

This information was taken from **Documentation for MathWorks Products (Release 13)**.

MATLAB® is a high-performance language for technical computing. The MATLAB system consists of five main parts:

Development Environment. This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, an editor and debugger, and browsers for viewing help, the workspace, files, and the search path.


The MATLAB Mathematical Function Library. This is a vast collection of computational algorithms such as sum, sine, cosine, matrix inverse, and matrix eigenvalues.

The MATLAB Language. This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features.

Graphics. MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs.

The MATLAB Application Program Interface (API). This is a library that allows you to write C and Fortran programs that interact with MATLAB.

Starting MATLAB

On Windows platforms, to start MATLAB, double-click the MATLAB shortcut icon  on your Windows desktop.

On UNIX platforms, to start MATLAB, type `matlab` at the operating system prompt.

After starting MATLAB, the MATLAB desktop (see below) opens.

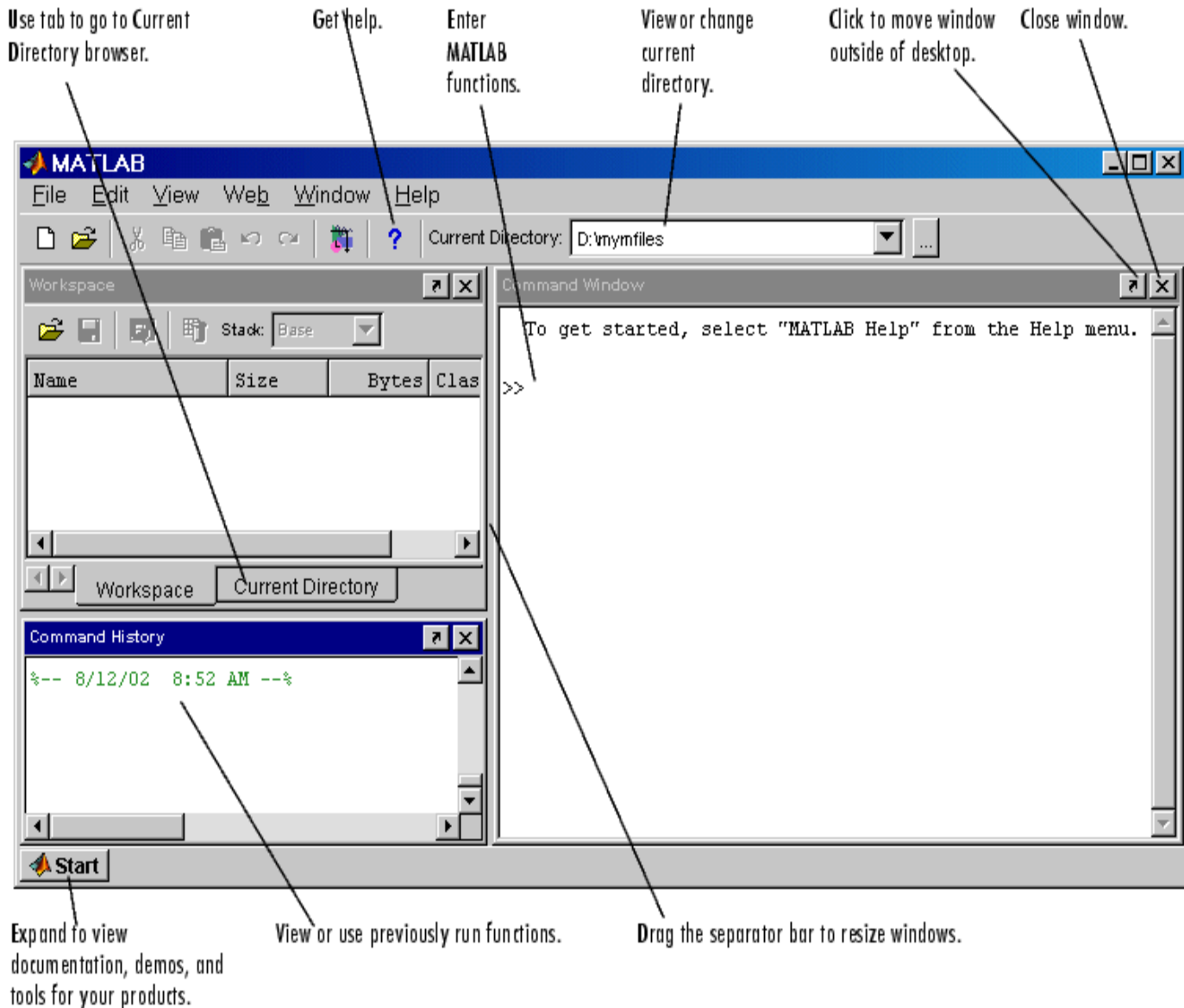
Quitting MATLAB

To end your MATLAB session, select **Exit MATLAB** from the **File** menu in the desktop, or type `quit` in the Command Window. To execute specified functions each time MATLAB quits, such as saving the workspace, you can create and run a `finish.m` script.

MATLAB Desktop

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB.

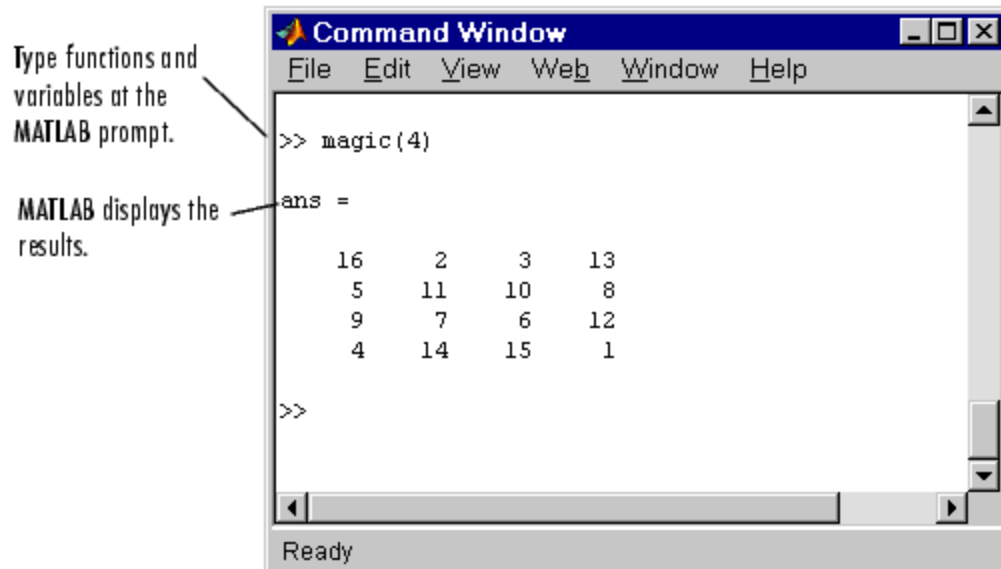
The first time MATLAB starts, the desktop appears as shown in the following illustration.



You can change the way your desktop looks by opening, closing, moving, and resizing the tools in it. Use the **View** menu to open or close the tools. You can also move tools outside the desktop or move them back into the desktop (docking). All the desktop tools provide common features such as context menus and keyboard shortcuts.

Command Window

Use the Command Window to enter variables and run functions and M-files.



Matrices

In MATLAB, a matrix is a rectangular array of numbers. Special meaning is sometimes attached to 1-by-1 matrices, which are scalars, and to matrices with only one row or column, which are vectors.

Entering Matrices

The best way for you to get started with MATLAB is to learn how to handle matrices. Start MATLAB and follow along with each example.

You can enter matrices into MATLAB in several different ways:

- Enter an explicit list of elements.
- Load matrices from external data files.
- Create matrices with your own functions in M-files.

Start by entering Dürer's matrix as a list of its elements. You only have to follow a few basic conventions:

- Separate the elements of a row with blanks or commas.
- Use a semicolon, `;`, to indicate the end of each row.
- Surround the entire list of elements with square brackets, `[]`.

To enter Dürer's matrix, simply type in the Command Window

- `A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]`

MATLAB displays the matrix you just entered.

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

This exactly matches the numbers in the engraving. Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as `A`. Now that you have `A` in the workspace, take a look at what makes it so interesting. Why is it magic?

sum, transpose, and diag

You are probably already aware that the special properties of a magic square have to do with the various ways of summing its elements. If you take the sum along any row or column, or along either of the two main diagonals, you will always get the same number. The first statement to try is

- `sum(A)`

MATLAB replies with

- `ans =`
- `34 34 34 34`

When you do not specify an output variable, MATLAB uses the variable `ans`, short for *answer*, to store the results of a calculation. You have computed a row vector containing the sums of the columns of `A`. Sure enough, each of the columns has the same sum, the *magic* sum, 34. How about the row sums? MATLAB has a preference for working with the columns of a matrix, so the easiest way to get the row sums is to transpose the matrix, compute the column sums of the transpose, and then transpose the result. The transpose operation is denoted by an apostrophe or single quote, `'`. It flips a matrix about its main diagonal and it turns a row vector into a column vector.

So

- `A'`

produces

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

And

- `sum(A')'`

produces a column vector containing the row sums

- `ans =`
- `34`
- `34`

- 34
- 34

Subscripts

The element in row i and column j of A is denoted by $A(i, j)$. For example, $A(4, 2)$ is the number in the fourth row and second column. For our magic square, $A(4, 2)$ is 15. So to compute the sum of the elements in the fourth column of A , type

- $A(1,4) + A(2,4) + A(3,4) + A(4,4)$
-

This produces

- `ans =`
- 34

but is not the most elegant way of summing a single column.

It is also possible to refer to the elements of a matrix with a single subscript, $A(k)$. This is the usual way of referencing row and column vectors. But it can also apply to a fully two-dimensional matrix, in which case the array is regarded as one long column vector formed from the columns of the original matrix. So, for our magic square, $A(8)$ is another way of referring to the value 15 stored in $A(4, 2)$.

The Colon Operator

The colon, `:`, is one of the most important MATLAB operators. It occurs in several different forms. The expression

- `1:10`

is a row vector containing the integers from 1 to 10

- 1 2 3 4 5 6 7 8 9 10

To obtain nonunit spacing, specify an increment. For example,

- `100:-7:50`

is

- 100 93 86 79 72 65 58 51

and

- `0:pi/4:pi`

is

- 0 0.7854 1.5708 2.3562 3.1416

Subscript expressions involving colons refer to portions of a matrix.

- $A(1:k, j)$

is the first k elements of the j th column of A . So

- `sum(A(1:4,4))`

computes the sum of the fourth column. But there is a better way. The colon by itself refers to *all* the elements in a row or column of a matrix and the keyword `end` refers to the *last* row or column.

So

- `sum(A(:,end))`

computes the sum of the elements in the last column of `A`.

- `ans =`
- `34`

Why is the magic sum for a 4-by-4 square equal to 34? If the integers from 1 to 16 are sorted into four groups with equal sums, that sum must be

- `sum(1:16)/4`

which, of course, is

- `ans =`
- `34`

Expressions

Like most other programming languages, MATLAB provides mathematical *expressions*, but unlike most programming languages, these expressions involve entire matrices. The building blocks of expressions are

- Variables
- Numbers
- Operators
- Functions

Variables

MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

- `num_students = 25`

creates a 1-by-1 matrix named `num_students` and stores the value 25 in its single element. Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB uses only the first 31 characters of a variable name. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. `A` and `a` are *not* the same variable. To view the matrix assigned to any variable, simply enter the variable name.

Numbers

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. *Scientific notation* uses the letter `e` to specify a power-of-ten scale factor. *Imaginary numbers* use either `i` or `j` as a suffix. Some examples of legal numbers are

- `3` `-99` `0.0001`
- `9.6397238` `1.60210e-20` `6.02252e23`
- `1i` `-3.14159j` `3e5i`

Operators

Expressions use familiar arithmetic operators and precedence rules.

+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Left division (described in "Matrices and Linear Algebra" in the MATLAB documentation)
^	Power
'	Complex conjugate transpose
()	Specify evaluation order

Functions

- MATLAB provides a large number of standard elementary mathematical functions, including `abs`, `sqrt`, `exp`, and `sin`. Taking the square root or logarithm of a negative number is not an error; the appropriate complex result is produced automatically.

Some of the functions, like `sqrt` and `sin`, are *built in*. They are part of the MATLAB core so they are very efficient, but the computational details are not readily accessible. Other functions, like `gamma` and `sinh`, are implemented in M-files.

Several special functions provide values of useful constants.

<code>pi</code>	3.14159265...
<code>i</code>	Imaginary unit, $\sqrt{-1}$
<code>j</code>	Same as <code>i</code>
<code>eps</code>	Floating-point relative precision, 2^{-52}
<code>realmin</code>	Smallest floating-point number, 2^{-1022}
<code>realmax</code>	Largest floating-point number, $(2-\epsilon)2^{1023}$
<code>Inf</code>	Infinity
<code>NaN</code>	Not-a-number

Infinity is generated by dividing a nonzero value by zero, or by evaluating well defined mathematical expressions that *overflow*. Not-a-number is generated by trying to evaluate expressions like $0/0$ or $\text{Inf}-\text{Inf}$ that do not have well defined mathematical values. The function names are not reserved. It is possible to overwrite any of them with a new variable, such as

- `eps = 1.e-6`

and then use that value in subsequent calculations. The original function can be restored with

- `clear eps`

Examples of Expressions

You have already seen several examples of MATLAB expressions. Here are a few more examples, and the resulting values.

- `rho = (1+sqrt(5))/2`
- `rho =`
- `1.6180`

- `a = abs(3+4i)`
- `a =`
- `5`

- `z = sqrt(besselk(4/3,rho-i))`
- `z =`
- `0.3730+ 0.3214i`

- `huge = exp(log(realmax))`
- `huge =`
- `1.7977e+308`

- `toobig = pi*huge`
- `toobig =`
- `Inf`

M-Files

You can create your own matrices using *M-files*, which are text files containing MATLAB code. Use the MATLAB Editor or another text editor to create a file containing the same statements you would type at the MATLAB command line. Save the file under a name that ends in `.m`.

For example, create a file containing these five lines.

- `A = [...`
- `16.0 3.0 2.0 13.0`
- `5.0 10.0 11.0 8.0`
- `9.0 6.0 7.0 12.0`
- `4.0 15.0 14.0 1.0];`

Store the file under the name `magik.m`. Then the statement

- `magik`

reads the file and creates a variable, A , containing our example matrix.

Command Line Editing

Various arrow and control keys on your keyboard allow you to recall, edit, and reuse statements you have typed earlier. For example, suppose you mistakenly enter

- `rho = (1 + sqrt(5))/2`

You have misspelled `sqrt`. MATLAB responds with

- Undefined function or variable 'sqrt'.

Instead of retyping the entire line, simply press the \uparrow key. The statement you typed is redisplayed. Use the \leftarrow key to move the cursor over and insert the missing `r`. Repeated use of the \uparrow key recalls earlier lines. Typing a few characters and then the \uparrow key finds a previous line that begins with those characters.

Following is the list of arrow and control keys you can use in the Command Window. If the preference you select for Command line key bindings is Emacs (MATLAB standard), you can also use the **Ctrl+key** combinations shown.

Key	Control Key for Emacs (MATLAB standard) Preference	Operation
\uparrow	Ctrl+P	Recall <i>previous</i> line. Works only at command line.
\downarrow	Ctrl+N	Recall <i>next</i> line. Works only at command line if you previously used the up arrow or Ctrl+P .
\leftarrow	Ctrl+B	Move <i>back</i> one character.
\rightarrow	Ctrl+F	Move <i>forward</i> one character.
Ctrl+ \rightarrow		Move <i>right</i> one word.
Ctrl+ \leftarrow		Move <i>left</i> one word.
Home	Ctrl+A	Move to beginning of command line.
End	Ctrl+E	Move to end of command line.
Ctrl+Home		Move to top of Command Window.
Ctrl+End		Move to end of Command Window.
Esc	Ctrl+U	Clear command line.
Delete	Ctrl+D	Delete character at cursor in command line.
Backspace	Ctrl+H	Delete character before cursor in command line.

Creating a Plot

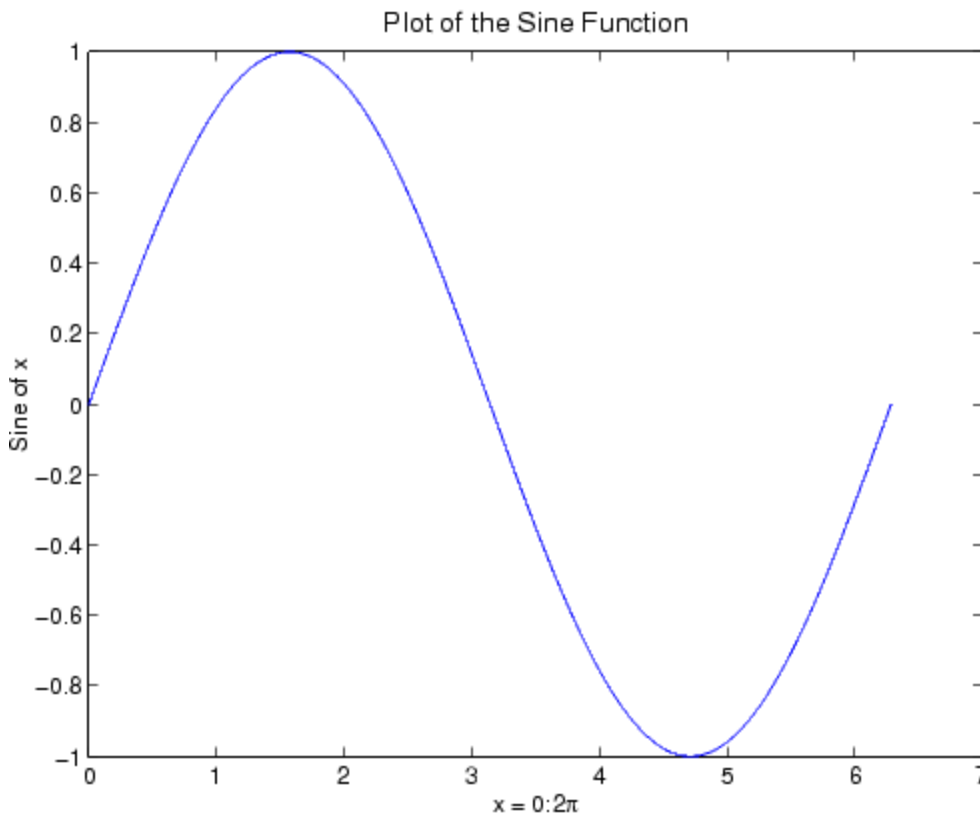
The `plot` function has different forms, depending on the input arguments. If `y` is a vector, `plot(y)` produces a piecewise linear graph of the elements of `y` versus the index of the elements of `y`. If you specify two vectors as arguments, `plot(x,y)` produces a graph of `y` versus `x`.

For example, these statements use the `colon` operator to create a vector of `x` values ranging from zero to 2π , compute the sine of these values, and plot the result.

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

Now label the axes and add a title. The characters `\pi` create the symbol π .

- `xlabel('x = 0:2\pi')`
- `ylabel('Sine of x')`
- `title('Plot of the Sine Function','FontSize',12)`



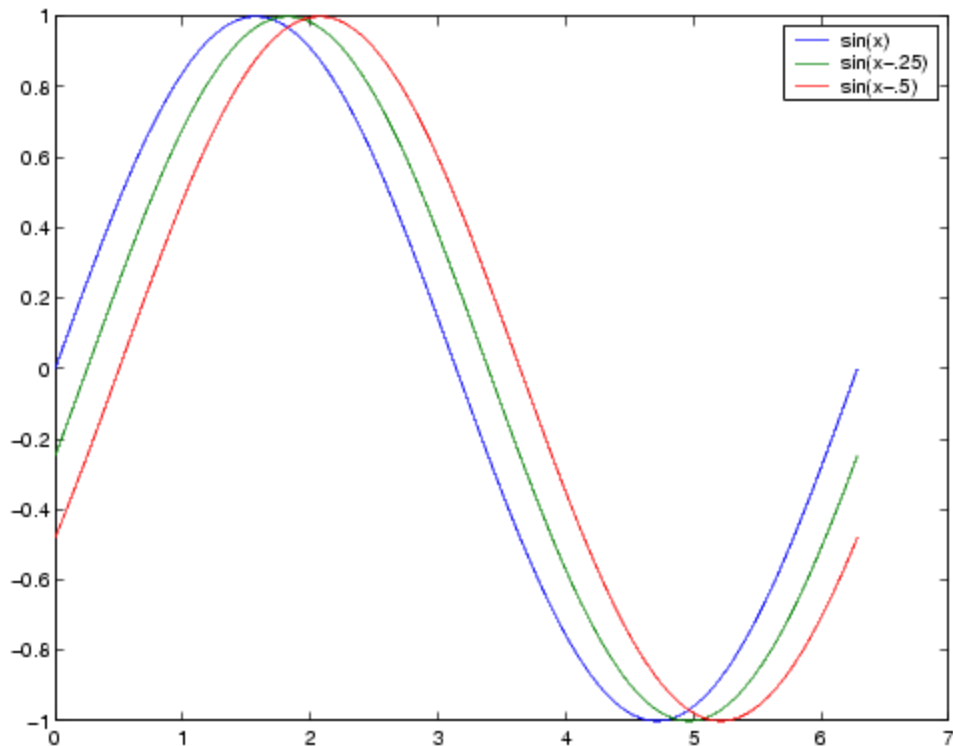
Multiple Data Sets in One Graph

Multiple x - y pair arguments create multiple graphs with a single call to `plot`. MATLAB automatically cycles through a predefined (but user settable) list of colors to allow discrimination among sets of data. For example, these statements plot three related functions of x , each curve in a separate distinguishing color.

- `y2 = sin(x-.25);`
- `y3 = sin(x-.5);`
- `plot(x,y,x,y2,x,y3)`

The `legend` command provides an easy way to identify the individual plots.

- `legend('sin(x)', 'sin(x-.25)', 'sin(x-.5)')`



Controlling the Axes

The `axis` command supports a number of options for setting the scaling, orientation, and aspect ratio of plots.

Setting Axis Limits

By default, MATLAB finds the maxima and minima of the data to choose the axis limits to span this range. The `axis` command enables you to specify your own limits

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

or for three-dimensional graphs,

- `axis([xmin xmax ymin ymax zmin zmax])`

Use the command

- `axis auto`

to reenable MATLAB automatic limit selection.

Setting Axis Aspect Ratio

`axis` also enables you to specify a number of predefined modes. For example,

- `axis square`

makes the *x*-axes and *y*-axes the same length.

- `axis equal`

makes the individual tick mark increments on the *x*- and *y*-axes the same length. This means

- `plot(exp(i*[0:pi/10:2*pi]))`

followed by either `axis square` or `axis equal` turns the oval into a proper circle.

- `axis auto normal`

returns the axis scaling to its default, automatic mode.

Setting Axis Visibility

You can use the `axis` command to make the axis visible or invisible.

- `axis on`

makes the axis visible. This is the default.

- `axis off`

makes the axis invisible.

Setting Grid Lines

The `grid` command toggles grid lines on and off. The statement

- `grid on`

turns the grid lines on and

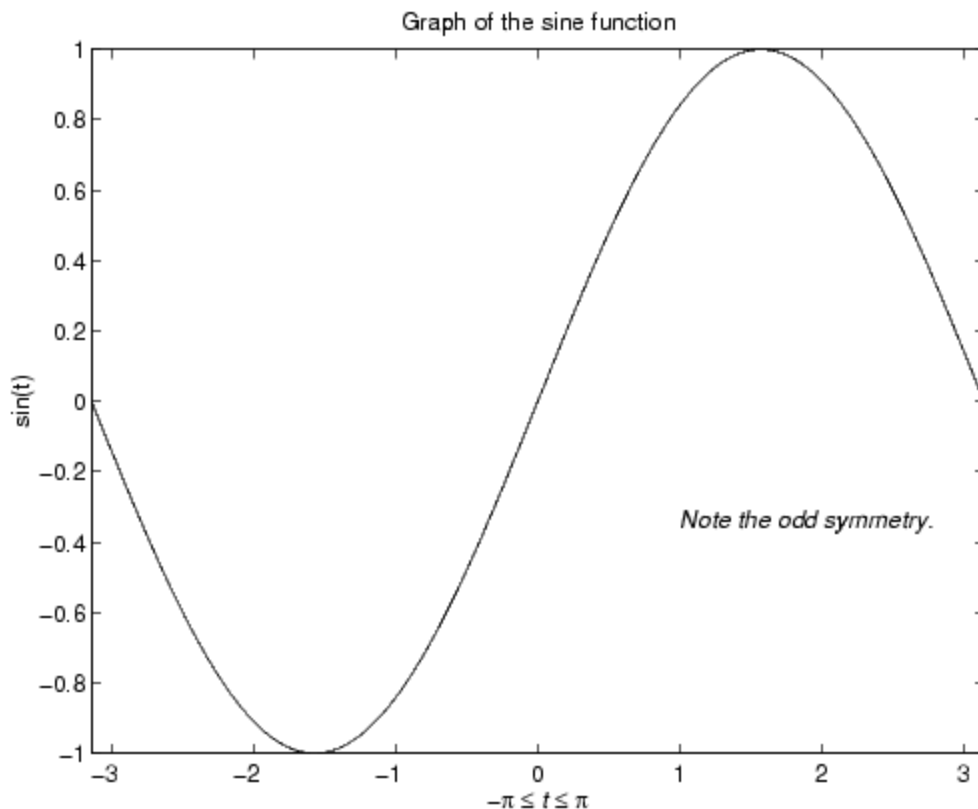
- `grid off`

turns them back off again.

Axis Labels and Titles

The `xlabel`, `ylabel`, and `zlabel` commands add x -, y -, and z -axis labels. The `title` command adds a title at the top of the figure and the `text` function inserts text anywhere in the figure. A subset of TeX notation produces Greek letters.

- `t = -pi:pi/100:pi;`
- `y = sin(t);`
- `plot(t,y)`
- `axis([-pi pi -1 1])`
- `xlabel('-\pi \leq t \leq \pi')`
- `ylabel('sin(t)')`
- `title('Graph of the sine function')`
- `text(1,-1/3,{'\itNote the odd symmetry.'})`



Saving a Figure

To save a figure, select **Save** from the **File** menu. To save it using a graphics format, such as TIFF, for use with other applications, select **Export** from the **File** menu. You can also save from the command line--use the `saveas` command, including any options to save the figure in a different format.