ASEN 2004 Lab 2 Executive Summary

Hannah Blanchard Obolsky, Luca Bonarrigo, Nick Larson, Jason Popich, Ryan Sievers, Mark Wilbourne
Smead Aerospace Engineering Sciences; University of Colorado - Boulder

I. Introduction

"To infinity and beyond!" is a great concept in theory. However, when it comes to designing a bottle rocket in reality, it takes a bit more work to bring that concept to fruition. This report outlines the process used to develop a model for a bottle rocket's trajectory, how that process (along with variations of key parameters) was used to maximize downrange distance, and how error analysis was employed to ensure that the model was as accurate and precise as possible. While the bottle rocket didn't quite reach infinity, the engineering methods employed here are an excellent proof of concept of how simple thermodynamics and MATLAB coding can be used in real world applications.

II. Methods

A. Description of Phases

During the first phase, the rocket is launched from an initial height of 0.25 meters and from an initial distance of 0 meters. The exhaust velocity depends on the changing pressure inside the bottle, which in turn determines the volumetric flow rate of the air and the thrust leaving the bottle. After all the water is gone, isentropic relationships are used to estimate the end pressure and temperature. During the first phase of the launch, friction, thrust, gravity, and drag act on the bottle rocket. Because friction acts over such a short amount of time, it can be assumed that its impact on the trajectory is negligible.

Once the water is expelled, the second phase takes over. While the pressure in the bottle is greater than the ambient pressure, the volumetric flow rate of air remains zero. The critical pressure can be determined as a result of the current pressure and the specific heat ratio known as γ. Once calculated, it can be used to determine if the flow from the bottle is in choked flow. The resulting velocities, mass flows, and thrust forces depend on the pressure and thus choked or unchoked flow. During the second phase, thrust, gravity, and drag act on the bottle rocket. Once the pressure is equalized, the force of thrust stops propelling the bottle.

The final phase begins once the pressure inside the bottle is equalized with the ambient air pressure. As there is nothing left for the bottle rocket to expel, drag and gravity are the only forces acting on the bottle as it follows a parabolic flight path and eventually hits the ground.

The majority of the time will be spent in phase three, followed by phase one, and finally phase two. In addition, the forces acting on the bottle change as a result of the orientation of the bottle, causing a further change in the behavior of the trajectory. The equation of motion, Equation 1, was applied to each phase. Note that this is the most expanded version of the equation; certain values will be zero for different phases, based on the behavior outlined above.

$$\Sigma F = \dot{m}a = T - D - F_g = (T - D)\hat{\imath} + (T - D)\hat{\jmath} + (T - D - mg)\hat{k} \tag{1}$$

Within this equation, T represents thrust, D represents drag, F_g represents gravity, and the vector notation corresponds to the xyz coordinate system ($\hat{\imath}$ for downrange, $\hat{\jmath}$ for crossrange, and $\hat{k}$ for altitude). Because the mass of the bottle rocket decreases with time, it is represented with $\dot{m}$. Other values that change with time include the pressure inside the bottle, the flow rate of fluids out of the bottle, the heading vector, and the value of thrust. Drag and gravity are constant for all three phases.

B. Experimental Methods

1. Static Test Stand

The static test stand was used to generate thrust data based on the propulsion phase of the rocket. The data was collected by fastening load cells to the bottle rocket and "launching" the bottle rocket (the rocket was clamped down to prevent it from actually leaving the ground). The data was in turn used in developing the Ideal Rocket Equation and the Thrust Interpolation models. To do this, the area under the force curve was calculated to derive the specific impulse of the rocket I_{sp} and associated values of max/average thrust. The static test stand report found an I_{sp} of 1.58 ± 0.14 s and a peak thrust of 208 ± 20 N.

2. LA Baseline Rocket Launch

The LA Baseline rocket launch was used to provide calibration for the bottle rocket model. For this experiment, a series of measurements were made on the bottle and the surrounding environment, including the five parameters that would later be varied in the sensitivity analysis. With the preliminary measurements thus made, the bottle was launched and its final distance was recorded by the LAs. This was then used in comparison with the different models to determine which model was the most accurate.

C. Computational Methods

Three models were developed to predict the trajectory of the bottle rocket. The first was a thermodynamic model, based off of the thermodynamics of water and air expansion. The second was interpolating a thrust curve of the bottle rocket calculated from static test stand data. The third and final model was created using the ideal rocket equation and the specific impulse (I_{sp}) of the water bottle rocket calculated from static test stand data. There were advantages and disadvantages to each of the models used. Of all these models, however, the thermodynamic model was determined to be the most accurate when compared to the LA launch data.

III. MATLAB Modeling

The thermodynamic model was chosen over both the thrust interpolation and specific impulse models, with the main reason being that the thermodynamic model relied on the changes of the air and the water within the bottle over time, making the variation of these parameters easier. With the thrust interpolation model, the thrust data was given with time and did not rely on the air or water at any given point. Additionally, with the rocket equation model, the specific impulse was calculated by integrating the thrust data and dividing by the mass of propellant, and assumed the thrust to be instantaneous, thus creating an immense drag force at the very start of the launch. In both of these situations, many of the variable parameters were not factors, therefore rendering the interpolation and rocket equation models less malleable and thus less ideal than the thermodynamic model (see Appendix A for the code flow chart describing the computational logic for the sensitivity analyses and rocket design).

IV. Baseline Rocket & Payload

The LA Baseline Rocket, which was used for calibration, had a dry mass of 160 grams, a coefficient of drag of 0.3, a launch angle of 45°, and three triangular fins in a symmetrical arrangement. The rocket also carried an altimeter to track the altitude of the rocket, and its mass was added in the dry mass total. However, the added mass to the nose of the rocket needed to be accounted for in the calculation of the center of gravity. Having an added mass on top of the rocket shifted the center of gravity forward, making the rocket more stable. Because the nosecone covered the altimeter, leaving the nosecone unchanged would have blocked ambient air from getting to the sensor and thus given false readings for altitude. Adding holes fixed this issue by providing the ambient air a path to reach the sensor so it could read the pressure correctly. Figures 1 and 2 depict the trajectory of the LA's bottle rocket. Figure 1 was created by processing the video of the rocket's flight using Logger Pro software, while Figure 2 was produced by the MATLAB GUI that processed altimeter data. Both the software and GUI were provided by the instructional staff.

V. Modified Rocket Design

To determine the optimized design for the modified rocket, five parameters were varied individually to determine which of the five would produce a rocket that would travel the furthest downrange distance. To determine the optimum value of each parameter, a line of best fit was fitted to each curve (outliers were disregarded to increase the accuracy of these best fit lines).

A. Launch Pad Angle

The launch pad angle for the rocket was varied from 0 to 90 degrees. As depicted by Figure 3, launch angles of 0 and 90 degrees produced near no downrange distance. The baseline launch angle was 45 degrees, and the line of best

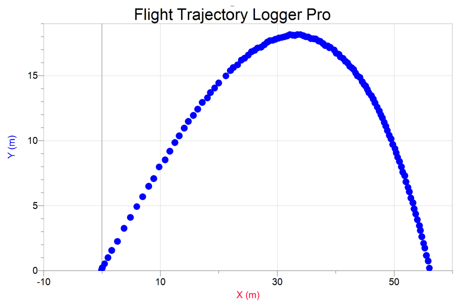

Fig. 1 Flight Trajectory Video Capture

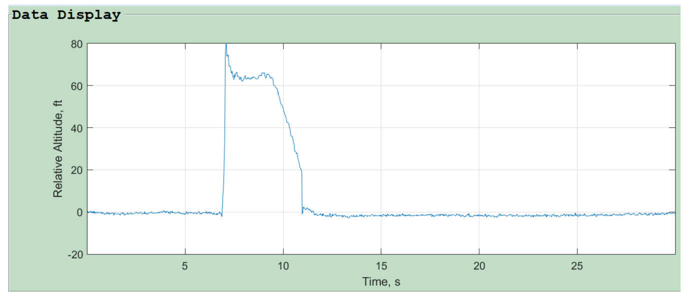

Fig. 2 Altitude vs Time MATLAB GUI

fit found a similar optimum launch angle of 44.5^o. The slight difference may reasonably be attributed to a potential difference in drag.

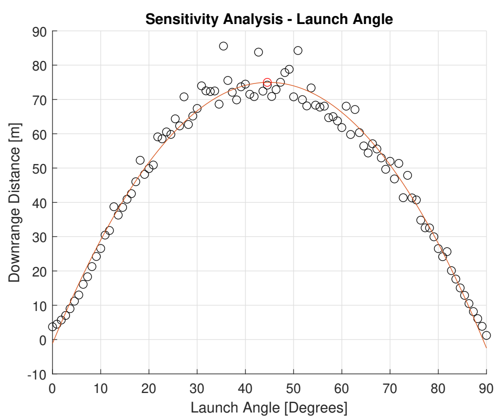

Fig. 3 Launch Angle Sensitivity Analysis

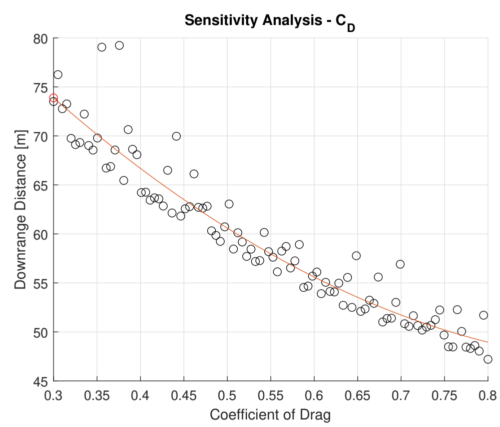

Fig. 4 Coefficient of Drag Sensitivity Analysis

B. Coefficient of Drag

The coefficient of drag was varied between the baseline of 0.3 and 0.8. As expected, there was a negative linear correlation between increasing drag coefficient and the downrange distance traveled, as more drag will reduce the rocket's travelling distance. As seen in Figure 4, the distance traveled was maximized for a coefficient of drag of 0.3. The data was also fairly consistent, with only a few moderate outliers existing throughout the different coefficients of drag.

C. Mass of the Water Propellant

The initial water mass was varied between 0.3 kg and 1 kg, with a baseline of 0.6 kg. The downrange distance followed a parabolic curve starting and ending at about 60 meters downrange, with the apex given by the best fit line at 74.65 meters as shown in Figure 5, which is a gain of 1.15 meters from the baseline value. The new value of initial water mass to achieve this gain is 0.63 kg. As with the rest of the sensitivity analyses, the outliers were ignored in creating the best fit line.

D. Density of the Water Propellant

Water density was varied throughout the sensitivity analysis from 900 to $1100\frac{kg}{m^3}$. As shown in Figure 6, there was a very weak correlation between the water density and the distance traveled. The mean distance traveled fell between 74 and 75 meters, though they ranged from 70 to about 83 meters with a fairly even distribution throughout different water densities. Because of the weak correlation, this parameter was removed from consideration for optimizing the rocket's design.

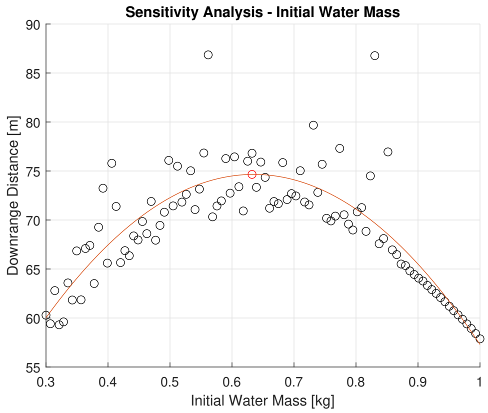

Fig. 5 Initial Mass of Water Sensitivity Analysis

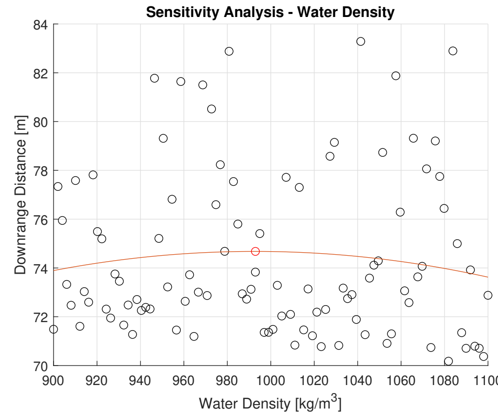

Fig. 6 Density of Water Sensitivity Analysis

E. Temperature of the Water Propellant

The temperature of the water propellant was varied from 0 to 30 degrees Celsius, or 273.15 to 303.15 degrees Kelvin, as shown in Figure 7. Similar to water density, there was little correlation between the downrange distance and the temperature of the water, with a majority of the downrange distances falling relatively evenly between 70 and 76 meters regardless of temperature. There were a few outliers that occurred near the extremes of the temperature range, with none occurring between 285 and 295 Kelvin, though this pattern holds little consistency. Again, due to the weak correlation, this parameter was removed from consideration for optimizing the rocket's design.

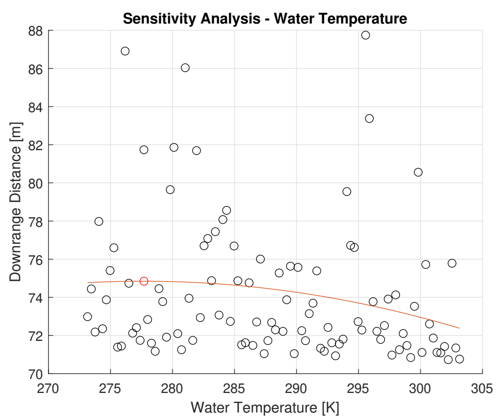

Fig. 7 Temperature of Water

F. Chosen Modified Parameter

The parameter chosen for modification for the rocket launch was the initial water mass. The baseline for this value was 0.6 kg, to which 0.03 kg will be added for a total initial water mass of 0.63 kg for the modified rocket design. Both the temperature of water and the density of water were disregarded as changeable parameters due to their lack of correlation between change in the parameter and change in downrange distance. Launch angle and coefficient of drag were the other two variables that were considered, but ultimately decided against because they had little or no change from the baseline value; for example, the optimum launch angle was determined from the line of best fit to be 44.5^o. As a protractor has a human error of $\pm 1^o$, it would be very difficult to ensure that the launch angle is correct to the degree of change that is required. The next best option fell to the initial water mass, which has a reasonable change from the baseline value and the second highest gain apart from launch angle.

VI. Predictions for Rocket Flight

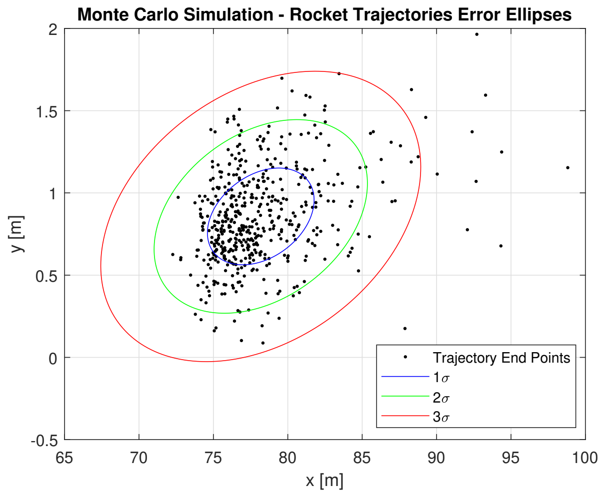

Fig. 8 Flight Predictions - Monte Carlo Simulation

A. Monte Carlo Simulation Results

Once the parameter was chosen and modified, a Monte Carlo simulation was run to estimate the landing site of the modified rocket. Figure 8 depicts these results plotted within error ellipses. The 3σ portion of the error ellipses extends roughly between 67 and 87 meters, or over a 20 meter span downrange. Cross range, it extends between -0.05 and 1.7 meters or over a 1.75 meter span.

B. List of Uncertainties

Errors were determined based on the increments of the measurement tool (protractor, scale, ruler, etc) and on the chance of human interference. It was assumed that systematic errors and inaccuracies were negligible within the tools used for measurements. It was also assumed that the errors were all normally distributed with a mean of 0, with each standard deviation outlined below:

- Launch Pad Angle $\pm 1°$
- Mass $\pm 0.5g$
- Pressure in Bottle $\pm 0.1psi$ or $\pm 689Pa$
- Coefficient of Drag ± 0.05
- Wind ± 0.5 mph or $\pm 10\%$ of max speed
- Wind Direction $\pm 22.5°$
- Density of Water $\pm 1\frac{kg}{m^3}$
- Sizing Measurements $\pm 0.001m$

VII. Lessons Learned

If the team had the chance to redo this lab, the three initial models (thermodynamic, rocket equation, and interpolation) could have been combined in order to increase the fidelity of the rocket model. This may have facilitated the process of determining which of the five parameters should be modified to optimize the rocket design, as a combination of models may have produced less outliers in the sensitivity analyses. In addition, restraints due to COVID-19 meant this lab was executed virtually; if this lab were to be completed in person, a greater exploration of the errors could have been completed with further experimentation. For example, the instruments used for measurement could have been further calibrated based on small-scale experiments. Overall, this lab provided the team with insights on how various parameters affect the thrust curve and overall flight trajectory of a rocket, and made use of various engineering tools to analyse these parameters and design a bottle rocket.

Acknowledgments

Thank you to the ASEN 2004 instructional team and teaching assistants for their assistance on this project.

VIII. References

[1] ASEN 2004, *ASEN 2004 Water Bottle Rocket Lab Deliverable 2:Static Test Stand Report*
[2] ASEN 2004, *ASEN 2004 Water Bottle Rocket Lab Deliverable 3: Rocket Thrust Model*
[3] ASEN 2004, *ASEN 2004 Water Bottle Rocket Lab Deliverable 4: Executive Summary*
[4] ASEN 2004, *Procedure for AltimeterGUI.m*

Appendix

A. MATLAB Code Flow Chart

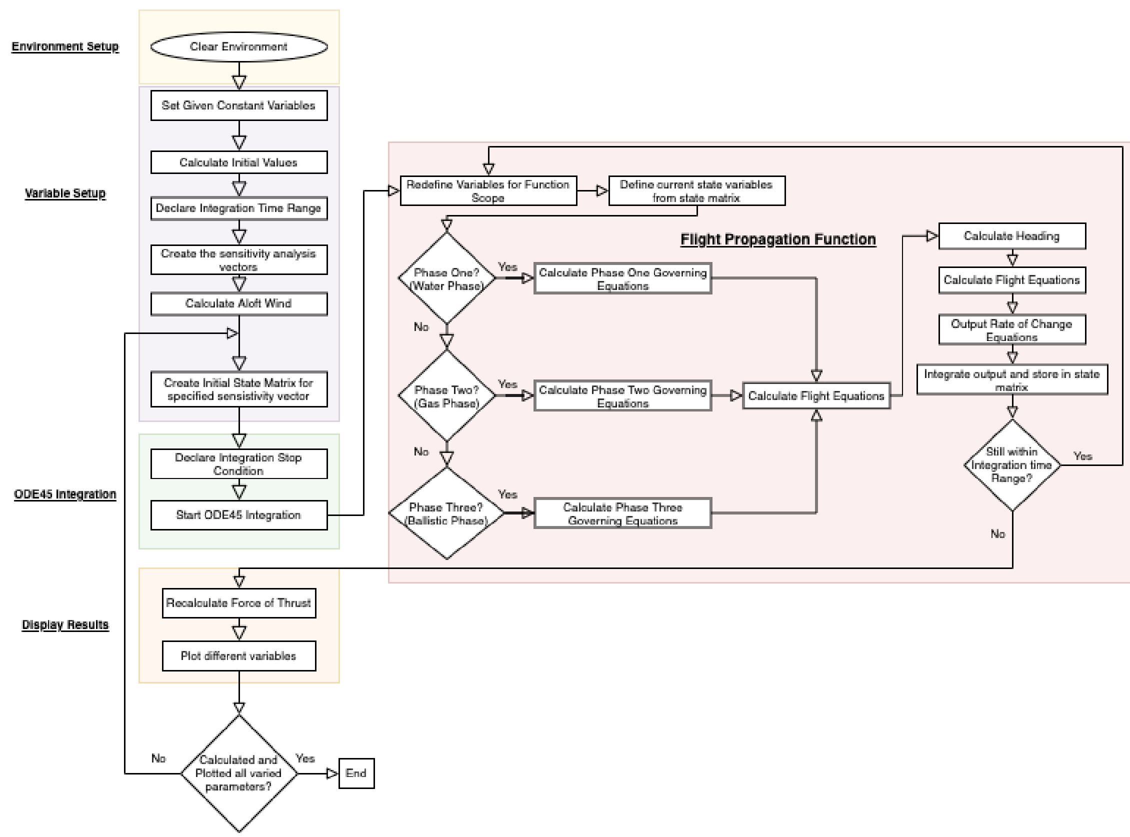

Fig. 9 MATLAB Code Flow Chart

B. MATLAB Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                          Lab #2 - ASEN 2004                                  %
%                            Bottle Rocket                                     %
%                               V 2.0                                          %
%                                                                              %
%                                                                              %
%                             04/18/2020                                       %
%                  Configured for LA Optimized Rocket                          %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Set Environment
clear
clc

%% Set Variables

% Set Launch Datasheet Vars
C_D = 0.30;                        % Drag Coefficient ----
temperature_air = 63;              % Initial Temperature of Air [degrees F] ----
p_air_gage_initial = 40;           % Gage pressure of inside of bottle [psi] ----
m_bottle = 0.160;                  % Mass of the Bottle [kg]
m_water_initial = 0.600;           % Initial Mass of the Water [kg]
mass_initial = 0.760;              % Total Initial Mass [kg] -----

theta = 45;                        % Initial Vertical Angle of Rocket [Degrees] ----
beta = 40;                         % Initial Horizontal Angle of Rocket from north [Degrees] ----

V_w_g = 0;                         % Ground Wind Speed [mph] !! MAX WIND SPEED OF 10 [mph] !! ----
V_w_a = 3;                         % Aloft Wind Speed [mph] !! MAX WIND SPEED OF 10 [mph] !! ----
W_d_g = 0;                         % Ground Wind Direction relative to rocket nose at 0 degrees [degrees] ----
W_g_e = 11.25;                     % Ground Wind Direction Error [Degrees]
W_d_a = 45;                        % Aloft Wind Direction relative to rocket nose at 0 degrees [degrees] ----
W_a_e = 11.25;                     % Aloft Wind Direction Error [Degrees]

% Constant Variables
R = 287;                           % Gas Constant [J*kg^-1*K^-1]
g = 9.81;                          % Acceleration of Gravity [m/s^2]
c_d = 0.8;                         % Discharge Coefficient
y = 1.4;                           % Specific Heat Ratio of Air [Unitless?]
temperature_air_initial = ((temperature_air - 32) * 5/9) + 273.15;     % Initial Temperature of Air [degrees K]

% Measurements
d_throat = 2.1;                    % Diameter of throat [cm]
d_bottle = 10.5;                   % Diameter of bottle [cm]
```

```
% Pressure
p_ambient = 12.1;                % Ambient pressure [psi]

% Volume
% vol_water_initial = 0.001;       % Volume of water in bottle [m^3]
vol_bottle = 0.002;              % Vol of bottle [m^3]

% Density
rho_water = 1000;                % Density of Water [kg/m^3]
rho_amb_air = 0.961;             % Density of Ambient Air [kg/m^3]

% Flight Variables
V_X_0 = 0;                       % Initial Velocity of Rocket in X Direction [m/s]
V_Y_0 = 0;                       % Initial Velocity of Rocket in Y Direction [m/s]
V_Z_0 = 0;                       % Initial Velocity of Rocket in Z Direction [m/s]

X_0 = 0;                         % Initial X Distance of Rocket [m]
Y_0 = 0;                         % Initial Y Distance of Rocket [m]
Z_0 = 0.25;                      % Initial Z Distance of Rocket [m]

l_s = 0.5;                       % Initial Length of Test Stand [m]

V_w_g = V_w_g*0.44704;           % Ground Wind Speed [m/s] !! MAX WIND SPEED OF 4.47 [m/s] !!
V_w_a = V_w_a*0.44704;           % Aloft Wind Speed [m/s] !! MAX WIND SPEED OF 4.47 [m/s] !!

%% Calculate Initial Values

% Initial Angle of Rocket in Radians
theta_rad = deg2rad(theta);

% Pressure inside Rocket
p_air_initial = (p_air_gage_initial*6894.76) + (p_ambient*6894.76); % [N/m^2]

% Initial Volume of Water
vol_water_initial = m_water_initial / rho_water;

% Initial Volume of Air
vol_air_initial = vol_bottle - vol_water_initial; % volume of air in bottle [m^3]

% Initial Mass of Rocket
% m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial); % Mass of Air [kg]

% m_water_initial = rho_water*(vol_bottle - vol_air_initial); % Mass of Water [kg]
% m_water_initial = 1.001; % [kg]

```

```
% m_bottle = 0.15; % Mass of Bottle [kg]
% m_bottle = 0.128; % [kg]

% mass_initial = m_bottle + m_water_initial + m_air_initial;  % Total Mass [kg]
% mass_initial = 1.129; % Total Initial Mass [kg] -----

% Area of Throat
A_t = pi*((d_throat*0.01)/2)^2; % m^2

% Area of Bottle
A_b = pi*((d_bottle*0.01)/2)^2; % m^2

% Time Range for Solver; 0-5 Seconds
tspan = [0 10];

%% Sensitivity Analysis
% Vary: C_D, Mass of water, density of water, temperature of water, launch
% pad angle
n = 100;                                               % Number of Iterations for Simulation

theta_varied = linspace(0,90);                         % Varied Launch Pad Angle [Degrees]

C_D_varied = linspace(0.3,0.8);                        % Varied Coefficient of Drag [unitless]

rho_water_varied = linspace(900,1100);                 % Varied Water Density [kg/m^3]

m_water_initial_varied = linspace(0.300,1.000);        % Varied Initial Mass of the Water [kg]

temperature_air_initial_varied = linspace(0,30);       % Varied Initial Temperature of Air/Water [C]
temperature_air_initial_varied = temperature_air_initial_varied + 273.15;       % Varied Initial Temperature of Air/Water [K]

%% Set values
% Adjust Angles to be relative to rocket
W_d_g = beta - W_d_g;
W_d_a = beta - W_d_a;

% Calculate Ground Wind in xy-plane
W_g_x = V_w_g*cos(deg2rad(W_d_g));
W_g_y = V_w_g*sin(deg2rad(W_d_g));
W_g_z = 0;

% Calculate Aloft Wind in xy-plane
W_a_x = V_w_a*cos(deg2rad(W_d_a));
W_a_y = V_w_a*sin(deg2rad(W_d_a));
W_a_z = 0;

```

```
%% BASELINE
% Initial Volume of Water
vol_water_initial = m_water_initial / rho_water;

% Initial Volume of Air
vol_air_initial = vol_bottle - vol_water_initial; % volume of air in bottle [m^3]

% Initial Mass of Rocket
m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial); % Mass of Air [kg]

% Variable Declaration for Solver
vars = [rho_water rho_amb_air c_d C_D A_t A_b p_air_initial vol_air_initial...
    y p_ambient vol_bottle l_s g temperature_air_initial m_air_initial...
    R deg2rad(theta) Z_0 X_0 Y_0 W_g_x W_g_y W_g_z W_a_x W_a_y W_a_z];

% Initial State of Rocket
initial_state = [Z_0 X_0 Y_0 mass_initial m_air_initial vol_air_initial V_Z_0 V_X_0 V_Y_0];

% Set Terminal Condition (i.e. Rocket hit ground)
terminalCond = odeset('Events', @hitGround);

% Propagate Flight
[t,state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state, terminalCond);

baseline = state(end,2);

gain = zeros(1,5);

%% 1: LAUNCH ANGLE
figure()
for i=1:n
    % Initial Volume of Water
    vol_water_initial = m_water_initial / rho_water;
    mass_initial = .160 + m_water_initial;

    % Initial Volume of Air
    vol_air_initial = vol_bottle - vol_water_initial; % volume of air in bottle [m^3]

    % Initial Mass of Rocket
    m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial); % Mass of Air [kg]

    % Variable Declaration for Solver
    vars = [rho_water rho_amb_air c_d C_D A_t A_b p_air_initial vol_air_initial...
        y p_ambient vol_bottle l_s g temperature_air_initial m_air_initial...
        R deg2rad(theta_varied(i)) Z_0 X_0 Y_0 W_g_x W_g_y W_g_z W_a_x W_a_y W_a_z];

```

```
    % Initial State of Rocket
    initial_state = [Z_0 X_0 Y_0 mass_initial m_air_initial vol_air_initial
        V_Z_0 V_X_0 V_Y_0];

    %% ODE SOLVER
    % Set Terminal Condition (i.e. Rocket hit ground)
    terminalCond = odeset('Events', @hitGround);

    % Propagate Flight
    [t,state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state,
        terminalCond);

    % Plot
    hold on
    plot3(state(:,2),state(:,3),state(:,1));
    grid

    % Save End Points for each trajectory
    X(i) = state(end,2);
    Y(i) = state(end,3);

    % Calculate Force of Thrust
    [F_thrust,phaseChange] = f_thrust(t,state,vars);
end
hold off
xlabel("Downrange Distance [m]");
ylabel("Crossrange Distance [m]");
zlabel("Altitude [m]");
title("Sensitivity Analysis - Launch Angle");

poly = polyfit(theta_varied,X,2);
x2 = polyval(poly,theta_varied);

figure()
hold on
plot(theta_varied,X,'ko');
plot(theta_varied,x2);
[maximum,idx] = max(x2);
plot(theta_varied(idx),maximum,'ro');
grid
ylabel("Downrange Distance [m]");
xlabel("Launch Angle [Degrees]");
title("Sensitivity Analysis - Launch Angle");
hold off

fprintf('Baseline launch angle: %.2f degrees\n',theta);
fprintf('Optimum launch angle: %.2f degrees\n',theta_varied(idx));
fprintf('Gain in downrange distance: %.2f m\n\n',maximum-baseline);
gain(1) = maximum-baseline;


%% 2: COEFFICIENT OF DRAG
figure()
for i=1:n
```

```
    % Initial Volume of Water
    vol_water_initial = m_water_initial / rho_water;
    mass_initial = .160 + m_water_initial;

    % Initial Volume of Air
    vol_air_initial = vol_bottle - vol_water_initial; % volume of air in bottle [m^3]

    % Initial Mass of Rocket
    m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial); % Mass of Air [kg]

    % Variable Declaration for Solver
    vars = [rho_water rho_amb_air c_d C_D_varied(i) A_t A_b p_air_initial vol_air_initial...
        y p_ambient vol_bottle l_s g temperature_air_initial m_air_initial...
        R deg2rad(theta) Z_0 X_0 Y_0 W_g_x W_g_y W_g_z W_a_x W_a_y W_a_z];

    % Initial State of Rocket
    initial_state = [Z_0 X_0 Y_0 mass_initial m_air_initial vol_air_initial V_Z_0 V_X_0 V_Y_0];

    %% ODE SOLVER
    % Set Terminal Condition (i.e. Rocket hit ground)
    terminalCond = odeset('Events', @hitGround);

    % Propagate Flight
    [t,state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state, terminalCond);

    % Plot
    hold on
    plot3(state(:,2),state(:,3),state(:,1));
    grid

    % Save End Points for each trajectory
    X(i) = state(end,2);
    Y(i) = state(end,3);

    % Calculate Force of Thrust
    [F_thrust,phaseChange] = f_thrust(t,state,vars);
end
hold off
xlabel("Downrange Distance [m]");
ylabel("Crossrange Distance [m]");
zlabel("Altitude [m]");
title("Sensitivity Analysis - C_D");

poly = polyfit(C_D_varied,X,2);
x2 = polyval(poly,C_D_varied);

figure()
hold on
plot(C_D_varied,X,'ko');
```

```
plot(C_D_varied,x2);
[maximum,idx] = max(x2);
plot(C_D_varied(idx),maximum,'ro');
grid
ylabel("Downrange Distance [m]");
xlabel("Coefficient of Drag");
title("Sensitivity Analysis - C_D");
hold off

fprintf('Baseline coefficient of drag: %.2f\n',C_D);
fprintf('Optimum coefficient of drag: %.2f\n',C_D_varied(idx));
fprintf('Gain in downrange distance: %.2f m\n\n',maximum-baseline);
gain(2) = maximum-baseline;

%% 3: DENSITY OF WATER
figure()
for i=1:n
    % Initial Volume of Water
    vol_water_initial = m_water_initial / rho_water_varied(i);
    mass_initial = .160 + m_water_initial;

    % Initial Volume of Air
    vol_air_initial = vol_bottle - vol_water_initial; % volume of air in
        bottle [m^3]

    % Initial Mass of Rocket
    m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial
        ); % Mass of Air [kg]

    % Variable Declaration for Solver
    vars = [rho_water_varied(i) rho_amb_air c_d C_D A_t A_b p_air_initial
        vol_air_initial...
         y p_ambient vol_bottle l_s g temperature_air_initial m_air_initial...
         R deg2rad(theta) Z_0 X_0 Y_0 W_g_x W_g_y W_g_z W_a_x W_a_y W_a_z];

    % Initial State of Rocket
    initial_state = [Z_0 X_0 Y_0 mass_initial m_air_initial vol_air_initial
        V_Z_0 V_X_0 V_Y_0];

    %% ODE SOLVER
    % Set Terminal Condition (i.e. Rocket hit ground)
    terminalCond = odeset('Events', @hitGround);

    % Propagate Flight
    [t,state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state,
        terminalCond);

    % Plot
    hold on
    plot3(state(:,2),state(:,3),state(:,1));
    grid

    % Save End Points for each trajectory
    X(i) = state(end,2);
```

```
    Y(i) = state(end,3);

    % Calculate Force of Thrust
    [F_thrust,phaseChange] = f_thrust(t,state,vars);
end
hold off
xlabel("Downrange Distance [m]");
ylabel("Crossrange Distance [m]");
zlabel("Altitude [m]");
title("Sensitivity Analysis - Water Density");

poly = polyfit(rho_water_varied,X,2);
x2 = polyval(poly,rho_water_varied);

figure()
hold on
plot(rho_water_varied,X,'ko');
plot(rho_water_varied,x2);
[maximum,idx] = max(x2);
plot(rho_water_varied(idx),maximum,'ro');
grid
ylabel("Downrange Distance [m]");
xlabel("Water Density [kg/m^3]");
title("Sensitivity Analysis - Water Density");
hold off

fprintf('Baseline water density: %.2f kg/m^3\n',rho_water);
fprintf('Optimum water density: %.2f kg/m^3\n',rho_water_varied(idx));
fprintf('Gain in downrange distance: %.2f m\n\n',maximum-baseline);
gain(3) = maximum-baseline;

%% 4: INITIAL MASS OF WATER
figure()
for i=1:n
    % Initial Volume of Water
    vol_water_initial = m_water_initial_varied(i) / rho_water;
    mass_initial = .160+m_water_initial_varied(i);

    % Initial Volume of Air
    vol_air_initial = vol_bottle - vol_water_initial; % volume of air in
        bottle [m^3]

    % Initial Mass of Rocket
    m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial
        ); % Mass of Air [kg]

    % Variable Declaration for Solver
    vars = [rho_water rho_amb_air c_d C_D A_t A_b p_air_initial
        vol_air_initial...
        y p_ambient vol_bottle l_s g temperature_air_initial m_air_initial...
        R deg2rad(theta) Z_0 X_0 Y_0 W_g_x W_g_y W_g_z W_a_x W_a_y W_a_z];

    % Initial State of Rocket
```

```
    initial_state = [Z_0 X_0 Y_0 mass_initial m_air_initial vol_air_initial V_Z_0 V_X_0 V_Y_0];

    %% ODE SOLVER
    % Set Terminal Condition (i.e. Rocket hit ground)
    terminalCond = odeset('Events', @hitGround);

    % Propagate Flight
    [t,state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state, terminalCond);

    % Plot
    hold on
    plot3(state(:,2),state(:,3),state(:,1));
    grid

    % Save End Points for each trajectory
    X(i) = state(end,2);
    Y(i) = state(end,3);

    % Calculate Force of Thrust
    [F_thrust,phaseChange] = f_thrust(t,state,vars);
end
hold off
xlabel("Downrange Distance [m]");
ylabel("Crossrange Distance [m]");
zlabel("Altitude [m]");
title("Sensitivity Analysis - Initial Water Mass");

poly = polyfit(m_water_initial_varied,X,2);
x2 = polyval(poly,m_water_initial_varied);

figure()
hold on
plot(m_water_initial_varied,X,'ko');
plot(m_water_initial_varied,x2);
[maximum,idx] = max(x2);
plot(m_water_initial_varied(idx),maximum,'ro');
grid
ylabel("Downrange Distance [m]");
xlabel("Initial Water Mass [kg]");
title("Sensitivity Analysis - Initial Water Mass");
hold off

fprintf('Baseline water mass: %.2f kg\n',m_water_initial);
fprintf('Optimum water mass: %.2f kg\n',m_water_initial_varied(idx));
fprintf('Gain in downrange distance: %.2f m\n\n',maximum-baseline);
gain(4) = maximum-baseline;

%% 5: TEMPERATURE
figure()
for i=1:n
    % Initial Volume of Water
    vol_water_initial = m_water_initial / rho_water;
```

```
    mass_initial = .160 + m_water_initial;

    % Initial Volume of Air
    vol_air_initial = vol_bottle - vol_water_initial; % volume of air in bottle [m^3]

    % Initial Mass of Rocket
    m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial_varied(i)); % Mass of Air [kg]

    % Variable Declaration for Solver
    vars = [rho_water rho_amb_air c_d C_D A_t A_b p_air_initial vol_air_initial...
        y p_ambient vol_bottle l_s g temperature_air_initial_varied(i) m_air_initial...
        R deg2rad(theta) Z_0 X_0 Y_0 W_g_x W_g_y W_g_z W_a_x W_a_y W_a_z];

    % Initial State of Rocket
    initial_state = [Z_0 X_0 Y_0 mass_initial m_air_initial vol_air_initial V_Z_0 V_X_0 V_Y_0];

    %% ODE SOLVER
    % Set Terminal Condition (i.e. Rocket hit ground)
    terminalCond = odeset('Events', @hitGround);

    % Propagate Flight
    [t,state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state, terminalCond);

    % Plot
    hold on
    plot3(state(:,2),state(:,3),state(:,1));
    grid

    % Save End Points for each trajectory
    X(i) = state(end,2);
    Y(i) = state(end,3);

    % Calculate Force of Thrust
    [F_thrust,phaseChange] = f_thrust(t,state,vars);
end
hold off
xlabel("Downrange Distance [m]");
ylabel("Crossrange Distance [m]");
zlabel("Altitude [m]");
title("Sensitivity Analysis - Water Temperature");

poly = polyfit(temperature_air_initial_varied,X,2);
x2 = polyval(poly,temperature_air_initial_varied);

figure()
hold on
plot(temperature_air_initial_varied,X,'ko');
plot(temperature_air_initial_varied,x2);
```

```
[maximum,idx] = max(x2);
plot(temperature_air_initial_varied(idx),maximum,'ro');
grid
ylabel("Downrange Distance [m]");
xlabel("Water Temperature [K]");
title("Sensitivity Analysis - Water Temperature");
hold off

fprintf('Baseline water temperature: %.2f K\n',temperature_air_initial);
fprintf('Optimum water temperature: %.2f K\n',temperature_air_initial_varied(idx));
fprintf('Gain in downrange distance: %.2f m\n\n',maximum-baseline);
gain(5) = maximum-baseline;

%% Chosen parameter: initial water mass
m_water_initial = 0.630;

%% Monte Carlo Simulation
n = 500;                                              % Number of Iterations for Simulation

theta_error = 1;                                      % Launch Pad Angle Error [Degrees]
thetaN = ( randn(n,1) * theta_error ) + theta;        % Normal Distribution of Launch Pad Angle [Degrees]

p_error = 0.689;                                      % Pressure in bottle error [Pa]
pN = ( randn(n,1) * p_error ) + p_air_initial;        % Normal Distribution of Pressure in Bottle [Pa]

ground_wind_error = 0.22352;                          % Ground Wind Speed Error [m/s]
ground_windN = ( randn(n,1) * ground_wind_error ) + V_w_g;  % Normal Distribution of Ground Wind Speed Error [m/s]

ground_wind_ang_error = W_g_e;                          % Ground Wind Angle Error [degrees]
ground_wind_angN = ( randn(n,1) * ground_wind_ang_error ) + W_d_g; % Normal Distribution of Ground Wind Angle Error [Degrees]

aloft_wind_error = 0.22352;                           % Aloft Wind Speed Error [m/s]
aloft_windN = ( randn(n,1) * aloft_wind_error ) + V_w_a;  % Normal Distribution of Aloft Wind Speed Error [m/s]

aloft_wind_ang_error = W_a_e;                           % Aloft Wind Angle Error [Degrees]
aloft_wind_angN = ( randn(n,1) * aloft_wind_ang_error ) + W_d_a; % Normal Distribution of Aloft Wind Angle Error [Degrees]

rho_error = 1;                                        % Density of Water Error [kg/m^3]
```

```
rhoN = ( randn(n,1) * aloft_wind_ang_error ) + rho_water; % Normal
    Distribution of Water Density Error [kg/m^3]

mass_error = 5*(10^(-4));                          % Weight Error [kg]
massN = ( randn(n,1) * mass_error ) + mass_initial;     % Normal Distribution
     of Weight Error [kg/m^3]

f1 = figure;
for i=1:n
    %% Set values
    % Adjust Angles to be relative to rocket
    W_d_g = beta - ground_wind_angN(i);
    W_d_a = beta - aloft_wind_angN(i);

    % Calculate Ground Wind in xy-plane
    W_g_x = ground_windN(i)*cos(deg2rad(W_d_g));
    W_g_y = ground_windN(i)*sin(deg2rad(W_d_g));
    W_g_z = 0;

    % Calculate Aloft Wind in xy-plane
    W_a_x = aloft_windN(i)*cos(deg2rad(W_d_a));
    W_a_y = aloft_windN(i)*sin(deg2rad(W_d_a));
    W_a_z = 0;

    vol_water_initial = m_water_initial / rho_water;
    mass_initial = .160 + m_water_initial;

    % Initial Volume of Air
    vol_air_initial = vol_bottle - vol_water_initial; % volume of air in
        bottle [m^3]

    % Initial Mass of Rocket
    m_air_initial = (pN(i)*vol_air_initial)/(R*temperature_air_initial); %
        Mass of Air [kg]

    % Variable Declaration for Solver
    vars = [rhoN(i) rho_amb_air c_d C_D A_t A_b pN(i) vol_air_initial...
        y p_ambient vol_bottle l_s g temperature_air_initial m_air_initial...
        R deg2rad(thetaN(i)) Z_0 X_0 Y_0 W_g_x W_g_y W_g_z W_a_x W_a_y W_a_z];

    % Initial State of Rocket
    initial_state = [Z_0 X_0 Y_0 massN(i) m_air_initial vol_air_initial V_Z_0
        V_X_0 V_Y_0];

    %% ODE SOLVER
    % Set Terminal Condition (i.e. Rocket hit ground)
    terminalCond = odeset('Events', @hitGround);

    % Propagate Flight
    [t,state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state,
        terminalCond);

    % Plot
    hold on
```

```
    plot3(state(:,2),state(:,3),state(:,1));
    grid

    % Save End Points for each trajectory
    X(i) = state(end,2);
    Y(i) = state(end,3);

    % Calculate Force of Thrust
    [F_thrust,phaseChange] = f_thrust(t,state,vars);
end
hold off
xlabel("Downrange Distance [m]");
ylabel("Crossrange Distance [m]");
zlabel("Altitude [m]");
title("Monte Carlo Simulation - 3D Rocket Trajectories Plot");

% Plot Error Elipses
figure;
plot(X,Y,'k.','markersize',6)
grid on;
title("Monte Carlo Simulation - Rocket Trajectories Error Ellipses");
xlabel('x [m]');
ylabel('y [m]');
hold on;

% Calculate covariance matrix
P = cov(X,Y);
mean_x = mean(X);
mean_y = mean(Y);

fprintf('IMPACT LOCATION\n')
fprintf('Mean X Distance: %f\n', mean_x);
fprintf('Mean Y Distance: %f\n', mean_y);
fprintf('Drift Angle: %f\n', rad2deg(atan(mean_y/mean_x)));

% 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')
legend('Trajectory End Points','1\sigma','2\sigma','3\sigma');


%% Function Definitions

% Flight Propagation
```

```
%
% The function that is recursively called by ODE45 and propagates flight
% parameters according to the initial condition passed through by ODE45.
%
% @param t The input time
% @param state_i The input state matrix
% @param vars The constants vector that holds all relevant constants
%
function state = flightProp(t,state_i,vars)
    % Define Variables
    rho_water = vars(1);                     % Density of Water [kg/m^3]
    rho_amb = vars(2);                       % Density of Air [kg/m^3]
    c_d = vars(3);                           % Discharge Coefficient
    C_D = vars(4);                           % Drag Coefficient
    A_t = vars(5);                           % Area of the Throat [m^2]
    A_b = vars(6);                           % Area of the Throat [m^2]
    p_air_initial = vars(7);                 % Initial Pressure of Air [N/m^2]
    vol_air_initial = vars(8);               % Initial Volume of Air [m^3]
    y = vars(9);                             % Specific Heat Ratio of Air
    p_ambient = vars(10);                    % Ambient Pressure [psi]
    p_ambient = (p_ambient*6894.76);         % Convert to [N/m^2]
    vol_bottle = vars(11);                   % Volume of Bottle [m^3]
    l_s = vars(12);                          % Length of Test [m]
    g = vars(13);                            % Gravitational Acceleration Constant
        [m/s^2]
    temperature_air_initial = vars(14);      % Temperature of Air [degrees K]
    m_air_initial = vars(15);                % Initial Mass of the Air [kg]
    R = vars(16);                            % Gas Constant
    launch_angle = vars(17);                 % Angle in Radians of Rocket Initially
    Z_0 = vars(18);                          % Initial Z Position of Rocket
    X_0 = vars(19);                          % Initial X Position of rocket
    Y_0 = vars(20);                          % Initial Y Position of rocket
    W_g_x = vars(21);                        % Ground Wind Speed in X direction
    W_g_y = vars(22);                        % Ground Wind Speed in Y direction
    W_g_z = vars(23);                        % Ground Wind Speed in Z direction
    W_a_x = vars(24);                        % Aloft Wind Speed in X direction
    W_a_y = vars(25);                        % Aloft Wind Speed in Y direction
    W_a_z = vars(26);                        % Aloft Wind Speed in Z direction

    % Define State Vector
    state = zeros(9,1);                  % Vector of 7 elements
    Z = state_i(1);                      % Z Distance of Rocket [m]
    X = state_i(2);                      % X Distance of Rocket [m]
    Y = state_i(3);                      % Y Distance of Rocket [m]
    m_r = state_i(4);                    % Mass of Rocket [kg]
    m_air = state_i(5);                  % Mass of the Air [kg]
    v = state_i(6);                      % Volume of Air [m^3]
    V_z = state_i(7);                    % Velocity in the Z direction [m/s]
    V_x = state_i(8);                    % Velocity in the X direction [m/s]
    V_y = state_i(9);                    % Velocity in the Y direction [m/s]

    % Water Expulsion Phase
    if (v < vol_bottle)
        % Calculate rate of change of Volume over time
```

```
            v_dot = c_d*A_t*sqrt((2/rho_water)*(p_air_initial*((vol_air_initial/v)
                ^y)-p_ambient));
            % Calculate the pressure in the bottle over time
            p_air = p_air_initial*(vol_air_initial./v).^y;
            % Calculate the force of thrust from water expulsion over time
            F_thrust = 2*c_d*A_t*(p_air-p_ambient);
            % Calculate the change in mass of the rocket over time
            m_dot_r = (-c_d)*A_t*sqrt(2*rho_water*(p_air-p_ambient));
            % Calculate the change in mass of the air over time
            m_dot_a = 0;
        else
            % Pressure of Air at end of Water Expulsion Phase
            p_end = p_air_initial*(vol_air_initial/vol_bottle)^y;
            % Temperature of Air at end of Water Expulsion Phase
            t_end = temperature_air_initial*(vol_air_initial/vol_bottle)^(y-1);
            % Calculate Pressure
            p_air = p_end*(m_air./m_air_initial).^y;
        end

        % Gas Expulsion Phase
        if (v >= vol_bottle) && (p_air > p_ambient)
            % Change in Volume
            v_dot = 0;
            % Calculate Critical Pressure
            p_crit = p_air*(2/(y+1))^(y/(y-1));
            % Determine Air Density
            rho_air = m_air / vol_bottle;
            % Determine Gas Temperature
            T = p_air / (rho_air*R);

            % Determine Flow Type
            if (p_crit > p_ambient) % CHOKED FLOW
                % Determine Exit Temperature
                T_e = (2/(y+1))*T;
                % Determine Exit Velocity
                V_e = sqrt(y*R*T_e);
                % Determine Exit Density of Air
                rho_e = p_crit / (R*T_e);
                % Calculate rate of change of mass of air
                m_dot_a = -c_d*rho_e*A_t*V_e;
                % rate of change of mass of rocket
                m_dot_r = m_dot_a;
                % Define Exit Pressure
                p_e = p_crit;
            elseif (p_crit <= p_ambient) % NOT CHOKED FLOW
                % Determine Exit Mach Number
                M_e = sqrt((((p_air/p_ambient)^((y-1)/y))-1)/((y-1)/2));
                % Determine Exit Temperature
                T_e = T / (1+(((y-1)/2)*(M_e^2)));
                % Determine Exit Velocity
                V_e = M_e*sqrt(y*R*T_e);
                % Determine Exit Density of Air
                rho_e = p_ambient * (R*T_e);
                % Calculate rate of change of mass of air
```

```
            m_dot_a = -c_d*rho_e*A_t*V_e;
            % rate of change of mass of rocket
            m_dot_r = m_dot_a;
            % Define Exit Pressure
            p_e = p_ambient;
        end
        % Calculate the force of thrust from air expulsion over time
        F_thrust = (c_d*rho_e*A_t*V_e*V_e) + (p_ambient-p_e)*A_t;
    end

    % Ballistic Phase
    if (v >= vol_bottle) && (p_air < p_ambient)
        % Set everything to zero because we have no prop left
        F_thrust = 0;
        m_dot_r = 0;
        m_dot_a = 0;
        v_dot = 0;
    end

    % Calculate Wind (Scaled by altitude)
    if (Z <= 22)
        W_x = ((Z - 0) / (22 - 0)) * (W_a_x - W_g_x) + W_g_x;
        W_y = ((Z - 0) / (22 - 0)) * (W_a_y - W_g_y) + W_g_y;
        W_z = ((Z - 0) / (22 - 0)) * (W_a_z - W_g_z) + W_g_z;
    else
        W_x = W_a_x;
        W_y = W_a_y;
        W_z = W_a_z;
    end

    % Calculate Relative Velocity of Rocket
    V = sqrt(((V_x-W_x)^2)+((V_z-W_z)^2)+((V_y-W_y)^2));

    % Calculate Angle of Rocket
    % Check to see if still on stand...
    if (sqrt((X-X_0)^2+(Z-Z_0)^2) <= l_s)
        % At launch pad heading if still on stand
        H_x = cos(launch_angle);
        H_z = sin(launch_angle);
        H_y = 0;
    else
        H_x = (V_x - W_x)/V;
        H_z = (V_z - W_z)/V;
        H_y = (V_y - W_y)/V;
    end

    % Calculate Drag on Rocket
    drag = (1/2)*rho_amb*(V^2)*C_D*A_b;

    % Calculate Acceleration
    accel_x = (F_thrust*H_x - drag*H_x + 0)/m_r;               % Acceleration
        in X Direction
    accel_y = (F_thrust*H_y - drag*H_y + 0)/m_r;               % Acceleration
        in Y Direction
```

```
    accel_z = (F_thrust*H_z - drag*H_z - (m_r*g))/m_r;          % Acceleration in Z Direction

    % Output State
    state(1) = V_z;                                        % Velocity in Z direction
    state(2) = V_x;                                        % Velocity in X Direction
    state(3) = V_y;                                        % Velocity in Y Direction
    state(4) = m_dot_r;                                    % Rate of Change of Mass of Rocket
    state(5) = m_dot_a;                                    % Rate of Change of Mass of Air
    state(6) = v_dot;                                      % Rate of Change of Volume of Air
    state(7) = accel_z;                                    % Rate of Change of Velocity in the Z Direction
    state(8) = accel_x;                                    % Rate of Change of Velocity in the X Direction
    state(9) = accel_y;                                    % Rate of Change of Velocity in the Y Direction
end

% F_thrust
%
% The function takes the ODE45 Flight Propagation state matrix that was output
% and calculates the force of thrust for plotting.
%
% @param state The state matrix from the ODE45 Flight Propagation call
% @param vars The constants vector that holds all relevant constants
%
function [out,phaseChange] = f_thrust(t,state,vars)
    % Define Variables
    c_d = vars(3);                    % Discharge Coefficient
    A_t = vars(5);                    % Area of the Throat [m^2]
    p_air_initial = vars(7);          % Initial Pressure of Air [N/m^2]
    vol_air_initial = vars(8);        % Initial Volume of Air [m^3]
    y = vars(9);                      % Specific Heat Ratio of Air
    p_ambient = vars(10);             % Ambient Pressure [psi]
    p_ambient = (p_ambient*6894.76); % Convert to [N/m^2]
    vol_bottle = vars(11);            % Volume of Bottle [m^3]
    m_air_initial = vars(15);         % Initial Mass of the Air [kg]
    R = vars(16);                     % Gas Constant

    % Variables for Phase Change logging
    phaseChange = [];
    phaseFlag = true;

    % Iterate through state matrix data
    for i = 1:size(state,1)
        % Get important values from state matrix
        m_air = state(i,5);               % Mass of the Air [kg]
        v = state(i,6);                   % Volume of Air [m^3]

        % Water Expulsion Phase
        if (v < vol_bottle)
```

```
            if (phaseFlag == true)
                phaseChange(1) = t(i+1);
                phaseFlag = false;
            end
            % Calculate the pressure in the bottle over time
            p_air = p_air_initial*(vol_air_initial./v).^y;
            % Calculate the force of thrust from water expulsion over time
            F_thrust = 2*c_d*A_t*(p_air-p_ambient);
        else
            % Pressure of Air at end of Water Expulsion Phase
            p_end = p_air_initial*(vol_air_initial/vol_bottle)^y;
            % Calculate Pressure
            p_air = p_end*(m_air./m_air_initial).^y;
        end

        % Gas Expulsion Phase
        if (v > vol_bottle) && (p_air > p_ambient)
            if (phaseFlag == false)
                phaseChange(2) = t(i+1);
                phaseFlag = true;
            end
            % Calculate Critical Pressure
            p_crit = p_air*(2/(y+1))^(y/(y-1));
            % Determine Air Density
            rho_air = m_air / vol_bottle;
            % Determine Gas Temperature
            T = p_air / (rho_air*R);

            % Determine Flow Type
            if (p_crit > p_ambient) % CHOKED FLOW
                % Determine Exit Temperature
                T_e = (2/(y+1))*T;
                % Determine Exit Velocity
                V_e = sqrt(y*R*T_e);
                % Determine Exit Density of Air
                rho_e = p_crit / (R*T_e);
                % Define Exit Pressure
                p_e = p_crit;
            elseif (p_crit <= p_ambient) % NOT CHOKED FLOW
                % Determine Exit Mach Number
                M_e = sqrt((((p_air/p_ambient)^((y-1)/y))-1)/((y-1)/2));
                % Determine Exit Temperature
                T_e = T / (1+(((y-1)/2)*(M_e^2)));
                % Determine Exit Velocity
                V_e = M_e*sqrt(y*R*T_e);
                % Determine Exit Density of Air
                rho_e = p_ambient * (R*T_e);
                % Define Exit Pressure
                p_e = p_ambient;
            end
            % Calculate the force of thrust from air expulsion over time
            F_thrust = (c_d*rho_e*A_t*V_e*V_e) + (p_ambient-p_e)*A_t;
        end

```

```
        % Ballistic Phase
        if (v > vol_bottle) && (p_air < p_ambient)
            if (phaseFlag == true)
                phaseChange(3) = t(i+1);
                phaseFlag = false;
            end
            % Set everything to zero because we have no prop left
            F_thrust = 0;
        end
        % Output the Force of Thrust
        out(i) = F_thrust;
    end
end

% hitGround
%
% The conditional stop function that checks the state matrix such that if
% the condition Z < 0ft is met the integration is terminated (i.e. hit the
    ground)
%
% @param t The input time
% @param state The input state matrix
%
function [value, isterminal, direction] = hitGround(t, state)
    value      = (state(1) < 0);
    isterminal = 1;   % Stop the integration
    direction  = 0;
end
```