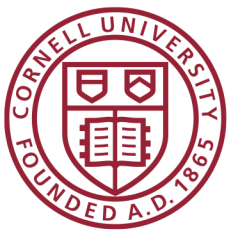
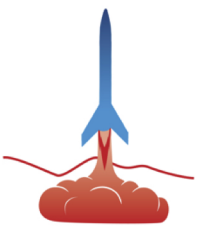
Spring 2021 Technical Report

Cornell Rocketry Team
MATLAB Dynamics Model
Structures

Cornell University
Sibley School of Mechanical and Aerospace Engineering
141 Hoy Road
Ithaca, NY 14850





SPACEPORT AMERICA®
CUP

Summary

1 Technical Report

1.1 System Overview

1.1.1 System Function

This system's function is to create and simulate a 1-dimension dynamics model of the launch vehicle using MATLAB and simulink. The goal of the model is to estimate the trajectory of the launch vehicle with different nose cones and their associated various drag profiles.

This system's parent is the launch vehicle and its children are the following nose cones:

- Von Karman 5.5:1
- Ogive 5:1
- Ogive 4:1
- Ogive 3:1
- Conical 5:1
- Conical 4:1

The aforementioned nose cones' drag profiles were completed in a different team member's previous ANSYS analysis project, and its results are an assumption for this dynamics model.

The constraints to this model are the physical properties of the launch vehicle and nose cones being tested. In particular, constraints consisted of the generated thrust, mass of launch vehicle, and mass of propellant.

1.1.2 MATLAB Model

The functionality of the MATLAB model is as follows: establish initial launch vehicle variables and then determine how each variable changes over the duration of a launch. By doing this, altitude can be determined.

The model begins by establishing these 'base conditions,' such as the launch vehicle's cross-sectional area and propellant weight. Such values were calculated prior to the model. Further primary conditions are set – initial altitude and velocity are both 0. In order to calculate and plot changing variables throughout flight, it was established that the model would calculate values every 0.2 seconds.

The function 'myrhs,' or 'my right hand side,' produces the value zdot, which is BLAH. First, struct values are unpacked and placed into variables. Next, the variables 'y' and 'v' are created to keep track of altitude and velocity over time. In order to calculate the thrust at any arbitrary timestamp throughout the flight, the 'linearinterp' function is called with an attached spreadsheet. Linearinterp works by using Euler's method of approximation to estimate thrust at precise times, with a known general thrust curve:

$$y_n = y_{n-1} + (t_n - t_{n-1})y'(t_{n-1}, y_{n-1}) \tag{1}$$

The 'totImp' variable, which is total impulse, is calculated by integrating force (ie; thrust) with respect to time:

$$Impulse = \int F\,dt \tag{2}$$

With the aforementioned variables and functions created, the model then determines the mass of the launch vehicle:

$$Mass = (TotalWeight) - (\frac{\int_{t_0}^{t} T dt}{\int_{t_0}^{t_{end}} T dt} \cdot PropellantWeight) + (EmptyWeight) \tag{3}$$

This equation works by finding impulses at all times throughout the launch. Then the ratio of impulses is calculated to determine the fraction of propellant burned. By multiplying the fraction burned by the total weight, the amount of propellant burned at time t is determined. Subtracting propellant burned from the total weight gives the mass of the total launch vehicle at time t.

In order to calculate drag, the velocity in terms of mach was calculated by:

$$Mach = \frac{velocity}{340.29} \tag{4}$$

In order to calculate drag, density is calculated by the following equation:

$$\rho = \rho_t((1 + \frac{\gamma - 1}{2} M^2))^(\frac{-1}{\gamma - 1}) \tag{5}$$

With all values necessary to calculate drag established, the model then determines drag:

$$Drag = \frac{Cd \cdot \rho \cdot V^2 \cdot A}{2} \tag{6}$$

After calculating drag, the model can then determine the total net force on the launch vehicle. The net force is equal to thrust minus the gravitation force minus the drag force:

$$\sum F = T - mg - D \tag{7}$$

Following this calculation, velocity and acceleration (ydot and vdot respectively) are placed inside of the zdot array to be stored and later utilized. With the zdot array filled, zarray and an array of timestamps are placed into the 'ode2' differential equation for calculations. The result of ode2 are arrays of altitude (y_array) and velocity (v_array). Both of these arrays of values are then graphed versus time for a complete 1-D dynamic model representation of the launch vehicle.

1.1.3 MATLAB Model Code

%%%%%%% OUTLINE.m %%%%%%%

```
clc
clear all

p.Cd = 0.4;      %populating struct with values
p.g = 9.8;
p.A = 0.01929028; % in m^2
p.gamma = 1.4;
p.propWeight = 20.00342; % in kg
p.lvWeight = 57.447; % in kg

tend = 1023;              %time elapsed
npoints = 5;          %number of timesteps per second
ntimes = tend * npoints + 1;%total number of timesteps
tarray = linspace(0, tend, ntimes); %creates array of the time values
h = tend/ntimes;        %time variable for use in Euler method later

y0 = 0;
v0 = 0;
z0 = [y0; v0];

eqn = @(t,z) myrhs(t,z,p); %calls ODE

% Establish the max time and the tolerances
options.maxtime = 10;

tolerances = [1, 0.5, 0.25, 0.125, 0.0625, 0.03125];

t_Midpoint = []; % initializing the t_Midpoint vector

for i = 1:length(tolerances)
    options.RelTol = tolerances(i); % communicates with ODE1 and ODE2

    tic % start keeping track of time for options.maxtime

    % Use the Midpoint Method to solve the Forced Damped Oscillator
    [tarray, zarray] = ode2(eqn, tspan, z0, options);

    t_Midpoint(i) = toc;

end

y_array = zarray(1);
v_array = zarray(2);
```

```
hold on
figure(1)
title('Altitude vs time')
plot(y_array, tarray);
xlabel('altitude');
ylabel('time');
hold off
shg

hold on
figure(1)
title('vel vs time')
plot(v_array, tarray);
xlabel('vel');
ylabel('time');
hold off
shg

function zdot = myrhs(t,z,p);
Cd = p.Cd; %Unpack parameters
g = p.g;
gamma = p.gamma;
A = p.A;
propWeight = p.propWeight;
lvWeight = p.lvWeight;

y = z(1); % unpack state variables
v = z(2);

T = linerinterp(filename, t); %FILENAME

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
totImp = Trapz(T(:,1), T(:,2)); %calculates total impulse, integral of
                                %force w.r.t. time
m = zeros(length(T(:,1))); %mass

for i = 1:length(T(:,1))
    imp = Trapz((T(i,1), T(i,2));%goes through all times and finds impulses
    fractionBurned = imp / totImp;%ratio of impulses for fraction burned
    propBurned = propWeight * fractionBurned;%amount of propellent burned at
        time
    m(i) = lvWeight - propBurned; %gives us mass at time
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Mach = v / 340.29;
```

```
rho = calc_rho(Mach, gamma, y);
D = Cd * rho * v^2 * A /2; %DRAG

Ftot = T - mg - D;

ydot = v; % first two of 4 ODES
vdot = Ftot/m; % second two of 4 ODEs
zdot = [ydot;vdot]; %places rdot and vdot in zdot
end

function rho = calc_rho(mach,gamma,altitude)
rho_t = linearinterp('Standard Table.xlsx',altitude); % use standard table
    table to get value
rho = (rho_t)*(1 + ((gamma-1)/2)*(mach^2))^(-1/(gamma-1));
end

end
```

%%%%%%% linearinterp.m %%%%%%%

```
function output = linearinterp(filename, t)
T = readtable(filename);
T = table2array(T);

x = T(:,1)'; %time
y = T(:,2)'; %thrust
output = zeros(length(x), 3);
output(:, 1) = x;
output(:, 2) = y;
m = zeros(length(x)-1);


for i=1:length(x)-1 %calculates all slopes btwn points, populates m
    %(y-y1)=m(x-x1)
    m(i) = (y(i+1)-y(i)) / (x(i+1) - x(i));
end


for i=1:length(x)-1
    if t < x(i)
        output(:, 3) = (t-x(i-1))*m(i-1) + y(i-1);
    end

end
end
```

1.1.4 MATLAB Model: Results and Analysis

After running the MATLAB 1-D dynamics model, it became clear that there were issues as faulty data was produced. An additional analysis of the code was performed to determine the cause of the bug. It was discovered that the model was not running time-dependant. Therefore, values that are crucial for determining the altitude, namely thrust and mass, were not being updated properly as time progressed throughout the launch vehicle's simulated flight. Without this time dependency, these values were instead static and did not accurately represent changing conditions throughout flight.

In order to combat this bug, and still create a 1-D dynamics model, a Simulink model was made that operated identically to the envisioned MATLAB one.

1.1.5 Simulink Model

To combat the MATLAB model's inability to handle time dependency, the code was transitioned into a Simulink environment. Using Simulink allowed for a clock-block to increment time and update values such as thrust at any time. Thus, the Simulink model allows for time dependency.

The Simulink model is shown in the following figures:

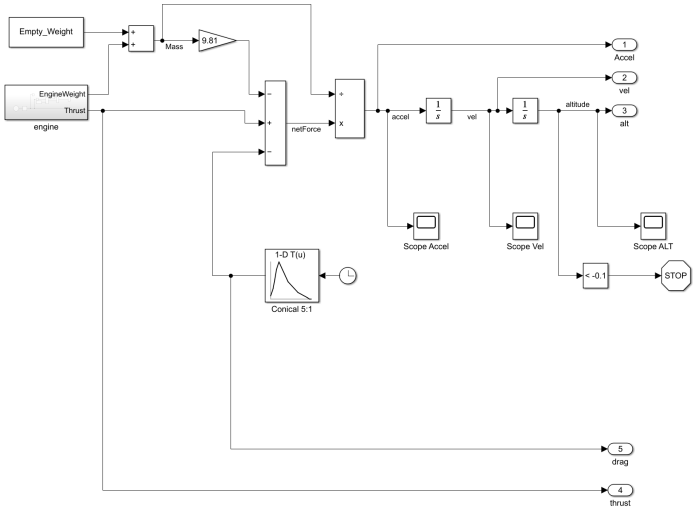


Figure 1: **Full Simulink 1-D Model**

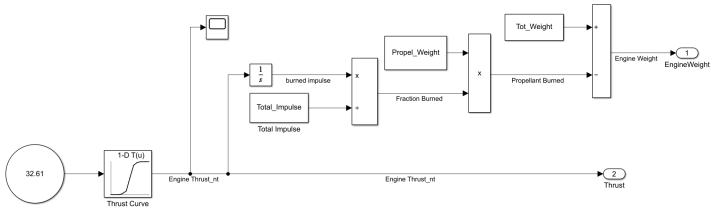


Figure 2: **Engine Weight Changing With Thrust**

Figure 1 shows a representation of the entire Simulink model. The model uses the engine subsystem – seen in Figure 2 – to determine the mass of the engine with respect to time. This is multiplied by the gravitational force to get the force applied due to gravity. Thrust with respect to time is also calculated from the engine subsystem. At this point, all forces are added to establish the net force. From the force, the model determines acceleration, velocity, and then altitude with respect to time by integration.

Figure 2, the engine subsystem, determines thrust and engine weight with respect to time. The model does this by taking in a given thrust curve, determining impulse through integration, and then finding amount of propellant burned.

1.1.6 Simulink Model Code

In order for the Simulink model to function properly, it is fed the following script to utilize drag forces calculated in a previous member's ANSYS project. These values, along with initial launch conditions, are used in the model.

```
T = readtable(Thrust. xlsx );
T = table2array(T);

x = T(:,1) ; %time
y = T(:,2) ; %thrust

output = zeros(length(x), 3);
output(:, 1) = x;
output(:, 2) = y;

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

%D = readtable(conical 5_1. xlsx ); %retrieves drag curves for each
%D = readtable(conical 4_1. xlsx ); %nose cone
%D = readtable(ogive 5_1. xlsx ); %uncomment one at a time to get
%D = readtable(ogive 4_1. xlsx ); %each nose cone
%D = readtable(ogive 3_1. xlsx );
D = readtable(von karman 55_1.xlsx);
D = table2array(D);
```

```
x = D(:,1) ; %time
y = D(:,2) ; %drag
output = zeros(length(x), 3);
output(:, 1) = x;
output(:, 2) = y;

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

Thrust_Time = T(:,1) %time values for thrust curve
ThrustNt = T(:,2); %force values for thrust curve
Drag_Time = D(:,1); %time values for drag curve
DragNt = D(:,2); %force values for drag curve
Total_Impulse = trapz(T(:,1), T(:,2)); %total impulse from thrust curve
Propel_Weight = 20.00342; %total propellant weight
Tot_Weight = 57.447; %total weight of launch vehicle
Empty_Weight = 37.445163287421030; %weight of launch vehicle without
%propellant
```

1.1.7 Simulink Model: Results and Analysis

The Simulink model was run with each nose-cone design's drag profile and produced the following altitude vs. time graph(s):

Figure 3: **Launch Vehicle Altitude vs. Time**

Figure 4: **Zoomed-in Launch Vehicle Altitude vs. Time**

The data collected shows maximum altitude – apogee – of a simulated launch of the launch vehicle with each nose cone's drag profile:

Nose Cone	Apogee (m)
Conical 4:1	1195
Ogive 5:1	1166
Von Karman 5.5:1	1162
Ogive 3:1	1161
Ogive 4:1	1159
Conical 5:1	1157

Table 1: Nose Cone vs. Apogee

As seen in Table 1, the Conical 4:1 nose cone design yielded the highest apogee, 1195 m. On the contrary, the Conical 5:1 nose cone design yielded the lowest simulated apogee, 1157 m. Key to note is the range of apogees determined by the model: 1157-1195 m. The launch vehicle is designed with a 'goal' of an apogee near 10,000 feet – roughly 3048 m. Clearly, the model yields a significantly lower result. It was determined that these results are due to incorrect drag profiles of the nose cone designs – a flaw from the previous ANSYS project whose results are utilized in this Simulink model. Hence, the model's results are flawed. In future iterations of this model, more accurate drag profiles will be needed to produce a more accurate estimation of trajectory and apogee.