## Variables, Vectors, Arithmetic

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

```
>> x=3;
>> y=5;
>> 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`. The last line doesn't end in a semicolon so that we can see what the result of the computation is.

MATLAB knows about functions like the ones found on your calculator, you just need to know how to type their names.

```
>> sqrt(2)

ans =

    1.4142

>> sin(pi/4)

ans =

    0.7071

>> sqrt(2)/2

ans =

    0.7071
```

By default, MATLAB doesn't display a lot of digits, but if you want you can see more.

```
>> format long
>> 1/sqrt(2)

ans =

   0.707106781186547

>> sin(pi/4)
```

```
ans =

   0.707106781186547

>> format short
>> 1/sqrt(2)

ans =

    0.7071
```

Notice that MATLAB knows about the constant $\pi$. It's name is `pi`. The trigonometric functions all take arguments in radians. If you want degrees, there's a different function `sind`, for example. Or you can just convert from degrees to radians explicitly, which is the standard approach.

```
>> sin(pi/4)

ans =

    0.7071

>> sind(45)

ans =

    0.7071

>> sin(45 * (2*pi)/360)

ans =

    0.7071
```

## Arrays

MATLAB shines at doing computations with arrays of numbers. The first kind of array is called a vector and is just a list of numbers.

```
>> x=[ 3, 1, 7]
x =

   3   1   7
```

MATLAB knows about two different kinds of vectors. The example above is a **row** vector. It's entries are arranged from left to right. The other kind of vector is a **column** vector, which has entries arranged from top to bottom.

```
>> y=[2 ; 9]
y =

   2
   9
```

The only difference when entering them is the comma versus the semicolon. You can also skip the commas and get a row vector by default.

```
>> x = [3 1 7]

x =

     3     1     7
```

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

```
>> x=[3,1,7]

x =

   3   1   7

>> y=[2,9,8]

y =

   2   9   8

>> 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

```
>> z(1)

ans =

     5

>> z(2)

ans =

    10

>> z(3)

ans =

    15
```

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

```
>> 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.

```
>> w=z+z

w =

    10   20   30
```

You can also multiply a vector by a number.

```
>> x=[3,1,7]

x =

   3   1   7

>> 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.

```
>> A = [ 4, 9, 8; 1, -3, 5]

A =

   4   9   8
   1  -3   5
```

We say that `A` is a $2 \times 3$ 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 $1 \times 5$ matrix, and a column vector with 5 entries is a $5 \times 1$ matrix.

There is a notion of matrix multiplication that we'll talk about later in the class and it's very different from anything you might have guessed. MATLAB uses matrix multiplication by default, and this is the reason for the error message below.

```
>> x = [1, 5, 2]

x =

   1   5   2

>> y = [3, 4, 9]

y =

   3   4   9

>> x*y

Error using  *
Incorrect dimensions for matrix multiplication. Check that the number of columns in t
number of rows in the second matrix. To perform elementwise multiplication, use '.*'.

>> x*z

ans =  41
```

The error occurs because you can't multiply a $1 \times 3$ matrix by a $1 \times 3$ matrix using the rules of matrix multiplication. You don't need to know what those rules are, but you do need to be able to recover from an error message like this. Happily, the error message tells you what the fix is! If you mean to just multiply the entries of the two vectors term by term, use `.*` with a period in the front.

For example:

```
>> x = [2, 5, 2]
x =

   2   5   2

>> y = [3, 4, 9]
y =

   3   4   9

>> 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).

```
>> 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).

```
>> x.^2
ans =

    4   25    4
```

## Basic Plotting

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

```
>> 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:

```
>> x=[0:0.05:1];
>> 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:

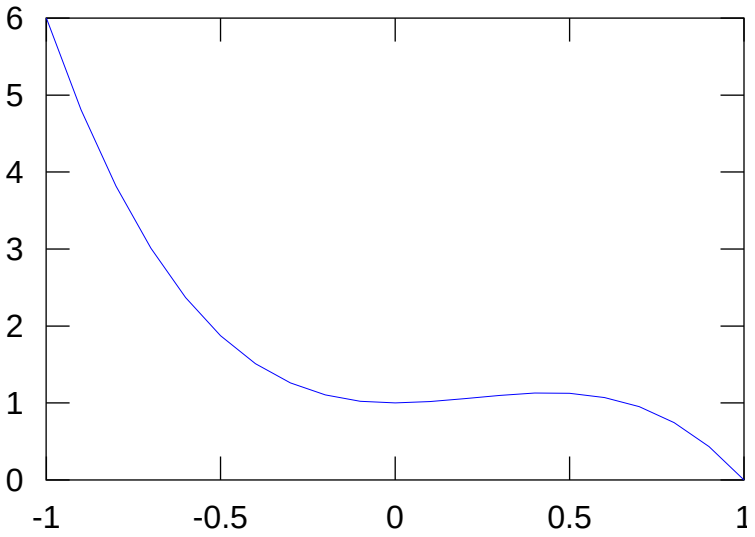
```
>> x=[-1:0.1:1];
```

Then we make a vector for the $y$ values:

```
>> y = 1 + 2*x.^2 - 3*x.^3;
```

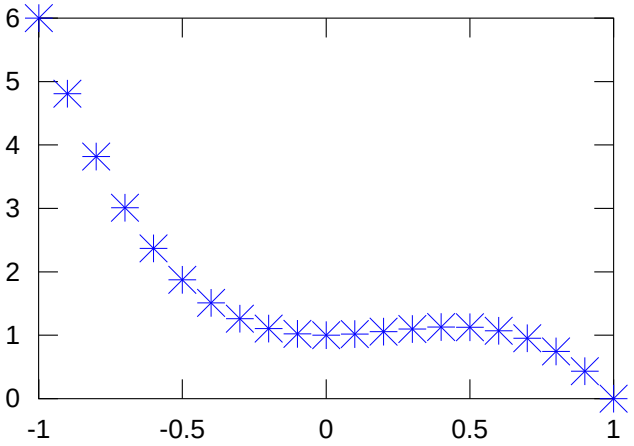
Finally, we plot these $x$ and $y$ coordinate points with

```
>> plot(x,y)
```




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

```
>> plot(x,y,'*')
```

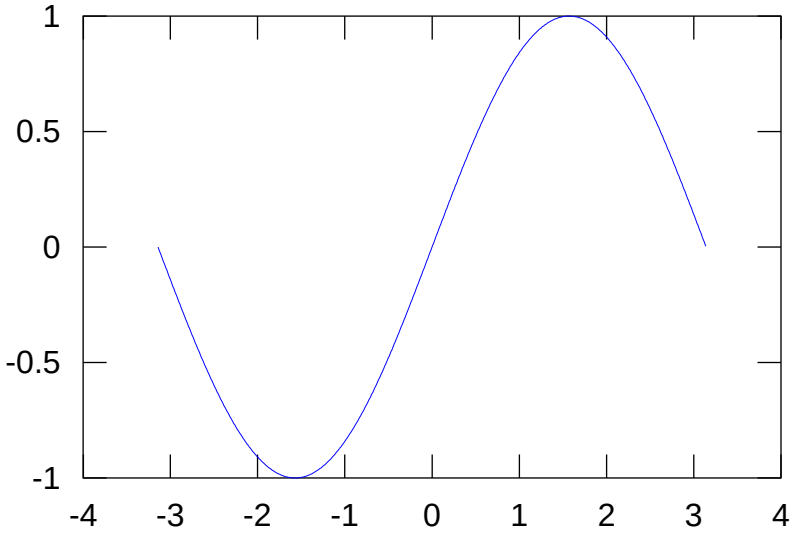



In this next example, we make a plot of the function $\sin(x)$ over the range $-\pi \leq x \leq \pi$.

```
>> x=[-pi:0.01:pi];
```

MATLAB'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.

```
>> y=sin(x);
>> plot(x,y)
```




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

| MATLAB 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) |

# Getting Help

MATLAB has a basic 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.

MATLAB has a more sophisticated Documentation library as well. You can find it under the Help menu.

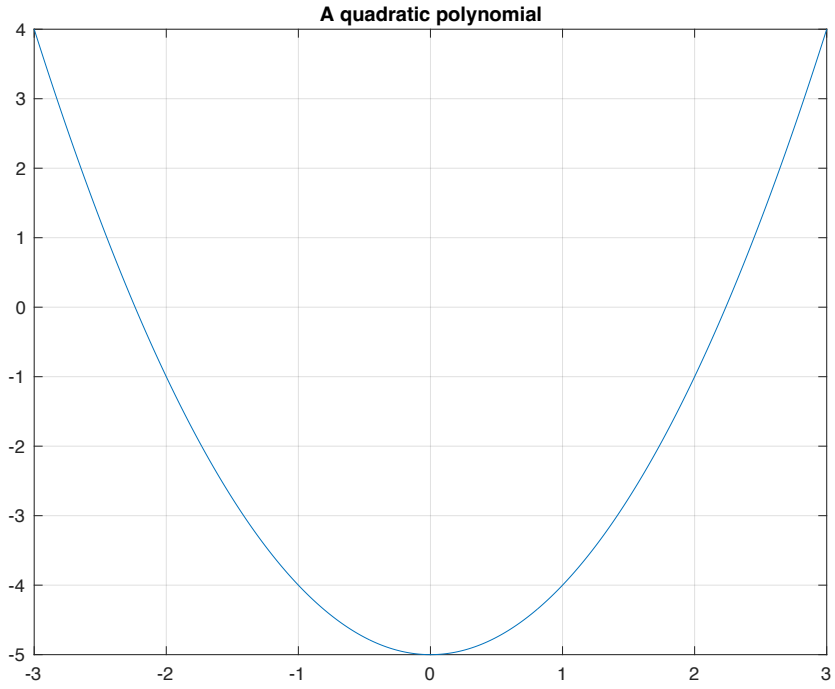
Google can also be used to find help on MATLAB problems. Try, for example, searching for `matlab sin degrees`.

# Making Scripts

A script in MATLAB is a sequence of commands to be run in order. For example, we might want to make a nice graph of the function $F(x) = x^2 - 5$ over the interval $-3 \leq x \leq 3$. The commands are:

```
x = linspace(-3,3,0.01);
y = x.^2-5;
plot(x,y);
title('A quadratic polynomial')
xlabel('x')
ylabel('y')
grid
```

Don't worry about the commands here that you don't know yet, such as `title`; these are explained in the next section. For now, just notice that there's a lot of words needed to make the following figure.




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. MATLAB has a built in text editor where you can edit scripts and run them. Your TA will demonstrate these features.

You will make MATLAB scripts for your homework, and there is a sample solution script on the course website. Here's part of it.

```
% Homework 1 Demo Solutions
```

```
% Gilat 1.10 #1
disp('----------------------')
disp('Problem 1.10 #1')

disp('Part (a)')
(5-19/7+2.5^3)^2

disp('Part (b)')
7*3.1+sqrt(120)/5-15^(5/3)
```

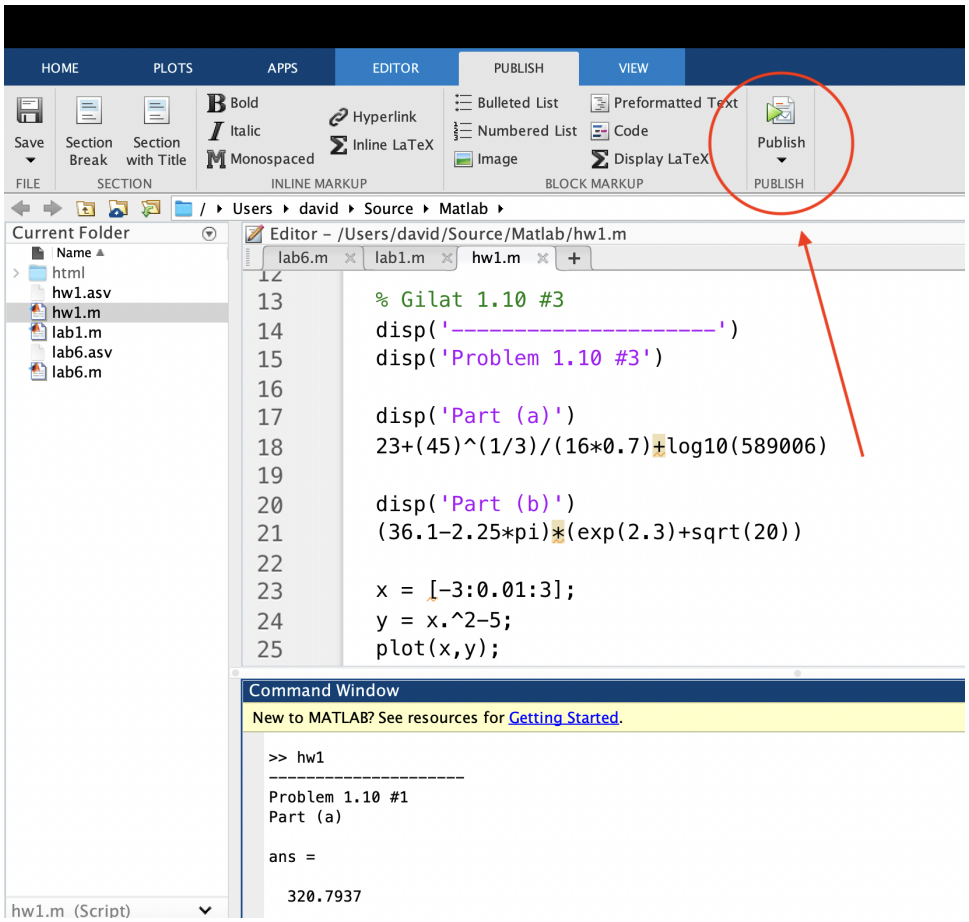
The lines starting with % are comments. They are ignored by MATLAB and are meant for human readers. Documenting your work is an important skill. The `disp` commands are used to display text information in your output. If you run this script, the output is

```
----------------------
Problem 1.10 #1
Part (a)

ans =

  320.7937

Part (b)

ans =

  -67.3421
```

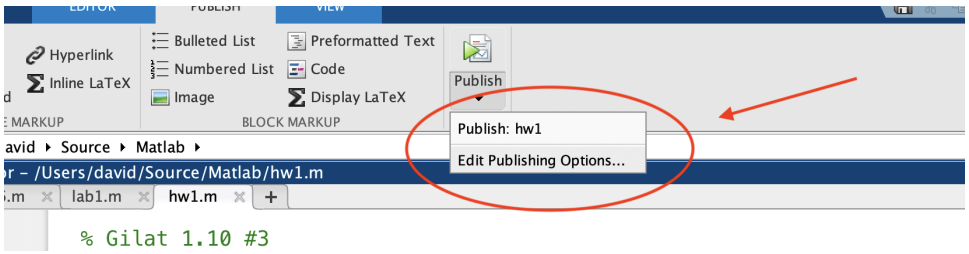
Notice how `disp` is used to annotate the output.

## Publishing Scripts

MATLAB has a Publish facility used for combining the input and the output of a script, including any plots, in a single document. Use the Publish tab and the green Publish button.




By default, MATLAB creates an HTML file (a web page). You'll want to change the settings so that it creates a PDF file instead. You can access these settings like this:

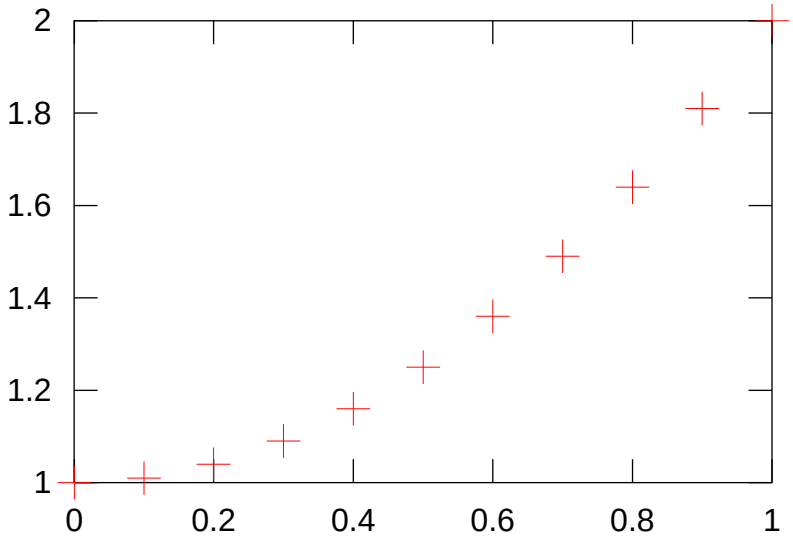



# More on plots

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.

```
>> x=[0:0.1:1];
>> y=1+x.^2;
>> plot(x,y,'r+')
```

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



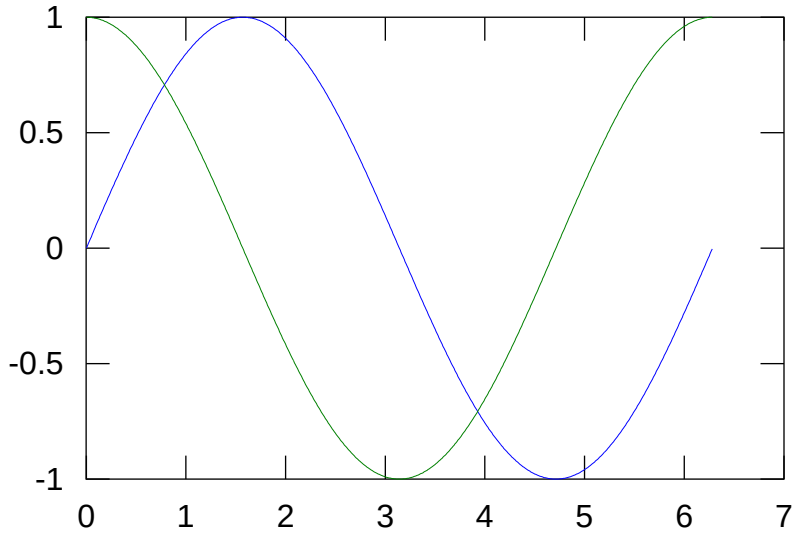

Some color styles: `r` (red), `g` (green), `b` (blue), `c` (cyan), `m` (magenta), `k` (black).

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.

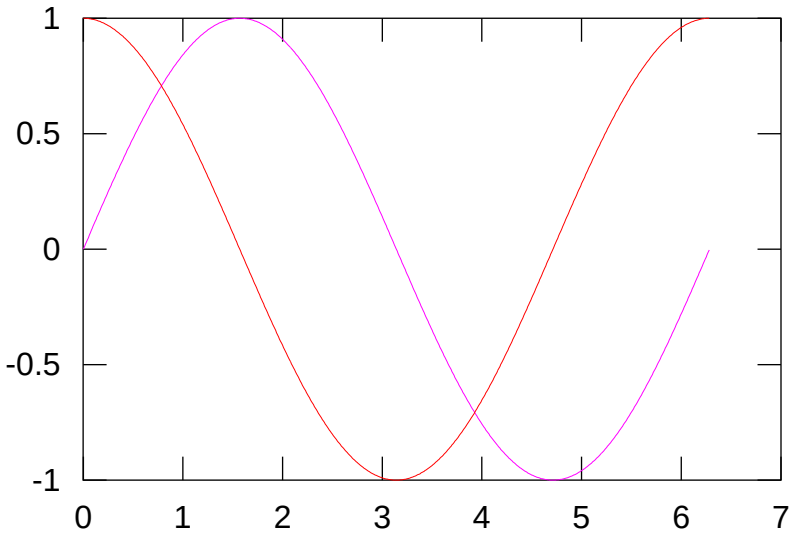
```
>> x=[0:0.01:2*pi];
>> plot(x,sin(x),x,cos(x))
```




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

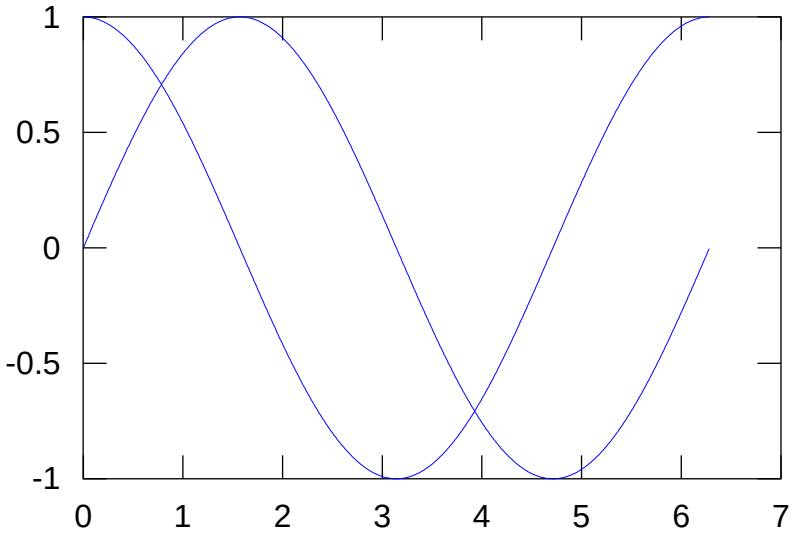
```
>> x=[0:0.01:2*pi];
```

```
>> 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.

```
>> x=[0:0.01:2*pi];
>> plot(x,sin(x))
>> hold on
>> plot(x,cos(x))
>> hold off
```

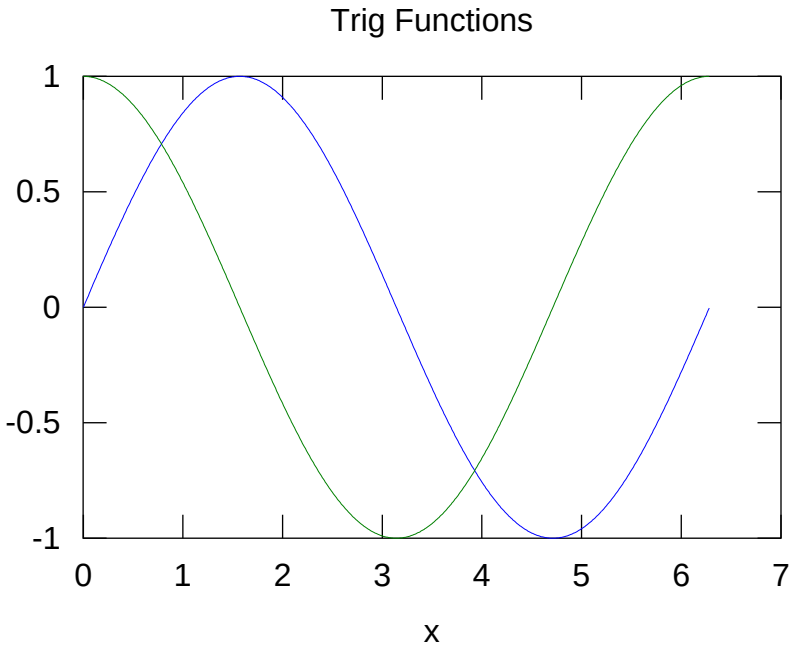



Everything you plot between `hold on` and `hold off` will appear in the same figure.

## Labels

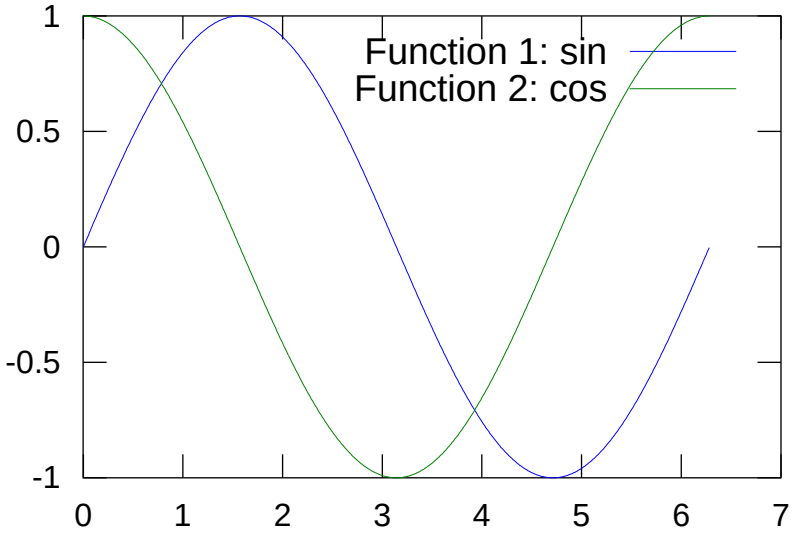
It can be helpful to add some text labels to a plot. Here are some examples:

```
>> plot(x,sin(x),x,cos(x))
>> title('Trig Functions')
>> ylabel('y')
>> xlabel('x')
```




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

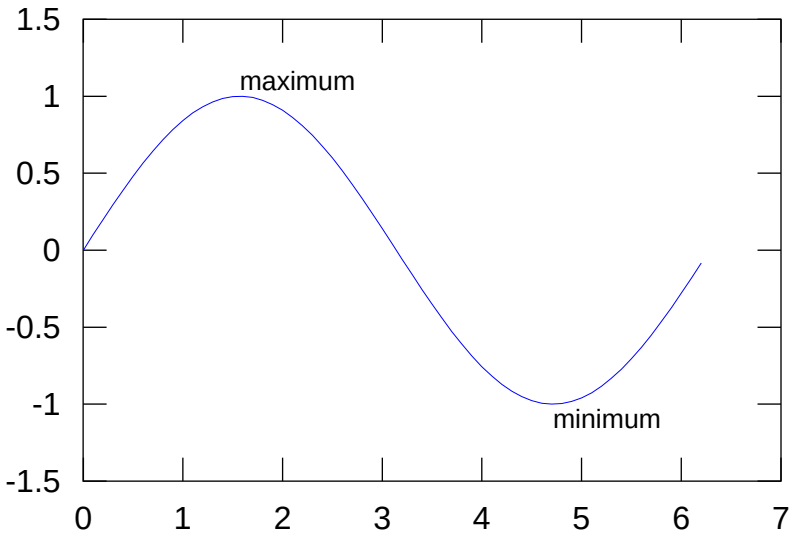
```
>> plot(x,sin(x),x,cos(x))
>> legend('Function 1: sin','Function 2: cos')
```




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

```
>> x=[0:0.1:2*pi];
```

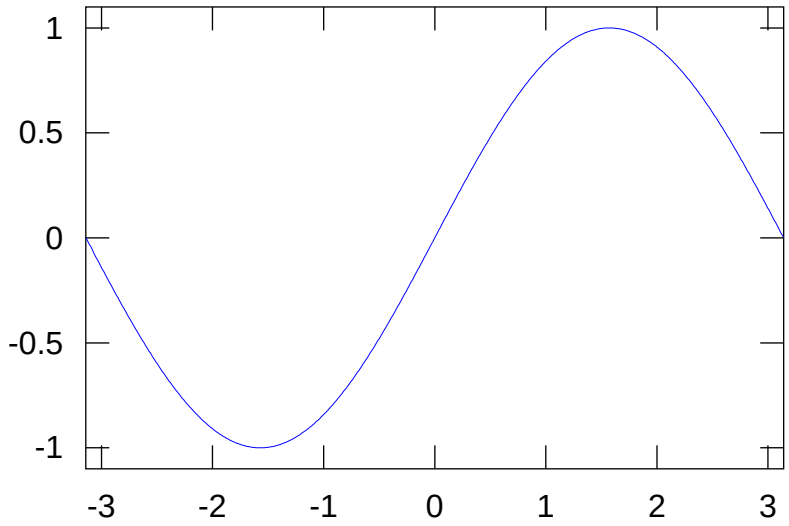
```
>> plot(x,sin(x))
>> text(pi/2,1.1,'maximum')
>> text(3*pi/2,-1.1,'minimum')
```




## Miscellaneous

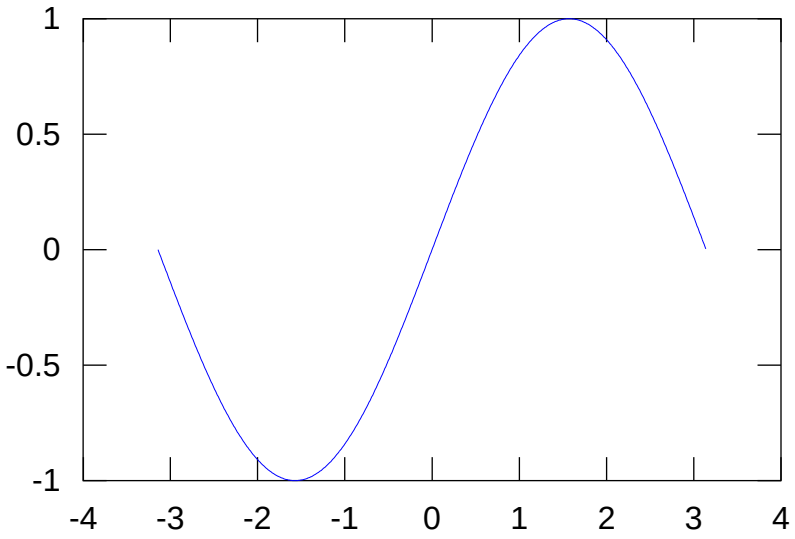
You might want to adjust the viewing region of a plot. Use `axis([xmin,xmax,ymin,ymax])`. Example:

```
>> x=[-pi:0.01:pi];
>> plot(x,sin(x))
>> axis([-pi,pi,-1.1,1.1])
```



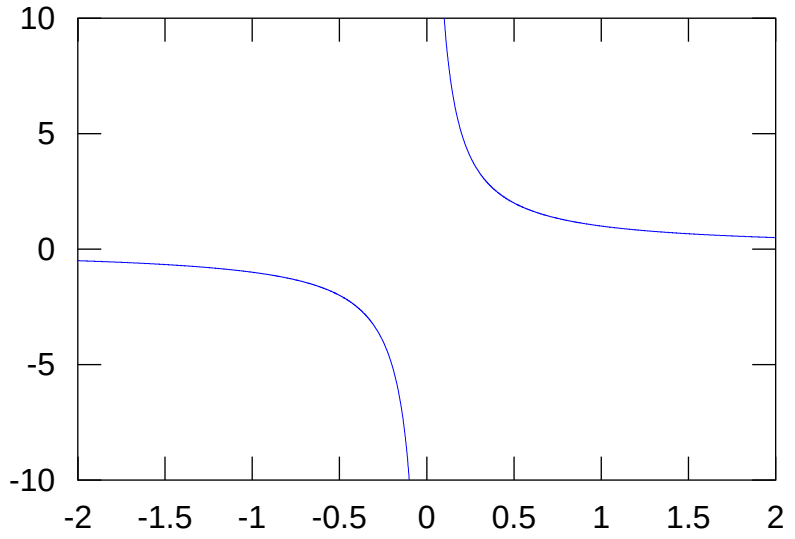

Without the `axis` command the plot would have looked like:

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



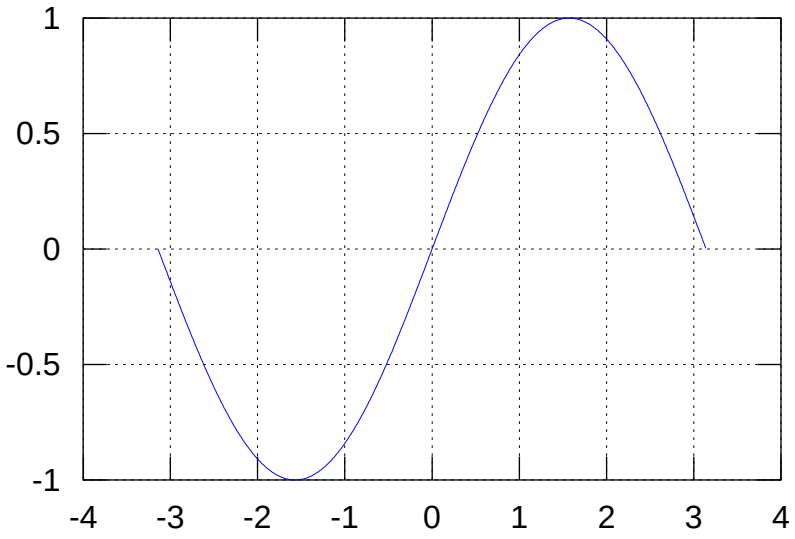

This is especially helpful for plotting functions that have singularities, like $1/x$

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




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

```
>> x=[-pi:0.01:pi];
>> plot(x,sin(x))
>> grid
```


1
0.5
0
-0.5
-1
-4
-3
-2
-1
0
1
2
3
4