1. For statement

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

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

Will produce a certain *n*-vector and the statement. This will produced the same vector in reverse order

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

The statements will produced and print to the screen the $m \times n$ hilbert matrix. The semicolon on the inner statement suppresses printing of unwanted intermediate results while the last *H* displays the final result.

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

2. While statement

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 to the power of *n* is > *a*:

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

3. If statement

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 illustrated by

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

4. 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 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, for example.

```
a = rand(5);
b = triu(a);
 a == b;
```

Where TRIU is extract upper triangular part.

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
```

5. 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, you can declare a variable to be global if you wish. We first illustrate with a simple example of a function file.

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

A more general version of this function is given as follow:

```
function y = 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.
%   randint(m,n,a,b) returns entries between integers a and b.
if nargin < 3, a=0; b=9; end
y = floor((b-a+1)*rand(m,n))+a;
```

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 given above.

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

```
function y = 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
```

6. Text strings, error messages, input

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

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

Assign 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') or
fprintf('this message is hereby displayed\n')
```

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.

7. 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.

8. Graphics

MATLAB can produce planar plots, images, and 3-D mesh surface plots.

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)
```

As a second example, we will draw the graph of a exponential function:

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

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

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

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 specific coordinates

For example, the command `title('Best Least Squares Fit')` gives the 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* = [*xmin,xmax,ymin,ymax*] is a 4-element vector, then **axis(*c*)** sets the axis scaling to the prescribed 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. 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,y2,y3)
```

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 the **hold** command. 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.

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

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-type are:

- **Line types**: dashed(--), dotted(:), dash dot(-.), and the default solid(-)
- **Mark types**: point(.), plus(+), star(*), circle(o), x-mark(x), square(s), diamond(d), up-triangle(v), down-triangle(^), left-triangle(<), right-triangle(>), pentagram(p), hexagram(h)
- **Colors**: yellow(y), magenta(m), cyan(c), red(r), green(g), white(w), black(k), and the default blue(b)

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

Images

Matrices could be viewed as an image, since any digital image is like a matrix in that it has rows and columns of values. The command **image (*A*)** will "image" the matrix A. The matrix will be displayed differently depending on the colormap. The colormap can consist of 16 colors or 256 colors; changing how many colors are used will alter the way the matrix is displayed. In order to avoid this alteration due to differences in the number of colors you can use the command **imagesc(*A*)** which will automatically scale your matrix to span the entire spectrum of colors possible.

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 the 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 gives partitions of the sides of the rectangle. With the function **meshgrid** (mesh grid) on 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] = meshgrid(xx,yy);
```

The following example will draws the graph of *z = exp(-sqr(x)-sqr(y))* over the square [-2,2] \times [-2,2].

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

Alternatively, you could replace the first three lines of the preceding example with:

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

For more details regarding the command **mesh**, you can refer to online help or the User's Guide.
Others command related to 3-D graphics are **plot3** and **surf**; where **plot3** is used to draw a line in the 3-D space while **surf** is used to draw a surface in the 3-D space.

9. Plotting with Matlab

All Matlab variables must have numerical values:

```
x= -10:.1:10;
plot(sin(x));
plot(x, sin(x));
```

We can plot the "inverse relationship" (for example, the squaring function and +/- square root) easily:

```
x= -10:.1:10;
plot(x, x.^2)
plot(x.^2, x)
```

Or a spiral in two or in three dimensions:

```
t = 0:.1:10;
plot( t .* cos(t), t .* sin(t) )
plot3( t .* cos(t), t .* sin(t), t )
```

Plot several curves simultaneously with **plot(x1, y1, x2, y2, ...)**:

```
x= -10:.1:10;
 plot( x, cos(x), x, 1 - x.^2./2, x, 1 - x.^2./2 + x.^4./24 )
```

Let's add some options [first plot, then options] -- the order of the options can affect the result! We will *first* make the scales [the "ratio"] equal, then set the plotting window to be -6 < x < 6, -2 < y < 2.

```
axis equal, axis([-6 6 -2 2] )
```

Here's another example, a semilog plot. In the graphic we also have introduced another variation, the "plot matrix" -- to learn more, enter the command help subplot.

```
plot( x, exp(x) ), set(gca,'yscale','log')
```

Here are other ways to graph multiple curves, using matrices (plotted by columns) and using "hold."

```
m = [1 .5 .3; .5 .3 .25; .3 1 .2];
plot(m)
```

```
x= [0:.1:2.5];
polar(x, cos(x))
hold
polar(x, sin(x))
```

Functions of two variables may be plotted, as well, but some "setup" is required!

```
[x y] = meshgrid(-3:.1:3, -3:.1:3);
 z = x.^2 - y.^2;
```

Here are two options for plotting the surface.

```
[x y] = meshgrid(-3:.1:3, -3:.1:3);
z = x.^2 - y.^2;
subplot(1,2,1);
mesh(x,y,z)
subplot(1,2,2);
surf(x,y,z)
```