

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.

$$\Sigma F = \dot{m}a = T - D - F_g = (T - D)\hat{i} + (T - D)\hat{j} + (T - D - mg)\hat{k} \quad (1)$$

Within this equation, T represents thrust, D represents drag, F_g represents gravity, and the vector notation corresponds to the xyz coordinate system (\hat{i} for downrange, \hat{j} 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

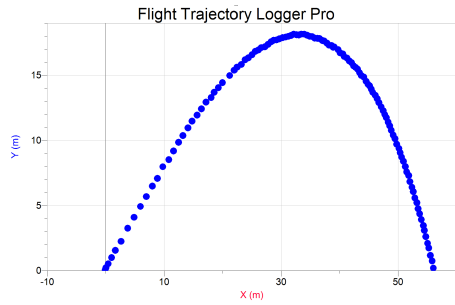


Fig. 1 Flight Trajectory Video Capture

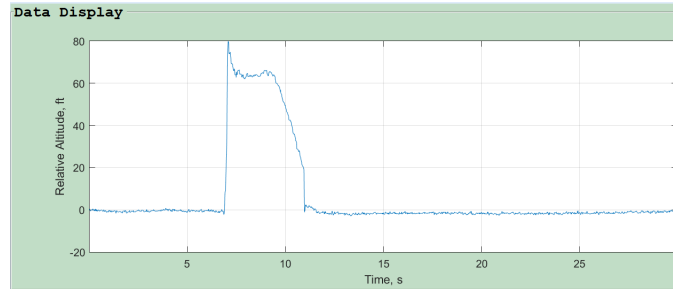


Fig. 2 Altitude vs Time MATLAB GUI

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

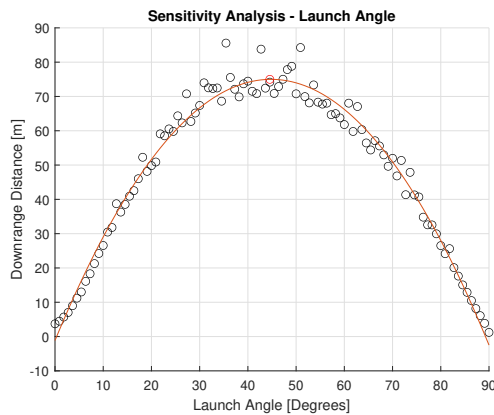


Fig. 3 Launch Angle Sensitivity Analysis

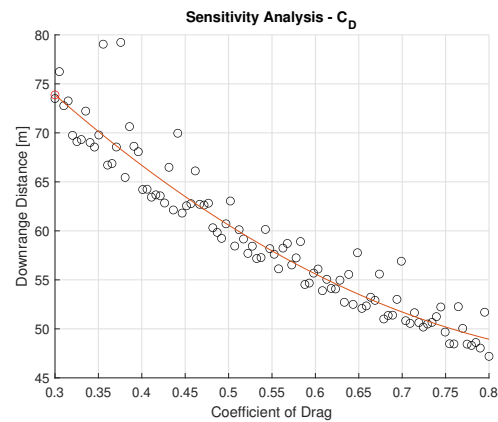


Fig. 4 Coefficient of Drag Sensitivity Analysis

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 $900 \frac{\text{kg}}{\text{m}^3}$ to $1100 \frac{\text{kg}}{\text{m}^3}$. 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.

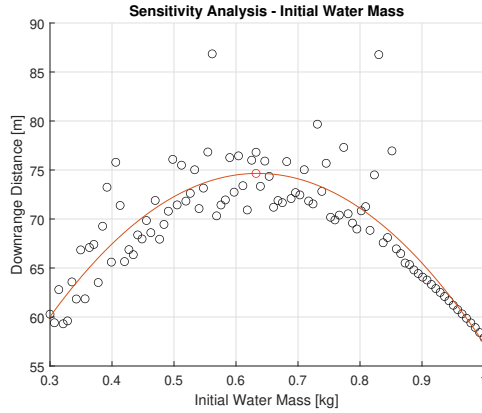


Fig. 5 Initial Mass of Water Sensitivity Analysis

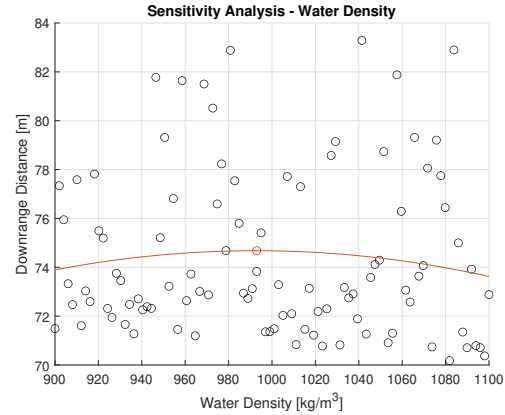


Fig. 6 Density of Water Sensitivity Analysis

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.

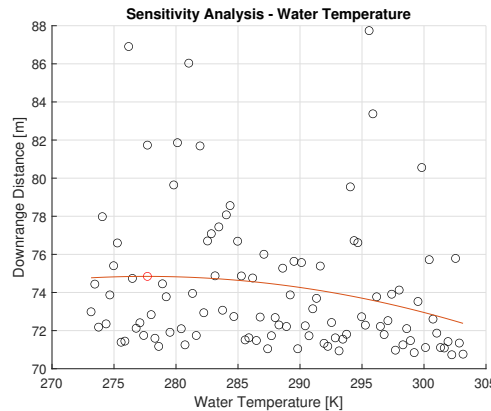


Fig. 7 Temperature of Water

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° . As a protractor has a human error of $\pm 1^\circ$, 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

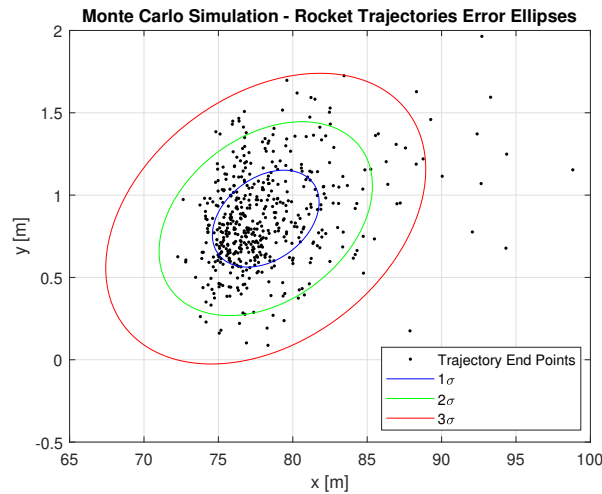


Fig. 8 Flight Predictions - Monte Carlo Simulation

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 $\pm 1^\circ$
- Mass $\pm 0.5g$
- Pressure in Bottle $\pm 0.1psi$ or $\pm 689Pa$
- Coefficient of Drag ± 0.05
- Wind $\pm 0.5 mph$ or $\pm 10\%$ of max speed
- Wind Direction $\pm 22.5^\circ$
- 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

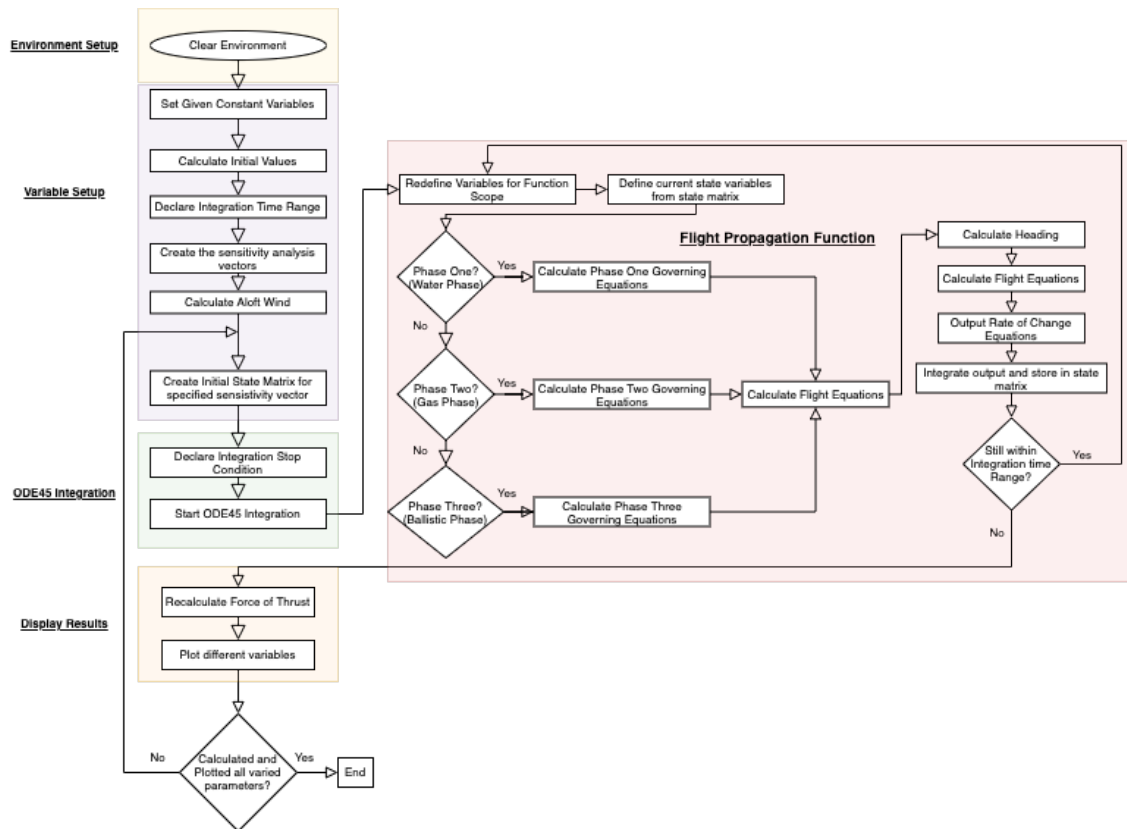


Fig. 9 MATLAB Code Flow Chart

B. MATLAB Code

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %
3  %                               Lab #2 – ASEN 2004
4  %                               Bottle Rocket
5  %                               V 2.0
6  %
7  %
8  %                               04/18/2020
9  %                               Configured for LA Optimized Rocket
10 %
11 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12
13 %%% Set Environment
14 clear
15 clc
16
17 %%% Set Variables
18
19 % Set Launch Datasheet Vars
20 C_D = 0.30; % Drag Coefficient ----
21 temperature_air = 63; % Initial Temperature of Air [degrees F] ----
22 p_air_gage_initial = 40; % Gage pressure of inside of bottle [psi] ----
23 m_bottle = 0.160; % Mass of the Bottle [kg]
24 m_water_initial = 0.600; % Initial Mass of the Water [kg]
25 mass_initial = 0.760; % Total Initial Mass [kg] -----
26
27 theta = 45; % Initial Vertical Angle of Rocket [Degrees]
28 %----
29 beta = 40; % Initial Horizontal Angle of Rocket from
30 % north [Degrees] ----
31
32 V_w_g = 0; % Ground Wind Speed [mph] !! MAX WIND SPEED OF
33 % 10 [mph] !! ----
34 V_w_a = 3; % Aloft Wind Speed [mph] !! MAX WIND SPEED OF
35 % 10 [mph] !! ----
36 W_d_g = 0; % Ground Wind Direction relative to rocket
37 % nose at 0 degrees [degrees] ----
38 W_g_e = 11.25; % Ground Wind Direction Error [Degrees]
39 W_d_a = 45; % Aloft Wind Direction relative to rocket nose
40 % at 0 degrees [degrees] ----
41 W_a_e = 11.25; % Aloft Wind Direction Error [Degrees]
42
43 % Constant Variables
44 R = 287; % Gas Constant [J*kg^-1*K^-1]
45 g = 9.81; % Acceleration of Gravity [m/s^2]
46 c_d = 0.8; % Discharge Coefficient
47 y = 1.4; % Specific Heat Ratio of Air [Unitless?]
48 temperature_air_initial = ((temperature_air - 32) * 5/9) + 273.15; %
49 % Initial Temperature of Air [degrees K]
50
51 % Measurements
52 d_throat = 2.1; % Diameter of throat [cm]
53 d_bottle = 10.5; % Diameter of bottle [cm]

```

```

47
48 % Pressure
49 p_ambient = 12.1; % Ambient pressure [psi]
50
51 % Volume
52 % vol_water_initial = 0.001; % Volume of water in bottle [m^3]
53 vol_bottle = 0.002; % Vol of bottle [m^3]
54
55 % Density
56 rho_water = 1000; % Density of Water [kg/m^3]
57 rho_amb_air = 0.961; % Density of Ambient Air [kg/m^3]
58
59 % Flight Variables
60 V_X_0 = 0; % Initial Velocity of Rocket in X Direction [m
    / s]
61 V_Y_0 = 0; % Initial Velocity of Rocket in Y Direction [m
    / s]
62 V_Z_0 = 0; % Initial Velocity of Rocket in Z Direction [m
    / s]
63
64 X_0 = 0; % Initial X Distance of Rocket [m]
65 Y_0 = 0; % Initial Y Distance of Rocket [m]
66 Z_0 = 0.25; % Initial Z Distance of Rocket [m]
67
68 l_s = 0.5; % Initial Length of Test Stand [m]
69
70 V_w_g = V_w_g*0.44704; % Ground Wind Speed [m/s] !! MAX WIND SPEED OF
    4.47 [m/s] !!
71 V_w_a = V_w_a*0.44704; % Aloft Wind Speed [m/s] !! MAX WIND SPEED OF
    4.47 [m/s] !!
72
73 %% Calculate Initial Values
74
75 % Initial Angle of Rocket in Radians
76 theta_rad = deg2rad(theta);
77
78 % Pressure inside Rocket
79 p_air_initial = (p_air_gage_initial*6894.76) + (p_ambient*6894.76); % [N/m^2]
80
81 % Initial Volume of Water
82 vol_water_initial = m_water_initial / rho_water;
83
84 % Initial Volume of Air
85 vol_air_initial = vol_bottle - vol_water_initial; % volume of air in bottle [m
    ^3]
86
87 % Initial Mass of Rocket
88 % m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial);
    % Mass of Air [kg]
89
90 % m_water_initial = rho_water*(vol_bottle - vol_air_initial); % Mass of Water
    [kg]
91 % m_water_initial = 1.001; % [kg]
92

```



```

93  % m_bottle = 0.15; % Mass of Bottle [kg]
94  % m_bottle = 0.128; % [kg]
95
96  % mass_initial = m_bottle + m_water_initial + m_air_initial; % Total Mass [kg
    ]
97  % mass_initial = 1.129; % Total Initial Mass [kg] -----
98
99  % Area of Throat
100 A_t = pi*((d_throat*0.01)/2)^2; % m^2
101
102 % Area of Bottle
103 A_b = pi*((d_bottle*0.01)/2)^2; % m^2
104
105 % Time Range for Solver; 0-5 Seconds
106 tspan = [0 10];
107
108 %% Sensitivity Analysis
109 % Vary: C_D, Mass of water, density of water, temperature of water, launch
110 % pad angle
111 n = 100; % Number of Iterations
    for Simulation
112
113     theta_varied = linspace(0,90); % Varied Launch Pad
        Angle [Degrees]
114
115     C_D_varied = linspace(0.3,0.8); % Varied Coefficient
        of Drag [unitless]
116
117     rho_water_varied = linspace(900,1100); % Varied Water Density
        [kg/m^3]
118
119     m_water_initial_varied = linspace(0.300,1.000); % Varied Initial Mass
        of the Water [kg]
120
121     temperature_air_initial_varied = linspace(0,30); % Varied Initial
        Temperature of Air/Water [C]
122     temperature_air_initial_varied = temperature_air_initial_varied + 273.15;
        % Varied Initial Temperature of Air/Water [K]
123
124 %% Set values
125 % Adjust Angles to be relative to rocket
126 W_d_g = beta - W_d_g;
127 W_d_a = beta - W_d_a;
128
129 % Calculate Ground Wind in xy-plane
130 W_g_x = V_w_g*cos(deg2rad(W_d_g));
131 W_g_y = V_w_g*sin(deg2rad(W_d_g));
132 W_g_z = 0;
133
134 % Calculate Aloft Wind in xy-plane
135 W_a_x = V_w_a*cos(deg2rad(W_d_a));
136 W_a_y = V_w_a*sin(deg2rad(W_d_a));
137 W_a_z = 0;
138

```

```

139 %% BASELINE
140 % Initial Volume of Water
141 vol_water_initial = m_water_initial / rho_water;
142
143 % Initial Volume of Air
144 vol_air_initial = vol_bottle - vol_water_initial; % volume of air in bottle [m
    ^3]
145
146 % Initial Mass of Rocket
147 m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial); %
    Mass of Air [kg]
148
149 % Variable Declaration for Solver
150 vars = [rho_water rho_amb_air c_d C_D A_t A_b p_air_initial vol_air_initial...
151         y p_ambient vol_bottle l_s g temperature_air_initial m_air_initial...
152         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];
153
154 % Initial State of Rocket
155 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];
156
157 % Set Terminal Condition (i.e. Rocket hit ground)
158 terminalCond = odeset('Events', @hitGround);
159
160 % Propagate Flight
161 [t, state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state,
    terminalCond);
162
163 baseline = state(end,2);
164
165 gain = zeros(1,5);
166
167 %% 1: LAUNCH ANGLE
168 figure()
169 for i=1:n
170     % Initial Volume of Water
171     vol_water_initial = m_water_initial / rho_water;
172     mass_initial = .160 + m_water_initial;
173
174     % Initial Volume of Air
175     vol_air_initial = vol_bottle - vol_water_initial; % volume of air in
        bottle [m^3]
176
177     % Initial Mass of Rocket
178     m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial
        ); % Mass of Air [kg]
179
180     % Variable Declaration for Solver
181     vars = [rho_water rho_amb_air c_d C_D A_t A_b p_air_initial
        vol_air_initial...
182             y p_ambient vol_bottle l_s g temperature_air_initial m_air_initial...
183             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];
184

```

```

185 % Initial State of Rocket
186 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];
187
188 %%% ODE SOLVER
189 % Set Terminal Condition (i.e. Rocket hit ground)
190 terminalCond = odeset('Events', @hitGround);
191
192 % Propagate Flight
193 [t, state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state,
    terminalCond);
194
195 % Plot
196 hold on
197 plot3(state(:,2), state(:,3), state(:,1));
198 grid
199
200 % Save End Points for each trajectory
201 X(i) = state(end,2);
202 Y(i) = state(end,3);
203
204 % Calculate Force of Thrust
205 [F_thrust, phaseChange] = f_thrust(t, state, vars);
206 end
207 hold off
208 xlabel("Downrange Distance [m]");
209 ylabel("Crossrange Distance [m]");
210 zlabel("Altitude [m]");
211 title("Sensitivity Analysis - Launch Angle");
212
213 poly = polyfit(theta_varied, X, 2);
214 x2 = polyval(poly, theta_varied);
215
216 figure()
217 hold on
218 plot(theta_varied, X, 'ko');
219 plot(theta_varied, x2);
220 [maximum, idx] = max(x2);
221 plot(theta_varied(idx), maximum, 'ro');
222 grid
223 ylabel("Downrange Distance [m]");
224 xlabel("Launch Angle [Degrees]");
225 title("Sensitivity Analysis - Launch Angle");
226 hold off
227
228 fprintf('Baseline launch angle: %.2f degrees\n', theta);
229 fprintf('Optimum launch angle: %.2f degrees\n', theta_varied(idx));
230 fprintf('Gain in downrange distance: %.2f m\n\n', maximum-baseline);
231 gain(1) = maximum-baseline;
232
233
234 %%% 2: COEFFICIENT OF DRAG
235 figure()
236 for i=1:n

```

```

237 % Initial Volume of Water
238 vol_water_initial = m_water_initial / rho_water;
239 mass_initial = .160 + m_water_initial;
240
241 % Initial Volume of Air
242 vol_air_initial = vol_bottle - vol_water_initial; % volume of air in
    bottle [m^3]
243
244 % Initial Mass of Rocket
245 m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial
    ); % Mass of Air [kg]
246
247 % Variable Declaration for Solver
248 vars = [rho_water rho_amb_air c_d C_D_varied(i) A_t A_b p_air_initial
    vol_air_initial ...
249     y p_ambient vol_bottle l_s g temperature_air_initial m_air_initial ...
250     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];
251
252 % Initial State of Rocket
253 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];
254
255 %%% ODE SOLVER
256 % Set Terminal Condition (i.e. Rocket hit ground)
257 terminalCond = odeset('Events', @hitGround);
258
259 % Propagate Flight
260 [t, state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state,
    terminalCond);
261
262 % Plot
263 hold on
264 plot3(state(:,2), state(:,3), state(:,1));
265 grid
266
267 % Save End Points for each trajectory
268 X(i) = state(end,2);
269 Y(i) = state(end,3);
270
271 % Calculate Force of Thrust
272 [F_thrust, phaseChange] = f_thrust(t, state, vars);
273 end
274 hold off
275 xlabel("Downrange Distance [m]");
276 ylabel("Crossrange Distance [m]");
277 zlabel("Altitude [m]");
278 title("Sensitivity Analysis - C_D");
279
280 poly = polyfit(C_D_varied, X, 2);
281 x2 = polyval(poly, C_D_varied);
282
283 figure()
284 hold on
285 plot(C_D_varied, X, 'ko');

```

```

286 plot(C_D_varied,x2);
287 [maximum,idx] = max(x2);
288 plot(C_D_varied(idx),maximum,'ro');
289 grid
290 ylabel("Downrange Distance [m]");
291 xlabel("Coefficient of Drag");
292 title("Sensitivity Analysis - C_D");
293 hold off
294
295 fprintf('Baseline coefficient of drag: %.2f\n',C_D);
296 fprintf('Optimum coefficient of drag: %.2f\n',C_D_varied(idx));
297 fprintf('Gain in downrange distance: %.2f m\n\n',maximum-baseline);
298 gain(2) = maximum-baseline;
299
300 %% 3: DENSITY OF WATER
301 figure()
302 for i=1:n
303     % Initial Volume of Water
304     vol_water_initial = m_water_initial / rho_water_varied(i);
305     mass_initial = .160 + m_water_initial;
306
307     % Initial Volume of Air
308     vol_air_initial = vol_bottle - vol_water_initial; % volume of air in
        bottle [m^3]
309
310     % Initial Mass of Rocket
311     m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial
        ); % Mass of Air [kg]
312
313     % Variable Declaration for Solver
314     vars = [rho_water_varied(i) rho_amb_air c_d C_D A_t A_b p_air_initial
        vol_air_initial...
315         y p_ambient vol_bottle l_s g temperature_air_initial m_air_initial...
316         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];
317
318     % Initial State of Rocket
319     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];
320
321     %% ODE SOLVER
322     % Set Terminal Condition (i.e. Rocket hit ground)
323     terminalCond = odeset('Events', @hitGround);
324
325     % Propagate Flight
326     [t,state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state,
        terminalCond);
327
328     % Plot
329     hold on
330     plot3(state(:,2),state(:,3),state(:,1));
331     grid
332
333     % Save End Points for each trajectory
334     X(i) = state(end,2);

```

```

335     Y(i) = state(end,3);
336
337     % Calculate Force of Thrust
338     [F_thrust, phaseChange] = f_thrust(t, state, vars);
339 end
340 hold off
341 xlabel("Downrange Distance [m]");
342 ylabel("Crossrange Distance [m]");
343 zlabel("Altitude [m]");
344 title("Sensitivity Analysis - Water Density");
345
346 poly = polyfit(rho_water_varied, X, 2);
347 x2 = polyval(poly, rho_water_varied);
348
349 figure()
350 hold on
351 plot(rho_water_varied, X, 'ko');
352 plot(rho_water_varied, x2);
353 [maximum, idx] = max(x2);
354 plot(rho_water_varied(idx), maximum, 'ro');
355 grid
356 ylabel("Downrange Distance [m]");
357 xlabel("Water Density [kg/m^3]");
358 title("Sensitivity Analysis - Water Density");
359 hold off
360
361 fprintf('Baseline water density: %.2f kg/m^3\n', rho_water);
362 fprintf('Optimum water density: %.2f kg/m^3\n', rho_water_varied(idx));
363 fprintf('Gain in downrange distance: %.2f m\n\n', maximum-baseline);
364 gain(3) = maximum-baseline;
365
366 %% 4: INITIAL MASS OF WATER
367 figure()
368 for i=1:n
369     % Initial Volume of Water
370     vol_water_initial = m_water_initial_varied(i) / rho_water;
371     mass_initial = .160 + m_water_initial_varied(i);
372
373     % Initial Volume of Air
374     vol_air_initial = vol_bottle - vol_water_initial; % volume of air in
        bottle [m^3]
375
376     % Initial Mass of Rocket
377     m_air_initial = (p_air_initial*vol_air_initial)/(R*temperature_air_initial
        ); % Mass of Air [kg]
378
379     % Variable Declaration for Solver
380     vars = [rho_water rho_amb_air c_d C_D A_t A_b p_air_initial
        vol_air_initial...
381         y p_ambient vol_bottle l_s g temperature_air_initial m_air_initial...
382         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];
383
384     % Initial State of Rocket

```

```

385     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];
386
387     %% ODE SOLVER
388     % Set Terminal Condition (i.e. Rocket hit ground)
389     terminalCond = odeset('Events', @hitGround);
390
391     % Propagate Flight
392     [t, state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state,
                      terminalCond);
393
394     % Plot
395     hold on
396     plot3(state(:,2), state(:,3), state(:,1));
397     grid
398
399     % Save End Points for each trajectory
400     X(i) = state(end,2);
401     Y(i) = state(end,3);
402
403     % Calculate Force of Thrust
404     [F_thrust, phaseChange] = f_thrust(t, state, vars);
405 end
406 hold off
407 xlabel("Downrange Distance [m]");
408 ylabel("Crossrange Distance [m]");
409 zlabel("Altitude [m]");
410 title("Sensitivity Analysis - Initial Water Mass");
411
412 poly = polyfit(m_water_initial_varied, X, 2);
413 x2 = polyval(poly, m_water_initial_varied);
414
415 figure()
416 hold on
417 plot(m_water_initial_varied, X, 'ko');
418 plot(m_water_initial_varied, x2);
419 [maximum, idx] = max(x2);
420 plot(m_water_initial_varied(idx), maximum, 'ro');
421 grid
422 ylabel("Downrange Distance [m]");
423 xlabel("Initial Water Mass [kg]");
424 title("Sensitivity Analysis - Initial Water Mass");
425 hold off
426
427 fprintf('Baseline water mass: %.2f kg\n', m_water_initial);
428 fprintf('Optimum water mass: %.2f kg\n', m_water_initial_varied(idx));
429 fprintf('Gain in downrange distance: %.2f m\n\n', maximum-baseline);
430 gain(4) = maximum-baseline;
431
432 %% 5: TEMPERATURE
433 figure()
434 for i=1:n
435     % Initial Volume of Water
436     vol_water_initial = m_water_initial / rho_water;

```

```

437     mass_initial = .160 + m_water_initial;
438
439     % Initial Volume of Air
440     vol_air_initial = vol_bottle - vol_water_initial; % volume of air in
         bottle [m^3]
441
442     % Initial Mass of Rocket
443     m_air_initial = (p_air_initial*vol_air_initial)/(R*
         temperature_air_initial_varied(i)); % Mass of Air [kg]
444
445     % Variable Declaration for Solver
446     vars = [rho_water rho_amb_air c_d C_D A_t A_b p_air_initial
         vol_air_initial ...
447         y p_ambient vol_bottle l_s g temperature_air_initial_varied(i)
         m_air_initial ...
448         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];
449
450     % Initial State of Rocket
451     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];
452
453     %% ODE SOLVER
454     % Set Terminal Condition (i.e. Rocket hit ground)
455     terminalCond = odeset('Events', @hitGround);
456
457     % Propagate Flight
458     [t, state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state,
         terminalCond);
459
460     % Plot
461     hold on
462     plot3(state(:,2), state(:,3), state(:,1));
463     grid
464
465     % Save End Points for each trajectory
466     X(i) = state(end,2);
467     Y(i) = state(end,3);
468
469     % Calculate Force of Thrust
470     [F_thrust, phaseChange] = f_thrust(t, state, vars);
471 end
472 hold off
473 xlabel("Downrange Distance [m]");
474 ylabel("Crossrange Distance [m]");
475 zlabel("Altitude [m]");
476 title("Sensitivity Analysis - Water Temperature");
477
478 poly = polyfit(temperature_air_initial_varied, X, 2);
479 x2 = polyval(poly, temperature_air_initial_varied);
480
481 figure()
482 hold on
483 plot(temperature_air_initial_varied, X, 'ko');
484 plot(temperature_air_initial_varied, x2);

```



```

485 [maximum,idx] = max(x2);
486 plot(temperature_air_initial_varied(idx),maximum,'ro');
487 grid
488 ylabel("Downrange Distance [m]");
489 xlabel("Water Temperature [K]");
490 title("Sensitivity Analysis - Water Temperature");
491 hold off
492
493 fprintf('Baseline water temperature: %.2f K\n',temperature_air_initial);
494 fprintf('Optimum water temperature: %.2f K\n',temperature_air_initial_varied(
    idx));
495 fprintf('Gain in downrange distance: %.2f m\n\n',maximum-baseline);
496 gain(5) = maximum-baseline;
497
498 %% Chosen parameter: initial water mass
499 m_water_initial = 0.630;
500
501 %% Monte Carlo Simulation
502 n = 500; % Number of Iterations
    for Simulation
503
504     theta_error = 1; % Launch Pad Angle
        Error [Degrees]
505     thetaN = ( randn(n,1) * theta_error ) + theta; % Normal Distribution
        of Launch Pad Angle [Degrees]
506
507     p_error = 0.689; % Pressure in bottle
        error [Pa]
508     pN = ( randn(n,1) * p_error ) + p_air_initial; % Normal Distribution
        of Pressure in Bottle [Pa]
509
510     ground_wind_error = 0.22352; % Ground Wind Speed
        Error [m/s]
511     ground_windN = ( randn(n,1) * ground_wind_error ) + V_w_g; % Normal
        Distribution of Ground Wind Speed Error [m/s]
512
513     ground_wind_ang_error = W_g_e; % Ground Wind Angle
        Error [degrees]
514     ground_wind_angN = ( randn(n,1) * ground_wind_ang_error ) + W_d_g; % Normal
        Distribution of Ground Wind Angle Error [Degrees]
515
516     aloft_wind_error = 0.22352; % Aloft Wind Speed
        Error [m/s]
517     aloft_windN = ( randn(n,1) * aloft_wind_error ) + V_w_a; % Normal
        Distribution of Aloft Wind Speed Error [m/s]
518
519     aloft_wind_ang_error = W_a_e; % Aloft Wind Angle
        Error [Degrees]
520     aloft_wind_angN = ( randn(n,1) * aloft_wind_ang_error ) + W_d_a; % Normal
        Distribution of Aloft Wind Angle Error [Degrees]
521
522     rho_error = 1; % Density of Water
        Error [kg/m^3]

```

```

523 rhoN = ( randn(n,1) * aloft_wind_ang_error ) + rho_water; % Normal
      Distribution of Water Density Error [kg/m^3]
524
525 mass_error = 5*(10^(-4)); % Weight Error [kg]
526 massN = ( randn(n,1) * mass_error ) + mass_initial; % Normal Distribution
      of Weight Error [kg/m^3]
527
528 f1 = figure;
529 for i=1:n
530     %% Set values
531     % Adjust Angles to be relative to rocket
532     W_d_g = beta - ground_wind_angN(i);
533     W_d_a = beta - aloft_wind_angN(i);
534
535     % Calculate Ground Wind in xy-plane
536     W_g_x = ground_windN(i)*cos(deg2rad(W_d_g));
537     W_g_y = ground_windN(i)*sin(deg2rad(W_d_g));
538     W_g_z = 0;
539
540     % Calculate Aloft Wind in xy-plane
541     W_a_x = aloft_windN(i)*cos(deg2rad(W_d_a));
542     W_a_y = aloft_windN(i)*sin(deg2rad(W_d_a));
543     W_a_z = 0;
544
545     vol_water_initial = m_water_initial / rho_water;
546     mass_initial = .160 + m_water_initial;
547
548     % Initial Volume of Air
549     vol_air_initial = vol_bottle - vol_water_initial; % volume of air in
      bottle [m^3]
550
551     % Initial Mass of Rocket
552     m_air_initial = (pN(i)*vol_air_initial)/(R*temperature_air_initial); %
      Mass of Air [kg]
553
554     % Variable Declaration for Solver
555     vars = [rhoN(i) rho_amb_air c_d C_D A_t A_b pN(i) vol_air_initial...
556            y p_ambient vol_bottle l_s g temperature_air_initial m_air_initial...
557            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];
558
559     % Initial State of Rocket
560     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];
561
562     %% ODE SOLVER
563     % Set Terminal Condition (i.e. Rocket hit ground)
564     terminalCond = odeset('Events', @hitGround);
565
566     % Propagate Flight
567     [t, state] = ode45(@(t,s) flightProp(t,s,vars), tspan, initial_state,
      terminalCond);
568
569     % Plot
570     hold on

```

```

571     plot3(state(:,2),state(:,3),state(:,1));
572     grid
573
574     % Save End Points for each trajectory
575     X(i) = state(end,2);
576     Y(i) = state(end,3);
577
578     % Calculate Force of Thrust
579     [F_thrust,phaseChange] = f_thrust(t,state,vars);
580 end
581 hold off
582 xlabel("Downrange Distance [m]");
583 ylabel("Crossrange Distance [m]");
584 zlabel("Altitude [m]");
585 title("Monte Carlo Simulation - 3D Rocket Trajectories Plot");
586
587 % Plot Error Ellipses
588 figure;
589 plot(X,Y,'k.','markersize',6)
590 grid on;
591 title("Monte Carlo Simulation - Rocket Trajectories Error Ellipses");
592 xlabel('x [m]');
593 ylabel('y [m]');
594 hold on;
595
596 % Calculate covariance matrix
597 P = cov(X,Y);
598 mean_x = mean(X);
599 mean_y = mean(Y);
600
601 fprintf('IMPACT LOCATION\n')
602 fprintf('Mean X Distance: %f\n', mean_x);
603 fprintf('Mean Y Distance: %f\n', mean_y);
604 fprintf('Drift Angle: %f\n', rad2deg(atan(mean_y/mean_x)));
605
606 % Calculate the define the error ellipses
607 n=100; % Number of points around ellipse
608 p=0:pi/n:2*pi; % angles around a circle
609
610 [eigvec,eigval] = eig(P); % Compute eigen-stuff
611 xy_vect = [cos(p'),sin(p')] * sqrt(eigval) * eigvec'; % Transformation
612 x_vect = xy_vect(:,1);
613 y_vect = xy_vect(:,2);
614
615 % Plot the error ellipses overlaid on the same figure
616 plot(1*x_vect+mean_x, 1*y_vect+mean_y, 'b')
617 plot(2*x_vect+mean_x, 2*y_vect+mean_y, 'g')
618 plot(3*x_vect+mean_x, 3*y_vect+mean_y, 'r')
619 legend('Trajectory End Points','1\sigma','2\sigma','3\sigma');
620
621
622 %%% Function Definitions
623
624 % Flight Propagation

```

```

625 %
626 % The function that is recursively called by ODE45 and propagates flight
627 % parameters according to the initial condition passed through by ODE45.
628 %
629 % @param t The input time
630 % @param state_i The input state matrix
631 % @param vars The constants vector that holds all relevant constants
632 %
633 function state = flightProp(t, state_i, vars)
634     % Define Variables
635     rho_water = vars(1); % Density of Water [kg/m^3]
636     rho_amb = vars(2); % Density of Air [kg/m^3]
637     c_d = vars(3); % Discharge Coefficient
638     C_D = vars(4); % Drag Coefficient
639     A_t = vars(5); % Area of the Throat [m^2]
640     A_b = vars(6); % Area of the Throat [m^2]
641     p_air_initial = vars(7); % Initial Pressure of Air [N/m^2]
642     vol_air_initial = vars(8); % Initial Volume of Air [m^3]
643     y = vars(9); % Specific Heat Ratio of Air
644     p_ambient = vars(10); % Ambient Pressure [psi]
645     p_ambient = (p_ambient*6894.76); % Convert to [N/m^2]
646     vol_bottle = vars(11); % Volume of Bottle [m^3]
647     l_s = vars(12); % Length of Test [m]
648     g = vars(13); % Gravitational Acceleration Constant
649     % [m/s^2]
650     temperature_air_initial = vars(14); % Temperature of Air [degrees K]
651     m_air_initial = vars(15); % Initial Mass of the Air [kg]
652     R = vars(16); % Gas Constant
653     launch_angle = vars(17); % Angle in Radians of Rocket Initially
654     Z_0 = vars(18); % Initial Z Position of Rocket
655     X_0 = vars(19); % Initial X Position of rocket
656     Y_0 = vars(20); % Initial Y Position of rocket
657     W_g_x = vars(21); % Ground Wind Speed in X direction
658     W_g_y = vars(22); % Ground Wind Speed in Y direction
659     W_g_z = vars(23); % Ground Wind Speed in Z direction
660     W_a_x = vars(24); % Aloft Wind Speed in X direction
661     W_a_y = vars(25); % Aloft Wind Speed in Y direction
662     W_a_z = vars(26); % Aloft Wind Speed in Z direction
663
664     % Define State Vector
665     state = zeros(9,1); % Vector of 7 elements
666     Z = state_i(1); % Z Distance of Rocket [m]
667     X = state_i(2); % X Distance of Rocket [m]
668     Y = state_i(3); % Y Distance of Rocket [m]
669     m_r = state_i(4); % Mass of Rocket [kg]
670     m_air = state_i(5); % Mass of the Air [kg]
671     v = state_i(6); % Volume of Air [m^3]
672     V_z = state_i(7); % Velocity in the Z direction [m/s]
673     V_x = state_i(8); % Velocity in the X direction [m/s]
674     V_y = state_i(9); % Velocity in the Y direction [m/s]
675
676     % Water Expulsion Phase
677     if (v < vol_bottle)
678         % Calculate rate of change of Volume over time

```

```

678     v_dot = c_d*A_t*sqrt((2/rho_water)*(p_air_initial*((vol_air_initial/v)
        ^y)-p_ambient));
679     % Calculate the pressure in the bottle over time
680     p_air = p_air_initial*(vol_air_initial./v).^y;
681     % Calculate the force of thrust from water expulsion over time
682     F_thrust = 2*c_d*A_t*(p_air-p_ambient);
683     % Calculate the change in mass of the rocket over time
684     m_dot_r = (-c_d)*A_t*sqrt(2*rho_water*(p_air-p_ambient));
685     % Calculate the change in mass of the air over time
686     m_dot_a = 0;
687 else
688     % Pressure of Air at end of Water Expulsion Phase
689     p_end = p_air_initial*(vol_air_initial/vol_bottle)^y;
690     % Temperature of Air at end of Water Expulsion Phase
691     t_end = temperature_air_initial*(vol_air_initial/vol_bottle)^(y-1);
692     % Calculate Pressure
693     p_air = p_end*(m_air./m_air_initial).^y;
694 end
695
696 % Gas Expulsion Phase
697 if (v >= vol_bottle) && (p_air > p_ambient)
698     % Change in Volume
699     v_dot = 0;
700     % Calculate Critical Pressure
701     p_crit = p_air*(2/(y+1))^(y/(y-1));
702     % Determine Air Density
703     rho_air = m_air / vol_bottle;
704     % Determine Gas Temperature
705     T = p_air / (rho_air*R);
706
707     % Determine Flow Type
708     if (p_crit > p_ambient) % CHOKED FLOW
709         % Determine Exit Temperature
710         T_e = (2/(y+1))*T;
711         % Determine Exit Velocity
712         V_e = sqrt(y*R*T_e);
713         % Determine Exit Density of Air
714         rho_e = p_crit / (R*T_e);
715         % Calculate rate of change of mass of air
716         m_dot_a = -c_d*rho_e*A_t*V_e;
717         % rate of change of mass of rocket
718         m_dot_r = m_dot_a;
719         % Define Exit Pressure
720         p_e = p_crit;
721     elseif (p_crit <= p_ambient) % NOT CHOKED FLOW
722         % Determine Exit Mach Number
723         M_e = sqrt((((p_air/p_ambient)^(y-1)/y))-1)/((y-1)/2));
724         % Determine Exit Temperature
725         T_e = T / (1+(((y-1)/2)*(M_e^2)));
726         % Determine Exit Velocity
727         V_e = M_e*sqrt(y*R*T_e);
728         % Determine Exit Density of Air
729         rho_e = p_ambient * (R*T_e);
730         % Calculate rate of change of mass of air

```

```

731         m_dot_a = -c_d*rho_e*A_t*V_e;
732         % rate of change of mass of rocket
733         m_dot_r = m_dot_a;
734         % Define Exit Pressure
735         p_e = p_ambient;
736     end
737     % Calculate the force of thrust from air expulsion over time
738     F_thrust = (c_d*rho_e*A_t*V_e*V_e) + (p_ambient-p_e)*A_t;
739 end
740
741 % Ballistic Phase
742 if (v >= vol_bottle) && (p_air < p_ambient)
743     % Set everything to zero because we have no prop left
744     F_thrust = 0;
745     m_dot_r = 0;
746     m_dot_a = 0;
747     v_dot = 0;
748 end
749
750 % Calculate Wind (Scaled by altitude)
751 if (Z <= 22)
752     W_x = ((Z - 0) / (22 - 0)) * (W_a_x - W_g_x) + W_g_x;
753     W_y = ((Z - 0) / (22 - 0)) * (W_a_y - W_g_y) + W_g_y;
754     W_z = ((Z - 0) / (22 - 0)) * (W_a_z - W_g_z) + W_g_z;
755 else
756     W_x = W_a_x;
757     W_y = W_a_y;
758     W_z = W_a_z;
759 end
760
761 % Calculate Relative Velocity of Rocket
762 V = sqrt(((V_x-W_x)^2)+((V_z-W_z)^2)+((V_y-W_y)^2));
763
764 % Calculate Angle of Rocket
765 % Check to see if still on stand...
766 if (sqrt((X-X_0)^2+(Z-Z_0)^2) <= 1_s)
767     % At launch pad heading if still on stand
768     H_x = cos(launch_angle);
769     H_z = sin(launch_angle);
770     H_y = 0;
771 else
772     H_x = (V_x - W_x)/V;
773     H_z = (V_z - W_z)/V;
774     H_y = (V_y - W_y)/V;
775 end
776
777 % Calculate Drag on Rocket
778 drag = (1/2)*rho_amb*(V^2)*C_D*A_b;
779
780 % Calculate Acceleration
781 accel_x = (F_thrust*H_x - drag*H_x + 0)/m_r; % Acceleration
782         in X Direction
783 accel_y = (F_thrust*H_y - drag*H_y + 0)/m_r; % Acceleration
784         in Y Direction

```

```

783     accel_z = (F_thrust*H_z - drag*H_z - (m_r*g))/m_r;           % Acceleration
        in Z Direction
784
785     % Output State
786     state(1) = V_z;           % Velocity in Z direction
787     state(2) = V_x;           % Velocity in X Direction
788     state(3) = V_y;           % Velocity in Y Direction
789     state(4) = m_dot_r;        % Rate of Change of Mass
        of Rocket
790     state(5) = m_dot_a;        % Rate of Change of Mass
        of Air
791     state(6) = v_dot;          % Rate of Change of Volume
        of Air
792     state(7) = accel_z;        % Rate of Change of
        Velocity in the Z Direction
793     state(8) = accel_x;        % Rate of Change of
        Velocity in the X Direction
794     state(9) = accel_y;        % Rate of Change of
        Velocity in the Y Direction
795 end
796
797 % F_thrust
798 %
799 % The function takes the ODE45 Flight Propagation state matrix that was output
800 % and calculates the force of thrust for plotting.
801 %
802 % @param state The state matrix from the ODE45 Flight Propagation call
803 % @param vars The constants vector that holds all relevant constants
804 %
805 function [out,phaseChange] = f_thrust(t, state, vars)
806     % Define Variables
807     c_d = vars(3);             % Discharge Coefficient
808     A_t = vars(5);             % Area of the Throat [m^2]
809     p_air_initial = vars(7);    % Initial Pressure of Air [N/m^2]
810     vol_air_initial = vars(8);  % Initial Volume of Air [m^3]
811     y = vars(9);               % Specific Heat Ratio of Air
812     p_ambient = vars(10);       % Ambient Pressure [psi]
813     p_ambient = (p_ambient*6894.76); % Convert to [N/m^2]
814     vol_bottle = vars(11);      % Volume of Bottle [m^3]
815     m_air_initial = vars(15);   % Initial Mass of the Air [kg]
816     R = vars(16);              % Gas Constant
817
818     % Variables for Phase Change logging
819     phaseChange = [];
820     phaseFlag = true;
821
822     % Iterate through state matrix data
823     for i = 1:size(state,1)
824         % Get important values from state matrix
825         m_air = state(i,5);     % Mass of the Air [kg]
826         v = state(i,6);         % Volume of Air [m^3]
827
828         % Water Expulsion Phase
829         if (v < vol_bottle)

```

```

830     if (phaseFlag == true)
831         phaseChange(1) = t(i+1);
832         phaseFlag = false;
833     end
834     % Calculate the pressure in the bottle over time
835     p_air = p_air_initial*(vol_air_initial./v).^y;
836     % Calculate the force of thrust from water expulsion over time
837     F_thrust = 2*c_d*A_t*(p_air-p_ambient);
838 else
839     % Pressure of Air at end of Water Expulsion Phase
840     p_end = p_air_initial*(vol_air_initial/vol_bottle)^y;
841     % Calculate Pressure
842     p_air = p_end*(m_air./m_air_initial).^y;
843 end
844
845 % Gas Expulsion Phase
846 if (v > vol_bottle) && (p_air > p_ambient)
847     if (phaseFlag == false)
848         phaseChange(2) = t(i+1);
849         phaseFlag = true;
850     end
851     % Calculate Critical Pressure
852     p_crit = p_air*(2/(y+1))^(y/(y-1));
853     % Determine Air Density
854     rho_air = m_air / vol_bottle;
855     % Determine Gas Temperature
856     T = p_air / (rho_air*R);
857
858     % Determine Flow Type
859     if (p_crit > p_ambient) % CHOKED FLOW
860         % Determine Exit Temperature
861         T_e = (2/(y+1))*T;
862         % Determine Exit Velocity
863         V_e = sqrt(y*R*T_e);
864         % Determine Exit Density of Air
865         rho_e = p_crit / (R*T_e);
866         % Define Exit Pressure
867         p_e = p_crit;
868     elseif (p_crit <= p_ambient) % NOT CHOKED FLOW
869         % Determine Exit Mach Number
870         M_e = sqrt((((p_air/p_ambient)^(y/(y-1)))-1)/((y-1)/2));
871         % Determine Exit Temperature
872         T_e = T / (1+((y-1)/2)*(M_e^2));
873         % Determine Exit Velocity
874         V_e = M_e*sqrt(y*R*T_e);
875         % Determine Exit Density of Air
876         rho_e = p_ambient * (R*T_e);
877         % Define Exit Pressure
878         p_e = p_ambient;
879     end
880     % Calculate the force of thrust from air expulsion over time
881     F_thrust = (c_d*rho_e*A_t*V_e*V_e) + (p_ambient-p_e)*A_t;
882 end
883

```



```

884     % Ballistic Phase
885     if (v > vol_bottle) && (p_air < p_ambient)
886         if (phaseFlag == true)
887             phaseChange(3) = t(i+1);
888             phaseFlag = false;
889         end
890         % Set everything to zero because we have no prop left
891         F_thrust = 0;
892     end
893     % Output the Force of Thrust
894     out(i) = F_thrust;
895 end
896 end
897
898 % hitGround
899 %
900 % The conditional stop function that checks the state matrix such that if
901 % the condition Z < 0ft is met the integration is terminated (i.e. hit the
902 % ground)
903 %
904 % @param t The input time
905 % @param state The input state matrix
906 %
907 function [value, isterminal, direction] = hitGround(t, state)
908     value = (state(1) < 0);
909     isterminal = 1; % Stop the integration
910     direction = 0;
911 end

```