MATLAB Primer

Second Edition

Kermit Sigmon
Department of Mathematics
University of Florida

Department of Mathematics • University of Florida • Gainesville, FL 32611
sigmon@math.ufl.edu • sigmon@ufpine.bitnet

Department of Mathematics • University of Florida • Gainesville, FL 32611
sigmon@math.ufl.edu • sigmon@ufpine.bitnet

Introduction

MATLAB is an interactive, matrix-based system for scientific and engineering calculations. You can solve complex numerical problems without actually writing a program. The name MATLAB is an abbreviation for MATrix LABoratory.

The purpose of this Primer is to help you begin to use MATLAB. They can best be used hands-on. You are encouraged to work at the computer as you read the Primer and freely experiment with examples.

You should liberally use the on-line help facility for more detailed information. After entering MATLAB as described in section 1, the command `help` will display a list of functions for which on-line help is available; the command `help` *functionname* will give information about a specific function. The command `help eig`, for example, will give information about the eigenvalue function `eig`. You can preview some of the features of MATLAB by entering the command `demo`.

The scope and power of MATLAB go far beyond these notes. Eventually you will want to consult the MATLAB User's Guide and Reference Guide. The documentation which accompanies the Student Edition of MATLAB is an excellent source. Copies of the complete User's Guide are often available for review at locations such as consulting desks, terminal rooms, computing labs, and the reserve desk of the library. Consult your instructor or your local computing center to learn where the User's Guides are located at your institution.

MATLAB is available for a number of environments: Sun/Apollo/VAXstation/HP workstations, VAX, MicroVAX, Gould, PC and AT compatibles, 80386 and 80486 computers, Apple Macintosh, and several parallel machines. There is a relatively inexpensive Student Edition available from Prentice Hall publishers. The information in these notes applies generally to all of these environments.

While the second edition of the Primer is based on version 3.5 of MATLAB, it is compatible with version 4.0 with the fundamental differences noted. An edition of the Primer based on version 4.0 is under development.

The plain TEX source (and a PostScript file `primer35.ps`) of this second edition of the MATLAB Primer is available via anonymous ftp from `math.ufl.edu` as the file `primer35.tex` in the directory `pub/matlab`. If ftp is unavailable to you, it can be obtained via `listserv` by sending an email message to `listserv@math.ufl.edu` which contains the single line `send matlab/primer35.tex`. It can also be obtained by sending a request to the author at `sigmon@math.ufl.edu`. The latest edition of the Primer will always be available as indicated above as the file `primer.tex`. A Spanish version is also available there.

MATLAB is licensed by The MathWorks, Inc., Cochituate Place, 24 Prime Park Way, Natick, MA 01760, (508)653-1415, Fax: (508)653-2997, Email: info@mathworks.com.

(6-93)

Contents

Page

1. Accessing MATLAB.

On most systems, after logging in one can enter MATLAB with the system command `matlab` and exit MATLAB with the command `exit` or `quit`. On a PC, for example, if properly installed, one may enter MATLAB with the command:

```
C> matlab
```

and exit it with the command:

```
>> quit
```

On systems permitting multiple processes, such as a Unix system, you will find it convenient, for reasons discussed in section 14, to keep both MATLAB and your local editor active. If you are working on a workstation which runs processes in multiple windows, you will want to keep MATLAB active in one window and your local editor active in another. You should consult your instructor or your local computer center for details of the local installation.

2. Entering matrices.

MATLAB works with essentially only one kind of object—a rectangular numerical matrix with possibly complex entries; all variables represent matrices. In some situations, 1-by-1 matrices are interpreted as scalars and matrices with only one row or one column are interpreted as vectors.

Matrices can be introduced into MATLAB in several different ways:

- Entered by an explicit list of elements,
- Generated by built-in statements and functions,
- Created in M-files (see sections 12 and 14 below),
- Loaded from external data files (see User's Guide).

For example, either of the statements

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

and

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

creates the obvious 3-by-3 matrix and assigns it to a variable A. Try it. The elements within a row of a matrix may be separated by commas as well as a blank.

When listing a number in exponential form (e.g. 2.34e-9), blank spaces must be avoided. Listing entries of a large matrix is best done in an M-file, where errors can be easily edited away (see sections 12 and 14).

The built-in functions `rand`, `magic`, and `hilb`, for example, provide an easy way to create matrices with which to experiment. The command `rand(n)` will create an $n \times n$ matrix with randomly generated entries distributed uniformly between 0 and 1, while `rand(m,n)` will create an $m \times n$ one. `magic(n)` will create an integral $n \times n$ matrix which is a magic square (rows and columns have common sum); `hilb(n)` will create the $n \times n$ Hilbert matrix, the king of ill-conditioned matrices (m and n denote, of course, positive integers). Matrices can also be generated with a for-loop (see section 6 below).

Individual matrix and vector entries can be referenced with indices inside parentheses in the usual manner. For example, $A(2,3)$ denotes the entry in the second row, third column of matrix A and $x(3)$ denotes the third coordinate of vector x. Try it. A matrix or a vector will only accept *positive* integers as indices.

3. Matrix operations, array operations.

The following matrix operations are available in MATLAB:

$+$	addition
$-$	subtraction
$*$	multiplication
$\widehat{\ }$	power
$'$	transpose
$\backslash$	left division
$/$	right division

These matrix operations apply, of course, to scalars (1-by-1 matrices) as well. If the sizes of the matrices are incompatible for the matrix operation, an error message will result, except in the case of scalar-matrix operations (for addition, subtraction, and division as well as for multiplication) in which case each entry of the matrix is operated on by the scalar.

The "matrix division" operations deserve special comment. If A is an invertible square matrix and b is a compatible column, resp. row, vector, then

$x = A\backslash b$ is the solution of $A * x = b$ and, resp.,
$x = b/A$ is the solution of $x * A = b$.

In left division, if A is square, then it is factored using Gaussian elimination and these factors are used to solve $A * x = b$. If A is not square, it is factored using Householder orthogonalization with column pivoting and the factors are used to solve the under- or over- determined system in the least squares sense. Right division is defined in terms of left division by $b/A = (A'\backslash b')'$.

Array operations. The matrix operations of addition and subtraction already operate entry-wise but the other matrix operations given above do not—they are *matrix* operations. It is important to observe that these other operations, $*$, $\widehat{\ }$, $\backslash$, and $/$, can be made to operate entry-wise by preceding them by a period. For example, either `[1,2,3,4].*[1,2,3,4]` or `[1,2,3,4].^2` will yield `[1,4,9,16]`. Try it. This is particularly useful when using Matlab graphics.

4. Statements, expressions, and variables; saving a session.

MATLAB is an *expression* language; the expressions you type are interpreted and evaluated. MATLAB statements are usually of the form

variable = *expression*, or simply
expression

Expressions are usually composed from operators, functions, and variable names. Evaluation of the expression produces a matrix, which is then displayed on the screen and assigned to the variable for future use. If the variable name and = sign are omitted, a variable `ans` (for answer) is automatically created to which the result is assigned.

A statement is normally terminated with the carriage return. However, a statement can be continued to the next line with three or more periods followed by a carriage return. On the other hand, several statements can be placed on a single line if separated by commas or semicolons.

If the last character of a statement is a semicolon, the printing is suppressed, but the assignment is carried out. This is essential in suppressing unwanted printing of intermediate results.

MATLAB is case-sensitive in the names of commands, functions, and variables. For example, `solveUT` is not the same as `solveut`.

The command `who` will list the variables currently in the workspace. A variable can be cleared from the workspace with the command `clear` *variablename*. The command `clear` alone will clear all nonpermanent variables.

The permanent variable `eps` (epsilon) gives the machine precision—about 10^{-16} on most machines. It is useful in determining tolerences for convergence of iterative processes.

A runaway display or computation can be stopped on most machines without leaving MATLAB with CTRL-C (CTRL-BREAK on a PC).

Saving a session. When one logs out or exits MATLAB all variables are lost. However, invoking the command `save` before exiting causes all variables to be written to a non-human-readable diskfile named `matlab.mat`. When one later reenters MATLAB, the command `load` will restore the workspace to its former state.

5. Matrix building functions.

Convenient matrix building functions are

eye	identity matrix
zeros	matrix of zeros
ones	matrix of ones
diag	see below
triu	upper triangular part of a matrix
tril	lower triangular part of a matrix
rand	randomly generated matrix
hilb	Hilbert matrix
magic	magic square
toeplitz	see `help toeplitz`

For example, `zeros(m,n)` produces an m-by-n matrix of zeros and `zeros(n)` produces an n-by-n one; if A is a matrix, then `zeros(A)` produces a matrix of zeros of the same size as A.

If x is a vector, `diag(x)` is the diagonal matrix with x down the diagonal; if A is a square matrix, then `diag(A)` is a vector consisting of the diagonal of A. What is `diag(diag(A))`? Try it.

Matrices can be built from blocks. For example, if A is a 3-by-3 matrix, then

```
B = [A, zeros(3,2); zeros(2,3), eye(2)]
```

will build a certain 5-by-5 matrix. Try it.

6. For, while, if — and relations.

In their basic forms, these MATLAB flow control statements operate like those in most computer languages.

For. For example, for a given n, the statement

```
x = []; for i = 1:n, x=[x,i^2], end
```

or

```
x = [];
for i = 1:n
   x=[x,i^2]
end
```

will produce a certain n-vector and the statement

```
x = []; for i = n:-1:1, x=[x,i^2], end
```

will produce the same vector in reverse order. Try them. Note that a matrix may be empty (such as `x = []`). The statements

```
for i = 1:m
   for j = 1:n
      H(i, j) = 1/(i+j-1);
   end
end
H
```

will produce and print to the screen the m-by-n hilbert matrix. The semicolon on the inner statement suppresses printing of unwanted intermediate results while the last `H` displays the final result.

While. The general form of a `while` loop is

```
while relation
   statements
end
```

The statements will be repeatedly executed as long as the relation remains true. For example, for a given number a, the following will compute and display the smallest nonnegative integer n such that $2^n \geq a$:

```
n = 0;
while 2^n < a
   n = n + 1;
end
n
```

If. The general form of a simple `if` statement is

```
if relation
   statements
end
```

The statements will be executed only if the relation is true. Multiple branching is also possible, as is illustrated by

```
if n < 0
   parity = 0;
elseif rem(n,2) == 0
   parity = 2;
else
   parity = 1;
end
```

In two-way branching the elseif portion would, of course, be omitted.

Relations. The relational operators in MATLAB are

<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
==	equal
~=	not equal.

Note that "=" is used in an assignment statement while "==" is used in a relation. Relations may be connected or quantified by the logical operators

&	and
|	or
~	not.

When applied to scalars, a relation is actually the scalar 1 or 0 depending on whether the relation is true or false. Try `3 < 5`, `3 > 5`, `3 == 5`, and `3 == 3`. When applied to matrices of the same size, a relation is a matrix of 0's and 1's giving the value of the relation between corresponding entries. Try `a = rand(5)`, `b = triu(a)`, `a == b`.

A relation between matrices is interpreted by `while` and `if` to be true if each entry of the relation matrix is nonzero. Hence, if you wish to execute *statement* when matrices A and B are equal you could type

```
if A == B
   statement
end
```

but if you wish to execute *statement* when A and B are not equal, you would type

```
if any(any(A ~= B))
   statement
end
```

or, more simply,

```
if A == B else
   statement
end
```

Note that the seemingly obvious

```
if A ~= B, statement, end
```

will not give what is intended since *statement* would execute only if *each* of the corresponding entries of A and B differ. The functions `any` and `all` can be creatively used to reduce matrix relations to vectors or scalars. Two `any`'s are required above since `any` is a vector operator (see section 8).

The `for` statement permits any matrix to be used instead of `1:n`. See the User's Guide for details of how this feature expands the power of the `for` statement.

7. Scalar functions.

Certain MATLAB functions operate essentially on scalars, but operate element-wise when applied to a matrix. The most common such functions are

sin	asin	exp	abs	round
cos	acos	log (natural log)	sqrt	floor
tan	atan	rem (remainder)	sign	ceil

8. Vector functions.

Other MATLAB functions operate essentially on a vector (row or column), but act on an m-by-n matrix ($m \geq 2$) in a column-by-column fashion to produce a row vector containing the results of their application to each column. Row-by-row action can be obtained by using the transpose; for example, `mean(A')'`. A few of these functions are

max	sum	median	any
min	prod	mean	all
sort		std	

For example, the maximum entry in a matrix A is given by `max(max(A))` rather than `max(A)`. Try it.

9. Matrix functions.

Much of MATLAB's power comes from its matrix functions. The most useful ones are

eig	eigenvalues and eigenvectors
chol	cholesky factorization
svd	singular value decomposition
inv	inverse
lu	LU factorization
qr	QR factorization
hess	hessenberg form
schur	schur decomposition
rref	reduced row echelon form
expm	matrix exponential
sqrtm	matrix square root
poly	characteristic polynomial
det	determinant
size	size
norm	1-norm, 2-norm, F-norm, ∞-norm
cond	condition number in the 2-norm
rank	rank

MATLAB functions may have single or multiple output arguments. For example,

`y = eig(A)`, or simply `eig(A)`

produces a column vector containing the eigenvalues of A while

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

produces a matrix U whose columns are the eigenvectors of A and a diagonal matrix D with the eigenvalues of A on its diagonal. Try it.

10. Command line editing and recall.

The command line in MATLAB can be easily edited. The cursor can be positioned with the left/right arrows and the Backspace (or Delete) key used to delete the character to the left of the cursor. Other editing features are also available. On a PC try the Home, End, and Delete keys; on other systems see `help cedit` or `type cedit`.

A convenient feature is use of the up/down arrows to scroll through the stack of previous commands. One can, therefore, recall a previous command line, edit it, and execute the revised command line. For small routines, this is much more convenient that using an M-file which requires moving between MATLAB and the editor (see sections 12 and 14). For example, flopcounts (see section 15) for computing the inverse of matrices of various sizes could be compared by repeatedly recalling, editing, and executing

```
a = rand(8); flops(0), inv(a); flops
```

If one wanted to compare plots of the functions $y = \sin mx$ and $y = \sin nx$ on the interval $[0, 2\pi]$ for various m and n, one might do the same for the command line:

```
m=2; n=3; x=0:.01:2*pi; y=sin(m*x); z=cos(n*x); plot(x,y,x,z)
```

11. Submatrices and colon notation.

Vectors and submatrices are often used in MATLAB to achieve fairly complex data manipulation effects. "Colon notation" (which is used both to generate vectors and reference submatrices) and subscripting by vectors are keys to efficient manipulation of these objects. Creative use of these features permits one to minimize the use of loops (which slows MATLAB) and to make code simple and readable. *Special effort should be made to become familiar with them.*

The expression `1:5` (met earlier in `for` statements) is actually the row vector `[1 2 3 4 5]`. The numbers need not be integers nor the increment one. For example,

```
0.2:0.2:1.2
```

gives `[0.2, 0.4, 0.6, 0.8, 1.0, 1.2]`, and

`5:-1:1` gives `[5 4 3 2 1]`.

The following statements will, for example, generate a table of sines. Try it.

```
x = [0.0:0.1:2.0]';
y = sin(x);
[x y]
```

Note that since `sin` operates entry-wise, it produces a vector y from the vector x.

The colon notation can be used to access submatrices of a matrix. For example,

`A(1:4,3)` is the column vector consisting of the first four entries of the third column of A.

A colon by itself denotes an entire row or column:

`A(:,3)` is the third column of A, and `A(1:4,:)` is the first four rows.

Arbitrary integral vectors can be used as subscripts:

`A(:,[2 4])` contains as columns, columns 2 and 4 of A.

Such subscripting can be used on both sides of an assignment statement:

`A(:,[2 4 5]) = B(:,1:3)` replaces columns 2,4,5 of A with the first three columns of B. Note that the *entire* altered matrix A is printed and assigned. Try it.

Columns 2 and 4 of A can be multiplied on the right by the 2-by-2 matrix [1 2;3 4]:

```
A(:,[2,4]) = A(:,[2,4])*[1 2;3 4]
```

Once again, the entire altered matrix is printed and assigned.

If x is an n-vector, what is the effect of the statement `x = x(n:-1:1)`? Try it.

To appreciate the usefulness of these features, compare these MATLAB statements with a Pascal, FORTRAN, or C routine to effect the same.

12. M-files.

MATLAB can execute a sequence of statements stored on diskfiles. Such files are called "M-files" because they must have the file type of ".m" as the last part of their filename. Much of your work with MATLAB will be in creating and refining M-files.

There are two types of M-files: *script files* and *function files.*

Script files. A script file consists of a sequence of normal MATLAB statements. If the file has the filename, say, `rotate.m`, then the MATLAB command `rotate` will cause the statements in the file to be executed. Variables in a script file are global and will change the value of variables of the same name in the environment of the current MATLAB session.

Script files are often used to enter data into a large matrix; in such a file, entry errors can be easily edited out. If, for example, one enters in a diskfile `data.m`

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

then the MATLAB statement `data` will cause the assignment given in `data.m` to be carried out.

An M-file can reference other M-files, including referencing itself recursively.

Function files. Function files provide extensibility to MATLAB. You can create new functions specific to your problem which will then have the same status as other MATLAB functions. Variables in a function file are by default local. However, version 4.0 permits a variable to be declared global.

We first illustrate with a simple example of a function file.

```
function a = randint(m,n)
%RANDINT Randomly generated integral matrix.
%    randint(m,n) returns an m-by-n such matrix with entries
%    between 0 and 9.
a = floor(10*rand(m,n));
```

A more general version of this function is the following:

```
function a = randint(m,n,a,b)
%RANDINT Randomly generated integral matrix.
%    randint(m,n) returns an m-by-n such matrix with entries
%    between 0 and 9.
%    rand(m,n,a,b) return entries between integers a and b.
if nargin < 3, a = 0; b = 9; end
a = floor((b-a+1)*rand(m,n)) + a;
```

This should be placed in a diskfile with filename `randint.m` (corresponding to the function name). The first line declares the function name, input arguments, and output arguments; without this line the file would be a script file. Then a MATLAB statement `z = randint(4,5)`, for example, will cause the numbers 4 and 5 to be passed to the variables a and b in the function file with the output result being passed out to the variable z. Since variables in a function file are local, their names are independent of those in the current MATLAB environment.

Note that use of `nargin` ("number of input arguments") permits one to set a default value of an omitted input variable—such as a and b in the example.

A function may also have multiple output arguments. For example:

```
function [mean, stdev] = stat(x)
% STAT Mean and standard deviation
%    For a vector x, stat(x) returns the
%    mean and standard deviation of x.
%    For a matrix x, stat(x) returns two row vectors containing,
%    respectively, the mean and standard deviation of each column.
[m n] = size(x);
if m == 1
   m = n; % handle case of a row vector
end
mean = sum(x)/m;
stdev = sqrt(sum(x.^2)/m - mean.^2);
```

Once this is placed in a diskfile `stat.m`, a MATLAB command `[xm, xd] = stat(x)`, for example, will assign the mean and standard deviation of the entries in the vector x to xm and xd, respectively. Single assignments can also be made with a function having multiple output arguments. For example, `xm = stat(x)` (no brackets needed around xm) will assign the mean of x to xm.

The `%` symbol indicates that the rest of the line is a comment; MATLAB will ignore the rest of the line. However, the first few comment lines, which document the M-file, are available to the on-line help facility and will be displayed if, for example, `help stat` is entered. Such documentation should *always* be included in a function file.

This function illustrates some of the MATLAB features that can be used to produce efficient code. Note, for example, that `x.^2` is the matrix of squares of the entries of x, that `sum` is a vector function (section 8), that `sqrt` is a scalar function (section 7), and that the division in `sum(x)/m` is a matrix-scalar operation.

The following function, which gives the greatest common divisor of two integers via the Euclidean algorithm, illustrates the use of an error message (see the next section).

```
function a = gcd(a,b)
% GCD Greatest common divisor
%    gcd(a,b) is the greatest common divisor of
%    the integers a and b, not both zero.
a = round(abs(a)); b = round(abs(b));
if a == 0 & b == 0
   error('The gcd is not defined when both numbers are zero')
else
   while b ~= 0
      r = rem(a,b);
      a = b; b = r;
   end
end
```

Some more advanced features are illustrated by the following function. As noted earlier, some of the input arguments of a function—such as *tol* in the example, may be made optional through use of `nargin` ("number of input arguments"). The variable `nargout` can be similarly used. Note that the fact that a relation is a number (1 when true; 0 when false) is used and that, when `while` or `if` evaluates a relation, "nonzero" means "true" and 0 means "false". Finally, the MATLAB function `feval` permits one to have as an input variable a string naming another function.

```
function [b, steps] = bisect(fun, x, tol)
%BISECT Zero of a function of one variable via the bisection method.
%    bisect(fun,x) returns a zero of the function.  fun is a string
%    containing the name of a real-valued function of a single
%    real variable; ordinarily functions are defined in M-files.
%    x is a starting guess.  The value returned is near a point
%    where fun changes sign.  For example,
%    bisect('sin',3) is pi.  Note the quotes around sin.
%
%    An optional third input argument sets a tolerence for the
%    relative accuracy of the result.  The default is eps.
%    An optional second output argument gives a matrix containing a
%    trace of the steps; the rows are of form [c f(c)].

% Initialization
if nargin < 3, tol = eps; end
trace = (nargout == 2);
if x ~= 0, dx = x/20; else, dx = 1/20; end
a = x - dx; fa = feval(fun,a);
b = x + dx; fb = feval(fun,b);

% Find change of sign.
while (fa > 0) == (fb > 0)
   dx = 2.0*dx;
   a = x - dx; fa = feval(fun,a);
   if (fa > 0) ~= (fb > 0), break, end
   b = x + dx; fb = feval(fun,b);
end
if trace, steps = [a fa; b fb]; end

% Main loop
while abs(b - a) > 2.0*tol*max(abs(b),1.0)
   c = a + 0.5*(b - a); fc = feval(fun,c);
   if trace, steps = [steps; [c fc]]; end
   if (fb > 0) == (fc > 0)
      b = c; fb = fc;
   else
```

```
        a = c; fa = fc;
     end
  end
```

Some of MATLAB's functions are built-in while others are distributed as M-files. The actual listing of any M-file—MATLAB's or your own—can be viewed with the MATLAB command `type` *functionname*. Try entering `type eig`, `type vander`, and `type rank`.

13. Text strings, error messages, input.

Text strings are entered into MATLAB surrounded by single quotes. For example,

```
s = 'This is a test'
```

assigns the given text string to the variable `s`.

Text strings can be displayed with the function `disp`. For example:

```
disp('this message is hereby displayed')
```

Error messages are best displayed with the function `error`

```
error('Sorry, the matrix must be symmetric')
```

since when placed in an M-File, it causes execution to exit the M-file.

In an M-file the user can be prompted to interactively enter input data with the function `input`. When, for example, the statement

```
iter = input('Enter the number of iterations:  ')
```

is encountered, the prompt message is displayed and execution pauses while the user keys in the input data. Upon pressing the return key, the data is assigned to the variable `iter` and execution resumes.

14. Managing M-files.

While using MATLAB one frequently wishes to create or edit an M-file and then return to MATLAB. One wishes to keep MATLAB active while editing a file since otherwise all variables would be lost upon exiting.

This can be easily done using the !-feature. If, while in MATLAB, you precede it with an !, any system command—such as those for editing, printing, or copying a file—can be executed without exiting MATLAB. If, for example, the system command `ed` accesses your editor, the MATLAB command

```
>> !ed rotate.m
```

will let you edit the file named `rotate.m` using your local editor. Upon leaving the editor, you will be returned to MATLAB just where you left it.

As noted in section 1, on systems permitting multiple processes, such as one running Unix, it may be preferable to keep both MATLAB and your local editor active, keeping one process suspended while working in the other. If these processes can be run in multiple windows, as on a workstation, you will want to keep MATLAB active in one window and your editor active in another.

You may consult your instructor or your local computing center for details of the local installation.

Version 4.0 has many debbugging tools. See `help dbtype` and references given there.

When in MATLAB, the command `dir` will list the contents of the current directory while the command `what` will list only the M-files in the directory. The MATLAB commands `delete` and `type` can be used to delete a diskfile and print a file to the screen, respectively, and `chdir` can be used to change the working directory. While these commands may duplicate system commands, they avoid the use of an !.

M-files must be accessible to MATLAB. On most mainframe or workstation network installations, personal M-files which are stored in a subdirectory of one's home directory named `matlab` will be accessible to MATLAB from any directory in which one is working. See the discussion of MATLABPATH in the User's Guide for further information.

15. Comparing efficiency of algorithms: flops and etime.

Two measures of the efficiency of an algorithm are the number of floating point operations (flops) performed and the elapsed time.

The MATLAB function `flops` keeps a running total of the flops performed. The command `flops(0)` (not `flops = 0`!) will reset flops to 0. Hence, entering `flops(0)` immediately before executing an algorithm and `flops` immediately after gives the flop count for the algorithm.

The MATLAB function `clock` gives the current time accurate to a hundreth of a second (see `help clock`). Given two such times `t1` and `t2`, `etime(t2,t1)` gives the elapsed time from `t1` to `t2`. One can, for example, measure the time required to solve a given linear system $Ax = b$ using Gaussian elimination as follows:

```
t = clock; x = A\b; time = etime(clock,t)
```

You may wish to compare this time—and flop count—with that for solving the system using `x = inv(A)*b;`. Try it. Version 4.0 has the more convenient `tic` and `toc`.

It should be noted that, on timesharing machines, `etime` may not be a reliable measure of the efficiency of an algorithm since the rate of execution depends on how busy the computer is at the time.

16. Output format.

While all computations in MATLAB are performed in double precision, the format of the displayed output can be controlled by the following commands.

`format short`	fixed point with 4 decimal places (the default)
`format long`	fixed point with 14 decimal places
`format short e`	scientific notation with 4 decimal places
`format long e`	scientific notation with 15 decimal places

Once invoked, the chosen format remains in effect until changed.

The command `format compact` will suppress most blank lines allowing more information to be placed on the screen or page. It is independent of the other format commands.

17. Hardcopy.

Hardcopy is most easily obtained with the **diary** command. The command

`diary` *filename*

causes what appears subsequently on the screen (except graphics) to be written to the named diskfile (if the filename is omitted it will be written to a default file named `diary`) until one gives the command `diary off`; the command `diary on` will cause writing to the file to resume, etc. When finished, you can edit the file as desired and print it out on the local system. The !-feature (see section 14) will permit you to edit and print the file without leaving MATLAB.

18. Graphics.

MATLAB can produce both planar plots and 3-D mesh surface plots. To preview some of these capabilities in version 3.5, enter the command `plotdemo`.

Planar plots. The `plot` command creates linear x-y plots; if x and y are vectors of the same length, the command `plot(x,y)` opens a graphics window and draws an x-y plot of the elements of x versus the elements of y. You can, for example, draw the graph of the sine function over the interval -4 to 4 with the following commands:

```
x = -4:.01:4; y = sin(x); plot(x,y)
```

Try it. The vector x is a partition of the domain with meshsize 0.01 while y is a vector giving the values of sine at the nodes of this partition (recall that `sin` operates entrywise).

When in the graphics screen, pressing any key will return you to the command screen while the command `shg` (show graph) will then return you to the current graphics screen. If your machine supports multiple windows with a separate graphics window, you will want to keep the graphics window exposed—but moved to the side—and the command window active.

As a second example, you can draw the graph of $y = e^{-x^2}$ over the interval -1.5 to 1.5 as follows:

```
x = -1.5:.01:1.5; y = exp(-x.^2); plot(x,y)
```

Note that one must precede ^ by a period to ensure that it operates entrywise (see section 3).

Plots of parametrically defined curves can also be made. Try, for example,

```
t=0:.001:2*pi; x=cos(3*t); y=sin(2*t); plot(x,y)
```

The command `grid` will place grid lines on the current graph.

The graphs can be given titles, axes labeled, and text placed within the graph with the following commands which take a string as an argument.

title	graph title
xlabel	x-axis label
ylabel	y-axis label
gtext	interactively-positioned text
text	position text at specified coordinates

For example, the command

```
title('Best Least Squares Fit')
```

gives a graph a title. The command `gtext('The Spot')` allows a mouse or the arrow keys to position a crosshair on the graph, at which the text will be placed when any key is pressed.

By default, the axes are auto-scaled. This can be overridden by the command `axis`. If $c = [x_{\min}, x_{\max}, y_{\min}, y_{\max}]$ is a 4-element vector, then `axis(c)` sets the axis scaling to the precribed limits. By itself, `axis` freezes the current scaling for subsequent graphs; entering `axis` again returns to auto-scaling. The command `axis('square')` ensures that the same scale is used on both axes. In version 4.0, `axis` has been significantly changed; see help axis.

Two ways to make multiple plots on a single graph are illustrated by

```
x=0:.01:2*pi;y1=sin(x);y2=sin(2*x);y3=sin(4*x);plot(x,y1,x,y2,x,y3)
```

and by forming a matrix `Y` containing the functional values as columns

```
x=0:.01:2*pi; Y=[sin(x)', sin(2*x)', sin(4*x)']; plot(x,Y)
```

Another way is with `hold`. The command `hold` freezes the current graphics screen so that subsequent plots are superimposed on it. Entering `hold` again releases the "hold." The commands `hold on` and `hold off` are also available in version 4.0.

One can override the default linetypes and pointtypes. For example,

```
x=0:.01:2*pi; y1=sin(x); y2=sin(2*x); y3=sin(4*x);
plot(x,y1,'--',x,y2,':',x,y3,'+')
```

renders a dashed line and dotted line for the first two graphs while for the third the symbol `+` is placed at each node. The line- and mark-types are

Linetypes: solid (`-`), dashed (`--`). dotted (`:`), dashdot (`-.`)
Marktypes: point (`.`), plus (`+`), star (`*`), circle (`o`), x-mark (`x`)

See `help plot` for line and mark colors.

The command `subplot` can be used to partition the screen so that up to four plots can be viewed simultaneously. See `help subplot`.

Graphics hardcopy*. A hardcopy of the graphics screen can be most easily obtained with the MATLAB command `print`. It will send a high-resolution copy of the current graphics screen to the printer, placing the graph on the top half of the page.

In version 4.0 the `meta` and `gpp` commands described below have been absorbed into the `print` command. See `help print`.

Producing unified hard copy of several plots requires more effort. The Matlab command `meta` *filename* stores the current graphics screen in a file named *filename.met* (a "metafile") in the current directory. Subsequent `meta` (no filename) commands append a new current graphics screen to the previously named metafile. This metafile—which may now contain several plots—may be processed later with the graphics post-processor (GPP) program to produce high-resolution hardcopy, two plots per page.

---

* The features described in this subsection are not available with the Student Edition of Matlab. Graphics hardcopy can be obtained there only with a screen dump: Shift-PrtScr.

The program GPP (graphics post-processor) is a *system* command, not a MATLAB command. However, in practice it is usually involked from within MATLAB using the "!" feature (see section 14). It acts on a device-independent metafile to produce an output file appropriate for many different hardcopy devices.

The selection of the specific hardcopy device is made with the option key "`/d`". For example, the system commands

```
gpp filename /dps
gpp filename /djet
```

will produce files `filename.ps` and `filename.jet` suitable for printing on, respectively, PostScript and HP LaserJet printers. They can be printed using the usual printing command for the local system—for example, `lpr filename.ps` on a Unix system. Entering the system command `gpp` with no arguments gives a list of all supported hardcopy devices. On a PC, most of this can be automated by appropriately editing the file `gpp.bat` distributed with MATLAB.

3-D mesh plots. Three dimensional mesh surface plots are drawn with the function `mesh`. The command `mesh(z)` creates a three-dimensional perspective plot of the elements of the matrix z. The mesh surface is defined by the z-coordinates of points above a rectangular grid in the x-y plane. Try `mesh(eye(10))`.

To draw the graph of a function $z = f(x, y)$ over a rectangle, one first defines vectors xx and yy which give partitions of the sides of the rectangle. With the function `meshdom` (mesh domain; called `meshgrid` in version 4.0) one then creates a matrix x, each row of which equals xx and whose column length is the length of yy, and similarly a matrix y, each column of which equals yy, as follows:

```
[x,y] = meshdom(xx,yy);
```

One then computes a matrix z, obtained by evaluating f entrywise over the matrices x and y, to which `mesh` can be applied.

You can, for example, draw the graph of $z = e^{-x^2-y^2}$ over the square $[-2, 2] \times [-2, 2]$ as follows (try it):

```
xx = -2:.1:2;
yy = xx;
[x,y] = meshdom(xx,yy);
z = exp(-x.^2 - y.^2);
mesh(z)
```

One could, of course, replace the first three lines of the preceding with

```
[x,y] = meshdom(-2:.1:2, -2:.1:2);
```

You are referred to the User's Guide for further details regarding `mesh`.

In version 4.0, the 3-D graphics capabilities of MATLAB have been considerably expanded. Consult the on-line help for `plot3`, `mesh`, and `surf`.

19. Reference.

There are many MATLAB features which cannot be included in these introductory notes. Listed below are some of the MATLAB functions and operators available, grouped by subject area[1]. Use the on-line help facility or consult the User's Guide for more detailed information on the functions.

There are many functions beyond these. There exist, in particular, several "toolboxes" of functions for specific areas; included among such are signal processing, control systems, robust-control, system identification, optimization, splines, chemometrics, μ-analysis and synthesis, state-space identification, and neural networks[2]. These can be explored via the command `help`.

GENERAL	
help	help facility
demo	run demonstrations
who	list variables in memory
what	list M-files on disk
size	row and column dimensions
length	vector length
clear	clear workspace
computer	type of computer
^C	local abort
exit	exit MATLAB
quit	same as exit

MATRIX OPERATORS		ARRAY OPERATORS	
+	addition	+	addition
−	subtraction	−	subtraction
*	multiplication	.*	multiplication
/	right division	./	right division
\	left division	.\	left division
^	power	.^	power
'	conjugate transpose	.'	transpose

RELATIONAL AND LOGICAL OPERATORS			
<	less than	&	and
<=	less than or equal	|	or
>	greater than	~	not
>=	greater than or equal		
==	equal		
~=	not equal		

[1] Source: MATLAB User's Guide, version 3.5

[2] The toolboxes, which are optional, may not be installed on your system.

Special Characters	
=	assignment statement
[	used to form vectors and matrices
]	see [
(	arithmetic expression precedence
)	see (
.	decimal point
...	continue statement to next line
,	separate subscripts and function arguments
;	end rows, suppress printing
%	comments
:	subscripting, vector generation
!	execute operating system command

Special Values	
ans	answer when expression not assigned
eps	floating point precision
pi	π
i, j	$\sqrt{-1}$
inf	∞
NaN	Not-a-Number
clock	wall clock
date	date
flops	floating point operation count
nargin	number of function input arguments
nargout	number of function output arguments

Disk Files	
chdir	change current directory
delete	delete file
diary	diary of the session
dir	directory of files on disk
load	load variables from file
save	save variables to file
type	list function or file
what	show M-files on disk
fprintf	write to a file
pack	compact memory via **save**

Special Matrices	
compan	companion
diag	diagonal
eye	identity
gallery	esoteric
hadamard	Hadamard
hankel	Hankel
hilb	Hilbert
invhilb	inverse Hilbert
linspace	linearly spaced vectors
logspace	logarithmically spaced vectors
magic	magic square
meshdom	domain for mesh points
ones	constant
pascal	Pascal
rand	random elements
toeplitz	Toeplitz
vander	Vandermonde
zeros	zero

Matrix Manipulation	
rot90	rotation
fliplr	flip matrix left-to-right
flipud	flip matrix up-to-down
diag	diagonal matrices
tril	lower triangular part
triu	upper triangular part
reshape	reshape
.'	transpose
:	convert matrix to single column; `A(:)`

Relational and Logical Functions	
any	logical conditions
all	logical conditions
find	find array indices of logical values
isnan	detect NaNs
finite	detect infinities
isempty	detect empty matrices
isstr	detect string variables
strcmp	compare string variables

Control Flow	
if	conditionally execute statements
elseif	used with **if**
else	used with **if**
end	terminate **if, for, while**
for	repeat statements a number of times
while	do while
break	break out of **for** and **while** loops
return	return from functions
pause	pause until key pressed

Programming and M-Files	
input	get numbers from keyboard
keyboard	call keyboard as M-file
error	display error message
function	define function
eval	interpret text in variables
feval	evaluate function given by string
echo	enable command echoing
exist	check if variables exist
casesen	set case sensitivity
global	define global variables
startup	startup M-file
getenv	get environment string
menu	select item from menu
etime	elapsed time

Text and Strings	
abs	convert string to ASCII values
eval	evaluate text macro
num2str	convert number to string
int2str	convert integer to string
setstr	set flag indicating matrix is a string
sprintf	convert number to string
isstr	detect string variables
strcomp	compare string variables
hex2num	convert hex string to number

Command Window	
clc	clear command screen
home	move cursor home
format	set output display format
disp	display matrix or text
fprintf	print formatted number
echo	enable command echoing

Graph Paper	
plot	linear X-Y plot
loglog	loglog X-Y plot
semilogx	semi-log X-Y plot
semilogy	semi-log X-Y plot
polar	polar plot
mesh	3-dimensional mesh surface
contour	contour plot
meshdom	domain for mesh plots
bar	bar charts
stairs	stairstep graph
errorbar	add error bars

Graph Annotation	
title	plot title
xlabel	x-axis label
ylabel	y-axis label
grid	draw grid lines
text	arbitrarily position text
gtext	mouse-positioned text
ginput	graphics input

Graph Window Control	
axis	manual axis scaling
hold	hold plot on screen
shg	show graph window
clg	clear graph window
subplot	split graph window

Graph Window Hardcopy	
print	send graph to printer
prtsc	screen dump
meta	graphics metafile

Elementary Math Functions	
abs	absolute value or complex magnitude
angle	phase angle
sqrt	square root
real	real part
imag	imaginary part
conj	complex conjugate
round	round to nearest integer
fix	round toward zero
floor	round toward $-\infty$
ceil	round toward ∞
sign	signum function
rem	remainder
exp	exponential base e
log	natural logarithm
log10	log base 10

Trigonometric Functions	
sin	sine
cos	cosine
tan	tangent
asin	arcsine
acos	arccosine
atan	arctangent
atan2	four quadrant arctangent
sinh	hyperbolic sine
cosh	hyperbolic cosine
tanh	hyperbolic tangent
asinh	hyperbolic arcsine
acosh	hyperbolic arccosine
atanh	hyperbolic arctangent

Special Functions	
bessel	bessel function
gamma	gamma function
rat	rational approximation
erf	error function
inverf	inverse error function
ellipk	complete elliptic integral of first kind
ellipj	Jacobian elliptic integral

Decompositions and Factorizations	
balance	balanced form
backsub	backsubstitution
cdf2rdf	convert complex-diagonal to real-diagonal
chol	Cholesky factorization
eig	eigenvalues and eigenvectors
hess	Hessenberg form
inv	inverse
lu	factors from Gaussian elimination
nnls	nonnegative least squares
null	null space
orth	orthogonalization
pinv	pseudoinverse
qr	orthogonal-triangular decomposition
qz	QZ algorithm
rref	reduced row echelon form
schur	Schur decomposition
svd	singular value decomposition

Matrix Conditioning	
cond	condition number in 2-norm
norm	1-norm,2-norm,F-norm,∞-norm
rank	rank
rcond	condition estimate (reciprocal)

Elementary Matrix Functions	
expm	matrix exponential
logm	matrix logarithm
sqrtm	matrix square root
funm	arbitrary matrix function
poly	characteristic polynomial
det	determinant
trace	trace
kron	Kronecker tensor product

Polynomials	
poly	characteristic polynomial
roots	polynomial roots—companion matrix method
roots1	polynomial roots—Laguerre's method
polyval	polynomial evaluation
polyvalm	matrix polynomial evaluation
conv	multiplication
deconv	division
residue	partial-fraction expansion
polyfit	polynomial curve fitting

Column-wise Data Analysis	
max	maximum value
min	minimum value
mean	mean value
median	median value
std	standard deviation
sort	sorting
sum	sum of elements
prod	product of elements
cumsum	cumulative sum of elements
cumprod	cumulative product of elements
diff	approximate derivatives
hist	histograms
corrcoef	correlation coefficients
cov	covariance matrix
cplxpair	reorder into complex pairs

Signal Processing	
abs	complex magnitude
angle	phase angle
conv	convolution
corrcoef	correlation coefficients
cov	covariance
deconv	deconvolution
fft	radix-2 fast Fourier transform
fft2	two-dimensional FFT
ifft	inverse fast Fourier transform
ifft2	inverse 2-D FFT
fftshift	FFT rearrangement

Numerical Integration	
quad	numerical function integration
quad8	numerical function integration

Differential Equation Solution	
ode23	2nd/3rd order Runge-Kutta method
ode45	4th/5th order Runge-Kutta-Fehlberg method

Nonlinear Equations and Optimization	
fmin	minimum of a function of one variable
fmins	minimum of a multivariable function
fsolve	solution of a system of nonlinear equations (zeros of a multivariable function)
fzero	zero of a function of one variable

Interpolation	
spline	cubic spline
table1	1-D table look-up
table2	2-D table look-up