Table of Contents


```
% Author(s): Andrew Patella
% Assignment title: Coding Challenge 5
% Purpose: Model flight of balloon using ode45
% Creation date: 11/15/2023
% Revisions: N/A

close all; clear; clc;
```

Initial conditions and constants

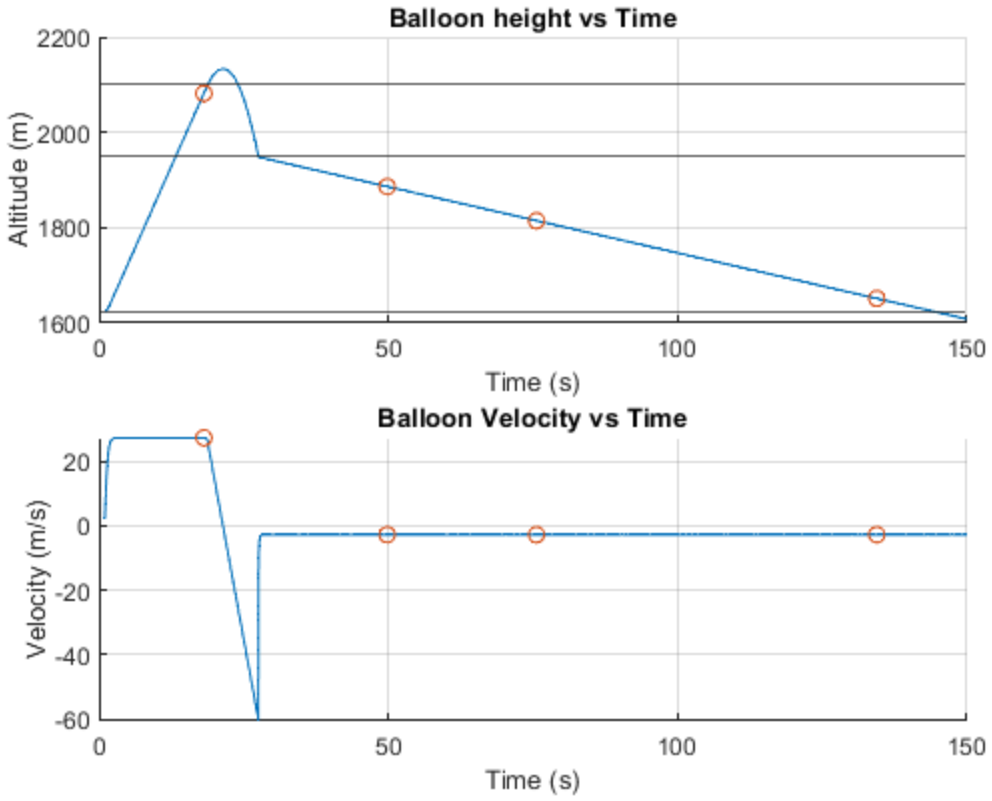
```
%Creating a constant structure using the getConst function
const = getConst();

%Creating t vector for ode45
t = [1 150];

%Creating initial values vector
initial_conditions = [const.hBoulder;const.init_velocity];

%Function handle for dz/dt to put into ode45
f = @(t,y)dzdt_fun(t,y,const);
```

ode45 and quiz question calculations

```
%Calculating the times and positions and velocity of balloon using ode45
[time,z_pos] = ode45(f,t,initial_conditions);

%Calculating the times and velocities for the quiz
%interpolation is necessary for these values since none of the time values
%from the quiz are in the time matrix
t_quiz = [134.4,18.125,75.6,49.8];
z_quiz = interp1(time,z_pos(:,1),t_quiz,"linear","extrap");
v_quiz = interp1(time,z_pos(:,2),t_quiz,"linear","extrap");
t_ground =
 interp1(z_pos([200,1200],1),time([200,1200],1),const.hBoulder,'linear','extrap');
%Originally I calculated this by hand by looking in my matrix for two
%points on either side and creating a line between them using
%y-y1=m(x-x1).Then I plugged in Boulder's altitude and this gave me the
%correct value.
```

Plotting results

```
% Creating a plot of the position and velocity
figure(1);

subplot(2,1,1);
hold on;

plot(time,z_pos(:,1),'Linewidth',1);

grid on;
xlabel('Time (s)');
ylabel('Altitude (m)');
title('Balloon height vs Time');

%Significant altitudes to ensure visually the code is working
yline(const.hBoulder);
yline(const.hpop);
yline(const.hchute);

% for checking the quiz values
% text(t_quiz(1)+1,z_quiz(1)+50,sprintf('%f,%f',t_quiz(1),z_quiz(1)));
% text(t_quiz(2)+1,z_quiz(2)+50,sprintf('%f,%f',t_quiz(2),z_quiz(2)));
% text(t_quiz(3)+1,z_quiz(3)+50,sprintf('%f,%f',t_quiz(3),z_quiz(3)));
% text(t_quiz(4)+1,z_quiz(4)+50,sprintf('%f,%f',t_quiz(4),z_quiz(4)));
scatter(t_quiz,z_quiz)
hold off;

subplot(2,1,2);
hold on;
plot(time,z_pos(:,2),'Linewidth',1);
grid on;
xlabel('Time (s)');
ylabel('Velocity (m/s)');
title('Balloon Velocity vs Time');
scatter(t_quiz,v_quiz);
% text(t_quiz(1)+1,v_quiz(1)+1,sprintf('%f,%f',t_quiz(1),v_quiz(1)));
% text(t_quiz(2)+1,v_quiz(2)+1,sprintf('%f,%f',t_quiz(2),v_quiz(2)));
% text(t_quiz(3)+1,v_quiz(3)+1,sprintf('%f,%f',t_quiz(3),v_quiz(3)));
% text(t_quiz(4)+1,v_quiz(4)+1,sprintf('%f,%f',t_quiz(4),v_quiz(4)));
hold off;
```



Constants

The values I get for velocity and position make sense. The balloon still has a negative velocity of -2.8 m/s, which is a reasonable velocity for the balloon to acheive. The balloon would continue at this velocity until it hits the ground. ode45 does not give a solution to the altitude that is exactly at the initial altitude because it is just iterating and calculating as it thinks is necessary. In order to find when it hits the ground, the data must be interpolated. Besides this, no processing is required after ode45. These values are logical and make physical sense.

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.init_velocity = 2; %m/s

end

function dzdt = dzdt_fun(t,y,const)
% This is a function that returns a column vector that is the state vector
% for ode45 to integrate. It works for a balloon floating up to a certain
% altitude, where it pops, free falls until a certain point, and deploys a
% chute and then falls at terminal velocity until hitting the ground.


% dy/dt = d  [y1;    = [y2]  = [y2      ]
%         dt  y2']   = [y2'] = [F_net/m ]


    %Devectorizing the y vector for simplicity later
    y1 = y(1);
    y2 = y(2);

    %Taking constants and scalars from the const struct for simplicity
    %later
    mtot = const.mass_tot;
    mPay = const.mass_payload;
    rhoA = const.rhoAir;
    g = const.g;
    C1 = const.cd1;
    C2 = const.cd2;
    C3 = const.cd3;
    A = const.A;
    V = const.V;
    pop = const.hpop;
    chute = const.hchute;

    %If statements to determine which portion of the flight the balloon is
    %in, based on altitude and velocity direction
    if (y1<pop && y2>0)
       %Balloon is rising under bouyant force
       y2_dot = (1/mtot)*(rhoA*V*g-mtot*g-.5*C1*A*rhoA*y2^2);

    elseif (y1>pop)
       %Balloon has popped, is still rising because it had momentum
        y2_dot = (1/mPay)*(-mPay*g+.5*C2*A*rhoA*y2^2);

    elseif(y1<pop && y1>chute && y2<0)
        %Payload is falling but the chute has not opened
        y2_dot = (1/mPay)*(-mPay*g+.5*C2*A*rhoA*y2^2);

    else
        %Payload is falling under the chute
        y2_dot = (1/mPay)*(-mPay*g+.5*C3*A*rhoA*y2^2);
```

```
    end

    %Creating a column vector of the derivatives of overrightarrow{y}
    dzdt = [y2;y2_dot];


end
```

Published with MATLAB® R2022a