Digital Signal Processing

Part 3

Discrete-Time Signals & Systems
Case Studies

S R Taghizadeh <*srt@unl.ac.uk*>

January 2000

UNIVERSITY OF
NORTH LONDON

Introduction

Matlab and its applications in analysis of continuous-time signals and systems has been discussed in part 1 and 2 of this series of practical manuals. The purpose of part 3 is to discuss the way Matlab is used in analysis of discrete-time signals and systems. Each section provides a series of worked examples followed by a number of investigative problems. You are required to perform each of the worked examples in order to get familiar to the concept of Matlab environment and its important functions. In order to test your understanding of the concept of discrete-time signals & systems analysis, you are required to complete as many of the investigation / case study problems as possible. The areas covered are designed to enforce some of the topics covered in the formal lecture classes. These are:

- Signal Generation and Presentation
- Discrete Fourier Transform
- Spectral analysis
- Autocorrelation and Cross correlation
- Time delay estimation

Signal Generation and Manipulation

Sinusoidal Signal Generation

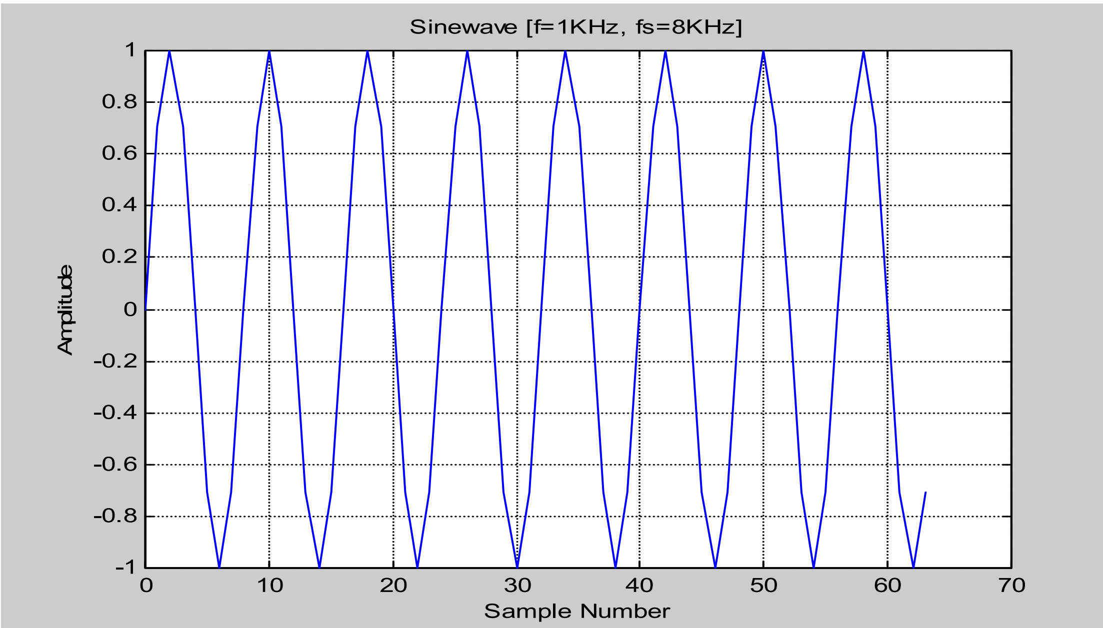
Consider generating 64 samples of a sinusoidal signal of frequency 1KHz, with a sampling frequency of 8KHz.

A sampled sinusoid may be written as:

$$x(n) = A\sin(2\pi \frac{f}{f_s} n + \vartheta)$$

where f is the signal frequency, f_s is the sampling frequency, θ is the phase and A is the amplitude of the signal. The program and its output is shown below:

```
% Program: W2E1b.m
% Generating 64 samples of x(t)=sin(2*pi*f*t) with a
% Frequency of 1KHz, and sampling frequency of 8KHz.
N=64;                  % Define Number of samples
n=0:N-1;               % Define vector n=0,1,2,3,...62,63
f=1000;                % Define the frequency
fs=8000;               % Define the sampling frequency
x=sin(2*pi*(f/fs)*n);  % Generate x(t)
plot(n,x);             % Plot x(t) vs. t
grid;
title('Sinewave [f=1KHz, fs=8KHz]');
xlabel('Sample Number');
ylabel('Amplitude');
```



Note that there are 64 samples with sampling frequency of 8000Hz or sampling time of 0.125 mS (i.e. 1/8000). Hence the record length of the signal is 64x0.125=8mS. There are exactly 8 cycles of sinewave, indicating that the period of one cycle is 1mS which means that the signal frequency is 1KHz.

Task: Generate the following signals

(i) 64 Samples of a cosine wave of frequency 25Hz , sampling frequency 400Hz, amplitude of 1.5 volts and phase =0.

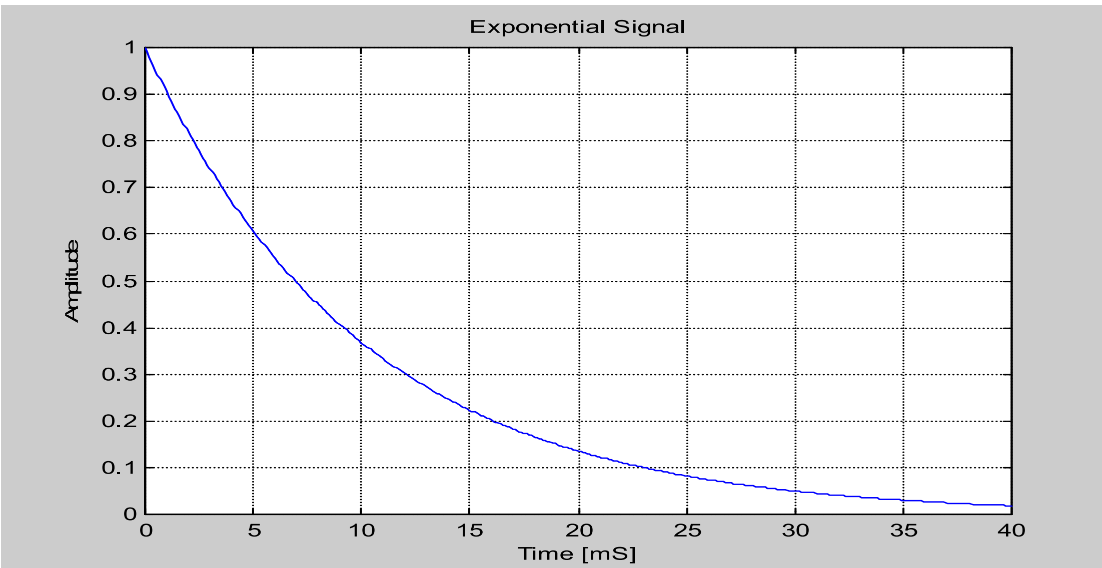
(ii) The same signal as in (i) but with a phase angle of π/4 (i.e. 45°).

Exercise 2: Exponential Signal Generation

Generating the signal $x(t) = e^{-0.1t} \quad for \quad t = 0 \text{ to } 40mS \text{ in steps of } 0.1mS$

```
% Program W2E2.m
% Generating the signal x(t)=exp(-0.1t)
t=0:0.1:40;
x=exp(-0.1*t);
plot(t,x);
grid;
title('Exponential Signal');
xlabel('Time [mS]');
ylabel('Amplitude');
```

And the output is:

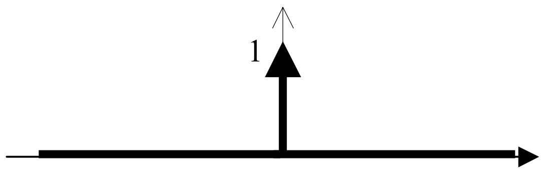


Task: Generate the signal: $x(t) = e^{-0.1t}\sin(0.6t)$ for t = 0 to 40ms in steps of 0.1 Sec

Exercise 3: Unit Impulse Signal Generation

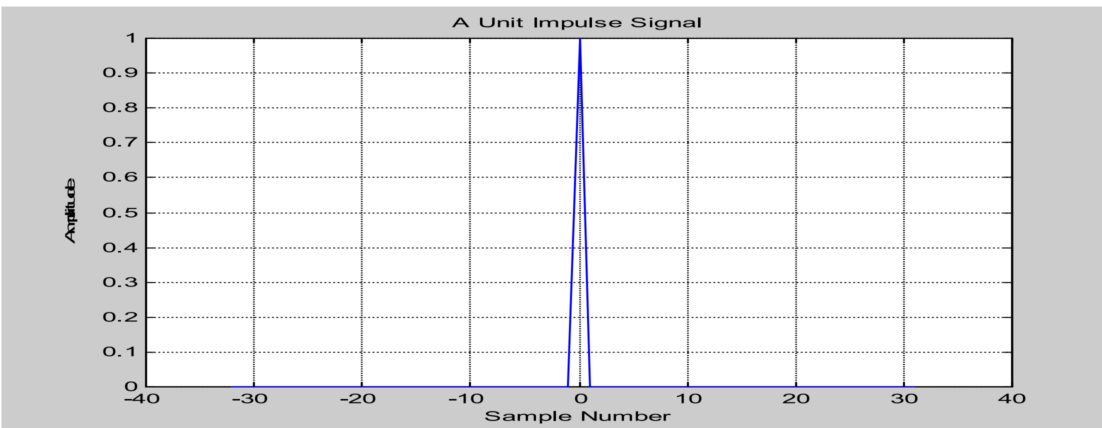
An impulse is defined as follows:

$$\delta(n) = \begin{cases} 1 & n = 0 \\ 0 & elsewhere \end{cases}$$



The following Matlab program generates a unit impulse signal.

```
% Program W2E3.m
% Generating 64 Samples of a unit impulse signal
N=64;                   % Define the number of samples
n=-(N/2):((N/2)-1);     % Define a vector of sample numbers
x=zeros(1,N);           % Define a vector of zeros
x((N/2)+1)=1.0;         % Make the first sample to be 1 (i.e.at
t=0)
plot(n,x);              % Plot the impulse
grid;
title('A Unit Impulse Signal');
xlabel('Sample Number');
ylabel('Amplitude');
```



Task: Generate 40 samples of the following signals:

(for n=-20,-19,-18,…,0,1,2,3,…,18,19)

(i) x(n)=2δ(n-10)

(ii) x(n)=5δ(n-10)+2.5δ(n-20)

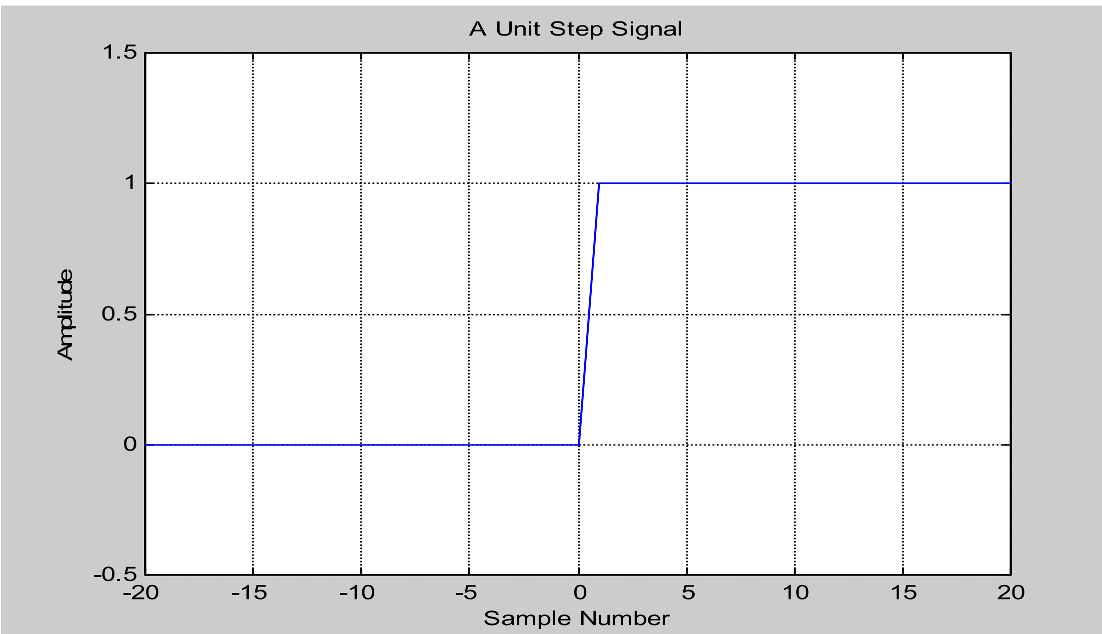
Exercise 4: Unit Step Signal Generation

A step signal is defined as follows:

$$u(n) = \begin{cases} 1 & n \geq 0 \\ 0 & otherwise \end{cases}$$

The following Matlab Program generates and plots a unit step signal:

```
% Program: W2E4.m
% Generates 40 samples of a unit step signal, u(n)
N=40;          % Define the number of samples
n=-20:20;      % Define a suitable discrete time axis
u=[zeros(1,(N/2)+1),ones(1,(N/2))];   % Generate the
signal
plot(n,u);                            % Plot the signal
axis([-20,+20,-0.5,1.5]);             % Scale axis
grid;
title('A Unit Step Signal');
xlabel('Sample Number');
ylabel('Amplitude');
```



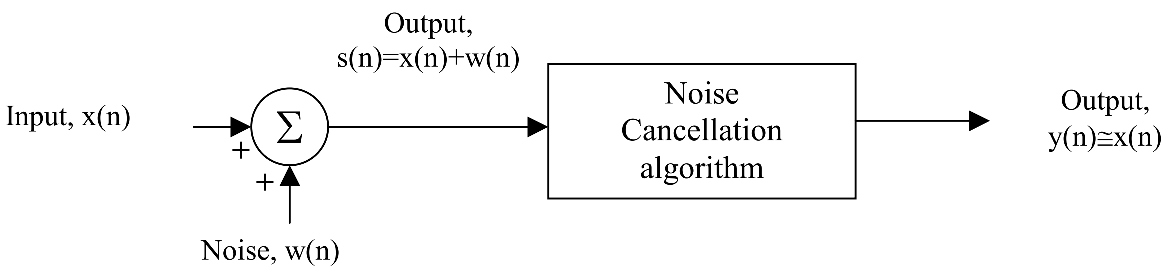
Task: Generate 40 samples of each of the following signals using an appropriate discrete time scale:

(i) x*(n)=u(n)-u(n-1)*

(ii) *g(n)=u(n-1)-u(n-5)*

Exercise 5: Generating Random Signals

Random number generators are useful in signal processing for testing and evaluating various signal processing algorithms. For example, in order to simulate a particular noise cancellation algorithm (technique), we need to create some signals which is contaminated by noise and then apply this signal to the algorithm and monitor its output. The input/ output spectrum may then be used to examine and measure the performance of noise canceller algorithm. Random numbers are generated to represent the samples of some noise signal which is then added to the samples of some the wanted signal to create an overall noisy signal. The situation is demonstrated by the following diagram.



Matlab provides two commands, which may be applied to generate random numbers.

Normally Distributed Random Numbers.

randn(N)

Is an N-by-N matrix with random entries, chosen from a normal distribution with mean zero and variance one.

randn(M,N), and *randn([M,N])* are M-by-N matrices with random entries.

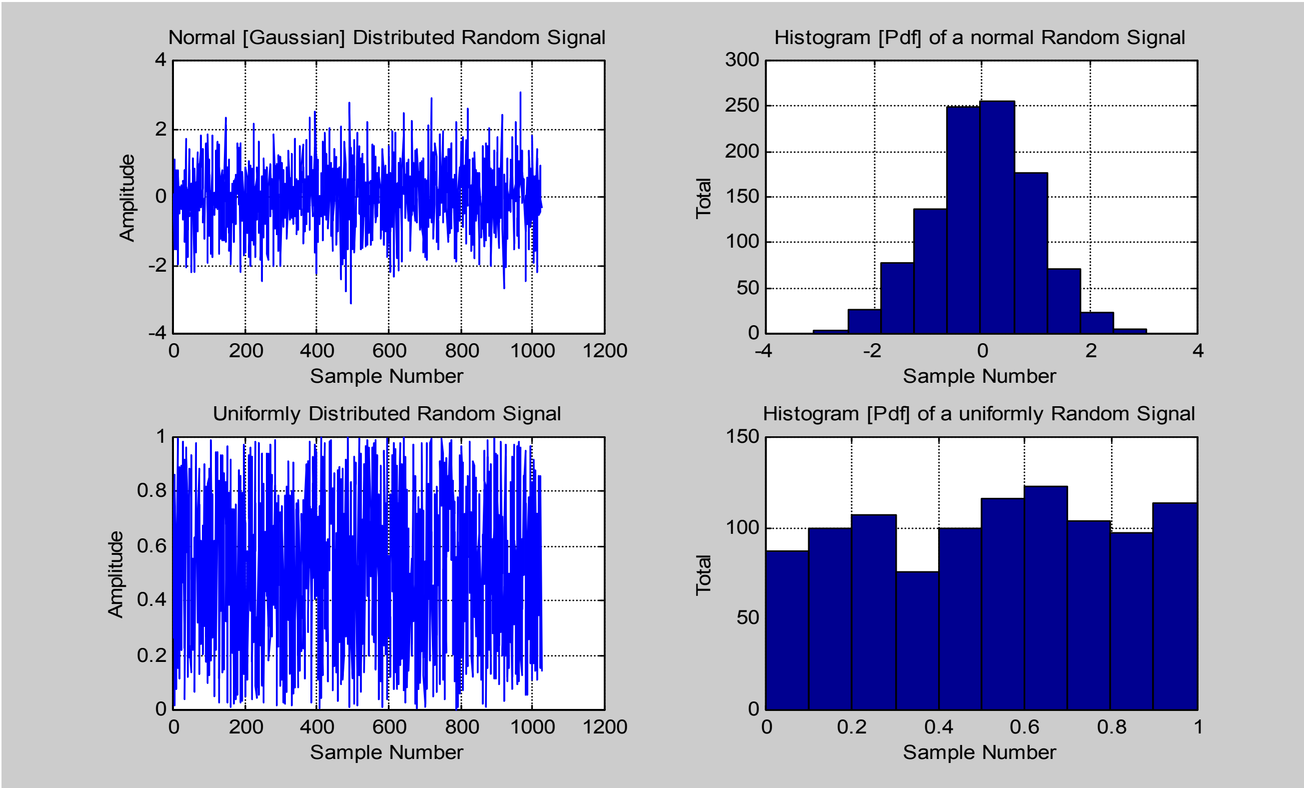
Uniformly Distributed Random Numbers.

rand(N) is an N-by-N matrix with random entries, chosen from a uniform distribution on the interval (0.0,1.0).

rand(M,N) and *rand([M,N])* are M-by-N matrices with random entries.

The following Matlab program generates random signals using each distribution.

```
% Program: W2E5.m
% Generates Uniformly and Normally Distributed random
signals

N=1024;            % Define Number of samples
R1=randn(1,N);     % Generate Normal Random Numbers
R2=rand(1,N);      % Generate Uniformly Random Numbers
figure(1);         % Select the figure
subplot(2,2,1);    % Subdivide the figure into 4 quadrants
plot(R1);          % Plot R1 in the first quadrant
grid;
title('Normal [Gaussian] Distributed Random Signal');
xlabel('Sample Number');
ylabel('Amplitude');

subplot(2,2,2);    % Select the second qudrant
hist(R1);          %  Plot the histogram of R1
grid;
title('Histogram [Pdf] of a normal Random Signal');
xlabel('Sample Number');
ylabel('Total');

subplot(2,2,3);
plot(R2);
grid;
title('Uniformly Distributed Random Signal');
xlabel('Sample Number');
ylabel('Amplitude');

subplot(2,2,4);
hist(R2);
grid;
title('Histogram [Pdf] of a uniformly Random Signal');
xlabel('Sample Number');
ylabel('Total');
```

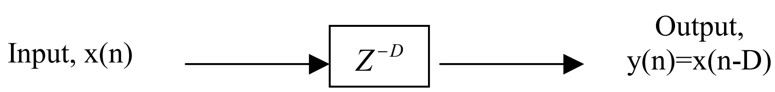


Task: Generate 128 Uniformly random numbers between $-\pi$ to $+\pi$

Exercise 6: Signal Manipulation

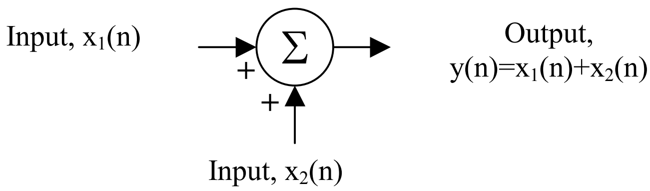
Some of the basic signal manipulation may be listed as follows:

❖ ***Signal Shifting / Delay***



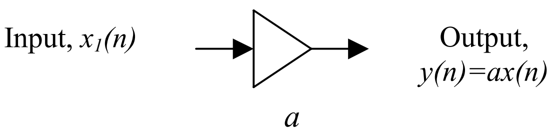
Here 'D' is the number of samples which the input signal must be delayed. For example if x(n)=[1, 2, 3, 4, 5], then x(n-3)=[0,0,0,1,2,3,4,5].

❖ *Signal Addition / Subtraction*



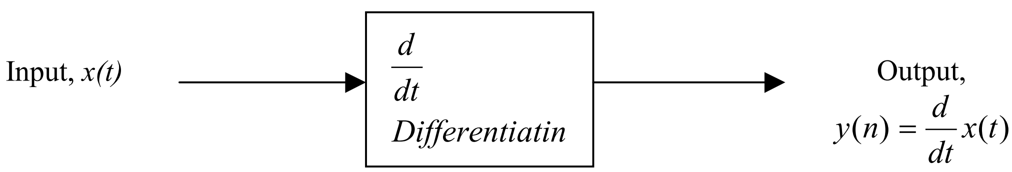
When adding two signals together, signals must have the same number of samples. If one signal has less than number of samples than the other, then this signal may be appended with zeros in order to make it equal length to the second signal before adding them.

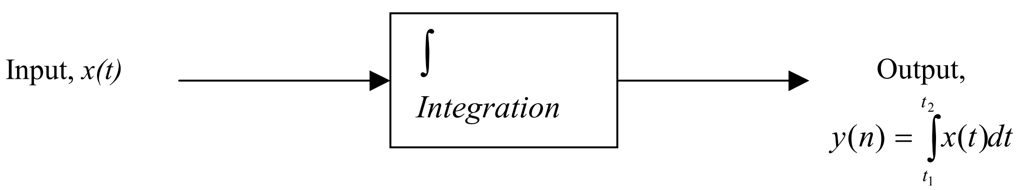
❖ *Signal Amplification / Attenuation*



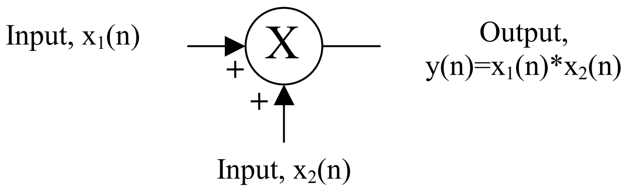
'a' is a numerical constant. If a>1, then the process is referred to as 'amplification' , if 0<a<1, the process is referred to as 'attenuation'.

❖ *Signal Differentiation / Integration*





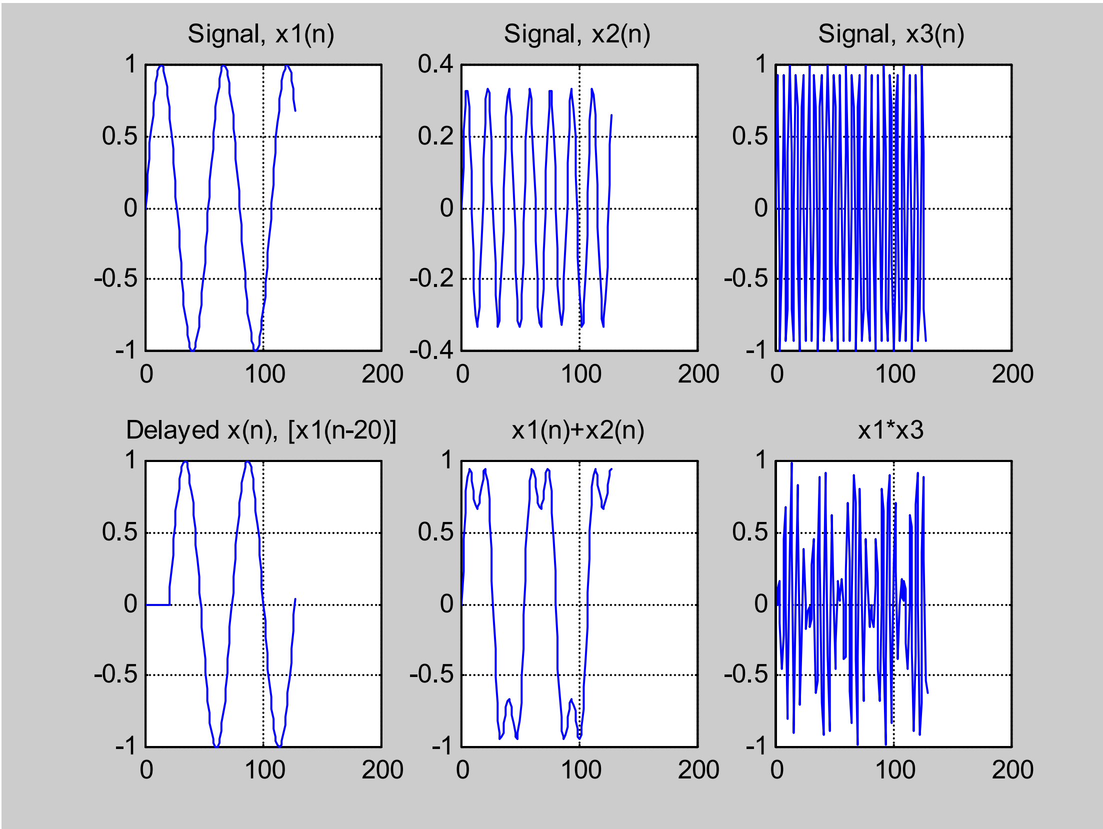
❖ *Signal Multiplication / Division*



A Matlab program given below, provide an example of each of the above basic operations:

```
% Program: W2E6.m
% Program demonstrating Basic Signal Manipulation
N=128;
f1=150;
f2=450;
f3=1500;
fs=8000;
n=0:N-1;
x1=sin(2*pi*(f1/fs)*n);
x2=(1/3)*sin(2*pi*(f2/fs)*n);
x3=sin(2*pi*(f3/fs)*n);
figure(1);
subplot(1,1,1);
subplot(2,3,1);
plot(n,x1);
grid;
title('Signal, x1(n)');

subplot(2,3,2);
plot(n,x2);
grid;
title('Signal, x2(n)');

subplot(2,3,3);
plot(n,x3);
grid;
title('Signal, x3(n)');

% Signal Delay
x1d=[zeros(1,20), x1(1:N-20)];
subplot(2,3,4);
plot(n,x1d);
```

```
grid;
title('Delayed x(n), [x1(n-20)]');
% Signal Addition
xadd=x1+x2;
subplot(2,3,5);
plot(n,xadd);
grid;
title('x1(n)+x2(n)');
 % Signal Multiplication
xmult=x1.*x3;
subplot(2,3,6);
plot(xmult);
grid;
title('x1*x3');
```



Discrete Fourier Transform

[1] Given a sampled signal x(n), its Discrete Fourier Transform (DFT) is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N} \qquad k=0,1,2,\ldots,N-1$$

The magnitude of the X(k) [i.e. the absolute value] against k is called the spectrum of x(n). The values of k is proportional to the frequency of the signal according to:

$$f_k = \frac{kF_s}{N}$$

Where F_s is the sampling frequency.

Assuming x is the samples of the signal of length N, then a simple Matlab program to perform the above sum is shown below:

```
n=[0:1:N-1];
k=[0:1:N-1];
WN=exp(-j*2*pi/N);
nk=n'*k;
WNnk=WN .^ nk;
Xk=x * WNnk;
```

Use the above routine to determine and plot the DFT of the following signal:

$$x(t) = cos(2\pi f_1 t) + cos(2\pi f_2 t)$$

where

$$f_1 = 1000Hz \; and \; f_2 = 400Hz, with \; F_s = 8000Hz, and \; N = 128$$

Note: Plot the magnitude and phase of Xk as follows:

```
MagX=abs(Xk);
PhaseX=angle(Xk)*180/pi;
figure(1);
```

```
subplo(2,1,1);
plot(k,MagX);
subplot(2,1,2);
plot(k,PhaseX);
```

Investigate:

(i) At what value of the index *k* does the magnitude of the DFT of *x* has major peaks.

(ii) What is the corresponding frequency of the two peaks.

(iii) Perform the above for N=32, 64 and 512.

(iv) Comment on the results

(v) Set 128 and generate x(n) for n=0,1,2,…,N-1. Append a further 512 zeros to x(n) as shown below: [Note the number of samples, N=128+512=640].

Xe=[x, zeros(1,512)];

Perform the DFT of the new sequence and plot its magnitude and phase. Compare with the previous result. Explain the effect of zero padding a signal with zero before taking the discrete Fourier Transform.

[2] Inverse DFT is defined as:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j2\pi nk/N} \quad \text{for } 0 \le n \le N-1$$

A simple Matlab routine to perform the inverse DFT may be written as follows:

```
n=[0:1:N-1];
k=[0:1:N-1];
WN=exp(-j*2*pi/N);
nk=n'*k;
WNnk=WN .^ (-nk);
x=(Xk * WNnk)/N;
```

Use x(n)={1,1,1,1}, and N=4, determine the DFT. Record the Magnitude and phase of the DFT.

Use the IDFT to transfer the DFT results (i.e. Xk sequence) to its original sequence.

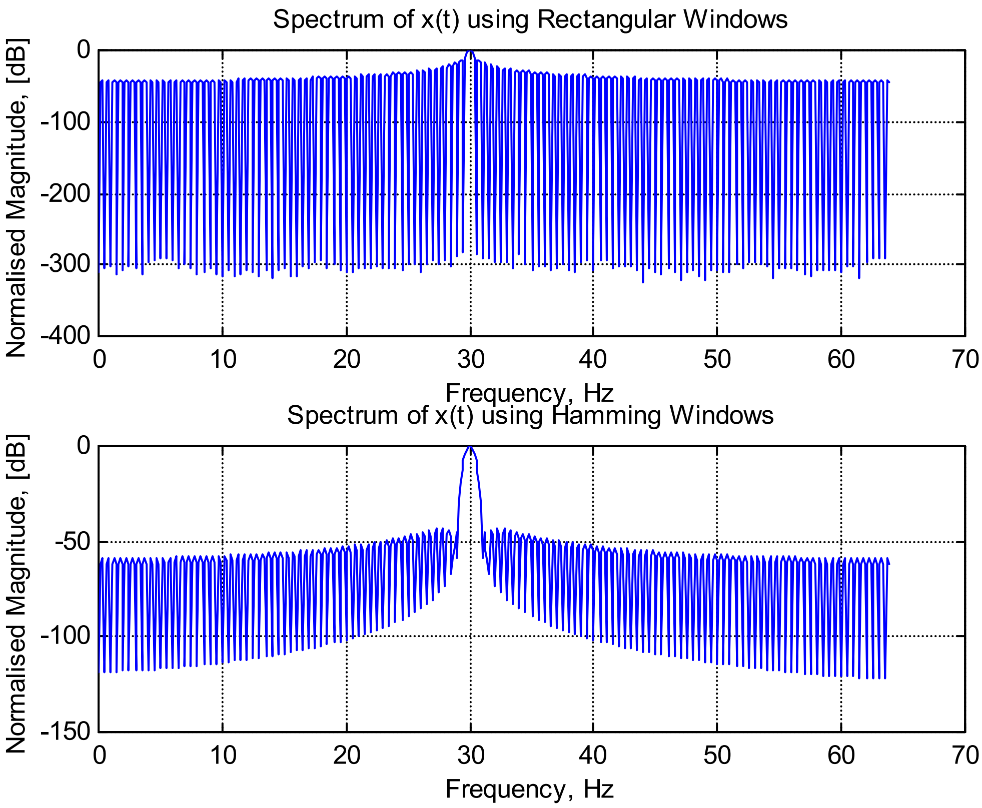
Spectral Analysis of a signal

Let $x(t) = cos(2\pi f_0 t)\ where\ f_0 = 30Hz$, assuming the sampling frequency $f_s = 128Hz\ and\ N = 256\ samples$, obtain the FFT of the windowed signal using rectangular and hamming windows, zero padded to N1=1024. Plot the normalised FFT magnitude of the windowed signals. Which windowed signal shows a narrower mainlobe?. Which windowed signal shows the smaller peak sidelobe?.

A Matlab program implementing the spectrum is shown below:

```
f1=30;              % Signal frequency
fs=128;             % Sampling frequency
N=256;              % Number of samples
N1=1024;            % Number of FFT points
n=0:N-1;            % Index n
f=(0:N1-1)*fs/N1;       % Defining the frequency points [axis]
x=cos(2*pi*f1*n/fs);    % Generate the signal
XR=abs(fft(x,N1));      % find the magnitude of the FFT using No
                        % windowing (i.e. Rectangular window)
xh=hamming(N);          % Define the hamming samples
xw=x .* xh';            % Window the signal
XH=abs(fft(xw,N1));     % find the magnitude of the FFT of the
                        % windowed signal.
subplot(2,1,1);         % Start plotting the signal
plot(f(1:N1/2),20*log10(XR(1:N1/2)/max(XR)));
title('Spectrum of x(t) using Rectangular Windows');
grid;
xlabel('Frequency, Hz');
ylabel('Normalised Magnitude, [dB]');
subplot(2,1,2);
plot(f(1:N1/2),20*log10(XH(1:N1/2)/max(XH)));
title('Spectrum of x(t) using Hamming Windows');
grid;
xlabel('Frequency, Hz');
ylabel('Normalised Magnitude, [dB]');
```

And here is the output from the program:



It can be seen from the spectrum plots that both the rectangular window and hamming window have their peak amplitude at f=30Hz corresponding to the signal frequency. While the rectangular window has a narrower mainlobe, hamming window provides less peak sidelobes than the rectangular window. There are many other types of windows which are available as Matlab functions and are listed below:

bartlett(N)	returns the N-point Bartlett window
blackman(N)	returns the N-point Blackman window
boxcar(N)	***returns the N-point rectangular window***
hamming(N)	***returns the N-point Hamming window.***
hanning(N)	returns the N-point Hanning window in a column vector.
kaiser(N,beta)	returns the BETA-valued N-point Kaiser window.
triang(N)	returns the N-point triangular window.

Further Spectral Analysis

The spectrum of a signal is simply a plot of the magnitude of the components of a signal against their corresponding frequencies. For example if a signal consists of two sinusoidal at 120Hz and 60Hz, then ideally the spectrum would be a plot of magnitude vs. frequency and would contain two peaks: one at 60 Hz and the other at 120 Hz illustrated in the next example. The following Matlab commands are the basis of determining the spectrum of a signal:

fft
psd
spectrum

In the following examples, we illustrate their use. One of the most important aspects of spectral analysis is the interpretation of the spectrum and its relation to the signal under investigation.

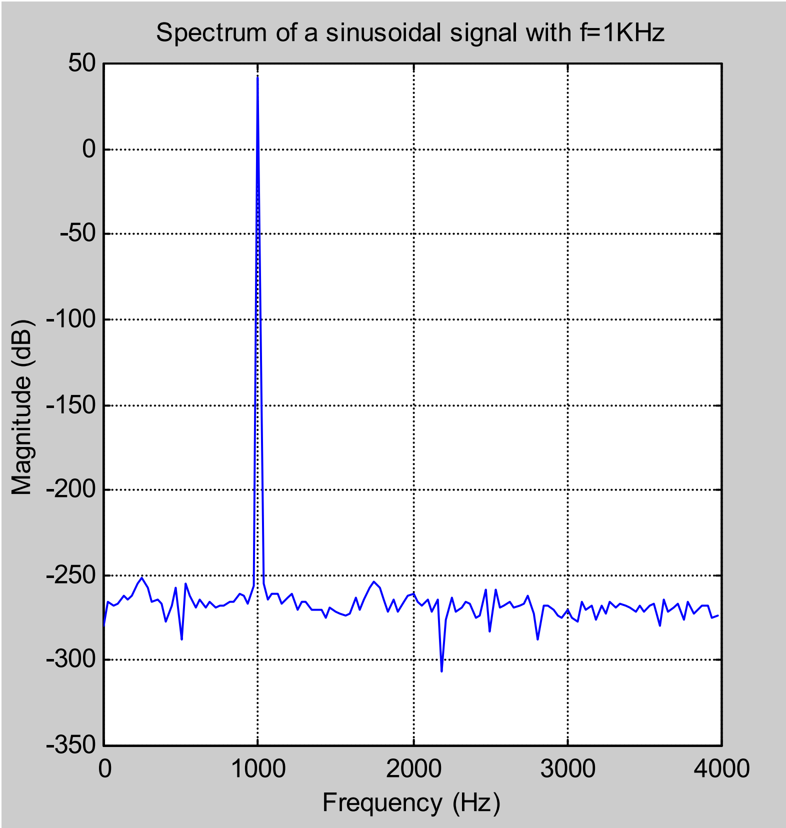
Task 1:

Generate 256 samples of a sinusoidal signal of frequency 1KHz with a sampling frequency of 8KHz. Plot the signal and its spectrum.

Here is the program:

```
N=256;  % Total Number of Samples
fs=8000;  % Sampling frequency set at 1000Hz
f=1000;
n=0:N-1;
% Now generate the sinusoidal signal
x=sin(2*pi*(f/fs)*n);
% Estimate its spectrum using fft command
X=fft(x);
magX=abs(X);
% Build up an appropriate frequency axis
fx=0:(N/2)-1;  % first make a vector for
f=0,1,2,...(N/2)-1
fx=(fx*fs)/N;  % Now scale it so that it represents
frequencies in Hz
figure(1);
subplot(1,1,1);
plot(fx,20*log10(magX(1:N/2)));
grid;
title('Spectrum of a sinusoidal signal with f=1KHz');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
```

And the output:



It is clear from the spectrum, that the signal consists of a single sinusoidal components of frequency 1000Hz. Other artefacts in the figure are due to the limited number of samples, windowing effects, and computation accuracy.

Task 1:

Consider, the notch filter in the previous section. The transfer function of the notch filter is repeated here for convenience:

Input, x(n) →

$$H(z) = 0.9969\,\frac{1 - 1.8996z^{-1} + z^{-2}}{1 - 1.8537z^{-1} + 0.9937z^{-2}}$$

impulse response, h(n)

→ *Output, y(n)*

Apply an input signal:

$x(n) = \sin(\frac{2\pi f_1 n}{f_s}) + \sin(\frac{2\pi f_2 n}{f_s})$, with f_1=120 Hz , f_2=60 Hz and f_s , the sampling frequency at 1000Hz.

Plot the followings:

Input Spectrum

Magnitude response of the filter

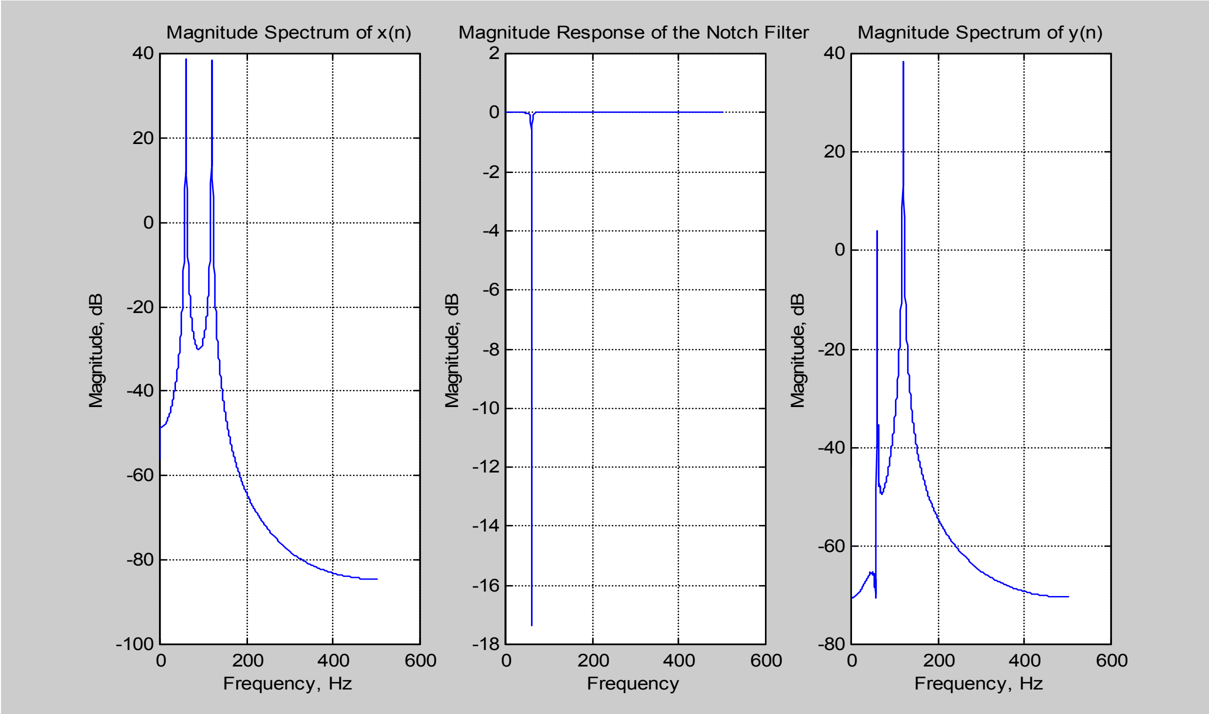
Output Spectrum

Here is the Matlab Program:

```
% Program Name: Tut16.m
clear;
N=1024;    % Total Number of Samples
fs=1000;   % Sampling frequency set at 1000Hz
f1=120;
f2=60;
n=0:N-1;
x=sin(2*pi*(f1/fs)*n)+sin(2*pi*(f2/fs)*n);
[pxx,fx]=psd(x,2*N,fs);
plot(fx,20*log10(pxx));
grid;
title('Magnitude Spectrum of x(n)');
xlabel('Frequency, Hz');
ylabel('Magnitude, dB');
sin(2*pi*(f2/fs)*n);
b=[1 -1.8596 1];
a=[1 -1.8537 0.9937];
k=0.9969;
b=k*b;
figure(1);
subplot(1,1,1);

subplot(1,3,1);
```

```
[pxx,fx]=psd(x,2*N,fs);
plot(fx,20*log10(pxx));
grid;
title('Magnitude Spectrum of x(n)');
xlabel('Frequency, Hz');
ylabel('Magnitude, dB');

[h,f]=freqz(b,a,1024,fs);   % Determines the frequency response
                            % of
                            % the filter with coefficients 'b'
                            % and 'a', using 1024 points around
                            % the unit circle with a sampling
                            % frequency of fs.  The function
                            % returns the values of the transfer
                            % function h, for each frequency in
                            % f in Hz.
magH=abs(h);           % Calculates the magnitude of the filter
phaseH=angle(h);       % Calculates the phase angle of the filter
subplot(1,3,2);        % Divides the figure into two rows and one
                       % Column and makes the first row the active
                       % one
plot( f, 20*log10(magH));   % plots the magnitude in dB against
                            % frequency
grid;                       % Add a grid


title('Magnitude Response of the Notch Filter');
xlabel('Frequency');
ylabel('Magnitude, dB');

y=filter(b,a,x);

[pyy,fy]=psd(y,2*N,fs);   % determine the Spectrum using 'psd'
subplot(1,3,3);           % Select the third column in the
figure
plot(fy,20*log10(pyy));   % Plot the output spectrum in dB
grid;                     % Add a grid
title('Magnitude Spectrum of y(n)');
xlabel('Frequency, Hz');
ylabel('Magnitude, dB');
```

Here is the output from the program:



It can be seen from the plot, that the input signal consists of two sinusoidal components at 60Hz and 120Hz. From the magnitude response of the filter, it is clear that the filter attenuates components whose frequencies are at 60 Hz. Therefore, the output spectrum consists of the 120 Hz signal and attenuated (0 dB) signal at 60Hz.

A final example considers the use of the Matlab command *'spectrum'* in order to estimate the spectrum of a given signal.

Task 3:

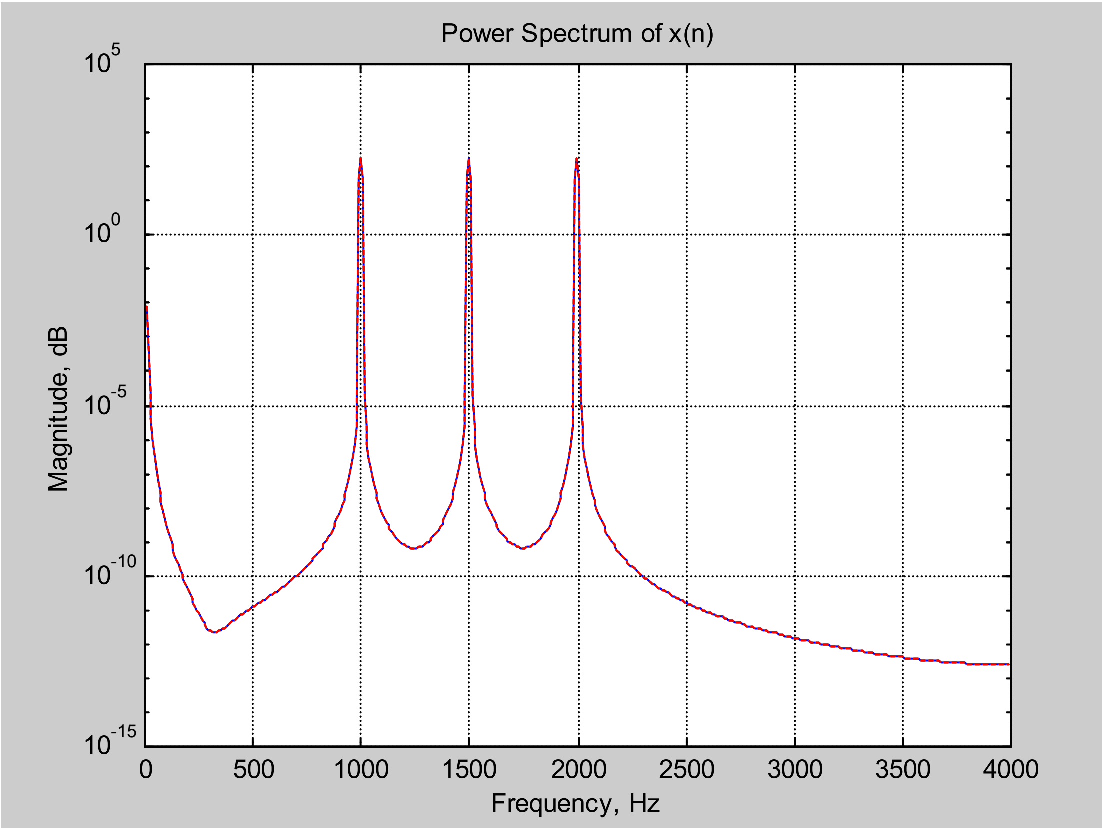
Consider estimating the spectrum of a signal consisting of three sinusoidal signals at frequencies, f1=500Hz, f2=1000Hz, and f3=1500Hz. Assume a sampling frequency of 8KHz.

The Matlab Program and its output is shown next:

```
% Program Name: Tut17.m
clear;
N=1024; % Total Number of Samples
fs=8000;% Sampling frequency set at 1000Hz
f1=500;
f2=1000;
f3=1500;
n=0:N-1;
% Generate the signal
```

```
x=sin(2*pi*(f1/fs)*n)+sin(2*pi*(f2/fs)*n)+sin(2*pi*(f3/fs)*n);
pxx=spectrum(x,N);        % Estimate the Spectrum
specplot(pxx,fs);         % Plot the spectrum
grid;
title('Power Spectrum of x(n)');
xlabel('Frequency, Hz');
ylabel('Magnitude, dB');
```

And the output:



Autocorrelation and Crosscorrelation in Matlab

Both in signal and Systems analysis, the concept of autocorrelation and crosscorrelation play an important role. In the following sections, we present some simple examples of how these two functions may be estimated in Matlab. The final part of this section provides some applications of autocorrelation and crosscorrelation functions in signal detection and time-delay estimations.

The autocorrelation function of a random signal describes the general dependence of the values of the samples at one time on the values of the samples at another time. Consider a random process x(t) (i.e. continuous-time), its autocorrelation function is written as:

$$R_{xx}(\tau) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} x(t)x(t+\tau)dt \qquad (1)$$

Where T is the period of observation.

$R_{xx}(\tau)$ is always real-valued and an even function with a maximum value at $\tau = 0$.

For sampled signal (i.e. sampled signal), the autocorrelation is defined as either biased or unbiased defined as follows:

:

$$R_{xx}(m) = \frac{1}{N-|m|} \sum_{n=1}^{N-m+1} x(n)x(n+m-1) \qquad \textit{[Biased Autocorrelation]}$$

(2)

$$R_{xx}(m) = \frac{1}{N} \sum_{n=1}^{N-m+1} x(n)x(n+m-1) \qquad \textit{[Uniased Autocorrelation]}$$

for m=1,2,…,M+1

where M is the number of lags.

Some of its properties are listed in table 1.1.

The *cross correlation* function however measures the dependence of the values of one signal on another signal. For two WSS (Wide Sense Stationary) processes x(t) and y(t) it is described by:

$$R_{xy}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-T}^{T} x(t)y(t+\tau)dt \tag{3}$$

or

$$R_{yx}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-T}^{T} y(t)x(t+\tau)dt \tag{4}$$

where T is the observation time.

For sampled signals, it is defined as:

$$R_{yx}(m) = \frac{1}{N} \sum_{n=1}^{N-m+1} y(n)x(n+m-1) \tag{5}$$

m=1,2,3,..,N+1

Where N is the record length (i.e. number of samples).

The properties of cross correlation function are listed in table 1.2.

Autocorrelation Properties
[1] Maximum Value: *The magnitude of the autocorrelation function of a wide sense stationary random process at lag m is upper bounded by its value at lag m=0:* $$R_{xx}(0) \geq \lvert R_{xx}(k) \rvert \ for\ k \neq 0$$
[2] Periodicity: *If the autocorrelation function of a WSS random process is such that:* $R_{xx}(m_0) = R_{xx}(0)$ *for some* m_0, *then* $R_{xx}(m)$ *is periodic with period* m_0. *Furthermore* $E[\lvert x(n) - x(n-m_0) \rvert^2] = 0$ *and x(n) is said to be mean-square periodic.*
[3] The autocorrelation function of a periodic signal is also periodic: *Example: if* $x(n) = A\,sin(\omega_0 n + \varphi)$, *then,* $R_{xx}(m) = \frac{A^2}{2} cos(\omega_0 m)$ *Therefore if* $\omega_0 = \frac{2\pi}{N}$, *then* $R_{xx}(m)$ *is periodic with periodic N and x(n) is mean-square periodic.*
[4] Symmetry: *The autocorrelation function of WSS process is a conjugate symmetric function of m:* $$R_{xx}(m) = R^{*}_{xx}(-m)$$ *For a real process, the autocorrelation sunction is symmetric:* $R_{xx}(m) = R_{xx}(-m)$
[5] Mean Square Value: *The autocorrelation function of a WSS process at lag, m=0, is equal to the mean-square value of the process:*

$$R_{xx}(0) = E\{\lvert x(n)\rvert^2\} \geq 0$$ *[6] If two random processes x(n) and y(n) are uncorrelated, then the autocorrelation of the sum x(n)=s(n)+w(n) is equal to the sum of the autocorrelations of s(n) and w(n):* $$R_{xx}(m) = R_{ss}(m) + R_{ww}(m)$$
[7] The mean value: *The mean or average value (or d.c.) value of a WSS process is given by:* $mean, \bar{x} = \sqrt{R_{xx}(\infty)}$

Table 1.1 Properties of Autocorrelation function

Properties of cross correlation function
[1] $R_{xy}(m)$ *is always a real valued function which may be positive or negative.*
[2] $R_{xy}(m)$ *may not necessarily have a maximum at m=0 nor* $R_{xy}(m)$ *an even function.*
[3] $R_{xy}(-m) = R_{yx}(m)$
[4] $\lvert R_{xy}(m)\rvert^2 \leq R_{xx}(0)R_{yy}(0)$
[5] $\lvert R_{xy}(m)\rvert \leq \frac{1}{2}\left[R_{xx}(0) + R_{yy}(0)\right]$
[6] When $R_{xy}(m) = 0$*, x(n) and y(n) are Sid to be 'uncorrelated' or they are said to be statistically independent (assuming they have zeros mean.)*

Table 1.1 Properties of cross correlation function

Matlab Implementation

Matlab provides a function called ***xcorr.m*** which may be used to implement both auto and cross correlation function. Its use is indicated in the following examples:

Example 1: **Autocorrelation of a sinewave**

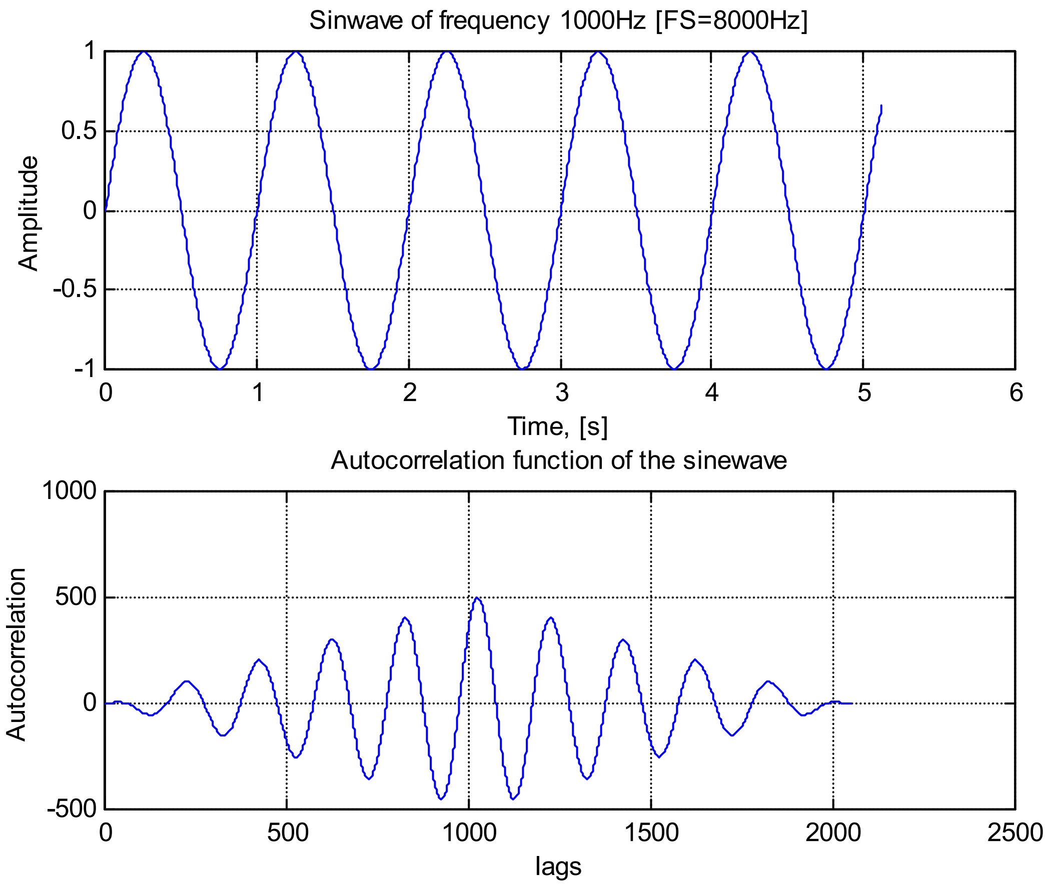
Plot the autocorrelation sequence of a sinewave with frequency 1 Hz, sampling frequency of 200 Hz.

The Matlab program is listed below:

```
N=1024;                 % Number of samples
f1=1;                   % Frequency of the sinewave
FS=200;                 % Sampling Frequency
n=0:N-1;                % Sample index numbers
x=sin(2*pi*f1*n/FS);     % Generate the signal, x(n)
t=[1:N]*(1/FS);          % Prepare a time axis
subplot(2,1,1);          % Prepare the figure
plot(t,x);               % Plot x(n)
title('Sinwave of frequency 1000Hz [FS=8000Hz]');
xlabel('Time, [s]');
ylabel('Amplitude');
grid;
Rxx=xcorr(x);            % Estimate its autocorrelation
subplot(2,1,2);          % Prepare the figure
plot(Rxx);               % Plot the autocorrelation
grid;
title('Autocorrelation function of the sinewave');
xlable('lags');
ylabel('Autocorrelation');
```

The output of this program is shown next.

Notice that when using the function *xcorr, to estimate* the autocorrelation sequence , it has double the number of samples as the signal x(n). An important point to remember when using the function *xcorr* is that the origin is in the middle of the figure (here it is at lag=1024).



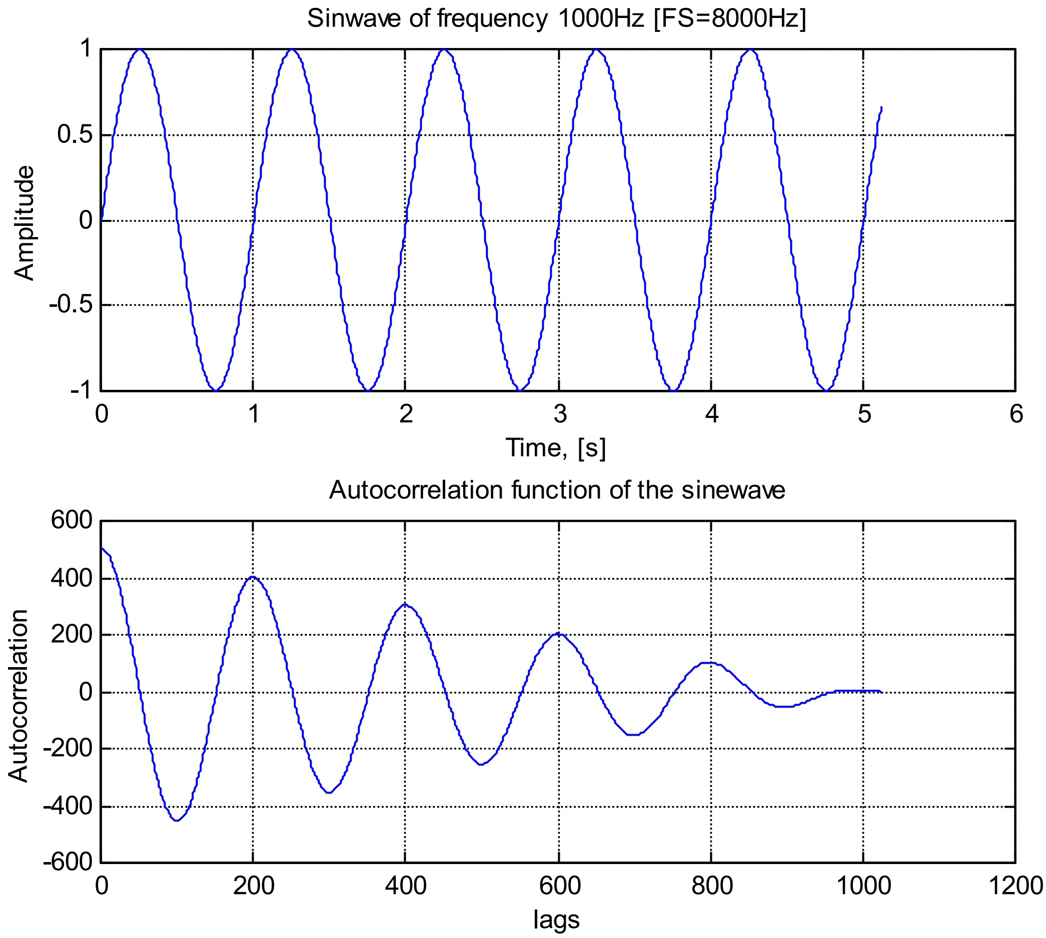
Note that if you want to write your own autocorrelation function, you may use equation (2) to do this. Here is a how it may be written:

```
function [Rxx]=autom(x)
% [Rxx]=autom(x)
% This function Estimates the autocorrelation of the sequence
of
% random variables given in x as: Rxx(1), Rxx(2),…,Rxx(N),
where N is
% Number of samples in x.

N=length(x);
Rxx=zeros(1,N);
for m=1: N+1
   for n=1: N-m+1
      Rxx(m)=Rxx(m)+x(n)*x(n+m-1);
   end;
end;
```

To use the above function, you need to save this under autom.m and then it may be used as follows:

```
N=1024;
f1=1;
FS=200;
n=0:N-1;
x=sin(2*pi*f1*n/FS);
t=[1:N]*(1/FS);
subplot(2,1,1);
plot(t,x);
title('Sinwave of frequency 1000Hz [FS=8000Hz]');
xlabel('Time, [s]');
ylabel('Amplitude');
grid;
Rxx=autom(x);
subplot(2,1,2);
plot(Rxx);
grid;
title('Autocorrelation function of the sinewave');
xlabel('lags');
ylabel('Autocorrelation');
```

Note this version of estimating the autocorrelation function generates the same number of samples as the signal itself and that the maximum is now placed at the origin. ($R_{xx}(1)$ is the origin).



Example 2: Crosscorrelation

Plot the crosscorrelation of the following signal:

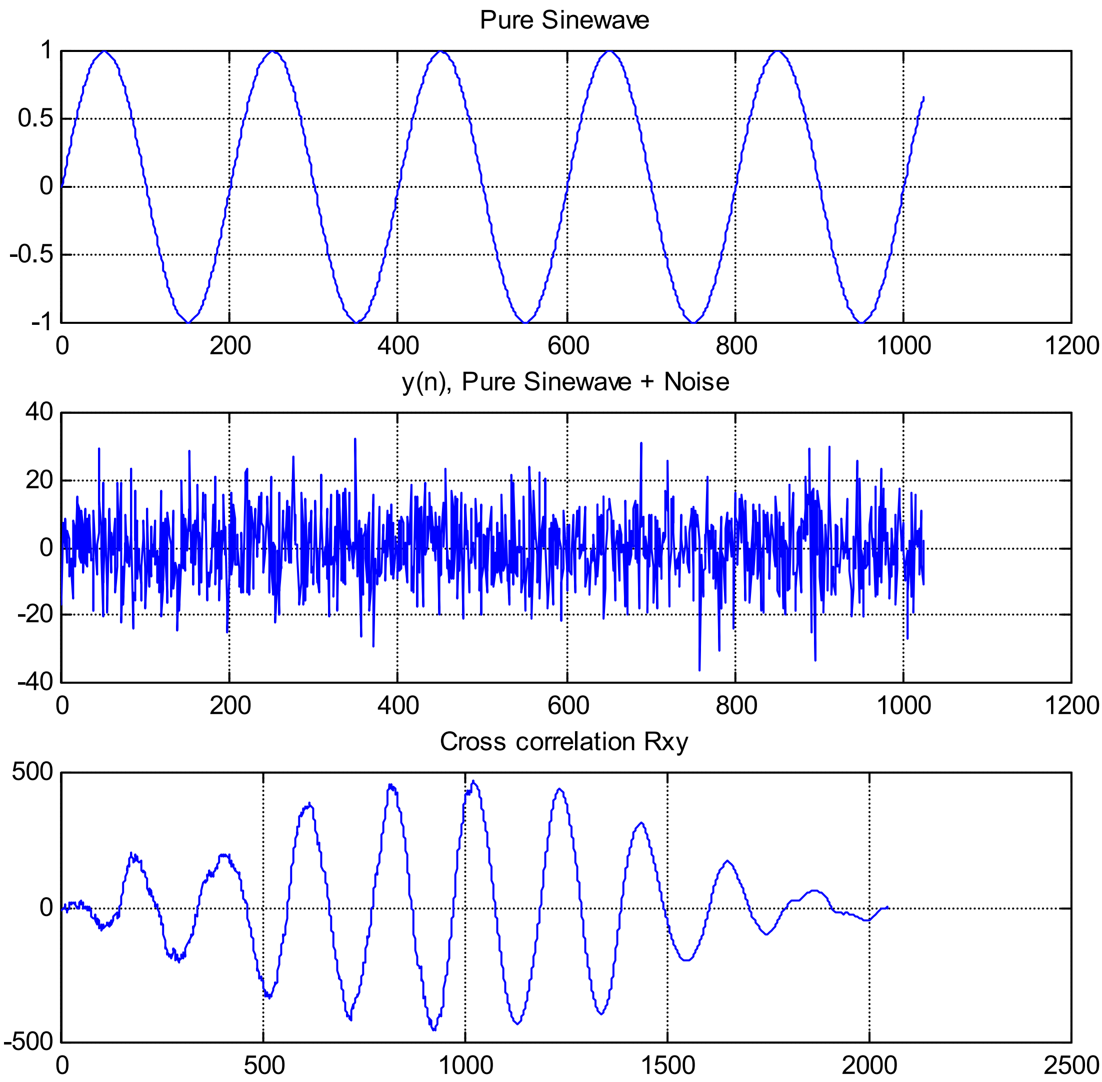
$$x(n) = sin(2\pi f_1 t) \quad with\ f_1 = 1Hz$$
$$y(n) = x(n) + w(n)$$

where *w(n)* is a zeros mean, unit variance of Gaussina random process.

```
N=1024;               % Number of samples to generate
f=1;                  % Frequency of the sinewave
FS=200;               % Sampling frequency
n=0:N-1;              % Sampling index
x=sin(2*pi*f1*n/FS);   % Generate x(n)
y=x+10*randn(1,N);     % Generate y(n)
subplot(3,1,1);
plot(x);
title('Pure Sinewave');
grid;
subplot(3,1,2);
```

```
plot(y);
title('y(n), Pure Sinewave + Noise');
grid;
Rxy=xcorr(x,y);        % Estimate the cross correlation
subplot(3,1,3);
plot(Rxy);
title('Cross correlation Rxy');
grid;
```

The output is:



Question [1]

From the results shown for example 2, what function cross correlation has performed.

Question [2]

Given $x(n) = A\,sin(\omega_0 t + \vartheta)$, prove that its autocorrelation function is given by:

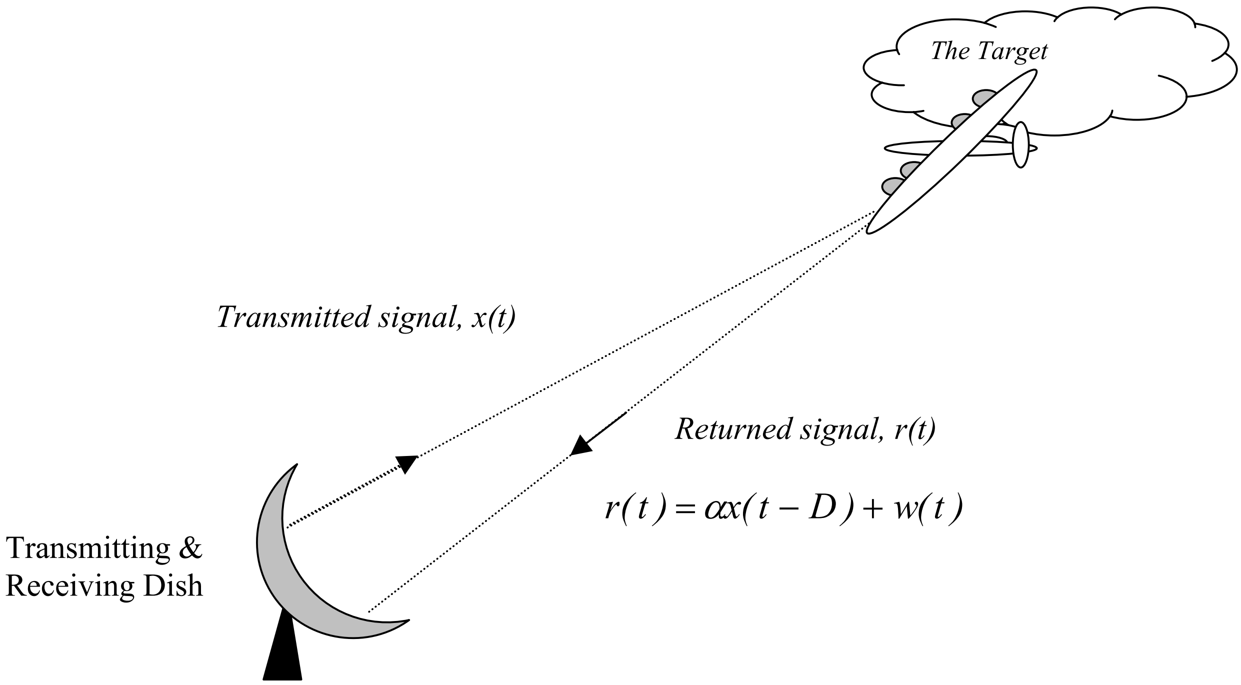
$$R_{xx}(\tau) = \frac{A^2}{2} cos(\omega\tau)$$

Explain this result.

Case Study

Time Delay Estimation
Processing Radar Returned Signal

Consider a simple radar illustration shown below:



A pulse x(t) is transmitted, the reflected signal from an object is returned to the receiver. The returned signal *(r(t))* is delayed (i.e. D seconds) , noisy and attenuated. The objective is to measure (estimate) the time delay between the transmitted and the returned signal.

Analysis: Let the transmitted signal be x(t), then the returned signal r(t) may be modelled as:

$$r(t) = \alpha x(t - D) + w(t)$$

where: *w(t)* is assumed to be the additive noise during the transmission.

α is the attenuation factor (<1).

D is the delay which is the time taken for the signal to travel from the transmitter to the target and back to the receiver.

A common method of estimating the time delay D is to compute the cross-correlation function of the received signal with the transmitted signal x(t). i.e.

$$\begin{aligned} R_{rx}(\tau) &= E\{r(t)x(t+\tau)\} \\ &= E\{[\alpha x(t - D) + w(t)][x(t+\tau)]\} \\ &= E\{\alpha x(t - D)x(t+\tau) + w(t)x(t+\tau)\} \end{aligned} \qquad (6)$$

Hence

$$R_{rx}(\tau) = \alpha R_{xx}(\tau - D) + R_{wx}(\tau)$$

Note, '*E*' is the expectation operator.

Therefore, the cross correlation $R_{rx}(\tau)$ is equal to the sum of the scaled autocorrelation function of the transmitted signal (i.e. $\alpha R_{xx}(\tau)$) and the cross correlation function between x(t) and the contaminated noise signal w(t). If we now assume that the noise signal w(t) and the transmitted signal x(t) are uncorrelated then,

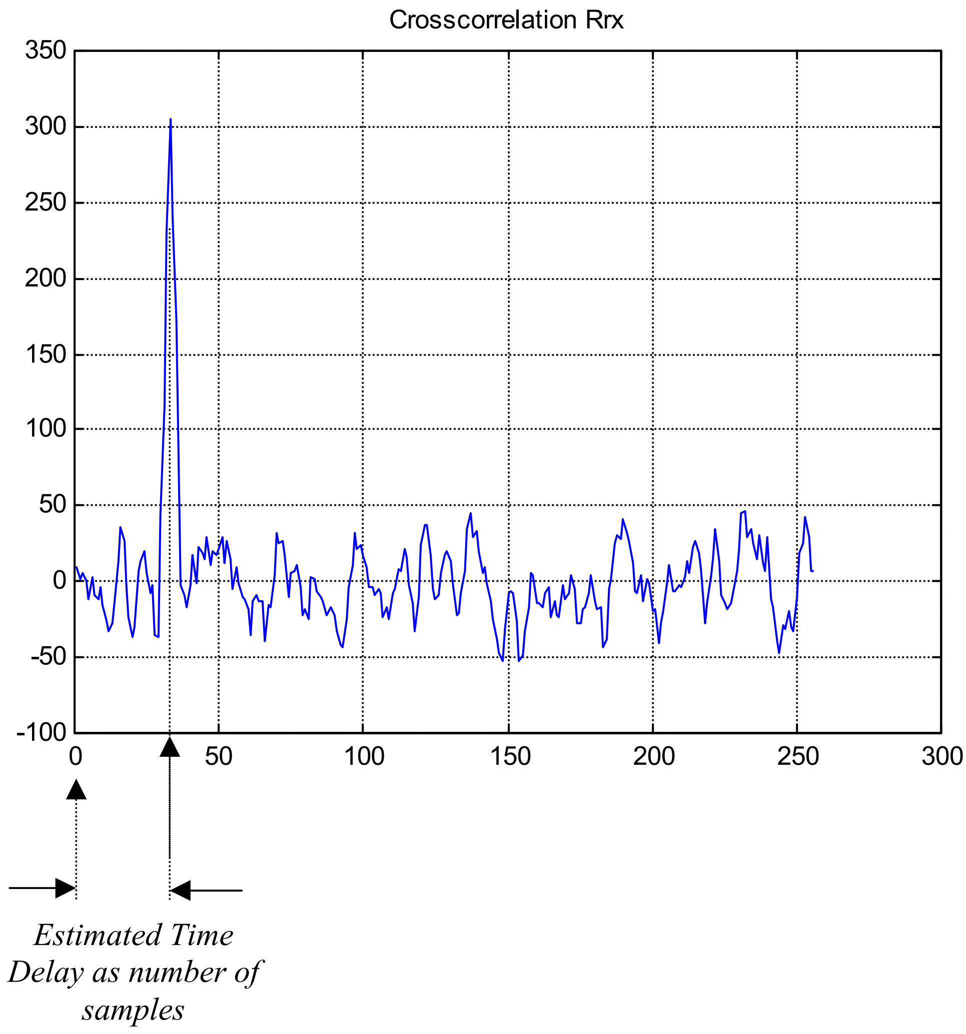
$$R_{wx}(\tau) = 0$$

This is also stated in table 1.2 under the property number [6].

Hence the cross-correlation function between the transmitted signal and the received signal may be written as:

$$R_{rx}(\tau) = \alpha R_{xx}(\tau - D) \qquad (7)$$

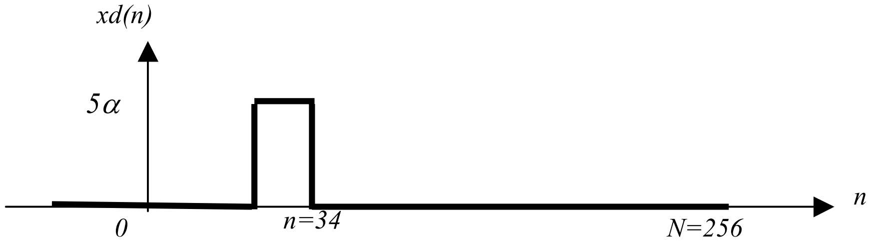
Therefore if we plot $R_{rx}(\tau)$, it will only have one peak value that will occur at $\tau = D$. A typical plot of $R_{rx}(\tau)$ is shown below:



Investigation

[1] Generate a single pulse for the transmitted signal as shown below:

[2] Delay the signal by say 32 samples and reduce its amplitude by an attenuation factor of say $\alpha = 0.7$, call this xd(n) as shown below:



[3] Generate N=256 samples of Gaussian random signal and call this w(n).

[4] Generate the simulated received signal by adding the transmitted signal x(n) and the noise signal w(n), i.e.

$$r(n) = \alpha x(n-d) + sigman \times w(n)$$

Where *sigman* is the noise amplitude (initially set this to 1.

[5] using subplot(2,2,1), plot the signals x(n), xd(n), and r(n). Appropriately label and grid the each plot.

[6] Estimate the cross-correlation sequence $R_{rx}(m)$ and plot in the fourth quadrant of the figure. Note , plot only half the samples of the cross correlation sequence returned by the function xcorr. This can be done as follows:

```
% Assuming there are N samples of x
Y=xcorr(r,x);
R=Y(1:N);
Rrx=fliplr(R);
.
.
```

[7] From the plot estimate the delay. Does it agree with the theoretical delay value?

[8] Repeat the simulation for some values of α , sigman, and N?

[9] Comment on your findings?

Time delay Estimation in Frequency Domain

Consider the returned signal once again:

$$r(t) = \alpha x(t - D) + w(t)$$

Taking the Fourier transform of both sides yields:

$$P_r(\omega) = \alpha X(\omega)e^{-j\omega D} + W(\omega)$$

Or taking the Fourier Transform of the cross correlation of r(t) and x(t) gives:

$$P_{rx}(\omega) = \alpha P_{xx}(\omega)e^{-j\omega D} + P_{wx}(\omega)$$

Assuming that the transmitted signal and the contaminated noise are uncorrelated, we get:

$$P_{xr}(\omega) = \alpha P_{xx}(\omega)e^{-j\omega D}$$

Therefore by having an estimate of the cross spectral function of the transmitted and the received signal, we can estimate the time delay from its phase:

$$\begin{aligned} Phase &= \omega D \\ &= 2\pi f D \end{aligned}$$

Hence by taking the slope of the phase (i.e. $\frac{d}{d(f)} Phase$), we have the slope of the phase which may be written as:

$$Slope = 2\pi D$$

Or

$$D = \frac{Slope}{2\pi}$$

Here is how you may estimate the cross spectrum between the transmitted signal and the received signal and to obtain the phase plot.

```
% Estimation of time delay using the phase of the Cross
Spectrum
Prx=csd(x,r);        % Estimate the Cross Spectrum
pha=angle(Prx);      % Get the phase
phase=unwrap(pha);   % Unwrap the phase
plot(phase);       % Plot the phase
```

Investigation

Generate N=256 samples of a sinusoidal signal of amplitude 1 volts, frequency, f_1=1Hz. Use a sampling frequency, F_s=200Hz, and call this signal the transmitted signal, x(t).(i.e. $x(t) = sin(2\pi f_1 t)$)

Generate a received signal r(t) as follows:
r(t) is also a sinusoidal of the same frequency whose amplitude has been attenuated by 0.5 and delayed by 2.5 Seconds. Add white noise with zero mean and a variance of 0.01. (i.e. $r(t) = 0.5\, sin(2\pi f_1(t - 2.5)) + w(t)$)

Plot x(t) and y(t).

Estimate and plot:

- The spectrum of the transmitted signal and the received signal.
- The autocorrelation of x(t) and r(t).
- The cross correlation between x(t) and r(t).

From this plot estimate the delay time (Time where it peaks) and compare to the theoretical value of 2.5s.

Estimate the attenuation factor using:

$$\alpha = \frac{R_{xr}(D)}{R_{xx}(1)} = \frac{\textit{Value of the peak at } \tau = D}{\textit{The peak value of the Autocorrelation of } x(t)}$$

Compare this value to the theoretical value of 0.5.

- Repeat the experiment for the noise variance of 0, 0.1,0.4, and 0.8.
- Estimate the error in each case.
- The cross spectral density and phase between x(t) and r(t).

From the phase plot, estimate the slope of the plot in the following table:

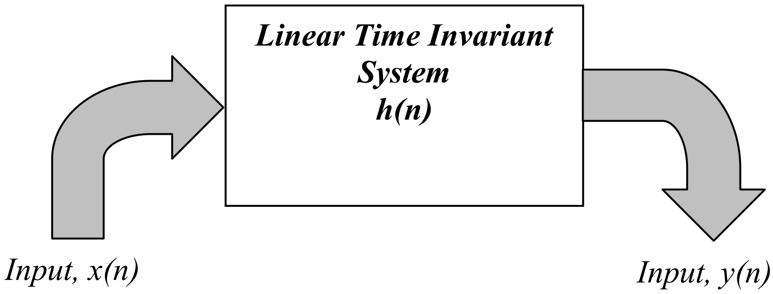
N	**Slope**
128	
256	
512	
1024	

What is the relationship between the time delay and the slope of the phase?

[Note: That yet another method of estimating the time delay is via a system identification approach. This method is described in the appendix for those interested.]

Application of Correlation Functions in System Analysis

Theorem: For a linear time-invariant system:



"The cross correlation of the output ,y(n) and the input, x(n) [i.e. $R_{yx}(m)$] is equal to its impulse response when the input is white noise". i.e.

$$R_{yx}(m) = h(n) \quad \text{when } x(n) \text{ is zero mean unit variance white Gaussian Noise}$$

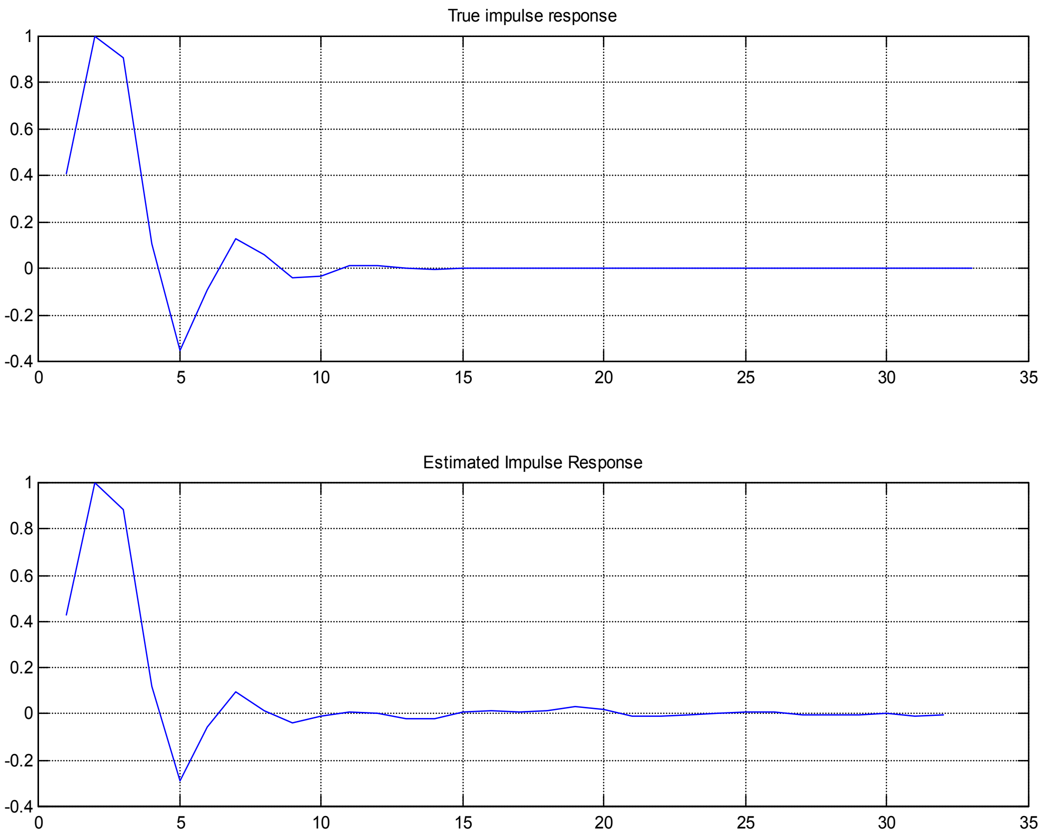
The objective of this part of the case study is to illustrate this property.

Here is a Matlab program which performs the above simulation:

```
% M_ex4
% By: saeed Reza Taghizadeh [January 2000]
% Application of cross correlation in System Analysis

% Defining a simple low pass filter
FS=2500;
fHz=[0 250 500 750 1000 1250];
m0=[1 1 1 0 0 0];
FH=fHz/(FS/2);
[b,a]=yulewalk(4,FH,m0);
[h,f]=freqz(b,a,1024);
% Now find the impulse response
N=32;
delta=[1,zeros(1,N)]; % define an impulse
h=filter(b,a,delta);     % apply the impulse as input
subplot(2,1,1);
plot(h/max(h));       % Plot the normalize impulse response
title('True impulse response');
```

```
grid;
% Now use the cross correlation

% Define the input
for i=1: 128            % Performs the experiment 128 times

   x=randn(1,N);        % Generate white noise
   y=filter(b,a,x);     % Apply it to the filter to find the
output
   ryx=xcorr(y,x);      % Estimate the cross correlation
   Ryx=fliplr(ryx(1:N)); % Get the appropriate portion
   PA(:,i)=Ryx';         % Save the results
end;
clear Ryx;               % Clear the variable Ryx
Ryx=sum(PA')/128;        % Average over all the runs
subplot(2,1,2);
plot(Ryx/max(Ryx));      % Plot the estimated impulse response
title('Estimated Impulse Response');
grid;
```

Which produces the following output:



Note that the cross correlation technique produces a good estimate of the impulse response of the system.

Note also that since the input is white noise and being random, the experiment will not be valid for simply one run, hence as shown in the program, the simulation is carried out for 128 times and the final result is the average of the all the runs. When simulating with random processes, you must perform a simulation over a number of runs and average the results for the final outcome.

Investigation

A digital filter is described by the following transfer function:
One of the most simplest and yet effective digital filters are 'avergers'. This is a system which computes the average of previous q-samples of its input. For example a 5-point averager is defined as:

$$y(n) = \frac{1}{5}\left[x(n) + x(n-1) + x(n-2) + x(n-3) + x(n-4)\right]$$

For this system:

- Draw the filter structure.
- Use Matlab to plot its frequency response. Use:

```
a=1;
b=[1/5, 1/5, 1/5, 1/5, 1/5];
[h,f]=freqz(b,a,256);
H=abs(h);
f=f/(2*pi);
plot(f,H);
```

- Plot its impulse response. Use:

```
N=128;
delta=[1, zeros(1,N-1)];
ht=filter(b,a,delta);
plot(ht);
```

- Use the cross correlation procedure illustrated above to estimate its impulse response by applying white noise with zero mean and unit variance and plot its response (call this the estimated response, he).

- Plot the error as *E= The true response (i.e ht) – The estimated impulse response (i.e. he).*

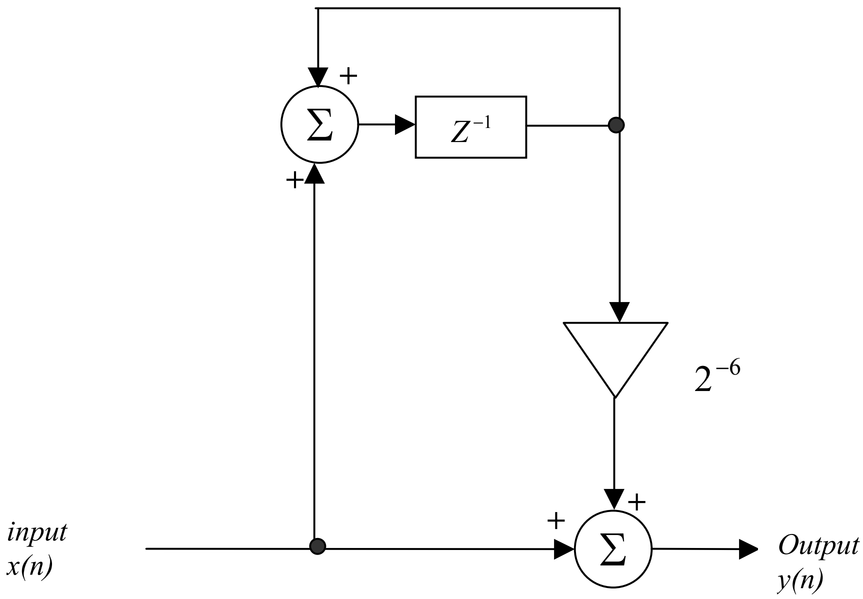
- Comment on your results.

Digital filters play an important role in every digital audio or video equipment. Some simple digital filters are to be examined in this section both analytically and via simulation.

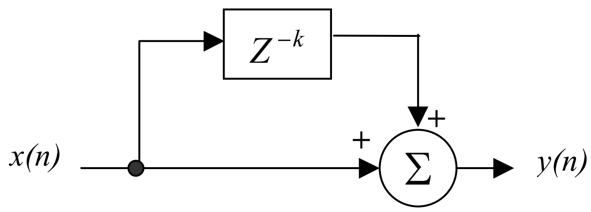
Investigation

For each of the following systems,:

- Determine the difference equation
- Determine the transfer function $H(z)$ of the system, plot the frequency and phase response using Matlab.
- Determine the poles and zeros diagram. [Use Matlab to confirm]
- Determine the impulse response of the system. [Use Matlab to confirm]

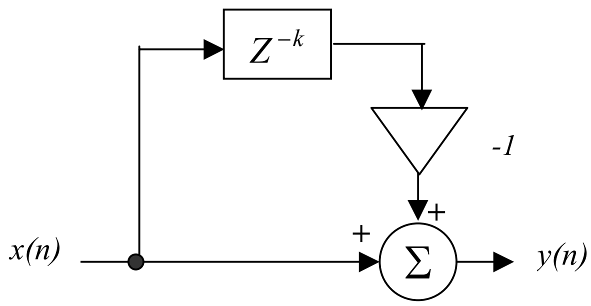


[Case Number 1]



K=4, 5, 6

[Case Number 2]



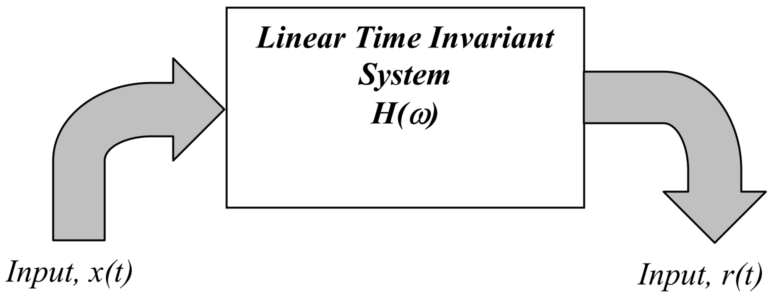
K=4, 5, 6

[Case Number 3]

Appendix A

System Identification approach in Time Delay estimation

Here, we assume that the transmitted signal r(t) is the output of a linear time-invariant system whose input is the transmitted signal x(t). Another word, we model a system whose transfer function characteristic processes x(t) in a way as to produce the transmitted signal r(t) in its output. Once this is done, the information concerning the estimation of delay (i.e. D) may be extracted from its transfer function. Here is the mathematical analysis of the above procedure.



From the theory of cross correlation, we have:

$$P_{xr}(\omega) = H(\omega).P_{xx}(\omega)$$

Which states that "The cross spectrum of the input-output, $P_{xr}(\omega)$ is equal to the transfer function $H(\omega)$ times the power spectrum of the input signal, $P_{xx}(\omega)$."

We therefore have:

$$H(\omega) = \frac{P_{xr}(\omega)}{P_{xx}(\omega)}$$

Having obtained the transfer function of the model, the followings may be obtained:

$$\alpha = |H(\omega)|$$
$$\phi(\omega) = \angle H(\omega) = \angle P_{xr}(\omega)$$

The time delay *(i.e.D)* may be obtained from the slope of the phase angle $\phi(\omega)$ as before.

.