ASEN 2012 Project 2 - Bottle Rocket Design

0dc91b091fd8

University of Colorado - Boulder

This project focused on the analysis and optimization of a Bottle Rocket's trajectory. In order to do this, a model for the rocket's position was developed using Newton's Laws of Motion, isentropic relations, and the Ideal Gas Law. Combining these together created a system of ODEs that could be implemented in and solved by MATLAB's built in ODE45 function, a Runge-Kutta 4th order ODE numeric ODE solver. Using this MATLAB tool, the model was verified and a sensitivity analysis was done on the independent initial conditions to the problem. The sensitivity analysis resulted in the conclusion that the amount of water initially in the rocket was most influential to its trajectory. With this knowledge, an optimal set of launch parameters was discovered which allowed the rocket to land with 0.1 m of its 85 m range goal. This lab showed the power and understanding that sensitivity analysis has on the optimization of dynamical systems.

Nomenclature

$P_{air,\,init}$	=	initial air pressure (Pa)
γ	=	specific heat ratio for air
ρ_{atm}	=	atmospheric air density (kg/m^3)
$\theta(t)$	=	angle of the rocket trajectory to the horizontal (rad)
A_{bottle}	=	Cross-Sectional Area of bottle (m^2)
$c_{discharge}$	=	discharge coefficient
C_{drag}	=	Drag Coefficient
$D(t)$	=	Drag (N)
$F(t)$	=	Thrust (N)
$g(t)$	=	gravitational acceleration (m/s^2)
$m_{air}(t)$	=	mass of air in bottle (kg)
$m_R(t)$	=	mass of rocket (kg)
$m_{water}(t)$	=	mass of water in bottle (kg)
ODE	=	Ordinary Differential Equation
$P(t)$	=	air pressure in bottle (Pa)
P_{atm}	=	atmospheric air pressure (Pa)
R	=	gas constant for air $(Jkg^{-}1K^{-}1)$
t	=	time (s)
$V(t)$	=	magnitude of velocity (speed) of the rocket (m/s)
$v(t)$	=	air volume (m^3)
$v_{air,init}$	=	initial air volume (m^3)
v_{bottle}	=	volume of bottle (m^3)
V_{exit}	=	exit velocity of fluid from rocket nozzle (m/s)

I. Introduction

The equations of motion used to determine the trajectory involved the solutions of a system of ODEs. This system is based on Newton's laws of motion and can be formulated as such:

$$\begin{aligned} m_R(t)\frac{dV(t)}{dt} &= F(t) - D(t) - m_R(t)\, g\, sin\theta(t) \\ V(t)\frac{d\theta(t)}{dt} &= -g\, cos\theta(t) \\ \frac{dx(t)}{dt} &= V(t)\, cos\theta(t) \\ \frac{dz(t)}{dt} &= V(t)\, sin\theta(t) \\ m_{rocket} &= m_{bottle} + m_{air} + m_{water} \end{aligned} \tag{1}$$

where $D(t)$ is calculated as such:

$$D(t) = \frac{\rho_{atm}}{2}\, V^2\, C_{drag}\, A_{bottle} \tag{2}$$

These basic equations allow for calculation of the rocket's position $x(t)$ and $y(t)$ at any time t. However, in order to find all of the variables needed to calculate derivative in Eqn. 1, the rocket's flight will have to partitioned into three separate modes of flight. These modes of flight are defined as: the flight-time where the rocket's thrust is provided by expelled water, the flight-time where the rocket's thrust is provided by the air pressure inside the rocket with no water left inside the tank, and finally the flight-time where the rocket's has no thrust and acts as a ballistic projectile.

In order to fully track all of the flight variables, in Eqn. 1, we must also keep track of m_{air}, m_{water}, and $v(t)$, as these variables will allow us to determine which flight regime the rocket currently in.

I.A. Water Thrust

In order to calculate the rocket's trajectory, we must use a set of relations to determine the V_{exit} of the water from the rocket and how the mass of the rocket changes, in order to calculate the required parameters of Eqns. 1 & 2. For the first flight regime, while there is still water in the rocket, the thrust of the rocket is determined by the expulsion of water from the rocket nozzle. This is done by first finding $P(t)$ and assuming isentropic[1] expansion of air during the expulsion of water, thus permitting the use of:

$$P(t) = P_{air,\, init} * \left(\frac{v_{air_i nt}}{v(t)}\right)^{\gamma} \tag{3}$$

where the rate of change of water mass is given as:

$$\frac{dm_{water}}{dt} = c_{discharge}\, \rho_{water}\, A_t\, V_{exit} \tag{4}$$

we can then find V_{exit} using:

$$V_{exit} = \sqrt{\frac{2\,(P(t) - P_{atm})}{\rho_{water}}} \tag{5}$$

and thus the thrust of the rocket in this period is given by:

$$F(t) = \frac{dm}{dt}\, V_{exit} \tag{6}$$

We must also calculate the change in $v(t)$ during this time to satisfy Eqns. 1 & 2. This can be done using:

$$\frac{v(t)}{dt} = c_{discharge}\, A_{throat}\, V_{exit} \tag{7}$$

With these equations, we can now use Eqns. 1 & 2 to calculate the rocket's trajectory during the time when the thrust is still governed by the expulsion of water from the rocket's nozzle.

I.B. Air Thrust

In this flight regime, the rocket's thrust is determined solely by the expulsion of air out of the rocket nozzle due to higher-than-atmospheric air pressure still in the rocket. This regime is starts when $v(t)$ is equal to the volume of the bottle. In order to calculate the rocket's trajectory, we must use a set of relations to determine the V_{exit} of the air from the rocket and how the mass of the rocket changes, in order to calculate the required parameters of Eqns. 1 & 2. Once again, this is done by first finding $P(t)$ and assuming isentropic[1] expansion of air, thus permitting the use of the following:

$$P_{end} = P_{air,init} \left(\frac{v_{air,init}}{v_{bottle}} \right)^{\gamma}, \quad P(t) = P_{end} \left(\frac{m_{air}}{m_{air,init}} \right)^{\gamma} \tag{8}$$

and using the Ideal Gas Law:[1]

$$\rho = \frac{m_{air}}{v_{bottle}} \quad T = \frac{P(t)}{\rho \, R} \tag{9}$$

Now, to define the flow out of the rocket nozzle, we must make sure to use the following definition for critical pressure P_{crit} so that we may determine if there is choked flow. By finding whether the flow is chocked, we can use the proper equations to determine V_{exit}:

$$P_{crit} = P(t) \left(\frac{2}{\gamma + 1} \right)^{\frac{\gamma}{\gamma - 1}} \tag{10}$$

As the air in the rocket is expelled to create thrust, the mass of the air in the rocket, and thus the weight of the rocket itself can be found using the following:

$$\frac{dm_{air}}{dt} = c_{discharge} \, \rho_{exit} \, A_{throat} \, V_{exit} \tag{11}$$

Using Eqn. 11 and the relations in Eqns. 13 & 14 for choked and un-choked flow, the thrust from the expulsion of air from the rocket nozzle is defined as:

$$F = \frac{dm_{air}}{dt} \, V_{exit} + (P_{exit} - P_{atm}) \, A_{throat} \tag{12}$$

I.B.1. Choked Flow

Choked flow is defined as the condition where the exit mach number $M_{exit} = 1$. Under these conditions, V_{exit} and ρ_{exit} are defined as follows, where $P_{exit} = P_{crit}$:

$$T_{exit} = \left(\frac{2}{\gamma + 1} \right) T, \quad \rho_{exit} = \frac{P_{crit}}{R \, T_{exit}}, \quad V_{exit} = \sqrt{\gamma \, R \, T_{exit}} \tag{13}$$

Eqn. 13 is used to solve for V_{exit} and ρ_{exit} in order to satisfy Eqns. 12 & 11. Then, using the value for thrust found from Eqn. 12, along with drag found in Eqn. 2, the trajectory of the rocket is calculated using Eqn. 1.

I.B.2. Flow not Choked

If the flow is not choked, then the exit air pressure $P_{exit} = P_{atm}$, and the following system of equation can be solved to determine V_{exit} and ρ_{exit}:

$$\begin{aligned} \frac{P(t)}{P_{atm}} &= \left(1 + \frac{\gamma - 1}{2} \, {M_{exit}}^2 \right)^{\frac{\gamma}{\gamma - 1}} \\ \frac{T_{exit}}{T} &= \left(1 + \frac{\gamma - 1}{2} \, M_{exit} \right) \\ \rho_{exit} &= \frac{P_{atm}}{R \, T_{exit}} \end{aligned} \tag{14}$$

Eqn. 14 must be solved simultaneously, and allows for the solution of M_{exit} so that V_{exit} and ρ_{exit} might solved to satisfy Eqns. 12 & 11. Then, using the value for thrust found from Eqn. 12, along with drag found in Eqn. 2, the trajectory of the rocket is calculated using Eqn. 1.

I.C. Ballistic Regime

After the rocket has exhausted all of its water, and the air pressure $P(t)$ has dropped to ambient pressures P_{atm}, the rocket no longer is producing any thrust ($F = 0$) and is simply a ballistic projectile at that point. During this period, the trajectory of the rocket is calculated purely from Eqn. 1. The rocket is no longer changing mass, the volume of air is constant at the volume of the bottle, and the mass of water is zero kg.

II. Methodology

In order to solve the system of equations as developed and described by Eqns. 1 - 14, an analytic tool was developed to allow for the efficient solution of system of ODEs to a high accuracy. For this problem, MATLAB was utilized to develop code for the solution and analysis of this system of ODEs. The MATLAB function ODE45, which utilizes a fourth-order Runge-Kutta numerical approximation, was the primary tool used to analyze the rocket trajectory.

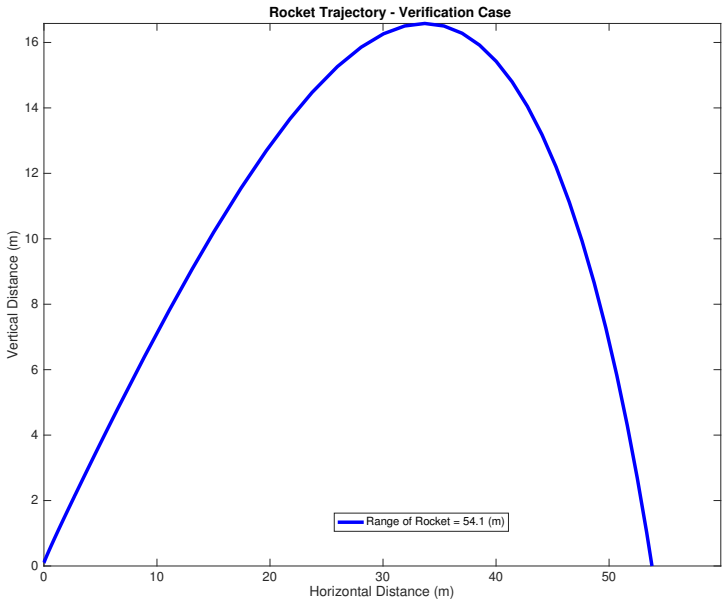

Figure 1. The output of my code under the given test conditions.

To solve the system of ODEs, code that took advantage of ODE45's ability to solve systems of ODEs had to be developed. Several functions were written to this purpose; one function `ASEN_2012_Project_2.m` passed all of the flight parameters (A_{throat}, C_{drag}, etc.) and initial conditions ($P_{air,init}$, $v_{air,init}$, etc.) to both the main script and to the derivative calculator function, while the other substantial function `ASEN_2012_Project_2_dy_dt.m` simply calculates all of the derivatives at a given time value for all seven tracked variables: $x(t)$, $z(t)$, $V(t)$, $\theta(t)$, $m_{air}(t)$, $m_{water}(t)$, $v_{air}(t)$. The first step in calculating the trajectory of the rocket under certain flight conditions is to get the initial conditions from `ASEN_2012_Project_2.m`, which returns both the initial conditions and all of the other flight parameters needed to solve the system of ODEs. Next, these initial conditions, along with a t range and the derivative function in `ASEN_2012_Project_2.m` are both passed to ODE45. ODE45 will evaluate the derivative function for each time step, using `ASEN_2012_Project_2.m` to find the rate of change of all seven tracked variables. ODE45 takes the change in each variable, adds it to the previous value of that variable, and then recalculates all seven variable's values using these new values and at the next t value. This process continues until it reaches the maximum t value, at which point ODE45 returns a matrix with the value of each variable evaluated at every t value.

In order to verify that this process was correct, the code was verified and checked against a given test case. The results of that verification are given in Figure 1, which shows the trajectory of the rocket under the test case conditions. The final range of the rocket was found to be $(53.4 - 54.1)m$, as shown in the Figure 1.

The solution process is applied every time ODE45 is called with a chosen set of initial conditions and flight parameters, which allows for a sensitivity analysis to be done. A sensitivity analysis involves varying one of the four independent initial conditions / flight parameters: θ_{init}, $m_{water,init}$ / $v_{water,init}$, C_{drag}, $P_{air,init}$. These four parameters are what determine the trajectory of the rocket, however it is unknown which of these has the greatest influence on the range and maximum height obtained by the rocket. By performing a sensitivity analysis on these four variables, the most impactful parameter on the range and height of the

rocket can be found. This analysis was carried out by setting the parameters such that the range of the projectile was 85 m, and then varying them about 20% from their 85 m values. The program would then be given the flight conditions such that all of the parameters except for the varied parameter were constant at their 85 m values. Every time the code would calculate and plot the trajectory for the case of one varied parameter, it would increment this parameter (or change which parameter it was varying) and again find and plot the trajectory under the influence of this new parameter. This process was done seven times for each of the four independent parameters, producing four plots of the trajectories of with one of the parameters varied.

These plots could be used to determine which variable was most influential on the trajectory of the rocket, as certain parameters when varied about the same percentage would produce much larger changes in trajectory or would even change how the rocket would respond to increasing that parameter. By visually inspecting the graphs, it can be clearly seen which parameters cause unexpected or non-linear changes in the trajectory of the rocket, and which parameters are directly proportional to the range of the rocket. Therefore, parameter that exhibited that cause large, non-linear changes in the trajectory of the rocket was determined to be the most "sensitive" parameter. This knowledge then allowed for a more precise and efficient method of finding the optimal launch conditions to reach 85 m.

A more detailed account of my algorithm can be found in the flow chart, or in my code itself (look at the second subsection for more readable code if you are not copy-pasting it into MATLAB), which is in the Appendix.

III. Results

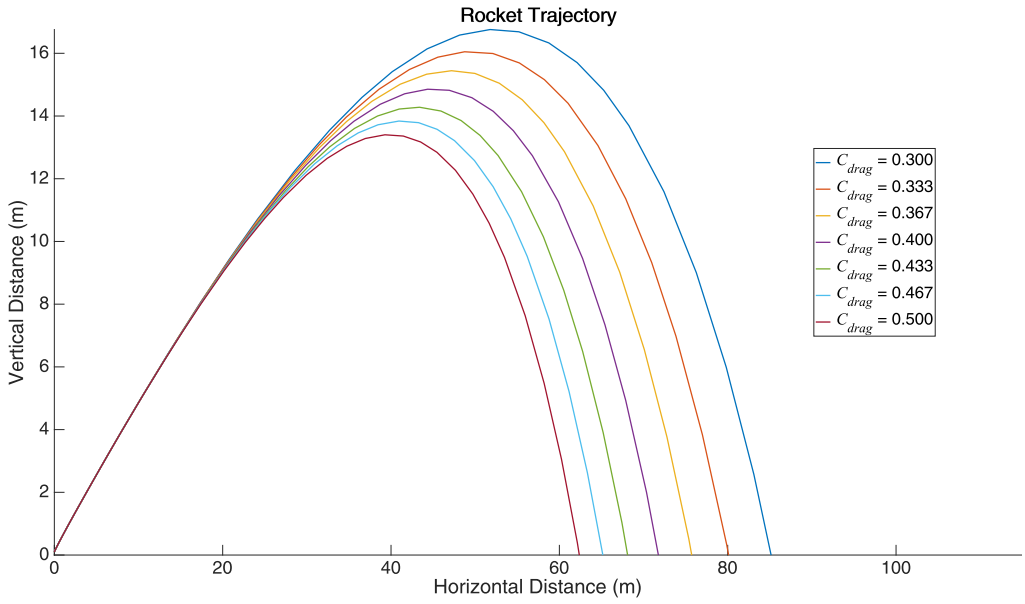

Figure 2. Plot showing the effect of the Drag Coefficient on the trajectory of the rocket. *Plot illustrates the very small effect of drag coefficient on shorter flights; it only starts to change to trajectory after longer flight times. Uniformly decreases the range of the rocket as it is increased, so it can be easily chosen to minimize to get maximum range out of the rocket.*

The results of the sensitivity analysis are shown below, witch each plot representing how the trajectory of the rocket responded to changing just one parameter. These plots were generated using the algorithm described in the Methodology section and much more thoroughly in the Appendix.

Figure 2 shows the trajectory's dependence on the drag coefficient, C_{drag}, and how the rocket's range decreases uniformly with an increased C_{drag}. Increasing this parameter had more influence on the trajectory as the flight time increased. As the rocket's range increased, drag became a more and more important variable in changing the trajectory.

Figure 3 shows the trajectory's dependence on the drag coefficient, $P_{air,init}$, and how the rocket's range

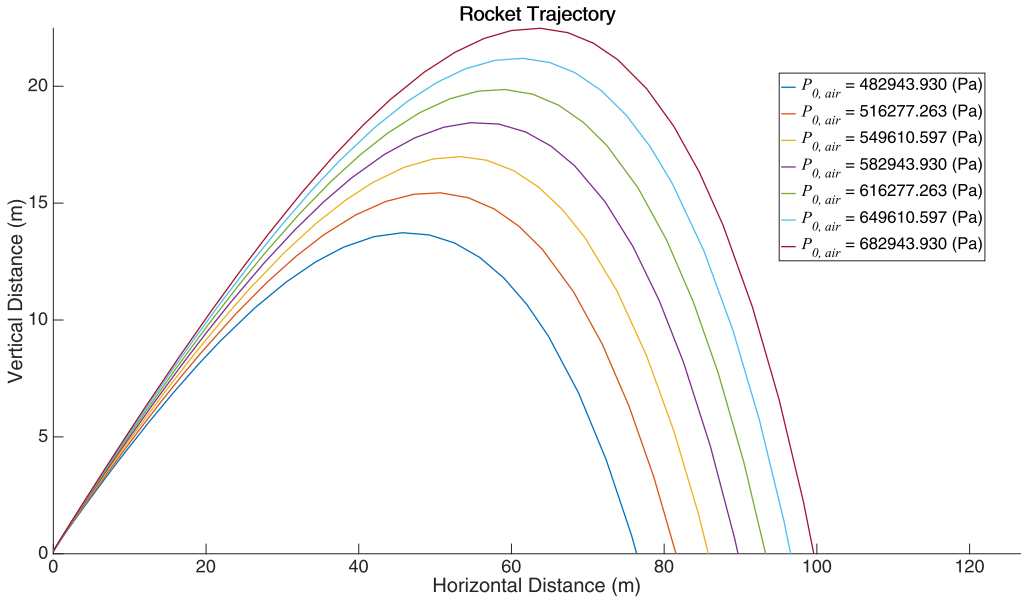

Figure 3. Plot showing the effect of the Initial Air Pressure on the trajectory of the rocket. *The air pressure in the rocket again has a fairly simple impact on the trajectory of the rocket, as it uniformly causes the range of the rocket to increase.*

increases uniformly with an increased $P_{air,init}$. Increasing this parameter again had more influence on the trajectory as the flight time increased. As the rocket's range increased, pressure became a more and more important variable in changing the trajectory.

Figure 4 shows the trajectory's dependence on theta, θ, and how the rocket's range does not directly correlate with increased θ. Increasing this parameter did not necessarily have more influence on the trajectory as the flight time increased. The trajectories were very unique with each different θ.

Figure 5 shows the trajectory's dependence on the amount of water initially in the rocket, $m_{water,init}$, and how the rocket's range does not directly correlate with increased $m_{water,init}$. Increasing this parameter did not necessarily have more influence on the trajectory as the flight time increased. The trajectories were very unique with each different $m_{water,init}$. While this parameter behaves similarly to θ's influence on trajectory, it is clear from looking at Figures 4 & 5 that changing the amount of water in the rocket initially has a much larger affect on the trajectory and thus on the maximum range and height as compare to the effect that launch angle has on range and maximum height.

After performing this sensitivity analysis, the most sensitive parameter was chosen to be $m_{water,init}$, as it can be clearly seen from Figure 5 that changing $m_{water,init}$ can result in large, nonlinear changes to the trajectory relative to the changes in $m_{water,init}$. With this knowledge, The parameters developed to hit were all fine tuned from the results of this analysis, and it was chosen to vary the mass of water very little, as its large and non-linear influence on the trajectory meant that optimizing it was a much bigger challenge. The results of this analysis is shown in Figure 6. The parameters used to produce were eventually optimized to be: $\theta_{init} = 33.7$ deg, $m_{water,init} = 1\ kg$, $C_{drag} = 0.3$, $P_{air,init} = 461949\ Pa = 67\ psi$. These values were chosen through iterative changes to each variable to based on the results of the sensitivity analysis, and thus the parameters changed the most were C_{drag} and $P_{air,init}$, and θ last. The pressure was chosen such that it had about a 1.5 factor of safety with regards to the approximate maximum pressure allowable in a bottle of $130psi$. This combination of parameters allowed me to hit the 85 m within 0.1 m.

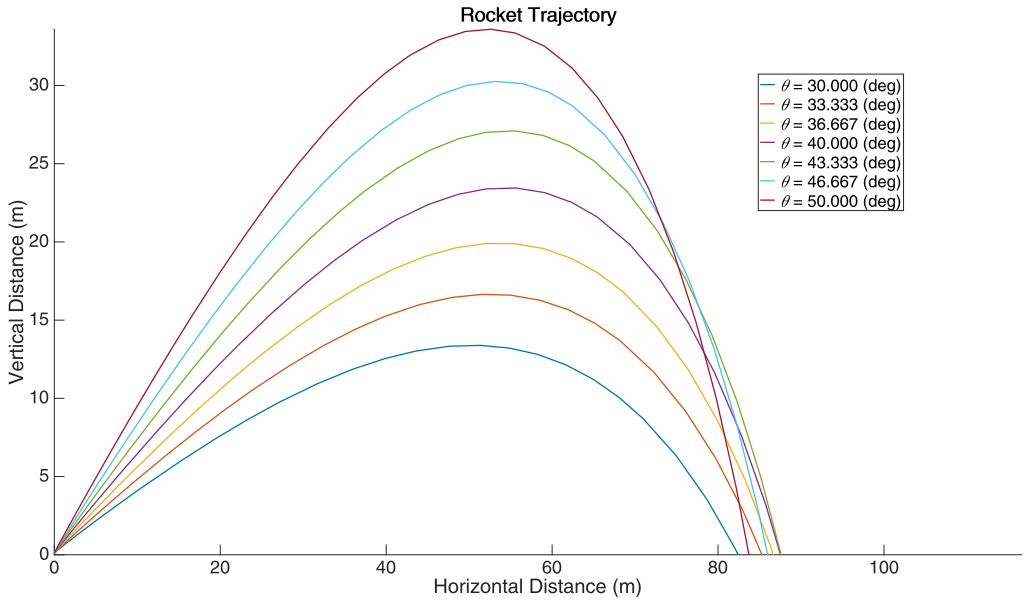

Figure 4. Plot showing the effect of the initial launch angle on the trajectory of the rocket. *The launch angle's effect on the rocket trajectory was less straightforward than it was for pressure and drag coefficient. As can be seen from the graph, continually increasing the launch angle does not cause the range of the projectile to uniformly increase.*

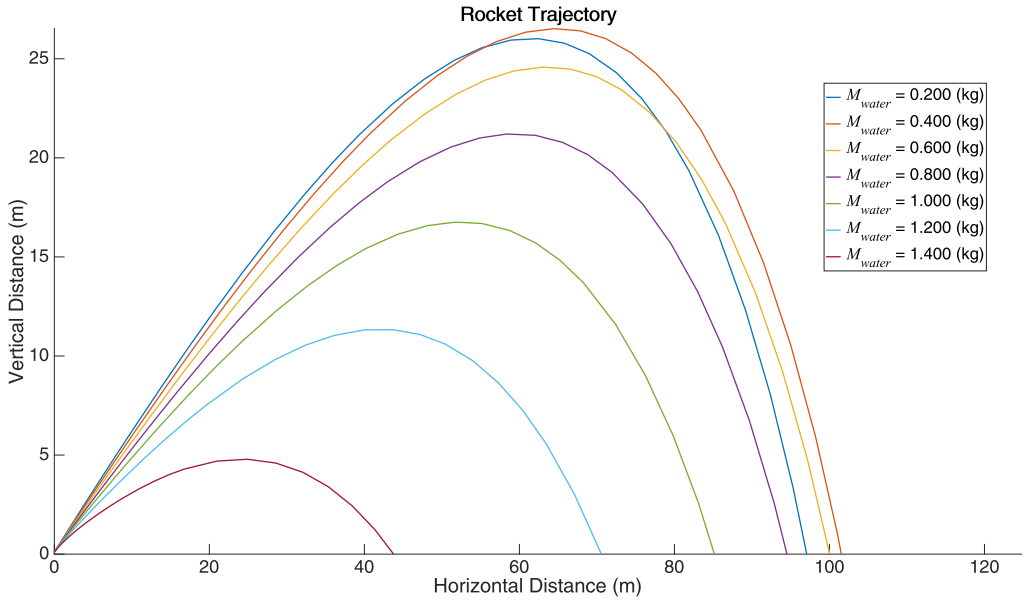

Figure 5. Plot showing the effect of the initial mass of water on the trajectory of the rocket. *The amount of water in the rocket affects the behavior of the rocket's trajectory much differently than do pressure or drag coefficient. While the pressure and drag coefficient, when increased, uniformly increase the rocket's range, increasing the amount of water does not always increase the range. As can be seen on the plot, increasing the mass of water much above 0.4 kg will cause the range of the rocket to decrease, while increasing it above 0.2 kg actually leads to a greater range.*

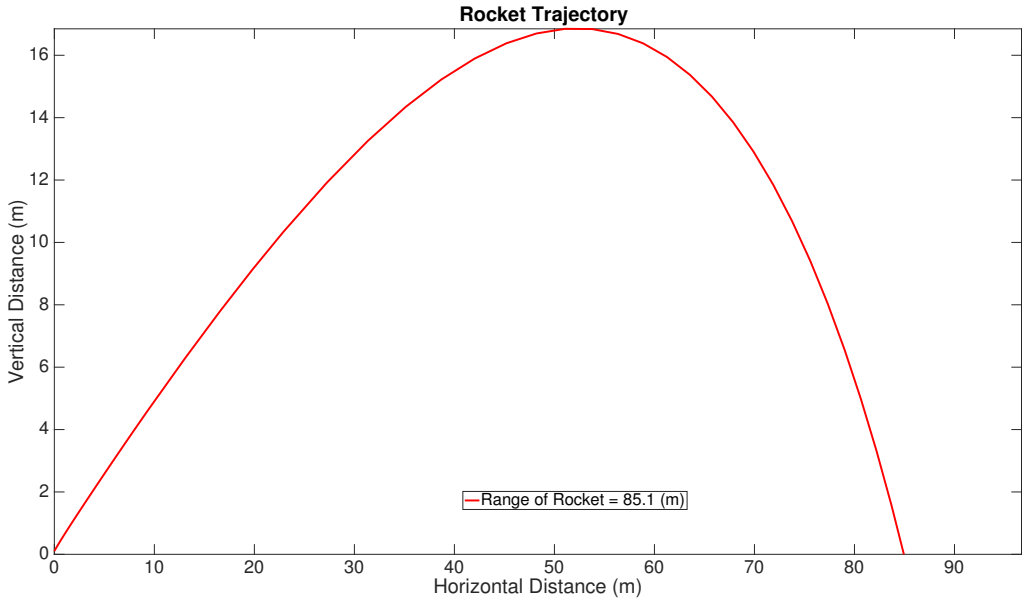

Figure 6. Plot showing the effect of the initial mass of water on the trajectory of the rocket. *The amount of water in the rocket affects the behavior of the rocket's trajectory much differently than do pressure or drag coefficient.*

IV. Discussion

As was discussed in the results section, the variables varied in Figures 2 - 5 did not all affect the trajectory of the rocket in the same way. By far, of all the parameters varied, the $m_{water,init}$ was the most influential on the final trajectory of the rocket. While the pressure and drag coefficient, when increased, uniformly increase the rocket's range, increasing the amount of water does not always increase the range. As can be seen in Figure 5, increasing the mass of water much above 0.4 kg will cause the range of the rocket to decrease, while increasing it above 0.2 kg actually leads to a greater range. This means there is an optimal amount of water somewhere between 0.4 and 0.2 kg, and would require more advanced analysis to fully optimize. Similarly, the launch angle also behaved in a similar way to $m_{water,init}$, as can be seen in Figure 4. In the same way as for $m_{water,init}$, there is a bifurcation value of θ, where the angle becomes too great and causes the rocket to actually decrease its range. Again, there is more optimization to be done with regards to the launch angle, as from Figure 4, as the range maximizing launch angle is between 40 and 45 degrees. However, for getting the rocket to 85 m I lowered the angle so I could be have more control over the less sensitive parameters in terms of controlling the range of the rocket.

V. Conclusion

By starting with the basic equations of motion as given by Eqn. 1, and building them up using aerodynamic and thermodynamic assumptions, an accurate model of the trajectory of a bottle rocket was developed and implemented in MATLAB. Verification was done on this MATLAB program, making sure this analytic tool was producing the correct results. This analysis tool was then used to find the most sensitive parameter - the parameter affecting the trajectory of the bottle rocket the most. After discovering how each parameter affected the rocket's trajectory, a set of initial conditions was developed that allowed the rocket to accurately and reliably hit the 85 m range mark.

It was found that the most sensitive parameter was the amount of water initially in the rocket. This parameter made the rocket behave in an interesting way, as increasing or decreasing the amount of water did not have a constant effect on the range of the rocket. This means that there is even more optimization to be done. In the same way, theta also had a non-trivial trend with range, and it increasing the launch angle did not consistently increasing the rocket's range.

Therefore, if the range of the rocket were to be thoroughly optimized, the value of θ would need to be optimized between 40 and 45 degrees, and the value of $m_{water,init}$ would also need to be optimized to a value between 0.2 and 0.4 kg. Finding these optimal values for θ and $m_{water,init}$, along with minimizing the drag coefficient and maximizing the initial pressure in the rocket, would allow for the furthest range possible. Combining this sensitivity analysis with the great speed and efficiency of programmatic optimization demonstrates the analytic ability of this procedure generalized to any systematic problem. This process demonstrated how sensitivity analysis leads to the ability to find how and what to optimize in any system.

References List of source that provide information that helped you carry out the study

References

[1]Anderson D, A. Jr., *Introduction to Flight*, McGraw-Hill Education Publishing, New York, 2016.

Appendix

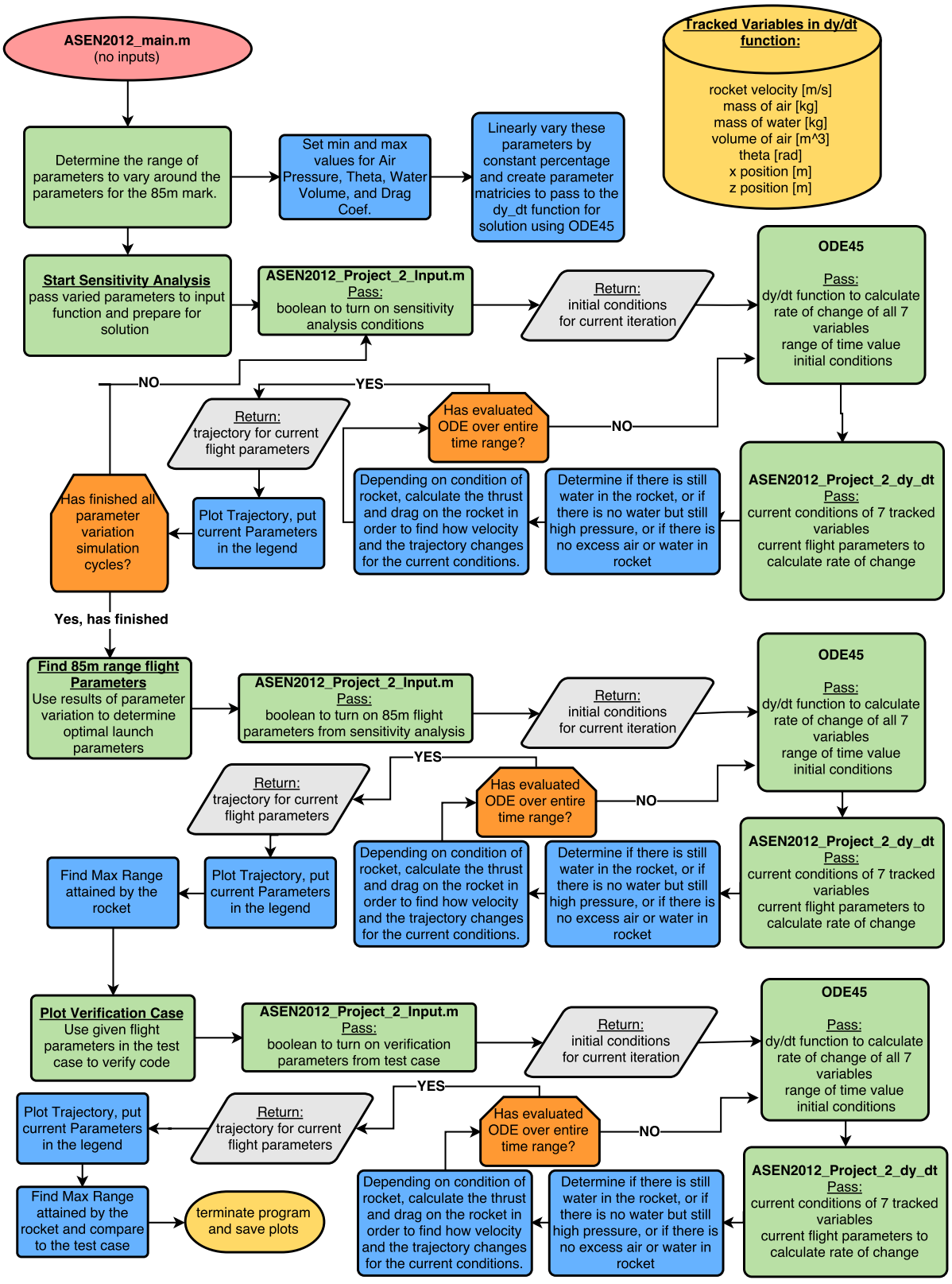

Figure 7. Flow Chart detailing the structure of the code.

Code for Copy-Pasting

In the next subsection you can find code that is formatted better visually, but will not directly work if copy-pasted directly into MATLAB. PLEASE DO NOT COPY-PASTE from the next subsection, it will not work. Unfortunately, in order to make the following code copy-pastable, much of its formatting had to be removed so that it would not interfere with the pdf's ability to display it in such a way that it would still run when copied into MATLAB. If you want to see the code in a way that more closely resembles its original format, see the next section of the Appendix.

The code below is named `ASEN_2012_Project_2_main.m`:

```
%%% Main Script, runs ODE45 Simulations of Rocket Trajectory

%%% purpose: this code is is the driver script, and it runs all of the
%%%          simulations necessary to find the parametrs that allow the
%%%          rocket to reach 85m, and the range at 85m depends on the
%%%          launch angle, the initial vol/mass of water, the drag
%%%          coefficient, and the inital air pressure in the rocket.
%%%
%%% inputs: does not have any required inputs, but does call
%%%         ASEN_Project_2_Input to get the necessary data it needs to run.
%%%
%%%
%%% outputs: has no defined outputs, but will plot the results of its
%%%          sensitivity analysis, the 85m trajectory, and the verification
%%%          case in separate windows.
%%%
%%%
%%% assumptions: This driver script really makes no assumptions, as it does
%%%              not do any of the actual scientific calculation.
%%%
%%%
%%% author's ID: 0dc91b091fd8
%%% date created: 11/25/2016
%%% date modified: 12/1/2016


clear all %#ok<CLALL>
clc
close all

NUM_SENSITIVITY_ITERATIONS = 5;
NUM_SENSITIVITY_VARS = 4;

%%% return maximum and minimum allowed value for each global variable
% [ P_init, m_water, C_drag,     theta ]
% [ (Pa), (kg), (unit-less), (radians) ]
P_atm = 82943.93; % [Pa]
max_global_vals = [600000 + P_atm, 1.4, 0.5, 50 * (pi / 180)];
min_global_vals = [400000 + P_atm, 0.2, 0.3, 30 * (pi / 180)];

% plotting setup

sens_names = {'P_{0, air}', 'M_{water}', 'C_{drag}', 'theta'};
sens_names_form = {sprintf('{\\fontname{times} \\it%s}', sens_names{1}), ...
```

```
                  sprintf('{\\fontname{times} \\it%s}', sens_names{2}), ...
                  sprintf('{\\fontname{times} \\it%s}', sens_names{3}), ...
                  sprintf('{\\fontname{times} \\it\\%s}', sens_names{4})};
sensitvity_units = {'(Pa)', '(kg)', '', '(deg)'}; % convert to degress and
                                                  % for display


%% Creating Vectors of Varied Parameters

% initialize testing matrix
sens_var_vec = cell(1, NUM_SENSITIVITY_VARS);

for i = 1:NUM_SENSITIVITY_VARS

    % create linearly spaced vectors with 1 variable varied from its min to
    % its max with all of the rest of the variables constant at their
    % current values
    var_index = i;

    current_var_max = max_global_vals(var_index);
    current_var_min = min_global_vals(var_index);

    % Create Basic Test Matrix, a NUM_SENSITIVITY_VARS x
    % NUM_SENSITIVITY_ITERATIONS matrix that contains the values of the
    % sensitivity parameters that are to be varied. For each sensitivity
    % variable, one test matrix will have NUM_SENSITIVITY_ITERATIONS values
    % ranging from the minimum to the maximum global varibale value to do
    % sensitivity analysis on.

    % use the parameters that allow for 85m to vary the parameters around
    var_mat = [461949 + P_atm, 1.0, 0.3, 33.5 * (pi / 180)];

    temp_mat = repmat(var_mat, [NUM_SENSITIVITY_ITERATIONS, 1]);
    sens_var_vec{var_index} = temp_mat;

    % creating vector that varies from min and max param value, with
    % NUM_SENSITIVITY_ITERATIONS steps
    param_var_vec = linspace(current_var_min, current_var_max,...
                             NUM_SENSITIVITY_ITERATIONS);

    % Create Varying Paramter column in current Parameter matrix
    sens_var_vec{var_index}(:, var_index) = param_var_vec;

    % Make Legend String
    for j = 1:NUM_SENSITIVITY_ITERATIONS

        % convert radians back to degrees for readability
        if i == 4 % looking at theta [rad]
            curr_sens_val = sens_var_vec{var_index}(j, var_index)*(180/pi);
        else
            curr_sens_val = sens_var_vec{var_index}(j, var_index);
        end

        LEGEND_STRING{i}{j} = sprintf('%s = %0.3f %s',...
                                      sens_names_form{i}, ...
```

```
                                    curr_sens_val, ...
                                    sensitvity_units{i}); %#ok<SAGROW>

    end

end


%% Sensitivity Analysis

% set which parameters to use in ODE45 based on whether you are doing
% sensitivity analysis, finding 85m params, or running a verification case
run_sens_analysis = true;
find_85m_range = false;
run_verification_case = false;

% set global bools that control parameter data flow
setGlobalBools(run_sens_analysis, find_85m_range, run_verification_case);

for i = 1:NUM_SENSITIVITY_VARS

    %% new plot for each variable analyzed
    hFig = figure(i);
    set(gca,'FontSize', 28)
    set(hFig, 'Position', [100 100 1600 900])
    hold on

    % Print Simulation Status to Command Window
    fprintf('Current Parameter Being Varied: %s\n', sens_names{i})

    for j = 1:NUM_SENSITIVITY_ITERATIONS

        %%% Determine value of global variable, and which variable will be
        %%% modified for the current solution iteration
        global_conditions = sens_var_vec{i}(j, :);
        setGlobals(global_conditions);

        %% define initial conditions and flight variables

        % Initialize variables, return initial conditions vector
        [~, init_conds] = ASEN2012_Project_2_Input(run_sens_analysis, ...
                                                   find_85m_range, ...
                                                   run_verification_case);

        % Definging the range of time values to solve the system of ODEs
        INIT_TIME = 0; % seconds
        FINAL_TIME = 5*((j + 6)/3); % last time of projectile - seconds
        time_interval = [INIT_TIME, FINAL_TIME];


        %% Solve System of ODEs and return numeric results
        [~, rocketVars] = ode45('ASEN2012_Project_2_dy_dt', ...
                                time_interval, init_conds);

        % Set Trajectory Variables
```

```
        x = rocketVars(:, 5);
        z = rocketVars(:, 6);

        plot(x, z, 'LineWidth', 2)

        fprintf('Simultion Cycle %d/%d Completed.\n', j,...
                NUM_SENSITIVITY_ITERATIONS)

    end

    %% Plot Settings
    legend(LEGEND_STRING{i}, 'location', 'northwest')
    ylim([0, inf])
    xlim([0, inf])
    xlabel('Horizontal Distance (m)')
    ylabel('Vertical Distance (m)')
    title('Rocket Trajectory')

    hold off

    fprintf('\n')

end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Find conditions for landing at 85 [m]

% set which parameters to use in ODE45 based on whether you are doing
% sensitivity analysis, finding 85m params, or running a verification case
run_sens_analysis = false;
find_85m_range = true;
run_verification_case = false;

% set global bools that control parameter data flow
setGlobalBools(run_sens_analysis, find_85m_range, run_verification_case);

% Print Simulation Status to Command Window
fprintf('Finding Parameters for 85m Range Target.\n\n')

%% define initial conditions and flight variables

% Initialize variables, return initial conditions vector
[~, init_conds] = ASEN2012_Project_2_Input(run_sens_analysis, ...
                                           find_85m_range, ...
                                           run_verification_case);

% Definging the range of time values to solve the system of ODEs
INIT_TIME = 0; % seconds
FINAL_TIME = 5; % last time of projectile - seconds
time_interval = [INIT_TIME, FINAL_TIME];


%% Solve System of ODEs and return numeric results over time interval
```

```
[~, rocketVars] = ode45('ASEN2012_Project_2_dy_dt', time_interval, ...
                        init_conds);

% Set Trajectory Variables
x = rocketVars(:, 5);
z = rocketVars(:, 6);

% Find Range of Rocket
z_0 = 0; % looking for when rocket hits ground
near_x_0 = find(x > 30, 1, 'first');
diff_vec = abs(z(near_x_0:end) - z_0);
[~, idx] = min(diff_vec); % index of closest value
idx = idx + length(z(1:near_x_0 - 1)); % adjusting index

range = x(idx);

%% Plot 85m Solution
hFig = figure(NUM_SENSITIVITY_VARS + 1);
set(hFig, 'Position', [100 100 1600 900])

plot(x, z, 'r','LineWidth', 3)

ylim([0, inf])
xlim([0, inf])
new_legend_string = sprintf('Range of Rocket = %0.3g (m)', range);
legend(new_legend_string, 'location', 'northwest')
xlabel('Horizontal Distance (m)')
ylabel('Vertical Distance (m)')
title('Rocket Trajectory')
set(gca,'FontSize', 28)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Run and Plot Verification Case

% set which parameters to use in ODE45 based on whether you are doing
% sensitivity analysis, finding 85m params, or running a verification case
run_sens_analysis = false;
find_85m_range = false;
run_verification_case = true;

% set global bools that control parameter data flow
setGlobalBools(run_sens_analysis, find_85m_range, run_verification_case);

% Print Simulation Status to Command Window
fprintf('Running Verification Case.\n')

%% define initial conditions and flight variables

% Initialize variables, return initial conditions vector
[~, init_conds] = ASEN2012_Project_2_Input(run_sens_analysis, ...
                                           find_85m_range, ...
                                           run_verification_case);

% Definging the range of time values to solve the system of ODEs
```

```
INIT_TIME = 0; % seconds
FINAL_TIME = 5; % last time of projectile - seconds
time_interval = [INIT_TIME, FINAL_TIME];


%% Solve System of ODEs and return numeric results over time interval
[~, rocketVars] = ode45('ASEN2012_Project_2_dy_dt', time_interval, ...
                        init_conds);

% Set Trajectory Variables
x = rocketVars(:, 5);
z = rocketVars(:, 6);

% Find Range of Rocket
z_0 = 0; % looking for when rocket hits ground
near_x_0 = find(x > 30, 1, 'first');
diff_vec = abs(z(near_x_0:end) - z_0);
[~, idx] = min(diff_vec); % index of closest value
idx = idx + length(z(1:near_x_0 - 1)); % adjusting index

range = x(idx);

%% Plot Verification Case
hFig = figure(NUM_SENSITIVITY_VARS + 2);
set(hFig, 'Position', [100 100 1600 900])

plot(x, z, 'b', 'LineWidth', 3)

ylim([0, inf])
xlim([0, inf])
new_legend_string = sprintf('Range of Rocket = %0.3g (m)', range);
legend(new_legend_string, 'location', 'northwest')
xlabel('Horizontal Distance (m)')
ylabel('Vertical Distance (m)')
title('Rocket Trajectory - Verification Case')
set(gca,'FontSize', 28)


function dydt = ASEN2012_Project_2_dy_dt(~, y)

    %%% Function for use in ODE45, calculates rate of change of launch
    %%% angle, the initial vol/mass of water, the drag coefficient, and the
    %%% inital air pressure in the rocket as functions of time.

    %%% purpose: This function calculates the rate of change of the tracked
    %%%          variables - launch angle, the initial vol/mass of water,
    %%%          the drag coefficient, and the inital air pressure in the
    %%%          rocket -  as functions of time. Formats its input and
    %%%          output for use in ODE45.
    %%%
    %%%
    %%% inputs: takes a vector y that contains all of the tracked variables
    %%%         that is used to calculate the new rate of change of all of
    %%%         these same tracked variables.
    %%%
```

```
%%%
%%% outputs: outputs a vector containing how each tracked variable
%%%          changed with each iteration.
%%%
%%%
%%% assumptions: assumes that the system is isentropic, that the gas
%%%              can be treated as an Ideal Gas, that water is
%%%              incompressible, and assumes steady atmospheric
%%%              conditions for the duration of the flight.
%%%
%%%
%%% author's ID: 0dc91b091fd8
%%% date created: 11/25/2016
%%% date modified: 12/1/2016

% sensitivity varibales are set each time sensitivity analysis is ran
[global_vars] = GetGlobals;

% Set the Value of the 4 Globals
P_air_init = global_vars(1);
C_drag = global_vars(3);

% determine wheter doing sensitivity analysis or not
[do_sens_analysis, do_85m_analysis, run_verif_case] = getGlobalBool;

%% resolving y vec. into its constituent variables for readability
Velocity = y(1);
mass_air = y(2);
mass_water = y(3);
theta = y(4);
x_pos = y(5); % only here for debugging
z_pos = y(6); % only here for debugging
V_air = y(7);


%% Initializing Needed Parameters for Trajectory Calculation
[flight_params, ~] = ASEN2012_Project_2_Input(do_sens_analysis, ...
                                              do_85m_analysis, ...
                                              run_verif_case);

g = flight_params(1);
C_discharge = flight_params(2);
V_bottle = flight_params(3);
P_atm = flight_params(4);
gamma = flight_params(5);
rho_water = flight_params(6);
A_throat = flight_params(7);
A_bottle = flight_params(8);
R_air = flight_params(9);
mass_bottle = flight_params(10);
V_air_init = flight_params(11);
Rho_air_atm = flight_params(12);
T_air_init = flight_params(13);
m_air_init = flight_params(14);
```

```
    %%% Make sure that air volume does not exceed bottle volume
    if V_air > V_bottle
        V_air = V_bottle;
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Determining Flight Regime of Rocket and Related parameters

    if V_air < V_bottle % Water Thrust Period of Flight

        % Air pressure at Time t
        P_air = P_air_init * (V_air_init / V_air) ^ gamma;

        % Exhaust Velocity
        Vel_exhaust = sqrt((2/rho_water) * (P_air - P_atm));

        % Mass flow rate of water out the Throat of the Bottle and of Air
        dmWater_dt = -C_discharge * rho_water * A_throat * Vel_exhaust;
        dmAir_dt = 0;

        % Thrust of Rocket
        F = 2 * C_discharge * (P_air - P_atm) * A_throat;

        %% Rate of Change of Volume of Air
        if V_air < V_bottle
            dVair_dt = C_discharge * A_throat * Vel_exhaust;
        else
            dVair_dt = 0;
        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    else % Air Thrust Period of Flight

        % Pressure and Temperature of air in bottle when all of the water
        % is expelled.
        P_end = P_air_init * (V_air_init / V_bottle) ^ gamma;
        T_end = T_air_init * (V_air_init / V_bottle) ^ (gamma - 1);

        % Air Pressure at Time t
        P_air_mass_eqn = P_end * (mass_air / m_air_init) ^ gamma;


        if P_air_mass_eqn > P_atm

            % Local Air Denisty & Air Temp
            rho_air = mass_air / V_bottle;
            T_air = P_air_mass_eqn / (rho_air * R_air);

            % Critical Pressure
            crit_pressure = P_air_mass_eqn * (2 / (gamma + 1)) ...
                                            ^ (gamma / (gamma - 1));

            %% Check for Choked Flow (M_exit = 1)
            if crit_pressure > P_atm % Choked Flow
```

```
    T_exit = (2 / (gamma + 1)) * T_air;
    rho_exit = crit_pressure / (R_air * T_exit);
    Velocity_exit = sqrt(gamma * R_air * T_exit);
    P_air_exit = crit_pressure;

elseif crit_pressure <= P_atm % Not Choked

    % Exit air Pressure is ambient air pressure
    P_air_exit = P_atm;

    %%% Solve System of Equations %%%
    syms M_exit T_exit rho_exit
    press_eqn = ( P_air_mass_eqn / P_atm == ...
        (1 + ((gamma - 1) / 2) * M_exit ^ 2) ^ (gamma / (gamma - 1)) );
    temp_eqn = ( T_exit / T_air == 1 + ((gamma - 1) / 2) * M_exit ^ 2 );
    rho_eqn = ( rho_exit == P_atm / (R_air * T_exit) );
    solution = solve(press_eqn, temp_eqn, rho_eqn, M_exit, T_exit, rho_exit);

    %%% Find Exit Mach Number %%%
    M_exit_indeces_real = find(imag(solution.M_exit) == 0);
    M_exit_real = double(solution.M_exit(M_exit_indeces_real));

    M_exit_indeces_positive = M_exit_real > 0;
    M_exit = M_exit_real(M_exit_indeces_positive);


    %%% Find Exit Temperature %%%
    T_exit_indeces_real = find(imag(solution.T_exit) == 0);
    T_exit_real = double(solution.T_exit(T_exit_indeces_real));

    T_exit_no_Duplicates = unique(T_exit_real);
    if length(T_exit_no_Duplicates) > 1
        error('duplicates homie');
    end
    T_exit = unique(T_exit_real);


    %%% Find Exit Air Density %%%
    rho_exit_indeces_real = find(imag(solution.rho_exit) == 0);
    rho_exit_real = double(solution.rho_exit(rho_exit_indeces_real));

    rho_exit_no_Duplicates = unique(rho_exit_real);
    if length(rho_exit_no_Duplicates) > 1
        error('duplicates homie');
    end
    rho_exit = unique(rho_exit_real);

    % Find exit Velocity
    Velocity_exit = M_exit * sqrt(gamma * R_air * T_exit);

    if isempty(Velocity_exit)
        error('All imaginary Solutions')
    end
```

```
            end

            % Volume of Air no longer is changing, V_air == V_bottle
            dVair_dt = 0;

            %% Rate of Change Air Mass, Water Mass
            dmAir_dt = -C_discharge * rho_exit * A_throat * Velocity_exit;
            dmWater_dt = 0;

            %% Calculate Thrust
            F = -dmAir_dt * Velocity_exit + (P_air_exit - P_atm) * A_throat;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        else % Ballistic (No Thrust Period) Phase

            F = 0; % no thrust anymore :'(

            % no change in amount of water/air in bottle
            dmAir_dt = 0;
            dmWater_dt = 0;

            % Volume of Air no longer is changing, V_air == V_bottle
            dVair_dt = 0;

        end


    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Drag
    Drag = (Rho_air_atm / 2) * (Velocity ^ 2) * C_drag * A_bottle;


    %% Basic Governing Equations

    % Mass of Rocket
    if mass_water < 0
        mass_water = 0;
    end

    mass_rocket = mass_bottle + mass_air + mass_water;

    dVel_dt = (F - Drag - mass_rocket * g * sin(theta)) / mass_rocket;

    if Velocity > 1
        dTheta_dt = (1/Velocity) * -g * cos(theta);
    else
        dTheta_dt = 0;
    end

    dx_dt = Velocity * cos(theta);
    dz_dt = Velocity * sin(theta);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Assign the change in y to the dydt vector for ode45

    dydt(1) = dVel_dt;
    dydt(2) = dmAir_dt;
    dydt(3) = dmWater_dt;
    dydt(4) = dTheta_dt;
    dydt(5) = dx_dt;
    dydt(6) = dz_dt;
    dydt(7) = dVair_dt;

    dydt = dydt';

end

function [flight_params,...
          initial_conditions] = ASEN2012_Project_2_Input(use_globals,...
                                                         find_85m_range, ...
                                                         do_verification)

    %%% Input function where constants for project are set.

    %%% purpose: This function exists to pass data to both the main and
    %%%          dy_dt function. This allows the flight parameters for
    %%%          solution in ODE45 to be consolodated to one place and
    %%%          allows for easy switching of which flight parameters are
    %%%          returned based on what type of simulation you are running.
    %%%
    %%% inputs: takes three bool inputs, which tell the function whether to
    %%%         return the specific flight parameters for running the
    %%%         sensitivity analysis, the 85m range finding, or the
    %%%         verification case.
    %%%
    %%%
    %%% outputs: returns a formatted vector of flight parameters needed in
    %%%          each iteration of the dy_dt ODE45 function for the
    %%%          calculation of the trajectory and a formatted vector of
    %%%          initial conditions needed in ODE45 to tell it where to
    %%%          begin solving the ODE.
    %%%
    %%%
    %%% assumptions: assumes that all parameters will be set in SI units,
    %%%              that water is incompressible, and that the gas in the
    %%%              rocket is an Ideal Gas for at lest the beginning of
    %%%              the flight.
    %%%
    %%%
    %%% author's ID: 0dc91b091fd8
    %%% date created: 11/25/2016
    %%% date modified: 12/1/2016

    %% Define Contants

    g = 9.81; % [m/s^2]
    C_discharge = 0.8; % discharge coefficient [dimensionless]
```

```
Rho_air_atm = 0.961; % ambient air density [kg/m^3]
V_bottle = 0.002; % volume of empty 2 litre bottle [m^3]
P_atm = 82943.93; % atmospheric pressure [Pa]
gamma = 1.4; % ratio of specific heats for air [dimensionless]
rho_water = 1000; % density of water [kg/m^3]
D_throat = 0.021; % diameter of throat [m]
D_bottle = 0.105; % diameter of bottle [m]
R_air = 287; % gas constant for air [J/kg/K]
m_bottle = 0.07; % mass of empty bottle [kg]
T_air_init = 300; % initial temperature of air [K]

A_throat = (pi/4) * D_throat ^ 2; % cross-sect area of throat [m^2]
A_bottle = (pi/4) * D_bottle ^ 2; % cross-sect area of bottle [m^2]

%% Define Initial Conditions

if use_globals

    % Set Global Variables for Sensitivity Analysis
    [global_vars] = GetGlobals;

    % Set the Value of the 4 Globals
    P_air_init = global_vars(1);
    m_water_init = global_vars(2);
    C_drag = global_vars(3);
    theta_init = global_vars(4);

    % re-calculate vol of water based on chosen mass of water
    V_water_init = m_water_init / rho_water; % initial vol.
                                             % of water in bottle [m^3]

elseif do_verification % run simulation on verification case

    C_drag = 0.5; % drag coefficient [dimensionless]

    %%% Initial Absolute Pressure of Air in Bottle [Pa]
    P_gauge_air = 344738; % initial gauge pressure of air in bottle[Pa]
    P_air_init = P_atm + P_gauge_air;

    V_water_init = 0.001; % initial vol. of water in bottle [m^3]

    % Initial Mass of Water
    m_water_init = rho_water * V_water_init; % [kg]

    theta_init = 45 * (pi / 180); % theta [rad]

elseif find_85m_range % use preset values for sensitivity params

    C_drag = 0.3; % drag coefficient [dimensionless]

    %%% Initial Absolute Pressure of Air in Bottle [Pa]
    P_gauge_air = 461949; % initial gauge pressure of air in bottle[Pa]
    P_air_init = P_atm + P_gauge_air;

    V_water_init = 0.001; % initial vol. of water in bottle [m^3]
```

```
        % Initial Mass of Water
        m_water_init = rho_water * V_water_init; % [kg]

        theta_init = 33.7 * (pi / 180); % theta [rad]

    end

    % upodate global conditions so that dydt uses preset values
    global_conditions = [P_air_init, m_water_init, C_drag, theta_init];
    setGlobals(global_conditions);

    %%% Initial Conditions of Tracked Variables
    Vel_init = 0; % initial velocity of the rocket [m/s]
    x_init = 0; % initial horizontal position [m]
    y_init = 0.1; % initial vertical position [m]

    % Initial Mass of Air in Bottle
    V_air_init = V_bottle - V_water_init; % [kg]
    m_air_init = (P_air_init * V_air_init) / (R_air * T_air_init); % [kg]


    %% Formatting Output

    flight_params = [g, C_discharge, V_bottle, P_atm, gamma,...
                     rho_water, A_throat, A_bottle, R_air, m_bottle, ...
                     V_air_init, Rho_air_atm, ...
                     T_air_init, m_air_init];

    initial_conditions = [Vel_init, m_air_init, m_water_init,...
                          theta_init, x_init, y_init, V_air_init];

end

function [bool_out_1, bool_out_2, bool_out_3] = getGlobalBool

    %%% function that gets the global boolean parameter control values

    %%% purpose: function allows the global bool variables: "if you are
    %%%          running a sensitivity analysis", "if you are trying to
    %%%          find the 85m range paramters", and "if you are running the
    %%%          verification case" to be passed to anywhere in any of the
    %%%          functions or scripts in use. This allows the user to get
    %%%          the parameters that control which flight params are passed
    %%%          to the dy_dt function used ODE45.
    %%%
    %%%
    %%% inputs: no reauired inputs.
    %%%
    %%%
    %%% outputs: returns the bool values for the bool
    %%%          parameters for use in other functions.
    %%%
    %%%
    %%% assumptions: assumes that the global vars have already been set
```

```
    %%%              before asssigning them to anything.
    %%%
    %%%
    %%% author's ID: 0dc91b091fd8
    %%% date created: 11/25/2016
    %%% date modified: 12/1/2016

    global is_doing_sensitiv_analysis find_85m_range do_verification

    % return the current setting for which paramters are
    % used in ODE45
    bool_out_1 = is_doing_sensitiv_analysis;
    bool_out_2 = find_85m_range;
    bool_out_3 = do_verification;

end

function [global_vars] = GetGlobals

    %%% function that gets the global variation parameters

    %%% purpose: function allows the global variation parameters: launch
    %%%          angle, the initial vol/mass of water, the drag
    %%%          coefficient, and the inital air pressure in the rocket to
    %%%          be passed to any other function or script without the
    %%%          other functions knowing about the variables
    %%%
    %%%
    %%% inputs: takes no inputs.
    %%%
    %%%
    %%% outputs: returns a vector of numeric values for the variation
    %%%          parameters.
    %%%
    %%%
    %%% assumptions: assumes that the global vars have already been set
    %%%              before asssigning them to anything.
    %%%
    %%%
    %%% author's ID: 0dc91b091fd8
    %%% date created: 11/25/2016
    %%% date modified: 12/1/2016

    % define pair i the initial pressure of air (the limit being the burst
    % pressure of the bottle, with some factor of safety), the initial
    % volume fraction (or equivalently initial mass) of water, the drag
    % coefficient and the launch angle as global variables to run the
    % simulation multiple times, all varying these paramters each time
    global P_air_init m_water_init C_drag theta_init

    % return current global variable values
    global_vars = [P_air_init, m_water_init, C_drag, theta_init];

end
```

```
function setGlobalBools(bool_in_1, bool_in_2, bool_in_3)

    %%% function that sets the global boolean parameter control values

    %%% purpose: function allows the global bool variables: "if you are
    %%%          running a sensitivity analysis", "if you are trying to
    %%%          find the 85m range paramters", and "if you are running the
    %%%          verification case" to be set from anywhere in any of the
    %%%          functions or scripts in use. This allows the user to
    %%%          change which parameters are passed to the dy_dt function
    %%%          used ODE45.
    %%%
    %%%
    %%% inputs: takes the bool values for the bool
    %%%         parameters that you want to assign to their respective
    %%%         global vars.
    %%%
    %%%
    %%% outputs: no defined outputs, but will set the 3 bool global
    %%%          variable values for the entire workspace.
    %%%
    %%%
    %%% assumptions: assumes that the input of bool global conditions is
    %%%              formatted properly so that the right values can be
    %%%              assigned to the right variables.
    %%%
    %%%
    %%% author's ID: 0dc91b091fd8
    %%% date created: 11/25/2016
    %%% date modified: 12/1/2016

    global is_doing_sensitiv_analysis find_85m_range do_verification

    % Set whether running sensitivty analysis, 85m optimization, or are
    % running a verification case
    is_doing_sensitiv_analysis = bool_in_1;
    find_85m_range = bool_in_2;
    do_verification = bool_in_3;

end

function setGlobals(global_conditions)

    %%% function that sets the global variation parameters

    %%% purpose: function allows the global variation parameters: launch
    %%%          angle, the initial vol/mass of water, the drag
    %%%          coefficient, and the inital air pressure in the rocket to
    %%%          be set from anywhere in any of the functions or scripts in
    %%%          use.
    %%%
    %%%
    %%% inputs: takes a vector of numeric values for the variation
    %%%         parameters that you want to assign to their respective
    %%%         global vars.
```

```
    %%%
    %%%
    %%% outputs: no defined outputs, but will set the 4 variation global
    %%%          variable values for the entire workspace.
    %%%
    %%%
    %%% assumptions: assumes that the vector of global conditions is
    %%%              formatted properly so that the right values can be
    %%%              assigned to the right variables.
    %%%
    %%%
    %%% author's ID: 0dc91b091fd8
    %%% date created: 11/25/2016
    %%% date modified: 12/1/2016

    % define pair i the initial pressure of air (the limit being the burst
    % pressure of the bottle, with some factor of safety), the initial
    % volume fraction (or equivalently initial mass) of water, the drag
    % coefficient and the launch angle as global variables to run the
    % simulation multiple times, all varying these paramters each time
    global P_air_init m_water_init C_drag theta_init

    % Set the Value of the 4 Globals
    P_air_init = global_conditions(1);
    m_water_init = global_conditions(2);
    C_drag = global_conditions(3);
    theta_init = global_conditions(4);


end
```

Code for Viewing

PLEASE DO NOT COPY THIS CODE DIRECTLY INTO MATLAB, it will not work. This sub-section is here just so you can view the code as you would see it in MATLAB, with all of its formatting preserved. Unfortunately, it will not work if you just copy it into MATLAB, there are issues with how the PDF puts extra spaces in when it is formatted as below. //

```
%%% Main Script, runs ODE45 Simulations of Rocket Trajectory

%%% purpose: this code is is the driver script, and it runs all of the
%%%          simulations necessary to find the parametrs that allow the
%%%          rocket to reach 85m, and the range at 85m depends on the
%%%          launch angle, the initial vol/mass of water, the drag
%%%          coefficient, and the inital air pressure in the rocket.
%%%
%%% inputs: does not have any required inputs, but does call
%%%         ASEN_Project_2_Input to get the necessary data it needs to
   run.
%%%
%%%
%%% outputs: has no defined outputs, but will plot the results of its
```

```
%%%          sensitivity analysis, the 85m trajectory, and the
   verification
%%%          case in separate windows.
%%%
%%%
%%% assumptions: This driver script really makes no assumptions, as it
   does
%%%              not do any of the actual scientific calculation.
%%%
%%%
%%% author's ID: 0dc91b091fd8
%%% date created: 11/25/2016
%%% date modified: 12/1/2016


clear all %#ok<CLALL>
clc
close all

NUM_SENSITIVITY_ITERATIONS = 5;
NUM_SENSITIVITY_VARS = 4;

%%% return maximum and minimum allowed value for each global variable
% [ P_init, m_water, C_drag,     theta ]
% [ (Pa), (kg), (unit-less), (radians) ]
P_atm = 82943.93; % [Pa]
max_global_vals = [600000 + P_atm, 1.4, 0.5, 50 * (pi / 180)];
min_global_vals = [400000 + P_atm, 0.2, 0.3, 30 * (pi / 180)];

% plotting setup

sens_names = {'P_{0, air}', 'M_{water}', 'C_{drag}', 'theta'};
sens_names_form = {sprintf('{\\fontname{times} \\it%s}', sens_names{1})
   , ...
                   sprintf('{\\fontname{times} \\it%s}', sens_names{2})
                      , ...
                   sprintf('{\\fontname{times} \\it%s}', sens_names{3})
                      , ...
                   sprintf('{\\fontname{times} \\it\\%s}', sens_names
                      {4})};
sensitvity_units = {'(Pa)', '(kg)', '', '(deg)'}; % convert to degress
   and
                                                  % for display


%% Creating Vectors of Varied Parameters

% initialize testing matrix
sens_var_vec = cell(1, NUM_SENSITIVITY_VARS);

for i = 1:NUM_SENSITIVITY_VARS

    % create linearly spaced vectors with 1 variable varied from its
       min to
    % its max with all of the rest of the variables constant at their
```

```
    % current values
    var_index = i;

    current_var_max = max_global_vals(var_index);
    current_var_min = min_global_vals(var_index);

    % Create Basic Test Matrix, a NUM_SENSITIVITY_VARS x
    % NUM_SENSITIVITY_ITERATIONS matrix that contains the values of the
    % sensitivity parameters that are to be varied. For each
       sensitivity
    % variable, one test matrix will have NUM_SENSITIVITY_ITERATIONS
       values
    % ranging from the minimum to the maximum global varibale value to
       do
    % sensitivity analysis on.

    % use the parameters that allow for 85m to vary the parameters
       around
    var_mat = [461949 + P_atm, 1.0, 0.3, 33.5 * (pi / 180)];

    temp_mat = repmat(var_mat, [NUM_SENSITIVITY_ITERATIONS, 1]);
    sens_var_vec{var_index} = temp_mat;

    % creating vector that varies from min and max param value, with
    % NUM_SENSITIVITY_ITERATIONS steps
    param_var_vec = linspace(current_var_min, current_var_max,...
                             NUM_SENSITIVITY_ITERATIONS);

    % Create Varying Paramter column in current Parameter matrix
    sens_var_vec{var_index}(:, var_index) = param_var_vec;

    % Make Legend String
    for j = 1:NUM_SENSITIVITY_ITERATIONS

        % convert radians back to degrees for readability
        if i == 4 % looking at theta [rad]
            curr_sens_val = sens_var_vec{var_index}(j, var_index)*(180/
               pi);
        else
            curr_sens_val = sens_var_vec{var_index}(j, var_index);
        end

        LEGEND_STRING{i}{j} = sprintf('%s = %0.3f %s',...
                                      sens_names_form{i}, ...
                                      curr_sens_val, ...
                                      sensitvity_units{i}); %#ok<SAGROW
                                         >

    end

end


%% Sensitivity Analysis
```

```
% set which parameters to use in ODE45 based on whether you are doing
% sensitivity analysis, finding 85m params, or running a verification
    case
run_sens_analysis = true;
find_85m_range = false;
run_verification_case = false;

% set global bools that control parameter data flow
setGlobalBools(run_sens_analysis, find_85m_range, run_verification_case
    );

for i = 1:NUM_SENSITIVITY_VARS

    %% new plot for each variable analyzed
    hFig = figure(i);
    set(gca,'FontSize', 28)
    set(hFig, 'Position', [100 100 1600 900])
    hold on

    % Print Simulation Status to Command Window
    fprintf('Current Parameter Being Varied: %s\n', sens_names{i})

    for j = 1:NUM_SENSITIVITY_ITERATIONS

        %%% Determine value of global variable, and which variable will
            be
        %%% modified for the current solution iteration
        global_conditions = sens_var_vec{i}(j, :);
        setGlobals(global_conditions);

        %% define initial conditions and flight variables

        % Initialize variables, return initial conditions vector
        [~, init_conds] = ASEN2012_Project_2_Input(run_sens_analysis,
            ...
                                                   find_85m_range, ...
                                                   run_verification_case
                                                       );

        % Definging the range of time values to solve the system of
            ODEs
        INIT_TIME = 0; % seconds
        FINAL_TIME = 5*((j + 6)/3); % last time of projectile - seconds
        time_interval = [INIT_TIME, FINAL_TIME];


        %% Solve System of ODEs and return numeric results
        [~, rocketVars] = ode45('ASEN2012_Project_2_dy_dt', ...
                                time_interval, init_conds);

        % Set Trajectory Variables
        x = rocketVars(:, 5);
        z = rocketVars(:, 6);

        plot(x, z, 'LineWidth', 2)
```

```
        fprintf('Simultion Cycle %d/%d Completed.\n', j,...
                 NUM_SENSITIVITY_ITERATIONS)

    end

    %% Plot Settings
    legend(LEGEND_STRING{i}, 'location', 'northwest')
    ylim([0, inf])
    xlim([0, inf])
    xlabel('Horizontal Distance (m)')
    ylabel('Vertical Distance (m)')
    title('Rocket Trajectory')

    hold off

    fprintf('\n')

end



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

%% Find conditions for landing at 85 [m]

% set which parameters to use in ODE45 based on whether you are doing
% sensitivity analysis, finding 85m params, or running a verification
   case
run_sens_analysis = false;
find_85m_range = true;
run_verification_case = false;

% set global bools that control parameter data flow
setGlobalBools(run_sens_analysis, find_85m_range, run_verification_case
   );

% Print Simulation Status to Command Window
fprintf('Finding Parameters for 85m Range Target.\n\n')

%% define initial conditions and flight variables

% Initialize variables, return initial conditions vector
[~, init_conds] = ASEN2012_Project_2_Input(run_sens_analysis, ...
                                           find_85m_range, ...
                                           run_verification_case);

% Definging the range of time values to solve the system of ODEs
INIT_TIME = 0; % seconds
FINAL_TIME = 5; % last time of projectile - seconds
time_interval = [INIT_TIME, FINAL_TIME];


%% Solve System of ODEs and return numeric results over time interval
```

```
[~, rocketVars] = ode45('ASEN2012_Project_2_dy_dt', time_interval, ...
                        init_conds);

% Set Trajectory Variables
x = rocketVars(:, 5);
z = rocketVars(:, 6);

% Find Range of Rocket
z_0 = 0; % looking for when rocket hits ground
near_x_0 = find(x > 30, 1, 'first');
diff_vec = abs(z(near_x_0:end) - z_0);
[~, idx] = min(diff_vec); % index of closest value
idx = idx + length(z(1:near_x_0 - 1)); % adjusting index

range = x(idx);

%% Plot 85m Solution
hFig = figure(NUM_SENSITIVITY_VARS + 1);
set(hFig, 'Position', [100 100 1600 900])

plot(x, z, 'r','LineWidth', 3)

ylim([0, inf])
xlim([0, inf])
new_legend_string = sprintf('Range of Rocket = %0.3g (m)', range);
legend(new_legend_string, 'location', 'northwest')
xlabel('Horizontal Distance (m)')
ylabel('Vertical Distance (m)')
title('Rocket Trajectory')
set(gca,'FontSize', 28)


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

%% Run and Plot Verification Case

% set which parameters to use in ODE45 based on whether you are doing
% sensitivity analysis, finding 85m params, or running a verification
   case
run_sens_analysis = false;
find_85m_range = false;
run_verification_case = true;

% set global bools that control parameter data flow
setGlobalBools(run_sens_analysis, find_85m_range, run_verification_case
   );

% Print Simulation Status to Command Window
fprintf('Running Verification Case.\n')

%% define initial conditions and flight variables

% Initialize variables, return initial conditions vector
[~, init_conds] = ASEN2012_Project_2_Input(run_sens_analysis, ...
```

```
                                                find_85m_range, ...
                                                run_verification_case);

% Definging the range of time values to solve the system of ODEs
INIT_TIME = 0; % seconds
FINAL_TIME = 5; % last time of projectile - seconds
time_interval = [INIT_TIME, FINAL_TIME];


%% Solve System of ODEs and return numeric results over time interval
[~, rocketVars] = ode45('ASEN2012_Project_2_dy_dt', time_interval, ...
                        init_conds);

% Set Trajectory Variables
x = rocketVars(:, 5);
z = rocketVars(:, 6);

% Find Range of Rocket
z_0 = 0; % looking for when rocket hits ground
near_x_0 = find(x > 30, 1, 'first');
diff_vec = abs(z(near_x_0:end) - z_0);
[~, idx] = min(diff_vec); % index of closest value
idx = idx + length(z(1:near_x_0 - 1)); % adjusting index

range = x(idx);

%% Plot Verification Case
hFig = figure(NUM_SENSITIVITY_VARS + 2);
set(hFig, 'Position', [100 100 1600 900])

plot(x, z, 'b', 'LineWidth', 3)

ylim([0, inf])
xlim([0, inf])
new_legend_string = sprintf('Range of Rocket = %0.3g (m)', range);
legend(new_legend_string, 'location', 'northwest')
xlabel('Horizontal Distance (m)')
ylabel('Vertical Distance (m)')
title('Rocket Trajectory - Verification Case')
set(gca,'FontSize', 28)
```

```
function [flight_params,...
          initial_conditions] = ASEN2012_Project_2_Input(use_globals,...
                                                         find_85m_range,
                                                             ...
                                                         do_verification
                                                            )

    %%% Input function where constants for project are set.

    %%% purpose: This function exists to pass data to both the main and
    %%%          dy_dt function. This allows the flight parameters for
    %%%          solution in ODE45 to be consolodated to one place and
    %%%          allows for easy switching of which flight parameters
```

```
    are
%%%           returned based on what type of simulation you are
    running.
%%%
%%% inputs: takes three bool inputs, which tell the function
    whether to
%%%          return the specific flight parameters for running the
%%%          sensitivity analysis, the 85m range finding, or the
%%%          verification case.
%%%
%%%
%%% outputs: returns a formatted vector of flight parameters needed
     in
%%%           each iteration of the dy_dt ODE45 function for the
%%%           calculation of the trajectory and a formatted vector
    of
%%%           initial conditions needed in ODE45 to tell it where to
%%%           begin solving the ODE.
%%%
%%%
%%% assumptions: assumes that all parameters will be set in SI
    units,
%%%               that water is incompressible, and that the gas in
    the
%%%               rocket is an Ideal Gas for at lest the beginning
    of
%%%               the flight.
%%%
%%%
%%% author's ID: 0dc91b091fd8
%%% date created: 11/25/2016
%%% date modified: 12/1/2016

%% Define Contants

g = 9.81; % [m/s^2]
C_discharge = 0.8; % discharge coefficient [dimensionless]
Rho_air_atm = 0.961; % ambient air density [kg/m^3]
V_bottle = 0.002; % volume of empty 2 litre bottle [m^3]
P_atm = 82943.93; % atmospheric pressure [Pa]
gamma = 1.4; % ratio of specific heats for air [dimensionless]
rho_water = 1000; % density of water [kg/m^3]
D_throat = 0.021; % diameter of throat [m]
D_bottle = 0.105; % diameter of bottle [m]
R_air = 287; % gas constant for air [J/kg/K]
m_bottle = 0.07; % mass of empty bottle [kg]
T_air_init = 300; % initial temperature of air [K]

A_throat = (pi/4) * D_throat ^ 2; % cross-sect area of throat [m^2]
A_bottle = (pi/4) * D_bottle ^ 2; % cross-sect area of bottle [m^2]

%% Define Initial Conditions

if use_globals
```

```
        % Set Global Variables for Sensitivity Analysis
        [global_vars] = GetGlobals;

        % Set the Value of the 4 Globals
        P_air_init = global_vars(1);
        m_water_init = global_vars(2);
        C_drag = global_vars(3);
        theta_init = global_vars(4);

        % re-calculate vol of water based on chosen mass of water
        V_water_init = m_water_init / rho_water; % initial vol.
                                                 % of water in bottle [
                                                    m^3]

    elseif do_verification % run simulation on verification case

        C_drag = 0.5; % drag coefficient [dimensionless]

        %%% Initial Absolute Pressure of Air in Bottle [Pa]
        P_gauge_air = 344738; % initial gauge pressure of air in bottle
           [Pa]
        P_air_init = P_atm + P_gauge_air;

        V_water_init = 0.001; % initial vol. of water in bottle [m^3]

        % Initial Mass of Water
        m_water_init = rho_water * V_water_init; % [kg]

        theta_init = 45 * (pi / 180); % theta [rad]

    elseif find_85m_range % use preset values for sensitivity params

        C_drag = 0.3; % drag coefficient [dimensionless]

        %%% Initial Absolute Pressure of Air in Bottle [Pa]
        P_gauge_air = 461949; % initial gauge pressure of air in bottle
           [Pa]
        P_air_init = P_atm + P_gauge_air;

        V_water_init = 0.001; % initial vol. of water in bottle [m^3]

        % Initial Mass of Water
        m_water_init = rho_water * V_water_init; % [kg]

        theta_init = 33.7 * (pi / 180); % theta [rad]

    end

    % upodate global conditions so that dydt uses preset values
    global_conditions = [P_air_init, m_water_init, C_drag, theta_init];
    setGlobals(global_conditions);

    %%% Initial Conditions of Tracked Variables
    Vel_init = 0; % initial velocity of the rocket [m/s]
    x_init = 0; % initial horizontal position [m]
```

```
    y_init = 0.1; % initial vertical position [m]

    % Initial Mass of Air in Bottle
    V_air_init = V_bottle - V_water_init; % [kg]
    m_air_init = (P_air_init * V_air_init) / (R_air * T_air_init); % [
        kg]


    %% Formatting Output

    flight_params = [g, C_discharge, V_bottle, P_atm, gamma,...
                     rho_water, A_throat, A_bottle, R_air, m_bottle, ...
                     V_air_init, Rho_air_atm, ...
                     T_air_init, m_air_init];

    initial_conditions = [Vel_init, m_air_init, m_water_init,...
                          theta_init, x_init, y_init, V_air_init];

end
```

```
function dydt = ASEN2012_Project_2_dy_dt(~, y)

    %%% Function for use in ODE45, calculates rate of change of launch
    %%% angle, the initial vol/mass of water, the drag coefficient, and
         the
    %%% inital air pressure in the rocket as functions of time.

    %%% purpose: This function calculates the rate of change of the
        tracked
    %%%          variables - launch angle, the initial vol/mass of
        water,
    %%%          the drag coefficient, and the inital air pressure in
        the
    %%%          rocket -  as functions of time. Formats its input and
    %%%          output for use in ODE45.
    %%%
    %%%
    %%% inputs: takes a vector y that contains all of the tracked
        variables
    %%%         that is used to calculate the new rate of change of all
         of
    %%%         these same tracked variables.
    %%%
    %%%
    %%% outputs: outputs a vector containing how each tracked variable
    %%%          changed with each iteration.
    %%%
    %%%
    %%% assumptions: assumes that the system is isentropic, that the
        gas
    %%%              can be treated as an Ideal Gas, that water is
    %%%              incompressible, and assumes steady atmospheric
    %%%              conditions for the duration of the flight.
    %%%
```

```
%%%
%%% author's ID: 0dc91b091fd8
%%% date created: 11/25/2016
%%% date modified: 12/1/2016

% sensitivity varibales are set each time sensitivity analysis is
   ran
[global_vars] = GetGlobals;

% Set the Value of the 4 Globals
P_air_init = global_vars(1);
C_drag = global_vars(3);

% determine wheter doing sensitivity analysis or not
[do_sens_analysis, do_85m_analysis, run_verif_case] = getGlobalBool
   ;

%% resolving y vec. into its constituent variables for readability
Velocity = y(1);
mass_air = y(2);
mass_water = y(3);
theta = y(4);
x_pos = y(5); % only here for debugging
z_pos = y(6); % only here for debugging
V_air = y(7);


%% Initializing Needed Parameters for Trajectory Calculation
[flight_params, ~] = ASEN2012_Project_2_Input(do_sens_analysis, ...
                                              do_85m_analysis, ...
                                              run_verif_case);

g = flight_params(1);
C_discharge = flight_params(2);
V_bottle = flight_params(3);
P_atm = flight_params(4);
gamma = flight_params(5);
rho_water = flight_params(6);
A_throat = flight_params(7);
A_bottle = flight_params(8);
R_air = flight_params(9);
mass_bottle = flight_params(10);
V_air_init = flight_params(11);
Rho_air_atm = flight_params(12);
T_air_init = flight_params(13);
m_air_init = flight_params(14);


%%% Make sure that air volume does not exceed bottle volume
if V_air > V_bottle
    V_air = V_bottle;
end

%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% Determining Flight Regime of Rocket and Related parameters

if V_air < V_bottle % Water Thrust Period of Flight

    % Air pressure at Time t
    P_air = P_air_init * (V_air_init / V_air) ^ gamma;

    % Exhaust Velocity
    Vel_exhaust = sqrt((2/rho_water) * (P_air - P_atm));

    % Mass flow rate of water out the Throat of the Bottle and of
       Air
    dmWater_dt = -C_discharge * rho_water * A_throat * Vel_exhaust;
    dmAir_dt = 0;

    % Thrust of Rocket
    F = 2 * C_discharge * (P_air - P_atm) * A_throat;

    %% Rate of Change of Volume of Air
    if V_air < V_bottle
        dVair_dt = C_discharge * A_throat * Vel_exhaust;
    else
        dVair_dt = 0;
    end

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

else % Air Thrust Period of Flight

    % Pressure and Temperature of air in bottle when all of the
       water
    % is expelled.
    P_end = P_air_init * (V_air_init / V_bottle) ^ gamma;
    T_end = T_air_init * (V_air_init / V_bottle) ^ (gamma - 1);

    % Air Pressure at Time t
    P_air_mass_eqn = P_end * (mass_air / m_air_init) ^ gamma;


    if P_air_mass_eqn > P_atm

        % Local Air Denisty & Air Temp
        rho_air = mass_air / V_bottle;
        T_air = P_air_mass_eqn / (rho_air * R_air);

        % Critical Pressure
        crit_pressure = P_air_mass_eqn * (2 / (gamma + 1)) ...
                                          ^ (gamma / (gamma - 1));

        %% Check for Choked Flow (M_exit = 1)
        if crit_pressure > P_atm % Choked Flow

            T_exit = (2 / (gamma + 1)) * T_air;
```

```
        rho_exit = crit_pressure / (R_air * T_exit);
        Velocity_exit = sqrt(gamma * R_air * T_exit);
        P_air_exit = crit_pressure;

    elseif crit_pressure <= P_atm % Not Choked

        % Exit air Pressure is ambient air pressure
        P_air_exit = P_atm;

        %%% Solve System of Equations %%%
        syms M_exit T_exit rho_exit
        press_eqn = ( P_air_mass_eqn / P_atm == (1 + ((gamma -
            1) / 2) * M_exit ^ 2) ^ (gamma / (gamma - 1)) );
        temp_eqn = ( T_exit / T_air == 1 + ((gamma - 1) / 2) *
            M_exit ^ 2 );
        rho_eqn = ( rho_exit == P_atm / (R_air * T_exit) );
        solution = solve(press_eqn, temp_eqn, rho_eqn, M_exit,
            T_exit, rho_exit);

        %%% Find Exit Mach Number %%%
        M_exit_indeces_real = find(imag(solution.M_exit) == 0);
        M_exit_real = double(solution.M_exit(
            M_exit_indeces_real));

        M_exit_indeces_positive = M_exit_real > 0;
        M_exit = M_exit_real(M_exit_indeces_positive);


        %%% Find Exit Temperature %%%
        T_exit_indeces_real = find(imag(solution.T_exit) == 0);
        T_exit_real = double(solution.T_exit(
            T_exit_indeces_real));

        T_exit_no_Duplicates = unique(T_exit_real);
        if length(T_exit_no_Duplicates) > 1
            error('duplicates homie');
        end
        T_exit = unique(T_exit_real);


        %%% Find Exit Air Density %%%
        rho_exit_indeces_real = find(imag(solution.rho_exit) ==
            0);
        rho_exit_real = double(solution.rho_exit(
            rho_exit_indeces_real));

        rho_exit_no_Duplicates = unique(rho_exit_real);
        if length(rho_exit_no_Duplicates) > 1
            error('duplicates homie');
        end
        rho_exit = unique(rho_exit_real);

        % Find exit Velocity
        Velocity_exit = M_exit * sqrt(gamma * R_air * T_exit);
```

```
                if isempty(Velocity_exit)
                    error('All imaginary Solutions')
                end

            end

            % Volume of Air no longer is changing, V_air == V_bottle
            dVair_dt = 0;

            %% Rate of Change Air Mass, Water Mass
            dmAir_dt = -C_discharge * rho_exit * A_throat *
               Velocity_exit;
            dmWater_dt = 0;

            %% Calculate Thrust
            F = -dmAir_dt * Velocity_exit + (P_air_exit - P_atm) *
               A_throat;


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

        else % Ballistic (No Thrust Period) Phase

            F = 0; % no thrust anymore :'(

            % no change in amount of water/air in bottle
            dmAir_dt = 0;
            dmWater_dt = 0;

            % Volume of Air no longer is changing, V_air == V_bottle
            dVair_dt = 0;

        end


    end

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

    %% Drag
    Drag = (Rho_air_atm / 2) * (Velocity ^ 2) * C_drag * A_bottle;


    %% Basic Governing Equations

    % Mass of Rocket
    if mass_water < 0
        mass_water = 0;
    end

    mass_rocket = mass_bottle + mass_air + mass_water;

    dVel_dt = (F - Drag - mass_rocket * g * sin(theta)) / mass_rocket;
```

```
    if Velocity > 1
        dTheta_dt = (1/Velocity) * -g * cos(theta);
    else
        dTheta_dt = 0;
    end

    dx_dt = Velocity * cos(theta);
    dz_dt = Velocity * sin(theta);

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

    %% Assign the change in y to the dydt vector for ode45

    dydt(1) = dVel_dt;
    dydt(2) = dmAir_dt;
    dydt(3) = dmWater_dt;
    dydt(4) = dTheta_dt;
    dydt(5) = dx_dt;
    dydt(6) = dz_dt;
    dydt(7) = dVair_dt;

    dydt = dydt';

end
```

```
function setGlobals(global_conditions)

    %%% function that sets the global variation parameters

    %%% purpose: function allows the global variation parameters:
       launch
    %%%          angle, the initial vol/mass of water, the drag
    %%%          coefficient, and the inital air pressure in the rocket
        to
    %%%          be set from anywhere in any of the functions or
       scripts in
    %%%          use.
    %%%
    %%%
    %%% inputs: takes a vector of numeric values for the variation
    %%%         parameters that you want to assign to their respective
    %%%         global vars.
    %%%
    %%%
    %%% outputs: no defined outputs, but will set the 4 variation
       global
    %%%          variable values for the entire workspace.
    %%%
    %%%
    %%% assumptions: assumes that the vector of global conditions is
    %%%              formatted properly so that the right values can be
    %%%              assigned to the right variables.
```

```
    %%%
    %%%
    %%% author's ID: 0dc91b091fd8
    %%% date created: 11/25/2016
    %%% date modified: 12/1/2016

    % define pair i the initial pressure of air (the limit being the
       burst
    % pressure of the bottle, with some factor of safety), the initial
    % volume fraction (or equivalently initial mass) of water, the drag
    % coefficient and the launch angle as global variables to run the
    % simulation multiple times, all varying these paramters each time
    global P_air_init m_water_init C_drag theta_init

    % Set the Value of the 4 Globals
    P_air_init = global_conditions(1);
    m_water_init = global_conditions(2);
    C_drag = global_conditions(3);
    theta_init = global_conditions(4);


end
```

```
function [global_vars] = GetGlobals

    %%% function that gets the global variation parameters

    %%% purpose: function allows the global variation parameters:
       launch
    %%%          angle, the initial vol/mass of water, the drag
    %%%          coefficient, and the inital air pressure in the rocket
        to
    %%%          be passed to any other function or script without the
    %%%          other functions knowing about the variables
    %%%
    %%%
    %%% inputs: takes no inputs.
    %%%
    %%%
    %%% outputs: returns a vector of numeric values for the variation
    %%%          parameters.
    %%%
    %%%
    %%% assumptions: assumes that the global vars have already been set
    %%%              before asssigning them to anything.
    %%%
    %%%
    %%% author's ID: 0dc91b091fd8
    %%% date created: 11/25/2016
    %%% date modified: 12/1/2016

    % define pair i the initial pressure of air (the limit being the
       burst
    % pressure of the bottle, with some factor of safety), the initial
```

```
    % volume fraction (or equivalently initial mass) of water, the drag
    % coefficient and the launch angle as global variables to run the
    % simulation multiple times, all varying these paramters each time
    global P_air_init m_water_init C_drag theta_init

    % return current global variable values
    global_vars = [P_air_init, m_water_init, C_drag, theta_init];

end
```

```
function setGlobalBools(bool_in_1, bool_in_2, bool_in_3)

    %%% function that sets the global boolean parameter control values

    %%% purpose: function allows the global bool variables: "if you are
    %%%          running a sensitivity analysis", "if you are trying to
    %%%          find the 85m range paramters", and "if you are running
        the
    %%%          verification case" to be set from anywhere in any of
       the
    %%%          functions or scripts in use. This allows the user to
    %%%          change which parameters are passed to the dy_dt
       function
    %%%          used ODE45.
    %%%
    %%%
    %%% inputs: takes the bool values for the bool
    %%%         parameters that you want to assign to their respective
    %%%         global vars.
    %%%
    %%%
    %%% outputs: no defined outputs, but will set the 3 bool global
    %%%          variable values for the entire workspace.
    %%%
    %%%
    %%% assumptions: assumes that the input of bool global conditions
       is
    %%%              formatted properly so that the right values can be
    %%%              assigned to the right variables.
    %%%
    %%%
    %%% author's ID: 0dc91b091fd8
    %%% date created: 11/25/2016
    %%% date modified: 12/1/2016

    global is_doing_sensitiv_analysis find_85m_range do_verification

    % Set whether running sensitivty analysis, 85m optimization, or are
    % running a verification case
    is_doing_sensitiv_analysis = bool_in_1;
    find_85m_range = bool_in_2;
    do_verification = bool_in_3;

end
```

```
function [bool_out_1, bool_out_2, bool_out_3] = getGlobalBool

    %%% function that gets the global boolean parameter control values

    %%% purpose: function allows the global bool variables: "if you are
    %%%          running a sensitivity analysis", "if you are trying to
    %%%          find the 85m range paramters", and "if you are running
     the
    %%%          verification case" to be passed to anywhere in any of
    the
    %%%          functions or scripts in use. This allows the user to
    get
    %%%          the parameters that control which flight params are
    passed
    %%%          to the dy_dt function used ODE45.
    %%%
    %%%
    %%% inputs: no reauired inputs.
    %%%
    %%%
    %%% outputs: returns the bool values for the bool
    %%%          parameters for use in other functions.
    %%%
    %%%
    %%% assumptions: assumes that the global vars have already been set
    %%%              before asssigning them to anything.
    %%%
    %%%
    %%% author's ID: 0dc91b091fd8
    %%% date created: 11/25/2016
    %%% date modified: 12/1/2016

    global is_doing_sensitiv_analysis find_85m_range do_verification

    % return the current setting for which paramters are
    % used in ODE45
    bool_out_1 = is_doing_sensitiv_analysis;
    bool_out_2 = find_85m_range;
    bool_out_3 = do_verification;

end
```