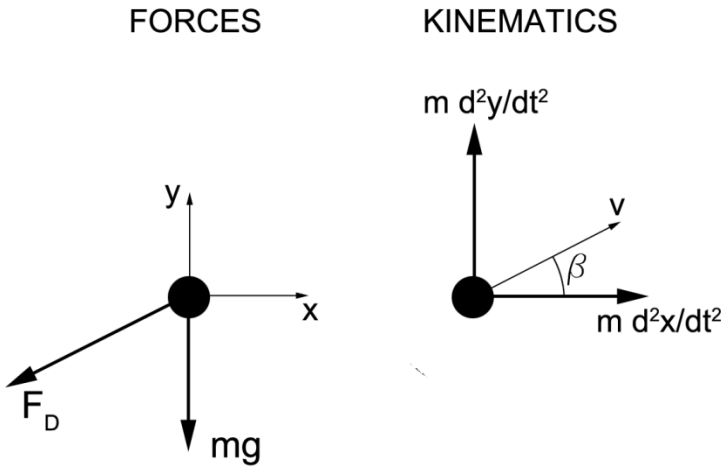
EM375 MECHANICAL ENGINEERING EXPERIMENTATION
PROJECTILE MOTION WITH AIR RESISTANCE

This handout presents the theory of a projectile with air resistance, and how to solve for the motion using a Runge-Kutta numerical solution using MATLAB.

For projectile motion where air resistance cannot be ignored, there are two forces of importance: the projectile's weight *mg* which is constant and is always directed down, and the drag due to air resistance F_D which is directed in the opposite direction to the instantaneous velocity. The magnitude of the drag force changes with speed and direction of the velocity vector.

The instantaneous angle of flight β is the same as the instantaneous direction of the velocity vector. This angle is defined as positive when it is above the horizontal and negative when it is below. As the projectile flies through the air, β will start at the initial launch angle β_0. It will reduce to zero at the apogee, and then become negative as the projectile falls down.

The force and kinematics diagrams are:



Recalling that $\dot{x}$ (read as x-dot) is shorthand for $\frac{dx}{dt}$ and $\ddot{x}$ (x-double-dot) is shorthand for $\frac{d^2x}{dt^2}$ the two equations of motion governing the flight of the projectile are:

x-direction:

$$-F_D cos(\beta) = m\ddot{x}$$

y-direction:

$$-mg - F_D sin(\beta) = m\ddot{y}$$

The magnitude of the drag force depends upon the speed (magnitude of the velocity vector) of the projectile and several parameters:

$$F_D = C_D A_f \rho_{air} \frac{v^2}{2}$$

A_f is a reference area related to the body. For this project the area is the silhouette area of the ping pong ball.

ρ_{air} is the density of air (mass per unit volume).

C_D is a dimensionless drag coefficient. The drag coefficient depends on a number of factors that will be addressed in your fluid dynamics course. For a ping pong ball travelling at the Reynold's number achieved in this project a drag coefficient of 0.47 ± 0.03 at 95% confidence is appropriate.

The two nonlinear 2nd order coupled differential equations of motion for the x- and y-directions can be solved numerically in MATLAB using the Runge-Kutta "ODE45" function. The function only permits a single integration, and not the double integration that is needed for these second order equations. Therefore the two 2nd order equations must be converted into a system of four first order equations. Once this is done the "ODE45" routine can be used to solve for the *x* and *y* positions and the *x* and *y* velocities as functions of time. The equations of motion are developed below and the appropriate MATLAB code shown. You will need to modify this code to match the requirements of your specific project, especially the initial and boundary conditions.

The instantaneous velocity $\bar{v}$ can be presented as its horizontal and vertical components $\dot{x}$ and $\dot{y}$. Using these components, we can establish:

$$v^2 = (\dot{x}^2 + \dot{y}^2)$$

$$\dot{x} = v\, cos(\beta)$$

$$\dot{y} = v\, sin(\beta)$$

We now combine and rearrange these equations so that we get the instantaneous trigonometric functions as functions of the velocity components $\dot{x}$ and $\dot{y}$:

$$cos(\beta) = \frac{\dot{x}}{v} = \frac{\dot{x}}{\sqrt{\dot{x}^2 + \dot{y}^2}}$$

$$sin(\beta) = \frac{\dot{y}}{v} = \frac{\dot{y}}{\sqrt{\dot{x}^2 + \dot{y}^2}}$$

Rearranging the second order equations of motion and substituting the above trigonometric formulae gives:

For the x-direction:

$$\ddot{x} = -\frac{F_D cos(\beta)}{m}$$

$$\ddot{x} = -\frac{C_D A_f \rho_{air}}{2m} v^2 cos(\beta) = -\frac{C_D A_f \rho_{air}}{2m} (\dot{x}^2 + \dot{y}^2) \frac{\dot{x}}{\sqrt{\dot{x}^2 + \dot{y}^2}}$$

$$\ddot{x} = -\frac{C_D A_f \rho_{air}}{2m} \dot{x} \sqrt{\dot{x}^2 + \dot{y}^2}$$

for the y-direction:

$$\ddot{y} = \frac{-mg - F_D sin(\beta)}{m}$$

$$\ddot{y} = -g - \frac{C_D A_f \rho_{air}}{2m} v^2 sin(\beta) = -g - \frac{C_D A_f \rho_{air}}{2m} (\dot{x}^2 + \dot{y}^2) \frac{\dot{y}}{\sqrt{\dot{x}^2 + \dot{y}^2}}$$

$$\ddot{y} = -g - \frac{C_D A_f \rho_{air}}{2m} \dot{y} \sqrt{\dot{x}^2 + \dot{y}^2}$$

In preparation for entering these equations into MATLAB, the above equations can now be reduced to a system of 1st order differential equations by making the following substitutions. The substitutions are chosen in this particular order because when we use MATLAB's ODE45 (which uses a Runge-Kutta solution method to solve the simultaneous equations using the code presented later in this handout) it expects[1] input arrays that are ordered as:

(*variable*$_1$), (differential of *variable*$_1$), (*variable*$_2$), (differential of *variable*$_2$).

And so, for this project we will establish the order as;

(x), ($\dot{x}$), (y), ($\dot{y}$).

On this course when you studied nonlinear regression, you recall setting up a MATLAB variable that could be passed as the first argument into a user-defined function. This variable was a vector of parameter numbers. Similarly, the ODE45 functions requires us to set up a vector of parameters. The order of the parameters has to be the same order as the input.

[1] As with all things "Matlab", there are several different ways of accomplishing the same thing. The method presented here is a) self-consistent; and b) works for this project!

We choose to use variable p for the parameters, and have the established order of variables:

$$p(1) = p_1 = x$$
$$p(2) = p_2 = \dot{x}$$
$$p(3) = p_3 = y$$
$$p(4) = p_4 = \dot{y}$$

We now need equations that can calculate the differentials of those parameters.

For the x-direction:

$$\dot{p}_1 = \dot{x} = p_2$$

$$\dot{p}_2 = \ddot{x} = \frac{-C_D A_f \rho_{air}}{2m} p_2 \sqrt{p_2^2 + p_4^2}$$

For the y-direction:

$$\dot{p}_3 = \dot{y} = p_4$$

$$\dot{p}_4 = \ddot{y} = -g - \frac{C_D A_f \rho_{air}}{2m} p_4 \sqrt{p_2^2 + p_4^2}$$

For this exposition the origin of the coordinate system is taken at the unstretched point of the "sling," just as the ball starts to come out of the pouch. You will need to change these for your launcher/project. The initial conditions at *t* = 0 are:

$$\text{at } t = 0: x_0 = 0; y_0 = 0;\ \dot{x}_0 = V_0 cos(\beta_0);\ \dot{y}_0 = V_0 sin(\beta_0)$$

where V_0 is the initial launch speed and β_0 is the launch angle. You will need to set a different origin for your project; perhaps pick ground level as an easily identifiable location on the launcher. This will also change the *x* and *y* initial conditions x_0 and y_0.

Now onto the MATLAB code. We need two separate script files: a function to calculate the differentials, and a main script file that calls the Runge-Kutta routine and plots the final results.

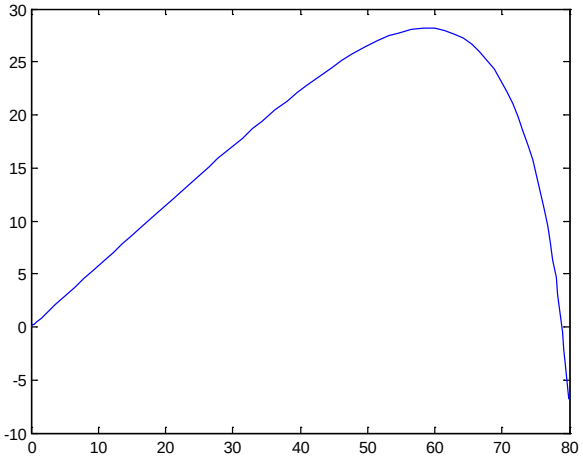
The function shown below, function secondode.m, includes the above equations for the four 1st order system of equations. The function receives the current time (variable *t*) and the values of the current *x* position, *x* velocity, *y* position, and *y* velocity (as an array in variable *indata*). The function returns the values of the four derivatives calculated using the previous equations.

```
function [ p ] = secondode( t, indata )
%% simultaneous second order differentials for projectile
% motion with air resistance
% output vector p has the four differential outputs
% assumed units: meters, seconds, Newtons, kg, radians
global C  g % these are defined globally so they can be changed
% outside the function - means this function doesn't need editing
% for different coordinate systems

p = zeros(4,1); % initialize output space
p(1) = indata(2); % p(1) is the differential of x, which is x_dot,
which is indata(2)
p(2) = -C*sqrt(indata(2)^2 + indata(4)^2)* indata(2); % from the
function
p(3) = indata(4); % same idea as p(1), but for y
p(4) = -g -C*sqrt(indata(2)^2 + indata(4)^2)* indata(4); % from the
function
end
```

The script file Projectile.m calls on MATLAB's ode45 function which uses a Runge-Kutta algorithm to calculate the results. The function ode45 calls on user function secondode.m described above.

```
%% Projectile.m
% written by Colin Ratcliffe
% assumed units: meters, seconds, Newtons, kg, radians
clear;clc
global C  g
g=9.81; % m/s^2
V0=300; % m/s initial launch speed
beta0=30*pi/180; % initial launch angle in degrees converted to radians
m=6.58/1000; % mass of projectile, kg
d=0.0355; % diameter of spherical projectile, meters
Cd=0.5; % assumed
rho=1.2041; % density of air, kg/m^3 Is this a "good" number?

A=pi*d^2/4; % silhouette area, m^2
C=Cd*A*rho/2/m; % the drag force constant

%% perform projectile calcs
tmax=5; % do calculations for 5 seconds of flight time – you will need
to change this to set different upper limits on hang time
tspan = [0 tmax];
% initial conditions as [x0, vx0, y0, vy0]
IC = [0; V0*cos(beta0); 0; V0*sin(beta0)];
[t, oput] = ode45(@secondode, tspan, IC); % Runge-Kutta to solve

x= oput(:,1); % extract x-position from 1st column
vx= oput(:,2); % extract x-velocity from 2nd column
y= oput(:,3); % extract y-position from 3rd column
vy= oput(:,4); % extract y-velocity from 4th column

figure(1);clf;
plot(x,y); % plot to see the projectile's path
```

For the above example (with 5 seconds of flight time) note that the projectile ends up at a lower height than the launch position (it has gone underground). Also note that this graph is awful. No units, captions, legends, axis line at y=-0, etc.



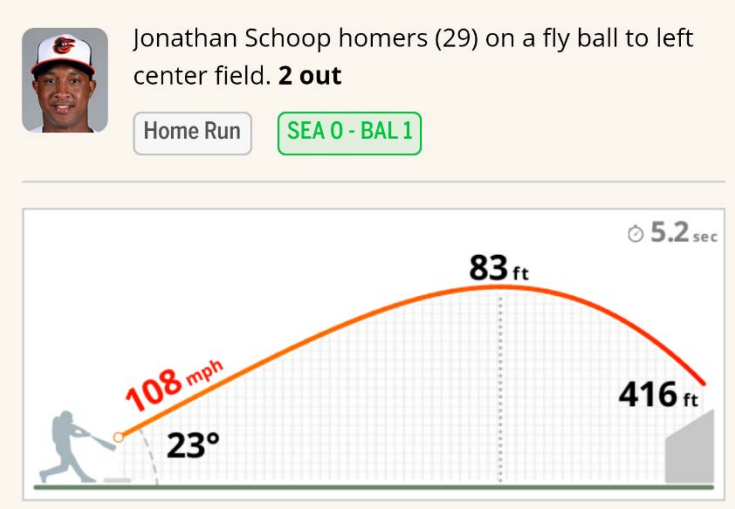
You will need to modify the code to include your values of mass, diameter, etc. Also change initial conditions to match your origin. It is recommended that you set the height origin at floor level. That way the range of the projectile can be identified as where it goes "subsurface" and the calculated y-position transitions from positive to negative.

You should also code MATLAB to print out the achieved range; use *fprintf*. You may consider some linear interpolation in order to get an accurate value.

PROJECTILE TEST DATA

As with all good engineering simulations, before you use your MATLAB script code for the cannon lab, you should check the output against known test data. This is a quick way of checking that your code does not have any gross errors.

On August 29, 2017, Jonathan Schoop (Orioles 2nd baseman) hit a home run against the Seattle Mariners. The statcast is shown below.



1) Use classical mechanics to calculate the range assuming no air resistance.

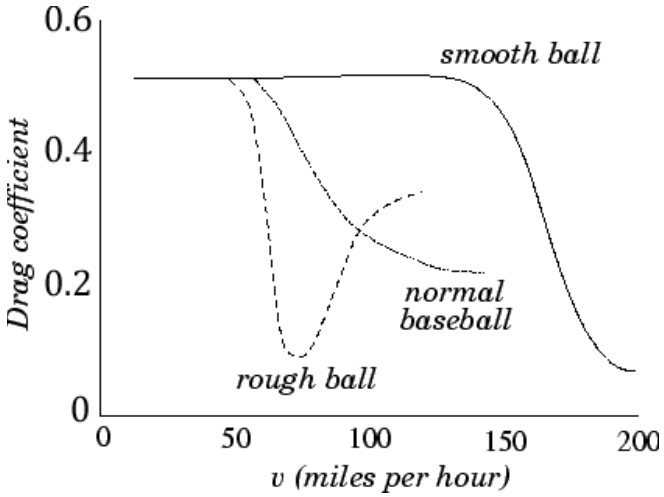
(561 ft)

2) Modify the variables in your MATLAB script to match a baseball. Set the drag coefficient to zero and confirm your MATLAB code gets the same range.

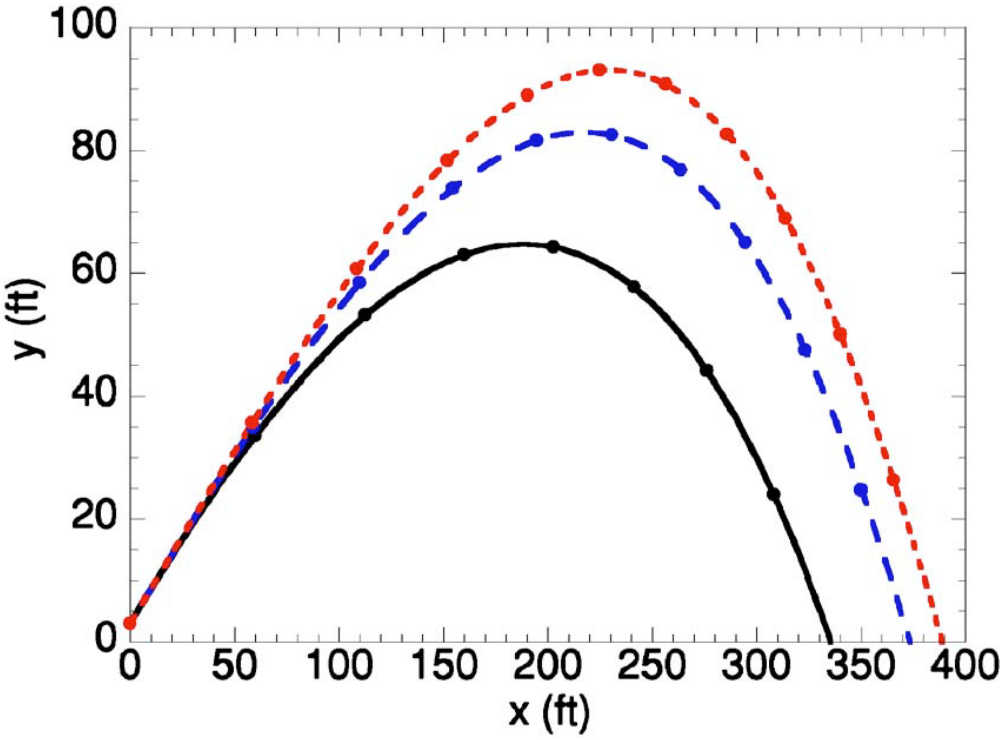
2) Now put air resistance back into your MATLAB script and determine how far the baseball goes this time. You should use a drag coefficient of about 0.3 (see plot below, the average speed for Schoop's ball as it slowed down during flight was about 92.5 mph). What positional initial conditions should you use? How does your calculated range compare with the statcast distance for the home run? Remember it should be slightly less since back spin will tend to increase the actual range and hang time compared with your calculations[2].

(360 ft)

[2] "The effect of spin on the flight of a baseball," Alan M. Nathan, American Journal of Physics, Volume 76, Number 2, February 2008, pages 119-124



Variation of the drag coefficient, C_D, with speed, v, for normal, rough, and smooth baseballs.
From The physics of baseball, R.K. Adair (Harper & Row, New York NY, 1990)



This is figure (7) from Nathan/AmJPhys V76N2pp119-124. Original caption follows:
Fig. 7. Calculated trajectories of a hit baseball with an initial speed of 100 mph, angle of 30°, height of 3 ft, and backspin of 0 rpm (solid), 1000 rpm (long-dashed), and 2000 rpm (short-dashed). The points indicate the location of the ball in 0.5 s intervals. These calculations utilize the CD values of Adair (Ref. 7) and the CL values from the parametrization of Sawicki et al. (Ref. 5).