Variables, Vectors, Arithmetic

At a basic level, Octave is a fancy calculator. Here's a snippet of a session in Octave:

```
octave-3.2.3:1> x=3;
octave-3.2.3:2> y=5;
octave-3.2.3:3> z=x+y
z =  8
```

In this session, we made three variables: x, y, and z. We assigned the values of x and y directly. The variable z is new and contains the sum of the values of x and y.

Octave knows about vectors as well as plain numbers. Here's how you make vectors.

```
octave-3.2.3:1> x=[ 3, 1, 7]
x =

   3   1   7
octave-3.2.3:2> y=[2 ; 9]
y =

   2
   9
```

Notice that x is a row vector; its entries are listed horizontally. On the other hand, y is a column vector. Its entries are listed vertically. The only difference when you enter them is the comma or semicolon used to separate the entries.

If two vectors have the same number of entries and are both column or both row vectors, you can add them together.

```
octave-3.2.3:3> x=[3,1,7]
x =

   3   1   7
octave-3.2.3:4> y=[2,9,8]
y =

   2   9   8

octave-3.2.3:5> z=x+y
z =

    5   10   15
```

The first entry of z is 5 because 3 + 2 = 5. The second entry of z is 10 because 1 + 9 = 10. And the third entry of z is 15 because 7 + 8 = 15.

We can access the entries of z as follows

```
octave-3.2.3:6> z(1)
ans =  5
octave-3.2.3:7> z(2)
ans =  10
octave-3.2.3:8> z(3)
ans =  15
```

You can do things with the new vector z just like you did with z and y. For example

```
octave-3.2.3:7> z+z
ans =

    10     20    30
```

The result of this computation is stored in a temporary variable called ans. You could also have assigned z+z to a new variable.

```
octave-3.2.3:8> w=z+z
w =

    10   20   30
```

You can also multiply a vector by a number.

```
octave-3.2.3:31> x=[3,1,7]
x =

   3   1   7

octave-3.2.3:32> z=4*x
z =

   12    4   28
```

Multiplying two vectors is a little more tricky, and we need to start by discussing matrices. Matrices are like vectors, except they are two-dimensional. Here's how you enter a matrix.

```
octave-3.2.3:6> A = [ 4, 9, 8; 1, -3, 5]
A =

   4   9   8
   1  -3   5
```

We say that `A` is a 2×3 matrix because it has two rows and three columns. You separate entries of a row with commas, and you separate rows with semicolons. Vectors are special cases of matrices: a row vector with 5 entries is a 1×5 matrix, and a column vector with 5 entries is a 5×1 matrix.

There is a notion of matrix multiplication, but it might be different than you might have guessed at first. You can multiply a $n \times m$ matrix by a $m \times k$ matrix, and the result will be a $n \times k$ matrix.

Consider the following session:

```
octave-3.2.3:10> x = [1, 5, 2]
x =

   1   5   2

octave-3.2.3:12> y = [3, 4, 9]
y =

   3   4   9

octave-3.2.3:13> z = [3; 4; 9]
z =

   3
   4
   9

octave-3.2.3:14> x*y
error: operator *: nonconformant arguments (op1 is 1x3, op2 is 1x3)
octave-3.2.3:14> x*z
ans =  41
```

The first multiplication gives an error because you can't multiply a 1×3 matrix by a 1×3 matrix. The second multiplication works because `x` is a 1×3 matrix and `y` is a 3×1 matrix. The result is a 1×1 matrix, which in Octave is the same thing as a plain old number. Notice that the answer, 41, is the dot product of `x` and `y`. Is it legal to multiply `z*x`? What size of a matrix will you get? We'll talk more about matrix multiplication later in the class.

Often you want to multiply the entries of two vectors together term by term. Octave has a special command for this: `.*` (i.e. dot-times). For example:

```
octave-3.2.3:16> x = [2, 5, 2]
x =

   2   5   2
```

```
octave-3.2.3:17> y = [3, 4, 9]
y =

   3   4   9

octave-3.2.3:18> x.*y
ans =

    6   20   18
```

The first entry of ans is 6 since the product of 2 and 3 is 6.

You can also divide the entries of two vectors term by term with `./` (i.e dot-divide).

```
octave-3.2.3:22> x./y
ans =

   0.66667   1.25000   0.22222
```

And you can raise each entry of a vector to a power with `.^` (i.e. dot-power).

```
octave-3.2.3:23> x.^2
ans =

    4   25    4
```

Basic Plotting

Here's a handy way to make a large vector with equally spaced entries:

```
octave-3.2.3:24> x=[0:0.1:1]
x =

 Columns 1 through 8:

   0.00000   0.10000   0.20000   0.30000   0.40000   0.50000   0.60000   0.70000

 Columns 9 through 11:

   0.80000   0.90000   1.00000
```

The command `[0:0.1:1]` makes a row vector that starts at 0 and increases each entry by 0.1, until it reaches 1.

For very large vectors, you might not want to see the output. For example, if you enter `x = [0: 0.01 :1]` then `x` will have 101 entries. If you end a line with a semi-colon, you won't see the output. For example:

```
octave-3.2.3:37> x=[0:0.05:1];
octave-3.2.3:38> x
x =

 Columns 1 through 8:

   0.00000   0.05000   0.10000   0.15000   0.20000   0.25000   0.30000   0.35000

 Columns 9 through 16:

   0.40000   0.45000   0.50000   0.55000   0.60000   0.65000   0.70000   0.75000

 Columns 17 through 21:

   0.80000   0.85000   0.90000   0.95000   1.00000
```

Suppose we want to plot the polynomial $1 + 2 * x^2 - 3x^3$ over the interval $[-1, 1]$. As a first step, we make a vector for x values:

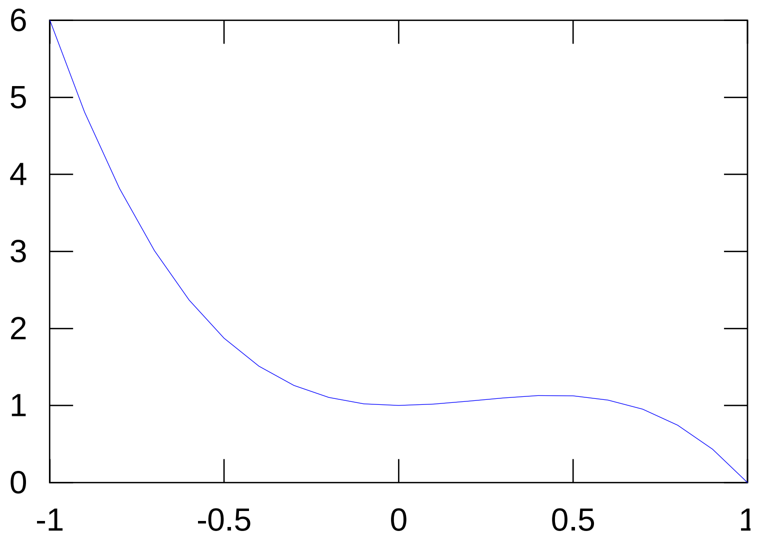
```
octave-3.2.3:39> x=[-1:0.1:1];
```

Then we make a vector for the y values:

```
octave-3.2.3:40> y = 1 + 2*x.^2 - 3*x.^3;
```

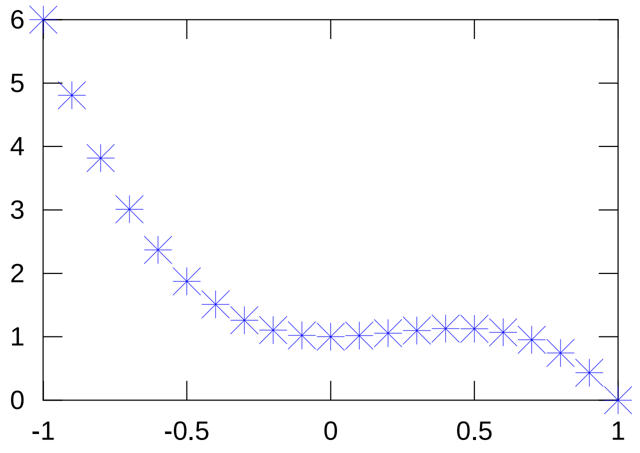
Finally, we plot these x and y coordinate points with

```
octave-3.2.3:41> plot(x,y)
```



This may look like a continuous curve, but Octave has plotted each x and y coordinate and connected them with straight lines. You can see the individual points that were plotted using

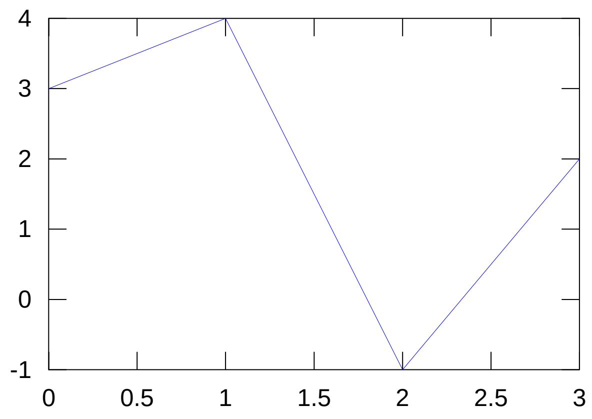
```
octave-3.2.3:41> plot(x,y,'*')
```



Basic Plots

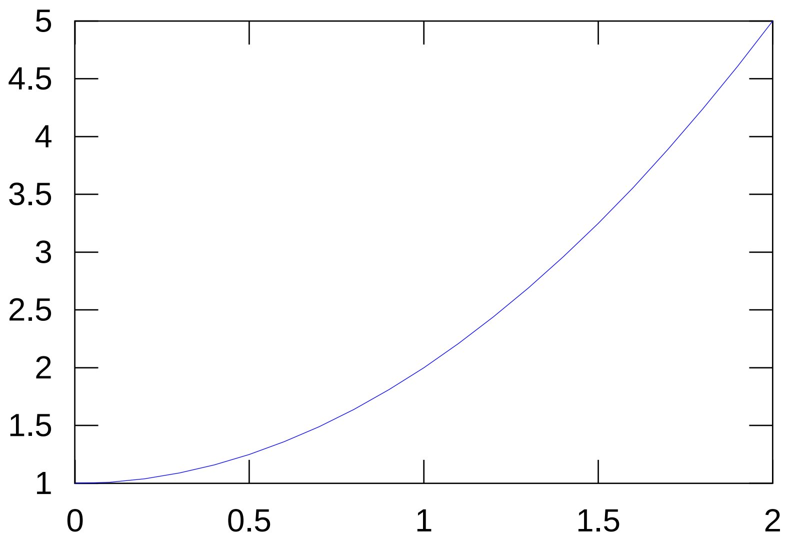
If x is a vector of 10 x-coordinates and y is a vector of 10 y-coordinates, then `plot(x,y)` plots 10 points where the x-coordinates come from the x vector and the y-coordinates come from the y vector. The points are connected with straight lines.

```
octave-3.2.3:11> x=[0,1,2,3];
octave-3.2.3:12> y=[3,4,-1,2];
octave-3.2.3:13> plot(x,y)
```



It is important that the x and y vectors must have the same number of entries. This is usually done by first constructing a vector of x entries, and then using x to make a vector of the same length with the y-coordinates. For example:

```
octave-3.2.3:11> x=[0:0.1:2];
octave-3.2.3:12> y=1+x.^2;
octave-3.2.3:13> plot(x,y)
```

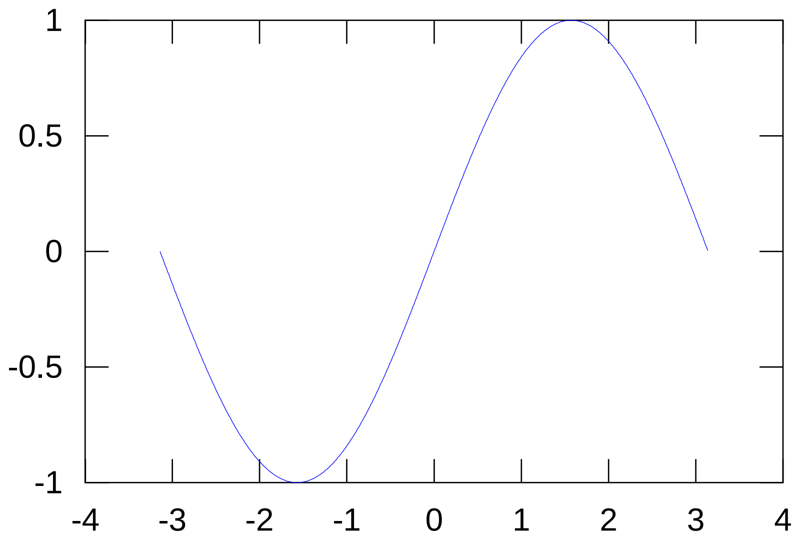


In this next example, we make a plot of the function $\sin(x)$ over the range $-\pi \le x \le \pi$. To create the vector of x-coordinates, it would be helpful if Octave knew about the value of π; it does:

```
octave-3.2.3:22> x=[-pi:0.01:pi];
```

Octave's sin function behaves nicely when you give it a vector of input; it returns a vector of the same size where the sin function has been applied to each entry. This is exactly what you want for making a plot.

```
octave-3.2.3:23> y=sin(x);
octave-3.2.3:24> plot(x,y)
```



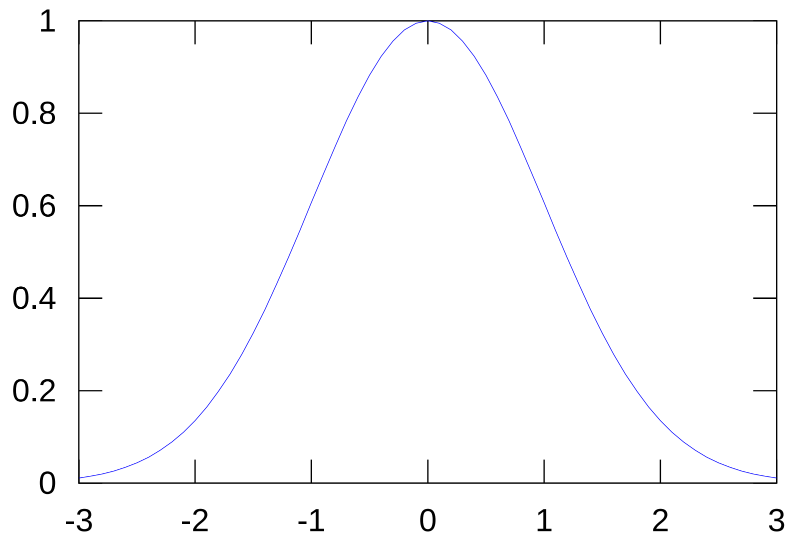
Most of the mathematical functions you know are available in Octave, and behave the same way when you feed them vector inputs.

Octave function	Mathematical function
`sin`	sin
`cos`	cos
`tan`	tan
`sqrt`	$\sqrt{x}$ (square root)
`exp`	exp (i.e. $\exp(x) = e^x$)
`log`	ln (logarithm base e)
`log10`	$\log_{10}$ (logarithm base 10)
`atan`	arctan (the inverse function of tan)
`asin`	arcsin
`factorial`	$x!$ (factorial)
`abs`	$\lvert x\rvert$ (absolute value)

Inline functions

Sometimes it is handy to make your own mathematical functions. For example, you might be working with the Gaussian function $g(x) = e^{-x^2/2}$. You can plot this function as follows

```
octave-3.2.3:88> x=[-3:0.1:3];
octave-3.2.3:89> plot(x,exp(-x.^2/2))
```



This function is the well-known bell curve. The following commands make a new function gauss that behaves much the same way as `sin`, `cos`, and so forth.

```
octave-3.2.3:90> gauss = @(t) exp(-t.^2/2)
gauss =
@(t) exp (-t .^ 2 / 2)
```

These commands say that gauss is a function that takes one input (called `t` for the purposes of defining the function) and returns $e^{-t^2/2}$. You can now use your new function:

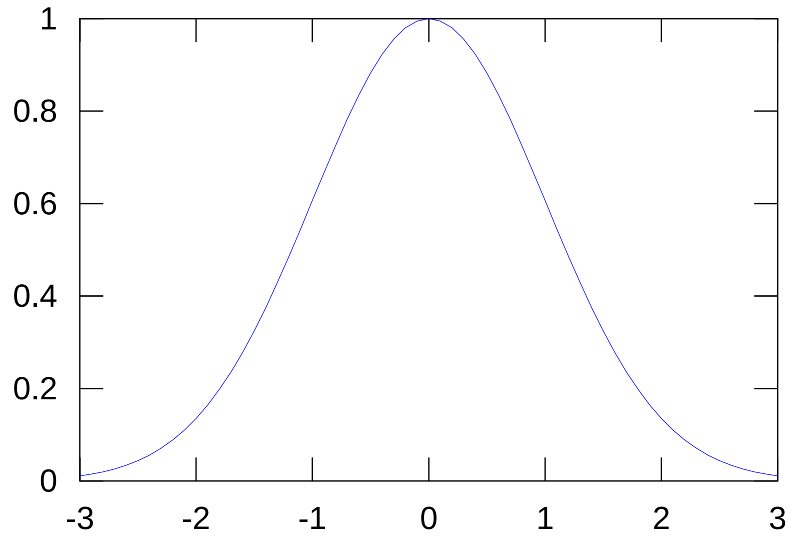
```
octave-3.2.3:91> gauss(0)
ans =  1
octave-3.2.3:92> gauss(1)
ans =  0.60653
octave-3.2.3:93> 1/sqrt(e)
ans =  0.60653
octave-3.2.3:94> gauss(2)
ans =  0.13534
octave-3.2.3:95> 1/(e*e)
ans =  0.13534
octave-3.2.3:96> gauss([0,1,2])
ans =

   1.00000   0.60653   0.13534
```

Take a moment and figure out what was going on in the computations above. Why is `gauss(2)` the same as $1/e^2$? (Notice that Octave knows about the number e, it's just `e`!).

We can easily plot the gauss function:

```
octave-3.2.3:97> x=[-3:0.1:3];
octave-3.2.3:98> plot(x,gauss(x))
```

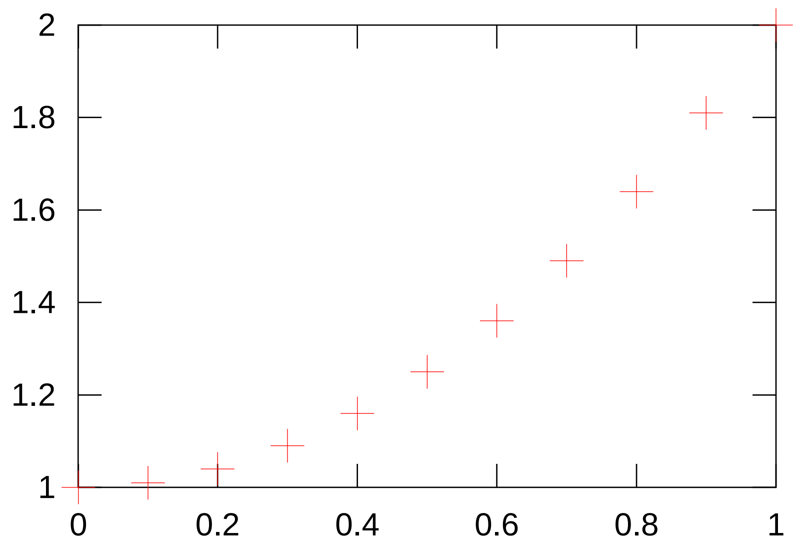


Properties of plots

The `plot` command allows you to specify certain properties of the graph. For example, you might want a different color line, or just points plotted rather than a line.

```
octave-3.2.3:42> x=[0:0.1:1];
octave-3.2.3:43> y=1+x.^2;
octave-3.2.3:44> plot(x,y,'r+')
```

The extra argument `'r+'` specifies that that red crosses should be used for the plot:



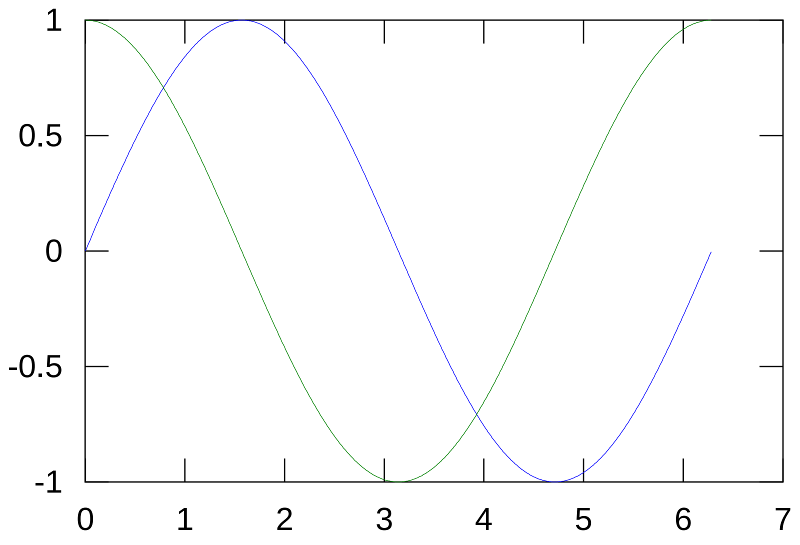
Some color styles: `r` (red), `g` (green), `b` (blue), `c` (cyan), `m` (magenta), `k` (black).

Some symbol styles: `.` (dots), `s` (squares or diamonds), + (crosses), ('*') (stars), `o` (circles or triangles).

Plotting more than one graph

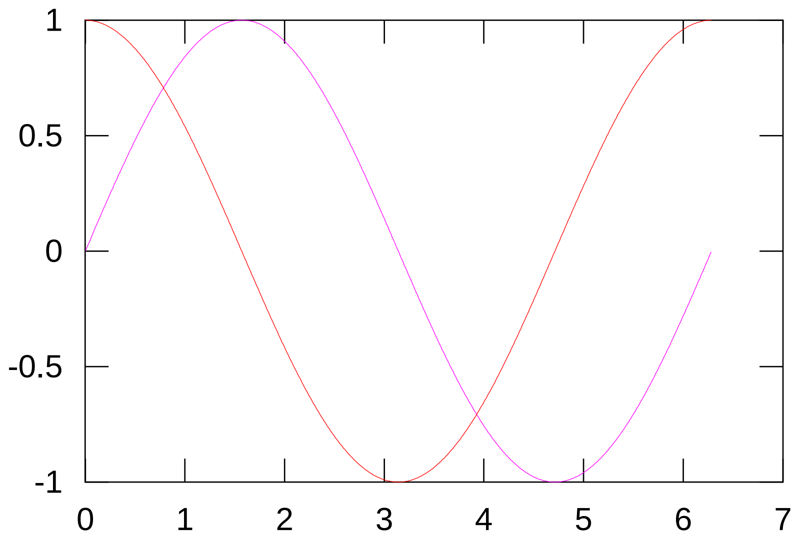
You can plot more than one graph at once with the plot command. Just specify each x and y coordinate vector.

```
octave-3.2.3:53> x=[0:0.01:2*pi];
octave-3.2.3:54> plot(x,sin(x),x,cos(x))
```



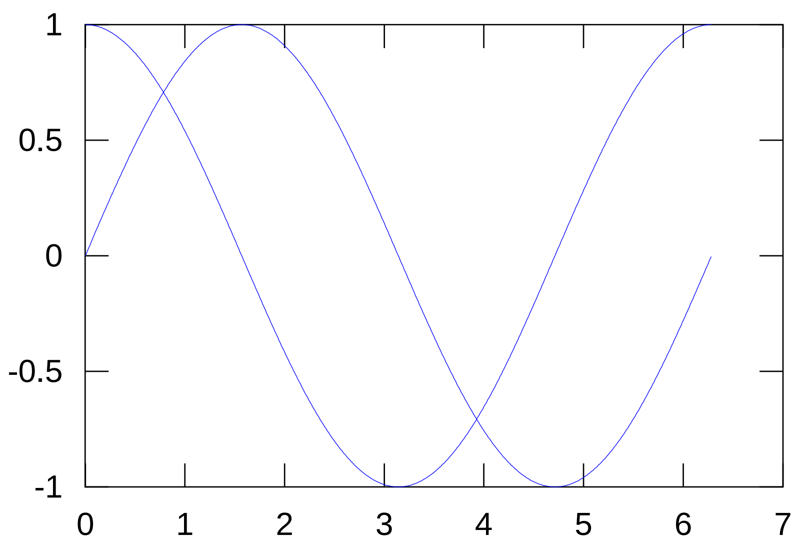
Octave picks colors for each curve. You can change them if you want.

```
octave-3.2.3:53> x=[0:0.01:2*pi];
octave-3.2.3:54> plot(x,sin(x),'m',x,cos(x),'r')
```



If you have already made a plot, you can add more curves to that plot using the `hold` command.

```
octave-3.2.3:60> x=[0:0.01:2*pi];
octave-3.2.3:61> plot(x,sin(x))
octave-3.2.3:62> hold on
octave-3.2.3:63> plot(x,cos(x))
octave-3.2.3:64> hold off
```

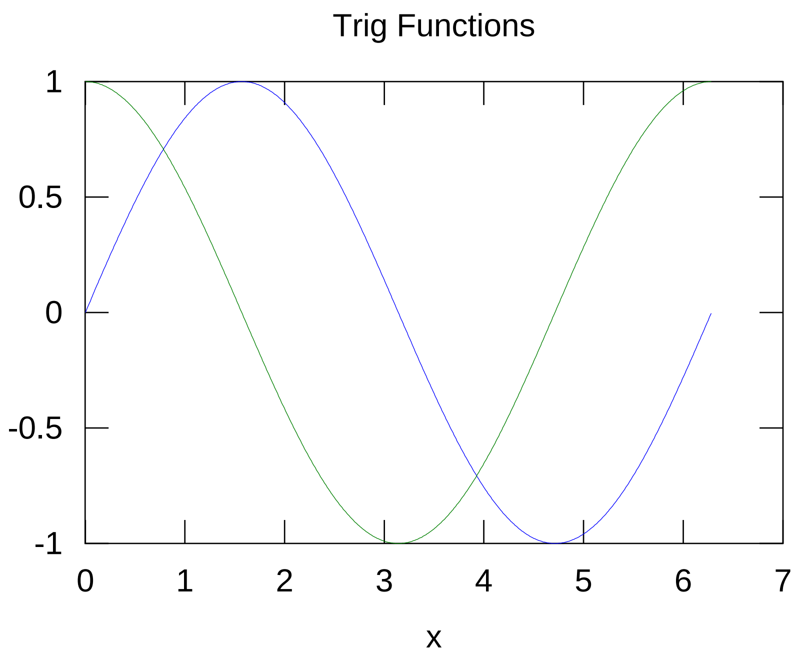


Everything you plot between `hold on` and `hold off` will appear in the same figure.

Labels

It can be helpful to add some text labels to a plot. Here are some examples:

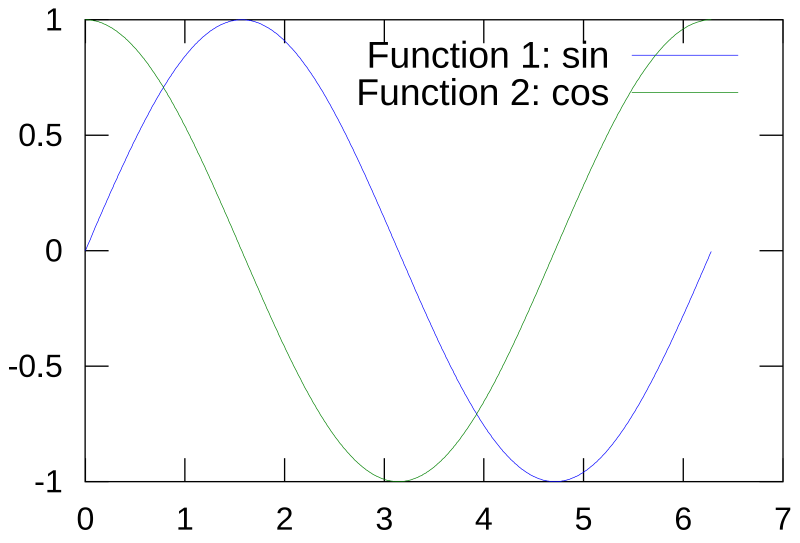
```
octave-3.2.3:66> plot(x,sin(x),x,cos(x))
octave-3.2.3:67> title('Trig Functions')
octave-3.2.3:68> ylabel('y')
octave-3.2.3:69> xlabel('x')
```



Unfortunately, on the Mac the labels are sometimes cut off (my plot above is missing the y label). Free software sometimes has this kind of glitch.

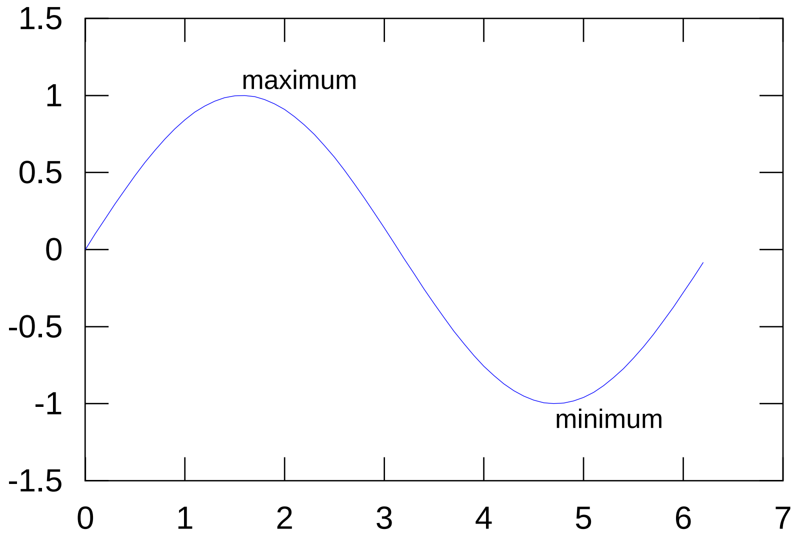
It can be helpful to add a legend to a plot as well.

```
octave-3.2.3:66> plot(x,sin(x),x,cos(x))
octave-3.2.3:68> legend('Function 1: sin','Function 2: cos')
```



You might want at some point to add a text label somewhere in a plot

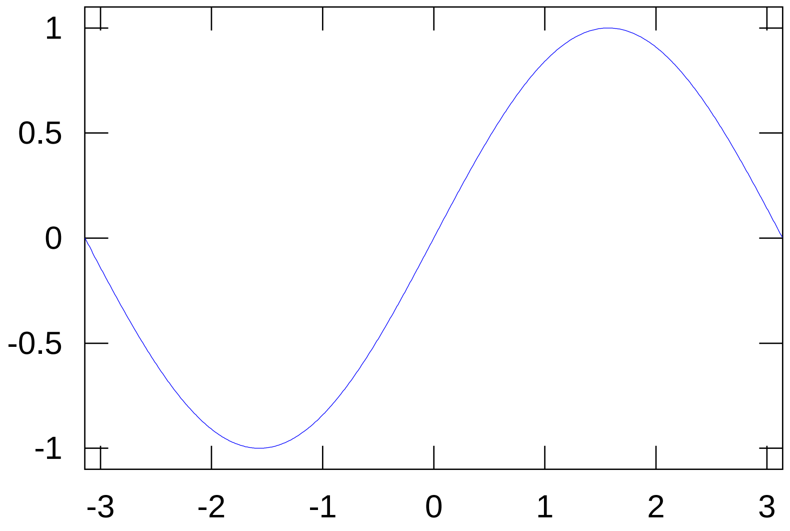
```
octave-3.2.3:113> x=[0:0.1:2*pi];
octave-3.2.3:114> plot(x,sin(x))
octave-3.2.3:115> text(pi/2,1.1,'maximum')
octave-3.2.3:116> text(3*pi/2,-1.1,'minimum')
```



Miscellaneous

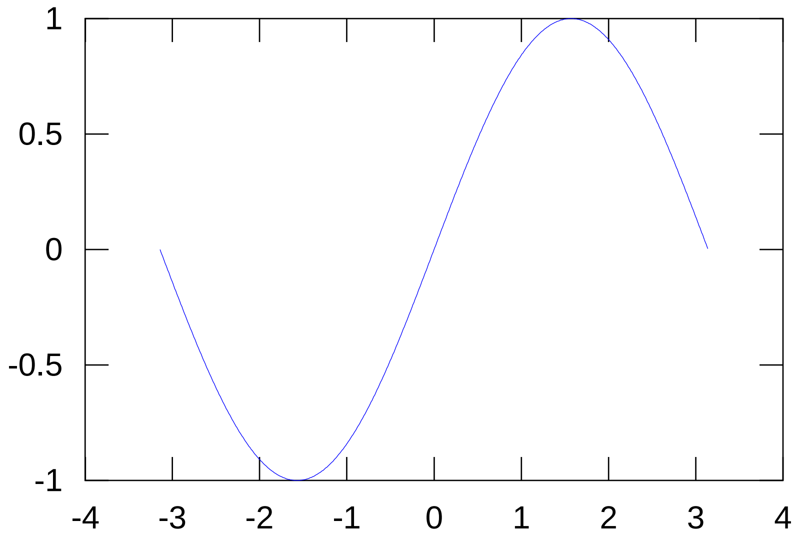
You might want to adjust the viewing region of a plot. Use `axis([xmin,xmax,ymin,ymax])`. Example:

```
octave-3.2.3:151> x=[-pi:0.01:pi];
octave-3.2.3:152> plot(x,sin(x))
octave-3.2.3:153> axis([-pi,pi,-1.1,1.1])
```



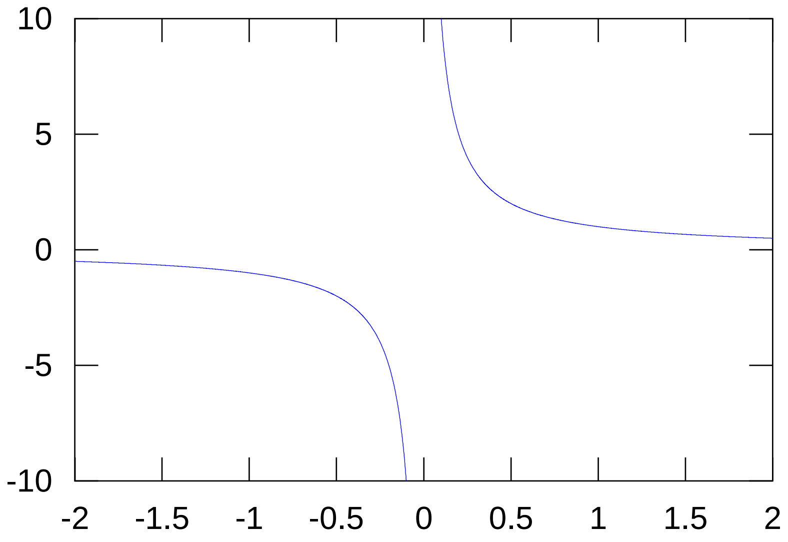
Without the `axis` command the plot would have looked like:

```
octave-3.2.3:151> x=[-pi:0.01:pi];
octave-3.2.3:152> plot(x,sin(x))
```



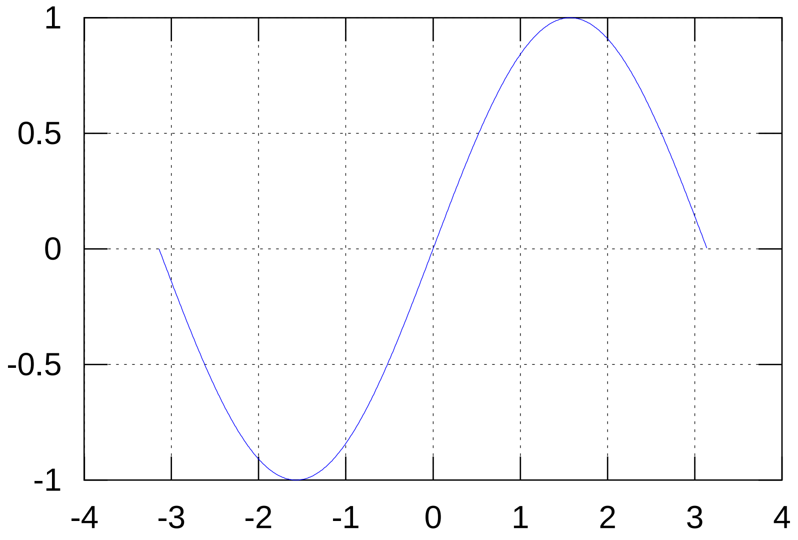
This is especially helpful for plotting functions that have singularities, like $1/x$

```
octave-3.2.3:151> x=[-2:0.001:2];
octave-3.2.3:152> plot(x,1./x)
octave-3.2.3:153> axis([-2,2,-10,10])
```



You might want to add a grid to the background of a plot. Use `grid`. Example:

```
octave-3.2.3:151> x=[-pi:0.01:pi];
octave-3.2.3:152> plot(x,sin(x))
octave-3.2.3:153> grid
```



Getting help

Octave has a help facility. If you enter `help plot`, you will see a help page for the plot command. The help pages can be a bit cryptic to read sometimes, but they can point you in the right direction. If a help page has more than one screen, you use the space-bar or 'f' to advance to the next page ('b' is used to go back, and 'q' is used to stop reading the help page.)

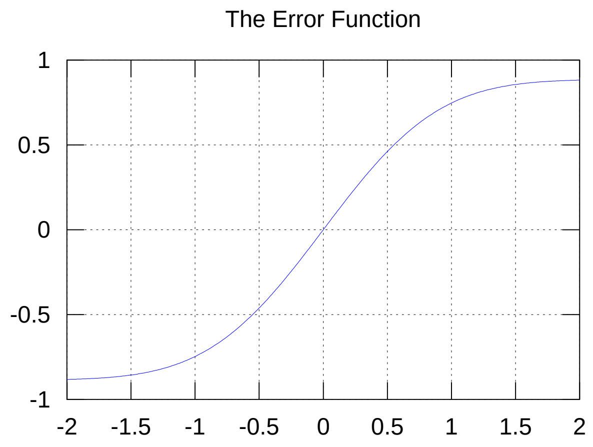
The Octave plotting commands are related to Matlab plotting commands. So you can use Google to search for `matlab plot` and find a Matlab help page as well. Keep in mind, however, that Matlab's plotting facility is much more advanced than Octave's, so not everything you find will work.

Making Scripts

A script in Octave is a sequence of commands to be run in order. For example, we might want to make a nice graph of the function $F(x) = \int_0^x e^{-s^2}\,ds$ over the interval $-2 \le x \le 2$. The commands are:

```
f = @(x) exp(-x.^2);
F = @(x) quad(f,0,x);
x=-2:0.01:2;
plot(x,arrayfun(F,x));
title('The Error Function')
grid
```

If you save these commands in a script you can make changes to them later if, for example, want to adjust the title or change the domain. To do this, you'll need a text editor. On Windows, Notepad will do. On the Mac, you can use TextEdit. Open your application and enter the commands above. If you are using the TextEdit, you'll also need to select "Make Plain Text" from the "Format" menu (or change this preference permanently under "Preferences"). Now save the document as `myplot.m` in a location that Octave knows about (via its path). The directory where you saved your file `df.m` will work, for example. Now within Octave you can simply type `myplot` and your plot will appear. If you adjust the commands in the file `myplot.m` and rerun `myplot` in Octave, a new plot will appear. The plot from these commands looks as follows:



Exercises

In the subsequent exercises, $x = [7, 1, 8, 2]$ and $y = [4, 9, 8, -11]$. For exercises 1 through 5, you should copy your input and Octave's output into a a word processing program or text editor (such as Notepad on Windows and TextEdit on the Mac) and print it out.

Exercise 1:

Enter all of the commands in this tutorial into Octave. Really!

Exercise 2:

Use Octave to compute $x + y$.

Exercise 3:

Use Octave to compute $-3 * x$.

Exercise 4:

Use Octave to compute a vector that contains the square root of all the entries of x. Hint: square root is the $1/2$ power.

Exercise 5:

Let $p(t) = -1 + 3t - 2t^3$ – that is, p is a polynomial. Use Octave to compute the value of p at each of the entries of x. The first entry of this matrix should be $p(7)$ since the first entry of x is 7. The last entry should be $p(2)$ since 2 is the last entry of x.

Exercise 6:

Use Octave to plot $p(t) = -1 + 3t - 2t^3$ over the interval $0 \leq t \leq 2$. Hand in a printout of your graph.

Exercise 7:

Plot the curves $y = Ce^x$ for $C = 1$, $C = 1/2$, $C = 0$, $C = -1/2$, and $C = -1$ over the range $-1 \leq x \leq 1$ all in the same figure. Add a helpful legend to your plot. Hand in a printout of your plot.

Exercise 8:

Use Octave to generate a nice plot for your work in Section 1.2, problem 15. Hand in a printout of your plot.

Exercise 9:

Let $\text{logistic}(x) = \dfrac{1}{1 + e^{-x}}$.

a. Following the example from Section , use Octave to define a function `logisitic` for this function.

b. Verify that your function works correctly by computing `logistic(0)`, `logisitic(1)`, and `logistic([0,1])`. Do you obtain the right answers? (Hint: if you have a error when you test with vector input, think about the dot operators `.*`, `./` and so forth.)

c. Plot the `logistic` function over the range $-2 \le x \le 2$. Add a red square or diamond that marks the point $(1, \text{logistic}(1))$.

Hand in a transcript of the Octave commands you used in parts a) through c) as well as a printout of your plot.