Department of Electrical and Computer Engineering

ELEC 442 Discrete-Time Signal Processing

Lab Manual

William E. Lynch and Maria Amer

June 2019

Table of Contents

Submission Instructions

- For each lab problem, create MATLAB script files.
- Include your name and ID as comments in each script file.
- For each lab problem, submit as part of the written lab report, the script file and any required plots.
- It is suggested to save the script file under meaningful name, such as `Lab1_Prob1.m`.

Lab 1. Linear Constant Coefficient Difference Equations (LCCDE)

Objectives

- To learn how to use MATLAB to generate the output of the LTI systems that are represented using Linear Constant Coefficient Difference Equation (LCCDE).

Introduction

Linear Constant Coefficient Difference Equation (LCCDE) is the equation that describes the relationship between the input and the output of a discrete time LTI system. It describes the operation of the system in the time domain and provides a method for computing the response of a system $y[n]$, to an arbitrary input $x[n]$. An *N*th-order LCCDE is expressed as

Equation 1-1:

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{m=0}^{M} b_m x[n-m]$$

$$y(n) = \frac{1}{a_0}\left\{\sum_{m=0}^{M} b_m x[n-m] - \sum_{k=1}^{N} a_k y[n-k]\right\}$$

With the initial conditions $y(-1) = y_1,\ y(-2) = y_2,\ \ldots,\ y(-N) = y_N$ and the coefficients a_k $(a_0 \neq 0)$ and b_m as constants that define the system. The relation in **Equation 1-1**, is a general linear relation between $x[n]$, the input, and $y[n]$, the output of the discrete time LTI system. We also say that N is the degree of the system. An example is the following difference equation

Equation 1-2:

$$\sum_{k=0}^{1} a_k y[n-k] = \sum_{m=0}^{0} b_m x[n-m] \quad , a_0 = 1, a_1 = -2, b_0 = 1$$

$$y[n] - 2y[n-1] = x[n]$$

where N (the order of the discrete difference equation) here is 1. Thus, it is a first-order difference equation.

If the difference equation has one or more terms a_k that are nonzero for $k > 0$, the difference equation is said to be *recursive.* On the other hand, if all of the coefficients $a_k, k > 0$ are equal to zero, the difference equation is said to be *non*-recursive [1][2].

LTI System as a Filter

A given discrete time LTI system that has been designed to achieve a particular purpose or perform a given operation, such as frequency selection or attenuation, is called a Filter. Such an LTR filter takes an input signal and produces an output signal. The relationship between the input and the output can be described using the LCCDE as in **Equation 1-1**. An LTI filter can be described also by its impulse response $h[n]$ or by its system function, also called transfer function, $H(z) = \sum_{n=-\infty}^{\infty} h[n]z^{-n}$. LTI filters can be classified into *finite impulse response (FIR) filters* and *infinite impulse response (IIR) filters* according to their impulse responses. The definitions of FIR and IIR filters will be explained below.

The Impulse Response

The impulse response is the output of a discrete time LTI system when the input is an impulse signal $\delta[n]$. We call the output, $y[n]$, as $h[n]$ when the input $x[n]$ is $\delta[n]$. This is shown in **Figure 1-1**.

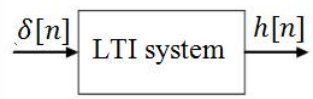


Figure 1-1: An LTI system can be classified, according to its impulse response, into an FIR or IIR system.

Finite Impulse Response (FIR) Systems

If the impulse response of an LTI system is of finite duration, the system is said to be a finite impulse response (FIR) system (filter). It is also called a *non-recursive system* because the output samples can be computed using only the current and previous input samples. An FIR filter can be represented by the following difference equation

Equation 1-3:

$$y[n] = \sum_{m=0}^{M} b_m x[n-m]$$

The output of an FIR filter, like other LTI systems, can also be computed using the convolution operation as follow

Equation 1-4:

$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{\infty} h[m]x[n-m] = \sum_{m=0}^{M} h[m]x[n-m]$$

which is equivalent to the difference equation (**Equation 1-3**) in this case by considering b_m as $h[m]$.

Infinite Impulse Response (IIR) Systems

If the impulse response of an LTI system is of infinite duration, the system is said to be an infinite impulse response (IIR) system (filter). It is also called a *recursive system* because it produces output samples based on the current and possibly previous input samples and the previous values of its own output. It should be noted that if the impulse response is absolutely summable, the filter will be stable. An IIR filter can be represented by the difference equation (**Equation 1-1**).

Solution of an LCCDE Using MATLAB

The MATLAB built-in function **`filter`** can be used to compute the response of LTI systems that are represented using LCCDE. The **`filter`** function with the following syntax [3]

```
y = filter(b, a, x)
```

filters the input data **`x`** using a rational transfer function defined by the numerator and denominator coefficients **`b`** and **`a`** (see **Equation 1-1**). It should be noted that the output signal generated from **`filter`** function has same length as the input signal.

The MATLAB built-in function **`conv`** can also be used to compute the response of LTI systems with impulse response **`h`** as follow [3] **`y = conv(x, h)`**

The length of the output signal using the above regular convolution will be as follow

```
length(x) + length(h) - 1
```

Example 1: Find the impulse response for a causal LTI system whose input $x[n]$ and output $y[n]$ are related by the difference equation

Equation 1-5:

$$y[n] - \frac{1}{2}y[n-1] = x[n]$$

Solution: This is a recursive equation, and the system is causal LTI. So, the initial rest condition can be applied ($y[n] = 0$ for all $n < 0$). The input signal is $x[n] = \delta[n]$, so we can compute $y[n]$ for $n \geq 0$ as follows

Equation 1-6:

$$y[0] = x[0] + \frac{1}{2}y[-1] = 1$$

$$y[1] = x[1] + \frac{1}{2}y[0] = \frac{1}{2}$$

$$y[2] = x[2] + \frac{1}{2}y[1] = \left(\frac{1}{2}\right)^2$$

$$y[3] = x[3] + \frac{1}{2}y[2] = \left(\frac{1}{2}\right)^3$$

and so on.

Thus, the solution can be written as

Equation 1-7:

$$y[n] = h[n] = \left(\frac{1}{2}\right)^n u[n]$$

The impulse response $h[n]$ is of infinite duration, so the system is infinite impulse response (IIR). Now we want to verify this using filter function in MATLAB. In order to obtain and plot impulse response of the system, we can write the code below:

```
a = [1 -1/2]; % denominator coefficients (a) according to LCCDE
b = [1]; %numerator coefficients (b) according to LCCDE
n = (-50 : 1 : 50)'; %time interval
impulse = n == 0; %a unit impulse signal
```

```
subplot(1, 2, 1);
stem(n, impulse);
title('Impulse Signal');
xlabel('n');
ylabel('\delta[n]');
y = filter(b, a, impulse); %y will be the impulse response of the system
subplot(1, 2, 2);
stem(n, y);
title('Impulse Response');
xlabel('n');
ylabel('y[n]');
```

Figure 1-2 shows the impulse response of the system which is a decreasing exponential function and is the same as the solution we obtained above.

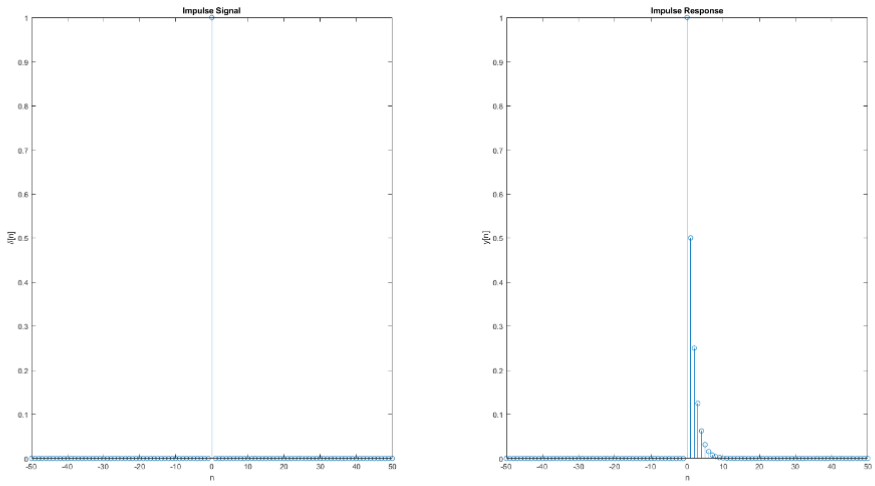


Figure 1-2: Unit Impulse (left) and Impulse Response of the system (right)

Lab Problems

➢ **Problem 1.**

(a) Consider a causal LTI system whose input $x[n]$ and output $y[n]$ are related by the LCCDE

Equation 1-8: $$\boldsymbol{y[n] = \frac{1}{4}y[n-1] + x[n]}$$

Determine the impulse response $h[n]$ and then determine $y[n]$ for $x[n] = u[n] - u[n-10]$.

(b) Let $0 \leq n \leq 50$. Use MATLAB **`conv`** (convolution) function to find the output $y[n]$ from part (a). **Hint**: If we want to use **`conv`** function we need to know the impulse response of the

system. Here we can use **`impz`** function (see MATLAB help [3]) or we can use **`filter`** function with the impulse as its input.

(c) Again let $0 \leq n \leq 50$. Use MATLAB **`filter`** command to find the output $y[n]$ from part (a). **Note**: plot the input signal and the output signal in the same figure for each part. Adjust the title, xlabel and ylabel accordingly.

(d) Explain any differences between using **`conv`** and **`filter`** commands.

➢ **Problem 2.**

Consider an LTI system whose input $x[n]$ and output $y[n]$ are related by the following LCCDE

Equation 1-9: $$\boldsymbol{y[n] = \frac{1}{2}y[n-1] - \frac{1}{4}y[n-2] + x[n] + 2x[n-1] + x[n-3]}$$

(a) Determine and plot the impulse response of the system over $0 \leq n \leq 100$, and determine the stability of LTI system from $h[n]$. **Hint:** An LTI system with impulse response $h[n]$ is stable if and only if $h[n]$ is absolutely summable.

(b) Determine and plot the output $y[n]$ over $0 \leq n \leq 200$, if the input to this system is

Equation 1-10: $$\boldsymbol{x[n] = [15 + 3\cos(0.2\pi n) - 15\sin(0.7\,\pi n)]u[n]}$$

Note: Plot the input signal and the output signal in the same figure for each part. Adjust the title, xlabel and ylabel accordingly.

➢ **Problem 3.**

A digital differentiator is given by

Equation 1-11: $$\boldsymbol{y[n] = x[n] - x[n-1]}$$

which computes a backward first-order difference of the input sequence. Implement this differentiator on the following sequences using **`filter`** function, plot the results and give your comments on the output results.

(a) $x[n] = 5(u[n] - u[n-30])$, (rectangle pulse)

(b) $x[n] = n(u[n] - u[n-20]) + (40-n)(u[n-20] - u[n-40])$, (triangle pulse)

(c) $x[n] = 15\sin\left(\frac{\pi n}{25}\right)(u[n] - u[n-100])$, (sinusoidal pulse)

Note: plot the input signal and the output signal in the same figure for each part. Adjust the title, xlabel and ylabel accordingly.

➢ **Problem 4.**

A common example of a digital signal processing application is the removal of noise. Let $x[n]$ be the input signal to the LTI system-1 in Figure 1-3, and assume its output signal $y[n]$ to be corrupted by a random noise $d[n]$ such that the noisy signal $v[n] = y[n] + d[n]$. The objective is to operate on $v[n]$ so to generate a signal $\hat{y}[n]$ which is a reasonable approximation to $y[n]$ and then recover the input signal $\hat{x}[n]$. A simple approach is averaging a number of input samples around the sample at instant n. For example, a three-point moving weighted average filter is given by

Equation 1-12: $$\hat{\boldsymbol{y}}[\boldsymbol{n}] = \frac{\mathbf{1}}{\mathbf{4}}(\boldsymbol{v}[\boldsymbol{n}-\mathbf{1}] + \mathbf{2}\boldsymbol{v}[\boldsymbol{n}] + \boldsymbol{v}[\boldsymbol{n}+\mathbf{1}])$$

In order to compute $\hat{y}[n]$ using **`filter`** function at first we change the variable $n \rightarrow n-1$, now we have:

Equation 1-13: $$\hat{\boldsymbol{y}}_{\boldsymbol{new}}[\boldsymbol{n}] = \hat{\boldsymbol{y}}[\boldsymbol{n}-\mathbf{1}] = 1/4(\boldsymbol{v}[\boldsymbol{n}-\mathbf{2}] + \mathbf{2}\boldsymbol{v}[\boldsymbol{n}-\mathbf{1}] + \boldsymbol{v}[\boldsymbol{n}])$$

After computing the $\hat{y}_{new}[n]$ using **`filter`** function we should consider the following equation to compute $\hat{y}[n]$

Equation 1-14: $$\hat{\boldsymbol{y}}[\boldsymbol{n}] = \hat{\boldsymbol{y}}_{\boldsymbol{new}}[\boldsymbol{n}+\mathbf{1}]$$

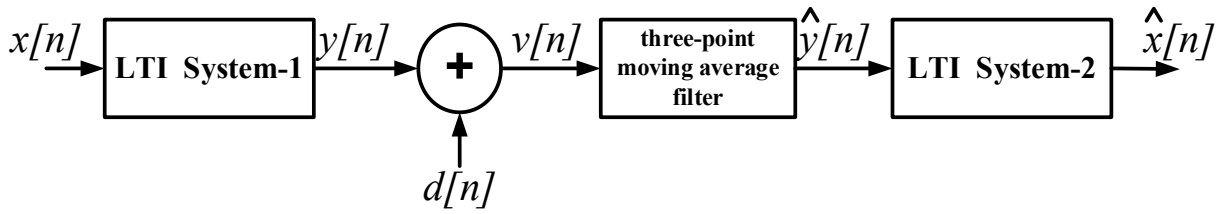


Figure 1-3: A recovering process for additive noise

Given the above system in Figure 1-3 where

- $x[n]$ is given by the following MATLAB code

```
%%=========================
Number_of_samples = 200;
Time_length = 20;
Ts = Time_length / (Number_of_samples - 1);
n = 0 : Ts : Time_length;
%real signal, non-periodic
X = ((0.9) .^ n) .* cos(2 * pi .* n);
%%=========================
```

- LTI system-1 is given by

Equation 1-15: $$\boldsymbol{y}[\boldsymbol{n}] - \frac{\mathbf{1}}{\mathbf{4}}\boldsymbol{y}[\boldsymbol{n}-\mathbf{1}] = \boldsymbol{x}[\boldsymbol{n}]$$

- $d[n]$ is a Gaussian noise with zero mean and 0.2 standard deviation.
- LTI system-2 is given by

Equation 1-16: $\hat{\boldsymbol{x}}[\boldsymbol{n}] = \hat{\boldsymbol{y}}[\boldsymbol{n}] - \frac{\mathbf{1}}{\mathbf{4}}\hat{\boldsymbol{y}}[\boldsymbol{n}-\mathbf{1}]$

(a) Determine and plot each of the following signals:

$x[n], y[n], d[n], v[n], \hat{y}[n], \hat{x}[n]$;

make sure that you plot all signal in the same figure using subplot command and adjust the title, xlabel and ylabel accordingly.

(b) Calculate the mean square error (MSE) between $x[n]$ and $\hat{x}[n]$ and the ratio of the MSE to the squared size of $x[n]$.

Hint 1: If we have an original signal $x[n]$ and a recovered signal $\hat{x}[n]$, both of length N, the mean square error is calculated as

Equation 1-17:

$$MSE = \frac{1}{N}\sum_{n=0}^{N-1}(x[n] - \hat{x}[n])^2$$

Also, the squared size of the $x[n]$ (energy) is calculated as

Equation 1-18:

$$S = \sum_{n=1}^{N}(x[n])^2$$

Hint 2: In order to generate a Gaussian noise with determined mean and variance we can use the following MATLAB code

```
Gaussian_noise = sqrt(v) * randn(size(signal)) + m;
```

where **`v`** is variance, **`m`** is mean of the Gaussian noise, and **`randn`** is a MATLAB function to generate normally distributed random numbers [3].

Hint 3: we can use the MATLAB built-in function **`filter`** to compute the output of LTI systems and Moving Average Filter in **Figure 1-3**.

Lab 2. The Fourier Transform and the Z-Transform

Objectives

- To introduce the Discrete Fourier Transform (DFT) as a signal analysis tool
- To see how Z-transform can be used to answer questions concerning stability and causality
- To see how Z-transform can be used to determine system output
- To study the use of Z-transform in the systems design process
- To see how Z-transform is used in the analysis for frequency response

Signal Acquisition

Prior to coming to the lab, you should acquire a real-life signal of some sort. Recall that a signal is a stream of data. The stream should be single channel and one dimensional. It should of course be a digital signal. You can use MATLAB or any other software to acquire a digital signal.

You should know the sampling rate and the unit of the samples (for example, if the signal is the output of a stereo amplifier to a speaker, the unit might be volts). If you cannot say what unit it is, then explain it as best you understand it.

The file containing your signal should be ASCII readable format (text). It should be such that it can be loaded into MATLAB using the "load" command. This means that the file can take one of the two following formats:

1. The number representing each sample is placed on a new line. Thus, there is only one number on each line (column vector).
2. The numbers representing the samples are all on the same line each separated by one or more spaces. This file contains one line (row vector).

The Fourier Transform: The Spectrum of a Signal

We learned how to express a discrete-time signal in the frequency domain using the Discrete-Time Fourier Transform (DTFT), see chapter 2 of Oppenheim and Schaefer [1]. The transform takes the following equation

Equation 2-1:

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

This transform must be done on pencil and paper and cannot be done using a digital computer because it is a function of a continuous-variable (the frequency ω). The Discrete Fourier Transform (DFT) is similar to the DTFT in **Equation 2-1**, but can be done on a digital computer since it is a function of a discrete-variable (the frequency index *k*). The DFT is defined as

Equation 2-2:

$$X(k) = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}nk} = \sum_{n=0}^{N-1} x[n]W_N^{nk}$$

for $k = 0, \dots, N-1$, where $W_N \triangleq e^{-j\frac{2\pi}{N}}$. The DFT *X(k)* can be considered as taking samples of the DTFT $X(e^{j\omega})$. The DFT is a different Fourier transform than the DTFT. On the other hand, the Fast Fourier Transform (FFT) is not a new Fourier transform but a method of evaluating the DFT that is computationally more efficient than the DFT. The Discrete Fourier transform (DFT) is performed in MATLAB by using the function **`fft`**. The output of the transform, which we denote by *X* is a complex sequence of length *N* samples representing the phase and magnitude information contained in the spectrum. The important point to note here is that the samples of the DFT spectrum are spaced in normalized frequency by

Equation 2-3: $\boldsymbol{\omega_k = \frac{2\pi k}{N} \quad for \;\; k = 0, 1, \dots, N-1}$

This means that the spectral width (frequency separation) of each DFT sample is

Equation 2-4: $\frac{2\pi}{N} \;\; rad/sec$

Now consider the following example from MATLAB help for the **`fft`** function [3].

➢ **Example 1.**

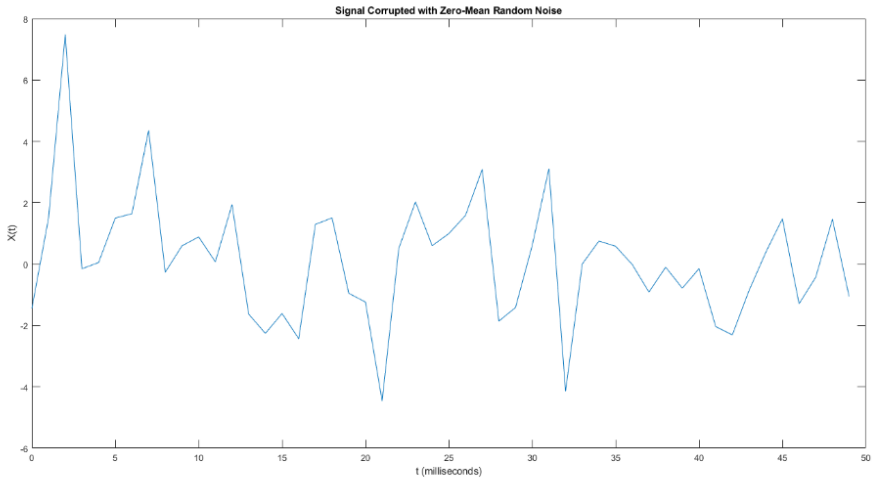
```
Fs = 1000;              % Sampling frequency
T = 1/Fs;               % Sampling period
L = 1500;               % Length of signal
t = (0:L-1)*T;          % Time vector

%Form a signal containing a 50 Hz sinusoid of amplitude 0.7 and
% a 120 Hz sinusoid of amplitude 1.
S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);

%Corrupt the signal with zero-mean white noise with a variance of 4.
X = S + 2*randn(size(t));

%Plot the noisy signal in the time domain.
plot(1000 * t(1 : 50), X(1 : 50))
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('t (milliseconds)')
ylabel('X(t)')

%Compute the Fourier transform of the signal.
Y = fft(X);

%Compute the two-sided spectrum P2.
%Then compute the single-sided spectrum P1 based on P2 and
% the even-valued signal length L.

P2 = abs(Y / L);
P1 = P2(1 : L / 2 + 1);
P1(2 : end - 1) = 2 * P1(2 : end - 1);

%Define the frequency domain f and plot the single-sided amplitude
% spectrum P1.
f = Fs * (0 : (L / 2)) / L;
figure, plot(f, P1)
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
```

As can be seen in the code above, a signal containing a 50 Hz sinusoid of amplitude 0.7 and a 120 Hz sinusoid of amplitude 1 is formed and then corrupted with zero-mean white noise with a variance of 4. The Fourier transform of the signal is computed using **`fft`** function. After computing Fourier coefficients the two-sided and single-sided spectrum will be computed. One of the most important points here is about the scaling of the resulted Fourier coefficients by $1/L$. L here is the length of the input signal. In general, to return FFT amplitudes equal to the amplitudes of the input signal we need to normalize FFT coefficients by the length of the input signal (the number of sample points of the input signal).

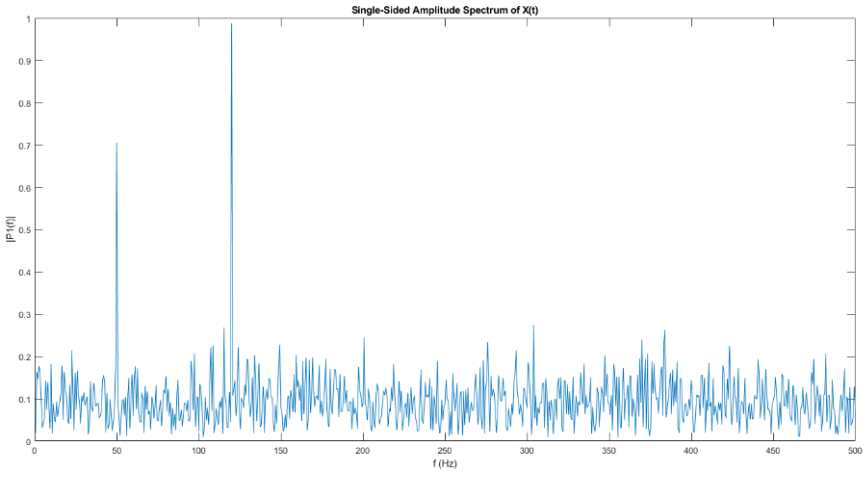
We also know that the Fourier transform of a real-valued signal is conjugate symmetric [2]. It means that the negative coefficients are conjugate of the positive as in

Equation 2-5: $$\boldsymbol{X(-k) = X^*(k)}$$

One half of the spectrum is the positive frequencies, and the other half is the negative. Since in the above example the single-sided amplitude spectrum is to be computed, the amplitude of each frequency (except DC) is going to be doubled to account for the contributions of data on the other side of the spectrum. The results of the above code has been shown in **Figure 2-1**.



(a)



(b)

Figure 2-1: Signal corrupted with zero-mean random noise (a), and single-sided amplitude spectrum (b)

Lab Problems I

➢ **Problem 1.**

Look in the folder "Data_for_DSP_Labs" and find the README file. Read this file carefully. It will tell you the names of several signals, names of the corresponding corrupted signals and a short description of each of the signals. Now, take the spectrum of each of the signals and their corruption. Plot the signals in the time domain and provide sketches of them in the frequency domain. Describe qualitatively what you see in the plots.

The Z-Transform

Laplace transform can be considered to be a generalization of the Fourier Transform (FT) in the continuous-time domain. Similarly, in the discrete-time domain, Z-transform is a generalization of the discrete-time Fourier Transform (DTFT). Z-transform plays an important role in both analysis and design of discrete-time systems. It provides another domain in which signals and systems can be investigated. The two–sided or bilateral Z-transform can be written as the following summation

Equation 2-6:

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

or $X(z) = \mathcal{Z}\{x[n]\}$. Two important properties of Z-transform are apparent from its definition. First, this transform is linear. Its second property enables the treatment of shifted sequences, i.e.

Equation 2-7:

$$x[n] \Leftrightarrow X(z) \quad \Rightarrow \quad x[n-k] \Leftrightarrow z^{-k}X(z)$$

Z-transform has many other valuable properties that make it a powerful analysis tool. One of the most important properties is the equivalence of convolution of two sequences and the multiplication of Z-transform in the transform domain, i.e.

Equation 2-8:

$$x[n] * y[n] \Leftrightarrow X(z)Y(z)$$

where $*$ denotes the convolution operation in time domain. By expressing the complex variable z in the polar form as $z = re^{j\omega}$, Z-transform has an interpretation in terms of Fourier transform. With z so expressed, the Z-transform becomes

Equation 2-9:

$$X(re^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n](re^{j\omega})^{-n} = \sum_{n=-\infty}^{\infty} x[n]r^{-n}e^{-j\omega m}$$

If $r = 1$, i.e. $|z| = 1$, the Z-transform is equivalent to the DTFT of the sequence $x[n]$.

In this experiment, we motivate the use of Z-transform by the role it plays in both design and analysis of discrete-time systems.

Lab Problems II

➢ **Problem 2.**

Knowledge of the impulse response of a system is very important in digital signal processing. In this problem, you will find the system transfer function of an unknown system by determining the system impulse response. Consider the following MATLAB function **usys**:

```
function output = usys(input)
output = zeros(1, length(input));
output(1) = input(1);
for i = 2 : length(input)
    output(i) = 1 / 2 * output(i - 1) + input(i);
end
```

(a) Show that the above function **usys** represents an LTI system.

(b) Find the impulse response of the unknown system defined by the above function **usys.** This can be done by using an impulse as the input to **usys**. Use the MATLAB command "stem" to show the results. (You should place **usys** in an m-file)

i. Generate an N-point sample input (take N=32 for this experiment).

$$x[n] = \delta[n] = \begin{cases} 1, & n = 0 \\ 0, & 0 < n \leq N-1 \end{cases}$$

ii. Using this input, determine the output of the system by calling the function **usys**.

(c) If we want to compute the impulse response of the system **usys** theoretically, it can be done easily as in **Equation 2-10**. Plot the resulted impulse response and compare it with the output of part a).ii of this problem 2.

Equation 2-10:

$$y[n] - \frac{1}{2}y[n-1] = x[n]$$

$$Y(z) - \frac{1}{2}z^{-1}Y(z) = X(z) \rightarrow H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1-\frac{1}{2}z^{-1}}$$

$$\rightarrow h[n] = \left(\frac{1}{2}\right)^n u[n], \qquad \text{for } ROC = |z| > \frac{1}{2}$$

(d) Do the following:

i. Define $x_1[n] = 2\delta[n-3]$ using 100 samples.

ii. Find the output $y_1[n]$ for the input $x_1[n]$ by calling the function **usys**.

iii. Using the convolution function **conv** in MATLAB, evaluate $y_2[n]$ as

$$y_2[n] = x_1[n] * h[n]$$

(e) Is this system stable? Explain why.

➢ **Problem 3.**

The focus in this problem is on demonstrating how the Z-transform answers questions concerning causality, stability and convergence. Consider the second-order system transfer function

Equation 2-11:

$$H(z) = \frac{1 - 1.7z^{-1}}{1 - 2.05\,z^{-1} + z^{-2}}$$

and an input sequence specified by

Equation 2-12:

$$X(z) = \frac{1}{1 - 0.9z^{-1}}$$

The input sequence is taken to have a *region of convergence (ROC)* given by $|z| > 0.9$. If we want to derive theoretically the inverse z-transform of $H(z)$ associated with the different regions of convergence, we can study the following equation:

Equation 2-13:

$$H(z) = \frac{1 - 1.7z^{-1}}{1 - 2.05z^{-1} + z^{-2}} = \frac{1 - 1.7z^{-1}}{(1 - 1.25z^{-1})(1 - 0.8z^{-1})}$$

$$\rightarrow H(z) = \frac{A}{(1 - 1.25z^{-1})} + \frac{B}{(1 - 0.8z^{-1})}$$

$$\rightarrow A = -1, B = 2$$

$$\rightarrow H(z) = \frac{-1}{(1 - 1.25z^{-1})} + \frac{2}{(1 - 0.8z^{-1})}$$

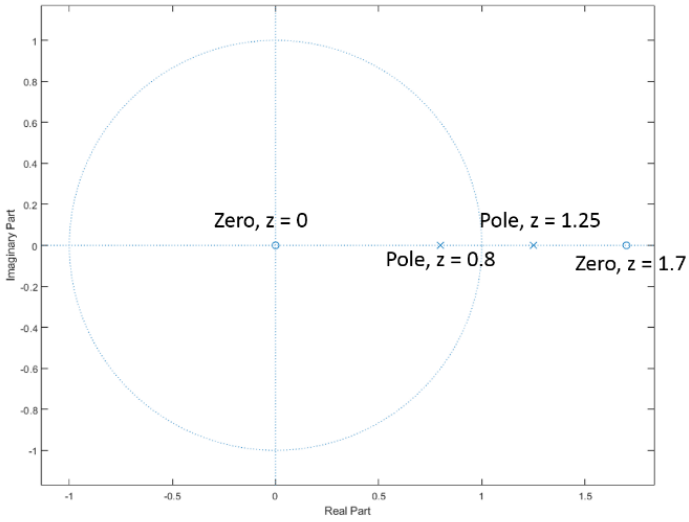


Figure 2-2: Pole-Zero locations of the system

If ROC is $|z| < 0.8$ it does not include unit circle (the Fourier Transform does not exist) therefore, the system is not stable, and as the ROC is not right sided the system is not causal.

If ROC is $0.8 < |z| < 1.25$ it includes unit circle (the Fourier Transform exists) therefore, the system is stable and as the ROC is a ring, therefore the system is not causal.

If ROC is $|z| > 1.25$ it does not include unit circle (the Fourier Transform does not exist) therefore, the system is not stable and as the ROC is right sided, the system can be causal.

(a) Use MATLAB built-in function **`zplane`** to plot the true pole/zero locations.

(b) If we want to calculate the output of the system to the input sequence corresponding to $X(z)$ with the stable sequence corresponding to $H(z)$ we will have

Equation 2-14:

$$Y(z) = X(z)H(z) = \frac{1 - 1.7z^{-1}}{(1 - 1.25z^{-1})(1 - 0.8z^{-1})(1 - 0.9z^{-1})}$$

$$\rightarrow Y(z) = \frac{A}{(1 - 1.25z^{-1})} + \frac{B}{(1 - 0.8z^{-1})} + \frac{C}{(1 - 0.9z^{-1})}$$

$$\rightarrow Y(z) = \frac{-3.57}{(1 - 1.25z^{-1})} + \frac{-16}{(1 - 0.8z^{-1})} + \frac{20.43}{(1 - 0.9z^{-1})}, ROC: 0.9 < |z| < 1.25$$

$$\rightarrow y[n] = -3.57 \times -1 \times (1.25)^n u[-n - 1] - 16 \times (0.8)^n u[n] + 20.43 \times (0.9)^n u[n]$$

Generate the convolution of the input sequence for *X(z)* with the stable sequence(s) corresponding to *H(z)* using the MATLAB function **conv**. Consider an input sequence of length *N = 10* samples and impulse response of length *N = 250* samples. Then compare these results with the theoretical results of the above analysis.

The Frequency Response from the Z-Transform

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

Equation 2-15: $$\boldsymbol{H(e^{j\omega}) = H(z)|_{z=e^{j\omega}}}$$

Using this methodology, it is possible to sketch the frequency response of the system that is related to its poles and zeros. Here we will only consider the magnitude response, while phase response can be considered in a similar way. Let $H(z)$ be written in its factorized form

Equation 2-16:

$$H(z) = \frac{C(z - b_1)(z - b_2) \dots (z - b_M)}{(z - a_1)(z - a_2) \dots (z - a_N)}$$

Then, the magnitude response can be written as

Equation 2-17:

$$|H(z)| = \frac{|C||(e^{j\omega} - b_1)||(e^{j\omega} - b_2)| \dots |(e^{j\omega} - b_M)|}{|(e^{j\omega} - a_1)||(e^{j\omega} - a_2)| \dots |(e^{j\omega} - a_N)|}$$

A geometrical interpretation can be formed by evaluating the magnitude response of an arbitrary term $(e^{j\omega} - z_i)$. The point $e^{j\omega}$ lies on the unit circle at an angle of ω as in Figure 2-3. The location of $z_i = |z_i|e^{j\theta}$ is also shown in the figure. The quantity of interest is the magnitude of the difference between these two vectors as shown in Figure 2-3.

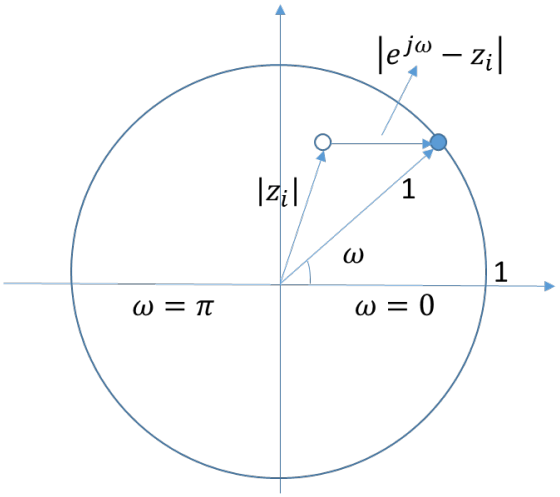


Figure 2-3: Geometric interpretation of the contribution of the magnitude response due to a single root at an arbitrary frequency.

It is evident that as the vector $e^{j\omega}$ rotates from 0 to 2π in a counter-clock-wise direction, the magnitude of the difference vector $(e^{j\omega} - z_i)$ also varies. The implication of the geometric representation is that we can evaluate the magnitude response due to the root z_i as the length of this difference vector at different frequencies in the range $[0, 2\pi]$.

In Figure 2-3, at $\omega = 0$, the magnitude of the difference vector is some finite value. As ω starts to increase, this value starts decreasing and is minimum when $\omega = \theta$, the angle of the root z_i. If we continue to increase ω, the difference magnitude again starts to increase until $\omega = 2\pi$ at which point everything is repeated again.

From this geometrical interpretation, the idea of periodicity of DTFT is evident. That is, if we continue to rotate $e^{j\omega}$ around the unit circle beyond 2π, the magnitude $(|e^{j\omega} - z_i|)$ starts to repeat. Each such revolution of the magnitude around the unit circle corresponds to a single period which leads to the idea of periodicity of the DTFT.

The example of Figure 2-3 illustrates how the graphical representation can be generalized to represent any arbitrary transfer function $H(z)$. The magnitude response can be represented as the product of individual difference vectors for the zeros divided by the product of difference vectors for the poles at each frequency.

At this time, several useful observations relating the magnitude response and pole/zero locations can be made. Zeros close to the unit circle cause the DTFT magnitude dip in the region near the zeros and the minimum value of the dip occurs at the angle of the zero. If the zero is on the unit circle, the DTFT magnitude is zero for that frequency (angle of zero). Similarly, the poles close to the unit circle force a peak in the magnitude response at frequencies close to the angle of the poles. With these observations in mind, we are able to generate a rule for sketching the magnitude response of $H(z)$. As we move around the unit circle the magnitude peaks when we pass close to a pole and dips when we pass near a zero. The sharpness of the peaks and valleys depend on the closeness of the poles and zeros on the unit circle. Poles and zeros far from the unit circle do not affect the magnitude response significantly. Consider the following example:

- **Example 2.** Given an all-pole filter (IIR) transfer function

Equation 2-18: $$H(z) = \frac{1}{1-0.5z^{-1}+0.2z^{-2}-0.1z^{-3}+0.007z^{-4}+0.14z^{-5}+0.15z^{-6}}$$

i. Find its magnitude response using the MATLAB function **`freqz`**. From the magnitude response what can be said about the pole locations?

ii. From the magnitude response, try to sketch the pole/zero plot.

iii. Using **`zplane`** function, plot the true pole/zero locations.

```
b = [1]; %numerator coefficients (b) according to LCCDE
a = [1 -0.5 0.2 -0.1 0.007 0.14 0.15]; %denominator coefficients (a)
according to LCCDE
[h, w] = freqz(b, a, 'whole', 2001); %computing the frequency response
of the filter
plot(w / pi, 20 * log10(abs(h)));
xlabel('Normalized Frequency (\times\pi rad/sample)')
ylabel('Magnitude (dB)')
figure, zplane(b, a); %Zero-Pole plot
```

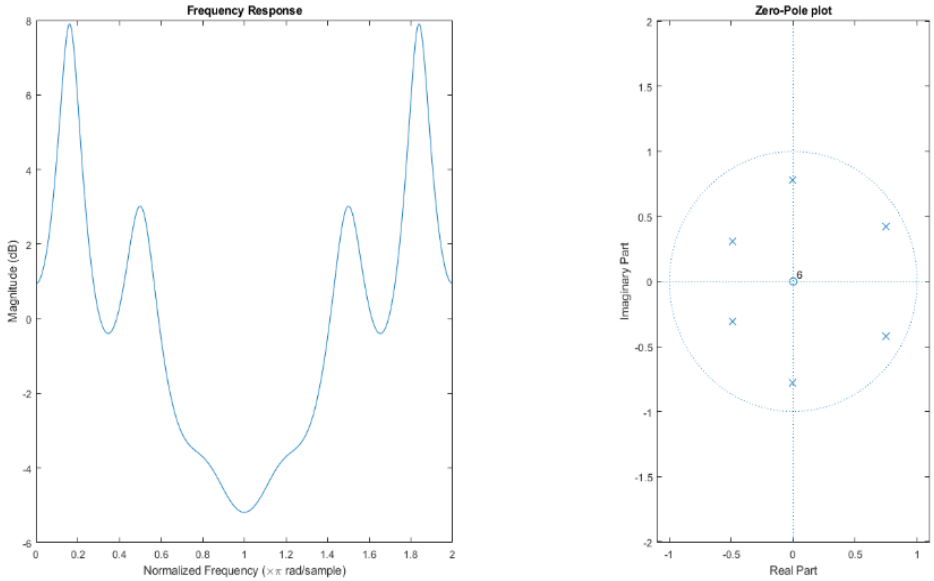


Figure 2-4: The relation between magnitude of frequency response and Zero-Pole plot

In **Figure 2-4** (left), it is obvious that we have two strong peaks around (0.16π *and* 1.84π) which informs us about two poles near the unit circle for those two angles. We also have two smaller peaks at approximately ($\pi/2$ *and* $3\pi/2$) which informs us about the existence of two other poles approximately near the unit circle. And if we zoom in, we can see two small local maximum in the magnitude of frequency response approximately at (0.8π *and* 1.2π) which relate to those two poles which are located far away from unit circle.

In order to find the poles for this system, we can also use the roots MATLAB built-in function as follow:

```
a = [1 -0.5 0.2 -0.1 0.007 0.14 0.15]
roots(a)
```

The results will be as follow (which are exactly the same as poles location in **Figure 2-4**)

```
 0.7496 + 0.4188i
 0.7496 - 0.4188i
-0.0090 + 0.7803i
-0.0090 - 0.7803i
-0.4906 + 0.3056i
-0.4906 - 0.3056i
```

Lab Problems III

➢ **Problem 4.**

(a)For the all-zero (FIR) filter transfer function given by H(z)

Equation 2-19: $$\boldsymbol{H(z) = 1 + 1.9z^{-1} + 0.8z^{-2} - 0.8z^{-3} - 0.7z^{-4}}$$

i. Find its magnitude response using the MATLAB function **`freqz.`** From the magnitude response what can be said about the pole locations?

ii. From the magnitude response, try to sketch the pole/zero plot.

iii. Using **`zplane`** function, plot the true pole/zero locations.

(b) Repeat (a) for the following transfer function (use a value of $\alpha = 0.8$)

Equation 2-20:

$$H(z) = \frac{1 - 2\cos\omega_0\ z^{-1} + z^{-2}}{1 - 2\alpha\cos\omega_0\ z^{-1} + z^{-2}}, \qquad \omega_0 = \pi/4$$

i. What visible clues in the magnitude response of the given transfer function provide information about the pole/zero locations?

ii. How are these clues affected by the poles and zeros away from the unit circle?

iii. How does α affect the pole locations? What values can α take such that the system is stable? **Hint**: Consider $|\alpha| > \sqrt{2}$ and $|\alpha| < \sqrt{2}$ and explain your observations.

iv. Based on your sketch of the magnitude response from the pole/zero plot, what can you say about the transfer function of this filter (What does this system do)?

➢ **Problem 5.**

In this discussion, the effect of pole locations on the output of a discrete-time system is considered. The effect that the poles have on system dampening behavior is also investigated by examining a simple system. An oscillator can be understood as a system that continues to generate an output between prescribed limits once it has been started, i.e. a marginally stable system. An example of two pole oscillator can be defined as

Equation 2-21:

$$h(n) = [r^n\cos(\omega_0 n)]u(n) \Leftrightarrow H(z) = \frac{1 - [rcos\omega_0]z^{-1}}{1 - [2rcos\omega_0]z^{-1} + r^2z^{-2}}, \qquad |z| > r$$

If the sampling rate is 1200Hz and the frequency of the system (resonant or center frequency) taken to be 35Hz, then:

Equation 2-22:

$$\Omega_0 = 2\pi \times 35 \left(\frac{rad}{sec}\right), f_s = 1200\ Hz \rightarrow \omega_0 \left(\frac{rad}{sample}\right) = \left(\frac{\Omega_0}{f_s}\right) = (\frac{2\pi \times 35}{1200})$$

The output of the oscillator is to be a sequence of real numbers. A two pole oscillator has a complex-conjugate pair of poles that ensure a real system. The output of this two pole oscillator contains a sinusoid of frequency 35 Hz. Considering **Equation 2-21** answer the following questions:

(a) Using **`zplane`** function, plot the true pole/zero locations for $r = 1$. Give an impulse to the input and plot the output. Generate and plot 256 samples of the output. Do you see the oscillations?

(b) Using **`zplane`** function, plot the true pole/zero locations for $r < 1$ (or $r = 1 - \varepsilon$). Give an impulse to the input and plot the output. Generate and plot 256 samples of the output. Do you see the output of the oscillator be dampened?

Lab 3. The Sampling Theorem

Objectives

- To understand the issues involved in sampling of a continuous-time signal
- To study the effects of different sampling rates

Sampling Issues

A signal can be classified as either continuous or discrete based on its independent variable (time). Continuous-time signals are defined along a continuum of time whereas discrete-time signals are defined at discrete times and represented as sequences of numbers. Continuous-time signals are often referred to as analog signals. Speech signals or the scan of the electrical activity of the brain (electroencephalogram or EEG) are examples of such signals.

In addition, the signal amplitude may also be either continuous or discrete. If both time and amplitude of the signal are discrete, it is called a digital signal. Examples of which are computer data and telegraph signals.

For processing, transmitting, and storing of continuous-time signals, a huge size of data should be manipulated which is not efficient and applicable in most of signal processing applications. Furthermore, numerous digital signal processors, software, tools, and etc. exist todays which can only be employed for manipulating digital signals. For these reasons it is essential to sample continuous-time signals. Sampling process may result in information loss. Sampling theorem is a theorem that expresses the process of sampling operation through which information loss will not happen.

Periodic Sampling

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 denotes the sampling period) are retained. A sampled signal can be generated by using a method referred to as sampling by modulation as shown in **Figure 3-1**, where $x_c(t)$ is the continues-time signal and $s(t)$ is an impulse train (the carrier) defined as

Equation 3-1:

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

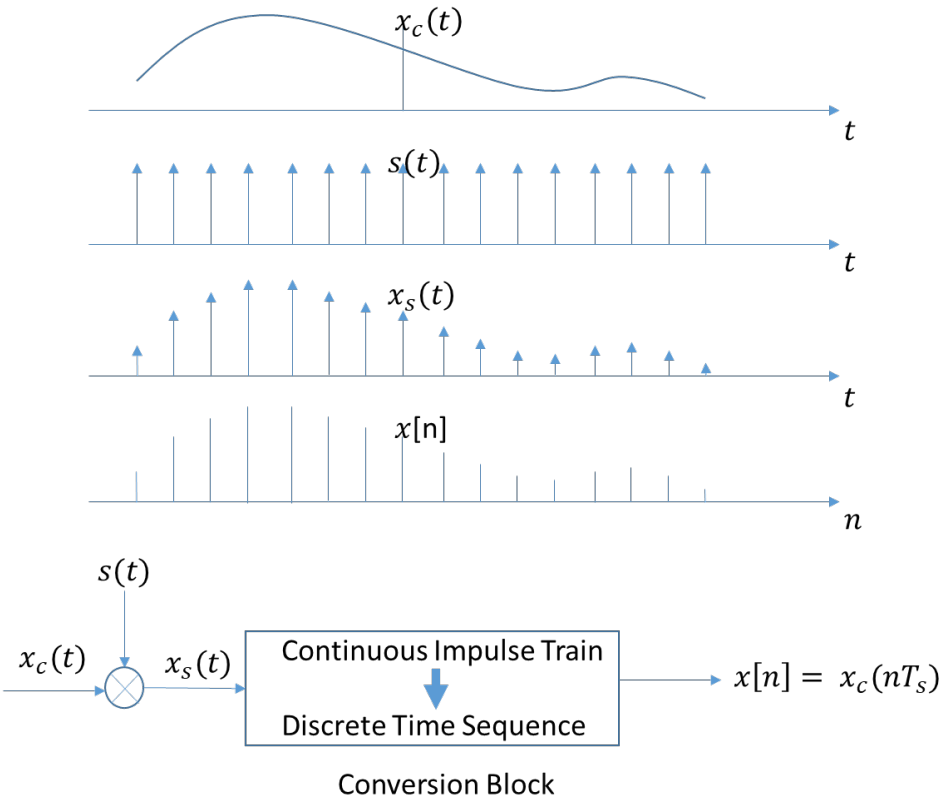


Figure 3-1: Sampling by modulation

The sampled signal $x_s(t)$ (where the subscript "s" refers to sampled) can now be represented as

Equation 3-2: $x_s(t) = x_c(t)s(t)$

Note that $x_s(t)$ is still a continuous-time signal. In order to understand the effect of sampling of a continuous-time signal, it will be more straightforward to analyse it in frequency domain. The frequency domain representation of **Equation 3-2** can be obtained as follow [2]:

Equation 3-3:

$$X_s(j\Omega) = \frac{1}{T_s}\sum_{k=-\infty}^{+\infty} X_c\big(j(\Omega - k\Omega_s)\big)$$

where $\Omega_s = \frac{2\pi}{T_s}$ is the sampling frequency. It means that sampling of a continuous-time signal with sampling period of T_s will cause the frequency spectrum to be periodic with Ω_s (sampling frequency). It can also be shown [1] that

Equation 3-4:

$$X\big(e^{j\omega}\big) = X_s(j\Omega), \qquad \Omega = \frac{\omega}{T_s}$$

Now with **Equation 3-3** and **Equation 3-4** we obtain

Equation 3-5:

$$X\big(e^{j\omega}\big) = \frac{1}{T_s}\sum_{k=-\infty}^{+\infty} X_c\left(j\left(\frac{\omega}{T_s} - \frac{2\pi k}{T_s}\right)\right)$$

Note that in the above equations we used t, and Ω for continues-time signals and also n, and ω for the discrete ones. If $x_c(t)$ is an example of a band-limited signal, then according to the above equations $X_c(j\Omega)$, $X_s(j\Omega)$, and $\mathrm{X}\big(e^{j\omega}\big)$ can be depicted as shown in **Figure 3-2**. According to **Equation 3-3**, and **Equation 3-5**, the magnitude of the frequency spectrum has been scaled by $\frac{1}{T_s}$.

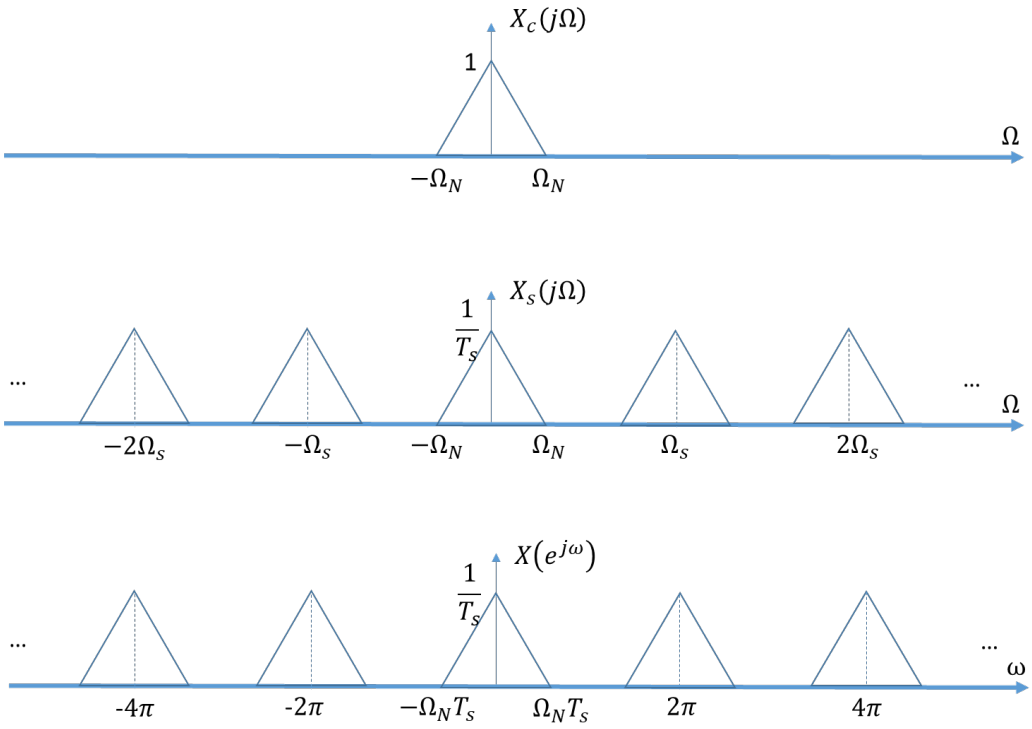


Figure 3-2: Frequency representation of periodic sampling

Nyquist Theorem

The example signal shown in **Figure 3-2** (the first) is a band-limited signal. It means that

Equation 3-6:

$$X_c(j\Omega) = 0, \qquad |\Omega| \geq \Omega_N$$

As mentioned above, sampling of a continuous-time signal with sampling period of T_s will cause the frequency spectrum to be periodic with Ω_s (sampling frequency). This can be seen in **Figure 3-2** (the middle). As the sampling period T_s is decreased (Ω_s increases), all replicas of $X_c(j\Omega)$ move farther apart. On the other hand, if T_s increases (Ω_s decreases), the replicas of $X_c(j\Omega)$ move closer together. As T_s is continually increased, a point will be reached where the replicas will begin to overlap as shown in **Figure 3-3**. This overlap of the frequency spectrum is known as aliasing or folding. According to **Figure 3-2** (the middle) the maximum sampling period (equivalent to minimum sampling frequency) at which there is no aliasing (overlapping) is attained when

Equation 3-7:

$$\Omega_s - \Omega_N > \Omega_N \quad \rightarrow \quad \Omega_s > 2\Omega_N$$

It should be noted that for above equation to be valid, the continuous-time signal should be band-limited. This theorem is very popular and is called Nyquist theorem.

> **Nyquist Theorem**: If $x_c(t)$ is a band-limited continuous-time signal and sampling frequency (Ω_s) is greater than two times of maximum frequency component of the signal (Ω_N) (**Equation 3-7**), then $x_c(t)$ can be reconstructed completely from its discrete time sequence ($x[n]$).

In order to avoid information loss due to aliasing the Nyquist theorem should be satisfied. In the Nyquist theorem, Ω_N is called *Nyquist frequency* and $\Omega_s = 2\Omega_N$ is called *Nyquist rate*.

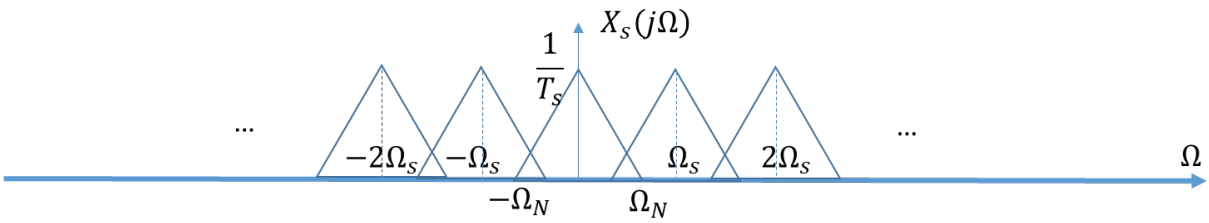


Figure 3-3: As Ω_s is continually decreased the replicas will begin to overlap. This overlap of the frequency spectrum is known as aliasing

Reconstruction

If Nyquist theorem is satisfied while sampling a band-limited continuous-time signal, the aliasing will not be happened and therefore, that signal can be reconstructed completely using its samples. As mentioned before, sampling a continuous-time signal will cause its frequency spectrum to be periodic. For reconstruction purpose, the frequency components related to the band-limited continuous-time signal should be retained while frequencies related to its replicas should be removed. Therefore, as shown in **Figure 3-4**, the reconstruction process can be done by applying an ideal low pass filter with gain T_s and cut-off frequency Ω_c.

Due to sampling operation as the magnitude of frequency spectrum is scaled by $\frac{1}{T_s}$, the gain value of the applied ideal low pass filter is considered to be T_s to rescale the spectrum to its original value.

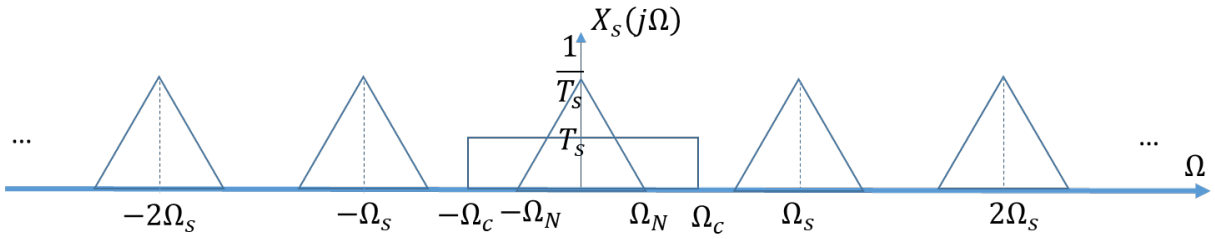


Figure 3-4: Reconstruction of band-limited signal by applying an ideal low pass filter

The cut-off frequency Ω_c should be determined in the range

Equation 3-8:

$$\Omega_N < \Omega_c < \Omega_s - \Omega_N$$

As mentioned above, $\Omega_s > 2\Omega_N \Rightarrow \frac{\Omega_s}{2} > \Omega_N$ or $-\frac{\Omega_s}{2} < -\Omega_N$. With minor modification we get $\Omega_s - \frac{\Omega_s}{2} < \Omega_s - \Omega_N$, and thus $\frac{\Omega_s}{2} < \Omega_s - \Omega_N$. Now since $\frac{\Omega_s}{2} > \Omega_N \Rightarrow \Omega_N < \frac{\Omega_s}{2} < \Omega_s - \Omega_N$. Therefore, if we consider $\Omega_c = \frac{\Omega_s}{2}$, **Equation 3-8** will be satisfied and we get

Equation 3-9:

$$\Omega_c = \frac{\Omega_s}{2} = \frac{2\pi}{T_s} \times \frac{1}{2} = \frac{\pi}{T_s}$$

The reconstruction process can be seen in **Figure 3-5**, where

Equation 3-10:

$$x_s(t) = \sum_{n=-\infty}^{+\infty} x[n]\delta(t - nT_s)$$

and $H_r(j\Omega)$ is the ideal low pass filter where its time representation is

Equation 3-11:

$$h_r(t) = sinc\left(\frac{\pi t}{T_s}\right) = \frac{sin\left(\frac{\pi t}{T_s}\right)}{\frac{\pi t}{T_s}} = \frac{sin(\Omega_c t)}{\Omega_c t}$$

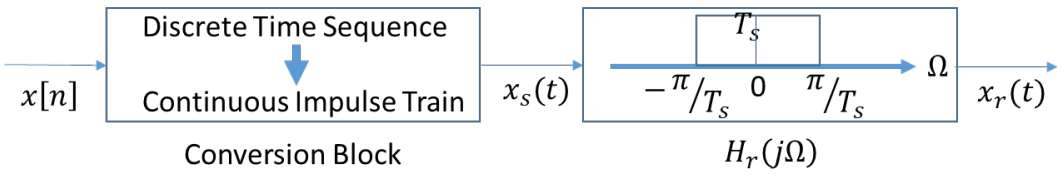


Figure 3-5: The reconstruction process

Then using convolution operation, $x_c(t)$ can be expressed (reconstructed) exactly in terms of its samples $x_c(nT_s)$

Equation 3-12:

$$x_c(t) \cong x_r(t) = x_s(t) * h_r(t) = \sum_{n=-\infty}^{+\infty} x[n] sinc\big(\Omega_c(t - nT_s)\big) = \sum_{n=-\infty}^{+\infty} x_c(nT_s) sinc\big(\Omega_c(t - nT_s)\big)$$

In practice, the conversion of a sampled signal to a continuous-time form, referred to as the *digital to analog (D/A)* operation, involves two parts, namely a hold operation followed by low pass filtering.

The most common form of hold operation is the zero-order hold or sample and hold. The impulse response for the sample and hold device is given by

Equation 3-13:

$$h_{hold}(t) = \begin{cases} 1, & 0 \le t < T_s \\ 0, & otherwise \end{cases}$$

The operation of the sample and hold is illustrated in **Figure 3-6**.

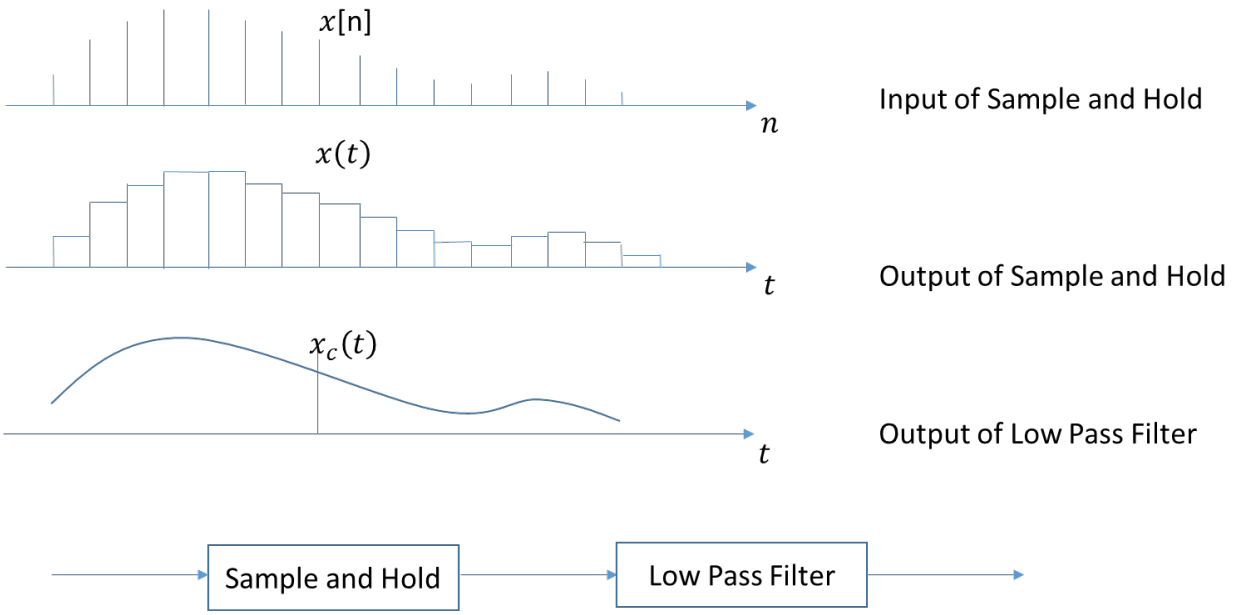


Figure 3-6: Reconstruction of a continuous time signal using its samples. The input to the "Sample and Hold" is $x[n]$ and the output is $x(t)$. The output of the "Low Pass filter" is $x_c(t)$.

Lab Problems

- **Problem 1.** Consider the sinusoidal signal

  $x(t) = \sin \Omega_0 t$, where $\Omega_0 = 10\pi\ rad/sec$.

  (a) Using a sampling rate 20 times the *Nyquist rate*, simulate the continuous-time signal $x(t)$. Using the MATLAB **`plot`** function, generate the graph over the time interval $0 \leq t \leq 1$ seconds.

  Hint: For this purpose, the maximum frequency of the signal should be found ($\frac{\Omega_0}{2\pi}$) and then the Nyquist rate will be computed easily. Here the sampling rate is considered to be 20 times of Nyquist rate so compute the wanted sampling rate (F_s) and sampling period (T_s). See the following code segment for more help

```
t = 0 : Ts : 1; %time interval
x_s = sin(2 * pi * f_max * t);
```

  (b) Following **Equation 3-11** and **Equation 3-12**, one can reconstruct the signal using its samples according to the following code segment:

```
h_r = sinc(pi * t / Ts);
x_c = conv(h_r, x_s, 'full');
l = length(x_c);
t_c = linspace(0, 1, l / 2);
figure, plot(t_c, x_c(1 : l / 2));
```

Run the code, see the results, and explain the reason of writing each of the statement of the above code completely.

(c) Now sample $x(t)$ at 1, 2, 5, and 10 times of the *Nyquist rate* and reconstruct the signal using the corresponding samples. Explain your observations of the reconstructed signals.

(d) Sample $x(t)$ at 5 times of the *Nyquist rate* and then apply a low-pass filter, with $\Omega_c = 0.2\pi$, on this signal and see the result. Reconstruct the signal using the sample and hold operation by holding the sample for 6 next samples. Now apply a low-pass filter on this signal and plot the reconstructed signal.

Hint 1: A low-pass filter removes high-frequency content from a signal and preserves low-frequency content. We dedicate Lab 5 for filtering operations. For this lab, you can use the following code segment for low-pass filtering:

```
b = fir1(48, 0.2); %Design a 48th-order FIR lowpass filter with w <= .2
* pi
x_c = filter(b, 1, x); %Appling filter on x using b coefficients
```

Hint 2: For simulating the sample and hold operation in MATLAB, if the length of the signal $x(t)$ is L, and we want to hold each sample for n next samples (here $n = 6$), we can define a new signal with length $L \times (n + 1)$ and then fill its elements using for loop.

➢ **Problem 2.** The signal of interest is

Equation 3-14:

$$S(t) = \frac{1}{2}\sin(14\pi t) + \frac{1}{3}\sin(18\pi t) + \frac{1}{5}\sin(24\pi t) + \frac{1}{7}\sin(30\pi t)$$

(a) Sample $S(t)$ on the interval $0 \leq t \leq 20$ at 20 times the *Nyquist rate*. Plot its two-sided spectrum.

Hint: In order to plot the two-sided spectrum of signal S with sampling frequency F_s you can use **`fftshift`** MATLAB built-in function as follow:

```
X = fft(S);
Y = fftshift(X); %Shift zero-frequency component to center of spectrum
L = length(Y);
P = abs(Y / L);

%Define the frequency domain f and plot the two-sided amplitude spectrum P.
f = Fs * (-L / 2 : L / 2 - 1) / L;
plot(f, P)
```

(b) Plot the two-sided spectrum of $S(t)$ with the following sampling frequencies:

i. 3/2 time the *Nyquist rate*

ii. 5/6 times the *Nyquist rate*

iii. 13/30 times the *Nyquist rate*

(c) Explain the differences in signals spectra in all of the cases of part b. In case of Nyquist criterion (no aliasing) and violating it (having aliasing). In case of aliasing as shown in **Figure 3-3** completely explain why the resulting frequency components exist in the observed locations? (What are these frequency locations?)

(d) Given that the sampling rate for $y(t)$ is fixed to be 20Hz, how would it be possible to ensure that the spectrum of the sampled signal $y[n]$ for the frequencies $0 \leq f \leq 10$ is a true representation. Draw a block diagram of the procedure to meet such an objective.

Lab 4. Multirate Digital Signal Processing

Objectives

- To learn how to use MATLAB to implement *sampling rate conversion* and *multirate signal processing* systems.

Key Issues

A multirate Digital Signal Processing (DSP) system is characterized by the use of multiple sampling rates in its realization. That is, signal samples at various points in the systems may not correspond to the same physical sampling frequency. Some applications of multi-rate processing include

- Sampling rate conversion between different digital audio standards
- Digital anti-aliasing filtering (used in commercial CD players)
- Sub-band coding of speech and video signals

In some applications, the need for multi-rate processing comes naturally from the problem definition (e.g. sampling rate conversion). In other applications, multi-rate processing is used to achieve improved performance of a DSP function (e.g. lower binary rate in speech compression).

In our previous experiments, we have assumed that all signals in a given system have the same sampling rate. We have interpreted the time index n as an indicator of the physical time nT_s, where T_s is the sampling interval. In many practical applications of DSP, one is faced with the problem of changing the sampling rate of a signal, either increasing it or decreasing it by some ratio. For example, in telecommunication systems that transmit and receive different types of signals (e.g. speech, video, etc.), there is a requirement to process the various signals at different rates commensurate with the corresponding bandwidths of the signals. The process of converting a signal from a given rate to a different rate is called *sampling rate conversion*. In turn, systems that employ multiple sampling rates in the processing of digital signals are called *multirate digital signal processing systems*.

The two basic operations in a multirate system are decreasing and increasing the sampling rate of a signal. The former is called decimation or down-sampling. The latter is called interpolation or up-sampling.

Downsampling and Decimation

Downsampling can be considered as the discrete-time counterpart of sampling. Whereas in sampling we start with a continuous-time signal $x_c(t)$ and convert it to a sequence of samples $x[n]$, in downsampling we start with a discrete-time signal $x[n]$ and convert it to another discrete-time signal $x_d[n]$, which consists of subsamples of $x[n]$. Thus, the formal definition of M-fold downsampling is

Equation 4-1:

$$x_d[n] = x[Mn]$$

where M is a positive integer. The block diagram representation of the downsampling operation is shown in **Figure 4-1**.

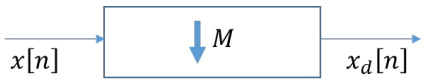


Figure 4-1: Downsampling by integer factor M.

Where the output sequence $x_d[n]$ is sampled at sampling rate equal to $(\frac{1}{M})^{th}$ of the sampling rate of input sequence $x[n]$. Downsampling operation can be implemented by keeping all the input samples that have indices equal to an integer multiple of M.

For example **Figure 4-2** shows downsampling by integer factor $M = 2$.

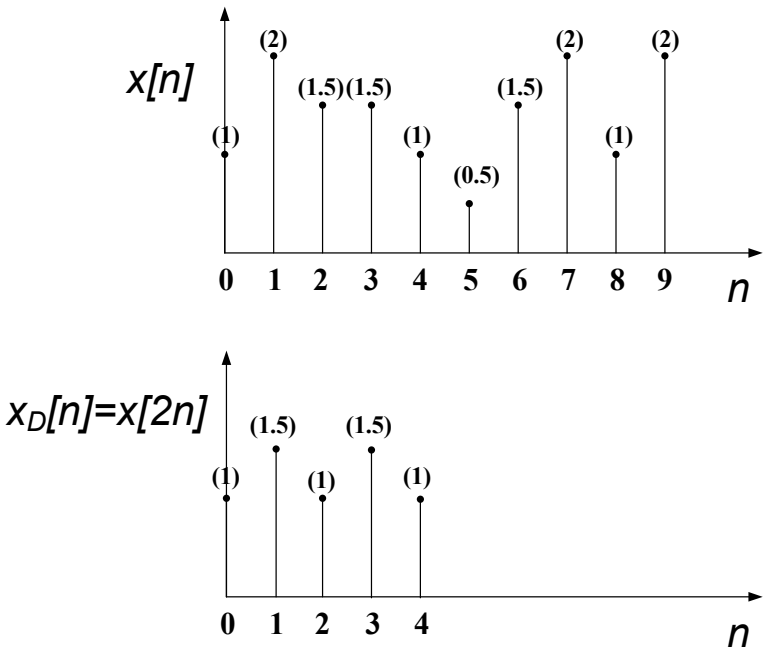


Figure 4-2: Down-sampling by integer factor $M = 2$

Note that the samples corresponding to $n = 1, 3, 5, 7, 9$ are lost by downsampling.

It is more convenient to analyse downsampling operation in frequency domain. The frequency representation of downsampling operation is given [1] by

Equation 4-2:

$$X_d(e^{j\omega}) = \frac{1}{M}\sum_{i=0}^{M-1} X\left(e^{j\left(\frac{\omega-2\pi i}{M}\right)}\right)$$

According to **Equation 4-2**, downsampling by integer factor M has three effects on the frequency spectrum:

1. It expands the frequency spectrum by M.
2. It makes the frequency spectrum be periodic with 2π and sums up all the resulted replicas.
3. It scales the magnitude by $\frac{1}{M}$.

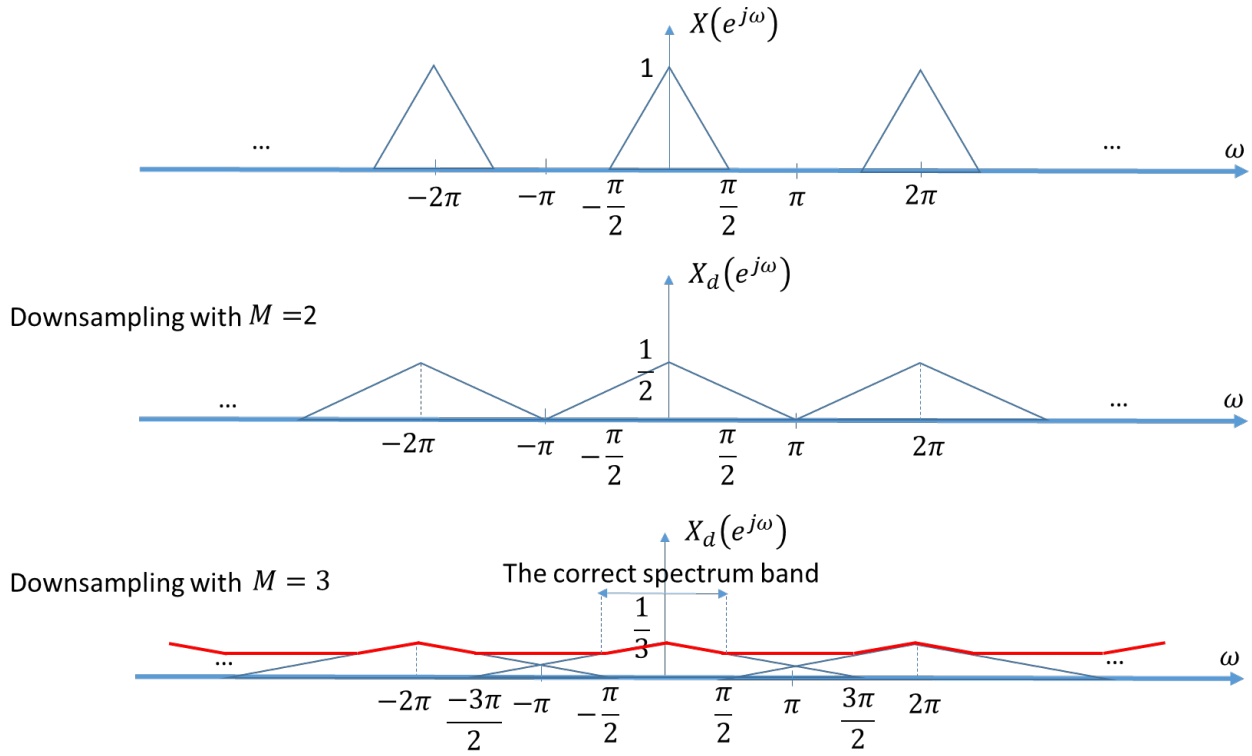


Figure 4-3: The effect of downsampling operation by factors $M = 2$ (the second plot), and $M = 3$ (the third plot)

In Figure 4-3 these three effects can be seen on an example signal by considering two downsampling factors $M = 2$, and $M = 3$. In Figure 4-3 (the second plot) aliasing does not happen (there isn't any overlap between replicas), but in this example when $M = 3$, downsampling leads to aliasing. As can be seen in Figure 4-3 (the third plot) red line shows the resulted frequency spectrum after downsampling by $M = 3$, and the correct frequency spectrum band which is not affected by aliasing is in the range $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. In Figure 4-4 a lowpass filter with cut-off frequency $\Omega_c = \frac{\pi}{3}$, and gain 1 is applied on the signal before downsampling operation. Figure 4-4 second plot shows the result of applying this filter. Due to the downsampling by factor $M = 3$ the frequency spectrum will be expanded by 3, which will now cover the range $[-\pi, \pi]$. Although in this case we have still information loss but compared to Figure 4-3 (the third plot), aliasing does not happen and the correct frequency spectrum band covers whole of the range $[-\pi, \pi]$. The downsampling filter is called anti-aliasing filter or decimation filter.

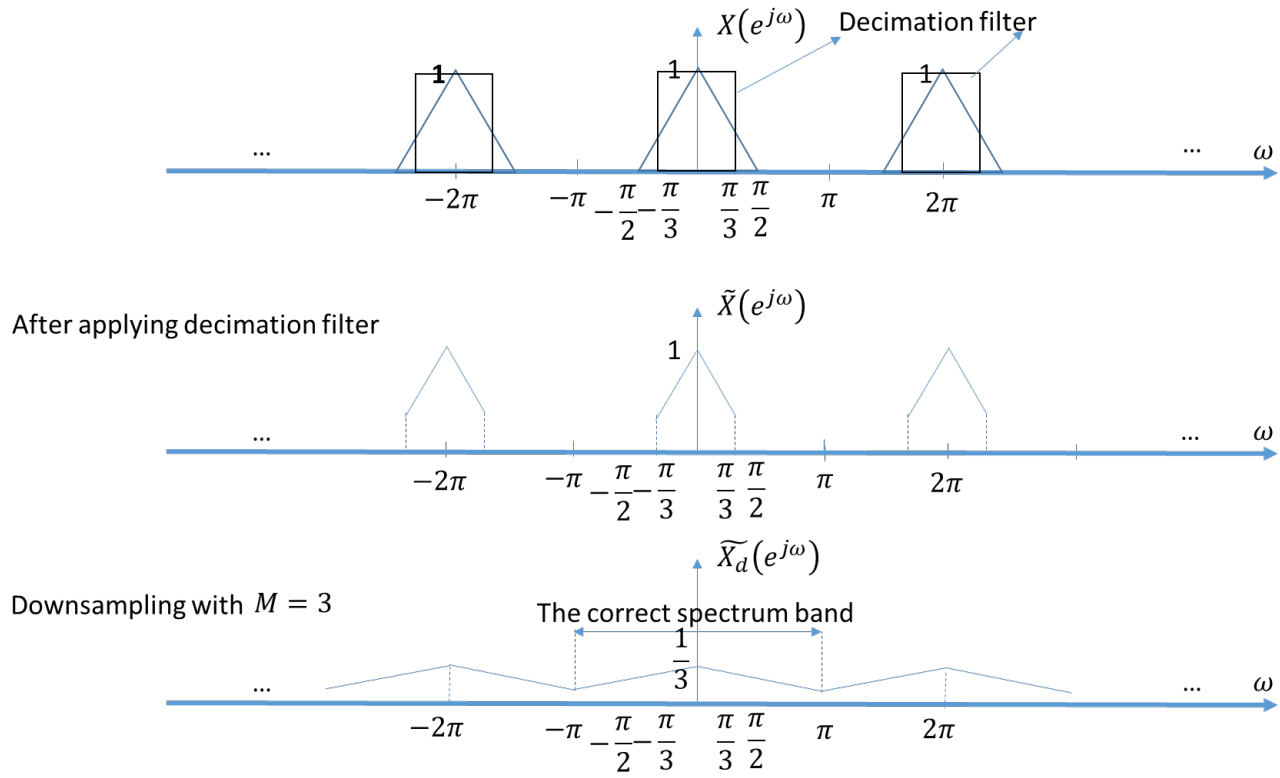


Figure 4-4: The effect of applying lowpass filtering before downsampling operation

According to the above mentioned, it is desirable to precede the downsampler with an anti-aliasing filter. Unlike sampling, here the input signal is already in discrete time, so we use a digital anti-aliasing filter. The anti-aliasing filter (also called the decimation filter), should approximate an ideal lowpass filter with cut-off frequency $\frac{\pi}{M}$, as shown in **Figure 4-5**. The combination of the lowpass filter followed by a downsampler is called *decimation*.

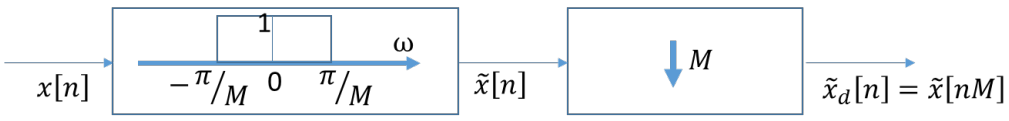


Figure 4-5: A general decimation system: lowpass filter has gain equal to 1 and cut-off frequency π/M combined with down-sampler by integer factor M.

Upsampling and Interpolation

Upsampling is another operation on a discrete-time signal that yields a discrete-time signal. Upsampling by factor L, adds $L-1$ new samples between each two samples of the input discrete-time signal. The magnitude of these new added samples will be estimated by interpolation technique. Similar to the continuous-time signals, in order to perform interpolation in discrete-time signals, an ideal digital lowpass filter can be applied on the signal. This filtering process will not generate additional samples, therefore, before interpolation operation an upsampler (expander) will be employed which adds $L-1$ new samples with zero magnitude between each two samples of the original signal. If the input signal is $x[n]$, the output signal of upsampler (expander) is $x_e[n]$ which is defined as follow:

Equation 4-3:

$$x_e[n] = \begin{cases} x\left[\frac{n}{L}\right] & n = 0, \pm L, \pm 2L, \dots \\ 0 & Otherwise \end{cases}$$

The block diagram representation of an expander with integer factor L is shown in **Figure 4-6**.

Figure 4-6: Expander by integer factor L

For an example **Figure 4-7** shows expanding by integer factor $L = 2$.

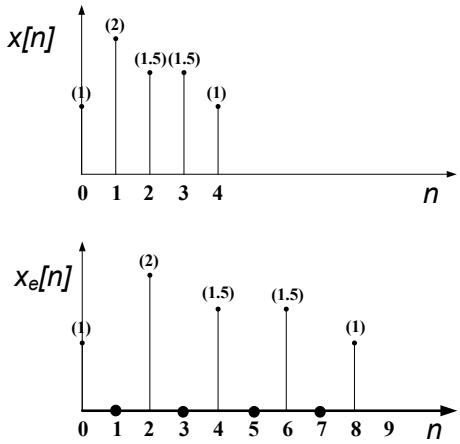


Figure 4-7: Expanding by integer factor $L = 2$

The frequency representation of an expander is [1] as follow:

Equation 4-4:

$$X_e(e^{j\omega}) = X(e^{j\omega L})$$

From **Equation 4-4** it is obvious that the upsampler (expander) will compress the frequency spectrum of the signal by L. In the other words aliasing (and information loss) will not be happened. As can be seen in **Figure 4-8** an example signal is expanded by $L = 2$, and because of this its frequency spectrum has been compressed by 2.

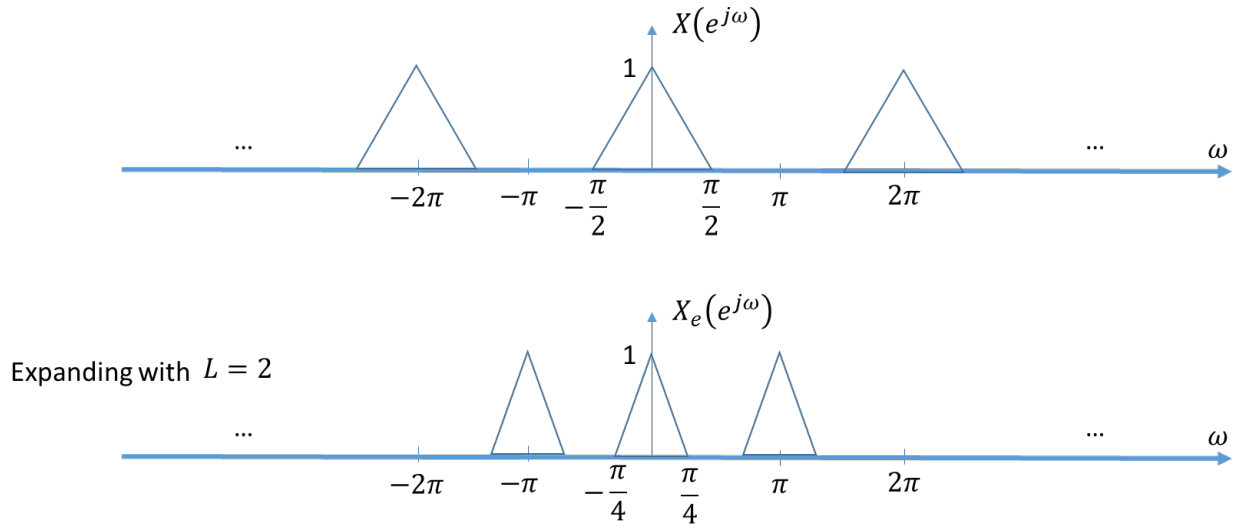


Figure 4-8: The effect of expanding a signal with $L = 2$

Upsampler (expander) will place additional samples with zero magnitude between each two samples of the original signal. Now it is the time to estimate the value of those samples by applying an ideal interpolation filter which is an ideal digital lowpass filter. It is worth noting that upsampler (expander) also yields undesired replicas of the signal's spectrum in the range $\omega \in [-\pi, \pi]$. These replicas can be eliminated by applying the mentioned interpolation filter. The *interpolation filter* is also called *anti-imaging filter*. This filter has cut-off frequency $\omega_c = {}^{\pi}/_{L}$ and gain equal to L [1]. The combination of upsampler (expander) followed by an interpolation filter is called interpolation system as in **Figure 4-9**.

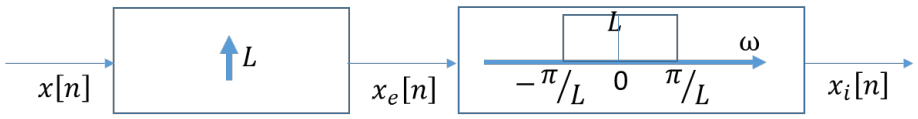


Figure 4-9: A general interpolation system: upsampler (expander) with integer factor L combined with interpolation filter.

It should be noted that the upsampler and downsampler operations are linear but time-varying (not time-invariant) discrete-time systems. The upsampler and the downsampler building blocks are often involved in *multirate signal processing*.

Sampling Rate Conversion

A common use of multirate signal processing is for sampling rate conversion. Sampling-rate conversion can be accomplished by first upsampling by factor L followed by lowpass filtering and then down-sampling by factor M as it is observed in **Figure 4-10**. In this figure the sampling time interval for all input and output signals have been shown under the name of each signal. Interpolation and decimation filters can also be combined with each other (**Figure 4-11**). In this case the lowpass filter performs both interpolation of the upsampled signal and anti-aliasing before downsampling. If the sampling rate is to be increased, then we have $L > M$. The lowpass filter should then have a cutoff frequency $\frac{\pi}{L}$. If the sampling rate is to be decreased, then we have $L < M$. The lowpass filter should then have a cutoff frequency $\frac{\pi}{M}$. Thus, the sampling-rate conversion filter should always have a cut-off frequency $\Omega_c = min\left(\frac{\pi}{L}, \frac{\pi}{M}\right)$.

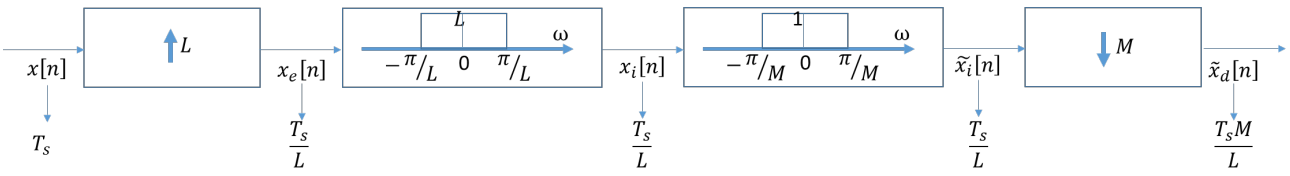


Figure 4-10: System for changing the sampling rate by a non-integer factor with interpolation and decimation filters.

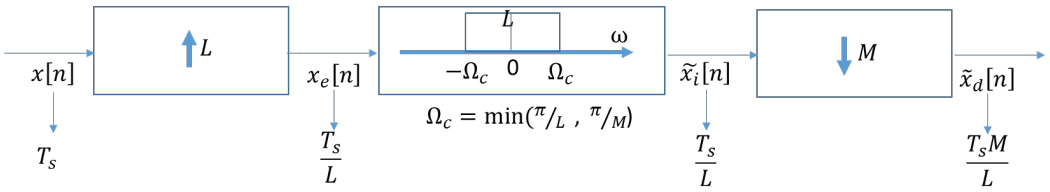


Figure 4-11: System for changing the sampling rate by a non-integer factor with combined interpolation and decimation filters.

Suppose that we are given a digital signal $x[n]$ sampled at interval T_1 and we wish to obtain from it a signal $y[n]$ sampled at interval T_2. The techniques of decimation and interpolation enable this operation, provided the ratio $\frac{T_1}{T_2}$ is a rational number say $\frac{L}{M}$. We distinguish now between two possibilities:

1) $T_1 > T_2$, meaning that the sampling rate should be increased. This is always possible without aliasing.
2) $T_1 < T_2$, meaning that the sampling rate should be decreased. This is possible without aliasing only if $x[n]$ is band limited to a frequency range not higher than $\pm\pi(\frac{T_1}{T_2})$. If $x[n]$ does not fulfill this condition, a part of its frequency contents must be eliminated before decimation to avoid aliasing.

Lab Problems

➢ **Problem 1.**

(a) Write a MATLAB script to generate the signals $x_1[n]$, $x_2[n]$, and $x_3[n]$ that have the DTFT shown in **Figure 4-12** (a), (b), and (c), respectively. Make sure that each signal has length of 1025 samples. Plot the resulting DFT over the range $-\pi \leq \omega \leq \pi$.

Hint: For this purpose you can use **`fir2`** MATLAB function. **`b=fir2(n,f,m)`** produces an n^{th}order FIR filter with frequencies f and magnitudes m, returning the $n+1$ filter coefficients b [3]. For example the following code segment will produce a function which has frequency spectrum similar to **Figure 4-12** (a).

```
N = 1024;
T=1/N;
f = 0 : T : 1; %Frequency range
```

```
L = length(f); %Length of f
m = zeros(1, L); %Frequency magnitude
m(1 : 1/4 * L)= -4 * f(1: 1/4 * L) + 1; %Frequency magnitude
x = fir2(n, f, m);   %nth-order FIR filter with frequency-magnitude
characteristics specified in the vectors f and m.
X = fft(x); %Fourier Transform of x
L = length(X); %Length of X
fshift = linspace(-pi, pi, L); %Generate linearly spaced vector
X = fftshift(X); %Shift zero-frequency component to center of spectrum
plot(fshift, abs(X));
title('FFT of input signal X');
ax = gca; %Current axes or chart
ax.XTick = -3:.2:3;
ax.YTick = 0:.5:1;
```

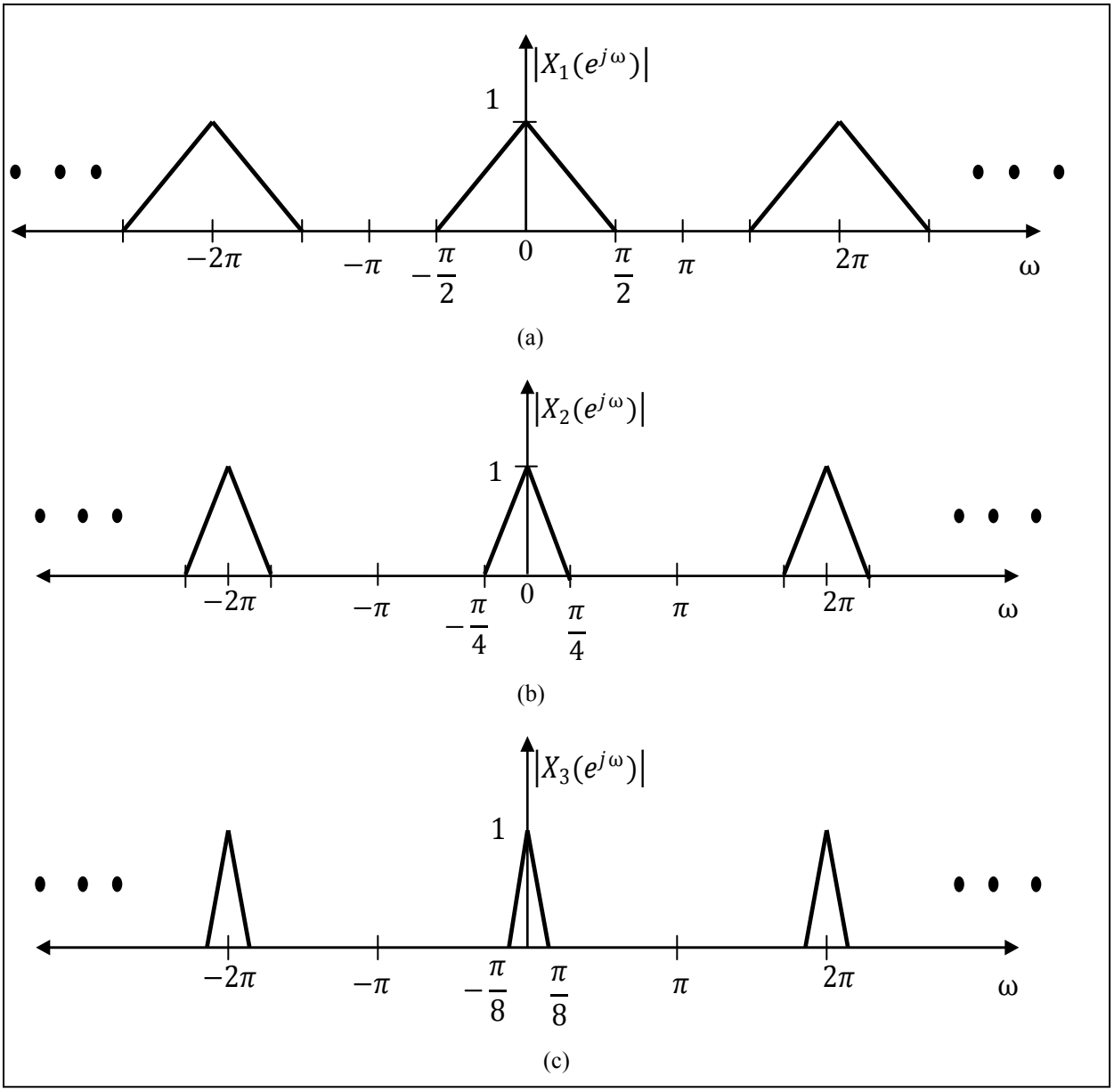


Figure 4-12: Discrete Fourier transform of signals $x_1[n]$, $x_2[n]$ and $x_3[n]$.

(b) Use $x_1[n]$ of part (a) as the input to the systems (a), (b), (c), (d), and (e) shown in Figure 4-13 and write a MATLAB script to generate the output $y[n]$ for each system. Plot the discrete time Fourier transform (over the range $-\pi \leq \omega \leq \pi$) of the input signal $x_1[n]$ and the output signal $y[n]$ on same figure using "subplot" command for each system. Adjust the title, xlabel and ylabel for each figure accordingly. Repeat this scenario for $x_2[n]$, and $x_3[n]$.

Hint: In order to design a low-pass filter with order 128, cutoff frequency $\frac{\pi}{3}$, and gain g, you can use **`fir1`** MATLAB function as follow:

```
b = g * fir1(128, 1/3); %Window-based low pass FIR filter cutoff pi/3
y = filter(b, 1, x); %y is the output of applying the designed filetr
on signal x
```

In order to do downsampling and upsampling you can use **`downsample`** and **`upsample`** MATLAB functions [3].

(c) Explain the reason of each resulted plots in part (b) completely.

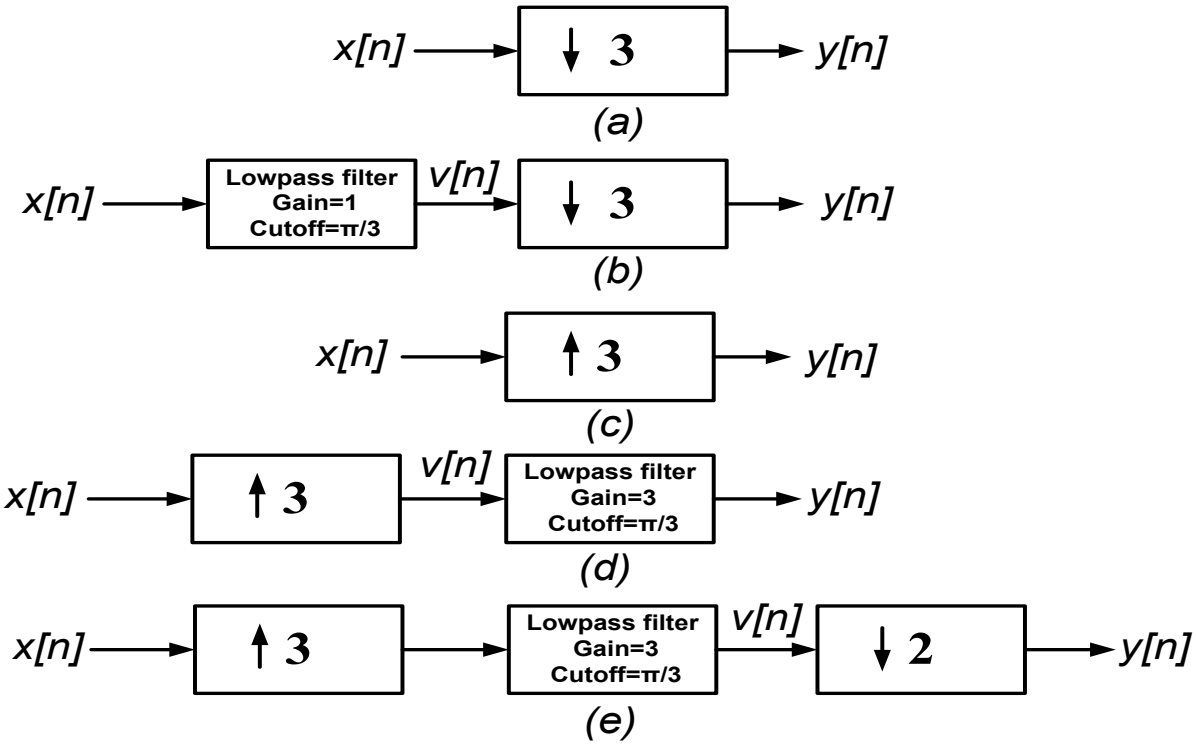


Figure 4-13: Down-sampler, decimation system, up-sampler, interpolation system and a rate-conversion system.

➢ **Problem 2.**

Consider the analysis-synthesis system shown in **Figure 4-14**.

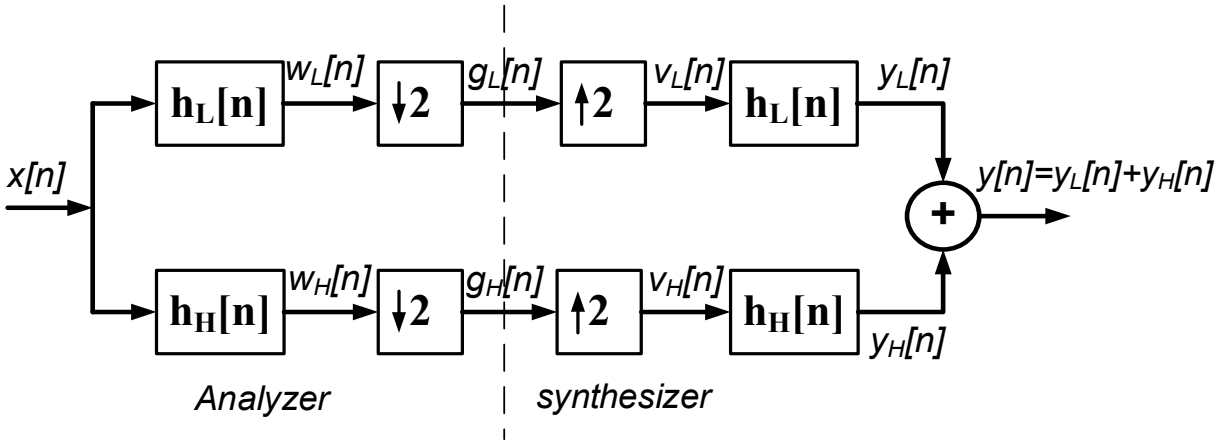


Figure 4-14: An analysis-synthesis system

The lowpass filter $h_L[n]$ and the highpass filter $h_H[n]$ are identical in the analyzer and synthesizer parts. The Fourier transforms of $h_L[n]$ and $h_H[n]$ are related by

Equation 4-5:

$$H_H(e^{j\omega}) = H_L(e^{j(\omega+\pi)})$$

or its Z-transform counterpart:

Equation 4-6:

$$H_H(Z) = H_L(e^{j\pi}Z) = H_L(-Z)$$

Using Z-transform to find a general expression for $Y(Z)$ in term of $X(Z), H_L(Z)$ and $H_H(Z)$ we will have

Equation 4-7:

$$W_L(Z) = X(Z)H_L(Z)$$

Using **Equation 4-2** when downsampling factor is $M = 2$, we can derive the following equation

Equation 4-8:

$$G_L(Z) = \frac{1}{2}\left[W_L\left(Z^{\frac{1}{2}}\right) + W_L\left(-Z^{\frac{1}{2}}\right)\right] = \frac{1}{2}\left[X\left(Z^{\frac{1}{2}}\right)H_L\left(Z^{\frac{1}{2}}\right) + X\left(-Z^{\frac{1}{2}}\right)H_L\left(-Z^{\frac{1}{2}}\right)\right]$$

Now using **Equation 4-4** when upsampling factor is $L = 2$, we can derive the following equation.

Equation 4-9:

$$V_L(Z) = G_L(Z^2) = \frac{1}{2}[X(Z)H_L(Z) + X(-Z)H_L(-Z)]$$

and now we will have

Equation 4-10:

$$Y_L(Z) = V_L(Z)H_L(Z) = \frac{1}{2}[X(Z)H_L(Z) + X(-Z)H_L(-Z)]H_L(Z)$$

If we follow the same scenario for the second part of the system similarly we will have

Equation 4-11:

$$Y_H(Z) = \frac{1}{2}[X(Z)H_H(Z) + X(-Z)H_H(-Z)]H_H(Z)$$

and now we will have:

Equation 4-12:

$$\begin{aligned} Y(Z) = Y_L(Z) + Y_H(Z) \\ &= \frac{1}{2}[X(Z)H_L(Z) + X(-Z)H_L(-Z)]H_L(Z) \\ &+ \frac{1}{2}[X(Z)H_H(Z) + X(-Z)H_H(-Z)]H_H(Z) \end{aligned}$$

According to **Equation 4-6** we can write

Equation 4-13:

$$Y(Z) = Y_L(Z) + Y_H(Z) = \frac{1}{2}X(Z)\left[\left(H_L(Z)\right)^2 + \left(H_H(Z)\right)^2\right] + X(-Z)[H_L(Z)H_H(Z)]$$

(a) Design a lowpass FIR filter $h_L[n]$, with order 128, cutoff frequency $\frac{\pi}{2}$, and gain 1. You can use **`fir1`** MATLAB function.

(b) Design a highpass FIR filter $h_H[n]$, with order 128, cutoff frequency $\frac{\pi}{2}$, and gain 1. You can use **`fir1`** MATLAB function.

For the following parts, plot on the same figure using "subplot" command for each system. Adjust the title, xlabel and ylabel for each figure accordingly.

(c) Plot the magnitude of both $H_L(e^{j\omega})$ and $H_H(e^{j\omega})$ over the range $-\pi \leq \omega \leq \pi$.

(d) Plot the magnitude of $\left(H_L(e^{j\omega})\right)^2 + \left(H_H(e^{j\omega})\right)^2$ over the range $-\pi \leq \omega \leq \pi$.

(e) Plot the magnitude of $H_L(e^{j\omega})H_H(e^{j\omega})$ over the range $-\pi \leq \omega \leq \pi$.

(f) According to the results of parts (d) and (e) and considering **Equation 4-13** what is your idea about the value of $Y(Z)$.

(g) Let $x[n] = x_c(nTs)$, where the continuous time signal $x_c(t)$ is given by

$$x_c(t) = A[\cos(\Omega_1 t) + \cos(\Omega_2 t) + \cos(\Omega_3 t) + \cos(\Omega_4 t)]$$

where, $\Omega_1 = 1.2\pi \times 10^3\ rad/sec$, $\Omega_2 = 2.4\pi \times 10^3\ rad/sec$, $\Omega_3 = 3.6\pi \times 10^3\ rad/sec$,

$\Omega_4 = 4.8\pi \times 10^3\ rad/sec$, and the sampling frequency is $f_s = 1/T_s = 6 \times 10^3\ Hz$.

Plot $x[n]$ over the range $0 \leq n \leq 200$ and plot $\left|X(e^{j\omega})\right|$ over the range $-\pi \leq \omega \leq \pi$.

(h) Plot $y[n]$ over the range $0 \leq n \leq 200$ and plot $\left|Y(e^{j\omega})\right|$ over the range $-\pi \leq \omega \leq \pi$.

(i) According to the results of part (h), explain the difference between $x[n]$ and $y[n]$ and determine whether the aliasing phenomenon occurred or not.

Lab 5. Signal Restoration and Filter Design

Objectives

- In this lab experiment, you will learn how to use LTI filters as signal restoration tools and how to use MATLAB for designing filters.

Signal Restoration

"Signal Restoration" usually refers to the enhancement or improvement of signals that have been degraded in some way, for example, through noise or echo. Signal restoration recovers a signal from a degraded version. It is a fundamental problem in signal processing, and it also provides a testbed for more general inverse (or recovery) problems, that is, problems for which information that is lost (due, for example, to transmission or compression), is attempted to be recovered. Key issues that need to be addressed are the quality of the restored signal, the computational complexity of the solution (algorithm), and the estimation of necessary parameters such as those of the impulse response (or the point-spread function PSF). To restore a signal, we need to design a filter suitable for the problem to solve. To measure restored signal quality, we need to design objective quality measure.

Objective Quality Measurement

A corrupted signal is degraded version of the original. For each corrupted (noisy and echo) signal we would like to quantify how much corruption has taken place. A common measure of degradation is Mean Squared Error (MSE). If we have a signal x[n] and the same signal corrupted by some way called $x\_corrupt\ [n]$, each of length N, then the following is the MSE

Equation 5-1:

$$MSE = \frac{1}{N}\sum_{n=1}^{N}\left(x[n] - x_{corrupt}[n]\right)^2$$

Where the lower the MSE value the better the signal quality. We can also measure signal quality using the Signal to Noise Ratio (SNR) in decibels (dB), where 0.5 to 1 dB is said to be a perceptible difference

between two signals. The SNR indicates the strength of the signal with respect to the distortion (such as the noise or echo) power and is defined as

Equation 5-2:

$$SNR = 10log\frac{\sum_{n=1}^{N}|x[n]|^2}{MSE}.$$

The higher the SNR value the better the signal quality.

MATLAB's Filter Designer tool

In order to restore a signal corrupted in some way, say noise or echo, we need to develop an appropriate filtering operation. The Filter Designer tool in MATLAB is a powerful user interface for designing and analyzing filters quickly. It enables you to design digital FIR or IIR filters by setting filter specifications, by importing filters from your MATLAB workspace, or by adding, moving or deleting poles and zeros. Filter Designer also provides tools for analyzing filters, such as magnitude and phase response and pole-zero plots. Filter Designer seamlessly integrates additional functionality from other Mathworks products. To use the Filter Designer in MATLAB, type in command window

>>filterDesigner

Filter Designer will be opened. You can select FIR or IIR filter, order of filter, and cutoff frequency of a filter (either HPF, LPF or BPF). That code will automatically generate .m file for you.

An introduction to the Filter Designer tool is reproduced in FilterDesignerMATLAB.pdf in "Data_for_DSP_Labs" → "Lab 5" folder, and is available online here https://www.mathworks.com/help/signal/examples/introduction-to-filter-designer.html, which is

Lab Problems

- **Problem 1.**

  (a) In the folder "Data_for_DSP_Labs"→"Lab5", you find a file with the name "HappyBirthday.wav", which is a noise-free audio signal. Write a MATLAB code to read this signal. Add Gaussian noise to it with mean equal to zero and variance equal σ^2, that is, if your original noise-free signal is $x[n]$ then the noisy signal $x_{noisy}[n] = x[n] + noise$.

Create 3 noisy signals with different variances 0.1, 0.3, and 0.5. Listen to all generated noisy signals and finally save all of the noisy signals with meaningful names.

Hint 1: In order to generate a Gaussian noise with length "L", mean "m" and variance "$noise\_var$", you can use the following code segment:

```
noise = sqrt(noise_var) * randn(L, 1) + m; %Generate a Gaussian noise
with variance "noise_var" and mean "m"
```

Hint 2: In order to process audio signals you may need to use these MATLAB functions: **`audioread`**, **`audiowrite`**, **`soundsc`**, and **`pause`** [3].

(b) Determine the MSE between the original signal and the noisy signals you generated in part (a). What is the relationship between the computed MSEs and the variance of each Gaussian noise signal?

(c) Write a MATLAB code to read the noise-free signal in part (a) and add three different Gaussian noise signals with the SNRs (Signal-to-Noise-Ratio) [10 5.22 3.01].

Hint 1: In order to add a Gaussian noise to a signal with requested SNR you can use **`awgn`** MATLAB function as follow:

```
s_noisy = awgn(s, SNR); %Add Gaussian noise with requested SNR to s
```

Hint 2: For this purpose you can also use the following code segment:

```
sigPower = 1; %Signal power supposed to be 1
noisePower = sigPower / (10 ^ (reqSNR / 10)); %Power of the noise
signal considering the requested SNR
noise = sqrt(noisePower) * randn(1, L); % Noise signal of size L
x = x + noise; %Noisy signal with requested SNR
```

(d) Determine the MSE between the original signal and the noisy signals you generated in part (c). Derive the relationship between requested SNR and noise variance (use Hint 2 of part(c)). What is the relationship between the computed MSEs and the variance of each Gaussian noise?

Hint: Using Hint 2 of part (c), if we consider "$noisePower$" as the "$Variance$" of the noise we can derive a logarithmic formula which relates the requested SNR to the noise $Variance$.

(e) In the "Data_for_DSP_Labs"→"Lab 5" folder, there is a file with the name "HappyBirthday_Echo.wav" which is the echo corrupted signal of the original music signal. Determine the MSE between the original signal and this echo corrupted signal.

- **Problem 2.**

One method to reduce noise is applying a low-pass filter on the noisy signal. For each of the noisy signals created in Problem 1 parts (a) and (c), try to recover the original signal by designing an LTI lowpass filter using the MATLAB Filter Designer tool. Your LTI filter will take the corrupted signal as input. The output should be as close to the original signal as possible. Use *MSE* as a quality measure for this task and state the *MSE* that you obtain with your filter. Do the following

(a) Design your system using an FIR filter of length 40 with normalized $wpass = 0.3$, and $wstop = 0.35$.

(b) Design your system using an IIR Butterworth filter of length 40 with normalized $wc = 0.3$.

(c) Design your system using any filter you want. Try to reach less MSE than in parts (a) and (b).

Hint: In order to employ your designed filter via Filter Designer you can select export from file menu of "Filter Designer" window, and then set the parameters Export To: "MAT-File", Export As: "Objects", and finally determine the name of Discrete Filter (which is Hd by default) and then save it in your workspace as MAT-file (for example, myFilter.mat). Now for applying the filter on your input signal you can use the following code:

```
load myFilter.mat; %Your designed discrete filter (Hd) using
filterDesigner will be loaded
s_filtered = filter(Hd, s_noisy); %Filter the noisy signal (s_noisy)
```

- **Problem 3:**

A method to recover the echo corrupted audio signal in Problem 1 part (e) will be explained next. If the original audio signal is $x[n]$, in order to model the echo corrupted audio signal $x_{corrupt}[n]$, we can add a shifted version of the input signal $x[n-\Delta]$ attenuated by α as follow

Equation 5-3: $$\boldsymbol{x[n] + \alpha x[n-\Delta] = x_{corrupt}[n]}$$

The Z-transform of the above equation will be:

Equation 5-4:

$$X(Z)\left(1+\alpha Z^{-\Delta}\right) = X_{corrupt}(Z) \rightarrow X(Z) = \frac{X_{corrupt}(Z)}{(1+\alpha Z^{-\Delta})}$$

According to **Equation 5-4**, in order to recover $X(Z)$, we can filter $X_{corrupt}(Z)$ by $\frac{1}{(1+\alpha Z^{-\Delta})}$ which is an IIR filter (see Example 2 of Lab 2). But we need to find the value of α and Δ. α is an attenuation parameter and should be less than 1 (for example, 0.5). For finding the value of Δ we can use autocorrelation. To compute an unbiased estimate of the signal autocorrelation that corresponds to lags greater than zero we can use the following code segment, where Fs is the sampling frequency:

```
[R, lags] = xcorr(x_corrupt, 'unbiased');
R = R(lags > 0);
lags = lags(lags > 0);
plot(lags / Fs, R)
```

The autocorrelation has a sharp peak at the lag (Δ) at which the echo arrives; the lag can be found according to the following code segment:

```
[~, delta] = findpeaks(R, lags, 'MinPeakHeight', 0.08);
```

In the above code, **`findpeaks`** returns only those peaks higher than 'MinimumPeakHeight' (0.08), which can be seen checking the autocorrelation plot for this specific audio signal.

After calculating the value of Δ ('delta'), according to **Equation 5-4** the nominator and denominator coefficients of this IIR filter are now determined. Apply this filter on $x_{corrupt}[n]$ using the MATLAB function **`filter`** (similar to Example 2 of Lab 2). Calculate the MSE as a quality measure for this task and state the MSE that you obtain with this filter. Listen to the recovered signal and finally save it as an audio file with name "recovered_from_echo.wav".

References

[1] A. Oppenheim and R. Schafer, Discrete-Time Signal Processing, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1989.
[2] Signals and Systems, 2nd ed. , A.V. Oppenheim and A.S Willsky, Prentice-Hall, ISBN 0-13-814757-4, 1997.
[3] https://www.mathworks.com/help/pdf_doc/signal/signal_tb.pdf

Acknowledgment

Thank you to students, teaching assistants, lab coordinators, and course instructors for their comments and suggestions that helped to improve this lab manual over the years. The 2019 revision was assisted by Muhammad R. Pourshahabi. An early version of the DSP lab manual was coordinated by Dr. Rajeev Agarwal in 1994.

Latest revision: June 20, 2019