***The objective of this Lab is to introduce some of the fundamentals of MATLAB programming***

MATLAB is a powerful computing system for handling the calculations involved in scientific and engineering problems.

* The name MATLAB stands for MATrixLABoratory, because the system was designed to make matrix computationsparticularly easy.
* There are two essential requirements for successful MATLAB programming:

➤You need to learn the *exact* rules for writing MATLAB statements.

➤You need to develop a logical plan of attack for solving particular problems.

* When MATLAB starts, the MATLAB desktop opens as shown inFigure 1.1. The windowin the desktop that concerns us for this chapter is theCommand Window, where the special \_ prompt appears. This prompt meansthat MATLAB is waiting for a command



1-Since we have experience doing arithmetic, we want to examine if MATLAB does this correctly:

>> 2+3

ans =

5

>> 3-2

ans =

1

>> 2\*3

ans =

6

>> 1/2

ans =

0.5000

>> 2^3

ans =

8

>> 2/1

ans =

2

* The line with the >> prompt is called the *command line*.
* You can edit a MATLAB command before pressing Enter by using various combinations of the Backspace, Left-arrow, Right-arrow and Del keys. This helpful feature is called *command line editing*.
* You can select (and edit) previous commands you have entered using Up-arrow and Down-arrow. But remember to press Enterto get the command carried out (i.e. to *run* or to *execute* thecommand).
* MATLAB has a useful editing feature called *smart recall*. Just typethe first few characters of the command you want to recall, e.g.type the characters 2\* and press the Up-arrow key—this recallsthe most recent command starting with 2\*.

2- Now let us assign values to variables to do arithmetical operations withthe variables.

>> a=2

a =

2

>> a=a+7

a =

9

>> a=a\*10

a =

90

Now enter the statement

>> b=3;

The semicolon (;) prevents the value of b from being displayed. However, b still has the value 3 as you can see by entering its name without a semicolon, i.e. by executing the following command:

>>b

b =

3

>>pi

ans =

3.1416

>>sqrt(pi)

ans =

1.7725

calculate sin (90O).

Trigonometric functions like sin(x) expect the argument x to be in *radians*. Multiply degrees by *π/*180 to get radians.

>>sin(90\*pi/180)

ans =

1

The exponential function *ex*is computed in MATLAB as exp(x).find e­1 and e2

>>exp(1)

ans =

2.7183

>>exp(2)

ans =

7.3891

>>pi=4

pi =

4

>>sqrt(pi)

ans =

2

>>whos

Name Size Bytes Class Attributes

ans 1x1 8 double

pi 1x1 8 double

>>clear pi

>>whos

Name Size Bytes Class Attributes

ans 1x1 8 double

>>sqrt(pi)

ans =

1.7725

>>clear

>>whos

Go to desktop and desktop layout and then select default, by doing so all the windows will come at once…..

MATLAB also has numerous *commands*, such as clc (for *clear commandwindow*).

Variables such as a and b above are called *scalars*; they are single-valued.MATLAB also handles *vectors* (generally referred to in MATLAB as *arrays*),which are the key to many powerful features of the language. The easiestway of defining a vector where the elements (components) increase by thesame amount is with a statement like

>> x=0:10;

That is a *colon* (:) between the 0 and the 10. There’s no need to leave aspace on either side of it, but it makes it more readable. Enter x to checkthat x is a vector; it is a *row vector*, i.e. it is a single row by 11 columnsarray. Type the following command to verify that this is the case:

>>size(x)

ans =

1 11

Now enter following command

>> x=0:10;

>> y=2.\*x

y =

0 2 4 6 8 10 12 14 16 18 20

>> w=y./x

w =

NaN 2 2 2 2 2 2 2 2 2 2

Another special value that you may meet is NaN, which stands for *Not-a-Number*. It is the answer to calculations like 0/0.

>>z=sin(x)

z =

Columns 1 through 9

0 0.8415 0.9093 0.1411 -0.7568 -0.9589 -0.2794 0.6570 0.9894

Columns 10 through 11

0.4121 -0.5440

All you have to do to draw a reasonably nice graph of sin (*x*) is to enterthe following commands:

>> x = 0 : 0.1 : 10;

>> z=sin(x);

>>plot(x,z), grid



Note that the first command line above has three numbers after theequal sign. When there are three numbers separated by two colonsin this way, the middle number is the *increment*. The increment of0.1 was selected to give a reasonably smooth graph. The commandgrid following the comma in the last command line adds a grid tothe graph.

>> z=sin(2\*x);

>>plot(x,z), grid



Systems of linear equations are very important in engineering and scientificanalysis. A simple example is finding the solution of two simultaneousequations, e.g.

*x1*+ 2x2= 4,

2*x1*–*x2*= 3*.*

***matrix*method approach**

>> a = [1 2; 2 -1];

>> b = [4; 3];

>> x = a\b AX=B

X=A-1 B

x =

2

1

>>x(1)+2\*x(2)% should give ans = 4

ans =

4

>> 2\*x(1)-x(2)% should give ans = 3

ans =

3

The % symbol is a *flag* that indicates all information to the right is *not* part of the command. It is a comment.

See also

>> help elfun

Elementary math functions.

>>lookfor eigenvalue

**solve function approach**

>> syms x y

>> [x,y]=solve(x+2\*y==4,2\*x-y==3)

x =

2

y =

1Suppose you want to draw the graph of *e*−0*.*2*x*sin (*x*) over the domain 0 to 6*π*,From the MATLAB desktop select **File -> New -> M-file**, Now type the following two lines in the Editor, exactlyas they appear here:

x=0:pi/20:6\*pi;

plot(x,exp(-0.2\*x).\*sin(x),'r'),grid



*The objective of Lab#2 is to introduce some of the fundamentals of MATLAB programming, including:*

➤variables, operators and expressions;

➤arrays (including vectors and matrices);

➤basic input and output;

➤repetition (**for**);

➤decisions (**if**).

**Variables**

Variables are fundamental to programming. In a sense, the art of programming is *getting the right values in the right variables at the right time.*

A variable name must comply with the following two rules:

1. It may consist only of the letters a–z, the digits 0–9 and the underscore ( \_ ).

2. It must start with a letter.

Examples of valid variable names: r2d2 pay\_day

Examples of invalid names: pay-day 2a

A variable is created simply by assigning a value to it at the command line or

in a program, e.g.

a = 98

If you attempt to refer to a non-existent variable you will get the error message

??? Undefined function or variable ...

The official MATLAB documentation refers to all variables as *arrays*, whetherthey are single-valued (scalars) or multi-valued (vectors or matrices). In otherwords, a scalar is a 1x1 array, i.e. an array with a single row and a singlecolumn which, of course, is an array of one item.

MATLAB is *case sensitive*, which means it distinguishes between upper- and lowercase letters. So balance, BALANCE and BaLance are three different variables.

Now enter the command who. You should see x variable as follows:

whos

Name Size Bytes Class Attributes

x 1x121 968 double

All the variables you create during a session remain in the workspace untilyou clear them. You can use or change their values at any stage during thesession.

The command who lists the names of all the variables in your workspace.

The function ans returns the value of the last expression evaluated but not

assigned to a variable.The command whos lists the size of each variable as well.

Each variable here occupies eight *bytes* of storage. A byte is the amount ofcomputer memory required for one character (if you are interested, one byte isthe same as eight *bits*).

These variables each have a *size* of ‘1-by-1’, because they are *scalars*, asopposed to vectors or matrices (although as mentioned above MATLAB regardsthem all as 1-by-1 arrays).

double means that the variable holds numeric values as double-precisionfloating point.

A particular variable can be removed from the workspace, e.g. clear x

More than one variable can also be cleared, e.g. clear x y

When you run a program, any variables created by it remain in the workspaceafter it has run. This means that existing variables with the same names getoverwritten.

**Arrays: vectors and matrices**

MATLAB stands for MATrixLABoratory because MATLAB has been designed to work with *matrices*.

A *vector* is a special type of matrix, having only one row, or one column. Vectors are also called *lists* or *arrays* in other programming languages.

>> x = [1 3 0 -1 5]

x =

1 3 0 -1 5

>>disp(x)

1 3 0 -1 5

>> a = [5,6,7]

a =

5 6 7

>> x = [130-15] % don’t forget space or commas.

x =

115

>> a = [1 2 3];

b = [4 5];

c = [a -b];

>>disp(c)

1 2 3 -4 -5

>> a = [1 3 7];

a = [a 0 -1];

>>disp(a)

1 3 7 0 -1

**Initializing vectors: the colon operator**

>> x = 1:10

x =

1 2 3 4 5 6 7 8 9 10

x = 1:0.5:4

x =

1.0000 1.5000 2.0000 2.5000 3.0000 3.5000 4.0000

>> x = 10:-1:1

x =

10 9 8 7 6 5 4 3 2 1

>> x = 1:2:6

x =

1 3 5

>> x = 0:-2:-5

x =

0 -2 -4

>> x = 1:0

x =

Empty matrix: 1-by-0

>>linspace(0, pi/2, 10)

ans =

Columns 1 through 9

0 0.1745 0.3491 0.5236 0.6981 0.8727 1.0472 1.2217 1.3963

Column 10

1.5708

Linespace creates a vector of 10 equally spaced points from 0 to *π/*2 (inclusive).

>> x = 1:5

x =

1 2 3 4 5

>>x'

ans =

1

2

3

4

5

>> y = [1 4 8 0 -1]'

y =

1

4

8

0

-1

>> r=rand(1,7)

r =

0.8407 0.2543 0.8143 0.2435 0.9293 0.3500 0.1966

>>r(3)

ans =

0.8143

>>r(2:4)

ans =

0.2543 0.8143 0.2435

<<r(1:2:7)

ans =

0.8407 0.8143 0.9293 0.1966

**Matrices**

>> a = [1 2 3; 4 5 6]

a =

1 2 3

4 5 6

>> x = 0:30:180;

table = [x' sin(x\*pi/180)']

table =

0 0

30.0000 0.5000

60.0000 0.8660

90.0000 1.0000

120.0000 0.8660

150.0000 0.5000

180.0000 0.0000

>> a = [2 4 8];

>> b = [3 2 2];

>>a .\* b

ans =

6 8 16

>>a ./ b

ans =

0.6667 2.0000 4.0000

The operations are sometimes called *array operations*, or *element-by-element*operations because the operations are performed element by element. Forexample, a .\* b results in the following vector (sometimes called the *arrayproduct*):

[a(1)\*b(1) a(2)\*b(2) a(3)\*b(3)]

i.e. [6 8 10].

Array operations also apply to operations between a scalar and a non-scalar.

<<3 .\* a

ans =

6 12 24

>> a = [2 4 8];

>> a.^2

ans =

4 16 64

>> a = [2 4 8];

>> b = [3 2 2];

>> a.\*b

ans =

6 8 16

**Output**

There are two straightforward ways of getting output from MATLAB:

1. By entering a variable name, assignment or expression on the commandline, without a semicolon;

2. With the disp statement, e.g. disp( x ).

The general form of disp for a numeric variable is**disp**(***variable***)

To display a message and a numeric value on the same line use the followingtrick:

>> x = 2;

>>disp( ['IDREES…The answer is ', num2str(x)] );

The answer is 2

>> x = 2;

>>disp( ['Welcome---------- ', num2str(x)] );

We convert the number x to its *string representation* with the function num2str; read this as ‘number to string’.

>>a

a =

2 4 8

>>b

b =

3 2 2

>>x

x =

2

>>disp( [a b x] )

2 4 8 3 2 2 2

The square brackets create a vector with three elements, which are all displayed.

**Repeating with for**

>>fori = 1:5, disp(i), end

1

2

3

4

5

**Square roots with Newton’s method**

The square root *x* of any positive number *a* may be found using only the arithmeticoperations of addition, subtraction and division with *Newton’s method*.This is an iterative (repetitive) procedure that refines an initial guess.

The structure plan of the algorithm to find the square root and a program withsample output for *a*=2 are as follows.

The structure plan:

1. Initialize *a*

2. Initialize *x* to *a/*2

3. Repeat 6 times (say)

Replace *x* by (*x* + *a/x*)*/*2

Display *x*

4. Stop.

**Matlab Program**

>> a = 2;

x = a/2;

disp(['The approach to sqrt(a) for a = ', num2str(a)])

for i = 1:6

x = (x + a / x) / 2;

disp( x )

end

disp( 'Matlab''s value: ')

disp(sqrt(2) )

The approach to sqrt(a) for a = 2

1.5000

1.4167

1.4142

1.4142

1.4142

1.4142

Matlab's value:

1.4142

>>sqrt(2)

ans =

1.4142

*n*! = 1 × 2 × 3 ×*. . .* × (*n* − 1) ×*n.*

>> n = 5;

fact = 1;

for k = 1:n

fact = k \* fact;

disp( [k fact] )

end

1 1

2 2

3 6

4 24

5 120

In general the most common form of the for loop (for use in a program, not

on the command line) is

for*index* = *j*:*k*

*statements*

end

or

for *index* = *j*:*m*:*k*

*statements*

end

**for in a single line**

If you insist on using for in a single line, here is the general form:

for *index* = *j*:*k*, *statements*, end

or

for *index* = *j*:*m*:*k*, *statements*, end

Note:

1. Don’t forget the commas (semicolons will also do if appropriate). If you leave them out you will get an error message.

2. Again, statements can be one or more statements, separated by commas or semicolons.

3. If you leave out end, MATLAB will wait for you to enter it. Nothing will happenuntil you do so.

**Decisions**

The MATLAB function rand generates a random number in the range 0–1. Enter

the following two statements at the command line:

>> r = rand

if r > 0.5 disp( 'greater indeed' ), end

r =

0.9058

greater indeed

>> 2>0

ans =

1

>> 2<0

ans =

0

>> x = 2;

if x < 0 disp( 'neg' ), else disp( 'non-neg' ), end

>> x = 2;

if x < 0 disp('Ibrahim' ), else disp( 'Idrees' ), end

non-neg

>> x =- 2;

if x < 0 disp( 'neg' ), else disp( 'non-neg' ), end

non-neg

**Review:**

For Loop:

The *for* loop allows us to repeat certain commands. If you want to repeat some action in a predetermined way, you can use the *for* loop. All of the loop structures in matlab are started with a keyword such as *for*, or *while* and they all end with the word *end*.

for j =1:4,

disp( j)

end

j =

1

j =

2

j =

3

j =

4

>>for j=1:4,

v(j) = j+10;

end

>>v

v =

11 12 13 14

>> A = [ [1 2 3]' [3 2 1]' [2 1 3]']

A =

1 3 2

2 2 1

3 1 3

>>for j=2:3,

A(j,:) = A(j,:) - A(j-1,:)

end

A =

1 3 2

1 -1 -1 % A(2)-A(1)

3 1 3

A =

1 3 2

1 -1 -1

2 2 4 % A(3)-A(2)

>> A = [ [1 2 3]' [3 2 1]' [2 1 3]']

A =

1 3 2

2 2 1

3 1 3

>> B=A

B =

1 3 2

2 2 1

3 1 3

>>B(2,:)

ans =

2 2 1

>>B(:,2)

ans =

3

2

1

**Gauss Elimination Method:(HOME TASK)**

>>B = [ [1 2 3]' [3 2 1]' [2 1 3]']

B =

1 3 2

2 2 1

3 1 3

>>for j=2:3,

fori=j:3,

B(i,:) = B(i,:) - B(j-1,:)\*B(i,j-1)/B(j-1,j-1)

end

end

B =

1 3 2

0 -4 -3

3 1 3

B =

1 3 2

0 -4 -3

0 -8 -3

B =

1 3 2

0 -4 -3

0 0 3

## While Loop

## If you don’t like the for loop , you can also use a while loop . The while loop repeats a sequence of commands as long as some condition is met.

## Overview of While versus for loop

For loop has fixed number of iteration and while loop iterate until condition is met.

While(condition)

Do -------;

end

>> N=1;

while log(N)<=10;

N=N+1;

end

disp(N);

22027

>>log(22027)

ans =

10.0000

>>log(22028)

ans =

10.0001

>>sum=0;

>>fori=1:1:5

sum=sum+i;

end

>>disp(sum)

15

Let turn this for loop into while loop

>>sum=0;

i=1;

whilei<6

sum=sum+i;

i=i+1;

end

>>disp(sum)

15

# If Statement

>> a = 2;

b = 3;

if (a<b)

j = -1

end

>>j

j =

-1

>> a = 3;

b = 2;

if (a<b)

j = -1;

end