Table of Contents


```
% Author(s): Andrew Patella
% Assignment title: Project 2
% Purpose: Model flight of bottle rocket using ode45
% Creation date: 11/06/2023
% Revisions: N/A

close all; clear; clc;
```

Constants and initial conditions

```
% Storing constants in a struct using the getConst() function
const = getConst();

% Calculating initial velocities in x and z directions for initial state
%Since the total velocity is zero this doesn't really matter but for
%completeness I included it

vx0 = const.v_0*cos(const.theta_i);
vz0 = const.v_0*sin(const.theta_i);

% Creating a column vector of initial conditions
initial_conditions = [const.x_0 ; vx0 ; const.z_0 ; vz0 ; const.m_0tot ;
 const.V_0a ; const.m_0a];

%Integration time vector
int_time = [0 5];

%Function handle of state matrix function
int_fun = @(t,state) state_matrix_func(const,t,state);
```

Integration and Thrust Calculation

```
%Integrating using ode45
[tout,stateOut] = ode45(int_fun,int_time,initial_conditions);

%Calculating the thrust and stage using a for loop
thrust = zeros(length(tout),1);
```

```
stage = zeros(length(tout),1);

for i = 1:length(tout)
    [~,thrust(i),stage(i)] = state_matrix_func(const,tout(i),stateOut(i,:));
end
```

Verification data

```
verification = load('project2verification.mat');
time_verification = verification.verification.time;
height_verification = verification.verification.height;
distance_verification = verification.verification.distance;
thrust_verification = verification.verification.thrust;
velocity_x_verification = verification.verification.velocity_x;
velocity_z_verification = verification.verification.velocity_y;
volume_air_verification = verification.verification.volume_air;
```

Stage calculations

```
%Index where the change occurs
stage2 = find(stage==2,1);
stage3 = find(stage==3,1);
stage4 = find(stage==4,1);

% The transitions between stages in time
transition1_time = tout(stage2);
transition2_time = tout(stage3);
transition3_time = tout(stage4);

%The transition between stages in x position
transition1_x = stateOut(stage2,1);
transition2_x = stateOut(stage3,1);
transition3_x = stateOut(stage4,1);
```

Maxima of rocket states

```
maxX = max(stateOut(:,1));
maxZ = max(stateOut(:,3));
maxF = max(thrust);

fprintf('Max horizontal distance: %2.2f m\n',maxX);
fprintf('Max vertical distance: %2.2f m\n',maxZ);
fprintf('Max thrust: %2.2f N\n',maxF);
```

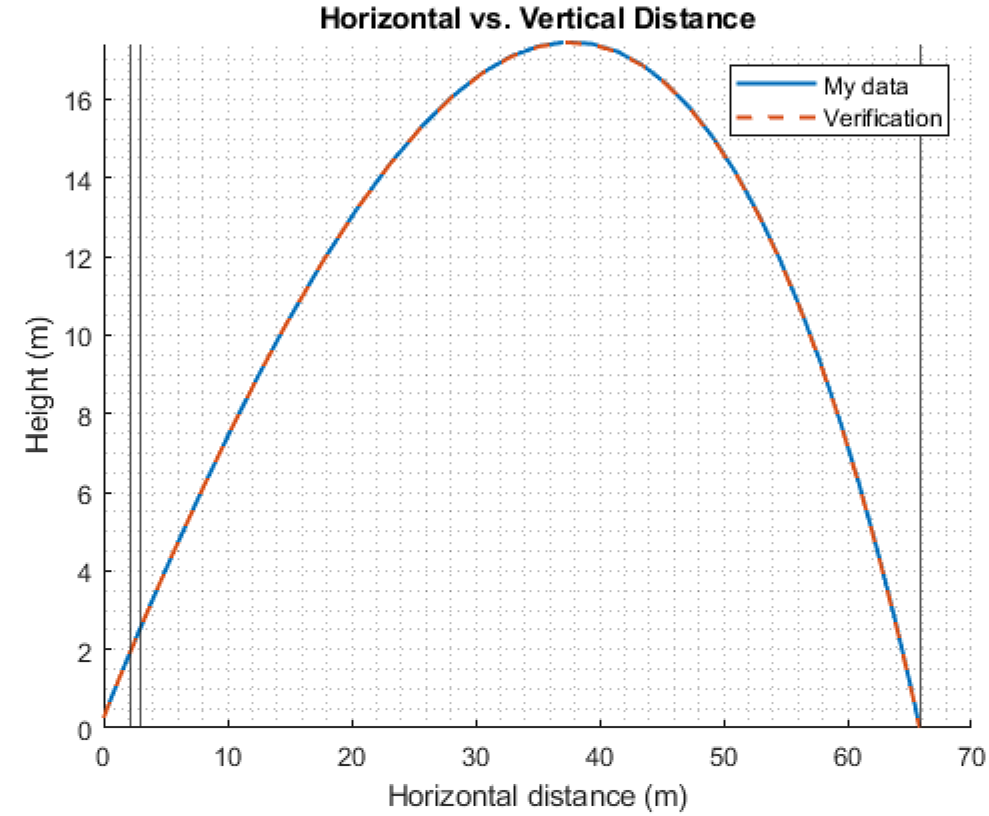
```
Max horizontal distance: 65.83 m
Max vertical distance: 17.46 m
Max thrust: 198.69 N
```

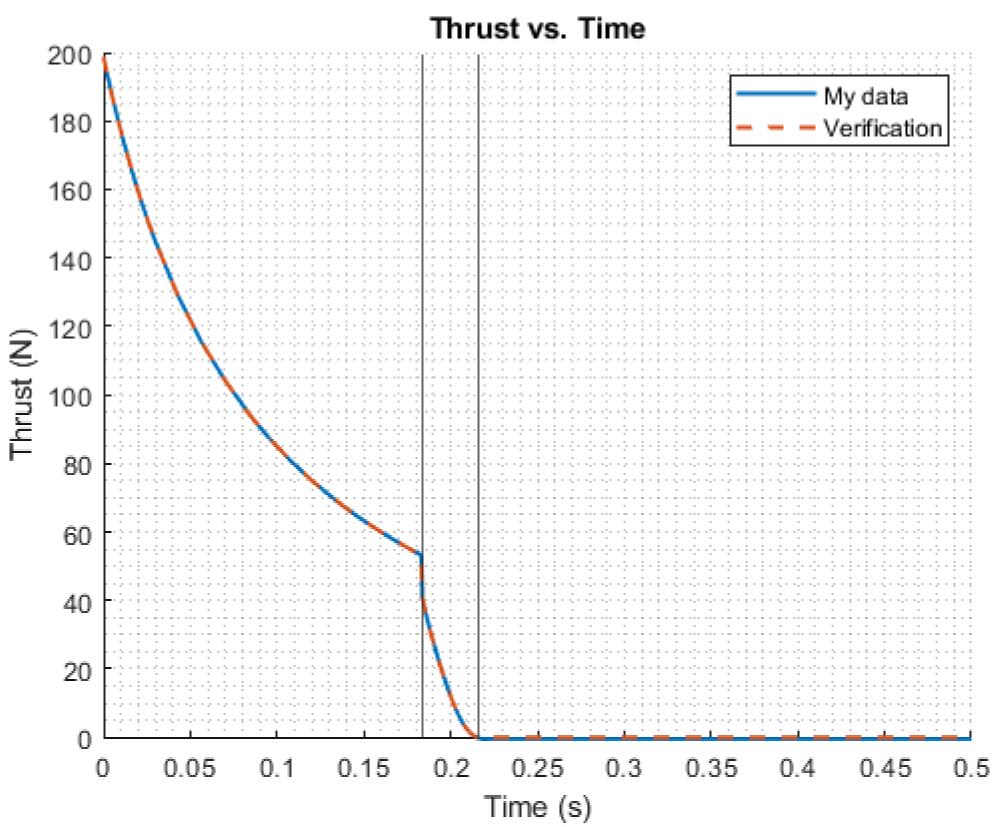
Plots

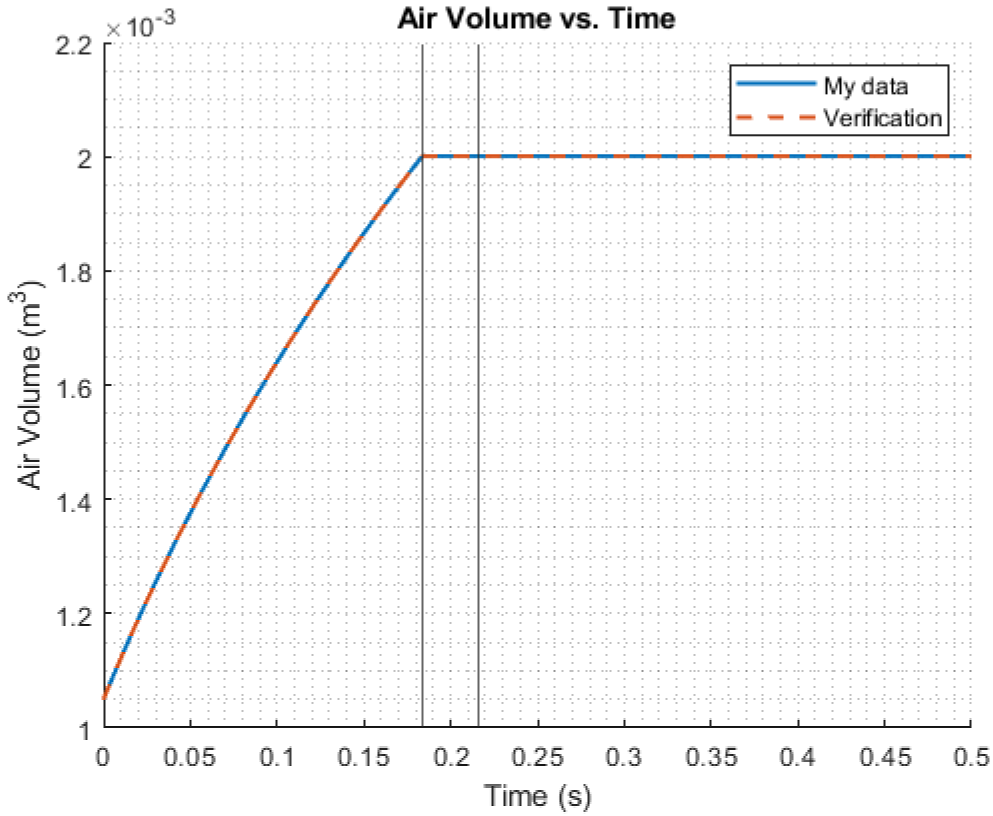
```
figure(1);
%POSITION PLOT
```

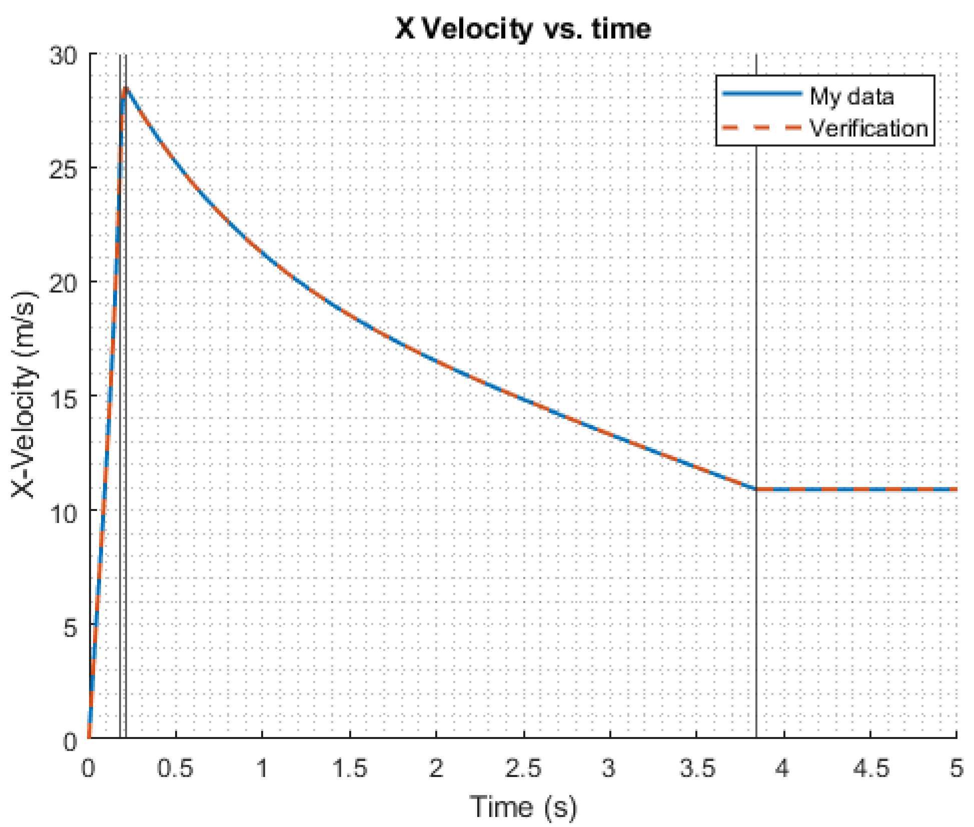
```
hold on;
plot(stateOut(:,1),stateOut(:,3),'LineWidth',1.5);
plot(distance_verification,height_verification,'--','LineWidth',1.5);
xline(transition1_x);
xline(transition2_x);
xline(transition3_x);

grid minor;
legend('My data','Verification');
ylabel('Height (m)');
xlabel('Horizontal distance (m)');
title('Horizontal vs. Vertical Distance');
ylim([0,max(height_verification)]);
hold off;



figure(2);
%THRUST PLOT
hold on;
plot(tout,thrust,'linewidth',1.5);
plot(time_verification,thrust_verification,'--','LineWidth',1.5);
xline(transition1_time);

xline(transition2_time);
xline(transition3_time)
legend('My data','Verification');
ylabel('Thrust (N)');
xlabel('Time (s)');
grid minor;
xlim([0,0.5]);
title('Thrust vs. Time');
hold off;


figure(3);
%AIR VOLUME PLOT
hold on;
plot(tout,stateOut(:,6),'linewidth',1.5);
plot(time_verification,volume_air_verification,'--','linewidth',1.5);
xline(transition1_time);
xline(transition2_time);
xline(transition3_time);
grid minor;
ylabel('Air Volume (m^3)');
xlabel('Time (s)');
legend('My data','Verification');
xlim([0,0.5]);
title('Air Volume vs. Time')
hold off;


figure(4);
%X VELOCITY PLOT
```

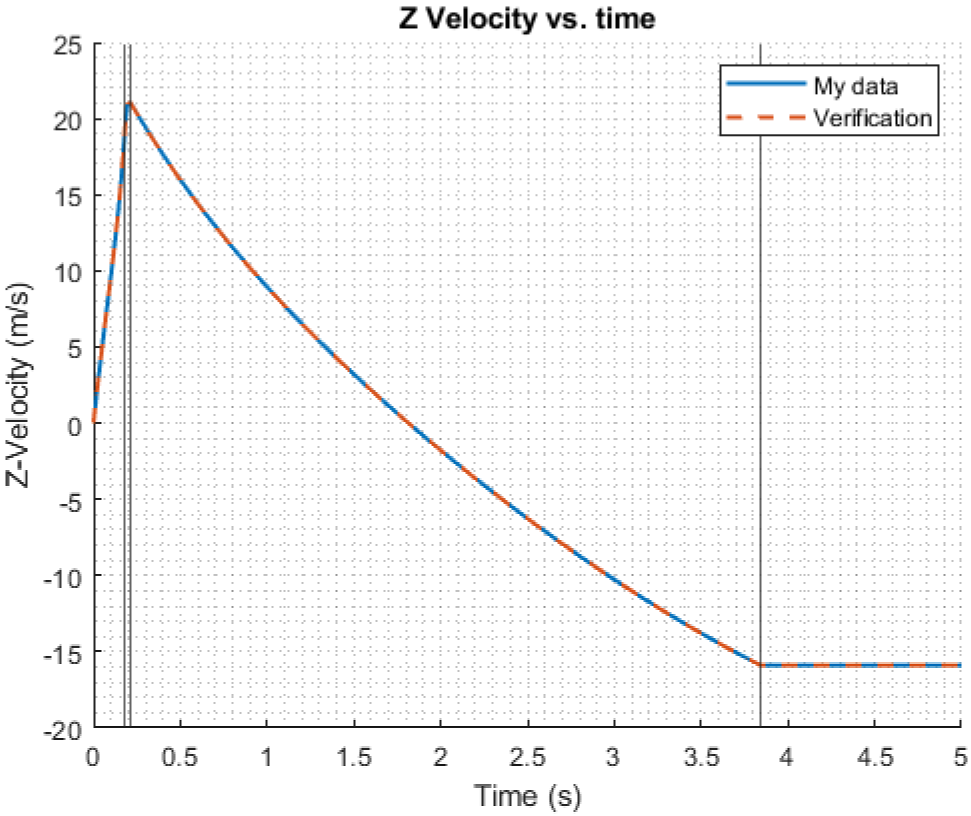
```
hold on;
plot(tout,stateOut(:,2),'linewidth',1.5);
plot(time_verification,velocity_x_verification,'--','LineWidth',1.5);
xline(transition1_time);
xline(transition2_time);
xline(transition3_time);
legend('My data','Verification');
grid minor;
xlabel('Time (s)');
ylabel('X-Velocity (m/s)');
title('X Velocity vs. time');
hold off;



figure(5);
%Z VELOCITY PLOT
hold on;
plot(tout,stateOut(:,4),'linewidth',1.5);
plot(time_verification,velocity_z_verification,'--','LineWidth',1.5);
xline(transition1_time);
xline(transition2_time);
xline(transition3_time);
grid minor;
xlabel('Time (s)');
ylabel('Z-Velocity (m/s)');
legend('My data', 'Verification');
title('Z Velocity vs. time');
hold off;
```


Horizontal vs. Vertical Distance
My data
Verification
Height (m)
Horizontal distance (m)
0
2
4
6
8
10
12
14
16
0
10
20
30
40
50
60
70


Thrust vs. Time
My data
Verification
Thrust (N)
Time (s)
0
20
40
60
80
100
120
140
160
180
200
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5


Air Volume vs. Time
×10^-3
2.2
2
1.8
1.6
1.4
1.2
1
Air Volume (m^3)
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Time (s)
My data
Verification


X Velocity vs. time
30
25
20
15
10
5
0
X-Velocity (m/s)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Time (s)
My data
Verification



FUNCTIONS

```
function const = getConst()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This function creates a structure of initial conditions and relevant
% constants for the main function. This truncates the state_matrix_func
% inputs by taking in only this structure.

% No inputs are required for the function.
% Output is a structure with multiple different entries

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

    const.g = 9.807; %m/s^2 (gravitational constant)
    const.c_dis = 0.8; %(Discharge constant)
    const.rho_air = 0.961; %kg/m^3 (Density of ambient air)
    const.V_b = 0.002; %m^3 (Volume of bottle)
    const.p_atm = 12.1 * 6894.76; %psia to Pa (pressure of atmosphere)
    const.gamma = 1.4; %unitless (specific heat reatio constant)
    const.rho_w = 1000; %kg/m^3 (density of water)
    const.d_e = 2.1; %cm (diameter of exit)
    const.d_b = 10.5; %cm (diameter of bottle)
    const.R_air = 287; %J/kgK (Gas constant for air)
    const.m_b = 0.15; %kg (mass of bottle)
```

```
    const.c_D = 0.48; %Coefficient of drag)
    const.p_0 = 52* 6894.76 + const.p_atm; %psig to Pa (initial pressure in
 bottle) 52* 6894.76 + const.p_atm
    const.V_0w = 0.00095; %m^3 (Initial volume of water)
    const.T_0 = 300; %k (Initial temperature of air)
    const.v_0 = 0.0; %m/s (initial velocity)
    const.theta_i = 42*(pi/180); % degrees to radian (launch angle)
    const.x_0 = 0; %m (initial x position)
    const.z_0 = 0.25; %m (initial z position)
    const.l_s = 0.5; %m (length of launch stand)


    % Calculating other necessary constants and initial conditions
    const.At  =   pi*((const.d_e/2)*0.01)^2; %m^2 (Cross sectional area of
 throat)
    const.Ab  =   pi*((const.d_b/2)*0.01)^2; %m^2 (Cross sectional area of
 bottle)

    const.m_0w = const.rho_w * const.V_0w; %kg (initial mass of water)
    rho_0a = const.p_0/(const.R_air*const.T_0); %kg/m^3 (initial density of
 air)
    const.V_0a = const.V_b - const.V_0w; %m^3 (Initial volume of air)
    const.m_0a = const.V_0a * rho_0a; %kg (initial mass of air)

    const.m_0tot = const.m_b + const.rho_w * (const.V_b - const.V_0a) +
 (const.p_0*const.V_0a)/(const.R_air*const.T_0);
end

function [state_matrix,F,stage] = state_matrix_func(const,~,state)

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

% Function that takes in the current state of the rocket, and has
% equations for calculating the derivatives based on current state

% This function will output the derivatives of each part of the state
% matrix

% state_matrix = d/dt[x;vx;z;vz;mr;V;m_air]

% This function outputs the values of each of these derivatives for ode45,
% and F, the thrust, and stage, an integer value of the stage the rocket is
% in.

% This function requires input of the constant structure, time, and the
% current state since the derivatives are dependant on the current state.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %Constants
   %These come from the const struct that is inputted

    g = const.g; %m/s^2
    c_dis = const.c_dis;
    rho_air = const.rho_air; %kg/m^3
```

```
    V_b = const.V_b; %m^3
    p_atm = const.p_atm; %pa
    gamma = const.gamma;
    rho_w = const.rho_w; %kg/m^3
    R_air = const.R_air; %J/kgK
    c_D = const.c_D;
    p_0 = const.p_0; %pa
    theta_i = const.theta_i; %degree
    z0 = const.z_0; %m
    l_s = const.l_s; %m
    At = const.At; %m^2
    Ab = const.Ab; %m^2
    V_0a = const.V_0a; %m^3
    m_0a = const.m_0a; %kg

    % Lowercase v is velocity, uppercase V is  volume

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

    %Devectorizing the State matrix
    x1 = state(1);
    x2 = state(2);

    z1 = state(3);
    z2 = state(4);

    m_r = state(5);
    V_air = state(6);
    m_air = state(7);

     %Calculating distance and theta, to see if the bottle is of the launch
     %stand, and locking theta if it is.
    distance = sqrt(x1^2+(z1-z0)^2);

    if distance < l_s
        theta = theta_i;
    else
        theta = atan(z2/x2);
    end

    %Calculating dynamic pressure and drag at any point in the integration
    v_bottle = sqrt(x2^2+z2^2);
    q = 0.5 * rho_air * (v_bottle^2);
    drag = q*Ab*c_D;



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

Stages of flight

```
    if V_air < V_b  % Stage 1: Water expulsion
```

```
        %Calculating pressure of air
        p = p_0*(V_0a/V_air)^gamma;


        %Calculating state of exhaust
        v_e = sqrt(2*(p-p_atm)/rho_w); %Velocity of the exhausted water

        %calulating mass flow rate of water
        mdot_w = c_dis * rho_w * At * v_e;

        %Calculating thrust
        F = mdot_w * v_e;  %Thrust created by exhausted water

        %Calculating the time rate of change of water
        dVdt = c_dis * At * v_e;

        %Finding net forces in x and z directions
        FnetZ = F*sin(theta)-drag*sin(theta)-m_r*g;
        FnetX = F*cos(theta)-drag*cos(theta);

        ddt1 = x2; %x Velocity
        ddt2 = FnetX/m_r; %Thrust - drag
        ddt3 = z2; %Z velocity
        ddt4 = FnetZ/m_r;  %Thrust-drag-gravity
        ddt5 = -mdot_w; % Current total mass change in kg/s
        ddt6 = dVdt; %Change in volume of air
        ddt7 = 0; %Change in mass of air
        %disp('Water expulsion');
        stage = 1;

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

    elseif V_air >= V_b   % Stages 2-4


        %Calculating P end and T end for calculating pressure later
        p_end = p_0*(V_0a/V_b)^gamma;


        %Calculating density, Temperature, and pressure of air
        rho = m_air/V_b;
        p = p_end*(m_air/m_0a)^gamma;
        T = p/(rho*R_air);

        %these are the actual current temperature and density

        %Determining if the flow is choked to find v_e
        p_star = p*(2/(gamma+1))^(gamma/(gamma-1));

        if p_star > p_atm
            T_e = (2/(gamma+1))*T;
            rho_e = p_star/(R_air*T_e);
            v_e = sqrt(gamma*R_air*T_e);
```

```
            p_e = p_star;

        else
            M = sqrt(((p/p_atm)^((gamma-1)/gamma)-1)*(2/(gamma-1)));
            T_e = T*(1+(gamma-1)/2*M^2)^-1;
            rho_e = p_atm/(R_air*T_e);
            p_e = p_atm;

            v_e = M*sqrt(gamma*R_air*T_e);


        end


        %Calculating thrust
        m_dota = c_dis*rho_e*At*v_e;
        F = m_dota*v_e + (p_e-p_atm)*At;

        %Calculating change in total mass
        mdot_r = -c_dis*rho_e*At*v_e;

        FnetZ = F*sin(theta)-drag*sin(theta)-m_r*g;
        FnetX = F*cos(theta)-drag*cos(theta);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if p>p_atm %Stage 2: Air Expulsion

            ddt1 = x2; %x Velocity
            ddt2 = FnetX/m_r; %Thrust - drag
            ddt3 = z2; %Z velocity
            ddt4 = FnetZ/m_r;  %Thrust-drag-gravity
            ddt5 = mdot_r;
            ddt6 = 0;
            ddt7 = mdot_r;
            %disp('Air Expulsion');
            stage = 2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        elseif p <= p_atm && z1 > 0  %Stage 3: Ballistic Stage

            ddt1 = x2;
            ddt2 = -cos(theta)*drag/m_r;
            ddt3 = z2;
            ddt4 = -sin(theta)*drag/m_r-g;
            ddt5 = 0;
            ddt6 = 0;
            ddt7 = 0;
            %disp('Ballistic');
            stage = 3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        elseif z1<=0   %Stage 4: Landed (no change in any variable)

             ddt1 = 0;
```

```
            ddt2 = 0;
            ddt3 = 0;
            ddt4 = 0;
            ddt5 = 0;
            ddt6 = 0;
            ddt7 = 0;
            %disp('Landing');
            stage = 4;

        end


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

    else  %Error phase just in case
            ddt1 = 0;
            ddt2 = 0;
            ddt3 = 0;
            ddt4 = 0;
            ddt5 = 0;
            ddt6 = 0;
            ddt7 = 0;
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Creating the state matrix with the parts to integrate

```
%This will be outputted by the function and will be put into ode45

    state_matrix = [ddt1;ddt2;ddt3;ddt4;ddt5;ddt6;ddt7];

end
```

Published with MATLAB® R2022a