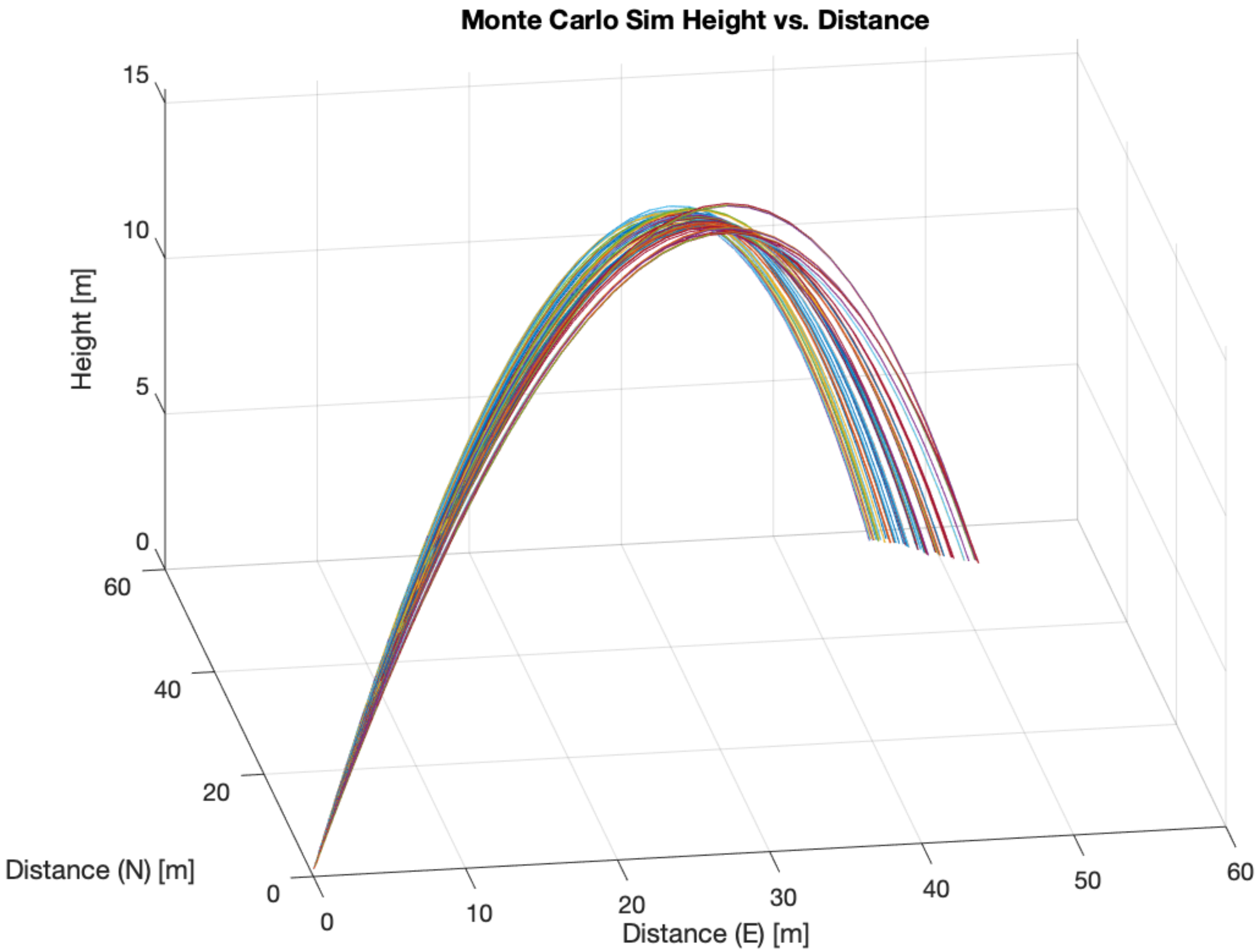

Monte Carlo Sim Height vs. Distance
Height [m]
15
10
5
0
Distance (N) [m]
60
40
20
0
Distance (E) [m]
0
10
20
30
40
50
60

Developing a Thermodynamic Model for a Bottle Rocket with Varying Parameters and Uncertainties

Travis Choy*
The University of Colorado Boulder, Boulder, CO 80309

The purpose of this paper is to develop two bottle rocket launch models derived from aerodynamic and thermodynamic principles. The first model will be 2-dimensional and several parameters will be changed in order to analyze and discuss their implications on the rocket's trajectory and thrust. From there, it will be altered to become a 3-dimensional model in order to account for external forces acting on the rocket. Lastly, a Monte Carlo simulation will be done to reflect any uncertainties within the initial data. For each model, there will be verification data from past launches to ensure accuracy within the models. Instead of finding a specific solution for the optimized rocket, this paper serves as a discussion on the process and methods used to create the model.

Nomenclature

$\vec{a}$	=	Rocket Acceleration
A_B	=	Cross-Sectional Area of the Front of the Bottle
A_t	=	Throat Area
c_d	=	Discharge Coefficient (<1)
C_D	=	Drag Coefficient (0.3 to 0.5)
g	=	Specific Heat Ratio (1.4)
$\vec{g}$	=	Gravity Vector ($g_z = 9.8\ m/s^2$)
$\vec{F}$	=	Rocket Thrust
m_{air}	=	Mass of Air in the Rocket
m_B	=	Mass of the Empty Bottle
m_r	=	Mass of the Rocket
M_e	=	Exit Mach Number
p_{air}	=	Air Pressure in the Rocket
p_a	=	Ambient Pressure
p_e	=	Exit Pressure
p_*	=	Critical Pressure
q	=	Dynamic Pressure
R	=	Ideal Gas Law Constant (287 $Jkg^{-1}K^{-1}$)
T_{air}	=	Temperature of Air in the Rocket
v_{air}	=	Volume of Air in the Rocket
v_B	=	Volume of Bottle
V_e	=	Exhaust Velocity
$\vec{V}$	=	Rocket Velocity
ρ_{air}	=	Density of Air in the Rocket
ρ_w	=	Density of Water in the Rocket

*Aerospace Engineering Undergraduate, 2763 Nipoma St, San Diego, CA 92106, AIAA Student Officer at CU Chapter

I. Introduction

Numerical, computational, or mathematical modeling is a common engineering tool that helps to understand or predict the behavior of a physical system, like a bottle rocket. Engineers can use the results of these numerical simulations to determine how to best design the system. The goal of this project is to practice using these tools to model the trajectory of the bottle rocket launch, using numerical integration of a system of ordinary differential equations.

A bottle rocket is a simple rocket consisting of a plastic bottle filled partially with a liquid and pressurized by air. When the launch begins, the stopper is removed allowing the water to be pushed out by the pressurized air creating a reactionary force that propels the bottle forward, according to Newton's laws of motion.

The goal is to develop a MATLAB code to determine the thrust as a function of time of this bottle rocket, and predict the resulting height and range of the rocket using the thermodynamics of water and air expansion. Then, this process will be repeated through numerical simulation to understand the functional dependence of bottle rocket performance on the design parameters. To better understand and explore the parameter space, a combination of variables will be changed to find what will allow the rocket to land within say, 1 meter of a 80 meter marker. From here, another model will be developed adding a 3rd dimension to the simulation in order to account for external factors in any direction.

Throughout the process of developing these models, they will be validated against actual launch data from a baseline rocket case. After developing a functional model, we will use it predict the performance of an optimized rocket, given parameter values previously determined. Lastly, the final model will undergo a Monte Carlo simulation with varying parameter values reflecting their respective uncertainties. This final model will be plotted and discussed.

II. Phases of Launch

A bottle rocket is made up two forms of propulsion: the fluid expelled by the pressurized air trapped inside, along with the air itself which drives the rocket forward as the pressure balances with the that of the atmosphere. This creates three different phases throughout the launch of the rocket. The first is the water expulsion phase, followed by the gas expulsion phase, and finishing with the ballistic phase. Each of these have their respective conditions and equations which are explained in the *ASEN 2012 - Bottle Rocket Design 2020 - Equations of Motion* document, but are also summarized in the following subsections.

A. Water Expulsion Phase

In the first phase of the launch, the water is being released propelling the rocket forward. The air is still trapped inside meaning the mass of the air, m_{air}, stays constant while the volume of the air, v_{air} increases, therefore decreasing air density, ρ_{air}. As we assume isentropic air expansion, an adiabatic process, and no friction loss, we arrive at the equation to approximate air pressure, p:

$$\frac{p}{p^i_{air}} = \left(\frac{v^i_{air}}{v}\right)^g \tag{1}$$

Along with the water's mass flow rate and rocket's thrust (F):

$$\dot{m} = c_d \rho_w A_t V_e \tag{2}$$

$$F = \dot{m} V_e + (p_e - p_a) A_t \tag{3}$$

From here, because the water is incompressible we can apply the Bernoulli equation for incompressible flows and derive the equation for exhaust velocity.

$$V_e = \sqrt{\frac{2(p - p_a)}{\rho_w}} \tag{4}$$

In this phase, the exit air pressure is equal to the ambient air pressure, greatly simplifying Eq. 3. Now, by plugging in Eq. 2 and Eq. 4, we arrive at the following equation for thrust:

$$F = \dot{m} V_e = 2 c_d A_t (p - p_a) \tag{5}$$

Also, by utilizing Eq. 1 and Eq. 4, we can find the rate of change of air volume.

$$\frac{dv}{dt} = c_d A_t V_e = c_d A_t \sqrt{\frac{2\left(p - p_a\right)}{\rho_w}} = c_d A_t \sqrt{\frac{2}{\rho_w}\left(p^i_{air}\left(\frac{v^i_{air}}{v}\right)^g - p_a\right)} \tag{6}$$

This needs to be solved using a 4th order Runge-Kutta or, as we will discuss later, the ODE45 function in MATLAB. Once the volume of the air in the bottle is equal to the volume of the bottle itself, we know we no longer need to iterate this function. Using the v just found, we can also solve for p using Eq. 1 once again.

Now, because we know the water mass is leaving the rocket, we can use Eq. 4 to set up the mass of the rocket as a function of discharge coefficient, throat area, water density, air pressure, and ambient air pressure; all values we've found at this point.

$$\dot{m}_r = -\dot{m} = -c_d \rho_w A_t V_e = -c_d A_t \sqrt{2\rho_w\left(p - p_a\right)} \tag{7}$$

Lastly, we can create an equation for the initial mass of the rocket knowing it is equal to the mass of the bottle plus the mass of the water and the mass of the air. The mass of the water can be found using its relationship with density and volume while the mass of the air can be found with the Ideal Gas Law yielding the following equation:

$$m^i_r = m_B + \rho_w\left(v_B - v^i_{air}\right) + \frac{p^i_{air} v^i_{air}}{R T^i_{air}} \tag{8}$$

B. Gas Expulsion Phase

Before continuing into the next phase, we must first establish the final air pressure and temperature of the previous one.

$$p_{end} = p^i_{air}\left(\frac{v^i_{air}}{v_B}\right)^g ; T_{end} = T^i_{air}\left(\frac{v^i_{air}}{v_B}\right)^{g-1} \tag{9}$$

Similar the idea of density changing in the water expulsion phase, in the gas expulsion phase air volume remains constant while its mass decreases, therefore decreasing the density. Now, because there is isentropic air expansion until air pressure drops to ambient air pressure, we can solve for the pressure at any time.

$$\frac{p}{p_{end}} = \left(\frac{m_{air}}{m^i_{air}}\right)^g \tag{10}$$

The density and temperature can also be found given by:

$$\rho = \frac{m_{air}}{v_B} ; T = \frac{p}{\rho R} \tag{11}$$

Using these values, we now solve for critical pressure, p_*.

$$p_* = p\left(\frac{2}{g+1}\right)^{\frac{g}{g-1}} \tag{12}$$

From here, depending on whether or not the flow is choked determines which set of equations to use. If the flow is choked, meaning $p_* > p_a$, the following formulas are used:

$$M_e = 1 \tag{13}$$

$$V_e = \sqrt{g R T_e} \tag{14}$$

Where:

$$T_e = \left(\frac{2}{g+1}\right) T; \rho_e = \frac{p_e}{RT_e}; p_e = p_* \tag{15}$$

If the flow is not choked, meaning $p_* < p_a$, the following formulas are used:

$$\frac{p}{p_a} = \left(1 + \frac{g-1}{2} M_e^2\right)^{\frac{g}{g-1}} \tag{16}$$

$$\frac{T}{T_e} = \left(1 + \frac{g-1}{2} M_e^2\right); \rho_e = \frac{p_a}{RT_e}; p_e = p_a \tag{17}$$

$$V_e = M_e \sqrt{gRT_e} \tag{18}$$

From here, thrust is found for both cases by,

$$F = \dot{m_{air}} V_e + (p_a - p_e) A_t \tag{19}$$

Where

$$\dot{m_{air}} = c_d \rho_e A_t V_e \tag{20}$$

So, similar to Eq. 7, the mass of the rocket is given by:

$$\dot{m_R} = -\dot{m_{air}} = -c_d \rho_e A_t V_e \tag{21}$$

C. Ballistic Phase

As mentioned earlier, thrust is generated by two forces: the water and the pressurized air. By the end of the water expulsion phase, no more thrust is generated by water and by the end of the gas expulsion phase, the pressure in the bottle is equal to the ambient pressure so no more thrust is generated by it either. Thrust at this point is equal to zero and the only effects acting on the rocket is gravity.

$$F = 0; m_R \sim m_B \tag{22}$$

III. 2-D Propulsion Model

In order to take a look at the effects of varying certain parameters, we will begin by developing a 2-dimensional model in the horizontal (x) and vertical (z) directions using the equations discussed.

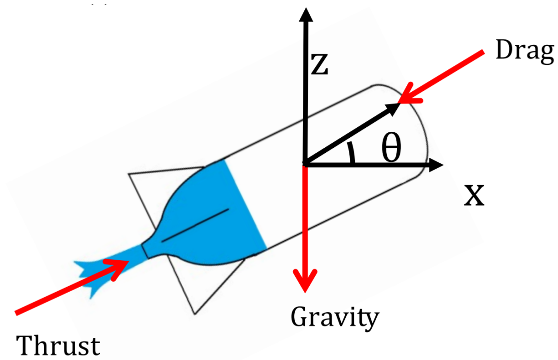


Fig. 1 Free Body Diagram of Bottle Rocket[†]

By examining the FBD in Fig. 1 above and applying Newton's laws of motion, the following equation can be derived for the sum of forces acting on the rocket:

$$\Sigma Forces = m_r\vec{a} = m_r \begin{bmatrix} a_x \\ a_z \end{bmatrix} = m_r\vec{\dot{V}} = \vec{F} - \vec{D} + m_r\vec{g} \tag{23}$$

With a drag force equal to the product of the dynamic pressure, the drag coefficient, and the cross-sectional area of the front of the bottle.

$$D = qC_D A_B = \frac{1}{2}\rho V^2 C_D A_B \tag{24}$$

A. Model Outline

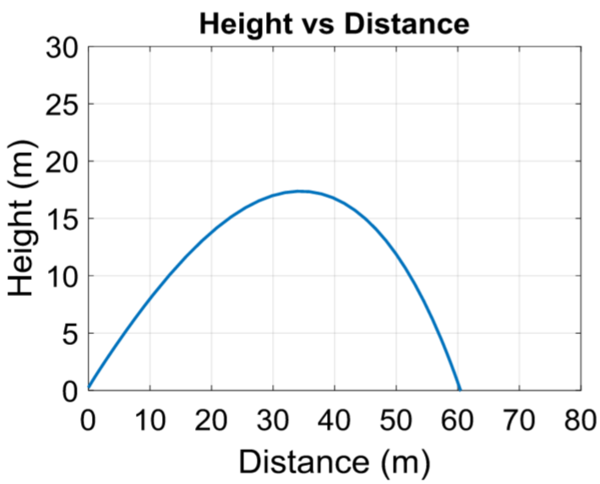
Now, we have everything we need to begin programming the model in MATLAB. For the full MATLAB Script of the 2-D Model, see Appendix A. However, it is outlined below:

- Establish constants
- Create initial state vector
- Create state function
  - Load in constants
  - Load in state vector (for the first iteration, this will be the initial state vector established earlier)
  - Convert heading vector in terms of theta (angle of flight) and overall velocity
  - Determine which phase of flight the rocket is in
    - Phase 1: Volume of Air < Volume of Bottle
    - Phase 2: Volume of Air = Volume of Bottle and Air Pressure in Bottle > Ambient Pressure
      - Determine if flow is choked
    - Phase 3: Volume of Air = Volume of Bottle and Air Pressure in Bottle = Ambient Pressure
  - Find final state values
    - Determine magnitude of drag force
    - Sum of forces in the x-direction and z-direction
    - Find accelerations in the x-direction and z-direction from their respective forces
  - Confirm rocket is above ground (if not, set all state vector values to zero)
  - Establish new state vector
- Run function through ODE45
- Plot resulting trajectory and thrust

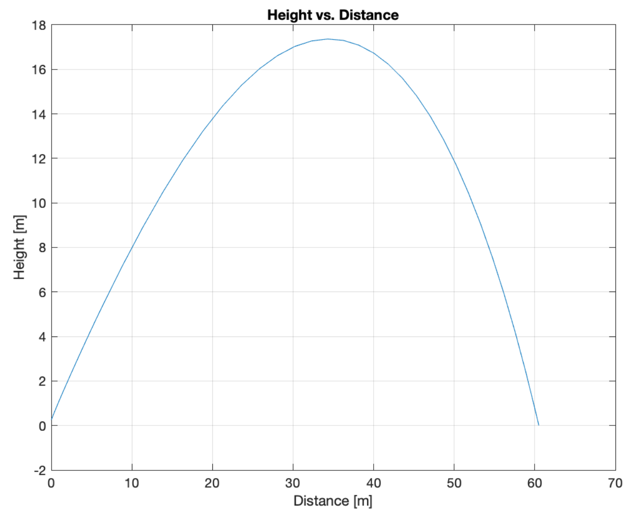
[†] Image from *ASEN 2012 - Bottle Rocket Design 2020 - Equations of Motion* lab document

B. Model Results

Once the script was completed, the resulting trajectory plot was compared to that of the provided verification case.



Verification Case Trajectory



Model Trajectory

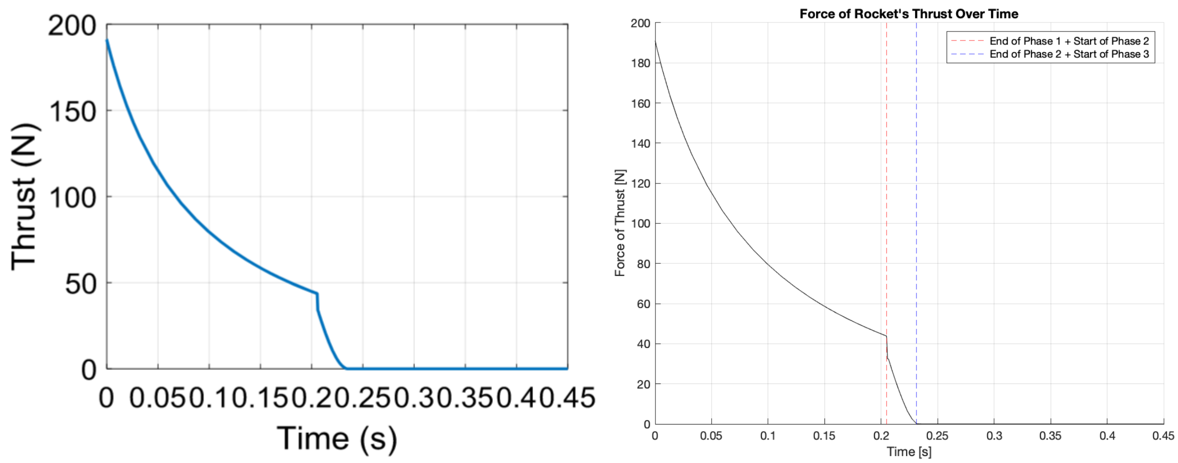
Fig. 2 Verification and Model Trajectory Comparison

Consideration	Verification Case	Model
Max Height [m]	17.37	17.36
Max Distance [m]	60.45	60.50

Table 1 Trajectory Comparison

After comparing the model with the verification case, it's apparent that the developed model is quite accurate. The maximum trajectory height is only off by 0.01 m, or 1 cm! Likewise, maximum trajectory distance is only 0.05 m, or 5 cm, greater than the verification case.

However, despite the incredible precision of the model's trajectory, it is also important to compare the thrust plots of the two cases to ensure model accuracy:



(a) Verification Case Thrust **(b) Model Thrust**

Fig. 3 Verification and Model Thrust Comparison

Phase Change	Verification Case	Model
Phase 1 $\longrightarrow$ Phase 2 [s]	0.205	0.205
Phase 2 $\longrightarrow$ Phase 3 [s]	0.230	0.230

Table 2 Thrust Comparison

Although the values in Table 2 are estimated based on visual inspection of the graphs in Figure 3, it seems that the phase change between 1 (Water Expulsion) and 2 (Gas Expulsion), as well as 2 (Gas Expulsion) and 3 (Ballistic), occur at similar times. Obviously there is uncertainty due to a simple visual inspection, however, the models seem similar enough to verify the accuracy of the model.

As mentioned, the bottle rocket flight consists of three distinct phases:

1) From the moment the stopper is removed until the water is exhausted
2) After the water is exhausted until the air pressure drops to the ambient value and the thrust phase ends
3) Ballistic phase

In the figure, we can see how the first two generate all of the thrust for the flight, however it is only a small fraction of the total flight time.

IV. Analyzing Parameter Changes

After creating a model that accurately simulates that of the verification case, four parameters need to be changed in order to reach a new target distance, 80 m for example. Each parameter affects the rocket model in different ways. In this model, four possible flight parameters will be changed and their effects on the rocket will be analyzed. The four changeable flight parameters are as followed:

- Drag Coefficient
- Volume of Water
- Air Pressure
- Launch Angle

To analyze the effects of changes these parameters, each one will varied for the original model (originally mimicking the verification case).

A. Drag Coefficient

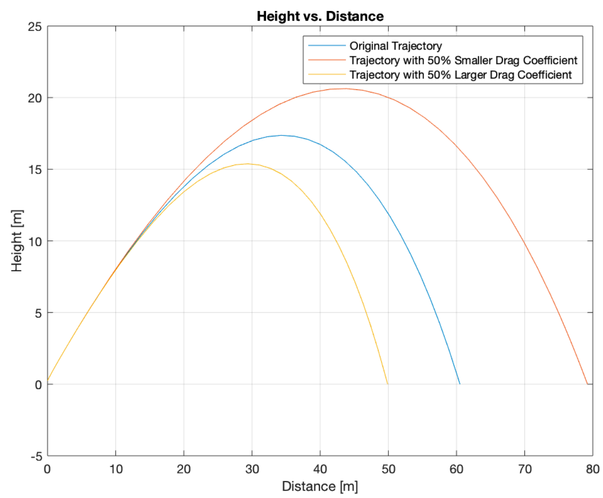


Fig. 4 Rocket Trajectory with Varying Drag Coefficient

The drag coefficient is essentially the friction caused by the air, so a smaller drag coefficient results in less "air friction," and a larger distance traveled. Similarly, a larger drag coefficient results in more "air friction," and a smaller distance traveled. Both of these effects can be seen in Figure 4. Unlike other parameters, there are not any drawbacks caused by a smaller drag coefficient making this an efficient parameter to change to optimize rocket performance.

B. Water Volume

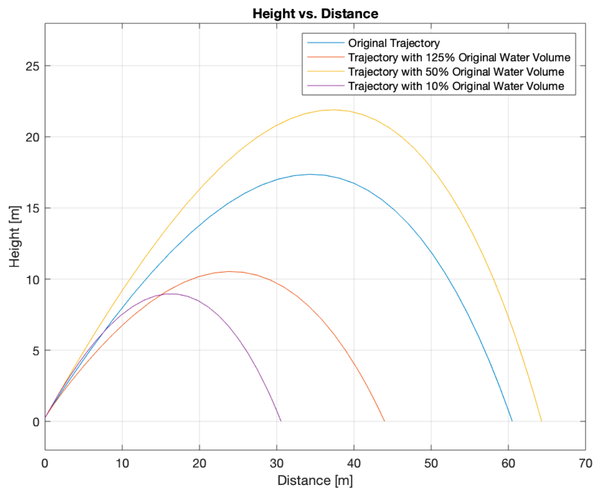


Fig. 5 Rocket Trajectory with Varying Water Volume

The volume of water contributes to propulsion during Phase 1 (Water Expulsion) and overall mass of the rocket. Increasing or decreasing water volume will not directly increase or decrease distance traveled due to drawbacks between weight and propulsion. Figure 5 demonstrates how a decrease in water volume will increase distance, but too large of a water volume decrease will then decrease distance.

C. Air Pressure

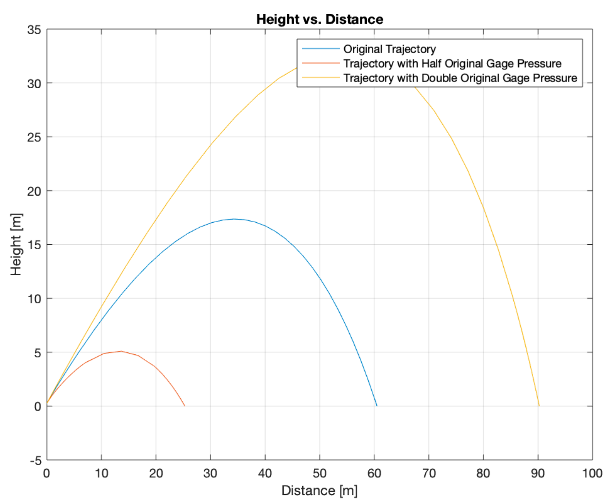


Fig. 6 Rocket Trajectory with Varying Air Pressure

As previously discussed, air pressure creates thrust during Phase 2, the Gas Expulsion Phase. Figure 7 demonstrates that increasing air pressure will increase propulsion potential of rocket, without any drawbacks. This means a decrease in air pressure results in less propulsion, and a smaller distance traveled while an increase in air pressure results in more propulsion, and a larger distance traveled.

D. Launch Angle

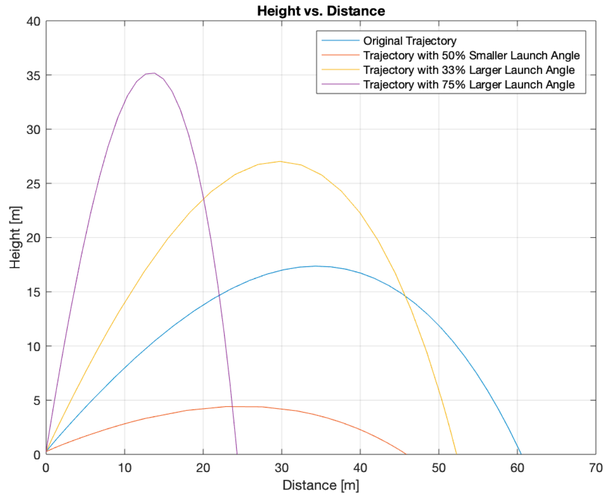


Fig. 7 Rocket Trajectory with Varying Launch Angle

After varying launch angles, it is apparent that 45° is actually the ideal launch angle. Launch angles smaller than 45° result in the rocket hitting the ground before utilizing all propulsion potential. On the other hand, launch angles larger than 45° result in the rocket using too much propulsion potential moving in the z-direction, and not enough in the x-direction. Ultimately, the optimized rocket's parameters will have a launch angle of 45° to ensure maximum downrange distance traveled.

E. Combining Parameter Changes

Hypothetically, the rocket can travel a specific distance with precision by combining multiple parameter changes. In this example, the goal is to travel 80 +/- 0.1 meters. There are an infinite number of combinations to do so, however the mix found in Figure 8 below yielding a distance of 80.05 meters is:

- A decrease in drag coefficient from 0.50 to 0.45 (increasing downrange distance)
- A decrease in water volume from 1000 cm^3 to 253 cm^3 (decreasing downrange distance)
- An increase in air pressure from 50 psi to 95 psi (increasing downrange distance)
- A decrease in launch angle from 45° to 30° (decreasing downrange distance)

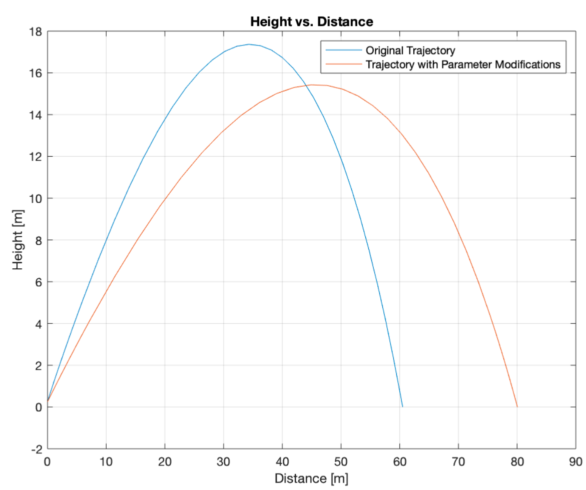


Fig. 8 Original Trajectory vs Trajectory with Parameter Modifications

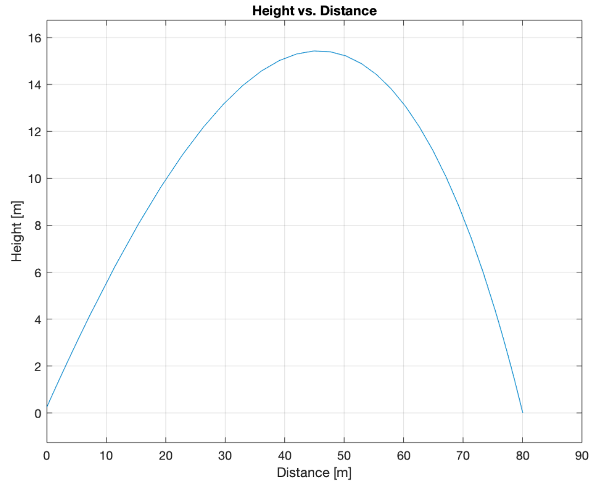


Fig. 9 Modified Trajectory

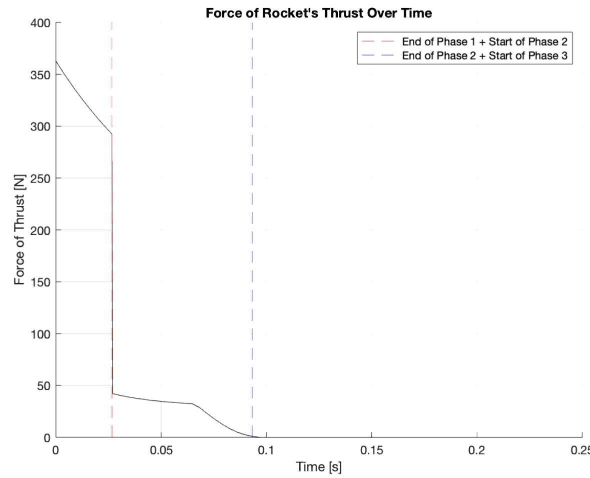


Fig. 10 Modified Thrust

Several factors affect the overall distance change and precision of the rocket. The first parameter discussed, drag coefficient, was decreased in this example, making the rocket more efficient during flight as there is less opposition. However, some parameters contributed to a decrease in downrange distance. For example, a smaller launch angle prevents the rocket from completing its full flight. Additionally, less water volume results in less fluid to be expelled. Probably most interesting though is the concept that a higher air pressure results in greater thrust increasing potential for maximizing distance but also results in exhaustion of water faster.

To better understand this, by looking at the modified thrust plot, Figure 10, it's apparent that it is quite different from the original in Figure 3b.

During Phase 1, the initial thrust is much higher than that of the verification case. This is due to the greater air pressure expelling the water from the rocket at a faster rate, increasing $\dot{m}$ and therefore F. However, it also seems that Phase 1 ends sooner than the original. This is due to a combination of higher air pressure causing water to release at a higher rate, as well as less initial water in the rocket meaning it will deplete quicker.

Now, taking a look at Phase 2, it seems that this one actually lasts longer. This is due to the higher air pressure which requires more time to balance with the surrounding air pressure.

Lastly, Phase 3 remains the same (0 N) for both cases because there are no other forces acting on the system other than gravity.

It's apparent that each parameter affects the downrange difference and overall performance of the rocket in different ways. In order to land at the desire distance, parameters differ to increase distance traveled (to avoid coming up short) and decrease distance traveled (to avoid exceeding target range).

Changes[‡] for *minimizing* distance:

- Decrease air pressure
- Inc/Dec water volume (significantly)
- Increase drag coefficient
- Changing launch angle from 45°

Changes[‡] for *maximizing* distance:

- Increase air pressure
- Decrease water volume (marginally)
- Decrease drag coefficient
- 45° launch angle

A balance between parameters using their respective trade-offs allow to narrow distance traveled to 80 meters, or whatever specified downrange distance for that matter.

[‡]In terms of changing original verification case parameters

V. Modified 3-D Model

Now that there is an established understanding on the effects of varying modeling parameters, as well as an accurate 2-dimension rocket model, it's time to alter the model into one that is 3-dimension. This allows us to account for external forces, such as wind, and incorporate more aspects of uncertainty to make the model more realistic.

The main alteration that needs to be done is changing the heading vector to incorporate a third component, the y-direction. To do so, it would be the easiest to understand by changing the heading from theta and a magnitude to three components in an $\hat{\imath}$, $\hat{\jmath}$, $\hat{k}$ heading vector in the East, North, and up directions respectively. This also allows us to easily account for external forces, $\vec{v_w}$ since we can simple subtract their respective magnitudes from the current $\vec{v_g}$ to give us $\vec{v_{rel}}$.

$$\vec{v_{rel}} = \begin{bmatrix} v_{gx} - v_{wx} \\ v_{gy} - v_{wz} \\ v_{gz} - v_{wz} \end{bmatrix} \tag{25}$$

From here, we can find the heading vector by:

$$\vec{h} = \frac{\vec{v_{rel}}}{\left\| \vec{v_{rel}} \right\|} \tag{26}$$

Next, it's quite simple to adjust the MATLAB script to account for the two extra values in the state vector, v_y and a_y. All that needs to be done is add the third direction wherever the x-direction and z-directions are used. For the most part, the main state function and ODE45 call remain the same. However, the updated script can be found in Appendix B.

In this example, we're examining a 3 mph wind pointing on a 45° heading from the North. The only thing that needs to be done to account for this is the code is adding it to the velocity vector in the respective direction(s) at the beginning of the state function. The force caused by the wind creates differences in velocities in all three direction. The resulting flight path (orange) can be compared to the original aimed path from the 2-dimensional model (blue) in Figure 11.

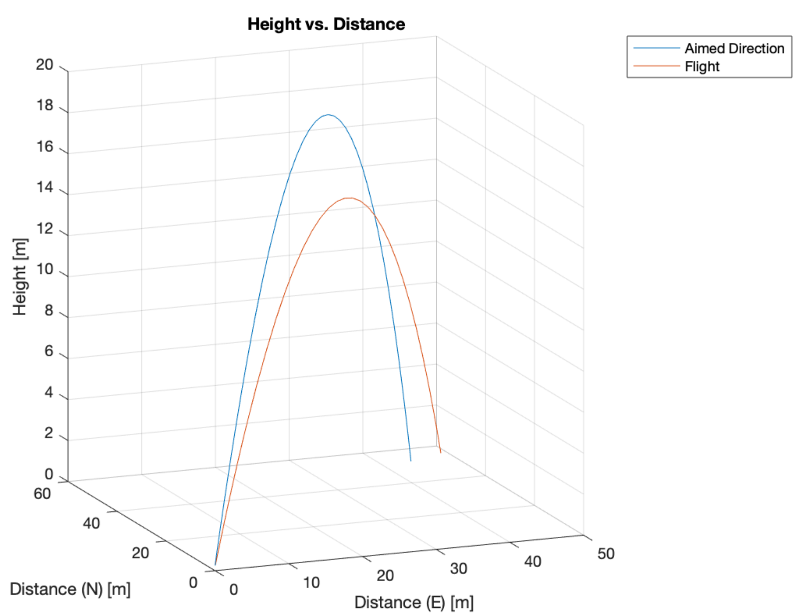


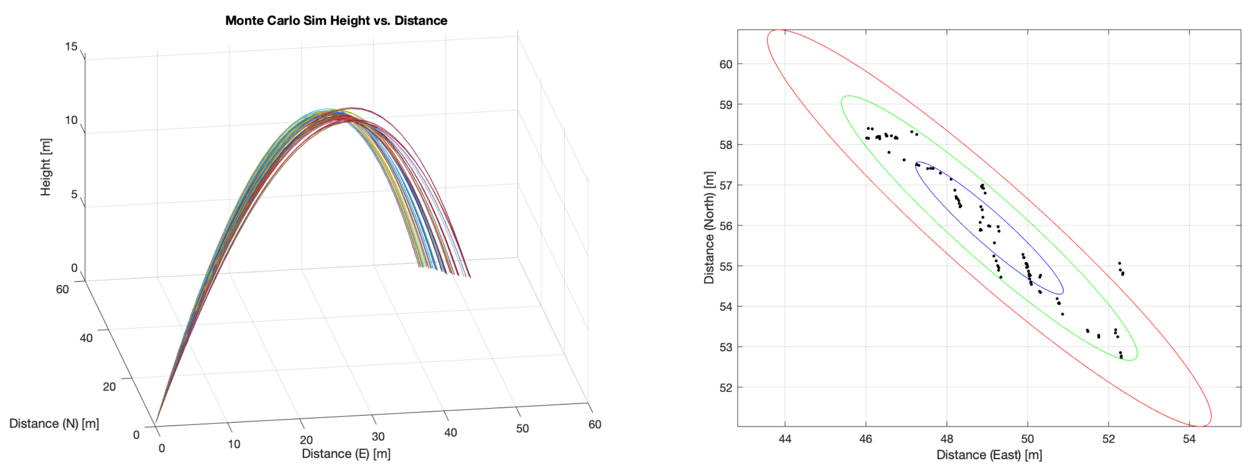
Fig. 11 Original Trajectory with No External Forces vs Actual Expected Flight Path

As you can see, the North-East wind will carry the rocket toward the right of the original path, with less height but a further downrange distance.

VI. Monte Carlo Simulation

The last factor that needs to be taken into account is uncertainty within the parameters. From the direction of the wind heading vector to uncertainty in mass measurements to fluctuations in air temperature, most values will have some sort of uncertainty. To understand how these uncertainties affect the trajectory of the rocket, a Monte Carlo simulation was done.

First, the original parameters were changed to the ideal expected values of an optimized bottle rocket launch (0.30 drag coefficient, 8 mph wind, etc.). From here, the uncertainties were taken into account through random number generation based on their severity. For example, wind had a heading of 45 +/- 11.25° so a random angle between 33.75° and 56.25° was generated and ran through the constants or initial state vector of the following function. For the model here, wind uncertainty (45 +/- 11.25°) and initial water mass uncertainty (0.600 +/- 0.0005 kg) were accounted for. These random values were generated and flight trajectories were plotted through a loop of 100 iterations. All 100 of these trajectories are shown below in Figure. 12a.



(a) Trajectory of 100 Launches

(b) Landing Locations with Error Ellipses

Fig. 12 Resulting Plots from the Monte Carlo Simulation

As you can see, the 100 Monte Carlo iterations have the trajectories modeled simulating the possible combinations of uncertainties in both wind direction and mass of water in the rocket.

The landing locations of each launch were also plotted from a vertical perspective (looking in the $-\hat{k}$ direction) in Figure 12b. The downrange distance is predicted by taking the mean of all of these landing points, in this case 74.39m. Their error ellipses are also plotted to give a more general prediction of the landing zones.

VII. Conclusion and Recommendations

Improvements to Be Made:

Obviously, the current model has a few apparent flaws. For starters, in the 3-dimensional model only two parameter uncertainties were accounted for, those being wind heading and initial water mass. Although incorporating more parameter uncertainties will make the landing area less precise, they will make the error ellipses more accurate and we can better predict the general area where the rocket will land. Another improvement that can be made is improving the Monte Carlo simulation itself. Additional iterations will, theoretically, help condense the error ellipses as more values results in outliers or rare occurrences having a smaller impact on the rest of the data.

Conclusion:

It's difficult to accurately model something like a bottle rocket launch due to all the factors and uncertainties involved. However, functions like MATLAB's ODE45 and computational algorithms like Monte Carlo simulations can effectively and efficiently improve your model. This modeling process has helped the user not only apply the aerodynamic and thermodynamic properties of a simple rocket launch to real application, but has also taught the user how essential programming software can be. An infinite number of simulations can be done and improvements can be made before the first real rocket launch is done. This helps minimize time and money for the engineers and the company. Overall, this was an exciting application and it was interesting to learn how to create and run the model.

VIII. Appendix A

ASEN 2012 MATLAB Script (2-Dimensional Trajectory and Parameter Analysis)

```matlab

% ASEN 2012 Project 2
% Author: Travis Choy
% ID: 109181287

% Date Created:  November 13, 2020
% Date Modified: December  1, 2020
% Due Date:      December  4, 2020

% Purpose: To utulize ODE45 to numerically integrate an ordinary
% differential equation such as that of the flight of a bottle rocket. Plot
% and analyze the flight of the rocket and the effects of varying specific
% parameters.

%% Housekeeping

clear
close all
clc

%% Constants

g = 9.81;                           % Acceleration Due to Gravity [m/s^2]
C_d = 0.8;                          % Discharge Coefficient
p_amb = 0.961;                      % Ambient Air Density [kg/m^3]
V_bottle = 0.002;                   % Volume of Empty Bottle [m^3]
P_amb = 12.1;                       % Atmospheric Pressure [psi]
P_amb = P_amb * 6894.76;            % Atmospheric Pressure [Pa]
cp_rat = 1.4;                       % Ratio of Specific Heats for Air
p_water = 1000;                     % Density of Water [kg/m^3]
D_throat = 2.1;                     % Diameter of Throat [cm]
D_throat = D_throat/100;            % Diameter of Throat [m]
A_throat = pi * (D_throat/2)^2;     % Area of Throat [m^2]
D_bottle = 10.5;                    % Diameter of Bottle [cm]
D_bottle = D_bottle/100;            % Diameter of Bottle [m]
A_bottle = pi * (D_bottle/2)^2;     % Cross-Sectional Area of Bottle [m^2]
R = 287;                            % Gas Constant of Air [J/kgK]
m_bottle = 0.15;                    % Mass of Empty 2-Liter Bottle with Cone and Fins [kg]
cd = 0.5;                           % Drag Coefficient
ls= 0.5;                            % Length of test stand [m]

%% Initial Value

P_gage_0 = 50;                                % Initial Gage Pressure of Air in Bottle [psi]
P_gage_0 = P_gage_0 * 6894.76;                % Initial Gage Pressure of Air in Bottle [Pa]
P_bottle = P_amb + P_gage_0;                  % Initial Total Pressure of Air in Bottle [Pa]
```

```
V_water_0 = 0.001;                              % Initial Volume of Water Inside Bottle [m^3]
m_water_0 = p_water * V_water_0;                % Initial Mass of Water [kg]
V_air_0 = V_bottle - V_water_0;                 % Initial Volume of Air Inside Bottle [m^3]
T_air_0 = 300;                                  % Initial Temperature of Air [K]
m_air_0 = P_bottle * V_air_0 / ( R * T_air_0 ); % Initial Mass of Air [kg]
v_0 = 0.0;                                      % Initial Velocity of Rocket [m/s]
v_x_0 = 0.0;                                    % Initial Velocity of Rocket in X-Direction [m/s]
v_z_0 = 0.0;                                    % Initial Velocity of Rocket in Z-Direction [m/s]
theta_0 = pi/4;                                 % Initial Angle of Rocket (45) [radians]
x_0 = 0.0;                                      % Initial Horizontal Distance [m]
z_0 = 0.25;                                     % Initial Vertical Height [m]
x_stand = ls * cos(theta_0) + x_0;              % Initial X Distance of Rocket on Stand [m]
z_stand = ls * sin(theta_0) + z_0;              % Initial Z Distance of Rocket on Stand [m]
R_stand = sqrt( x_stand ^ 2 + z_stand ^ 2 );    % Initial Distance of Rocket on Stand [m]

m_bottle_0 = m_water_0 + m_air_0 + m_bottle;    % Initial Mass of Rocket [kg]
                                                % Equation (11)

constants = [V_bottle, V_air_0, P_bottle, cp_rat, C_d, p_water, A_throat,
    P_amb, p_amb, cd, A_bottle, T_air_0, m_air_0, R, g, R_stand, theta_0];

initial_state_vector = [x_0; z_0; v_x_0; v_z_0; m_bottle_0; m_air_0; V_air_0];

% Considering Part 2 Hypothetical:
hypothetical = false;                           % Change to True for Hypothetical conditions

if hypothetical
    P_gage_0 = 95;                                    % Initial Gage Pressure of Air in Bottle [psi]
        P_gage_0 = P_gage_0 * 6894.76;                % Initial Gage Pressure of Air in Bottle [Pa]
        P_bottle = P_amb + P_gage_0;                  % Initial Total Pressure of Air in Bottle [Pa]
    V_water_0 = 0.000253;                             % Initial Volume of Water Inside Bottle [m^3]
    cd = 0.40;                                        % Drag Coefficient
    theta_0 = pi / 6;                                 % Initial Angle of Rocket [radians]
        m_water_0 = p_water * V_water_0;              % Initial Mass of Water [kg]
        V_air_0 = V_bottle - V_water_0;               % Initial Volume of Air Inside Bottle [m^3]
```

```
        m_air_0 = P_bottle * V_air_0 / ( R * T_air_0 ); % Initial Mass of Air [
            kg]
        x_stand = ls * cos(theta_0) + x_0;              % Initial X Distance of
             Rocket on Stand [m]
        z_stand = ls * sin(theta_0) + z_0;              % Initial Z Distance of
             Rocket on Stand [m]
        R_stand = sqrt( x_stand ^ 2 + z_stand ^ 2 );    % Initial Distance of
            Rocket on Stand [m]
        m_bottle_0 = m_water_0 + m_air_0 + m_bottle;

        constants_par = [V_bottle, V_air_0, P_bottle, cp_rat, C_d, p_water,
            A_throat, P_amb, p_amb, cd, A_bottle, T_air_0, m_air_0, R, g,
            R_stand, theta_0];
        initial_state_vector_par = [x_0; z_0; v_x_0; v_z_0; m_bottle_0; m_air_0
            ; V_air_0];

end

%% Calling ODE45

tspan = [0 5]; % [s]

[t,state_vector] = ode45(@(t,y) rocket_fun(t, y, constants), tspan,
    initial_state_vector);

if hypothetical
    [t_par,state_vector_par] = ode45(@(t,y) rocket_fun(t, y, constants_par),
        tspan, initial_state_vector_par);
end

%% Plot

% Plot Trajectory
figure(1)
plot(state_vector(:,1),state_vector(:,2))
hold on
xlabel("Distance [m]")
ylabel("Height [m]")
title("Height vs. Distance")
grid on
hold off

% Create Vector of Values for Thrust Graph
F_thrust = zeros(length(t),1);
P_air = zeros(length(t),1);
for i= 1: length(t)
    [F_thrust(i),P_air(i)] = rocket_thrust_graph(t, state_vector(i,:),
        constants);
    if state_vector(i,7) < V_bottle
        t_phase_1(i) = t(i);
    elseif state_vector(i,7) >= V_bottle && P_amb < P_air(i)
        t_phase_2(i) = t(i);
    end
end
```

```
% Plot Thrust
figure(2)
xline(t_phase_1(end),"--r");
hold on
xline(t_phase_2(end),"--b");
plot(t, F_thrust,"k")
xlim([0 0.45])
xlabel("Time [s]")
ylabel("Force of Thrust [N]")
title("Force of Rocket's Thrust Over Time")
legend("End of Phase 1 + Start of Phase 2", "End of Phase 2 + Start of Phase
    3")
grid on
hold off

% Plot Hypothetical Trajectory
if hypothetical
figure(3)
plot(state_vector(:,1),state_vector(:,2))
hold on
plot(state_vector_par(:,1),state_vector_par(:,2))
legend("Original Trajectory","Trajectory with Parameter Modifications")
xlabel("Distance [m]")
ylabel("Height [m]")
title("Height vs. Distance")
grid on
hold off

% Create Vector of Values for Thrust Graph
F_thrust_par = zeros(length(t_par),1);
P_air_par = zeros(length(t_par),1);
for i= 1: length(t_par)
    [F_thrust_par(i),P_air_par(i)] = rocket_thrust_graph(t_par,
        state_vector_par(i,:),constants_par);
    if state_vector_par(i,7) < V_bottle
        t_phase_1_par(i) = t_par(i);
    elseif state_vector_par(i,7) >= V_bottle && P_amb < P_air_par(i)
        t_phase_2_par(i) = t_par(i);
    end
end

% Plot Thrust
figure(4)
xline(t_phase_1_par(end),"--r");
hold on
xline(t_phase_2_par(end),"--b");
plot(t_par, F_thrust_par,"k")
xlim([0 0.25])
xlabel("Time [s]")
ylabel("Force of Thrust [N]")
title("Force of Rocket's Thrust Over Time")
legend("End of Phase 1 + Start of Phase 2", "End of Phase 2 + Start of Phase
    3")
```

```
grid on
hold off

end

%% Function Initial Conditions

function state_vector = rocket_fun(t, y, constants)

    % Declare Constants
    V_bottle = constants(1);
    V_air_0 = constants(2);
    P_bottle = constants(3);
    cp_rat = constants(4);
    C_d = constants(5);
    p_water = constants(6);
    A_throat = constants(7);
    P_amb = constants(8);
    p_amb = constants(9);
    cd = constants(10);
    A_bottle = constants(11);
    T_air_0 = constants(12);
    m_air_0 = constants(13);
    R = constants(14);
    g = constants(15);
    R_stand = constants(16);
    theta_0 = constants(17);

% State Values
    x = y(1);
    z = y(2);
    v_x = y(3);
    v_z = y(4);
    m_bottle = y(5);
    m_air = y(6);
    V_air = y(7);

% Theta
rocket_pos = sqrt( x ^ 2 + z ^ 2 );
if ( rocket_pos > R_stand ) % determine if rocket has left stand
    theta = atan( v_z / v_x );
else
    theta = theta_0;
end

% Velocity
v = sqrt( v_x ^ 2 + v_z ^ 2 );

%% PHASE 1: Water Expulsion

    if V_air < V_bottle

        % Air Pressure
        % Equation (3)
```

```
        P_air = P_bottle * (V_air_0 / V_air) ^ cp_rat;

        % Exhaust Velocity
        % Equation (7)
        v_exhaust = sqrt( 2 * ( P_air - P_amb ) / p_water );

        % Mass Flow Rate of Water
        % Equation (4)
        m_dot_w = C_d * p_water * A_throat * v_exhaust;

        % Mass Flow Rate of Rocket
        % Equation (10)
        m_dot_r = -m_dot_w;

        % Mass Flow Rate of Air
        m_dot_a = 0; % Air mass doens't change in this phase

        % Rate of Change of Volume of Air
        % Equation (9)
        V_dot = C_d * A_throat * v_exhaust;

        % Force of Thrust
        % Equation (8)
        F_thrust = 2 * C_d * A_throat * (P_air - P_amb);

    else % set up Pressure and Temperatures for Phase 2 and 3

        % Air Pressure and Temperature After Water Exhausted
        % Equation (13)
        P_end = P_bottle * ( V_air_0 / V_bottle ) ^ cp_rat;

        % New Pressure Equation
        % Equation (14)
        P_air = P_end * (( m_air / m_air_0)) ^ cp_rat;

    end

%% PHASE 2: Gas Expulsion

    if (V_air >= V_bottle) && (P_air > P_amb)

        % Change in Volume
        V_dot = 0;

        % Calculate Density and Temperature
        % Equation (15)
        rho = m_air / V_bottle;
        T = P_air / ( rho * R );

        % Critical Pressure
        % Equation (16)
        P_crit = P_air * ( 2 / ( cp_rat + 1 )) ^ ( cp_rat / ( cp_rat - 1));

        % If Choked Flow
```

```
        if ( P_crit > P_amb )

            % Exit Temperature
            % Equation (18)
            T_exit = ( 2 / ( cp_rat + 1 )) * T;

            % Exit Velocity
            % Equation (17)
            V_exit = sqrt( cp_rat * R * T_exit );

            % Exit Pressure
            % Equation (18)
            P_exit = P_crit;

        % If Not Choked Flow
        else

            % Exit Mach Number
            % Equation (19)
            Mach_exit = sqrt(abs((( P_air / P_amb ) ^ (( cp_rat - 1) / cp_rat)
                - 1 ) / (( cp_rat - 1 ) / 2)));

            % Exit Temperature
            % Equation (20)
            T_exit = T / ( 1 + ( ( cp_rat - 1) / 2 ) * Mach_exit ^ 2);

            % Exit Pressure
            % Equation (20)
            P_exit = P_amb;

            % Exit Velocity
            % Equation (21)
            V_exit = Mach_exit * sqrt(abs( cp_rat * R * T_exit ));

        end

        % Exit Density
        % Equation (18) and (20)
        rho_exit = P_exit / ( R * T_exit);

        % Change in Air Mass
        % Equation (23)
        m_dot_a = -C_d * rho_exit * A_throat * V_exit;

        % Force of Thrust
        % Equation (22)
        F_thrust = (-m_dot_a * V_exit) + (( P_amb - P_exit ) * A_throat);

        % Change in Rocket Mass
        % Equation (24)
        m_dot_r = m_dot_a;

    end
```

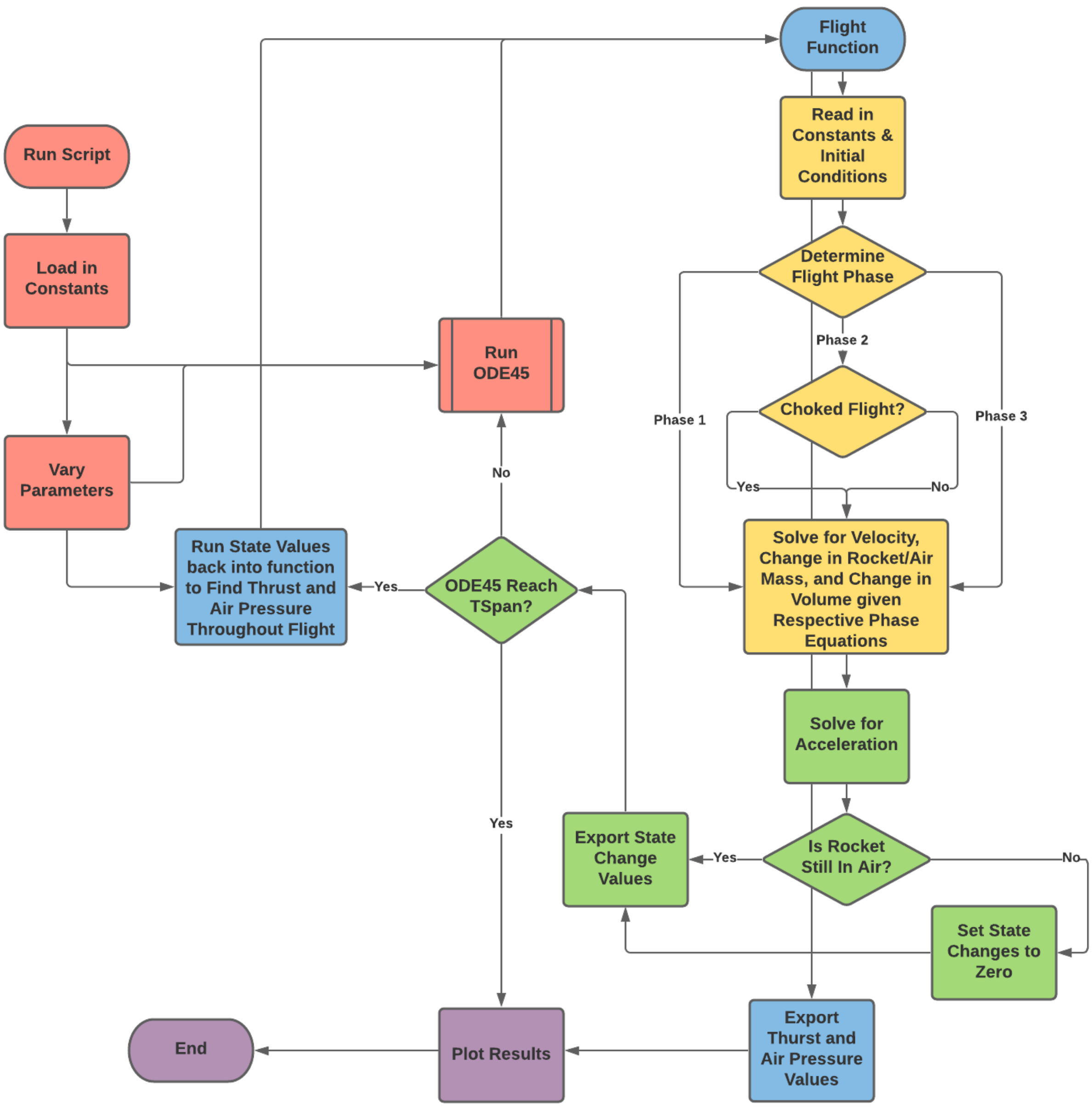
```
%% PHASE 3: Ballistic Phase

    if (V_air >= V_bottle) && (P_air <= P_amb)

        % No more water or gas so theres no change in mass, volume, or thrust
        % Equation (25)
        F_thrust = 0;
        m_dot_r = 0;
        m_dot_a = 0;
        V_dot = 0;

    end

%% Final State Function Calculations

    % Force of Drag
    % Equation (2)
    F_drag = (1/2) * p_amb * (v ^ 2) * cd * A_bottle;

    % Sum of Forces
    % Equation (1)
    sum_F_x = F_thrust * cos(theta) - F_drag * cos(theta);
    sum_F_z = F_thrust * sin(theta) - F_drag * sin(theta) - m_bottle * g;

    % Acceleration
    % Equation (1)
    acc_x = sum_F_x / m_bottle;
    acc_z = sum_F_z / m_bottle;

    % Keep Above x-axis
    if z <= 0
        v_x = 0;
        v_z = 0;
        acc_x = 0;
        acc_z = 0;
        m_dot_r = 0;
        m_dot_a = 0;
        V_dot = 0;
    end

    % Resulting State Values
    state_vector(1,1) = v_x;
    state_vector(2,1) = v_z;
    state_vector(3,1) = acc_x;
    state_vector(4,1) = acc_z;
    state_vector(5,1) = m_dot_r;
    state_vector(6,1) = m_dot_a;
    state_vector(7,1) = V_dot;


end

%% Function for Thrust Plot

```

```
function [F_thrust, P_air] = rocket_thrust_graph(t, y, constants)

    % Declare Constants
    V_bottle = constants(1);
    V_air_0 = constants(2);
    P_bottle = constants(3);
    cp_rat = constants(4);
    C_d = constants(5);
    A_throat = constants(7);
    P_amb = constants(8);
    m_air_0 = constants(13);
    R = constants(14);

% State Values
    v_x = y(3);
    v_z = y(4);
    m_air = y(6);
    V_air = y(7);

% Velocity
v = sqrt( v_x ^ 2 + v_z ^ 2 );

%% PHASE 1: Water Expulsion

    if V_air < V_bottle

        % Air Pressure
        % Equation (3)
        P_air = P_bottle * (V_air_0 / V_air) ^ cp_rat;

        % Force of Thrust
        % Equation (8)
        %F_thrust = m_dot_r * v_exhaust;
        F_thrust = 2 * C_d * A_throat * (P_air - P_amb);

    else % set up Pressure and Temperatures for Phase 2 and 3

        % Air Pressure and Temperature After Water Exhausted
        % Equation (13)
        P_end = P_bottle * ( V_air_0 / V_bottle ) ^ cp_rat;

        % New Pressure Equation
        % Equation (14)
        P_air = P_end * (( m_air / m_air_0)) ^ cp_rat;

    end

%% PHASE 2: Gas Expulsion

    if (V_air >= V_bottle) && (P_air > P_amb)

        % Calculate Density and Temperature
        % Equation (15)
        rho = m_air / V_bottle;
```

```
T = P_air / ( rho * R );

% Critical Pressure
% Equation (16)
P_crit = P_air * ( 2 / ( cp_rat + 1 )) ^ ( cp_rat / ( cp_rat - 1));

% If Choked Flow
if ( P_crit > P_amb )

    % Exit Temperature
    % Equation (18)
    T_exit = ( 2 / ( cp_rat + 1 )) * T;

    % Exit Velocity
    % Equation (17)
    V_exit = sqrt( cp_rat * R * T_exit );

    % Exit Pressure
    % Equation (18)
    P_exit = P_crit;

% If Not Choked Flow
else

    % Exit Mach Number
    % Equation (19)
    Mach_exit = sqrt(abs((( P_air / P_amb ) ^ (( cp_rat - 1) / cp_rat)
        - 1 ) / (( cp_rat - 1 ) / 2)));

    % Exit Temperature
    % Equation (20)
    T_exit = T / ( 1 + ( ( cp_rat - 1) / 2 ) * Mach_exit ^ 2);

    % Exit Pressure
    % Equation (20)
    P_exit = P_amb;

    % Exit Velocity
    % Equation (21)
    V_exit = Mach_exit * sqrt(abs( cp_rat * R * T_exit ));

end

% Exit Density
% Equation (18) and (20)
rho_exit = P_exit / ( R * T_exit);

% Change in Air Mass
% Equation (23)
m_dot_a = -C_d * rho_exit * A_throat * V_exit;

% Force of Thrust
% Equation (22)
F_thrust = (-m_dot_a * V_exit) + (( P_amb - P_exit ) * A_throat);
```

```
    end

%% PHASE 3: Ballistic Phase

    if (V_air >= V_bottle) && (P_air <= P_amb)

        % No more water or gas so theres no change in mass, volume, or thrust
        % Equation (25)
         F_thrust = 0;

    end

end
```

ASEN 2012 MATLAB Script Flow Chart



IX. Appendix B

ASEN 2004 MATLAB Script (3-Dimensional Trajectory and Monte Carlo Simulation)

```
% ASEN 2004 Lab 2: Individual Modeling
% Author: Travis Choy
% ID: 109181287

% Date Created:  November 13, 2020
% Date Modified: April    12, 2021
% Due Date:      April    13, 2021

% Purpose: To plot and analyze the flight of the rocket and the effects of
% varying specific parameters. Verify and determine the predicted distance
% and error ellipses of the optimize rocket provided.

%% Housekeeping

clear
close all
clc

%% Constants

g = 9.81;                               % Acceleration Due to Gravity [m/s^2]
C_d = 0.8;                              % Discharge Coefficient
p_amb = 0.961;                          % Ambient Air Density [kg/m^3]
V_bottle = 0.002;                       % Volume of Empty Bottle [m^3]
P_amb = 12.1;                           % Atmospheric Pressure [psi]
P_amb = P_amb * 6894.76;                % Atmospheric Pressure [Pa]
cp_rat = 1.4;                           % Ratio of Specific Heats for Air
p_water = 1000;                         % Density of Water [kg/m^3]
D_throat = 2.1;                         % Diameter of Throat [cm]
D_throat = D_throat/100;                % Diameter of Throat [m]
A_throat = pi * (D_throat/2)^2;         % Area of Throat [m^2]
D_bottle = 10.5;                        % Diameter of Bottle [cm]
D_bottle = D_bottle/100;                % Diameter of Bottle [m]
A_bottle = pi * (D_bottle/2)^2;         % Cross-Sectional Area of Bottle [m^2]
R = 287;                                % Gas Constant of Air [J/kgK]
%m_bottle = 0.128;                       % Mass of Empty 2-Liter Bottle with Cone
    and Fins [kg] (Baseline)
m_bottle = 0.160;                       % Mass of Empty 2-Liter Bottle with Cone
    and Fins [kg] (Optimized)
%cd = 0.38;                              % Drag Coefficient (Baseline)
cd = 0.30;                              % Drag Coefficient (Optimized)
ls= 0.5;                                % Length of test stand [m]
%wind = 3;                               % Wind Speed [mph] (Baselined)
wind = 8;                               % Wind Speed [mph] (Optimized)
wind = wind * 1609.34 / 3600;           % Wind Speed [m/s]
wind_angle = 45;                        % Wind Angle from North [deg]
wind_angle = 90 - wind_angle;           % Wind Angle from East [deg]
wind_y = wind * sind(wind_angle);       % Wind Speed from South Direction [m/s]
wind_x = wind * cosd(wind_angle);       % Wind Speed from West Direction [m/s]
```

```
wind_uncertainty = 11.25;           % Wind Angle Uncertainty [deg]
l_heading = 40;                     % Launch Heading is 40 deg from North, -50 from East
l_heading = 90 - l_heading;         % Launch Heading from x-axis (East) [deg]
l_heading = l_heading * (pi/180);   % Launch Heading [rad]

%% Initial Value

P_gage_0 = 40;                                  % Initial Gage Pressure of Air in Bottle [psi]
P_gage_0 = P_gage_0 * 6894.76;                  % Initial Gage Pressure of Air in Bottle [Pa]
P_bottle = P_amb + P_gage_0;                    % Initial Total Pressure of Air in Bottle [Pa]
m_water_uncertainty = 0.0005;                   % Initial Mass of Water Uncertainty [kg]
%m_water_0 = 1.001;                              % Initial Mass of Water [kg] (Baseline)
m_water_0 = 0.600;                              % Initial Mass of Water [kg] (Optimized)
V_water_0 = m_water_0 / p_water;                % Initial Volume of Water Inside Bottle [m^3]
V_air_0 = V_bottle - V_water_0;                 % Initial Volume of Air Inside Bottle [m^3]
%T_air_0 = 62;                                   % Initial Temperature of Air [F] (Baseline)
T_air_0 = 63;                                   % Initial Temperature of Air [F] (Optimized)
T_air_0 = (T_air_0 - 32) * (5 / 9) + 273.15;    % Initial Temperature of Air [K]
m_air_0 = P_bottle * V_air_0 / ( R * T_air_0 ); % Initial Mass of Air [kg]
v_0 = 0.0;                                      % Initial Velocity of Rocket [m/s]
v_x_0 = 0.0;                                    % Initial Velocity of Rocket in X-Direction [m/s]
v_y_0 = 0.0;                                    % Initial Velocity of Rocket in Y-Direction [m/s]
v_z_0 = 0.0;                                    % Initial Velocity of Rocket in Z-Direction [m/s]
theta_0 = pi/4;                                 % Initial Angle of Rocket (45) [radians]
x_0 = 0.0;                                      % Initial Horizontal (X) Distance [m]
y_0 = 0.0;                                      % Initial Horizontal (Y) Distance [m]
z_0 = 0.25;                                     % Initial Vertical Height [m]
h_stand = ls * cos(theta_0) + x_0;              % Initial Horizontal Distance of Rocket on Stand [m]
v_stand = ls * sin(theta_0) + z_0;              % Initial Vertical Distance of Rocket on Stand [m]
R_stand = sqrt( h_stand ^ 2 + v_stand ^ 2 );    % Initial Length of Rocket on Stand [m]

m_bottle_0 = m_water_0 + m_air_0 + m_bottle;    % Initial Mass of Rocket [kg]
```

```
                                                        % Equation (11)

constants_wind = [V_bottle, V_air_0, P_bottle, cp_rat, C_d, p_water, A_throat,
     P_amb, p_amb, cd, A_bottle, T_air_0, m_air_0, R, g, R_stand, theta_0,
    wind_x, wind_y, l_heading];
constants = [V_bottle, V_air_0, P_bottle, cp_rat, C_d, p_water, A_throat,
    P_amb, p_amb, cd, A_bottle, T_air_0, m_air_0, R, g, R_stand, theta_0, 0,
    0, l_heading];

initial_state_vector = [x_0; z_0; v_x_0; v_z_0; m_bottle_0; m_air_0; V_air_0;
    y_0; v_y_0];

%% Calling ODE45

tspan = [0 5]; % [s]

[t,state_vector] = ode45(@(t,y) rocket_fun(t, y, constants), tspan,
    initial_state_vector);
[t,state_vector_wind] = ode45(@(t,y) rocket_fun(t, y, constants_wind), tspan,
    initial_state_vector);

%% Plot

% Plot Trajectory
figure(1)
plot3(state_vector(:,1),state_vector(:,8),state_vector(:,2)) % Aimed Direction
hold on
plot3(state_vector_wind(:,1),state_vector_wind(:,8),state_vector_wind(:,2)) %
    Flight
xlabel("Distance (E) [m]")
%xlim([-5 60])
ylabel("Distance (N) [m]")
%ylim([-10 10])
zlabel("Height [m]")
title("Height vs. Distance")
legend("Aimed Direction", "Flight")
grid on
hold off

dist = sqrt( max(state_vector(:,1))^2 + max(state_vector(:,8))^2 );
dist_wind = sqrt( max(state_vector_wind(:,1))^2 + max(state_vector_wind(:,8))
    ^2 );

%% Monte Python Simulation

Number_of_Iterations = 100;
monte_max_values = zeros(Number_of_Iterations,3);

for i = 1:Number_of_Iterations

    % Change Wind Parameter
    wind_change = wind_uncertainty * (rand(1)*2-1); % Rand(1) finds random #
        between 0 and 1, multiply by 2 and subtract 1 to find # between -1 and
         1
```

```
    wind_angle_monte = 45 + wind_change;            % Wind Angle from North [deg]
    wind_angle_monte = 90 - wind_angle_monte;       % Wind Angle from East [deg]
    wind_y_monte = wind * sind(wind_angle_monte);   % Wind Speed from South Direction [m/s]
    wind_x_monte = wind * cosd(wind_angle_monte);   % Wind Speed from West Direction [m/s]

    % Change Mass of Water Parameter
    m_water_change = m_water_uncertainty * (rand(1)*2-1);          % Rand(1) finds random # between 0 and 1, multiply by 2 and subtract 1 to find # between -1 and 1
    m_water_0_monte = m_water_0 + m_water_change;                          % Initial Mass of Water [kg]
    V_water_0_monte = m_water_0_monte / p_water;                   % Initial Volume of Water Inside Bottle [m^3]
    V_air_0_monte = V_bottle - V_water_0_monte;                    % Initial Volume of Air Inside Bottle [m^3]
    m_air_0_monte = P_bottle * V_air_0_monte / ( R * T_air_0 );    % Initial Mass of Air [kg]
    m_bottle_0_monte = m_water_0_monte + m_air_0_monte + m_bottle; % Initial Mass of Rocket [kg]

    % Redefine Constants and Initial State Vector
    constants_wind = [V_bottle, V_air_0_monte, P_bottle, cp_rat, C_d, p_water, A_throat, P_amb, p_amb, cd, A_bottle, T_air_0, m_air_0_monte, R, g, R_stand, theta_0, wind_x_monte, wind_y_monte, l_heading];
    initial_state_vector_monte = [x_0; z_0; v_x_0; v_z_0; m_bottle_0_monte; m_air_0_monte; V_air_0_monte; y_0; v_y_0];

    % Call ODE45
    [t,state_vector_monte] = ode45(@(t,y) rocket_fun(t, y, constants_wind), tspan, initial_state_vector);

    % Assign Values to Table
    monte_max_values(i,1) = max(state_vector_monte(:,1)); % Assign Max X-Value
    monte_max_values(i,2) = max(state_vector_monte(:,8)); % Assign Max Y-Value
    monte_max_values(i,3) = max(state_vector_monte(:,2)); % Assign Max Z-Value

    % Plot Trajectory
    figure(2)
    hold on
    plot3(state_vector_monte(:,1),state_vector_monte(:,8),state_vector_monte(:,2))
    xlabel("Distance (E) [m]")
    ylabel("Distance (N) [m]")
    zlabel("Height [m]")
    title("Monte Carlo Sim Height vs. Distance")
    grid on
    hold off

end

```

```
[x_sim_mean, y_sim_mean] = error_ellipses(monte_max_values);
mean_dist = sqrt( x_sim_mean^2 + y_sim_mean^2 );

%% State Results

fprintf("The model predicts a total horizontal distance of %.2f m without wind.\n",dist)
fprintf("The model predicts a total horizontal distance of %.2f m with wind.\n",dist_wind)
fprintf("The Monte Carlo Simulation predicts an average total horizontal distance of %.2f m with wind.\n",mean_dist)

%% Function Initial Conditions

function state_vector = rocket_fun(t, y, constants)

    % Declare Constants
    V_bottle = constants(1);
    V_air_0 = constants(2);
    P_bottle = constants(3);
    cp_rat = constants(4);
    C_d = constants(5);
    p_water = constants(6);
    A_throat = constants(7);
    P_amb = constants(8);
    p_amb = constants(9);
    cd = constants(10);
    A_bottle = constants(11);
    T_air_0 = constants(12);
    m_air_0 = constants(13);
    R = constants(14);
    g = constants(15);
    R_stand = constants(16);
    theta_0 = constants(17);
    wind_x = constants(18);
    wind_y = constants(19);
    l_heading = constants(20);

    horiz_0 = cos(theta_0); % Because theta was used in 2012 code, let's just keep it and convert it to vector direction
    z_dir_0 = sin(theta_0);
    x_dir_0 = cos(l_heading) * horiz_0;
    y_dir_0 = sin(l_heading) * horiz_0;
    dir_0_mag = sqrt( x_dir_0^2 + y_dir_0^2 + z_dir_0^2);

% State Values
    p_x = y(1);
    p_z = y(2);
    v_x = y(3) + wind_x;
    v_z = y(4);
    m_bottle = y(5);
    m_air = y(6);
    V_air = y(7);
    p_y = y(8);
```

```
    v_y = y(9) + wind_y;

% Theta
rocket_pos = sqrt( p_x ^ 2 + p_y ^ 2 + p_z ^ 2 );
if ( rocket_pos > R_stand ) % determine if rocket has left stand
    v_mag = sqrt( v_x^2 + v_y^2 + v_z^2);
    z_dir = v_z / v_mag;
    y_dir = v_y / v_mag;
    x_dir = v_x / v_mag;
else
    x_dir = x_dir_0 / dir_0_mag;
    y_dir = y_dir_0 / dir_0_mag;
    z_dir = z_dir_0 / dir_0_mag;
end

% Velocity
v = sqrt( v_x ^ 2 + v_y ^ 2 + v_z ^ 2 );

%% PHASE 1: Water Expulsion

    if V_air < V_bottle

        % Air Pressure
        % Equation (3)
        P_air = P_bottle * (V_air_0 / V_air) ^ cp_rat;

        % Exhaust Velocity
        % Equation (7)
        v_exhaust = sqrt( 2 * ( P_air - P_amb ) / p_water );

        % Mass Flow Rate of Water
        % Equation (4)
        m_dot_w = C_d * p_water * A_throat * v_exhaust;

        % Mass Flow Rate of Rocket
        % Equation (10)
        m_dot_r = -m_dot_w;

        % Mass Flow Rate of Air
        m_dot_a = 0; % Air mass doens't change in this phase

        % Rate of Change of Volume of Air
        % Equation (9)
        V_dot = C_d * A_throat * v_exhaust;

        % Force of Thrust
        % Equation (8)
        F_thrust = 2 * C_d * A_throat * (P_air - P_amb);

    else % set up Pressure and Temperatures for Phase 2 and 3

        % Air Pressure and Temperature After Water Exhausted
        % Equation (13)
        P_end = P_bottle * ( V_air_0 / V_bottle ) ^ cp_rat;
```

```
        % New Pressure Equation
        % Equation (14)
        P_air = P_end * (( m_air / m_air_0)) ^ cp_rat;

    end

%% PHASE 2: Gas Expulsion

    if (V_air >= V_bottle) && (P_air > P_amb)

        % Change in Volume
        V_dot = 0;

        % Calculate Density and Temperature
        % Equation (15)
        rho = m_air / V_bottle;
        T = P_air / ( rho * R );

        % Critical Pressure
        % Equation (16)
        P_crit = P_air * ( 2 / ( cp_rat + 1 )) ^ ( cp_rat / ( cp_rat - 1));

        % If Choked Flow
        if ( P_crit > P_amb )

            % Exit Temperature
            % Equation (18)
            T_exit = ( 2 / ( cp_rat + 1 )) * T;

            % Exit Velocity
            % Equation (17)
            V_exit = sqrt( cp_rat * R * T_exit );

            % Exit Pressure
            % Equation (18)
            P_exit = P_crit;

        % If Not Choked Flow
        else

            % Exit Mach Number
            % Equation (19)
            Mach_exit = sqrt(abs((( P_air / P_amb ) ^ (( cp_rat - 1) / cp_rat)
                - 1 ) / (( cp_rat - 1 ) / 2)));

            % Exit Temperature
            % Equation (20)
            T_exit = T / ( 1 + ( ( cp_rat - 1) / 2 ) * Mach_exit ^ 2);

            % Exit Pressure
            % Equation (20)
            P_exit = P_amb;

```

```
            % Exit Velocity
            % Equation (21)
            V_exit = Mach_exit * sqrt(abs( cp_rat * R * T_exit ));

        end

        % Exit Density
        % Equation (18) and (20)
        rho_exit = P_exit / ( R * T_exit);

        % Change in Air Mass
        % Equation (23)
        m_dot_a = -C_d * rho_exit * A_throat * V_exit;

        % Force of Thrust
        % Equation (22)
        F_thrust = (-m_dot_a * V_exit) + (( P_amb - P_exit ) * A_throat);

        % Change in Rocket Mass
        % Equation (24)
        m_dot_r = m_dot_a;

    end

%% PHASE 3: Ballistic Phase

    if (V_air >= V_bottle) && (P_air <= P_amb)

        % No more water or gas so theres no change in mass, volume, or thrust
        % Equation (25)
        F_thrust = 0;
        m_dot_r = 0;
        m_dot_a = 0;
        V_dot = 0;

    end

%% Final State Function Calculations

    % Force of Drag
    % Equation (2)
    F_drag = (1/2) * p_amb * (v ^ 2) * cd * A_bottle;

    % Sum of Forces
    % Equation (1)
    sum_F_x = F_thrust * x_dir - F_drag * x_dir;
    sum_F_y = F_thrust * y_dir - F_drag * y_dir;
    sum_F_z = F_thrust * z_dir - F_drag * z_dir - m_bottle * g;

    % Acceleration
    % Equation (1)
    acc_x = sum_F_x / m_bottle;
    acc_y = sum_F_y / m_bottle;
    acc_z = sum_F_z / m_bottle;
```

```matlab

    % Keep Above x-axis
    if p_z <= 0
        v_x = 0;
        v_z = 0;
        acc_x = 0;
        acc_z = 0;
        m_dot_r = 0;
        m_dot_a = 0;
        V_dot = 0;
        v_y = 0;
        acc_y = 0;
    end

    % Resulting State Values
    state_vector(1,1) = v_x;
    state_vector(2,1) = v_z;
    state_vector(3,1) = acc_x;
    state_vector(4,1) = acc_z;
    state_vector(5,1) = m_dot_r;
    state_vector(6,1) = m_dot_a;
    state_vector(7,1) = V_dot;
    state_vector(8,1) = v_y;
    state_vector(9,1) = acc_y;

end

%% Error Ellipses Function

function [mean_x, mean_y] = error_ellipses(xyz_data)

N = length(xyz_data);
x = xyz_data(:,1);
y = xyz_data(:,2);

figure(3)
plot(x,y,'k.','markersize',6)
axis equal
grid on
xlabel('Distance (East) [m]')
ylabel('Distance (North) [m]')
hold on

% Calculate Covariance Matrix
P = cov(x,y);
mean_x = mean(x);
mean_y = mean(y);

% Calculate the define the error ellipses
n=100; % Number of points around ellipse
p=0:pi/n:2*pi; % angles around a circle

[eigvec,eigval] = eig(P); % Compute eigen-stuff
xy_vect = [cos(p'),sin(p')] * sqrt(eigval) * eigvec'; % Transformation
```

```
x_vect = xy_vect(:,1);
y_vect = xy_vect(:,2);

% Plot the error ellipses overlaid on the same figure
plot(1*x_vect+mean_x, 1*y_vect+mean_y, 'b')
plot(2*x_vect+mean_x, 2*y_vect+mean_y, 'g')
plot(3*x_vect+mean_x, 3*y_vect+mean_y, 'r')

end
```

X. References and Acknowledgements

A. References

[1] Anderson, J. D., Jr., Introduction to Flight, 7th Ed., McGraw-Hill (2009).

[2] Sutton, G. and Biblarz, O., Rocket Propulsion Elements, 8th Ed., Wiley (2010).

[3] J. Sellers, W. Astore, R. Giffen, and W. Larson, "Understanding Space: An Introduction to Astronautics, 3th Ed.", McGraw Hill, 2005.

[4] Taylor, John R., "An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements, 2th Ed.", University Science Books, 1982.

[5] The MathWorks Inc., MATLAB, version 2020a, accessed Apr 12, 2021, https://mathworks.com/.

[6] Zagrodzki, Maciej, "CSCI 1320: Computer Science 1: Starting Computing-Engineering Applications – Spring 2020", Lecture series at the University of Colorado Boulder, 2020.

[7] Thayer, J. and Mah, J., "ASEN 2002: Introduction to Thermodynamics and Aerodynamics – Fall 2020", Lecture series at the University of Colorado Boulder, 2020.

[8] Jackson, Jelliffe, "Experimental Computational Methods – Fall 2020", Lecture series at the University of Colorado Boulder, 2020.

[9] ASEN 2012 Course Staff, "Verification Case For Project 2", University of Colorado Boulder, 2020.

[10] ASEN 2012 Course Staff, "ASEN 2012 - Bottle Rocket Design 2020 - Equations of Motion", University of Colorado Boulder, 2020.

[11] ASEN 2012 Course Staff, "ASEN 2012 Project 2: Bottle Rocket Modeling", Lab instruction with the University of Colorado Boulder, 2020.

[12] Johnson, A. and Mah, J., "ASEN 2004: Vehicle Design and Performance – Spring 2021", Lecture series at the University of Colorado Boulder, 2021.

[13] Johnson, A. and Schwartz, T., "ASEN 2004: Vehicle Design and Performance – Spring 2021", Lab Instruction with the University of Colorado Boulder, 2021.

[14] Schwartz, Trudy, "ASEN 2004 - Optimized Launch Data Sheet Post flight", University of Colorado Boulder, 2021.

[15] ASEN 2004 Course Staff, "ASEN 2004 - Baseline Launch Data Sheet", University of Colorado Boulder, 2021.

[16] ASEN 2004 Course Staff, "ASEN 2004 - Water Bottle Rocket Lab - Rocket Model Check", University of Colorado Boulder, 2021.

[17] ASEN 2004 Course Staff, "ASEN 2004 - Water Bottle Rocket Lab - Appendix A: Thrust Model Details", University of Colorado Boulder, 2021.

B. Acknowledgements

[18] Teaching Assistants, "ASEN 2012: Experimental Computational Methods – Fall 2020", Lab instruction with the University of Colorado Boulder, 2020.

[19] Teaching Assistants, "ASEN 2004: Vehicle Design and Performance – Spring 2021", Lab instruction with the University of Colorado Boulder, 2020.