Table of Contents


```
% Author(s): Andrew Patella
% Assignment title: Coding challenge 6
% Purpose: Model Flight of hot air balloon
% Creation date: 12/04/2023
% Revisions:

clear;clc;close all;
```

Calculations

```
%Const struct, tspan, function handle, initial conditions
const = getConst();

tspan = [0,150];

f= @(t,state)ddt_fun(t,state,const);

init = [const.x0,const.init_velocityX,const.hBoulder,const.init_velocityZ];

%Using ode45 to calculate flight
[t,State] = ode45(f,tspan,init);
```

Quiz Questions

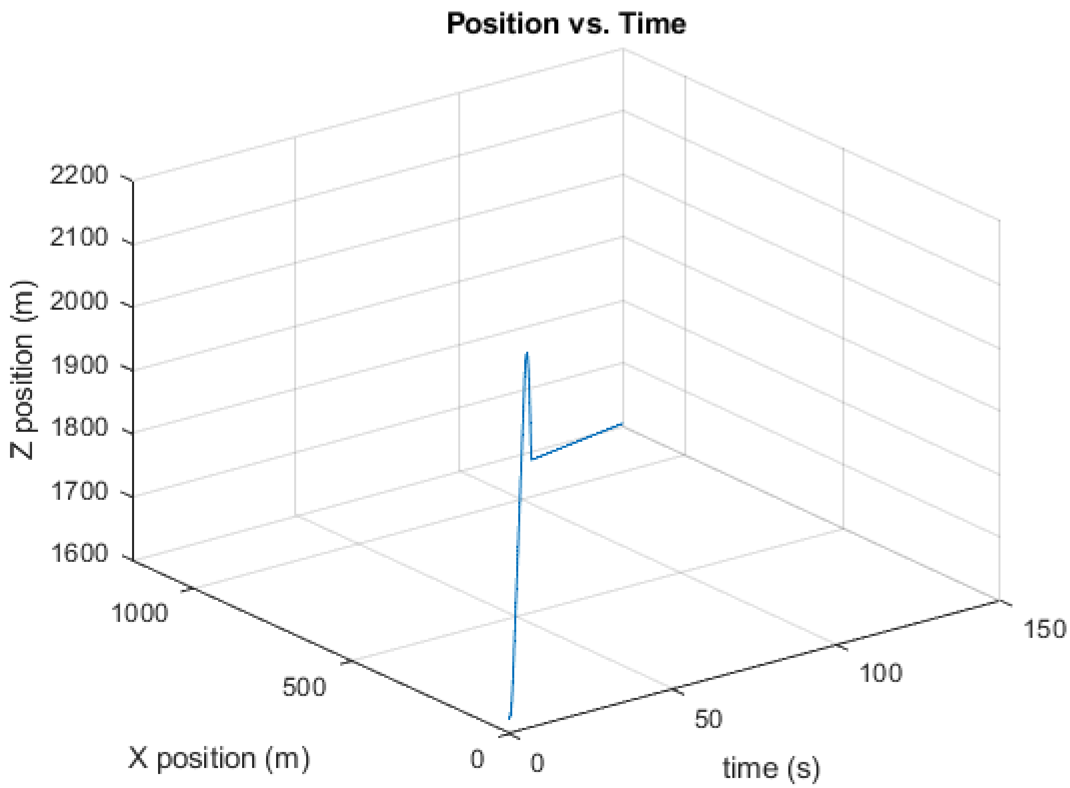
```
%Finding the displacement vector when the balloon lands

%Index value of the first place where z<=0
final_index= find(State(:,3)<=const.hBoulder,2);
final_index = final_index(2);

%Finding the final position and time of landing
final_position = [State(final_index,1),State(final_index,3)];
landing_time = t(final_index);

%Vector of times for interp1
t_quiz = [99.9,33.1,123.45,49.8];

%Calculating the values of z and vz at the t_quiz times
z = interp1(t,State(:,3),t_quiz,"linear","extrap");
```

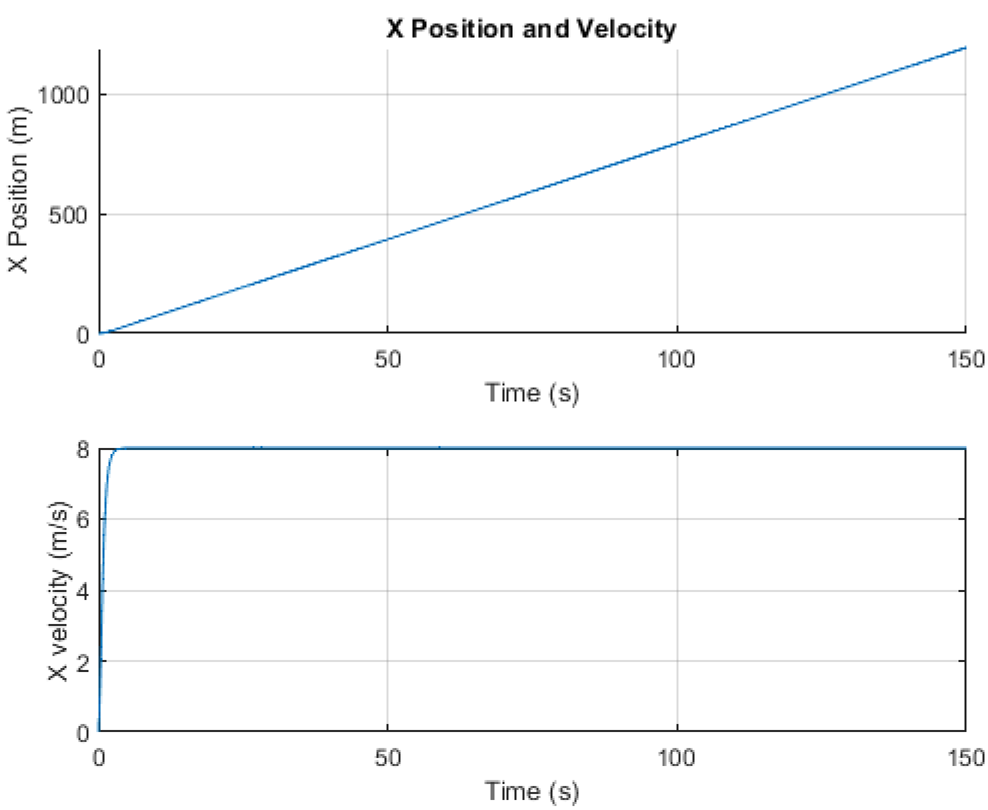
```
vz = interp1(t,State(:,4),t_quiz,"linear","extrap");

%Calculating the values of x and vx at the t_quiz times
x = interp1(t,State(:,1),t_quiz,"linear","extrap");
vx = interp1(t,State(:,2),t_quiz,"linear","extrap");

%Creating vectors of the positions
r1 = [x(1);z(1)];
r2 = [x(2);z(2)];
r3 = [x(3);z(3)];
r4 = [x(4);z(4)];

%Creating vectors of the position, adjusted for the launch altitude
r1_adj = [x(1);z(1)-const.hBoulder];
r2_adj = [x(2);z(2)-const.hBoulder];
r3_adj = [x(3);z(3)-const.hBoulder];
r4_adj = [x(4);z(4)-const.hBoulder];

%Calculating the displacement of the balloon at each quiz time
displacement1 = norm(r1_adj);
displacement2 = norm(r2_adj);
displacement3 = norm(r3_adj);
displacement4 = norm(r4_adj);

%Creating the velocity vector at each quiz time
v1 = [vx(1);vz(1)];
v2 = [vx(2);vz(2)];
v3 = [vx(3),vz(3)];
v4 = [vx(4),vz(4)];

%Calculating the magnitude of the velocity at each quiz time
v_mag1 = norm(v1);
v_mag2 = norm(v2);
v_mag3 = norm(v3);
v_mag4 = norm(v4);

%Printing values to command window
fprintf('Distance traveled at landing: %2.4f m\n',final_position(1));
fprintf('Time of flight: %2.2f seconds \n',t(final_index));
fprintf('Magnitude of displacement and velocity at t1: %2.4f m, %2.4f m/s
\n',displacement1,v_mag1);
fprintf('Magnitude of displacement and velocity at t2: %2.4f m, %2.4f m/s
\n',displacement2,v_mag2);
fprintf('Magnitude of displacement and velocity at t3: %2.4f m, %2.4f m/s
\n',displacement3,v_mag3);
fprintf('Magnitude of displacement and velocity at t4: %2.4f m, %2.4f m/s
\n',displacement4,v_mag4);
```

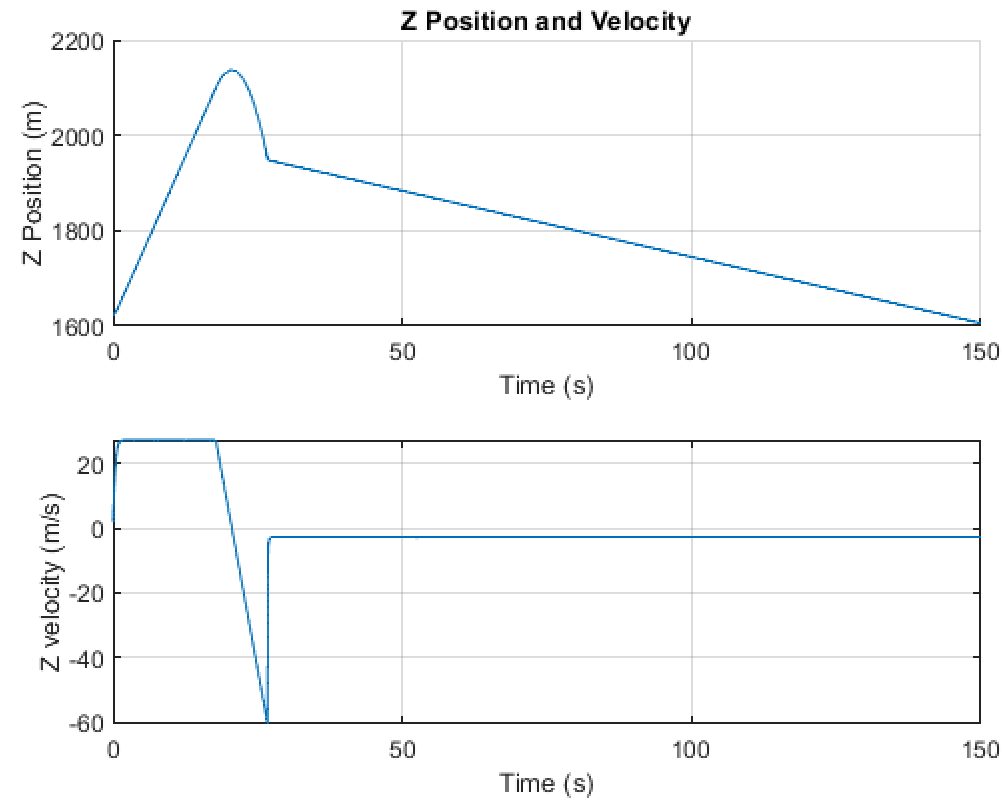
```
Distance traveled at landing: 1141.6818 m
Time of flight: 143.52 seconds
Magnitude of displacement and velocity at t1: 801.8713 m, 8.4680 m/s
Magnitude of displacement and velocity at t2: 400.7770 m, 8.4676 m/s
Magnitude of displacement and velocity at t3: 982.6652 m, 8.4681 m/s
Magnitude of displacement and velocity at t4: 470.3414 m, 8.4685 m/s
```

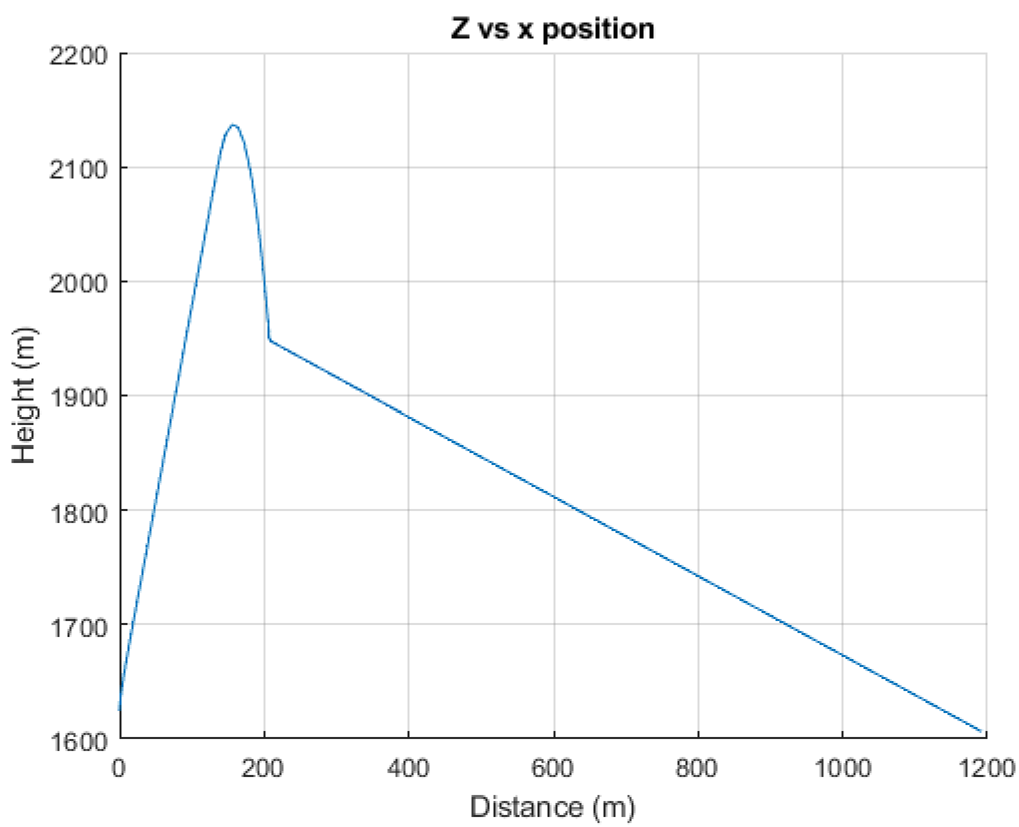
Graphs

```
figure();
plot3(t,State(:,1),State(:,3),'Linewidth',1);
grid on;
hold on;
ylabel('X position (m)');
zlabel('Z position (m)');
xlabel('time (s)');
title('Position vs. Time');
hold off;

figure();
subplot(2,1,1);
hold on;
plot(t,State(:,1),'Linewidth',1);
xlabel('Time (s)');
ylabel('X Position (m)');
title('X Position and Velocity');
grid on;
hold off;

subplot(2,1,2);
plot(t,State(:,2),'Linewidth',1);
grid on;
ylabel('X velocity (m/s)');
xlabel('Time (s)');

figure();
subplot(2,1,1);
hold on;
plot(t,State(:,3),'Linewidth',1);
xlabel('Time (s)');
ylabel('Z Position (m)');
title('Z Position and Velocity');
grid on;
hold off;

subplot(2,1,2);
plot(t,State(:,4),'Linewidth',1);
grid on;
ylabel('Z velocity (m/s)');
xlabel('Time (s)');

figure();
hold on;
plot(State(:,1),State(:,3),'Linewidth',1);
grid on;
title('Z vs x position');
xlabel('Distance (m)');
ylabel('Height (m)');
```


Position vs. Time
Z position (m)
2200
2100
2000
1900
1800
1700
1600
1000
500
0
X position (m)
0
50
100
150
time (s)


X Position and Velocity
X Position (m)
1000
500
0
0
50
100
150
Time (s)
X velocity (m/s)
8
6
4
2
0
0
50
100
150
Time (s)


Z Position and Velocity
Z Position (m)
Time (s)
Z velocity (m/s)
Time (s)


Z vs x position
Height (m)
Distance (m)

Comments

```
% Even though the wind is a constant speed, ode45 still must be used to
% calculate the trajectory in the x direction. As the balloon begins moving
% as a result of the wind, it will gain speed. This effectively slows down
% the wind, from the balloon's perspective. That means that over time, the
% drag force is decreasing, and the force is a function of the balloon's
% velocity. Therefore, this is a second order differential equation, which
% is not linear, since the v term is squared. This has no elementary
% solution and is basically impossible to compute. Therefore, to gain a
% sense of what the balloon is doing in the x direction, one must use ode45
% to model its flight.
```

Functions

```
function const = getConst()
    % This is a void function which can be used in the main code to set a
    % variable to the struct this creates
    const.mass_payload = 450; %kg
    const.r = 17; %m
    const.rhoAir = 1.225; %kg/m^3
    const.rhoHe = 0.1786; %kg/m^3
    const.g = 9.81; %m/s^2
    const.V = (4/3)*pi*(const.r^3); %m^3
    const.A = pi*const.r^2; %m^2
    const.mass_gas = const.V*const.rhoHe; %kg
    const.mass_tot = const.mass_gas+const.mass_payload; %kg
    const.cd1 = 0.5;
    const.cd2 = 0;
    const.cd3 = 1.03;
    const.hpop = 2100; %m
    const.hchute = 1950; %m
    const.hBoulder = 1624; %m
    const.x0 = 0; %m
    const.init_velocityZ = 2; %m/s
    const.init_velocityX = 0; %m/s
    const.v_wind = 8; %m/s

end

function ddt = ddt_fun(t,state,const)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This function calculates the derivatives of the balloon's state for
% ode45 to integrate

%INPUTS:
% t is the time value (no functions are dependant on t so this is unused)
% State is the current state, a column vector of x,vx,z,vz.
% const is a struct of all initial relevant constants.

%OUTPUTS:
```

```
% dzdt is a column vector of the derivatives of state (d/dt[x;vx;z;vz])
% that will be integrated to model the flight of the balloon

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

    %Devectorizing the y vector for simplicity later
    vx = state(2);
    z = state(3);
    vz = state(4);

    %Taking constants and scalars from the const struct for simplicity
    %later
    mtot = const.mass_tot; %kg
    mPay = const.mass_payload; %kg
    rhoA = const.rhoAir; %kg/m^3
    g = const.g; %m/s^2
    C1 = const.cd1;
    C2 = const.cd2;
    C3 = const.cd3;
    A = const.A; %m^2
    V = const.V; %m^3
    pop = const.hpop; %m (altitude of pop)
    chute = const.hchute; %m (altitude of chute deployment)


    %velocities of wind and balloon
    v_wind = [const.v_wind;0];
    v_balloon = [vx;vz];

    %relative velocity of balloon
    v_rel = v_balloon - v_wind;

    %Heading
    h = v_rel/norm(v_rel);

    %If statements to determine which portion of the flight the balloon is
    %in, based on altitude and velocity direction
    if (z<pop && vz>0)
       %Balloon is rising under bouyant force
       D = .5*C1*A*rhoA*norm(v_rel.^2);
       F_drag = -h*D;
       F_buoy = [0;rhoA*V*g];
       F_weight = [0;-mtot*g];
       acceleration = (1/mtot)*(F_drag+F_weight+F_buoy);
       %disp('stage 1');

    elseif (z>pop)
       %Balloon has popped, is still rising because it had momentum
       D = .5*C2*A*rhoA*norm(v_rel.^2);
       F_drag = -h*D;
       F_buoy = [0;0];
       F_weight = [0;-mPay*g];
```

```
        acceleration = (1/mPay)*(F_drag+F_weight+F_buoy);
        %disp('stage 2');

    elseif(z<pop && z>chute && vz<0)
         %Payload is falling but the chute has not opened
        D = .5*C2*A*rhoA*norm(v_rel.^2);
        F_drag = -h*D;
        F_buoy = [0;0];
        F_weight = [0;-mPay*g];
        acceleration = (1/mPay)*(F_drag+F_weight+F_buoy);
        %disp('stage 3');

    else
         %Payload is falling under the chute
        D = .5*C3*A*rhoA*norm(v_rel.^2);
        F_drag = -h*D;
        F_buoy = [0;0];
        F_weight = [0;-mPay*g];
        acceleration = (1/mPay)*(F_drag+F_weight+F_buoy);
        %disp('stage 4');
    end


    %Creating a column vector of the derivatives of overrightarrow{y}
    ddt = [vx;acceleration(1);vz;acceleration(2)];


end
```

Published with MATLAB® R2022a