|  |  |  |
| --- | --- | --- |
| **MATLAB** | FOR | **ENGINEERING STUDENTS - 1** |

A QUICK INTRODUCTION

|  |
| --- |
| Sunil Anandatheertha |
| PES University |

**Contents**

|  |  |
| --- | --- |
| **Sl. No.** | **Topic** |
| **1** | **Introduction** |
|  | Structure of this book; Roadmaps presented in this book; |
| **2** | **Simple arithmetic and relational operations** |
|  | Command window; =, +, workspace; -, \*, /, \, ^, sqrt, exp, logarithmic functions, modulo operation, trigonometric and inverse trigonometric functions, NaN, tips for handling complicated arithmetic operations; Accessing help in MATLAB; strings and string operations: strcat, ischar |
| **3** | **Matrices and array operations** |
|  | Creating matrices: manual, :, linspace, logspace; indexing methods: row and column indices and element number; size and numel commands; ones matrix; zeros matrix; identity matrix; transpose, determinant and inverse of a matrix; eigen values of a matrix; sparse matrices: conversion from general matrix form to sparse forms and vice-versa; matrix multiplication; element wise operations; |
| **4** | **Writing and executing script files and function files** |
|  | Script files – creation, execution and shortcomings, file naming conventions; Function files: format, function file naming conventions, function calling, feval command, dos and don’ts; |
| **5** | **More about creating and using your own functions** |
|  | Variable number of arguments; Inline functions; Anonymous functions |
| **6** | **For loops and while loops in MATLAB** |
|  | Format, examples, multiple for loops, break command, matrix initialization, tic-toc; while loop, break command, examples |
| **7** | **Conditional statements in MATLAB** |
|  | If-elseif-else-end and switch-case-otherwise-end |
| **8** | **MATLAB 2 and 3 dimensional graphics** |
|  | 2D plotting: Simple plot command; Plotting multiple curves using hold command; types of lines and markers and their size and associated coloring, Plot axis formatting – axis names, title, legends and their location, controlling font style and size, axis scaling commands – equal, tight and square; 2D plotting (Contd.): line command, histogram plots, pie plots, polar plots, logarithmic plots; 3D plotting: plot3, mesh, colorbar, meshz, surf; contour, contour |
| **9** | **Polynomials and curve fitting** |
|  | poly2sym, sym2poly, roots, poly, Conv, deconv, polyval, polyfit |
| **10** | **Solving system of linear equations** |
|  | Symbolic solution, Numerical solution, Graphical solution |
| **11** | **Symbolic integration and differentiation** |
|  | Syms, Int, diff, Pretty, double, simplify |
| **12** | **Symbolic and numerical solution of simple 1st order differential equations** |
|  | Symbolic solution, Numerical solution, comparison of solutions |
| **13** | **Simple input and output operations** |
|  | input command, output commands: fprintf, disp, sprint |

**CHAPTER 2**

**Simple arithmetic and relational operations**

A value is assigned to a variable using the Assignment Operator (which is the ‘equals to’ symbol, ‘=’). The value being assigned to the variable might be either directly entered as a numerical value or evaluated using an expression written in-terms of pre-assigned variables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Look at this MATLAB code segment | | | | |
| Line  No. | Input commands entered in command window | | Line No. | Displayed output in MATLAB command window |
| 1 | a = 10.1; | | 1 | - |
| 2 | b = pi; | *Note*: ‘pi’ is a MATLAB inbuilt. | 2 | - |
| 3 | c = 23.2 | | 3 | c = 23.2 |
| 4 | d = a + b | | 4 | d = 13.2416 |
| 5 | A | | 5 | 10.1 |
| 6 | a = sqrt(4) | | 6 | a = 2 |
| 7 | e = linspace(0,10,5) | | 7 | 0.0 2.5 5.0 7.5 10.0 |

**Line 1**: *a* is a variable being assigned as 10.1. Note the ‘colon’ (that is, ‘;’) at the termination of this line of command. A colon is used to hide the display of the variable value. If it is missing, the variable value will be displayed. Since, it is present in this case however, no display is produced and hence the output for this input is not displayed. Since, it is a direct numerical value assignment and moreover, *a* does not depend on any other variable, *a* is an independent variable.

**Line 2**: The value of pi ( = 22 / 7) is being assigned to the variable *b*.

**Line 3**: Since the ‘;’ is missing at the end of this command line, the numerical value being assigned to *c* will be displayed in the command window. Hence the output is present.

**Line 1 – 3**: In all these lines, the numerical value has been directly assigned. This type of variable assignment is done only in the case of independent variables.

**Line 4**: Here, the value of summation of two *pre-assigned variables* *a* and *b* is to be assigned to a different variable called *d*. This makes *d*, a dependent variable, in which case, it is dependent on the variables *a* and *b*. Since the colon is missing, the result is displayed. It is to be observed, this type of variable assignment happens only in the case of dependent variables.

**Line 5**: Here, it is required to display the value stored in the variable *a*.

**Line 6:** Here, the variable *a* is ‘*re-assigned*’ with the value of an evaluated expression (which is sqrt(4). NOTE: ‘sqrt’ stands for square root). The previous value of *a* ( = 10.1) is now rewritten with the value of the evaluated numerical expression. Thus, *a* now takes, and has the numerical value 2.

**Exercise problems**

|  |  |
| --- | --- |
| Evaluate | Evaluate at x = 12 |
| Evaluate | Evaluate |
| Convert 30 degrees to radians | Evaluate . All angles are in degrees. Solve with and without considering the sind(..) and cosd(..) commands |
| Calculate by asking user inputs for in radians |
| Calculate the volume of a hollow cylinder of length 20mm, inner radius 10mm and outer radius 15 mm. | Evaluate Where, x = 1 and y = 2 |
| Verify whether the trigonometric identity are correct, at | Evaluate: a = 10, b = 12, c = 14, d = 16 |

**CHAPTER 2**

**Matrices and array operations**

**A = [1 3 5].**Here, a row vector is created

A = 1 3 5

**A = [1; 3; 5]** Here, the elements are directly entered in a column array

A =

1

3

5

**A = [1 3 5]’** Here, elements are first entered into a row matrix which is later transposed to arrive at the required column matrix

A =

1

3

5

**B = [1 2 3; 4 2 3; 9 7 2; 0 8 0]** Here, the elements [1 2 3], [4 2 3], [9 7 2] and [0 8 0] form the 1st, 2nd, 3rd, and 4th row entries respectively, each having 3 columns

B =

1 2 3

4 2 3

9 7 2

0 8 0

**A = 1:1.5:4** Here, 1 and 4 are the first and last values of the required array and 1.5 is the increment

A =

1.0000 2.5000 4.0000

**A = linspace(1, 4, 5)** Here, 1 and 4 are the first and last values of the required array and 5 indicates the number of elements to create between 1 and 4 (with 1 and 4 included)

A =

1.0000 1.7500 2.5000 3.2500 4.0000

**A = zeros(2,3)** This creates,

A =

0 0 0

0 0 0

**A = ones (1,4); B = ones(3,2)** This creates:

A = [1 1 1 1]

B = [1 1

1 1

1 1]

**A = eye (4,4), B = eye (3,4), C = eye (2,4), D = eye (1,4), E = eye (4,3), F = eye (4,2) and G = eye (4,1)** This creates a matrix/array with unit principle diagonal element elements

A = [ 1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1 ]

D = [ 1 0 0 0 ]

G = [ 1

0

0

0 ]

B = [ 1 0 0 0

0 1 0 0

0 0 1 0 ]

E = [ 1 0 0

0 1 0

0 0 1

0 0 0 ]

C = [ 1 0 0 0

0 1 0 0 ]

F = [ 1 0

0 1

0 0

0 0 ]

**A = rand(2,2)** This might create:

A =

0.8147 0.1270

0.9058 0.9134

**Exercise problems:**

**NOTE: [1].** Exercises 1, 3, 4 and 5 are to be done in the lab. **[2]**. Exercise 2 is to be done outside of the lab and should be written in record book, failing which marks will be deducted

|  |  |
| --- | --- |
| **1. Create arrays (linear spacing only) using appropriate commands for the following data:**  -**A**. initial = +1, increment = +2, final = +7  -**B**. initial = -2, increment = +2, final = +6  -**C**. initial = 0, final = +1, number of in-between values = +4  -**D**. a row vector containing +1, +10 and -3 in the right order | **2. Answer the following questions in record book**  **-A**. How do you access the 3rd row and 4th column of an array of the name H?  **-B**. Write command to calculate the size of an array  **-C**. Write command to calculate the number of elements in an array  **-D**. Refer to “help command name” method and write down the introductory help statements, the format and one example of the following commands (**D1**) ones (**D2**) rand (**D3**) repmat (**D4**) reshape  (**E**) How do you extract 2nd and 3rd rows of an array named H, at once?  (**F**) How do you extract 2nd and 3rd columns of an array named H, at once?  (**G**) How do you extract 2nd and 3rd rows and 4th and 5th columns of an array named H, at once?  (**H**) What is the default number of elements created between two number by linspace and logspace commands if the number of values has not been specified in the command. |
| **3.** If A = [-2 -7 9 0.456 0 56], then sort A in ascending order, assign it to B, and sort A in descending order, assign it to C |
| **4. Make an array A = [4 3 2; 1 0 9; 4 5 6]; and X = rand(3,3); having 3 rows and three columns.** Assign the variable:  [**i**] **B** with element in 2nd row and 2nd column position in A  [**ii**] **C** with all elements in 3rd row of A  [**iii**] **D** with 2nd and 3rd rows and 1st and 2nd columns of A  [**iv**] **E** with all elements in the 2nd column of **D**  [**v**] **F** with the transpose of E  [**vi**] **G** with the column matrix obtained by reshaping matrix **D** using the reshape command  [**vii**] **H** with 2A + 2X  [**viii**] **I** with A2  [**ix**] **J** with A2 with element wise squaring  [**x**] **L** with AX with element wise multiplication  [**xi**] **M** with -½ AX.A + A^2 + A.^2  [**xii**] N with transpose of L |
| **5**. Create an array having 3 linearly spaced values between 2 and 5. Assign it to **A**. Repeat it twice along row direction, twice along column direction and assign it to **B**. |
| **6**. Create logarithmically spaced array between 1 and 3 having 5 values. |

**CHAPTER 3**

**Writing and executing script files and function files**

|  |  |
| --- | --- |
| 1 | **function**  .  End |
| 2 | **function** [OUT1, OUT2, …, OUTn] = **FunctionName(**IN1, IN2, …, INm**)**  .  end |
| 3 | FuncName = **inline(**‘function definition’**)** |

A MATLAB script is basically a collection of commands arranged in a logical execution order. Even a simple command as “a = sqrt(4)” can form a script. The inputs of a script can be defined both inside and outside of the script.

A MATLAB function file on the other hand is much more specific than a simple script file. **A MATLAB function file is defined using a name called function name, a set of inputs and a set of outputs**. A variable assigned inside a function will be available only to this function and not to any other functions or scripts unless otherwise that the variable is defined as a global variable or that the variable is included in the function output.

A function file starts with the line:

**[Output 1, Output 2, … ] = Function\_Name(Input1, Input2, …)**

This is followed by comments, which give information about the author, function, version of the function, etc.

# NOTE: All the input variables which are passed on to this function must and should be defined prior to the use of this function. There should be no output variables that are not undefined

A MATLAB script file is .m file containing a collection of MATLAB commands, comments, etc, which execute in a logical sequence of order.

**Creating new script files**

To create a new script file, press “**Ctrl + n**” OR To create a new script file, use **File > New**

**Rules to be followed while creating script files**

1. *The filename constraints:* 
   1. **They should start with a letter, contain only letters, numbers or underscore characters.**

That is,

* + 1. Filename should not have any spaces in between
    2. Filename can contain both lower case and upper case alphabets
    3. Filename cannot start with a number or an underscore character
    4. Filename cannot contain special characters other than underscore
  1. **The filename should not be same as any of the reserved words in MATLAB**

1. *Tips for creating good script file names:*
   1. For the sake of this laboratory, please save the file under the following filename:

**Exp\_NUMBER\_USN\_PROB\_NUMBER**.m

**Example:** Write a MATLAB script file and a function calculate the transpose of a matrix

USING SCRIPT FILE:

Open a new m file

Write as follows:

|  |
| --- |
| A = input(‘Please enter the matrix’)  B = A’ |

Then run this file by pressing the function key “F5”. You can also do it by using the run command available in the drop down menus.

USING FUNCTION FILE:

Open a new m file

Write as follows:

|  |
| --- |
| function [B] = CalculateTranspose(A)  B = A’;  End |

Then go to command line. Execute the following command:

A = input(‘Please enter the matrix’);

Enter the required matrix as [1 2 3; 4 5 6; 6 7 8]

Then press enter.

Then call the function “CalculateTranspose” by issuing the executing the following command in the command line:

[B] = CalculateTranspose(A);

This gives the required output.

**Exercise problems:**

**NOTE: All problems to be solved in class.**

1. Create a script file to convert degrees to radians. Write a function file to do the same operation.
2. Create a MATLAB script file and function file to calculate the deflection of a cantilever beam. Note 1: Deflection at the end of a cantilever beam loaded by a point load at the end is given by the equation , where . You can use the “input” command to procure user inputs for P, L, E and I. Additionally if you are using functions, write a separate function file to calculate the moment of inertia of the beam. Note 3: E = 200 GPa, L = 1 m, Section: Rectangular, Height = 25mm, Thickness = 10mm. Note 3: Format of input command is as follows: *P = input(‘Please enter the load in Newton’)*
3. Create a MATLAB function file to calculate the deflection of a simply supported beam. Note 1: Deflection at the mid of a cantilever beam loaded by a point load at the center is given by the equation , where . Consider load acting from 0 kN to 2kN in steps of 500 N. Note 1: E = 200 GPa, L = 1 m, Section: Rectangular, Height = 25mm, Thickness = 10 mm.

**CHAPTER 4**

**For loops and while loops in MATLAB**

**Format:**

|  |  |
| --- | --- |
| for ITERATION\_DEFINITION  Execute these  end | while ITERATION\_DEFINITION  Execute these  end |

**For loop**

‘For’ loop allows the repetition of a set of commands over and over again as long as we want it. In order for MATLAB to correctly execute the ‘for’ loop completely, it has to know when to start and when to end. A vector in the “for” statement can be provided, which will be looped over by MATLAB for each value in the vector starting from the beginning till the end of the vector.

Lets say that, we need to increment a number, say ‘a’ by 2, 20 times, so that we end up with an array of values. This is explained in the following code:

**Example**:

|  |  |
| --- | --- |
| Line 1 | a = zeros(1, 20); |
| Line 2 | a(1, 1) = 2; |
| Line 3 | for i = 2 : size(a, 2) |
| Line 4 | a(1, I) = a(1, i-1) + 2; |
| Line 5 | end |

**Line 1** is simply pre-assigning the variable a with an array of the required size, but containing zeros. These zeroes will be replaced by correct numbers from within the for loop.

**Line 2** is defining the starting number and hence it occupies the first element location in a.

**Line 3** marks the beginning of **for** loop. i is the iteration variable which takes values from 2 to size(a,2). ‘size(a, 2)’ is a in-built command to know the number of columns (Cn) in a (similarly, ‘size(a, 1)’ is a in-built command to know the number of rows (Rn) in a. ‘size(a)’ command gives the answer Rn x Cn.)

**Line 4** takes the previous entry (that is a(1, i-1)), adds 2 to it and updates the location a(1, i).

**Line 5** marks the end of this for-loop.

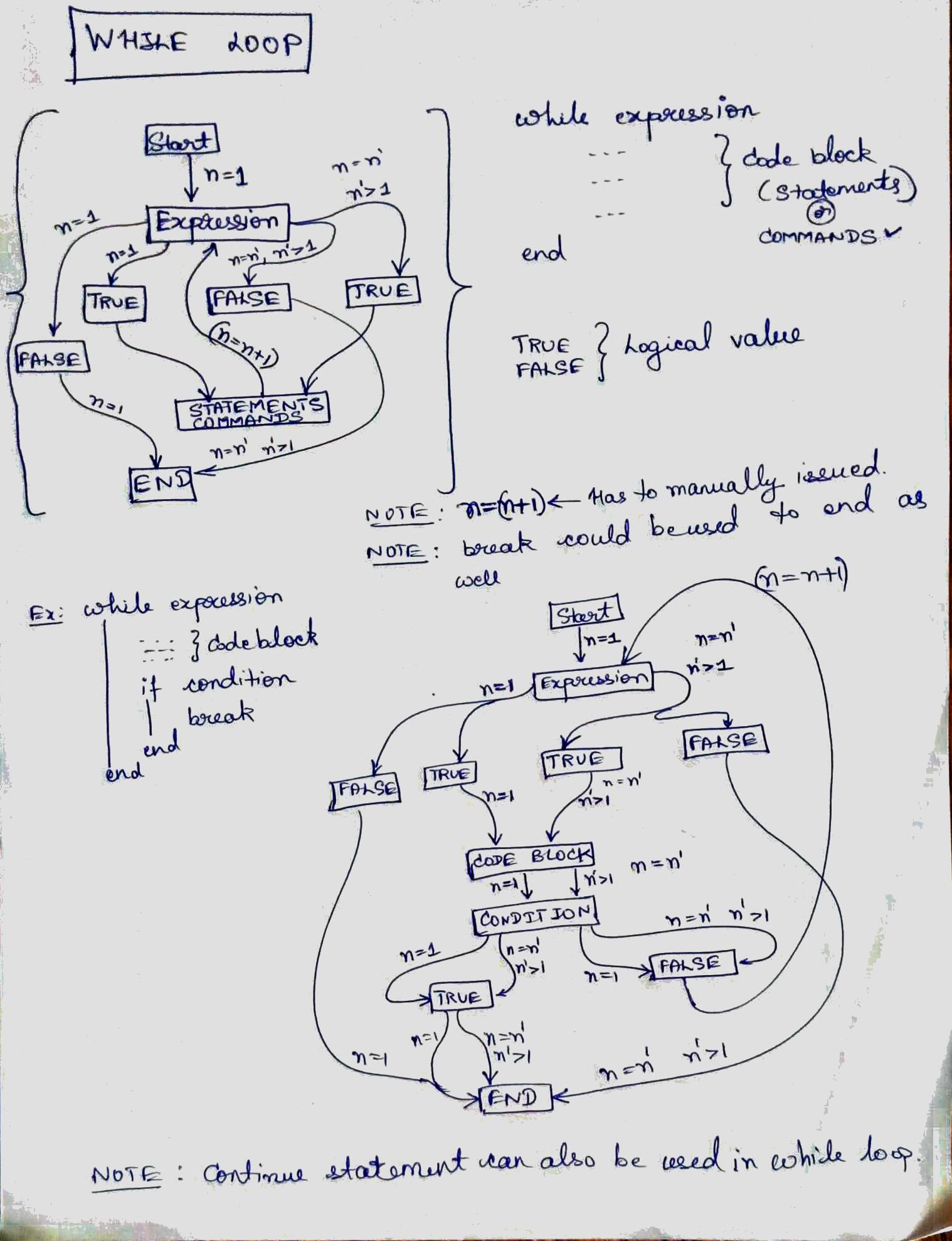
Line 4 is repeated for as many number of times until the iteration variable ‘*i*’ becomes equal to size(a, 2).

Many for – loops can be nested within each other as per the need. A for loop can be forced to stop by issuing a break command.

|  |  |  |
| --- | --- | --- |
| **for j=1:4,**  **j**  **end**  **>> v = [1:3:10]**  **v =**  **1 4 7 10** | **for j=1:4,**  **v(j) = j;**  **end**  **>> v**  **v =**  **1 2 3 4** | Here is an example of Gaussian elimination using for loops:  B =  1 3 2  2 2 1  3 1 3  **for** j = 2 : 3  **for** I = 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 |
| The following example numerically solves the differential equation , y(0) = 1, using Euler’s Method. First define step size h. Then solve. The solution is a vector, y, in which the entry y(j) is the approximation at x(j)  h = 0.1; % Define step size  x = [0 : h : 2]; % Define domain  y = 0 **\*** x; % initialize solution vector  y(1) = 1; % Define condition  **for** I = 2 : numel(x)  y(i) = y(i**-**1) **+** h**\***(x(i**-**1)^2 **-** y(i**-**1)^2);  % That is,  **end**  plot(x,y)  plot(x,y,'go')  plot(x,y,'go',x,y) | | |

# While loop

This loop repeats until some condition is satisfied and hence does not require any specification of the number of iterations which are to be performed. They are very helpful in problems where we need to stop calculations until the desired accuracy is achieved.



**Example**:

|  |  |
| --- | --- |
| Line 1 | nfactorial(1, 1) = 1; |
| Line 2 | nfactorial(1, 2) = 2; |
| Line 3 | n = 3; |
| Line 4 | while n <= 10 |
| Line 5 | nfactorial(1, n) = nfactorial(1, n-1) \* n |
| Line 6 | n = n + 1; |
| Line 7 | end |

**Line 1** defines the first element of the array ‘nfactorial’ as 1.

**Line 2** defines the second element of the array as 2 factorial = 2. Notice that it is at this step that the array becomes a row array.

**Line 3** defines an increment or count operator which is basically used to count how many iterations are performed.

**Line 4** This marks the beginning of while statement. It is accompanied by a condition which tells MATLAB when not to stop. Here the while loop should not stop at any condition which says that the iteration count is less than or equal to 10. That is, starting from n = 4 and up until 10 is reached, the while loop executes. When the condition is such that n becomes 11, then it violates n<=10 and hence, the while loop quits on its own.

**Line 5** defines the actual operation to be performed. At the instant when the while loop is entered, n is 3. When it sees nfactorial(1,n), it creates a space for another entry into the nfactorial array, and puts the value of nfactorial(1, n-1) \* n, which is basically the factorial operation (that is “n factorial = n(n-1)(n-2)..”)

**Line 6** updates the count number by unity which is required to keep the while loop going.

**Lines 5** and **6** together constitute the body of the while loop.

**Line 7** marks the end of while loop.

Exercise problems:

1. Create an M-by-N array of random numbers (use **rand**). Move through the array, element by element, and set any value that is less than 0.2 to 0 and any value that is greater than (or equal to) 0.2 to 1.
2. Axial tensile load applied on a steel bar of Young’s modulus 210 GPa, 10 mm diameter and 80 mm varies from 0.5 kN to 2 kN in steps of 0.25 kN. Use “for” loop and calculate tensile stress developed in the bar, deformation underwent by the member and strain developed in the member for different load values.
3. An axial tensile load starting from 0.5 kN is increasing applied on a steel bar of Young’s modulus 210 GPa, 10 mm diameter and 80 mm in steps of 0.25 kN until the strain developed in the member is 1 % of the member’s total length. Use “while” loop and calculate tensile stress developed in the bar, deformation underwent by the member and strain developed in the member for different load values till the last load.
4. Write a for loop and while loop to evaluate the factorial of a number greater than 0
5. Evaluate the following sum using for loop and while loop

**CHAPTER 5**

**Conditional statements in MATLAB**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | **if** **CONDITION**  EXECUTE THESE **end** | **2** | **if** **CONDITION**  EXECUTE THESE  **else**  EXECUTE THESE  **end** | | **3** | **if** **CONDITION1**  EXECUTE THESE  **elseif CONDITION2**  EXECUTE THESE  **end** |
| **4** | **if** **CONDITION1**  EXECUTE THESE  **elseif CONDITION2**  EXECUTE THESE  **else**  EXECUTE THESE  **end** | **5** | **switch** **METHOD\_STRING**  **case** **METHOD1**  EXECUTE THESE  **case** **METHOD2**  EXECUTE THESE  **case** **METHOD3**  EXECUTE THESE  **end** | | 6 | **if** **CONDITION1a**  **if** **CONDITION2a**  EXECUTE THESE  **elseif** **CONDITION2b**  EXECUTE THESE  **end**  **elseif CONDITION1b**  EXECUTE THESE  **else**  EXECUTE THESE  **end** |
| 7 | **switch** **METHOD\_STRING**  **case** **METHOD1**  **if** **CONDITION**  EXECUTE THESE  **end**  **case** **METHOD2**  EXECUTE THESE  **case** **METHOD3**  **if** **CONDITION1**  EXECUTE THESE  **elseif CONDITION2**  EXECUTE THESE  **else**  EXECUTE THESE  **end**  **end** | | |  | | |

Conditional statements are basically used to decide what to do upon having choices to make and also when the conditions needed to make the right choice is also available. We shall study two conditional statements, which are ‘if-elseif-else-end’ and ‘switch-case-end’ statement.

**if-elseif-else-end**

Say for example, if the choices that you have are to study and to not to study and the conditions that are to be met are that you need to progress and you need not progress, then the conditional statement takes the form:

***if*** *“you want to progress”*

*You have to study and understand.*

***elseif*** *“you don’t want to progress”*

*You need not study and understand.*

***end***

Thus, depending upon the right condition, commands are executed.

**Example**: The use of *if-end* statements: look at the following code segment:

A = 10;

if A < 8

B = 1;

end

First, variable A is assigned with a value of 10. Then, if block begins. It compares the value of A and 8. Only if found less than 8, then the command B = 1 is executed. If it finds that A < 8 is false, the command B = 1 is ignored and proceeds to next control statement, in this case, it is ‘end’ and the ‘if-end’ ends.

**Example**: The use of *if-else-end* statements: look at the following code segment:

A = 10;

if A < 8

B = 1;

else

B = 0;

end

First, A is assigned to 10. The first condition, A < 8 is not met, so MATLAB proceeds further to the next control statement which is ‘else’. The moment it sees this, it executes whatever command is given between this control statement and the end statement. Hence, the command B = 0 is executed.

If in case, A were assigned with a value of 5, then the command B = -1 would get executed.

**Example**: The use of *if-elseif-else-end* statements: look at the following code segment:

A = 10;

if A < 8

B = 1;

elseif A > 8 || A <=10

B = 0;

elseif A == 11 && A ==13

B = -1;

else

B = -2;

end

First, A is assigned to 10. The first condition, A < 8 is not met, so MATLAB proceeds further to the next control statement which is ‘elseif’. The condition to be met is written as “A > 8 && A <=10 ”. Here, the ‘||’ signifies ‘or’ operation. This condition has two sub-conditions A > 8 and A <= 10. Since, we use ||, any one of these commands needs to be satisfied. In this case, since the value of A is 10, the second sub-condition satisfies. The moment one of the condition is satisfied, all the other control statements are ignored and after executing the command B = -1, MATLAB directly jumps to end and ends the execution of this code segment.

**switch-case-end**

In some cases, given that you have something to be done, which could be anything like make

many options and a lot of sorting of tasks to be done according to the available options, and that the task to be done , then the switch case command comes very much handy.

Name\_of\_the\_option1 = 'Coffee';

Name\_of\_the\_option2 = 'AssignMEnt';

Name\_of\_the\_option3 = 'morning\_6am';

NOTE: When something is enclosed within single inverted comma, it means it is not a numerical value but a string.

**For example**: 456 is a number while ‘456’ is a string (that is a set of characters. Here, 4 5 and 6 are all treated as alphabets instead of numbers).

String assignments are case sensitive on their own (provided that no extra commands are provided to convert the string case to what we need).

**For example**: 'AssignMEnt' is different from 'assignment'.

Commands such as **upper()** and **lower()** are used to convert all the characters in a strings to upper case and lower case respectively.

**For example**: the command upper(‘MATlab’) returns ‘MATLAB’ while the command

lower (MATlab) returns ‘matlab’.

|  |  |  |
| --- | --- | --- |
| Name\_of\_the\_option1 = 'Coffee';  Name\_of\_the\_option2 = 'AssignMEnt';  Name\_of\_the\_option3 = 'morning\_6am'; | | |
| **switch** lower(Name\_of\_the\_option1)  **case** {'coffee'}  disp('Do you want coffee')  **case** 'idly'  disp('Only idly ?. What else do you need ?…')  **case** 'chatney'  disp('don’t want idly?')  **otherwise**  disp(' I don’t get it. What is it that you want ')  **end** | **switch** lower(Name\_of\_the\_option2)  **case** {'AssignMEnt'}  disp('Did you forget do do ?’)  **case** 'Test'  disp( ‘Study well’)  **case** 'chatney'  disp('don’t want idly?')  **otherwise**  disp(‘All is well')  **end** | **switch** Name\_of\_the\_option3  **case** {'morning\_6AM', ‘0600hrs’}  disp(‘Waky wakyyy. Breakfast ready’)  **case** '1000PM'  disp( ‘No more movies. Go to sleep’)  **case** '0230pm'  disp('don’t want idly?')  **otherwise**  disp(‘All is well')  **end** |
| The lower command next to switch is provided in order to improve the flexibility of the program. It comes in handy when Name\_of\_the\_option1 is entered as ‘COFfee’ instead of ‘coffee’. But this will not work if the ‘coffee’ written next to case statement is written as ‘COFFEE’ !. Good care has to be taken to avoid such mistakes. | This works in the same way as the previous switch case statement. You can easily figure it out on your own. Use curly brackets {} to enclose include case checks in a single case statement, and nevertheless, there is no harm in using it for a single case check also, as is done in this case. | Nothing will display here. It is because, the variable, Name\_of\_the\_option3 is set to 'morning\_6am'. But it does not match with any of the available cases. Hence, this switch case statement exits. |

**Example**: The use of *switch-case-end* statements:

Consider the following code segment.

MethodToUse = ‘ColonOperator’;

**switch** lower(MethodToUse)

**case** ‘colonoperator’

disp(‘Using the colon operator to make a linear array’)

A = 0:2:10;

**case** ‘linspace’

disp(‘Using linspace function to make a linear array’)

A = linspace(0,10,2);

**end**

The MethodToUse is set to ‘ColonOperator’ in the beginning. The *switch case* statement begins from the next line. The method to be compared against the different cases is first converted to lower case using lower() command at the end of which the ‘ColonOperator’ becomes ‘colonoperator’. Next, different cases are presented. In the first case, this lower(MethodToUse) (whos value is ‘colonoperator’) is compared against ‘colonoperator’ next to the first case statement. MATLAB finds that there is a match. The moment it finds a match, it ignores all other case statements and executes all the commands between itself and the next control statement. Thus, the commands, disp(‘Using the colon operator to make a linear array’) and A = 1:2:10; are executed.

If in case, MethodToUse = ‘linspace’, then the second case-statement would satisfy and the commands disp(‘Using linspace function to make a linear array’) and A = linspace(0,10,2) would get executed.

If you were to use upper() instead of lower(), the above code would be:

MethodToUse = ‘ColonOperator’;

**switch** upper(MethodToUse)

**case** ‘COLONOPERATOR’

disp(‘Using the colon operator to make a linear array’)

A = 1:2:10;

**case** ‘LINSPACE’

disp(‘Using linspace function to make a linear array’)

A = 1:2:10;

**end**

Notice the difference between the two code segments.

**Exercise problems**

1. Write a simple MATLAB function to calculate the determinant of a 3x3 matrix. Your function should be able reject non-square matrices
2. Generate a random column matrix having 10 elements. Use for loop to loop through all the elements in this matrix. Replace the element in the matrix with 100 if it is less than 0.5 and with 200 if it is greater than or equal to 0.5.
3. Generate a random column matrix having 10 elements. Use for loop to loop through all the elements in this matrix. Replace the element in the matrix with 100 if it is greater than 0.0 and less than 0.33; with 200 if it is greater than or equal to 0.33 and less than 0.67; and with 300 if greater than or equal to 0.67 and lesser than 1.0.
4. Write a simple MATLAB function to calculate the deflection of a beam. Use switch case statement inside the function file to calculate deflection to cantilever beam and simply supported beam separately. The inputs for the functions should be in the order as follows:

[deflection] = functionname(beamtype, thickness, height, PointLoadInNewtons, LoadLocationX)

**CHAPTER 6**

**MATLAB 2 and 3 dimensional graphics**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | plot(X, Y) | | | |
|  | plot(X, Y, '-k', 'LineWidth', 2) | | | |
|  | plot3(X, Y, Z, 'ro', 'MarkerFaceColor', 'r', 'MarkerSize', 2) | | | |
| 2 | [X, Y] = meshgrid(x,y); Where x and y are one dimensional arrays, i.e, row or column array | | | |
| 3 | mesh(X,Y,Z) | meshz(X,Y,Z) | | |
| 4 | surf(X,Y,Z) | | | |
| 5 | contour(X,Y,Z) | | contourf(X,Y,Z) | |
| 6 | xlabel('X-axis') | | ylabel('Y-axis') | zlabel('Z-axis') |
| 7 | title('Y = X^2') | | | |
| 8 | legend('Legend entry 1', 'Legend entry 2',............., 'Legend entry n')  legend('Legend entry 1', 'Legend entry 2', ............., 'Legend entry n', 'Location', 'NorthWest')  legend('Legend entry 1', 'Legend entry 2', ............., 'Legend entry n','Location', 'NorthEast')  legend('Legend entry 1', 'Legend entry 2', ............., 'Legend entry n','Location', 'SouthWest')  legend('Legend entry 1', 'Legend entry 2', ............., 'Legend entry n','Location', 'SouthEast')  legend('Legend entry 1', 'Legend entry 2', ............., 'Legend entry n','Location', 'Best') | | | |

# MATLAB has good visualization tools, which come very handy to an engineer and a scientist. Data visualization provides a very quick and yet broad insight into the data at hand, especially to a mechanical engineer. Suppose you have some data and would like to visualize it, then you can make use of a variety of tools available in MATLAB. The tools that we shall be using are:

|  |  |  |
| --- | --- | --- |
| Example 1 | | |
| Method 1: OK | Method 2: GOOD | Method 3: BETTER (**Not there in syllabus**) |
| LINE 01: X = 1:100;  LINE 02: Y = X.^2;  LINE 03: **plot(X, Y)** | LINE 01: X = 1:100;  LINE 02: Y = X.^2;  LINE 03: **plot(X, Y, '-k', 'LineWidth', 2)**  LINE 04: axis square  LINE 05: axis tight  LINE 06: **xlabel**('X-axis')  LINE 07: **ylabel**('Y-axis')  LINE 08: **title**('Y = X^2')  LINE 09: **legend('Legend entry 1', 'Location', 'NorthWest')** | LINE 01: X = 1:100;  LINE 02: Y = X.^2;  LINE 03: plot(X, Y, '--k', 'LineWidth', 2)  LINE 04: axis square  LINE 05: axis tight  LINE 06: hxl = xlabel('X-axis');  LINE 07: hyl = ylabel('Y-axis');  LINE 08: ht1 = title('Y = X^2');  LINE 09: hleg1 = legend('Legend entry 1', 'Location', 'NorthWest');  LINE 10: set([hxl hyl ht1], 'fontsize', 12)  LINE 11: set(hleg1, 'box', 'off') |
| Explanation:  *Line 01 and 02* assigns the variables X and Y with linear arrays.*Line 03* plots the two arrays. The 1st argument in the command always corresponds to x-axis, and the 2nd to y-axis. | Explanation:  In Line 03, the third argument in the plot command is ‘-k’, and it means continuous black line. The 4th and 5th arguments, 'LineWidth' and 2 tells that the width of line should 2.  Line 04 and 05 formats the figure properly. Line 06, 07, 08 and 09 sets the xlabel, ylabel, figure title and legend entries. | Explanation:  In this, more control is exercised on the plot. In Line 03, ‘--k’ is used instead (as opposed to ‘-k’ in method 2) which means, dashed black line. The handles for xlabel, ylabel, title are used to change their font-sizes and provided in Line 10. The legend handle is used to remove the box around it, and the command to do this is provided in Line 11. |
| Resulting plot  GraphicsPlot001 | Resulting plot  GraphicsPlot002 | Resulting plot  GraphicsPlot003 |

**Two-dimensional contour plot:**

|  |  |
| --- | --- |
| Example 2 | |
| **LINE 01:** xmin = 01;  **LINE 02:** xmax = 25;  **LINE 03:** ymin = 01;  **LINE 04:** ymax = 20;  **LINE 05:** [X, Y] = **meshgrid**(xmin:xmax,ymin:ymax);  **LINE 06:** Z = sin(X/(xmax-xmin))+cos(Y/(ymax-ymin));  **LINE 07:** figure(1)  **LINE 08:** **contour(X, Y, Z)**  **LINE 09:** figure(2)  **LINE 10:** **contourf(X, Y, Z)**  **LINE 11:** colormap('cool')  **LINE 12:** figure(3)  **LINE 13:** **contourf(X, Y, Z, 20)**  **LINE 14:** colormap('copper') | Figure 1  GraphicsContour001  You can hardly see the contours here! |
| Figure 2  GraphicsContour002 | Figure 3  GraphicsContour003 |
| LINE 01 - 04: Define the x and y coordinates  LINE 05: Make a mesh/grid out of x and y coordinates  LINE 06: Find Z at every X and Y in the grid of points  LINE 07: Start a new figure window with the name '1'  LINE 08: Make a contour plot having only contour lines  LINE 09: Start a new figure window with the name '2'  LINE 10: Make a filled contour plot  LINE 11: Change the color map of this contour plot  LINE 12: Start a new figure window with the name '3'  LINE 13: Make a filled contour plot with 20 contours  LINE 14: Change the color map of this contour plot to 'copper' | |

# Three dimensional graphics in MATLAB

**Example for plot 3**: Plot the following function using plot3 3D data visualization in MATLAB

|  |  |
| --- | --- |
| **LINE 01:** % plot a function using plot(3) command  **LINE 02:** x = -1000:10:1000;  **LINE 03:** y = -1000:10:1000;  **LINE 04:** **[X, Y] = meshgrid(x,y)**;  **LINE 05:** Z = exp(X/max(x)).\*sind(X) + exp(Y/max(y)).\*cosd(Y);  **LINE 06:** **plot3(X, Y, Z, 'ro', 'MarkerFaceColor', 'r', 'MarkerSize', 2)**  **LINE 07:** axis square  **LINE 08:** axis tight  **LINE 09:** box on  **LINE 10:** xlabel('X-axis')  **LINE 11:** ylabel('Y-axis')  **LINE 12:** zlabel('Z-axis')  **LINE 13:** title('Z = e^{X/max(x)}sind(X) + e^{Y/max(y)}cosd(Y). mesh plot') | D:\SunilAnandatheertha\MATLAB_manual\as on 30 07 2014 - smaller folder size\MATLAB Codes\GraphicsPlot3demo001.jpeg  This figure is the 3D plot of the function:  Z = exp(X/max(x)).\*sind(X) + exp(Y/max(y)).\*cosd(Y); |
| LINE 01: Header comment  LINE 02, 03: Create a vector for x-axis and y-axis  LINE 04: Make a grid of points from the x-vectors and y-vectors  LINE 05: this is the function which defines the Z coordinate a function of x and y coordinates  LINE 06: Use plot3 and plot the function  LINE 07: Make the figure window square in shape  LINE 08: Fit everything that is visible to the size of figure window  LINE 09: Make a box around the plot  LINE 10-12: Make x, y, and z – axis labels  LINE 13: make title for the plot | |

Plot the following function using 3D data visualization in MATLAB

*NOTE*: Choose an appropriate domain in x and y.

|  |  |
| --- | --- |
| Example 3 | |
| **Line 01:** x = -1000:10:1000;  **Line 02:** y = -1000:10:1000;  **Line 03:** [X, Y] = **meshgrid**(x,y);  **Line 04:** Z = exp(X/max(x)).\*sind(X) + exp(Y/max(y)).\*cosd(Y);  **Line 05:** figure(1)  **Line 06:** **mesh**(X,Y,Z)  **Line 07:** axis square  **Line 08:** axis tight  **Line 09:** box on  **Line 10:** xlabel('X-axis')  **Line 11:** ylabel('Y-axis')  **Line 12:** zlabel('Z-axis')  **Line 13:** title('Z = e^{X/max(x)}sind(X) + e^{Y/max(y)}cosd(Y). mesh plot')  **Line 14:** print('-djpeg100',strcat(pwd,'/GraphicsMesh001.jpeg'))  **Line 15:** figure(2)  **Line 16:** **meshc**(X,Y,Z)  **Line 17:** axis square  **Line 18:** axis tight  **Line 19:** box on  **Line 20:** xlabel('X-axis')  **Line 21:** ylabel('Y-axis')  **Line 22:** zlabel('Z-axis')  **Line 23:** title('Z = e^{X/max(x)}sind(X) + e^{Y/max(y)}cosd(Y). meshc plot') | Figure 1  GraphicsMesh001 |
| Figure 2  GraphicsMesh002 |
| LINE 01 - 02: Define the x and y coordinates  LINE 03: Make a mesh/grid out of x and y coordinates  LINE 04: Find Z at every X and Y in the grid of points  LINE 05: Start a new figure window with the name '1'  LINE 06: Make a simple mesh plot of X, Y and Z  LINE 07-13: Make the necessary figure settings  LINE 14: Print the figure to jpeg file  LINE 15: Start a new figure window with the name '2'  LINE 16: Make a mesh plot with contours projected onto the x-y plane  LINE 17-23: Make the necessary figure settings  LINE 24: Print the figure to jpeg file | |

1. Graph the function in the range [0 , 360] (in degrees) using 2D plot command. Include figure title, axis labels and legends. Specify a line thickness of 2. The font sizes of labels and titles should be 14. Legend entry should not contain a box around it. No grids are needed. The plot should fit completely in the figure. Axis should be square.
2. Graph the function in the range [0, 2π] (in radians) using 2D plot command. Include figure title, axis labels and legends. Specify a line thickness of 2. The font sizes of labels and titles should be 14. Legend entry should not contain a box around it. No grids are needed. The plot should fit completely in the figure. Axis should be square.
3. Graph the function in the range [0, 1] using 2D plot command. Include figure title, axis labels and legends. Specify a line thickness of 2. The font sizes of labels and titles should be 14. Legend entry should not contain a box around it. No grids are needed. The plot should fit completely in the figure. Axis should be square.
4. Graph the function in the range of *i* in [1, 4] using 2D plot command. Include figure title, axis labels and legends. Specify a line thickness of 2. The font sizes of labels and titles should be 14. Legend entry should not contain a box around it. No grids are needed. The plot should fit completely in the figure. Axis should be square.
5. Graph the function (Use 10 divisions along x-axis and 15 divisions along y-axis) using **contour**, **contour**, **mesh** and **surf** commands. Make separate figures for each type of plots. Include the figure title and axis labels. The three dimensional plots must have a box around it. No grids are needed. The plot should fit completely in the figure. Axis should be square.
6. Graph the function (Use a range of 0 to 1000, in steps of 50) using **contour**, **contour**, **mesh** and **surf** commands. Make separate figures for each type of plots. Include the figure title and axis labels. The three dimensional plots must have a box around it. No grids are needed. The plot should fit completely in the figure. Axis should be square.
7. Graph the function (Use a range of 0.25 to 2) using **contour**, **contour**, **mesh** and **surf** commands. Make separate figures for each type of plots. Include the figure title and axis labels. The three dimensional plots must have a box around it. No grids are needed. The plot should fit completely in the figure. Axis should be square.

**CHAPTER 7**

**Polynomials and curve fitting**

**poly2sym**

|  |  |
| --- | --- |
| The format of this command is poly2sym(coefficient matrix of the polynomial). It is used to obtain the symbolic representation of a polynomial function defined in terms of its co-efficients. | Example:  >> coeff = [1 2 3 4]  >> A = poly2sym(coeff)  >> A =  x^3 + 2\*x^2 + 3\*x + 4 |

**sym2poly**

|  |  |
| --- | --- |
| The format of this command is sym2poly(symbolic representation of the polynomial). It is used to obtain the co-efficients of a symbolically represented polynomial function. | Example:  >> sym2poly(A)  >> ans =  1 2 3 4 |

**roots**

|  |  |
| --- | --- |
| **roots(**coefficient matrix of the polynomial**)**  Used to calculate the roots of a polynomial when its co-efficients are known | roots([1 6 9])  **ans =**  **-3.0000 + 0.0000i**  **-3.0000 - 0.0000i**  coeff = [1 1 1 -3**];**  roots([coeff])  **ans =**  **-1.0000 + 1.4142i**  **-1.0000 - 1.4142i**  **1.0000 + 0.0000i** |

poly

|  |  |
| --- | --- |
| **poly([root1 root2 root3])**  Used to calculate the co-efficients of the polynomial when its roots are known | P = poly([2 2])  **P =**  **1 -4 4**  Root1 = 2;  Root2 = 2;  P = poly([Root1 Root2])  **P =**  **1 -4 4** |
| The roots of x^2 - 4x +4 = 0 are +2 and +2. Suppose only the roots are known and we need to find the polynomial, then issue the command:  P = poly([2 2])  MATLAB return the p vector as: P = [ 1 -4 4 ], the elements of which are the coefficients of the polynomial | |

conv

|  |  |
| --- | --- |
| **conv(**polcoeff01, polcoeff02**)**  Used to multiply two polynomials | A = [1 2 3];  B = [3 1];  C = conv(A,B)  **C =**  **3 7 11 3** |

deconv

|  |  |
| --- | --- |
| [quotient, remainder] = **deconv(**A, B**)**  Used to divide one polynomial (i.e. B) into another polynomial (i.e. A) | A = [1 2 3];  B = [3 1];  C = conv(A,B);  [quo, rem] = deconv(C, A)  **quo =**  **3 1**  **rem =**  **0 0 0 0**  [quo, rem] = deconv(C, B)  **quo =**  **1 2 3**  **rem =**  **0 0 0 0** |

**polyval**

|  |  |  |  |
| --- | --- | --- | --- |
| **polyval(**coefficient matrix of the polynomial, number at which the polynomial has to be evaluated**)**  It is used to evaluate a polynomial for given numerical values |  | polyval(coeff, 10.243)  **ans =**  **1.1868e+03**  polyval(coeff, [3 2 4 17 9])  **ans =**  **36 11 81 5216 816** | |
| Suppose you have a polynomial y = x2 + 3x – 10 representing a certain geometric curve and you have to find out what is the value of y at some value of x (say x = 10). This can be done by using the polyval command: >> PolCoeff = [ 1 3 -10 ]; >> polyval( PolCoeff , [10]).  suppose you want to find the value at x = 10, 20 and 2, then use the command: >> y = polyval( PolCoeff , [10 20 2] ) | | |

**polyfit**

|  |  |
| --- | --- |
| FitData = **polyfit(**xdata, ydata, polynomial degree**)**  It is used to fit nth order polynomial equation to the 2-dimensional data available | x = [1 2 3 4 5 6 7 8 9 10];  y = [10 9 7 8 9 6 4 3 1 2];  sol1 = polyfit(x,y,1) % linear polynomial  sol2 = polyfit(x,y,2) % cubic polynomial  y1 = polyval(sol1, x)  y2 = polyval(sol2, x)  plot(x, y, 'ko’)% plot of given data  hold on  plot(x,y1,'-r^') % plot of fit linear polynomial  plot(x,y2,'--bs') % plot of fit cubic polynomial |
| sol1 =  -0.9879 11.3333  Sol2 =  -0.0530 -0.4045 10.1667  y1 =  Columns 1 through 7  10.3455 9.3576 8.3697 7.3818 6.3939 5.4061 4.4182  Columns 8 through 10  3.4303 2.4424 1.4545  y2 =  Columns 1 through 7  9.7091 9.1455 8.4758 7.7000 6.8182 5.8303 4.7364  Columns 8 through 10  3.5364 2.2303 0.8182 |  |

**Fit 0th, 1st, 2nd and 10th order polynomial to the below experimentally observed variation of temperature (ter) with time (T): T = [01 02 03 04 05 06 07 08 09]; tem = [03 01 04 06 10 14 20 18 30];**

SOLUTION:

**L01**: T = [01 02 03 04 05 06 07 08 09]; tem = [03 01 04 06 10 14 20 18 30];

**L02**: plot(T, tem, 'ko', 'markerfacecolor', 'g', 'markersize', 10); hold on

|  |  |  |
| --- | --- | --- |
| **%%%%% N = 0 %%%%%**  **L03**: curve\_fit\_eqn\_0 = polyfit(T, tem, 0);  **L04**: tem\_0 = polyval(curve\_fit\_eqn\_0, T);  **L05:** plot(T, tem\_0, '-ko') |  | **%%%%% N = 2 %%%%%**  **L09:** curve\_fit\_eqn\_2 = polyfit(T, tem, 2);  **L10:** tem\_2 = polyval(curve\_fit\_eqn\_2, T);  **L11:** plot(T, tem\_2, '-bx') |
| **%%%%% N = 1 %%%%%**  **L06:** curve\_fit\_eqn\_1 = polyfit(T, tem, 1);  **L07:** tem\_1 = polyval(curve\_fit\_eqn\_1, T);  **L08:** plot(T, tem\_1, '-rs') |  | **%%%%% N = 10 %%%%%**  **L12:** curve\_fit\_eqn\_10 = polyfit(T, tem, 10);  **L13:** tem\_10 = polyval(curve\_fit\_eqn\_10, T);  **L14:** plot(T, tem\_10, ':k^', 'linewidth', 2) |

**L15:** legend('t vs temp\_{exp}', 'N = 0', 'N = 1', 'N = 2', 'N = 10',…

‘location', 'northwest')

***A detailed step-step explanation of this code is provided below***:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L01 :** | **T = [01…09]; tem = [03…30];**  *Define time and temperature values in T and tem row matrices* | |  | Matrix creation methods (Chap. 3) |
| **L02 :** | **plot(T, tem, 'ko', 'markerfacecolor', 'g', 'markersize', 10); hold on**  *Plot T on horizontal-axis and tem on vertical-axis and issue a hold on command so that the plots of curve fit equations can also be superimposed on this plot; this makes comparison easy* | |  | Topic numbers AA, BB, CC and DD |
| **L03 :** | **curve\_fit\_eqn\_0 = polyfit(T, tem, 0);** | L06, L09, L12 |  | Topic number AA |
| **L04 :** | **tem\_0 = polyval(curve\_fit\_eqn\_0, T);** | L07, L10, L13 |  | Topic number AA |
| **L05 :** | **plot(T, tem\_0, '-ko')** | L08, L11, L14 |  | Topic numbers AA, BB, CC |
| **L15 :** | **legend('t vs temp\_{exp}', 'N = 0',…**  **'N = 1', 'N = 2', 'N = 10',…**  **‘location', 'northwest')** | |  | Topic numbers AA and BB |

1. Calculate the roots of the equations , and
2. The roots of a cubic equation are *Root1* = -1.0000 + 1.4142i; *Root2* = -1.0000 + 1.4142i; *Root3* = -1.0000 + 1.4142i; Generate the polynomial and express it symbolically
3. Calculate the value of the polynomial function at x = 10, using MATLAB’s inbuilt commands dealing with polynomials.
4. Evaluate for 48 values of x ranging from 0 to 5
5. Evaluate for 48 values of x ranging from 0 to 5
6. Calculate the value of the polynomial function at x = 12, by making use of MATLAB’s inbuilt commands dealing with polynomials
7. Calculate the value of function at x = 14, by making use of MATLAB’s inbuilt commands dealing with polynomials

**CHAPTER 8**

Solving system of linear equations

A linear equation is basically an equation containing linear summation of scalar multiples of independent variables. For example, if x, y and z be the independent variables, then a1 x + a2 y + a3 z = k is a linear equation, wherein k is some constant numerical value independent of x, y and z. Consider two straight lines of the form a1 x + a2 y = k1 and b1 x + b2 y = k2. The closed form solution is: and . However, when the number of equations increases, it will be difficult to obtain, remember and apply the closed form solution as done in the above example. For this, we can use any one of the three commands given above. However, the solve command cannot be used for very large systems.

**Using solve command**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| >> a = solve('2\*x+3\*y=10, 3\*x+2\*y=9')  a =  x: [1x1 sym]  y: [1x1 sym]   |  |  | | --- | --- | | >> a.x  ans =  7/5 | >> a.y  ans =  12/5 | |  | >> a = solve('2\*x+3\*y+2\*z=10, 3\*x+2\*y+3\*z=9, -3\*x+2\*y+0\*z=7')  a =  x: [1x1 sym]  y: [1x1 sym]  z: [1x1 sym]  >>[a.x a.y a.z]  ans =  [ -11/15, 12/5, 32/15] |

**Using graphical method**

|  |  |
| --- | --- |
| >> a = **solve**('2\*x+3\*y=10, 3\*x+2\*y=9')  >> H1= **ezplot**('2\*x+3\*y=10');  >> **hold on**  >> H2= **ezplot**('3\*x+2\*y=9')  >> **set**([H1 H2], 'color', 'k') |  |

Here the two lines are the graphical representation of the two lines. The point of intersection represents the solution.

**Using inv command**

AX = B

X = A-1B

|  |  |
| --- | --- |
| >> A = [2 3; 3 2]; B = [10 9]';  >> X = **inv**(A)\*B  >> X =  1.4000  2.4000 | Clearly these values are the same as a.x and a.y obtained above, which is 7/5 and 12/5 respectively. |

**Using \ (left division) operation**

[matrix-division-matlab](http://tutorial45.com/wp-content/uploads/2015/07/matrix-division-matlab6.png?88f717)

>> X = A\B

>> X =

1.4000

2.4000

**Using solve command**

|  |  |
| --- | --- |
| >> a = **solve**('2\*x+3\*y=10, 3\*x+2\*y=9')  >> H1= **ezplot**('2\*x+3\*y=10');  >> **hold on**  >> H2= **ezplot**('3\*x+2\*y=9')  >> **set**([H1 H2], 'color', 'k')  >> plot(a.x, a.y, 'ks', ‘MarkerFaceColor’, ‘r’)  >> grid on |  |

Solve the following equations using solve command, graphical method, using inv command and using the \ (left division) operation.

1. **Using solve command**

>> a = solve('2\*x+3\*y+2\*z=10, 3\*x+2\*y+3\*z=9, -3\*x+2\*y+4\*z=7');

1. **Using graphical method**

The given equations can be re-written as: , ,

>> ezmesh('-1\*x-3/2\*y+10/2'); hold on;

>> ezmesh( '-x-2/3\*y+9/3');

>> ezmesh( '+3/4\*x-2/4\*y + 7/4')

1. **Using inv command**

>> A = [2 3 2; 3 2 3; -3 2 4];

>> B = [10 9 7]';

>> X = inv(A)\*B;

1. **Using \ operation**

X1 = A\B

Let us see all the solutions together:

Solution obtained ‘graphically’ and using ‘solve’ command:

|  |  |
| --- | --- |
| >> **ezmesh**('-1\*x-3/2\*y+10/2'); hold on;  >> **ezmesh**( '-x-2/3\*y+9/3');  >> **ezmesh**( '+3/4\*x-2/4\*y + 7/4')  >> **plot3**(a.x, a.y, a.z, ‘ks’, ‘MarkerFaceColor’, ‘b’) |  |

It can be seen from the figure that the two solutions are same.

Solution obtained ‘solve’ command, using ‘inv’ command and using ‘\’ operation:

The solutions obtained using ‘solve’ command, ‘inv’ command and ‘\’ operation are respectively stored in the variables ‘a’, ‘X’ and ‘X1’ respectively.

|  |  |  |
| --- | --- | --- |
| >> [a.x a.y a.z]  ans =  [ 17/35, 12/5, 32/35] | >> X  X =  0.4857  2.4000  0.9143 | >> X1 =  0.4857  2.4000  0.9143 |

Thus it can be seen that all answers are same.

**Solve the following system of linear equations using all the methods that you have learnt in this session:**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**CHAPTER 9**

Symbolic integration and differentiation

**syms** stands for symbolic representations. **FORMAT:** syms x y a b. Here, x, y, a and b will be treated as symbols instead of variables

**int** stands for symbolic integration **FORMAT:** int(fun, IV, LL, UL). Here, fun, is the function to be integrated. It must be defined before use in **int** code. IV stands for integration variable. LL means lower limit and UL means upper limit. **EXAMPLE:** If integrating w.r.t ‘x’, and fun is y\*sin(x) then:

>> syms x y; fun = y\*sin(x); int(fun, ‘x’, 0, 1)

**diff** stands for symbolic differentiation. **FORMAT:** diff(fun, DV). Here, fun, is the function to be differentiation. It must be defined before use in **diff** code. DV stands for differentiation variable. **EXAMPLE:** If differentiating w.r.t ‘y’, and fun is y\*sin(x) then:

>> syms x y; fun = y\*sin(x); diff(fun, ‘y’)

**pretty** displays the result in a more “Human” readable format. But, not always. You can always do a trial and error to find out whether it improves the readability or makes no change or even makes it worse. This command basically prettifies the output display for symbolic computations. **EXAMPLE:**

>> syms x y

>> fun = y\*sin(x);

>> pretty(fun)

>> pretty(diff(fun)\*sin(x)/tanh(x))

**double(A)** This calculates the numerical value of the symbolic answer provided by MATLAB

**Symbolic integration in MATAB**

This method is used whenever; we need a closed form expression or the integral of a certain known function.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Integral** | **Code and answer** | | **Integral** | **Code and answer** |
| **1)** | syms x  fun = x;  int(fun)  **ANSWER**  x^2/2 | | **2)** | syms x a b  fun = x;  int(fun, a, b)  **ANSWER**  b^2/2 - a^2/2 |
| **3)** | syms x  fun = x;  int(fun, 0, 1)  **ANSWER**  b^2/2 - a^2/2 | | **4)** | syms x y a b  fun = x\*y;  int(fun, a, b)  **ANSWER**  -(y\*(a^2 - b^2))/2 |
| **5)** | syms x y a b  fun = x\*y;  int(fun, 'x', a, b)  **ANSWER**  -(y\*(a^2 - b^2))/2 | | **6)** | syms x y a b  fun = x\*y;  int(fun, 'y', a, b)  **ANSWER**  -(x\*(a^2 - b^2))/2 |
| **7** | syms x y a b  fun = x\*y;  int(fun, 'a', a, b)  **ANSWER**  -x\*y\*(a - b) | | **8** | int(int(fun, y), x)  **OR**  A = int(fun, y);  B = int(A, x)  **ANSWER**  (x^2\*y^2)/4 |
| **9** | syms x y  fun = x\*y;  A = int(fun,y, 1-x, 1-x^2)  B = int(A, x, 0, 1)  **ANSWER**  1/24  double(B)  0.0417 | | **10** | **CODE:**  syms x  fun = x^2/(1+x^3);  A = int(fun,'x', 0, 1);  **ANSWER:**  A =  log(2)/3  double(A)  0.2310 |
| **11** | | A = int(fun, y, 1-x, 1-x^2)  B = int(A, x, a, b)  integral1 =  (x^2\*(x - 1)^2\*(x + 2))/2  integral2 =  b^3\*(b\*(b^2/12 - 3/8) + 1/3) - a^3\*(a\*(a^2/12 - 3/8) + 1/3)  **pretty(integral2)**  / / 2 \ \ / / 2 \ \  3 | | b | | 3 | | a | |  b | b | -- - 3/8 | + 1/3 | - a | a | -- - 3/8 | + 1/3 |  \ \ 12 / / \ \ 12 / / | | |

**SOLVE EACH OF THE FOLLOWING DEFINITE INTEGRALS USING “int()” command**

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Calculate the area between and the x-axis between the points of its intersection with x-axis, x = 0 and x = 2 | 2 |  |
| 3 |  | 4 |  |
| 5 |  | 6 |  |
| 7 |  | 8 |  |

**Symbolic differentiation in MATAB:**

The function diff can be used to carry out symbolic differentiation.

|  |  |  |  |
| --- | --- | --- | --- |
| **1)**  W.R.T ‘x’ | syms x  fun = x^3;  diff(fun, ‘x’)  OR  diff(fun)  **ANSWER**  3\*x^2 | **2)**  W.R.T ‘y’ | syms x  fun = x^3;  diff(fun, ‘y’)  **ANSWER**  0 |
| **3)** | syms x  fun = x^3;  fun = (x^2+4\*x+1)/((sqrt(x)-1)^3)  diff(fun, ‘x’)  **ANSWER**  (2\*x + 4)/(x^(1/2) - 1)^3 - (3\*(x^2 + 4\*x + 1))/(2\*x^(1/2)\*(x^(1/2) - 1)^4) | **4)**  NOTE: Find 3rd derivative | syms x  fun = sin(x^2);  diff(fun, ‘x’, 3)  **ANSWER**  - 12\*x\*sin(x^2) - 8\*x^3\*cos(x^2) |
| **4)** | diff([sin(x) cos(x); x^2 x],'x')  **ANSWER**  [ cos(x), -sin(x)]  [ 2\*x, 1] |  |  |

**Calculate the following derivatives:**

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Find and evaluate it at x = 1, y = 1 and z = 1  syms x y z  A = '(x\*y\*z)/(x+y+z)'  B = diff(diff(diff(A, x),y),z);  feval(fun, 1, 1, 1) | 2 | Find and evaluate it at x = 1, y = 1 and z = 1 |

**CHAPTER 10**

**Symbolic and numerical solution of simple 1st order differential equations**

MATLAB has excellent differential equation solver capabilities. Three highly used MATLAB commands for this purpose are: dsolve, ode23 and ode45. ode23 and ode45 are numerical solvers whereas dsolve is a symbolic solver. “dsolve” can be used to solve many differential equations for which a symbolic solution is possible. In many cases, ode45 is more accurate than ode23, but computationally more expensive. When using this command, a separate function file containing the actual list of differential equations must also be provided. The initial conditions for each differential equation should also be provided. **NOTE**: The default independent variable will be taken as ‘t’, if it has not been specified.

|  |  |
| --- | --- |
| **Example 01:**  **CODE:**  dsolve('Dy = x','x')  **ANSWER:**  x^2/2 + C2 | **Example 02:**  **CODE:**  dsolve('Dy = x')  **ANSWER:**  C4 + t\*x  **NOTE**: Here, the independent variable was not specified explicitly and hence the default‘t’ was considered. This is evident in the answer obtained |
| **Example 03:**  **CODE:**  dsolve('Dy = y','x')  **ANSWER:**  C8\*exp(x) | **Example 04:**  0  **CODE:**  dsolve('Dy = y+20','x')  **ANSWER:**  C16\*exp(x) – 20 |
| **Example 05:**  Note: At y = 0, x = 10  **CODE:**  dsolve('Dy = y','y(0)=10','x')  **ANSWER:**  C8\*exp(x) |  |
| **QUESTION 06:**  Note:  Use the domain: [0.5, 5]  Condition: At x = 0.5, y = 2  Plot the solution using ezplot command   |  |  | | --- | --- | | Code | Answer | | A = dsolve('Dy=(x^2-2\*x+3)/(y^2)','y(0.5) = 2','x') | 3^(1/3)\*((x\*(x^2 - 3\*x + 9))/3 + 11/8)^(1/3) | | pretty(A) | **/ 2 \1/3**  **1/3 | x (x - 3 x + 9) |**  **3 | ---------------- + 11/8 |**  **\ 3 /** | | ezplot(A,[0.5 5]) |  | | |

**USING ode23 command**:

Solve

Note:

Use the domain: [0.5, 5]

Condition: At x = 0.5, y = 2

First, create a function file:

|  |
| --- |
| function [DyDx] = MyODE01(x, y)  DyDx = (x^2-2\*x+3)/(y^2);  end |

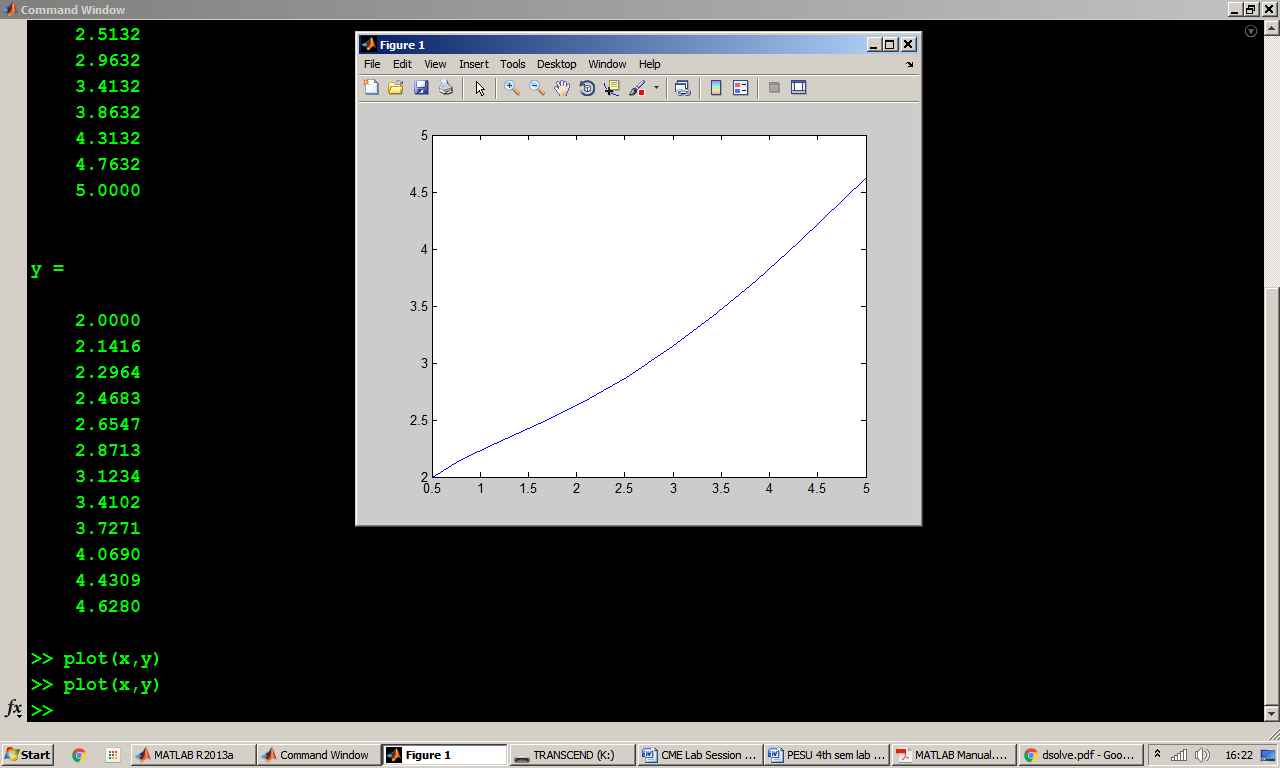
*In COMMAND WINDOW:*

domain = [0.5 5];

y\_at\_xi = 2;

[x, y] = ode23(‘MyODE01’, domain, y\_at\_xi);

plot(x,y)



**Solve the following ODE using *dsolve* and *ode23* commands. Also plot the solution when applicable**

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | x = [1, 3] and y at x = 1 is 4.2 | 2 | 𝑥= [0, 4] and 𝑦(0) = −0.5 |

**CHAPTER 11**

**Simple input and output operations input command, output commands: fprintf, disp, sprint**