Elements of MATLAB

Lloyd D. Fosdick
Elizabeth R. Jessup
Carolyn J. C. Schauble

19 August 1988
Revised
February 20, 1996

High Performance Scientific Computing
University of Colorado at Boulder

The following are members of
the HPSC Group of the Department of Computer Science
at the University of Colorado at Boulder:

Lloyd D. Fosdick
Elizabeth R. Jessup
Carolyn J. C. Schauble
Gitta O. Domik

Contents

Trademark Notice

Elements of MATLAB*

Lloyd D. Fosdick
Elizabeth R. Jessup
Carolyn J. C. Schauble

19 August 1988
Revised
February 20, 1996

1 What is MATLAB?

MATLAB is an interactive system for matrix computations. It has a simple command language that allows you to easily multiply and invert matrices, solve systems of linear equations, and perform many other operations on rectangular arrays of numbers. It is also easy to plot data on the screen or printer with MATLAB.

MATLAB is often used interactively as if it were a very powerful hand calculator. But you can also use MATLAB in a programmable mode; you may write scripts for it just as you do for other command languages. You can also write your own functions, and these can be invoked interactively or from scripts or from other functions.

The examples in this document were run on UNIX workstations; both a Sun 3/60 under the SunView window environment and a DECstation

*This work has been supported by the National Science Foundation under an Educational Infrastructure grant, CDA-9017953. It has been produced by the HPSC Group, Department of Computer Science, University of Colorado, Boulder, CO 80309. Please direct comments or queries to Elizabeth Jessup at this address or e-mail `jessup@cs.colorado.edu`.

5000/200 under the X-Window system were used. A basic knowledge of UNIX is assumed for the remainder of this tutorial. MATLAB is available on other platforms, including PC's; the following MATLAB material applies to those platforms as well.

2 Getting started

These notes are intended to get you started, providing only the bare essentials. The *MATLAB User's Guide* [MathWorks 92b] and the *MATLAB Reference Guide* [MathWorks 92a] are the basic manuals.

2.1 Bringing up MATLAB

If you have to login to a different machine than the server in order to run MATLAB and you are using an X terminal, make sure the `DISPLAY` environment is set properly. Without the proper `DISPLAY` setting, the figure window cannot appear on your screen. This can be set by typing the command

```
setenv DISPLAY yourterminalname:0
```

from the UNIX shell.

If this does not work, you may need to type

```
xhost + remotemachinename
```

where *remotemachinename* is the name of the remote machine with MATLAB installed on it. In some cases, you may need to do this before logging into the remote machine. This should enable that machine to display on your local screen.

Once the `DISPLAY` environment is set correctly, get into the directory from which you wish to use MATLAB. Start MATLAB by typing the command

```
matlab
```

from the shell. The basic MATLAB environment is activated, and your window should appear as shown in the top of figure 1. Use the `quit` command to exit MATLAB.

```
% matlab

                         < M A T L A B (R) >
             (c) Copyright 1984-1993 The MathWorks, Inc.
                         All Rights Reserved
                             Version 4.1
                             Jun 10 1993

Commands to get started:  intro, demo, help help
Commands for more information:  help, whatsnew, info, subscribe

>> ...
   ⋮
>> quit

 0 flop(s).

%
```

Figure 1: MATLAB window: A sample session.

2.2 Standard help

Notice the message on the initial MATLAB window:

```
Commands to get started: intro, demo, help help
Commands for more information: help, whatsnew, info, subscribe
```

Each of these facilities may be entered by typing the appropriate name. When `help` is entered, a list of topics appears. To narrow the choice, just enter

`help` *aparticulartopic*

and helpful information on *aparticulartopic* is brought to the screen.

The `info` command provides the address of The MathWorks, Inc.; it also tells you how to obtain more information on MATLAB.

The `terminal` command lists the graphics terminals capable of running MATLAB.

Typing `demo` brings up MATLAB *Expo*; this is a mouse-driven facility with MATLAB demonstrations. These demos include examples, games, and snazzy graphics. Expo was completely implemented using MATLAB with the MATLAB *UI* (User Interface) tools. Try a few of the demos to see what MATLAB can do.

3 Some examples

This section demonstrates some basic matrix and plotting commands. As you read about each, type in the statements, as printed in `this font`, followed by a carriage return. MATLAB prints out each variable as it is assigned.

The constructs and syntax in these statements will be described in detail in section 4. This section merely provides some examples to give you the flavor of MATLAB.

3.1 Simple matrix operations

The following statements produce matrices `A` and `B`:

```
A = [ 1 2; 3 5]
B = [ 4 5; 6 7]
```

The matrices made by these statements are:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 5 \end{pmatrix}, \quad B = \begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix}$$

You can create new matrices by using A and B in expressions. The statements

```
C = A + B
D = A * B
```

produce the matrices

$$C = \begin{pmatrix} 5 & 7 \\ 9 & 12 \end{pmatrix}, \quad D = \begin{pmatrix} 16 & 19 \\ 42 & 50 \end{pmatrix}$$

Unlike some other programming languages, the MATLAB multiplication operator * performs correct matrix multiplication when working with two matrix operands; this is *not* an elementwise operation.

The statement

```
E = A'
```

makes `E` the transpose of `A`; i.e.,

$$E = \begin{pmatrix} 1 & 3 \\ 2 & 5 \end{pmatrix}$$

And the statement

```
F =  A * A'
```

makes `F` the product of `A` and its transpose; i.e.,

$$F = \begin{pmatrix} 5 & 13 \\ 13 & 34 \end{pmatrix}$$

The statements

```
Y = [1; -1]
X = A \ Y
```

give the solution to the equation

$$A * X = Y$$

that is,

$$X = \begin{pmatrix} -7 \\ 4 \end{pmatrix}$$

In order to get a feel for the notation here, think of the backslash operator, \, as denoting division from the left so that `A \ Y` in MATLAB is equivalent to the mathematical expression $A^{-1} \times Y$.

We can also use the backslash operator to solve a system with a rectangular coefficient matrix. In the case that the coefficient matrix A has more rows than columns, MATLAB returns the least squares approximation to the solution.

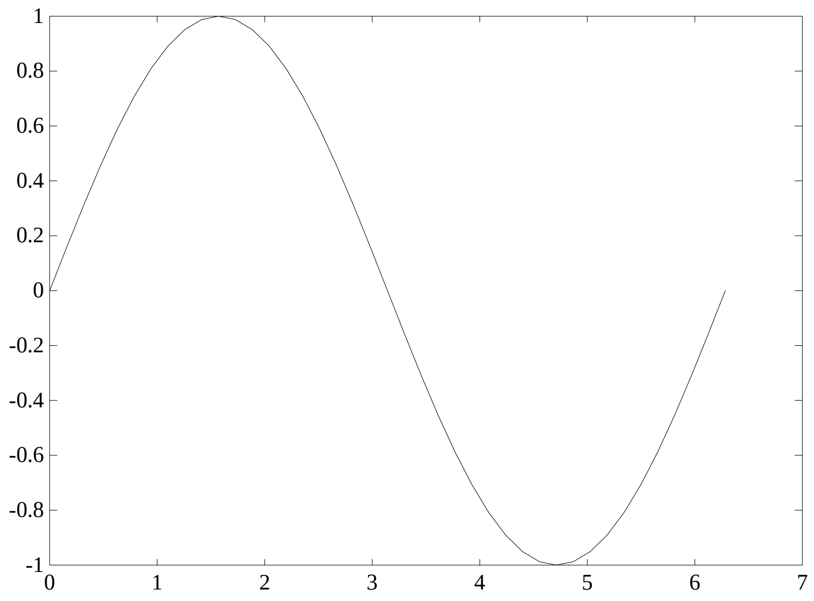


Figure 2: Plot of sine function on $[0, 2\pi]$.

3.2 Simple plots

The statements

```
U = 0:pi/20:2*pi
W = sin(U)
plot(U,W)
```

generate a plot of the sine function[1] on the interval $[0, 2\pi]$ as shown in figure 2. The first statement creates a vector of 41 values beginning at zero, in increments of $\pi/20$, the last value being 2π. The second statement produces a vector of 41 values equal to the sines of the 41 values in `U`. The last statement makes a plot of the curve whose abscissae (the x-axis values) are given by the values of the elements of `U` and whose ordinates (the y-axis values) are given by the elements of `W`. Note that the name `pi` in a MATLAB statement denotes a constant equal to π.

[1]The use of functions in MATLAB is described in more detail in sections 5 and 6.

Type in these three statements. Observe that when values are assigned to a vector, those values are printed out across the screen. Extra lines are used if needed, and the columns are numbered.

Notice that a new window is generated by the first plotting command; this graphics window is called the *figure window.* If you are using an X terminal, the mesh grid outlining the MATLAB figure window may appear first, allowing you to place it anywhere on the screen. Use the mouse to drag it to your preferred location and then click the lefthand button of the mouse.

The figure window remains until you exit your MATLAB session or until you use the `close` command. Any additional plotting commands reuse this figure window, unless you open a new figure window with the `figure` command. On most windowing systems, the figure windows can be closed, reopened, moved, or resized, in the same manner as any other window.

This is a very simple plot. You may feel the need to label the axes and provide a title. This is not difficult. MATLAB commands for labelling plots and commands for controlling the size of the axes and grid are covered in section 8. Image processing is discussed there as well.

4 Short outline of the language

This section provides information about the basic syntax and semantics for MATLAB commands. For additional information, use the `help` command or see your MATLAB manual.

4.1 Types

Fundamentally there is one type, a rectangular array of numbers. There are no type declarations. The dimensions of an array are determined by the context.

Nevertheless, it is convenient to think of three types in the language: *scalar*, actually an array consisting of one row and one column; *vector*, actually an array consisting of one row and c columns, or an array consisting of r rows and one column; and *matrix*, an array consisting of r rows and c columns.

4.2 Names

Names consist of a letter followed by zero or more letters, digits, and underscore characters. Only the first 19 characters are significant. Uppercase and lowercase letters are distinguished; thus, `A1` and `a1` denote different variables.

4.3 Scalar constants

These values are written with an optional decimal point and an optional power of 10. A minus sign is placed at the front of negative values. No blanks are permitted within a value. Examples of legal values are:

```
99  39.24  -0.0075  1.35e-24  0.2E-5  12.0e44
```

Complex numbers are also allowed. If you type

```
 z=2-5i
```

at the MATLAB prompt, the response will be

```
z =
   2.0000 - 5.0000i
```

The character j may also represent the value of $\sqrt{-1}$ as in the following:

```
 zz = 3 + 2j

zz =
   3.0000 + 2.0000i
```

4.4 Display format

MATLAB has the ability to display the values of variables in several different ways, including **`short`**, **`long`**, **`short e`**, and **`long e`**. The default format is called a **`short`** format and shows the number to 4 decimal places. For instance, if you type

```
 x = 32.75
```

MATLAB responds with

```
         x =
            32.7500
```

Should you specify that you wish to use the short display format at this format, MATLAB displays `x` in the same manner.

```
          format short
          x

         x =
            32.7500
```

The `long` format has fourteen decimal places.

```
          format long
          x

         x =
           32.75000000000000

          z

         z =
           2.00000000000000 - 5.00000000000000i
```

The two `e` formats give values in scientific form (i.e., floating-point), both long and short:

```
          format short e
          x

         x =
            3.2750e+01

          format long e
          x

         x =
              3.275000000000000e+01
```

It is also possible to display values in hexadecimal or in bank format (with up to two decimal places). Try

```
help format
```

for information on other format options. It is important to know that all values are stored as double precision numbers regardless of the format chosen for output.

4.5 Vector constants

A vector constant may be expressed explicitly as in

```
[99  39.24  -0.0075]
```

a vector (1 row, 3 columns) of three elements, or it may be expressed implicitly as in

```
[1:0.5:3]
```

which is equivalent to the expression

```
[1  1.5  2  2.5  3]
```

The expression `1:0.5:3.0` is a *constructor*. The semantics of this constructor are given by:

$$initial\,value : \; step : \; final\,value$$

The parameter *step* may be negative, as in

```
[3:-0.5:1]
```

If the *step* parameter is omitted, it is assumed to be 1. The elements of a vector may be separated by one or more blanks, as above, or by commas. Type the statement

```
V = [6:-0.3:3]
```

and observe the resultant vector.

These vectors are called *row* vectors. *Column* vector consist of a single column with one or more rows. A column vector may be defined by the expression

```
[9; -45.4; 0.22]
```

where a semicolon (;) is used to terminate each row. The transpose of a row vector also forms a column vector. Try entering the statements

```
VR = [-5; 0.25; 3.5; 0.0; 6.2]
VT = V'
```

to see the column vectors produced.

4.6 Matrix constants

A matrix constant may be expressed by explicitly listing the elements, with rows separated by a semicolon, as in

```
[1  2  3  4; 1  4  9  16; 0.5  1.0  4.5  -8]
```

which is a matrix consisting of 3 rows and 4 columns. The rows of a matrix may be written on separate lines of the input, omitting the semicolon, as in

```
[1 2 3 4
1 4 9 16
0.5 1.0 4.5 -8]
```

A row can be specified with a vector constructor as in

```
[1:4; 1  4  9  16; 0.5  1.0  4.5  -8]
```

Type the statement

```
M = [1:4; 1  4  9  16; 0.5  1.0  4.5  -8]
```

and observe the resultant matrix.

4.7 Arithmetic operators

The arithmetic operators are

```
+  -  *  /  \  ^
```

standing for addition, subtraction, multiplication, right division, left division, and exponentiation. The precedence of these operators is as expected; namely, ^ is done first, *, /, and \ next, and then + and -. Of course, parentheses may be used to alter this operation order.

Addition, subtraction, multiplication, and exponentiation have their usual meanings when applied to matrices, vectors, and scalars. The left division and right division operators act as ordinary division when applied to scalars. Their meaning in matrix operations is defined as follows: `A \ B` is equivalent to the mathematical expression $A^{-1} \times B$; `A / B` is equivalent to the mathematical expression $A \times B^{-1}$.

When a period character "." appears in front of an arithmetic operator, it means the operation should be performed element-by-element. For operations with scalars and for the addition and subtraction of vectors or matrices, there is no change in the operation. Recall the 2×2 matrices `A` and `B` defined earlier:

```
 A

A =
            1.00          2.00
            3.00          5.00
 B

B =
            4.00          5.00
            6.00          7.00
```

Now consider the following example :

```
C = A .* B
D = A ./ B
```

The matrices computed here are:

$$C = \begin{pmatrix} 4 & 10 \\ 18 & 35 \end{pmatrix}, \quad D = \begin{pmatrix} 0.2500 & 0.4000 \\ 0.5000 & 0.7143 \end{pmatrix}$$

The *MATLAB User's Guide* [MathWorks 92b] refers to these as *array operations.*

Using matrices defined earlier in this tutorial, try some of these operations to verify your understanding of them.

4.8 Expressions and statements

Expressions are formed in the usual way with parentheses used to denote grouping. MATLAB does a lot of checking; for instance, if you try to do something stupid like multiply a 3×3 matrix by a 4×4 matrix, then MATLAB squawks at you.

Normally, each line you write is an assignment statement as in the examples above. However there are exceptions, as in the use of the `plot` command that appeared in section 3.2 and for the control statements described in section 6.5.

When you have completed typing in an assignment statement, you get an echo on the screen that shows the value of the expression on the right of the assignment, as we have observed earlier. You can suppress the echo by putting a semicolon at the end of the line. If you type a line containing only an expression, as in

```
A + B
```

then the value is assigned to a default variable `ans`.

A long line of input can be continued on the next line by using an ellipsis as in

```
A = A + B + C ...
       + D
```

Short expressions can be placed on the same line separated by commas

```
 x = 4, y = 3, z = 4

x =
         4.00


y =
         3.00


z =
         4.00
```

allowing each expression value to be returned in the same order, or they may be separated by semi-colons,

```
x = 4; y = 3; z = 4;
```

suppressing the response.

4.9 Compatibility

Operations on arrays and vectors must be compatible in the usual sense of matrix algebra. In the expression

```
A * B
```

the number of rows of `B` must equal the number of columns of `A`. In the expression

```
A .* B
```

the number of rows of `A` must equal the number of rows of `B` and likewise, for the columns.

If `x` is a scalar, and `A` is a matrix, then the expressions

```
A * x
A + x
A - x
A / x
```

are all valid; they mean that the indicated operation is to be performed element-by-element with the scalar, yielding an array of the same dimension as `A`. The expression

```
A ^ x
```

implies that the matrix `A` is to be multiplied by itself `x-1` times.

4.10 Matrix references

The usual subscript notation can be used to reference the elements of a matrix. Thus `A(2,3)` is the element of `A` in the second row and third column.

You also can refer to rows of a matrix, columns of a matrix, and blocks of rows and columns. Thus `A(:,2)` refers to the second column of A. In particular, if A is the matrix defined earlier, then the statement

```
X = A(:,2)
```

gives us the vector

$$X = \begin{pmatrix} 2 \\ 5 \end{pmatrix}$$

Similarly, `A(2,:)` refers to the second row of `A`, i.e., $\begin{pmatrix} 3 & 5 \end{pmatrix}$.

Now, suppose that `M` is a 12×12 matrix. The expression `M(3:5,5:10)` refers to a block, or submatrix of `M`, that consists of the elements in rows 3 through 5 that are also in columns 5 through 10. It is as if you cut out a 3×6 piece of `M`, as illustrated in figure 3.

4.11 Relational and logical operators

Relational expressions can be used in MATLAB as in other programming languages, such as Fortran or C.

4.11.1 Relational operators

The relational operators are

```
<, <=, >, >=, ==, and ~=
```

These can be used with scalar operands or with matrix operands. A one or a zero is returned as the result, depending on whether or not the relation proves to be true or false. When matrix operands are used, a matrix of zeros and ones is returned, formed by componentwise comparison of the matrix elements.

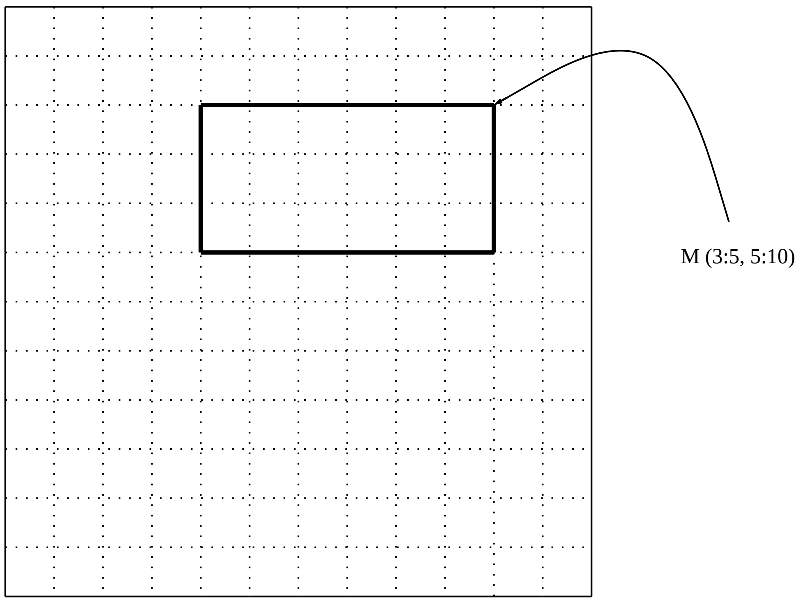


Figure 3: Submatrix of 12 x 12 matrix M.

4.11.2 Logical operators

Relational expressions can be combined using the MATLAB logical operators:

```
&, |, and ~
```

meaning `AND`, `OR`, and `NOT`, respectively. These operators are applied element-by-element.

As in other programming languages, logical operations have lower precedence than relational operations which, in turn, are lower in precedence than arithmetic operations.

5 Built-in functions

There are many built-in functions within MATLAB. You can browse the manuals to see what is available. You can also type

```
help
```

in MATLAB to obtain a list of built-in functions.

The usual math functions are built-in. We have already used the trigonometric function `sin` in the example in section 3.2 to produce a simple plot. Those MATLAB commands are repeated here:

```
U = 0:pi/20:2*pi
W = sin(U)
plot(U,W)
```

Both `sin` and `plot` are built-in functions. Notice that the `sin` function has a single argument, the vector `U`. This function returns a vector of the same length as `U` where each element is the sine of the value of the corresponding element in `U`; in other words, $w_1 = \sin(u_1)$, $w_2 = \sin(u_2)$, etc.

In the above example, the `plot` function (or command) has two arguments, `U` and `W`; both of these arguments are vectors. MATLAB plots the elements in the first vector `U` against the elements in the second vector `W`. The `plot` function can also be used with a single vector argument; in this case, the elements of the vector are plotted against the indices. Type

```
help plot
```

in MATLAB to see what other arguments can be used.

Some built-in functions are rather special. An example is the function `ones` that generates an array of ones; using this, the expression

```
ones(r,c)
```

produces an $r \times c$ array with every element equal to 1. There is a corresponding function `zeros`. Similarly, the expression

```
eye(r)
```

produces the $r \times r$ identity matrix. For more information on any of these functions, use the `help` command.

Some of the functions save a great deal of programming work. For instance, the `polyfit` function is useful for curve fitting and providing polynomial approximations; this is discussed in section 5.1. The `eig` function provides the eigenvalues and eigenvectors of a matrix argument; this is described below in section 5.2. Section 6.4 tells how to write your own functions.

It was noted earlier that MATLAB is case-sensitive. All built-in functions have lower case names.

5.1 Polynomial curve fitting

In section 3.1, we saw how to find the least squares solution to an overdetermined linear system. Sometimes, the least squares problem is not presented as a matrix problem but rather as a collection of data to be approximated in the least squares sense by a polynomial. One way to determine the coefficients of that polynomial is to set up and solve the appropriate overdetermined linear system. Another way is to call the matlab function **`polyfit`** to determine those coefficients.

Suppose, for example, that we want to make a polynomial approximation of the function $y = \sin(x)$ in the x-interval $[0, \pi]$ given the values

$$y = [0,\ 0.7071,\ 1.0000,\ 0.7071,\ 0.0000]$$

at the x values

$$x = [0,\ 0.7854,\ 1.5708,\ 2.3562,\ 3.1416].$$

Typing

```
p = polyfit(x,y,2)
```

finds the coefficients

$$p = [-0.3954,\ 1.2420,\ -0.0049]$$

of the quadratic approximating the given data y in the least squares sense. The degree of the approximating polynomial is equal to the third argument of **`polyfit`**.

In this case, the approximating polynomial is $y = p_1 x^2 + p_2 x + p_3$. Typing

```
pvals = polyval(p,x)
```

evaluates this quadratic at the given values of x.

Figure 4 shows the the sine function on the interval $[0, \pi]$ with the sampled points marked by circles. The least squares quadratic is shown by the dotted line. This figure was created with MATLAB; how to plot points and how to graph multiple curves on the same plot is covered in section 8.

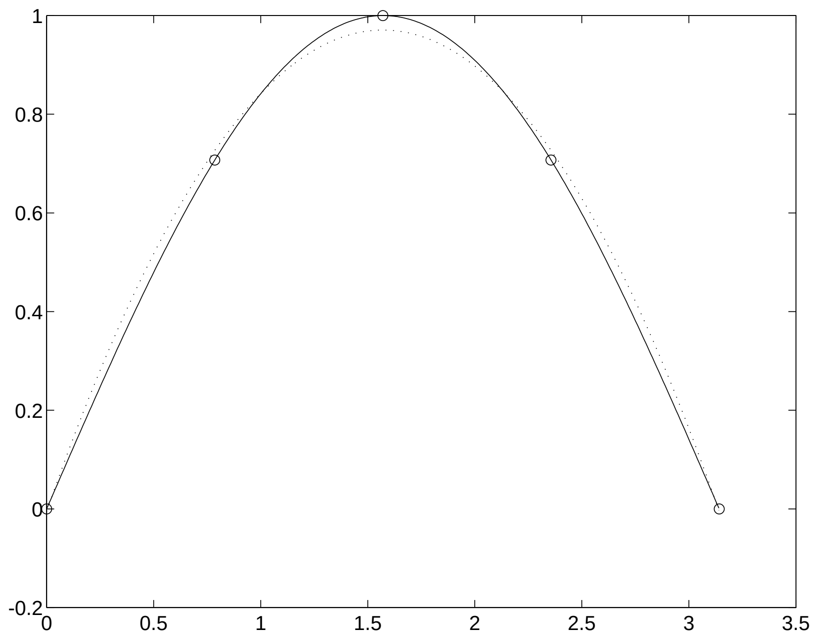


Figure 4: A quadratic fit to the sine function on $[0, \pi]$.

5.2 Eigenvalues and eigenvectors

A different MATLAB function makes it easy to compute the eigenvalues and eigenvectors of a matrix. Suppose that we have defined a matrix `A`. We can compute its eigenvalues by typing `eig(A)` as in the following example.

```
    A = [2 1 0;  1 2 1;  0 1 2];
    eig(A)

ans =

     3.4142
     2.0000
     0.5858
```

In this case, the eigenvalues are the elements of the column vector `ans`. We can put the eigenvalues into any column vector `y` by typing `y = eig(A)`.

To also compute its eigenvectors, we must provide a place for MATLAB to store them:

```
    [X,D] = eig(A)

X =

    0.5000   -0.7071   -0.5000
    0.7071    0.0000    0.7071
    0.5000    0.7071   -0.5000

D =

    3.4142         0         0
         0    2.0000         0
         0         0    0.5858
```

In this case, the eigenvalues are stored on the diagonal of the matrix `D` and the corresponding eigenvectors are stored as the columns of the matrix `X`.

We can check the quality of the computed eigenvalues and eigenvectors by computing the *residual errors.* We will generally find that the computed quantity `A*X - X*D` is a matrix with very small elements and that `A*X(:,j) - D(j,j)*X(:,j)` is a vector with very small elements, for `j` $= 1, 2, 3$. It is generally convenient to express the residual error in terms of the norm of these quantities. In exact arithmetic, these matrices and vectors and their norms would be exactly zero.

6 MATLAB scripts and user-defined functions

As mentioned earlier, it is possible to write programs for MATLAB. There are two types of MATLAB programs: *scripts* and *functions*.

A script is a program, containing regular MATLAB commands that could be entered interactively during a MATLAB session. When the name of a script is entered at the command prompt, the script is executed. This means that the commands within the script are executed, affecting the variables in the global workspace.

A function is also a program. As might be expected, a function returns a value, but otherwise it does not affect the variables in the global workspace.

Like a script, a function is executed by typing its name at the command prompt. If a function has parameters, they are entered enclosed in parenthesis following the function name.

Both scripts and functions should be stored as files with a `.m` extension, e.g., `myscript.m` or `myfunction.m`. Because of this extension, scripts and functions are typically referred to as *M-files*. Any of the commands discussed above can be used in a MATLAB script or function.

When creating and testing new MATLAB scripts and functions, you may find it useful to have two command windows open: one from which you are running MATLAB and one from which you may be editing the new script or function.

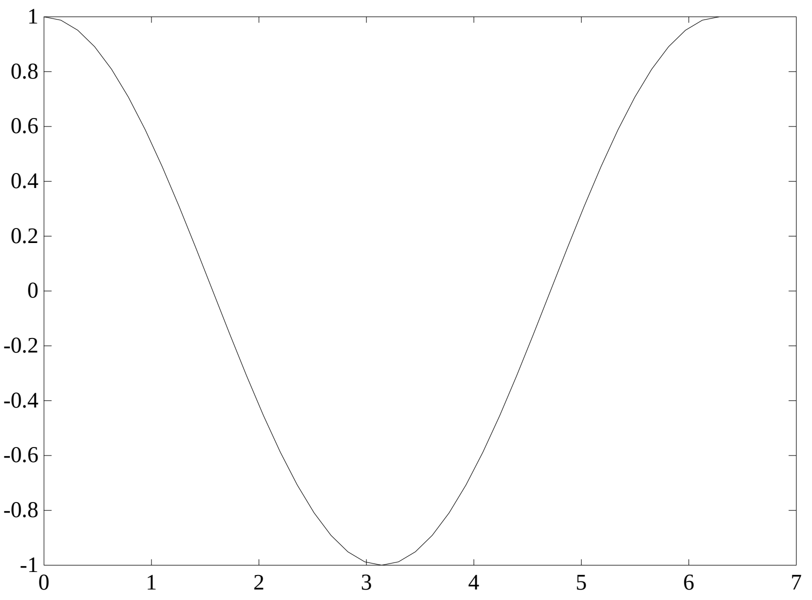


Figure 5: Plot of cosine function on $[0, 2\pi]$.

6.1 A sample script

The following is a simple script that plots a cosine curve in MATLAB.

```
%  This is a sample MATLAB script
%    that plots a cosine curve

U = 0:pi/20:2*pi
Z = cos(U)
plot(U,Z)     % This statement does the plotting.
```

The first two lines of this script are comments followed by a blank line; see section 6.2 for a more detailed discussion on the use of comments and white space in MATLAB scripts. Except for the comment at the end of the last line, the rest of the script is similar to the commands that produced the sine curve in figure 2.

To use a script within MATLAB, just enter the script file name without the `.m` extension; typing `myscript` will execute the script `myscript.m`. If the sample script above is stored in a file named `plotcos.m` in the directory from which you are running MATLAB, you need only type

```
plotcos
```

to run the script, causing the appropriate plot to appear in your figure window, as shown in figure 5. Note that running this script may alter the values of `U` and `Z`.

6.2 Comments and white space

For both scripts and functions, the percent symbol `%` precedes a comment:

```
% This is a comment in MATLAB.
```

The `%` symbol may be in any position of the line; whatever follows the `%` is considered to be part of that comment. A comment may even follow a MATLAB command on the same line.

```
plot(U,Z)    % This is a plot command
```

It is useful to place explanatory comments at the beginning of your MATLAB script or function as documentation. If you later type the command

```
help myprog
```

MATLAB responds by printing out the first contiguous block of comments in the script or function stored in `myprog.m`.

Blank lines may be inserted in a MATLAB script or function; this provides white space and promotes the readability of the program. The blank line following the first block of comments in the `myscript` script indicates the end of the lines to be printed by the `help myscript` command.

6.3 Continued lines

At times it is desirable to break up a MATLAB command line into two or more separate lines. As discussed above, an ellipsis `...` at the end of any MATLAB command line indicates that that line is to be continued onto the next line. This symbol may consist of three or more consecutive periods.

```
S = [ 1  ...
         2;  3   ....
            4 ]
```

The definition of a matrix may require several lines since each line represents a row of the matrix. The line for each row may itself be a continued line.

6.4 A sample function

Like a script, a function is a collection of MATLAB commands stored in an M-file. Unlike a script, a function can itself be evaluated. A function may take on a scalar or an array value. A scalar function value can be viewed by typing the file name without the extension, e.g., `myfunction`, or it can be assigned directly to a variable, as in `y = myfunction`. The value of an array-valued function must be assigned to an array variable. User-defined functions may be used not only interactively but also within scripts or other functions.

Note: the name of the function must match the name of the M-file for it. In other words, if you are creating a function named `myfunction`, the file containing the MATLAB commands that define that function must be stored as `myfunction.m`.

A function may require input arguments. Once the arguments have been defined, the function is evaluated by typing its file name (minus the extension) followed by the argument list, e.g., `y = myfunction(arg1, arg2, ..., argn)`. For example, the following function evaluates a cubic polynomial at the larger of the two input arguments.

```
%  This is a sample MATLAB function
%    that evaluates a cubic polynomial
%    at the larger of the two arguments x1 and x2.

function y = mycubic(x1,x2)
x = max(x1,x2);
y = x^3 + 2*x^2 + 1; % This statement determines
                     % the function value.
```

If the function `mycubic` is stored in the M-file `mycubic.m`, we can evaluate the function `mycubic` by typing a series of statements like `z1 = 1; z2 = 2; z = mycubic(z1, z2)`. This series of statements causes the value 17 to be assigned to the variable `z`.

In the following example, the array-valued function `trigfunction` takes an angle `theta` (in radians) as argument and returns both its sine and cosine.

```
%  This is a sample MATLAB function
%    to evaluate the sine and cosine
%    of the input angle.

function [costheta,sintheta] = trigfunction(theta)
costheta = cos(theta);
sintheta = sin(theta);
```

To evaluate this function, we must assign its value to an array: `[c,s] = trigfunction(0)`. After this call, we see that `c = 1` and `s = 0`.

Function arguments may be manipulated within a function, but input values are the same on exit as on entry.

6.5 Control statements

MATLAB contains `for`, `while`, and `if` statements. The syntax of each is illustrated in the examples below. These statements may be used interactively, but are more commonly included with MATLAB scripts or functions.

It is important to recognize that many of the operations that might require one of these statements in a language like C or Fortran do not require them in MATLAB. Matrix multiplication is the most obvious example, since the simple expression

```
A * B
```

produces the multiplication of the matrices `A` and `B` without any looping.

6.5.1 `for` statement

The sum of all the elements of the vector `V` can be computed using the `for` statement:

```
for  i = 1 : n
   S = S + V(i)
end
```

However, this is more efficiently done by using the built-in `sum` function

```
S = sum(V)
```

The `for` statement may be nested and a constructor with an arbitrary step can be used to define the loop index.

```
total = 0
for  i = firsti : deltai : lasti
   S(i) = 0
   for  j = firstj : deltaj : lastj
      S(i) = S(i) + fun(i,j)
   end
   total = total + S(i)
end
```

where `fun` is some function of `i` and `j`.

6.5.2 `while` statement

A `while` statement can be used to control the number of iterations of a loop:

```
while  err > maxerr
    n = n + 1
    err = funapprox(n,x) - funexact(x)
end
```

A group of `while` statements can be nested and any relational expressions may be used (see section 4.11).

6.5.3 `if` statement

```
for  i = 1 : maxrow
    for  i = 1 : maxrow
        if  abs(A(i,j)) < thresh
            A(i,j) = 0
        else
            A(i,j) = sign(A(i,j))
        end
    end
end
```

6.5.4 Further help

The `help` command gives information on these control statements; e.g., type

```
help if
```

Then try

```
help break
```

7 Input/output

This section discusses methods for creating input data for MATLAB as well as ways to output the data. Graphical output is covered in section 8.

In addition to the commands discussed here, there are a number of MATLAB file input/output functions that resemble those in the C programming language. These include such functions as `fread`, `fwrite`, `fscanf`, `fprintf`, `fopen`, and `fclose`. See the *MATLAB Reference Guide* [MathWorks 92a] or use `help` for more information on using these functions.

7.1 UNIX commands within MATLAB

While in MATLAB, it is sometimes useful to run normal UNIX commands. This can be done with the *escape* command (!). For instance, to display the contents of the current directory (when you can't remember the name of your M-file), just type

```
!ls
```

Fortran and C programs can be edited, compiled, and run in the same manner.

```
!vi myprog.f
!f77 -O -o myprog myprog.f
!myprog > myoutput
```

Then the output of these programs can be used as input to MATLAB scripts or can be edited to form M-files to produce plots or other data in MATLAB.

7.2 Session log

The `diary` command make it possible to save a log of partial or entire MATLAB sessions. If you type

```
diary mylogfile
```

all the lines subsequently appearing in the MATLAB window are saved into a file named `mylogfile`. If the filename is omitted, the name `diary` is used. This feature can be turned off by the command

```
diary off
```

or by exiting MATLAB.

This not only provides a log of your session; it also suggests a method for saving the results to be later edited into another format.

7.3 Saving data

An alternate method for storing results from your MATLAB runs is using the `save` command. For example, suppose you have created the following array

```
    M = [1:1:3; 10:2:14; 31:3:37; 5:5:15]

  M =
       1     2     3
      10    12    14
      31    34    37
       5    10    15
```

and you wish to store this data for use elsewhere. Just type

```
    save mydatafile M /ascii
```

where the `/ascii` option of the command assures the results are in text format. Then the file, `mydatafile`, contains the following:

```
    1.0000000e+00   2.0000000e+00   3.0000000e+00
    1.0000000e+01   1.2000000e+01   1.4000000e+01
    3.1000000e+01   3.4000000e+01   3.7000000e+01
    5.0000000e+00   1.0000000e+01   1.5000000e+01
```

7.4 MAT-files

MATLAB data may also be read or stored using *MAT-files* with the `load` and `save` commands. There are some special routines and examples (in both Fortran and C) to assist the user. See the chapter on *Disk Files* in the *Tutorial* section of the *MATLAB User's Guide* [MathWorks 92b].

8 Graphics

The sample script given in section 6.1 illustrated how to make a plot of the cosine function using the `plot` function as shown in figure 5. This plot is a simple two-dimensional X-Y plot drawn in the current MATLAB figure window. Many other types of plots are available in MATLAB. This section introduces you to a number of these and describes how to label, combine, store, and print MATLAB plots.

8.1 Types of two-dimensional plots

There are two types of two-dimensional or linear X-Y plots: *line* and *point*. Three-dimensional wire frame and contour plots are also available; these are discussed in section 8.5. Polar, logarithmic, semi-log, and bar plots can be employed as well; see the entries on **polar**, **loglog**, **semilogx**, **semilogy**, and **bar** in the *MATLAB Reference Guide* [MathWorks 92a] or type

```
help
```

for more information on these plots.

In the graphical examples discussed earlier in this document, the *line* type of X-Y plot is used. The other type is a *point* plot; points are included as part of figure 4. With a line plot, the gaps between the points are filled in smoothly so that you get a continuous curve; in the point type, no fill-in is done so only the points you specify are plotted.

An example of a statement that gives a point plot is

```
plot(U,W,'+')
```

The sine curve appears as 41 distinct points (marked "+"), as shown in figure 6. The statement

```
plot(U,W,'+',U,W)
```

gives a point plot for the first plot, and a line plot for the second. Since both curves are the same, the effect is to highlight the points on the curve. This last feature is convenient for showing a least squares fit to experimental data. You can plot the distinct data points and the smooth curve that fits the data in the same picture. More information on graphing two curves in the same plot is given in section 8.6.1.

You can use any symbol in the set {`. + * o x`} for point plots. You can use any symbol in the set {`- -- : -.`} for line plots. If nothing is specified, as in most of the examples here, the default line plot (symbol "`-`") is employed. You can also plot with different colors when using a color monitor; look at

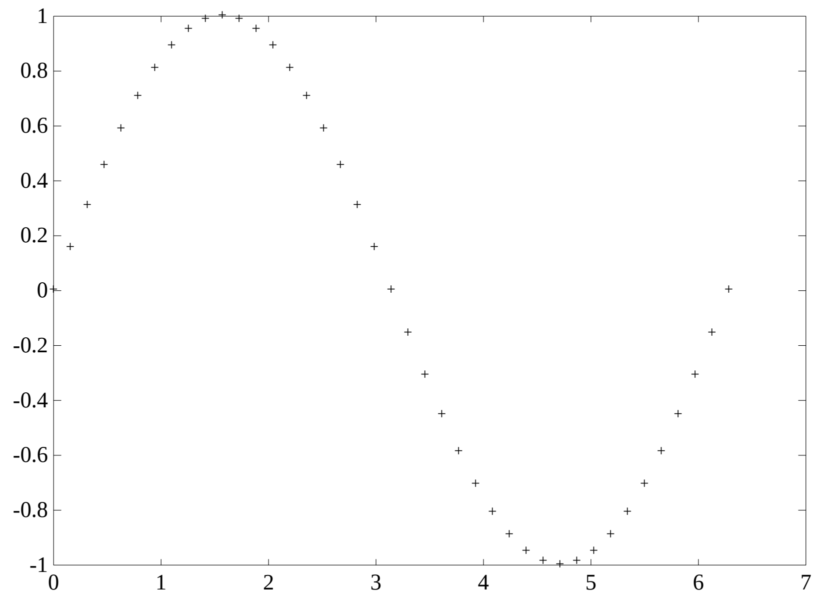


Figure 6: Point plot of sine function.

```
help plot
```

for information on specifying color arguments.

8.2 Labelling plots

The picture can be labelled, the axes can be labelled, and you can display grid lines in the coordinate system. The following script does all of this for the sine plot previously shown in figure 2:

```
%  This MATLAB script plots a sine curve

U = 0:pi/20:2*pi
W = sin(U)
plot(U,W)                      % produces basic plot
title('Sine Function')         % places title at top
xlabel('angle in radians')     % labels x-axis
```

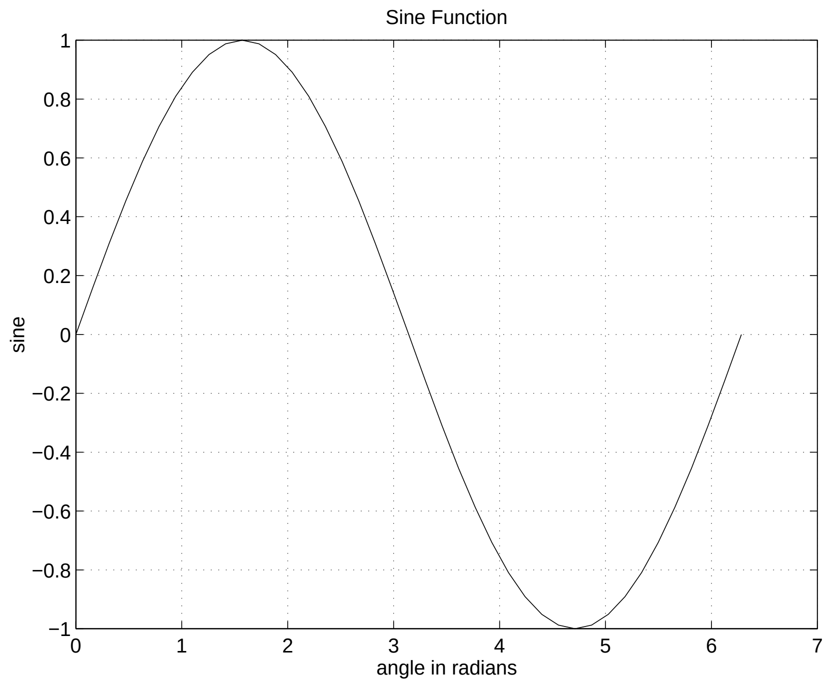


Figure 7: Labeled plot of sine function.

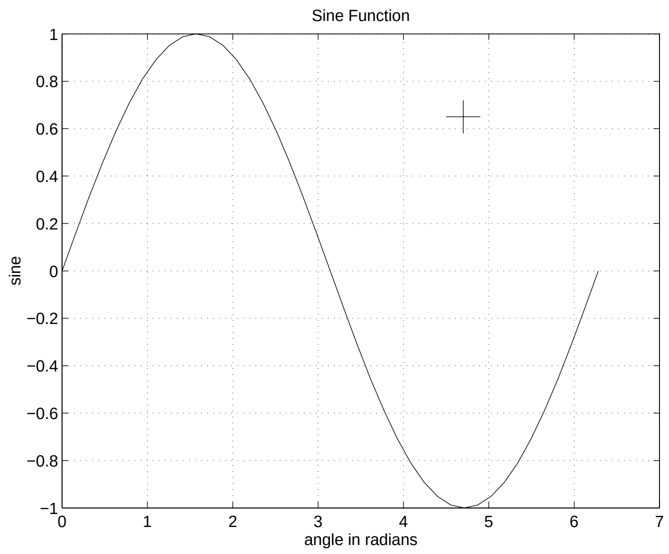
```
ylabel('sine')                    % labels y-axis
grid                              % adds grid marking
```

Notice that the labelling commands are given after the plot is created. The result is shown in figure 7.

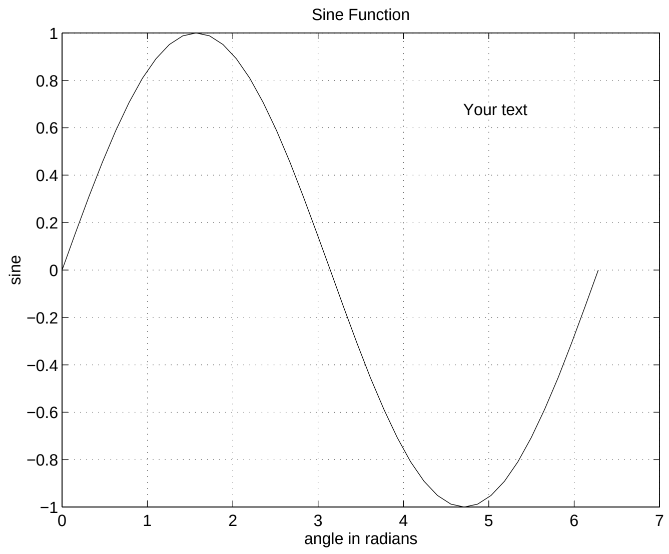
It is also possible to place text on the graph while it is in the figure window by using the mouse. The command

```
gtext('Your text')
```

makes a crosshair appear on the window containing the plot, as in figure 8(a). Just move the mouse to the desired location and click; the label containing `Your text` appears there, with the first letter placed in the inside corner of the northeast quadrant of the crosshair. The final result is in figure 8(b).



(a)



(b)

Figure 8: Sine function plot: (a) with **`gtext`** crosshair; and (b) with label entered by **`gtext`**.

8.3 Handle Graphics

Plots can also be labelled or manipulated in other ways using what MATLAB calls *handle graphics*. When the figure window is formed, it is assigned a unique number called its *handle*. This permits more than one figure window to exist at a time.[2] If you type

```
gcf
```

the value returned contains the handle of the current figure window.

A figure window is provided with a set of default properties. You can use the figure window handle to redefine any of these properties. As an example, we can alter some of the properties of the current figure window as follows:

```
handfig = gcf
set (handfig, 'Position', [0, 0, 300, 280])
```

The first line stores the handle of the current figure window in the variable `handfig`. The second line moves the window to the bottom left corner of the screen and resizes it to be 300×280 pixels. Type

```
help gcf
help set
help get
```

to learn to alter other figure window properties.

In MATLAB, every graphical object has a handle. The list of graphical objects includes the screen itself, the figure windows, and the axes of the plot within the figure windows. For instance, the handle of the root screen is the integer zero. Images, lines, surfaces, and text along with user interface controls and menus are also graphical objects. This hierarchy of graphical objects with handles allows the user to control the MATLAB graphics environment as he wishes.

The `gca` function returns the handle of the axes object in the current figure window. Using this and the `axes` function, the ticknames or ticks on the axes of the current figure may be modified. The older `axis` function can be used for a similar purpose. See the *MATLAB Reference Guide* [MathWorks 92a] or type

[2]For more information on multiple figure windows, see section 8.6.3.

```
help gca
help axes
help axis
```

for more information.

8.4 Hardcopy plots

When a plotting command is executed, the plot appears in the active figure window. This is the current figure window for all plots. Later plots may erase this plot, unless a new figure window is created or control is given to another window. However, if you type

```
print
```

on some systems, a hardcopy of the plot in the current figure window is printed on your default printer.

You can also save a copy of the plot with the `print` statement. For example,

```
print myplot -dps
```

translates the current plot into *PostScript* and stores it in a file named `myplot.ps`.

It is possible to convert the plot into a color or an encapsulated PostScript file by using other options of the `print` command. For example,

```
print mycplot -dpsc
```

generates the color PostScript file, `mycplot.ps`, and

```
print myeplot -deps
```

produces the encapsulated PostScript file, `myeplot.eps`. The PostScript files can be printed out from the UNIX shell with the `lpr` command

```
lpr myplot.ps
lpr mycplot.ps
```

and both the PostScript and encapsulated PostScript files can be incorporated into figures within the text of a paper.

Many other options are available for this command. Refer to the section on `print` in the *MATLAB Reference Guide* [MathWorks 92a] for more information.

8.5 Three-dimensional plotting

At times, three-dimensional grid plots or contour plots are desired. MATLAB provides some facility for these.

8.5.1 Three-dimensional grids

Mesh plots show a three-dimensional surface as a mesh or wire frame surface. Here the x and y values merely provide the size of the grid; a $rank(x) \times rank(y)$ matrix gives the values of the grid points of the surface – one value for each xy point. Thus, the only needed argument to the `mesh` command is that matrix.

The plot in figure 9 is created by the following script:

```
%  This MATLAB script plots a 3-D sine curve as mesh

U = 0:pi/20:2*pi;
X = ones(size(U'))*U;
Y = U'*ones(size(U));
W2 = sin(X) + sin(Y);

mesh(W2)
title('Mesh plot:  sin(X) + sin(Y)')
```

A surface plot with shading can be obtained by adding the following two commands:

```
surf(W2)
title('Surface plot:  sin(X) + sin(Y)')
```

This is shown in figure 10.

If we use the `fill3` function,

```
fill3(X, Y, W2, U)
title('3-D polygon plot:  sin(X) + sin(Y)')
```

we no longer have a surface. Instead, the grid is broken up into three-dimensional polygons, one per column. This is displayed in figure 11. Consult the manual or type `help` for a more detailed explanation of the `fill3` and related functions.

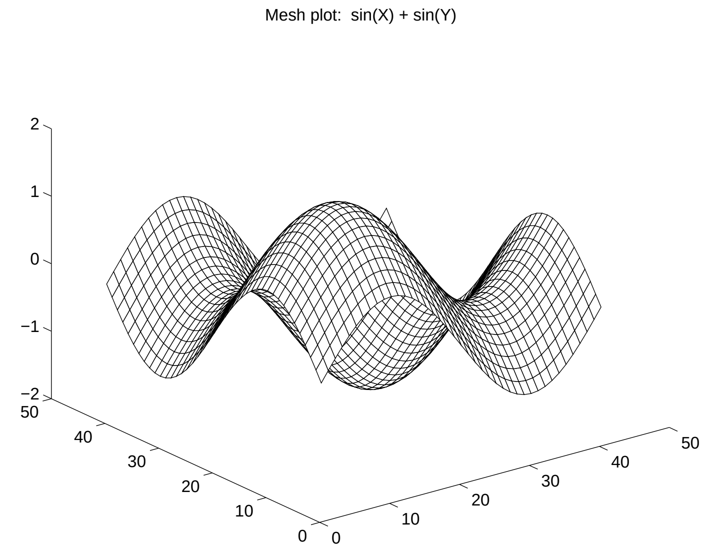


Figure 9: Mesh plot of $\sin X + \sin Y$.

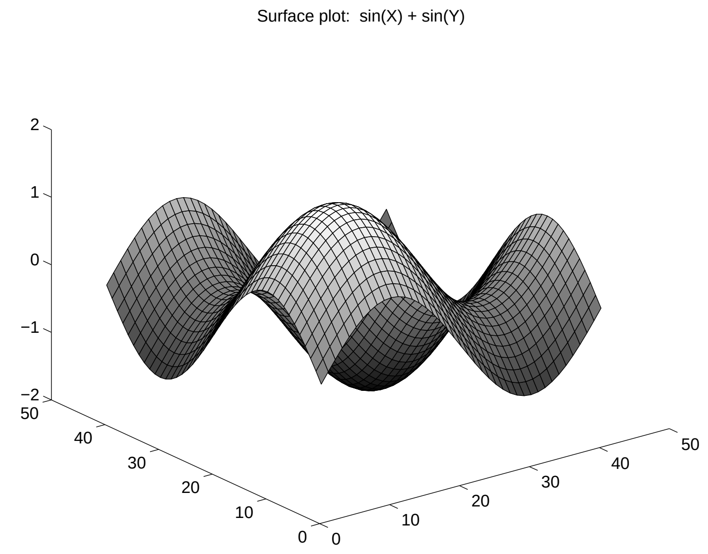


Figure 10: Shaded surface plot of $\sin X + \sin Y$.

8.5.2 Contour plots

The command to produce a contour plot of a surface works in much the same way as the `mesh` and `surf` commands. The plot in figure 12 is generated by the same script as in section 8.5.1, substituting the following lines for the original `mesh` and `title` commands.

```
contour(W2)
title('Contour plot:  sin(X) + sin(Y)')
```

8.6 Multiple plots

Often there are times when you need to observe more than one plot at the same time. You may wish to plot more than one function on the same plot. You may wish to have more than one plot in the same window. You may even wish to have more than one figure window containing plots.

8.6.1 Multiple functions in a plot

Suppose you want two curves plotted on the same graph as in figure 4. The script below graphs the sine and cosine curves together. This plot is shown in figure 13.

```
%  Plots both sine and cosine curves together

U = 0:pi/20:2*pi
W = sin(U)
Z = cos(U)
plot(U,W,U,Z)
```

The pattern illustrated here holds true in general, and the vector of abscissae (`U` in this example) need not be the same in each case. Thus

```
plot(X1,Y1,X2,Y2,X3,Y3)
```

plots the three curves $(f(X1,Y1)$, $g(X2,Y2)$, $h(X3,Y3))$ in the same plane.

An alternate method for putting one plot on top of another is to use the `hold` command. This tells MATLAB not to erase the contents of the figure window until the command

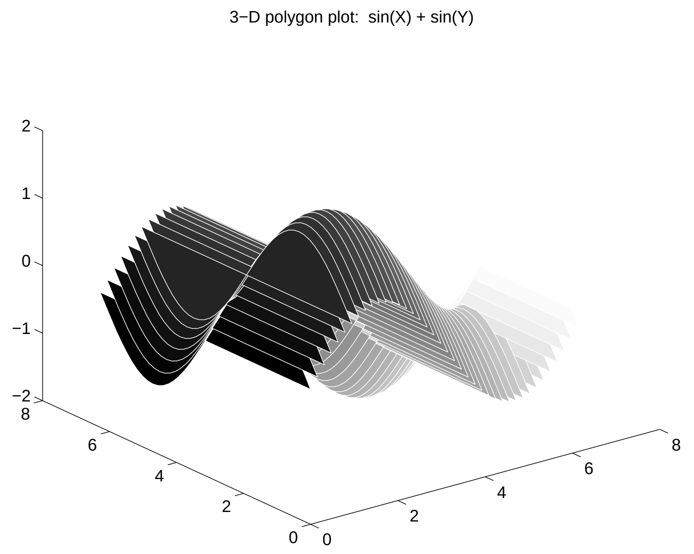


Figure 11: Three-dimensional polygon plot of $\sin X + \sin Y$.

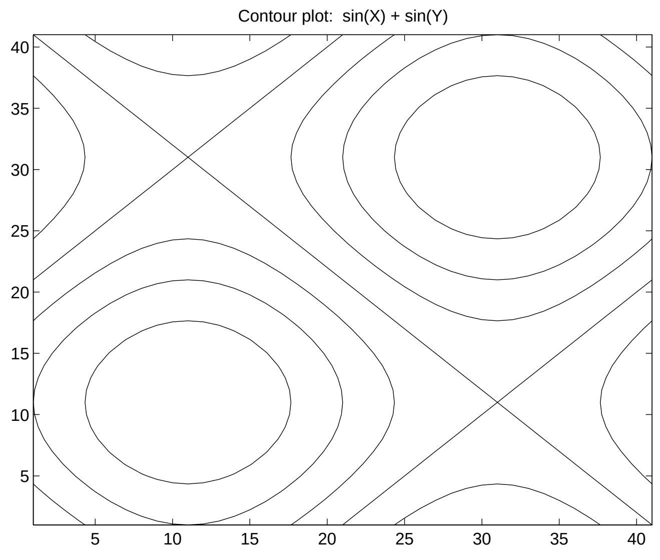


Figure 12: Contour plot of $\sin X + \sin Y$.

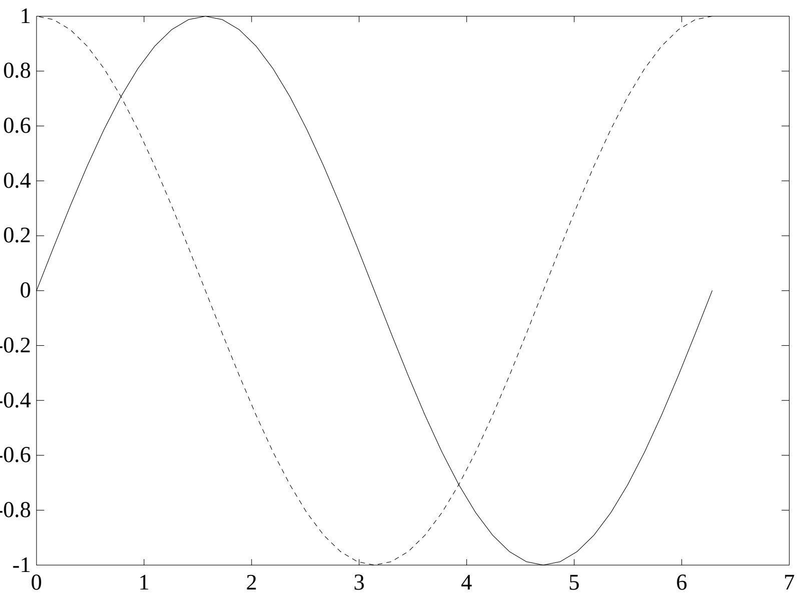


Figure 13: Plot of sine and cosine.

```
hold off
```

is entered. In this way, any number of plots can be placed within the same plane with the axes remaining constant.

Some built-in functions provide multiple plots. The **surfc** function combines a surface plot with a contour plot of the same three-dimensional object. Figure 14 shows the effect of this function on the same sine function used in figures 9 through 12.

8.6.2 Multiple plots in a window

Occasionally, it is useful to have more than one plot within in the figure window. This can be done; an example is shown in figure 15 of drawing both the mesh and contour figures shown in figures 9 and 12 in one window. The two plot statements that produced this example were preceded by the function named **subplot**, as follows

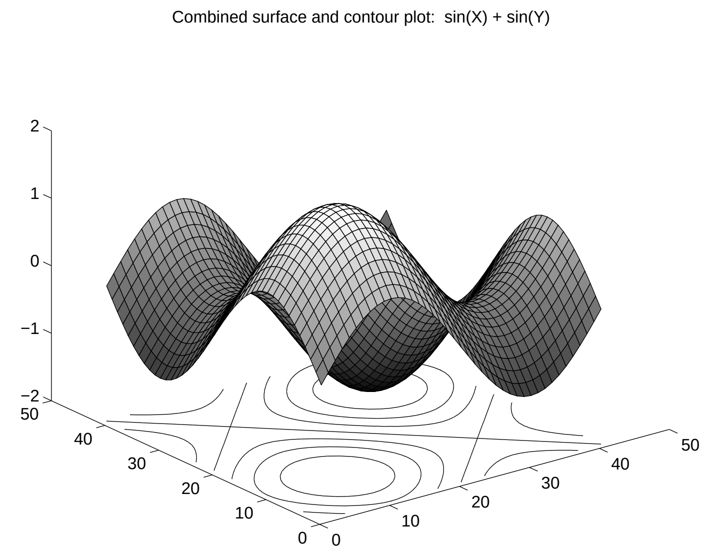


Figure 14: Combined shaded surface plot and contour plot of $\sin X + \sin Y$.

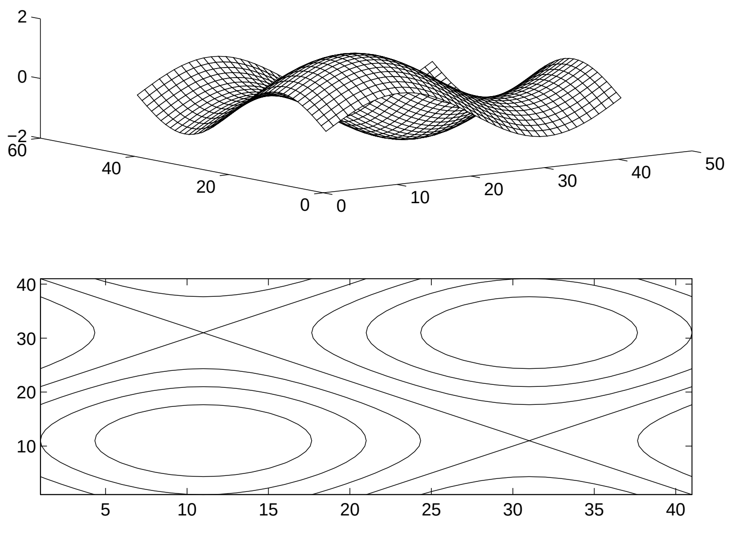


Figure 15: Subplots of mesh and contour plots.

```
subplot(2,1,1), mesh(W2)
subplot(2,1,2), contour(W2)
```

There are three numeric arguments to `subplot`; these are all positive numbers. The first number specifies the vertical number of plots desired in the figure window; the second specifies the horizontal number of plots. The last number tells in which subplot region the plotting command (`plot`, `mesh`, `contour`) is to plot.

Figure 16 contains four plots in a single figure window. The MATLAB script that produced these plots is below:

```
%  This script uses the 'sphere' and 'cylinder'
%  functions to produce subplots of a sphere and
%  the cylinder constructed with one of the
%  columns of the matrix defining its y-axis.

[Sxx, Syy, Szz] = sphere(16);
subplot(2,2,1), mesh(Sxx, Syy, Szz)
title('Mesh plot:  sphere(16)')

subplot(2,2,2), surfl(Sxx, Syy, Szz)
title('Shaded Surface plot:  sphere(16)')

[Cxx, Cyy, Czz] =  cylinder(2+Syy(8,:));
subplot(2,2,3), mesh(Cxx, Cyy, Czz)
title('Mesh plot: cylinder(2+Syy(8,:))')

subplot(2,2,4), surf(Cxx, Cyy, Czz)
title('Surface plot:  cylinder(2+Syy(8,:))')
```

There are two built-in functions used in this script: `sphere` and `cylinder`. The `sphere` function returns three matrices containing the coordinates of the sphere; in this case, each of the matrices will be of size 17×17. The `cylinder` function is similar, but we have specified the radius of the cylinder to be 2 plus the elements of the middle row of the matrix that defines the y-coordinates of the sphere.

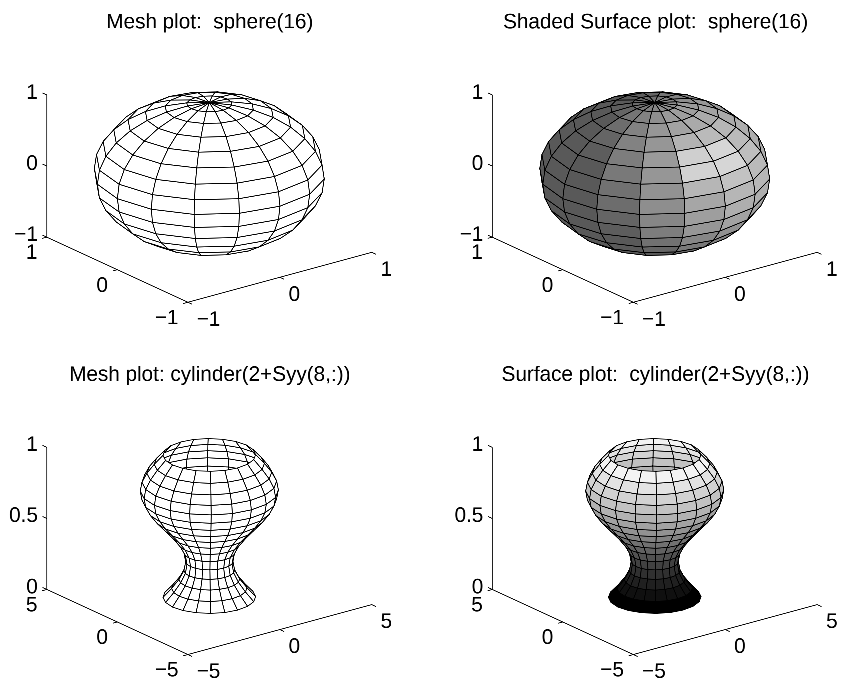

Mesh plot: sphere(16)
Shaded Surface plot: sphere(16)
Mesh plot: cylinder(2+Syy(8,:))
Surface plot: cylinder(2+Syy(8,:))
1
0
−1
0.5
5
−5

Figure 16: Subplots showing the mesh and shaded surface plots of a sphere determined by three 17×17 matrices of coordinates (named `Sxx`, `Syy`, and `Szz`) and the mesh and surface plots of a cylinder whose width is two plus the y coordinates of the sphere.

8.6.3 Multiple figure windows

The `figure` command is used to create additional figure windows. If you type

```
hnum = figure
```

a new figure window is activated, and the handle number of the new figure window is assigned to `hnum`. If you wish to make a new figure window with a particular handle (say 123), just type

```
figure(123)
```

The `figure` command can also be used to specify other parameters for the new window. These include the size of the window, the title of the window, the position of the window on the screen, and the background color. See the *MATLAB Reference Guide* [MathWorks 92a] or type

```
help figure
```

for more information.

8.7 Creating images

Besides producing various plots, it is possible to create an image in a figure window with MATLAB. These images may be colored by different color maps. Refer to the manuals or type

```
help image
help colormap
help ColorSpec
```

to find out more about these possibilities.

9 That's it!

Well, that is a brief overview of MATLAB. Be sure to do all of the examples given here while you are on the computer. This gets you started.

We have tried to make this short so that you would be able to quickly get into using MATLAB. On the other hand, this has forced us to leave out a lot of stuff that is potentially useful. Now it is up to you to learn more from the manuals and on-line documentation.

10 Acknowledgements

We would like to thank Jim Tung of The MathWorks, Inc. for his careful proofreading and many suggestions.

References

[MathWorks 92a] The MathWorks, Inc., Natick, MA. [Aug 1992]. *MATLAB: Reference Guide.*

[MathWorks 92b] The MathWorks, Inc., Natick, MA. [Aug 1992]. *MATLAB: User's Guide for UNIX Workstations.*

[Sigmon 94] SIGMON, KERMIT. [1994]. *MATLAB Primer.* CRC Press, Inc., Boca Raton, FL, 4th edition.