Ahsanullah University of Science and Technology

Department of Electrical and Electronic Engineering

AUST/EEE

LABORATORY MANUAL

FOR

ELECTRICAL AND ELECTRONIC SESSIONAL COURSES

Student Name :

Student ID :

Course no : EEE 3218

Course Title : Digital Signal Processing Lab

For the students of

Department of Electrical and Electronic Engineering

3rd Year, 2nd Semester

An Overview of DSP Lab

SIGNALS, WAVES, AND DIGITAL PROCESSING

Two of the human senses, sight and hearing, work via the detection of waves. Useful information from both light and sound is gained by detection of certain characteristics of these waves, such as frequency and amplitude. Modern telecommunication depends on transducing sound or light into electrical quantities such as voltage, and then processing the voltage in many different ways to enable the information to be reliably stored or conveyed to a distant place and then regenerated to imitate (i.e., reconstruct) the original sound or light phenomenon.

For example, in the case of sound, a microphone detects rapid pressure variations in air and converts those variations to an output voltage which varies in a manner proportional to the variation of pressure on the microphone's diaphragm. The varying voltage can be used to cut a corresponding wave into a wax disc, to record corresponding wave-like variations in magnetism onto a ferromagnetic wire or tape, to vary the opacity of a linear track along the edge of a celluloid film (i.e., the sound-track of a motion picture film) or perhaps to modulate a carrier wave for radio transmission.

In recent decades, signal processing and storage systems have been developed that use discrete samples of a signal rather than the entire continuous time domain (or **analog**) signal. Several useful definitions are as follows:

AUST/EEE

- A **sample** is the amplitude of an analog signal at an instant in time.

- A system that processes a signal in sampled form (i.e., a sequence of samples) is known as a **Discrete Time Signal Processing System**.

- In a **Digital Signal Processing** system, the samples are converted to numerical values, and the values (numbers) stored (usually in binary form), transmitted, or otherwise processed.

The difference between conventional analog systems and digital systems is illustrated in Fig. 1. At (a), a conventional analog system is shown, in which the signal from a microphone is sent directly to an analog recording device, such as a tape recorder, recorded at a certain tape speed, and then played back at the same speed some time later to reproduce the original sound. At (b), samples of the microphone signal are obtained by an **Analog-to-Digital Converter (ADC)**, which converts instantaneous voltages of the microphone signal to corresponding numerical values, which are stored in a digital memory, and can later be sent to a **Digital-to-Analog Converter (DAC)** to reconstruct the original sound.

In addition to recording and reproducing analog signals, most other kinds of processing which might be performed on an analog signal can also be performed on a sampled version of the signal by using numerical algorithms. These can be categorized into two broad types of processing, time domain and frequency domain, which are discussed in more detail below.

ADVANTAGES OF DIGITAL SIGNAL PROCESSING

The reduction of continuous signals to sequences of numerical values (samples) that can be used to process and/or reconstruct the original signal, provides a number of benefits that cannot be achieved with continuous or analog signal processing. The following are some of the benefits of digital processing:

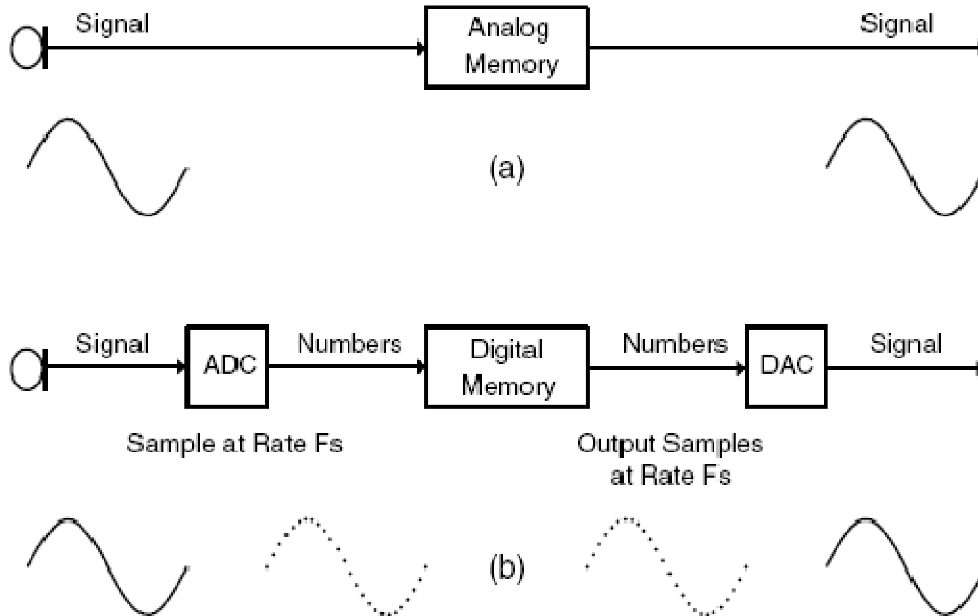


Figure 1: (a) Conventional analog recording and playback system; (b) A digital recording and playback system.

1. Analog hardware, such as amplifiers, filters, comparators, etc., is very susceptible to noise and deterioration through aging. Digital hardware works with only two signal levels rather than an infinite number, and hence has a high signal to noise ratio. As a result, there is little if any gradual deterioration of performance with age (although as with all things, digital hardware can suddenly and totally fail), and copies of signal files are generally perfect, absent component failure, media degeneration or damage, etc. This is not true with analog hardware and recording techniques, in which every copy introduces significant amounts of additional noise and distortion.

2. Analog hardware, for the most part, must be built for each processing function needed. With digital processing, each additional function needed can be implemented with a software module, using the same piece of hardware, a digital computer. The computing power available to the average person has increased enormously in recent years, as evidenced by the incredible variety of inexpensive, high quality devices and techniques available. Hundreds of millions or billions of operations per second can be performed on a signal using digital hardware at reasonable expense; no reasonably-priced alternative exists using analog hardware and processing.

3. Analog signal storage is typically redundant, since wave-related signals (audio, video, etc.) are themselves typically redundant. For example, by taking into account this redundancy as well as the physiological limitations of human hearing, storage needs for audio signals can be reduced up to about 95%, using digitally-based compression techniques, such as MP3, AC3, AAC, etc. Digital processing makes possible highly efficient security and error-correction coding. Using digital coding, it is possible, for example, for many signals to be transmitted at very low power and to share the same bandwidth. Modern cell phone techniques, such as CDMA (Code Division Multiple Access) rely heavily on advanced, digitally-based signal processing techniques to efficiently achieve both high quality and high security.

DSP NOMENCLATURE AND TOPICS

Figure 1.2 shows a broad overview of digital signal processing. Analog signals enter an ADC from the left, and samples exit the ADC from the right, and may be 1) processed strictly in the discrete time domain (in which samples represent the original signal at instants in time) or they may be 2) converted to a frequency domain representation (in which samples represent amplitudes of particular frequency components of the original signal) by a time-to-frequency transform, processed in the frequency domain, then converted back to the discrete time domain by a frequency-to-time transform. Discrete time domain samples are converted back to the continuous time domain by the DAC.

Note that a particular signal processing system might use only time domain processing, only frequency domain processing, or both time and frequency domain processing, so either or both of the signal processing paths shown in Fig. 2 may be taken in any given system.

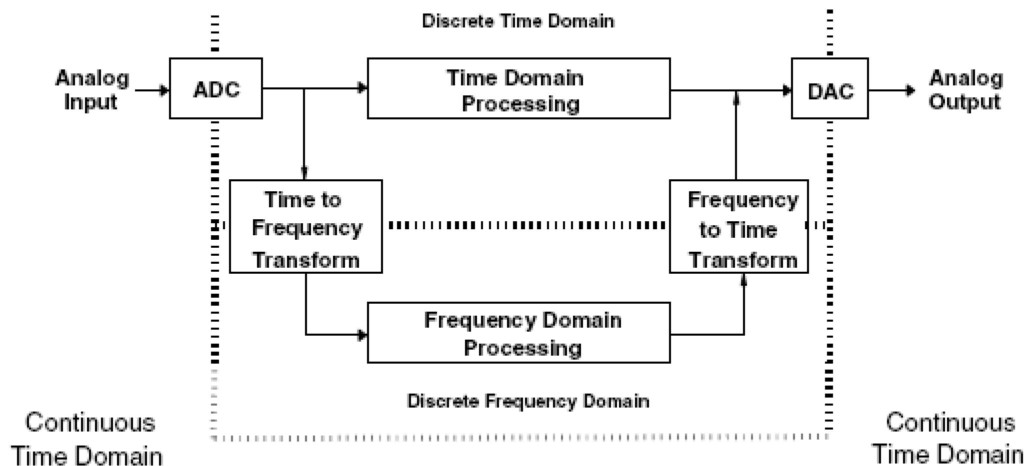


Figure 2: A broad, conceptual overview of digital signal processing.

TIME DOMAIN PROCESSING

Filtering, in general, whether it is done in the continuous domain or discrete domain, is one of the fundamental signal processing techniques; it can be used to separate signals by selecting or rejecting certain frequencies, enhance signals (such as with audio equalization, etc.), alter the phase characteristic, and so forth. Hence a major portion of the study of digital signal processing is devoted to digital filtering. Filtering in the continuous domain is performed using

combinations of components such as inductors, capacitors, resistors, and in some cases active elements such as op amps, transistors, etc. Filtering in the discrete or digital domain is performed by mathematically manipulating or processing a sequence of samples of the signal using a discrete time processing system, which typically consists of registers or memory elements, delay elements, multipliers, and adders. Each of the preceding elements may be implemented as distinct pieces of hardware in an efficient arrangement designed to function for a particular purpose (often referred to as a **Pipeline Processor**), or, the equivalent functions of all elements may be implemented on a general purpose computer by specifically designed software.

FREQUENCY TRANSFORMS

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

FREQUENCY DOMAIN PROCESSING

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

AUST/EEE

REFERENCE BOOKS

Digital Signal Processing Using Matlab V4 - Ingle and Proakis
Digital Signal Processing - Computer Based Approach - Sanjit K. Mitra
Digital Signal Processing using MATLAB - André Quinquis
Digital Filter Design Using Matlab - Timothy J. Schlichter

Discrete Signals and Concepts

DISCRETE SEQUENCE NOTATION

Digital Signal Processing must necessarily begin with a signal, and most signals, such as sound, images, etc., originate as continuous-valued (or analog) signals, and must be converted into a sequence of samples to be processed using digital techniques. Figure 3 depicts a continuous-domain sine wave, with eight samples marked, sequentially obtained every 0.125 second. The signal values input to the ADC at sample times 0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, etc., are 0, 0.707, 1, 0.707, 0, -0.707, -1, etc.

The samples within a given sample sequence are normally indexed by the numbers 0, 1, 2, etc. which represent multiples of the sample period *T*. For example, in Fig. 3, we note that the sample period is 0.125 second, and the actual sampling times are therefore 0 sec., 0.125 sec., 0.25 sec., etc. The continuous sine function shown has the value

$$f(t) = \sin(2\pi f t)$$

where t is time, f is frequency, and in this particular case, $f = 1$ Hz. Sampling occurs at times nT where n = 0, 1, 2,...and $T = 0.125$ second. The sample values of the sequence would therefore be $\sin(0)$, $\sin(2\pi(T))$, $\sin(2\pi(2T))$, $\sin(2\pi(3T))$, etc., and we would then say that $s[0] = 0$, $s[1] = 0.707$, $s[2] = 1.0$, $s[3] = 0.707$, etc. where $s[n]$ denotes the n-th sequence value, the amplitude of which is equal to the underlying continuous function at time nT (note that brackets surrounding a function argument mean that the argument can only assume discrete values, while parentheses surrounding an argument indicate that the argument's domain is continuous). We can also say that

$$s[n] = \sin[2\pi nT]$$

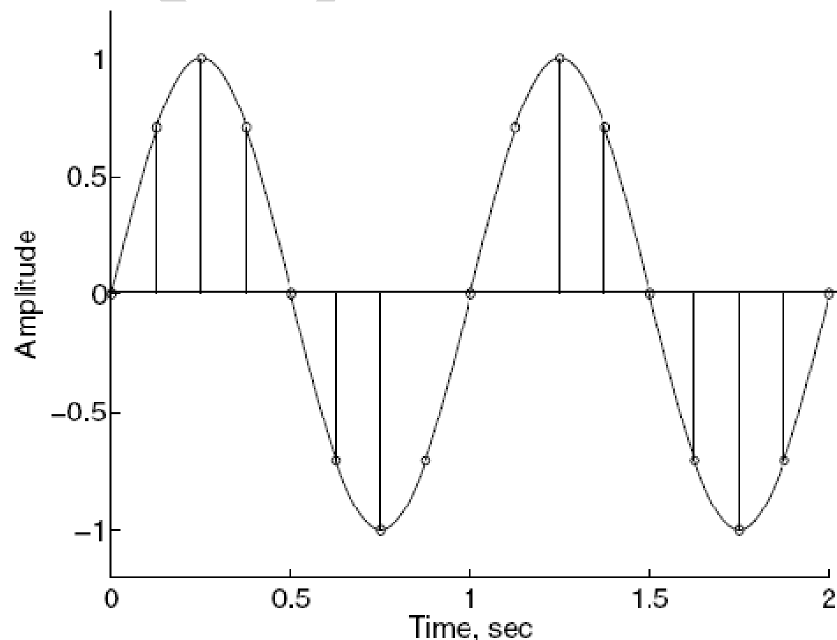


Figure 3: An analog or continuous-domain sine wave, with eight samples per second marked.

This sequence of values, the samples of the sine wave, can be used to completely reconstruct the original continuous domain sine wave using a DAC. There are, of course, a number of conditions to ensure that this is possible, and they are taken up in detail in the next chapter. To compute and plot the sample values of a 2-Hz sine wave sampled every 0.05 second on the time interval 0 to 1.1 second, make the following MathScript call:

```
t = [0:0.05:1.1]; figure; stem(t,sin(2*pi*2*t))
```

where the *t* vector contains every sample time *nT* with *T* = 0.05. Alternatively, we might write

```
T = 0.05; n = 0:1:22; figure; stem(n*T,sin(2*pi*2*n*T))
```

both of which result in Fig. 4.

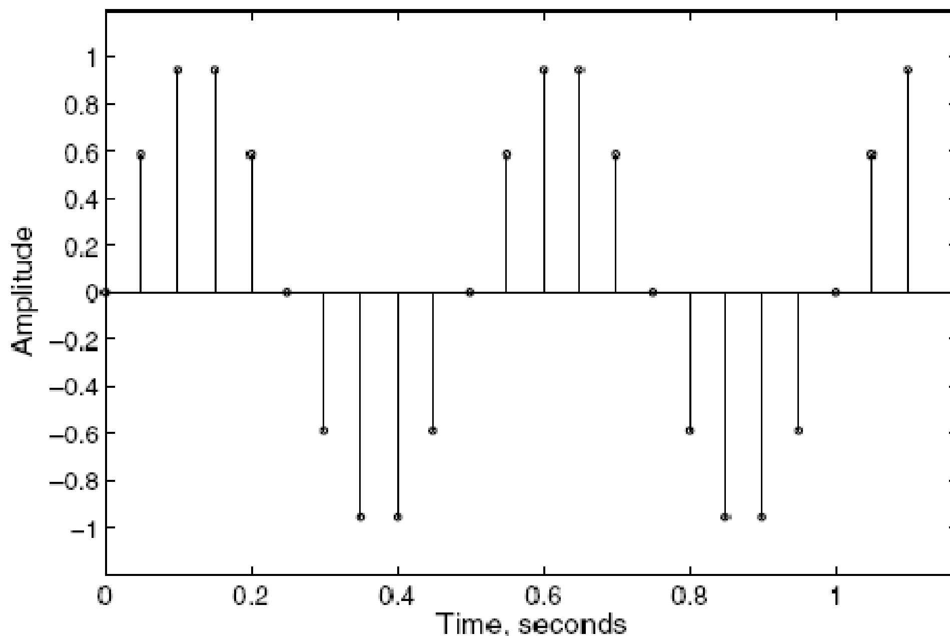


Figure 4: A plot of the samples of a sine wave having frequency 2 Hz, sampled every 0.05 second up to time 1.1 second.

USEFUL SIGNALS, SEQUENCES, AND CONCEPTS

SINE AND COSINE

We saw above that a sine wave of frequency *f* periodically sampled at the time period *T* has the values

$$s[n] = \sin[2\pi f\, nT]$$

Once we have a sampled sine wave, we can mathematically express it without reference to the sampling period by defining the sequence length as *N*. We would then have, in general,

$$s[n] = \sin[2\pi nk/N]$$

where n is the sample index, which runs from 0 to $N-1$, and k is the number of cycles over the sequence length N. For the sample sequence marked in Fig. 2.1, we would have

$$s[n] = \sin[2\pi n 2/16]$$

where we have noted that there are two full cycles of the sine over 16 samples (the 17th sample is the start of the third cycle). The correctness of this formula can be verified by noting that for the 17th sample, $n = 16$, and $s[16] = 0$, as shown. Picking another sample, for $n = 2$, we get $s[2] = \sin[2\pi(2)2/16] = \sin[\pi/2] = 1$, as shown.
A phase angle is sometimes needed, and is denoted θ by in the following expression:

$$s[n] = \sin[2\pi nk/N + \theta]$$

Note that if $\theta = \pi/2$, then

$$s[n] = \cos[2\pi nk/N]$$

We can illustrate this by generating and displaying a sine wave having three cycles over 18 samples, then the same sine wave, but with a phase angle of $\pi/2$ radians, and finally a cosine wave having three cycles over 18 samples and a zero phase angle. A suitable MathScript call, which results in Fig. 5, is

```
n = 0:1:17; y1 = sin(2*pi*n/18*3); subplot(311); stem(n,y1);
y2 = sin(2*pi*n/18*3 +pi/2); subplot(312); stem(n,y2);
y3 = cos(2*pi*n/18*3); subplot(313); stem(n,y3)
```

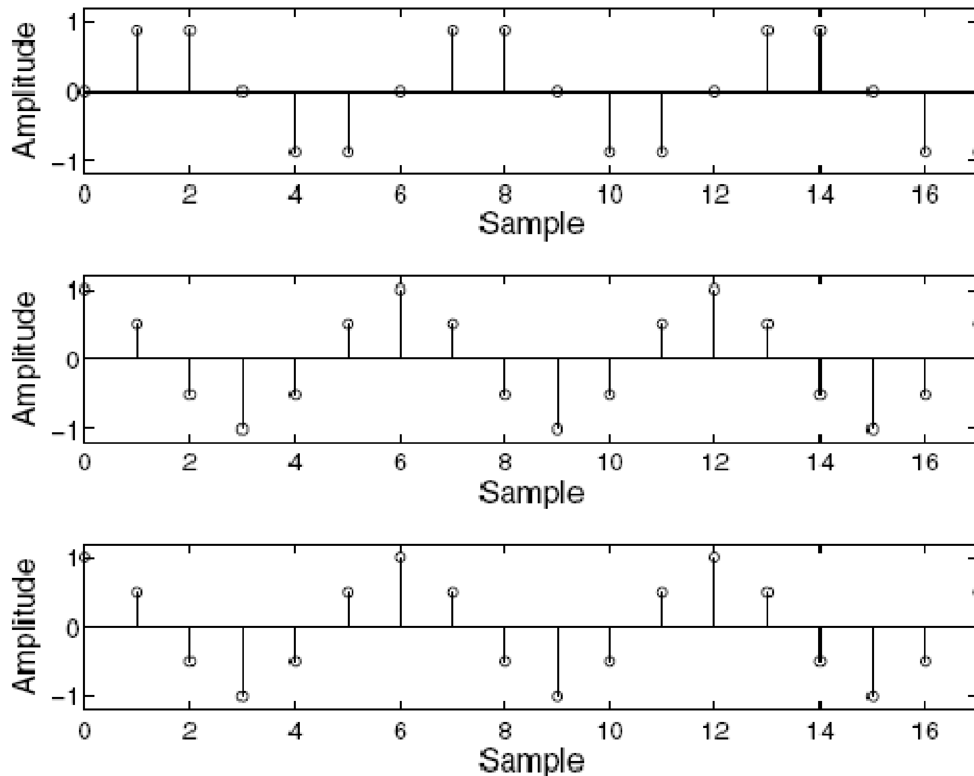


Figure 5: (a) Three cycles of a sine wave over 18 samples, with phase angle 0 radians; (b) Same as (a), with a phase angle of $\pi/2$ radians; (c) Three cycles of a cosine wave over 18 samples, with a phase angle of 0 radians.

Problem: 1

Write a Matlab script to plot the signal $x(n) = cos\omega_o n$ for $\omega_o = 0, \pi/2, \pi/8, \pi/4, \pi/2, \pi$. The output should look like the Figure 6.

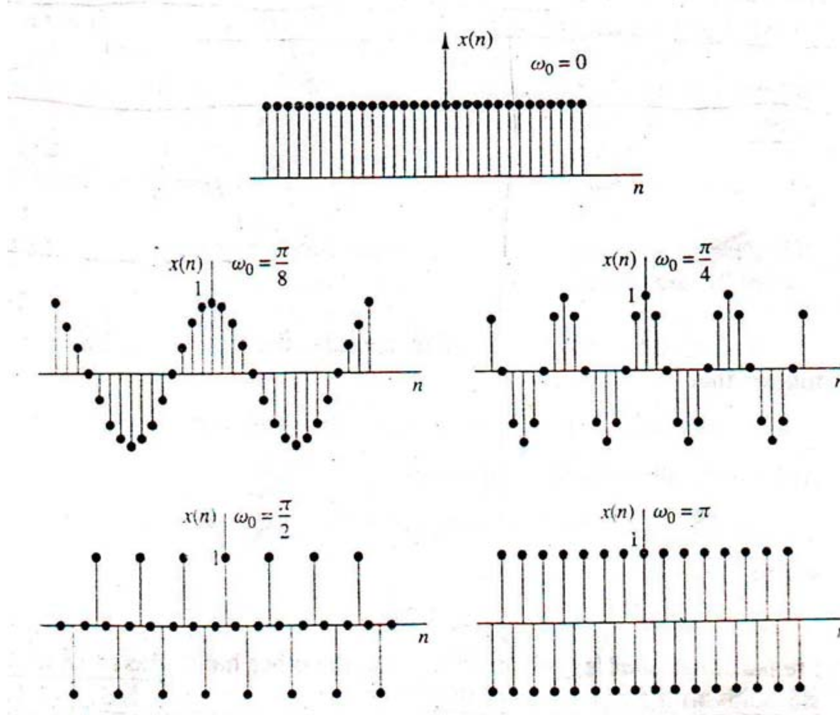


Figure 6: Signal $x(n) = cos\omega_o n$ for various values of frequency ω_o.

Problem: 2

Write a Matlab script to plot the signal $x(n) = Ca^n$ for (i) C and a are real for $a > 1$, $0 < a < 1$ $a = 1$, $a < -1$, $-1 < a < 0$ and $a = -1$ (ii) C complex, a Complex with Unity Magnitude [$C = Ae^{j\phi}$ and $a = e^{j\Omega o}$] (iii) Both C complex, a Complex [$C = Ae^{j\phi}$ and $a = e^{\Sigma o + j\Omega o}$]. The output should look like the Figure 7.

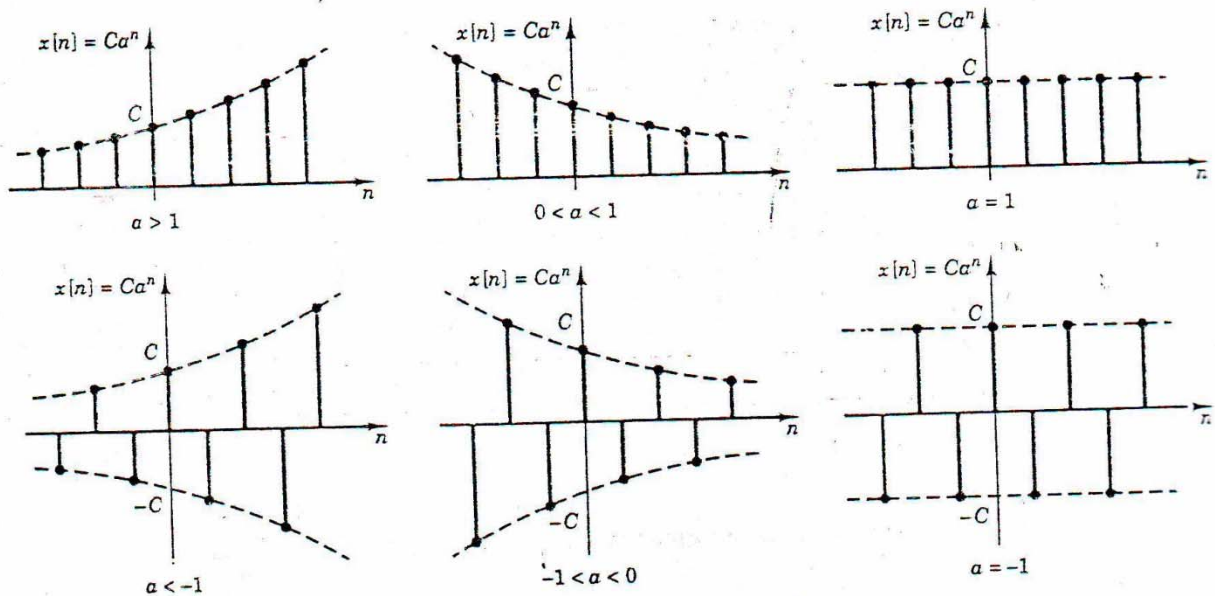


Figure 7: Signal x(n) for C and a are real for $a > 1$, $0 < a < 1$ $a = 1$, $a < -1$, $-1 < a < 0$ and $a = -1$.

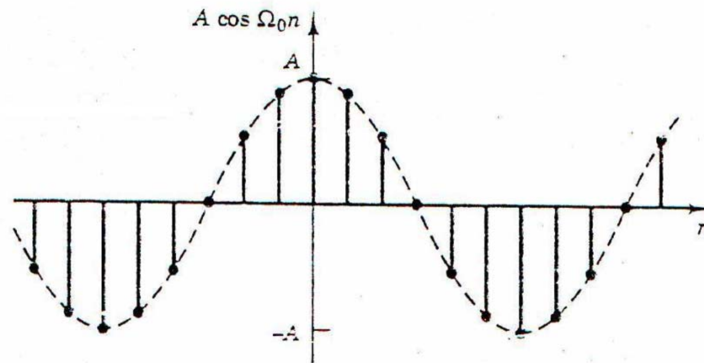


Figure 7: Signal x(n) for C complex, *a* Complex with Unity Magnitude [$C = Ae^{j\phi}$ and $a = e^{j\Omega o}$]

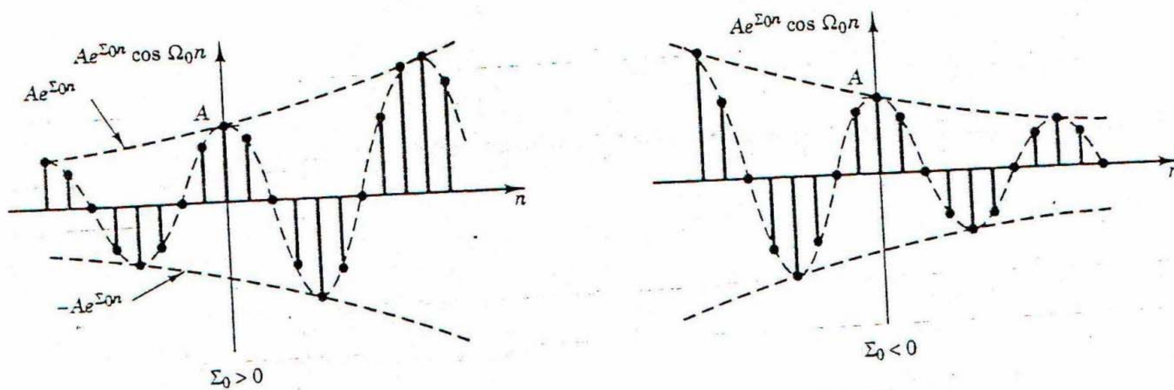


Figure 7: Signal x(n) for both C complex, *a* Complex [$C = Ae^{j\phi}$ and $a = e^{\Sigma o + j\Omega o}$].

AUS

OPERATION ON SEQUENCES

Certain operations on two sequences, such as addition and multiplication, require that the sequences be of equal length, and that their proper positions in time be preserved. Consider the sequence $x1$ = [1,2,3,4], which was sampled at sample time indices $n1$ = [-1,0,1,2], which we would like to add to sequence $x2$ = [4,3,2,1], which was sampled at time indices $n2$ = [2,3,4,5]. To make these two sequences equal in length, we'll prepend and postpend zeros as needed to result in two sequences of equal length that retain the proper time alignment. We see that the minimum time index is -1 and the maximum time index is 5. Since $x1$ starts at the minimum time index, we postpend zeros to it such that we would have $x1$ = [1,2,3,4,0,0,0], with corresponding time indices [-1,0,1,2,3,4,5]. Similarly, we prepend zeros so that $x2$ = [0,0,0,4,3,2,1], with the same total time or sample index range as the modified version of $x1$. Figure 2.4 depicts this process. The sum is then

```
x1 + x2 = [1,2,3,4,0,0,0] + [0,0,0,4,3,2,1] = [1,2,3,8,3,2,1]
```

and has time indices [-1,0,1,2,3,4,5].

These two ideas, that sequences to be added or multiplied must be of equal length, but also properly time-aligned, lead us to write several MathScript functions that will automatically perform the needed adjustments and perform the arithmetic operation.

```
function [y,n] = sigadd(x1,n1,x2,n2)
% implements y(n) = x1(n)+x2(n)
% -----------------------------
% [y,n] = sigadd(x1,n1,x2,n2)
%   y = sum sequence over n, which includes n1 and n2
%  x1 = first sequence over n1
%  x2 = second sequence over n2 (n2 can be different from n1)
%
n = min(min(n1),min(n2)):max(max(n1),max(n2)); % duration of y(n)
y1 = zeros(1,length(n)); y2 = y1;              % initialization
y1(find((n>=min(n1))&(n<=max(n1))==1))=x1;     % x1 with duration of y
y2(find((n>=min(n2))&(n<=max(n2))==1))=x2;     % x2 with duration of y
y = y1+y2;                                     % sequence addition
```

The following script will perform addition of offset sequences $y1$ and $y2$ that have respective time indices $n1$ and $n2$ using the method of prepending and postpending zeros. The `function [y,n] = sidadd(y1, n1, y2, n2)` works the same way, with the addition operator (+) in the final statement being replaced with the operator for multiplying two vectors on a sample-by-sample basis, a period following by an asterisk (.*).

```
function [y,n] = sigmult(x1,n1,x2,n2)
% implements y(n) = x1(n)*x2(n)
% -----------------------------
% [y,n] = sigmult(x1,n1,x2,n2)
%   y = product sequence over n, which includes n1 and n2
%  x1 = first sequence over n1
%  x2 = second sequence over n2 (n2 can be different from n1)
%
n = min(min(n1),min(n2)):max(max(n1),max(n2)); % duration of y(n)
y1 = zeros(1,length(n)); y2 = y1;              %
y1(find((n>=min(n1))&(n<=max(n1))==1))=x1;     % x1 with duration of y
y2(find((n>=min(n2))&(n<=max(n2))==1))=x2;     % x2 with duration of y
y = y1 .* y2;                                  % sequence multiplication
```

TYPES OF SEQUENCES

THE UNIT IMPULSE (DELTA) FUNCTION

The **Unit Impuls**e or **Delta Function is** defined as $\delta[n] = 1$ when $n = 0$ and 0 for all other values of n. The time of occurrence of the impulse can be shifted by a certain number of samples k using the notation $\delta[n - k]$ since the value of the function will only be 1 when $n - k = 0$.

Unit sample sequence:

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} = \left\{ \cdots, 0, 0, \underset{\uparrow}{1}, 0, 0, \cdots \right\}$$

$$\delta(n - n_0) = \begin{cases} 1, & n = n_0 \\ 0, & n \neq n_0 \end{cases}$$

```
function [x,n] = impseq(n0,n1,n2)
% Generates x(n) = delta(n-n0); n1 <= n <= n2
% ----------------------------------------------
% [x,n] = impseq(n0,n1,n2)
%
n = [n1:n2]; x = [(n-n0) == 0];
```

THE UNIT STEP FUNCTION

The **Unit Step Function is** defined as $u[n] = 1$ when $n \geq 0$ and 0 for all other values of n. The time of occurrence of the step (the value 1) can be shifted by a certain number of samples k using the notation $u[n - k]$ since the value of the function will only be 1 when $n - k \geq 0$.

$$u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases} = \left\{ \cdots, 0, 0, \underset{\uparrow}{1}, 1, 1, \cdots \right\}$$

$$u(n - n_0) = \begin{cases} 1, & n \geq n_0 \\ 0, & n < n_0 \end{cases}$$

```
function [x,n] = stepseq(n0,n1,n2)
% Generates x(n) = u(n-n0); n1 <= n <= n2
% ------------------------------------------
% [x,n] = stepseq(n0,n1,n2)
%
n = [n1:n2]; x = [(n-n0) >= 0];
```

PERIODIC SEQUENCES

A sequence that repeats itself exactly is called periodic. A periodic sequence can be generated from a given sequence S of length M by using the outer vector product of the sequence in column vector form and a row vector of N ones. This generates an M-by-N matrix each column

of which is the sequence S. The matrix can then be converted to a single column vector using MathScript's colon operator, and the resulting column vector is converted to a row vector by the transposition operator, the apostrophe. The following function will generate *n* periods of the sequence *y*:

```
function nY = LVMakePeriodicSeq(y,N)
% LVMakePeriodicSeq([1 2 3 4],2)
y = y(:); nY = y*([ones(1,N)]); nY = nY(:)';
```

To illustrate use of the above, we will generate a sequence having three cycles of a cosine wave having a period of 11 samples. One period of the desired signal is

```
cos(2*pi*[0:1:10]/11)
```

and a suitable call that computes and plots the result is therefore

```
N= 3; y = [cos(2*pi*[0:1:10]/11)]';
nY = LVMakePeriodicSeq(y,N); stem(nY)
```

FOLDING

From time to time it is necessary to reverse a sequence in time, i.e., assuming that $x[n]$ = [1,2,3,4], the folded sequence would be $x[-n]$.The operation is essentially to flip the sequence from left to right around index zero. For example, the sequence [1,2,3,4] that has corresponding sample indices [3,4,5,6], when folded, results in the sequence [4,3,2,1] and corresponding indices [-6,-5,-4,-3]. To illustrate the above ideas, we can, for example, let $x[n]$ = [1,2,3,4] with corresponding sample indices n = [3,4,5,6], and compute $x[-n]$ using MathScript. We can write a simple script to accomplish the folding operation:

$$y(n) = \{x(-n)\}$$

```
function [y,n] = sigfold(x,n)
% implements y(n) = x(-n)
% -----------------------
% [y,n] = sigfold(x,n)
%
y = fliplr(x); n = -fliplr(n);
```

SHIFTING

$$y(n) = \{x(n-k)\}$$

```
function [y,n] = sigshift(x,m,n0)
% implements y(n) = x(n-n0)
% -------------------------
% [y,n] = sigshift(x,m,n0)
%
n = m+n0; y = x;
```

EVEN AND ODD DECOMPOSITION

Any real sequence can be decomposed into two components that display even and odd symmetry about the midpoint of the sequence. A sequence that exhibits even symmetry has its first and last samples equal, its second and penultimate samples equal, and so on. A sequence that exhibits odd symmetry has its first sample equal to the negative of the last sample, its second sample equal to the negative of its penultimate sample, etc. An even decomposition *xe* of a sequence *x* can be obtained as

```
xe = 0.5*(x + fliplr(x))
```

and the corresponding odd decomposition *xo* is

```
xo = 0.5*(x - fliplr(x))
```

We can write a simple function that generates the even and odd components of an input sequence *x*[*n*] as follows:

```
function [xe, xo, m] = evenodd(x,n)
% Real signal decomposition into even and odd parts
% -----------------------------------------------
% [xe, xo, m] = evenodd(x,n)
%
if any(imag(x) ~= 0)
      error('x is not a real sequence')
end
m = -fliplr(n);
m1 = min([m,n]); m2 = max([m,n]); m = m1:m2;
nm = n(1)-m(1); n1 = 1:length(n);
x1 = zeros(1,length(m));
x1(n1+nm) = x; x = x1;
xe = 0.5*(x + fliplr(x));
xo = 0.5*(x - fliplr(x));
```

CONVOLUTION

An LTI system is completely characterized in the time domain by the impulse response shown below:

$$x(n) \longrightarrow \boxed{h(n)} \longrightarrow y(n) = x(n) * h(n)$$

Example:1

Given the following two sequences

$$x(n) = \left[3, 11, 7, \underset{\uparrow}{0}, -1, 4, 2\right], \quad -3 \leq n \leq 3; \qquad h(n) = \left[2, \underset{\uparrow}{3}, 0, -5, 2, 1\right], \quad -1 \leq n \leq 4$$

determine the convolution $y(n) = x(n) * h(n)$.

Solution :

The conv function assumes that the two sequences begin at $n = 0$ and is invoked by

```
>> y = conv(x,h);
```

For example, to do the convolution in Example 2.5, we could use

```
>> x = [3, 11, 7, 0, -1, 4, 2];
>> h = [2, 3, 0, -5, 2, 1];
>> y = conv(x,h)
y =
  6    31    47    6    -51    -5    41    18    -22    -3    8    2
```

to obtain the correct $y(n)$ values. However, the conv function neither provides nor accepts any timing information if the sequences have arbitrary support. What is needed is a beginning point and an end point of $y(n)$. Given finite duration $x(n)$ and $h(n)$, it is easy to determine these points. Let

$$\{x(n);\ n_{xb} \le n \le n_{xe}\} \quad \text{and} \quad \{h(n);\ n_{hb} \le n \le n_{he}\}$$

be two finite-duration sequences. Then referring to Example 2.6 we observe that the beginning and end points of $y(n)$ are

$$n_{yb} = n_{xb} + n_{hb} \quad \text{and} \quad n_{ye} = n_{xe} + n_{he}$$

respectively. A simple extension of the conv function, called conv_m, which performs the convolution of arbitrary support sequences can now be designed.

```
function [y,ny] = conv_m(x,nx,h,nh)
% Modified convolution routine for signal processing
% -----------------------------------------------------
% [y,ny] = conv_m(x,nx,h,nh)
%  [y,ny] = convolution result
%  [x,nx] = first signal
%  [h,nh] = second signal
%
nyb = nx(1)+nh(1); nye = nx(length(x)) + nh(length(h));
ny = [nyb:nye];
y = conv(x,h);
```

The output should look like:

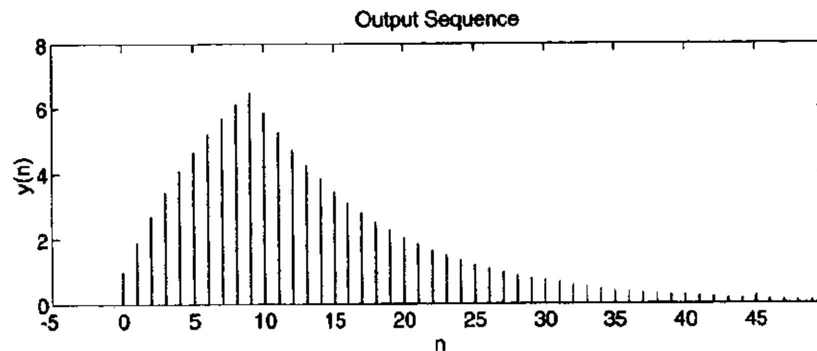


Figure 8: Convolution of the sequences x(n) and h(n)

CORRELATION

The crosscorrelation $r_{yx}(\ell)$ can be put in the form

$$r_{yx}(\ell) = y(\ell) * x(-\ell)$$

with the autocorrelation $r_{xx}(\ell)$ in the form

$$r_{xx}(\ell) = x(\ell) * x(-\ell)$$

Therefore these correlations can be computed using the conv function if sequences are of finite duration.

Example: 2

In this example we will demonstrate one application of the crosscorrelation sequence. Let

$$x(n) = \left[3, 11, 7, \underset{\uparrow}{0}, -1, 4, 2\right]$$

be a prototype sequence, and let $y(n)$ be its noise-corrupted-and-shifted version

$$y(n) = x(n-2) + w(n)$$

where $w(n)$ is Gaussian sequence with mean 0 and variance 1. Compute the crosscorrelation between $y(n)$ and $x(n)$.

Solution:

From the construction of $y(n)$ it follows that $y(n)$ is "similar" to $x(n-2)$ and hence their crosscorrelation would show the strongest similarity at $\ell = 2$. To test this out using MATLAB, let us compute the crosscorrelation using two different noise sequences.

```
% noise sequence 1
>> x = [3, 11, 7, 0, -1, 4, 2]; nx=[-3:3]; % given signal x(n)
>> [y,ny] = sigshift(x,nx,2);               % obtain x(n-2)
>> w = randn(1,length(y)); nw = ny;         % generate w(n)
>> [y,ny] = sigadd(y,ny,w,nw);              % obtain y(n) = x(n-2) + w(n)
>> [x,nx] = sigfold(x,nx);                  % obtain x(-n)
>> [rxy,nrxy] = conv_m(y,ny,x,nx);          % crosscorrelation
>> subplot(1,1,1), subplot(2,1,1);stem(nrxy,rxy)
>> axis([-5,10,-50,250]);xlabel('lag variable l')
>> ylabel('rxy');title('Crosscorrelation: noise sequence 1')
%
% noise sequence 2
>> x = [3, 11, 7, 0, -1, 4, 2]; nx=[-3:3]; % given signal x(n)
>> [y,ny] = sigshift(x,nx,2);               % obtain x(n-2)
>> w = randn(1,length(y)); nw = ny;         % generate w(n)
>> [y,ny] = sigadd(y,ny,w,nw);              % obtain y(n) = x(n-2) + w(n)
>> [x,nx] = sigfold(x,nx);                  % obtain x(-n)
>> [rxy,nrxy] = conv_m(y,ny,x,nx);          % crosscorrelation
>> subplot(2,1,2);stem(nrxy,rxy)
>> axis([-5,10,-50,250]);xlabel('lag variable l')
>> ylabel('rxy');title('Crosscorrelation: noise sequence 2')
```

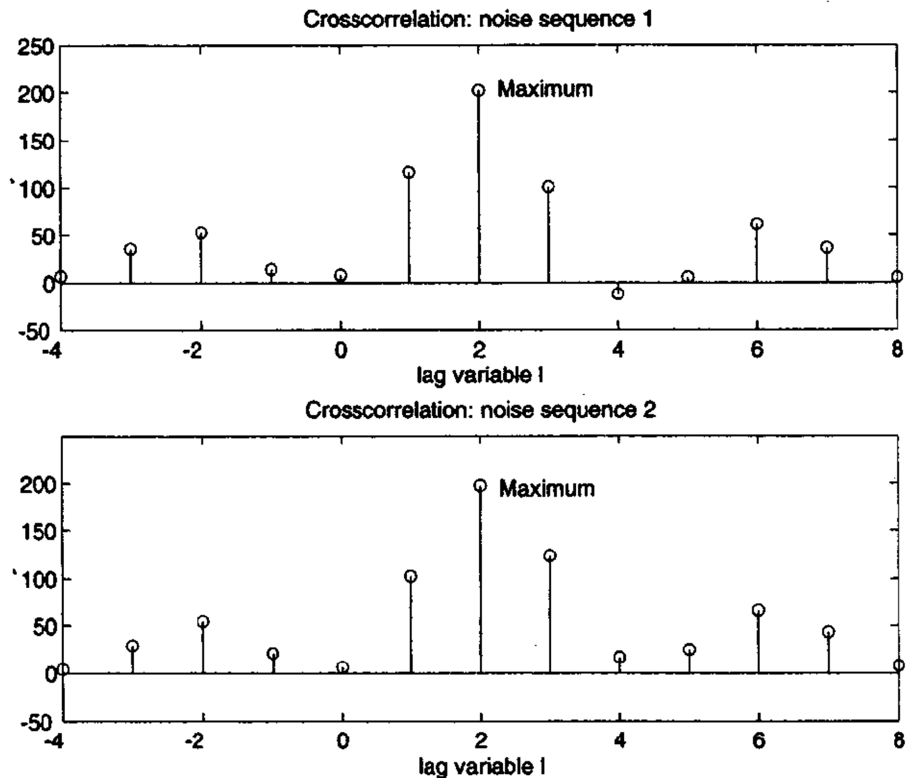


Figure 9 : *Crosscorrelation sequence with two different noise realizations*

we observe that the crosscorrelation indeed peaks at $\ell = 2$, which implies that $y(n)$ is similar to $x(n)$ shifted by 2. This approach can be used in applications like radar

signal processing in identifying and localizing targets.

It should be noted that the signal-processing toolbox in MATLAB also provides a function called **xcorr** for sequence correlation computations. In its simplest form

```
>> xcorr(x,y)
```

computes the crosscorrelation between vectors **x** and **y**, while

```
>> xcorr(x)
```

computes the autocorrelation of vector **x**.

Impulse response and Step response

The causal LTI system can be simulated in MATLAB using the function filter function

```
y = filter (p, d, x)
```

processes the input data vector x using the system characterized by the coefficient vectors p and d to generate the output vector y assuming zero initial conditions. The length of y is the same as the length of x.

Example 3:
Determine the first 41 samples of the impulse response and step response of the causal LTI system defined by--

$$y[n] + 0.7y[n-1] - 0.45y[n-2] - 0.6y[n-3] = 0.8x[n] - 0.44x[n-1] + 0.36x[n-2] + 0.02x[n-3]$$

AUST

Solution:
Illustration of Impulse Response Computation

```
N = input ('Desired impulse response length =');
p = input ('Type in the vector p =');
d = input ('Type in the vector d =');
x = [1 zeros(1,N-1)];
y = filter(p,d,x);
k = 0:1:N-1;
stem(k,y)
xlabel('Time index n'); ylabel('Amplitude')

N =41
p =[0.8   -0.44     0.36]
d =[1     0.7  -0.45 -0.6]
```

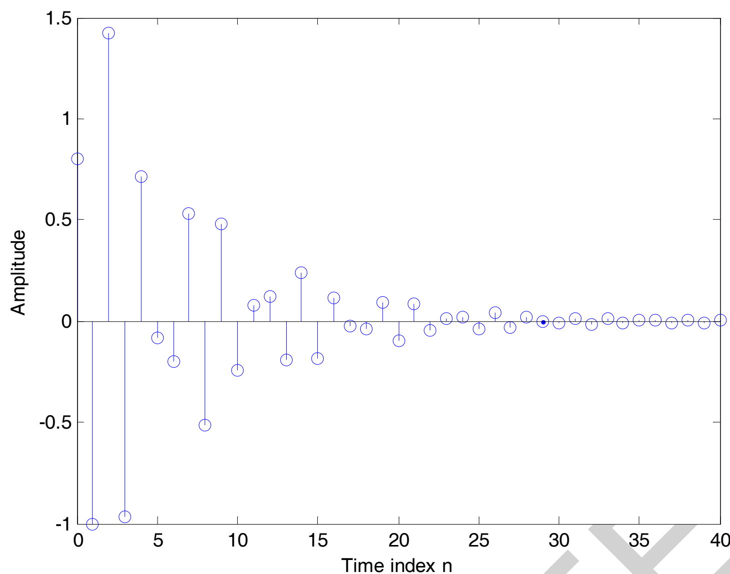


Figure 10: Impulse response of Example 3

Problem: 1

Let $x(n) = \{1, 2, \underset{\uparrow}{3}, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1\}$. Determine and plot the following sequences.

a. $x_1(n) = 2x(n-5) - 3x(n+4)$
b. $x_2(n) = x(3-n) + x(n)\,x(n-2)$

Problem: 2

Generate and plot the samples (use the **stem** function) of the following sequences using MATLAB.

a. $x_1(n) = \sum_{m=0}^{10} (m+1)\,[\delta(n-2m) - \delta(n-2m-1)], \quad 0 \le n \le 25.$
b. $x_2(n) = n^2\,[u(n+5) - u(n-6)] + 10\delta(n) + 20(0.5)^n\,[u(n-4) - u(n-10)].$

Problem: 3

Let $x(n) = (0.8)^n\,u(n)$. Determine $x(n) * x(n)$.

Problem: 4

Let $x(n) = \{1, -2, 4, 6, -5, 8, 10\}$ **and** $h(n) = \left[2, \underset{\uparrow}{3}, 0, -5, 2, 1\right]$**. Determine the crosscorrelation and convolution among the sequences.**

Problem: 5

Determine the first 41 samples of the step response of the causal LTI system defined by **Example 3**. [Hints: Replace in the above program the statement `x = (1 zeros (1,N-1))` with the statement `x = [ones(1, N)]`

SAMPLING AND ALIASING PROBLEM

INTRODUCTION

Aliasing literally means "by a different name" and is used to explain the effect of under-sampling a continuous signal, which causes frequencies to show up as different frequencies. This aliased signal is the signal at a different frequency. This is usually seen as higher frequencies being aliased to lower frequencies. For a 1-dimensional signal in time, the aliased frequency components sound lower in pitch. In 2-dimensional space, such as images, this can be observed as parallel lines in pinstripe shirts aliasing into large wavy lines. For 2-dimensional signals that vary in time, an example of aliasing would be viewing propellers on a plane that seem to be turning slow when they are actually moving at very high speeds.

Note:
The Nyquist sampling rate is twice the highest frequency of the signal. This is the minimum rate needed to prevent aliasing.

In Figure 1 a 500Hz cosine signal is shown in red, and an under-sampled version of the signal in blue.

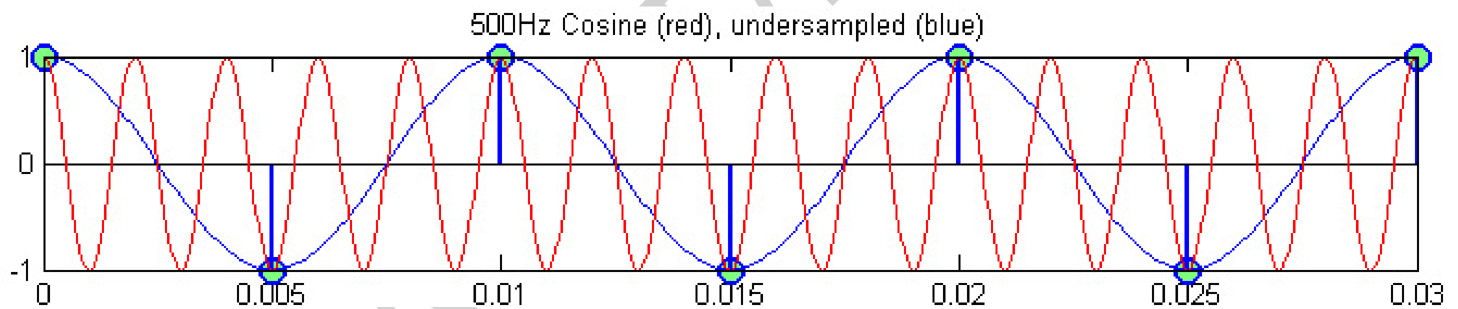


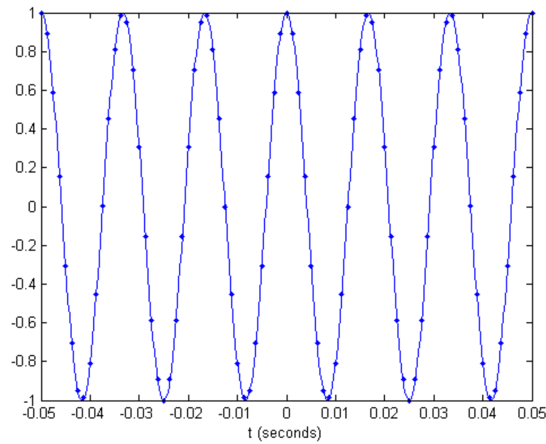
Figure 1: Aliased Signal

Let's start with a continuous-time cosine signal at 60 Hz.

```
f = 60;   % Hz
tmin = -0.05;
tmax = 0.05;
t = linspace(tmin, tmax, 400);
x_c = cos(2*pi*f * t);
plot(t,x_c)
xlabel('t (seconds)')
```

Let's sample x_c with a sampling frequency of 800 Hz.

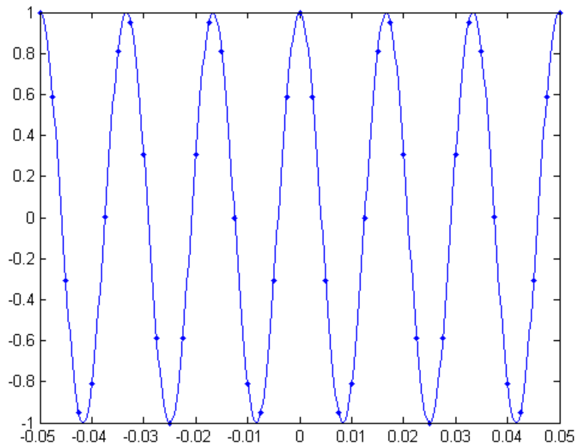
```
T = 1/800;
nmin = ceil(tmin / T);
nmax = floor(tmax / T);
n = nmin:nmax;
x1 = cos(2*pi*f * n*T);
hold on
plot(n*T, x1, 'o')
hold off
```



The sampling frequency of 800 Hz is well above 120 Hz, which is twice the frequency of the cosine. And you can see that the samples are clearly capturing the oscillation of the continuous-time cosine.

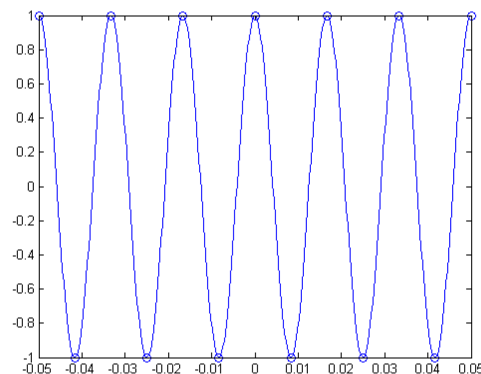
Let's try a lower sampling frequency of 400 Hz which is well above 120 Hz.

```
T = 1/400;
nmin = ceil(tmin / T);
nmax = floor(tmax / T);
n = nmin:nmax;
x1 = cos(2*pi*f * n*T);
plot(t, x_c)
hold on
plot(n*T, x1, 'o')
hold off
```



The samples above are still adequately capturing the shape of the cosine. Now let's drop the sampling frequency down to exactly 120 Hz, twice the frequency of the 60 Hz cosine.

```
T = 1/120;
nmin = ceil(tmin / T);
nmax = floor(tmax / T);
n = nmin:nmax;
x1 = cos(2*pi*f * n*T);
plot(t, x_c)
hold on
plot(n*T, x1, 'o')
hold off
```



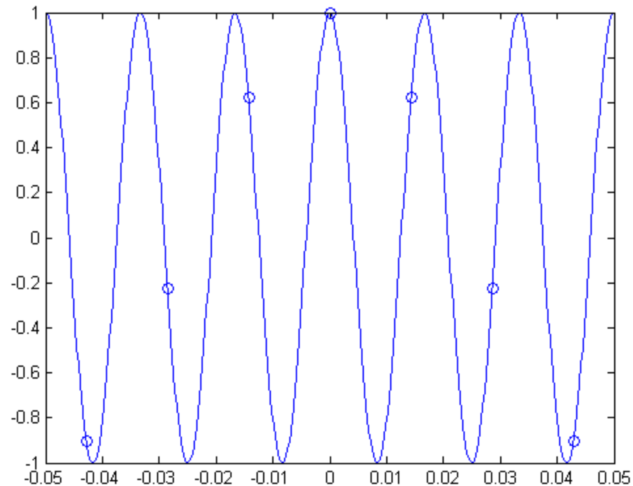
See the samples jump back and forth between 1 and -1. And they capture only the extremes of each period of the cosine oscillation. This is the significance of "twice the highest frequency of the signal" value for sampling frequency. If you'll allow a "hand-

wavy" explanation here, we would say that this sampling frequency of 120 Hz is just enough to capture the cosine oscillation.

But aliasing is worse that "just" losing information. When we drop the sampling frequency too low, the samples start to look increasingly like they came from a different, lower-frequency signal.

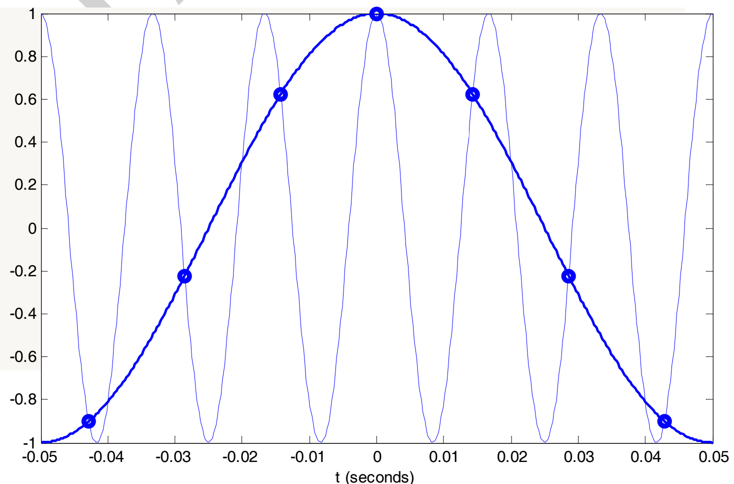
Let's try 70 Hz.

```
T = 1/70;
nmin = ceil(tmin / T);
nmax = floor(tmax / T);
n = nmin:nmax;
x1 = cos(2*pi*f * n*T);
plot(t, x_c)
hold on
plot(n*T, x1, 'o')
hold off
```



The samples above look like they actually could have come from a 10 Hz cosine signal, instead of a 60 Hz cosine signal. Take a look:

```
T = 1/70;
x_c = cos(2*pi*10 * t);
nmin = ceil(tmin / T);
nmax = floor(tmax / T);
n = nmin:nmax;
x1 = cos(2*pi*f * n*T);
plot(t, x_c)
hold on
plot(n*T, x1, 'o')
hold off
```



That's the heart of the "problem" of aliasing. Because the sampling frequency was too low, a high-frequency cosine looked like a low-frequency cosine after we sampled it.

UP-SAMPLER AND DOWN-SAMPLER

An up-sampler with an up-sampling factor L, where L is a positive integer, develops an output sequence $x_u[n]$ with a sampling rate that is L times larger than that of the input sequence x[n]. The up-sampling operation is implemented by inserting L-1 equidistant

zero-valued samples between two consecutive samples of the input sequence x[n] according to the relation

$$x_u[n] = \begin{cases} x\left[\frac{n}{L}\right] & n = 0, \quad \pm L, \quad \pm 2L \ldots\ldots \\ 0 & otherwise \end{cases}$$

Example 1: Study the up-sampling of a sinusoidal input sequence.

```
clear all;
echo on;
N = input('Input length = ') ;
L = input('Up-sampling factor = ');
fo = input('Input Signal Frequency = ');
% Generate the input sinusoidal sequence
n = 0:N-1;
x = sin(2*pi*fo*n);
% Generate up-sampled sequence
y = zeros(1, L*length(x));
y([1:L:length(y)])= x;
% Plot the input and the output sequences
subplot(211)
stem(n,x);
title('Input Sequence');
xlabel('n'); ylabel('Amplitude');
subplot(212)
stem(n,y(1:length(x)));
title(['Output sequence up-sampled by',num2str(L)]);
xlabel('n'); ylabel('Amplitude');
```

AUST/EEE

A down-sampler with a down-sampling factor M, where M is a positive integer, develops an output sequence y[n] with a sampling rate that is [1/M]th of that of the input sequence x[n]. The down-sampling operation is implemented by keeping every Mth sample of the input sequence and removing M-1 in-between samples, to generate the output sequence according to the relation

$$y[n] = x[nM]$$

As a result, all input samples with indices equal to an integer multiple of M are retained at the output and all others are discarded.

Example 2: Study the down-sampling of a sinusoidal input sequence.

```
clear all;
echo on;
N = input('Output length = ') ;
M = input('Down-sampling factor = ');
```

```
fo = input('Input Signal Frequency = ');
% Generate the input sinusoidal sequence
n = 0:N-1;
m = 0:N*M-1;
x = sin(2*pi*fo*m);
% Generate down-sampled sequence
y = x([1:M:length(x)]);
% Plot the input and the output sequences
subplot(211)
stem(n,x(1:N));
title('Input Sequence');
xlabel('n'); ylabel('Amplitude');
subplot(212)
stem(n,y);
title(['Output sequence down-sampled by',num2str(M)]);
xlabel('n'); ylabel('Amplitude');
```

INTERPOLATION PROCESS

```
clear all;
echo on;
N = input('length of input signal = ') ;
L = input('Up-sampling factor = ');
f1 = input('Input Signal Frequency of first sinusoid= '); %
f1 = 0.043
f2 = input('Input Signal Frequency of second sinusoid= ');%
f2 = 0.031
% Generate the input sinusoidal sequence
 n= 0:N-1;
x = sin(2*pi*f1*n)+ sin(2*pi*f2*n);
% Generate interpolated output sequence
y = interp(x,L)
% Plot the input and the output sequences
subplot(211)
stem(n,x(1:N));
title('Input Sequence');
xlabel('n'); ylabel('Amplitude');
subplot(212)
m = 0:N*L-1
stem(m,y(1:N*L));
title('Output sequence');
xlabel('n'); ylabel('Amplitude');
```

Problem : 1

```
clear all; %clears all variables
```

AUST/EEE

```
t=0:.1:20;
F1=.1;
F2=.2;
x=sin(2*pi*F1*t)+sin(2*pi*F2*t);

%plotting
figure(1);
subplot(2,1,1);
plot(t,x);
title('Original signal')
xlabel('t');
ylabel('x(t)');

subplot(2,1,2);
x_samples=x(1:10:201); %gets 21 samples of x.
stem(x_samples,'filled');
title('Sampled signal')
xlabel('n');
ylabel('x_s(n)');
axis([0 20 -2 2]);

%creating dialog box with explanations
l1=[blanks(10),'Sample by sample reconstruction.'];
l2='Blue dots: Input samples.';
l3='Blue curve: reconstructed signal.';
l4='Red curve: contribution to output sample from current
sample.';
l5='Press any key to update with 1 iteration.';
l6='(You can keep this window open while watching the
reconstruction)';
information ={l1,'',l2,l3,l4,'',l5,'',l6};

%starting reconstruction process
figure(2);
messagebox=msgbox(information,'Information','help');
subplot(2,1,2);
plot(t,x,'black');
hold on;
plot([0 20],[0 0],'black');
hold off;
xlabel('t');
ylabel('x(t)');
title('Original signal');
grid;

x_recon=0;
subplot(2,1,1);
```

AUST/EEE

```
for k=0:length(x_samples)-1
    stem(0:length(x_samples)-1,x_samples,'filled');
    if k==length(x_samples)-1
       title('Reconstruction finished');
    else
        title('Sample by sample reconstruction');
    end
    grid on;
    l=k:-.1:-20+k;
    x_recon=x_recon+x_samples(k+1)*sinc(l);
    axis([0 20 -2 2]);
    hold;
    plot(t,x_samples(k+1)*sinc(l),'r')
    plot(t,x_recon);
    hold off;
    waitforbuttonpress;
end
```

Modify the code to generate aliased signal from the signal x. Plot both original and aliased signal in the same scale.

AUST/EEE

z-transform operations with MATLAB

MATLAB Signal Processing Toolbox provides a fast and convenient means of performing a variety of z-transform and inverse z-transform operations for DSP systems design and analysis.

Inverse z-transform

The key MATLAB functions for performing inverse z-transform operations are the **`deconv`** and **`residuez.`** The **`deconv`** function is used to perform the long division required in the power series method. The **`residuez`** function is used to find the partial fraction coefficients (residues) and poles of the z-transform.

Power series expansion with MATLAB

In the power series method, the key operation is polynomial division. The MATLAB function **`deconv`** performs the deconvolution operation. In the power series method, we exploit the fact that the deconvolution operation is equivalent to polynomial division. Thus, given a z-transform X(z), of the form:

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

the format of the command for deconvolution is

```
[q,r] = deconv(b,a)
```

where b and a are vectors representing the numerator and denominator polynomials, b(z) and a(z), respectively, in increasing negative powers of z. The quotient of the polynomial division is returned in the vector q and the remainder is contained in *r.* To implement the power series method, the long division operation is applied successively depending on the number of points required in the inverse operation.

AUST/EEE

Example: 1

Find the first five terms of the inverse z.-transform, x(n), using the power series (polynomial division) method and MATLAB. Assume that the z-transform, X(z), has the following form:

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

Solution:

The coefficient for the numerator and denominator polynomials are formed, zeros are appended to the coefficient vector b to ensure the correct dimension for MATLAB, and then the command **`deconv`** is used to compute the inverse z-transform.

```
b=[1 2 1];
```

```
a=[1 -1 0.3561];
n=5;
b=[b zeros(1, n-1)];
[x, r]=deconv(b,a);
disp(x)

1.0000    3.0000    3.6439    2.5756    1.2780
```

Thus, x(0) = *1, x(1)* = 3, x(2) = 3.6439, x(4) = 2.5756, and x(5) = 1.2780.

Example: 2

Find the first five values of the inverse z-transform of the following using the power series (polynomial division) method and MATLAB:

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

where

$N_1(z) = 1 - 1.22346z^{-1} + z^{-2}$
$N_2(z) = 1 - 0.437833z^{-1} + z^{-2}$
$N_3(z) = 1 + z^{-1}$

$D_1(z) = 1 - 1.4334509z^{-1} + 0.85811z^{-2}$
$D_2(z) = 1 - 1.293601z^{-1} + 0.556929z^{-2}$
$D_1(z) = 1 - 0.612159z^{-1}$

Solution:

The z-transform has three pairs of numerator and denominator polynomials. In the MATLAB implementation (Example-1), the vectors containing the polynomial coefficient are first formed. The MATLAB function **`sos2tf`** (second order sections-to-transfer function)is then used to convert the three pairs of polynomials into a transfer function with a pair of rational polynomials, *b(z)/a(z):*

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

The **`deconv`** function is used to generate the inverse z-transform coefficients. The first five values of the inverse z-transform are

x(0) = *1.0000,* x(1) = 4.6915, *x(2)* = 11.4246, x(3) = 19,5863, *x(4)* = 27.0284

```
n = 5;  % number of power series points
N1 =    [1 -1.122346 1];
D1 = [1 -1.433509 0.858111];
N2 =    [1 1.474597 1];
D2 = [1 -1.293601 0.556929]
N3 = [1 1 0];
D3 = [1 -0.612159 0];
B = [N1; N2; N3];
```

AUST/EEE

```
A = [D1; D2; D3];
[b,a] = sos2tf([B A]);
b = [b zeros(1,n-1 )];
[x,r] = deconv(b,a); %perform long division
disp(x);
```

Partial fraction expansion with MATLAB

The MATLAB function **`residuez`** may be used to perform partial fraction expansion of a z-transform, X(z), expressed as a ratio of two polynomials. The syntax for the **`residuez`** command is

```
[r, p, k] = residuez(b, a)
```

where b and a are vectors representing the numerator and denominator polynomials, *b(z)* and a(z), respectively, in increasing negative powers of z as follows:

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

If the poles of H(z) are distinct, its partial fraction expansion has the form

$$\frac{b(z)}{a(z)} = \frac{r_1}{1 - p_1 z^{-1}} + \cdots \quad \cdots + \frac{r_n}{1 - p_n z^{-1}} + k_1 + k_2 z^{-1} + \cdots \quad \cdots + k_{m-n} z^{-(m-n)}$$

The **`residuez`** function returns the residues of the rational polynomial *b(z)/a(z)* in the vector *r,* the pole positions in p, and the constant terms in k.

Example: 3

Find the partial fraction expansion of the following z-transform.

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

Solution:

```
[r, p, k] = residuez([1,2,1], [1, -1, 0.3561])

r =
  -0.9041 - 5.9928i
  -0.9041 + 5.9928i
p =
   0.5000 + 0.3257i
   0.5000 - 0.3257i
k =
     2.8082
```

Thus, the z-transform, expressed as a partial fraction expansion, becomes

$$X(z) = 2.8082 + \frac{r_1}{1 - p_1 z^{-1}} + \frac{r_2}{1 - p_2 z^{-1}}$$

Where

AUST/EEE

r_1 = -0.9041 - 5.9928i r_2 = -0.9041 + 5.9928i

p_1= 0.5000 + 0.3257i p_2 = 0.5000 - 0.3257i

Example: 4
Find the partial fraction expansion of the function given in Example 2.

Solution:

The MATLAB function **`sos2tf`** is used to convert the numerator and denominator polynomials into a single pair of polynomials, *b(z)/a(z).* The **`residuez`** function is then used to find the partial fraction expansion.

```
N1 = [1 -1.122346 1];
N2 = [1 -0.437833 1];
N3 = [1 1 0];
D1 = [1 -1.433509 0.85811];
D2 = [1 -1.293601 0.556929];
D3 = [1 -0.612159 1];
sos = [N1 D1; N2 D2; N3 D3];
[b,a] = sos2tf(sos);
[r,p,k] = residuez(b,a)
```

Pole-zero diagram

The MATLAB function, zplane, allows the computation and display of the pole-zero dial The syntax for the command is

`zplane(b,a)`

where b and a are the coefficient vectors of the numerator and denominator polynomials, b(z)/a(z). In this format, the command first finds the locations of the poles and zeros (i.e. the roots of b(z) and a(z), respectively) and then plots the z-plane diagram.

Example: 5
A discrete-time system is characterized by the following transfer function:

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

Obtain and plot its pole-zero diagram. Use MATLAB in each case and assume a sampling frequency of 500 Hz and a resolution of < 1 Hz for the frequency response.

Solution:

```
b = [1 -1.6180 1];  % form numerator and denominator polynomials
a = [1 -1.5161 0.878];
```

AUST/EEE

```
zplane(b,a) % compute and plot the pole-zero diagram
```

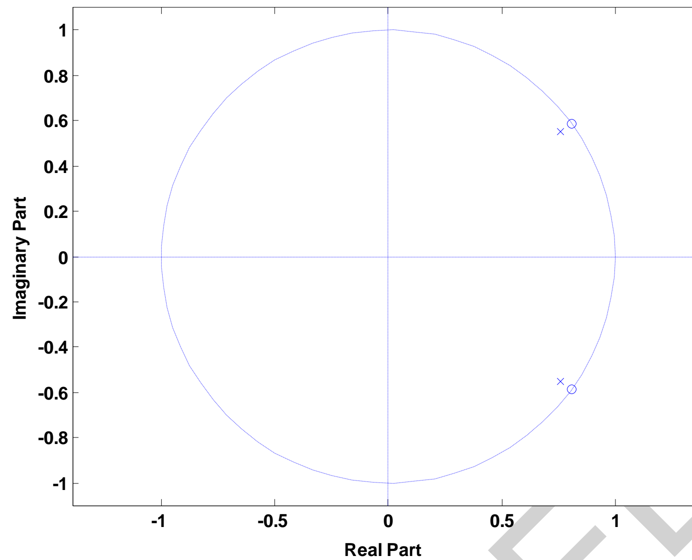


Figure 1: Pole-zero plot

AUST/EEL

If the locations of the poles and zeros are known, these can be used as inputs to the **zplane** command. The syntax of the command in this case is **zplane(z, p),** where z and p are the zeros and poles.

The locations of the poles and zeros can be found directly using the roots command. This is useful for converting between pole and zero and the transfer function representations. For example, an IIR filter is represented by

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

```
b = [1 -1.618 11];
a = [1 -1 .5161 0.878];
zk = roots(b);
pk = roots(a);
```

The numerator and denominator polynomials, b(z) and a(z), can be obtained using the **poly** function:

```
B = poly(zk); A = poly(pk);
```

Frequency response estimation

The Signal Processing Toolbox contains many useful functions for computing and displaying the frequency response of discrete-time systems. The most widely used is the **fregz** function. Given the transfer function of a system in the following form:

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

the **fregz** function uses an FFT-based approach to compute the frequency response. The function has a variety of formats. A useful format is **[h,f] = freqz(b, a, npt, Fs)**, where the variables b and a are the vectors of the numerator and denominator polynomials. Fs is the sampling frequency and npt the number of frequency points between 0 and Fs/2. In the MATLAB Toolbox, the Nyquist frequency (i.e. Fs/2) is the unit of normalized frequency, Using the **fregz** command without output arguments plots the magnitude and phase responses automatically.

Example: 6
Repeat **Example 5** using **freqz**.

Solution:

```
b = [1 -1.6180 1]; % form numerator and denominator coefficient
%vectors
a = [1 -1.5161 0.878];
freqz(b,a,256,500); % compute and plot the frequency response
```

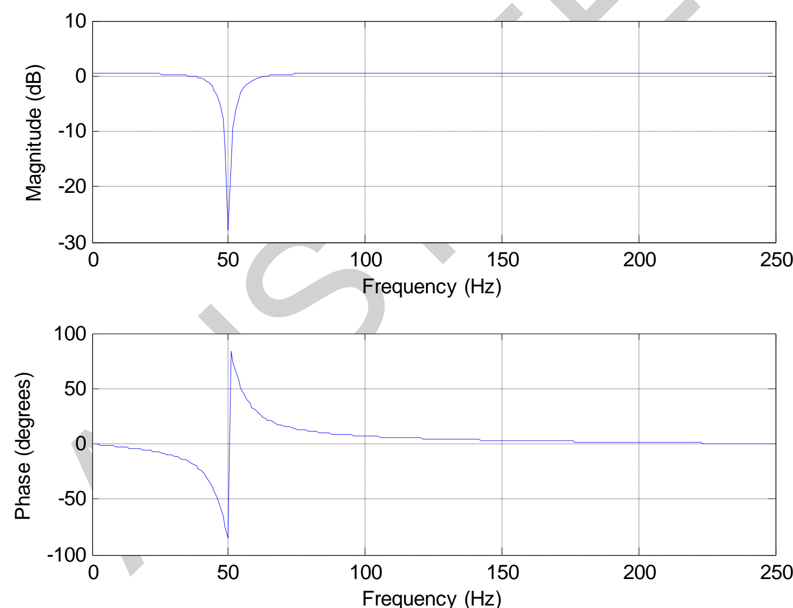


Figure 2: Frequency and phase response

Conversion between structures - cascade-to-parallel conversion

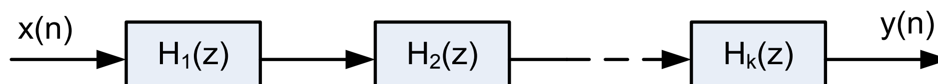


Figure 3: General structure for cascade realization

For cascade realization, the transfer function H(z) is factored as

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

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

$$H_i(z) = \frac{b_0 + b_{1i}z^{-1} + b_{2i}z^{-2}}{1 + a_{1i}z^{-1} + a_{2i}z^{-2}} \qquad second-order$$

$$H_i(z) = \frac{b_0 + b_{1i}z^{-1}}{1 + a_{1i}z^{-1}} \qquad first-order$$

where k is the integer part of (M+1)/2

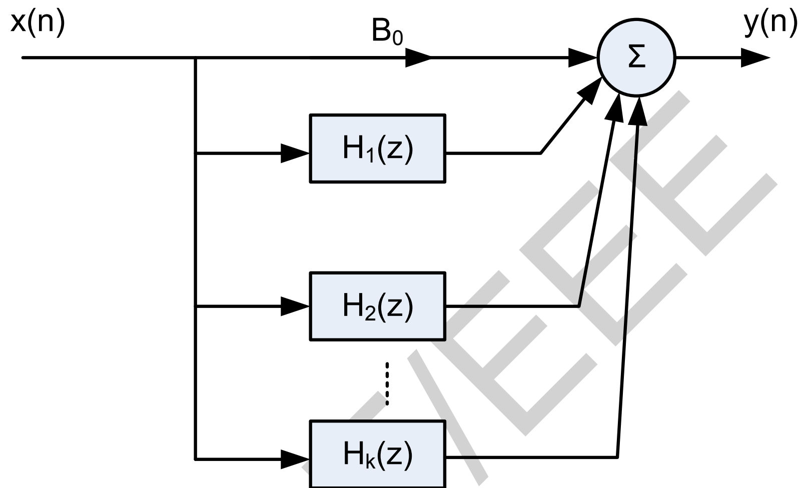


Figure 4: General structure for parallel realization

AUST/EEE

For parallel realization, the transfer function H(z) is decomposed using partial fractions to give

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

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

$$H_i(z) = \frac{a_{0i} + a_{1i}z^{-1}}{1 + b_{1i}z^{-1} + b_{2i}z^{-2}} \qquad second-order$$

$$H_i(z) = \frac{a_0}{1 + +b_{1i}z^{-1}} \qquad first-order$$

where k is the integer part of (M+1)/2 and $B_0 = a_N/b_M$

MATLAB provides a set of functions that allow the conversion between different formats and structures that are used in DSP relatively easily. The ability to convert between the parallel and cascade structures is particularly useful.

Example: 7
Repeat **Example 2** for converting cascade-to-parallel structure.

Solution:

```
nstage=2;
N1 = [1 0.481199 1];
N2 = [1 1.474597 1];
D1 = [1 0.052921 0.831731];
D2 = [1 -0.304609 0.238865];
sos = [N1 D1; N2 D2];
[b, a] = sos2tf(sos);
[c, p, k] = residuez(b, a);
m = length(b);
b0 = b(m)/a(m);
j=1;
for i=1:nstage
    bk(j)=c(j)+c(j+1);
    bk(j+1)=-(c(j)*p(j+1)+c(j+1)*p(j));
    ak(j)=-(p(j)+p(j+1));
    ak(j+1)=p(j)*p(1+j );
    j=j+2;
end
b0
ak
bk
c
p
k
```

UST/EEE

Problem: 1

Determine the inverse z-transform of

$$X(z) = \frac{1 + 0.4\sqrt{2}z^{-1}}{1 - 0.8\sqrt{2}z^{-1} + 0.64z^{-2}}$$

so that the resulting sequence is causal and contains no complex numbers.

Problem: 2

Given a causal system

$$y(n) = 0.9y(n-1) + x(n)$$

a. Find $H(z)$ and sketch its pole-zero plot.
b. Plot $\left|H(e^{j\omega})\right|$ and $\angle H(e^{j\omega})$.
c. Determine the impulse response $h(n)$.

Discrete Time Fourier Series (DTFS)

Let us consider a sequence *x(n)* with period N, that is *x(n)=x(n+N)* for all *n* .The Fourier series representation of *x(n)* consists of *N* harmonically related exponential functions $e^{\frac{j2\pi kn}{N}}$, $k = 0,1,...,N\text{-}1$
And expressed as

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

This equation is often called the discrete-time Fourier series (DTFS), Fourier coefficients $\{c_k\}$,k=0,1,….,N-1 provided the description of x(n) is in frequency domain. $\{c_k\}$ can b computed as

Note that $c_{k+N} = c_k$

That is, $\{c_k\}$ is a periodic sequence with fundamental period N. Thus the spectrum of a signal *x(n),*which is periodic with period N, is a periodic sequence with period N.

Average power can be given as

$$P_x = \sum_{k=0}^{N-1} |c_k|^2 = \frac{1}{N}\sum_{k=0}^{N-1} |x(n)|^2$$

The sequence $|c_k|^2$ for k=0,1,….,N-1 is the distribution of power as a function of frequency and is called the power density spectrum of the periodic signal.
If the signal x(n) is real [$x^*(n) = x(n)$],then it can be shown that $c_k^* = c_{-k}$.
Again the following symmetry relationship holds

$|c_k| = |c_{N-k}|$ and $\angle c_k = -\angle c_{N-k}$

$|c_{\frac{N}{2}}| = |c_{\frac{N}{2}}|$ and $\angle c_{\frac{N}{2}} = 0$ if N is even

$|c_{(N-1)/2}| = |c_{(N+1)/2}|$ and $\angle c_{(N-1)/2} = -\angle c_{(N+1)/2}$ if N is odd.

AUST/EEE

Example :1

Calculate Fourier Series Coefficients of a continuous rectangular pulse sequence of 1ms period and pulse width 0.1ms.The signal is sampled at 100kHz and sampled discrete with n its index is shown below.

Solution:

The following MATLAB code calculates the Fourier series coefficients from the first principle and from the Fourier series coefficients, again reconstructs the original periodic sequence.

```
%CALCULATION OF FOURIER SERIES COEFFICIENTS

clear all;
echo on;
Fs = 100e3;
dt = 1/Fs;
%GENERATING THE RECTANGULAR PULSE TRAIN
T = 1e-3;   %PERIOD OF THE PULSE TRAIN
D = 0.1;    %DUTY CYCLE
PW = D*T;   %PULSE WIDTH
f = 1/T;    %ANALOG FREQUENCY
t = -T/2:dt:T/2;    %TIME INTERVAL FOR A PERIOD
n = t/dt;           %INDEX FOR DATA POINTS IN A PERIOD
L = PW/dt;      %DATA POINTS IN THE THE HIGH TIME

x = zeros(1,length(t));     %INITIALIZING A SINGLE
%RECTANGULAR PULSE
x(find(abs(n)<=L/2)) = 1.1;     %GENERATION OF A SINGLE
%RECTANGULAR PULSE
%END OF THE RENTANGULAR PULSE TRAIN
figure(1)
subplot(211)
plot(t,x)
xlabel('Time (Seconds)')
ylabel('x(t)')
title('Continuous signal')
subplot(212)
stem(n,x)
xlabel('n')
ylabel('x(n)')
title('Discrete signal')
N = length(x); % TOTAL NO OF DATA POINTS IN A PERIOD
Nc = N; %TOTAL NO COEFFICIENTS
if mod(Nc,2) == 0
    k = -(Nc/2):(Nc/2)-1;
else
```

AUST/EEE

```
    k = -(Nc-1)/2:(Nc-1)/2;
end
c = zeros(1,length(k));   %INITIALIZING FOURIER COEFFICIENTS
for i1=1:length(k)
    for i2=1:length(x)
        c(i1)=c(i1)+1/N*x(i2)*exp(-i*2*pi*k(i1)*n(i2)/N);
    end
end
figure(2)
subplot(211)
stem(k,abs(c));
xlabel('k')
ylabel('|c_k|')
title('Fourier series coefficients c_k')
subplot(212)
stem(k,angle(c)*130/pi)
xlabel('k')
ylabel('angle(c_k)')
%START OF RECONSTRUCTION OF SIGNAL
t_remax = T/2;
t_re = -t_remax:dt:t_remax;
n_re=t_re/dt;
x_re=zeros(1,length(n_re));
for i1=1:length(k)
    for i2=1:length(x_re)
x_re(i2)= x_re(i2)+c(i1)*exp(i*2*pi*k(i1)*n_re(i2)/N);
    end
end
%END OF RECONSTRUCTION OF SIGNAL
```

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

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

```
figure(3)
subplot(211)
stem(n_re, x_re)
xlabel('n')
ylabel('x_{reconstructed}')
title('Reconstructed signal')
subplot(212)
plot(t_re, x_re)
xlabel('t')
ylabel('x_{reconstructed}')
title('Reconstructed signal')
```

AUST/EEE

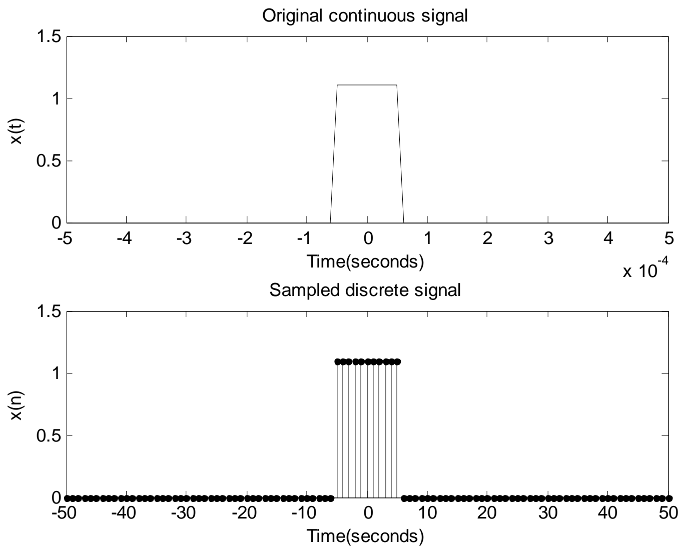


Figure: 1 Rectangular pulse in continuous domain and its sampled version

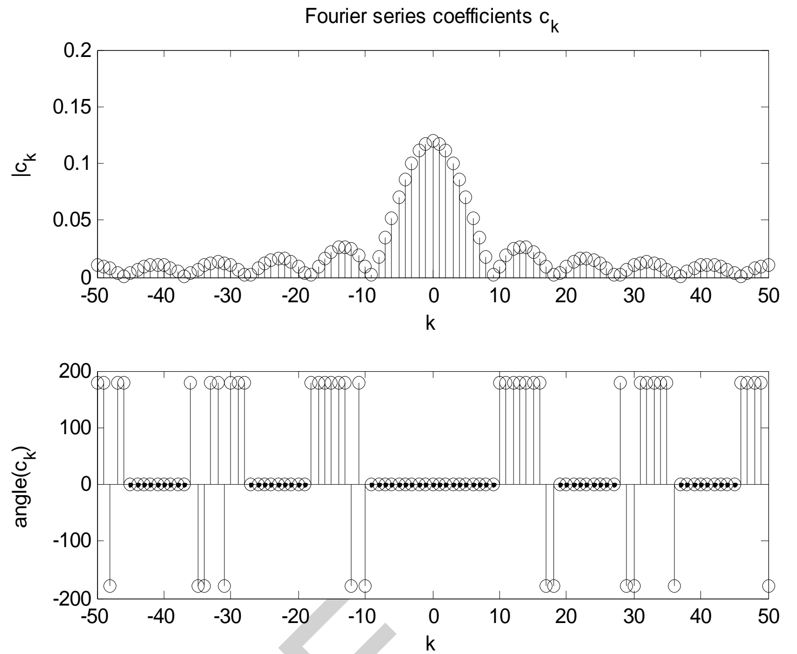


Figure 2: Magnitude and phase spectrum of the discrete time signal. The phase spectrum is not skew-symmetric with respect to k=0. It is an artifact of MATLAB and θ= +180 and -180 are in fact synonymous.

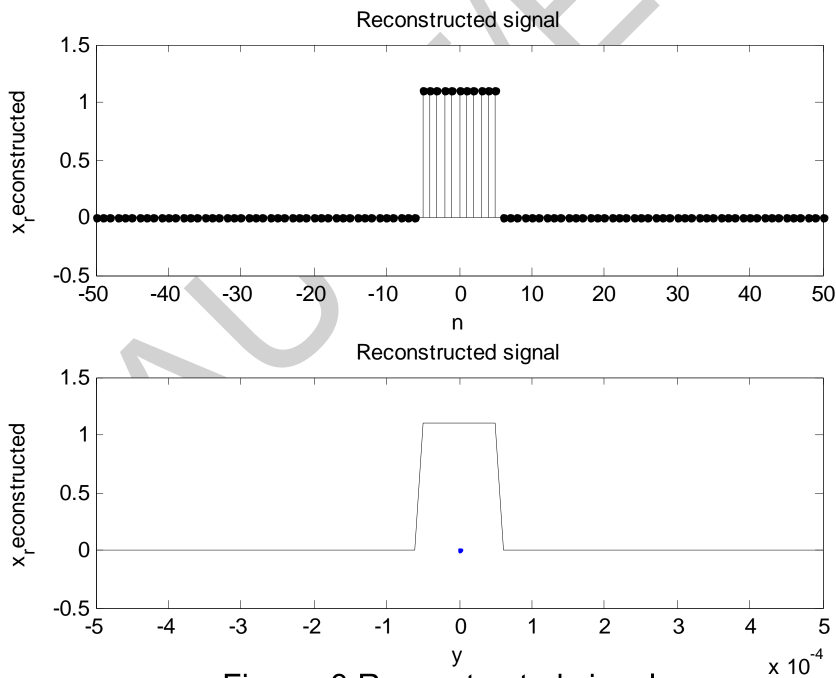


Figure: 3 Reconstructed signal.

Discrete Time Fourier Transform (DTFT)

If $x(n)$ is absolutely summable, that is, $\sum_{-\infty}^{\infty} |x(n)| < \infty$, then its discrete-time Fourier transform is given by

$$X(e^{j\omega}) \triangleq \mathcal{F}[x(n)] = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \quad \text{------------------------------- (1)}$$

The inverse discrete-time Fourier transform (IDTFT) of $X(e^{j\omega})$ is given by

$$x(n) \triangleq \mathcal{F}^{-1}\left[X(e^{j\omega})\right] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}d\omega \quad \text{---------------------- (2)}$$

If $x(n)$ is of finite duration, then MATLAB can be used to compute $X(e^{j\omega})$ numerically at any frequency ω. The approach is to implement (1) directly. If, in addition, we evaluate $X(e^{j\omega})$ at equispaced frequencies between $[0, \pi]$, then (1) can be implemented as a *matrix-vector multiplication* operation. To understand this, let us assume that the sequence $x(n)$ has N samples between $n_1 \leq n \leq n_N$ (i.e., not necessarily between $[0, N-1]$) and that we want to evaluate $X(e^{j\omega})$ at

$$\omega_k \triangleq \frac{\pi}{M}k, \quad k = 0, 1, \ldots, M$$

which are $(M+1)$ equispaced frequencies between $[0, \pi]$. Then (1) can be written as

$$X(e^{j\omega_k}) = \sum_{\ell=1}^{N} e^{-j(\pi/M)kn_\ell}x(n_\ell), \quad k = 0, 1, \ldots, M$$

When $\{x(n_\ell)\}$ and $\{X(e^{j\omega_k})\}$ are arranged as *column* vectors $\mathbf{x}$ and $\mathbf{X}$, respectively, we have

$$\mathbf{X} = \mathbf{W}\mathbf{x} \quad \text{---------------------------- (3)}$$

where $\mathbf{W}$ is an $(M+1) \times N$ matrix given by

$$\mathbf{W} \triangleq \left\{e^{-j(\pi/M)kn_\ell};\ n_1 \leq n \leq n_N, \quad k = 0, 1, \ldots, M\right\}$$

In addition, if we arrange $\{k\}$ and $\{n_\ell\}$ as *row* vectors $\mathbf{k}$ and $\mathbf{n}$ respectively, then

$$\mathbf{W} = \left[\exp\left(-j\frac{\pi}{M}\mathbf{k}^T\mathbf{n}\right)\right]$$

In MATLAB we represent sequences and indices as row vectors; therefore taking the transpose of (3), we obtain

$$\mathbf{X}^T = \mathbf{x}^T\left[\exp\left(-j\frac{\pi}{M}\mathbf{n}^T\mathbf{k}\right)\right] \quad \text{---------------------------- (4)}$$

Example 2:
Determine the discrete-time Fourier transform of the following finite-duration sequence x(n)={1,2,3,4,5} at 501 equispaced frequencies between (0,π).
↑

Solution:

```
n = -1:3; x = 1:5;
k = 0:500; w = (pi/500)*k;
X = x * (exp(-j*pi/500)) .^ (n'*k);
magX = abs(X); angX = angle(X);
realX = real(X); imagX = imag(X);
subplot(2,2,1); plot(k/500,magX);grid
xlabel('frequency in pi units'); title('Magnitude Part')
subplot(2,2,3); plot(k/500,angX/pi);grid
xlabel('frequency in pi units'); title('Angle Part')
subplot(2,2,2); plot(k/500,realX);grid
xlabel('frequency in pi units'); title('Real Part')
subplot(2,2,4); plot(k/500,imagX);grid
xlabel('frequency in pi units'); title('Imaginary Part')
```

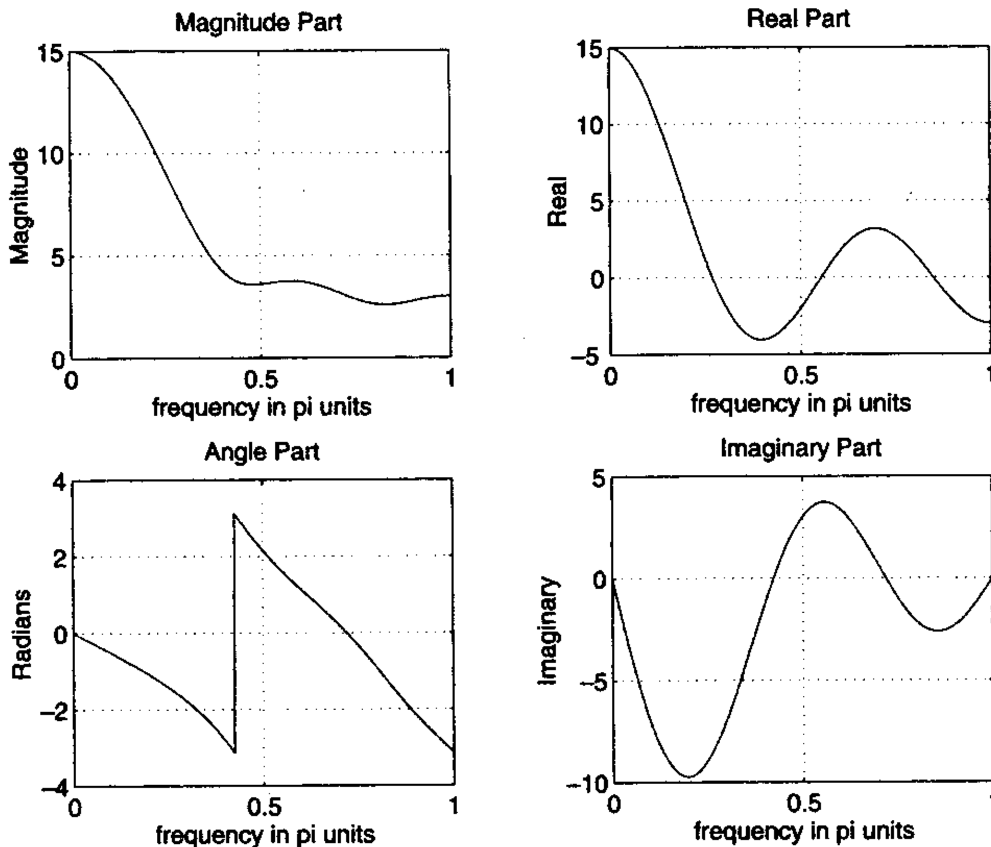


Figure: 4 Amplitude and phase spectrum

DTFT equations in matrix form

We can arrange DTFT equations in Matrix form.
Let w be the vector containing M frequency points between [0, 2 π] (M X 1)
x be the data sequence having N data points (N X 1)
n be the time index vector (N X 1)

$$X = Wx^T$$

Where W is an M×N matrix defined as

$$W=\begin{pmatrix} e^{-j\omega(0)n(0)} & \cdots & -j\omega(0)n(N-1) \\ \vdots & \ddots & \vdots \\ e^{-j\omega(M-1)n(0)} & \cdots & -j\omega(M-1)n(N-1) \end{pmatrix}$$

Example 3:
Calculates the DTFT of a sequence x= [1,3,-9,5,10].

Solution:

```
x=[1,3,-9,5,10];
n1=-1;                  %Defining the index of first element of
x
n2=3;                   %Defining the index of last element of
x
n=n1:n2;                %Index of x
M=500;                  %Total number of points in the
frequency range
w=(-M/2:M/2)*2*pi/M     %Frequency grid
W=exp(-j*w'*n)           %Matrix formation
X=W*x';
subplot(2,1,1)
plot(w/(2*pi),abs(X),'k')
xlabel('Digital frequency ,f')
ylabel('X|f|'),title('Magnitude Spectrum')
subplot(2,1,2)
plot(w/(2*pi),angle(X)*180/pi,'k')
xlabel('Digital frequency ,f')
ylabel('angle X(f)')
title('Phase Spectrum')
```

AUST/EEE

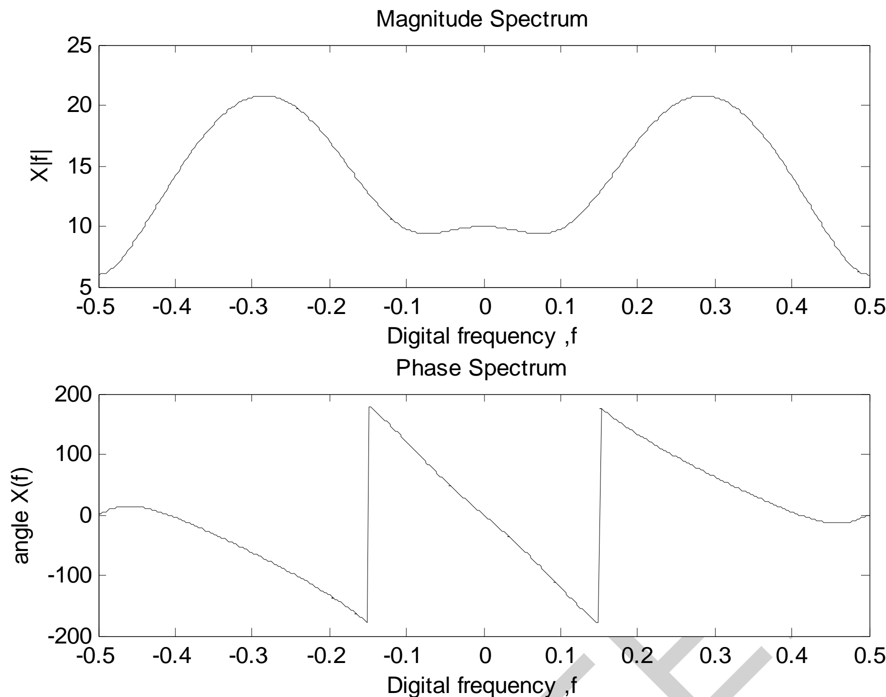


Figure 5: DTFT of sequence calculated by matrix method

AUST/EE

Discrete Fourier Transform (DFT)

We know that aperiodic finite energy signals have continuous spectra (DTFT).

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

In case of a finite length sequence x(n), $0 \le n \le L-$,only L values of X(ω) over its period, called the frequency samples, are sufficient to determine x(n) and hence X(ω). This leads to the concept of discrete Fourier transform (DFT) which is obtained by aperiodic sampling of X(ω)(DTFT).

We often compute a higher point (N point) DFT where N>L. This is because padding the sequence x(n) with N-L zeros and computing an N point DFT results in a “better display” of the Fourier transform X(ω).

To summarize, the formulas are (for causal sequence)

$$X(k)=\sum_{n=0}^{N-1} x(n)e^{-\frac{j2\pi kn}{N}} = \quad x(n)e^{\frac{j \quad kn}{}} (DFT),k=0,1,...,N-1$$

$$x(n)=\frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{\frac{j2\pi kn}{N}} (IDFT),n=0,1,...,N-1$$

DFT matrix equation is given by

$$X = W_N x \qquad \text{DFT equation}$$

$$X = \frac{1}{N} W_N x \qquad \text{IDFT equation}$$

Where $W_N = e^{-j2\pi kn}$

$$x = [\,x(0)x(1)...x(N\ \ 1)]^T$$
$$X = [\,X(0)X(1)...X(N\ \ 1)]^T$$

$$W_{N=}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & \cdots & W_N^{N-} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & W_N^{N\ 1} & W_N^{2(N-1)} & \cdots & W_N^{(N\ 1)^2} \end{bmatrix} \quad \downarrow\text{k (frequency index)}$$

→n (time index)

IST/EEE

Note: W_N matrix can be generated by *dftmtx()* function.

The functions **`dfs(xn,N)`** and **`idfs(Xk,N)`** can be used for DFS and IDFS:

```
function [Xk] = dfs(xn,N)
% Computes Discrete Fourier Series Coefficients
% -----------------------------------------------
% [Xk] = dfs(xn,N)
% Xk = DFS coeff. array over 0 <= k <= N-1
% xn = One period of periodic signal over 0 <= n <= N-1
%  N = Fundamental period of xn
%
n = [0:1:N-1];                % row vector for n
k = [0:1:N-1];                % row vecor for k
WN = exp(-j*2*pi/N);          % Wn factor
nk = n'*k;                    % creates a N by N matrix of nk values
WNnk = WN .^ nk;              % DFS matrix
Xk = xn * WNnk;               % row vector for DFS coefficients
```

```
function [xn] = idfs(Xk,N)
% Computes Inverse Discrete Fourier Series
% ----------------------------------------
% [xn] = idfs(Xk,N)
% xn = One period of periodic signal over 0 <= n <= N-1
% Xk = DFS coeff. array over 0 <= k <= N-1
%  N = Fundamental period of Xk
%
n = [0:1:N-1];                 % row vector for n
k = [0:1:N-1];                 % row vecor for k
WN = exp(-j*2*pi/N);           % Wn factor
nk = n'*k;                     % creates a N by N matrix of nk values
WNnk = WN .^ (-nk);            % IDFS matrix
xn = (Xk * WNnk)/N;            % row vector for IDFS values
```

Example 4:

A periodic "square wave" sequence is given by

$$\tilde{x}(n) = \begin{cases} 1, & mN \le n \le mN + L - 1 \\ 0, & mN + L \le n \le (m+1)N - 1 \end{cases}; \quad m = 0, \pm 1, \pm 2, \ldots$$

where N is the fundamental period and L/N is the duty cycle.

Plot the magnitude $|X(k)|$ for $L = 5$, $N = 20$; $L = 5$, $N = 40$; $L = 5$, $N = 60$; and $L = 7$, $N = 60$.

Solution:

```
L = 5; N = 20; k = [-N/2:N/2];                  % Sq wave parameters
xn = [ones(1,L), zeros(1,N-L)];                 % Sq wave x(n)
Xk = dfs(xn,N);                                 % DFS
magXk = abs([Xk(N/2+1:N) Xk(1:N/2+1)]);  % DFS magnitude
subplot(2,2,1); stem(k,magXk); axis([-N/2,N/2,-0.5,5.5])
xlabel('k'); ylabel('Xtilde(k)')
title('DFS of SQ. wave: L=5, N=20')
```

Example 5:

Let $x(n) = (0.7)^n u(n)$. Sample its z-transform on the unit circle with $N = 5$, 10, 20, 50 and study its effect on the time domain.

Solution:

z-transform of $x(n)$ is

$$X(z) = \frac{1}{1 - 0.7z^{-1}} = \frac{z}{z - 0.7}, \quad |z| > 0.7$$

```
N = 5; k = 0:1:N-1;                                      % sample index
wk = 2*pi*k/N; zk = exp(j*wk);                           % samples of z
Xk = (zk)./(zk-0.7);                                     % DFS as samples of X(z)
xn = real(idfs(Xk,N));                                   % IDFS
xtilde = xn'* ones(1,8); xtilde = (xtilde(:))'; % Periodic sequence
subplot(2,2,1); stem(0:39,xtilde);axis([0,40,-0.1,1.5])
xlabel('n'); ylabel('xtilde(n)'); title('N=5')
```

The functions `dft(xn,N)` and `idft(Xk,N)` can be used for DFT and IDFT:

```
function [Xk] = dft(xn,N)
% Computes Discrete Fourier Transform
% -----------------------------------
% [Xk] = dft(xn,N)
% Xk = DFT coeff. array over 0 <= k <= N-1
% xn = N-point finite-duration sequence
%  N = Length of DFT
%
n = [0:1:N-1];                 % row vector for n
k = [0:1:N-1];                 % row vecor for k
WN = exp(-j*2*pi/N);           % Wn factor
nk = n'*k;                     % creates a N by N matrix of nk values
WNnk = WN .^ nk;               % DFT matrix
Xk = xn * WNnk;                % row vector for DFT coefficients
```

```
function [xn] = idft(Xk,N)
% Computes Inverse Discrete Transform
% -----------------------------------
% [xn] = idft(Xk,N)
% xn = N-point sequence over 0 <= n <= N-1
% Xk = DFT coeff. array over 0 <= k <= N-1
%  N = length of DFT
%
n = [0:1:N-1];                 % row vector for n
k = [0:1:N-1];                 % row vecor for k
WN = exp(-j*2*pi/N);           % Wn factor
nk = n'*k;                     % creates a N by N matrix of nk values
WNnk = WN .^ (-nk);            % IDFT matrix
xn = (Xk * WNnk)/N;            % row vector for IDFT values
```

Example 6:

Let $x(n)$ be a 4-point sequence:

$$x(n) = \begin{cases} 1, & 0 \le n \le 3 \\ 0, & \text{otherwise} \end{cases}$$

Compute the 4-point DFT of $x(n)$

Solution:

```
x = [1,1,1,1]; N = 4;
X = dft(x,N);
magX = abs(X), phaX = angle(X)*180/pi
```

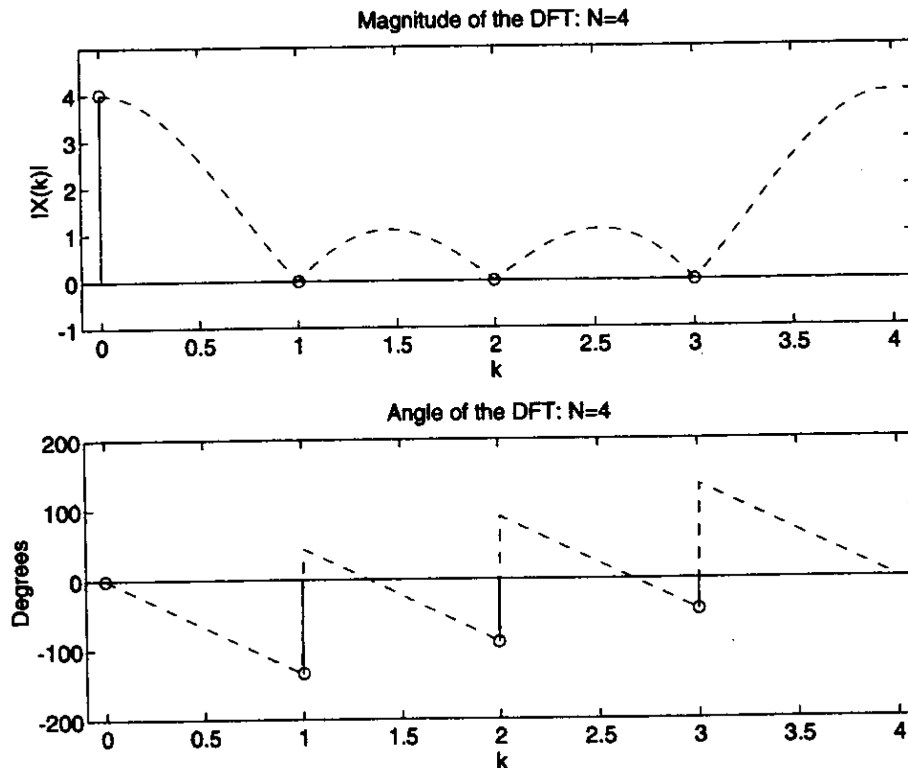


Figure 6: DTF plot of x(n)

Fast Fourier Transform (FFT)

There are four built-in functions in MATLAB for the computation of the DFT and the IDFT:

`fft(x)`, `fft(x,N)`, `ifft(X)`, `ifft(X,N)`

All of these functions make use of FFT algorithms which are computationally highly efficient compared to the direct computation of DFT and the inverse DFT.

The function `fft(x)` computes the R-point DFT of a vector `x`, with R being the length of `x`. For computing the DFT of a specific length N, the function `fft(x,N)` is used. Here, if $R > N$, it is truncated to the first N samples, whereas, if $R < N$, the vector x is zero-padded at the end to make it into a length-N sequence. Likewise, the function `ifft(X)` computes the R-point IDFT of a vector `X`, where R is the length of `X`, while `ifft(X,N)` computes the IDFT of `X`, with the size N of the IDFT being specified by the user. As before, if $R > N$, it is automatically truncated to the first N samples, whereas, if $R < N$, the DFT vector `X` is zero-padded at the end by the program to make it into a length-N DFT sequence.

Example 7:
A signal sinc(100t) is modulated with cos(500πt) carrier. Write a Matlab program to plot entire modulation and demodulation using fft and ifft.

Solution:

```
function [M,m,df1]=fftseq(m,ts,df)
fs=1/ts;
n1=fs/df;
n2=length(m);
n=2^(max(nextpow2(n1),nextpow2(n2)));
M=fft(m,n);
m=[m,zeros(1,n-n2)];
df1=fs/n;

clear all;
echo on;
t0 = 0.2;
ts =  8.3333e-004;
fc = 250;
fs = 1/ts;
df = 0.3;
t = [-t0/2:ts:t0/2];
m = sinc(100*t); % MESSAGE SIGNAL
subplot(241)
plot(t,m)
xlabel('t')
ylabel('Amplitude')
title('Message Signal')
c = cos(2*pi*fc*t); % carrier
u = m.*c; % Modulation
[M,m,df1] = fftseq(m,ts,df);
M = M/fs;
 [C,m,df1] = fftseq(c,ts,df);
 C = C/fs;
[U,u1,df1] = fftseq(u,ts,df);
U = U/fs;
f = [0:df1:df1*(length(m)-1)]-fs/2;
subplot(242)
plot(f,abs(fftshift(M)))
xlabel('f')
ylabel('Amplitude')
title('Spectrum of the Message Signal')
subplot(243)
plot(f,abs(fftshift(C)))
xlabel('f')
```

AUST/EEE

```
ylabel('Amplitude')
title('Spectrum of the carrier Signal')
subplot(244)
plot(f,abs(fftshift(U)))
xlabel('f')
ylabel('Amplitude')
title('Spectrum of the Modulated Signal')
d = u.*c;  % Demodulation
[D,m,df1] = fftseq(d,ts,df);
D = D/fs;
subplot(245)
plot(f,abs(fftshift(D)));
xlabel('f')
ylabel('Amplitude')
title('Spectrum of the demodulated Signal')
f_cutoff = 100;
n_cutoff = floor(f_cutoff/df1);
H = zeros(size(f));
H(1:n_cutoff) = 2*ones(1,n_cutoff);
H(length(f)-n_cutoff +1:length(f)) = 2*ones(1,n_cutoff); %
Define requied Lowpass Filter
subplot(246)
plot(f*df,abs(fftshift(H)))
xlabel('f')
ylabel('Amplitude')
title('Spectrum of the Lowpass Filter')
DEM = D.*H;
subplot(247)
plot(f,abs(fftshift(DEM)))
xlabel('f')
ylabel('Amplitude')
title('Spectrum of the Reconstructed Message Signal')
dem = real(ifft(DEM))*fs;
subplot(248)
plot(t,dem(1:length(t)))
xlabel('t')
ylabel('Amplitude')
title('Reconstructed Message Signal')
```

AUST/EEE

Figure 7: Implementation of fft to demonstrate Amplitude Modulation

Problem: 1

Consider the following FDM system.

$$m_1(t) = \begin{cases} \sin c(100t) & |t| \le t_o \\ 0 & otherwise \end{cases} \qquad m_2(t) = \begin{cases} \sin c^2(100t) & |t| \le t_o \\ 0 & otherwise \end{cases}$$

Let carriers $c_1(t) = \cos(2\pi f c_1 t)$, where $fc_1 = 250$ Hz and $c_2(t) = \cos(2\pi f c_2 t)$, where $fc_2 = 750$ Hz are used to modulate and demodulate $m_1(t)$ and $m_2(t)$ respectively to transmit a noiseless transmission system as shown in Figure 8. Write a Matlab code obtain magnitude spectra at different stages of the system.

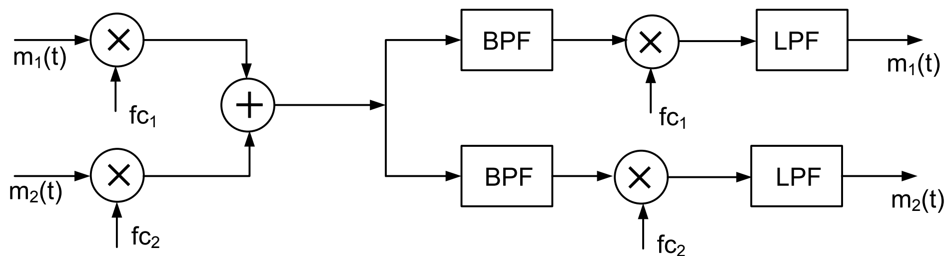


Figure 8: Simple FDM system.

FILTER SPECIFICATION

Figure 1 illustrates a typical design specification for an FIR lowpass filter. There is generally a certain amount of ripple in both passbands and stopbands. In Fig. 1, the maximum deviation (considered as an acceptable tolerance) from the average value in the passband is designated δ_P , while the deviation from zero in the stopband is designated as δ_S. This manner of defining the requirements for passband and stopband ripple is called an absolute specification; another manner (more common) is to specify the levels of ripple in decibels relative to the maximum magnitude of response, which is 1 + δ_P .

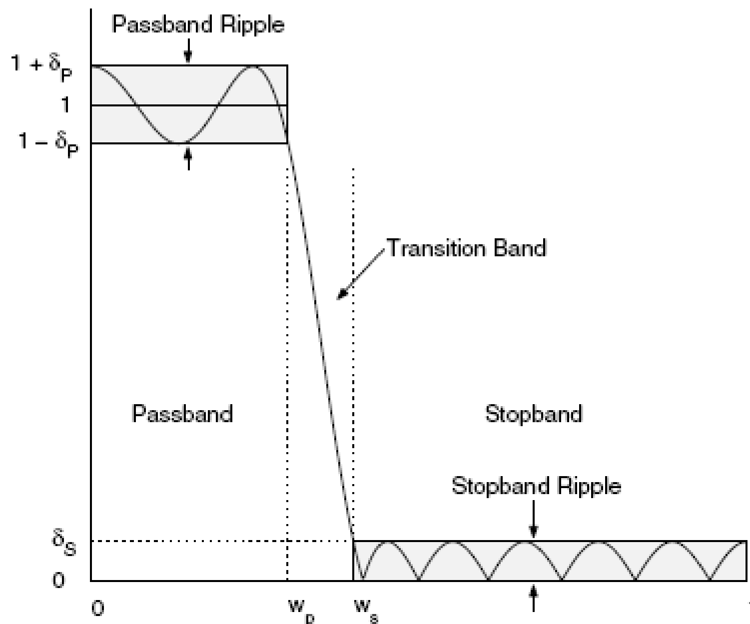


Figure 1: Design criteria for an FIR.

Figure 2 depicts another lowpass filter design specification in relative terms. The values of *R* and *A* are in decibels, and represent the passband ripple amplitude (or minimum passband response when the maximum filter/passband response is 0 db) and minimum stopband attenuation, respectively. The relationship between δ_P and δ_S and *R* and *A* are

$$R = -20 \log 10(\frac{1 - \delta_P}{1 + \delta_P})$$

and

$$A = -20 \log 10(\frac{\delta_S}{1 + \delta_P})$$

To determine δ_P when R and A are given, use

$$\delta_P = \frac{1 - 10^{-R/20}}{1 + 10^{-R/20}}$$

and

$$\delta_S = (1 + \delta_P)10^{-A/20}$$

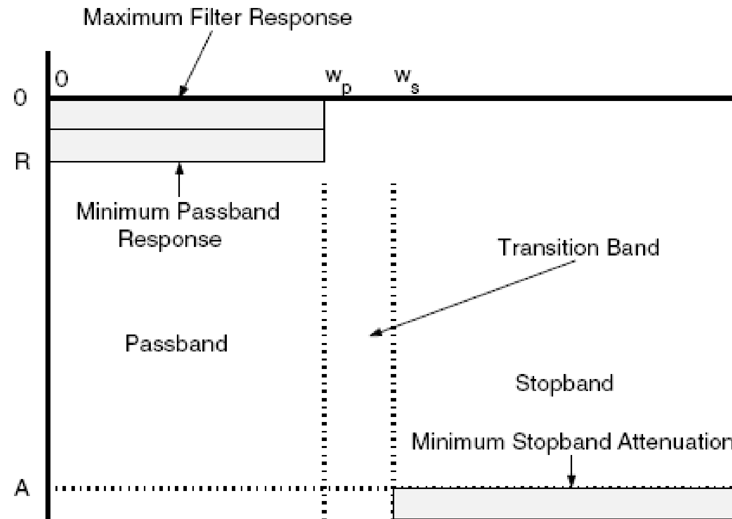


Figure 2: A relative filter design specification, with the (horizontal) frequency axis at the top, and (vertical) logarithmic amplitude (dB) axis at the left.

PROPERTIES OF LINEAR PHASE FIR FILTERS

Impulse response symmetry	*Number of coefficients N*	*Frequency response H(ω)*	*Type of linear phase*
Positive symmetry, $h(n) = h(N-1-n)$	Odd	$e^{-j\omega(N-1)/2} \sum_{n=0}^{(N-1)/2} a(n)\cos(\omega n)$	1
	Even	$e^{-j\omega(N-1)/2} \sum_{n=1}^{N/2} b(n)\cos[\omega(n-\frac{1}{2})]$	2
Negative symmetry, $h(n) = -h(N-1-n)$	Odd	$e^{-j[\omega(N-1)/2-\pi/2]} \sum_{n=1}^{(N-1)/2} a(n)\sin(\omega n)$	3
	Even	$e^{-j[\omega(N-1)/2-\pi/2]} \sum_{n=1}^{N/2} b(n)\sin[\omega(n-\frac{1}{2})]$	4

$a(0) = h[(N-1)/2]$; $a(n) = 2h[(N-1)/2 - n]$
$b(n) = 2h(N/2 - n)$

Four types of linear phase FIR filters

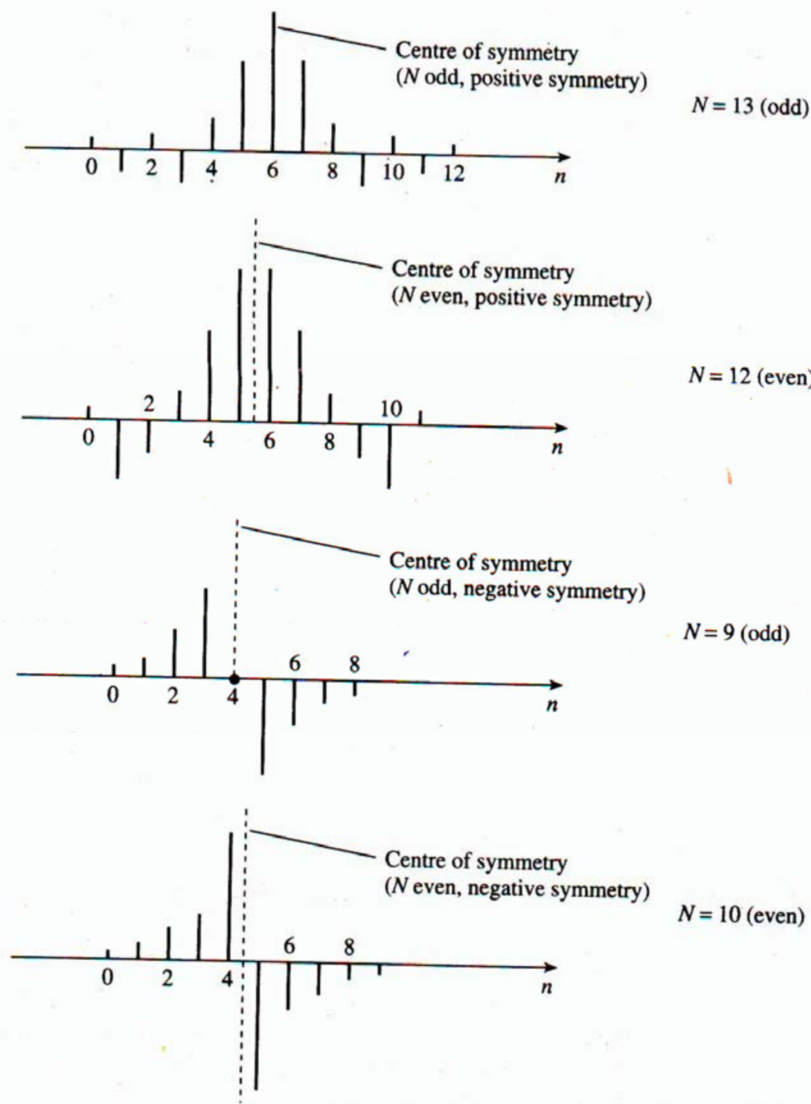


Figure: 3 Comparison of the impulse of the four types of Linear Phase Filters.

(a) Type 1 FIR Filter: Either an even number or no zeros at $z = 1$ and $z = -1$.

(b) Type 2 FIR Filter: Either an even number or no zeros at $z = 1$, and an odd number of zeros at $z = -1$.

(c) Type 3 FIR Filter: An odd number of zeros at $z = 1$ and $z = -1$.

(d) Type 4 FIR Filter: An odd number of zeros at $z = 1$, and either an even number or no zeros at $z = -1$.

The presence of zeros at $z = \pm 1$ leads to the following limitations on the use of these linear-phase FIR filters for designing filters. For example, since the Type 2 FIR filter always has a zero at $z = -1$, it cannot be used to design a highpass filter. Likewise, the Type 3 FIR filter has zeros at both $z = 1$ and $z = -1$ and, as a result, cannot be used to design either a lowpass or a highpass or a bandstop filter. Similarly, the Type 4 FIR filter is not appropriate to design a lowpass filter due to the presence of a zero at $z = 1$. Finally, the Type 1 FIR filter has no such restrictions and can be used to design almost any type of filter.

The MATLAB routine freqz computes the frequency response but we cannot determine the amplitude response from it because there is no function in MATLAB comparable to the abs function that can find amplitude. However, it easy to write simple routines to compute amplitude responses for each of the four types. We provide four functions to do this.

Example : 1

Let $h(n) = \{-4, \underset{\uparrow}{1}, -1, -2, 5, 6, 5, -2, -1, 1, -4\}$. Determine the amplitude response $H_r(\omega)$ and the locations of the zeros of $H(z)$.

1. Hr_type1:

```
function [Hr,w,a,L] = Hr_Type1(h);
% Computes Amplitude response Hr(w) of a Type-1 LP FIR filter
% -----------------------------------------------------------
% [Hr,w,a,L] = Hr_Type1(h)
% Hr = Amplitude Response
%  w = 500 frequencies between [0 pi] over which Hr is computed
%  a = Type-1 LP filter coefficients
%  L = Order of Hr
%  h = Type-1 LP filter impulse response
%
 M = length(h);
 L = (M-1)/2;
 a = [h(L+1) 2*h(L:-1:1)]; % 1x(L+1) row vector
 n = [0:1:L];              % (L+1)x1 column vector
 w = [0:1:500]'*pi/500;
Hr = cos(w*n)*a';
```

MATLAB Script

```
>> h = [-4,1,-1,-2,5,6,5,-2,-1,1,-4];
>> M = length(h); n = 0:M-1;
>> [Hr,w,a,L] = Hr_Type1(h);
>> a,L
a = 6    10    -4    -2    2    -8
L = 5
>> amax = max(a)+1; amin = min(a)-1;
>> subplot(2,2,1); stem(n,h); axis([-1 2*L+1 amin amax])
>> xlabel('n'); ylabel('h(n)'); title('Impulse Response')
>> subplot(2,2,3); stem(0:L,a); axis([-1 2*L+1 amin amax])
>> xlabel('n'); ylabel('a(n)'); title('a(n) coefficients')
>> subplot(2,2,2); plot(w/pi,Hr);grid
>> xlabel('frequency in pi units'); ylabel('Hr')
>> title('Type-1 Amplitude Response')
>> subplot(2,2,4); pzplotz(h,1)
```

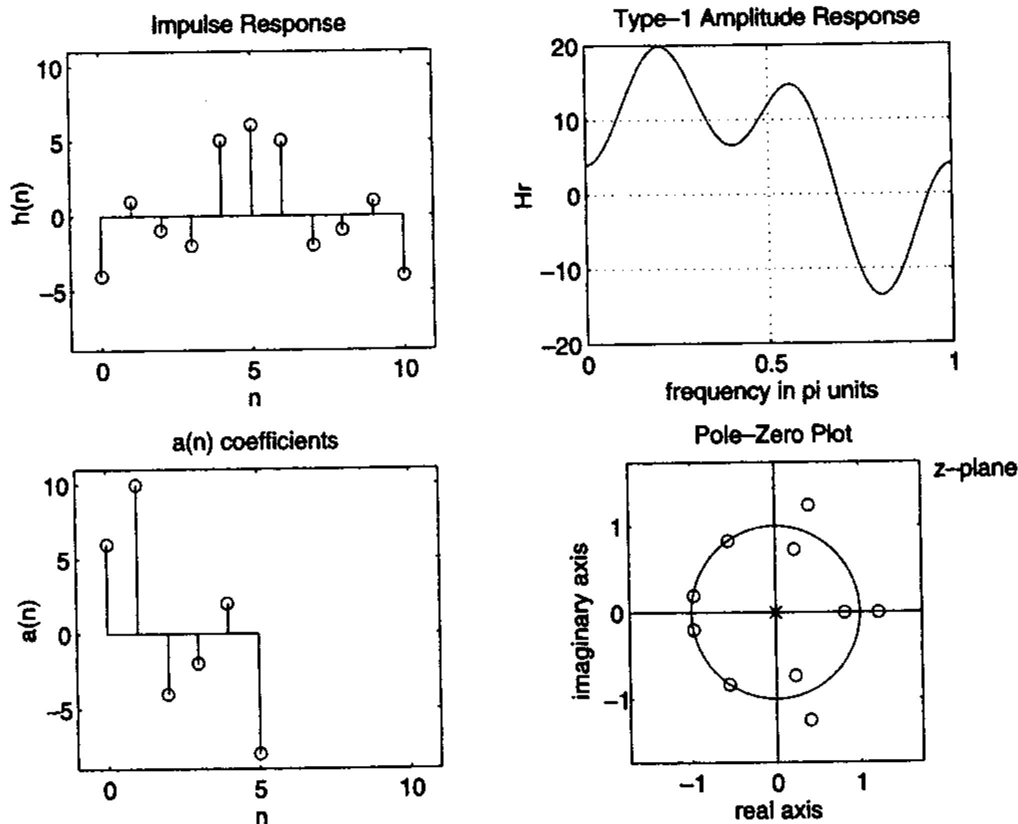


Figure: 3 Example 1

Example 2:

Let $h(n) = \{-4, \underset{\uparrow}{1}, -1, -2, 5, 6, 6, 5, -2, -1, 1, -4\}$. Determine the amplitude response $H_r(\omega)$ and the locations of the zeros of $H(z)$.

2. Hr_type2:

```
function [Hr,w,b,L] = Hr_Type2(h);
% Computes Amplitude response of a Type-2 LP FIR filter
% ---------------------------------------------------
% [Hr,w,b,L] = Hr_Type2(h)
% Hr = Amplitude Response
%  w = frequencies between [0 pi] over which Hr is computed
%  b = Type-2 LP filter coefficients
%  L = Order of Hr
%  h = Type-2 LP impulse response
%
 M = length(h);
 L = M/2;
 b = 2*[h(L:-1:1)];
 n = [1:1:L]; n = n-0.5;
 w = [0:1:500]'*pi/500;
Hr = cos(w*n)*b';
```

MATLAB Script

```
>> h = [-4,1,-1,-2,5,6,6,5,-2,-1,1,-4];
>> M = length(h); n = 0:M-1;
>> [Hr,w,a,L] = Hr_Type2(h);
>> b,L
b = 12    10    -4    -2    2    -8
L = 6
>> bmax = max(b)+1; bmin = min(b)-1;
>> subplot(2,2,1); stem(n,h); axis([-1 2*L+1 bmin bmax])
>> xlabel('n'); ylabel('h(n)'); title('Impulse Response')
>> subplot(2,2,3); stem(1:L,b); axis([-1 2*L+1 bmin bmax])
>> xlabel('n'); ylabel('b(n)'); title('b(n) coefficients')
>> subplot(2,2,2); plot(w/pi,Hr);grid
>> xlabel('frequency in pi units'); ylabel('Hr')
>> title('Type-1 Amplitude Response')
>> subplot(2,2,4); pzplotz(h,1)
```

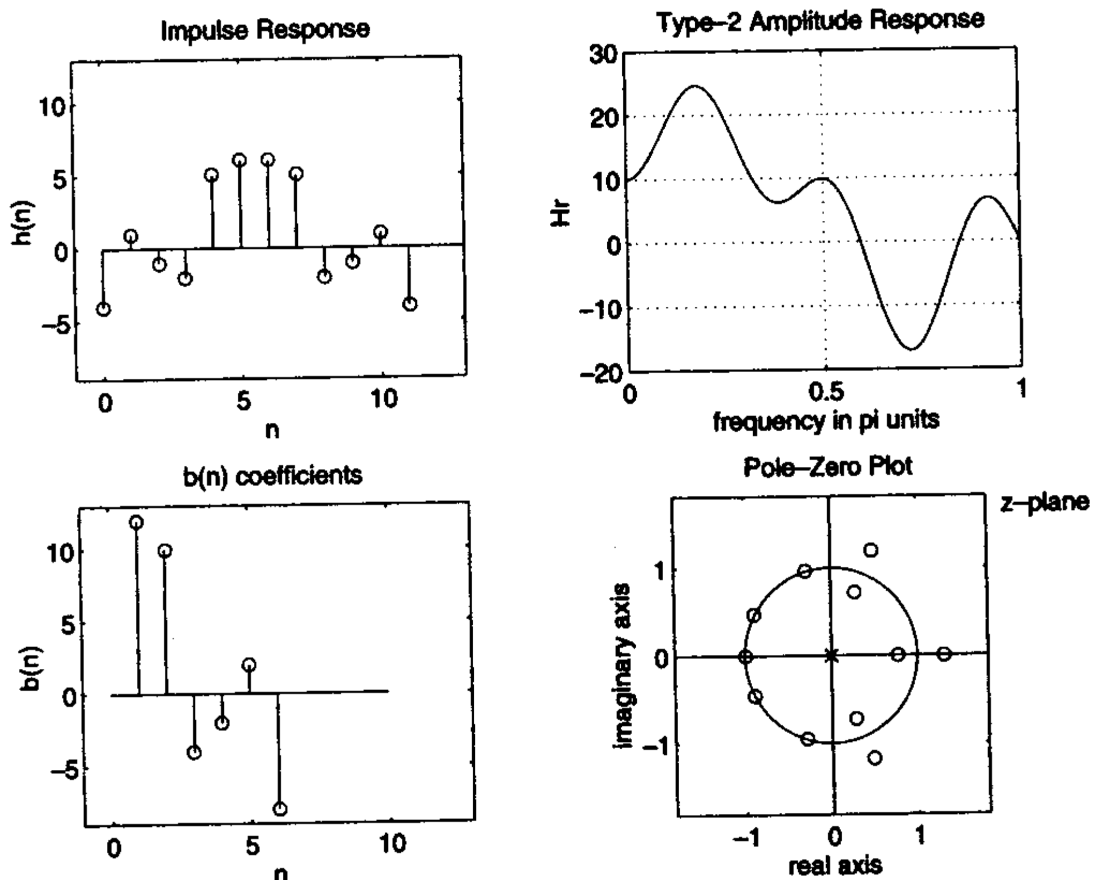


Figure: 4 Example 2

Example 3:

Let $h(n) = \{\underset{\uparrow}{-4}, 1, -1, -2, 5, 0, -5, 2, 1, -1, 4\}$. Determine the amplitude response $H_r(\omega)$ and the locations of the zeros of $H(z)$.

3. Hr_type3:

```
function [Hr,w,c,L] = Hr_Type3(h);
% Computes Amplitude response Hr(w) of a Type-3 LP FIR filter
% -----------------------------------------------------------
% [Hr,w,c,L] = Hr_Type3(h)
% Hr = Amplitude Response
%  w = frequencies between [0 pi] over which Hr is computed
%  c = Type-3 LP filter coefficients
%  L = Order of Hr
%  h = Type-3 LP impulse response
%
 M = length(h);
 L = (M-1)/2;
 c = [2*h(L+1:-1:1)];
 n = [0:1:L];
 w = [0:1:500]'*pi/500;
Hr = sin(w*n)*c';
```

MATLAB Script

```
>> h = [-4,1,-1,-2,5,0,-5,2,1,-1,4];
>> M = length(h); n = 0:M-1;
>> [Hr,w,c,L] = Hr_Type3(h);
>> c,L
a = 0    10    -4    -2    2    -8
L = 5
>> cmax = max(c)+1; cmin = min(c)-1;
>> subplot(2,2,1); stem(n,h); axis([-1 2*L+1 cmin cmax])
>> xlabel('n'); ylabel('h(n)'); title('Impulse Response')
>> subplot(2,2,3); stem(0:L,c); axis([-1 2*L+1 cmin cmax])
>> xlabel('n'); ylabel('c(n)'); title('c(n) coefficients')
>> subplot(2,2,2); plot(w/pi,Hr);grid
>> xlabel('frequency in pi units'); ylabel('Hr')
>> title('Type-1 Amplitude Response')
>> subplot(2,2,4); pzplotz(h,1)
```

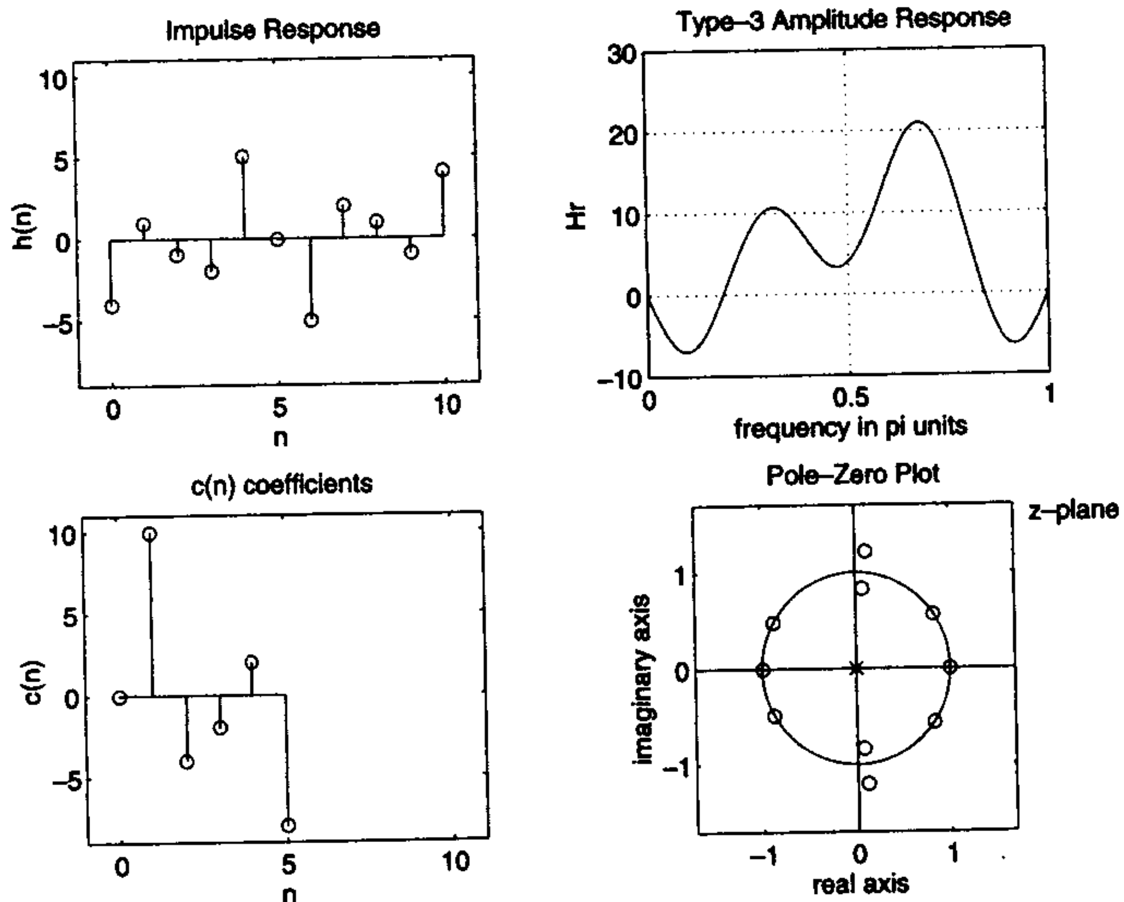


Figure: 5 Example 3

Example 4:

Let $h(n) = \{-4, \underset{\uparrow}{1}, -1, -2, 5, 0, -5, 2, 1, -1, 4\}$. Determine the amplitude response $H_r(\omega)$ and the locations of the zeros of $H(z)$.

4. <u>Hr_type4</u>:

```
function [Hr,w,d,L] = Hr_Type4(h);
% Computes Amplitude response of a Type-4 LP FIR filter
% -------------------------------------------------------
% [Hr,w,d,L] = Hr_Type4(h)
% Hr = Amplitude Response
%  w = frequencies between [0 pi] over which Hr is computed
%  d = Type-4 LP filter coefficients
%  L = Order of d
%  h = Type-4 LP impulse response
```

```
%
 M = length(h);
 L = M/2;
 d = 2*[h(L:-1:1)];
 n = [1:1:L]; n = n-0.5;
 w = [0:1:500]'*pi/500;
Hr = sin(w*n)*d';
```

MATLAB Script

```
>> h = [-4,1,-1,-2,5,6,-6,-5,2,1,-1,4];
>> M = length(h); n = 0:M-1;
>> [Hr,w,d,L] = Hr_Type4(h);
>> b,L
d = 12    10    -4    -2    2    -8
L = 6
>> dmax = max(d)+1; dmin = min(d)-1;
>> subplot(2,2,1); stem(n,h); axis([-1 2*L+1 dmin dmax])
>> xlabel('n'); ylabel('h(n)'); title('Impulse Response')
>> subplot(2,2,3); stem(1:L,d); axis([-1 2*L+1 dmin dmax])
>> xlabel('n'); ylabel('d(n)'); title('d(n) coefficients')
>> subplot(2,2,2); plot(w/pi,Hr);grid
>> xlabel('frequency in pi units'); ylabel('Hr')
>> title('Type-1 Amplitude Response')
>> subplot(2,2,4); pzplotz(h,1)
```

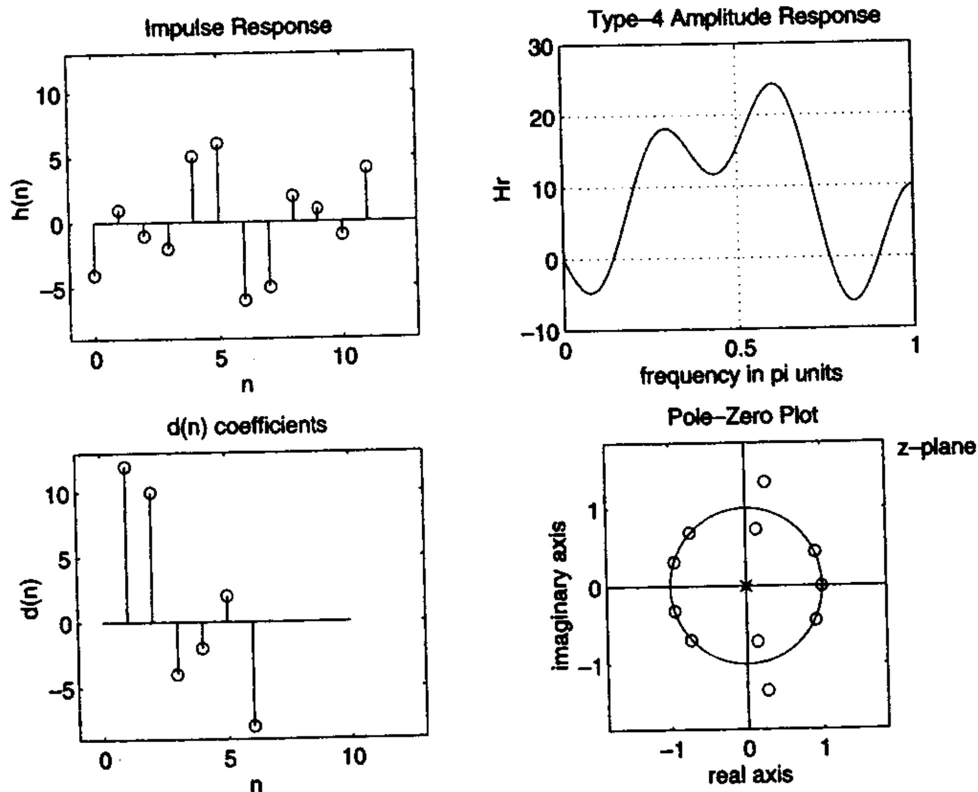


Figure: 6 Example 4

WINDOW BASED FIR FILTER DESIGN USING MATLAB

1. Specify the desired frequency response.
2. Select a window function and estimate the number of filter coefficients, N.
3. Obtain the ideal impulse response, $h_D(n)$ (truncated to N values).
4. Obtain N coefficients of the window function, $w(n)$.
5. Obtain the FIR filter coefficients by applying the window, $h(n) = h_D(n) \times w(n)$.

WINDOW FUNCTIONS FOR FIR FILTER DESIGN

Name of window	Time-domain sequence, $h(n), 0 \le n \le M-1$
Bartlett (triangular)	$1 - \dfrac{2\left\lvert n - \frac{M-1}{2}\right\rvert}{M-1}$
Blackman	$0.42 - 0.5\cos\dfrac{2\pi n}{M-1} + 0.08\cos\dfrac{4\pi n}{M-1}$
Hamming	$0.54 - 0.46\cos\dfrac{2\pi n}{M-1}$
Hanning	$\frac{1}{2}\left(1 - \cos\dfrac{2\pi n}{M-1}\right)$
Kaiser	$\dfrac{I_0\left[\beta\sqrt{\left(\frac{M-1}{2}\right)^2 - \left(n - \frac{M-1}{2}\right)^2}\right]}{I_0\left[\beta\left(\frac{M-1}{2}\right)\right]}$
Lanczos	$\left\{\dfrac{\sin\left[2\pi\left(n - \frac{M-1}{2}\right)\Big/(M-1)\right]}{2\pi\left(n-\frac{M-1}{2}\right)\Big/\left(\frac{M-1}{2}\right)}\right\}^L \quad L > 0$
Tukey	$1, \left\lvert n - \frac{M-1}{2}\right\rvert \le \alpha\frac{M-1}{2} \quad 0 < \alpha < 1$ $\frac{1}{2}\left[1 + \cos\left(\dfrac{n-(1+a)(M-1)/2}{(1-\alpha)(M-1)/2}\pi\right)\right]$ $\alpha(M-1)/2 \le \left\lvert n - \frac{M-1}{2}\right\rvert \le \frac{M-1}{2}$

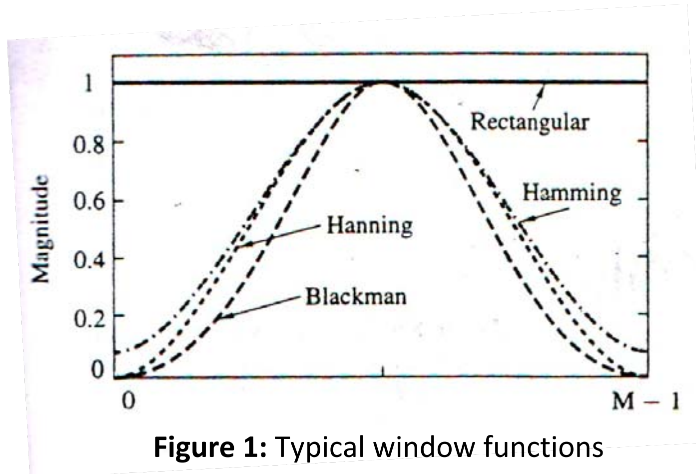


Figure 1: Typical window functions

The Signal processing toolbox of MATLAB includes the following functions for generating the windows.

w = blackman(L)	w = hamming(L)	w = hann(L)	w = chebwin(L,Rs)
w = kaiser(L,beta)	w = bartlett(L)	etc.	

The above functions generate a vector *w* of window coefficients of odd length *L*.

Type “>> help window” in the command prompt to get more help of window function.

Example : 1

Illustrating FIR coefficient calculation using the Kaiser window Determine the coefficients and plot the magnitude-frequency response of a bandpass FIR filter, using the Kaiser window and MATLAB, that meets the following specifications:

passband	150-250 Hz
transition width	50 Hz
passband ripple	0.1 dB
stopband attenuation	60 dB
sampling frequency	1 kHz

Solution :

When β = 0, the Kaiser window corresponds to the rectangular window, and when, it is 5.44, the resulting window is very similar, though not identical, to the Hamming window. The value of β is determined by the stopband attenuation requirements and may be estimated from one of the following empirical relationships:

$$\beta = 0 \qquad \text{if } A \leqslant 21\text{ dB}$$
$$\beta = 0.5842(A-21)^{0.4} + 0.078\,86(A-21) \qquad \text{if } 21\text{ dB} < A < 50\text{ dB}$$
$$\beta = 0.1102(A-8.7) \qquad \text{if } A \geqslant 50\text{ dB}$$

where $A = -20\,log_{10}(\delta)$ is the stopband attenuation, ($\delta = \min(\delta_p, \delta_s)$), since passband and stopband ripples are nearly equal, δ_P is the desired passband ripple δ_S is the desired stopband ripple. The number of filter coefficients, *N,* is given by

$$N \geqslant \frac{A - 7.95}{14.36\,\Delta f}$$

AUST/EEE

where Δf is the normalized transition width. The values of β and N are used to compute the coefficients for the Kaiser window w(n).

```
clear all;
echo on;
FS=1000; %  Sampling frequency
FN=FS/2;   %    Nyquist frequency
N=73;   %   Filter length
beta=5.65;  % Kaiser window Ripple parameter
fc1=125/FN; % Normalized cut off frequencies
fc2=275/FN;
FC=[fc1 fc2];   % Band edge frequency vector
hn=fir1(N-1, FC, kaiser(N, beta)); % Obtain windowed filter coefficients
[H, f]=freqz(hn, 1, 512, FS);
mag=20*log10(abs(H));  %  Compute frequency response
plot (f, mag), grid on
xlabel (`Frequency (Hz)')
ylabel ('Magnitude Response (dB)')
```

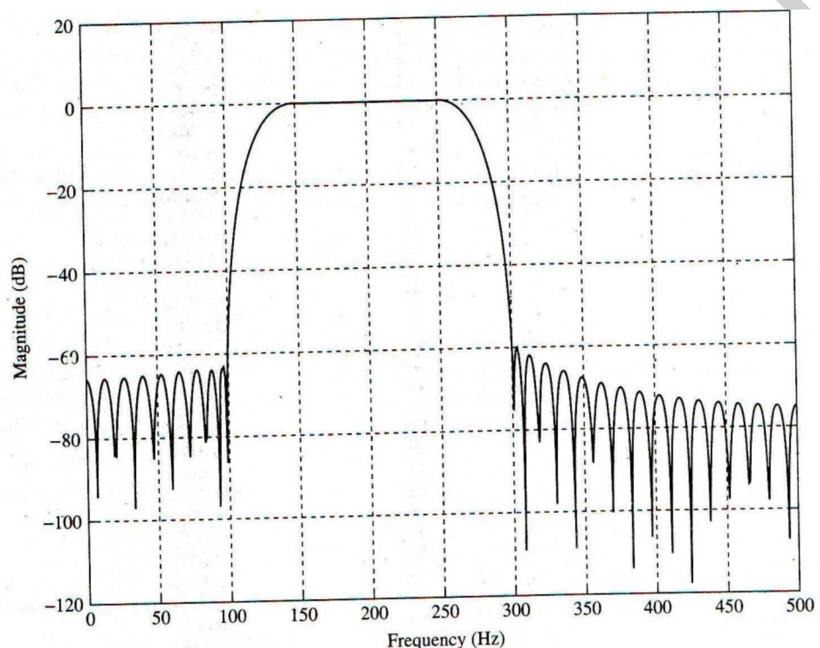


Figure 2: Example 1

DESIGN STAGES FOR DIGITAL IIR FILTERS

The design of IIR filters can be conveniently broken down into five main stages.

- Filter specification, at which stage the designer gives the function of the filter (for example, lowpass) and the desired performance.
- Approximation or coefficient calculation, where we select one of a number of methods and calculate the values of the coefficients, b_k, and a_k, in the transfer function, H(z), such that the specifications given in stage 1 are satisfied.
- Realization, which is simply converting the transfer function into a suitable filter structure. Typical structures for IIR filters are parallel and cascade of second and/or first-

order filter sections.

- Analysis of errors that would arise from representing the filter coefficients and carrying out the arithmetic operations involved in filtering with only a finite number of bits.
- Implementation, which involves building the hardware and/or writing the software codes, and carrying out the actual filtering operation.

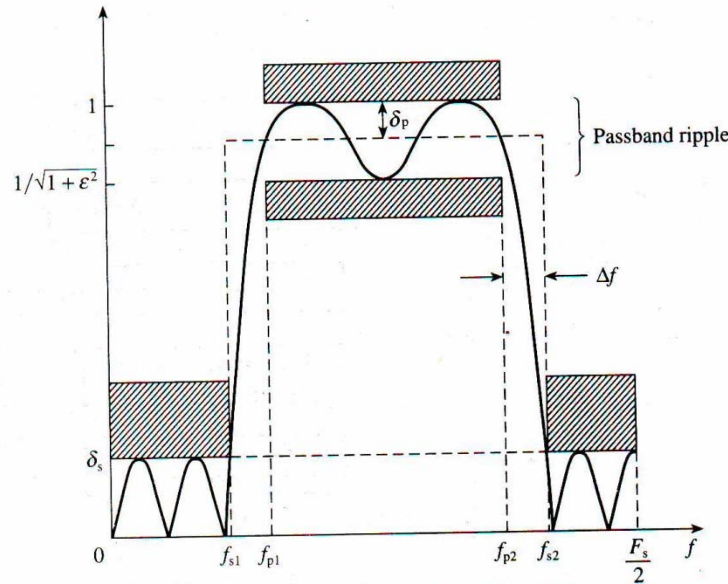


Figure 3 Tolerance scheme for an IIR bandpass filter.

ε^2	passband ripple parameter
δ_p	passband deviation
δ_s	stopband deviation
f_{p1} and f_{p2}	passband edge frequencies
f_{s1} and f_{s2}	stopband edge frequencies

The bandedge frequencies are sometimes given in normalized form, that is a fraction of the sampling frequency (*f/F_S*), but we shall specify them in standard frequency units of hertz or kilohertz as these are less confusing, especially to the inexperienced designer. Passband and stopband deviations may be expressed as ordinary numbers or in decibels: the passband ripple in decibels is

$$A_p = 10\log_{10}(1+\varepsilon^2) = -20\log_{10}(1-\delta_p)$$

and the stopband attenuation in decibel is

$$A_s = -20\log_{10}(\delta_s)$$

COEFFICIENT CALCULATION METHODS FOR IIR FILTERS:

- Pole-zero replacement
- Impulse invariant
- Matched z-transform

- Biliear z-transform

Example : 2

The halfwave rectified sine can be represented by the trigonometric Fourier series

$$f(t) = \frac{A}{\pi} + \frac{A}{2}\sin t - \frac{A}{\pi}\left[\frac{\cos 2t}{3} + \frac{\cos 4t}{15} + \frac{\cos 6t}{35} + \frac{\cos 8t}{63} + \dots\right]$$

we want to filter out just the first *2* terms. To simplify this expression, we let $A = 3\pi$ and we truncate it by eliminating all cosine terms except *cos2t* the term. Then,

$$g(t) = 3 + 1.5\sin t - \cos 2t$$

The problem now reduces to design a lowpass digital filter, and use the **filter** command to remove the higher order cosine terms.

Solution:
We will use a *6* pole digital lowpass Butterworth filter because we must have a sharp transition between the *1rad/s* and *2rad/s* frequency range. Also, since the highest frequency component is *2rad/s*, to avoid aliasing, we must specify a sampling frequency of $\omega_s = 4rad/s$. Thus, the sampling frequency must be $f_s = \omega_s/2\pi = 2/\pi$ and therefore, the sampling period will be $T_s = 1/f_s = \pi/2$. We choose $T_s = 0.5$; this is sufficiently small. Also, we choose the cutoff frequency of the filter to be $\omega_c = 1.5rad/s$ in order to attenuate the cosine terms.

The MATLAB script below will perform the following steps:

- Will compute coefficients of the numerator and denominator of the transfer function with normalized cutoff frequency.
- Will recompute the coefficients for the desired frequency.
- Use the **bilinear** function to map the analog transfer function to a digital transfer function, and will plot the frequency response of the digital filter.
- Will recompute the digital filter transfer function to account for the warping effect.
- Will use the **filter** function to remove the cosine terms

```
% Step 1
%
clear all;

[z,p,k]=buttap(6);
[b,a]=zp2tf(z,p,k);
%
% Step 2
%
wc=1.5; % Chosen cutoff frequency
[b1,a1]=lp2lp(b,a,wc); % Convert to actual cutoff frequency
%
% Step 3
%
T=0.5; % Define sampling period
```

AUST/EEE

```
[Nz,Dz]= bilinear(b1,a1,1/T); % Map to digital filter using the bilinear
%transformation
w =0:2*pi/300:pi; % Define range for plot
Gz=freqz(Nz,Dz,w); % The digital filter transfer function
%
clf;
%
plot(w,abs(Gz)); axis([0 2 0 1]); grid; hold on;
% We must remember that when z is used as a function of
% normalized frequency, the range of frequencies of G(z) are
% from zero to pi and the normalized cutoff frequency on the
% plot is wc*T=1.5*0.5=0.75 r/s
%
xlabel('Radian Frequency w in rads/sec'),
ylabel('Magnitude of G(z)'),
title('Digital Filter Response in Normalized Frequency');
%
fprintf('Press any key to continue \n');
%
pause;
%
% Step 4
%
p=6; T=0.5; % Number of poles and Sampling period
wc=1.5; % Analog cutoff frequency in rad/sec
wd=wc*T/pi; % Normalized digital filter cutoff frequency
[Nzp,Dzp]=butter(p,wd);
fprintf('Summary: \n\n');
fprintf('WITHOUT PREWARPING: \n\n');
%
fprintf('The num N(z) coefficients in descending order of z are: \n\n');
fprintf('%8.4f \t',[Nz]);
fprintf('\n\n');
fprintf('The den D(z) coefficients in descending order of z are: \n\n');
fprintf('%8.4f \t',[Dz]);
fprintf('\n\n');
fprintf('WITH PREWARPING: \n\n');
%
fprintf('The num N(z) coefficients in descending order of z are: \n\n');
fprintf('%8.4f \t',[Nzp]);
fprintf('\n\n');...
fprintf('The den D(z) coefficients in descending order of z are: \n\n');
fprintf('%8.4f \t',[Dzp]);
fprintf('\n\n');

% Step 5
%
n=0:150;
T=0.5;
gt=3+1.5*sin(n*T)-cos(2*n*T);
yt=filter(Nzp,Dzp,gt);
%
% We will plot the unfiltered analog signal gta
%
t=0:0.1:12;
gta=3+1.5*sin(t)-cos(2*t);
subplot(211), plot(t,gta), axis([0,12, 0, 6]); hold on;
```

AUST/EEE

```
xlabel('Continuous Time t'); ylabel('Function g(t)');
%
% We will plot the filtered analog signal y(t)
%
subplot(212), plot(n*T,yt), axis([0,12, 0, 6]); hold on;
xlabel('Continuous Time t'); ylabel('Filtered Output y(t)');
%
fprintf('Press any key to continue \n'); pause;
%
% We will plot the unfiltered discrete time signal g(n*T
%
subplot(211), stem(n*T,gt), axis([0,12, 0, 6]); hold on;
xlabel('Discrete Time nT'); ylabel('Discrete Function g(n*T)');
%
% We will plot the filtered discrete time signal y(n*T)
subplot(212), stem(n*T,yt), axis([0,12, 0, 6]); hold on;
xlabel('Discrete Time nT'); ylabel('Filtered Output y(n*T)')
```

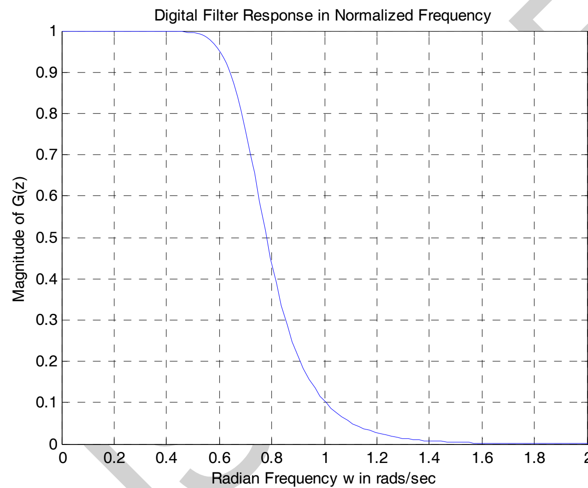


Figure 4: Digital Filter Response in Normalized Frequency

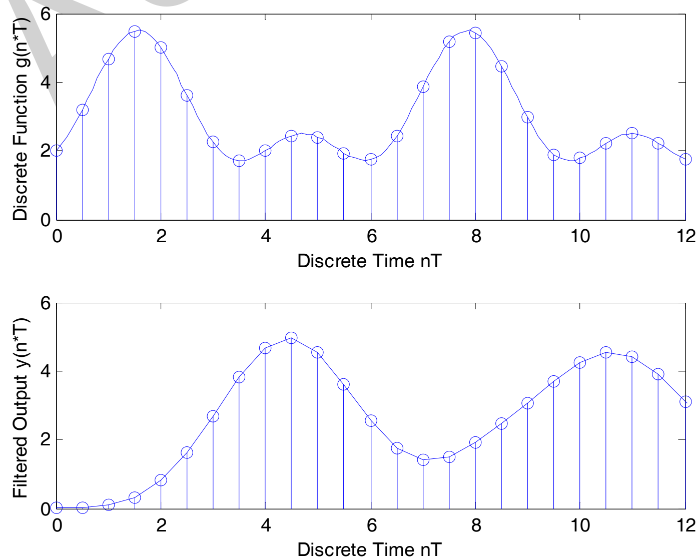


Figure 5: Discrete time input and output waveforms

Problem 1:
Use the MATLAB commands **cheb1ap** and **lp2hp** to derive the transfer function of a 3 pole Chebyshev Type I highpass analog filter with cutoff frequency f_c = 5kHz.

Problem 2:
Use the MATLAB **bilinear** function to derive the lowpass digital filter transfer function G(z) from a second-order Butterworth analog filter with 3dB a cutoff frequency at 50Hz, and sampling rate 500Hz.

Problem 3:
A lowpass, discrete-time filter, with Butterworth characteristics, is required to meet the following specifications:

cutoff frequency	300 Hz
filter order	5
sampling frequency	1000 Hz

(1) With the aid of MATLAB, obtain and plot
  (a) the magnitude-frequency and group delay responses of the filter using the impulse invariant method;
  (b) the magnitude-frequency and group delay responses of the filter using the bilinear z-transform method.
(2) Compare the two methods (bilinear z-transform and impulse invariant invariant method) in terms of the magnitude response distortions due to the Nyquist effect.

AUST/EEE

Problem 4:
Design of a bandpass filter with specified bandedge frequencies and pass- and stopband ripples
A requirement exists for a bandpass digital filter, with a Butterworth magnitude-frequency response, that satisfies the following specifications:

lower passband edge frequency	200 Hz
upper passband edge frequency	300 Hz
lower stopband edge frequency	50 Hz
upper stopband edge frequency	450 Hz
passband ripple	3 dB
stopband attenuation	20 dB
sampling frequency	1 kHz

(1) Determine, using the BZT method and MATLAB:
  (i) the order, N, of the filter;
  (ii) the poles, zeros, gain, coefficients and transfer function of the discrete-time filter.
(2) Plot the magnitude-frequency response and the pole-zero diagram of the filter.

LAB-8

MOVING AVERAGE FILTER MATLAB PROGRAMS

The mean and weighted average filter operation can be achieved by “conv.m” MATLAB built-in function.

Example : 1

```
r = conv(y, [1, 1, 1]/3)
```

gives the 3-point mean filter.

```
r = conv(y, [0.25, 0.5, 0.25])
```

gives 3-point weighted average filter.

Solution:

```
clear all;
echo on;
x = linspace(1, 10, 100);
f = 5 + x.*sin(x); %signal
y = f + randn(1, 100).*2; %Gaussian noise N(0, 22) added
subplot(321)
plot(x, f);
title('Original signal')
subplot(322)
plot(x, y);
title('Noisy signal')
rf1 = conv(y, [0.25, 0.5, 0.25]); %3-point weighted average
size(rf1) %note length of rf is NOT 100
rf1 = rf1(2:101); %middle 100 points give the solution
subplot(323)
plot(x, rf1);
title('Filtered with 3-point weighted average')
rf2 = conv(y, ones(1, 3)/3);
subplot(324)
plot(x, rf2(2:101));
title('Filtered with 3-point moving average')
rf3 = conv(y, ones(1, 7)/7);
subplot(3,2,[5:6])
plot(x, rf3(4:103)); %note the length of rf3
title('Filtered with 7-point moving average')
```

AUST/EEE

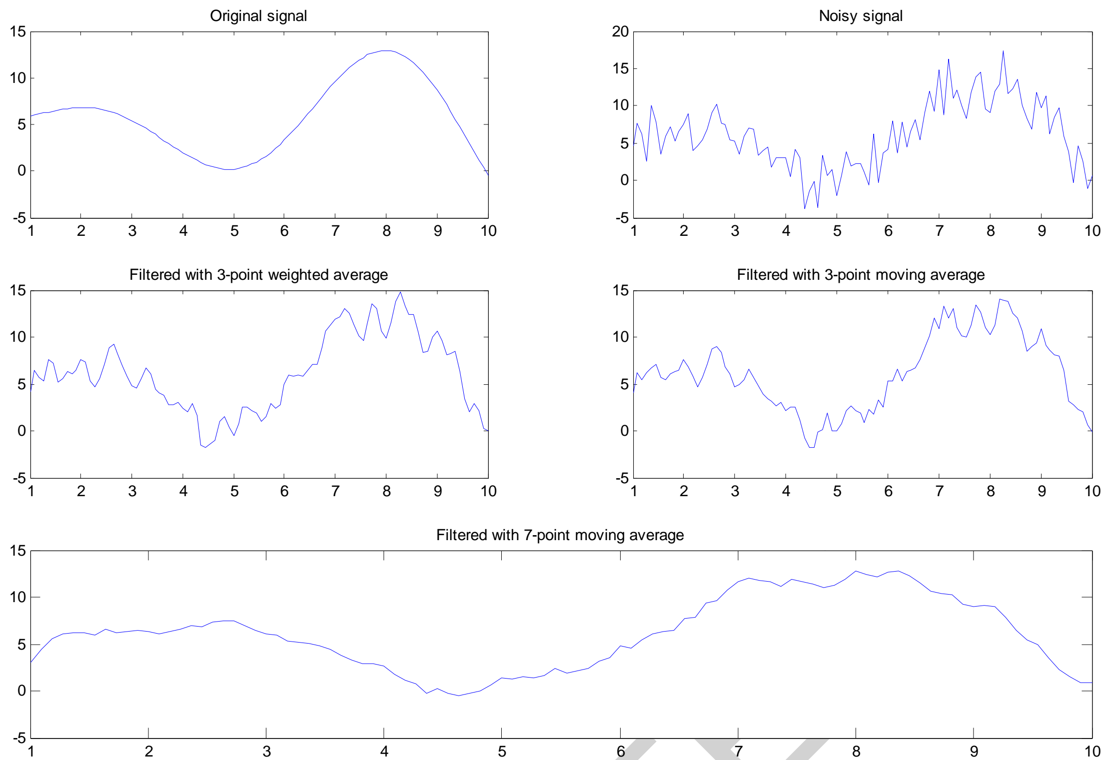


Figure 1: Example 1

AUST/EE

Example: 2

Moving average using filter.m MATLAB built-in function

```
clear all;
t=0:.01:1;
f=5;
y=sin(2*pi*f*t);
%Generation of random signal
g=0.5*randn(size(t));
z=g+y;
N=10; %order required
b=1/N*(ones(1,N));
x=filter(b,1,z); %filters noice
subplot(3,1,1);
plot(t,y);
ylabel('pure signal');
subplot(3,1,2);
plot(t,z);
ylabel('noise buried');
subplot(3,1,3);
plot(t,x);
ylabel('filtered signal');
xlabel('Time in seconds');
```

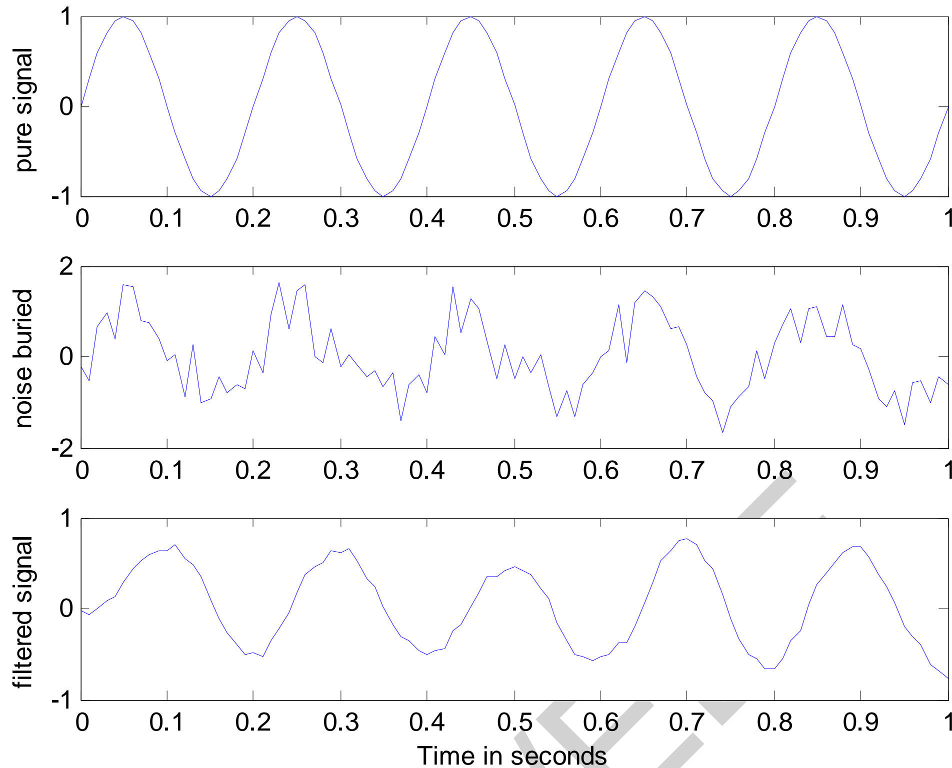


Figure 2: Example 2

Median filter

Example 3 :
3-point and 5-point median filters are given by

$$\hat{f}_i = \text{median}(y_{i-1}, y_i, y_{i+1})$$

and

$$\hat{f}_i = \text{median}(y_{i-2}, y_{i-1}, y_i, y_{i+1}, y_{i+2})$$

respectively.
We may use “sm1d.m” to compute median filter values.

```
function    y = sm1d(x, window, choice)
%SM1D       Smooth a 1d signal. You can have the choice of mean
% or median smoothing, and also you can control the smoothing %
window size.USAGE      sm1d(x, h, choice), where
%       x = the signal data
%       window = window width, eg 3 means span 3 smoothing.
%               choice = 'mean' if mean smoothing
%              = 'median' if median smoothing
n = length(x);
h = floor((window - 1)/2);
y = x;
```

```
if strcmp(choice, 'mean')
  for i = 1:n
    bg = max(1, i-h);
    ed = min(n, i+h);
    y(i) = mean(x(bg:ed));
  end
elseif strcmp(choice, 'median')
  for i = 1:n
    bg = max(1, i-h);
    ed = min(n, i+h);
    y(i) = median(x(bg:ed));
  end
else
  error('Wrong smoothing method');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
echo on;
x = linspace(1, 10, 100);
f = 5 + x.*sin(x); %signal
y = f + randn(1, 100).*2; %Gaussian noise N(0, 22) added
subplot(311)
plot(x, f);
title('Original signal')
subplot(312)
plot(x, y);
title('Noisy signal')
mf = sm1d(y, 5, 'median'); %5-point median
subplot(313)
plot(x, mf);
title('filtered signal');
```

AUST/EEE

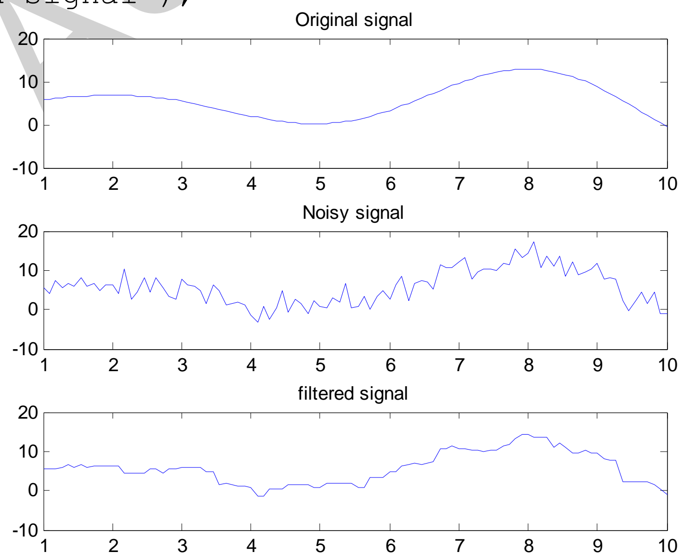


Figure 3: Example 3

Comb Filters

The basic comb filter is given by

$$H(z) = 1 - r^L z^{-L}$$

which has *L* equally-spaced zeros or poles at the radius *r*. In the diagram below, *r* is very close to the unit circle.

```
function [b,a] = COMB(r,L)
%  COMB
%      [b,a] = COMB(r,L)
%    generates L zeros equally spaced around a circle of radius r
b = [1 zeros(1,L-1) -r^L];
a = [1 zeros(1,L-1)];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
L = 16;
r = 0.995
[b,a] = comb(r,L);
[z,p,k] = tf2zp(1,b); % IIR comb
subplot(311)
zplane(z,p)

 % FIR Comb
[h,w] = freqz(1,b);
subplot(312)
plot(w/pi,abs(h));
xlabel ('Normalized frequency (Nyquist==1)')
ylabel ('Magnitude Response')
title('IIR Comb')

% IIR comb
[h,w] = freqz(b,1);
subplot(313)
plot(w/pi,abs(h));
ylabel 'Magnitude Response'
xlabel 'Normalized frequency (Nyquist==1)'
title('FIR Comb')
```

AUST/EEE

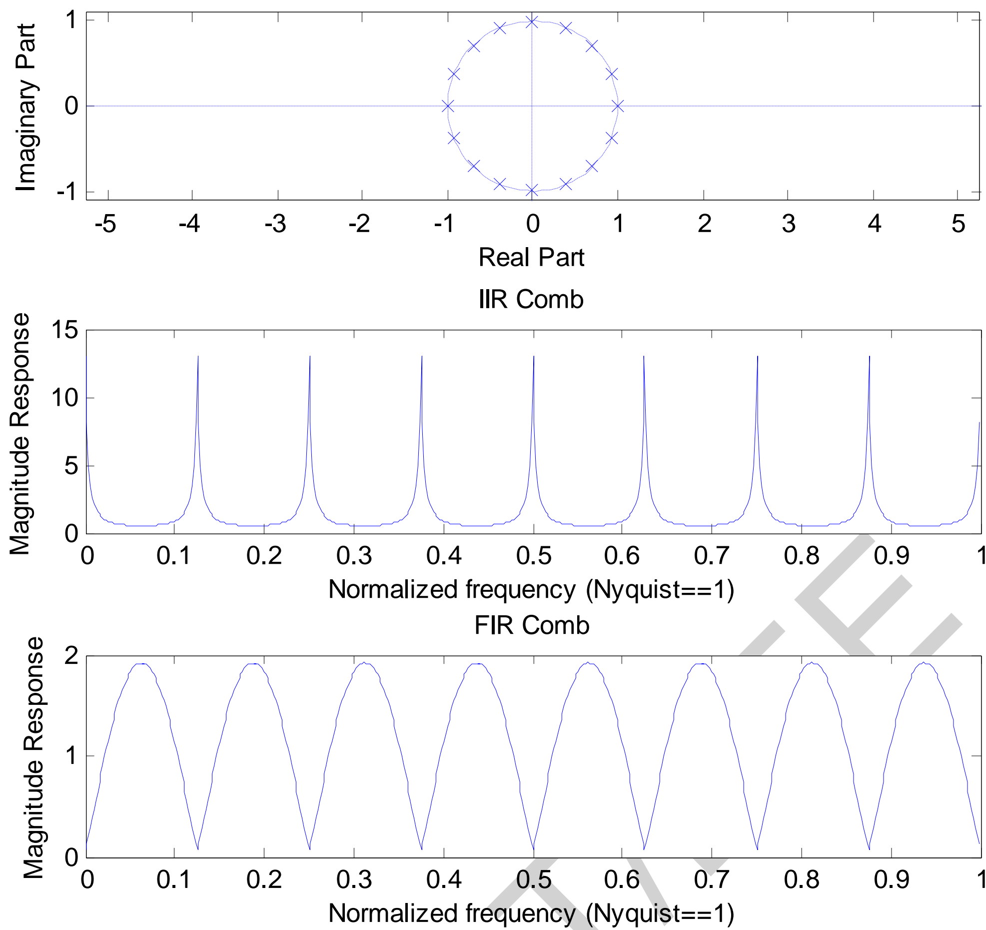


Figure 4: Comb Filter

Equivalence of Moving Average and CIC filter

Understanding on the cascaded integrator comb (CIC) filter

For understanding the cascaded integrator comb (CIC) filter, firstly let us understand the moving average filter, which is accumulation latest N samples of an input sequence x(n).

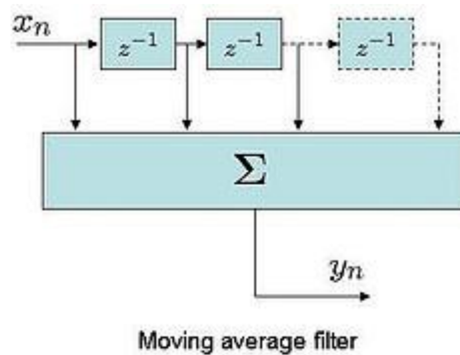


Figure 5: Moving average filter

AUST-EE

The frequency response of the moving average filter is:

$$H_f = \left| \frac{\sin(\pi f N)}{\sin(\pi f)} \right|.$$

```
clear all;
N = 10;
xn = sin(2*pi*[0:.1:10]);
hn = ones(1,N);
y1n = conv(xn,hn);
% transfer function of Moving Average filter
hF = fft(hn,1024);
plot([-512:511]/1024, abs(fftshift(hF)));
xlabel('Normalized frequency')
ylabel('Amplitude')
title('Frequency response of Moving average filter')
```

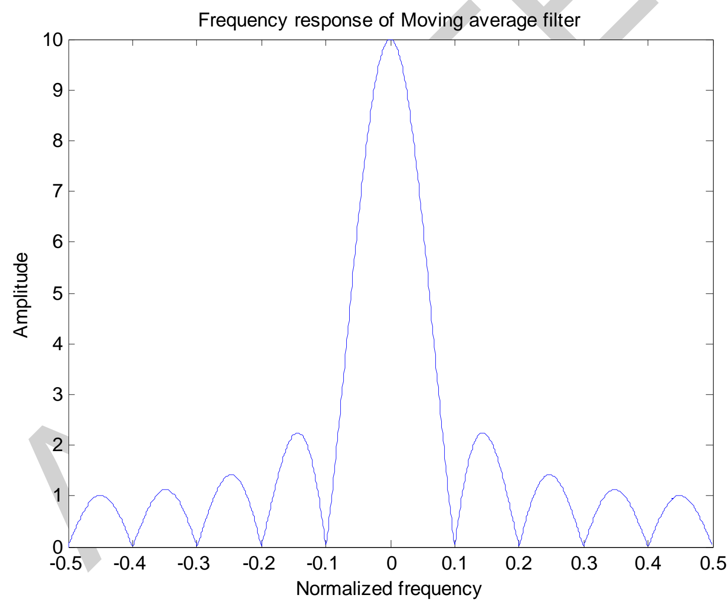


Figure 6: Frequency response of moving average filter