Ahsanullah University of Science and Technology

Department of Electrical and Electronic Engineering

LABORATORY MANUAL

FOR

ELECTRICAL AND ELECTRONIC SESSIONAL COURSES

Student Name:

Student ID:

Course No: EEE 3218

Course Name: **Digital Signal Processing Laboratory**

For the students of

Department of Electrical and Electronic Engineering

3^{rd} Year 2^{nd} Semester

Table of contents

Department of Electrical & Electronic Engineering
Ahsanullah University of Science and Technology (AUST)

LAB 1:
Overview of DSP LAB (EEE 3218)

Objectives

The main objectives of this lab are:

- Introduction to basic terminologies
- Motivation for the DSP Lab
- Overview of the DSP Lab

PRELAB TASKS

- Read this laboratory tutorial carefully before coming to the laboratory class, so that you know what is required.
- Follow the lecture notes of EEE 3217.
- Familiarize yourself with relevant MATLAB functions and codes necessary for this experiment.
- Do not bring any prepared MATLAB code in the lab with any portable device.

PART 1

Introduction

A **signal** is a function of a set of independent variables with time being perhaps the most prevalent single variable. The signal itself carries some kind of information available for observation. Most signals of interest in practice are recorded values of a physical quantities, represented as a 1-D functions, such a time function , **x(t)** like speech or 2-D/3-D functions, such as a spatial function **f(x,y)** like image or function of both space and time **f(x,y,t)** like video . A signal carries **information** and contains **energy**.
By the term **processing**, we mean operating in some fashion on a signal to extract some useful **information**. In many cases, this processing will be a nondestructive **"transformation"** of the given data signal; however, some essential processing methods turn out to be irreversible and thus destructive.
Finally, the word **digital** means that the processing is done with a digital computer or specific purpose digital hardware. For better clarification, we can briefly define **analog**, **discrete**, and **digital** signals as follows:

- **Analog Signal:** a function x(t), continuous in amplitude, of a continuous independent variable **t** (e.g., time).
- **Discrete Signal:** a function $\mathbf{x[n] = x(nt_0)}$, continuous in amplitude, but defined only at a set of discrete values of the independent variable, $\mathbf{t = 0, \pm t_0, \pm 2t_0, ...}$
- **Digital Signal:** a discrete signal with quantized (finite) amplitude values. For example, there are $2^8 = 256$ gray scale levels in an 8-bit digital image (Figure 1.1).

Figure. 1.1: A digital image of a DSLR Camera

Advantages & Applications of Digital Signal Processing

Advantages of using digital signals, including digital signal processing (DSP) and communication systems, include the following:

- Digital signals can convey information being less susceptible to noise, distortion, and interference.
- Digital circuits can be reproduced easily in mass quantities at comparatively low costs.
- Digital signal processing is more flexible because DSP operations can be altered using digitally programmable systems.
- Digital signal processing is more secure because digital information can be easily encrypted and compressed.
- Digital systems are more accurate, and the probability of error occurrence can be reduced by employing error detection and correction codes.
- Digital signals can be easily stored on any magnetic media or optical media using semiconductor chips.

Applications of DSP are increasing in many areas where analog electronics are being replaced by DSP chips, and new applications are depending on DSP techniques. With the cost of DS processors decreasing and their performance increasing, DSP will continue to affect engineering design in our modern daily life. Some application examples using DSP are listed in Table 1.1.

Table 1.1: Applications of Digital Signal Processing

Digital Audio and Speech	Digital audio coding such as CD players, digital crossover, digital audio equalizers, digital stereo and surround sound, noise reduction systems, speech coding, data compression and encryption, speech synthesis and speech recognition.
Digital Telephone	Speech recognition, high-speed modems, echo cancellation, speech synthesizers, DTMF (Dual-Tone Multifrequency) generation and detection, answering machines.
Automobile Industry	Active noise control systems, active suspension systems, digital audio and radio, digital controls.
Electronic Communications	Cellular phones, digital telecommunications, wireless LAN (Local Area Networking), satellite communications.
Medical Imaging equipment	ECG analyzers, cardiac monitoring, medical imaging and image recognition, digital X-rays and image processing.
Multimedia	Internet phones, audio and video; hard disk drive electronics; digital pictures; digital cameras; text-to-voice and voice-to-text technologies.

However, the list in the table by no means covers all DSP applications. Many more areas are increasingly being explored by engineers and scientists. Applications of DSP techniques will continue to have profound impacts and improve our lives.

DSP Nomenclatures & Topics

Figure 1.2 shows a broad overview of digital signal processing. Analog signals enter an ADC from the left, and samples exit the ADC from the right, and may be 1) processed strictly in the discrete time domain (in which samples represent the original signal at instants in time) or they may be 2) converted to a frequency domain representation (in which samples represent amplitudes of particular frequency components of the original signal) by a time-to-frequency transform, processed in the frequency domain, then converted back to the discrete time domain by a frequency-to-time transform. Discrete time domain samples are converted back to the continuous time domain by the DAC. Note that a particular signal processing system might use only time domain processing, only frequency domain processing, or both time and frequency domain processing, so either or both of the signal processing paths shown in Fig. 1.2 may be taken in any given system.

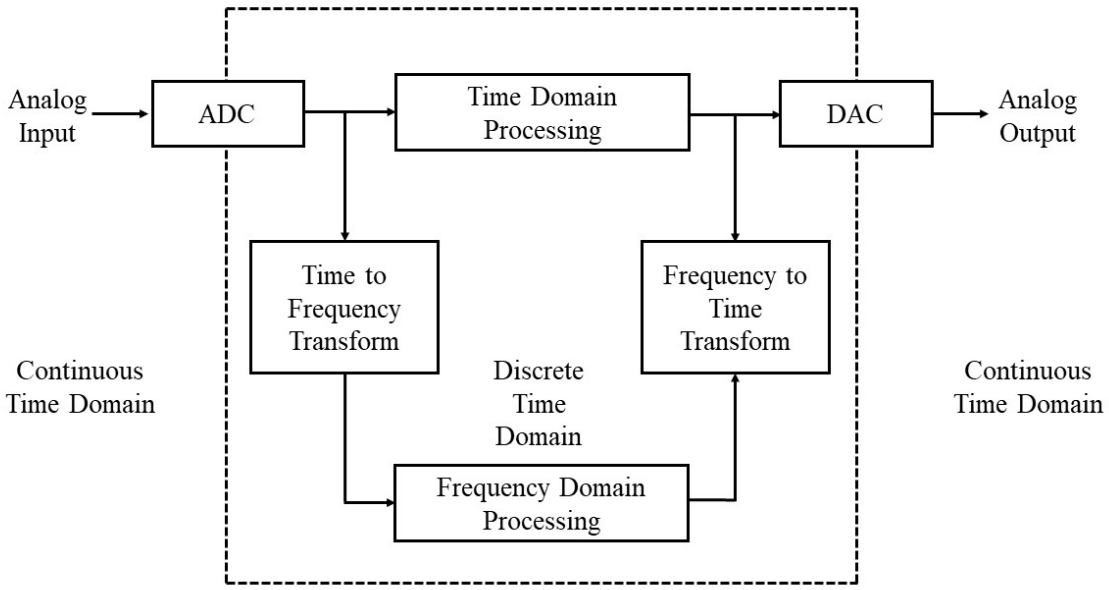


Figure. 1.2: A broad, conceptual overview of digital signal processing.

Time Domain Processing

In time domain processing the variation of the input and output variables are observed with respect to time. The examples of time domain processing includes

- Basic signal generations.
- Signal manipulation in time domain.
- Signal composition and decompositions.
- Linear Time Invariant (LTI) system response in time domain and convolution.
- Auto-correlation and Cross-correlation etc.

Frequency Transforms

A time-to-frequency transform operates on a block of time domain samples and evaluates the frequency content thereof. A set of frequency coefficients is derived which can be used to quantify the amplitudes (and usually phases) of frequency components of the original signal or the coefficients can be used to reconstruct the original time, domain samples using an inverse transform (a frequency-to-time transform). The most well-known and widely-used of these transforms is the **Discrete Fourier Transform (DFT)**, usually implemented by the **FFT (for Fast Fourier Transform)**, the name of a class of algorithms that allow efficient computation of the **DFT. Z-transforms** also convert a time domain signal to complex frequency domain.

Frequency Domain Processing

Most signal processing that can be done in the time domain can be also equivalently done in the frequency domain. Each domain has certain advantages for a given type of problem. Time domain filtering, for example, can be performed using frequency transforms such as the DFT, and in certain cases, efficiency can be greatly improved using this technique. A second use is in digital filter design, in which the desired filter frequency response is specified in the frequency domain, i.e., as a set of DFT coefficients, for example. Yet a third and very prevalent use is **Transform Coding**, in which signals are coded using a frequency transform (usually eliminating as much redundant information as possible) and then reconstructed from the transform coefficients. Transform Coding is a powerful tool for compression algorithms, such as those employed with MP3 (MPEG II, Level 3) for audio signals, JPEG, a common image compression format, etc. The use of such compression algorithms has revolutionized the audio and video fields, making storage of audio and video data very economical and deliverable via Internet.

Probable List of Lab Tasks

1 Week 1: introduction to DSP

Time Domain Processing

- Week 2: Discrete Time Signals and Linear System.

- Week 3: Convolution and Discrete LTI System response.
- Week 4: Sampling.

Domain Transformation

- Week 5: Frequency Transformation:
- Week 6: Z-transform:

Filter Design

- Week 7: IIR Filter Design
- Week 8: FIR Filter Design
- Week 9: Application of all the process learnt to a real life signal.

Assessment Procedure and Marks Distribution (Tentative)

Method	Percentage
a. Attendance	10
b. Lab Reports & Performance	30
C. Lab Examination 1	10
d. Surprised Test (Lab Exam 2)	10
e. Lab Examination 3	10
f. Lab Final	30
Total	100

Answer the following questions and attach them with your lab report.
Questions:

1. Can you mention name of some physical signals along with their responding human organ? Are they Analog or Digital?
2. Are the terms data and signal synonymous? Briefly clarify with example.
3. Do you think all digital data are always represented by 1 and 0 only? Briefly Clarify.
4. When speed of processing is the only factor to be considered, whether Analog or Digital Processing should be used? Why?
5. Which processing requires less power? Analog or Digital? Why?

Part 2
An Overview of Using MATLAB and Octave

- Demonstration of signal-processing example using MATLAB
- Recap of MATLAB/OCTAVE & Some warm up problems.
- Functions to be used: help(),plot(), sin(), cos(),sind(),cosd()

Instructions

- Organizing files and folder properly for later use.
- Make a unique named folder for this lab like **EEE3218_SP20_A1_180105001** under any drive other than C drive.
- For every week make a subfolder to keep the all the tasks in a specific week separated from other tasks.
- Always try to work in the folder specifically created for the current week.
- Follow MATLAB/OCTAVE naming convention for giving a meaningful name before saving it in MATLAB/OCTAVE.
- While running a code be judicious in choosing *add to path* or *change directory*. Use add to path when you are using a file from different folder.

About MATLAB

MATLAB is an environment for scientific computation especially suitable for computations that require extensive use of arrays and graphical analysis. In this lab, all works will be done using MATLAB. So a brief overview on how to use it arises naturally.

MATLAB works with three types of windows on computer screen. These are the Command window, the Figure window and the Editor window. Any kind of commands are typed in the command window, such as the command to run a function, indicated by the prompt (>>). All the output and error will appear in this command window. The Figure window only pops up whenever you plot something. The Editor window is used for writing and editing MATLAB programs (called M files). Current Drive or Folder allows you to access your saved files. Workspace explore data that you create or import from files. A figure is shown in figure 1.3 to show the desktop when you start MATLAB.

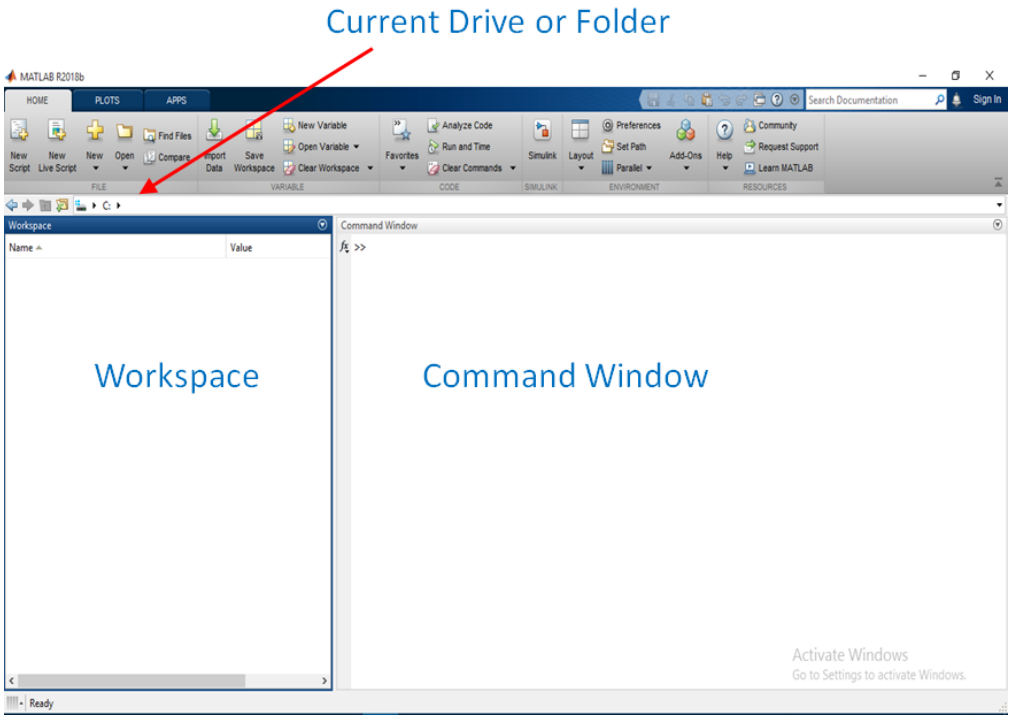


Figure. 1.3: MATLAB Window

About OCTAVE

OCTAVE is an open-source program that you can use as an alternative to MATLAB. Its basic numerical functions are very similar to MATLAB, in terms of appearance and usage. In addition, because the OCTAVE language is similar to MATLAB, most MATLAB programs should be able to run on OCTAVE.

You can download the OCTAVE from the following link.

https://www.gnu.org/software/octave/download

When you start OCTAVE, the desktop appears in its default layout as the following figure 1.4.

- Current Directory — Create or access your files.
- File Browser — You can browse your files in any directory throughout your computer.
- Workspace — The workspace contains variables that you create or import into Octave from data files or other programs. You can view and edit the contents of the workspace in the Workspace browser or in the Command Window.
- Command History — The Command History window displays a log of statements that you ran in the current and previous Octave sessions. The Command History lists the time and date of each session in the short date format for your operating system, followed by the statements from that session.

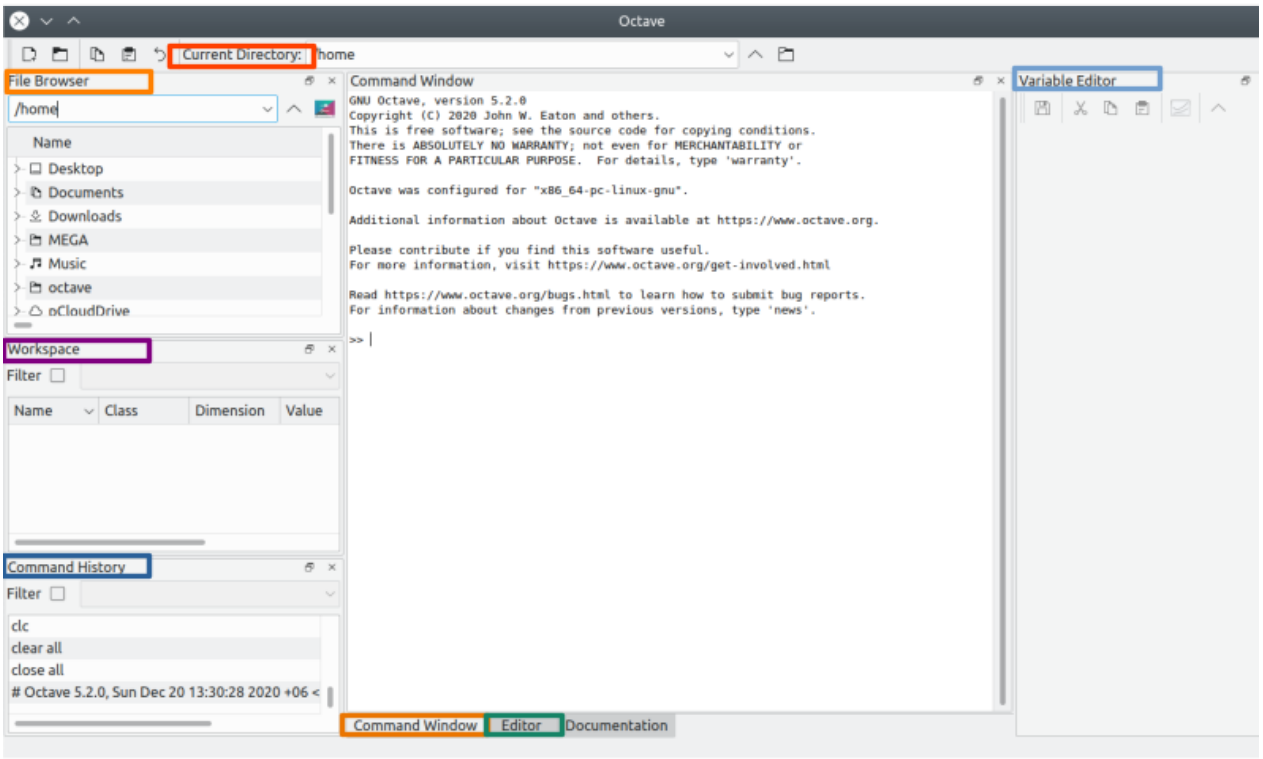


Figure. 1.4: OCTAVE Window

- Variable Editor — The workspace consists of the variables you create and store in memory during a Octave session. You can create new variables in the workspace by running Octave code or using existing variables. You can view and edit those variables using Variable Editor.
- Command Window — Command Window is the main window where you type commands directly to the Octave interpreter.
- Editor — The Editor window is a simple text editor where you can load, edit and save complete Octave programs. The Editor window also has a menu command (Debug/Run) which allows you to submit the program to the command window.
- Documentation — The Documentation window gives you access to a great deal of useful information about the Octave language and Octave computing environment. It also has a number of example programs and tutorials.

Figure 1.5 shows the Editor window of the Octave.
However, functions in Octave are divided into different packages. These packages do not load in Octave by default. You need to manually load them in the environment. To do this, you need to write the following command in the command window.

pkg load package_name

Here, package_name is the name of the package that you will use. For DSP lab, we need the signal package. That means, the package name will be signal. Therefore, you need to write

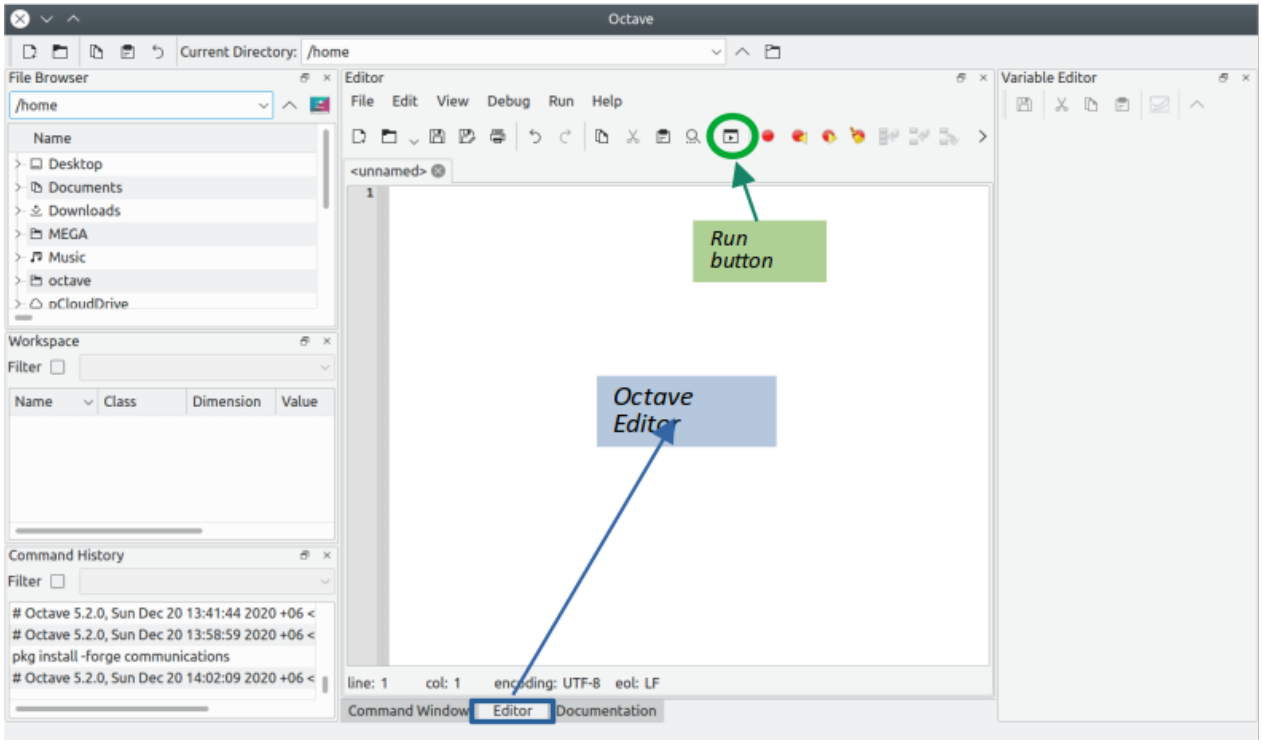


Figure. 1.5: OCTAVE Editor Window

the following command in the command window each time you open/start the Octave.

pkg load signal

MATLAB/OCTAVE Basics

Variable Declaration

While declaring a variable in MATLAB/OCTAVE, the following things should be remembered:

- Variable Names are case-sensitive i.e. **A** and **a** are different variables.
- A variable name in MATLAB/OCTAVE can contain upto 63 characters.
- Variable names must begin with a letter. Following that, any number of letters, digits and underscores can be added.

Following code snippet shows different ways of declaring variables in MATLAB/OCTAVE. The fifth variable declaration will give an error upon compiling because it started with a number instead of a letter. And also any name that resembles with built in MATLAB/OCTAVE variable like **pi** is not recommended.

Listing 1: Examples of Variable Declaration

```
clc;  % use to clear all the previous lines in the command window
clear all; % use to clear all the previous variables
close all; % use to close all the figure windows used previously

x = 2; % Valid
a_b_c = 1; % Valid
X = 10; %This X and the first x are not the same
        % MATLAB is case-sensitive
this_is_a_variable = 3; % Valid
3abc = 15; %This is not a valid variable name
           % This will give an error
pi = 5; %This is also not a valid variable name as pi is the
        % built in variable for MATLAB. This will not give any error
        % but not recommended
```

Data Type

The default data type in MATLAB/OCTAVE is double precision array. While we'll use this type of data mostly in this lab, several other data types are also important. MATLAB/OCTAVE variables can be used in command prompt without pre-definition. As arrays will be used mostly in our lab, we'll discuss how to define and access arrays. Few rules of defining and accessing arrays:

- Array indices begin with 1. So if $A = [1, 2, 3, 4, 5]$ then A[1] = 1 and A[0] will give an error.
- Column elements are separated by a comma (,) and row elements are separated by semicolon (;).
- When defined, array elements are contained between the [and] symbols.
- The element in **i** row and **j** column of matrix A($\mathbf{A_{ij}}$) is accessed by **A(i,j)**.
- A sequential list of elements can be generated by the colon (:) operator using the following form: **initial value: increment : final value**.
- Complex entries are used using **i** and **j**.

Few examples corresponding the above points are given below in a MATLAB/OCTAVE Code snippet.

Listing 2: MATLAB/OCTAVE Codes for declaring and accessing Arrays

```
clear all;
clc;
```

```
3
4  a = [1, 3, 5];  % creating a 1x3 row array
5  b = [2; 4; 6];  % creating a 3x1 column array
6
7  A = [1, 3, 5; 2, 4, 6]; %creating a 2x3 matrix
8
9  c = A(1,2); % c is the element of 1st row and 2nd column of A
10 d = A(2,2); % d is the element of 2nd row and 2nd column of A
11
12 B = 2:2:10; % B is an array containing element from 2 to 10 with an increment
       2
13 C = [2,4,6,8,10]; % B and C are the same array
14
15 D = [1+2*j, 2+1*j, 4-5*j] % creating a 1x3 complex row vector
```

Array Operations

The arithmatic operations +,-,* and / by default correspond to true matrix computation. Few points regarding array operations.

- **A*B** is the usual matrix product between A and B. Here the dimension mismatch will give an error.
- **A.*B** is the element by element manipulation. Any operator followed by **dot(.)** will be element by element operation.
- Any function of a matrix will result in an output with the individual element as argument of the function. For example exp(A) will result in a matrix with element being the exponential of each element of A.

Listing 3: Sample code from MATLAB/OCTAVE on array operations

```
1  clc;
2  clear all;
3
4  A = [1, 2, 3]; % a row array of 3 elements
5  B = [2, 4, 6]; % a row array of 3 elements
6  C = [1; 2; 3]; % a column array of 3 elements
7
8  D = A+B ; % D is the sum of the two arrays.
9  E = A-B;  % E is the subtraction of the two arrays
10 F = C*A;  % F is the matrix product of A and C
11 G = A.*B; % G is the element wise product of A and B
12 H = exp(A)+sin(B); % This computes two functions of A and B. The size of any
     function of any array is that of the array.
```

Extracting Sub-matrix

A portion of a matrix can be extracted and stored in a smaller matrix by specifying the rows and columns to extract. The syntax is:

$$\textbf{sub\_matrix} = \textbf{matrix(r1:r2, c1:c2)}$$

where r1(c1) and r2(c2) denote the beginning and ending rows(columns). Suppose we have a 4×5 matrix A given by the following matrix and we want to extract the portion marked with lines.

$$\left(\begin{array}{ccccc} 1 & 3 & 5 & 7 & 9 \\ 0 & 2 & 4 & 6 & 8 \\ \hline 9 & 7 & 5 & 3 & 1 \\ 8 & 10 & 6 & 4 & 5 \end{array}\right)$$

Listing 4: Extracting Submatrix

```
clc;
clear all;
close all;

A = [1 3 5 7 9; 0 2 4 6 8; 9 7 5 3 1; 8 10 6 4 5]; % defining the matrix

% We want to extract the submatrix [9 7 5; 8 10 6] which consists of
% elements of 3rd and 4th row and 1st, 2nd and 3rd column of A
sub_mat = A(3:4, 1:3);

Output:

sub_mat =

     9     7     5
     8    10     6
```

Plotting with MATLAB/OCTAVE

MATLAB/OCTAVE has a lot of functions for plotting. The basic one will plot one vector vs. another. Command **plot(a,b)** will plot the **b** vector vs. **a** vector treating the elements of a vector as abscissa (or x axis) and the second as the ordinate (or y axis). The vectors have to be the same length. If the second vector is not given, MATLAB/OCTAVE will plot the vector vs. its own index. There are many functions that can be used to format the figures. One can edit the title, the plotting style, color and so on. You can know about any MATLAB/OCTAVE functions by typing **help function-name** in the command window.

Listing 5: Sample code from MATLAB/OCTAVE on plotting

```
clc;
clear all;
close all;


x = 1:0.1:10; % creating an array of elements from 1 to 10 with 0.1 increment

y = x.^2 + 20*x + 15*x.*sin(x); %computing a function of x

plot(x,y, 'Linewidth', 2); % plotting the figure
```

stem is used for displaying a discrete-time signal. The command format is the same as plot. We'll learn more about stem in the next lab. Whenever multiple plots are needed to be shown in a figure, the function **subplot** is used. The basic syntax is **subplot(m,n,p)**. **subplot(mnp)** breaks the Figure window into an m-by-n matrix of small axes, selects the p^{th} axes object for for the current plot, and returns the axis handle. The axes are counted along the top row of the Figure window, then the second row, etc. For example

subplot(2,1,1), plot(time, signal1)

subplot(2,1,2), plot(time, signal2)

The whole figure window is divided into a 2-by-1 matrix. In the upper portion of the window, signal1 is plotted and signal2 is plotted in the lower portion.
An example code is given below:

Listing 6: Use of Subplot

```
clc;
clear all;
close all;

%In this problem we'll generate two signals
%and plot them in a single figure using subplot

t = -10:0.01:10; %defining time variable

signal1 = t.^2; % signal1 = t^2
signal2 = t.^3+4*t.^2+5; % signal2 = t^2+2*t+sqrt(t)

subplot(2,1,1)
plot(t, signal1, 'Linewidth', 2); % it will plot the signal in the top
   portion ...
                                  % of the window
subplot(212) %note that (2,1,2) or (212) both are correct
plot(t, signal2, 'Linewidth', 2);
```

For Loop

Like any other programming language, MATLAB also has functions for iteration structures. We'll see only the **for** function here. The syntax for writing for loop is:

```
for <variable = expression> do
    sentence;
    sentence;
    ...  ...
    ...  ...
    sentence;
end
```

An example code is given here:

Listing 7: Sample code on for loop

```
clc;
clear all;

% This code computes the sum of the elements of a given array
% using for loop
a = [1, 2, 3, 4, 5, 6]; % given array
n = length(a); % length gives the length of the vector
sum = 0; % initializing the sum
for i = 1:n %remember in matlab indices start from 1
    sum = sum + a(i); %computes the sum
end % every for loop has an end!
```

Sample Problem 1: Compile and run the codes used for **plotting** and **subplotting** different signals and draw the output waveforms.
Sample Problem 2: Generate a sinusoidal signal with 10 kHz and show it for 10ms. Assume amplitude values and necessary data. Use **sin()** and **plot()** function first. Then use **cos()**, **sind()**, **cosd()** functions to repeat the same. [Hint: use doc sin command in command prompt]
Sample Problem 3: Generate a sinusoidal signal with 1 kHz with a initial value = $\frac{A}{\sqrt{2}}$. **A** is the amplitude value = 10 here show its first 4 cycles. Use sin() and plot() function first. Then use cos(), sind(), cosd() functions to repeat it.
Sample Problem 4: Generate the following waveshape given in figure 1.6 with proper labeling.

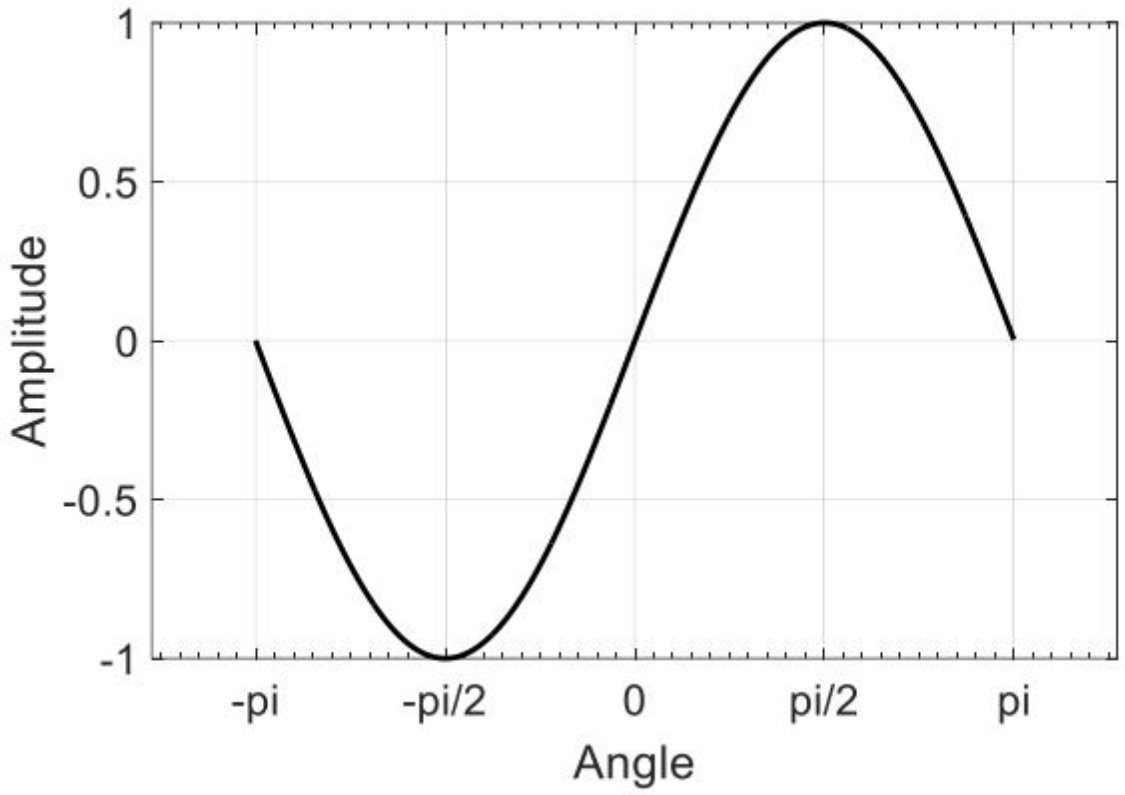


Figure. 1.6: Waveform for Sample Problem 4

Post Lab Tasks

- **Go through the Overview of MATLAB/OCTAVE Part thoroughly.**
- **Submit the codes along with the plots as a MS word file strictly by the next week. Also work out the sample problems.**
- **Use a cover page containing the lab name and code along with student's detailed particulars.**

LAB 2:
Discrete Time Signals & Linear System

Objectives

The main objectives of this lab are:

- Introduction to some elementary discrete time signals
- Concept of elementary Signal Operations
- Basic idea of Linear & Non-linear System

PART 1

Introduction

Depending on the nature of independent variables, signals are broadly classified into continuous and discrete signals. A continuous signals will be denoted by $x(t)$ in which the independent variable **t** can take any value. A discrete signal will be denoted by $x[n]$ in which the independent variable **n** can take only integer values. First we'll discuss different types of DT signals, some elementary operations on the signals and finally we'll discuss the idea of a linear and non-linear system. The DT signals discussed here very important because they can be used to construct or represent more complex signals.

Some Elementary DT Signals

Unit Impulse Sequence

It is also called *unit sample sequence* or *Kronecker Delta function* and is denoted by $\delta[\mathbf{n}]$. It is defined by the following way:

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

More generally,

$$\delta[n - n_0] = \begin{cases} 1, & n = n_0 \\ 0, & n \neq n_0 \end{cases} \tag{2}$$

Unit Step Signal

The unit step signal, denoted by **u[n]** is defined as:

$$u[n] = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases} \tag{3}$$

More generally,

$$u[n-n_0] = \begin{cases} 1, & n \geq n_0 \\ 0, & n < n_0 \end{cases} \tag{4}$$

Sinusoidal Sequence

DT sinsoidal sequence is represented by: $x[n] = A\sin(n\omega + \phi)$ where ω is the angular frequency and ϕ is the phase. Unlike continuous sinusoid signals, DT sinusoids may or may not be periodic depending on the value of ω. For a DT sinusoid to be periodic, ω must be a **rational multiple** of 2π i.e. $\omega = k \times 2\pi$ where, k is a rational number.

Elementary Signal Operations

In this section, we'll consider some simple modifications or manipulations involving the independent variable and the signal amplitude (dependent variable). Transformation of the independent variable(time) can be of three types:

1. Time Shifting
2. Time Reversal
3. Time Scaling

Time Shifting

Let us consider a discrete-time signal x[n]. If we replace the independent variable **n** by **n-k** (k is an integer), this results in a shift of the signal in time. If $k > 0$, the time shift result in a delay of the signal by k units in time. On the other hand, if $k < 0$, the time shift results in an advance of the signal by $|k|$ units in time.

Time Reversal

If we replace the independent signal **n** by **-n**, the transformation results in the folding or reflection of the signal about the time origin $n = 0$. This is called **Time Reversal** operation. Time reversal and time shifting operations are required while computing convolution of two signals.

Time Scaling

A third modification of the independent variable is the replacement of **n** by **nk** where k is an integer. This is termed as **Time Scaling**. Time scaling of a discrete signal is related to the rate of sampling. If $k > 1$, this modification is *downsampling* or *decimation* and if $0 < k < 1$, this is called *upsampling* or *interpolation*. We'll learn about downsampling and upsampling in detail in Lab 4.

The transformation of dependent variable or signal amplitude include *Amplitude scaling*,

signal addition and *signal multiplication*. Amplitude scaling is simply multiplying the signal values by a constant number. Addition and multiplication of two signals are defined on a sample-to-sample basis. For example, while computing addition of two signals $x_1[n]$ and $x_2[n]$, first sample of $x_1[n]$ will be added to first sample of $x_2[n]$. For this reason, in order to add or multiply two discrete signals, they must be of the same length.

Even-Odd Decomposition of a Signal

A real valued signal $x[n]$ is called even or symmetric if it satisfies the condition: $x[-n] = x[n]$ i.e. time reversal of the signal will not alter it. On the other hand, a signal is called odd or anti-symmetric if $x[-n] = -x[n]$. Examples of signals with even and odd symmetry are shown in figure.
It can be shown that any **real** signal can be expressed uniquely as the sum of an even signal and an odd signal. The even signal component is formed by adding $x[n]$ to $x[-n]$ and dividing by 2.

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

(Verify this is an even signal!). In the same manner, the odd component $x_0[n]$ can be formed:

$$x_0[n] = \frac{1}{2}(x[n] - x[-n])$$

(Verify this is an odd signal!). You can check that adding $x_e[n]$ and $x_0[n]$ will result in the original signal $x[n]$. Note that, the even-odd decomposition depends on the location of the origin. If the signal is shifted, the decomposition will change.

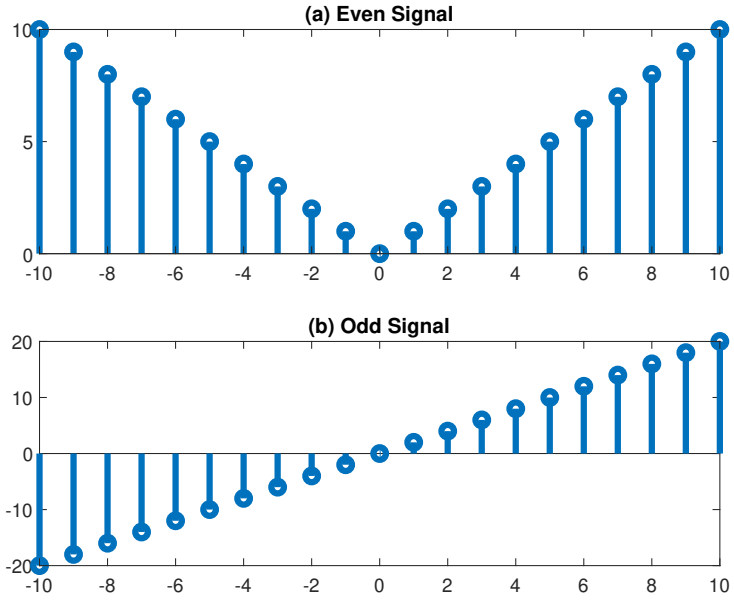


Figure. 2.1: Example of Even and Odd Signal

Discrete Time Systems: Linear & Non-Linear System

A discrete time system is a device or an algorithm that operates on an input discrete-time signal according to some well defined rules to produce an output discrete time signal. The output signal is also called the response of the system. Mathematically, a system is a function. The input signal $x[n]$ is transformed by the system into a signal $y[n]$ which we express mathematically,

$$\mathbf{y[n] = \mathcal{T}(x[n])}$$

where $\mathcal{T}$ represents the transformation. Here it is noted that, in general $y[n]$ is a function of the entire sequence $\{x[n]\}$ not just the single time point $x[n]$.

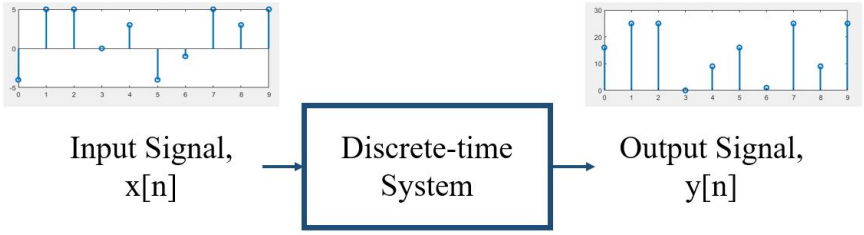


Figure. 2.2: Block Diagram of a discrete-time system

Discrete time systems can be subdivided into **linear** and **non-linear** system. A linear system is one that satisfies the **additive** and **homogeneity** property. **Additivity** property implies that the sum of the input signals will be transformed into the sum of the individual output signals. For example, suppose $y_1[n]$ and $y_2[n]$ are the corresponding output signals for inputs $x_1[n]$ and $x_2[n]$ respectively. Now if the input is the sum of $x_1[n]$ and $x_2[n]$ i.e. $x[n] = x_1[n] + x_2[n]$ the output will be the sum of $y_1[n]$ and $y_2[n]$: $y[n] = y_1[n] + y_2[n]$. And **homogeneity** means if the input is scaled by a factor, the output will be scaled by the same factor. For example, if $y[n]$ is the output for an input sequence $x[n]$, then $ax[n]$ will give the output $ay[n]$. If a system satisfies these two properties, it is called **Linear System** otherwise it is known as **Non-Linear System**. The two properties are, sometimes, combined into one: **superposition principle**. So if a signal satisfies the superposition principle, it is linear otherwise, it is non-linear. For example, system described by $y[n] = 10x[n]$ is linear but $y[n] = x^2[n]$ is non-linear.
Linear Systems are important and useful in digital signal processing. Because linear systems are easier to analyze and many non-linear systems are approximately linear to the first order.

PRELAB TASKS

- Sketch qualitatively the following DT signals in a paper: i) $\delta[n+2] + 2 * u[n-1] + 3 * u[n-4]$.
- Find the period of the following DT signals: i) $\sin(\sqrt{3}\pi n)$, ii) $\sin(\frac{\pi}{10}n) + \cos(\frac{\pi}{5}n)$.
- If x[n] = {2, $\underset{\uparrow}{1}$, 3, 4}, sketch x[-n+3].
- Make a list of three practical examples of linear and non-linear systems.

Part 2
Discrete-Time Signal Operations in MATLAB/OCTAVE

- Generating the Discrete Time Signals using MATLAB/OCTAVE
- Elementary Signal operations implemented using MATLAB/OCTAVE
- How to define a function
- Functions to use: **stem()**, **exp()**, **sin()**, **min()**, **max()**, **fliplr()**.

Instructions

- Organizing files and folder properly for later use.
- Make a unique named folder for this lab like **EEE3218_SP20_A1_180105001** under any drive other than C drive.
- For every week make a subfolder to keep the all the tasks in a specific week separated from other tasks.
- Always try to work in the folder specifically created for the current week.
- Follow MATLAB/OCTAVE naming convention for giving a meaningful name before saving it in MATLAB/OCTAVE.
- While running a code be judicious in choosing *add to path* or *change directory*. Use add to path when you are using a file from different folder.

In this part of our lab, we'll show how to generate the discussed Discrete time signals and how to implement various signal operations i.e. time shifting, time reversal, addition and multiplication of signals using MATLAB/OCTAVE. Finally, we'll see the definitive property of linear and non-linear system using some examples.
In this week's MATLAB/OCTAVE code, we'll write some functions so that we can use them later. The basic syntax of any function is:

Listing 1: Syntax for writing a Function in MATLAB/OCTAVE

```
function [output vectors] = name_of_the_function(input vectors)

%Write the code what you want to do with the function

end
```

Remember: the function name must not conflict with any built in function's name and save the function with the name written in the .m file. For example, here we've written a function to generate $\delta[n - n_0]$ named unit_impulse. The inputs to this function are the starting (n_1) and end point(n_2) of the index/time variable and the location of the impulse (n_0).

Listing 2: Unit Impulse Sequence

```
function [x,n] = unit_impulse(n0, n1, n2)

% Generates x[n] = delta(n-n0)
% n1<=n<=n2
% n0 is the origin of the unit impulse function
% We'll use logical index concept here to compute
% unit impulse

n = n1:n2; % defining the index vector n

x = (n==n0); % here logical indexing is used i.e. x = 1 if n==n0.

end
```

Later call this function with appropriate inputs and outputs in your main code like the following code snippet. Here the **unit_impulse** function has been called to generate $\delta[n-2]$ and plot them for $-5 \leq n \leq 5$.

Listing 3: Examples of generating unit impulse sequence

```
clc;
clear all;
close all;

[x,n] = unit_impulse(2,-5,5); %calling unit_step function
stem(n,x); %displaying the output
```

Similarly, unit step function $u[n]$ can be generated as shown in the following code snippet:

Listing 4: Unit Step Function

```
function [u, n] = unit_step(n0, n1, n2)

% n0 = the origin
% n1 = starting index number
% n2 = ending index number

```

```
n = n1:n2; %defining the index vector

u = (n>=n0); % u[n-n0] = 1 if n>=n0. This implemented here using
               ... logical indexing
end
```

Listing 5: Example Code to generate unit step sequence

```
clc;
clear all;
close all;


[x,n] = unit_step(2,-5,7); %calling unit_step function
stem(n,x); %displaying the output
```

The following code has been written to generate a general sinusoidal with zero damping. The expression for this function is:

$$x[n] = A \exp(\text{an}) \sin(n\omega_0 + \phi)$$

where **A** is the amplitude, **a** is the damping constant, ω_0 is the angular frequency and ϕ is the phase. In the following code, A = 10, a = 0, $\omega_0 = \frac{\pi}{5}$ and $\phi = 0$. It has been plotted for $-10 \leq n \leq 10$.

Listing 6: Generating Sinusoidal Sequence

```
clc;
clear all;
close all;

%In this code we'll show a sinusoidal function
% of the form x[n] = A*exp(a*n)*sin(w0*n+phi)
n = -10:10; %defining the index variable
w0 = pi/5; %angular frequency of the sinusoid
a = 0; %damping factor
phi = 0; %phase in radian
A = 10; %amplitude
x = A*exp(a*n).*sin(w0*n+phi); %Look at the elementwise operation!!

stem(n,x); %plotting the output
```

Now we'll write two functions to show time shifting and time reversal operation. In time reversal operation, a built in function is used: **fliplr()**. You can know more about this function by going through the documentation of this function.

Listing 7: **sigshift** Function to implement time shifting

```
function [y,n] = sigshift(x,n1,n0)
```

```
%We'll generate y[n] = x[n-n0]

n = n1+n0; %new index
y = x; % Here only the index will change, not the actual signal's value

end
```

Listing 8: **sigfold** Function to implement time reversal

```
function [y,n] = sigfold(x, m)

% This code will generate y[n] = x[-n]
% The folding operation
% Use fliplr function which flips the elements of an array left to right
n = -fliplr(m); % generating n = -m index
y = fliplr(x);
end
```

You can call these functions to your main code to implement $x[n-4]$ or $x[-n]$ given function $x[n]$.
Now we'll add two signals using MATLAB/OCTAVE. Remember from part 1, while adding, the length of the signals must be the same. In order to make the length of the signals same, required additional zeros can be inserted before or after a signal. For example: in case of $x_1 = \{1, \underset{\uparrow}{2}, 3\}$ $x_2[n] = \{3, 4, \underset{\uparrow}{5}, 6\}$, a zero can be inserted at the beginning of x_1 so that its length and indices get matched to those of $x_2[n]$. This is called **zero padding**. In the following code snippet, at first this zero padding is done. Then element wise addition has been carried out.

Listing 9: **sigadd** function to add two signals

```
function [y,n] = sigadd(n1, x1, n2, x2)

% In order to add two signals, we need to
% make their length equal by padding zeros otherwise dimension mismatch
% will occur
%n1 = index vector of signal x1
%n2 = index vector of signal x2
%x1 = signal 1
%x2 = signal 2

n = min(min(n1),min(n2)):max(max(n1),max(max(n2))); % Defining the index
%vector for the output signal. For example if n1 = -3:2 and n2 = 0:4
% then n will run from -3 to 4. So we need to define 'n' from the minimum
% of individual minimum of n1 and n2. The same goes for the maximum.
y1 = zeros(1, length(n)); %initializing two vectors to store
y2 = y1;  ... the zero padded signals
```

```
y1 (((n>=min(n1)) & (n<=max(n1)))) = x1; % The idea is that if min(n1)<=n<=
    max(n1)
y2 (((n>=min(n2)) & (n<=max(n2)))) = x2; ... then y1 = x1(y2=x2).
y = y1+y2;

end
```

Finally this function is called to exectue the addition of two signals.

Listing 10: Code to add two signals using **sigadd**

```
clc;
clear all;
close all;


n1 = -2:2;
n2 = -3:1;

x1 = n1; %x1 = 2n1
x2 = n2; %x2 = square(n2)

[y,n] = sigadd(n1,x1,n2,x2); %calling sigadd function to add two signals
%you can plot the output signals to visualize them
```

In this exact process following addition, signal multiplication can also be done. Using **sigadd** and **sigfold** functions, one can decompose any signal $x[n]$ into even and odd components. First generate $x[-n]$ and then use the formula for even and odd component given in part 1.

Finally we'll demonstrate the definitive property of linear and non-linear system. We'll show that if the superposition of two signals are given input to a linear system, the output will also be the superposition of the individual outputs and for non-linear system this will not be the case.

For this reason, we're defining two systems: system1 and system2. **system1** is defined by the equation: $y[n] = 0.5x[n]$ and **system2** is defined by: $y[n] = x^2[n] + 5x$. Then we'll take two signals $x_1[n] = n + 2$ and $x_2[n] = n$ for $0 \leq n \leq 9$. We'll give the following inputs to **system1** and **system2**: $2x_1[n]$, $3x_2[n]$ and $2x_1[n] + 3x_2[n]$. Then we show the output curves for each of these signals. The code for doing this task is given below. In the following codes, it has been assumed that the input signals are of the same length to avoid complexity.

Listing 11: Defining **system1**

```
function [y,n] = system1(x,n)

y = 0.5*x;

end
```

Listing 12: Defining **system2**

```
function [y,n] = system2(x,n)

y = x.^2+5*x;

end
```

Listing 13: Output results for **system1**

```
clc;
clear all;
close all;

%Linear System & Non Linear system demonstration

n = 0:9; %index vector
x1 = n+2; %input signal 1
x2 = n; %input signal 2
%output of system1 & system2 defined by functions system1 and system2
% we're assuming x1 and x2 are of the same length for avoiding complexity
[y1, n1] = system1(2*x1, n);
[y2,n2] = system1(3*x2,n);
[y3,n3] = system1(2*x1+3*x2,n);

subplot(211)
stem(n1, y1+y2);
subplot(212)
stem(n3, y3);
```

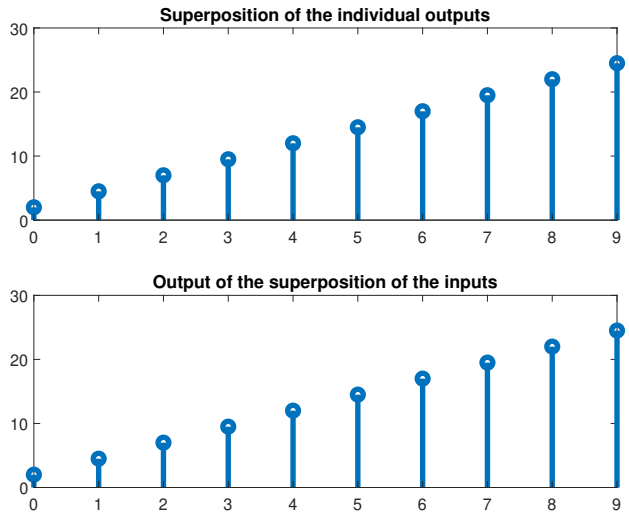


Figure. 2.3: Output Plots for system1

Observing the plot figure(2.3), it is seen that the sum of the individual output is equal to the output corresponding to the superposed inputs. So **system 1** is a linear system.

Post Lab Tasks

- **Problem 1**: Modify the function **sigadd** renaming **sigmult** to multiply two signals.
- **Problem 2**: Using function **sigfold** and **sigadd**, write a code that will decompose any real signal into its even and odd counterparts. Plot the component.
- **Problem 3**: Verify that **system2** is non-linear by writing a code just like the linear one.
- **Problem 4**: If x[n] = {1, 1, 2, 3, $\underset{\uparrow}{5}$, 8, 13, 21, 34, 55}, using the functions **sigfold** and **sigshift** show i) x[-n-3] and ii) x[-n+4].
- **Problem 5**: Are time shifting and time reversal operations commutative? Verify this by using **problem 4**.

LAB 3:
Convolution
and
Discrete LTI System Response

Objectives

The main objectives of this lab are:

- Introduction to Convolution
- Properties of Convolution
- Introduction to Cross-correlation and Auto-correlation

PART 1

In our previous lab week, we studied systems and the difference between a linear and non-linear system. The output of a system is also termed as *response* of the system. When the input to a system is a delta signal i.e. $\delta(t)$ or $\delta[n]$, the corresponding output is called **Impulse Response** of the system. Among various types of systems, Linear Time Invariant (LTI) systems are of special attention due to their mathematical simplicity. The impulse response of system can be determined in time domain by solving the characteristic difference equation. And an LTI system is completely characterized by its impulse response in time domain. Using the concept of **convolution sum**, we can determine the output of any linear time invariant system to any arbitrary input signal from its impulse response.

Decomposition of a DT signal into impulses

Recall that the impulse/unit impulse sequence $\delta[n - n_0]$ is zero everywhere except $n = n_0$ where it has a value 1. Consider an arbitrary signal x[n]. If we multiply x[n] by $\delta[n]$, we'll get x[0] as $\delta[n]$ is zero everywhere except n = 0. So, we can write

$$\mathbf{x[n]}\delta[\mathbf{n}] = \mathbf{x[0]}\delta[\mathbf{n}]$$

This is called sifting property. Due to this property, we can extract any sample from x[n] by multiplying it by a shifted delta, $\delta[n - k]$. In general,

$$\mathbf{x[k]}\delta[\mathbf{n - k}] = \mathbf{x[k]}\delta[\mathbf{n - k}]$$

is a sequence that is zero everywhere except at n = k where its value is x[k]. Thus if we repeat this multiplication over all possible values of k, $-\infty \leq k \leq \infty$, and sum all the product sequences, the result will be a sequence equal to the sequence x[n]. That is,

$$\mathbf{x}[\mathbf{n}] = \sum_{\mathbf{k}=-\infty}^{\infty} \mathbf{x}[\mathbf{n}]\delta[\mathbf{n - k}]$$

. Thus any arbitrary input signal can be expressed as the sum of shifted and scaled unit impulses. For example,

$$x[n] = \{2, 3, \underset{\uparrow}{5}, 8, 13\} = 2\delta[n+2] + 3\delta[n+1] + 5\delta[n] + 8\delta[n-1] + 13\delta[n-2]$$

Now, we'll see due to this decomposition as well as linearity and time invariance property, the response of an LTI system to an arbitrary input signal can be expressed in terms of the unit impulse response of the system.

Convolution Sum

Let us consider a discrete time signal **x[n]** input to an LTI system with impulse response **h[n]** (figure 3.1). We denote the output of the system sa **y[n]**. Recall that, **h[n]** is the response of the system to the unit impulse function δ[**n**].

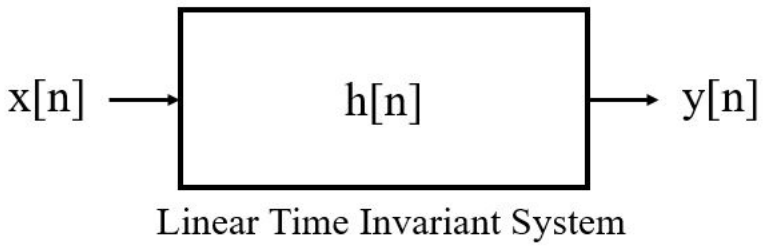


Figure. 3.1: Block Diagram of a Linear Time Invariant System with input x[n], impulse response h[n] and output y[n].

We represent this as: $\boldsymbol{\delta}$**[n]** $\longrightarrow$ **h[n]**. This means that h[n] is the output of the system to a $\delta[n]$ input.

Because the system is time invariant: $\boldsymbol{\delta}$**[n-k]** $\longrightarrow$ **h[n-k]**

Because the system is linear, we apply homogeneity property: **x[k]**$\boldsymbol{\delta}$**[n-k]** $\longrightarrow$ **x[k]h[n-k]**

Because of additivity property due to linearity: $\sum_{\mathbf{k}=-\infty}^{\infty}$ **x[k]** $\boldsymbol{\delta}$**[n-k]** $\longrightarrow$ $\sum_{\mathbf{k}=-\infty}^{\infty}$ **x[k]h[n-k]**

Now from the previous section, we recognize that $\sum_{\mathbf{k}=-\infty}^{\infty}$ **x[k]**$\boldsymbol{\delta}$**[n-k]** is just the input signal x[n] and $\sum_{\mathbf{k}=-\infty}^{\infty}$ **x[k]h[n-k]** is the total output y[n] of the system. The total response of the system is termed as the **convolution sum** of the sequence x[n] and h[n]. The convolution sum operator is represented by putting $*$ symbol between the sequences. So, we can summarize the above discussion by stating that the input sequence x[n] is convolved with the impulse

response h[n] to yield the output y[n]. Mathematically,

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

Therefore, as we said earlier, an LTI system is completely characterized by its impulse response. The actual computation process of convolution sum will be discussed in theory (EEE 3217). In MATLAB, we'll compute this sum using a built-in function named **conv**. We'll discuss more about this function in the coding section of this manual.

Properties of Convolution

In this section, we'll mention some properties of convolution. We'll not going in detail discussion or formal proof of these properties.

Commutative Property

Commutative property states that the order in which two sequences are convolved is not important. Mathematically, this means that

$$y[n] = x[n] * h[n] = h[n] * x[n]$$

From a system point of view, this property states that a system with input x[n] and impulse response h[n] behaves exactly the same way as a system with input h[n] and impulse response x[n]. This is shown in figure 3.2(a).

Associative Property

Convolution operator satisfies associative property which is

$$y[n] = (x[n] * h_1[n]) * h_2[n] = x[n] * (h_1[n] * h_2[n])$$

From a system point of view this property states that if two systems with impulse response $h_1[n]$ and $h_2[n]$ are cascaded as shown in figure 3.2(b), the equivalent system will have an impulse response equal to the convolution of $h_1[n]$ and $h_2[n]$.

$$h_{\text{eq}}[n] = h_1[n] * h_2[n]$$

Distributive Property

The distributive property states that

$$x[n] * (h_1[n] + h_2[n]) = x[n] * h_1[n] + x[n] * h_2[n]$$

This property implies that if two systems with impulse response $h_1[n]$ and $h_2[n]$ are connected in parallel as shown in figure 3.2(c), the equivalent system will have an impulse response equal to the summation of $h_1[n]$ and $h_2[n]$.

$$h_{\text{eq}}[n] = h_1[n] + h_2[n]$$

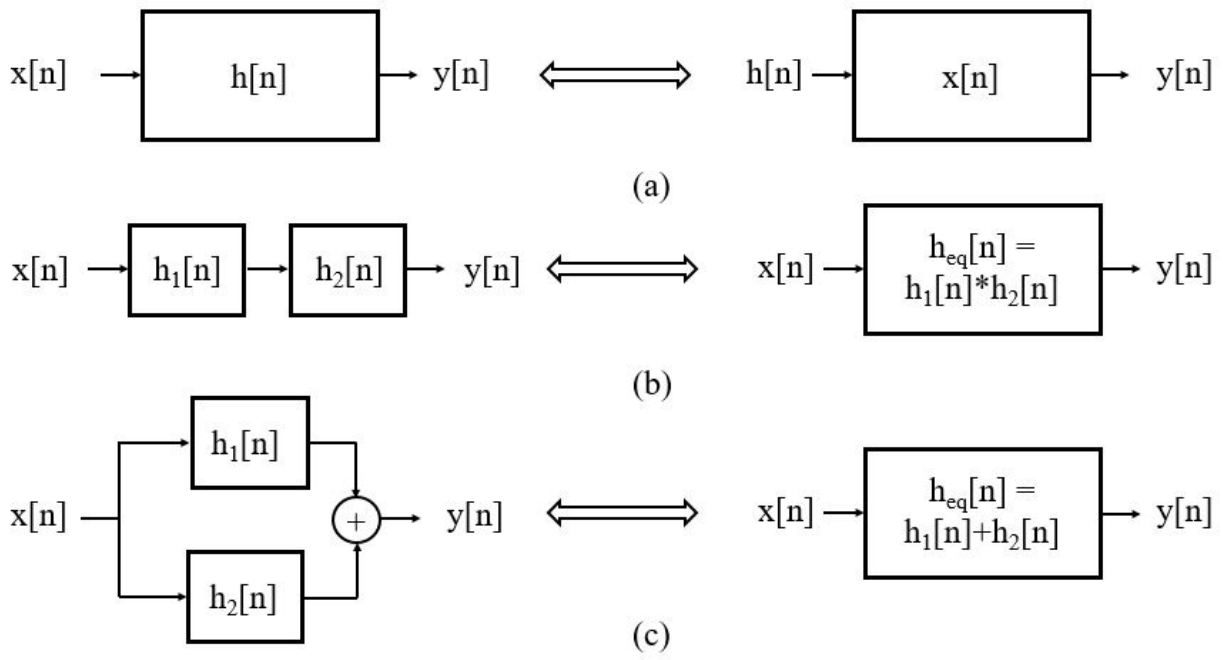


Figure. 3.2: Interpretation of Convolution properties from a system point of view.

System Response From Difference Equation

Another way to find the response of a system to an input is solving difference equations. In this section, we'll discuss briefly about a family of linear time invariant systems described by an input-output relation called a **difference equation with constant coefficients**. You can think of a difference equation 'the discrete version' of a differential equation.
A causal, linear, time-invariant system can be described by a difference equation having the following general form:

$$y[n] + a_1 y[n-1] + ... + a_N y[n-N] = b_0 x[n] + b_1 x[n-1] + ... + b_M x[n-M]$$

where $a_1, ..., a_N$ and $b_0, ..., b_M$ are the coefficients of the difference equation. The above equation can be written as:

$$y[n] = -\sum_{i=1}^{N} a_i y[n-i] + \sum_{k=0}^{M} b_k x[n-k]$$

Notice that y[n] is the current output, which depends on the past output samples $y[n-1], ..., y[n-N]$, the current input sample x[n], and the past input samples, $x[n-1], ...x[n-M]$. Notice that in this equation the coefficient of y[n] is 1. We'll see an example on how to solve a difference equation using MATLAB.

Correlation

A mathematical operation that closely resembles convolution is correlation. Correlation is computed to measure the degree to which two signals are similar. Correlation is used in radar, active sonar applications and in digital communication system. Suppose that we have two real signal sequences x[n] and y[n] each with finite energy. The cross-correlation of x[n] and y[n] is a sequence $r_xy[l]$ which is defined as:

$$\mathbf{r_{xy}[l]} = \sum_{\mathbf{n}=-\infty}^{\infty} \mathbf{x[n]y[n-l]}$$

or equivalently, as

$$r_{xy}[l] = \sum_{n=-\infty}^{\infty} x[n+l]y[n]$$

The index l is the (time) shift (or lag) parameters and the subscripts xy on the cross-correlation sequence $r_{xy}[l]$ indicate the sequences being correlated. It can easily be checked that

$$r_{xy}[l] = x[l] * y[-l]$$

Cross correlation is simply the convolution of one signal with folded version of another. This makes cross-correlation a non-commutative operation.
A distinct peak in the cross-correlation function indicates that the two signals are matched for that particular time shift. This has important applications in signal detection and system identification.
In the special case where y[n] = x[n], we have the *autocorrelation* of x[n] which is defined as the sequence:

$$\mathbf{r_{xx}[l]} = \sum_{\mathbf{n}=-\infty}^{\infty} \mathbf{x[n]x[n-l]}$$

Autocorrelation function of a periodic signal is periodic. Autocorrelation can be used to detect a noisy periodic signal, estimate the impulse response of system and find the period of a periodic signal etc.

PRELAB TASKS

- Decompose the signal into impulse sequences: x[n] = {1,4,9, $\underset{\uparrow}{16}$, 25}
- Is time invariant property a necessary condition to determine the output of a system using convolution sum defined in part 1? Explain why.
- If $y[n] = x[n] * h[n]$ what is the relationship between the time index vector of y[n] and those of x[n] and h[n]? Give an example.
- Compute cross-correlation of x[n] = {1,2,$\underset{\uparrow}{2}$} and y[n] = {1,$\underset{\uparrow}{0}$,1}. Make a list of five applications of corrletaion in signal Processing and communication system.

Part 2
Discrete Time System Response using MATLAB/OCTAVE

- Convolution in MATLAB/OCTAVE
- Verification of Convolution Properties in MATLAB/OCTAVE
- Solving Difference Equation in MATLAB/OCTAVE
- Applications of Correlation
- Functions to use: **stem()**, **conv()**, **min()**, **max()**, **xcorr()**, **randn()**

Instructions

- Organizing files and folder properly for later use.
- Make a unique named folder for this lab like **EEE3218_SP20_A1_180105001** under any drive other than C drive.
- For every week make a subfolder to keep the all the tasks in a specific week separated from other tasks.
- Always try to work in the folder specifically created for the current week.
- Follow MATLAB/OCTAVE naming convention for giving a meaningful name before saving it in MATLAB/OCTAVE.
- While running a code be judicious in choosing *add to path* or *change directory*. Use add to path when you are using a file from different folder.

In this part of our lab, we'll show how to compute convolution sum of two sequences using **conv** function and verify the properties of convolution in MATLAB/OCTAVE. We'll also solve a difference equation using two different approach and finally we'll how to apply the concept of correlation in signal processing.
First we'll discuss how can we compute convolution sum of two discrete sequences using **conv()** function. w = conv(a,b) returns the convolution of vectors a and b. However, conv() function does not provide any information about the time index of the output signals. So

the time index vector of the sum is to be derived from the signals to be convolved. Recall that the lowest time index of the convolution is the sum of the lowest time index of the individual signals. The same is applicable in case of the highest time index of the sum. So along with the signals, we have to provide their individual time index vectors. For example, if a signal is given by: x[n] = {1,1,$\underset{\uparrow}{0}$,1,1}, the corresponding time index vector is n_x = [-2,-1,0,1,2](Recall that, the $\underset{\uparrow}{d}$enotes the origin). Similarly, if there is another signal h[n] = {1,-2,-3,$\underset{\uparrow}{4}$} the corresponding index vector is n_h = [-3,-2,-1,0]. So the convolution of x[n] and h[n] will have the time index vector $n = (-2-3) : (2+0) = -5 : 2$, where (:) indicates the colon operator of MATLAB/OCTAVE. At first we'll write a MATLAB/OCTAVE function which will take two signals along with their time index vectors and return the convolution sum with its time index.

Listing 1: **convolution_sum** Function

```
function [y,k] = convolution_sum(nx, x, nh, h)

% input to the function
% nx is the index vector of x
% x is the first signal (input signal)
% nh is the index vector y
% h is the second signal (impulse response)

% output to the function
% y is the convoluted output signal
% k is the index vector of y

kmin = min(nx)+min(nh); % lowest time index of the convolution sum
                        ... is the sum of the lowest time indices of the two
                        ... signals to be convolved.
kmax = max(nx)+max(nh); % same as the kmin

k = kmin:kmax; % index vector k
y = conv(x,h); % convolution of x and h
end
```

Then we'll call that function to compute the convolution sum of any two discrete time signals. For example we'll compute the convolution sum of signals mentioned earlier: x[n] = {1,1,$\underset{\uparrow}{0}$,1,1} and h[n] = {1,-2,-3,$\underset{\uparrow}{4}$}.

Listing 2: Calculating convolution using the function **convolution_sum**

```
clc;
clear all;
close all;


```

```
x = [1, 1, 0, 1, 1]; %first signal
nx = [-2, -1, 0, 1, 2]; %time index of signal x

h = [1, -2, -3, 4]; % second signal
nh = [-3, -2, -1, 0]; % time index of signal h

% now call the function to compute convolution
[y, n] = convolution_sum(nx, x, nh, h);

%displaying the output signal
stem(n,y, 'Linewidth', 2);
```

Now we'll verify the properties of convolution. To verify **commutative property**, we'll take x[n] and h[n] again. First we'll compute $x[n] * h[n]$ and then calculate $h[n] * x[n]$. Plotting the two output results will verify that these two are equal i.e. $x[n] * h[n] = h[n] * x[n]$. Hence, convolution satisfies commutative property.

Listing 3: Verifying **commutative property**

```
clc;
close all;
clear all;

x = [1, 1, 0, 1, 1]; %first signal
nx = [-2, -1, 0, 1, 2]; %time index of signal x

h = [1, -2, -3, 4]; % second signal
nh = [-3, -2, -1, 0]; % time index of signal h

% now call the function to compute convolution x*h
[y, n] = convolution_sum(nx, x, nh, h);

% again call the function to compute h*x
[z, k] = convolution_sum(nh, h, nx, x);

%displaying the results
subplot(211)
stem(n, y, 'Linewidth', 2);
subplot(212)
stem(k, z, 'Linewidth', 2);
```

Alternatively you can calculate $y[n] - z[k]$ to see that the result will be a zero vector. We'll next verify the distributive property. Verification of associative property will be left as an exercise! We'll take three signals $x[n], h_1[n]$ and $h_2[n]$. First we'll compute $y[n] = x[n] * (h_1[n] + h_2[n]$ and then we'll compute $p[n] = x[n] * h_1[n] + x[n] * h_2[n]$. Finally we'll show that $y[n] = p[n]$. Here to add the signals we'll need **sigadd** function from Lab week 2.

Listing 4: **sigadd** Function

```
function [y,n] = sigadd(n1, x1, n2, x2)

% In order to add two signals, we need to
% make their length equal by padding zeros otherwise dimension mismatch
% will occur
%n1 = index vector of signal x1
%n2 = index vector of signal x2
%x1 = signal 1
%x2 = signal 2

n = min(min(n1),min(n2)):max(max(n1),max(max(n2))); % Defining the index
%vector for the output signal. For example if n1 = -3:2 and n2 = 0:4
% then n will run from -3 to 4. So we need to define 'n' from the minimum
% of individual minimum of n1 and n2. The same goes for the maximum.
y1 = zeros(1, length(n)); %initializing two vectors to store
y2 = y1;  ... the zero padded signals
%y1((find(n>=n1(1))&(n<=n1(end)))==1) = x1; % The idea is that if min(n1)<=n
   <=max(n1)
%y2((find(n>=n2(1))&(n<=n2(end)))==1) = x2; ... then y1 = x1(y2=x2).
y1 (((n>=min(n1)) & (n<=max(n1)))) = x1;
y2 (((n>=min(n2)) & (n<=max(n2)))) = x2;
y = y1+y2;

end
```

Listing 5: Verifying **distributive property** Function

```
clc;
clear all;
close all;

% verifying distributive law
% x*(h1+h2) = x*h1+x*h2

nx = -2:0; % index for x
x = [1,2,3]; % signal x
nh1 = -3:2; % index for h1
h1 = nh1.^2; % signal h1 = n^2
nh2 = 0:3; % index for h2
h2 = nh2+2; % signal h2 = n+2

% first calculate y = h1+h2 using sigadd function
[y, ny] = sigadd(nh1, h1, nh2, h2);

```

```
%now calculate z= x*y = x*(h1+h2)
[z,k] = convolution_sum(nx, x, ny, y);

% now we'll calculate x*h1+x*h2
% p1 = x*h1
[p1, np1] = convolution_sum(nx, x, nh1, h1);
% p2 = x*h2
[p2, np2] = convolution_sum(nx, x, nh2, h2);
% finally p = p1+p2 = x*h1+x*h2
[p, np] = sigadd(np1, p1, np2, p2);

%displaying the results to show z = p
subplot(211)
stem(k,z, Linewidth, 2);
subplot(212)
stem(np, p, Linewidth, 2);
```

Now we'll solve a difference equation using MATLAB/OCTAVE. While solving a difference equation we'll require initial conditions. However, as we're considering only causal linear systems, we'll assume that values of signal for negative time indices are zero. For example, consider a difference equation, $y[n] - y[n-1] + 0.9y[n-2] = x[n]$. Here to find impulse response $x[n] = \delta[n]$ and $y[-1] = y[-2] = 0$ etc. The code for solving this difference equation is given below:

Listing 6: Solving a difference equation to find the impulse response

```
clc;
clear all;
close all;

% y[n] - y[n-1]+0.9y[n-2] = x[n]
% for impulse response, h[n] - h[n-1]+0.9h[n-2] = delta[n]
% assuming causal system h[-1] = h[-2] = ... = 0
% we'll start index from i = 2, where h(1) = h[-2] = 0, h(2) = h[-1] = 0, h
    (3) = h[0] etc.

n = -20:100; % time index vector
x = (n==0); % delta[n]
h = zeros(1,length(n)); %defining

for i = 3:length(n)
    h(i) = x(i) + h(i-1) - 0.9*h(i-2);
end

stem(n,h);
```

The same solution can be obtained using MATLAB's built in function **filter**. The syntax for

filter is **filter(b,a,x)** where b is the vector containing coefficients of x[n], x[n-1], x[n-2] etc., a is the vector containing the coefficients of y[n]'s and x is the input sequence. Following code demonstrates the use of filter function.

Listing 7: Impulse response using **filter** function

```
clc;
clear all;
close all;


% y[n] - y[n-1] +0.9y[n-2] = x[n]
% for impulse response: h[n] - y[n-1] + 0.9y[n-2] = delta[n]

n = -20:100; % time index vector
b = [1]; % coefficent vector of x[n]/delta[n]
a = [1 -1 0.9]; % coefficient vector of y[n]/h[n]

x = (n==0); % defining the delta sequence

h = filter(b,a,x);

stem(n,h);
```

Thus the response of any system to any signal can be determined either by solving difference equation or by using filter function.
In the last part of this week's lab, we'll discuss about correlations. Recall that cross-correlation measures the similarity between a sequence x and shifted (lagged) copies of another sequence y as a function of the lag. For example, consider two signals x[n] and y[n] where y[n] = x[n-5]. If we calculate the cross-correlation of these two signals, we'll find that the largest spike occurs at the lag value where x[n] and y[n] match exactly and that is at lag 5 as y[n] is resulted from a shift in time by 5 samples.

Listing 8: Example of Cross-Correlation

```
clc;
clear all;
close all;

nx = 0:15; % defining time index

x = 0.05*nx.^2; %defining a signal n

% y = x[n-5]
n0 = 5;
[y, ny] = sigshift(x, nx, n0);

% calculating cross-correlation using convolution
```

```
% rxy = y[n]*x[-n]

[x, nx] = sigfold(x, nx);

[rxy, nxy] = convolution_sum(nx, x, ny, y);
stem(nxy, rxy);
```

The output is shown in figure 3.3. From the figure, we see that the largest spike indeed occurs when the lag is 5.

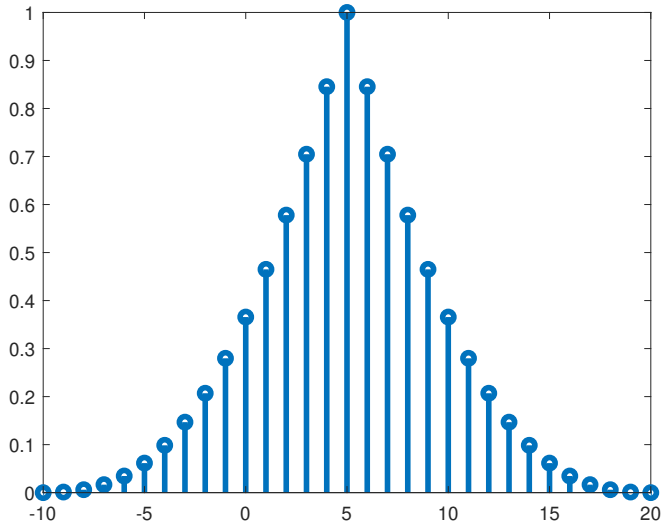


Figure. 3.3: Output of the cross-correlation of x[n] and y[n] = x[n-5]

In the above example, we calculated the cross-correlation using convolution. MATLAB/OCTAVE has a built in function for this purpose. **xcorr(x,y)** computes the cross-correlation between two sequences given by x and y. Go through the documentation of xcorr to know the details of this. In this case, you've to provide the time index vector to show the output correctly. Autocorrelation can also be computed using **xcorr**. Just make y = x in the argument of the function. In the following example, we'll compute the auto-correlation of a sin function.

Listing 9: Example of Auto-Correlation

```
clc;
clear all;
close all;

n = -20:20; %defining the time index vector
x = sin(pi/5*n); %the signal

z = xcorr(x); %auto-correlation of x
```

```
index = min(n)+min(n):max(n)+max(n); %corresponding time index vector of z
plot(index, z/max(z), 'Linewidth', 3); % showing normalized output
```

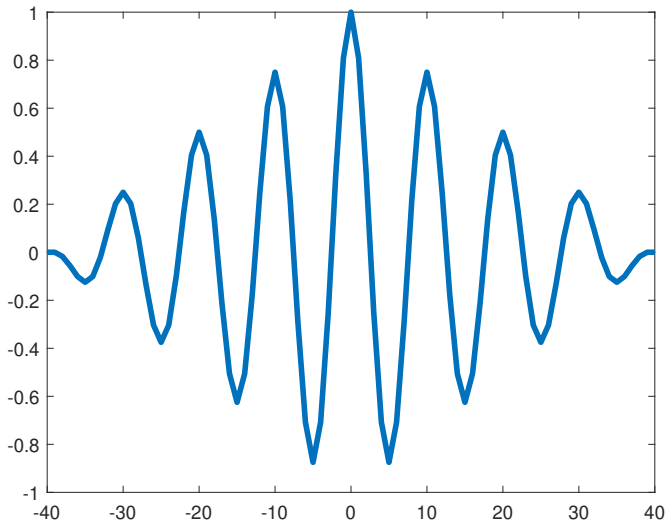


Figure. 3.4: Output of the auto-correlation of a sine wave.

Similar to the cross-correlation case, the peak occurs at zero lag.

Post Lab Tasks

- **Problem 1**: Verify the **Associative Property** of Convolution using **convolution_sum** function.
- **Problem 2**: A causal, LTI system is described by the second-order difference equation: **y[n]-4y[n-1]+4y[n-2]=x[n]-x[n-1]**. Determine impulse response of the system i) by solving the difference equation and ii) by using **filter** function. Show the output for n = -20:25. Show the output of the system if the input is $\boldsymbol{x[n] = (5 + 3\cos 0.2\pi n + 4\sin 0.6\pi n)u[n]}$.
- **Problem 4**: Consider an audio signal $\boldsymbol{x[n] = \cos 0.2\pi n + 0.5\cos 0.6\pi n}$. If the signal is reflected from a barrier and reaches the listeners, they will experience the signal as: $\boldsymbol{y[n] = x[n] + 0.1x[n - 20]}$. Generate 200 samples of y[n] and show its autocorrelation. Can you find the delay between x[n] and y[n]?
- **Problem 5**: Given two signals, x[n] = $\{\underset{\uparrow}{1},2,1,1\}$ and y[n] = $\{3, \underset{\uparrow}{5}, 8, 13, 21\}$. Using these two signals show that cross-correlation is non-commutative. Show the corresponding cross-correlation output figures.

LAB 4:
Sampling

Objectives

The main objectives of this lab are:

- To understand the concept of Sampling and aliasing.
- To study the effect of different sample rates and verify **Sampling Theorem**
- To understand the concept of Upsampling and Downsampling

PART 1

Most signals encountered in real life are analog in nature such as speech, biological signals (ECG, PPG), communication signals etc. These signals must be processed for various purposes. In a previous lab, it was mentioned that Digital Signal Processing of an analog signal offers numerous advantages over analog processing such as, better accuracy, better storing capability, cost efficiency and so on. Hence, in order to process these analog signals, the signals must be converted into digital domain. Converting an analog signal into digital one requires the process of sampling, quantization and encoding. In this lab, we'll discuss various concepts and issues involved in sampling of a continuous-time signal.

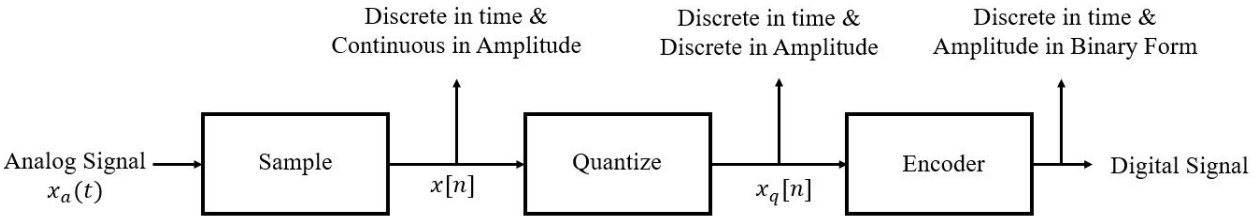


Figure. 4.1: Analog to Digital Converter

Sampling

Sampling is the process of converting a continuous-time signal into a discrete time one. It is the first step in conversion of an analog signal to digital domain. One of the most popular sampling methods is *periodic sampling*. In the periodic sampling operation, exact values of the continuous-time signal at uniformly spaced discrete intervals (nT_s), where T_s is sampling period and its reciprocal, $f_s = \frac{1}{T_s}$ is the sampling frequency, are retained.
It is convenient to represent the sampling process mathematically in the two stages. The stages consist of multiplying (modulating) the continuous time signal by an impulse train (a series of impulses) followed by conversion of the impulse train to a sequence. The periodic

impulse train is

$$s(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT_s) \tag{1}$$

where $\delta(t)$ is the Delta function. The product of *s(t)* and continuous signal, $x_c(t)$ is:

$$x_s(t) = x_c(t)s(t) \tag{2}$$

Figure 4.2 shows the process of sampling using impulse train modulation. Note that, $x_s(t)$ is still, in a sense, a continuous signal which is zero, except at integer multiples of T_s. On the other hand, x[n] is indexed on the integer variable n.

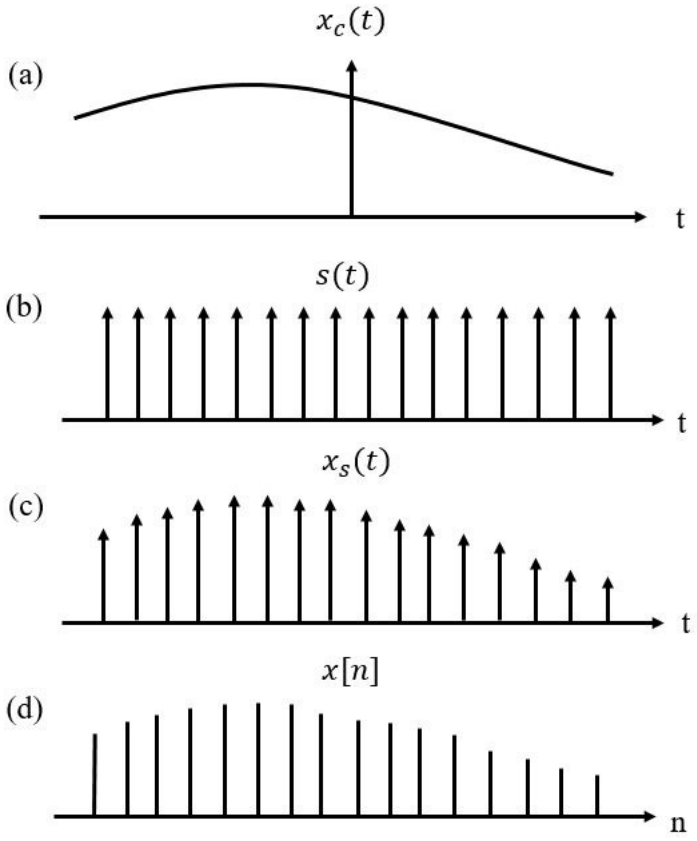


Figure. 4.2: (a) Continuous-time Signal, $x_c(t)$ (b) Impulse Train, s(t) (c) Signal, $x_s(t) = x_c(t)s(t)$ (d) Discrete-time signal obtained from sampling.

In general, the sampling operation is not invertible, i.e. given the output x[n], it is not possible in general to reconstruct $x_c(t)$, the input to the sampler, since many continous-time signals can produce the same output sequences of samples. However, if a continuous time signal is to be uniquely represented and recovered from its samples, then the signal must be band-limited. A band-limited signal is one whose Fourier Transform is non-zero on only a finite interval of the frequency axis. Figure 4.3 shows a band-limited signal.

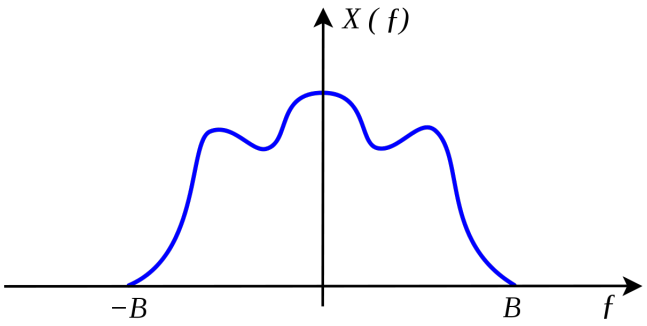


Figure. 4.2: Spectrum of a Band-limited Signal. Note that the Fourier Transform is non zero only for frequency |f|≤B.

Further we have to realize that the samples must be sufficiently close and the Sampling Rate must bear certain relation with the highest frequency component of the original signal. The **sampling theorem** also known as **Nyquist-Shannon Sampling Theorem** gives us the minimum sampling frequency for a continuous-time signal so that it can be uniquely reconstructed from its samples. If the sampling frequency is less than this minimum value, a phenomenon known as **aliasing** occurs.

Sampling Theorem:

Let $x_c(t)$ be a bandlimited signal with

$$X_c(\mathrm{f}) = 0 \text{ for } |\mathrm{f}| \geq \mathrm{f}_N.$$

Then $x_c(t)$ is uniquely determined by its samples $x[n] = x_c(nT_s), \mathrm{n} = 0, \pm 1, \pm 2....$, if the sampling frequency

$$\mathrm{f}_s = \frac{2\pi}{T_s} \geq 2f_N.$$

The frequency f_N is commonly referred to as the *Nyquist frequency*, and the frequency $2\mathrm{f_N}$ as the *Nyquist rate.*

For example, in case of a continuous time sinusoidal signal, $x_c(t) = \cos 4000\pi t$, signal frequency is 2000Hz. So in order to reconstruct the signal uniquely from its samples, the sampling frequency should be at least 4000Hz. Note that, $\mathrm{f_N}$ mentioned in the sampling theory is the maximum frequency content of the signal. Now let's see what happens if we sample a signal below the corresponding Nyquist rate. To discuss this, we need to consider the signal in frequency domain.

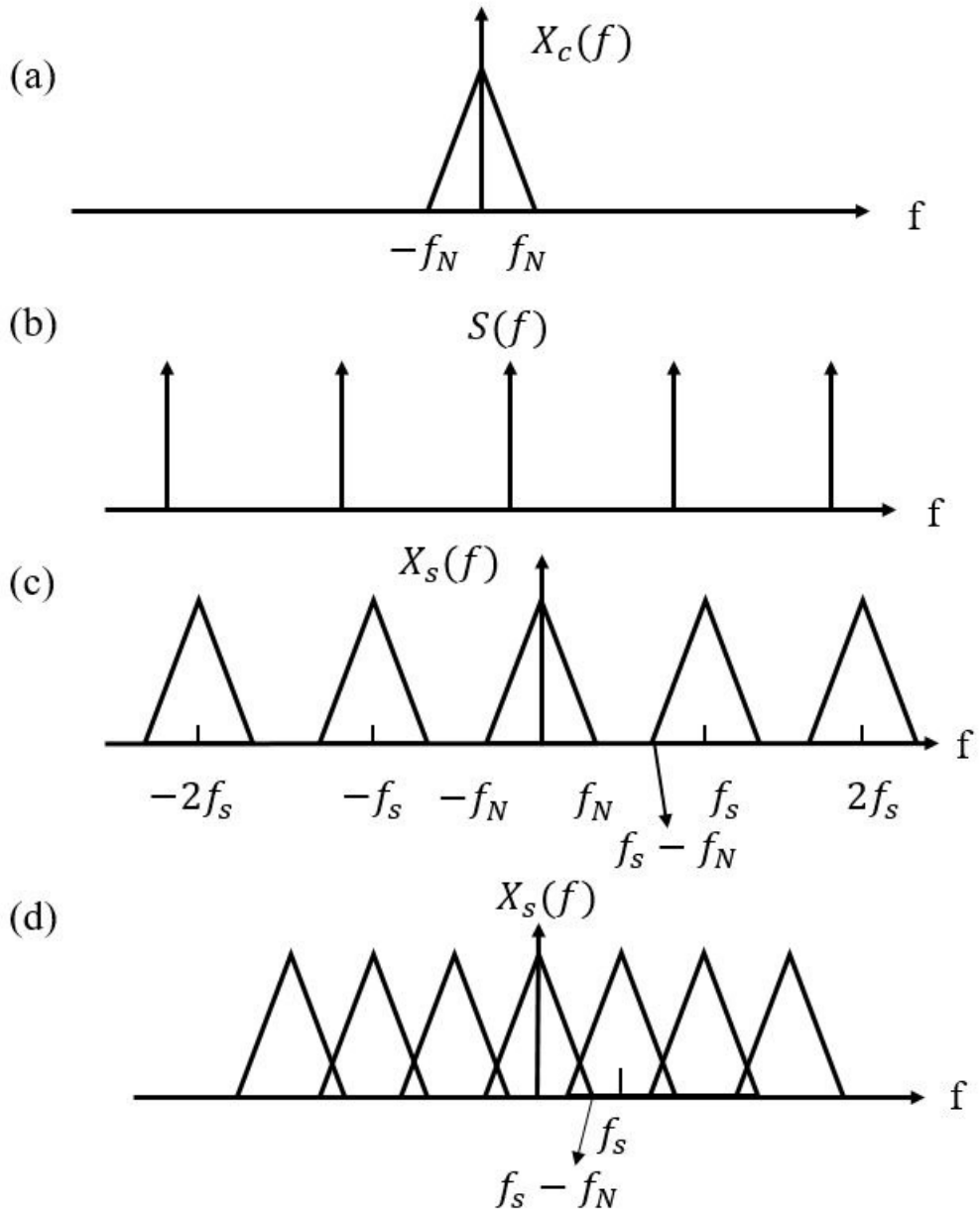


Figure. 4.3: Frequency-domain representation of sampling in the time domain. (a) Spectrum of the original signal. (b) Fourier Transform of the sampling function. (c) Fourier Transform of the sampled signal with $f_s > 2f_N$. (d) Fourier Transform of the sampled signal with $f_s < 2f_N$

Figure 4.3 shows the frequency-domain representation of the impulse train sampling. Figure 4.3(c) shows the sampling of $x_c(t)$ with sampling frequency $f_s > 2f_N$. Here the replicas of $X_c(f)$ do not overlap and therefore, $x_c(t)$ can be recovered from $x_s(t)$ with an ideal low pass filter with cut-off frequency (f_c) such that $f_N \leq f_c \leq (f_s - f_N)$. However, from figure 4.3(d) shows the case where the sampling frequency $f_s < 2f_N$. Here the copies of $X_c(f)$ overlap, so that when they are added together, $X_c(f)$ is no longer recoverable by low-pass filtering. In this case, the reconstructed output is related to the original continuous-time

input through a distortion which is known as **aliasing**. So aliasing is an unwanted case of sampling, where minimum condition for accurate sampling is not met. Physically, in aliasing, signals of different frequencies become indistinguishable from each other. The high-frequency components of the signal spectrum take the identity of the lower frequencies in the spectrum of the sampled signal. Aliasing can be avoided by sampling the signal at a higher rate than the Nyquist rate or by using an anti-aliasing filter which is essentially a low pass filter.

Sample Rate Conversion: Upsampling and Downsampling

In many practical applications of digital signal processing, one is faced with the problem of changing the sampling rate of a signal, either increasing or decreasing it by some amount. For example, in telecommunication system that transmit and recieve different types of signals, there is a requirement to process these various signals at different rates according to the bandwidth of the signals. The process of converting a signal from a given rate to a different rate is called **sample rate conversion**. Increasing the sampling rate of an already sampled signal is known as upsampling and decreasing the sampling rate is known as downsampling.

Upsampling

The purpose of upsampling is to manipulate a signal in order to artificially increase the sampling rate to avoid aliasing. Upsampling a signal by an integral factor L means inserting (L-1) zeros between successive sampled data of original signal. The input-output relation of an upsampler system is given by:

$$y[n] = \begin{cases} x\left[\frac{n}{L}\right], n = 0, \pm L, \pm 2L, \ldots \\ 0, \text{ otherwise} \end{cases} \tag{3}$$

Downsampling

The operation of reducing the sampling rate is called downsampling. The sampling rate of a sequence can be reduced by sampling it i.e. by defining a new sequence:

$$x_d[n] = x[nM] = x_c(nMT_s).$$

Downsampling a signal by a factor L means that we'll take one sample from each group of 'L' samples. For example, y[n] = x[2n] represents downsampling x[n] by a factor of 2. Here we'll discard the odd-numbered samples in x[n] and retain the even-numbered samples. In other words, we'll keep one sample from every successive two samples. Figure 4.4 shows the representation of an upsampler and downsampler.

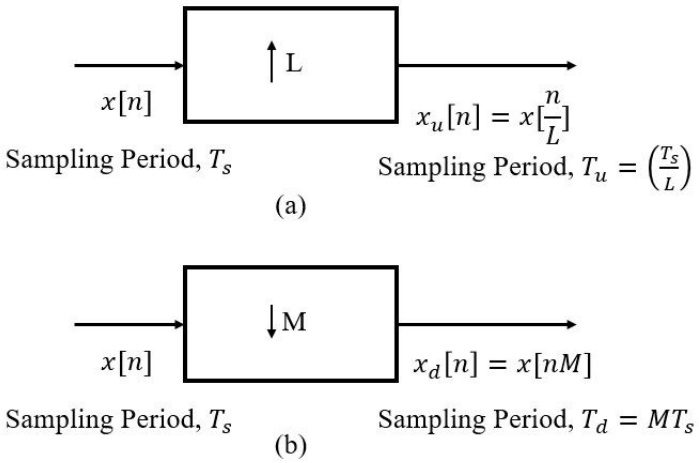


Figure. 4.4: (a) General System of Upsampling a signal by L (b) General system for downsampling a signal by M

PRELAB TASKS

- Given a continuous time signal $x_c(t) = \sin(2000\pi t) + \cos(4000\pi t) + \sin(5000\pi t)$, what is the minimum sampling frequency in Hz that satisfies Nyquist Criteria.
- If a signal $x_c(t) = \cos(8000\pi t)$ is sampled at a frequency 2000Hz, draw the spectrum of the i) original signal and ii) sampled signal. Write down the frequency clearly in the spectrum.
- Let x[n] = {1,1,3,4,5,6,$\underset{\uparrow}{8}$,1,1,3,4} be a discrete-time version of some continuous signal. Sketch the signal x[2n] and x[$\frac{n}{3}$].

Part 2
Sampling using MATLAB/OCTAVE

- Sampling and aliasing in MATLAB/OCTAVE
- Upsampling and Downsampling in MATLAB/OCTAVE
- Reconstruction of a sampled signal in MATLAB/OCTAVE
- Functions to use: **stem(), plot(), subplot()**

Instructions

- Organizing files and folder properly for later use.
- Make a unique named folder for this lab like **EEE3218_SP20_A1_180105001** under any drive other than C drive.
- For every week make a subfolder to keep the all the tasks in a specific week separated from other tasks.
- Always try to work in the folder specifically created for the current week.
- Follow MATLAB/OCTAVE naming convention for giving a meaningful name before saving it in MATLAB/OCTAVE.
- While running a code be judicious in choosing *add to path* or *change directory*. Use add to path when you are using a file from different folder.

In this part of our lab, we'll dive into the concept and issues of sampling and aliasing using MATLAB/OCTAVE. Then we'll show upsampling and downsampling of a signal. Finally we'll show how to reconstruct a signal from its samples. Let us consider a continuous-time signal $x_c(t) = 10\cos(200\pi t)$. Frequency of the signal is $F_0 = 100$ Hz and time period is $T_0 = \frac{1}{100} = 0.01$s. So the corresponding nyquist rate is 200Hz. At first, we'll sample the signal at a rate of 800Hz frequency which is much higher than nyquist rate. From figure 4.5, it is seen that the original signal can be uniquely identified from its samples. The code for this is given below:

Listing 1: Sampling a continuous time signal with sampling frequency 800Hz

```
1 clc;
2 clear all;
```

```
close all;
% We'll sample a signal at a frequency 500 Hz
% xc = 10*cos(200*pi*t)
% Signal Frequency, Fo = 100 and Signal Period, To = 1/100
% Sampling Frequency, Fs = 500 and Sampling Period, Ts = 1/500
Fo = 100; % Signal Frequency
To = 1/Fo; % Signal Period
Fs = 500; % Sampling Frequency which is greater than Nyquist Rate
Ts = 1/Fs;

t = 0:To/100:3*To; % Time axis for continuous signal
xc = 10*cos(200*pi*t); % continuous signal

t1 = 0:Ts:3*To; % Sampled time axis
xs = 10*cos(200*pi*t1); % Sampled Signal

subplot(211)
plot(t, xc, 'Linewidth', 2); % plotting continuous-time signal
hold on
stem(t1, xs, 'Linewidth', 2); % showing the sampled values in the same plot
subplot(212)
stem(t1, xs, 'Linewidth', 2); % showing the sampled values in a separate plot
```

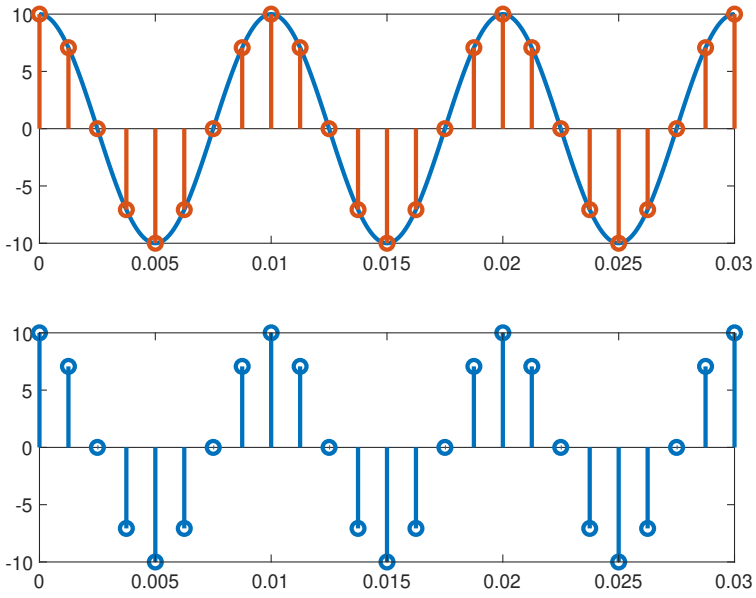


Figure. 4.5: (a) Continuous-time signal and its samples obtained by sampling at a rate of 800Hz (b) Samples of the continuous-time signal in (a)

From the samples of signal $x_c(t)$ the discrete time signal x[n] can be obtained by dividing the time axis of the sampled signal i.e. figure 4.5(b) by the sampling period T_s.

Now let's sample the signal $x_c(t)$ at a sampling frequency equal to the nyquist rate i.e. 200Hz and the corresponding output is shown in figure 4.6. It is seen from the figure that we can recognize the signal from its samples although this is more difficult than the previous case where the sampling rate is higher than this. However, the trend in the signal's behaviour can be recognized from its samples. In fact, the samples only capture the extremes of each period of the cosine oscillation. This is the significance of of `twice the highest frequency of the signal` value for sampling frequency. We can say that, the samples are just enough to capture the signal's oscillation if it is sampled exactly at the nyquist rate.

Listing 2: Sampling a continuous time signal with sampling frequency 200Hz

```
clc;
clear all;
close all;

% We'll sample a signal at a frequency 500 Hz
% xc = 10*cos(200*pi*t)
% Signal Frequency, Fo = 100 and Signal Period, To = 1/100
% Sampling Frequency, Fs = 500 and Sampling Period, Ts = 1/500

Fo = 100; % Signal Frequency
To = 1/Fo; % Signal Period
Fs = 200; % Sampling Frequency which is greater than Nyquist Rate
Ts = 1/Fs;

t = 0:To/100:3*To; % Time axis for continuous signal
xc = 10*cos(200*pi*t); % continuous signal

t1 = 0:Ts:3*To; % Sampled time axis
xs = 10*cos(200*pi*t1); % Sampled Signal

subplot(211)
plot(t, xc, 'Linewidth', 2); % plotting continuous-time signal
hold on

stem(t1, xs, 'Linewidth', 2); % showing the sampled values in the same plot

subplot(212)
stem(t1, xs, 'Linewidth', 2); % showing the sampled values in a separate plot
```

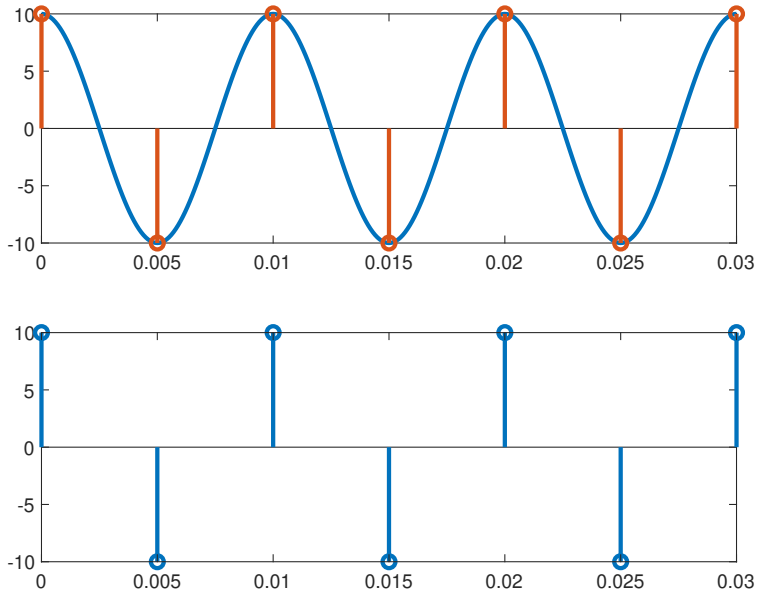


Figure. 4.6: (a) Continuous-time signal and its sample obtained by sampling at a rate of 200 Hz (b) Samples of the continuous-time signal in (a)

Finally we'll sample the signal at a frequency lower than the nyquist rate and we'll observe aliasing. If the sample frequency drops too low, the samples seem to come from a different signal. For example, if the signal $x_c(t) = 10\cos(200\pi t)$ is sampled at a frequency 150Hz, the resulting discrete time signal is

$$\begin{aligned} \text{x[n]} &= x(nT_s) = 10\cos(200\pi n T_s) \\ &= 10\cos\left(200\pi n \frac{1}{150}\right) \\ &= 10\cos\left(\frac{4\pi n}{3}\right) = 10\cos\left(\frac{2\pi n}{3}\right) = 10\cos\left(2\pi\frac{n}{3}\right) \end{aligned}$$

So the sampled signal will be an alias of another signal $x_{ac}(t) = 10\cos(100\pi t)$. Let us verify the result from MATLAB Figures.

Listing 3: Aliasing occurs when the signal is sampled below the Nyquist Rate

```
clc;
clear all;
close all;

% We'll sample a signal at a frequency 500 Hz
% xc = 10*cos(200*pi*t)
% Signal Frequency, Fo = 100 and Signal Period, To = 1/100
% Sampling Frequency, Fs = 500 and Sampling Period, Ts = 1/500

Fo = 100; % Signal Frequency
```

```
To = 1/Fo; % Signal Period
Fo1 = 50; % Aliased Frequency
To1 = 1/Fo1; % Aliased signal period
Fs = 150; % Sampling Frequency which is greater than Nyquist Rate
Ts = 1/Fs;

%% continuous signal
t = 0:To/100:3*To; % Time axis for continuous signal
xc = 10*cos(200*pi*t); % continuous signal

%% sampled signal
t1 = 0:Ts:3*To; % Sampled time axis
xs = 10*cos(200*pi*t1); % Sampled Signal

%% aliased continuous signal
t2 = 0:To1/100:3*To; % Time axis for aliased signal
xc2 = 10*cos(100*pi*t2); % Aliased Signal

%% plotting all the signals together
plot(t, xc, 'Linewidth', 2); % plotting continuous-time signal
hold on
plot(t2, xc2,'g', 'Linewidth', 2); % plotting the aliased signal
stem(t1, xs, 'r','filled', 'Linewidth', 2); % showing the sampled values in
   the same plot
```

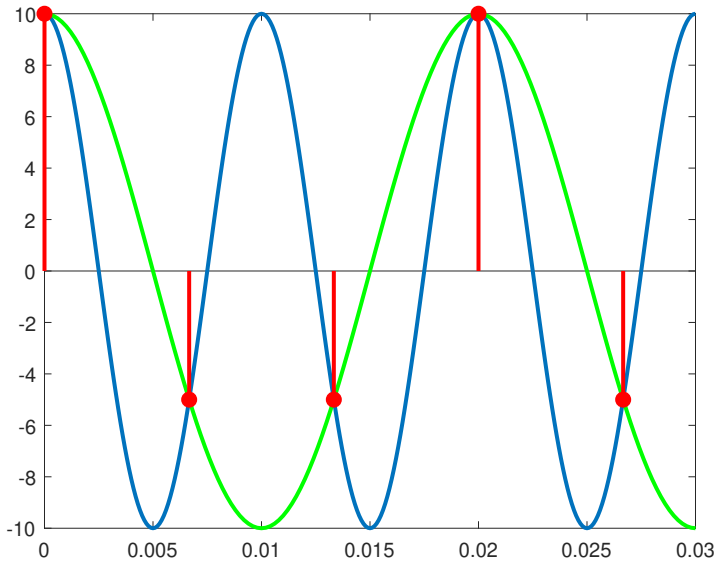


Figure. 4.7: Both the signal $x_c(t) = 10\cos(200\pi t)$ and $x_{c2}(t) = 10 * cos(100\pi t)$ go through the samples when $x_c(t)$ is sampled below the Nyquist Rate. So $x_c(t)$ and x_{c2} are alias signals.

So we see that when a signal is sampled below the Nyquist rate, it cannot be reconstructed uniquely as there exists another signal with those samples.
Now we'll write two functions that will perform upsampling by a factor L and downsampling by a factor M on an already sampled signal.

Listing 4: Function to perform upsampling

```
function [xup, nup] = upsampling(x, n, L)

% L is the upsampling factor
% x is the original sampled signal
% n is the original index vector

nup = n(1)*L:n(end)*L; % Upsampled time index
xup = zeros(1, length(nup)); % initializing the upsampled signal
xup(1:L:length(nup)) = x; % Insert L-1 zeros between two successive samples.

end
```

Listing 5: Function to perform downsampling

```
function [xdown, ndown] = downsampling(x, n, M)

% M downsampling factor
% x input sampled signal
% n is the index vector of original sampled signal

pos = (find(mod(n,M) == 0)); % Finding the position of the index which is
                                ... the multiple of M
xdown = x(pos); % taking samples at the indices which is a multiple of M
                ... i.e. take one sample from every M samples.
ndown = n(pos)/M; % downsampled time index vector
end
```

] Let's consider an already sampled discrete signal $x[n] = \sin(0.7n)$ defined for $-10 \leq n \leq 10$. We'll upsample x[n] by a factor of 3 and downsample it by a factor of 2. Let's see what happens.

Listing 6: Upsample a signal by a factor of 3 and downsample it by a factor of 2

```
close all;
clear all;
clc;

n=-10:10; % index vector of original signal
x=sin(0.7*n); % original samples
subplot(311),
stem(n,x, 'Linewidth', 2);
```

```
title('Original Signal')

%% Up-sampling
L=3;            %Scaling factor
[xup, nup] = upsampling(x, n, L);
subplot(312)
stem(nup,xup, 'Linewidth',2);
title(['Up-sampled Signal by ',num2str(L)]);

%% Down-sampling
M = 2;
[xdown, ndown] = downsampling(x,n,M);
subplot(313)
stem(ndown, xdown, 'Linewidth', 2);
title(['Down-sampled Signal by ',num2str(M)]);
```

Figure 4.8 shows the output of this code.

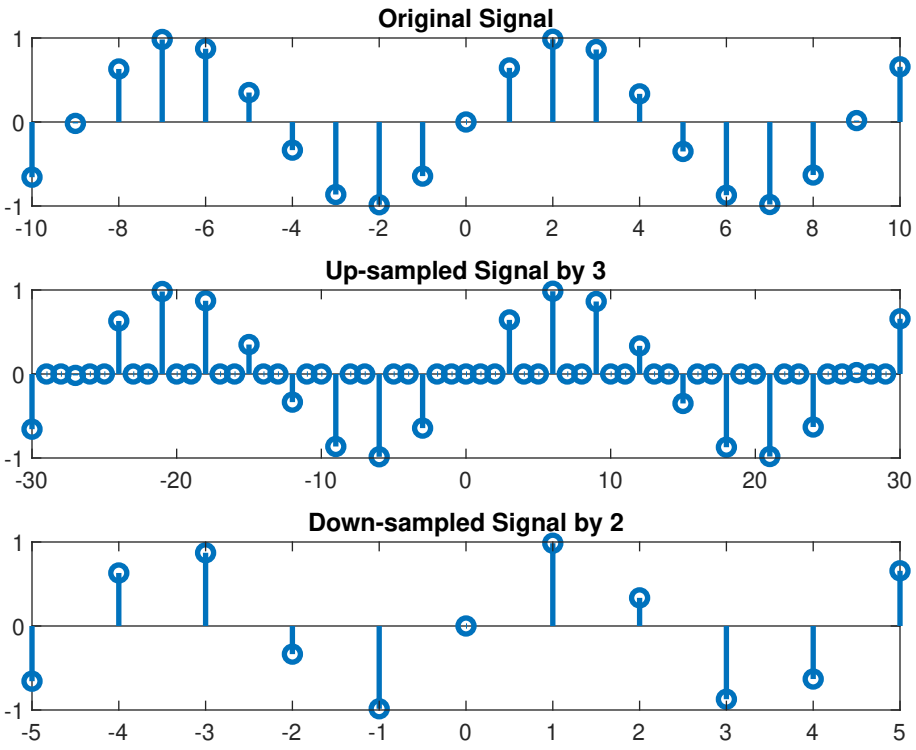


Figure. 4.8: (a) Original Sample (b) Upsampled signal by a factor of 3 (c) Downsampled signal by a factor of 2.

From figure 4.8 we see that when the signal is upsampled by a factor of 3, two zeros are inserted between successive samples and the total number of samples gets increased. On the

contrary, when the signal is downsampled by a factor of 2, one sample from two successive samples are taken and shown in 4.8(c) i.e. the original sampling frequency is halved. In other words, here we've taken only the even numbered samples and discarded the odd-numbered ones. This is how the rate of sampling can be converted by upsampling or downsampling.
For the last topic of our lab week 4, we'll see how to reconstruct a signal from its samples. For this, we need a bit of theoretical background. In order to reconstruct a signal from its samples, we need to pass the samples through a low pass filter whose cut-off frequency(f_c) satisfies the condition: $f_N \leq f_c \leq (f_s - f_N)$. In time domain this implies multiplying the samples by an interpolation function or sinc function. In other word, if we consider low pass filter (LPF) a system, the output of the system will be the convolution sum of the samples and the impulse response ($h_r(t)$) of the LPF. $h_r(t)$ for an LPF with cutoff frequency $\frac{\pi}{T_s}$ is given by:

$$h_r(t) = \frac{\sin\left(\frac{\pi t}{T_s}\right)}{\frac{\pi t}{T_s}} \tag{4}$$

This impulse response is shown in figure (4.9).

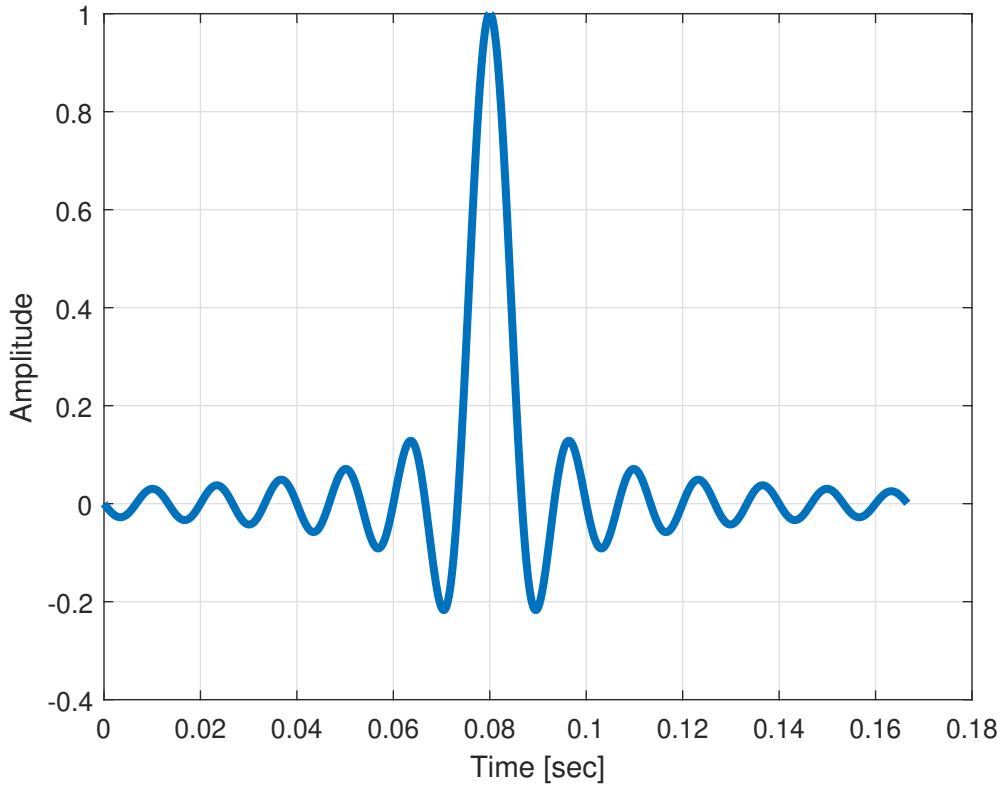


Figure. 4.9: Impulse Response of an ideal Low Pass Filter.

The final reconstructed signal $x_r(t)$ which is the convolution sum of $x[n]$ and $h_r(t)$ is given

by the formula:

$$x_r(t) = \sum_{n=-\infty}^{\infty} x[n] \frac{\sin\left(\frac{\pi(t-nT_s)}{T_s}\right)}{\frac{\pi(t-nT_s)}{T_s}} \tag{5}$$

We'll use equation (5) to reconstruct a signal from its samples. Here we're considering sampling frequency higher than the nyquist rate so that we can properly reconstruct the original signal.

Listing 7: Reconstruction of a signal from its samples

```
clc;
close all;
clear all;


% First we'll sample the signal at a higher rate than
% Nyquist Rate
% xc(t) = 5*cos(200*pi*t)
% Signal Frequency, Fo = 100 and period, To = 1/100
% Sampling Frequency, Fs = 800 and period, Ts = 1/800

Fo = 100; % signal frequency
To = 1/Fo; % signal period
Fs = 800; % sampling frequency
Ts = 1/Fs; % sampling period

tc = 0:To/100:3*To; % defining time axis for xc upto 3 cycles
xc = 5*cos(200*pi*tc); % constructing the continuous-time signal

ts = 0:Ts:3*To; % sampled the time axis a rate Fs
xs = 5*cos(200*pi*ts); % sampled signal at a frequency Fs
N = length(ts); % number of samples

%% Reconstruction by using the formula:
% xr(t) = sum over n = 0...., N-1 : x(nT)*sin(pi*(t-nT)/T)/(pi*(t-nT)/T)
% Here sin(pi*(t-nT)/T)/(pi*(t-nT)/T) = sinc((pi*(t-nT))/T)

xr = zeros(1, length(tc)); % initialize the reconstructed signal which is the
                           ... same length as
                              continuous-time
                              axis
sinc_fun = zeros(N, length(tc)); % initializing the interpolation function or
    sinc function

for t = 1:length(tc)
        for n = 0:N-1
```

```
            %this line computes the convolution of xr and sinc_fun
            xr(t) = xr(t) + xs(n+1)*sin(pi*(tc(t)-n*Ts)/Ts)/(pi*(tc(t)-n*Ts)/
                Ts);

end

end


%% plot the results

plot(tc, xc);
hold on
stem(ts, xs);
hold on
plot(tc, xr);
```

If you run the above code, you can see the original signal, its samples and reconstructed signal in a single plot.

POST LAB TASKS

1. Consider $\mathbf{x(t) = 10\cos(120\pi t) + 5\sin(100\pi t + 30^\circ) + 4\sin(150\pi t + 45^\circ)}$. Sample the signal for i) $F_s = 4 * F_o$, ii) $F_s = 2 * F_o$ and iii) $F_s = F_0$ where F_o is the maximum frequency content of x(t). Show the original signal and its samples in one figure for 3 cycles.

2. Consider a signal $\mathbf{x(t) = 10\cos(250\pi t + 60^\circ) + 5\sin(200\pi t + 75^\circ)}$. Convert the signal into a sequence x[n] for $-10 \leq n \leq 10$ by choosing an appropriate sampling frequency. Upsample x[n] by a factor of 3 and downsample the signal by a factor of 4. Show i) original signal, ii) sampled signal, iii) discrete signal, iv) upsampled signal and v) final output.

3. Consider $\mathbf{x(t)} = \frac{1}{2}\sin(\mathbf{14}\pi\mathbf{t}) + \frac{1}{3}\sin(\mathbf{18}\pi\mathbf{t}) + \frac{1}{5}\sin(\mathbf{24}\pi\mathbf{t}) + \frac{1}{7}\sin(\mathbf{30}\pi\mathbf{t})$. Sample the signal on the interval $0 \leq t \leq 2$ at i) $F_s = 5F_o$ and ii) $F_s = 1.5F_o$. Where F_o is the maximum frequency content of x(t). Reconstruct the signal in each case from its samples. What is the alias signal's time domain expression in case (ii).

4. There is another way to reconstruct the original signal by using *interpolation method.* MATLAB has a built-in function `interp1()` for this purpose. Write a MATLAB Code using this function to reconstruct the signal of problem 3.

LAB 5:
Frequency Analysis of Discrete-time Signals

Objectives

The main objectives of this lab are:

- To be familiar with the concept of Discrete time fourier transform (DTFT) and discrete fourier transform (DFT)
- To understand signal processing in frequency domain

PART 1

The Fourier Transform is one of the several mathematical tool that is useful in the analysis and design of LTI systems. Another is the Fourier Series. These are basically signal representations just like the unit impulse decomposition covered in a previous lab. However, in these representations, a signal is represented in terms of sinusoidal or complex exponential components. With such a decomposition, a signal is said to be represented in the **frequency domain**. For a class of periodic signal, this decomposition is called **Fourier Series** and in case of aperiodic signal, it is called **Fourier Transform**. These decompositions are very important in analyzing an LTI system because the response of an LTI system to sinusoidal input is also a sinusoid of the same frequency with different amplitude and phase. Furthermore, the linearity property of an LTI system implies that a linear sum of sinusoidal components at the input produces a similar sum of sinusoids at the output with different amplitudes and phases. In this lab, we will discuss how to transform a signal from its time domain to frequency domain representation and also see how an LTI system's output can be obtained in frequency domain.

Discrete Time Fourier Series

Consider a periodic sequence x[n] with period N, that is $x[n+N] = x[n]$ for all n. The Fourier Series representation for x[n] consists of N harmonically related exponential functions:

$$e^{\frac{j2\pi kn}{N}},\ \text{k} = 0,\ 1,\ 2,\ ....,N-1$$

And the series is expressed as:

$$x[n] = \sum_{k=0}^{N-1} c_k e^{\frac{j2\pi kn}{N}} \tag{1}$$

where $\{c_k\}$ are the coefficients in this representation. In deriving the expression for c_k's, the following property of complex exponentials are used:

$$\sum_{n=0}^{N-1} e^{j2\pi kn/N} = \begin{cases} N, & k = 0, \pm N, \pm 2N, \ldots \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

The Fourier series coefficient, c_l can be obtained by multiplying both sides of equation (1) by the exponential $e^{\frac{-j2\pi ln}{N}}$ and summing the product from n = 0 to N-1. Finally, we get

$$c_l = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{\frac{-j2\pi ln}{N}},\ l = 0, 1, ...N-1 \tag{3}$$

Equation (1) is called the synthesis and (2) is called the analysis equation. Note that $c_{k+N} = c_k$ that is, $\{c_k\}$ is a periodic sequence with fundamental period N.

Discrete Time Fourier Transform

The Fourier Transform of a finite-energy discrete time signal x[n] is defined as:

$$X(\omega) = \sum_{n=-\infty}^{n=\infty} x[n]e^{-j\omega n} \tag{4}$$

Physically, $X(\Omega)$ represents the frequency content of the signal x[n]. Here x[n] must be absolutely summable for the sum in equation (4) to converge.

$$\sum_{n=-\infty}^{\infty} |x[n]| < \infty \tag{5}$$

Indeed, equation (5) is the sufficient condition for the existence of the discrete-time Fourier transform. Now in equation (4), $X(\omega)$ is a periodic function with period 2π(Verify this!). This property is just a consequence of the fact that the frequency range for any discrete-time signal is limited to $(-\pi, \pi)$ or $(0, 2\pi)$. The spectrum $X(\omega)$ is, in general, a complex-valued function of frequency. It may be expressed as

$$X(\omega) = |X(\omega)|e^{j\Theta(\omega)} \tag{6}$$

where $|X(\omega)|$ is the magnitude spectrum and $\Theta(\omega) = \angle X(\omega)$ is the phase spectrum.
The inverse DTFT (IDTFT) equation is used to reconstruct the signal from its frequency spectrum $X(\omega)$ and given by:

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega)e^{j\omega n}d\omega \tag{7}$$

As $X(\omega)$ is periodic with period 2π, any interval of length 2π can be used to determine the integral given in equation(7).

Discrete Fourier Transform

Although DTFT gives the frequency component of a signal, it is still continuous over frequency which is unsuitable for DSP. In addition, an infinite number of time domain samples of the signal x[n] are required to compute DTFT which is also not feasible. In case of a finite

length sequence x[n], $0 \leq n \leq L-1$, only L samples of $X(\omega)$ over its period are sufficient to determine x[n] and hence $X(\omega)$. This leads to the concept of discrete Fourier Transform (DFT) which is obtained by periodic sampling of $X(\omega)$.
Although the L-point DFT is sufficient to uniquely represent the sequence x[n] in the frequency domain, however, it does not provide sufficient detail to yield a good picture of the spectral characteristics of x[n]. In order to obtain a better spectral picture, we often compute a higher point (N-point) DFT where $N > L$. This is called **N-point DFT**. While N can be any positive integer, **a power of two is usually chosen** i.e. 128, 256, 512, etc. The formulas for the Discrete-Fourier Transform(DFT) and Inverse Discrete Fourier Transform (IDFT) are

$$\text{DFT: } X[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}},\ \text{k} = 0,\ 1,\ 2,\ ....,N-1 \tag{8}$$

$$\text{IDFT: } x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{\frac{j2\pi kn}{N}}, \text{n} = 0,1,2....,N-1 \tag{9}$$

The formulas for DFT and IDFT given by equation (8) and 9 may be expressed as:

$$\text{DFT: } X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn},\ \text{k} = 0,\ 1,\ 2,\ ....,N-1 \tag{10}$$

$$\text{IDFT: } x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn}, \text{n} = 0,1,2....,N-1 \tag{11}$$

where, $W_N = e^{-\frac{j2\pi}{N}}$, which is an Nth root of unity. Equation 10 can be expressed in the matrix form as:

$$\mathbf{X_N} = \mathbf{W}_N \mathbf{x}_N \tag{12}$$

where x_N is the column matrix containing the elements of the sequence x[n] and X_N is the N-point vector containing the frequency samples. Mathematically x_N and X_N is given by:

$$x_N = \begin{bmatrix} x(0) \\ x(1) \\ . \\ . \\ . \\ x(N-1) \end{bmatrix}, X_N = \begin{bmatrix} X(0) \\ X(1) \\ . \\ . \\ . \\ X(N-1) \end{bmatrix}$$

So to match the dimension for matrix multiplication in equation (12), W_N must be an $N \times N$ matrix and given by:

$$\mathbf{W}_N = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & \cdots & W_N^{N-1} \\ & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \cdots : & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix} \tag{13}$$

The IDFT equation in equation(11) can also be represented by:

$$\mathbf{x}_N = \mathbf{W}_N^{-1}\mathbf{X}_N = \frac{\mathbf{1}}{\mathbf{N}}\mathbf{W}_N^{*}\mathbf{X}_N \tag{14}$$

where W_N^* denotes the complex conjugate of the matrix W_N.

Convolution Using DFT

In continuous time signal, multiplication of two signals in frequency domain implies convolution in time domain. However, in case of DFT, multiplication of two signals in frequency domain corresponds to **circular convolution** in time domain. Circular convolution of two DT signals or sequences $x_1[n]$ and $x_2[n]$ is defined by the following formula:

$$y[m] = \sum_{n=0}^{N-1} x_1[n]x_2([m-n])_N \text{ , m = 0, 1, ..., } N-1 \tag{15}$$

The basic difference between linear and circular convolution is that, in circular convolution, the folding and shifting(rotating) operations are performed in a circular fashion by computing the index of one of the sequences modulo N. In linear convolution, there is no modulo N operation. This will be discussed broadly in theory. However, linear convolution (the type of convolution sum we defined in Lab week 3) can also be computed by multiplying two signals in frequency domain. The steps for computing linear convolution, $y[n] = x[n] * h[n]$ using DFT is given below. We'll only mention the procedure. We won't go into the detailed theory.

1. Let N = length(x)+length(h)-1. Pad each of the original sequences, x[n] and h[n] with zeros until they are both length N.
2. Compute N-point DFT of each padded sequences, X[k] and H[k].
3. Compute Inverse DFT of the product Y[k] = X[k]H[k]. This will be equivalent to $y[n] = x[n] * h[n]$.

Fast Fourier Transform(FFT)

Fast Fourier Transform(FFT) is a method for computing DFT and IDFT efficiently. In MATLAB there's a built in function for computing FFT. We'll look into the syntax of FFT in the coding part of this Lab and learn about FFT algorithms in EEE 3217.

PRELAB TASKS

1. Determine the Discrete time Fourier Series spectrum of x[n] where x[n] is periodic with period N = 4 and x[n] = $\underset{\uparrow}{1}$,0,1,0. Draw the magnitude and phase spectra.

2. Compute and draw the magnitude and phase spectra of x[n] = u[n] - u[n-4].

3. Using DFT and Inverse DFT, compute the circular convolution of $x_1[n] = \{\underset{\uparrow}{1}, 2, 3\}$ and $x_2[n] = \{\underset{\uparrow}{4}, 3, 1\}$.

4. Using DFT and Inverse DFT, compute the linear convolution for the signals given in previous problem.

Part 2
Frequency Analysis of DT Signals using MATLAB/OCTAVE

- Computing DTFS, DTFT and DFT in MATLAB/OCTAVE
- Computing DFT using FFT in MATLAB/OCTAVE
- Frequency Analysis of a signal using DFT in MATLAB/OCTAVE
- Functions to use: **stem(), fft(), fftshift(), abs(), dftmtx(), angle(), ifft()**

Instructions

- Organizing files and folder properly for later use.
- Make a unique named folder for this lab like **EEE3218_SP20_A1_180105001** under any drive other than C drive.
- For every week make a subfolder to keep the all the tasks in a specific week separated from other tasks.
- Always try to work in the folder specifically created for the current week.
- Follow MATLAB/OCTAVE naming convention for giving a meaningful name before saving it in MATLAB/OCTAVE.
- While running a code be judicious in choosing *add to path* or *change directory*. Use add to path when you are using a file from different folder.

In this part of our lab, we'll implement the ideas covered in Lab in MATLAB to compute various types of frequency transformation and their spectra. However, after introducing the concept and syntax of fft we'll mostly use this function to compute DFT of a signal. So, the code for DTFS, DTFT and DFT are simply to give an idea how these things work. At first, we'll implement equation 3 to compute discrete time Fourier Series (DTFS) of a periodic signal. For this we'll first generate a periodic rectangular pulse and then compute its spectrum. The corresponding analysis and synthesis code is given below. dtfs_analysis function takes any periodic discrete-time signal and decomposes it into fourier series coefficients.

Listing 1: dtfs_analysis function to compute Discrete-time Fourier Series coefficient.

```
function [K, C] = dtfs_analysis(m, n, x)

% m = how many periods of the fourier series coefficient
% are to observed
% n = index vector of x
% x = periodic signal

N = length(x); % Total number of data points
Nc = m*N;  % Total no. of coefficients

if mod(Nc,2)== 0
    k = -Nc/2:Nc/2-1; % Determining the range of ck's
else
    k = -(Nc-1)/2:(Nc-1)/2;
end

C = zeros(1,length(k));       %Initializing the fourier coefficients

for i1 = 1:length(k) % this loop is for computing all the ck's
    for i2 = 1:length(x) % this loop is for the sum in equation (3)
        % ck = 1/N*sum(x[n]exp(-i2*pi*kn/N)) from n = 0 to N-1
        C(i1)= C(i1)+1/N*x(i2)*exp(-i*2*pi*k(i1)*n(i2)/N);
    end
end
K = k;
end
```

We'll use this function to decompose a periodic rectangular pulse with period 1ms into its fourier series coefficients. Finally we'll reconstruct the original signal using these coefficients.

Listing 2: Discrete-Time Fourier Series Analysis and synthesis of a rectangular periodic signal

```
clc;
clear all;
close all;

% Here we'll compute DTFS of a periodic rectangular pulse
% So, first we'll need to generate the rectangular pulse

Fs = 100e3;       %Sampling frequency
dt = 1/Fs;        %Sampling period

% Generating the rectangular pulse train

T = 1e-3;                    %Period of the pulse train
D = .1;                      %duty cycle
```

```
PW = D*T;                       %Pulse width
f = 1/T;                        %Analog frequency
t = -T/2:dt:T/2;                %Time interval for a period

n = t/dt;                       %Index for data points for a period
L = PW/dt;                      %Data points in the high time
x = zeros(1,length(t));         %Initializing a single rectangular pulse
x(find(abs(n)<=L/2))=1.1;    %Generation of a single rectangular pulse

% Visualizing the rectangular pulse
figure(1)
plot(t,x);
title('Original Continuous Rectangular Pulse');
xlabel('Time (seconds)');
ylabel('x(t)');
figure(2)
stem (n,x);
title('Sampled Rectangular Pulse');
xlabel('Index (n)');
ylabel('x(n)');

%% Fourier Series Analysis
N = length(x); % no. of data points
m = 1; % number of periods to be observed of cks

[k, c] = dtfs_analysis(m, n, x);

figure (3)
subplot(211)
stem(k, abs(c));
subplot(212)
stem(k, angle(c));

%% Reconstruction of signal from Fourier coefficients

t_remax = T; % Maximum time upto which the original...
                ... signal will be reconstructed
t_re = -t_remax:dt:t_remax; % time axis for reconstructed signal
n_re = t_re/dt; % index vector for the reconstructed signal

x_re = zeros(1, length(n_re)); % initializing the reconstructed signal

% To compute x_re, we need to compute the sum:
% x[n] = sum (ck*exp(j*2*pi*kn/N)) from k = 0 to N-1
% for each n
```

```
60
61 for i1 = 1:length(k)
62     for i2 = 1:length(x_re)
63         x_re(i2) = x_re(i2)+c(i1)*exp(1i*2*pi*k(i1)*n_re(i2)/N);
64     end
65 end
66
67 figure(4)
68 subplot(211)
69 plot(n_re, real(x_re));
70 subplot(212)
71 plot(t_re, real(x_re));
```

The output of the code is given in figure 5.1:

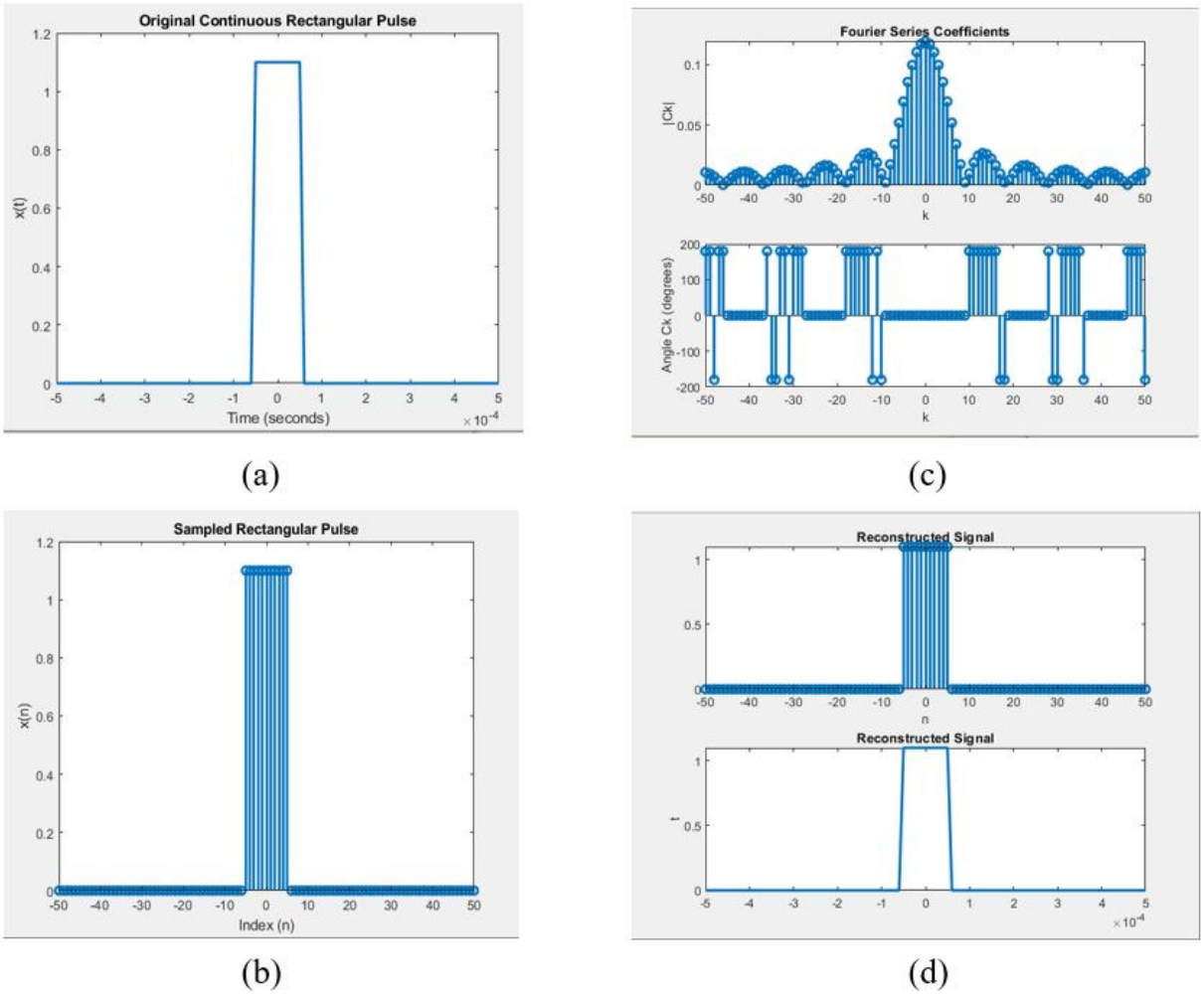


Figure. 5.1: (a) Original continuous Signal (b) Sampled Signal (c) Fourier Series coefficients (d) Reconstructed signal from its fourier series coefficients.

By varying **t_remax** in the code (reconstruction part) and observed that periodic repition is obtained. You can also observed the periodicity of $|c_k|$ by changing the total number of coefficients by varying **m** in the code(analysis part). Note that if $\mathbf{m} < 1$, the reconstructed signal will not be the same as original signal (Verify this!).
Now we'll look into the code to calculate DTFT of an aperiodic finite energy signal from first principle i.e by using equation(4). Here, x[n] = {1,2,3,4,5}. The code to compute DTFT from first principle is:

Listing 3: Discrete-Time Fourier Transfrom of an aperiodic signal

```
clc;
clear all;
close all;

n = -1:3; % time index for the signal
x = 1:5; % signal

M = 501; % Number of points in digital frequency grid
w = linspace(-pi, pi, M); % Defining digital frequency grid
X = zeros(1, length(w)); % Defining the spectrum vector of x

% To find out X, we need to carry out the sum in equation (4)
for i1=1:M
    for i2=1:length(x)
        X(i1)=X(i1)+x(i2)*exp(-1i*w(i1)*n(i2));
    end
end
```

The output spectrum of the signal is given below:

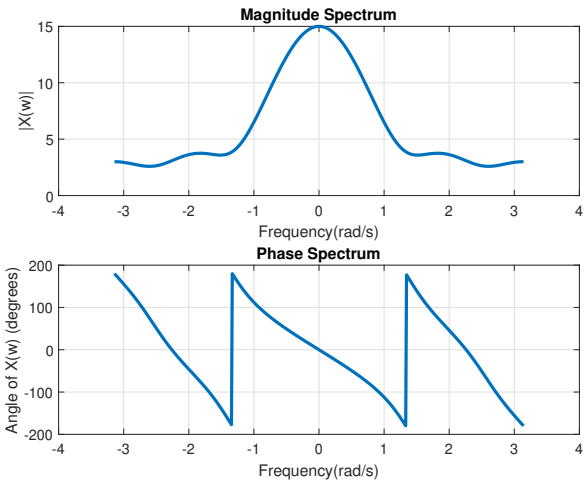


Figure. 5.2: Magnitude and Phase Spectrum of x[n] = {1,2,3,4,5}

As previously mentioned, DTFT is continuous in frequency, so DFT is mainly used to observe the frequency spectrum of any signal. First we'll compute DFT and IDFT from the matrix equation given in 12 and 14. Then we'll introduce the concept and syntax of Fast Fourier Transform (FFT) and we'll use FFT to compute discrete Fourier transform of a DT signal. The matrix W_N given in equation(12) can be generated using MATLAB function dftmtx(). However, we'll not use this function instead we'll generate the matrix from definition given in equation(13). We've defined two functions named **DFT** and **IDFT** to decompose a signal into its frequency components and then reconstructing it from its spectrum. While calculating DFT, we'll consider the case $N \geq L$ where L is the length of the sequence and N is the number of DFT points to avoid frequency domain aliasing.

Listing 4: **DFT** function to compute N-point Discrete Time Fourier Transform

```
function [Xk, k] = DFT(xn, N)
% computes N point DFT of xn
% Xk = DFT coefficient array over 0<=k<=N-1
% xn = N-point finite duration sequence
% N = length of DFT
n = 0:N-1; % index of data sequence
k = 0:N-1; % index of frequency sample

L = length(xn);
xn = [xn zeros(1, N-L)]; %  zero padding to make xn N point
WN = exp(-1j*2*pi/N); % WN = e^(-j2pi/N);
nk = n'*k; % creates a N x N matrix of nk values
WN_nk = WN.^nk; % Forming the WN_nk matrix given in equation (13)
Xk = xn*WN_nk; % Computes the matrix multiplication in equation (12)

end
```

Listing 5: **IDFT** function to reconstruct a signal from its frequency spectrum

```
function [xn, n] = IDFT(Xk, N)
% computes Inverse DFT from Xk
% Xk = DFT coefficient array over 0<=k<=N-1
% xn = N-point finite duration sequence over 0<=n<=N-1
% N = length of DFT
n = 0:N-1; % row vector for n
k = 0:N-1; % row vector for k

WN = exp(-1j*2*pi/N); % WN = e^(-j2pi/N);
nk = n'*k; % creates a N x N matrix of nk values
WN_nk = WN.^(-nk); % IDFT Matrix
xn = (Xk*WN_nk)/N; % Computes the matrix multiplication in equation (12)

end
```

Using these two functions, we'll compute the 5 point DFT of the sequence x[n] = {1,1,0,0,1} and the reconstruct the original sequence using IDFT.

Listing 6: 5 point DFT and reconstruction of a sequence using **DFT** and **IDFT** respectively.

```
clc;
clear all;
close all;

x = [1, 1, 0, 0, 1];
N = 5;
[X, k] = DFT(x, N);
MagX = abs(X); % Magnitude Spectrum
PhaseX = angle(X)*180/pi; % Showing angle in degree
figure(1)
subplot(211) stem(k, MagX, 'Linewidth', 2); xlabel('k'); ylabel('Magnitude
    Spectrum');
subplot(212) stem(k, PhaseX, 'Linewidth', 2); xlabel('k'); ylabel('Phase
    Spectrum');
% IDFT to obtain the original signal from its DFT
[xn, n] = IDFT(X, N);
figure(2)
stem(n, real(xn), 'Linewidth', 2); % showing only the real part of the idft
```

The output of the code is given below:

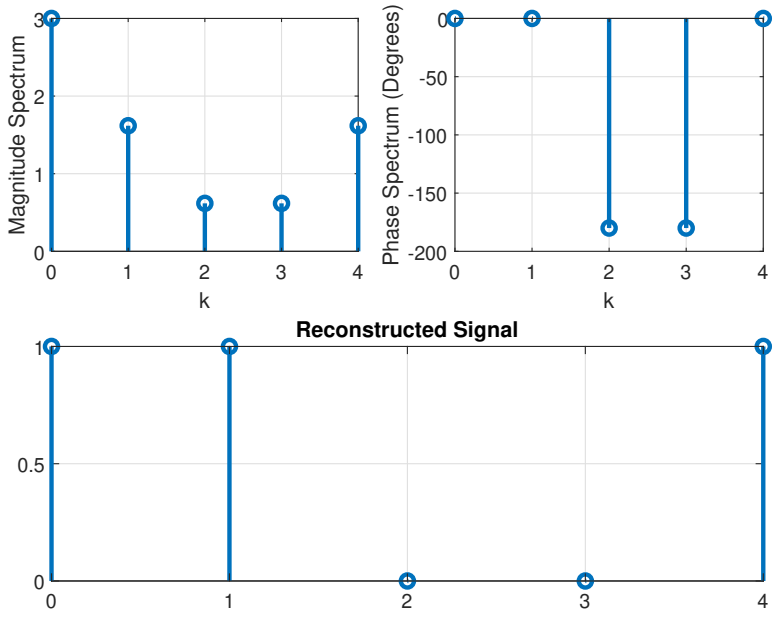


Figure. 5.3: Frequency spectrum and reconstruction of a sequence x[n] = {1,1,0,0,1} using DFT and IDFT functions respectively.

From this part of our lab, we'll use **fft** and **ifft** function in MATLAB to compute the DFT of and IDFT of a signal. Syntaxes for fft function are:

$$\mathbf{Y = fft(X)}$$
$$\mathbf{Y = fft(X, N)}$$

The first line computes the n-point DFT of signal X where n =length(X) and the second one computes the N-point DFT of a signal given by the user. While plotting any fft output either magnitude or phase, we'll first fftshift the fft of X. fftshift(X) rearranges a Fourier transform X by shifting the zero-frequency component to the center of the array.
In the following example, we'll find out the frequency spectrum of a sinc function. Remember that a sinc function is defined by the following equation:

$$\text{sinc}\ (t) = \begin{cases} \frac{\sin \pi t}{\pi t} & t \neq 0 \\ 1 & t = 0 \end{cases} \tag{16}$$

In our example, we'll consider $x(t) = \text{sinc}(100\text{t})$. First we'll sample the signal with sampling period 0.8333 ms and then computes its spectrum using FFT. Following is code for this purpose:

Listing 7: Example of computing DFT of a sinc signal using **fft** function

```
clc;
clear all;
close all;

% Discrete Fourier Transform of a sinc function using FFT

t0 = 0.2; % duration of sinc pulse
ts = 8.3333e-4; % sampling period
t = -t0/2:ts:t0/2; % defining time variable
x = sinc(100*t); % sinc function
fs = 1/ts; % sampling frequency
N = 256; % Number of points DFT will be calculated
X = fft(x, N);
df = fs/N;
f = -fs/2:df:fs/2-df+df/2*mod(N,2);
subplot(311)
plot(t, x, 'Linewidth', 2); title('sinc Pulse'); ylabel('Amplitude'); xlabel(
    't');
subplot(312)
stem(f, abs(fftshift(X)), 'Linewidth', 2); % We'll use fftshift while
    plotting an fft
title('Magnitude Spectrum'); ylabel('|X(k)|'); xlabel('f');
subplot(313)
stem(f, angle(fftshift(X))*180/pi, 'Linewidth', 2); title('Phase Spectrum');
ylabel('Phase (Degrees)'); xlabel('f');
```

The result is shown in the following figure.

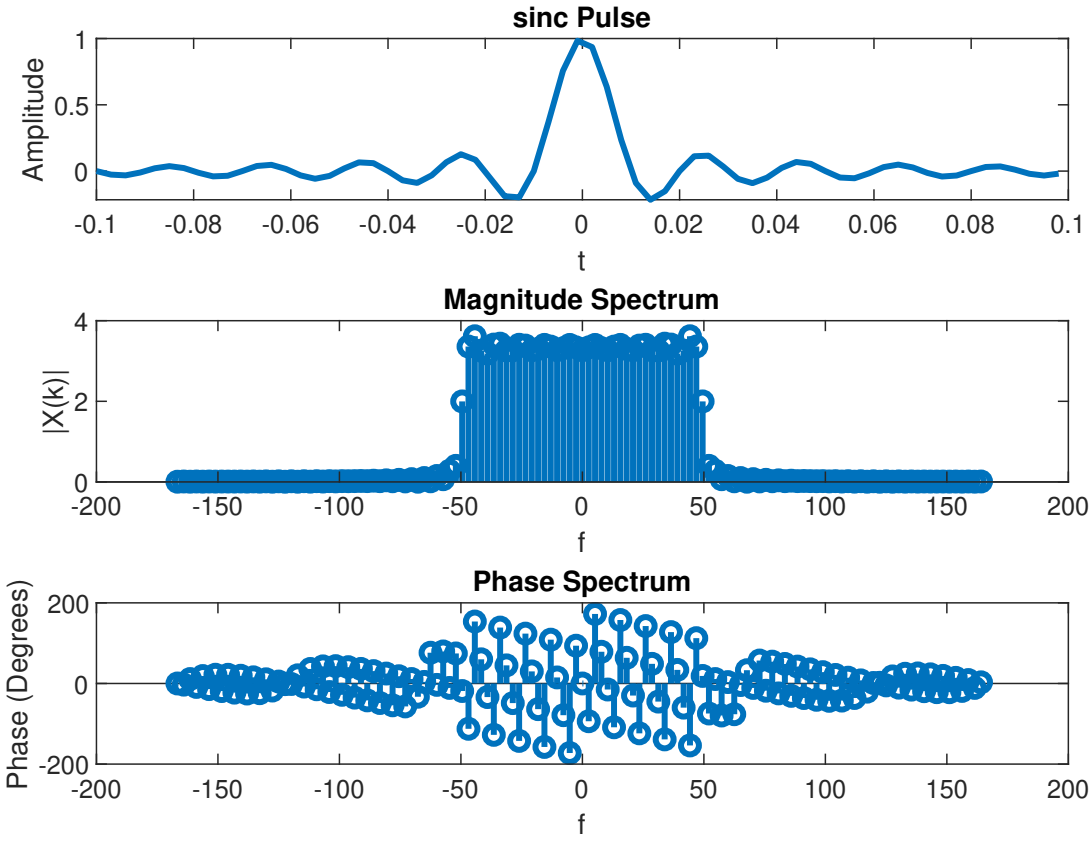


Figure. 5.4: Frequency spectrum of a sinc pulse.

In our next example, we'll discuss how to compute convolution sum of two sequences in frequency domain. It might be a good idea to review the section named **Convolution Using DFT**. Here we considered two signal, $x[n] = \underset{\uparrow}{2}, 1, 2, 1$ and $h[n] = \{\underset{\uparrow}{1}, 2, 3, 4\}$.

Listing 8: Convolution in frequency domain

```
clc;
clear all;
close all;

% Linear convolution using DFT

x = [2, 1, 2, 1]; % signal x
nx = 0:3; % index vector of x
h = [1, 2, 3, 4]; % signal h
nh = 0:3; % index vector of h
ny = min(nx)+min(nh):max(nx)+max(nh); % index vector of y
```

```
len = length(ny);
% padding zero after x and h to make it of length len
x = [x zeros(1, len-length(x))];
h = [h zeros(1, len-length(h))];

X = fft(x, len); % len point DFT of x
H = fft(h, len); % len point DFT of h

Y = X.*H; % Y is the product of X and H
y = ifft(Y); % y in time domain
```

The result for convolution is given by: $y[n] = \{2, \underset{\uparrow}{5}, 10, 16, 12, 11, 4\}$. You can verify the result using **convolution_sum** from lab week 3. In a similar way, correlation can also be calculated using fft.
Finally, we'll explore the application of fft to observe modulation and demodulation of a message signal in frequency domain. Let x[n] be a discrete signal with with DTFT $X(\Omega)$. If we modulate x[n] by a cosine carrier signal the output signal will simply be the multiplication of x[n] and $\cos(\omega_0 n)$. In frequency domain, the result will be as follows:

$$x(n)\cos\omega_o n \overset{\text{DTFT}}{\longleftrightarrow} \frac{1}{2}\left[X\left(\omega+\omega_o\right)+X\left(\omega-\omega_o\right)\right] \tag{17}$$

So, the original spectrum of x[n] will shift at ω_0 and $-\omega_0$. In our example, we considered a sinc function as a message signal.

$$m(t) = \begin{cases} \sin c(100t), |t| \leq t_o \\ 0, \quad \text{otherwise} \end{cases} \tag{18}$$

where, $t_o = 0.1s$. Let, carrier $c(t) = cos(2\pi f_c t)$ where $f_c = 250Hz$. So after modulation, the original signal's spectrum will shift to 250Hz and -250 Hz. Code for modulation is given below:

Listing 9: Modulation

```
clc;
clear all;
close all;
% DSB-SC Modulation of message signal using FFT

%% message signal and carrier signal generation
% we'll use a sinc function as a messeage signal
% and a cosine function with 250 Hz frequency as carrier
to = 0.2; % duration of the sinc function
ts = 0.001; % sampling period
fc = 250; % Carrier frequency
fs = 1/ts; % Sampling Frequency
t = -to/2:ts:to/2; % time axis for message signal

```

```
m = sinc(100*t); % message signal
c = cos(2*pi*fc*t); % carrier signal

%% Modulation
u = m.*c; % DSB-AM modulated signal
N = 1024; % FFT Bin size

M = fft(m, N); % N point DFT of message signal
M = M/fs; % scaling
U = fft(u, N); % N point DFT of modulated signal
U = U/fs;

% Visualizing modulated spectrum
fn = [0:1/N:1-1/N]*fs-fs/2; % Frequency axis for spectrum
figure(1)
subplot(211), plot(fn, abs(fftshift(M)), 'Linewidth', 2); title('Spectrum of
    the message signal');
subplot(212), plot(fn, abs(fftshift(U)), 'Linewidth', 2); title('Spectrum of
    the modulated signal');
```

The resulting spectrum is shown in figure 5.5.

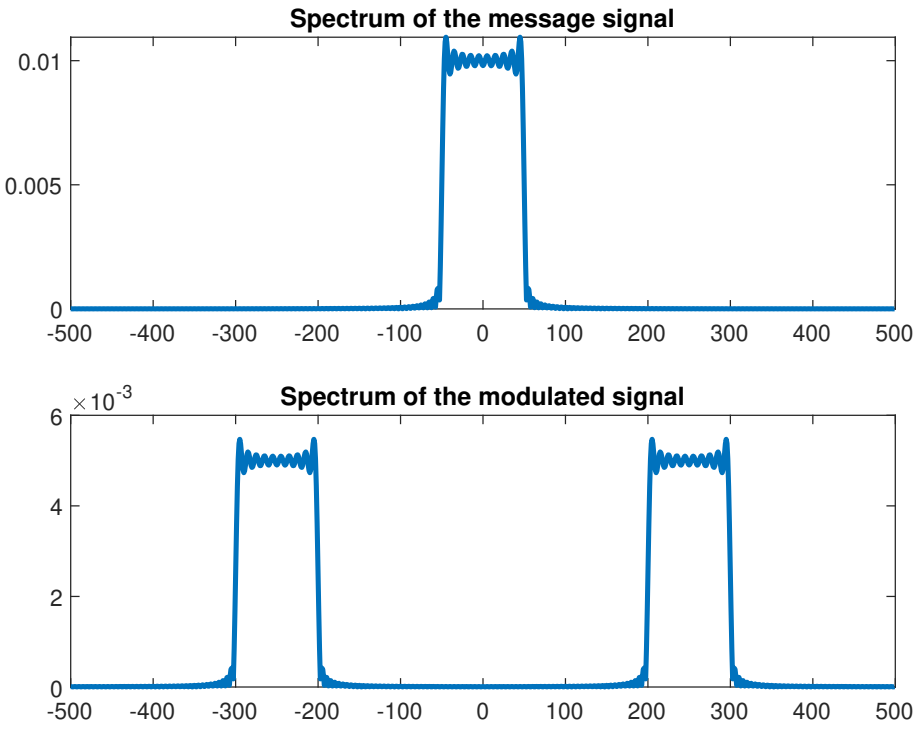


Figure. 5.5: After modulation, spectrum of the original signal is shifted to the carrier frequency and magnitude is halved.

Now consider the demodulation of the DSB-SC AM signal. Demodulation is done by multiplying the modulated signal again by the carrier signal.

$$x(n)\cos^2\omega_o n \overset{DTFT}{\longleftrightarrow} \frac{1}{2}X(\omega) + \frac{1}{4}\left[X\left(\omega + 2\omega_o\right) + X\left(\omega - 2\omega_o\right)\right] \tag{19}$$

As a result, the spectrum has three components. The lower frequency component corresponds to the original signal. This component is retrieved by using a low pass filter. To adjust the magnitude of the message signal, a low pass filter with gain 2 is used. The code and corresponding frequency spectra are as follows:

Listing 10: Demodulation

```
% continue the code after 'modulation' part
%% Demodulation
% Here we'll demodulate the modulated signal
% and then filter the demodulated signal to obtain message signal
y = u.*c; % demodulation
Y = fft(y,N); % N-point DFT of demodulated signal
Y = Y/fs;
figure(2)
subplot(211)
plot(fn, abs(fftshift(Y)), 'Linewidth', 2); % showing the demodulated
    spectrum
title('Spectrum of Demodulated signal');

%% filtering
fcut = 200; % cut-off frequency of the low pass filter
ncut = floor(fcut*fs/N); % index vector for filter upto cut-off frequency
H = zeros(1,N); % Filter vector
H(1:ncut) = 2*ones(1,ncut); % Low pass filter with gain 2
H(N-ncut+1:N) = 2*ones(1, ncut); % Other portion of the low pass filter
Ufiltered = Y.*H; % Filtering the modulated spectrum
subplot(212)
plot(fn, abs(fftshift(Ufiltered)), 'Linewidth', 2); title('Spectrum of
    Demodulated Signal');

%% visualizing the original and reconstructed message signal in time domain
ufiltered = real(ifft(Ufiltered))*fs; % ufiltered is the time domain version
    of Ufiltered
figure(3)
subplot(211) plot(t, m, 'Linewidth', 2); title('Message Signal');
subplot(212) plot(t, ufiltered(1:length(t)), 'Linewidth', 2); title('
    Reconstructed Signal');
```

The spectrum of demodulation and filtered signal is shown below:

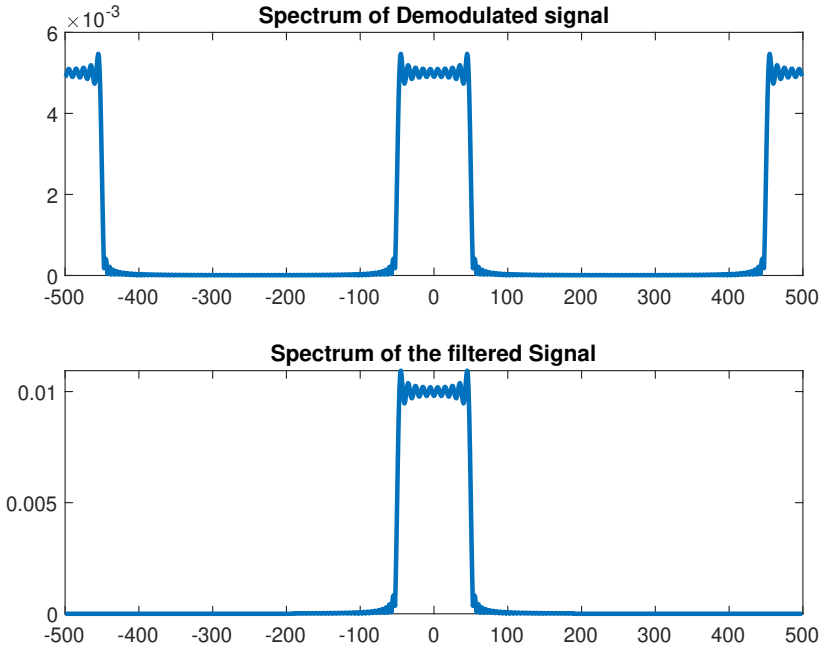


Figure. 5.6: Sepctra for demodulation and filtering.

You can also observed the original signal and filtered signal in time domain.

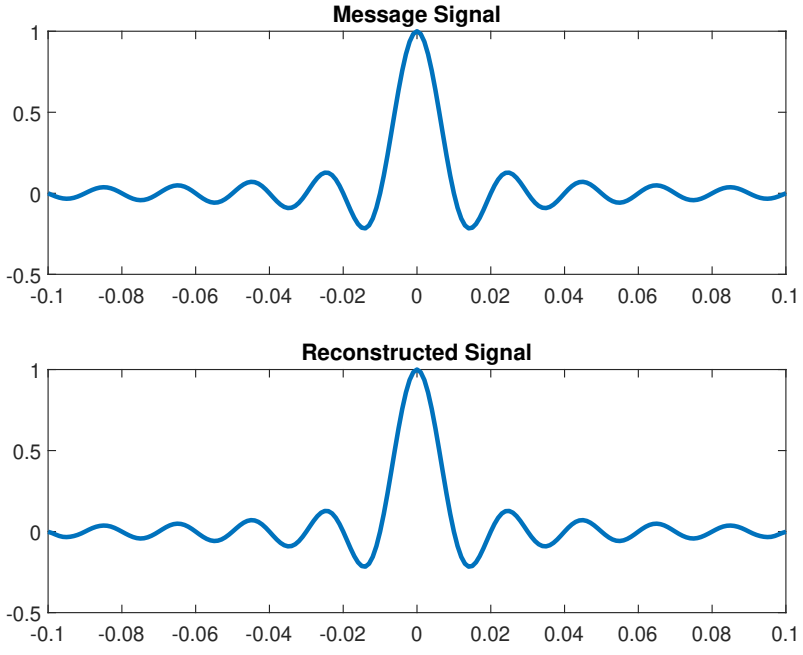


Figure. 5.7: Original signal and Reconstructed signal

Post Lab Tasks

- In example 1 of DTFS spectrum, i) show the reconstructed signal from its samples for m = 0.5, 1 and 3. ii) Show the spectrum for m = 1 and 3. What do you observe? iii) Show the reconstructed signal from its samples for $t_{remax} = T$ and $3 * T$. What difference do you observe in each case?
- Consider a signal $x(t) = 3\cos(200\pi t) + 4\sin(500\pi t) - \cos(800\pi t)$. Determine the DTFS spectrum of this signal. Assume sampling frequency, $F_s = 1.6$KHz.
- Consider a signal $x(t) = sinc^2(100t)$. Sample the signal at a frequency 5KHz and find the DTFT spectrum using the code in Listing 3.
- Consider a signal $x[n] = \{\underset{\uparrow}{1}, 1, 2, 3, 5, 8, 13\}$. Find autocorrelation of x[n] using FFT. Show the spectrum and time domain plot for the result.
- Consider the following system shown in figure 5.8. If

$$m_1(t) = \begin{cases} \sin c(100t), |t| \leq t_o \\ 0, \quad \text{otherwise} \end{cases}, \quad m_2(t) = \begin{cases} \sin c^2(100t), |t| \leq t_o \\ 0, \quad \text{otherwise} \end{cases}$$

where $t_0 = 0.1s$. Let carrier $c_1(t) = \cos(2\pi f_{c1} t)$ where $f_{c1} = 250Hz$ and $c_2(t) = \cos(2\pi f_{c2} t)$ where $f_{c2} = 750Hz$. Obtain magnitude spectrum of m_1, m_2 and final output y(t) of the system.

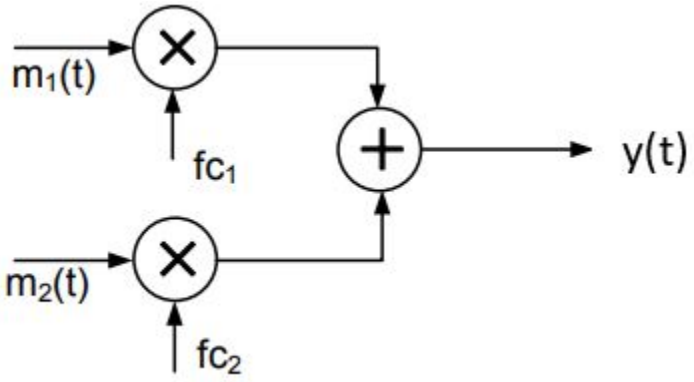


Figure. 5.8

LAB 6:
Z-Transform

Objectives

The main objectives of this lab are:

- To investigate signals and systems in three domain (Z-domain, frequency domain and time domain)
- To learn how Z transform can be used to analyze system output
- To learn the use of Z transform in the design process of a system
- To find pole-zero plot and frequency response from Z transform
- Stability and causality of a system using Z transform
- Cascade and parallel operations of systems (e.g. filters) using Z transform

PART 1

Z transform is an important tool in analyzing discrete-time (DT) signals and linear time-invariant (LTI) DT systems. It plays the same role as Laplace transform plays in analyzing continuous time signals and systems. Z transform is the generalization of the discrete-time fourier transform (DTFT). In this experiment, we'll discuss different features of z-transform and its application to characterize an LTI system. We'll discuss how system causality and stability can be determined from z-transform. We'll see how to get the frequency response of a system from z-transform.

Z Transform

The z-transform of a discrete-time signal x[n] is defined

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \tag{1}$$

where z is a complex variable. Since z-transform is an infinite power series, it exists only for those values of z for which the series given in 1 converges. The **region of convergence (ROC)** of X(z) is the set of all values of z for which X(z) attains a finite value. So while mentioning a z-transform, its ROC should also be indicated. Another handy notation to represent z-transform is:

$$X(z) = \mathcal{Z}\{x[n]\} \tag{2}$$

Consider a finite-duration signal x[n] = {2,$\underset{\uparrow}{1}$,2,1,3}. Its z-transformed signal,

$$\begin{aligned} X(z) &= \sum_{n=-\infty}^{\infty} x[n]z^{-n} = \sum_{n=-1}^{3} x[n]z^{-n} \\ &= 2z^1 + 1 + 2z^{-1} + z^{-2} + 3z^{-3} \end{aligned} \tag{3}$$

ROC of X(z) is entire z-plane except z = 0 and z = ∞.
Depending on the nature of the signal, ROC can be different. For example, ROC of a causal signal is the exterior of a circle of some radius r_2 while the ROC of an anticausal signal is the interior of a circle of some radius r_1. And in case of a non-causal signal, ROC is a ring in the z-plane. These points will be discussed in detail in theory.

Inverse Z-transform

Inverse z-transform is the process of converting a z-transformed signal into time domain. An inversion formula for obtaining x[n] from X(z) can be derived by using the *Cauchy integral theorem*. However, in case of rational z-transforms, partial fraction expansion and power series expansion methods are used. Partial fraction expansion is used to convert a cascade system into a parallel one. This point will be discussed in the coding section.

Poles and Zeros of Z-transform

Let us consider a rational X(z) in the form:

$$X(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots\cdots + b_M z^{-M}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots\cdots + a_N z^{-N}} \tag{4}$$

where, $a_N \neq 0$. Here we'll give the definitions of **poles** and **zeros**. Zeros of a z-transform X(z) are the values of z for which X(z) = 0 and poles are the values of z for which X(z) = ∞. In other words, the roots of numerator in (4) are called zeros and the roots of denominator in (4) are called poles of X(z).
If $a_0 \neq 0$ and $b_0 \neq 0$, we can factor out the terms $b_0 z^{-M}$ and $a_0 z^{-N}$,

$$X(z) = \frac{B(z)}{A(z)} = \frac{b_0 z^{-M}}{a_0 z^{-N}} \frac{z^M + (b_1/b_0)z^{M-1} + \cdots b_M/b_0}{z^N + (a_1/a_0)z^{N-1} + \cdots + a_N/a_0} \tag{5}$$

Since the numerator and denominator in equation (5) are polynomials, they can be expressed in factored form:

$$X(z) = \frac{B(z)}{A(z)} = \frac{b_0}{a_0} z^{-M+N} \frac{(z-z_1)(z-z_2)\cdots(z-z_M)}{(z-p_1)(z-p_2)\cdots(z-p_N)} \tag{6}$$

Thus X(z) has M finite zeros at $z = z_1, z_2, \cdots z_M$ (roots of the numerator polynomial), N finite poles at $z = p_1, p_2 \cdots p_N$ (the roots of the denominator polynomial) and $|N - M|$ zeros (if $N > M$) or poles (if N<M) at z = 0. For example consider a z-transform $X(z) =$

$\frac{z+1}{(z-\frac{1}{2})(z+\frac{3}{5})}$. The numerator has a root at z = -1 and two poles at $z = \frac{1}{2}$ and $z = -\frac{3}{5}$. So the zero of X(z) is $\frac{1}{2}$ and poles are $\{\frac{1}{2}, -\frac{3}{5}\}$.

Pole-Zero Plot

X(z) can be graphically represented by a **pole-zero plot** in the complex plane, which shows the location of poles by crosses (x) and the location of zeros by circles (∘). The multiplicity of multiple-order poles or zeros is indicated by a number close to the corresponding cross or circle. By definition, the ROC of a z-transform should not contain any poles.
The pole-zero plot of a z-transform provides important information about the system's behaviour. The MATLAB/OCTAVE function zplane allows the computation and visualization of the pole-zero plot of a z-transform. Zeros and poles can individually be determined from a z-transform using the MATLAB/OCTAVE function roots. We'll see the applications of these functions in the coding section.

Pole Zero Location and System's Behavior

Convolution property of z-transform implies that the convolution of x[n] and h[n] in time domain corresponds to multiplication in z-domain. If x[n] is the input sequence to an LTI system with impulse response h[n], the output sequence y[n] is given by the following equation for convolution sum:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{7}$$

Using the convolution property, we can write equation (7) in the z-domain as:

$$Y(z) = X(z)H(z) \tag{8}$$

where Y(z), X(z) and H(z) are the z-transform of y[n], x[n] and h[n] respectively. So if we know any x[n] and observe the corresponding y[n], we can determine the impulse response by first solving H(z) from the relation

$$H(z) = Y(z)/X(z) \tag{9}$$

and then evaluation the inverse z-transform of H(z). As H(z) and h[n] are equivalent description of a system in the two domains, we can deduce many important properties of a system from H(z). H(z) is called the **system function**.
Here, the coefficients of the denominator a_k are called the are the 'feed-backward' coefficients and the coefficients of the numerator are the 'feed-forward' coefficients, b_k. These are called filter coefficients which will be discussed in detail in the next two labs.
Now we'll discuss the impact of pole location and time domain behavior of system function. We'll consider real and causal signals. First we'll see that the characteristics behavior of causal signals depends on whether the poles of the transform are contained in the region

$|z| < 1$, or in the region $|z| > 1$ or the circle $|z| = 1$. Since the circle $|z| = 1$ has a radius of 1, it is called the unit circle. It acts as a reference line for pole-zero plot. The Unit Circle at the Z-plane is the set of points z to which the Z-Transform equals the Discrete Time Fourier Transform (DTFT).
In case of a single pole, the signal is decaying if the pole is inside the unit circle, constant if the pole is on the unit circle and growing if the pole is outside the unit circle. In addition, negative pole results in a signal that alternates in sign. Following figure illustrates these points.

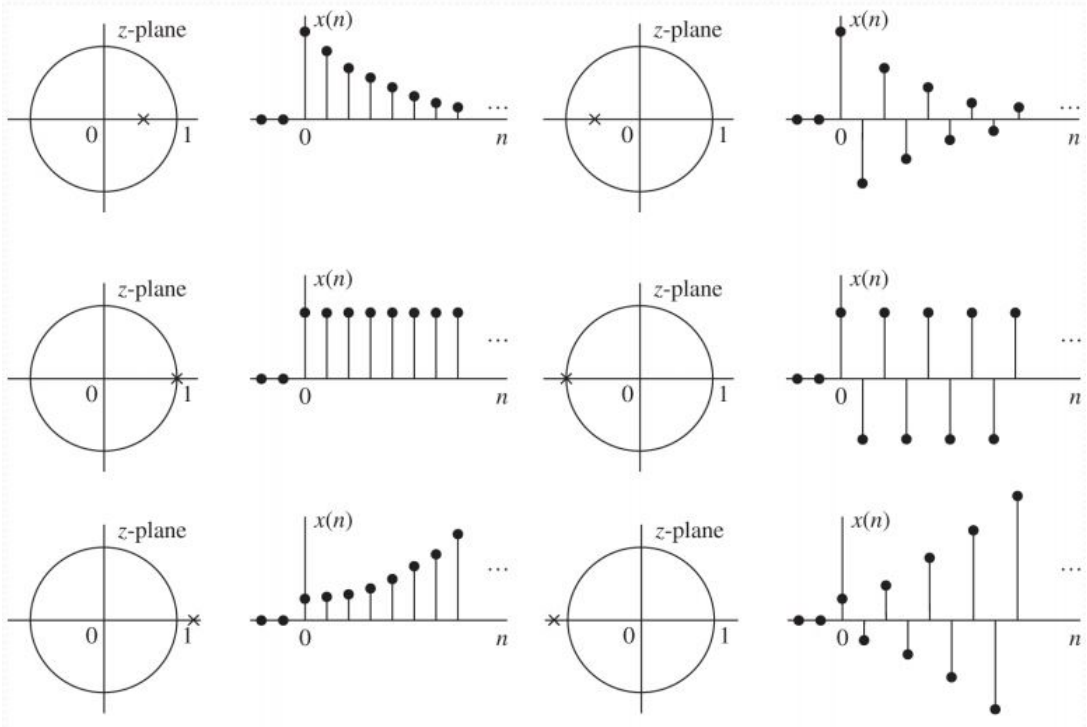


Figure. 6.1: Time-domain behavior of a single real pole causal signal as a function of the pole location with respect to the unit circle.

In case of double pole, the only exception is that a single pole on the unit circle corresponds to a constant signal but a double real pole on the unit circle results in an unbounded signal. A pair of complex conjugate poles results in an exponentially weighted sinusoidal signal. The amplitude of the signal is growing if r>1, constant if r = 1 (sinusoidal signals) and decaying if r<1 (figure(6.3).

Depending on the ROC of H(z) and its pole-zero location we can determine if the LTI system described by H(z) is causal and stable or not.

1. An LTI system is causal if and only if the ROC of H(z) is the exterior of a circle of radius, $r < \infty$.

2. An LTI System is BIBO stable if and only if the ROC of H(z) includes the unit circle.

So, a causal LTI system is BIBO stable if and only if all the poles of H(z) are inside the unit

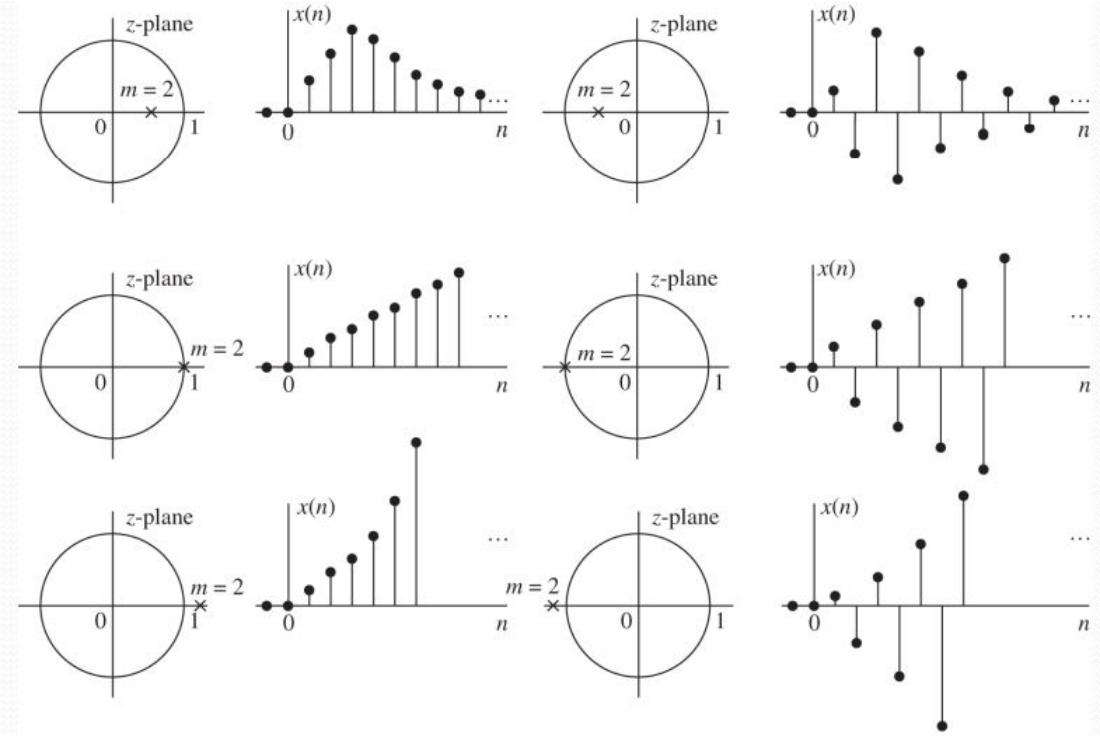


Figure. 6.2: Time domain behavior of causal signals corresponding to a double real pole as a function of the pole location.

circle. ROC can be observed from the pole-zero plot of H(z). We'll discuss this more in the class.

Frequency Response from Z-transform

Frequency reponse of a system is equivalent to evaluating the transfer function H(z) of that system on the unit circle, i.e. $z = e^{j\omega}$.

$$H\left(e^{j\omega}\right) = H(z)|_{z=e^{j\omega}} \tag{10}$$

Using this methodology, it is possible to sketch the frequency response of the system that is related to its poles and zeros. We'll compute the frequency response of a system $H(e^{j\omega})$ from H(z) using freqz function in MATLAB/OCTAVE.

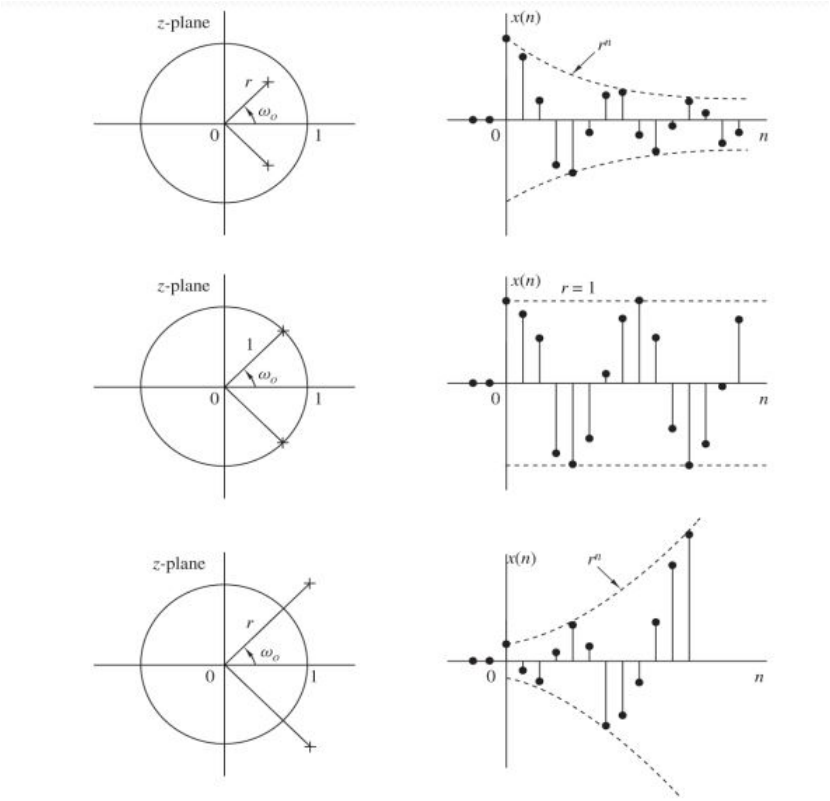


Figure. 6.3: A pair of complex conjugate poles corresponding to causal signals with oscillatory behavior.

PRELAB TASKS

1. Determine the z-transform of the signal x[n] = $(\frac{1}{4})^n$u[n] and mention its ROC.

2. Draw the pole-zero plot for X(z) = $\dfrac{1+z^{-1}}{1-z^{-1}+0.5z^{-2}}$.

3. Consider a transfer function H(z) has four real poles. If three of them are inside the unit circle and one of them is outside the unit circle. Comment on the time-domain behavior of H(z).

4. A linear time invariant system is characterized by the system function: $X(z) = \dfrac{1+6z^{-1}+z^{-2}}{(1-2z^{-1}+2z^{-2})(1-0.5z^{-1})}$. Specify all the ROCs of the system. In which case, the system is causal and stable.

Part 2
Z Transform in MATLAB/OCTAVE

- Z domain representation in MATLAB/OCTAVE
- Pole-zero plot in MATLAB/OCTAVE
- Frequency response of a system from its z-transform using MATLAB/OCTAVE
- Cascade and Parallel operations of systems using z transform in MATLAB/OCTAVE
- Functions to use: **zplane(), deconv(), residuez()**

Instructions

- Organizing files and folder properly for later use.
- Make a unique named folder for this lab like **EEE3218_SP20_A1_180105001** under any drive other than C drive.
- For every week make a subfolder to keep the all the tasks in a specific week separated from other tasks.
- Always try to work in the folder specifically created for the current week.
- Follow MATLAB/OCTAVE naming convention for giving a meaningful name before saving it in MATLAB/OCTAVE.
- While running a code be judicious in choosing *add to path* or *change directory*. Use add to path when you are using a file from different folder.

In this part of our lab, we'll implement the ideas covered in Lab in MATLAB to discuss the idea and application of z-transform. W'll see how to compute partial fraction expansion using residuez function. Syntax for the function is:

$$\textbf{[r, p, z] = residuez(b,a)}$$

This finds the residues, poles, and direct terms of a partial fraction expansion of the ratio of numerator and denominator polynomials, b and a. Consider a rational z-transform X(z) given by:

$$X(z) = \frac{1 + 2z^{-1} - z^{-2}}{1 - z^{-1} + 0.3561z^{-2}} \tag{11}$$

We'll determine its partial fraction expansion. For this type of X(z) we need to provide the numerator and denominator polynomials in MATLAB. In this example, numerator polynomial is num = [1 2 1] and denominator polynomial is den = [1, -1, 0.3561]. Then we'll call the **residuez** function.

Listing 1: Partial Fraction using residuez function

```
clc;
clear all;
close all;

% Partial fraction expansion of X(z) = (1+2z^-1-z^-2)/(1-z^-1+0.3561z^-2)

num = [1, 2, -1]; % numerator polynomials
den = [1, -1, 0.3561]; % denominator polynomials

[r,p,k] = residuez(num, den); % calling the residuez function
% r is the resiudes
% p is the poles
% k is the direct term


output:
r =

   1.9041 - 1.6822i
   1.9041 + 1.6822i

p =

   0.5000 + 0.3257i
   0.5000 - 0.3257i

k =

   -2.8082
```

So, X(z) can be written using partial fraction expansion:

$$X(z) = -2.8082 + \frac{1.9041 - 1.6822i}{1 - (0.5 + 0.3257i)z^{-1}} + \frac{1.9041 + 1.6822i}{1 - (0.5 - 0.3257i)z^{-1}}$$

By using this partial fraction equation and the look up table inverse z-transform can be computed. Remember that, the inverse z-transform will depend on the ROC of the z-transform. This will be discussed in the theory class. We'll not go into details here.
We'll now visualize the pole-zero plot of a z-transform using the zplane function in MATLAB/OCTAVE. Syntax for the function is

zplane(z,p)

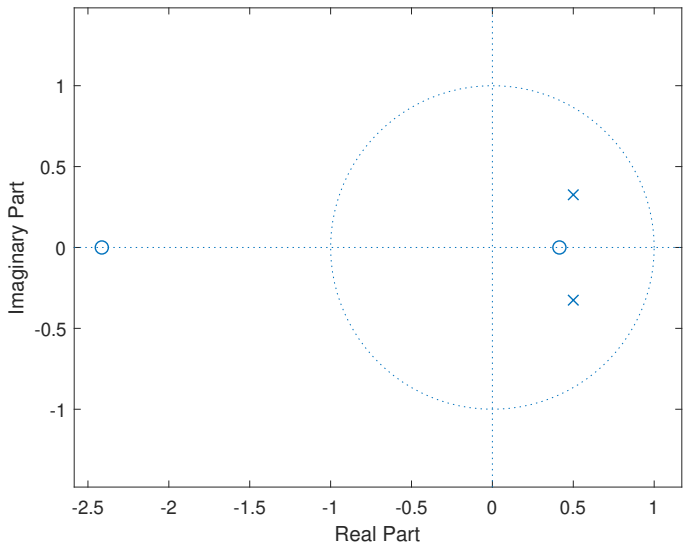


Figure. 6.4: Pole-zero plot of X(z) given by equation (11)

zplane(b,a), where b and a are row vectors, first uses **roots** to find the zeros and poles of the transfer function represented by the numerator coefficients b and the denominator coefficients a. The symbol 'o' represents a zero and the symbol 'x' represents a pole. The plot includes the unit circle for reference. Consider the z-transform given in equation (11).

Listing 2: Pole-zero plot of X(z)

```
clc;
clear all;
close all;


%pole-zero plot of X(z) with numerator given by b and
% denominator given by a
b = [1, 2, -1]; % numerator
a = [1, -1, 0.3561]; % denominator

zplane(b,a);
```

In the figure (6.4), the dotted line indicates the unit circle. As mentioned earlier this is the set of points z to which the z-transform equals the DTFT. The ∘ represents the zeros and the x represents the pole. The system has two complex conjugate poles inside the unit circle. If the locations of the poles and zeros are known, these can be used as inputs to the **zplane** command. The syntax of the command in this case is

zplane(z,p)

This plots the zeros specified in column vector z and the poles specified in column vector p in the current figure window. The zeros and poles can be found directly using the **roots**

command.
If we know the zeros and poles of a system's transfer function we can obtain its z-transfrom equation using the poly function in MATLAB/OCTAVE.
We'll now see how to use the freqz function to estimate the frequency response of a system. Given the transfer function of a system in the following form:

$$X(z) = \frac{b_0 + b_1 z^{-1} + \cdots\cdots + b_n z^{-n}}{a_0 + a_1 z^{-1} + \cdots\cdots + a_m z^{-m}} = \frac{b(z)}{a(z)} \tag{12}$$

the **freqz** function uses an FFT-based approach to compute the frequency response. The function has a variety of formats. A useful format is

$$\textbf{[h,f] = freqz(b,a, npt, Fs)}$$

where the variables b and a are the vectors of the numerator and denominator polynomials. In case of transfer function, vector 'b' is related to the input of a system and vector 'a' is related to the output of a system. These coefficients control the response of a system. Fs is the sampling frequency and npt the number of frequency between 0 and $\frac{Fs}{2}$. Using the freqz command without output arguments plots the magnitude and phase responses automatically. Consider the transfer function:

$$H(z) = \frac{1 - 1.16180 z^{-1} + z^{-2}}{1 - 1.5161 z^{-1} + 0.878 z^{-2}} \tag{13}$$

We'll compute its frequency response using freqz command.

Listing 3: Frequency response of H(z) using **freqz** command

```
clc;
clear all;
close all;

% Frequency response from transfer function

b = [1 1.16180 1]; % numerator
a = [1 -1.5161 0.878]; % denominator
% Fs = 256 and number of points between 0 and Fs/2 is 512
freqz(b, a, 256, 512);
```

The output response is shown in figure(6.5).

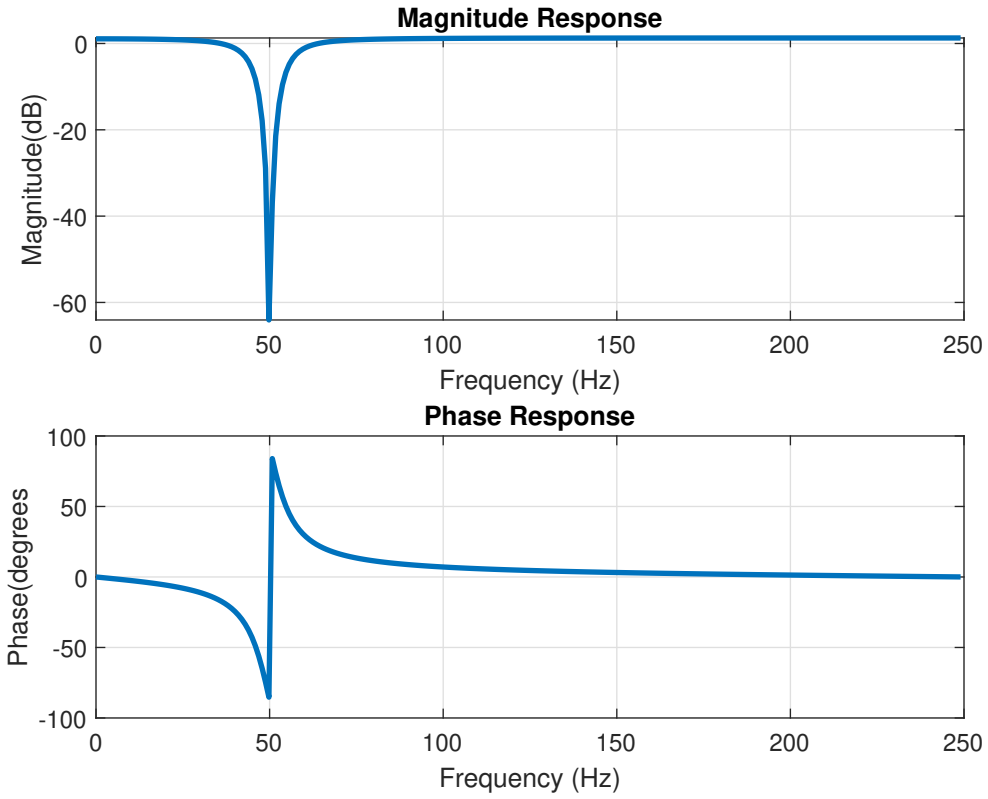


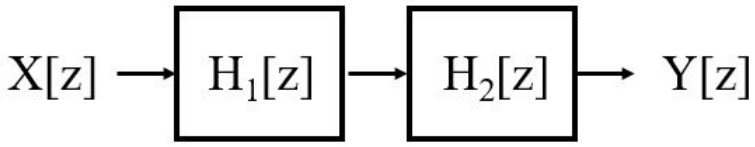
Figure. 6.5: Frequency Response of H(z). Top one is the magnitude plot and the bottom one is the phase plot.

This is the response of a notch filter. A notch filter is a band-stop filter with sharp stop-band. From the magnitude response, we see that it has a very sharp dip at frequency 50 Hz. So the filter only rejects signals having frequency 50Hz and allows the other ones. 50 Hz notch filter is used in many equipment in order to block the power signal which introduces a noise of 50 Hz. The response of this filter can be changed by changing the filter coefficients a which will change the poles of H(z).
Now we'll discuss how to convert between structures. Multiple Digital systems can be connected in cascade or parallel or a combination of these two. The two types of connection for two systems are shown in the figure (6.6). For cascade realization, the transfer function H(z) is factored as

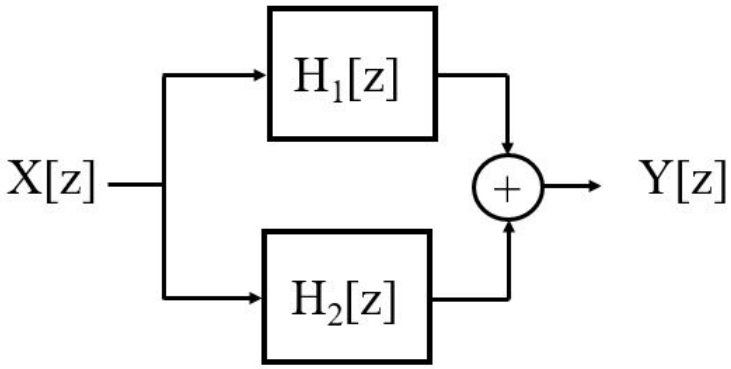
$$H(z) = H_1(z)H_2(z)H_3(z)\cdots H_k(z) = \prod_{i=1}^{k} H_i(z) \tag{14}$$

where $H_i(z)$ is either a second/first order section.

$$\begin{aligned} H_i(z) &= \frac{b_0 + b_{1i}z^{-1} + b_{2i}z^{-2}}{1 + a_{1i}z^{-1} + a_{2i}z^{-2}}, \text{ second order} \\ H_i(z) &= \frac{b_0 + b_{1i}z^{-1}}{1 + a_{1i}z^{-1}}, \text{ first order} \end{aligned} \tag{15}$$



(a) Cascade Connection



(a) Parallel Connection

Figure. 6.6: (a) Cascade Connection (b) Parallel Connection

where k is the integer part of $\frac{M+1}{2}$ (M is the order of the denominator of H(z)) For parallel realization, the transfer function H(z) is decomposed using partial fractions to give

$$H(z) = B_0 + \sum_{i=1}^{k} H_i(z) \tag{16}$$

where $H_i(z)$ is either a second/first order section and $B_0 = \frac{b_N}{a_M}$. Let us consider a cascaded system given by:

$$H(z) = \frac{(1 + 0.481199z^{-1} + z^{-2})(1 + 1.474597z^{-1} + z^{-2})}{(1 + 0.052921z^{-1} + z^{-2})(1 - 0.304609z^{-1} + 0.238865z^{-2})}$$

We'll convert it into parallel realization.In this problem, the z-transform has two pairs of numerator and denominator polynomials. So it is a cascade connection of two subsystems. The final z-transform will have a numerator that is the product of the numerators and a denominator that is the product of the denominators of the subsystems. To do this, we'll use MATLAB/OCTAVE function sos2tf to convert the two pairs of polynomials into a transfer function with a pair of rational polynomials of the form $\frac{b(z)}{a(z)}$. Remember that, **sos2tf** can be used only with polynomials with degree at most 2. **conv** function can be used in case of polynomials higher than degree two to convert the product of two polynomials into a single one. After obtaining the overall transfer function, we can use residuez function to find out the partial fraction.

Listing 4: Cascade to parallel structure conversion

```
clc;
clear all;
close all;

nstage = 2; % number of individual subsystems
N1 = [1 0.481199 1]; % numerator of H1(z)
N2 = [1 1.474597 1]; % numerator of H2(z)
D1 = [1 0.052921 0.831731]; % denominator of H1(z)
D2 = [1 -0.304609 1]; % denominator of H2(z)
sos = [N1 D1; N2 D2]; % system matrix
[b,a] = sos2tf(sos); % The overall system function
[c, p, k] = residuez(b,a); % partial fraction expansion
m = length(b);
b0 = b(m)/a(m); % B0 term in the parallel realization
j = 1;
for i = 1:nstage
    bk(j) = c(j)+c(j+1);
    bk(j+1) = -(c(j)*p(j+1)+c(j+1)*p(j));
    ak(j) = -(p(j)+p(j+1));
    j = j+2;
end

% The loop is used to add the first order systems to make it
% second order
```

Without the loop in the above code, the result will be:

$$H(z) = 1.2023 + \frac{-0.8535 - 1.7034i}{1 - (0.1523 + 0.9883i)z^{-1}} + \frac{-0.8535 + 1.7034i}{1 - (0.1523 - 0.9883i)z^{-1}} + \frac{0.7524 + 0.4716i}{1 - (-0.0265 + 0.9116i)z^{-1}} + \frac{0.7524 - 0.4716i}{1 - (-0.0265 - 0.9116i)z^{-1}}$$

The function of the loop is to sum the terms with complex conjugate poles. After the execution of loop,

$$H(z) = 1.2023 + \frac{-1.707 + 3.6271z^{-1}}{1 - 0.3046z^{-1} + z^{-2}} + \frac{1.5047 - 0.82z^{-1}}{1 - 0.0529z^{-1} + 0.8317z^{-2}}$$

Finally we'll see how the system property like stability and causality depend on the pole location. For example, consider a system with transfer function

$$H(z) = \frac{3 - 4z^{-1}}{1 - 3.5z^{-1} + 1.5z^{-2}} \tag{17}$$

The system has poles at $z = \frac{1}{2}$ and $z = 3$. So, the possible ROCs for the system are:

i) $|z| < \frac{1}{2}$

ii) $\frac{1}{2} < |z| < 3$

iii) $|z| > 3$

The pole-zero plot with each of the ROC and their corresponding time domain impule response are shown in the following figure:

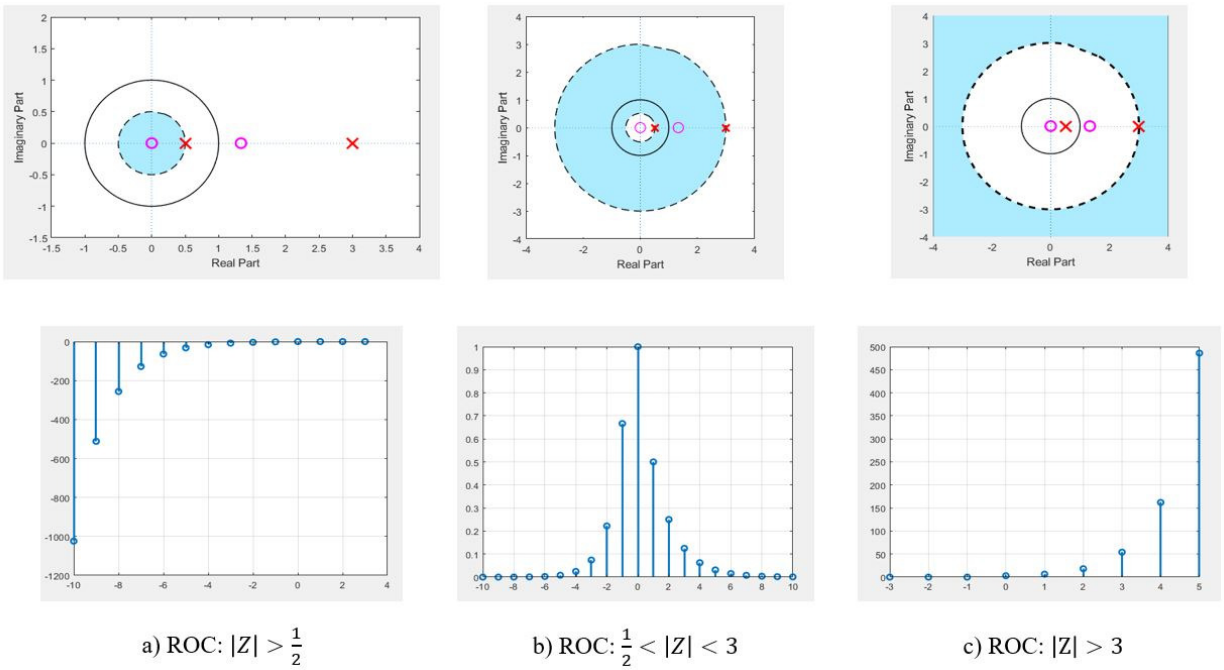


Figure. 6.7: Pole-zero plots with ROC (upper portion) and corresponding impulse response (lower portion). The Blue region is the ROC for each cases.

From the figure (6.7), it is seen that in case of ROC: $|z| < \frac{1}{2}$, the ROC does not contain the unit circle and as ROC is the interior of the circular region, the system is both unstable and anti-causal. Impulse response for this case confirms this. Similarly for $|z| > 3$, the ROC also does not contain the unit circle but the ROC is the exterior of the circular region and hence the system is causal but unstable. In case of $\frac{1}{2} < |z| < 3$, the ROC contains the unit circle and is a ring. So the system is non-causal and stable. The time domain signal shown in the figure confirms this. Thus pole-zero plot along with ROC of a system gives us important information about a system's causality and stability. By controlling the filter coefficients, we can control the location of poles and zeros of a system which controls the time domain behavior of the system. For example, by placing all the poles of a causal system inside the unit circle, we can make it stable. These points will be delineated in later labs when we'll discuss filter design.

Post Lab Tasks

1. Consider the system $H(z) = \dfrac{z^{-1} + \dfrac{1}{2}z^{-2}}{1 - \dfrac{3}{5}z^{-1} + \dfrac{2}{25}z^{-2}}$. Determine the frequency response of the system.

2. Consider the system: $H(z) = \dfrac{z^3 - 2z^2 + 2z - 1}{(z-1)(z-0.5)(z-0.2)}$. ROC of the z transform is given by: $|z| > 2$. Show the pole-zero diagram of the system. Is the system stable?

3. Determine the partial fraction expansion of $X(z) = \dfrac{2z^4 + 16z^3 + 44z^2 + 56z + 32}{3z^4 + 3z^3 - 15z^2 + 18z - 12}$. Find its pole-zero plot and determine the ROC of the z-transform. What are the ROCs for which the system is causal and stable? Also determine the system's frequency response.

4. Determine the parallel realization of the system given as follows:

$$X(z) = \frac{N_1(z)N_2(z)N_3(z)}{D_1(z)D_2(z)D_3(z)}$$

where,

$$\begin{aligned}
N_1(z) &= 1 - 1.22346z^{-1} + z^{-2} \\
N_2(z) &= 1 - 0.437833z^{-1} + z^{-2} \\
N_3(z) &= 1 + z^{-1} \\
D_1(z) &= 1 - 1.4334509z^{-1} + 0.85811z^{-2} \\
D_2(z) &= 1 - 1.293601z^{-1} + 0.556929z^{-2} \\
D_3(z) &= 1 - 0.612159z^{-1}
\end{aligned}$$

LAB 7:
IIR Filter Design

Objectives

The main objectives of this lab are:

- To be familiar with the basic idea of digital filter and filter specification
- To understand Infinite Impulse Response (IIR) Filter
- To design digital IIR filters from specification through coefficient calculation

PART 1

In digital signal processing, we often need to change the relative amplitudes of the frequency components of the signal or remove unwanted frequency components from a signal. This process is known as filtering. Digital filters are used in numerous signal processing applications, for example, signal restoration, telecommunication system etc. In this lab, we'll discuss how to design an infinite impulse response (IIR) filter considering given filter specifications.

FIR and IIR Filters

Digital filters require both the time and frequency domain techniques. This is because the desired filter characteristics are specified in the frequency domain but filters are usually described by difference equation in time domain. In the filter design process, we determine the coefficients of the filter's difference equation.
A general linear and time invariant causal digital filter with input x[n] and output y[n] can be described by linear constant coefficient difference equations of the form:

$$y[n] = -\sum_{k=1}^{N} b_k y[n-k] + \sum_{k=0}^{M} a_k x[n-k] \tag{1}$$

where a_k and b_k are the coefficients which characterizes the filter. a_k's are the feedforward coefficients (affect the input signal) and b_k's are the feedback coefficients (affect the output signal). In case of impulse response, the input is a delta function. If we denote the impulse response h[n], equation (1) becomes

$$h[n] = -\sum_{k=1}^{N} b_k h[n-k] + \sum_{k=0}^{M} a_k \delta[n-k] \tag{2}$$

If we take z-transform of equation (1), we get the system function as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_k \sum_{k=0}^{M} z^{-k}}{1 + \sum_{k=1}^{N} b_k z^{-k}} \tag{3}$$

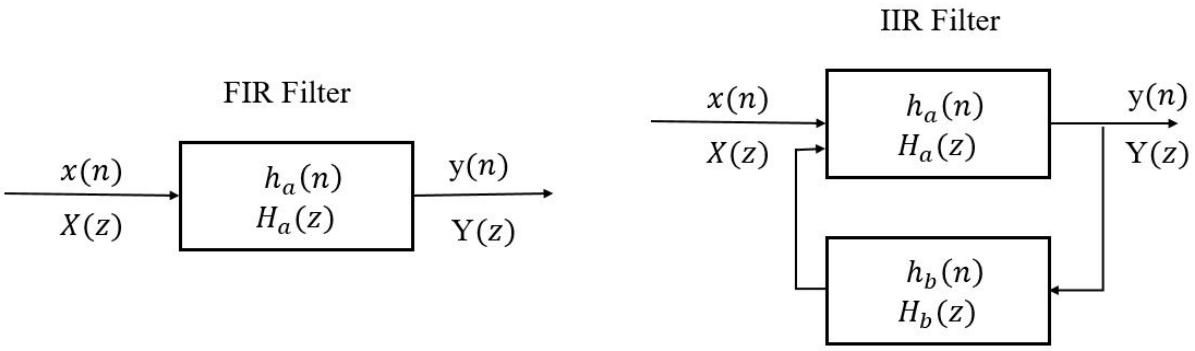


Figure. 7.1: (a) Block Diagram of FIR filter (b) Block Diagram of IIR Filter. Note that, FIR Filter only consists of feedforward block but IIR filter has both feedforward and feedbackward blocks.

The system function of the filter has M zeros and N poles. Depending on the nature of the impulse response, digital filters are of two types: infinite impulse response (IIR) and finite impulse response (FIR) filters. If $b_k = 0$ for all k in equation (2), then the filter is FIR filter. In case of FIR filter, h[n] and H(z) are given by:

$$\begin{aligned} h[n] &= \sum_{k=0}^{M} a_k \delta[n-k] \\ H(z) &= \sum_{k=0}^{M} a_k z^{-k} \end{aligned} \tag{4}$$

Since in case of FIR filter, (4) is not recursive, the impulse response has finite duration M and hence the name finite impulse response filter. On the other hand, $b_k \neq 0$ represents an IIR filter. In this case, the equation (2) is recursive and generates an impulse response which is non-zero for $n \to \infty$. The block diagram of FIR and IIR filter is shown in figure(7.1).

Design Specification of a Practical Filter

In designing any filter, the ideal frequency response characteristics are desirable but they are not realizable and necessary in most practical applications. In figure(7.2), the magnitude characteristics of a physically realizable low pass filter is shown. The bold black line represents the ideal response. In practical filter, magnitude of $H(e^{j\omega}$ in the passband (the allowable frequency range) is not constant like the ideal response but has a small amount of ripple. Similarly in the stop band (the forbidden frequency), the filter response $|H(e^j\omega)|$ is not zero. The transition of the frequency response from passband to stopband defines the *transition band* or *transition region* of the filter. The bandedge frequency ω_p defines the edge of the passband while the frequency ω_s denotes the beginning of the stopband. The width of the transition band is $\omega_s - \omega_p$. The width of the passband is usually called the *bandwidth* of the filter. If δ_p is the amount of ripple in the passband, the magnitude $|H(e^{j\omega}|$ varies between the limits $1 \pm \delta_p$.

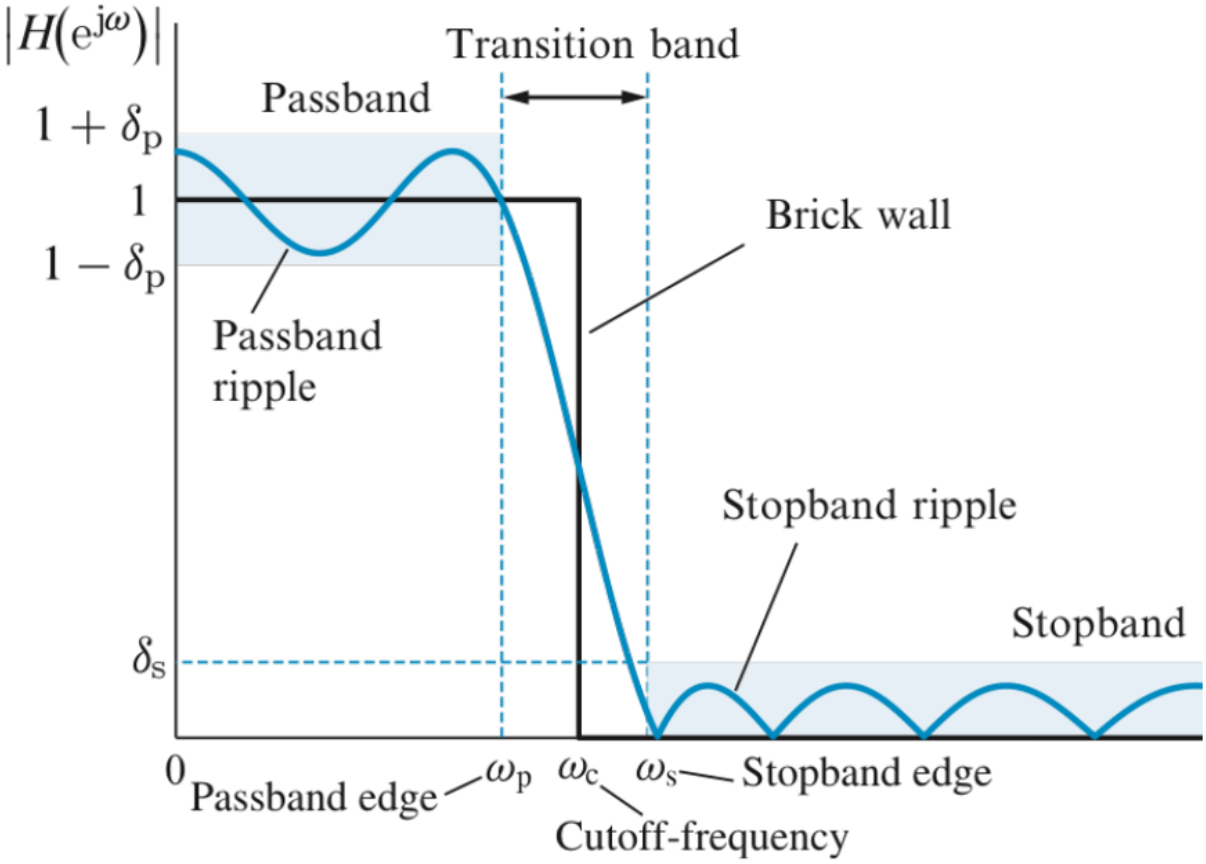


Figure. 7.2: Magnitude characteristics of physically realizable filters

The ripple in the stopband of the filter is denoted by δ_s. Usually the response of any filter is given in log scale. In that case, passband ripple and stopband attenuation in decibels are given by

$$\begin{aligned} &\text{Passband ripple, } R = -20\log_{10}(1-\delta_p) \\ &\text{Stopband Attenuation, } A = 20\log_{10}\delta_s \end{aligned} \tag{5}$$

In any filter design problem, the following requirements are specified:

- The maximum tolerable passband ripple
- The maximum tolerable stopband ripple
- The passband edge frequency, ω_p
- The stopband edge frequency, ω_s

Based on these specifications, we can select the filter coefficients $\{a_k\}$ and $\{b_k\}$ in the frequency response characteristics.

Design of IIR Filters

Design of a digital IIR filter implies to determine the coefficients of the equation given in eqaution(1) according the filer specifications. A simple way to obtain the IIR filter coefficients is to place the poles and zeros in the z-plane such that the resulting filter has the desired frequency response. This method is known as the pole-zero placement method and only useful for simple filters such as notch-filter design. A more efficient approach is first to design an analog filter satisfying the desired specifications and then to convert it into an equivalent digital filter. Most IIR filters are designed this way. Three of the most common methods of converting analog filters into equivalent digital filters are the impulse invariant, the matched z-transform and the bilinear z-transform methods. In our lab, we'll consider the pole-zero placement and the bilinear z-transform method to design an IIR filter.

Steps of filter design using pole-zero placement method

By strategically placing poles and zeros on the z-plane, simple low pass or other frequency selective filters can be designed. While designing simple digital filters using pole-zero placement method the following steps are to be followed:

- The location of poles and zeros are to be determined from the desired filter specifications.
- The transfer function $H(z)$ is written in terms of poles and zeros. If $z_1, z_2 \cdot z_M$ are the zeros of the transfer function and $p_1, p_2 \cdots p_N$ are the poles, then

$$H(z) = \frac{(z - z_1)(z - z_2) \cdots (z - z_M)}{(z - p_1)(z - p_2) \cdots (z - p_N)} \tag{6}$$

- From H(z), the corresponding difference equation is obtained and finally the filter coefficients $\{a_k\}$ and $\{b_k\}$ are determined.

We'll illustrate these points in the MATLAB coding section with an example.

Bilinear Transformation

Most of the digital filters are designed from their analog counterparts using a transformation. In bilinear transformation method, an analog filter $H(s)$ (where H(s) is the s-domain representation of the transfer function of the system) is converted into a digital filter H(z) by replacing s as follows:

$$s = \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}} \tag{7}$$

The above transformation maps H(s) from the s-plane to H(z) in z-plane. However, the direct replacement of s in H(s) as shown in equation (7) may lead to a digital filter with undesirable response. This is because the behavior of the desired filter response $H(e^{j\omega})$ at some analog frequency $\omega = \omega_a$ does not appear at the digital frequency ω_a but at frequency ω_d, where

$$\omega_d = \frac{2}{T} \left(\tan^{-1} \frac{\omega_a T}{2} \right) \tag{8}$$

This is termed as frequency warping which will be discussed in detail in theory. For this reason, before replacing s by the formula given in equation (7), the cut-off frequency of the filter must be pre-warped by the following equation:

$$\omega_a = k \tan\left(\frac{\omega_d T}{2}\right)$$

where,
ω_d = specified cutoff frequency
ω'_a = prewarped cutoff frequency
T = sampling period

Now if we summarize the steps for this process:

- Given the digital filter frequency specifications, we need to prewarp the digital frequency specifications to the analog frequency specifications. For the low pass filter and high pass filter:

  $$\omega_a = \frac{2}{T} \tan\left(\frac{\omega_d T}{2}\right)$$

  For the bandpass and bandstop filter:

  $$\omega_{al} = \frac{2}{T} \tan\left(\frac{\omega_l T}{2}\right),\ \omega_{ah} = \frac{2}{T} \tan\left(\frac{\omega_{dh} T}{2}\right)$$

- Then we need to perform prototype transforming by replacing s in the transfer function, H(s) using one of the following transformations, depending on the type of filter required:

  $$\begin{aligned} s &= \frac{s}{\omega_a} \text{ from lowpass to lowpass} \\ s &= \frac{\omega_a}{s} \text{ from lowpass to highpass} \\ s &= \frac{s^2 + \omega_0^2}{sW} \text{ from lowpass to bandpass} \\ s &= \frac{sW}{s^2 + \omega_0^2} \text{ from lowpass to bandstop} \end{aligned} \tag{9}$$

  where, $\omega_0 = \sqrt{\omega_{al}\omega_{ah}}$ and $W = \omega_{ah} - \omega_{al}$

- Finally we need to substitue $s = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}$ in H(s) to obtain the digital filter.

We'll explain these steps using an example in coding section.

PRELAB TASKS

1. Write down three practical applications of both FIR and IIR filters.

2. Consider a causal IIR system with system function:

$$H(z) = \frac{1 + 2z^{-1} + 3z^{-2} + 2z^{-3}}{1 + 0.9z^{-1} - 0.8z^{-2} + 0.5z^{-3}}$$

   Write down the feedforward and feedback coefficients for this filter.

3. Consider a normalized low pass analog filter with cut-off frequency 1 rad/sec given as:

$$H(s) = \frac{1}{s + 1}$$

   If we want to convert the analog filter into a digital one by bilinear transformation, what is the prewarped analog frequency?

Part 2
IIR Filter Design in MATLAB/OCTAVE

- Realization of IIR filters in MATLAB/OCTAVE
- Functions to use: **zplane()**, **bilinear()**, **poly()**, **freqz()**

Instructions

- Organizing files and folder properly for later use.
- Make a unique named folder for this lab like **EEE3218_SP20_A1_180105001** under any drive other than C drive.
- For every week make a subfolder to keep the all the tasks in a specific week separated from other tasks.
- Always try to work in the folder specifically created for the current week.
- Follow MATLAB/OCTAVE naming convention for giving a meaningful name before saving it in MATLAB/OCTAVE.
- While running a code be judicious in choosing *add to path* or *change directory*. Use add to path when you are using a file from different folder.

In this section, we'll discuss the design procedure mentioned in the theoretical section with some examples. Consider an example in which we want to design a bandpass digital filter using pole-zero placement method to meet the following specifications:

- complete signal rejection at dc and 250 Hz
- a narrow passband centered at 125 Hz
- a 3dB bandwidth of 10 Hz

The sampling frequency is 500 Hz. Let us first determine the pole-zero location of the desired filter. Since complete rejection of signals is required at DC i.e. 0Hz and 250Hz, we need to place the zeros at an angle 0° and $2\pi \times \frac{250}{500} = \pi$ on the unit circle. To have a narrow passband centered at 125 Hz, we need to place poles at $2\pi \times \frac{125}{500} = \frac{\pi}{2}$. Now in order to ensure that the filter coefficients are real, we need to have a complex conjugate pole pair and so the angle

corresponding to the pole will be $\pm\frac{\pi}{2}$.
Now we need to determine the magnitude of the poles which is governed by the bandwidth specifications. An approximate relationship between r (for r>0.9) and bandwidth (bw) is given by:

$$r \approx 1 - \left(\frac{\text{bw}}{F_s}\right)\pi \tag{10}$$

For the problem, bw = 10Hz and $F_s = 500Hz$. So plugging this number in equation(10), we get r = 0.937. The pole-zero diagram for this filter is given below:

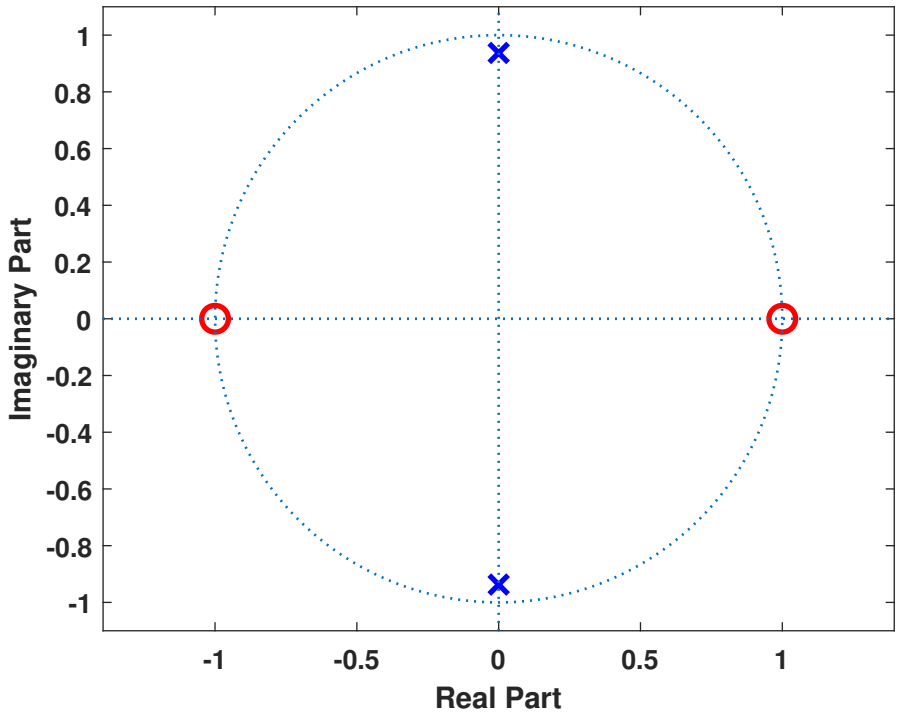


Figure. 7.3: Pole-zero plot of the filter

Now the transfer function of the filter can be written in z-domain:

$$\begin{aligned} H(z) &= \frac{(z-1)(z+1)}{(z - re^{j\pi/2})(z - re^{-j\pi/2})} \\ &= \frac{z^2 - 1}{z^2 + 0.877969} \\ &= \frac{1 - z^{-2}}{1 + 0.877969z^{-2}} \end{aligned}$$

Now from the above equation we find that the filter is a second order one with the coefficients:

$$\begin{aligned} a_0 &= 1 \qquad b_1 = 0 \\ a_1 &= 0 \qquad b_2 = 0.877969 \\ a_2 &= -1 \end{aligned}$$

The code and filter's magnitude response is given below:

Listing 1: Design of a band-pass filter using pole-zero placement

```
clc; clear all;close all;

Fs = 500; % Sampling frequency
bw = 10; % bandwidth

%% zero-calculation
Fr1 = 0; % DC frequency
Fr2 = 250; % 250Hz frequency

% angle corresponding to the zeros
theta1 = 2*pi*(Fr1/Fs);
theta2 = 2*pi*(Fr2/Fs);

z1 = exp(1j*theta1);
z2 = exp(1j*theta2);

%% pole-calculation
Fp = 125; % passband frequency
thetap = 2*pi*(Fp/Fs); % angle corresponding to the pole

r = 1-(bw/Fs)*pi; % magnitude of the pole
p1 = r*exp(1j*thetap); %first pole
p2 = r*exp(-1j*thetap); % second pole
% coefficient a, b must be real so the poles and zeros will be complex
% conjugate pairs (if any complex pole or zero exists)

%% pole-zero plot

zplane([z1, z2]', [p1, p2]');
%% determine H(z) = N(z)/D(z)
num = poly([z1,z2]); % numerator vector
den = poly([p1, p2]); % denominator vector

[h, f] = freqz(num, den, 256, Fs);
figure(1)
subplot(211) plot(f, abs(h)/max(abs(h)), 'linewidth', 2);
xlabel('Frequency(Hz)'); ylabel('Magnitude Response');
subplot(212) plot(f, angle(h)*180/pi, 'linewidth', 2);
xlabel('Frequency(Hz)'); ylabel('Phase Response');
```

After computing the pole and zero, the numerator and denominator polynomials are determined using the MATLAB/OCTAVE function poly. Then we computed the frequency

response of the filter using **freqz** function which was introduced to you in the previous lab.

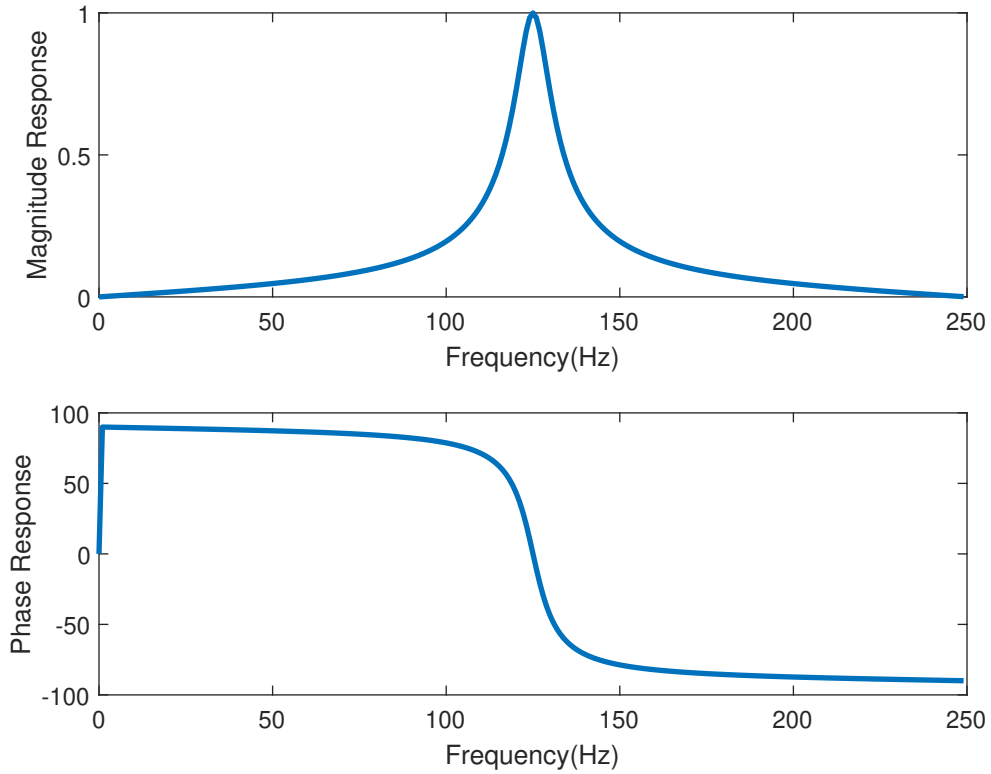


Figure. 7.4: Magnitude (normalized) and phase response of the designed bandpass filter

From the filter response it is seen that the filter has a narrow passband with a peak at frequency 125 Hz and at 0 and 250 Hz, the filter's magnitude is zero which were the design criteria. (verify that the filter has a 3 dB bandwidth of 10Hz!). Thus by pole-zero placement method, we can design simple filters like the one shown in this example.
Now we'll discuss how we can design filters using bilinear transformation in MATLAB/OCTAVE. Let us consider the design of a high pass filter from an analog prototype low pass filter. The normalized transfer function of an RC low pass filter is given by:

$$H(s) = \frac{1}{s+1} \tag{11}$$

We'll design a high pass filter assuming sampling frequency 150 Hz and cut-off frequency 30 Hz.
To design the digital high pass filter, we'll first calculate the pre-warped analog frequency. For 30 Hz cut-off frequency the corresponding digital frequency, $\omega_d = 2\pi \times 30 = 60\pi$ and the prewarped analog frequency,

$$\omega_a = \frac{2}{T_s} tan\Big(\frac{\omega_d T_s}{2}\Big) = 217.9628 rad/s$$

Then as we'll design a high-pass digital filter from a low pass filter, we'll use the required transformation. Following table summarizes the transformation and corresponding MATLAB/OCTAVE commands.

Filter Transformation	Transformation of s	MATLAB/OCTAVE Command
Low-pass to low-pass	$s = \frac{s}{\omega_a}$	lp2lp(Ap, Bp, wa)
Low-pass to high-pass	$s = \frac{\omega_a}{s}$	lp2hp(Ap, Bp, wa)
Low-pass to band-pass	$s = \frac{s^2 + \omega_0^2}{sW}$	lp2bp(Ap, Bp, w0, W)
Low-pass to band-stop	$s = \frac{sW}{s^2 + \omega_0^2}$	lp2bs(Ap, Bp, w0, W)

Figure. 7.5: Transformation from Low pass filter to other filters and corresponding MATLAB/OCTAVE Commands. Here Ap and Bp are the numerator and denominator vector of lowpass prototype respectively. wa is the cutoff frequency of the lowpass or highpass filter, w0 is the center frequency and W is the bandwidth of the bandpass or bandstop filter.

After this transformation of s, we'll substitute $s = \frac{2}{T_s}\frac{1 - z^{-1}}{(1 + z^{-1}}$ to complete the bilinear transformation. This is done in MATLAB/OCTAVE by using the function bilinear. Syntax for the function is

$$[a,b] = \text{bilinear}(A, B, Fs)$$

Here, A and B are the numerator and denominator vectors of the analog filter respectively. a and b are the numerator and denominator vectors of the digital filter respectively. In other words, a is the feedforward coefficient vector and b is the feedback coefficient vector of the digital filter. Fs is the sampling frequency. After obtaining H(z) we'll use freqz to determine the frequency response of the filter. The code is given below:

Listing 2: Design of a high pass filter from low pass prototype filter using Bilinear Transformation

```
clc;clear all;close all;
% Designing a high pass filter from an analog low pass filter
% using bilinear transform
Fc = 30; % Cut-off frequency
Fs = 150; % Sampling frequency
Ts = 1/Fs; % Sampling period
% Filter pre-warped frequency calculation
wd = 2*pi*Fc; % Digital frequency
wa = 2/Ts*tan(wd*Ts/2); % pre-warped frequency
% analog filter coefficients H(s) = 1/(1+s)
num = [1]; % numerator coefficients
```

```
den = [1 1]; % denominator coefficients
% filter transformation from low pass to high pass
[A, B] = lp2hp(num, den, wa); % for MATLAB use this code, comment out
%the next line
[A, B] = sftrans(num, den, 1, wa, 1); % for Octave use this code,
%comment out the previous line
[a, b] = bilinear(A, B, Fs); % Bilinear Transformation
% Frequency response
N = 512; % FFT point number
[hz, f] = freqz(a, b, N, Fs); % computing frequency response
phi = 180*unwrap(angle(hz))/pi; % Unwarping the phase angle
subplot(211)
plot(f, abs(hz), 'linewidth', 2);
ylabel(Magnitude Response);xlabel(Frequency(Hz));
subplot(212)
plot(f, phi, 'linewidth', 2);
ylabel(Phase Response); xlabel(Frequency(Hz));
```

The output response of this digital filter is:

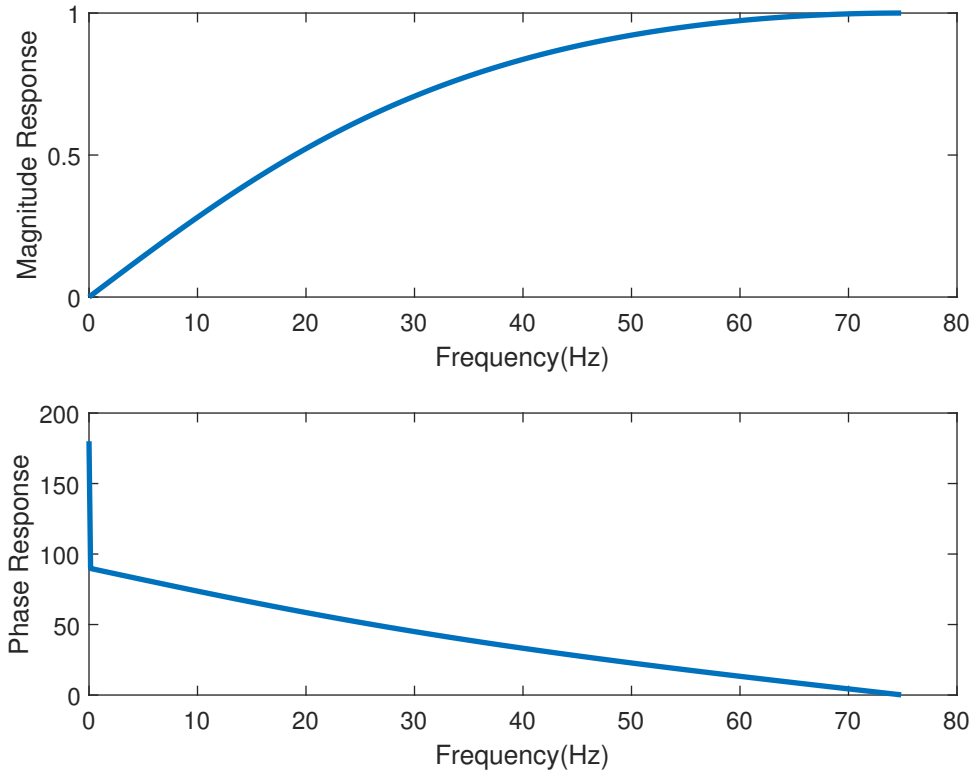


Figure. 7.6: Magnitude and Phase response of the high pass filter.

From figure(**??**, we see that the filter acts a high pass filter with cut-off frequency 30Hz

as it attenuates signals whose frequency is less than 30Hz.

1. Design a notch filter using pole-zero placement method with following specifications:

   - notch frequency 50 Hz
   - 3 dB width of the notch ±5 Hz
   - sampling frequency 500 Hz

   Show the magnitude and phase response of the filter. Consider a signal, $x(t) = 2\cos(60\pi t) + \cos(100\pi t)$. If x[n] is the signal obtained from x(t) by sampling it at a frequency 500 Hz, show the output signal after x[n] is passed through your designed notch filter.

2. Design a bandstop filter with the following property using pole-zero placement method.

   - Center frequency, $\Omega_0 = \dfrac{\pi}{10}$ (complete attenuation)
   - Bandstop width, $\Omega_w = 2\Omega_{\text{cf}} = \dfrac{\pi}{20}$

   Show the frequency response of the filter and mention the filter coefficients.

3. A discrete bandpass filter with following specifications is to be designed using bilinear transformation from a low pass filter prototype.

   - passband, 200-300 Hz
   - sampling frequency, 2KHz
   - filter order, N = 2

   The transfer function of prototype low pass filter is $H(s) = \dfrac{1}{s+1}$. Determine the filter coefficients and show the frequency response of the filter.

LAB 8:
FIR Filter Design

Objectives

The main objectives of this lab are:

- To understand the basic idea of FIR Filter
- To design digital FIR filters from specification by windowing method

PART 1

In our last lab, we covered the basic idea of a filter and how to design an IIR (Infinite impulse response) filter by pole-zero placement and bilinear transform methods. In this lab, we'll discuss about an FIR (Finite Impulse Response) filter and how to design FIR filters.

FIR and IIR Filters

As mentioned in the previous lab, a general linear and time invariant causal digital filter with input x[n] and output y[n] can be described by linear constant coefficient difference equations of the form:

$$y[n] = -\sum_{k=1}^{N} b_k y[n-k] + \sum_{k=0}^{M} a_k x[n-k] \tag{1}$$

where a_k and b_k are the coefficients which characterizes the filter. a_k's are the feedforward coefficients (affect the input signal) and b_k's are the feedback coefficients (affect the output signal). In case of impulse response, the input is a delta function. If we denote the impulse response h[n], equation (1) becomes

$$h[n] = -\sum_{k=1}^{N} b_k h[n-k] + \sum_{k=0}^{M} a_k \delta[n-k] \tag{2}$$

If we take z-transform of equation (1), we get the system function as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_k \sum_{k=0}^{M} z^{-k}}{1 + \sum_{k=1}^{N} b_k z^{-k}} \tag{3}$$

The system function of the filter has M zeros and N poles. In case of an filter the feedback coefficients $b_k's$ are zero and so h[n] and H(z) are given by:

$$\begin{aligned} h[n] &= \sum_{k=0}^{M} a_k \delta[n-k] \\ H(z) &= \sum_{k=0}^{M} a_k z^{-k} \end{aligned} \tag{4}$$

Since in case of FIR filter, (4) is not recursive, the impulse response has finite duration M and hence the name finite impulse response filter. The FIR filter in equation(4) has filter length = M+1. FIR Filters can have exactly linear phase response and very simple to implement.

FIR Filter Design using Window Method

Frequency response of a filter $H_D(\omega)$ is related to its impulse response $h_D[n]$ by the Inverse Discrete time Fourier Transform:

$$h_D[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_D(\omega)e^{j\omega n}d\omega \tag{5}$$

The subscript 'D' is used to denote the ideal behavior. So if $H_D(\omega)$ is known, $h_D[n]$ can be determined from equation (5). Let us illustrate this with an example of Low Pass Filter (LPF). Consider an ideal LPF with cut-off frequency ω_c (normalized) and frequency response $H_D(\omega)$ given by:

$$H_D\left(e^{jw}\right) = \begin{cases} 1, |\omega| \leq \omega_c \\ 0, \omega_c < |\omega| \leq \pi \end{cases} \tag{6}$$

Impulse response, $h_D[n]$ for this filter can be found by:

$$\begin{aligned} h_D[n] &= \frac{1}{2\pi}\int_{-\pi}^{\pi} 1 \times e^{j\omega n}d\omega \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{j\omega n}d\omega \\ &= \frac{2f_c \sin(n\omega_x)}{n\omega_c}, n \neq 0, -\infty \leq n \leq \infty \\ &= 2f_c, n = 0 \end{aligned} \tag{7}$$

Now as $h_D[n]$ is infinite duration signal, its z transform $H_D(z)$ will also be of infinite duration:

$$\begin{aligned} H(z) &= \sum_{n=-\infty}^{\infty} h(n)z^{-n} \\ &= \cdots + h(-2)z^2 + h(-1)z^1 + h(0) + h(1)z^{-1} + h(2)z^{-2} + \cdots \end{aligned} \tag{8}$$

It is apparent from equation (7) and (8) that $h_D[n]$ is both non-causal and infinite in duration. Hence FIR filter given in equation (7) is not practically realizable.

An obvious solution is to truncate the filter coefficients to $i = -\frac{M}{2}, \cdots 0, \cdots + \frac{M}{2}$, then shifting to the right by $\frac{M}{2}$ gives the filter coefficient for an M^{th} order physically realizable Low-Pass Filter. However this introduces undesirable ripples and overshoot (Gibbs phenomenon). Figure (8.1) illustrates the effects of truncating the filter coefficients. Increasing the filter length does not reduce the problems.

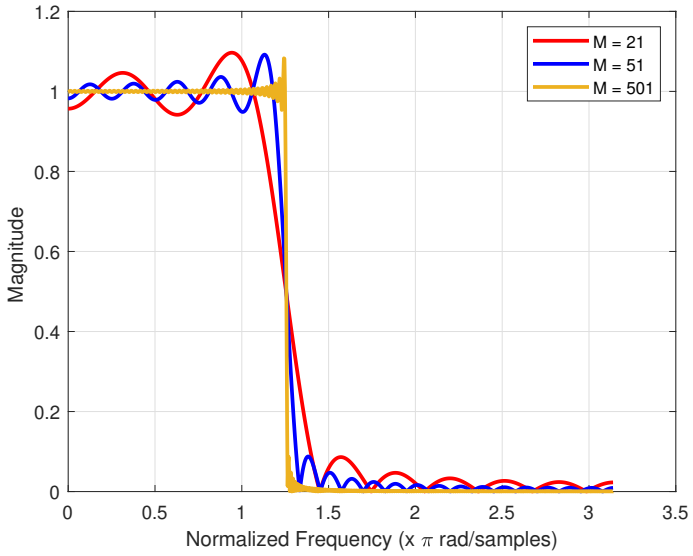


Figure. 8.1: Effects on the frequency response of truncating the ideal impulse response to M = 21, 51 and 501 coefficients.

Window method is developed to get rid of this undesirable Gibbs oscillation in the passband and stopband of the designed FIR filter. As Gibbs oscillations are the result of abrupt truncation of the infinite length filter coefficient sequence. Hence a window function is required which is symmetrical and can gradually weight the desired FIR coefficients down to zeros at the both ends for the range of $-M \leq n \leq M$. Applying the window sequence to the filter coefficients gives

$$h_w[n] = h[n]\dot{w}[n] \tag{9}$$

where w[n] is the window function. Desirable frequency characteristics can be obtained by making a better selection for the window, w[n]. There are several window functions are used for this purpose. Some of them are as follows:

1. Rectangular window:

$$w_{rec}[n] = 1, \quad 0 \leq n \leq M-1$$

2. Triangular (Bartlett) window:

$$w_{tri}[n] = \begin{cases} \dfrac{2n}{M-1}, 0 \leq n \leq \frac{M-1}{2} \\ 2 - \dfrac{2n}{M-1}, \frac{M-1}{2} \leq n \leq M-1 \end{cases}$$

3. Hanning window:

$$w_{han}[n] = 0.5 - 0.5\cos\left(\frac{n\pi}{M}\right), 0 \leq n \leq M(\text{total M+1 points})$$

4. Hamming window:

$$w_{ham}[n] = 0.54 - 0.46\cos\left(\frac{n\pi}{M}\right), 0 \leq n \leq M$$

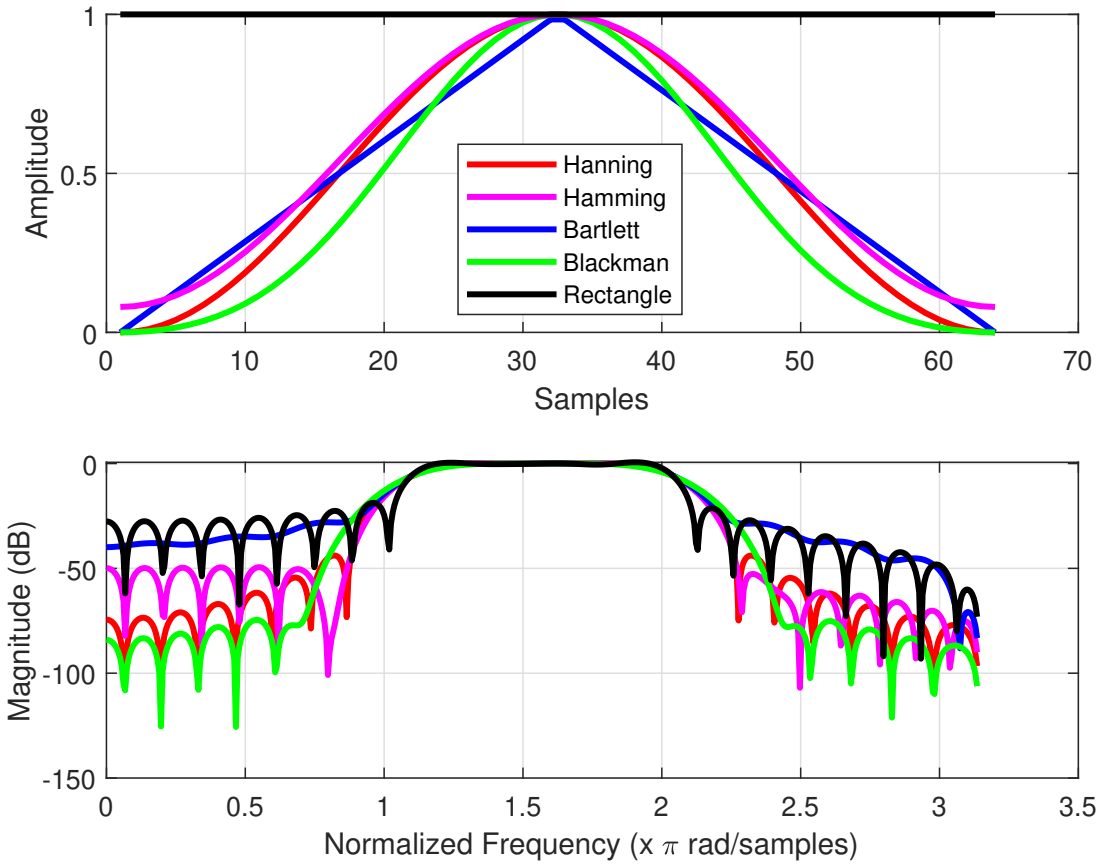


Figure. 8.2: Shapes of Window functions for the case of M = 64 (Upper Figure) (b) Frequency response of different window functions

5. Blackman window:

$$w_{\text{black}}[n] = 0.42 - 0.5\cos\left(\frac{n\pi}{M}\right) + 0.08\cos\left(\frac{2n\pi}{M}\right), 0 \leq n \leq M$$

Shapes of different window functions in time domain is given in figure (8.2) for M = 64. In window-based filter design, there are two key frequency domain effects important to the design: the transition band roll-off and the passband and stopband ripple. Two parameters in the spectrum influence these filter properties.

- The filter's roll-off is related to the width of the center lobe of the window's magnitude spectrum
- The ripple is influenced by the ratio of the mainlobe amplitude to the first sidelobe amplitude (or difference in dB scale).

These two window spectrum parameters are not independent. Theoretical values of these two parameters for the aforementioned windows are given in Table 1.

Table 1

Window Name	Transition Width Δf (Normalized)	Main Lobe relative to side lobe (dB)	Stopband Attenuation (dB) (maximum)	Passband Ripple (dB)
Rectangular	$\frac{0.9}{M}$	13 dB	21	0.7416
Hanning	$\frac{3.1}{M}$	31 dB	44	0.0546
Hamming	$\frac{3.3}{M}$	41 dB	53	0.0194
Blackman	$\frac{5.5}{M}$	57 dB	75	0.0017

PRELAB TASKS

1. Consider an FIR Filter:

   $$y[n] = 0.1x[n] + 0.25x[n] + 0.2x[n-2]$$

   Determine the transfer function, feedforward coefficients, filter length and impulse response of the filter.

2. What are the advantages of FIR filters over IIR filters?

3. Why is window method necessary for designing FIR filters?

Part 2
FIR Filter Design in MATLAB/OCTAVE

- Realization of FIR filters in MATLAB/OCTAVE
- Functions to use: **fir(), freqz()**

Instructions

- Organizing files and folder properly for later use.
- Make a unique named folder for this lab like **EEE3218_SP20_A1_180105001** under any drive other than C drive.
- For every week make a subfolder to keep the all the tasks in a specific week separated from other tasks.
- Always try to work in the folder specifically created for the current week.
- Follow MATLAB/OCTAVE naming convention for giving a meaningful name before saving it in MATLAB/OCTAVE.
- While running a code be judicious in choosing *add to path* or *change directory*. Use add to path when you are using a file from different folder.

In this section, we'll discuss the design procedure of an FIR filter by window method. The steps are as follow:

- We need to specify the ideal or desired frequency response of filter, $H_D(\omega)$.
- Then we need to compute $h_D[n]$ of the desired filter by evaluating inverse Fourier Transform. For the standard frequency selective filters, the expressions for $h_D[n]$ are summarized in Table 2.
- We need to select a window function that satisfies the passband or attenuation specifications and determine the number of filter coefficients using the appropriate relationship between the filter length and the transition width, δf (expressed as a fraction of the sampling frequency).
- Then w[n] is obtained for the chosen window function and the values of the actual FIR coefficients, h[n] by multiplying $h_D[n]$ by w[n]:

$$h[n] = h_D[n]w[n]$$

Table 2: Ideal Impulse Response for different Filters

Filter Type	Ideal Impulse Response, $h_D[n]$	
	$h_D[n], n \neq 0$	$h_D[0]$
Low Pass	$\frac{2f_c \sin(n\omega_c)}{n\omega_c}$	$2f_c$
High Pass	$-\frac{2f_c \sin(n\omega_c)}{n\omega_c}$	$1 - 2f_c$
Band Pass	$\frac{2f_2 \sin(n\omega_2)}{n\omega_2} - \frac{2f_1 \sin(n\omega_1)}{n\omega_1}$	$2(f_2 - f_1)$
Band Stop	$\frac{2f_1 \sin(n\omega_1)}{n\omega_1} - \frac{2f_2 \sin(n\omega_2)}{n\omega_2}$	$1 - 2(f_2 - f_1)$

In the table 2, ω_c (f_c) is the cut-off frequency for low and high pass filters. ω_1 and ω_2 are two cut-off frequencies for a bandpass and bandstop filters. The above procedures will be followed while calculating filter coefficients by hand. However in MATLAB, fir1() command is used to design an FIR filter by windowing method following the above procedures. The syntax for this function is:

fir1(n, wn, ftype, window)

where,

n is the order (number of zeros) of the FIR Filter
wn is the frequency requirement (passband edge, stopband edge) of the filter. wn is a number between 0 and 1, where 1 corresponds to the Nyquist frequency, half the sampling frequency
ftype is the type of filter we'll use for example 'low', 'high', 'bandpass' and 'stop'
window is the type of window we want to use.

Let us consider an example to show the application of FIR filter design procedure. We'll design a low pass FIR filter with the following specifications:

- Passband edge frequency 1.5 KHz
- Transition Width 0.5 KHz
- Stopband Attenuation >50 dB
- Sampling Frequency 8 KHz

From table 1, we find that Hamming or Blackman window provides stopband attenuation >50 dB and hence can be used to design this filter. We'll use the Blackman window. Normalized transition width, $\Delta f = \frac{0.5}{8} = 0.0625$. So filter order, $M = \frac{5.5}{0.0625} = 88$. Due to the smearing effect of the window on the filter response, the cutoff frequency of the

resulting filter will be different from that given in the specifications. To account for this, we'll use and f_c that is centered on the tranistion band:

$$f_c^{'} = f_c + \Delta F/2 = 1.5 + 0.5/2 = 1.5 + 0.25 = 1.75\text{kHz}$$

Following is the code for computing filter coefficients for this filter:

Listing 1: Design of a low pass FIR filter using Blackman Window

```
clc;
clear all;
close all;

Fs = 8e3; % sampling frequency
TW = 0.5e3; % Transition width
PBE = 1.5e3; % Passband Edge frequency

delf = TW/Fs; % normalized transition width
M = round(5.5/delf); % Filter order for Blackman filter
corner = PBE+TW/2; % corner frequency = Passband_edge+transition_width/2
wn = 2*pi*corner/Fs; % converting wn to 2*pi range
a = fir1(M, wn/pi, 'low', blackman(M+1)); % filter coefficients
% M = order of the filter
% wn = passband edge frequency
% 'low' = lowpass filter
% blackman (M+1) = M+1 length blackman window
% a = filter coefficients
N = 512; % number of points for calculating frequency response
[h,f] = freqz(a, 1, N, Fs);
figure(1)
subplot(211)
%plot(f, abs(h), 'linewidth', 2);
plot(f, 20*log10(abs(h)), 'linewidth', 2);
xlabel(Frequency (Hz));
ylabel(Magnitude (dB));
subplot(212)
plot(f, unwrap(angle(h)), 'linewidth', 2);
xlabel(Frequency (Hz));
ylabel(Angle (Degree));
```

The magnitude and phase response of the filter is given in the following figure:

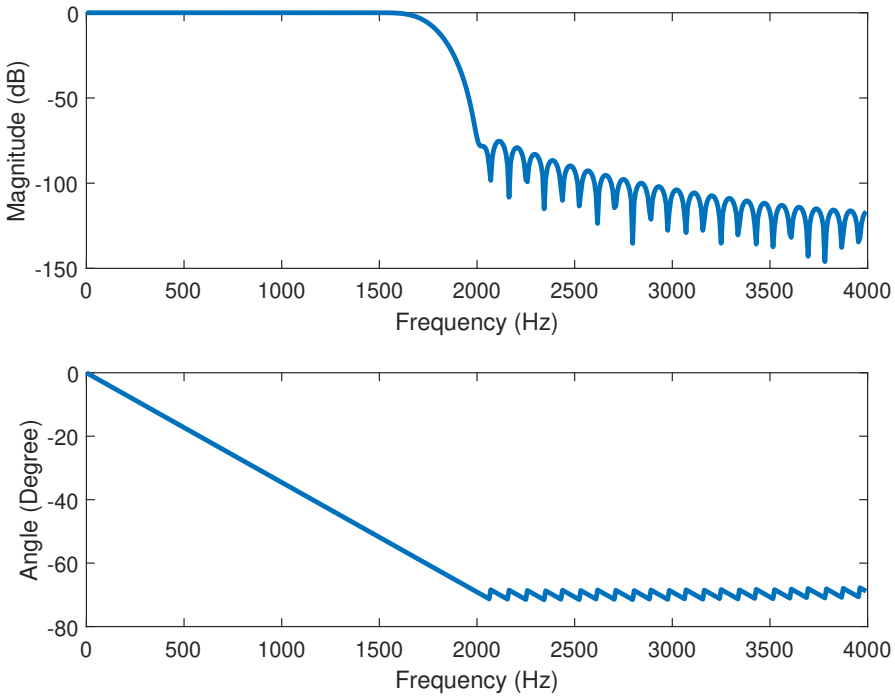


Figure. 8.3: Magnitude and Phase response of the designed FIR filter.

From the figure(8.3), the magnitude response falls gradually after transition band edge and there are ripples with attenuation > 50 dB in the stopband. The phase reponse of the FIR filter is observed to be linear as expected. The same filter for the above specifications can also be designed by Hamming window.

POST LAB TASKS

1. Plot the magnitude and phase responses for a 50th order Highpass filter assuming a sampling frequency of 5KHz and a cut-off frequency of 1KHz for each of the windows mentioned in the lab manual.

2. Design an FIR bandpass filter to meet the following specifications:

   - passband 150-250 Hz
   - transition width 50 Hz
   - passband ripple 0.05 dB
   - stopband attenuation 50 dB
   - sampling frequency 1 KHz

   Choose a suitable window method and explain why your reasoning of choosing that particular window. Plot the magnitude and phase response of the FIR filter.

References

The following books have been consulted while preparing this Lab Manual:

- J.G Proakis and D.G Manolakis, **Digital Signal Processing, Principles, Algorithm and Applications**, 3rd ed., India: Patience-Hall, 2000.
- Alan V.Oppenheim and Ronald W. Schafer, **Digital Signal Processing**.
- Li Tan, **Digital Signal Processing: Fundamentals and Applications**
- J.G Proakis and V.K. Ingle, **Digital Signal Processing Using MATLAB**.

Acknowledgement

1. Prepared By: Bejoy Sikder

   Lecturer, Department of EEE, AUST.

2. Special Thanks to:

   i) Prof. Dr. A.K.M Baki

   Professor, Department of EEE, AUST

   ii) Monjur Morshed

   Associate Professor, Department of EEE, AUST

   iii) Muhammad Jakaria Rahimi

   Associate Professor, Department of EEE, AUST

   iv) Md. Adnan Quaium

   Assistant Professor, Department of EEE, AUST