

2-D Projectile Motion -- A Bottle Cap Tossing Simulation

Newton's 2nd law of motion states that the rate of change of momentum is equal to the sum of the applied forces, or

$$\frac{d}{dt}(m\vec{v}) = \vec{F} \quad (1)$$

For constant mass systems, this reduces to

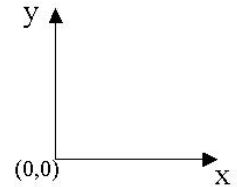
$$m \frac{d}{dt} \vec{v} = \vec{F} \quad (2)$$

This is a vector equation. In a 2-D system, for example, this becomes

$$m \frac{d}{dt} v_x = F_x \quad \text{and} \quad m \frac{d}{dt} v_y = F_y \quad (3)$$

where F_x represents the sum of all x-directed forces acting on mass m , and F_y is the sum of all the y-directed forces.

Consider an x-y coordinate system as shown in the sketch, where +y is in the upward direction and +x is to the right. If the only force present on an object of mass m is the downward force of gravity (i.e. we neglect air friction), then the resulting equations are simply



$$m \frac{d}{dt} v_x = 0 \quad \text{and} \quad m \frac{d}{dt} v_y = -mg \quad (4)$$

Both these 1st order differential equations are separable and are simple to solve, as follows:

In the x direction,

$$dv_x = 0 \, dt \quad \text{and} \quad v_x = c_1 = v_{x0}$$

where v_{x0} is the initial x-directed velocity.

In the y-direction,

$$dv_y = -g \, dt \quad \text{and} \quad v_y = -gt + c_2$$

which leads to

$$v_y = -gt + v_{y0}$$

where v_{y0} is the initial y-directed velocity.

Thus, eqn. (4) leads to

$$v_x(t) = v_{x0} \quad \text{and} \quad v_y(t) = v_{y0} - gt \quad (5)$$

These expressions represent the velocity components versus time.

Now we know that velocity is simply the rate of change of position, or

$$\frac{dx}{dt} = v_x \quad \text{and} \quad \frac{dy}{dt} = v_y \quad (6)$$

These give two more relatively simple separable ODEs that can be solved as follows:

In the x-direction,

$$dx = v_x dt, \quad dx = v_{x0} dt, \quad x(t) = v_{x0} t + c_1$$

which gives

$$x(t) = v_{x0} t + x_0$$

and for $x(0) = x_0 = 0$, we have

$$x(t) = v_{x0} t \quad (7a)$$

Similarly for the y direction, we have

$$dy = v_y dt, \quad dy = (v_{y0} - gt) dt, \quad y(t) = v_{y0} t - \frac{1}{2} gt^2 + c_2$$

which gives

$$y(t) = v_{y0} t - \frac{1}{2} gt^2 + y_0$$

or, in standard form, we have

$$y(t) = y_0 + v_{y0} t - \frac{1}{2} gt^2 \quad (7b)$$

Equations (7a) and (7b) give the x,y coordinates of the object versus time, t.

Now, instead of specifying the initial values of v_x and v_y (i.e. v_{x0} and v_{y0}), one usually writes these in terms of the initial speed (magnitude of \vec{v}) and the initial angle relative to horizontal, θ . In particular, the velocity vector can be written as

$$\vec{v} = v_x \hat{i} + v_y \hat{j}$$

with the magnitude given by

$$v = |\vec{v}| = \sqrt{v_x^2 + v_y^2}$$

and the individual velocity components written as

$$v_x = v \cos \theta \quad \text{and} \quad v_y = v \sin \theta$$

Thus, at $t = 0$, we have

$$v_{x0} = v_0 \cos \theta_0 \quad \text{and} \quad v_{y0} = v_0 \sin \theta_0 \quad (8)$$

The above development gives all the equations necessary for describing the trajectory of an object under the influence of only a gravity force. It is important to emphasize that these

equations are valid when the only force on the object is due to a constant gravitational acceleration. In particular, they assume that air resistance is negligible, which leads to the approximation that the x-directed velocity is constant and that the y-directed acceleration is constant.

As an application of the above equations, let's consider the following situation. Assume you are sitting at your desk and you open a bottle of soda. However, the garbage can is on the other side of the room. Instead of getting up and walking over to the wastebasket, you decide to attempt to toss the bottle cap into the trash can from your current position. Let's assume the geometry illustrated in Fig. 1 for our current application.

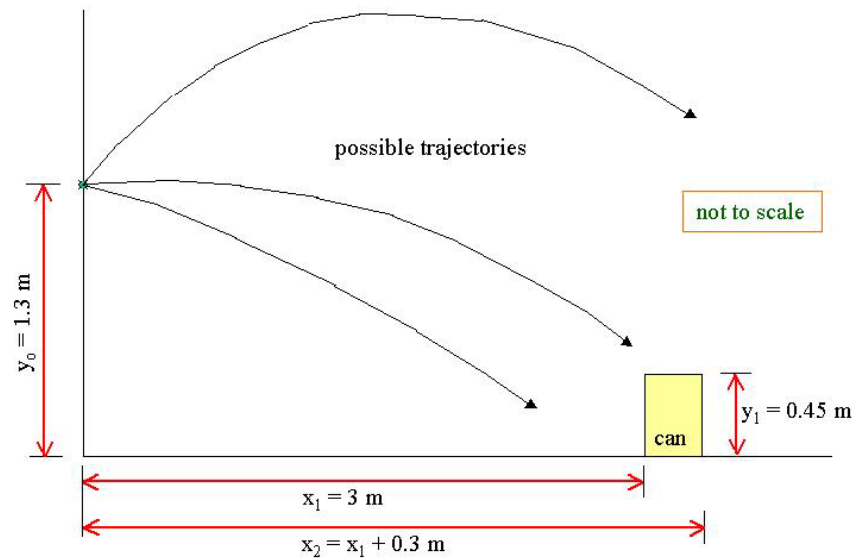


Fig. 1 Rough geometry for bottle cap tossing simulation.

The initial height of the bottle cap represents the height of your hand with your arm extended while you are in a seated position. The trash can is roughly 0.45 m tall and it has a diameter of about 0.3 m. The front edge of the can is approximately 3 m from your seat.

Now, the only other data needed for the simulation are v_0 and θ_0 (the initial speed and angle). For the current simulations, we will assume that $v_0 = 6$ m/s (see note below). Thus, the only free variable is the initial angle, θ_0 , and our goal will be to determine, via graphical analysis, what values of θ_0 lead to success!

Note: Repeated tests of throwing a bottle cap with average effort gave an average speed of between 15 – 25 ft/s, which translates to 4.6 – 7.6 m/s. Thus, we will choose a value of 6 m/s as a reasonable value for v_0 .

A simple algorithm for solving this problem is as follows:

1. Set geometry parameters and fixed initial conditions.
2. Set up a vector of possible initial angles and a vector of time values. Note here that a simulation time of about 2 seconds is reasonable and that no special consideration will be given for computing negative y values (we will simply force the plots to show $y > 0$ with use of the *axis* command).
3. Since the x and y positions of the projectile are functions of both time, t, and the initial angle, θ_0 , we will store the values of $x(t, \theta_0)$ and $y(t, \theta_0)$ in 2-D arrays. These positions will be computed for all t values using vector arithmetic, and an outer loop will be used to treat one value of θ_0 at a time. Equations (7a) and (7b) will be evaluated within the *for ... end* looping structure in Matlab.
4. With known values of $x(t, \theta_0)$ and $y(t, \theta_0)$, we can plot y(t) versus x(t) for each θ_0 to show a range of possible trajectories. We can also sketch an outline of the garbage can directly on the plot, so we can easily see what initial angles lead to success in getting the bottle cap into the trash can. This will give a graphical solution to the problem...

The above algorithm was implemented into Matlab program **projectile2d_1.m**, and a listing is given in Table 1. The resultant plot from the first part of the program is presented in Fig. 2. From this plot we can see that there are two ranges of angles that lead to success -- a direct path, low-angle trajectory that leads to the shortest path to the can, and a more indirect path with a high-angle trajectory that takes a longer time to reach the can. Although it is a little difficult to see because of the multitude of trajectories, it appears that low-angle paths at about 5 - 15 degrees and high-angle trajectories in the 60 - 70 degree range are the only successful ones that occur for an initial speed of 6 m/s.

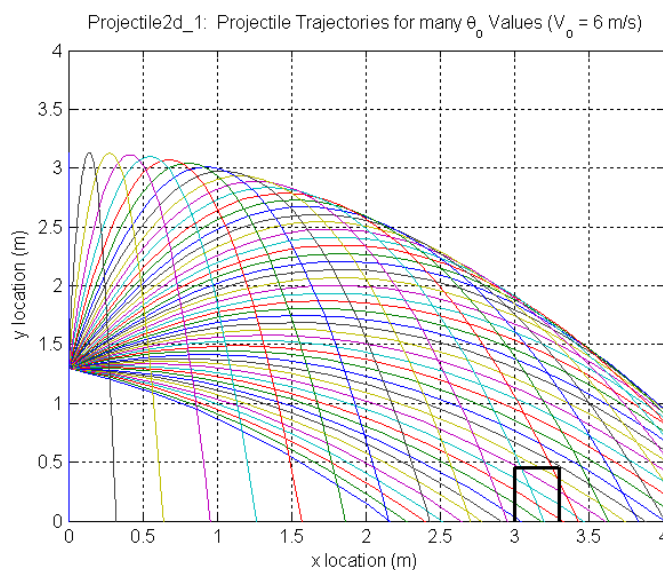


Fig. 2 Various projectile trajectories in the x-y plane for $v_0 = 6$ m/s.


```

% now evaluate desired quantities
%
% Note: "dot arithmetic" is used here to evaluate the desired quantities at Nt
% different times in a single statement -- this vector processing capability
% is one of the many very nice features available in Matlab. However, we also
% want to do all these different calculations for Nth different values of theta0.
% This will be handled with a looping structure. When finished, we will have a
% matrix of x and y values --> since they are functions of two variables, t = time
% and theta0 = initial angle of bottle cap.
%
    x = zeros(Nt,Nth);    y = zeros(Nt,Nth);    % initialize variables
    for i = 1:Nth        % do the calculations
        th = theta0(i);
        vx0 = Vo*cos(th);    vyo = Vo*sin(th);
        x(:,i) = vx0*t;
        y(:,i) = yo + vyo*t - (g/2)*t.*t;
    end
%
% plot y(t) vs x(t) (for all values of theta0) along with an outline of the can
nfig = nfig+1;    figure(nfig)
plot(x,y), grid, hold on
plot([x1 x1],[0 y1],'k-',[x2 x2],[0 y1],'k-',[x1 x2],[y1 y1],'k-','LineWidth',2)
axis([0 4 0 4])
title(['Projectile2d\ 1: Projectile Trajectories for many \theta_o Values (V_o = ', ...
    num2str(Vo),' m/s)'])
xlabel('x location (m)'),ylabel('y location (m)')
hold off
%
% from above, we can see that there are two ranges of angles that lead to success
% direct path -- low angle trajectory that leads to the shortest path to the can
% indirect path -- high angle path that takes a longer time to reach the can
%
% for the current case with Vo = 6 m/s, it appears that the low angle paths are at
% about 5 - 15 degrees and the high angle successful trajectories are in the 60 - 70
% degree range.
%
% to explore these specific angles, let's put in a interactive loop and ask the user
% for specific values of theta0
%
% first let's draw the x-y grid and the garbage can in a new figure window
nfig = nfig+1;    figure(nfig)
plot(0,yo,'ks'), grid, hold on
plot([x1 x1],[0 y1],'k-',[x2 x2],[0 y1],'k-',[x1 x2],[y1 y1],'k-','LineWidth',2)
axis([0 4 0 4])
title(['Projectile2d\ 1: Projectile Trajectories in X-Y Plane (V_o = ', ...
    num2str(Vo),' m/s)'])
xlabel('x location (m)'),ylabel('y location (m)')
%
% now loop over user-specified values of theta0 and plot x-y trajectory for each
cont = 1;
while cont == 1
    th_d = input('Input a value for theta0 in degrees: ');
    th = th_d*pi/180;
    vx0 = Vo*cos(th);    vyo = Vo*sin(th);
    xx = vx0*t;    yy = yo + vyo*t - (g/2)*t.*t;
    plot(xx,yy,'LineWidth',2),gtext(['\theta_o = ',num2str(th_d),' ^o'])
    cont = menu('Perform another simulation?',' ...
        'Yes -- I want to try another angle', ...
        'No Thanks -- I have seen enough');
end
hold off
%
% end of program

```

To explore these specific angles further, an interactive loop that asks the user for specific values of θ_0 was implemented towards the end of **projectile2d_1.m**. After a little trial and error, the successful low-angle range was determined to be approximately 8 – 11.5 degrees. Similarly, the high altitude angles that were successful occur at about 64 – 66.5 degrees. A final run that just highlights these angles was made and the resultant plot is given in Fig. 3. Any initial angles outside this range simply miss the target. Thus, you need to be pretty skilled if you want to be consistently successful -- since the range of successful angles is fairly small.

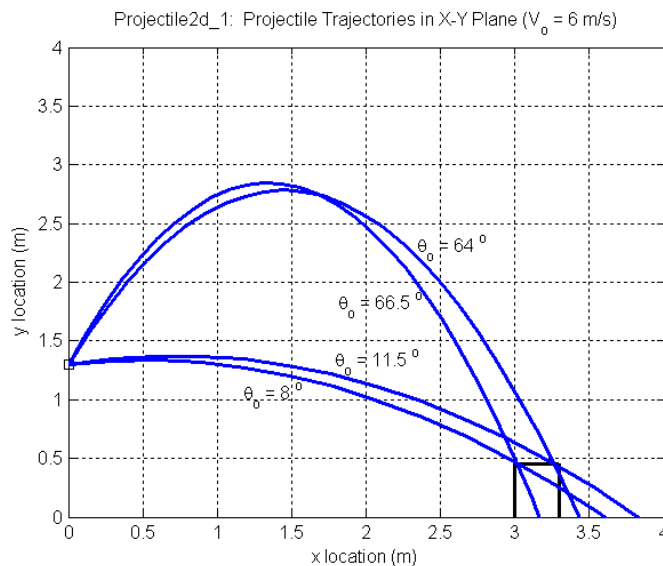


Fig. 3 Range of successful trajectories for bottle cap tossing simulation.

Well, this was a fun application that represents another good example of 2-D function evaluation in Matlab and the use of 2-D plotting techniques to help visualize and interpret functions of two independent variables. As apparent in Figs. 2 and 3, drawing an outline of the trash can directly on the projectile trajectory curves allowed us to easily visualize the successful paths. Matlab's **hold on** and **hold off** commands were used effectively to do this and allow us to access the same plot multiple times. This is also the first time we use the **while ... end** structure in Matlab. A **while ... end** looping arrangement is usually utilized when the number of passes through a set of code is not pre-determined. Instead, a conditional test is used to determine if the loop should be continued. Here, the code segment, **cont == 1**, determines if the variable **cont** is equal to **1** (note the double = sign for this test). If this is true, the loop continues. If it is false, the while loop is complete, and code execution continues after the **end** command.

Also new in this example is our use of the **menu** command. This is a simple structure that allows interactive selection from a list of options. The value of the variable is set based on the selection number that is made (note that the first text entry is the title within the menu that pops up on the screen). In this case, if the first option (the "Yes ..." path) is selected, **cont** is set to **1**, and the loop continues. If the "No ..." path is selected, **cont** is set to **2**, and the loop stops, as discussed above.

Well, this was a pretty simple example, but I hope it illustrates how some relatively simple graphics can be utilized effectively to solve real problems. Of course, we again emphasized the use of dot arithmetic to do the mathematical evaluations using Matlab's vector processing capability. Also, with the use of a *for ... end* loop, a *while ... end* structure, and logical variables (the result of the `cont == 1` test), along with the application of the interactive *menu* command, we are starting to learn how to do some real programming in Matlab. In fact, this example is a great introduction to our next subject -- "Programming in Matlab" -- where we will expand upon these ideas and introduce a number of other problem-solving techniques available within the Matlab language. It should be fun, fun, fun!!! I can hardly wait ...