GETTING STARTED:

>> command;

- Means an instruction command has been issued at the MATLAB prompt.
- If a semicolon (;) is placed at the end of a command, then all output from that command is suppressed.
- Multiple commands can be placed on the same line, separated by semicolons ;.
- Comments are marked by the percent sign (%),

>> help command;

- provide information on the inputs, outputs, usage, and functionality of the command

>>clc - clears all input and output from the Command Window display, giving you a clean screen.

- **clear -** clears all functions and variables from memory.
- **home** – moves the cursor to the upper left corner of the window.
- **quit** - quits Matlab, although you can always do it the amateur way.

- ✓ marks (,, ;, :, etc.) have special meanings.
- ✓ 1.23 is represented as simply 1.23
- ✓ While the real number 4.56×10^7 can be written as 4.56e7.
- ✓ The imaginary number $\sqrt{-1}$ is denoted either by 1i or 1j,
- ✓ We will use the symbol 1j.
- ✓ The complex number whose real part is 5 and whose imaginary part is 3 will be written as 5+1j*3.
- ✓ Other constants preassigned by MATLAB are pi for π,
- ✓ inf for ∞, and NaN for not a number (for example, 0/0).
- ✓ These preassigned constants are very important and, to avoid confusion, should not be redefined by users.

VARIABLES

In MATLAB, which stands for MATrix LABoratory,

a. Matrix: A matrix is a two-dimensional set of numbers arranged in rows and columns. Numbers can be real- or complex-valued.

b. Array**:** This is another name for matrix.

The following are four types of matrices (or arrays):

• **Scalar:** This is a 1 × 1 matrix or a single number that is denoted by the *variable* symbol, that is, lowercase italic typeface like $a = a_{11}$

• **Column vector:** This is an (N × 1) matrix or a vertical arrangement of numbers. It is denoted by the *vector* symbol, that is, lowercase bold typeface like

$$\boldsymbol{x} = [\boldsymbol{x_{i1}}]_{i:1....N} = \begin{bmatrix} \boldsymbol{x_{11}} \\ \boldsymbol{x_{12}} \\ \vdots \\ \boldsymbol{x_{13}} \end{bmatrix}$$

A typical vector in linear algebra is denoted by the column vector.

- **Row vector:** This is a (1 ×M) matrix or a horizontal arrangement of Numbers. It is also denoted by the vector symbol, that is,

$$y = [y_{1j}]_{j=1,....,M} = [y_{11}\ y_{12} \dots.\ y_{1M}]$$

A one-dimensional discrete-time signal is typically represented By An array as a row vector.

- **General matrix:** This is the most general case of an ($N \times M$) matrix and is denoted by the matrix symbol, that is, uppercase bold typeface like

$$\mathbf{A} = [a_{ij}]_{i=1,\ldots,N;j=1,\ldots,m} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1M} \\ a_{21} & a_{22} & \cdots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NM} \end{bmatrix}$$

This arrangement is typically used for two-dimensional discrete-time signals or images.

MATLAB does not distinguish between an array and a matrix except for operations. The following assignments denote indicated matrix types in MATLAB:

a = [3] is a scalar,
x = [1,2,3] is a row vector,
y = [1;2;3] is a column vector, and
A = [1,2,3;4,5,6] is a matrix.

Exercise-1

1. **Create a 5-by-1 column vector of zeros.**
2. **Create a matrix of zeros with 2 rows and 4 columns**
3. **Create the row vector of odd numbers through 21,**
   L = 1 : 2 : 21
   L = 1 3 5 7 9 11 13 15 17 19 21
4. **Create a 6 x 1 vector a of zeros using the zeros command.**
5. **Create a row vector b from 325 to 405 with an interval of 20.**
6. **Use sum to find the sum a of vector b's elements.**
7. **Create two different vectors of the same length and add them.**
   a = [2, 1, 3]; b = [4 2 1]; c = a + b
8. **Now subtract them.** c = a - b
9. **Perform element-by-element multiplication on them.** c = a .* b
10. **Perform element-by-element division on them.** c = a ./ b
11. **Raise one of the vectors to the second power.** c = a .^ 2

MATLAB FUNCTIONS

- zeros(M,N) for creating a matrix of all zeros,
- ones(M,N) for creating matrix of all ones,
- eye(N) for creating an $N \times N$ identity matrix, etc.

OPERATORS

= assignment	== equality
+ addition	- subtraction or minus
* multiplication	.* array multiplication
^ power	.^ array power
/ division	./ array division
<> relational operators	& logical AND
| logical OR	~ logical NOT
’ transpose	.’ array transpose

ARRAY OPERATIONS

These operations treat matrices as arrays. They are also known as *dot operations* because the arithmetic operators are prefixed by a dot (.), that is, .*, ./, or .^.

a) Array multiplication:

i. For it to be a valid operation, both arrays must be the same size. Thus we have
   1. $x.*y \rightarrow 1D\ array$
   2. X.$*Y \rightarrow 2D\ array$

b) **Array exponentiation:** In this operation**, a scalar** is raised to the power equal to every element in an array, that is,

$$\mathtt{a.\hat{}x} \equiv \begin{bmatrix} a^{x_1} \\ a^{x_2} \\ \vdots \\ a^{x_N} \end{bmatrix}$$

is an ($N \times 1$) array,

whereas is an (N

$$\mathtt{a.\hat{}X} \equiv \begin{bmatrix} a^{x_{11}} & a^{x_{12}} & \cdots & a^{x_{1M}} \\ a^{x_{21}} & a^{x_{22}} & \cdots & a^{x_{2M}} \\ \vdots & \vdots & \ddots & \vdots \\ a^{x_{N1}} & a^{x_{N2}} & \cdots & a^{x_{NM}} \end{bmatrix}$$

$\times M$) array.

MATRIX OPERATION

a) **Matrix addition and subtraction: A**rray addition and subtraction. Two matrix operands be *exactly* the same size.

b) **Matrix conjugation:** This operation is meaningful only for complex valued matrices. It produces
   i. Matrix in which all imaginary parts are negated. It is denoted by $\boldsymbol{A}^*$ in analysis and
   ii. by conj(A) in MATLAB.

c) **Matrix transposition:** This is an operation in which every row (column) is turned into column (row). Let **X** be an ($N \times M$) matrix

$$\boldsymbol{X'} = [\boldsymbol{x_{ji}}]; \boldsymbol{j} = \mathbf{1}, \ldots., \boldsymbol{M}, \boldsymbol{i} = \mathbf{1}, \ldots., \boldsymbol{N}$$

is an ($M \times N$) matrix. To obtain just the transposition, we use the array operation of conjugation, that is, $\boldsymbol{A}.'$will do just the transposition.

d) **Multiplication by a scalar:**

$$ab \Rightarrow \mathtt{a*b}\ (\text{scalar})$$

$$a\mathrm{x} \Rightarrow \mathtt{a*x}\ (\text{vector or array})$$

$$a\mathrm{X} \Rightarrow \mathtt{a*X}\ (\text{matrix})$$

e) **Vector-vector multiplication:** Let **x** be an ($N \times 1$) and **y** be a ($1 \times M$) vectors. Then

$$\mathrm{x} * \mathrm{y} \Rightarrow \mathrm{xy} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \begin{bmatrix} y_1 & \cdots & y_M \end{bmatrix} = \begin{bmatrix} x_1 y_1 & \cdots & x_1 y_M \\ \vdots & \ddots & \vdots \\ x_N y_1 & \cdots & x_N y_M \end{bmatrix}$$

produces a matrix. If $M = N$, then

$$\mathrm{y} * \mathrm{x} \Rightarrow \mathrm{yx} = \begin{bmatrix} y_1 & \cdots & y_M \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix} = x_1 y_1 + \cdots + x_M y_M$$

f) **Matrix-vector multiplication:** If the matrix and the vector are compatible (i.e., the number of matrix-columns is equal to the vector-rows), then this operation produces a column vector:

$$\mathtt{y = A*x} \Rightarrow \mathrm{y} = \mathrm{Ax} = \begin{bmatrix} a_{11} & \cdots & a_{1M} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NM} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

Exercise-2

12. Find the sum S of vector L's elements

13. Form the matrix 2*4 matrix

```
A =
     2     3     2
     1     0     1
```

14. Write a Matlab program that create 4*4 matrix that name is a,

15. **Performed A=a*a and a+10;**
16. **Calculate sin(a)**
17. **Calculate the transpose of matrix a.**
18. **Create a 3*3 matrix and display the first row of and the second column on the screen.**
    d = [1 2 3; 2 3 4; 4 5 6]; d(1,:), d(:,2)

Matlab matrix and Vectors command Commands:

+	Element-by-element addition. (Dimensions must agree)
-	Element-by-element subtraction. (Dimensions must agree)
.*	Element-by-element multiplication. (Dimensions must agree)
./	Element-by-element division. (Dimensions must agree)
.^	Element-by-element exponentiation.
:	When used as the index of a matrix, denotes "ALL" elements of that dimension.
A(:,j)	j-th column of matrix A (column vector).
A(i,:)	i-th row of matrix A (row vector).
.'	Transpose (Reverses columns and rows).
'	Conjugate transpose (Reverses columns and rows and takes complex conjugates of elements).
*	Matrix multiplication, Cayley product (row-by-column, not element-by-element).

INDEXING OPERATIONS:

- x = [a:b:c], we can generate numbers from a to c in b increments. If b is positive (negative) then, we get increasing (decreasing) values in the sequence x.
- The fragment x(a:b:c) accesses elements of x beginning with index a in steps of b and ending at c.

- Similarly, the : operator can be used to extract a submatrix from a matrix. For example, B = A(2:4,3:6) extracts a 3×4 submatrix starting at row 2 and column 3.

- Another use of the **:** operator is in forming column vectors from row vectors or matrices.
- When used on the right-hand side of the equality (=) operator, the fragment x=A(:) forms a long column vector x of elements of A by concatenating its columns.

- Similarly, x=A(:,3) forms a vector x from the third column of A. when used on the right-hand side of the = operator, the fragment A(:)=x reformats elements in x into apredefined size of A.

PLOTTING

Basic 1D and more Plot Commands

figure	Creates a figure window to which MATLAB directs graphics output. An existing figure window can be made current using the command figure(n), where n is the figure number specified in the figure's title bar.
plot(x,y,'s')	Generates a plot of y w.r.t. x with color, line style and marker specified by the character string s. For example, plot(x,y,'c:+') plots a cyan dotted line with a plus marker at each data point. The string s is optional and default values are assumed if s is not specified. For default values, list of available colors, line styles, markers and their corresponding character representations, type help plot.
plot(x,y1,x,y2, ...)	Creates a multiple plot of y1 vs. x, y2 vs. x and so on, on the same _g- ure. MATLAB cycles through a predefined set of colors to distinguish between the multiple plots.
axis([xmin,xmax, ymin,ymax])	Specifies axis limits for the x- and y- axes. This command is optional and by default MATLAB determines axes limits depending on the range of data used in plotting.
title('...')	Adds a title to the graph in the current figure window. The title is specified as a string within single quotes.
xlabel('...')	Adds a label to the x-axis of the graph in the current figure window This is again specified as a string within single quotes.
ylabel('...')	Similar to the xlabel command.
grid on	Adds grid lines to the current axes.
grid off	Removes grid lines from the current axes.
hold on	This is used to add plots to an existing graph. When hold is set to on, MATLAB does not reset the current _gure and any further plots are drawn in the current _gure
hold off	This stops plotting on the same _gure and resets axes properties to their default values before drawing a new plot.
legend	Adds a legend to an existing _gure to distinguish between the plotted curves.
ezplot('f(x)', [x0,xn])	Plots the function represented by the string f(x) in the interval x0 _ x _ xn.

Excersize-3

19. To create two-dimensional line plots, use the plot function. For example, plot the value of the sine function from 0 to 2π:

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y
```

20. You can label the axes and add a title.

```
xlabel('x')
ylabel('sin(x)')
title('Plot of the Sine Function')
```

21. Let us plot projectile trajectories using equations for ideal projectile motion

$$y(t) = y_0 - \frac{1}{2}gt^2 + (v_0 \sin(\theta_0))\, t,$$
$$x(t) = x_0 + (v_0 \cos(\theta_0))\, t,$$

where y(t) is the vertical distance and x(t) is the horizontal distance traveled by the projectile in metres, g is the acceleration due to Earth's gravity = 9:8 m/s2 and t is time in seconds. Let us assume that the initial velocity of the projectile v0 = 50:75 m/s and the projectile's launching angle _0 = 5π/12 radians. The initial vertical and horizontal positions of the projectile are given by y0 = 0 m and x0 = 0 m. Let us now plot y vs. t and x vs. t in two separate graphs with the vector:

t=0:0.1:10 representing time in seconds. Give appropriate titles to the graphs and label the axes. Make sure the grid lines are visible.

Solution:

We first plot x and y in separate figures:

```
>> t = 0 : 0.1 : 10;
>> g = 9.8;
>> v0 = 50.75;
>> theta0 = 5*pi/12;
>> y0 = 0;
>> x0 = 0;
>> y = y0 - 0.5 * g * t.^2 + v0*sin(theta0).*t;
>> x = x0 + v0*cos(theta0).*t;
>>
>> figure;
>> plot(t,x);
>> title('x(t) vs. t');
>> xlabel('Time (s)');
>> ylabel('Horizontal Distance (m)');
>> grid on;
>>
>> figure;
>> plot(t,y);
>> title('y(t) vs. t');
>> xlabel('Time (s)');
>> ylabel('Altitude (m)');
>> grid on;
```

22. The range of the projectile is the distance from the origin to the point of impact on horizontal ground.

$$\theta_0^1 = \left(\frac{5\pi}{12} - 0.255\right) \text{ radians and}$$
$$\theta_0^2 = \left(\frac{5\pi}{12} - 0.425\right) \text{ radians.}$$

23. Plot the following equation:

(a) Using the plot command for multiple plots, plot y = sin(x) and y = cos(x) on the same graph for values of x de_ned by: x = 0:pi/30:2*pi.

b) Using the plot command for a single plot and the hold commands, plot y = sin(x) and y = cos(x) on the same graph for values of x de_ned by: x = 0:pi/30:2*pi.

c) Using the ezplot command, plot y = 2/3 cos(x) for values of x such that 0 <=x<= 2* pi.

Solution:

```
>> x = 0 : pi/30 : 2*pi;
>> plot(x,sin(x),x,cos(x));
>> title('y = sin(x) and y = cos(x)');
>> xlabel('x');
>> ylabel('y');
>> legend('y = sin(x)','y = cos(x)');
>> grid on;
>> x = 0 : pi/30 : 2*pi;
>> plot(x,sin(x));
>> title('y = sin(x) and y = cos(x)');
>> xlabel('x');
>> ylabel('y');
>> grid on;
>> hold on;
>> plot(x,cos(x),'r');
```

```
>> legend('y = sin(x)','y = cos(x)');
>> ezplot('(2/3)*cos(pi*x)',[0,2*pi]);
>> title('High Frequency Cosine Function');
>> xlabel('x');
>> ylabel('y');
>> grid on;
```

24. Plot the following command:

a) Using the plot command for multiple plots, plot y = atan(x) and y = acot(x) on the same graph for values of x de_ned by x = -pi/2:pi/30:pi/2.

b) Using the plot command for a single plot and the hold commands, plot y = atan(x) and y = acot(x) on the same graph for values of x de_ned by x = -pi/2:pi/30:pi/2.

c) Using the ezplot command, plot y = 2 3 sin(9_x), for values of x such that 0 <= x <= 2*pi.

Command Plotting function II:

log(n)	Calculates the natural logarithm (base e) of n
semilogy(x,y)	Graphs a plot of y vs. x using a logarithmic scale (powers of ten) on the y-axis.
semilogx(x,y)	Graphs a plot of y vs. x using a logarithmic scale (powers of ten) on the x-axis.
loglog(x,y)	Graphs a plot of y vs. x using a logarithmic scale (powers of ten) onboth axes. The logarithmic scales prove most useful when the value spans multiple orders of magnitude.

25. Graph the efficiency of several programming algorithms according to big-O notation, a method of describing the running time of algorithms.

Algorithm #1: O(n)
Algorithm #2: O(n2)
Algorithm #3: O(n3)
Algorithm #4: $O(2^n)$
Algorithm #5: O(e^n)

After generating an initial graph with ranging from 0 to 8, use logarithmic scaling on the y-axis of a second graph to make it more readable. You can also use the mouse to change the y-axis scale. Go to the main menu of the _gure, click Edit>Axes Properties. . . , the property editor dia-logue will pop out. There, you can also change the font, the range of the axes, . . .Try to play with it.
Solution:

```
>> n=0:0.01:8;
>> plot(n,n,n,n.^2,n,n.^3,n,2.^n,n,exp(n))
>> title('Big-O characteristics of Algorithms: Linear Plot')
>> ylabel('Estimate of Running Time')
>> xlabel('n (number of elements)')
>> legend('O(n)','O(n^2)','O(n^3)', 'O (2^n)','O(e^n)')
>> grid on;
```

Or

```
>> n = 0:0.01:8;
>> semilogy(n,n,'b',n,n.^2,'r',n,n.^3,'g',n,2.^n,'c',n,exp(n),'k')
>> title('Big-O characteristics: Logarithmic Plot')
>> ylabel('Estimate of Running Time')
>> xlabel('n (number of elements)')
>> legend('O(n)','O(n^2)','O(n^3)', 'O(2^n)','O(e^n)')
```

CONTROL-FLOW

a. If-else:

```
if condition1
   command1
elseif condition2
  command2
else
  command3
end
```

b. for..end loop:

```
for index = values
    program statements
:
  End
```

EXAMPLE 1.1 Consider the following sum of sinusoidal functions.

$$x(t) = \sin(2\pi t) + \tfrac{1}{3}\sin(6\pi t) + \tfrac{1}{5}\sin(10\pi t) = \sum_{k=1}^{3} \frac{1}{k}\sin(2\pi kt), \qquad 0 \le t \le 1$$

Using MATLAB, we want to generate samples of $x(t)$ at time instances 0:0.01:1. We will discuss three approaches.

for i = a:b	The for loop repeats statements a speci_c number of times, starting with i = a and ending with i = b, incrementing i by 1 each iteration of the loop. The number of iterations will be b - a + 1.
while condition	The while loop repeats statements an inde_nite number of times as long as the user-de_ned condition is met.
for i = a:h:b	The for loop works exactly the same except that i is incremented by h after each iteration of the loop.
clear	Clears all previously de_ned variables and expressions.
fprintf	Outputs strings and variables to the Command Window. See below for an example.
abs(x)	Returns the absolute value of the de_ned variable or expression x.
factorial(n)	Returns the factorial of the de_ned variable or expression n. The ellipses can be used to break up long lines by providing a contin- uation to the next line. Strings must be ended before the ellipses but can be immediately restarted on the next line. Examples below show this.

fprintf:

```
This is an example of how to use fprintf to display text to the command window.
fprintf ('\nOrdinary Differential Equations are not so ordinary.\n');
fprintf ('-----------------------------------------------------'...'----------------\n');
fprintf ('This course is CME %g: ODEs for Engineers. My expected'...' grade is %g\n',102,100);
x = 100; y = 96;
fprintf ('The previous course was CME %g: Vector Calculus for '...'Engineers. My grade was: %g\n',x,y);
```

The Matlab command window displays:

```
Ordinary Differential Equations are not so ordinary.
---------------------------------------------------------------------
This course is CME 102: ODEs for Engineers. My expected grade is 100
The previous course was CME 100: Vector Calculus. My grade was: 96
```

The command fprintf takes a string and prints it as is. The character nn is one of several \Escape Characters" for fprintf that can be placed within strings given to fprintf. nn speci_es a new line. %g is one of many \Speci_ers" that fprintf uses and it represents a placeholder for a value given later in the call to fprintf. The order of the arguments given to fprintf determine which %g is replaced with which variable or number. Experiment with the code above to see what nn can do and how %g can be used.

SCRIPTS AND FUNCTIONS

a. Scripts:

The script is executed by typing the name of the script at the command prompt. The script file Must be in the current directory on in the directory of the path environment.
Example 1.1. A general form of this function is

$$x(t) = \sum_{k=1}^{K} c_k \sin(2\pi kt) \qquad (1.1)$$

If we want to experiment with different values of the coefficients *ck* and/or the number of terms *K*, then we should create a script file. To implement the third approach in Example 1.1,

we can write a script file % Script file to implement (1.1)

```
t = 0:0.01:1; k = 1:2:5; ck = 1./k;
```

```
xt = ck * sin(2*pi*k'*t);
```

b. Functions:

Functions are also m-files (with extension .m). A major difference between script and function files is that the first executable line in a function file begins with the keyword function followed by an output-input variable declaration. As an example, consider the computation of the $x(t)$ function in Example 1.1 with an arbitrary number of sinusoidal terms, which we will implement as a function stored as m-file

sinsum.m.

```
function xt = sinsum(t,ck)
% Computes sum of sinusoidal terms of the form in (1.1)
% x = sinsum(t,ck)
%
K = length(ck); k = 1:K;
ck = ck(:)'; t = t(:)';
xt = ck * sin(2*pi*k'*t);
```

The vectors t and ck should be assigned prior to using the sinsum function. Note that ck(:)' and t(:)' use indexing and transposition operations to force them to be row vectors.

M-Files/Scripts:

A new M-file can be created by clicking on the \New M-file" icon on the top left of the Command Window. An M-file has a .m extension. The name of the file should not conict with any existing MATLAB command or variable.

Exercise-4

26. Taylor Series approximation to represent one of these functions:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots + (-1)^n \frac{x^{2n+1}}{(2n+1)!} + \ldots$$

You will use x = 3 as the value at which to evaluate the function.

a) Compute and display the exact error of using the first 2 and 5 terms of the series as compared to the actual solution when the function is evaluated at x =x/3

b) Compute and display the number of terms necessary for the function to be within the allowed error.

Solution:

a) CODE from M-file

```
clear;
x = pi/3;
% Iterate 2 terms for our approximation.
SIN_Approx2 = 0;
for j=0:2
SIN_Approx2 = SIN_Approx2 + (-1)^j*x^(2*j+1)/factorial(2*j+1);
end
SIN_Error2 = abs(SIN_Approx2 - sin(x));
% Iterate 5 terms for our approximation.
SIN_Approx5 = 0;
for j=0:5
SIN_Approx5 = SIN_Approx5 + (-1)^j*x^(2*j+1)/factorial(2*j+1);
End

SIN_Error5 = abs(SIN_Approx5 - sin(x));
fprintf('\nError with 2 terms:\n')
fprintf ('--------------------------\n')
fprintf ( 'sin(pi/3): %g\n',SIN_Error2 )
fprintf ('\nError with 5 terms: \n')
fprintf ('--------------------------\n')
fprintf ( 'sin(pi/3): %g\n',SIN_Error5)
```

b) CODE from M-_le:

```
clear;
SIN_APP = 0; % This is the sine approximation.
```

```
n = 0; x = 3;
% Iterate until our approximation is below the error tolerance.
while abs( SIN_APP - sin(x) ) >= 0.001
SIN_APP = SIN_APP + (-1)^n*x^(2*n+1)/factorial(2*n+1);
n = n + 1;
end
SIN_Terms = n;
SIN_Error = abs( SIN_APP - sin(x) );
% Output
fprintf ('\nNumber of Terms Needed for the function to be within the'...
' allowed error:\n');
fprintf ('-------------------------------------------------------------'...
'-------------\n');
fprintf ('sin(3): %g terms | Error = %g\n',SIN_Terms,SIN_Error);
```

Symbols and Strings: Differentiation and Integration

syms x y z	Declares variables x, y, and z as symbolic variables.
diff(f)	Symbolically computes derivative of the single-variable symbolic expression f.
diff(f,n)	Symbolically computes the n-th derivative of the single-variable symbolic expression f
diff(f,'x')	Symbolically di_erentiates f with respect to x.
diff(f,'x',n)	**Symbolically computes the n-th derivative with respect to x of f.**
int(f)	Symbolically integrates the single-variable symbolic expression f.
int(f,x)	Symbolically integrates f with respect to x.

27. Define the following function using syms:

$$f(x) = x^2 e^x - 5x^3.$$

```
syms x
f = x^2*exp(x) - 5*x^3;
```

28. Compute the integral, and the first and second derivatives of the above function symbolically.

```
>> int(f)
ans =
x^2*exp(x)-2*x*exp(x)+2*exp(x)-5/4*x^4
>> diff(f)
ans =
2*x*exp(x)+x^2*exp(x)-15*x^2
>> diff(f,2)
ans =
        2*exp(x)+4*x*exp(x)+x^2*exp(x)-30*x
```

29. Defne the following function using syms:

$$f(x, y, z) = x^2 e^y - 5z^2.$$

Compute the integral with respect to x and second derivative with respect to z.

```
>> syms x y z
>> f = x^2*exp(y) - 5*z^2;
>> int(f,'x')
ans =
1/3*x^3*exp(y)-5*z^2*x
>> diff(f,'z',2)
```

Stem():

30.

The following set of commands displays a discrete-time sine function using these constructs

```
>> n = 0:1:40; % sample index from 0 to 20
>> x = sin(0.1*pi*n); % Evaluate sin(0.2 pi n)
>> Hs = stem(n,x,'b','filled'); % Stem-plot with handle Hs
>> set(Hs,'markersize',4); % Change circle size
>> xlabel('n'); ylabel('x(n)'); % Label axis
>> title('Stem Plot of sin(0.2 pi n)'); % Title plot
```

31.

Mixes both plot() and stem() in same function:

```
>> plot(t,xt,'b'); hold on; % Create plot with blue line
>> Hs = stem(n*0.05,xn,'b','filled'); % Stem-plot with handle Hs
```

```
>> set(Hs,'markersize',4); hold off; % Change circle size
```

32.

Subplot():

```
>>subplot(2,1,1); % Two rows, one column, first plot
>> plot(t,x,'b'); % Create plot with blue line
. . .
>> subplot(2,1,2); % Two rows, one column, second plot
>> Hs = stem(n,x,'b','filled'); % Stem-plot with handle Hs
```