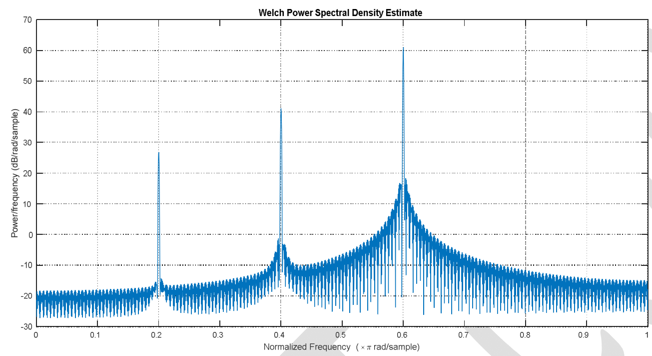
P1: Plotting PSD of known sinusoid:

Code:

```
fs=1000; % the sampling frequency
dt = 1/fs;
t = dt:dt:10000*dt; % to gather signal data till 10000th sample
cosine = 2*cos(2*pi*100*t); % a cosine wave with 100Hz frequency
cosine1 = 10*cos(2*pi*200*t); % a cosine wave with 200Hz frequency
sine = 100*sin(2*pi*300*t); % a sine wave with 300Hz frequency
y = cosine + cosine1 + sine; % adding the above 3 signal
pwelch(y,[], [], [], 'psd') % plotting the PSD using welch method
```

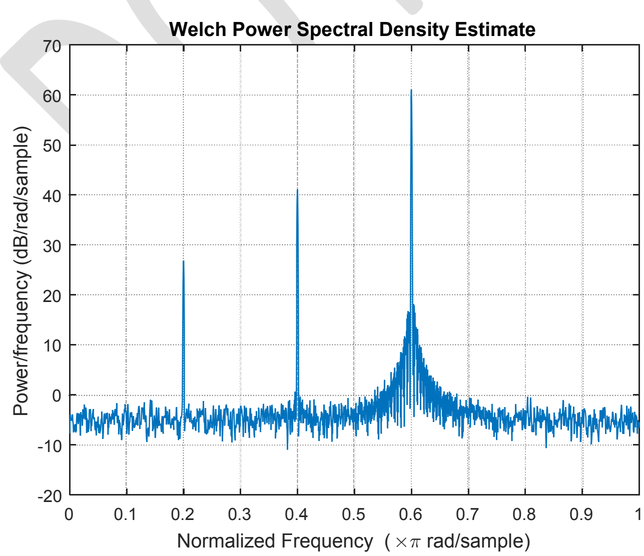
Output:



P2: Plotting PSD of known sinusoid with AWGN:

Code:

```
fs=1000; % the sampling frequency
dt = 1/fs;
t = dt:dt:10000*dt; % to gather signal data till 10000th sample
cosine = 2*cos(2*pi*100*t); % a cosine wave with 100Hz frequency
cosine1 = 10*cos(2*pi*200*t); % a cosine wave with 200Hz frequency
sine = 100*sin(2*pi*300*t); % a sine wave with 300Hz frequency
y = cosine + cosine1 + sine; % adding the above 3 signal
y=awgn(y,0); % adding AWGN to the resultant summed up signal
pwelch(y,[], [], [], 'psd') % plotting the PSD using welch method
```

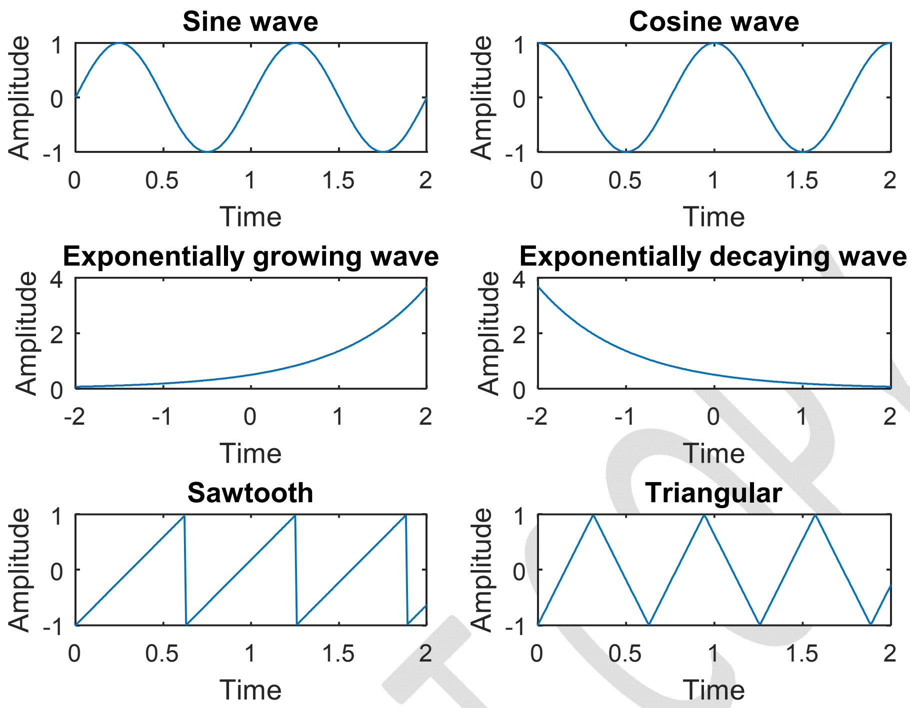


P3: Generation of Continuous Time Signals

Code:

```
%Generation of CTS
clearall;
closeall;
clc;
%Generation of sine wave
t=0:0.01:2;
x=sin(2*pi*t);
subplot(3,2,1)
plot(t,x);
xlabel('Time');
ylabel('Amplitude');
title('Sine wave');
%Generation of cosine wave
t=0:0.01:2;
x=cos(2*pi*t);
subplot(3,2,2)
plot(t,x);
xlabel('Time');
ylabel('Amplitude');
title('Cosine wave');
%Generation of exponentially growing wave
t=-2:0.01:2;
a=0.5;
x=a*exp(t);
subplot(3,2,3)
plot(t,x);
xlabel('Time');
ylabel('Amplitude')
title('Exponentially growing wave');
%Generation of exponentially decaying wave
t=-2:0.01:2;
a=0.5;
x=a*exp(-t);
subplot(3,2,4)
plot(t,x);
xlabel('Time');
ylabel('Amplitude')
title('Exponentially decaying wave');
%saw tooth graph
t=0:0.01:2;
x=sawtooth(10*t);
subplot(3,2,5);
plot(t,x);
xlabel('Time');
ylabel('Amplitude')
title('Sawtooth');
%triangular graph
t=0:0.01:2;
x=sawtooth(10*t,0.5);
subplot(3,2,6);
plot(t,x);
xlabel('Time');
ylabel('Amplitude')
title('Triangular');
```

Output:



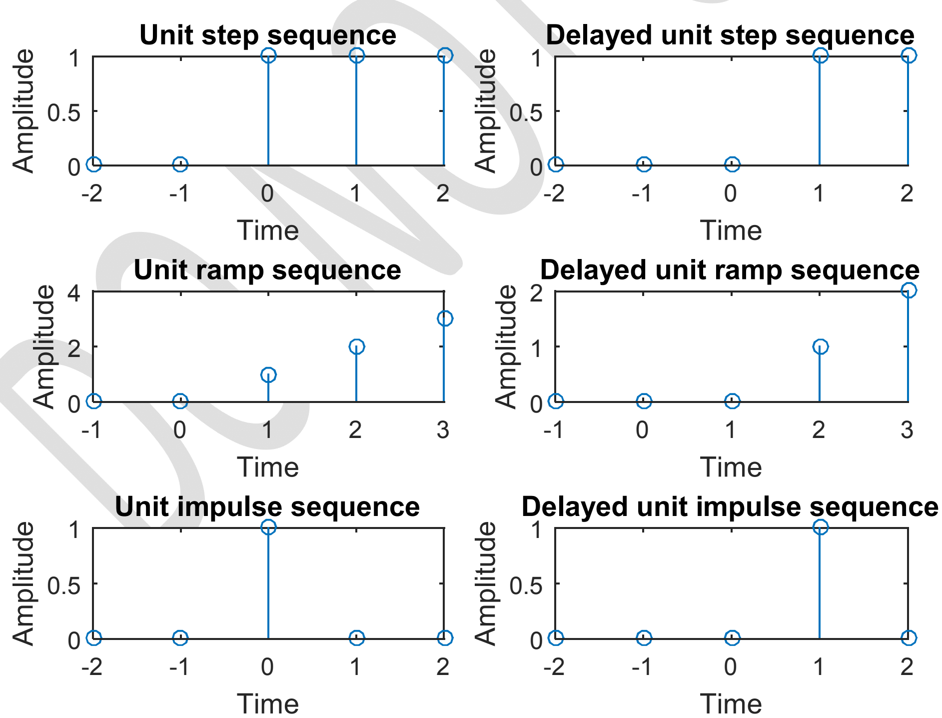
P4:Generation of Discrete Time Signals

Code:

```
%Generation of DTS
clearall;
closeall;
clc;
%Generation of unit step sequence
t=-2:1:2;
x=[0 0 1 1 1];
subplot(3,2,1);
stem(t,x);
xlabel('Time');
ylabel('Amplitude');
title('Unit step sequence');
%Delayed unit step sequence
t=-2:1:2;
x=[0 0 0 1 1];
subplot(3,2,2);
stem(t,x);
xlabel('Time');
ylabel('Amplitude');
title('Delayed unit step sequence');
%Generate unit ramp sequence
t=-1:1:3;
x=[0 0 1 2 3];
subplot(3,2,3);
stem(t,x);
xlabel('Time');
ylabel('Amplitude');
title('Unit ramp sequence');
```

```
%Delayed unit ramp sequence
t=-1:1:3;
x=[0 0 0 1 2];
subplot(3,2,4);
stem(t,x);
xlabel('Time');
ylabel('Amplitude');
title('Delayed unit ramp sequence');
%Generate unit impulse sequence
t=-2:1:2;
x=[0 0 1 0 0];
subplot(3,2,5);
stem(t,x);
xlabel('Time');
ylabel('Amplitude');
title('Unit impulse sequence');
%Delayed unit impulse sequence
t=-2:1:2;
x=[0 0 0 1 0];
subplot(3,2,6);
stem(t,x);
xlabel('Time');
ylabel('Amplitude');
title('Delayed unit impulse sequence');
```

Output:



P5: Autocorrelation of any sequence:

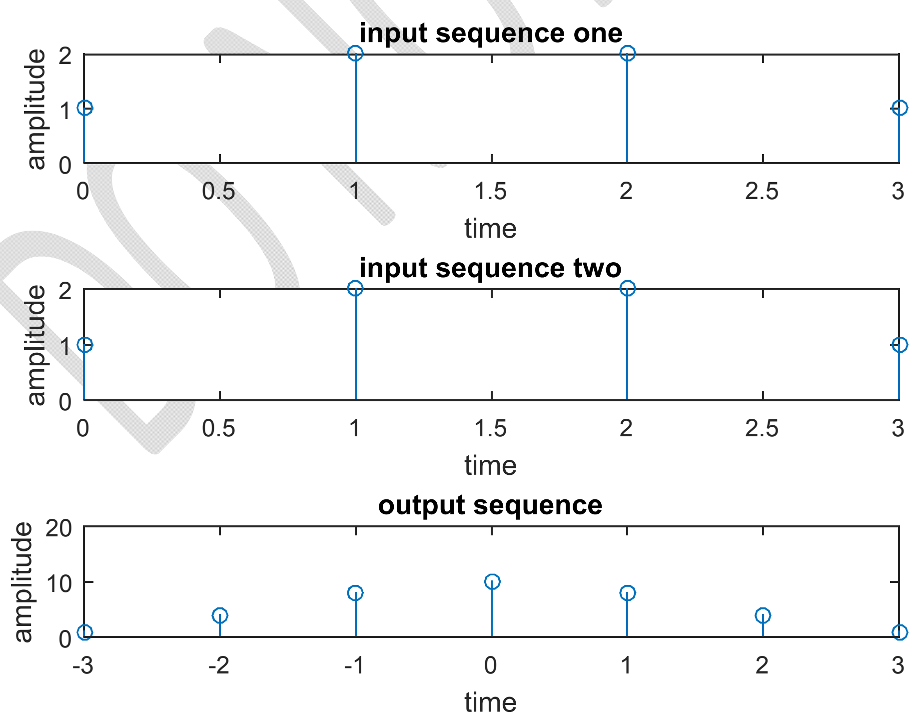
Code:

```
%to perform auto corelation
closeall;
clearall;
clc;
x=input('enter the sequence one');
n=input('enter sequence interval');
h=x;
y=xcorr(x);
p=(min(n)-max(n):1:max(n)-min(n));
subplot(3,1,1);
stem(n,x);
xlabel('time');
ylabel('amplitude');
title('input sequence one');
subplot(3,1,2);
stem(n,h);
xlabel('time');
ylabel('amplitude');
title('input sequence two');
subplot(3,1,3);
stem(p,y);
xlabel('time');
ylabel('amplitude');
title('output sequence');
```

Input:`enter the sequence one[1 2 2 1]`

`enter sequence interval0:1:3`

Output:



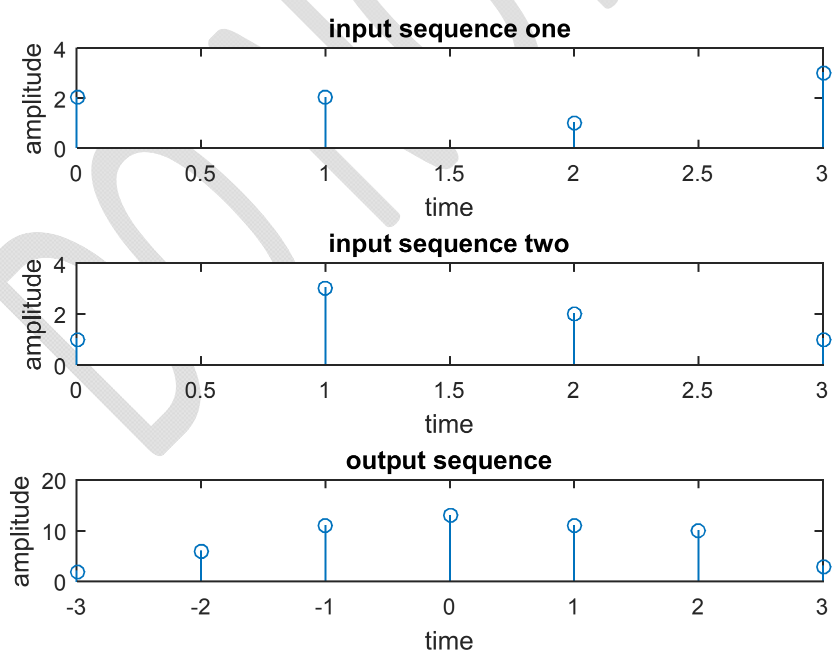
P6: Cross-correlation of any two sequences:

Code:

```
%to perform cross correlation
closeall;
clearall;
clc;
x=input('enter the sequence one');
n=input('enter sequence interval');
h=input('enter the sequence two');
m=input('enter sequence interval');
y=xcorr(x,h);
p=(min(n)-max(m):1:max(n)-min(m));
subplot(3,1,1);
stem(n,x);
xlabel('time');
ylabel('amplitude');
title('input sequence one');
subplot(3,1,2);
stem(m,h);
xlabel('time');
ylabel('amplitude');
title('input sequence two');
subplot(3,1,3);
stem(p,y);
xlabel('time');
ylabel('amplitude');
title('output sequence');
```

Input: enter the sequence one[2 2 1 3]enter sequence interval0:1:3enter the sequence two[1 3 2 1]
enter sequence interval0:1:3

Output:

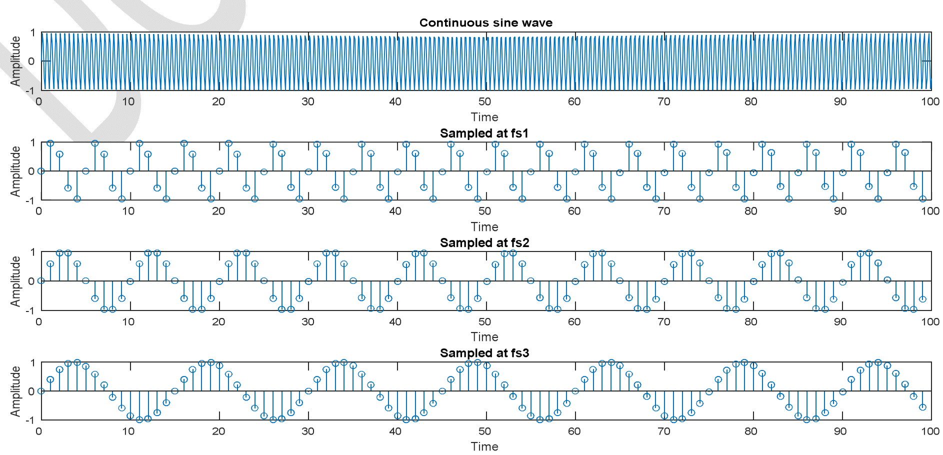


P7: Sampling of Continuous waveform:

```
clearall;
closeall;
clc;
f=input('Enter frequency');
%T=1/f;
fs1=input('Enter the sampling frequency fs1');
fs2=input('Enter the sampling frequency fs2');
fs3=input('Enter the sampling frequency fs3');
t=0:0.1:100;
t1=0:1:99;
x=sin(2*3.14*f*t);
subplot(4,1,1);
plot(t,x);
xlabel('Time');
ylabel('Amplitude');
title('Continuous sine wave');
y=sin(2*3.14*f*t1/fs1);
subplot(4,1,2);
stem(t1,y);
xlabel('Time');
ylabel('Amplitude');
title('Sampled at fs1');
y=sin(2*3.14*f*t1/fs2);
subplot(4,1,3);
stem(t1,y);
xlabel('Time');
ylabel('Amplitude');
title('Sampled at fs2');
y=sin(2*3.14*f*t1/fs3);
subplot(4,1,4);
stem(t1,y);
xlabel('Time');
ylabel('Amplitude');
title('Sampled at fs3');
```

Input:

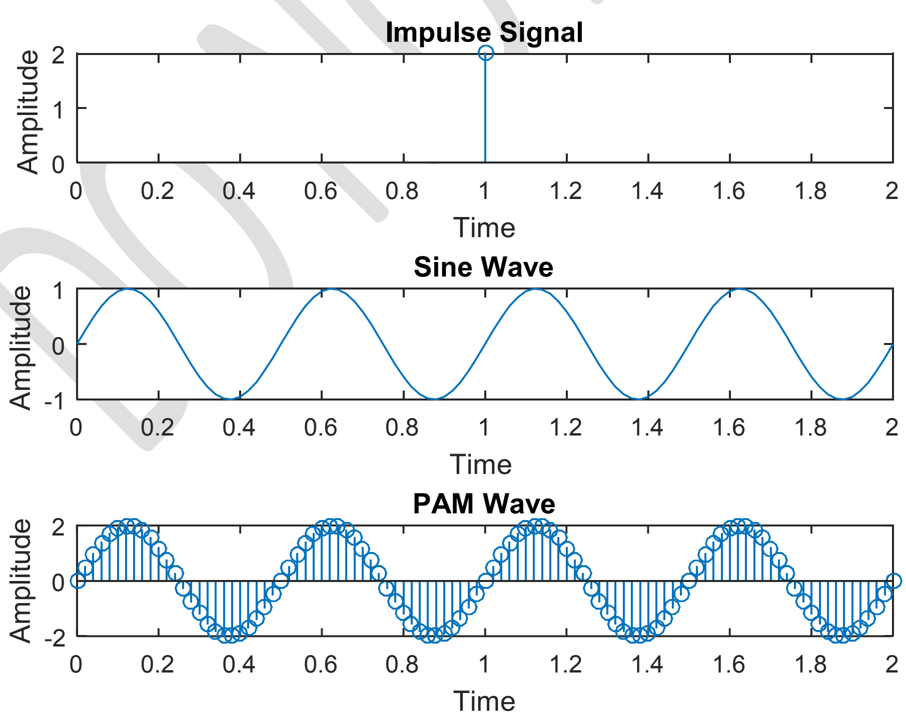
Enter frequency2 ;Enter the sampling frequency fs110 ; Enter the sampling frequency fs220
Enter the sampling frequency fs330 ;Output:



P8: PAM

```
clc;
closeall;
clearall;
a = input('Enter the amplitude = ');
f = input('Enter the frequency  = ');
t = 0:0.02:2; % for a total of 20 samples
x1 = 2:1:2; %generation of impulse signal
x2 = sin(2*pi*f*t); %generation of sine wave
y = x1.*x2; %modulation step
subplot(3,1,1); %for impulse signal plot
stem(x1);
title('Impulse Signal');
xlabel('Time');
ylabel('Amplitude ');
subplot(3,1,2) %for sine wave plot
plot(t,x2);
title('Sine Wave');
xlabel('Time ');
ylabel('Amplitude ');
subplot(3,1,3) %for PAM wave plot
stem(t,y);
title('PAM Wave');
xlabel('Time');
ylabel('Amplitude');
```

```
Input:
Enter the amplitude = 2 ; Enter the frequency  = 2

Output:
```

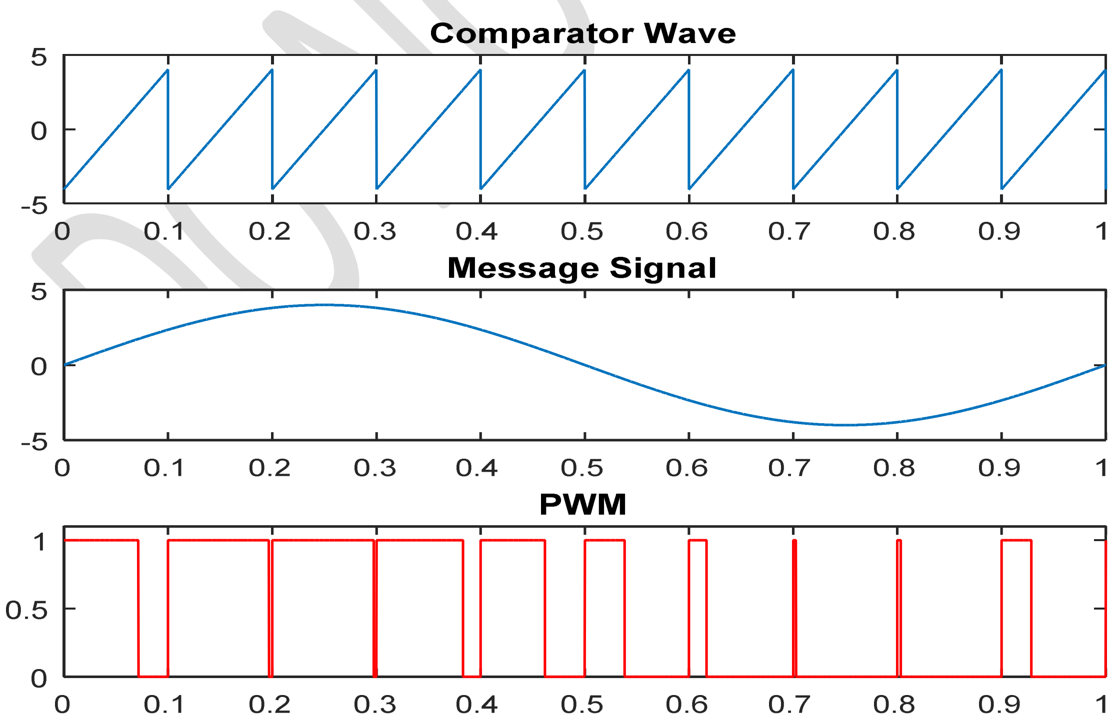


P9: PWM

```
fs=input('Comparator Sawtooth frequency:');
fm=input('Message frequency(Assuming it to be a sine wave):');
a=input('Enter Amplitude of Message:');
 t=0:0.0001:1; %sampling rate of 10kHz
stooth=1.01*a.*sawtooth(2*pi*fs*t); %generating a sawtooth wave
%to make the two non zero lobes of pwm not to overlap the amplitude of
%sawtooth wave must be atleast more than a bit to the message amplitude
subplot(3,1,1);
plot(t,stooth); % plotting the sawtooth wave
title('Comparator Wave');
msg=a.*sin(2*pi*fm*t); %generating message wave
subplot(3,1,2);
plot(t,msg); %plotting the sine message wave
title('Message Signal');
fori=1:length(stooth)
if (msg(i)>=stooth(i))
pwm(i)=1; %is message signal amplitude at ith sample is greater than
%sawtooth wave amplitude at ith sample
else
pwm(i)=0;
end
end
subplot(3,1,3);
plot(t,pwm,'r');
title('PWM');
axis([0 1 0 1.1]); %to keep the pwm visible during plotting.
```

Input:Comparator Sawtooth frequency:10 ; Message frequency(Assuming it to be a sine wave):1 ; Enter Amplitude of Message:4

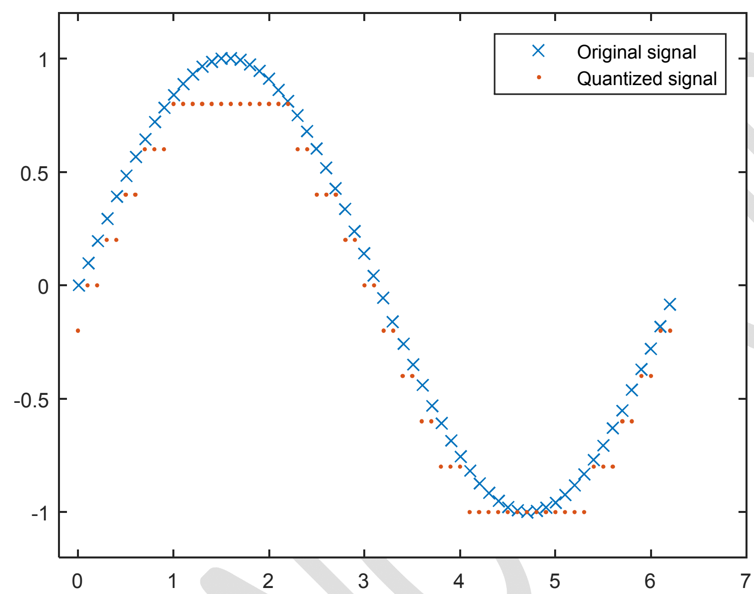
Output:



P10: Linear Quantization

```
t = [0:.1:2*pi]; % Times at which to sample the sine function
sig = sin(t); % Original signal, a sine wave
partition = [-1:.2:1]; % Length 11, to represent 12 intervals
codebook = [-1.2:.2:1]; % Length 12, one entry for each interval
[index,quants] = quantiz(sig,partition,codebook); % Quantize.
plot(t,sig,'x',t,quants,'.')
legend('Original signal','Quantized signal');
axis([-.2 7 -1.2 1.2])
```

Output:



P11: Quantization Process for Sine/Sawtooth/Random

```
% This script creates a signal, and then quantizes it to a specified number
% of bits.  It then calculates the quantization error.
% see if you run the script.

fprintf('\nE71 Lab, Sampling and Quantization\n');

b=3;                                % Number of bits.
N=120;                              % Number of samples in final signal.
n=0:(N-1);                     %Index

% Choose the input type.
choice = questdlg('Choose input','Input',...
'Sine','Sawtooth','Random','Random');

fprintf('Bits = %g, levels = %g, signal = %s.\n', b, 2^b, choice);

% Create the  input data sequence.
switch choice
case'Sine'
```

```
 x=sin(2*pi*n/N);
case'Sawtooth'
        x=sawtooth(2*pi*n/N);
case'Random'
        x=randn(1,N);        % Random data
        x=x/max(abs(x));     % Scale to +/- 1
end

% Signal is restricted to between -1 and +1.
x(x>=1)=(1-eps);             % Make  signal from -1 to just less than 1.
x(x<-1)=-1;

% Quantize a signal to "b" bits.
xq=floor((x+1)*2^(b-1));     % Signal is one of 2^n int values (0 to 2^n-1)
xq=xq/(2^(b-1));             % Signal is from 0 to 2 (quantized)
xq=xq-(2^(b)-1)/2^(b);       % Shift signal down (rounding)

xe=x-xq;                     % Quantization error

stem(x,'b');
holdon;
stem(xq,'r');
holdon;
stem(xe,'g');
legend('exact','quantized','error','Location','Southeast')
title(sprintf('Signal, Quantized signal and Error for %g bits, %g
quantization levels',b,2^b));
holdoff
```

P12: Quantization

```
% This script creates a random signal, and then quantizes it.  The signal
% is oversampled and then decimated.
% * Oversampling is the process of taking in samples at a faster rate (in
% this script "os" times faster) than you need.
% * Decimation is the process Decimation is the process of taking only one
% of every "os" samples of an oversampled signal to get the final sampling
% rate.
%
% Processing of the oversampled signal can give some benefit, as you will
% see if you run the script.

fprintf('\n\nE71 Lab, Oversampling and Quantization\n');

b=3;                             % Number of bits.
N=100;                           % Number of samples in final signal.

% Choose the input type.
choice = questdlg('Choose input','Input',...
'Sine','Sawtooth','Random','Random');

fprintf('Bits = %g, levels = %g, signal = %s.\n', b, 2^b, choice);

% This large loop generates and analyzes data at several different
% oversampling rates (all powers of two).
foros_pow=0:4
os=2^os_pow;                     % Oversampling rate.
```

```
N_os=N*os;                    % Number of samples in oversampled signal
    n=0:(N_os-1);                 % Index

% Create the oversampled input data sequence.
switch choice
case'Sine'
            x=sin(2*pi*n/N_os);
case'Sawtooth'
            x=sawtooth(2*pi*n/N_os);
case'Random'
            x=randn(1,N_os);          % Random data
% Smooth to begin to remove fast variations.
            x=filter(ones(1,4*os)/4/os,1,x);
            x=x/abs(max(x));          % Scale to +/- 1
end

% Signal is restricted to between -1 and +1.
x(x>=1)=(1-eps);         % Make  signal from -1 to just less than 1.
x(x<-1)=-1;

%Quantize the oversampled raw signal
xq=floor((x+1)*2^(b-1));    %Signal is one of 2^b int values (0 to 2^b-1)
xq=xq/(2^(b-1));            %Signal is from 0 to 2 (quantized)
xq=xq-(2^(b)-1)/2^(b);      %Shift signal down (rounding)

%Smooth (running average) the quantized oversampled signal
x_qs=filter(ones(1,os)/os,1,xq);

%Smooth the oversampled signal
x_s=filter(ones(1,os)/os,1,x);

%Quantize the oversampled smoothed signal
x_sq=floor((x_s+1)*2^(b-1)); % Signal is one of 2^n int values (0 to 2^n-1)
x_sq=x_sq/(2^(b-1));         % Signal is from 0 to 2 (quantized)
x_sq=x_sq-(2^(b)-1)/2^(b);   % Shift signal down (rounding)

% Resample at lower rate (decimation)
x_sq=x_sq(1:os:end);        % Quant noise, smoothed, then quantized.
x_s=x_s(1:os:end);          % Smoothed signal
x_qs=x_qs(1:os:end);        % Quant noise, quantized, then smoothed.

xe_qs=x_s-x_qs;             % Error between smoothed and quantized/smoothed
xe_sq=x_s-x_sq;             % Error between smoothed and smoothed/quantized

subplot(211)
stem(x_s,'b');
holdon;
stem(x_sq,'r');
holdon;
stem(xe_sq*2^(b-1),'g');
legend('exact','quantized','error*2^{(b-1)}','Location','Southeast')
axis([0 N -2 2]);


title(sprintf('Smoothed then quantized, %g bits, %g levels, os=%g ',b,2^b,
os));
holdoff
```

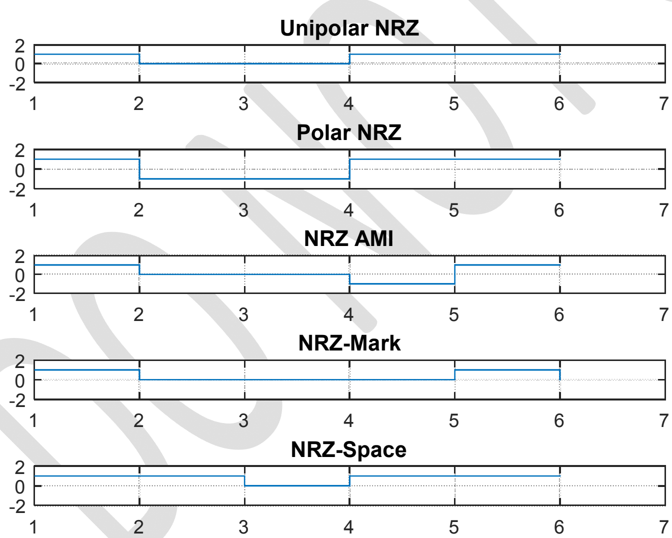
```
subplot(212)
stem(x_s,'b');
holdon;
stem(x_qs,'r');
holdon;
stem(xe_qs*2^(b-1),'g');
legend('exact','quantized','error*2^{(b-1)}','Location','Southeast')
axis([0 N -2 2]);
title(sprintf('Quantized then smoothed, %g bits, %g levels, os=%g ',b,2^b,
os));
holdoff

sqnr_sq=10*log10(sum(x_s.^2)/sum(xe_sq.^2));
sqnr_qs=10*log10(sum(x_s.^2)/sum(xe_qs.^2));

fprintf('Oversampling = %g, ', os);
fprintf('sqnr_sq = %g, sqnr_qs = %g, sqnr_qs-sqnr_sq = %g\n',...
sqnr_sq, sqnr_qs, sqnr_qs-sqnr_sq);

pause(1);
end
```

P13: Line Coding / PCM Waveform

```
%Pulse data coding techniques
a=[ 1 0 0 1 1];
U=a;
n=length(a);
U(n+1)=U(n);
%POLAR
P=a;
for k=1:n;
if a(k)==0
P(k)=-1;
end
P(n+1)=P(n);
end
%Bipolar
B=a;
f = -1;
for k=1:n;
if B(k)==1;
if f==-1;
B(k)=1; f=1;
else
B(k)=-1; f=-1;
end
end
B(n+1)=B(n);
end
%Mark
M(1)=1;
for k=1:n;
M(k+1)=xor(M(k), a(k));
end
%Space
S(1)=1;
for k=1:n
S(k+1)=not(xor(S(k), a(k)));
end
```

```
%Plotting Waves
subplot(5, 1, 1);
stairs(U)
axis([1 n+2 -2 2])
title('Unipolar NRZ')
gridon
subplot(5, 1, 2);
stairs(P)
axis([1 n+2 -2 2])
title('Polar NRZ')
gridon
subplot(5, 1, 3);
stairs(B)
axis([1 n+2 -2 2])
title('NRZ AMI')
gridon
subplot(5, 1, 4);
stairs(M)
axis([1 n+2 -2 2])
title('NRZ-Mark')
gridon
subplot(5, 1, 5);
stairs(S)
axis([1 n+2 -2 2])
title('NRZ-Space')
gridon
```

Output:

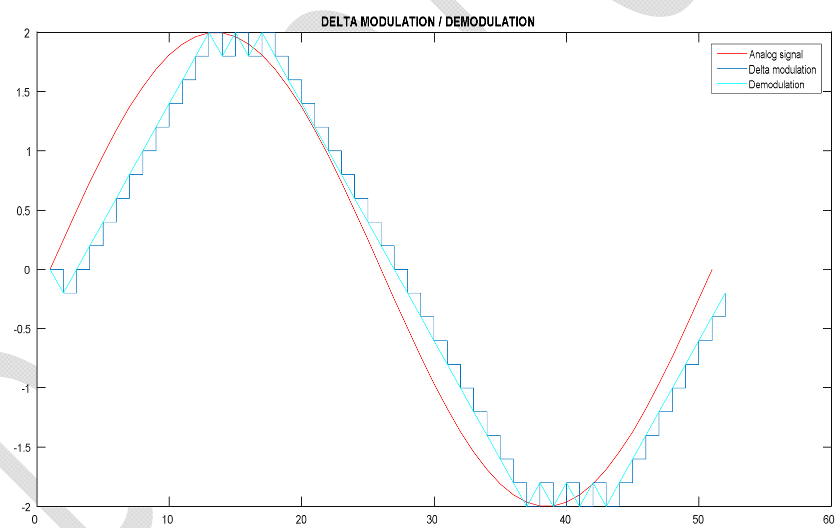


P14: Delta Modulation & Demodulation

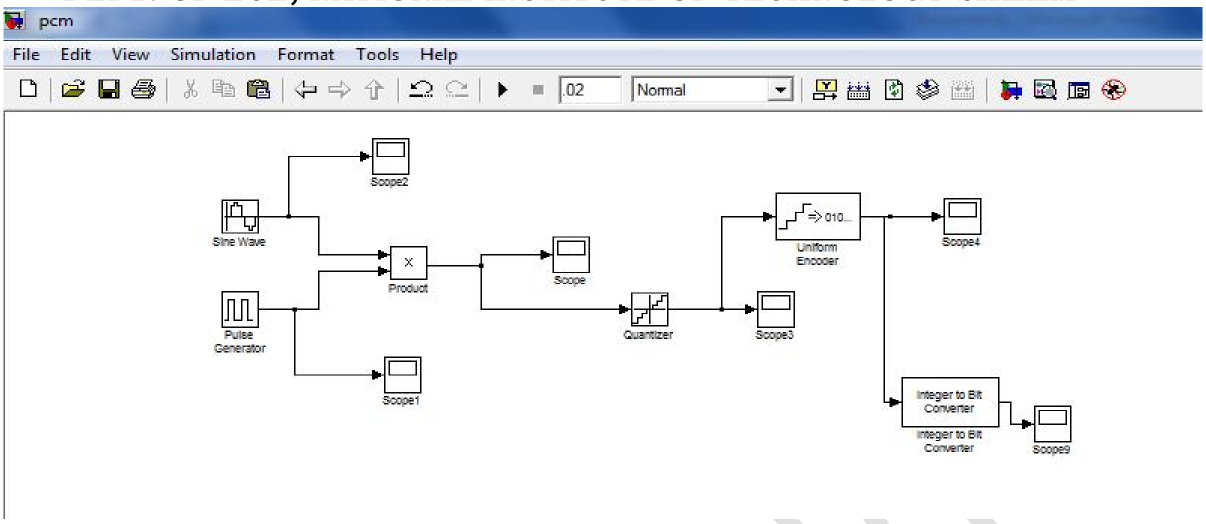
```
clc;
clearall;
closeall;
a=2;
t=0:2*pi/50:2*pi;
x=a*sin(t);
```

```
l=length(x);
plot(x,'r');
delta=0.2;
holdon
xn=0;
fori=1:l;
if x(i)>xn(i)
d(i)=1;
xn(i+1)=xn(i)+delta;
else
d(i)=0; xn(i+1)=xn(i)-delta;
end
end
stairs(xn)
holdon
fori=1:d
if d(i)>xn(i)
d(i)=0;
xn(i+1)=xn(i)-delta;
else
d(i)=1; xn(i+1)=xn(i)+delta;
end
end
plot(xn,'c');
legend('Analog signal','Deltamodulation','Demodulation')
title('DELTA MODULATION / DEMODULATION ')
```

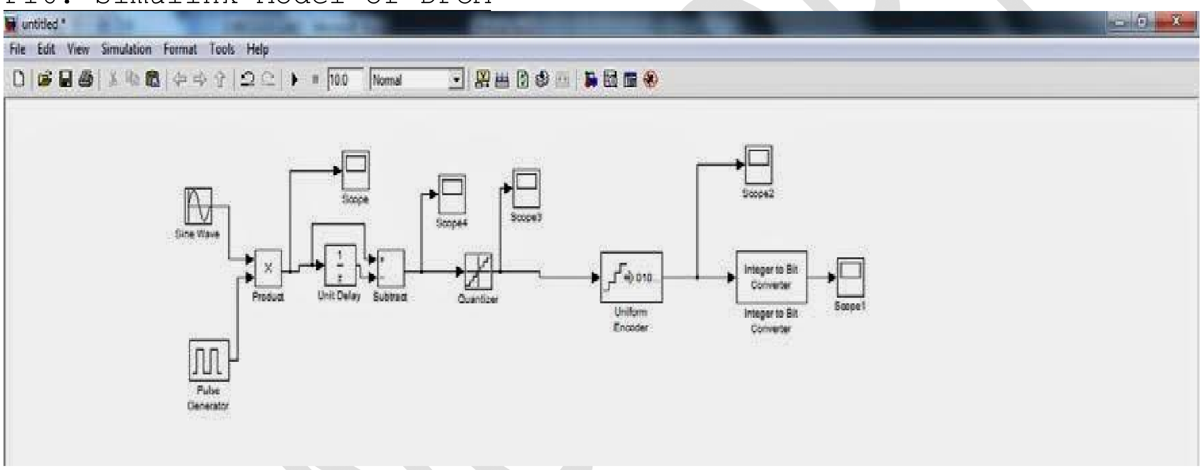
Output:



P15: Simulink Model of PCM



P16: Simulink Model of DPCM



P17: Simulink Model of DM