University of Colorado Boulder

ASEN 3801: Aircraft Dynamics

September 15, 2023

ASEN 3801 LAB 1: Simulate Equations of Motion

Author: 1
Tyler Hoover[1]

Author: 2
Alexander Keller[2]

Author: 3
Rishab Pally[3]

Professor: Eric Frew

[1]SID: 109795203
[2]SID: 109842690
[3]SID: 109519936

Ann and H.J. Smead
Aerospace Engineering Sciences
UNIVERSITY OF COLORADO BOULDER

1 1A

For answer 1A, we simulate nonlinear dynamic system equations for 20 seconds using the ODE45 solver, and we visualize the four provided functions to produce subplots for each variable against time.

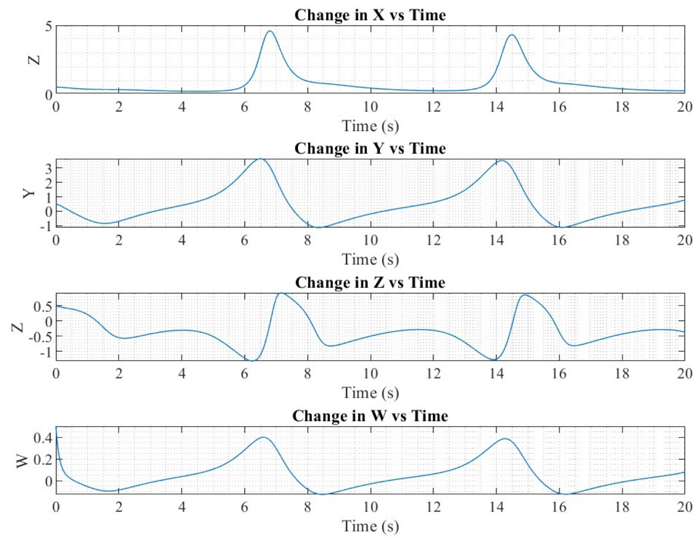

Figure 1: X, Y, Z, and W versus time

2 1B

For this step we simulated new set of equations for 0.2 sec in which everything is the same as above except we replaced the first equation with a new ω equation. We still wanted to represent the solutions to a system of four differential equations for question 1-B.

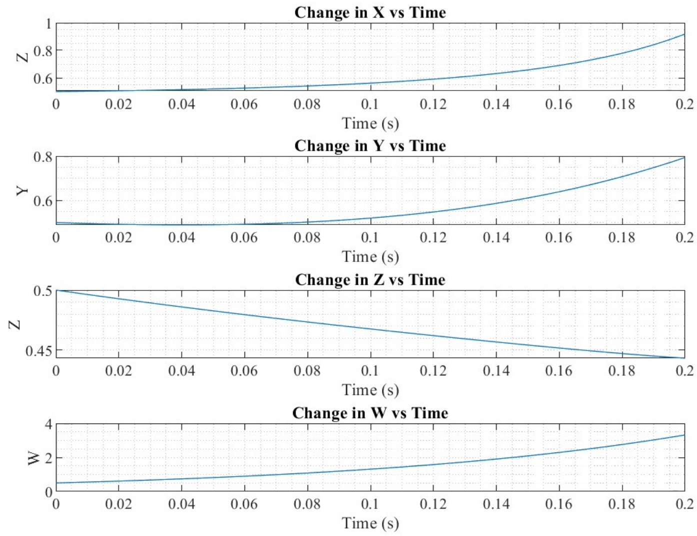

Figure 2: X, Y, Z, and W versus time (.2 seconds)

The biggest difference between the first and second result by changing $\dot{w}$ from -9w to 9w is much smoother curves. When it was negative initially, every graph had consistent peaks that affected each other. This is because with the negative, it means w decays exponentially which affects all other positive w's in the other ODEs. When it switches to positive, all of the w's in each ODE have the same sign meaning no oscillations back to zero can occur.

3 1C

The reason it takes so long to run or fail to complete is because ode45 uses errors to determine the step size needed, and as our function gets larger, it increases exponentially in 3 of the graphs, meaning as it gets larger, the error becomes greater, meaning it will need more time steps to get an accurate answer. Increasing from 0.2 to 2 seconds is a 10x difference, which increases errors and the number of steps required substantially.

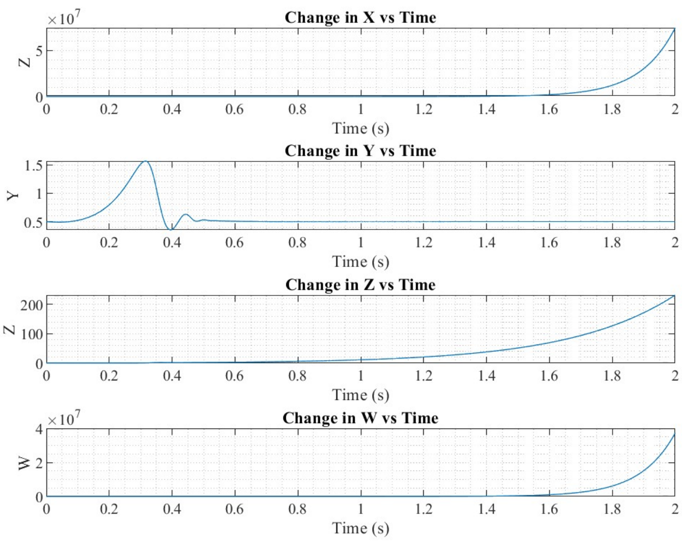

Figure 3: X, Y, Z, and W versus time (2 seconds)

4 2A

In order to answer question 2, we move past simulating the differential equations and start simulating the trajectory of a sphere in the air while being affected by gravity and drag. The ball's travel starts at the origin of the inertial frame, where this problem is taking place. In response to question 2-A, we are to design a function that computes the derivative of a vector that has initial components representing both initial location and initial velocity. This leaves us with a vector that contains the ball's components for acceleration and velocity.

5 2B

An initial location vector of [0,0,0], an initial velocity vector of [0,20,-20], and an initial wind vector of [0,0,0] were provided for our first simulation of the ball in motion. We can see that the ball travels in a direction that is almost parabolic but falls short of the point at which a perfectly parabolic arc would have landed after running the simulation. We can conclude that these findings were reasonable because when drag was taken into account, the ball would lose energy during flight and not move as far as it would on a exact parabolic course, but due to no wind, the path should not deviate far from the relative path.

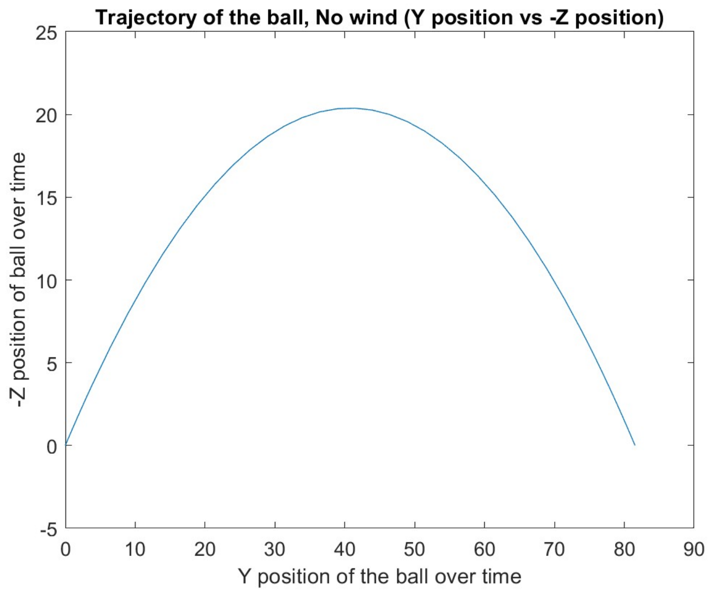

Figure 4: Ball Trajectory Without Wind

6 2C

We factor in the wind into the equations in Question 2-C. We need to compare how far the ball moves when exposed to various levels of wind in the X-direction. Three different wind vectors, each moving in the direction of the ball at various different speeds were used in this simulation. Consequently, the ball moved farther along the X-axis the higher the value of the wind was.

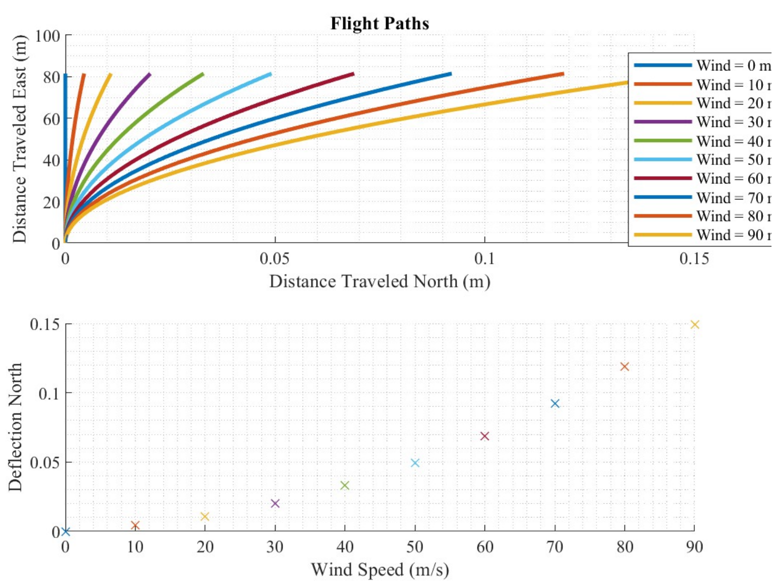

Figure 5: Flight Paths with respect to Wind Speed

7 2D

We can observe from the graph above how the wind vector's speed affects how far the ball travels. This makes sense as if the wind were to move with the ball, that the ball would travel farther the faster the wind was blowing. More wind means more distance, which has an impact on both the ball's horizontal displacement and its overall travel distance/trajectory.

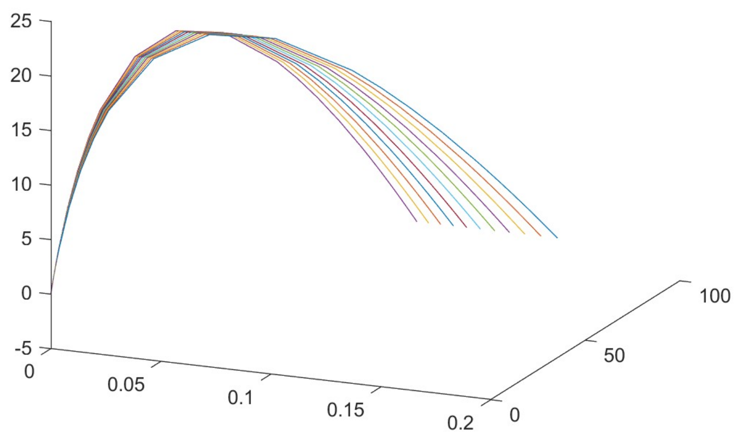

Figure 6: Landing Distance with Varying Altitude

The figure above shows the distance travelled, which is used to determine the landing location distance and a vector showing the altitude growing. As seen in the figure, the lower the altitude, the less distance the ball will be able to travel, thus a smaller minimum landing location. Because the air is less dense at higher altitudes, there is less drag on the ball, which facilitates the ball's movement through the atmosphere.

8 2E

The choice between a lighter or heavier ball is dependent on some aerodynamic factors but a lighter golf ball may have an advantage in achieving a longer distance. This is because a lighter ball can reach a higher velocity with the same amount of available kinetic energy. It is important to note that the choice between a lighter or heavier golf ball should also consider factors like control as well as the wind vector. We can also prove this concept by deriving the kinetic energy in relation with velocity. The new velocity in respect to the new mass can be determined by multiplying the unit vector of the initial velocity by the speed. The heavier ball needs can overcome factors such as drag but also should not be too heavy where the initial velocity is impacted negatively. Therefore we can range values of the mass to determine the most optimal value in respect to both distance and velocity.

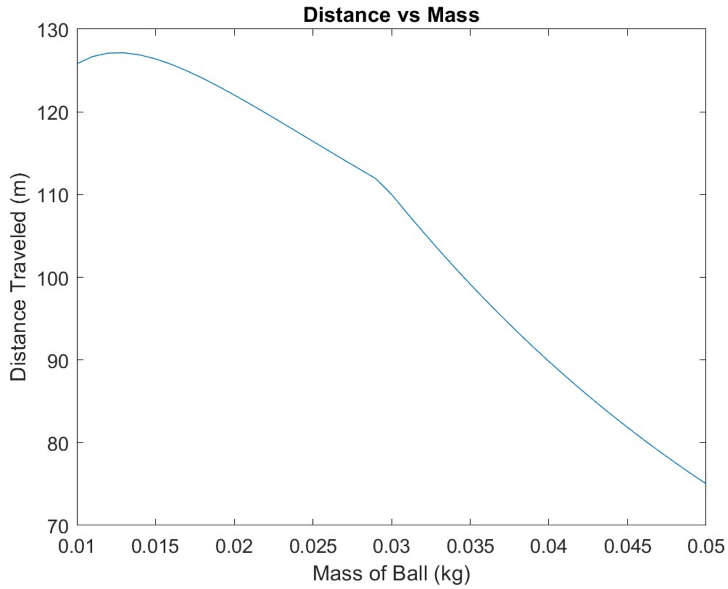

Figure 7: Mass's Impact on Distance

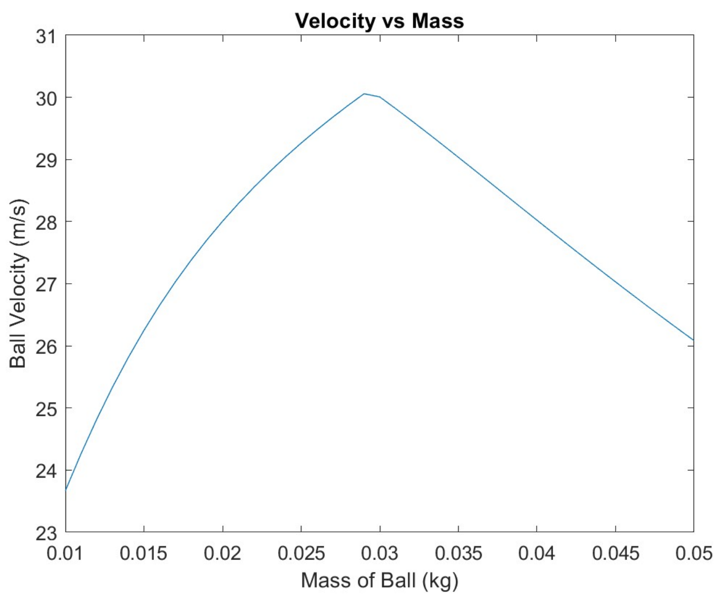

Figure 8: Mass's Impact on Velocity

9 Acknowledgments

9.1 Member Contributions

Deliverable	**Tyler Hoover**	**Alexander Keller**	**Rishab Pally**
Lab Objectives	33%	33%	33%
Matlab Component	33%	33%	33%
Lab Write up	33%	33%	33%
TOTAL	**100%**	**100%**	**100%**

Signature of Approval: Tyler Hoover, Alexander Keller, Rishab Pally

10 Matlab Derivations

```
%ASEN 3801/3128 Lab 1
%Tyler Hoover, Alexander Keller, Rishab Pally
%Created 9/1/23

clc; close;
%% Problem 1

% define initial conditions\
angular_velocity= 0.5;
xpi= 0.5;
y_position_i= 0.5;
z_position_i= 0.5;

initial_state_v= [xpi; y_position_i; z_position_i; angular_velocity];


 %% %%%%%%%%%%%%%%%%%%%%%%%%% Part 1A %%%%%%%%%%%%%%%%%%%%%%%%%
tspan_vals= [0 20];
[t, pos_val1]= ode45(@(t, initial_state_v) EquationsOfmotionFun1A(t, initial_state_v), tspa

%plot figures
figure(1)
subplot(4,1,1)
plot(t,pos_val1(:,1))
title("Change in X vs Time")
xlabel("Time (s)")
ylabel("Z")
set(gca, 'FontName', 'Times New Roman')
grid minor

subplot(4,1,2)
plot(t,pos_val1(:,2))
title("Change in Y vs Time")
xlabel("Time (s)")
ylabel("Y")
set(gca, 'FontName', 'Times New Roman')
grid minor

subplot(4,1,3)
plot(t,pos_val1(:,3))
title("Change in Z vs Time")
xlabel("Time (s)")
ylabel("Z")
set(gca, 'FontName', 'Times New Roman')
grid minor

subplot(4,1,4)
```

```
plot(t,pos_val1(:,4))
title('Change in W vs Time')
xlabel('Time (s)')
ylabel('W')
set(gca, 'FontName', 'Times New Roman')
grid minor
%% %%%%%%%%%%%%%%%%%%%%%%%%%%Part 1B%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulating w a different eom for W and a shorter Time span
tspan_vals2= [0 0.2];
[t, pos_val2]= ode45(@(t, initial_state_v) EquationsOfmotionFun1B(t, initial_state_v), tspa

%plot figures
figure(2)
subplot(4,1,1)
plot(t,pos_val2(:,1))
title("Change in X vs Time")
xlabel("Time (s)")
ylabel("Z")
set(gca, 'FontName', 'Times New Roman')
grid minor

subplot(4,1,2)
plot(t,pos_val2(:,2))
title("Change in Y vs Time")
xlabel("Time (s)")
ylabel("Y")
set(gca, 'FontName', 'Times New Roman')
grid minor

subplot(4,1,3)
plot(t,pos_val2(:,3))
title("Change in Z vs Time")
xlabel("Time (s)")
ylabel("Z")
set(gca, 'FontName', 'Times New Roman')
grid minor

subplot(4,1,4)
plot(t,pos_val2(:,4))
title("Change in W vs Time")
xlabel("Time (s)")
ylabel("W")
set(gca, 'FontName', 'Times New Roman')
grid minor

%% %%%%%%%%%%%%%%%%%%%%%%%%%%Part 1C%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulating for 2 rather than .2 seconds
tspan_vals3= [0 2];
[t, pos_val3]= ode45(@(t, initial_state_v) EquationsOfmotionFun1B(t, initial_state_v), tspa

%plot figures
figure(3)
subplot(4,1,1)
plot(t,pos_val3(:,1))
```

```
title("Change in X vs Time")
xlabel("Time (s)")
ylabel("Z")
set(gca, 'FontName', 'Times New Roman')
grid minor

subplot(4,1,2)
plot(t,pos_val3(:,2))
title("Change in Y vs Time")
xlabel("Time (s)")
ylabel("Y")
set(gca, 'FontName', 'Times New Roman')
grid minor

subplot(4,1,3)
plot(t,pos_val3(:,3))
title("Change in Z vs Time")
xlabel("Time (s)")
ylabel("Z")
set(gca, 'FontName', 'Times New Roman')
grid minor

subplot(4,1,4)
plot(t,pos_val3(:,4))
title("Change in W vs Time")
xlabel("Time (s)")
ylabel("W")
set(gca, 'FontName', 'Times New Roman')
grid minor



%% Problem 2
%initial parameters

mass= 50; %g
diameter= 2.0 * 10.^(-2); %mass
Cd= 0.6;
air_rho= 1.14; %kg/m3 ( air density in boulder co)
A= (pi/4).*diameter.^2; %m^2
g= 9.81; %m/s2


%% Section B
vector_wind = [0;0;0]; %m/s
xpi= [0;0;0;0;20;-20]; % initial velocity component
tspan= [0 5];
[t, X]= ode45(@(t,X) objectEOM(t,X,air_rho,Cd,A,mass,g,vector_wind), tspan, xpi);

%Trajectory of the Ball
figure()
plot(X(:,2), -X(:,3))
title("Trajectory of the ball, No vector_wind (Y position vs -Z position)")
xlabel("Y position of the ball over Time")
```

```
ylabel("-Z position of ball over Time")

% figure()
% plot(X(:,2),X(:,3), LineWidth=2)
% xlabel('Distance Traveled North (mass)')
% % ylabel('Distance Traveled East')
% ylabel('Height (mass)')
% grid minor
% set(gca, 'FontName', 'Times New Roman')
% title('Section B')
%% Section C
% need to characterize Landing Sensitivity to vector_wind

for i= 0:10:90


vector_wind= [i;0;0]; %m/s
xpi= [0;0;0;0;20;-20];
tspan= [0 5];
[t, X_C]= ode45(@(t,X_C) objectEOM(t,X_C,air_rho,Cd,A,mass,g,vector_wind), tspan, xpi);

varStruct((i/10)+1).px1= X_C(:,1);
varStruct((i/10)+1).py1= X_C(:,2);
varStruct((i/10)+1).pz1= X_C(:,3);
varStruct((i/10)+1).VeloX= X_C(:,4);
varStruct((i/10)+1).VeloY= X_C(:,5);
varStruct((i/10)+1).VeloZ= X_C(:,6);
end

figure()
title('Section C')
subplot(2,1,1)
title("Flight Paths")
xlabel("Distance Traveled North (mass)")
ylabel("Distance Traveled East (mass)")
zlabel("Height (mass)")
grid minor
set(gca, 'FontName', 'Times New Roman')
hold on
for i=1:10
plot3(varStruct(i).px1, varStruct(i).py1,varStruct(i).pz1, LineWidth=2)
end
hold off
legend('vector_wind= 0 m/s', 'vector_wind= 10 m/s','vector_wind= 20 m/s','vector_wind= 30 r
    'vector_wind= 50 m/s', 'vector_wind= 60 m/s','vector_wind= 70 m/s','vector_wind= 80 m/s


subplot(2,1,2)
xlabel("vector_wind Speed (m/s)")
ylabel("Deflection North")
grid minor
set(gca, 'FontName', 'Times New Roman')
hold on
for i=1:10
```

```
plot(10*(i-1), varStruct(i).px1(end), 'x')
end

figure()
title("Flight Paths")
xlabel("Distance Traveled North (mass)")
ylabel("Distance Traveled East (mass)")
zlabel("Height (mass)")
grid minor
set(gca, 'FontName', 'Times New Roman')
hold on
for i=1:10
plot3(varStruct(i).px1, varStruct(i).py1,-varStruct(i).pz1, LineWidth=2)

end



%% Section D

alt_range = 0:500:5000;

figure()
for i = 0:10:90
for j = 1:numel(alt_range)

[Txxxx,axxx,Pxxxx,rho] = atmosisa(alt_range(j));

wind = [i;0;0]; %m/s
X0 = [0;0;0;0;20;-20];
tspan = [0 5];
[t, X_C] = ode45(@(t,X_C) objectEOM(t,X_C,rho,Cd,A,m,g,wind), tspan, X0);

varStruct((i/10)+1).PosX = X_C(:,1);
varStruct((i/10)+1).PosY = X_C(:,2);
varStruct((i/10)+1).PosZ = X_C(:,3);
varStruct((i/10)+1).VeloX = X_C(:,4);
varStruct((i/10)+1).VeloY = X_C(:,5);
varStruct((i/10)+1).VeloZ = X_C(:,6);

values = X_C;
plot3(values(:,1),values(:,2),-1*values(:,3))
hold on

end
end

%% Section E
% comassputing diffence in distance due to varying massasses
vector_wind= [0;0;0];

%setting a vector of different massasses of the ball
mass_s= (10:50)./1000; %kg
```

```
%for loop to run through each of the massasses and calculate state variables
%for each
for j= 1:length(mass_s)

%Kinetic energy of the ball at the initial 30 gramasss and 20 m/s east and
%north velocities
calc1= sqrt(20^2 + 20^2)
calcsquare= (calc1).^2
calc2= (mass./1000);
KE= 0.5*calc2*calcsquare;

%Calculating Velocities in the y and z directions for each run through of
%the massass vector
vmcalc= (2*KE);
vmcalc1= mass_s(1,j);
vm_main= vmcalc./vmcalc1;
mass_velocityvec= sqrt(vm_main);
angle1= sind(45);
angle2= cosd(45);
yvity= mass_velocityvec.*angle1;
zvity= mass_velocityvec.*angle2;

%Putting the velocity for the current massass of ball in the initial state
%vector
xpi= [0;0;0;0;yvity;-zvity];

%Calling ODE45
[t, range_x]= ode45(@(t,range_x) objectEOM(t,range_x,air_rho,Cd,A,mass_s(1,j),g,vector_win

%defining each row of the output state vector
x_pstn= range_x(:,1);
yp1= range_x(:,2);
zp1= range_x(:,3);
xv= range_x(:,4);
yv= range_x(:,5);
zv= range_x(:,6);
py_inertial_vec(j)= yp1(end);
vx_inertial_vec(j)= xv(end);
vy_inertial_vec(j)= yv(end);
vz_inertial_vec(j)= zv(end);
end

%Calculating initial ball velocity for each massass of ball
ball_veloc= sqrt((vx_inertial_vec.^2) + (vy_inertial_vec.^2) + (vz_inertial_vec.^2));

%Plot of massass of ball vs distance traveled
figure()
plot(mass_s,py_inertial_vec)
xlabel("Mass of Ball (kg)")
ylabel("Distance Traveled (mass)")
title("Distance vs massass")

%Plot of massass of ball vs initial ball velocity
figure()
```

```
plot(mass_s, ball_veloc)
xlabel("Mass of Ball (kg)")
ylabel("Ball Velocity (m/s)")
title("Velocity vs mass")
```

```
%% Part 1 A Eq function
function [delta_var]= EquationsOfmotionFun1A(t, initial_state_v)
x= initial_state_v(1);
y= initial_state_v(2);
z= initial_state_v(3);
w= initial_state_v(4);

omega= -9 * w + y ;
range_xot= 4 * w * x * y - x^2;
y_dot= 2 * w - x - 2*z ;
z_dot= x * y - y^2 - 3 * z^3 ;

delta_var= [range_xot; y_dot; z_dot; omega];

end

%% Part 1 B Eqmasso function
function [delta_var]= EquationsOfmotionFun1B(t, initial_state_v)
x= initial_state_v(1);
y= initial_state_v(2);
z= initial_state_v(3);
w= initial_state_v(4);

calc_w= 9*w;
omega= calc_w+ y;
calc_range= 4*x*y*w;
calcx_square= x^2;
range_xot= calc_range- calcx_square;
calcw2= 2*w;
calcz2= 2*z;
y_dot= calcw2-x-calcz2;
calcxy= x*y;
calc_ys= y^2;
calc3z= 3* z^3;
z_dot= calcxy-calc_ys- calc3z;
delta_var= [range_xot; y_dot; z_dot; omega];
end


%% Problem 2 EOM
% part A
```

```
function xdot= objectEOM(t,X,air_rho,Cd,A,mass,g,vector_wind)
%{
Inputs: X - State Vector containing x,y,z positions and velocities
= [Px, Py, Pz, x_inertial_vec, y_inertial_vec, z_inertial_vec]';
 air_rho - Air density
 Cd - Coefficient of Drag
 A - Cross Sectional Area
 mass- mass of ball
 g - gravitational constant
 vector_wind - vector_wind vector in xyz= [Wx, Wy, Wz]'
Outputs:
 xdot - Vector containing changes in position and velocity
= [x_inertial_vec, y_inertial_vec, z_inertial_vec, Ax, Ay, Az]'
%}
    if X(3) > 0
     % once Z hits zero, all changes will stop
    xdot= [0; 0; 0; 0; 0; 0];
    else
    % Defining positions and velocities from state vector
    px1= X(1);
    py1= X(2);
    pz1= X(3);
    x_velocity= X(4);
    y_velocity= X(5);
    z_velocity= X(6);
    % calc rel velocity wrt vector_wind
    inertial_v_vec= [x_velocity; y_velocity; z_velocity];
    vr_vw= inertial_v_vec-vector_wind;
    x_inertial_vec= vr_vw(1);
    y_inertial_vec= vr_vw(2);
    z_inertial_vec= vr_vw(3);
    % vector_head unit vector to define drag vector
    vector_head= vr_vw./norm(vr_vw);
    % fdrag
    dragcalc= .5*air_rho*norm(vr_vw).^2
    dragforce= -vector_head*dragcalc*Cd*A;
    grav_force_vec= [0 ; 0; mass.*g];
    % separating vectors into individual
    df1_vec= dragforce(1)+ grav_force_vec(1);
    df2_vec= dragforce(2)+ grav_force_vec(2);
    df3_vec= dragforce(3)+ grav_force_vec(3);
    % calc acceleration components for output vector
    ax= df1_vec./mass;
    ay= df2_vec./mass;
    az= df3_vec./mass;
    xdot= [x_velocity; y_velocity; z_velocity; ax; ay; az];
    end

end
```