

## Exploring Derivatives with *Mathematica*

The techniques introduced in this notebook will help learn how to use *Mathematica* to explore the concepts introduced in the first four chapters of the text.

### ■ Estimating Slopes Numerically and Graphically

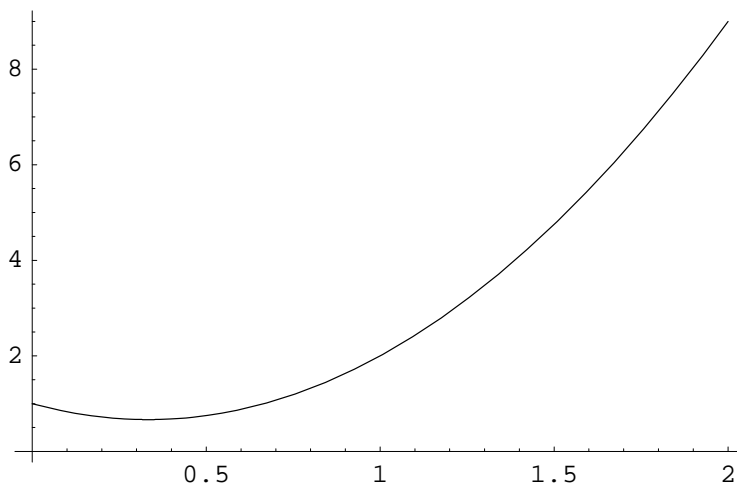
Suppose, for example, you want to use graphical and numerical methods to investigate the rate of change of  $y$  with respect to  $x$  for  $y = 3x^2 - 2x + 1$  at  $x = 1$ . A good starting point is to define a function  $f$  such that  $f(x) = 3x^2 - 2x + 1$ .

```
In[1]:= Clear[f];  
f[x_] = 3 x^2 - 2 x + 1
```

```
Out[2]= 1 - 2 x + 3 x^2
```

In order to estimate the rate of change graphically, the `Plot` command is used to obtain a graph of the function near  $x = 1$ .

```
In[3]:= Plot[f[x], {x, 0, 2}]
```

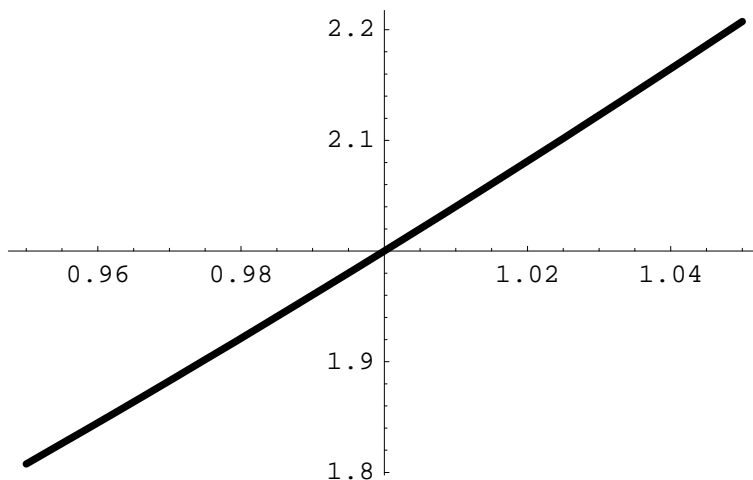


```
Out[3]= - Graphics -
```

The values of  $x$  in the following `Plot` command have been modified in order to zoom in on the graph near  $x = 1$ . The option, `PlotStyle -> Thickness[.01]`, has been added to increase the thickness of the graph of  $y = f(x)$ .

The corresponding graph in the output cell below looks almost like a line. (Note that the horizontal and vertical lines appearing in the graph are *not* the  $x$ -axis and  $y$ -axis. Rather, they are lines *parallel* to the  $x$ -axis and  $y$ -axis, respectively, intersecting at  $(1, 2)$ .)

```
In[4]:= Plot[f[x], {x, .95, 1.05}, PlotStyle -> Thickness[.01]]
```



```
Out[4]= - Graphics -
```

The graph above appears to pass through the point (1, 2). To find the approximate coordinates of another point on the curve, click on the graph above and a box should appear around the selected graph. Then, while holding down the **Ctrl** key, move your mouse so that the cursor is on top of the upper right-hand corner of the graph. When you do this, you will see the list {1.05, 2.21} (or something very close to this) in bottom left-hand corner of the notebook window. This represents the point (1.05, 2.21).

Therefore the rate of change of  $y$  with respect to  $x$  can be estimated by calculating the slope of the line passing through (1, 2) and (1.05, 2.21).

```
In[5]:= 
$$\frac{2.21 - 2}{1.05 - 1}$$

```

```
Out[5]= 4.2
```

The slope estimate above assumes that  $f(1.05) \approx 2.21$ . A better estimate of the slope is obtained by replacing 2.21 with  $f(1.05)$ .

```
In[6]:= 
$$\frac{f[1.05] - f[1]}{1.05 - 1}$$

```

```
Out[6]= 4.15
```

Using Equation (3) in Section 1.4, the rate of change can be estimated by examining the value of  $\frac{f(1+h)-f(1)}{h}$  as  $h$  gets close to 0. In the following input cell, the `Table` command is used to output the value of the list  $\{10^{-k}, \frac{f(1+10^{-k})-f(1)}{10^{-k}}\}$  for  $k = 1, 2, \dots, 5$ . The value of  $h$  is represented by  $10^{-k}$  since  $10^{-k}$  gets closer to 0 as  $k$  increases. Adding `// TableForm` to the end of the command will result in the list being displayed in the form of a table with the values of  $10^{-k}$  displayed in the first column and the corresponding values of  $\frac{f(1+10^{-k})-f(1)}{10^{-k}}$  displayed in the second column.

```
In[7]:= Table[{10.^-k,  $\frac{f[1 + 10.^{-k}] - f[1]}{10.^{-k}}$ }, {k, 1, 5}] // TableForm
```

```
Out[7]//TableForm=
```

```
0.1          4.3
0.01         4.03
0.001        4.003
0.0001       4.0003
0.00001      4.00003
```

It appears that  $\frac{f(1+h)-f(1)}{h} \rightarrow 4$  as  $h \rightarrow 0$ . Another way to state this is to write  $\lim_{h \rightarrow 0} \frac{f(1+h)-f(1)}{h} = 4$ .

The *Mathematica* command for computing  $\lim_{x \rightarrow x_0} g(x)$  is `Limit[g[x], x -> x0]`. To verify that  $\lim_{h \rightarrow 0} \frac{f(1+h)-f(1)}{h} = 4$ , *Mathematica* is now instructed to compute the limit.

```
In[8]:= Limit[ $\frac{f[1 + h] - f[1]}{h}$ , h -> 0]
```

```
Out[8]= 4
```

## ■ More Examples of Computing Limits

The `Limit` command is used to compute  $\lim_{x \rightarrow 0} \frac{\sin(x)}{x}$  in the following input.


```
In[9]:= Limit[ $\frac{\text{Sin}[x]}{x}$ , x -> 0]
```

```
Out[9]= 1
```

The reserved *Mathematica* word for  $\infty$  is `Infinity`. Next, the value of  $\lim_{x \rightarrow \infty} (1 + \frac{1}{2x})^x$  is computed.

```
In[10]:= Limit[( $1 + \frac{1}{2x}$ )^x, x -> Infinity]
```

```
Out[10]=  $\sqrt{e}$ 
```

The reserved word `Infinity` can be replaced with the  $\infty$  symbol by clicking on  in the palette *BasicInput* (the palette *BasicInput* can be opened from the Help menu by selecting **File**, then **Palettes**, and then **BasicInput**). The values of  $\lim_{x \rightarrow \infty} \frac{|x|}{x+1}$  and  $\lim_{x \rightarrow -\infty} \frac{|x|}{x+1}$  are computed next. (The *Mathematica* function `Abs[x]` represents the absolute value of  $x$ .)

```
In[11]:= Limit[ $\frac{\text{Abs}[x]}{x + 1}$ , x ->  $\infty$ ]
```

```
Limit[ $\frac{\text{Abs}[x]}{x + 1}$ , x ->  $-\infty$ ]
```

```
Out[11]= 1
```

```
Out[12]= -1
```

The command `Limit[g[x], x -> a, Direction -> 1]` is used to compute the left-hand limit  $\lim_{x \rightarrow a^-} f(x)$  and `Limit[g[x], x -> a, Direction -> -1]` represents the right-hand limit  $\lim_{x \rightarrow a^+} f(x)$ .

```
In[13]:= Limit[1/x, x -> 0, Direction -> 1]
          Limit[1/x, x -> 0, Direction -> -1]
```

```
Out[13]= -∞
```

```
Out[14]= ∞
```

*Mathematica* can compute some symbolic limits also. For example, using the definition of  $f'(x)$  found in equation (3) in Section 1.7 of the text, *Mathematica* can be used to find the derivative of  $\cos(x)$ .

```
In[15]:= Limit[Cos[x+h] - Cos[x], h -> 0]
```

```
Out[15]= -Sin[x]
```

## ■ Computing Derivatives using the Derivative Definition

Using the definition of a derivative,  $f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$ , you can compute the derivative of a function. The derivative is assigned the function name `fprime[x]` in the following input.

```
In[16]:= Clear[f, fprime];
          f[x_] = 4 x^3 - 8 x + 6
          fprime[x_] = Limit[(f[x+h] - f[x])/h, h -> 0]
```

```
Out[17]= 6 - 8 x + 4 x^3
```

```
Out[18]= 4 (-2 + 3 x^2)
```

The equation of the line tangent to  $y = f(x)$  at the point  $(a, f(a))$  is  $y = f(a) + f'(a)(x - a)$ . So if, for example, you want to find the equation of the line tangent to the curve at  $(-1, f(-1))$ , you can compute the line with *Mathematica* (here the function name `tanline[x]` is used to represent the tangent line function).

```
In[19]:= tanline[x_] = f[-1] + fprime[-1] (x + 1)
```

```
Out[19]= 10 + 4 (1 + x)
```

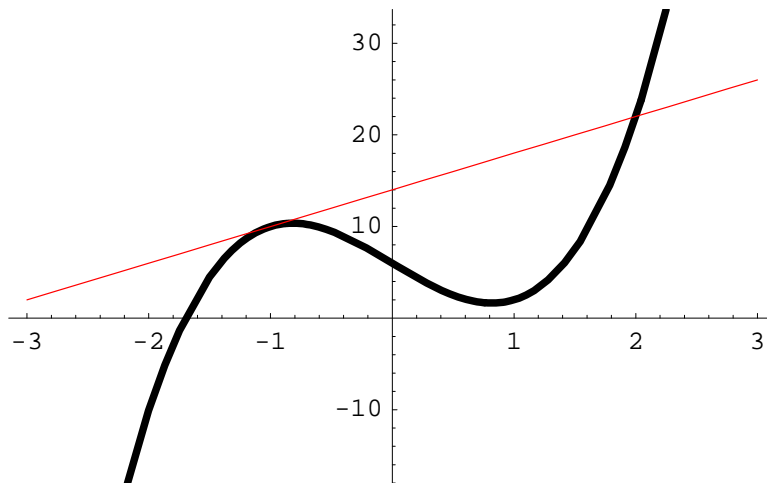
The *Mathematica* command `Simplify[%]` simplifies the last result. The symbol `%` refers to the last output (`%19` could also be used in place of `%`).

```
In[20]:= Simplify[%]
```

```
Out[20]= 14 + 4 x
```

Now the function and the tangent line are graphed together. The option `PlotStyle -> {Thickness[.01], RGBColor[1, 0, 0]}` applies the option `Thickness[.01]` to  $y = f(x)$  (increasing the thickness of the curve) and the option `RGBColor[1, 0, 0]` instructs *Mathematica* to plot the tangent line in color red. Other colors can be used also - for example, `RGBColor[0, 1, 0]` corresponds to green and `RGBColor[0, 0, 1]` corresponds to blue. Consult the *Mathematica* Help menu for more information about `RGBColor`.

```
In[21]:= Plot[{f[x], tanline[x]}, {x, -3, 3},
  PlotStyle -> {Thickness[.01], RGBColor[1, 0, 0]}
```



```
Out[21]= - Graphics -
```

## ■ Computing Derivatives

The following function will be used to illustrate how to compute derivatives with *Mathematica*.

```
In[22]:= Clear[f];
  f[x_] = 5 x2 - 7 x + 10;
```


From the keyboard, type in  $D[f[x], x]$  or  $f'[x]$  to compute the derivative of  $f$  with respect to  $x$ .

```
In[24]:= D[f[x], x]
```

```
Out[24]= -7 + 10 x
```

```
In[25]:= f'[x]
```

```
Out[25]= -7 + 10 x
```

To use the palette *BasicInput* to compute the derivative with respect to  $x$ , click on  and type in  $f[x]$ , then press the **Tab** key and then type in  $x$ .

```
In[26]:= ∂x f[x]
```

```
Out[26]= -7 + 10 x
```

*Mathematica* can also be used to verify common derivative laws, such as the product rule, quotient rule and chain rule.

```
In[27]:= Clear[f, g];
  D[f[x] g[x], x]
```

```
Out[28]= g[x] f'[x] + f[x] g'[x]
```

```
In[29]:= D[ $\frac{f[x]}{g[x]}$ , x] // Simplify
```

```
Out[29]=  $\frac{g[x] f'[x] - f[x] g'[x]}{g[x]^2}$ 
```

```
In[30]:= D[f[g[x]], x]
```

```
Out[30]= f'[g[x]] g'[x]
```

The equation of the line tangent to  $y = f(x)$  at the point  $(a, f(a))$  is  $y = f(a) + f'(a)(x - a)$ .

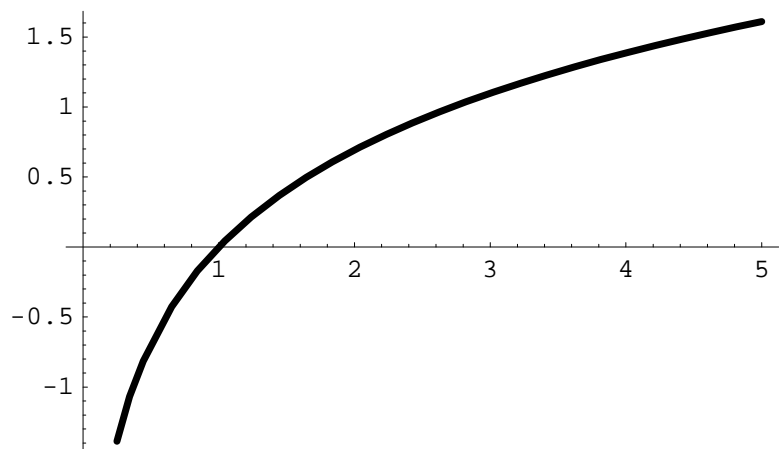
For example, if you want to find the equation of the line tangent to the curve at  $(2, f(2))$ , you can compute the line with *Mathematica* (here the function name `tanline[x]` is used).

```
In[31]:= Clear[f, tanline];
f[x_] = Log[x];
tanline[x_] = f[2] + f'[2] (x - 2) // Simplify
```

```
Out[33]=  $-1 + \frac{x}{2} + \text{Log}[2]$ 
```

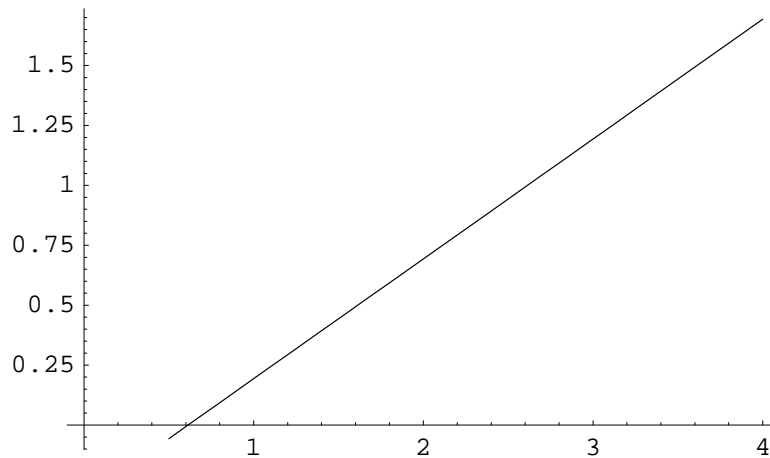
Here is a plot of the curve  $y = f(x)$ . Note that the plot is assigned the name *curve*.

```
In[34]:= curve = Plot[{f[x]}, {x, .25, 5}, PlotStyle -> Thickness[.01]];
```



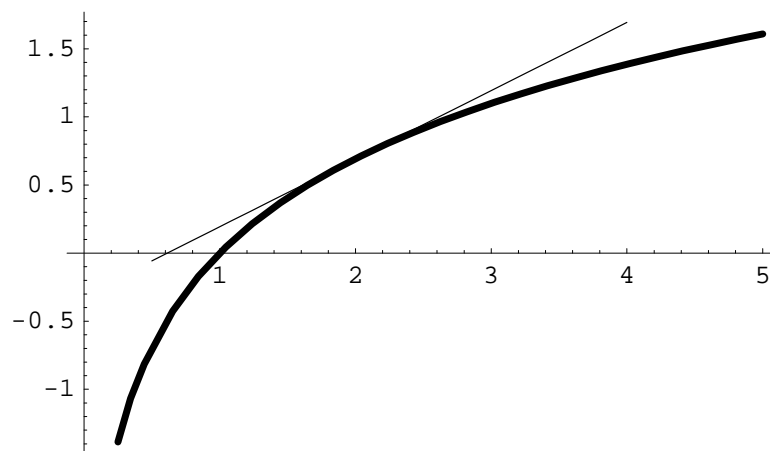
Now a plot of the tangent line is given and it is given the name *tangraph*.

```
In[35]:= tangraph = Plot[tanline[x], {x, .5, 4}];
```



To display both graphs simultaneously, use the Show command.

```
In[36]:= Show[{curve, tangraph}]
```



```
Out[36]= - Graphics -
```

Suppose you want to display the curve and the line simultaneously, but not individually. The Plot option `DisplayFunction -> Identity` suppresses the graph from being displayed.

```
In[37]:= Clear[curve, tangraph];
curve = Plot[{f[x]}, {x, .25, 5},
  PlotStyle -> Thickness[.01], DisplayFunction -> Identity]
```

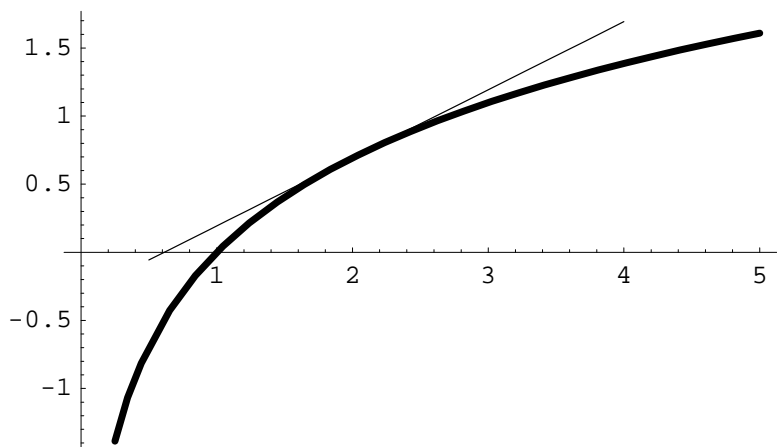
```
Out[38]= - Graphics -
```

```
In[39]:= tangraph = Plot[tanline[x], {x, .5, 4}, DisplayFunction -> Identity]
```

```
Out[39]= - Graphics -
```

Now the option `DisplayFunction -> $DisplayFunction` can be used together with the Show command to display the previously hidden graphs.

```
In[40]:= Show[{curve, tangraph}, DisplayFunction -> $DisplayFunction]
```



```
Out[40]= - Graphics -
```

## Higher-order Derivatives

Enter  $f''[x]$  to compute the second derivative,  $f'''[x]$  to compute the third derivative, and so forth.

```
In[41]:= f[x_] = Sin[x] / (x^2 - x)
```

```
Out[41]= Sin[x] / (-x + x^2)
```

```
In[42]:= f'''[x]
```

```
Out[42]= - 2 (-1 + 2 x) Cos[x] / (-x + x^2)^2 - Sin[x] / (-x + x^2) + ( 2 (-1 + 2 x)^2 / (-x + x^2)^3 - 2 / (-x + x^2)^2 ) Sin[x]
```

Adding `//Simplify` to the end of the derivative will simplify the derivative.

```
In[43]:= f'''[x] // Simplify
```

```
Out[43]= -2 x (1 - 3 x + 2 x^2) Cos[x] + (2 - 6 x + 5 x^2 + 2 x^3 - x^4) Sin[x] / (-1 + x)^3 x^3
```

Another way to compute the  $n$ th derivative of  $f$  is to enter  $D[f[x], \{x, n\}]$ . For example, the command in the following input cell instructs *Mathematica* to compute the third derivative.

```
In[44]:= D[f[x], {x, 3}] // Simplify
```

```
Out[44]= 1 / (-1 + x)^4 x^4 (-x (6 - 24 x + 35 x^2 - 15 x^3 - 3 x^4 + x^5) Cos[x] + 3 (2 - 8 x + 11 x^2 - 4 x^3 - 5 x^4 + 2 x^5) Sin[x])
```

## ■ Implicit Differentiation

You can use *Mathematica* to find do implicit differentiation. For example, suppose you want to find  $\frac{dy}{dx}$  where  $(\frac{4}{5}x^2 + y^2)^3 = 3x^2 - 10y^3$ . In the following input cell, the name `eq` is assigned to the equation. Note that `=` represents an assignment and `==` represents an equals sign in an equation.

$$\text{In[45]:= eq} = \left(\frac{4}{5}x^2 + y^2\right)^3 == 3x^2 - 10y^3$$

$$\text{Out[45]=} \left(\frac{4}{5}x^2 + y^2\right)^3 == 3x^2 - 10y^3$$

The command `Dt[eq, x]` will compute the derivative of the equation with respect to  $x$ . The notation `Dt[y, x]` appearing in the output represents  $\frac{dy}{dx}$ .

$$\text{In[46]:= impdreq} = \text{Dt}[eq, x]$$

$$\text{Out[46]=} 3 \left(\frac{4}{5}x^2 + y^2\right)^2 \left(\frac{8}{5}x + 2y \text{Dt}[y, x]\right) == 6x - 30y^2 \text{Dt}[y, x]$$

Now the `Solve` command is used to find  $\frac{dy}{dx}$ . The command `Solve[impdreq, Dt[y, x]]` instruct *Mathematica* to solve the equation `impdreq` for `Dt[y, x]`.

$$\text{In[47]:= dersol} = \text{Solve}[\text{impdreq}, \text{Dt}[y, x]]$$

$$\text{Out[47]=} \left\{ \left\{ \text{Dt}[y, x] \rightarrow -\frac{-6x + \frac{24}{5}x \left(\frac{4}{5}x^2 + y^2\right)^2}{30y^2 + 6y \left(\frac{4}{5}x^2 + y^2\right)^2} \right\} \right\}$$

The value of the derivative can be used to plot lines tangent to the equation. The name `slope` is assigned to the slope formula in the next input statement.

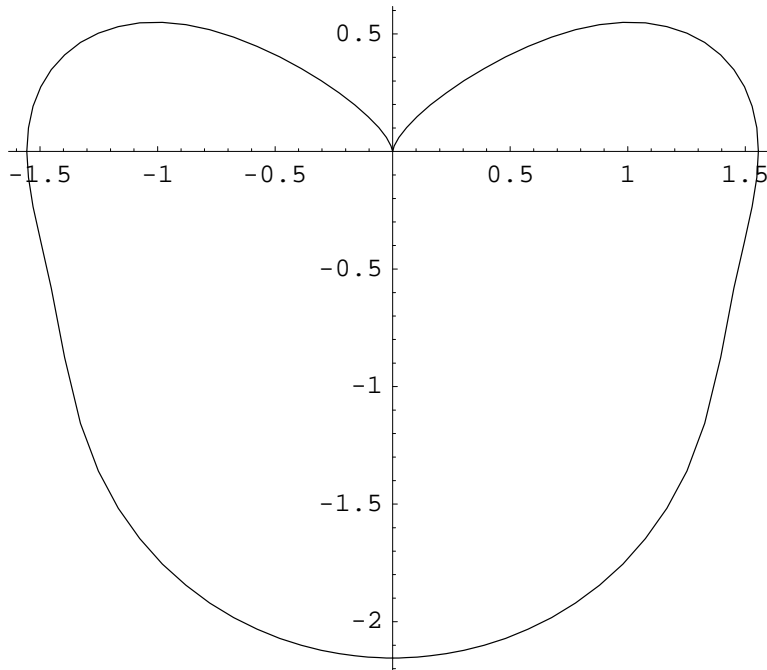
$$\text{In[48]:= slope} = \text{dersol}[[1, 1, 2]]$$

$$\text{Out[48]=} -\frac{-6x + \frac{24}{5}x \left(\frac{4}{5}x^2 + y^2\right)^2}{30y^2 + 6y \left(\frac{4}{5}x^2 + y^2\right)^2}$$

The Add-on command, `ImplicitPlot`, can be used to graph equations. First the package with the same name, found in the `Graphics` directory, must be loaded into memory.

$$\text{In[49]:=} \ll \text{Graphics`ImplicitPlot`}$$

```
In[50]:= impgr = ImplicitPlot[eq, {x, -2, 2}]
```



```
Out[50]= - Graphics -
```

Now suppose you want to find the equations of the lines tangent to the curve at  $x = 1$ .

The *Mathematica* command `NSolve[expr, y]` will find approximate solutions to an equation of one variable  $y$  (represented by `expr` here) for  $y$ . So what you want to do here is replace the value of  $x$  in `eq` with 1 and then solve the resulting equation for  $y$ . The operator `/.` is used to replace the value of a given variable. (For example, entering `eq /. x -> 1` instructs *Mathematica* to replace the value of  $x$  in the equation `eq` with 1.) Then the `NSolve` command is used to solve the resulting equation for  $y$ .

```
In[51]:= sols = NSolve[eq /. x -> 1, y]
```

```
Out[51]= {{y -> -1.73354}, {y -> -0.362773 - 0.572758 i}, {y -> -0.362773 + 0.572758 i},
          {y -> 0.549706}, {y -> 0.954688 - 2.18372 i}, {y -> 0.954688 + 2.18372 i}}
```

The output above reveals two approximate, real solutions for  $y$ :  $-1.73354$  and  $0.549706$ . (The other solutions are complex numbers and they can be ignored.)

Now the real solutions for  $y$  are assigned the names `y0` and `y1`.

```
In[52]:= y0 = sols[[1, 1, 2]]
```

```
y1 = sols[[4, 1, 2]]
```

```
x0 = 1
```

```
x1 = 1
```

```
Out[52]= -1.73354
```

```
Out[53]= 0.549706
```

```
Out[54]= 1
```

```
Out[55]= 1
```

Next, the replacement operator /. is used again to compute the slope of the tangent lines at  $(1, -1.73355)$  and  $(1, 0, 549706)$ .

```
In[56]:= Clear[x, y];
         m1 = slope /. {x -> x0, y -> y0}
```

```
Out[57]= 1.05052
```

```
In[58]:= m2 = slope /. {x -> x1, y -> y1}
```

```
Out[58]= 0.0129278
```

The tangents lines are computed in the next input cell.

```
In[59]:= tanline1 = y0 + m1 (x - x0) // Simplify
         tanline2 = y1 + m2 (x - x1) // Simplify
```

```
Out[59]= -2.78405 + 1.05052 x
```

```
Out[60]= 0.536778 + 0.0129278 x
```

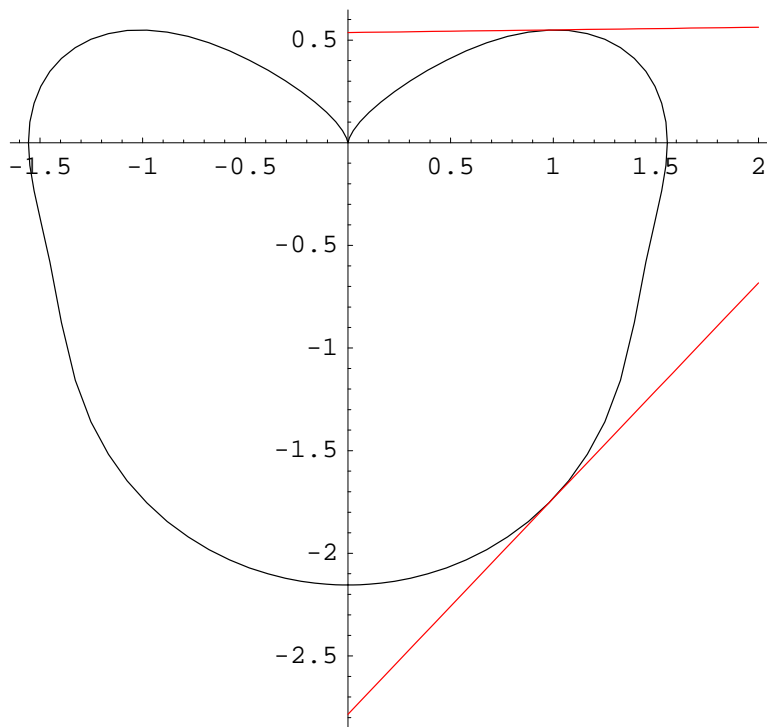
Now the graphs of the tangent lines are assigned the name tanlinegr.

```
In[61]:= tanlinegr = Plot[{tanline1, tanline2}, {x, 0, 2},
                        PlotStyle -> RGBColor[1, 0, 0], DisplayFunction -> Identity]
```

```
Out[61]= - Graphics -
```

Finally, the graph of the equation along with the tangent lines are displayed using the Show command.

```
In[62]:= Show[{impgr, tanlinegr}, DisplayFunction -> $DisplayFunction]
```

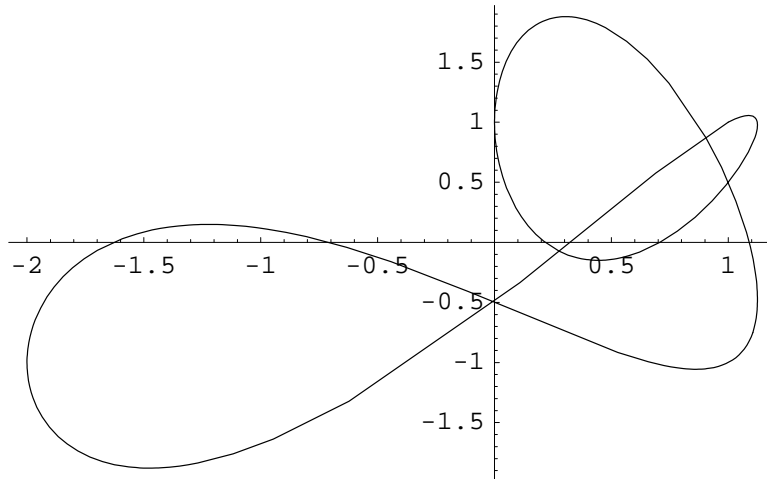


```
Out[62]= - Graphics -
```

## ■ Parametric Equations

The command `ParametricPlot[{f[t],g[t]},{t,a,b}]` is used to plot the graph of the curve defined parametrically by  $x = f(t)$ ,  $y = g(t)$ . For example, suppose we want to plot the graph represented by the parametric equations  $x = \sin(t) + \cos(2t)$  and  $y = \sin(t) + \cos(3t)$ . The following graph is given the name `gr1`.

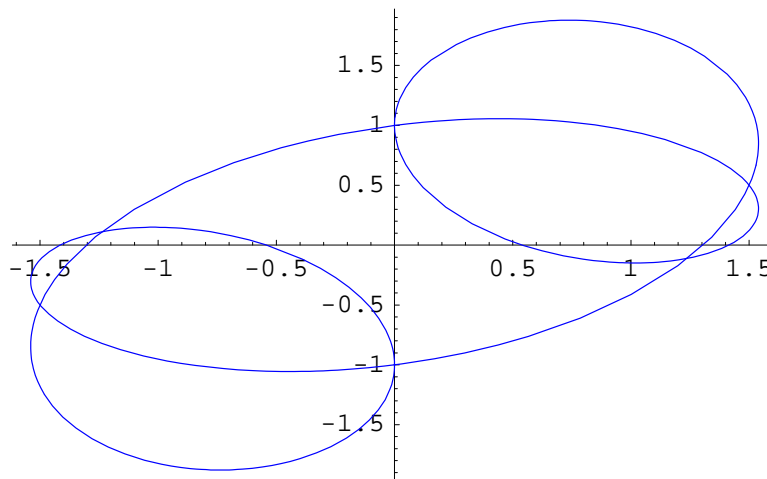
```
In[63]:= gr1 = ParametricPlot[{Sin[t] + Cos[2 t], Sin[t] + Cos[3 t]}, {t, 0, 2 π}]
```



```
Out[63]= - Graphics -
```

Here is another example and notice the additional plot option added so that the curve is sketched in the color blue.

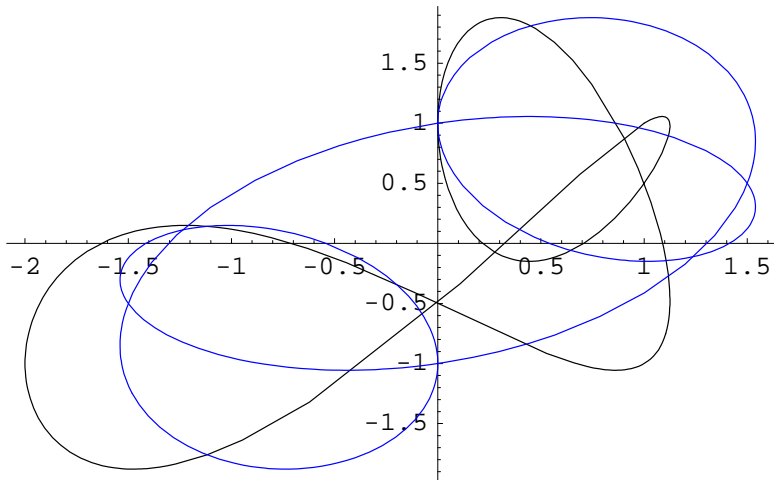
```
In[64]:= gr2 = ParametricPlot[{Sin[t] + Sin[3 t], Sin[t] + Cos[3 t]},  
{t, 0, 2 π}, PlotStyle -> RGBColor[0, 0, 1]]
```



```
Out[64]= - Graphics -
```

The `Show` command can then be used to plot both curves simultaneously.

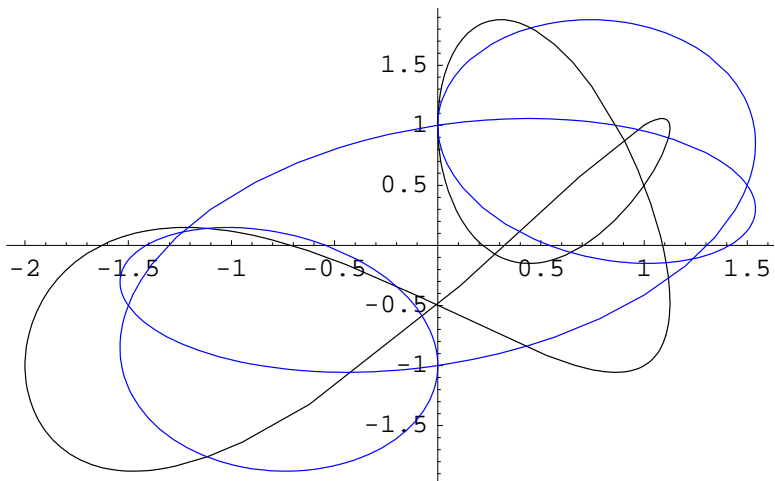
```
In[65]:= Show[gr1, gr2]
```



```
Out[65]= - Graphics -
```

Of course, both graphs can be plotted simultaneously with a single `ParametricPlot` command. Study the following input statement.

```
In[66]:= ParametricPlot[
  {{Sin[t] + Cos[2 t], Sin[t] + Cos[3 t]}, {Sin[t] + Sin[3 t], Sin[t] + Cos[3 t]}},
  {t, 0, 2 π}, PlotStyle -> {RGBColor[0, 0, 0], RGBColor[0, 0, 1]}]
```



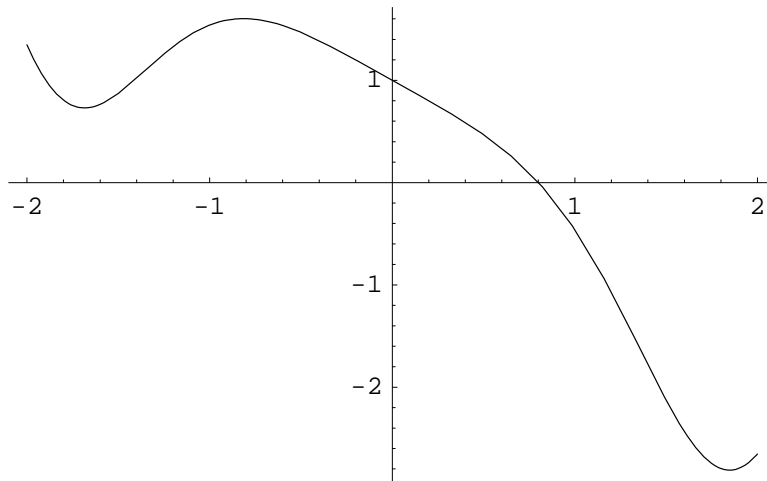
```
Out[66]= - Graphics -
```

## ■ Newton's Method

Suppose you want to use Newton's method, introduced in Section 4.2, to find the zero of  $f(x) = \cos(x^2) - x$  using the first iterate  $x_1 = 1.5$ .

You could begin by defining and plotting the function to see the location of the zero.

```
In[67]:= Clear[f];
f[x_] = Cos[x^2] - x;
Plot[f[x], {x, -2, 2}]
```



```
Out[69]= - Graphics -
```

From the graph above, it appears that the zero is approximately 0.8.

Using Newton's Method, substitute the value of  $x_1 = 1.5$  into the equation  $x - \frac{f(x)}{f'(x)}$ .

```
In[70]:= x - f[x] / f'[x] /. x -> 1.5
```

```
Out[70]= 0.861718
```

So  $x_2 = 0.861718$ . You can cut and paste the above input cell and place it below, then replace 1.5 with % (recall the % is the value of the last output) to obtain  $x_3$ .

```
In[71]:= x - f[x] / f'[x] /. x -> %
```

```
Out[71]= 0.804001
```

Continuing on in this fashion, you will find that the zero is approximately 0.801071.

```
In[72]:= x - f[x] / f'[x] /. x -> %
```

```
Out[72]= 0.801078
```

```
In[73]:= x - f[x] / f'[x] /. x -> %
```

```
Out[73]= 0.801071
```

```
In[74]:= x - f[x] / f'[x] /. x -> %
```

```
Out[74]= 0.801071
```

Another approach is to represent the  $x_i$  by the user-defined function  $x[i]$ . In the following input statement, the function is defined and the first iterate  $x[1]$  is defined to be 1.5.

```
In[75]:= x[i_] := x[i - 1] -  $\frac{f[x[i - 1]]}{f'[x[i - 1]]}$ ;
x[1] = 1.5;
```

Now you can compute the values of  $x[2]$ ,  $x[3]$  and so forth.

```
In[77]:= x[2]
Out[77]= 0.861718

In[78]:= x[3]
Out[78]= 0.804001

In[79]:= x[4]
Out[79]= 0.801078

In[80]:= x[5]
Out[80]= 0.801071
```

A quicker way to compute these values is to use a `Table` command.

```
In[81]:= Table[{i, x[i]}, {i, 1, 10}] // TableForm
Out[81]//TableForm=


|    |          |
|----|----------|
| 1  | 1.5      |
| 2  | 0.861718 |
| 3  | 0.804001 |
| 4  | 0.801078 |
| 5  | 0.801071 |
| 6  | 0.801071 |
| 7  | 0.801071 |
| 8  | 0.801071 |
| 9  | 0.801071 |
| 10 | 0.801071 |


```

When the above `Table` command is executed, you might notice that *Mathematica* is a little bit slow at displaying the results. This is because of the way in which *Mathematica* computes the values of  $x[i]$  internally. When for example, *Mathematica* computes  $x[3]$ , it first computes  $x[1]$  and  $x[2]$  again and again, when *Mathematica* computes  $x[4]$ , it first computes  $x[1]$ ,  $x[2]$  and  $x[3]$ , even though *Mathematica* has computed these values previously. So *Mathematica* does not "remember" the values of  $x[1]$  through  $x[n-1]$ , when computing  $x[n]$ . By replacing  $x[i_] := x[i - 1] - \frac{f[x[i-1]]}{f'[x[i-1]]}$  with  $x[i_] := x[i] = x[i - 1] - \frac{f[x[i-1]]}{f'[x[i-1]]}$ , *Mathematica* will "remember" the previously computed values of  $x[1]$  through  $x[n-1]$  when computing  $x[n]$ . Therefore, *Mathematica* will compute the list of iterates much more quickly.

```
In[82]:= Clear[x];
x[i_] := x[i] = x[i - 1] -  $\frac{f[x[i - 1]]}{f'[x[i - 1]]}$ ;
x[1] = 1.5;
Table[{i, x[i]}, {i, 1, 10}] // TableForm
```

```
Out[85]//TableForm=
1      1.5
2      0.861718
3      0.804001
4      0.801078
5      0.801071
6      0.801071
7      0.801071
8      0.801071
9      0.801071
10     0.801071
```

## ■ Euler's Method

Suppose you wish to use Euler's method to obtain an approximate solution to the initial value problem  $\frac{dy}{dt} = G(t, y) = t + y$ ,  $y(0) = 1$  on the interval  $[0, 2]$  with  $\Delta t = 0.1$ .

To do this with *Mathematica*, let  $g[t, y]$  be defined to equal the value of  $G(t, y)$  in the following input cell and using the palette *BasicInput*, click on  $\Delta$  to obtain the greek letter delta and set  $\Delta t = 1$ . The values of  $t_k$  and  $y_k$  will be represented by the functions  $t[k]$  and  $y[k]$ , respectively. Since  $t_0 = 0$ , set  $t[0] = 1$  and since  $y_0 = 1$ , let  $y[0] = 1$ . Finally define  $t[k_] := t[k - 1] + \Delta t$  since  $t_{k+1} = t_k + \Delta t$  and let  $y[k_] := y[k - 1] + \Delta t g[t[k - 1], y[k - 1]]$  since  $y_{k+1} = y_k + \Delta t G(t_k, y_k)$ .

```
In[86]:= Clear[g, Δt, t, y];
g[t_, y_] = t + y;
Δt = 0.1;
y[0] = 1;
t[0] = 0;
t[k_] := t[k - 1] + Δt;
y[k_] := y[k - 1] + Δt g[t[k - 1], y[k - 1]];
```

Now the Table command is used to show the values of  $(t_k, y_k)$  for  $k = 0, 1, 2, \dots, 15$ .

```
In[93]:= Table[{t[k], y[k]}, {k, 0, 15}] // TableForm
```

```
Out[93]//TableForm=
```

0	1
0.1	1.1
0.2	1.22
0.3	1.362
0.4	1.5282
0.5	1.72102
0.6	1.94312
0.7	2.19743
0.8	2.48718
0.9	2.8159
1.	3.18748
1.1	3.60623
1.2	4.07686
1.3	4.60454
1.4	5.195
1.5	5.8545

Depending upon the speed of the computer you are using, *Mathematica* may be a little slow at computing the table above. To speed up the calculation of the values of  $t[k]$  and  $y[k]$ , replace  $t[k_] := t[k - 1] + \Delta t$  with  $t[k_] := t[k] = t[k - 1] + \Delta t$  and  $y[k_] := y[k - 1] + \Delta t g[t[k - 1], y[k - 1]]$  with  $y[k_] := y[k] = y[k - 1] + \Delta t g[t[k - 1], y[k - 1]]$ . (See the previous subsection for a discussion of why this speeds up the computing process.)

```
In[94]:= Clear[g, Δt, t, y];
g[t_, y_] = t + y;
Δt = 0.1;
y[0] = 1;
t[0] = 0;
t[k_] := t[k] = t[k - 1] + Δt;
y[k_] := y[k] = y[k - 1] + Δt g[t[k - 1], y[k - 1]];
```

The output below is computed quickly by *Mathematica*, showing the values of the approximate solution.

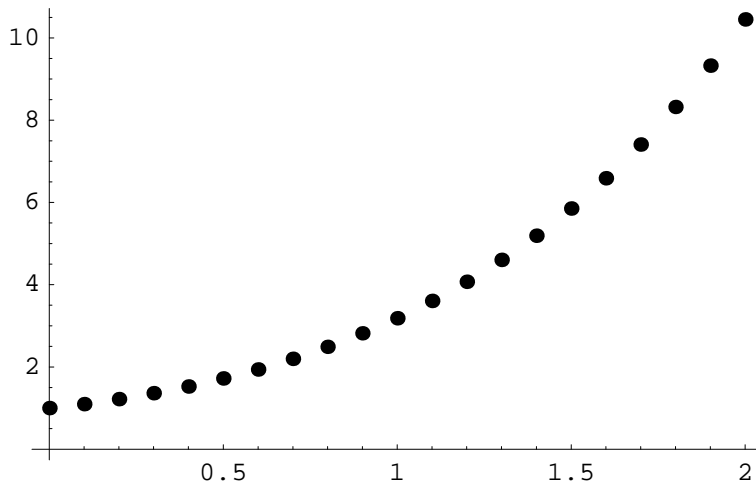
```
In[101]:= appsol = Table[{t[k], y[k]}, {k, 0, 20}];  
appsol // TableForm
```

```
Out[102]//TableForm=
```

0	1
0.1	1.1
0.2	1.22
0.3	1.362
0.4	1.5282
0.5	1.72102
0.6	1.94312
0.7	2.19743
0.8	2.48718
0.9	2.8159
1.	3.18748
1.1	3.60623
1.2	4.07686
1.3	4.60454
1.4	5.195
1.5	5.8545
1.6	6.58995
1.7	7.40894
1.8	8.31983
1.9	9.33182
2.	10.455

Now we plot the ordered pairs with the `ListPlot` command to see the graph of the approximate solution.

```
In[103]:= approxgr = ListPlot[appsol, PlotStyle -> PointSize[.02]]
```



```
Out[103]= - Graphics -
```