

## Matlab basic commands list

---

### Basic commands

1. `clc`: clear screen
2. `clear all`: clear all variables from workspace
3. `x=4`; assign value to the variable
4. `x=1:10`; assign an array `x` with values from 1 to 10;
5. `x=1:.5:10`; assign values from 1 to 10 with gap of .5
6. `y=x^2`; using one variable to assign the value in another
7. `sin(x)`; in radian
8. `sind(x)`; in degree
9. `asin(x)`; inverse of sine
10. `sinh(x)`; hyperbole of sine
11. `exp(x)`; exponential of `x`
12. `log(x)`; log of `x`
13. `log10(x)`; log with base 10
14. `log2(x)`; log with base 2
15. `sqrt(x)`; square root of `x`
16. `min(x)`; minimum value in array `x`
17. `max(x)`; maximum value in array `x`
18. `mean(x)`; mean of `x`
19. `mode(x)`; to find mode of array `x`
20. `median(x)`; median of `x`
21. `var(x)`; variance of `x`
22. `2+3*j`; complex number
23. `abs(x)`; absolute value of `x`
24. `imag(x)`; imaginary part of `x`
25. `isreal(x)`; check if number is real
26. `factor(x)`; to find prime factor of `x`
27. `factorial(x)`; factorial of `x`
28. `gcd(x)`; gcd of elements in array `x`
29. `lcm(x)`; lcm of elements in array `x`
30. `size(x)`; size of `x`
31. `rand(4)`; generate random numbers array

## Some links:

[Vectors](#)

[Functions](#)

[Plotting](#)

[Polynomials](#)

[Matrices](#)

[Printing](#)

[Using M-files in Matlab](#)

[Getting help in Matlab](#)

---

Matlab is an interactive program for numerical computation and data visualization; it is used extensively by control engineers for analysis and design. There are many different toolboxes available which extend the basic functions of Matlab into different application areas; in these tutorials, we will make extensive use of the Control Systems Toolbox. Matlab is supported on Unix, Macintosh, and Windows environments; a student version of Matlab is available for personal computers. For more information on Matlab, contact the [Mathworks](#).

The idea behind these tutorials is that you can view them in one window while running Matlab in another window. You should be able to re-do all of the plots and calculations in the tutorials by cutting and pasting text from the tutorials into Matlab or an m-file.

### Vectors

Let's start off by creating something simple, like a vector. Enter each element of the vector (separated by a space) between brackets, and set it equal to a variable. For example, to create the vector a, enter into the Matlab command window (you can "copy" and "paste" from your browser into Matlab to make it easy):

```
a = [1 2 3 4 5 6 9 8 7]
```

Matlab should return:

```
a =  
1 2 3 4 5 6 9 8 7
```

Let's say you want to create a vector with elements between 0 and 20 evenly spaced in increments of 2 (this method is frequently used to create a time vector):

```
t = 0:2:20  
t =  
0 2 4 6 8 10 12 14 16 18 20
```

Manipulating vectors is almost as easy as creating them. First, suppose you would like to add 2 to each of the elements in vector 'a'. The equation for that looks like:

```
b = a + 2
```

```
b =  
3 4 5 6 7 8 11 10 9
```

Now suppose, you would like to add two vectors together. If the two vectors are the same length, it is easy. Simply add the two as shown below:

```
c = a + b
```

```
c =  
4 6 8 10 12 14 20 18 16
```

Subtraction of vectors of the same length works exactly the same way.

## Functions

To make life easier, Matlab includes many standard functions. Each function is a block of code that accomplishes a specific task. Matlab contains all of the standard functions such as sin, cos, log, exp, sqrt, as well as many others. Commonly used constants such as pi, and i or j for the square root of -1, are also incorporated into Matlab.

```
sin(pi/4)
```

```
ans =
```

```
0.7071
```

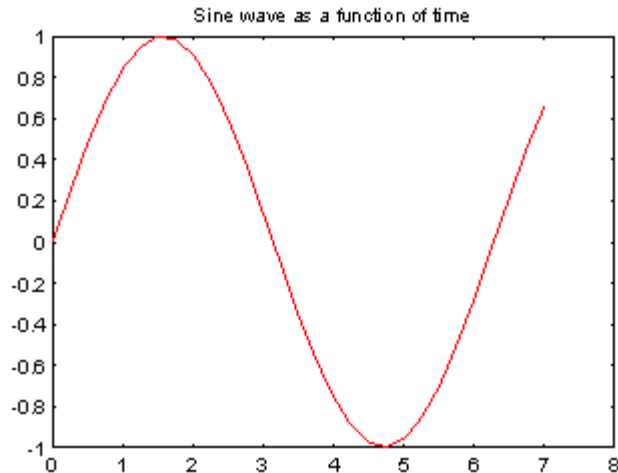
To determine the usage of any function, type help [function name] at the Matlab command window.

Matlab even allows you to write your own functions with the function command; follow the link to learn how to write your own functions and see a listing of the functions we created for this tutorial.

## Plotting

It is also easy to create plots in Matlab. Suppose you wanted to plot a sine wave as a function of time. First make a time vector (the semicolon after each statement tells Matlab we don't want to see all the values) and then compute the sin value at each time.

```
t=0:0.25:7;  
y = sin(t);  
plot(t,y)
```



The plot contains approximately one period of a sine wave. Basic plotting is very easy in Matlab, and the plot command has extensive add-on capabilities. I would recommend you visit the [plotting](#) page to learn more about it.

## Polynomials

In Matlab, a polynomial is represented by a vector. To create a polynomial in Matlab, simply enter each coefficient of the polynomial into the vector in descending order. For instance, let's say you have the following polynomial:

$$s^4 + 3s^3 - 15s^2 - 2s + 9$$

To enter this into Matlab, just enter it as a vector in the following manner

$$x = [1 \ 3 \ -15 \ -2 \ 9]$$

$$x = \\ 1 \ 3 \ -15 \ -2 \ 9$$

Matlab can interpret a vector of length n+1 as an nth order polynomial. Thus, if your polynomial is missing any coefficients, you must enter zeros in the appropriate place in the vector. For example,

$$s^4 + 1$$

would be represented in Matlab as:

$$y = [1 \ 0 \ 0 \ 0 \ 1]$$

You can find the value of a polynomial using the polyval function. For example, to find the value of the above polynomial at s=2,

$$z = \text{polyval}([1 \ 0 \ 0 \ 0 \ 1], 2)$$

$$z = \\ 17$$

You can also extract the roots of a polynomial. This is useful when you have a high-order polynomial such as

$$s^4 + 3s^3 - 15s^2 - 2s + 9$$

Finding the roots would be as easy as entering the following command;

$$\text{roots}([1 \ 3 \ -15 \ -2 \ 9])$$

$$\text{ans} = \\ -5.5745 \\ 2.5836 \\ -0.7951 \\ 0.7860$$

Let's say you want to multiply two polynomials together. The product of two polynomials is found by taking the convolution of their coefficients. Matlab's function `conv` that will do this for you.

```
x = [1 2];  
y = [1 4 8];  
z = conv(x,y)
```

```
z =  
    1  6 16 16
```

Dividing two polynomials is just as easy. The `deconv` function will return the remainder as well as the result. Let's divide `z` by `y` and see if we get `x`.

```
[xx, R] = deconv(z,y)
```

```
xx =  
    1  2  
R =  
    0  0  0  0
```

As you can see, this is just the polynomial/vector `x` from before. If `y` had not gone into `z` evenly, the remainder vector would have been something other than zero.

If you want to add two polynomials together which have the same order, a simple `z=x+y` will work (the vectors `x` and `y` must have the same length). In the general case, the user-defined function, `polyadd` can be used. To use `polyadd`, copy the function into an m-file, and then use it just as you would any other function in the Matlab toolbox. Assuming you had the `polyadd` function stored as a m-file, and you wanted to add

the two uneven polynomials, x and y, you could accomplish this by entering the command:

```
z = polyadd(x,y)
```

```
x =  
  1  2
```

```
y =  
  1  4  8
```

```
z =  
  1  5 10
```

## Matrices

Entering matrices into Matlab is the same as entering a vector, except each row of elements is separated by a semicolon (;) or a return:

```
B = [1 2 3 4;5 6 7 8;9 10 11 12]
```

```
B =  
  1  2  3  4  
  5  6  7  8  
  9 10 11 12
```

```
B = [ 1 2 3 4  
     5 6 7 8  
     9 10 11 12]
```

```
B =  
  1  2  3  4
```



```
5 6 7 8
9 10 11 12
```

Matrices in Matlab can be manipulated in many ways. For one, you can find the transpose of a matrix using the apostrophe key:

```
C = B'
```

```
C =
 1 5 9
 2 6 10
 3 7 11
 4 8 12
```

It should be noted that if C had been complex, the apostrophe would have actually given the complex conjugate transpose. To get the transpose, use .' (the two commands are the same if the matrix is not complex).

Now you can multiply the two matrices B and C together. Remember that order matters when multiplying matrices.

```
D = B * C
```

```
D =
 30 70 110
 70 174 278
 110 278 446
```

```
D = C * B
```

```
D =
 107 122 137 152
 122 140 158 176
```

137 158 179 200  
152 176 200 224

Another option for matrix manipulation is that you can multiply the corresponding elements of two matrices using the .\* operator (the matrices must be the same size to do this).

E = [1 2;3 4]  
F = [2 3;4 5]  
G = E .\* F

E =  
1 2  
3 4

F =  
2 3  
4 5

G =  
2 6  
12 20

If you have a square matrix, like E, you can also multiply it by itself as many times as you like by raising it to a given power.

E^3

ans =  
37 54  
81 118

If wanted to cube each element in the matrix, just use the element-by-element cubing.

```
E.^3
```

```
ans =
```

```
1 8  
27 64
```

You can also find the inverse of a matrix:

```
X = inv(E)
```

```
X =  
-2.0000 1.0000  
1.5000 -0.5000
```

or its eigenvalues:

```
eig(E)
```

```
ans =  
-0.3723  
5.3723
```

There is even a function to find the coefficients of the characteristic polynomial of a matrix. The "poly" function creates a vector that includes the coefficients of the characteristic polynomial.

```
p = poly(E)
```

```
p =
```

*1.0000 -5.0000 -2.0000*

Remember that the eigenvalues of a matrix are the same as the roots of its characteristic polynomial:

roots(p)

*ans =*

*5.3723*

*-0.3723*

## Printing

Printing in Matlab is pretty easy. Just follow the steps illustrated below:

### Macintosh

To print a plot or a m-file from a Macintosh, just click on the plot or m-file, select Print under the File menu, and hit return.

### Windows

To print a plot or a m-file from a computer running Windows, just select Print from the File menu in the window of the plot or m-file, and hit return.

### Unix

To print a plot on a Unix workstation enter the command:

`print -P<printername>`

If you want to save the plot and print it later, enter the command:

```
print plot.ps
```

Sometime later, you could print the plot using the command "lpr -P plot.ps" If you are using a HP workstation to print, you would instead use the command "lpr -d plot.ps"

To print a m-file, just print it the way you would any other file, using the command "lpr -P <name of m-file>.m" If you are using a HP workstation to print, you would instead use the command "lpr -d plot.ps<name of m-file>.m"

## Using M-files in Matlab

There are slightly different things you need to know for each platform.  
Macintosh

There is a built-in editor for m-files; choose "New M-file" from the File menu. You can also use any other editor you like (but be sure to save the files in text format and load them when you start Matlab).

## Windows

Running Matlab from Windows is very similar to running it on a Macintosh. However, you need to know that your m-file will be saved in the clipboard. Therefore, you must make sure that it is saved as filename.m

## Unix

You will need to run an editor separately from Matlab. The best strategy is to make a directory for all your m-files, then cd to that directory before running both Matlab and the editor. To start Matlab from your Xterm window, simply type: matlab.

You can either type commands directly into matlab, or put all of the commands that you will need together in an m-file, and just run the file.

If you put all of your m-files in the same directory that you run matlab from, then matlab will always find them.

## Getting help in Matlab

Matlab has a fairly good on-line help; type

help command name

for more information on any given command. You do need to know the name of the command that you are looking for; a list of the all the ones used in these tutorials is given in the [command listing](#); a link to this page can be found at the bottom of every tutorial and example page.

Here are a few notes to end this tutorial.

You can get the value of a particular variable at any time by typing its name.

B

```
B =  
    1  2  3  
    4  5  6  
    7  8  9
```

You can also have more than one statement on a single line, so long as you separate them with either a semicolon or comma.

Also, you may have noticed that so long as you don't assign a variable a specific operation or result, Matlab will store it in a temporary variable called "ans".

---

## User feedback

We would like to hear about difficulties you had with the tutorials, suggestions you have for improvement, errors that you found, or any

other comments that you have. This feedback is anonymous; include your email address if you want a reply.

## Some Useful MATLAB Commands

---

First, the most important command in MATLAB:

>> *help command\_name*

Displays complete information about any command. Take note of any cross references at the end of each help entry. *help* with no command specified lists all categories of available commands. Note: *helpwin* brings up a separate gui help window -- useful if you're doing other things in the main window.

Interacting with MATLAB:

>> *command*  
>> *command*;

A semicolon after any command suppresses output from that

	command to the MATLAB window (but not, for example, to a figure) - especially useful if the output is a very long vector.
>> <UP ARROW>	Recalls the last command entered. You can repeat it to go farther back, or press <DOWN ARROW> to scroll forward.
>> <i>abc</i> <UP ARROW>	Recalls the last command entered that starts with "abc".
>> more on >> more off	Turns on and off more, which displays all output (including help information) one screen at a time.
>> who	Tells you what variables currently exist.
>> clear >> clear <i>variable</i>	Deletes all variables, or the specified variable.
>> format long g >> format compact	Displays variables (but not systems) with more digits. Does not affect storage or computation. Gets rid of the extra blank lines in



the display.

### Commands for making and formatting plots:

```
>> p1=plot(t1,y1);  
>> delete(p1)
```

Plots y1 vs. t1. Note that setting the plot equal to a name lets you delete it afterwards - useful if you're plotting several things in the same figure and make a mistake. y1 and t1 must be vectors of the same size.

```
>> plot(y(:,1),y(:,2))
```

Plots the second column of matrix y vs. the first column. See note at end of section.

```
>> subplot(m,n,p)
```

Breaks the figure up into mxn separate graphs, and selects the p<sup>th</sup> one as current; if there are already mxn graphs, leaves graphs as they are and selects the p<sup>th</sup> one.

```
>> xlabel('label_text')  
>> ylabel('label_text')  
>> title('title_text')
```

*Labels the axes or the entire subplot with specified text (note single quotes).*

```
>> g1 = gtext('plot_text');  
>> delete(g1)
```

*Lets you place the given text (in single quotes) by clicking on the graph. Note that setting the gtext equal to a name lets you delete it afterwards.*

```
>> grid
```

*Toggles on and off a grid superimposed over the current graph; you can also type grid on and grid off.*

```
>> sgrid
```

Draws a grid on a plot of the s plane (root locus plot or pole and zero locations), which consists of lines of constant damping coefficient (zeta) and natural frequency (wn).

```
>> hold
```

Toggles on and off the plot hold, which adds new plots to any already in the graph (without it, new plots delete previous ones). You can also type hold on and hold off.

```
>> axis([xmin xmax ymin  
ymax])
```

*Sets the limits of the x and y axes manually, or lets them be set*

```
>> axis auto
```

*automatically. There are also many other options available for axis.*

```
>> zoom
```

*Lets you select an area of the plot (using the mouse) to zoom in on. Also, clicking the left mouse button once will zoom in, and clicking the right button will zoom out, by a factor of two. Double clicking the right mouse button returns to the original scale; typing zoom again turns off the zoom function.*

```
>> [x, y] = ginput(n)
```

```
>> [x, y] = ginput
```

Lets you input the coordinates of points on the graph with the mouse; collects n points and stores them in the vectors x and y, or if n is absent, keeps collecting points until you press <ENTER>.

Matrix and vector manipulation commands:

```
>> k=linspace(k1,k2)
```

```
>> k=linspace(k1,k2,n)
```

Returns a vector of 100 (or, if specified, n) points equally spaced between x1 and x2.

>> max(y)		Returns the largest element in the vector y.
>> inv(A)		Returns the inverse of the square, nonsingular matrix A.
>> det(A)		Calculates the determinant of the matrix A.
>> eig(A) >> [V,D] = eig(A)		Returns the eigenvalues of A, or sets V to a matrix containing the eigenvectors of A and sets D to a diagonal matrix containing the corresponding eigenvalues.
>> rank(A)		Returns the rank of any matrix A.
Commands useful in system analysis:		
>> roots(f)		Returns the roots of a polynomial, where f is a vector containing the coefficients of the polynomial.
>> conv(P1,P2)		Multiplies two polynomials (P1 and P2 are vectors containing the coefficients of the polynomials) and returns the resulting

	<p>coefficients. This is actually a convolution of the two vectors, which also works as coefficient multiplication.</p>
<pre>&gt;&gt; sys1=tf(num,den) &gt;&gt; sys2=ss(A,B,C,D)  &gt;&gt; sys1=tf(sys2) &gt;&gt; sys2=ss(sys1)</pre>	<p>Creates a system, as a transfer function or state-space representation. Also converts between two different representations of a system. The zpk (zero/pole/gain) command works similarly.</p>
<pre>&gt;&gt; [R,P,K]=residue(num,den)</pre>	<p>Finds the partial fraction expansion of a function <math>H(s)</math>, where num is a vector containing the coefficients of the numerator, and den of the denominator, of <math>H(s)</math>. Returns the numerators (R) and poles (P) of the partial fractions and the remaining polynomial (K):</p> $H(s) = R_1/(s-P_1) + R_2/(s-P_2) + \dots + R_n/(s-P_n) + K(s).$
<pre>&gt;&gt; sys3=series(sys1,sys2)</pre>	<p>Finds the result of putting Systems 1 and 2 in series, and returns either the resulting transfer function</p>

	<p>numerator and denominator or the resulting state space matrices.</p> <p>Mixing system descriptions will work.</p>
<pre>&gt;&gt; sys3=feedback(sys1,sys2) &gt;&gt; sys2=feedback(sys1,tf(1,1))</pre>	<p>Finds the result of adding System 2 as a feedback loop to System 1, assuming a negative feedback, and returns either the resulting transfer function numerator and denominator or the resulting state space matrices.</p>
<pre>&gt;&gt; impulse(sys) &gt;&gt; step(sys)</pre>	<p>Plots the impulse response or step response of the given system.</p> <p>Useful trick: if you have a Laplace transform <math>F(s)</math> of a time function <math>f(t)</math>, plotting the impulse response of <math>F(s)</math> is the same as plotting <math>f(t)</math>.</p>
<pre>&gt;&gt; [y,t,x]=impulse(sys) &gt;&gt; [y,t,x]=step(sys)</pre>	<p>Giving impulse and step output variables returns the output (<math>y</math>), time (<math>t</math>), and states(<math>x</math>) (if state space) vectors, which you can then plot or manipulate.</p>
<pre>&gt;&gt; initial(sys,x0)</pre>	<p>Plots the behavior of the given</p>

	state-space system with initial condition $x_0$ and no input.
<code>&gt;&gt; lsim(sys,u,t,x0)</code>	Plots the response of the given system to the input signal $u(t)$ . The initial condition $x_0$ can be added for state-space systems.
<code>&gt;&gt; bode(sys1,sys2,...)</code> <code>&gt;&gt;</code> <code>[mag,phase,w]=bode(sys,w)</code> <code>&gt;&gt; nyquist(sys)</code> <code>&gt;&gt; {re,im,w}=nyquist(sys,w)</code>	Plots the Bode or Nyquist diagram for the given system(s). A plot is drawn if no return arguments. The frequency points may be specified in the vector $w$ .
<code>&gt;&gt; margin(sys)</code> <code>&gt;&gt;</code> <code>[gm,pm,wcg,wcp]=margin(sys)</code>	Finds the gain margin and phase margin, and the frequencies at which they occur, of the given system. If run with no output arguments, margin also displays the Bode plot, with the margins marked on it and their values displayed in the figure title.
<code>&gt;&gt; rlocus(sys,k)</code>	Plots the root locus for the given system, i.e. where $\text{den}(s) + k \cdot \text{num}(s) = 0$ (or equivalent for the state space form). The vector of

		values for k is optional.
<pre>&gt;&gt; rlocfind(sys) &gt;&gt; [k,poles]=rlocfind(sys)</pre>		Lets you select a pole location from a root locus plot using the mouse, and returns the value of k needed to give such a pole, as well as all the resulting pole locations in the vector poles (if present). rlocfind picks the point on the locus closest to the crosshairs; note that you must already have the root locus graphed to be able to see points you might want to pick.
<pre>&gt;&gt; sys2=canon(sys1,'form')</pre>		Finds a canonical form of the given system; the argument "form" can be either "modal" or "companion" (in single quotes).
<pre>&gt;&gt; rltool</pre>		Opens a tool for designing controllers using the root locus plots.