Question1:

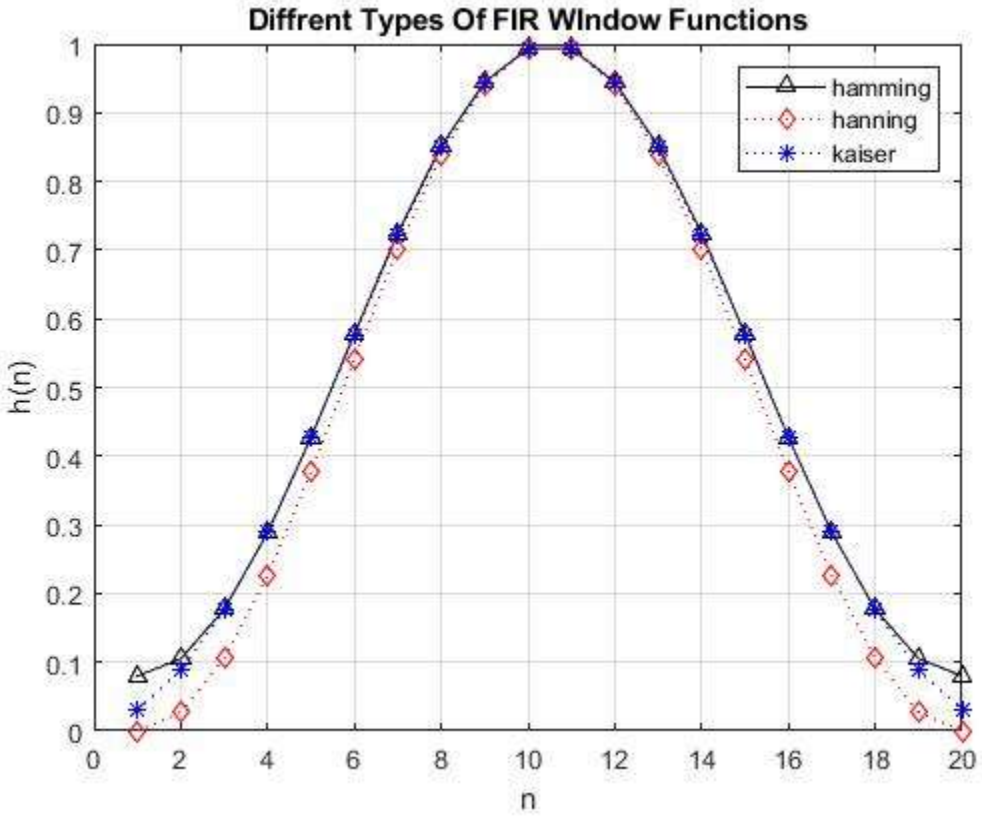
Plot Hamming, Hann and Kaiser window functions used in design of a FIR filter. Comment on the results.

Solution:

```
beta = 5.2; % Required for Kaiser Window
N = 20;     % Length of the filter
n = 1:1:20; % Range of the filter
y = hamming(N); % For Hamming window
y1= hann(N);    % For Hanning window
y2 = kaiser(N,beta); % For Kaiser window
plot(n, y, 'k^-',n,y1,'rd:',n,y2,'b*:'); % Plot 3
windows
                                         % k = For
Black color
                                         % r = For Red
color
                                         % b = For blue
color
                                         % ^ = For
Pyramid Shape
                                         % - = Graph
Represents as '--'
                                         % d = For
Diamond
                                         % : = Graph
Represents as '..'
                                         % * = Graph

Represents as '**'
xlabel('n');
ylabel('h(n)');
title('Different Types Of FIR Window Functions')
legend('hamming','hanning','kaiser');    % Marking 3
types of figures
grid on ;
```

Output:



Observation:

The window methods that we can see from above graph, Hanning, Hamming and Kaiser are used in Finite Impulse Response(FIR) filtering technique like Low Pass Filter (LP), High Pass Filter(HP), Band Pass Filter (BP), Band Stop Filter(BS) or Notch Filter for designing FIR Filter. Where Hanning window = 0.5 + 0.5cos(2*pi*n/N) , Hamming window = 0.56 + 0.42cos(2*pi*n/N) etc and N is the sample size of the signal. These windows are multiplied by the ideal imulse response (hd) to generate the practical impuse respose that provides a visual understanding how related the windows with impulse responses to calculate the filter cofficients that will be used further implementation. Mentioned the all three windows tipically same with each other for few numbers of cofficients but for a large number of cofficients kaiser window is the best for saving memory.

Question2:

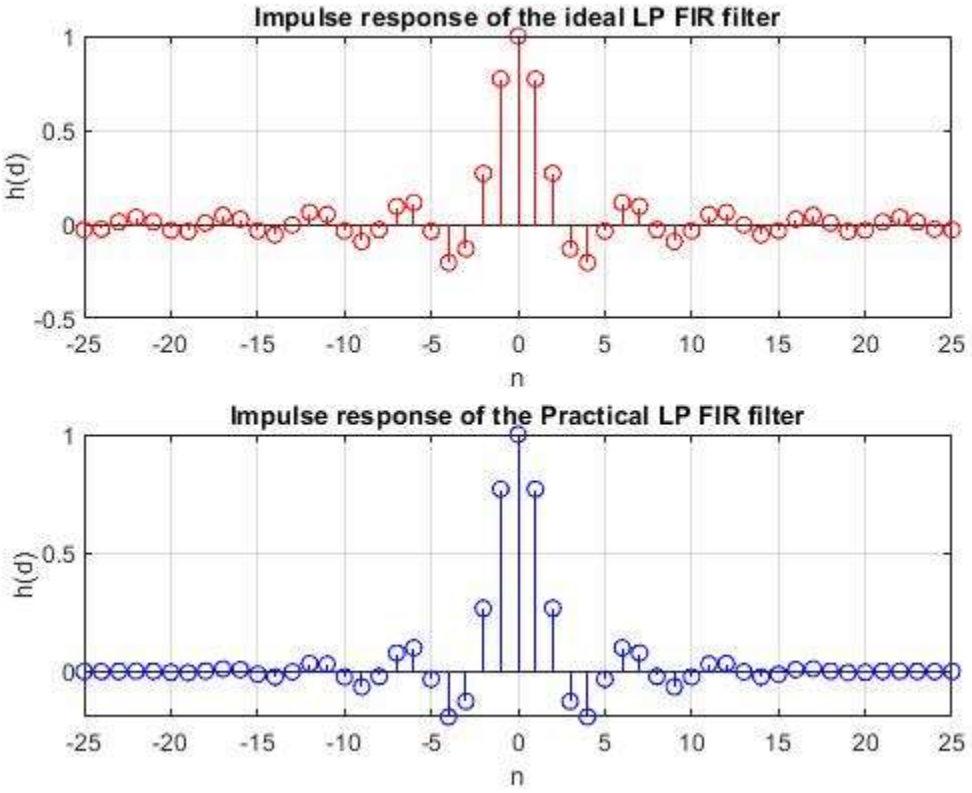
Let us observe the impact of window function in impulse response of a LP FIR filter. Compare between ideal and practical FIR LP filter.

Solution:

```
% Low Pass Filter (LP)
% First Part: Calculate Ideal Impulse Response hd
N = 25; % Length Of The Filter
n = -N:1:N; % Range Of The Filter
fc = 0.1930; % Cutoff Frequency Calculated by fp +
(delf/Fs)/2
hd = sinc(2*fc*n); % hd = Ideal Impulse Response
subplot(2,1,1);
stem(n,hd,'r');         % stem for descrete ideal
impulse sequence
xlabel('n');
ylabel('h(d)');
title('Impulse response of the ideal LP FIR filter');
grid on ;

wn = hann(2*N + 1); % For Hanning Window
subplot(2,1,2);
% Second Part: Calculate Practical Impulse Response
h(n)
h = hd'.*wn;          % Vector Multiplication
stem(n,h,'b');
xlabel('n');
ylabel('h(d)');
title('Impulse response of the Practical LP FIR
filter');
grid on ;
```

Output:



Observation:

From the above code and graph we can see that an ideal impulse response (hd) of the low pass filter (LP) and a practical impulse response (h(n)) of the low pass filter (LP) of a signal typically similar with each other. In where we can also see that a Hanning Window method is used for generating the practical impulse responses (h(n)) by multiplying the corresponding ideal impulse responses (hd). In deeply brief, samples from n =-25 to -15 and n = 15 to 25 represents an ignorable changed in practical impulse response (h(n)) compared to the ideal impulse response (hd). But from n = -10 to n =10 in practical impulse response represents as much as similar with the ideal impulse response (hd).That means by applying hanning window in LP filter to generate practical impulse response (h(n)) is much accurate as the ideal impulse response (hd).

Home Work:

Compare the performance of ideal and practical LP filter using hamming window function for the following specifications.

Fs = 8KHz

Passband edge frequency 1.5 KHz

Transition width 0.5KHs

Stop band attenuation >50dB

Solution:

Given Fs = 8KHz

fp = 1.5KHz

delf = 0.5KHZ

So normalized delf = delf/Fs = 0.5/8 = 0.0625

So, the normalized fp = fp/Fs = 1.5/8 = 0.1875

And the cutoff frequency fc = fp + delf/2 = 0.1875 + 0.0625/2 = 0.03125

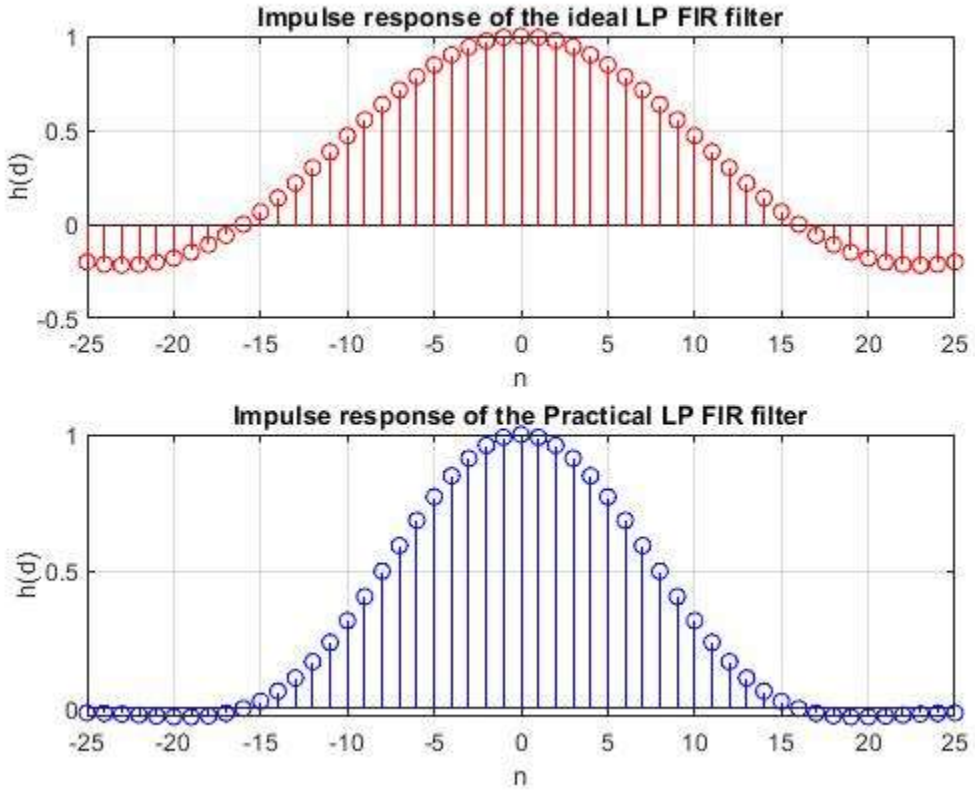
Code:

```
% Low Pass Filter (LP)
% First Part: Calculate Ideal Impulse Response hd
N = 25; % Length Of The Filter
n = -N:1:N; % Range Of The Filter
fc = 0.03125; % Cutoff Frequency Calculated by fp +
(delf/Fs)/2
hd = sinc(2*fc*n); % hd = Ideal Impulse Response
subplot(2,1,1);
stem(n,hd,'r');          % stem for discrete ideal
impulse sequence
xlabel('n');
```

```
ylabel('h(d)');
title('Impulse response of the ideal LP FIR filter');
grid on ;

wn = hamming(2*N+1); % For Hamming Window
subplot(2,1,2);
% Second Part: Calculate Practical Impulse Response
h(n)
h = hd'.*wn;          % Vector Multiplication
stem(n,h,'b');
xlabel('n');
ylabel('h(d)');
title('Impulse response of the Practical LP FIR
filter');
grid on ;
```

Output:



Observation:

From the above code and graph we can see that an ideal impulse response (hd) of the low pass filter (LP) and a practical impulse response (h(n)) of the low pass filter (LP) of a signal typically similar with each other. In where we can also see that a Hamming Window method is used for generating the practical impulse responses (h(n)) by multiplying the corresponding ideal impulse responses (hd). In deeply brief, samples from n =-25 to -14 and n = 14 to 25 represents a slightly changed in practical impulse response (h(n)) compared to the ideal impulse response (hd). But from n = -15 to n =15 in practical impulse response represents as much as same with the ideal impulse response (hd).That means by applying hamming window in LP filter to generate practical impulse response (h(n)) is much accurate as the ideal impulse response (hd).

Question3:

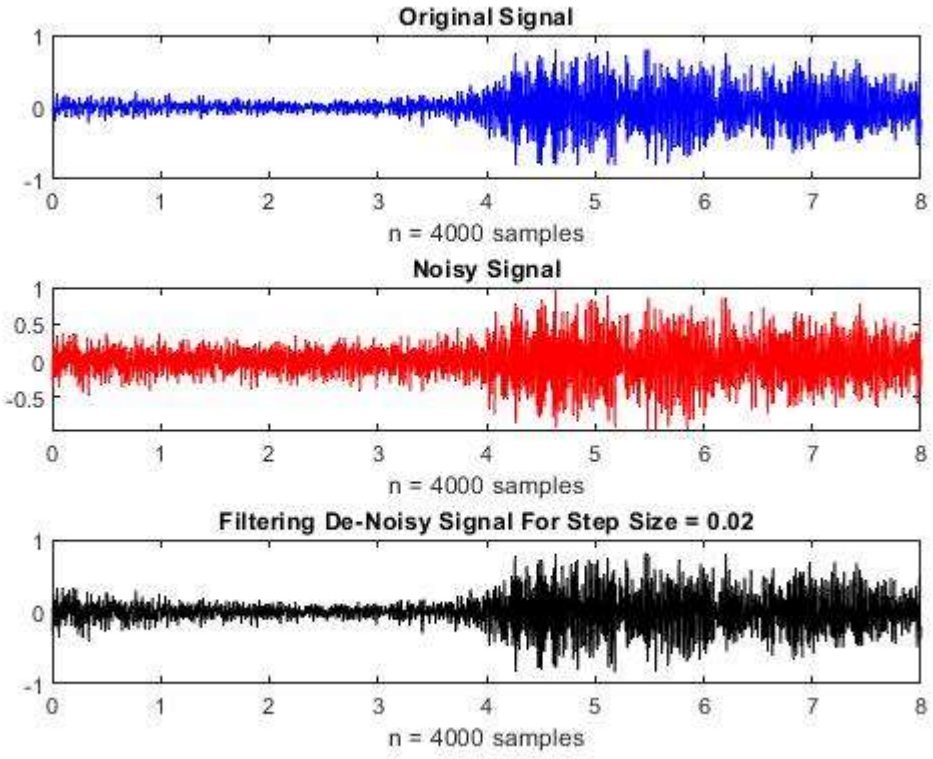
Apply LMS adaptive filter for a speech signal. Verify the sound quality using: sound(s), sound(sn) and sound(e) taking 4000 and 20,000 samples.

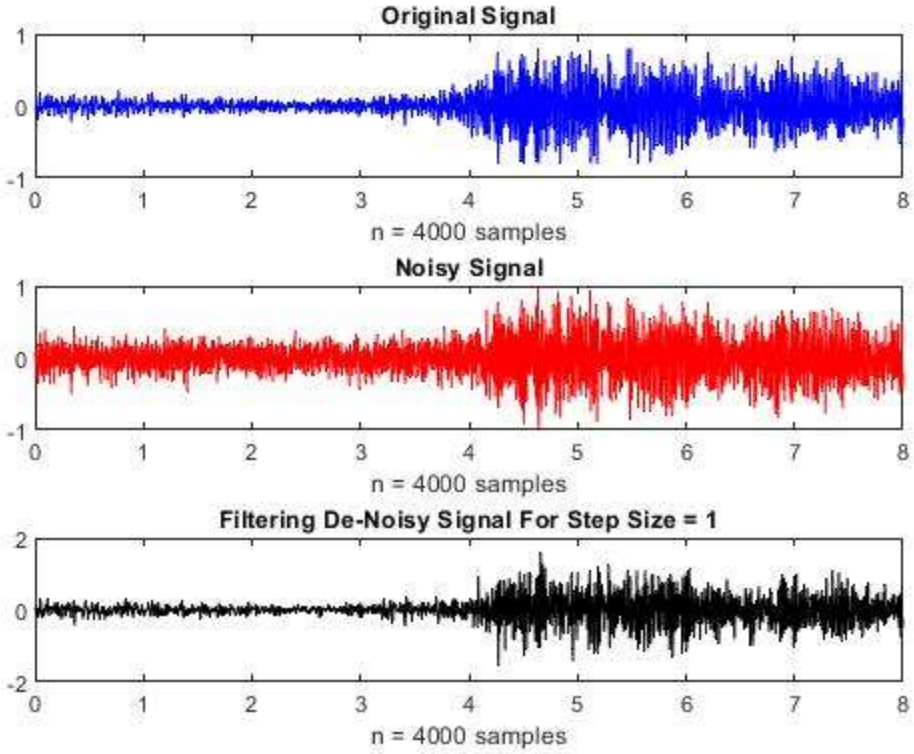
Solution:

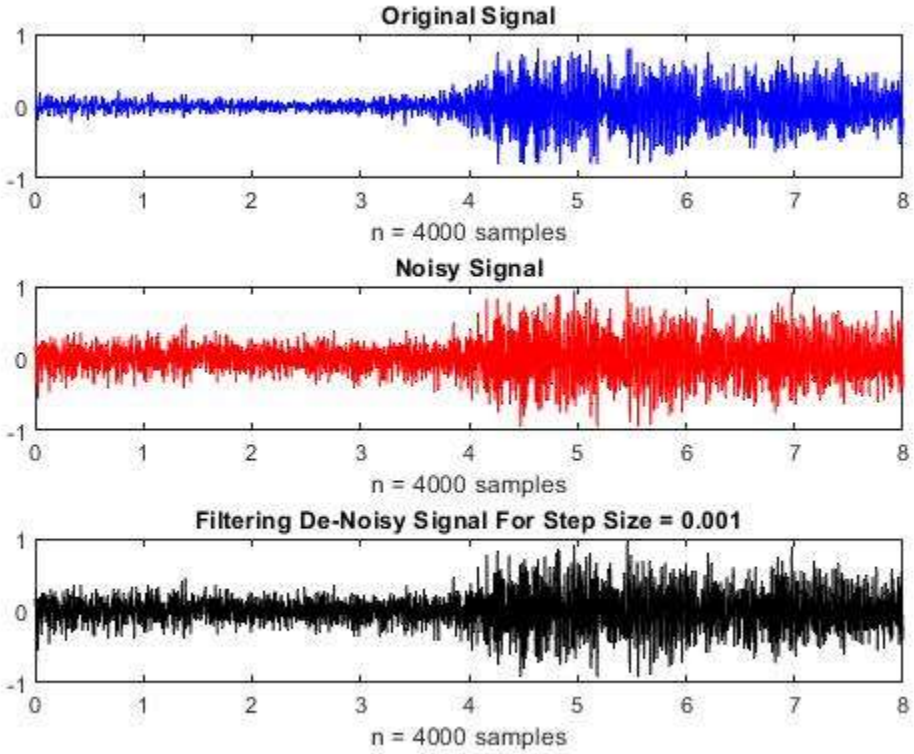
```
load handel % Load an original signal
s = y(1:4000); % %taking 4000 samples of sound wave
s = s';
noise = 0.12*randn(1,length(s)); % Generate a random
noise
sn = s + noise; % Adding noise with original sound
%Now Filtering the signal using LMS algorithm with step
size mu = 0.02
lmsfilt = dsp.LMSFilter('Length', 16 ,
'Method','Normalized LMS', 'StepSize' ,0.02);
[y,e,w] = lmsfilt(noise', sn'); % y = original sound ,
e = Estimated error , w = weighted factor
t = 1:1:length(s);
t = t / 500;
subplot(3,1,1);
```

```
plot(t,s,'b') % Plot the original continuous sound
signal
title('Original Signal');
subplot(3,1,2);
plot(t,sn,'r') % Plot the noisy continuous sound signal
title('Noisy Signal');
subplot(3,1,3);
plot(t,e,'k'); % Plot the filtering de-noisy continuous
sound signal
title('Filtering De-Noisy Signal');
```

Output:




Original Signal
n = 4000 samples
Noisy Signal
n = 4000 samples
Filtering De-Noisy Signal For Step Size = 1
n = 4000 samples


Original Signal
n = 4000 samples
Noisy Signal
n = 4000 samples
Filtering De-Noisy Signal For Step Size = 0.001
n = 4000 samples

Observation:

In the above code and graph represents three different types of signals taking 4000 samples

a. Original Signal
b. Noisy Signal and
c. Filtered de-noisy signal for
    I. Step size, mu = 0.02
    II. Step size, mu = 1 and
    III. Step size, mu = 0.001

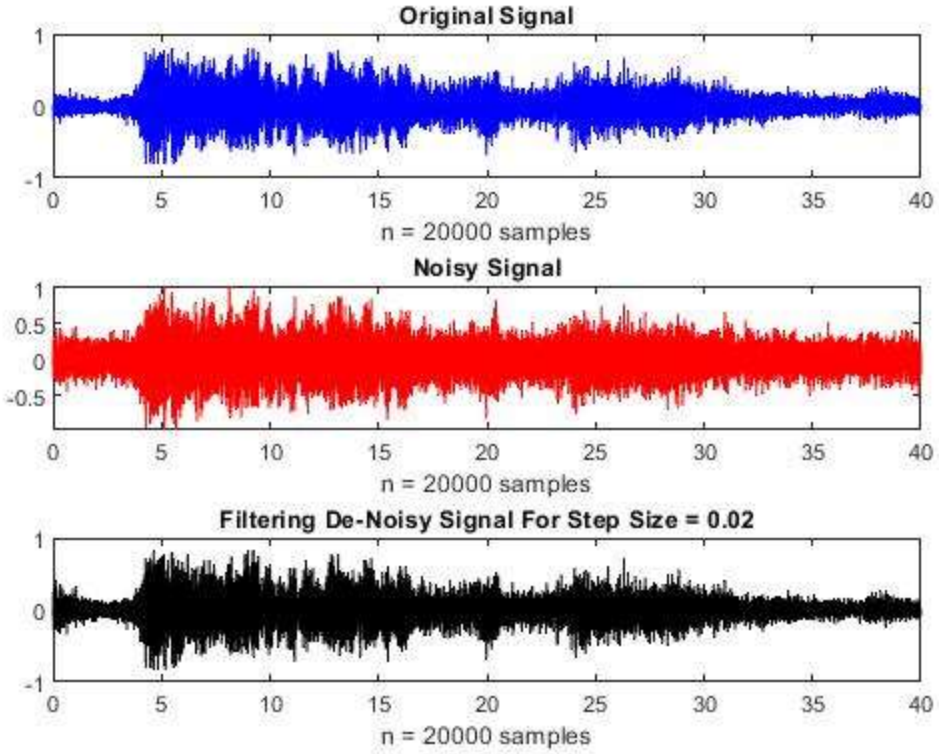
When using step size mu = 0.02 then the adaptive filter performs a good filtering because it removes noise smoothly as we want. As a result, we are getting an expected de-noisy signal that is the same as our original signal respectively. But in deeply we can observe that from n = 0 to n = 1 in figure-1, adaptive filter does not provide a good result compared to the original because in the meantime adaptive filter taking time to adjust its auto adjusting coefficients with its weighting factors.

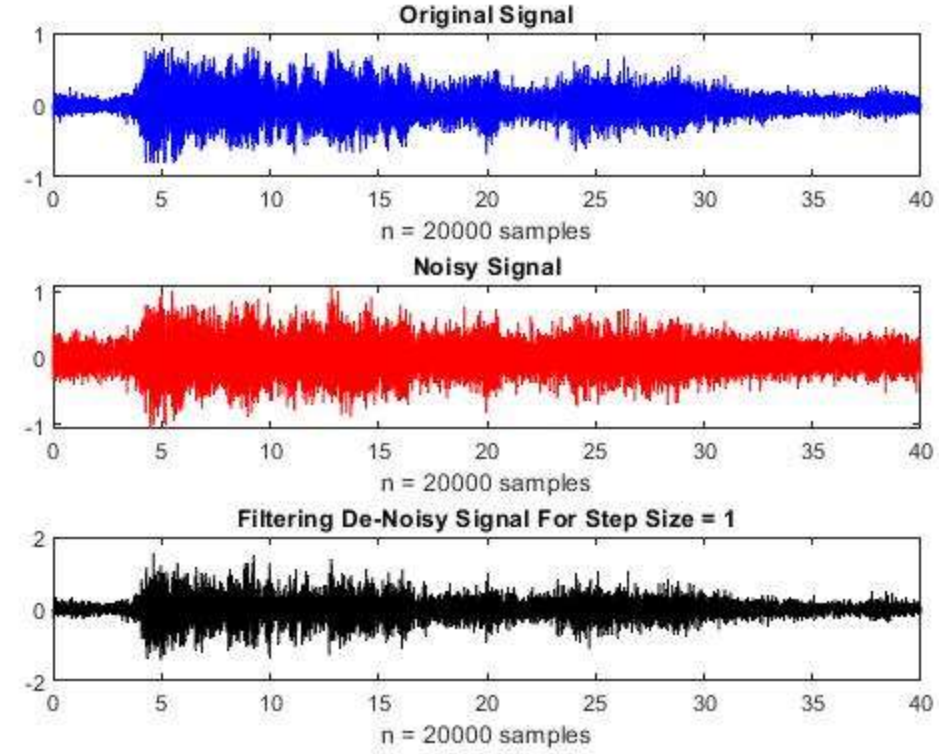
When using step size mu = 1 then the adaptive filter does not perform a good filtering because it cannot remove noise smoothly due to faster step. As a result, we are getting an unexpected de-noisy signal that is not much similar as our original signal. But in deeply we can observe that from n = 4 to n = 8 in figure-2, adaptive filter does not provide a good result compared to the original signal that we want.

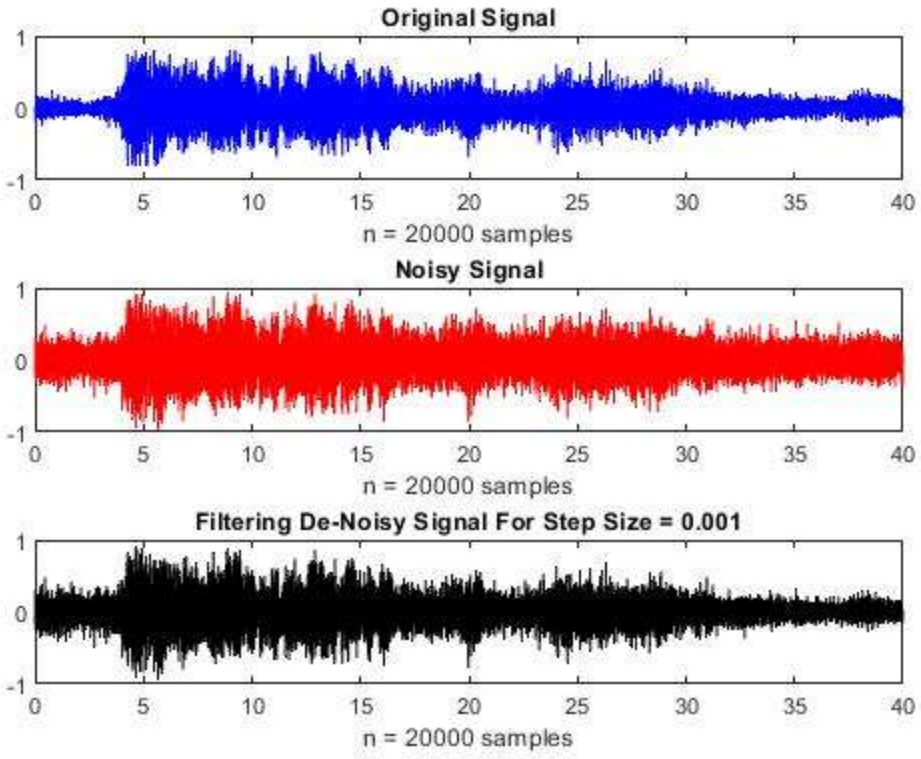
When using step size mu = 0.001 then the adaptive filter does not perform a good filtering but better result compared to the mu =1 because it cannot remove noise smoothly due to slower step. As a result, we are getting an unexpected de-noisy signal that is not much similar as our original signal. But in deeply we can observe that from n = 4 to n = 8 in figure-2, adaptive filter provides a good result compared to the original signal that we want.

```
% Next Part: Taking 20000 samples of sound wave
load handel % Load an original signal
s = y(1:20000); % Taking 4000 samples of sound wave
s = s';
noise = 0.12*randn(1,length(s)); % Generate a random
noise
sn = s + noise; % Adding noise with original sound
%Now Filtering the signal using LMS algorithm with step
size mu = 0.02
lmsfilt = dsp.LMSFilter('Length', 16 ,
'Method','Normalized LMS', 'StepSize' ,0.02);
[y,e,w] = lmsfilt(noise', sn'); % y = original sound ,
e = Estimated error , w = weighted factor
t = 1:1:length(s);
t = t / 500;
subplot(3,1,1);
plot(t,s,'b') % Plot the original continuous sound
signal
title('Original Signal');
subplot(3,1,2);
plot(t,sn,'r') % Plot the noisy continuous sound signal
title('Noisy Signal');
subplot(3,1,3);
plot(t,e,'k'); % Plot the filtering de-noisy continuous
sound signal
title('Filtering De-Noisy Signal');
```

Output:


Original Signal
n = 20000 samples
Noisy Signal
n = 20000 samples
Filtering De-Noisy Signal For Step Size = 0.02
n = 20000 samples


Original Signal
n = 20000 samples
Noisy Signal
n = 20000 samples
Filtering De-Noisy Signal For Step Size = 1
n = 20000 samples



Observation:

In the above code and graph represents three different types of signals taking 20000 samples

d. Original Signal
e. Noisy Signal and
f. Filtered de-noisy signal for
    IV. Step size, mu = 0.02
    V. Step size, mu = 1 and
    VI. Step size, mu = 0.001

When using step size mu = 0.02 then the adaptive filter performs a good filtering because it removes noise smoothly as we want. As a result, we are getting an expected de-noisy signal that is the same as our original signal respectively. But in deeply we can observe that from n = 0 to n = 1 in figure-1, adaptive filter does not

provide a good result compared to the original because in the meantime adaptive filter taking time to adjust its auto adjusting coefficients with its weighting factors.

When using step size mu = 1 then the adaptive filter does not perform a good filtering because it cannot remove noise smoothly due to faster step. As a result, we are getting an unexpected de-noisy signal that is not much similar as our original signal. But in deeply we can observe that from n = 4 to n = 35 in figure-2, adaptive filter does not provide a good result compared to the original signal that we want.

When using step size mu = 0.001 then the adaptive filter does not perform a good filtering but better result compared to the mu =1 because it cannot remove noise smoothly due to slower step. As a result, we are getting an unexpected de-noisy signal that is not much similar as our original signal. But in deeply we can observe that from n = 4 to n = 40 in figure-2, adaptive filter provides a good result compared to the original signal that we want.