MATLAB

A Tutorial By

MASOOD EJAZ

Note: This tutorial is written specifically for *CET 3464 – Software Programming in Engineering Technology*, a course offered as part of the BSECET program at Valencia College. This tutorial can be used with any version of MATLAB or Octave.

INTRODUCTION

MATLAB (**Mat**rix **Lab**oratory), a product of Mathworks, is a scientific software package designed to provide integrated numeric computation and graphics visualization in high-level programming language. MATLAB program consists of standard and specialized toolboxes allowing users great flexibility in the manipulation of data and in the form of matrix arrays for computation and visualization.

MATLAB inputs can be entered at the "command line" or from "m-files", also called *scripts*, which contain a programming-like set of instructions to be executed by MATLAB. In the aspect of programming, MATLAB works differently from FORTRAN, C, or Basic, e.g. no dimensioning required for matrix arrays and no object code file generated. MATLAB offers some standard toolboxes and many optional toolboxes (at extra cost, of course!) such as financial toolbox, wavelets toolbox, statistics toolbox, bioinformatics toolbox etc. Users may create their own toolboxes consisted of "m-files" written for specific applications. The original version of MATLAB was written in FORTRAN but later was rewritten in C. You are encouraged to use MATLAB's on-line help files for functions and commands associated with available toolboxes.

DOWNLOAD:

MATLAB is not free to download but there is another software that works like MATLAB. This software is free to download and it is called GNU Octave. Its environment is very similar to MATLAB. It can be downloaded from: https://www.gnu.org/software/octave/

MATLAB ENVIRONMENT

MATLAB environment is very simple; it is a command line editor as against LabVIEW which is a graphic editor software. Two MATLAB windows that you will use very frequently are *command window* and *editor*. As mentioned earlier, command window is where you can write an instruction and press *enter* to execute it, whereas editor is where you can write programs or scripts (group of instructions) and run them such that all the instructions inside a program will run sequentially. These two windows are shown in *figure # 1*. In this figure, editor is *tiled* inside command window; you can un-tile it too

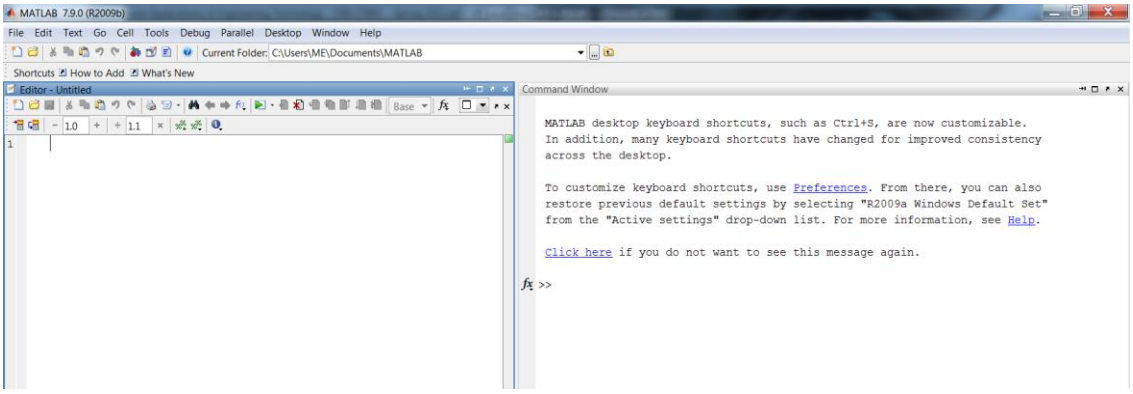


Figure 1: MATLAB Editor (left) and Command Window (right)

In addition to these windows, you can also open some other windows from *Desktop* menu. These include *Command History*, *Current Folder* and *Workspace* windows. Command History window keeps the log of each command entered into the Command Window. Current Folder shows contents of your current selected folder. Workspace window shows values of all the variables along with their minimum and maximum values used in MATLAB by user either from command window or by executing a script.

A script file can be executed in two ways; from editor, by pressing *RUN* button from toolbar menu or *Debug* → *Run*, and from command window by typing the name of your script file. Make sure when you are running your program from command window, you should be in the same folder where your program is saved.

Name of a script file or m-file can only be start with a letter, not a digit. Numbers can be part of name as well as underscore (_), the only special character allowed in naming format.

VARIABLES

In MATLAB, data can be assigned to different variables. Data can be scalar, vector, matrix or string of characters. Variable name follows the same pattern as script file name, i.e. it cannot start with a digit, only letter. Numbers and underscore can be part of variable name. Also, upper case and lower case letters are treated as two different variables. Hence *MATLAB, Matlab, MaTlaB, MATlab* etc. will all be treated as different variables.

Variables are assigned using " = " sign. Different variables can be assigned either in the same line with commas (,) or semicolon (;) separating them, or they can be defined individually in separate lines. Anytime you place a semicolon in front of a variable, its value will not be printed in the command window.

```
>> MATLAB = 23; Matlab = 76; MaTLab = 100

MaTLab =

   100
```

Note that only *MaTLab = 100* is being printed in the command window as there is no semi-colon in front of it.

To observe the current variables available in MATLAB workspace, you can use the following commands,

who: shows all the variables currently present in the MATLAB workspace

whos: shows current variables, their type, and size

```
>> who

Your variables are:

MATLAB  MaTLab  Matlab

>> whos
  Name        Size            Bytes  Class     Attributes

  MATLAB      1x1                 8  double
  MaTLab      1x1                 8  double
  Matlab      1x1                 8  double
```

To assign a *string* to a variable, characters have to be inside ‘’

```
>> Masood = 'ejaz'

Masood =

ejaz
```

Note that value of *Masood* is printed since there is no semi-colon after *‘ejaz’*

Maximum length allowed for a variable name is 63 characters. It can be found out by typing the following command: *namelengthmax*.

```
>> namelengthmax

ans =

    63
```

Observe that value of *namelengthmax* is saved in a variable *ans*. If an operation is carried out in the command window that yields a result and you do not assign any variable to save the result, it will be saved in the default variable *ans*.

To delete all the variables in the workspace, simply type *clear*. To delete some of the variables, type *clear* followed by the variable names separated by blank spaces.

```
>> who

Your variables are:

MATLAB  MaTLaB  Masood  Matlab  ans

>> clear MATLAB Masood
>> who

Your variables are:

MaTLaB  Matlab  ans
```

NOTE: *Any command that is written in the command window can be recalled either from 'command history' window or by pressing "↑", although"↑"cycles through previous commands sequentially.*

NUMBER FORMATS

In MATLAB, *floating point* or *decimal* numbers can be printed in different formats as explained in the following table.

Table 1: Number Formats

Format	*Explanation*
short	Floating point fixed with five digits
long	Floating point fixed with 7 digits for single class and 15 digits for double class
short e	Floating point exponential with 5 digits – scientific notation
long e	Floating point exponential with 7 digits for single class and 15 digits for double class – scientific notation
short g	Default format. Best of fixed or exponential floating point with 5 digits
long g	Best of fixed or floating point with 7 digits for single class and 15 digits for double class
short eng	Floating point exponential with 5 digits – engineering notation
long eng	Floating point exponential with 7 digits for single class and 15 digits for double class – engineering notation
bank	Floating point fixed with two decimals

Formatting does not affect integers; they are printed as they are.

As it is mentioned above, default MATLAB format to display floating numbers is *short g*. To change to any other format, for example short eng, simply type *format short eng* and press enter. Format for each floating number from that point on will be set to short engineering. To return back to the default format, type *format* and press enter.

MATHEMATICAL OPERATORS

Addition, subtraction, multiplication, division, and power operations are represented by " + ", " - ", " * ", " / ", and " ^ " symbols respectively. Order in which MATLAB evaluates a mathematical expression is the standard order; power will be evaluated first, followed by the expression in parentheses ( ), then multiplication or division whichever comes first followed by addition or subtraction whichever comes first, from left to right. Note that square ([ ]) and curly ({ }) brackets are not allowed in expressions.

EXAMPLE 1

Evaluate the following expression in MATLAB

$$\frac{8\times 5\div 6}{7+9\div 8}\times 5+\frac{4\times 10^2}{9\times 12^3}$$

```
>> ((8*5/6)/(7+9/8))*5 + (4*10^2)/(9*12^3)

ans =

    4.1283
```

EXAMPLE 2

Evaluate the following expression for x = 27. Display your results in long engineering format

$$f(x)=\frac{x^3+2x^2+6x+120}{3x^2+5x+2}$$

```
>> format long eng
>> x = 27;
>> fx = (x^3 + 2*x^2 + 6*x + 120)/(3*x^2 + 5*x + 2)

fx =

    9.21815834767642e+000
```

Note that value(s) of the variable(s) for which you are evaluating an expression or equation should be defined before writing down the expression, as shown in the above example.

EXERCISE 1

Evaluate the following expression for $x = 4$, $y = 3$, and $z = 6$

$$f(x,y,z) = \frac{5xy + 6xz^2 - \dfrac{12y^2z^3}{31xyz}}{9x^3yz + xy^4z}$$

MATHEMATIC AND TRIGONOMETRIC FUNCTIONS

Functions are MATLAB scripts (m-files) that can be *called* from the command window or another program to do some specific job. There are two types of functions; MATLAB built-in or pre-written functions and user-defined or user-written functions. We are going to discuss user-defined functions later. Let's have a look at some of the MATLAB's mathematical and trigonometric functions.

Table 2: MATLAB'sMathematic and Trigonometric Functions

Function	*Explanation*
abs(x)	Computes the absolute value or magnitude of *x*
sqrt(x)	Computes the square root of *x*
round(x)	Rounds *x* to the nearest integer
floor(x)	Round *x* to the nearest integer towards -∞
ceil(x)	Round *x* to the nearest integer towards ∞
sign(x)	Returns -1 if *x*< 0, 0 if *x* = 0, and 1 if *x*> 0
exp(x)	Computes exponential value of *x*, i.e. e^x
log(x)	Computes natural logarithm of *x*, i.e. *ln(x)*
log10(x)	Computes base-10 logarithm of *x*
sin(x)	Computes sine of angle *x*, where *x* is in radians
sind(x)	Computes sine of angle *x*, where *x* is in degrees
cos(x)	Computes cosine of angle *x*, where *x* is in radians
cosd(x)	Computes cosine of angle *x*, where *x* is in degrees
tan(x)	Computes tangent of angle *x*, where *x* is in radians
tand(x)	Computes tangent of angle *x*, where *x* is in degrees
asin(x)	Computes arcsine or $\sin^{-1}$ of *x*. Resultant angle is in radians
asind(x)	Computes arcsine or $\sin^{-1}$ of *x*. Resultant angle is in degrees
acos(x)	Computes arccosine or $\cos^{-1}$ of *x*. Resultant angle is in radians
acosd(x)	Computes arccosine or $\cos^{-1}$ of *x*. Resultant angle is in degrees
atan(x)	Computes arctangent or $\tan^{-1}$ of *x*. Resultant angle is in radians
atand(x)	Computes arctangent or $\tan^{-1}$ of *x*. Resultant angle is in degrees
atan2(y,x)	Computes arctangent or $\tan^{-1}$ of *y/x*. Resultant angle is in radians and ranges from 0 to π if *x* and *y* are in the first two quadrants and from 0 to –π if *x* and *y* are in the last two quadrants

EXERCISE 2

Evaluate the following expression for $x = 25$ and $y = 2.34$, where x is in degrees and y is in radians

$$f(x,y) = \frac{2\sin(x) + 3\cos(4y) + 5\sin(y)\tan(3x)}{6\sin(x)\cos(y)}$$

EXAMPLE 3

Write a program that evaluates the following expressions,

$$y = \frac{2x\sin(x)\cos(4x)}{200\log_{10}(x)} + \sqrt{x^5\tan(4x)}\,;\; x = 2.45$$

$$z = \ln(y)e^x + \sin(x)\cos(y) + \sqrt{xy}$$

Script

```
%   This program will evaluate two equations sequentially

x = 2.45;

y = (2*x*sin(x)*cos(4*x))/(200*log10(x)) + sqrt(x^5*tan(4*x))

z = log(y)*exp(x) + sin(x)*cos(y) + sqrt(x*y)
```

Output

```
>> example3

y =

    5.8592


z =

   24.8585
```

NOTE: *In a script file, any statement followed by " % " is taken as 'comments'. It is ignored by MATLAB while executing the program.*

HELP

An extremely important command is *help*. If you want to check explanation of any command or function, type *help* followed by the function name and you will see all the relevant information about the function or command. Also, at the end of help documentation, it will show you some of the other closely related function.

MATHEMATICAL CONSTANTS & COMPELX FUNCTIONS

Some of the commonly used mathematical constants and complex functions are given in *table 3*

Table 3: Constants & Complex Number Functions

Constants/Functions	*Explanation*
pi	π
inf	∞
i , j	imaginary part of a complex number √-1
real(x)	Real part of a complex number *x*
imag(x)	Imaginary part of the complex number *x*
conj(x)	Complex conjugate of a complex number *x*
abs(x)	Magnitude of the complex number *x*
angle(x)	Angle corresponding to the complex number in radians

EXERCISE 3

Write a program that evaluates the following set of equations

$$V_1 = \frac{3\cos(x)\sin(4y) + e^{xy}}{\ln(x^2 y)}; x = 1.2, y = 2.3$$

$$V_2 = 5\tan(V_1)\sin(5xy)\log_{10}(xV_1)$$

$$V_3 = \frac{9\ln(xyV_1)}{15\log_{10}(V_1 V_2)}$$

VECTORS

In MATLAB, a variable containing several numbers divided in either a single row or a single column is called a *vector*. A *row vector* is a single row of numbers and a *column vector* is a single column of numbers. Hence *dimension* of a row vector is always 1 × *m*, where *m* is the number of columns and dimension of a column vector is *n* × 1, where *n* is the number of rows.

To define a vector, square brackets ([ ]) are used. For row vector, elements can either be separated by commas or simply blank spaces. For column vector, elements are separated by semi-colon.

```
>> A = [12 34 76 98 100]

A =

    12    34    76    98   100

>> B = [12; 34; 76; 98; 100]

B =

    12
    34
    76
    98
   100
```

To check dimension of a vector, type *size(vector)*. Dimension is always expressed in *rows* × *columns*. Also, to check length of a vector, i.e. total number of elements irrespective of row or column vector, type *length(vector)*

```
>> size(B)

ans =

     5     1

>> length(B)

ans =

     5
```

To transpose a vector, i.e. to convert a row vector into a column or vice-versa, simply place an *apostrophe* (‘) in front of it.

```
A =

    12    34    76    98   100

>> A'

ans =

    12
    34
    76
    98
   100


B =

    12
    34
    76
    98
   100

>> B'

ans =

    12    34    76    98   100
```

If a vector has its elements in a specific sequence, i.e. its elements have a constant interval between them, it can easily be defined in the following format:

first element:interval:last element

If interval is only one, first element and last element will be sufficient to define a vector

```
>> A = 2:3:15

A =

     2     5     8    11    14

>> B = 1:1:7

B =

     1     2     3     4     5     6     7

>> C = 1:7

C =

     1     2     3     4     5     6     7
```

MATHEMATICS OF VECTORS

To carry out addition, subtraction, multiplication, and division between the elements of two or more vectors, all vectors should have the same dimensions. Furthermore, to carry out multiplication and division between vectors of the same dimension, before the " * " or " / " sign, you have to place a period (.). Also, you have to place a period if you want to take power of each element of the vector.

```
A =

     1     3     5     7     9

>> B = 1:5

B =

     1     2     3     4     5

>> C = A + B

C =

     2     5     8    11    14

>> D = A.*B

D =

     1     6    15    28    45

>> E = B./A

E =

    1.0000    0.6667    0.6000    0.5714    0.5556

>> F = B.^2

F =

     1     4     9    16    25
```

EXAMPLE 4

Evaluate the following equation for the following vectors:

X = [2, 6, 9, 10, 5]; **Y** = [4, 7, 2, 1, 9]

$$z = 4x\cos(y) + 2\log(y)\ln(x) + \frac{e^{xy}}{e^{x^y}}$$

```
>> x = [2 6 9 10 5]; y = [4 7 2 1 9];
>> z = 4*x.*cos(y) + 2*log10(y).*log(x) + exp(x.*y)./exp(x.^y)

z =

   -4.3942   21.1221  -13.6584   22.6121  -15.1510
```

In *example 4*, observe how periods have been used to carry out multiplication, division, and power operations for vectors. Note that if you are multiplying a vector by a *scalar*, i.e. a single number, you do not have to use period before the operation.

EXERCISE 4

Write a script to evaluate the following equation,

$$w = \frac{xy}{4z} + \frac{3x^2y^3}{z^2} + \frac{4e^x}{yz\ln(z)}$$

given x = [0.2, 0.4, 0.6, 0.8, 1.0]; y = [2 4 6 8 10]; z = [1 3 5 7 9]

NOTE: Since first element of z is '1' and *ln(1)* = 0, hence your first answer should be ∞.

Inner or Dot Product

When a row vector is multiplied by a column vector, both with same length, it results in a scalar value. This type of vector multiplication is called *inner* or *dot* product

```
>> A = [2 3 6 8]; B = [5 7 9 10]';
>> C = A*B

C =

   165
```

Outer Product

When a column vector is multiplied by a row vector, both can be either same of different lengths, it results in a *matrix* with number of rows to be the same length as column vector and number of columns to be the same length as row vector. This is called *outer* product

```
>> A = [2 3 6 8]; B = [5 7 9 10]';
>> D = B*A

D =

    10    15    30    40
    14    21    42    56
    18    27    54    72
    20    30    60    80
```

Some of the commonly used functions related to vectors are given in *table 4*

Table 4: Functions to be used with Vectors

Function	*Explanation*
max(x)	Maximum value of vector *x*
min(x)	Minimum value of vector *x*
mean(x)	Calculates mean or average value of vector *x*
std(x)	Calculates standard deviation of vector *x*
var(x)	Calculates variance of vector *x*
median(x)	Calculates median of vector *x*
sort(x)	Sort vector *x* in ascending order
sort(x, 'descend')	Sort vector *x* in descending order
sum(x)	Calculates sum of the components of vector *x*
prod(x)	Calculates product of the components of vector *x*

It should be noted that to get the content of any position of a vector, you just have to type *vector(position number)*

```
A =

     2     3     6     8

>> A(3)

ans =

     6
```

MATRICES

A *Matrix* can have multiple rows and multiple columns. Hence, vector is a special case of matrix where there is either a single row or single column. Example of a 4×4 matrix is the outer product that we have observed earlier. All the functions that have been discussed in *table 2* can be used with matrices too. Those functions will operate on each of the column of matrix and generate result for each column. Hence each of the function will yield a row vector.

```
>> A = [3 4 8 7;9 2 3 8;1 2 8 10]

A =

     3     4     8     7
     9     2     3     8
     1     2     8    10

>> sort(A)

ans =

     1     2     3     7
     3     2     8     8
     9     4     8    10

>> prod(A)

ans =

    27    16   192   560

>> sum(A)

ans =

    13     8    19    25
```

Note that how matrix **A** is defined; semi-colon is used to go to the next row. Transpose can also be used with matrices to switch their rows and columns

```
>> A

A =

     3     4     8     7
     9     2     3     8
     1     2     8    10

>> A'

ans =

     3     9     1
     4     2     2
     8     3     8
     7     8    10
```

size and *length* commands can also be used with matrices to check their dimension (*rows* × *columns)* and length of either rows or columns, whichever is greater.

```
A =

     3     4     8     7
     9     2     3     8
     1     2     8    10

>> size(A)

ans =

     3     4

>> length(A)

ans =

     4
```

There are some functions that can generate special matrices as shown in *Table 5.*

Table 5: Some Special Matrices

Functions	*Explanation*
ones(n)	Generates a matrix of size *n*×*n* comprised of ones.
ones(n,m)	Generates a matrix of size *n*×*m* comprised of ones.
zeros(n)	Generates a matrix of size *n*×*n* comprised of zeroes.
zeros(n,m)	Generates a matrix of size *n*×*m* comprised of zeroes.
eye(n)	Generates an identity matrix of size *n*×*n*

Two very important functions that are used in conjunction with matrices are *inv(x)* and *det(x)*. *inv(x)* calculates the inverse of matrix *x* and *det(x)* calculates the determinant of matrix *x*. Remember that to calculate inverse and determinant of a matrix, it has to be a square matrix, i.e. a matrix with same number of rows and columns.

```
A =

    4.8538    4.2176    9.5949
    8.0028    9.1574    6.5574
    1.4189    7.9221    0.3571

>> inv(A)

ans =

   -0.1773    0.2714   -0.2193
    0.0235   -0.0433    0.1637
    0.1836   -0.1183    0.0390
```

To get the value of any element of a matrix, use *matrix(row#,column#)*

```
>> A = [23 45 78;34 56 78]

A =

    23    45    78
    34    56    78

>> A(2,3)

ans =

    78
```

To extract one row or one column from a matrix, use *matrix(:, column#)* or*matrix(row#,:)*

```
>> A(:,1)

ans =

    23
    34

>> A(1,:)

ans =

    23    45    78
```

To append a column or a row to an existing matrix, use *matrix(:, column #)*or *matrix(row#, :)*

```
A =

    23    45    78
    34    56    78

>> A(3,:) = [21 46 70]

A =

    23    45    78
    34    56    78
    21    46    70

>> A(:,4)=[56;1;2]

A =

    23    45    78    56
    34    56    78     1
    21    46    70     2
```

To delete a row or a column, use *matrix(row#,:)* = [] or *matrix(:,column#)* = [], where [] stands for *empty matrix* or *empty vector*.

```
A =

    23    45    78    56
    34    56    78     1
    21    46    70     2

>> A(:,3) = []

A =

    23    45    56
    34    56     1
    21    46     2

>> A(1,:) = []

A =

    34    56     1
    21    46     2
```

Addition, subtraction, multiplication, division and exponentiation operations for the elements of matrices are carried out exactly similar to the vectors. Matrices taking part in the operation should be of the same size, and periods will be required before multiplication, division and power operations.

Like outer product for vectors that produces a matrix, *matrix product* (**A**×B) between two matrices A and **B** produces a matrix with number of rows to be equal to the number of rows of **A** and number of columns to be equal to number of columns of **B**. Also, to carry out matrix product between two matrices **A** and **B**, number of columns of **A** should be equal to number of rows of **B.**

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

A =

     1     3     5
     3     7     8


B =

     2     4
     5     6
     7     8

>> % Matrix Product
>> A*B

ans =

    52    62
    97   118
```

```
>> C = [34 12 13;78 100 98]

C =

    34    12    13
    78   100    98

>> %Element-wise multiplication
>> A.*C

ans =

    34    36    65
   234   700   784
```

EXAMPLE 5

Use Cramer's rule to solve the following set of linear equations

$$5x + 7y = 10$$
$$4x + 2y = 3$$

Script

```
%   Cramer's Rule to solve the following set of linear equations
%
%   5x + 7y = 10
%   4x + 2y = 3

A = [5 7;4 2];
C = [10;3];

x = det([10 7;3 2])/det(A)

y = det([5 10;4 3])/det(A)
```

Output

```
x =

    0.0556


y =

    1.3889
```

EXERCISE 5

Any set of linear equations can be written and solved using matrices. For example, set of linear equations from *example 5* can be written as,

$$\begin{bmatrix} 5 & 7 \\ 4 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 10 \\ 3 \end{bmatrix}$$

which can be solved as follows,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 & 7 \\ 4 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 10 \\ 3 \end{bmatrix}$$

where $\mathbf{A}^{-1}$ represents inverse of matrix **A**.

Write a script file to solve the above set of linear equations using matrix inversion.

INPUT FUNCTION

If it is required from user to enter some data (numbers or string) to perform some operation, *input* is used. *input* is a built-in MATLAB function that asks user to enter numbers (scalar, vector, or matrix) or string to be processed by the program. It has the following format,

For numerical input:

A = input('text to be displayed for the user')

For string input:

A = input('text to be displayed for the user', 's')

User response will be stored in variable *A*, which can be scalar, vector, matrix or string.

```
>> A = input('Please enter a number between 5 and 10: ')
Please enter a number between 5 and 10: 7

A =

     7

>> B = input('What is your name? ', 's')
What is your name? Masood

B =

Masood
```

Note that any time during the text displayed for the user inside *input* argument, it is required to move to the next line, next line parameter " \n " can be used,

```
>> B = input('What is \nyour name? ', 's')
What is
your name? Masood

B =

Masood
```

EXAMPLE 6

Ask user to enter value(s) of the input variable *x* for which the following equation should be evaluated,

$$f(x) = \frac{3x\cos(x)}{\sin(5x)}$$

Script

```
%   Ask user to enter value of x, where x can be either scalar or vector,
%   and evaluate the following equation
%
%   f(x) = 3xcos(x)/sin(5x)
%
%
x = input('Please input variable x for which "3xcos(x)/sin(5x)" should be evaluated: ');
fx = 3*x.*cos(x)./sin(5*x)
```

Output

```
>> example6
Please input variable x for which "3xcos(x)/sin(5x)" should be evaluated: 1:5

fx =

   -1.6903    4.5897  -13.7015   -8.5917  -32.1487
```

EXERCISE 6

Ask user to enter the coefficients of a quadratic equation, $Ax^2 + Bx + C = 0$, i.e. *A*, *B*, and *C*, and calculate the roots of the equation using the quadratic formula,

$$x_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

EXERCISE 7

Write a program to solve a system of linear equations,

$$A_{11}x_1 + A_{12}x_2 + A_{13}x_3 + \cdots\cdots + A_{1n}x_n = c_1$$
$$A_{21}x_1 + A_{22}x_2 + A_{23}x_3 + \cdots\cdots + A_{2n}x_n = c_2$$
$$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$$
$$A_{n1}x_1 + A_{n2}x_2 + A_{n3}x_3 + \cdots\cdots + A_{nn}x_n = c_n$$

using matrix inversion. Ask user to enter coefficient matrix **A** and constant vector **c** and calculate the value of unknown variables x_p, where p = 1, 2, 3, ….., n. Check *exercise 5* for further help.

OUTPUT FUNCTIONS

There are two functions used to output results; *disp* and *fprintf*.

disp is a very simple function. Anything written as *disp(x)*, where *x* can be scalar, vector, matrix, or string of characters, will be printed in the command window even if you put semi-colon after the command.

Note that you can display any value, numbers or string, just by putting the variable name and pressing enter. The difference between that and using *disp* is that *disp* does not show the variable name, only its value.

```
>> disp('My name is Masood Ejaz');
My name is Masood Ejaz
>> disp(A)
     2     3     5     7
     1     8     9     2
     0     2     8     5
    10     2     8     4
```

An extremely versatile output function is *fprintf*. *fprintf* writes formatted data to a file or command window. *fprintf* is used to print both text and variable values. The format to be used in printing the values can also be specified. It can also be defined to print values or text in the new line and if values should have a tab space between them. Hence *fprintf* gives a lot more power to print output as against *disp*. Format of *fprintf* is as follows,

fprintf('text, variables' format', variables)

Under *fprintf* argument, *text* is the text you want to write. Anywhere in the text you can insert *format* of variables to be printed. This format can be given by specifiers *%f, %d, %e, %g, or %s*, such that variables will be displayed in floating, integer, exponential, floating or exponential, whichever fits best, or string of characters, respectively. *variables* are the variables to be printed

using the format defined in text, in sequence, separated by commas. Under *text*, "\n" can be used anywhere to go to the new line. Also "%t" can be used to insert tab inside text.

```
>> A = 23.456; B = 1e3;
>> fprintf('Value of A is %f and value of B is %e\n',A,B)
Value of A is 23.456000 and value of B is 1.000000e+003
>> fprintf('Value of A is %f \nand value of B is %e\n',A,B)
Value of A is 23.456000
and value of B is 1.000000e+003
>>
```

Observe the effect of "\n" in the above outputs. Formatting can also be restricted to certain number of significant figures. If %*f* is modified to %0.*nf*, where *n* is an integer then output will be printed with *n* significant figures after decimal

```
>> fprintf('Value of A is %f and value of B is %0.2f\n',A,B)
Value of A is 23.456000 and value of B is 1000.00
```

EXAMPLE 7

Ask user to enter scores for test # 1, test # 2, and test # 3 for three students. Calculate their average scores and variance of scores and print out an appropriate message regarding average and variance for each student.

Script

```
%   Example 7 - MATLAB Tutorial
%

clc;
score1 = input('Enter scores of tests 1 to 3 for student # 1 in a vector form: ');
score2 = input('Enter scores of tests 1 to 3 for student # 2 in a vector form: ');
score3 = input('Enter scores of tests 1 to 3 for student # 3 in a vector form: ');

avg_score1 = mean(score1); %    average score of student # 1
avg_score2 = mean(score2); %    average score of student # 2
avg_score3 = mean(score3); %    average score of student # 3

var_score1 = var(score1); %    variance of test scores of student # 1
var_score2 = var(score2); %    variance of test scores of student # 2
var_score3 = var(score3); %    variance of test scores of student # 3

fprintf('Average score of student # 1 from three tests is %0.2f and variance of test scores is %0.2f\n',avg_score1, var_score1);
fprintf('Average score of student # 2 from three tests is %0.2f and variance of test scores is %0.2f\n',avg_score2, var_score2);
fprintf('Average score of student # 3 from three tests is %0.2f and variance of test scores is %0.2f\n',avg_score3, var_score3);

```

Sample Output

```
Enter scores of tests 1 to 3 for student # 1 in a vector form: [100 80 75]
Enter scores of tests 1 to 3 for student # 2 in a vector form: [90 88 76]
Enter scores of tests 1 to 3 for student # 3 in a vector form: [80 70 95]
Average score of student # 1 from three tests is 85.00 and variance of test scores is 175.00
Average score of student # 2 from three tests is 84.67 and variance of test scores is 57.33
Average score of student # 3 from three tests is 81.67 and variance of test scores is 158.33
```

Observe that *line 4* of the script file from *example 7*has a command "*clc*". *clc* stands for *clear command window* and this command erases everything from the command window.

EXERCISE 8

Ask user to enter values of some resistors. Calculate the value of equivalent resistor if all resistances are connected in series and if all are connected in parallel using the following formulae

Series: $R_{eq} = R_1 + R_2 + R_3 + \ldots\ldots\ldots\ldots + R_n$

Parallel: $R_{eq} = \dfrac{1}{\frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_3}+\ldots\ldots\frac{1}{R_n}}$

Print both the values with the following message:

'If all the resistors are connected in series, equivalent resistor will be x and if all the resistors are connected in parallel, equivalent resistor will be y'

where *x* is the series equivalent value and *y* is the parallel equivalent value

Hint: MATLAB function 'sum' should be helpful.

EXERCISE 9

Include *fprintf* in *exercise 6* to print the roots of the quadratic equations with the following message:

Roots of the quadratic equation $Ax^2+Bx+C = 0$ *are Root # 1 & Root # 2*

where *Root # 1* and *Root # 2* are the first and second calculated roots of the quadratic equation.

Sample Output

```
Enter the cofficients of a quadratic equation in a vector form: [1 5 10]
Roots of the quadratic equation 1x^2 + 5x + 10 = 0 are -2.50+j1.94 and -2.50-j1.94
```

NOTE: A good approach to print out complex numbers is first to convert them in a string using *num2str()* function and then print them as a string.

PLOTS

There are several different types of plotting functions in MATLAB; the most common one is *plot*. *plot* is used to make an *x*-*y* graph, whether *x* and *y* are entered as discrete vectors or *y* is a function of *x* evaluated through some expression. If you are creating discrete *x* and *y* vectors (for example, data collected from some lab experiment), you have to make sure that both vectors should have the same length. When *y* is a function of *x* evaluated through some expression for specific values of *x*, their length will always be the same. Syntax to use plot command is *plot(x,y)*.

```
>> voltage = [0,1.2,2.3,4.4,5.6,7.9,9];
>> current = [0,3.2e-3,5.5e-3,7e-3,8.9e-3,10e-3,13.2e-3];
>> plot(voltage,current)
```

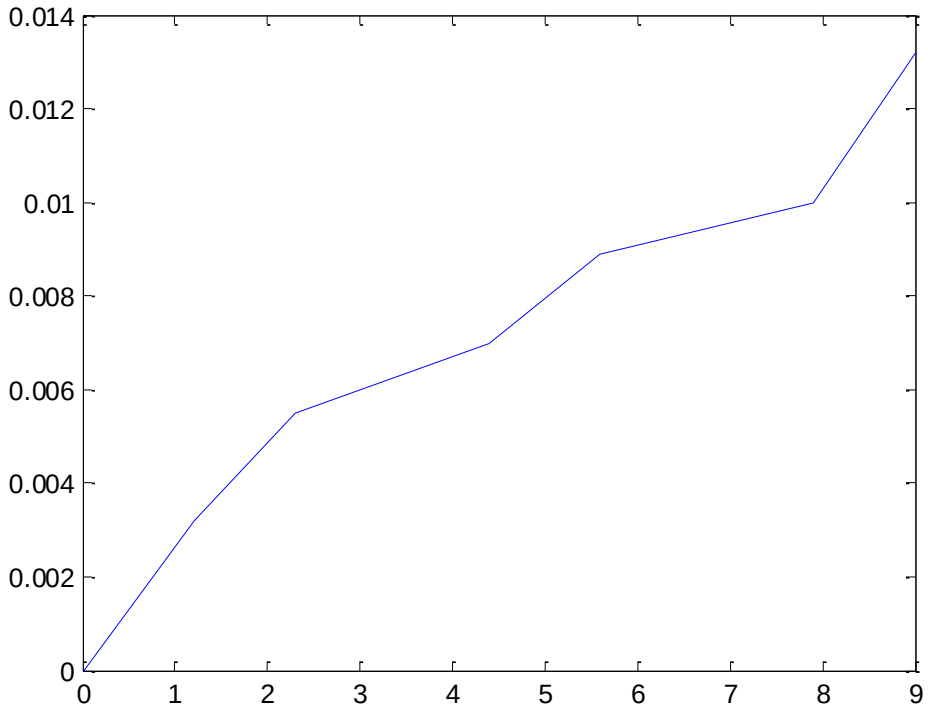


Figure 2: Plot of Current vs. Voltage

Most of the times, plots are labeled with the information about *x-axis*, *y-axis*, and a suitable title. *Legends* can also be used specially if there are multiple plots in the same figure. A *grid* can also

be placed on the plot to present the information in a better way. Observe how *figure # 2* can be accentuated with labels and a grid.

```
>> current = [0,3.2,5.5,7,8.9,10,13.2];
>> plot(voltage,current), xlabel('Volt'), ylabel('milliampere'), title('Current vs. Voltage for Circuit # 3'), grid
```

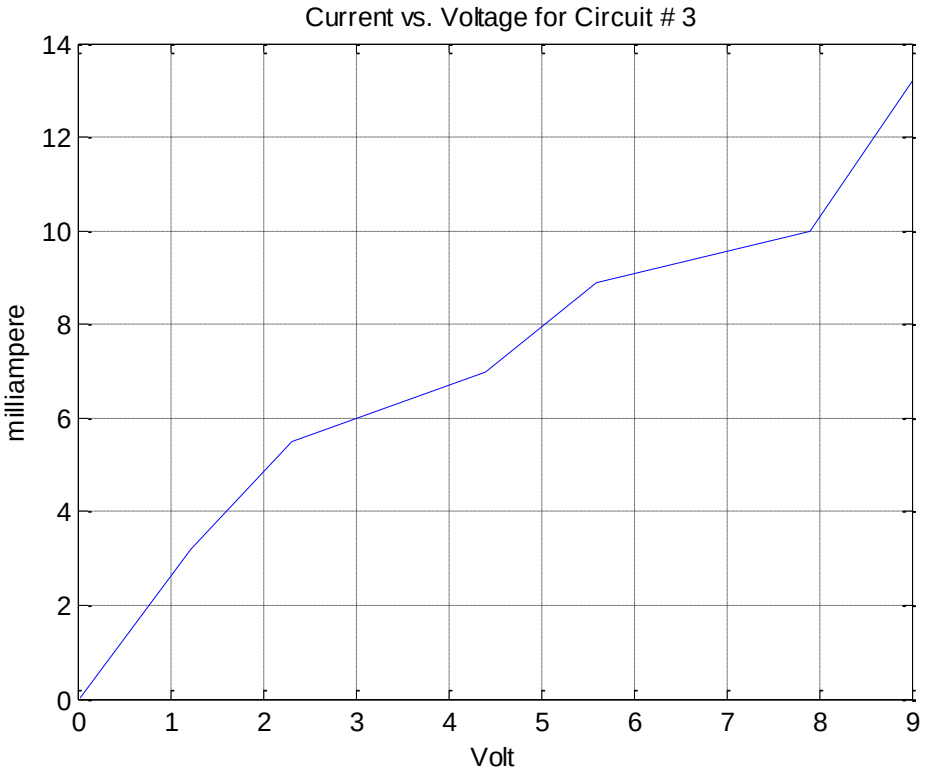


Figure 3: Plot with Labels

Note that since only current vector is being changed from *milli* (e-3) as being part of the values, to just the magnitude of the current without any exponent, as *milliampere* is now the label associated with the *y-axis*; hence, only current vector is to be recreated. Voltage vector is still the same so no need to rewrite it.

Plotting line colors and styles can also be changed by adding a third field after *plot(x,y); plot(x,y,'colorstyle')*, where *color* stands for desired line color and *style* stands for desired line style. Permissible line colors and styles can be found by checking help for the *plot* function. Note that if you just want to change color of the line, do not put any style, and if you just want to change style of the plot without changing the default color (blue), do not put anything for color, just choose a desired style.

EXAMPLE 8

Create a plot in *green* for the following voltage as a function of time.

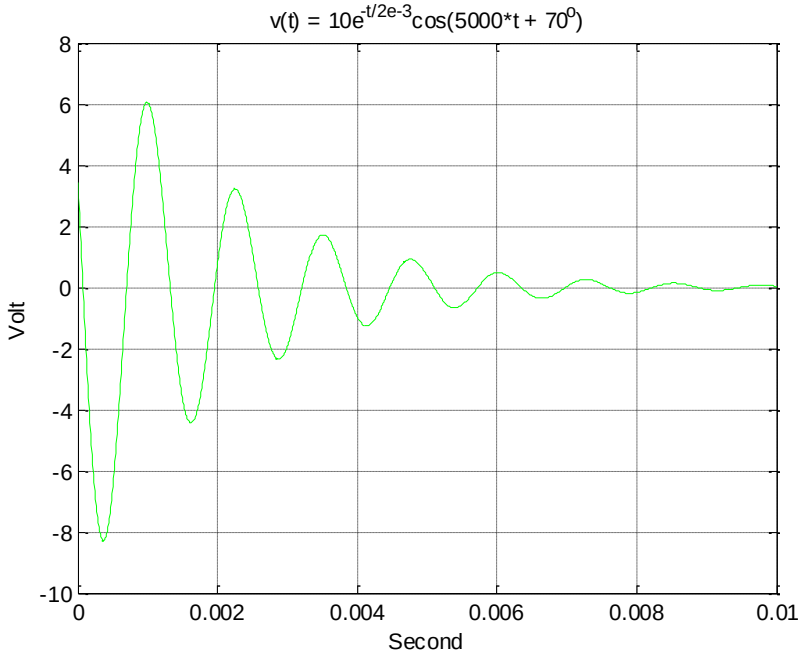
$$v(t) = 10e^{-\frac{t}{2e-3}}\cos(5000t + 70^0)\ V$$

NOTE: Cosine argument is a combination of radians and degrees; '*5000t*' is in radians and '70^o' is in degrees. If *cos(x)* is used, 70^o has to be converted into radians, and if *cosd* is used, *5000t* has to be converted into degrees.

Script

```
%   Masood Ejaz - Example 8 - MATLAB Tutorial

t = 0:0.01e-3:10e-3;    %   time vector

%   voltage function; observe that 70 degrees is converted into radians
v = 10*exp(-t/2e-3).*cos(5000*t + 70*pi/180);

plot(t,v,'g'), xlabel('Second'), ylabel('Volt'), title('v(t) = 10e^-^t^/^2^e^-^3cos(5000*t + 70^o)'),grid
```

Output



Observe the *superscript* text in the title. Under *xlabel*, *ylabel* and *title*, any character right after '^' will be displayed as superscript, and any character right after '_' will be displayed as subscript. For multiple characters in superscript or subscript, you have to use '^' or '_' that many times as shown in the script file for the displayed text.

MULTIPLE PLOTS

There are three different ways to have multiple plots; plots in different figures or windows, multiple plots in the same figure or window, and dividing a figure into sub-figures and have plots in them.

Each plot window or *figure* is associated with a figure number. To plot *different plots in different figures*, all you have to do is to mention in which figure number the specific plot should be created, as shown in the following example.

EXAMPLE 9

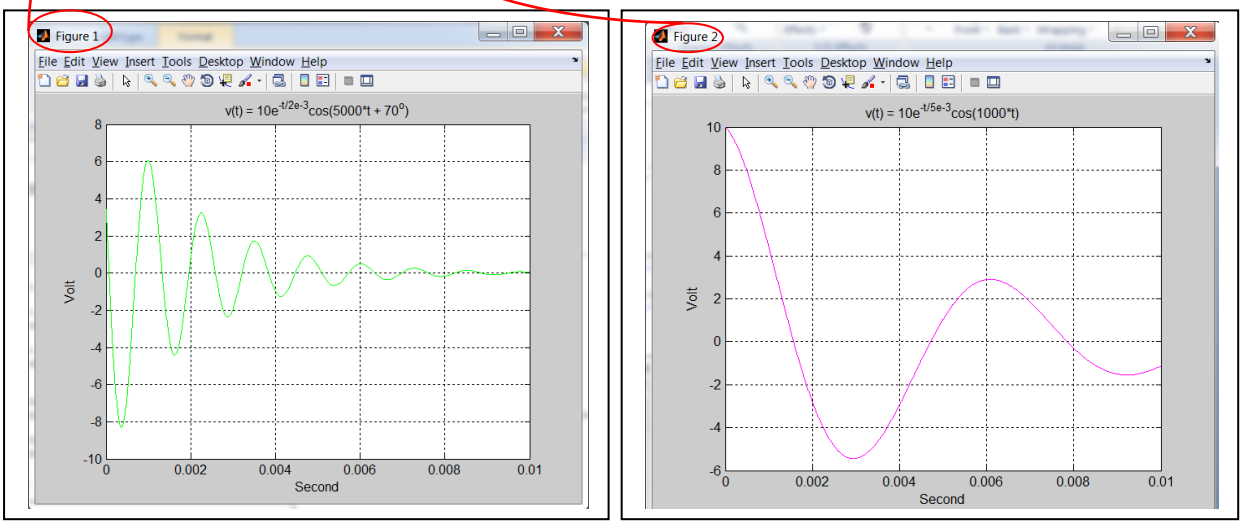
Plot another voltage function vs. time in addition to the one given in *example 8* in two different figures. New voltage function is given as follows.

$$v(t) = 10e^{-\frac{t}{5e-3}}\cos(1000t)V$$

Script

```
%   Masood Ejaz - Example 9 - MATLAB Tutorial

t = 0:0.01e-3:10e-3;    %   time vector

%   First voltage function; observe that 70 degrees is converted into radians
v1 = 10*exp(-t/2e-3).*cos(5000*t + 70*pi/180);

%   Second Voltage function
v2 = 10*exp(-t/5e-3).*cos(1000*t);


figure(1)   %   plot the first function in figure(1)
plot(t,v1,'g'), xlabel('Second'), ylabel('Volt'), title('v(t) = 10e^-^t^/^2^e^-^3cos(5000*t + 70^o)'),grid

figure(2)   %   plot the second function in figure(2)
plot(t,v2,'m'), xlabel('Second'), ylabel('Volt'), title('v(t) = 10e^-^t^/^5^e^-^3cos(1000*t)'),grid
```

Output



EXERCISE 10

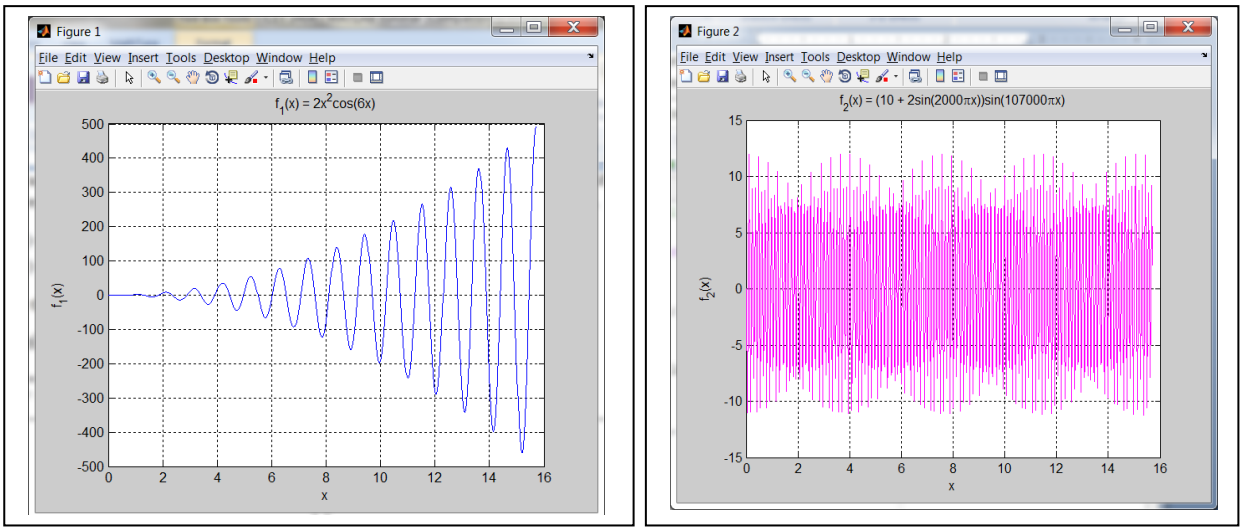
Ask user to enter vector *x* to evaluate and plot the following two functions in two different figures. Choose different colors for each of the plots and choose suitable labels for them.

$$f_1(x) = 2x^2\cos(6x)$$
$$f_2(x) = (10 + 2\sin(2000\pi x))\sin(107000\pi x)$$

Note that argument of trigonometric functions is in radians.

Sample Output

```
>> exercise10
Enter a vector x to evaluate and plot f1(x) and f2(x): 0:pi/100:5*pi
```



Observe that title of *figure 2* has Greek letter π. To print Greek letters, you have to place *\Greekletter* in the text of labels, where *Greekletter* is the specific letter you want to print. For example, in this case \pi should be used

To plot *several plots in the same figure*, plot function has to be extended for all the *x* and *y* associated with different plots; *plot*(x_1,y_1,x_2,y_2, …, x_n, y_n). Each plot will have a different color but if you want any specific color or style, you can add color and style field for any set of *x* and *y*. Also, a legend is usually used to specify some property or text for each plot in the figure. Legend can be printed in the figure by using *legend('property 1', 'property 2 ', ..., 'property n')*, where *property 1* is associated with the plot of (x_1, y_1), *property 2* is associated with the plot of (x_2, y_2), and so on.

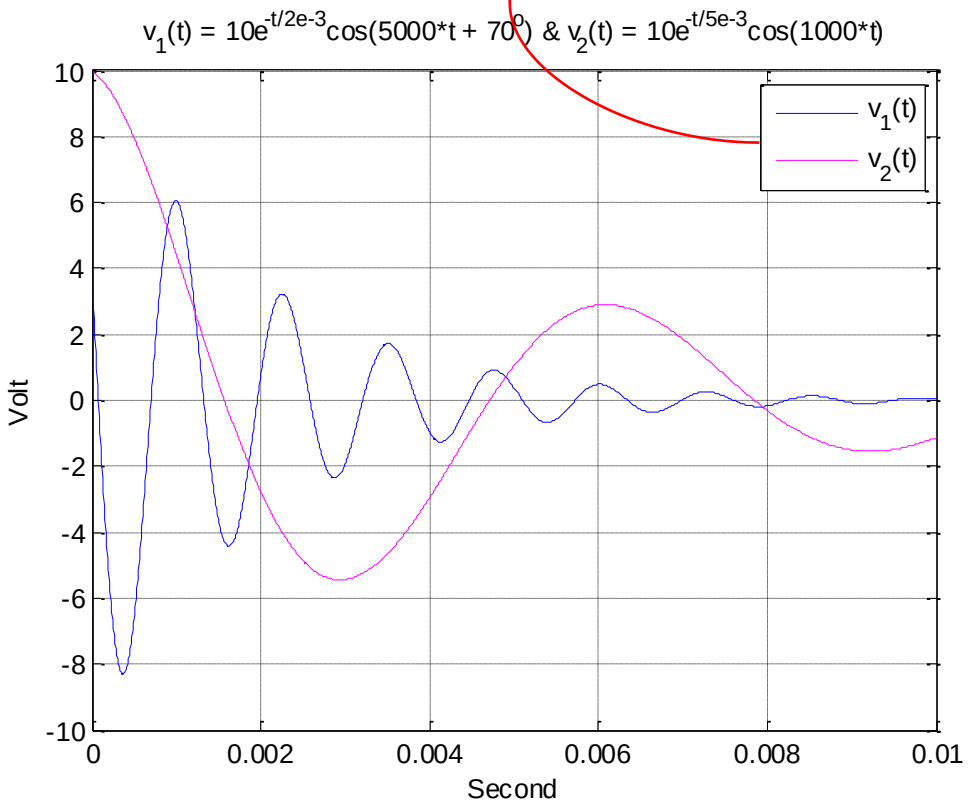
EXAMPLE 10

Plot both the functions given in *Example 9* in the same figure.

Script

```
%   Masood Ejaz - Example 10 - MATLAB Tutorial

t = 0:0.01e-3:10e-3;    %   time vector

%   First voltage function; observe that 70 degrees is converted into radians
v1 = 10*exp(-t/2e-3).*cos(5000*t + 70*pi/180);

%   Second Voltage function
v2 = 10*exp(-t/5e-3).*cos(1000*t);

plot(t,v1,t,v2, 'm'), xlabel('Second'), ylabel('Volt'), title('v_1(t) = 10e^-^t^/^2^e^-^3cos(5000*t + 70^o) & v_2(t)
grid, legend('v_1(t)', 'v_2(t)')
```

Output



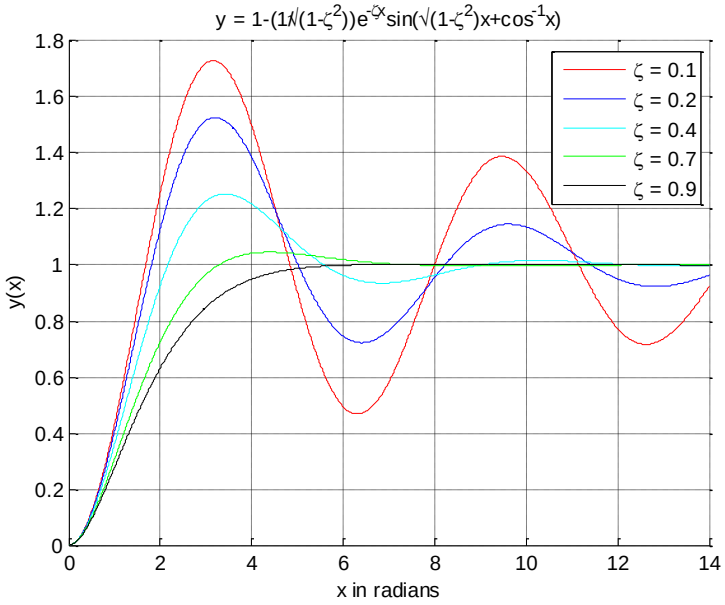
EXERCISE 11

Create plots for the *second-order* control system, in the same figure, given by the following expression for different values of the *damping ratio*ζ (zeta),

$$y = 1 - \frac{1}{\sqrt{1-\zeta^2}} e^{-\zeta x} \sin(\sqrt{1-\zeta^2}\,x + \cos^{-1}(\zeta))$$

Use five different values for ζ; 0.1, 0.2, 0.4, 0.7, 0.9. Also, ask user to enter vector *x* that represents range (*x-axis)* over which plots will be created.

Sample Output



Hint: Create five vectors; each corresponding to a specific value of ζ

Sub-Plots

To divide one figure into different windows and plot one or more plots in each of them is done using *subplot* function. Syntax of the subplot is as follows:

subplot(m,n,p) or *subplot(mnp)* divides the figure into *m* x *n* matrix of small axes and selects the *p*-th axis for the current plot. The axes are counted along the top row of the figure window, then the second row, and so on.

EXAMPLE 11

Plot all of the following functions in a single figure using subplots. Three subplots with one function each and fourth one with all of them.

$$f_1(x) = (5x^2 + 6x + 9)\cos(5x)$$
$$f_2(x) = (4x^2 + 9x + 10)\sin(7x + 19^\circ)$$
$$f_3(x) = (x^3 - x^2 + x)\cos(8x + \frac{\pi}{6})$$

Ask user to enter vector *x*.

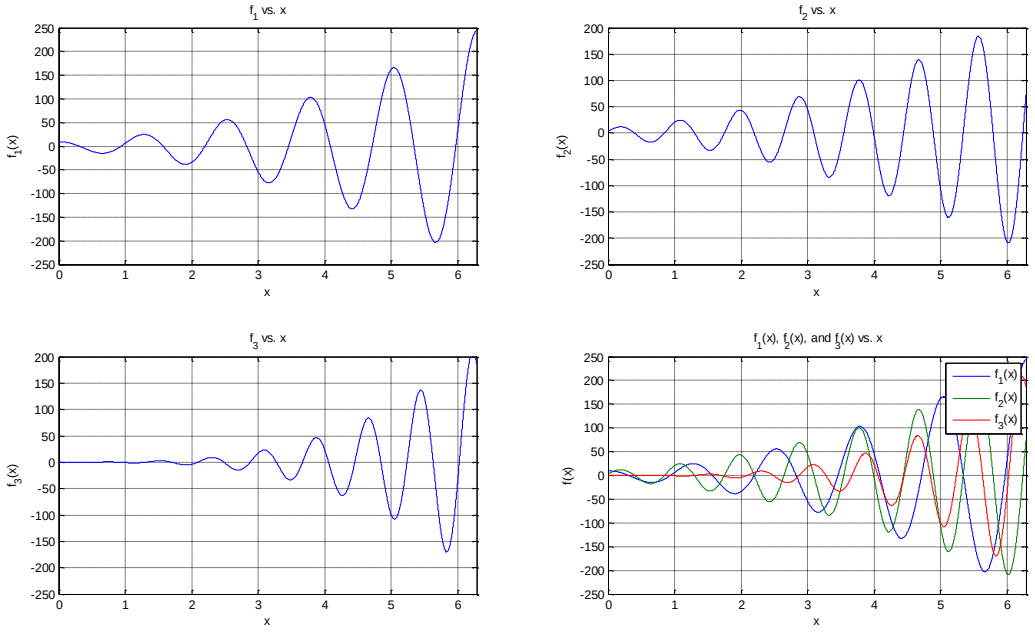
Script

```
%   Masood Ejaz - Example 11 - MATLAB Tutorial

x = input('Enter vector x against which functions will be plotted: ');

f1 = (5*x.^2 + 6*x + 9).*cos(5*x);
f2 = (4*x.^2 + 9*x + 10).*sin(7*x + 19*pi/180);
f3 = (x.^3 - x.^2 + x).*cos(8*x + pi/6);

%   subplot in row 1, column 1 window
subplot(2,2,1), plot(x,f1), xlabel('x'), ylabel('f_1(x)'), title('f_1 vs. x'), grid
axis([0,2*pi,-250,250])

%   subplot in row 1, column 2 window
subplot(2,2,2), plot(x,f2), xlabel('x'), ylabel('f_2(x)'), title('f_2 vs. x'), grid
axis([0,2*pi,-250,200])

%   subplot in row 2, column 1 window
subplot(2,2,3), plot(x,f3), xlabel('x'), ylabel('f_3(x)'), title('f_3 vs. x'), grid
axis([0,2*pi,-250,200])

%   subplot in row 2, column 2 window
subplot(2,2,4), plot(x,f1,x,f2,x,f3), xlabel('x'), ylabel('f(x)'), title('f_1(x), f_2(x), and f_3(x) vs. x')
grid, legend('f_1(x)', 'f_2(x)', 'f_3(x)'), axis([0,2*pi,-250,250])
```

Output

```
>> example11
Enter vector x against which functions will be plotted: 0:0.001:2*pi
```

(*continued on the next page*)



Observe the encircled commands in the script; *axis([x_{min}, x_{max}, y_{min}, y_{max}])*. This function is used to set axes of the figure according to the required range (if default range is not appropriate). x_{min} and x_{max} are the minimum and maximum values of the *x-axis*, respectively, and y_{min} and y_{max} are the minimum and maximum values of the *y-axis*, respectively.

EXERCISE 12

Plot the function *y(x, ζ)* from *Exercise 11* for ζ = 0.3, 0.5, and 0.8. Create one figure with four subplots and plot *y* for three different values of ζ in three subplots and fourth subplot should have multiple plots of *y* for all the values of ζ.

3-D Plots

There are two common types of 3-D plots in MATLAB; *line 3-D plots* and *surface 3-D plots* also called *mesh plots. line 3-D plots* create a function *f(x,y)* in 3-D with a simple line whereas *mesh* plots create a surface in 3-D from function *f(x,y)*.

Syntax to create line 3-D plots is simply *plot3(x,y,z)*, where *z* is a function of *x* and *y* or *x*, *y*, and *z* are three vectors of same length. Syntax to create mesh plots is *mesh(X,Y,Z)* where *X*, *Y, and Z* are not the same as *x*, *y*, and *z* vectors, rather they are matrices and together they represent all the possible combinations of the points from *x* and *y* vectors. Matrices *X* and *Y* are created from vectors *x* and *y* using *meshgrid* function;

[X, Y] = meshgrid(x,y)

Once *X* and *Y* matrices are created, function *Z* will be evaluated according to the given expression using *X* and *Y*. *Example 12 and 13* shows 3-D plots using both the functions just discussed.

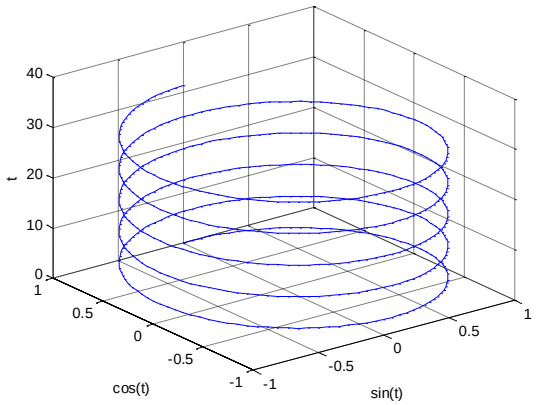
EXAMPLE 12

Plot a line 3-D plot between three vectors given by,

$$x = \sin(t)$$
$$y = \cos(t)$$
$$z = t$$

where $t = 0 : \frac{\pi}{50} : 10\pi$

Output



Script

```
%   Masood Ejaz - Example 12 - MATLAB Tutorial

t = 0:pi/50:10*pi;

x = sin(t); y = cos(t); z = t;

plot3(x,y,z), xlabel('sin(t)'), ylabel('cos(t)'), zlabel('t'), grid
```

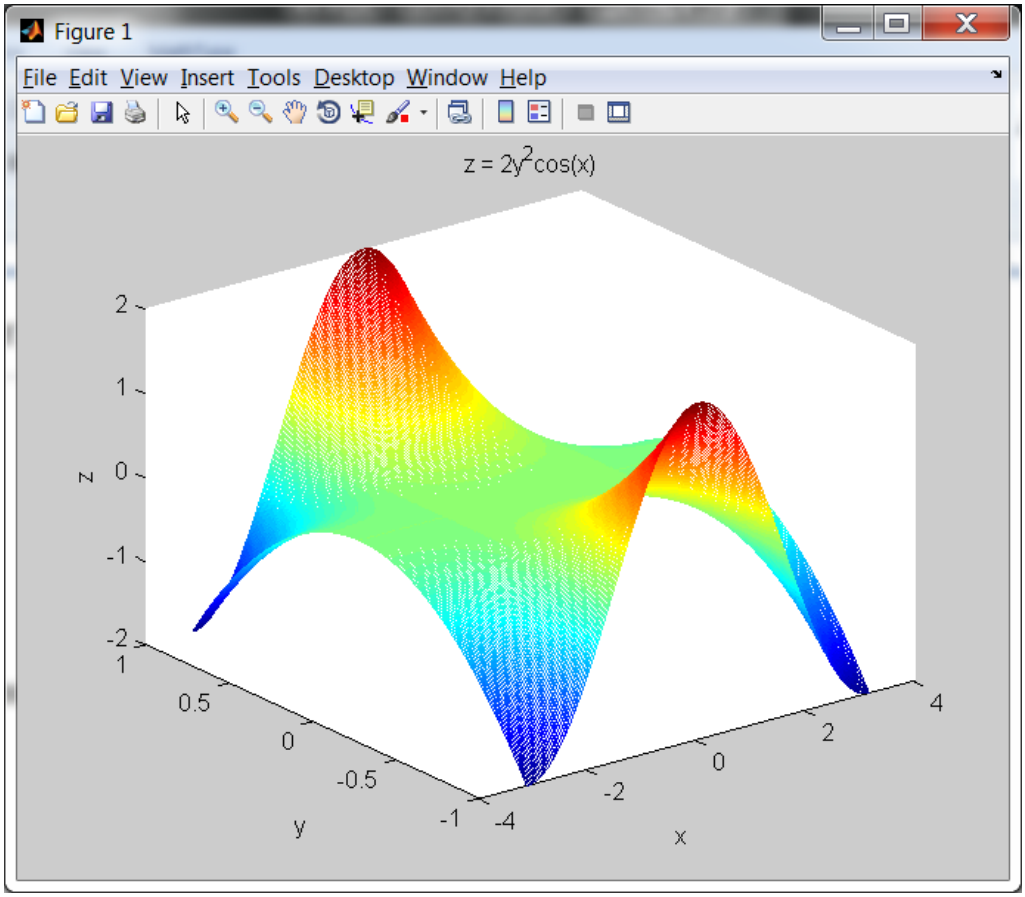
EXAMPLE 13

Create a mesh plot for $z = 2y^2cos(x)$ for x = -π:π/100:π, and y = -1:0.01:1

Script

```
%   Masood Ejaz - Example 13 - MATLAB Tutorial

x = -pi:pi/100:pi;
y = -1:0.01:1;

[X,Y] = meshgrid(x,y); %    Creating matrices X and Y
Z = 2*Y.^2.*cos(X);

mesh(X,Y,Z), xlabel('x'), ylabel('y'), zlabel('z'), title('z = 2y^2cos(x)'), grid
```

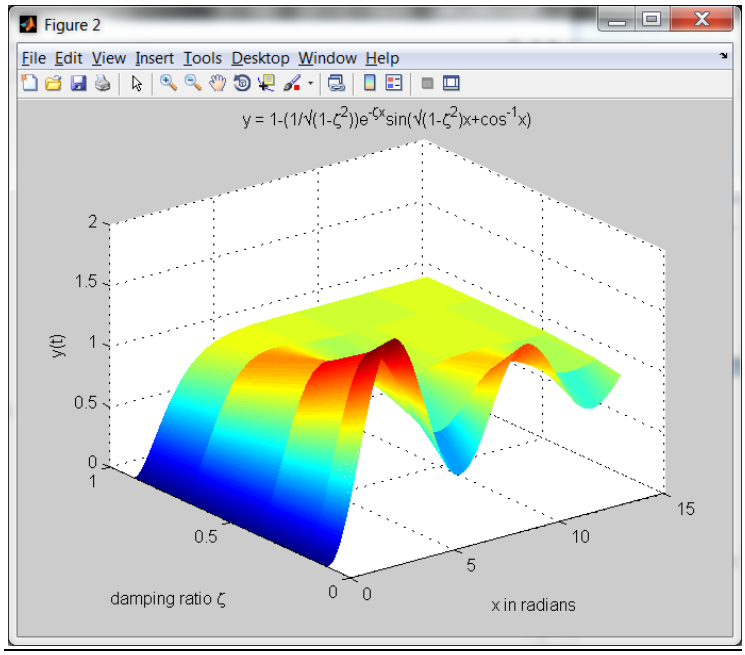
Output



EXERCISE 13

For the function *y* given in *Exercise 11*, create a mesh plot of *y* as a function of ζ and *x*. Use the same values of ζ as given in *Exercise 11* and use *x* = 0:0.01:14 radians

Sample Output



USER-DEFINED FUNCTIONS

So far you have used MATLAB built-in functions (*fprintf, clc, plot, length, size etc.)*, now it is time to write your own functions. *Functions* are programs that can be called from the command window or from any other program. Difference between writing functions and any other program is that functions cannot be executed, they can only be called. Make sure never execute or run your function from the script file.

There are two types of variables or set of variables associated with every function. These variables are called arguments; *input* arguments and *output* arguments. *Input* arguments are variables that are associated with the execution of an equation, formula or a program, and *output* arguments are the variables that hold the results after your program (function) is executed. The format to write your function is to start your program with the following syntax, i.e. the first line of the program should have the format,

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

Note that function always starts with the word *function.* If there is only a single output argument (scalar, vector, or matrix), there is no need for the square brackets. If there are multiple output arguments, square brackets are required. Parentheses are required to give input variables whether there is a single input argument or multiple arguments. If there are multiple input and output arguments, they have to be separated by commas.

Functions are always saved by the same file name as you give to the function. If you save your function by any other name that is different from your function name, you will not be able to call it. Once you write a function in MATLAB editor and try to save it, it will give you the option of saving it by the function name by default. Do not change it! Make sure not to give any name to your function that is the same as any built-in function of MATLAB, especially most commonly used functions (*length, max, min, size, fprintf* etc.)

One of the main differences between a *function* and a regular program is that you do not ask user to enter something using *input* function; input is always given through input arguments. Also, most of the time you do not print out the output value from the function; it is just assigned to the output arguments. In addition to that, never use *clear* at the beginning of any function else it will clear all the input arguments that are passed to the function when it is called, and it will give an error.

If some piece of program or some equation or expression is used over and over again to solve something, it is always wise to create a function out of it. Majority of the MATLAB functions are created by different users from all around the world who are experts in their fields.

EXAMPLE 14

Write a function that evaluates $y = 4sin(x) + 5cos(6x) + 2x^2$ for any input variable x. x can either be a scalar, vector or matrix.

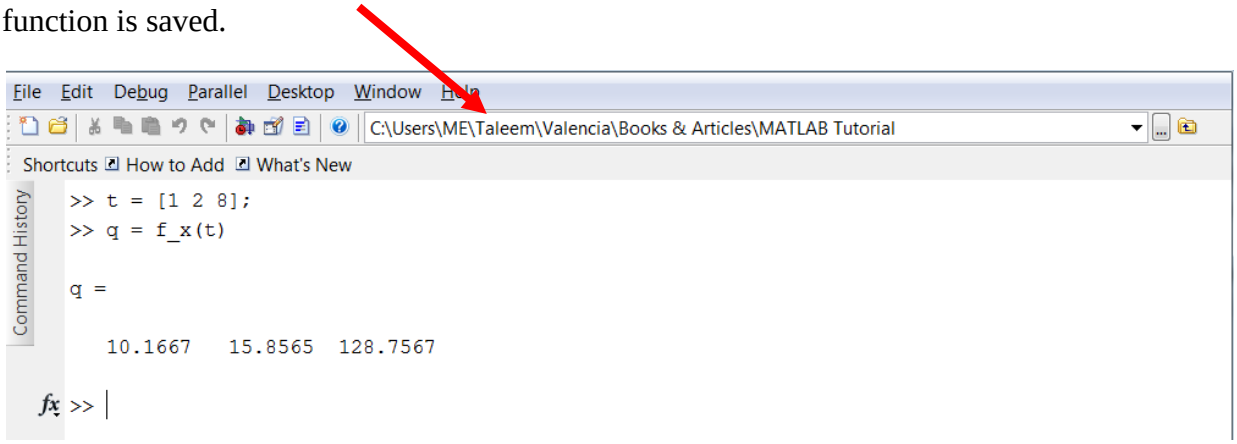
Script

```
1   function y = f_x(x)
2   %
3   %   format: y = f_x(x)
4   %
5   %   This function evaluates the expressions 4sin(x) + 5cos(6x) + 2x^2
6   %
7   %   Written by Masood Ejaz - October 12, 2013 - Example 14
8   %
9
10  y = 4*sin(x) + 5*cos(6*x) + 2*x.^2;
```

Observe that the first line of the program starts with the word *function*. *y* is the output argument and since there is only one output argument, there is no need for the square brackets. Function name is *f_x* and input argument is *x*. After the first line that tells MATLAB that this is not a regular program rather this is a function, you can write help for your function as comments. Any time someone wants to check help for your function with the *help* command [*help f_x*], it will show whatever you have written as comments after the first line. After the help comments, required expression is evaluated and result is assigned to the output argument *y*. Make sure you always have to assign results to the output argument(s) else you will get an error when you will call the function.

Output

Make sure that your current path for the command window is the same as the folder where the function is saved.



Note that you can assign any variable as output argument and input argument; it does not necessarily be the same that you used in the function. You can even write down the input argument value(s) without assigning it to any variable as shown below.

```
>> p = f_x([4 9 8 7;2 4 0 1])

p =

   31.0937  159.5019  128.7567   98.6280
   15.8565   31.0937    5.0000   10.1667
```

Description of the Function

```
>> help f_x
    format: y = f_x(x)

    This function evaluates the expressions 4sin(x) + 5cos(6x) + 2x^2

    Written by Masood Ejaz - October 12, 2013 - Example 14
```

EXAMPLE 15

Create a function that converts a complex number given in the Cartesian form $x+jy$ into polar form, $r\angle\theta^o$.

Script

```
function [r,t] = rect_to_pol(a)
%
%   syntax: function [r,t] = rect_to_pol(a)
%
%   This function converts a complex number given in the cartesian form
%   into polar form. Output arguments are 'r' for the magnitude and 't' for
%   the angle in degrees
%
%   Created by Masood Ejaz - October 12, 2013 - Example 15
%
real_a = real(a);   %   real part of the input complex number
imag_a = imag(a);   %   imaginary part of the input complex number

[q,w] = cart2pol(real_a, imag_a);   %   converting cartesian number into polar form using built-in MATLAB function
                                    %   Note that 'q' is angle in radians
                                    %   and 'w' is the magnitude
r = w;  %   assigning magnitude to the output variable
t = q*180/pi;   %   assigning angle to the output variable. Since angle required is in degrees,
                %   hence it is converted from radian to degrees first.

```

Sample Output

```
>> [r, t] = rect_to_pol(5-7i)

r =

    8.6023


t =

  -54.4623

>> help rect_to_pol
    syntax: function [r,t] = rect_to_pol(a)

    This function converts a complex number given in the cartesian form
    into polar form. Output arguments are 'r' for the magnitude and 't' for
    the angle in degrees

    Created by Masood Ejaz - October 12, 2013 - Example 15


>>
```

EXERCISE 14

Write a function *Your first name_Last name_Ex14* with input arguments to be the magnitude and angle (in degrees) of a complex number and output argument to be the Cartesian form of the same number, i.e. *x+yi*.

[*Hint: You can use 'pol2cart' function as part of your program. Check help for pol2cart function to become familiar with it*]

Sample Output

```
>> x = Masood_Ejaz_Ex14(5,63)

x =

   2.2700 + 4.4550i
```

EXERCISE 15

Write a function *Your first name_Last name_Ex15* with three variables *x*, *y*, and *z* as input arguments. These variables can either be scalars, vectors (of same length), or matrices (of same dimensions). Calculate the following three equations and assign the results to the output arguments.

$$s = 2\cos(x) + \frac{2y^3}{4e^{-z}}$$

$$t = 3x^2 + 5\sin(6s) + yz/x$$

$$u = 4st + 0.5xyz$$

Sample Output

```
>> [s, t, u] = Masood_Ejaz_Ex15(1,2,3)

s =

   81.4228


t =

    4.0009


u =

   1.3061e+03
```

RELATIONAL EXPRESSIONS

Relational expressions represent the relationship or comparison between two quantities. Output of the comparison is either '1', if comparison is true, or '0', if it is false. Following table outlines all of the comparators that can be used in MATLAB.

Comparator	*MATLAB symbol*
Equal	== (two equal signs together)
Not equal	~=
Less than	<
Greater than	>
Less or equal	<=
Greater or equal	>=

'IF' STATEMENT

if statement is used to analyze and evaluate something if some specific condition is met (i.e. true). For example, if you are entering a value for some variable *x*, you can use *if* to check if *x* value is less than 10. In that case you can choose to manipulate value of *x* to some other value, say add 5 to *x*, and then use the new value to perform rest of the operations. If *x* is greater or equal to 10, then just use the original value of *x* to perform rest of the operations.

```
x = input('Enter a value for x:' );

if x < 10
    x = x + 5;  %   updated value of x
end

y = x^2;
```

Observe that *if* statement always concludes with *end* statement. In the above code, value of *x* will only be updated and subsequently used to calculate *y* if *x* is less than 10. If *x* is greater or equal to 10, *if* statement will yield a false or '0' value and it will conclude without running anything between *if* and *end* statements; hence, value of *x* will not be updated and *y* will be calculated based on the original value entered by the user.

if statements can also contain two or more conditions to be satisfied. For example, in the above code we can check if *x* is greater or equal to zero but less or equal to 10.

```
x = input('Enter a value for x: ');

if (x >= 0) & (x <= 10)
    x = x+5;
end
y = x^2;
```

Observe the use of logical AND (&) between the two conditions of *x*. If you have to satisfy all of the conditions for *x*, you have to use logical AND (&) but if you have to satisfy at least one condition out of several, you have to use logical OR (|).

if-else Statements

If there is a situation that calls for two different operations based on some condition, i.e. if condition is true, perform one operation, and if it is false, perform the other operation, *if-else* statement is used. For example, if *x* is greater or equal to zero but less or equal to 10, calculate *y* according to the expression $2x^3 + sin(x)$, otherwise calculate *y* according to the expression $2x^2 + cos(x^2)$.

```
x = input('Enter a value for x: ');

if (x >= 0) & (x <= 10)
    y = 2*x^3 + sin(x);
else
    y = 2*x^2 + cos(x^2);
end
```

Observe that *if-else* statement also terminates with *end* statement.

if-elseif Statements

If more than two different operations have to be carried out based on the variable of interest taking on multiple (at least three) different conditions, *if-elseif* statements are used. For example, if *x* is greater or equal to zero but less than ten, evaluate *y* according to the expression $2x^3 + sin(x)$, if it is greater or equal to ten but less than 20, evaluate *y* according to the expression $2x^2 + cos(x^2)$, if it is greater or equal to 20 but less or equal to 30, evaluate *y* according to the expression $3x$. For any other value of *x*, $y = 0$.

```
x = input('Enter a value for x: ');

if (x >= 0) & (x < 10)
    y = 2*x^3 + sin(x);
elseif (x >= 10) & (x < 20)
    y = 2*x^2 + cos(x^2);
elseif (x >= 20) & (x <= 30)
    y = 3*x;
else
    y = 0;
end
```

EXAMPLE 16

Write a program that asks for an input from the user. If input is between [0, 10], evaluate *x*, if it is between (10, 20], evaluate *y*, and if it is less than 0 or greater than 20, evaluate *z*. *x*, *y*, and *z* are given as follows. Observe that the square bracket means that the number is inclusive (i.e. included in the range) and parenthesis means that the number is exclusive (i.e. not included in the range). Also, print out the result in the command window with the user input, range for which the specific equation is selected and the selected equation.

$x = 2a(\cos(a))\ \log_{10}(a);\ y = 9\ln(a)\tan(a);\ z = 20\sin(a)$

where *a* is the user input.

Script

```
%   Masood Ejaz - Example 16 - MATLAB Tutorial - 11/08/2013

a = input('Please enter a number: ');

if (a >= 0) & (a <= 10)
    x = 2*a*cos(a)*log10(a);
    fprintf('User entered number is %0.2f which is between the range of [0,10]. ', a)
    fprintf('Equation that is evaluated for this number is 2x*cos(x)*log(x) and the result is %0.2f\n', x);
elseif (a > 10) & (a <= 20)
    y = 9*log(a)*tan(a);
    fprintf('User entered number is %0.2f which is between the range of (10,20]. ', a)
     fprintf('Equation that is evaluated for this number is 9*ln(x)*tan(x) and the result is %0.2f\n', y);
else
    z = 20*sin(a);
    fprintf('User entered number is %0.2f which is outside the range of [0, 20]. ', a)
    fprintf('Equation that is evaluated for this number is 20sin(x) and the result is %0.2f\n',z);
end
```

(continued)

Output

```
>> example16
Please enter a number: 5.32
User entered number is 5.32 which is between the range of [0,10]. Equation that is evaluated for this number is 2x*cos(x)*log(x) and the result is 4.41
>> example16
Please enter a number: 16.23
User entered number is 16.23 which is between the range of (10,20]. Equation that is evaluated for this number is 9*ln(x)*tan(x) and the result is 14.43
>> example16
Please enter a number: -20
User entered number is -20.00 which is outside the range of [0, 20]. Equation that is evaluated for this number is 20sin(x) and the result is -18.26
>>
```

EXERCISE 16

Ask user to enter values for some resistors (a vector). If any of the value is negative, print a message in the command window '*Resistors cannot be negative'*. If all of the values are positive, calculate series and parallel equivalent of the resistors using the following formulae and print out the results in the command window as shown in the sample output.

Equivalent resistance for the resistors connected in series: $R_1 + R_2 + R_3 + R_4 + \ldots\ldots$

Equivalent resistance for the resistors connected in parallel: $\frac{1}{\frac{1}{R1}+\frac{1}{R2}+\frac{1}{R3}+\cdots\ldots}$

(Useful functions: *sum*, *find*)

Sample Outputs

```
Enter resistor values as a vector: [23 45 67 98 100 250 87]
Series Equivalent Resistance: 670.00
Parallel Equivalent Resistance: 8.60
>>
```

```
Enter resistor values as a vector: [23 45 67 98 100 -98 250 87]
Resistors cannot be negative
>>
```

'SWITCH' STATEMENT

If there is a situation that calls for choosing different operations based on different inputs (integers or strings), *switch* statement is used. For example, if input is '1', do certain operation, if it is '2', do some other operation and so on, or if input is 'A' (or any other string), do one operation, if it is 'B' (or any other string), do another operation and so on, *switch* statement can be used. In other words, *switch* statement is used when choice has to be made from a pool of values and operation is done based on which value has been selected. The main difference between *if* and *switch* is that one of the switch conditions (cases) has to be equal to the input condition in order for the specific operation to be carried out; there cannot be any inequality. Also, input condition can only be integer or string.

General format of the *switch* statement is as follows,

```
switch expression
    case expression value 1
        operations to be carried out
    case expression value 2
        operations to be carried out
            :
            :
            :
    otherwise
        operations to be carried out
end
```

Note that *expression* is the variable where input value is stored, and *case* represents possible cases or possible conditions of *expression* as given by *expressions values*.

EXAMPLE 17

Display the average score for the term or final exam for the MATLAB class based on the user's inquiry. Assume that the term exam average is 75.4 out of 85 or 88.7% and the final average is 70 out of 85 or 79.5%.

Script

```
%   Masood Ejaz - Example 17 - Nov 1, 2013 - MATLAB Tutorial

exam = input('Please select the exam for which you want to check the score (midterm or final) as a string: ' );

switch lower(exam)
    case 'midterm'
        fprintf('Midterm average score for the MATLAB section is 75.4 out of 85 or 88.7 percent\n');
    case 'final'
        fprintf('Final average score for the MATLAB section is 70 out of 85 or 79.5 percent\n');
    otherwise
        fprintf('Your selection is invalid\n')
end
```

Output

```
>> example17
Please select the exam for which you want to check the score (midterm or final) as a string: 'midterm'
Midterm average score for the MATLAB section is 75.4 out of 85 or 88.7 percent
>> example17
Please select the exam for which you want to check the score (midterm or final) as a string: 'final'
Final average score for the MATLAB section is 70 out of 85 or 79.5 percent
>> example17
Please select the exam for which you want to check the score (midterm or final) as a string: 'Exam I'
Your selection is invalid
>>
```

Note: Observe the function *lower* used in the switch statement. Check help for *lower* to understand what it does. Also, anytime input is a string, it has to be entered inside *commas* as you can observe under *output*.

EXAMPLE 18

Modify *example 17* such that for either *Exam I* or *midterm*, same score will be displayed.

Script

```
%   Masood Ejaz - Example 18 - Nov 14, 2013 - MATLAB Tutorial

exam = input('Please select the exam for which you want to check the score as a string: ' );

switch lower(exam)
    case {'midterm', 'exam i'}
        fprintf('Midterm average score for the MATLAB section is 75.4 out of 85 or 88.7 percent\n\n');
    case 'final'
        fprintf('Final average score for the MATLAB section is 70 out of 85 or 79.5 percent\n\n');
    otherwise
        fprintf('Your selection is invalid; You can choose either Midterm or Exam I, or Final \n\n')
end
```

(*cont'd*)

Output

```
>> example18
Please select the exam for which you want to check the score as a string: 'Midterm'
Midterm average score for the MATLAB section is 75.4 out of 85 or 88.7 percent

>> example18
Please select the exam for which you want to check the score as a string: 'Exam I'
Midterm average score for the MATLAB section is 75.4 out of 85 or 88.7 percent

>> example18
Please select the exam for which you want to check the score as a string: 'Final'
Final average score for the MATLAB section is 70 out of 85 or 79.5 percent

>> example18
Please select the exam for which you want to check the score as a string: 'Exam II'
Your selection is invalid; You can choose either Midterm or Exam I, or Final

>>
```

Note: In order to have multiple choices to produce same results or go through same operations, choices have to separated by *commas* inside *braces* {}, as you can observe in the script on *line 6*.

EXAMPLE 19

Given the following set of linear equations, solve for the unknown variables using Cramer's rule or matrix inversion based on whichever method is chosen by the user. Ask user to choose Cramer's rule by entering *1* and matrix inversion method by entering *2*.

$$5x + 6y = 11$$

$$7x + 3y = 13$$

Script

```
1     %   Masood Ejaz - Example 19 - Nov 14, 2013 - MATLAB Tutorial
2
3 -   fprintf('Please select the method you would like to use to solve the following set of linear equations\n');
4 -   fprintf('5x + 6y = 11 & 7x + 3y = 13\n');
5 -   method = input('Enter 1 for Cramers rule and 2 for matrix inversion method: ' );
6
7 -   switch method
8 -       case 1
9 -           fprintf('\n Solution using Cramers rule\n');
10 -          x = det([11 6;13 3])/det([5 6;7 3])
11 -          y = det([5 11;7 13])/det([5 6;7 3])
12 -      case 2
13 -          fprintf('\n Solution using matrix inversion\n')
14 -          X = inv([5 6;7 3])*[11 13]';
15 -          x = X(1)
16 -          y = X(2)
17 -      otherwise
18 -          fprintf('Your selection is invalid; please use 1 for Cramers rule and 2 for matrix inversion\n\n')
19 -  end
```

(cont'd)

Output Samples

```
>> example19
Please select the method you would like to use to solve the following set of linear equations
5x + 6y = 11 & 7x + 3y = 13
Enter 1 for Cramers rule and 2 for matrix inversion method: 2

 Solution using matrix inversion

x =

    1.6667


y =

    0.4444

>>
```

```
>> example19
Please select the method you would like to use to solve the following set of linear equations
5x + 6y = 11 & 7x + 3y = 13
Enter 1 for Cramers rule and 2 for matrix inversion method: 1

 Solution using Cramers rule

x =

    1.6667


y =

    0.4444

>>
```

```
>> example19
Please select the method you would like to use to solve the following set of linear equations
5x + 6y = 11 & 7x + 3y = 13
Enter 1 for Cramers rule and 2 for matrix inversion method: 4
Your selection is invalid; please use 1 for Cramers rule and 2 for matrix inversion

>>
```

EXERCISE 17

In Digital Signal Processing, *window* functions are used to smooth out the data to observe a relatively less noisy spectrum. Ask user to enter input data. Also, ask user to choose one of the three window functions as defined below. Generate *window* vector (same length as the input data vector) and multiply it by the input data vector to get *windowed-input* data vector. Plot both the input data vector and windowed-input data vector in a single plot using two subplots.

Triangular Window: $w_{tri}(n) = 1 - \frac{|2n-N+1|}{N-1}$

Hamming Window: $w_{hm}(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right)$

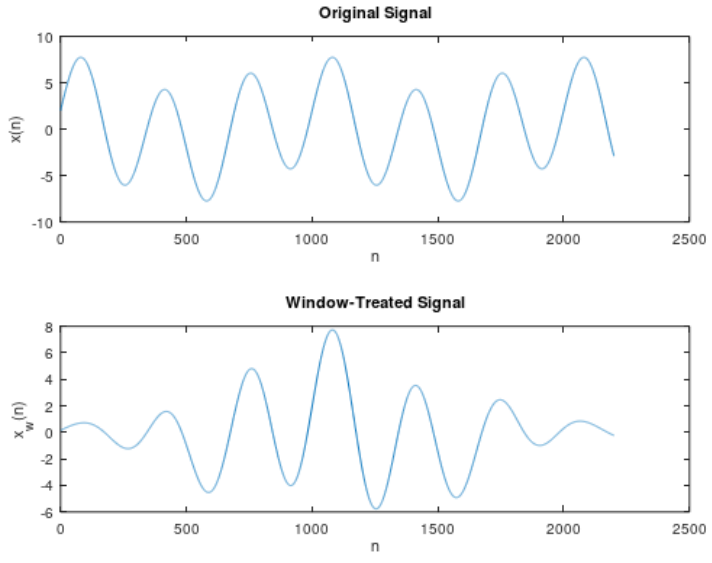
Hanning Window: $w_{hn}(n) = 0.5 - 0.5\cos\left(\frac{2\pi n}{N-1}\right)$

Note that for each of the window function, *N* is the length of the vector which is the same as the length of the input data vector, and *n* = 0, 1, 2, 3, ……, *N*-1.

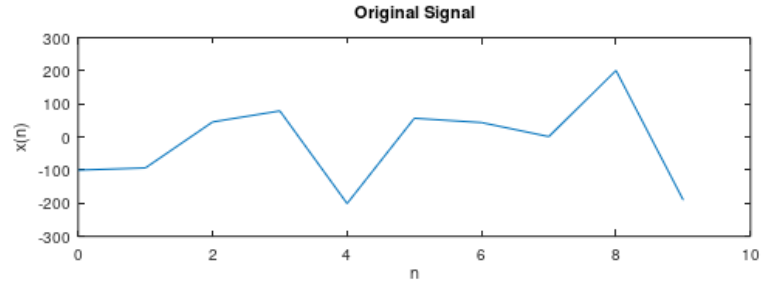
Note: You must use *switch* to work on this exercise.

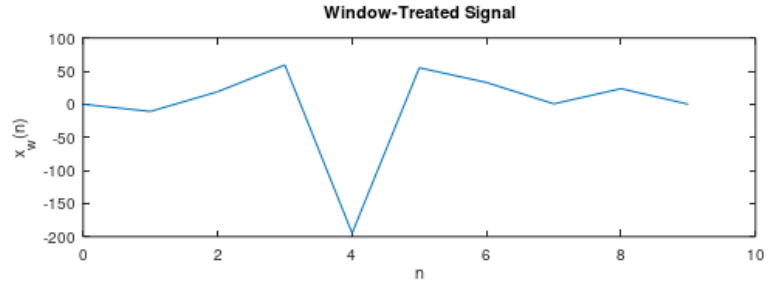
Sample Outputs

```
Enter a data vector required to be treated by one of the window functions: 2*cos(2000*pi*(0:1e-6:2.2e-3)) + 6*sin(6000*pi*(0:1e-6:2.2e-3))
Which window function do you want to use: Triangular, Hamming, or Hanning? hamming
>>
```

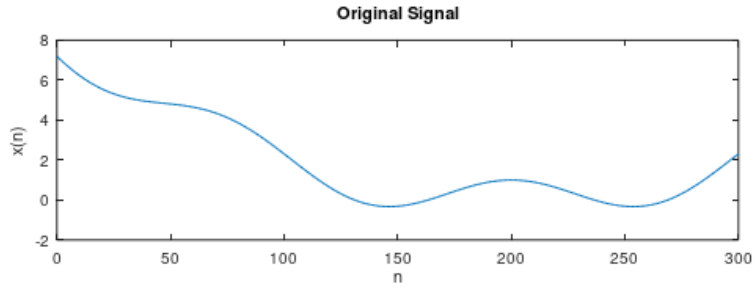


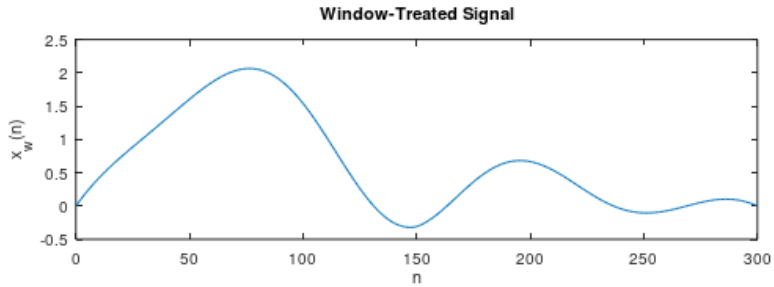
```
Enter a data vector required to be treated by one of the window functions: [-100, -93.3, 45.6, 78.9, -200.8, 56.7, 43.7, 1.3, 200.9, -190.3]
Which window function do you want to use: Triangular, Hamming, or Hanning? Hanning
>>
```






```
Enter a data vector required to be treated by one of the window functions: 2*(-2:0.01:1).^2 + cos(1.6*pi*(-2:0.01:1))
Which window function do you want to use: Triangular, Hamming, or Hanning? TRIANGULAR
>>
```





'WHILE' LOOP

While loop is used to execute an operation or set of operations over and over again as long as some condition is true. As soon as condition becomes false, *while* loop is terminated. General format of *while* loop structure is as follows:

```
while expression
      statements or operations
end
```

EXAMPLE 20

Ask user to enter a vector *x*. Add the elements of *x* as long as sum is smaller than 100. As soon as sum becomes greater or equal to 100, stop the summing process and print out the number of elements that produced the sum and value of the sum. If sum of all of the elements produce a value that is less, equal or greater than 100, print out sum of all of the elements and input vector length.

Script

Note: Please pay attention on the comments to properly understand different steps of the code.

```
%   Masood Ejaz - Example 20 - MATLAB Tutorial - Nov 16, 2013

x = input('Enter a vector: ');

a = 1;  %   A variable to keep track of iteration count
k = x(a);   %   a-th element of x is assigned to k; in this case, the first element

while (k < 100) & (a < length(x))  %   checking two conditions; if sum < 100 and current iteration is less than the total length of vector x
    a = a + 1;  %   updating the iteration count
    k = k + x(a);   %   adding the next element of x in the previous sum 'k' and assigning the updated value back to 'k'.
end

if a == length(x)   %   if total number of iterations is equal to the length of vector x
    fprintf('Adding all of the elements of the input vector gives %0.2f. Length of the input vector is %d\n', k, a);
else
    fprintf('Adding up to element # %d of the input vector produced %0.2f which is greater than or equal to 100\n', a, k);
end
```

Output Samples

```
>> example20
Enter a vector: [12.3 45.6 78 3 21 9 4.5 7.8 19.87]
Adding up to element # 3 of the input vector produced 135.90 which is greater than or equal to 100
>>


>> example20
Enter a vector: [1 2 3 4 5 6 7 8 9 10]
Adding all of the elements of the input vector gives 55.00. Length of the input vector is 10
>>
```

Random Number Generation

In MATLAB, there are three functions that can produce random numbers; *rand*, *randn*, and *randi*. *rand* produces a random number (*rand* or *rand(1)*), a random vector (*rand(1,N)* or *rand(M,1)*, where *N* is number of columns and *M* is number of rows) or a random matrix (*rand(N)*, a random *N-by-N* square matrix or *rand(M,N)*, a random *M-by-N* matrix). Random numbers that are generated have standard uniform distribution and range over open interval (*0,1*), i.e. real numbers from *0* to *1*, excluding *0* and *1*.

randi generates random integers having discrete uniform distribution from *1* to a maximum number given by the user. Once again, a single number (*randi(Max#, 1)* generates a single number from *1* to any *Max#*), a vector (*randi(Max#, 1, N)* or *randi(Max#, M, 1)*), or a matrix (*randi(Max#, N)* or *randi(Max#, M, N))* can be generated using *randi*.

randn generates random numbers from standard normal or Gaussian distribution. Once again, you can generate a single number, a vector, or a matrix in the same way as described for *rand* function.

EXERCISE 18

Generate a random integer between *1* and *10*. Ask user to guess the number. If user's guess is incorrect, ask to guess again until user gets it right. At this point, tell user that he guessed it correctly and print out the number of attempts it took for the correct guess.

Output Samples

```
>> exercise18
Please guess an integer between 1 and 10: 5
Your guess is incorrect. Please try again: 2
You guessed it correctly in 2 attempts
>>
```

```
>> exercise18
Please guess an integer between 1 and 10: 2
Your guess is incorrect. Please try again: 3
Your guess is incorrect. Please try again: 6
Your guess is incorrect. Please try again: 5
Your guess is incorrect. Please try again: 8
Your guess is incorrect. Please try again: 9
Your guess is incorrect. Please try again: 10
Your guess is incorrect. Please try again: 1
Your guess is incorrect. Please try again: 4
Your guess is incorrect. Please try again: 7
You guessed it correctly in 10 attempts
>>
```

'FOR' LOOP

for loop is also used to repeat some operation(s) or statement(s) over and over again like *while* loop does. The difference between *for* and *while* loops is that *for* loop is run for a pre-determined number of times whereas *while* loop keeps on running until the condition for the loop becomes false. General format for the *for* loop is as follows.

```
for k = initial value : interval : end value
    operations or statements
end
```

for loop is going to run starting from the *initial value* assigned to the variable *k*, which will be updated by an amount given by *interval* each time loop is going to run. Note that *k* may or may not be the iteration count for the *for* loop. If *interval* is *1* and initial value is also *1*, *k* will be the same as iteration count each time loop is going to run. For any other initial value or/and *interval*, value of *k* and loop iteration count will be different. Note that *k* is a vector and number of times loop is going to run can easily be found out by taking the length of the vector *k* or by updating a variable by *1* each time loop is going to run. This variable will serve as an iteration count variable. If *interval* value is *1*, there is no need to write down the interval; just *initial value* : *final value* will update the variable *k* by *1* for each loop iteration.

EXAMPLE 21

Write a program with *for* loop that will print a column of real numbers from 1.5 to 3.1 in steps of 0.2.

Script

```
1    %   Masood Ejaz - Example 21 - MATLAB Tutorial - Nov 17, 2013
2
3 -  for k = 1.5:0.2:3.1
4 -      fprintf('%0.2f\n',k);
5 -  end
```

Output

```
>> example21
1.50
1.70
1.90
2.10
2.30
2.50
2.70
2.90
3.10
>>
```

EXAMPLE 22

Ask user to enter a vector *x*. Examine each element of *x*, one by one, and if the element is greater than 10, evaluate the expression *y = 4sin(x)cos(x)*. If the element is less than or equal to 10, evaluate *y = 3cos(3x)sin(5x)*. Make sure to keep *y* as a vector that will keep all of the evaluated results. Also, print out an appropriate message with the input value, equation that is used to evaluate the new value, and the evaluated value.

Script

```
%   Masood Ejaz - Example 22 - MATLAB Tutorial - Nov 17, 2013

x = input('Enter a vector: ');
clear y; %  clearning up variable y

for k = 1:length(x) %   observe that you can use functions and variables to define the range of k
    if x(k) > 10    %   this will check each value of x as k goes from 1 to the last element of x
        y(k) = 4*sin(x(k))*cos(x(k)); %   creating a new vector y
        fprintf('Since x(%d) is %0.2f which is greater than 10, hence y(%d) = %0.2f' , k, x(k), k, y(k));
        fprintf(' which is evaluated according to the expression y = 4sin(x)cos(x)\n');
    else
        y(k) = 3*sin(5*x(k))*cos(3*x(k)); %   creating a new vector y
        fprintf('Since x(%d) is %0.2f which is less than or equal to 10, hence y(%d) = %0.2f' , k, x(k), k, y(k));
        fprintf(' which is evaluated according to the expression y = 3sin(5x)cos(3x)\n');
    end   % end for the if statement
end  %end for the for loop
```

Sample Output

```
>> example22
Enter a vector: [2.3 45 12 -9.3 56 18.3 5.6]
Since x(1) is 2.30 which is less than or equal to 10, hence y(1) = -2.14 which is evaluated according to the expression y = 3sin(5x)cos(3x)
Since x(2) is 45.00 which is greater than 10, hence y(2) = 1.79 which is evaluated according to the expression y = 4sin(x)cos(x)
Since x(3) is 12.00 which is greater than 10, hence y(3) = -1.81 which is evaluated according to the expression y = 4sin(x)cos(x)
Since x(4) is -9.30 which is less than or equal to 10, hence y(4) = 1.63 which is evaluated according to the expression y = 3sin(5x)cos(3x)
Since x(5) is 56.00 which is greater than 10, hence y(5) = -1.78 which is evaluated according to the expression y = 4sin(x)cos(x)
Since x(6) is 18.30 which is greater than 10, hence y(6) = -1.78 which is evaluated according to the expression y = 4sin(x)cos(x)
Since x(7) is 5.60 which is less than or equal to 10, hence y(7) = -0.37 which is evaluated according to the expression y = 3sin(5x)cos(3x)
>>
```

EXERCISE 19

Write a program using *for* loop that generates a multiplication table for any integer (given by the user) up to any multiple (also given by the user). Your output should be similar to what is given in the sample below.

Sample Output

```
>> exercise19
Enter the number for which you want to create the multiplication table: 91
Enter the number up to which multiplication table is required: 5
91 X 1 = 91
91 X 2 = 182
91 X 3 = 273
91 X 4 = 364
91 X 5 = 455
>>
```

EXERCISE 20

Work on *Exercise 11* again using *for* loop.

Hint: Define the values of ζ in a vector. Use *for* loop to cycle through the values of ζ ($\zeta(k)$). Save the output *y* as a row vector each time *for* loop is going to run; *y*(k, :), where *k* goes from 1 to 5 for the five values of ζ. Resultant *y* will be a matrix with five rows and *N* columns, where *N* is the length of the vector *x* that represents the *x-axis*. Make sure to ask user to enter vector *x* for which *y* values are calculated and plotted.

Nested 'for' Loops

Nested *for* loops mean a *for* loop inside a *for* loop. One can have multiple *for* loops, one inside the other, making a multiple nested *for* loop structure. Nested *for* loops are used when multiple variables are changing and process has to be carried out over all or some of the changing variables. When nested *for* loops are used, each of the *for* statement has to be closed with the *end* statement; hence, nested *f*or loops have multiple *end* statements too.

EXAMPLE 23

Ask user to enter a matrix. Examine each element of the matrix and if element is greater than 50, subtract 30 from it and put it in a new matrix at the element position corresponding to the original matrix. If element is less than 50, than add 10 and put it in the new matrix. When your new matrix will be ready, it will have same number of rows and columns as your original matrix.

Script

```
%   Masood Ejaz - Example 23 - MATLAB Tutorial - November 22, 2013

A = input('Enter a matrix: ');
size_A = size(A);   %   Checking how many rows and columns are there in matrix A
clear B;        % creating a new matrix B

for j = 1:size_A(1) %   creating a for loop that goes through all the rows
    for k = 1:size_A(2) %   creating a for loop that goes through all the columns
        if A(j,k) > 50  %   Testing element of matrix A
            B(j,k) = A(j,k) - 30;   %   Creating a new matrix B
        else
            B(j,k) = A(j,k) + 10;
        end %   This end is for 'if' statement
    end %   This end is for the inner for loop (k)
end %   This end is for the outer for loop (j)

disp('New matrix is')
disp(B);    %   displaying matrix B

```

(continued)

Output Sample

```
Enter a matrix: [2 55 64 37; 98 12 45 30]
New matrix is
    12    25    34    47
    68    22    55    40

>>
```

When nested *for* loops are used, first the inner *for* loop runs through all of its values for the first value of the outer loop. Outer loop then goes to its second value and again inner loop runs through all of its values for the second value of the outer loop. Outer loop then gets updated to its third value and this process keeps on going until the outer loop also goes through all of its values. At this point, program comes out of the nested *for* loop structure and moves on to the next phase.

EXERCISE 21

Ask user to enter a matrix. Examine each element of the matrix and if element is between [0,20], create a new matrix with element at the corresponding location to be $y = 4\sin(x)\cos(3x)$, where y is the new matrix element and x is the original matrix element. If x is between (20, 70], evaluate corresponding $y = 5\sin(6x)\cos(5x)$. For all other values of x, evaluate corresponding $y = 10\sin(x)$. Assume x to be in degrees for all the equations. Display the new matrix at the end of your program.

Series Summation

For loops are generally used in summation processes, as shown in *Example 24*.

EXAMPLE 24

Write a MATLAB script to add and plot the following summation series. Ask user to enter the vector *x* over which the series is calculated and plotted.

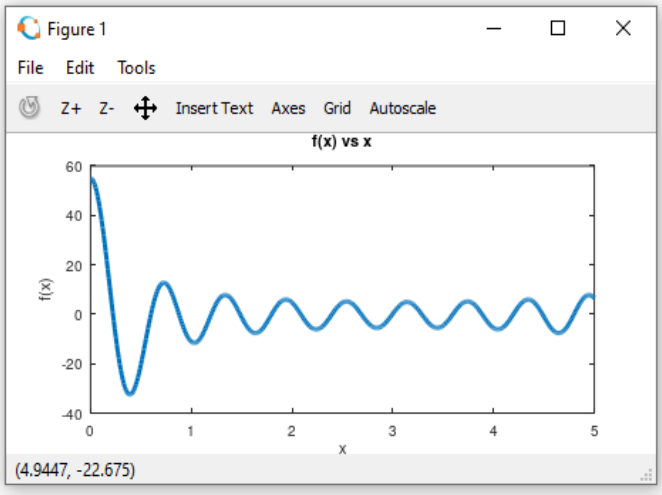
$$f(x) = \sum_{k=1}^{10} k\cos(kx)$$

Script

```
% Example 24 - MATLAB Tutorial - Masood Ejaz - 06/18/2021
%
% In this example, a For loop will be used to produce result of a summation series
% and then plot of the series over a user-defined vector will be created.
%
x = input('Enter the vector over which the summation series will be evalauted and plotted: ');
fx = 0; % This is the initial value of the summation series

for k = 1:10
   fx = fx + k*cos(k*x); % The value of k*cos(kx) will be evaluated for the current value of k
                         % and added to the previous value and the updated result will be assigned to fx
endfor

plot(x,fx,'Linewidth', 3), xlabel('x'), ylabel('f(x)'), title('f(x) vs x')
% observe the use of Linewidth in the plot function, which can change the width of the plot line
```

Output Sample

```
>> example24
Enter the vector over which the summation series will be evalauted and plotted: 0:0.01:5
>>
```



EXERCISE 22

Calculate and plot the following summation series over *x*. Ask user to enter the values of ζ as a vector and values of *x* over which the summation series will be evaluated and plotted, also a vector.

$$y = \sum_{k=1}^{n} 1 - \frac{1}{\sqrt{1-\zeta(k)^2}} e^{-\zeta(k)x} \sin\left(\left(\sqrt{1-\zeta(k)^2}\right)x + \cos^{-1}(\zeta(k))\right)$$