```matlab
%{
Author: Alex Godbout
Assignment: ASEN2012 Project 2
Creation Date: 11/21/2024
Inputs: Initial Bottle Parameters (Begins on line 21)
Outputs: Trajectory and Thrust of Bottle Rocket
Purpose: Devlope a model that can be used to find the ideal starting
parameters for a bottle rocket to acheive maximum distance traveled.
%}


clear;
clc;
close all;
%--------------------------%
%---Load Data/Constants----%
%--------------------------%

load("project2verification (1).mat")
const = constFunc;
statevector_0 = initialFunc(const);
tspan = [0, 5]; %

%---------------%
%---Run ode45---%
%---------------%

[t, stateVectors] = ode45(@(t, stateVector) d_State_dt(t, stateVector,
const), tspan, statevector_0);
stateVectors(:,8) = t;
bottle_distance = stateVectors(:,1);
bottle_height = stateVectors(:,3);
%Extract Intermediate Values
Vars = zeros(height(stateVectors), 4);
for i=1:height(stateVectors)
    [~, Vars(i,:)] = StateFunction(t, stateVectors(i,:),const);
end
Thrust = Vars(:,1);

%------------------%
%---Process Data---%
%------------------%

maxThrust = max(Thrust);
maxDistance = max(bottle_distance);
maxHeight = max(bottle_height);
[answers.XmaxHeight,~] = find(bottle_height == maxHeight);
answers.XmaxHeight = bottle_distance(answers.XmaxHeight);

%---------------%
%---Plot Data---%
%---------------%
```

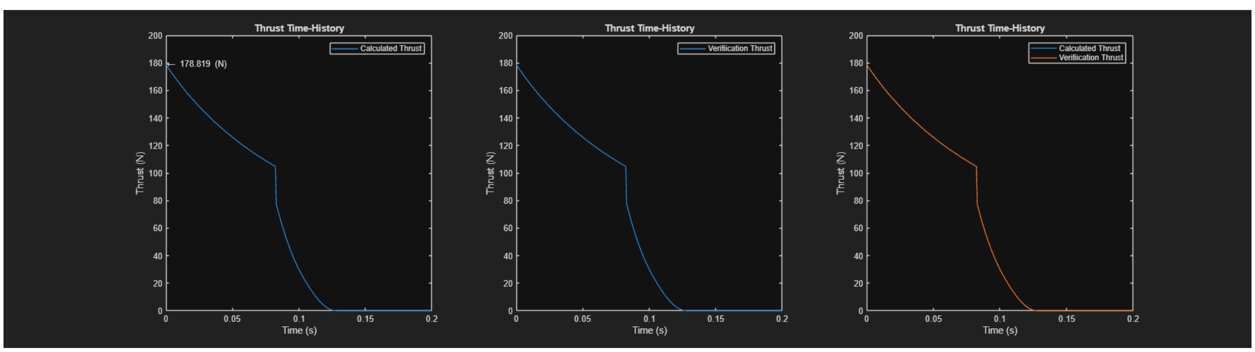
```
%Thrust Plots
figure; hold on;
set(gcf, 'Units', 'Normalized', 'Position', [0, .04, 1, .48]);

subplot(1,3,2)
plot(verification.time, verification.thrust)
axis([0,0.2,0,200])
xlabel('Time (s)')
ylabel('Thrust (N)')
title('Thrust Time-History')
legend('Verifiication Thrust')


subplot(1,3,1);
plot(t,Thrust);
axis([0,0.2,0,200])
xlabel('Time (s)')
ylabel('Thrust (N)')
title('Thrust Time-History')
legend('Calculated Thrust')
txt = '\leftarrow ' + string(round(maxThrust,3)) + ' (N)';
text(0,maxThrust,txt)

subplot(1,3,3); hold on
box on;
plot(t,Thrust);
plot(verification.time, verification.thrust)
axis([0,0.2,0,200])
xlabel('Time (s)')
ylabel('Thrust (N)')
title('Thrust Time-History')
legend( 'Calculated Thrust' ,'Verifiication Thrust')

%Trajectory Plots
figure; hold on;
set(gcf, 'Units', 'Normalized', 'Position', [0, .52, 1, .46]);

subplot(1,3,2);
plot(verification.distance,verification.height);
xlabel("Distance (m)")
ylabel("Height (m)")
title('Flight Trajectory')
legend('Verifiication', Location='northwest')


subplot(1,3,1); hold on;
plot(bottle_distance, bottle_height);
xlabel("Distance (m)")
ylabel("Height (m)")
title('Flight Trajectory')
legend('Calcilated', Location='northwest')

txt = '\uparrow Max Height (' + string(round(answers.XmaxHeight,2)) + 'm,' +
```

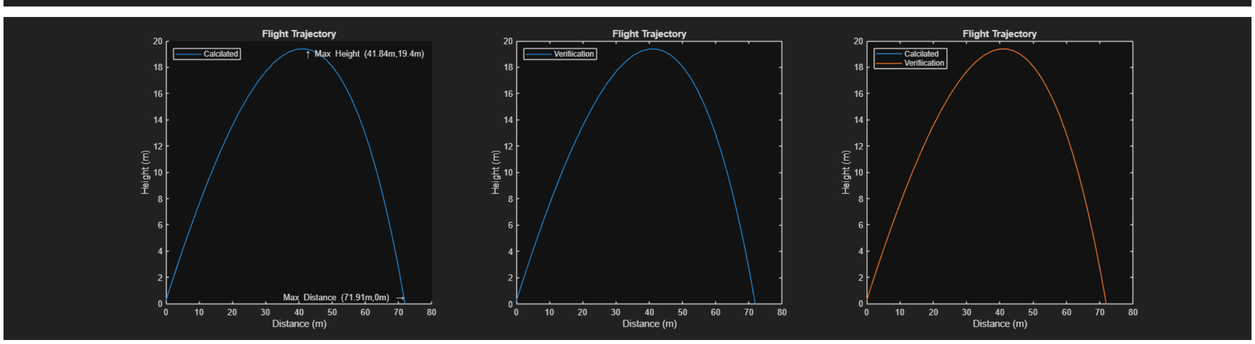
```
string(round(maxHeight,2)) + 'm)';
text(answers.XmaxHeight,maxHeight,txt,"HorizontalAlignment","left",VerticalAl
ignment="top")
txt =  'Max Distance (' + string(round(maxDistance,2)) +'m,0m) \rightarrow';
text(maxDistance,0,txt,"HorizontalAlignment","right","VerticalAlignment","bot
tom")

subplot(1,3,3);
box on;
plot(bottle_distance, bottle_height,
verification.distance,verification.height);
xlabel("Distance (m)")
ylabel("Height (m)")
title('Flight Trajectory')
legend('Calcilated', 'Verifiication', Location='northwest')


% Function to model the state change
function [d_statevector_dt, linVars] = StateFunction(t, stateVector, const)
    % Extract state variables
    x = stateVector(1);
    v_x = stateVector(2);
    z = stateVector(3);
    v_z = stateVector(4);
    mTotal = stateVector(5);
    Vol_air = stateVector(6);
    massAir = stateVector(7);


    F_Thrust = 0;
    F_Grav = [0 -mTotal*const.g];
    F_Normal = 0;
    d_Vair = 0;
    d_mAir = 0;
    d_mTot = 0;

    v_exit = 0;
    rho_exit = 0;
    M = 0;
    %---On the Stand---%
    if x < const.l_s*cosd(const.theta_0)

            h = [cosd(const.theta_0) sind(const.theta_0)];
            h = h/norm(h);

            %Didnt Need this lmao, I think it should be included tho
            %Account for Normal Force of Launcher
            %magNorm = abs(F_Grav(2)/cosd(const.theta_0));
            %F_Normal = [-magNorm*sind(const.theta_0)
magNorm*cosd(const.theta_0)];

    else %Off Launcher
            h = [v_x v_z];
            h = h/norm(h);
```

```
    end

    %Always calculate so p_end can be used in Stage 2 if statement
    p_end = const.p_0*(const.vAir_0/const.volumeBottle)^(const.gamma);

    %---Stage 1---%
    if Vol_air < const.volumeBottle

        p = (const.p_0)*((const.vAir_0)/Vol_air).^const.gamma;
        %Calculate the Change in Volume
        %Latex form p_{\text{air}}' = C_d \cdot A_t \cdot \sqrt{\frac{2}
{\rho_w} \left( p_0 \left( \frac{V_{w,0}}{V_{\text{air}}} \right)^\lambda -
p_{\text{atm}} \right)}
        d_Vair = (const.c_dis)*(const.A_Throat)*sqrt(2/(const.rhoWater)*(p-
const.atomosphericPressure));
        d_mTot = -d_Vair*const.rhoWater;

        %Calculate Net Force
        F_Thrust = 2*(const.c_dis)*(const.A_Throat)*(p-
const.atomosphericPressure);

    %---Stage 2---%
    elseif p_end*(massAir/const.mAir_0)^(const.gamma) >
const.atomosphericPressure


        %Compute State
        T_end = const.T_0*(const.vAir_0/const.volumeBottle)^(const.gamma-1);
        p = p_end*(massAir/const.mAir_0)^(const.gamma);
        rho_air = massAir/const.volumeBottle;
        T = p/(rho_air*const.Rair);
        %Type of Flow
        p_star = p*(2/(const.gamma+1)).^(const.gamma/(const.gamma-1));
        %Choked
        if p_star > const.atomosphericPressure
            p_exit = p_star;
            T_exit = T*(2/(const.gamma+1));
            rho_exit = p_exit/(const.Rair*T_exit);
            v_exit = sqrt(const.gamma*const.Rair*T_exit);
        %Unchoked
        else
            p_exit = const.atomosphericPressure;
            M = sqrt( 2/(const.gamma-1) * ((p/
const.atomosphericPressure)^((const.gamma-1)/(const.gamma))-1));
            T_exit = T/(1+(const.gamma-1)/2*M^2);
            rho_exit = const.atomosphericPressure/(const.Rair*T_exit);
            v_exit = M*sqrt(const.gamma*const.Rair*T_exit);
        end

        d_mAir = -const.c_dis*rho_exit*const.A_Throat*v_exit;
        F_Thrust = (-d_mAir*v_exit + (p_exit-
const.atomosphericPressure)*const.A_Throat);
        d_mTot = d_mAir;
    end
```

```
    %---Stage 3---%
    %Because the only force in stage 3 is the drag force and drag force is
    %always implemtned we do not have a stage 3 if statement

    %Always account for drag
    F_Drag = 1/2*const.rhoAir*norm([v_x v_z])^2*const.c_D*const.A_Bottle;

    %Add a Normal force to 'model' impact force
    if z <=0
        F_Normal = -h *(v_x^2 +v_z^2)*1000;
    end

    %Calculate Net Force and Acceleration (h breaks thrust and drag into x
and z components)
    F_Net = h*(F_Thrust - F_Drag) + F_Normal + F_Grav;
    acceleration = F_Net/mTotal;


    linVars = [F_Thrust, v_exit , rho_exit,M];
    d_statevector_dt = [ v_x ;  acceleration(1)  ; v_z ; acceleration(2)   ;
d_mTot;d_Vair ; d_mAir];
end

function const = constFunc

%Control Variables
const.vWater_0 = 0.0005;%M^3
const.theta_0 = 40; %Degrees
const.p_0 = 330948.34944+83426.5; %N/m

%Posible Adjustable Variables
const.A_Throat =(.021)^2/4*pi; %m
const.A_Bottle = (.105)^2/4*pi; %m
const.T_0 = 310;% K

%Position Vars
const.x_0 = 0; %m
const.z_0 = .25;%m
const.v_0 = 0;% m/s

%Launcher Vars
const.volumeBottle = 0.002;% m^3
const.l_s = 0.5;% m
const.massBottle = 0.15;% kg

%Constants
const.gamma = 1.4;
const.atomosphericPressure = 83426.563088; %N/m^2
const.rhoAir = 0.961;% kg/m^3
const.g = 9.81;% m/s^2
const.rhoWater = 1000;% kg/m^3
const.Rair = 287;% J/(kg*K)
const.c_D = 0.425;
```

```
const.c_dis = 0.78;

%Helpful For Calculations
const.vAir_0 = const.volumeBottle-const.vWater_0;
const.mAir_0 = (const.vAir_0*const.p_0)/(const.T_0*const.Rair);
const.mass_0 = const.massBottle + const.vWater_0*const.rhoWater +
const.mAir_0;
end

function initial = initialFunc(const)
initial = [const.x_0 const.v_0 const.z_0 const.v_0 const.mass_0 const.vAir_0
const.mAir_0];
end

%Function Des
function compareData(MyData, TheirData, t_Range, dataName)
    figure; hold on
    axis([t_Range,0,1]);

    plot(MyData(:,1),MyData(:,2));
    plot(TheirData(:,1),TheirData(:,2));
    xlabel(dataName(1));
    ylabel(dataName(2));
    legend('Experimental Data', 'Verification Data');
    title(dataName(3));
end

%Function to seperate linear variables and derivatives for ode45
function d_statevector_dt=d_State_dt(t, stateVector, const)
    [d_statevector_dt,~] = StateFunction(t, stateVector, const);
end
```





Published with MATLAB® R2024a