


# Demo 6.2 – Sampling and Reconstruction (Methods)

XMUT220 Signals and Systems

## Overview

1. Sampling a signal.
2. Zero order and first order sampling methods.
3. Comparison of sampling methods.
4. Upsampling (Interpolation).
5. Downsampling (Decimation).
6. Resampling of signals.

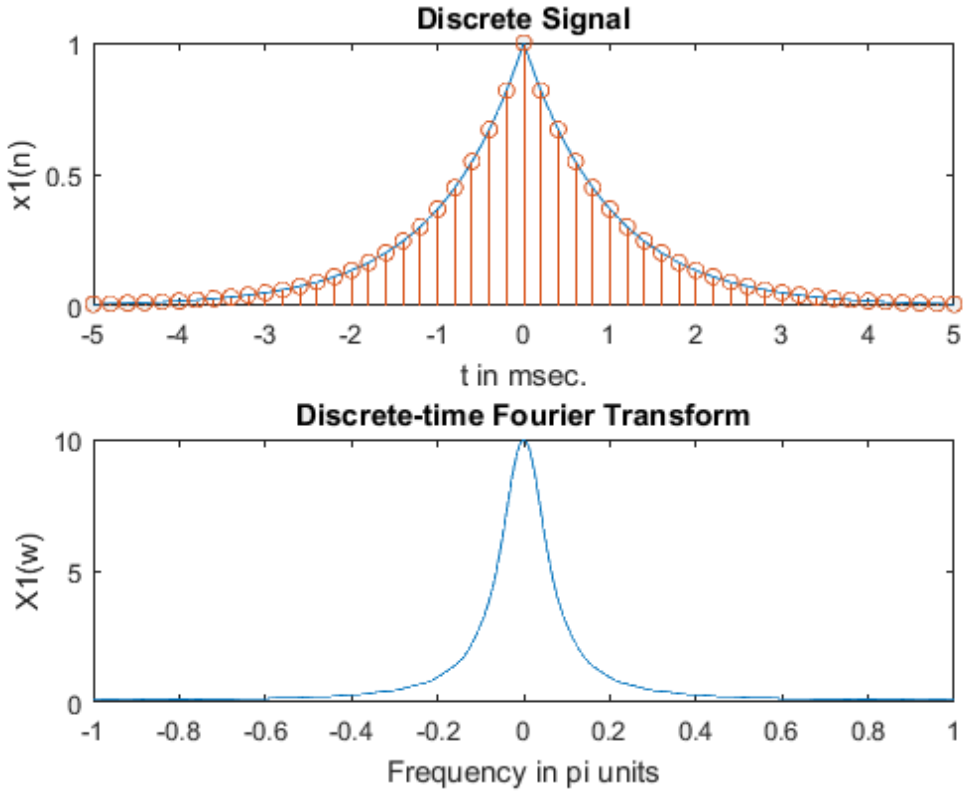
**Exercise 1** - Sampling a Signal

The following MATLAB code shows the algorithm for creating time-domain transient and frequency spectrum of sampling a signal.

```
% samplingmethod1.m

% Analog Signal
Dt = 0.00005;
t = -0.005:Dt:0.005;
xa = exp(-1000*abs(t));

% Discrete-time Signal
Ts = 0.0002;
n = -25:1:25;
x = exp(-1000*abs(n*Ts));

% Discrete-time Fourier transform
K = 500;
k = 0:1:K;
w = pi*k/K;
```

```
X = x * exp(-j*n'*w);
X = real(X);

w = [-fliplr(w),w(2:K+1)];

X = [fliplr(X),X(2:K+1)];

subplot(2,1,1);
plot(t*1000,xa);
xlabel('t in msec.');
ylabel('x1(n)');
title('Discrete Signal');
hold on;
stem(n*Ts*1000,x);
hold off;

subplot(2,1,2);
plot(w/pi,X);
xlabel('Frequency in pi units');
ylabel('X1(w)');
title('Discrete-time Fourier Transform');
```

The following plot shows the resulting time domain transient and frequency spectrum of discrete signals of sampled signals.



**Exercise 2** - Zero-Order Hold and First-Order Hold Sampling Methods

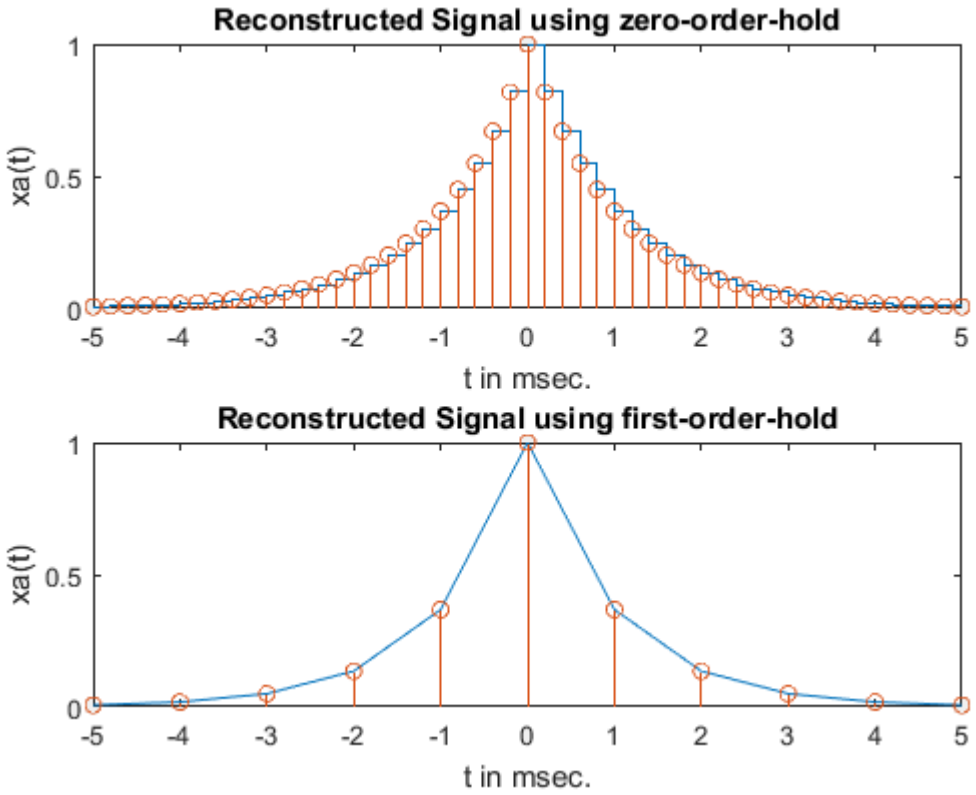
The following MATLAB code shows the algorithm for realising zero-order hold and first-order hold sampling methods.

```
% samplingmethod2.m (zero-order hold and first-order hold
sampling method)

% Discrete-time Signal x1(n) : Ts = 0.0002

Ts = 0.0002;
n = -25:1:25;
nTs = n*Ts;

x = exp(-1000*abs(nTs));
```

```
% Plots
subplot(2,1,1);

stairs(nTs*1000,x);

xlabel('t in msec.');
ylabel('xa(t)')
title('Reconstructed Signal using zero-order-hold');
hold on;
stem(n*Ts*1000,x);
hold off;

% Discrete-time Signal x2(n) : Ts = 0.001
Ts = 0.001;
n = -5:1:5;
nTs = n*Ts;
x = exp(-1000*abs(nTs));
```

```
% Plots

subplot(2,1,2);
plot(nTs*1000,x);

xlabel('t in msec.');
ylabel('xa(t)')
title('Reconstructed Signal using first-order-hold');
hold on;

stem(n*Ts*1000,x);
hold off;
```

The following plot shows the resulting sampling signals of zero-order hold and first-order hold sampling methods.



**Exercise 3** - Comparisons of Perfect, ZOH and FOH Sampling Methods

The following MATLAB code shows time-domain transient response of the comparison between perfect, zero-order hold and first-order hold sampling methods.

For zero-order interpolation, the interpolated samples can be generated by holding the current sample until the new sampling instant. The impulse response of such a system is a rectangular function.

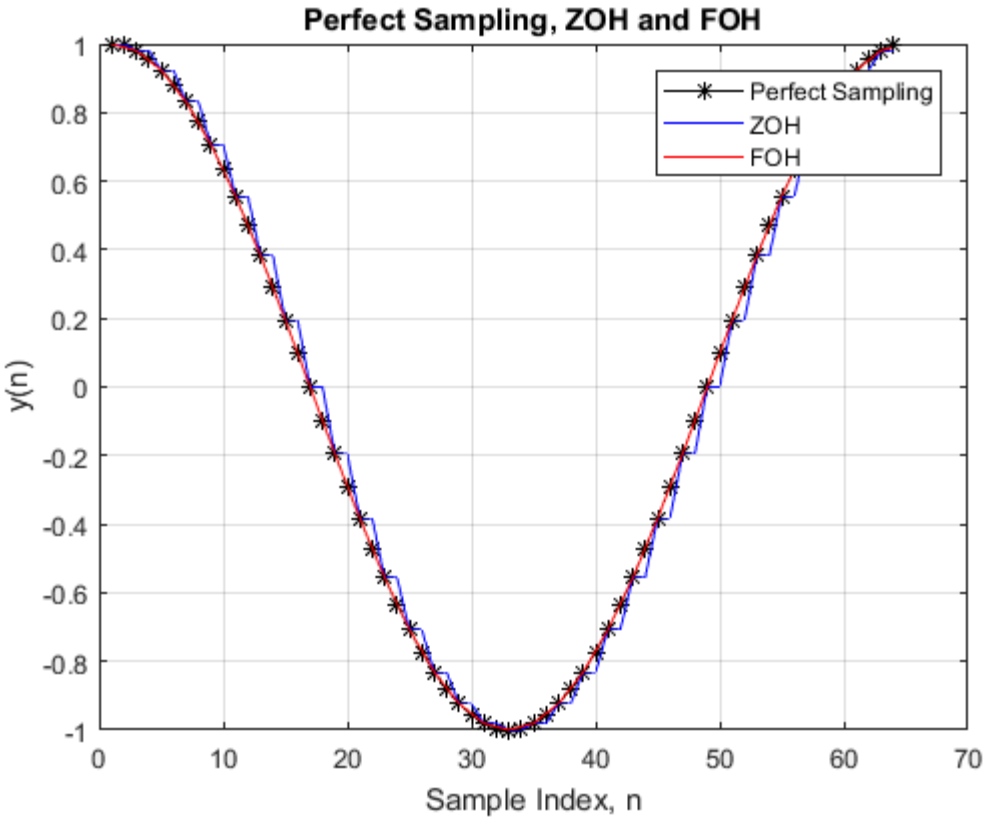
For first-order interpolation with delay, the interpolated samples are generated by piece wise linear interpolation between the current sample and the previous sample. As the prediction based on current sample and previous sample can have 'jumps' at the next sampling instant, delay of one sample is provided. The impulse response of such a system is a triangular function.

For zero-order interpolation, the filter function $h_{ZOH}$ is a rectangular function i.e. $h_{ZOH} = [1\ 1]$. For the delayed first-order interpolation, the filter function $h_{FOH}$is a triangular function i.e.$h_{ZOH} = [1\ 2\ 1]/2$. The factor of 1/2 is for normalization.

```
% samplingmethod3.m (analysing transient response of sampling
methods)

N = 32;

x=cos(2*pi*(1/N)*[0:2*N-1]);

xu=[x;zeros(size(x))];
xu=xu(:).'; % upsampled version of x

h_zoh=[1 1];
h_foh=[1 2 1]/2;
y_zoh=conv(xu,h_zoh); % zero-order hold
y_foh=conv(xu,h_foh); % delayed first order hold
y_pfct=cos(2*pi*(1/(2*N))*[0:2*N-1]); % perfect sine

y_zoh=y_zoh(1:64); % taking one period of the cosine
y_foh=y_foh(2:65); % % one period of cosine with delay removed
```

```
plot(y_pfct,'k-*');
hold on; grid on
plot(y_zoh,'b');
plot(y_foh,'r');
legend('Perfect Sampling', 'ZOH', 'FOH');
xlabel('Sample Index, n');
ylabel('y(n)');
title('Perfect Sampling, ZOH and FOH');
```

The following plot shows the resulting time-domain transient response of the comparison of perfect, zero-order hold and first-order sampling methods.

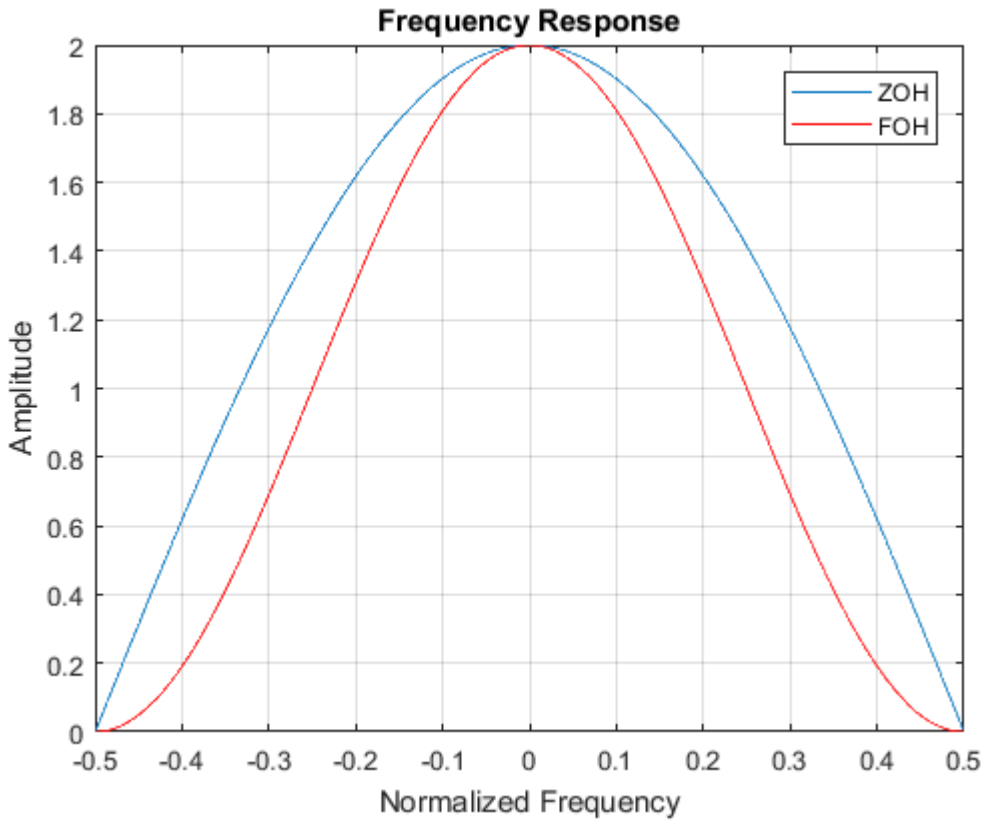


The following MATLAB code outlines frequency response of the comparison between zero-order hold and first-order hold sampling methods.

```
% samplingmethod4.m (analyzing the frequency response of sampling
methods)

h_zoh = [1 1 ];
h_foh = [1 2 1]/2;

H_zoh = fft(h_zoh, 1024); % 1024 pt fft of zero order hold filter
H_foh = fft(h_foh, 1024); % 1024 pt fft of first order hold filter
```

```
figure
plot([-512:511]/1024,fftshift(abs(H_zoh)));
hold on

plot([-512:511]/1024,fftshift(abs(H_foh)),'r');

grid on

xlabel('Normalized Frequency');
ylabel('Amplitude');
title('Frequency Response')
legend('ZOH', 'FOH')
```

The following plot shows the frequency response of the comparison of zero-order hold and first-order sampling methods.



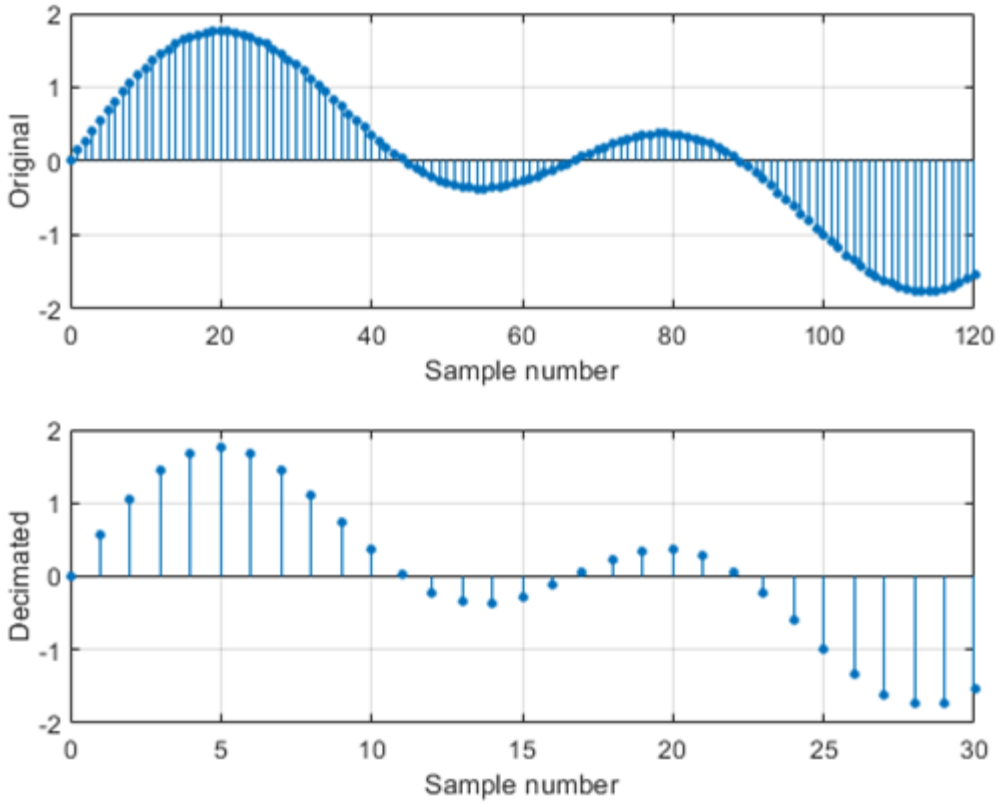
**Exercise 4** – Downsampling (Decimation) of Signal

The following MATLAB code is a sinusoidal signal sampled at 4 kHz decimated by a factor of four.

```
% decimate1.m

t = 0:.00025:1;
x = sin(2*pi*30*t) + sin(2*pi*60*t);

y = decimate(x,4);

% Plot the original and decimated signals.

subplot(2,1,1);
stem(0:120,x(1:121),'filled','markersize',3);
grid on;
xlabel('Sample number');
ylabel('Original');
```

```
subplot(2,1,2);
stem(0:30,y(1:31),'filled','markersize',3);
grid on;
xlabel('Sample number');
ylabel('Decimated')
```

The following plot shows the original and decimated signals.



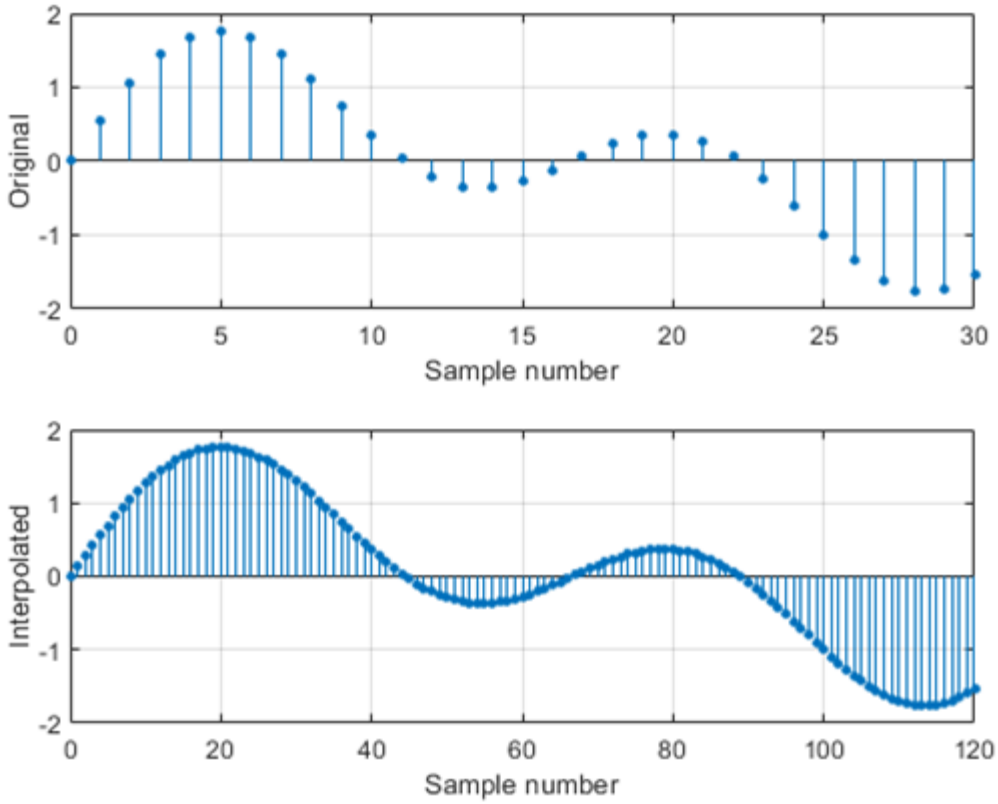
**Exercise 5** – Upsampling (Interpolation) of Signal

The following MATLAB code illustrates a sinusoidal signal sampled at 1 kHz interpolated by a factor of four.

```
% interpolate1.m

t = 0:0.001:1;
x = sin(2*pi*30*t) + sin(2*pi*60*t);

y = interp(x,4);

% plot the original and interpolated signals.

subplot(2,1,1);
stem(0:30,x(1:31),'filled','markersize',3);
grid on;
xlabel('Sample number');
ylabel('Original');
```

```
subplot(2,1,2)
stem(0:120,y(1:121),'filled','markersize',3);
grid on;
xlabel('Sample number');
ylabel('Interpolated');
```

The following plots show the original and interpolated signals.

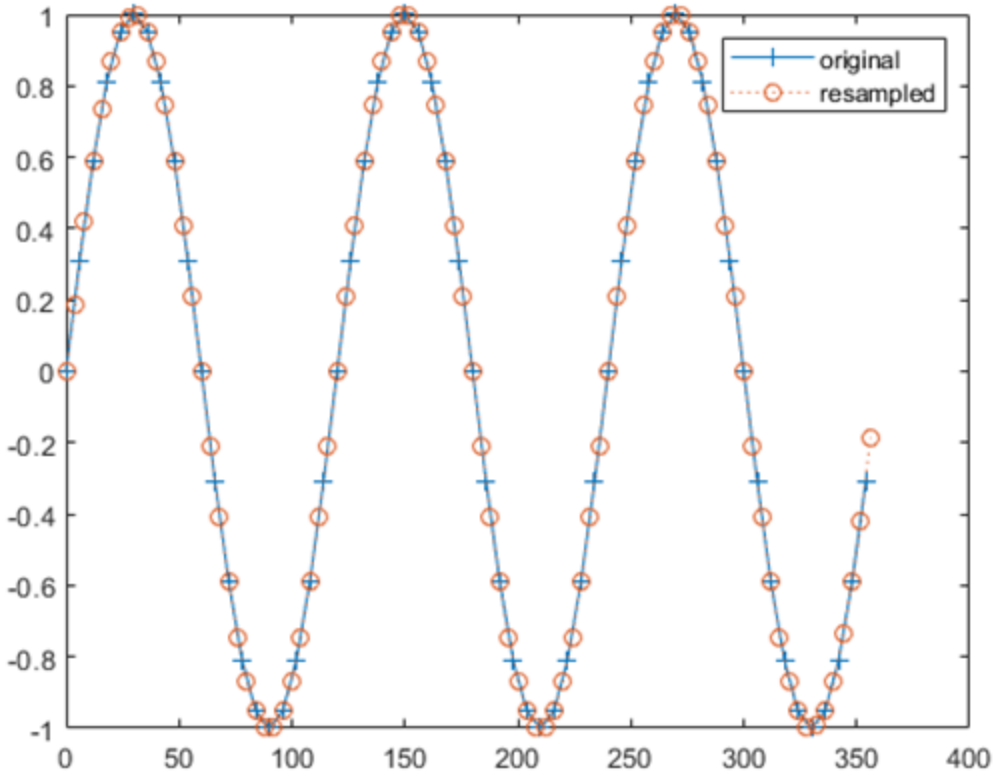


**Exercise 6** – Resampling Signal

The MATLAB code below generates 60 samples of a sinusoid and resample it at 3/2 the original rate.

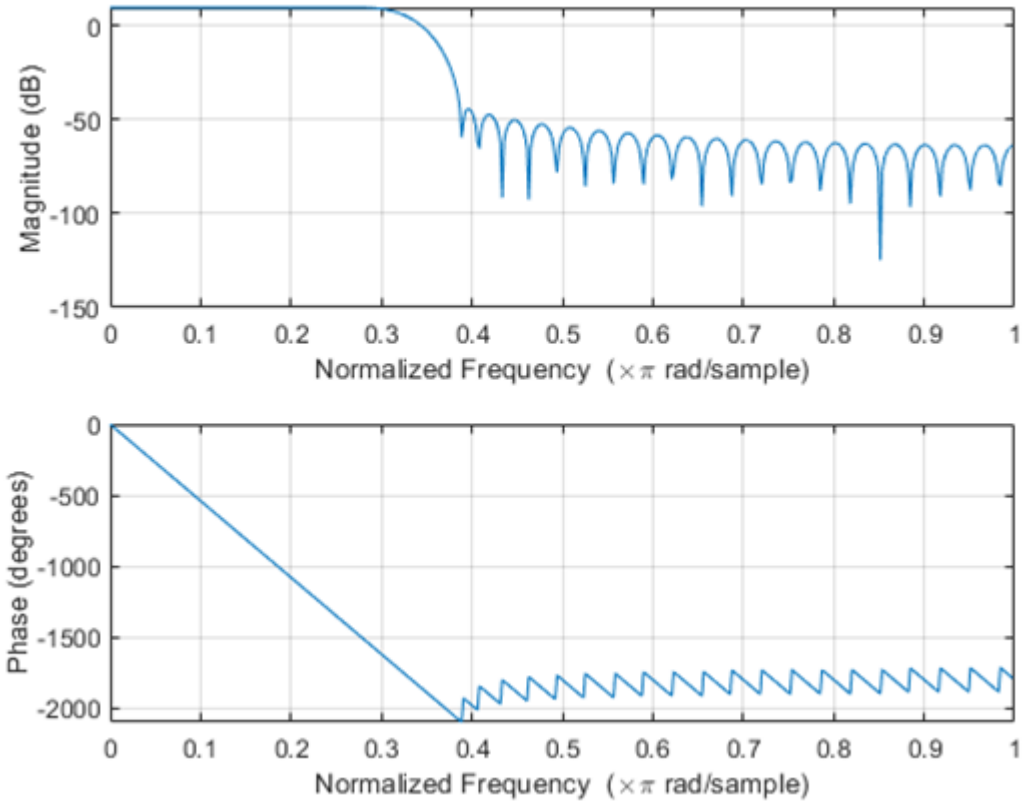
```
tx = 0:6:360-3;
x = sin(2*pi*tx/120);

ty = 0:4:360-2;
[y,by] = resample(x,3,2);

plot(tx,x,'+-',ty,y,'o:')
legend('original','resampled')
```

The plot below displays the original and resampled signals.



Plot the frequency response of the anti-aliasing filter.

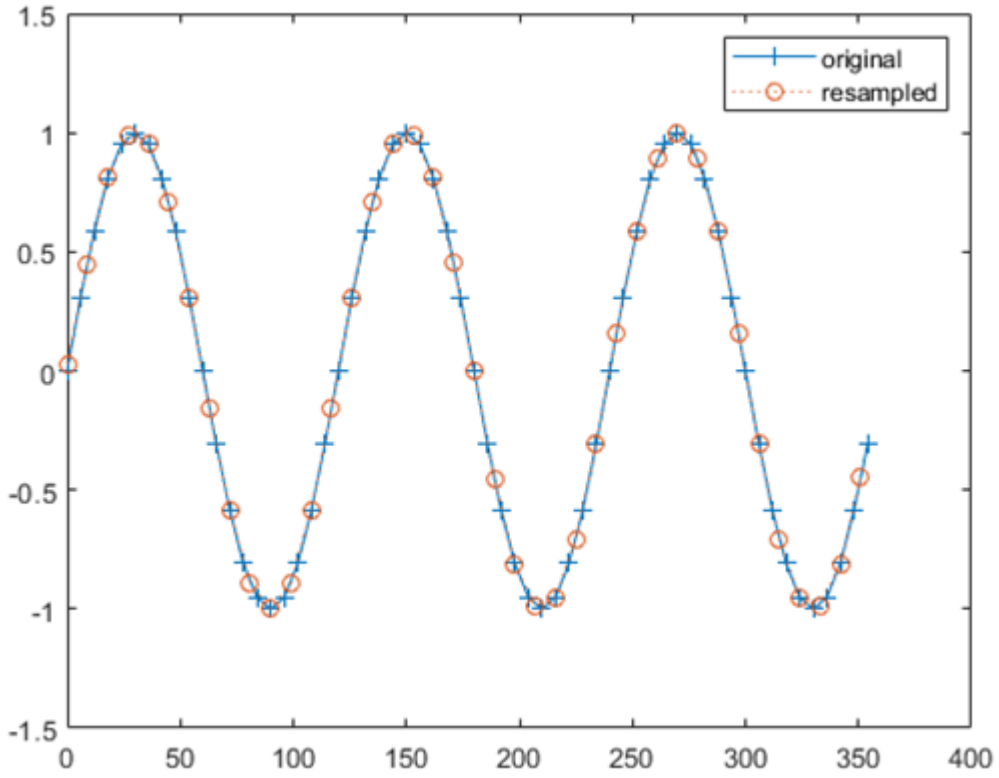
```
freqz(by)
```



The following code resamples the signal at 2/3 the original rate.

```
tz = 0:9:360-9;
[z,bz] = resample(x,2,3);

plot(tx,x,'+-',tz,z,'o:')
legend('original','resampled')
```

The plot given below displays the original signal and its resampling.



Plot the impulse response of the new low-pass filter.

```
impz(bz)
```

