University of Baghdad
College of Science
Department of Mathematics

First Class
Second Course

Introduction to Matlab

Matlab Basics

1

Some basics of using Matlab

To begin, you can use Matlab for simple arithmetic problems. Symbols like + (plus), - (minus), * (multiply), and / (divide) all work as you would expect. In addition, ^ is used for exponentiation. For example, if you type

```
>> 75 - 32 * 2 + 4 / 2
```

in the Command Window (note that the >> is added by Matlab) and press <Enter>, Matlab calculates the value of the expression and responds with

```
ans =
      13
```

The order of precedence

In general, the order of precedence is:

1. parentheses
2. exponentiation
3. multiplication, division
4. addition, subtraction

```
>> 10/20/10
ans =
      0.05
```

and

```
>> 10/(20/10)
```

```
ans =
     5
```

If you are unsure, always add parentheses. That also makes many expressions easier to read. Writing

```
>> 75 - (32 * 2) + (4 / 2)
ans =
     13
```

does not change the outcome, but you could argue that it is somewhat easier to read. As expressions become more complex, the difference becomes more pronounced.

Some algebraic functions, special characters, and tips

In Matlab, a few characters have special meaning and are used frequently.

, Comma

This can be used for separating commands that are written on the same line. You can,

```
>> sqrt(4),sqrt(9)
ans=
     2
     3
```

and

; Semicolon

This suppresses output when commands are executed.

```
>> 5+9
ans=
     14
```

While

```
>> 5+9;
```

nothing is displayed in the Command Window. This is useful when you run programs and do not want intermediary calculations to be displayed.

Also we have

% Comments

Everything after the %-character until the end of the line is ignored.

Basic algebraic functions include:

sqrt(9)	Square root of 9.
3^4	Exponentiation 3^4; 3 to the power of 4 (i.e., 3*3*3*3).
exp(4)	e^4; exponentiation with base e (≈ 2.7183).
log(5)	The natural logarithm of 5 (note: with base e, not with base 10).
log10(5)	The logarithm of 5, using base 10.
abs(-3)	The absolute value of -3.
round(1.3)	Round to the nearest integer.
floor(1.3)	Round to the nearest smaller integer (i.e., towards minus infinity).
ceil(1.3)	Round to the nearest larger integer (i.e., towards plus infinity).
fix(1.3)	Round to the nearest integer towards zero (i.e., downwards for positive numbers and upwards for negative).
sign(1)	The sign of 1: -1 if < 0; 1 if > 0; and 0 if = 0.

You can nest functions, for example

```
>> round(abs(-4)^3)
ans =
     64
```

In addition, you will probably want to use the following:

clc	Clears the Command Window. Only the window is cleared, no

	variables or anything else is changed.
clear	Deletes variables. clear alone deletes all variables, clear and one or several variable names, for example clear abc, deletes only the named variable(s).
↑	Pressing this arrow-key brings back the previous command lines one-by-one. If you type one or several characters first and then press the arrow key, Matlab skips to those lines that begin with the same characters.

Variables

We can define variables as follows:

```
>> abc = 10/20/10
abc =
    0.05
```

the equality sign, in this context, tells Matlab to assign a value to a variable, it is called the assignment operator.

When a variable has been defined, you can ask for its value by entering the name:

```
>> abc
abc =
    0.05
```

You can also reassign variables using other variables or even using the variable itself.

```
>> abc = abc*2
abc =
    0.1
```

Variable names must begin with a letter and can be at most 63 characters long.

Note that Matlab is case sensitive, so c and C are two different variables, as are interestRate and interestrate.

Predefined variables

A few variables are predefined in Matlab.

ans	"answer" A variable that holds the (latest) value that has been calculated with no specified variable name.
pi	The constant $\pi = 3.141592653589793\ldots$
inf	Infinity Rather than infinity this means "overflow". This can be the output of a calculation. For example, 1/0 gives the result inf. Note that you can also use it in calculations. For example, 3*inf is inf.
NaN	Not-a-number This is used when it is not possible to record a valid numerical value, for example when you have missing observations in a data set. It can also be used in calculations or be the outcome of a calculation. NaN*2 and 0/0 both resolve to NaN.

Different types of variables

Variables can be of different types. We need to distinguish between numerical, character, and logical variables.

1 Numerical and character variables

It is important to note that there are different types of variables.

Example of numerical variable:

```
>> abc = 10/20/10
abc =
    0.05
```

To tell Matlab that you want it to interpret something as a string of characters, you enclose it within single quotes (').

Example of character variable (string variable):

```
>> 'def'
ans =
def
```

Consequently, it is very different to enter 123 and to enter '123'. If we enter both, separating them with a comma, we get

```
>> 123, '123'
ans =
       123
ans =
123
```

The two answers look deceivingly similar. The main difference is that the first 123 has a few white spaces in front of it, so that it is located a bit off from the left side. The result is, naturally, the number 123.

In the second case, '123', is interpreted as a string of characters. The fact that a string of 1, 2 and 3 can be interpreted as a number is ignored. There are convenient ways to translate numerical variables to character strings and vice versa.

str2num	Converts a string of characters to a number, given that Matlab can interpret the string as a number. For example, str2num('542') is 542.
num2str	Converts a number to a string of characters. For example, num2str(542) is the string '542'.

Strings have several different functions in Matlab. They are frequently used to display messages and in addresses to files on the computer, for instance when importing data. Some functions also take string input arguments. You can also enter formatting commands for plots as strings.

2 Logical variables

Logical variables are a type of variable that can only take the values true (1) and false (0).

```
>> abc = 7;
>> abc > 5
ans =
     1
```

The answer is 1.The variable ans can still be used in algebraic expressions. Then it is treated as an ordinary zero or one, depending on its value. If we issue

```
>> ans*1
ans =
     1
```

Matrices, Vectors and Scalars

2

Creating matrices

The most straightforward way to create a matrix in Matlab is to enter it element-by-element.

```
>> testMatrix = [1 32 7 8 ; 2 4 1 9 ; 3 19 3 9]
testMatrix =
     1    32     7     8
     2     4     1     9
     3    19     3     9
```

Commands for creating matrices

[... ; ...]	Typing all values manually within square brackets and indicating a new row with a semicolon. For example, [1 2 3 ; 4 5 6].
1:3:20	The colon operator This can only be used for creating a matrix with one row; a row vector. The first element will be equal to the first number (1 in the example). Subsequent elements will increase by the number in the middle (the increment; 3 in the example) and the last element will be no higher than 20. The resulting matrix in the example is [1 4 7 10 13 16 19]. 20 is not included as the next element would have been 19+3=22, which is greater than 20.

	If the middle number is omitted, Matlab uses an increment of 1. Consequently, 4:8 is the matrix [4 5 6 7 8].
linspace(2,9,5)	Linear spacing This creates a matrix with only one row; a row vector. The first element is 2, the last element is 9, and there are 5 linearly spaced elements (i.e., the distance between each is the same). In the example, the resulting matrix is [2 3.75 5.5 7.25 9]. Note that the difference between each consecutive number is 1.75 and that there are 5 elements.
zeros(2,3)	Creates a 2x3 matrix of zeros.
ones(2,3)	Creates a 2x3 matrix of ones.
eye(4)	Creates a 4x4 square matrix with a diagonal of ones and all other elements equal to zero.
rand(5,4)	Creates a 5x4 matrix of random numbers between 0 and 1 (drawn from a uniform distribution).
randn(5,4)	Creates a 5x4 matrix of random numbers between minus infinity and plus infinity (drawn from a standard normal distribution, meaning the numbers will usually be between -3 and 3).
repmat(A,2,3)	Repeats the matrix A twice vertically and three times horizontally.
[A B], [A;B]	Concatenation Assuming that A and B are already defined matrices; this produces a matrix with elements equal to those of A and B. In the first case, the matrices are lined up beside each other, and in the second, they are stacked on top of each other. Note that, A and B must have the same number of rows in the first case, and the same number of columns in the second.
diag(X)	Diagonal

	This command has different meanings depending on the input matrix, X. If X is a matrix and not a vector (i.e., if it has at least two rows and two columns), the command creates a row vector containing the values of the diagonal of X, beginning with the top-left element. if X is a vector, the command creates a square matrix with the values of X on the diagonal and zeros elsewhere.

To see an example of some of these commands, consider the following. (Note that the semicolons suppress all output except the last one.)

```
>> A = 2:2:6; B = linspace(10,16,3); C = zeros(1,3);
>> D = [A ; B ; C]; E = [D eye(3)]
E =
     2     4     6     1     0     0
    10    13    16     0     1     0
     0     0     0     0     0     1
```

Addressing parts of matrices

Sometimes we want to pick out, or change, a subset of the elements of a matrix. There are three different ways of addressing a subset.

```
>> testMatrix(3,3)
ans =
     3
```

You can also pick out a larger subset of a matrix.

```
>> testMatrix([1 2 3],[1 4])
ans =
     1     8
     2     9
     3     9
```

If you want to pick out all the elements in one dimension (i.e., all rows or all columns), there is an even simpler way to do that. Note that, in

the example we pick out all rows from testMatrix. In such cases, you may enter a colon, without any start or end indicators. Both of these two methods produce the same result as the previous example.

```
>> testMatrix(1:3,[1 4]), testMatrix(:,[1 4])
ans =
     1     8
     2     9
     3     9
ans =
     1     8
     2     9
     3     9
```

Oftentimes, we want to pick out all elements from some starting row, or column, until the last one.

```
>> testMatrix(1:end-1,[1 4])
ans =
     1     8
     2     9
```

Addressing using a single index

Row or column vectors can be addressed with just one index that indicates where the subset is located. For example, if **testVector** is the row vector [1 3 5 7 9 11], then the first four elements canbe picked out as

```
>> testVector(1:4)
ans =
     1     3     5     7
```

However, matrices can also be addressed using a single index. In that case, the elements are numbered columnwise, starting with the upper-left element. The first seven elements of testMatrix are

```
>> testMatrix(1:7)
ans =
```

```
     1     2     3    32    4    19    7
```

Similarly to addressing using row and column numbers, you can pick out all elements using the colon operator alone.

```
>> testMatrix(:)
ans =
     1
     2
     3
    32
     4
    19
     7
     1
     3
     8
     9
     9
```

In this case, the result is a column vector.

Addressing using conditional statements

Matrix subsets can also be picked out based on criteria regarding the elements themselves.

```
>> index_gt5 = find(testMatrix > 5)
index_gt5 =
     4
     6
     7
    10
    11
    12
```

index_gt5, is a list of indices of which elements in testMatrix that are greater than 5. Note that the numbers in this list correspond to addressing the matrix in the single index style (i.e., numbering them columnwise). We can then use the index to pick out the elements in testMatrix that are greater than 5.

```
>> testMatrix(index_gt5)
ans =
     32
     19
     7
     8
     9
     9
```

You can do the same thing nesting the commands as:

```
>> testMatrix(find(testMatrix > 5))
```

There is also an alternative strategy for conditional addressing, using logical variables. If we issue the command

```
>> index_gt5_logic = testMatrix > 5
index_gt5_logic =
     0     1     1     1
     0     0     0     1
     0     1     0     1
```

This type of index can also be used to pick out the values in testMatrix that are greater than 5.

```
>> testMatrix(index_gt5_logic)
ans =
     32
     19
     7
     8
```

```
     9
     9
```

Alternatively, you can nest the commands as:

```
>> testMatrix(testMatrix > 5)
```

which is somewhat shorter than the example using find().

Changing parts of a matrix

The way to do that is to address the subset that you want to change, followed by the assignment operator (=) and, lastly, what you want to change the subset to.

```
>> testMatrix(2:3,4) = [7 ; 6]
testMatrix =
     1    32     7     8
     2     4     1     7
     3    19     3     6
```

Note also that you can do the same thing using the single index addressing method:

```
>> testMatrix(11:12) = [7 ; 6]
testMatrix =
     1    32     7     8
     2     4     1     7
     3    19     3     6
```

You can also use conditional addressing to change data. Suppose you are only interested in observations that are less than 10. Then you can set the rest to NaN (not-a-number) by issuing

```
>> testMatrix(testMatrix >= 10) = NaN
testMatrix =
     1   NaN     7     8
     2     4     1     7
```

```
     3     NaN     3     6
```

Conditional addressing is also useful when we want to select a certain period from the data.

```
>> testMatrix(testMatrix(:,1) >= 2,:)
ans =
     2     4     1     7
     3     NaN 3     6
```

Reducing and increasing the size of a matrix

Sometimes you will want to delete parts of a matrix (i.e., reduce its size). Since matrices cannot have empty entries, you can only delete full rows or full columns. For example, to delete column two of the test matrix, we enter

```
>> testMatrix(:,2) = [ ]
testMatrix =
     1     7     8
     2     1     7
     3     3     6
```

Increasing the size of a matrix is done automatically if you add one or several elements outside the existing rows or columns.

```
>> testMatrix(4,1) = 4
testMatrix =
     1     7     8
     2     1     7
     3     3     6
     4     0     0
```

Some special commands for handling matrices

To handle matrices, it is useful to know the following commands.

find(X>3)	Creates a vector of numbers that indicate which elements in X that are greater than 3. (Note: uses single index addressing.) [I,J] = find(X>3), where you specify that you want two outputs, creates two vectors. One vector, I, with row numbers and another one, J, with column numbers. This is, consequently, an example of a function that works differently depending on which output you ask for.
size(X)	Creates a 1x2 vector where the first element is the number of rows in X and the second element is the number of columns.
transpose(X)	This flips the columns and rows of the matrix X. Note: instead of the command transpose(X), you can write a single quote after the matrix that is to be transposed: X'.
sort(X)	X sorted from smallest to largest (columnwise). Each column is sorted separately.
sortrows(X,3)	X sorted as a group from smallest to largest values in the third column. The third column is sorted and observations in the other columns stick to the corresponding values of that column. Entering a negative sorting column number produces a sorting from largest to smallest instead.

Note that, character strings are also matrices; character matrices. Therefore, many matrix commands, including all the ones above, operate on strings as well. Try, for instance, transpose(sort('Matlab')).

Suppose now that we want to sort the data in testMatrix with respect to how much it has rained in the first location (i.e., the second column). We then enter

```
>> sortrows(testMatrix,2)
```

```
ans =
     4     0     0
     2     1     7
     3     3     6
     1     7     8
```

Note that column 2 is now in ascending order. The values in columns 1 and 3 still correspond to the same values in column 2 as they did before, though. Suppose, instead, we enter

```
>> sort(testMatrix)
ans =
     1     0     0
     2     1     6
     3     3     7
     4     7     8
```

Then all columns are sorted independently of each other, and the relations to the observations in the other columns are lost.

```
>> testMatrix(4,1) = [ ]
testMatrix =
     1     7     8
     2     1     7
     3     3     6
```

Mathematical Operations with Matrices

3

Functions that operate element-by-element

To multiply or divide matrices element-by-element using the ordinary operators * and /, you have touse so called dot-notation. This means that you put a dot in front of the operators. For example

```
>> X = [2 4; 3 4]; Y = [2 2; 3 1]; X.*Y, X./Y
ans =
     4     8
     9     4
ans =
     1     2
     1     4
```

Each element in the first answer is the product of the corresponding elements in X and Y, and each element in the second answer is the corresponding element in X divided by its counterpart in Y. Note that, the matrices must have exactly the same dimensions. The exception is if either X or Y is a scalar. Other mathematical functions we have seen that use parentheses, such as the square root, also operate element-by-element.

```
>> sqrt(X)
ans =
    1.4142    2
    1.7321    2
```

An example comparing two full matrices instead is

```
>> X > Y
ans =
     0     1
     0     1
```

Elementary mathematical functions that operate columnwise

Matrices often contain data observations and it is practical to calculate ordinary summary statistics directly on them.

min(X)	The minimum value of X (columnwise). [**val**,**row**]=**min**(X) produces two vectors. val contains the maximum values for each column and row contains the row numbers where each corresponding maximum is located.
max(X)	The maximum value of X (columnwise). [**val**,**row**]=**max**(X) works similarly to **min**(X).
mean(X)	The mean value of each column of X.
median(X)	The median value of each column of X.
std(X)	The standard deviation of each column of X.
var(X)	The variance of each column of X.
sum(X)	The columnwise sum of all values in X.
cumsum(X)	The columnwise cumulative sum of the values in X.
prod(X)	The columnwise product of all values in X.
cumprod(X)	The columnwise cumulative product of the values in X.
diff(X)	The difference between each consecutive element in X, columnwise.
cov(X)	Calculates the covariance matrix, assuming that each column in X represents outcomes of a variable.
corrcoef(X)	Calculates a matrix of correlation coefficients, assuming that each column in X represents outcomes

	of a variable.

For example, we want to calculate the mean amount of rain for the two remaining locations in testMatrix, we enter

```
>> mean(testMatrix(:,2:end))
ans =
    3.6667    7
```

If you want the commands to operate rowwise, instead of columnwise, a convenient way of doing that is to use the transpose operator twice, as in

```
>> mean(testMatrix(:,2:end)')'
ans =
    7.5
    4
    4.5
```

Note that, if X is a row vector, the commands here do not operate on the columns of X, but on the single row. For example

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

Matrix algebra

Here, we only present definitions of basic algebraic operators and how to use these operators in Matlab.

Matrix addition and subtraction

To add or subtract two matrices, they have to be of the same dimensions and all operations are element by-element. The definition for matrix addition is

$$\boldsymbol{X}+\boldsymbol{Y}=\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{T,1} & x_{T,2} & \cdots & x_{T,n} \end{bmatrix}+\begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,n} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{T,1} & y_{T,2} & \cdots & y_{T,n} \end{bmatrix}=\begin{bmatrix} x_{1,1}+y_{1,1} & x_{1,2}+y_{1,2} & \cdots & x_{1,n}+y_{1,n} \\ x_{2,1}+y_{2,1} & x_{2,2}+y_{2,2} & \cdots & x_{2,n}+y_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{T,1}+y_{T,1} & x_{T,2}+y_{T,2} & \cdots & x_{T,n}+y_{T,n} \end{bmatrix},$$

where both X and Y are Txn matrices (i.e., they have T rows and n columns).

```
>> X=[1 2 ; 3 4]; Y = [4 3 ; 2 1]; X+Y, Y-X
ans =
     5     5
     5     5
ans =
     3     1
    -1    -3
```

Note that you can add or subtract a scalar from a matrix of any dimensions. The scalar is then added to, or subtracted from, each element in the matrix.

```
>> X=[1 2 ; 3 4]; X-2
ans =
    -1     0
     1     2
```

Matrix multiplication

To multiply two matrices, the number of columns of the first matrix must equal the number of rows of the second. If X is a Txk matrix and Y is a kxn matrix, then X*Y is a Txn matrix.

$$\boldsymbol{X} * \boldsymbol{Y} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,k} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{T,1} & x_{T,2} & \cdots & x_{T,k} \end{bmatrix} * \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,n} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{k,1} & y_{k,2} & \cdots & y_{k,n} \end{bmatrix}$$

$$= \begin{bmatrix} x_{1,1} * y_{1,1} + x_{1,2} * y_{2,1} + \cdots + x_{1,k} * y_{k,1} & x_{1,1} * y_{1,2} + x_{1,2} * y_{2,2} + \cdots + x_{1,k} * y_{k,2} & \cdots & x_{1,1} * y_{1,n} + x_{1,2} * y_{2,n} + \cdots + x_{1,k} * y_{k,n} \\ x_{2,1} * y_{1,1} + x_{2,2} * y_{2,1} + \cdots + x_{2,k} * y_{k,1} & x_{2,1} * y_{1,2} + x_{2,2} * y_{2,2} + \cdots + x_{2,k} * y_{k,2} & \cdots & x_{2,1} * y_{1,n} + x_{2,2} * y_{2,n} + \cdots + x_{2,k} * y_{k,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{T,1} * y_{1,1} + x_{T,2} * y_{2,1} + \cdots + x_{T,k} * y_{k,1} & x_{T,1} * y_{1,2} + x_{T,2} * y_{2,2} + \cdots + x_{T,k} * y_{k,2} & \cdots & x_{T,1} * y_{1,n} + x_{T,2} * y_{2,n} + \cdots + x_{T,k} * y_{k,n} \end{bmatrix}$$

Note that the definition implies that, in general, X*Y ≠ Y*X, and that or it to be possible to calculate both X*Y and Y*X, both matrices have to be square matrices.

In Matlab, you calculate matrix multiplication by using the ordinary multiplication operator, *. For example, using the previously defined X and Y

```
>> X*Y, Y*X
ans =
     8     5
     20    13
ans =
     13    20
     5     8
```

Inverting a matrix

In Matlab, you calculate matrix inverse with inv() or by raising the matrix to the power of -1.

```
>> inv(X), X^-1
ans =
     -2     1
     1.5    -0.5
ans =
     -2     1
     1.5    -0.5
```

Matlab, however, also allows for matrix inversion using the division operators, / and \, although this notation is nonstandard. In these cases, the matrix that is above the division sign (i.e., to the left of / or to the right of \) is multiplied with the inverse of the other matrix. For example,

```
>> eye(2)/X, X\eye(2)
ans =
     -2     1
     1.5    -0.5
```

```
ans =
      -2     1
      1.5   -0.5
```

give the same results as above, and consequently also invert X. Furthermore

```
>> Y/X, Y*inv(X)
ans =
      -3.5   2.5
      -2.5   1.5
ans =
      -3.5   2.5
      -2.5   1.5
```

are two different ways of calculating $Y*X^{-1}$.

Matlab Exercises 4

1. Create a vector x containing integer numbers from 1 to 100.

```
>>
```

2. Create a vector y containing numbers 1, 0.9, 0.8, 0.7, . . . 0.1, 0 in this order.

```
>>
```

3. From x create y containing first 25 elements of x.

```
>>
```

4. From x create z containing elements of x with indexes from 50 to 75.

```
>>
```

5. From x create w containing elements with even indexes.

```
>>
```

6. Create matrix 2 by 5 with all elements equal to 0.37.

```
>>
```

7. Create a vector x = [3 1 2 5 4]. From x create y containing the same elements in the reverse order, find indices of elements greater than 2, create z containing elements of x which are smaller than 4.

```
>>

>>

>>

>>
```

8. Given the same matrix m = [1 2 3; 2 1 5; 4 6 4; 2 3 2], create matrix n with rows sorted in a descending order of elements in the second column.

Example:

```
1 2 3          4 6 4
2 1 5    =>    2 3 2
4 6 4          1 2 3
2 3 2          2 1 5
```

```
>>
```

9. Run the Matlab code:

```
>> a = 1:5

>> d = a+2 * a

>> e = d'
```

10. Given $A = \begin{bmatrix} 1 & 2 & 4 \\ 1 & 1 & 1 \\ 2 & 3 & 1 \end{bmatrix}$, and $B = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$.Run the Matlab code:

```
>> GREATER= A>B

>> GREATER2 = A>1
```

11. Given $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$, $x = [-5 \ -10 \ -15]$. Run the Matlab code:

```
>> D1 = diag(A)

>> D2 = diag(diag(A))

>> D3 = diag(x)
```

12. Rewrite the following equation as Matlab expression:

$$y = \frac{4x + 2}{x}$$

```
>>
```

13. Rewrite the following equation as Matlab expression:

$$y = 3x^2 + \sqrt{x^2 + y^2} + e^{\ln(x)}$$

```
>>
```

14. Rewrite the following equation as Matlab expression:

$$y = \frac{\log(ax^2 + bx + c) - \sin(ax^2 + bx + c)}{4\pi x^2 + \cos(x - 2)(ax^2 + bx + c)}$$

```
>>
```

15. Rewrite the following equation as Matlab expression:

$$\frac{1}{2 + 3^2} + \frac{4}{5} \times \frac{6}{7}$$

```
>>
```

16. Rewrite the following equation as Matlab expression:

$$23\left(-8 + \frac{\sqrt{607}}{3}\right) + \left(\frac{40}{8} + 4.7^2\right)^2$$

```
>>
```

17. Rewrite the following equation as Matlab expression:

$$509^{\frac{1}{3}} - 4.5^2 + \frac{ln200}{1.5} + 75^{\frac{1}{2}}$$

```
>>
```

18. Run the following Matlab code:

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


>> A([3 1],[3 2])


>>A=[A(1,:); A(2,:); [7 8 0]]


>>B=[A 10*A; -A eye(3)]
```

19. Run the following Matlab code:

```
>> DF=1:4

>> DF(5:10)=10:5:35

>> AD=[5 7 2]

>> AD(8)=4
```

```
>> AR(5)=24
```

20. Run the following Matlab code:

```
>> AR=zeros(6,6); AR(3:4,:)=ones(2,6); AR(:,3:4)=ones(6,2)
```

21. Run the following Matlab code:

```
>> A=[2:3:17; 3:3:18; 4:3:19; 5:3:20; 6:3:21]; B=[5:5:30; 30:5:55; 55:5:80]; v=[99:-1:91];
>> A([1 3 4 5],3:6)=[B([1 2],1:4); v(5:8); B(3,2:5)]
```

22. Using the zeros, ones, and eye commands create the following arrays:

a) $\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$ *b)* $\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$ *c)* $\begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$

Solving Systems of Linear Equations

5

Matrix algebra provides an easy way to solve systems of linear equations. Suppose we have the following system of three equations:

$$\begin{aligned} x_1 \ +2x_2+3x_3&=15 \\ x_1 \ +2x_2+ \ x_3 &=13 \\ -2x_1+5x_2-2x_3&=19 \end{aligned}$$

Such a system can be written in matrix notation as A*X = Y, where

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 1 \\ -2 & 5 & -2 \end{bmatrix}, X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \ and \ Y = \begin{bmatrix} 15 \\ 13 \\ 19 \end{bmatrix}$$

Now, if we multiply each side of the equality by the inverse of A, we get

$$A^{-1}*A*X = A^{-1}*Y$$

$$I*X = A^{-1}*Y$$

$$X = A^{-1}*Y$$

To calculate the solution in Matlab, we use either the inv() command or \.

```
>> A=[1 2 3 ; 1 2 1 ; -2 5 -2]; Y=[15 13 19]'; X=A\Y, X=inv(A)*Y
X =
     2
     5
     1
```

```
X =
     2
     5
     1
```

Consequently, $x_1 = 2$, $x_2 = 5$, and $x_3 = 1$. To check this, issue

```
>> A*X
ans =
     15
     13
     19
```

which equals Y.

Let's take another example, consider the system of equations:

$$x + 2y + 3z = 1$$
$$3x + 3y + 4z = 1$$
$$2x + 3y + 3z = 2$$

To find the solution X to the system of equations.

```
>>A = [1 2 3; 3 3 4; 2 3 3];
>>b = [1; 1; 2];
>>x = A\b
x =
     -0.5000
     1.5000
     -0.5000
```

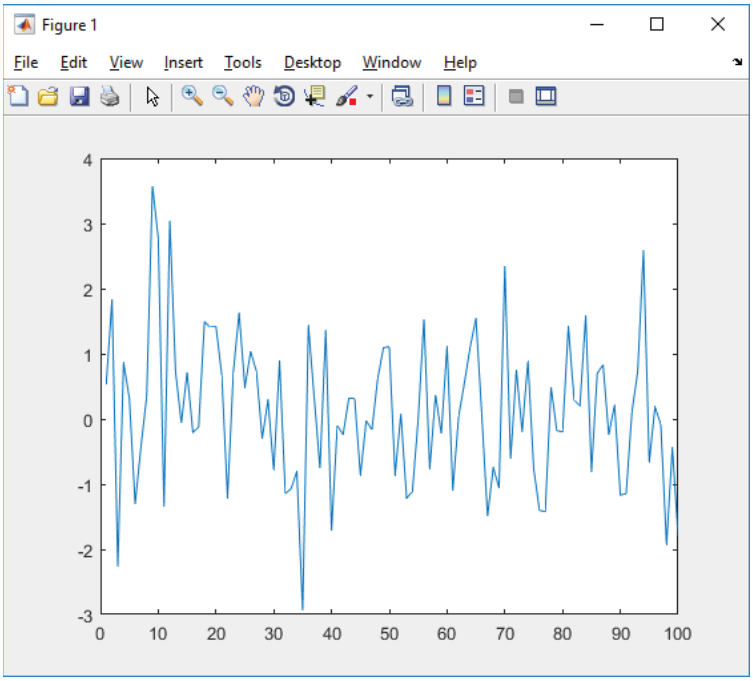
Graphics

6

There are numerous ways to produce graphs in Matlab. To begin, we need some data to work with. For simplicity, let us use some random data.

```
>> x=randn(1,100);
>> plot(x)
```

This produces a plot of the variable X.



The most frequently used plotting commands include the following.

plot(x,y)	• Produces a two-dimensional plot. x should be a vector of X-values and y should be a vector or a

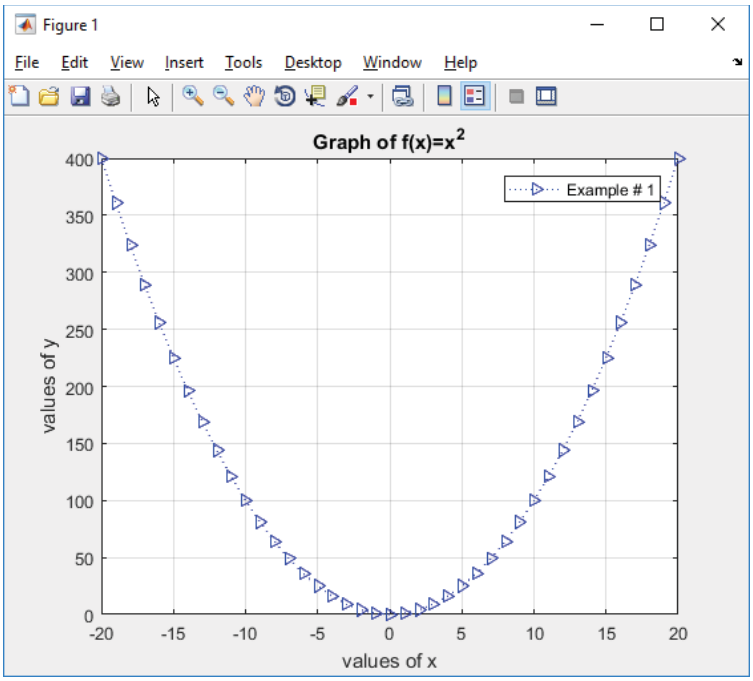
	matrix with corresponding Y-values. • They must be of equal length. • Note that, if there is already a plot in the figure window, it is erased and replaced by the new one. • You can plot several lines by entering x- and y-vectors alternatively in the plot command, for example plot(x1,y1,x2,y2,x3,y3). • You can also add code within single quotes about how you want the data to be presented. For example, plot(x,y ,'r:d') produces a red dotted line with diamond markers. If no line type is entered, solid is the default. Codes to enter include: ○ *colors:* b blue; g green; r red; c cyan; m magenta; y yellow; k black; w white ○ *lines:* - solid; : dotted; -. dashdot; -- dashed ○ *markers:* . point; o circle; x x-mark; + plus; * star; s square; d diamond; v triangle; ^ triangle; < triangle; > triangle; p pentagram.
hold	• Keeps an existing plot in the figure window when a new one is plotted. • hold on turns the feature on and hold off turns it off. • hold toggles between on and off.
grid	• Adds a grid to the graph. • grid on turns the grid on and grid off turns it off. • grid toggles between on and off.
xlabel, ylabel	• Adds a label to the X- and Y-axis, respectively. For example, xlabel('year') makes the text "year" appear below the X-axis.
title	• Adds a name at the top of the graph. For example, title ('Interest rates').
legend	• Adds a legend to the graph indicating what the lines represent. For example, legend('short rate','long rate')

axis	• Controls the appearance of the axes. axis([0 10 2 20]) makes the X-axis run from 0 to 10 and the Y-axis from 2 to 20.
xlim, ylim	• Similar to axis but operate only on the X- and Y-dimension, respectively.
Figure	• Opens up a new figure window (so that a new plot will not erase an old one).
Subplot	• Partitions the figure window into several rows and/or columns and selects which of these to issue plotting commands to. For example, subplot(4,2,7) partitions the figure window into four rows and two columns and selects the seventh subsection. (Note: counting rowwise, not columnwise. The seventh subsection is, consequently, on the fourth row and in the first column.)

Now, let's take some example:

```
>> x=-20:20;
>> y=x.*x;
>> plot(x,y,'b:>')
>> grid
>> title('Graph of f(x)=x^2')
>> xlabel('values of x'), ylabel('values of y')
>> legend('Example # 1')
```

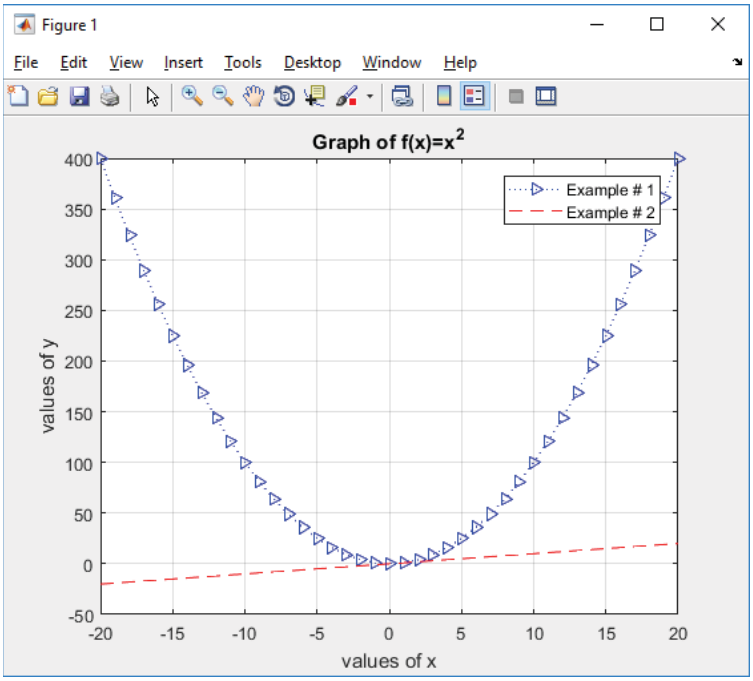
and the result of the above code will be the following figure:



Now, we will continue with the following lines of code:

```
>> z=x;
>> hold on
>> plot(x,z,'r--')
>> legend('Example # 1','Example # 2')
```

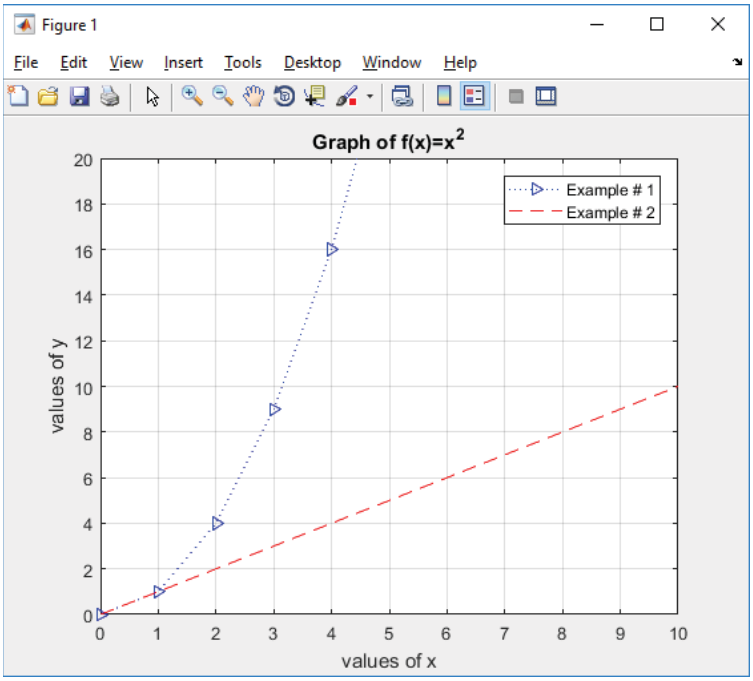
The figure will be updated to be as follows



We can restrict the view for a specific range

```
>> xlim ([0 10])
>> ylim ([0 20])
```

and the result will be the following figure:



Note that we can replace the above two lines of code by:

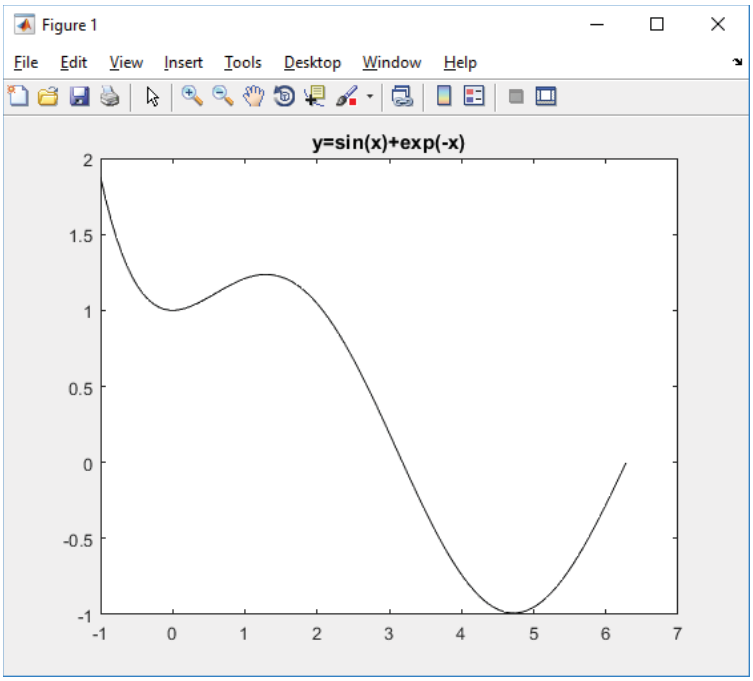
```
>> axis([0 10 0 20])
```

to get the same result.

Let's take another example:

```
>> x=linspace(-1,2*pi,100);
>> y=sin(x)+exp(-x);
>>plot(x,y,'k-');
>> title('y=sin(x)+exp(-x)');
```

This code generates two vectors x and y, then plots the curve that represent the relation between them using a solid black line with the title (y=sin(x)+exp(-x).

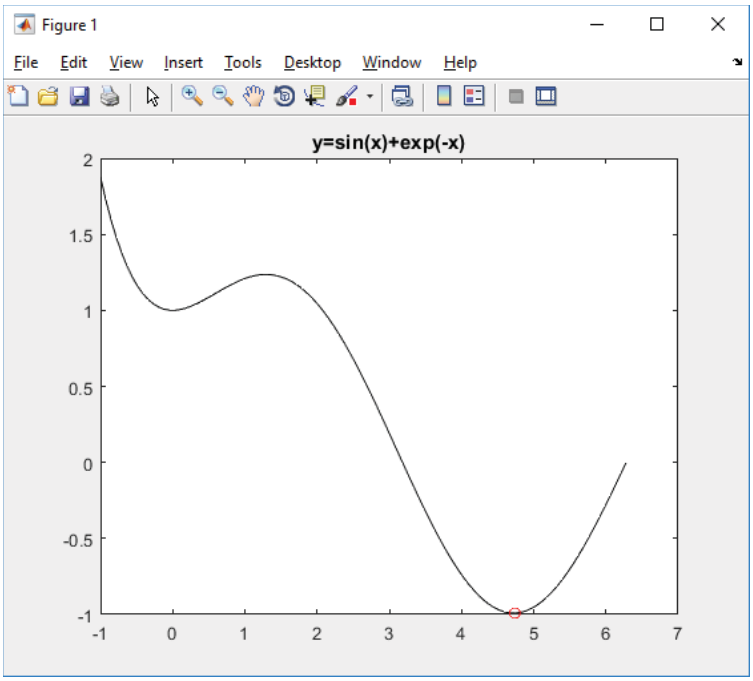


For a given set of X- and Y-values, it is also easy to add markers at specific values, for example at a function minimum. When using the min() function with two output arguments, it supplies both the minimum value and the location of that number

```
>> [minVal,minLoc] = min(y)
minVal =
       -0.99091
minLoc =
       79
```

Since the X- and Y-values are paired, the X-value that corresponds to the minimum Y-value must be in the same location in x. We can then put a red o-marker in the correct spot by issuing

```
>> hold on; plot(x(minLoc),y(minLoc),'ro')
```

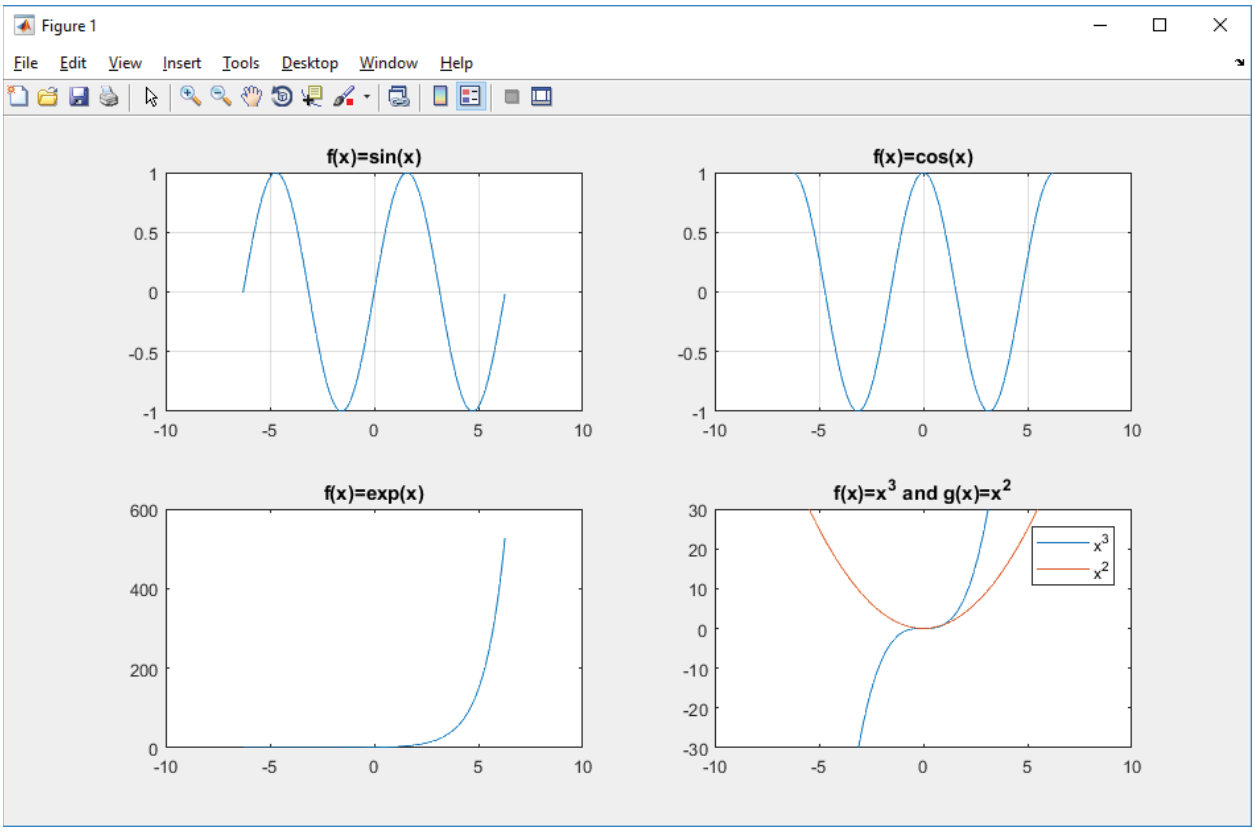


Several graphs in one window and other types of graphs

It is often useful to plot several graphs in the same figure window. To accomplish this, we use the command subplot() that partitions the figure window into several rows and/or columns.

```
>> x=-2*pi:0.05:2*pi;
>> y=sin(x); z=cos(x); w=exp(x); r=x.^3; t=x.^2;
>> subplot(2,2,1), plot(x,y), grid on
>> title('f(x)=sin(x)')
>> subplot(2,2,2), plot(x,z), grid on
>> title('f(x)=cos(x)')
>> subplot(2,2,3), plot(x,w), grid off
>> title('f(x)=exp(x)')
>> subplot(2,2,4), plot(x,r,x,t), grid off
>> title('f(x)=x^3 and g(x)=x^2')
>> axis([-10 10 -30 30])
>> legend('x^3','x^2')
```

In the following figure you can see the graphs in the partitioned figure window.



Scripts

7

So far in these lab sessions, all the commands were executed in the Command Window. The problem is that the commands entered in the Command Window cannot be saved and executed again for several times. Therefore, a different way of executing repeatedly commands with MATLAB is:

1. to create a file with a list of commands,
2. save the file, and
3. run the file.

If needed, corrections or changes can be made to the commands in the file. The files that are used for this purpose are called script files or scripts for short. This section covers the following topics:

- M-File Scripts
- M-File Functions

M-File Scripts

A script file is an external file that contains a sequence of MATLAB statements. Script files have a filename extension .m and are often called M-files. M-files can be scripts that simply execute a series of MATLAB statements, or they can be functions that can accept arguments and can produce one or more outputs.

Example 1: Consider the system of equations

$$x_1 + 2x_2 + 3x_3 = 1$$
$$3x_1 + 3x_2 + 4x_3 = 1$$
$$2x_1 + 3x_2 + 3x_3 = 2$$

Find the solution x to the system of equations.

Solution:

- Use the MATLAB editor to create a file: File ➔ New ➔ M-file.
- Enter the following statements in the file:

```
A = [1 2 3; 3 3 4; 2 3 3];
b = [1; 1; 2];
x = A\b
```

- Save the file, for example, **example1.m**.
- Run the file, in the command line, by typing:

```
>> example1
x =
      -0.5000
       1.5000
      -0.5000
```

Example 2: Plot the following cosine functions, $y_1 = 2\cos(x)$, $y_2 = \cos(x)$, and $y_3 = 0.5 * \cos(x)$, in the interval $0 \leq x \leq 2\pi$.

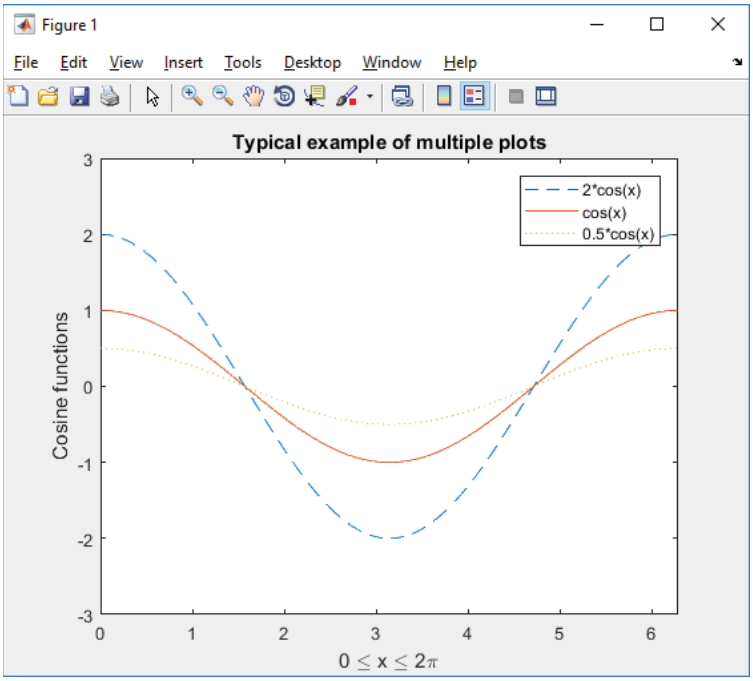
Solution:

- Create a file, say **example2.m**, which contains the following commands:

```
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])
```

- Run the file by typing example2 in the Command Window.



M-File functions

Functions are programs (or routines) that accept input arguments and return output arguments. Each M-file function (or function or M-file for short) has its own area of workspace, separated from the MATLAB base workspace.

The general form looks like this

function [outputs] = function_name(inputs)

Function file can have none, one, or several output arguments.

Example of input and output arguments

function C=FtoC(F)	One input argument and one output argument
function area=TrapArea(a,b,h)	Three inputs and one output
function [h,d]=motion(v,angle)	Two inputs and two outputs

Example 1: Write a function to calculate the factorial value for any number.

Solution:

```
function f = factorial(n)
% FACTORIAL(N) returns the factorial of N.
% Compute a factorial value.
f = prod(1:n);
end
```

As an example, for n = 5, the result is,

```
>> f = factorial(5)
f =
      120
```

Example 2: Write a function to calculate the values of $u= x^2+y^2$ and $v=x^2-y^2$ for any given (x,y).

Solution:

```
function [u v] = plmin(x,y)
u=x^2+y^2;
v=x^2-y^2;
end
```

As an example, for x = 2 and y=4, the result is,

```
>> [res1 res2]=plmin(2,4)
res1 =
   20
res2 =
  -12
```

Note that if you write plmin(2,4) without the left hand side you will get the value of the first result only.

```
>> plmin(2,4)
ans =
   20
```

You can re-use you predefined function within another function.

Example 3: Write a function to calculate the product of the factorials of given x and y. Hint use the function in example1.

Solution:

```
function r = fp(x,y)
r= factorial(x)*factorial(y);
end
```

As an example, for x = 2 and y=4, the result is,

```
>> fp(2,4)
ans =
   48
```

Input to a script file

When a script file is executed, the variables that are used in the calculations within the file must have assigned values. The assignment of a value to a variable can be done in three ways.

1. The variable is defined in the script file.
2. The variable is defined in the command prompt.
3. The variable is entered when the script is executed.

We have already seen the two first cases. Here, we will focus our attention on the third one.

```
% This script file calculates the average of marks
% scored in three Exams.
exam1 = input('Enter the marks scored in the first exam ');
exam2 = input('Enter the marks scored in the second exam ');
exam3 = input('Enter the marks scored in the third exam ');
average = (exam1+exam2+exam3)/3
```

The following shows the command prompt when this script file (saved as exampleav) is executed.

```
>> exampleav
Enter the marks scored in the first exam 90
Enter the marks scored in the second exam 95
Enter the marks scored in the third exam 65

average =
      83.3333
```

Exercises

1. Ahmed buys three apples, a dozen bananas, and one cantaloupe for $2.36. Ali buys a dozen apples and two cantaloupes for $5.26. Noor buys two bananas and three cantaloupes for $2.77. How much do single pieces of each fruit cost?

2. Write a function file that converts temperature in degrees Fahrenheit (±F) to degrees Centigrade (±C). Recall the conversion formulation, C = 5/9 * (F - 32).

3. 3. The distance d from a point $(x_0, y_0)$to a line $Ax + By + C = 0$ is given by:

$$d = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$$

Write a function to calculate that distance and determine the distance of the point (3,-4) from the line $2x-7y-10=0$.

Hint: First define the variables A, B, C, x_0, and y_0, and then calculate d. (Use the abs and sqrt functions.)

Control Flow

8

MATLAB has four control flow structures: the **if** statement, the **for** loop, the **while** loop, and the **switch** statement.

1. The "if...end" structure

MATLAB supports the variants of " if " construct.

- if ... end
- if ... else ... end
- if ... elseif ... else ... end

The simplest form of the if statement is

```
if expression
    statements
end
```

Here are some examples based on the familiar quadratic formula.

```
d = b*b - 4*a*c;
if d < 0
    disp('Warning: discriminant is negative, roots are imaginary');
end
```

anther example:

```
d = b*b - 4*a*c;
if d < 0
    disp('Warning: discriminant is negative, roots are imaginary');
else
    disp('Roots are real, but may be repeated')
```

```
end
```

and here is another example:

```
d = b*b - 4*a*c;
if d < 0
        disp('Warning: discriminant is negative, roots are imaginary');
elseif d == 0
        disp('Discriminant is zero, roots are repeated')
else
        disp('Roots are real')
end
```

A relational operator compares two numbers by determining whether a comparison is true or false.

Relational and logical operators

Operator	Description
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
~=	Not equal to
&	AND operator
|	OR operator
~	NOT operator

Lets take another example

```
function grade = score2grade(testScore)
if testScore >= 90
        grade = 'A';
elseif testScore >= 80
```

```
        grade = 'B';
elseif testScore >= 70
        grade = 'C';
else
        grade = 'fail';
end
```

2. The "for...end" loop

In the for ... end loop, the execution of a command is repeated at a fixed and predetermined number of times. The syntax is

```
for variable = expression
        statements
end
```

Usually, expression is a vector of the form i:s:j. A simple example of for loop is

```
for ii=1:4
        x=ii*ii
end
```

running the above code gives:

```
x =
     1
x =
     4
x =
     9
x =
    16
```

Multiple for loops can be nested, the following statements form the 5-by-5 symmetric matrix A with (i; j) element i/j for j ≥ i:

```
n = 5; A = eye(n);
    for j=2:n
        for i=1:j-1
            A(i,j)=i/j;
            A(j,i)=i/j;
        end
    end
```

and the result will be

```
A =
   1.0000   0.5000   0.3333   0.2500   0.2000
   0.5000   1.0000   0.6667   0.5000   0.4000
   0.3333   0.6667   1.0000   0.7500   0.6000
   0.2500   0.5000   0.7500   1.0000   0.8000
   0.2000   0.4000   0.6000   0.8000   1.0000
```

To give an example of a simple loop, consider the case of the Fibonacci numbers. This is a series of numbers where the first two are equal to one, and then each consecutive number is equal to the sum of the previous two. The first twelve are then 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, and 144.

```
function fib = fibonacci(maxn)
    fib = ones(1,maxn);
    for i=3:maxn
        fib(i) = fib(i-1)+fib(i-2);
    end
```

Now we can use the above function to calculate the first 10 terms of the fibonacci sequence

```
>> fibonacci(10)
ans =
   1   1   2   3   5   8   13   21   34   55
```

3. The "while...end" loop

This loop is used when the number of passes is not specified. The looping continues until a stated condition is satisfied. The while loop has the form:

```
while expression
      statements
end
```

The statements are executed as long as expression is true.

```
x = 1
while x <= 10
      x = 3*x
end
```

running this code gives

```
x =
    1
x =
    3
x =
    9
x =
   27
```

It is important to note that if the condition inside the looping is not well defined, the looping will continue indefinitely. If this happens, we can stop the execution by pressing Ctrl-C.

4. The "switch...end" loop

A switch block conditionally executes one set of statements from several choices. Each choice is covered by a case statement.

```
switch expression
  case value1
    statements
  case value2
```

```
        statements
        ...
        ...
    otherwise
        statements
end
```

Lets take an example:

```
grade = 'B';
  switch(grade)
  case 'A'
    disp('Excellent!' );
  case 'B'
    disp('Well done' );
  case 'C'
    disp('Well done' );
  case 'D'
    disp('You passed' );
  case 'F'
    disp('Better try again' );
  otherwise
    disp('Invalid grade' );
  end
```

running the above code gives

```
Well done
```

Lets take another example

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

```
        disp('zero')
    case 1
        disp('positive one')
    otherwise
        disp('other value')
end
```

running the above code gives

```
Enter a number: 1
positive one

Enter a number: -1
negative one

Enter a number: 0
Zero

Enter a number: 4
other value
```

Our last example is:

```
result = 52;
switch(result)
   case 52
      disp('result is 52')
   case {52, 78}
      disp('result is 52 or 78')
end
```

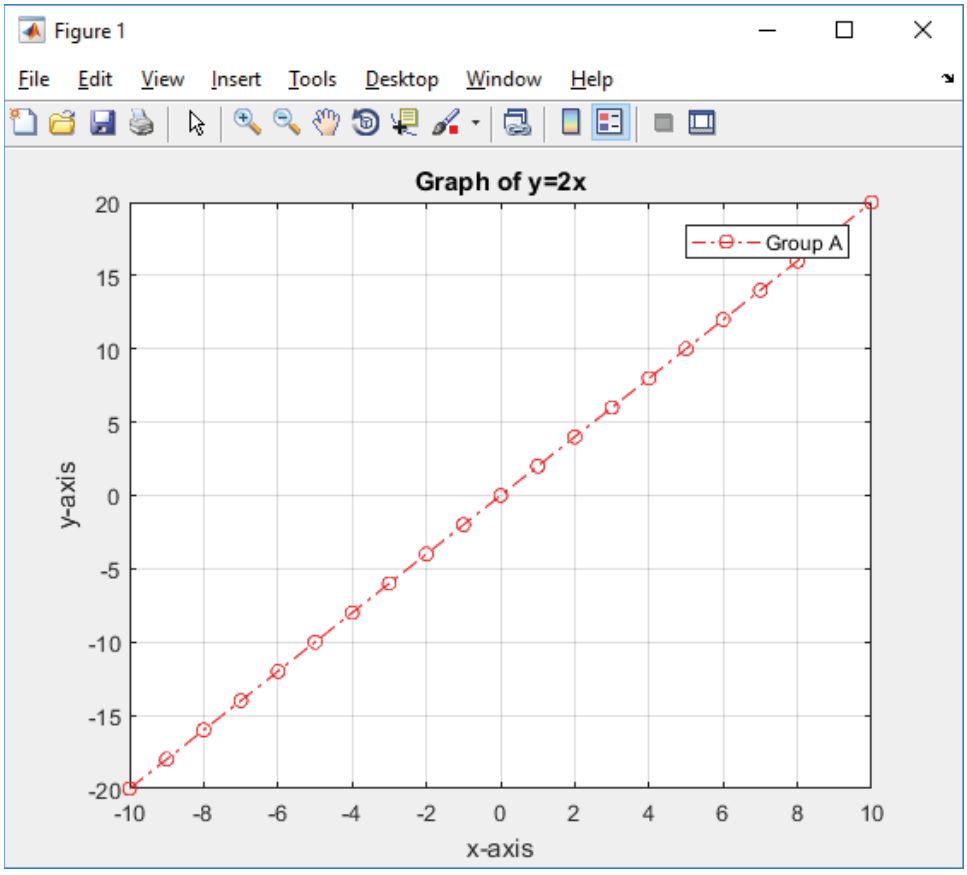
running the above code gives

```
result is 52
```

Matlab Exercises 9

1. Trace the following code:

```
x=-10:10; y=2*x;
plot(x,y,'r-.o'); grid
title('Graph of y=2x')
xlabel('x-axis'), ylabel('y-axis')
legend('Group A')
```



2. Given the following function:

```
function f = fr(n)
n=n+1;
y=1:n;
f = sum(sum(y));
```

What will be the result of the following command?

```
>> fr(5)
ans =
   21
```

3. Write a function file that converts amount of money from Dollar to Iraqi Dinar. Note that, each Dollar equal to 1200 Iraqi Dinar.

```
function dinar = us2iq(dollar)
dinar=1200*dollar;
```

and we can use the above function as follows:

```
>>us2iq(120)
ans =
    144000
```

4. Trace the following code:

```
for i=1:3
   for j=1:2
      switch i
         case 2
            A(i,j)=3;
         otherwise
            A(i,j)=j;
      end
```

```
   end
end
A
```

Running the above code gives

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

5. Trace the following code:

```
for i=1:2
   for j=1:3
      if j>1
         A(i,j)=3;
      else
         A(i,j)=i;
      end
   end
end
A
```

Running the above code gives

```
A =


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

6. Given the following function:

```
function f = fr(n)
a=eye(n+1);
a(4)=3;
f=a+1;
```

What will be the result of the following command?

```
>> fr(2)
ans =
    2    4    1
    1    2    1
    1    1    2
```

7. Write a function file that compute the distance between two points (x1,y1) and (x2,y2) using the following equation

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
function dis = d(x1,y1,x2,y2)
dis=sqrt((x2-x1)^2+(y2-y1)^2);
```

and we can use the above function to find the distance between (10,2) and (4,3) as follows:

```
>>d(10,2,4,3)
ans =
   6.0828
```

8. Trace the following code:

```
for i=1:3
   for j=1:3
      switch i
         case 3
```

```
            A(i,j)=j+1;
        otherwise
            A(i,j)=2;
    end
  end
end
A
```

Running the above code gives

```
A =
    2    2    2
    2    2    2
    2    3    4
```

9. Given the following function:

```
function f = fr(n)
a=ones(n+1);
a(6)=4;
f=a+1;
```

What will be the result of the following command?

```
>> fr(2)
ans =
    2    2    2
    2    2    2
    2    5    2
```

10. Trace the following code:

```
for i=1:3
   for j=1:3
      if j==1
         A(i,j)=3;
      else
         A(i,j)=i;
      end
   end
end
A
```

Running the above code gives

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

11. How many times will the display command in the following script be executed?

```
x = 3;
while (x < 8)
      disp('Am I done yet?')
      x = x + 2.5;
end
```

it is easy to see that x will take the values 3, and 5.5 then it will be greater than 8, so the display command will executed ***2*** times.

12. What will be displayed when you run the code below?

```
a = 0;
while a < 10
        a = a + 3;
end
disp (a)
```

The answer is:

```
   12
```

13. Given matrix A = [0 2 1; 3 1 0; 4 6 4; 2 0 2], create a matrix with 1's at locations where A has zeros and 0's elsewhere.

```
A=[0 2 1; 3 1 0; 4 6 4; 2 0 2]
[r c]=size(A);
for i=1:r
   for j=1:c
      if A(i,j)==0
         A(i,j)=1;
      else
         A(i,j)=0;
      end
   end
end
A
```

Running the above code gives:

```
A =
    0    2    1
    3    1    0
    4    6    4
    2    0    2
```

```
A =
    1    0    0
    0    0    1
    0    0    0
    0    1    0
```

14. Given matrix A = [0 2 1; 3 1 0; 4 6 4; 2 0 2], Create a matrix containing all 0's except the maximum elements in each row of a (i.e. b = [0 2 0; 3 0 0; 0 6 0; 2 0 2]).

```
A=[0 2 1; 3 1 0; 4 6 4; 2 0 2]
[r c]=size(A);
for i=1:r
   m=max(A(i,:));
   for j=1:c
      if A(i,j)~=m
         A(i,j)=0;
      end
   end
end
A
```

Running the above code gives:

```
A =
    0    2    1
    3    1    0
    4    6    4
    2    0    2
A =
    0    2    0
    3    0    0
```

```
    0     6     0
    2     0     2
```

15. Given a vector x = [3 1 4] and integer number n = 5, create vector y containing n-times x(1), n-times x(2), etc. (i.e., y = [3 3 3 3 3 1 1 1 1 1 4 4 4 4 4]).

```
Clear all
x=[3 1 4]
k=0;
a=input('How many times you want to repeat ');
[r c]=size(x);
for i=1:c
   for j=1:a
      k=k+1;
      y(k)=x(i);
   end
end
y
```

Running the above code gives:

```
x =
    3     1     4
How many times you want to repeat 3
y =
    3     3     3     1     1     1     4     4     4
```

16. Given a vector x = [3 1 7 0 3 0 2 0 0 4 0], create a vector that contains the same elements of x but without zeros. (i.e., y = [3 1 7 3 2 4]).

```
clear all
x=[3 1 7 0 3 0 2 0 0 4 0]
```

```
k=0;
[r c]=size(x);
for i=1:c
   if x(i)~=0
      k=k+1;
      y(k)=x(i);
   end
end
y
```

Running the above code gives:

```
x =
    3    1    7    0    3    0    2    0    0    4    0
y =
    3    1    7    3    2    4
```

*17. Write the screen display for the following script and associated functions in the space provided.

```
%script
x = 4
y = 3
z = [2 4 5 3 1];
for i = 1:2:5
        if (i <= 2)
            back1 = fcn1(i,x,y,z)
        elseif (i > 3)
            disp('done')
        else
            back2 = fcn2(i,y,x,z)
        end
end
```

```
function [out2] = fcn1(j,y,x,k)
out2 = [0 0 0 0 0]
for i=1:3:5
        out2(i) = fcn2(i,x,y,k);
end
```

```
function [out1] = fcn2(j,d,s,k)
switch (k(j))
        case {1,2}
            out1 = d+9
        case {3,4}
            out1 = 2*s
        otherwise
            out1 = d+s
end
```

Solution:

Display #	Display
1	x = 4
2	y = 3
3	out2 = 0 0 0 0 0
4	out1 = 12
5	out1 = 8
6	back 1 = 12 0 0 8 0
7	out1 = 7
8	back2 = 7
9	done