Communication Systems Laboratory

ECE/EEE F311

Experiment   2

Introduction to MATLAB

BITS-Pilani, Hyderabad Campus

January 2021

EXPERIMENT 2

Introduction to MATLAB programming

1.1 Introduction

In MATLAB when all the commands were executed in the Command Window those commands cannot be saved and executed again for several times. Therefore, a different way of executing repeatedly commands with MATLAB is:

1. to *create* a file with a list of commands,
2. *save* the file, and
3. *run* the file.

- Programming in MATLAB is very similar to that found in C/C++.
- The most important in programming is tools available in MATLAB, which enables the programmer to develop large scientific applications with shorter code and in far less time.
- Program files in MATLAB are called M files with programming features like for, while loops or switch, break and if control statements.
- MATLAB program files are named with extension. m.

If needed, corrections or changes can be made to the commands in the file. The files that are used for this purpose are called script files or *scripts* for short.
This section covers the following topics:
- M-File Scripts
- M-File Functions

1.2 Creating M Files:

The editor available in MATLAB can be invoked in a number of ways as,
- Choose new files or open file from the file sub menu in MATLAB desktop.
- Click on the icon for new file or open file on the tool bar available in MATLAB desktop.
- Enter the command *edit* or *edit file name* at the command prompt in the MATLAB command window.
- When the edit command is given, the Editor is started with the untitled and the file can be named while storing or saving it in the appropriate folder.
- The commands given in the command window are saved in the Command history window.
- M-file can be created using these statements by selecting them in the Command History Window, right-clicking and selecting Create M-file option.

## 1.3 Types of M-Files:

There are two types of M-file:
- ❖ Script files
- ❖ Functions

A *script file* is an external file that contains a sequence of MATLAB statements. Script files have a filename extension .m and are often called M-files. M-files can be *scripts* that simply execute a series of MATLAB statements, or they can be *functions* that can accept arguments and can produce one or more outputs.

Both types of files contain MATLAB statements, but have the following major differences:

- The Script files are the simplest of M-files that basically do not accept input arguments or return output arguments.
- Function files, on the other hand, accept input arguments and return output arguments. These can be 'called' or 'continued' within a Script file or another Function file.
- The script files operate on data in the workspace. Any variable, that script creates, remains in the workspace even after the script finishes and can be used for further computations. On the other hand, Functions files treat the internal variables as local to that function and have their own workspace.
- Function file starts with keyboard *function.*

  - Another function which takes an input array and returns the sum and product of its elements as outputs
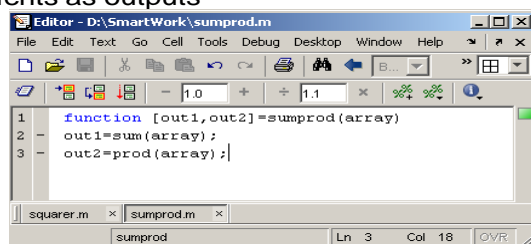
  - The function sumprod(.) can be called from command window or an m-file as
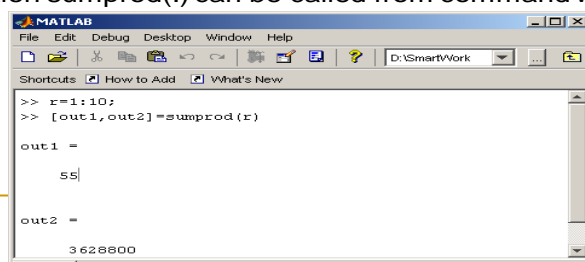
All variables created in a script file are added to the workspace.
- Variables already existing in the workspace may be overwritten.
- The execution of the script can be affected by the state variables in the workspace.

As a result, because scripts have some undesirable side-effects, it is better to code any complicated applications using rather function M-file.

Example 1

Consider the system of equations:

$x + 2y + 3z = 1$

$3x + 3y + 4z = 1$

$2x + 3y + 3z = 2$

Find the solution x to the system of equations.

Solution:
 Use the MATLAB *editor* to create a file: File -> New -> M-file.
 Enter the following statements in the file:

A = [1 2 3; 3 3 4; 2 3 3];

b = [1; 1; 2];

x = A\b

 Save the file, for example, example1. m.
 Run the file, in the command line, by typing:

>> example1

x =

-0.5000
1.5000
-0.5000

When execution completes, the variables (A, b, and x) remain in the workspace. To see a listing of them, enter whos at the command prompt.

Note: The MATLAB editor is both a text editor specialized for creating M-files and a graphical MATLAB debugger. The MATLAB editor has numerous menus for tasks such as *saving*, *viewing*, and *debugging*. Because it performs some simple checks and also uses color to differentiate between various elements of codes, this text editor is recommended as the tool of choice for writing and editing M-files.

Example 2

Plot the following cosine functions, $y1 = 2\cos(x)$, $y2 = \cos(x)$, and $y3 = 0.5 * \cos(x)$, in the interval $0 < x < 2\pi$.

Create a file, say example2.m, which contains the following commands:

```
x = 0:pi/100:2*pi;
y1 = 2*cos(x);
y2 = cos(x);
y3 = 0.5*cos(x);
plot (x, y1,'--',x,y2,'-',x,y3,':')
xlabel ('0 \leq x \leq 2\pi')
ylabel ('Cosine functions')
legend('2*cos(x)','cos(x)','0.5*cos(x)')
title ('Typical example of multiple plots')
axis ([0 2*pi -3 3])
```

Run the file by typing example2 in the Command Window.

Anatomy of M-File

This simple function shows the basic parts of an M-file.

```
function f = factorial(n) (1)
% FACTORIAL(N) returns the factorial of N. (2)
% Compute a factorial value. (3)
f = prod (1: n); (4)
```

The first line of a function M-file starts with the keyword function. It gives the function *name* and order of *arguments*. In the case of function factorial, there are up to one output argument and one input argument. Table 4.1 summarizes the M-file function.

As an example, for $n = 5$, the result is,

```
>> f = factorial (5)
f = 120
```

Part no.	M-file element	Description
(1)	Function definition line	Define the function name, and the number and order of input and output arguments
(2)	H1 line	A one line summary description of the program, displayed when you request Help
(3)	Help text	A more detailed description of the program
(4)	Function body	Program code that performs the actual computations

Both *functions* and *scripts* can have all of these parts, except for the *function definition line* which applies to *function* only.

Differences between script files and function files

SCRIPTS	FUNCTIONS
- Do not accept input arguments or return output arguments.	- Can accept input arguments and return output arguments.
- Store variables in a workspace that is shared with other scripts	- Store variables in a workspace internal to the function.
- Are useful for automating a series of commands	- Are useful for extending the MATLAB language for your application

1.4 Input and Output Commands used in script file
When a script file is executed, the variables that are used in the calculations within the file must have assigned values. The assignment of a value to a variable can be done in three ways.
1. The variable is defined in the script file.
2. The variable is defined in the command prompt.
3. The variable is entered when the script is executed.

When the file is executed, the user is *prompted* to assign a value to the variable in the command prompt. This is done by using the input command. Here is an example.

% This script file calculates the average of points
% scored in three games.
% The point from each game are assigned to a variable
% by using the `input' command.
game1 = input ('Enter the points scored in the first game ');
game2 = input ('Enter the points scored in the second game ');
game3 = input ('Enter the points scored in the third game ');
average = (game1+game2+game3)/3

The following shows the command prompt when this script file (saved as example3) is executed.

```
>> example3
>> Enter the points scored in the first game 15
>> Enter the points scored in the second game 23
>> Enter the points scored in the third game 10
average =
16
```

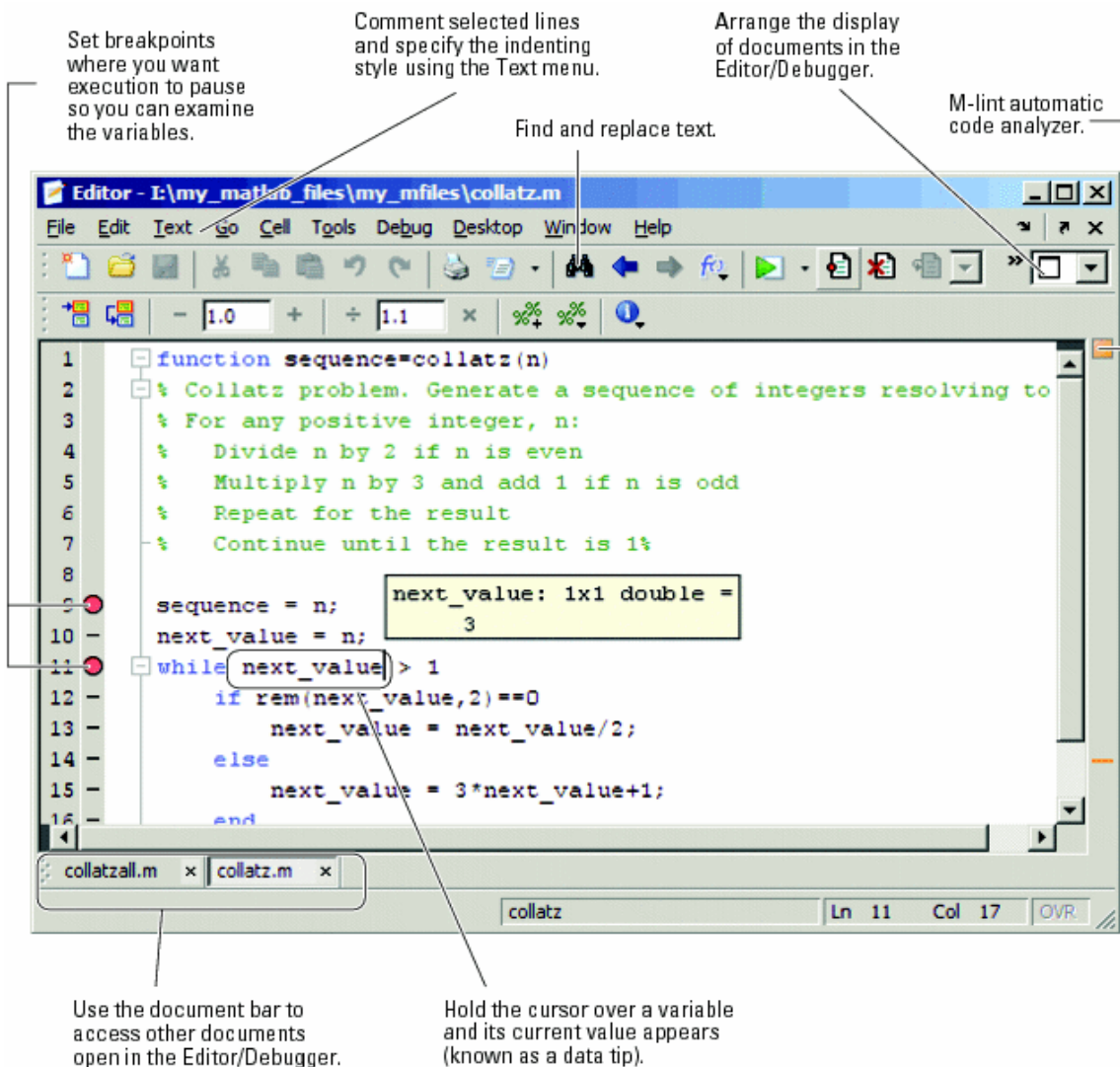The input command can also be used to assign *string* to a variable.

As discussed before, MATLAB automatically generates a *display* when commands are executed. In addition to this automatic display, MATLAB has several commands that can be used to generate displays or outputs.
Two commands that are frequently used to generate output are: disp and fprintf.

```
disp      . Simple to use.
          . Provide limited control over the appearance of output


fprintf   . Slightly more complicated than disp.
          . Provide total control over the appearance of output
```

1.5   MATLAB Editor
- MATLAB Editor not only contains basic text editing features but also provides the user with additional facilities such as debugging tools.
- The Editor/Debugger has proper graphical user interface for ease in programming.
- It also supports automatic indenting and highlighting to indicate special parts of the program such as keywords, text strings and comments.
- It also allows a user to observe the values of the variables/constants in the program.
- Features like setting/clearing breakpoints and executing programs line by line are also available to aid the programmer in effectively debugging the program

Set breakpoints where you want execution to pause so you can examine the variables.

Comment selected lines and specify the indenting style using the Text menu.

Find and replace text.

Arrange the display of documents in the Editor/Debugger.

M-lint automatic code analyzer.

Use the document bar to access other documents open in the Editor/Debugger.

Hold the cursor over a variable and its current value appears (known as a data tip).

Opening the Editor:

The Editor can be opened in a number of ways, which are as follows:

- Selecting *New* from the File menu in the MATLAB desktop main menu.
- Clicking on the new file icon on the toolbars.
- Typing the command edit in the command window at the command prompt.

If a file already exists and is to be modified, the Editor may be started by

- Selecting open from the file menu in the MATLAB desktop main menu.
- Clicking on the open file icon on the toolbar.
- Typing the command edit filename in the command window at the command prompt.

EDITOR MAIN MENU:

The appearance of the Editor Window is similar to normal Windows present in the Windows XP Operating System.

- It contains Title bar that displays the name of the file being edited.
- If the changes made in the file have not been saved is indicated by an asterisk (*) that appears after the name of the file in the Title bar. The Editor window also contains main menu and tool bar.
- The main menu is composed of File, Edit, Text, Debug, Tool etc.

❖ **File Menu** contains options to open save and print files and also quick/exit from Editor/debugger or MATLAB, and has choice for importing data from outside or saving the workspace in*.mat files.

❖ **Edit Menu** contains items that are found in normal text editor like undo, redo, cut, copy, paste, find, replace text, select all, delete etc. It also has features for clearing the command window, command history and work space.

❖ **Text Menu** contains the item Evaluate Selection which is used to evaluate an expression and display the answer in the command window. This is very helpful when a large section of program needs to be commented. It is included in the text menu is **code folding**. Its function is just like a windows explorer which hides the folder contents or displays them when desired by the user.

❖ **Go Menu** includes options for moving *forward or back* in the program text. It also includes option for moving to a particular line in the text files that are open in the editor. It opens a separate dialogue box which prompts the user to enter the line number and select the program text file from the ones that are correctly open in the editor. It also provides *bookmark.*

❖ **Cell Menu** deals with such section i.e. cells of the program code.

❖ **Tools Menu** contains options called M-Lint. M-Lint displays inconsistencies and suspicious constructs in program files. These options can be utilized by the user to find such inconsistencies. Another tool available in this file is *profiler.*

❖ **Debug Menu** contains various features embedded in the editor for debugging the program and also contains options to run the program in the current text window.

❖ **Desktop Menu** allows the user to dock or undock the editor window and organize the desktop to the satisfaction of the user.

❖ **Window Menu** provides choice of how the different text files in the editor window should appear on the desktop i.e. in tiled manner or the two files viewed in horizontally or vertically split screen etc. It also allows the user to move the command window, command history window, help window or work space window.

❖ **Help Menu** allows the use to seek the help about various features of MATLAB editor/debugger.

TOOLBAR:
- The editor window in addition to the main menu contains the complete functional tool bar.
- The features found in other text editor like new file, open file, save, cut, copy, paste, undo, redo, print or available on the tool bar.
- The Toolbar also provides graphical interface for finding a particular text move forward or backward in the code.
- It also contains icons for the complete set of debugging tools i.e. setting or clearing break points, executing program line by line, stepping into the functions, stepping out, continuing execution after stopping at a statement indicated by the break points and quitting debugging.

1.6 Control Structures
There are two broad categories of these control structures:
- **Looping structures**
- **Branching structures**

❖ **Looping structures** cause specific group of instructions to be repeated for a fixed number of times or until the given condition is specified. Some of the loop structures are for, while etc.

❖ **Branching structures** select the specific blocks of the program for execution depending upon the condition satisfied. Some of the branch structures are if, if-else, switch etc.

Besides these, there are some other commands such as break, error and continue to control the execution of such structures.

1.6.1   Looping Structures:
The loops are classified depending upon the manner in which the iterations in a program are controlled.
There are two forms of loops
  ➢ *for*
  ➢ *while*
In case of for loop, the number of repetitions is already known, whereas in while loop, execution is repeated until the condition is satisfied.

❖ For loop:
The for loop control structure is used for unconditional looping. In this control structure, a group of statements, say 'n' in number, are executed repeatedly for a fixed specified number of times, which is known before the loop starts and is decided by an index.

The syntax of for statement is given as follows:

> *for index=initial value: increment: last value*
> *Statement 1;*
> *Statement 2;*
> *:*
> *Statement n;*
> *end*
> *next statement*

The index is assigned an initial value and then a group of statements is executed till end statement. After completion of one loop, the index is incremented by increment that is specified and then the group of statements is executed again, provided the new value of index does not exceed the last value. After this, *for loop* is terminated and the control is passed to the statement next to the end statement corresponding to a particular for loop.

In more generalized form, *for loop* syntax is expressed as follows:
> *for index=expression,*
> *Statement 1;*
> *Statement 2;*
> *:*
> *:*
> *Statement n;*
> *end*

❖ Nested for loop:
When one for loop is embedded in another for loop, the two loops are called nested for loops. The necessary condition is that one loop is completely nested within the other.

❖ **While Loop:**

A while loop is used for executing a group of statements repeatedly on the basis of a condition. In this control structure, the group of statements associated with the while statement are executed repeatedly or indefinitely until the given condition is true (or non-zero).

The general form of while loop is

> *While(condition),*
> *Statement 1;*
> *:*
> *:*
> *Statement n;*
> *end;*

1.6.2   Branching Control Structures

The branch control structures allow selecting and executing specific blocks of statements out of many statement blocks depending upon the condition. These include different types of *if structures, switch control structure* and so on. The *if structure* is a conditional control structure which allows some group of statements to be executed only if the given condition is true.

❖ *if control structures:*

In *if control structure*, a group of statements are executed only if the condition is true.

Different forms of *if control structures:*

- *if structure*
- *if-else structure*
- *if-elseif-else structure*
- *nested if structure*

❖ *if control structure:*

Syntax:

> *if expression*
> > *statement 1*
> > *statement 2*
> > *:*
> *end*
> *next statement*

where expression is a logical expression and gives the result either true or false (non-zero or 0). If the expression returns true result, then the group of statement after if statement is executed and the control goes to the next statement after the end statement. If the expression is false, then this group of statements is skipped and control goes to next statement which appears after the end statement.

❖ *if-else structure:*

Syntax:

> *if expression*
> > *statement 1*
> > *statement 2*
> > *:*
> *else*
> > *statement 1*
> > *statement 2*
> > *:*
> *end*
> *next statement*

❖ *if - else if - else structure:*

10

Syntax:
if expression
       statement 1
       statement 2
       :
elseif
       statement 1
       statement 2
       :
else
       statement 1
       statement 2
       :
end
next statement

❖ **nested if structure:**
Syntax:

if (condition1)
       statement 1
       statement 2
       :
       :
if(condition2)
       statement 1
       statement 2
end
statement 3
statement 4
end

❖ **switch statement:**
    The switch-case-otherwise structure is used for multi-way branching. It allows selection of a particular block of statements from several available blocks. A flag (any variable or expression) is used as a switch and based on the current value of the flag; particular block of statements is selected. This structure is used for logical branching. A flag can be a single integer, character or expression.
Syntax:
    switch flag,
    case value1,
       block1 statements:
    ----
    case value2,
       block2 statements:
    ----
    otherwise,
       Last block statements;
    end;
    next statement

    Here, depending upon the value of the flag, the particular block of statement is executed. If flag=value1, block1 statements are executed and the program control will exit the switch structure; if flag=value2, block2 is executed and the program control will exit the switch structure; if the flag does not match any case value then the last block associated with otherwise is executed.
    The matching end statement indicates the end of the switch structure.

❖ **Break statement:**

The break statement inside a loop is used to terminate the execution of the loop and transfer the control to the next statement after the end statement of the loop. The control is shifted even if the condition for execution of the loop is true.

❖ **Continue statement:**

The continue statement is used to skip the statement in the current pass in loop but will not exit from the loop. The remaining iterations are continued for remaining iterations are continued for remaining values of the index or decision variable.

❖ **Error statement:**

Sometimes, on getting incorrect input value, the program needs to be terminated and it is desired to display a message. This can be done using error command.

Syntax:

e*rror ('message to be displayed on encountering error')*

❖ **try-catch structure:**

Generally, while executing a program if an error occurs, MATLAB aborts the program showing the error message. However, by using try-catch structure, errors found by MATLAB are captured, giving the user ability to control the way MATLAB responds to errors. This allows the user to handle errors within the program without the program without stopping it.

Syntax:

try
    statement 1
    statement 2
    ----
catch
    statement 1
    statement 2

End

Here, firstly the statement in try block are executed, then one of the following two cases is possible:
- If no error occurs, then statements in the catch block are skipped and program control shifts to statement after try-catch structure.
- In case an error occurs in any of the statements of try block, then the program stops executing the remaining statements in try block and executes the statements in the catch block and finally exits from the try-catch structure.

Control structures	Description
for	Repeat the statements specific number of times
while	Repeat the statements if the condition is true or non-zero
if	Conditionally execute the statements
switch	Switch between cases depending upon the condition
break	Terminate the execution of for or while loop
continue	Pass the control to next iteration for or while loop
error	Display message and abort function
try-catch	Catch the error generated by MATLAB

1.7 Writing Functions in MATLAB Programing

Function Sub Programs
- A large and complex task can be easily executed if divided into smaller subprograms
- Function subprograms are program segments that are written for implementing a sub-task or a sub-module
- These are developed separately and executed independently.
- MATLAB also has in its store numerous function subprograms for common tasks related to many fields such as mathematical functions, matrices, graphs, solving differential equations and Fourier Transforms, etc.
- Besides such pre-defined functions, MATLAB also contains function subprograms which may be needed frequently for solving many Scientific/Engineering problems.
- Functions are M-files that accept input arguments and return output arguments. The structure of the function subprogram is similar to that of any other program, expect for the function definition statement that should be included at the start of the function. The function subprogram includes commands and function and the function file is normally named after the function name.

❖ **Function Definition Line:**

The function definition line informs MATLAB complex that the M-file defines a function subprogram.

Syntax:

Function [output arguments] = function _ name (input arguments)

Where
1. 'function' is a keyword that indicates that the program following this statement is a function subprogram.
2. 'function name' is the name of the function and defined just as other variable names in the program. Normally, it is chosen same as the name of a function subprogram file name.
3. The list of input arguments, if present, is enclosed in parentheses separated by commas.
4. The function output arguments are enclosed in square brackets separated by commas.

Some valid function definition lines and their corresponding file names are as follows:

Function Definition Line	File Name	Remarks
function [ ] = circle _area (r) ;	circle_area.m	no output arguments
function rect_area(l,b)	rect_area.m	no output arguments
function [r1,r2] = roots(a,b,c)	roots.m	r1&r2 are output arguments

1.8 Types of Functions

Functions can be classified into different categories based on their features and the manner in which they are applied. Some of them are as follows:

A . sub functions

B. private functions

C. nested functions
D. Inline functions

❖ **Sub functions:**

Function files can contain code for more than one function. The first function in the file is called primary function. The primary function is called by the name of the function M-file. Additional functions defined within the same file are called sub functions. These functions are visible only to the primary function or other sub functions defined in the same M-file. Each sub function begins with its function definition line. These sub function follow each other and may occur in any order, as long as the primary function appears first.

❖ **Private Functions**

Private functions are functions that are stored in subfolders with the special name 'private'. These functions are visible only to the functions in their parent folder. For example, let the folder 'student' be on the MATLAB search pat. A subfolder called "private' cab be created in the folder 'student'. The folder 'private' can contain functions that can be called only be the functions in its parent folder 'student'. Private functions are invisible outside their parent folder. Therefore, they can have the same names as functions in other folders. This is useful if the user wants to create a different version of a particular function while retaining the original in another folder.

❖ **Nested Functions**

Just as a for loop may be nested within another for loop, a function may also be nested within another function. The nested functions provide a way to pass information, without passing it through input/output arguments and without using global variables;

Basic format of a   simple nested function is as follows;
F1=primary function (………………)

--------------------------   *% Code of function F1*

F2 = nested function (……………………………)

--------------------------   *% Code of nested function F2*
----
end   *%end of nested function f2*
--------------------------   *%remaining code of F1*

--------------------------   *% end of nested function F1*

A primary function can contain several nested functions within itself. The end statements mark the end of each nested function and the primary function.

❖ **Inline function:**

   Mathematical expressions can be evaluated by creating functions defining the expression as follows:
function y = mexp(x)
*Y=x^3 + 4*x^2 +5* +6;*
Another way to represent the mathematical function at the command prompt is to create an inline object of the function mexp mentioned can be defined as:
*FX=inline ('x^3 + 4*x^2 + 5* +6', 'x')*
General form of an inline function is
        Function name = inline ('character string expression', 'list of variables')
Where character string expression and list of variables are enclosed within single quotes.

❖ Function Handles:

A function Handle stores all the information about the function which is needed for its execution later.

A function handle can be created by any of the following ways:

   a. Using @ operator

   b. Using str2func function.

Using @ operator:

      To create a function f(t) = 4e^(-2t) for t in the range of 0 to 5.

      Function timres = func(t)

      Timres = 4* exp(-2*t)

      End

Function handle can be created by the following line:

Handl = @func

Where handl is the name of the function handle and func is the name of the function. The function func(T) can now be executed by just typing the name of the function handle and enclosing the arguments in the parenthesis. The following MATLAB command will evaluate the function func(t):

      Handle = @func

Ex:

Handle (0 :5)

Ans:

   = 4.000   0.5413   0.0733   0.0099   0.0013   0.002

Anonymous function:

Function handle created using @ operator can also be used to create functions called anonymous functions.

Syntax:

      Function name = @ (list of variables/arguments in function) function expression

Where @ indicates that the function name mentioned on the left hand side is a function handle and the list of variables appearing in the expression is enclose in brackets followed by expression without any punctuation marks.

Using str2func function:

Function handle can be created and used by typing the following line in the command window:

 Handle = str2func('func')

Handle (0: 1 :5)

Ans:

   = 4.000   0.5413   0.0733   0.0099   0.0013   0.002

Which gives the value if 4e^-2t for different values of t.

1.9   ERRORS AND WARNINGS

In many cases, it is desirable to take specific actions when different kinds of errors occur. For example, one may want to prompt the user for more input, display extended error or warning information, or repeat a calculation using default values. Error-handling capabilities of MATLAB allow checking of particular error conditions and execute appropriate program depending on the situation.

Types of Errors

There are basically two types of errors:

- Syntax errors
- Runtime errors

Syntax errors are caused by grammatical mistakes in the statement included in the program i.e. using or not using comma, semicolon, bracket or parenthesis, it may be caused by misspelling a function name or using wrong array indices and so on. MATLAB detects most of the syntax errors when the program

is compiled before its execution and displays a message describing the error, showing the line number of the statement where the error has occurred in the program file.

Runtime errors are errors that are caused mainly due to wrong logic used by the programmer. The statement given in the program may be syntactically correct but if the logic or algorithm is erroneous, the output or result of the program that is obtained on its execution will be erroneous. The programmer may be using a wrong variable or value of a variable or performing incorrect computations. The runtime errors become apparent when one obtains erroneous and unexpected results.

Debugging from the Command Line

The debugging commands can also be given and executed from the command prompt in the Command window. The different commands and their description are given in table:

Debugging command	Description	Syntax
dbstop	Set breakpoint	dbstop file name line number
dbclear	Clear breakpoint	dbclear filename line number
dbclear all	Clear all breakpoints	dbclear all
dbstop if error	Stop on warning, error or NaN/Inf generation	dbstop if warning error naninf infnan
dbstep	Single step execution	dbstep
dbstep in	Step into a function	dbstep in
dbcont	continue execution	dbcont
dbquit	Quit debugging	dbquit
dbstack	List function call stack	dbstack
dbstatus	List all breakpoints	dbstatus file_name
dbstep nlines	Execute one or more lines	dbstep nlines
dbtype	List M-file with line numbers	dbtype file_name
dbdown	Change local workspace context(down)	dbdown
dbup	Change local workspace context(up)	dbup
dbquit	Quit debug mode	dbquit

Questions

1. Given two matrices A and B of dimensions m x n and p x q respectively, the Kronecker product of A and B is a mp x nq dimensional matrix defined as $A \otimes B =$
$\begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$. Write a MATLAB program to find the Kronecker product of 2 given matrices without using the instruction "kron".

2. Write a MATLAB function named "matmul.m" to multiply two real-valued matrices. Using this function, compute the matrix exponential defined as $e^A = \sum_{i=1}^{\infty} \frac{1}{i!} A^i$. For your computation, you may use the first 10 terms in the series to approximate the matrix exponential.

3. Consider the sequence $\{a^n\}$, where n takes on the values 0, 1, 2, … and a is a non-negative less than 1. Write a MATLAB program to output the value of n such that the sequence value is no greater than a given threshold value for the first time. For instance, you may choose the threshold value to be 10^{-5}, and output the first value of n for which $|a^n| < 10^{-5}$. Use loops to implement your scheme. Derive this value theoretically, and check if your output from MATLAB matches this theoretical value.

4. Using MATLAB, generate a random vector such that each element in that vector is a zero mean, unit variance Gaussian random variable. Let the length of this vector be N. For various values of N, compute the quantity $\frac{1}{N}\sum_{i=1}^{N} x_i$, where x_i is the ith element in the vector. Use loops to implement your scheme.