CMR Institute of Technology
Department of ECE
IAT -3 Scheme and Solution
BEC358B – MATLAB Programming

USN

Internal Assessment Test 3 – March 2024

Sub:	**MATLAB Programming**	Sub Code:	**BEC358B**	Branch:	ECE
Date: 06/03/2024	Duration: 90 minutes	Max Marks: 50	Sem/Sec:	3^{rd} (A,B,C,D)	OBE

	ANSWER ANY FIVE FULL QUESTIONS	MARKS	CO	RBT
1(i)	Create three anonymous functions corresponding to the following expressions: $$f(x) = x^4 - 8x^3 + 17x^2 - 4x - 20$$ $$g(x) = x^2 - 4x + 4$$ $$h(x) = x^2 - 4x - 5$$ (i) Create anonymous function to evaluate $f(x) - g(x)h(x)$ at $x = 3$ (ii) Create anonymous function to evaluate $f(x) - g(x)h(x)$ at $x = [1\ 2\ 3\ 4\ 5]$ (iii) Create anonymous function to plot $f(x)$ and $\frac{f(x)}{g(x)h(x)}$ over $x \in [-5, 5]$	6	CO3	L3
1(ii)	Explain briefly about the following MATLAB commands: (i) pwd (ii) dir (iii) ls (iv) cd	4	CO3	L1
2(i)	Solve the following set of simultaneous linear algebraic equations using appropriate MATLAB Symbolic computations $$x + 3y - z = 2$$ $$x - y + z = 3$$ $$3x - 5y = 4$$	6	CO3	L3
2(ii)	Demonstrate briefly about reshaping matrices using MATLAB commands and suitable examples	4	CO4	L2
3(i)	Illustrate with examples the three different kinds of files for reading data in MATLAB's workspace	6	CO3	L2
3(ii)	What are Command-Line Functions? Explain with examples	4	CO4	L1
4(i)	Write a MATLAB script file to solve the set of linear system equations: $$5x_1 + 2rx_2 + rx_3 = 2$$ $$3x_1 + 6x_2 + (2r-1)x_3 = 3$$ $$2x_1 + (r-1)x_2 + 3rx_3 = 5$$ Use $r = 1$	6	CO5	L3
4(ii)	Illustrate the Recursion function in MATLAB with appropriate examples	4	CO5	L2
5	List with simple statements the use of MATLAB commands $\boldsymbol{break}$, $\boldsymbol{error}$ and $\boldsymbol{return}$ to control the execution of scripts and functions.	10	CO5	L2
6(i)	Compute the following MATLAB commands: (i)$\boldsymbol{fix}([-2.33\ \ 2.66])$ (ii)$\boldsymbol{floor}([-2.33\ \ 2.66])$ (iii)$\boldsymbol{ceil}([-2.33\ \ 2.66])$ (iv)$\boldsymbol{round}([-2.33\ \ 2.66])$	4	CO4	L3

6(ii)	Apply the six relational operations in MATLAB to the operands $\boldsymbol{x} = [\mathbf{1\ 5\ 3\ 7}]$ and $\boldsymbol{y} = [\mathbf{0\ 2\ 8\ 7}]$.Comment on the results obtained	6	CO4	L3
7	Write short notes on (i) M-Lint Code Analyzer (ii) Nested Functions (iii)$\boldsymbol{for\ loops\ and\ while\ loops}$	10	CO5	L2

1. (i)

```
>> f=@(x) x^4 - 8*x^3 + 17* x^2 - 4*x -20

f =

  function_handle with value:

    @(x)x^4-8*x^3+17*x^2-4*x-20

>> g=@(x) x^2 - 4*x +4

g =

  function_handle with value:

    @(x)x^2-4*x+4

>> h=@(x) x^2 - 4*x -5

h =

  function_handle with value:

    @(x)x^2-4*x-5

>> h=@(x) x^2 - 4*x -5
```

(i) Anonymous function to evaluate $\boldsymbol{f(x) - g(x)h(x)}$ at $\boldsymbol{x = 3}$

```
>> v=@(x)f(x) - g(x)*h(x)

v =

  function_handle with value:

    @(x)f(x)-g(x)*h(x)

>> v(3)

ans =

    -6
```

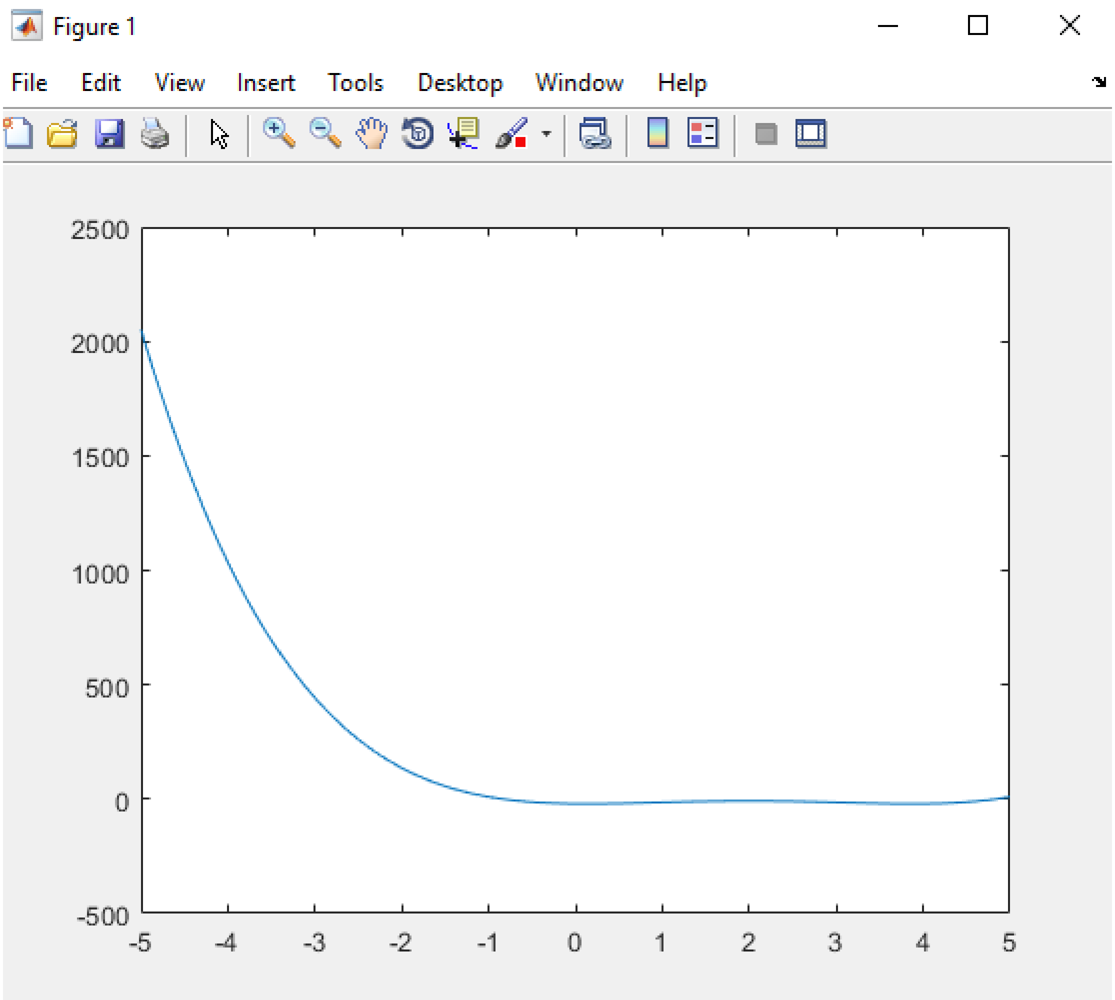
(ii)Anonymous function to evaluate $f(x) - g(x)h(x)$ at $x = [1\ 2\ 3\ 4\ 5]$

```
>> f=@(x)x.^4 - 8*x.^3 + 17* x.^2 - 4*x - 20

f =

  function_handle with value:

    @(x)x.^4-8*x.^3+17*x.^2-4*x-20

>> g = @(x)x.^2 - 4*x + 4

g =

  function_handle with value:

    @(x)x.^2-4*x+4

>> h=@(x)x.^2 - 4*x - 5

h =

  function_handle with value:

    @(x)x.^2-4*x-5
```

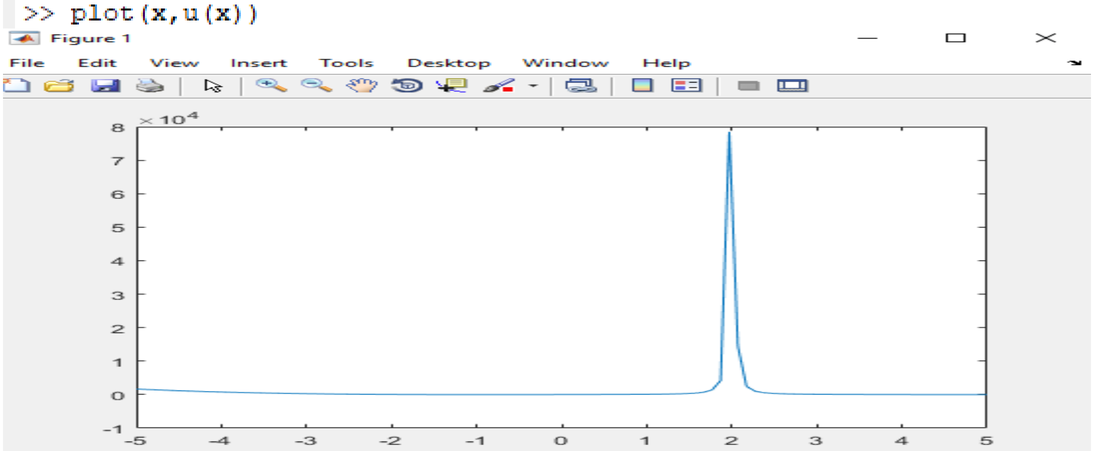
```
>> w=@(x)f(x)-g(x).*h(x)

w =

  function_handle with value:

    @(x)f(x)-g(x).*h(x)

>> x=[1 2 3 4 5]

x =

     1     2     3     4     5

>> w(x)

ans =

    -6    -8    -6     0    10
```

(ii) Anonymous function to plot $f(x)$ and $\frac{f(x)}{g(x)h(x)}$ over $x \in [-5, 5]$

```
>> x=linspace(-5,5);
>> plot(x,f(x))
```




```
>> u=@(x)f(x)./g(x).*h(x)

u =

  function_handle with value:

    @(x)f(x)./g(x).*h(x)

>> plot(x,u(x))
```



1. (ii) MATLAB Commands

pwd – Print(show)working directory

```
Command Window
>> pwd

ans =

    'C:\Users\admin'
```

- dir – Show the contents of the current directory

```
log.m
mocktest-1.ipynb
ntuser.dat.LOG1
ntuser.dat.LOG2
ntuser.ini
pol.m
random module.ipynb
random.txt
set.py
solvexf.m
strings.ipynb
stud.txt
student.txt
test3D.txt
test3b.txt
xydata.mat
```

- ls – List the contents of the directory
- cd x - change the directory to x

2. (i)Solution of simultaneous linear algebraic equations using appropriate MATLAB Symbolic computations

Command Window

```
>> syms x y z
>> exp1='x+3*y-z-2'

exp1 =

    'x+3*y-z-2'

>> exp2='x-y+z-3'

exp2 =

    'x-y+z-3'

>> exp3='3*x-5*y-4'

exp3 =

    '3*x-5*y-4'
```

```
>> [x,y,z]=solve(exp1,exp2,exp3)
```

```
x =

33/16


y =

7/16


z =

11/8
```

```
>> subs(exp1)
```

```
ans =

0
```

2. (ii) Reshaping matrices using MATLAB commands

- Matrices can be reshaped into a vector or any other appropriately sized matrix:

➢ As a vector:

- All the elements of matrix A can be strung into a single-column vector b by the command $b = A\ (:)$ (matrix A is stacked in vector b column wise)

➢ As a differently sized matrix:

- If matrix A is an $m\ x\ n$ matrix, it can be reshaped into a $p\ x\ q$ matrix, as long as $m\ x\ n = p\ x\ q$, with the command $\boldsymbol{reshape\ (A\ ,\ p,\ q)}$
- For a 6 x 6 matrix A,

  $reshape\ (A\ , 9,\ 4)$ transforms A into a 9 x 4 matrix

  $reshape(A, 3{,}12)$ transforms A into a 3 x 12 matrix

3. (i) Three different kinds of files for reading data in MATLAB's workspace

- Three different kinds of files for reading data into MATLAB's workspace:

➢ Mat-file: This is MATLAB's native binary format file for saving data

➢ Two commands - ***save*** and ***load*** make it particularly easy to save data into and load data from these files

➢ M-file: If you have a text file containing data, or you want to write a text file containing data that you would eventually like to read in MATLAB, making it an M-file may be an excellent option.

➢ first create this file and save it as TempData .m

```
% TempData: Script file containing data on monthly maximum temperature
Sl_No = [1:12]';
Month = char('January','February','March','April','May','June',...
        'July','August','September','October','November','December');
Ave_Tmax = [22 25 30 34 36 30 29 27 24 23 21 20]';
```

➢ Microsoft Excel file - data can be imported from a Microsoft Excel spreadsheet into MATLAB

➢ use MATLAB's import wizard, invoked by typing ***uiimport*** on the command prompt , or by clicking on ***File ->Import Data***

➢ MATLAB also provides a special function ***xlsread*** for reading Excel's spreadsheets as .xls files

➢ Here ***xlsread*** is used to read mixed data from an Excel file **TempData.xls**

- This file contains column titles in the first row, numeric data in the first and third column, and text data in the second column.

TempData.xls

SN	Month	Ave. Tmax
1	January	22
2	February	25
3	March	30
4	April	34
5	May	36
6	June	30
7	July	29
8	August	27
9	September	24
10	October	23
11	November	21
12	December	20

3. (ii) Command-Line Functions

   (a)Inline Functions

A mathematical function, such as F(x) or F(x, y), usually requires just the values of the independent variables for computing the value of the function. One way to evaluate such functions is by programming them in function files. This is done by defining inline functions-functions that are created on the command line. We can define these functions using the built-in function inline

The syntax for creating an inline function is particularly simple:

```
F = inline('function formula')
```

Thus, a function such as $F(x) = x^2 \sin(x)$ can be coded as

```
F = inline('x^2 * sin(x)')
```

Our inline function can only take a scalar x as an input argument. We can modify it easily by changing the arithmetic operator to accept array argument:
F=inline (‘x. - 2. *sin (x) ').
Once the function is created, you can use it as a function independently (e.g., type F (5) to evaluate the function at x = 5) or in the input argument of other functions that can evaluate it.

(b)Anonymous Functions

Anonymous functions are functions without names, created and referred by their handles. A function handle, created internally by MATLAB and stored in a user defined variable, is basically the identity of the function. An anonymous function is created by the command

```
f = @(input list) mathematical expression
```

where f is the function handle. The input list can contain a single variable or several variables separated by commas. After creating the function, you can use it with its handle to evaluate the function or pass it as an argument to other functions.

Examples:

`fx = @(x) x^2 - sin(x);`	creates a function $f(x) = x^2 - \sin x$,
`fx(5)`	evaluates $f(x)$ at $x = 5$,
`fxy = @(x,y) x^2 + y^2;`	creates a function $f(x, y) = x^2 + y^2$,
`fxy(2,3)`	evaluates $f(x, y)$ at $x = 2$ and $y = 3$,
`fx = @(x) x.^2 - sin(x)`	*vectorizes* the function $f(x)$, and
`x=[0:.1:pi/2]; plot(x,fx(x))`	plots $f(x)$ over 0 to $\pi/2$.

4. (i) MATLAB script file to solve the set of linear system equations

Solution depend on the format $Ax = b$

A depends on the parameter r

$$\begin{bmatrix} 5 & 2r & r \\ 3 & 6 & 2r-1 \\ 2 & r-1 & 3r \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 2 \\ 3 \\ 5 \end{Bmatrix}$$

Program1.m

```
%Program to solve System of Linear Equations
r=1;
A = [ 5 2*r r; 3 6 2*r-1; 2 (r-1) 3*r];
b = [2;3;5];
det_A = det(A)
x = A\b
```

```
>> Program1

det_A =

    64


x =

   -0.0312
    0.2344
    1.6875
```

4. (ii)Recursion Function in MATLAB

- The MATLAB programming language supports recursion, i.e. , a function can call itself during its execution

- Thus, recursive algorithms can be directly implemented in MATLAB
- Example:

➢ Computing the nth term in the Fibonacci (0, 1, 1, 2, 3, 5, 8…)

➢ Label the terms asF_0,F_1,F_2etc…

➢ Then the recursion relationship for generating this sequence is:
$F_k = F_{k-1} + F_{k-2}$ for $k > 2$

➢ The seeds are F_0 = 0 and F_1= F_2 = 1

➢ The nth term in this sequence can be computed by the following recursive function:

➢ The nth term in this sequence can be computed by the following recursive function:

```
function Fn = fibonacci(n)
% FIBBONACI: computes nth term in the Fibonacci sequence
% written by Abhay, May 15, 09, modified by RP, June 1, 09
if n==0, Fn = 0;                              % Fn=0 for n=0
  elseif n==1 | n==2, Fn = 1;                 % Fn=1, for n=1 OR n=2
  else Fn = fibonacci(n-1) + fibonacci(n-2);  % recursion relation
end
```

5. The use of MATLAB commands $\boldsymbol{break}$, $\boldsymbol{error}$ and $\boldsymbol{return}$ to control the execution of scripts and functions

Control Flow - Break

- The command break inside a for or while loop terminates the execution of the loop, even if the condition for execution of the loop is true
- Examples: (Assume that the variables used in the codes below are predefined)

```
1.  for i=1:length(v)
        if v(i) < 0          % check for negative v
            break            % terminate loop execution
        end
        a = a + v(i);        % do something
    end
```

```
2.  x = exp(sqrt(163));
    while 1
        n = input('Enter max. number of iterations ')
        if n <= 0
            break        % terminate loop execution
        end
        for i=1:n
            x = log(x);  % do something
        end
    end
```

- If the loops are nested then break terminates only the innermost loop

Control Flow - Error

- The command error (' message') inside a function or a script aborts the execution, displays the error message *message*, and returns the control to the keyboard
- Example:

```
function c = crossprod(a,b);
% crossprod(a,b) calculates the cross product axb.
if nargin~=2          % if not two input arguments
   error('Sorry, need two input vectors')
end
if length(a)==2       % begin calculations
   ....
end
```

Control Flow - Return

- The command *return* simply returns the control to the invoking function
- Example:

```
function animatebar(t0,tf,x0);
% animatebar animates a bar pendulum.
:
disp('Do you want to see the phase portrait?')
ans = input('Enter 1 if YES, 0 if NO ');
                          % see text for description
if ans==0                 % if the input is 0
   return                 % exit function


 else
    plot(x,...)            % show the phase plot
 end
```

6. (i)

`fix`	round toward 0, *Example:* `fix([-2.33 2.66])` $= [-2 \quad 2]$.
`floor`	round toward $-\infty$, *Example:* `floor([-2.33 2.66])` $= [-3 \quad 2]$.
`ceil`	round toward $+\infty$, *Example:* `ceil([-2.33 2.66])` $= [-2 \quad 3]$.
`round`	round toward the nearest integer, *Example:* `round([-2.33 2.66])` $= [-2 \quad 3]$.

6. (ii)

There are six relational operators in MATLAB:

<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal
~=	not equal

These operations result in a vector or matrix of the same size as the operands, with 1 where the relation is true and 0 where it is false.

Examples: If $x = [1\ \ 5\ \ 3\ \ 7]$ and $y = [0\ \ 2\ \ 8\ \ 7]$, then

`k = x < y`	results in $k = [0\,0\,1\,0]$	because $x_i < y_i$ for $i = 3$,
`k = x <= y`	results in $k = [0\,0\,1\,1]$	because $x_i \leq y_i$ for $i = 3$ and 4,
`k = x > y`	results in $k = [1\,1\,0\,0]$	because $x_i > y_i$ for $i = 1$ and 2,
`k = x >= y`	results in $k = [1\,1\,0\,1]$	because $x_i \geq y_i$ for $i = 1, 2,$ and 4,
`k = x == y`	results in $k = [0\,0\,0\,1]$	because $x_i = y_i$ for $i = 4$, and
`k = x ~= y`	results in $k = [1\,1\,1\,0]$	because $x_i \neq y_i$ for $i = 1, 2,$ and 3.

7. Short notes on (i) M-Lint Code Analyzer (ii) Nested Functions (iii)*for loops and while loops*

(i)M – Lint Code Analyzer

When you write a program in MATLAB, and create a script or a function, you want to make sure that your program

- uses correct syntax for each statement,
- has proper function definition line if it is a function,
- uses appropriate built-in functions, and
- contains no unresolvable references.

MATLAB provides an assistant to help in this task. It is called the M-Lint Automatic Code Analyzer (offers automatic corrections too). It is an excellent facility for helping in developing error free codes. There are two basic ways in which we can use M-Lint code analyzer:

- **On fresh code as you write it:** When you open a new M-file in the MATLAB editor using File->New->Blank M- File or Function M-File from the MATLAB menu, M-Lint code analyzer is pressed into service automatically. As you write the lines of code, M-Lint, as a nice assistant, starts working quietly, watching over your shoulders, and lists its objections politely and symbolically in the right-hand margin of the editor window. There is a colored small square on the top (in the right margin) that indicates the level of M-Lint's happiness with your code-a red-faced square indicates error, a green-faced square is a signal to march on. Below the square, there may be orange or red-colored lines corresponding to a particular line of code. Place your cursor on these colored lines (or, alternatively, on the underlined items in your code) one by one to see the message your assistant has left for you while it checked the line. Orange lines contain advisories (warnings) but red lines must necessarily be attended to. Many a times, fixing one error gets rid of many other warning lines too.

- **On existing M-files:** You can open an existing M-file in the editor and see M-Lint's messages just the way you would on a new M-file. Alternatively, you can run M-Lint on the whole directory and produce reports for each M-file in the directory with a single click-go to the Current Directory pane, click on the Action icon (the little gear icon in the menu bar of the pane) , select Reports->M- Lint Code Check Report from the pop-up menu. You are presented with the M-Lint report for all M-files.

(ii)Nested Functions

Nested functions are functions written inside a main function, just like sub functions but with the following important distinctions:
Each nested function must be terminated by an end statement. For example:

```
function [x, y] = main_fun(t, a, b)
:
      function x = nested_fun1(a,b)
      :
      end
      function y = nested_fun2(t)
      :
      end
end
```

Here, nested _fun l and nested_fun2 are nested functions inside the main function main_fun. All nested functions share the workspace of functions in which they are nested. Thus nested_fun1 and main_fun share their workspace variables and so do nested_fun2 and main_fun, but nest ed_fun1 and nested_fun2 do not share workspace variables. This facility of sharing workspace makes it easy for the nested functions to access each other's variables and their values without any explicit declaration (e.g., global) or passing them in the input list.
Functions can be nested to any level; that is, nested functions can also have their own nested functions. Of course, nested functions are not visible or accessible from outside the main function. They can, however, be made accessible from outside by creating their explicit function handles

(iii) For loops and while loops

For loops

A `for` loop is used to repeat a statement or a group of statements for a fixed number of times. Here are two examples:

Example 1:
```
for m=1:100
    num = 1/(m+1)
end
```
Example 2:
```
for n=100:-2:0, k = 1/(exp(n)), end
```

The *counter* in the loop can also be given explicit increment: `for i=m:k:n` to advance the counter i by k each time (in the second example, n goes from 100 to 0 as 100, 98, 96, ..., etc.). You can have nested `for` loops, that is, `for` loops within `for` loops. *Every* `for`*, however, must be matched with an* `end`.

While loops

A `while` loop is used to execute a statement or a group of statements for an indefinite number of times until the condition specified by `while` is no longer satisfied. For example:

```
% let us find all powers of 2 below 10000
v = 1;   num = 1;  i=1;
while num < 10000
      v = [v; num];
      i = i + 1;
      num = 2^i;
end
v     % display v
```

Once again, a `while` must have a matching `end`.