Elements of Matlab

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

19 August 1988
Revised
20 September 1993

High Performance Scientific Computing
University of Colorado at Boulder

Contents

Elements of Matlab*

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

19 August 1988
Revised
20 September 1993

1 What is Matlab?

Matlab[1] 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. X-Y plots on the screen or printer can be done easily in Matlab.

It is often used interactively as if it were a very powerful hand calculator. But you can also use Matlab in a programmable mode; you just write scripts for it much as you do for other command languages. You can also create your own functions which can be invoked interactively or from scripts.

The examples in this document were run on *UNIX*[2] workstations; both a SUN 3/60 under the SunView[3] window environment and a DECstation[4] 5000/200 under the X Window System[5] were used. A basic knowledge of *UNIX* is assumed for the remainder of this text.

*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`.

[1]Matlab is a trademark of The MathWorks, Inc.

[2]UNIX is a trademark of AT&T.

```
% matlab

                    < P R O - M A T L A B >
               (c) Copyright  The MathWorks, Inc.  1984-1991
                         All Rights Reserved
                      Version 3.5e    24-Jul-1991

           HELP, DEMO, INFO, and TERMINAL are available
>> ...
   ⋮
>> quit

 0 flop(s).

%
```

Figure 1: Matlab window – initial state.

2 Getting Started

These notes are intended to get you started, providing only the bare essentials. The Matlab manual [MW9 90] is the basic reference.

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. This can be done by typing the command

[3]SUN and SunView are trademarks of Sun Microsystems, Inc.

[4]DEC is a trademark of Digital Equipment Corporation.

[5]X Window System is a trademark of M.I.T.

setenv DISPLAY *yourterminalname*:0

from the *UNIX* shell. Without this, the plot window will not appear on your screen.

Once you are in the directory from which you wish to use Matlab, you can start it by typing the command

```
matlab
```

from the shell. The basic Matlab environment will be activated, and your window should appear as shown in Fig. 1.

2.2 Standard Help

Notice the message on the initial Matlab window:

```
HELP, DEMO, INFO, and TERMINAL are available
```

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* will be 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 graphic terminals capable of running Matlab.

Typing `demo` brings a list of possible Matlab demonstrations to the screen. You only need to type the demo number and follow the instructions in the window to see some pretty plots. Try a few 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.

3.1 Simple Matrix Manipulation

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

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

The matrices created 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
```

create the matrices

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

Notice that the multiplication operator, `*`, used with two matrices produces correct matrix multiplication.

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.

3.2 Polynomial Curve Fitting

In the preceding 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'd like 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)

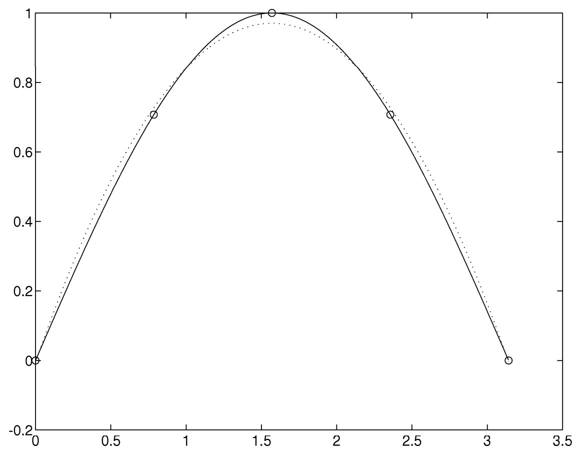


Figure 2: A cubic fit to the sine function on $[0, \pi]$.

finds the coefficients

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

of the cubic approximating the given data y in the least squares sense. Typing

pvals = polyval(p,x)

then evaluates that cubic at the given values of x.

Figure 2 shows the the sine function on the interval $[0, \pi]$ with the sampled points marked by circles. The least squares cubic is shown by the dotted line.

3.3 Simple Plots

The statements

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

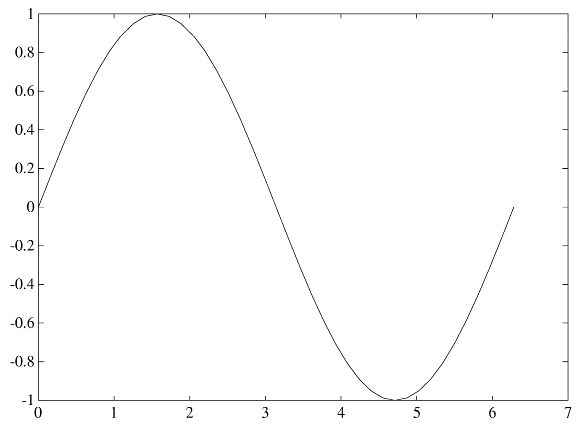


Figure 3: Plot of sine function on $[0, 2\pi]$.

create a plot of the sine function on the interval $[0, 2\pi]$ as shown in Fig. 3. 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 creates a vector of 41 values, equal to the sines of the 41 values in `U`. The last statement creates 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 `V`. Note that the name `pi` in a Matlab statement denotes a constant equal to π.

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 window will remain until you exit (with the `quit` command) your Matlab session. Any additional plotting commands will reuse this plot window. On most windowing systems, it can be closed, reopened, moved, or resized, as any other window.

If you are using an X terminal, the mesh grid outlining the Matlab plot 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.

4 Short Outline of the Language

This section provides information about the basic syntax and semantics for Matlab commands.

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
```

4.4 Display Format

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

```
 x = 32.75
```

Matlab will respond with

```
x =
   32.7500
```

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

```
 format short
 x

x =
   32.7500
```

The `long` format has fourteen decimal places.

```
 format long
 x

x =
  32.75000000000000
```

The remaining two 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 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]
```

which is 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]
```

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.

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

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

Type the statement

```
M = [1:1:4;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 add, subtract, multiply, right divide, left divide, 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.

Add, subtract, multiply, and exponentiation have their usual meanings when applied to matrices, vectors, and scalars. The left divide operator and the right divide operator 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 adding and subtracting vectors or matrices, there will be no change in the operation. However, consider the following example in which we assume `A` and `B` to be the 2×2 matrices defined earlier:

```
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 manual [MW9 90] 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; e.g., if you try to do something stupid like multiply a 3×3 matrix by a 4×4 matrix, then Matlab will squawk at you.

Normally, each line you write is an assignment statement as in the examples above. However there are exceptions, as in the call to the plot subroutine which appeared above, and control statements which we describe briefly below.

When you have completed typing in an assignment statement, you will get an echo on the screen which 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
```

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 meaningful: 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 Array References

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

You also can refer to rows of an array, columns of an array, 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 Fig. 4.

4.11 Relational and Logical Operators

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

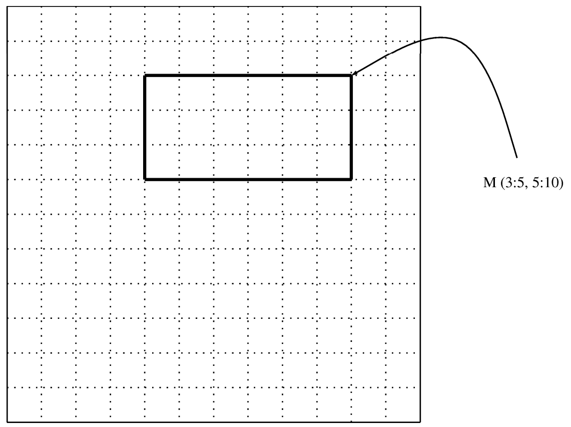


Figure 4: Submatrix of 12 x 12 matrix M.

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.

4.11.2 Logical operators

Relational expressions can be combined using the Matlab logical operators:

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

which mean `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.

4.12 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*, which are programs made up of Matlab commands.

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.

4.12.1 `for` statement

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

The `for` statement may be nested and a constructor with an arbitrary step can be used.

```
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
```

4.12.2 `while` statement

```
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).

4.12.3 `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
```

4.12.4 Further help

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

```
help if
```

Then try

```
help break
```

4.13 Built-in Functions

There are many built-in functions and you can peruse the manuals to see what is available. You can also type

```
help
```

in Matlab. This will give you a list of built-in functions. Then if you want more information on a particular built-in function, like `ones`, just type

```
help ones
```

The usual math functions, like the trig functions, are built-in. Some of the functions save a great deal of work. For instance, the `eig` function provides the eigenvalues and eigenvectors of a matrix argument. There are other built-in functions that are rather special. An example is the function, `ones`, which can be used to create an array of ones; thus the expression

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

creates 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.

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

5 Matlab Scripts and User-Defined Functions

As mentioned earlier, it is possible to write programs for Matlab. These are called Matlab *scripts* or *functions* and should be stored as files with a `.m` extension, e.g., `myscript.m` or `myfunction.m`. Because of this extension, these scripts 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.

5.1 A Sample Script

The following is a simple script which will plot a cosine curve in Matlab.

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

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

To use a script within Matlab, just type the script file name without the `.m` extension, i.e., `myscript`; the script (`myscript.m`) will be executed. If this example script has been stored in a file named `plotcos.m` in the directory from which you are running Matlab, you need only type

```
plotcos
```

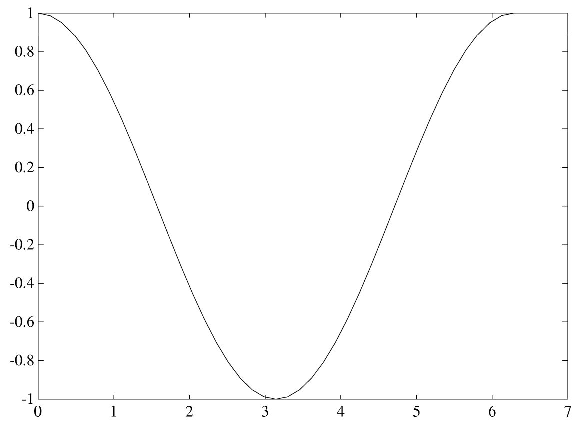


Figure 5: Plot of cosine function on $[0, 2\pi]$.

to run the script, and the appropriate plot will appear in your plot window, as shown in Fig. 5. Also the global values of `U` and `V` will be affected.

5.2 A Sample Function

Like a script, a function is a collection of Matlab commands stored in an M-file. Unlike a script, a function itself 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, i.e., `myfunction`, or it can be assigned directly to a variable, i.e., `y = myfunction`. The value of an array-valued function must be assigned to an array variable. User-defined functions may be used not only on the command line 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 `myftn`, the file containing the Matlab commands which define that function must be stored as `myftn.m`.

A function may require input arguments. In this case, after 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 = myscript(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
%    that evaluates 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 will 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.

5.3 Comments

The percent symbol, `%`, precedes a comment:

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

The % 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(X,Y)    % This is a plot command
```

It is useful to place explanatory comments at the beginning of your Matlab script as documentation. Later, if you type the command

```
help myscript
```

Matlab responds by printing out the first few lines of comments in the script named `myscript`.

5.4 White Space

Blank lines may be inserted in a Matlab script; this will provide white space and promote the readability of the script.

5.5 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 Input/Output

This section discusses methods for creating input data for Matlab as well as ways to output the data.

6.1 *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 edited to form *M-files* to create plots or other data in Matlab.

6.2 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 will be saved into a file named `mylogfile`. If the filename is omitted, the name `diary` will be 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 which can be later edited into another format.

6.3 Saving Data

An alternate method of 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 will be in text format. Then the file, `mydatafile`, will contain 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
```

6.4 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 manual [MW9 90].

7 Graphics

The sample script given in Section 5.1 illustrated how to create a plot of the cosine function, as shown in Fig. 5. This plot is only given in the Matlab plot window, but there are times where you would like to save such plots for use elsewhere.

7.1 Hardcopy Plots

When the `plot` statement in the script, `plotcos.m`, is executed, the plot of the cosine function appears in the plot window. This is the graphics window for all plots; so any later plots will erase the cosine plot. However, if you type

```
     print
```

on some systems, a hardcopy of the current plot in the graphics window will be printed on your default printer.

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

```
     meta myplot
```

puts the current plot in a file named `myplot.met`. Repeating this command after a second plot will concatenate the second plot to the first by appending it to the *met-file*, `myplot.met`; it will not destroy the original copy.

It is possible to convert the plot into a *PostScript* [6] file or a *pic* file by using the *UNIX* `gpp` (*graphic post-processor*) command. From the *UNIX* shell, the command

```
     gpp myplot -dps
```

transforms the file, `myplot.met`, into a PostScript file named `myplot.ps`. This can be printed out from the shell with the command

```
     lpr myplot.ps
```

or can later be included in the text of a paper. The result of the command

```
     gpp myplot -dpic
```

is a pic file named `myplot.pic`. As such, the graph can be included in the text of a paper in `troff` format. Many other options are available for this command. Refer to the section on *UNIX* in the Matlab manual [MW9 90] for more information.

[6]PostScript is a trademark of Adobe Systems, Inc.

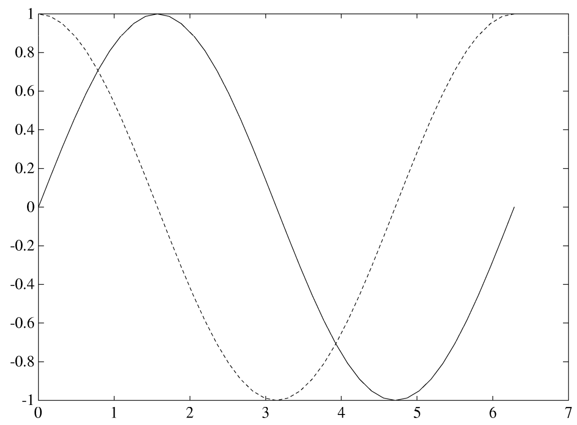


Figure 6: Plot of sine and cosine.

7.2 Multiple Plots

Suppose you want two curves plotted on the same graph. The script below will graph the sine and cosine curves together. This plot is shown in Fig. 6.

```
%  Plots both sine and cosine curves together

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

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

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

will plot the three curves ($f(X1, Y1)$, $f(X2, Y2)$, $f(X3, Y3)$) in the same plane.

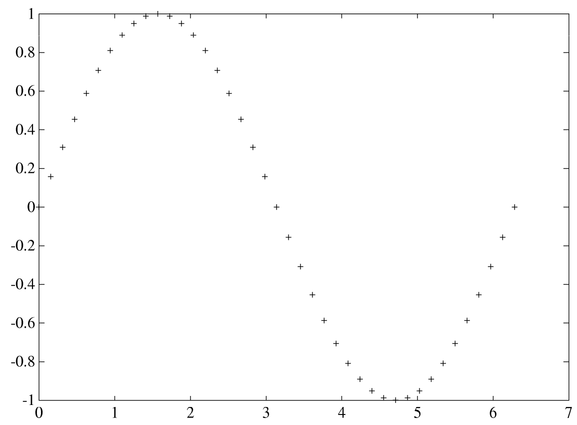


Figure 7: Point plot of sine function.

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

```
hold off
```

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

7.3 Types of X-Y Plots

There are two types of linear X-Y plots: *line* and *point*. Three-dimensional grids and contour plots are also available; these will be discussed in a later section. Polar, logarithmic, semi-log, and bar plots can be employed as well; see the Matlab manual [MW9 90] or type

```
help
```

for more information on these types of plots.

In the graphical examples discussed earlier in this document, the *line* type of X-Y plot is used. The other type is *point*. With the line type, 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,V,'+')
```

The sine curve will appear as 41 distinct points (marked "`+`"), as shown in Fig. 7. The statement

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

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.

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

```
help plot
```

for that.

7.4 Labeling Plots

The picture can be labeled, the axes can be labeled, and you can display grid lines in the coordinate system. The following script will do all of this for the previous sine plot:

```
%  This Matlab script plots a sine curve

U = 0:pi/20:2*pi
```

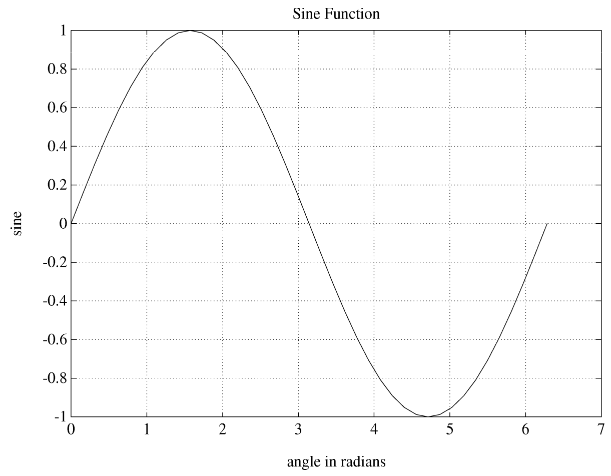


Figure 8: Labeled plot of sine function.

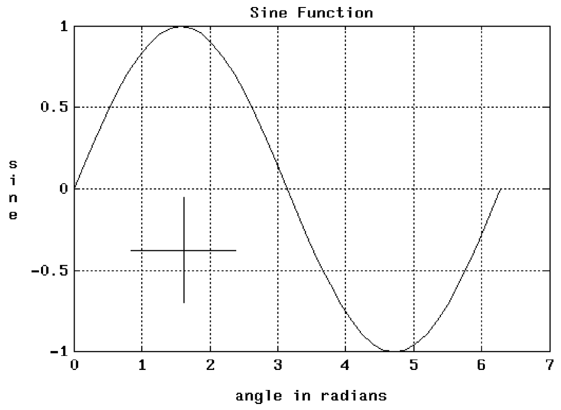
```
V = cos(U)
plot(U,V)

title('Sine Function')          % places title at top
xlabel('angle in radians')      % labels x-axis
ylabel('sine')                  % labels y-axis
grid                            % adds grid marking
```

Notice that the labeling commands are given after the plot has been created. The result is shown in Fig. 8.

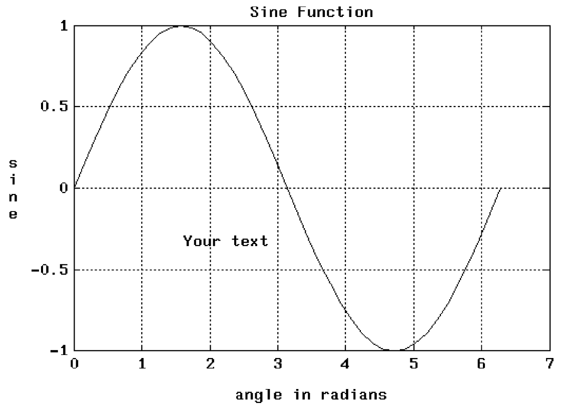
It is also possible to place text on the graph while it is in the plot window by using the mouse. The command

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

makes a crosshair appear on the window containing the plot, as in Fig. 9(a). Just move the mouse to the desired location and click; the label containing



(a)



(b)

Figure 9: Sine function plot: (a) with `gtext` crosshair; and (b) with label entered by `gtext`.

Your text will appear there, with the first letter placed in the inside corner of the northeast quadrant of the crosshair. The final result is in Fig. 9(b).

7.5 Three-Dimensional Plotting

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

7.5.1 Three-dimensional grids

Mesh plots show a three-dimensional surface as a mesh. 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 points of the surface – one value for each xy point. Thus, the only needed argument to the `mesh` command is that matrix.

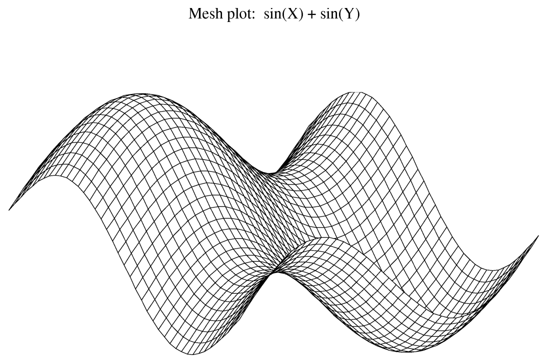


Figure 10: Mesh plot of $\sin X + \sin Y$.

The plot in Fig. 10 was created by the following script:

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

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

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

7.5.2 Contour plots

The command to create a contour plot of a surface works in much the same way as the `mesh` command. The plot in Fig. 11 was created by the same script

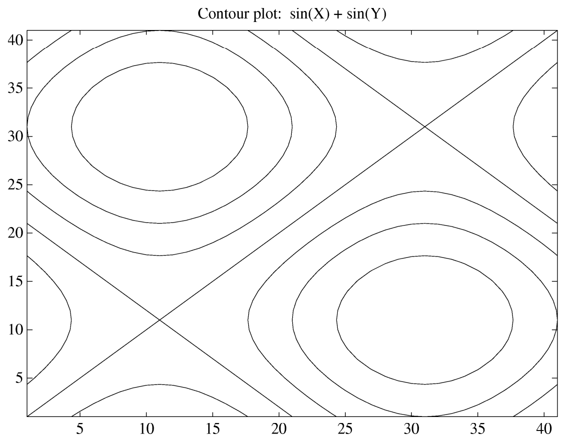


Figure 11: Contour plot of $\sin X + \sin Y$.

as in Section 7.5.1, substituting the following lines for the original `mesh` and `title` commands.

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

7.6 Subplots

Occasionally, it is useful to have more than one plot shown separately in the plot window. This can be done; an example is shown in Fig. 12. The two plot statements which produced this example were preceded by the function named `subplot`, as follows

```
subplot(211), mesh(Z2)
subplot(212), contour(Z2)
```

The numeric argument to `subplot` is made up of three digits. The first number specifies the vertical number of plots desired in the plot window;

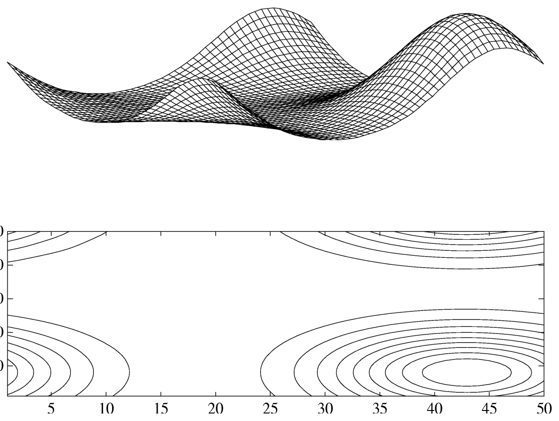


Figure 12: Subplots of mesh and contour plots.

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.

Example 13 shows three of four plots in a single plot window. The Matlab script that produced this example is below:

```
%  This script produces some sine and cosine plots

U = 0:pi/20:2*pi
V = sin(U)
W = cos(U)

subplot(221), plot(U,V)
subplot(222), plot(U,W)
subplot(223), plot(V,W)
```

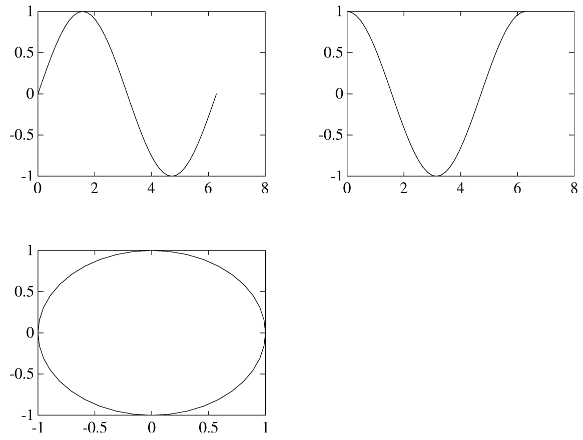


Figure 13: Subplots of sine and cosine plots.

8 That's it!

Well, that is a brief overview of Matlab. Now go back over this material while you are on the computer and do all of the examples given here. This will get 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 manual.

References

[MW9 90] The MathWorks, Inc., South Natick, MA. [Feb 1990]. *Matlab for Unix Computers.*

[Sigmon 89] SIGMON, KERMIT. [1989]. MATLAB primer. Technical report, University of Florida, Gainesville, FL.