0.1 Matlab Code for ODE-45 function call

This the code chunk which integrates the 6DOF system of equations

```
%% code for ode function calls
clc

% profile on
%% intial conditions
tspan = [0 0.6]; % event collision will detect when
   to stop integration, higher values lead to
   smoother curves
E0 = DCM2Euler(7*pi/180,0,0); %intial euler
   parameter matrix
r1 = (Euler2DCM(E0))*[0;0;0.41];
x0 = post2;
%x0 = [r1(1);r1(2);r1(3);0;0;0;40.15;0;0;E0(1);E0(2)
   ;E0(3);E0(4)]; %intial conditions
options = odeset('RelTol', 1e-9, 'AbsTol', 1e-6,'
   Events',@collision); %ode settings
%% ode 45 call
[t,x]=ode89(@(t,x) dof(t,x),tspan,x0,options); %
   function call
%% system of ODES defination
function xdot = dof(t,x)
% function to get system of odes
v0 = x(4:6); %velocity of com
w0 = x(7:9); % angular velocity of frame C2 wrt O
I = PInertiaTens(const.m,const.r0,const.h0); %
   principle inertia tensor matrix
Ixx = I(1,1); %principle moments
Iyy = I(2,2);
Izz = I(3,3);
e = x(10:13); %Euler parameters
R1 = Euler2DCM(e); %Rotation matrix to get to C2
   frame basis from O

f = drag1(v0,w0,const.r0,const.h0,const.Lt,const.Ln,
   const.b1)+R1*[0;0;-9.81*const.m]; %force net on
   body C2 frame basis
```

```
T = torque1(v0,w0,const.r0,const.h0,const.Lt,const.
   Ln); %torque net acting on body C2 frame basis
% 6dof system 13 equation variation
% cheat sheet for state-space
% x1 = x
% x2 = y
% x3 = z
% x4 = u
% x5 = v
% x6 = w
% x7 = wx
% x8 = wy
% x9 = wz
% x10 = e0
% x11 = e1
% x12 = e2
% x13 = e3

xdot =[
    x(4) - x(8)*x(3) + x(9)*x(2); %xdot
    x(5) + x(7)*x(3) - x(9)*x(1); %ydot
    x(6) - x(7)*x(2) + x(8)*x(1); %zdot
    f(1)/const.m - x(8)*x(6)  + x(9)*x(5); %udot
    f(2)/const.m + x(7)*x(6)  - x(9)*x(4); %vdot
    f(3)/const.m - x(7)*x(5)  + x(8)*x(4); %wdot
    T(1)/Ixx + (Iyy - Izz)*x(8)*x(9)/Ixx; %wxdot
    T(2)/Iyy + (Izz - Ixx)*x(7)*x(9)/Iyy; %wydot
    T(3)/Izz + (Ixx - Iyy)*x(7)*x(8)/Izz; %wzdot
    -2*x(11)*x(7)/4 - 2*x(12)*x(8)/4 - 2*x(13)*x(9)
       /4; %e0dot
    2*x(10)*x(7)/4 - 2*x(13)*x(8)/4 + 2*x(12)*x(9)
       /4; %e1dot
    2*x(13)*x(7)/4 + 2*x(10)*x(8)/4 - 2*x(11)*x(9)
       /4; %e2dot
    2*x(11)*x(8)/4 - 2*x(12)*x(7)/4 + 2*x(10)*x(9)
       /4; %e3dot
]
end
%%
```

0.2 Matlab code for Drag

Code chunk used to calculate the drag, this function is used in the body of the ode45 function and is used at every time step

```
function [f] = drag1(v0,w0,r0,h0,Lt,Ln,b)
%drag1 returns the drag force acting on the cylinder
    in terms of the state variables
u = v0(1);
v = v0(2);
w = v0(3);
wx = w0(1);
wy = w0(2);
wz = w0(3);
%% 1 top plane
df1t_x = @(r,p) -Lt.*r.*((w - r.*wy.*cos(p) + r.*wx
   .*sin(p)).^2 + (v - (h0.*wx)./2 + r.*wz.*cos(p))
   .^2 + (u + (h0.*wy)./2 - r.*wz.*sin(p)).^2).^(b
   /2).*(u + (h0.*wy)./2 - r.*wz.*sin(p));
f1t_x = integral2(df1t_x,0,r0,0,2*pi,Method="auto");
df1t_y = @(r,p) -Lt.*r.*((w - r.*wy.*cos(p) + r.*wx
   .*sin(p)).^2 + (v - (h0.*wx)./2 + r.*wz.*cos(p))
   .^2 + (u + (h0.*wy)./2 - r.*wz.*sin(p)).^2).^(b
   ./2).*(v - (h0.*wx)./2 + r.*wz.*cos(p));
f1t_y = integral2(df1t_y,0,r0,0,2*pi,Method="auto");
f1t_z = 0;
f1n_x = 0;
f1n_y = 0;
df1n_z = @(r,p) -Ln.*r.*sign(w - r.*wy.*cos(p) + r.*
   wx.*sin(p)).*(sign(w - r.*wy.*cos(p) + r.*wx.*sin
   (p)) + 1).*((w - r.*wy.*cos(p) + r.*wx.*sin(p))
   .^2 + (v - (h0.*wx)./2 + r.*wz.*cos(p)).^2 + (u +
    (h0.*wy)./2 - r.*wz.*sin(p)).^2).^(b./2).*(w - r
   .*wy.*cos(p) + r.*wx.*sin(p));
f1n_z = integral2(df1n_z,0,r0,0,2*pi,Method="auto");
f1 = [f1t_x + f1n_x;f1t_y + f1n_y;f1t_z+f1n_z]; %
   total force on top plane
%% 2 bottom plane
df2t_x = @(r,p) Lt.*r.*((w - r.*wy.*cos(p) + r.*wx.*
   sin(p)).^2 + (v + (h0.*wx)./2 + r.*wz.*cos(p)).^2
    + ((h0.*wy)./2 - u + r.*wz.*sin(p)).^2).^(b./2)
   .*((h0.*wy)./2 - u + r.*wz.*sin(p));
f2t_x  = integral2(df2t_x,0,r0,0,2*pi,Method="auto")
   ;
df2t_y = @(r,p) -Lt.*r.*(v + (h0.*wx)./2 + r.*wz.*
   cos(p)).*((w - r.*wy.*cos(p) + r.*wx.*sin(p)).^2
   + (v + (h0.*wx)./2 + r.*wz.*cos(p)).^2 + ((h0.*wy
   )./2 - u + r.*wz.*sin(p)).^2).^(b./2);
f2t_y = integral2(df2t_y,0,r0,0,2*pi,Method="auto");
f2t_z = 0;
f2n_x = 0;
f2n_y = 0;
df2n_z = @(r,p) -Ln.*r.*sign(w - r.*wy.*cos(p) + r.*
   wx.*sin(p)).*(sign(w - r.*wy.*cos(p) + r.*wx.*sin
   (p)) - 1).*(w - r.*wy.*cos(p) + r.*wx.*sin(p))
   .*((w - r.*wy.*cos(p) + r.*wx.*sin(p)).^2 + (v +
   (h0.*wx)./2 + r.*wz.*cos(p)).^2 + ((h0.*wy)./2 -
   u + r.*wz.*sin(p)).^2).^(b/2);
f2n_z = integral2(df2n_z,0,r0,0,2*pi,Method="auto");
f2 = [f2t_x + f2n_x;f2t_y + f2n_y;f2t_z+f2n_z]; %
   total force on bottom plane
%% 3 lateral force
df3n_x = @(h,p) -Ln.*r0.*sign(u.*cos(p) + v.*sin(p)
   + h.*wy.*cos(p) - h.*wx.*sin(p)).*cos(p).*(sign(u
   .*cos(p) + v.*sin(p) + h.*wy.*cos(p) - h.*wx.*sin
   (p)) + 1).*((w - r0.*wy.*cos(p) + r0.*wx.*sin(p))
   .^2 + (v - h.*wx + r0.*wz.*cos(p)).^2 + (u + h.*
   wy - r0.*wz.*sin(p)).^2).^(b./2).*(u.*cos(p) + v
   .*sin(p) + h.*wy.*cos(p) - h.*wx.*sin(p));
f3n_x = integral2(df3n_x,-h0/2,h0/2,0,2*pi,Method="
   auto");
df3n_y = @(h,p) -Ln.*r0.*sign(u.*cos(p) + v.*sin(p)
   + h.*wy.*cos(p) - h.*wx.*sin(p)).*sin(p).*(sign(u
   .*cos(p) + v.*sin(p) + h.*wy.*cos(p) - h.*wx.*sin
   (p)) + 1).*((w - r0.*wy.*cos(p) + r0.*wx.*sin(p))
   .^2 + (v - h.*wx + r0.*wz.*cos(p)).^2 + (u + h.*
   wy - r0.*wz.*sin(p)).^2).^(b./2).*(u.*cos(p) + v
   .*sin(p) + h.*wy.*cos(p) - h.*wx.*sin(p));
f3n_y = integral2(df3n_y,-h0/2,h0/2,0,2*pi,Method="
   auto");
f3n_z = 0;
```

```
df3t_x = @(h,p) Lt.*r0.*((w - r0.*wy.*cos(p) + r0.*
   wx.*sin(p)).^2 + (v - h.*wx + r0.*wz.*cos(p)).^2
   + (u + h.*wy - r0.*wz.*sin(p)).^2).^(b/2).*((u.*
   cos(2.*p))./2 - u./2 + (v.*sin(2.*p))./2 - (h.*wy
   )./2 + r0.*wz.*sin(p) + (h.*wy.*cos(2.*p))./2 - (
   h.*wx.*sin(2.*p))./2);
f3t_x = integral2(df3t_x,-h0/2,h0/2,0,2*pi,Method="
   auto");
df3t_y = @ (h,p) Lt.*r0.*cos(p).*((w - r0.*wy.*cos(p
   ) + r0.*wx.*sin(p)).^2 + (v - h.*wx + r0.*wz.*cos
   (p)).^2 + (u + h.*wy - r0.*wz.*sin(p)).^2).^(b
   ./2).*(u.*sin(p) - v.*cos(p) - r0.*wz + h.*wx.*
   cos(p) + h.*wy.*sin(p));
f3t_y = integral2(df3t_y,-h0/2,h0/2,0,2*pi,Method="
   auto",AbsTol=1e-09);
df3t_z = @ (h,p) Lt.*r0.*((w - r0.*wy.*cos(p) + r0.*
   wx.*sin(p)).^2 + (v - h.*wx + r0.*wz.*cos(p)).^2
   + (u + h.*wy - r0.*wz.*sin(p)).^2).^(b./2).*((u.*
   cos(2.*p))./2 - u./2 + (v.*sin(2.*p))./2 - (h.*wy
   )./2 + r0.*wz.*sin(p) + (h.*wy.*cos(2.*p))./2 - (
   h.*wx.*sin(2.*p))./2);
f3t_z = integral2(df3t_z,-h0/2,h0/2,0,2*pi,Method="
   auto");
f3 = [f3t_x + f3n_x;f3t_y + f3n_y;f3t_z+f3n_z]; %
   total force on the lateral surface (needed only
   for 3d coin)
%%
f= f1+f2+f3; %force net return value
end
```

0.3 Matlab code for Torque

Code chunk to calculate the total torque acting on the coin,like drag,this function is also used in every time step of the ode45 routine

```
function [t] = torque1(v0,w0,r0,h0,Lt,Ln)
%torque1 returns the torque acting on the cylinder
   as a function of state
%variables
u = v0(1);
```

```
v = v0(2);
w = v0(3);
wx = w0(1);
wy = w0(2);
wz = w0(3);
%% 1 %all integrands are defined as function handles
    in vars r p and h
dT1_x = @(p,r) (Lt.*h0.*r.*(v - (h0.*wx)./2 + r.*wz
   .*cos(p)))./2 - Ln.*r.^2.*sign(w - r.*wy.*cos(p)
   + r.*wx.*sin(p)).*sin(p).*(sign(w - r.*wy.*cos(p)
    + r.*wx.*sin(p)) + 1).*(w - r.*wy.*cos(p) + r.*
   wx.*sin(p));
dT1_y = @(p,r) Ln.*r.^2.*sign(w - r.*wy.*cos(p) + r
   .*wx.*sin(p)).*cos(p).*(sign(w - r.*wy.*cos(p) +
   r.*wx.*sin(p)) + 1).*(w - r.*wy.*cos(p) + r.*wx.*
   sin(p)) - (Lt.*h0.*r.*(u + (h0.*wy)./2 - r.*wz.*
   sin(p)))./2;
dT1_z = @(p,r) Lt.*r.^2.*sin(p).*(u + (h0.*wy)./2 -
   r.*wz.*sin(p)) - Lt.*r.^2.*cos(p).*(v - (h0.*wx)
   ./2 + r.*wz.*cos(p));
%% 2
dT2_x = @(p,r) -(Lt.*h0.*r.*(v + (h0.*wx)./2 + r.*wz
   .*cos(p)))./2 - Ln.*r.^2.*sign(w - r.*wy.*cos(p)
   + r.*wx.*sin(p)).*sin(p).*(sign(w - r.*wy.*cos(p)
    + r.*wx.*sin(p)) - 1).*(w - r.*wy.*cos(p) + r.*
   wx.*sin(p));
dT2_y = @(p,r) Ln.*r.^2.*sign(w - r.*wy.*cos(p) + r
   .*wx.*sin(p)).*cos(p).*(sign(w - r.*wy.*cos(p) +
   r.*wx.*sin(p)) - 1).*(w - r.*wy.*cos(p) + r.*wx.*
   sin(p)) - (Lt.*h0.*r.*((h0.*wy)./2 - u + r.*wz.*
   sin(p)))./2;
dT2_z = @(p,r) -Lt.*r.^2.*cos(p).*(v + (h0.*wx)./2 +
    r.*wz.*cos(p)) - Lt.*r.^2.*sin(p).*((h0.*wy)./2
   - u + r.*wz.*sin(p));
%% 3
dT3_x = @(h,p) -h.*(Lt.*r0.*cos(p).*(u.*sin(p) - v.*
   cos(p) - r0.*wz + h.*wx.*cos(p) + h.*wy.*sin(p))
   - Ln.*r0.*sign(u.*cos(p) + v.*sin(p) + h.*wy.*cos
   (p) - h.*wx.*sin(p)).*sin(p).*(sign(u.*cos(p) + v
   .*sin(p) + h.*wy.*cos(p) - h.*wx.*sin(p)) + 1).*(
   u.*cos(p) + v.*sin(p) + h.*wy.*cos(p) - h.*wx.*
   sin(p))) - Lt.*r0.^2.*sin(p).*(w - r0.*wy.*cos(p)
    + r0.*wx.*sin(p)); % x component of dtorque
dT3_y = @(h,p) h.*(Lt.*r0.*((u.*cos(2.*p))./2 - u./2
    + (v.*sin(2.*p))./2 - (h.*wy)./2 + r0.*wz.*sin(p
   ) + (h.*wy.*cos(2.*p))./2 - (h.*wx.*sin(2.*p))
   ./2) - Ln.*r0.*sign(u.*cos(p) + v.*sin(p) + h.*wy
   .*cos(p) - h.*wx.*sin(p)).*cos(p).*(sign(u.*cos(p
   ) + v.*sin(p) + h.*wy.*cos(p) - h.*wx.*sin(p)) +
   1).*(u.*cos(p) + v.*sin(p) + h.*wy.*cos(p) - h.*
   wx.*sin(p))) + Lt.*r0.^2.*cos(p).*(w - r0.*wy.*
   cos(p) + r0.*wx.*sin(p));
dT3_z = @(h,p) Lt.*r0.^2.*(u.*sin(p) - v.*cos(p) -
   r0.*wz + h.*wx.*cos(p) + h.*wy.*sin(p));
%% integration to get total torques
T1_x = integral2(dT1_x,0,2*pi,0,r0,Method="auto",
   AbsTol=1e-05);
T1_y = integral2(dT1_y,0,2*pi,0,r0,Method="auto",
   AbsTol=1e-05);
T1_z = integral2(dT1_z,0,2*pi,0,r0,Method="auto",
   AbsTol=1e-05);
T1 = [T1_x;T1_y;T1_z];
%%
T2_x = integral2(dT2_x,0,2*pi,0,r0,Method="auto",
   AbsTol=1e-05);
T2_y = integral2(dT2_y,0,2*pi,0,r0,Method="auto",
   AbsTol=1e-06);
T2_z = integral2(dT2_z,0,2*pi,0,r0,Method="auto",
   AbsTol=1e-05);
T2 = [T2_x;T2_y;T2_z];
%%
T3_x = integral2(dT3_x,-h0/2,h0/2,0,2*pi,Method="
   auto",AbsTol=1e-05);
T3_y = integral2(dT3_y,-h0/2,h0/2,0,2*pi,Method="
   auto",AbsTol=1e-05);
T3_z = integral2(dT3_z,-h0/2,h0/2,0,2*pi,Method="
   auto",AbsTol=1e-05);
T3 = [T3_x;T3_y;T3_z];
%% return value
t = T1+T2+T3;
```

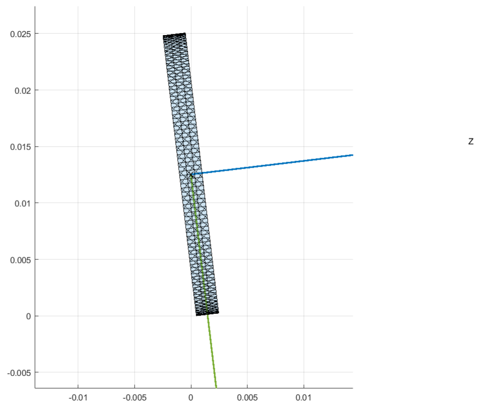
```
end
```

0.4 Matlab code for collision detection

Event function to let MATLAB know when to stop integrating

```
function [check,isterminal,direction] = collision(t,
   x)
%event function to detect impact with the floor and
   stop the ode solver,
%is called each time stpe and executes when the
   check value is zero
%
e = x(10:13); % Euler parameters at that time step
R1 = Euler2DCM(e); % rotation matrix to get to C2
   frame from O frame
OcC2 = transpose(R1); %rotation matrix to get to O
   frame from C2
phi = acos(R1(3,3)); % angle between the k axes of
   the two frames
posC2 = x(1:3); % position vector of com in C2 frame
posO = OcC2*posC2;  % position vector of com in O
   frame

check = posO(3) - (const.r0*abs(sin(phi))+const.h0
   /2*abs(cos(phi))); % position of coin center
   accounting for radius of coin
isterminal = 1; % Do I want to stop the code when
   event happens, 1 for yes 0 for no
direction = -1; % do I care from which direction the
    check hits 0, value of 0 means I dont care
end
```

Image showing the instance at which the solver stops and the position of the coin at that instance



0.5 Matlab code for finding out point of impact on coin

Code chunk to solve for the position of the point which hits the floor, this information in needed for solving the impact equations

```
function [phi] = impact_point(t,x)
%returns the angle phi which denotes the position of
    impact on the
%coin,which is needed to compute the post impact
   state space
syms phi0
xi = transpose(x(length(t),1:end));
e = [xi(10);xi(11);xi(12);xi(13)];
R = Euler2DCM(e);
cphi = R(3,3);
r_A_C2_C2 = [const.r0*cos(phi0);const.r0*sin(phi0);-
   sign(cphi)*const.h0/2];
r_C2_O_C2 = [xi(1);xi(2);xi(3)];
r_A_O_C2  = r_A_C2_C2+r_C2_O_C2;
r_A_O_O = transpose(Euler2DCM(e))*r_A_O_C2;
phi = double(solve(r_A_O_O(3),phi0)); % equates the
   z co-ord to zero in inertial space
end
```

0.6 Matlab code for solving impact equations

Code chunk which solves the pre-impact-post-impact equations, I am sure I am making a major mistake here.(But i have not been able to figure what it is).

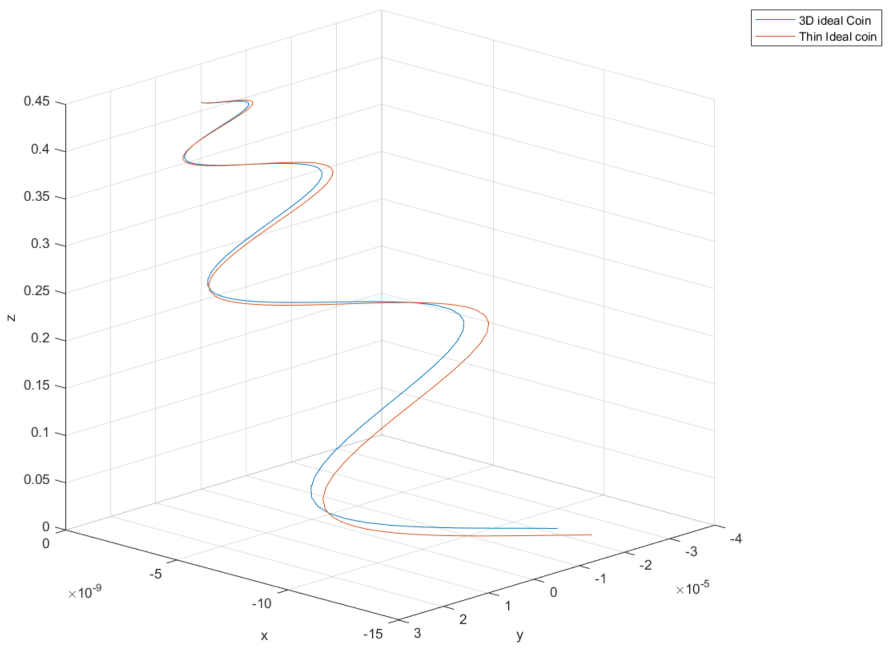
```
function[x1] = impact_solver(t,x)
%function to solve newtons hypothesis alongside
   momentum conservation and
%constraint equations to return post impact state-
   space vector to feed into
%ode 89 again
xi = transpose(x(length(t),1:end)); % pre impact
   state-space in C2 basis
C2cO = Euler2DCM(xi(10:13)); %rotation matrix to get
    to C2 space from O space
OcC2 = transpose(C2cO); % rotation matrix to get to
   O space from C2 space
psi = C2cO(3,3); % angle between Ko and Kc2 axes
phi = impact_point(t,x); %angle to define the polar
   co-ords of the point of impact
w = OcC2*xi(7:9); %angular vel of C2 wrt to O in O
   frame basis pre impact
v = OcC2*xi(4:6); % vel of COM wrt to O in C2 space
   pre impact
rc2 = OcC2*[const.r0*cos(phi(1));const.r0*sin(phi(1)
   );-sign(psi)*const.h0/2];
r = OcC2*[const.r0*cos(phi(1));const.r0*sin(phi(1))
   ;-sign(psi)*const.h0/2]; %position vector of A
   wrt to C2 in C2 frame basis
va = OcC2*(vec2cross(xi(7:9))*rc2 + xi(4:6)); %vel
   of Point A wrt to O in O frame basis pre impact
B = [0;0;-va(3)*const.chi]; % vel of point A wrt to
   O in C2 frame basis post impact (restitution)
I = OcC2*PInertiaTens(const.m,const.r0,const.h0)*
   C2cO; %inertia tensor of coin

%% defining the coeffs of the matrix A for the Ax =
   B sys of equations
a11 =1;
a18 = r(3);
```

```
a19 = -r(2);
b1 = B(1);

a22 = 1;
a29 = r(1);
a27 = -r(3);
b2 = B(2);

a33  = 1;
a37 = r(2);
a38 = -r(1);
b3 = B(3);

a44  = -1;
a55 = -1;
a66 = -1;
a41 = const.m;
a52 = const.m;
a63 = const.m;
b4 = const.m*v(1);
b5 = const.m*v(2);
b6 = const.m*v(3);

a77 = I(1,1);
a76 = -r(2);
a75 = r(3);

a84 = -r(3);
a86 = r(1);
a88 = I(2,2);

a94 = r(2);
a95 = -r(1);
a99 = I(3,3);

b7 = I(1,1)*w(1);
b8 = I(2,2)*w(2);
b9 = I(3,3)*w(3);
```

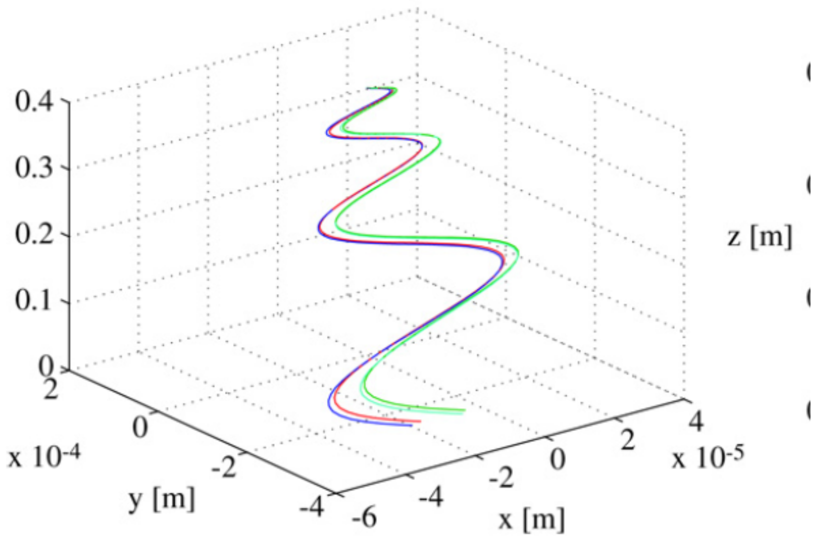
```
A = [a11 0 0 0 0 0 0 a18 a19;
     0 a22 0 0 0 0 a27 0 a29;
     0 0 a33 0 0 0 a37 a38 0;
     a41 0 0 a44 0 0 0 0 0;
     0 a52 0 0 a55 0 0 0 0;
     0 0 a63 0 0 a66 0 0 0;
     0 0 0 0 a75 a76 a77 0 0;
     0 0 0 a84 0 a86 0 a88 0;
     0 0 0 a94 a95 0 0 0 a99]; %main part, use
        symbolic toolbox in a different script to
        generate the coeffs of the matrix
b = transpose([b1 b2 b3 b4 b5 b6 b7 b8 b9]);%main
   part, use sym toolbox in a diff script to get
   values
xo = A\b; % moneyshot
velc2 = C2cO*[xo(1:3)];
wc2 = C2cO*[xo(7:9)];
x1 = real([xi(1);xi(2);xi(3);velc2(1);velc2(2);velc2
   (3);wc2(1);wc2(2);wc2(3);xi(10);xi(11);xi(12);xi
   (13)]);
%x1 = OcC2*[xo(1);xo(2);xo(3)]
%x1 = B
end

%%
```

1 Plots of free-fall and bouncing back

My plot for the freefall trajectory of the coin COM for two cases below

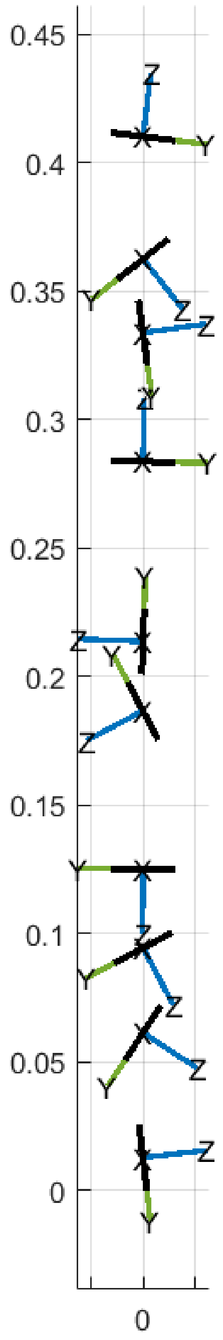


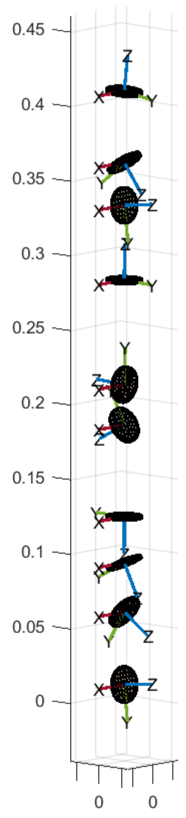
Plots in paper for the same two cases below



Note:(The shapes of the plots match almost perfectly but there seems to be a mislabelling of the axes in the paper)

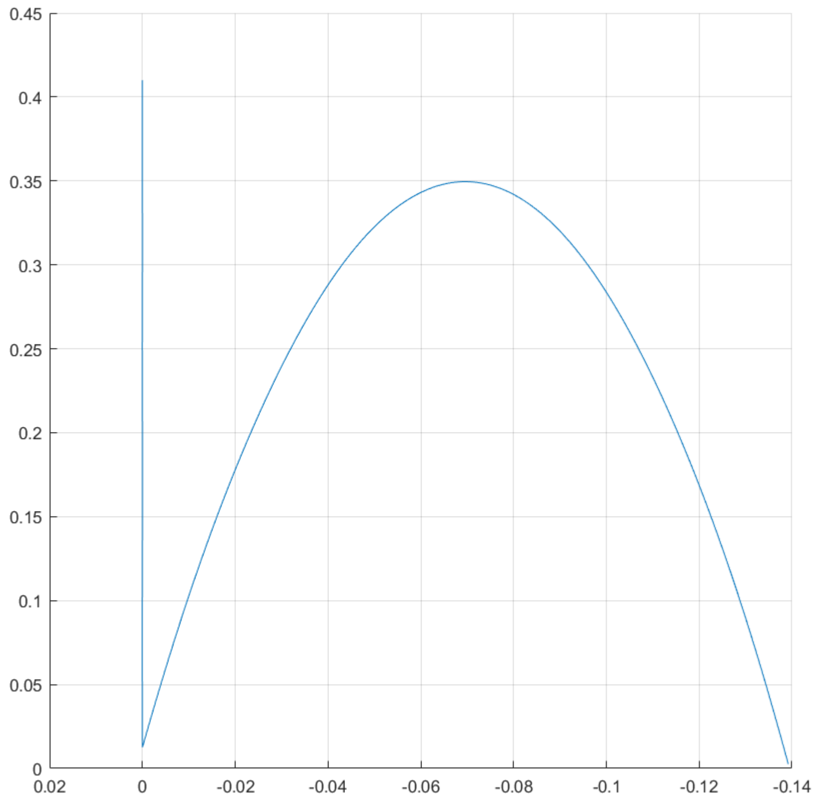
2 Visualization of the coin motion





2.1 Plots of Bouncing Back

Note: (I attempted to simulate the impact,but i could only get coherent results for a single bounce, any more bounces were throwing a lot of errors and I have not been able to resolve them.) Below is the image of one sucessful bounce i did get, It does not match the one in the paper however.


0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0.02
0
-0.02
-0.04
-0.06
-0.08
-0.1
-0.12
-0.14