MATLAB PROGRAMMING BASICS

IEEE STUDENT BRANCH, NIT TRICHY

B.HANUMANTHA RAO,

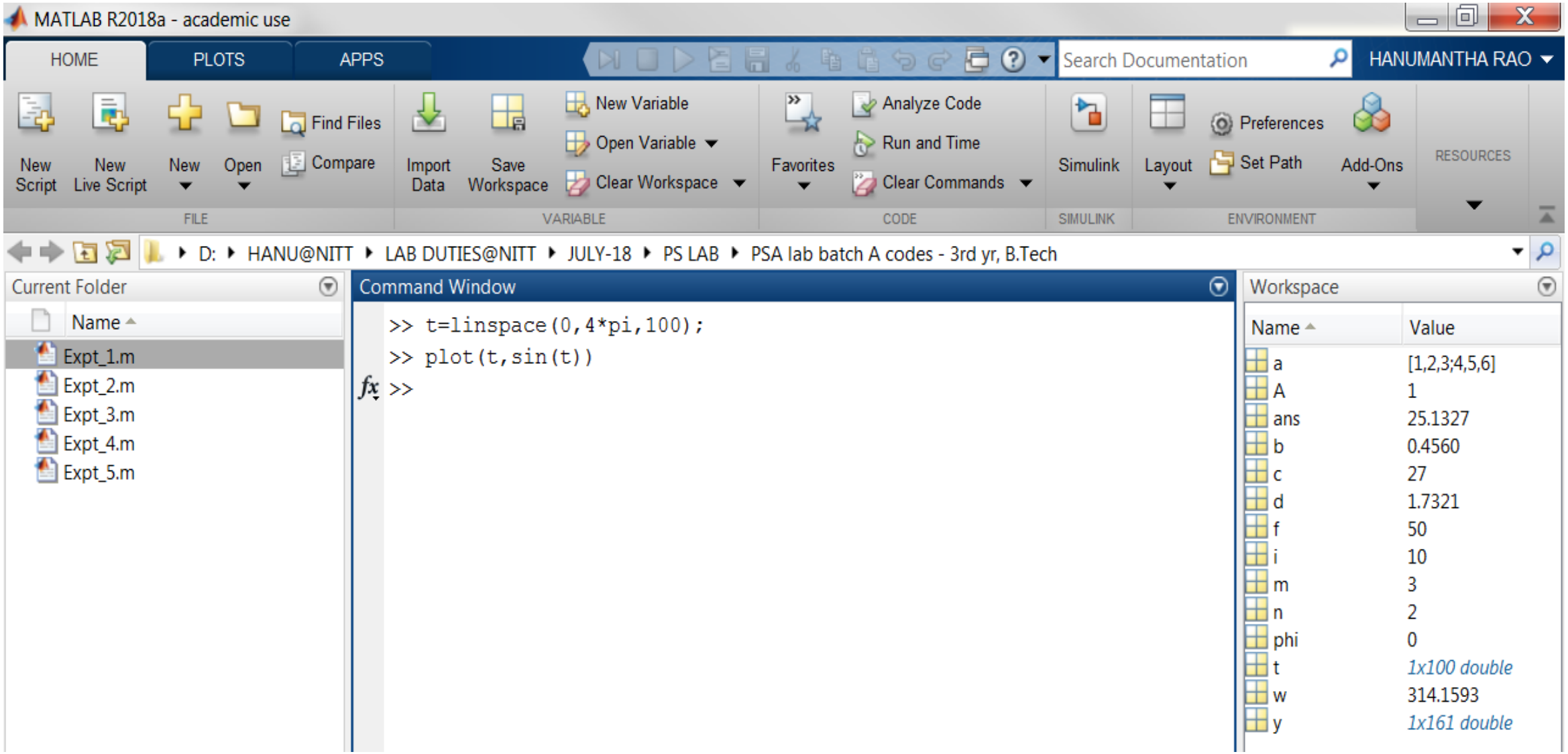
Research Scholar, EEE

Contents:

Features of Matlab:

- ✓ **High-level language of technical computing**
- ✓ **Development environment for engineers, scientists**
- ✓ **Interactive tools for design, problem solving**
- ✓ **Mathematical function libraries**
- ✓ **Graphics and data visualization tools**
- ✓ **Custom GUIs**
- ✓ **External Interfaces: C, C++, Fortran, Java, COM, Excel, .NET**

Matlab Screen



MATLAB variables:

- MATLAB variables are created with an assignment statement. The syntax of variable as assignment is

  variable name = a value (or an expression);

  For example,

  >> x = expression;

  where expression is a combination of numerical values, mathematical operators, variables, and function calls. On other words, expression can involve:

  manual entry

  built-in functions

  user-defined functions

TASK1: Execute all the commands in pdf from serial no.1 to 3.

Entering Arrays:

- Depending on its dimensions, a variable is called as a scalar, a vector, or a matrix.

Scalar	⟹	A single value is stored.(1-by-1)
Row vector	⟹	Values are stored in 1 row and n columns.(1-by-n)
Column vector	⟹	Values are stored in n row and 1 columns.(n-by-1)
Matrix	⟹	Values are stored in m rows and n columns.(m-by-n)

- In general, a variable that contains multiple values is called an array. Scalars, vectors, and matrices are therefore (numeric) *arrays*.

Create and Run a Script:

- One way to create a new blank script is to use the command edit.

```
>> edit          %creates a blank script named untitled.m
```

- You can also provide the filename as a part of the edit command.

```
>> edit newScript
```

- The above command creates a blank script named *newScript.m*.

- Not only does the edit command create a new file, but it also opens existing files.

```
>> edit filename
```

- You may run a script by typing the file name without the extension in the Command Window.

Code Sections:

How to divide the code into sections:

- To create a sections in script file type %% followed by space and then enter section title.
- You can create several sections in your scripts this way.
- If your file is large, you can navigate your sections using goto drop down menu in menu bar>choose one of the section titles you provided earlier>note that the current section is highlighted in yellow.

Comments:

- Text following a percent sign, %, is a comment. A comment is not executed in the code file. Comments can appear on lines by themselves, or they can be appended to the end of a line.
- The first contiguous block of comments is reserved for help information which is displayed when you use the help or doc commands.
- >>help filename

Creating vectors:

Creating Normal vectors:

- V=[1 2 3 4 5] ⟶ A vector with 5 elements.
- V=[first:increment:last] ⟶ A vector with equally spaced elements and the 'increment' can be negative.

Creating Evenly-Spaced Vectors:

- In some cases, when creating an evenly-spaced vector, you may know the number of elements you want the vector to contain, and not the spacing between the elements. In these cases, you can use *linspace*.
- M=linspace(first,last,n) ⟶ A vector with 'n' elements linearly spaced.
- Note that both the *colon* operator and *linspace* create row vectors.

TASK2: Execute all the commands in from serial no.4 to 5.

Vector Indexing:

- The location of each element in a vector is referred to as an index.
- In MATLAB, the index of the first element in a vector is 1. You can access an element by providing the index within the parenthesis after the variable name.
- Instead of using a number as an index directly, you can use a variable that contains a number as an index.
- The following command assigns the value of the variable y to the fourth element of the vector x.

```
>> x(4) = y;
```

- You can create new elements in a vector by assigning value to an index that doesn't exist.
- When you separate the values by semicolons ; MATLAB creates a column vector (n-by-1)

```
>> varName = [1;3];
```

- The operator " ' " can also be used to calculate the complex conjugate transpose.
- If you want to transpose the vector of complex numbers without calculating the complex conjugate, use the operator " .' "

- **TASK3:** Execute all the commands in serial no.6.

Creating Matrices:

- A matrix is a two-dimensional array consisting of *m* rows and *n* columns.
- To type a matrix into MATLAB you must begin with a square bracket, [separate elements in a row with spaces or commas , use a semicolon (;) to separate rows end the matrix with another square bracket].
- Here is a typical example. To enter a matrix A, such as,

```
>>A =[1 2 3;4 5 6;7 8 9];
```

Special Matrices:

- zeros → generates matrix with all the elements are 1.
- ones → generates matrix with all the elements are 1.
- eye → generates an identity matrix.
- rand → generates random matrix.

- **TASK4:** Execute all the commands from serial no.7 to 8.

Operations on Matrices:

- Arithmetic Operators : **+, - ,*, / ,^, ‘**
- Logical Operators : **Element wise (.), &, |, ~**
- Colon Operation : (**:**)
- Creating a sub-matrix/Deleting row or column.
- Eigen values/eigen Vectors.
- Concatenating matrices

TASK5: Execute all the commands from serial no.9 to 10.

TASK6: Generate the matrix shown.

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	2	2	2	1	0
0	1	2	3	2	1	0
0	1	2	2	2	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Statistical Operations on Matrices:

- Generally, statistical analysis functions in MATLAB work on each column of a matrix independently, by default.
- The *cumsum* function returns a matrix the same size as the input, with the cumulative sum over each column.
- You can also calculate the cumulative product of a matrix using the *cumprod* function.
- In the min, max, functions, the third input is reserved for the dimension to operate along. To skip defining the second input value, use an empty array, [].

```
>> minRowElement = min(M,[],2);
```

Summary: Statistical Operations on Matrices:

mean	⟶	Average or mean value
median	⟶	Median value
mode	⟶	Most frequent values
std	⟶	Standard deviation
var	⟶	Variance

- **TASK7:** Execute all the commands from serial no.12 to 14.

Matrices import and export:

Importing Syntax:

- >>a = xlsread(filename); reads the first worksheet in the Microsoft Excel spreadsheet workbook named filename and returns the numeric data in a matrix.
- >>b = xlsread(filename,sheet); reads the specified worksheet.
- >>c = xlsread(filename,xlRange); reads from the specified range of the first worksheet in the workbook.
- >>d= xlsread(filename,sheet,xlRange); reads from the specified worksheet and range.

Exporting Syntax:

- >>e=xlswrite(filename,A); writes matrix A to the first worksheet in the Microsoft Excel spreadsheet workbook filename starting at cell A1.
- >>f=xlswrite(filename,A,sheet); writes to the specified worksheet.
- >>g=xlswrite(filename,A,xlRange); writes to the rectangular region specified by xlRange in the first worksheet of the workbook.
- >>h=xlswrite(filename,A,sheet,xlRange); writes to the specified worksheet and range.

- TASK8: Use all the above commands to import and export matrices.

Introduction of Table:

- You can use the table function with the workspace variables as inputs to create a table. The following code creates a table, data with variables a, b, and c.

  >> T = table(a,b,c)

- You can use the ‘array2table function’ to convert from a homogenous array to a table. The following code creates a table, data from a matrix A.

  >>T= array2table(A)

- The following code creates a table named data with custom variable names.

  >> data = array2table(A, ‘VariableNames', {'X','Y','Z'})

- you can import the data programmatically in the command window using the *readtable* function.

  >> EPL = readtable('EPLresults.xls')

- Use the writetable function to create a file from a table.

  >> writetable(tableName,'fileName')

- TASK12: Execute all the above commands.

Operations on Table:

- You can sort a table on a specific variable using the sortrows function.

```
>> tSort = sortrows(tableName, 'SortingVariable')
```

- The sortrows function returns the values sorted in ascending order rather than descending order. You can supply the optional 'descend' parameter to the sortrows function to sort in descending order.

```
>> tSort = sortrows(tableName, 'SortingVariable','descend')
```

- To sort on two variables, supply them in order to the sortrows function as a cell array.

```
>> tSort = sortrows(tableName, {'var1','var2'},'descend')
```

- TASK12: Extract Portions of a Table named ‘table’

Visualizing Matrices :

2-D Plotting functions:

```
>>x=[0:2*pi/20:2*pi];
>>y=sin(x);
>>z=cos(x);
```

- A plot of y with respect to x is obtained by plot(independent variable,dependent variable).

```
>>plot(x,y);

>> plottools                    %for editing the plots
```

- plotting sine and cosine waveforms together in a single figure window using one plot command.

```
>>plot(x,y,x,z)
```

- plotting sine and cosine waveforms in a singe figure window sequentially.

```
>>plot(x,y)
>>hold on
>>plot(x,z)
>>hold off
```

Plotting Against Index Value:

- If only a single vector is passed to plot, then the values on the x-axis are the index values. They always begin with 1 and end with the number of elements in the vector.

TASK9: Execute all the commands from serial no.17 to 18.

Annotating Plots:

- You can use commands to add annotations to your plot so that you can easily replicate the same plot programmatically.
- Labels can be added to plots using plot annotation functions, such as xlabel. The input to these functions must be text which is enclosed in single quotes, also called a character array.

```
>> xlabel('Text Label for X-Axis')
>> ylabel('Text Label for Y-Axis')
```

- The grid command will add a grid to your plot.

```
>> grid on
```

- You may remove the grid using the grid command.

```
>> grid off
```

- **Adding a Plot Legend**

```
>>legend('first label','second label')
```

- The legend function also accepts two optional arguments: the keyword 'Location' and a text description of the location which is given by compass points, such as 'north' or 'southwest'.

```
>>legend('lbl1','lbl2','Location','west')
```

- **TASK10:** Plot the voltage, current and power in single phase, AC, RL-series circuit.

Flow control:

- 1. **If/elese/elseif:**
- **%if condition%**

if condition

Commands;

end

%if-else condition%

if condition

commands1;

else

commands2;

end

2. SWITCH CASE:

EX:

```
n = input('Enter a number: ');
 switch n
    case -1
        disp('negative one')
    case 0
        disp('zero')
    case 1
        disp('positive one')
    otherwise
```

Thank You

1. **Some useful commands**:
   who, whos, clc, clear, clear all

   - who →List all variables in the workspace.
   - whos →Lists all variables and their sizes.
   - clc →Clear the screen.
   - clear a,b →Clears variable a and b from the workspace.
   - clear all →Clears all the variables form the workspace.

2. **Entering scalars**:

   - a=1
   - b=2.5
   - c=pi; →' ; ' suppresses display

3. **Operations with the scalars**:

   - c=a+b →Addition.
   - c=a-b →Subtraction.
   - c=a*b →Multiplication.
   - c=a/b →Division.
   - c=2+3i, d=3+j*4 →Complex numbers.

4. **Entering Vectors**:

   - v=[1 2 3 4 5] →A vector with 5 elements.
   - V=[first:increment:last] →A vector with equally spaced elements and the 'increment' can be negative.

     Ex: V=[0:0.5:5]

   - M=linspace(first,last,n) →A vector with 'n' elements linearly spaced.

     Ex: M=linspace(1,10,10)

   - M=logspace(first,last,n) →A vector with 'n' elements logarithmically spaced.

     Ex: M=logspace(1,5,3)

5. **Operations on Vectors**:
   if x=[1:1:10]; y=[2:2:20]; (NOTE: Vectors must be of identical dimension)

- x.*y →Element-by-element multiplication.
- x.$\hat{2}$ →Each element raised to power 2.
- x./y →Element-by-element division.
- **find** →Finds indices of nonzero elements.
- **length** →Computes number of elements.
- **max** →largest element.
- **min** →smallest element.
- **size** →array size.
- **sort** →Sorts each column.

6. **Vector Indexing**:

   - MATLAB indexing starts with 1, not 0.
   - a(n) returns the nth-element
     a=[3 4 2 6 9]=[a(1), a(2), a(3), a(4), a(5)]
   - The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.
     Ex: x=[12 13 5 8]
     a=x(2:3); →a=[13 5];
     b=x(1:end-1); →b=[12 13 5];
   - MATLAB contains functions to help you find desired values within a vector or matrix. Ex: vec = [5 3 1 9 7] To get the minimum value and its index: [minVal,minInd] = min(vec); (max works the same way) To find any the indices of specific values or ranges. ind = find(vec == 9); ind = find($vec \geq 2 \& vec \leq 6$)

7. **Entering matrices**:

   - A=[1 2;3 4]; →2-BY-2 Real matrix.
   - B=[1+j*2 2-j*3;3+j*1 4-j*5]; →2-BY-2 complex matrix.
   - c=[1 2; 3 4]+j*[5 6 ; 7 8]; →2-BY-2 complex matrix.

8. **Matrix generation**:

   - zeros(3) →3-by-3 matrix of zeros.
   - ones(2) →2-by-2 matrix of ones.
   - eye(3) →3-by-3 Identity matrix.
   - rand(3) →3-by-3 matrix of Uniformly distributed pseudo random numbers.

- randi(IMAX,N) →returns an N-by-N matrix with maximum values of 'IMAX'.

9. **Operations on Matrices**:

- A + B →Matrix addition.
- A - B →Matrix subtraction.
- A * B →Matrix multiplication.
- A' →Complex conjugate transpose of matrix 'A'.
- inv(A) →Inverse of a matrix.
- det(A) →Determinant of a matrix.
- diag(A) →Diagonal elements of a matrix.
- eig(A) →Eigenvalues of a matrix.
- rank(A) →Rank of a matrix.
- triu(A) →Upper triangular part a matrix.
- tril(A) →Lower triangular part a matrix.
- size(A) →size of a matrix.

10. **Initialization/Accessing of Matrices**:

- A([3,5],[1,2,4])=ones(2,3) →Selective initialization to 1s and 0s.
- C=B(:,2) →Accessing 2nd column of 'B' matrix.
- A(1:2,2:3) →Creating a 2-by-2 submatrix with A(1,2),A(1,3),A(2,2),A(2,3) elements.
- C=[A B] →Horizontal concatenation.
- C=[A; B] →Vertical concatenation.
- A(3,:)=[5 6] →Adding a 3rd row to a matrix A.
- B(:,2)=[ ] →Deleting 2nd column of matrix B.
- A([1,2],:)=A([2,1],:) →Interchanges rows 1 and 2.
- c=A(:) →Recording of elements in A.

11. **Logical operations**:

- & →Logical AND.
- | →Logical OR.
- ~ →Logical NOT.
- xor →Logical Exclusive-OR.

12. **Logical Functions**:

- find →Find indices of the non-zero elements.
- isnan →True for not a number.
- isinf →True for infinite elements.
- isempty →True for empty matrices.
- isequal →True if arrays are numerically equal
- isreal →True for matrix of only real elements.
- issparse →True if matrix is sparse.
- isstr →True for text string.

13. **Simple math functions**:

- sin, cos, tan (angles must be in radians)
- sind, cosd, tand (angles must be in degrees)
- asin, acos, atan. (result will be in radians)
- asind, acosd, atand. (result will be in degrees)
- log, exp, sqrt.

14. **Permanent functions**:

- eps →2.22*10-16 user defined value.
- pi →π pre-defined value.
- inf →1/0.
- nan →Not a number.

15. **Display formats**:

- format →Default(same as short)
- format short →Scaled fixed point format with 5 digits. Eg: 1.3333
- format long →Scaled fixed point format with 15 digits. Eg: 1.33333333333333

16. **some useful commands**:
save, load, help, input, disp

- save session →To save the workspace content in file session.mat
- load session →To load the variables from file session.mat to workspace.

- help →To search help about specific commands.
- disp('hello world') →To display on screen.
- R=input('enter input value of R') →To initialize variables from the keyboard

17. **2-D Plotting functions**:

```
x=[0:2*pi/20:2*pi];
y=sin(x);
z=cos(x);
```

- A plot of y with respect to x is obtained by plot(independent variable,dependent variable).

```
plot(x,y);
```

- plotting sine and cosine waveforms together in a single figure window using one plot command.

```
plot(x,y,x,z)
```

- plotting sine and cosine waveforms in a singe figure window sequentially.

```
plot(x,y)
hold on
plot(x,z)
```

18. **2-D Plotting functions**:

- If you want to change the color of your plot, you can use an additional input to the plot function to do this. The following command will plot y versus x in magenta.

```
plot(x,y,'m')
```

- Some other color codes are: 'b' blue ,'g' green, 'r' red, 'c' cyan, 'k' black.
- You can also change the line style using an optional argument. The following command will plot y versus x as a dotted line.

```
plot(x,y,':')
```

- Some other line style codes are: '' solid, ' ' dashed, '.' dash-dot.
- You can even combine these codes! Generally, the order doesn't matter, so the following two commands produce the same plot using a red dotted line.

```
plot(x,y,'r:')        plot(x,y,':r')
```

- In addition to color and line style, you can specify a marker style. The following command will plot y versus x using stars.
  plot(x,y,'*')
- Some other marker style codes are:'.' Points, 'o' circles, 'x' crosses.

19. **Extracting Portions of a Table:**:

- When indexing into a table, it can oftentimes be easier to remember a variable name as opposed to figuring out the specific column number. So, as an alternative to numeric indexing, you can create a subset using the variable name in single quotes.
  homeGoalsFor = EPL(:,'HomeGF');
- You can select multiple variables by name using a cell array of strings as input.
  goalsFor = EPL(:,{'HomeGF','AwayGF'});
- It is possible to index into a table using a combination of indexing by number and name.
  fhw = EPL(2:2:8,{{'Team','HomeWins'});
- You can index into the variable using regular array indexing once it has been extracted using dot notation. The following command finds the average value of the rows 11 through end of the variable HomeGA.
  ave = mean(EPL.HomeGA(11:end))
- You can add a new variable to a table using dot notation.
  tableName.NewVarName = newData
- If you want to access data from multiple variables, you can do that by indexing with curly braces. Inside the curly braces, you can specify the desired variables by either a numeric array containing the column numbers or as a cell array with the variable names.
  The following code creates two equivalent arrays from EPL.
  wins = EPL{{:, [27]};
  wins = EPL{{:, {'HomeWins','AwayWins'}};
- You can add multiple variables to a table using curly braces. Create the new variable names in a cell array.
  EPL{{:, {'Rand1','Rand2'}} = randn(20,2);

20. **Flow control**:

- *for - loop:*
  for $k = first : increment : last$
  – – – –
  – – – –
  end

- *if - statement:*
  $if(condition)$
  $----$
  end
- *if -else statement:*
  $if(condition1)$
  $----$
  $elseif(condition2)$
  $----$
  $else$
  $----$
  end
  end
- *while -loop statement:*
  $while(condition)$
  $----$
  end
- switch switch_expression
  case case_expression
  statements
  case case_expression
  statements

  otherwise
  statements
  end

- while expression
  statements
  end
- a = 10;
  while (a ¡ 20)
  fprintf('value of a: a = a+1;
  if(a $\geq$ 15)
  % terminate the loop using break statement
  break;
  end
- a = 10;
  while a ¡ 20
  if a == 15
  a = a + 1;
  continue;
  end fprintf('value of a: a = a + 1;
  end

1: RESIDENTIAL ENERGY MANAGEMENT SYSTEM

The electrical and electronics appliances installed in a residential building and their duration of operation, power rating are given in the following tables. The maximum consumption limit (MCL) of a particular consumer is assumed to be 2.5 kW throughout a day. The energy cost is Rs. 3/unit. The consumer has to pay a penalty of Rs. 6 per unit for consumption above MCL.

Table -1

Must_run Loads				
S.No	**Loads**	**Power (kW)**	**Duration of Operation**	**Number of Loads**
1	Fan	0.1	0.00 to 6.00	4
			6.00 to 9.00	2
			17.00 to 21.00	2
			21.00 to 24.00	4
2	Tube Light	0.06	5.00 to 7.00	3
			18.00 to 22.00	6
3	CFL	0.04	0.00 to 5.00	3
			5.00 to 7.00	6
			18.00 to 22.00	6
			22.00 to 24.00	3
4	TV	0.25	6.00 to 8.00	1
			17.00 to 22.00	1
5	Laptop/ Mobile charging	0.05	6.00 to 7.00	2
			7.00 to 8.00	2
			17.00 to 18.00	2
			18.00 to 19.00	2

Table-2

Flexible Loads				
S.No	**Loads**	**Power (kW)**	**Without Scheduling**	
			Runtime (h)	**Starting Time of Operation**
1	Well pump	1.50	1	06.00
2	PHEV charging	2.30	3	06.00
3	Washing Machine	2.70	2	18.00

Must_run=[0.52 0.52 0.52 0.52 0.52 0.82 0.92 0.55 0.2 0 0 0 0 0 0 0 0 0.55 1.15 1.05 1.05 1.05 0.52 0.52]

(Q) Compute the per day electricity bill payable to the utility.

	<MCL	**>MCL**
Total Consumption (kWh)		
Electricity Bill (Rs.)		
Total Electricity Bill Per Day (Rs.)		

2. COMPUTATION OF POWER AND POWER FACTOR

Use voltage and current samples given in the excel file to compute the real power and power factor.

Write a MATLAB program.

V (rms)	
I_1 (rms)	
P_1	
pf_1	
I_2 (rms)	
P_2	
pf_2	
I_3 (rms)	
P_3	
pf_3	

Questions

3 **Write a MATLAB program to formulate the bus admittance matrix of the system with the data given in Table - 1**

Table -1

Line No.	From Bus	To Bus	Resistance (p.u.)	Reactance (p.u.)	Half line charging admittance (p.u)
1	1	2	0.01008	0.05040	0.05125
2	1	3	0.00744	0.03720	0.03875
3	2	4	0.00744	0.03720	0.03875
4	3	4	0.01272	0.06360	0.06375