AOE 4404 NUMERICAL METHODS SEMESTER PROJECT

ONE DIMENSIONAL HYBRID ROCKET FLIGHT SIMULATION

Submitted by:

MATTHEW MARTI

AEROSPACE AND OCEAN ENGINEERING DEPARTMENT

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

BLACKSBURG, VIRGINIA

07 MAY, 2019

Abstract

Hybrid rocket motors consist of a propellant system in which one component is stored in a liquid state and the other component is stored as a solid. Typically the oxidizer is stored in liquid form, and is injected across a solid fuel grain in the combustion chamber. Hybrid rocket motors usually have higher Specific Impulse than that of solid motors, and can be throttled, however they tend to be heavier due to complex plumbing and oxidizer cooling systems. A MATLAB suite was developed to analyze the performance of the Space Shuttle class hybrid booster presented as an example in Sutton [1]. Of interest in booster design is the Maximum Dynamic Pressure (Max q). The functions developed for homework in the Numerical Methods class will be used to compute Max q, time of flight of Max q, and the altitude of Max q.

1. Background and Theory

1.1. Nozzle Performance

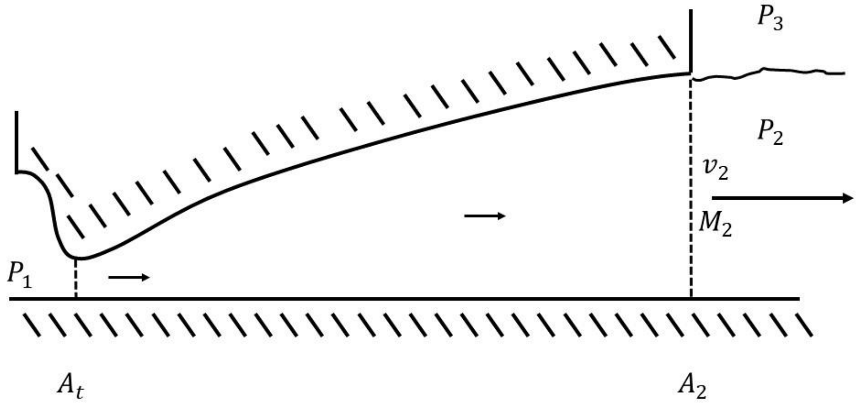

Figure 1.1: Nozzle Schematic

Nozzles convert thermal energy into kinetic energy [1]. The general design of a nozzle is shown above (*Figure 1.1*). Gas is heated in the Combustion Chamber with some pressure P_1, which then is compressed as it flows to the nozzle throat. Compressing the gas accelerates it to Mach 1 due to the Venturi Effect [2] for subsonic flow to the throat. After passing the throat, the gas becomes supersonic, and an increase in cross-sectional area increases the speed of the gas. Gas exits the nozzle with velocity v_2 and pressure P_2. The ambient pressure is denoted P_3. The optimum operating condition of the nozzle occurs when $P_3 = P_2$.

In order to model the flight of a rocket, the thrust is of the most interest. The equation for thrust is

$$F = C_F A_t P_1$$

where C_F is the force coefficient, A_t is the nozzle throat area, and P_1 is the combustion chamber pressure. The throat area is a design parameter, however, in some models the throat area erodes over time. This simulation does not account for that in order to expedite code development. Nozzle throat diameter can be saved as another element of the state, and integrated using the Runge-Kutta scheme to model this (See *Section 3.4. Runge-Kutta Integration*). combustion chamber pressure depends on the combustion of the hybrid motor and is explained in *Section 1.2 Hybrid Motors*. Force coefficient relates exit Mach number to thrust according to the following equation

$$C_F = \sqrt{\frac{2k^2}{k-1}\left(\frac{2}{k+1}\right)^{\frac{k+1}{k-1}}\left(1-\left(\frac{P_2}{P_1}\right)^{\frac{k-1}{k}}\right)} + \frac{P_2 - P_3}{P_1}\left(\frac{A_2}{A_t}\right)$$

where k is the ratio of specific heats of the gas mixture in the nozzle, P_2 is the nozzle exit pressure, P_3 is the ambient pressure, A_2 is the nozzle exit area. The ratio of specific heats k is determined by the regression rate, which is described in Section 1.2 Hybrid Motors. The ambient pressure P_3 is determined according to altitude, see *Section 1.3 Standard Atmosphere and Drag*. The nozzle exit Area A_2 is a design parameter. Again, this can erode over time in a more rigorous model, but erosion is not modeled to expedite development of the code. The exit pressure is calculated according to Nozzle Theory using the following equation

$$P_2 = P_1 / \left(\frac{P_1}{P_2}\right)$$

where the $\left(\frac{P_1}{P_2}\right)$ term is the ratio of combustion chamber pressure to exit pressure, and can be computed according to the equation

$$\left(\frac{P_1}{P_2}\right) = \left(1 + \frac{1}{2}(k-1)M_2^2\right)^{\frac{k}{k-1}}$$

where M_2 is the exit Mach number of the nozzle. The exit Mach number requires numerical methods to solve for, and depends on the expansion ratio, which is the ratio of exit area to throat area. The function is difficult to solve for M_2:

$$\frac{A_2}{A_t} = \frac{1}{M_2}\left(2\frac{\left(1 + \frac{1}{2}(k-1)M_2^2\right)}{(k+1)}\right)^{\frac{1}{2}\left(\frac{k+1}{k-1}\right)}$$

where all of the variables have already been defined. This function is solved using a secant root finding method, see *Section 3.2 Secant Method*. Interestingly, taking the Natural Log of both sides of the equation speeds the convergence of the Secant Method by making the problem look more linear:

$$0 = \log\left(\frac{1}{M_2}\left(2\frac{\left(1 + \frac{1}{2}(k-1)M_2^2\right)}{(k+1)}\right)^{\frac{1}{2}\left(\frac{k+1}{k-1}\right)}\right) - \log\left(\frac{A_2}{A_t}\right)$$

1.2. Hybrid Motors

Hybrid rocket motors store one part of the propellant in liquid form and the other in a solid form. Typically the oxidizer is stored in liquid state in a pressurized tank above the solid fuel grain. Propellant combinations include liquid oxygen (LOX) for oxidizer with hydroxyl-terminated polybutadiene (HTPB) (a rubber) solid grain, pressurized nitrous oxide as oxidizer and paraffin wax or HTPB as the solid grain. Hybrid motors operate by passing the liquid oxidizer over the solid fuel grain as they combust. The solid fuel melts and evaporates away, mixing with the oxidizer in the fuel port, as it combusts. The rate at which the fuel grain melts is called the Regression Rate, and depends on the oxidizer mass flow rate.

The equations specific to the hybrid motor were used to obtain the Combustion Chamber Pressure P_1. The equation is given by Sutton:

$$P_1 = \frac{\dot{m}c^*}{A_t}$$

where $\dot{m}$ is the mass flow rate, c^* is the characteristic velocity, and A_t is the nozzle throat area, the design parameter. The characteristic velocity of the motor depends on the ratio of oxidizer to fuel in the combustion chamber. No function to calculate this value is given in Sutton, however he does provide a table of characteristic velocity values and ratio of specific heat values for a range of propellant ratios. This table was provided for HTPB and LOX, so those were chosen as the fuel type for this hybrid rocket model (see *Section 2.2. Booster Design*). The oxidizer fuel ratio changes over time as the fuel grain regresses, so the values in this table were interpolated using a cubic spline, see *Section 3.5 Cubic Spline* for details. The mass flow rate $\dot{m}$ is computed by:

$$\dot{m} = \dot{m}_o + \dot{m}_f$$

where $\dot{m}_o$ is the oxidizer mass flow rate and $\dot{m}_f$ is the fuel mass flow rate. For the purposes of this simulation, it was assumed that the oxidizer mass flow rate was set to a constant value, chosen as a design parameter. The fuel mass flow rate depends on the regression rate according to

$$\dot{m}_f = \rho_f A_b \dot{r}$$

where ρ_f is the fuel grain density, A_b is the surface area of burning fuel grain, and $\dot{r}$ is the fuel regression rate. The fuel grain density depends on the type of fuel chosen, and is subsequently a design parameter. The surface area of burning fuel grain depends on the port area and geometry. For an equation that describes the burning surface area, see *Section 2.2 Booster Design*, since the surface area depends directly on the geometry of the grain design. The fuel regression rate is the key to solving the problem of hybrid rocket thrust. There are many different methods of solving for regression rate, some of which take into account combustion chamber pressure and injector port diameter, etc. To expedite software development, a simple equation that depends on oxidizer mass velocity was used to compute regression rate

$$\dot{r} = aG_o^n$$

where a and n are empirically determined coefficients, and G_o is the oxidizer mass velocity. Sutton provides values for a and n for HTPB and LOX, which also drove the choice of these propellants for the design. Oxidizer mass velocity is computed according to:

$$G_o = \dot{m}_o / A_p$$

where $\dot{m}_o$ is the oxidizer mass flow rate and A_p is the combustion port area. The combustion port area depends on grain geometry and is described in *Section 2.2 Booster Design*.

1.3. Standard Atmosphere and Drag

The rocket will be flying through the atmosphere on Earth, so the ambient pressure and density must be computed in order to model motor thrust, and drag and dynamic pressure, respectively. The Standard Atmosphere model presented in Anderson [2] was used for this. The Standard Atmosphere is derived from the hydrostatic equation with variable temperature. The hydrostatic equation is given by:

$$dP = g_0 \rho(h) dh$$

where g_0 is the gravitational acceleration at sea level, $\rho(h)$ is the density at altitude, and h is the Geopotential Altitude. The Geopotential Altitude differs from the Geometric Altitude, by adjusting for gravity. Geometric Altitude is the more intuitive altitude measurement, defined as the distance from sea level. The equation for Geopotential Altitude is:

$$h = \frac{h_g R_E}{R_E + h_g}$$

where h_g is the Geometric Altitude, and R_E is the radius of the Earth. The Geopotential Altitude is used for the Hydrostatic Equation because it makes integration easier by allowing gravity to be kept constant with altitude (see Anderson for proof [2]).

The density also changes with altitude. It can be computed using the Ideal Gas Law:

$$\rho = \frac{RT}{P}$$

where R is the Ideal Gas Constant, P is the ambient pressure, and T is the temperature. Temperature is the key to the Standard Atmosphere model. The Standard Atmosphere assumes multiple layers of the atmosphere, alternating between temperature gradient layers and isothermal layers. The temperature of the atmosphere in the Standard Atmosphere varies according to Figure 2.1.

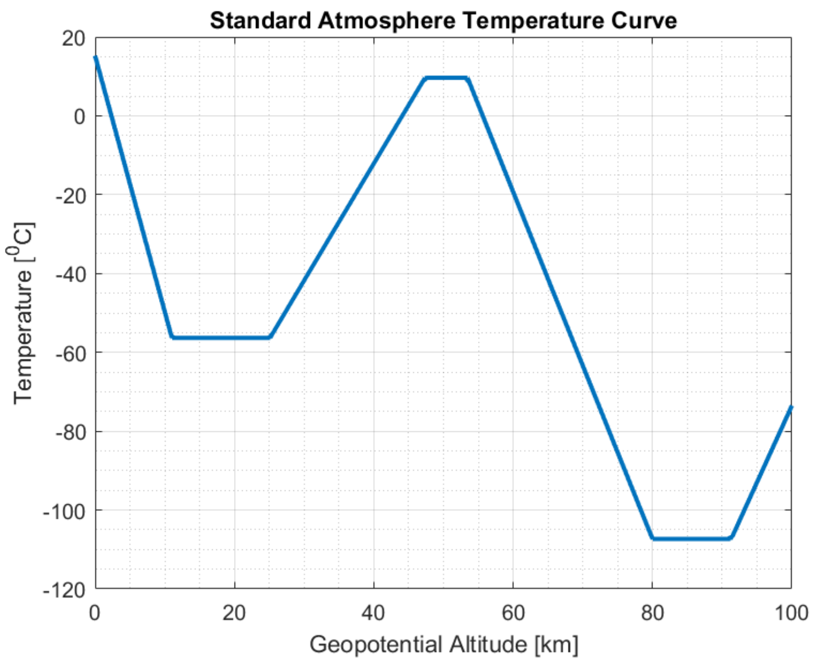

Figure 1.1: Standard Atmosphere Temperature as a function of Altitude, according to the temperature gradient and isothermal layers presented in Anderson [2].

Once the local temperature is obtained from the Standard Atmosphere Model, it can be used to solve the Hydrostatic Equation for pressure, and then to use the Ideal Gas Law to find density. Pressure at a temperature gradient layer is given by:

$$P(h) = P_0 \left(\frac{T(h)}{T_0}\right)^{-\frac{g0}{a(h)R}}$$

where P_0 is the pressure at the minimum altitude of the temperature gradient layer, T_0 is the temperature at the minimum altitude of the temperature gradient layer, g_0 is the sea level gravity, $a(h)$ is the temperature gradient, and R is the ideal gas constant. The temperature gradient is given in Anderson [2] along with the altitudes at which the layers start.

Pressure at the isothermal thermal layers is given by:

$$P(h) = P_0 \exp\left(-\frac{g_0}{(RT_0)(h - h_0)}\right)$$

where P_0 is again the pressure at the start of the layer, R is the Ideal Gas Constant, T_0 is the temperature at the start of the layer, h is the Geopotential Altitude, and h_0 is the altitude of the start of the isothermal layer.

1.4. Differential Equations

The flight of the rocket is modeled by integrating the equations of motion. Position is the integral of velocity, and velocity is the integral of acceleration. Acceleration is computed as the sum of the forces acting on the rocket, including thrust, drag, and gravity. Other forces are neglected to expedite development of the code, such as wind gusts, centrifugal acceleration from the Earth rotation, the J2 effect, etc. To further speed development time, the flight was modeled in only one spatial dimension. In order to compute Drag, the Standard Atmosphere was used (*Section 1.3*). Nozzle theory and the hybrid motor were used to model the thrust. To simulate the flight, these characteristics of the rocket were modeled in a system of differential equations:

$$\begin{bmatrix} v \\ a \\ \dot{r} \\ \dot{m}_o \\ \dot{m}_f \\ \dot{m} \end{bmatrix} = f\left(t, \begin{bmatrix} h \\ v \\ d_b \\ m_o \\ m_f \\ m \end{bmatrix}, p \right)$$

Introduced in this equation are the total rocket mass m, rocket mass flow rate $\dot{m}$, fuel grain width d_b, and parameter vector p. The fuel grain width will be defined in *Section 2.2. Booster Design*. The parameter vector contains the design variables which do not change during flight. This system of equations was solved using a Runge-Kutta integration scheme, see *Section 3.4*.

1.5. Maximum Search

As stated in *Section 2.1. General Objectives*, the maximum dynamic pressure is needed. A local maximum occurs when the derivative of a function is zero. Therefore, the Max q can be found by differentiating the dynamic pressure in time, and solving for the root of the derivative at a point close to where the maximum is expected to be. Finite difference (Section 3.3) or the cubic spline (Section 3.5) can be used to calculate the derivative of the function. The Secant Method can be used to search for roots of the dynamic pressure rate of change. However, Runge-Kutta integration returns a solution to the differential equation in the form of a discrete set of values. Finite difference is not continuous either. This poses a problem for the Secant Method, which requires a continuous function. The cubic spline is used to fix this problem by interpolating between integration points.

2. Design

2.1. General Objectives

The objectives of this project are to model the flight of a hybrid rocket booster and to determine the Maximum Dynamic Pressure (Max q). Also of importance are the time and altitude at which the Maximum Dynamic Pressure occurs. The highest drag forces experienced by the rocket during flight are experienced at Max q. In fact, Max q is the point of highest structural loads on the rocket due to the high drag. An ideal design for a booster would have Max q occur at a very high altitude, where the pressure and density are low, hence the force of drag on the rocket will be low.

Also of interest is the maximum altitude the booster can achieve and the maximum velocity. Altitude and velocity give the most intuitive measure of the performance of the booster.

2.2. Booster Design

The booster for this particular hybrid rocket simulation was designed similarly to the Space Shuttle-class hybrid booster presented in Example 16.4 of Sutton [1]. This booster has 3.1 million pounds of thrust in a vacuum, with a total burn time of 120 seconds. The radius of the booster was assumed to be 160 inches. The structural mass of the rocket is assumed to be 5100 kg, however no structural analysis whatsoever was used to obtained this number; it was a wild guess. However, accuracy of the structural mass is not necessary to validate the flight simulation. For future use, however, it would be prudent to have structural analysis done before simulating the flight.

The propellant design consists of HTPB rubber for the solid fuel grain, and LOX for the liquid oxidizer. This propellant combination was chosen because the specific heat ratio, characteristic velocity, and fuel regression rate coefficient data are readily available in Sutton. Table 2.1 displays the specific heat ratio and characteristic velocity as a function of oxidizer to fuel ratio. The initial oxidizer flow rate was designed to yield an initial mass mixture ratio of 2.0.

Table 2.1: Theoretical Characteristic Velocity and Specific Heat Ratio for HTPB / LOX Ratio

Mass Mixture Ratio	c^* (ft/s)	k
1.0	4825	1.308
1.2	5180	1.282
1.4	5543	1.239
1.6	5767	1.201
1.8	5882	1.171
2.0	5912	1.152
2.2	5885	1.143
2.4	5831	1.138
2.6	5768	1.135
2.8	5703	1.133
3.0	5639	1.132

Additionally, for HTPB/LOX propellant the fuel regression rate equation is given by Sutton as:

$$\dot{r} = aG_0^n$$

with the coefficients $a = 3.045015375e^{-05}$ and $n = 0.680825577$ [1].

The fuel grain was designed with seven burn ports in order to increase burning surface area. Figure 2.1 displays a diagram of the fuel grain cross section. The ports are all designed to be circular, which actually wastes space, as the burning geometry is still considered to be perfectly circular. A more efficient design would be to use trapazoidal shaped ports around the outer diameter of the grain. However, circular ports were used to again expedite development. Seven fuel ports give a port area of:

$$A_p = N\pi\big(R_0 + d_{b0} - d_b(t)\big)^2$$

where N is the number of fuel grain ports, R_0 is the initial port radius, d_{b0} is the initial fuel grain width, and $d_b(t)$ is the fuel grain width as a function of time. The variable $d_b(t)$ is solved for during Runge-Kutta integration of the system of ordinary differential equations described in *Section 1.4 Differential Equations*.

The burning surface area is given by:

$$A_b = 2N\pi\big(R_0 + d_{b0} - d_b(t)\big)L$$

where L is the length of the fuel grain.

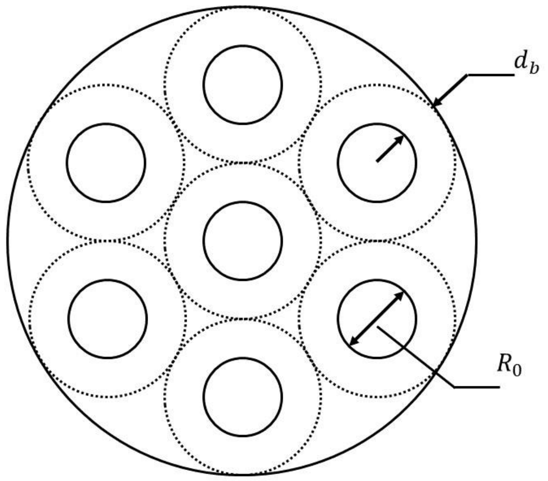

Figure 2.1: Fuel Grain Cross Section with seven burn ports

3. Numerical Methods

3.1. Gauss Elimination

The Gauss Elimination function developed as a solution to Homework Assignment 2 was used to solve all linear equations in this code. Minor modifications to the function were made to decrease the runtime of the code. During row elimination, if an element in the pivot column is zero, that row is skipped. This is meant to increase the speed of the algorithm for sparse matrices, such as those used in the cubic spline function. Partial pivoting is used for when a pivot element is zero valued. The Gauss Elimination function is a dependency of the cubic spline function and the finite difference function.

3.2. Secant Method

The Secant Method, developed for Homework Assignment 1, was adapted into a function and used to solve for roots to equations. This method is advantageous over Newton's Method because it is guaranteed to converge if the convergence criteria is met, and doesn't require the function derivative. The Secant Method was used to solve for the exit Mach number given the expansion ratio in the thrust calculations, and to find the time of Max q by finding the root of the time rate of change of the dynamic pressure curve.

3.3. Finite Difference Derivative

A central finite difference function was developed from the solution to Homework Assignment 6. This function was used to compute the time rate of change of the dynamic pressure after solving the system of differential equations. The central finite difference was used where possible, and was taken as the average between the forward finite difference and backwards finite difference. Gauss Elimination was used to solve the Taylor Series approximate for any given order of accuracy. At the beginning and end of the dataset, the forward and backwards finite difference methods were used, respectively, as the central finite difference method would be unable to be calculated due to insufficient data on either side of the data point. Ultimately, a central finite difference of Order 2 was used to calculate the derivative. The lower order finite difference method was used because it seemed to work better for discontinuities in the differential equation.

3.4. Runge-Kutta Integration

The Runge-Kutta function which was the solution to Homework Assignment 6 was adapted for the hybrid rocket flight simulator. This function acts as the workhorse which drives the simulation. The code was modified to include the option of specifying the number of integration steps, with an option to specify a custom Butcher Tableau. Lower order methods were used for debugging the code, while the higher order was used to obtain the final results presented in this paper. Specifically, a six step, fifth order Dormand-Prince Runge-Kutta integration scheme was used, which incorporates polynomial interpolation between the integration steps. A time step of one-tenth of a second was used to integrate the system of equations.

3.5. Cubic Spline

The cubic spline function which was the solution to Homework Assignment 5 was used to interpolate points. The function was modified from the homework code to output the first derivative of the interpolated values in addition to the interpolation value itself. This was done to facilitate a local maximum search for Max q. The function was also used to interpolate values presented in Table 2.1 for the hybrid motor thrust and mass flow rate calculations. The cubic spline function suffers from a severe inefficiency however. Every time the function is called, a system of linear equations must be solved to obtain the spline data. This becomes inefficient when the spline function is called many times, for example, during the Secant Root solver function. This inefficiency could be fixed in further development of the code, but was left in this iteration of the simulator in order to expedite development time.

4. Results

4.1. Altitude and Velocity Curves

Preliminary results of the simulation included altitude and velocity curves. The rocket achieved a maximum altitude of approximately 2500 kilometers (Figure 4.1), achieving a maximum speed of approximately 6.5 kilometers per second (Figure 4.2). The flight was simulated to include free fall through the atmosphere. At the end of the flight at approximately 1500 second, the rocket falls into the dense part of the atmosphere and experiences high acceleration to slow it down, which can be seen in Figure 4.2.

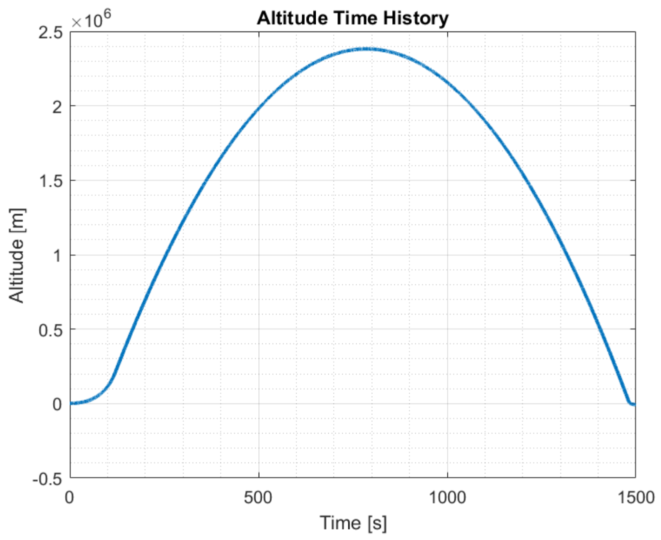

Figure 4.1: Altitude Profile for duration of simulation

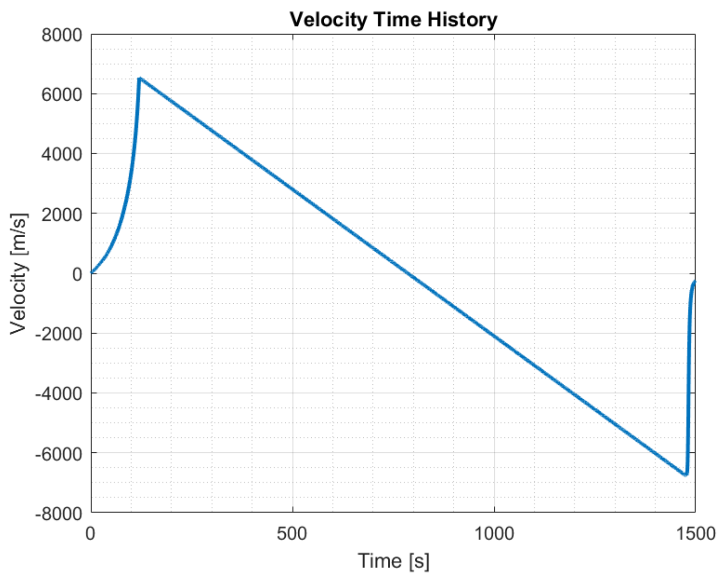

Figure 4.2: Velocity Profile for duration of simulation

Of interest are the altitude and velocity profiles during ascent. Ascent is considered to be 120 seconds, or the duration of the burn. Figure 4.3 shows that the rocket achieves an altitude of 20 kilometers at burnout, and Figure 4.4 shows a maximum burnout velocity of 6.5 kilometers per second. This booster design is not quite efficient enough to reach orbit, which requires approximately 7.7 kilometers per second for Low Earth Orbit.

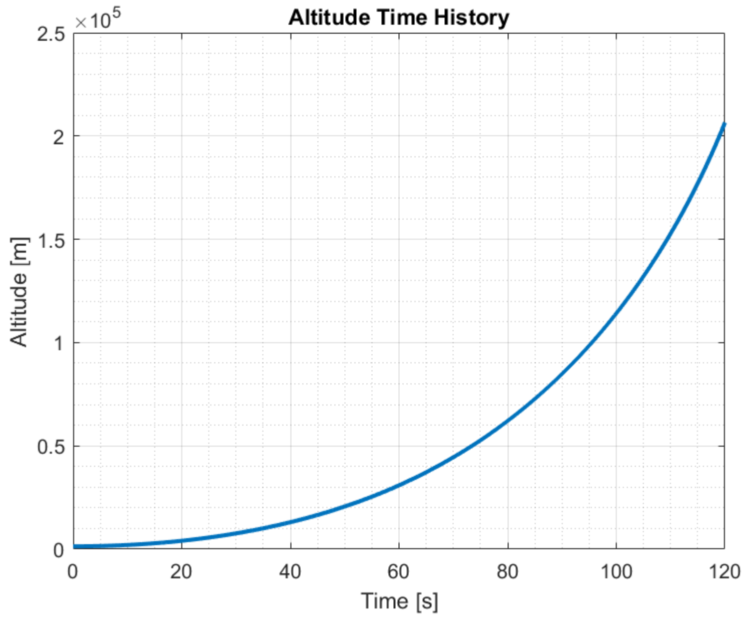

Figure 4.3: Altitude Profile during Ascent.

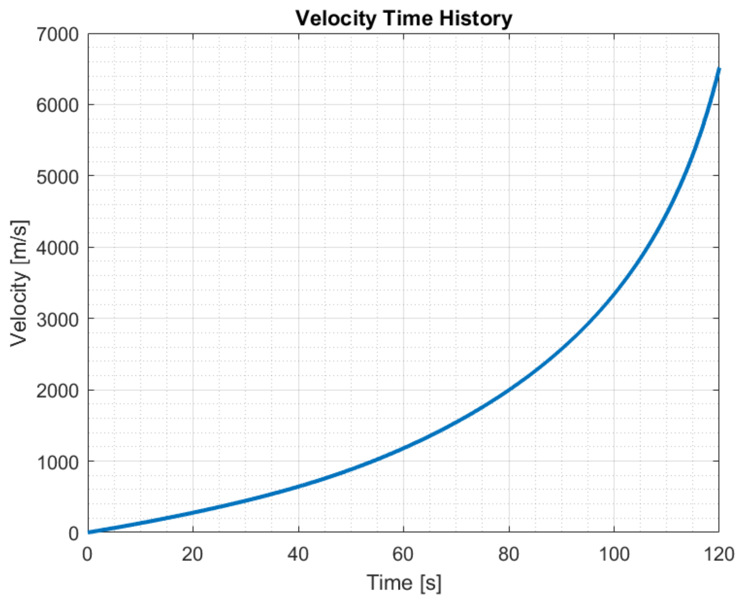

Figure 4.4: Velocity Profile during Ascent.

4.2. Acceleration Curves

Also of importance is the acceleration profile during flight. Acceleration was not saved to the rocket state, but was instead returned from the Runge-Kutta integration function as the function solved the system of equations. FIgure 4.5 shows two acceleration peaks. One occurs at burnout at 120 seconds into the flight. The second occurs at about 1500 seconds, when the rocket falls back into the lower parts of the atmosphere. Of more interest is the acceleration at burnout. Figure 4.6 shows that the maximum acceleration at burnout is 300 meters per second, or about 30 times the force of gravity. This acceleration would put a lot of loading on the structures in the rocket, and it would be worth it in future designs to experiment with throttling down the hybrid motor oxidizer flow rate to prevent this high acceleration.

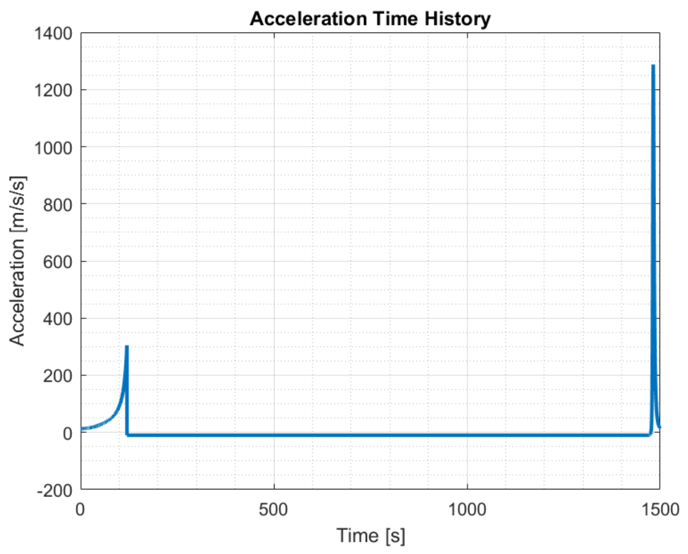

Figure 4.5: Acceleration Profile for duration of simulation.

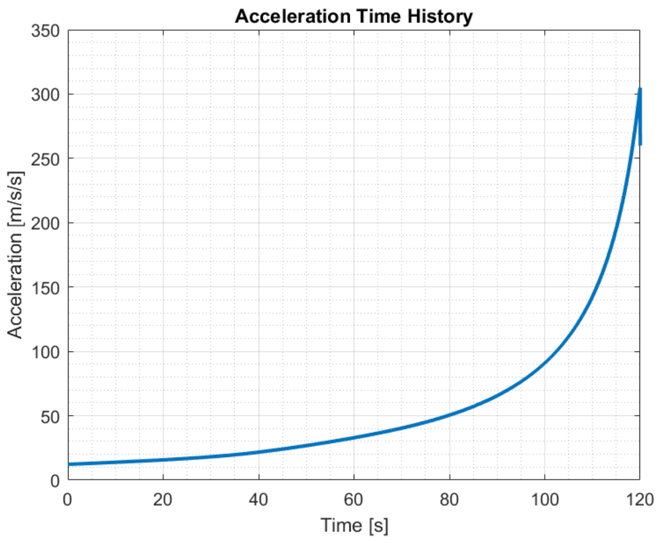

Figure 4.6: Acceleration Profile during Ascent.

4.3. Engine Performance

The fuel grain width was saved as a part of the state of the hybrid rocket motor that was governed by the differential equation outlined in *Section 1.4 Differential Equation*. Fuel grain regression rate determines performance of the rocket. The regression rate during ascent is shown in Figure 4.7. The regression rate decreases as the port area increases (d_b decreases with time more slowly). Flight performance of the hybrid rocket can be improved by experimenting with throttling the motor, however care must be taken to keep the mass ratios within the bounds of the data provided by Sutton. The fuel and oxidizer mass flow rates are shown in Figure 4.8, which can be used to verify that the design is not about to violate the bounds of Table 2.1. According to Figure 4.8, it looks like the oxidizer flow rate can be reduced by as much as 25% to try to throttle back the motor before burnout.

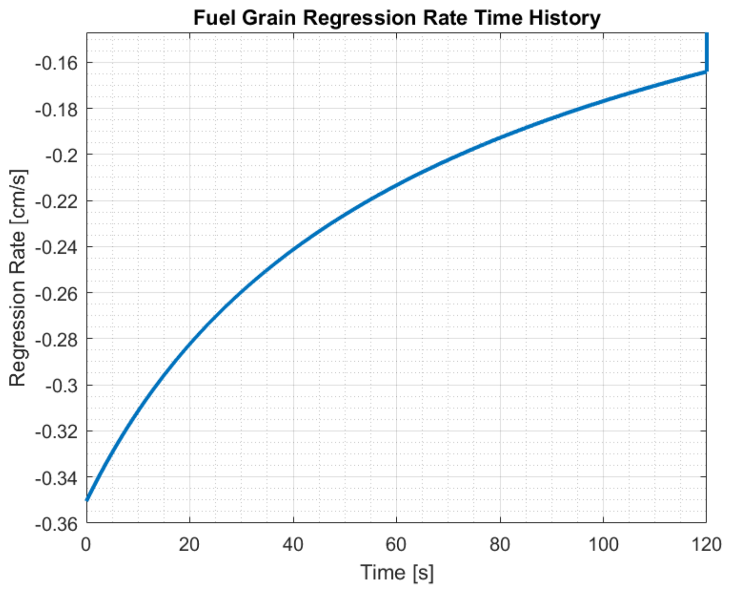

Figure 4.7: Fuel grain regression rate profile during ascent.

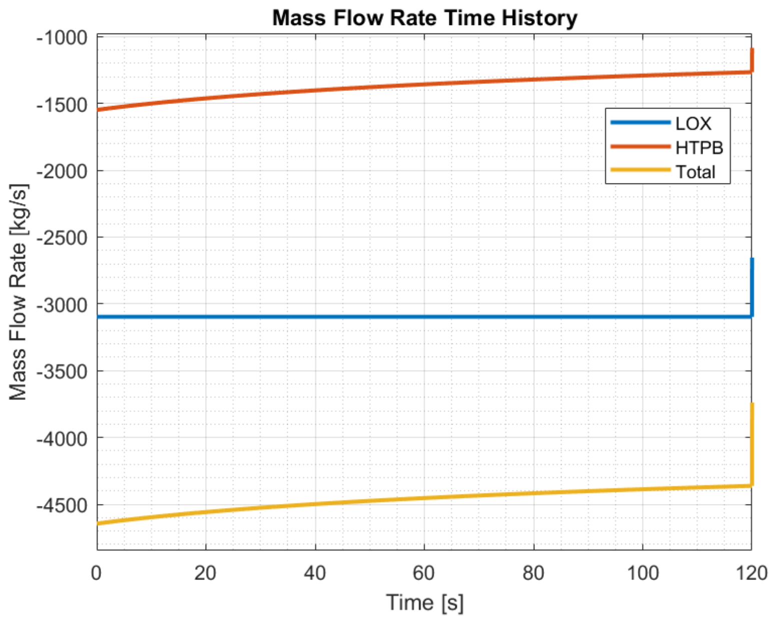

Figure 4.8: Mass flow rate profiles during ascent.

4.2. Dynamic Pressure and Drag

The final analysis performed on the simulation is the calculation of Max q. The dynamic pressure profile during ascent is displayed in Figure 4.9. It is important to note that there is a second maximum for dynamic pressure on descent during the intense acceleration at 1500 seconds. However, the important part of the flight analysis is during ascent, so the descent dynamic pressure is neglected. The output of the code indicates that Max q occurs at 36.23

seconds into the flight, and at an altitude of 10.785 kilometers. The Max q value is 59.55 kilo-Pascals, which yields a total maximum drag force on acting on the rocket of 772.5 kilo-Newtons.

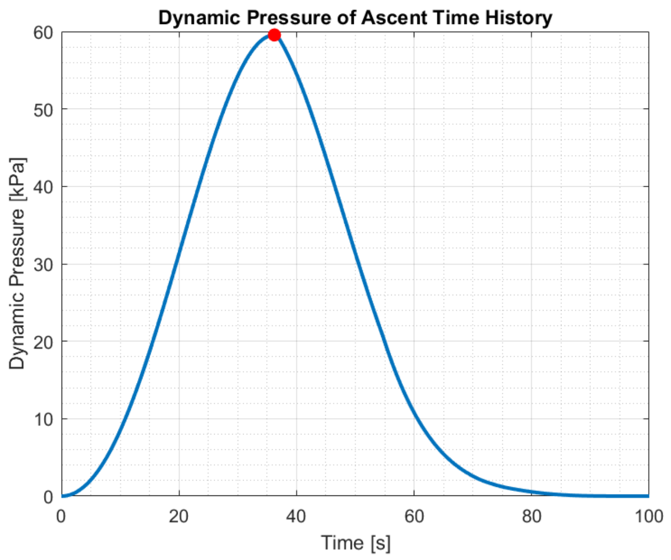

Figure 4.9: Dynamic pressure profile during ascent.

Max q was actually quite difficult to solve for. Interestingly, Max q occurred less then half a kilometer from the boundary between a temperature gradient layer and an isothermal layer of the atmosphere. The boundary is located at 11 kilometers altitude, and is the first transition between layers in the atmosphere. This caused a jump discontinuity in the derivative atmospheric density, which played havoc with the central finite difference method of calculating the derivative. The rate of change of the dynamic pressure is displayed in Figure 4.10 around this point, using the central finite difference method and a cubic spline. The change from temperature gradient to isothermal region affected the derivative so much that interpolating the finite difference derivative returned an incorrect result for the maximum. The cubic spline was affected less, so the spline was ultimately used to search for the local maximum.

A closer inspection of the maximum dynamic pressure was required to verify that Max q was computed correctly. The spline interpolation is displayed over top the integrated dynamic pressure in Figure 4.11. Also, in Figure 4.11 the temperature gradient and isothermal layer boundary can be seen where the cubic spline deviates from the true dynamic pressure curve.

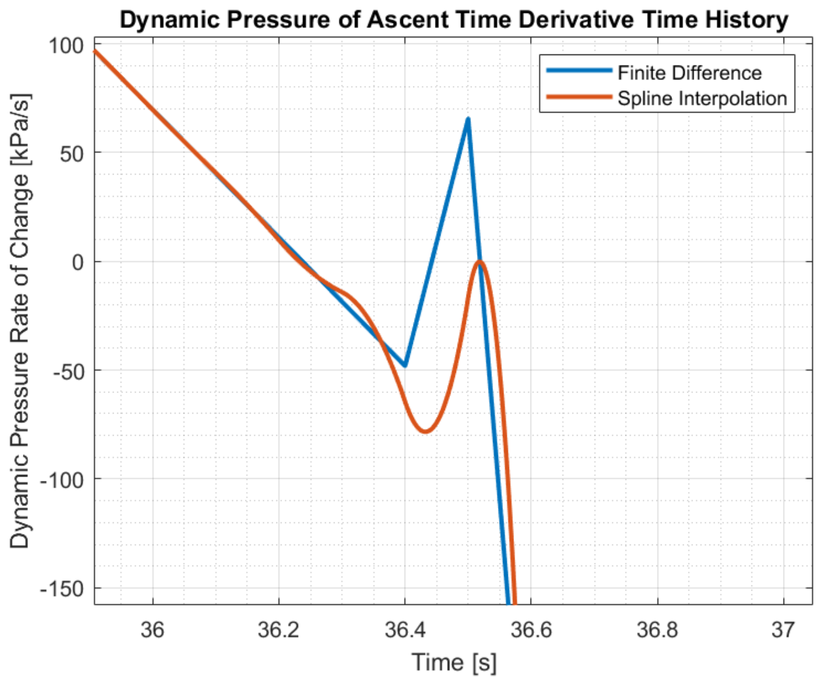

Figure 4.10: Time rate of change of dynamic pressure using two derivative calculation methods, near the temperature gradient and isothermal layer boundary.

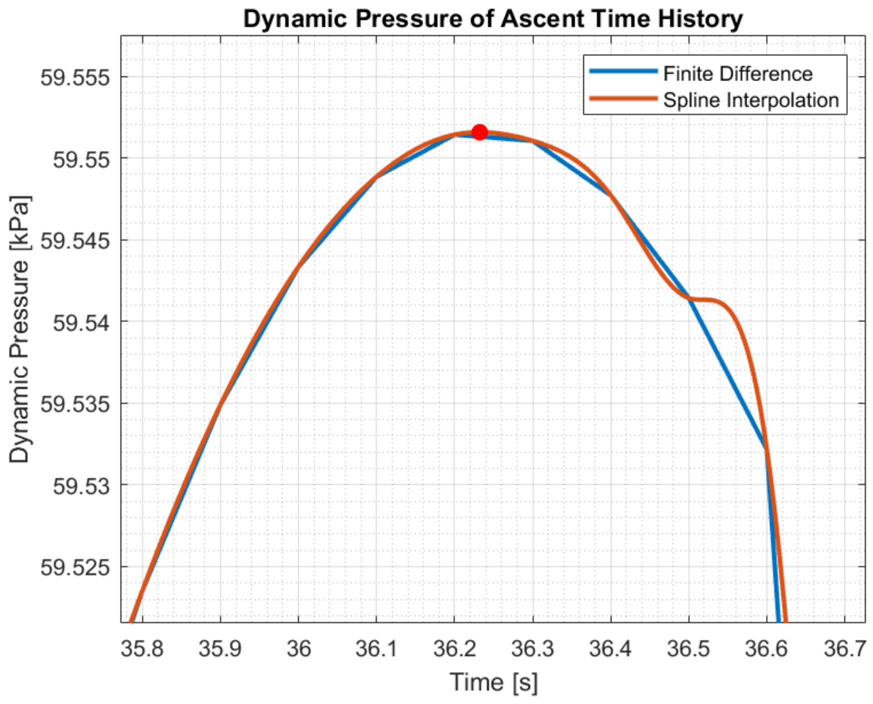

Figure 4.11: Cubic spline interpolation of maximum dynamic pressure near the temperature gradient boundary of 11 kilometers. Notice the jump in the cubic spline interpolation of the function.

5. Conclusions

The flight performance of a hybrid rocket motor was successfully simulated with the numerical methods presented in AOE 4404 Numerical Methods. Although there were some numerical caveats, the maximum dynamic pressure was calculated successfully using a combination of many numerical methods. Many of the models used to simulate the flight were simplified in order to expedite development time of the code, however. Given more time, a more comprehensive model can be developed. It would be useful to be able to model the three dimensional flight of a rocket eventually, and perhaps to run Monte-Carlo simulations to predict landing locations of high powered rockets. There are still some inefficiencies in the functions, namely the cubic spline, that should be fixed to improve run time. Eventually, this code may be developed to aid Rocketry at Virginia Tech in the development of a high powered hybrid rocket motor for use in the Intercollegiate Rocket Engineering Competition (IREC). Since the numerical methods don't rely on MATLAB functions, it may also be relatively easy to convert the code to C++ or python for use on the flight computer to predict altitude for active motor throttle in IREC. This could potentially make achieving the target apogee much easier. Additionally, the modifications of the functions from the homework allow the functions to be used in other scenarios. For example, since the cubic spline function was changed to return the interpolated derivative value, the cubic spline can now be used in an optimization scheme or an Extended Kalman Filter [3].

References

[1] Sutton, G. P., Biblarz, O., “Rocket Propulsion Elements”, 8-th Edition, Copyright 2010, John Wiley & Sons, Inc.

[2] Anderson J. D. Jr., "Introduction to Flight", 8-th Edition, Copyright 2016, McGraw Hill.

[3] Bar-Shalom, Y., “Estimation with Applications to Tracking and Navigation”, copyright 2001, John Wiley & Sons Inc.

Appendix A: Main Code

Below is the code from the Main function, which contains parameters used to model the flight of the rocket. It would be unreasonable to copy all of the code into this pdf however, as 25 different functions were used in the simulation.

```
%% main_HRFS.m
%
% Driver script for the Hybrid Rocket Flight Simulation.
%
% This function simulates the flight of a hybrid rocket to determine the
% maximum altitude that can be achieved with the given parameters. This
% suite is the project for Virginia Tech AOE 4404: Numerical Methods.
%
% Eventually (hopefully) this MATLAB suite will be converted to C++ to
% predict the altitude of the rocket for Rocketry@VT.
%
% @author: Matt Marti
% @date: 2019-05-07

clear, clc, clear global
include_HRFS('.');
constants_HRFS;


%% Parameters

% Integration parameters
t0 = 0;
tend = 1500; % Maximum time of simulation
dt = 1e-1; % Simulation delta time step
nRK = 6; % Runge-Kutta integration number of steps

% Differentiation parameters
nFD = 2; % Order of finite difference differentiation

% Dynamic Pressure Root Search Parameters
t0_maxq_1 = t0; % Ascent Start
tend_maxq_1 = t0 + 100; % Ascent End
ta_maxq_1 = t0_maxq_1 + 20;
tb_maxq_1 = tend_maxq_1 - 40;

% Rocket initial position and velocity conditions
pos0 = 1400; % [m] Altitude, Spaceport America, New Mexico
vel0 = 0; % [m/s] Initial velocity

% Oxidizer Mass Flow profile
modot_fun = @(tk) 3096.55536595829;

% Rocket Mass
me = 5100; % [kg] Empty mass
mo0 = 3.7159e+05; % [kg] Oxidizer mass
% mf = 0; % [kg] Fuel mass (Calculated from fuel density and geometry)
m_error_margin = mo0*1e-4;

% Propellant Characteristics for HTPB/LOX
etacombust = 0.95; % Combustion efficiency
rhof = 913.436110336788; % [kg/m^3] Fuel density (Thrust)
a = 3.045015375e-05; % Fuel grain regression coefficient (Thrust)
n = 0.680825577; % Fuel grain regression exponent (Thrust)
regress_fun = @(Go) a*(Go^n);
rsplinevec = (1:.2:3)'; % Mass Mixture Ratio spline data
cstarsplinedata = [
    4825;
    5180;
    5543;
    5767;
    5882;
    5912;
    5885;
    5831;
    5768;
    5703;
    5639] * 0.3048; % [m/s] Characteristic velocity spline data
ksplinedata = [
    1.308;
    1.282;
    1.239;
    1.201;
    1.171;
    1.152;
    1.143;
    1.138;
    1.135;
    1.133;
    1.132]; % [no units] Specific heat ratio spline data

% Fuel Grain Design
Nt = 7; % Number of fuel grain ports
L = 30.2371738789983; % [m] Fuel grain length
R0 = 0.363557686863076; % [m] Initial Port radius
db0 = 0.271442313136924; % [m] Initial Fuel Grain Width
Ap_fun = @(db) pi * Nt * (R0+db0-db)^2; % Port area function
Ab_fun = @(db) Nt * 2 * (R0+db0-db) * pi * L; % Burn area function
mf0 = rhof*Nt*L*pi*((db0+R0)^2 - db0^2);

% Nozzle Parameters
At = 1.64750631789099; % [m^2] Nozzle throat area (Thrust)
A2 = 12.7187487741184; % [m^2] Nozzle Exit area (Thrust)

% Areodynamic Parameters
Af = pi*(160/12*.3048)^2; % [m^2] Rocket front area (Drag)
Cd = 0.25; % CD for powered flight
RE = 6.356766e6; % [m] Earth Radius
g0 = 9.807; % [m/s/s] Earth sea level gravity acceleration
atm_fun = @(h) stdAtmosphereCalc_hgp(h, RE, g0);


%% Simulate flight

% Flight initial conditions
m0 = me + mo0 + mf0;
x0 = [ pos0; vel0; db0; mo0; mf0; m0 ];

% Compile parameter values
p = struct( ...
        'rhof', rhof, ... % Fuel density
        'regress_fun', regress_fun, ... % Fuel regression rate function
        'eta', etacombust, ... % Fuel/Oxidizer burn efficiency
        'At', At, ... % Throat Area
        'A2', A2, ... % Nozzle Exit Area
        'Ap_fun', Ap_fun, ... % Port area function
        'Ab_fun', Ab_fun, ... % Burn area function
        'Af', Af, ... % Frontal area
        'Cd', Cd); % Drag coefficient

% Differential equation to be solved
xdot_fun = @(tk, xk) fHRFSdynamics_1D(tk, xk, p, modot_fun, ...
    rsplinevec, cstarsplinedata, ksplinedata, atm_fun);

% Integrate using Runge-Kutta
tlims = [t0; tend];
[xhist, thist, xdothist] ...
    = mmatth3_rungekuttaint_fun(xdot_fun, x0, tlims, dt, nRK);
Nt = length(thist);

% Parse integration output
hhist = xhist(1,:);
vhist = xhist(2,:);
ahist = xdothist(2,:);
dbhist = xhist(3,:);
```

```
mohist = xhist(4,:);
mfhist = xhist(5,:);
mhist = xhist(6,:);
rdothist = xdothist(3,:);
modothist = xdothist(4,:);
mfdothist = xdothist(5,:);
mdothist = xdothist(6,:);


%% Plots

% Altitude curve
figure(1); hold off
plot(thist, hhist, 'linewidth', 2);
title('Altitude Time History');
xlabel('Time [s]')
ylabel('Altitude [m]');
grid on, grid minor

% Velocity curve
figure(2); hold off
plot(thist, vhist, 'linewidth', 2);
title('Velocity Time History');
xlabel('Time [s]')
ylabel('Velocity [m/s]');
grid on, grid minor

% Acceleration curve
figure(3); hold off
plot(thist, ahist, 'linewidth', 2);
title('Acceleration Time History');
xlabel('Time [s]')
ylabel('Acceleration [m/s/s]');
grid on, grid minor

% Fuel grain width curve
figure(4); hold off
plot(thist, dbhist*1e2, 'linewidth', 2);
title('Fuel Grain Width Time History');
xlabel('Time [s]')
ylabel('Fuel Grain Width [cm]');
grid on, grid minor

% Fuel mass curve
figure(5); hold off
plot(thist, [mohist; mfhist], 'linewidth', 2);
title('HTPB/LOX Mass Time History');
xlabel('Time [s]')
ylabel('Mass [kg]');
legend({'LOX', 'HTPB'});
grid on, grid minor

% Fuel Grain Regression Rate curve
figure(6); hold off
plot(thist, rdothist*1e2, 'linewidth', 2);
title('Fuel Grain Regression Rate Time History');
xlabel('Time [s]')
ylabel('Regression Rate [cm/s]');
grid on, grid minor

% Mass flow rate curve
figure(7); hold off
plot(thist, [modothist; mfdothist; mdothist], 'linewidth', 2);
title('Mass Flow Rate Time History');
xlabel('Time [s]')
ylabel('Mass Flow Rate [kg/s]');
legend({'LOX', 'HTPB', 'Total'});
grid on, grid minor

% Mass flow ratio
figure(8); hold off
```

```
plot(thist, modothist ./ mfdothist, 'linewidth', 2);
title('Mass Flow Rate Ratio Time History');
xlabel('Time [s]')
ylabel('modot / mfdot');
grid on, grid minor

% Total Mass Time History
figure(9); hold off
plot(thist, mhist, 'linewidth', 2);
title('Total Mass Time History');
xlabel('Time [s]')
ylabel('Rocket Mass [kg]');
grid on, grid minor


%% Dynamic Pressure Analysis

% Compute dynamic pressure
Dhist = zeros(Nt, 1);
qhist = zeros(Nt, 1);
rhohist = zeros(Nt, 1);
Thist = zeros(Nt, 1);
for i = 1:Nt
    [~, rhoi, Thist(i)] = stdAtmosphereCalc_hgp(hhist(i), RE, g0);
    [Dhist(i), qhist(i)] = dragCalc(Cd, Af, rhoi, vhist(i));
    rhohist(i) = rhoi;
end
qdothist = mmatth3_finitedifference_fun(qhist, dt, nFD);
qdotdothist = mmatth3_finitedifference_fun(qdothist, dt, nFD);

% Develop spline interpolation based function for first dynamic pressure
thist_1 = t0_maxq_1:dt:tend_maxq_1;
Nq_1 = length(thist_1);
qhist_1 = qhist(1:Nq_1);
qdothist_1 = qdothist(1:Nq_1);
qdotdothist_1 = qdotdothist(1:Nq_1);
qdotlim = [qdothist(1); qdothist(Nq_1)];
q_1_fun = @(t) mmatth3_cubicspline_fun(...
    thist_1, qhist_1, t, qdotlim);
qdotdotlim = [qdotdothist(1); qdotdothist(Nq_1)];
qdot_1_fun = @(t) mmatth3_cubicspline_fun(...
    thist_1, qdothist_1, t, qdotdotlim);
qdot_2_fun = @(t) anonymousFuncSecondArg(q_1_fun, t);
D_1_fun = @(t) mmatth3_cubicspline_fun(...
    thist_1, Dhist(1:Nq_1), t, qdotlim);
thist_2 = t0_maxq_1:dt/100:tend_maxq_1;

% Ascent Dynamic Pressure curve
figure(10); hold off
plot(thist_1, qhist_1*1e-3, 'linewidth', 2);
title('Dynamic Pressure of Ascent Time History');
xlabel('Time [s]')
ylabel('Dynamic Pressure [kPa]');
grid on, grid minor

% Search for first dynamic pressure maximum
[t_qmax_1, niter, erra] = mmatth3_secantrootsolve_fun(...
    qdot_2_fun, ta_maxq_1, tb_maxq_1);
qmax_1 = q_1_fun(t_qmax_1);
hold on
plot(t_qmax_1, qmax_1*1e-3, 'r.', 'markersize', 25);

% Ascent Dynamic Pressure Derivative curve
figure(11); hold off
plot(thist_1, qdothist_1, 'linewidth', 2);
hold on
plot(thist_2, qdot_2_fun(thist_2), 'linewidth', 2);
title('Dynamic Pressure of Ascent Time Derivative Time History');
xlabel('Time [s]')
ylabel('Dynamic Pressure Rate of Change [kPa/s]');
legend({'Finite Difference', 'Spline Interpolation'})
```

```
grid on, grid minor

% Output maximum dynamic pressure
fprintf('Max Q for Ascent Time: %.2f [s]\n', t_qmax_1);
fprintf('Max Q for Ascent Value: %.2f [kPa]\n', 1e-3*qmax_1);
fprintf('Drag Force at Max Q: %.2f [kN]\n', 1e-3*D_1_fun(t_qmax_1));

% Altitude of Max Q
h_fun = @(t) mmatth3_cubicspline_fun(...
    thist_1, hhist(1:Nq_1), t);
fprintf('Altitude of Max Q: %.3f [km]\n', 1e-3*h_fun(t_qmax_1));

% Ascent Dynamic Pressure curve 2
figure(12); hold off
plot(thist_1, qhist_1*1e-3, 'linewidth', 2);
hold on
plot(thist_2, q_1_fun(thist_2)*1e-3, 'linewidth', 2)
plot(t_qmax_1, qmax_1*1e-3, 'r.', 'markersize', 25);
title('Dynamic Pressure of Ascent Time History');
xlabel('Time [s]')
ylabel('Dynamic Pressure [kPa]');
legend({'Finite Difference', 'Spline Interpolation'})
grid on, grid minor
```