1. Accessing MATLAB.

2. Entering matrices.

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.

For example, either of the statements

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

and

```
A=[
123
456
789]
```

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.

MATLAB allows complex numbers in all its operations and functions. Two convenient ways to enter complex matrices are:

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

you may generate a new imaginary unit with, say, ii = sqrt(-1).

The command **rand(n)** will create an n × n matrix with randomly generated entries distributed uniformly between 0 and 1, while **rand(m,n)** will create an m × n one.

magic(n) will create an integral n × n matrix which is a magic square (rows, columns, and diagonals have common sum

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.

3. Matrix operations, array operations.

The following matrix operations are available in MATLAB:

+	addition
-	subtraction
*	multiplication
^	power
'	conjugate transpose
\	left division
/	right division

These matrix operations apply, of course, to scalars (1-by-1 matrices) as well.

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*.

Array operations.

*, ^, \, 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

If the variable name and = sign are omitted, a variable **ans** (for answer) is automatically created to which the result is assigned.

MATLAB is **case-sensitive** in the names of commands, functions, and variables.

The command **who** (or whos) 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 unit roundoff—about 10^{-16} on most machines. It is useful in specifying 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	create or extract diagonals
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(size(A)) produces a matrix of zeros having 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 =  l:n,   x=[x,i^2],   end
```

or

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

will produce a certain n-vector and the statement

```
x =   [];   for  i = n:-l:l,   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 for statement permits *any* matrix to be used instead of l:n. The variable just consecutively assumes the value of each column of the matrix. For example,

```
s = 0; for c = A
      s =  s + sum(c); end
```

computes the sum of all entries of the matrix *A* by adding its column sums, => **sum(sum(A))**

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 $2^n > a$:

```
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
```

if n < 0

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

Relations.

<	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.

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, oo-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. UP/DOWN arrows to switch between previous commands.

11. Submatrices and colon notation.

"Colon notation" (which is used both to generate vectors and reference submatrices) minimize the use of loops (which slows MATLAB

The expression 1:5 (met earlier in for statements) is actually the row vector [12 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 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(:,3) is the third column of A, and A(1:4,:) is the first four rows.

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

A(:,[2 4 5]) = B(:,1:3) replaces columns 2,4,5 of A with the first three columns of B.

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]
```

If x is an n-vector, what is the effect of x = x(n:-1:1)? Also try y = fliplr(x) and y = flipud(x').

12. M-files.

Function files.

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**

Then a MATLAB statement z = randint(4,5)

Note that use of nargin ("number of input arguments")

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 of  x;
%       [mean,   stdev]   = stat(x)   both the mean and standard deviation.
%       For a matrix x,   stat(x)   acts columnwise.
 [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);
```

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 aborts execution of 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:    ')
```

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
format	rat	approximation by ratio of small integers
format	hex	hexadecimal format
format	bank	fixed dollars and cents
format	+	+,-,blank

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

18. Graphics.

MATLAB can produce planar plots of curves, 3-D plots of curves, 3-D mesh surface plots, and 3-D faceted surface plots. The primary commands for these facilities are **plot, plot3, mesh,** and **surf**, respectively. An introduction to each of these is given below.

To preview some of these capabilities, enter the command demo and select some of the graphics options.

Planar plots.

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

=> figure, plot – figure(1), plot – figure(2), plot

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

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

MATLAB supplies a function **fplot** to easily and efficiently plot the graph of a function.

```
function y = expnormal(x);
y =  exp(-x.^2);
```

Then the command

fplot('expnormal',   [-1.5,1.5]) will produce the graph. Try it.

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)
```

LOOK HELP!!!

title	**graph title**
xlabel	**x-axis label**
ylabel	**y-axis label**
gtext	**place text on the graph using the mouse**
text	**position text at specified coordinates**

title('Best Least Squares Fit')

gives a graph a title. The command gtext('The Spot ') …

By default, the axes are auto-scaled. This can be overridden by the command axis.

axis([x_{min},x_{max},y_{min},y_{max}])	set axis scaling to prescribed limits
axis(axis)	freezes scaling for subsequent graphs
axis auto	returns to auto-scaling
v = axis	returns vector *v* showing current scaling
axis square	same scale on both axes
axis equal	same scale and tic marks on both axes
axis off	turns off axis scaling and tic marks
axis on	turns on axis scaling and tic marks

The axis command should be given *after* the plot command.

Two ways to make **<u>multiple plots</u>** on a single graph are illustrated by

x=0:.01:2*pi;yl=sin(x);y2=sin(2*x);y3=sin(4*x);plot(x,yl,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 on freezes the current graphics screen so that subsequent plots are superimposed on it. The axes may, however, become rescaled. Entering **hold off** releases the "hold."

<u>Linetypes:</u> solid (-), dashed (--). dotted (:), dashdot (-.)

Marktypes: point (.), plus (+), star (*), circle (o), x-mark (x)

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

plot(x,y1,'--',x,y2,':',x,y3,'+')

<u>Colors:</u> yellow (y), magenta (m), cyan (c), red (r),

green (g), blue (b), white (w), black (k)

plot(x,y,'r--') plots a red dashed line.

subplot partitions the screen so that several small plots can be placed in one figure. See help!