```
% APPEDNIX A: The Trajectory Program

function
jordanrkbounceflight(e,u,pres,r,p,A,m,xi,yi,zi,vi,wup,wright,phi,psi,dt,filename)

% save filename as .csv and put in single quotes ' '
% pull onto desktop to open with exel.

% potive y is to the left...

% e is the coefficient of restitution
% u is the coefficient of sliding friction

% math

    % ma* = Fd* + Fl* + Fs* + mg*    (* denotes vectors)

    % a = Bv^2[-Cdv* + Cll* + Cs(l* x v*)] + g*

      % B = pA/2m (p = air density, A = cross sectional area, m = mass)

      % (theta = angle of the flight height of the ball, z axis and x-y plane)
      % (phi = angle between z-y axis, off of z so inverted, add 90 -)
      % (psi = angle between x-y axis on the ground)

    % v* = sin(phi)cos(psi)x* + sin(phi)sin(psi)y* + cos(phi)z*

    % l* = -cos(phi)cos(psi)x* - cos(phi)sin(psi)y* + sin(phi)z*

    % (l* x v*) = -sin(phi)x* + cos(psi)y*

% r = radius of the ball

% sp = rw/v
    % r = radius of the ball
    % v = velocity of ball
    % w is angular velocity
        % wup is topspin: backspin is +
        % wright is side: spin CCW is +, + Z

 % inital conditions

i = 1;

g = -9.80665;

B = ((p*A)/(2*m));

% tb = time duration of the bounce
```

```
    % c = cirumference of the ball, pres is the air pressure in the ball
    % (Wesson)

c = 2*r*(pi);

tb = (pi)*sqrt(m/(c*pres))

angphi = (90-phi)*(pi)/180;
angpsi = (psi)*(pi)/180;

vx(i) = vi*sin(angphi)*cos(angpsi);
vy(i) = vi*sin(angphi)*sin(angpsi);
vz(i) = vi*cos(angphi);

x(i) = xi;
y(i) = yi;
z(i) = zi;
t(i) = 0;
v(i) = vi;
wup(i) = wup;
wright(i) = wright;

w(i) = sqrt(wup(i)^2 + wright(i)^2);

sp(i) = (r*w(i)/v(i));

absp(i) = sqrt(sp^2);

% Cd

if (0.05 < absp(i)) && (absp(i) < 0.30);

    Cd(i) = 0.56*(absp(i)) + 0.147;

elseif (v(i) < 11.00);

    Cd(i) = 0.460;

elseif (v(i) > 29.5);

    Cd(i) = 0.190;

else

    Cd(i) = 6.629 - 1.204*v(i) + 0.08250*(v(i)^2) - 0.002480*(v(i)^3) + .
00002765*(v(i)^4);

end
```

```
if (vx(i) < 0)
    Cd(i) = -Cd(i);
else
    Cd(i) = Cd(i);
end
% cut off at high sp, 0.3.....
% Cl
sp1(i) = (r*wup(i)/v(i));
absp1(i) = sqrt(sp1(i)^2);
if (absp1(i) < 0.25)
    Cl(i) = 1.14*(sp1(i));
elseif (wup(i) < 0)
    Cl(i) = -0.3;
else
    Cl(i) = 0.3;
end
% Cs
sp2(i) = (r*wright(i)/v(i));
absp2(i) = sqrt(sp2(i)^2);
if (absp2(i) < 0.25)
    Cs(i) = 1.14*(sp2(i));
elseif (wright(i) < 0)
    Cs(i) = -0.3;
else
    Cs(i) = 0.3;
```

```
end

% for the bounce
    % stop when velocity is less than one meter per second

while (i < 2500)


%for the ball in flight

% spin decay

wup(i+1) = wup(i)*exp(((-0.002*v(i))/r)*t(i));
wright(i+1) = wright(i);


% runge-kutta functions
    % x

    xdxa(i) = vx(i)*dt;
    xdva(i) = B*(v(i)^2)*[-Cd(i)*(sin(angphi)*cos(angpsi)) + Cl(i)*
(-cos(angphi)*cos(angpsi)) + Cs(i)*(-sin(angphi))]*dt;

    xdxb(i) = (vx(i)+(0.5*xdva(i)))*dt;
    xdvb(i) = B*((v(i)+(0.5*xdva(i)))^2)*[-Cd(i)*(sin(angphi)*cos(angpsi)) + Cl(i)*
(-cos(angphi)*cos(angpsi)) + Cs(i)*(-sin(angphi))]*dt;

    xdxc(i) = (vx(i)+(0.5*xdvb(i)))*dt;
    xdvc(i) = B*((v(i)+(0.5*xdvb(i)))^2)*[-Cd(i)*(sin(angphi)*cos(angpsi)) + Cl(i)*
(-cos(angphi)*cos(angpsi)) + Cs(i)*(-sin(angphi))]*dt;

    xdxd(i) = (vx(i)+(xdvc(i)*dt))*dt;
    xdvd(i) = B*((v(i)+(xdvc(i)))^2)*[-Cd(i)*(sin(angphi)*cos(angpsi)) + Cl(i)*
(-cos(angphi)*cos(angpsi)) + Cs(i)*(-sin(angphi))]*dt;


    x(i+1) = x(i) + (xdxa(i)/6) + (xdxb(i)/3) + (xdxc(i)/3) + (xdxd(i)/6);
    vx(i+1) = vx(i) + (xdva(i)/6) + (xdvb(i)/3) + (xdvc(i)/3) + (xdvd(i)/6);

    % y

    ydxa(i) = vy(i)*dt;
    ydva(i) = B*(v(i)^2)*[-Cd(i)*(sin(angphi)*sin(angpsi)) + Cl(i)*
(-cos(angphi)*sin(angpsi)) + Cs(i)*(cos(angpsi))]*dt;

    ydxb(i) = (vy(i)+(0.5*ydva(i)))*dt;
    ydvb(i) = B*((v(i)+(0.5*ydva(i)))^2)*[-Cd(i)*(sin(angphi)*sin(angpsi)) + Cl(i)*
(-cos(angphi)*sin(angpsi)) + Cs(i)*(cos(angpsi))]*dt;
```

```
    ydxc(i) = (vy(i)+(0.5*ydvb(i)))*dt;
    ydvc(i) = B*((v(i)+(0.5*ydvb(i)))^2)*[-Cd(i)*(sin(angphi)*sin(angpsi)) + Cl(i)*
(-cos(angphi)*sin(angpsi)) + Cs(i)*(cos(angpsi))]*dt;

    ydxd(i) = (vy(i)+(ydvc(i)*dt))*dt;
    ydvd(i) = B*((v(i)+(ydvc(i)))^2)*[-Cd(i)*(sin(angphi)*sin(angpsi)) + Cl(i)*
(-cos(angphi)*sin(angpsi)) + Cs(i)*(cos(angpsi))]*dt;


    y(i+1) = y(i) + (ydxa(i)/6) + (ydxb(i)/3) + (ydxc(i)/3) + (ydxd(i)/6);
    vy(i+1) = vy(i) + (ydva(i)/6) + (ydvb(i)/3) + (ydvc(i)/3) + (ydvd(i)/6);

    % z

    zdxa(i) = vz(i)*dt;
    zdva(i) = (B*(v(i)^2)*[-Cd(i)*(cos(angphi)) + Cl(i)*(sin(angphi)) + Cs(i)*0] +
g)*dt;

    zdxb(i) = (vz(i)+(0.5*zdva(i)))*dt;
    zdvb(i) = (B*((v(i)+(0.5*zdva(i)))^2)*[-Cd(i)*(cos(angphi)) + Cl(i)*(sin(angphi)) +
Cs(i)*0] + g)*dt;

    zdxc(i) = (vz(i)+(0.5*zdvb(i)))*dt;
    zdvc(i) = (B*((v(i)+(0.5*zdvb(i)))^2)*[-Cd(i)*(cos(angphi)) + Cl(i)*(sin(angphi)) +
Cs(i)*0] + g)*dt;

    zdxd(i) = (vz(i)+(zdvc(i)*dt))*dt;
    zdvd(i) = (B*((v(i)+(zdvc(i)))^2)*[-Cd(i)*(cos(angphi)) + Cl(i)*(sin(angphi)) +
Cs(i)*0] + g)*dt;


    z(i+1) = z(i) + (zdxa(i)/6) + (zdxb(i)/3) + (zdxc(i)/3) + (zdxd(i)/6);
    vz(i+1) = vz(i) + (zdva(i)/6) + (zdvb(i)/3) + (zdvc(i)/3) + (zdvd(i)/6);

% new velocity

v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2 + vz(i+1)^2);

% new sp

w(i+1) = sqrt(wup(i+1)^2 + wright(i+1)^2);

sp(i+1) = (r*w(i+1)/v(i+1));

absp(i+1) = sqrt(sp(i+1)^2);

% determine if ball bounces (or part is incase it bounces but the change in
% velocity doesnt cause the trajectory to pass z = 0 for some reason.
```

```
if (z(i+1) >= 0) || (z(i) < 0) && (z(i+1) < 0)

t(i+1) = t(i) + dt;

else

    t(i+1) = t(i) + tb;

    % bounce is rolling
        % vz is negative in if statement bc it is traveling towards earth
        % here

        vz(i+1) = (e * (-vz(i)));
        vx(i+1) = (vx(i) + (u*g*tb));
        vy(i+1) = (vy(i) + (u*g*tb));
        wup(i+1) = wup(i) + ((1/(0.5*(r^2)))*((vx(i)*u*g*tb) + (vy(i)*u*g*tb) + vz(i)*(1
- e)));
        wright(i+1) = 0.6*wright(i+1);

% change in spin
% all backspin turns to topspin (observation)

% will be determined via observation

% new velocity

v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2 + vz(i+1)^2);

% new sp

w(i+1) = sqrt(wup(i+1)^2 + wright(i+1)^2);

sp(i+1) = (r*w(i+1)/v(i+1));

absp(i+1) = sqrt(sp(i+1)^2);

end

% C coefficients

if (0.05 < absp(i+1)) && (absp(i+1) < 0.30);

    Cd(i+1) = 0.56*(absp(i+1)) + 0.147;

elseif (v(i+1) < 11.00);

    Cd(i+1) = 0.460;
```

```
elseif (v(i+1) > 29.5);

    Cd(i+1) = 0.190;

else

    Cd(i+1) = 6.629 - 1.204*v(i+1) + 0.08250*(v(i+1)^2) - 0.002480*(v(i+1)^3) + .
00002765*(v(i+1)^4);

end

if (vx(i+1) < 0)

    Cd(i+1) = -Cd(i+1);

else

    Cd(i+1) = Cd(i+1);

end

% cut off at high sp, 0.3.....
% Cl
sp1(i+1) = (r*wup(i+1)/v(i+1));

absp1(i+1) = sqrt(sp1(i+1)^2);

if (absp1(i+1) < 0.25)

    Cl(i+1) = 1.14*(sp1(i+1));

elseif (wup(i+1) < 0)

    Cl(i+1) = -0.3;

else

    Cl(i+1) = 0.3;

end

% Cs

sp2(i+1) = (r*wright(i+1)/v(i+1));

absp2(i+1) = sqrt(sp2(i+1)^2);

if (absp2(i+1) < 0.25)
```

```
    Cs(i+1) = 1.14*(sp2(i+1));

elseif (wright(i+1) < 0)

    Cs(i+1) = -0.3;

else

    Cs(i+1) = 0.3;

end

% add to the count

i = i + 1;

end


% make position values the same index as other values.

%output

    % Make a matrix of all the wanted arrays

M = [t; x; y; z; v; vx; vy; vz; wup; wright; Cs; Cl; Cd];

A = M';

% a backup just in case

save datark.out A -ASCII

csvwrite(filename, A);

%graphics

datacursormode on

    % 2D

plot(t,x,t,y,t,z);

grid on

subplot(1,2,1);

    % 3D
```

```
plot3(x,y,z);

grid on

subplot(1,2,2);

end
```

Appendix B

A look at a bounce using Newton's second law and torques. Assume that the only forces acting on the ball during the bounce are gravity and friction. Friction causes a force on the ball r distance away from the center leading to rotation. This torque only adds topspin to the ball, Δ wup, therefore in this analysis Δ $wright$ is not analyzed.

$$\sum \tau_{up} = I\alpha_{up} \tag{1}$$

where α is angular acceleration and I is the moment of inertia.

$$I = C_r m r^2 \tag{2}$$

where C_r is the moment coefficient, use shell/sphere value, m is the ball's mass, and r is the ball's radius.

Solving equation 1 for α_{up} and plugging in $\tau_{up} = F_b r$ gives the analytical angular differential equation. Integrating twice gives:

$$\omega up = \omega up_0 + \frac{F_b}{C_r m r} t_b \tag{3}$$

where $F_b = \mu m g$, and t_b is the duration of the bounce.

Appendix C

Looking at the bounce in terms of energy. There is translational and rotational energy for the ball in flight, both changing upon bounce.

$$E_{trans} = \frac{1}{2}mv^2 + mgh \tag{1}$$

$$E_{rot} = \frac{1}{2}I\omega^2 \tag{2}$$

$$E_{tot} = \frac{1}{2}I\omega^2 + \frac{1}{2}mv^2 + mgh \tag{3}$$

where I is the moment of inertia, ω is angular velocity, v is linear velocity, m is mass, g is the gravitational constant, and h is the height of the ball off of the ground. During the bounce h is zero, so the energy after the bench is:

$$\frac{1}{2}mv_0^2 + \frac{1}{2}I\omega_0^2 = \frac{1}{2}mv_1^2 + \frac{1}{2}I\omega_1^2 \tag{4}$$

breaking v and ω up into cartesian coordinates yields

$$\frac{1}{2}m({v_x}_0^2 + {v_y}_0^2 + {v_z}_0^2) + \frac{1}{2}I(\omega_{up0}^2 + \omega_{right0}^2) = \frac{1}{2}m({v_x}_1^2 + {v_y}_1^2 + {v_z}_1^2) + \frac{1}{2}I(\omega_{up1}^2 + \omega_{right1}^2) \tag{5}$$

Assuming there is no side-spin, $\omega_{right0} = \omega_{right1} = 0$, using the following definition: $v_{i1} \equiv e_i v_{i0}$ where i is x, y, or z (e_z is typically called the coefficient of restitution), and solving for post-bounce rotational energy gives:

$$\frac{1}{2}mv_{x0}^2(1-e_x) + \frac{1}{2}mv_{y0}^2(1-e_y) + \frac{1}{2}mv_{z0}^2(1-e_z) + \frac{1}{2}I\omega_{up0}^2 = \frac{1}{2}I\omega_{up1}^2 \tag{6}$$

Using $I = C_r m r^2$ and solving for w_{up1} gives

$$\omega_{up1} = \sqrt{\omega_{up0}^2 + \frac{1}{C_r r^2}[v_{x0}^2(1-e_x) + v_{y0}^2(1-e_y) + v_{z0}^2(1-e_z)]} \tag{7}$$

Solving Newton's second law for translational velocity using only frictional and gravitational forces gives $v_{i1} = -\mu g t_b + v_{i0}$, where i is x or y, $v_{z1} = e_z v_{z0}$. Using $e_i \equiv \frac{v_{x1}}{v_{x0}}$ and plugging in the post bounce cartesian velocities gives:

$$e_i = 1 - \frac{\mu g t_b}{v_{i0}} \tag{8}$$

Plugging this into the post-bounce topspin equation and knowing e_z is e, the coefficient of restitution, gives

$$\omega up = \sqrt{{\omega up_0}^2 + \frac{1}{C_r r^2}[v_{x0}\mu g t_b + v_{y0}\mu g t_b + v_{z0}^2(1-e)]} \tag{9}$$

```
% APPENDIX D: The Tracking Program

% n is name, x, y are scales of axis

function m = picture2(n)

% filename in single quotes and ends in .csv

a = n;

% convert picture to a 'truecolor RGB 3-D array/matrix

b = imread(a);

% gather info on the picture

imfinfo(n)

% display image

    % RGB image : Red, Green, Blue

z = image(b);

grid on
datacursormode on
subplot(2,2,1)

    % HSV image : Hue, Saturation, value components

q = rgb2hsv(b);

image(q);

grid on
datacursormode on
subplot (2,2,2)

% b = picture array
% (height, width, which dimension 1 = R, 2 = G, 3 = B)
    % height and width a:b gives table of pixel height a and width b
   % to pick out single pixel a,b...

    % get pixel height and width of picture via imfinfo

i = 1;
j = 1;
k = 1;
kk = 1;
```

```
% h = height in pixels,
% w = width in pixels
% numfram = number of frames

h = 240;     % 96 for 1200 fps, ... 240 for 60 fps
w = 320;    % 336 for 1200 fps, ... 320 for 60 fps

R1 = b(1:h,1:w,1);
G1 = b(1:h,1:w,2);
B1 = b(1:h,1:w,3);

H1 = q(1:h,1:w,1);
S1 = q(1:h,1:w,2);
V1 = q(1:h,1:w,3);

Y(k) = 0;
X(k) = 0;
YY(kk) = 0;
XX(kk) = 0;

while (j <= w);

i = 1;

while (i <= h);

R(i,j) = b(i,j,1);
G(i,j) = b(i,j,2);
B(i,j) = b(i,j,3);

H(i,j) = q(i,j,1);
S(i,j) = q(i,j,2);
V(i,j) = q(i,j,3);

if ((R(i,j)/(B(i,j)) > 2) && (G(i,j)/(B(i,j)) > 1.5) && (G(i,j)/(R(i,j)) < 1.5)) ||
((R(i,j) > 230) && (G(i,j) > 230) && (B(i,j) < 200))

    A(i,j,1) = 255;

    Y(k) = i;
    X(k) = j;

    k = k + 1;

else

    A(i,j,1) = 0;
```

```
end

if (H(i,j) < 0.14)

    AA(i,j,1) = 255;

    YY(kk) = i;
    XX(kk) = j;

    kk = kk + 1;

else

    AA(i,j,1) = 0;

end

i = i + 1;

end

j = j + 1;

end

Y1 = median(Y)
X1 = median(X)

YY1 = median(YY)
XX1 = median(XX)

image(A)
datacursormode on
grid on
subplot(2,2,3)

image(AA)
datacursormode on
grid on
subplot(2,2,4)


%x and y are two-dimensional vectors; x is x axis scale [x0 xf] , y is y axis scale [y0
yf], n is the name and location of the
% file

csvwrite('AA.csv', AA);
csvwrite('A.csv', A);
csvwrite('RGB.csv',b);
```

```
csvwrite('HSV.csv', q);
csvwrite('Rpict.csv', R1);
csvwrite('Gpict.csv', G1);
csvwrite('Bpict.csv', B1);
csvwrite('Hpict.csv', H1);
csvwrite('Spict.csv', S1);
csvwrite('Vpict.csv', V1);

end
```