Math 166: Calculus II

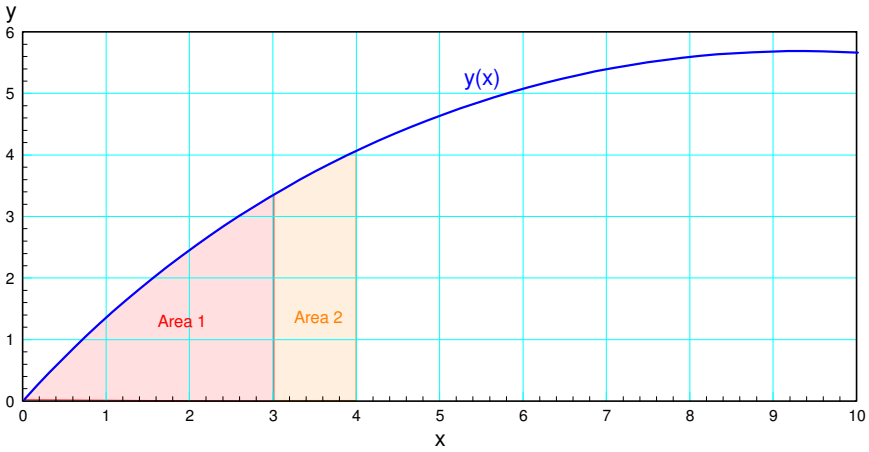
Topics

- 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 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

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

The latter is the method used in Math 166 to find the integral of a function. For example, from Math 165, you'll learn

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

Hence

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

Math 166 gets more difficult. The chain rule from Math 165 has

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

Integration by parts is the inverse of this:

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

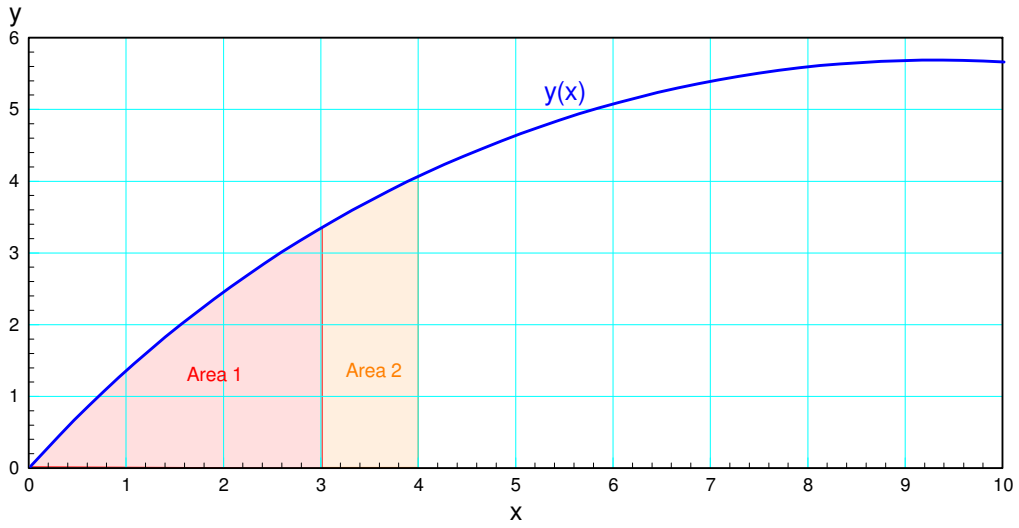
In other words, if you can express a function y(x) as the sum of two terms which are equal to two other functions in the form of

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

then the integral of y is ab. Finding functions a and b can be pretty tricky: Math 166 is not an easy class.

Fortunately for us, the former definition of integral leads to a much simpler numerical way to compute the integral of a function:

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



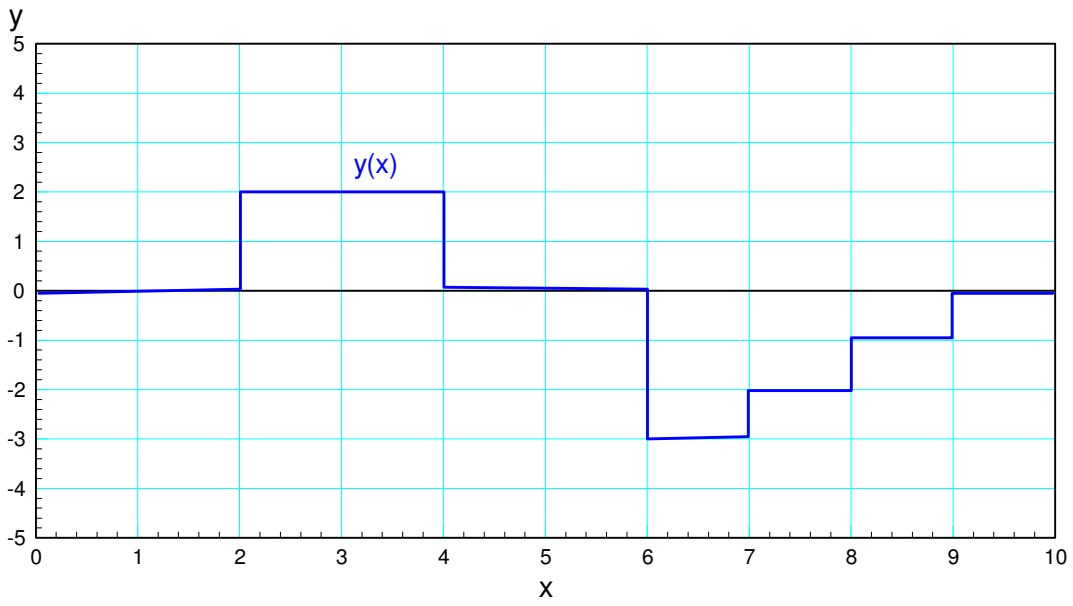
The integral of a function between points a and b is the area under the curve over this interval

Graphical Integration

Given a graph, y(x), you can determine the integral at point x by

- Adding up the total area under the curve up to time x, or
- Add to the previous area you computed at point (x-1), the area between (x-1) and (x).

For example, sketch the integral of the following curve:



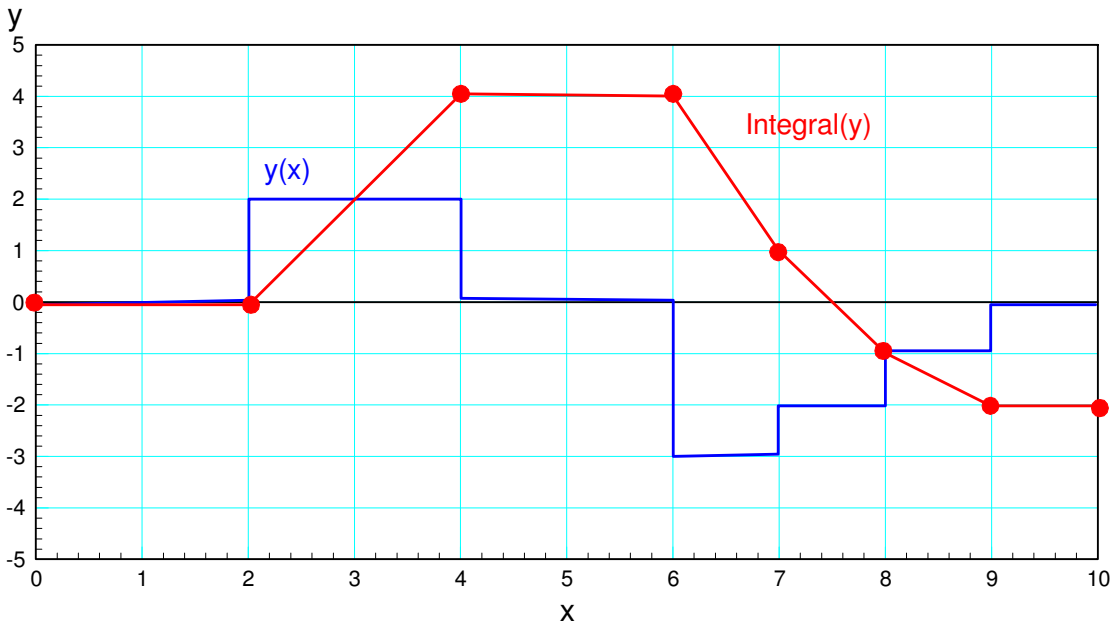
One way to think about this is

- Assume y(x) is how much money you're depositing into your checking account.
- The balance at any time is the integral: it's the net money you've added up to time x

One problem with integration is you have to know where you're starting from: your bank balance after ten days depends upon how much money you started at. This is termed *an integration constant.* Assume your bank account starts out at $0 (or the integral at x=0 is zero).

- 0<x<2: The area under the curve is zero. Add zero to the starting value (0)
- 2<x<4: The area under the curve is four. Add four to the previous total (net area = 4)
- 4<x<6: The area under the curve is zero. Add zero the previous total (net area = 4)
- 6<x<7: The area is minus three. Subtract 3 from the previous total (net area = 1)
- 7<x<8: The area is minus two. Subtract 2 from the previous total (net area = -3)
- 8<x<9: The area is minus one. Subtract 1 from the previous total (net area = -4);
- 9<x<10: The area is zero. The net area remains unchanged (net area = -4).

The integral is then a curve connecting these points:

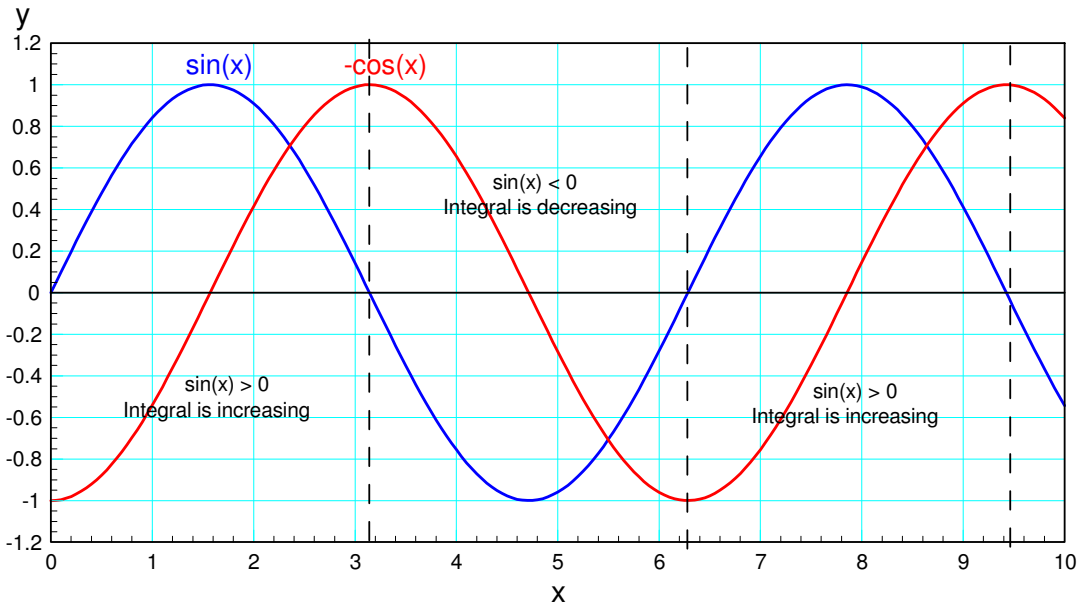


The integral of y(x) is the area to the left of x.

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

In calculus, you learn to integrate functions by hand. With Matlab, you can integrate functions using numerical methods.

The basis for numerical integration is 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

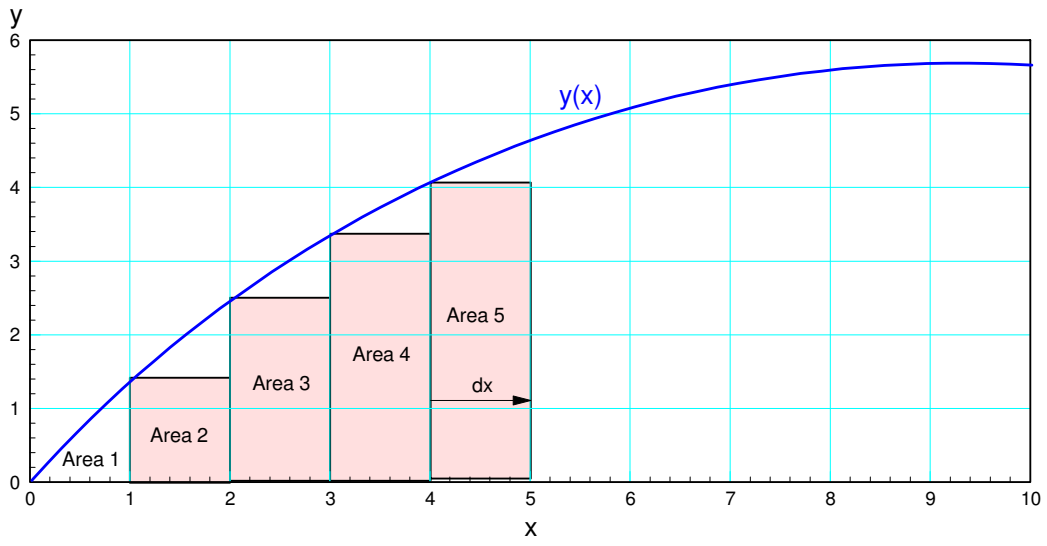
There are several ways to calculate the area under a curve.

Euler Integration: With Euler integration, you approximate the area under the curve using rectangles. The approach is

- You sample y(x) with spacing of dx (dx = 1 in the example below)
- The integral (the area under the curve) is the sum of the area of the rectangles up to time x

$$\int_x^{dx} y(x) \cdot dx \approx y(x) \cdot dx$$

The good thing about Euler integration is it's simple. The bad thing about Euler integration is it's slightly off: the rectangles miss some of the area. If you reduce the step size, dx, however, you can minimize this error.



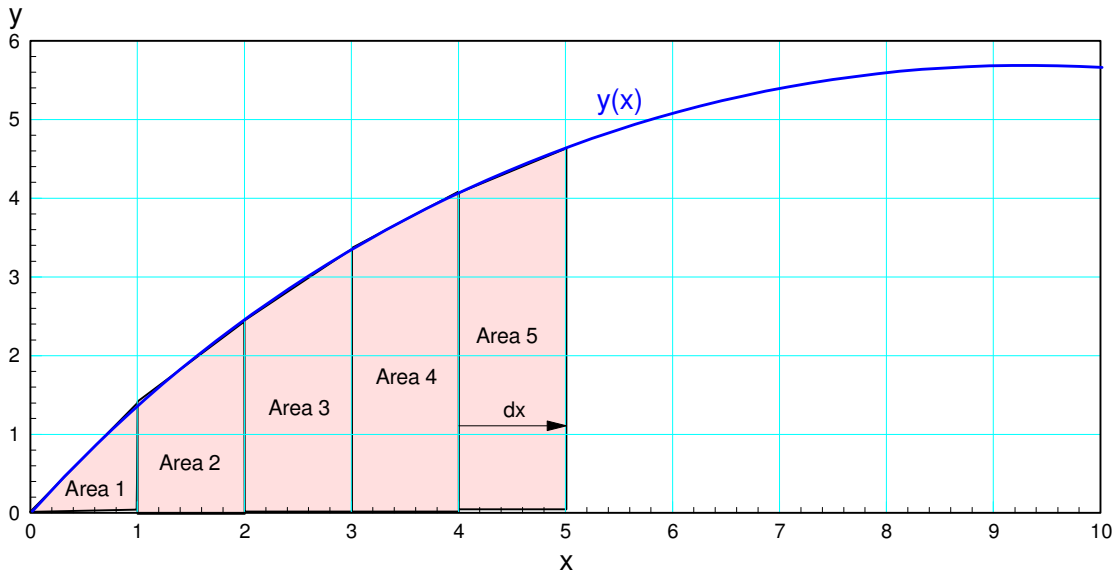
Euler Integration: The area under the curve is the sum of the area of rectangles

Bilinear Integration: A better form of integration is bilinear. Here, you approximate the area under the curve with trapezoids. The procedure is similar to before:

- Sample y(x) with a spacing of dx
- Compute the area of each section using a trapezoid:

$$\int_x^{x+dx} y(x)\cdot dx \approx \left(\frac{y(x+dx)+y(x)}{2}\right) dx$$

From the graph below, you can see that bilinear integration is much more accurate than Euler integration. As you reduce the step size, dx, the numerical solution gets more and more accurate as well.



Bilinear Integration: Approximate the area under the curve using trapezoids

Runge-Kutta Integration: An even better (but more complicated) form of integration is to approximate y(x) with a polynomial from x to x+dx. The higher-order the polynomial, the better the approximation.

Any of these integration schemes can be implemented in Matlab. For now, let's implement bilinear integration.

Assume you have a function y(x) in Matlab. To calculate the area to the left of x (i.e. the integral), a for-loop is used along with the calculation:

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

In the following code, the area under the curve at point x is equal to

- The area under the curve at point x-dx (i.e. the previous area you computed), plus
- The area under the curve from x-dx to x

In Matlab, a function to do this is as follows:

```
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
```

Any time you write a function in Matlab, it's good practice to test this function with something where you know the answer. For example, from differentiation

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

meaning

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

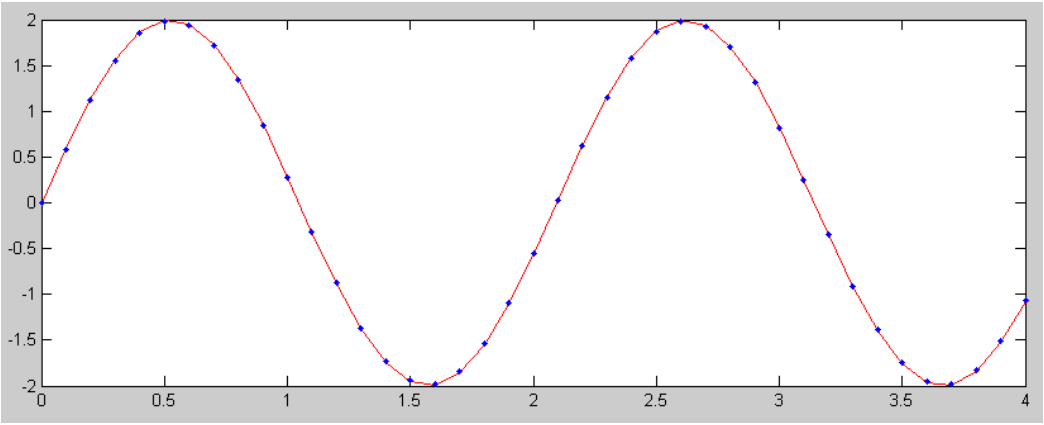
Let

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

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

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)

The computed values match the actual integral - meaning I'm fairly confident the subroutine is correct.

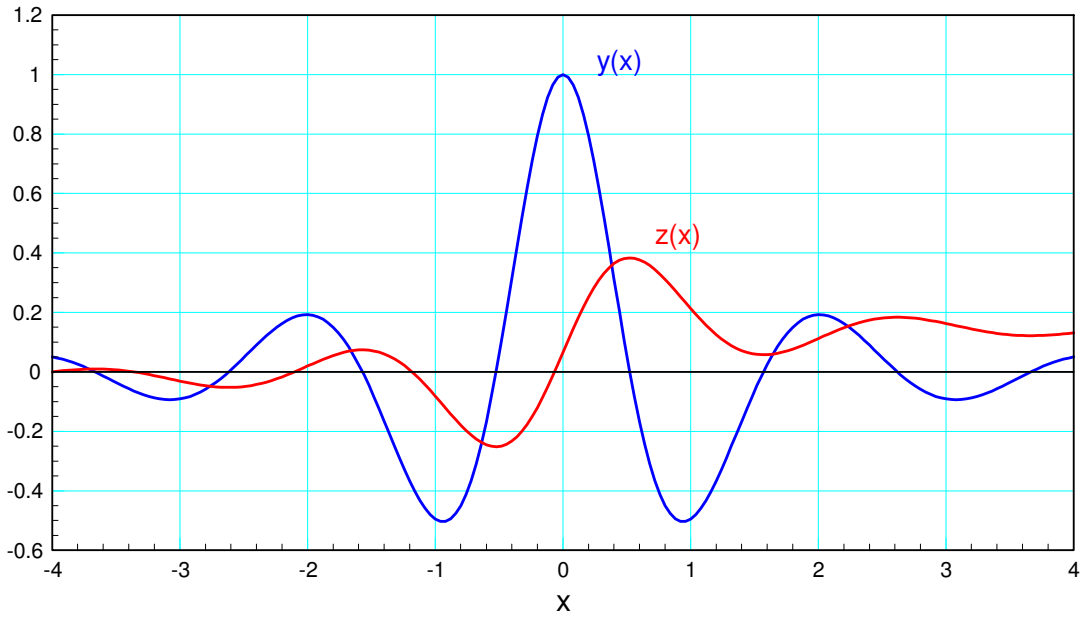
One *huge* advantage of numerical integration is you can find the integral of functions which are really hard to do by hand. For example, determine the integral of

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

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

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)

Note that with Matlab, I can find the integral of y(x) even if I can't find the integral by hand.

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).

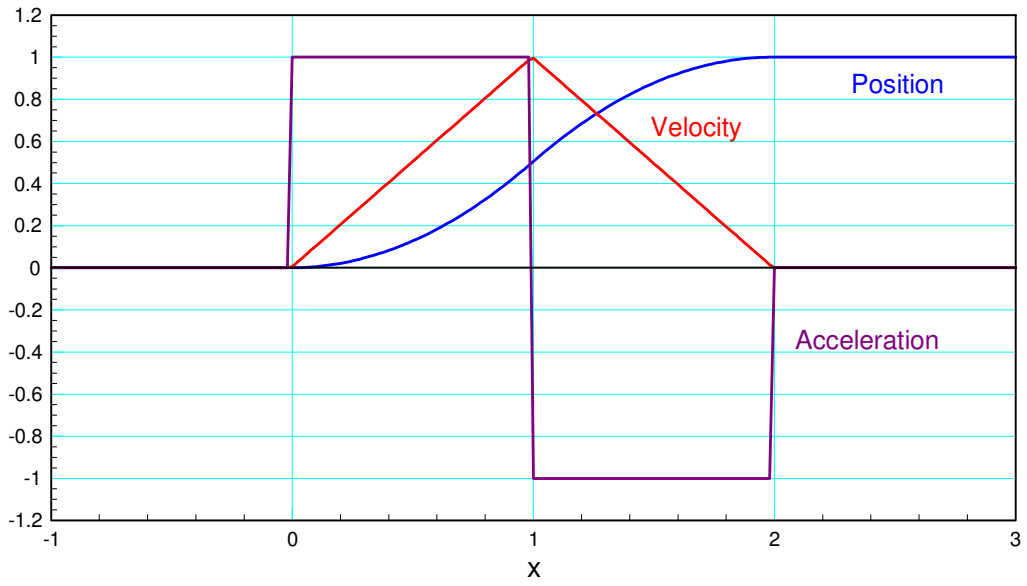
Assume the acceleration is a constant

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

The velocity and position can be found using integration. Each of these is scaled by a constant so that the final position is one

```
>> 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);
>> max(y)
     1.0000
>> plot(x,y,x,dy,x,ddy)
```

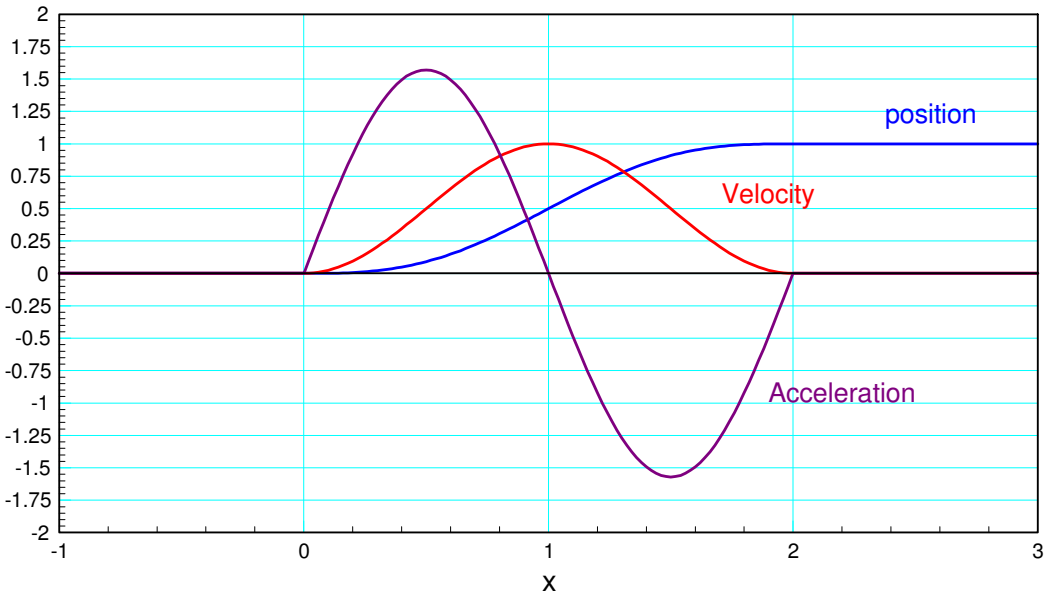
Since y(x) goes from 0 to 1, no scaling is needed



Path for a robotic arm with constant acceleration

This path has jump discontinuities in acceleration. If you want to avoid that, you could use the following:

```
>> 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)
```



Path Planning by specifying the acceleration (without jump discontinuities)

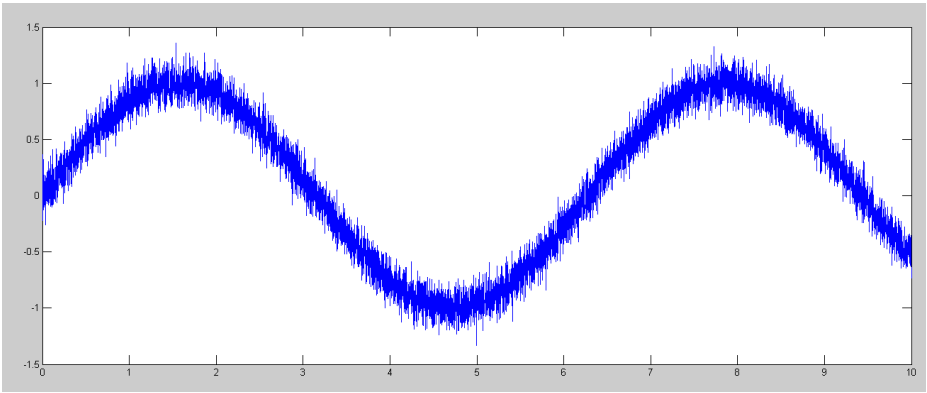
Integration and Noise

Typically, students prefer differentiation over integration. With differentiation, you simply apply a set of rules to find the derivative. With integration, you often have to apply several tricks to get the function into a form that you can integrate.

In practice, integration is preferred over differentiation. Almost all signals have noise. This can be caused by wires picking up radio signals, browning motion (electrons moving about creating small voltages), etc. When you differentiate a signal which has noise, you amplify the noise. When you integrate a signal with noise, you clean up the noise.

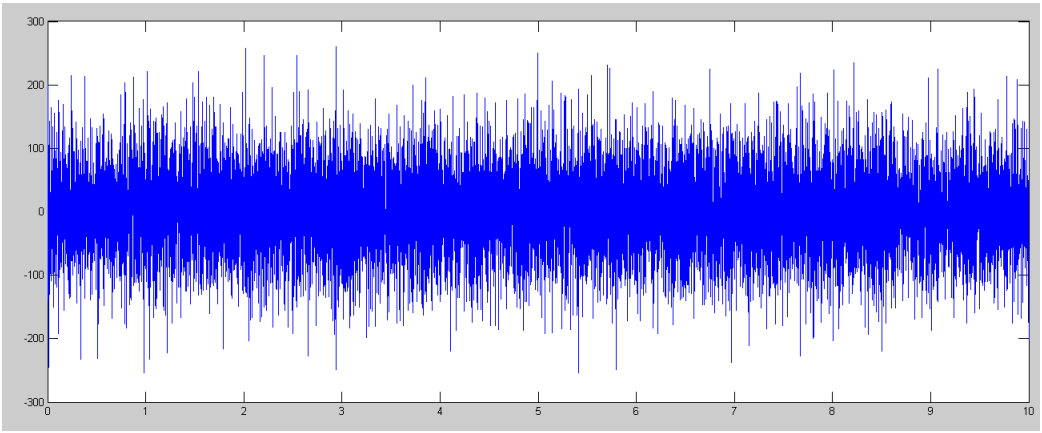
For example, if y(t) is a sine wave with gaussian white noise with a standard deviation of 0.1V, the signal looks like the following:

```
>> 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:

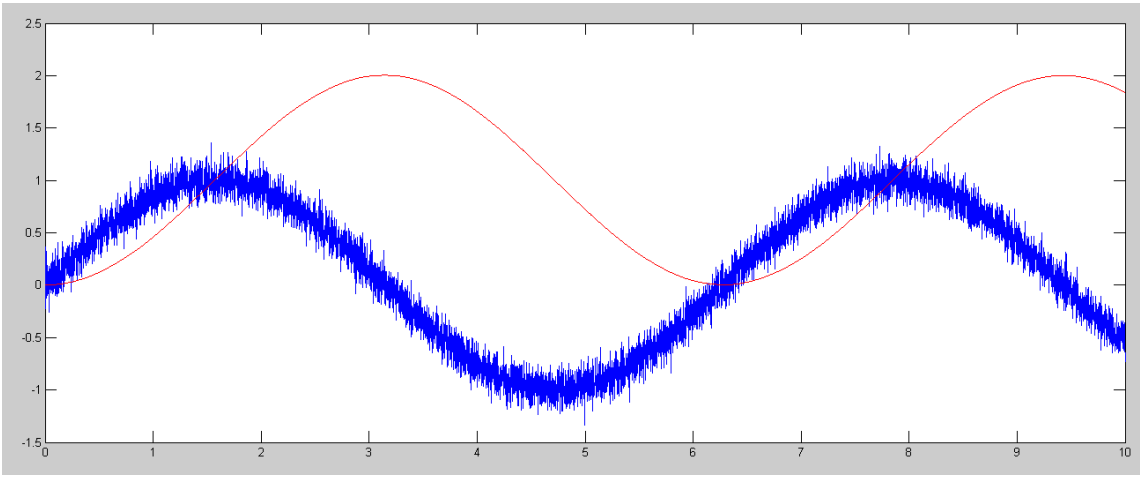
```
>> plot(t,derivative(t,y))
```



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.

Likewise, in practice, differentiation is avoided if at all possible. It's better to integrate than it is to differentiate.

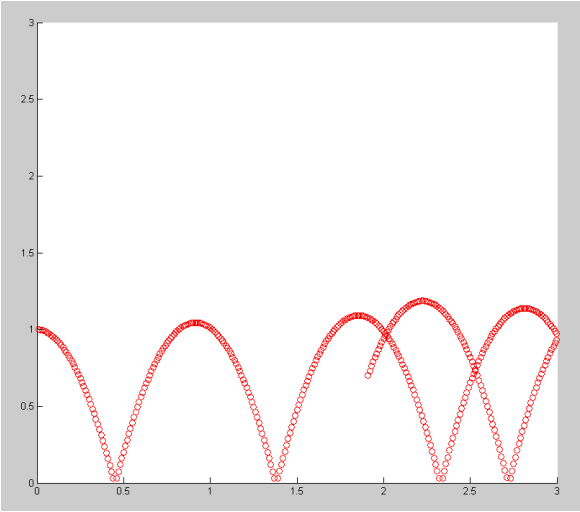
Fun with Integration: Bouncing Ball

With numerical integration, you can do all sorts of simulations and animations. For example, simulate a ball bouncing in a box. Assume

- Gravity is pulling down in the -y direction
- If the ball hits the floor (y<0), it bounces up (the velocity in the y direction becomes positive)
- If the ball hits the right wall (x>3), the ball bounces left (the x-velocity becomes negative)
- If the ball hits the left wall (x<0), the ball bounces right (the y-velocity becomes positive)

A Matlab script to do this is as follows

```
% 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
```



Fun with Integration: Shoot Game

Another example is to simulate the launch of a tennis ball to hit a target. 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 similar to the bouncing ball to calculate the velocity and position of the tennis ball at each time.

When the tennis ball hits the ground (y=0), return how far away you were from the target.

A Matlab function do to this:

```
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
```

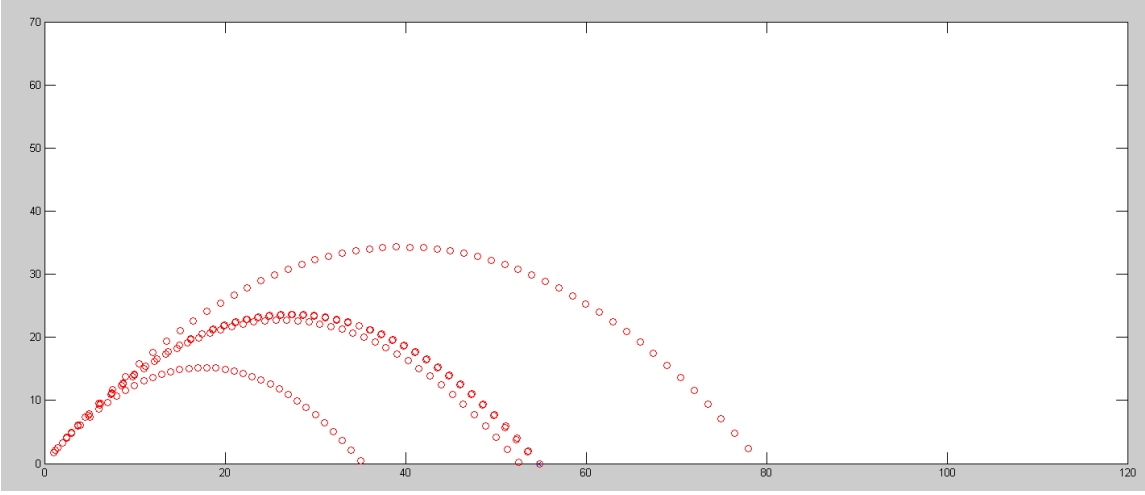
The tennis ball hit 10.3829 meters short of the target

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
```



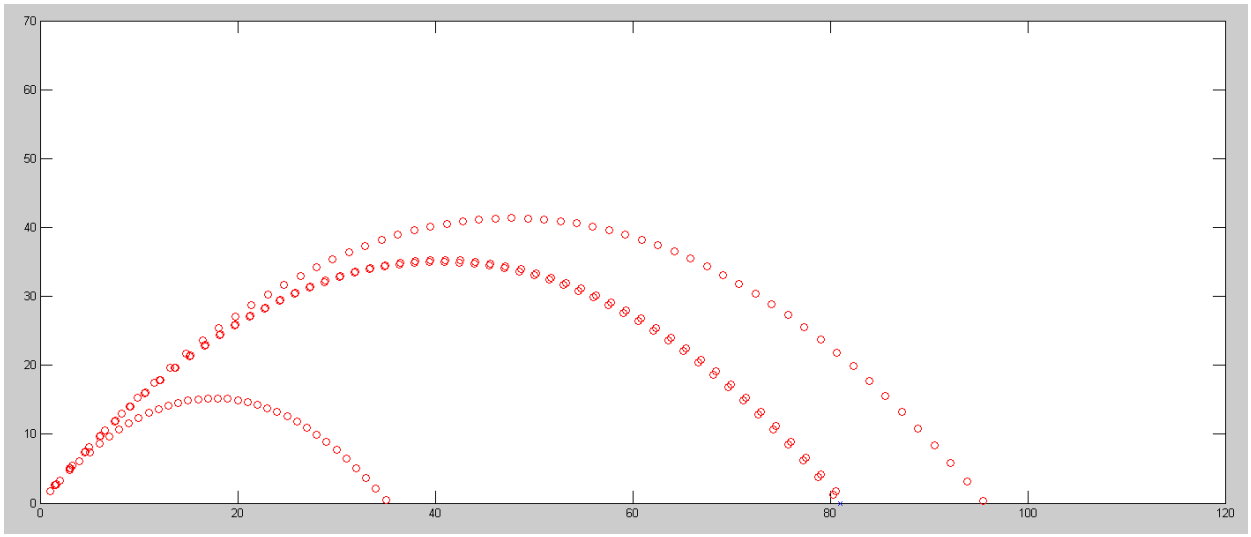
California Method

You can also use 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
```



Tennis Ball Trajectory using Newton's Method

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.