Signals & Systems

EEE-223

Lab. Manual

Name	Muhammad Haris Irfan
Registration Number	FA18-BCE-090
Class	BCE-6B
Instructor's Name	Sir Bilal Qasim

Introduction

This is the Lab Manual for EEE – 223 Signals and Systems. The labs constitute 25 % of the total marks for this course.

During the labs you will work in groups (no more than three students per group). You are required to complete the 'Pre-Lab' section of the lab before coming to the lab. You will be graded for this and the 'In-Lab' tasks during the in-lab viva. You will complete the 'Post-Lab' section of each lab before coming to the next week's lab.

You are not allowed to wander in the lab or consult other groups when performing experiments. Similarly, the lab reports must contain original efforts. CUI has a zero tolerance anti-plagiarism policy.

Apart from these weekly labs you will sit in two lab sessional exams accompanied by viva. The lab sessional exams will be based on tasks performed in the last 3-4 labs before the respective sessional week. Your lab #12 will be an open-ended lab task as opposed to the previous eleven labs that are instruction-based. Also, a Final Project will be given that will be graded as your Lab Final Exam. The grading policy is already discussed in the Course Description File.

Acknowledgement

The labs for EEE 223 Signals and Systems were designed by Dr. M. Faisal Siddiqui and Mr. Ali Waqar Azim. The first version was completed in Session Spring 2012, The second version was completed during the summer break of 2016 and now the third iteration of its revision is done in January 2017. Typesetting and formatting of this version was supervised by Dr. Omar Ahmad and was carried out by Mr. Abdul Rehman, Mr Suleman & Mr Baqir Hussain.

History of Revision

Date of Issue	Team	Comments
April 2012	Dr. Faisal Siddiqui Mr. Ali Waqar Azim	This is the first editable draft of EEE223 lab manual.
July 2016	-	First revision
January 2017	Ms. Mehwish Mehmood Dr. Muhammad Aurangzeb Khan Mr. Bilal Qasim	Second revision
February 2019	Mr. Mubeen Sabir Dr. Shurjeel Wyne	Third revision
March 2020	Nida Zamir	Fourth revision

Safety Precautions

- Be calm and relaxed, while working in lab.
- First check your measuring equipment.
- When working with voltages over 40 V or current over 10 A , there must be at least two people in the lab at all time.
- Keep the work area neat and clean.
- Be sure about the locations of fire extinguishers and first aid kit.
- No loose wires or metals pieces should be lying on the table or neat the circuit.
- Avoid using long wires, that may get in your way while making adjustments or changing leads.
- Be aware of bracelets, rings, and metal watch bands (if you are wearing any of them). Do not wear them near an energized circuit.
- When working with energize circuit use only one hand while keeping rest of your body away from conducting surfaces.
- Always check your circuit connections before power it ON.
- Always connect connection from load to power supply.
- Never use any faulty or damage equipment and tools.
- If an individual comes in contact with a live electrical conductor.
    - Do not touch the equipment, the cord, the person.
    - Disconnect the power source from the circuit breaker and pull out the plug using insulated material

Table of Contents

LAB # 01

Introduction to MATLAB: Basic Commands and Array Manipulation

Lab 01- Introduction to MATLAB: Basic Commands and Array Manipulation

Once MATLAB® is installed; it is time to start using it. MATLAB is best understood by entering the instructions in your computer one at a time and by observing and studying the responses. Learning by doing is probably the most effective way to maximize retention.

I hear and I forget
I see and I remember
I do and I understand

Confucius

Pre-lab

1.1 MATLAB

MATLAB (short for MATrix LABoratory) is a platform (program) organized for optimized engineering programming and scientific calculations. The MATLAB program implements the MATLAB programming language and provides an extensive library of predefined functions and make technical programming tasks easier and more efficient. MATLAB has incredibly large variety of functions and toolkits with more functions and various specialties.

1.1.1 The Advantage of MATLAB

Some of the advantages of MATLAB are:

1. Ease of Use.
2. Platform Independence.
3. Predefined Functions
4. Device independent plotting.
5. Graphical User Interface.

1.1.2 The disadvantages of MATLAB

The only disadvantage of MATLAB is that it is an interpreted language rather than complied language. Because of the same reason the it executes more slowly than compiled languages such as C, C++.

1.2 Getting Started with MATLAB

Open MATLAB through the start menu or from the desktop icon. The interface will be like this

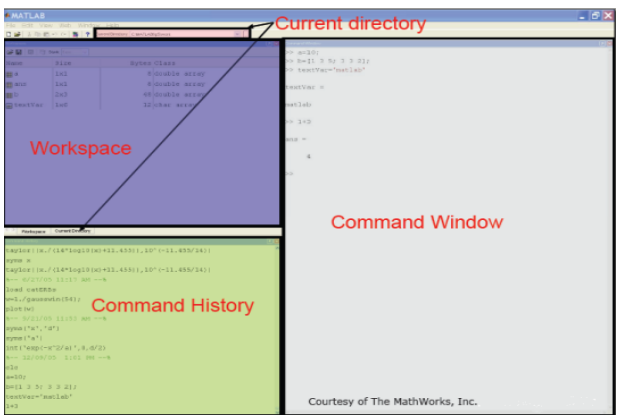


Fig.1. MATLAB Interface

Current Directory: Shows the current folder of MATLAB.
Workspace: Shows the size of variables which are initiated on the command window.
Command History: Shows the history of commands used previously on command window.
Command Window: Command are written in command windows.

1.3 Working with Variables

The fundamental unit of data in MATLAB is **array.** A common **programming mistake** is to use a variable without declaring it first. This is not a problem in MATLAB. In fact, MATLAB is very **flexible about variables and data types**. MATLAB variable is automatically created when they are initialized. There are three common ways to initialize a variable in MATLAB:

1. Assign data to the variable in the assignment statement.
2. Input data into the variable from the keyboard.
3. Read data from a file.

The first way will be defined here and the remaining two approached will be discussed in later labs if time permits.

1.3.1 Initializing variables in Assignments Statements:

The simplest way to create a variable is to assign it one or more values in an assignment statement. An assignment statement has the general form.

var = expression

where **var** is the name of the variable, and the expression is a scalar constant, an array, or combination of constants, vectors or matrix.

1.3.2 Variable Initialization:

If an undefined variable is assigned a value, for example:

```
a = 1;
```

MATLAB will assume that the variable **a** should be created, and that the value **1** should be stored in it. If the variable **already exists**, then it will have the number **1** **replace its previous value**. This works with **scalars, vectors, and matrices**. The **numbers stored in variables can be integers** (e.g., 3), **floating-points** (e.g., 3.1), or **even complex** (e.g., 3.1+5.2j).

```
a = 3;
b = 3.1;
c = 3.1+5.2j;
```

1.3.3 Semicolon Operator:

A semicolon **";"** is often put at the **end of a command**. This has the effect of **suppressing** the output of the resulting values. Leaving of the semicolon does not affect the program, just what is **printed to the screen**, though printing to the screen can make a **program run slowly**. If a **variable name is typed on a line by itself**, then the **value(s) stored for that variable will be printed**. This also works inside a function or a program. A variable on a line by itself (without a semicolon) will be displayed.

```
>> a
a =
   3
>> b
b =
   3.1000
>> c
c =
   3.1000 + 5.2000i
```

1.3.4 Complex Variable:

MATLAB shows the complex value stored in **c** with **i** representing the complex part (the square root of -1). In the above example of complex number **j** is used for indicating the square root of -1, instead of **j, i** can also be used to indicate the square root of -1.

1.4 MATLAB Programming Basics

This section covers the main things that you need to know to work with MATLAB.

1.4.1 How to clear command screen:

The command **clc** **clears the command window**, and is similar to the clear command in the Unix or Dos environments.

1.4.2 How to get rid of variables:

The command **clear**, by the way, tells MATLAB to **get rid of all variables**, or can be used to get rid of a **specific variable**, such as **clear a**.

1.4.3 Information about variables:

To see a list of the variables in memory, type **who**. The similar command **whos** gives the same list, but with more information.

```
>> who
Your variables are:
a  b  c
>> whos
  Name      Size            Bytes  Class     Attributes
  a         1x1                 8  double
  b         1x1                 8  double
  c         1x1                16  double    complex
>> clear
>> who
>> whos
```

In the example above, we see that the variable **a**, **b** and **c** are defined, and the difference between **who** and **whos**. After the clear statement, the **who** and **whos** command returns nothing since we got rid of the variables.

1.5 Scalars, Vectors, and Matrices

1.5.1 Scalar Declaration:

A variable can be given a **"normal"** (scalar) value, such as **b=3** or **b=3.1**, but it is also easy to give a variable **several values** at once, making it a **vector**, or even a **matrix**.

1.5.2 Vector Declaration:

For example, **c=[1 2]** stores both numbers in variable **c**.

1.5.3 Matrix Declaration:

To **make a matrix**, use a semicolon (";") to separate the rows, such as:

d=[1 2; 3 4; 5 6]

This makes d a matrix of 3 rows and 2 columns. Please note that

d(row specifier, column specifier)

is the correct version of to **access the element** at specified row and specific column.

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

1.5.3.1 How to add a row in matrix:

What if you want to add another row onto **d**? This is easy, just give it a value: **d(4,:)=[7 8].** Notice the colon **":"** in the place of the column specifier. This is intentional; it is a little trick to **specify the whole range**. We could have used the command **d(4,1:2)=[7 8]** to say the same thing, but the first version is easier (and less typing).

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

1.5.3.2 How to delete a Column from a matrix:

You can delete rows and columns from a matrix using just a pair of square brackets. Start with

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12;4 15 14 1]
>> X = A;
```

Then, to delete the second column of X, use

```
>> X(:,2) = []
```

This changes X to

```
X =
   16    2   13
    5   11    8
    9    7   12
    4   14    1
```

1.5.3.3 How to select a specific row or column in a matrix:

What if we want to select column 1 only? The following code does this.

```
>> d(:,1)
ans =
        1
        3
        5
        7
```

Compare the previous output to the following command.

```
>> d(1,:)
ans =
        1        2
```

The last two examples show how to access column 1 and row 1, respectively.

1.5.4 Size and Length Commands:

Now let's have a little taste about the size of vector.

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
        1        2        3
        4        5        6
        7        8        9
>> b = [0;1;0]
b =
        0
        1
        0
>> size(A)
ans =
        3        3
>> size(b)
ans =
        3        1
>>[row,col] = size(A)
row =
        3
col =
        2
>>[row,column]=size(b)
row =
    3
column =
    1
>>length (A)
ans =
        3
>>length (b)
ans =
```

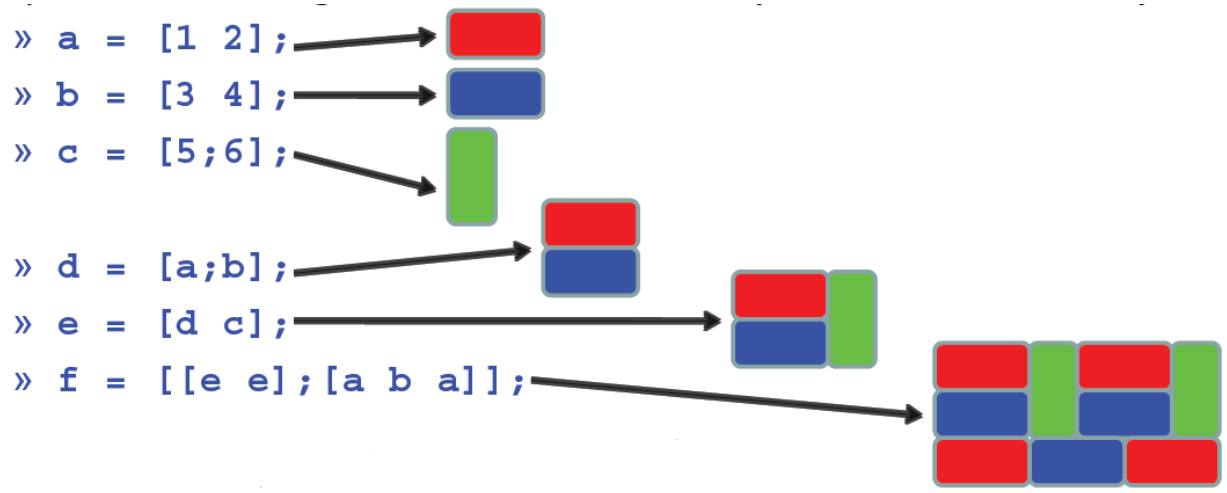
```
     3
```

The command **length(A)** returns the **number of elements of A**, if A is a **vector**, or the **largest value** of either n or m, if it is an **n × m matrix**. The MATLAB command **size(A)** returns the **size of the matrix A** (number of rows by the number of column), and the command **[row, col] = size(A)** returns the number of rows assigned to the variable **row** and the number of columns of A assigned to the variable **col.**

1.5.5 Concatenation:

Arrays can be built out of other arrays, as long as the **sizes are compatible**:

```
>> [A b]
ans =
     1     2     3     0
     4     5     6     1
     7     8     9     0
>> [A;b]
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.
>> B = [[1 2;3 4] [5;6]]
B =
     1     2     5
     3     4     6
```



1.6 Number Ranges

A variable can hold a range of numbers. For example, imagine that variable N should hold all numbers 1, 2, 3, 4, etc., up to the number 100. Here is one way to create N:

```
for index=1:100
N(index)=index;
end
```

1.6.1 The Colon Operator:

This does the job, but there is a faster and simpler way to say the same thing. Below, we set N equal to a range, specified with a colon **":"** symbol between the starting and ending values.

N=Start: Increment/Decrement: End

N=1:100;

If a third number is specified, then the second one will be used as the increment. For example, **M=0:0.4:2** would assign the sequence **{0, 0.4, 0.8, 1.2, 1.6, 2}** to variable **M**. The following example shows all numbers between 8 and 20, with an increment of 2.

```
>> 8:2:20
ans =
     8    10    12    14    16    18    20
```

1.6.2 The linspace Command:

The **linspace** command do more or less the same thing as done by colon operator. This command provides the user with more flexibility in such a way that we can get the exact number of elements in a row vector we want.

```
>> linspace(1,2,9)
```

In the above statement, a row vector would be formed such that it contains 9 entries equally spaced between 1 and 2. Now, in certain case if number of entries of the vector is not defined, the command will automatically generate a row vector which will contain 100 entries equally spaced between 1 and 2.

```
>> linspace(1,2)
```

1.7 Referencing Elements

It is frequently necessary to access one or more of the elements of a matrix. Each dimension is given a single index or vector of indices. The result is a block extracted from the matrix. The colon is often a useful way to construct array indices. Here are some examples, consider the following matrix

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> b = [0;1;0]
b =
     0
     1
     0
>> A(2,3) % single element
ans =
     6
>> b(2) % one index for a vector
ans =
     1
>> b([1 3]) % multiple elements
ans =
```

```
        0
        0
>> A(1:2,2:3) % a submatrix
ans =
        2        3
        5        6
>> B(1,2:end) % first row, all but first column
ans =
        2        5
>> A(1:end-1,end) % all but last row, last column
ans =
        3
        6


>> B(:,3) % all rows of column 3
ans =
        5
        6
>> b(:,[1 1 1 1]) % multiple copies of a column
ans =
        0        0        0        0
        1        1        1        1
        0        0        0        0
```

1.8 Arithmetic Operations

1.8.1 Scalar Mathematics:

Scalar mathematics involves operations on **single-valued** variables. MATLAB provides support for scalar mathematics similar to that provided by a calculator.

1.8.1.1 Numbers:

MATLAB represents numbers in two form, fixed point and floating point.

Fixed point: Decimal form, with an optional decimal point. For example:2.6349, -381, 0.00023

Floating point: Scientific notation, representing m × 10e. For example: 2.6349 × 105 is represented as 2.6349e5. It is called floating point because the decimal point is allowed to move. The number has two parts:
• mantissa m: fixed point number (signed or unsigned), with an optional decimal point (2.6349 in the example above)
• exponent e: an integer exponent (signed or unsigned) (5 in the example).
• Mantissa and exponent must be separated by the letter e (or E).
Scientific notation is used when the numbers are very small or very large. For example, it is easier to represent 0.000000001 as 1e-9.

1.8.1.2 Operators

The evaluation of expressions is achieved with arithmetic operators, shown in the table below.

Table 1: Mathematical Operators in MATLAB

Operation	Algebraic Form	MATLAB	Expression
Addition	a+b	a+b	5+3
Multiplication	Axb	a*b	23-12
Subtraction	a-b	a-b	3.14*0.85
Right Division	a/b	a/b	56/8
Left Division	b/a	a\b	8/56
Exponentiation	a^b	a^b	2^3

Operators operate on operands (a and b in the table).

Examples of expressions constructed from numbers and operators, processed by MATLAB:

```
>> 3 + 4
ans =
        7
>> 3 - 3
ans =
        0
>> 4/5
ans =
```

Operations may be chained together. For example:

```
>> 3 + 5 + 2
ans =
        10
>> 4*22 + 6*48 + 2*82
ans =
        540
>> 4 * 4 * 4
ans =
        64
```

Instead of repeating the multiplication, the MATLAB exponentiation or power operator can be used to write the last expression above as

```
>> 4^3
ans =
        64
```

Left division may seem strange: divide the right operand by the left operand. For scalar operands the expressions 56/8 and 8\56 produce the same numerical result.

```
>> 56/8
ans =
        7
>> 8\56
ans =
        7
```

We will later learn that matrix left division has an entirely different meaning.

1.8.1.3 Syntax:

MATLAB cannot make sense out of just any command; commands must be written using the correct syntax (rules for forming commands). Compare the interaction above with

```
>> 4 + 6 +
??? 4 + 6 +
|
Missing operator, comma, or semi-colon.
```

Here, MATLAB is indicating that we have made a syntax error, which is comparable to a misspelled word or a grammatical mistake in English. Instead of answering our question, MATLAB tells us that we've made a mistake and tries its best to tell us what the error is.

1.8.1.4 Precedence of operations (order of evaluation)

Since several operations can be combined in one expression, there are rules about the order in which these operations are performed:

1. Parentheses, innermost first
2. Exponentiation (^), left to right
3. Multiplication (*) and division (/ or \) with equal precedence, left to right
4. Addition (+) and subtraction (−) with equal precedence, left to right

When operators in an expression have the same precedence the operations are carried out from left to right. Thus 3 / 4 * 5 is evaluated as (3 / 4) * 5 and not as 3 / (4 * 5).

1.8.1.5 Variables and Assignment Statements:

Variable names can be assigned to represent numerical values in MATLAB. The rules for these variable names are:

- Must start with a letter
- May consist only of the letters a-z, digits 0-9, and the underscore character (_)
- May be as long as you would like, but MATLAB only recognizes the first 31 characters
- Is case sensitive: items, Items, itEms, and ITEMS are all different variable names.

Assignment statement: MATLAB command of the form:

- variable = number
- variable = expression

When a command of this form is executed, the expression is evaluated, producing a number that is assigned to the variable. The variable name and its value are displayed. If a variable name is not specified, MATLAB will assign the result to the default variable, ans, as shown in previous examples.

```
>> screws = 32
screws =
                32
>> bolts = 18
bolts =
                18
>> rivets = 40;
>> items = screws + bolts + rivets
items =
        90
>> cost = screws * 0.12 + bolts * 0.18 + rivets * 0.08
cost =
```

10.2800

• Variables: screws, bolts, rivets, items, cost
• Results displayed and stored by variable name
• Semicolon at the end of a line (as in the line >> rivets=40;) tells MATLAB to evaluate the line but not to display the results

1.8.2 Vector and Matrix Arithmetic

First thing to learn in arithmetic operations is to learn how to do element wise operations. For element wise operation you will have to use **.,.*, ./, .^.** In order to use these operators the dimensions of both the vectors should be same.

>> a=[1 2 3];b=[4;2;1];
>> a.*b, a./b, a.^b -> all errors

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}.*\begin{bmatrix}4\\2\\1\end{bmatrix} = ERROR$$

$$\begin{bmatrix}1\\2\\3\end{bmatrix}.*\begin{bmatrix}4\\2\\1\end{bmatrix}=\begin{bmatrix}4\\4\\3\end{bmatrix}$$

$$3\times1.*3\times1=3\times1$$

>> a.*b', a./b', a.^(b') all valid

$$\begin{bmatrix}1 & 1 & 1\\2 & 2 & 2\\3 & 3 & 3\end{bmatrix}.*\begin{bmatrix}1 & 2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{bmatrix}=\begin{bmatrix}1 & 2 & 3\\2 & 4 & 6\\3 & 6 & 9\end{bmatrix}$$

$$3\times3.*3\times3=3\times3$$

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}.^{\wedge}2=\begin{bmatrix}1^2 & 2^2\\3^2 & 4^2\end{bmatrix}$$

Can be any dimension

Here are some examples of common arithmetic operations in MATLAB. First, we will define some variables to work with.

```
>> A = [20 1 9 16]
A =
        20      1       9       16
>> A2 = [20, 1, 9, 16] % Commas can separate entries
A2 =
        20      1       9       16
```

As we see above, adding commas after each element also works, and is a bit clearer.

```
>> B = [5 4 3 2]
B =
        5       4       3       2
>> C = [0 1 2 3; 4 5 6 7; 8 9 10 11]
C =
        0       1       2       3
        4       5       6       7
        8       9       10      11
>> D = [8; 12; 0; 6] % Semicolons separate rows
D =
        8
        12
        0
        6
>> F = [31 30 29 28; 27 26 25 24; 23 22 21 20]
F =
        31      30      29      28
        27      26      25      24
        23      22      21      20
```

Semicolons can be used to separate rows of a matrix, as seen above, where all of the operations are assignment statements. Most of the examples below are not assignment statements; that is, the results are just printed to the screen and do not affect the above matrices. A variable followed by a period then the single-quote character indicates the transpose of that variable. The notation: **A.'** simply represents the transpose of the array A. We can obtain the same effect by using **transpose(A)** instead. The example below shows the transpose of A, that is, views it as a 4 by 1 matrix instead of a 1 by 4 matrix. We can verify this with the size function. On a matrix like C, the result is to swap the rows with the columns. Similarly, we see that C is a 3 by 4 matrix, while the transpose of C is 4 by 3.

```
>> A.' % transpose A
ans =
        20
        1
        9
        16
>> size(A.')
ans =
```

```
        4       1
>> size(A)
ans =
        1       4
>> C.' % transpose C
ans =
        0       4       8
        1       5       9
        2       6       10
        3       7       11
>> size(C.')
ans =
        4       3
>> size(C)
ans =
        3       4
```

Addition and subtraction can be carried out as expected. Below, we see that a constant can be added to every element of a matrix, e.g., **A+7**. We can add every element of two matrices together, as in **A+B**. Subtraction is easily done with **A-B** and **B-A**.

```
>> A+7
ans =
        27      8       16      23
>> A+B      % Note that A and B must be the same size.
ans =
        25      5       12      18
>> A-B
ans =
        15      -3      6       14
>> B-A
ans =
        -15     3       -6      -14
```

To find the summation of an array, use the sum function. It also works with two dimensional matrices, though it will add all of the numbers along the columns and return an array. Notice below that there are two ways of expressing the same thing. The command **sum(A+B)** says to add matrix **A** to matrix **B**, then find the sum of the result. The second way to do this, **sum(A)+sum(B)** says to find the sum of **A** and the sum of **B**, then add the two sums together. As in any programming language, there are often multiple ways to do the same thing. This becomes important when performance is poor; it may be possible to restate the operations in a more efficient way.

```
>> sum(A)
ans =
        46
>> sum(B)
ans =
```

```
        14
>> % Add A to B, then find their sum
>> sum(A+B)
ans =
        60
>> % find sum of A and sum of B, then add
>> sum(A) + sum(B)
ans =
        60
```

The next examples are of multiplication operations. Multiplication can be a bit tricky, since there are several similar operations. First, let us start by trying to multiply **A** with **B**.

```
>> A*B
??? Error using ==> *
Inner matrix dimensions must agree.
```

We cannot multiply a 4X1 matrix with a 4 x1 matrix! If we want to multiply two matrices together, we must make sure that the first matrix has as many columns as the second matrix has rows. In this case, we must transpose either **A** or **B** in order to multiply it by the other. Note that when we multiply **A** by the transpose of **B**, we are multiplying the nth elements together, and adding the results.

```
>> A.'*B        % Multiply transpose(A) by B
ans =
        100     80      60      40
        5       4       3       2
        45      36      27      18
        80      64      48      32
>> A*B.'        % Multiply A by transpose(B).
ans =
        163
>> A(1)*B(1) + A(2)*B(2) + A(3)*B(3) + A(4)*B(4)
ans =
        163
```

What if we want to multiply the nth elements together, but not add the results? This can be accomplished by putting a period before the multiplication sign, to say that we want to multiply the individual elements together. Notice that now we do not have to transpose either array.

```
>> A.*B         % ans = [A(1)*B(1) A(2)*B(2) A(3)*B(3) A(4)*B(4)]
ans =
        100     4       27      32
```

We can multiply an array by a scalar value, as seen below. Also, we can take the results of such an operation and add them together with the sum function.

```
>> A*4          % multiply all values of A by 4
```

```
ans =
        80      4       36      64
>> sum(A*4)
ans =
        184
```

When it comes to multiplying matrices with arrays, the results are as one would expect from linear algebra.

```
>> [1 2 3]*C   % Multiply row vector [1 2 3] by C
% result(1) = 0*1 + 4*2 + 8*3
% result(2) = 1*1 + 5*2 + 9*3
% result(3) = 2*1 + 6*2 + 10*3
% result(4) = 3*1 + 7*2 + 11*3
% Results are all in 1 row.
ans =
        32      38      44      50
>> C*[1 2 3 4] % Multiply C by [1 2 3 4] as a column vector
% result(1) = 0*1 + 1*2 + 2*3 + 3*4
% result(2) = 4*1 + 5*2 + 6*3 + 7*4
% result(3) = 8*1 + 9*2 + 10*3 + 11*4
% Results are all in 1 column.
ans =
        20
        60
        100
```

There are several different ways to multiply **F** and **C.** Trying **F*C** does not work, because the dimensions are not correct. If **F** or **C** is transposed, then there is no problem. The numbers from these two matrices are shown multiplied and added in the variable result, showing that the multiplication result is what we should expect. We use the three dots (ellipsis) after result below to continue on the next line; this makes the lines fit.

```
>> F'.*C      % transpose(F) * C
ans =
        292     373     454     535
        280     358     436     514
        268     343     418     493
        256     328     400     472
>> result = ...
[31*0+27*4+23*8, 31*1+27*5+23*9, 31*2+27*6+23*10, 31*3+27*7+23*11;
30*0+26*4+22*8, 30*1+26*5+22*9, 30*2+26*6+22*10, 30*3+26*7+22*11;
29*0+25*4+21*8, 29*1+25*5+21*9, 29*2+25*6+21*10, 29*3+25*7+21*11;
28*0+24*4+20*8, 28*1+24*5+20*9, 28*2+24*6+20*10, 28*3+24*7+20*11]
result =
        292     373     454     535
        280     358     436     514
        268     343     418     493
        256     328     400     472
```

```
>> F*C.'
ans =
        172     644     1116
        148     556     964
        124     468     812
>> result = ...
[31*0+30*1+29*2+28*3, 31*4+30*5+29*6+28*7, 31*8+30*9+29*10+28*11;
27*0+26*1+25*2+24*3, 27*4+26*5+25*6+24*7, 27*8+26*9+25*10+24*11;
23*0+22*1+21*2+20*3, 23*4+22*5+21*6+20*7, 23*8+22*9+21*10+20*11]
result =
        172     644     1116
        148     556     964
        124     468     812
```

We can change the order of the matrices, and find that we get the same results as above, except transposed. An easy way to verify this is to subtract one (transposed) result from the other, to see that all subtraction results are zero.

```
>> C'.*F % gives same answer as F'.*C, except transposed.
ans =
        292     280     268     256
        373     358     343     328
        454     436     418     400
        535     514     493     472
>> (F'.*C).- C'.*F
ans =
        0       0       0       0
        0       0       0       0
        0       0       0       0
        0       0       0       0
>> C.*F' % gives same answer as F.*C', except transposed.
ans =
        172     148     124
        644     556     468
        1116    964     812
>> (F.*C').- C.*F'
ans =
        0       0       0
        0       0       0
        0       0       0
```

Suppose we want to multiply every element of **C** with the corresponding element of **F**. We can do this with the **.*** operator, as we did before. Incidentally, reversing the order of operands produces the same result.

```
>> C.*F
ans =
```

```
        0      30      58      84
        108 130        150     168
        184 198        210     220
>> result = [0*31, 1*30, 2*29, 3*28;
4*27, 5*26, 6*25, 7*24;
8*23, 9*22, 10*21, 11*20]
result =
        0      30      58      84
        108    130     150     168
        184    198     210     220
```

Like multiplication, division has a couple of different interpretations, depending on whether **./** or **/** is specified. For example, **A./B** specifies that elements of **A** should be divided by corresponding elements of **B**.

```
>> A = [20 1 9 16];
>> B = [5 4 3 2];
>> A./B
ans =
        4.0000          0.2500          3.0000          8.0000
>> result = [ 20/5, 1/4, 9/3, 16/2 ]
result =
        4.0000          0.2500          3.0000          8.0000
```

We can specify that elements of **B** should be divided by corresponding elements of **A**, in two different ways: using the backslash **A.\B** or the forward slash **A./B.** Referring to the values of result from above, we can divide the value 1 by each element. Also shown below is that we can divide each element of matrix **C** by the value **5**.

```
>> A.\B
ans =
        0.2500          4.0000          0.3333          0.1250
>> B./A
ans =
        0.2500          4.0000          0.3333          0.1250
>> 1./result
ans =
        0.2500          4.0000          0.3333          0.1250
>> C/5
ans =
        0               0.2000          0.4000          0.6000
        0.8000          1.0000          1.2000          1.4000
        1.6000          1.8000          2.0000          2.2000
```

Next, we divide the elements of matrix **C** by the corresponding elements of **F.** The very first result is **0**. If we switched the two matrices, i.e., using **F**'s elements for nominators (the top part of the fractions) while **C**'s elements

are used for denominators (the bottom part of the fractions), it would still work, but the division by zero would cause the first result to be **Inf** (infinity).

```
>> C./F
ans =
         0         0.0333    0.0690    0.1071
    0.1481         0.1923    0.2400    0.2917
    0.3478         0.4091    0.4762    0.5500
>> result = [ 0/31, 1/30, 2/29, 3/28;
4/27, 5/26, 6/25, 7/24;
8/23, 9/22, 10/21, 11/20]
result =
         0         0.0333    0.0690    0.1071
    0.1481         0.1923    0.2400    0.2917
    0.3478         0.4091    0.4762    0.5500
```

Finally, we can divide one matrix by another. Here is an example, which uses "nice" values for the second matrix. The inv function returns the matrix's inverse. We expect the result of matrix division to be the same as multiplication of the first matrix by the inverse of the second. In other words, $M/N = M N^{-1}$, just as it works with scalar values. The code below verifies that our results are as we expect them.

```
M = [1 2 3; 4 5 6; 7 8 9]
M =
     1     2     3
     4     5     6
     7     8     9
>> N = [1 2 0; 3 0 4; 0 5 6]
N =
     1     2     0
     3     0     4
     0     5     6
>> inv(N)
ans =
    0.3571    0.2143   -0.1429
    0.3214   -0.1071    0.0714
   -0.2679    0.0893    0.1071
>> M/N        % This is the same as M*inv(N)
ans =
    0.1964    0.2679    0.3214
    1.4286    0.8571    0.4286
    2.6607    1.4464    0.5357
>> M*inv(N)
ans =
    0.1964    0.2679    0.3214
    1.4286    0.8571    0.4286
    2.6607    1.4464    0.5357
```

There are many other operators in MATLAB. For a list, type help , that is the word help followed by a space then a period.
Let us have a look at the following table which in fact summarizes the arithmetic operations in MATLAB.

Table 1.2: Matlab Operations

Symbol	Operation	Example	Answer
+	Addition	z=4+2	z=6
-	Subtraction	z=4-2	z=2
/	Right Division	z=4/2	z=2
\	Left Division	z=2\4	z=2
*	Multiplication	z=4*2	z=8
^	Exponentiation	z=4^2	z=16
Functions such as: sqrt, log	Square root, log2	z=sqrt(4) z=log2(4)	z=2 z=2

Then the hierarchy of operations is as follows:
i. Functions such as sqrt(x), log(x), and exp(x)
ii. Exponentiation (^)
iii. Products and division (*, /)
iv. Addition and subtraction (+, −)

1.9 Some other basic commands

- **Empty Vector**

```
>> Y = []
Y =
    []
```

- **Zeros Matrix**

```
>> M = zeros(2,3) % 1st parameter is row, 2nd parameter is col.
M =
     0     0     0
     0     0     0
```

- **Ones Matrix**

```
>> m = ones(2,3)
m =
     1     1     1
     1     1     1
```

- **Identity Matrix**

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

- **Random Number Generation**

```
>> R = rand(1,3)
```

```
R =
    0.9501    0.2311    0.6068
```

- **Round**

```
>> round([1.5 2; 2.2 3.1]) % round command rounds to nearest 1.
ans =
     2     2
     2     3
```

- **Mean**

Suppose a is a vector

```
>> a=[1 4 6 3]
a =
    1    4    6    3
>> mean(a)
ans =
    3.5000
```

- **Standard Deviation**

```
>> std(a)
ans =
    2.0817
```

- **Maximum**

```
>> max(a)
ans =
     6
```

Task 01: Are the following true or false? Assume A is a generic n×n matrix. Please provide a proper reasoning for your answer.

(a) **Aˆ(-1)** equals **1/A**

(b) **A.ˆ(-1**) equals **1./A**

a)

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

A =

     1     2     3
     4     5     6
     7     8     9

>> A^(-1)
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.202823e-18.

ans =

   1.0e+16 *

    0.3153   -0.6305    0.3153
   -0.6305    1.2610   -0.6305
    0.3153   -0.6305    0.3153

>> 1/A
Error using  /
Matrix dimensions must agree.
```

b)

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

A =

     1     2     3
     4     5     6
     7     8     9

>> A.^(-1)

ans =

    1.0000    0.5000    0.3333
    0.2500    0.2000    0.1667
    0.1429    0.1250    0.1111

>> 1./A

ans =

    1.0000    0.5000    0.3333
    0.2500    0.2000    0.1667
    0.1429    0.1250    0.1111
```

Task 02: Vector Generation

(a) Generate the following vectors:

A = [1 0 4 5 3 9 0 2]

a = [4 5 0 2 0 0 7 1]

Note: Be aware that Matlab are case sensitive. Vector A and a have different values.

(b) Generate the following vectors:

B = [A a]

C = [a, A]

(c) Generate the following vectors using function zeros and ones:
D = [0 0 0 . . . 0] with fifty 0's.
E = [1 1 1 . . . 1] with a hundred 1's.

(d) Generate the following vectors using the colon operator
F = [1 2 3 4 . . . 30]
G = [25 22 19 16 13 10 7 4 1]
H = [0 0.2 0.4 0.6 . . . 2.0]

a)

```
>> A=[1 0 4 5 3 9 0 2]

A =

     1     0     4     5     3     9     0     2

>> a=[ 4 5 0 2 0 0 7 1]

a =

     4     5     0     2     0     0     7     1
```

b)

```
>> B=[A a]

B =

     1     0     4     5     3     9     0     2     4     5     0     2     0     0     7     1

>> C=[a,A]

C =

     4     5     0     2     0     0     7     1     1     0     4     5     3     9     0     2
```

d)

```
>> D= zeros(1,50)

D =

  Columns 1 through 20

     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 21 through 40

     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 41 through 50

     0     0     0     0     0     0     0     0     0     0
```

e)

```
>> E= ones(1,100)

E =

  Columns 1 through 20

     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1

  Columns 21 through 40

     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1

  Columns 41 through 60

     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1

  Columns 61 through 80

     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1

  Columns 81 through 100

     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1
```

f)

```
>> F=[1:30]

F =

  Columns 1 through 20

     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17    18    19    20

  Columns 21 through 30

    21    22    23    24    25    26    27    28    29    30
```

g)

```
>> G=[25:-3:1]

G =

    25    22    19    16    13    10     7     4     1
```

H)

```
>> H=[0:0.2:2]

H =

         0    0.2000    0.4000    0.6000    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
```

In-lab Tasks

Task 03: Operate with the vectors
V1 = [1 2 3 4 5 6 7 8 9 0]
V2 = [0.3 1.2 0.5 2.1 0.1 0.4 3.6 4.2 1.7 0.9]
V3 = [4 4 4 4 3 3 2 2 2 1]
(a) Calculate, respectively, the sum of all the elements in vectors V1, V2, and V3.

(b) How to get the value of the fifth element of each vector? What happens if we execute the command V1(0) and V1(11)? Remember if a vector has N elements, their subscripts are from 1 to N.

(c) Generate a new vector V4 from V2, which is composed of the first five elements of V2. Generate a new vector V5 from V2, which is composed of the last five elements of V2.

(d) Derive a new vector V6 from V2, with its 6th element omitted. Derive a new vector V7 from V2, with its 7th element changed to 1.4. Derive a new vector V8 from V2, whose elements are the 1st, 3rd, 5th, 7th, and 9th elements of V2.

(e) What are the results of

9-V1, V1*5, V1+V2, V1-V3, V1.*V2, V1*V2, V1.^2, V1.^V3, V1^V3

a)

```
>> V1

V1 =

     1     2     3     4     5     6     7     8     9     0

>> sum(V1)

ans =

    45
```

b)

```
>> V2

V2 =

    0.3000    1.2000    0.5000    2.1000    0.1000    0.4000    3.6000    4.2000    1.7000    0.9000

>> sum(V2)

ans =

   15.0000

>> V3

V3 =

     4     4     4     4     3     3     2     2     2     1

>> sum(V3)

ans =

    29
```

```
>> V1(5)

ans =

     5

>> V2(5)

ans =

    0.1000

>> V3(5)

ans =

     3
```

```
>> V4=V2(1:5)

V4 =

    0.3000    1.2000    0.5000    2.1000    0.1000

>> V5=V2(6:10)

V5 =

    0.4000    3.6000    4.2000    1.7000    0.9000
```

c)

```
>> V6=V2([1:5,7:end])

V6 =

    0.3000    1.2000    0.5000    2.1000    0.1000    3.6000    4.2000    1.7000    0.9000
```

d)

```
>> V7=V2

V7 =

    0.3000    1.2000    0.5000    2.1000    0.1000    0.4000    3.6000    4.2000    1.7000    0.9000

>> V7(7)=1.4

V7 =

    0.3000    1.2000    0.5000    2.1000    0.1000    0.4000    1.4000    4.2000    1.7000    0.9000

>> V8=V2

V8 =

    0.3000    1.2000    0.5000    2.1000    0.1000    0.4000    3.6000    4.2000    1.7000    0.9000

>> V8(10)=[]

V8 =

    0.3000    1.2000    0.5000    2.1000    0.1000    0.4000    3.6000    4.2000    1.7000

>> V8(8)=[]

V8 =

    0.3000    1.2000    0.5000    2.1000    0.1000    0.4000    3.6000    1.7000

>> V8(6)=[]

V8 =

    0.3000    1.2000    0.5000    2.1000    0.1000    3.6000    1.7000

>> V8(4)=[]

V8 =

    0.3000    1.2000    0.5000    0.1000    3.6000    1.7000

>> V8(2)=[]

V8 =

    0.3000    0.5000    0.1000    3.6000    1.7000
```

e)

```
>> 9-V1

ans =

     8     7     6     5     4     3     2     1     0     9

>> V1*5

ans =

     5    10    15    20    25    30    35    40    45     0

>> V1+V2

ans =

    1.3000    3.2000    3.5000    6.1000    5.1000    6.4000   10.6000   12.2000   10.7000    0.9000

>> V1-V3

ans =

    -3    -2    -1     0     2     3     5     6     7    -1

>> V1.*V2

ans =

    0.3000    2.4000    1.5000    8.4000    0.5000    2.4000   25.2000   33.6000   15.3000         0

>> V1*V2
Error using  *
Incorrect dimensions for matrix multiplication. Check that the number of columns in the first matrix matches
of rows in the second matrix. To perform elementwise multiplication, use '.*'.

>> V1.^2

ans =

     1     4     9    16    25    36    49    64    81     0

>> V1.^V3

ans =

     1    16    81   256   125   216    49    64    81     0

>> V1^V3
Error using  ^
Incorrect dimensions for raising a matrix to a power. Check that the matrix is square and the power is a scal
perform elementwise matrix powers, use '.^'.
```

Task 04: Suppose p is a row vector such that p=[4 2 3 1]. What does this line do? Please provide a detailed answer stepwise
[length(p)-1:-1:0] .* p

```
>> p

p =

     4     2     3     1

>> [length(p)-1:-1:0].*p

ans =

    12     4     3     0
```

Task 05: Suppose A is any matrix. What does this statement do? Please provide a reasonable reason.
A(1:size(A,1)+1:end)

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

A =

     1     2
     3     4

>> A(1: size(A,1)+1: end)

ans =

     1     4

>> A=[ 1 2 ;3 4]

A =

     1     2
     3     4

>> A(1: size(A,1)+1: end)

ans =

     1     4

>> size(A,1)

ans =

     2

>> size (A)

ans =

     2     2
```

Task 06: Try to avoid using unnecessary brackets in an expression. Can you spot the errors in the following expression? (Test your corrected version with MATLAB.)
(2(3+4)/(5*(6+1))^2

```
>> (2*(3+4)/(5*(6+1))^2)

ans =

    0.0114
```

Task 07: Set up a vector n with elements 1, 2, 3, 4, 5. Use MATLAB array operations on it to set up the following four vectors, each with five elements:
(a) 2, 4, 6, 8, 10
(b) 1/2, 1, 3/2, 2, 5/2
(c) 1, 1/2, 1/3, 1/4, 1/5

```
n =

     1     2     3     4     5

>> 2*n

ans =

     2     4     6     8    10
```

a)

```
>> n/2

ans =

    0.5000    1.0000    1.5000    2.0000    2.5000
```

b)

```
>> 1./n

ans =

    1.0000    0.5000    0.3333    0.2500    0.2000
```

c)

Task 08: Suppose vectors a and b are defined as follows:
a = [2 –1 5 0];
b = [3 2 –1 4];
Evaluate by hand the vector c in the following statements. Check your answers with MATLAB.
(a) c = a – b;
(b) c = b + a – 3;
(c) c = 2 * a + a .^ b;
(d) c = b ./ a;
(e) c = b . a;
(f) c = a .^ b;
(g) c = 2.^b+a;
(h) c = 2*b/3.*a;
(i) c = b*2.*a;

```
>> a=[2 -1 5 0]

a =

     2    -1     5     0

>> b=[ 3 2 -1 4]

b =

     3     2    -1     4

>> a-b

ans =

    -1    -3     6    -4

>> b+a-3

ans =

     2    -2     1     1

>> 2*a+a.^b

ans =

   12.0000   -1.0000   10.2000         0

>> b./a

ans =

    1.5000   -2.0000   -0.2000       Inf

>> b.a
Dot indexing is not supported for variables of this type.
```

```
>> a.^b

ans =

    8.0000    1.0000    0.2000         0

>> 2.^b+a

ans =

   10.0000    3.0000    5.5000   16.0000

>> 2*b/3.*a

ans =

    4.0000   -1.3333   -3.3333         0

>> b*2.*a

ans =

    12    -4   -10     0
```

Task 09: Make a vector v=[1 2 3 4 5 6 7 8 9 10], develop an algorithm such that the first element of the vector is multiplied by length(v), second element by length(v)-1and similarly the last element i.e. 10 is multiplied by length(v)-9. The final vector should be f=[10 18 24 28 30 30 28 24 18 10]. The algorithm devised should only use the length of vector v to achieve vector f.

```
>> Vl=[length(v): -1 :1] .*v

Vl =

    10    18    24    28    30    30    28    24    18    10
```

Task 10: (a) Make a matrix M1 which consists of two rows and three columns and all the entries in the matrix are ones.
(b) Make a vector V1 consisting of three ones.
(c) Make a 3x3 matrix M2 in which the diagonal entries are all fives.
(d) Now make a matrix M3 from M1, M2 and V1 which look like the matrix given below

$$M3=\begin{bmatrix}1&1&1&5&0&0\\1&1&1&0&5&0\\0&0&0&0&0&5\end{bmatrix}$$

(e) Now use the referencing element concept to make three vectors V2, V3 and V4 such that V2 consists of first row of M3, V3 consists of second row of M3 and V4 consists of third row of M3.
(f) Now alter the fourth entry of vectors V2, fifth entry of V3 and sixth entry of V4 to 1.4 and make a new vector M4 which looks like the matrix given below.

$$M4 = \begin{bmatrix} 1 & 1 & 1 & 1.4 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1.4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.4 \end{bmatrix}$$

a)

```
>> M1= ones(2,3)

M1 =

     1     1     1
     1     1     1
```

b)

```
>> V1= ones(1,3)

V1 =

     1     1     1
```

```
>> M2=eye(3)

M2 =

     1     0     0
     0     1     0
     0     0     1

>> M2*5

ans =

     5     0     0
     0     5     0
     0     0     5
```

c)

```
>> A=[M1;V1]

A =

     1     1     1
     1     1     1
     1     1     1

>> B=5*M2

B =

     5     0     0
     0     5     0
     0     0     5

>> M3=[A B]

M3 =

     1     1     1     5     0     0
     1     1     1     0     5     0
     1     1     1     0     0     5
```

d)

```
>> M3(3,1)=0

M3 =

     1     1     1     5     0     0
     1     1     1     0     5     0
     0     1     1     0     0     5

>> M3(3,2)=0

M3 =

     1     1     1     5     0     0
     1     1     1     0     5     0
     0     0     1     0     0     5

>> M3(3,3)=0

M3 =

     1     1     1     5     0     0
     1     1     1     0     5     0
     0     0     0     0     0     5
```

e)

```
>> M4=[V2;V3;V4]

M4 =

    1.0000    1.0000    1.0000    1.4000         0         0
    1.0000    1.0000    1.0000         0    1.4000         0
         0         0         0         0         0    1.4000
```

f)

Post-lab Task

Critical Analysis / Conclusion

In this lab, we learnt about the basic functionality of MATLAB.

We implemented commands and observed their Outputs, furthermore we completed several tasks using these commands that we had studied.

Lab Assessment		
Pre-Lab	**/1**	**/10**
In-Lab	**/5**	
Critical Analysis	**/4**	
Instructor Signature and Comments		

LAB # 02

Introduction to MATLAB: Signal Plotting and Basic Programming

Lab 02- Introduction to MATLAB: Signal Plotting and Basic Programming

Pre-lab Tasks

2.1 Logical and Relational Operators in MATLAB

2.1.1 Relational Operators:

There are certain relational operators in MATLAB. Table 2.1 shows a list of such operators

Table 2.1: Relational Operators in MATLAB

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

Note that a single **'='** denotes **assignment** and never a test for equality in MATLAB. Comparisons between scalars produce logical 1 if the relation is true and logical 0 if it is false. Comparisons are also defined between matrices of the same dimension and between a matrix and a scalar, the result being a matrix of logical in both cases. For matrix-matrix comparisons corresponding pairs of elements are compared, while in matrix-scalar comparisons the scalar is compared with each matrix element. For example:

```
>> A = [1 2; 3 4]; B = 2*ones(2);
>> A == B
ans =
     0     1
     0     0
» A > 2
ans =
     0     0
     1     1
>> A > B
ans =
   0   0
   1   1
>> A < B
ans =
   1   0
   0   0
>> A >= B
ans =
   0   1
   1   1
```

```
>> A <= B
ans =
    1    1
    0    0
```

For Information

To test whether arrays A and B are equal, that is, of the same size with identical elements, the expression **isequal(A,B)** can be used:

```
>> isequal(A,B)
ans =
         0
```

The function **isequal** is one of many useful logical functions whose names begin with is, a selection of which is listed in Table 2.2. For example, **isinf(A)** returns a logical array of the same size as A containing true where the elements of A are plus or minus inf and false where they are not:

```
>> A = [1 inf; -inf NaN]
A =
    1   Inf
 -Inf   NaN
>> isinf(A)
ans =
          0          1
          1          0
```

The function **isnan** is particularly important because the test **x == NaN** always produces the result 0 (false), even if x is a NaN! (A NaN is defined to compare as unequal and unordered with everything.)

```
>> isnan(A)
ans =
    0    0
    0    1
>> A == NaN
ans =
    0    0
    0    0
```

Note that an array can be real in the mathematical sense, but not real as reported by **isreal**. For **isreal(A)** is true if A has no imaginary part. Mathematically, A is real if every component has zero imaginary part. How a mathematically real A is formed can determine whether it has an imaginary part or not in MATLAB. The distinction can be seen as follows:

```
>> a = 1;
>> b = complex(1,0);
>> c = 1 + Oi;
>> [a b c]
ans =

                1 1 1

>> whos a b c
  Name      Size            Bytes  Class     Attributes
```

```
  a       1x1             8  double
  b       1x1            16  double    complex
  c       1x1             8  double

>> [isreal(a) isreal(b) isreal(c)]
ans =
    1     0     1
```

Table 2.2: is functions

Command	Description
Ischar	Test for char array (string)
Isempty	Test for empty array
Isequal	Test if arrays are equal
isequalwithequalnans	Test if arrays are equal, treating NaNs as equal
Isfinite	Detect finite array elements
Isfloat	Test for floating point array (single or double)
Isinf	Detect infinite array elements
Isinteger	Test for integer array
Islogical	Test for logical array
Isnan	Detect NaN array elements
Isnumeric	Test for numeric array (integer or floating point)
Isreal	Test for real array
Isscalar	Test for scalar array
Issorted	Test for sorted vector
Isvector	Test for vector array

2.1.2 Logical Operators:

There are certain logical operators in MATLAB as well. A list of these logical operators is given in Table 2.3

Table 2.3: Logical Operators in MATLAB

Operator	Description
&	Logical AND
&&	Logical AND for scalars
|	Logical OR
||	Logical OR for scalars
~	Logical NOT
Xor	Exclusive OR
All	True if all elements of a vector are non zero
Any	True if any element of the vector is non zero

Like the relational operators, the **&**, **|**, and **~** operators produce matrices of logical 0s and 1s when one of the arguments is a matrix. When applied to a vector, the **all** function returns 1 if all the elements of the vector are nonzero and 0 otherwise. The any function is defined in the same way, with **"any"** replacing **"all"**. Examples:

```
>> x = [-1 1 1] ; y = [1 2 -3] ;
>> x > 0 & y > 0
ans =
     0     1     0
>> x > 0 | y > 0
ans =
     1     1     1
>> xor(x > 0, y > 0)
ans =
```

```
     1     0     1
>> any(x > 0)
ans =
     1

>> all(x > 0)
ans =
     0
```

Note that xor must be called as a function: xor (a,b). The and, or, and not operators and the relational operators can also be called in functional form as and(a,b) etc. The operators && and || are special in two ways. First, they work with scalar expressions only, and should be used in preference to & and | for scalar expressions.

```
>> any(x>0) && any(y>0)
ans =
     1
>>  x>0 && y>0
??? Operands to the 11 and && operators must be convertible to logical scalar values.
```

The second feature of these "double barreled" operators is that they short-circuit the evaluation of the logical expressions, where possible. In the compound expression exprl && expr2, if exprl evaluates to false then expr2 is not evaluated. Similarly, in exprl || expr2, if exprl evaluates to true then expr2 is not evaluated.

2.1.3 Precedence of Operators:

The precedence of arithmetic, relational, and logical operators is summarized in Table 2.4 (which is based on the information provided by help precedence). For operators of equal precedence MATLAB evaluates from left to right. Precedence can be overridden by using parentheses. Note, in particular, that and has higher precedence than or, so a logical expression of the form

Table 2.4: Precedence of Operators

Precedence Level	Operator
1 (highest)	Parentheses ()
2	Transpose (. ') > power (.^), complex conjugate,
3	Unary plus (+), unary minus (-), logical negation (~)
4	Multiplication (.*), right division (./), left division
5	Addition(+), Subtraction(-)
6	Colon operator (:)
7	Less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=), equal to (==), not equal to (~=)
8	Logical AND (&)
9	Logical OR (|)
10	Logical short circuit AND (&)
11(lowest)	Logical short circuit OR (|)

2.2 MATLAB M files

There are **four** ways of doing **code in MATLAB**. One can directly enter code in a **terminal window**. This amounts to using MATLAB as a kind of calculator, and it is good for simple, low-level work. The second method is to create a **script M-file**. Here, one makes a file with the same code one would enter in a terminal window. When the file is “run”,

the script is carried out. The third method is the **function M-file**. This method actually creates a function, with inputs and outputs. The fourth method will not be discussed here. It is a way to incorporate C code or FORTRAN code into MATLAB; this method uses .mex. We will not discuss it here.

2.2.1 Terminal Input

One can type the code in the MATLAB terminal window. For example, if we wish to plot $xsin(3x^2)e^{-\frac{x^2}{4}}$ between the interval $[-\pi, \pi]$, we could type the following in the terminal window.

```
>> x=-pi:pi/40:pi;
>> y=x.*sin(3*x.^2).*exp(-x.^2/4);
>> plot(x,y)
```

The code listed above creates the row vector x, and then uses it to form a row vector y whose entries are the values of the function $xsin(3x^2)e^{-\frac{x^2}{4}}$ for each entry in x. The operations preceded by a "dot" such as .* or .^ are array operations. They allow entry-by entry multiplications or powers. Without the "dot" these are matrix operations. After creating the arrays x and y, an x-y plot is made. This method of doing calculations is good for short, one-time-only calculations or for calculations where one does not wish to change any parameters.

2.2.2 Script M files:

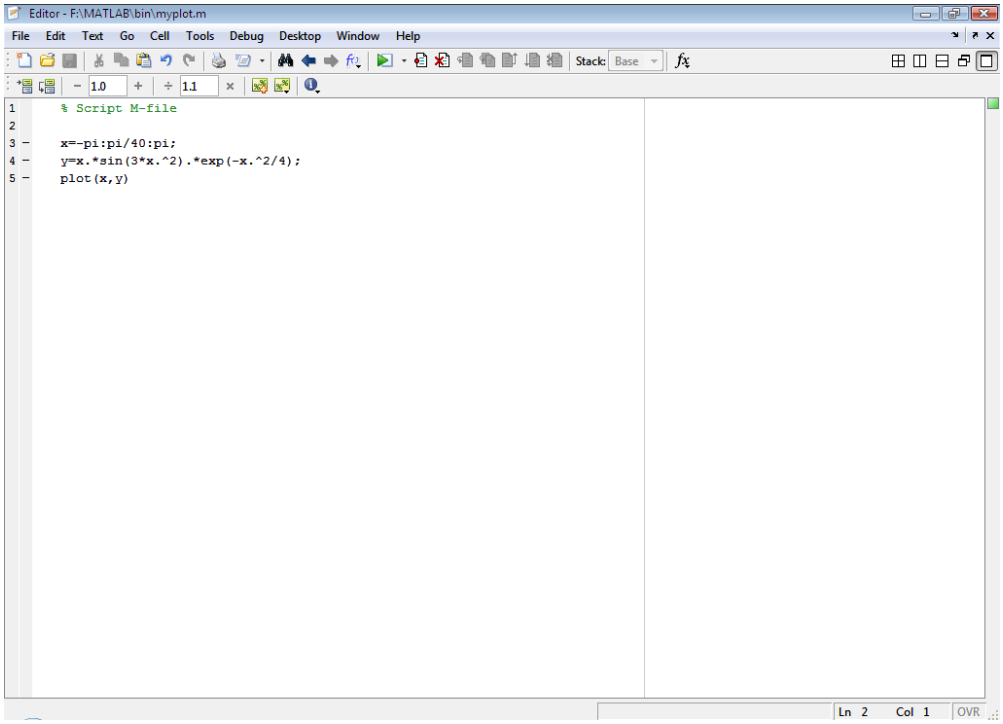
If we wish to execute repeatedly some set of commands, and possibly change input parameters as well, then one should create a script M-file. Such a file always has a ".m" extension, and consists of the same commands one would use as input to a terminal.

2.2.2.1 Create a Script M-file:

To create a new script M-file, please follow the instructions on MATLAB interface. Please note that these instructions are true for MATLAB R2011b. A new script file can also be created by **Ctrl+N** command

To create a new script file in previous versions of MATLAB, please follow the following instructions

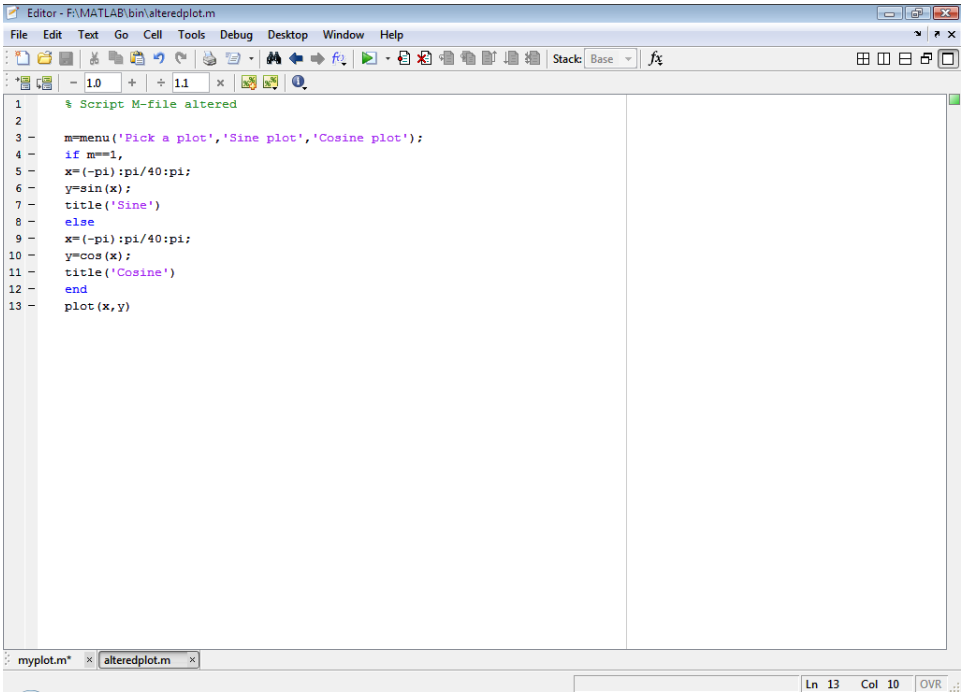
For example, to do the plot in section 1.2.1, one would create the file myplot.m:



Now in order to execute the commands, write the following commands on command (terminal) window

>> myplot

Script M-files are ideal for repeating a calculation, but with some parameters changed. They are also useful for doing demonstrations. As an exercise, create and execute the script file alteredplot.m:



Now, in order to execute the altered plot, write alteredplot on the command window. A menu bar will appear and select the appropriate choice.

2.2.3 Function M-files:

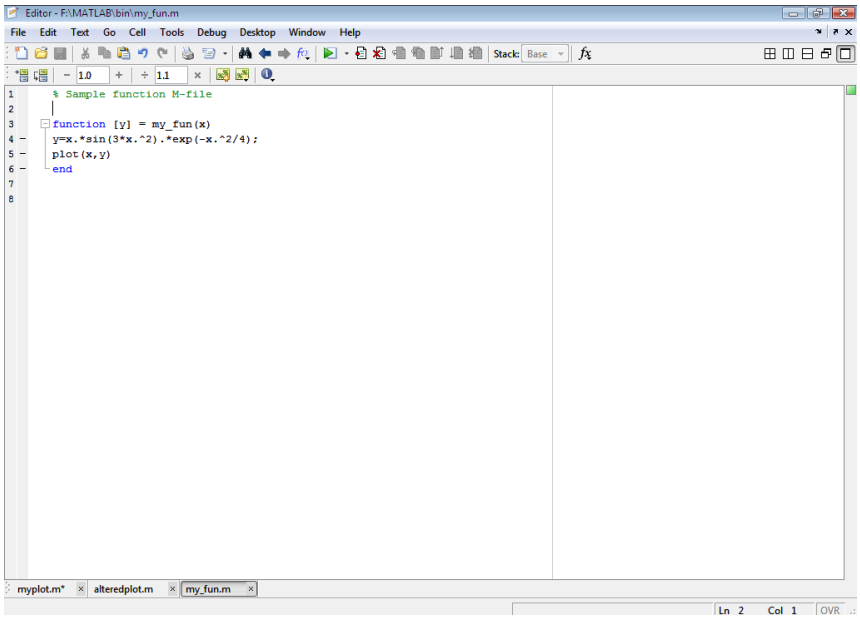
Most of the M-files that one ultimately uses will be function M-files. These files again have the ".m" extension, but they are used in a different way than scripts. Function files have input and output arguments, and behave like FORTRAN subroutines or C-functions. In order to create a function M-file in MATLAB R2011b follow the instructions

But in the previous versions of MATLAB, a function file is created as follows. The only difference is that you have to write function initialization yourself.

The structure of a typical function file, say my_fun.m, is as follows:

```
function outputs=my_fun(inputs)
                    code
                    .
                    .
                    .
                    code
                    outputs=…..
```

Note that the word function appears at the start of the file, and in lower case letters. In addition, the outputs and inputs and name of the function are listed. Let us return to the plot done in section 1.2.1. Suppose that instead of giving the vector x, we want to make it a variable. At the same time, we want to have available the data that we plotted. Here is a function file that would do this.



Now, in order to execute the function M-file, there should be some input to the function on which the function can execute. For example in the above case if we write my_fun on the command terminal an error will occur that the input arguments are not available. So, in order to execute the function input arguments must be given to the function, following command should be written on command terminal.

```
>> x=-pi:pi/40:pi;
>> my_fun(x)
```

Function files are normally used to combine functions in MATLAB to get new functions. For example, suppose that we want to have at our disposal a function that computes the inverse of the square of a matrix, and returns an error message if the matrix is close to singular or singular. Call this file inv_sq.m. One more thing, in the code below, note that A^2 is used, not A.^2. This is because we are computing the square of a matrix, not a matrix with the entries of A squared.

```
function B=inv sq(A)
if abs(det(A))< 0.0001,
error('The matrix is singular')
else
B=inv(A^2);
end
```

2.3 Loops in MATLAB

Loops are MATLAB constructs that permit us to execute a sequence of statements **more than once**. There are **two** basic forms of loop constructs **while loops** and **for loops**. The major difference in these two types of loops is in how the repetition is controlled. The code in a while loop is repeated an **indefinite number** of times until some user specified condition is satisfied. By contrast, the code in a for loop is repeated for a **specified number** of times, and the number of repetitions is known before the loops start.

2.3.1 The basic for construct

In general the most common form of the for loop (for use in a program, not on the command line) is
for index = j:k
statements
end
or
for index = j:m:k
statements
end
Note the following points carefully:

- j:k is a vector with elements j, j + 1, j + 2, . . . , k.
- j:m:k is a vector with elements j, j + m, j + 2m, . . . , such that the last element does not exceed k if m > 0 or is not less than k if m < 0.
- index must be a variable. Each time through the loop it will contain the next element of the vector j:k or j:m:k, and statements (there may be one or more) are carried out for each of these values.

2.3.2 for in a single line

If you insist on using for in a single line, here is the general form:
for index = j:k, statements, end
or
for index = j:m:k, statements, end
Note the following:

- Don't forget the commas (semicolons will also do if appropriate). If you leave them out you will get an error message.
- Again, statements can be one or more statements separated by commas or semicolons.
- If you leave out end, MATLAB will wait for you to enter it. Nothing will happen until you do so.

2.3.3 Avoid for loops by vectorizing!

There are situations where a 'for' loop is essential, as in many of the examples in this section so far. However, given the way MATLAB has been designed, for loops tend to be inefficient in terms of computing time. If you have written a 'for' loop that involves the index of the loop in an expression, it may be possible to vectorize the expression, making use of array operations where necessary, as the following examples show. Suppose you want to evaluate:

$$\sum_{n=1}^{100000} n$$

(And can't remember the formula for the sum). Here's how to do it with a for loop (run the program, which also times how long it takes):

```
t0 = clock;
s = 0;
for n = 1:100000
s = s + n;
end
etime(clock, t0)
```

The MATLAB function clock returns a six-element vector with the current date and time in the format year, month, day, hour, minute, seconds. Thus, t0 records when the calculation starts. The function etime returns the time in seconds elapsed between its two arguments, which must be vectors as returned by clock. On an Intel Core 2 Duo processor with MATLAB 2011b, it returned about 0.2070 seconds, which is the total time for this calculation. (If you have a faster PC, it should take less time.)

Now try to vectorize this calculation (before looking at the solution). Here it is:

```
t0 = clock;
n = 1:100000;
s = sum( n );
etime(clock, t0)
```

This way takes only 0.04 seconds on the same PC—more than 50 times faster!

There is a neater way of monitoring the time taken to interpret MATLAB statements: the tic and toc function. Suppose you want to evaluate:

$$\sum_{n=1}^{n=100000} \frac{1}{n^2}$$

Here's the for loop version:

```
tic
s = 0;
for n = 1:100000
s = s + 1/n^2;
end
toc
```

which takes about 0.046065 seconds on the same PC. Once again, try to vectorize the sum:

```
tic
n = 1:100000;
s = sum( 1./n.^2 );
toc
```

The same PC gives a time of about 0.05 seconds for the vectorized version—more than 100 times faster! (Of course, the computation time in these examples is small regardless of the method applied. However, learning how to improve the efficiency of computation to solve more complex scientific or engineering problems will be helpful as you develop good programming skills.

2.3.4 While Loop

Syntax for while loop is
while expression
statements
end
This loop is used when the programmer does not know the number of repetitions a priori. Here is an almost trivial problem that requires a use of this loop. Suppose that the number is divided by 2. The resulting quotient is divided by 2 again. This process is continued till the current quotient is less than or equal to 0.01. What is the largest quotient that is greater than 0.01?
To answer this question we write a few lines of code

```
q = pi;
while q > 0.01
q = q/2;
end
q =
0.0061
```

2.4 Decisions

The MATLAB function rand generates a random number in the range 0–1. Enter the following two statements at the command line:

```
r = rand
if r > 0.5 disp( 'greater indeed' ), end
```

MATLAB should only display the message greater indeed if r is in fact greater than 0.5 (check by displaying r). Repeat a few times—cut and paste from the Command History window (make sure that a new r is generated each time).As a slightly different but related exercise, enter the following logical expression on the command line: 2 > 0. Now enter the logical expression–1 > 0. MATLAB gives a value of 1 to a logical expression that is true and 0 to one that is false.

2.4.1 The one-line if statement

In the last example MATLAB has to make a decision; it must decide whether or not r is greater than 0.5. The if construct, which is fundamental to all computing languages, is the basis of such decision making. The simplest form of if in a single line is
if condition statement, end

Note the following points:

- condition is usually a logical expression (i.e., it contains a relational operator), which is either true or false. MATLAB allows you to use an arithmetic expression for condition. If the expression evaluates to 0, it is regarded as false; any other value is true. This is not generally recommended; the if statement is easier to understand (for you or a reader of your code) if condition is a logical expression.
- If condition is true, statement is executed, but if condition is false, nothing happens.
- condition may be a vector or a matrix, in which case it is true only if all of its elements are nonzero. A single zero element in a vector or matrix renders it false.

Here are more examples of logical expressions involving relational operators, with their meanings in parentheses:

b^2 < 4*a*c ($b^2 < 4ac$)
x >= 0 ($x \geq 0$)
a ~= 0 ($a \neq 0$)
b^2 == 4*a*c ($b^2 = 4ac$)
Remember to use the double equal sign (==) when testing for equality:
if x == 0 disp('x equals zero'), end

1.4.2 The if-else construct

If you enter the two lines

```
x = 2;
if x < 0 disp( 'neg' ), else disp( 'non-neg'), end
```

do you get the message non-neg? If you change the value of x to−1 and execute the if again, do you get the message neg this time?
Most banks offer differential interest rates. Suppose the rate is 9% if the amount in your savings account is less than $5000, but 12% otherwise. The Random Bank goes one step further and gives you a random amount in your account to start with! Run the following program a few times:

```
bal = 10000 * rand;
if bal < 5000
rate = 0.09;
else
rate = 0.12;
end
newbal = bal + rate * bal;
disp( 'New balance after interest compounded is:' )
format bank
disp( newbal )
```

Display the values of bal and rate each time from the command line to check that MATLAB has chosen the correct interest rate.
The basic form of if-else for use in a program file is

```
if condition
statementsA
else
statementsB
end
```

Note that

- statementsA and statementsB represent one or more statements.

- If condition is true, statementsA are executed, but if condition is false, statementsB are executed. This is essentially how you force MATLAB to choose between two alternatives.
- else is optional.

1.4.3 The one-line if-else statement

The simplest general form of if-else for use on one line is
if condition statementA, else statementB, end
Note the following:

- Commas (or semicolons) are essential between the various clauses.
- else is optional.
- end is mandatory; without it, MATLAB will wait forever.

1.4.4 elseif

Suppose the Random Bank now offers 9% interest on balances of less than $5000, 12% for balances of $5000 or more but less than $10,000, and 15% for balances of $10,000 or more. The following program calculates a customer's new balance after one year according to this scheme:

```
bal = 15000 * rand;
if bal < 5000
rate = 0.09;
elseif bal < 10000
rate = 0.12;
else
rate = 0.15;
end
newbal = bal + rate * bal;
format bank
disp( 'New balance is:' )
disp( newbal )
```

Run the program a few times, and once again display the values of bal and rate each time to convince yourself that MATLAB has chosen the correct interest rate. In general, the elseif clause is used:

```
if condition1
statementsA
elseif condition2
statementsB
elseif condition3
statementsC
...
else
statementsE
end
```

This is sometimes called an elseif ladder. It works as follows:

1. condition1 is tested. If it is true, statementsA are executed; MATLAB then moves to the next statement after end.
2. If condition1 is false, MATLAB checks condition2. If it is true, statementsB are executed, followed by the statement after end.

3. In this way, all conditions are tested until a true one is found. As soon as a true condition is found, no further elseifs are examined and MATLAB jumps off the ladder.
4. If none of the conditions is true, statementsE after else are executed.
5. Arrange the logic so that not more than one of the conditions is true.
6. There can be any number of elseifs, but at most one else.
7. elseif must be written as one word.
8. It is good programming style to indent each group of statements as shown.

2.5 Plotting in MATLAB:

2.5.1 Basic Two Dimensional Graphs:

Graphs (in 2D) are drawn with the **plot** statement. In its simplest form, **plot** takes a single vector argument, as in plot(y). In this case, the elements of y are plotted against their indexes. For example, plot(rand(1, 20)) plots 20 random numbers against the integers 1–20, joining successive points with straight lines, as in Figure 2.1. Axes are automatically scaled and drawn to include the minimum and maximum data points. For example,

```
>> plot(rand(1,20))
```

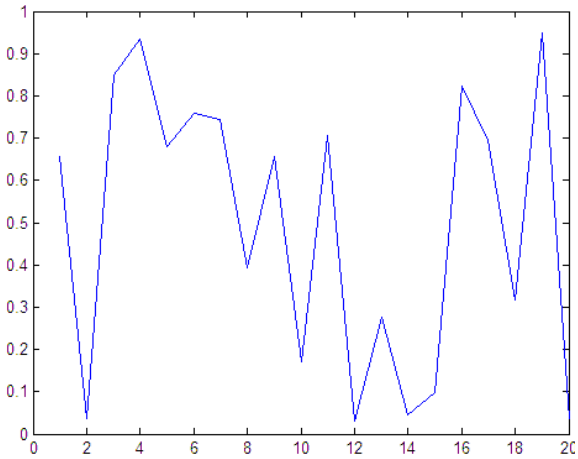


Figure 2.1 Plot of 20 random numbers

Probably the most common form of plot is plot(x, y), where x and y are vectors of the same length:

```
>> x=0:pi/40:4*pi;
>> plot(x, sin(x))
```

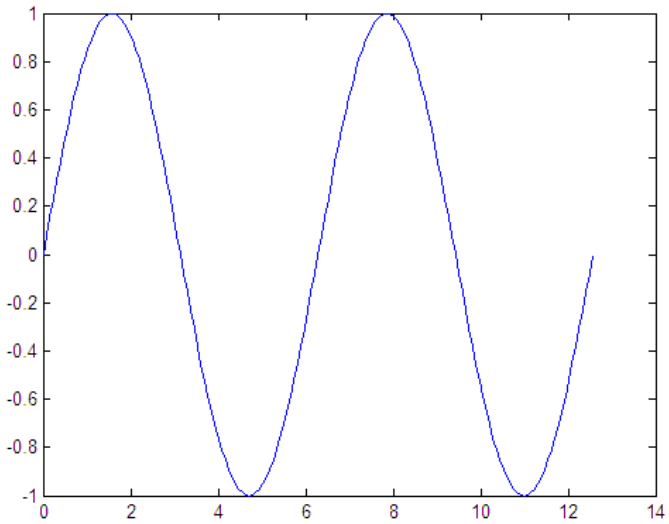


Figure 2.2 Plot of sine function

Straight-line graphs are drawn by giving the x and y coordinates of the end points in two vectors. For example, to draw a line between the points with Cartesian coordinates (0, 1) and (4, 3), use the statement

plot([0 4], [1 3])

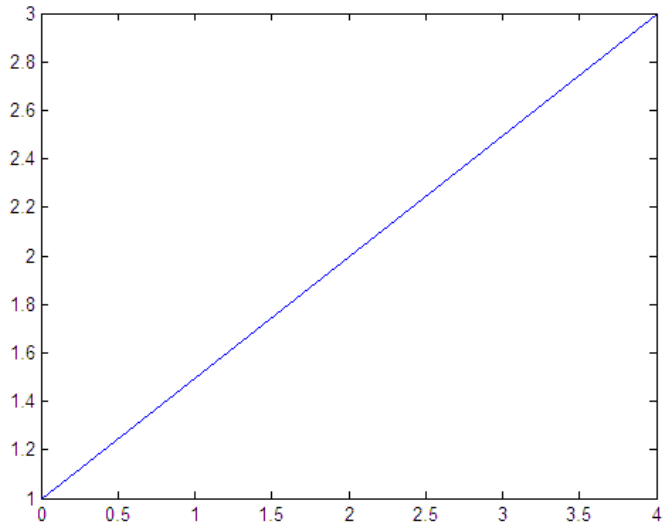


Figure 2.3 Plot of numbers

That is, [0 4] contains the x coordinates of the two points, and [1 3] contains the y coordinates.

MATLAB has a set of easy-to-use plotting commands, all starting with the string **ez**. The easy-to-use form of **plot** is **ezplot**. It is important to note that there is no requirement of a vector x in order to execute the ezplot command. This command is mostly used in order to plot the build in trigonometric function of MATLAB such as acos(x), cosh(x) etc.

>> ezplot('tan(x)')

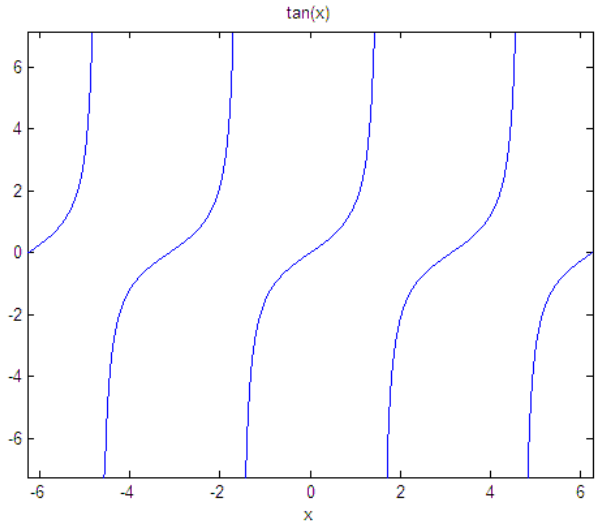


Figure 2.4 Plot of tangent function using ezplot

2.5.1.1 Labels

Graphs may be labeled with the following statements:

gtext('text') writes a string (text) in the graph window. It puts a crosshair in the graph window and waits for a mouse button or keyboard key to be pressed. The crosshair can be positioned with the mouse or the arrow keys. For example,

gtext('X marks the spot')

Text may also be placed on a graph interactively with **Tools→Edit Plot** from the figure window. grid adds/removes grid lines to/from the current graph. The grid state may be toggled.

text(x, y, 'text') writes text in the graphics window at the point specified by x and y. If x and y are vectors, the text is written at each point. If the text is an indexed list, successive points are labeled with its corresponding rows.

title('text') writes the text as a title at the top of the graph.

xlabel('horizontal') labels the x-axis.

ylabel('vertical') labels the y-axis.

2.5.1.2 Multiple plots on the same axes:

There are at least two ways of drawing multiple plots on the same set of axes (which may however be rescaled if the current data falls outside the range of the previous data).

- Simply to use hold to keep the current plot on the axes. All subsequent plots are added to the axes until hold is released, with either hold off or just hold, which toggles the hold state.
- Use plot with multiple arguments. For example,
  plot(x1, y1, x2, y2, x3, y3, ...)
  plots the (vector) pairs (x1, y1), (x2, y2), and so on. The advantage of this method is that the vector pairs may have different lengths. MATLAB automatically selects a different color for each pair. If you are plotting two graphs on

the same axes, you may find **plotyy** useful—it allows you to have independent y-axis labels on the left and the right:

plotyy(x,sin(x), x, 10*cos(x))

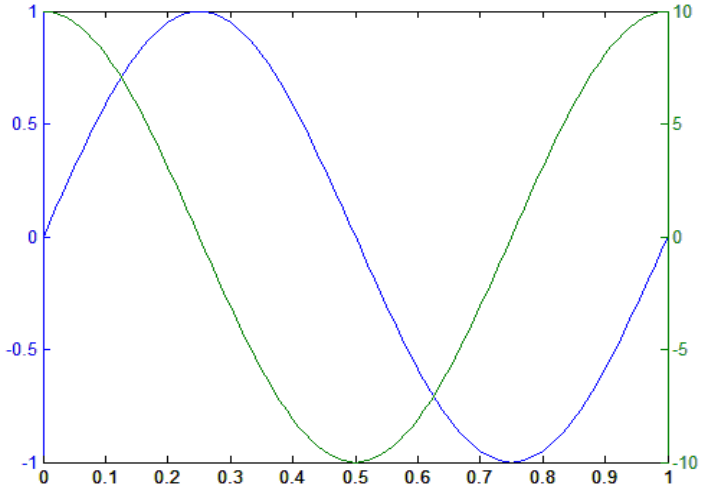


Figure 2.5 plotyy command

2.5.1.3 Line styles, markers, and color:

Line styles, markers, and color for a graph may be selected with a string argument to plot. For example, plot(x, y, '--') joins the plotted points with dashed lines, whereas plot(x, y, 'o') draws circles at the data points with no lines joining them. You can specify all three properties: plot(x,sin(x), x, cos(x), 'om--') which plots sin(x) in the default style and color and cos(x) with circles joined by dashes in magenta. The available colors are denoted by the symbols c, m, y, k, r, g, b, w.

Table 2.4: Line Styles and Colours in MATLAB

Colours		**Line Styles**	
Yellow	Y	Point	.
Magenta	M	Circle	o
Cyan	C	X-mark	x
Red	R	Plus	+
Green	G	Solid	-
Blue	B	Star	*
While	W	Dotted	:
Black	K	Dashed dot	-.
		Dashed	--

2.5.1.4 Axis limits:

Whenever you draw a graph, MATLAB automatically scales the axis limits to fit the data. You can override this with axis([xmin, xmax, ymin, ymax])

which sets the scaling on the current plot. In other words, draw the graph first, then reset the axis limits.

If you want to specify the minimum or maximum of a set of axis limits and have MATLAB autoscale the other, use Inf or -Inf for the autoscaled limit.

2.5.1.5 Multiple plots in a figure: subplot:

You can show a number of plots in the same figure window with the subplot function. It looks a little curious at first, but getting the hang of it is quite easy. The statement

subplot(m,n,p)

divides the figure window into m × n small sets of axes and selects the pth set for the current plot (numbered by row from the left of the top row). For example,

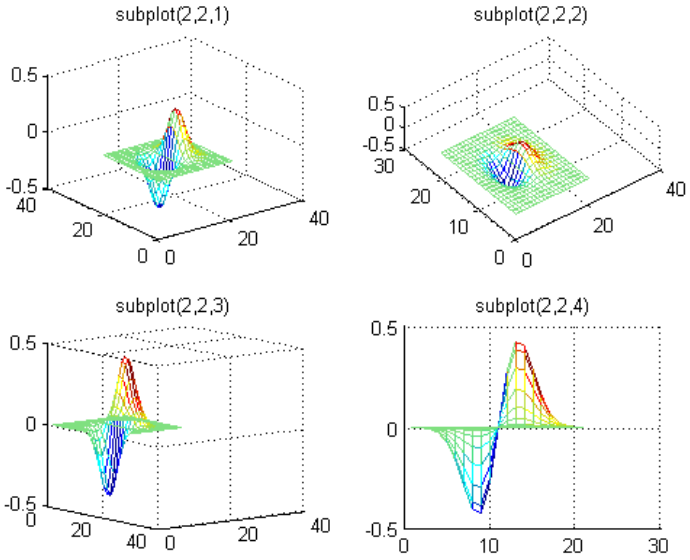


Figure 2.6 Plot using subplot command

2.5.1.6 Logarithmic Plots:

The command

semilogy(x, y)

plots y with a log10 scale and x with a linear scale. For example,

```
x = 0:0.01:4;
semilogy(x, exp(x)), grid
```

produce the graph in figure below. Equal increments along the y-axis represent multiples of powers of 10, so, starting from the bottom, the grid lines are drawn at 1, 2, 3, . . . , 10, 20, 30. . . , 100, ... Incidentally, the graph of e^x on these axes is a straight line because the equation $y = e^x$ transforms into a linear equation when you take logs of both sides.

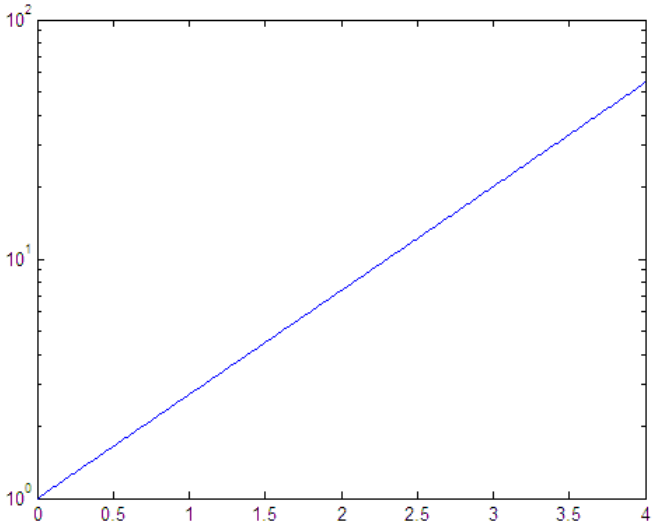


Figure 2.7 logarithmic Plot

2.5.1.7 Hold:

A call to plot clears the graphics window before plotting the current graph. This is not convenient if we wish to add further graphics to the figure at some later stage. To stop the window being cleared hold command is used. Let us see the following example:

```
clear all
close all
t=0:0.001:1;
x=sin(2*pi*t);
y=cos(2*pi*t);
plot(t,x,'r--'), grid on, hold on
plot(t,y,'b-')
```

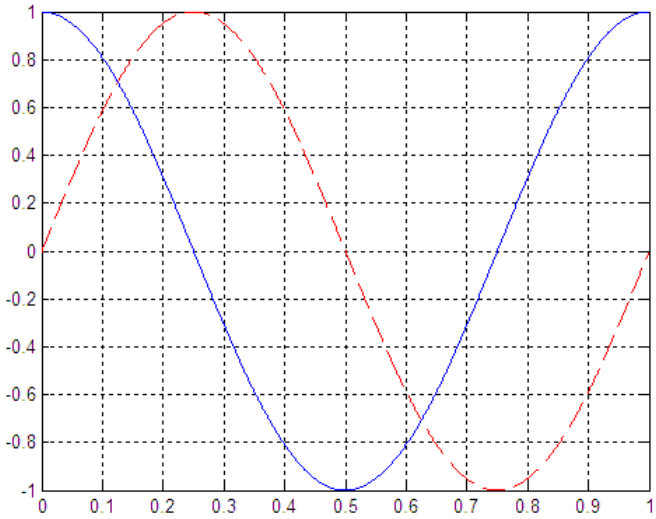


Figure 2.8 Hold on command

2.5.1.8 3D plots:

MATLAB also helps in 3D plotting of functions. 3D plotting is in fact beyond the scope of this lab but some command for 3D plotting that you should be familiar with are given below: Consider the following commands for 3D plotting

```
t = -4*pi:pi/16:4*pi;
x = cos(t);
y = sin(t);
z = t;
figure(1), plot3(x,y,z),grid on
[x, y] = meshgrid(-3:0.1:3,-3:0.1:3);
z = x.^2 - y.^2;
figure(2),mesh(x,y,z),grid on
figure(3),surf(x,y,z), grid on
figure(4),plot3(x,y,z),grid on
```

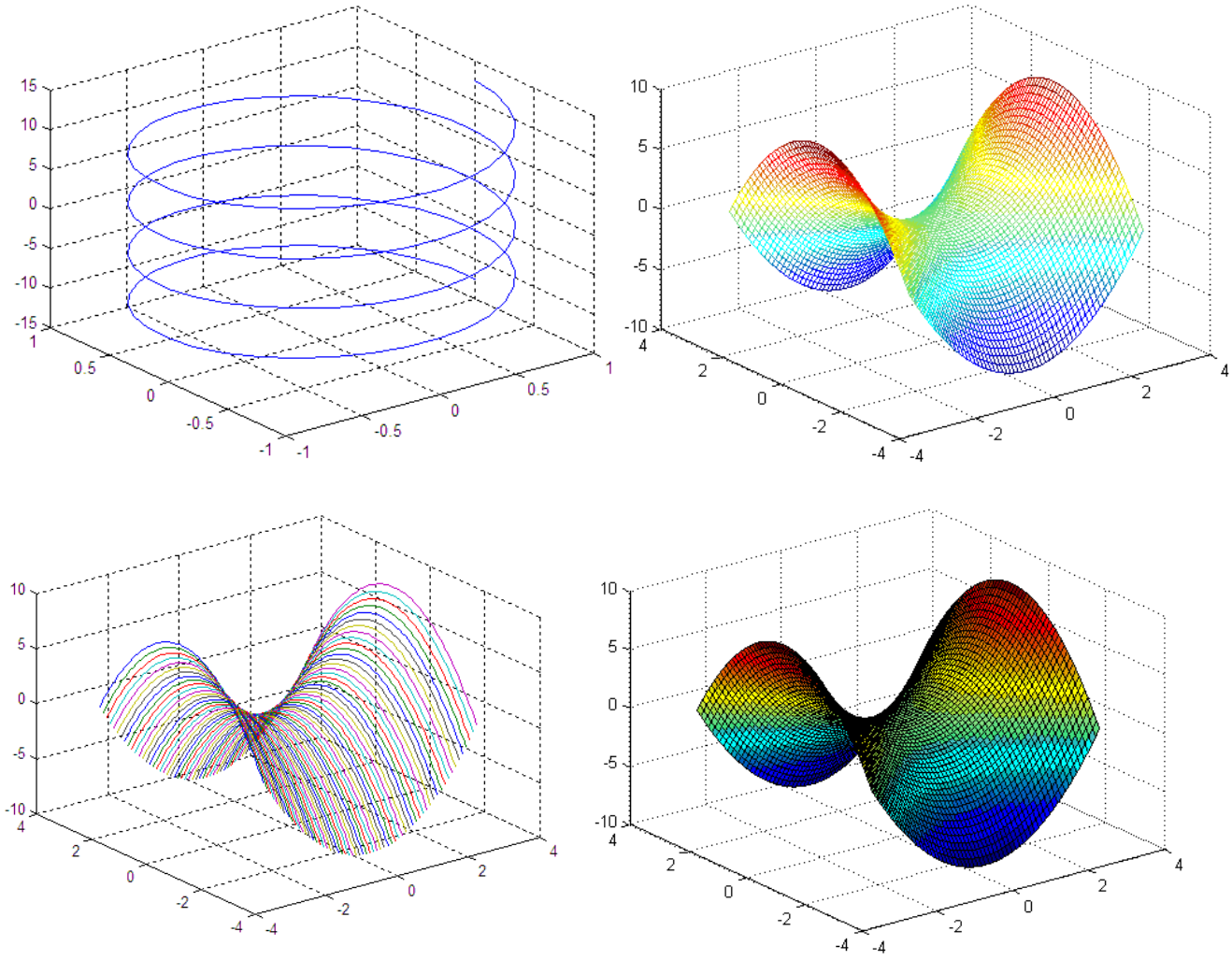


Figure 2.9 3D Plots

2.5.1.8 Various plotting functions:

MATLAB supports various types of graph and surface plots such as

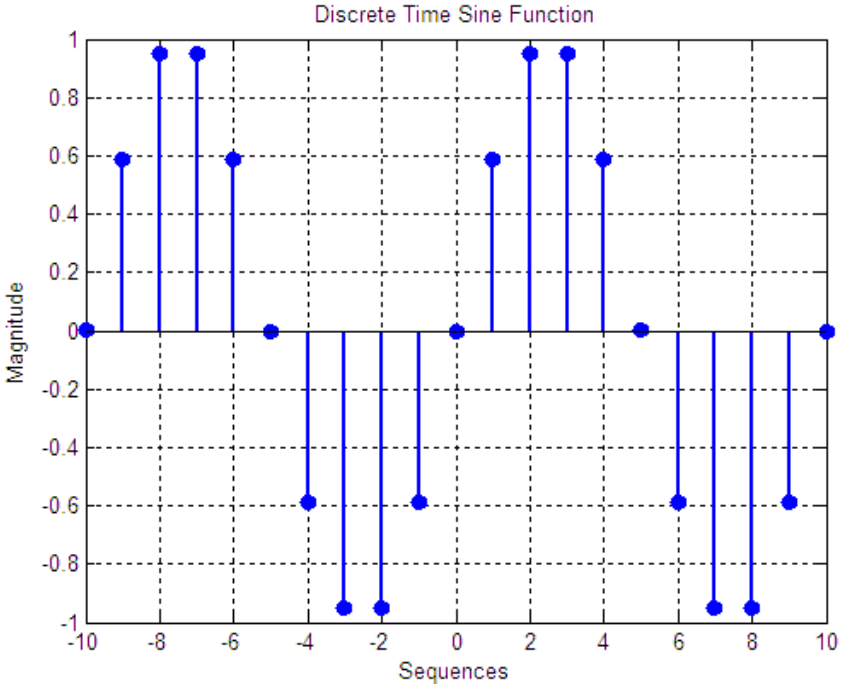
- dimensions line plots (x vs. y),
- filled plots,
- bar charts,
- pie charts,

- parametric plots,
- polar plots,
- contour plots,
- density plots,
- log axis plots,
- surface plots,
- parametric plots in 3 dimensions
- and spherical plots.

2.6. Plotting Discrete Time Sequences in MATLAB

The plot of continuous time signals in MATLAB is an artificial concept because MATLAB only deals with discrete data noting is continuous in MATLAB. In order to generate a continuous time signal the increment is so small that MATLAB artificially plots a continuous time signal. In order to draw a discrete time signal **stem** command is used and an example is given below.

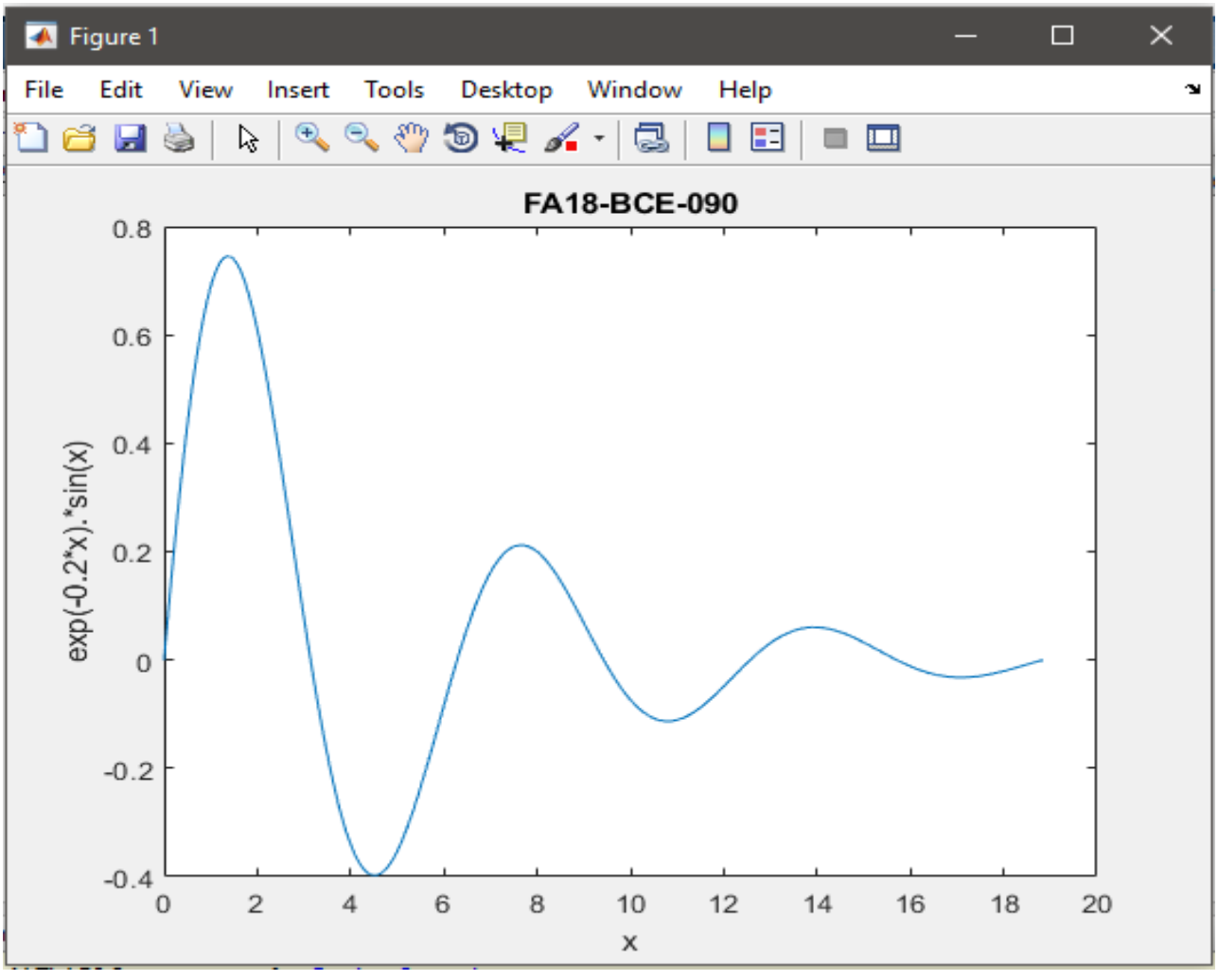
```
clear all;
close all;
n=-10:10;
y=sin(2*pi/10*n);
stem(n,y,'fill','Linewidth',2), grid on
xlabel('Sequences'); ylabel('Magnitude');
title('Discrete Time Sine Function');
```



Pre-lab Tasks

Task 01: Plot a function $e^{-0.2x}sin(x)$ between the interval 0 to 6π .

```
x = 0:pi/100:6*pi;
y = exp(-0.2*x).*sin(x);
plot(x,y); title('FA18-BCE-090'); xlabel('x'); ylabel('exp(-0.2*x).*sin(x)');
```



In-lab Tasks

Task 02: Make a function 'my_sum' which takes two vectors x and y as input. The vectors should have different sizes. Firstly, make size of both the input vectors same by padding (concatenating) zeros. Now add both the vectors element wise. The function should return a variable z which should contain the sum of all the elements of the vectors formed as a result of summation of input vectors x and y.

```
function [sum] = LAB1_task2(x,y)
a = length(x);
b = length(y);
if a>b
    temp = a-b;
    adj = zeros(1,temp);
    y = [y adj];
    disp('FA18-BCE-085');
    sum = x+y;
elseif b>a
    temp = b-a;
    adj = zeros(1,temp);
    x = [x adj];
    disp('FA18-BCE-085');
    sum = x+y;
else
    disp('FA18-BCE-085');
    sum = x+y;
end


>> x= [5 6]

x =

     5     6

>> y= [1,2,3,4]

y =

     1     2     3     4

>> LAB1_task2(x,y)
FA18-BCE-090

ans =

     6     8     3     4
```

Task 03: Write MATLAB programs to find the following with for loops and vectorization. Time both versions in each case.

a) $1^2 + 2^2 + 3^2 + 4^2 + \cdots + 1000^2$

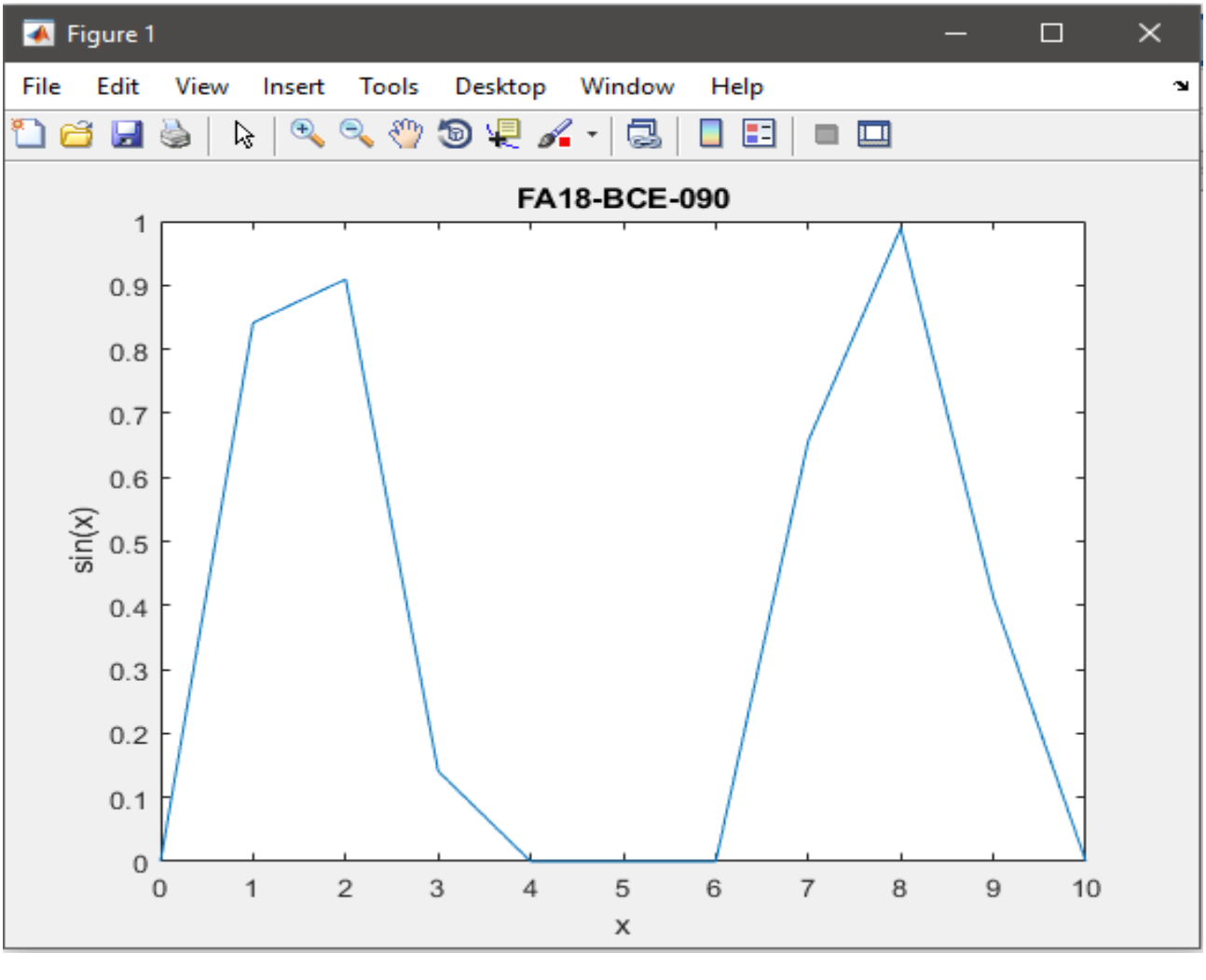
b) $1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots\cdots - \frac{1}{1003}$

```
%A.
tic
total = 0;
for i = 1:1000;
    total = total+i^2;
end
disp('a)')
total
toc
%B.
tic
total = 0;
for i = 1:1003;
    total = total +(-1^i+1/2*i-1);
end
disp('b)')
total
toc


>> LAB1_task3
a)

total =

   333833500

Elapsed time is 0.001035 seconds.
b)

total =

      249747
```

Task 04: Graph the following function in MATLAB over the range of 0 to 10.

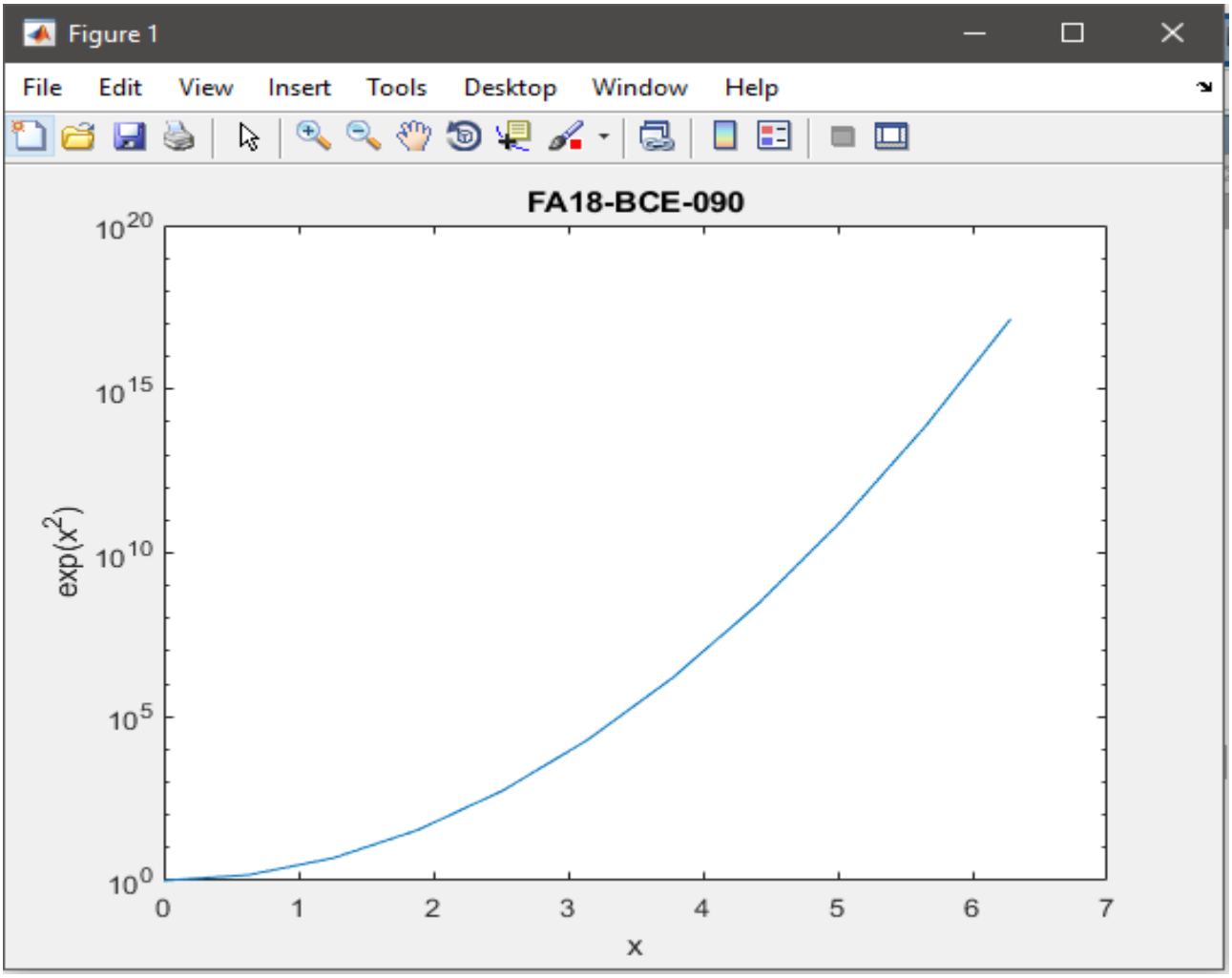
$$y(x) = \begin{cases} sin(x), & (sin(x) > 0) \\ 0, & (sin(x) \leq 0) \end{cases}$$

```
x = 0:10;
for i = 0:10
if sin(i) > 0
    y(1,i+1) = sin(i);
else
    y(1,i+1) = 0;
end
end
plot(x,y); title('FA18-BCE-090'); xlabel('x'); ylabel('sin(x)');
```



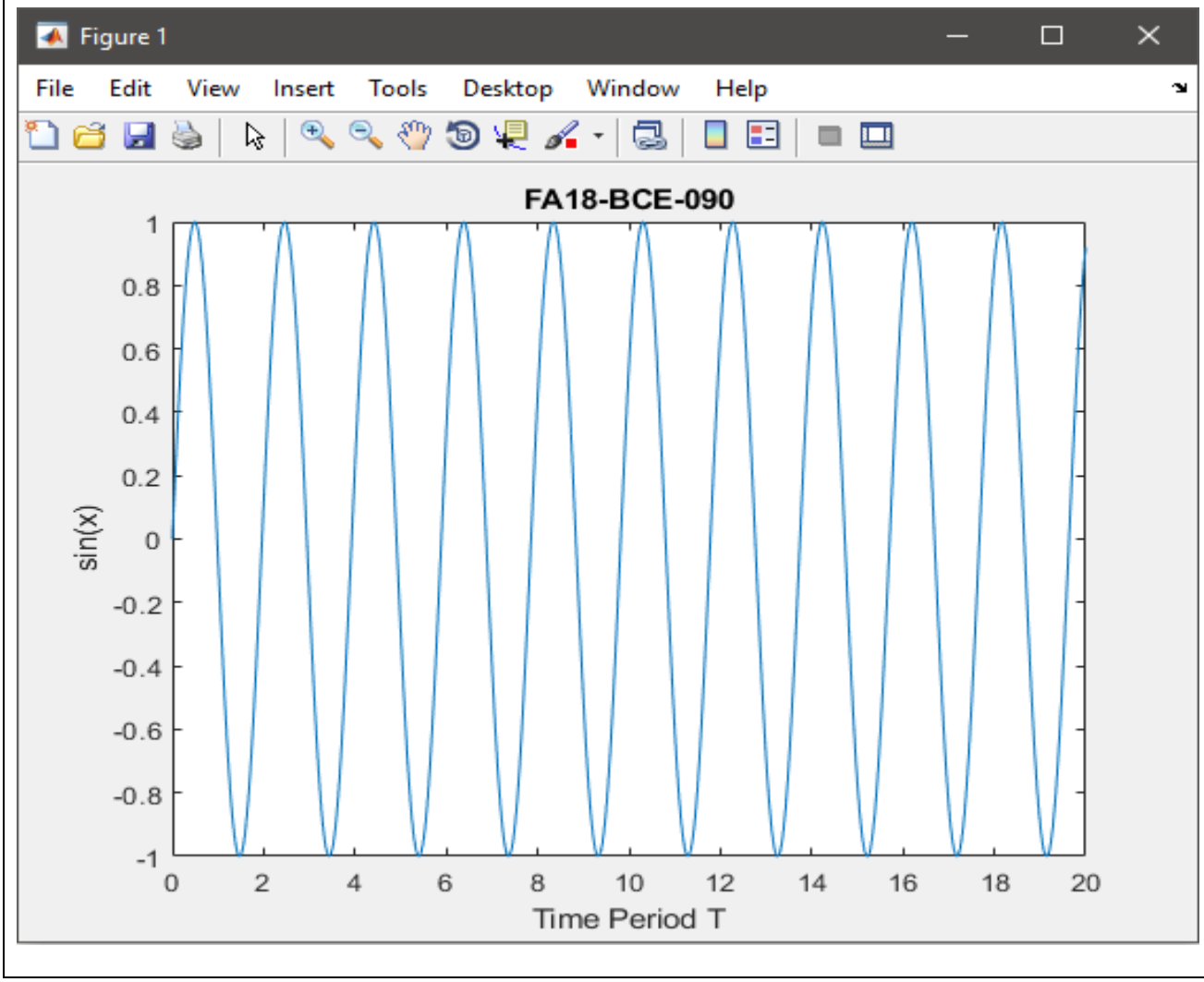
Task 05: Use the semi log graph to graph $x^2, x^3, x^4 and\ e^{x^2}$ over the interval of $0 \leq x \leq 2\pi$.

```
%A
x = 0:pi/5:2*pi;
y = x.^2;
%semilogy(x,y);title('FA18-BCE-090'); xlabel('x'); ylabel('x^2');
%B
x = 0:pi/5:2*pi;
y = x.^3;
%semilogy(x,y);title('FA18-BCE-090'); xlabel('x'); ylabel('x^3');
%C
x = 0:pi/5:2*pi;
y = x.^4;
%semilogy(x,y);title('FA18-BCE-090'); xlabel('x'); ylabel('x^4');
%D
x = 0:pi/5:2*pi;
y = exp(x.^2);
semilogy(x,y);title('FA18-BCE-090'); xlabel('x'); ylabel('exp(x^2)');
```



Task 06: Plot the first ten cycles of sinusoid (sin) with time period of 2 seconds. The horizontal axis corresponds to time period of the sinusoid while vertical axis depicts the values of sinusoid.

```
x = 0:0.01:20;
y = sin(3.2*x);
plot(x,y);title('FA18-BCE-090'); xlabel('Time Period T'); ylabel('sin(x)');
```



Task 07: Plot the following discrete sequences

a. $x[n] = \alpha^n \quad 0 \leq n \leq 10, \alpha = 1.5$
b. $x[n] = \beta^n \quad -10 \leq n \leq -1, \beta = 1.5$
c. $x[n] = \gamma^n \quad 0 \leq n \leq 10, \gamma = 0.5$
d. $x[n] = \varphi^n \quad -10 \leq n \leq -1, \varphi = 0.5$

Horizontal axis corresponds to instances n and vertical axis depicts $x[n]$ values.

```
%1.
n = 0:10;
alpha=1.5;
y = alpha.^n;
stem(n,y,'fill','Linewidth',2),grid on;xlabel('n');ylabel('x[n]');title('FA18-
BCE-090');
%2.
n = -10:-1;
beta = 1.5;
y = beta.^n;
stem(n,y,'fill','Linewidth',2),grid on;xlabel('n');ylabel('x[n]');title('FA18-
BCE-090');
%3.
n = 0:10;
gamma = 0.5;
y= gamma.^n;
stem(n,y,'fill','Linewidth',2),grid on;xlabel('n');ylabel('x[n]');title('FA18-
BCE-090');
%4.
n = -10:-1;
phi = 0.5;
y = phi.^n;
stem(n,y,'fill','Linewidth',2),grid on;xlabel('n');ylabel('x[n]');title('FA18-
BCE-090');
```

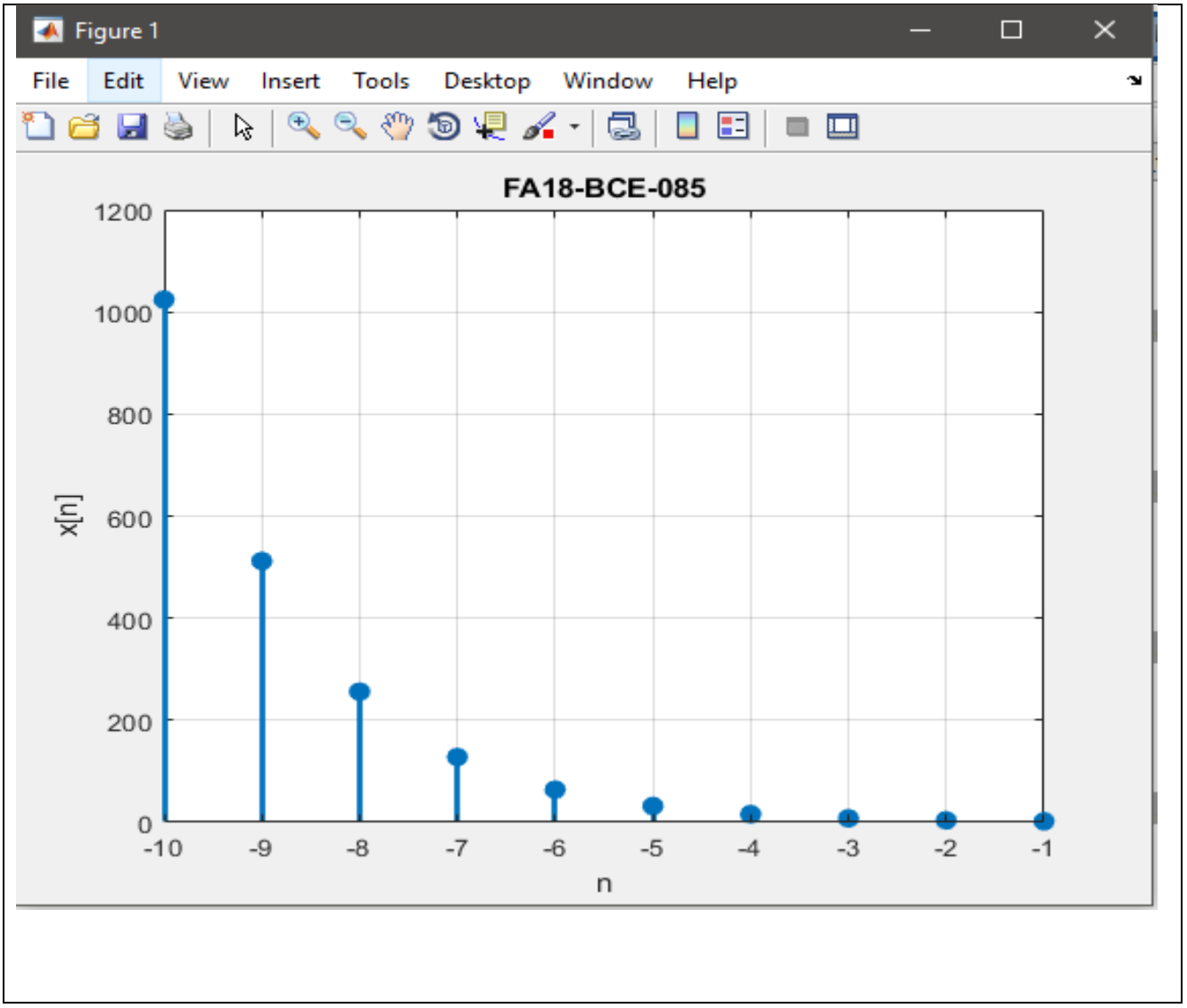


Figure 1
File Edit View Insert Tools Desktop Window Help
FA18-BCE-085
1200
1000
800
600
400
200
0
x[n]
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1
n

Task 08: $x(t) = A\,\mathbb{R}\{e^{j(\omega_0 t+\varphi)}\}$

Where A = 1, f_0 = 10 Hz, $\varphi = 0$

Plot the second and third harmonics of the above sinusoid. Use SUBPLOT command for drawing in the same manner as in previous question. After plotting explain the relation of fundamental Time period and harmonics Time period clearly. Your answer should corroborate the graphs drawn.

```
x = 1:0.001:2;

A=1; f0=10; f1=20; f2=30; s=0;
P=2*pi*f0*x+s;
Q=2*pi*f1*x+s;
R=2*pi*f2*x+s;
p1=exp(1i.*P);
p2=exp(1i.*Q);
p3=exp(1i.*Q);
g1=A*real(p1);
g2=A*real(p2);
g3=A*real(p3);

subplot(3,1,1);plot(x,g1);title('FA18-BCE-090'); ylabel('f=10hz');
subplot(3,1,2);plot(x,g2);title('FA18-BCE-090'); ylabel('f=20hz');
subplot(3,1,3);plot(x,g3);title('FA18-BCE-090'); ylabel('f=30hz');
```

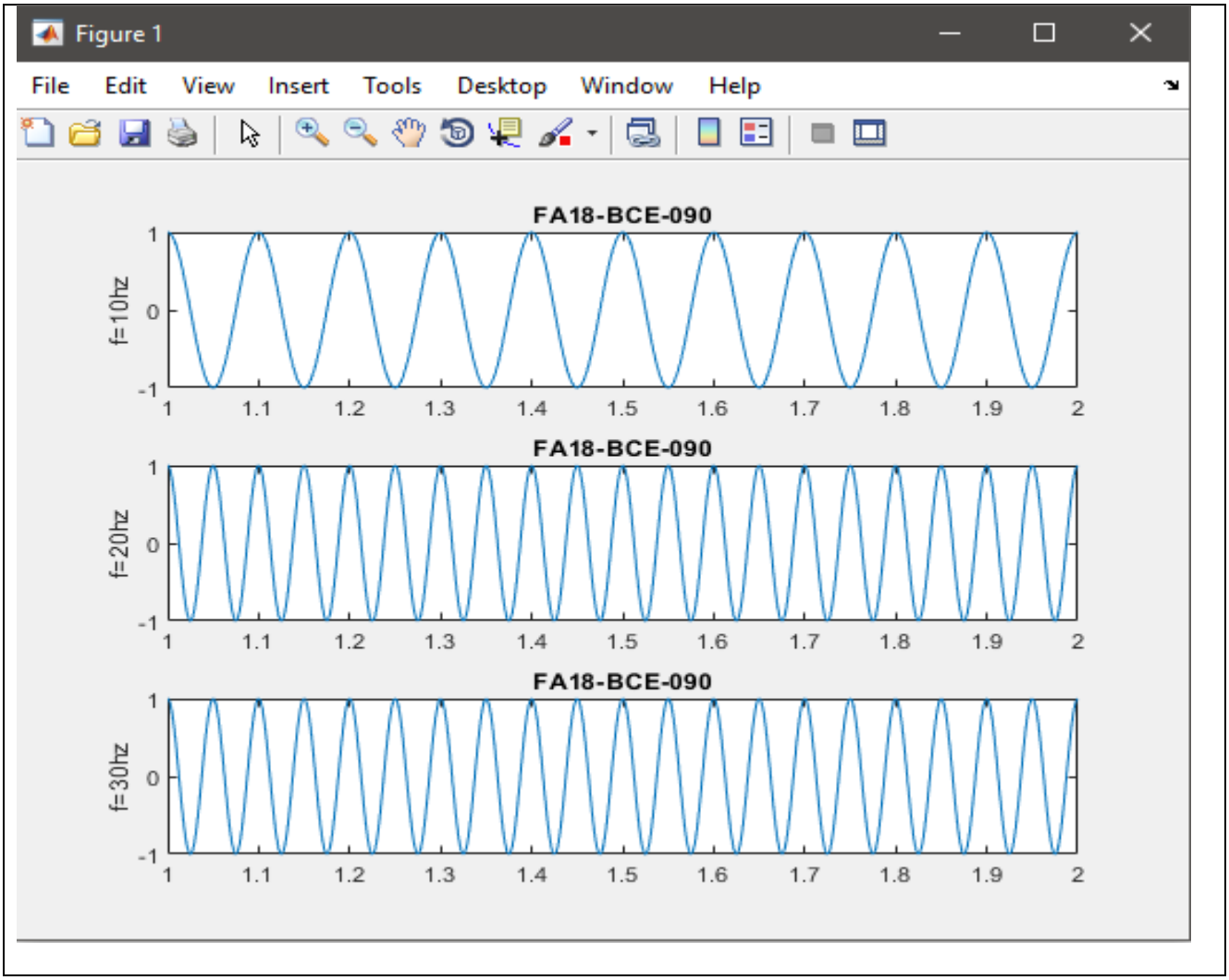


Figure 1
File Edit View Insert Tools Desktop Window Help
FA18-BCE-090
f=10hz
FA18-BCE-090
f=20hz
FA18-BCE-090
f=30hz
1 0 -1
1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2

Post-lab Task

Critical Analysis / Conclusion

In this lab, we learnt some new features of MATLAB, which includes graphs for different types of functions as an output with the help of programming (loops, if-else statements). We also learned how to use different operational commands for mathematical operations and how to program piece-wise defined functions and plotting the graph with defined labels.

Lab Assessment		
Pre-Lab	**/1**	**/10**
In-Lab	**/5**	
Critical Analysis	**/4**	
Instructor Signature and Comments		

LAB # 03

Study of Signal characteristics using MATLAB

Lab 03- Study of Signal characteristics using MATLAB

Pre-lab Tasks

3.1 Basic Signal Generation in MATLAB:

3.1.1 Unit Step Function:

The Heaviside step function, or the unit step function, usually denoted by u, is a discontinuous function whose value is zero for negative argument and one for positive argument

The function is used in the mathematics of control theory and signal processing to represent a signal that switches on at a specified time and stays switched on indefinitely. Mathematically, a continuous time unit step function is written as

$$u(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases}$$

In discrete form, a unit step function can be mathematically written as

$$u[n] = \begin{cases} 1 & n \leq 0 \\ 0 & n > 0 \end{cases}$$

As explained in the previous lab, generation of continuous time signals is an artificial concept in MATLAB, so we would be dealing with discrete time unit step sequence. Graphically, unit step sequence is shown in figure 3.1.

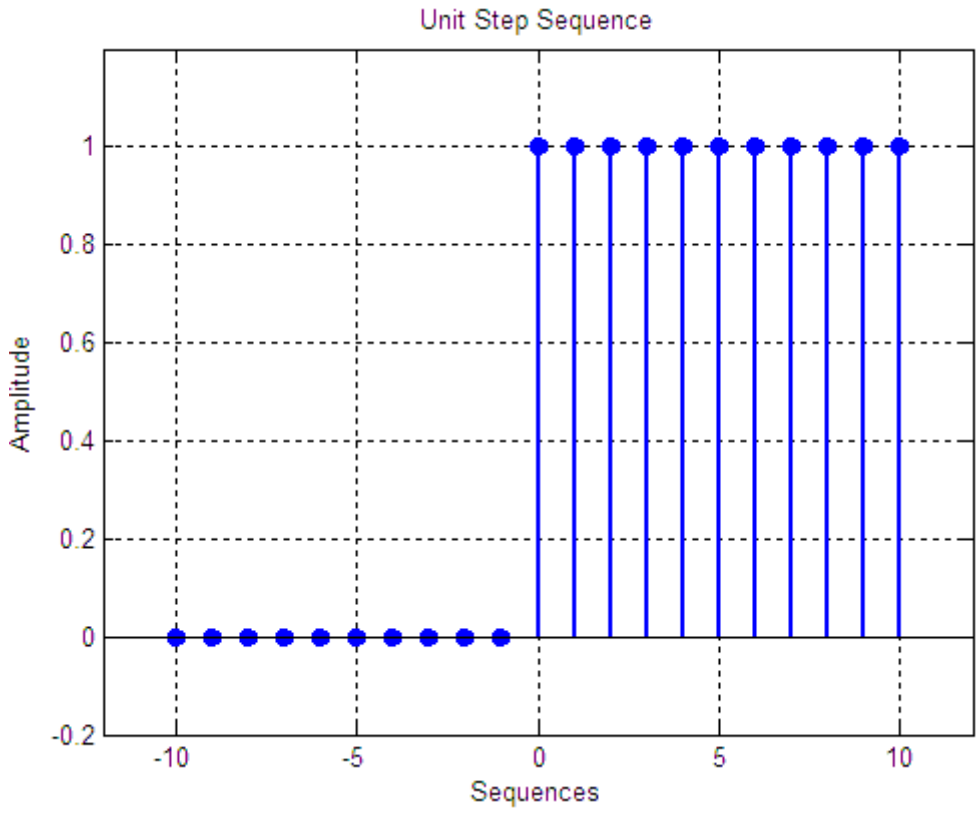


Figure 3.1: Unit Step Sequence

3.1.2 Unit Impulse Sequence:

Unit Impulse sequence is very important to characterize the impulse response of the system. Mathematically an impulse sequence is represented as:

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$

Graphically an impulse sequence is shown in figure 3.2

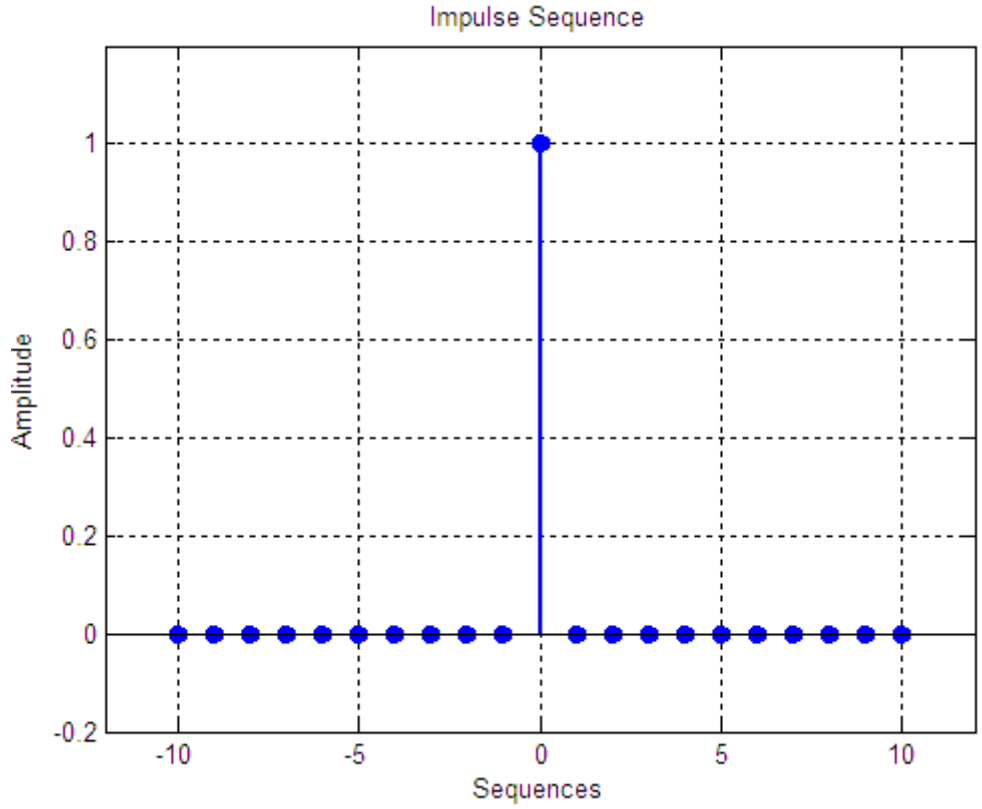


Figure 3.2: Impulse Sequence

3.1.3 Ramp Sequence:

Mathematically a ramp sequence is defined as:

$$r[n] = \begin{cases} n & n \geq 0 \\ 0 & n < 0 \end{cases}$$

Graphically a ramp sequence is shown in figure 3.3

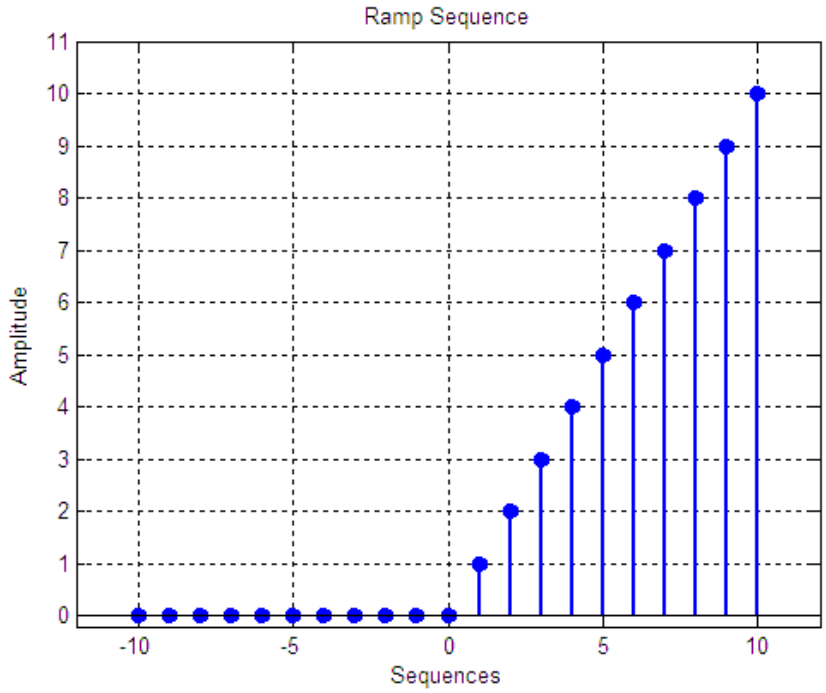


Figure 3.3: Ramp Sequence

3.1.4 Signum Sequence:

Signum sequence can be mathematically written as:

$$\text{sgn}[n] = \begin{cases} -1 & n < 0 \\ 1 & n \geq 0 \end{cases}$$

Graphically, signum sequence is shown in figure 3.4

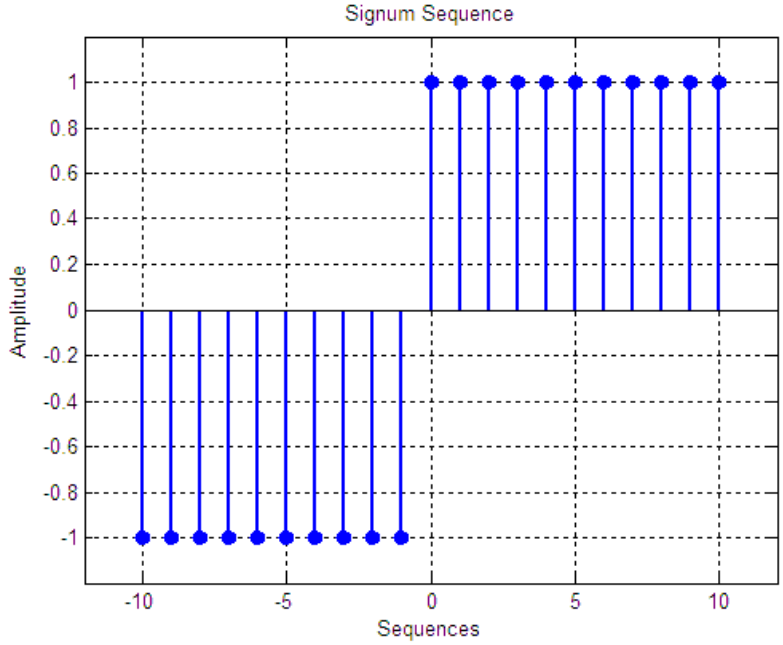


Figure 3.4: Signum Sequence

3.1.5 Square Wave:

Square wave is very important in communications related aspect. For example all the line coding techniques like RZ, NRZ etc consists of square waves. For wireline communications the line coded bits are communicated and for wireless communications the bits are first coded and then modulated. 5Hz square wave is shown in shown in figure 3.5.

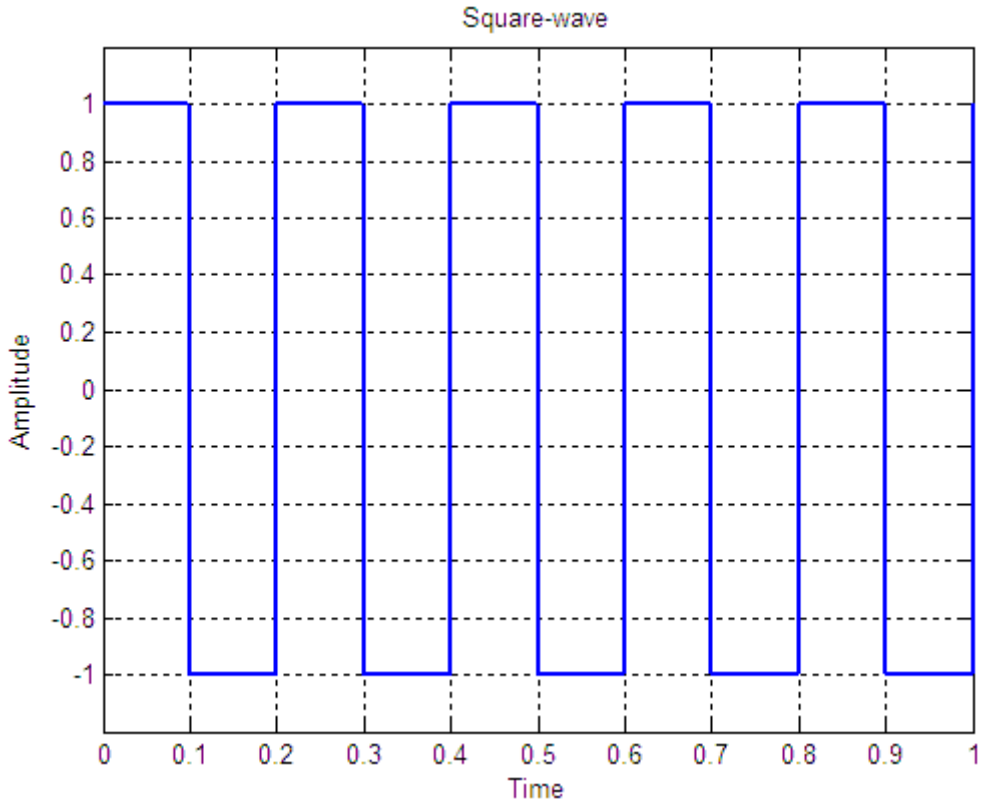


Figure 3.5: Square Wave

3.2 Periodic Sequences in MATLAB:

A discrete-time signal $x[n]$ is said to be periodic if there exist a positive constant N (the period) for which $x[n+N]=x[n]$, for all n. We can easily generate periodic signal in MATLAB. For example, consider we have a vector $x=[1,1,0,0,-1,-1]$, and we want to periodically repeat the given vector four times, following is the MATLAB code to do this

```
x=[1 1 0 0 -1 -1];
n=1:length(x);
subplot(2,1,1)
stem(n,x,'fill', 'Linewidth',2),grid on
y=repmat(x,1,4);
n1=1:length(y);
subplot(2,1,2)
stem(n1,y,'fill', 'Linewidth',2),grid on
```

repmat is the command which is used to repeat a matrix or a vector.

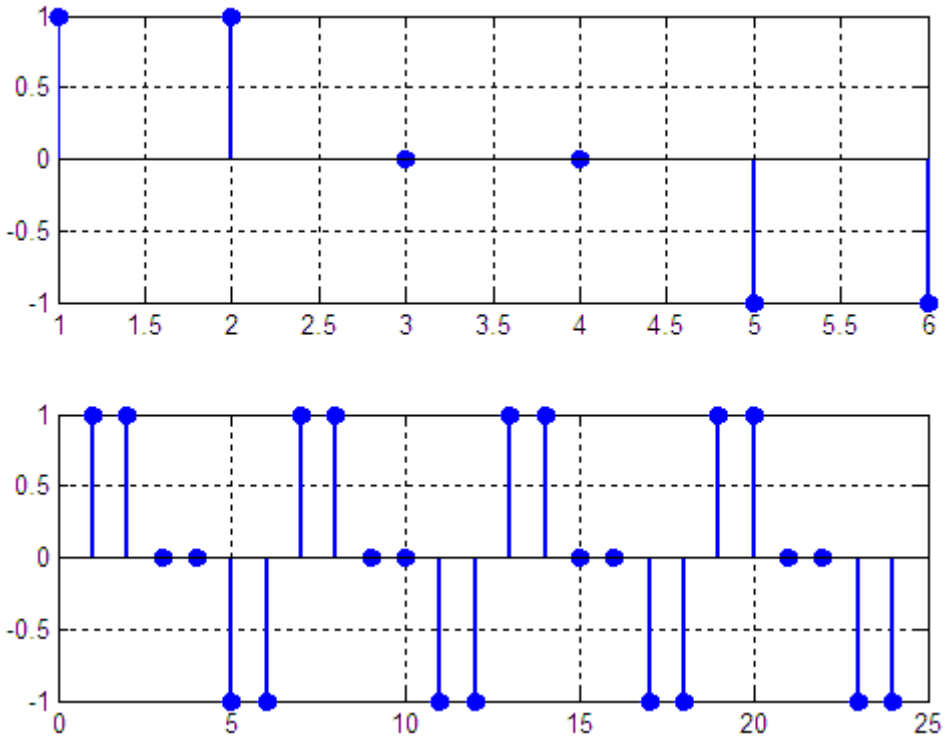


Figure 3.6: Periodic Sequences

3.3 Energy and Power of Continuous Time Signal in MATLAB:

The term signal energy of a signal is defined as the area under the square of magnitude of the signal. The signal energy of a continuous time signal $x(t)$ is given as

$$E_x = \int_{-\infty}^{\infty} |x(t)|^2 \, dt$$

Consider the following code to find the energy of a signal $x(t) = 2t$ over the interval of 0 to π.

```
t=0:0.0001:10;
x=2*t;
xsq=x.^2;
Ex=trapz(t,abs(xsq))
Ex =
1.3333e+003
```

Sometimes, there are signals with infinite energy, in those cases it is more convenient to deal with the average power of the signal. The average power of the periodic signal $x(t)$ is defined by

$$Px = \frac{1}{T}\int_T |x(t)|^2 \, dt$$

Consider the following MATLAB code for finding the power of $x(t) = 2t$ over the interval of 0 to π.

```
t=-5:0.1:5;
x=-3*t;
xsq=x.^2;
```

```
Px=(trapz(t,abs(xsq)))/10
Px =
75.0150
```

3.4 Energy and Power of Discrete Time Sequences in MATLAB:

The signal energy of a discrete time sequence is defined by

$$E_x = \sum_{n=-\infty}^{\infty} |x[n]|^2$$

Consider the following code, to determine the energy of a signal $x[n] = (0.9)^{|n|} \sin(2\pi n/4)$

```
n=-100:100;
x=((0.9).^(abs(n))).*sin(2*pi*n/4);
xsq=x.^2;
Ex=sum(xsq)
Ex =
4.7107
```

Similarly, signal power is defined by the following mathematical relation

$$Px = \frac{1}{N} \sum_{\langle n=N \rangle} |x[n]|^2$$

```
n=-100:100;
x=((0.9).^(abs(n))).*sin(2*pi*n/4);
xsq=x.^2;
Ex=sum(xsq)/200
Ex =
0.0236
```

3.5 Deterministic and Stochastic Signals:

A signal is deterministic if no randomness is involved in its value, i.e. its each value can be determined by some mathematical expression. On the other hand a signal is stochastic or non-deterministic is there is some randomness in the signal and the values of the signal cannot be predicted.

3.6 Even and Odd Signals:

A signal $x(t)$ is even if $x(-t) = x(t)$ and a signal is odd if $x(-t) = -x(t)$ for $-\infty < t < \infty$.

See the following example:

Find out if the signals $x(t) = t^2$ and $y(t) = t^3$ are even or odd.

Of course, we cannot examine these two signals over the time interval $(-\infty, \infty)$. The symmetry of the signals is tested in the time interval $-4 \le t \le 4$.

```
t1=0:4                t1=0  1.00  2.00  3.00   4.00
t2=0:-1:4             t2=0 -1.00 -2.00 -3.00  -4.00
x1=t1.^2              x1=0  1.00  4.00  9.00   16.00
```

x2=t2.^2 **x2=0 1.00 4.00 9.00 16.00**
y1=t1.^3 **y1=0 1.00 8.00 27.00 64.00**
y2=t2.^3 **y2=0 -1.00 -8.00 -27.00 -64.00**

From the above example, it is clear that $x(-t)=x(t)$, so it is an even signal and $y(-t)=-y(t)$ so, it is an odd signal.

A signal is not always even or odd. For example, the signal $x(t)=t^2/(t+5)$ is neither odd nor even. But all the signals can be expressed as sum of an even signal $x_e(t)$ and an odd signal $x_o(t)$; that is any signal $x(t)$ is expressed as $x(t)=x_e(t)+x_o(t)$, where

$$x_e(t)=\frac{1}{2}\left[x(t)+x(-t)\right]$$

$$x_o(t)=\frac{1}{2}\left[x(t)-x(-t)\right]$$

In discrete-time signals case, same rules are valid. A signal $x[n]$ is even if $x[n]=x[-n]$ and odd if $-x[n]=x[-n]$. Moreover, a discrete time signal $x[n]$ can be expressed as the sum of an even signal $x_e[n]$ and an odd signal $x_o[n]$, where $x_e[n]$ and $x_o[n]$ are given by

$$x_e[n]=\frac{1}{2}\left(x[n]+x[-n]\right)$$

$$x_o[n]=\frac{1}{2}\left(x[n]-x[-n]\right)$$

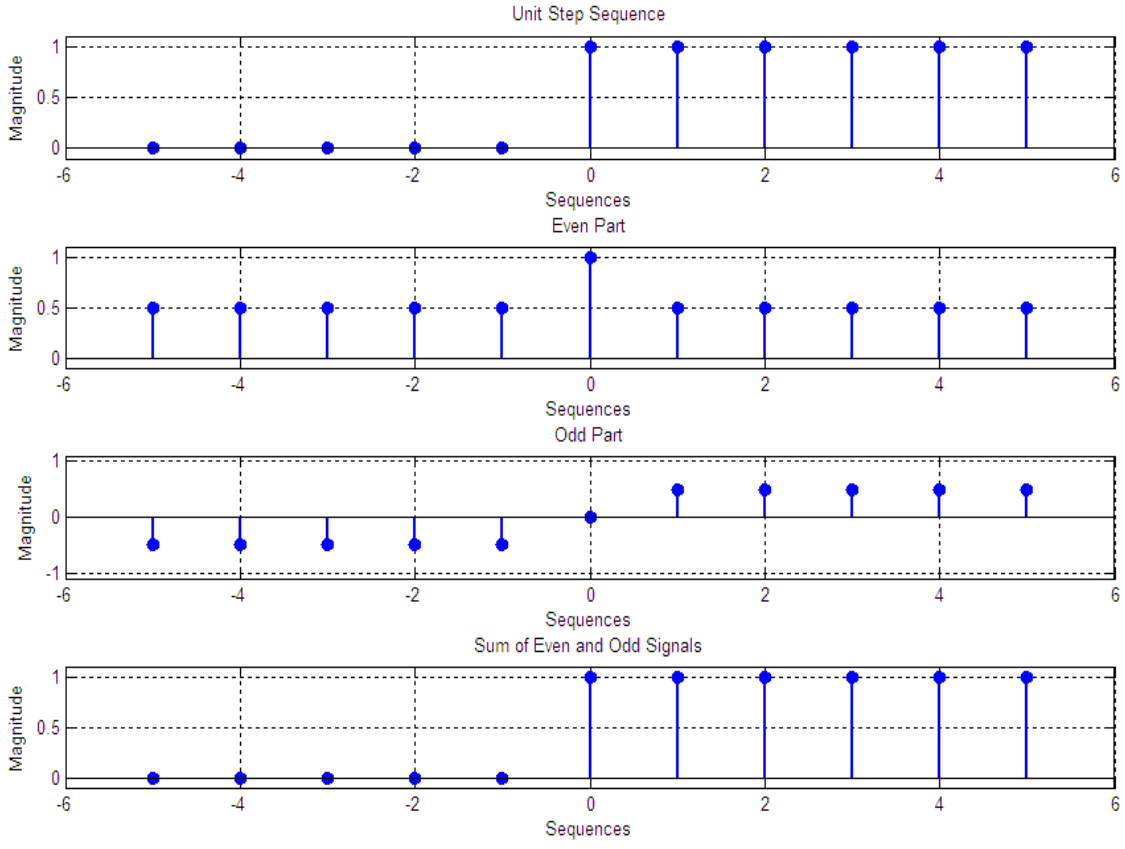


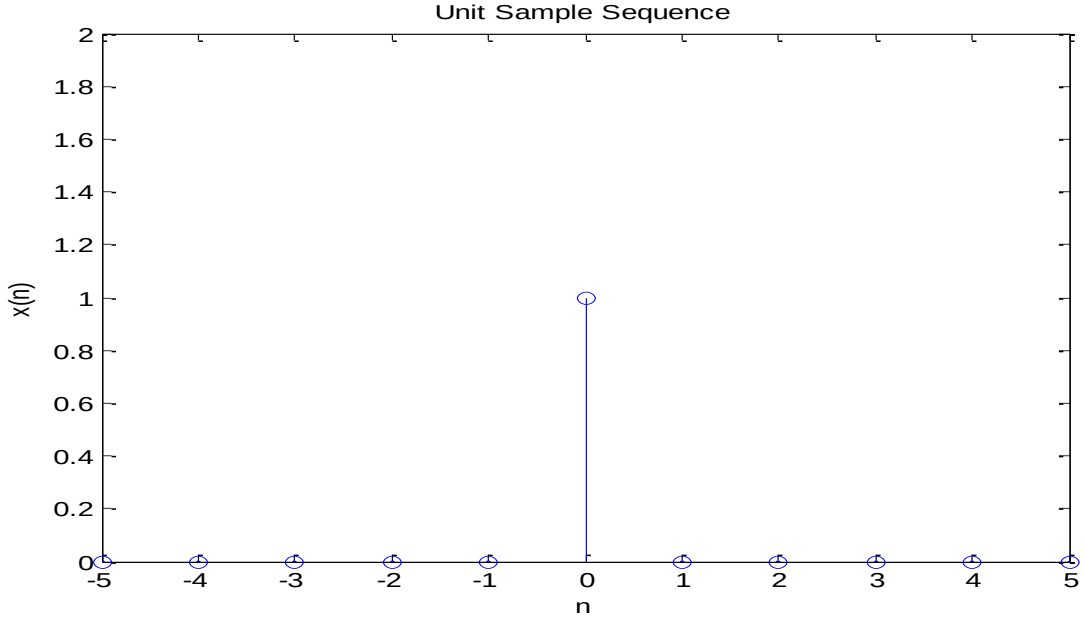
Figure 3.7: Unit Sequence with its even and odd parts

In-lab Tasks

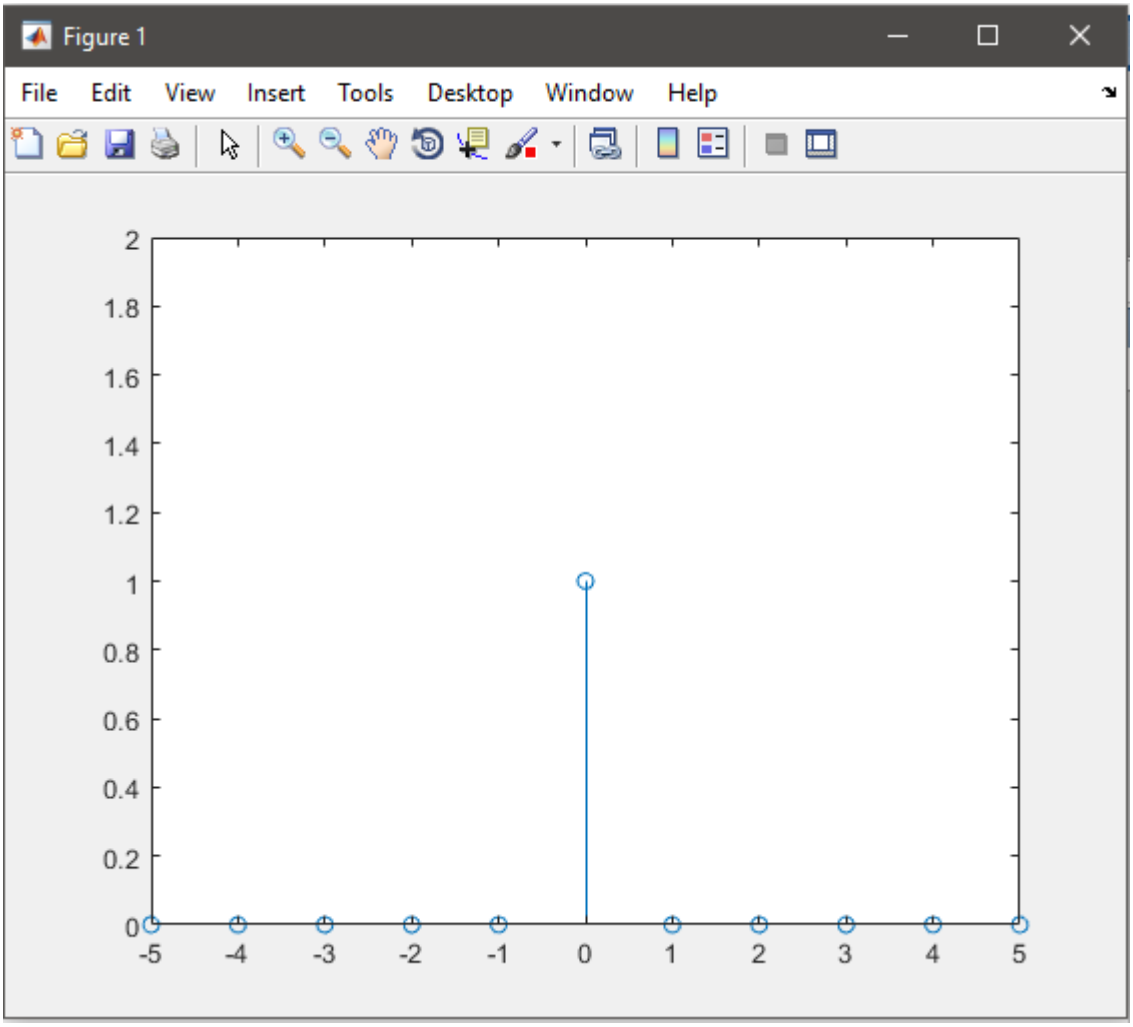
Task 01: Create a function "impseq", which performs following operations:

Function [x,n]=impseq(n0,n1,n2)

- Takes three parameters (n0, n1, n2) as input, where 'n1' and 'n2' are lower and upper limits of n-axis, and 'n0' is the delay.
- Generates a unit-impulse sequence using above mentioned three parameters.
- There should be two output arguments [x, n] of function 'impseq', where 'x' is impulse sequence and 'n' is its corresponding n-axis.
- Finally, plot unit impulse 'x' against vector 'n'.
- On the main window, type "[x,n]=impseq(0,-5,5)"
    - Unit Sample Sequence
        - The resulting plot looks like this



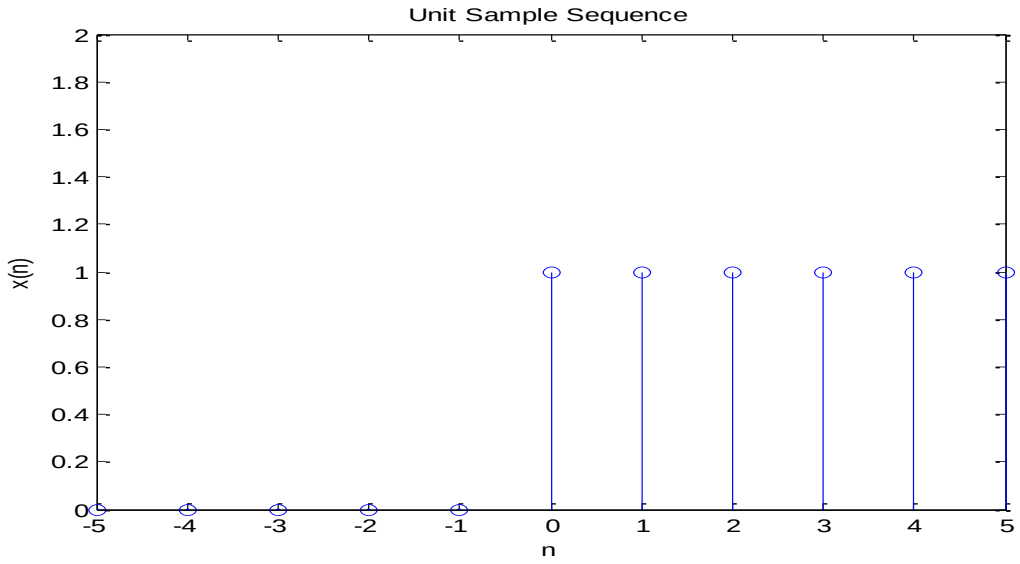
```
function [x, n]= Lab3Task1(n0,n1,n2)
n=n1:n2   ;           %n starts from n1 to n2
x= (n==0)  ;           %true: gives 1 at n=0
stem(n-n0,x);          % plots sequence x at the values specified in n-n0
axis([n1,n2,0,2]);     %axis limit defined
end
```



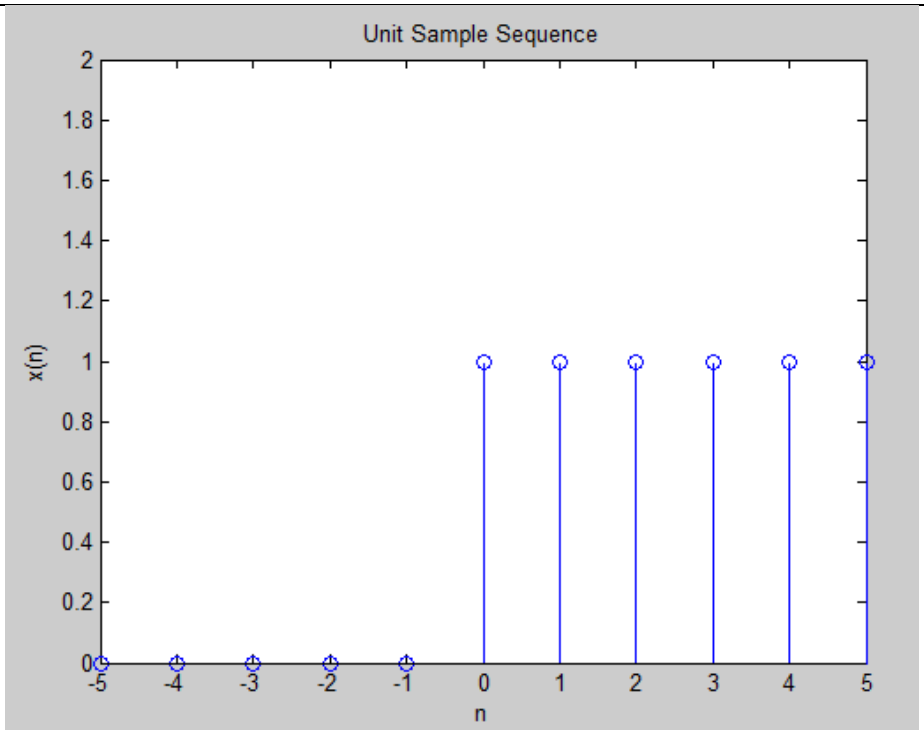
Task 02: Make a function to form "stepseq" function which will output unit-step sequence.

Function [x,n]=stepseq(n0,n1,n2)

- Unit Step Sequence
  - We can have another elegant way to produce a step function
  - Alternatively, we can use the "ones" function
  - Type "stepseq[x,n]=(0,-5,5)" we get:



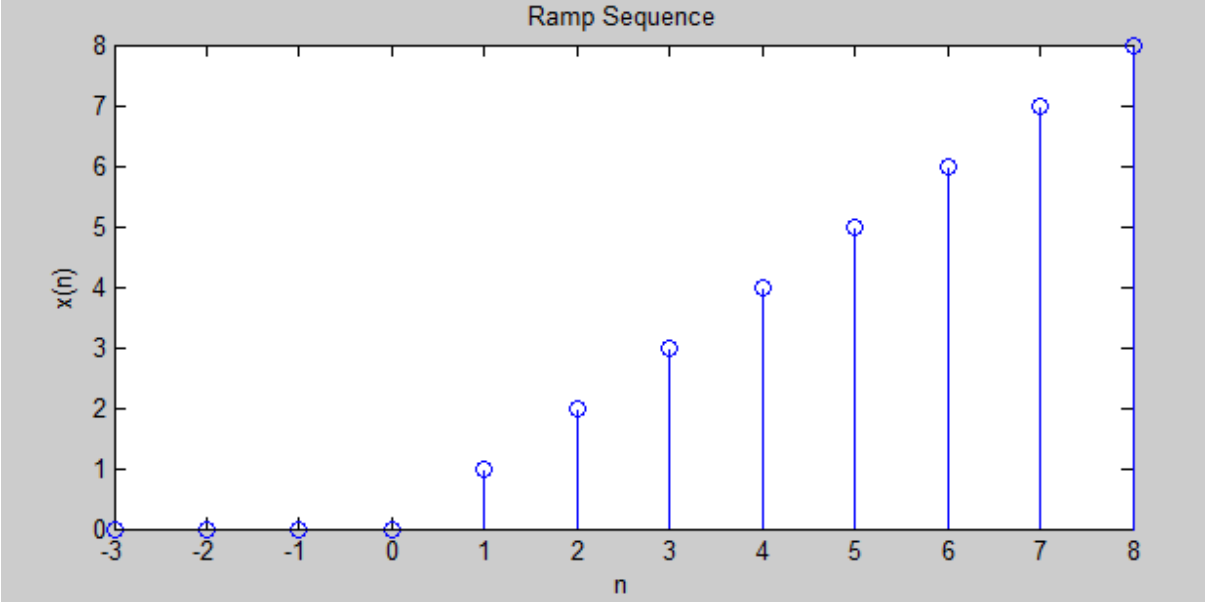

```
function [x, n] = stepseq(n0,n1,n2)
n = n1:n2;           %n starts from n1 to n2
x = (n>=0);          %true: gives 1 when n is 0 or grater than 0
stem(n-n0,x)         % plots sequence x at the values specified in n-n0
axis([n1,n2,0,2]);   %axis limit defined
xlabel('n');
ylabel('x(n)');
title('Unit Sample Sequence')
end


>>stepseq(0,-5,5)
```


Unit Sample Sequence
x(n)
n
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-5
-4
-3
-2
-1
0
1
2
3
4
5

Task 03: Create a function "rampseq", which performs following operations:
Function [x,n]=rampseq(n0,n1,n2)

- Takes three parameters (n0, n1, n2) as input, where 'n1' and 'n2' are lower and upper limits of n-axis, and 'n0' is the delay.
- Generates a ramp sequence using above mentioned three parameters.
- There should be two output arguments [x, n] of function 'rampseq', where 'x' is impulse sequence and 'n' is its corresponding n-axis.
- Finally, plot ramp impulse 'x' against vector 'n'.

```
function [x, n] = rampseq(n0,n1,n2)
n = n1:n2;           %n starts from n1 to n2
x = n.*(n>=0);       %Condition
stem(n-n0,x)         %plots sequence x at the values specified in n-n0
axis([n1,n2,0,n2]);  %axis limit defined
xlabel('n');
ylabel('x(n)');
title('Ramp Sequence')
end
```

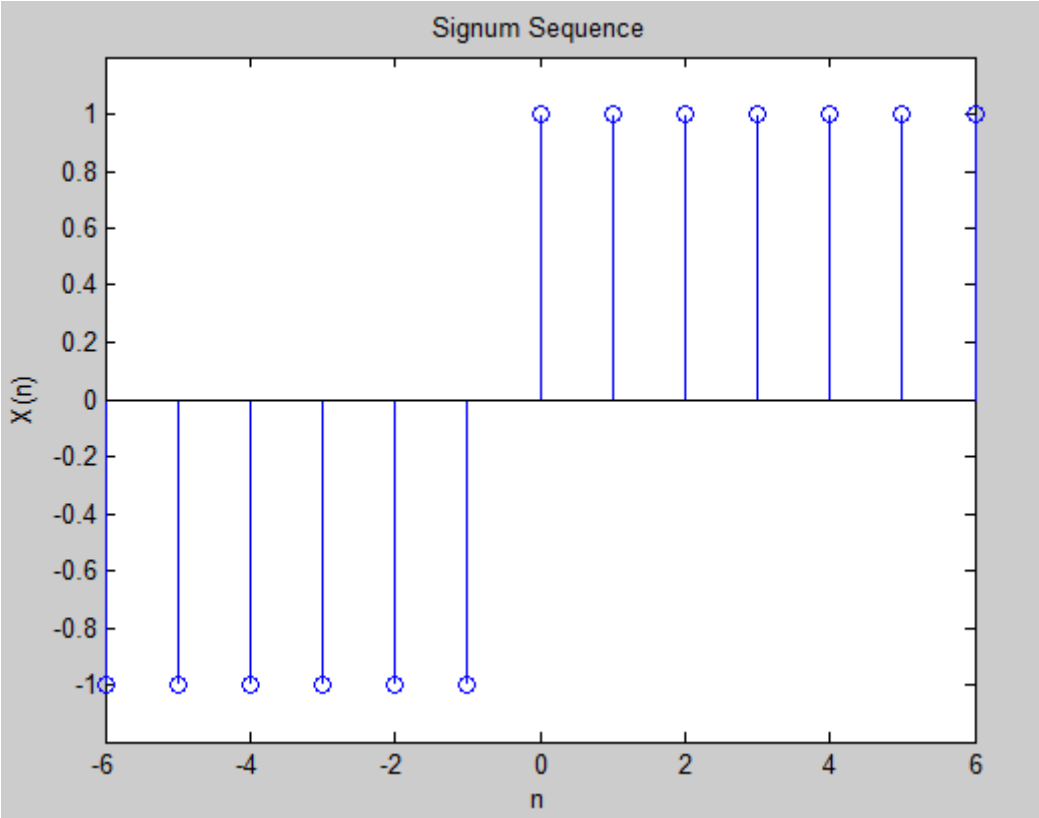


Task 04: Create a function "sigseq", which performs following operations:

Function [x,n]=sigpseq(n0,n1,n2)

- Takes three parameters (n0, n1, n2) as input, where 'n1' and 'n2' are lower and upper limits of n-axis, and 'n0' is the delay.
- Generates a signum sequence using above mentioned three parameters.
- There should be two output arguments [x, n] of function 'sigseq', where 'x' is impulse sequence and 'n' is its corresponding n-axis.
- Finally, plot signum sequence 'x' against vector 'n'.

```
function [x, n] = sigpseg(n0,n1,n2)
n = n1:n2;
x(n>=0) = 1;
x(n<0) = -1;
stem(n-n0, x);
axis([n1, n2, -1.2, 1.2]);
xlabel('n');
ylabel('X(n)');
title('Signum Sequence');
end
```



Task 05: Find E_∞ for the following signal

$$tri(t) = \begin{cases} 1 - |t| & |t| < 1 \\ 0 & |t| \geq 1 \end{cases}$$

```
t = 0:0.0001:10;
tri = 1 - abs(t);
E = trapz(t,tri.^2)


E =

 243.3333
```

Task 06: Find P_∞ for the following signal

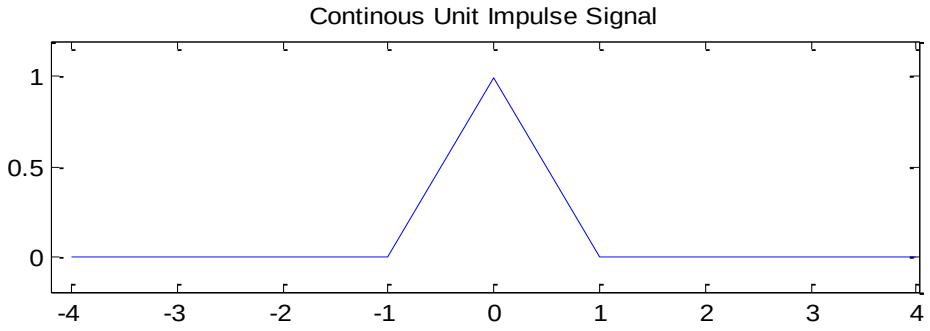
$$x[n] = \cos\left(\frac{\pi}{4}n\right)$$

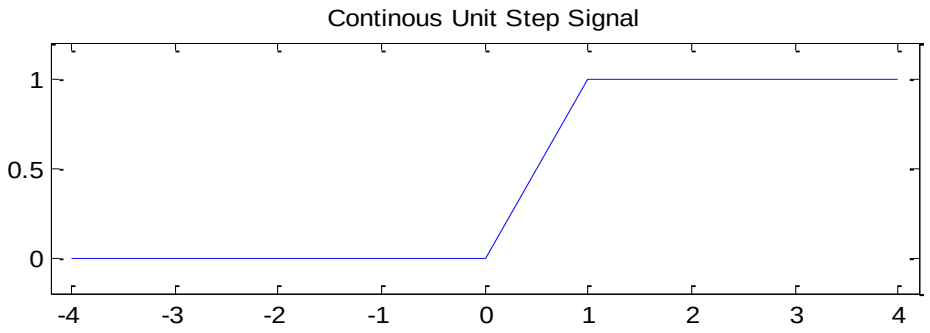
```
n = -100:100;
x = cos(pi/4.*n);
xsquare = x.^2;
s = sum(xsquare/200)


s =

   0.5050
```

Task 07: Write a function which plot or stem a unit impulse and unit step signals. The function takes values for starting and ending value of independent variable, i.e. t and n, and a character for identification of discrete and continuous signal. Finally t plot or stem the function or signal. e.g;
function f_name (arg1 (start) , arg2 (end) , arg3 (D/C))

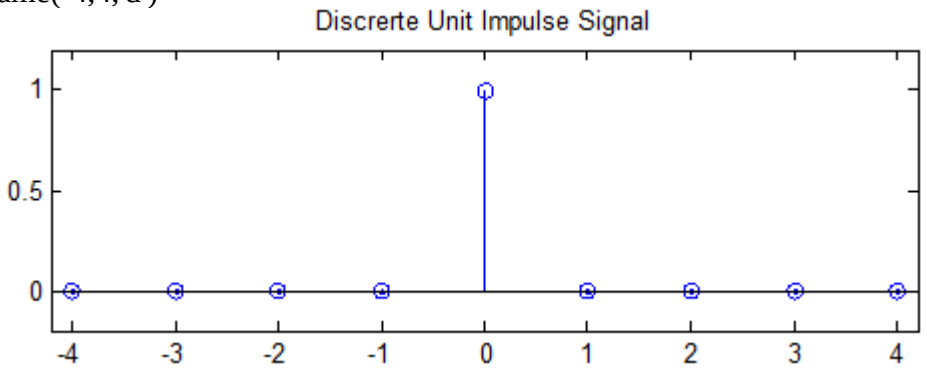
```
function [ y,t,x,n ] = f_name( t1,t2,cn )

if cn=='c'|cn=='C'
   t=t1:t2;
   y=(t==0);
   subplot(2,1,1)
   plot(t,y)
   axis([t1-0.2,t2+.02,-0.2,1.2])
   title('Continous Unit Impulse Signal')

   y=(t>0);
   subplot(2,1,2)
   plot(t,y)
   axis([t1-0.2,t2+0.2,-0.2,1.2])
   title('Continous Unit Step Signal')


else
   n=t1:t2;
   x=(n==0);
   subplot(2,1,1)
   stem(n,x)
   axis([t1-0.2,t2+0.2,-0.2,1.2])
   title('Discrerte Unit Impulse Signal')

   x=(n>0);
   subplot(2,1,2)
   stem(n,x)
   axis([t1-0.2,t2+0.2,-0.2,1.2])
   title('Discrete Unit Step Signal')
end

>> f_name(-5,5,'c')
```

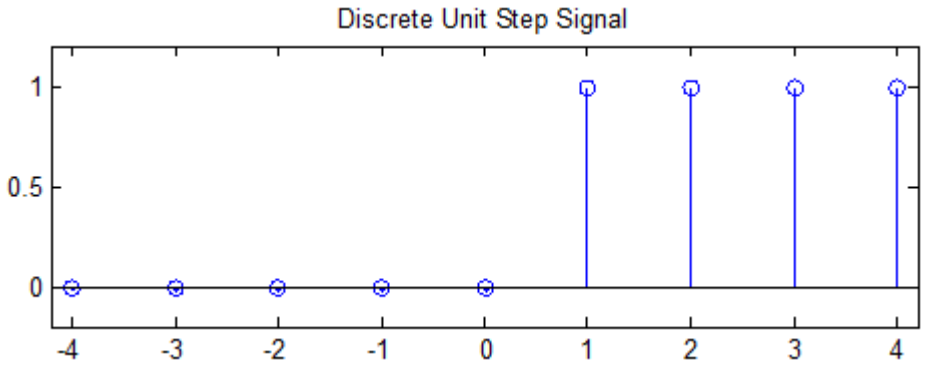
0 0 0 0 0 0 1 1 1 1 1





>> f_name(-4,4,'d')





Post-lab Task

Critical Analysis / Conclusion

In this lab, we studied about some basic signals. These include impulse sequence, Unit step sequence, ramp sequence and sigma sequence. We learnt how to plot these signals using MATLAB. We also observed the change in signals by changing the conditions provided to the signal.

Lab Assessment		
Pre-Lab	**/1**	**/10**
In-Lab	**/5**	
Critical Analysis	**/4**	
Instructor Signature and Comments		

LAB # 04

Signal Transformations (Scaling, Shifting and Reversal)

Lab 04 - Signal Transformations (Scaling, Shifting and Reversal)

Pre-lab Tasks

4.1 Transformations of the Time Variable for Continuous-Time Signals:

In many cases, there are signals related to each other with operations performed in the independent variable, namely, the time. In this lab, we will examine the basic operations that re performed on the independent variable.

4.1.1 Time Reversal or Reflection:

The first operation performed is the signal's reflection. A signal $y(t)$ is a reflection or a reflected version of $x(t)$ about the interval axis if $y(t) = x(-t)$.

The operation of time reversal is actually an alteration of the signal values between negative and positive time. Assume that x is the vector that denotes the signal $x(t)$ in time t. The MATLAB statement that plots the reflected version of $x(t)$ is **plot(-t,x).**

Example

Suppose that $x(t) = te^{-t}, -1 \le t \le 3.$ Plot the signal $x(-t)$.

```
t=-1:0.1:3;
x=t.*exp(-t);
subplot(2,1,1)
a=2;b=1/2;
plot(t,x,'-o','Linewidth',2),grid on
legend('x(t)')
subplot(2,1,2)
plot(-t,x,'-o','Linewidth',2),grid on
legend('x(-t)')
```

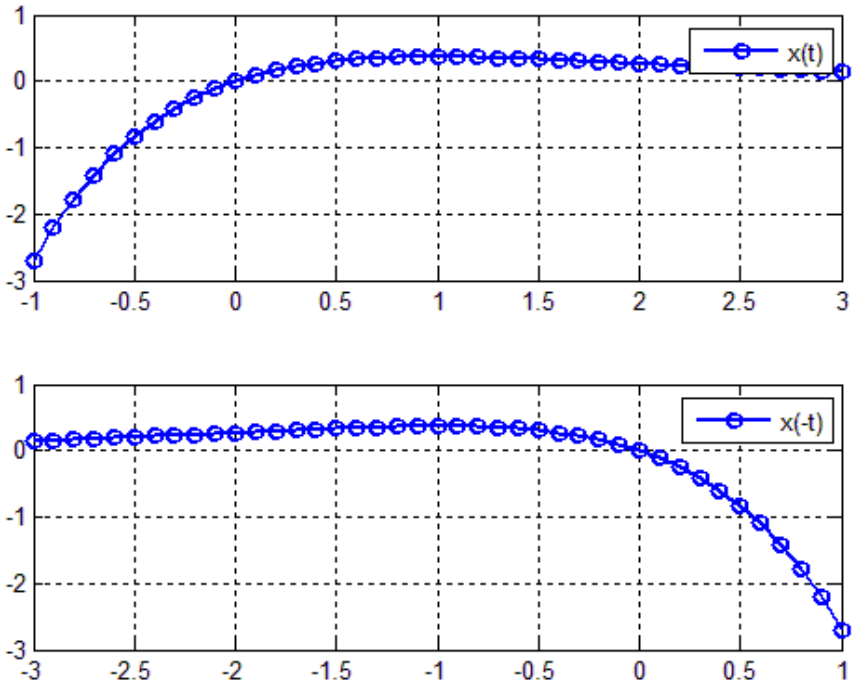


Figure 4.1: Graphs of original $x(t)$ and $x(-t)$

In order to draw $x(-t)$, we just to use the command **plot(-t,x).** The graph of $x(-t)$ is given in Figure 4.1 bottom.

4.1.2 Time Scaling:

The second operation discussed is time scaling. A signal $x_1(t)$ is a compressed version of $x(t)$ if $x_1(t) = x(at), a > 1$. The time compression practically means that the time duration of the signal is reduced by a factor of a. On the other hand, a signal $x_2(t)$ is an expanded version of $x(t)$ if $x_2(t) = x(at), 0 < a < 1$. In this case, the time duration of the signal is increased by a factor of $1/a$.

In order to plot in MATLAB a time-scaled version of $x(t)$, namely, a signal of the form $y(t) = x(at)$, the statement employed is **plot ((1/a)*t,x).** In contrast to what, someone would expect the vector of time t must be multiplied by $1/a$ and not a.

Example

Consider again the continuous time signal $x(t) = te^{-t}, -1 \le t \le 3$. We will plot the signal $x_1(t) = x(2t)$, which is a time compression of $x(t)$ by a factor of $a = 2$; and the signal $x_2(t) = x(0.5t)$, which is a time expansion of $x(t)$ by a factor $1/a = 2$.

```
t=-1:0.1:3;
x=t.*exp(-t);
subplot(3,1,1)
a=2;b=1/2;
plot(t,x,'-o','Linewidth',2),grid on
legend('x(t)')
subplot(3,1,2)
plot((1/a)*t,x,'-o','Linewidth',2),grid on
legend('x(2t)')
subplot(3,1,3)
```

```
plot((1/b)*t,x,'-o','Linewidth',2),grid on
legend('x(0.5t)')
```

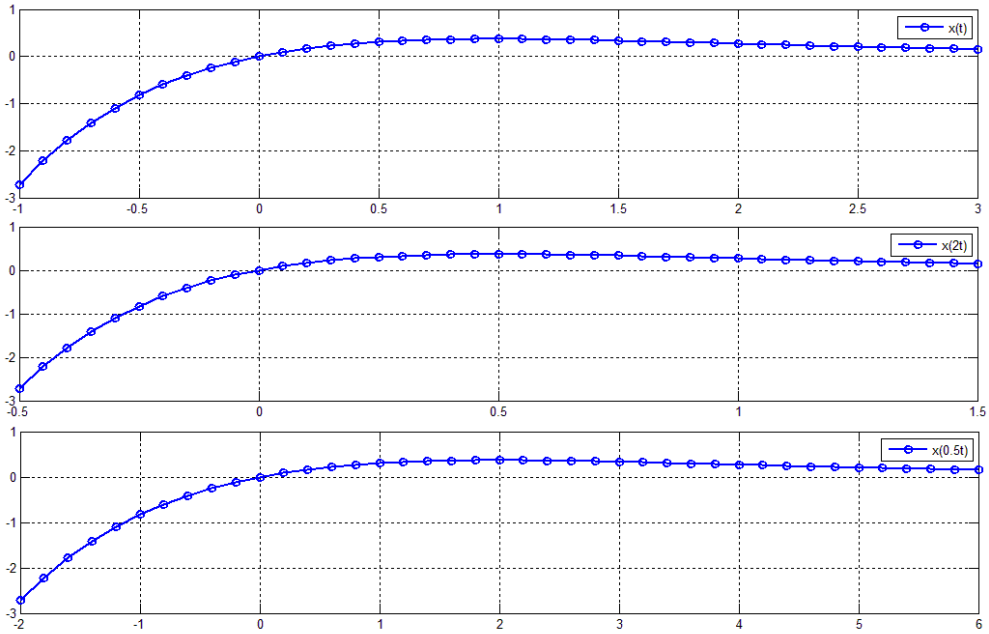


Figure 4.2: Graphs of $x(t), x(2t), x(0.5t)$

4.1.3 Time Shifting:

A third operation performed on time is one of time shifting. A signal $y(t)$ is a time shifted version of $x(t)$ if $y(t) = x(t - t_0)$, where t_0 is the time shift. If $t_0 > 0$, the signal $y(t) = x(t - t_0)$ is shifted by t_0 units to the right (i.e. towards $+\infty$); while if $t_0 < 0$, the signal $y(t) = x(t - t_0)$ is shifted by t_0 units to the left (i.e. towards $-\infty$). The time shifting is implemented in MATLAB in an opposite way to what may be expected. More specifically, in order to plot $x(t - t_0)$ the corresponding MATLAB statement is **plot(t+t0,x).**

Example

The signal $x(t) = te^{-t}, -1 \le t \le 3.$ is again considered. We will plot the signal $x_1(t) = x(t - 2)$, that is a shifted version of $x(t)$ by two units to the right (here $t_0 = 2$) and $x_2(t) = x(t + 3) = x(t - (-3))$, that is, a shifted version of $x(t)$ by 3 units to the left (here $t_0 = -3$).

```
t=-1:0.1:3;
x=t.*exp(-t);
t0=2;
t1=-3;
subplot(3,1,1)
plot(t,x,'-o','Linewidth',2),grid on
legend('x(t)')
subplot(3,1,2)
plot(t+t0,x,'-o','Linewidth',2),grid on
legend('x(t-2)')
subplot(3,1,3)
plot(t+t1,x,'-o','Linewidth',2),grid on
legend('x(t+3)')
```

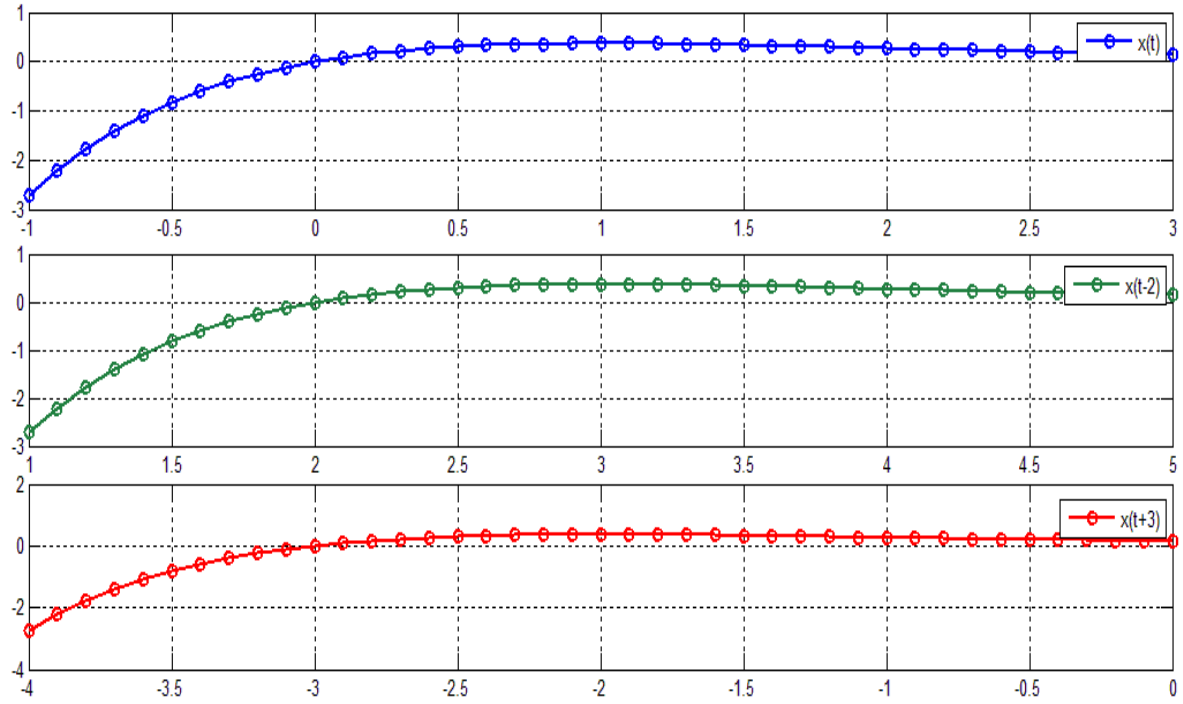


Figure 4.3: Graphs of $x(t), x(t-2), x(t+3)$

The order (shifting-scaling-reversal) must be strictly followed in order to correctly plot a signal. The fact that entire expression (which is the first argument of the command plot) is multiplied by scaling factor and afterward reversed is easily explained if we consider that it represents the time interval in which the signal is plotted.

4.2 Transformations of the Time Variable for Discrete-Time Signals:

Suppose that $x[n]$ is a discrete time signal defined over the time interval specified by n and n_0 is an integer number. The three transformations for the time variable for discrete time signals are the following:

- Time Shifting. This operation is similar to the continuous time operation. More specifically, the signal $y[n] = x[n-n_0]$ is shifted by n_0 units (or samples) to the right if $n_0 > 0$, and is shifted by n_0 units to the left if $n_0 < 0$. The associated MATLAB statement is **stem(n+n0,x).**

- Time Reversal is also similar to operation performed for the continuous-time signals. It is described by the relationship $y[n] = x[-n]$, where $y[n]$ is the reflected (about the vertical axis) signal. Time reversal is implemented in MATLAB by typing **stem(-n,x).**

- Time scaling. This transformation however is somehow different from the one described in the continuous time case. The relationship the time scaling operation is $y[n] = x[an]$. If $a > 1$ and $\in \mathbb{Z}$, the time scaling operation is called downsampling or decimation. Notice that a must be an integer as $x[n]$ is defined for fractional values of n. The downsampling operation results in time compression of the signal. Moreover, some samples of $x[n]$ are lost. The downsampling operation is implemented in MATLAB by using the command y=downsample(x,a)or the statement y=x(1:a:end). The variable end represents the last index in an indexing expression. If $0 < a < 1$, the signal $y[n] = x[an]$ is a time expanded version of $x[n]$. In this case, $(1/a) - 1$ zeros are inserted between two consecutive samples

of $x[n]$. The time expression operation is called upsampling, and is implemented in MATLAB command y=upsample(x,1/a) or with the statement y(1:1/a:end)=x.

Remarks

There are some limitations on the values that a can take. In case of downsampling a must be a positive integer, and in case of upsampling $1/a$ must be a positive integer.

The downsampling and the upsampling processes are illustrated in the next example for the discrete time signal $x[n]=[1,2,3,4,5,6], 0 \le n \le 5$.

Commands	Results	Comments
x=[1 2 3 4 5 6]	x=1 2 3 4 5 6	The discrete time signal $x[n]=[1,2,3,4,5,6], 0 \le n \le 5$
a=2; **xds=downsample(x,a)**	xds=1 3 5	The downsampled version of $x[n]$
xds=x(1:a:end)	xds=1 3 5	Alternative computation of the downsampled signal.
a=1/2; **xups=upsample(x,1/a)**	xups=1 0 2 0 3 0 4 0 5 0 6 0	Upsampling operation on $x[n]$
xups=zeros(1,1/a*length(x)) **xups(1:1/a:end)=x**	xups=1 0 2 0 3 0 4 0 5 0 6 0	Upsampling operation performed in an alternative way. Notice that the variable in which the upsampled signal is stored must be first defined as a vector of zeros with length $(1/a)$ .length $(x[n])$

Example

Consider the sequence $x[n]=0.9^n, -10 \le n \le 10$. Plot the sequences $x[n-10]$, $x[n+10]$, $x[3n]$, $x[n/3]$, and $x[-n]$.

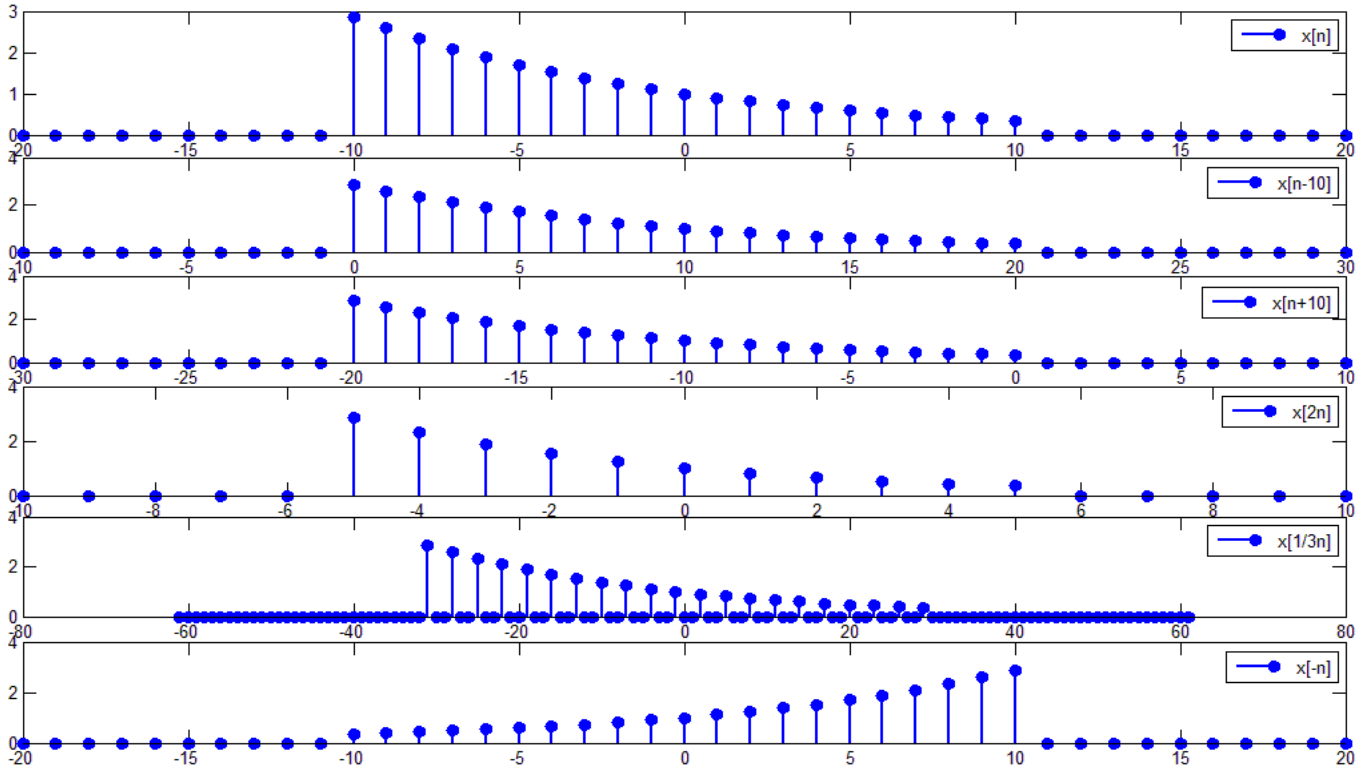


Figure 4.4: Graphs of $x[n], x[n-10], x[n+10], x[3n], x\left[n/3\right]$, and $x[-n]$

An alternative way to obtain the signal $x[-n]$ is by using the command fliplr. This command flips the order of the vector elements. To demonstrate its use, the signal $x[n]=0.9^n, -2 \le n \le 4$ is considered.

```
n=-2:4; %n= -2 -1 0 1 2 3 4
n1=-fliplr(n); %n1=-4 -3 -2 -1 0 1 2
x=0.9.^n; %x= 1.23 1.11 1.00 0.90 0.81 0.73 0.66
x1=fliplr(x); %x1=0.66 0.73 0.81 0.90 1.00 1.11 1.23
subplot(1,2,1)
stem(n,x,'fill','linewidth',2)
title('x[n]=0.9.^n')
subplot(1,2,2)
stem(n1,x1,'fill','linewidth',2)
title('x[-n]')
```

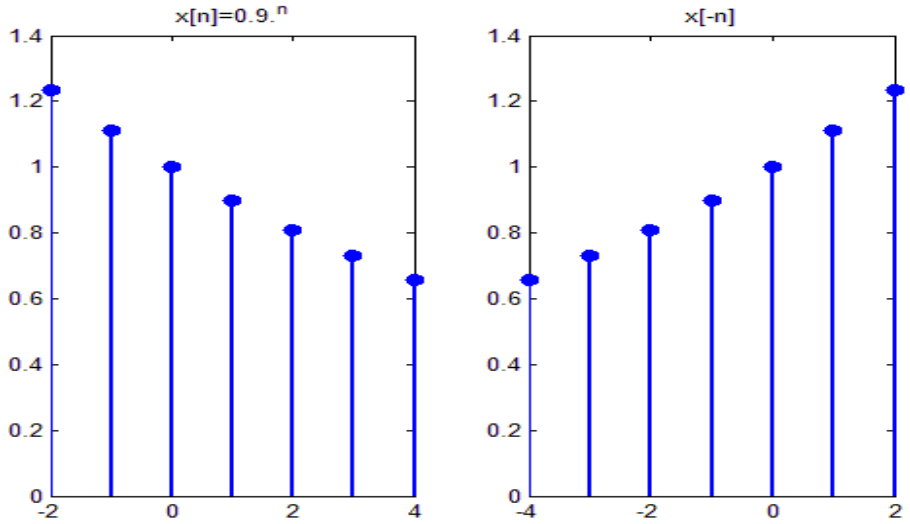
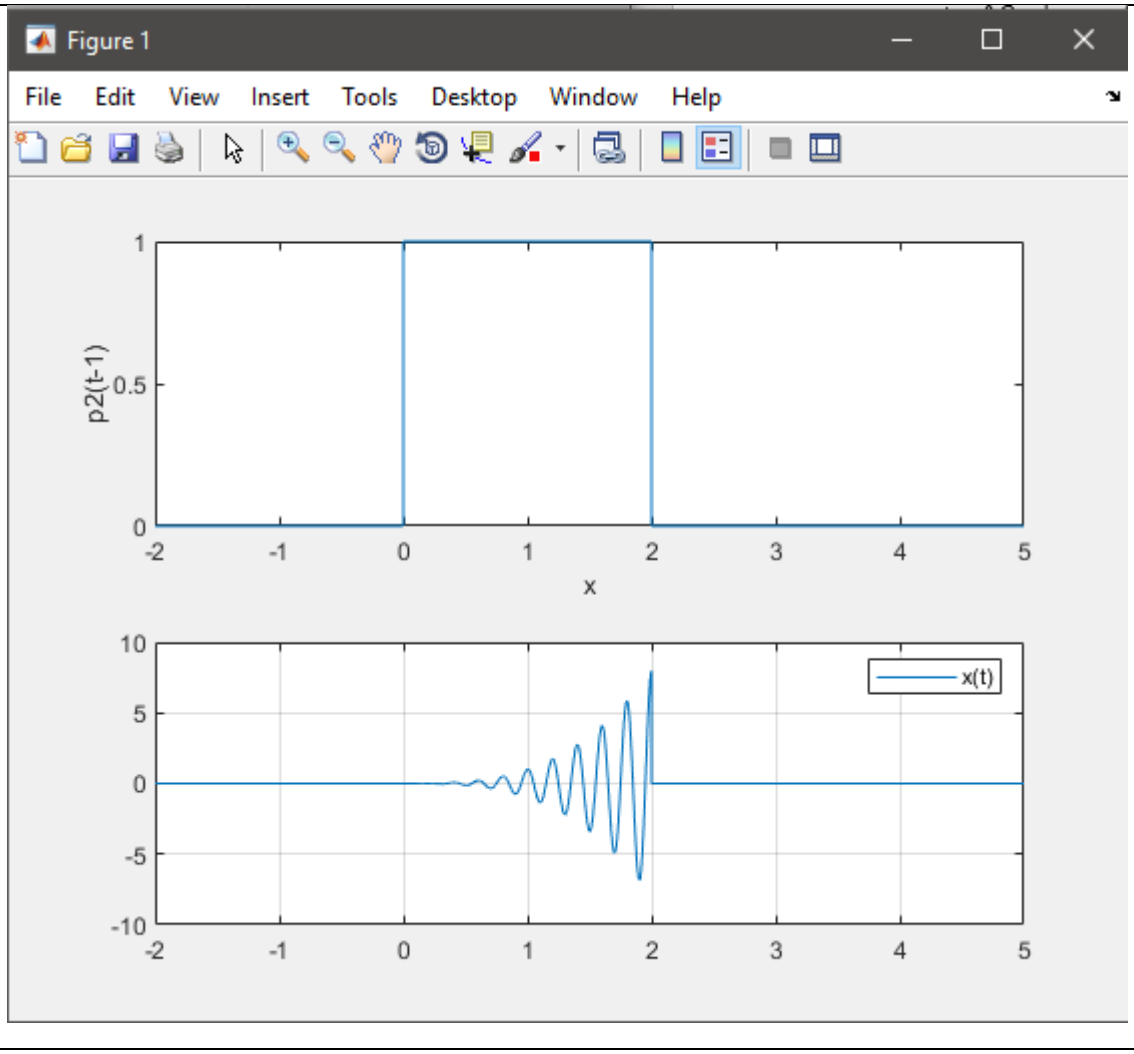


Figure 4.5: Graphs of $x[n]$ and $x[-n]$

In-lab Tasks

Task 01: Plot the signal $x(t) = t^3 \cos(10\pi t)\, p2(t-1)$ for $-2 \le 5$, where $pT(t)$ is a rectangular pulse of duration T, denoted by $p2(t-1) = u(t-1+2/2) - u(t-1-2/2) = u(t) - u(t-2)$.

```
t = -2:0.001:5;
u1(t>=0) = 1;
u2(t>=2) = 1;
u = u1 - u2;
subplot(2,1,1)
plot(t,u)
xlabel('x');
ylabel('p2(t-1)');
x = t.^3.*cos(10*pi*t).*u;
subplot(2,1,2)
plot(t,x)
legend('x(t)');
grid on
```



Task 02: Suppose that $x(t) = t\cos(2\pi t), 0 \le t \le 5.$ **Plot the signals**

(a) $x(t)$

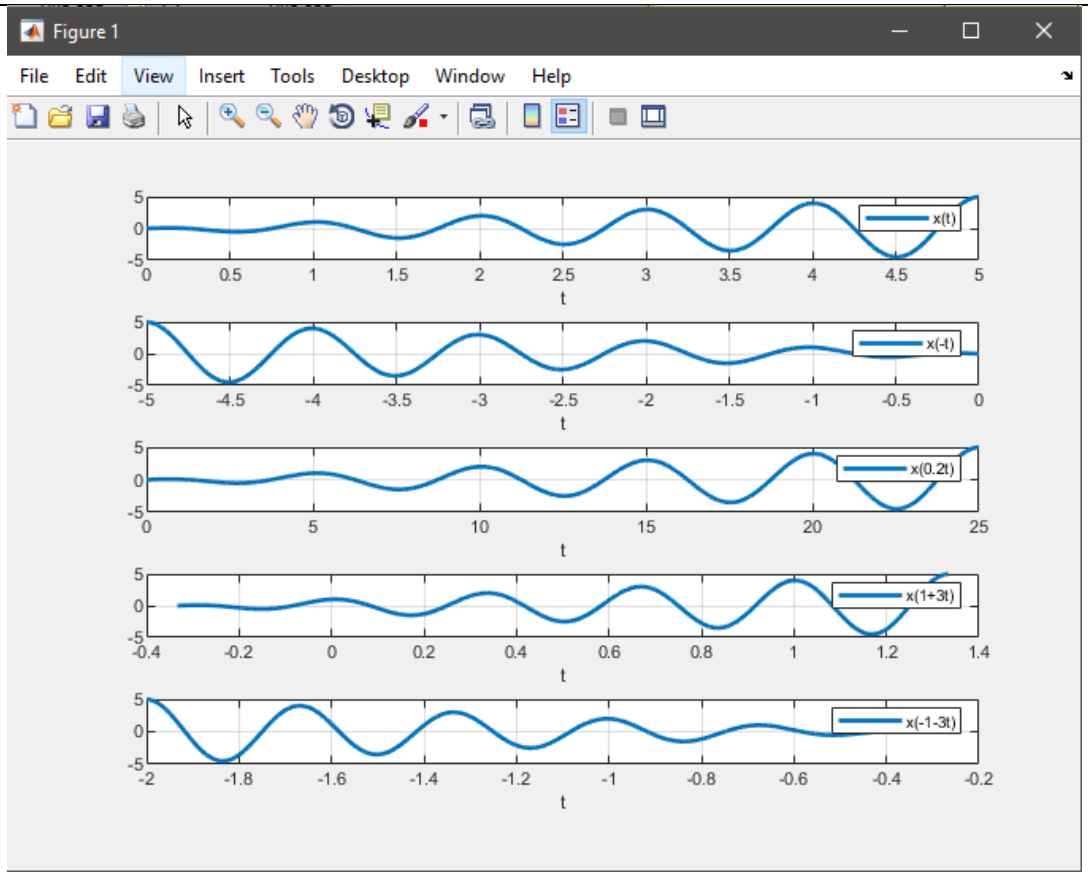
(b) $x(-t)$

(c) $x(t/5)$

(d) $x(1+3t)$

(e) $x(-1-3t)$

```
t = 0:0.0001:5;
x = t.*cos(2*pi*t);
subplot(5,1,1)
plot(t,x,'LineWidth',2)
grid on
xlabel('t');
legend('x(t)')


subplot(5,1,2)
plot(-t,x,'LineWidth',2)
grid on
xlabel('t');
legend('x(-t)')


a = 1/5;
subplot(5,1,3)
plot((1./a).*t,x,'LineWidth',2)
grid on
xlabel('t');
legend('x(0.2t)')


subplot(5,1,4)
plot((1/3).*(t-1),x,'LineWidth',2)
grid on
xlabel('t');
legend('x(1+3t)')


subplot(5,1,5)
plot((-1/3).*(t+1),x,'LineWidth',2)
grid on
xlabel('t');
legend('x(-1-3t)')
```

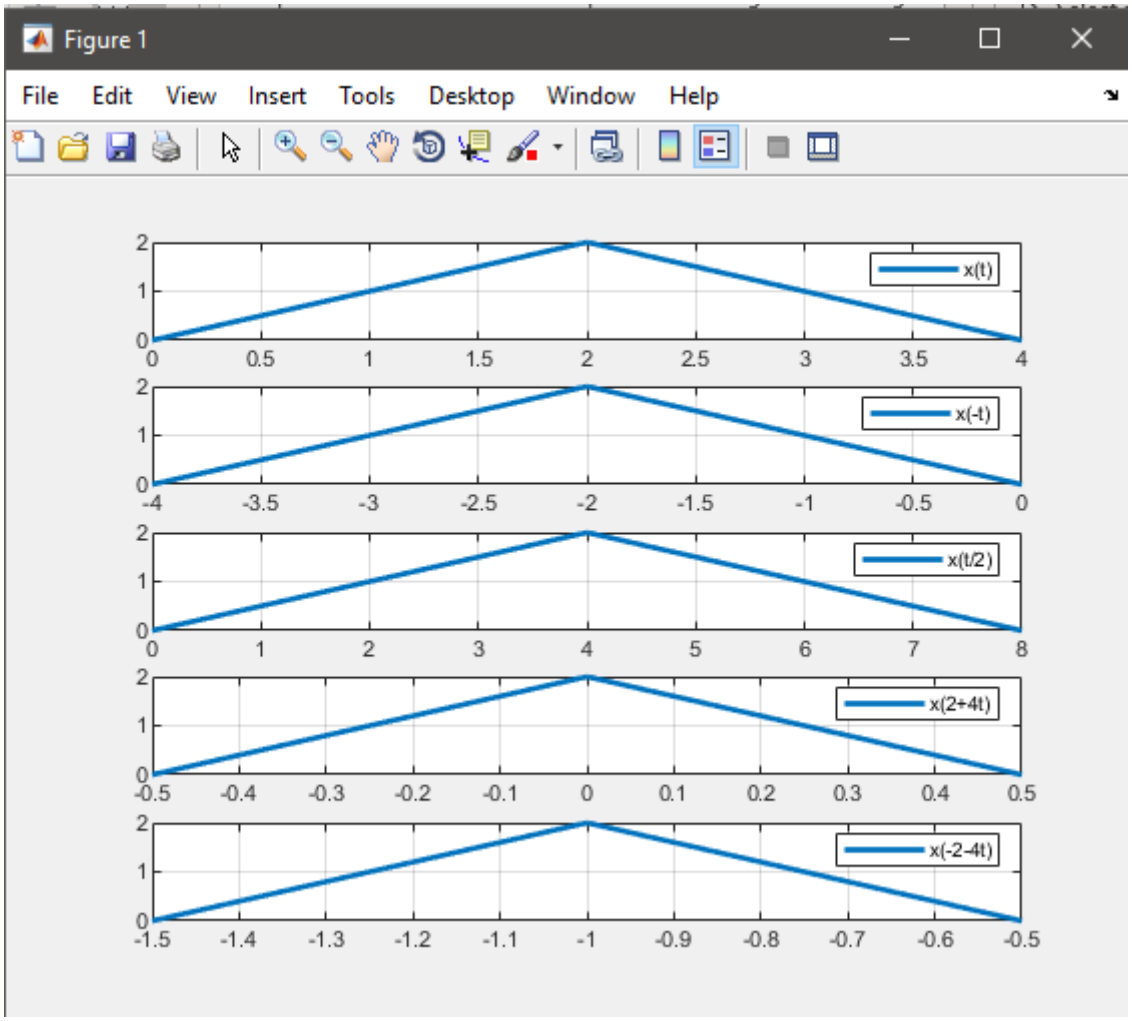


Task 03: Suppose that $x(t)=\begin{cases} t & 0\le t\le 2 \\ 4-t & 2<t\le 4 \end{cases}$**. Plot the signals**

(f) $x(t)$
(g) $x(-t)$
(h) $x(t/2)$
(i) $x(2+4t)$
(j) $x(-2-4t)$

```
t = 0:0.01:4;
x = t.*(t<=2) + (4-t).*(t>2);
subplot(5,1,1)
plot(t,x,'LineWidth',2)
grid on
legend('x(t)')


subplot(5,1,2)
plot(-t,x,'LineWidth',2)
grid on
legend('x(-t)')


subplot(5,1,3)
```

```
a = 1/2;
plot((1/a).*t,x,'LineWidth',2)
grid on
legend('x(t/2)')


subplot(5,1,4)
b = 4;
plot((1/b).*(t-2),x,'LineWidth',2)
grid on
legend('x(2+4t)')


subplot(5,1,5)
c = -4;
plot((1/c).*(t+2),x,'LineWidth',2)
grid on
legend('x(-2-4t)')
```



Task 04: Write a function that accepts a sequence $x[n]$, the discrete time n and a number n_0, a, b as input arguments, and returns the signals $x[n-n_0], x[-n], x[an]$ and $x[bn]$. Where $x[an]$ represents the time compressed version of $x[n]$ and $x[bn]$ is the time expanded version of $x[n]$.

```
function [  ] = sequence(x,n,n0,a,b)


subplot(5,1,1);
stem(n,x,'LineWidth',2);
grid on
legend ('x[n]');


subplot(5,1,2);
stem(n+n0, x,'LineWidth', 2);
grid on
legend('x[n-n0]')


subplot(5,1,3);
stem(-n , x,'LineWidth', 2);
grid on
legend('x[-n]')


x1=downsample(x,a);
n1=n(1:a:end);
subplot(5,1,4);
stem(n1,x1,'LineWidth',2);
grid on
legend('x[an])');


x2=upsample(x,b);
i=(length(x2)-1)/2;
n2=-i:i;
subplot(5,1,5);
stem(n2,x2,'LineWidth',2);
grid on
legend('x[bn]');


>> n = -3:3;

>> x = (n>=0);

>> sequence(x,n,1,2,3)
```

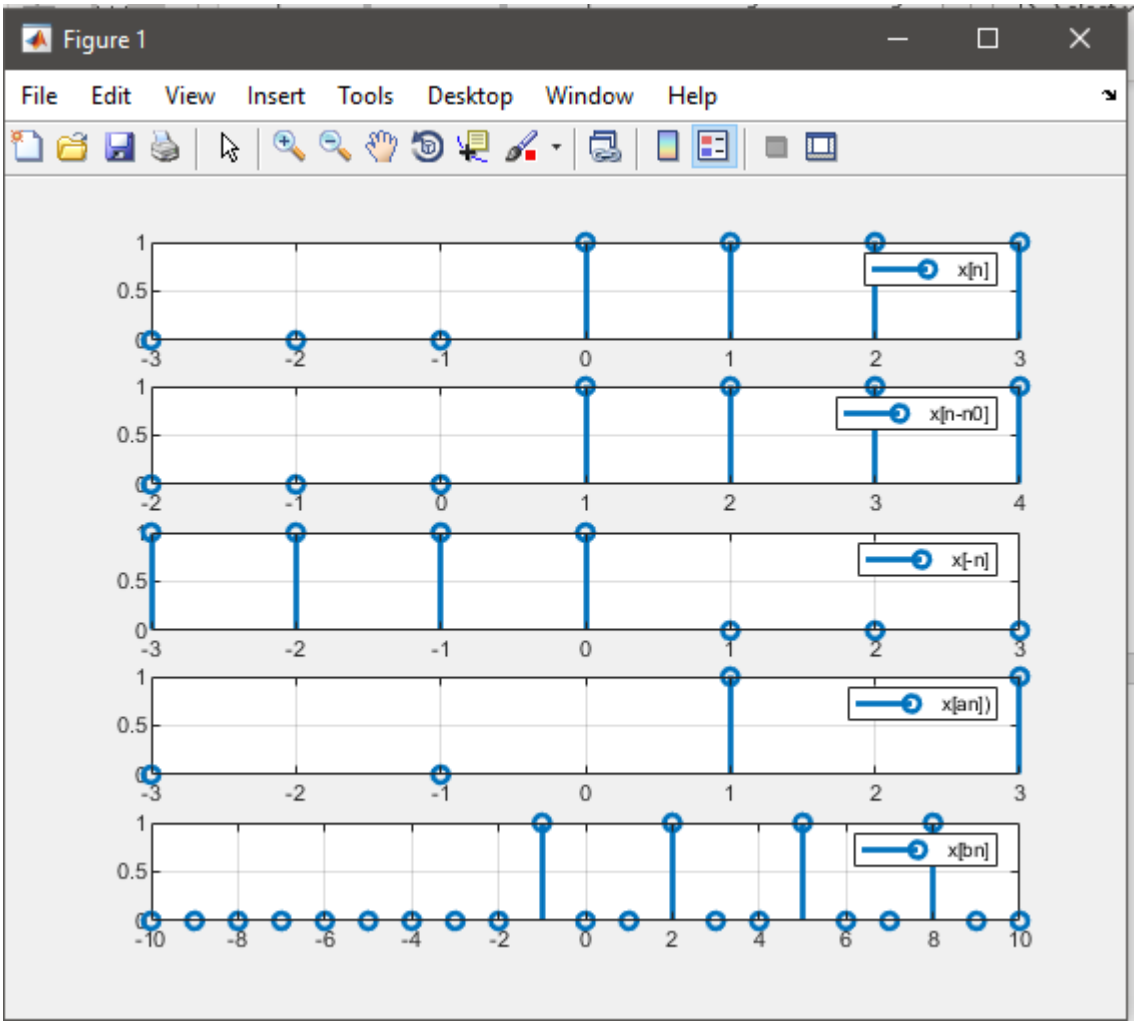


Figure 1
File Edit View Insert Tools Desktop Window Help
x[n]
x[n-n0]
x[-n]
x[an])
x[bn]

Post-lab Task

Critical Analysis / Conclusion

In this lab, we learned to perform operations which includes reversing, shifting, and scaling on both continuous and discrete signals. We also learned how to influence a single function by sending a complete wave signal through it and generating a series of signals.

Lab Assessment		
Pre-Lab	**/1**	**/10**
In-Lab	**/5**	
Critical Analysis	**/4**	
Instructor Signature and Comments		

LAB # 05

Study of Properties of Systems (Linearity, Causality, Memory, Stability and Time invariance)

Lab 05- Study of Properties of Systems (Linearity, Causality, Memory, Stability and Time invariance)

Pre-Lab Tasks

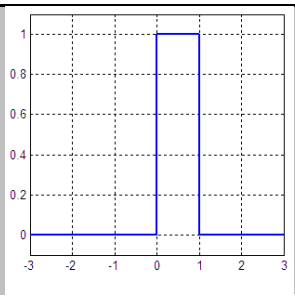
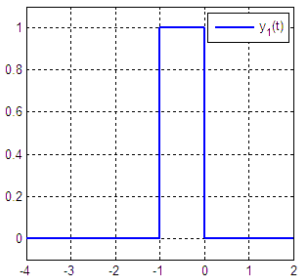
5.1 Properties of Systems:

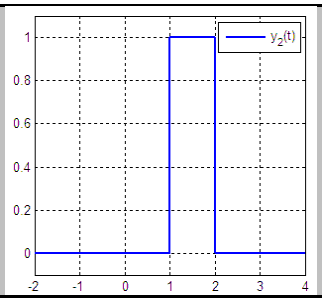
5.1.1 Causal and Non-causal Systems:

A system is causal if the system output $y(t_0)$ at time $t = t_0$ does not depend on values of the input $x(t)$ for $t > t_0$. In other words, for any input signal $x(t)$, the corresponding output $y(t)$ depends upon the present and past values of $x(t)$. So, if the input to a causal system is zero for $t < t_0$, the output of the system is also zero for $t < t_0$. Correspondingly a discrete time system is causal if its output $y[n_0]$ at time $n = n_0$ depends only on the values of input signal $x[n]$ for $n \le n_0$. All natural systems are causal.

Example:

Suppose the system S_1 is described by the I/O relationship $y(t) = x(t+1)$ while the I/O relationship of the system S_2 is given by $y(t) = x(t-1)$. Using the input signal $x(t) = u(t) - u(t-1)$ find out if the two systems are causal.

Commands	Results	Comments
t1=-3:0.1:0; **x1=zeros(size(t1));** **t2=0:0.1:1;** **x2=ones(size(t2));** **t3=1:0.1:3;** **x3=zeros(size(t3));** **t=[t1 t2 t3];** **x=[x1 x2 x3];** **plot(t,x,'linewidth',2),grid on** **ylim([-0.1 1.1]);** **legend('x(t)')**		Definition of the graph in the time interval $-3 \le t \le 3$ of the input signal $x(t) = u(t) - u(t-1)$
plot(t-1,x,'linewidth',2),grid on **ylim([-0.1 1.1]);** **legend('y_1(t)')**		The output of S_1 is given by $y(t) = x(t+1)$. The input signal $x(t)$ is zero for $t < 0$ but the output $y(t)$ is nonzero for $t < 0$, i.e., $y(t)$ depends upon the future values of $x(t)$; thus system S_1 is non-causal

plot(t+1,x,'linewidth',2),grid on ylim([-0.1 1.1]); legend('y_2(t)')		The output of S_2 is given by $y(t)=x(t-1)$. The output $y(t)$ is zero for $t<1$, i.e., $y(t)$ depends only on the past values of $x(t)$; thus system S_2 is causal

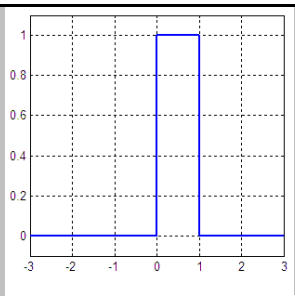
5.1.2 Static (Memory less) and Dynamic (with Memory) Systems:

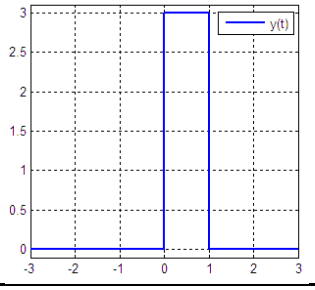
A system is static or memory less if for any input signal $x(t)$ or $x[n]$ the corresponding output $y(t)$ or $y[n]$ depends only on the value of the input signal at that time. A non-static system is called dynamic or dynamical.

Example:

Using the input signal $x(t)=u(t)-u(t-1)$ find out the systems described by the I/O relationship $y(t)=3x(t)$ and $y(t)=x(t)-x(t-1)$ are static or dynamic.

Commands	Results	Comments

t1=-3:0.1:0;
x1=zeros(size(t1));
t2=0:0.1:1;
x2=ones(size(t2));
t3=1:0.1:3;
x3=zeros(size(t3));
t=[t1 t2 t3];
x=[x1 x2 x3];
plot(t,x,'linewidth',2),grid on
ylim([-0.1 1.1]);
legend('x(t)')
		Definition of the graph in the time interval $-3\le t\le 3$ of the input signal $x(t)=u(t)-u(t-1)$

plot(t,3*x,'linewidth',2),grid on ylim([-0.1 3.1]); legend('y(t)')		The output of the system when I/O relationship $y(t)=3x(t)$ depends on the value of the input at the same time. Hence, it is a static (or memory less) system

In order to determine if the second system described by the I/O relationship $y(t)=x(t)-x(t-1)$ is static or dynamic, recall that $x(t)=u(t)-u(t-1)=1, 0\le t\le 1$; thus $x(t-1)=u(t-1)-u(t-2)=1, 1\le t\le 2$ and so $y(t)=u(t)-u(t-2)=1, 0\le t\le 2$. The values of $y(t)$ depend on past values of $x(t)$ so system is dynamic.

5.1.3 Linear and Non-linear Systems:

Let $y(t)$ denote the response of the system S to an input signal $x(t)$, that is, $y(t)=S\{x(t)\}$. System S is linear if for any input signal s $x_1(t)$ and $x_2(t)$ and any scalar a_1 and a_2 the following relationship (equation 5.1) holds.

$$S\{a_1x_1(t)+a_2x_2(t)\}=a_1S\{x_1(t)\}+a_2S\{x_2(t)\}$$

In other words, the response of the linear system to an input that is a linear combination of two signals is the linear combination of the responses of the system to each one of these signals. The linearity property is generalized for any number of input signals, and this is often referred to as the principle of superposition. The linearity property is the combination of two other properties: the additivity property and the homogeneity property. A system S satisfies the additivity property if for any input signals $x_1(t)$ and $x_2(t)$

$$S\{x_1(t)+x_2(t)\}=S\{x_1(t)\}+S\{x_2(t)\}$$

While the homogeneity property implies that for any scalar a and any input signal $x(t)$,

$$S\{ax(t)\}=aS\{x(t)\}$$

Example:

Let $x_1(t)=u(t)-u(t-1)$ and $x_2(t)=u(t)-u(t-2)$ be the input signals to the systems described by the I/O relationships $y(t)=2x(t)$ and $y(t)=x^2(t)$. Determine if the linearity property holds for these two systems.

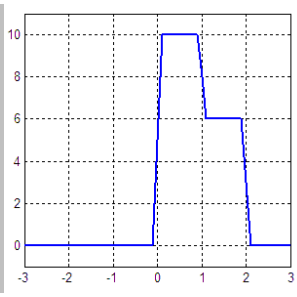
To examine if the systems are linear, we use the scalars $a_1=2$ and $a_2=3$. The time interval considered is $-3\le t\le 3$.

For the system described by the I/O relationship $y(t)=2x(t)$ the procedure is followed as

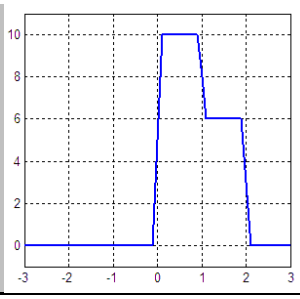
Commands	Results	Comments

t=-3:0.1:3;
x1=heaviside(t)-heaviside(t-1);
x2=heaviside(t)-heaviside(t-2);
		Definition of the input signals $x_1(t)$ and $x_2(t)$

%computation of the left side of equation 5.1
a1=2;
a2=3;
z=a1*x1+a2*x2;
		The expression $a_1x_1(t)+a_2x_2(t)$ is defined.

y=2*z; plot(t,y,'linewidth',2),grid on ylim([-1 11])		The left side of equation 5.1, namely, $S\{a_1x_1(t)+a_2x_2(t)\}$ is computed and the result is plotted.

%computation of the right side of equation 5.1
z1=2*x1;
z3=3*x2;
		Definition of $S\{x_1(t)\}$ and $S\{x_2(t)\}$.

y=a1*z1+a2*z2; plot(t,y,'linewidth',2),grid on ylim([-1 11])		The right side of equation 5.1, namely, $a_1S\{x_1(t)\}+a_2S\{x_2(t)\}$ is computed and the result is plotted.

The two graphs obtained are identical; hence the two sides of equation 5.1 are equal. Therefore the system described by the I/O relationship $y(t) = 2x(t)$ is linear.

5.1.4 Time-Invariant and Time-Variant Systems:

A system is time invariant, if a tune shift in the input signal results in the same time shift in the output signal. In other words, if $y(t)$ is the response of a time-invariant system to an input signal $x(t)$, then the system response to the input signal $x(t-t_0)$ is $y(t-t_0)$. The mathematical expression (equation 5.2) is

$$y(t-t_0) = S\{x(t-t_0)\}$$

Equivalently, a discrete time system is time or (more appropriately) shift invariant if

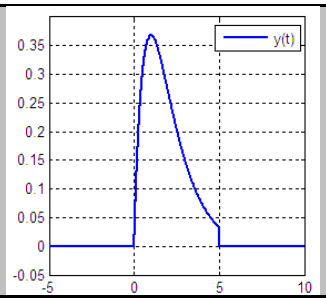
$$y[n-n_0] = S\{x[n-n_0]\}$$

From the above equations, we conclude that is a system is time invariant, the amplitude of the output signal is the same independent of the time instance the input is applied. The difference is time shift in the input signal. A non-time invariant system is called time-varying or time-variant system.

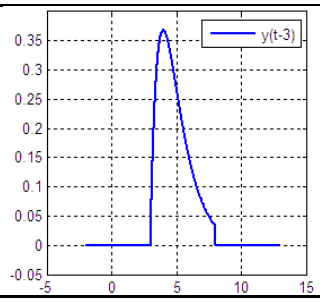
Example:

Suppose that response of a system S to an input signal $y(t) = te^{-t}x(t)$. Determine if the system is time invariant by using the input signal $x(t) = u(t) - u(t-5)$.

In order to determine if the system is time invariant, first we will have to compute and plot the system response $y(t)$ to the given input signal $x(t) = u(t) - u(t-5)$. Next, the computed output is shifted by 3 units to the right to represent the signal $y_1(t) = y(t-3)$. This corresponds to the left side of equation 5.2. As for the right side if equation 5.2, first the input signal $x(t)$ is shifted 3 units to the right in order to represent the signal $x(t-3)$. Next, the system response $y_2(t) = S\{x(t-3)\}$ is computed and plotted. If the two derived system responses are equal, the system under consideration is time invariant.

Commands	Results	Comments
t=-5:0.001:10; **p=heaviside(t)-heaviside(t-5);** **y=t.*exp(-t).*p;** **plot(t,y,'linewidth',2),grid on** **ylim([-0.05 0.4])** **legend('y(t)')**		The response $y(t)$ of the system to the input signal $x(t) = u(t) - u(t-5)$ is $y(t) = te^{-t}\left[u(t) - u(t-5)\right] = te^{-t}$, $0 \le t \le 5$.

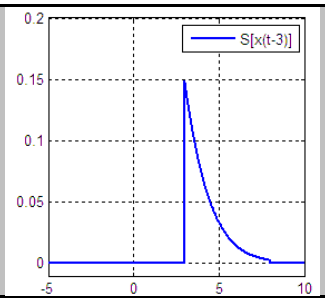
```
plot(t+3,y,'linewidth',2),grid on
ylim([-0.05 0.4])
legend('y(t-3)')
```



The output signal $y(t)$ is shifted 3 units to the right in order to obtain the signal $y_1(t) = y(t-3)$.

The input signal $x_2(t) = x(t-3)$ is given by $x_2(t) = u(t-3) - u(t-8)$ Thus the system response $y_2(t) = S\{x_2(t)\} = S\{x(t-3)\}$ is computed as $y_2(t) = te^{-t}\left[u(t-3) - u(t-8)\right]$.

Commands	Results

t=-5:0.001:10;
p=heaviside(t-3)-heaviside(t-8);
y=t.*exp(-t).*p;
plot(t,y,'linewidth',2),grid on
ylim([-0.01 0.2])
legend('S[x(t-3)]')
	

The two obtained graphs are not alike; thus the system described by the I/O relationship $y(t) = te^{-t}x(t)$ is time variant. A rule of thumb is that if the output of the system depends on time t outside of $x(t)$ the system is time variant.

5.1.5 Invertible and Non-Invertible Systems:

A system is invertible if the input signal $x(t)$ that is applied to the system can be derived from the system response $y(t)$. In other words, a system is invertible if the I/O relationship $y(t) = S\{x(t)\}$ is one to one, namely if different input values correspond to different output values.

Example:

Determine if the systems S_1 and S_2 described by the I/O relationships $y_1[n] = 3x[n]$ and $y_2[n] = x^2[n]$, respectively, are invertible. Consider the signal $x[n] = 2n, -2 \le n \le 2$ as the input signal.

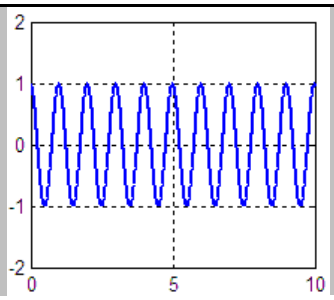
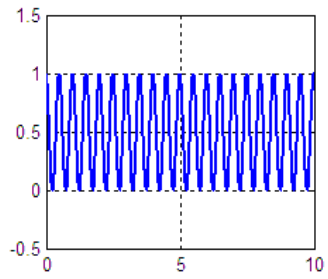
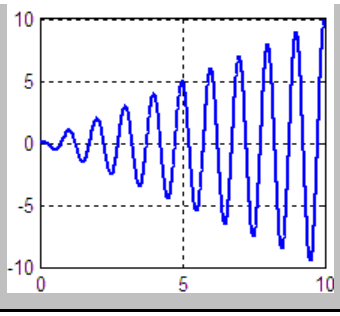
Commands	Results	Comments
n=-2:2; x=2*n;	x= -4 -2 0 2 4	Input signal $x[n]$
y1=3*x;	y1= -12 -6 0 6 12	The output signal $y_1[n] = 3x[n]$ of the system S_1
y2=x.^2;	y2= 16 4 0 4 16	The output signal $y_2[n] = x^2[n]$ of the system S_2

5.1.6 Stable and Unstable Systems:

Stability is very important system property. The practical meaning of a stable system is that for a small applied input the system response is also small (does not diverge). A more formal definition is that a system is stable or bounded-input bounded-output (BIBO) stable if the system response to any bounded-input signal is bounded-output signal. The mathematical expression is as follows: Suppose that a positive number $M < \infty$ exists, such that $|x(t)| \le M$. The system is stable if $\forall t \in$ a positive number $N < \infty$ exists, such that $|y(t)| \le N$. A non-stable system is called unstable system.

Example:

Suppose that input signal $x(t) = \cos(2\pi t)$ is applied to two systems described by the I/O relationships $y_1(t) = x^2(t)$ and $y_2(t) = tx(t)$. Determine if the two systems are stable.

Commands	Results	Comments
1t=0:0.01:10; **x=cos(2*pi*t);** **plot(t,x,'linewidth',2), grid on** **ylim([-2 2])**		Definition and graph of $x(t)$. The input signal is bounded as $-1 \le x(t) \le 1$, namely $x(t)$ is bounded by $M = 1$ as $\|x(t)\| \le M$.
y1=x.^2; **plot(t,y1,'linewidth',2), grid on** **ylim([-0.5 1.5])**		Definition of graph of $y_1(t)$. The out signal $y_1(t)$ is bounded as $0 \le y_1(t) \le 1$, namely $y_1(t)$ is bounded by $N = 1$, as $\|y_1(t)\| \le N$. Hence the system described by the I/O relationship $y_1(t) = x^2(t)$ is BIBO stable.
y2=t.*x; **plot(t,y2,'linewidth',2), grid on**		Definition of the graph of $y_2(t)$. The output signal $y_2(t)$ is not bounded as its amplitude is getting larger as time passes. Hence the system with I/O relationship $y_2(t) = tx(t)$ is not BIBO stable.

In-Lab Tasks

Task 01: Find out if the discrete-time system described by the I/O relationship $y[n] = x[-n]$ is:

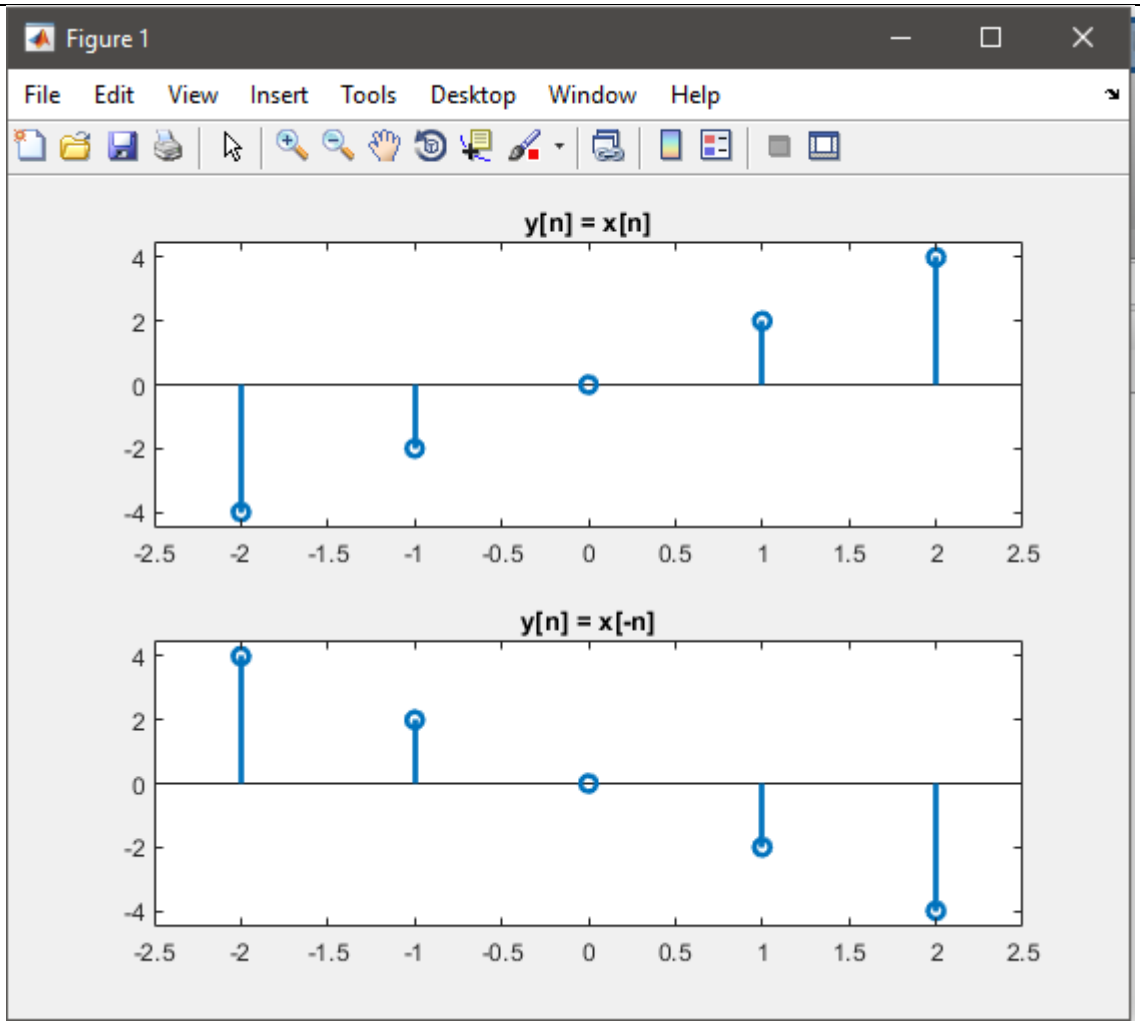
a) Static or Dynamic (input signal $x[n] = 2n, -2 \le n \le 2$)

b) Causal or non-causal (input signal $x[n] = 2n, -2 \le n \le 2$)

c) Linear or non-linear (input signals $x_1[n] = 2n, -2 \le n \le 4, x_2[n] = n/3, -2 \le n \le 4, a_1 = 2 \, and \, a_2 = 3$)

d) Shift invariant or shift variant (input signal $x[n] = 2n, -2 \le n \le 4$ and shift $n_0 = 3$)

```
n = -2:2;
x = 2.*n;
subplot(2,1,1)
stem(n,x,'LineWidth',2)
xlim([-2.5,2.5])
ylim([-4.5,4.5])
title('y[n] = x[n]')

y = -x;
subplot(2,1,2)
stem(n,y,'LineWidth',2)
xlim([-2.5,2.5])
ylim([-4.5,4.5])
title('y[n] = x[-n]')
```

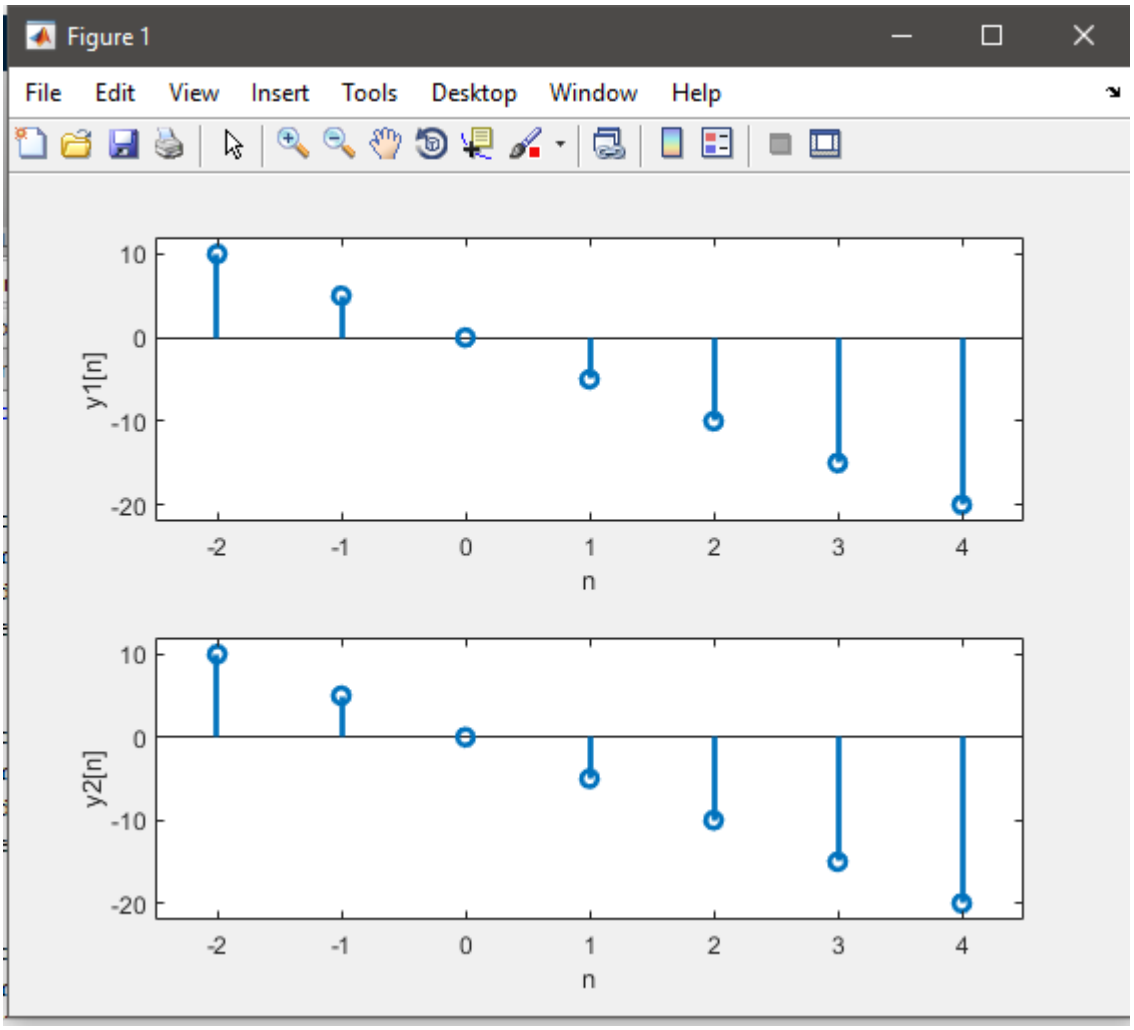


a) The system is dynamic.(with memory)

b) The system is Non-causal as the output depends on the future value of input.

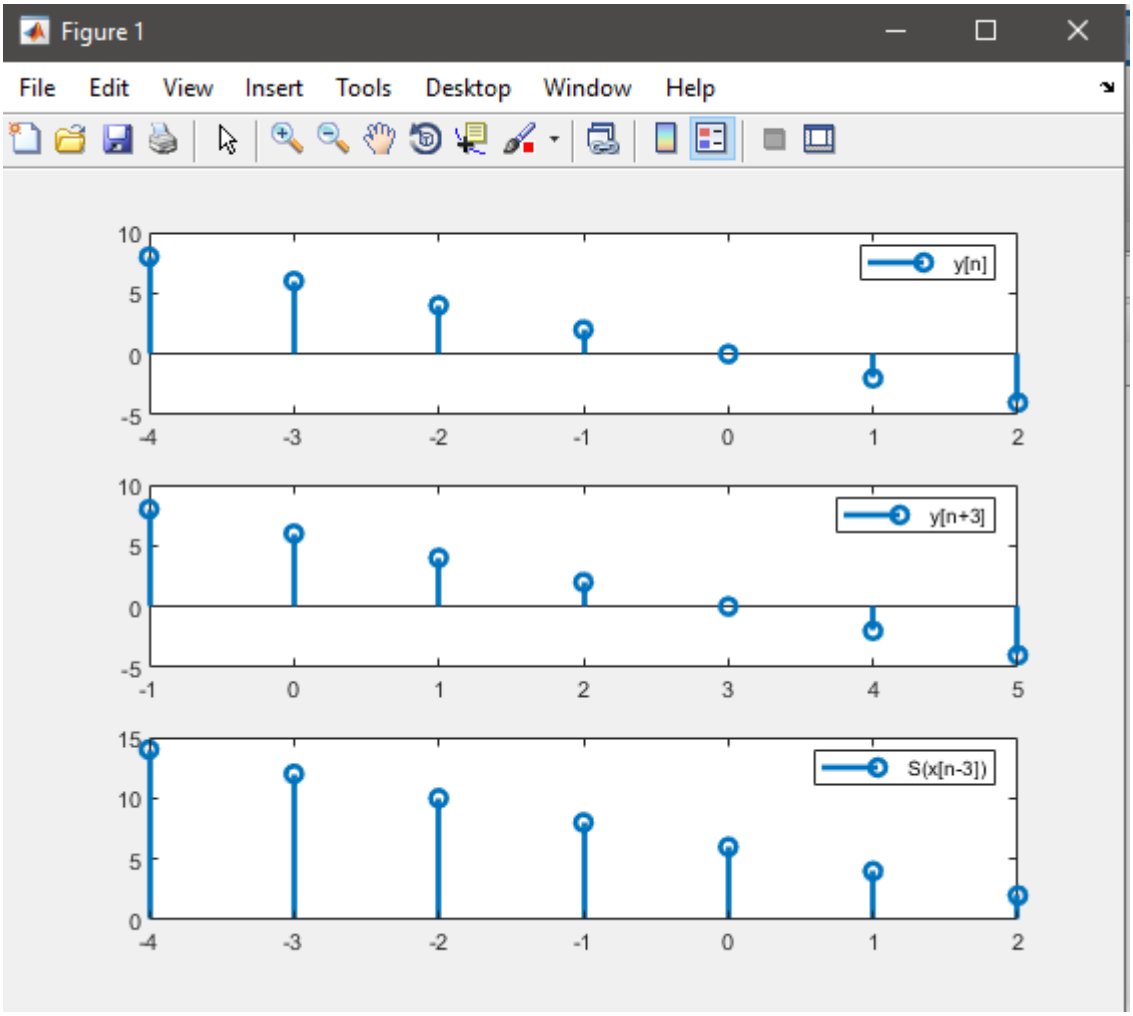
```
N = -2:4;
x1 = 2.*n;
x2 = n./3;
a1 = 2;
a2 = 3;
x = a1.*x1 + a2.*x2;
y = -x;
subplot(2,1,1)
stem(n,y,'LineWidth',2)
xlim([-2.5,4.5])
ylim([-22,12])
xlabel('n')
ylabel('y1[n]')

y1 = -x1;
y2 = -x2;
y = a1.*y1 + a2.*y2
subplot(2,1,2)
stem(n,y,'LineWidth',2)
xlim([-2.5,4.5])
```

```
ylim([-22,12])
xlabel('n')
ylabel('y2[n]')
```



c) As both the graphs are identical, hence the system is said to be linear.(as both the side of equation are equal)

```
n = -2:4;
x = 2.*n;
n0 = 3;
subplot(3,1,1)
stem(-n,x,'LineWidth',2)
legend('y[n]')
subplot(3,1,2)
stem(-(n-3),x,'LineWidth',2)
legend('y[n+3]')
x = 2.*(n+3);
subplot(3,1,3)
stem(-n,x,'LineWidth',2)
legend('S(x[n-3])')
```



d) as the graphs are not alike hence system is time variant

Task 02: Find out if the discrete-time system described by the I/O relationship $y[n] = x[1-2n]$ **is:**

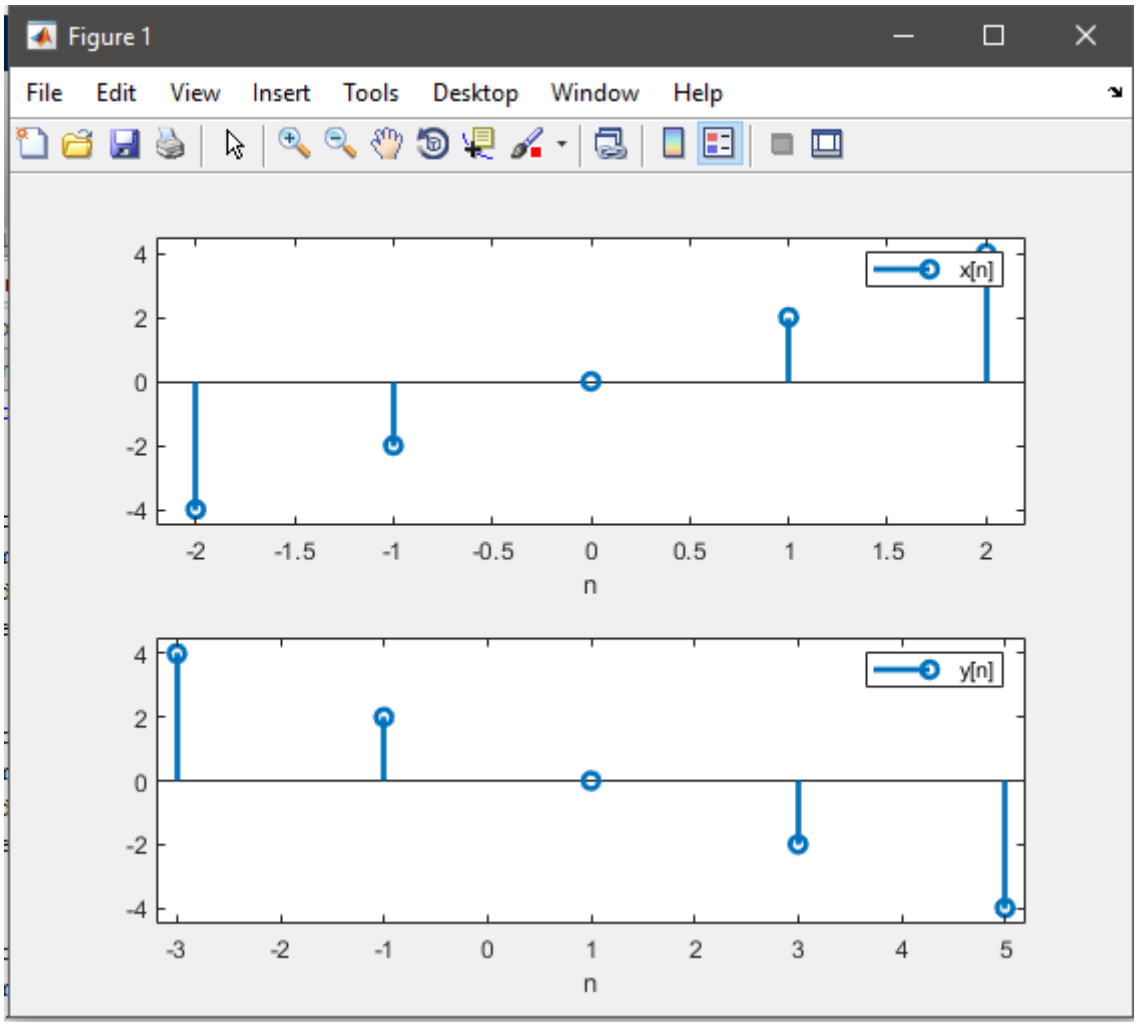
a) Static or Dynamic (input signal $x[n] = 2n, -2 \le n \le 2$)

b) Causal or non-causal (input signal $x[n] = 2n, -2 \le n \le 2$)

c) Linear or non-linear (input signals $x_1[n] = 2n, -2 \le n \le 4, x_2[n] = n/3, -2 \le n \le 4, a_1 = 2 \, and \, a_2 = 3$)

d) Shift invariant or shift variant (input signal $x[n] = 2n, -2 \le n \le 4$ and shift $n_0 = 3$)

```
n = -2:2;
x = 2.*n;
subplot(2,1,1)
stem(n,x,'LineWidth',2)       %print x[n]
xlim([-2.2 2.2])
ylim([-4.5 4.5])
legend('x[n]')
xlabel('n')

y = x;
subplot(2,1,2)
stem(1-2.*n,y,'LineWidth',2) %print y[n]
xlim([-3.2 5.2])
ylim([-4.5 4.5])
legend('y[n]')
xlabel('n')
```

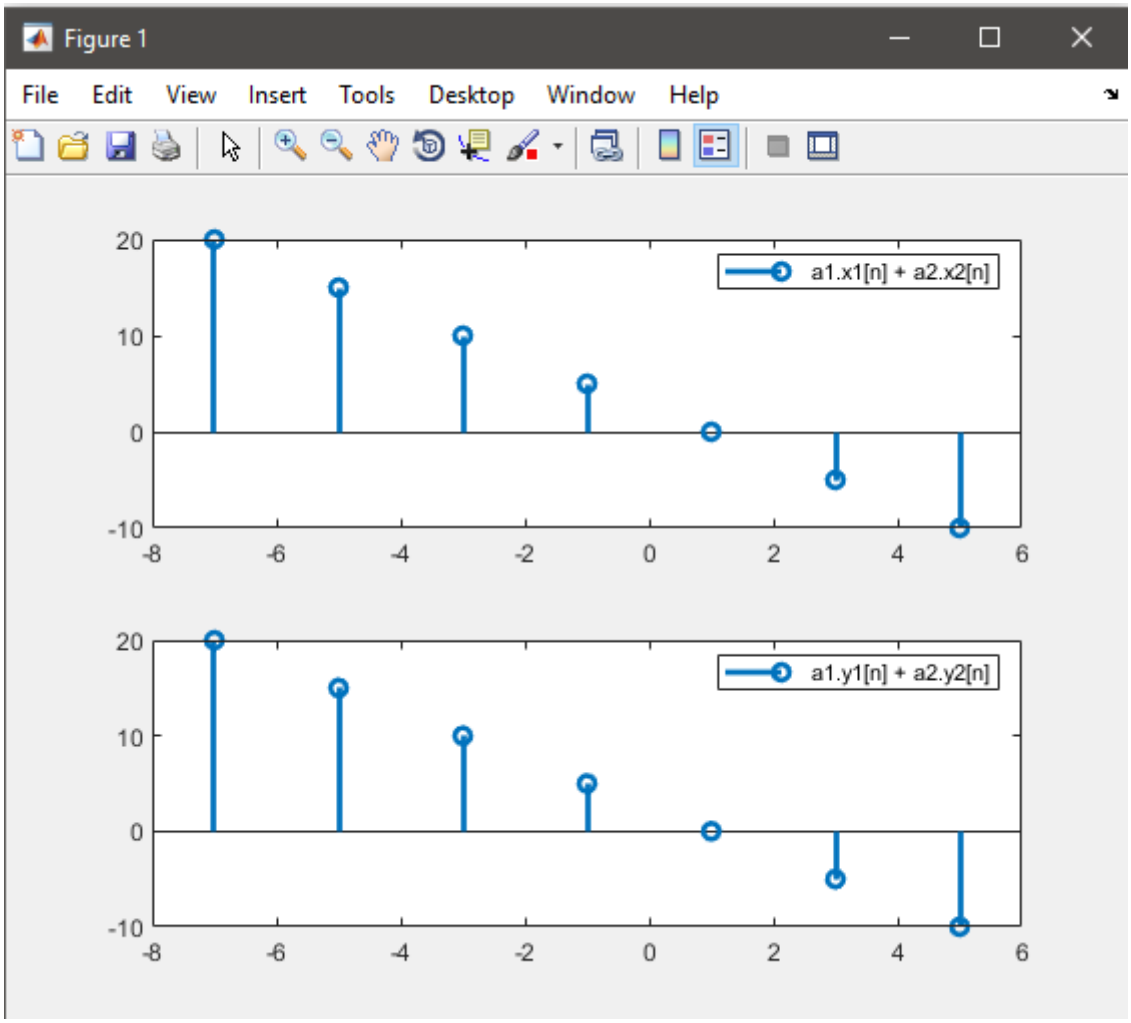


a) System is dynamic(with memory).

b) As the system depends on future as well as past values hence it is non causal.

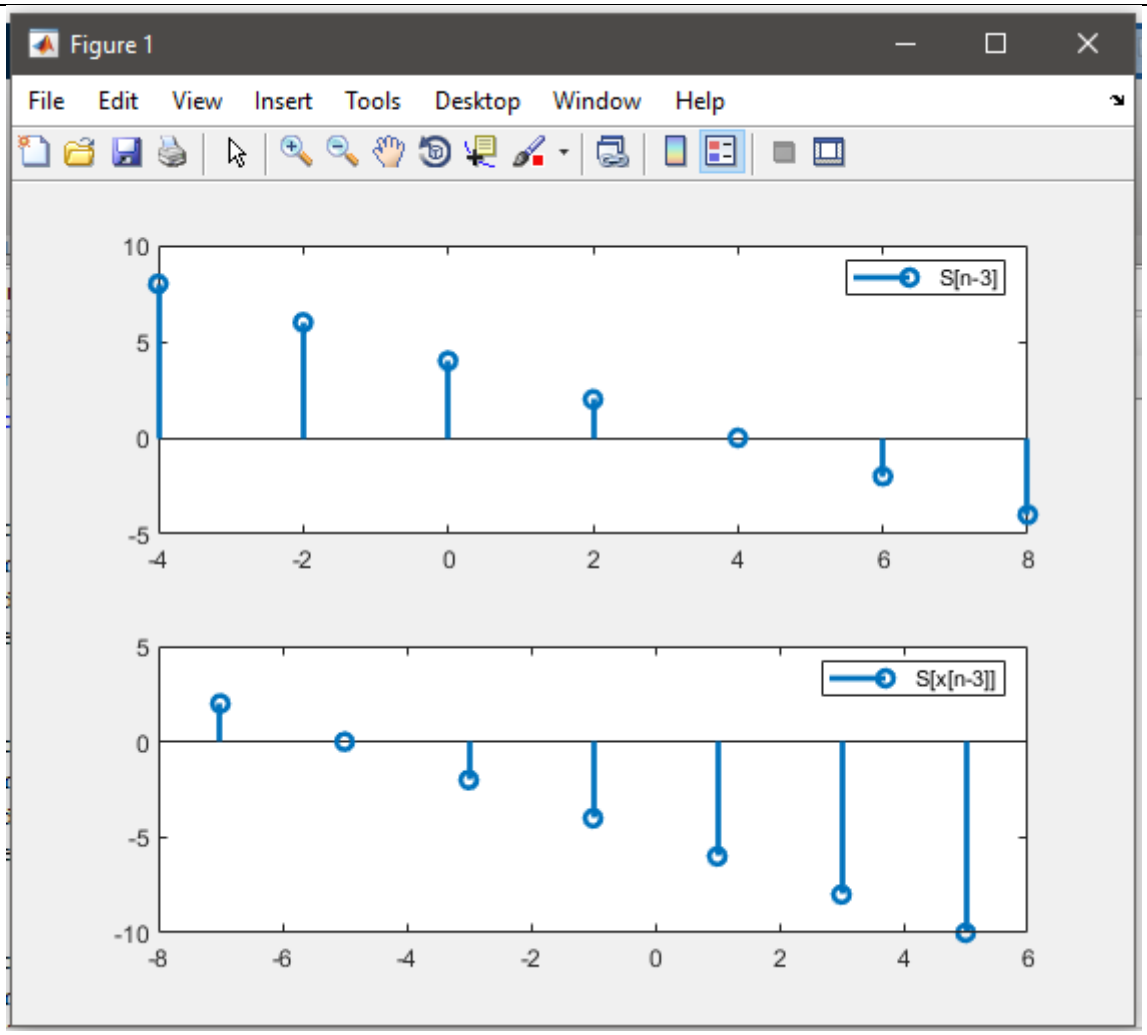
```
N = -2:4;
x1 = 2.*n;
x2 = n./3;
a1 = 2;
a2 = 3;
```

```
x = a1.*x1 + a2.*x2;
subplot(2,1,1)
stem(1-2.*n,x,'LineWidth',2)
legend('a1.x1[n] + a2.x2[n]')

legend('a1.x1[n] + a2.x2[n]')
subplot(2,1,2)
y1 = a1.*x1;
y2 = a2.*x2;
y = y1 + y2;
stem(1-2.*n,y,'LineWidth',2)
      legend('a1.y1[n] + a2.y2[n]')
```



c) As both the graphs are same, hence the system is linear.

```
N = -2:4;
x = 2.*n;
n0 = 3;
subplot(2,1,1)
stem(n0+1-2.*n,x,'LineWidth',2)
legend('S[n-3]')

y = 2.*(n-n0);
subplot(2,1,2)
stem(1-2.*n,y,'LineWidth',2)
legend('S[x[n-3]]')
```



d) As the graphs are not alike so the system is time variant.

Post-Lab Task

Critical Analysis / Conclusion

In this lab, we studied and practically implemented the concepts of Linearity, Causality, Memory, Stability and Time Variance. We completed various tasks and leant to differentiate the properties of these concepts using their graphs by plotting them on MATLAB.

Lab Assessment		
Pre-Lab	**/1**	**/10**
In-Lab	**/5**	
Critical Analysis	**/4**	
Instructor Signature and Comments		

LAB # 06

Analysis of Discrete LTI Systems using Convolution Sum

Lab 06- Analysis of Discrete LTI Systems using Convolution Sum

Pre-Lab Tasks

6.1 Discrete Time Convolution:

An LTI discrete-time system is (equivalently to the continuous-time case) completely describes by impulse response, which is usually denoted by $h[n]$. The impulse response of an LTI discrete-time system is the output of the system when the unit impulse sequence (or Kronecker delta function) is applied as input. Even though the unit impulse input consists of one no-zero term, the impulse response signal $h[n]$ usually consists of more than one non-zero elements. The explanation is that the system is dynamic (with memory); that is, the system responds over various time instances to the output applied at $n=0$. The knowledge of the impulse response $h[n]$ of a system allows the computation of the response $y[n]$ of the system to any input signal $x[n]$. The output signal is computed by discrete time convolution. The mathematical expression (6.1) of discrete time convolution is

$$y[n]=x[n]*h[n]=\sum_{k=-\infty}^{+\infty}x[k]h[n-k]=\sum_{k=-\infty}^{+\infty}h[k]x[n-k]$$

The convolution between two discrete time signals is computed by using the MATLAB command conv.

6.1.1 The Command conv:

In MATLAB, the command conv allows the direct computation of the convolution between two signals. In order to illustrate the conv command There are three rules that have to be applied for successful computation of the convolution between two continuous-time signals.

- First rule: Two signals (input and impulse response) should be defined in the same time interval. Hence both the signals are defined in the time interval $a \le n \le b$.
- Second rule: When a signal consists of multiple parts, the time intervals in which each part is defined must not overlap. For example,

$$h[n]=\begin{cases}1-n & 0\le n\le 1\\ 0 & 1<n\le 2\end{cases}$$

Note that the equality at $n=1$ is placed only in the upper part.

Having defined the input and impulse response signals the response of the system can be computed by convoluting the two signals. The convolution is implemented by the command y=conv(x,h). The response of the system is now computed and the only thing left to do is to plot it. However, the number of elements of the output vector y is not equal to the number of elements of vectors x or h.

The precise relationship is length(y)=length(x)+length(h)-1. To overcome this, the third rule must be applied.

- Third rule: The output of the system is plotted in the double time interval of the one in which the output and impulse response signals are defined.

6.1.2 The Unit Impulse Sequence as Input to a System:

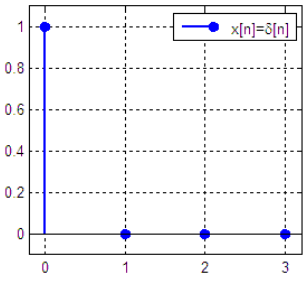
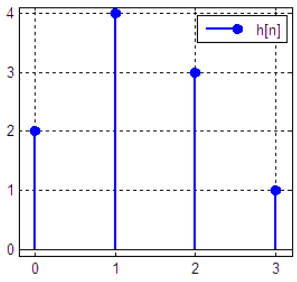
In this section, the impulse response of a discrete time system is discussed in detail. More specifically, it is established that if the unit impulse sequence $\delta[n]$ is the input to a system, the output of the system is the impulse response $h[n]$ of the system.

Example:

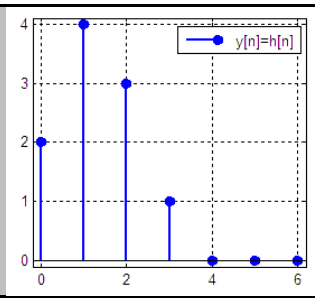
Suppose that a discrete time system is described by the impulse response $h[n]=[2\ 4\ 1\ 3], 0\le n\le 3$. Compute the response of the system to the input signal

$$x[n]=\delta[n]=\begin{cases}1 & n=0\\ 0 & n\neq 0\end{cases}$$

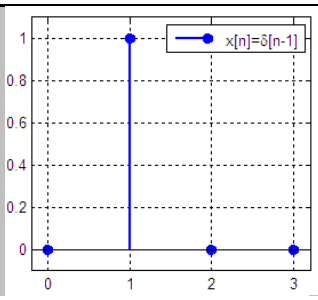
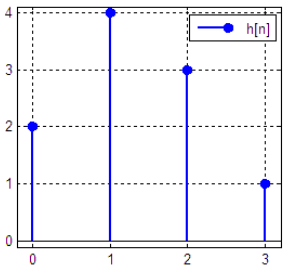
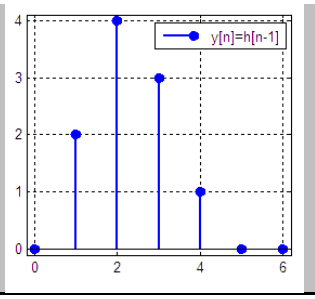
The response of the system is computed by convoluting the impulse response $h[n]$ with the input signal $\delta[n]$. Recall that $\delta[n]$ must be defined in the same time interval to $h[n]$; that is, $0\le n\le 3$. For illustration purposes both signals are plotted.

Commands	Results	Comments
x=[1 0 0 0]; **n=0:3;** **stem(n,x,'fill','linewidth',2),grid on** **axis([-0.2 3.2 -0.1 1.1])** **legend('x[n]=\delta[n]')**		Input signal $x[n]=\delta[n]$ plotted in $0\le n\le 3$.
H=[2 4 3 1]; **n=0:3;** **stem(n,h,'fill','linewidth',2),grid on** **axis([-0.2 3.2 -0.1 4.1])** **legend('h[n]')**		Impulse response $h[n]=[2\ 4\ 1\ 3]$ plotted in $0\le n\le 3$.

The number of elements of the output vector $y[n]$ compared to the number of elements of the input and impulse response vectors. The exact relationship is length (y)=length(x)+length(h)-1. Therefore the output of the system is plotted in the double time interval from input and impulse response signals.

Commands	Results	Comments
y=conv(x,h); **stem(0:6,y,'fill','linewidth',2),grid on** **axis([-0.2 6.2 -0.1 4.1])** **legend('y[n]=h[n]')**		The output $y[n]$ is computed and plotted for $0 \le n \le 6$. As expected the output and impulse response signals are the same.

The output and impulse response signals are the same; thus the identity property $\left(h[n]*\delta[n]=h[n]\right)$ clearly stands. The system under consideration is a linear shift invariant system. If at a linear and shift invariant system, with impulse response $h[n]$, the input signal $\delta[n-k]$, i.e., a shifted unit impulse sequence is applied, then the response of the system is also shifted by k units.

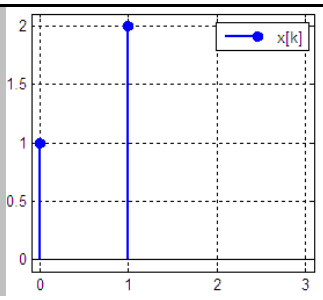
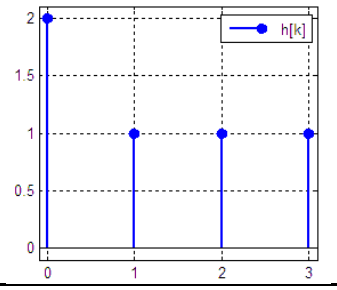
Commands	Results	Comments
x=[0 1 0 0]; **n=0:3;** **stem(n,x,'fill','linewidth',2),grid on** **axis([-0.2 3.2 -0.1 1.1])** **legend('x[n]=\delta[n-1]')**		The shifted unit impulse sequence $x[n]=\delta[n-1]$ is the input to the system.
H=[2 4 3 1]; **n=0:3;** **stem(n,h,'fill','linewidth',2),grid on** **axis([-0.2 3.2 -0.1 4.1])** **legend('h[n]')**		Impulse response $h[n]=[2\ 4\ 1\ 3]$, $0 \le n \le 3$.
Y=conv(x,h); **stem(0:6,y,'fill','linewidth',2),grid on** **axis([-0.2 6.2 -0.1 4.1])** **legend('y[n]=h[n-1]')**		The output of the system is its impulse response shifted by 1 unit to the right.

As expected, the response $y[n]$ of the system to the input signal $\delta[n-1]$ is $h[n-1]$. More generally, the response of a linear shift invariant system to a shifted unit impulse sequence $\delta[n-k]$ shifted by k units version of impulse response, i.e., the output of the system is $h[n-k]$.

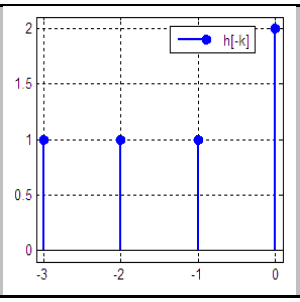
6.1.3 Computation of Discrete Time Convolution:

The process that has to be followed to analytically derive the convolution sum (equation 6.1) is as follows. First, the input and impulse response signals are plotted in the k-axis. One of the two signals is reversed about the amplitude axis and its reflection is shifted from $-\infty$ to $+\infty$ by changing appropriately the value of n. The output of the system is computed from the overlapping values of $x[n]$ and $h[n-k]$ according to the convolution sum $y[n] = x[n] * h[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n-k]$. Te convolution procedure for two discrete time signals is demonstrated by using the signals $x[n] = [1\ \ 2], 0 \le n \le 1$ and $h[n] = [2\ \ 1\ \ 1\ \ 1], 0 \le n \le 3$.

Step 1: The two signals are plotted at the k-axis.

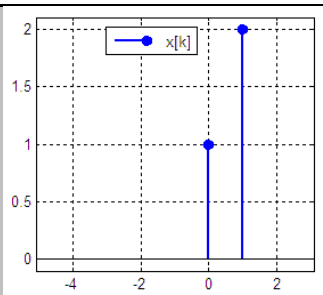
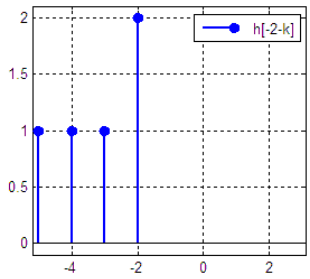
Commands	Results	Comments
kx=[0 1]; **x=[1 2];** **stem(kx,x,'fill','linewidth',2),grid on** **axis([-0.1 3.1 -0.1 2.1])**		Input signal $x[k]$.
Kh=0:3; **h=[2 1 1 1];** **stem(kh,h,'fill','linewidth',2),grid on** **axis([-0.1 3.1 -0.1 2.1])** **legend('h[k]')**		Impulse response signal $h[k]$.

Step 2: The reflected version of $h[k]$, namely, $h[-k]$ is plotted.

Commands	Results	Comments
stem(-kh,h,'fill','linewidth',2) ,grid on **axis([-3.1 0.1 -0.1 2.1])** **legend('h[-k]')**		The reflected signal $h[-k]$.

Step 3: The signal $h[n-k]$ is shifted from $-\infty$ to $+\infty$ by changing appropriately the value of n. The output if the system is computed at each shift through equation 6.1, in which only the values of overlapping parts of $x[n]$ and $h[n-k]$ are considered. In discrete time case we will compute the sum of $x[n]$ and $h[n-k]$. There are three stages, first there is zero overlap, next there is overlap and finally there is no overlap.

- First Stage: Zero Overlap. The stage occurs for $n \le -1$.

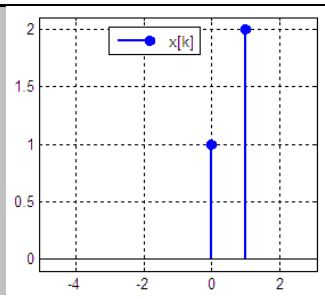
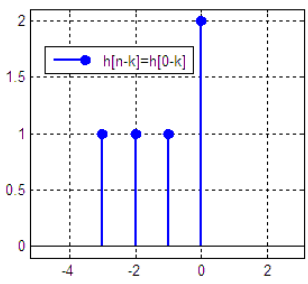
Commands	Results	Comments
stem(kx,x,'fill','linewidth',2),grid on axis([-5.1 3.1 -0.1 2.1]) legend('x[k]')		Input signal $x[k]$.
N=-2; stem(-kh+n,h,'fill','linewidth',2), grid on axis([-5.2 3.1 -0.1 2.1]) legend('h[-2-k]')		Impulse response signal $h[n-k]$ for $n=-2$.

Obviously the two signals do not overlap. Thus the output is $y[n]=0,\ n \le -1$.

- Second Stage: Overlap. The signals $x[n]$ and $h[n-k]$ overlap for $0 \le n \le 4$.

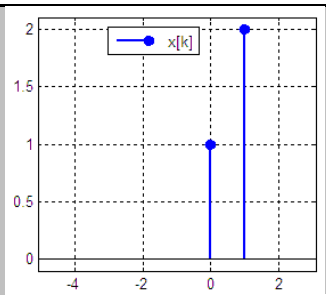
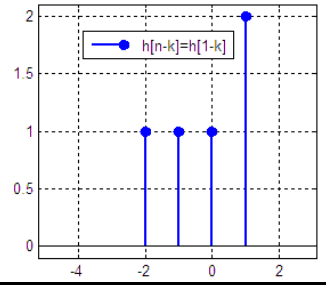
In order to compute the output, the two signals are plotted for $n = 0,1,...,4$. The output is computed through the convolution sum by taking into account only the overlapping samples.

For $n = 0,$

Commands	Results
stem(kx,x,'fill','linewidth',2),grid on axis([-5.1 3.1 -0.1 2.1]) legend('x[k]')	
n=0; stem(-kh+n,h,'fill','linewidth',2),grid on axis([-5.2 3.1 -0.1 2.1]) legend('h[n-k]=h[0-k]')	

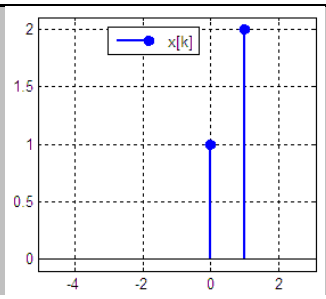
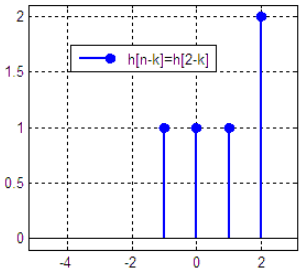
For $n = 0,$ the signals $x[k]$ and $h[n-k] = h[0-k]$ have one overlapping sample at $k = 0$. The output of the system for $n = 0$; that is, $y[0] = (1)(2)$, where 1 is the value of $x[k]$ and 2 is the value of $h[n-k]$ at the overlapping sample. In order to understand better the output calculation, notice that according to the definition

of the convolution sum $y[n] = x[n] * h[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n-k]$, the overlapping samples of $x[k]$ and $h[n-k]$ are multiplied. The sum is applied when more than one samples overlap. This is the case for $n=1$.

Commands	Results
stem(kx,x,'fill','linewidth',2),grid on **axis([-5.1 3.1 -0.1 2.1])** **legend('x[k]')**	
n=1; **stem(-kh+n,h,'fill','linewidth',2),grid on** **axis([-5.2 3.1 -0.1 2.1])** **legend('h[n-k]=h[0-k]')**	

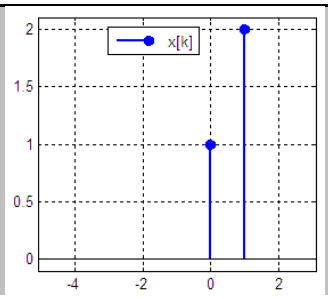
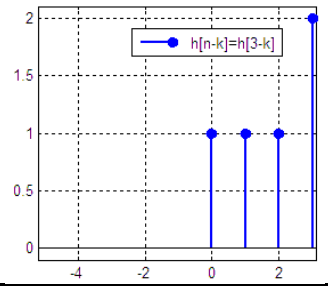
For $n=1$, the signals $x[k]$ and $h[n-k]=h[1-k]$ overlap with two samples, namely, at $k=0$ and at $k=1$. The output for $n=1$, that is, $y[n]=y[1]$ is computed as $y[1]=(2)(2)+(1)(1)$, where 2 is the value of $x[k]$ and $h[n-k]$ at $k=1$, while 1 is the value of $x[k]$ and $h[n-k]$ at $k=0$. Hence $y[1]=5$.

For $n=2$, we have

Commands	Results
stem(kx,x,'fill','linewidth',2),grid on **axis([-5.1 3.1 -0.1 2.1])** **legend('x[k]')**	
n=2; **stem(-kh+n,h,'fill','linewidth',2),grid on** **axis([-5.2 3.1 -0.1 2.1])** **legend('h[n-k]=h[2-k]')**	

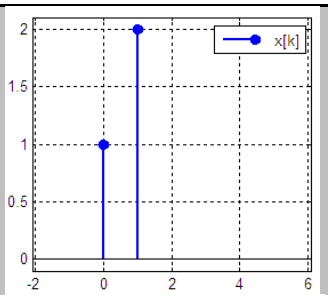
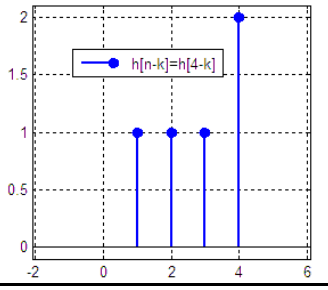
For $n=2$, the signals $x[k]$ and $h[n-k]=h[2-k]$ overlap at two points, at $k=0$ and at $k=1$. The output for $n=2$, that is, $y[n]=y[2]$ is computed as $y[2]=(2)(1)+(1)(1)$, where 2 is the value of $x[k]$ and 1 is the value of $h[n-k]$ at $k=1$, while 1 is the value of $x[k]$ and $h[n-k]$ at $k=0$. Hence $y[2]=3$.

For $n=3$, we have

Commands	Results
stem(kx,x,'fill','linewidth',2),grid on axis([-5.1 3.1 -0.1 2.1]) legend('x[k]')	
n=3; stem(-kh+n,h,'fill','linewidth',2),grid on axis([-5.2 3.1 -0.1 2.1]) legend('h[n-k]=h[3-k]')	

The output for $n=3$, i.e., $y[n]=y[3]$ is computed as $y[3]=(2)(1)+(1)(1)=3.$

For $n=4$,

Commands	Results
stem(kx,x,'fill','linewidth',2),grid on axis([-2.1 6.1 -0.1 2.1]) legend('x[k]')	
n=4; stem(-kh+n,h,'fill','linewidth',2),grid on axis([-2.1 6.1 -0.1 2.1]) legend('h[n-k]=h[4-k]')	

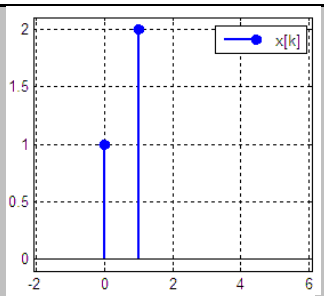
For $n=4$, there is only one overlap at $k=1$. Thus the output for $n=4$ is $y[4]=(2)(1)=2.$

- Third Stage: Zero Overlap. For $n\geq 5$, the input and impulse response signals do not overlap, hence $y[n]=0,\ n\geq 5$.

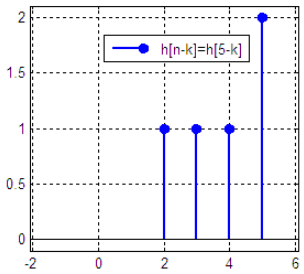
For $n=5$, we have

Commands	Results

```
stem(kx,x,'fill','linewidth',2),grid on
axis([-2.1 6.1 -0.1 2.1])
legend('x[k]')
```



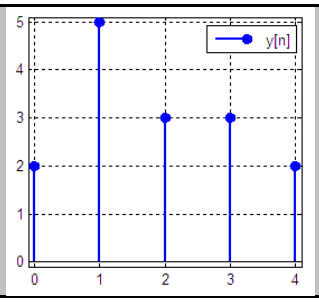

```
n=5;
stem(-kh+n,h,'fill','linewidth',2),grid on
axis([-2.1 6.1 -0.1 2.1])
legend('h[n-k]=h[5-k]')
```



Combining the derived results we conclude that the response of the system with impulse response $h[n]=[2\ 1\ 1\ 1],\ 0\le n\le 3$ to the input signal $x[n]=[1\ 2],\ 0\le n\le 1$ is $y[n]=[2\ 5\ 3\ 3\ 2],\ 0\le n\le 4$. The output signal is plotted in figure below.

Commands	Results

```
n=0:4;
y=[2 5 3 3 2];
stem(n,y,'fill','linewidth',2),grid on
axis([-0.1 4.1 -0.1 5.1])
legend('y[n]')
```



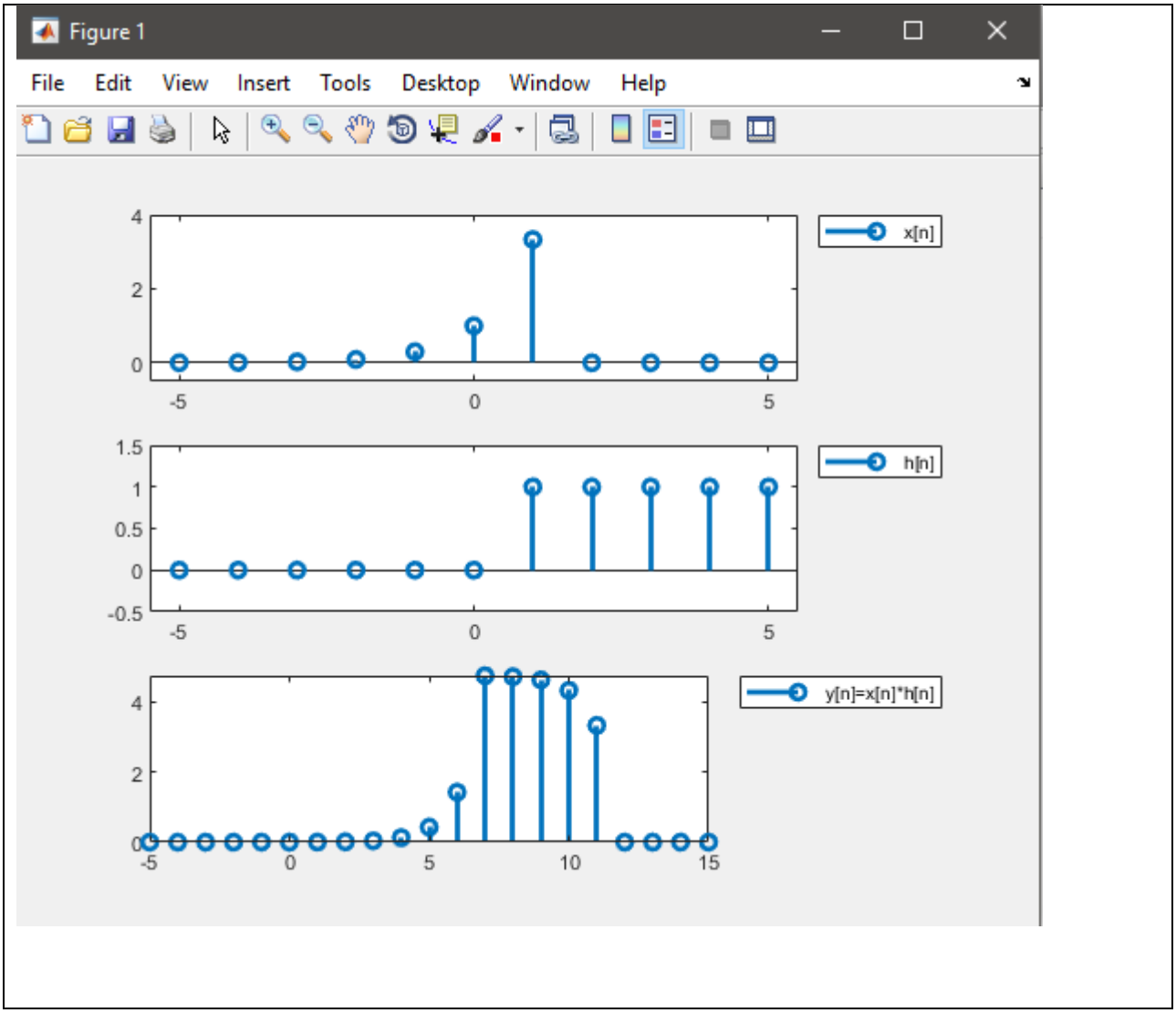
There are two basic principles that should be considered while computing the convolution between two discrete time signals $x[n]$ and $h[n]$ using conv command.

1. Suppose that the length of vector x is N and the length of vector of h is M. The outcome of command y=conv(x,h) is a vector y of length $M+N-1$. In other words, length(y)=length(x)+length(h)-1.
2. If non zeros values of $x[n]$ are in the interval $[a_x, b_x]$ and non zero values of $h[n]$ are in the interval $[a_h, b_h]$ then the non zero values of the output $y[n]$ are in the interval $[a_x+a_h, b_x+b_h]$.

In-Lab Tasks

Task 01: Compute and plot the convolution $y[n]=x[n]*h[n]$ by any of the two procedures, where $x[n]=\left(\frac{1}{3}\right)^{-n}u[-n-1]$ and $h[n]=u[n-1]$

```
n = -5:5;
u = (n<=1);
x = 0.3.^-n.*u;
subplot(3,1,1)
stem(n,x,'LineWidth',2)
axis([-5.5 5.5 -0.5 4])
legend('x[n]','Location','NorthEastOutside')

h=zeros(1,11);
h(7:11)=1;
subplot(3,1,2)
stem(n,h,'LineWidth',2)
axis([-5.5 5.5 -0.5 1.5])
legend('h[n]','Location','NorthEastOutside')

y =conv(h,x);
subplot(3,1,3)
stem(-5:15,y,'LineWidth',2)
legend('y[n]=x[n]*h[n]','Location','NorthEastOutside')
```



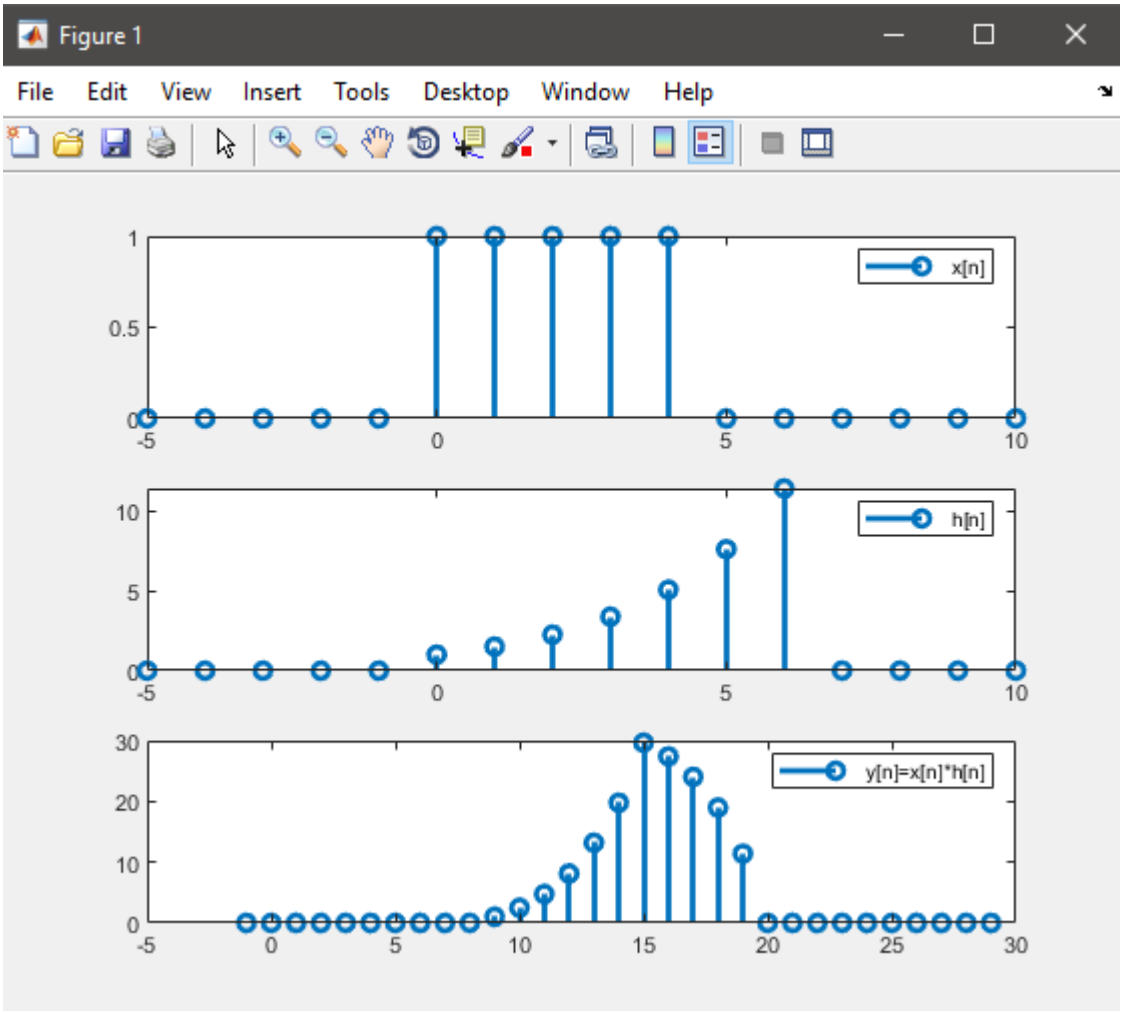
Task 02: Compute and plot the convolution of following signals (by both procedures)

$$x[n]=\begin{cases}1 & 0\le n\le 4\\ 0 & elsewhere\end{cases}$$

and

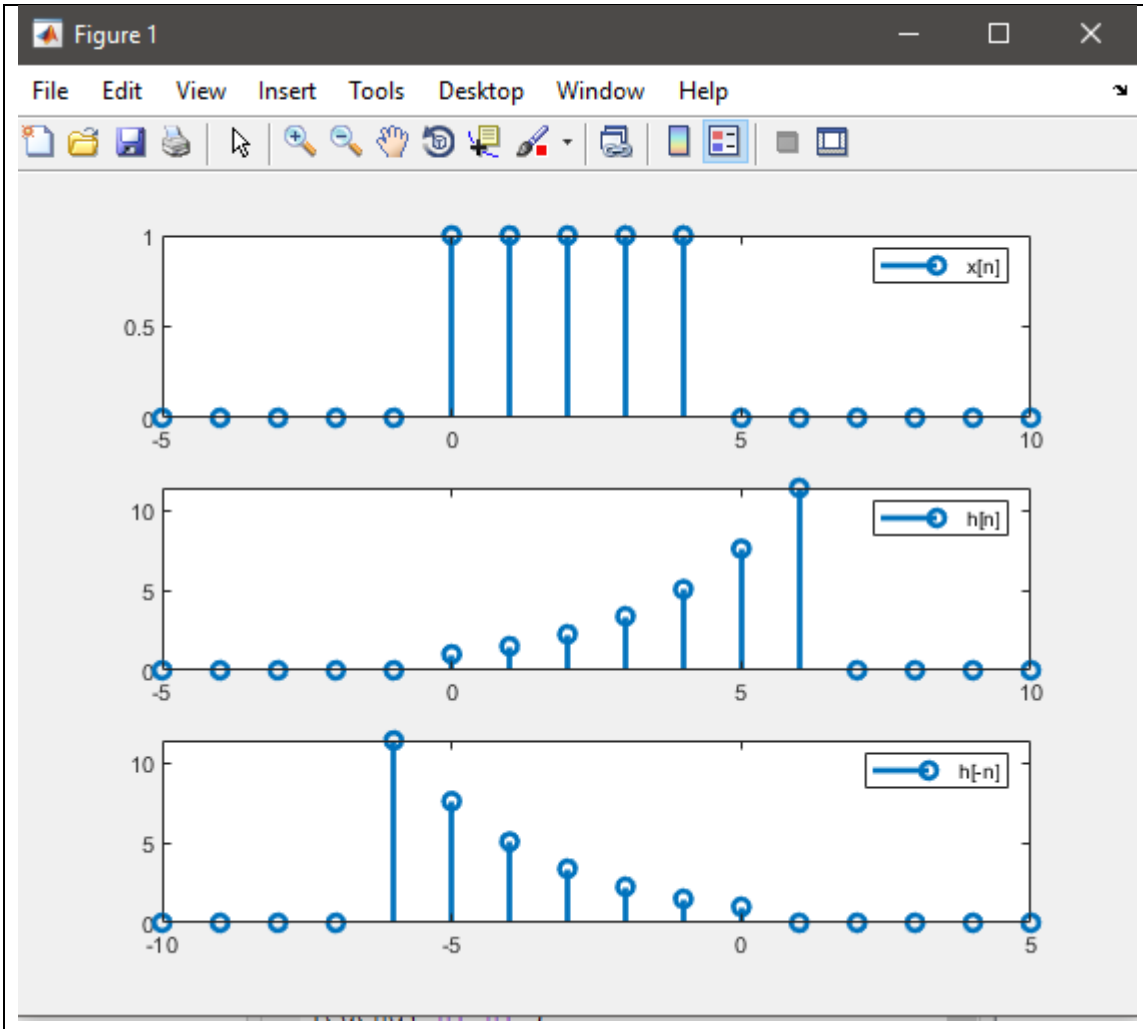
$$h[n]=\begin{cases}1.5^n & 0\le n\le 6\\ 0 & elsewhere\end{cases}$$

```
n = -5:10;
x = [0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0];

subplot(3,1,1)
stem(n,x,'LineWidth',2)
legend('x[n]')

subplot(3,1,2)
a1 = (n>=0)&(n<=6);
a2 = 1.5.^n;
h = a1.*a2;
stem(n,h,'LineWidth',2)
legend('h[n]')

y = conv(x,h);
subplot(3,1,3)
stem(-1:29,y,'LineWidth',2)
legend('y[n]=x[n]*h[n]')
```



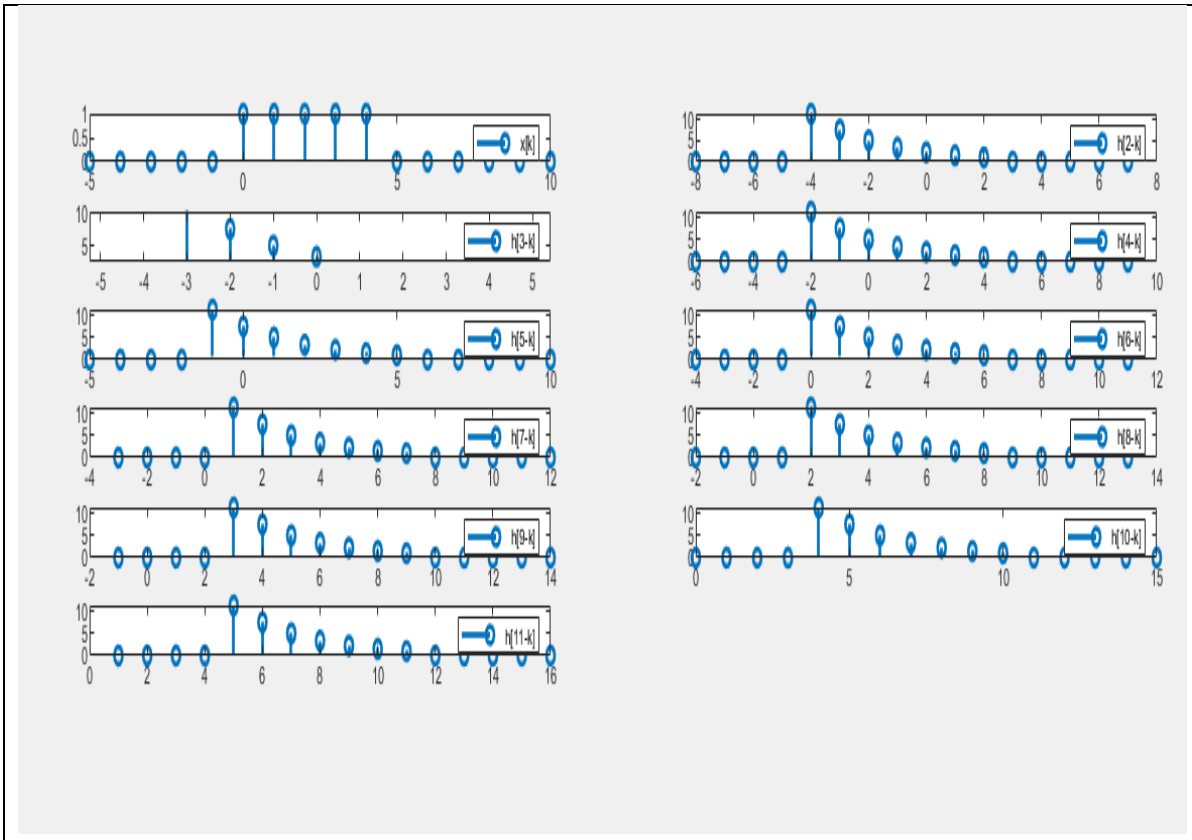
Other Method:

```
n = -5:10;
x = [0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0];
a1 = 1.5.^n;
a2 = (n>=0)&(n<=6);
h = a1.*a2;

subplot(3,1,1)
stem(n,x,'LineWidth',2)
legend('x[n]')

subplot(3,1,2)
stem(n,h,'LineWidth',2)
legend('h[n]')

subplot(3,1,3)
stem(-n,h,'LineWidth',2)
legend('h[-n]')
```



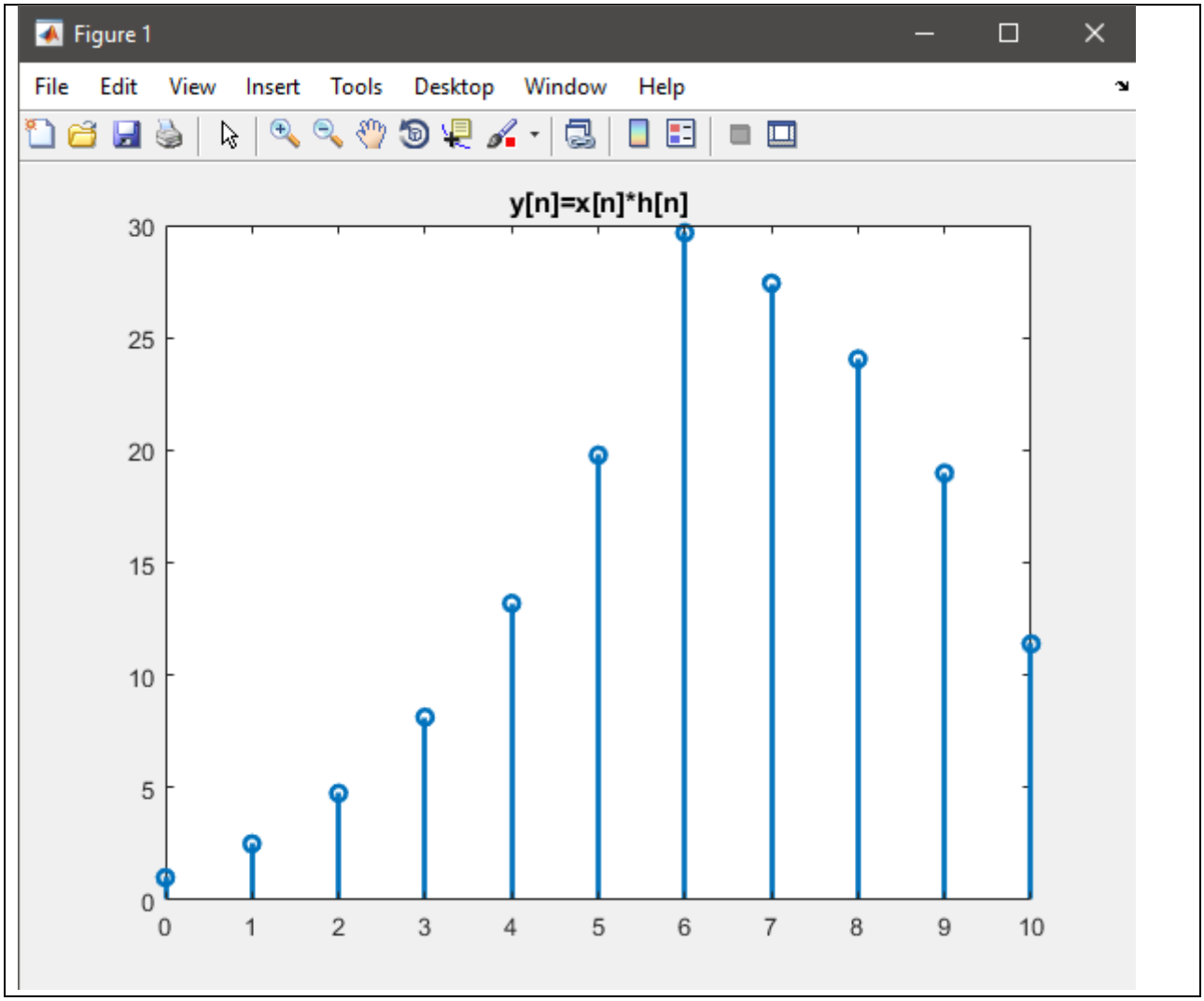

```
k = -5:10;   % changing axis
n = 0;       % delay
subplot(12,2,1)
stem(k,x,'LineWidth',2)
legend('x[k]')
subplot(12,2,2)
stem(-k+n,h,'LineWidth',2)
legend('h[0-k]')
n=2;       % delay
subplot(12,2,2)
stem(-k+n,h,'LineWidth',2)
legend('h[2-k]')

n=3;       % delay
subplot(12,2,3)
stem(-k+n,h,'LineWidth',2)
legend('h[3-k]')

n=4;       % delay
subplot(12,2,4)
```

```
stem(-k+n,h,'LineWidth',2)
legend('h[4-k]')

n=5;       % delay
subplot(12,2,5)
stem(-k+n,h,'LineWidth',2)
legend('h[5-k]')

n=6;       % delay
subplot(12,2,6)
stem(-k+n,h,'LineWidth',2)
legend('h[6-k]')

n=7;       % delay
subplot(12,2,7)
stem(-k+n,h,'LineWidth',2)
legend('h[7-k]')

n=8;       % delay
subplot(12,2,8)
stem(-k+n,h,'LineWidth',2)
legend('h[8-k]')

n=9;       % delay
subplot(12,2,9)
stem(-k+n,h,'LineWidth',2)
legend('h[9-k]')

n=10;       % delay
subplot(12,2,10)
stem(-k+n,h,'LineWidth',2)
legend('h[10-k]')

n=11;       % delay
subplot(12,2,11)
stem(-k+n,h,'LineWidth',2)
legend('h[11-k]')
```



Convolution

```
%convolution
y = [1 2.5 4.75 8.13 13.19 19.78 29.67 27.42 24.05 18.98 11.39];
yn = 0:10;
stem(yn,y,'LineWidth',2)
title('y[n]=x[n]*h[n]')
```



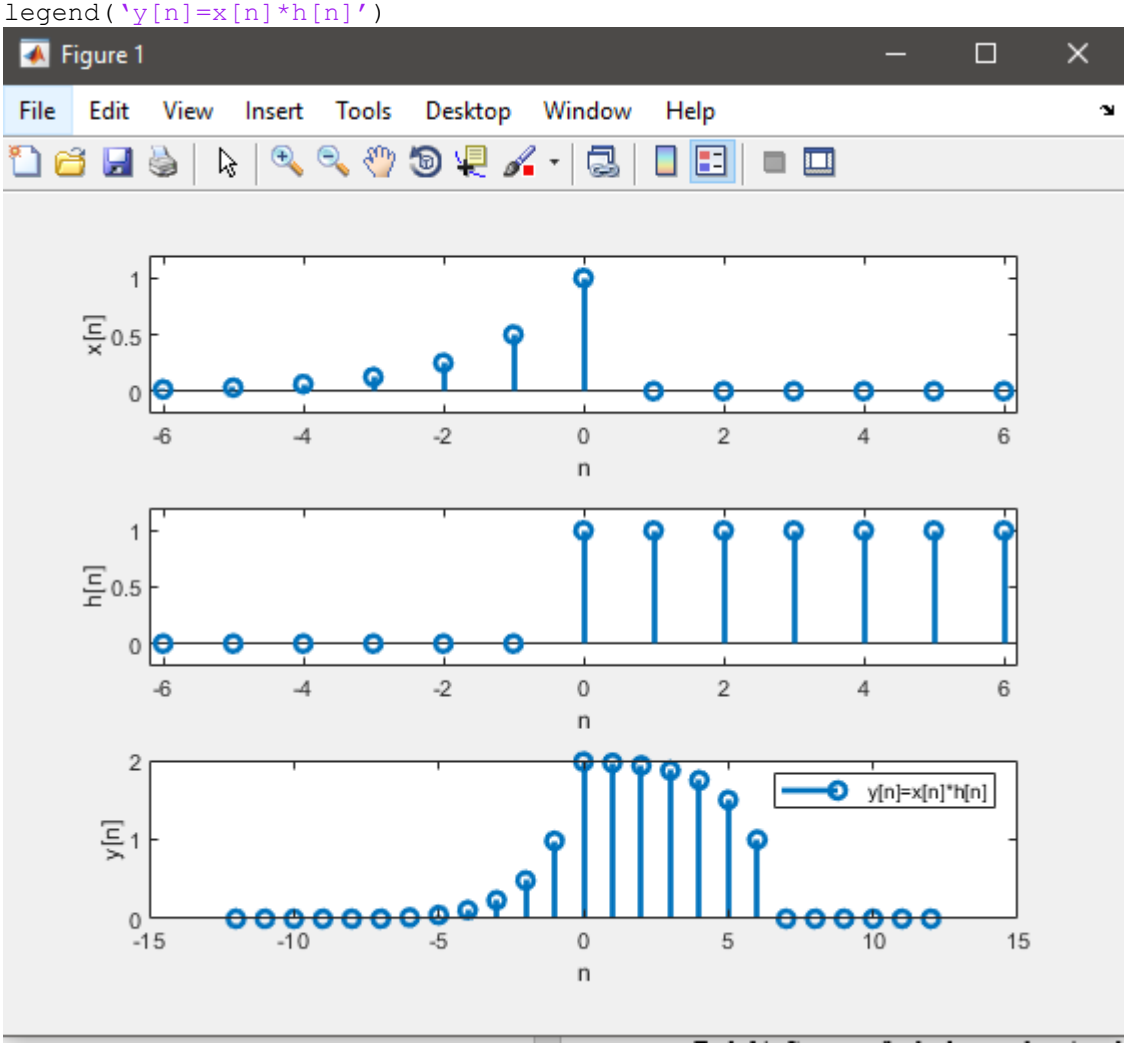
Task 03: Consider and LTI system with input $x[n]$ and unit impulse response $h[n]$ specified as

$x[n] = 2^n u[-n]$

$h[n] = u[n]$

Compute the response of the system (by both methods) where we have $-6 \le n \le 6$.

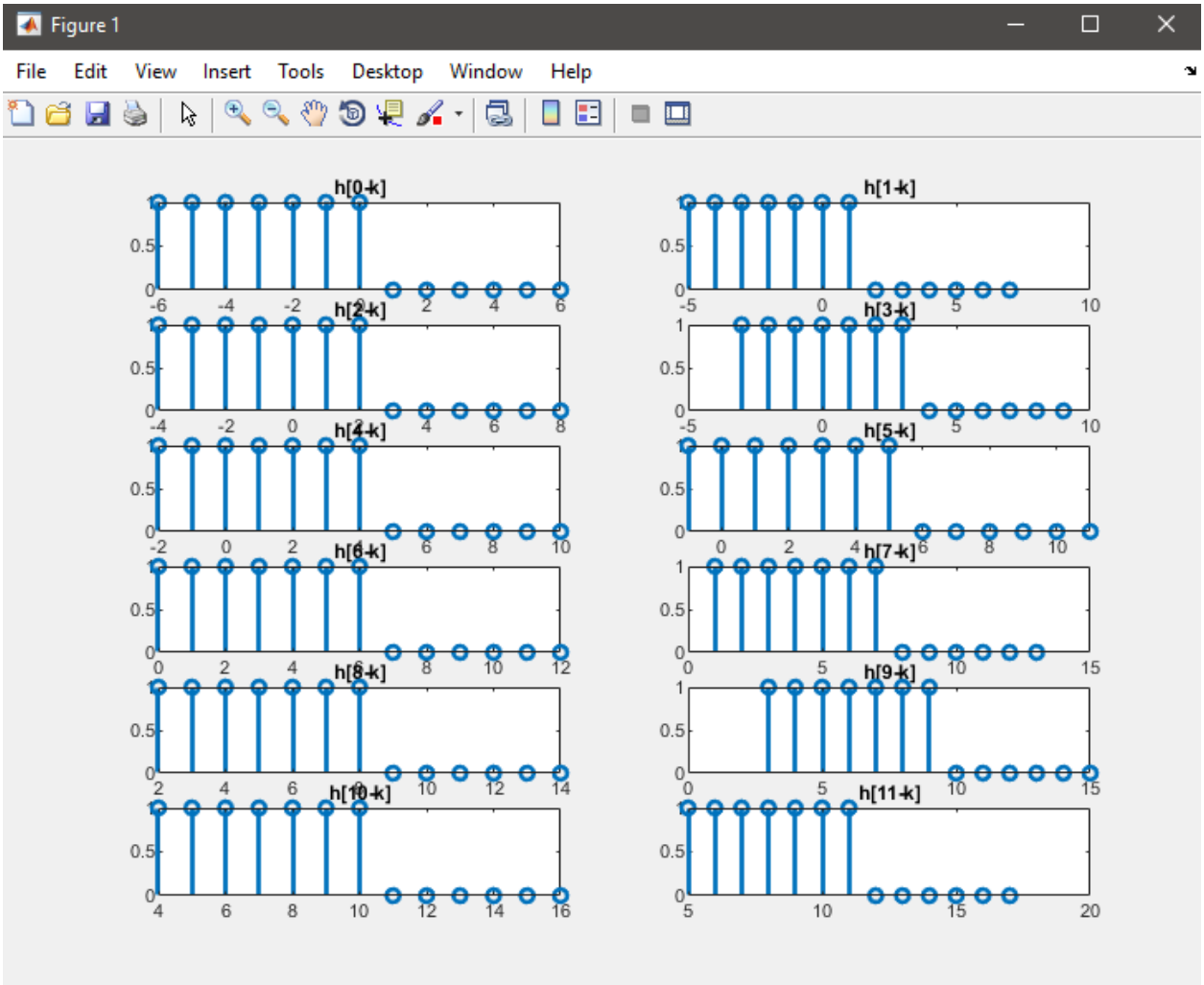
```
N = -6:6;
h = [0 0 0 0 0 0 1 1 1 1 1 1 1];
x = 2.^n.*(n<=0);
subplot(3,1,1)
stem(n,x,'LineWidth',2)
axis([-6.2 6.2 -0.2 1.2])
xlabel('n')
ylabel('x[n]')

subplot(3,1,2)
stem(n,h,'LineWidth',2)
axis([-6.2 6.2 -0.2 1.2])
xlabel('n')
ylabel('h[n]')

y = conv(h,x);
yn = -12:12;
subplot(3,1,3)
stem(yn,y,'LineWidth',2)
xlabel('n')
ylabel('y[n]')
legend('y[n]=x[n]*h[n]')
```



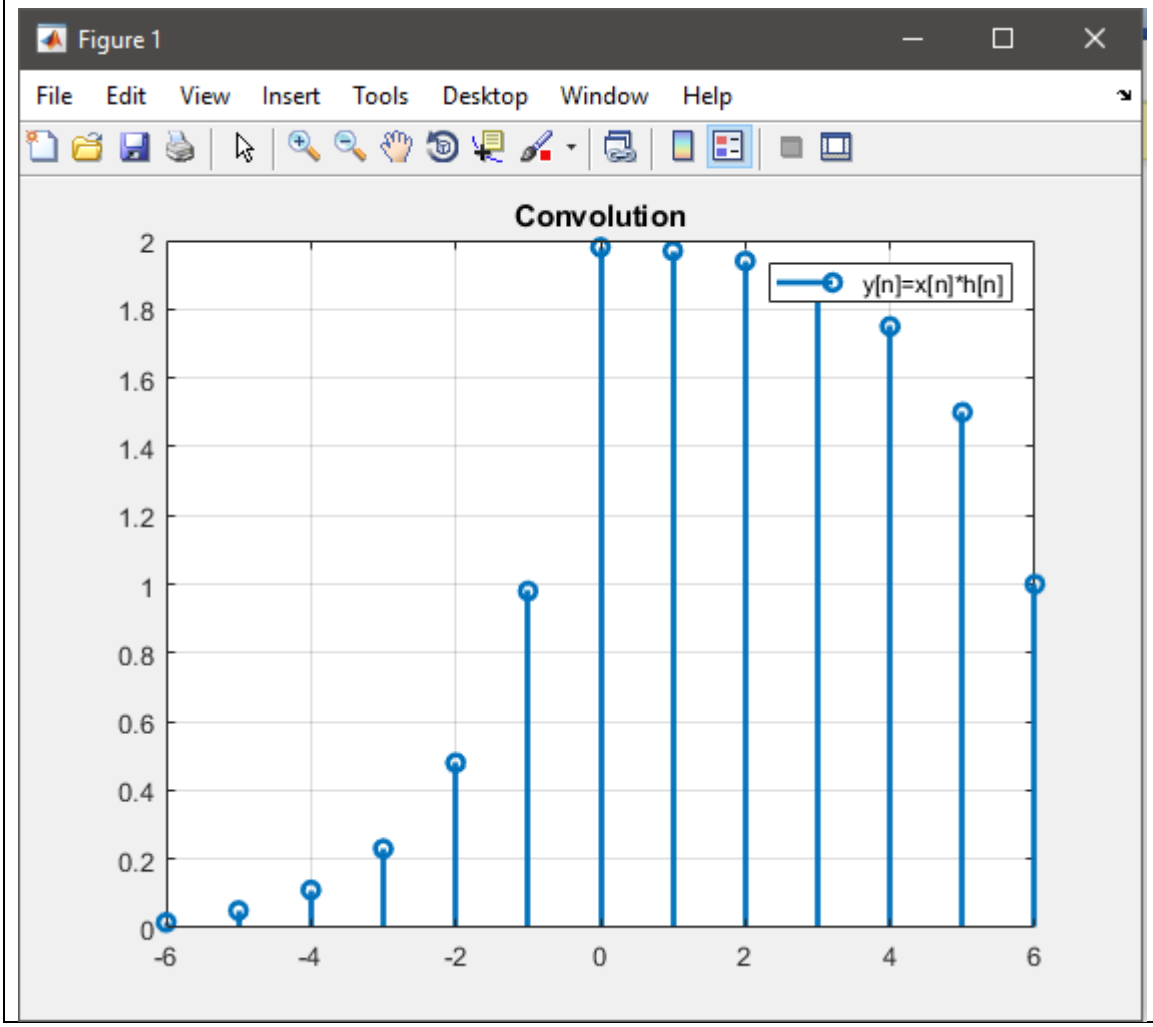
Other Method:

```
k = -6:6;        % changing axis
x = 2.^k.*(k<=0);
subplot(3,1,1)
stem(k,x,'LineWidth',2)
legend('x[k]')

h = [0 0 0 0 0 0 1 1 1 1 1 1 1];
subplot(3,1,2)
stem(k,h,'LineWidth',2)
legend('h[k]')

subplot(3,1,3)
stem(-k,h,'LineWidth',2)
legend('h[-k]')
shift=0;              % No shift
subplot(6,2,1)
stem(-k+shift,h,'LineWidth',2)
title('h[0-k]')


shift=1;              % 1 unit shift
subplot(6,2,2)
stem(-k+shift,h,'LineWidth',2)
title('h[1-k]')



shift=2;              % 2 unit shift
subplot(6,2,3)
stem(-k+shift,h,'LineWidth',2)
title('h[2-k]')

shift=3;              % 3 unit shift
subplot(6,2,4)
stem(-k+shift,h,'LineWidth',2)
title('h[3-k]')

shift=4;              % 4 unit shift
subplot(6,2,5)
stem(-k+shift,h,'LineWidth',2)
title('h[4-k]')

shift=5;              % 5 unit shift
subplot(6,2,6)
stem(-k+shift,h,'LineWidth',2)
title('h[5-k]')

shift=6;              % 6 unit shift
subplot(6,2,7)
stem(-k+shift,h,'LineWidth',2)
title('h[6-k]')

shift=7;              % 7 unit shift
subplot(6,2,8)
stem(-k+shift,h,'LineWidth',2)
```

```
title('h[7-k]')

shift=8;                % 8 unit shift
subplot(6,2,9)
stem(-k+shift,h,'LineWidth',2)
title('h[8-k]')

shift=9;                % 9 unit shift
subplot(6,2,10)
stem(-k+shift,h,'LineWidth',2)
title('h[9-k]')

shift=10;                % 10 unit shift
subplot(6,2,11)
stem(-k+shift,h,'LineWidth',2)
title('h[10-k]')



shift=11;                % 11 unit shift
subplot(6,2,12)
stem(-k+shift,h,'LineWidth',2)
title('h[11-k]')
```

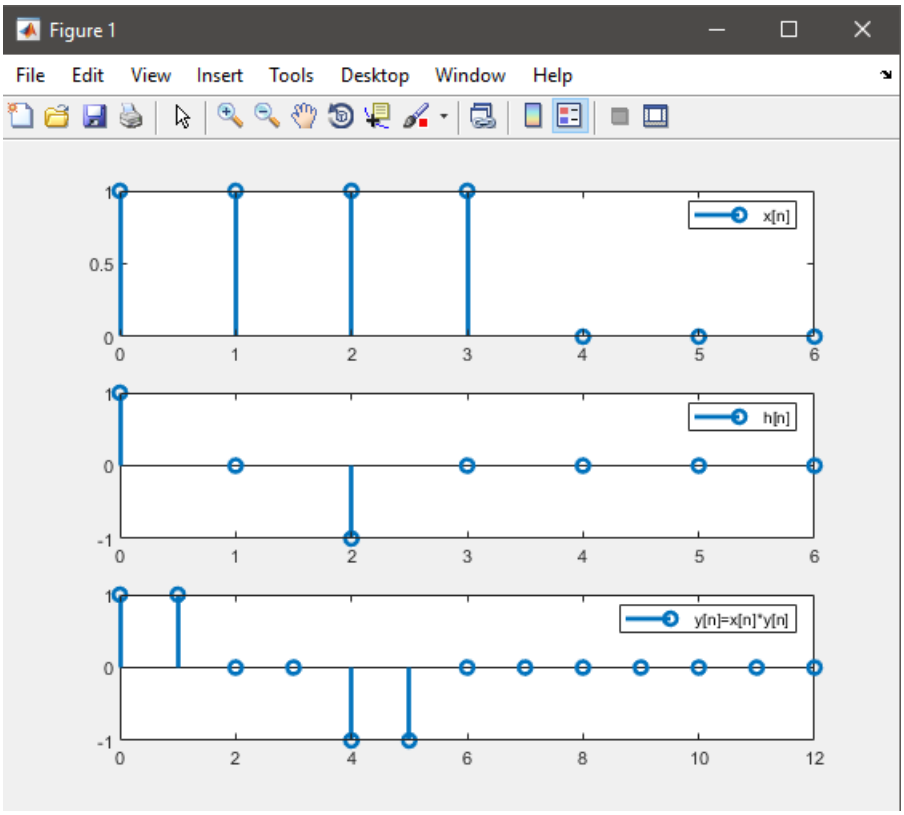


```
%Convolution
y=[ 0.0156 0.05 0.11 0.23 0.48 0.98 1.98 1.97 1.94 1.87 1.75 1.5 1.0];
yn=-6:6;
stem(yn,y,'LineWidth',2)
title('Convolution')
legend('y[n]=x[n]*h[n]')
grid on
```



Task 04: Compute (by both procedures) and graph the convolution $y[n]=x[n]*h[n]$, where $x[n]=u[n]-u[n-4]$ and $h[n]=\delta[n]-\delta[n-2]$.

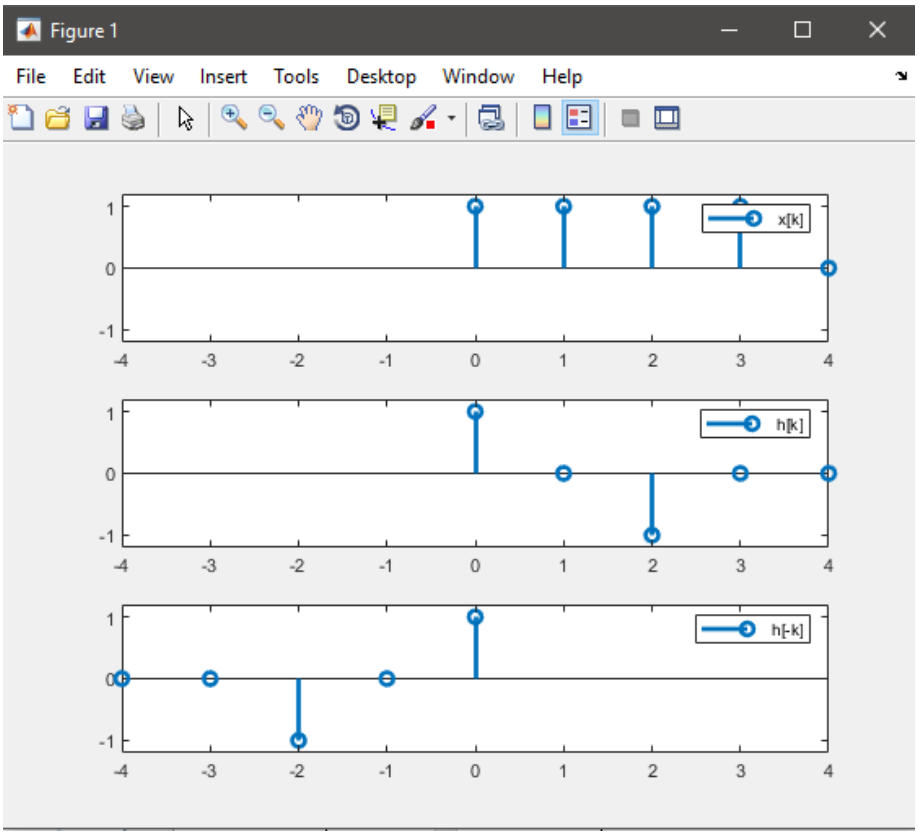
```
N = 0:6;
h = [1 0 -1 0 0 0 0];
x = [1 1 1 1 0 0 0];
subplot(3,1,1)
stem(n,x,'LineWidth',2)
legend('x[n]')

subplot(3,1,2)
stem(n,h,'LineWidth',2)
legend('h[n]')

y = conv(x,h);
subplot(3,1,3)
stem(0:12,y,'LineWidth',2)
legend('y[n]=x[n]*y[n]')
```



Other Method

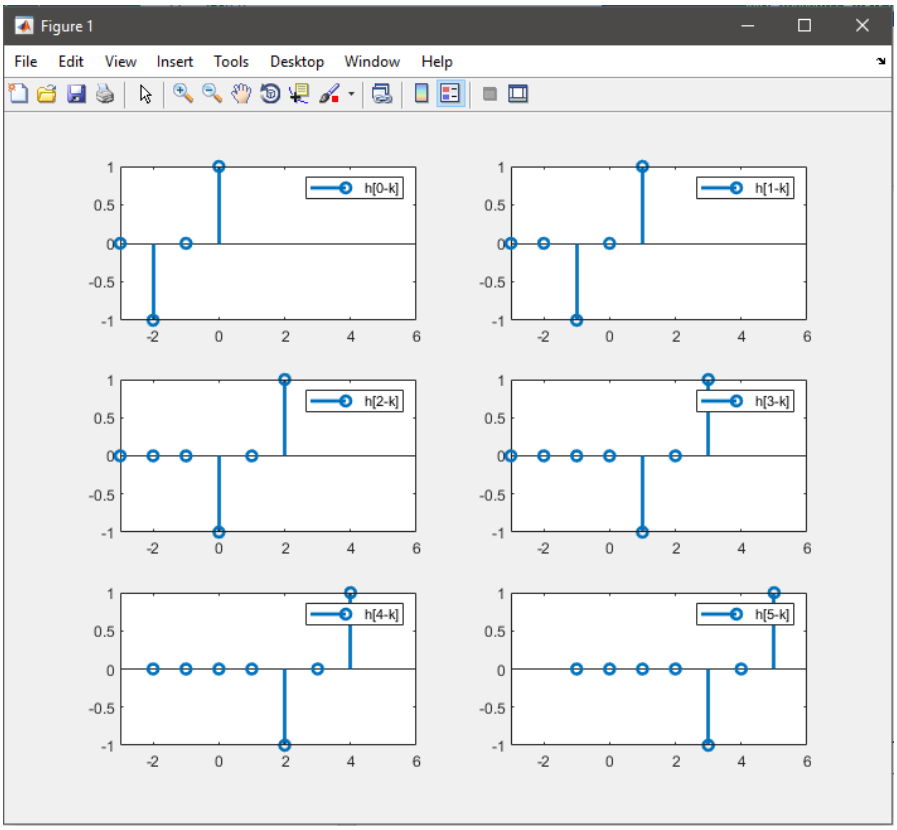
```
k = 0:6;        %changing axis
x = [1 1 1 1 0 0 0];
h = [1 0 -1 0 0 0 0];
subplot(3,1,1)
stem(k,x,'LineWidth',2)
legend('x[k]')
axis([-4 4 -1.2 1.2])

subplot(3,1,2)
```

```
stem(k,h,'LineWidth',2)
legend('h[k]')
axis([-4 4 -1.2 1.2])

subplot(3,1,3)
stem(-k,h,'LineWidth',2)
legend('h[-k]')
axis([-4 4 -1.2 1.2])
```

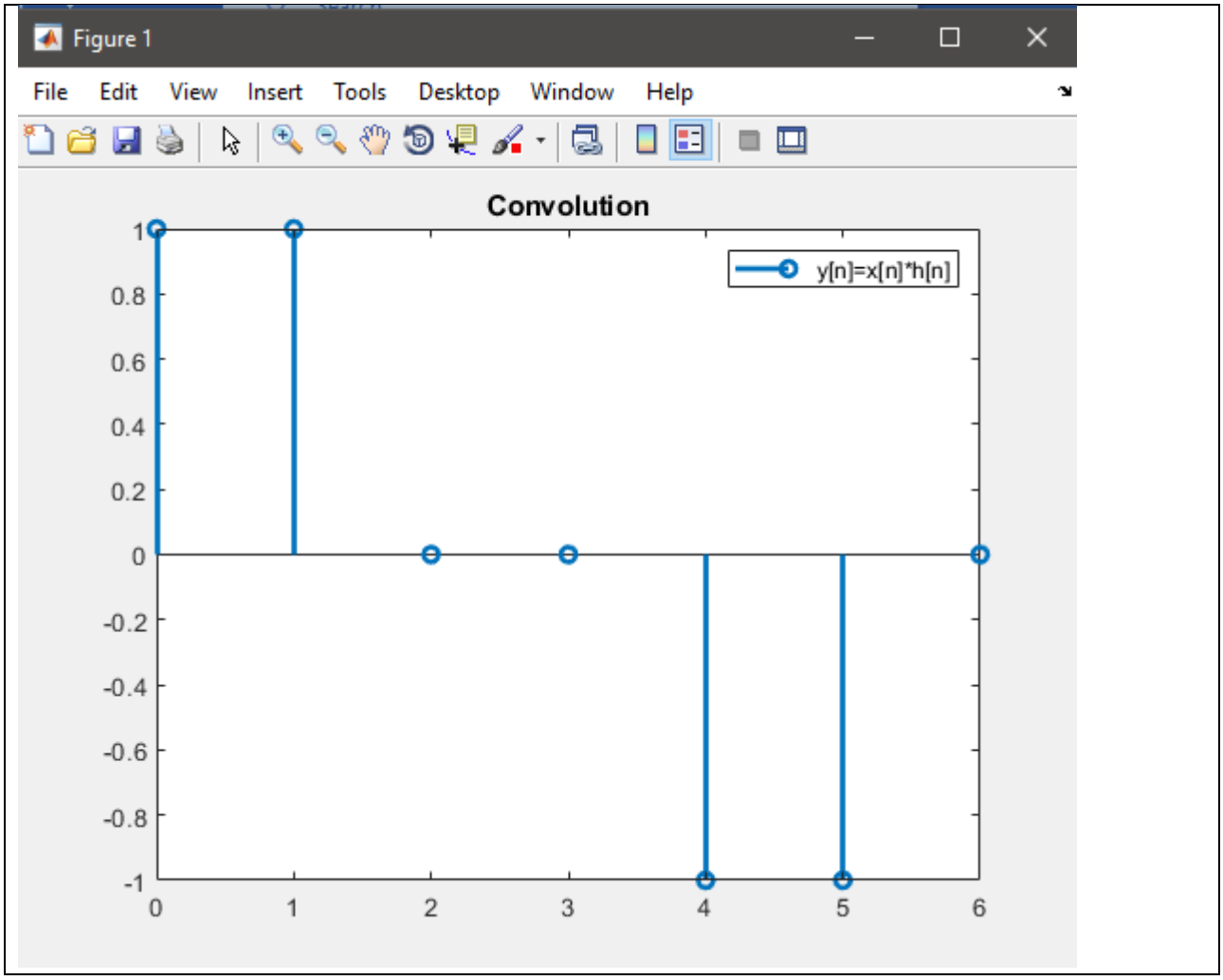



```
n=0;       %delay
subplot(3,2,1)
stem(-k+n,h,'LineWidth',2)
legend('h[0-k]')
xlim([-3 6])


n=1;       %delay
subplot(3,2,2)
stem(-k+n,h,'LineWidth',2)
legend('h[1-k]')
xlim([-3 6])

n=2;       %delay
subplot(3,2,3)
stem(-k+n,h,'LineWidth',2)
legend('h[2-k]')
xlim([-3 6])

n=3;       %delay
subplot(3,2,4)
```

```
stem(-k+n,h,'LineWidth',2)
legend('h[3-k]')
xlim([-3 6])

n=4;        %delay
subplot(3,2,5)
stem(-k+n,h,'LineWidth',2)
legend('h[4-k]')
xlim([-3 6])

n=5;        %delay
subplot(3,2,6)
stem(-k+n,h,'LineWidth',2)
legend('h[5-k]')
xlim([-3 6])
```




```
%Convolution
y = [1 1 0 0 -1 -1 0];
yn = 0:6;
stem(yn,y,'LineWidth',2)
title('Convolution')
legend('y[n]=x[n]*h[n]')
```



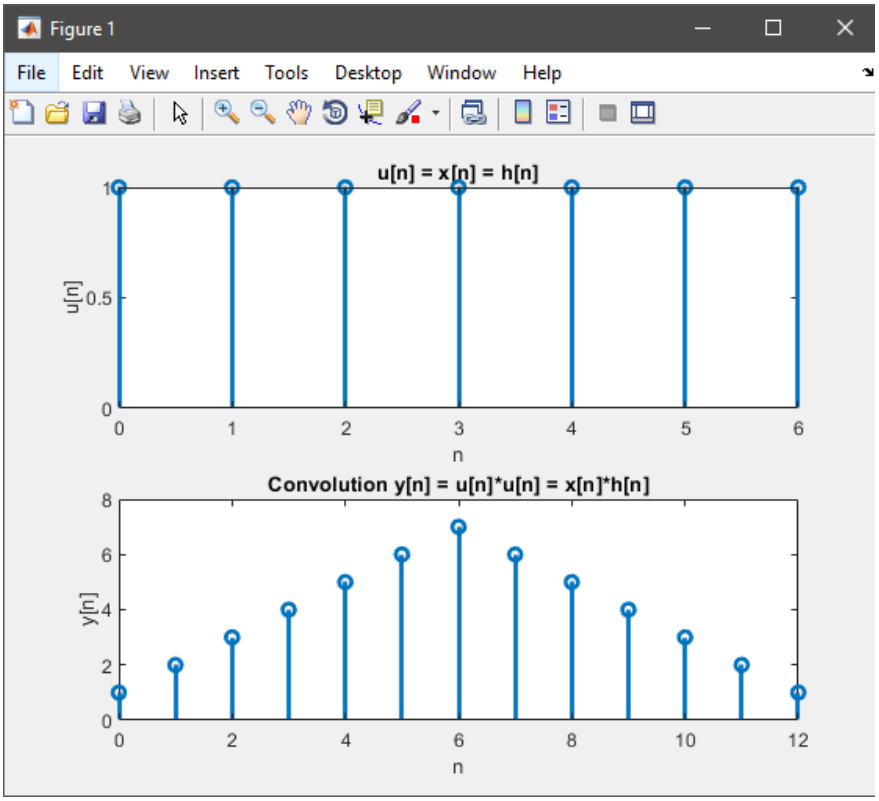
Task 05: Compute (by both procedures) and graph the convolution $y[n]$, where

(a) $y[n] = u[n] * u[n], 0 \le n \le 6$

(b) $y[n] = 3\delta[n-4] * (3/4)^n u[n], 0 \le n \le 5$

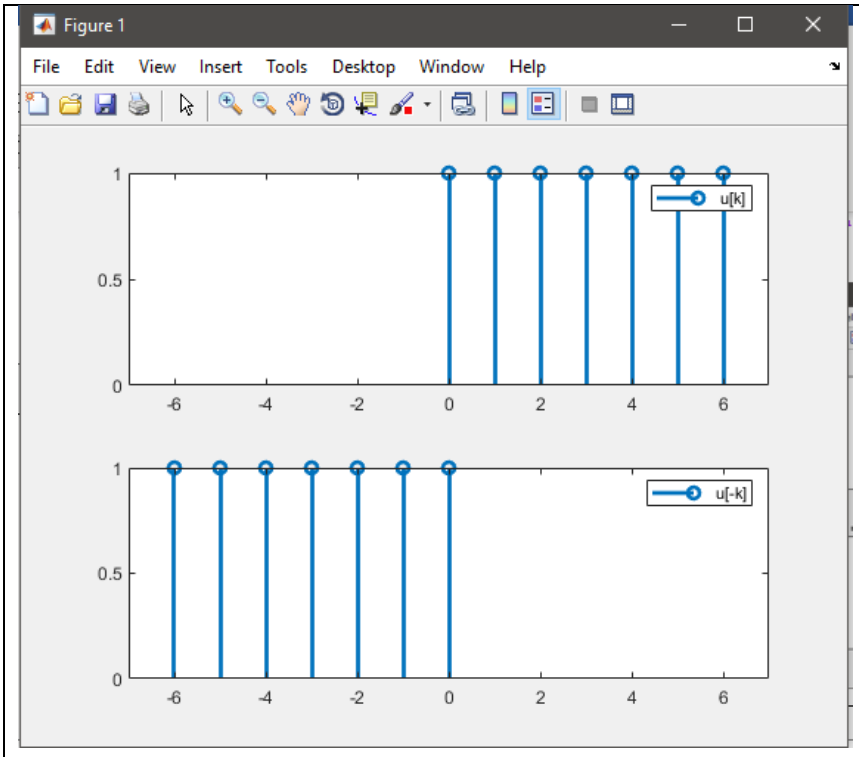
A)

```
n = 0:6;
u = ones(size(n));

subplot(2,1,1)
stem(n,u,'linewidth',2)
title('u[n] = x[n] = h[n]')
xlabel('n')
ylabel('u[n]')

y = conv(u,u);
subplot(2,1,2)
stem(0:12,y,'LineWidth',2)
title('Convolution y[n] = u[n]*u[n] = x[n]*h[n]')
xlabel('n')
ylabel('y[n]')
```

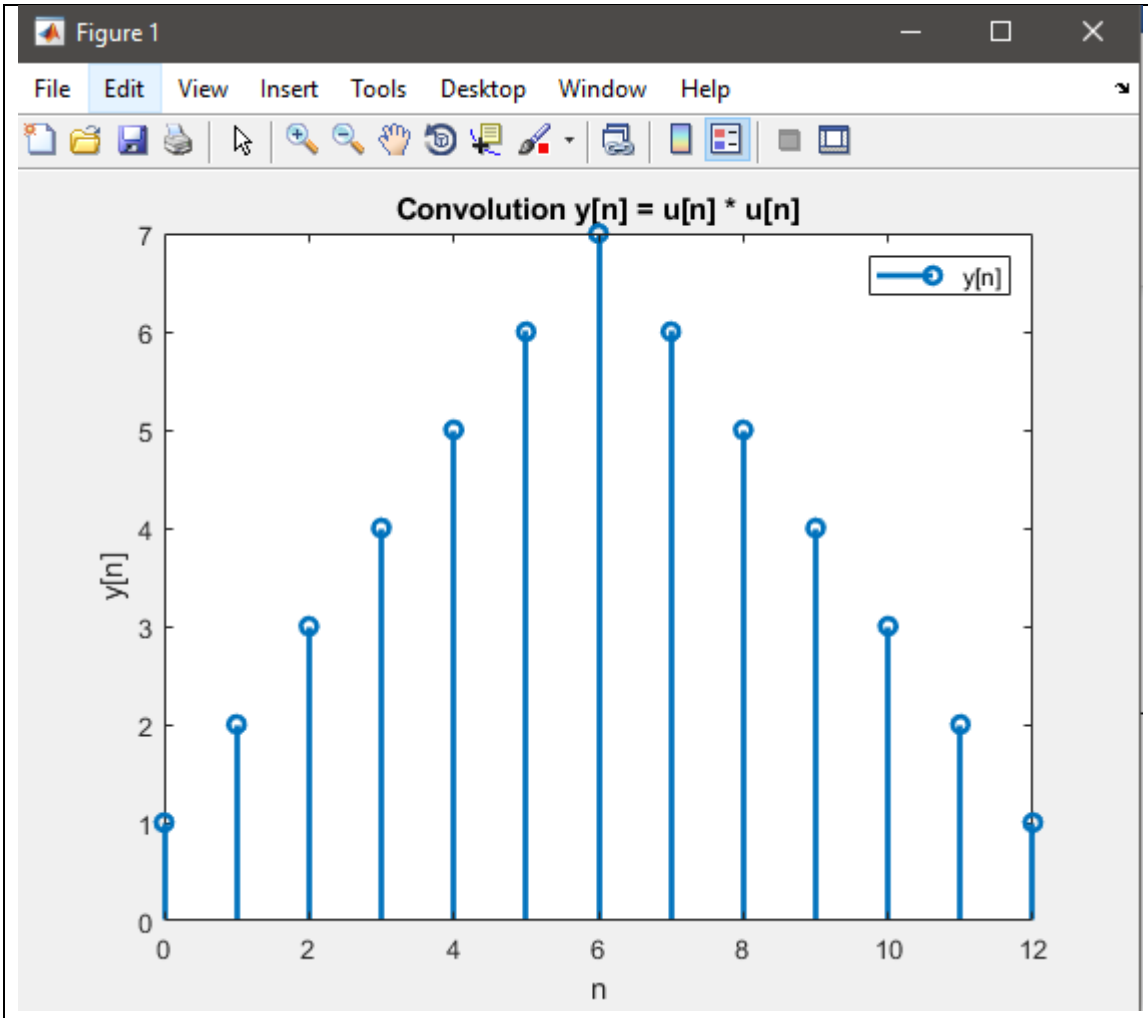


Other Method

```
k = 0:6;
u = ones(size(k));
subplot(2,1,1)
stem(k,u,'LineWidth',2)
legend('u[k]')
xlim([-7 7]);

subplot(2,1,2)
stem(-k,u,'LineWidth',2)
legend('u[-k]')
xlim([-7 7]);
```

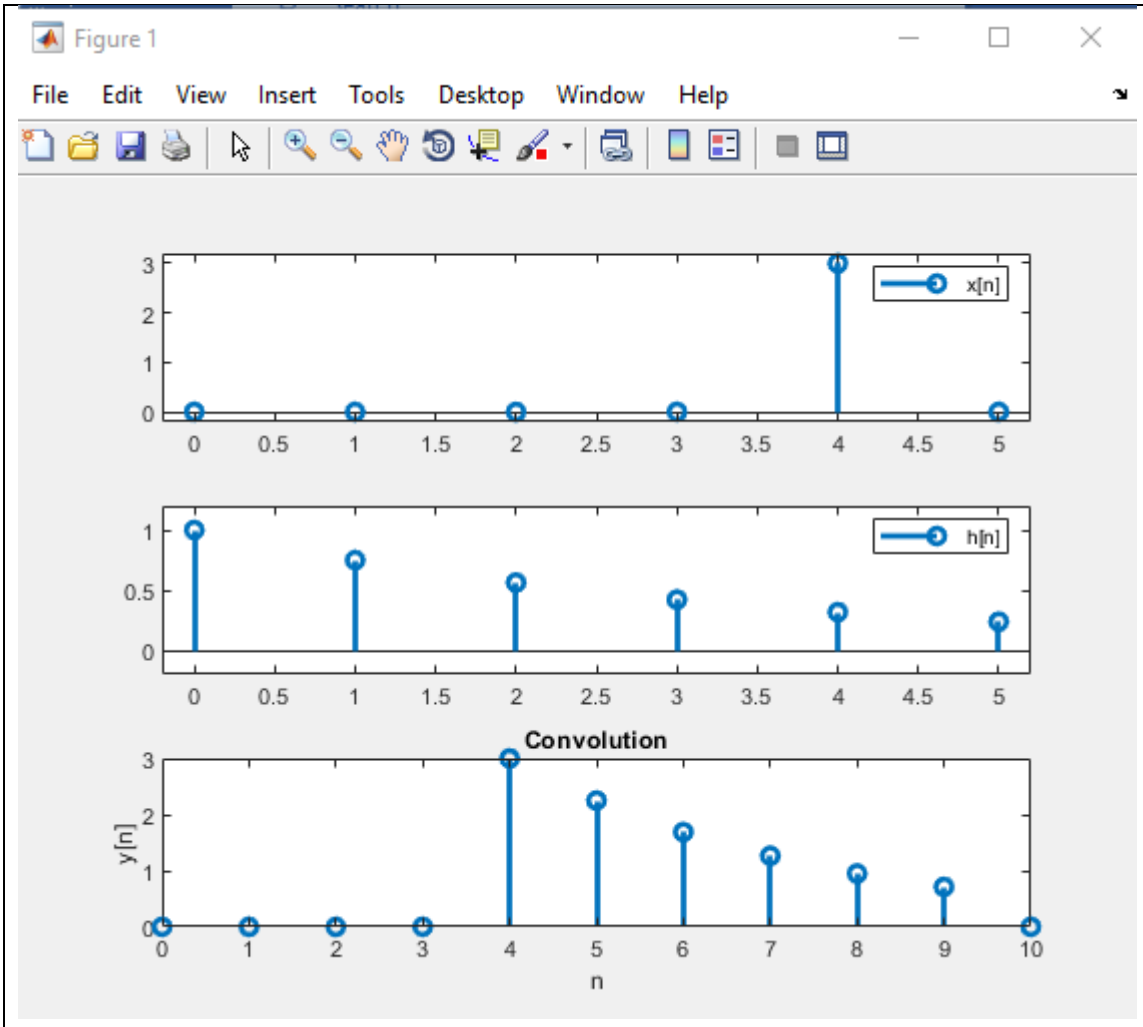



```
%Convolution
y = [1 2 3 4 5 6 7 6 5 4 3 2 1];
yn = 0:12;
stem(yn,y,'LineWidth',2)
legend('y[n]')
title('Convolution y[n] = u[n] * u[n]')
xlabel('n')
ylabel('y[n]')
```



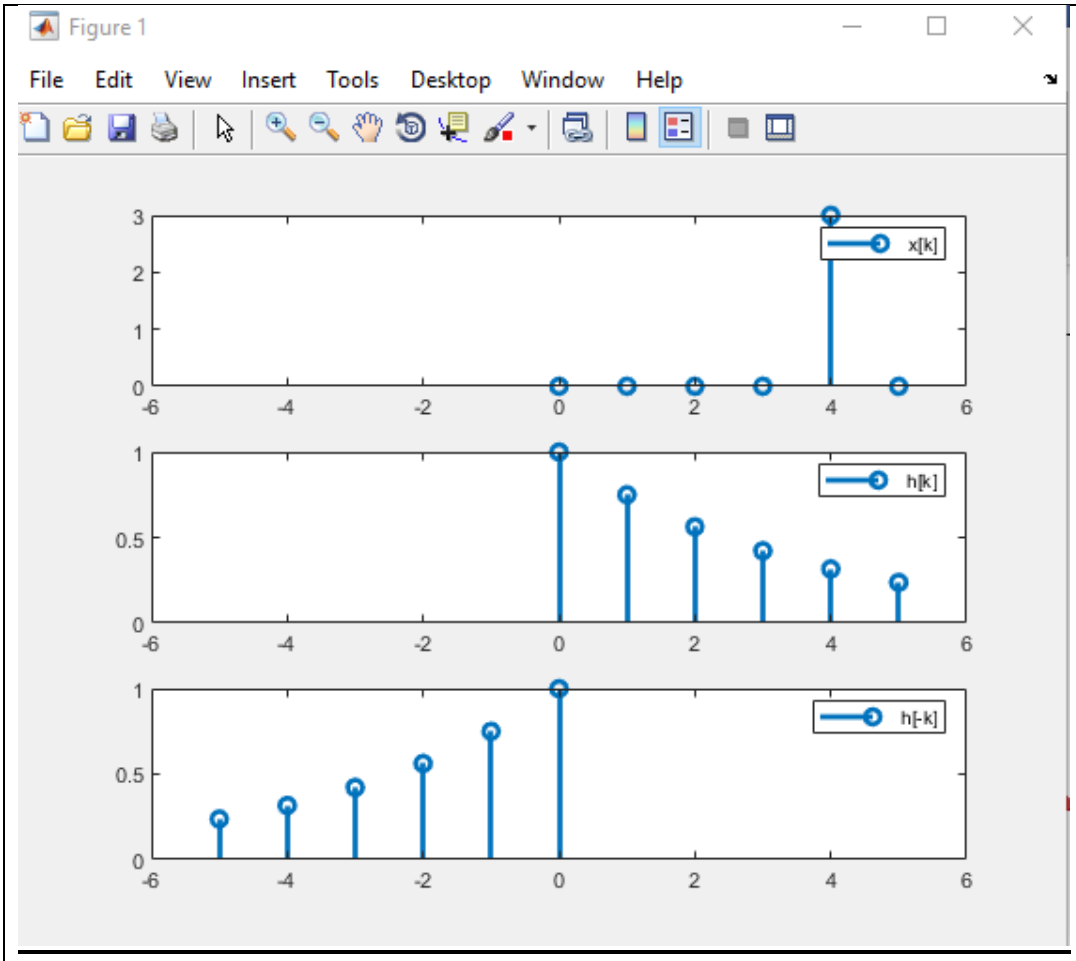
B)

```
n = 0:5;
h = (3./4).^n.*(n>=0);
x = [0 0 0 0 3 0];
subplot(3,1,1)
stem(n,x,'LineWidth',2)
legend('x[n]')
axis([-0.2 5.2 -0.2 3.2])

subplot(3,1,2)
stem(n,h,'LineWidth',2)
legend('h[n]')
axis([-0.2 5.2 -0.2 1.2])


y = conv(x,h);
subplot(3,1,3)
stem(0:10,y,'LineWidth',2)
title('Convolution')
ylabel('y[n]')
xlabel('n')
```



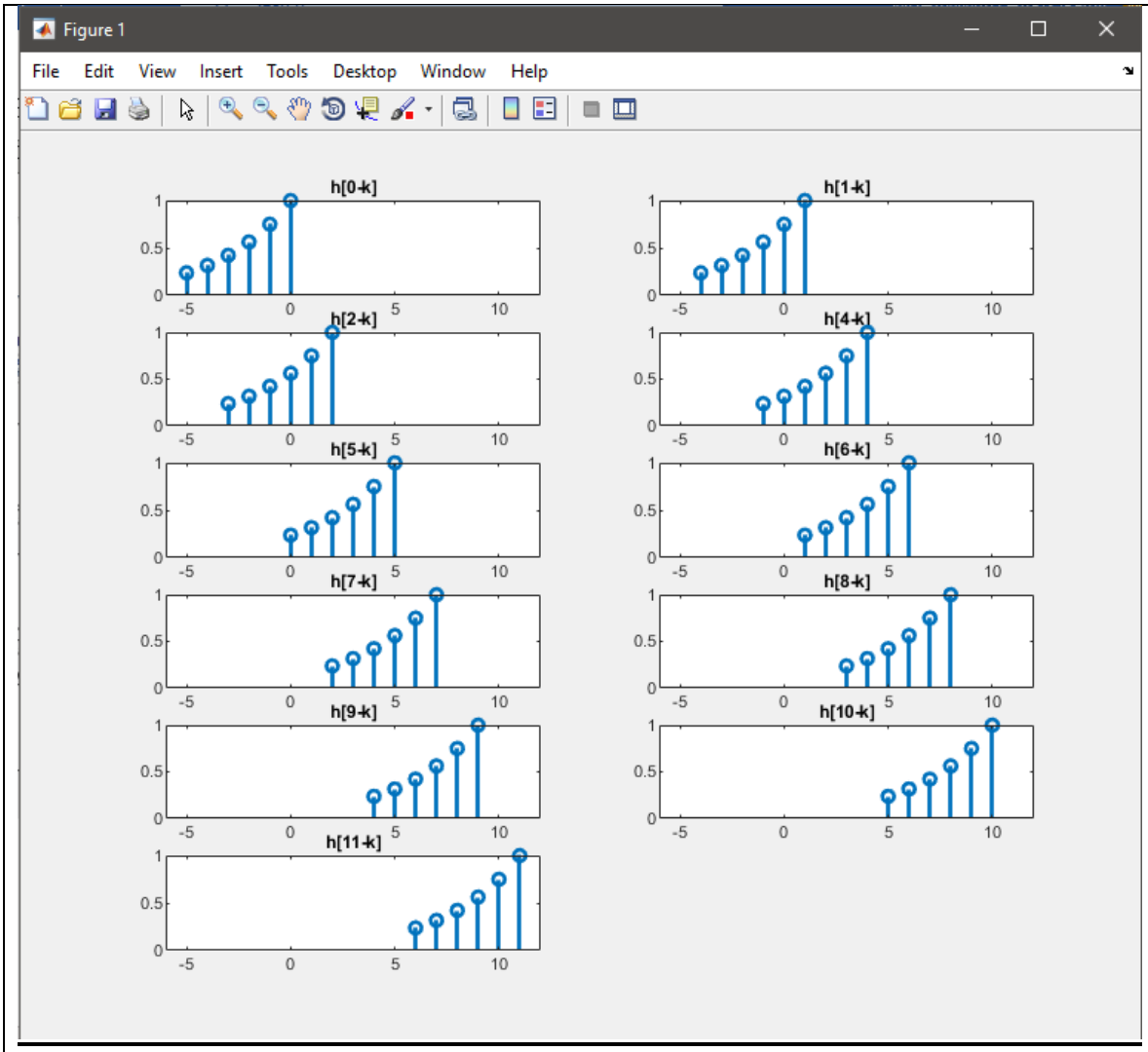
Other Method:

```
k = 0:5;
x = [0 0 0 0 3 0];
h = (3./4).^k.*(k>=0);
subplot(3,1,1)
stem(k,x,'LineWidth',2)
legend('x[k]')
xlim([-6 6])

subplot(3,1,2)
stem(k,h,'LineWidth',2)
legend('h[k]')
xlim([-6 6])

subplot(3,1,3)
stem(-k,h,'LineWidth',2)
legend('h[-k]')
xlim([-6 6])
```




```
n=0;          %delay
subplot(6,2,1)
stem(-k+n,h,'LineWidth',2)
title('h[0-k]')
xlim([-6 12])

n=1;          %delay
subplot(6,2,2)
stem(-k+n,h,'LineWidth',2)
title('h[1-k]')
xlim([-6 12])

n=2;          %delay
subplot(6,2,3)
stem(-k+n,h,'LineWidth',2)
title('h[2-k]')
xlim([-6 12])

n=3;          %delay
subplot(6,2,4)
stem(-k+n,h,'LineWidth',2)
title('h[3-k]')
xlim([-6 12])

n=4;          %delay
subplot(6,2,4)
```

```
stem(-k+n,h,'LineWidth',2)
title('h[4-k]')
xlim([-6 12])

n=5;          %delay
subplot(6,2,5)
stem(-k+n,h,'LineWidth',2)
title('h[5-k]')
xlim([-6 12])

n=6;          %delay
subplot(6,2,6)
stem(-k+n,h,'LineWidth',2)
title('h[6-k]')
xlim([-6 12])

n=7;          %delay
subplot(6,2,7)
stem(-k+n,h,'LineWidth',2)
title('h[7-k]')
xlim([-6 12])

n=8;          %delay
subplot(6,2,8)
stem(-k+n,h,'LineWidth',2)
title('h[8-k]')
xlim([-6 12])

n=9;          %delay
subplot(6,2,9)
stem(-k+n,h,'LineWidth',2)
title('h[9-k]')
xlim([-6 12])

n=10;          %delay
subplot(6,2,10)
stem(-k+n,h,'LineWidth',2)
title('h[10-k]')
xlim([-6 12])

n=11;          %delay
subplot(6,2,11)
stem(-k+n,h,'LineWidth',2)
title('h[11-k]')
xlim([-6 12])
```




```
%Convolution
y = [0 0 0 0 3 2.2 1.8 1.2 1 0.8 0];
yn = 0:10;
stem(yn,y,'LineWidth',2),grid on
legend('y[n]')
title('Convolution')
```

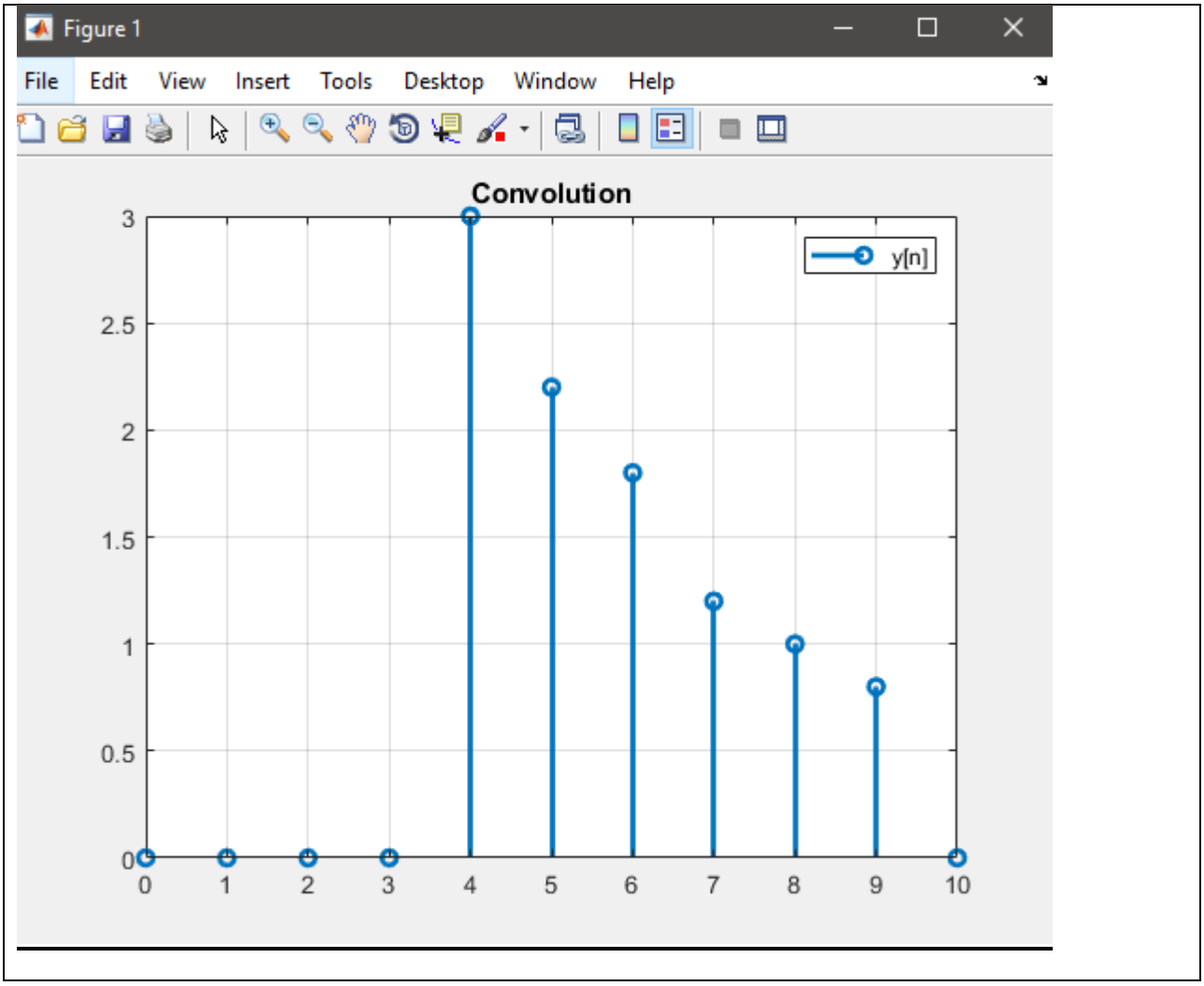


Figure 1
File Edit View Insert Tools Desktop Window Help
Convolution
y[n]
3
2.5
2
1.5
1
0.5
0
0 1 2 3 4 5 6 7 8 9 10

Post-Lab Tasks

Critical Analysis / Conclusion

In this lab we learnt how to perform convolution in Matlab in two different ways. Firstly, we used matlab built-in function conv() which takes two argument i.e. the input signal and Impulse response of the signal .

Secondly, we use the conventional method to convolve the two signals in which we flip one of the signals in first step then we shift the flipped signal until there is an overlap between the input and impulse response.

Lab Assessment		
Pre-Lab	**/1**	**/10**
In-Lab	**/5**	
Critical Analysis	**/4**	
Instructor Signature and Comments		

LAB # 07

Analysis of Continuous Time LTI Systems using Convolution Integral

Lab 07- Analysis of Continuous Time LTI Systems using Convolution Integral

Pre-Lab Tasks

7.1 Impulse Response:

The meaning of impulse response of the system is easily derived if one considers the terms that it is named from. The terms, 'response', denotes the output of a system, while the term, 'impulse' denotes that the input signal applied to the system is the unit impulse or Dirac Delta function. Hence, the impulse response of a causal linear and time invariant continuous time system is the output when the Dirac Delta function is the input applied to the system. Mathematically, impulse response of the system is denoted by

$$h(t) = S\{\delta(t)\}$$

7.2 Continuous-Time Convolution:

The impulse response of a linear time-invariant system completely specifies the system. More, specifically, if the impulse response of a system is known one can compute the system output for any input signal.

The response (or output) of a system to any input signal is computed by the convolution of the input signal with the impulse response of the system.

Suppose that $y(t)$ denotes the output of the system, $x(t)$ is the input signal, and $h(t)$ is the impulse response of the system. The mathematical expression of the convolution relationship is

$$y(t) = x(t) * h(t)$$

7.2.1 Computation of Convolution:

In order to calculate the convolution between two signals, a specific (and not exactly trivial) computational procedure has to be followed. The convolution computational procedure is introduced through an example.

Example

A linear time-invariant signal is described by the impulse response

$$h(t) = \begin{cases} 1-t & 0 \le t \le 1 \\ 0 & elsewhere \end{cases}$$

Calculate the response of the system to the input signal

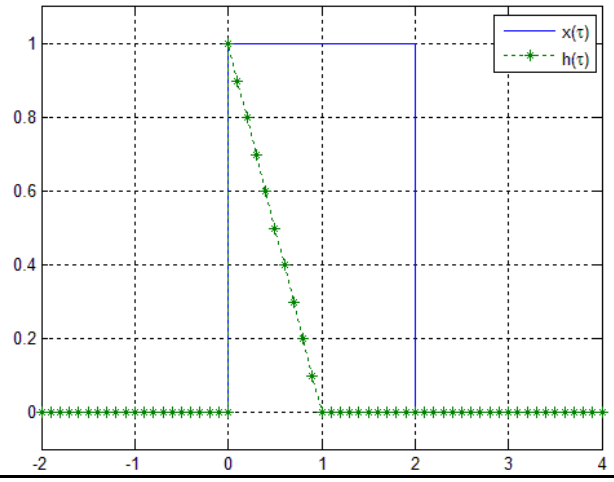
$$x(t) = \begin{cases} 1 & 0 \le t \le 2 \\ 0 & elsewhere \end{cases}$$

In order to compute the convolution between $x(t)$ and $h(t)$ the computational procedure is implemented in the following steps.

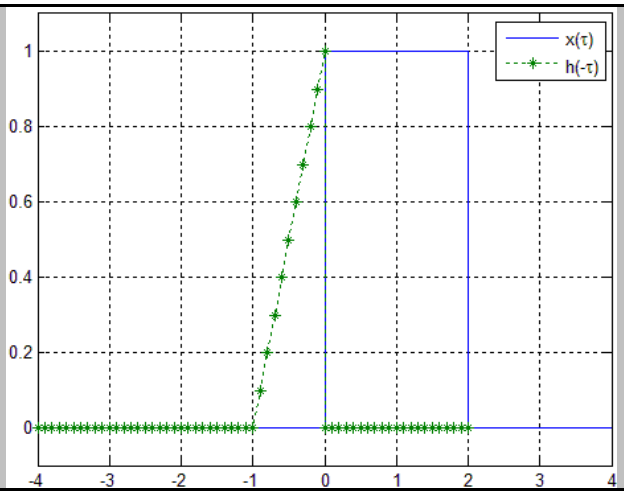
Step 1: Time impulse response signals are plotted in the τ -axis; that is, t is replaced with τ , or in other words the signals

$$h(\tau)=\begin{cases}1-\tau & 0\le\tau\le 1\\ 0 & elsewhere\end{cases} \text{ and } x(\tau)=\begin{cases}1 & 0\le\tau\le 2\\ 0 & elsewhere\end{cases}$$

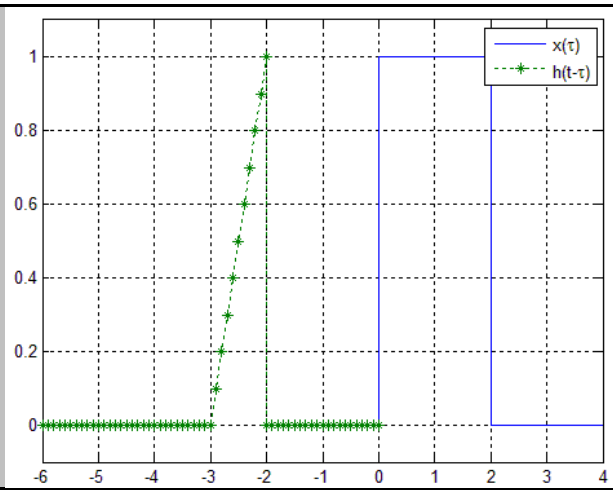
are defined and plotted.

Commands	Results	Comments
tx1=-2:0.1:0; **tx2=0:0.1:2;** **tx3=2:0.1:4;** **tx=[tx1 tx2 tx3];** **x1=zeros(size(tx1));** **x2=ones(size(tx2));** **x3=zeros(size(tx3));** **x=[x1 x2 x3];**		The signals $x(\tau)$ and $h(\tau)$ are defined in the usual way of constructing two part functions. The input signal $x(\tau)$ is plotted with solid line, while the impulse response signal $h(\tau)$ is plotted with dotted line and asterisks. Both the signals are defined in the time interval $-2\le\tau\le 4$. Notice that the signals are plotted in the τ -axis and how Greek letters are inserted into the legend.
Th1=-2:0.1:0; **th2=0:0.1:1;** **th3=1:0.1:4;** **th=[th1 th2 th3];** **h1=zeros(size(th1));** **h2=1-th2;** **h3=zeros(size(th3));** **h=[h1 h2 h3];** **plot(tx,x,th,h,':*')** **ylim([-0.1 1.1])** **legend('x(\tau)','h(\tau)')** **grid**		

Step2: The second step is known as reflection. One of the two signals is selected (in this example $h(\tau)$ is chosen, but $x(\tau)$ could have been also selected) and its symmetric with respect to the vertical axis, i.e., $h(-\tau)$ is plotted.

Commands	Results
plot(tx,x,-th,h,':*') **ylim([-0.1 1.1])** **legend('x(\tau)','h(-\tau)')** **grid**	

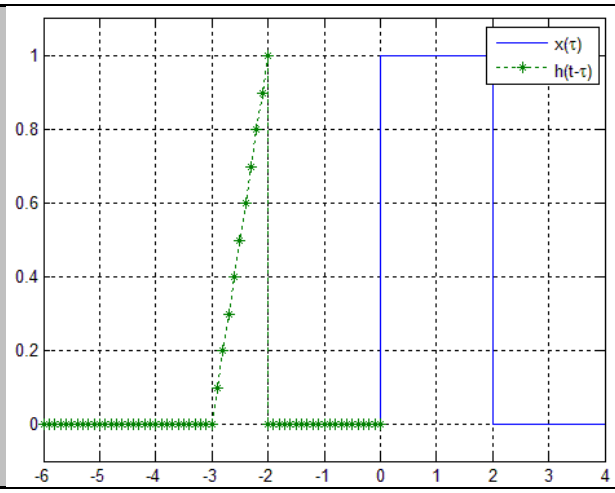
Step 3: The third step is shifting. The signal $h(-\tau)$ is shifted by t; that is, the signal $h(t-\tau)$ is plotted. Note that t is constant, as the variable of the defined signals is variable τ.

Commands	Results	Comments
t=-2; **plot(tx,x,-th+t,h,':*')** **ylim([-0.1 1.1])** **legend('x(\tau)','h(t-\tau)')** **grid**		The signal $h(t-\tau)$ is plotted for $t=-2$. Notice that $h(t-\tau)$ is shifted by 2 units to the left compared to $h(-\tau)$.

A very useful (for future computations) observation is the point in the τ-axis where the right end of $h(t-\tau)$ is. The right end $h(t-\tau)$ for $t=-2$; that is, the right end of $h(-2-\tau)$ is at the point $\tau=-2$. In the previous graph, it has been observed that the right end of $h(-\tau)$, that is, the right end of $h(0-\tau)$ is at $\tau=0$. Hence, we conclude that in general case the right end of $h(t-\tau)$ is the point $\tau=t$. On the other hand, the left end of $h(t-\tau)$ is at $\tau=t-T$, where T is the duration of $h(t-\tau)$. In this example we have $T=1$.

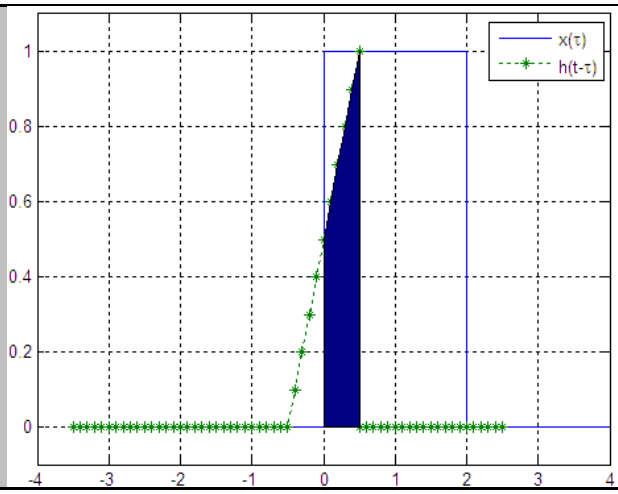
Step 4: The fourth step is sliding step. The value of the output signal $y(t)$ at time t, that is, the value of convolution between the input signal and impulse response signal at time t depends on overlap between $x(\tau)$ and $h(t-\tau)$ at time t. In order to compute the convolution for all t, we have to slide the signal $h(t-\tau)$ from $-\infty$ to $+\infty$ and to derive the kind (or stage) of overlap in reference to the value of t. Note that $x(t)$ remains still.

- First Stage: Zero Overlap

Commands	Results	Comments
t=-2; plot(tx,x,-th+t,h,':*') ylim([-0.1 1.1]) legend('x(\tau)','h(t-\tau)') grid		For $t<0$ (in this figure $t=-2$) the signal $h(t-\tau)$ and $x(\tau)$ do not overlap. Thus the output is $y(t)=0$.

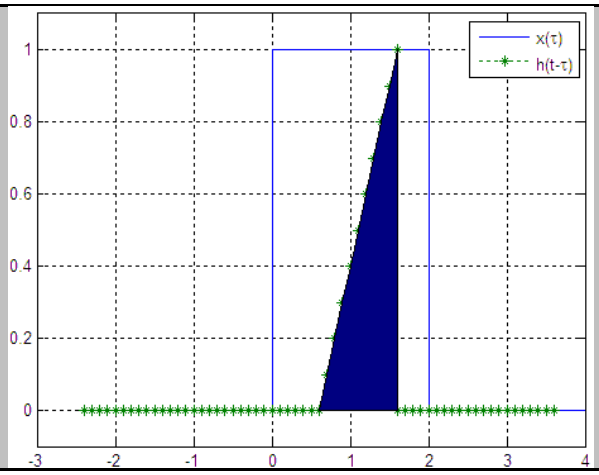
- Second Stage: Partial Overlap

Commands	Results	Comments

t=0.5;
plot(tx,x,-th+t,h,':*'),grid on
ylim([-0.1 1.1])
legend('x(\tau)','h(t-\tau)')
% the following code
% produce the shadowed
% area plot
T=1;
r=0:0.1:t;
a=1/T*r+1-t/T;
hold on; area(r,a);
hold off;
		For $0<t<1$, the signal $h(t-\tau)$ is entering at $x(\tau)$. The two signals are partially overlapped. The overlapped part is shadowed.

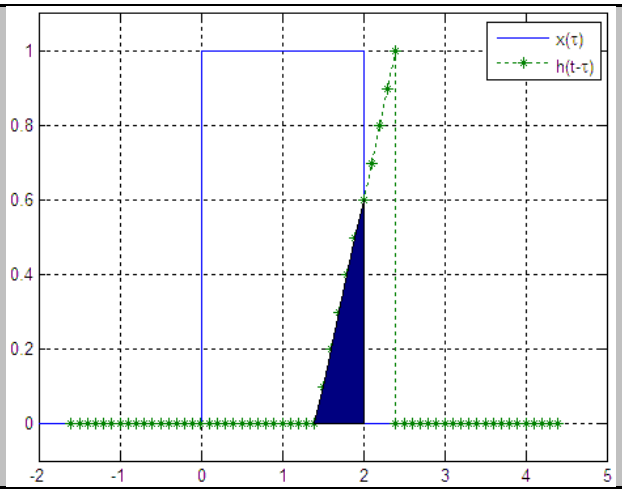
- Third Stage: Complete Overlap

Commands	Results	Comments

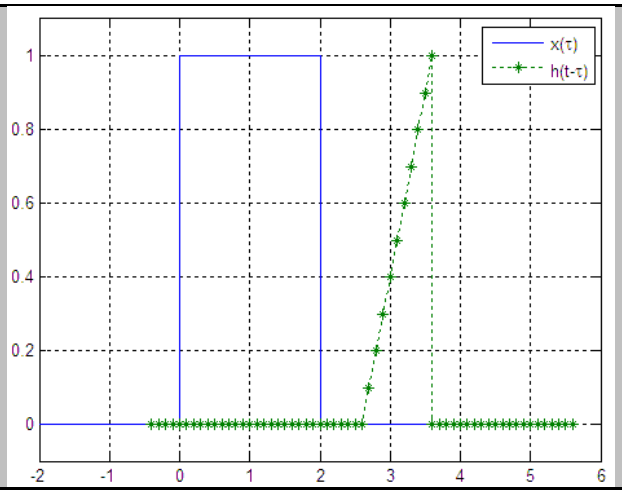
t=1.6;
plot(tx,x,-th+t,h,':*'),grid on
ylim([-0.1 1.1])
legend('x(\tau)','h(t-\tau)')
% the following code
% produce the shadowed
% area plot
T=1;
r=t-T:0.1:t;
a=1/T*r+1-t/T;
hold on; area(r,a);
hold off;
		For $1<t<2$, the signals $h(t-\tau)$ and $x(\tau)$ are completely overlapped.

- Fourth Stage: Exit-partial overlap

Commands	Results	Comments

t=2.4;
plot(tx,x,-th+t,h,':*'),grid on
ylim([-0.1 1.1])
legend('x(\tau)','h(t-\tau)')
% the following code
% produce the shadowed
% area plot
T=1;
r=t-T:0.1:2;
a=1/T*r+1-t/T;
hold on; area(r,a);
hold off;
		For $2<t<3$, the signal $h(t-\tau)$ exits $x(\tau)$. The two signals are partially overlapped.

- Fifth Stage: Zero Overlap

Commands	Results	Comments
t=3.6; plot(tx,x,-th+t,h,':*'),grid on ylim([-0.1 1.1]) legend('x(\tau)','h(t-\tau)')		For $t>3$, he signal $h(t-\tau)$ does not overlap with $x(\tau)$ do not overlap. Thus the output is $y(t)=0$.

Step 5: Specification of limits and calculation of the convolution integral. Having derived the time intervals for various stages of overlap, the convolution integral is computed separately for each stage. Before computing the integrals the integration limits have to be specified. Recall that the right end of $h(t-\tau)$ is at point $\tau=t$, while the left end of $h(t-\tau)$ is at point $\tau=t-T=t-1$.

1. For $t<0$, the two signals do not overlap (zero-overlap stage). Thus the response of the system is $y(t)=0$.
2. For $0<t<1$, the two signals start to overlap (entry stage-partial overlap). The limits of the integral are the limits that specified the shadowed area. Hence $y(t)=\int_0^t x(\tau)h(t-\tau)d\tau$. The input signal $x(\tau)$ is given by $x(\tau)=1$, while the impulse response signal $h(t-\tau)$ is given by $h(t-\tau)=1-(t-\tau)=1-t+\tau$. The expression of $h(t-\tau)$ is derived by substituting t with $t-\tau$ in $h(t)$. Thus the integral that has to be calculated is $y(t)=\int_0^t 1(1-t+\tau)d\tau=\int_0^t(1-t+\tau)d\tau$

Commands	Results	Comments
syms t r **f=1-t+r;** **y=int(f,r,0,t)**	y-t-1/2*t^2	The integral is computed and the response of the system at the entry stage is $y(t)=t-t^2/2, 0<t<1.$

3. For $1<t<2$, the two signals overlap completely (complete overlap stage). The only difference to the previous calculation is the integral limits. In this case, the output is given by $y(t)=\int_{t-1}^{t}(1-t+\tau)d\tau$.

Commands	Results	Comments
y=int(f,r,t-1,t) **simplify(y)**	Y=1-t+1/2*t^2-1/2*(t-1)^2 ans=1/2	The integral is computed and the result is simplified. The response of the system at complete overlap is $y(t)=0.5, 1<t<2.$

4. For $2<t<3$, the two signals overlap partially (exit stage). Thus, the output is given by $y(t)=\int_{t-1}^{2}(1-t+\tau)d\tau$.

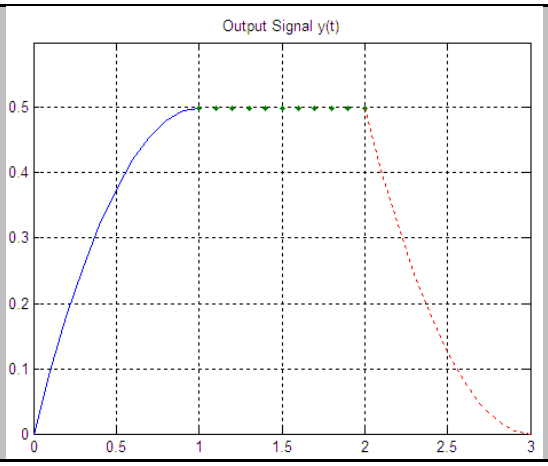
Commands	Results	Comments
y=int(f,r,t-1,2)	Y=5-t-t*(3-t)-1/2*(t-1)^2	The response of the system at exit stage is $y(t)=(3-t)^2/2, 2<t<3.$

5. For $t>3$, there is no overlap; thus the output is $y(t)=0, t>3$.

Combining all the derived results, we conclude that the response of the system with impulse response $h(t)=1-t, 0\le t\le 1$ to the input signal $x(t)=1, 0\le t\le 2$ is

$$y(t)=\begin{cases} t-t^2/2 & 0\le t\le 1 \\ 1/2 & 1\le t\le 2 \\ (3-t)^2/2 & 2\le t\le 3 \\ 0 & t<0 \text{ and } t>3 \end{cases}$$

Finally, the output $y(t)$ is plotted in MATLAB according to the technique of plotting multipart function.

Commands	Results
t1=0:0.1:1; **t2=1:0.1:2;** **t3=2:0.1:3;** **y1=t1-(t1.^2)/2;** **y2=0.5*ones(size(t2));** **y3=0.5*((3-t3).^2);** **plot(t1,y1,t2,y2,'.',t3,y3,':'),grid on** **ylim([0 0.6])** **title('Output Signal y(t)')**	

7.2.2 The Command conv:

The computational process followed in the previous subsection is the analytical way of deriving the convolution between two signals. In MATLAB, the command conv allows the direct computation of the convolution between two signals. In order to illustrate the conv command we consider the previously used example, namely the impulse response signal is given by $h(t)=1-t, 0\le t\le 1$ and an input signal given by $x(t)=1, 0\le t\le 2$. There are four rules that have to be applied for successful computation of the convolution between two continuous-time signals.

- First rule: Two signals (input and impulse response) should be defined in the same time interval. Hence both the signals are defined in the time interval $a\le t\le b$, where $t=a$ is the first time instance that at least one of the signals $x(t)$ or $h(t)$ is not zero and $t=b$ is the last time instance that at least one of the two signals is not zero. In our example, the time interval is $0\le t\le 2$. Thus the input signal $x(t)=1, 0\le t\le 2$ and the impulse response signal is expressed as

$$h(t)=\begin{cases}1-t & 0\le t\le 1\\ 0 & 0\le t\le 2\end{cases}$$

- Second rule: When a signal consists of multiple parts, the time intervals in which each part is defined must not overlap. Therefore, the impulse response signal is defined as

$$h(t)=\begin{cases}1-t & 0\le t\le 1\\ 0 & 1< t\le 2\end{cases}$$

Note that the equality at $t=1$ is placed only in the upper part.

Commands	Comments
step=0.01;	The time step has to be quite small in order to approximate accurately the continuous time signal.
T=0:step:2; **x=ones(size(t));**	The input signal $x(t)=1, 0\le t\le 2$ is defined.
T1=0:step:1; **t2=1+step:step:2;**	The intervals of two parts of $h(t)$ are defined in such a way that they do not overlap. More specifically, vector $t2$ is defined from the instance 1+step, i.e., is defined from the time instance 1.01 (second rule). Also notice that the time step is used at the definition of the two signals must be same.
H1=1-t1; **h2=zeros(size(t2));** **h=[h1 h2];**	The impulse response $h(t)$ is defined in the wider time interval, namely, in the same interval where the input $x(t)$ is defined (first rule).

Having defined the input and impulse response signals the response of the system can be computed by convoluting the two signals. The convolution is implemented by the command y=conv(x,h). However, there is still one detail that needs to be addressed and is described at the next rule.

- Third rule: The output of the conv command has to be multiplied with time step used in the definition of the input and impulse response signals, in order to correctly compute the output of the system. The rule emerges from the fact that the convolution integral is approximated by sum in MATLAB.

Commands	Comments
y=conv(x,h)*step;	The response $y(t)$ of the system is computed by convoluting the input signal $x(t)$ with the impulse response signal $h(t)$ and multiplying the result with the time step (third rule).

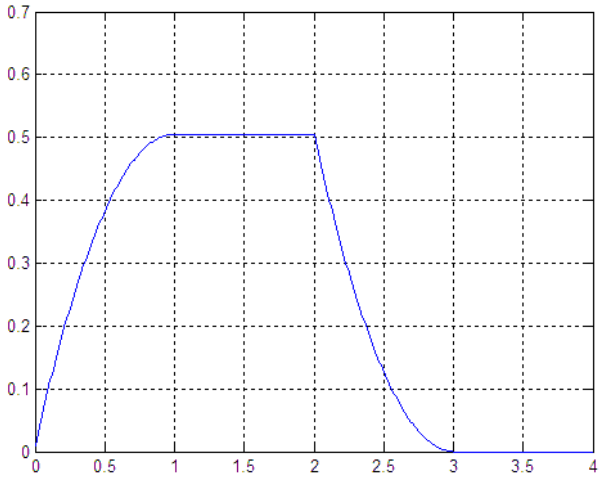
The response of the system is now computed and the only thing left to do is to plot it. However, the number of elements of the output vector y is not equal to the number of elements of vectors x or h.

The precise relationship is length(y)=length(x)+length(h)-1

Commands	Results	Comments
length(y) **length(x)** **length(h)**	ans=401 ans=201 and=201	Indeed the relationship length(y)=length(x)+length(h)-1 is true.

To overcome this, the fourth rule must be applied.

- Fourth rule: The output of the system is plotted in the double time interval of the one in which the output and impulse response signals are defined.

Commands	Results	Comments
ty=0:step:4;		The time interval in which the output signal y will be plotted is $0 \le t \le 4$, namely, it is double from time interval where the vectors x and h are defined.
Plot(ty,y),grid on		The response if the system $y(t)$ computed from the convolution between the input $x(t)$ and the impulse response $h(t)$ is plotted.

The output signal that was obtained through the MATLAB implementation (by using the command conv) is the same as the one derived with the analytical approach in section 7.2.1. In this point the necessity for applying the fourth rule must be already clear. In case if first and second rule was not followed it would not be possible to apply the fourth rule hardening the proper plotting of the output of the system.

7.2.3 Deconvolution:

Suppose that the impulse response of the system $h(t)$ and output of the system is $y(t)$ are available and we want to compute the input signal $x(t)$ that was applied to the system in order to generate the output $y(t)$. The process called deconvolution and is implemented in MATLAB by using the command deconv. The syntax is x=deconv(y,h), where x is the input vector, h is the impulse response vector, and y is the system response vector.

The deconvolution process is also useful for determining the impulse response of a system if the input and output signals are known. In this case the command is h=deconv(y,x).

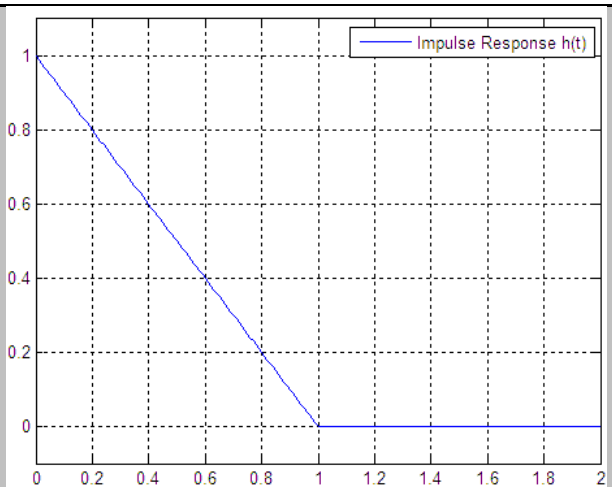
Remark

The commutativity property is not valid for deconv commad; hence, the output signal must be the first input argument of the command deconv.

Example:

We will use the same signals used in the previous example. Therefore, the problem is to compute the impulse response $h(t)$ of a system when the response of the system to the input $x(t)=1, 0\le t\le 2$ is the signal $y(t)$, which is depicted in the previous figure.

The signals $x(t)$ and $y(t)$ and the time t are already defined in the previous example, so we can directly use them to compute the impulse response $h(t)$.

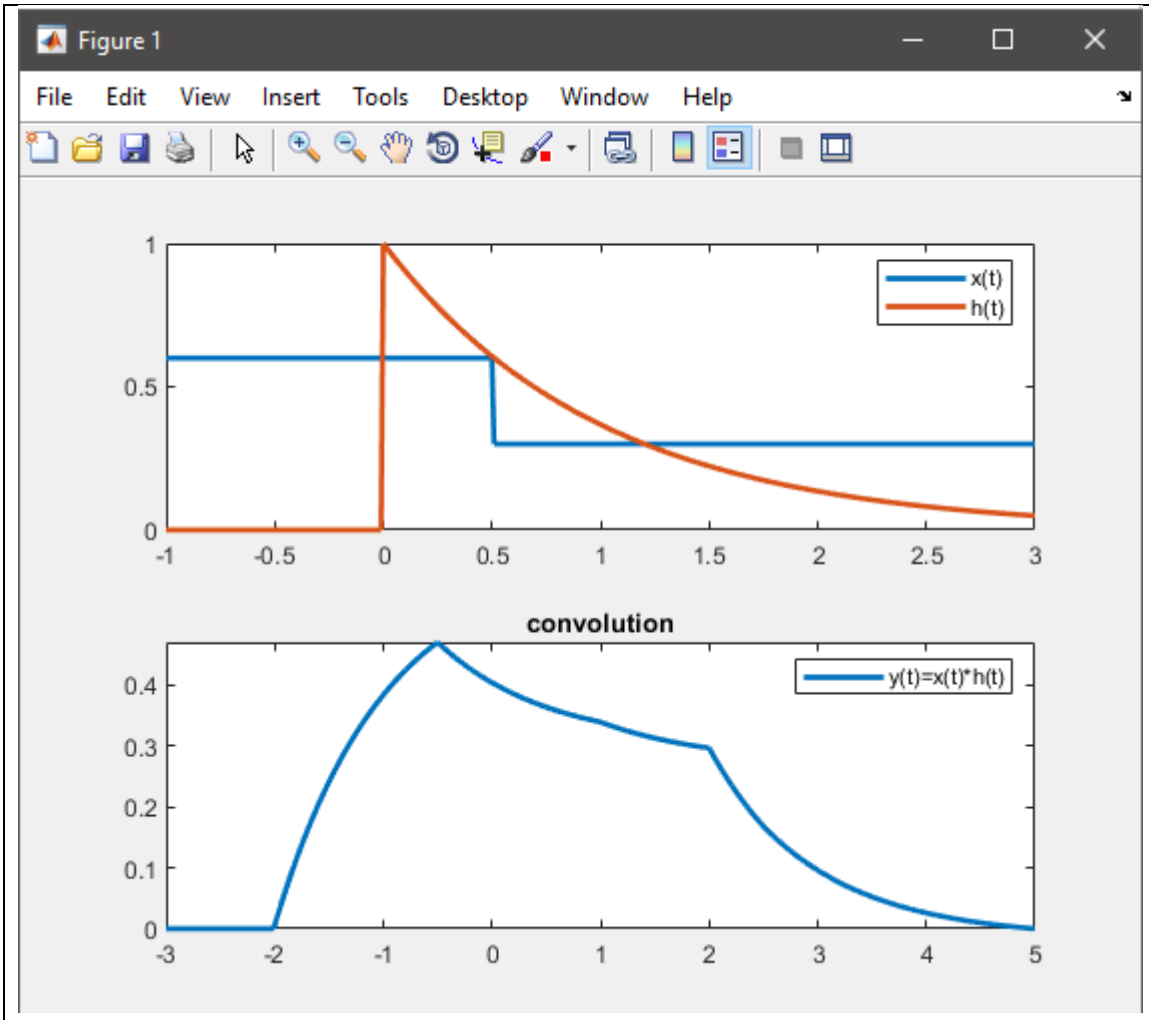
Commands	Results	Comments
hh=deconv(y,x)*(1/step); **plot(t,hh),grid on** **ylim([-0.1 1.1])** **legend('Impulse Response h(t)')**		The impulse response $h(t)$ is computed by multiplying the outcome of the command deconv by the quantity (1/step) as deconvolution is the inverse operation of convolution.

In-Lab Tasks

Task 01: Suppose that a LTI system is described by impulse response $h(t) = e^{-t}u(t)$. Compute the response of the system (by both methods) to the input signal

$$x(t) = \begin{cases} 0.6 & -1 \le t \le 0.5 \\ 0.3 & 0.5 \le t \le 3 \\ 0 & t < -1 \; and \; t > 3 \end{cases}$$

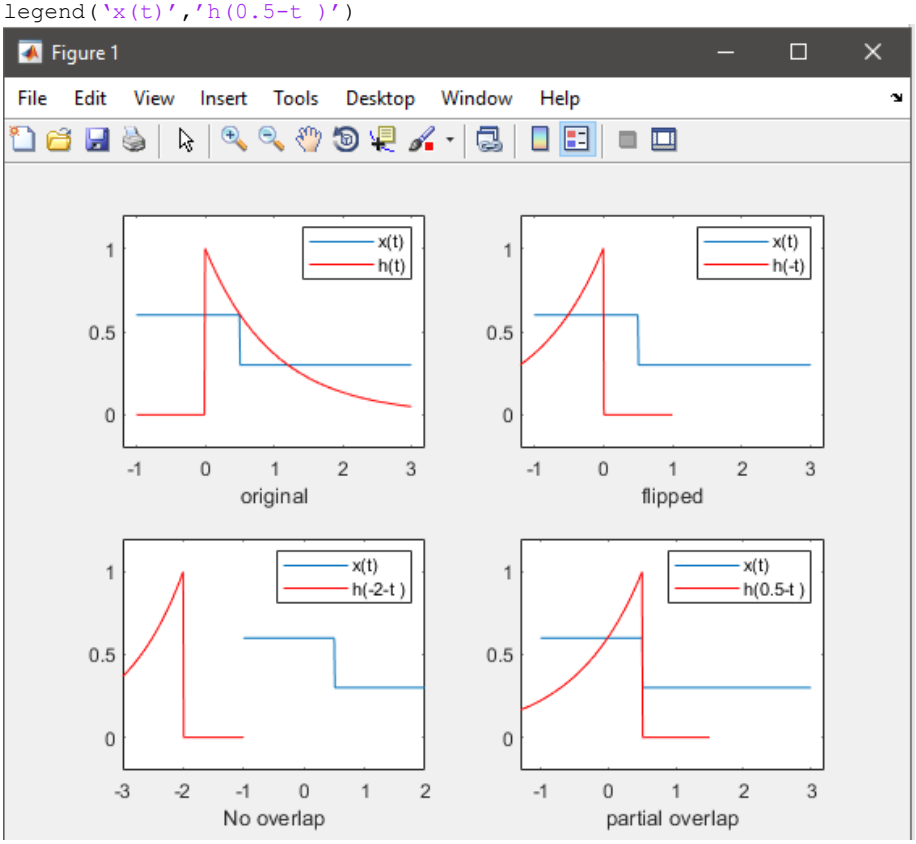
```
n = 0.01;
int1 = -1:n:0.5;
int2 = 0.5+n:n:3;
t = [int1 int2];

x1 = int1;
x1(:) = 1.*0.6;
x2 = int2;
x2(:) = 1.*0.3;
x = [x1 x2];

h = exp(-t).*(t>=0);
subplot(2,1,1);
plot(t,x,t,h,'LineWidth',2)
legend('x(t)','h(t)')

y = conv(x,h).*n;
z = -3:0.01:5;
subplot(2,1,2);
plot(z,y,'LineWidth',2)
legend('y(t)=x(t)*h(t)')
title convolution
```



Other Method:

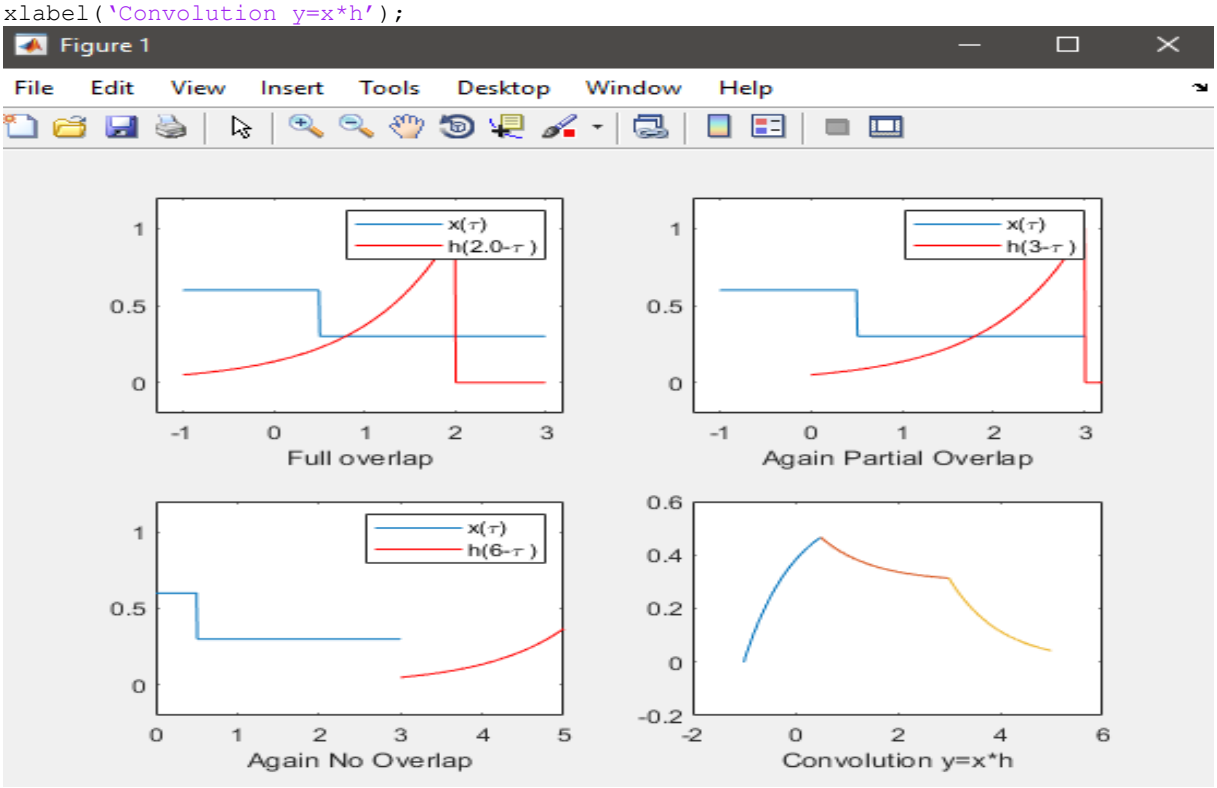
```
n = 0.01;
int1 = -1:n:0.5;
int2 = 0.5+n:n:3;
t = [int1 int2];

x1 = int1;
x1(😊 = 1.*0.6;
x2 = int2;
x2(😊 = 1.*0.3;
x = [x1 x2];

h = exp(-t).*(t>=0);
subplot(2,2,1)            %n1
plot(t,x,t,h,'r')
xlabel('original');
xlim([-1.2 3.2])
ylim([ -0.2 1.2])
legend('x(t)','h(t)')

subplot(2,2,2)            %n2
```

```
plot(t,x,-t,h,'r')        %flipped
xlabel('flipped');
xlim([-1.2 3.2])
ylim([ -0.2 1.2])
legend('x(t)','h(-t)')

subplot(2,2,3)            %n3
p = -2;                  %No Overlap
plot(t,x,-t+p,h,'r')
xlabel('No overlap');
xlim([-3 2])
ylim([ -0.2 1.2])
legend('x(t)','h(-2-t )')

subplot(2,2,4)            %n4
p = 0.5;                  %Partial Overlap
plot(t,x,-t+p,h,'r')
xlabel('partial overlap');
xlim([-1.3 3.2])
ylim([ -0.2 1.2])
legend('x(t)','h(0.5-t )')
```




```
subplot(2,2,1)            %Step5
p = 2;                   % Full Overlap
plot(t,x,-t+p,h,'r')
xlabel('Full overlap');
xlim([-1.3 3.2])
ylim([ -0.2 1.2])
legend('x(\tau)','h(2.0-\tau )')

subplot(2,2,2)            %Step6
p = 3;                    %Exiting Overlap
plot(t,x,-t+p,h,'r')
xlabel('Again Partial Overlap');
xlim([-1.3 3.2])
ylim([ -0.2 1.2])
legend('x(\tau)','h(3-\tau )')

subplot(2,2,3)            %Step7
p =6;                    %No Overlap
```

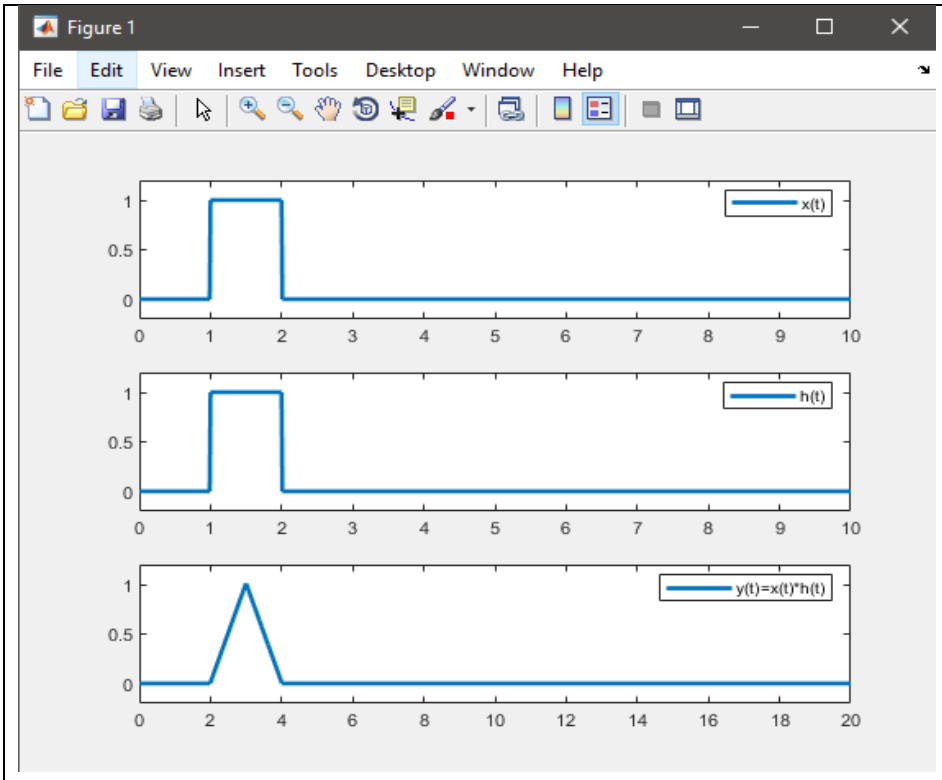
```
plot(t,x,-t+p,h,'r')
xlabel('Again No Overlap');
xlim([0 5])
ylim([ -0.2 1.2])
legend('x(\tau)','h(6-\tau )')

tt1 = -1:0.01:0.5;
tt2 = 0.5:0.01:3;
tt3 = 3:0.01:5;

yt1 = 3/5 - (3*exp(- tt1 - 1))/5;
yt2 = (3*exp(- tt2 - 1)*(exp(3/2) - 1))/5 - (3*exp(-tt2)*exp(1/2))/10 + 3/10;
yt3 = (3*exp(- tt3 - 1)*(exp(3/2) - 1))/5 + (3*exp(-tt3)*exp(1/2)*(exp(5/2) - 1))/10;
subplot(2,2,4);
plot(tt1,yt1,tt2,yt2,tt3,yt3);
ylim([-0.2 0.6]);
xlabel('Convolution y=x*h');
```



Task 02: Compute (by both methods) and plot the response of the system

$$x(t) = h(t) = \begin{cases} 0 & 0 < t < 1 \\ 1 & 1 \le t \le 2 \\ 0 & 2 \le t \le 10 \end{cases}$$

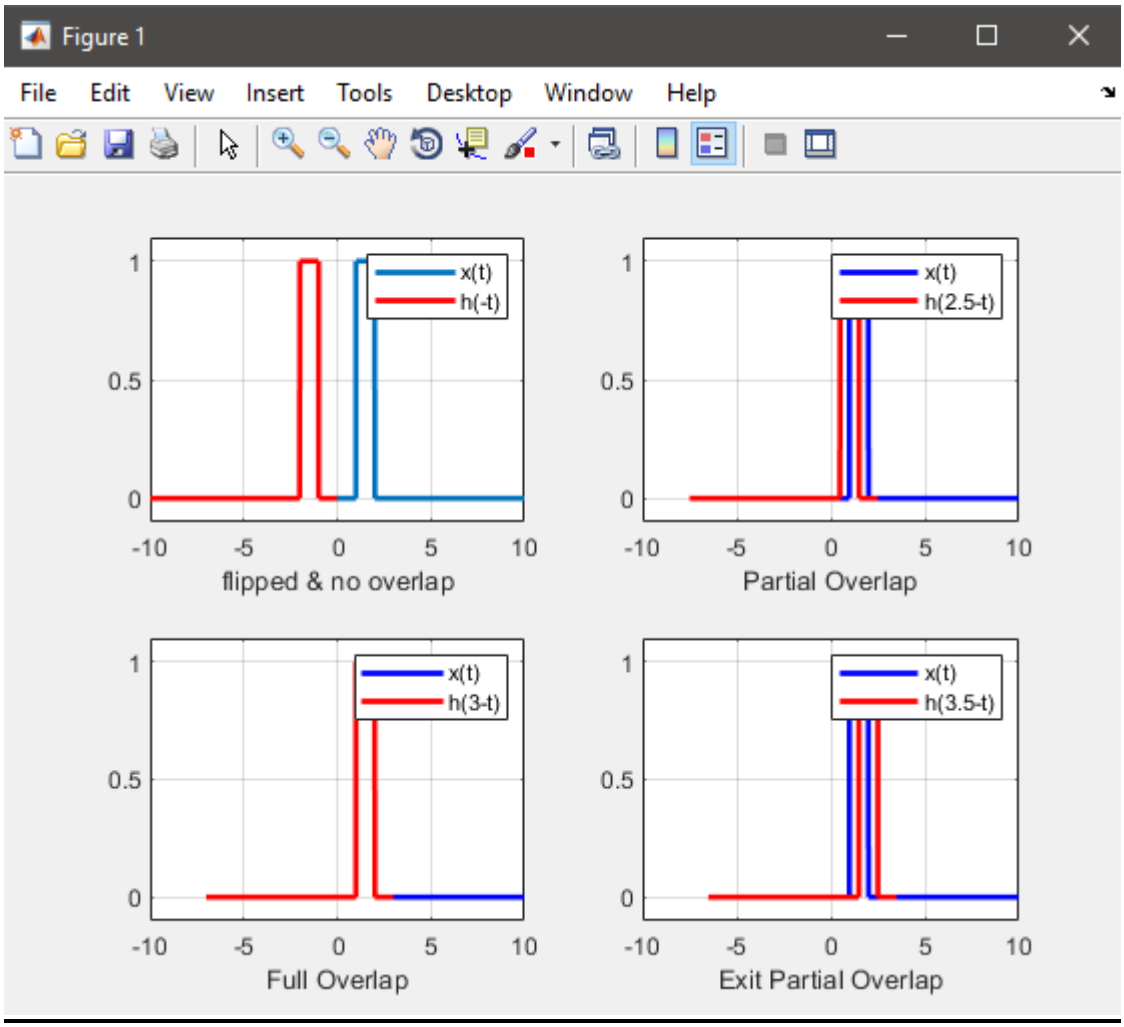
```
step = 0.01;
int1 = 0:step:1-step;
int2 = 1:step:2;
int3 = 2+step:step:10;
t_int = [int1 int2 int3];

x1 = int1;
x1(😊 = 0;
x2 = int2;
x2(😊 = 1;
x3 = int3;
x3(😊 = 0;
x = [x1 x2 x3];

h = x;
subplot(3,1,1)
plot(t_int,x,'LineWidth',2)
legend('x(t)')
ylim([-0.2 1.2])

subplot(3,1,2)
plot(t_int,h,'LineWidth',2);
legend('h(t)')
ylim([-0.2 1.2])

y = conv(x,h).*step;
z = 0:step:20;
subplot(3,1,3)
plot(z,y,'LineWidth',2);
legend('y(t)=x(t)*h(t)')
ylim([-0.2 1.2])
```



Other Method:

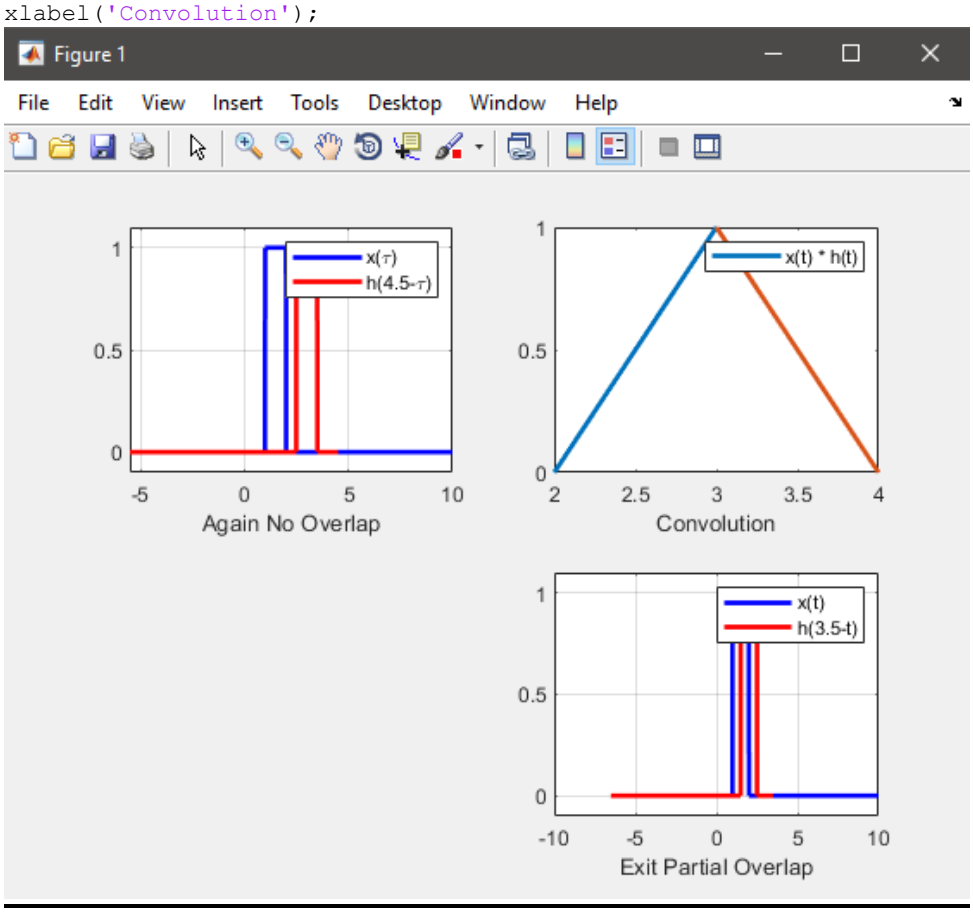
```
step = 0.01;
int1 = 0:step:1-step;
int2 = 1:step:2;
int3 = 2+step:step:10;
t_int = [int1 int2 int3];

x1 = int1;
x1(😊 = 0;
x2 = int2;
x2(😊 = 1;
x3 = int3;
x3(😊 = 0;
x = [x1 x2 x3];

h = x;

subplot(2,2,1)
plot(t_int,x,-t_int,h,'r','LineWidth',2)     %flipped
legend('x(t)','h(-t)')          %No Overlap
xlabel('flipped & no overlap');
ylim([-0.1 1.1])
grid on

shift = 2.5;
subplot(2,2,2)
plot(t_int,x,'b',-t_int+shift,h,'r','LineWidth',2)  %Partial Overlap
legend('x(t)','h(2.5-t)')
xlabel('Partial Overlap');
ylim([-0.1 1.1])
grid on

shift = 3;
subplot(2,2,3)
plot(t_int,x,'b',-t_int+shift,h,'r','LineWidth',2)  %Full Overlap
legend('x(t)','h(3-t)')
```

```
xlabel('Full Overlap');
ylim([-0.1 1.1])
grid on

shift = 3.5;
subplot(2,2,4)
plot(t_int,x,'b',-t_int+shift,h,'r','LineWidth',2)  %Partial Overlap
legend('x(t)','h(3.5-t)')
xlabel('Exit Partial Overlap');
ylim([-0.1 1.1])
grid on
```



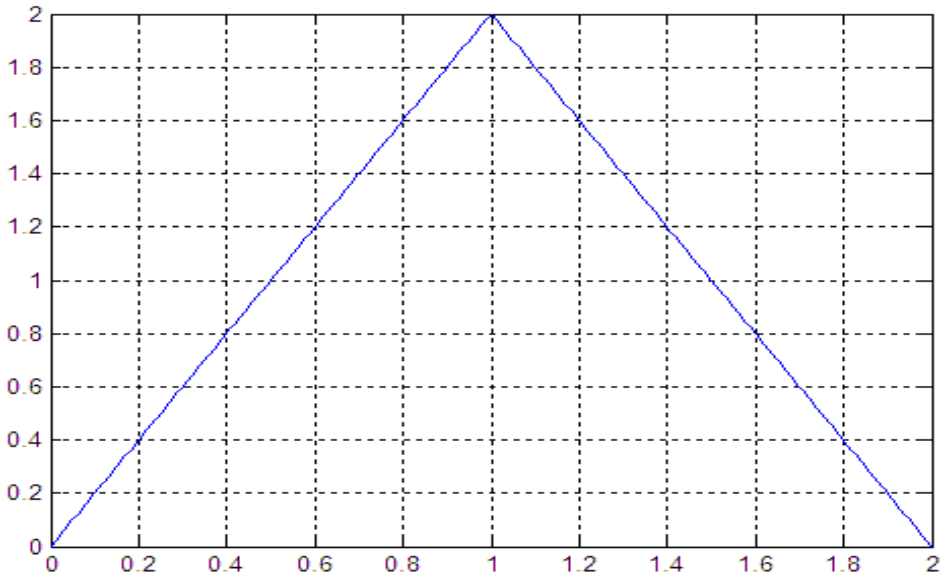

```
shift = 3.5;
subplot(2,2,4)
plot(t_int,x,'b',-t_int+shift,h,'r','LineWidth',2)  %Partial Overlap
legend('x(t)','h(3.5-t)')
xlabel('Exit Partial Overlap');
ylim([-0.1 1.1])
grid on


p = 4.5;
subplot(2,2,1)
plot(t_int,x,'b',-t_int+p,h,'r','LineWidth',2)  %No Overlap
legend('x(\tau)','h(4.5-\tau)')
xlabel('Again No Overlap');
ylim([-0.1 1.1])
grid on
```

```
t1 = 2:0.01:3;
t2 = 3:0.01:4;

y1 = t1-2;
y2 = -t2+4;
subplot(2,2,2);
plot(t1,y1,t2,y2,'linewidth',2);              %plotting the calculated equations
legend('x(t) * h(t)');
xlabel('Convolution');
```

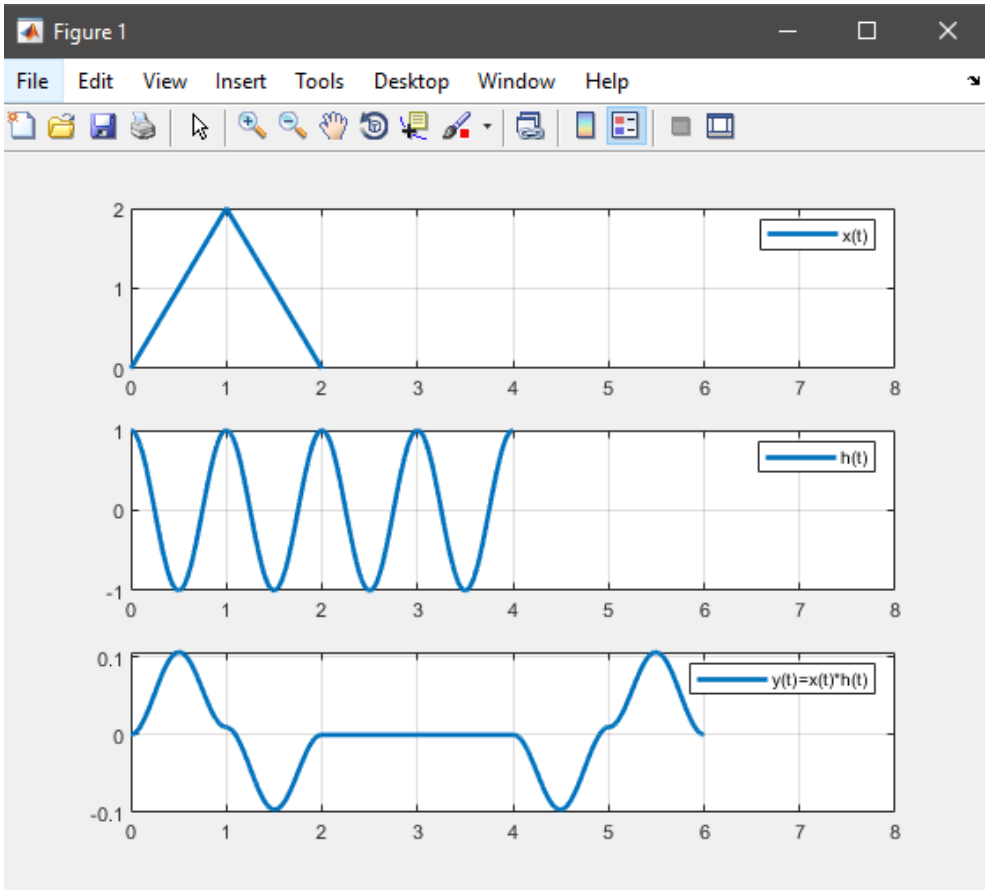


Task 03: Suppose that a system is

described by the impulse response $h(t) = \cos(2\pi t)(u(t) - u(t-4))$. Compute (by both methods) and plot the response of the system to the input shown in figure below



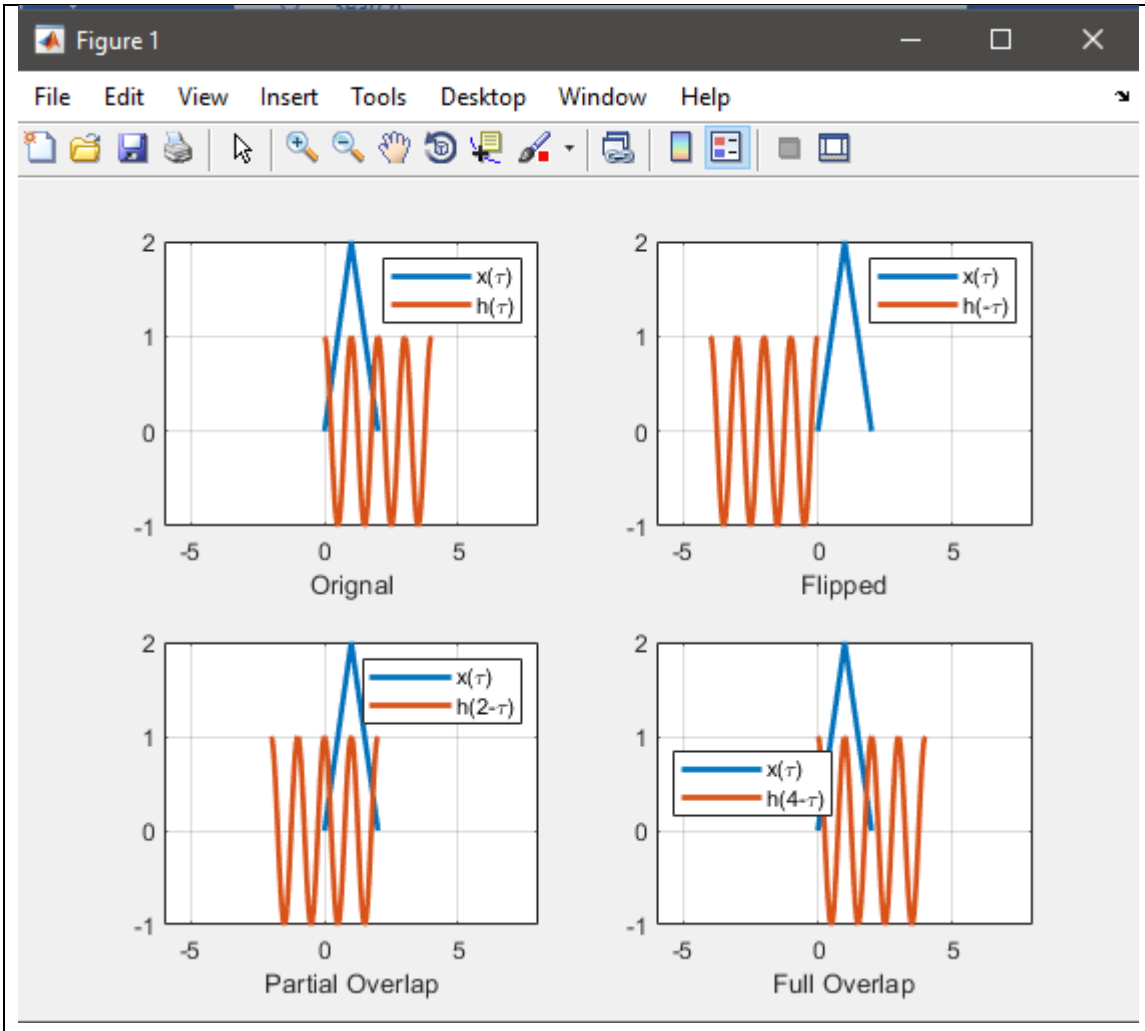

```
step = 0.01;
int1 = 0:step:1-step;
int2 = 1:step:2;
t_int = [int1 int2];

x1 = 2.*int1 ;
x2 = 4 - 2.*int2 ;
x = [x1 x2] ;

subplot(3,1,1)
plot(t_int,x,'LineWidth',2)
legend('x(t)')
xlim([0 8])
grid on

subplot(3,1,2)
h_int = 0:step:4;
h1 = h_int;
h1(:) = 1;
h2 = cos(2.*pi.*h_int);
h = h2.*h1;
plot(h_int,h,'LineWidth',2)
legend('h(t)')
xlim([0 8])
grid on

subplot(3,1,3)
y = conv(x,h).*step;
z = 0:step:6;
subplot(3,1,3)
plot(z,y,'LineWidth',2)
legend('y(t)=x(t)*h(t)')
xlim([0 8])
grid on
```



Other Method:

```
step = 0.01;
int1 = 0:step:1-step;
int2 = 1:step:2;
t = [int1 int2];

x1 = 2.*int1 ;
x2 = 4 - 2.*int2 ;
x = [x1 x2] ;

h_int = 0:step:4;
h1 = h_int;
```

```
h1(:) = 1;
h2 = cos(2.*pi.*h_int);
h = h2.*h1;

subplot(2,2,1)
plot(t,x,h_int,h,'LineWidth',2)  %orignal
legend('x(\tau)','h(\tau)')
xlabel('Orignal');
xlim([-6 8])
grid on

subplot(2,2,2)
plot(t,x,-h_int,h,'LineWidth',2)  %flipped
legend('x(\tau)','h(-\tau)')
xlabel('Flipped');
xlim([-6 8])
grid on

p = 2;
subplot(2,2,3)
plot(t,x,-h_int+p,h,'LineWidth',2)  %partial overlap
legend('x(\tau)','h(2-\tau)')
xlabel('Partial Overlap');
xlim([-6 8])
grid on

p = 4;
subplot(2,2,4)
plot(t,x,-h_int+p,h,'LineWidth',2)  %full overlap
legend('x(\tau)','h(4-\tau)','Location','west')
xlabel('Full Overlap');
xlim([-6 8])
grid on
```




```
p = 6;
subplot(2,2,1)
plot(t,x,-h_int+p,h,'LineWidth',2)  %exit partial overlap
legend('x(\tau)','h(6-\tau)','Location','west')
xlabel('Exit Partial Overlap');
xlim([-6 8])
grid on

p = 8;
subplot(2,2,2)
plot(t,x,-h_int+p,h,'LineWidth',2)  %no overlap
legend('x(\tau)','h(8-\tau)','Location','west')
xlabel('No Overlap');
xlim([-6 8])
grid on

subplot(2,2,3);
y = conv(x,h).*step;
z = 0:step:6;
plot(z,y,'linewidth',2);
legend('y(t)=x(t)*h(t)')
xlabel('convolution y=x*h');
xlim([0 8])
grid on
```

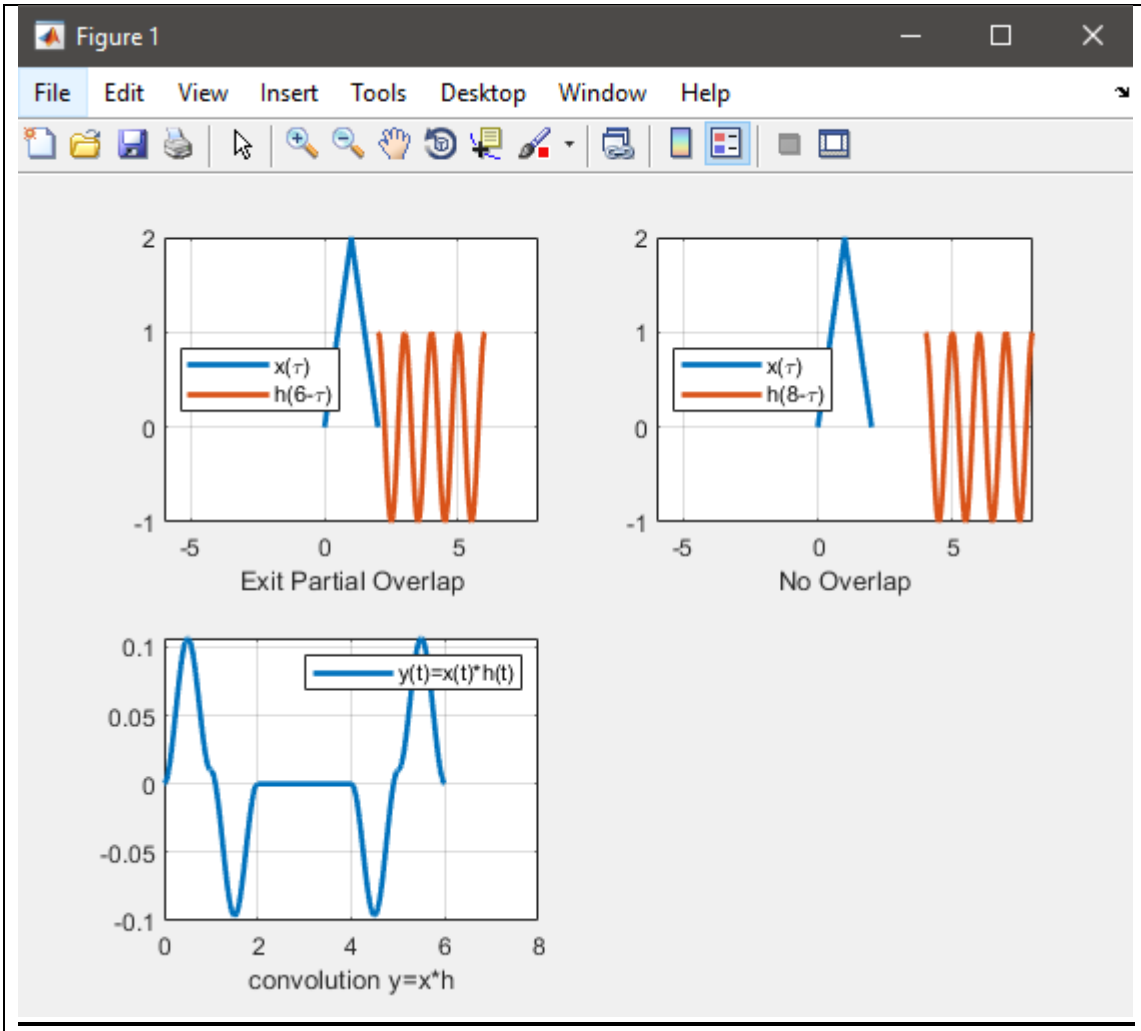


Figure 1
File Edit View Insert Tools Desktop Window Help
x(τ)
h(6-τ)
Exit Partial Overlap
x(τ)
h(8-τ)
No Overlap
y(t)=x(t)*h(t)
convolution y=x*h

Post-Lab Task

Critical Analysis / Conclusion

In this lab we learnt how to perform convolution of continuous time signals in matlab. We learned two different methods of finding convolution as we did for the discrete time signals. In first method we used built-in conv() function of matlab to convolve an input signal and impulse response of the system. While dealing with continuous time signals, we should convolve the output of conv() function with step in order to obtain correct results. In the second method we used conventional method for finding convolution which includes the flipping of any of the two signals and then shifting and watching for the overlap of two signals.

Lab Assessment		
Pre-Lab	**/1**	**/10**
In-Lab	**/5**	
Critical Analysis	**/4**	
Instructor Signature and Comments		

LAB # 08

Properties of Convolution

Lab 08- Properties of Convolution

Pre-Lab Tasks

8.1 Properties of Convolution:

In this section, we introduce the main properties of convolution through illustrative examples.

- Commutative Property

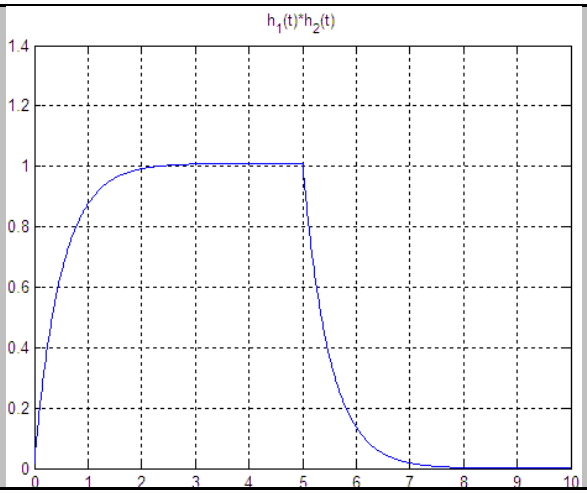
For two signals $h_1(t)$ and $h_2(t)$ the commutative property stands; that is equation 8.1, given as

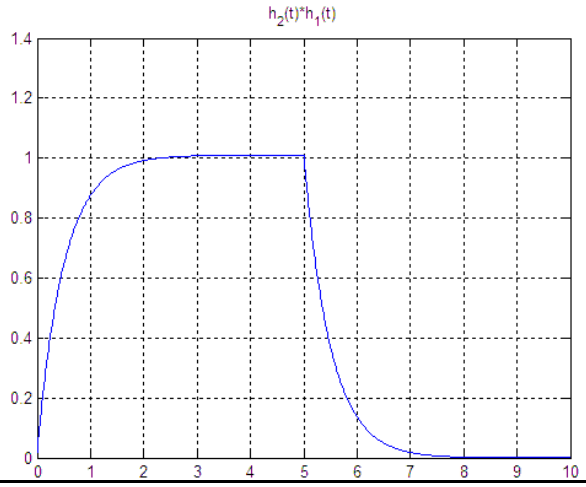
$$h_1(t) * h_2(t) = h_2(t) * h_1(t)$$

Example:

Verify the commutative property of the convolution supposing that $h_1(t) = 1, 0 \le t \le 5$ and $h_2(t) = 2e^{-2t}, 0 \le t \le 5$.

The left side of 8.1, i.e., the signal $y(t) = h_1(t) * h_2(t)$ is computed and plotted first while the right side of 8.1, namely, $z(t) = h_2(t) * h_1(t)$ is computed and plotted.

Commands	Results	Comments
t=0:0.01:5; **h1=ones(size(t));** **h2=2*exp(-2*t);** **y=conv(h1,h2)*0.01;** **plot(0:0.01:10,y),grid on** **title('h_1(t)*h_2(t)')**		The left side of 8.1.

z=conv(h2,h1)*0.01; **plot(0:0.01:10,y),grid on** **title('h_2(t)*h_1(t)')**		The right side of 8.1.

- Associative Property

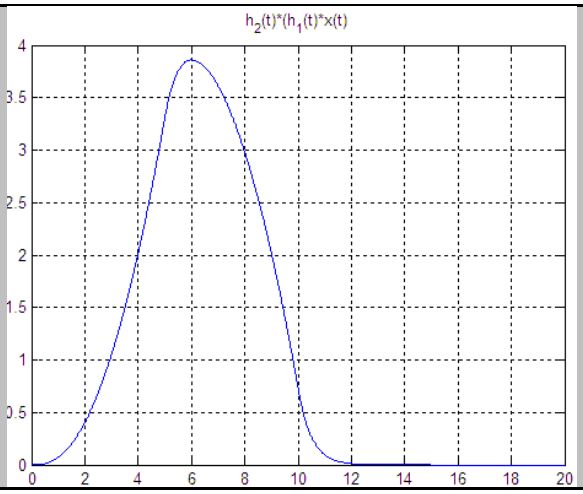
For three signals $h_1(t)$, $h_2(t)$ and $x(t)$ the associative property stands; that is 8.2, given as,

$$h_2(t)*\left[h_1(t)*x(t)\right]=\left[h_2(t)*h_1(t)\right]*x(t)$$

Example:

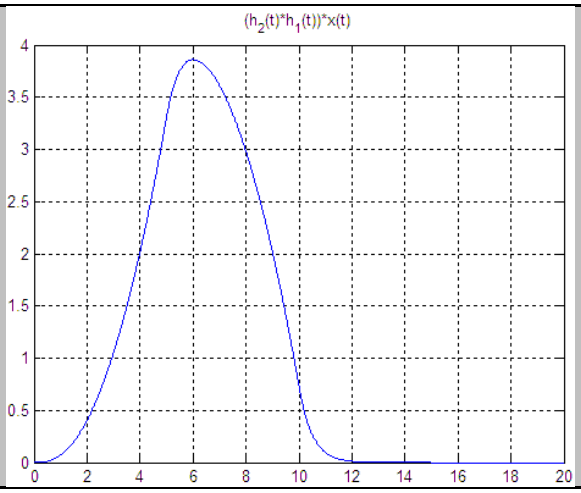
Verify the associative property of the convolution supposing that $h_1(t)=(1/\pi)t, 0\le t\le 5$ and $h_2(t)=2e^{-2t}, 0\le t\le 5$; and $x(t)=u(t)-u(t-5)$.

For the left side of 8.2, which is $y(t)=h_2(t)*\left[h_1(t)*x(t)\right]$, first the convolution $y_1(t)=h_1(t)*x(t)$ is computed. Next, the signal $h_2(t)$ is defined in the same time interval with $y_1(t)$ (at $0\le t\le 10$) and the result of their convolution is plotted.

Commands	Results	Comments
t=0:0.01:5; **x=ones(size(t));** **h1=1/pi*t;** **y1=conv(h1,x)*0.01;** **h2=2*exp(-2*t);** **th2=5.01:0.01:10;** **hh2=zeros(size(th2));** **h2=[h2 hh2];** **y=conv(h2,y1)*0.01;** **plot(0:0.01:20,y), grid on** **title('h_2(t)*(h_1(t)*x(t)')**		The left side of 8.2.

The right side of 8.2, which is $z(t)=\left[h_2(t)*h_1(t)\right]*x(t)=\left[h_1(t)*h_2(t)\right]*x(t)$, first the convolution $z_1(t)=h_1(t)*h_2(t)$ is computed. Next the signal $x(t)$ is defined in the same time interval with $z_1(t)$ (at $0\le t\le 10$) and the result of their convolution $z(t)$ is plotted.

Commands	Results	Comments

t=0:0.01:5;
h1=1/pi*t;
h2=2*exp(-2*t);
z1=conv(h1,h2)*0.01;
x=ones(size(t));
tx=5.01:0.01:10;
xx=zeros(size(tx));
x=[x xx];
z=conv(z1,x)*0.01;
plot(0:0.01:20,z), grid on
title('(h_2(t)*h_1(t))*x(t)')
		The right side of 8.2.

- Distributive Property

For three signals $h_1(t)$, $h_2(t)$ and $x(t)$ the distributive property stands; that is 8.3, given as

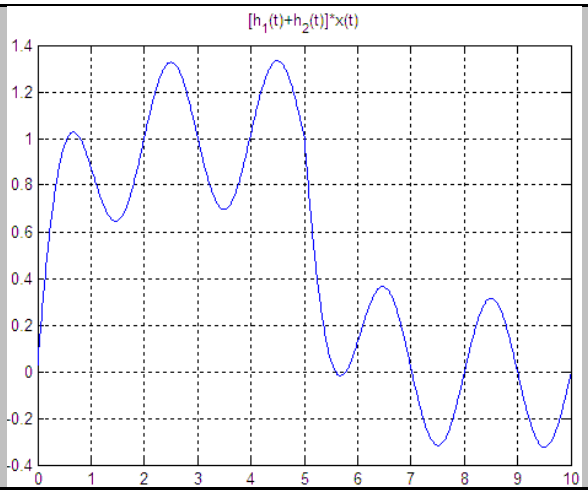
$$\left[h_1(t)+h_2(t)\right]*x(t)=h_1(t)*x(t)+h_2(t)*x(t)$$

Example:

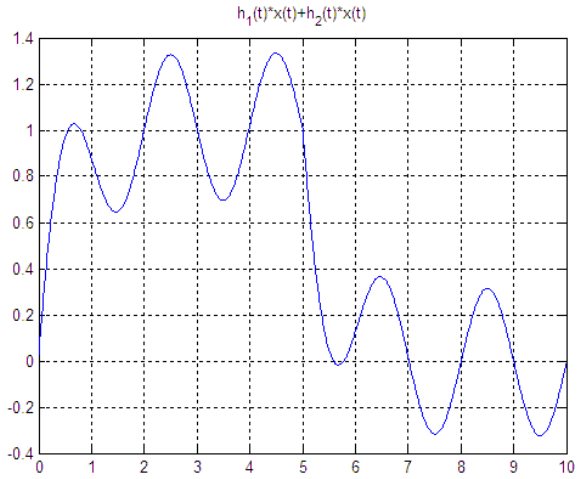
Illustrate the distributive property of convolution by using the signals; $h_2(t)=2e^{-2t}, 0\le t\le 5$; and $x(t)=u(t)-u(t-5)$.

In the similar vein to two previous examples, the left side of 8.3, that is, $y(t)=\left[h_1(t)+h_2(t)\right]*x(t)$ is compared to the right side of 7.3, that is, $z(t)=h_1(t)*x(t)+h_2(t)*x(t)$.

Commands	Results	Comments

t=0:0.01:5;
x=ones(size(t));
h1=cos(pi*t);
h2=2*exp(-2*t);
h=h1+h2;
y=conv(h,x)*0.01;
plot(0:0.01:10,y),grid on
title('[h_1(t)+h_2(t)]*x(t)')
		The left side of 8.3.

```
t=0:0.01:5;
x=ones(size(t));
h1=cos(pi*t);
h2=2*exp(-2*t);
z1=conv(h1,x)*0.01;
z2=conv(h2,x)*0.01;
z=z1+z2;
plot(0:0.01:10,z),grid on
title('h_1(t)*x(t)+h_2(t)*x(t)')
```



The right side of 8.3.

- Identity Property

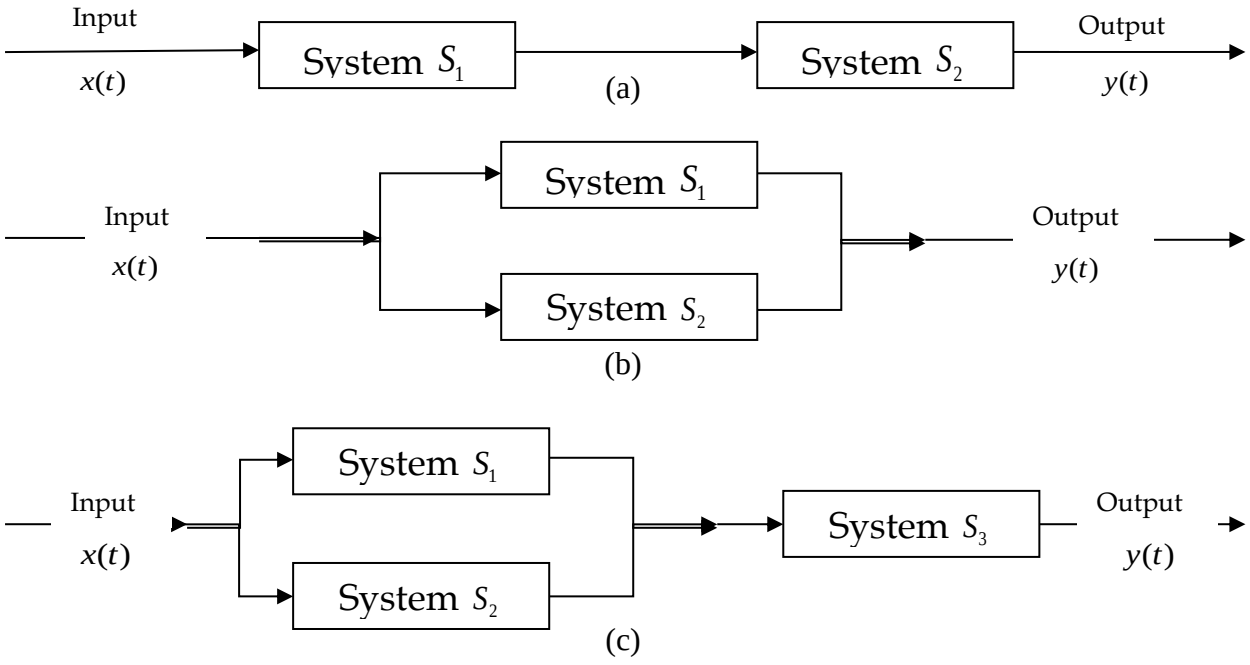
  If $\delta(t)$ is the Dirac delta function, then for any signal $h(t)$ the following expression is true:

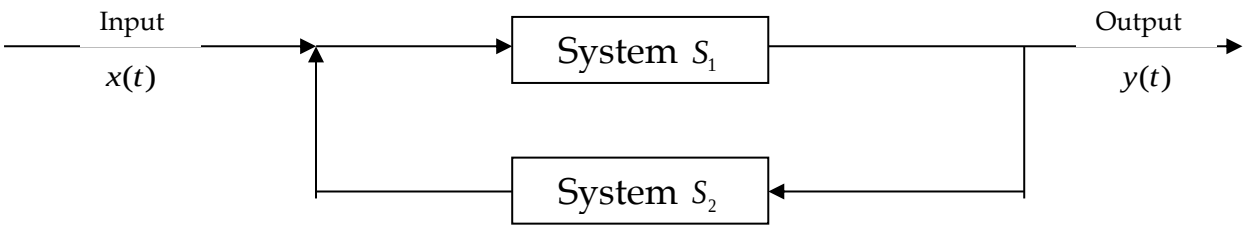
  $$h(t) * \delta(t) = h(t)$$

This property is straight forwardly proven from the definition of Dirac function. Nevertheless, an example will be provided in lab of discrete time convolution.

8.2 Interconnections of Systems:

Systems may be interconnections of other sub-systems. The basic interconnections are the cascade, the parallel, the mixed and the feedback. The block diagrams are illustrated in Figure 8.1.





(d)

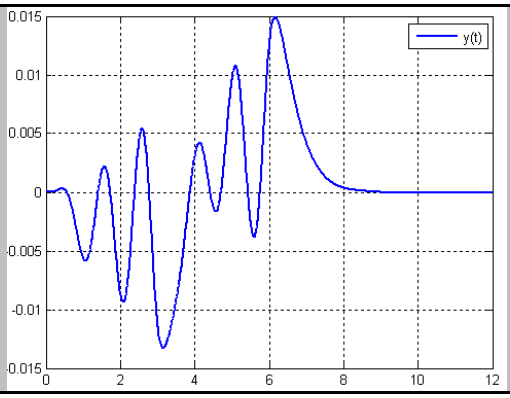
Figure 8.1: Interconnections of (sub) systems: (a) Cascade, (b) Parallel, (c) Mixed, and (d) Feedback

When two systems S_1 and S_2 are cascade (or serially) connected (Figure 8.1a), the output of the first system is the input of the second system. The block diagram off two parallel interconnected systems is presented in Figure 8.1b. The same input signals are applied to the two parallel-connected systems and the output of S_1 and S_2 are combined to generate the overall output. The mixed interconnection is a combination of cascade and parallel interconnections. In the block diagram of Figure 8.1c, systems S_1 and S_2 are parallel connected, and their output is input to the cascade-connected system S_3. Finally in Figure 8.1d the feedback interconnection block diagram is depicted. The output of S_1 is input to S_2, while the output of S_2 is fed back to S_1 and combined with the input signal produce the overall output of the system.

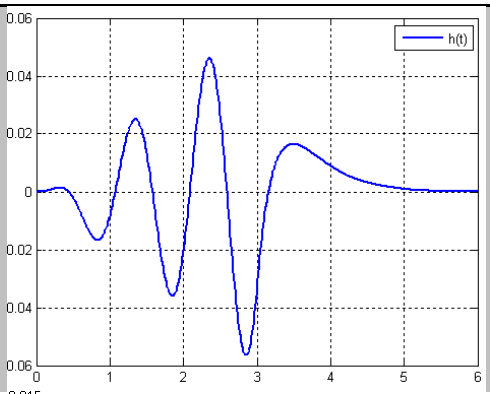
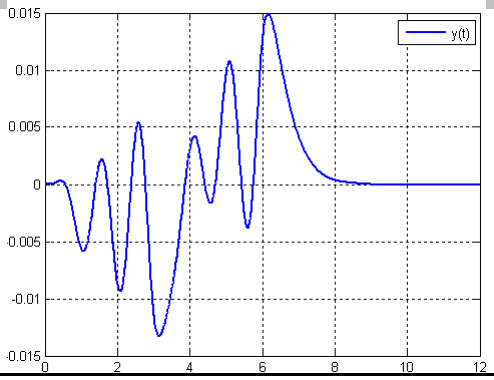
The interconnected subsystems can be considered as one system, i.e., an equivalent system described by one overall impulse response. In order to compute the output and the impulse response of the equivalent overall system for the various types of interconnections, suppose that the subsystems S_1 is described by the impulse response $h_1(t) = te^{-3t}\left[u(t)-u(t-3)\right]$ and the subsystem S_2 by the impulse response $h_2(t) = t\cos(2\pi t)\left[u(t)-u(t-3)\right]$. Finally, let $x(t) = u(t)-u(t-3)$ be the input signal.

- Cascade interconnection

The output of S_1 is input to S_2. Thus, the output of the equivalent systems is computed as $y(t) = \left[x(t) * h_1(t)\right] * h_2(t)$; that is, the input signal is first convoluted with the impulse response of S_1 and the computed output is convoluted with the impulse response of S_2.

Commands	Results	Comments
t=0:0.01:3; **x=ones(size(t));** **h1=t.*exp(-3*t);** **y1=conv(x,h1)*0.01;** **t1=0:0.01:3;** **h2a=t1.*cos(2*pi*t1);** **t2=3.01:0.01:6;** **h2b=zeros(size(t2));** **h2=[h2a h2b];** **y=conv(y1,h2)*0.01;** **plot(0:0.01:12,y,'linewidth',2),grid on** **legend('y(t)')**		Graph of the output $y(t) = \left[x(t) * h_1(t)\right] * h_2(t)$

To compute the impulse response of the overall system, the associative property of the convolution is applied. More specifically, applying the associative property to the output relationship $y(t)=\left[x(t)*h_1(t)\right]*h_2(t)$ yields $y(t)=x(t)*\left[h_2(t)*h_1(t)\right]$. Consequently, the impulse response of the overall equivalent system, if the subsystem, are cascade connected, is given by $h(t)=h_1(t)*h_2(t)$. Straightforwardly, the systems response of $x(t)$ is given by $y(t)=x(t)*h(t)$.

Commands	Results	Comments
t=0:0.01:3; **h1=t.*exp(-3*t);** **h2=t.*cos(2*pi*t);** **h=conv(h1,h2)*0.01;** **plot(0:0.01:6,h,'linewidth',2), grid on;** **legend('h(t)')**		The impulse response $h(t)$ is obtained by $h(t)=h_1(t)*h_2(t)$
t1=0:0.01:3; **t2=3.01:0.01:6;** **x1=ones(size(t1));** **x2=zeros(size(t2));** **x=[x1 x2];** **y=conv(x,h)*0.01;** **plot(0:0.01:12,y,'linewidth',2),grid on;** **legend('y(t)')**		The output of the system $y(t)$ is computed from the convolution between the input signal $x(t)$ and the overall impulse response $h(t)$, which was derived using the associative property.

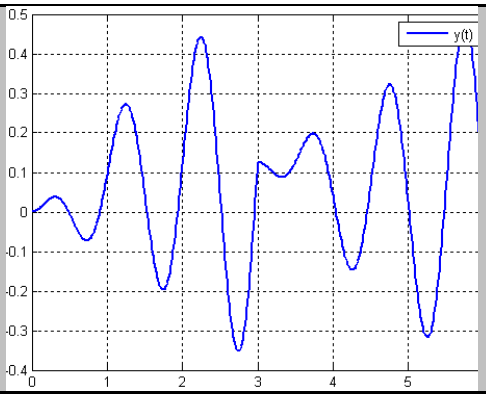
The two graphs are identical; hence, the computation of the impulse response and the output signal of the equivalent system are correct.

- Parallel Interconnection

In this type of interconnection, the same input is applied to both subsystems. The two outputs of subsystems are added to obtain the final output. The mathematical expression is $y(t)=h_1(t)*x(t)+h_2(t)*x(t)$.

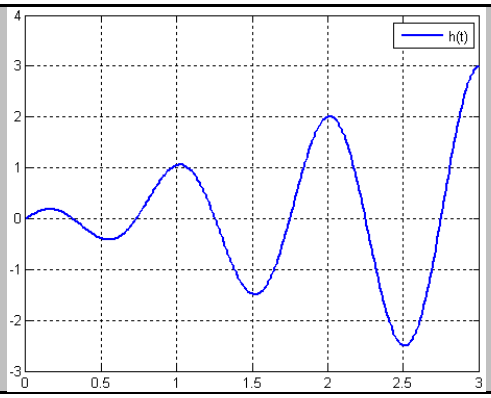
Commands	Results	Comments

```
t=0:0.01:3;
h1=t.*exp(-3*t);
h2=t.*cos(2*pi*t);
x=ones(size(t));
y1=conv(h1,x)*0.01;
y2=conv(h2,x)*0.01;
y=y1+y2;
plot(0:0.01:6,y,'linewidth',2), grid on;
legend('y(t)')
```

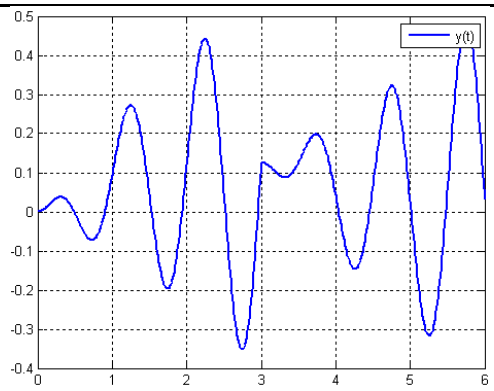


The response of the systems to the input signal $x(t)$ is computed by adding the outcome of the convolutions between the input signal and the impulse response of the subsystems; that is, it is computed as $y(t) = h_1(t) * x(t) + h_2(t) * x(t)$

To compute the impulse response of the overall system the distributive property of the convolution is applied. More specifically, applying the distributive property to the output relationship $y(t) = h_1(t) * x(t) + h_2(t) * x(t)$ yields $y(t) = \left[h_1(t) + h_2(t)\right] * x(t)$. Consequently, the impulse response of the equivalent system when the subsystems are parallel connected is given by $h(t) = h_1(t) + h_2(t)$. Straightforwardly, the output of the system is given by $y(t) = x(t) * h(t)$. To verify the conclusion, we consider the same signals used in the cascade interconnection case, namely, $h_1(t) = te^{-3t}\left[u(t) - u(t-3)\right]$, $h_2(t) = t\cos(2\pi t)\left[u(t) - u(t-3)\right]$ and $x(t) = u(t) - u(t-3)$.

Commands	Results	Comments
t=0:0.01:3; **h1=t.*exp(-3*t);** **h2=t.*cos(2*pi*t);** **h=h1+h2;** **plot(0:0.01:3,h,'linewidth',2), grid on;** **legend('h(t)')**		The overall impulse response of the system is the sum of the impulse responses of the subsystems; that it is computed by $h(t) = h_1(t) + h_2(t)$

```
x=ones(size(t));
y=conv(x,h)*0.01;
plot(0:0.01:6,y,'linewidth',2),grid on;
legend('y(t)')
```



The output of the system $y(t)$ is computed from the convolution between the input signal $x(t)$ and the impulse response of $h(t)$, which was derived using the distributive property.

The graphs of the system response are identical; hence, our computation of the impulse response and the output of the equivalent system is accurate.

Note: The implementation of the mixed interconnection is left as an exercise to the students.

8.3 Stability Criterion for Continuous Time Systems:

The concept of stability was introduced in lab session 5. A system is bounded-input bounded-output (BIBO) stable if for any bounded applied input; the response of the system is also bounded. The knowledge of the impulse response of the systems allows us to specify a new criterion about the stability of a system.

An LTI system is BIBO stable if its impulse response is absolutely integrable on $(-\infty, +\infty)$

The mathematical expression (equation 8.4) is

$$\int_{-\infty}^{+\infty} |h(t)|\, dt < \infty$$

Example:

A system is described by the impulse response $h(t) = e^{-t^2}$. Tell is this system is BIBO stable and verify your conclusion.

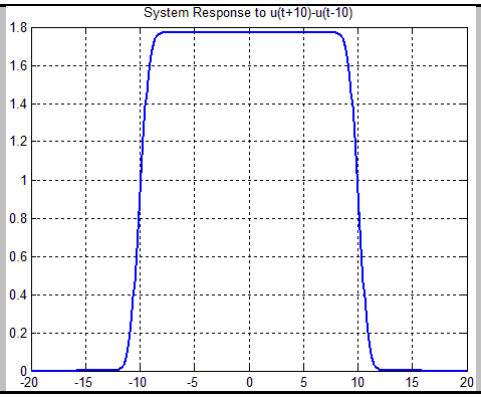
Commands	Results	Comments
syms t **h=exp(-t.^2);** **int(abs(h),t,-inf,inf)**	ans=pi^(1/2)	Condition 8.4 is fulfilled; hence, the system under consideration is BIBO stable.

A system is BIBO stable if the condition given in 8..1 is satisfied; that is, we have to examine if its impulse response is absolutely integrable.

In order to verify that this is a BIBO stable system, the bounded input signal $x(t) = u(t+10) - u(t-10)$ is applied to the system. The response of the system is expected to be also bounded.

Commands	Results	Comments

```
t=-10:0.1:10;
x1=ones(size(t));
h=exp(-t.^2);
y1=conv(x1,h)*0.1;
plot(-20:0.1:20,y1,'linewidth',2), grid on;
title('System Response to u(t+10)-u(t-10)'
```



The system response $y(t)$ is computed from the convolution between $x(t)$ and $h(t)$.

Indeed, the response of the systems is bounded $\left(|y(t)| < M = 2\right)$; thus the BIBO stability of the system is verified.

8.4 Stability Criterion for Discrete Time Systems:

In the previous section, we have established a criterion about the stability of continuous time LTI systems. More specifically, it was stated that a system is stable if the impulse response of the system is absolutely integrable. For discrete time systems a similar criterion can be established. More specifically, a discrete time linear shift invariant system is stable if and only if its impulse response $h[n]$ is absolutely summable. The mathematical expression (equation 8.2) is

$$\sum_{-\infty}^{+\infty} |h[n]| < \infty$$

Example:

A system is described by impulse response $h[n] = \left(1/2^n\right)u[n]$. Tell if this is a BIBO stable system.

A discrete time system is BIBO stable if the condition given in equation 8.2 is satisfied; that is; we examine if the impulse response of the system is absolutely summable.

Commands	Results	Comments
syms n **h=1/(2^n)** **symsum(abs(h),n,0,inf)**	**ans=2**	Condition 8.4 is not fulfilled; hence, the system is not BIBO stable.

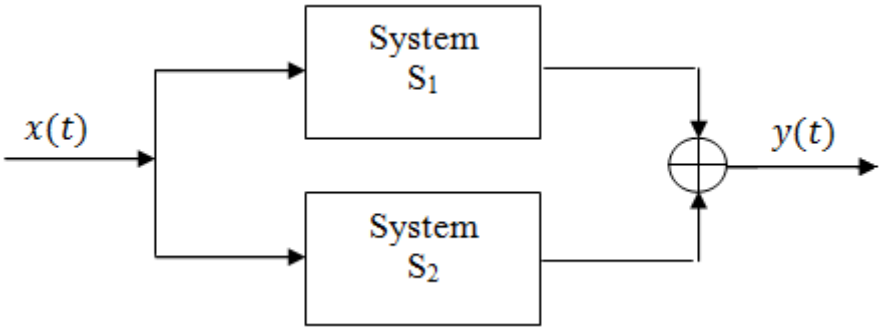
In-Lab Tasks

Task 01: A system is described by the impulse response $h(t) = t^2$. Tell if this is a BIBO stable system.

```
syms t
h=t^2;
int(abs(h),t,0,inf)

ans= inf
```

As the answer in infinite, hence the system h(t) is not BIBO stable

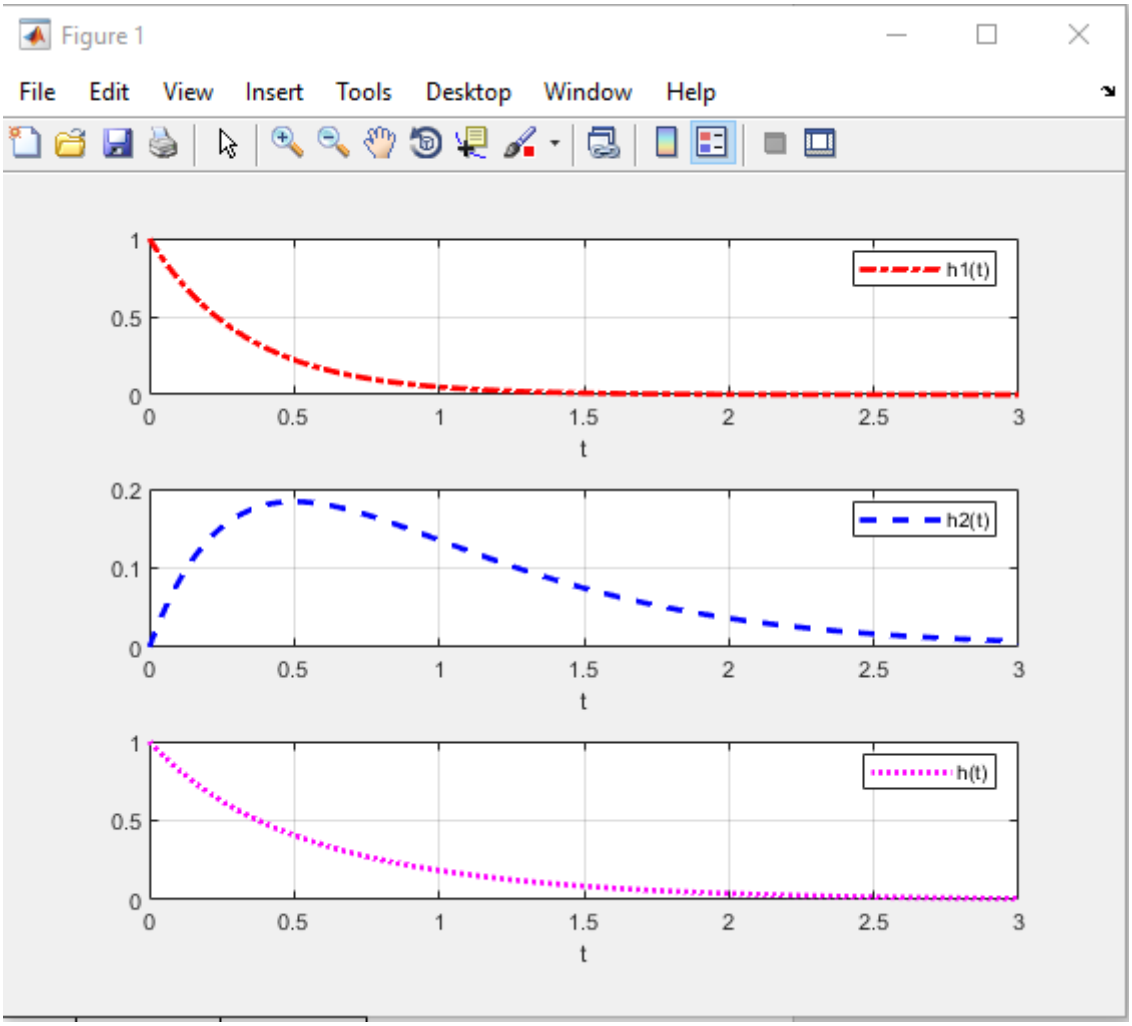
Task 02: Suppose that the impulse response of the subsystems S_1 and S_2 that are connected as shown in figure below are $h_1(t) = e^{-3t}u(t)$ and $h_2(t) = te^{-2t}u(t)$. Determine if the overall system is BIBO stable.



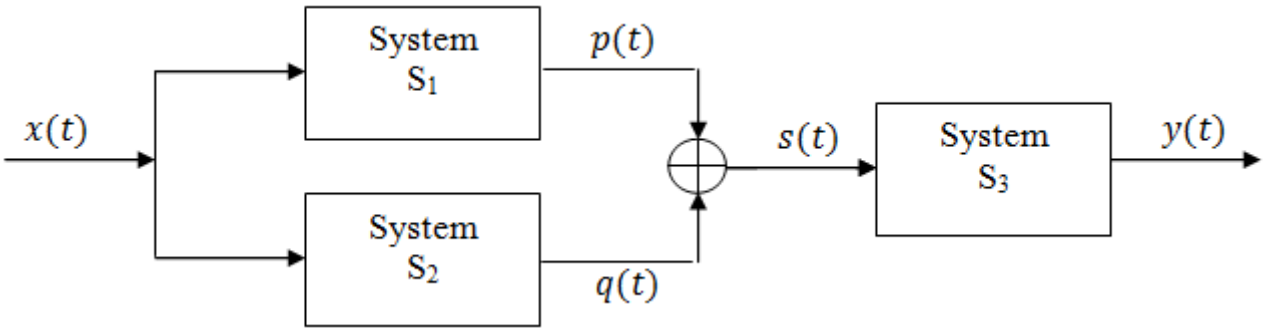

```
step=0.01;

t=0:step:3;


u=ones(size(t));

subplot(3,1,1);

h1=u.* exp(-3.*t);

plot(t,h1,'r-.','linewidth',2),grid on;

legend('h1(t)');

xlabel('t')


subplot(3,1,2)

h2=u.*(t.*exp(-2.*t));
```

```
plot(t,h2,'b--','linewidth',2),grid on;
legend('h2(t)');
xlabel('t')

h=h1+h2;
subplot(3,1,3)
plot(t,h,'m:','linewidth',2),grid on
legend('h(t)');
xlabel('t')

syms t
f=exp(-3.*t)+(t.*exp(-2.*t));
int(abs(f),t,0,inf)

ans =

7/12
```

System is BIBO Stable.



Task 03: Suppose that the impulse responses of the sub-systems S_1, S_2 and S_3 that are connected as shown in the figure below are $h_1(t) = t\cos(2\pi t), 0 \le t \le 4$; $h_2(t) = t\,e^{-2t}, 0 \le t \le 4$; and $h_3(t) = u(t) - u(t-5)$. Compute and plot in the appropriate time interval the impulse response of the overall system and the response of the overall system to the input signal $x(t) = te^{-2t}\,[u(t) - u(t-2)]$.



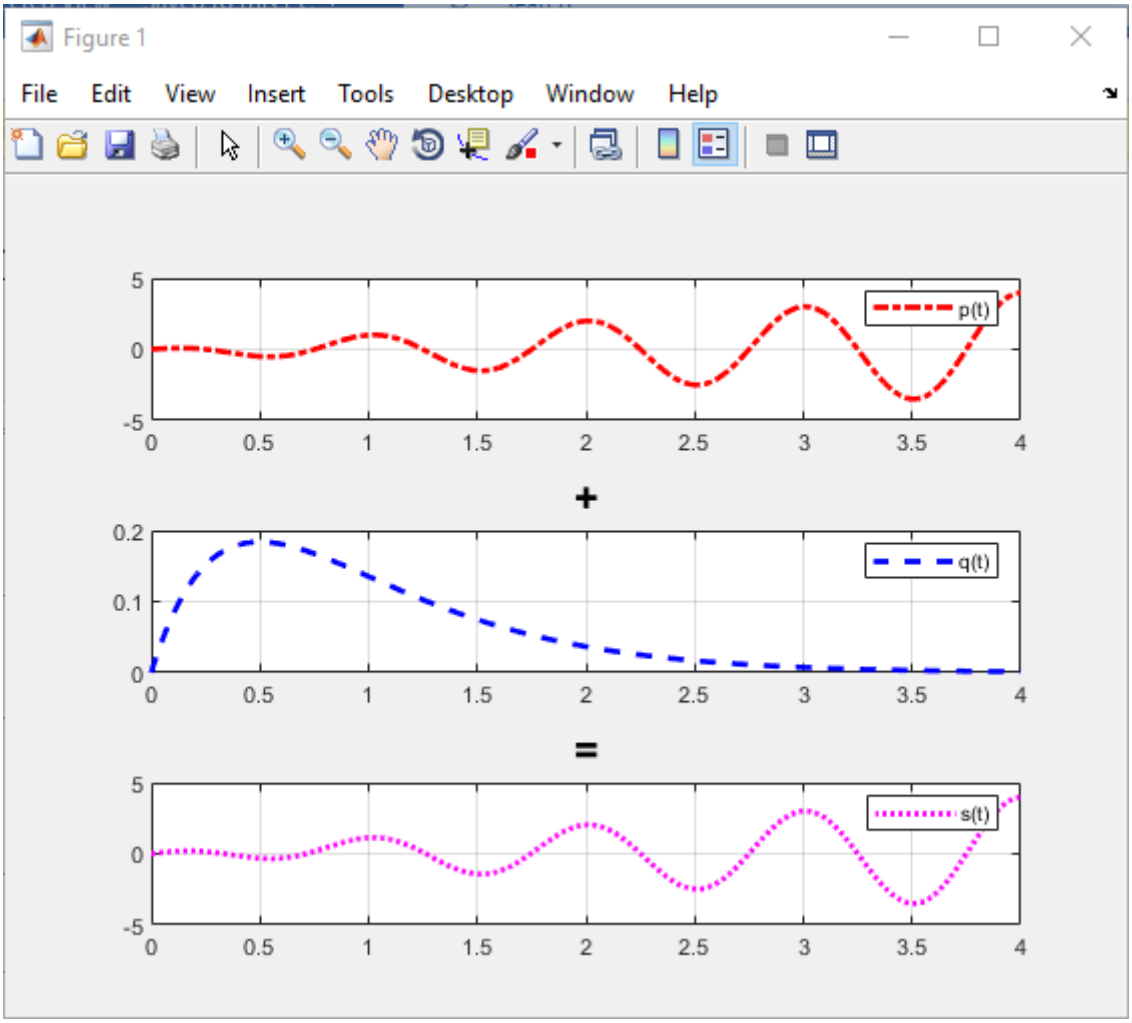
i. Make only one file for this task.

ii. Call functions within this m-file which is needed.
iii. Compute impulse response of the overall system.
iv. Compute the response of the overall system to the given input signal $x(t)$.
v. Plot all graphs $p(t), q(t), s(t)$ and $y(t)$.
vi. Determine if the overall system is BIBO stable or not.

```
close all
clc
clear all
step=0.01;

t1=0:0.01:4;
t2=t1;
t3=t2;
 figure();
h1= t1.*cos(2.*pi.*t1);
subplot(3,1,1);
plot(t1,h1,'r-.','linewidth',2),grid on
legend('p(t)');

h2= t2.*exp(-2.*t2);
subplot(3,1,2);
plot(t2,h2,'b--','linewidth',2),grid on
legend('q(t)');
title('+','fontsize',16)

h3=ones(size(t3));

h12=h1+h2;
subplot(3,1,3);
plot(t2,h12,'m:','linewidth',2),grid on
legend('s(t)');
title('=','fontsize',16)
%overall impulse response
figure();


th=0:step:8;
h= conv(h12,h3)*step;
subplot(3,1,1)
plot(th,h,'r-.','linewidth',2),grid on
legend('h(t)')
title('overall impulse response','fontsize',14)
 %input signal
tx=0:step:2;
x=tx.*exp(-2.*tx);
subplot(3,1,2);
plot(tx,x,'b--','linewidth',2),grid on
legend('x(t)')
title('Input signal','fontsize',14)
 %the response of the overall system to the given input signal ____
ty=0:0.01:10;
y=conv(x,h)*step;
subplot(3,1,3);
```

```
plot(ty,y,'m:','linewidth',2),grid on
legend('y(t)')
title('The response of the overall system to the given input
signal','fontsize',14)
```

BIBO STABLE



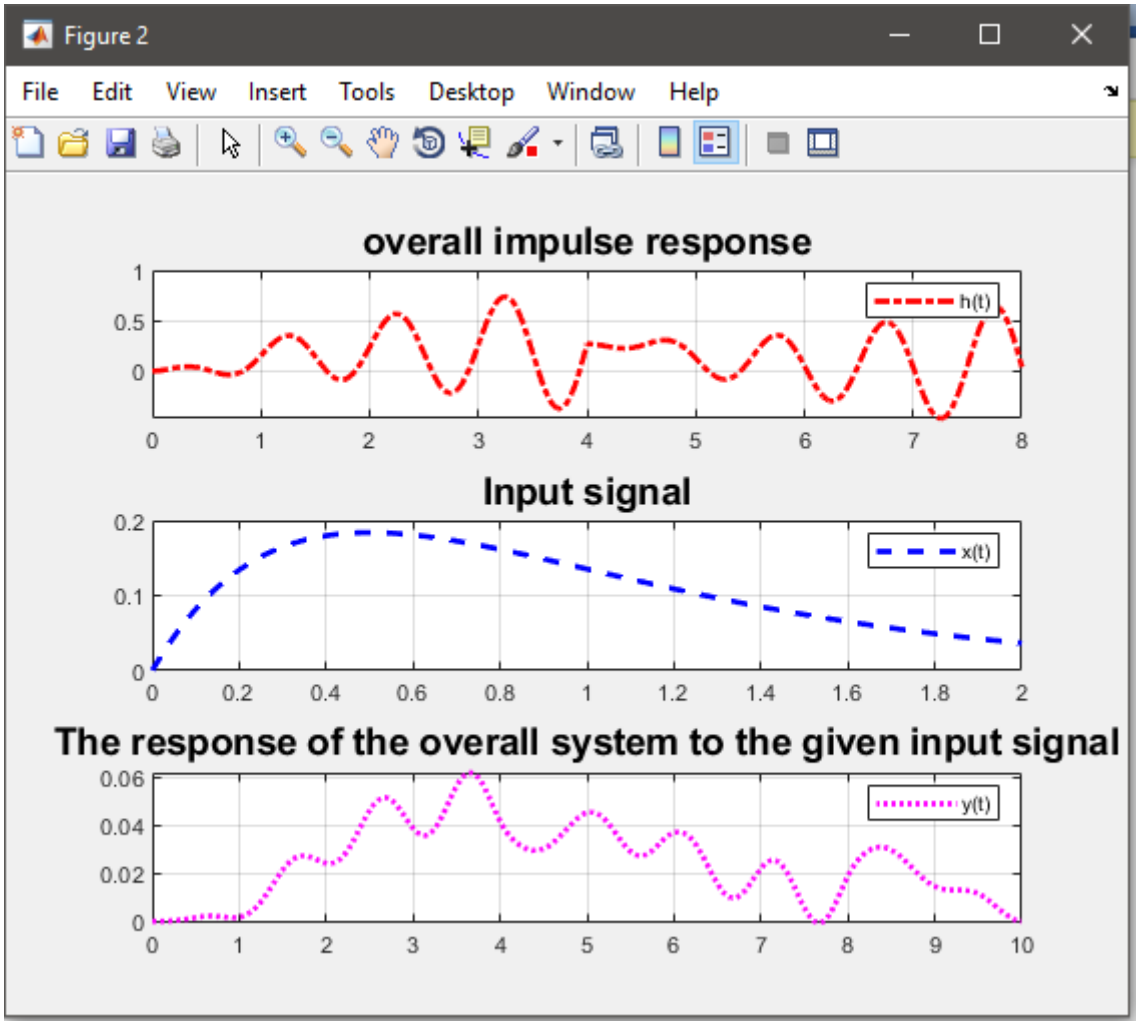
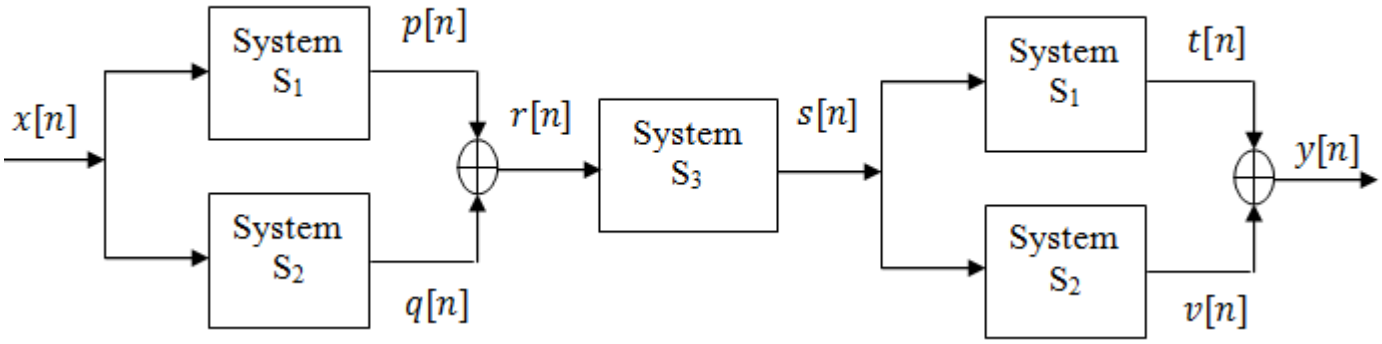


Figure 2
File Edit View Insert Tools Desktop Window Help
overall impulse response
h(t)
Input signal
x(t)
The response of the overall system to the given input signal
y(t)

Task 04: Suppose that the impulse responses of the sub-systems S_1, S_2 and S_3 that are connected as shown in the figure below are $h_1[n] = [2, 3, 4], 0 \leq n \leq 2$; $h_2[n] = [-1,3,1], 0 \leq n \leq 2$; and $h_3[n] = [1,1,-1], 0 \leq n \leq 2$, respectively. Compute



i. Make only one file for this task.
ii. Call functions within this m-file which is needed.
iii. Compute impulse response of the overall system.
iv. Compute the response of the overall system to the given input signal $x[n] = u[n] - u[n-2]$.
v. Plot all graphs $p[n], q[n], r[n], s[n], t[n], v[n]$ and $y[n]$.
vi. Determine if the overall system is BIBO stable or not.

```
n = 0:2;
h1 = [2,3,4];
h2 = [-1,3,1];
h3 = [1,1,-1];

subplot(5,1,1)
stem(n,h1,'fill','linewidth',2),grid on
legend('p[n]')

subplot(5,1,2)
stem(n,h2,'fill','linewidth',2),grid on
legend('q[n]')

h12=h1+h2;
subplot(5,1,3)
stem(n,h12,'fill','linewidth',2),grid on
legend('r[n]')

subplot(5,1,4)
stem(n,h3,'fill','linewidth',2),grid on
legend('h3[n]')

n123 = 0:4;
h123 = conv (h12,h3);
subplot(5,1,5)
```

```
stem(n123,h123,'fill','linewidth',2),grid on
legend('s[n]')


figure();
nt=0:6;
h1231=conv(h123,h1);
subplot(5,1,1)
stem(nt,h1231,'fill','linewidth',2),grid on
legend('t[n]')


nv=0:6;
h1232=conv(h123,h2);
subplot(5,1,2)
stem(nv,h1232,'fill','linewidth',2),grid on
legend('v[n]')



nh=0:6;
h=h1231+h1232;
subplot(5,1,3)
stem(nh,h,'g--','fill','linewidth',2),grid on
legend('h[n]')


xn=0:1;
x=ones(size(xn));
subplot(5,1,4)
```

```
stem(xn,x,'r:','fill','linewidth',2),grid on
legend('x[t]');


ty= 0:7;
y= conv(x,h);
subplot(5,1,5)
stem(ty,y,'r:','fill','linewidth',2),grid on
legend('y[n]')


syms n
ans=symsum (abs(h),n,0,inf)


ans =


[ Inf, Inf, Inf, Inf, Inf, Inf, Inf]
```

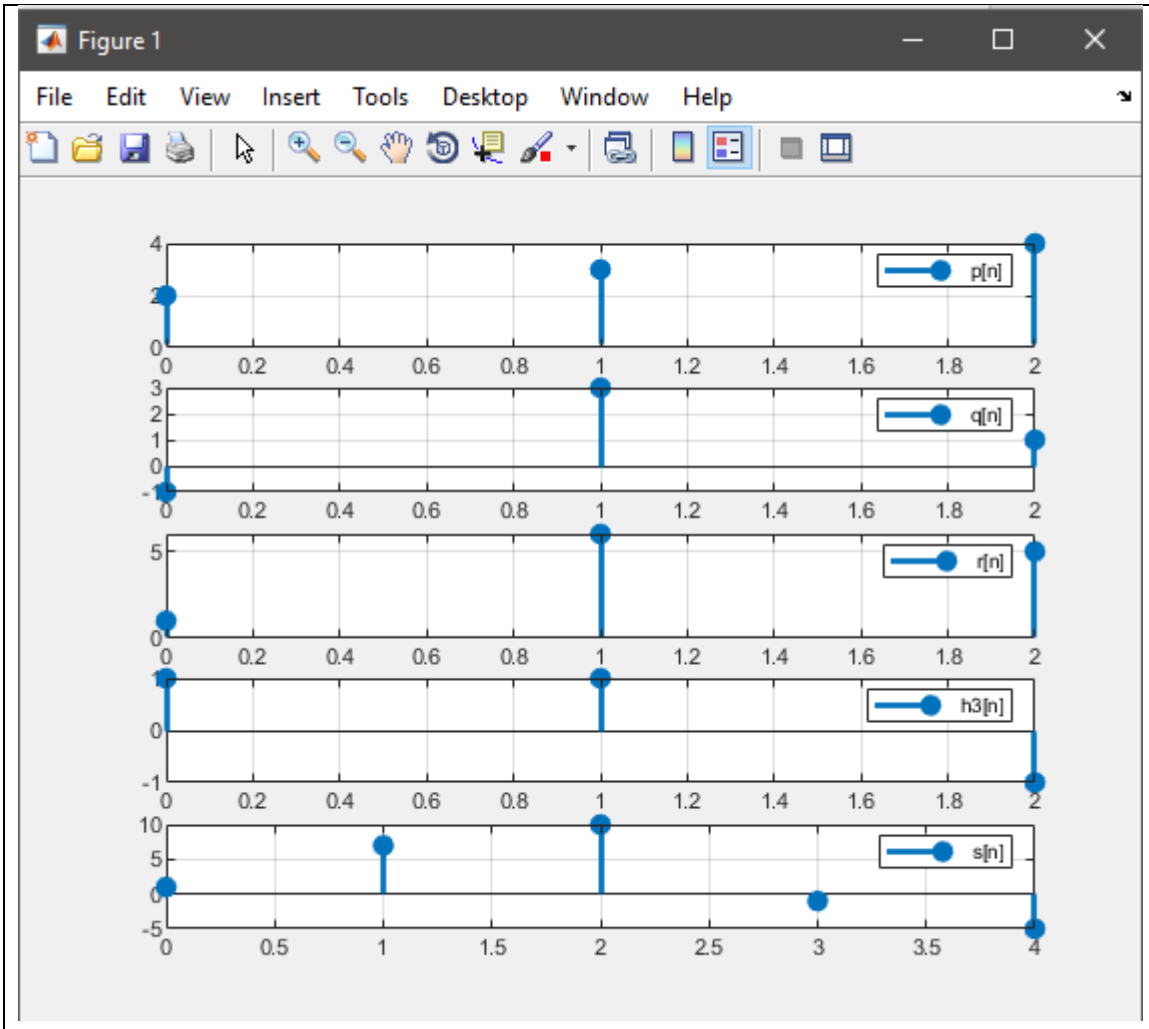


Figure 1
File Edit View Insert Tools Desktop Window Help
p[n]
q[n]
r[n]
h3[n]
s[n]

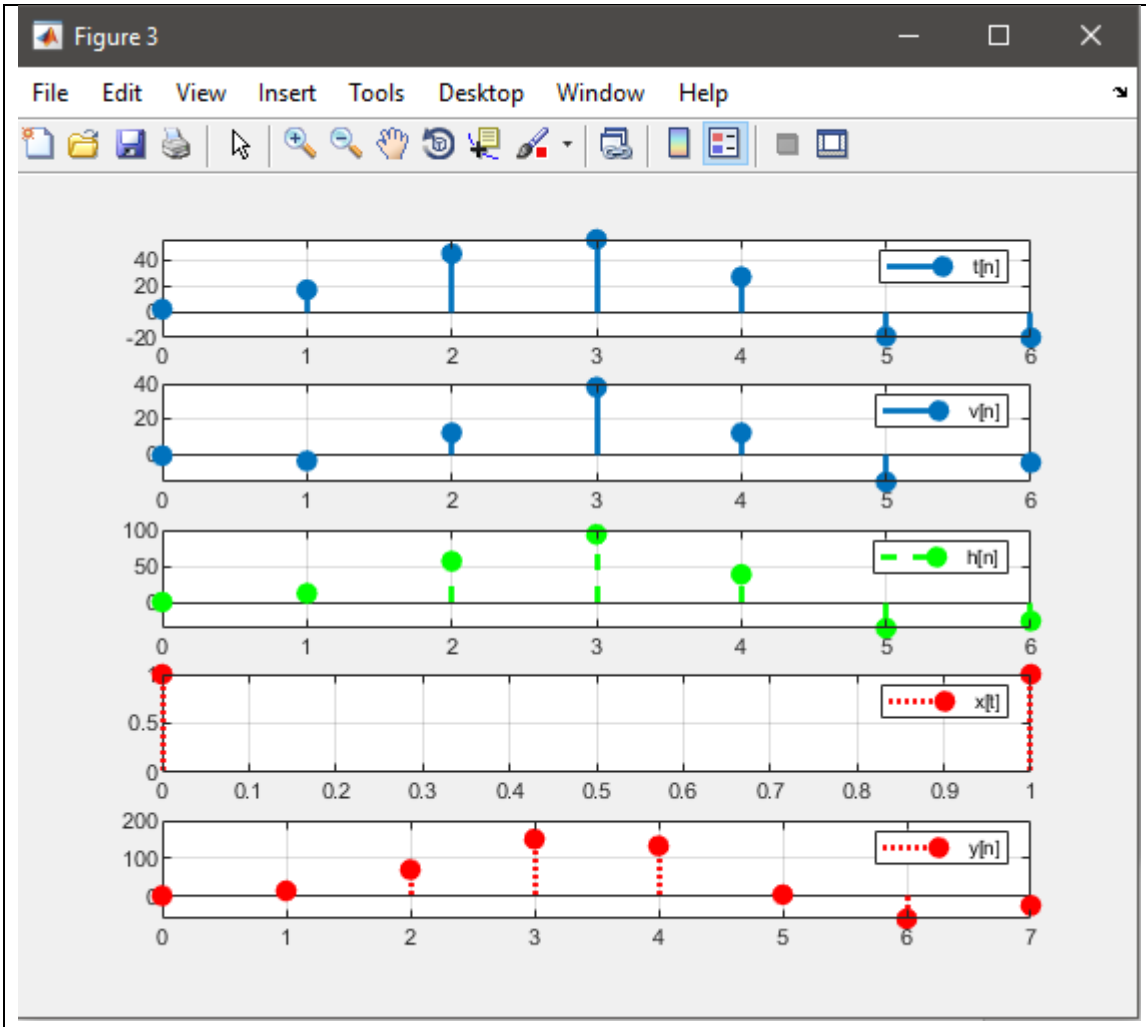


Figure 3
File Edit View Insert Tools Desktop Window Help
t[n]
v[n]
h[n]
x[t]
y[n]

Post-Lab Tasks

Critical Analysis / Conclusion

In this lab, we implemented various properties of convolution which includes commutative, associative, distributive and identity property. We also implemented them on different types of system: cascaded, parallel, mixed and feedback system and obtained the output using those properties in these systems.

Lab Assessment		
Pre-Lab	**/1**	**/10**
In-Lab	**/5**	
Critical Analysis	**/4**	
Instructor Signature and Comments		

LAB # 09

Complex Fourier Series Representation of Signals

Lab 09- Complex Fourier Series Representation of Signals

Pre Lab Tasks

In this lab session, we will introduce a way of analyzing/decomposing a continuous time signal into frequency components given by sinusoidal signals. This process in crucial in the signal processing field since it reveals the frequency content of signal and simplifies the calculation of systems' output. The analysis is based on the Fourier series. Up to this point, all signals were expressed in the time domain. With the use of Fourier series, a signal is expressed in the frequency domain and sometimes a frequency representation of a signal reveals more information about the signal than its time domain representation. There are three different and equal ways that can be used in order to express a signal into sum of simple oscillating functions, i.e., into a sum of sines, cosines, or complex exponentials. In this manual, symbols n and k are often swapped in order for the code written in examples to be in accordance with the theoretical mathematical equations.

9.1 Complex Exponential Fourier Series:

Suppose that a signal $x(t)$ is defined in the time interval $[t_0, t_0+T]$. Then, $x(t)$ is expressed in exponential Fourier series form (equation 9.1) as

$$x(t)=\sum_{k=-\infty}^{+\infty} a_k e^{jk\Omega_0 t}, \qquad t\in[t_0,t_0+T]$$

where,

Ω_0 is the fundamental frequency, and is given by $\Omega_0=(2\pi/T)$

t_0, T are real numbers

The terms a_k that appear in equation 9.1 are given by equation 9.2 as

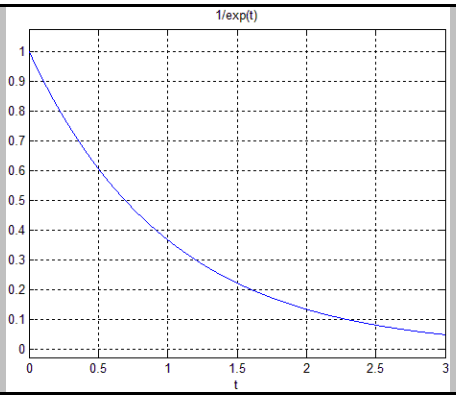
$$a_k=\frac{1}{T}\int_{t_0}^{t_0+T} x(t)e^{-jk\Omega_0 t}dt$$

The complex coefficients a_k are called complex exponential Fourier series coefficients, while a_0 is a real number and is called a constant or dc component. Each coefficient a_k corresponds to the projection of the signal $x(t)$ at the frequency $k\Omega_0$, which is known as k^{th} harmonic. The Fourier series expansion is valid only in the interval $[t_0, t_0+T]$, and the value of T defines the fundamental frequency Ω_0. As the Fourier series coefficients represent the signal in the frequency domain, they are also referred as the spectral coefficients of the signal.

Example:

Expand in complex exponential Fourier series the signal $x(t)=e^{-t}, 0\le t\le 3$.

The first thing that need to be done is to define the quantities $t_0=0$, $T=3$, and $\Omega_0=(2\pi/T)$. Moreover, the signal $x(t)$ is defined as symbolic expression.

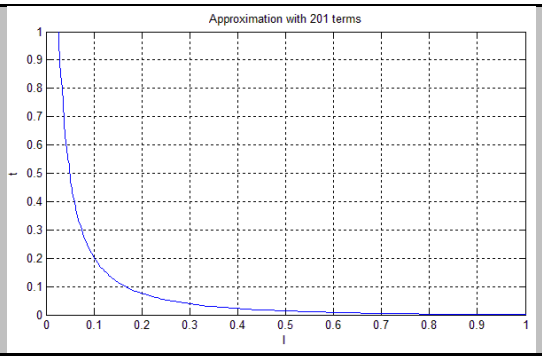
Commands	Results	Comments
t0=0; **T=3;** **w=2*pi/T;** **syms t** **x=exp(-t);** **ezplot(x,[t0 t0+T]), grid on**		Definition and graph of signal $x(t)$ in the time interval $[t_0, t_0+T]$

Afterwards, the coefficients a_k are computed according to equation 9.2. Looking into equation 9.1, we observe that Fourier coefficients a_k have to be calculated. Of course, this computation cannot be done in an analytical way. Fortunately, as the index k approaches toward $+\infty$ or toward $-\infty$, the Fourier coefficients a_k are approaching zero. Thus, $x(t)$ can be satisfactorily approximated by using a finite number of complex exponential Fourier series terms. Consequently, by computing the coefficients a_k for $-100 \le k \le 100$, i.e., by using first 201 complex exponential terms, a good approximation of $x(t)$ is expected. The approximate signal is denoted by $xx(t)$, and is computed by equation 9.3 given as

$$xx(t) = \sum_{k=-K}^{+K} a_k e^{jk\Omega_0 t}, \qquad t \in [t_0, t_0+T]$$

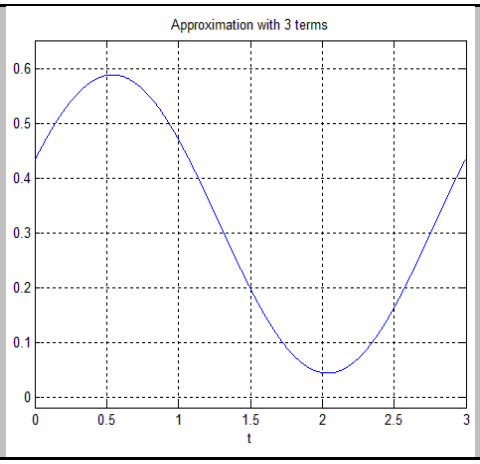
Commands	Comments
for k=-100:100 **a(k+101)=(1/T)*int(x*exp(-*k*w*t),t,t0,t0+T)** **end**	Calculation of coefficients a_k according to equation 9.2.

In order to define the vector a that contains the Fourier series coefficients a_k, $-100 \le k \le 100$, the syntax $a(k+101)$ is used for programming reasons, since in MATLAB the index of a vector cannot be zero or negative. Having calculated the coefficients a_k, the signal $x(t)$ is approximated according to equation 9.2, or more precisely according to equation 9.3. Note that equation 5.2 is sometimes called the synthesis equation, while we in this manual equation 9.3 is referred as analysis equation.

Commands	Results	Comments
for k=-100:100 **ex(k+101)=exp(j*k*w*t);** **end** **xx=sum(a.*ex)** **ezplot(xx,[t0 t0+T]), grid on** **title('Approximation with 201 terms')**		Initially the quantities $e^{jk\Omega_0 t}, -100 \le k \le 100$ are computed and afterward the signal is approximated according to equation 9.3.

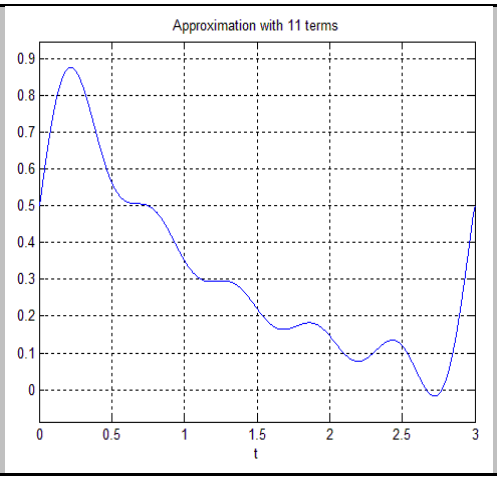
The plotted signal $xx(t)$ that is computed with the use of the complex exponential Fourier series is almost identical with the original signal $x(t)$. In order to understand the importance of number of terms used for approximation of original signal $x(t)$, the approximate signal $xx(t)$ is constructed for different values of k. First, the signal $x(t)$ is approximated by three exponential terms, i.e., the coefficients a_k are computed for $-1 \le k \le 1$.

Commands	Results	Comments

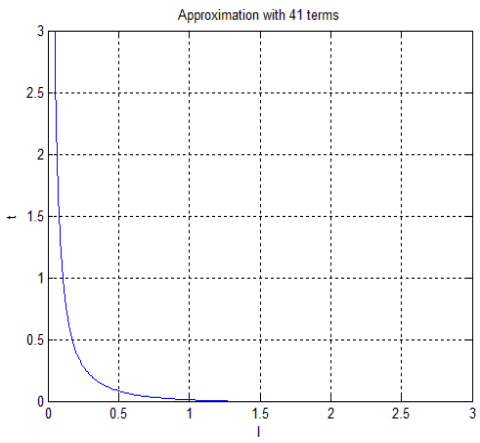
clear a ex;
for k=-1:1
a(k+2)=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);
end
for k=-1:1
ex(k+2)=exp(j*k*w*t);
end
xx=sum(a.*ex);
figure();
ezplot(xx,[t0 t0+T]), grid on
title('Approximation with 3 terms')
		When 3 terms are used in approximation of $x(t)$ by $xx(t)$,i.e., $-1 \le k \le 1$, the approximation signal $xx(t)$ is pretty dissimilar from the original signal $x(t)$.

Next, the signal $x(t)$ is approximated by 11 exponential terms, i.e., the coefficients a_k are computed for $-5 \le k \le 5$.

Commands	Results	Comments

for k=-5:5
a(k+6)=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);
end
for k=-5:5
ex(k+6)=exp(j*k*w*t);
end
xx=sum(a.*ex);
figure();
ezplot(xx,[t0 t0+T]), grid on
title('Approximation with 11 terms')
		It is clear that even when 11 terms are used, namely, $-5 \le k \le 5$, the approximation of $x(t)$ by $xx(t)$ is not good and is pretty dissimilar to $x(t)$.

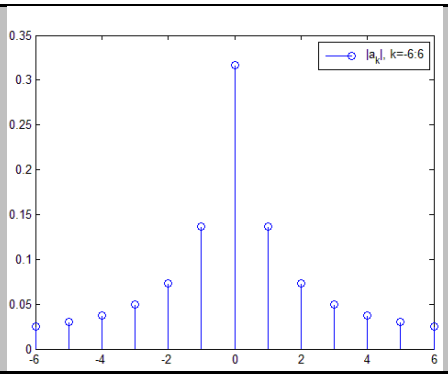
Finally, the signal $x(t)$ is approximated by 41 exponential terms, i.e., the coefficients a_k are computed for $-20 \le k \le 20$.

Commands	Results	Comments
for k=-20:20 **a(k+21)=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);** **end** **for k=-20:20** **ex(k+21)=exp(j*k*w*t);** **end** **xx=sum(a.*ex);** **figure();** **ezplot(xx,[t0 t0+T]), grid on** **title('Approximation with 41 terms')**		The signal $xx(t)$ is now computed from 41 terms and is starting to look quite similar to the original signal $x(t)$. Thus, this is a quite satisfactory approximation.

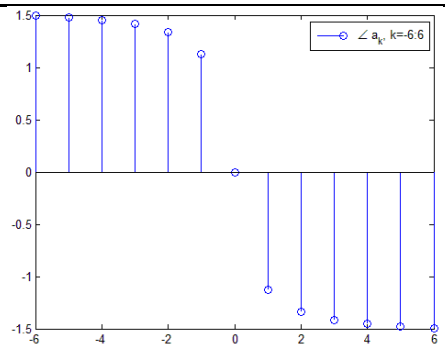
From the above analysis, it is clear that when many exponential terms are being considered in the construction of the approximate signal, a better approximation if the original signal is obtained. As illustrated in the beginning of the example, when 201 terms were used for the construction of the approximate signal, the obtained approximation was very good.

9.2 Plotting Fourier Series coefficients:

In the previous section, the Fourier coefficients were computed. In this section, the way of plotting them is presented. Once again, the signal the signal $x(t)=e^{-t}, 0\le t\le 3$ is considered. The coefficients a_k of the complex exponential form are the first that will be plotted for $-6\le k\le 6$ and for $-40\le k\le 40$. In the usual case, the coefficients a_k are complex numbers. A complex number z can be expressed as $z=|z|e^{j\angle z}$, where $|z|$ is the magnitude and $\angle z$ is the angle of z. Therefore, in order to create the graph of the coefficients a_k of the complex exponential form, the magnitude and the angle of each coefficient have to be plotted.

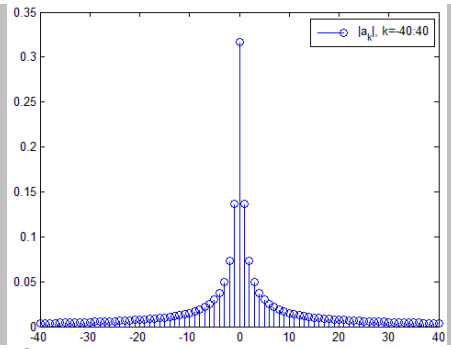
Commands	Results	Comments
syms t k n **x=exp(-t);** **t0=0;** **T=3;** **w=2*pi/T;** **a=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);** **k1=-6:6;** **ak=subs(a,k,k1);** **stem(k1,abs(ak));** **legend('|a_k|, k=-6:6')**		The signal $x=e^{-t}, 0\le t\le 3$ is defined and the magnitude of the coefficients is plotted for $-6\le k\le 6$.

```
stem(k1,angle(ak));
legend('\angle a_k, k=-6:6')
```

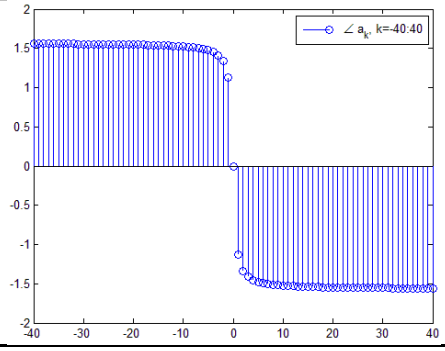


The angles of coefficients are plotted for $-6 \le k \le 6$. It is worth noticing the way that the angle symbol is drawn through the legend command.

```
k2=-40:40;
ak2=subs(a,k,k2);
stem(k2,abs(ak2));
legend('|a_k|, k=-40:40')
```




```
stem(k2,angle(ak2));
legend('\angle a_k, k=-40:40')
```



9.3 Fourier Series of Complex Signals:

So far, we were dealing with real signals. In this section, we examine the Fourier series representation of a complex valued signal.

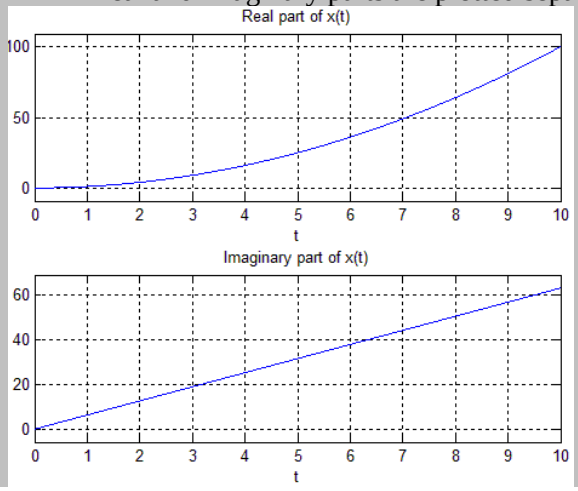
Example:

Compute the coefficients of the complex exponential Fourier series and the trigonometric Fourier series of the complex signal $x(t) = t^2 + j2\pi t, 0 \le t \le 10$. Moreover, plot the approximate signals using5 and 41 components of the complex exponential form.

Commands	Results/Commands

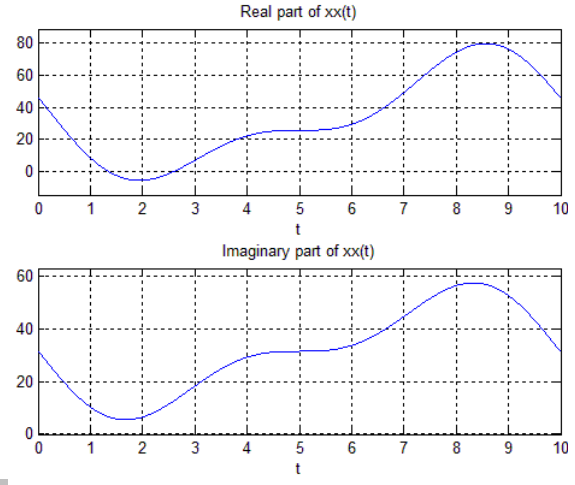
```
syms t
t0=0;
T=10;
w=2*pi/T;
x=t^2+j*2*pi*t;
subplot(211)
ezplot(real(x),[t0 T]),grid on;
title('Real part of x(t)');
subplot(212)
ezplot(imag(x),[t0 T]),grid on;
title('Imaginary part of x(t)');
```

The signal $x(t) = t^2 + j2\pi t$ is complex; hence, its real and imaginary parts are plotted separately.

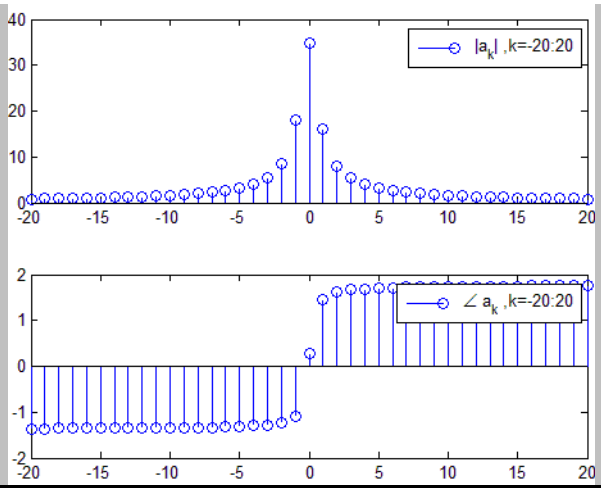



```
for k=-2:2
   a(k+3)=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);
   ex(k+3)=exp(j*k*w*t);
end
xx=sum(a.*ex);
subplot(211)
ezplot(real(xx),[t0 T]),grid on;
title('Real part of xx(t)');
subplot(212)
ezplot(imag(xx),[t0 T]),grid on;
title('Imaginary part of xx(t)');
```

Computation of the first five coefficients of complex exponential form and graph of the approximate signal is obtained with five terms. The coefficients of the complex exponential form are complex, thus their magnitude and phase are plotted.




```
a1=eval(a)
subplot(211);
stem(-20:20,abs(a1));
legend ('|a_k| ,k=-20:20')
subplot(212);
stem(-20:20,angle(a1));
legend ('\angle a_k ,k=-20:20')
```



Evaluate the approximation by 41 components yourself.

9.3 Fourier Series of Periodic Signals:

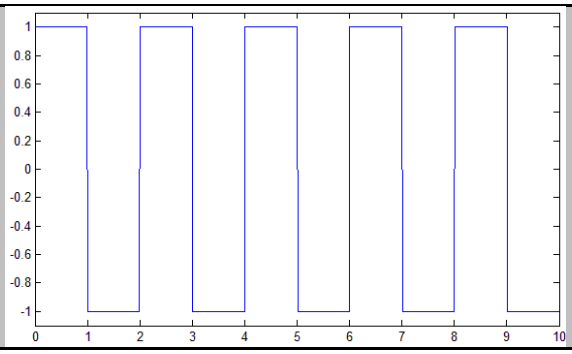
In the previous sections, the Fourier series expansion of a signal was defined in a close time interval $[t_0, t_0 + T]$. Beyond this interval, the Fourier series expansion does not always converge to the original signal $x(t)$. In this section, we introduce the case where the signal $x(t)$ is a periodic signal with period T, i.e., $x(t) = x(t+T)$. In this case, the Fourier series is also periodic with period T; thus converges to $x(t)$ for $-\infty \le t \le +\infty$.

Example:

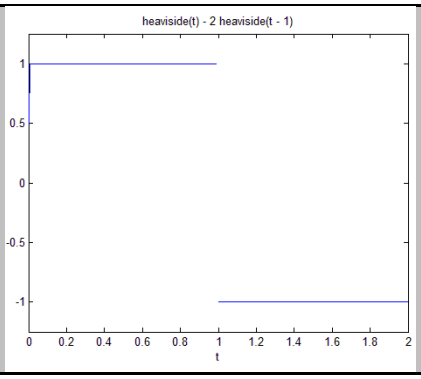
Approximate by Fourier series, the periodic signal that in one period is given by

$$x(t) = \begin{cases} 1, & 0 \le t \le 1 \\ -1, & 1 \le t \le 2 \end{cases}$$

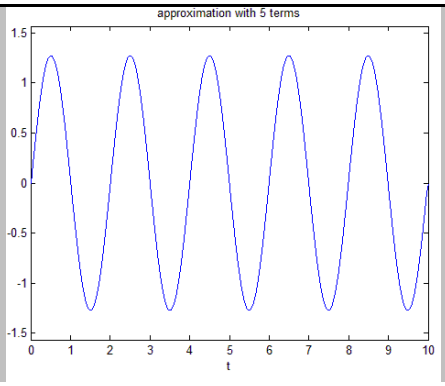
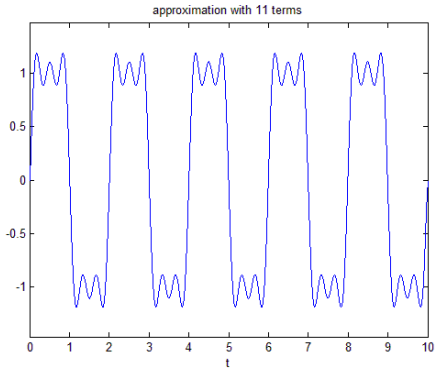
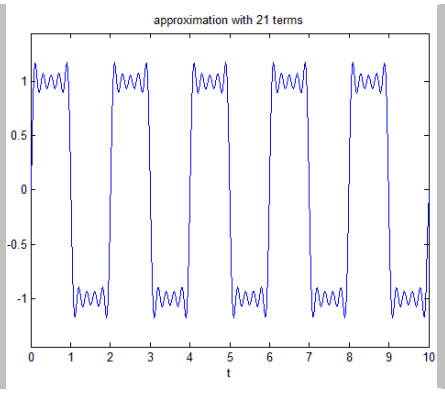
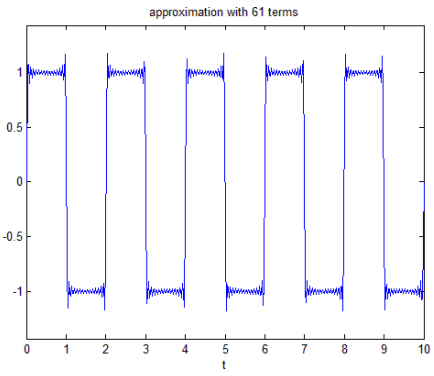
First the signal $x(t)$ is plotted over the time of five periods for reference reasons.

Commands	Results/Comments
t1=0:.01:1; **t2=1.01:.01:2;** **x1=ones(size(t1));** **x2=-ones(size(t2));** **x=[x1 x2];** **xp=repmat(x,1,5);** **t=linspace(0,10,length(xp));** **plot(t,xp)**	

Afterwards, the two part signal $x(t)$ is defined as single symbolic expression $x(t) = u(t) - 2u(t-1), 0 \le t \le 2$, where $u(t)$ is the unit step function. Note that the periodic signal $x(t)$ is entirely determined by its values over one period. Thus, the symbolic expression of $x(t)$ is only defined for time interval of interest, namely, $0 \le t \le 2$ (one period). The defined symbolic expression is plotted in one period for confirmation.

Commands	Results
syms t **x=heaviside(t)-2*heaviside(t-1);** **ezplot(x,[0 2]);**	

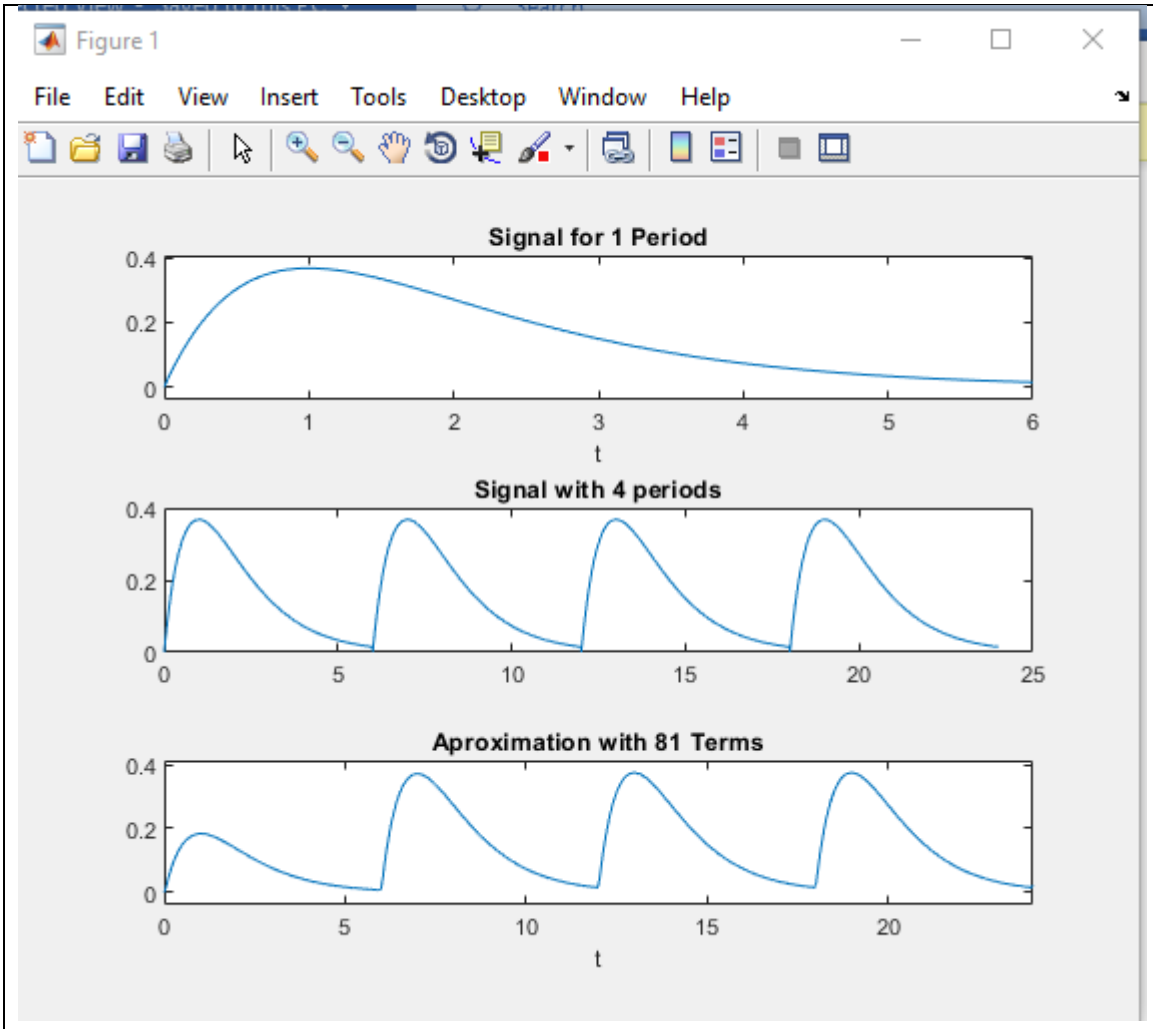
Finally, the complex exponential Fourier series coefficients a_k are calculated and the approximate signal $xx(t)$ is computed and plotted for $0 \le t \le 10$, i.e., for time of five periods. As in previous examples, $xx(t)$ is computed and plotted for various number of exponential terms used.

Commands	Results
k=-2:2; **t0 =0;** **T=2;** **w=2*pi/T;** **a=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T)** **xx=sum(a.*exp(j*k*w*t))** **ezplot(xx,[0 10])** **title('approximation with 5 terms')**	
k=-5:5; **a=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);** **xx=sum(a.*exp(j*k*w*t));** **ezplot(xx,[0 10])** **title('approximation with 11 terms')**	
k=-10:10; **a=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);** **xx=sum(a.*exp(j*k*w*t));** **ezplot(xx,[0 10])** **title('approximation with 21 terms')**	
k=-30:30; **a=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);** **xx=sum(a.*exp(j*k*w*t));** **ezplot(xx,[0 10])** **title('approximation with 61 terms')**	

In-Lab Tasks

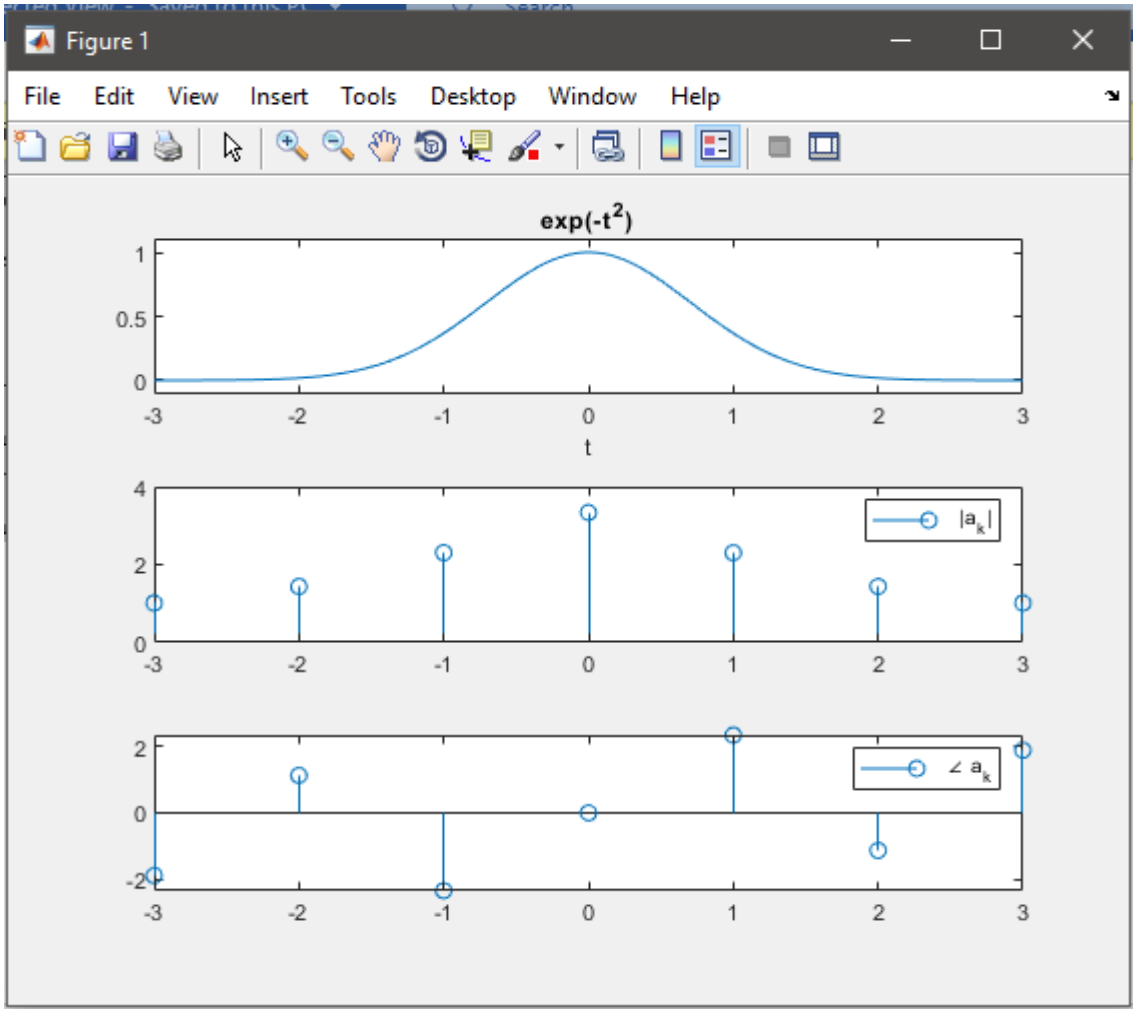
Task 01: The periodic signal $x(t)$ is defined in one period as $x(t) = te^{-t}, 0 \le t \le 6$. Plot in time of four periods the approximate signals using 81 terms of complex exponential form of Fourier series.

```
t0 = 0;
T = 6;
w = 2.*pi./T;
syms t
x = t.*exp(-t);
subplot(3,1,1)
ezplot(x, [t0 t0+T])
title('Signal for 1 Period')
t1 = t0:0.01:T;
xx = t1.*exp(-t1);
xrepeat = repmat(xx,1,4);
tt = linspace(0,4.*T,length(xrepeat));
subplot(3,1,2)
plot(tt,xrepeat)
title('Signal with 4 periods')
for k = -40:40
a(k+41) = (1/T).*int(x.*exp(-j*k*w*t), t, t0, t0+t);
end
for k = -40:40
ex(k+41) = exp(j*k*w*t);
end
xx1 = sum(a.*ex);
subplot(3,1,3)
ezplot(xx1, [t0 t0+4*T])
title('Aproximation with 81 Terms')
```



Task 02: Plot the coefficients of the complex exponential Fourier series for the periodic signal that in one period is defined by $x(t) = e^{-t^2}, -3 \le t \le 3$.

```
t0 = -3;
T = 6;
w = 2.*pi./T;
syms t
x = exp(-t.^2);
subplot(3,1,1)
ezplot(x, [t0 t0+T])
syms t k n
x = exp(-t);
a = (1/T)*int(x.*exp(-j*k*w*t), t, t0, t0+T);
k1 = -3:3;
ak = subs(a, k, k1);
subplot(3,1,2)
stem(k1,abs(ak));
legend('|a_k|')
subplot(3,1,3)
stem(k1,angle(ak));
legend('\angle a_k')
```



Task 03: The periodic signal $x(t)$ in a period is given by

$$x(t)=\begin{cases}1, & 0\le t\le 1\\ 0, & 1\le t\le 2\end{cases}$$

Plot in one period the approximate signals using 41 and 201 term of the complex exponential Fourier series. Furthermore, each time plot the complex exponential coefficients.

```
t0 = 0;
T = 2;
w = 2*pi/T;
syms t n k
x=heaviside(t)-heaviside(t-1)+heaviside(t-2)-heaviside(t-3);
ezplot(x, [-1 3])
title('heaviside(t)-heaviside(t-1)');
figure();
x=heaviside(t)-heaviside(t-1);

for k=-20:20
a(k+21)=(1/T)*int(x*exp(-1i*k*w*t),t,t0,t0+T);
end
for k=-20:20
   ex(k+21)=exp(1i*w*k*t);
end
f=sum(a.*ex);
ezplot(f,[-1 3])
title('approximated signal with 41 terms')
figure();

for k=-100:100
a(k+101)=(1/T)*int(x*exp(-1i*k*w*t),t,t0,t0+T);
end
for k=-100:100
   ex(k+101)=exp(1i*w*k*t);
end
f2=sum(a.*ex);
ezplot(f2,[-1 3])
title('approximated signal with 101 terms')
figure();

new_int = -100:100;
ak=subs(a,k,new_int);
subplot(2,1,1);
stem(new_int,abs(a));
legend('|a_k|')
subplot(2,1,2);
stem(new_int,angle(a));
legend('\angle a_k')
```

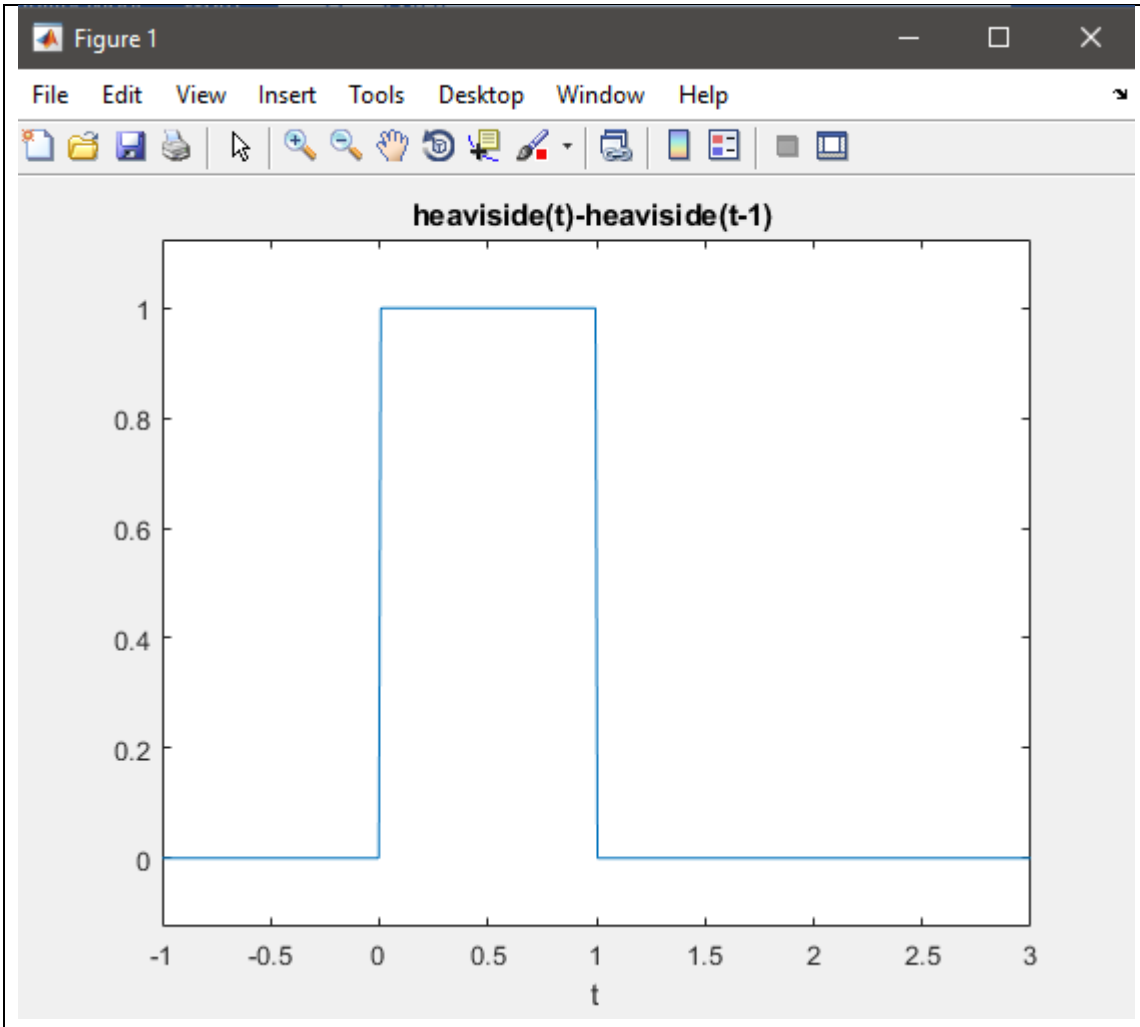


Figure 1
File Edit View Insert Tools Desktop Window Help
heaviside(t)-heaviside(t-1)
1
0.8
0.6
0.4
0.2
0
-1
-0.5
0
0.5
1
1.5
2
2.5
3
t

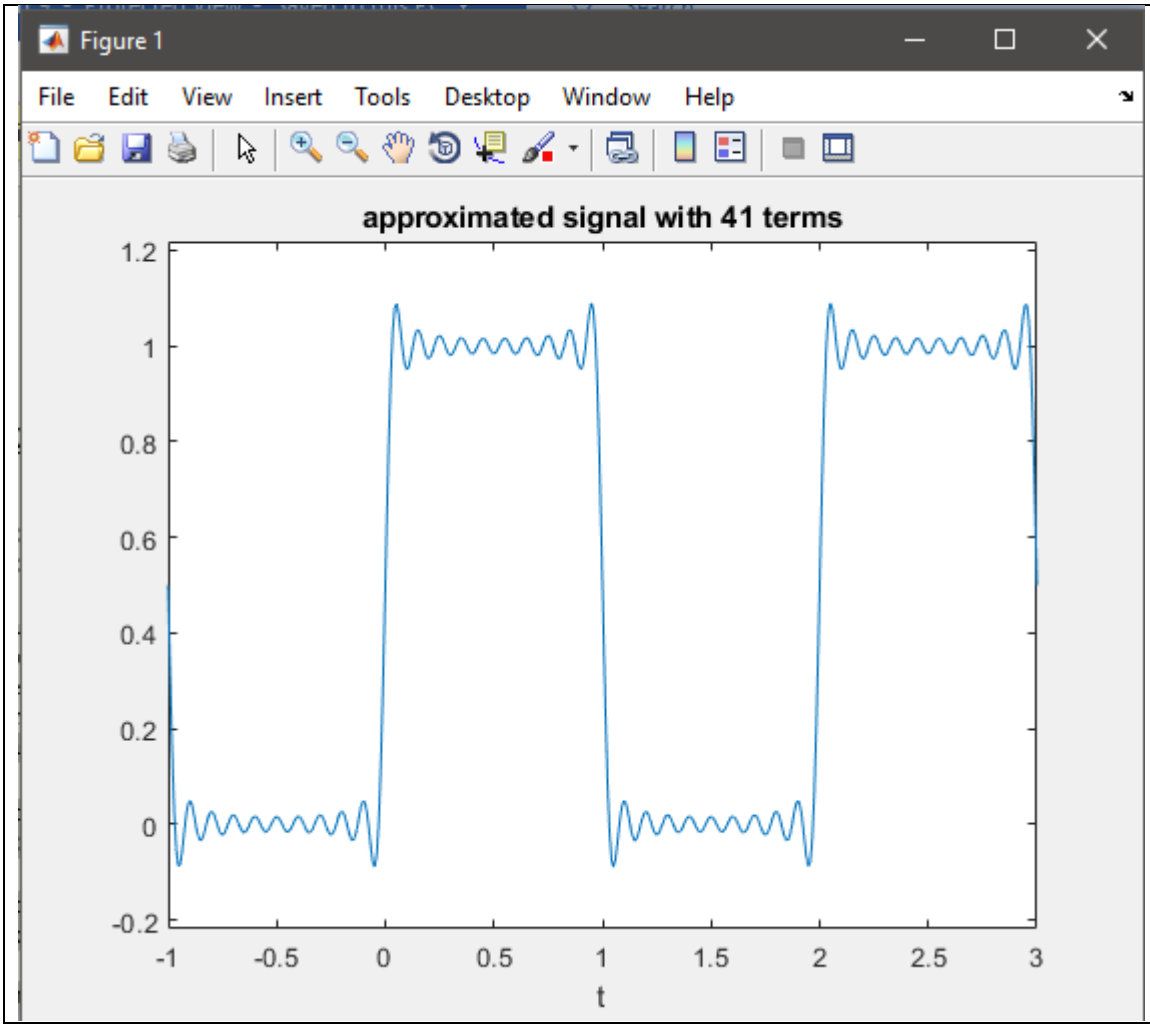


Figure 1
File Edit View Insert Tools Desktop Window Help
approximated signal with 41 terms
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-1 -0.5 0 0.5 1 1.5 2 2.5 3
t

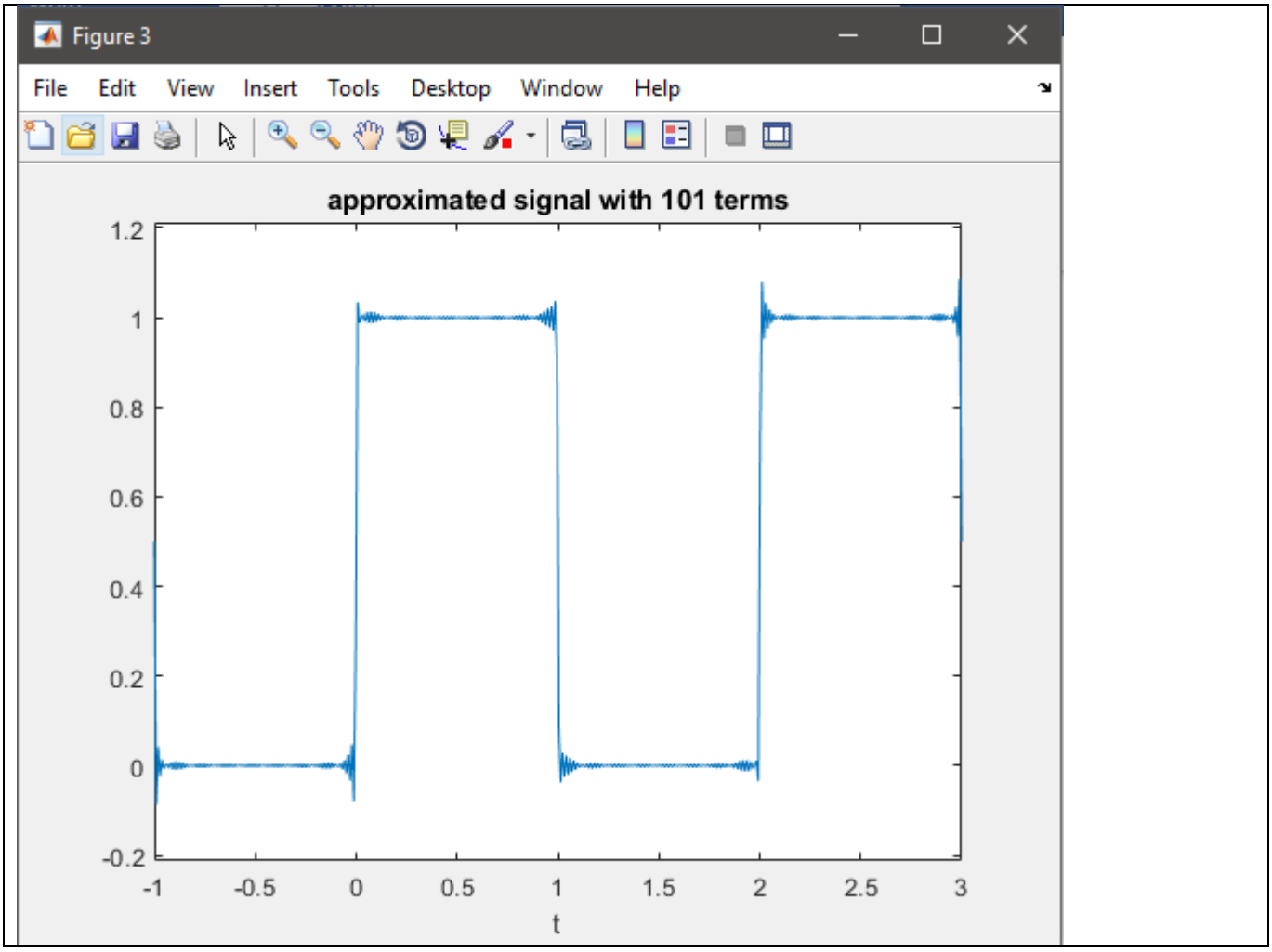
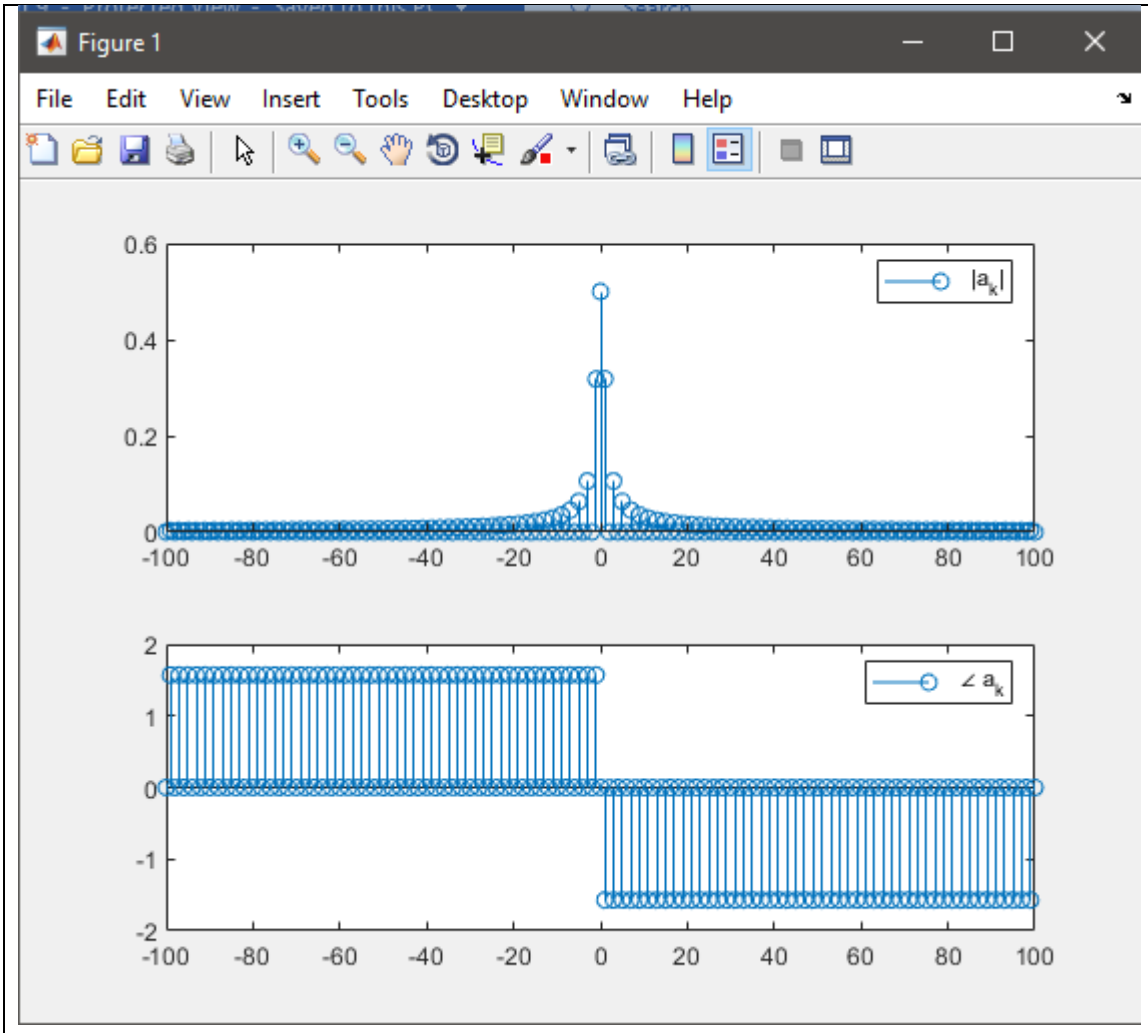


Figure 3
File Edit View Insert Tools Desktop Window Help
approximated signal with 101 terms
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-1 -0.5 0 0.5 1 1.5 2 2.5 3
t



Task 04: The periodic signal $x(t)$ in a period is given by

$$x(t)=\begin{cases}1, & 0\le t\le 1\\ 2-t, & 1\le t\le 2\end{cases}$$

Calculate the approximation percentage when the signal $x(t)$ is approximated by 3, 5, 7, and 17 terms of the complex exponential Fourier series. Furthermore, plot the signal in each case.

```
syms t k
x=(heaviside(t)-heaviside(t-1))+(2-t)*(heaviside(t-1)-heaviside(t-2));
inter = -1:3;
ezplot(x, inter)
t0=0;
T=2;
w=2*pi/T;

for k=-8:8
a(k+9)=(1/T)*int(x*exp(-1i*k*w*t),t,t0,t0+T);
end

for k=-8:8
   ex(k+9)=exp(1i*w*k*t);
end

x2=sum(a.*ex);
ezplot(x2,[-1 3])
title('approximated signal with 17 terms')

syms k
k1 = -8:8;
figure();
subplot(2,1,1);
stem(k1,abs(a));
legend('|a_k|')
subplot(2,1,2);
stem(k1,angle(a));
legend('\angle a_k')
```

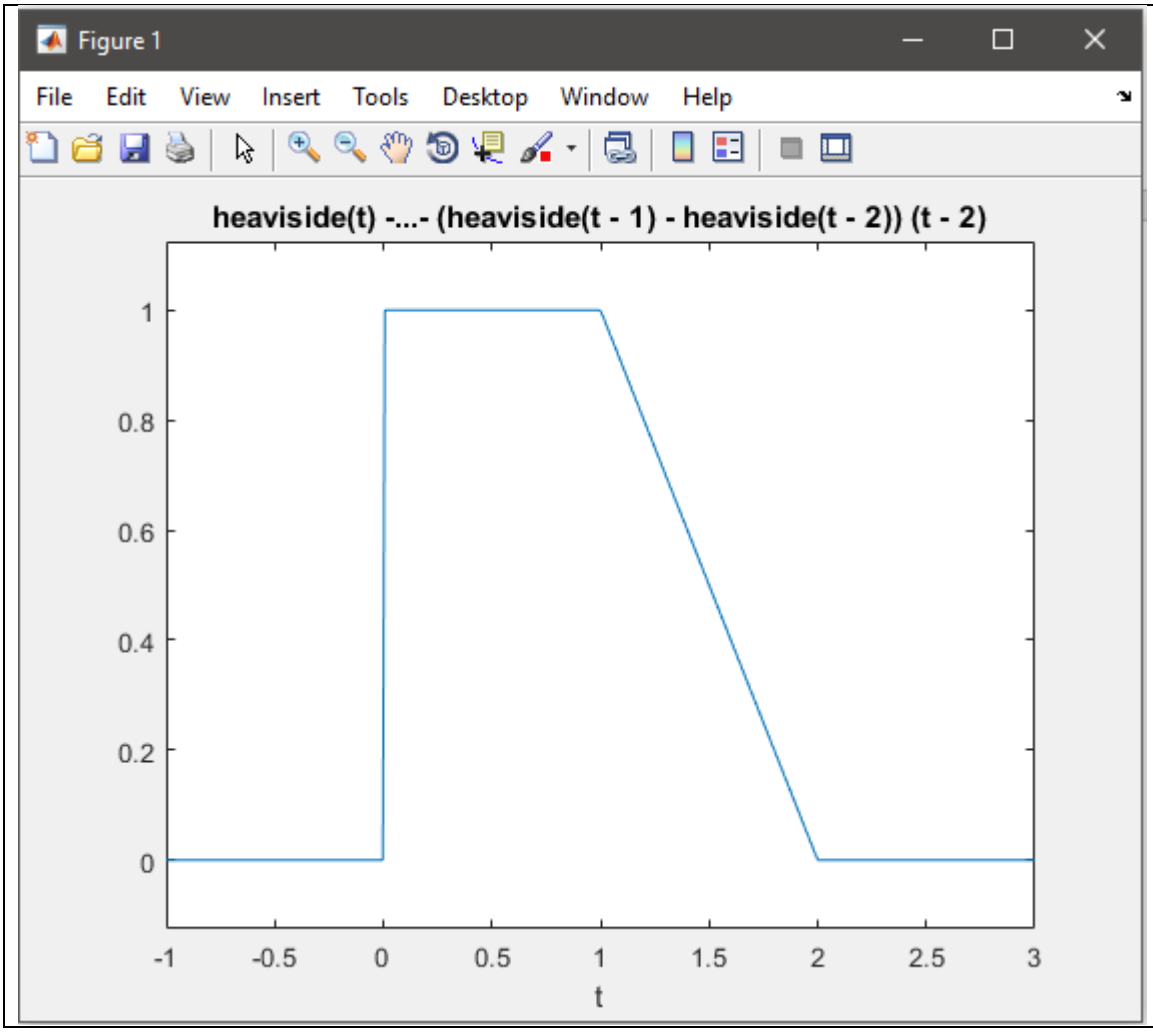


Figure 1
File Edit View Insert Tools Desktop Window Help
heaviside(t) -...- (heaviside(t - 1) - heaviside(t - 2)) (t - 2)
1
0.8
0.6
0.4
0.2
0
-1
-0.5
0
0.5
1
1.5
2
2.5
3
t

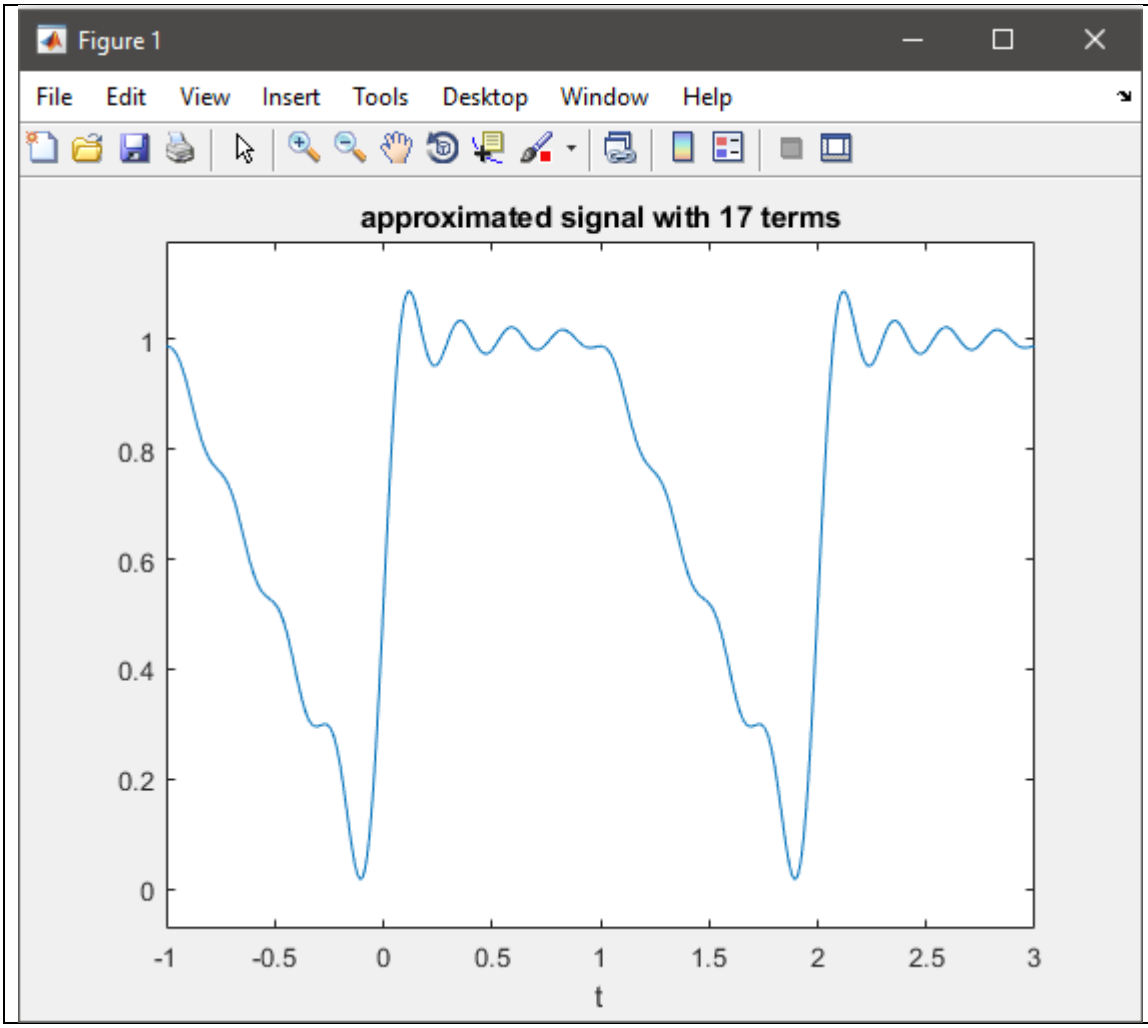


Figure 1
File Edit View Insert Tools Desktop Window Help
approximated signal with 17 terms
1
0.8
0.6
0.4
0.2
0
-1
-0.5
0
0.5
1
1.5
2
2.5
3
t

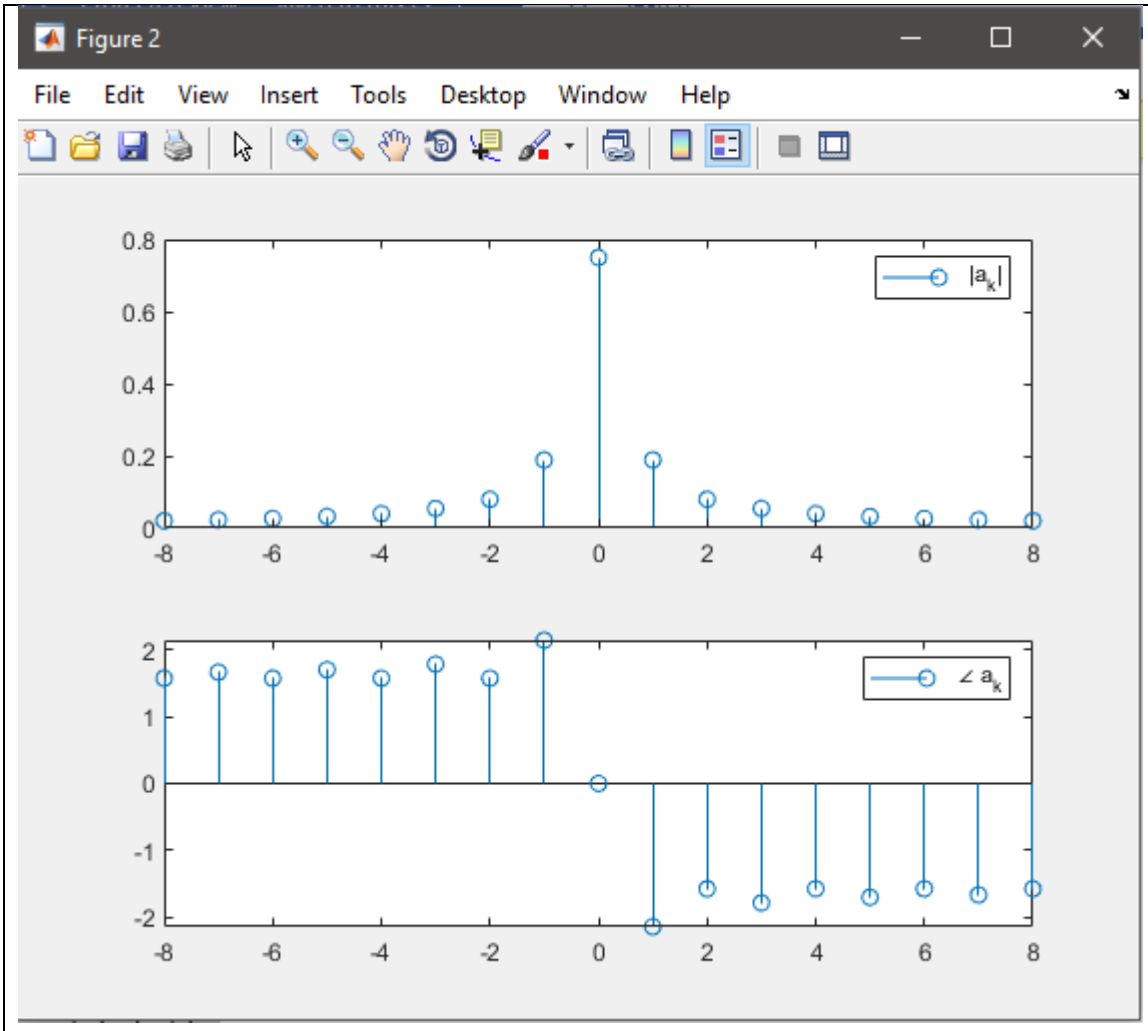


Figure 2
File Edit View Insert Tools Desktop Window Help
|a_k|
∠ a_k

Post-Lab Task

Critical Analysis /Conclusion

In this lab, we learnt the effect of increased number of terms on graphs of signals, we also found coefficients “ak” complex Fourier series in MATLAB. We plotted and observed the complex exponential Fourier series. Moreover, we observed signals approximated in more than one terms of Fourier Series.

Lab Assessment		
Pre-Lab	**/1**	**/10**
In-Lab	**/5**	
Critical Analysis	**/4**	
Instructor Signature and Comments		

LAB # 10

Trigonometric (Real) Fourier Series Representation and its Properties

Lab 10- Trigonometric (Real) Fourier Series Representation and its Properties

Pre-Lab Tasks

10.1 Trigonometric Fourier Series:

A second form of Fourier series is introduced in this section. Suppose that a signal $x(t)$ is defined in the time interval $[t_0, t_0+T]$. Then $x(t)$, by using the trigonometric Fourier series, can be expressed in time interval $[t_0, t_0+T]$ as a sum of sinusoidal signals, namely, sines and cosines, where each signal has frequency $k\Omega_0$ rad/s.

The mathematical expression (equation 10.1) is

$$x(t) = a_0 + \sum_{k=1}^{\infty} b_k \cos(k\Omega_0 t) + \sum_{k=1}^{\infty} c_k \sin(k\Omega_0 t)$$

The coefficients a_0, b_1, b_2, . . ., c_1, c_2, . . . of the trigonometric Fourier series are computed by

$$a_0 = \frac{1}{T}\int_{t_0}^{t_0+T} x(t)\,dt \qquad 10.2$$

$$b_n = \frac{2}{T}\int_{t_0}^{t_0+T} x(t)\cos(n\Omega_0 t)\,dt, \quad n = 1,2,3,... \qquad 10.3$$

$$c_n = \frac{2}{T}\int_{t_0}^{t_0+T} x(t)\sin(n\Omega_0 t)\,dt, \quad n = 1,2,3,... \qquad 10.4$$

Example:

The signal that will be expanded is the same signal used at the previous example. Thus, the problem is to expand in trigonometric Fourier series the signal $x(t) = e^{-t}, 0 \le t \le 3$.

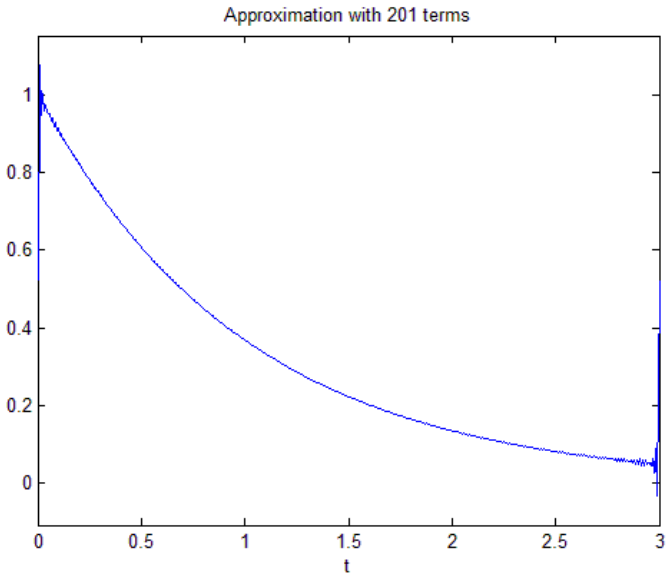
First the trigonometric Fourier coefficients b_n, c_n and the dc component a_0 are computed according to equations 10.3, 10.4 and 10.2, respectively, for $n = 1, 2, ..., 200$. Next, $x(t)$ is approximated according to the relationship (equation 10.5)

$$x(t) = a_0 + \sum_{k=1}^{N} b_k \cos(k\Omega_0 t) + \sum_{k=1}^{N} c_k \sin(k\Omega_0 t)$$

Commands	Results/Comments

T=3;
t0=0;
w=2*pi/T;
syms t
x=exp(-t);
	Definition of $t_0 = 0, T = 3, \Omega_0 = 2\pi/T$ and of the signal $x(t)$

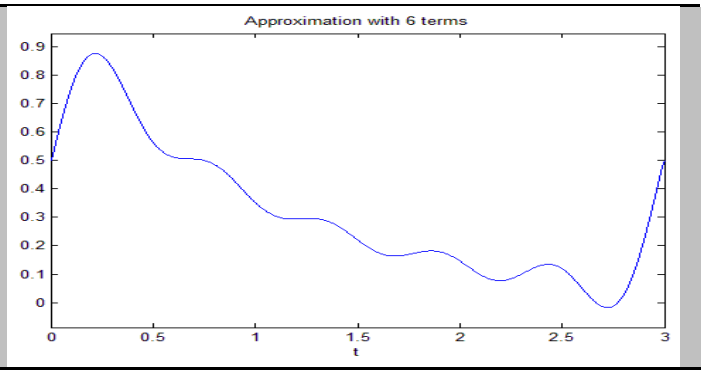
a0=(1/T)*int(x,t,t0,t0+T);
for n=1:200
b(n)=(2/T)*int(x*cos(n*w*t),t,t0,t0+T);
end
for n=1:200
c(n)=(2/T)*int(x*sin(n*w*t),t,t0,t0+T);
end
	Computation of trigonometric coefficients according to equations 10.2, 10.3 and 10.4.

k=1:200; xx=a0+sum(b.*cos(k*w*t))+sum(c.*sin(k*w*t))	The signal $x(t)$ is approximated by equation 10.1 or more precisely from equation 10.5 with N=200.
ezplot(xx, [t0 t0+T]); title('Approximation with 201 terms')	Graph of the approximate signal $xx(t)$ that was computed by the terms of the trigonometric Fourier series.  The approximation with 201 terms of the original signal $x(t)$ is very good.

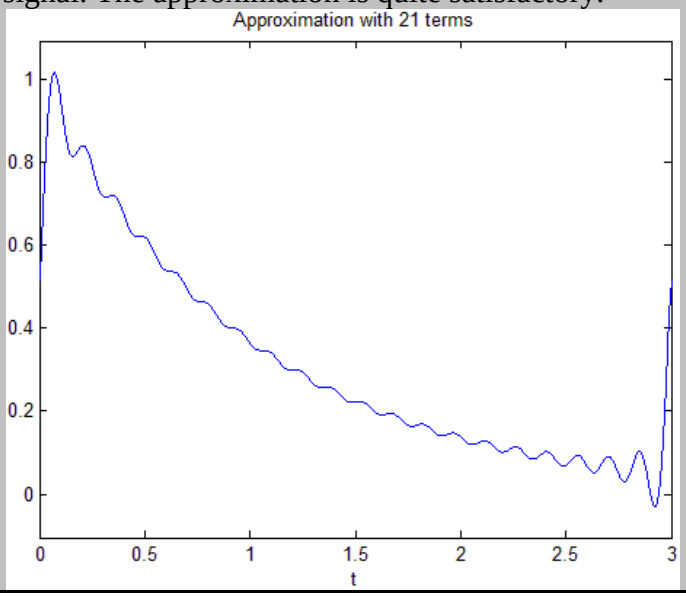
In order to understand the importance of the number terms used for the approximation of the original signal $x(t)$, the approximate signal $xx(t)$ is constructed for different values of n. Approximation with five terms ($n = 1,...,5$)

Commands	Results/Comments

clear b c
for n=1:5
b(n)=(2/T)*int(x*cos(n*w*t),t,t0,t0+T);
c(n)=(2/T)*int(x*sin(n*w*t),t,t0,t0+T);
end
k=1:5;
xx=a0+sum(b.*cos(k*w*t))+sum(c.*sin(k*w*t))
ezplot(xx, [t0 t0+T]);
title('Approximation with 6 terms')
	When the signal is approximated with 5 terms, it is pretty dissimilar to the original signal.



Approximation with 20 terms $(n = 1,...,20)$

Commands	Results/Comments
for n=1:20 **b(n)=(2/T)*int(x*cos(n*w*t),t,t0,t0+T);** **c(n)=(2/T)*int(x*sin(n*w*t),t,t0,t0+T);** **end** **k=1:20;** **xx=a0+sum(b.*cos(k*w*t)) +sum(c.*sin(k*w*t));** **ezplot(xx, [t0 t0+T]);** **title('Approximation with 21 terms')**	The approximated signal $xx(t)$ is now approximated by 20 terms and it is quite similar to the original signal. The approximation is quite satisfactory. 

10.2 Properties of Fourier Series:

10.2.1 Linearity

Suppose that the complex exponential Fourier series coefficients of the periodic signals $x(t)$ and $y(t)$ are denoted by a_k and b_k, respectively, or In other words $x(t) \to a_k$ and $y(t) \to b_k$. Moreover, let z_1, z_2 denote two complex numbers. Then

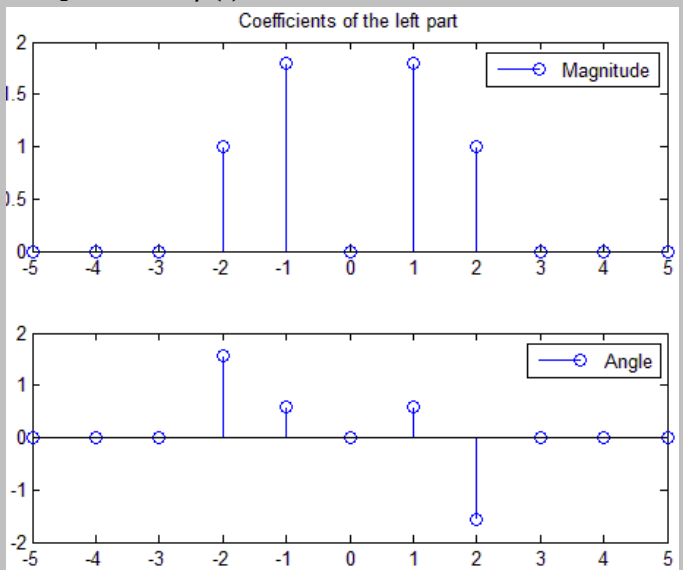
$$z_1 x(t) + z_2 y(t) \leftrightarrow z_1 a_k + z_2 b_k$$

To verify the linearity property, we consider the periodic signals $x(t) = \cos(t)$, $y(t) = \sin(2t)$ and the scalars $z_1 = 3 + 2i$ and $z_2 = 2$.

Commands	Results/Comments

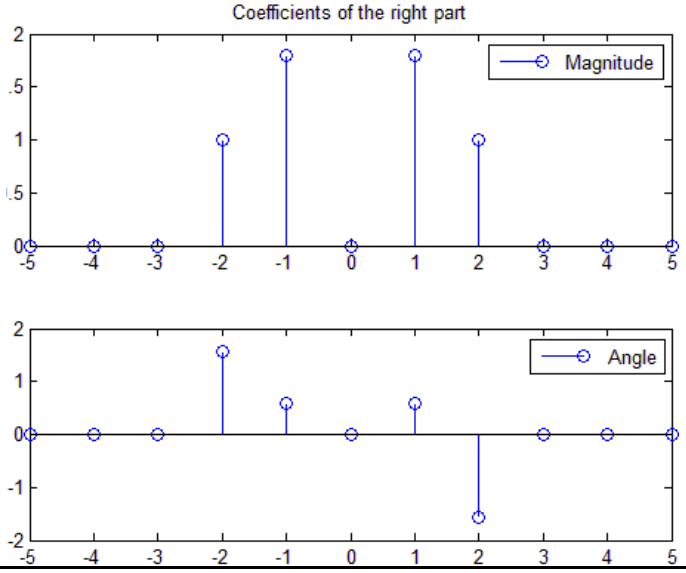
```
t0=0;
T=2*pi;
w=2*pi/T;
syms t
z1=3+2i; z2=2;
x=cos(t); y=sin(2*t);
f=z1*x+z2*y;
k=-5:5;
left=(1/T)*int(f*exp(-j*k*w*t),t,t0,t0+T);
left=eval(left);
subplot(211);
stem(k,abs(left));
legend('Magnitude');
title('Coefficients of the left part');
subplot(212);
stem(k,angle(left));
legend('Angle');
```

First we determine the complex exponential Fourier series coefficients of the left part; that is, we compute the coefficients of the signal $f(t) = z_1 x(t) + z_2 y(t)$. The period of $f(t)$ is $T = 2\pi$




```
a=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);
b=(1/T)*int(y*exp(-j*k*w*t),t,t0,t0+T);
right=z1*a+z2*b;
subplot(211);
right=eval(right);
stem(k,abs(right));
legend('Magnitude');
title('Coefficients of the right part');
subplot(212);
stem(k,angle(right));
legend('Angle');
```

In order to derive the right part of the linearity equation, first coefficients are computed and formulate the right part of the linearity equation.



The two graphs are identical; thus the linearity property is verified.

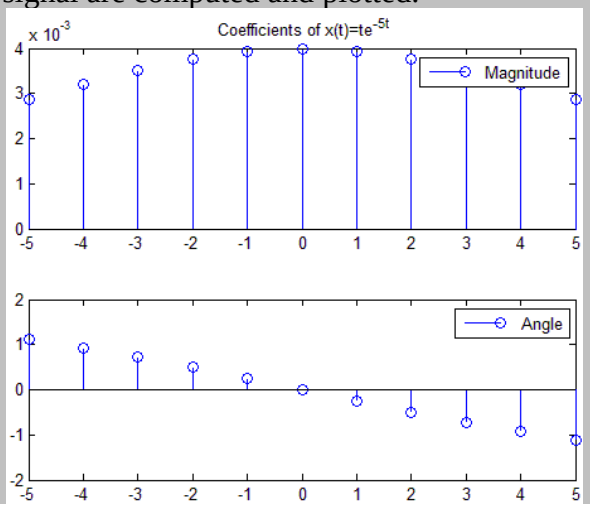
10.2.2 Time Shifting

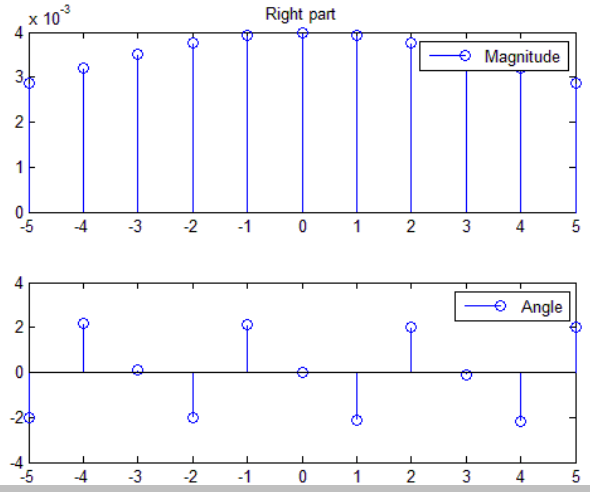
A shift in time of the periodic signal results on a phase change of the Fourier series coefficients. So, if $x(t) \rightarrow a_k$, the exact relationship is

$$x(t - t_1) \leftrightarrow e^{-jk\Omega_0 t_1} a_k$$

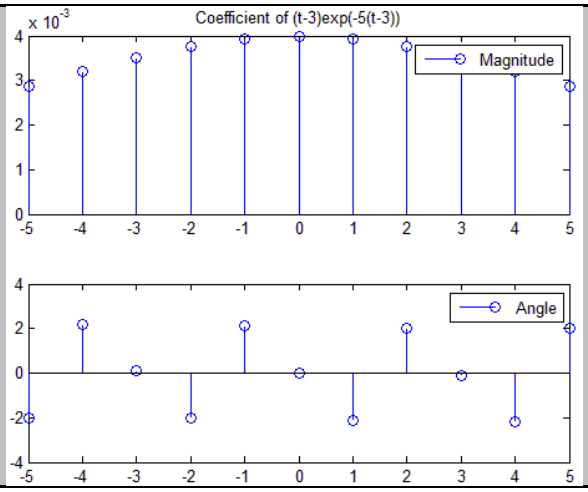
In order to verify time shifting property, we consider the periodic signal that in one period is given by $x(t)=te^{-5t}, 0\le t\le 10$. Moreover, we set $t_1=3$. Consequently the signal $x(t-t_1)$ is given by $x(t-t_1)=x(t-3)=(t-3)e^{-5(t-3)}$.

Commands	Results/Comments

t0=0;
T=10;
w=2*pi/T;
syms t
x=t*exp(-5*t)
k=-5:5;
a=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);
a1=eval(a);
subplot(211);
stem(k,abs(a1));
title(' Coefficients of x(t)=te^-^5^t');
legend('Magnitude');
subplot(212);
stem(k,angle(a1));
legend('Angle');
	First the Fourier series coefficients for the given signal are computed and plotted.
	 

t1=3;
right= exp(-j*k*w*t1).*a;
right =eval(right);
subplot(211);
stem(k,abs(right));
legend('Magnitude');
title('Right part');
subplot(212);
stem(k,angle(right));
legend('Angle');
	Next, the right part of the time shifting equation is computed.
	 

x=(t-t1).*exp(-5*(t-t1));
a=(1/T)*int(x*exp(-j*k*w*t),t,t0+t1,t0+T+t1);
coe=eval(a);
subplot(211);
stem(k,abs(coe));
legend('Magnitude');
title(' Coefficient of (t-3)exp(-5(t-3)) ');
subplot(212);
stem(k,angle(coe));
legend('Angle');
	Finally, the time shifted version of $x(t)$ is defined, i.e., $y(t)=(t-3)e^{-5(t-3)}$, and corresponding Fourier series coefficients are computed.



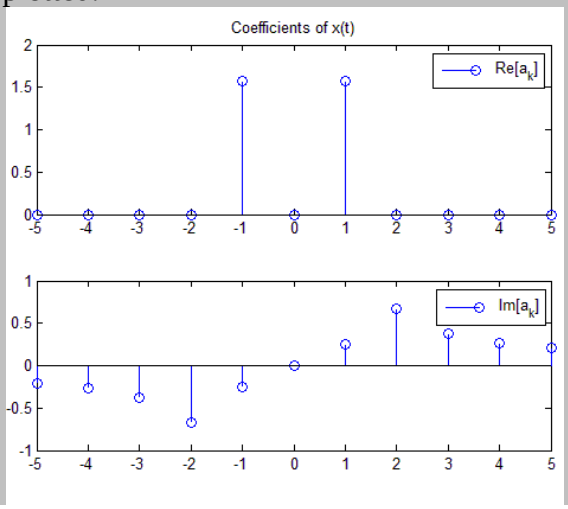
The two last graphs are identical; hence, the time shift property is confirmed. Comparing the two last graphs with the first one, we notice that indeed the magnitude does not change, but the phase is different.

10.2.3 Time Reversal

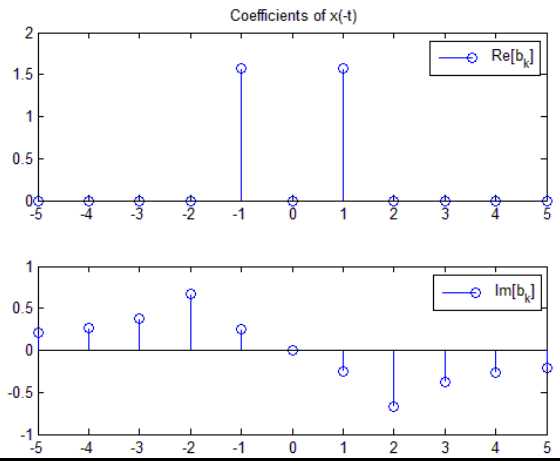
The Fourier series coefficients of the reflected version of a signal $x(t)$ are also a reflection of the coefficients of $x(t)$. So, if $x(t) \rightarrow a_k$, the mathematical expression is

$$x(-t) \leftrightarrow a_{-k}$$

In order to validate the time reversal property, we consider the periodic signal that in one period is given by $x(t) = t\cos(t), 0 \le t \le 2\pi$.

Commands	Results/Comments
t0=0; **T=2*pi;** **w=2*pi/T;** **syms t** **x=t*cos(t) ;** **k=-5:5;** **a=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);** **a1=eval(a);** **subplot(211);** **stem(k,real(a1));** **legend('Re[a_k]');** **title('Coefficients of x(t)');** **subplot(212);** **stem(k,imag (a1));** **legend('Im[a_k]');**	Fourier series coefficients a_k are computed and plotted. 
x_=-t*cos(-t) ; **b=(1/T)*int(x_*exp(-j*k*w*t),t,t0-T,t0);** **b1=eval(b)** **subplot(211);** **stem(k,real(b1));** **legend('Re[b_k]');**	Next the coefficients b_k are computed for the time reversed version $x(-t)$, and we notice that $b_k = a_{-k}$. Hence the time reversal property is confirmed.

```
title(' Coefficients of x(-t)');
subplot(212);
stem(k,imag (b1));
legend('Im[b_k]');
```

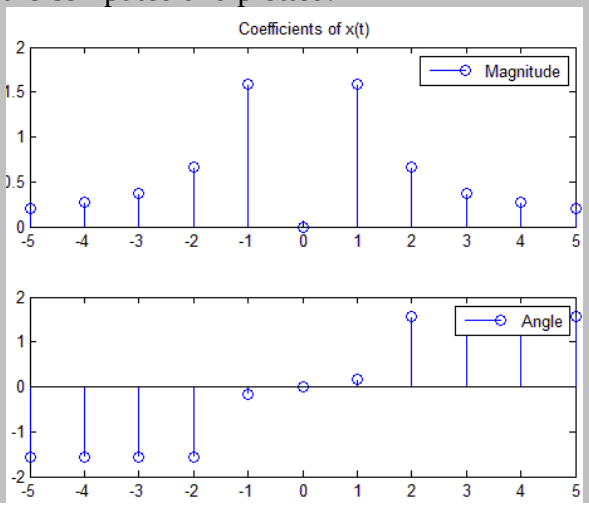


10.2.4 Time Scaling

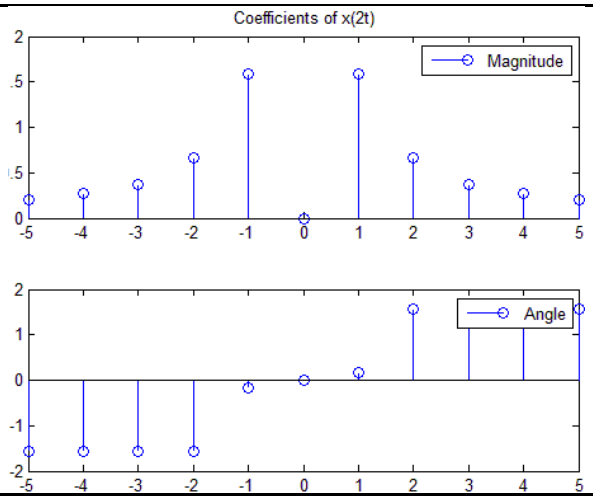
The Fourier series coefficients of a time scaled version $x(\lambda t)$ and $x(t)$ do not change. On the other hand, the fundamental period of the time scaled version becomes T/λ, and the fundamental frequency becomes $\lambda\Omega_0$. The mathematical expression is

$$x(\lambda t) \leftrightarrow a_k$$

The time scaling property is confirmed by using the periodic signal that in one period is given by $x(t) = t\cos(t), 0 \le t \le 2\pi$.

Commands	Results/Comments
syms t **t0=0;** **T=2*pi;** **w=2*pi/T;** **x=t*cos(t) ;** **k=-5:5;** **a=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);** **a1=eval(a)** **subplot(211);** **stem(k,abs(a1));** **legend('Magnitude');** **title(' Coefficients of x(t)');** **subplot(212);** **stem(k,angle(a1));** **legend('Angle');**	First the Fourier series exponential components $a_k, -5 \le k \le 5$ for the signal $x(t) = t\cos(t), 0 \le t \le 2\pi$ are computed and plotted. 
lamda=2; **T=T/ lamda ;** **w=2*pi/T;** **x= lamda *t*cos(lamda *t) ;** **k=-5:5;** **a=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);** **a1=eval(a)** **subplot(211);**	Next the coefficients b_k of the time scaled signal $x(2t) = 2t\cos(2t), 0 \le t \le \pi$, and it is seen that $a_k = b_k$. Hence the time scaling property is confirmed.

```
stem(k,abs(a1));
legend('Magnitude');
title(' Coefficients of x(2t)');
subplot(212);
stem(k,angle(a1));
legend('Angle');
```

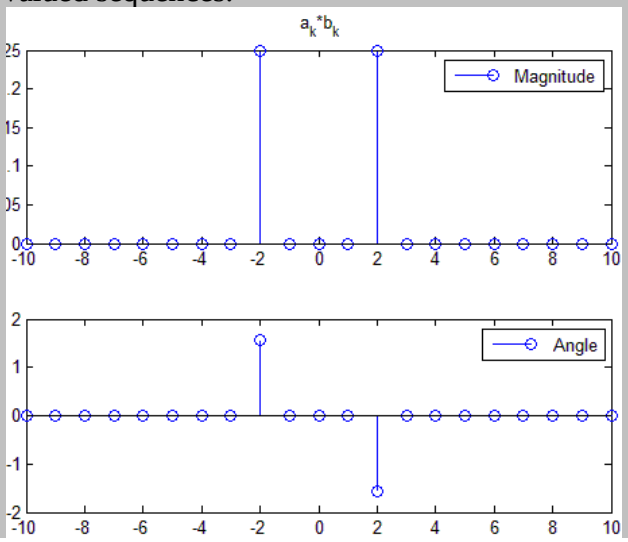


10.2.5 Signal Multiplication

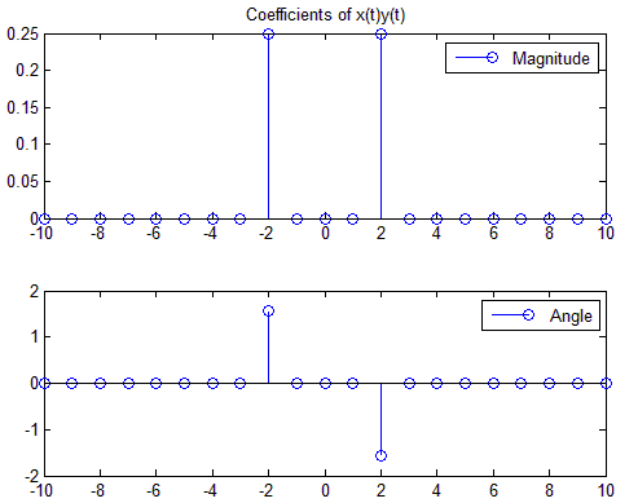
The Fourier series coefficient of the product of two signals equals the convolution of the Fourier series coefficients of each signal. Suppose that $x(t) \leftrightarrow a_k$ and $y(t) \leftrightarrow b_k$, we have 10. 6 as

$$x(t)y(t) \leftrightarrow a_k * b_k$$

Where * denotes discrete time convolution. To verify property 10.6, we consider the signals $x(t) = \cos(t)$ and $y(t) = \sin(t)$.

Commands	Results/Comments
syms t **t0=0;** **T=2*pi;** **w=2*pi/T;** **x=cos(t) ;** **k=-5:5;** **a=(1/T)*int(x*exp(-j*k*w*t),t,t0,t0+T);** **a1=eval(a);** **y=sin(t);** **b=(1/T)*int(y*exp(-j*k*w*t),t,t0,t0+T);** **b1=eval(b);** **left=conv(a1,b1);** **subplot(211);** **stem(-10:10,abs(left));** **legend('Magnitude');** **title(' a_k*b_k');** **subplot(212);** **stem(-10:10,angle(left));** **legend('Angle');**	First, the exponential Fourier series coefficients $a_k, -5 \le k \le 5$ and $b_k, -5 \le k \le 5$ and their convolution is computed. Notice that the convolution $d_k = a_k * b_k$ is implemented between two complex valued sequences. 
z=x*y; **k=-10:10;** **c=(1/T)*int(z*exp(-j*k*w*t),t,t0,t0+T);** **c1=eval(c)**	Next, the Fourier series coefficients c_k are computed for the signal $z(t) = \cos(t)\sin(t), -10 \le k \le 10$.

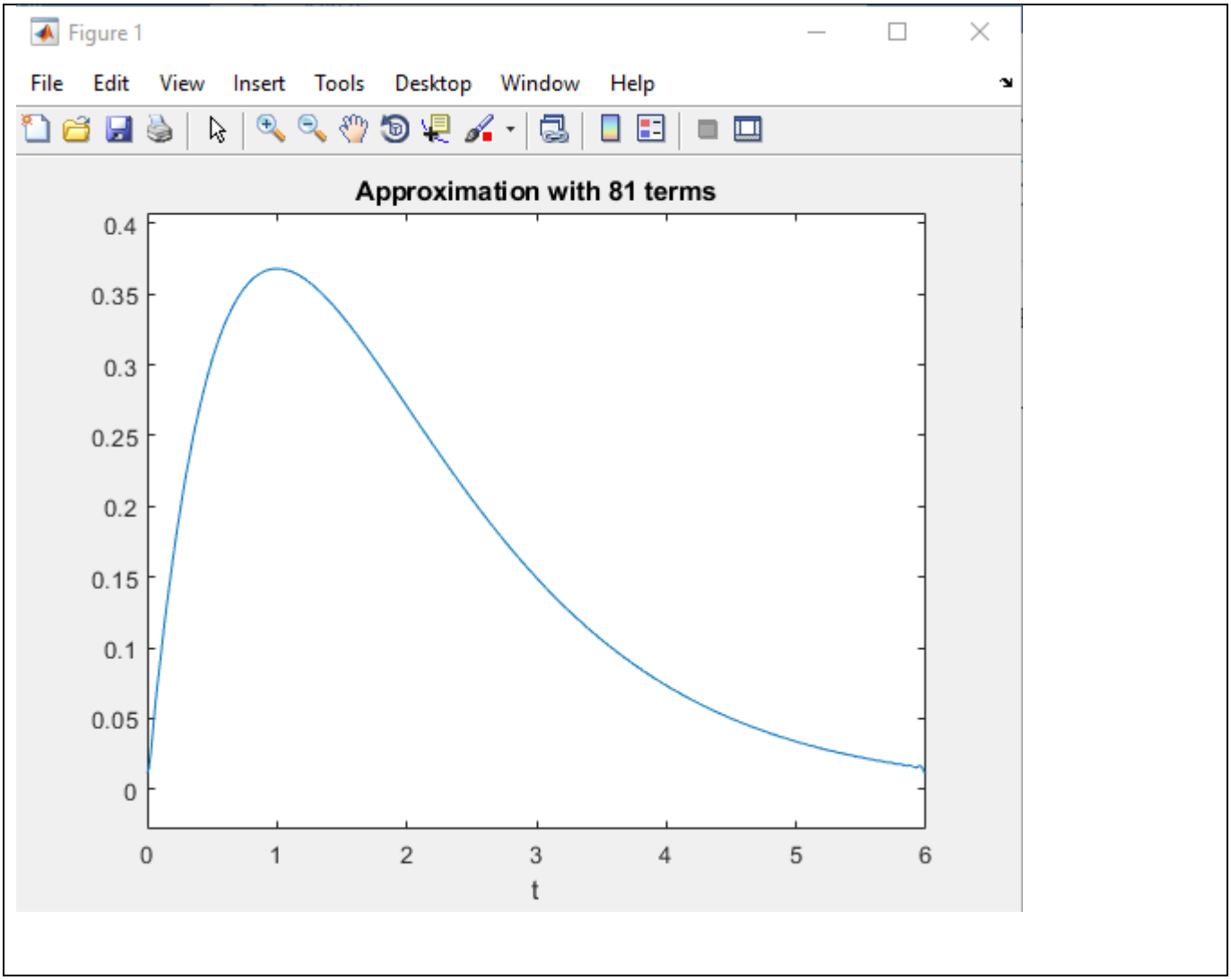
```
subplot(211);
stem(k,abs(c1));
legend('Magnitude');
title(' Coefficients of x(t)y(t)');
subplot(212);
stem(k,angle(c1));
legend('Angle');
```



In-Lab Tasks

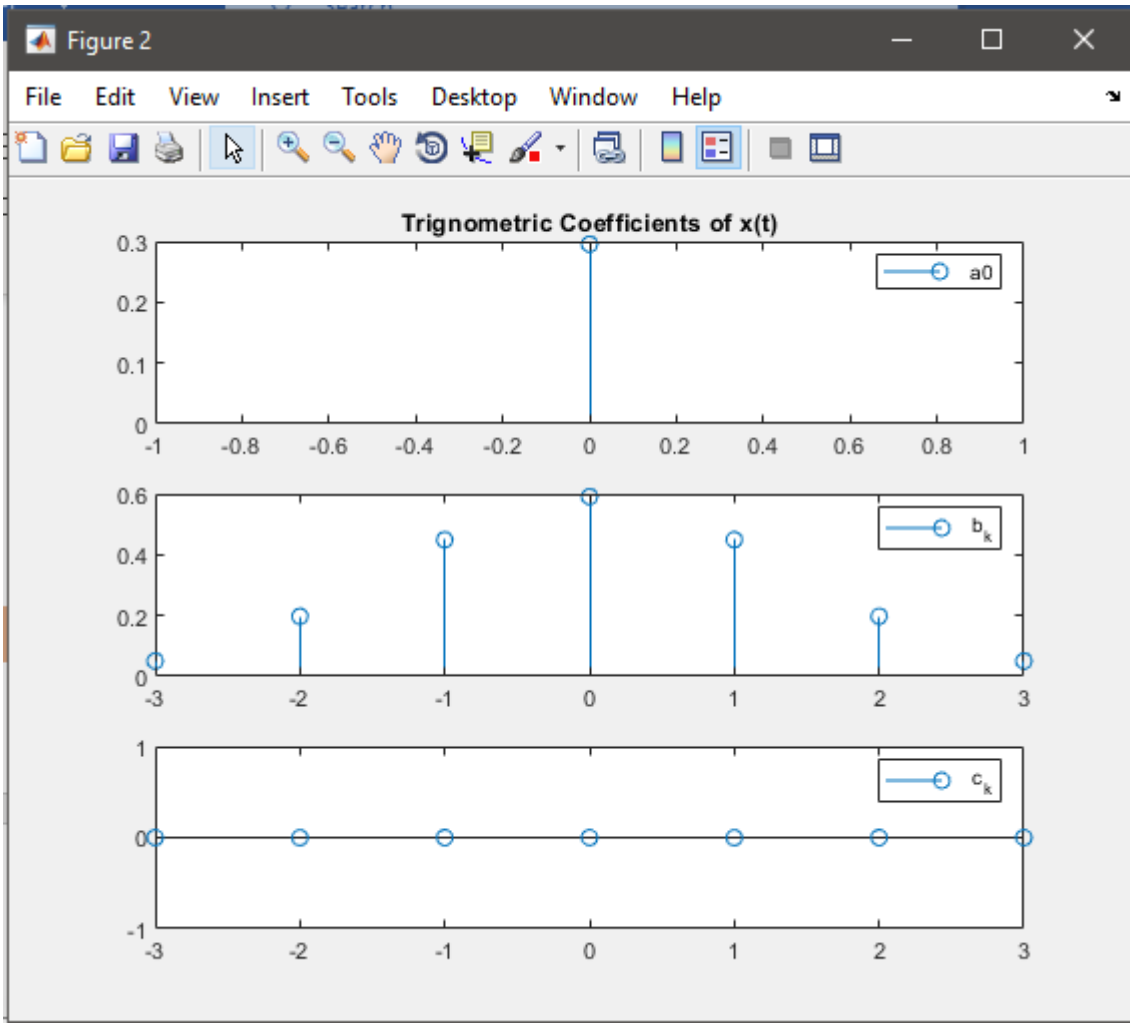
Task 01: The periodic signal $x(t)$ is defined in one period as $x(t) = te^{-t}, 0 \le t \le 6$. Plot approximate signal using 81 terms of trigonometric form of Fourier series.

```
T = 6;                  %Time Period
t0 = 0;
w = 2*pi/T;             %Angular Frequency

syms t                  %t as symbol declaration
x = t.*exp(-t);

a0 = (1/T)*int(x,t,t0,t0+T);
for n = 1:80
b(n) = (2/T)*int(x*cos(n*w*t),t,t0,t0+T);
end
for n = 1:80
c(n) = (2/T)*int(x*sin(n*w*t),t,t0,t0+T);
end
k = 1:80;
xx = a0+sum(b.*cos(k*w*t))+sum(c.*sin(k*w*t))
ezplot(xx, [t0 t0+T]);
title('Approximation with 81 terms')
```



Task 02: Plot the coefficients of the trigonometric Fourier series for the periodic signal that in one period is defined by $x(t) = e^{-t^2}, -3 \le t \le 3$**.**

```
T = 6;                  %Time Period
t0 = -3;
w = 2*pi/T;             %Angular Frequency

syms t                  %t as symbol declaration
x = exp(-t.^2);
k = -3:3;
a0 = (1/T)*int(x,t,t0,t0+T);
b = (2/T)*int(x*cos(k*w*t),t,t0,t0+T);
c = (2/T)*int(x*sin(k*w*t),t,t0,t0+T);

figure
subplot(3,1,1)
stem(0,a0),legend('a0'),title('Trignometric Coefficients of x(t)');
subplot(3,1,2)
stem(k,b),legend('b_k');
subplot(3,1,3)
stem(k,c),legend('c_k');
```

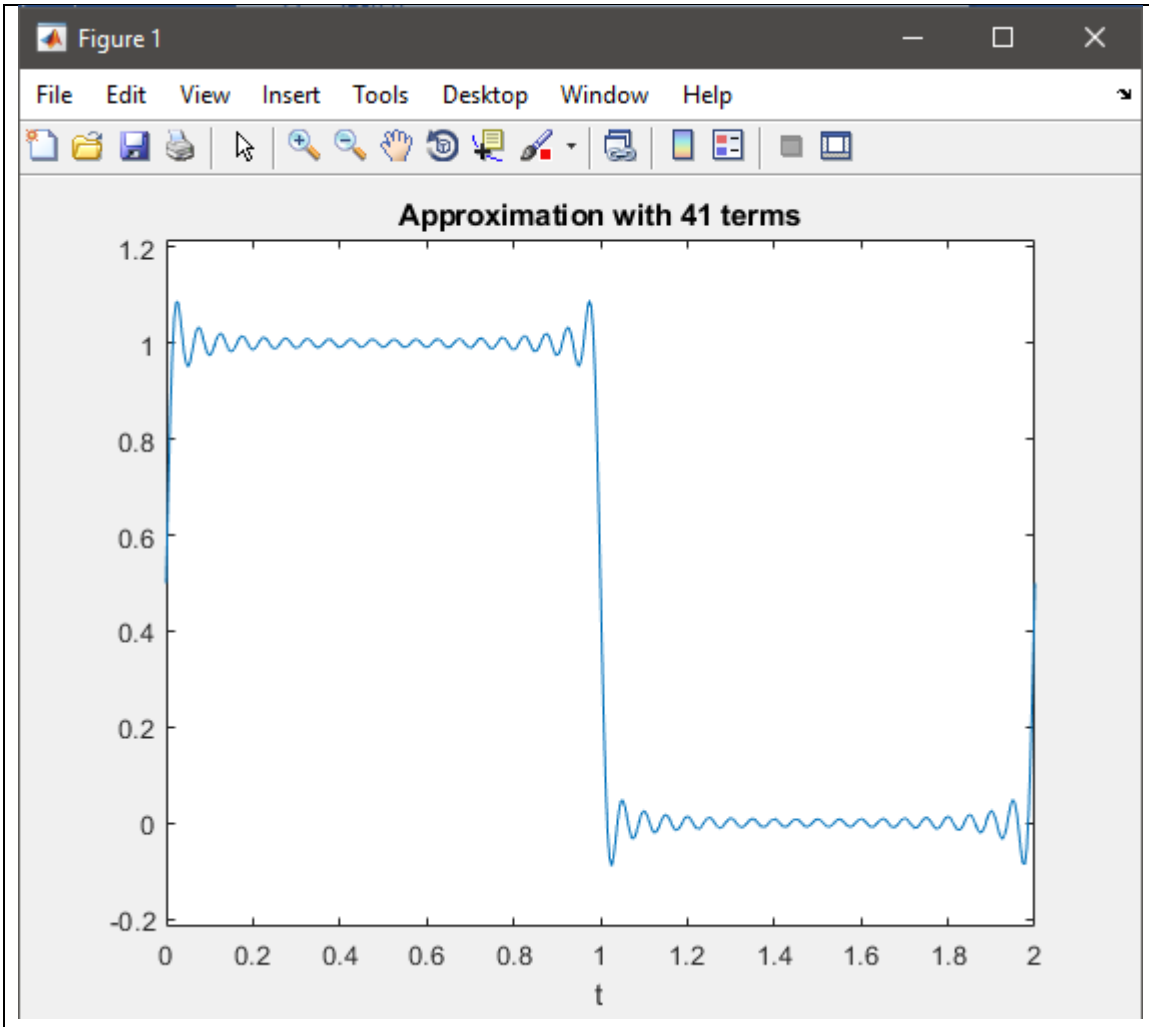


Task 03: The periodic signal $x(t)$ in a period is given by

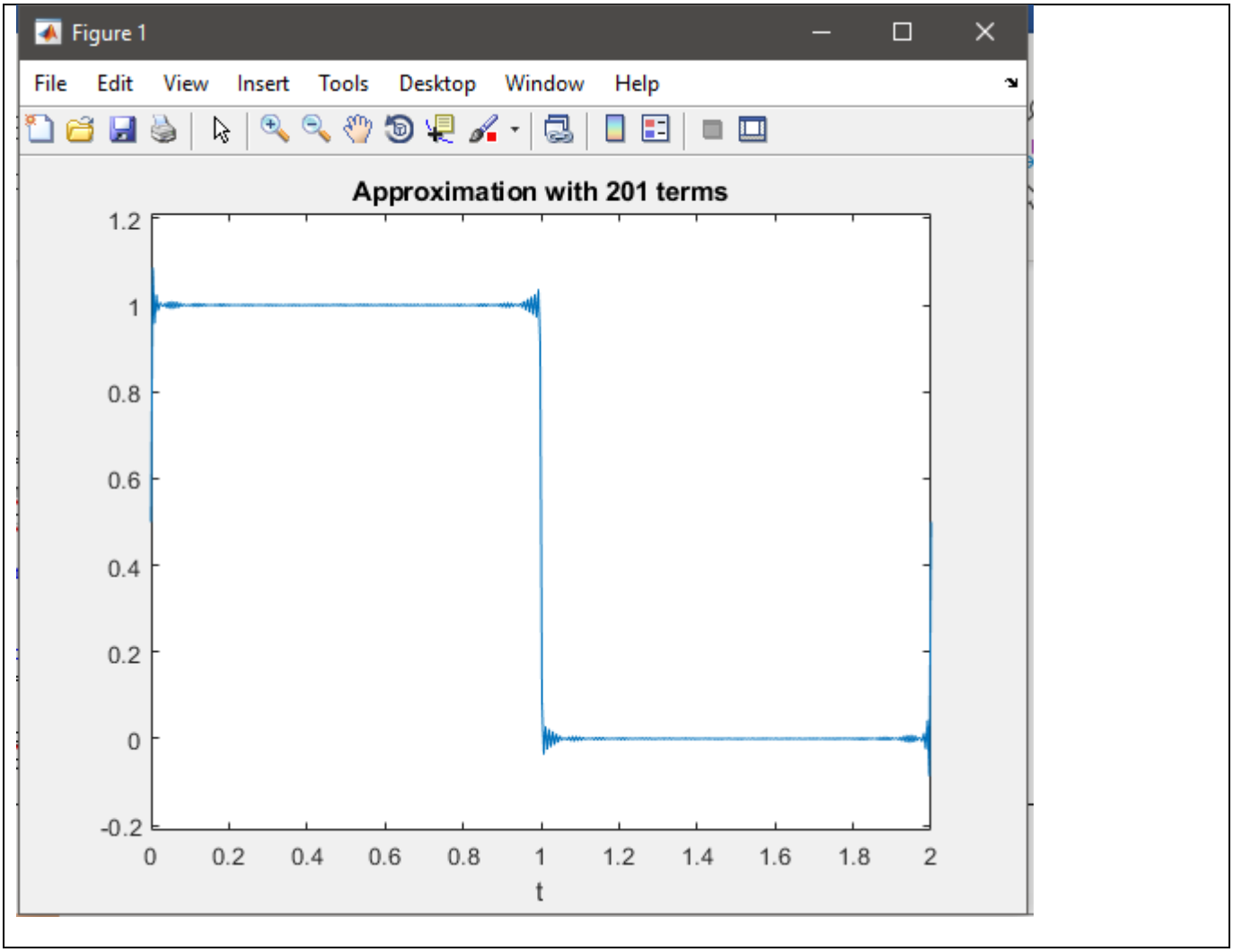
$$x(t)=\begin{cases}1, & 0\le t\le 1\\ 0, & 1\le t\le 2\end{cases}$$

Plot in one period the approximate signals using 41 and 201 term of the trigonometric Fourier series. Furthermore, each time plot the complex exponential coefficients.

```
t0 = 0;
T = 2;
w = 2*pi/T;
syms t
x=heaviside(t)-heaviside(t-1);

a0 = (1/T)*int(x,t,t0,t0+T);
for n = 1:40       %Approximation using 41 terms
    b(n) = (2/T)*int(x*cos(n*w*t),t,t0,t0+T);
end
for n = 1:40       %Approximation using 41 terms
    c(n) = (2/T)*int(x*sin(n*w*t),t,t0,t0+T);
end

k = 1:40;          %Approximation using 41 terms
xx = a0 + sum(b.*cos(k*w*t)) + sum(c.*sin(k*w*t));
ezplot(xx, [t0 t0+T]);
title('Approximation with 41 terms')
```




```
t0 = 0;
T = 2;
w = 2*pi/T;
syms t
x=heaviside(t)-heaviside(t-1);
a0 = (1/T)*int(x,t,t0,t0+T);
for n = 1:200
    b(n) = (2/T)*int(x*cos(n*w*t),t,t0,t0+T);
    c(n) = (2/T)*int(x*sin(n*w*t),t,t0,t0+T);
end
k = 1:200;
xx = a0 + sum(b.*cos(k*w*t)) + sum(c.*sin(k*w*t));
ezplot(xx, [t0 t0+T]);
title('Approximation with 201 terms')
```

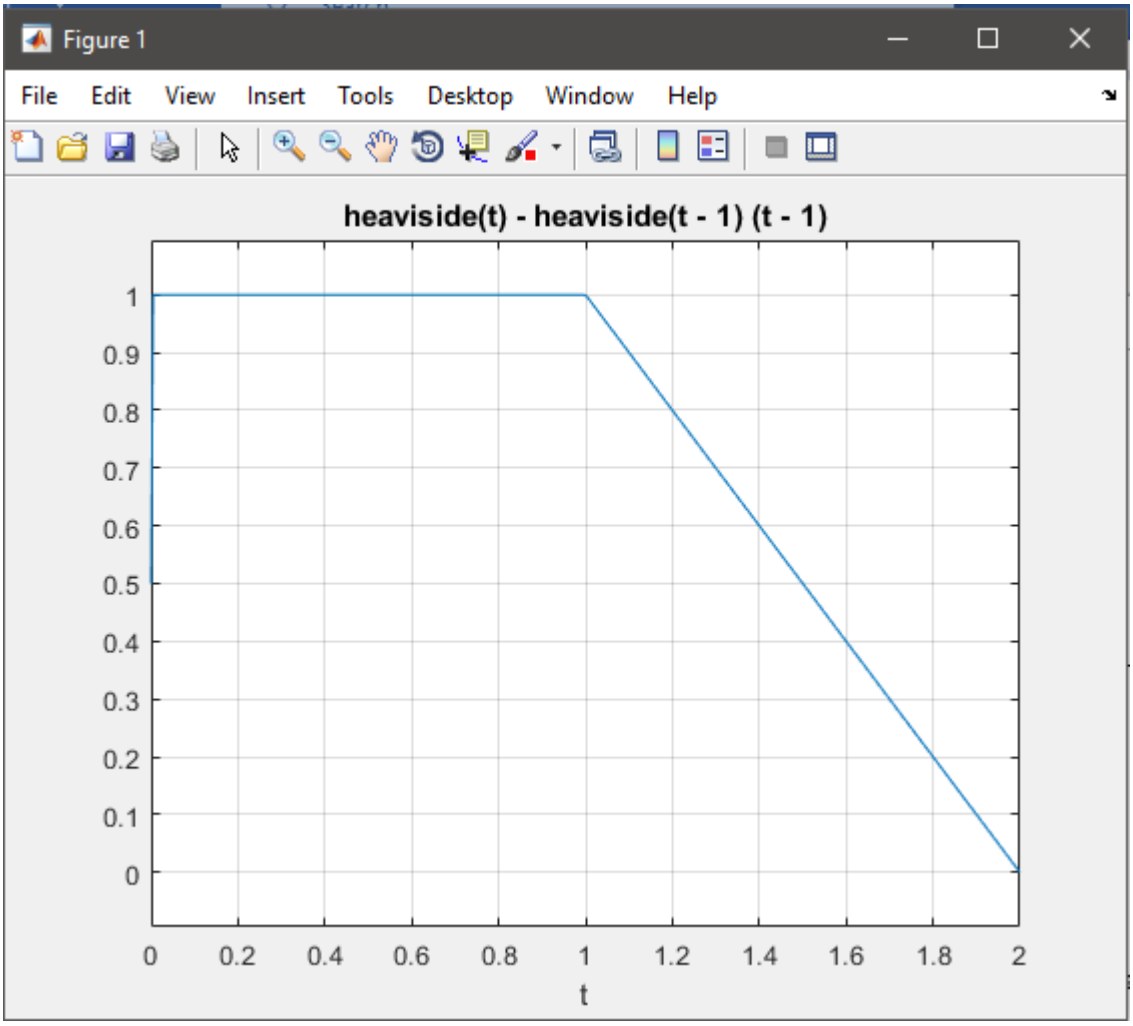


Task 04: The periodic signal $x(t)$ in a period is given by

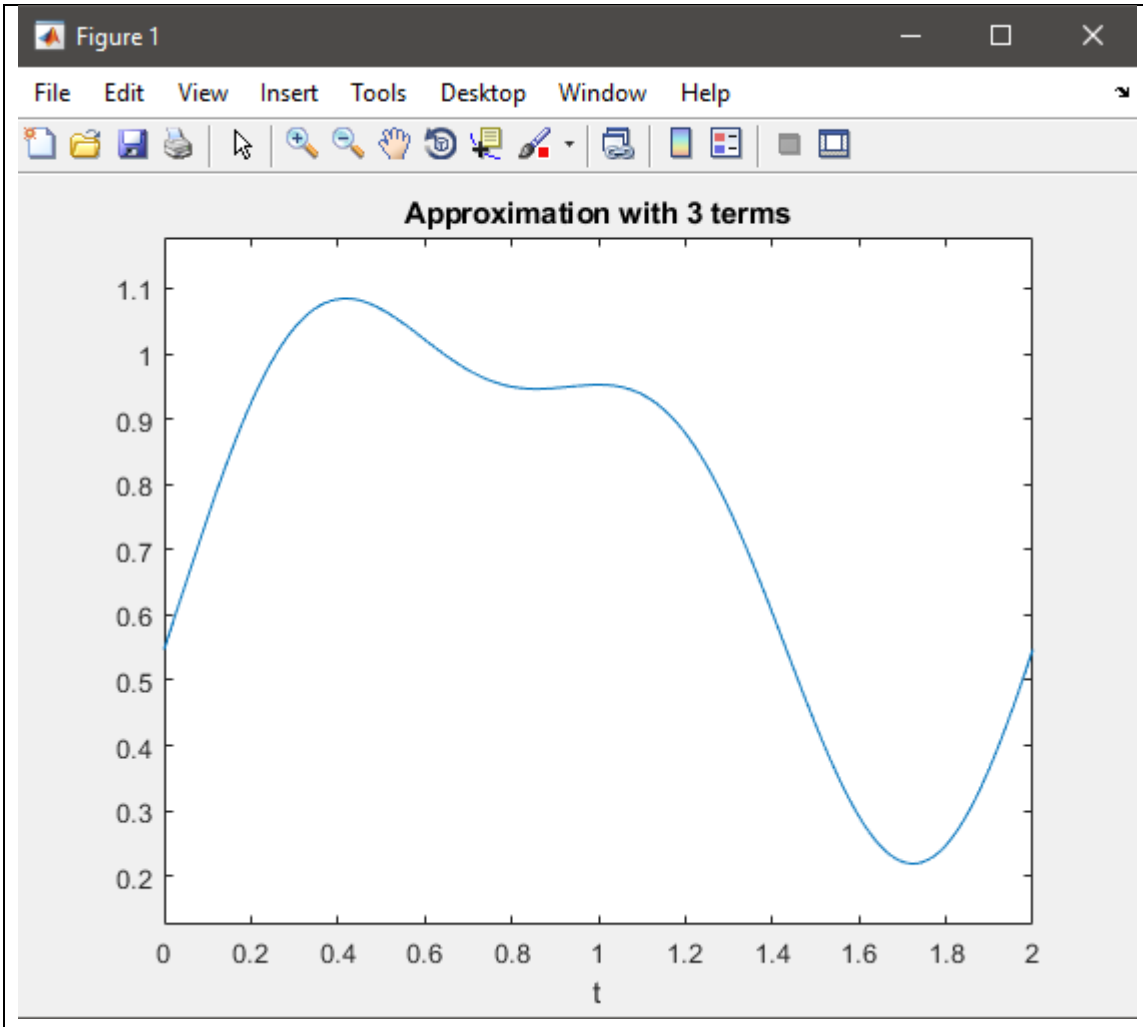
$$x(t)=\begin{cases} 1, & 0\le t\le 1 \\ 2-t, & 1\le t\le 2 \end{cases}$$

Calculate the approximation percentage when the signal $x(t)$ is approximated by 3, 5, 7, and 17 terms of the trigonometric Fourier series. Furthermore, plot the signal in each case.

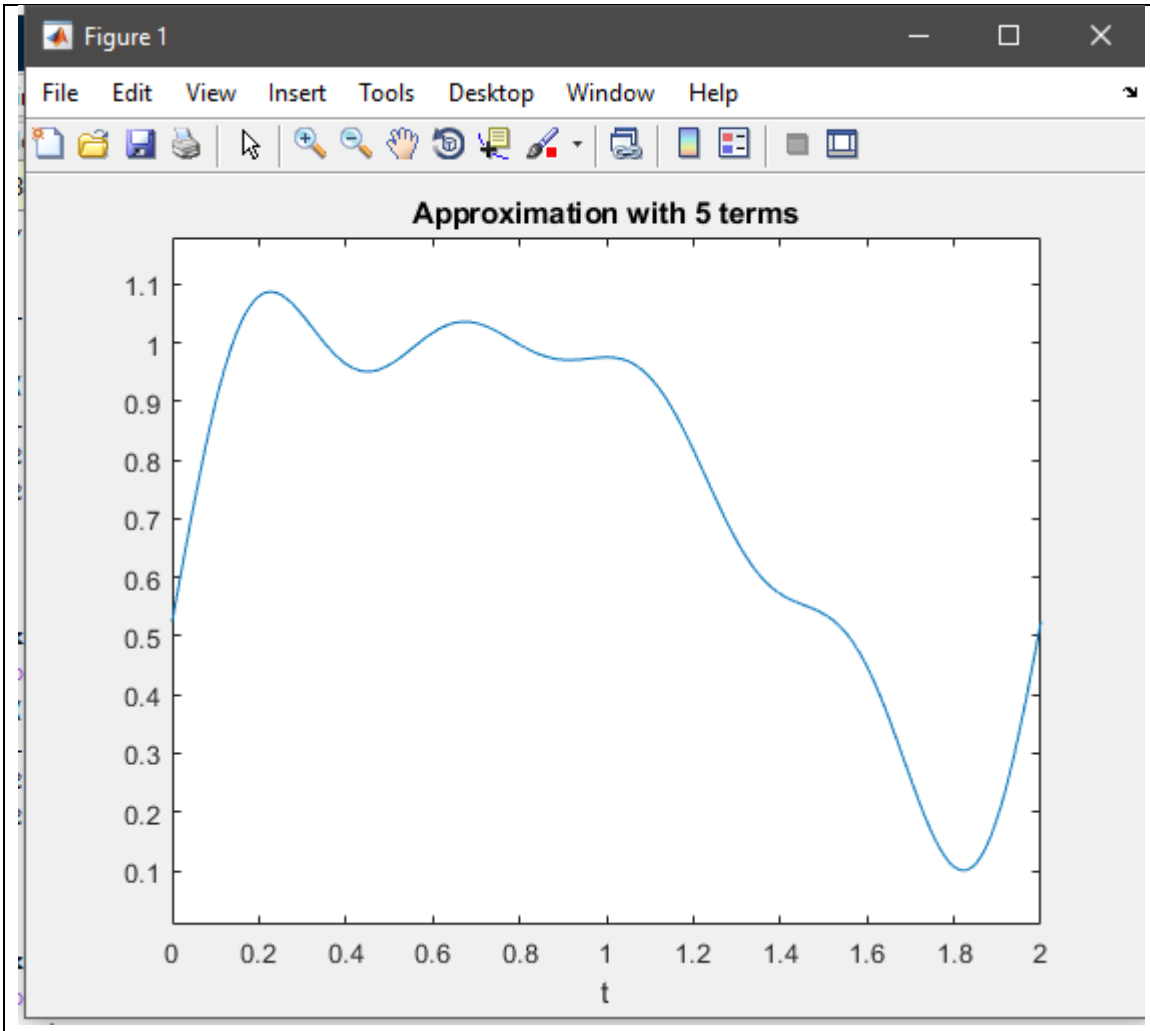
```
T = 2;
t0 = 0;
w = 2*pi/T;
syms t
x = heaviside(t)+((heaviside(t-1)).*(1-t));
ezplot(x,[t0 t0+T]),grid on
```



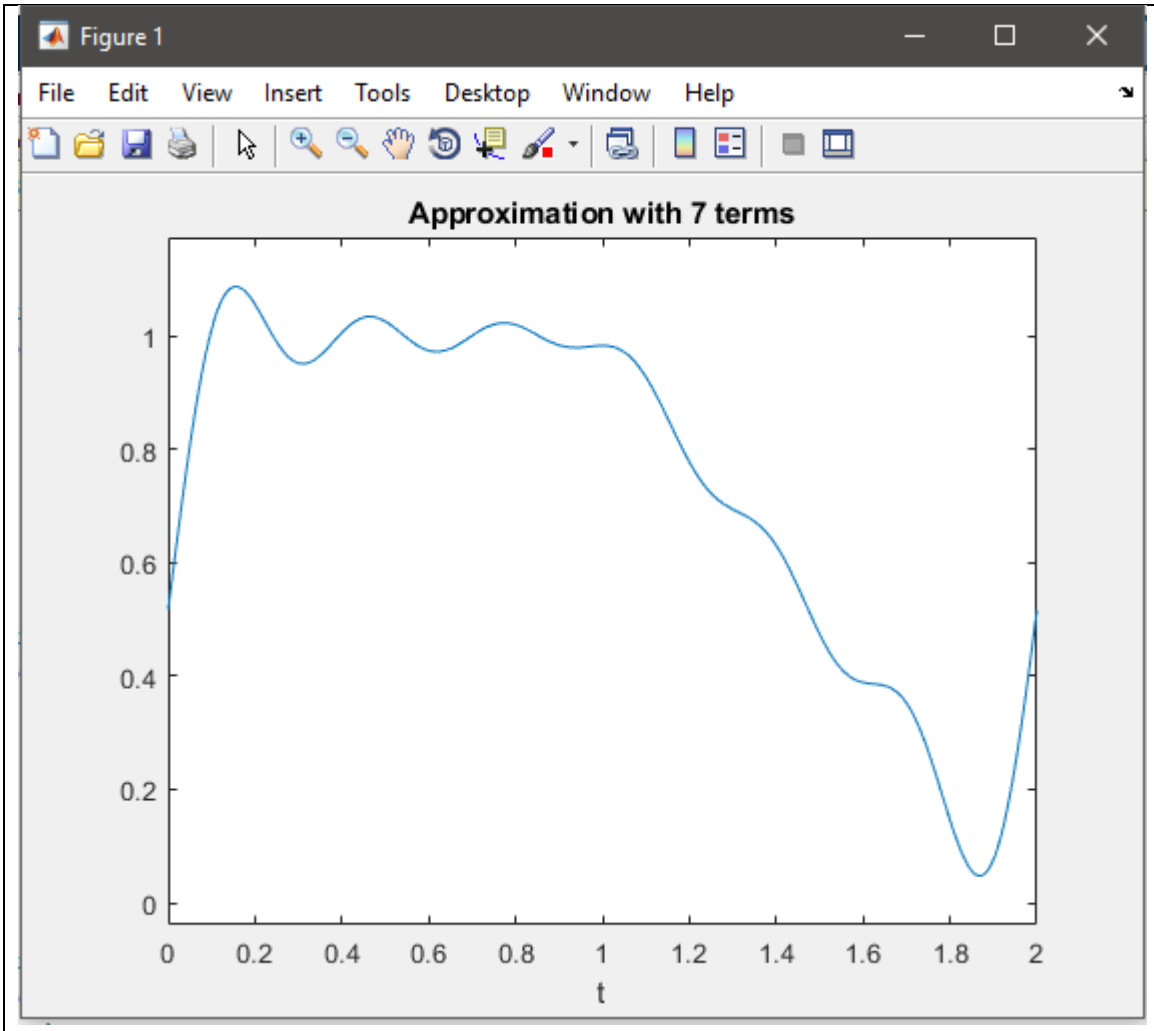

```
a0 = (1/T)*int(x,t,t0,t0+T);
for n = 1:2
b(n) = (2/T)*int(x*cos(n*w*t),t,t0,t0+T);
c(n) = (2/T)*int(x*sin(n*w*t),t,t0,t0+T);
end
k = 1:2;
xx1 = a0 + sum(b.*cos(k*w*t)) + sum(c.*sin(k*w*t));
ezplot(xx1, [t0 t0+T]);
title('Approximation with 3 terms')
```




```
a0 = (1/T)*int(x,t,t0,t0+T);
for n = 1:4
b(n) = (2/T)*int(x*cos(n*w*t),t,t0,t0+T);
c(n) = (2/T)*int(x*sin(n*w*t),t,t0,t0+T);
end
k = 1:4;
xx1 = a0 + sum(b.*cos(k*w*t)) + sum(c.*sin(k*w*t));
ezplot(xx1, [t0 t0+T]);
title('Approximation with 5 terms')
```




```
a0 = (1/T)*int(x,t,t0,t0+T);
for n = 1:6
b(n) = (2/T)*int(x*cos(n*w*t),t,t0,t0+T);
c(n) = (2/T)*int(x*sin(n*w*t),t,t0,t0+T);
end
k = 1:6;
xx1 = a0 + sum(b.*cos(k*w*t)) + sum(c.*sin(k*w*t));
ezplot(xx1, [t0 t0+T]);
title('Approximation with 7 terms')
```




```
a0 = (1/T)*int(x,t,t0,t0+T);
for n = 1:16
b(n) = (2/T)*int(x*cos(n*w*t),t,t0,t0+T);
c(n) = (2/T)*int(x*sin(n*w*t),t,t0,t0+T);
end
k = 1:16;
xx1 = a0 + sum(b.*cos(k*w*t)) + sum(c.*sin(k*w*t));
ezplot(xx1, [t0 t0+T]);
title('Approximation with 17 terms')
```

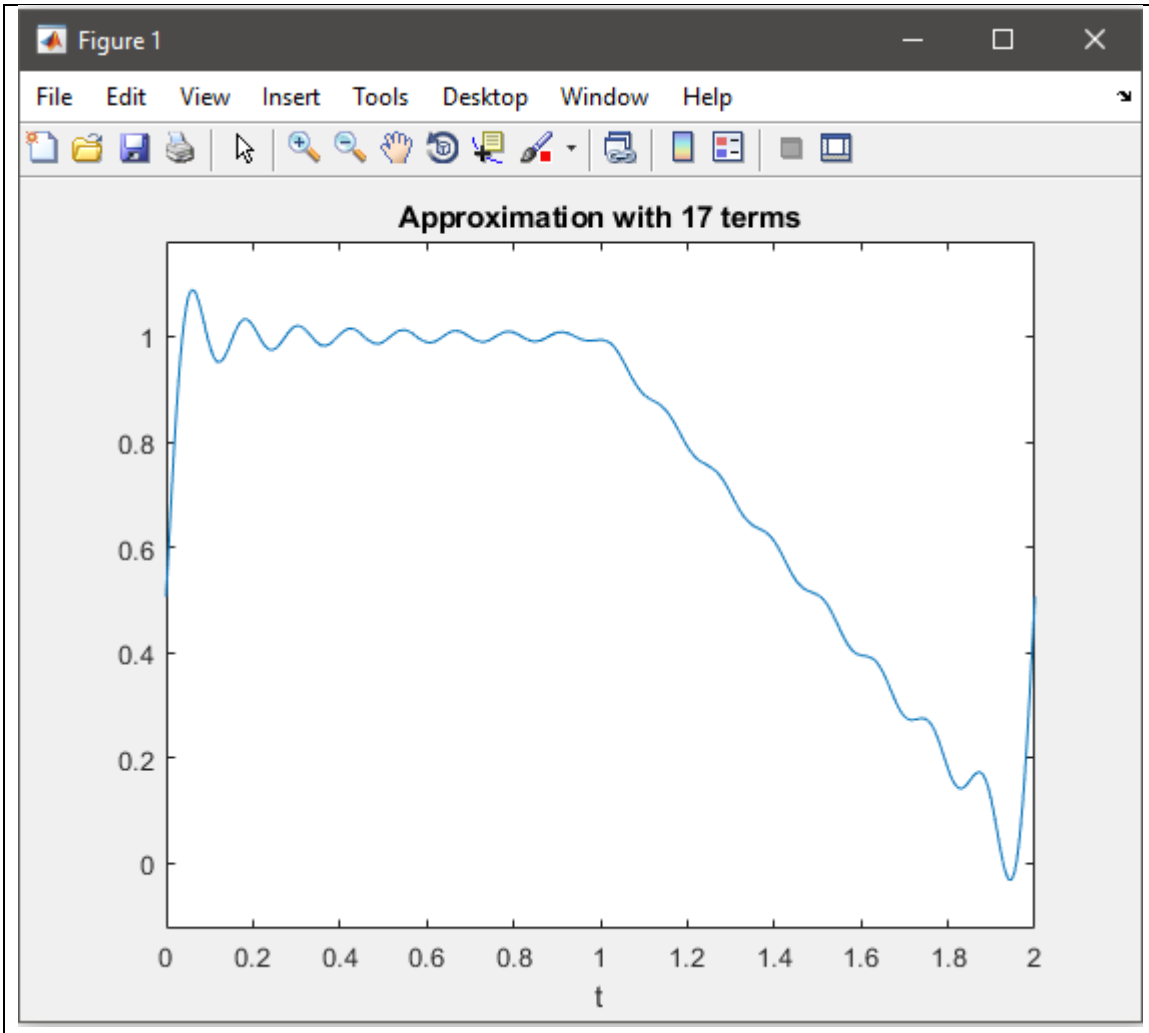


Figure 1
File Edit View Insert Tools Desktop Window Help
Approximation with 17 terms
1
0.8
0.6
0.4
0.2
0
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2
t

Post-Lab Task

Critical Analysis / Conclusion

In this lab, we learnt how to plot signals with approximations and observed the effect with increased number of terms and how to find coefficients “a_k” trigonometric Fourier series in MATLAB. Moreover, we plotted and observed the coefficients and angle of trigonometric exponential Fourier series.

Lab Assessment		
Pre-Lab	**/1**	**/10**
In-Lab	**/5**	
Critical Analysis	**/4**	
Instructor Signature and Comments		

LAB # 11

Continuous Time Fourier Transform (CTFT)

Lab 11- Continuous Time Fourier Transform (CTFT)

Pre Lab

Fourier transform is used to transform a time domain signal into frequency domain. As some times frequency domain reveals more information as compared to time domain. In this lab, the Fourier transform for continuous-time signals will be discussed which is known as continuous-time Fourier transform (CTFT). By applying, Fourier transform to a continuous time signal $x(t)$, we obtain a representation of the signal at the cyclic frequency domain Ω or equivalently at the frequency domain f.

The Fourier transform is denoted by the symbol $F\{.\}$; that is, one can write (11.1) as

$$X(\Omega) = F\{x(t)\} \tag{11.1}$$

In other words, the Fourier transform of a signal $x(t)$ is a signal $X(\Omega)$. An alternative way of writing eq. (11.1) is given in eq. (11.2) and eq. (11.3) shows mathematical from of Fourier transform.

$$x(t) \xrightarrow{F} X(\Omega) \tag{11.2}$$

$$X(\Omega) = \int_{-\infty}^{\infty} x(t)\, e^{-j\Omega t}\, dt, \tag{11.3}$$

From eq. (11.3) it is clear that $X(\Omega)$ is complex function of Ω. Where Ω =2πf substituting in eq. (11.3) we get

$$X(f) = \int_{-\infty}^{\infty} x(t)\, e^{-j2\pi f t}\, dt, \tag{11.4}$$

In order to return from the frequency domain back to the time domain the *inverse* Fourier transform is implemented. The inverse Fourier transform is denoted by the symbol; i.e. $F^{-1}\{.\}$

$$x(t) = F^{-1}\{X(\Omega)\} \tag{11.5}$$

or alternatively,

$$X(\Omega) \xrightarrow{F^{-1}} x(t) \tag{11.6}$$

Mathematically, inverse Fourier transform is given by

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\Omega)\, e^{j\Omega t}\, dt, \tag{11.7}$$

or

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\Omega)\, e^{j2\pi f t}\, dt, \tag{11.8}$$

The cyclic frequency Ω is measured in rad/s, while the frequency f is measured in Hertz. The Fourier transform of a signal is called (frequency) *spectrum.* MATALB command for Fourier transform is "fourier" and for inverse Fourier transform it is "ifourier".

In-Lab Tasks

Task-1

Compute the Fourier transform of $x(t) = e^{-t^2}$. MATLAB code is given in following, run this code and compare your output using eq. (11.3). Write your code and results in following.

```
syms w t
x = exp(-t.^2);
X=foprier(x,w);

OUTPUT
X =
pi^(1/2)*exp(-w^2/4)
```

Task-2

Compute the inverse Fourier transform of X = exp(-1/4*w^2)*pi^(1/2) using the command for inverse Fourier transform and also verify your result using eq. (11.7). Give your results in following.

```
syms t w
X = exp(-1/4*w^2)*pi^(1/2);
ifoprier(X,w)

OUTPUT
ans =

(3991211251234741*exp(-w^2))/(2251799813685248*pi^(1/2))
```

Task-3

Compute the inverse Fourier transform of the function $X(\Omega) = 1/(1 + j\Omega)$ using command of Fourier and then take inverse of the resultant x(t) to produce again $X(\Omega)$.

Part 1)

```
syms t w
x = 1/1+(1i*w))
foprier(x,t)
OUTPUT
x =
```

```
1/(1 + w*-i)

ans =
   -pi*exp(t)*(sign(t) - 1)
```

Part 2)

```
syms f
w = 2*pi*f
syms t
x = -pi*exp(t)*(sign(t) - 1)
ifoprier(x,w)
```

OUTPUT

```
w =
2*pi*f
x =
-pi*exp(t)*(sign(t) - 1)
ans =
                                1/(pi*f*2i + 1)
```

Task-4

Let x(t) = 1, compute its Fourier transform to produce X(w) and then take inverse Fourier transform of X(w) to get back x(t), using commands of Fourier transform.

Part 1)

```
syms t w
x=1
X=foprier(x,w)
```

OUTPUT

```
x =
     1
X =
2*pi*dirac(w)
```

Part 2)

```
syms t w
x=2*pi*dirac(w)
x=ifoprier(x,t)
```

OUTPUT

```
x =
2*pi*dirac(w)
x =
    1
```

Task-5

Let x(t) = u(t), compute its Fourier transform, take inverse Fourier transform of the resultant signal to get back x(t).

Part 1)

```
t=0:1:5
syms w
x=t>=0
X=foprier(x,w)
```

OUTPUT

```
t =
0 1 2 3 4 5
x =
1×6 logical array
1 1 1 1 1 1
X =
[ 2*pi*dirac(w), 2*pi*dirac(w), 2*pi*dirac(w), 2*pi*dirac(w),
2*pi*dirac(w), 2*pi*dirac(w)]
```

Part 2)

```
syms t w
x=[ 2*pi*dirac(w), 2*pi*dirac(w), 2*pi*dirac(w), 2*pi*dirac(w),
2*pi*dirac(w), 2*pi*dirac(w)]
x=ifoprier(x,t)
```

OUTPUT

```
x =
[ 2*pi*dirac(w), 2*pi*dirac(w), 2*pi*dirac(w), 2*pi*dirac(w),
2*pi*dirac(w), 2*pi*dirac(w)]
x =
                              [ 1, 1, 1, 1, 1, 1]
```

Task-6

Let x(t) = δ(t), compute its Fourier transform, take inverse Fourier transform of the resultant signal and state whether it is possible to get back x(t) or not?

Part 1)

```
syms t
x=dirac(t)
X=foprier(x,w)
```

OUTPUT

```
x =
dirac(t)
X =
1
```

Part 2)

```
syms t
x=1
X=ifoprier(x,t)
```

OUTPUT

```
x =
1

X =
dirac(t)
```

Hence it is possible to get x(t) back

Task-7

Prove that $x(t) = \delta(t-2)$ and $X(\Omega) = e^{-j2\Omega}$, are Fourier transform pairs of each other.

Part 1)

```
syms t
x=dirac(t-2)
X=foprier(x,w)
```

OUTPUT

```
x =
dirac(t - 2)
X =
exp(-w*2i)
```

Part 2)

```
syms t w
x=exp(-w*2i)
X=ifoprier(x,t)
```

OUTPUT

```
x =
exp(-w*2i)
X =
```

```
dirac(t - 2)
```

Hence x(t) and X(Ω) are Fourier transform pairs of each other.

Task-8

Prove that $x(t) = u(t-2)$ and $X(\Omega) = \exp(-2 * j * w) * (pi * \delta(w) - j/w)$, are Fourier transform pairs of each other.

Part 1)

```
syms w t x
a = heaviside(x-2);
A=foprier(a,w)
```

OUTPUT

```
A =
exp(-w*2i)*(pi*dirac(w) - 1i/w)
```

Part 2)

```
syms w t x
A=exp(-2*j*w)*(pi*dirac(w)-j/w);
a=ifoprier(A,w)
```

OUTPUT

```
a =
(pi + pi*sign(w - 2))/(2*pi)
```

Hence x(t) and X(Ω) are Fourier transform pairs of each other.

Critical Analysis / Conclusion

In this lab we learnt how to transform a continuous time domain signal into continuous frequency domain using the concept of Fourier Transform. we also learned how to take inverse Fourier Transform using MATLAB.

Lab Assessment		
Pre-Lab	**/1**	**/10**
In-Lab	**/5**	
Critical Analysis	**/4**	
Instructor Signature and Comments		

LAB#12

Open ended Lab

Lab 12- Open ended lab

Objectives

Objective of this lab is to encourage the analytical and problem-solving skills in a methodical way through literature survey, design and conduct of experiments, analysis and interpretation of experimental data and synthesis of information to derive the valid conclusions.

Pre-Lab

12.1 The Laplace Transform:

The Laplace transform expresses a signal in the complex frequency domain s (or s-domain); that is, a signal is described by a function F(s). Laplace transform is defined by the symbol L{.}; that is, one can write

$$F(s) = L\{f(t)\} \tag{12.1}$$

In order words, the Laplace transform of a function f(t) is a function F(s). An alternative way of writing (equation 12.1) is

$$f(t) \xrightarrow{L} F(s) \tag{12.2}$$

There are two available forms of Laplace transform. The first is two-sided (or bilateral) Laplace transform where the Laplace transform F(s) of a function f(t) is given by

$$F(s) = L\{f(t)\} = \int_{-\infty}^{\infty} f(t)e^{-st}\,dt \tag{12.3}$$

The second form is one-sided (or unilateral) Laplace transform, which is described by the relationship

$$F(s) = L\{f(t)\} = \int_{0}^{\infty} f(t)e^{-st}\,dt \tag{12.4}$$

In order to return from the s-domain back to the time domain, the inverse Laplace transform is applied. The inverse Laplace transform is defined by the symbol $L^{-1}\{.\}$; that is, one can write

$$f(t) = L^{-1}\{F(s)\} = \frac{1}{2\pi j}\int_{\sigma-j\infty}^{\sigma+j\infty} F(s)e^{st}dt\,, \tag{12.5}$$

or alternatively

$$F(s) \xrightarrow{L^{-1}} f(t) \tag{12.6}$$

Example 1:

Compute the (unilateral) Laplace transform of the function $f(t)=e^{-t}$.

Commands	Results	Comments
syms t s **t** **s** **f=exp(-t);** **laplace(f,t)**	 t=t s=s ans=1/(1+s)	The command syms is used to define the symbolic variables t and s. The function $f(t)=e^{-t}$ is defined as symbolic expression and the (unilateral) Laplace transform is computed.

Example 2:

Compute the inverse Laplace transform of the function $F(s)=\frac{1}{1+s}$.

Commands	Results	Comments
syms t s **F=1/(1+s);** **ilaplace(F,s)**	 ans=exp(-t)	The command syms is used to define the symbolic variables t and s. The function $F(s)=\frac{1}{1+s}$ is defined as symbolic expression and the inverse Laplace transform is computed.

12.2 The *residue* Function

The Matlab *residue* function converts its argument function, given in rational form, to partial fraction form. The syntax is [R, P, K] = residue (B,A), where B is the vector containing the coefficients of the numerator polynomial and A is the vector of the coefficients of the denominator polynomial. Suppose that a signal X(s) is written in the rational form

$$X(s)=\frac{B(s)}{A(s)}=\frac{b_m s^m+b_{m-1}s^{m-1}+\cdots+b_1 s+b_0}{a_n s^n+a_{n-1}s^{n-1}+\cdots+a_1 s+a_0}$$

By defining the vector $B=[b_m,\ldots,b_0]$ and $A=[a_n,\ldots,a_0]$, and executing the command [R, P, K]=residue(B,A), the signal X(s) can be written in partial fraction form as

$$X(s)=\frac{r_1}{s-p_1}+\frac{r_2}{s-p_2}+\cdots+\frac{r_n}{s-p_n}+K$$

Where, $R=[r_1,r_2,\ldots,r_n]$, $P=[p_1,p_2,\ldots,p_n]$,

and K is the residue of division between the two polynomials, K is null when m<n.

The command residue is illustrated in the following examples.

Example 3:

Express the signal transform $X(s)$ given below in partial fraction form

$$X(s)=\frac{s^2+3s+1}{s^3+5s^2+2s-8}$$

Commands	Results	Comments
num = [1 3 1]; **den = [1 5 2 -8];** **[R,P,K] = residue(num,den)**	R=0.5000 0.1667 0.3333 P=-4.0000 -2.0000 1.0000 K=[] The signal X(s) is written as $1/2\frac{1}{s+4}+1/6\frac{1}{s+2}+1/3\frac{1}{s-1}$	The signal X(s) is expanded in partial fraction form. Notice that K is empty; that is, there is no residue.
syms s **X=R(1)/(s-P(1))+R(2)/(s-P(2))+R(3)/(s-P(3))** **Pretty(X)**		

Example 4:

Express the signal transform $X(s)$ given below in partial fraction form

$$X(s)=\frac{s^2-3s+2}{s^2+4s+5}$$

In this case, the roots of the denominator polynomial are complex numbers. The complex roots do not really change anything in the followed procedure.

Commands	Results	Comments
num = [1 -3 2]; **den = [1 4 5];** **[R,P,K] = residue(num,den)**	R=-3.50-5.50i -3.50+3.50i P=-2.00+1.00i -2.00-1.00i K=1	The signal X(s) is expressed in partial fraction form as $X(s)=\frac{-3.5+5.5j}{s+2-j}+\frac{-3.5+5.5j}{s+2+j}+1$

Task 01: Compute the unilateral Laplace transform of the function $f(t) = -1.25 + 3.5te^{-2t} + 1.25e^{-2t}$. Also evaluate the inverse Laplace transform of your result.

```
syms t s
f = -1.25+3.5*t*exp(-2*t)+1.25*exp(-2*t);
l = laplace(f,s);
pretty(l);
il = ilaplace(l,t);
pretty(il);
```

OUTPUT:

```
  5        7                5
--------- + ---------- -     ---
4 (s + 2)    2(s+2)^2        4 s


exp(-2 t) 5   t exp(-2 t) 7      5
----------- + -------------   -    -
   4            2                4
```

Task 02: Compute the unilateral Laplace transform of the function $f(t)=1$.

```
syms t s
f = 1;
l = laplace(f,s);
pretty(l);
```

OUTPUT:

```
1
--
s
```

Task 03: Express in the partial fraction form the signal

$$X(s)=\frac{s^3-3s+2}{s^2+4s+5}$$

```
syms s t
num = [ 1 0 -3 2 ];
deno = [ 1 4 5 ];
[R P K] = residue(num,deno)
X = R(1)/(s-P(1)) + R(2)/(s-P(2)) + s*K(1) + K(2);
pretty(X);
```

OUTPUT:

```
    4 - 3i     4 + 3i
s + --------- + ---------- - 4
    s + 2 - i   s + 2 + 1i
```

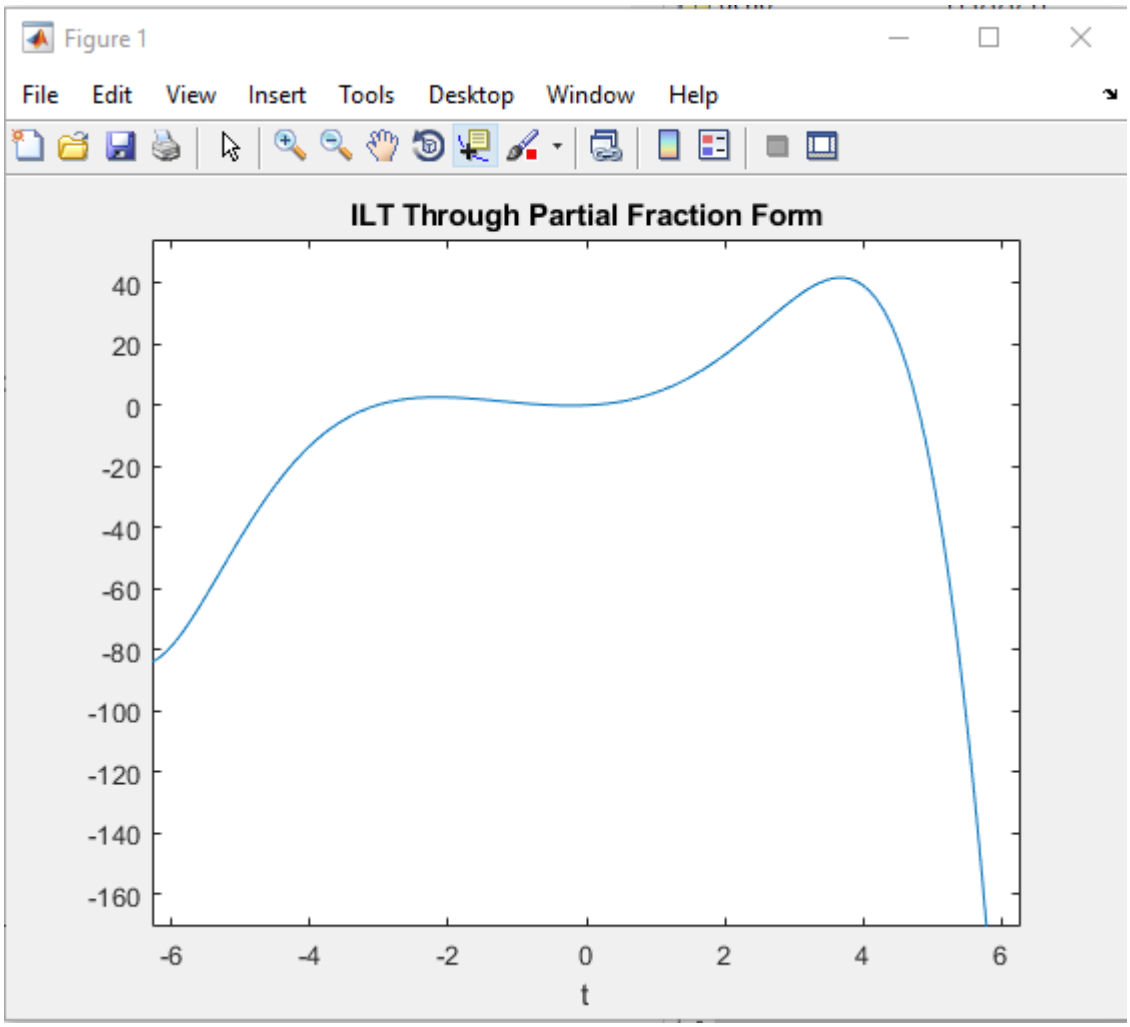
Task 04: Express in the partial fraction form the signal

$$X(s)=\frac{s^2+5s+4}{s^4+1}$$

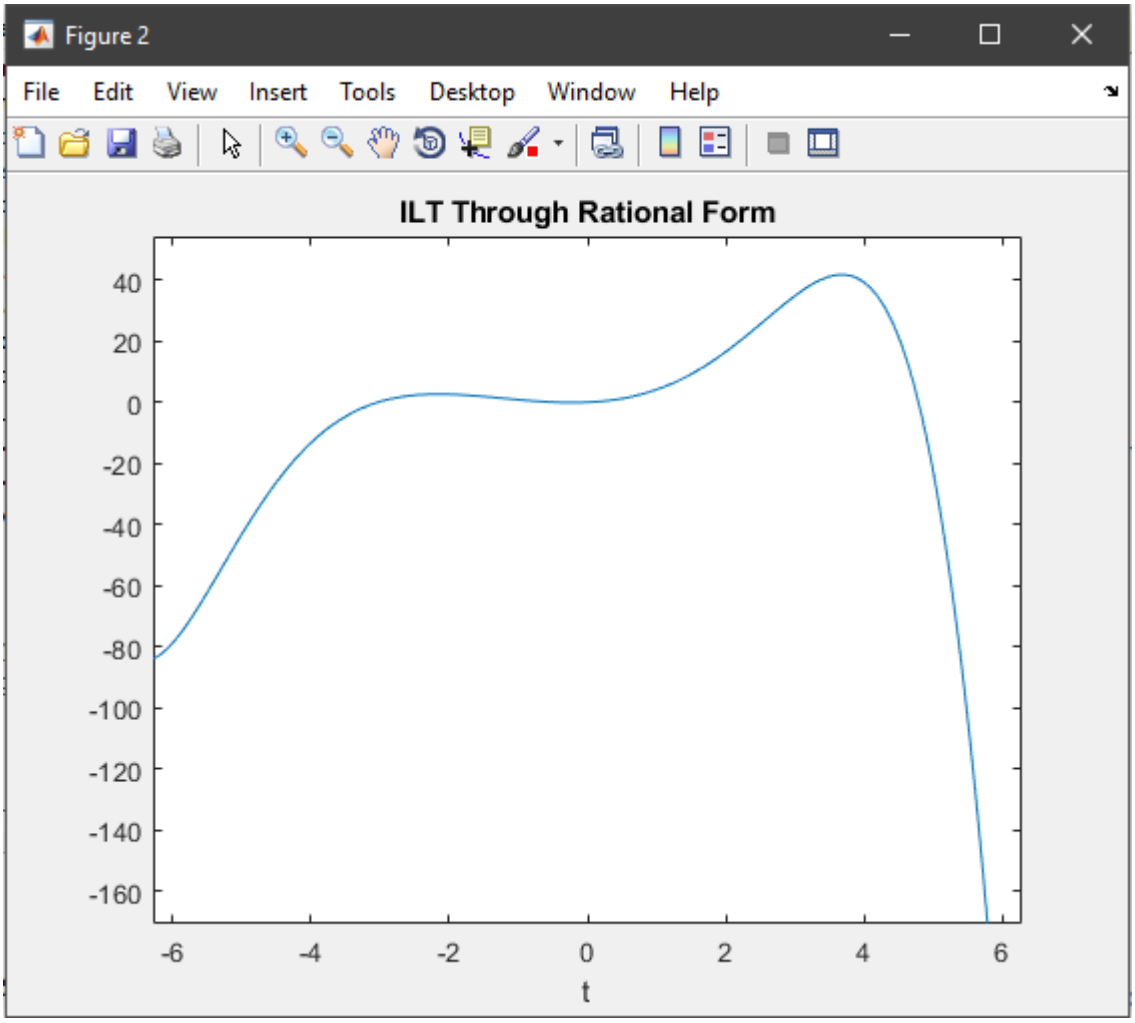
Verify your result by computing the inverse Laplace transform from both forms (partial fraction and rational) and plot both of the results.

```
syms s t
num = [ 1 5 4 ];
deno = [ 1 0 0 0 1 ];
[R P K] = residue(num,deno);
X = R(1)/(s-P(1)) + R(2)/(s-P(2)) + R(3)/(s-P(3)) + R(4)/(s-P(4)) ;
il = ilaplace(X,t)
ezplot(il);
title('ILT Through Partial Fraction Form');
syms s
figure();
rat = (s^2 + 5*s + 4)/(s^4 + 1);
il2 = ilaplace(rat,t);
ezplot(il2);
title('ILT Through Rational Form');
```

Inverse Laplace through Partial Fraction



Inverse Laplace through Rational Form



In-Lab Open ended Tasks

Task 01: Examine the network shown in figure 12.1 below. Assume the network is in steady state prior to $t = 0$.

I. **Plot the output current $i(t)$, for $t > 0$**
II. **Determine whether the system is stable or not?**

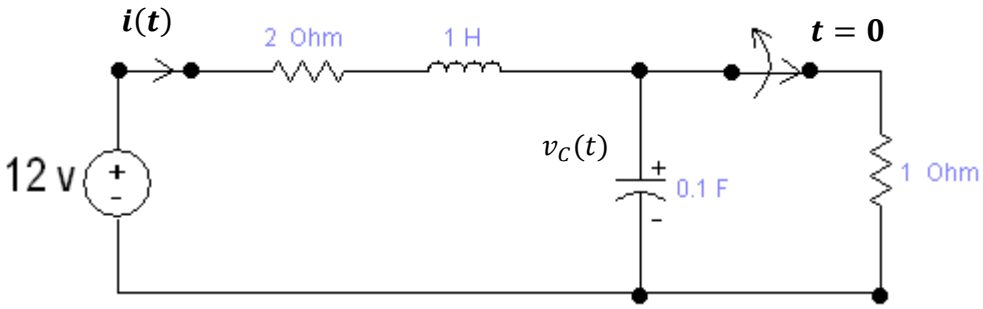
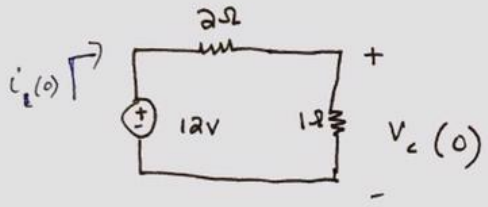


Figure 12.1

Task 1

for $t < 0$



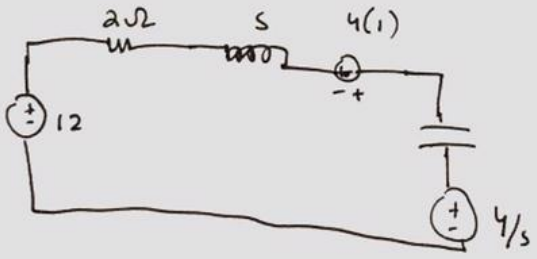
$$V_c(0) = \frac{1}{3}(12) \quad \therefore \text{ by voltage divider.}$$

$$\boxed{V_c(0) = 4V}$$

$$i_L(0) = \frac{V}{R} = \frac{12}{3}$$

$$\boxed{i_L(0) = 4A}$$

for $t > 0$

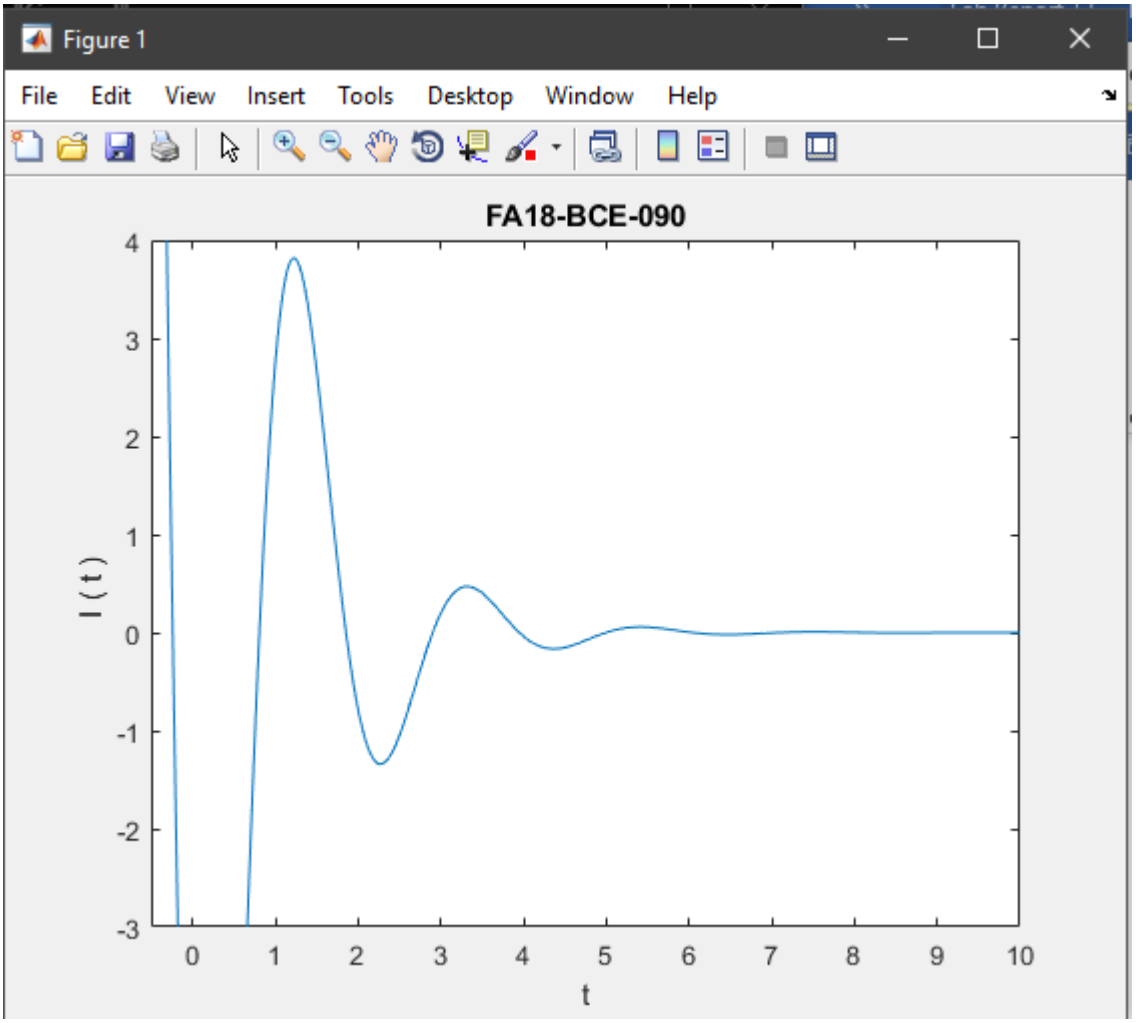


$$i(s) = \frac{12 + 4 - 4/s}{2 + s + 10/s} = \frac{4s - 1}{s + 2 + 10/s}$$

$$\boxed{i(s) = \frac{4s - 1}{s + 2 + 10/s}}$$

CODE:

```
syms t s
I=(4*s-1)/(2+s+10/s);
R = (2+s+10/s);
IL=ilaplace(I,t);
ezplot(IL,[-0.5 10]);
title('FA18-BCE-090');
ylim([-3 4]);
xlabel('t');
ylabel('I ( t ) ');
E=int(abs(I*R).^2,s,0,inf)
```



BIBO Unstable

```
E=int(abs(V).^2,s,-inf,inf)

E =

Inf
```

Task 02: For the circuit shown in figure 12.2, the input voltage is $v_i(t) = 10\cos(2t)\,u(t)$

I. **Plot the steady state output voltage $v_{oss}(t)$ for $t > 0$ assuming zero initial conditions.**
II. **Determine whether the system is stable or not?**

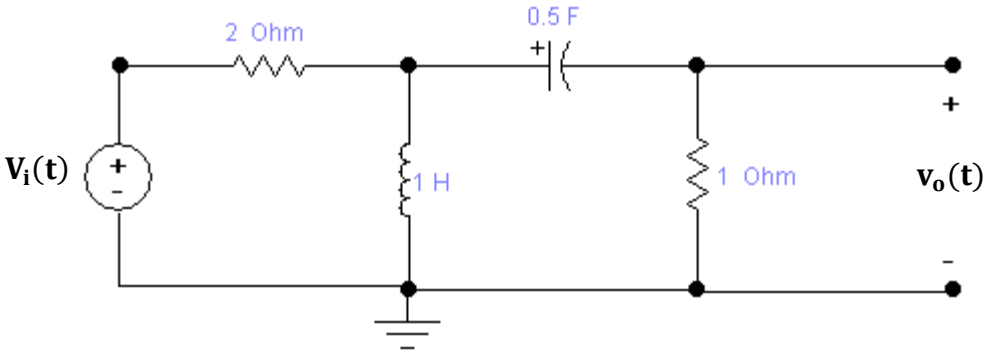
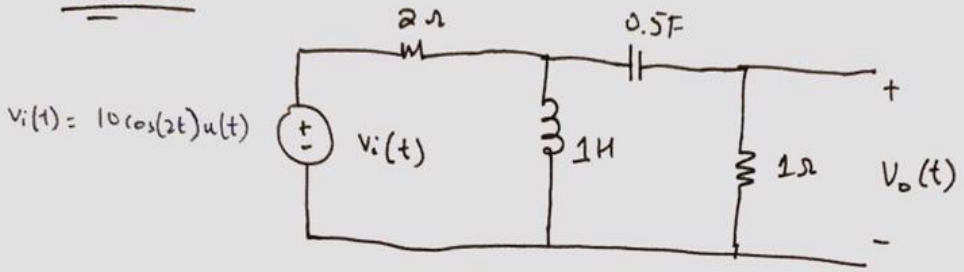
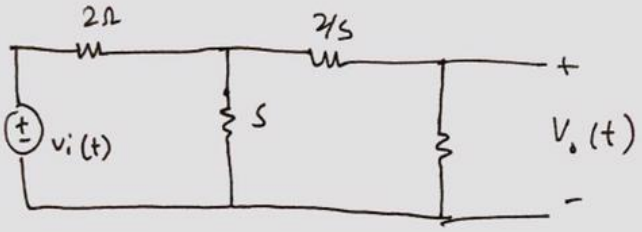


Figure 12.2

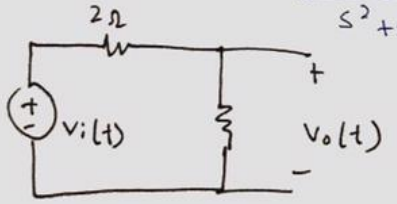
Task 2





$$1 + 2/s \;||\; s \Rightarrow \frac{\frac{s+2}{s}\cdot s}{s + \frac{s+2}{2}} = \frac{s+2}{\frac{s^2+s+2}{s}}$$

$$= \frac{(s+2)s}{s^2+s+2}$$



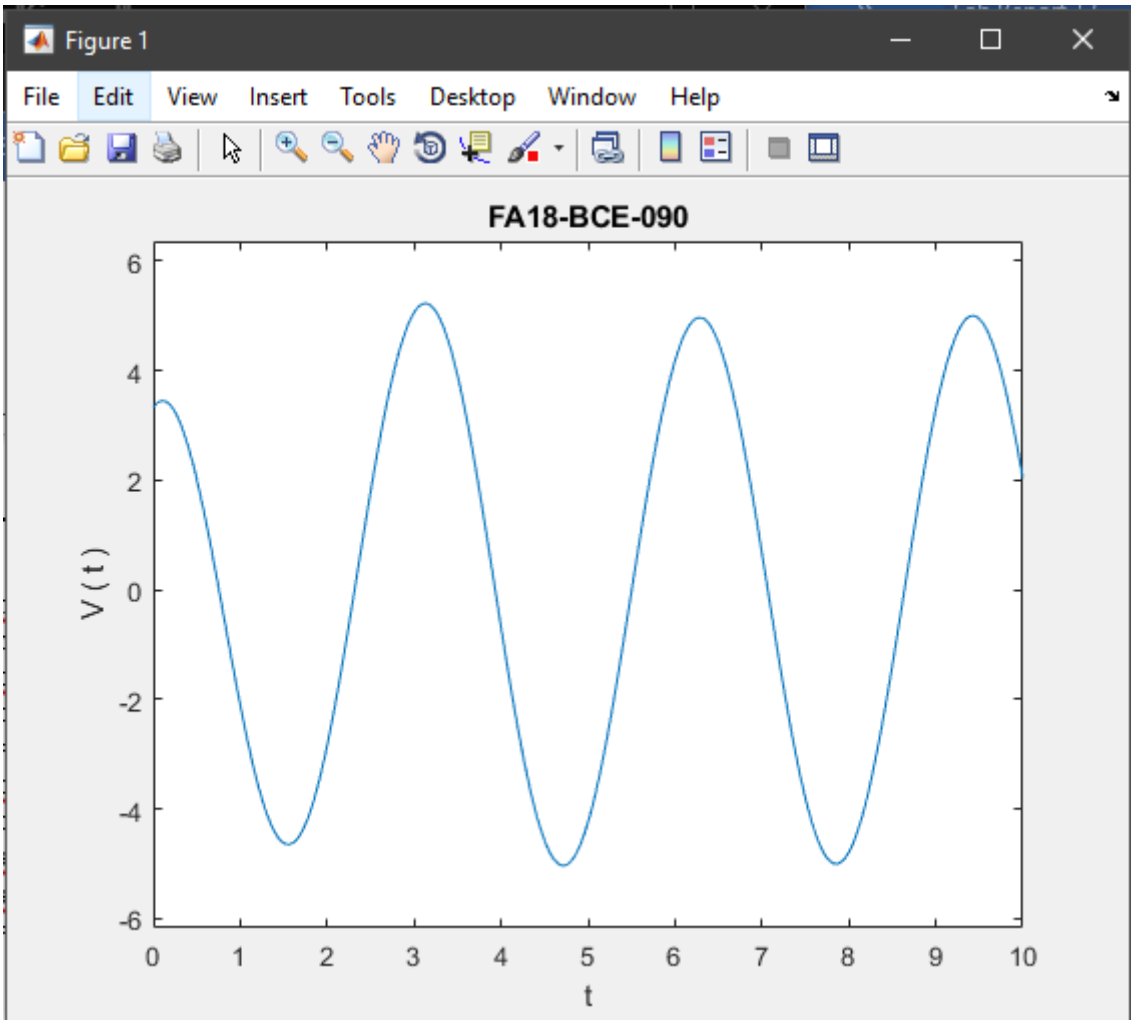
Applying voltage divider:

$$V_o(t) = \frac{\frac{(s+2)s}{s^2+s+2}}{2 + \frac{(s+2)s}{s^2+s+2}} \times v_i(t)$$

$$\boxed{V_o(t) = \left(\frac{s(s+2)}{3s^2+4s+4}\right) \times V_i(t)}$$

CODE :

```
syms t s
Vin=10*cos(2*t)*heaviside(t);
lap_Vin=laplace(Vin,s);


output=lap_Vin*((s*(s+2))/(3*(s^2)+4*s+4));
Vo=ilaplace(output,t);
ezplot(Vo,[0 10]);
title('FA18-BCE-090');
xlabel('t');
ylabel('V ( t )');
e = int(abs(output).^2,s,-inf,inf)
```



BIBO Stable

```
e = int(abs(output).^2,s,-inf,inf)

e =
 (25*pi*2^(1/2))/16
```

***Critical* Analysis / Conclusion**

In this lab we learned about Laplace transformation as well as Inverse Laplace transformation. We used “laplace” and “ilaplace” commands in MATLAB to convert from time (t) domain to Laplace (s) domain or from Laplace (s) domain to time (t) domain, respectively. These commands were applied in RLC circuits as well. We also learned the residue function which converts a given rational function into partial fraction.

Transform pair	Signal	Transform	ROC
1	$\delta(t)$	1	All s
2	$u(t)$	$\frac{1}{s}$	$\mathcal{R}e\{s\} > 0$
3	$-u(-t)$	$\frac{1}{s}$	$\mathcal{R}e\{s\} < 0$
4	$\frac{t^{n-1}}{(n-1)!}u(t)$	$\frac{1}{s^n}$	$\mathcal{R}e\{s\} > 0$
5	$-\frac{t^{n-1}}{(n-1)!}u(-t)$	$\frac{1}{s^n}$	$\mathcal{R}e\{s\} < 0$
6	$e^{-\alpha t}u(t)$	$\frac{1}{s+\alpha}$	$\mathcal{R}e\{s\} > -\alpha$
7	$-e^{-\alpha t}u(-t)$	$\frac{1}{s+\alpha}$	$\mathcal{R}e\{s\} < -\alpha$
8	$\frac{t^{n-1}}{(n-1)!}e^{-\alpha t}u(t)$	$\frac{1}{(s+\alpha)^n}$	$\mathcal{R}e\{s\} > -\alpha$
9	$-\frac{t^{n-1}}{(n-1)!}e^{-\alpha t}u(-t)$	$\frac{1}{(s+\alpha)^n}$	$\mathcal{R}e\{s\} < -\alpha$
10	$\delta(t-T)$	e^{-sT}	All s
11	$[\cos \omega_0 t]u(t)$	$\frac{s}{s^2+\omega_0^2}$	$\mathcal{R}e\{s\} > 0$
12	$[\sin \omega_0 t]u(t)$	$\frac{\omega_0}{s^2+\omega_0^2}$	$\mathcal{R}e\{s\} > 0$
13	$[e^{-\alpha t}\cos \omega_0 t]u(t)$	$\frac{s+\alpha}{(s+\alpha)^2+\omega_0^2}$	$\mathcal{R}e\{s\} > -\alpha$
14	$[e^{-\alpha t}\sin \omega_0 t]u(t)$	$\frac{\omega_0}{(s+\alpha)^2+\omega_0^2}$	$\mathcal{R}e\{s\} > -\alpha$
15	$u_n(t) = \frac{d^n\delta(t)}{dt^n}$	s^n	All s
16	$u_{-n}(t) = \underbrace{u(t) * \cdots * u(t)}_{n \text{ times}}$	$\frac{1}{s^n}$	$\mathcal{R}e\{s\} > 0$

Table 12.1: Transform pairs for Laplace transform

Time Domain	Frequency Domain (s-domain)
$f(t)$	$F(s)$
$Kf(t)$	$KL[f(t)]$
$f_1(t) + f_2(t) - f_3(t) + \ldots$	$F_1(s) + F_2(s) - F_3(s) + \ldots$
$\frac{df(t)}{dt}$	$sL[f(t)] - f(0^-)$
$\frac{d^2 f(t)}{dt^2}$	$s^2 L[f(t)] - sf(0^-) - \frac{df(0^-)}{dt}$
$\frac{d^n f(t)}{dt^n}$	$s^n L[f(t)] - s^{n-1} f(0^-) - s^{n-2}\frac{df(0^-)}{dt} - \ldots - \frac{d^{n-1} f(0^-}{dt^n}$
$\int_{0^-}^{t} f(t)dt$	$\frac{L[f(t)]}{s}$
$f(t-a)u(t-a), a > 0$	$e^{-as}L[f(t)]$
$e^{-at}f(t)$	$F(s+a)$
$f(at), a > 0$	$\frac{1}{a}F\left(\frac{s}{a}\right)$
$tf(t)$	$\frac{-dF(s)}{ds}$
$t^n f(t)$	$(-1)^n \frac{d^n F(s)}{ds^n}$
$\frac{f(t)}{t}$	$\int_s^\infty F(u)du$

Table 12.1: Properties of Laplace transform