Math 166: Calculus II

Integration

ECE 111 Introduction to ECE

Jake Glower - Week #6

Please visit Bison Academy for corresponding lecture notes, homework sets, and solutions

Math 166: Calculus II

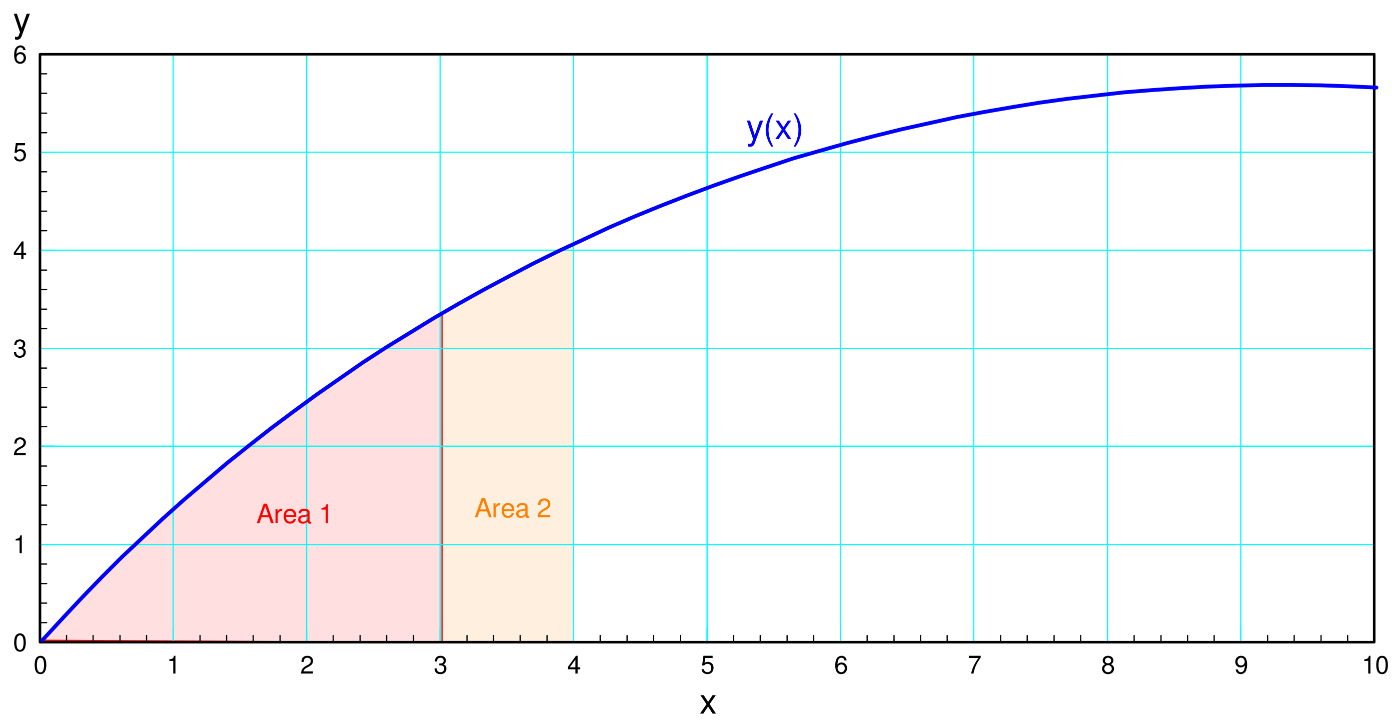
Topics

- Integration
- Numerical Integration
- Animation in Matlab (bouncing ball)
- Animation in Matlab (Shoot game)

Integration

Integration and differentiation both operate on functions:

- The derivative of a function is the slope
- The integral of a function is the area under the curve.



The integral of y(x) is the area under the curve to the left of x

Integration is useful: with it you can

- Determine the balance in your checking account given your daily deposits and withdrawals,
- Determining the velocity and position of a motor given its acceleration, and
- Do animation in Matlab where you determine the velocity and position of a ball as it bounces given its acceleration.

Integration & Differentiation

Integration and differentiation are also related:

- The integral of the derivative of a function is that function:

  $$\int \left(\frac{dy}{dx}\right) dx = y$$

- The derivative of the integral of a function is that function

  $$\frac{d}{dx}\left(\int y \cdot dx\right) = y$$

This is used in Math 166

- To find the integal of y(x)
- Find a function whose derivative is y(x)

  $$\frac{d}{dx}(a\sin(bx)) = ab\cos(bx)$$

Hence

$$a\sin(bx) = \int (ab\cos(bx)) \cdot dx.$$

Math 166 gets more difficult than Math 165

Example: Chain Rule:

$$\frac{d}{dx}(ab) = \frac{da}{dx} \cdot b + a \cdot \frac{db}{dx}$$

Integration by parts is the inverse of this:

$$ab = \int \left(\frac{da}{dx} \cdot b + a \cdot \frac{db}{dx} \right) dx$$

Translation:

- If you can express a function y(x) as

$$y(x) = \frac{da}{dx} \cdot b + a \cdot \frac{db}{dx}$$

then

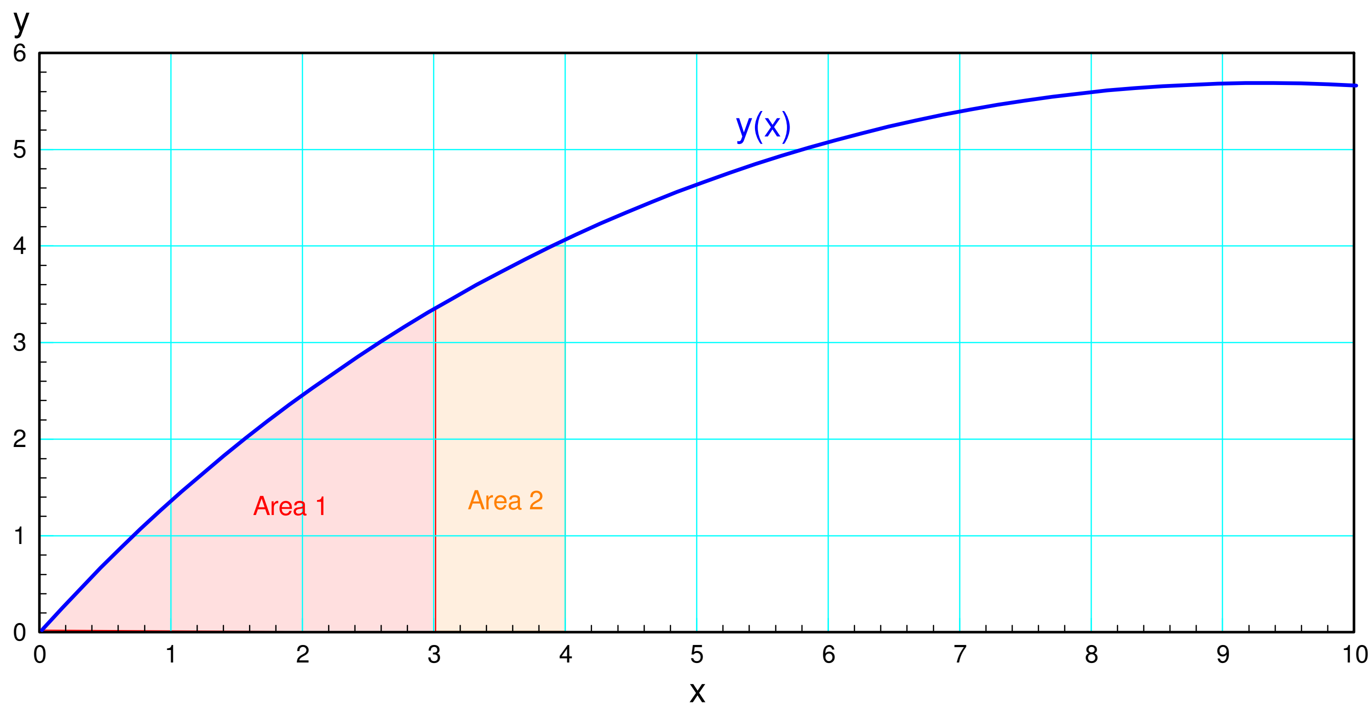
$$\int y(x) = ab$$

Coming up with a(x) and b(x) can be tricky...

Graphical Integration:

Fortunately, there is an easier solution

- The integral of a function is the area to the left
- The integral of y(x) at x=4 is
  - The integral of y(x) at x=3 (Area 1),
  - Plus the area from 3 to 4 (Area 2)

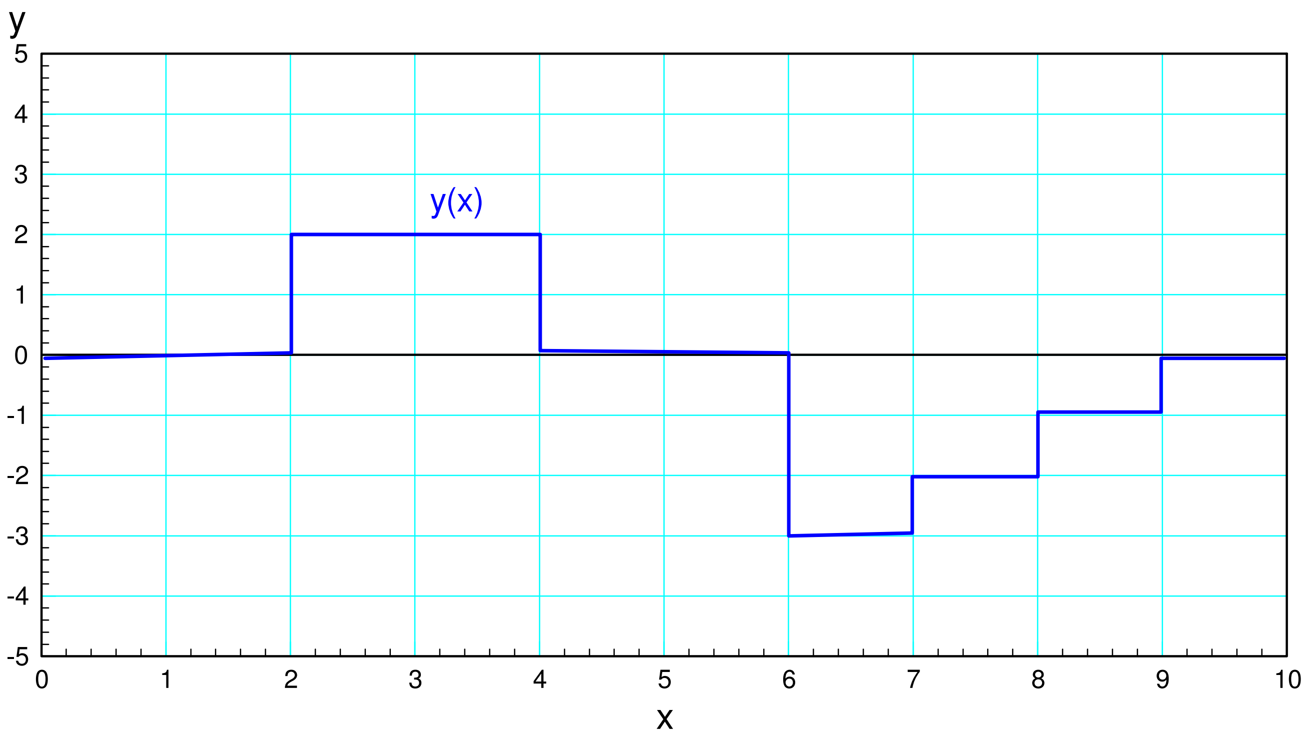


Example, sketch the integral of the following curve:

- Find the area under the curve to the left of point x

One way to think about this is

- Assume y(x) is how much money you're depositing at your bank
- The balance at any time is the integral (net balance)



Assume your starting balance is $0

x = 2:

- Nothing was added
- Balance ends up at 0

x = 4:

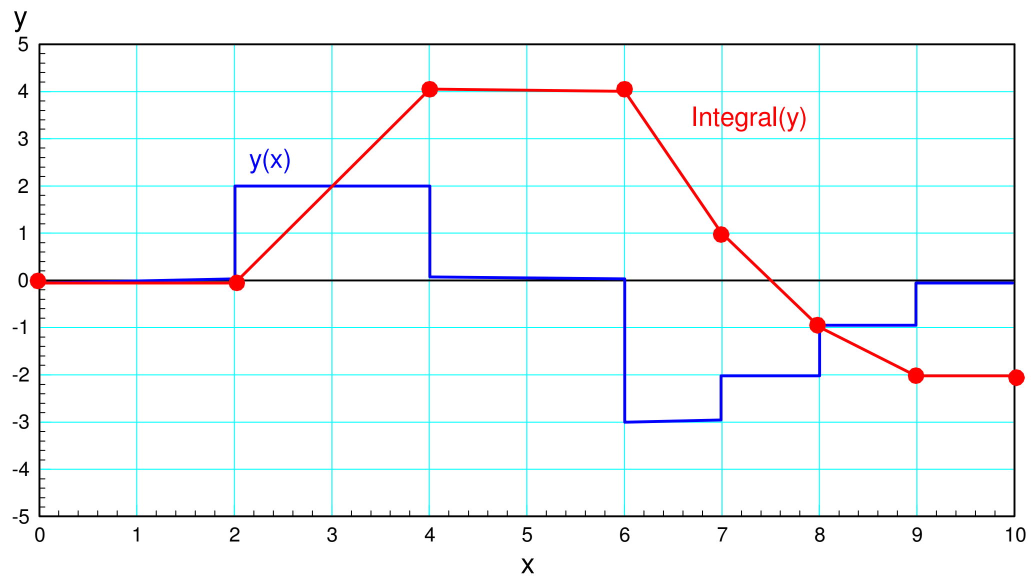
- Area under curve = +4
- Balance ends up at +4
- 0 + 4 = 4

x = 6:

- Nothing was added from 4..6
- Balance remains at +4
- 0 + 4 = 4

x = 7

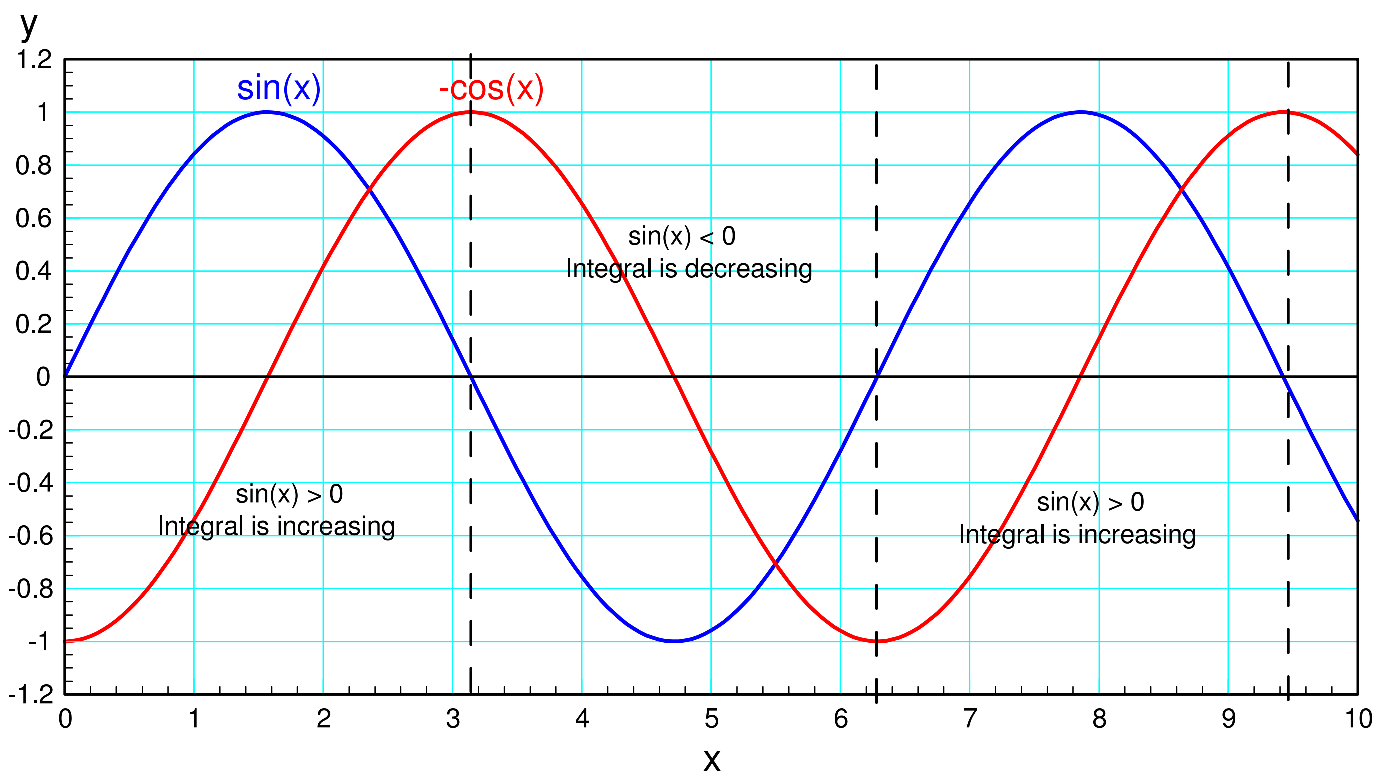
- Area under curve = -3
- Balance drops to +1



As a second example, in Math 166 you'll learn

$$\int \sin(x) \cdot dx = -\cos(x)$$

Graphically, this looks like the following:



When sin(x) > 0, its integral is increasing
When sin(x) < 0, its integral is decreasing

Numerical Integration

Matlab can integrate using numerical methods

The integral at point x is

- The net area to the left of x, or
- The net area to the left of (x-1), plus the area between x-1 and x.

The latter lets you set up a for-loop in Matlab.

At each point in x, the integral of y(x) is

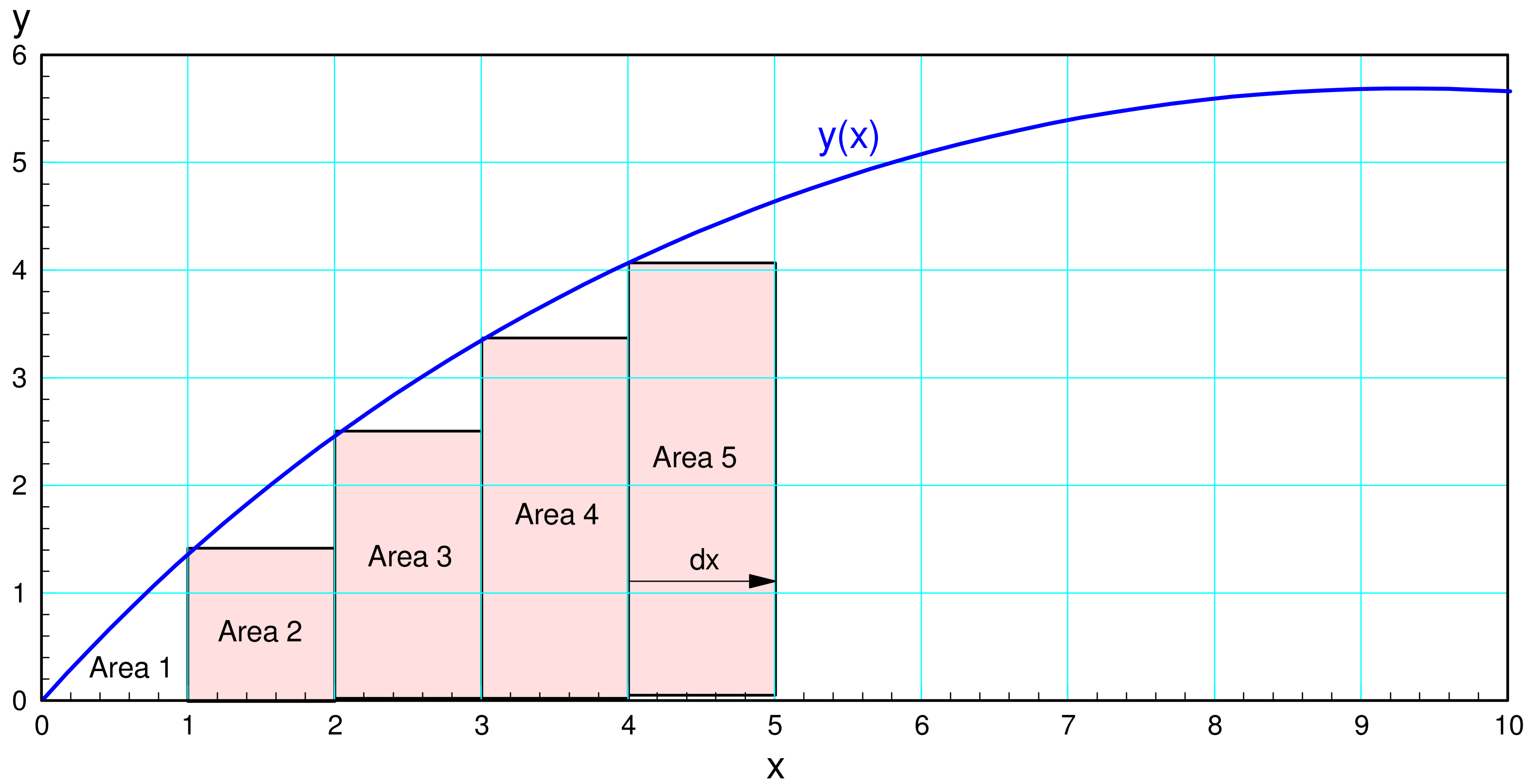
- The previous integral you calculated, plus
- The area between x-1 and x

There are several ways to calculate the area under a curve.

Euler Integration:

Approximate the area under the curve using rectangles.

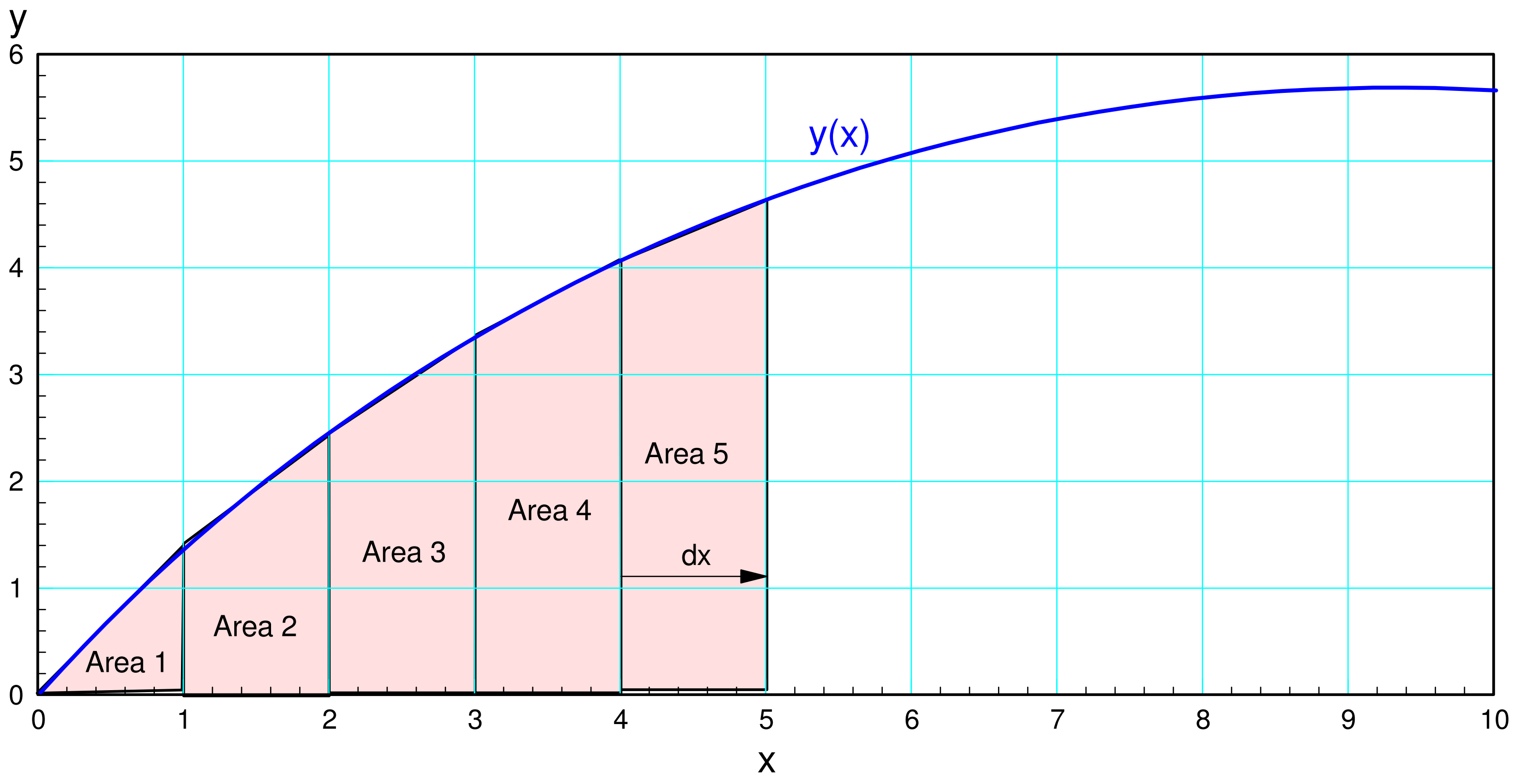
- Advantage: Simple
- Disadvantage: Slightly off



Bilinear Integration:

Approximate the area with trapezoids

- Better than Euler
- Still slightly off



Runge-Kutta Integration:

Approximate the area with polynomials

- More accurate, but
- More complicated

The higher-order the polynomial, the better the approximation.

All of these methods can be implemented in Matlab

Integrate.m

Let's implement bilinear integration.

$$\int_a^b y \cdot dx \approx \left(\frac{y(b)+y(a)}{2}\right) \cdot (b-a)$$

Matlab Code:

```
function [y ] = Integrate( x, dy )
% function [y ] = Integrate( x, dy )
% bilinear integration

npt = length(x);

y = 0*dy;

for i=2:npt
   y(i) = y(i-1) + 0.5*(dy(i) + dy(i-1)) * (x(i) - x(i-1));
end

end
```

Check vs. a known function

- Always a good idea

From before

$$\frac{d}{dx}(2\sin(3x)) = 6\cos(3x)$$

meaning

$$\int 6\cos(3x)dx = 2\sin(3x)$$

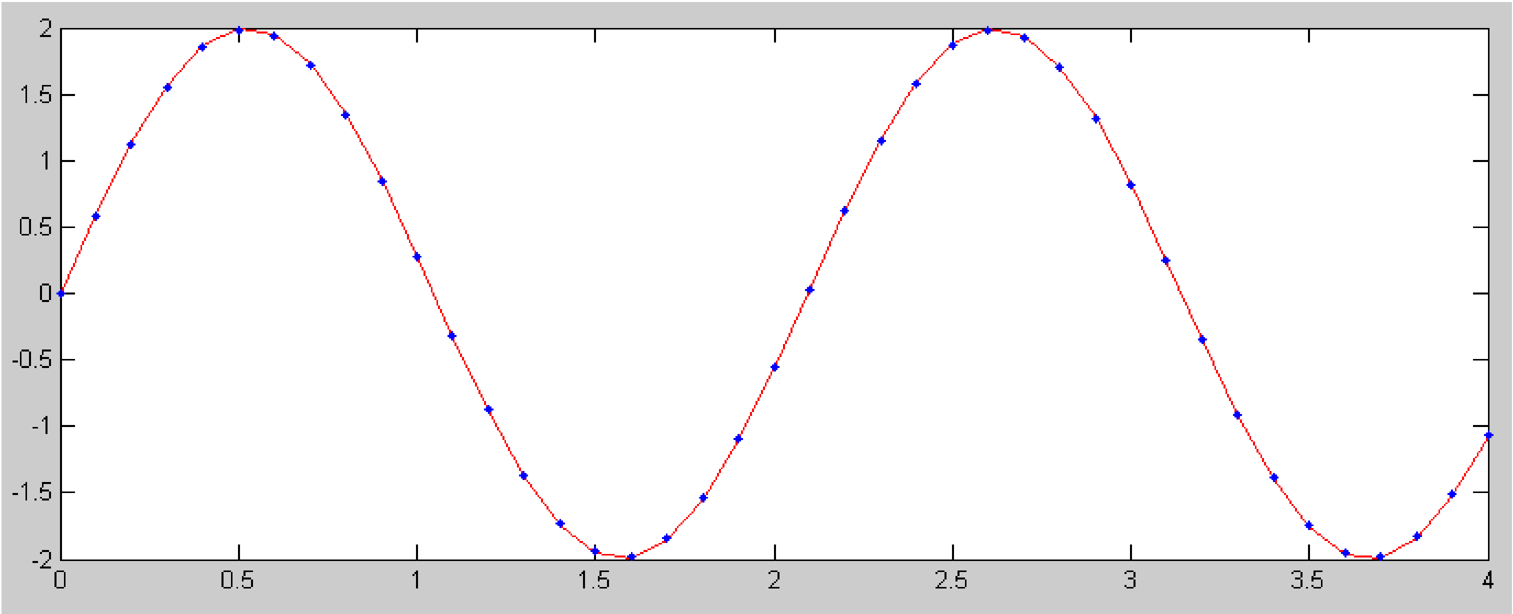
Let

$$dy = 6\cos(3x)$$

$$y = 2\sin(3x)$$

Check in Matlab:

```
>> x = [0:0.1:4]';
>> y = 2*sin(3*x);
>> dy = 6*cos(3*x);
>> plot(x,y,'r',x,Integrate(x,dy),'b.');
```



Actual integral of 6cos(3x) (red) and numerical solution (blue dots)

Try another function

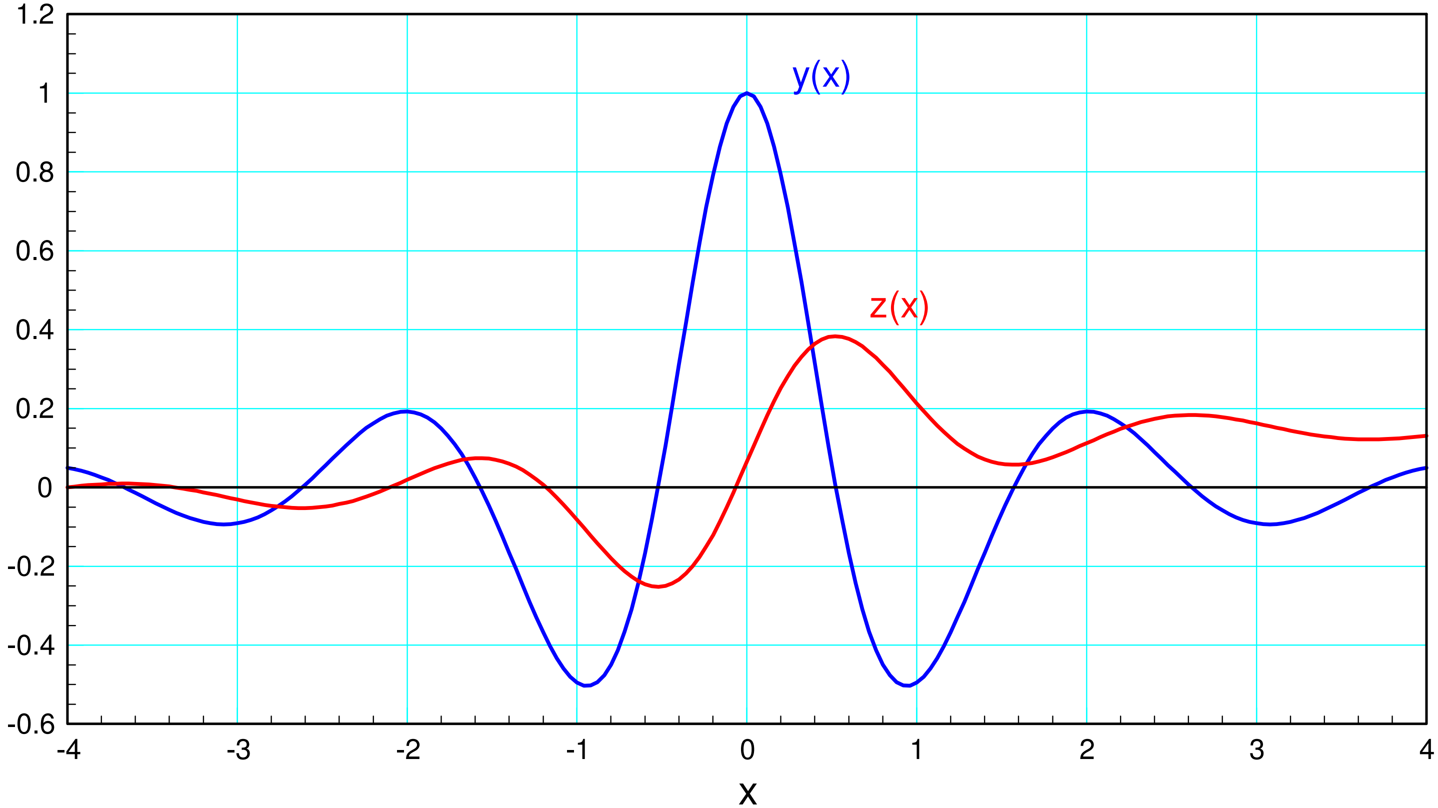
- The integral is hard to find using Math 166 techniques
- Easy to find using Matlab

$$y = \left(\frac{\cos(3x)}{x^2+1} \right)$$

$$z = \int y \cdot dx$$

As long as you can put y(x) into Matlab, you can find its integral. In Matlab:

```
>> dx = 0.01;
>> x = [-4:dx:4]';
>> y = cos(3*x) ./ ( x.^2 + 1 );
>> z = Integrate(x,y);
>> plot(x,y,'b',x,z,'r')
```



y(x) (blue) and its integral (red)

Path Planning using Integration

In our previous lecture, differentiation was used to determine the velocity and acceleration associated with a given path of a robot arm from point a to b. With integration, you can go the other way:

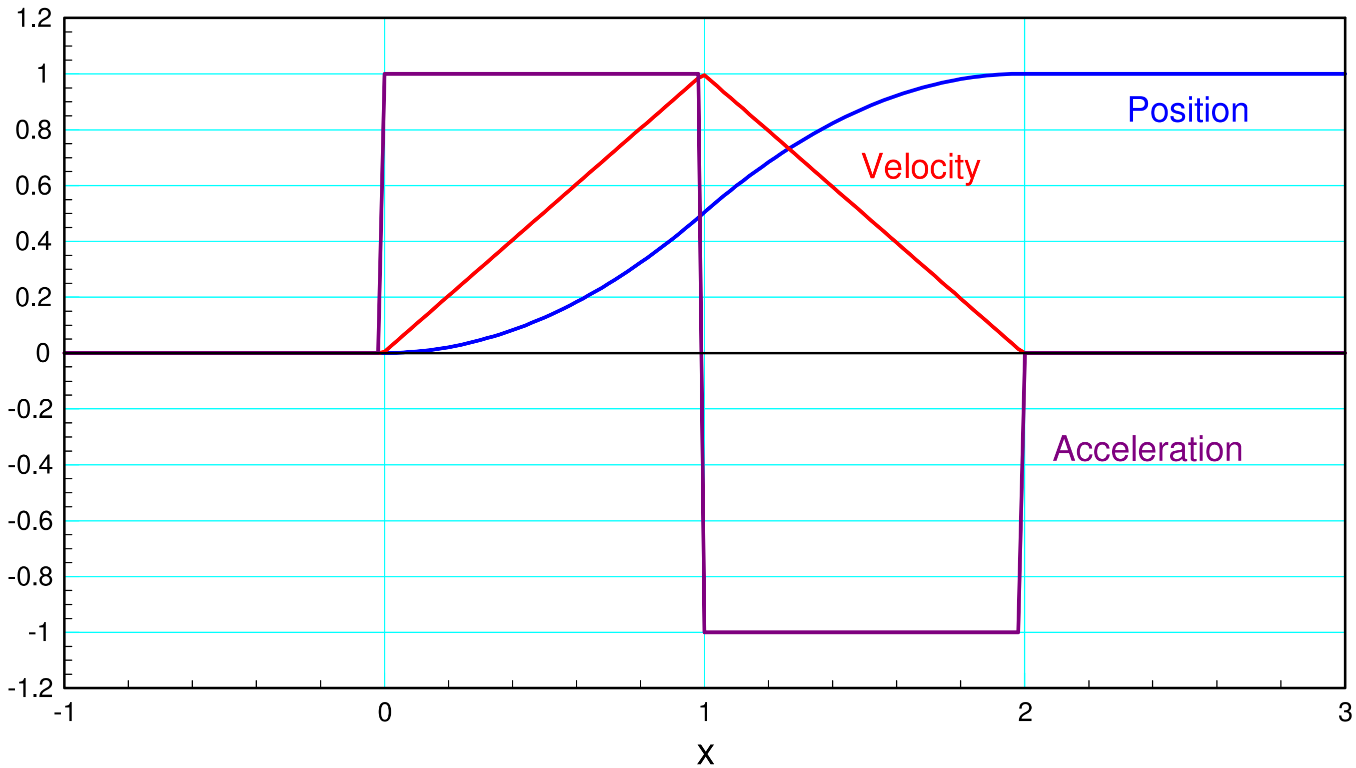
- Given the acceleration (i.e. the current to the motor), determine
- The implied velocity (1st integral), and
- The implied position (2nd integral).

Assume the acceleration is a constant

$$y'' = \begin{cases} +1 & 0 < t < 1 \\ -1 & 1 < t < 2 \end{cases}$$

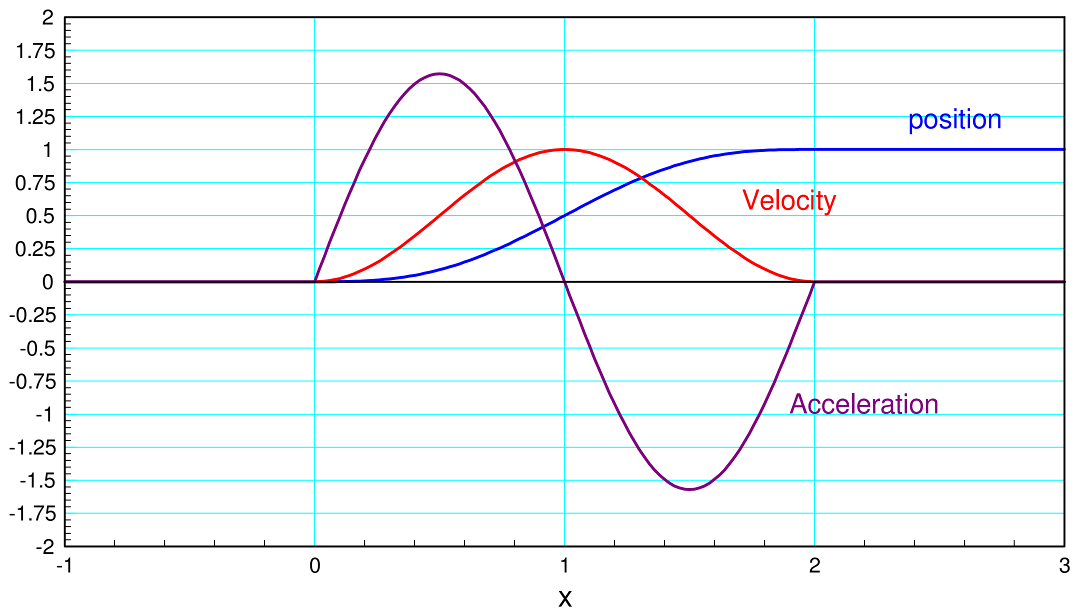
The velocity and position can be found using integration.

```
>> x = [-1:0.01:3]' + 1e-6;
>> ddy = 1*(x>0).*(x<1) -1*(x>1).*(x<2);
>> dy = Integrate(x,ddy);
>> y = Integrate(x,dy);
>> plot(x,y,x,dy,x,ddy)
```



Another path that avoids jump discontinuities:

```
>> ddy = sin(x*pi) .* (x>0) .* (x<2);
>> dy = Integrate(x,ddy);
>> y = Integrate(x,dy);
>> max(y)
    0.6366
>> ddy = ddy / 0.6366;
>> dy = dy / 0.6366;
>> y = y / 0.6366;
>> plot(x,y,x,dy,x,ddy)
```



Integration and Noise

Students tend to like differentiation

- Simply apply a set of rules to a function

Students tend to dislike integration

- You often have to guess the answer to find the answer
- Or guess some function so you can use integration by parts

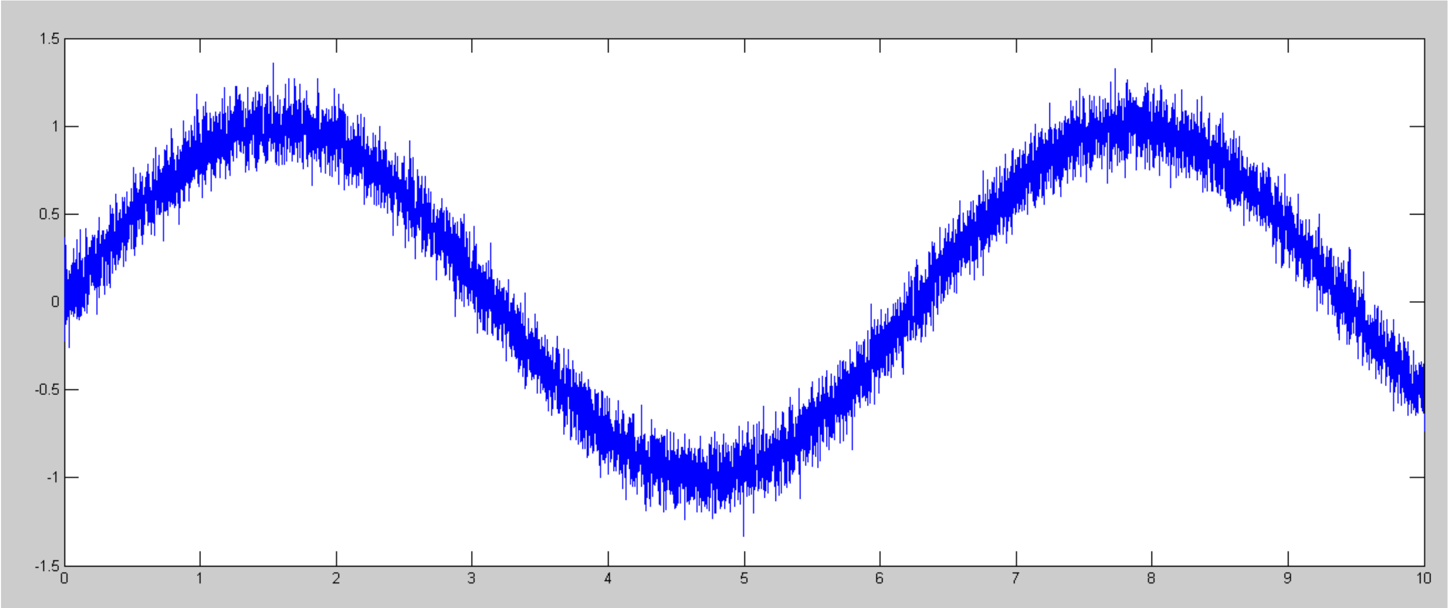
In practice, integration is preferred over differentiation

- Differentiation amplifies noise
- Integration removes noise

Example

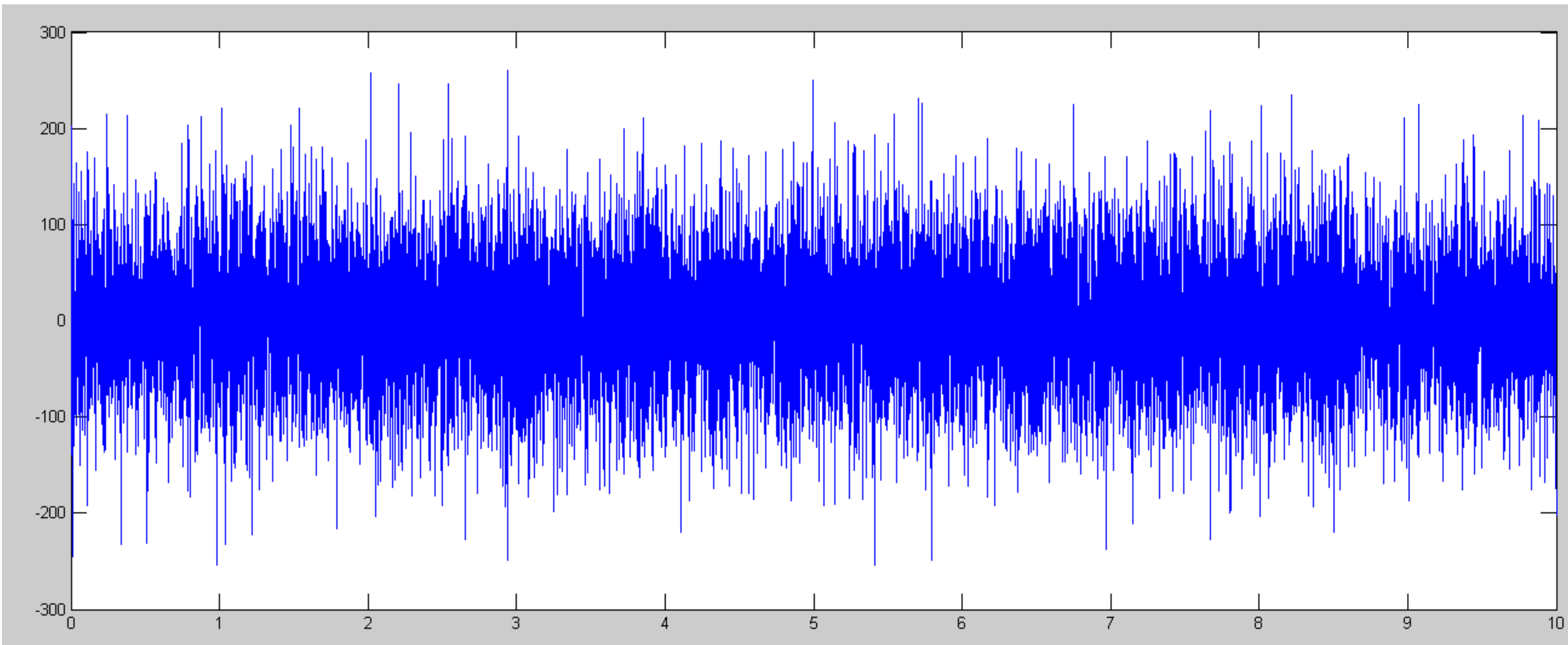
$$y(t) = \sin(t) + noise$$

```
>> t = [0:0.001:10]';
>> y = sin(t) + 0.1*randn(10001,1);
>> plot(t,y)
```



If you differentiate this signal, you amplify the noise:

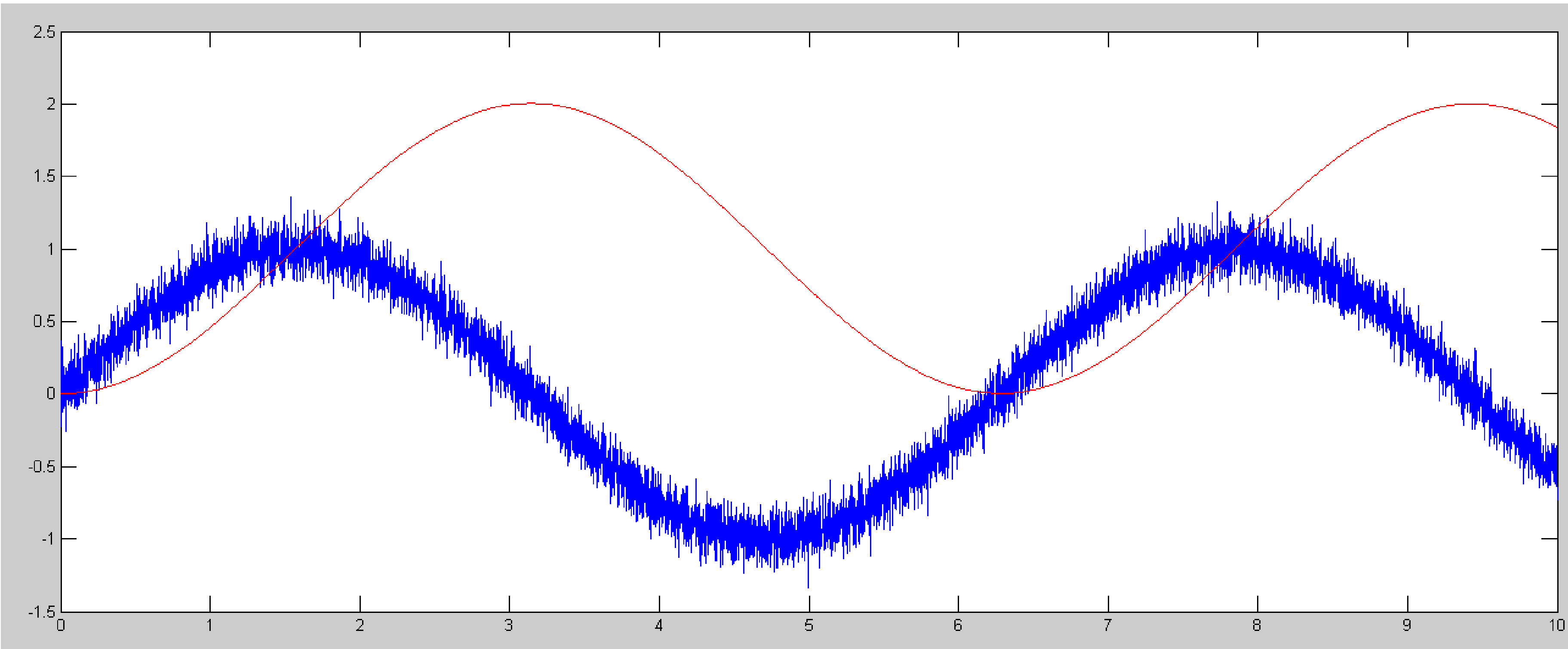
```
>> plot(t,derivative(t,y))
```



Derivative of y(t): differentiation amplifies noise

If you integrate this signal, you remove the noise

```
>> plot(t,y,'b',t,Integrate(t,y),'r')
```



y(t) (blue) and its integral (red). Integration cleans up a signal.

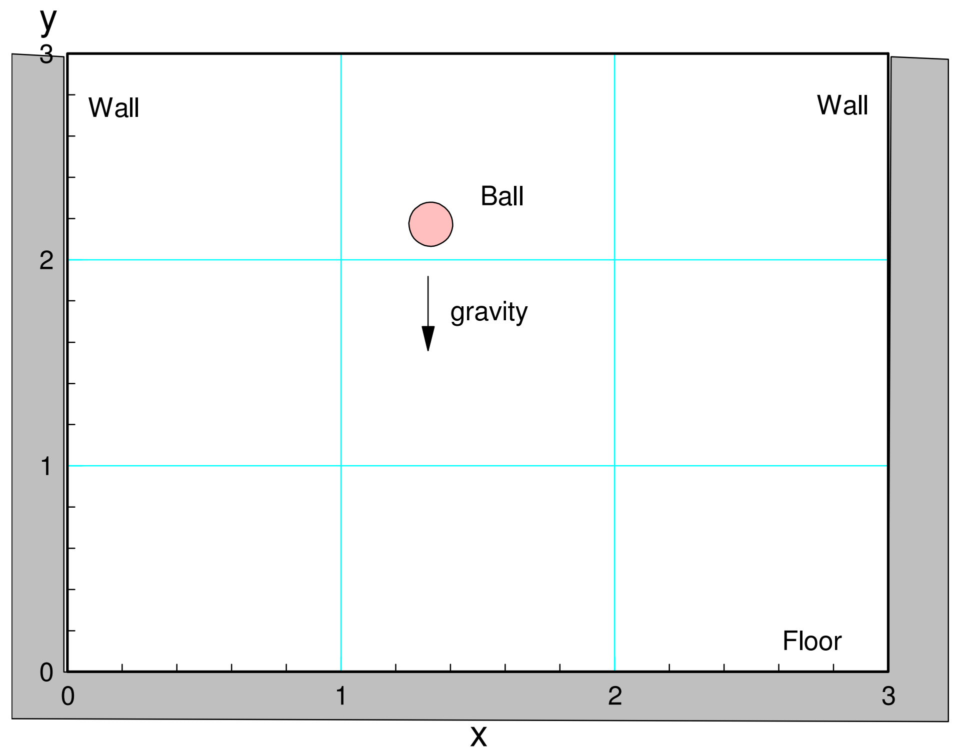
Moral: Avoid differentiation. Integration is OK though.

Fun with Integration: Bouncing Ball

Matlab has pretty good animation

Assume

- Gravity is in the -y direction
- Floor at y = 0
- Left wall at x = 0
- Right wall at x = 3
- If you hit the wall of floor, the velocity changes sign (bounces)



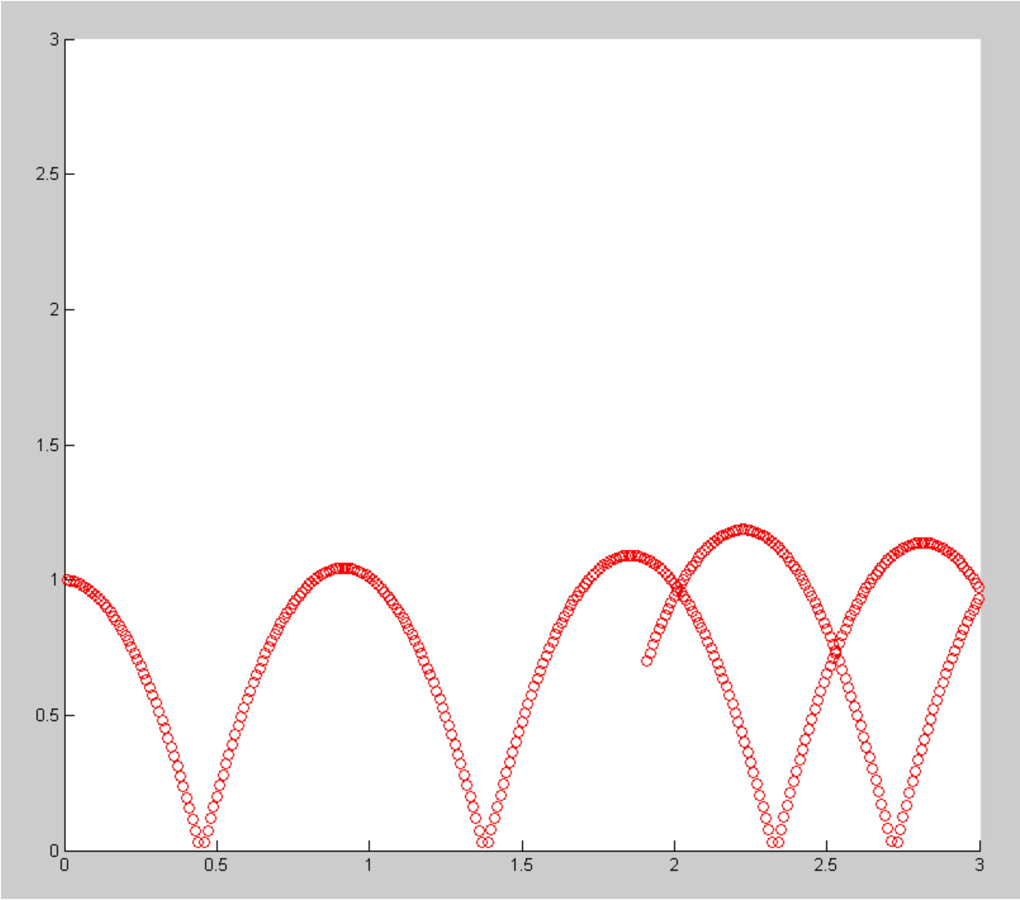
Matlab script

```
% Bouncing Ball
% Initial Conditions
x = 0;
y = 1;
dx = 1;
dy = 0;
t = 0;
dt = 0.01;
```

```
while(t<10)
    ddx = 0;
    ddy = -9.8;

    dx = dx + ddx*dt;
    dy = dy + ddy*dt;

   if(y<0) dy = abs(dy);  end
   if(x>3) dx = -abs(dx); end
   if(x<0) dx = abs(dx); end

   x = x + dx*dt;
   y = y + dy*dt;

   plot(x,y,'ro');
   xlim([0,3]);
   ylim([0,3]);
   pause(0.01);
end
```

Result:

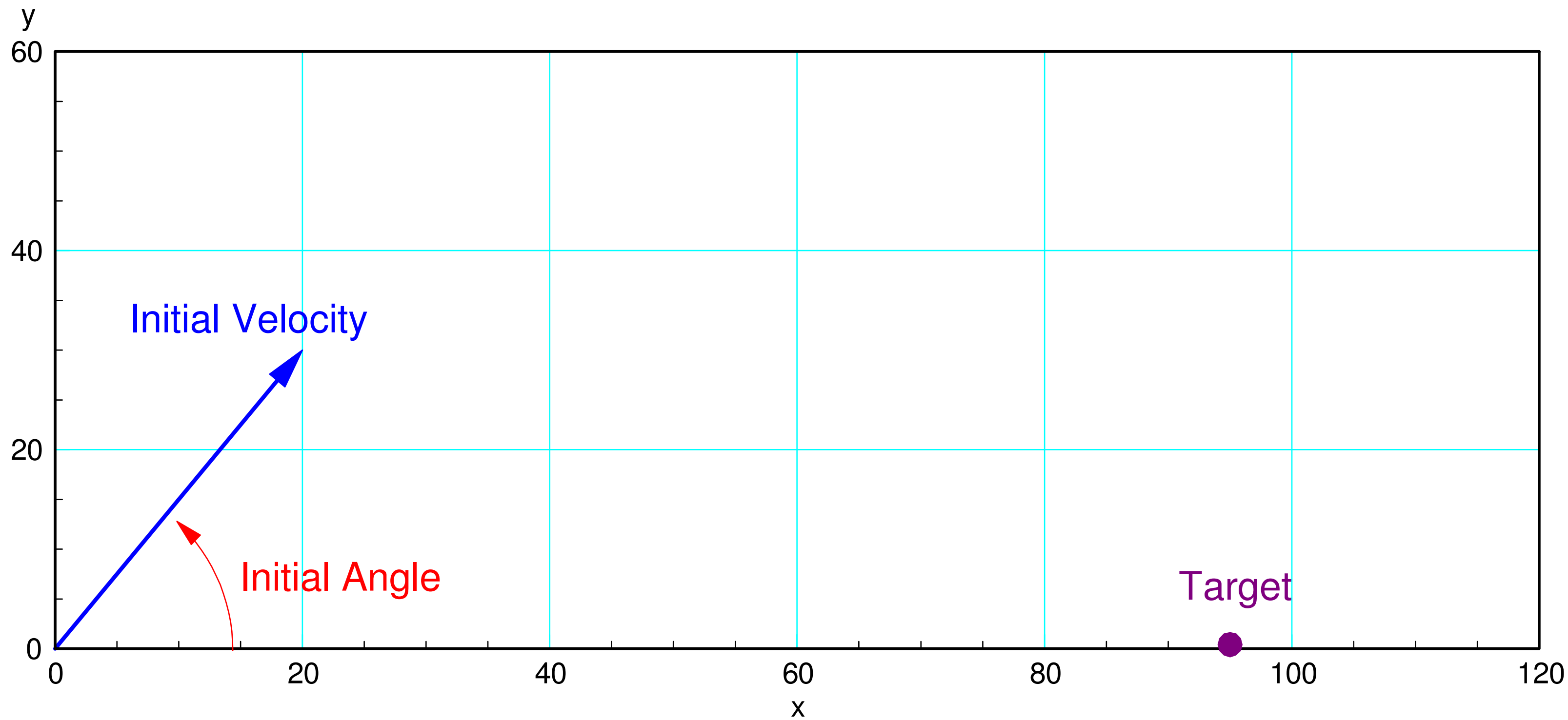
- Ball bounces off the floor and the walls
- (shows off better in Matlab)



Fun with Integration: Shoot Game

Launch a tennis ball. Call the function by specifying

- The initial velocity in m/s
- The initial angle in degrees, and
- The target position in meters.

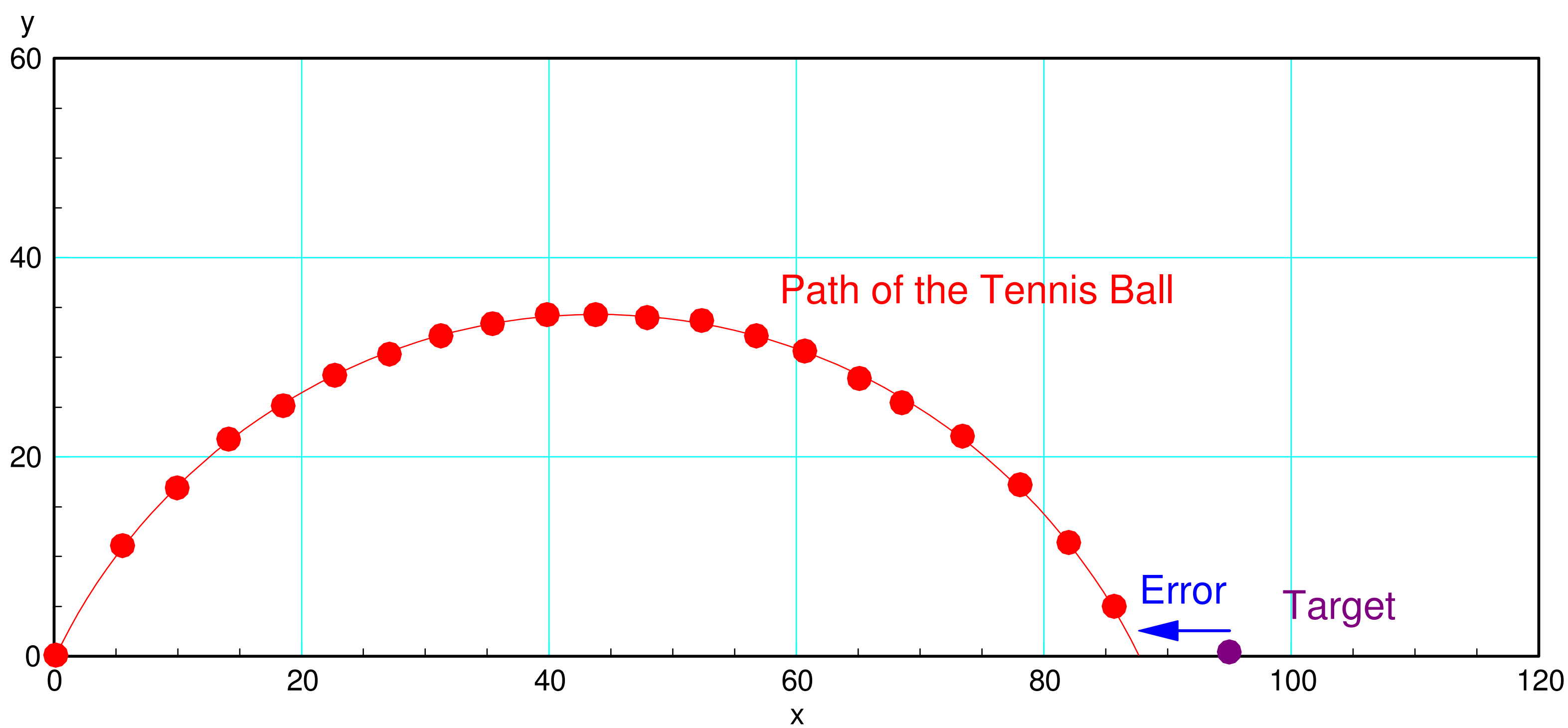


Use numerical integration

- Calculate the velocity based upon the acceleration
- Calculate the position based upon the velocity

When the tennis ball hits the ground (y=0)

- Return how far away you were from the target.

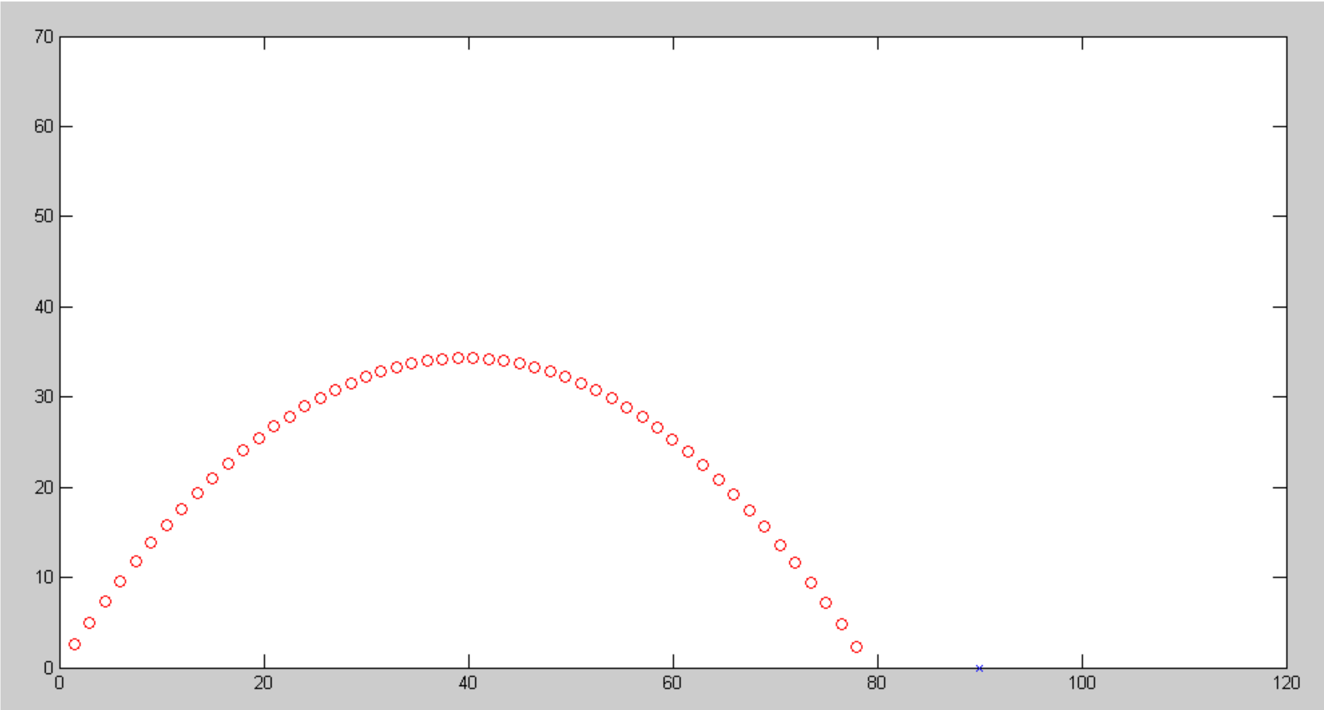


```
function [ Error ] = Shoot( Speed, Angle, Target )

 x = 0;
 y = 0;
 dx = Speed * cos(Angle*pi/180);
 dy = Speed * sin(Angle*pi/180);
 dt = 0.01;
 N = 0;

 plot(Target,0,'bx');
 xlim([0,120]);
 ylim([0,70]);
 hold on

 while(y >= 0)
   ddx = 0;
   ddy = -9.8;
   dx = dx + ddx*dt;
   dy = dy + ddy*dt;
   x = x + dx*dt;
   y = y + dy*dt;

   N = mod(N+1,10);
   if(N == 0) plot(x,y,'ro',Target,0,'bx'); end
   pause(0.01);
 end

x = x + y*(dx/dy);
Error = x - Target;
end
```

From the command window, you can call this function as

```
>> Shoot(30,60,90)

ans =  -10.3829
```

The tennis ball hit 10.3829 meters short of the target

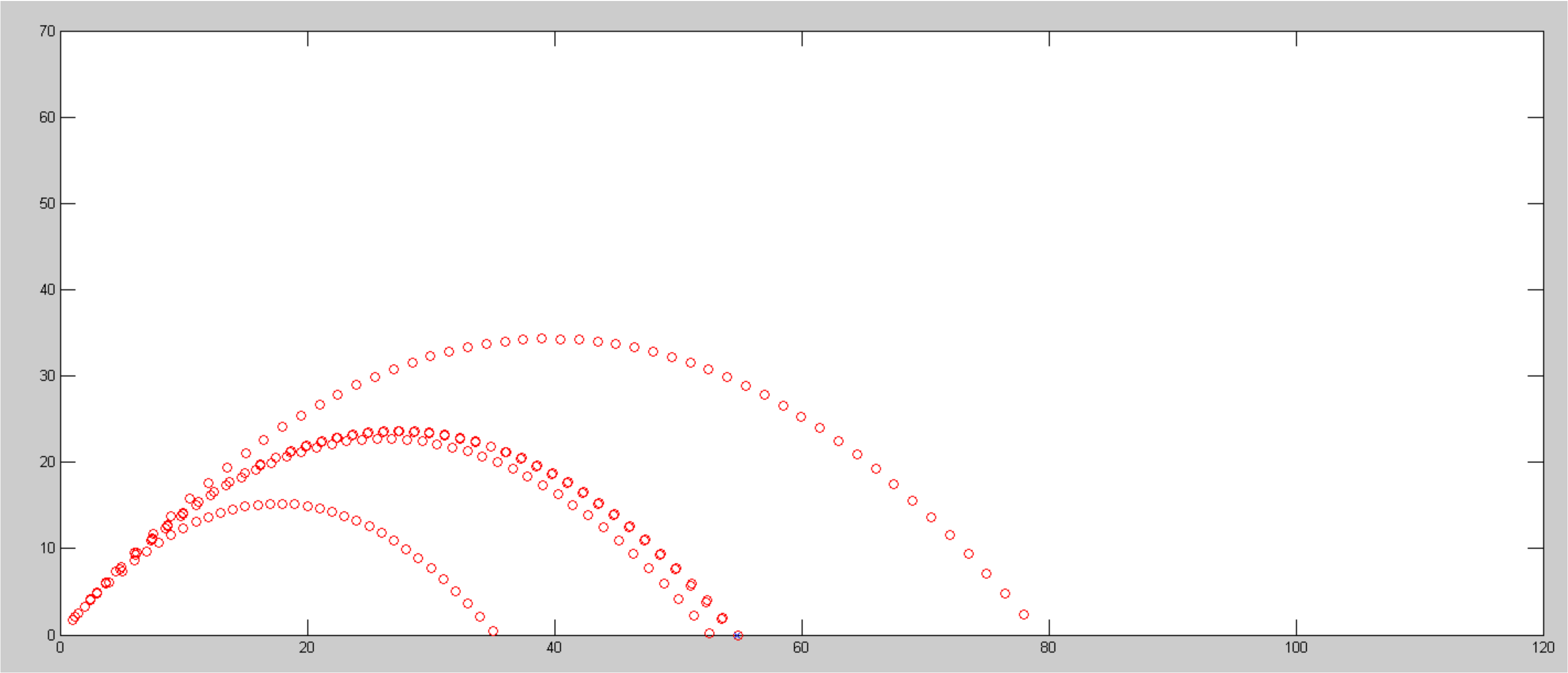


Hitting the target is a f(x) = 0 problem. Using California method:

```
Target = 50 + 50*rand;
clf

x0 = 20;
y0 = Shoot(x0, 60, Target);
x1 = 30;
y1 = Shoot(x1, 60, Target);
disp([0,x1,y1]);

for n=1:5
   x2 = x0 - (x1-x0)/(y1-y0)*y0;
   y2 = Shoot(x2, 60, Target);
   disp([n,x2,y2]);
   x0 = x1;
   y0 = y1;
   x1 = x2;
   y1 = y2;
end
```

This results in

```
  n          x        error
       0   30.0000   24.6189
  1.0000   24.4219   -2.1797
  2.0000   24.8756   -0.2055
  3.0000   24.9228    0.0021
  4.0000   24.9224   -0.0000
  5.0000   24.9224   -0.0000
```

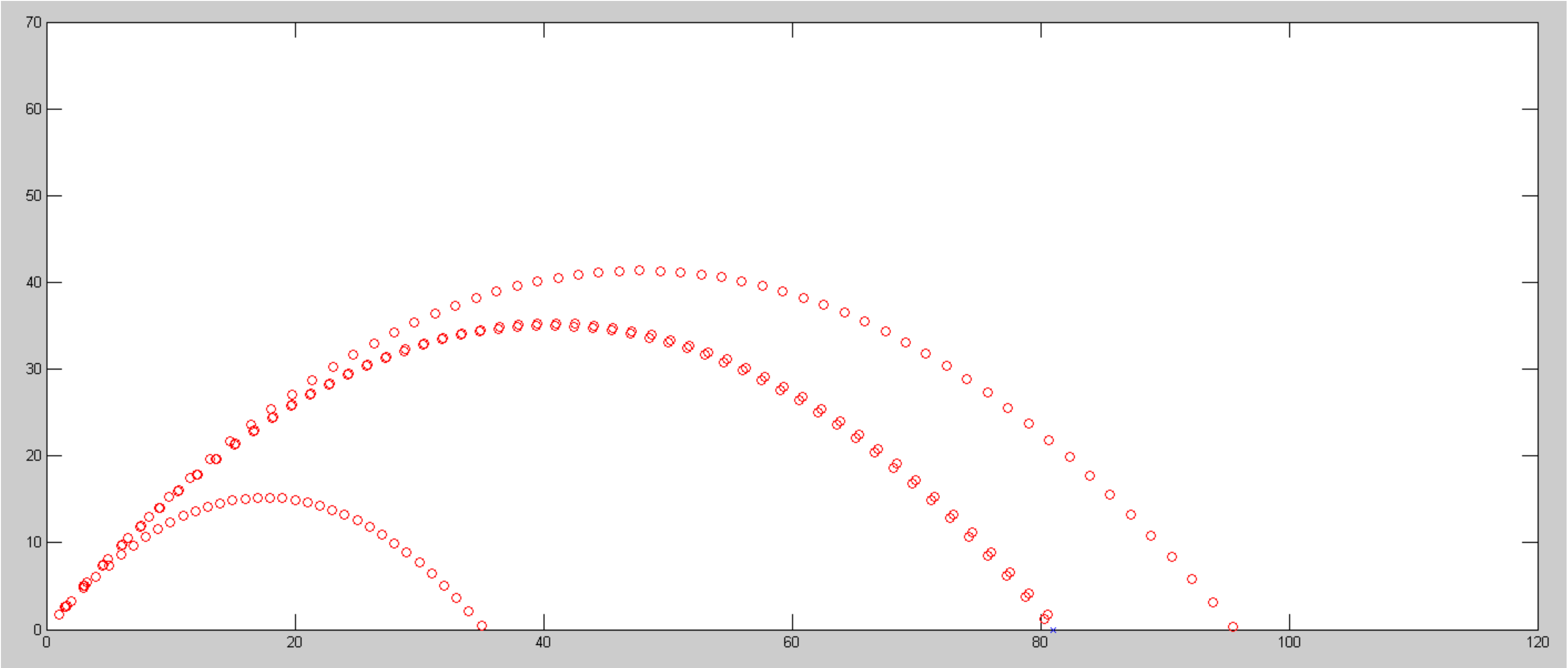


Using Newton's method to solve for f(x) = 0

```
Target = 50 + 50*rand;
clf

x2 = 20;

for n=1:5
   x0 = x2;
   y0 = Shoot(x0, 60, Target);
   disp([n, x0, y0])
   x1 = x0 + 0.1;
   y1 = Shoot(x1, 60, Target);
   x2 = x0 - (x1-x0)/(y1-y0)*y0;
end

disp(y0)
```

The results are

n	x	error
1.0000	20.0000	-45.7577
2.0000	32.9302	14.6581
3.0000	30.4131	0.5809
4.0000	30.3052	0.0020
5.0000	30.3048	0.0000



Summary:

Integration is pretty useful. With it, you can

- Determine the balance of your checking account given your deposits vs. time,
- Determine the path of a robotic arm given its acceleration, and
- Run animation in Matlab for bouncing balls, shooting tennis balls, and so on.

The nice thing about numerical integration is you can integrate any function you can get into Matlab.