MATLAB

INTRODUCTION TO MATLAB

The name MATLAB stands for matrix laboratory. MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

Typical uses include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

BASIC MATLAB PROGRAMMING

Compared to other programming languages, MATLAB offers some advantages to the user, for example:

- Automatic selection of variable type.
- Easy computation of complex numbers.
- Interpreted language.
- Integrated debugger.

MATLAB ENVIRONMENT

This is the set of tools and facilities that you work with as the MATLAB user or programmer. It includes facilities for managing the variables in your workspace and importing and exporting data. It also includes tools for developing, managing, debugging, and profiling M-files, MATLAB's applications.

MATLAB WINDOWS

THE COMMAND WINDOW

The Command Window is the main window in which you communicate with MATLAB. The MATLAB interpreter displays a prompt (>>) indicating that it is ready to accept commands from you. It can be used like a calculator and you can also run programs you have written.

CLEARING THE COMMAND WINDOW

Use clc to clear the Command Window. This does not clear the workspace, but only clears the view. After using clc, you still can use the up arrow key to see the history of the commands, one at a time.

INTERRUPTING A RUNNING PROGRAM

You can interrupt a running program by pressing **Ctrl-c** at any time.

WORKSPACE WINDOW

The MATLAB workspace contains a set of variables (named arrays) that you can manipulate from the MATLAB command line. You can use the who and whos commands to see what is currently in the workspace. The who command gives a short list, while the whos command also gives size and data type information. To delete all existing variables from the workspace, enter clear.

COMMAND HISTORY WINDOW

Statements you enter in the Command Window are logged in the Command History. In the Command History, you can view previously run statements, and copy and execute selected statements.

EDITOR WINDOW

The editor window lets you write, edit, and troubleshoot MATLAB programs. Use the editor to create and modify M-files (MATLAB scripts). M-files allow you to save and execute multiple commands or entire programs with a single command line entry. Creating an m-file

• Open the MATLAB editor

• Type in the commands you want to execute

• Save the file in a location accessible to MATLAB (usually the MATLAB work directory or current working directory)

• In the MATLAB command window, type in the name of the file

to execute the commands

HELP WINDOW

• MATLAB Help is an extremely powerful assistance to learning MATLAB

• Help not only contains the theoretical background, but also shows demos for implementation

• MATLAB Help can be opened by using the HELP pull-down menu

• Any command description can be found by typing the command in the search field

• We can also utilize MATLAB Help from the command window

WORKING IN MATLAB

There exists two different possibilities to work in MATLAB:

• **By directly typing commands in the Command window.**

This enables to make operations step by step.

• **By programming in MATLAB's environment (*.m)**

MATLAB permits to execute commands that are previously stored in a file. This acts as a batch file, and commands are executed one by one. The file must have a '.m' extension. The file can be executed from the command window by typing the name of the file.

BASIC OPERATIONS ON MATRICES

• All operators in MATLAB are defined on matrices: +, -, *, /, ^, sqrt, sin, cos, etc.

• Element-wise operators defined with a preceding dot: .*, ./, .^

EXPRESSIONS

Expressions can be created using values, variables that have already been created, operators, built-in functions, and parentheses. For numbers, these can include operators such as multiplication, and functions such as trigonometric functions. An example of such an expression is:

```
>> 2 * sin(1.4)
ans =
1.9709
```

The format function

The default in MATLAB is to display numbers that have decimal points with four decimal places, as shown in the previous example. (The default means if you do not specify otherwise, this is what you get.) The format command can be used to specify the output format of expressions.

There are many options, including making the format short (the default) or long. For example, changing the format to long will result in 15 decimal places. This will remain in effect until the format is changed back to short, as demonstrated in the following.

```
>> format long
>> 2 * sin(1.4)
ans =
1.970899459976920
>> format short
>> 2 * sin(1.4)
ans =
1.9709
```

Operator precedence rules

Some operators have precedence over others. For example, in the expression 4+5 * 3, the multiplication takes precedence over the addition, so first 5 is multiplied by 3, then 4 is added to the result. Using parentheses can change the precedence in an expression:

```
>> 4+5 * 3
ans =
19
>> (4+5) * 3
ans =
27
```

Within a given precedence level, the expressions are evaluated from left to right (this is called associativity). Nested parentheses are parentheses inside of others; the expression in the inner parentheses is evaluated first. For example, in the expression 5-(6*(4+2)), first the addition is performed, then the multiplication, and finally the subtraction, to result in -31. Parentheses can also be used simply to make an expression clearer. For example, in the expression ((4+(3 *5))-1), the parentheses are not necessary, but are used to show the order in which the parts of the expression will be evaluated. For the operators that have been covered thus far, the following is the precedence (from the highest to the lowest):

( ) parentheses

^ exponentiation

*, /, \ all multiplication and division

+, - addition and subtraction

Constants

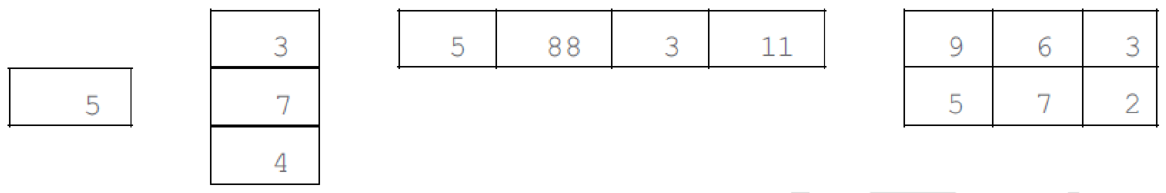
Variables are used to store values that might change, or for which the values are not known ahead of time. Most languages also have the capacity to store constants, which are values that are known ahead of time, and cannot possibly change. An example of a constant value would be pi, or π, which is 3.14159.... In MATLAB, there are functions that return some of these constant values, some of which include:

pi	3.14159. . . .
i	$\sqrt{-1}$
j	$\sqrt{-1}$
inf	infinity ∞
NaN	stands for “not a number” such as the result of 0/0

VECTORS AND MATRICES

Vectors and matrices are used to store sets of values, all of which are the same type. A vector can be either a row vector or a column vector. A matrix can be visualized as a table of values. The dimensions of a matrix are r×c, where r is the number of rows and c is the number of columns. This is pronounced “r by c.” If a vector has n elements, a row vector would have the dimensions 1×n, and a column vector would have the dimensions n×1.

A scalar (one value) has the dimensions 1×1. Therefore, vectors and scalars are actually just special cases of matrices. Here are some diagrams showing, from left to right, a scalar, a column vector, a row vector, and a matrix.



The scalar is 1×1, the column vector is 3×1 (three rows by one column), the row vector is 1×4 (one row by four columns), and the matrix is 2×3 (two rows by three columns). All of the values in these matrices are stored in what are called elements. MATLAB is written to work with matrices; the name MATLAB is short for “matrix laboratory.” Since MATLAB is written to work with matrices, it is very easy to create vector and matrix variables, and there are many operations and functions that can be used on vectors and matrices.

Creating row vectors

There are several ways to create row vector variables. The most direct way is to put the values that you want in the vector in square brackets, separated by either spaces or commas. For example, both of these assignment statements create the same vector v:

```
>> v =[1 2 3 4]
v =
    1   2   3   4
>> v = [1,2,3,4]
v =
    1   2   3   4
```

Both of these create a row vector variable that has four elements; each value is stored in a separate element in the vector.

The colon operator

If, as in the preceding examples, the values in the vector are regularly spaced, the colon operator can be used to iterate through these values. For example, 1:5 results in all of the integers from 1 to 5:

```
>> vec = 1:5
vec =
      1  2  3  4  5
```

Note that in this case, the brackets [ ] are not necessary to define the vector. With the colon operator, a step value can also be specified with another colon, in the form (first:step:last). For example, to create a vector with all integers from 1 to 9 in steps of 2:

```
>> nv = 1:2:9
nv =
   1  3  5  7  9
```

Creating column vectors

One way to create a column vector is to explicitly put the values in square brackets, separated by semicolons (rather than commas or spaces):

```
>> c= [1; 2; 3; 4]
c =
    1
    2
    3
    4
```

Creating matrix variables

Creating a matrix variable is simply a generalization of creating row and column vector variables. That is, the values within a row are separated by either spaces or commas, and the different rows are separated by semicolons. For example, the matrix variable mat is created by explicitly typing values:

```
>> mat = [4  3  1; 2  5  6]
mat =
        4    3    1
        2    5    6
```

There must always be the same number of values in each row. If you attempt to create a matrix in which there are different numbers of values in the rows, the result will be an error message, such as in the following:

```
>> mat = [3 5 7; 1 2]
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.
```

Iterators can also be used for the values in the rows using the colon operator. For example:

```
>> mat= [2:4; 3:5]
mat=
       2  3  4
       3  4  5
```

Useful Matrix Generators

MATLAB provides four easy ways to generate certain simple matrices. These are

zeros	a matrix filled with zeros
ones	a matrix filled with ones
rand	a matrix with uniformly distributed random elements
randn	a matrix with normally distributed random elements
eye	identity matrix

To tell MATLAB how big these matrices should be you give the functions the number of rows and columns. For example:

```
>> a = zeros(2,3)
a =
      0  0  0
      0  0  0
>> b = ones(2,2)/2
b =
    0.5000  0.5000
    0.5000  0.5000
>> u = rand(1,5)
u =
      0.9218  0.7382  0.1763  0.4057  0.9355
>> n = randn(5,5)
n =
      -0.4326   1.1909   -0.1867   0.1139  0.2944
      -1.6656   1.1892   0.7258   1.0668   -1.3362
      0.1253    -0.0376  -0.5883  0.0593  0.7143
      0.2877    0.3273   2.1832   -0.0956   1.6236
      -1.1465   0.1746   -0.136   -0.8323   -0.6918
>> eye(3)
ans =
        1  0  0
        0  1  0
        0  0  1
```

Subscripting

Individual elements in a matrix are denoted by a row index and a column index. To pick out the third element of the vector u type:

```
u =
      0.9218  0.7382  0.1763  0.4057  0.9355
```

```
>> u(3)
ans =
       0.1763
```

You can use the vector [1 2 3] as an index to u. To pick the first three elements of u type

```
>> u([1 2 3])
ans =
      0.9218 0.7382 0.1763
```

Remembering what the colon operator does, you can abbreviate this to

```
>> u(1:3)
ans =
      0.9218 0.7382 0.1763
```

You can also use a variable as a subscript:

```
>> i = 1:3;
>> u(i)
ans =
     0.9218 0.7382 0.1763
```

Two dimensional matrices are indexed the same way, only you have to provide two indices:

```
>> a = [1 2 3;4 5 6;7 8 9]
a =
    1  2  3
    4  5  6
    7  8  9
>> a(3,2)
ans =
      8
>> a(2:3,3)
```

```
ans =
       6
       9
>> a(2,:)
ans =
      4 5 6
>> a(:,3)
ans =
       3
       6
       9
```

The last two examples use the colon symbol as an index, which MATLAB interprets as the entire row or column. If a matrix is addressed using a single index, MATLAB counts the index down successive columns:

```
>> a(4)
ans =
      2
>> a(8)
ans =
      6
```

The colon symbol can be used as a single index to a matrix. Continuing the previous example, if you type a(:) MATLAB interprets this as the columns of the a-matrix successively strung out in a single long column:

```
>> a(:)
ans =
       1
       4
       7
       2
```

```
     5
     8
     3
     6
     9
```

End as a subscript

To access the last element of a matrix along a given dimension, use end as a subscript (MATLAB version 5 or later). This allows you to go to the final element without knowing in advance how big the matrix is. For example:

```
>> q = 4:10
q =
    4  5  6  7  8  9  10
>> q(end)
ans =
    10
>> q(end-4:end)
ans =
      6   7   8   9   10
>> q(end-2:end)
ans =
       8   9   10
```

Deleting Rows or Columns

To get rid of a row or column set it equal to the empty matrix [].

```
>> a = [1 2 3;4 5 6;7 8 9]
a =
    1  2  3
    4  5  6
    7  8  9
>> a(:,2) = []
a =
    1  3
    4  6
    7  9
```

Matrix Arithmetic

Matrix addition and subtraction are done element by element. Note that matrices must be of the same dimension for this to be valid (unless subtracting a scalar value from a matrix). Many program errors using matrix addition and subtraction fail because of improper dimensioning. MATLAB has tools for checking matrix dimensions, etc. More later.

```
>> c=a + b
ans  c=
        11   33
        44   66
        77   99
```

Note that a(1,1) = 1, b(1,1) = 10, and so c(1,1) = a(1,1)+ b(1,1) = 1+10 =11, etc.

Both matrix and array multiplications are supported in MATLAB. Operations are defaulted to matrix operations unless denoted by a period • as discussed in the following two sections.

1- Matrix Multiplication

Matrix multiplication is indicated with the use of an asterisk *.

```
A =                          B=
     1 2 3                       1 4 7
     4 5 6                       2 5 8
     7 8 0                       3 6 0
```

```
>> C= A * B
```

producing the new matrix C

```
C=
14 32 23
32 77 68
23 68 113
```

Note that C(1,1) = A(1,1)*B(1,1)+ A(1,2)*B(2,1)+ A(1,3)*B(3,1) = 1*1+ 2*2 +3*3 = 1+ 4+ 9 = 14, etc. This means that the order of multiplication is important (i.e., A*B and B*A gives different results).

2-Array Multiplication

Array multiplication is the multiplication of every element in the array by a scalar value and is indicated with a period before the multiplication asterisk as follows:

```
>> D = A.* 2
```

producing the new matrix D

```
D =
2  4  6
8  10  12
14  16  0
```

As with multiplication, both matrix and array division are supported in MATLAB. Operations are defaulted to matrix operations unless denoted by a period as in the operation •/ as discussed in the following two sections. Array division is the division of every element in the array by a scalar value and is indicated with a period before the multiplication asterisk as follows:

```
>> E = A./2
```

divides every element in A by 2

```
E =
0.5000  1.0000  1.5000
2.0000  2.5000  3.0000
```

```
3.5000  4.0000  0
```

Transpose

In general, the transpose of a matrix is a new matrix in which the rows and columns are interchanged. For vectors, transposing a row vector results in a column vector, and transposing a column vector results in a row vector. To convert rows into columns use the transpose symbol ’:

```
a =
    1    3
    4    6
    7    9
>> a’
ans =
      1   4   7
      3   6   9
>> b = [[1 2 3]’ [4 5 6]’]
b =
    1   4
    2   5
    3   6
```

VARIABLES AND ASSIGNMENT STATEMENTS

An assignment statement assigns a number to a variable. To store a value in a MATLAB session, or in a program, a variable is used. The Workspace Window shows variables that have been created. One easy way to create a variable is to use an assignment statement. The format of an assignment statement is

variablename = expression

The variable is always on the left, followed by the = symbol, which is the assignment operator (unlike in mathematics, the single equal sign does not mean equality), followed by an expression. The expression is evaluated and then that value is stored in the variable. For example, this is the way it would appear in the Command Window:

```
>> mynum= 6
mynum =6
>>
```

Here, the user (the person working in MATLAB) typed “mynum = 6” at the prompt, and MATLAB stored the integer 6 in the variable called mynum, and then displayed the result followed by the prompt again. Since the equal sign is the assignment operator, and does not mean equality, the statement should be read as “mynum gets the value of 6” (not “mynum equals 6”).
Note that the variable name must always be on the left, and the expression on the right. An error will occur if these are reversed.

```
>> 6 = mynum
??? 6 = mynum
Error: The expression to the left of the equals sign is not a valid target for an assignment.
>>
```

Putting a semicolon at the end of a statement suppresses the output. For example,

```
>> res = 9 – 2;
>>
```

This would assign the result of the expression on the right side the value 7 to the variable res; it just doesn’t show that result. Instead, another prompt appears immediately. However, at this point in the Workspace Window the variables mynum and res and their values can be seen.

The spaces in a statement or expression do not affect the result, but make it easier to read. The following statement, which has no spaces, would accomplish exactly the same thing as the previous statement:

```
>> res=9-2;
```

MATLAB uses a default variable named ans if an expression is typed at the prompt and it is not assigned to a variable. For example, the result of the expression 6+ 3 is stored in the variable ans.

```
>> 6+ 3
ans =9
```

This default variable is reused any time just an expression is typed at the prompt. A shortcut for retyping commands is to hit the up arrow ", which will go back to the previously typed command(s). For example, if you decided to assign the result of the expression 6+ 3 to the variable “result” instead of using the default ans, you could hit the up arrow and then the left arrow to modify the command rather than retyping the entire statement.

```
>> result= 6+ 3
result =9
```

This is very useful, especially if a long expression is entered with an error, and it is desired to go back to correct it.

To change a variable, another assignment statement can be used, which assigns the value of a different expression to it. Consider, for example, the following sequence of statements:

```
>> mynum = 3
mynum =3
>> mynum =4 +2
mynum=6
>> mynum = mynum +1
mynum =7
```

In the first assignment statement, the value 3 is assigned to the variable mynum. In the next assignment statement, mynum is changed to have the value of the expression 4 +2, or 6. In the third assignment statement, mynum is changed again, to the result of the expression mynum- 1. Since at that time mynum had the value 6, the value of the expression was 6 1, or 7.

At that point, if the expression mynum +3 is entered, the default variable ans is used since the result of this expression is not assigned to a variable. Thus, the value of ans becomes 10 but mynum is unchanged (it is still 7). Note that just typing the name of a variable will display its value.

```
>> mynum +3
ans =10
>> mynum
mynum =7
```

Initializing, incrementing, and decrementing

Frequently, values of variables change. Putting the first or initial value in a variable is called initializing the variable.

Adding to a variable is called incrementing. For example, the statement

mynum = mynum+ 1

increments the variable mynum by 1.

Variable names

Variable names are an example of identifier names. We will see other examples of identifier names, such as file names, in future chapters. The rules for identifier names are:

• The name must begin with a letter of the alphabet. After that, the name can contain letters, digits, and the underscore character (e.g., value_1), but it cannot have a space.

•There is a limit to the length of the name; the built-in function namelengthmax tells what this maximum length is.

• MATLAB is case-sensitive, which means there is a difference between upper- and lowercase letters. So, variables called mynum, MYNUM, and Mynum are all different (although this would be confusing and should not be done).

• Although underscore characters are valid in a name, their use can cause problems with some programs that interact with MATLAB, so some programmers use mixed case instead (e.g., partWeights instead of part_weights)

•There are certain words called reserved words, or key words, that cannot be used as variable names.

• Names of built-in functions can be but should not be used as variable names. Additionally, variable names should always be mnemonic, which means that they should make some sense. For example, if the variable is storing the radius of a circle, a name such as radius would make sense; x probably wouldn't. The Workspace Window shows the variables that have been created in the current Command Window and their values.

The following commands relate to variables:

• who shows variables that have been defined in this Command Window (this just shows the names of the variables)

• whos shows variables that have been defined in this Command Window (this shows more information on the variables, similar to what is in the Workspace Window)

• clear clears out all variables so they no longer exist

• clear variablename clears out a particular variable

• clear variablename1 variablename2 . . . clears out a list of variables (note: separate the names with spaces)

If nothing appears when who or whos is entered, that means there aren't any variables! For example, in the beginning of a MATLAB session, variables could be created and then selectively cleared (remember that the semicolon suppresses output).

```
>> who
>> mynum= 3;
>> mynum +5;
>> who
Your variables are:
ans mynum
>> clear mynum
>> who
Your variables are:
ans
```

Logical Operators

MATLAB has a logical data type, with the possible values 1, representing true, and 0, representing false. Logicals are produced by relational and logical operators/functions and by the functions true and false:

```
>> a = true
a =1
>> b = false
b =0
```

Relational operators allow the comparison of scalars (or matrices, element by element). The result of relational operators is scalars (or matrices of the same size as the arguments) of either 0' or 1's. If the result of comparison is true, the answer is 1; otherwise, it is 0. The following operators are available.

< - less than	**<= - less than or equal**
> - greater than	**>= - greater than or equal**
== - equal	**~= - not equal**

Relations may be connected or quantified by the logical operators

& - and

| - or

~ - not

For numerical operands, the use of these operators is straightforward. For example, 3 < 5 means “3 less than 5,” which is conceptually a true expression. In MATLAB, as in many programming languages, “true” is represented by the logical value 1, and “false” is represented by the logical value 0. So, the expression 3 < 5 actually displays in the Command Window the value 1 (logical) in MATLAB. Displaying the result of expressions like this in the Command Window demonstrates the values of the expressions.

```
>> 3 < 5
ans =1
>> 2 > 9
ans=0
```

The result type is logical. All logical operators operate on logical or Boolean operands. The not operator is a unary operator; the others are binary. The not operator will take a logical expression, which is true or false, and give the opposite value. For example, ~ (3 < 5) is false since (3 < 5) is true. The or operator has two logical expressions as operands. The result is true if either or both of the operands are true, and false only if both operands are false. The and operator also operates on two logical operands. The result of an and expression is true only if both operands are true; it is false if either or both are false. In addition to these logical operators, MATLAB also has a function xor, which is the exclusive-or function. It returns logical true if one (and only one) of the arguments is true.

Given the logical values of true and false in variables x and y, the truth table (see Table 1) shows how the logical operators work for all combinations. Note that the logical operators are commutative.

Table 1 Truth Table for Logical Operators

x	y	~x	x || y	x && y	xor(x,y)
true	true	false	true	true	false
true	false	false	true	false	true
false	false	true	false	false	false

As with the numerical operators, it is important to know the operator precedence rules. Table 2 shows the rules for the operators that have been covered thus far in order of precedence.

Table 2 Operator Precedence Rules

Operators	Precedence
parentheses: ( )	highest
transpose and power ' , ^	
unary: negation (−), not (~)	
multiplication, division *,/,\	
addition, subtraction +, −	
colon operator :	
relational <, <=, >, >=, ==, ~=	
and &&	
or ||	
assignment =	lowest

The following logical operators and functions perform elementwise logical operations on their inputs to produce a like-sized output array. The examples shown in the following table use vector inputs A and B, where

A = [0 1 1 0 1];

B = [1 1 0 0 1];

Operator	Description	Example
&	Returns 1 for every element location that is true (nonzero) in both arrays, and0 for all other elements.	A & B = 01001
|	Returns 1 for every element location that is true (nonzero) in either one or the other, or both arrays, and 0 for all other elements.	A | B = 11101
~	Complements each element of the input array, A.	~A = 10010
xor	Returns 1 for every element location that is true (nonzero) in only one array, and0 for all other elements.	xor(A,B)=10100

For operators and functions that take two array operands, (&, |, and xor), both arrays must have equal dimensions, with each dimension being the same size. The one exception to this is where one operand is a scalar and the other is not. In this case, MATLAB tests the scalar against every element of the other operand. Each of these operators has a representative function that is called whenever that operator is used. These are shown in the table below.

Logical Operation	Equivalent Function
A & B	and(A, B)
A | B	or(A, B)
~A	not(A)

Two other MATLAB functions that operate logically on arrays, but not in an element–wise fashion, are any and all. These functions show whether *any* or *all* elements of a vector, or a vector within a matrix or an array, are nonzero. The examples shown in the following table use array input A, where

A = [0 1 2;0 −3 8;0 5 0];

Function	Description	Example
any(A)	Returns 1 for a vector where *any* element of the vector is true (nonzero), and 0 if no elements are true.	any(A) ans = 0 1 1
all(A)	Returns 1 for a vector where *all* elements of the vector are true (nonzero), and 0 if all elements are not true.	all(A) ans = 0 1 0

The following functions perform bit−wise logical operations on nonnegative integer inputs. Inputs may be scalar or in arrays. If in arrays, these functions produce a like−sized output array.

The examples shown in the following table use scalar inputs A and B, where A = 28; % binary 11100, B=21;% binary 10101

Function	Description	Example
bitand	Returns the bit−wise AND of two nonnegative integer arguments.	bitand(A,B) = 20 (binary 10100)
bitor	Returns the bit−wise OR of two nonnegative integer arguments.	bitor(A,B) = 29 (binary 11101)
bitcmp	Returns the bit−wise complement as an n−bit number, where n is the second input argument to bitcmp.	bitcmp(A,5) = 3 (binary 00011)
bitxor	Returns the bit−wise exclusive OR of two nonnegative integer arguments.	bitxor(A,B) = 9 (binary 01001)

Arrays

MATLAB treats all data as arrays. An array is a `collection of data' (any data - numbers, characters etc.) that is stored in continuous locations in the computer's memory. All variables refer to arrays in the computer's memory. Even scalars are actually treated as 1×1 array.

Arrays are primarily of two types: Vectors (dimension 1) and Matrices (2 or more dimensions). The size of an array is the number of rows and columns in an array. (For higher dimensional arrays it includes the extent of all dimensions).

Example:

a=[1 2;3 4;5 6]	3 ×2 matrix
b =[1 2 3 4]	1 × 4 array, row vector
c=[1;2;3]	3 ×1 array, column vector

Built-in Functions

In MATLAB you will use both built-in functions as well as functions that you create yourself. MATLAB has many built-in functions. These include **sqrt**, **cos**, **sin**, **tan**, **log**, **exp**, and **atan** (for arctan) as well as more specialized mathematical functions.

Basic Matrix Functions

- sum(x)

The sum of the elements of x. For matrices, sum(x) is a row vector with the sum over each column.

```
>> x=[1 2 3;4 5 6];
>> sum(x)
ans = 5 7 9
```

sum (x,dim) sums along the dimension dim.

```
>> sum(x,2)
```

```
ans= 6
      15
```

In order to find the sum of elements that are stored in matrix with *n* dimensions, you must use **sum** command *n* times in cascade form.

```
>>sum(sum(x))
ans = 21
```

- mean(x)

The average of the elements of x. For matrices, mean(x) is a row vector with the average over each column.

```
x=[1 2 3; 4 5 6];
>> mean(x)
ans = 2.5  3.5  4.5
```

mean (x,dim) averages along the dimension dim.

```
>> mean(x,2)
ans =
2
5
>>mean(mean(x))
ans = 3.5000
```

- median(x)

The median value of the elements of x. For matrices, median (x) is a row vector with the median value for each column.

```
x=[4 6 8;10 9 1;8 2 5];
>> median(x)
ans = 8 6 5
```

median(x,dim) takes the median along the dimension dim of x.

```
>> median(x,2)
ans =
6
```

```
9
5
>> median(median(x))
ans =6
```

- max (x)

Find the largest element in a matrix or a vector.

```
>> x=[1 2 3;4 5 6];
>> max (x)
ans = 4 5 6
>> max(max(x))
ans = 6
```

-min (x)

Find the smallest element in a matrix or a vector.

```
>> x=[1 2 3;4 5 6];
>> min (x)
ans = 1 2 3
>> min(min(x))
ans = 1
```

- magic(N)

produce N Magic square.

```
>> magic(3)
ans =
8  1  6
3  5  7
4  9  2
```

- diag(x)

Return the diagonal of matrix x. if x is a vector then this command produce a diagonal matrix with diagonal x.

```
>> x=[1 2 3;4 5 6;7 8 9];
>> diag(x)
ans = 1
         5
        9
>> v=[1 2 3];
>> diag(v)
ans =
1 0 0
0 2 0
0 0 3
```

- prod(x)

Product of the elements of x. For matrices, Prod(x) is a row vector with the product over each column.

```
>> x=[1 2 3;4 5 6];
>> prod(x)
ans = 4 10 18
>> prod(prod(x))
ans = 720
```

-The length and size functions in MATLAB are used to find dimensions of vectors and matrices. The length function returns the number of elements in a vector. The size function returns the number of rows and columns in a vector or matrix. For example, the following vector vec has four elements so its length is 4. It is a row vector, so the size is 1 × 4.

```
>> vec = -2:1
vec = -2  -1  0 1
>> length(vec)
ans = 4
>> size(vec)
ans = 1 4
```

To create the following matrix variable mat, iterators are used on the two rows and then the matrix is transposed so that it has three rows and two columns, or in other words the size is 3 ×2.

```
>> mat = [1:3; 5:7]'
mat =
1 5
2 6
3 7
```

The size function returns the number of rows and then the number of columns, so to capture these values in separate variables we put a vector of two variables on the left of the assignment. The variable r stores the first value returned, which is the number of rows, and c stores the number of columns.

```
>> [r c] = size(mat)
r =3
c =2
```

If called as just an expression, the size function will return both values in a vector:

```
>> size(mat)
ans = 3 2
```

For a matrix, the length function will return either the number of rows or the number of columns, whichever is largest (in this case the number of rows, 3).

```
>> length(mat)
ans = 3
```

-Matrix Sorting

The MATLAB sort function sorts matrix elements along a specified dimension, using sort(A,1) to sort along columns and sort(A,2) to sort along rows. As with many built-in functions, omitting the specified dimension causes the function to operate column-wise, i.e. sort(A) is equivalent to sort(A,1).

```
A = [6 3 2 8; 5 1 3 7; 1, 6, 7, 2;4,5,4,1]
A =
6 3 2 8
5 1 3 7
1 6 7 2
4 5 4 1
sort(A)
ans =
1 1 2 1
4 3 3 2
5 5 4 7
6 6 7 8
sort(A,2)
ans =
2 3 6 8
1 3 5 7
1 2 6 7
1 4 4 5
```

By default, sort sorts in ascending order, but an additional argument can be used to specify descending-order sorting:

```
sort(A,2,'descend')
ans =
8 6 3 2
7 5 3 1
7 6 2 1
5 4 4 1
```

Basic Plotting

Overview

MATLAB has an excellent set of graphic tools. Plotting a given data set or the results of computation is possible with very few commands. You are highly encouraged to plot mathematical functions and results of analysis as often as possible.

1)When looking at plotting we have also seen how to create a row vector with elements from a to b with n regularly spaced elements using the linspace command. To review, ***linspace(a,b)*** creates a row vector of 100 regularly spaced elements between a and b, while ***linspace(a,b,n)*** creates a row vector of n regularly spaced elements between a and b. In both cases MATLAB determines the increment in order to have the correct number of elements.

```
>> linspace(1,5)
```

Create 100 elements begin with 1 and with 5.

```
>> x = linspace(1,5,3)
x =  1 3 5
```

2) MATLAB also allows you to create a row vector of n logarithmically spaced elements by typing ***logspace(a,b,n)***. This creates n regularly spaced elements between 10^a and 10^b. For example:

```
>> logspace(1,2,5)
ans = 10.0000    17.7828   31.6228   56.2341   100.0000
```

Or another example:

```
>> logspace(-1,1,6)
ans = 0.1000  0.2512  0.6310  1.5849  3.9811  10.0000
```

3) ***semilogx(x, y)*** generates a plot with a logarithmic x axis and a rectilinear y axis.

4) ***semilogy(x, y)*** generates a plot with a rectilinear x axis and a logarithmic y axis.

Creating Simple 2D Plots

The basic MATLAB graphing procedure, for example in 2D, is to take a vector of x-coordinates, x = (x1,...., xN), and a vector of y-coordinates, y = (y1,...., yN), locate the points (xi, yi), with i = 1, 2,...., n and then join them by straight lines. You need to prepare x and y in an identical array form; namely, x and y are both row arrays or column arrays of the same length.

The MATLAB command to plot a graph is plot(x,y). The vectors x = (1, 2, 3, 4, 5,6) and y = (3,-1, 2,4,5,1) produce the picture shown in Figure 1.

```
>> x = [1 2 3 4 5 6];
>> y = [3 -1 2 4 5 1];
>> plot(x,y)
```

If we specify two vectors, as mentioned above, plot(x,y) produces a graph of y versus x. For example, to plot the function sin (x) on the interval [0,2 π], we first create a vector of x values ranging from 0 to 2 π, then compute the sine of these values, and finally plot the result:

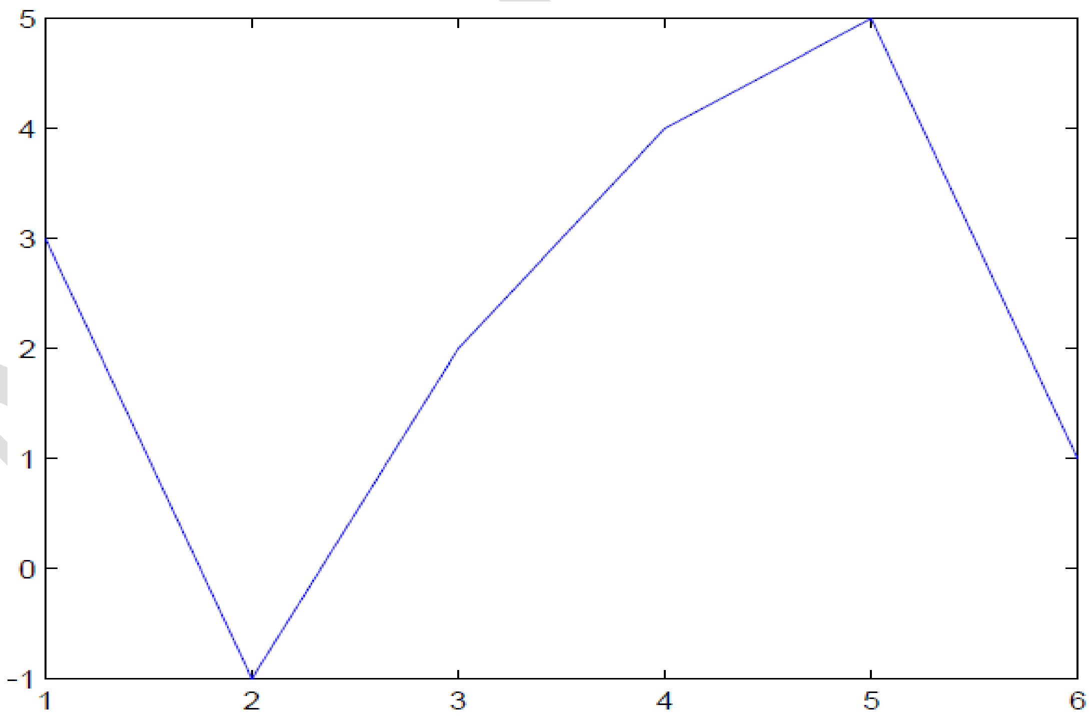


Figure 1: Plot for the vectors x and y

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

• 0:pi/100:2*pi yields a vector that

-starts at 0,

- takes steps (or increments) of π /100,

-stops when 2 π is reached.

• If you omit the increment, MATLAB automatically increments by 1.

Function ***fplot*** gets around our choice of interval used to generate the plot, and instead decides the number of plotting points. Generally, fplot will allow you to generate the most accurate plots possible. The formal call to fplot goes like ***fplot ('function string', [xstart, xend])***. The argument function string tells fplot the function you want to plot while xstart and xend define the range over which to display the plot.

```
>> fplot('exp(-2*t)*sin(t)',[0, 4]);
```

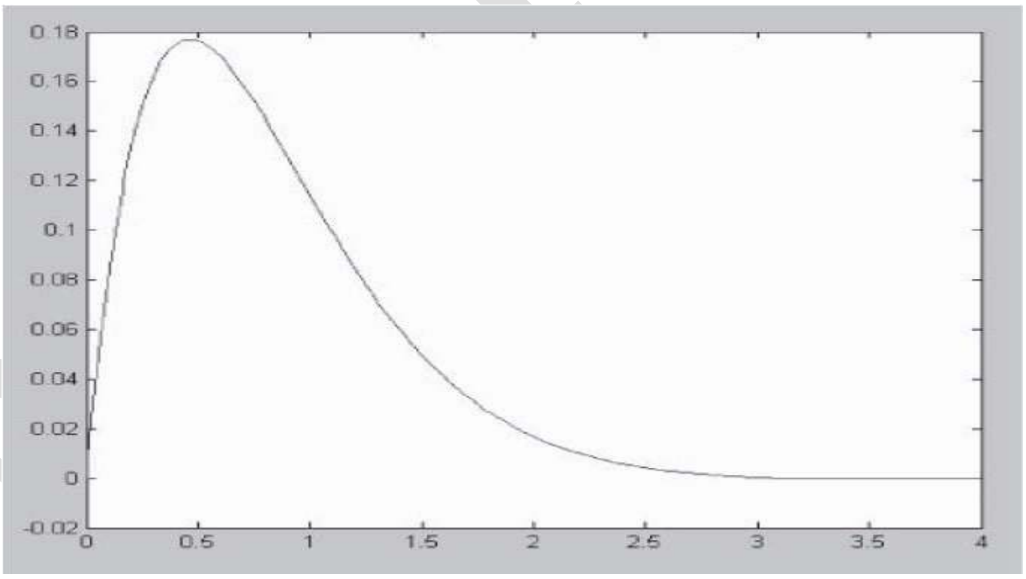


Figure 2: The function f(t)e−2tsint generated using the fplot command.

Adding Titles, Axis Labels

MATLAB enables you to add axis labels and titles. For example, using the graph from the previous example (the function of sin (x)), add an *x*- and *y*-axis labels. Now label the axes and add a title. The character \pi creates the symbol π. An example of 2D plot is shown in Figure 3.

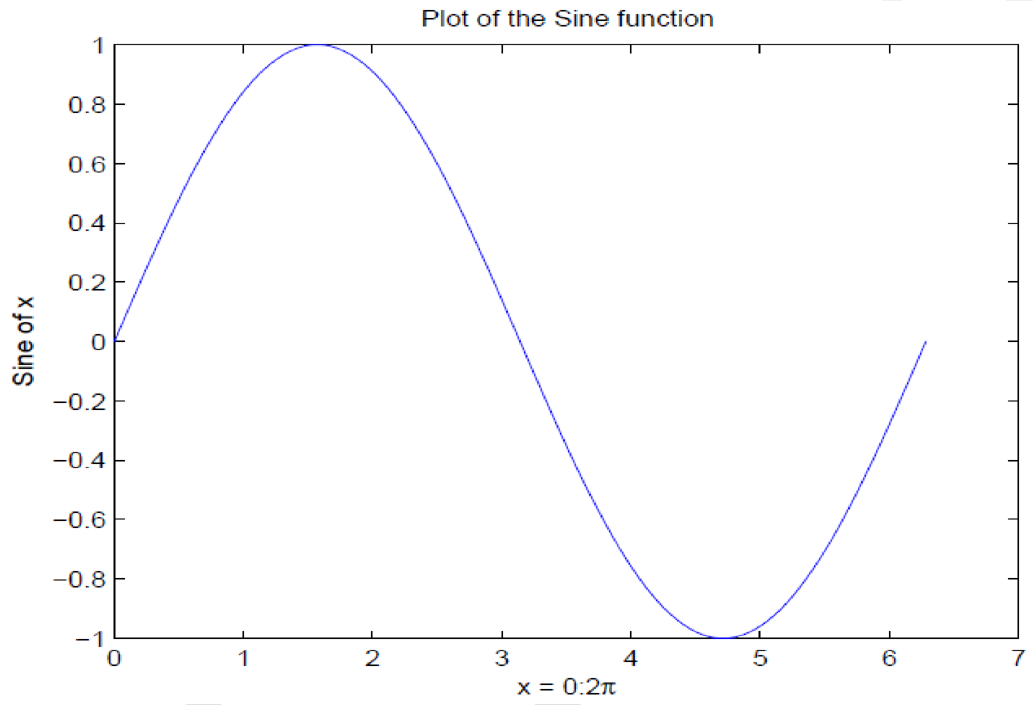


Figure 3: Plot of the Sine function

```
>> xlabel('x = 0:2\pi')
>> ylabel('Sine of x')
>> title('Plot of the Sine function')
```

Specifying Line Styles and Colors

It is possible to specify line styles, colors, and markers (e.g., circles, plus signs, . . .) using the plot command:

plot(x,y,'style_color_marker')

where style_color_marker is a triplet of values from the table below.

SYMBOL	COLOR	SYMBOL	LINE STYLE	SYMBOL	MARKER
k	Black	–	Solid	+	Plus sign
r	Red	– –	Dashed	o	Circle
b	Blue	:	Dotted	*	Asterisk
g	Green	–.	Dash-dot	.	Point
c	Cyan	none	No line	×	Cross
m	Magenta			*s*	Square
y	Yellow			*d*	Diamond

Setting Axis Limits

MATLAB automatically selects appropriate *x*-axis and *y*-axis scaling. Sometimes, it is useful for the user to be able to control the scaling. Control is accomplished with the axis function, the axis command can use to set the plot range.

axis ([xmin xmax ymin ymax])

- The minimum *x* value shown on the *x* -axis
- The maximum *x* value shown on the *x* -axis
- The minimum *y* value shown on the *y* -axis
- The maximum *y* value shown on the *y* -axis

Example:

Plot y = sin(2x + 3) for $0 \le x \le 5$ and $-1 \le y \le 1$.

```
>> x = [0:0.01:5];
>> y = sin(2*x + 3);
>> plot(x,y), axis([0 5 -1 1])
```

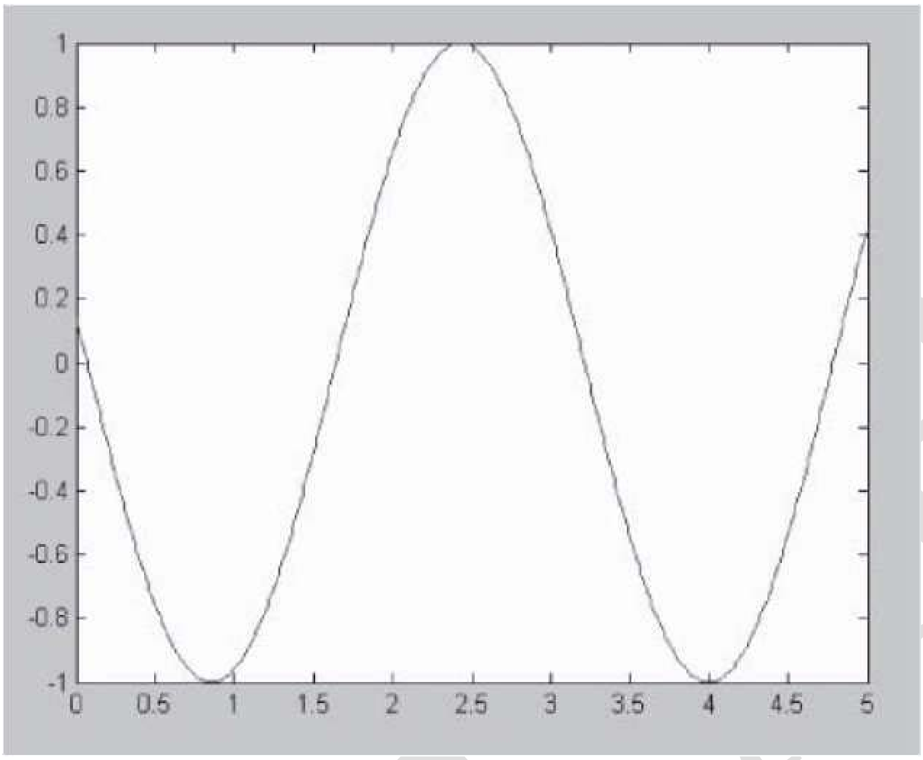


Figure 4: plot of y = sin(2x + 3) for 0 ≤ x ≤ 5

Multiple Data Sets in One Graph

Multiple (x, y) pairs arguments create multiple graphs with a single call to plot.

•***legend('Data1','Data2')*** is used to place a legend and label the data-sets when you have multiple data-sets on one plot. The legend add it to the line used for the plot(x, y) command by add a text string enclosed in single quotes for each curve you want to label.

For example, these statements plot three related functions of x: y1 = 2 cos(x), y2 = cos(x), and y3 = 0.5* cos(x), in the interval 0 ≤ x ≤2 π.

```
>> x = 0:pi/100:2*pi;
>> y1 = 2*cos(x);
>> y2 = cos(x);
>> y3 = 0.5*cos(x);
>> plot(x,y1,'--',x,y2,'-',x,y3,':')
>> xlabel('0 \leq x \leq 2\pi')
>> ylabel('Cosine functions')
>> legend('2*cos(x)','cos(x)','0.5*cos(x)')

>> title('Typical example of multiple plots')
>> axis([0 2*pi -3 3])
```

The result of multiple data sets in one graph plot is shown in Figure 5.

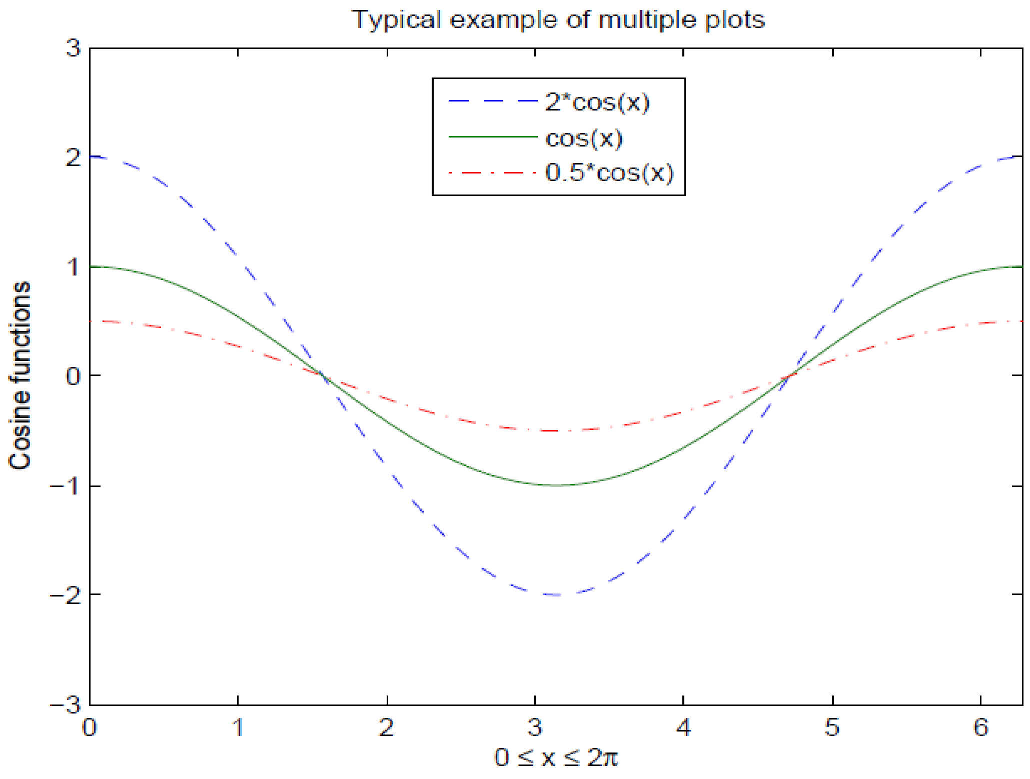


Figure 5: Typical example of multiple plots

MATLAB can add a grid to the plot. This is done by adding the phrase grid on to your plot statement.

Example:

Plot y = tanh(x) over the range $-6 \leq x \leq 6$ with a grid display.

```
>> x = [-6:0.01:6];
>> y = tanh(x);
>>plot(x,y);grid on
```

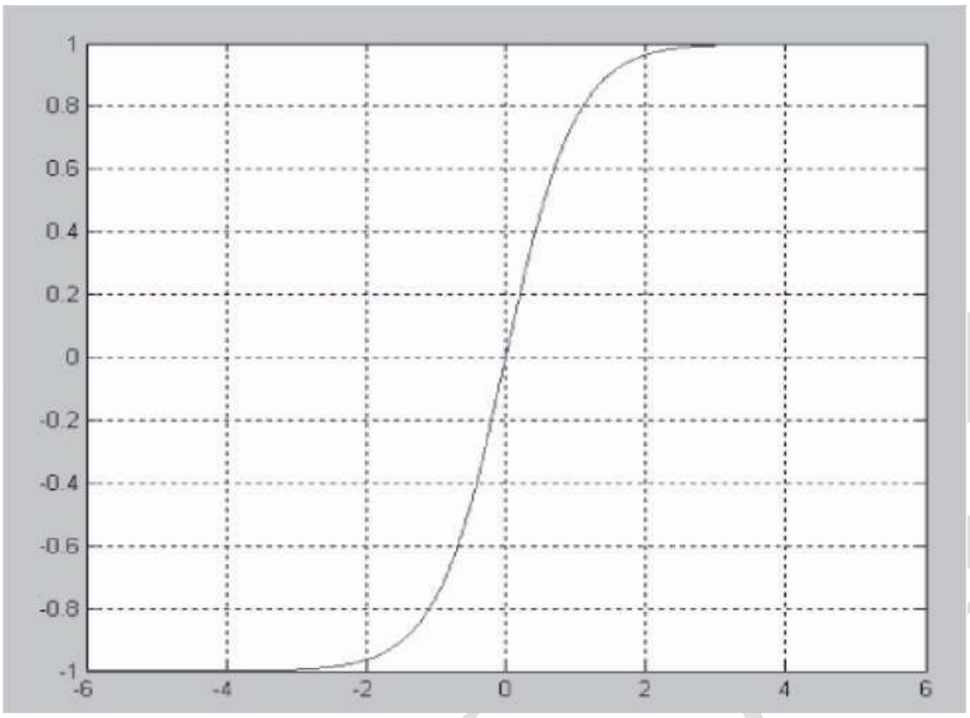


Figure 6: A plot made with the grid on command

Multiple Plots in One Figure

A subplot is one member of an array of plots that appears in the same figure. The subplot command is called using the syntax ***subplot(m, n, p)***. Here ***m*** and ***n*** tell MATLAB to generate a plot array with ***m*** rows and ***n*** columns. Then we use ***p*** to tell MATLAB where to put the particular plot we have generated.

Example:

Using subplot command to plot y $=e^{-1.2x}$ sin(20x) and y = e^{-2x} sin(20x) side by side for $0 \le x \le 5$, $-1 \le y \le 1$.

```
>> x = [0:0.01:5];
>> y = exp(-1.2*x).*sin(20*x);
>> subplot(1,2,1);
>> plot(x,y);xlabel('x');ylabel('exp(-1.2x)*sin(20x)');axis([0 5 -1 1])
```

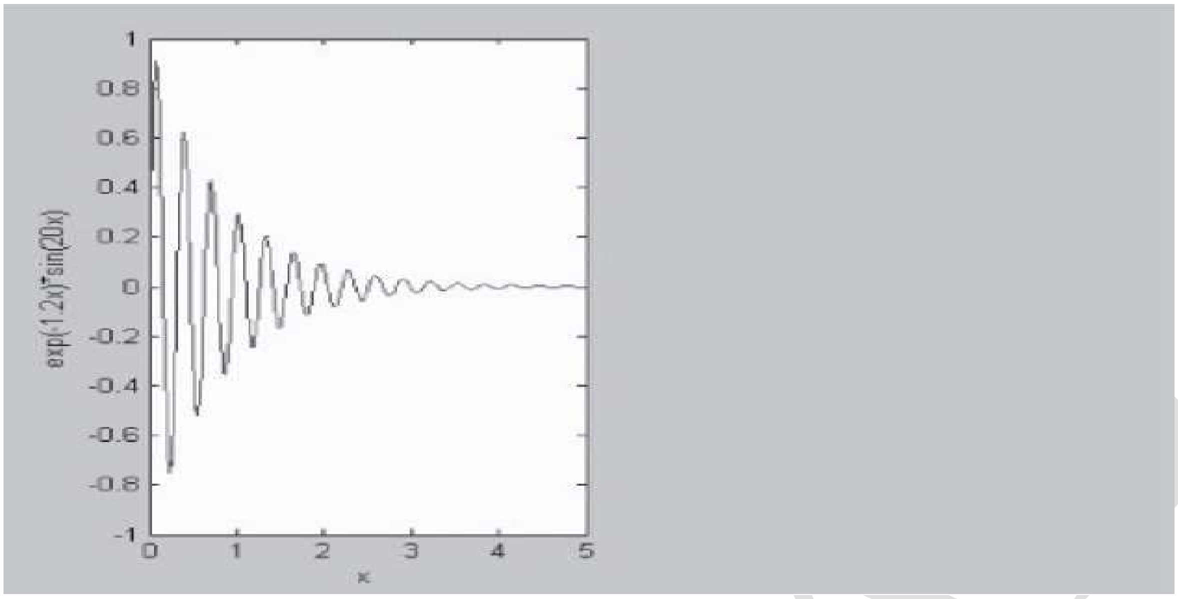


Figure 7: A glance at the MATLAB output after our first calls to subplot and plot

```
>> y = exp(-2*x).*sin(20*x);
>> subplot(1,2,2);
>> plot(x,y);xlabel('x');ylabel('exp(-2x)*sin(20x)');axis([0 5 -1 1])
```

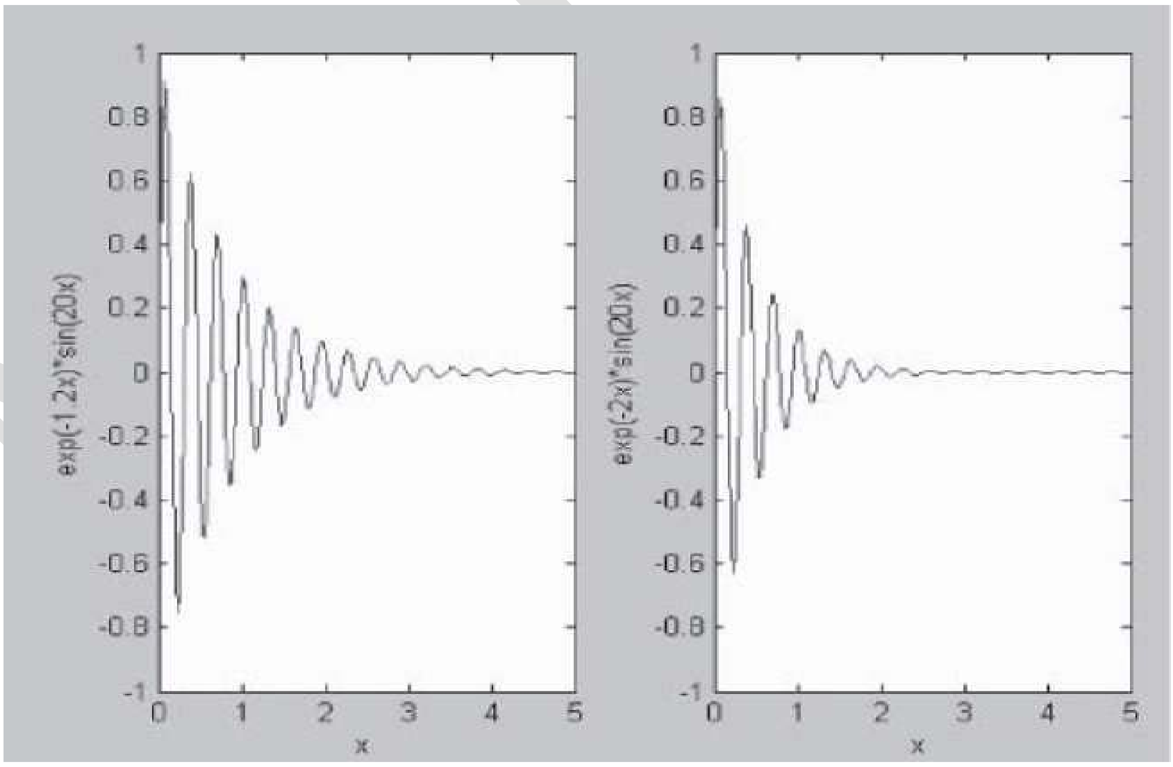


Figure 8: Two side-by-side plots generated by MATLAB

Adding Plots to an Existing Graph

You can add plots to an existing graph using the hold command. When you set ***hold on***, MATLAB does not remove the existing graph; it adds the new data to the current graph, rescaling if the new data falls outside the range of the previous axis limits. For example: Plot cos(x) and sin(x) for 100 linearly spaced points from 0 to 2π.

```
>> x = linspace(0,2*pi);
>> plot(x,cos(x))
```

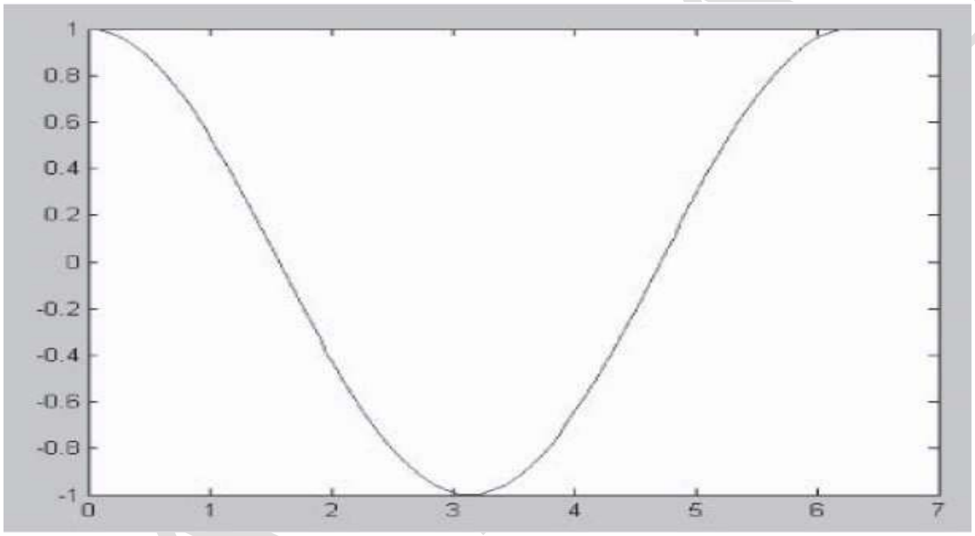


Figure 9: A plot of cos(x) generated using the linspace command

```
>> plot(x,sin(x))
```

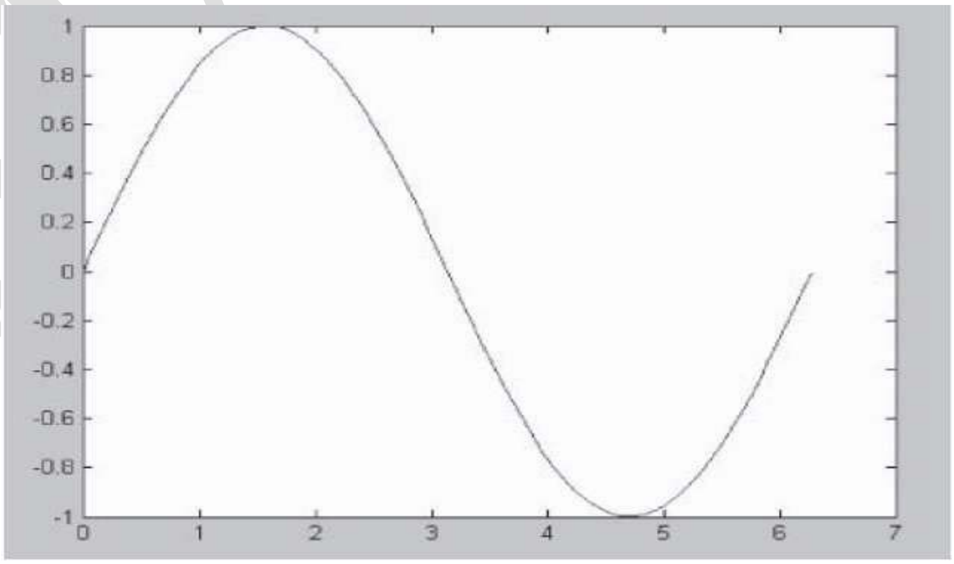


Figure 10: We overwrite the previous plot by typing plot[x, sin(x)]

Plot cos(x) and want to overlay sin(x) on the same graphic for 100 linearly spaced points from 0 to 2π.

```
>> x = linspace(0,2*pi);
>> plot(x,cos(x));axis([0 2*pi -1 1])
>> hold on
>> plot(x, sin(x)); axis ([0 2*pi -1 1])
```

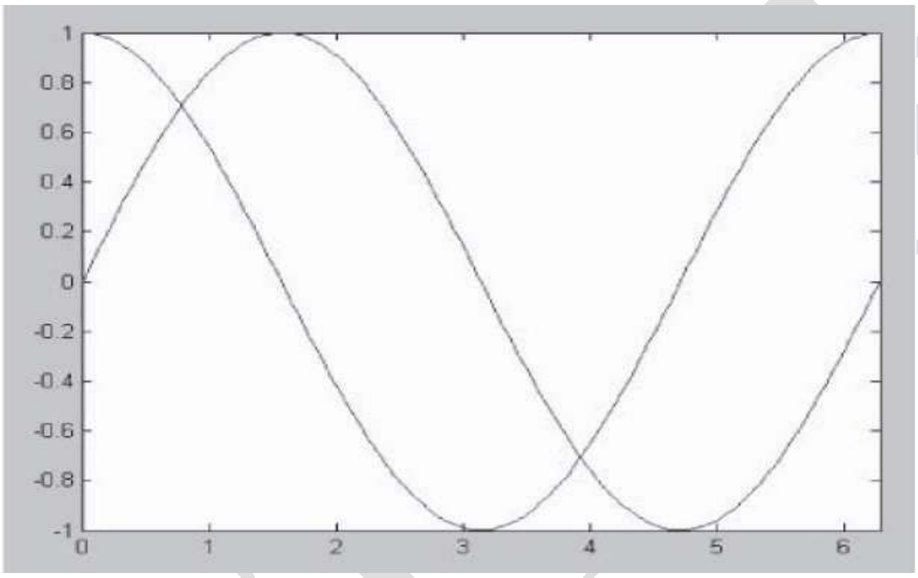


Figure 11: Both curves added on the same graph

M-FILES

The most common way to operate MATLAB is by entering commands one at a time in the command window. M-files provide an alternative way of performing operations that greatly expand MATLAB's problem-solving capabilities. An M-file consists of a series of statements that can be run all at once. Note that the nomenclature "M-file" comes from the fact that such files are stored with a .m extension. M-files come in two flavors: script files and function files.

Script Files

A script file is merely a series of MATLAB commands that are saved on a file. They are useful for retaining a series of commands that you want to execute on more than one occasion. The script can be executed by typing the file name in the command window or by invoking the menu selections in the edit window: Debug, Run.

EXAMPLE: Go to your file edit window to create the following file that you name myfile.m.

```
clear
x1=1;
y1=0.5;
x2=2;
y2=1.5;
x3=3;
y3=2;
plot(x1,y1,'o',x2,y2,'+',x3,y3,'*')
axis([0 4 0 4])
xlabel('xaxis')
ylabel('yaxis')
title('3points in a plane')
```

After creating and saving myfile.m, go to the MATLAB command window and enter myfile. MATLAB will execute the instructions in the order of the statements stored in your myfile.m file.

Function Files

Function files are M-files that start with the word function. In contrast to script files, they can accept input arguments and return outputs. The syntax for the function file can be represented generally as

```
function outvar = funcname (arglist)
% helpcomments
statements
outvar = value;
```

where outvar = the name of the output variable, funcname = the function's name, arglist = the function's argument list (i.e., comma-delimited values that are passed into the function), helpcomments = text that provides the user with information regarding the function (these can be invoked by typing Help funcname in the command window), and statements = MATLAB statements that compute the value that is assigned to outvar. The M-file should be saved as funcname.m. The function can then be run by typing funcname in the command window.

Example: Write a function file to solve the equivalent resistance of series connected resistors, R1, R2, R3, …, Rn.

```
function req = equiv_sr(r)
% equiv_sr is a function program for obtaining the equivalent resistance of series connected resistors
% usage: req = equiv_sr(r)
% r is an input vector of length n
% req is an output, the equivalent resistance (scalar)
```

```
n = length(r); % number of resistors
req = sum (r); % sum up all resistors
```

Suppose we want to find the equivalent resistance of the series connected resistors 10, 20, 15, 16 and 5
ohms. The following statements can be typed in the MATLAB command window to reference the function equiv_sr

```
>> a = [10 20 15 16 5];
>> Rseries = equiv_sr(a)
Rseries =
 66
```

Input statement

the input statement reads in values that have been entered by the user, or the person who is running the script. To let the user know what he or she is supposed to enter, the script must first prompt the user for the specified values.
The simplest input function in MATLAB is called input. The input function is used in an assignment statement. To call it, a string is passed that is the prompt that will appear on the screen, and whatever the user types will be stored in the variable named on the left of the assignment statement. For ease of reading the prompt, it is useful to put a colon and then a space after the prompt. For example,

```
>> rad =input('Enter the radius: ')
Enter the radius: 5
rad =
5
```

The input function can also return user input as a string. To do this, an 's' is appended to the function’s argument list. For example,

```
>> letter = input('Enter a char: ','s')
Enter a char: g
letter =
g
```

Output statements

Output statements display strings and/or the results of expressions. The simplest output function in MATLAB is disp, which is used to display the result of an expression or a string without assigning any value to the default variable ans. However, disp does not allow formatting. Its syntax is:

disp(value)

where value = the value you would like to display. For example,

```
>> disp('Hello')
Hello
>> disp(4^3)
64
```

CONDITIONAL STATEMENTS

A conditional statement is a command that allows MATLAB to make a decision of whether to execute a group of commands that follow the conditional statement, or to skip these commands. In a conditional statement a conditional expression is stated. If the expression is true, a group of commands that follow the statement are executed. If the expression is false, the computer skips the group.

THE IF STATEMENT

The if statement chooses whether another statement, or group of statements, is executed or not. The general form of the if statement follows:

```
if   condition
     action
end
```

A condition is a relational expression that is conceptually, or logically, true or false. The action is a statement, or a group of statements, that will be executed if the condition is true. When the if statement is executed, first the condition is evaluated. If the value of the condition is true, the action will be executed; if not, the action will not be executed. The action can be any number of statements until the reserved word end; the action is naturally bracketed by the reserved words if and end. (Note: This is different from the end that is used as an index into a vector or matrix.) The action is usually indented to make it easier to see.

For example, the following if statement checks to see whether the value of a variable is negative. If it is, the value is changed to a positive number by using the absolute value function; otherwise, nothing is changed.

```
if  num < 0
 num = abs(num)
end
```

If statements can be entered in the Command Window, although they generally make more sense in scripts or functions. In the Command Window, the if line would be entered, followed by the Enter key, the action, the Enter key, and finally end and Enter. The results immediately follow. For example, the preceding if statement is shown twice here.

```
>> num = -4;
>> if num < 0
num= abs(num)
end
num =
4
>> num= 5;
>> if num < 0
num =abs(num)
end
```

THE IF-ELSE STATEMENT

The if statement chooses whether or not an action is executed. Choosing between two actions, or choosing from several actions, is accomplished using if-else, nested if, and switch statements. The if-else statement is used to choose between two statements, or sets of statements. The general form is:

```
if   condition
   action1
else
  action2
end
```

First, the condition is evaluated. If it is true, then the set of statements designated as “action1” is executed, and that is the end of the if-else statement. If instead the condition is false, the second set of statements designated as “action2” is executed, and that is the end of the if-else statement. The first set of statements (“action1”) is called the action of the if clause; it iswhat will be executed if the expression is true. The second set of statements (“action2”) is called the action of the else clause; it is what will be executed if the expression is false.One of these actions, and only one, will be executed—which one depends on the value of the condition. For example, to determine and print whether or not a random number in the range from 0 to 1 is less than 0.5, an if-else statement could be used:

```
if rand < 0.5
disp('It was less than .5!')
else
disp('It was not less than .5!')
end
```

The elseif clause

In most programming languages, choosing from multiple options means using nested if-else statements. However, MATLAB has another method of accomplishing this using the elseif clause. To choose from among more than two actions, the elseif clause is used. For example, if there are n choices (where n > 3 in this example), the following general form would be used:

```
if condition1
action1
elseif condition2
action2
elseif condition3
action3
% etc: there can be many of these
else
actionn % the nth action
end
```

The actions of the if, elseif, and else clauses are naturally bracketed by the reserved words if, elseif, else, and end. For example, the previous example could be written using the elseif clause rather than nesting if-else statements:

Example: Write a script file to prompt the user to enter an integer, and then display whether the integer is zero, positive or negative.(named file “testNumber”)

Sol:

```
n=input('Enter an integer : ');
if n>0
disp('The number is Positive')
elseif n<0
disp('The number is Negative')
else
disp('The number is Zero')
end
```

```
>>testNumber
Enter an integer : 9
The number is Positive
>>testNumber
Enter an integer : 0
The number is Zero
```

Example 3: Create a MATLAB program that take three numbers as argument and display the maximum of the numbers.

SOLUTION:

```
a=input('a=')
b=input('b=')
c=input('c=')
if (a>b) & (a>c)
```

```
disp('max=a')
elseif (b>c)
disp('max=b')
else
disp('max=c')
end
```

Example 4:write a program to find the Y value when:

$$Y = \begin{cases} \frac{2}{x^3} & -10 \le x < 0 \\ 2 & x = 0 \\ \sqrt{x^2+4} & 0 < x \le 7 \end{cases}$$

SOLUTION:

```
x=input('Enter the x value : ');
if x<-10 | x>7
disp('Undefined the Y value')
elseif x>=-10 & x<0
Y=2/x^3;
elseif x>0 & x<=7
Y=sqrt(x^2+4)
else
y=2
end
```

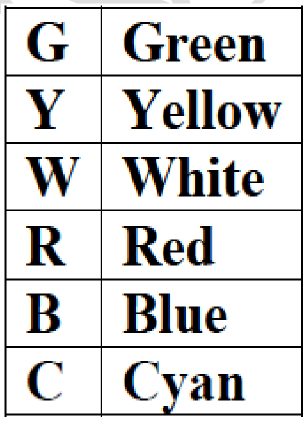
THE SWITCH STATEMENT

A switch statement can often be used in place of a nested if-else or an if statement with many elseif clauses. Switch statements are used when an expression is tested to see whether it is equal to one of several possible values. The general form of the switch statement is:

```
switch  switch_expression
case  caseexp1
action1
case  caseexp2
action2
case  caseexp3
action3
% etc: there can be many of these
otherwise
actionn
end
```

The switch statement starts with the reserved word switch, and ends with the reserved word end. The switch_expression is compared, in sequence, to the case expressions (caseexp1, caseexp2, etc.). If the value of the switch_expression matches caseexp1, for example, then action1 is executed and the switch statement ends. If the value matches caseexp3, then action3 is executed, and in general if the value matches caseexpi where i can be any integer from 1 to n, then the actioni is executed. If the value of the switch_expression does not match any of the case expressions, the action after the word otherwise is executed (the nth action, actionn).

Example 1: Write a program to select a color by entering the 1st letter of its name: And handle the invalid letter (named with "colortest")



G	Green
Y	Yellow
W	White
R	Red
B	Blue
C	Cyan

Sol:

```
color=input('Enter color letter : ','s');
switch color
case {'G','g'}
disp('The color is Green');
case {'Y','y'}
disp('The color is Yellow');
case {'R','r'}
disp('The color is Red');
case {'W','w'}
disp('The color is White');
case {'B','b'}
disp('The color is Blue');
case {'C','c'}
disp('The color is Cyan');
otherwise
disp('Undefined Color');
end
>> colortest
Enter color letter : y
The color is Yellow
>> colortest
Enter color letter : b
The color is Blue
>> colortest
Enter color letter : R
The color is Red
```

Example 2: Write a program to display the name of day by giving the day number as the following:

1	Saturday
2	Sunday
3	Monday
4	Tuesday
5	Wednesday
6	Thursday
7	Friday

Sol:

```
Nday=input('Enter The Day Number : ');
switch Nday
case 1
disp('The day is SATURDAY');
case 2
disp('The day is SUNDAY');
case 3
disp('The day is MONDAY');
case 4
disp('The day is TUESDAY');
case 5
disp('The day is WEDNESDAY');
case 6
disp('The day is THURSDAY');
case 7
disp('The day is FRIDAY');
otherwise
disp('Invalid');
end
```

>> **dayname**

```
Enter The Day Number : 3
The day is MONDAY
>> dayname
Enter The Day Number : 7
The day is FRIDAY
>> dayname
Enter The Day Number : 10
Invalid
```