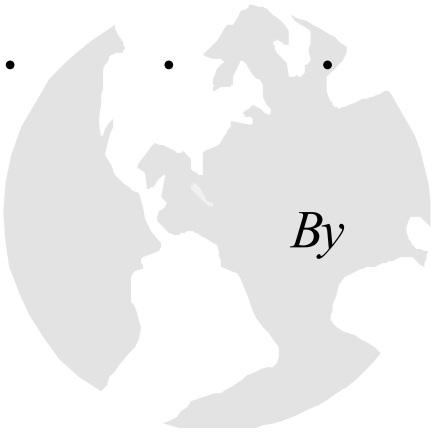
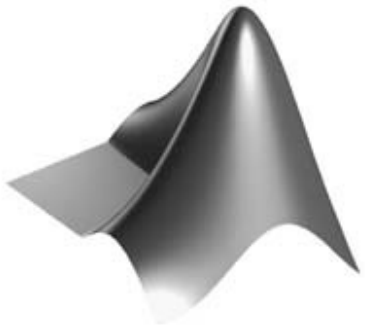
Beginner's Resource

Introduction to Matlab®





By

Dr. Sikander M. Mirza

Department of Physics and Applied Mathematics
Pakistan Institute of Engineering and Applied Sciences
Nilore, Islamabad 45650, Pakistan

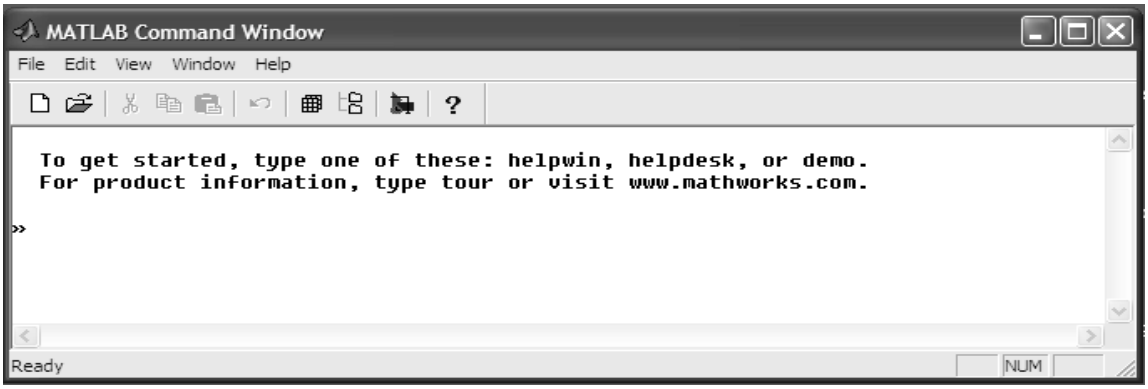
Table of Contents

General Features

Matlab is an interactive working environment in which the user can carry out quite complex computational tasks with few commands. It was originally developed in 1970s by Cleve Muller. The initial programming was in Fortran and over period of time, it has constantly evolved. The latest version is in C. As far as numerical programming is concerned, it removes programming of many routine tasks and allows one to concentrate on the task encouraging experimentation. The results of calculations can be view both numerically as well as in the form of 2D as well as 3D graphs easily and quickly. It incorporates state-of-the-art numerical solution tools, so one can be confident about the results. Also, quite complex computations can be performed with just a few commands. This is because of the fact that the details of programming are stored in separate script files called the 'm'- files and they can be invoked directly with their names. An m-file can invoke another m-file when required. In this way, a series of m-files running behind the scene allow execution of the required task easily. The user can write his/her own m-files. All such scripts are text readable files which can be read, modified and printed easily. This open-architecture of Matlab® allows programmers to write their own area specific set of m-files. Some such sets written by various experts world-wide have already been incorporated into the Matlab as tool boxes. So, with standard installations, you will find latterly dozens of tool boxes. If you wish, you can down-load even more from the internet.

Startup

When you click the Matlab icon, the MS Windows opens up the standard Matlab-window for you which has the following form:



The white area in the middle is the work area in which the user types-in the commands which are interpreted directly over there and the results are displayed on screen. The '>>' is Matlab prompt indicating that user can type-in command here. A previously entered command can be reached with the help of up-arrow and down-arrow buttons on the keyboard.

Simple Calculations

Matlab uses standard arithmetic operators + - / * ^ to indicate addition, subtraction, division, multiplication and raised-to-the-power respectively. For example, in order to calculate the answer for $2+3^4$, one would type the following:

```
» 2+3^4
ans =
     83
```

The first line is the user entered command while the second line is default variable used by Matlab for storing the output of the calculations and the third line shows the result of computation. If you wish to multiply this result with 2, proceed as below:

```
» ans*2
ans =
   166
```

As you can see, the result 83 stored in variable ans gets multiplied with 2, and the result of this new computation is again stored in variable 'ans.' In this case, its previous value gets over-written by the new variable value. If you wish, you can define your own variables. For example:

```
» pay=2400
pay =
        2400
```

In this case, you define the variable 'pay' and assign the variable a value 2400. Matlab echoes the assignment in the second and third line. This confirms the user that a value of 2400 has been assigned to the variable 'pay' which is quite useful at times. If you wish to remove this echo in Matlab, use a semicolon at the end of each command. For example:

```
» c=3*10^8;
»
```

The variable 'c' has been assigned a value $3x10^8$ and since there is a semicolon at the end of the command, therefore, no echo is seen in this case.

The arithmetic operators have the following precedence-levels:

1. Brackets first. In case they are nested, then the sequence is from inner-most to the outermost.

2. Raised to power next.
3. Multiplication and division next. If there are such competing operators, then the sequence is from left to right.
4. Addition and subtraction next. In this case also, if there are competing such operators, then the sequence is from left to right

Numbers and Storage

Matlab performs all calculations in double precision and can work with the following data types:

Numbers	**Details**
Integer	Numbers without any fractional part and decimal point. For example, 786
Real	Numbers with fractional part e.g., 3.14159
Complex	Numbers having real and imaginary parts e.g., 3+4i. Matlab treats 'i' as well as 'j' to represent $\sqrt{-1}$
Inf	Infinity e.g., the result of divided with zero.
NaN	Not a number e.g., 0/0

For display of the results, Matlab uses ***Format*** command to control the output:

Category	**Details**
format short	4 decimal places (3.1415)
format short e	4 decimal places with exponent (3.1415e+00)
format long e	normal with exponent (3.1415926535897e+00)
format bank	2 decimal places (3.14)

By using 'format' without any suffix means that from now onwards, the default format should be used. Also, 'format compact' suppresses any blank lines in the output.

Variable Names

Matlab allows users to define variable with names containing letters and digits provided that they start with a letter. Hyphen, % sign and other such characters are not allowed in variable names. Also, reserved names should not be used as variable names. For example, pi, i, j, and e are reserved. Similarly, the names of functions and Matlab commands should also be avoided.

Case Sensitivity

Matlab command structure is quite similar to the C-language. The variables are case sensitive. So, ALPHA and alpha are treated as separate variables. The case sensitivity is also applicable to Matlab commands. As a general rule, the lower-case variable names as well as commands are typically used.

Functions

Matlab has a potpourri of functions. Some of these are standard functions including trigonometric functions etc., and others are user-defined functions and third party functions. All of these enable user to carry out complex computational tasks easily.

Trigonometric Functions

These include sin, cos and tan functions. Their arguments should be in radians. In case data is in degrees, one should convert it to radians by multiplying it with pi/180. For example, let us calculate the value of $\sin^2(27^o)+\cos^2(27^o)$:

```
» (sin(27*pi/180))^2+(cos(27*pi/180))^2
ans =
     1
```

The result of these computations is no surprise. Note that in each case, the argument of the trigonometric function was converted to radians by multiplying it suitably.

The inverse functions are invoked by asin, acos and atan. For example, $\tan^{-1}(1)$ is computed as:

```
» atan(1)
ans =
    0.7854
```

Of course, $\pi/4 = 0.7854$.

Some Elementary Functions

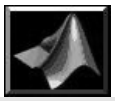
Typically used common functions include sqrt, exp, log and log10. Note that log function gives the natural logarithm. So,

```
» x=2; sqrt(x), exp(-x), log(x), log10(x)
ans =
    1.4142
ans =
```

```
    0.1353
ans =
    0.6931
ans =
    0.3010
```

Here, all four functions have been tested using the same command. As you can see, the semicolon suppresses the echo while the comma separates various computations. Summary of some functions is given below:

Function	Stands for
abs	Absolute value
sqrt	Square root function
sign	Signum function
conj	Conjugate of a complex number
imag	Imaginary part of a complex number
real	Real part of a complex number
angle	Phase angle of a complex number
cos	Cosine function
sin	Sine function
tan	Tangent function
exp	Exponential function
log	Natural logarithm
log10	Logarithm base 10
cosh	Hyperbolic cosine function
sinh	Hyperbolic sine function
tanh	Hyperbolic tangent function
acos	Inverse cosine
acosh	Inverse hyperbolic cosine
asin	Inverse sine
asinh	Inverse hyperbolic sine
atan	Inverse tan
atan2	Two argument form of inverse tan
atanh	Inverse hyperbolic tan
round	Round to nearest integer
floor	Round towards minus infinity
fix	Round towards zero
ceil	Round towards plus infinity
rem	Remainder after division



Vectors

In Matlab, there are two types of vectors: the row vectors and the column vectors.

The Row Vectors

The row vectors are entities enclosed in pair of square-brackets with numbers separated either by spaces or by commas. For example, one may enter two vectors U and V as:

```
» U=[1 2 3]; V=[4,5,6]; U+V
ans =
     5     7     9
```

The two row vectors were first defined and then their sum U+V was computed. The results are given as a row vector stored as ans. The usual operations with vectors can easily be carried out:

```
» 3*U+5*V
ans =
    23    31    39
```

The above example computed the linear combination of U and V. One can combine vectors to form another vector:

```
» W=[U, 3*V]
W =
   1   2   3   12   15   18
```

The vector U and V both of length 3, have been combined to form a six component vector W. The components of a vector can be sorted with the help of sort function:

```
» sort([8 4 12 3])
ans =
     3     4     8    12
```

The vector [8 4 12 3] has been sorted.

The Colon Notation

In order to form a vector as a sequence of numbers, one may use the colon notation. According to which, a:b:c yields a sequence of numbers starting with ‘a’, and possibly ending with ‘c’ in steps of ‘b’. For example 1:0.5:2 yields he following column vector:

```
» 1:0.5:2
ans =
   1.0000   1.5000   2.0000
```

Note that in some cases, the upper limit may not be attainable thing. For example, in case of 1:0.3:2, the upper limit is not reached and the resulting vector in this case is:

```
» 1:0.3:2
ans =
    1.0000    1.3000    1.6000    1.9000
```

If only two of the 'range' specifications are given then a unit step size is automatically assumed. For example 1:4 means:

```
» 1:4
ans =
     1     2     3     4
```

In case, the range is not valid, an error message is issued:

```
» 1:-1:5
ans =
  Empty matrix: 1-by-0
```

Here, the range of numbers given for the generation of row vector was from 1 to 5 in steps of -1. Clearly, one can not reach 5 from 1 using -1 step size. Therefore, the Matlab indicates that this is an empty matrix.

Sections of a Vector

Let us define a vector using the range notation:

```
» W=[1:3, 7:9]
W =
     1     2     3     7     8     9
```

Now, we would like to extract the middle two elements of this vector. This can be done with the range notation again. As you can see, the middle two elements are 3:4 range. Therefore, the required part of vector can be obtained as:

```
» W(3:4)
ans =
7
```

This really is the required part. There are many interesting things that can now be done using the range notation. For example, range 6:-1:1 is the descending range and when used with part-extraction of vector, it gives:

```
» W(6:-1:1)
ans =
     9     8     7     3     2     1
```

which is the vector W with all entries now in reverse order. So, a vector can be flipped easily. The 'size' function yields the length of a vector. For a given vector V, V(size(V):-1:1) will flip it. Note that flipping of sections of a vector is also possible.

Column Vectors

The column vectors in Matlab are formed by using a set of numbers in a pair of square brackets and separating them with semi-colon. Therefore, one can define two column vectors A and B and add them as below:

```
» A=[1;2;3]; B=[4;5;6]; A+B
ans =
     5
     7
     9
```

The two column vectors were defined first and then their sum was obtained. In similar way, all other standard operations with the column vectors can be carried out.

Transpose

Of course, the convenient way of creating a row vector does not have any similar method for the column vector. But, one can do it by first creating a row vector using the range notation and then transposing the resulting row vector into a column vector. The transpose is obtained with a ` as shown below:

```
» A=[1:4]; B=A'
B =
     1
     2
     3
     4
```

Here, first a row vector [1 2 3 4] is formed which is called A. This vector is then transposed to form the B—a column vector.

Note: If C is a complex vector, then C' will give its complex conjugate transpose vector.

```
» C=[1+i, 1-i]; D=C'
D =
   1.0000 - 1.0000i
   1.0000 + 1.0000i
```

The vector C was a complex vector and its complex conjugate is [1-i 1+i] vector. Vector D is clearly its complex conjugate transpose vector. Some times, one does not want the complex conjugate part. In order to get a simple transpose, use `.'` to get the transpose. For example:

```
» C=[1+i, 1-i]; E=C.'
```

```
E =
   1.0000 + 1.0000i
   1.0000 - 1.0000i
```

In this case, a plain transpose of C is stored in E and no complex conjugate part appears.

Diary and Session

In Matlab, one can start storing all text that appears on screen into a separate file by using 'diary filename' command. The filename should be any legal file name different from 'on' and 'off'. The record of diary can be turned 'on' and 'off' by using 'diary on' and 'diary off' commands.

Also, if one wishes to abort a session now and start from the same state next time, one can save and load session using 'save filename' and 'load filename' commands. The save command will save all variables used in this session into a file with name give in the 'save filename' command and the corresponding load command will read them back during a later session.

By the way, a complete list of all variables use so far in the current session can be seen using the 'who' command:

```
» who
Your variables are:
A          D          V          c
B          E          W          pay
C          U          ans        x
```

The values and further details are also available with the 'whos' command:

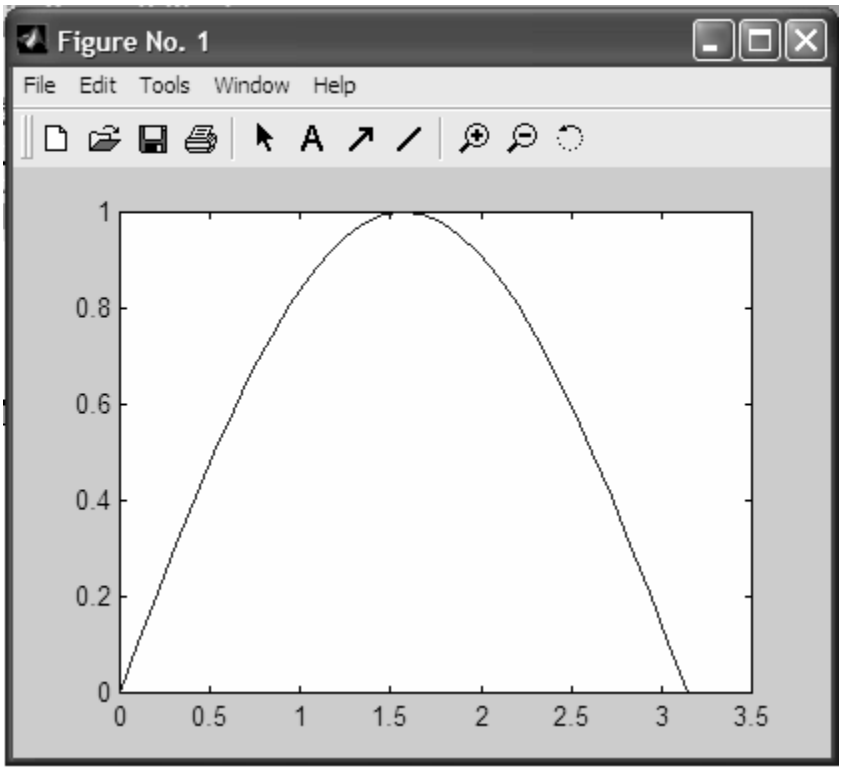
```
» whos
  Name       Size          Bytes  Class

  A          1x4              32  double array
  B          4x1              32  double array
  C          1x2              32  double array (complex)
  D          2x1              32  double array (complex)
  E          2x1              32  double array (complex)
  U          1x3              24  double array
  V          1x3              24  double array
  W          1x6              48  double array
  ans        2x1              32  double array (complex)
  c          1x1               8  double array
  pay        1x1               8  double array
  x          1x1                  8  double array
Grand total is 31 elements using 312 bytes
```

Elementary Plots and Graphs

Matlab offers powerful graphics and visualization tools. Let us start with some of the very basic graphics capabilities of Matlab. The graph of sine function in 0 to π can be obtained in the following way:

```
» N=30; h=pi/N; x=0:h:pi; y=sin(x); plot(x,y)
```

Here, in the first step, the total number of sampling points for the function is defined as N and it is assigned a value 30. Next, the step size 'h' is defined and the x row vector of size N+1 is defined along with the corresponding y row vector composed of the function values. The command 'plot(x,y)' generates the graph of this data and displays it in a separate window labeled Figure No. 1 as shown below:



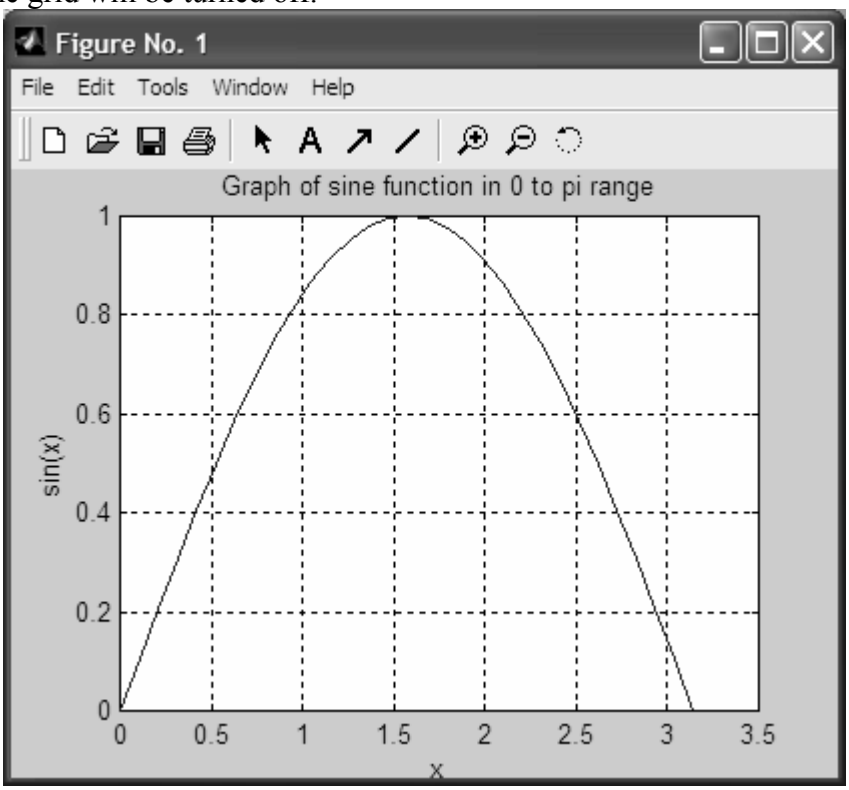
The graph displayed in this window can be zoomed-in and zoomed-out. Both x-any y-axes can also be rescaled with the help of mouse and using appropriate buttons and menu items.

The graph title, x- and y-labels can be assigned using the following commands:

```
>> title('Graph of sine function in 0 to pi range')
>> xlabel('x')
>> ylabel('sin(x)')
```

Note that by using these commands as such, one gets the corresponding response on the graph window immediately.

The grid lines on the graph can be switched on or off using the 'grid' command. By issuing this command once, grid will be turned on. Using it again, the grid will be turned off.



Matlab allows users to change the color as well as the line style of graphs by using a third argument in the plot command. For example, plot(x,y,'w-') will plot x-y data using white (w) color and solid line style (-). Further such options are given in the following table:

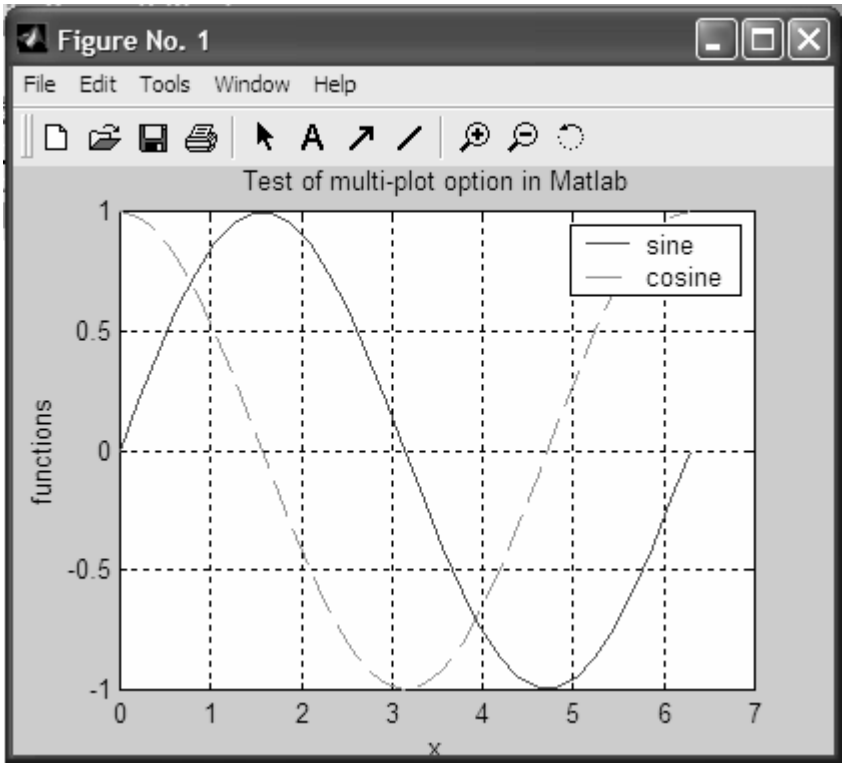
Color Symbol	Color	Line Symbol	Line type
y	Yellow	.	Point
m	Magenta	o	Circle
c	Cyan	x	x-mark
r	Red	+	Plus mark
g	Green	-	solid
b	Blue	*	Star
w	White	:	Dotted
b	Black	-.	Dash-dot
		--	dashed

Multiplots

Let us now try plotting more than one curves on the same graph. The functions are sine and cosine. The range is 0 to 2π in this case. The number of sampled points in this case will be just 15.

```
» N=15; h=pi/N; x=0:h:2*pi; plot(x,sin(x),'r-
',x,cos(x),'g--')
» legend('sine','cosine');
» grid
» xlabel('x');
» ylabel('functions');
» title('Test of multi-plot option in Matlab');
```

The result is the following plot:



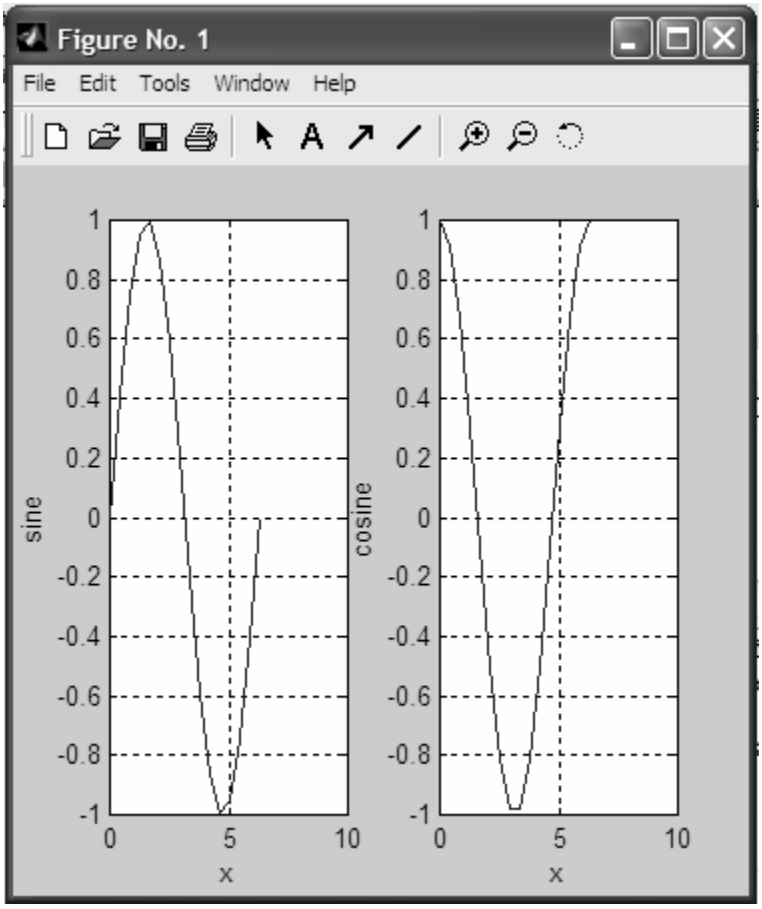
Note that the plot command with the same three options repeated twice generates a graph with two curves. This can be extended to fit your needs. Furthermore, the legend command allows one to generate the legend for this graph which can be positioned freely by the user by just clicking and dragging it over the graph, and releasing the mouse button when it is positioned as desired.

Each plot command erases the previous graphics window (the Figure No. 1) and draws on it. If you wish, you can send plot on the same window by first using the hold command and later sending plot to it with the plot command. The hold command can be switched off by using 'hold off' when desired.

Subplots

Let us now consider a different situation. We want to plot both sine and cosine functions again in the 0 to 2π range but on separate graphs. If we issue two separate plot commands, the previous graph is erased. If we use hold, then essentially, it is multiplot which you do not want. You want to plot these functions on two graphs placed next to each other. This is done with the help of subplot command, which splits the graphics window in to mxn array of sub-plot sections. Here, we create 1x2 panels (one row, two columns):

```
» N=15;h=2*pi/N; x=0:h:2*pi;
» subplot(122);plot(x,cos(x));xlabel('x');
  ylabel('cosine');grid
» subplot(121);plot(x,sin(x));xlabel('x');
  ylabel('sine');grid
```



The first subplot command picks the first column of this panel and plots the sine function in it. The second picks the second column and plots the cosine function in it. In this way, the graph is constructed.

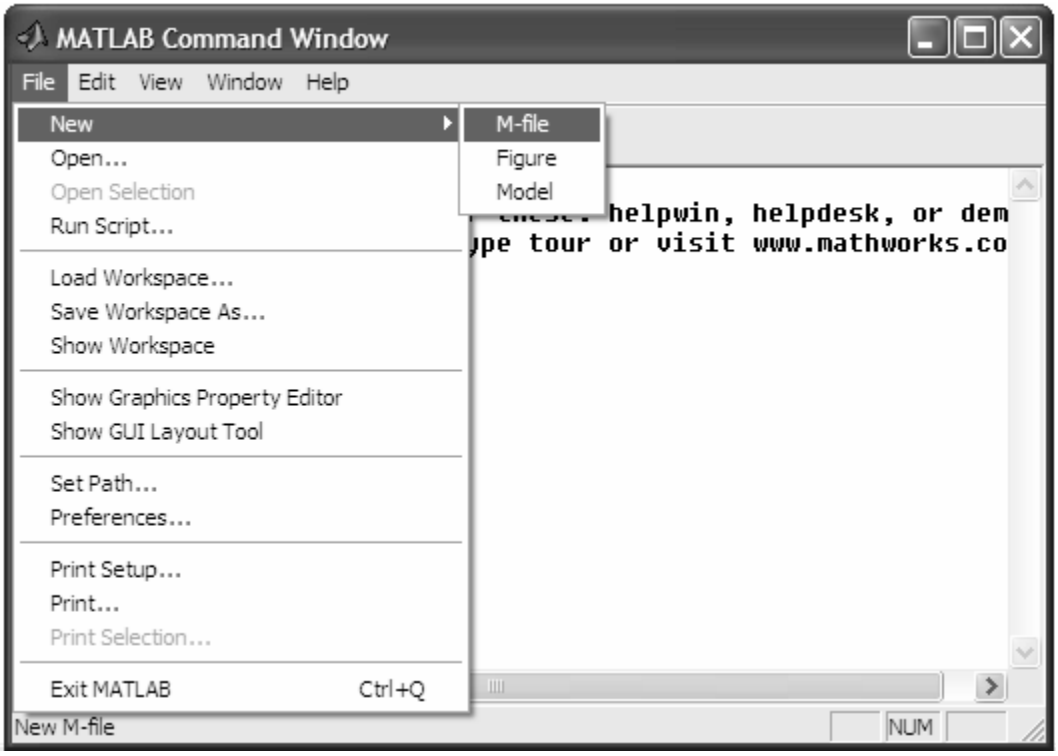
Axes Control

The axes of the graph can be controlled by the user with the help of axis command which accepts a row vector composed of four components. The first two of these are the minimum and the maximum limits of the x-axis and the last two are same for the y-axis. Matlab also allows users to set these axes with 'equal', 'auto', 'square' and 'normal' options. For example axis('auto') will scale the graph automatically for you. Similarly, axis([0 10 0 100]) will scale the graph with x-axis in [0, 10] range and y-axis in [0, 100] range.

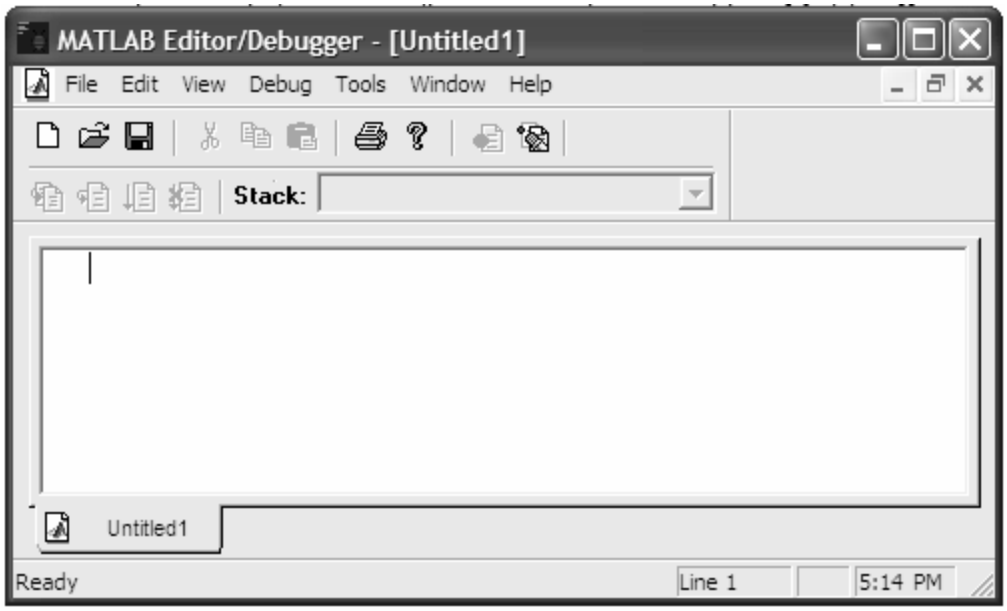
Scripts

Some times, it becomes necessary to give a set of Matlab commands again. In such cases, it becomes tedious to type-in every thing. Matlab offers a

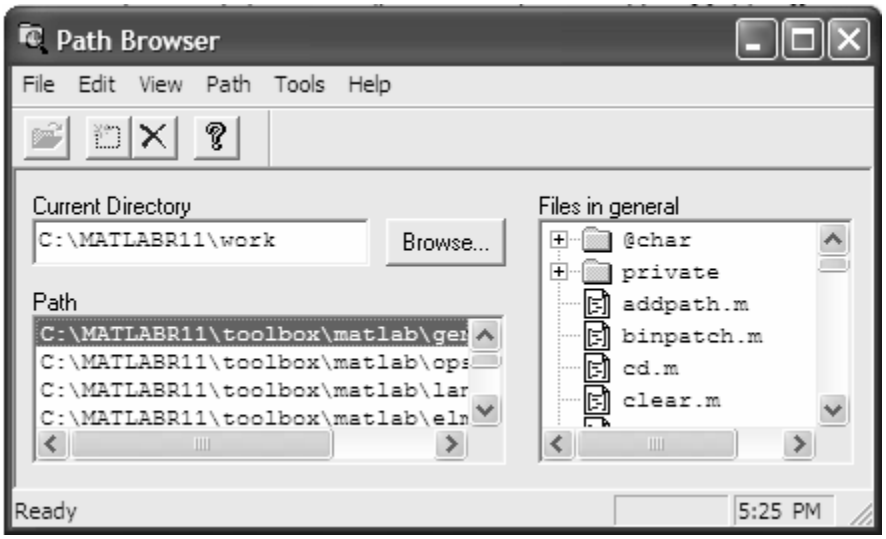
convenient way to handle this situation. The user can save the desired set of commands in a Matlab script file. It can have any legal name and it must have extension ‘m’ which stands for Matlab-script. It is standard ASCII text file. Matlab has built-in m-file editor designed specifically for this purpose. This can be accessed using File menu:



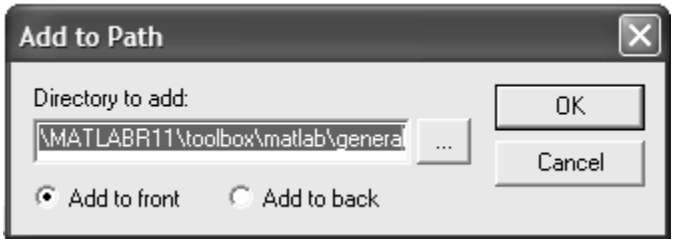
By clicking at the File—New—m file item, the m-file editor window pops up:



Here, one can type-in desired set of commands and save it. The default directory for these files is already in the search path of Matlab. If you wish to save the file into a directory of your own choice, please do not forget to include it in the Matlab search path. This can be clicking on the file—select path menu item which will open the path browser for you:



You can use the menu item path—add to path to add the directory of your choice to the Matlab path:



By clicking on the button with … on it, the directory browser dialog can be opened and by clicking on the desired directory, you can select the directory to be added. After that, just press OK button to add the directory to the path.

After saving the script file in a directory in Matlab path, the commands inside it can be invoked by just typing the name of the file (without the **.m** extension).

Working with Vectors and Matrices

Vectors can be manipulated in various ways. A scalar can be added to vector elements in Matlab using .+ notation:

» A=[1 2];

» B=2.+A

B =

3 4

Note the use of the 'dot' before the '+' sign which means apply it on element basis. In exactly same way, division, multiplication, subtraction and raised to the power operations can be carried out. For example, let us raise each element of a matrix to power 2 using the 'dot' notation:

```
>> B=[2 3 4; 5 4 6; 1 3 2]; B.^2
ans =
     4     9    16
    25    16    36
     1     9     4
>> B^2
ans =
    23    30    34
    36    49    56
    19    21    26
```

In the first case, each element of the matrix B has been raised to power 2. For this purpose, the dot notation was used. In the second case, the same matrix

has been raised to power 2 which is essentially B*B operation. Now, let us carryout the dot or inner product of a row U and a column vector V:

$$U = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}; \quad V = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
>> U=[1 2 3]; V=[1;2;3]; U*V
ans =
    14
```

Clearly, the result is 1+4+9 = 14; a scalar quantity. Now, let us change the order of multiplication. In this case, the result is expected to be a matrix:

```
>> V*U
ans =
     1     2     3
     2     4     6
     3     6     9
```

Now, let us compute the Euclidean norm of a vector which is defined as:

$$\|U\| = \sqrt{\sum_{i=1}^{3} |u_i|^2}$$

It can be obtained by the $\sqrt{U.U'}$; where U' is its complex conjugate transpose. Also, Matlab has a built-in function called norm, which carries out this operation for us:

```
>> sqrt(U*U'), norm(U)
ans =
   3.7417
ans =
   3.7417
```

The first computation returns the value of sqrt(U*U') as 3.7417 and exactly the same result is obtained using the norm function.

Now, let us compute angle between two vectors X and Y where:

$$X = \begin{bmatrix} 7 & 5 & 9 \end{bmatrix}; \quad Y = \begin{bmatrix} 15 & 3 & 7 \end{bmatrix}$$

In Matlab, we will compute the lengths of these vectors using the norm function, and divide the inner product of X and Y with these lengths, the result will be cosine of the angle and finally, using the acos function, we will get the final answer. Mathematically:

$$\theta = \cos^{-1}\left(\frac{X.Y}{\|X\|\|Y\|}\right)$$

```
>> X=[7 5 9]; Y=[15 3 7];
>> theta = acos(X*Y'/(norm(X)*norm(Y)))
theta =
    0.5079
```

Here, first both vectors have been initialized. Next, we apply the formula. The important thing to note in this case was the fact that since both vectors were defined as row vectors, we had to convert the 'Y' vector into a column vector by using transpose in order to compute the inner product.

Hadamard product

Although not in common use, the Hadamard is defined in mathematics as element by element product of two vectors of identical lengths and the result is again a vector of same length. For example if:

$$U = [u_1 \quad u_2 \quad \cdots \quad u_n]; \qquad V = [v_1 \quad v_2 \quad \cdots \quad v_n]$$

then the Hadamard dot product is defined as:

$$U.V = [u_1 v_1 \quad u_2 v_2 \quad \cdots \quad u_n v_n]$$

In Matlab, the Hadamard dot product is obtained using .* operator. For example, if U=[1 3 4 7] and V=[8 3 9 2], then their Hadamard dot product is given by:

```
>> U=[1 3 4 7]; V=[8 3 9 2]; U.*V
ans =
     8     9    36    14
```

Tabulation of Functions

The functions used in Matlab apply on element by element basis. In order to test it, let us prepare a table of the values of sine, and cosine for values of angles ranging from 0 to pi in steps of pi/10. For this, first, we construct a column vector of values of angles and call it X:

```
>> X=[0:pi/10:pi]'
X =
         0
    0.3142
    0.6283
    0.9425
    1.2566
    1.5708
    1.8850
    2.1991
    2.5133
    2.8274
    3.1416
```

Now, we use the two trigonometric functions with x as argument:

```
>> [X sin(X) cos(X)]
ans =
         0         0    1.0000
    0.3142    0.3090    0.9511
    0.6283    0.5878    0.8090
    0.9425    0.8090    0.5878
    1.2566    0.9511    0.3090
    1.5708    1.0000    0.0000
    1.8850    0.9511   -0.3090
    2.1991    0.8090   -0.5878
    2.5133    0.5878   -0.8090
    2.8274    0.3090   -0.9511
    3.1416    0.0000   -1.0000
```

which shows clearly that the functions actually apply on each element of the column vector. The first column in the above output is X, second is sin(X) and third is cos(X).

In order to test it for another case, let us try finding the limiting value of sin(y)/y for y approaching zero. The answer should be 1.0 and in Matlab, we first define a range of values of y:

```
>> y=[10 1 0.1 0.01 0.001];
```

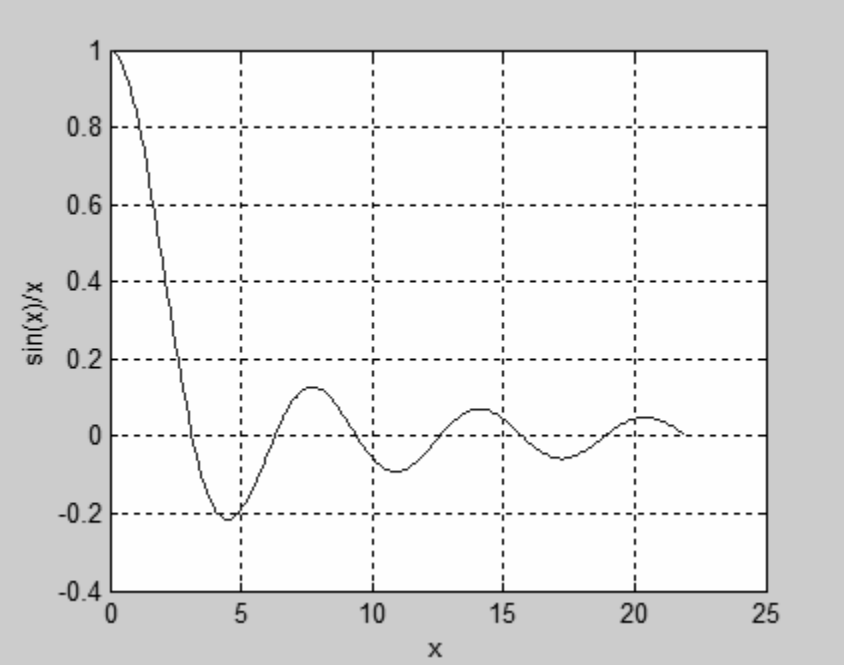
then we apply the expression. Note that the sine function will apply on element by element basis but the division must be forced to be carried out also on element by element basis which can easily be done using the ***dot*** in front of the division operator:

```
>> sin(y)./y
ans =
   -0.0544    0.8415    0.9983    1.0000    1.0000
```

and it is seen clearly that the limiting value is indeed 1.0 as y becomes smaller and smaller. In order to view things graphically, we first define a range of values of x from near 0 to 7π in steps of $\pi/10$. Then, it is plotted with grid-on state:

```
>> x=[0.0001:0.1:7*pi];plot(x,sin(x)./x);
xlabel('x'); ylabel('sin(x)/x'); grid on
```

which is seen as:



The function clearly approaches 1.0 as 'x' becomes smaller and smaller. In the range of values o 'x', zero was avoided otherwise Matlab gives a divided by zero error message.

Working with Matrices

Defining Matrices

A matrix is essentially a two dimensional array of numbers composed of rows and columns. A matrix can be entered in Matlab in either of the following three ways:

(a) Using carriage return key:

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

(b) Using semicolons to indicate the next line:

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

(c) Using the range notation with semicolon:

```
>> A=[1:3; 4:6; 7:9];
```

Some matrices can be defined simply using functions. For example, the zeros function defines a matrix with all entries zeros, the function ones defines matrix filled with ones and rand defines a matrix with all entries random numbers in the [0,1] range:

```
>> zeros(3)
ans =
     0     0     0
     0     0     0
     0     0     0
>> ones(3)
ans =
     1     1     1
     1     1     1
     1     1     1
>> rand(3)
ans =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214
```

The argument in each case is the size of the matrix. The same functions can also be used for defining some non-square matrix:

```
>> rand(3,4)
ans =
    0.4447    0.9218    0.4057    0.4103
    0.6154    0.7382    0.9355    0.8936
    0.7919    0.1763    0.9169    0.0579
```

In this case, one needs to supply two arguments, first for the number of rows and second for the number of columns.

Size of Matrices

The size function returns the size of any matrix. For example, let us define a zero matrix A with size 135x243:

```
>> A=zeros(135,243); size(A)
ans =
   135   243
```

The size function returns the row and column count of this matrix.

The Identity Matrix

The identity matrix having the form:

$$I = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}$$

can be defined in Matlab with the help of the 'eye' function with argument representing the size of the matrix:

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

The 'eye' function generates a 3x3 identity matrix when the argument is 3.

Transpose

As discussed before, the complex conjugate transpose of a matrix can be obtained by using the apostrophe. In order to obtain the regular transpose, one should use the dot-apostrophe. For example:

```
>> T=[1-j 1+j
      1+j 1-j];
>> T'
ans =
   1.0000 + 1.0000i   1.0000 - 1.0000i
   1.0000 - 1.0000i   1.0000 + 1.0000i
>> T.'
ans =
   1.0000 - 1.0000i   1.0000 + 1.0000i
   1.0000 + 1.0000i   1.0000 - 1.0000i
>>
```

First we have defined the complex matrix T. The T' is its complex conjugate transpose—the complex conjugate of each element has been used after

transposing the matrix. The T.' has simply transposed the matrix T and no complex conjugate has been taken in this case.

Diagonal Matrix

The diagonal matrix is similar to the identity matrix that both have off-diagonal elements all zeros. In order to generate the diagonal matrix, first a row vector containing the diagonal elements is required. For a square matrix of size nxn, the diagonal will be of size n. This row vector is used as an argument to the 'diag' function:

```
>> d=[1 2 3 4];
>> D = diag(d)
D =
     1     0     0     0
     0     2     0     0
     0     0     3     0
     0     0     0     4
```

Note that the diagonal matrix has all diagonal entries picked from the vector 'd'.

The same function returns the entries of the diagonal for any given matrix. For example, let us take the random matrix R:

```
>> R = rand(4)
R =
    0.3529    0.2028    0.1988    0.9318
    0.8132    0.1987    0.0153    0.4660
    0.0099    0.6038    0.7468    0.4186
    0.1389    0.2722    0.4451    0.8462
>> diag(R)
ans =
    0.3529
    0.1987
    0.7468
    0.8462
```

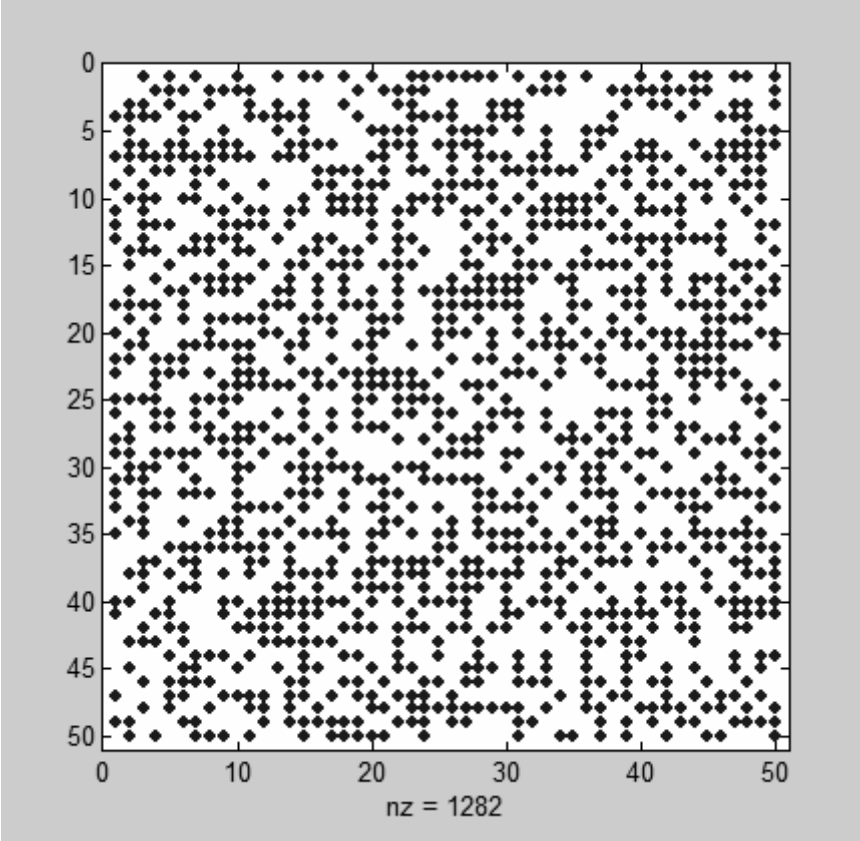
First, the random matrix R has been defined. Then, the function 'diag' with argument R returns the diagonal elements of R.

Spy Function

The sparsity pattern of a matrix is revealed with the help of the spy function which produces a graphical visualization of given matrix. For example, let us create a random matrix and convert it to integer values after adding 0.5 to all of its elements. The size of this matrix is 50x50:

```
>> B=fix(0.5+rand(50,50));spy(B)
```

The rand(50,50) generates a random matrix with elements in [0,1] range. By adding 0.5 to it, we shift the entire matrix range to [0.5, 1.5]. This means that now, about half of the entries and below 1 and the remaining half above 1. When fix function is applied, the matrix is converted to all entries in 0, 1 values. It is expected that roughly half of the entries will be 1 and remaining half zeros. The spy command gives the visualization of this matrix:



The value of 'nz' (number of zero entries) is 1282 which is roughly half of 50x50. Larger the nz value, more sparse the matrix.

Sections of Matrices

The range operation is used for extracting section of a given matrix. For example, let us consider a diagonal matrix D with diagonal entries ranging from 1 to 16. We are going to extract a 4x4 matrix B from it which have diagonal entries starting from 9.

```
>> d=[1:16]; D=diag(d); B = D(9:13, 9:13)

B =
```

```
      9      0      0      0      0
      0     10      0      0      0
      0      0     11      0      0
      0      0      0     12      0
      0      0      0      0     13
>>
```

As you can see, the required portion of the matrix has been extracted.

Product of Matrices

The * operator multiplies two matrices if they are conformable for multiplication while the .* operator is strictly for multiplication that is on element-by-element basis:

```
>> A=[1 2 3; 4 5 6]; B=[1 2; 3 4]; B*A, B.*B
ans =
     9    12    15
    19    26    33
ans =
     1     4
     9    16
```

In the above case, first, the two matrices are defined. Then, the B*A computes the standard dot or the inner product of the two matrices while the .* operation calculates the element by element product of the matrix B.

Matlab Programming

Matlab offers quite straight forward programming language of its own which is somewhat similar to the C-language in many respects. But there are certainly some differences between the two language as well. Here, we will start with some elementary stuff and later one can build on it.

For-Loops

For carrying out any repeated task, loops are needed. In Matlab, this is done using for-command which has the generic syntax:

For counter= legal list of values

- - statements to be executed repeatedly within this loop

end

As a simple example, let us define a row vector having 7 random values:

```
>> R=rand(1,7)
R =
0.3784 0.8600 0.8537 0.5936 0.4966 0.8998    0.8216
```

Next, we would like to find the sum of all entries in R. For this purpose, we define a variable sum and initialize it to be zero. Then, we construct a loop that executes exactly seven times with the help of a counter having range of values 1 to 7. Inside this loop, we simply add the various elements of R in turn to sum.

```
>> sum=0;
>> for i=1:7
     sum = sum + R(i);
   end
```

Now, we can find the average value of the elements of R by dividing the sum with 7:

```
>> avg = sum/7
avg =
    0.7005
>> sum
sum =
    4.9036
```

The answer is 0.7005 while the value of sum was 4.9036.

As another example of the use of loops, let us generate and find sum of the first 100 integers.

$$1 + 2 + 3 + \ldots + 100$$

Again the method is going to be the same. We will use a variable sum to do the summation within the loop:

```
>> sum = 0;
   for i=1:100
      sum=sum + i;
   end;
   sum
>>sum =
        5050
```

In this case, the result is 5050 which is clearly the true value of this sum.

Next, let us use Matlab to compute the Fibonnaci sequence:

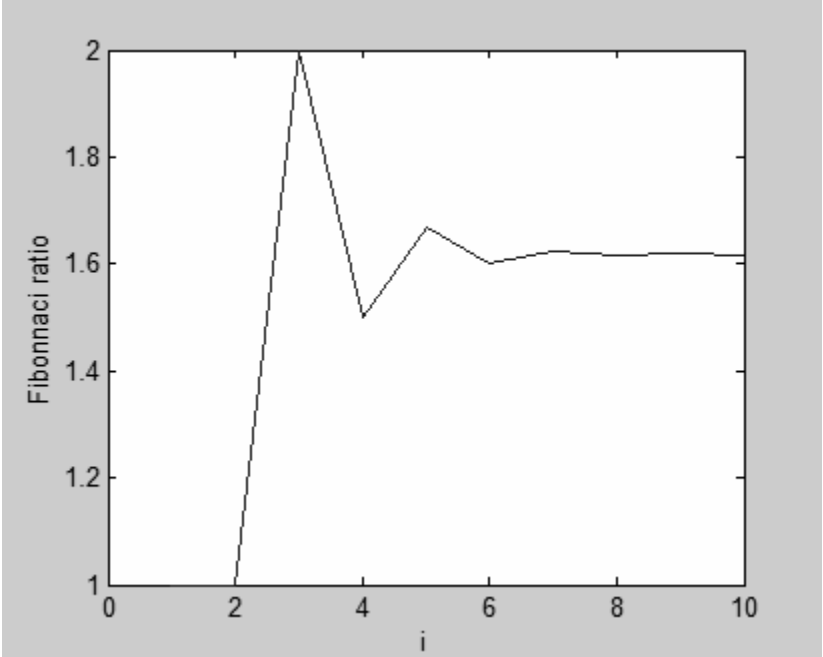
1, 1, 2, 3, 5, 8, 13, 21, . . .

which starts with two elements 1, 1; and each next element is just sum of the two previous elements. The ratio of two consecutive elements of this sequence approaches a fixed number which is also found in many aesthetically pleasing

structures e.g., the ratio of height to width in doors of some buildings. We do it by using the following m-script file in Matlab:

```
f(1)=1; f(2)=1;
ratio(1)=1; ratio(2)=1;
for i=3: 10
    f(i)=f(i-1)+f(i-2);
    ratio(i) = f(i)/f(i-1);
end
plot(ratio); xlabel('i');
ylabel('Fibonnaci ratio');
```

The result is the following plot, which clearly shows that in ten iterations, the ratio approaches the value 1.6176 which is close to the actual value 1.618



Logical Expressions

The following operators are used in logical expressions of various type:

Operator	Meanings	Operator	Meanings
<	Less than	>	Greater than
<=	Less or equal	>=	Greater or equal
==	Equal to	~=	Not equal

The resulting value is either true or false. Various logical expressions can further be combined using:

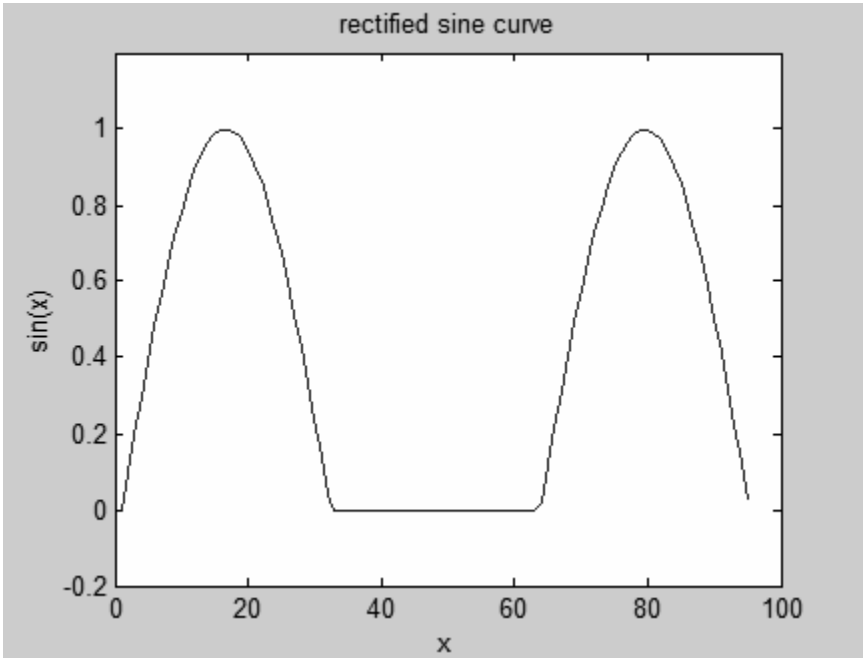
& meaning 'and' , | meaning 'or' , ~ meaning 'not'

Note that true evaluates as 1 and false as 0.

In order to apply it, let us construct a rectifier with it. Let us construct a range of values of x from 0 to 3π in steps of 0.1. Next, we compute the corresponding values of the sine function for these values of x as angles. Finally, we construct a vector of values of sin(x) which picks only the positive values of sin(x). The corresponding m-script file is:

```
x=[0:0.1:3*pi];
y=sin(x);
Z=(y>0).*y;
plot(Z)
xlabel('x');ylabel('sin(x)');
title('rectified sine curve');
```

and the corresponding output is:



While loop

This loop keeps on repeating while a certain logical condition remains true. The loop is terminated when the logical condition becomes false. For

example, the sum of integers from 1 to 100 can be found with the help of following code:

```
Sum = 0;
I = 1
while I <= 100
  Sum = Sum + I;
   I = I+1;
end
```

Note that the loop counter needs to be incremented inside the loop in this case.

Conditional Programming

The 'structured-if' is used for this purpose. It has the following generic form:

```
if  logical-expression-1
   statements executed when expression-1 is true
elseif logical-expression-2
  statements executed when expression-2 is true
elseif logical-expression-3
  statements executed when expression-3 is true
. . .number of elseif portions repeated as needed
else
   statements executed when
    none of logical expressions is true
end
```

Note that the elseif portion and the else portion are optional and should be used only when they are needed. Also, the else part should be the last part in this structure when needed.

As an example, let us initialize a variable with some value and test if it is even or odd. For this check, we will compare the result of integer division of the number with 2 and later multiplied with 2, with the actual number. For even values, the original number is obtained:

```
N=input('Please enter an integer:');
if fix(N/2)*2==N
    integer_type='even';
else
    integer_type='odd';
end
integer_type
```

First, the m-script prompts the user with text: "Please enter an integer:" which appears in the work area of the Matlab environment. User types-in some integer and this number is assigned to the variable 'N'. Then, the integer division of N with 2 is carried out. Note that if it is not even, then, there will be some fractional part which is dropped-off by the fix function. Next, the result is multiplied with 2 and it is then compared with 'N'. The two values will be same if it was an even integer, in which case, the text variable is assigned string-value 'even', otherwise, it is assigned 'odd'. Finally, the value of this text-variable is printed on screen. A typical execution gives:

```
Please enter an integer:786
integer_type =
even
>>
```

Function m-Scripts

In Matlab, all computations are normally done with the help of functions which are written in Matlab scripting language. The basic structure of a function is the following:

```
Function output_values = name(input_values)
% comments echoed when 'help name' used
- - -  body of the function - - -
end
```

The function m-script must be written in a file with name as the name of the function and extension .m.

As an example, let us write a function script quadratic which accepts three values as the coefficients a, b and c of quadratic equation, computes and returns the values of the corresponding roots as x_1 and x_2:

Equation: $$ax^2 + bx + c = 0$$

Roots: $$x_1, x_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Contents of the file: quadratic.m

```
function [x1,x2]=quadratic(a,b,c)
%Funtion quadratic
```

```
%solves the quadratic equation:
%   a x^2 + b x + c = 0
% using a, b, c coefficients as input
% and returns the values of two roots
% as x1, x2
%-----------------by Dr. Sikander Majid
%                    [January 2003]

d = b^2-4*a*c;
x1 = (-b+sqrt(d))/(2*a);
x2 = (-b-sqrt(d))/(2*a);
end
```

when help on the function script is invoked, we get:

```
>> help quadratic

 Funtion quadratic
 solves the quadratic equation:
    a x^2 + b x + c = 0
  using a, b, c coefficients as input
  and returns the values of two roots
  as x1, x2
 -----------------by Dr. Sikander Majid
                    [January 2003]
```

and when, we test it for the following equation:

$$342\,x^2 + 127\,x + 8720 = 0\,,$$

we get:

```
>> [x1,x2]=quadratic(342,127,8720); [x1, x2].'
ans =
  -0.1857 + 5.0460i
  -0.1857 - 5.0460i
```

We invoke the function with its name and supply the required arguments to it. The output of the function is stored in the row vector [x1, x2]. Then, this vector is printed as a column using the regular transpose found using the dot-apostrophe. The roots are complex and conjugate of each other.

Return Statement

Normally, the function 'returns' values when the 'end' statement in the function is reached. If one wishes to do it earlier, the return statement can be used and it will force the function to return values at that point.

Recursive Programming

Sometimes, it is possible to invoke the function from within it self to carry-out some computation. As a simple example, let us try computing the value of factorial of an integer. Let us write a function script for this purpose and name it 'factorial'. The input to this function is going to be 'n' an integer and output will be the corresponding value of the factorial of that integer. For simplicity, let us assume that user supplies only positive integer as argument. Now, one can compute factorial in the following way:

Recursive expression: Factorial(n) = n*factorial(n-1)

It is to be repeated till the argument of function becomes 1. Its Matlab function script implementation is given below:

```
function value=factorial(n)
%Funtion factorial
%computes the factorial of an integer
%in recursive manner
% input is an integer and
% returns the values factorial as integer
%-----------------by Dr. Sikander Majid
%                  [January 2003]

if n==1
    value = 1;
    return;
end
value = n*factorial(n-1);
end
```

When its help is invoked, we get:

```
>> help factorial

 Funtion factorial
 computes the factorial of an integer
 in recursive manner
  input is an integer and
```

```
  returns the values factorial as integer
 -----------------by Dr. Sikander Majid
                    [January 2003]
```

It returns the value of factorial quickly :

```
>> factorial(3)
ans =
     6
```

Matlab is mostly a collection of functions that can be invoked when needed. All these functions are simply m-scripts and can be modified at will. However, it is strongly recommended that user should copy the original functions to separate work files and edit/modify them. In this way, the working of Matlab will not be compromised in case the modified version does not perform the required task.

Function Visualization

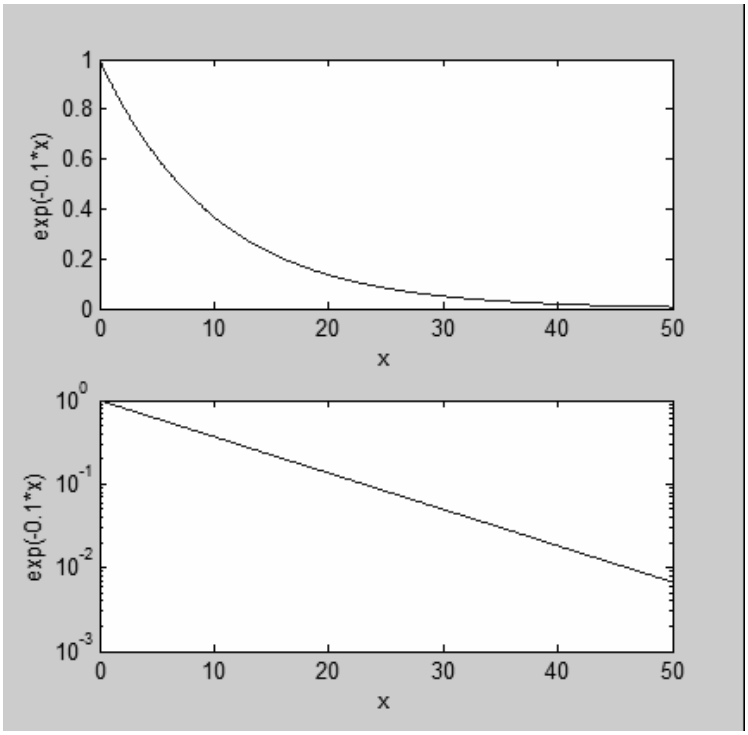
Now, we will look at various ways in which one can visualize various functions. Let us start with function of single variable. In this case, the plot command has already been introduced.

Semilog Plot

Let us plot exponential decay of a radioisotope. This can be done using linear-linear plot. If y-axis is chosen as semilog, the the graph is a straight line. In order to do this, we can use the function 'semilogy':

```
x=[0:0.1:50];y=exp(-0.1*x);
subplot(211),plot(x,y);
    xlabel('x');ylabel('exp(-0.1*x)');
subplot(212);semilogy(x,y);
    xlabel('x');ylabel('exp(-0.1*x)');
```

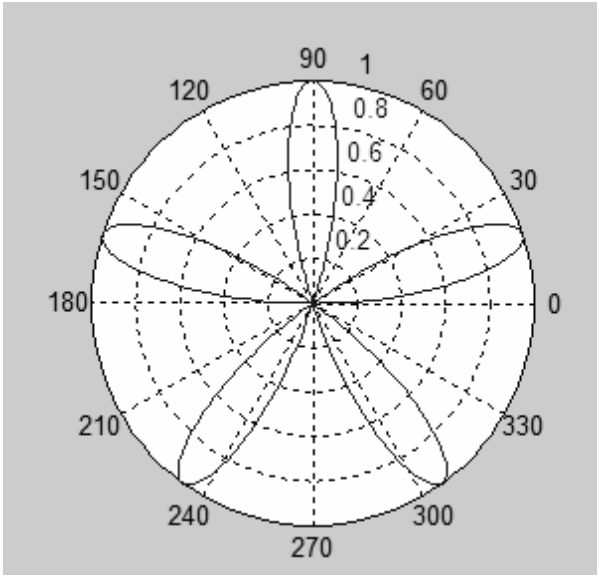
First, we generate uniformly filled array of numbers from 0 to 50 in steps of 0.1. Then, we compute the corresponding vector of values of y which is just the corresponding value of exp(-0.1*x). Then, we show it graphically using linear-linear graph and then using the 'semilogy' function. In the first plot, the graph is exponentially decreasing curve while in the second case, it is a straight line as expected.



Polar plot

This function accepts a range of values of the angle 'theta' and the corresponding values of the radius 'rho' and shows them in polar plot. For example, let us plot sin(5θ) using it.

```
>> theta=[0:0.01:pi]; rho=sin(5*theta);
   polar(theta,rho)
```

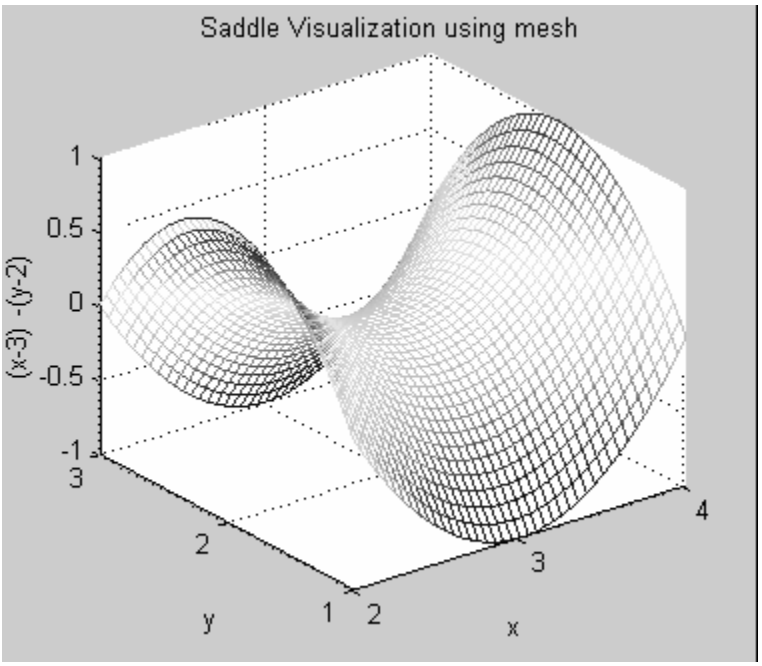


Mesh Plot

In this case, we visualize a function of two variables and as previously, we need the vector of values of the two independent variables, for which the corresponding values of the function are to be computed. Let us visualize the function: $Z = (x-3)^2 - (y-2)^2$ in the range $x \in [2,4], y \in [1,3]$. The corresponding Matlab commands are:

```
[x,y]=meshgrid(2:0.05:4,1:0.05:3);
z=(x-3).^2-(y-2).^2;
mesh(x,y,z);
xlabel('x'); ylabel('y');
zlabel('(x-3)^2-(y-2)^2');
title('Saddle Visualization using mesh');
```

and the corresponding output is:



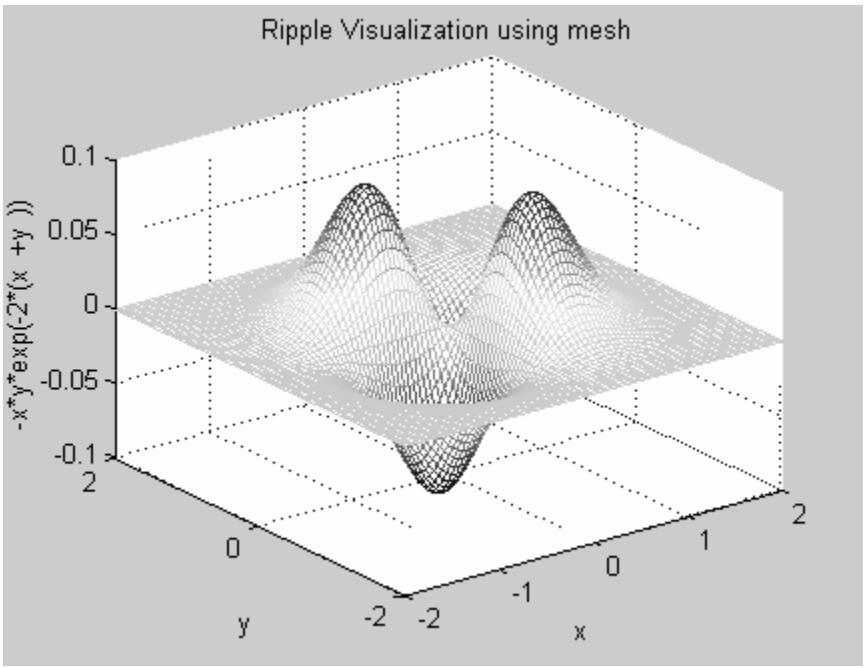
Next, we show the visualization of the function:

$$z = -x\,y\exp\left(-2\left(x^2 + y^2\right)\right)$$

For this purpose, we will use surface as well as contour plots of various types. The corresponding m-script is:

```
[x,y]=meshgrid(-2:0.05:2,-2:0.05:2);
z=-x.*y.*exp(-2*(x.^2+y.^2));
mesh(x,y,z);
xlabel('x'); ylabel('y');
zlabel('-x*y*exp(-2*(x^2+y^2))');
title('Ripple Visualization using mesh');
```

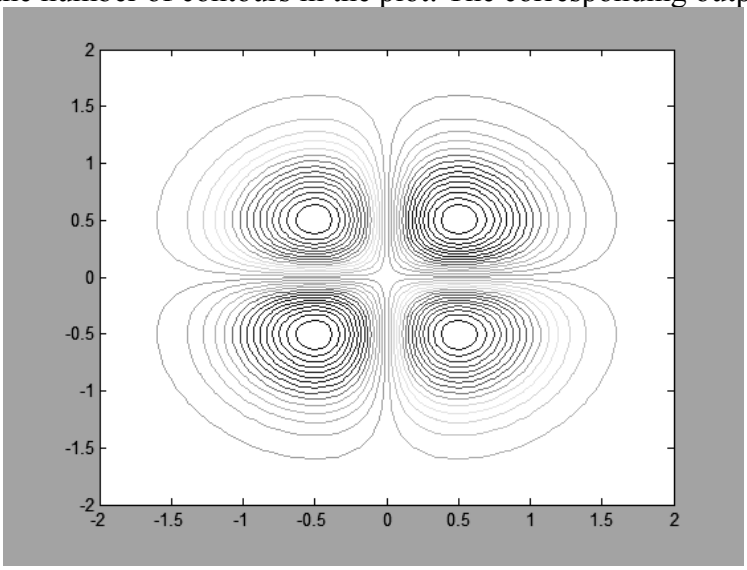
and the corresponding output is:



Now, we will draw contours for this function using:

```
Contour(x,y,z,30);
```

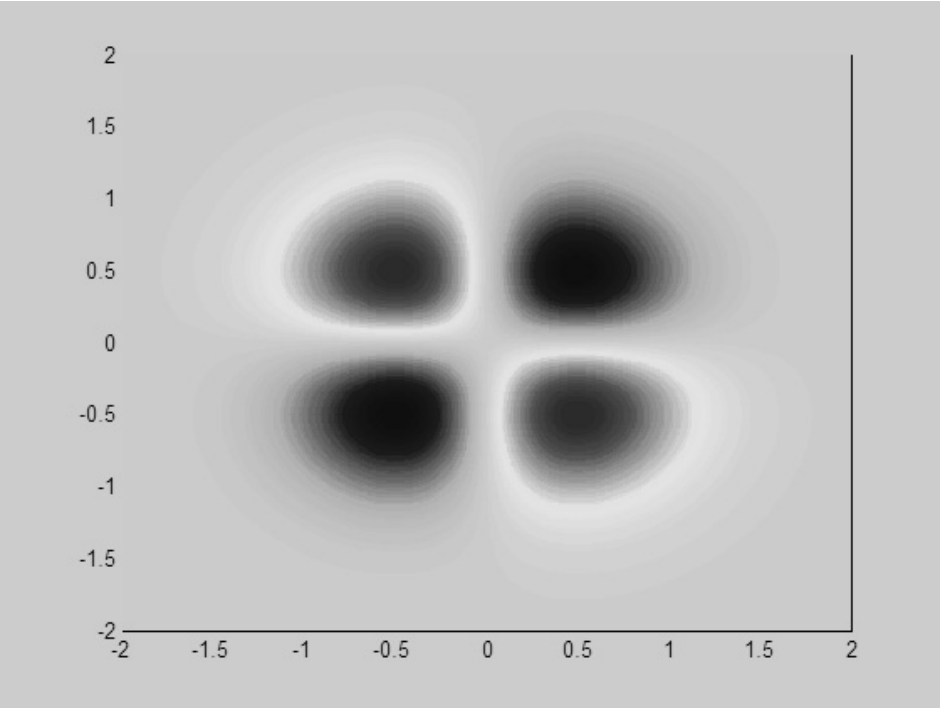
Where, 30 is the number of contours in the plot: The corresponding output is:



the following command gives a different visualization:

```
contourf(x,y,z,100);  shading flat;
```

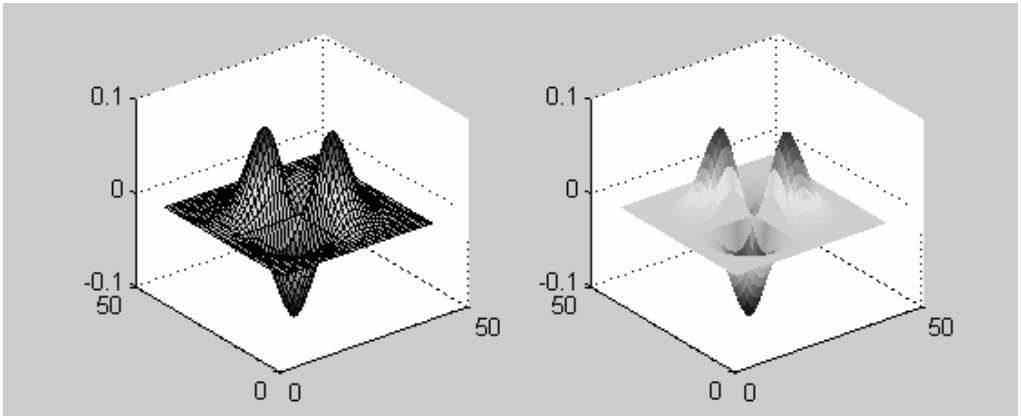
The result is:

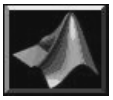


With the command:

```
subplot(121);surf(z);
subplot(122);surf(z);shading flat,colormap('jet');
```

the visualization is following pair of surfaces. The left surface has grid-lines visible while on the right surface, they are removed. Also, we have used 'jet' color map for the visualization.





Elapsed Time

In Matlab, various procedure may require different amount of computational time. This time can be obtained by using the 'tic', 'toc' pair of functions. The first function sets the starting time as zero and the second function returns the time elapsed since last call to 'tic' function. As an example, we wish to compute the time required for adding 2000 integers starting with 1. the corresponding Matlab code is saved as the m-script file called tictoc.m:

```
tic;
sum = 0;
for i=1:2000
    sum = sum+i;
end
sum
toc
```

When executed, it yields:

```
>> tictoc
sum =
     2001000
elapsed_time =
    0.0150
```

This value is in seconds. If you wish to compute the cpu-time used, then the function is cputime and the commands are:

```
t=cputime; your_operation; cputime-t
```

Please be ***cautioned*** that for longer operations, the timing calculations may wrap-around the time-storage and the results may become unreliable!

Managing commands and functions

Command	Meanings
help	On_line documentation_
doc	Load hypertext documentation
lookfor	Keyword search through the help entries
which	Locate functions
demo	Run demos_

Managing variables and the workspace

Command	Meanings
who	List current variables
whos	List current variables long form
load	Retrieve variables from disk
save	Save workspace variables to disk
clear	Clear variables and functions
from	memory
size	Size of matrix
length	Length of vector
disp	Display matrix or text

Working with files and the operating system

cd	Change current working directory
dir	Directory listing
delete	Delete file
!	Execute operating system command
unix	Execute operating system command & return result
diary	Save text of MATLAB session_

Controlling the command window

Command	Details
cedit	Set command line edit/recall facility parameters
clc	Clear command window
home	Send cursor home
format	Set output format
echo	Echo commands inside script files
more	Control paged output in command window
quit	Terminate MATLAB

Matrix analysis

Command	Details
cond	Matrix condition number
norm	Matrix or vector norm
rcond	LINPACK reciprocal condition estimator
rank	Number of linearly independent rows or columns
det	Determinant
trace	Sum of diagonal elements
null	Null space
orth	Orthogonalization_
rref	Reduced row echelon form

Linear equations

Command	Details
\ and /	Linear equation solution; use “help slash”
chol	Cholesky factorization
lu	Factors from Gaussian elimination
inv	Matrix inverse
qr	Orthogonal_ triangular decomposition_
qrdelete	Delete a column from the QR factorization
qrinsert	Insert a column in the QR factorization
nnls	Non_negative least_ squares
pinv	Pseudoinverse
lscov	Least squares in the presence of known covariance_

Eigenvalues and singular values

Command	Details
eig	Eigenvalues and eigenvectors
poly	Characteristic polynomial
polyeig	Polynomial eigenvalue problem
hess	Hessenberg form
qz	Generalized eigenvalues
rsf_csf	Real block diagonal form to complex diagonal form
cdf_rdf	Complex diagonal form to real block diagonal form
schur	Schur decomposition
balance	Diagonal scaling to improve eigenvalue accuracy
svd	Singular value decomposition

Matrix functions

Comand	Details
expm	Matrix exponential
expm1	M_file implementation of expm
expm2	Matrix exponential via Taylor series
expm3	Matrix exponential via eigen-values and eigenvectors
logm	Matrix logarithm
sqrtm	Matrix square root
funm	Evaluate general matrix function