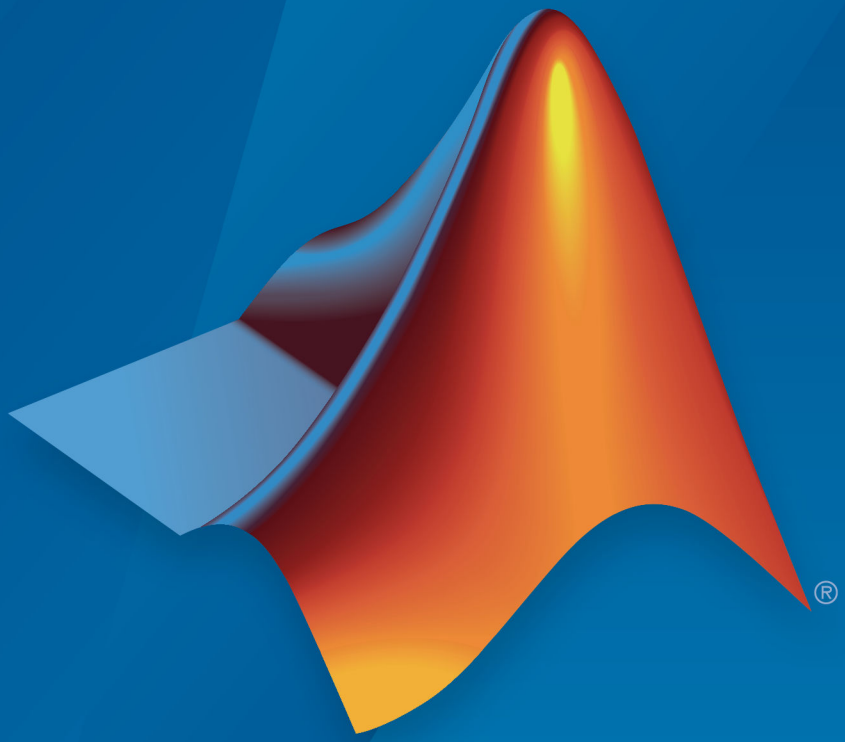
MATLAB®

Primer



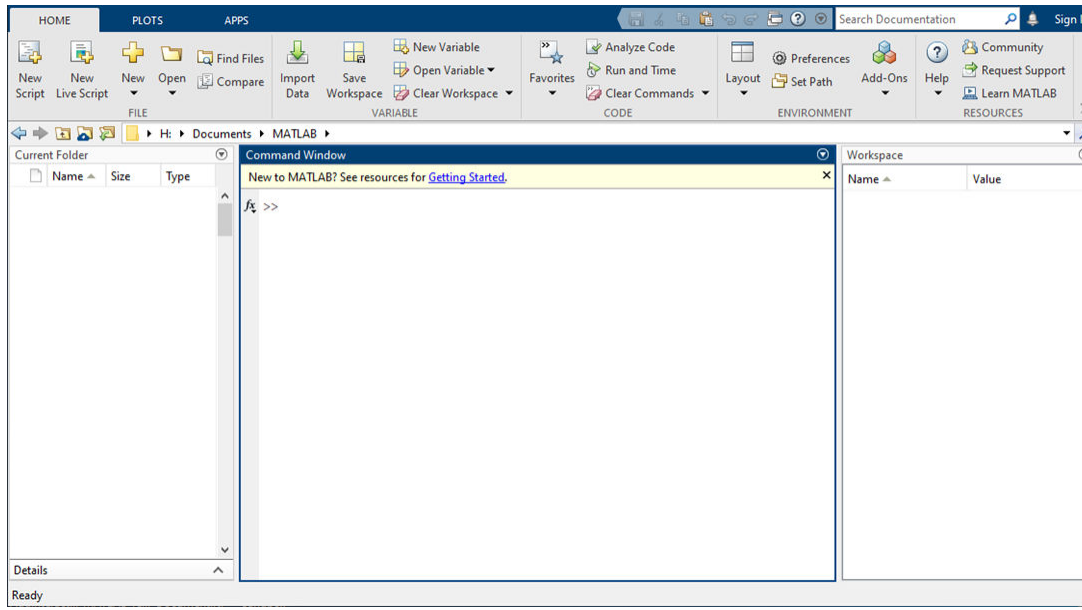
MATLAB®

R2019b

MathWorks®

Desktop Basics

When you start MATLAB, the desktop appears in its default layout.



The desktop includes these panels:

- **Current Folder** — Access your files.
- **Command Window** — Enter commands at the command line, indicated by the prompt (>>).
- **Workspace** — Explore data that you create or import from files.

As you work in MATLAB, you issue commands that create variables and call functions. For example, create a variable named `a` by typing this statement at the command line:

```
a = 1
```

MATLAB adds variable `a` to the workspace and displays the result in the Command Window.

```
a =

     1
```

Create a few more variables.

```
b = 2

b =

     2

c = a + b

c =

     3

d = cos(a)

d =

    0.5403
```

When you do not specify an output variable, MATLAB uses the variable `ans`, short for *answer*, to store the results of your calculation.

```
sin(a)

ans =

    0.8415
```

If you end a statement with a semicolon, MATLAB performs the computation, but suppresses the display of output in the Command Window.

```
e = a*b;
```

You can recall previous commands by pressing the up- and down-arrow keys, ↑ and ↓. Press the arrow keys either at an empty command line or after you type the first few characters of a command. For example, to recall the command `b = 2`, type `b`, and then press the up-arrow key.

Matrices and Arrays

MATLAB is an abbreviation for "matrix laboratory." While other programming languages mostly work with numbers one at a time, MATLAB® is designed to operate primarily on whole matrices and arrays.

All MATLAB variables are multidimensional *arrays*, no matter what type of data. A *matrix* is a two-dimensional array often used for linear algebra.

Array Creation

To create an array with four elements in a single row, separate the elements with either a comma (`,`) or a space.

```
a = [1 2 3 4]
```

```
a = 1×4

     1     2     3     4
```

This type of array is a *row vector*.

To create a matrix that has multiple rows, separate the rows with semicolons.

```
a = [1 2 3; 4 5 6; 7 8 10]
```

```
a = 3×3

     1     2     3
     4     5     6
     7     8    10
```

Another way to create a matrix is to use a function, such as `ones`, `zeros`, or `rand`. For example, create a 5-by-1 column vector of zeros.

```
z = zeros(5,1)
```

```
z = 5×1

     0
     0
     0
```

```
     0
     0
```

Matrix and Array Operations

MATLAB allows you to process all of the values in a matrix using a single arithmetic operator or function.

```
a + 10
```

```
ans = 3×3

    11    12    13
    14    15    16
    17    18    20
```

```
sin(a)
```

```
ans = 3×3

    0.8415    0.9093    0.1411
   -0.7568   -0.9589   -0.2794
    0.6570    0.9894   -0.5440
```

To transpose a matrix, use a single quote (`'`):

```
a'
```

```
ans = 3×3

     1     4     7
     2     5     8
     3     6    10
```

You can perform standard matrix multiplication, which computes the inner products between rows and columns, using the * operator. For example, confirm that a matrix times its inverse returns the identity matrix:

```
p = a*inv(a)
```

```
p = 3×3
```

```
    1.0000         0   -0.0000
         0    1.0000         0
         0         0    1.0000
```

Notice that `p` is not a matrix of integer values. MATLAB stores numbers as floating-point values, and arithmetic operations are sensitive to small differences between the actual value and its floating-point representation. You can display more decimal digits using the `format` command:

```
format long
p = a*inv(a)
```

```
p = 3×3

   1.000000000000000                   0  -0.000000000000000
                   0   1.000000000000000                   0
                   0                   0   0.999999999999998
```

Reset the display to the shorter format using

```
format short
```

`format` affects only the display of numbers, not the way MATLAB computes or saves them.

To perform element-wise multiplication rather than matrix multiplication, use the `.*` operator:

```
p = a.*a
```

```
p = 3×3

     1     4     9
    16    25    36
    49    64   100
```

The matrix operators for multiplication, division, and power each have a corresponding array operator that operates element-wise. For example, raise each element of `a` to the third power:

```
a.^3
```

```
ans = 3×3

           1           8          27
          64         125         216
         343         512        1000
```

Concatenation

Concatenation is the process of joining arrays to make larger ones. In fact, you made your first array by concatenating its individual elements. The pair of square brackets `[]` is the concatenation operator.

```
A = [a,a]
```

```
A = 3×6

     1     2     3     1     2     3
     4     5     6     4     5     6
     7     8    10     7     8    10
```

Concatenating arrays next to one another using commas is called *horizontal* concatenation. Each array must have the same number of rows. Similarly, when the arrays have the same number of columns, you can concatenate *vertically* using semicolons.

```
A = [a; a]
```

```
A = 6×3

     1     2     3
     4     5     6
     7     8    10
     1     2     3
     4     5     6
     7     8    10
```

Complex Numbers

Complex numbers have both real and imaginary parts, where the imaginary unit is the square root of `-1`.

```
sqrt(-1)
```

```
ans = 0.0000 + 1.0000i
```

To represent the imaginary part of complex numbers, use either `i` or `j`.

```
c = [3+4i, 4+3j; -i, 10j]
```

```
c = 2×2 complex

   3.0000 + 4.0000i   4.0000 + 3.0000i
   0.0000 - 1.0000i   0.0000 +10.0000i
```

Array Indexing

Every variable in MATLAB® is an array that can hold many numbers. When you want to access selected elements of an array, use indexing.

For example, consider the 4-by-4 magic square A:

```
A = magic(4)
```

```
A = 4×4

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

There are two ways to refer to a particular element in an array. The most common way is to specify row and column subscripts, such as

```
A(4,2)
```

```
ans = 14
```

Less common, but sometimes useful, is to use a single subscript that traverses down each column in order:

```
A(8)
```

```
ans = 14
```

Using a single subscript to refer to a particular element in an array is called *linear indexing*.

If you try to refer to elements outside an array on the right side of an assignment statement, MATLAB throws an error.

```
test = A(4,5)
```

```
Index exceeds matrix dimensions.
```

However, on the left side of an assignment statement, you can specify elements outside the current dimensions. The size of the array increases to accommodate the newcomers.

```
A(4,5) = 17
```

```
A = 4×5

    16     2     3    13     0
     5    11    10     8     0
     9     7     6    12     0
     4    14    15     1    17
```

To refer to multiple elements of an array, use the colon operator, which allows you to specify a range of the form `start:end`. For example, list the elements in the first three rows and the second column of `A`:

```
A(1:3,2)
```

```
ans = 3×1

     2
    11
     7
```

The colon alone, without start or end values, specifies all of the elements in that dimension. For example, select all the columns in the third row of `A`:

```
A(3,:)
```

```
ans = 1×5

     9     7     6    12     0
```

The colon operator also allows you to create an equally spaced vector of values using the more general form `start:step:end`.

```
B = 0:10:100
```

```
B = 1×11

     0    10    20    30    40    50    60    70    80    90   100
```

If you omit the middle step, as in `start:end`, MATLAB uses the default step value of `1`.

Workspace Variables

The workspace contains variables that you create within or import into MATLAB from data files or other programs. For example, these statements create variables `A` and `B` in the workspace.

```
A = magic(4);
B = rand(3,5,2);
```

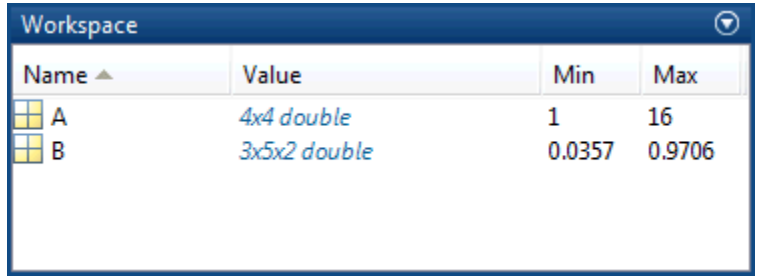
You can view the contents of the workspace using `whos`.

```
whos
```

```
  Name      Size              Bytes  Class     Attributes

  A         4x4                 128  double
  B         3x5x2               240  double
```

The variables also appear in the Workspace pane on the desktop.



Workspace variables do not persist after you exit MATLAB. Save your data for later use with the `save` command,

```
save myfile.mat
```

Saving preserves the workspace in your current working folder in a compressed file with a `.mat` extension, called a MAT-file.

To clear all the variables from the workspace, use the `clear` command.

Restore data from a MAT-file into the workspace using `load`.

```
load myfile.mat
```

Text and Characters

Text in String Arrays

When you are working with text, enclose sequences of characters in double quotes. You can assign text to a variable.

```
t = "Hello, world";
```

If the text includes double quotes, use two double quotes within the definition.

```
q = "Something ""quoted"" and something else."
```

```
q =

    "Something "quoted" and something else."
```

`t` and `q` are arrays, like all MATLAB variables. Their *class* or data type is `string`.

```
whos t
```

```
  Name        Size            Bytes  Class     Attributes
  t           1x1               174  string
```

Note Creating string arrays with double quotes was introduced in R2017a. If you are using an earlier release, create character arrays. For details, see “Data in Character Arrays” on page 1-14.

To add text to the end of a string, use the plus operator, `+`.

```
f = 71;
c = (f-32)/1.8;
tempText = "Temperature is " + c + "C"
```

```
tempText =
"Temperature is 21.6667C"
```

Similar to numeric arrays, string arrays can have multiple elements. Use the `strlength` function to find the length of each string within an array.

```
A = ["a","bb","ccc"; "dddd","eeeeee","fffffff"]
```

```
A =
  2×3 string array
    "a"       "bb"        "ccc"
    "dddd"    "eeeeee"    "fffffff"

strlength(A)

ans =

     1     2     3
     4     6     7
```

Data in Character Arrays

Sometimes characters represent data that does not correspond to text, such as a DNA sequence. You can store this type of data in a character array, which has data type `char`. Character arrays use single quotes.

```
seq = 'GCTAGAATCC';
whos seq
```

```
  Name      Size            Bytes  Class    Attributes
  seq       1x10               20  char
```

Each element of the array contains a single character.

```
seq(4)

ans =
    'A'
```

Concatenate character arrays with square brackets, just as you concatenate numeric arrays.

```
seq2 = [seq 'ATTAGAAACC']

seq2 =
    'GCTAGAATCCATTAGAAACC'
```

Character arrays are common in programs that were written before the introduction of string arrays. All MATLAB functions that accept `string` data also accept `char` data, and vice versa.

Calling Functions

MATLAB® provides a large number of functions that perform computational tasks. Functions are equivalent to *subroutines* or *methods* in other programming languages.

To call a function, such as `max`, enclose its input arguments in parentheses:

```
A = [1 3 5];
max(A)
```

```
ans = 5
```

If there are multiple input arguments, separate them with commas:

```
B = [10 6 4];
max(A,B)
```

```
ans = 1×3

    10     6     5
```

Return output from a function by assigning it to a variable:

```
maxA = max(A)
```

```
maxA = 5
```

When there are multiple output arguments, enclose them in square brackets:

```
[maxA,location] = max(A)
```

```
maxA = 5
```

```
location = 3
```

Enclose any character inputs in single quotes:

```
disp('hello world')
```

```
hello world
```

To call a function that does not require any inputs and does not return any outputs, type only the function name:

```
clc
```

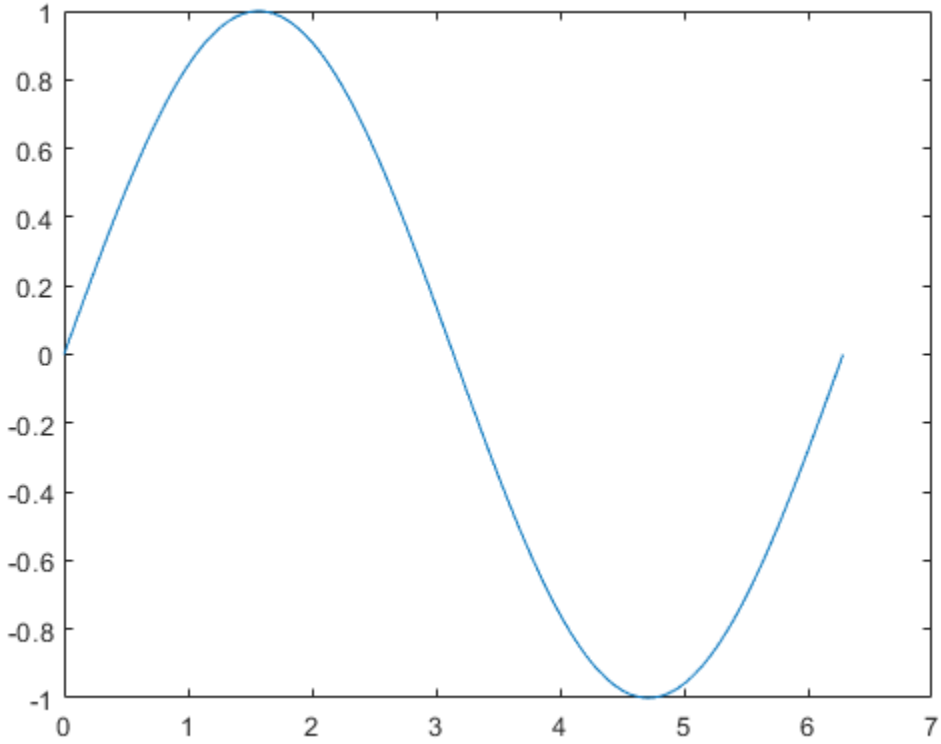
The `clc` function clears the Command Window.

2-D and 3-D Plots

Line Plots

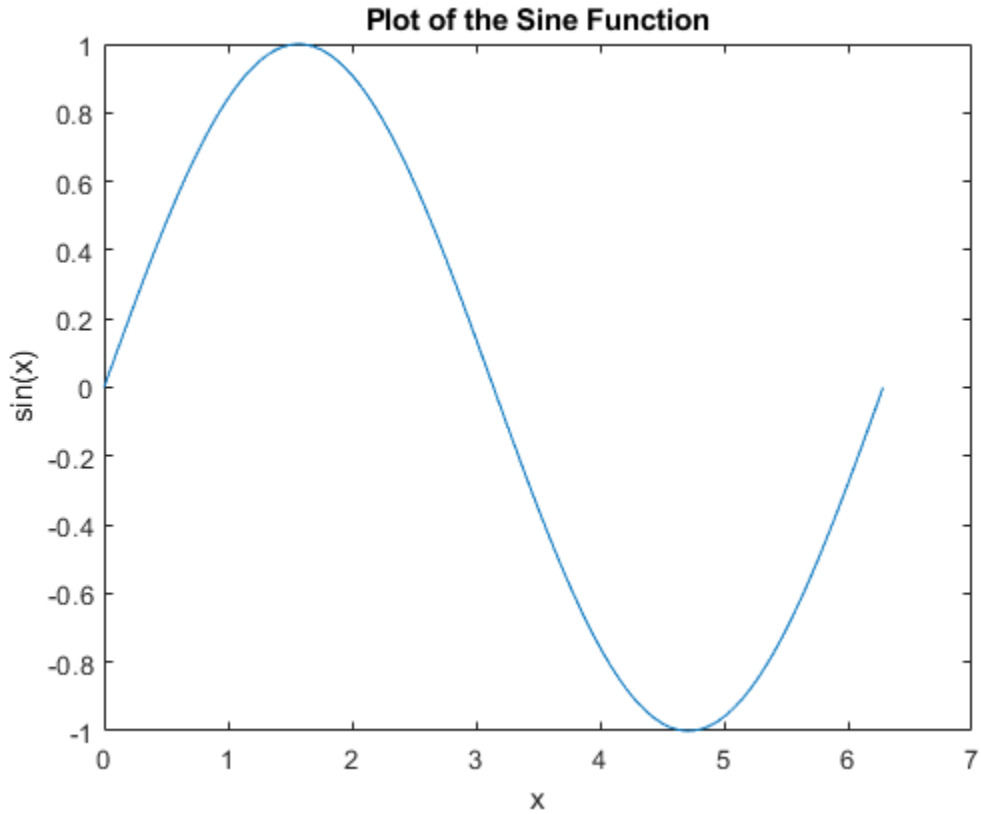
To create two-dimensional line plots, use the `plot` function. For example, plot the value of the sine function from 0 to 2π:

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```



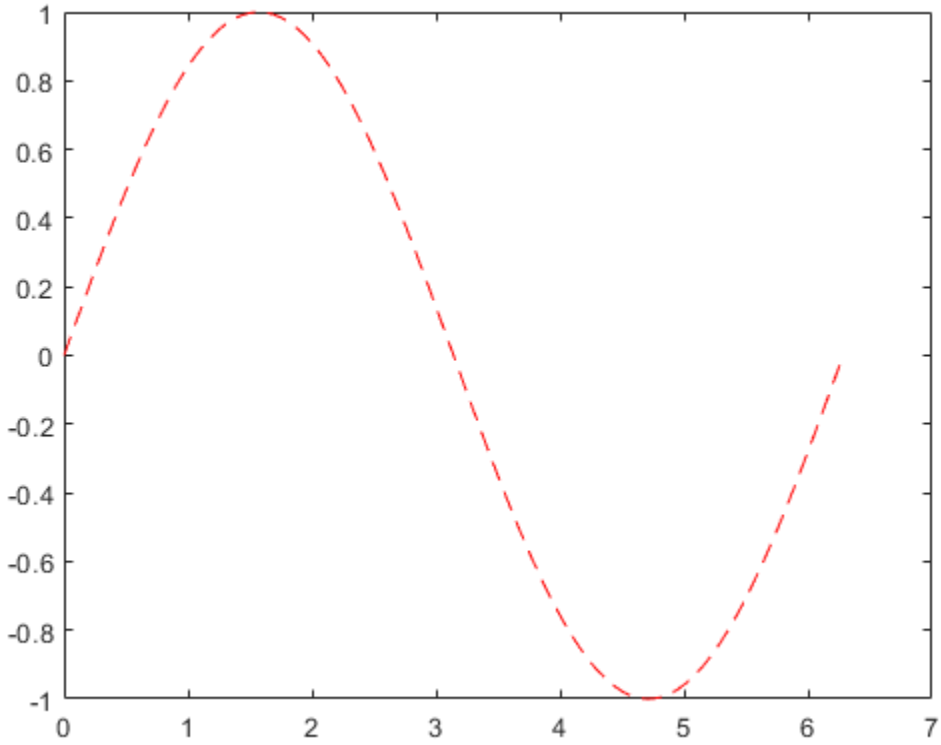
You can label the axes and add a title.

```
xlabel('x')
ylabel('sin(x)')
title('Plot of the Sine Function')
```



By adding a third input argument to the `plot` function, you can plot the same variables using a red dashed line.

```
plot(x,y,'r--')
```



`'r--'` is a *line specification*. Each specification can include characters for the line color, style, and marker. A marker is a symbol that appears at each plotted data point, such as a `+`, `o`, or `*`. For example, `'g:*'` requests a dotted green line with * markers.

Notice that the titles and labels that you defined for the first plot are no longer in the current *figure* window. By default, MATLAB® clears the figure each time you call a plotting function, resetting the axes and other elements to prepare the new plot.

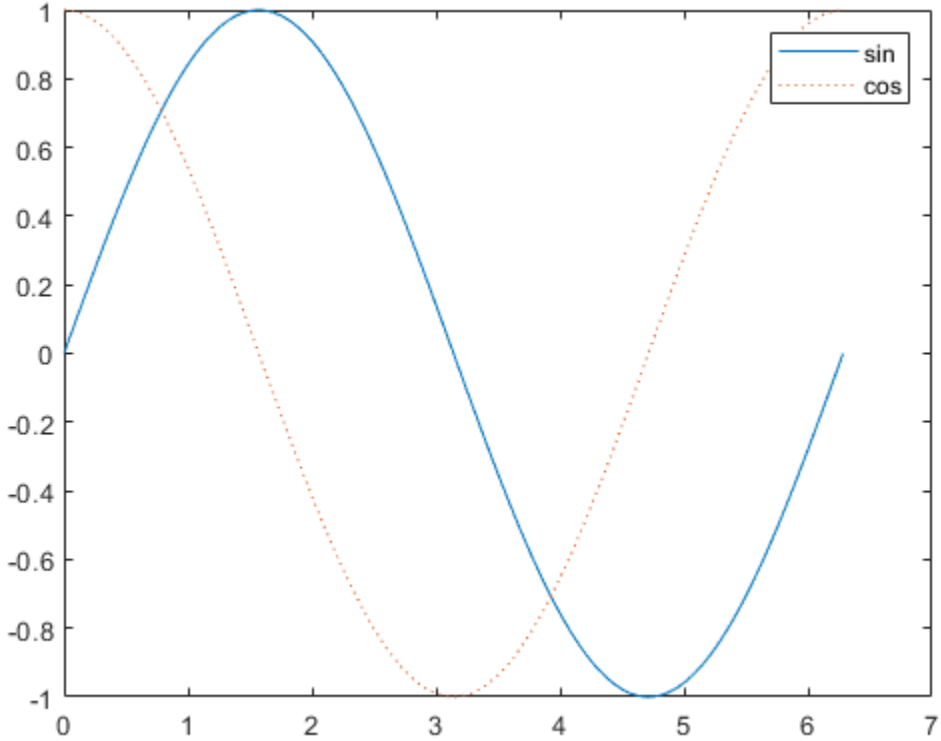
To add plots to an existing figure, use `hold on`. Until you use `hold off` or close the window, all plots appear in the current figure window.

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```

```
hold on

y2 = cos(x);
plot(x,y2,':')
legend('sin','cos')

hold off
```



Help and Documentation

All MATLAB functions have supporting documentation that includes examples and describes the function inputs, outputs, and calling syntax. There are several ways to access this information from the command line:

- Open the function documentation in a separate window using the `doc` command.

  ```
  doc mean
  ```

- Display function hints (the syntax portion of the function documentation) in the Command Window by pausing after you type the open parentheses for the function input arguments.

  ```
  mean(
  ```

- View an abbreviated text version of the function documentation in the Command Window using the `help` command.

  ```
  help mean
  ```

Access the complete product documentation by clicking the help icon  .