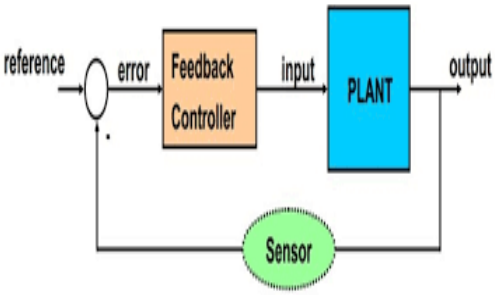
LABORATORY MANUAL

Digital Signal Processing
And
Control System Laboratory



Department of Instrumentation Engineering
JORHAT ENGINEERING COLLEGE
Assam-785007

Do’s

- Be punctual, maintain discipline & silence.
- Keep the Laboratory clean and tidy.
- Leave your shoes in the rack outside.
- Handle the equipments carefully.
- Save all your files properly.
- Come prepared with programs/algorithms/related manuals.
- Follow the procedure that has been instructed.
- Get the signature on experiment result sheet daily.
- For any clarification contact faculty/staff in charge only.
- Log off the system properly before switching off .

Don’ts

- Avoid unnecessary chat or walk.
- Disfiguring of furniture is prohibited.
- Avoid using cell phones unless absolutely necessary.
- Do not use personal pen drives without permission.
- Do not displace monitor, keyboard, mouse etc.
- Avoid late submission of laboratory reports.

IN505	DSP LABORATORY	Semester V	L-T-P 0-0-2	1 CREDIT

Experiment No.	Title of the Experiment	Objective of the Experiment
1	Representation of Basic signals i) Sine wave ii) Cosine wave iii) Unit impulse iv) Unit step wave v) Square wave vi) Exponential waveform vii) Sawtooth waveform	To distinguish between different waveforms.
2	i) Verification of sampling theorem ii) Demonstrate the effects of aliasing arising from improper sampling iii) Computation of Circular Convolution iv) Calculation of FFT	To illustrate the effect of sampling theorem.
3	Filtering by Convolution: i) Design of an analog filter: Band pass filter ii) Design of IIR filter using impulse invariance iii) Design of FIR filter: low pass filter iv) Design of FIR filter: highpass filter v) Design of FIR filter: bandstop filter vi) Design of FIR filter: bandpass filter	To make use of different filters in processing a signal
4	Estimation of power spectral density	To calculate the power of a signal
5	Application of Filtering techniques to a noisy biomedical signal and find its following features: Mean, Median, Auto correlation co-efficients.	To apply the concepts of signal processing to a biomedical signal and analyze it.
6	i) Introduction to DSP kit. ii) Implementation of Digital FIR and IIR filters on DSP Starter Kit	To experiment with the digital signal.

Text book: Digital Signal Processing Using MATLAB, John G Proakis, Vinay K. Ingle, Cengage Learning.

Student Profile	
Name	
Roll Number	
Department	
Year	

Student Performance		
Sl. No.	**Title of the Experiment**	**Remarks**
1	Representation of Basic signals i) Sine wave ii) Cosine wave iii) Unit impulse iv) Unit step wave v) Square wave vi) Exponential waveform Sawtooth waveform	
2	i) Verification of sampling theorem ii) Demonstrate the effects of aliasing arising from improper sampling iii) Computation of Circular Convolution iv) Calculation of FFT	
3	Filtering by Convolution: i) Design of an analog filter: Band pass filter ii) Design of IIR filter using impulse invariance iii) Design of FIR filter: low pass filter iv) Design of FIR filter: highpass filter v) Design of FIR filter: bandstop filter vi) Design of FIR filter: bandpass filter	
4	Estimation of power spectral density	
5	Application of Filtering techniques to a noisy biomedical signal and find its following features: Mean, Median, Auto correlation co-efficients.	
6	i) Introduction to DSP Kit ii) Implementation of Digital FIR and IIR filters on DSP Starter Kit	

Office Use	
Checked and found …… Grade/ Marks …… Signature ……	

EXPERIMENT: 1

TITLE: **Representation of Basic signals**

i) Sine wave
ii) Cosine wave
iii) Unit impulse
iv) Unit step wave
v) Square wave
vi) Exponential waveform
vii) Sawtooth waveform

AIM: To represent the basic signals using MATLAB

THEORY:

MATLAB is an interactive, matrix-based system for scientific and engineering numeric computation and visualization. Its strength lies in the fact that complex numerical problems can be solved easily and in a fraction of the time required by a programming language such as Fortran or C.

PROCEDURE:

```
% sine wave

t=0:0.01:1;
a=2; b=a*sin(2*pi*2*t); subplot(3,3,1); stem(t,b); xlabel('time'); ylabel('Amplitude'); title
('sinewave');

% Cosine wave

t=0:0.01:1;
a=2; b=a*cos(2*pi*2*t); subplot(3,3,2); stem(t,b); xlabel('time'); ylabel('Amplitude'); title ('Cos
wave');

% Square wave

t=0:0.01:1;
a=2; b=a*square(2*pi*2*t); subplot(3,3,3); stem(t,b); xlabel('time'); ylabel('Amplitude'); title
('square wave');

% Exponential waveform

t=0:0.01:1;
a=2;
b=a*exp(2*pi*2*t); subplot(3,3,4);
stem(t,b);
xlabel('time'); ylabel('Amplitude');
title ('exponential wave');

%sawtooth
```

```
t=0:0.01:1;
a=2; b=a*sawtooth(2*pi*2*t);

subplot(3,3,5); stem(t,b); xlabel('time'); ylabel('Amplitude'); title ('sawtooth wave');

% unit step signal

n=-5:5;
a = [zeros(1,5),ones(1,6)]; subplot(3,3,6);
stem(n,a);
Xlabel ('time');
Ylabel ('amplitude'); title('Unit step');

% unit impulse

n=-5:5;
a = [zeros(1,5),ones(1,1),zeros(1,5)]; subplot(3,3,7);
stem(n,a);
Xlabel ('time');
Ylabel ('amplitude');
title('Unit impulse');
```

RESULT:

Waveform	Graph	Graph from MATLAB

DISCUSSION:

EXPERIMENT: 2

TITLE:

i) Verification of sampling theorem
ii) Demonstrate the effects of aliasing arising from improper sampling
iii) Computation of Circular Convolution
iv) Calculation of FFT

AIM: To illustrate the effect of sampling theorem and perform Circular convolution and FFT on signal.

THEORY:

The definition of *proper sampling* is quite simple. Suppose you sample a continuous signal in some manner. If you can exactly *reconstruct* the analog signal from the samples, you must have done the sampling *properly*. Even if the sampled data appears confusing or incomplete, the key information has been captured if you can reverse the process.
Figure 3-3 shows several sinusoids before and after digitization. The continious line represents the analog signal entering the ADC, while the square markers are the digital signal leaving the ADC. In (a), the analog signal is a constant DC value, a cosine wave of *zero* frequency. Since the analog signal is a series of straight lines between each of the samples, all of the information needed to reconstruct the analog signal is contained in the digital data. According to our definition, this is *proper sampling.*
The sine wave shown in (b) has a frequency of 0.09 of the sampling rate. This might represent, for example, a 90 cycle/second sine wave being sampled at 1000 samples/second. Expressed in another way, there are 11.1 samples taken over each complete cycle of the sinusoid. This situation is more complicated than the previous case, because the analog signal cannot be reconstructed by simply drawing straight lines between the data points. Do these samples properly represent the analog signal? The answer is yes, because no other sinusoid, or combination of sinusoids, will produce this pattern of samples (within the reasonable constraints listed below). These samples correspond to only one analog signal, and therefore the analog signal can be exactly reconstructed. Again, an instance of *proper sampling*.
In (c), the situation is made more difficult by increasing the sine wave's frequency to 0.31 of the sampling rate. This results in only 3.2 samples per sine wave cycle. Here the samples are so sparse that they don't even appear to follow the general trend of the analog signal. Do these samples properly represent the analog waveform? Again, the answer is yes, and for exactly the same reason. The samples are a unique representation of the analog signal. All of the information needed to reconstruct the continuous waveform is contained in the digital data. How you go about doing this will be discussed later in this chapter. Obviously, it must be more sophisticated than just drawing straight lines between the data points. As strange as it seems, this is *proper sampling* according to our definition.
In (d), the analog frequency is pushed even higher to 0.95 of the sampling rate, with a mere 1.05 samples per sine wave cycle. Do these samples properly represent the data? *No, they don't!* The samples represent a *different* sine wave from the one contained in the analog signal. In particular, the original sine wave of 0.95 frequency misrepresents itself as a sine wave of 0.05 frequency in the digital signal. This phenomenon of sinusoids changing frequency during sampling is called **aliasing**. Just as a criminal might take on an assumed name or identity (an *alias*), the sinusoid assumes another frequency that is not its own. Since the digital data is no longer uniquely related to a particular analog signal, an unambiguous reconstruction is impossible. There is nothing in the sampled data to suggest that the original analog signal had a frequency of 0.95 rather than 0.05. The sine wave has hidden its true identity completely; the perfect crime has been committed! According to our definition, this is an example of *improper sampling*.

This line of reasoning leads to a milestone in DSP, the **sampling theorem**. Frequently this is called the *Shannon* sampling theorem, or the *Nyquist* sampling theorem, after the authors of 1940s papers on the topic. The sampling theorem indicates that a continuous signal can be properly sampled, *only if it does not contain frequency components above one-half of the sampling rate*. For instance, a sampling rate of 2,000 samples/second requires the analog signal to be composed of frequencies below 1000 cycles/second. If frequencies above this limit *are* present in the signal, they will be aliased to frequencies between 0 and 1000 cycles/second, combining with whatever information that was legitimately there.

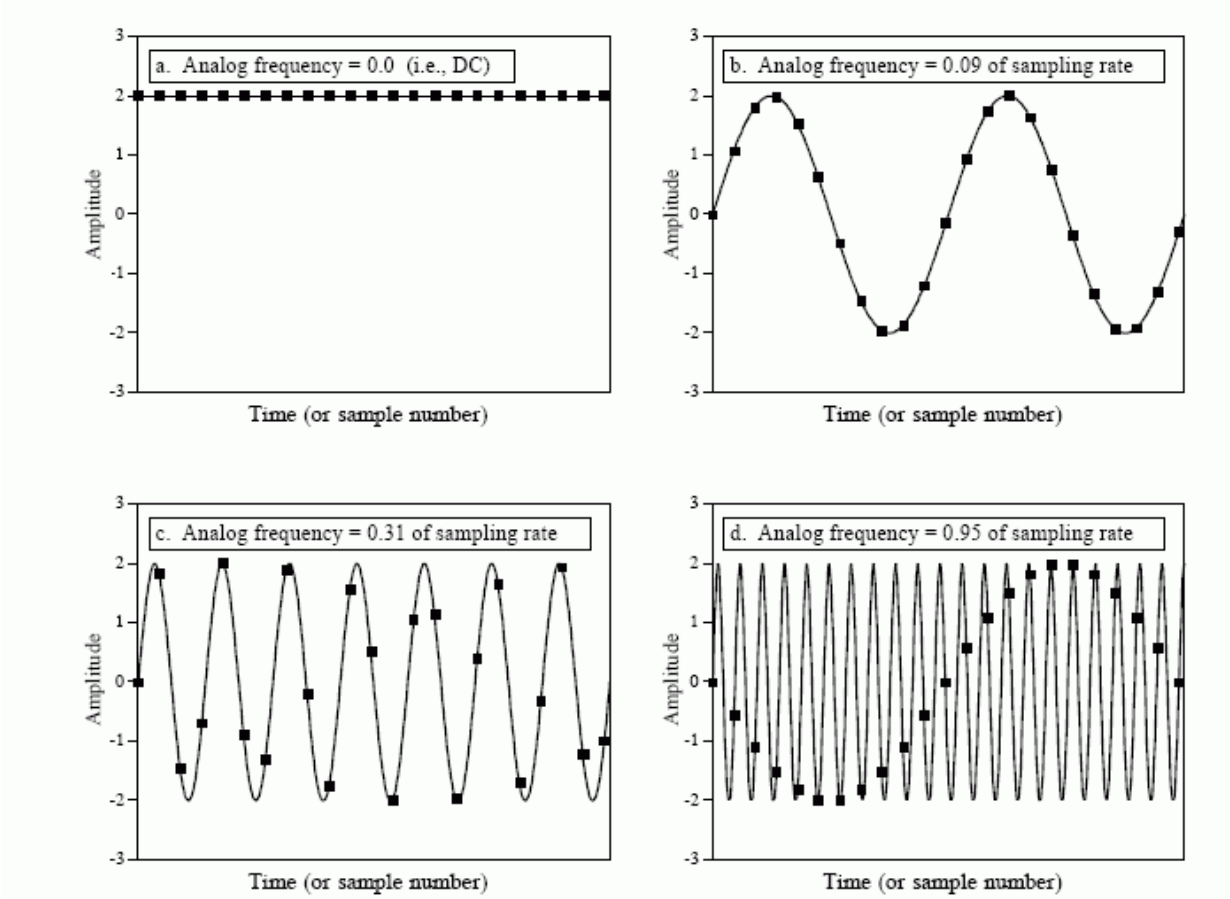


FIGURE 3-3
Illustration of proper and improper sampling. A continuous signal is sampled *properly* if the samples contain all the information needed to recreate the original waveform. Figures (a), (b), and (c) illustrate *proper sampling* of three sinusoidal waves. This is certainly not obvious, since the samples in (c) do not even appear to capture the shape of the waveform. Nevertheless, each of these continuous signals forms a unique one-to-one pair with its pattern of samples. This guarantees that reconstruction can take place. In (d), the frequency of the analog sine wave is greater than the Nyquist frequency (one-half of the sampling rate). This results in *aliasing*, where the frequency of the sampled data is different from the frequency of the continuous signal. Since aliasing has corrupted the information, the original signal cannot be reconstructed from the samples.

Two terms are widely used when discussing the sampling theorem: the **Nyquist frequency** and the **Nyquist rate**. Unfortunately, their meaning is not standardized. To understand this, consider an analog signal composed of frequencies between DC and 3 kHz. To properly digitize this signal it must be sampled at 6,000 samples/sec (6 kHz) or higher. Suppose we choose to sample at 8,000 samples/sec (8 kHz), allowing frequencies between DC and 4 kHz to be properly represented. In this situation their are four important frequencies: (1) the highest frequency in the signal, 3 kHz; (2) twice this frequency, 6 kHz; (3) the sampling rate, 8 kHz; and (4) one-half the sampling rate, 4 kHz. Which of these four is the *Nyquist frequency* and which is the *Nyquist rate*? It depends who you ask! All of the possible combinations are used. Fortunately, most authors are careful to define how they are using the terms. In this book, they are both used to mean *one-half the sampling rate*. Figure 3-4 shows how frequencies are changed during aliasing. The key point to remember is that a digital signal *cannot* contain frequencies above one-half the sampling rate (i.e., the Nyquist

When the frequency of the continuous wave is below the Nyquist rate, the frequency of the sampled data is a match. However, when the continuous signal's frequency is above the Nyquist rate, aliasing *changes* the frequency into something that *can* be represented in the sampled data. As shown by the zigzagging line in Fig. 3-4, every continuous frequency above the Nyquist rate has a corresponding digital frequency between zero and one-half the sampling rate. It there happens to be a sinusoid already at this lower frequency, the aliased signal will add to it, resulting in a loss of information. Aliasing is a double curse; information can be lost about the higher *and* the lower frequency. Suppose you are given a digital signal containing a frequency of 0.2 of the sampling rate. If this signal were obtained by proper sampling, the original analog signal *must* have had a frequency of 0.2. If aliasing took place during sampling, the digital frequency of 0.2 could have come from any one of an infinite number of frequencies in the analog signal: 0.2, 0.8, 1.2, 1.8, 2.2, … .

Just as aliasing can change the frequency during sampling, it can also change the *phase*. For example, look back at the aliased signal in Fig. 3-3d. The aliased digital signal is *inverted* from the original analog signal; one is a sine wave while the other is a negative sine wave. In other words, aliasing has changed the frequency *and* introduced a 180? phase shift. Only two phase shifts are possible: 0? (no phase shift) and 180? (inversion). The zero phase shift occurs for analog frequencies of 0 to 0.5, 1.0 to 1.5, 2.0 to 2.5, etc. An inverted phase occurs for analog frequencies of 0.5 to 1.0, 1.5 to 2.0, 3.5 to 4.0, and so on.

Now we will dive into a more detailed analysis of sampling and how aliasing occurs. Our overall goal is to understand what happens to the information when a signal is converted from a continuous to a discrete form. The problem is, these are very different things; one is a *continuous waveform* while the other is an *array of numbers*. This "apples-to-oranges" comparison makes the analysis very difficult. The solution is to introduce a theoretical concept called the **impulse train**. Figure 3-5a shows an example analog signal. Figure (c) shows the signal sampled by using an *impulse train*. The impulse train is a continuous signal consisting of a series of narrow spikes (impulses) that match the original signal at the sampling instants. Each impulse is infinitesimally narrow, a concept that will be discussed in Chapter 13. Between these sampling times the value of the waveform is zero. Keep in mind that the impulse train is a *theoretical* concept, not a waveform that can exist in an electronic circuit. Since both the original analog signal and the impulse train are continuous waveforms, we can make an "apples-apples" comparison between the two.

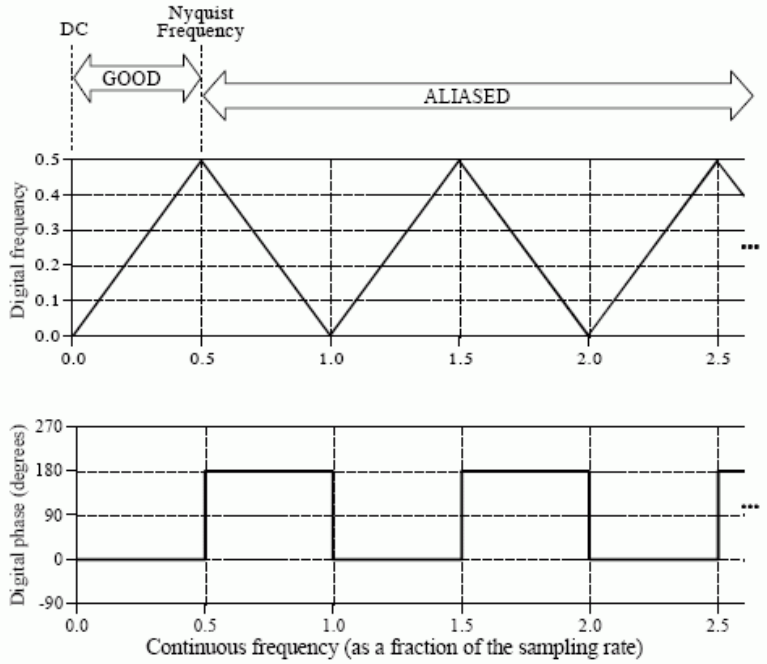


FIGURE 3-4
Conversion of analog frequency into digital frequency during sampling. Continuous signals with a frequency less than one-half of the sampling rate are directly converted into the corresponding digital frequency. Above one-half of the sampling rate, aliasing takes place, resulting in the frequency being misrepresented in the digital data. Aliasing always changes a higher frequency into a lower

Now we need to examine the relationship between the impulse train and the discrete signal (an array of numbers). This one is easy; in terms of *information content*, they are *identical*. If one is known, it is trivial to calculate the other. Think of these as different ends of a bridge crossing between the analog and digital worlds. This means we have achieved our overall goal once we understand the consequences of changing the waveform in Fig. 3-5a into the waveform in Fig. 3.5c.

Three continuous waveforms are shown in the left-hand column in Fig. 3-5. The corresponding *frequency spectra* of these signals are displayed in the right-hand column. This should be a familiar concept from you knowledge of electronics; every waveform can be viewed as being composed of sinusoids of varying amplitude and frequency. Later chapters will discuss the frequency domain in detail. (You may want to revisit this discussion after becoming more familiar with frequency spectra).

Figure (a) shows an analog signal we wish to sample. As indicated by its frequency spectrum in (b), it is composed only of frequency components between 0 and about 0.33 $f>_s$, where f_s is the sampling frequency we intend to use. For example, this might be a speech signal that has been filtered to remove all frequencies above 3.3 kHz. Correspondingly, f_s would be 10 kHz (10,000 samples/second), our intended sampling rate.

Sampling the signal in (a) by using an impulse train produces the signal shown in (c), and its frequency spectrum shown in (d). This spectrum is a *duplication* of the spectrum of the original signal. Each multiple of the sampling frequency, f_s, $2f_s$, $3f_s$, $4f_s$, etc., has received a *copy* and a *left-for-right flipped copy* of the original frequency spectrum. The copy is called the **upper sideband**, while the flipped copy is called the **lower sideband**. Sampling has generated *new* frequencies. Is this proper sampling? The answer is yes, because the signal in (c) can be transformed back into the signal in (a) by eliminating all frequencies above $?f_s$. That is, an analog low-pass filter will convert the impulse train, (b), back into the original analog signal, (a).

If you are already familiar with the basics of DSP, here is a more technical explanation of why this spectral duplication occurs. (Ignore this paragraph if you are new to DSP). In the time domain, sampling is achieved by multiplying the original signal by an impulse train of *unity amplitude* spikes. The frequency spectrum of this unity amplitude impulse train is also a unity amplitude impulse train, with the spikes occurring at multiples of the sampling frequency, f_s, $2f_s$, $3f_s$, $4f_s$, etc. When two time domain signals are multiplied, their frequency spectra are convolved. This results in the original spectrum being duplicated to the location of each spike in the impulse train's spectrum. Viewing the original signal as composed of both positive and negative frequencies accounts for the upper and lower sidebands, respectively. This is the same as amplitude modulation, discussed in Chapter 10.

Figure (e) shows an example of *improper sampling*, resulting from too low of sampling rate. The analog signal still contains frequencies up to 3.3 kHz, but the sampling rate has been lowered to 5 kHz. Notice that along the horizontal axis are spaced closer in (f) than in (d). The frequency spectrum, (f), shows the problem: the duplicated portions of the spectrum have invaded the band between zero and one-half of the sampling frequency. Although (f) shows these overlapping frequencies as retaining their separate identity, in actual practice they add together forming a single confused mess. Since there is no way to separate the overlapping frequencies, information is lost, and the original signal cannot be reconstructed. This overlap occurs when the analog signal contains frequencies greater than one-half the sampling rate, that is, we have proven the sampling theorem.

Convolution is a mathematical way of combining two signals to form a third signal. It is the single most important technique in Digital Signal Processing. Using the strategy of impulse decomposition, systems are described by a signal called the *impulse response*. Convolution is important because it relates the three signals of interest: the input signal, the output signal, and the impulse response. This chapter presents convolution from two different viewpoints, called the input side algorithm and the output side algorithm.

PROCEDURE:

i) Verification of sampling theorem

```
clc
clear all
close all

t=-100:01:100;
fm=0.02;
x=cos(2*pi*t*fm); subplot(2,2,1);
plot(t,x);
xlabel('time in sec');
ylabel('x(t)');
title('continuous time signal');
fs1=0.02;
n=-2:2;
x1=cos(2*pi*fm*n/fs1);
subplot(2,2,2);
stem(n,x1);
hold on
subplot(2,2,2);
plot(n,x1,':');
title('discrete time signal x(n) with fs<2fm');
xlabel('n');
ylabel('x(n)');
fs2=0.04;
n1=-4:4;
x2=cos(2*pi*fm*n1/fs2);
subplot(2,2,3);
stem(n1,x2);
hold on
subplot(2,2,3);
plot(n1,x2,':');
title('discrete time signal x(n) with fs>2fm');
xlabel('n');
ylabel('x(n)');
n2=-50:50;
fs3=0.5;
x3=cos(2*pi*fm*n2/fs3);
subplot(2,2,4);
stem(n2,x3);
hold on
subplot(2,2,4);
plot(n2,x3,':');
xlabel('n');
ylabel('x(n)');
title('discrete time signal x(n) with fs=2fm');
```

ii) Demonstrate the effects of aliasing arising from improper sampling

1. Consider an analog signal x(t) consisting of three sinusoids of frequencies of 1 kHz, 4 kHz, and 6 kHz:

$$x(t) = \cos(2\pi t) + \cos(8\pi t) + \cos(12\pi t)$$

where t is in milliseconds. Show that if this signal is sampled at a rate of fs = 5 kHz, it will be aliasedwith the following signal, in the sense that their sample values will be the same:

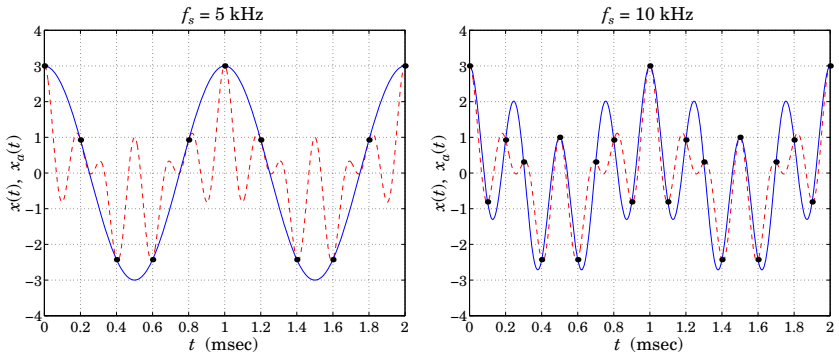
$$x_a(t) = 3\cos(2\pi t)$$

On the same graph, plot the two signals x(t) and xa(t) versus t in the range $0 \le t \le 2$ msec. To this plot, add the time samples x(tn) and verify that x(t) and xa(t) intersect precisely at these samples. These samples can be evaluated and plotted as follows:

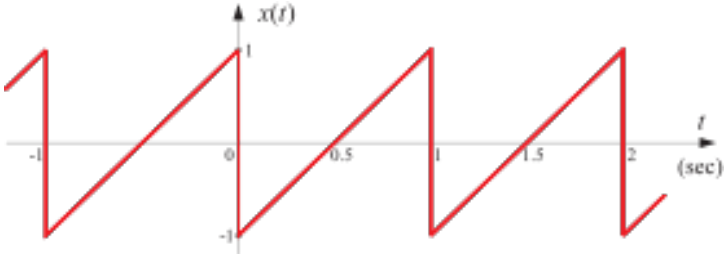
```
fs = 5; T = 1/fs;
tn = 0:T:2; xn = x(tn);
plot(tn, xn, '.');
```

2. Repeat part (a) with fs = 10 kHz. In this case, determine the signal $x_a(t)$ with which x(t) is aliased.

   Plot both x(t) and xa(t) on the same graph over the same range $0 \le t \le 2$ msec. Verify again that the two signals intersect at the sampling instants.



3. Consider a periodic sawtooth wave x(t) with period T0 = 1 sec shown below:



Mathematically, x(t) is defined as follows over one period, that is, over the time interval $0 \le t \le 1$:

$$x(t) = \begin{cases} 2t - 1, & \text{if } 0 < t < 1 \\ 0, & \text{if } t = 0 \text{ or } t = 1 \end{cases}$$

where t is in units of seconds. This periodic signal admits a Fourier series expansion containing only sine terms with harmonics at the frequencies fm = m/T0 , m = 1, 2, 3, 4, . . . Hz:

$$x(t) = \sum_{m=1}^{\infty} b_m \sin(2\pi m t) = b_1 \sin(2\pi t) + b_2 \sin(4\pi t) + b_3 \sin(6\pi t) + \cdots$$

The reason why the signal x(t) was defined to have the value zero at the discontinuity points has to do with a theorem that states that any finite sum of Fourier series terms will always pass through the mid-points of discontinuities.

4. To plot the above sawtooth waveform in MATLAB, you may use the functions upulse and ustep, which can be downloaded from the lab web page (see the link for the supplementary M-files.) The following code segment will generate and plot the above sawtooth waveform over one period:

```
x = inline('upulse(t-0.5,0,0.5,0) - upulse(t,0,0,0.5)');
    t = linspace(0,1,1001);
    plot(t, x(t));
```

In order to plot three periods, use the following example code that adds three shifted copies of the basic period defined in Eq. (1.1):

```
    t = linspace(0,3,3001);
    plot(t, x(t) + x(t-1) + x(t-2));
```

Repeat the above plot of the sawtooth waveform and xM(t) for the case M = 10, and then for M = 20.

The ripples that you see accumulating near the sawtooth discontinuities as M increases are an example of the so-called Gibbs phenomenon in Fourier series.

5. The sawtooth waveform x(t) is now sampled at the rate of fs = 5 Hz and the resulting samples are reconstructed by an ideal reconstructor. Using the methods of Example 1.4.6 of the text [1], show that the aliased signal xa(t) at the output of the reconstructor will have the form:

$$x_a(t)= a1\ \sin(2\pi t)+a2\ \sin(4\pi t)$$

Determine the coefficients a1 , a2 . On the same graph, plot one period of the sawtooth wave x(t) together with xa(t). Verify that they agree at the five sampling time instants that lie within this period.

6. Assume, next, that the sawtooth waveform x(t) is sampled at the rate of fs = 10 Hz. Show now that the aliased signal xa(t) will have the form:

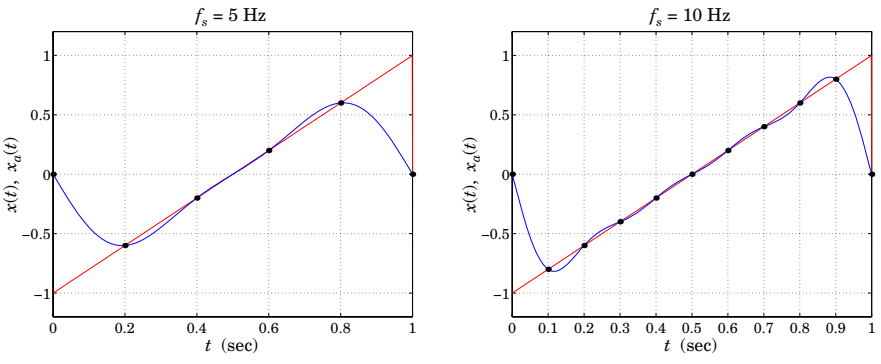
$$x_a(t)= a1\ \sin(2\pi t)+a2\ \sin(4\pi t)+a3\ \sin(6\pi t)+a4\ \sin(8\pi t)$$

where the coefficients a_i are obtained by the condition that the signals x(t) and xa(t) agree at the first four sampling instants $t_n= nT = n/10$ Hz, for n = 1, 2, 3, 4. These four conditions can be arranged into a 4×4 matrix equation of the form:

$$\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} * \\ * \\ * \\ * \end{bmatrix}$$

Determine the numerical values of the starred entries and explain your approach. Then, using MATLAB, solve this matrix equation for the coefficients a_i. Once a_i are known, the signal $x_a(t)$ is completely defined.

On the same graph, plot one period of the sawtooth waveform x(t) together with xa(t). Verify that they agree at the 10 sampling time instants that lie within this period.



iii) Circular Convolution

```
clc;
clear all;
close all;
x=input('enter the sequence');
h=input('enter the imp response'); % no of samples in x
n1=length(x);
n2=length(h);
n3=n1+n2-1;
n=max(n1,n2);
if(n3>=0)
h=[h,zeros(1,n3)];
else
x=[n,zeros(1,-n3)]; end;
for a =1:n y(a)=0;
for i=1:n j=a-i+1;
if(j<=0) j=n+j;
end;

y(a)=y(a)+[x(i)*h(j)]; end;
end; t=1:n
stem(t,y);
disp(y);
title('circular convolution');
xlabel('samples');
ylabel('amplitude');
```

iv) Calculation of FFT

```
function[Xk]=dft(x,N);
x=[1 1 1 1 4 5 6 1]; N=8;
n=[0:1:N-1]; k=[0:1:N-1];
a=(-i*2*pi/N); WN=exp(a);
nk=n'*k;
WNnk=WN.^nk;
Xk=x*WNnk
```

RESULT:

Verification of sampling Theorem

Show the graphs obtained from the execution of the program

Demonstrate the effects of aliasing arising from improper sampling

Show the plots from the experiment

Circular Convolution

1. Enter a sequence and impulse response to calculate the result. Perform it taking three different cases

Sequence x[n]	Impulse reponse h[n]	Result y[n]	GRAPH

Calculation of FFT

Change the values of X to find FFT of the signal. Do it for three different signals .

DISCUSSION:

EXPERIMENT: 3

TITLE:**Filtering by Convolution:**

i) Design of an analog filter: Band pass filter
ii) Design of IIR filter using impulse invariance
iii) Design of FIR filter: low pass filter
iv) Design of FIR filter: highpass filter
v) Design of FIR filter: bandstop filter
vi) Design of FIR filter: bandpass filter

AIM: To analyze different FILTERS
THEORY:
Digital and analog filters both take out unwanted noise or signal components, but filters work differently in the analog and digital domains. Analog filters will remove everything above or below a chosen cutoff frequency whereas digital filters can be more precisely programmed. Analog filters that remove signals above a certain frequency are called low pass filters because they let low frequency signals *pass* through the filter while blocking everything above the cutoff frequency. An analog filter that removes all signals below a certain frequency is a high pass filter, because it lets pass everything higher than the cutofffrequency.Finite impulse response (FIR) filters are the most popular type of filters implemented in software.A digital filter takes a digital input, gives a digital output, and consists of digital components. In a typical digital filtering application, software running on a digital signal processor (DSP) reads input samples from an A/D converter, performs the mathematical manipulations dictated by theory for the required filter type, and outputs the result via a D/A converter.There are many filter types, but the most common are lowpass, highpass, bandpass, and bandstop.
A lowpass filter allows only low frequency signals (below some specified cutoff) through to its output, so it can be used to eliminate high frequencies. A lowpass filter is handy, in that regard, for limiting the uppermost range of frequencies in an audio signal; it's the type of filter that a phone line resembles.
A highpass filter does just the opposite, by rejecting only frequency components below some threshold.
A bandpass filter only allow a certain band of frequency components to pass through it while rejecting the others.

A finite impulse response (FIR) filter is a filter structure that can be used to implement almost any sort of frequency response digitally. An FIR filter is usually implemented by using a series of delays, multipliers, and adders to create the filter's output.Figure 1 shows the basic block diagram for an FIR filter of length N. The delays result in operating on prior input samples. The h_k values are the coefficients used for multiplication, so that the output at time n is the summation of all the delayed samples multiplied by the appropriate coefficients.

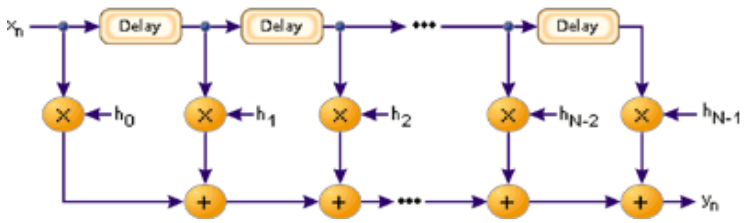


Figure 1. The logical structure of an FIR filter

The process of selecting the filter's length and coefficients is called filter design. The goal is to set those parameters such that certain desired stopband and passband parameters will result from running the filter. Most engineers utilize a program such as MATLAB to do their filter design

- A frequency response plot, like the one shown in Figure 1, which verifies that the filter meets the desired specifications, including ripple and transition bandwidth.
- The filter's length and coefficients.

The longer the filter (more taps), the more finely the response can be tuned.

With the length, N, and coefficients, float h[N] = { ... }, decided upon, the implementation of the FIR filter is fairly straightforward. Listing 1 shows how it could be done in C. Running this code on a processor with a multiply-and-accumulate instruction (and a compiler that knows how to use it) is essential to achieving a large number of taps.

The impulse invariance method of IIR filter design is based upon the notion that we can design a discrete filter whose time-domain impulse response is a sampled version of the impulse response of a continuous analog filter. If that analog filter (often called the prototype filter) has some desired frequency response, then our IIR filter will yield a discrete approximation of that desired response. The impulse response equivalence of this design method is depicted in Figure 2, where we use the conventional notation of d to represent an impulse function and hc(t) is the analog filter's impulse response. We use the subscript "c" in Figure 2 (a) to emphasize the continuous nature of the analog filter.

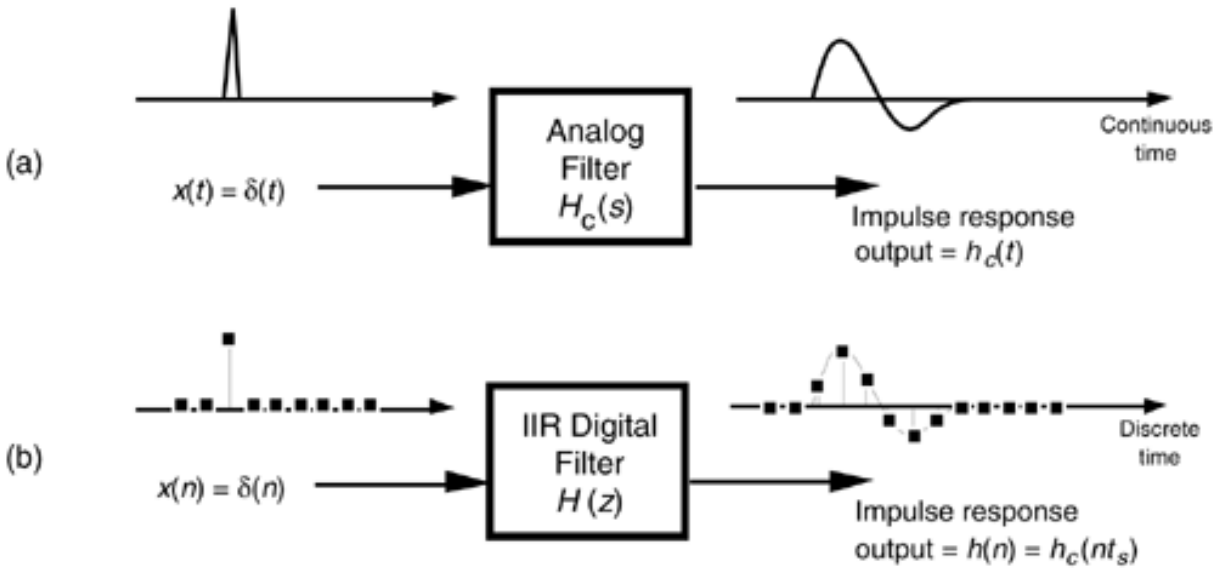


Figure 2. Impulse invariance design equivalence of
(a) analog filter continuous impulse response; (b) digital filter discrete impulse response.

Figure 2(b) illustrates the definition of the discrete filter's impulse response: the filter's time-domain output sequence when the input is a single unity-valued sample (impulse) preceded and followed by all zero-valued samples. Our goal is to design a digital filter whose impulse response is a sampled version of the analog filter's continuous impulse response. Implied in the correspondence of the continuous and discrete impulse responses is the property that we can map each pole on the s-plane for the analog filter's Hc(s) transfer function to a pole on the z-plane for the discrete IIR filter's H(z) transfer function. The designers have found is that the impulse invariance method does yield useful IIR filters, as long as the sampling rate is high relative to the bandwidth of the signal to be filtered. In other words, IIR filters designed using the impulse invariance method are susceptible to aliasing problems because practical analog filters cannot be perfectly band-limited.

PROCEDURE:

i) Analog filter: Band pass filter
clear all;

```
rp = input(‘pass ripple freq’);
rs = input(‘stop ripple freq’);
fp = input(‘pass band freq’);
fs = input(‘stop band freq’);
f = input(‘sample freq’);
w1 = 2*fp/f;
w2= 2*fs/f;
[nJ= buttord(w1 ,w2,rp,rs); wn= [w1,w2];
[b,a]= butter(n,wn,’bandpass’); w=0:0.1:pi;
[h,p]= freqz(b,a,w);
g= 20*log10(abs(h)); A=angle(h);
subplot (2,2,1); plot(p/pi,g); ylabel(‘amp’);
xlabel(‘ferq’);
title(‘amp,freq’);
subplot (2,2,2); plot(p/pi,A); xlabel(‘normal. freq’); ylabel(‘phase’);
title(‘normal. freq,phas&);
```

ii) IIR filter using impulse invariance: a sixth-order analog Butterworth lowpass filter to a digital filter using impulse invariance

```
 f = 2;
fs = 10;

[b,a] = butter(6,2*pi*f,'s');
[bz,az] = impinvar(b,a,fs);

freqz(bz,az,1024,fs)
```

iii) FIR filter: low pass filter

```
clc;
clear all;
wc=input('enter the value of cut off frequency');
N=input('enter the value of filter');
alpha=(N-1)/2;
eps=0.001;
%Rectangular Window
n=0:1:N-1;
hd=sin(wc*(n-alpha+eps))./(pi*(n-alpha+eps));
hn=hd
w=0:0.01:pi;
h=freqz(hn,1,w);
plot(w/pi,abs(h));
hold on
%Hamming Window
n=0:1:N-1;
wh=0.54-0.46*cos((2*pi*n)/(N-1));
hn=hd.*wh
w=0:0.01:pi;
h=freqz(hn,1,w);
plot(w/pi,abs(h),'ms');
hold off;
hold on
%Hanning Window
n=0:1:N-1;
```

```
hn=hd.*wh
w=0:0.01:pi;
h=freqz(hn,1,w);
plot(w/pi,abs(h),'blue');
hold off;
hold on
%Blackman Window
n=0:1:N-1; wh=0.42-0.5*cos((2*pi*n)/(N-1))+0.08*cos((4*pi*n)/(N-1));
hn=hd.*wh
w=0:0.01:pi;
h=freqz(hn,1,w);
plot(w/pi,abs(h),'green');
hold off;
```

iv) FIR filter: highpass filter

```
clc;
clear all;
wc=input('enter the value of cut off frequency');
N=input('enter the value of filter');
alpha=(N-1)/2;
eps=0.001;
%Rectangular Window
n=0:1:N-1;
hd=(sin(pi*(n-alpha+eps))-sin((n-alpha+eps)*wc))./(pi*(n-alpha+eps)); hn=hd
w=0:0.01:pi;
h=freqz(hn,1,w);
plot(w/pi,abs(h));
hold on
%Hamming Window
n=0:1:N-1;
wh=0.54-0.46*cos((2*pi*n)/(N-1));
hn=hd.*wh
w=0:0.01:pi;
h=freqz(hn,1,w);
plot(w/pi,abs(h),'ms');
hold off;
hold on
%Hanning Window
n=0:1:N-1;
wh=0.5-0.5*cos((2*pi*n)/(N-1));
hn=hd.*wh
w=0:0.01:pi;
h=freqz(hn,1,w);
plot(w/pi,abs(h),'blue');
hold off;
hold on
%Blackman Window
n=0:1:N-1; wh=0.42-0.5*cos((2*pi*n)/(N-1))-0.08*cos((4*pi*n)/(N-1));
hn=hd.*wh
w=0:0.01:pi;
h=freqz(hn,1,w);
plot(w/pi,abs(h),'green');
hold off;
```

v) FIR filter: bandstop filter

```
clc;
Wc1=input('enter the value of Wc1=');
Wc2=input('enter the value of Wc2=');
N=input('enter the value of N=');
alpha=(N-1)/2;
eps=0.001;
%Rectangular Window
n=0:1:N-1; hd=(sin(Wc1*(n-alpha+eps))-sin(Wc2*(n-alpha+eps)*pi))./((n-
alpha+eps)*pi); hn=hd
W=0:0.01:pi;
h=freqz(hn,1,W);
plot(W/pi,abs(h));
hold on;
%Hamming Window
n=0:1:N-1;
Wn=0.54-0.46*cos((2*pi*n)/(N-1));
hn=hd.*Wn
W=0:0.01:pi;
h=freqz(hn,1,W);
plot(W/pi,abs(h),'green');
hold on;
%Hanning Window
n=0:1:N-1;
Wn=0.5-0.5*cos((2*pi*n)/(N-1));
hn=hd.*Wn
W=0:0.01:pi;
h=freqz(hn,1,W);
plot(W/pi,abs(h),'red');
hold off;
%Blackman Window
n=0:1:N-1;
wh=0.42-0.5*cos((2*pi*n)/(N-1))-0.08*cos((4*pi*n)/(N-1));
hn=hd.*wh
w=0:0.01:pi;
h=freqz(hn,1,w);
plot(W/pi,abs(h),'green');
hold off;
```

vi) FIR filter: bandpass filter

```
clc;
Wc1=input('enter the value of Wc1=');
Wc2=input('enter the value of Wc2=');
N=input('enter the value of N=');
alpha=(N-1)/2;
eps=0.001;
%Rectangular Window
n=0:1:N-1;
hd=(sin(Wc1*(n-alpha+eps))-sin(Wc2*(n-alpha+eps)*pi))./((n-alpha+eps)*pi);
hn=hd
W=0:0.01:pi;
h=freqz(hn,1,W);
plot(W/pi,abs(h));
hold on;
%Hamming Window
```

```
Wn=0.54-0.46*cos((2*pi*n)/(N-1));
        hn=hd.*Wn
        W=0:0.01:pi;
        h=freqz(hn,1,W);
        plot(W/pi,abs(h),'green');
        hold on;
        %Hanning Window
        n=0:1:N-1;
        Wn=0.5-0.5*cos((2*pi*n)/(N-1));
        hn=hd.*Wn
        W=0:0.01:pi;
        h=freqz(hn,1,W);
        plot(W/pi,abs(h),'red');
        hold off;
        %Blackman Window
        n=0:1:N-1;
        wh=042-0.5*cos((2*pi*n)/(N-1))-0.08*cos((4*pi*n)/(N-1));
        hn=hd.*wh
        w=0:0.01:pi;
        h=freqz(hn,1,w);
        plot(W/pi,abs(h),'green');
        hold off;
```

RESULT:

FILTER	FREQUENCY SPECIFICATION	GRAPH

DISCUSSION:

EXPERIMENT: 4

TITLE:Estimation of power spectral density

AIM: To calculate the power spectral density of a signal

THEORY:

Power spectral density function (PSD) shows the strength of the variations(energy) as a function of frequency. In other words, it shows at which frequencies variations are strong and at which frequencies variations are weak. The unit of PSD is energy per frequency(width) and you can obtain energy within a specific frequency range by integrating PSD within that frequency range. Computation of PSD is done directly by the method called FFT or computing autocorrelation function and then transforming it.PSD is a very useful tool if you want to know frequencies and amplitudes of oscillatory signals in your time series data.

PROCEDURE:

```
clc;
clear all;

%% Define Parameters
% sampling frequency (Hz)
fs=100e6;
% length of time-domain signal
L=30e3;
% desired power specral density (dBm/Hz)
Pd=-100;
% number of FFT points
nfft=2^nextpow2(L);
% frequency plotting vector
f=fs/2*[-1:2/nfft:1-2/nfft];
% create
s=wgn(L,1,Pd+10*log10(fs),1,[],'dBm','complex');
%% Analysis
% analyze spectrum
N=nfft/2+1:nfft;
S=fftshift(fft(s,nfft));
S=abs(S)/sqrt(L*fs);
% time-average for spectrum
Navg=4e2;
b(1:Navg)=1/Navg;
Sa=filtfilt(b,1,S);
% convert to dBm/Hz
S=20*log10(S)+30;
Sa=20*log10(Sa)+30;
%% Plot
figure(1)
plot(f(N)/1e6,S(N));
hold on
plot(f(N)/1e6,Sa(N),'r')
xlabel('Frequency (MHz)')
ylabel('Power Density (dBm/Hz)')
title('PowerSpectral Density')
legend('Noise Spectrum','Time-Averaged Spectrum')
axis([10e-4 fs/2/1e6 -120 -60])
grid on
hold off
```

RESULT: Show the graph obtained

DISCUSSION:

EXPERIMENT: 5

TITLE:Application of Filtering techniques to a noisy biomedical signal and find its following features: Mean, Median frequency, Auto correlation co-efficients.

AIM: To analyze a noisy biomedical signal.

THEORY:

Inspite of the recent developments in the field of Biomedical Instrumentation and its extensive use in healthcare and research, many practical problems are encountered in the biomedical signal acquisition, processing and analysis. Inorder to study the signal in details, it is very essential to remove the artifacts in the signal. For example, Surface EMG includes the interference pattern of the activities of several motor units even at very low levels of muscular contraction.Feature extraction is a significant method to extract the useful information which is hidden in surface electromyography (EMG) signal and to remove the unwanted part and interferences. To be successful in classification of the EMG signal, selection of a feature vector ought to be carefully considered. Here you have to select Mean, Median, Auto correlation co-efficients. Let us take an EMG signal during flexion and extension. The EMG signals acquired during hand relaxation or at resting position were of constant amplitude. These irrelevant EMG signals were discarded by the EMG onset detection technique. The EMG having amplitude more than three times of the standard deviation (SD) of the EMG at resting position were extracted. The threshold value was fixed at three times the SD by observing the amplitude difference of the EMG signal during finger movements and at resting position.

Mean: It is the sum of the observations divided by the number of observations. It identifies the central location of the data and sometimes referred to as the average. The mean is calculated as follows:

$$M=\frac{\sum_{n=1}^{N} Xn}{N}$$

Where N and X_n are the length and n^{th}sample value of EMG respectively.

Median Frequency: Median frequency (MDF) are estimated based on power spectrum of the EMG signal and are expressed as

$$MDF=\frac{1}{2}\sum_{j=1}^{M} Pj$$

where Pj and fj are EMG power spectrum at frequency bin j respectively. M is the length of the frequency bin.

Auto-Regressive Coefficients: Auto-Regressive (AR) model is a prediction model which specifies that the output variable depends linearly on its own previous samples x_{ip}plus a white noise error term w_i. We used fourth order AR model which indicates the muscle force. The AR model is expressed as:

$$X_i=\sum_{p=1}^{P} a_p x_i + w_i$$

where P is the order of the AR model.

PROCEDURE

1. Take an EMG signal from the data base provided

2. Apply Onset Detection followed by suitable Filters done in the previous lab sessions to obtain the pre processed signal.
3. Find the features from the signals using the following function:

```
function [FEA_ch1,FEA_ch2]=func(B,e)
% B=denoised reconstructed signal from wavelets
% e= count

 ch1=B(:,1);
 ch2=B(:,2);

% MEAN::
sum_ch1 = 0;
sum_ch2 = 0;

N = length(ch2);
fori = 1:n
   sum_ch1 = sum_ch1 + ch1(i);
   sum_ch2 = sum_ch2 + ch2(i);
end
mean_ch1 = (sum_ch1/N);

mean_ch2 = (sum_ch2/N);

% AUTO_REGRESSIVE COEFFICIENT::
p=4;
a1 = arburg(ch1, p);
a2 = arburg(ch2, p);

% MODIFIED MEAN FREQUENCY::

sum_y= 0;
sum_u = 0;

sum_ch1 = 0;
sum_ch2 = 0;
sum_1 = 0;
sum_2 = 0;

n = length(ch1);

fori = 1:n
   sum_ch1 = sum_ch1 + ch1(i);
   sum_ch2 = sum_ch2 + ch2(i);
end
q=A(:,1);
fori = 1:n
   sum_1 = sum_1 + ch1(i)*q(i);
   sum_2 = sum_2 + ch2(i)*q(i);
end

mmnf_ch1 = sum_1/sum_ch1;
mmnf_ch2 = sum_2/sum_ch2;
```

RESULT:

Perform the experiment for three different EMG signal

ORIGINAL SIGNAL	DENOISED SIGNAL	FEATURE

EXPERIMENT: 6

TITLE:

i) **I**ntroduction to DSP Kit : Scientech 2653: DSP Spectrum analyzer
ii) Implementation of Digital FIR and IIR filters on DSP Starter Kit

AIM: To experiment with digital signal.

THEORY:

The hardware experiments in the DSP lab are carried out on the Texas Instruments TMS320C6713 DSP Starter Kit (DSK), based on the TMS320C6713 floating point DSP running at 225 MHz. The basic clock cycle instruction time is 1/(225 MHz)= 4.44 nanoseconds. During each clock cycle, up to eight instructions can be carried out in parallel, achieving up to 8×225 = 1800 million instructions per second (MIPS). DSP Spectrum Analyzer Trainer has been specially designed for learning how to use a Spectrum Analyzer for Frequency and Level Measurements and Tracking Generator applications like HF Filters and Amplifier Response, Channel Modulation, Mixers, study of Mobile and Cordless signals and study of Harmonics in sine, square & triangular waves. TV & FM transmitted signals can be analyzed on Spectrum Analyzer using this Trainer and audio output can be heard on the speaker mounted inside the trainer.

Application I

Filter Responses : Four filters, very precisely designed for checking filter responses on Spectrum Analyzer.

Types of filters are :

1. Low Pass Filter (Cut off 115 MHz approximately)
2. High Pass Filter (Cut off 100 MHz approximately)
3. Wide Band pass Filter (Cut off 25 MHz - 280 MHz approximately)
4. Notch Filter or Narrow Band Reject Filter (Cut off 95 MHz -160 MHz approx.)

Application II

DC Amplifier Frequency Response : A well designed DC Amplifier having 3 dB cut off bandwidth of approx. 130 MHz

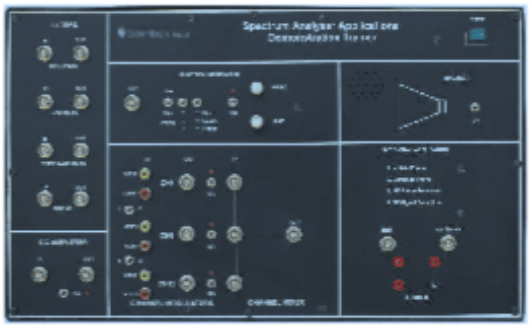
Application III

Harmonic Display & Analysis : A Function Generator of frequency range approx. 40 KHz - 400 KHz (Low) and 200 KHz - 2 MHz (High) having sine, square, & triangular output with frequency variation and output level variation (Max.1 V_{pp}). Any one output can be given to Spectrum Analyzer for harmonic display.

Application IV Cable TV application :
Three channel modulator (channel 4, channel 8, & channel 12) & one mixer is cascaded. Channel modulations have Audio and Video inputs. The output of modulators and mixer can be seen on Spectrum Analyzer.

Application V Live signal Application : Four types of signal can be seen on Spectrum Analyzer 1. Mobile phone forward and reverse link frequency 2. Cordless phone Transmitting and receiving frequencies 3. FM Radio Reception (Demodulated Audio output) 4. TV signal Reception (Demodulated Audio output) Demodulated output from Spectrum analyzer is given to the built- in speaker via Audio amplifier. A built in telescopic Antenna is provided with Spectrum Analyzer

Power Consumption : 2 VA (approximately)



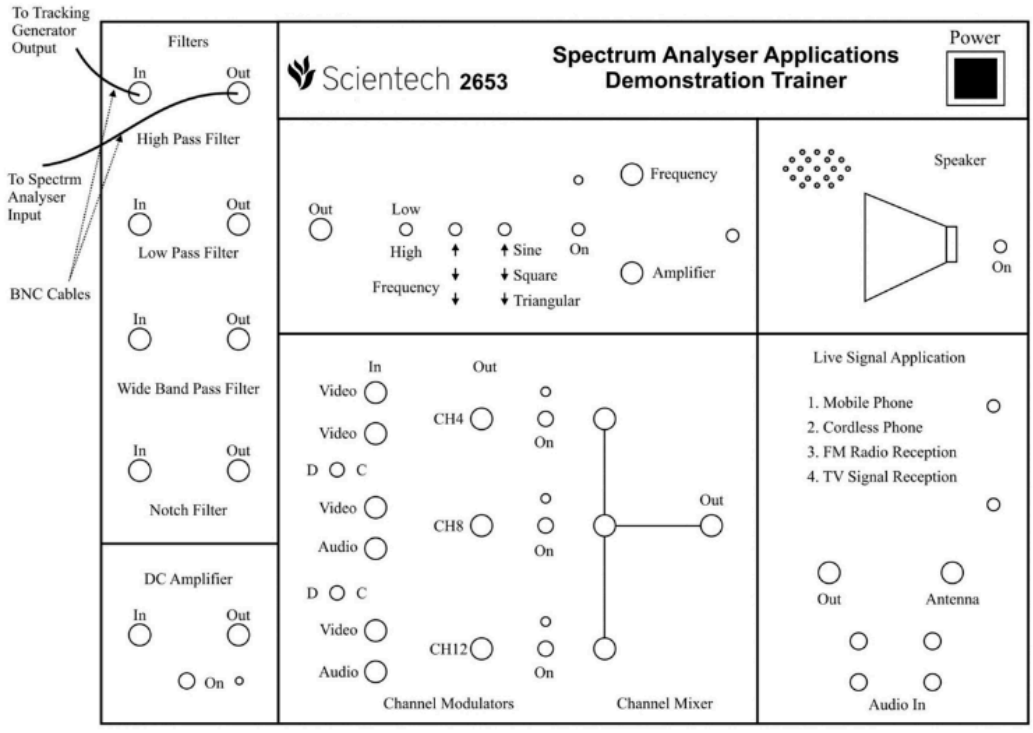
Observing the Frequency response of High Pass Filter:

- Make connection as shown in figure.
- Switch 'On' the spectrum analyzer.
- Set the Spectrum Analyzer as given below.

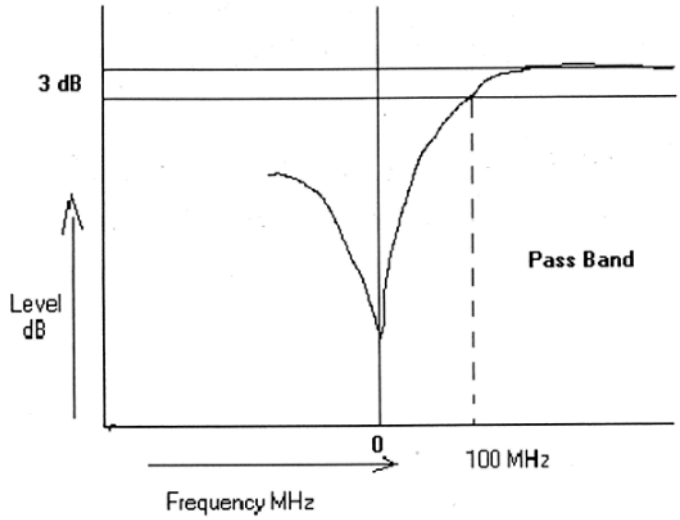
  Center Frequency : 000.0
  Attenuation : 10 dB = (10dB ×1) Pressed
  Scan Width : 2 MHz/Div.
- Set the tracking generator as below :

  Attenuation : Nil
  'On/Off' Switch : 'On'
  Level : +1dB (Clockwise)
- Keep all other functions/experiments on the trainer in 'Off' condition.
- Keep speaker 'Off'.

Connection Diagram:



- You will observe the curve as shown below (figure) adjust Y-POS knob of spectrum analyzer so that the top of the figure gets aligned with horizontal line of graticule.



- The 3dB below the highest level shall be the cutoff frequency of High Pass Filter and pass band is shown in figure.

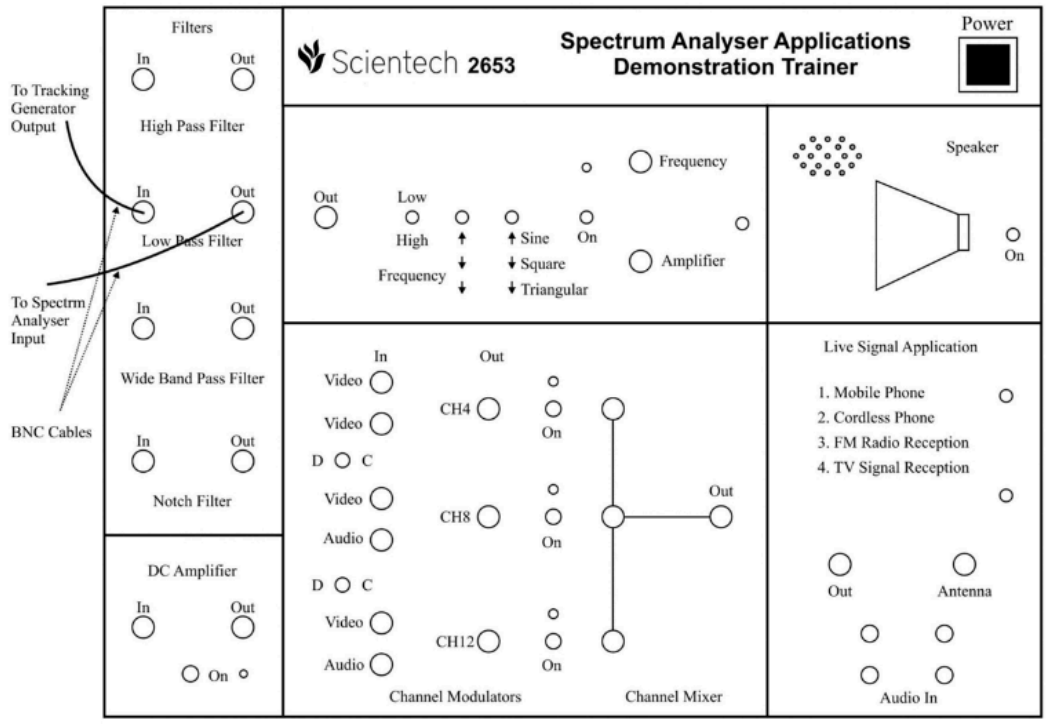
Observing the Frequency response of Low Pass Filter:

- Make connection as shown in figure.
- Switch 'On' the spectrum analyzer.
- Set the Spectrum Analyzer as given below :

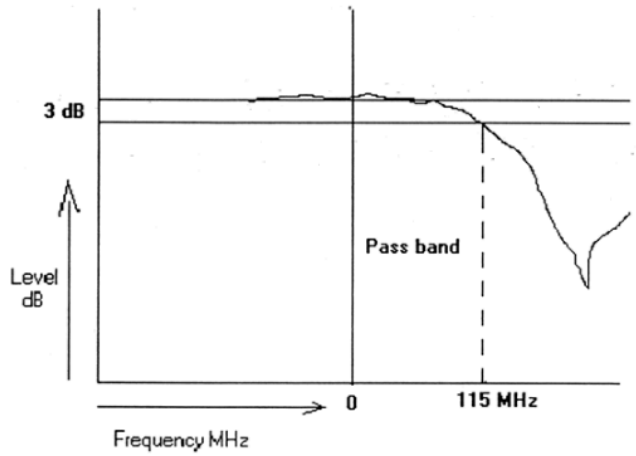
  Center Frequency : 000.0
  Attenuation : 10 dB = (10dB ×1) Pressed
  Scan Width : 50 MHz/Div
- Set the tracking generator as below :

  'On/Off' Switch : 'On'
  Attenuation : Nil
  Level : + 1dB (Clockwise)
- Keep all other functions/experiments on the trainer in 'Off' condition.
- Keep speaker 'Off'.

Connection Diagram:



- You will observe the curve as shown below (figure) adjust Y-POS knob of spectrum analyzer so that the top of the figure gets aligned with horizontal line of graticule.



- The 3dB below the highest level shall be the cutoff frequency of low pass filter and pass band is shown in figure.

iii) Implementation of Digital FIR and IIR filters on DSP Starter Kit

Lab requirements:

i) DSK Board.
ii) CCS software installed on the computer.
iii) Oscilloscope.
iv) Headphones

THEORY

The moving average filter is widely used in DSP and arguably is the easiest of all digital filters to understand. It is particularly effective at removing (high frequency) random noise from a signal or at smoothing a signal. The moving average filter operates by taking the arithmetic mean of a number of past input samples in order to produce each output sample. This may be represented by the equation

Where x (n) represents the n th sample of an input signal and y (n) the n th sample of the filter output. The moving average filter is an example of convolution using a very simple filter kernel or impulse response comprising N coefficients each of value 1 /N. The above equation may be thought of as a particularly simple case of the more general convolution sum implemented by a finite impulse response filter; that is,

where the FIR filter coefficients h (i) are samples of the filter impulse response and in the case of the moving average filter each is equal to 1 /N . As far as implementation is concerned, at the nth sampling instant we multiply N past input samples individually by 1/N and sum the N products. Program average.c, listed in Figure 4.1, uses this approach, even though it is not the most computationally efficient. The value of N defined near the start of the source file determines the number of previous input samples to be averaged. A more rigorous method of assessing the magnitude frequency response of the filter is to use a signal generator and an oscilloscope or spectrum analyzer to measure its gain at different individual frequencies. By using this method, it is straightforward to identify the distinct notches in the magnitude frequency response at 1600 Hz (corresponding to the 52 tone in test file mefsin.wav that is stored in folder average.c) and at 3200 Hz.

The theoretical frequency response of the filter can be found using Matlab by running the following two lines:

```
>> [H W]=freqz([0.2 0.2 0.2 0.2 0.2],1);
>> plot(W*4000/pi,20*log10(abs(H)))
```

PROCEDURE:

1. Design such a filter with MATLAB using the following values: f_S = 8 kHz, f_0 = 2 kHz, and filter order M = 50. Then, using the built-in MATLAB function freqz, or the textbook function dtft, calculate and plot in dB the magnitude response of the filter over the frequency interval $0 \le f \le 4$ kHz.
2. The designed 51-long impulse response coefficient vector h can be exported into a data file, h.dat, in a form that is readable by a C program by the following MATLAB command:

C_header('h.dat', 'h', 'M', h);

where C_headeris a MATLAB function in the directory c:\dsplab\common. A few lines of the resulting data file are shown below:

```
// h.dat  -  FIR impulse response coefficients
   // exported from MATLAB using C_header.m
   // ------------------------------------------
   #define M 50        // filter order
   float h[M+1] = {
      0.001018591635788,
     -0.000000000000000,
     -0.001307170629617,
      0.000000000000000,
      0.002075061044252,
```

```
    -0.001307170629617,
    -0.000000000000000,
     0.001018591635788
    };
// -------------------------------------------
```

The following complete C program called firex.c implements this example on the C6713 processor. The program reads the impulse response vector from the data file h.dat, and defines a linear 51- dimensional buffer array. The FIR filtering operation is based on the function fir().

```
// firex.c - FIR filtering example
// ----------------------------------------------------------------------------
#include "dsplab.h"          // DSK initialization declarations and function prototypes
float fir(int, float *, float *, float);
// float firc(int, float *, float *, float **, float);
// float firc2(int, float *, float *, int *, float);
short xL, xR, yL, yR;
#include "h.dat"
float w[M+1];
int on = 1;
// float *p;
// int q;
// left and right input and output samples from/to codec
// defines M=50, and contains M+1 = 51 filter coefficients
// delay line buffer
// turn filter on or off
// ----------------------------------------------------------------------------

void main() {

inti;

  for (i=0; i<=M; i++) w[i] = 0;
  // p=w;
  // q=0;
initialize();
sampling_rate(8);
audio_source(LINE);

while(1);

// initialize DSK board and codec, define interrupts
// possible sampling rates: 8, 16, 24, 32, 44, 48, 96 kHz
// LINE or MIC for line or microphone input
// keep waiting for interrupt, then jump to isr()
}
// ----------------------------------------------------------------------------
interrupt void isr()
{
float x, y;                // filter input & output
ead_inputs(&xL, &xR);
   if (on) {
      x = (float)(xL);
      y = fir(M,h,w,x);
      // y = firc(M,h,w,&p,x);
```

```
yL = (short)(y);
} else
yL = xL;
write_outputs(yL,yL);
return; }
// work with left input only
      // ---------------------------------------------------------------------------
```

Create and build a project for this program. You will need to add the file fir.c to the project. Using the following MATLAB code (same as in the aliasing example of Lab-2), generate a signal consisting of a 1-kHz segment, followed by a 3-kHz segment, followed by another 1-kHz segment, where all segments have duration of 1 sec:

```
fs = 8000; f1 = 1000; f2 = 3000; f3 = 1000;
L = 8000; n = (0:L-1);
A=1/5; % adjust playback volume

      x1 = A * cos(2*pi*n*f1/fs);
      x2 = A * cos(2*pi*n*f2/fs);
      x3 = A * cos(2*pi*n*f3/fs);
      sound([x1,x2,x3], fs);
```

First, set the parameter on=0 so that the filtering operation is bypassed. Send the above signal into the line input of the DSK and listen to the output. Then, set on=1 to turn the filter on, recompile and run the program, and send the same signal in. The middle 3-kHz segment should not be heard, since it lies in the filter's stopband.

c. Create breakpoints at the read_inputs and write_outputs lines of the isr() function, and start the profile clock. Run the program and record the number of cycles between reading the input samples and writing the computed outputs.

4. Uncommenttheappropriatelinesintheaboveprogramtoimplementthecircularbufferversionusing the function firc(). You will need to add it to your project. Recompile and run your program with the same input.

   Then, repeat part (c) and record the number of computation cycles.

5. Repeat part (d) using the circular-index function firc2().

   In comparing the computational costs of the various implementations, you will notice that for this

example, the linear buffer version is far more efficient, in contrast to what you observed in the case of multiple delays. The reason is that the function pwrap() gets called for each tap of the FIR filter, whereas in the case of multiple delays, it was essentially called once.

The circular buffer implementation of FIR filters can indeed be made far more efficient than the linear buffer case if one used an assembly language version that takes advantage of the built-in circular addressing capability of the C6713 processor.

RESULT:

DISCUSSION:

CONTROL SYSTEM LABORATORY

Experiment No. 1: Using MATLAB for Control Systems.

Objectives: This lab provides an introduction to MATLAB in the first part. The lab alsoprovides tutorial of vector, matrix, array and script writing and programming aspect of MATLAB from control systems view point.

List of Equipment/Software

Following equipment/software is required:

- MATLAB

Category Soft-Experiment

Deliverables

A complete lab report including the following:

- Summarized learning outcomes.
- MATLAB scripts and their results should be reported properly.

Part I: Introduction to MATLAB

Objective: The objective of this exercise will be to introduce students to the concept ofmathematical programming using the software called MATLAB. One shall study how to define variables, matrices etc, see how one can plot results and write simple MATLAB codes.

- Vectors

```
a=[1 2 3];
disp(a)
```

- Matrices:

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

- Matrix operations

i) Product of Matrices by a constant

```
a=[1 2 3;4 5 6;7 8 9];
 b=4*a;
disp(b)
```

ii) Perform addition of two matrices.

```
a=[1 2;3 4];
      b=[5 6;7 8];
        c=a+b;
disp(c)
```

iii) Perform multiplication of two matrices.

```
a=[1 2 3;4 5 6;7 8 9];
   b=[4 5 3;7 6 8;8 7 6];
    c=a*b;
disp(c)
```

iv) Find inverse of square matrix.

```
a=[1 2 3;4 5 6;7 8 9];
     b=inv(a);
disp(c)
```

Part II: Scripts, Functions & Flow Control in MATLAB

Objective: The objective of this session is to introduce students to writing M-file scripts,creating MATLAB Functions and reviewing MATLAB flow control like 'if-elseif-end', 'for loops' and 'while loops'.

Overview:

MATLAB is a powerful programming language as well as an interactive computational environment. Files that contain code in the MATLAB language are called M-files. One create M-files using a text editor, then use them as any other MATLAB function or command. There are two kinds of M-files:

- Scripts, which do not accept input arguments or return output arguments. They operate on data in the workspace. MATLAB provides a full programming language that enables you to write a series of MATLAB statements into a file and then execute

command that MATLAB associates with the program. The file extension of .m makes this a MATLAB M-file.

- Functions, which can accept input arguments and return output arguments. Internal variables are local to the function.

Scripts:

When invoke a script, MATLAB simply executes the commands found in the file. Scripts can operate on existing data in the workspace, or they can create new data on which to operate. Although scripts do not return output arguments, any variables that they create remain in the workspace, to be used in subsequent computations. In addition, scripts can produce graphical output using functions like plot. For example, create a file called 'myprogram.m' that contains these MATLAB commands:

```
% create random numbers and plot
Clc;
    Clear
    R=random (1, 50);
    Plot (r)
```

Typing the statement 'myprogram' at command prompt causes MATLAB to execute the commands, creating fifty random numbers and plots the result in a new window. After execution of the file is complete, the variable 'r' remains in the workspace.

Functions:

Functions are M-files that can accept input arguments and return output arguments. The names of the M-file and of the function should be the same. Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt. An example is provided below:

```
Function f=fact (n)
f=prod(1:n);
```

The first line of a function M-file starts with the keyword 'function'. It gives the function name and order of arguments. In this case, there is one input arguments and one output argument. The next several lines, up to the first blank or executable line, are comment lines that provide the help text. These lines are printed when you type 'help fact'. The first line of the help text is the H1 line, which MATLAB displays when you use the 'lookfor' command or request help on a directory. The rest of the file is the executable MATLAB code defining the function.

The variable n & f introduced in the body of the function as well as the variables on the first line are all local to the function; they are separate from any variables in the MATLAB workspace. This example illustrates one aspect of MATLAB functions that is not ordinarily found in other programming languages—a variable number of arguments. Many M-files work this way. If no output argument is supplied, the result is stored in ans. If the second input argument is not supplied, the function computes a default value.

Flow Control:

Conditional Control – if, else, switch

This section covers those MATLAB functions that provide conditional program control. if, else, and elseif. The if statement evaluates a logical expression and executes a group of statements when the expression is true. The optional elseif and else keywords provide for the execution of alternate groups of statements. An end keyword, which matches the if, terminates the last group of statements.

The groups of statements are delineated by the four keywords-no braces or brackets are involved as given below.

```
if <condition><statements>;

elseif<condition><statements>;
```

end

It is important to understand how relational operators and if statements work with matrices. When you want to check for equality between two variables, you might use
if A == B, ...
This is valid MATLAB code, and does what you expect when A and B are scalars. But when A and B are matrices, A == B does not test if they are equal, it tests where they are equal; the result is another matrix of 0's and 1's showing element-by-element equality. (In fact, if A and B are not the same size, then A == B is an error.)
Example of this type is

```
if A > B
'greater'
elseif A < B
'less'
elseif A == B
'equal'
else
error('Unexpected situation')
end
```

Switch and Case:

The switch statement executes groups of statements based on the value of a variable or expression. The keywords case and otherwise delineate the groups. Only the first matching case is executed. The syntax is as follows

```
switch <condition or expression>
case <condition>
<statements>;
…
case <condition>
…
otherwise
<statements>;
end
```

For, while, break and continue:

This section covers those MATLAB functions that provide control over program loops.

for:

The ‘for’ loop, is used to repeat a group of statements for a fixed, predetermined number of times. A matching ‘end’ delineates the statements. The syntax is as follows:

```
for <index> = <starting number>:<step or increment>:<ending number><statements>;

end
```

while:

The ‘while’ loop, repeats a group of statements indefinite number of times under control of a logical condition. So a while loop executes atleast once before it checks the condition to stop the execution of statements. A matching ‘end’ delineates the statements. The syntax of the ‘while’ loop is as follows:

```
while <condition>
<statements>;
end
```

continue:

The continue statement passes control to the next iteration of the for loop or while loop in which it appears, skipping any remaining statements in the body of the loop. The same holds true for continue statements in nested loops. That is, execution continues at the beginning of the loop in which the continue statement was encountered.

Experiment No. 2: Response of second order systems using MATLAB:

Objective: The objective of this exercise will be to study the performance characteristics ofsecond order systems using MATLAB.

List of Equipment/Software

Following equipment/software is required:

- MATLAB

Category Soft-Experiment

Deliverables

A complete lab report including the following:

- Summarized learning outcomes.
- MATLAB scripts and their results for Exercise 1 & 2 should be reported properly.

Overview Second Order Systems:

Consider the following Mass-Spring system shown in the Figure 1. Where *K* is the spring constant, *B* is the friction coefficient, *x(t)* is the displacement and *F(t)* is the applied force:

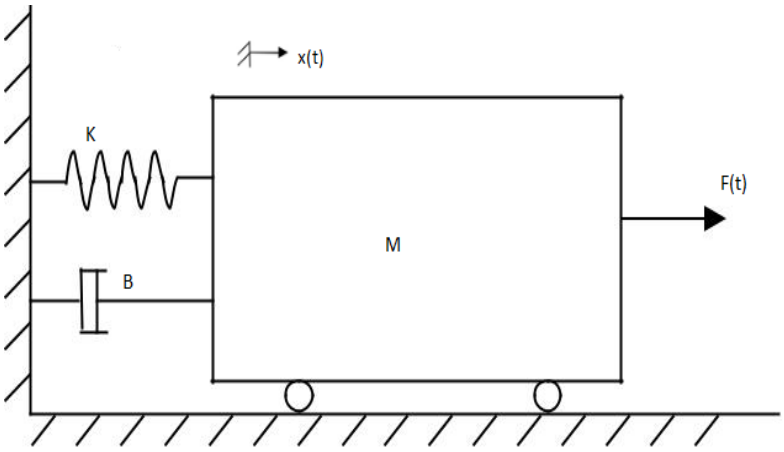


Fig. Mass spring damper system

The differential equation for the above Mass-Spring system can be derived as follows

$$M\frac{d^2y}{dx^2} + B\frac{dy}{dx} + Kx(t) = F(t) \qquad \text{(i)}$$

the transfer function representation of the system is given by

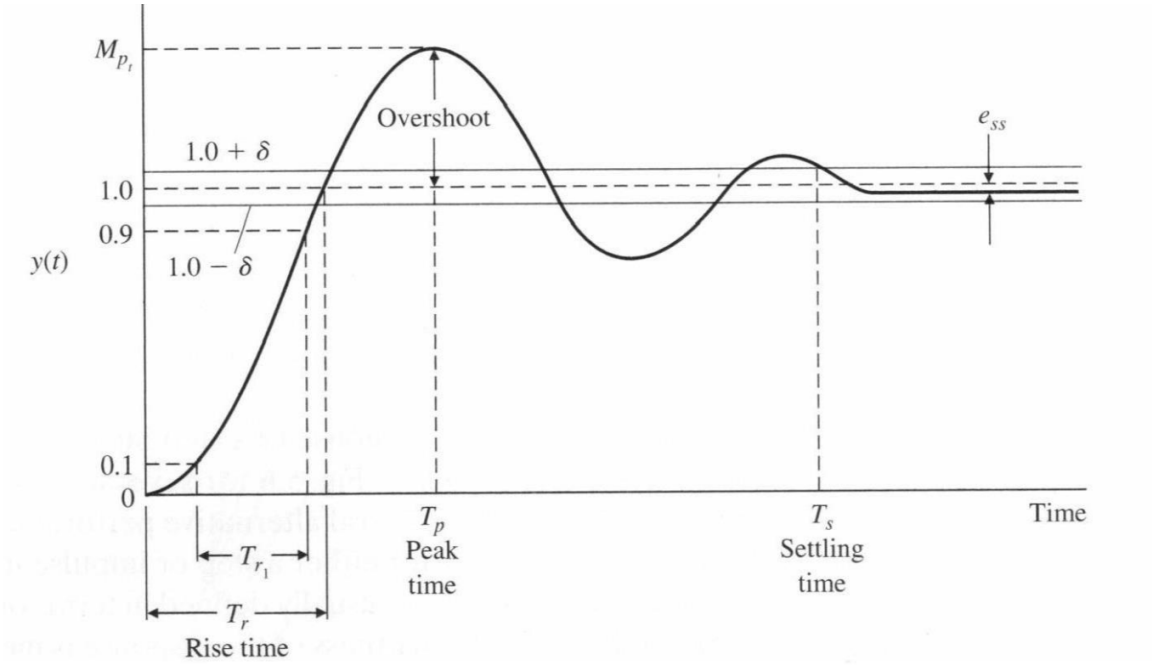
$$TF = \frac{output}{input} = \frac{X(s)}{F(s)} = \frac{1}{Ms^2 + Bs + K}$$

The above system is known as a second order system.

The generalized notation for a second order system described above can be written as

$$Y(s) = \frac{w^2 n}{s^2 + 2\xi wn + w^2 n} R(s) \qquad \text{(ii)}$$

The response of the second order system is shown below.



The performance measures could be described as follows:

Rise Time: The time for a system to respond to a step input and attains a response equal to a percentage of the magnitude of the input. The 0-100% rise time, Tr, measures the time to 100% of the magnitude of the input. Alternatively, Tr_1, measures the time from 10% to 90% of the response to the step input.

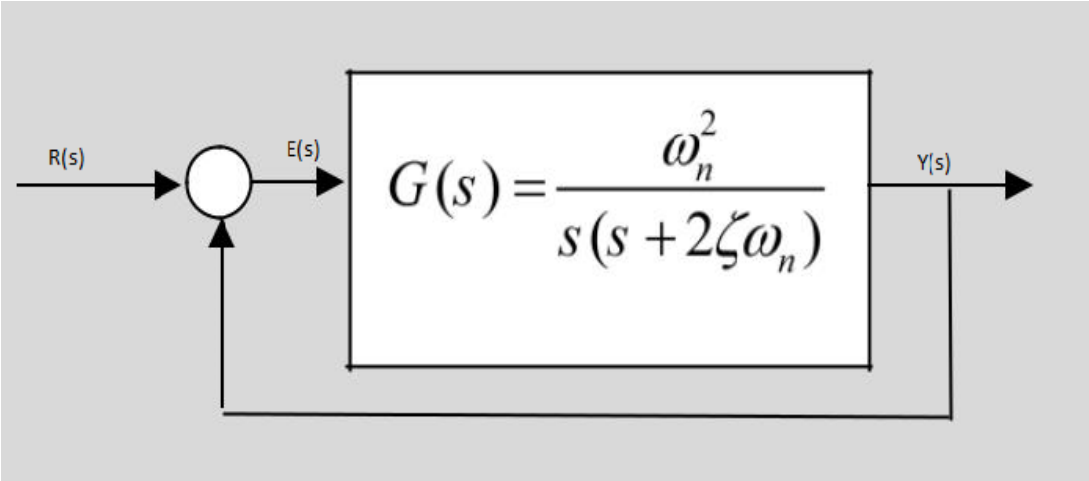
Peak Time: The time for a system to respond to a step input and rise to peak response.
Overshoot: The amount by which the system output response proceeds beyond the desiredresponse. It is calculated as

$$PO = \frac{M_p - fv}{fv} X100\%$$

where *MP*is the peak value of the time response, and *fv* is the final value of the response.
Settling Time: The time required for the system's output to settle within a certain percentage of the input amplitude (which is usually taken as 2%). Then, settling time, T_s, is calculated as

$$Ts = \frac{4}{\xi wn}$$

Ex. Effect of damping ratio ζ on performance measures. For a single-loop secondorder feedback system given below



Find the step response of the system for values of ωn = 1 and ζ = 0.1, 0.4, 0.7, 1.0 and 2.0. Plot all the results in the same figure window and fill the following table.

ζ	Rise time	Peak time	Overshoot	Settling time	Steady state value

Experiment No. 3 Modelling of Physical Systems using SIMULINK.

Objectives: The objective of this exercise is to use graphical user interface diagrams tomodel the physical systems for the purpose of design and analysis of control systems. We will learn how MATLAB/SIMULINK helps in solving such models.

List of Equipment/Software

Following equipment/software is required:

- MATLAB/SIMULINK

Category Soft-Experiment

Deliverables

A complete lab report including the following

- Summarized learning outcomes.
- MATLAB scripts, SIMULINK diagrams and their results for all the assignments and exercises should be properly reported.

Overview:

This lab introduces powerful graphical user interface (GUI), **Simulink** of Matlab. This software is used for solving the modeling equations and obtaining the response of a system to different inputs. Both linear and nonlinear differential equations can be solved numerically with high precision and speed, allowing system responses to be calculated and displayed for many input functions. To provide an interface between a system's modeling equations and the digital computer, block diagrams drawn from the system's differential equations are used. A block diagram is an interconnection of blocks representing basic mathematical operations in such a way that the overall diagram is equivalent to the system's mathematical model. The lines interconnecting the blocks represent the variables describing the system behavior. These may be inputs, outputs, state variables, or other related variables. The blocks represent operations or functions that use one or more of these variables to calculate other variables. Block diagrams can represent modeling equations in both input-output and state variable form.

We use MATLAB with its companion package **Simulink**, which provides a graphical user interface (GUI) for building system models and executing the simulation. These models are constructed by drawing block diagrams representing the algebraic and differential equations that describe the system behavior. The operations that we generally use in block diagrams are summation, gain, and integration. Other blocks, including nonlinear elements such as multiplication, square root, exponential, logarithmic, and other functions, are available. Provisions are also included for supplying input functions, using a signal generator block, constants etc and for displaying results, using a scope block.

An important feature of a numerical simulation is the ease with which parameters can be varied and the results observed directly. MATLAB is used in a supporting role to initialize parameter values and to produce plots of the system response. Also MATLAB is used for multiple runs for varying system parameters. Only a small subset of the functions of MATLAB will be considered during these labs.

SIMULINK

Simulink provides access to an extensive set of blocks that accomplish a wide range of functions useful for the simulation and analysis of dynamic systems. The blocks are grouped into libraries, by general classes of functions.

- Mathematical functions such as summers and gains are in the Math library.
- Integrators are in the Continuous library.
- Constants, common input functions, and clock can all be found in the Sources library.
- Scope, To Workspace blocks can be found in the Sinks library.

Simulink is a graphical interface that allows the user to create programs that are actually run in MATLAB. When these programs run, they create arrays of the variables defined in Simulink that can be made available to MATLAB for analysis and/or plotting. The variables to be used in MATLAB must be identified by Simulink using a "To Workspace" block, which is found in the Sinks library. (When using this block, open its dialog box and specify that the save format should be Matrix, rather than the default, which is called Structure.) The Sinks library also contains a Scope, which allows variables to be displayed as the simulated system responds to an input. This is most useful when studying responses to repetitive inputs.

Simulink uses blocks to write a program. Blocks are arranged in various libraries according to their functions. Properties of the blocks and the values can be changed in the associated dialog boxes.

GENERAL INSTRUCTIONS FOR WRITING A SIMULINK PROGRAM

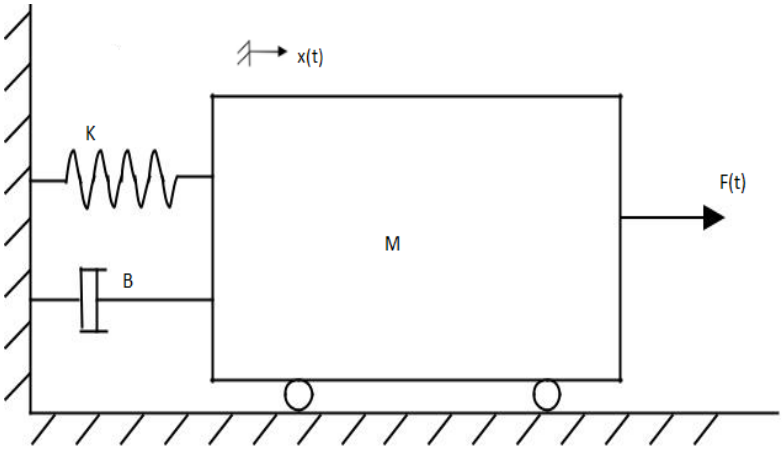
To create a simulation in Simulink, follow the steps:

- Start MATLAB.
- Start Simulink.
- Open the libraries that contain the blocks you will need. These usually will include the Sources, Sinks, Math and Continuous libraries, and possibly others.
- Open a new Simulink window.
- Drag the needed blocks from their library folders to that window. The Math library, for example, contains the Gain and Sum blocks.
- Arrange these blocks in an orderly way corresponding to the equations to be solved.
- Interconnect the blocks by dragging the cursor from the output of one block to the input of another block. Interconnecting branches can be made by right-clicking on an existing branch.
- Double-click on any block having parameters that must be established, and set these parameters. For example, the gain of all Gain blocks must be set. The number and signs of the inputs to a Sum block must be established. The parameters of any source blocks should also be set in this way.
- It is necessary to specify a stop time for the solution. This is done by clicking on the Simulation > Parameters entry on the Simulink toolbar.

At the Simulation > Parameters entry, several parameters can be selected in this dialog box, but the default values of all of them should be adequate for almost all of the exercises. If the response before time zero is needed, it can be obtained by setting the Start time to a negative value. It may be necessary in some problems to reduce the maximum integration step size used by the numerical algorithm. If the plots of the results of a simulation appear "choppy" or composed of straight-line segments when they should be smooth, reducing the max step size permitted can solve this problem.

Mass-Spring System Model

Consider the Mass-Spring system used in the previous exercise as shown in the figure. Where K is the spring constant, B is the viscous friction coefficient, x(t) is the displacement and F(t) is the applied force:



The differential equation for the above Mass-Spring system can then be written as follows

$$M\frac{d^2x}{dt^2}+B\frac{dx}{dt}+Kx(t)=F(t) \quad (1)$$

Exercise 1: Modelling of a second order system

Construct a Simulink diagram to calculate the response of the Mass-Spring system. The input force increases from 0 to 8 N at t = 1 s. The parameter values are M = 2 kg, K= 16 N/m, and B =4 N.s/m.

Steps:

- Draw the free body diagram.
- Write the modelling equation from the free body diagram
- Solve the equations for the highest derivative of the output.
- Draw a block diagram to represent this equation.
- Draw the corresponding Simulink diagram.
- Use Step block to provide the input F(t).
- In the Step block, set the initial and final values and the time at which the step occurs.
- Use the "To Workspace" blocks for t, F(t), x, and v in order to allow MATLAB to plot the desired responses. Set the save format to array in block parameters.

- Select the duration of the simulation to be 10 seconds from the Simulation > Parameters entry on the toolbar

Exercise 2: Simulation with system parameter variation

The effect of changing B is to alter the amount of overshoot or undershoot. These are related to a term called the damping ratio. Simulate and compare the results of the variations in B in exercise 1. Take values of B = 4, 8, 12, 25 N-s/m.

Steps:

Perform the following steps. Use the same input force as in Exercise 1.

- Begin the simulation with B = 4 N-s/m, but with the input applied at t = 0
- Plot the result.
- Rerun it with B = 8 N.s/m.
- Hold the first plot active, by the command hold on
- Reissue the plot command plot(t,x), the second plot will superimpose on the first.
- Repeat for B = 12 N-s/m and for B = 25 N-s/m
- Release the plot by the command hold off
- Show your result.

Running SIMULINK from MATLAB command prompt

If a complex plot is desired, in which several runs are needed with different parameters, thiscan using the command called "sim". "sim" command will run the Simulink model file fromthe Matlab command prompt. For multiple runs with several plot it can be accomplished by executing ex1_model (to load parameters) followed by given M-file. Entering the command ex1_plots in the command window results in multiple runs with varying values if B and will plot the results.

Experiment No.4: Effect of Feedback on disturbance & Control System Design.

Objective: The objective of this exercise will be to study the effect of feedback on theresponse of the system to step input and step disturbance taking the practical example of English Channel boring machine and design a control system taking in account performance measurement.

List of Equipment/Software

Following equipment/software is required:

- MATLAB

Category Soft - Experiment

Deliverables

A complete lab report including the following:

- Summarized learning outcomes.
- The Simulink model.
- MATLAB scripts and results for Exercise 1.

Overview:

The construction of the tunnel under the English Channel from France to the Great Britain began in December 1987. The first connection of the boring tunnels from each country was achieved in November 1990. The tunnel is 23.5 miles long and bored 200 feet below sea level. Costing $14 billion, it was completed in 1992 making it possible for a train to travel from London to Paris in three hours.

The machine operated from both ends of the channel, bored towards the middle. To link up accurately in the middle of the channel, a laser guidance system kept the machines precisely aligned. A model of the boring machine control is shown in the figure, where Y(s) is the actual angle of direction of travel of the boring machine and R(s) is the desired angle. The effect of load on the machine is represented by the disturbance, $T_d(s)$.

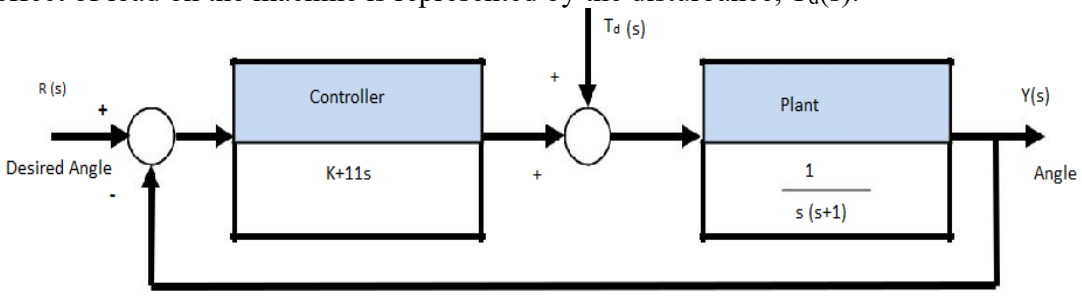


Fig. A block diagram model of a boring machine control system

A model of the boring machine control is shown in the figure, where Y (s) is the actual angle of direction of travel of the boring machine and R(s) is the desired angle. The effect of load on the machine is represented by the disturbance $T_d(s)$.

Exercise 1.

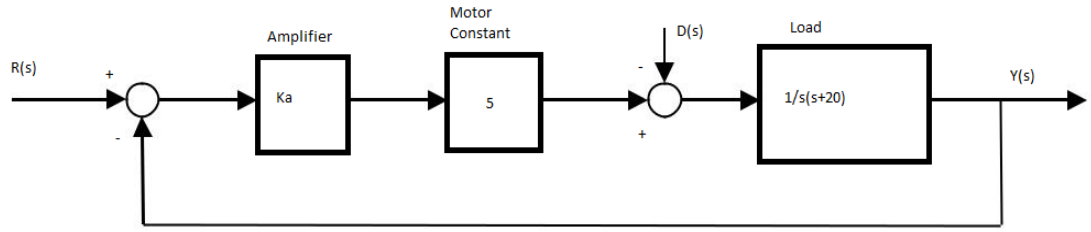
Perform the following: a) Get the transfer function from R (s) to Y (s)

b) Get the transfer function from D (s) to Y (s)

c) Generate system response using Matlab; K= 10, 20, 50, 100; due to a unit step input–r(t)

d) Generate system response using Matlab; K= 10, 20, 50, 100; due to a unit step disturbance–d(t)

e) For each case find the percentage overshoot (% O. S.), rise time, settling time, steady of y(t)
f) Compare the results of two cases.
g) Investigate the effect of changing the controller gain on the influence of the disturbance on the system output.



Exercise 2.

Design a second order feedback system based on performances.

For the motor system given above, we need to design feedback such that the overshoot is limited and there is less oscillatory nature in the response based on the specifications provided in the table. Assume no disturbance (D(s)=0).

Performance Measure	Desired value
Percentage overshoot	Less than 8%
Settling time	Less than 400 ms

Use MATLAB, to find the system performance for different values of K_a and find which value of the gain K_a satisfies the design condition specified. Use the following table.

Ka	20	30	50	60	80
Percentage overshoot					
Settling time					

Experiment No.5: Introduction to PID controller

Objective: Study the three term (PID) controller and its effects on the feedback loop response. Investigate the characteristics of the each of proportional (P), the integral (I), and the derivative (D) controls, and how to use them to obtain a desired response.

List of Equipment/Software

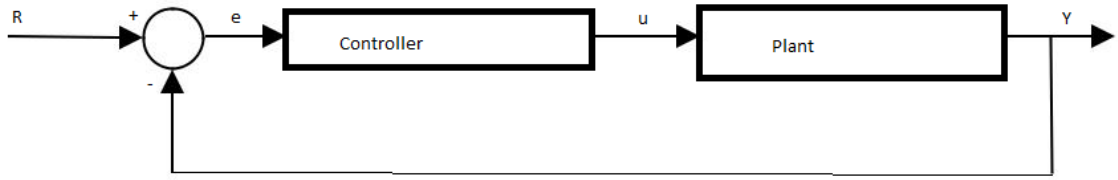
- MATLAB

Category Soft - Experiment

Deliverables A complete lab report including the following:

- Summarized learning outcomes.
- Controller design and parameters for each of the given exercises.

Introduction: Consider the following unity feedback system:



Plant: A system to be controlled.

Controller: Provides excitation for the plant; Designed to control the overall system behavior. The three-term controller: The transfer function of the PID controller looks like the following:

$$K_P + \frac{K_I}{s} + K_D s = \frac{K_D s^2 + K_P s + K_I}{s}$$

K_P=Proportional gain
K_I=Integral gain
K_D=Derivative gain

First, let's take a look at how the PID controller works in a closed-loop system using the schematic shown above. The variable (e) represents the tracking error, the difference between the desired input value (R) and the actual output (Y). This error signal (e) will be sent to the PID controller, and the controller computes both the derivative and the integral of this error signal. The signal (u) just past the controller is now equal to the proportional gain (KP) times the magnitude of the error plus the integral gain (KI) times the integral of the error plus the derivative gain (KD) times the derivative of the error.

Example Problem:

Suppose we have a simple mass, spring, and damper problem.

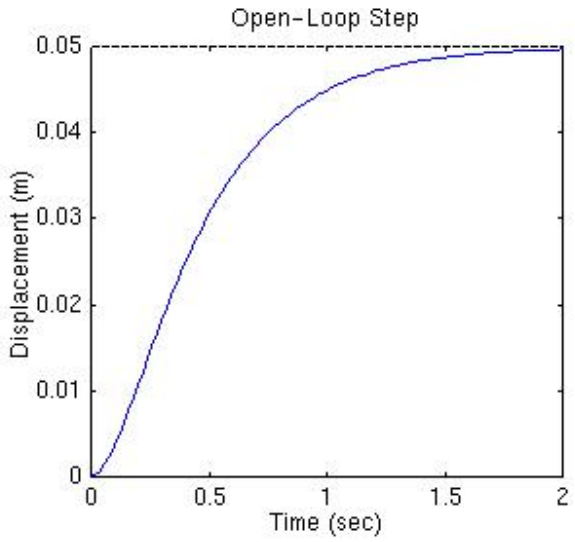
The TF of a mass, spring and damper system is given by,

$$\frac{X(s)}{F(s)} = \frac{1}{s^2 + 10s + 20}$$

The goal of this problem is to show you how each of K_p, K_i and K_d contributes to obtain

- Fast rise time
- Minimum overshoot
- No steady-state error

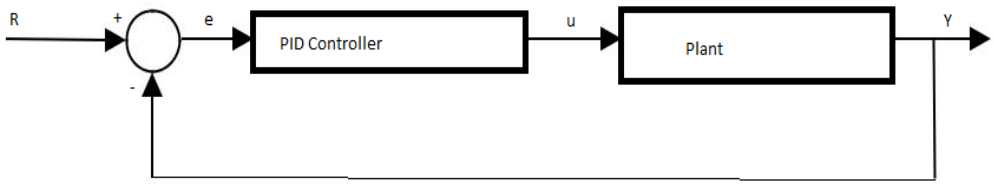
Open-loop step response: Let's first view the open-loop step response.



The DC gain of the plant transfer function is 1/20, so 0.05 is the final value of the output to a unit step input. This corresponds to the steady-state error of 0.95, quite large indeed. Furthermore, the rise time is about one second, and the settling time is about 1.5 seconds. Let's design a controller that will reduce the rise time, reduce the settling time, and eliminates the steady-state error.

Proportional-Integral-Derivative control:

Now, let's take a look at a PID controller. The closed-loop transfer function of the given system with a PID controller is:



PID Controller block is represented by the function,

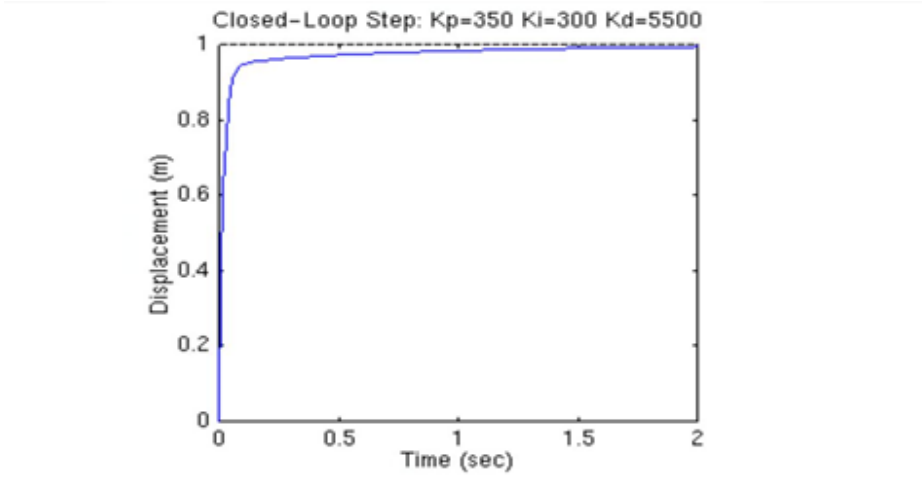
$$K_P + \frac{K_I}{s} + K_D s$$

The plant is represented by, $\frac{1}{s^2+10s+20}$

Now the closed loop TF with a PID controller is :

$$\frac{X(s)}{F(s)} = \frac{K_D s^2 + K_P s + K_I}{s^3 + (10 + K_D)s + 20K_I}$$

After several trial and error runs, the gains Kp=350, Ki=300, and Kd=50 provided the desired response. To confirm, enter the following commands to an m-file and run it in the command window. One should get the following step response.



Exercise.

Consider a process given below to be controlled by a PID controller,

$$G_p = \frac{400}{s(s+48.5)}$$

a) Obtain the unit step response of Gp(s).

b) Try PI controllers with (Kp=2, 10, 100), and Ki=Kp/10. Investigate the unit step response in each case, compare the results and comment.

c) Let Kp=100, Ki=10, and add a derivative term with (Kd=0.1, 0.9, 2). Investigate the unit step response in each case, compare the results and comment.
Based on your results in parts b) and c) above what do you conclude as a suitable PID controller for this process and give your justification.

Experiment No.6. Use MATLAB to check the Controllability and Observability of the system described by the following state space equation:

$$\dot{x} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & -4 & -3 \end{pmatrix} x + \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 1 \end{pmatrix} u, y = \begin{pmatrix} 0 & 1 & -1 \\ 1 & 2 & 1 \end{pmatrix} x$$

Objective: Study the Controllability and Observabilty of different system.

List of Equipment/Software

- MATLAB

Category Soft - Experiment

Deliverables A complete lab report including the following:

- Summarized learning outcomes.

Controllability:

Complete state controllability (or simply controllability if no other context is given) describes the ability of an external input to move the internal state of a system from any initial state to any other final state in a finite time interval.

A state x_0 is **controllable** at time t_0 if for some finite time t_1 there exists an input u(t) that transfers the state x(t) from x_0 to the origin at time t_1.

A system is called **controllable** at time t_0 if every state x_0 in the state-space is controllable.

Kalman's Test for checking Controllability is as follows,
A linear time invariant continuous system described by the state equation
$\dot{x} = Ax + BU$ (i)
$Y = Cx$ (ii)
Is completely controllable if and only if the rank of the controllability matrix is defined as
Q= [B: AB: A^2B:----------$A^{n-1}B$] is equal to rank 'n'.

Observability:

The state-variables of a system might not be able to be measured for any of the following reasons:

1. The location of the particular state variable might not be physically accessible (a capacitor or a spring, for instance).
2. There are no appropriate instruments to measure the state variable, or the state-variable might be measured in units for which there does not exist any measurement device.
3. The state-variable is a derived "dummy" variable that has no physical meaning.

If things cannot be directly observed, for any of the reasons above, it can be necessary to calculate or **estimate** the values of the internal state variables, using only the input/output relation of the system, and the output history of the system from the starting time. In other words, we must ask whether or not it is possible to determine what the inside of the system (the internal system states) is like, by only observing the outside performance of the system (input and output)? We can provide the following formal definition of mathematical **observability:**

A system with an initial state, $x(t_0)$ is **observable** if and only if the value of the initial state can be determined from the system output y(t) that has been observed through the time interval $t_0<t<t_f$. If the initial state cannot be so determined, the system is **unobservable**.

Complete Observability

A system is said to be **completely observable** if all the possible initial states of the system can be observed. Systems that fail these criteria are said to be **unobservable**.

Observability Test:

The necessary and sufficient condition for the system to be completely observable is that the nxnobservabilty matrix

V=[C; CA; ----------CA^{n-1}] has rank equal to 'n'.