = 287;
h = (Re/(Re + h_fix))*h_fix;

% Region 1
if h < 11000
    a1 = (216.66-T_amb)/11000;
    T1 = T_amb + a1*h;
    r1 = rho_amb * (T1/T_amb)^(-((g0/(a1*R)) + 1));

    T = T1;
    r = r1;

elseif h ≥ 11000 && h < 25000
    T2 = 216.66;

    a1 = (216.66-T_amb)/11000;
    r1 = rho_amb * (216.66/T_amb)^(-((g0/(a1*R)) + 1));

    r2 = r1 * exp(1)^(-(g0/(R*T2))*(h - 11000));

    T = T2;
    r = r2;
elseif h ≥ 25000 && h < 47000
    a3 = 3e-3;
    T3 = 216.66 + a3 * (h-25000);

    a1 = (216.66-T_amb)/11000;
    r1 = rho_amb * (216.66/T_amb)^(-((g0/(a1*R)) + 1));

    r2 = r1 * exp(1)^(-(g0/(R*216.66))*(h - 11000));

    r3 = r2 * (T3/216.66)^(-((g0/(a3*R)) + 1));

    T = T3;
    r = r3;

elseif h ≥ 47000 && h < 53000
    T4 = 282.66;

    a1 = (216.66-T_amb)/11000;
    r1 = rho_amb * (216.66/T_amb)^(-((g0/(a1*R)) + 1));

    r2 = r1 * exp(1)^(-(g0/(R*216.66))*(h - 11000));

    a3 = 3e-3;
    r3 = r2 * (282.66/216.66)^(-((g0/(a3*R)) + 1));

    r4 = r3 * exp(1)^(-(g0/(R*T4))*(h - 47000));

    T = T4;
    r = r4;

elseif h ≥ 53000 && h < 79000
    a5 = -4.5e-3;
    T5 = 282.66 + a5 * (h-53000);

    a1 = (216.66-T_amb)/11000;
    r1 = rho_amb * (216.66/T_amb)^(-((g0/(a1*R)) + 1));

    r2 = r1 * exp(1)^(-(g0/(R*216.66))*(h - 11000));

    a3 = 3e-3;
    r3 = r2 * (282.66/216.66)^(-((g0/(a3*R)) + 1));

    r4 = r3 * exp(1)^(-(g0/(R*282.66))*(h - 47000));

    r5 = r4 * (T5/282.66)^(-((g0/(a5*R)) + 1));

    T = T5;
    r = r5;

elseif h ≥ 79000 && h < 90000
    T6 = 165.66;

    a1 = (216.66-T_amb)/11000;
    r1 = rho_amb * (216.66/T_amb)^(-((g0/(a1*R)) + 1));

    r2 = r1 * exp(1)^(-(g0/(R*216.66))*(h - 11000));

    a3 = 3e-3;
    r3 = r2 * (282.66/216.66)^(-((g0/(a3*R)) + 1));

    r4 = r3 * exp(1)^(-(g0/(R*282.66))*(h - 47000));

    a5 = -4.5e-3;
    r5 = r4 * (165.66/282.66)^(-((g0/(a5*R)) + 1));

    r6 = r5 * exp(1)^(-(g0/(R*T6))*(h - 47000));

    T = T6;
    r = r6;

elseif h ≥ 90000 && h < 105000
    a7 = 4e-3;
    T7 = 165.66 + a7 * (h-90000);

    a1 = (216.66-T_amb)/11000;
    r1 = rho_amb * (216.66/T_amb)^(-((g0/(a1*R)) + 1));

    r2 = r1 * exp(1)^(-(g0/(R*216.66))*(h - 11000));

    a3 = 3e-3;
    r3 = r2 * (282.66/216.66)^(-((g0/(a3*R)) + 1));

    r4 = r3 * exp(1)^(-(g0/(R*282.66))*(h - 47000));

    a5 = -4.5e-3;
    r5 = r4 * (165.66/282.66)^(-((g0/(a5*R)) + 1));

    r6 = r5 * exp(1)^(-(g0/(R*165.66))*(h - 47000));

    r7 = r6 * (T7/165.66)^(-((g0/(a7*R)) + 1));

    T = T7;
    r = r7;

elseif h > 105000
    T = 0;
    r = 0;
end
end
```

A.7 gravity.m

```
%% Gravity Determination %%
% Solves for the gravity depending on the altitude
function g = gravity(h)
global asl
h   = h + asl;
g_0 = 9.81;                 %gravitational constant          (m/s^2)
R_e = 6378e3;               %radius of Earth                 (m)
g   = g_0*(R_e/(R_e+h))^2;  %adjusted gravitational const.   (m/s^2)
end
```

A.8 handcalcs.m

```
clc; clear;
g_0      = 9.81;                              %Gravitational acc. ... (m/s^2)
T        = 4690;                              %Thrust ... (N)
Isp      = 197;                               %Specific impulse ... (s)
lb2kg    = 0.453592;                          %lbs to kg conversion
m_0      = 248+10.6;                          %Initial mass ... (kg)
m_f      = 24;                                %Final mass ... (kg)
m_dot    = T/(Isp*g_0);                       %Mass flow rate ... (
diam     = 0.155194;                          %Cross-Sectional ... Diameter (m)
A        = pi*(diam/2)^2;                     %Reference Area ... (m^2)
%% Single Stage Case
D        = 0.5 * 0.7 * 400^2 * 0.3 * A;       %Drag ... (N)
c        = Isp*g_0;
t        = (c*log(m_0/m_f))/(g_0 + (D/m_0))
%t1 = (c/(g_0))*log(m_0/m_f)

h        = c/m_dot * (m_0*log(m_f/m_0) +  m_0 - m_f) + ... c*t*log(m_0/m_f) - 0.5*g_0*t^2 - 0.5*(D/m_f)*t^2

%% 3-Stage Case
% Calculating burn time of each individual stage
m_struc          = 24;                        %Structural Mass ... (kg)
m_fuel           = 10.6;                      %Fuel Mass ... (kg)

m_01 = 3*m_struc + 3*m_fuel;
m_f1 = 3*m_struc + 2*m_fuel;
m_02 = 2*m_struc + 2*m_fuel;
m_f2 = 2*m_struc + 1*m_fuel;
m_03 = m_struc + m_fuel;
m_f3 = m_struc;

m_av1 = (m_01+m_f1)/2;
m_av2 = (m_02+m_f2)/2;
m_av3 = (m_03+m_f3)/2;

D_1 = 0.5 * 1.225 * 100^2 * 0.3 * A;
D_2 = 0.5 * 1.05 * 250^2 * 0.3 * A;
D_3 = 0.5 * 0.3 * 400^2 * 0.3 * A;

v_bo1 = c*log(m_01/m_f1);
v_bo2 = c*log(m_02/m_f2);
v_bo3 = c*log(m_03/m_f3);

```

```
t_1 = v_bo1/(g_0 + (D_1/m_01));
t_2 = v_bo2/(g_0 + (D_2/m_02));
t_3 = v_bo3/(g_0 + (D_3/m_03));

v_bo1 = v_bo1 - (D_1/m_f1)*t_1;
v_bo2 = v_bo2 - (D_2/m_f2)*t_2;
v_bo3 = v_bo3 - (D_3/m_f3)*t_3;

t_a = (v_bo1 + v_bo2 + v_bo3)/((D_3/m_f3) + g_0)
t_c = t_a - t_2 - t_1;

a1 = c/m_dot * (m_01*log(m_f1/m_01) +  m_01 - m_f1) + ...
    c*t_1*log(m_01/m_f1);
a2 = c/m_dot * (m_02*log(m_f2/m_02) +  m_02 - m_f2) + ...
    c*t_2*log(m_02/m_f2);
a3 = c/m_dot * (m_03*log(m_f3/m_03) +  m_03 - m_f3) + ...
    c*t_3*log(m_03/m_f3);

d_1 = 0.5*(D_1/m_f3)*t_1^2;
d_2 = 0.5*(D_2/m_f3)*t_2^2;
d_3 = 0.5*(D_3/m_f3)*t_3^2;

h = a1 + a2 + a3 - d_1 -d_2 - d_3 - 0.5*g_0*t_a^2
```

B Test Data

The thrust curve and drag profile used for the single-stage case comparison is tabulated within Appendix B.

B.1 Thrust Curve

Table 5: Cesaroni O3400 Motor

Time (s)	**Thrust** (N)
0	0
0.04	3959.811
0.052	4432.624
0.101	4515.366
0.19	4420.804
0.38	4391.253
0.965	4444.444
2.176	4698.582
2.887	4592.199
3.658	4225.768
4.17	2854.61
4.493	2559.102
4.881	1619.385
5.483	868.794
6.137	248.227
6.322	0

B.2 Drag Profile

Table 6: Single Stage Rocket Drag Profile

Mach Number	C_D
0	0
0.05	0.39533
0.1	0.35821
0.15	0.34001
0.2	0.32821
0.25	0.31892
0.3	0.31193
0.35	0.30617
0.4	0.30127
0.45	0.29677
0.5	0.29277
0.55	0.28926
0.6	0.28622
0.65	0.28371
0.7	0.28184
0.75	0.28078
0.8	0.28104
0.85	0.2838
0.9	0.2929
0.95	0.32417
1	0.40027
1.05	0.50407
1.1	0.4282
1.15	0.37164
1.2	0.45966
1.25	0.47864
1.3	0.45567
1.35	0.43666
1.4	0.42557
1.45	0.4177
1.5	0.41073
1.55	0.40393
1.6	0.39714
1.65	0.39033
1.7	0.38355
1.75	0.3769
1.8	0.37032
1.85	0.3638
1.9	0.35738