XMUT220 Signals and Systems

Demo 6.2: Sampling and Reconstruction (Methods)

Section A. Methods of Signal Sampling

The following exercises show the constructions of zero-order hold and first-order hold sampling methods and their analysis in MATLAB.

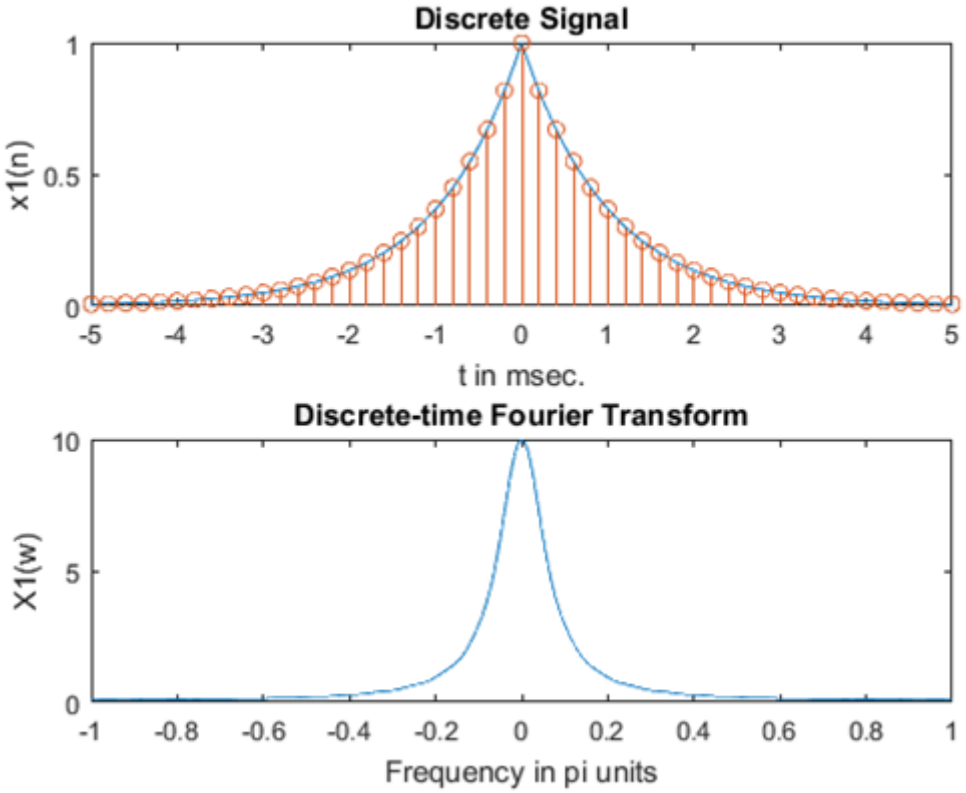
Exercise 1 - Sampling a Signal

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.



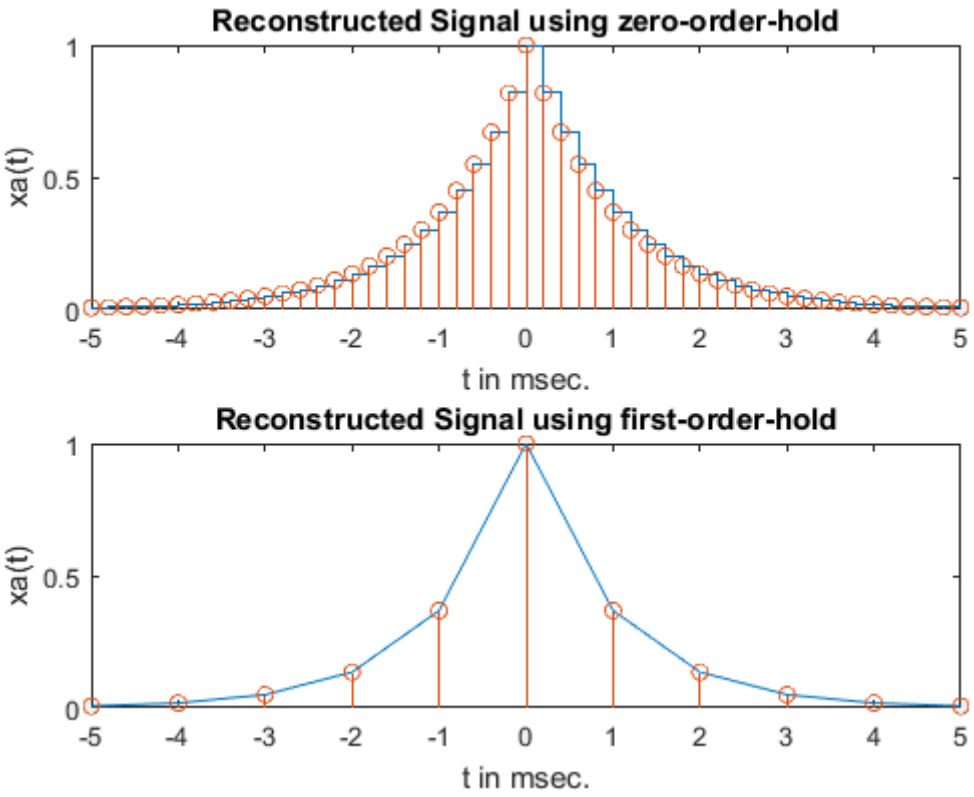
Exercise 2 - Zero-Order Hold and First-Order Hold Sampling Methods

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.

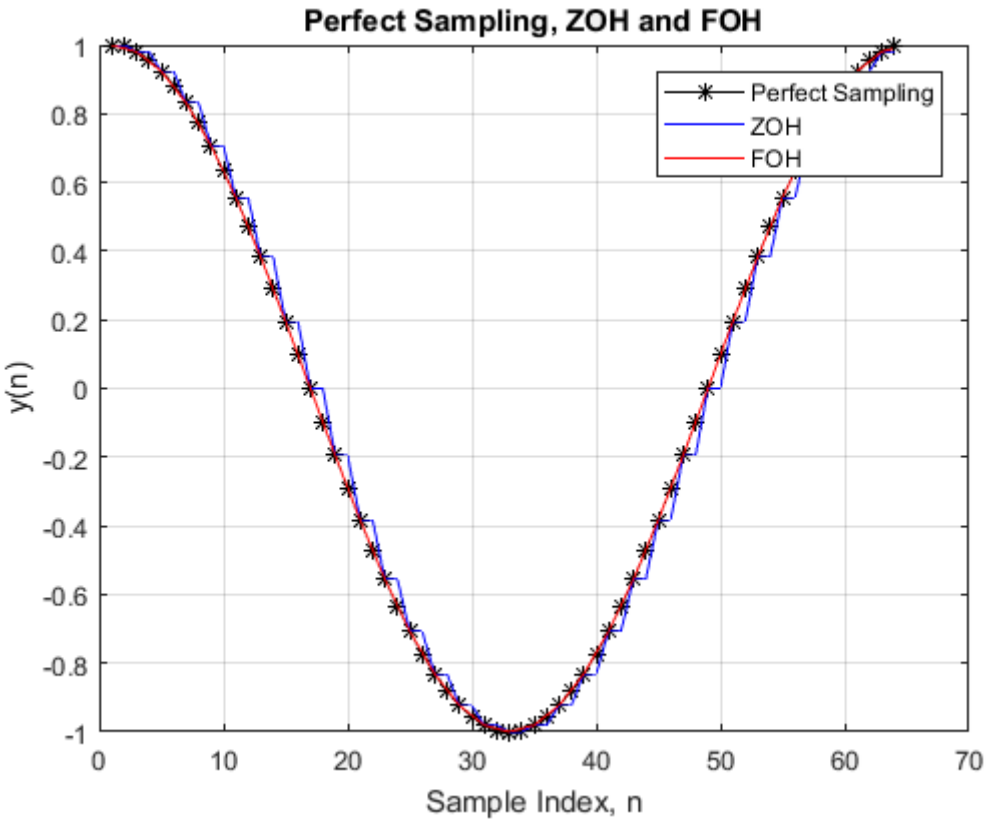
As the prediction based on current sample and previous sample can have 'jumps' at the next sampling instant, delay of one sample is provided. The impulse response of such a system is a triangular function.

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.

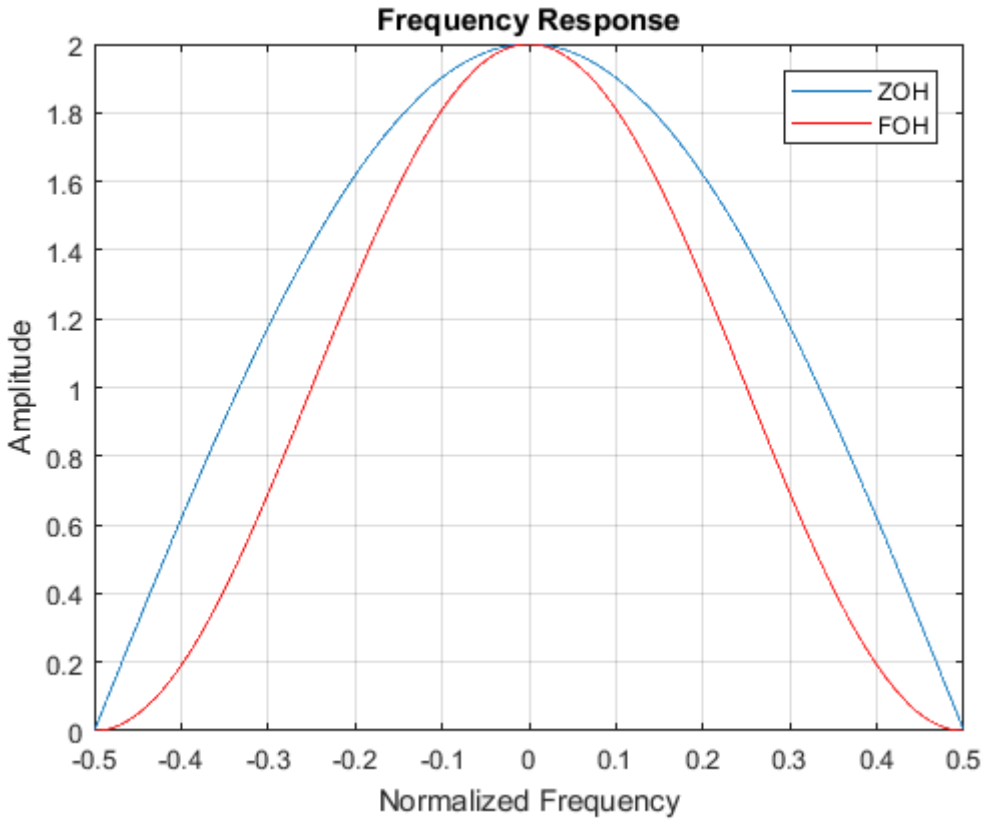


The following MATLAB code outlines frequency response of the comparison between zero-order hold and first-order hold sampling methods.

```
% 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.



Section B. Downsampling and Upsampling

The following exercises show the constructions of signal downsampling (decimation) and upsampling (interpolation) methods in MATLAB.

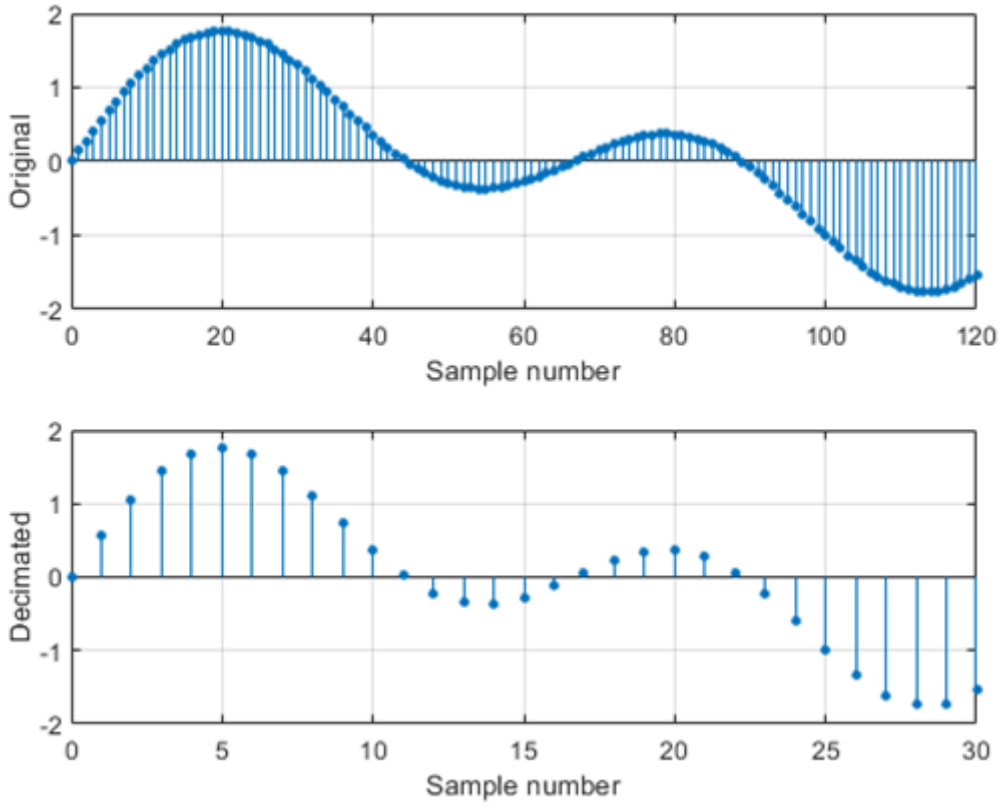
Exercise 4 - Decimation of Signal

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.



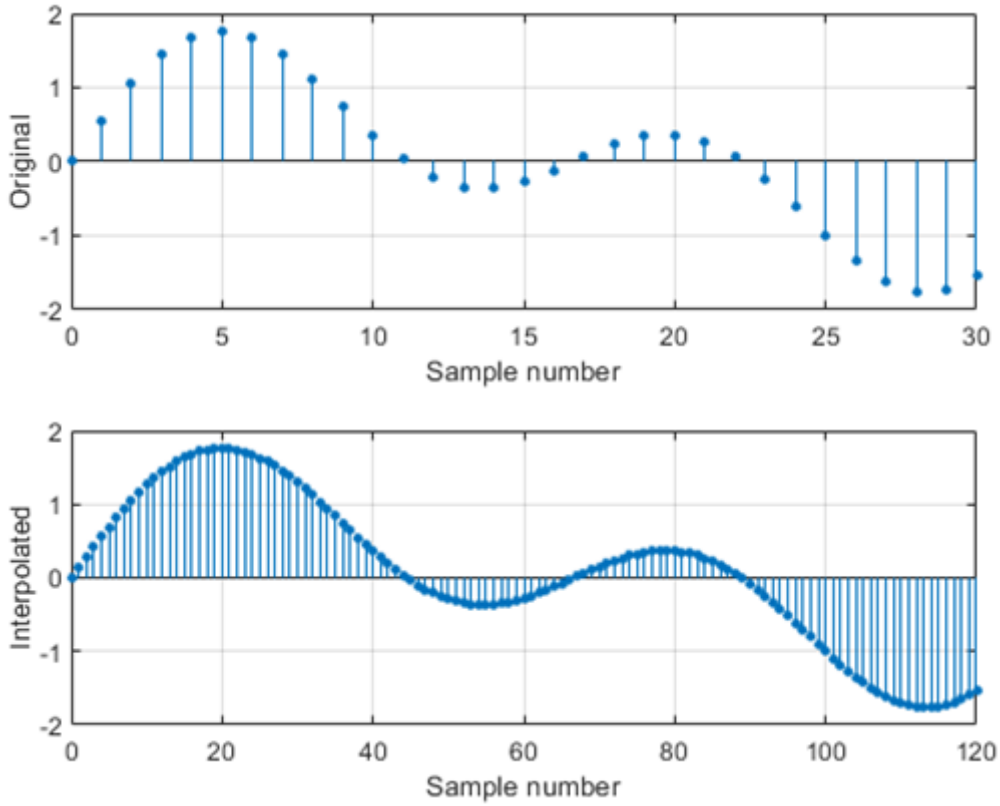
Exercise 5 - Interpolation of Signal

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.



Section C. Resampling a Sinusoid

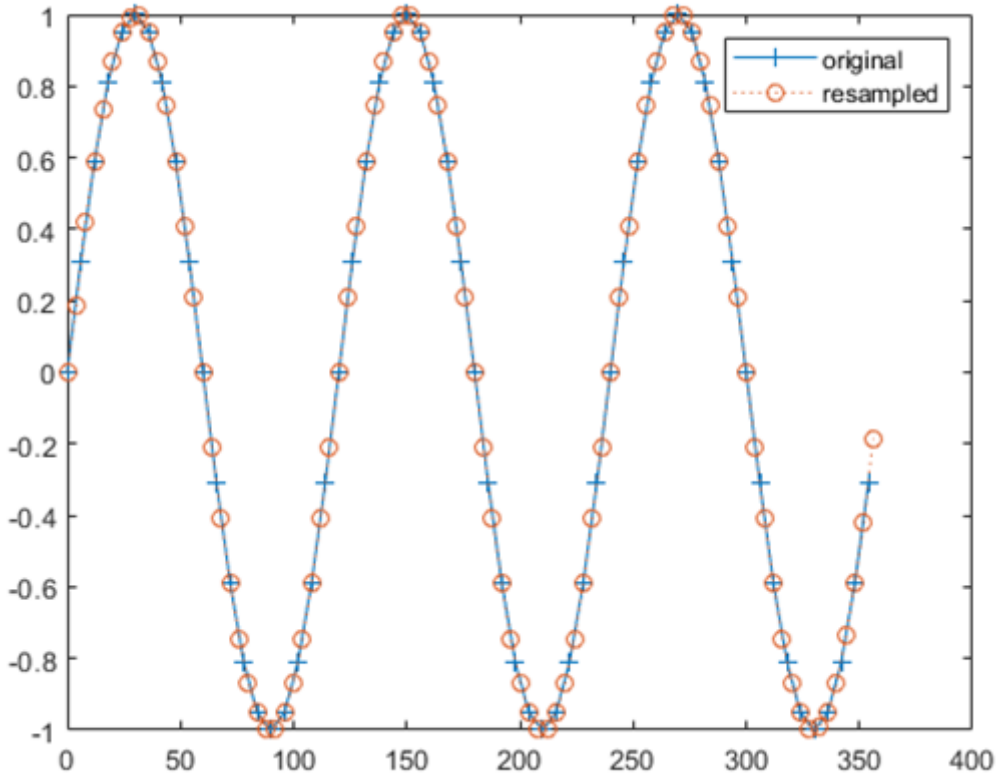
The following exercises show the applications of resampling of signal methods in MATLAB.

Exercise 6 – Resampling Signal

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

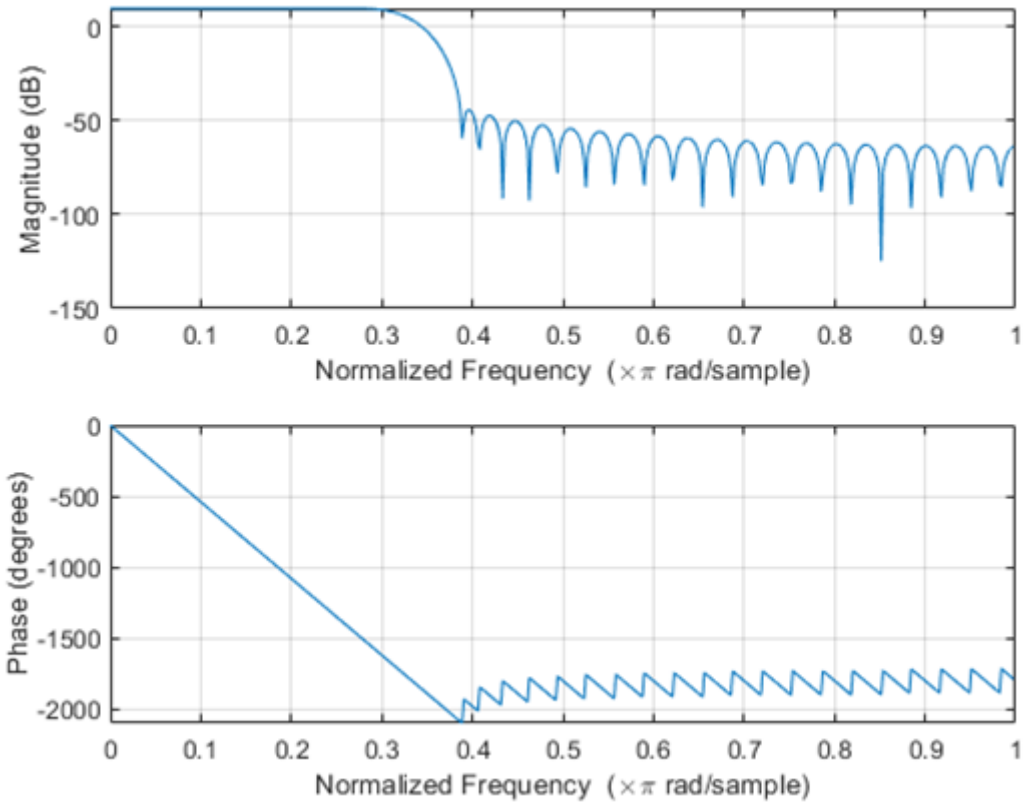
```
tx = 0:6:360-3;
x = sin(2*pi*tx/120);

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.

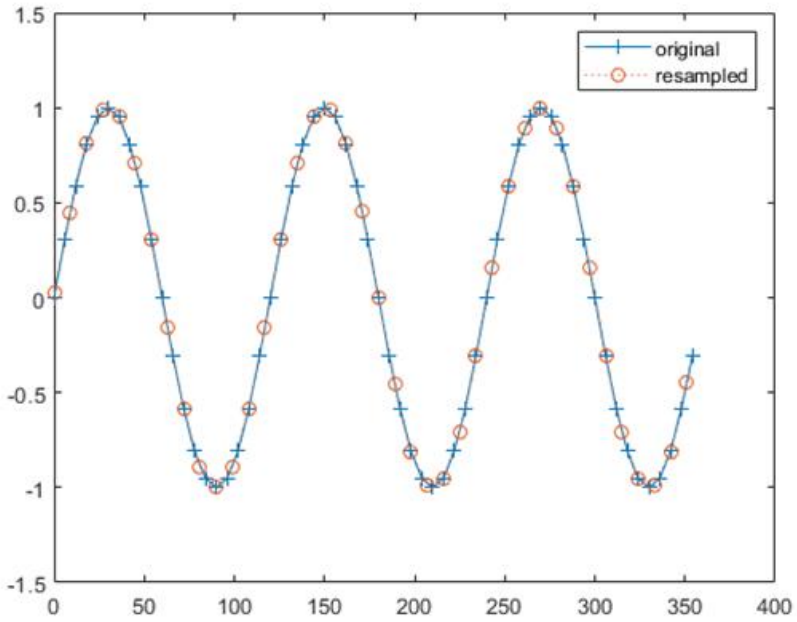
```
freqz(by)
```



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

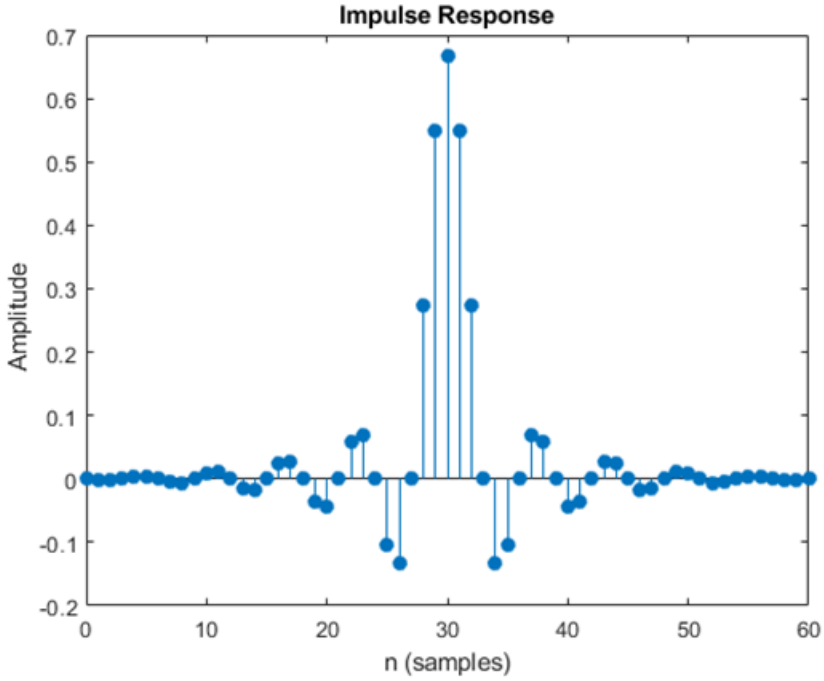
```
tz = 0:9:360-9;
[z,bz] = resample(x,2,3);

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)
```



Appendix – MATLAB Functions

1. `Fliplr()` Function

Flip array left to right

Syntax

```
B = fliplr(A)
```

Description

B = fliplr(A) returns A with its columns flipped in the left-right direction (that is, about a vertical axis).

If A is a row vector, then fliplr(A) returns a vector of the same length with the order of its elements reversed. If A is a column vector, then fliplr(A) simply returns A. For multidimensional arrays, fliplr operates on the planes formed by the first and second dimensions.

2. `fftshift()` function

Shift zero-frequency component to center of spectrum

Syntax

```
Y = fftshift(X)
```

```
Y = fftshift(X,dim)
```

Description

Y = fftshift(X) rearranges a Fourier transform X by shifting the zero-frequency component to the center of the array.

If X is a vector, then fftshift swaps the left and right halves of X.

If X is a matrix, then fftshift swaps the first quadrant of X with the third, and the second quadrant with the fourth.

If X is a multidimensional array, then fftshift swaps half-spaces of X along each dimension.

Y = fftshift(X,dim) operates along the dimension dim of X. For example, if X is a matrix whose rows represent multiple 1-D transforms, then fftshift(X,2) swaps the halves of each row of X.

3. `fft()` function

Fast Fourier transform

Syntax

```
Y = fft(X)

Y = fft(X,n)

Y = fft(X,n,dim)
```

Description

Y = fft(X) computes the discrete Fourier transform (DFT) of X using a fast Fourier transform (FFT) algorithm.

If X is a vector, then fft(X) returns the Fourier transform of the vector.

If X is a matrix, then fft(X) treats the columns of X as vectors and returns the Fourier transform of each column.

If X is a multidimensional array, then fft(X) treats the values along the first array dimension whose size does not equal 1 as vectors and returns the Fourier transform of each vector.

Y = fft(X,n) returns the n-point DFT. If no value is specified, Y is the same size as X.

If X is a vector and the length of X is less than n, then X is padded with trailing zeros to length n.

If X is a vector and the length of X is greater than n, then X is truncated to length n.

If X is a matrix, then each column is treated as in the vector case.

If X is a multidimensional array, then the first array dimension whose size does not equal 1 is treated as in the vector case.

Y = fft(X,n,dim) returns the Fourier transform along the dimension dim. For example, if X is a matrix, then fft(X,n,2) returns the n-point Fourier transform of each row.

4. Resample Function

The resample function changes the sample rate for a sequence to any rate that is proportional to the original by a ratio of two integers. The basic syntax for resample is:

```
y = resample(x,p,q)
```

Where the function resamples the sequence x at p/q times the original sample rate. The length of the result y is p/q times the length of x.

One resampling application is the conversion of digitized audio signals from one sample rate to another, such as from 48 kHz (the digital audio tape standard) to 44.1 kHz (the compact disc standard).

Resample applies a low-pass filter to the input sequence to prevent aliasing during resampling.

5. Decimate and Interpolation Functions

The `decimate` and `interp` functions are equivalent to resample with p = 1 and q = 1, respectively. These functions provide different antialiasing filtering options, and they incur a slight signal delay due to filtering.

5.1. Decimate Function

Decimation — decrease sample rate by integer factor

Syntax

```
y = decimate(x,r)
y = decimate(x,r,n)
y = decimate(x,r,'fir')
y = decimate(x,r,n,'fir')
```

Description

y = decimate(x,r) reduces the sample rate of x, the input signal, by a factor of r. The decimated vector, y, is shortened by a factor of r so that length(y) = ceil(length(x)/r). By default, decimate uses a lowpass Chebyshev Type I infinite impulse response (IIR) filter of order 8.

y = decimate(x,r,n) uses a Chebyshev filter of order n.

y = decimate(x,r,'fir') uses a finite impulse response (FIR) filter designed using the window method with a Hamming window. The filter has an order of 30.

y = decimate(x,r,n,'fir') uses an FIR filter of order n.

5.2. Interpolation Function

Interpolation — increase sample rate by integer factor

Syntax

```
y = interp(x,r)
y = interp(x,r,n,cutoff)
[y,b] = interp(x,r,n,cutoff)
```

Description

y = interp(x,r) increases the sample rate of x, the input signal, by a factor of r.

y = interp(x,r,n,cutoff) specifies two additional values:

n is half the number of original sample values used to interpolate the expanded signal.

Cut-off is the normalized cut-off frequency of the input signal, specified as a fraction of the Nyquist frequency.

[y,b] = interp(x,r,n,cutoff) also returns a vector, b, with the filter coefficients used for the interpolation.

6. `freqz()` function

Frequency response of digital filter

Syntax

```
[h,w] = freqz(b,a,n)

[h,w] = freqz(sos,n)

[h,w] = freqz(d,n)

[h,w] = freqz(___,n,'whole')

[h,f] = freqz(___,n,fs)

[h,f] = freqz(___,n,'whole',fs)

h = freqz(___,w)

h = freqz(___,f,fs)

freqz(___)
```

Description

[h,w] = freqz(b,a,n) returns the n-point frequency response vector h and the corresponding angular frequency vector w for the digital filter with transfer function coefficients stored in b and a.

[h,w] = freqz(sos,n) returns the n-point complex frequency response corresponding to the second-order sections matrix sos.

[h,w] = freqz(d,n) returns the n-point complex frequency response for the digital filter d.

[h,w] = freqz(___,n,'whole') returns the frequency response at n sample points around the entire unit circle.

[h,f] = freqz(___,n,fs) returns the frequency response vector h and the corresponding physical frequency vector f for a digital filter designed to filter signals sampled at a rate fs.

[h,f] = freqz(___,n,'whole',fs) returns the frequency vector at n points ranging between 0 and fs.

h = freqz(___,w) returns the frequency response vector h evaluated at the normalized frequencies supplied in w.

h = freqz(___,f,fs) returns the frequency response vector h evaluated at the physical frequencies supplied in f.

freqz(___) with no output arguments plots the frequency response of the filter.

7. `impz()` function

Impulse response of digital filter

Syntax

```
[h,t] = impz(b,a)

[h,t] = impz(sos)

[h,t] = impz(d)

[h,t] = impz(___,n)

[h,t] = impz(___,n,fs)

impz(___)
```

Description

[h,t] = impz(b,a) returns the impulse response of the digital filter with numerator coefficients b and denominator coefficients a. The function chooses the number of samples and returns the response coefficients in h and the sample times in t.

[h,t] = impz(sos) returns the impulse response of the filter specified by the second-order sections matrix sos.

[h,t] = impz(d) returns the impulse response of the digital filter d. Use designfilt to generate d based on frequency-response specifications.

[h,t] = impz(___,n) specifies what impulse-response samples to compute. You can specify the filter using any of the previous syntaxes.

[h,t] = impz(___,n,fs) returns a vector t with consecutive samples spaced 1/fs units apart.

impz(___) with no output arguments plots the impulse response of the filter.