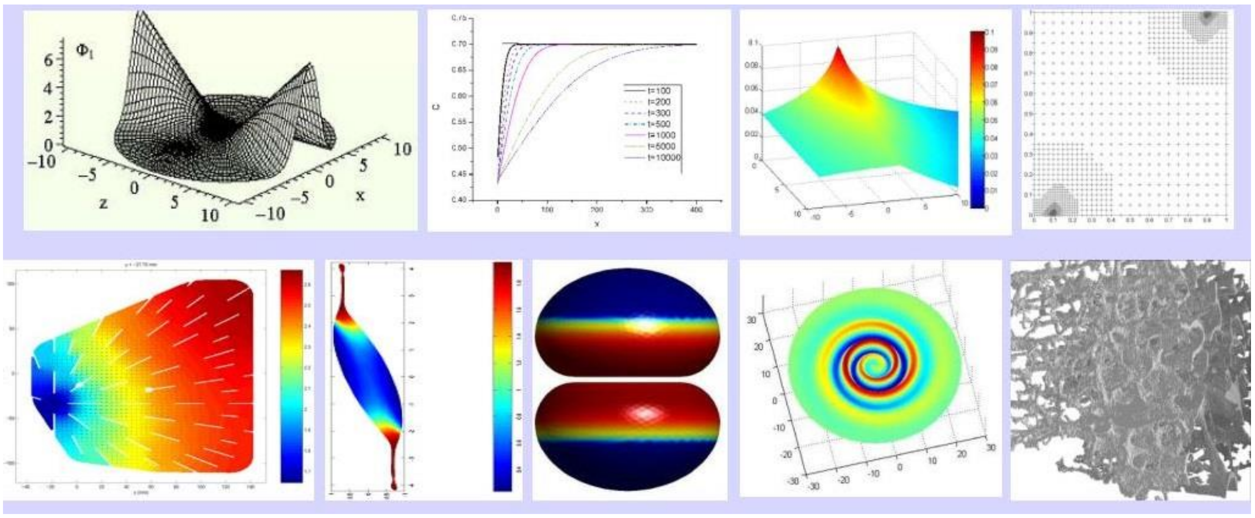
Lecture Notes in Computational Mathematics I (MATLAB) for Second Class of Mathematics



Imad A. Aziz

Erbil

2022-2023

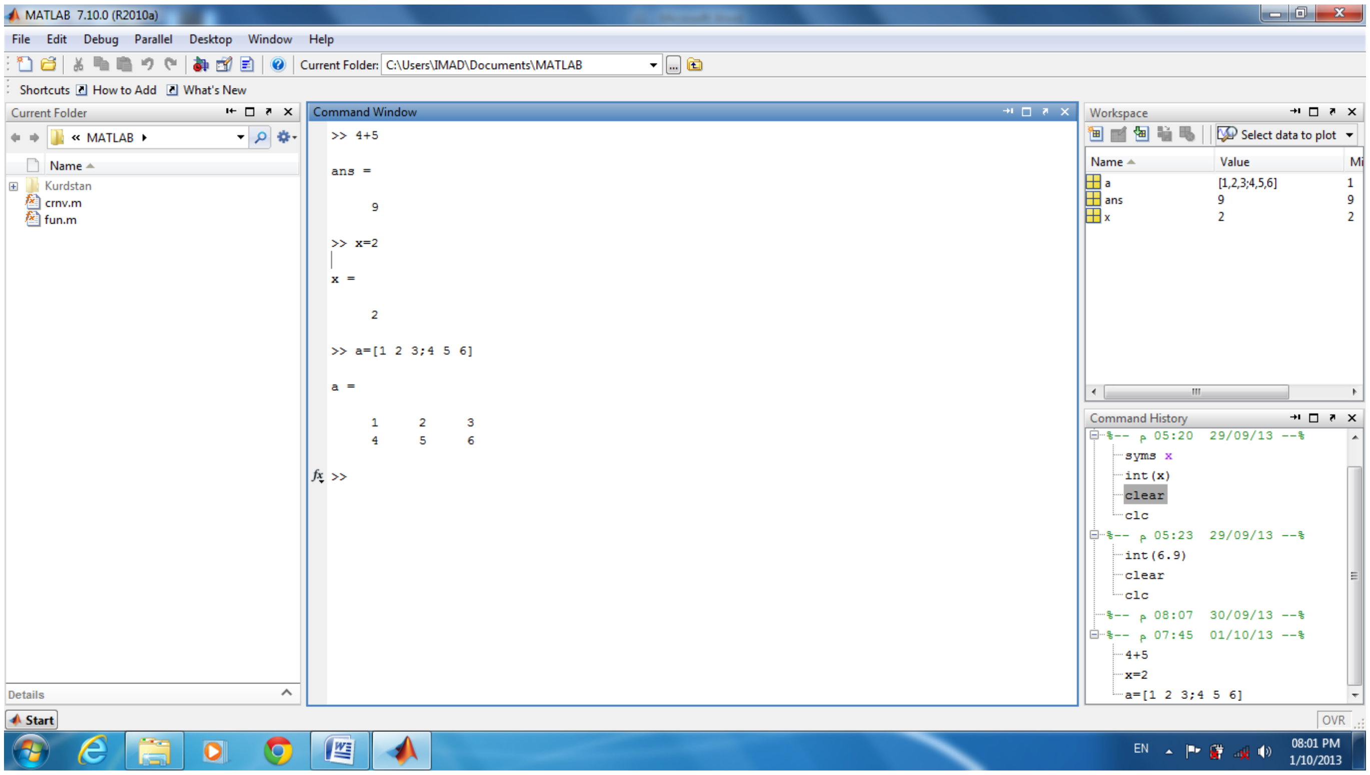
CHAPTER 1
INTRODUCTION to MATLAB

1.1 INTRODUCTION

MATLAB, which stands for **MAT**rix **LAB**oratory, is a state-of-the-art mathematical software package, which is used extensively in both academia and industry. It is an interactive program for numerical computation and data visualization, which along with its programming capabilities provides a very useful tool for almost all areas of science and engineering. Unlike other mathematical packages, such as MAPLE or MATHEMATICA, MATLAB cannot perform symbolic manipulations without the use of additional Toolboxes. It remains however, one of the leading software packages for numerical computation

1.1.1 Starting MATLAB

However you start MATLAB, you will briefly see a window that displays the MATLAB logo as well as some product information, and then a MATLAB Desktop window will launch. That window will contain a title bar, a menu bar, a tool bar and four embedded windows. The largest and most important window is the Command Window on the middle, the Command History Window and the Workspace in right, the Current Directory Browser in left. For now we concentrate on the Command Window in order to get you started issuing MATLAB commands as quickly as possible. At the top of the Command Window, you may see some general information about MATLAB, perhaps some special instructions for getting started or accessing help, but most important of all, you will see a command prompt (>>). If the Command Window is "active," its title bar will be dark, and the prompt will be followed by a cursor (a blinking vertical line). That is the place where you will enter your MATLAB commands. If the Command Window is not active, just click in it anywhere. Figure 1.1 contains an example of a newly launched MATLAB Desktop.



```
>> 4+5

ans =

     9

>> x=2

x =

     2

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

a =

     1     2     3
     4     5     6

>>
```

Name	Value	Mi
a	[1,2,3;4,5,6]	1
ans	9	9
x	2	2

```
%-- م 05:20  29/09/13 --%
    syms x
    int(x)
    clear
    clc
%-- م 05:23  29/09/13 --%
    int(6.9)
    clear
    clc
%-- م 08:07  30/09/13 --%
%-- م 07:45  01/10/13 --%
    4+5
    x=2
    a=[1 2 3;4 5 6]
```

Figure 1.1

a. **Command Window: -** Use the Command Window to enter variables and to run MATLAB functions and scripts. MATLAB displays the results. Press the up arrow key ↑ to recall a statement you previously typed. Edit the statement as needed, and then press Enter to run it.

b. **Command History:-**Statements you enter in the Command Window are logged with a timestamp in the Command History. From the Command History, you can view and search for previously run statements, as well as copy and execute selected statements.

c. **Workspace: -** The workspace consists of the set of variables built up during a session of using the MATLAB software and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces**.**

d. **Current Directory Browser:-**the files and subdirectories it contains are listed in the Current Directory Browser.

1.1.2 Typing in Command Window

Abort
In order to *abort* a command in MATLAB, hold down the control key and press c to generate a local abort with MATLAB.

The Semicolon (;)
If a semicolon (;) is typed at the end of a command the output of the command is not displayed.

Typing %
When percent symbol (%) is typed in the beginning of a line, the line is designated as a comment. When the *enter* key is pressed the line is not executed.

The clc Command
Typing *clc* command and pressing *enter* cleans the command window. Once the *clc* command is executed a clear window is displayed.

Help
MATLAB has a host of built-in functions. For a complete list, refer to MATLAB users
guide or refer to the *on line Help*. To obtain help on a particular topic in the list, *e.g.*, inverse, type *help inv*.

1.1.3 Display Format

MATLAB has several different screen output formats for displaying numbers. These formats can be found by typing the help command: help format in the Command Window. A few of these formats are shown in Table

Command	Description	Example
Format short	Fixed-point with 4 decimal digits	>> 351/7 ans = 50.1429
Format long	Fixed-point with 14 decimal digits	>> 351/7 ans = 50.14285714285715
Format short *e*	Scientific notation with 4 decimal digits	>> 351/7 ans = 5.0143*e*+001
Format long *e*	Scientific notation with 15 decimal digits	>> 351/7 ans = 5.014285714285715*e*001

Command	Description	Example
Format short *g*	Best of 5 digit fixed or floating point	>> 351/7 ans = 50.143
Format long *g*	Best of 15 digit fixed or floating point	>> 351/7 ans = 50.1428571428571
Format bank	Two decimal digits	>> 351/7 ans = 50.14
Format compact	Eliminates empty lines to allow more lines with information displayed on the screen	
Format loose	Adds empty lines (opposite of compact)	

1.2 Arithmetic Operations

The symbols for arithmetic operations with scalars are summarized below in Table

Arithmetic operation	Symbol	Example
Addition	+	$6 + 3 = 9$
Subtraction	–	$6 - 3 = 3$
Multiplication	*	$6 * 3 = 18$
Right division	/	$6 / 3 = 2$
Left division	\	$6 \backslash 3 = 3 / 6 = 1 / 2$
Exponentiation	^	$6 \wedge 3$ ($6^3 = 216$)

1.3 Elementary Math Built in Functions

MATLAB contains a number of functions for performing computations which require the use of logarithms, elementary math functions, and trigonometric math functions. List of these commonly used elementary MATLAB mathematical built-in functions are given in Tables.

Function	Description
abs(x)	Computes the absolute value of x.
sqrt(x)	Computes the square root of x.
round(x)	Rounds x to the nearest integer.
fix(x)	Rounds (or truncates) x to the nearest integer toward 0.
floor(x)	Rounds x to the nearest integer toward $-\infty$.
ceil(x)	Rounds x to the nearest integer toward ∞.
sign(x)	Returns a value of -1 if x is less than 0, a value of 0 if x equals 0, and a value of 1 otherwise.
rem(x,y)	Returns the remainder of x/y. for example, **rem(25, 4)** is 1, and **rem(100, 21)** is 16. This function is also called a **modulus** function.
exp(x)	Computes e^x, where e is the base for natural logarithms, or approximately 2.718282.
log(x)	Computes $\ln x$, the natural logarithm of x to the base e.
log10(x)	Computes $\log_{10} x$, the common logarithm of x to the base 10.

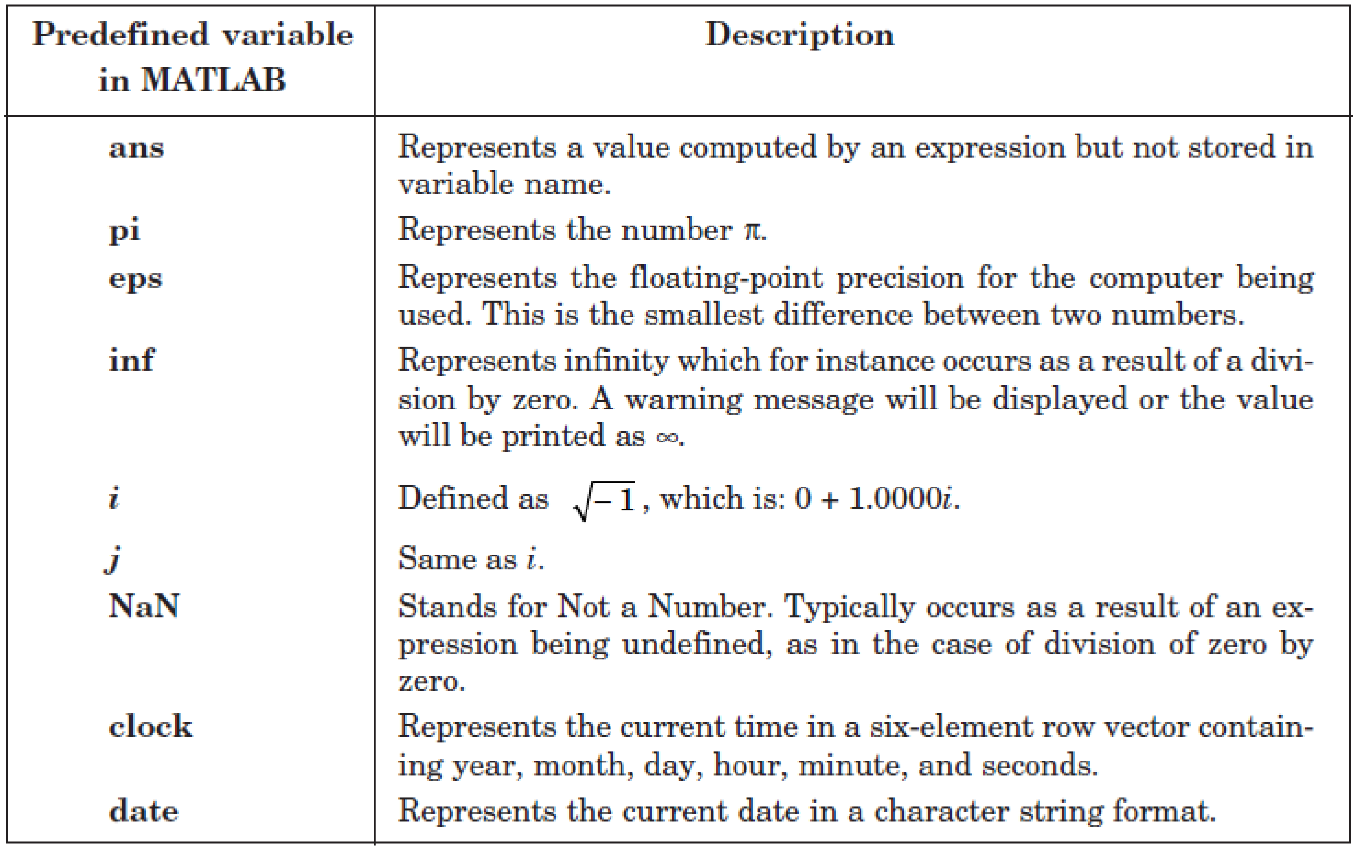
Function	**Description**
sin(x)	Computes the sine of x, where x is in radians.
cos(x)	Computes the cosine of x, where x is in radians.
tan(x)	Computes the tangent of x, where x is in radians.
asin(x)	Computes the arcsine or inverse sine of x, where x must be between -1 and 1. The function returns an angle in radians between $-\pi/2$ and $\pi/2$.
acos(x)	Computes the arccosine or inverse cosine of x, where x must be between -1 and 1. The function returns an angle in radians between 0 and π.
atan(x)	Computes the arctangent or inverse tangent of x. The function returns an angle in radians between $-\pi/2$ and $\pi/2$.
atan2(y, x)	Computes the arctangent or inverse tangent of the value y/x. The function returns an angle in radians that will be between $-\pi$ and π, depending on the signs of x and y.
sinh(x)	Computes the hyperbolic sine of x, which is equal to $\frac{e^x - e^{-x}}{2}$.
cosh(x)	Computes the hyperbolic cosine of x, which is equal to $\frac{e^x + e^{-x}}{2}$.
tanh(x)	Computes the hyperbolic tangent of x, which is equal to $\frac{\sinh x}{\cosh x}$.
asinh(x)	Computes the inverse hyperbolic sine of x, which is equal to $ln(x + \sqrt{x^2 + 1})$.
acosh(x)	Computes the inverse hyperbolic cosine of x, which is equal to $ln(x + \sqrt{x^2 - 1})$.
atanh(x)	Computes the inverse hyperbolic tangent of x, which is equal to $\ln \sqrt{\frac{1+x}{1-x}}$ for $\lvert x \rvert \leq 1$.

1.4 Variable Names

A variable is a name made of a letter or a combination of several letters and digits. Variable names can be up to 63 (in MATLAB 7) characters long (31 characters on MATLAB 6.0). MATLAB is case sensitive. For instance, XX, Xx, xX, and xx are the names of four different variables. It should be noted here that not to use the names of a built-in functions for a variable. For instance, avoid using: sin, cos, exp, sqrt, ..., etc. Once a function name is used to define a variable, the function cannot be used.

1.4.1 Predefined Variables

MATLAB includes a number of predefined variables. Some of the predefined variables that are available to use in MATLAB programs are summarized in Table



Predefined variable in MATLAB	Description
ans	Represents a value computed by an expression but not stored in variable name.
pi	Represents the number π.
eps	Represents the floating-point precision for the computer being used. This is the smallest difference between two numbers.
inf	Represents infinity which for instance occurs as a result of a division by zero. A warning message will be displayed or the value will be printed as ∞.
i	Defined as $\sqrt{-1}$, which is: $0 + 1.0000i$.
j	Same as i.
NaN	Stands for Not a Number. Typically occurs as a result of an expression being undefined, as in the case of division of zero by zero.
clock	Represents the current time in a six-element row vector containing year, month, day, hour, minute, and seconds.
date	Represents the current date in a character string format.

1.4.2 Command for Managing Variables

Table below lists commands that can be used to eliminate variables or to obtain information about variables that have been created. The procedure is to enter the command in the Command Window and the Enter key is to be pressed.

Command	Description
clear	Removes all variables from the memory.
clear x, y, z	Clears/removes only variables x, y, and z from the memory.
who	Lists the variables currently in the workspace.
whos	Displays a list of the variables currently in the memory and their size together with information about their bytes and class.

1.5 Complex Numbers

Complex numbers consist of two separate parts: a real part and an imaginary part. The basic imaginary unit is equal to the square root of -1. This is represented in MATLAB by either of two letters: i or j.

1.5.1 Creating Complex Numbers

The following statement shows one way of creating a complex value in MATLAB. The variable x is assigned a complex number with a real part of 2 and an imaginary part of 3:

x = 2 + 3i;

Another way to create a complex number is using the complex function. This function combines two numeric inputs into a complex output, making the first input real and the second imaginary:

```
x = 4;
y = -1;
z = complex(x, y)
z =
   4.0000 -1.0000i
```

1.5.2 Arithmetic operations with complex numbers

Operation	Result
$c_1 + c_2$	$(a_1 + a_2) + i(b_1 + b_2)$
$c_1 + c_2$	$(a_1 - a_2) + i(b_1 - b_2)$
$c_1 \bullet c_2$	$(a_1a_2 - b_1b_2) + i(a_1b_2 - a_2b_1)$
$\frac{c_1}{c_2}$	$\left(\frac{a_1a_2 + b_1b_2}{a_2^2 + b_2^2}\right) + i\left(\frac{a_2b_1 - b_2a_1}{a_2^2 + b_2^2}\right)$
$\lvert c_1 \rvert$	$\sqrt{a_1^2 + b_1^2}$ (magnitude or absolute value of c_1)
$c_1{}^*$	$a_1 - ib_1$ (conjugate of c_1)
(Assume that $c_1 = a_1 + ib_1$ and $c_2 = a_2 + ib_2$)	

1.5.3 Complex number functions

Function	Description
conj(x)	Computes the complex **conjugate** of the complex number x. Thus, if x is equal to $a + i\ b$, then **conj(x)** will be equal to $a - i\ b$.
real(x)	Computes the real portion of the complex number x.
imag(x)	Computes the imaginary portion of the complex number x.
abs(x)	Computes the absolute value of **magnitude** of the complex number x.
angle(x)	Computes the angle using the value of **atan2(imag(x), real(x));** thus, the angle value is between $-\pi$ and π.

CHAPTER 2
VECTORS AND MATRICES

2.1 Arrays

An array is a list of numbers arranged in rows and/or columns. A one-dimensional array is a row or a column of numbers and a two-dimensional array has a set of numbers arranged in rows and columns. An array operation is performed element-by-element.

2.1.1 Row Vector

A vector is a row or column of elements.
In a row vector the elements are entered with a space or a comma between the elements inside the square brackets. For example,
x = [7 – 1 2 – 5 8]

2.1.2 Column Vector

In a column vector the elements are entered with a semicolon between the elements inside the square brackets. For example,
x = [7 ; – 1 ; 2 ; – 5 ; 8]

2.2 Matrix

A matrix is a two-dimensional array which has numbers in rows and columns. A matrix
is entered row-wise with consecutive elements of a row separated by a space or a comma, and
the rows separated by semicolons or carriage returns. The entire matrix is enclosed within
square brackets. The elements of the matrix may be real numbers or complex numbers. For
example to enter the matrix,

$$A = \begin{bmatrix} 1 & 3 & -4 \\ 0 & -2 & 8 \end{bmatrix}$$

The MATLAB input command is

A = [1 3 – 4 ; 0 – 2 8]

Similarly, for complex number elements of a matrix B

$$B = \begin{bmatrix} -5x & \ln 2x + 7\sin 3y \\ 3i & 5 - 13i \end{bmatrix}$$

The MATLAB input command is

B = [– 5 * x log(2 * x) + 7 * sin (3 * y) ; 3i 5 – 13i]

2.3 Addressing Arrays

A colon can be used in MATLAB to address a range of elements in a vector or a matrix.

2.3.1 Colon for a vector

Va(:) – refers to all the elements of the vector Va (either a row or a column vector).
Va(m : n) – refers to elements m through n of the vector Va.
For instance

```
>> V = [2 5 – 1 11 8 4 7 – 3 11]
>> u = V(2 : 8)
u = 5 – 1 11 8 4 7 – 3 11
```

2.3.2 Colon for a matrix

Table below gives the use of a colon in addressing arrays in a matrix.

Command	Description
A(:, *n*)	Refers to the elements in all the rows of a column *n* of the matrix A.
A(*n*, :)	Refers to the elements in all the columns of row *n* of the matrix A.
A(:, *m* : *n*)	Refers to the elements in all the rows between columns *m* and *n* of the matrix A.
A(*m* : *n*, :)	Refers to the elements in all the columns between rows *m* and *n* of the matrix A.
A(*m* : *n*, *p* : *q*)	Refers to the elements in rows *m* through *n* and columns *p* through *q* of the matrix A.

2.3.3 Deleting Elements

An element, or a range of elements, of an existing variable can be deleted by reassigning
blanks to these elements. This is done simply by the use of square brackets with nothing typed in between them.

```
>> A = [1 2 3;4 5 6;7 8 10]
A =
        1 2 3
        4 5 6
        7 8 10
```

```
>> A(:, 2) = []
A =
        1 3
        4 6
        7 10
```

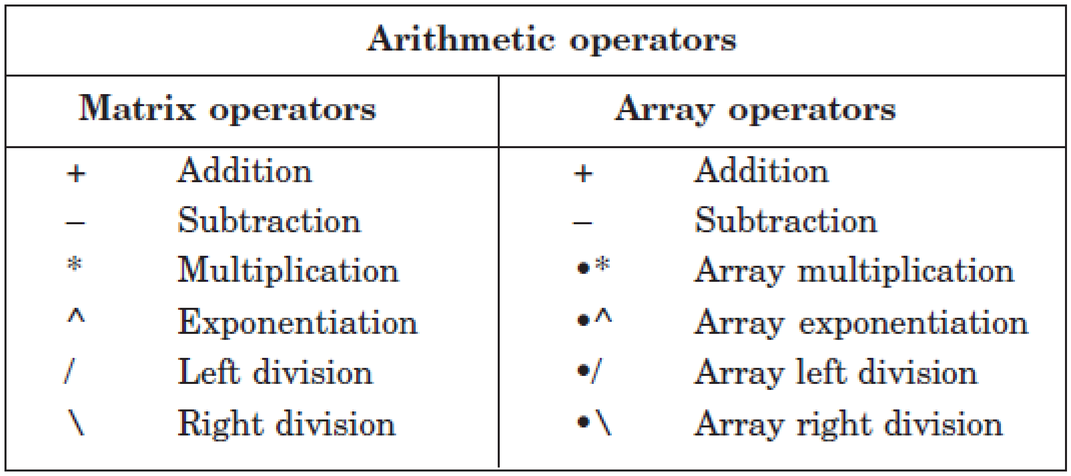
2.3.4 Adding Elements to a Vector or a Matrix

A variable that exists as a vector, or a matrix, can be changed by adding elements to it.
Addition of elements is done by assigning values of the additional elements, or by appending existing variables. Rows and/or columns can be added to an existing matrix by assigning values to the new rows or columns.

```
>> A = [A(:,1) [2 5 8]’ A(:,2)]
A =
        1 2 3
        4 5 6
        7 8 10
```

2.4 Element-by-element operations

Element-by-element operations can only be done with arrays of the same size. Elementby- element multiplication, division, and exponentiation of two vectors or matrices is entered in MATLAB by typing a period in front of the arithmetic operator. Table below lists these operations



Arithmetic operators			
Matrix operators		Array operators	
+	Addition	+	Addition
–	Subtraction	–	Subtraction
*	Multiplication	•*	Array multiplication
^	Exponentiation	•^	Array exponentiation
/	Left division	•/	Array left division
\	Right division	•\	Array right division

2.5 Identity, Ones and Zeros Matrix

The function *eye(m,n)*, ones(m,n) , zeros(m,n) returns an m-by-n rectangular identity ,all elements ones, all elements zero matrix and eye(n) returns an n-by-n square identity matrix and same for others.

2.6 Built-in Functions for Arrays

Table below lists some of the many built-in functions available in MATLAB for analyzing
arrays.

a. Transpose

In MATLAB, the transpose of the matrix A is denoted by A'

b. Diagonal of matrix

diag (A) :-When A is a matrix, creates a vector from the diagonal elements of A.

diag (*v*) When *v* is a vector, creates a square matrix with the elements of *v* in the diagonal.

```
>> v = [3 2 1];
>> A = diag(v)
A =
   3  0  0
   0  2  0
   0  0  1
```

```
>> A = [1 8 3 ; 4 2 6 ; 7 8 3]
A =
   1  8  3
   4  2  6
   7  8  3
>> vec = diag(A)
vec =
   1
   2
   3
```

Also we have tril or triu functions to find lower or upper triangular of matrix.

c.

Function	Description	Example

mean (*A*)	If A is a vector, returns the mean value of the elements	>> A = [3 7 2 16];
		>> mean (*A*)
		ans = 14

C = max (*A*)	If *A* is a vector, *C* is the largest element in *A*. If *A* is a matrix, *C* is a row vector containing the largest element of each column of *A*.	>> A = [3 7 2 16 9 5 18 13 0 4]; >> *C* = max (*A*) *C* = 18
[*d*, *n*] = max (*A*)	If *A* is a vector, *d* is the largest element in *A*, *n* is the position of the element (the first if several have the max value).	>> [*d*, *n*] = max (*A*) *d* = 18 *n* = 7

min (*A*)
 	
[*d*, *n*] = min (*A*)
	The same as max(*A*), but for the smallest element.
	
	The same as **[*d*, *n*] = max(*A*)**, but for the smallest element.
		>> *A* = [3 7 2 16];
		>> min (*A*)
		ans = 2

sum (*A*)	If *A* is a vector, returns the sum of the elements of the vector.	>> *A* = [3 7 2 16];
		>> sum (*A*)
		ans = 28

Function	Description	Example

sort (*A*)	If *A* is a vector, arranges the elements of the vector in ascending order.	>> *A* = [3 7 2 16];
		>> sort (A)
		ans = 2 3 7 16

median (*A*)	If A is a vector, returns the median value of the elements of the vector.	>> *A* = [3 7 2 16];
		>> median (A)
		ans = 5

std (*A*)	If A is a vector, returns the standard deviation of the elements of the vector.	>> *A* = [3 7 2 16];
		>> std (A)
		ans = 6.3770

det (*A*)	Returns the determinant of a square matrix *A*.	>> *A* = [1 2 3 4];
		>> det (A)
		ans = – 2

dot (*a*, *b*)	Calculates the scalar (dot) product of two vectors *a* and *b*. The vector can each be row or column vectors.	>> *a* = [5 6 7];
		>> *b* = [4 3 2];
		>> dot (*a*, *b*)
		ans = 52

cross (*a*, *b*)	Calculates the cross product of two vectors *a* and *b*, ($a \times b$). The two vectors must have 3 elements	>> *a* = [5 6 7];
		>> *b* = [4 3 2];
		>> cross (*a*, *b*)
		ans = – 9 18 – 9

inv (*A*)	Returns the inverse of a square matrix *A*.	>> *a* = [1 2 3; 4 6 8; – 1 2 3]; >> inv (A) ans = – 0.5000 0.0000 – 0.5000 – 5.0000 1.5000 1.0000 3.5000 – 1.0000 – 0.5000

2.7 Matrix dimension functions

- **length(A)**: - Length of vector or largest array dimension.
- **size(A)**: - returns the sizes of each dimension of array.
- **ndims(A)**: - Number of array dimensions.
- **reshape (A, *m*, *n*)**: - Rearrange a matrix A that has *r* rows and *s* columns to have *m* rows and *n* columns. *r* times *s* must be equal to *m* times *n*.

Example: **-** Let

$$A = \begin{pmatrix} 2 & 5 & 0 & 7 \\ 3 & 4 & -1 & 4 \end{pmatrix}$$

Then

```
>> length(A)
ans =
    4
>> size(A)
ans =
    2    4
>> ndims(A)
ans =
    2
>> reshape(A,4,2)
ans =
    2    0
    3   -1
    5    7
    4    4
```

2.8 Relational and Logical Operators

A relational operator compares two numbers by finding whether a comparison statement is true (1) or false (0). A logical operator examines true/false statements and produces a result which is true or false according to the specific operator.

Relational operators Table

Relational operator	Interpretation
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
= =	Equal
~ =	Not equal

Logical operators Table

Logical operator	Name	Description
& Example: *A* & *B*	AND	Operates on two operands (*A* and *B*). If both are true, the result is true (1), otherwise the result is false (0).
| Example: *A* | *B*	OR	Operates on two operands (*A* and *B*). If either one, or both are true, the result is true (1), otherwise (both are false) the result is false (0).
~ Example: ~ A	NOT	Operates on one operand (*A*). Gives the opposite of the operand. True (1) if the operand is false, and false (0) if the operand is true.

Example:-

```
>> 3>2
ans =
     1
>> 3<2
ans =
     0
>> k=3
k =
     3
>> k<9
```

```
a =
     1     0     2    -3
b =
     2     0    -1     5
>> b>a
ans =
     1     0     0     1
>> b~=1
ans =
     1     1     1     1
```

```
k =  -3
>> k>0 | k<0
ans =
     1
>> k>0 & k<0
ans =
     0
>> k<0
ans =
     1
```

```
a =
     1     0     2    -3
b =
     2     0    -1     5
>> a|b
ans =
     1     0     1     1
>> a&b
ans =
     1     0     1     1
```

ans = 1	>> a>0 ans = 1 0 1 0	>> ~(k<0) ans = 0	>> ~a ans = 0 1 0 0

Additional logical built-in functions Table

Function	Description	Example
xor(*a*, *b*)	Exclusive or. Returns true (1) if one operand is true and the other is false	>>xor(8, – 1) ans = 0 >>xor(8, 0) ans = 1
all(*A*)	Returns 1 (true) if all elements in a vector *A* are true (nonzero). Returns 0 (false) if one or more elements are false (zero). If *A* is a matrix, treats columns of *A* as vectors, returns a vector with 1's and 0's.	>>*A* = [5 3 11 7 8 15] >>all(*A*) ans = 1 >>*B* = [3 6 11 4 0 13] >>all(*B*) ans = 0
any(A)	Returns 1 (true) if any element in a vector *A* is true (nonzero). Returns 0 (false) if all elements are false (zero). If *A* is a matrix, treats columns of *A* as vectors, returns a vector with 1's and 0's.	>>*A* = [5 0 14 0 0 13] >>any(*A*) ans = 1 >>*B* = [0 0 0 0 0 0] >>any(*B*) ans = 0

find(*A*)
 	
find(*A* > *d*)
	If A is a vector, returns the indices of the non-zero elements.
	If A is a vector, returns the address of the elements that are larger than *d* (any relational operator can be used).
		>>*A* = [0 7 4 2 8 0 0 3 9]
		>>find(*A*)
		ans =
		2 3 4 11 8 9
		>>find(*A* > 4)
		ans =
		4 5 6

CHAPTER 3

PROGRAMMING IN MATLAB

3.1 M-files: Scripts and functions

To take advantage of MATLAB's full capabilities, we need to know how to construct long (and sometimes complex) sequences of statements. This can be done by writing the commands in a file and calling it from within MATLAB. Such files are called "m-files" because they must have the filename extension ".m". This extension is required in order for these files to be interpreted by MATLAB.

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

Script files contain a sequence of usual MATLAB commands, that are executed (in order) once the script is called within MATLAB. For example, if such a file has the name compute .m , then typing the command compute at the MATLAB prompt will cause the statements in that file to be executed. Script files can be very useful when entering data into a matrix.

Example:- Create the script file then write a program to find the roots of equation $2x^2 -3x+1=0$.

Sol:

```
% This Program written to Find the Roots of Equation 2x^2-
3x+1=0%
 a=2; b=-3; c=1;
 x1=(-b+sqrt(b^2-4*a*c))/2
 x2=(-b-sqrt(b^2-4*a*c))/2
```

3.2 Input and output command

a. Disp(X)

disp(X) displays an array, without printing the array name. If X contains a text string, the string is displayed. Another way to display an array on the screen is to type its name, but this prints a leading "X=," which is not always desirable. Note that *disp* does not display empty arrays.

```
>> disp('string expression');
string expression
>> A=[3,2;2,3];
disp(A);
     3     2
     2     3
```

b. Input

The input function can be used for requesting user input. For example,

```
r=input('value for r: ');
```

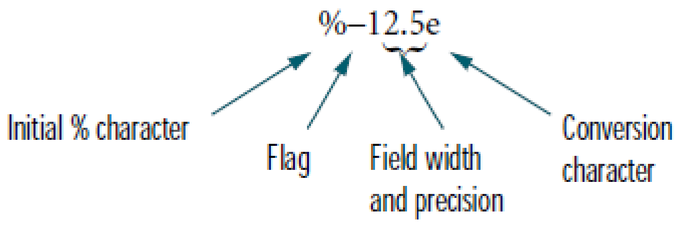
displays `value for r:`

to the screen and waits for the user to enter an expression which is then assigned to r.

c. fprintf

The *fprintf* command displays output (text and data) on the screen or saves it to a file. The output can be formatted using this command.

fprintf (format,A,...) writes to standard output—the screen. The format string specifies notation, alignment, significant digits, field width, and other aspects of output format. It can contain ordinary alphanumeric characters; along with escape characters, conversion specifiers, and other characters, organized as shown below:



For more information, see “Tables” and “References”

Code	Conversion instruction
`%s`	format as a string
`%d`	format with no fractional part (integer format)
`%f`	format as a floating-point value
`%e`	format as a floating-point value in scientific notation
`%g`	format in the most compact form of either `%f` or `%e`
`\n`	insert newline in output string
`\t`	insert tab in output string

Example:-

```
 >> x = 2; y = sqrt(x);
 >> fprintf('The squrt root of %g is %9.4f\n',x,y)
        The squrt root of 2 is    1.4142
```

Example:-

```
>> x = 1:4; y = sqrt(x);
>> fprintf('The squrt root of %d is %4.2f\n',[x;y])
The squrt root of 1 is 1.00
The squrt root of 2 is 1.41
The squrt root of 3 is 1.73
The squrt root of 4 is 2.00
```

3.3 Conditional Execution

3.3.1 Conditional Execution or Branching:

As the result of a comparison, or another logical (true/false) test, selected blocks of program code are executed or skipped. Conditional execution is implemented with if, if...else, and if...elseif constructs, or with a switch construct.

There are three types of if constructs

1. Plain if

2. if...else

3. if...elseif

if Constructs
Syntax:

```
 if expression

        block of statements

 end
```

The *block of statements* is executed only if the *expression* is true.

Example:

```
  if a < 0

    disp('a is negative');

  end
```

One-line format uses comma after if *expression*

```
if a < 0, disp('a is negative'); end
```

if. . . else

Multiple choices are allowed with **if. . . else** and **if. . . elseif** constructs

```
if x < 0
```

```
  error('x is negative; sqrt(x) is imaginary');
else
  r = sqrt(x);
end
```

if. . . elseif
It's a good idea to include a default **else** to catch cases that don't match preceding **if** and **elseif** blocks

```
if x > 0
   disp('x is positive');
elseif x < 0
   disp('x is negative');
else
   disp('x is exactly zero');
end
```

3.3.2 The switch Construct

A switch construct is useful when a test value can take on discrete values that are either integers or strings.

Syntax:

```
switch expression
  case value1,
    block of statements
  case value2,
    block of statements
    ...
 otherwise,
   block of statements
end
```

Example:

```
color = ’...’; % color is a string
switch color
 case ’red’
   disp(’Color is red’);
 case ’blue’
   disp(’Color is blue’);
 case ’green’
   disp(’Color is green’);
 otherwise
  disp(’Color is not red, blue, or green’);
end
```

3.4 Repetition or Looping

A sequence of calculations is repeated until either
1. All elements in a vector or matrix have been processed
or
2. The calculations have produced a result that meets a predetermined termination criterion
Looping is achieved with for loops and while loops.

3.4.1 for loops

for loops are most often used when each element in a vector or matrix is to be processed.

Syntax:

```
for index = expression
  block of statements
end
```

Example: Sum of elements in a vector

```
x = 1:5; % create a row vector
sumx = 0; % initialize the sum
for k = 1:length(x)
  sumx = sumx + x(k);
end
```

3.4.2 for loop variations

Example: A loop with an index incremented by two

```
for k = 1:2:n
  ...
end
```

Example: A loop with an index that counts down

```
for k = n:-1:1
  ...
end
```

Example: A loop with non-integer increments

```
for x = 0:pi/15:pi
 fprintf('%8.2f %8.5f\n',x,sin(x));
end
```

Note: In the last example, x is a scalar inside the loop. Each time through the loop, x is set equal to one of the columns of 0:pi/15:pi.

3.4.3 while loops

while loops are most often used when an iteration is repeated until some termination criterion is met.

Syntax:

```
while expression
  block of statements
end
```

The block of statements is executed as long as expression is true.

Example: Here is a simple example of a script M-file that uses while to numerically sum the infinite series 1/14 + 1/24 + 1/34 +· · ·, stopping only when the terms become so small (compared to the machine precision) that the numerical sum stops changing:

```
n = 1; oldsum = -1; newsum = 0;
while newsum > oldsum
  oldsum = newsum;
  newsum = newsum + nˆ(-4);
  n = n + 1;
end
newsum
```

Note:-It is (almost) always a good idea to put a limit on the number of iterations to be performed by a while loop.
An improvement on the preceding loop,

```
 n = 1; oldsum = -1; newsum = 0;
 maxit = 25; it = 0;
  while newsum > oldsum & it<maxit
    oldsum = newsum;
    newsum = newsum + nˆ(-4);
    n = n + 1;
    it = it + 1;
 end
newsum
```

CHAPTER 4
FUNCTIONS

4.1 Function files

on the other hand, play the role of user defined commands that often have input and output. You can create your own commands for specific problems this way, which will have the same status as other MATLAB commands. Let us give a simple example. The text below is saved in a file called log3.m and it is used to calculate the base 3 logarithm of a positive number. The text file can be created in a variety of ways, for example using the built-in MATLAB editor through the command edit (that is available with MATLAB 5.0 and above), or your favorite (external) text editor (e.g. *Notepad* or *Wordpad* in Microsoft Windows). You must make sure that the filename has the extension ".m" !

```
function [a] = log3(x)
a = log(abs(x))./log(3);
end
» log3(5)
ans =
      1.4650
```

Syntax:

The first line of a function m-file has the form:

```
    function [outArgs] = funName(inArgs)
```

outArgs are enclosed in [ ]

- outArgs is a comma-separated list of variable names
- [ ] is optional if there is only one parameter
- functions with no outArgs are legal

inArgs are enclosed in ( )

- inArgs is a comma-separated list of variable names
- functions with no inArgs are legal

Example:- Write the function to find Fibonnaci sequence.

```
function f = Fib1(n)
F=zeros(1,n+1);
F(2) = 1;
for i = 3:n+1
F(i) = F(i-1) + F(i-2);
End
End
```

Example:- Write the function to find area and perimeter of triangle.

```
function [A s] = area(a,b,c)

s = (a+b+c)/2;

A = sqrt(s*(s-a)*(s-b)*(s-c));

End
```

4.2 Loop and Function Cotroal

a. return

return is used to force an exit from a function. This can have the effect of escaping from a loop. Any statements following the loop that are in the function body are skipped.

b. break

break is used to escape from an enclosing while or for loop. Execution continues at the end of the enclosing loop construct.

Example: - write a function to check that the input number is prime or not.

```
function [p] = prim(x)
p=0;
for i=2:x-1
    if rem(x,i)==0
        p=1;
        return
    end
end
end
```

in above example we have only one element input but if we write program to work with array of element as an input we must write the program as follow

```
function p = prim(a)
p=zero(size(a));
for i=1:length(a)
  for j=2:a(i)-1
    if rem(a(i),j)==0
        p(i)=1;
        Break
    end
  end
end
end
```

c. **continue**

The continue statement passes control to the next iteration of the for loop or while loop in which it appears, skipping any remaining statements in the body of the loop. In nested loops, continue passes control to the next iteration of the for loop or while loop enclosing it.

Example:- find the s where

$$s = \sum_{\substack{i=0 \\ i\neq 3}}^{n} \frac{1}{i-3}$$

```
n=input('n=')
s=0;
for i=0:n
    if i==3, continue, end
    s=s+1/(i-3);
end
s
```

d. error (‘*text*’)

Terminates execution and displays the message contained in text on the screen.

Example: - Write the function to replace the last row with last column of input matrix.

```
function b = lr2lc(a)
 [n m]=size(a);
 if n~=m
     error('The matrix must be square')
 else
     b=a;
     b(n,:)=a(:,m);
     b(:,m)=a(n,:);
 end
 end
```

CHAPTER 5
STRINGS

5.1 Character Strings

While Matlab is primarily intended for number crunching, there are times when it is desirable to manipulate text, as is needed in placing labels and titles on plots. In Matlab, text is referred to as character strings or simply strings.

5.1.1 String Construction

Character strings in Matlab are special numerical arrays of ASCII values that are displayed as their character string representation. For example:

```
>> text = ’This is a character string’
   text =
          This is a character string
>> size(text)
   ans =
         1 26
>> whos
   Name    Size    Bytes    Class
   ans     1x2      16      double array
   text    1x26     52      char array
```

Grand total is 28 elements using 68 bytes

5.1.2 ASCII Codes

Each character in a string is one element in an array that requires two bytes per character for storage, using the ASCII code. This differs from the eight bytes per element required for numerical or double arrays. The ASCII codes for the letters 'A' to 'Z' are the consecutive integers from 65 to 90, while the codes for 'a' to 'z' are 97 to 122. The function abs returns the ASCII codes for a string.

```
>> text = ’This is a character string’
```

```
text =
     This is a character string
>> d = abs(text)
 d =
  Columns 1 through 12

    84  104  105  115  32  105  115  32  97  32  99

104

  Columns 13 through 24

    97  114  97  99  116  101  114  32  115  116  114

105

  Columns 25 through 26

    110  103
```

The function **char** performs the inverse transformation, from ASCII codes to a string:

```
>> char(d)
ans =
    is a character string
```

The relationship between strings and their ASCII codes allow you to do the following:

```
>> alpha = abs('a'):abs('z');

>> disp(char(alpha))

    Abcdefghijklmnopqrstuvwxyz
```

5.2 Strings are Arrays

Since strings are arrays, they can be manipulated with array manipulation tools:

```
>> text = 'This is a character string';
>> u = text(11:19)
u =
   character
```

As with matrices, character strings can have multiple rows, but each row must have an equal number of columns. Therefore, blanks are explicitly required to make all rows the same length.
For example:

```
>> v = [’Character strings having more than’
        ’one row must have the same number ’
        ’of columns - just like matrices ’]
v =
   Character strings having more than
   one row must have the same number
   of columns - just like matrices
```

```
>> size(v)
    ans =
         3 34
```

5.3 Concatenation of Strings

Because strings are arrays, they may be concatenated (joined) with square brackets. For example:

```
>> today = ’May’;
>> today = [today, ’ 18’]
   today =
          May 18
```

5.4 String Conversions

num2str(*x*) :- Converts the matrix ***x*** into a string representation with about 4 digits and an exponent if required.

The num2str function can be used to convert numerical results into strings for use in formatting displayed results with disp. For example;

```
a=input('a=');
for i=1:length(a);
    if isprime(a(i))
```

```
        disp([num2str(a(i)) ' is prime'])
    else
        disp([num2str(a(i)) ' is not prime'])
    end
end
```

run

```
 a=[2 8 5]
  2 is prime
  8 is not prime
  5 is prime
```

str2num(*s*) :- converts the string ***s*** which is an ASCII character representation of a numeric value, to numeric representation. **str2num** also converts string matrices to numeric matrices. If the input string does not represent a valid number or matrix, str2num(s) returns the empty matrix. For example;

```
 s='423'  »  str2num(s)=423

  s='3 0 2 5' »  a=str2num(s)
  a= 3 0 2 5 is matrix

  str2num('2 4; 6 8')

    ans =
            2     4
            6     8
```

5.5 String Functions

blanks(n) :-Returns a string of n blanks. Used with disp, eg.

```
disp ([’xxx’ blanks(10) ’yyy’]).
  xxx          yyy
```

disp(blanks(n)') moves the cursor down n lines.

deblank(*s*):- Removes trailing blanks from string s.

strtrim(*s*):- returns a copy of string s with all leading and trailing white-space characters removed.

```
        str='       AB   CD         '
       deblank(str)= '       AB   CD'
       strtrim(str)='AB   CD'
```

findstr(*s1, s2*) :- Find one string within another. Returns the starting indices of any occurrences of the shorter of the two strings in the longer.

```
s = 'Find the starting indices of the shorter string';

findstr(s, 'the')
ans =
     6    30

findstr('the', s)
ans =
     6    30
```

strcmp(*s1, s2*) :- Returns 1 if strings s1 and s2 are identical and 0 otherwise.

```
  strcmp('Yes', 'No')
  ans =
        0
  strcmp('Yes', 'Yes')
  ans =
        1
```

strncmp(*s1, s2, n*) :- Returns 1 if the first n characters of the strings s1 and s2 are identical and 0 otherwise.

strmatch(*str, strs*) :- Searches through the rows of the character array of strings strs to find strings that begin with string str, returning the matching row indices.

strrep(*s1, s2, s3*):- Replaces all occurrences of the string s2 in string s1 with the string s3. The new string is returned.

lower(*s*) :- Converts any uppercase characters in string s to the corresponding lowercase character and leaves all other characters unchanged.

upper(*s*):- Converts any lower case characters in s to the corresponding upper case character and leaves all other characters unchanged.

eval(*s*):- Execute the string s as a Matlab expression or statement.

Example:- write a program to draw the graph of input function.

```
f = input( ’Enter function (of x) to be plotted: ’,
’s’);
x = 0:0.01:10;
y= eval(f);
plot(x,y),grid
```

Here’s how the command line looks after you enter the function:

```
>> Enter function (of x) to be plotted:
    exp(-0.5*x) .* sin(x)
```

Example:- You can concatenate strings to create a complete expression for input to eval. This code shows how eval can create 10 variables named P1, P2, ...P10, and set each of them to a different value.

```
for i=1:10
  eval(['P',int2str(i),'= i.^2'])
 end
```