Introduction I

"Essential Mathematica for Students of Science",James J. Kelly , 2006

http://www.physics.umd.edu/courses/CourseWare

"*Mathematica 4.1* Notebooks -Complimentary software to accompany our textbook",John H. Mathews, and Russell W. Howell, 2002

• Introduction in Mathematica

```
ClearAll["Global`*"];
Off[General::spell, General::spell1]
```

• *Mathematica* Cells

The blue brackets on the right represent a *Mathematica* cell. You can double click on these blue brackets to expand and collapse cells. Try it, and you will find that cells which contain other cells can be expanded or collapsed. The little blue triangle tells you there is more information in the cell, so double click on it to expand it. Knowledge of this cell structure is not very important when you are first learning *Mathematica*, but you need to know it to open the cells below which contain information on using *Mathematica*.

```
CellPrint[Cell["Mathemaica Cells", "Text", FontWeight → "Plain",
  Background → RGBColor[1, 0, 1], CellMargins -> {{20, 0}, {20, 20}},
  CellTags → "T.14"]]
```

Mathemaica Cells

```
CellPrint[Cell["Mathemaica Cells", "text", CellDingbat -> "●",
   Background → Orange]]
```

● Mathemaica Cells

```
CellPrint[Cell["Mathemaica Cells", "Section", CellDingbat -> "●",
   Background → Yellow]]
```

● Mathemaica Cells

```
CellPrint[Cell["Mathemaica Cells", "Subsection", CellDingbat -> "●",
   Background → Yellow]]
```

● Mathemaica Cells

● Entering Input

When you start *Mathematica*, you are presented with a window into which you can enter commands. The commands are processed by the kernel, which is started when you first ask *Mathematica* to do something. You ask *Mathematica* to "do something" by pressing SHIFT-ENTER, or alternately the ENTER key on the keypad. Let's enter the number 8:

```
8
```

```
8
```

You can refer to previously entered data either by the input number (1 in the above) or using the % key (which refers to the last OUTPUT):

```
In[1]
```

```
%
```

You can use multiple % to refer to output before the previous output:

```
1
```

```
1
```

```
2
```

```
2
```

```
3
```

```
3
```

```
%%%
```

```
1
```

```
%%%
```

```
2
```

```
%%%
```

```
3
```

You can also refer to the `Out[ ]`:

```
Out[6]
```

Arithmetic

Mathematica can be used as simple calculator.

```
4 + 6
```

```
10
```

```
4 - 6
```

```
-2
```

```
4 * 6
```

```
24
```

In *Mathematica*, a space also denotes multiplication. This can lead to confusion, and you should always think about exactly what it is you are telling *Mathematica* to do when you encounter problems!

```
4 × 6
```

```
24
```

```
4 / 6
```

$$\frac{2}{3}$$

Notice that *Mathematica* leaves the result in the most accurate form if you use integers. If you use decimal numbers, *Mathematica* will produce a truncated decimal result.

```
4 / 6.
```

```
0.666667
```

Powers are defined with the `^` symbol.

```
4^6
```

```
4096
```

```
4096
```

```
4096
```

• Assignments

You can assign a value to a variable in two ways. The first immediately makes the definition:

```
a = 3.4
```

```
3.4
```

```
a
```

```
3.4
```

The second makes the definition only when the variable is later requested (that is, the definition is not immediately processed by the kernel):

```
b := 6.8
```

Notice there is no output from this! When we ask later for the value of **b**, the kernel processes the assignment and returns the result.

```
b
```

```
6.8
```

Once a variable is defined in the kernel, it is defined *even if you open another window!!* Try it. This is usually the problem when *Mathematica* behaves (seemingly) strangely. You can remove a definition in the following way:

```
Clear[a, b]
```

```
a
```

```
a
```

```
b
```

```
b
```

• Defining Functions

Here is how you define a function in *Mathematica*:

```
f[x_] = 4 * x / 3
```

$$\frac{4\,x}{3}$$

Notice the square brackets in the definition, and the underscore. Square brackets `[]` are used by *Mathematica* to define functions, round brackets `( )` as delimiters (ie, for multiplication), and curly brackets `{ }` for lists.

You can refer to it by typing `f[x]`:

```
f[x]
```

$$\frac{4x}{3}$$

You can make substitutions for the variable itself:

```
f[x-3]
```

$$\frac{4}{3}(-3+x)$$

```
f[3 (x / 2)^2]
```

$$x^2$$

```
f[t]
```

$$\frac{4t}{3}$$

```
a := 3
```

```
f[a]
```

$$4$$

```
f[1]
```

$$\frac{4}{3}$$

```
f[1.]
```

```
1.33333
```

Captization

Mathematica is very picky about the syntax you use, even to the point of capitalization.

```
f[x_] = (x - 3) / x
```

$$\frac{-3+x}{x}$$

```
f[y]
```

$$\frac{-3+y}{y}$$

Mathematica does not know what the following function is, so it simply returns the function as output:

```
F[y]
```

```
F[y]
```

Different Bracket Types

This has already been stated, but it is important enough to get its own section!

Square brackets `[]` are used by *Mathematica* to define functions, round brackets `( )` as delimiters (ie, for multiplication), and curly brackets `{ }` for lists.

Here is an example--notice that we have to be careful when doing division and powers to use round brackets to get the expression the way we want it. Notice the difference in the following two functions:

```
f[x_] = 5 * x / (x^2 + 4) + 6^(3 + x)
g[x_] = 5 * x / x^2 + 4 + 6^3 + x
```

$$6^{3+x} + \frac{5\,x}{4 + x^2}$$

$$220 + \frac{5}{x} + x$$

Here is a list of values, which is denoted using curly brackets:

```
{f[1], f[2], f[3], f[4]}
```

$$\left\{1297, \frac{31109}{4}, \frac{606543}{13}, 279937\right\}$$

• *Mathematica* Functions

Mathematica has a wealth of intrinsically defined functions, like Cos, Sin, Exp, etc. Generally, an intrinsically defined function begins with a capital letter. Since the function is already defined, you do not need to use an underscore if you use it.

```
Cos[x]
```

```
Cos[x]
```

```
Sin[2]
```

```
Sin[2]
```

```
Tan[3.]
```

```
-0.142547
```

```
Sqrt[100]
```

```
10
```

```
Sqrt[10]
```

$\sqrt{10}$

```
Sqrt[10.]
```

```
3.16228
```

```
ArcTan[3.]
```

```
1.24905
```

The factorial function is defined using the `!`, which is nice since that is how we write it by hand!

```
4!
```

```
24
```

There are many more.

```
Gamma[x]
```

```
Gamma[x]
```

The Exponential Function

One of the most common errors people new to *Mathematica* make is when they try to use exponential functions. Here is the correct way to input the exponential function $g(t) = e^t$:

```
g[t_] = Exp[t]
```

e^t

```
g[1.0]
```

```
2.71828
```

```
g[Pi]
```

e^π

The following will also work:

```
g[t_] = E^t
```

e^t

```
g[1.0]
```

```
2.71828
```

```
g[Pi]
```

e^π

What does not work is this:

```
g[t_] = e^t
```

e^t

```
g[1.0]
```

$e^{1.}$

```
g[Pi]
```

e^{π}

It looks very similar, but *Mathematica* treats the **e** as an undefined constant, not the special constant $e \sim 2.71828$.

The natural logarithm function, which we usually write as ln, is given by the command **Log** in *Mathematica*:

```
Log[Exp[-45]]
```

```
-45
```

If you want a logarithm with a base other than e, you use another parameter in the **Log** command:

```
Log[2, 2^3]
```

```
3
```

• Symbolic Calculations

The power of *Mathematica* is in its ability to do symbolic calculations. We saw that previously when *Mathematica* reduced 4/6 to 2/3. *Mathematica* knows the value of Pi:

```
Pi
```

π

Symbolic manipulations are important when working with functions:

```
f[t_] = Cos[2*t] (t-3)^2
```

$(-3+t)^2 \, \mathrm{Cos}[2\,t]$

```
g[t_] = t^2
```

t^2

We can use *Mathematica* to do compositions of functions:

```
f[g[t]]
```

$\left(-3+t^2\right)^2 \mathrm{Cos}\left[2\,t^2\right]$

```
g[f[t]]
```

$(-3+t)^4 \, \mathrm{Cos}[2\,t]^2$

```
f[f[t]]
```

$\left(-3+(-3+t)^2 \mathrm{Cos}[2\,t]\right)^2 \mathrm{Cos}\left[2\,(-3+t)^2 \mathrm{Cos}[2\,t]\right]$

• Converting Output to a Decimal

If we want a numerical estimate for a symbolic result, we use the command **N**:

```
N[Pi]
```

```
3.14159
```

Another command that will allow you to specify the number of decimals you want to keep is **SetPrecision:**

```
SetPrecision[Pi, 3]
```

```
3.14
```

```
SetPrecision[Pi, 298]
```

```
3.141592653589793238462643383279502884197169⋱
 3993751058209749445923078164062862089986280⋱
 3482534211706798214808651328230664709384460⋱
 9550582231725359408128481117450284102701938⋱
 5211055596446229489549303819644288109756659⋱
 3344612847564823378678316527120190914564856⋱
 6923460348610454326648213393607260249141
```

Algebra

Mathematica can help with algebra as well, using commands such as **Expand**, **Factor**, and **Simplify**.

```
Expand[(a - b)^2]
```

$9 - 6 b + b^2$

Whoops! We have forgotten that a has been defined to be 3 in the kernel. We need to clear it first if we wish to use it for something else:

```
Clear[a]
```

```
Expand[(a - b)^2]
```

$$a^2 - 2\,a\,b + b^2$$

```
Factor[%]
```

$$(a - b)^2$$

```
Expand[(a - b)^10]
```

$$a^{10} - 10\,a^9\,b + 45\,a^8\,b^2 - 120\,a^7\,b^3 + 210\,a^6\,b^4 - 252\,a^5\,b^5 + 210\,a^4\,b^6 - 120\,a^3\,b^7 + 45\,a^2\,b^8 - 10\,a\,b^9 + b^{10}$$

```
Factor[%]
```

$$(a - b)^{10}$$

```
Simplify[Expand[(a - b)^10]]
```

$$(a - b)^{10}$$

• Lists and Tables

Mathematica allows us to create lists and tables easily. Recall that before I said that curly brackets `{ }` are used by *Mathematica* to denote lists:

```
datalist1 = {1, 2, 3, 4, 5, 6, 7}
```

```
{1, 2, 3, 4, 5, 6, 7}
```

```
datalist2 = {{1, 2}, {2, 5}, {3, 16}, {4, 18}, {5, -2}}
```

```
{{1, 2}, {2, 5}, {3, 16}, {4, 18}, {5, -2}}
```

The above is a list of ordered pairs. You can think of the ordered pairs as data points, and we will see later how we can plot these points.

Some of the commands we will use in the future include **`Table`**, **`TableForm`**,

```
TableForm[datalist1]
```

```
1
2
3
4
5
6
7
```

```
TableForm[datalist2]
```

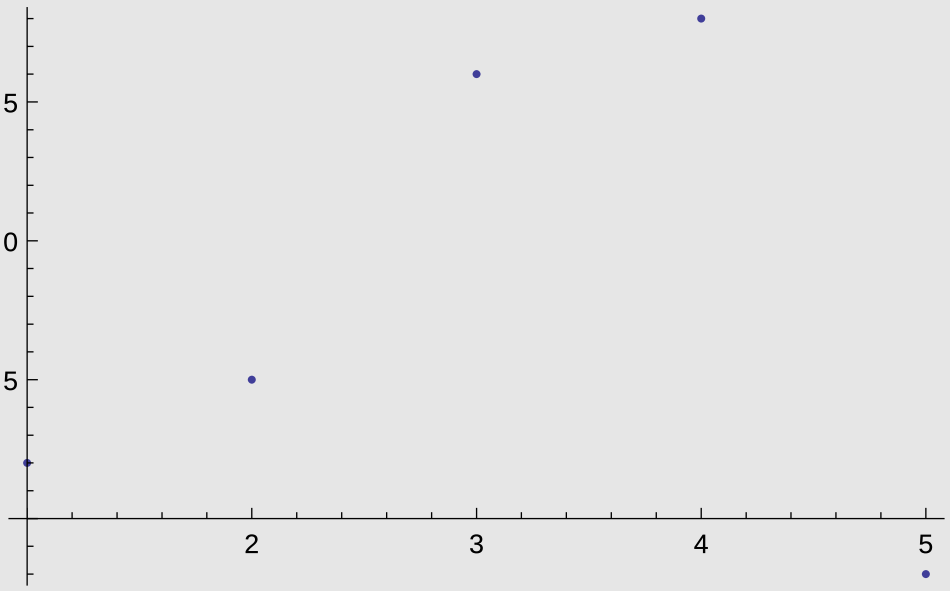
```
1  2
2  5
3  16
4  18
5  -2
```

Plotting

In *Mathematica* there are many ways to plot functions.

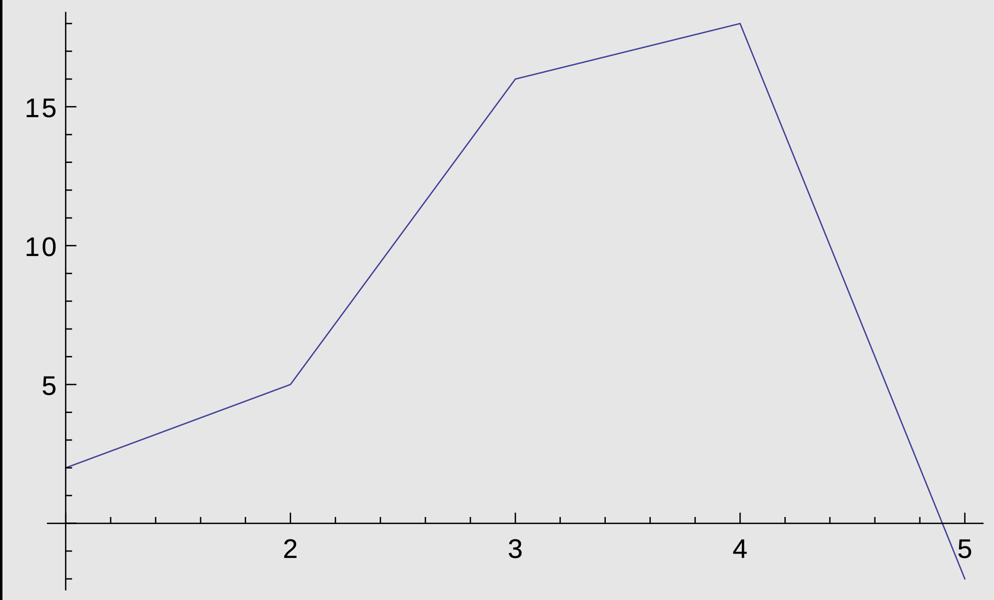
Let's say we want to plot the data we have in datalist2. We can create a scatter plot of the data in the following manner:

```
ListPlot[datalist2]
```



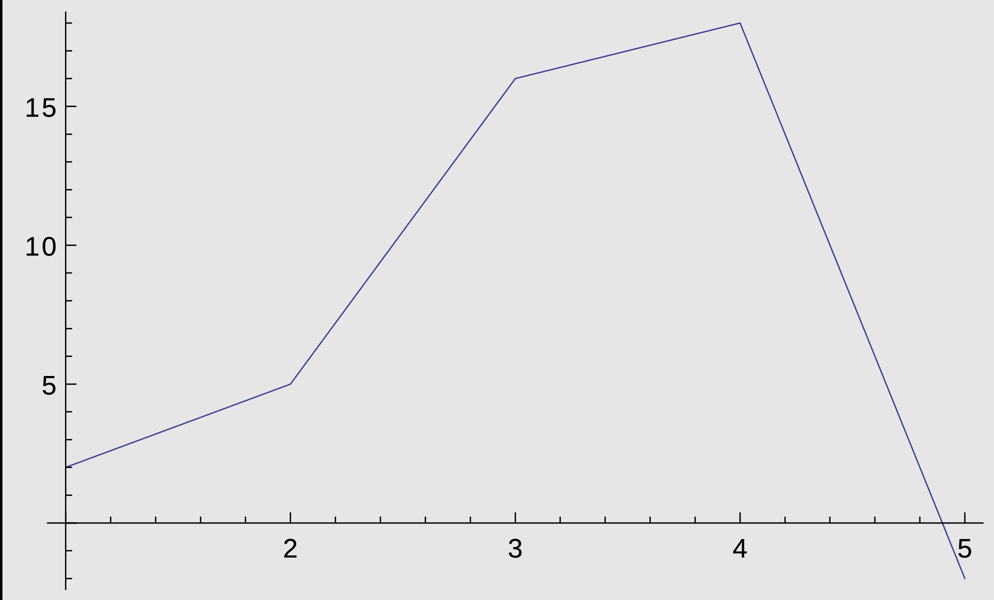
Sometimes it is hard to see the points, so we may want to join them

```
ListPlot[datalist2, Joined → True]
```



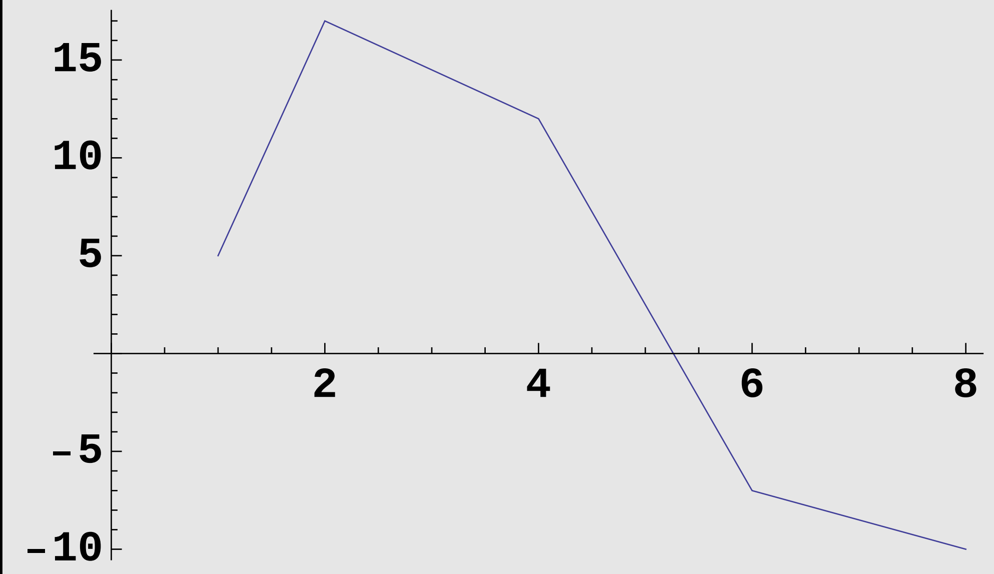
If we have a second data set we want to plot along with the first, we need to label the plots and use the **Show** command to plot them together.

```
plot1 = ListPlot[datalist2, Joined → True]
```

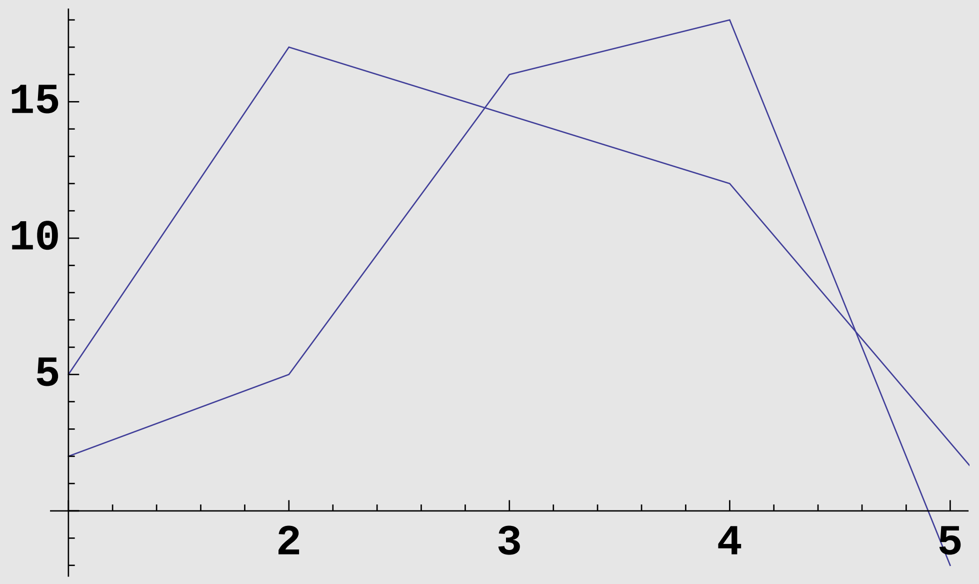


```
datalist3 := {{1, 5}, {2, 17}, {4, 12}, {6, -7}, {8, -10}}
```

```
plot2 = ListPlot[datalist3, Joined → True]
```

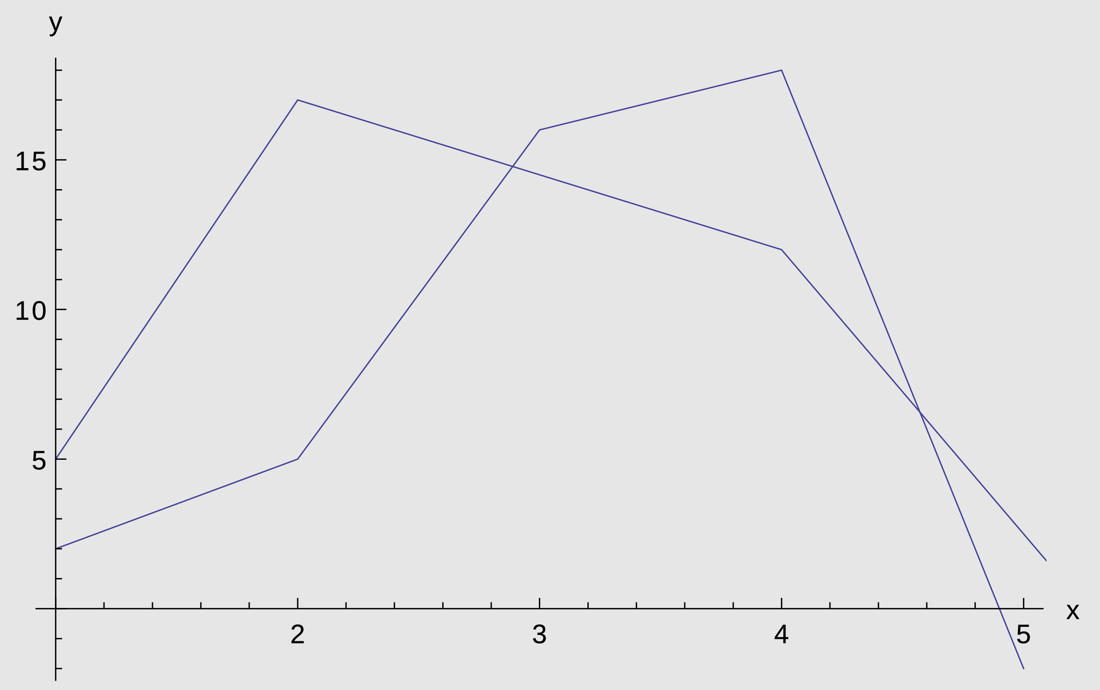


```
Show[plot1, plot2]
```



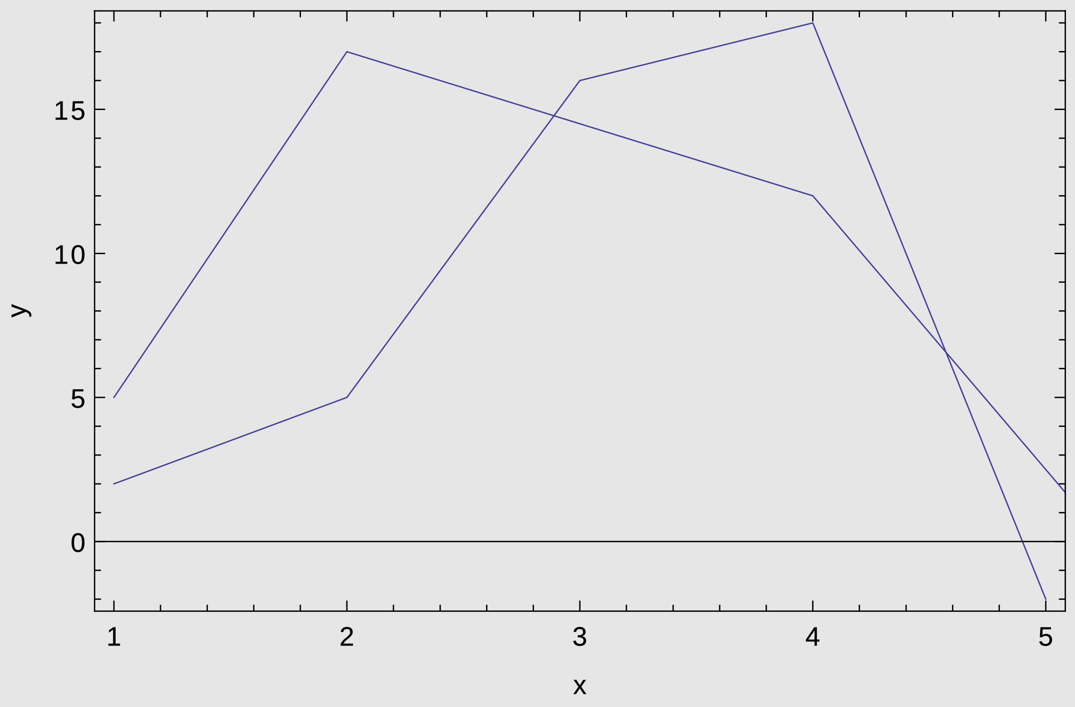
You can click on the image, then clcik and hold on one of the black squares at the image edge to resize the image. You could also add an option to specify the image's size when you plot it, and add labels to the axes:

```
Show[plot1, plot2, ImageSize → {400, Automatic}, AxesLabel → {"x", "y"}]
```



You can also add a frame around the image, and label the frame axes:

```
Show[plot1, plot2, ImageSize → {400, Automatic}, Frame → True,
 FrameLabel → {"x", "y"}]
```

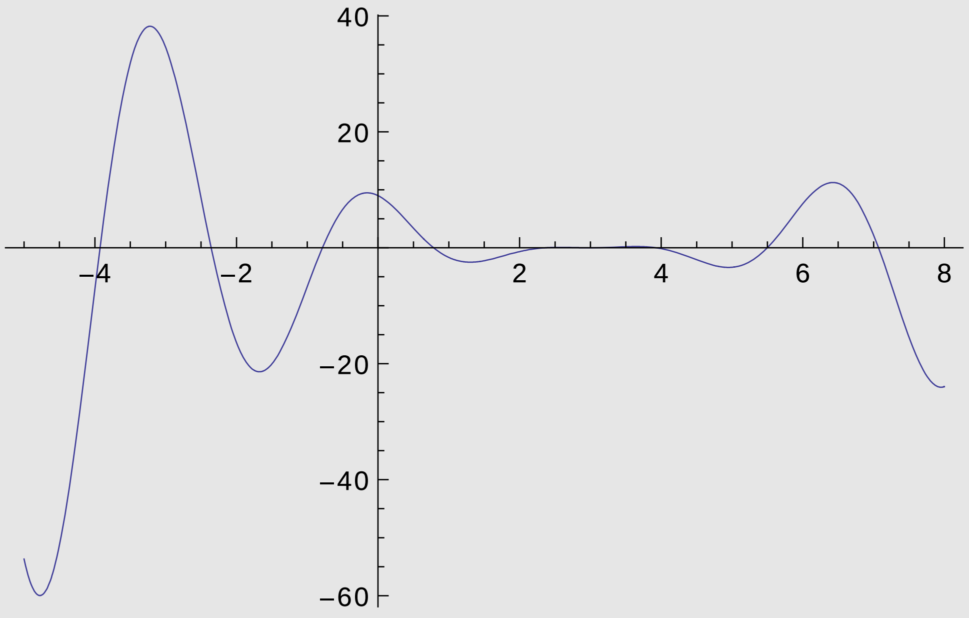


Notice the two lines are both solid. We can change the style of one of the lines (see below), but it does take a bit of work.

What if we want to plot functions, instead of points? Then the command is slightly different:

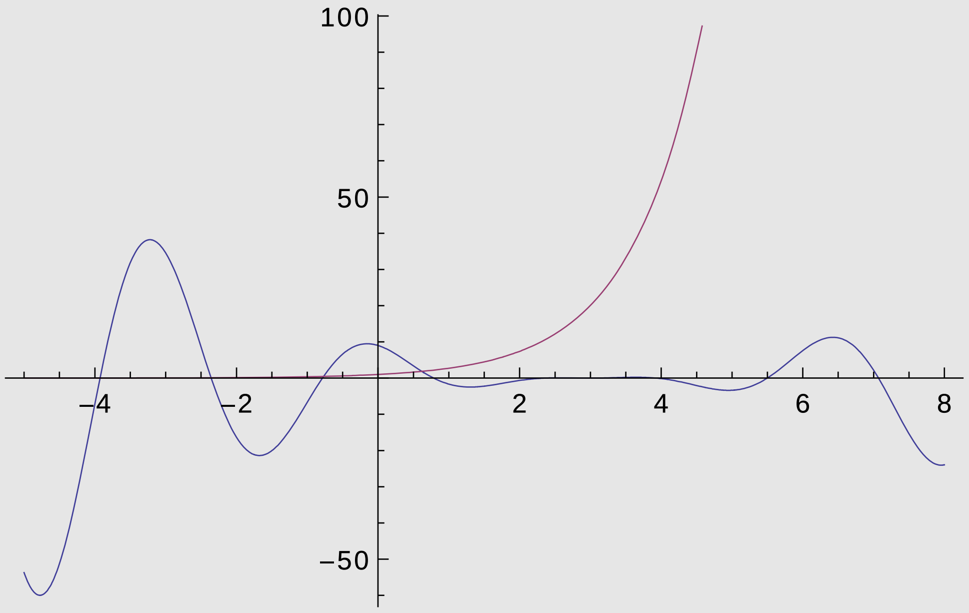
```
f[t_] := Cos[2 * t] (t - 3)^2
```

```
Plot[f[t], {t, -5, 8}]
```



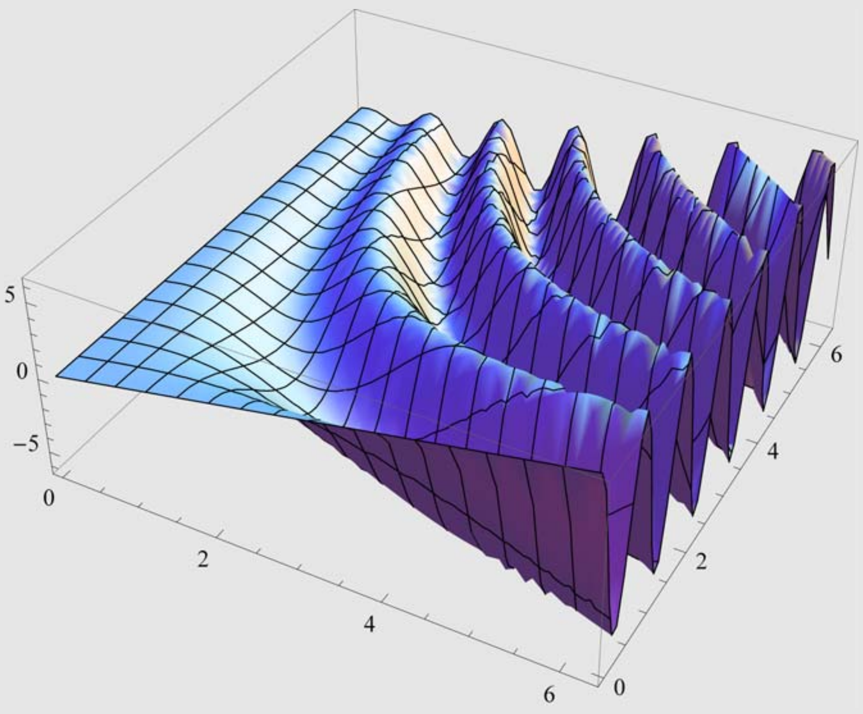
If you want to plot more than one function, you need to write the functions you want to plot as a list, so you need to use curly brackets:

```
Plot[{f[t], Exp[t]}, {t, -5, 8}]
```



You can plot functions of two variables using the command **Plot3D**:

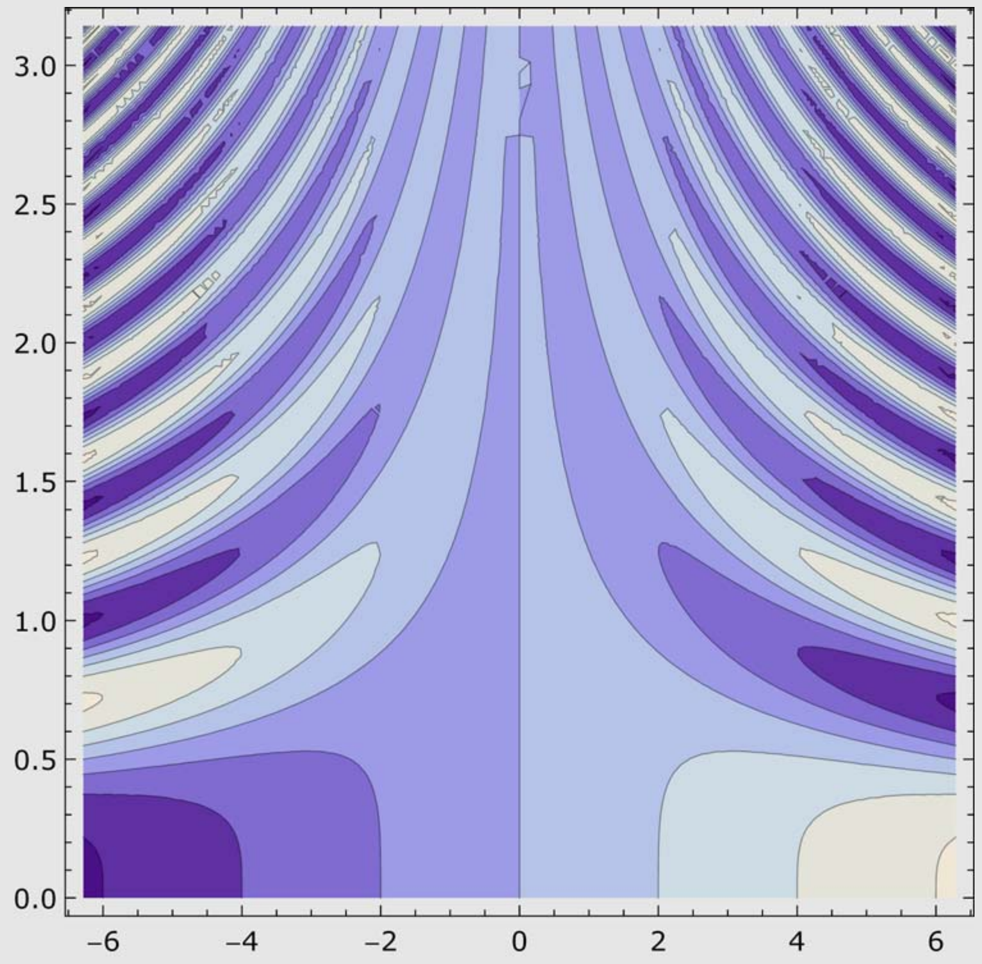
```
Plot3D[x * Cos[x * y], {x, 0, 2 Pi}, {y, 0, 2 Pi}]
```



You can click on the above object with the mouse and rotate the image to view it from different angles. How cool is that!

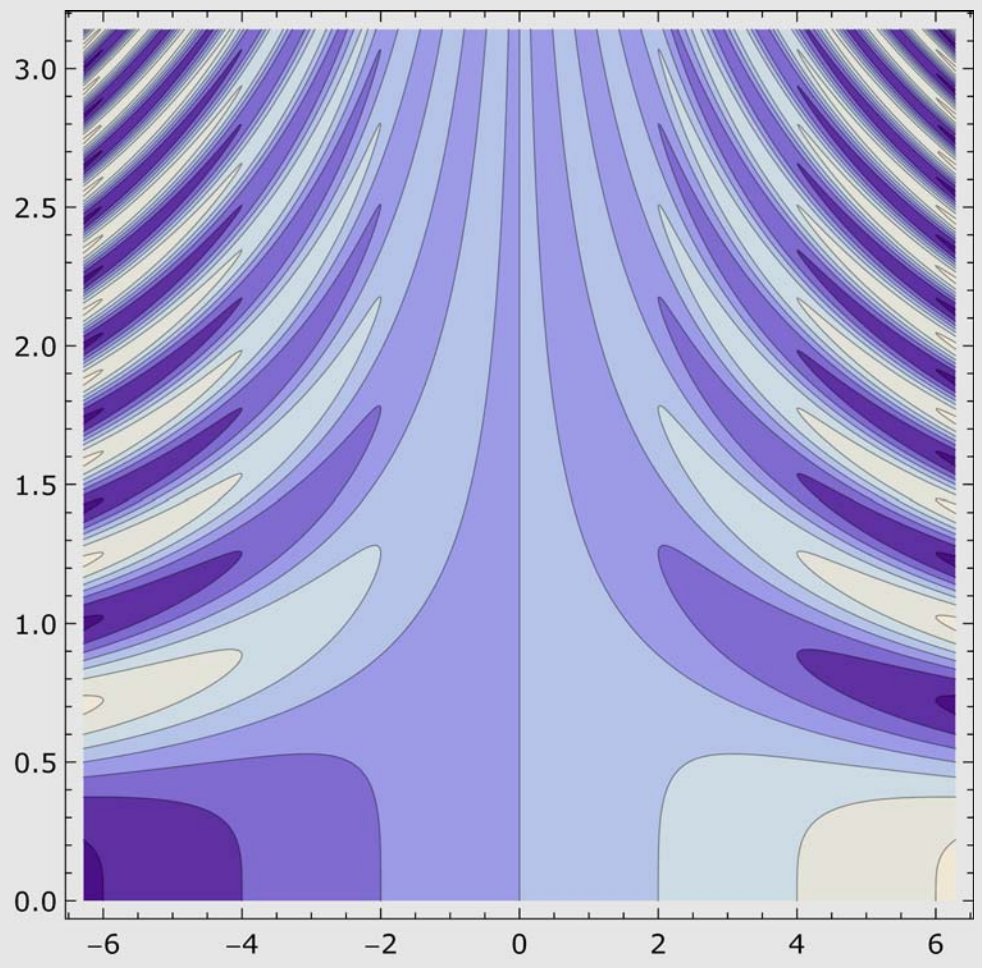
You can also create contour plots easily:

```
ContourPlot[x * Cos[x * y^2], {x, -2 Pi, 2 Pi}, {y, 0, Pi}]
```



Adding options to the plot can make it look better. Here, we should use more points to make the plot (this means the plot will take longer to create, of course). You creat the arrow → by typiing the dash symbol and then the less than sign. *Mathematica* will automatically join them to make the arrow.

```
ContourPlot[x * Cos[x * y^2], {x, -2 Pi, 2 Pi}, {y, 0, Pi},
 PlotPoints → 100]
```



You can learn more about what options are available for any specific command by typing the following

?? ContourPlot

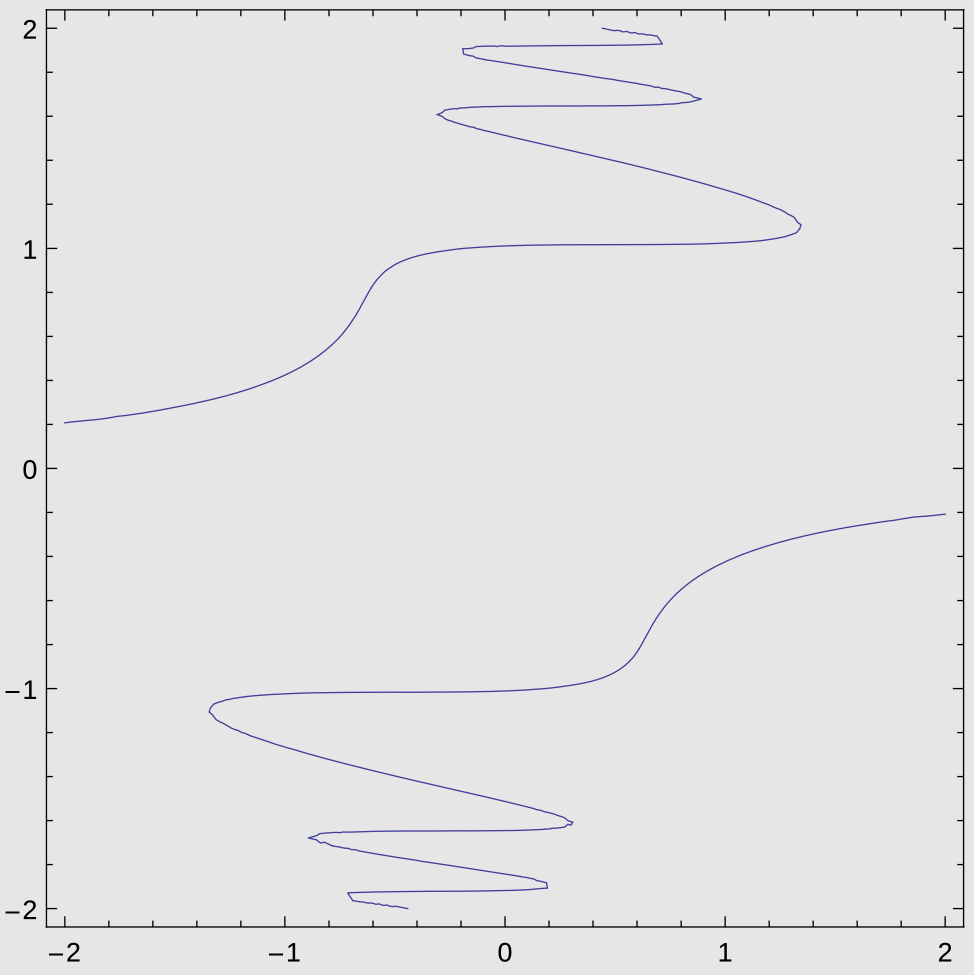
ContourPlot[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}]
generates a contour plot of f as a function of x and y.
ContourPlot[f == g, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots contour lines for which $f = g$.
ContourPlot[{f_1 == g_1, f_2 == g_2, …}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}]
plots several contour lines. ≫

```
Attributes[ContourPlot] = {HoldAll, Protected}

Options[ContourPlot] = {AlignmentPoint → Center, AspectRatio → 1,
  Axes → False, AxesLabel → None, AxesOrigin → Automatic, AxesStyle → {},
  Background → None, BaselinePosition → Automatic, BaseStyle → {},
  BoundaryStyle → None, ClippingStyle → None, ColorFunction → Automatic,
  ColorFunctionScaling → True, ColorOutput → Automatic,
  ContentSelectable → Automatic, ContourLabels → None,
  ContourLines → True, Contours → Automatic, ContourShading → Automatic,
  ContourStyle → Automatic, DisplayFunction :→ $DisplayFunction,
  Epilog → {}, Evaluated → System`Private`$Evaluated,
  EvaluationMonitor → None, Exclusions → Automatic,
  ExclusionsStyle → None, FormatType :→ TraditionalForm, Frame → True,
  FrameLabel → None, FrameStyle → {}, FrameTicks → Automatic,
  FrameTicksStyle → {}, GridLines → None, GridLinesStyle → {},
  ImageMargins → 0., ImagePadding → All, ImageSize → Automatic,
  LabelStyle → {}, MaxRecursion → Automatic, Mesh → None,
  MeshFunctions → {}, MeshStyle → Automatic, Method → Automatic,
  PerformanceGoal :→ $PerformanceGoal, PlotLabel → None,
  PlotPoints → Automatic, PlotRange → {Full, Full, Automatic},
  PlotRangeClipping → True, PlotRangePadding → Automatic,
  PlotRegion → Automatic, PreserveImageOptions → Automatic,
  Prolog → {}, RegionFunction → (True &), RotateLabel → True,
  Ticks → Automatic, TicksStyle → {}, WorkingPrecision → MachinePrecision}
```

You can plot implicit functions using the **`ContourPlot`** command:

```
ContourPlot[Cos[y * x + y^4] + x * y == 1 / 2, {x, -2, 2}, {y, -2, 2}]
```

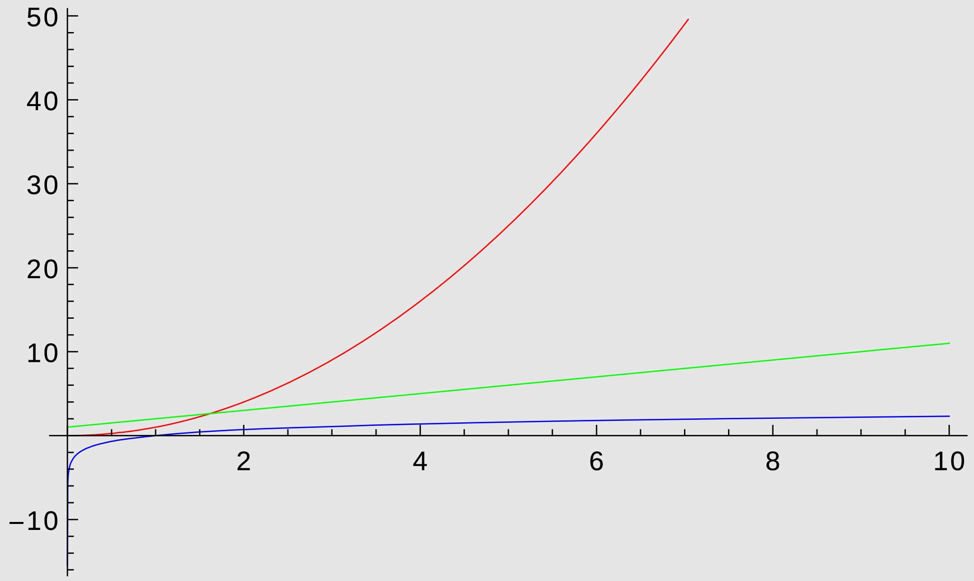


• Plotting Styles

Here we see some of the ways we can distinguish multiplie functions on a single plot. The **`PlotStyle`** option can change the color of the curves, or the thickness of the curve, or both.

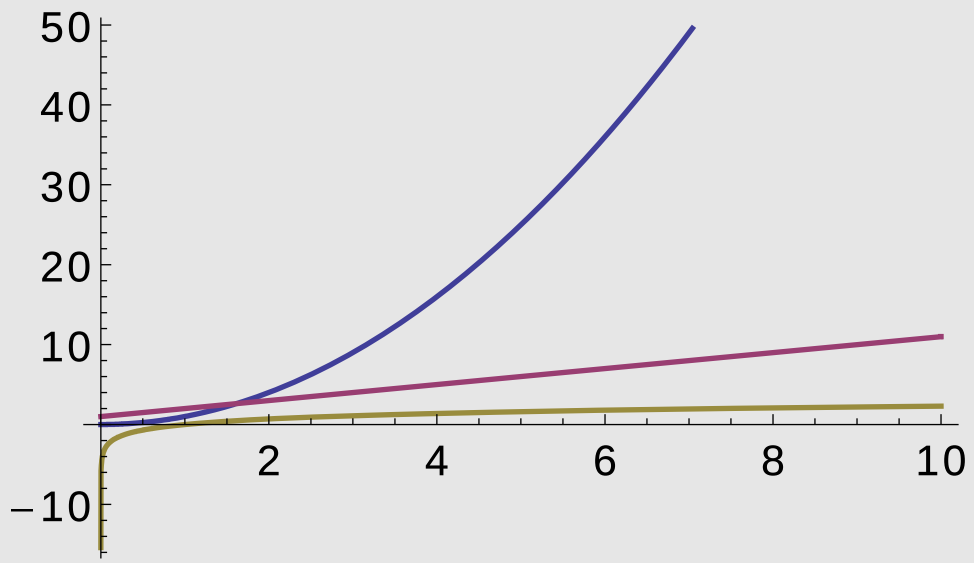
Notice that when we plot more than one function, we are plotting a list of functions, so they must be enclosed in curly brackets.

```
Plot[{x^2, x+1, Log[x]}, {x, 0, 10} , PlotStyle→{Red, Green, Blue}]
```

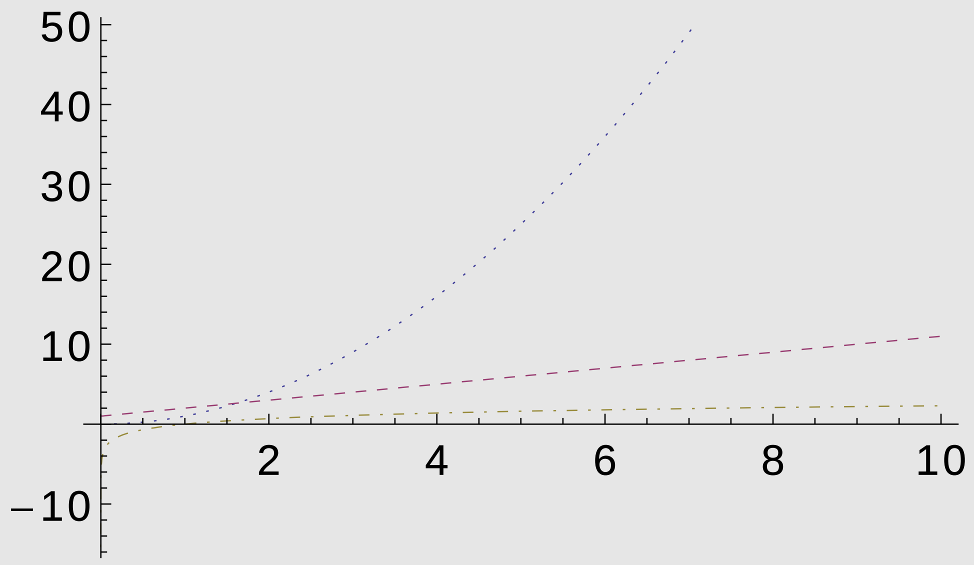


You can let *Mathematica* choose the color automatically, and make the lines thicker:

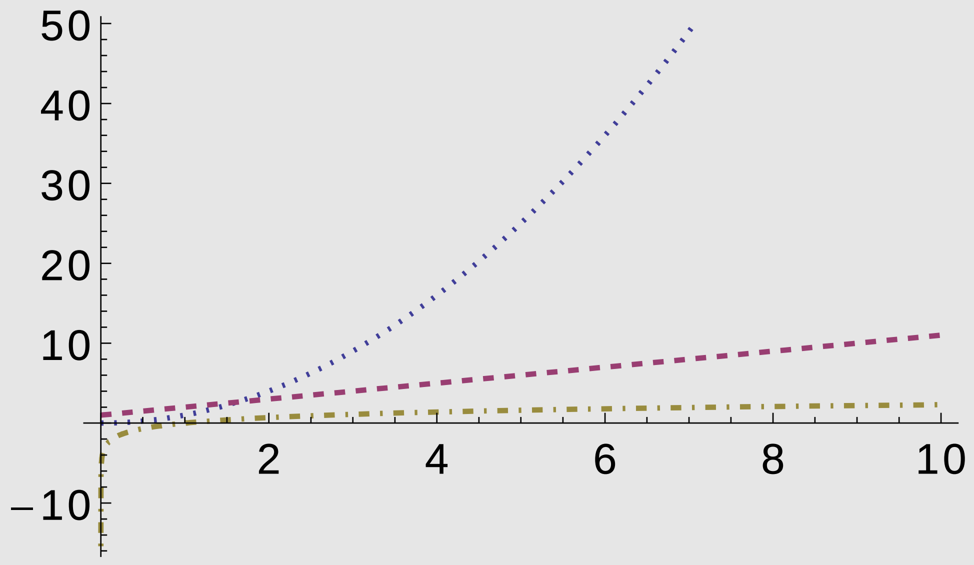
```
Plot[{x^2, x+1, Log[x]}, {x, 0, 10} , PlotStyle→Thick]
```



```
Plot[{x^2, x + 1, Log[x]}, {x, 0, 10} ,
 PlotStyle → {Dotted, Dashed, DotDashed}]
```



```
Plot[{x^2, x + 1, Log[x]}, {x, 0, 10} ,
 PlotStyle → {{Dotted, Thick}, {Dashed, Thick}, {DotDashed, Thick}}]
```



Solving Algebraic Equations

Before we solve an equation, we need to know another form of the "equals" sign used by *Mathematica*. The Double Equals, `==`, is used when defining equations or checking for equality:

```
3 == 3
```

```
True
```

```
3 == 4
```

```
False
```

```
equation = x^2 + 2 x - 1 == 0
```

$$-1 + 2x + x^2 == 0$$

Note that the variable equation is now defined in the kernel:

```
equation
```

$$-1 + 2x + x^2 == 0$$

```
Solve[equation, x]
```

$$\left\{\left\{x \to -1 - \sqrt{2}\right\}, \left\{x \to -1 + \sqrt{2}\right\}\right\}$$

The above is a symbolic solver, and we have to give it two parts--the equation we want it to work on, and the variable we want *Mathematica* to solve for. Let's look at it a little more closely:

```
quad = a x^2 + b x + c == 0
```

$$c + b\,x + a\,x^2 == 0$$

```
Solve[quad, x]
```

$$\left\{\left\{x \to \frac{-b - \sqrt{b^2 - 4\,a\,c}}{2\,a}\right\}, \left\{x \to \frac{-b + \sqrt{b^2 - 4\,a\,c}}{2\,a}\right\}\right\}$$

```
Solve[quad, b]
```

$$\left\{\left\{b \to \frac{-c - a\,x^2}{x}\right\}\right\}$$

Do you understand why the two answers are different?

We can tack an `N` in front of `Solve` to use *Mathematica*'s numerical solving abilities:

```
NSolve[equation, x]
```

```
{{x → -2.41421}, {x → 0.414214}}
```

Here is an example of a *Mathematica* result which may at first confuse you:

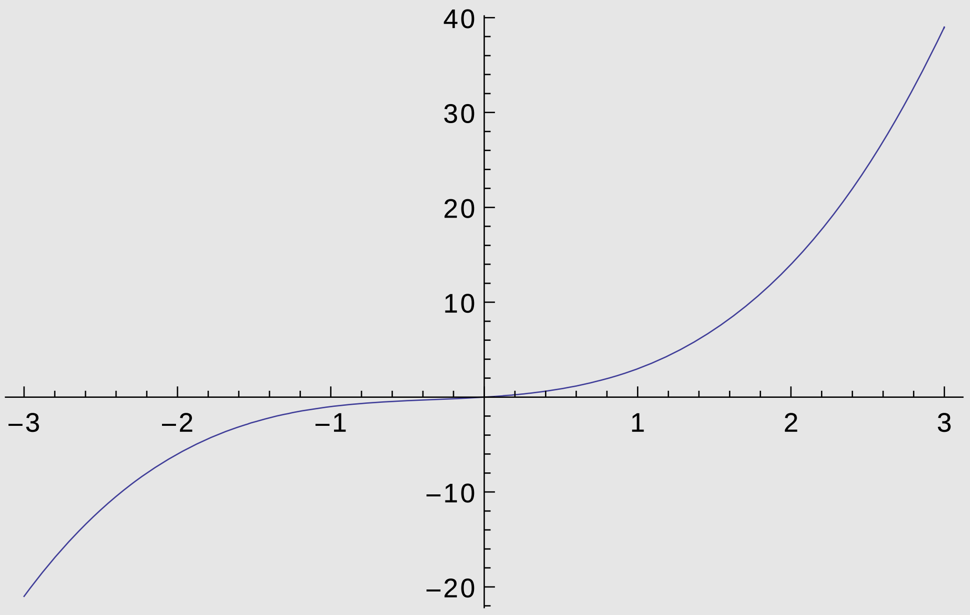
```
f[x_] = x^3 + 1 x^2 + 1 x
NSolve[f[x] == 0, x]
```

$$x + x^2 + x^3$$

```
{{x → -0.5 - 0.866025 ⅈ},
  {x → -0.5 + 0.866025 ⅈ}, {x → 0.}}
```

The i in the above refer to the imaginary part of a complex number, and results from taking the square root of a negative number in the quadratic formula that can be used to solve for the roots. Hopefully you will go on to study complex analysis, since it is very interesting and powerful. To interpret the above result, you need to know that the function **f[x]** has only one real root (as seen from the graph below), and the root is at x=0.

```
Plot[f[x], {x, -3, 3}]
```



Another useful solver available in *Mathematica* is **Reduce**. **Reduce** solves equations and includes conditions on the solution, whereas **Solve** does not include conditions.

```
Solve[{x == 1, x == a}, {x}]
```

```
{}
```

Solve tells us there is no solution to these two equations.

```
Reduce[{x == 1, x == a}, {x}]
```

```
a == 1 && x == 1
```

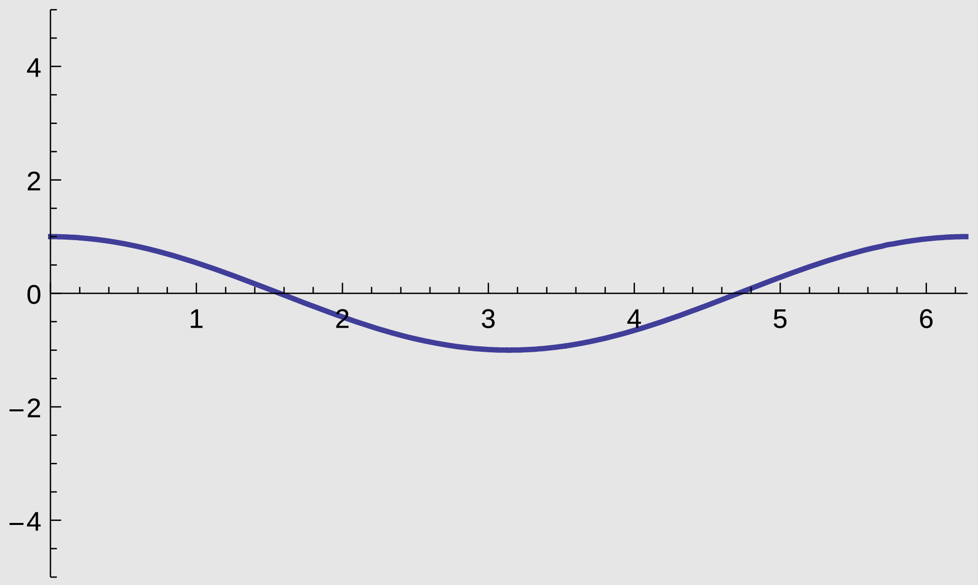
The symbol && means "and", the symbol || means "or" in *Mathematica.*

Reduce tells us that there is one solution, if $a = 1$ and $x = 1$.

Manipulate

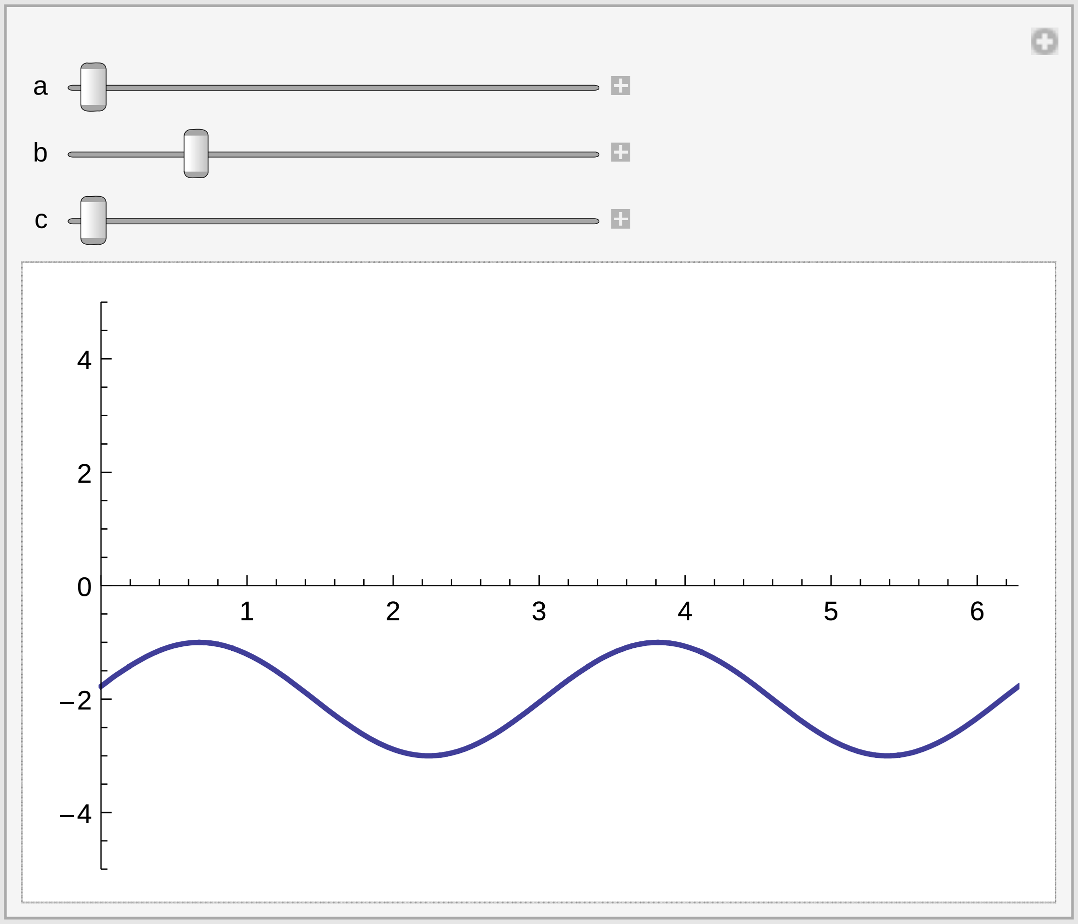
`Manipulate` is a wonderful command that produces output as a function of a parameter. It can be used to create animations, among other things.

```
Plot[Cos[x], {x, 0, 2 Pi}, PlotStyle → Thick,
 PlotRange → {{0, 2 Pi}, {-5, 5}}]
```



We can investigate the behaviour of the function $f(x) = \cos(a\,x + b) + c$ using `Manipulate`.

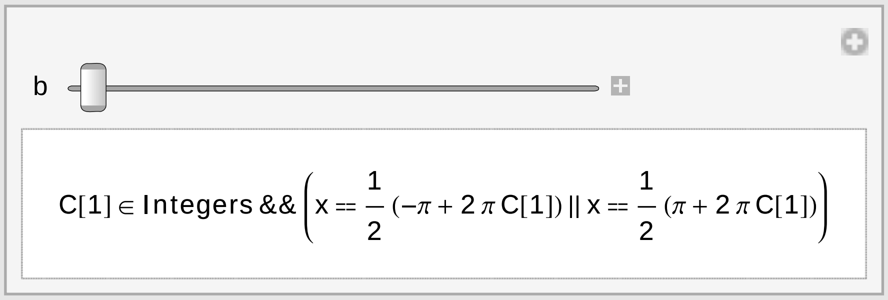
```
Manipulate[Plot[Cos[a * x + b] + c, {x, 0, 2 Pi}, PlotStyle → Thick,
  PlotRange → {{0, 2 Pi}, {-5, 5}}], {a, -2, 2}, {b, 0, 2 Pi},
 {c, -2, 2}]
```



If you click on the plus sign beside each slider, you can see the value of the parameter that the plot is for. You can use the slider to change the value of the parameter, and even create animations! Try it! Obvioulsy, too many parameters still becomes a bit much if you are doing animations, but certainly this can help you explore the behaviour of fucntions relative to a parameter.

Manipulate also works with things besides plots. Here I tell Manipulate to choose *b* values from −2 to 2 in steps of 1 (so *b* = −2, −1, 0, 1, 2):

```
Manipulate[Reduce[Cos[b * x] == -1, x], {b, -2, 2, 1}]
```



C[1] ∈ Integers && $\left(x == \frac{1}{2}(-\pi + 2\pi\, C[1]) \,||\, x == \frac{1}{2}(\pi + 2\pi\, C[1])\right)$

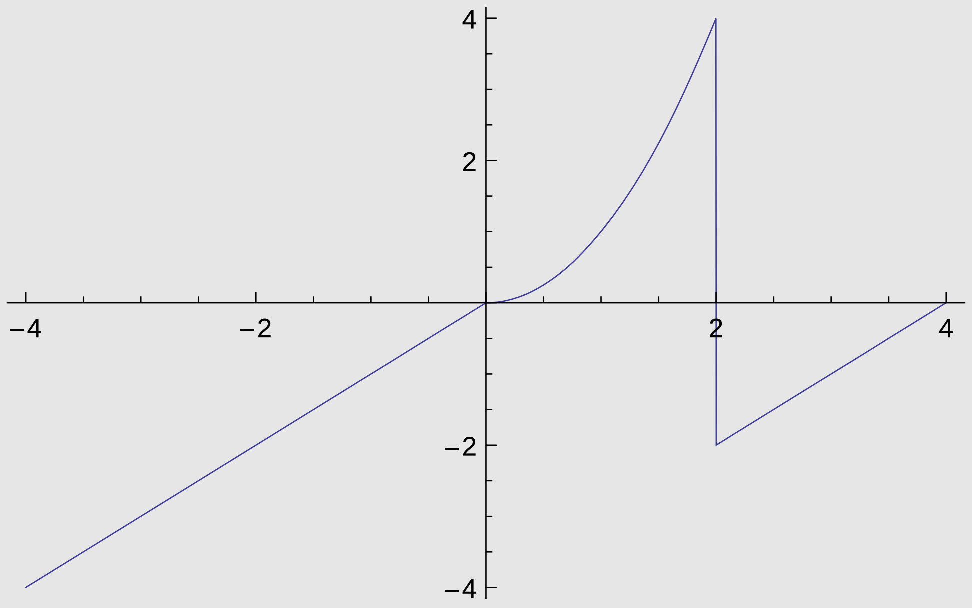
Defining a Piecewise Function

Here is how you define a piecewise function in *Mathematica*:

```
f[x_] = Piecewise[{{x, x < 0}, {x^2, 0 ≤ x ≤ 2}, {x - 4, x > 2}}]
```

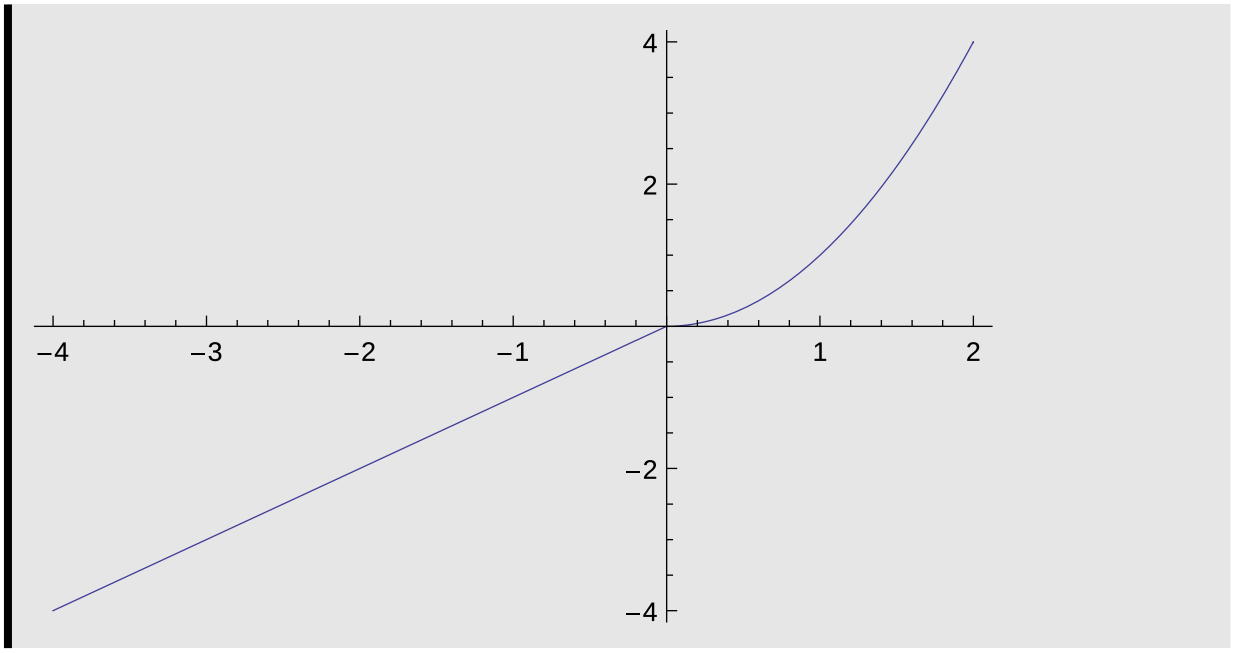
$$\begin{cases} x & x < 0 \\ x^2 & 0 \le x \le 2 \\ -4 + x & x > 2 \end{cases}$$

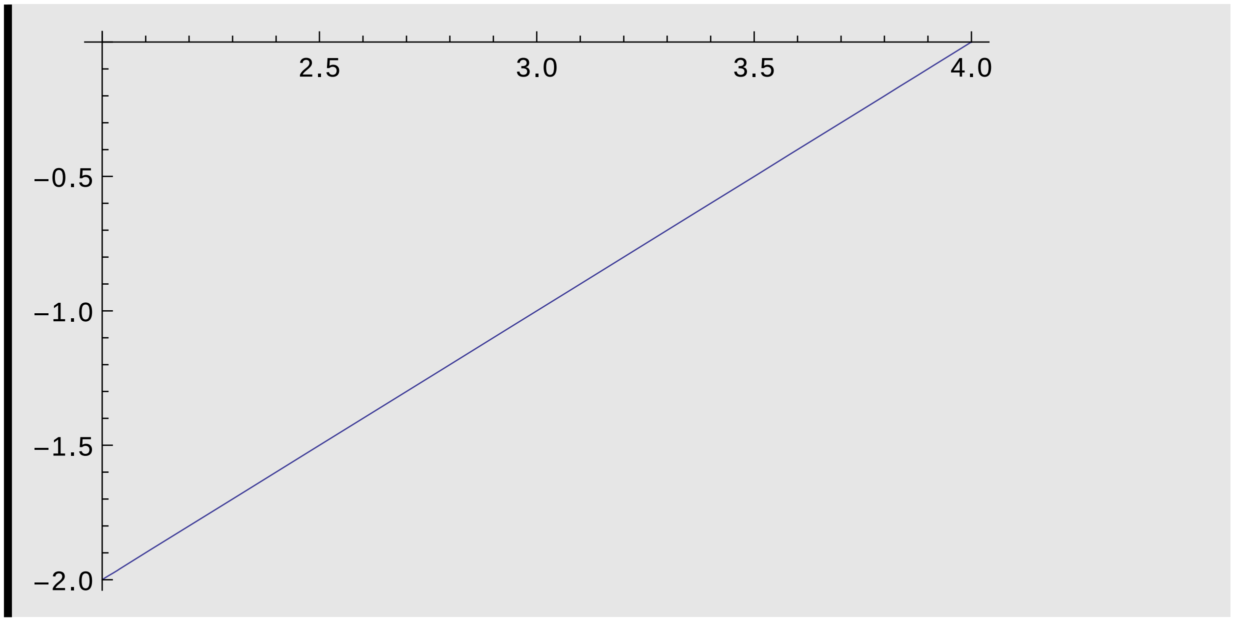
```
Plot[f[x], {x, -4, 4}]
```

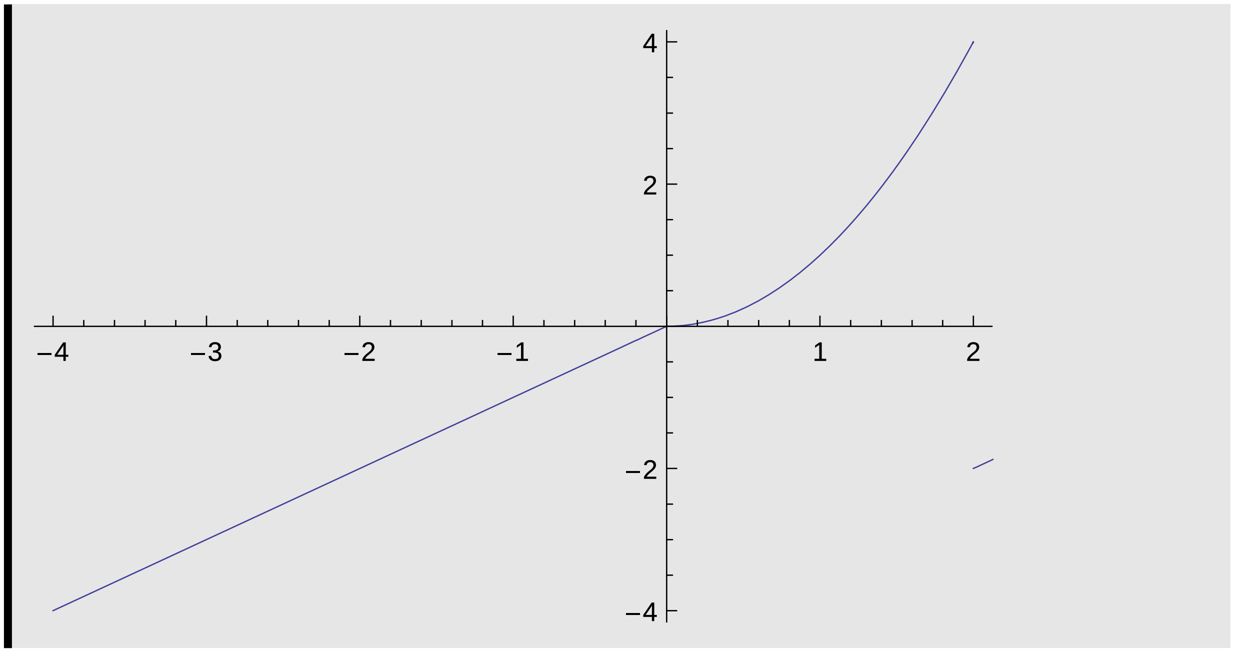


Note the problem in the graphics--since *Mathematica* assumes the function is continuous, it draws a line between the points (2, 4) and (2, −2), which is mathematically incorrect. The correct graph looks like:

```
plot1 = Plot[f[x], {x, -4, 2}]
plot2 = Plot[f[x], {x, 2, 4}]
Show[plot1, plot2]
```


4
2
−2
−4
−4
−3
−2
−1
1
2


2.5
3.0
3.5
4.0
−0.5
−1.0
−1.5
−2.0



• Confuse a Kernel

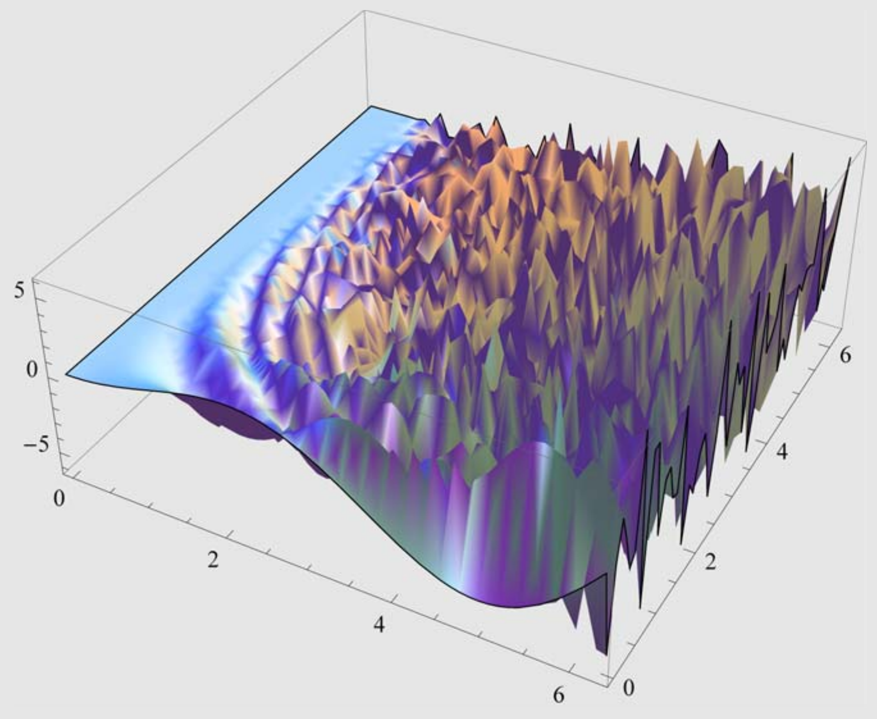
The *Mathematica* kernel stores all the information from a given session, even when the notebook is closed and another notebook is opened, until *Mathematica* itself is closed or the kernel is closed.

One thing you may want to notice is that under kernel in the menu there is a command Abort Evaluation. Sometimes this can help you. Sometimes you will need the command Quit Kernel, losing all the information stored in it! Quitting the kernel is therefore a last resort. All your typ-ing will not be lost, but all your data stored in the kernel will be.

It is therefore important to start with a new kernel whenever you begin a *Mathematica* session, so your calcululations will not be corrupted by information other people have entered into the kernel before you sat down at the computer. By making sure *Mathematica* is not already running when you start every single time you use *Mathematica* you will avoid possible confusion later.

Mathematica is pretty fast, but you can slow it down easily if you ask it to do too much. This easily happens with plots.

```
Plot3D[x * Cos[5 x * y] Sin[x^(1 + y)], {x, 0, 2 Pi}, {y, 0, 2 Pi},
 Mesh → False]
```



Adding more points will not make this sketch look better--the function is too oscillatory.

If *Mathematica* becomes sluggish, look at what you are asking it to do. If it takes a long time to complete a process, you may want to abort the evaluation or even quit the kernel!

• Curve Fitting

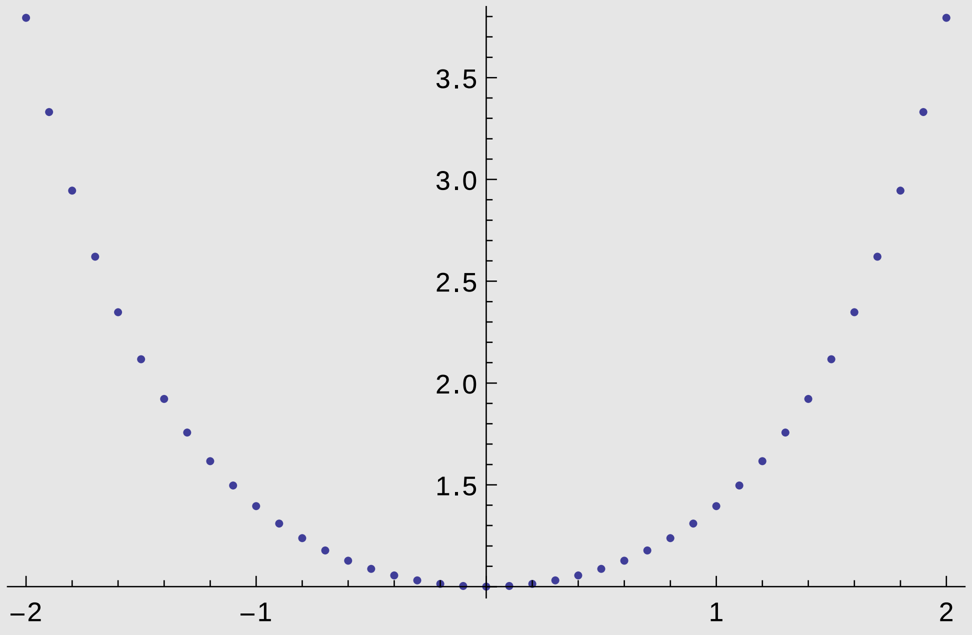
As an example of some of the power of *Mathematica*, let's fit a curve to some data. What we are going to do is a least squares fit of a polynomial function to the data.

You should know that this is not how I prefer to use *Mathematica*, and we are doing this only to help you see *Mathematica*'s usefulness. Using *Mathematica* commands without understanding the math behind them is NOT ONE OF OUR GOALS!! We shall be using *Mathematica* to help elucidate our understanding of concepts, and help us with things that are difficult to do by hand. We shall not be using it to do our thinking for us.

```
Clear[data]
data = Table[{i, Exp[i^2 / 3.]}, {i, -2, 2, 0.1}]
```

```
{{−2., 3.79367}, {−1.9, 3.3312}, {−1.8, 2.94468}, {−1.7, 2.62042},
  {−1.6, 2.34746}, {−1.5, 2.117}, {−1.4, 1.92194}, {−1.3, 1.75652},
  {−1.2, 1.61607}, {−1.1, 1.49681}, {−1., 1.39561}, {−0.9, 1.30996},
  {−0.8, 1.2378}, {−0.7, 1.17743}, {−0.6, 1.1275}, {−0.5, 1.0869}, {−0.4, 1.05478},
  {−0.3, 1.03045}, {−0.2, 1.01342}, {−0.1, 1.00334}, {1.11022×10^-16, 1.},
  {0.1, 1.00334}, {0.2, 1.01342}, {0.3, 1.03045}, {0.4, 1.05478}, {0.5, 1.0869},
  {0.6, 1.1275}, {0.7, 1.17743}, {0.8, 1.2378}, {0.9, 1.30996}, {1., 1.39561},
  {1.1, 1.49681}, {1.2, 1.61607}, {1.3, 1.75652}, {1.4, 1.92194}, {1.5, 2.117},
  {1.6, 2.34746}, {1.7, 2.62042}, {1.8, 2.94468}, {1.9, 3.3312}, {2., 3.79367}}
```

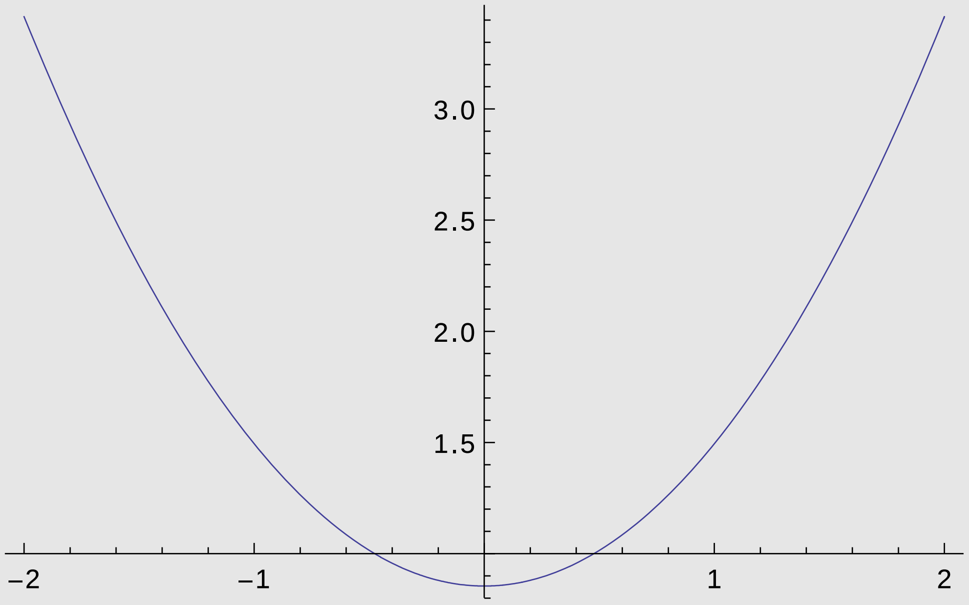
```
plotdata = ListPlot[data]
```



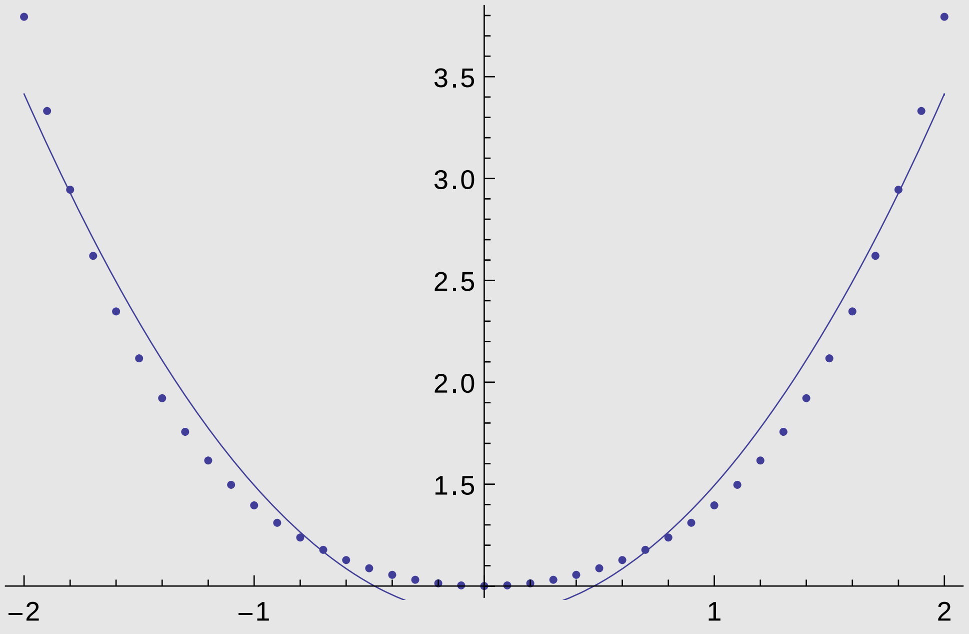
```
f[x_] = Fit[data, {1, x, x^2, x^3}, x]
```

$$0.854189 - 3.51694\times10^{-16}\,x + 0.640142\,x^2 + 2.13884\times10^{-16}\,x^3$$

```
plotmodel = Plot[f[x], {x, -2, 2}]
```



```
Show[plotdata, plotmodel]
```


3.5
3.0
2.5
2.0
1.5
−2
−1
1
2