FIR Filter Designer Tool Project - Phase #2

Team Members

Riham Al-Sayed Abdo	201601808
Fatma Moanes	201700346
Mahmoud Ashraf	201700989

Contact Info

s-rihamabdo@zewailcity.edu.eg

s-fatma_moanes@zewailcity.edu.eg

Sunday, December 15, 2019

Zewail City of Science and Technology
University of Science and Technology
Communications & Information Engineering Program
CIE 442 - Fall 2019

DSP Team

FIR Filter Designer | Matlab Tool

1. Introduction

In the digital control system, interference immensely influences the performance of the system. Thus, the input signal has to be processed in order to get a usable signal. Finite impulse response (FIR) filters play an important role in the processing of the digital signal. Applying an FIR filter can simplify the complicated computation in simulation and improve the performance. By utilizing methods of windowed-sinc or the least squares, the design of FIR filter can easily be done by Matlab.

1.1 Introduction to Digital Filters

A digital filter is a discrete system which performs a series of mathematical operations on the input signal, and therefore reducing or enhancing certain aspects of the input signal. The transfer function for a linear time-invariant digital filter is usually expressed as,

$$H(z) = \frac{\sum_{j=0}^{M} b_j z^{-j}}{1 + \sum_{i=o}^{N} a_i z^{-i}}$$

where a_i and b_j are the filter's coefficients in the z-domain.

Digital filters may be more expensive than an equivalent analogue filter due to their increased complexity, but they make practical many designs that are impractical or impossible as analogue filters. When a_i is always zero, we get ourselves a finite impulse response filter. If there is at least one non-zero coefficient, we get an infinite impulse response filter.

1.2 FIR Filters

A finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration because it settles to zero in finite time. An FIR is a special case of the previous equation and can be obtained through,

$$H(z) = \sum_{k=0}^{M-1} b_k z^{-k}$$

A direct form realization of the previous equation would be a filter of order 2, where M = 3 as shown in fig.1.

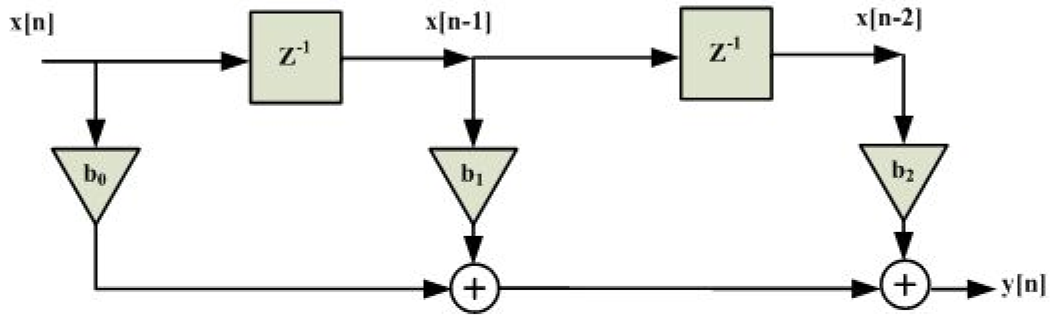


Fig.1. Direct form of an FIR filter of order2.

There are three coefficients and two delay blocks in fig.1, noting that the filter's order is two just representing the number of the delay blocks, not the number of coefficients. An FIR filter has two main advantages over an IIR filter:

- ➔ There is no feedback loop in the structure of an FIR filter. Due to its non-recursive structure, an FIR filter is inherently stable.
- ➔ An FIR filter can maintain a linear phase which is the FIR filters' main advantage. In order to maintain a linear phase, we must provide symmetry in the time domain.

Why a linear phase is a huge advantage?

Linear-phase response corresponds to a constant delay. A system with a nonlinear-phase response will cause distortion to the input. In such a system, different frequency components of the input will experience different time delays as they pass through the system.

2. Methodology: Design of FIR Filter

It is necessary to specify passband, stopband, and transition band when designing a frequency-selective filter. In passband, frequencies are needed to be passed unattenuated. In stopband, frequencies need to be completely attenuated. Transition band contains frequencies that are lying between the passband(s) and stopband(s). It is called the "don't care band". Therefore, the entire frequency range is split into one or more passbands, stopbands, and transition bands.

In practice, the magnitude is not necessarily constant in the passband of a filter. A small amount of ripple is usually allowed. Similarly, the filter response does not become zero in the stopband. A small, nonzero value is also tolerated in the stopband. Fig.2 shows the ripples of a practical low pass filter.

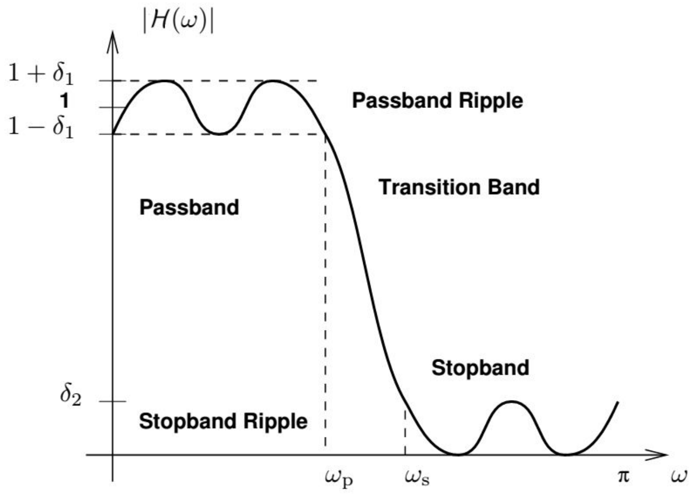


Fig.2. Bands of a practical filter. Image courtesy of the University of Michigan.

2.1 Window-Sinc Method

In this method, a truncated ideal lowpass filter with a certain bandwidth is generated, and then we use a chosen window to get certain stopband attenuation. The length of filter L can be adjusted to meet a specified roll-off rate in the transition band. Any finite-length ideal lowpass impulse response may be considered as a product of the infinite-length lowpass impulse response and a window function W.

$$b = \frac{sinc(\omega_c[n-M])}{\pi[n-M]} * W_L[n-M]$$

The result is a finite-length truncated low pass filter.

Types of windows we use

Rectangular	The simplest type of windows. It is equivalent to replacing all but N values of a data sequence by zeros, making it appear as though the waveform suddenly turns on and off. Described as, $\omega_n = 1$. *Why use it?* Because they have the sharpest transition band and highly selective of particular frequencies
Blackman	It's a general-purpose window since it offers good attenuation and good selectivity. Though it's not as surgical as a rectangle, it can still focus in well enough for audio purposes.
Chebyshev	The sidelobe attenuation can be controlled by the parameter α which is equal to,

$$\alpha = \cosh\left(\frac{1}{N}\cosh^{-1}\left(10^{\frac{A}{-20}}\right)\right)$$

Kaiser	Kaiser window is a simple approximation of the DPSS window using Bessel functions. The parameter β controls the relative sidelobe attenuation. As β increases, the relative sidelobe attenuation decreases and the mainlobe width increases.

In our tool, we use the function ***windowMethod()*** for the realization of the window method in Matlab. Find the Matlab code for the function in appendix.A.

```
% function inputs:
% f_s -> sampling frequency
% f_c -> cutoff frequency
% n -> no of filter taps
% wind -> window type
% par -> beta in case of choosing Kaiser, A in case of Chebyshev

% Output: hd -> filter coefficients.
```

2.2 Least Square Method

Suppose that we have an even symmetric filter with an odd number of taps(type 1); where h(1)= h(-1) , h(2) = h(-2) and so on. Therefore, if the length of the filter is N, we only need to store half of the taps in addition to h(0). Then, suppose that we have a filter of order 4 in the time domain , and a user wants to store for example 30 points in the frequency domain. How can we connect the points together and reach the optimal filter that is close to the ideal one? One solution to this problem is to use the least squares method which aims to minimize the square of the error. If we take DTFT to the TD filter taken into consideration the symmetry of the filter mentioned above, the equation becomes as follows:

$$H(\omega) = h(0) + \sum_{n=1}^{L} 2\,h(n) \times cos(\omega n)$$

The summation is from n =1 to L where L= (N-1)/2 . By solving the system of linear equations

```
1  2cos(w1)    2cos(2w1)   2cos(3w1) .... 2cos(Lw1)     h(0)       H(w1)
1  2cos(w2)    2cos(2w2)   2cos(3w2) .... 2cos(Lw2)     h(1)       H(w2)
.                                                        .           .
.                                                   *    .     =     .
.                                                        .           .
1  2cos(wk)    2cos(2wk)   2cos(3wk) .... 2cos(Lwk)     h(L)       H(wk)
```

And using linear algebra and some vector calculus we reach the equation of the desired filter coefficients:

$$H_{optimal} = (F^{T}F)^{-1}F^{T}H_{desired}$$

Which can be written like this in matlab : H_optimal = F\ H_desired '

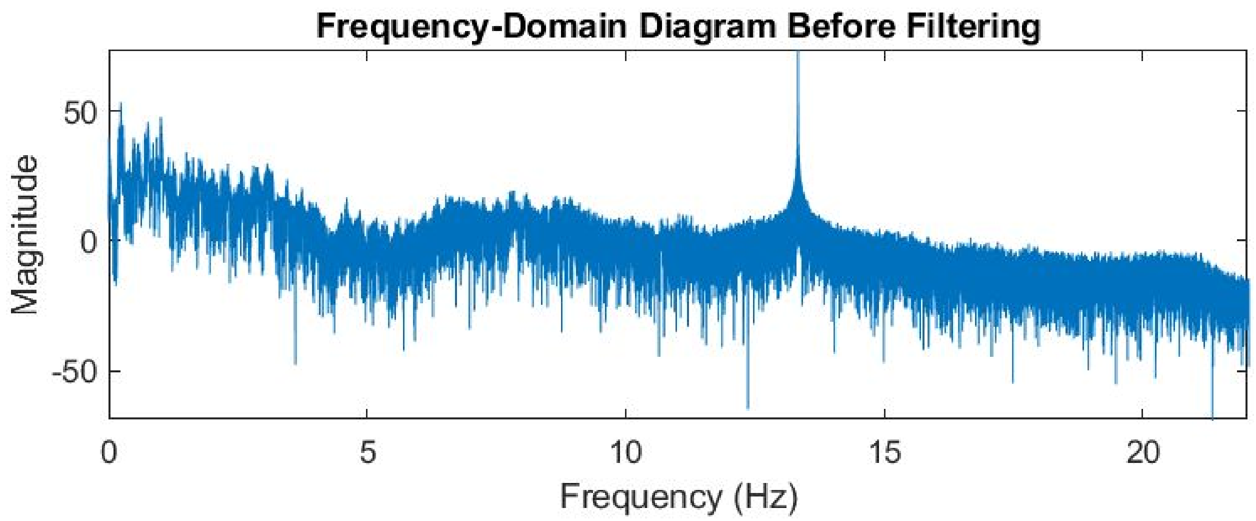
Function demonstration

The leastSquares function that we implemented takes as an input: the length of the filter N, the number of points in desired frequency response, and the cutoff frequency ω and returns the filter coefficients of the filter in the time domain H_{full}. For the weightedleastSquares function, it takes the same input in addition to the weights of the passband and the stopband, and also returns the optimal filter coefficients.
Find the matlab code of both functions in appendix A.

Practical Problem | Noise Removal

Problem Definition

We remove high frequency background noise from an audio file by applying a low pass filter. Spectral Analysis of the audio file is,

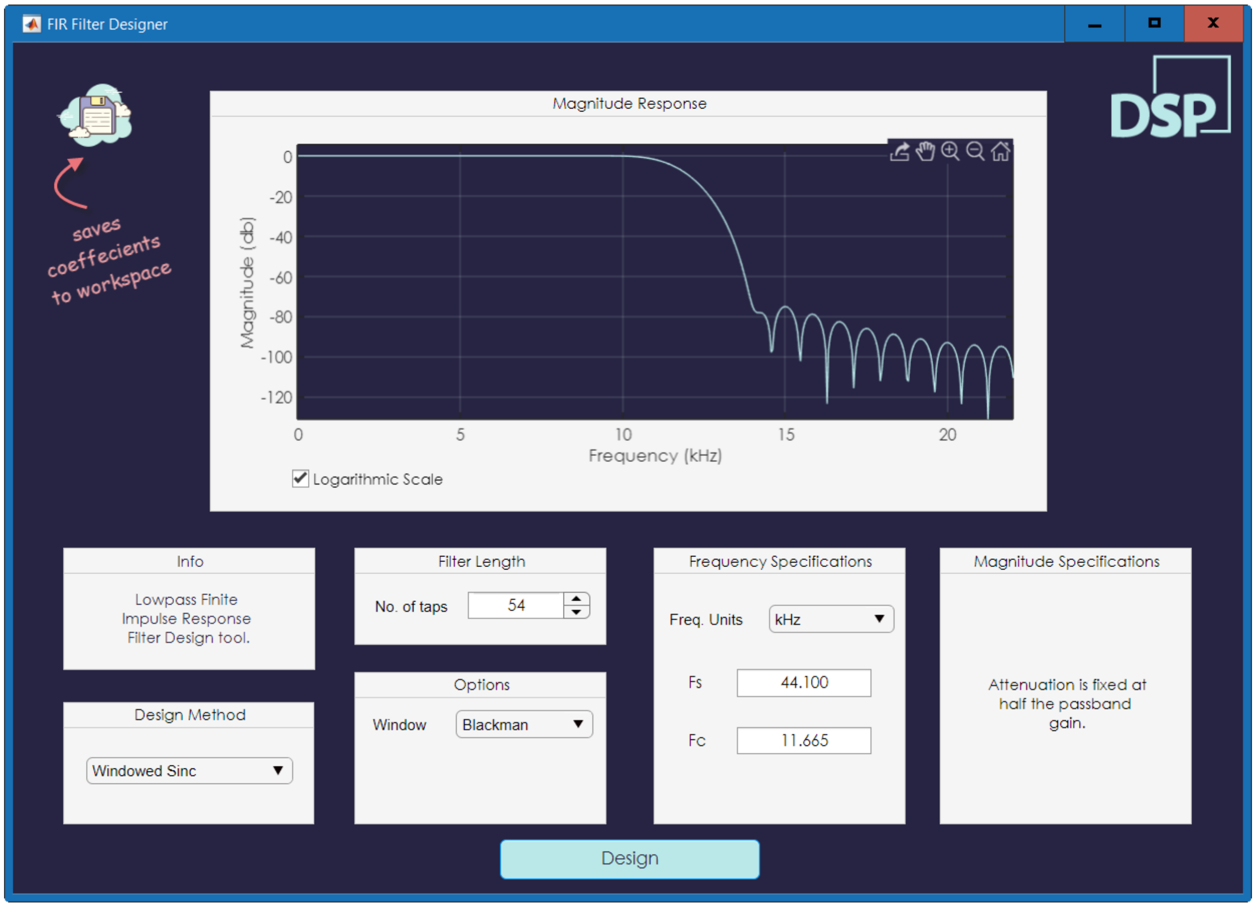


We know that the distortion lies in the high frequency range. A particular high frequency components lies around 13 kHz.

Specifications

- ➔ $f_s \; = \; 44.1 \; kHz$
- ➔ $f_{pass} \; = \; 10 \; kHz \; and \; f_{stop} = 13.33 \; kHz$
- ➔ $Transition \; band \; Width \; = \frac{f_{stop} - f_{pass}}{f_s} \; = \; 0.0755$
- ➔ $filter \; order \; = \; \frac{4}{\omega_t} \; \approx \; 53 \; .. \; No. \; of \; taps \; = \; 54$

We try the window method using blackman window since it meets the specifications above with a suitable stopband attenuation. Trying those specifications on our Designer tool,



✓ Logarithmic Scale

Info
Lowpass Finite Impulse Response Filter Design tool.

Design Method
Windowed Sinc

Filter Length
No. of taps 54

Options
Window Blackman

Frequency Specifications
Freq. Units kHz
Fs 44.100
Fc 11.665

Magnitude Specifications
Attenuation is fixed at half the passband gain.

Figure above shows the GUI of our tool filled to design low pass filter for the audio file. Next step is sending the coefficients to matlab workspace. Then, it is applied on the audio file in *main.m* file.

Designed Filter

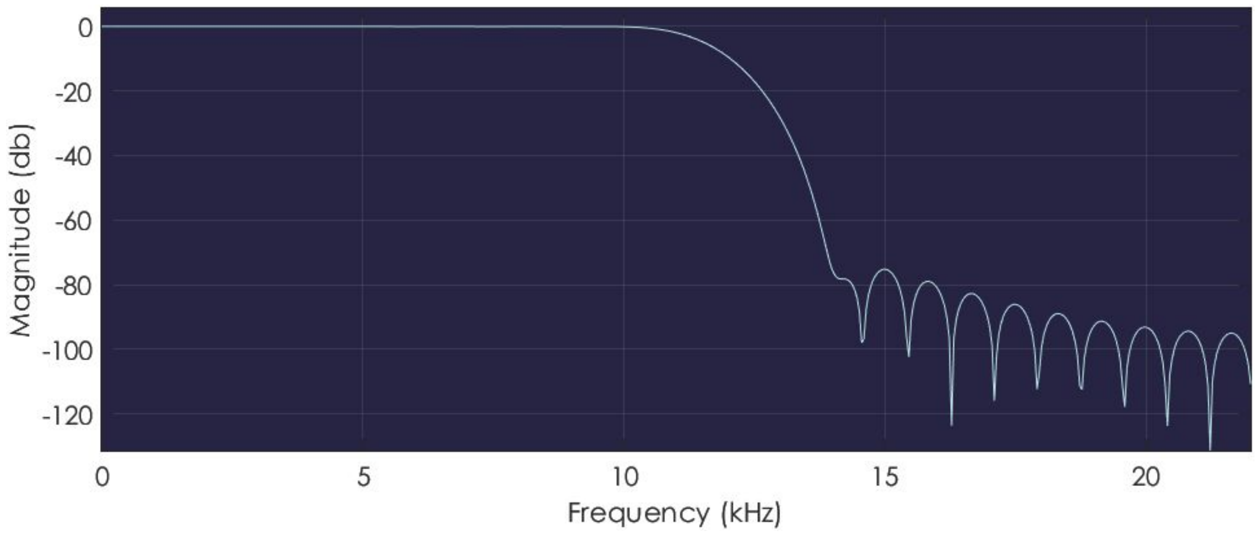


Fig.3. Filter Magnitude Response

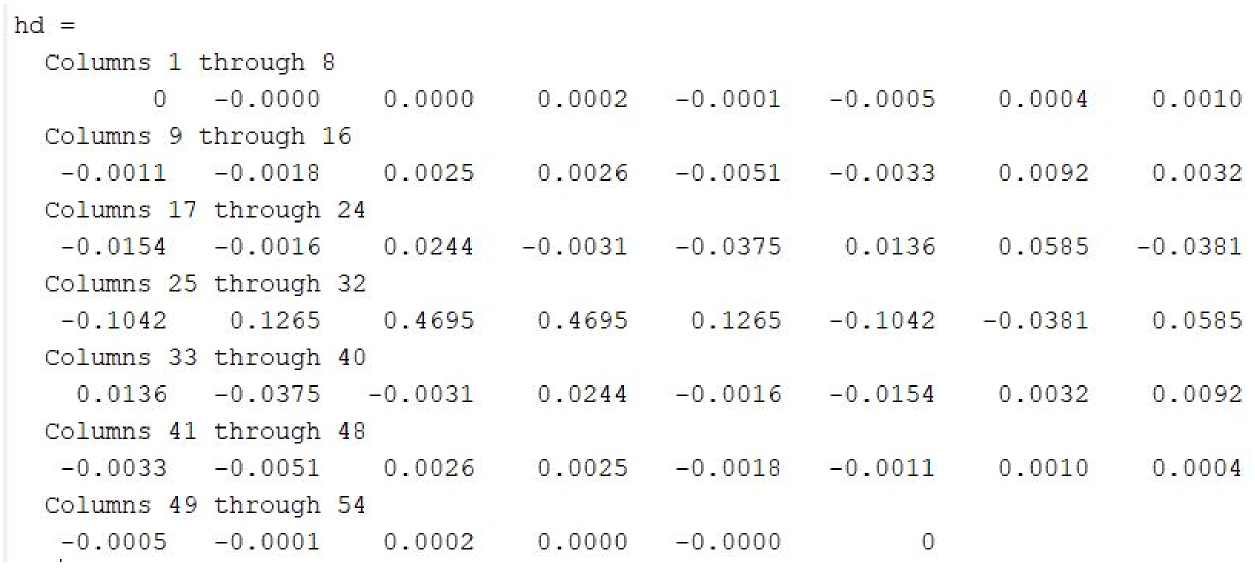

```
hd =
  Columns 1 through 8
         0   -0.0000    0.0000    0.0002   -0.0001   -0.0005    0.0004    0.0010
  Columns 9 through 16
   -0.0011   -0.0018    0.0025    0.0026   -0.0051   -0.0033    0.0092    0.0032
  Columns 17 through 24
   -0.0154   -0.0016    0.0244   -0.0031   -0.0375    0.0136    0.0585   -0.0381
  Columns 25 through 32
   -0.1042    0.1265    0.4695    0.4695    0.1265   -0.1042   -0.0381    0.0585
  Columns 33 through 40
    0.0136   -0.0375   -0.0031    0.0244   -0.0016   -0.0154    0.0032    0.0092
  Columns 41 through 48
   -0.0033   -0.0051    0.0026    0.0025   -0.0018   -0.0011    0.0010    0.0004
  Columns 49 through 54
   -0.0005   -0.0001    0.0002    0.0000   -0.0000         0
```

Fig.4. Filter Coefficients

Now, we compare the noisy and the filtered signal,

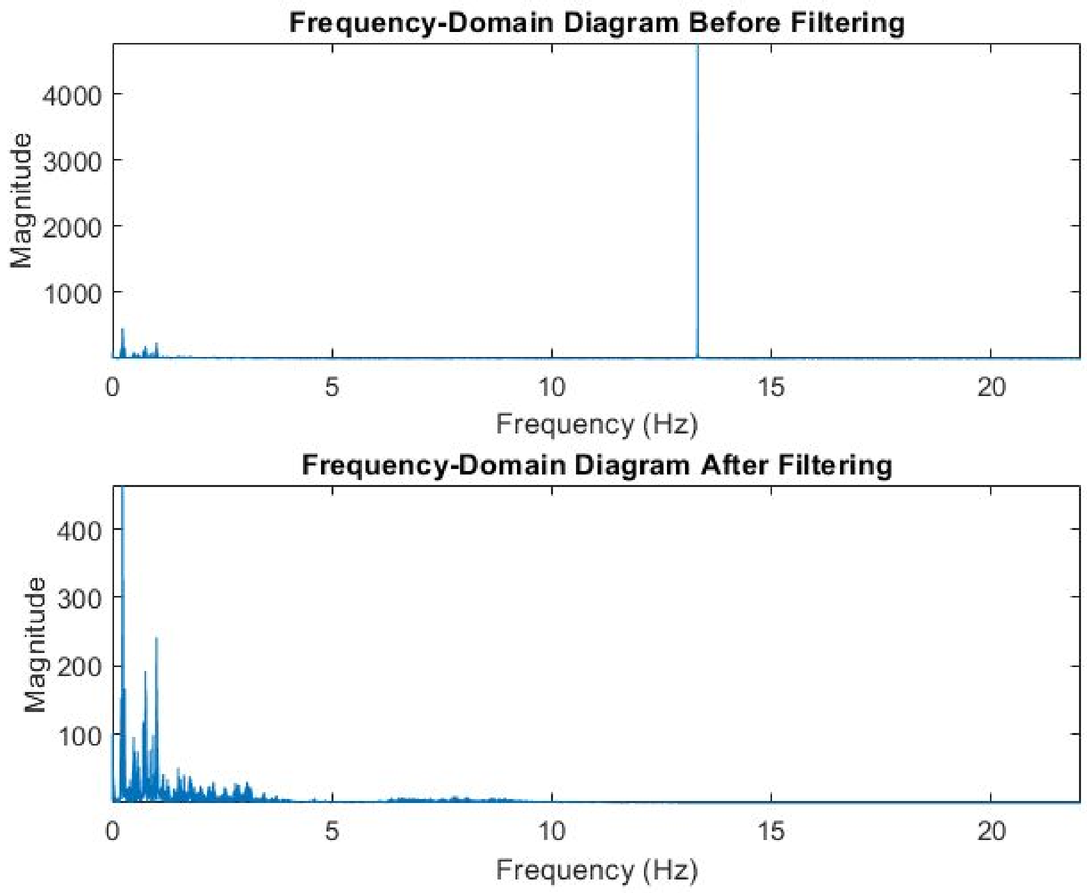


Fig.5. Comparing the frequency domain.

When listening to the two signals, we notice that the background noise is gone.

References

[1] Oppenheim, A.V., and R.W. Schafer. Discrete-Time Signal Processing. Upper Saddle River, NJ: Prentice-Hall, 1999, pp. 468-471.

[2] H. Rakshit and M. A. Ullah, “An efficient approach for designing FIR low-pass filter and its application,” 2015 2nd International Conference on Electrical Information and Communication Technologies (EICT), 2015.

Appendix

(A) Design Methods Functions

A.1 Window method

```
function hd = windowMethod(app)

    % 1 - Computing frequency vector
    f = repelem(2*app.fc/app.fs,2);
    freq = [0;f(:);1];

    % 2 - Computing magnitudes
    mag = rem(1+(0:1), 2);
    magnitude = repelem(mag(:),2);
    % 3 - Setting window according to its type.
    switch (app.windowType)
      case "Blackman"
        wind = blackman(app.nTabs);
      case "Chebyshev"
        sidelobeAtten = app.parEditField.Value;
        wind = chebwin(app.nTabs, sidelobeAtten);
      case "Kaiser"
        beta = app.parEditField.Value;
        wind = kaiser(app.nTabs, beta);
      case "Rectangular"
        wind = rectwin(app.nTabs);
    end
    wind = wind';

    % 4 - Calculating the unwindowed impulse response
    h_ideal = firls(app.nTabs - 1, freq, magnitude);

    % 5 - Windowing the impulse response in order to get the filter
    hd = h_ideal .* wind;
  end
```

A.2 east-squares

```
function [H_optimal,H_full] = leastSquares(N, k , w_cutoff)
%N :The length of the filter
%L :The number of taps in time domain(half the length of the filter -1)
%k :The number of points in the desired frequency response
%w_cutoff : The cutoff frequency in rad/sec
% 1 2cos(w1) 2cos(2w1) 2cos(3w1) .... 2cos(Lw1) h(0) H(w1)
% 1 2cos(w2) 2cos(2w2) 2cos(3w2) .... 2cos(Lw2) h(1) H(w2)
% . . .
% . * . = .
% . . .
% 1 2cos(wk) 2cos(2wk) 2cos(3wk) .... 2cos(Lwk) h(L) H(wk)
L = (N-1)/2;
w_k = pi * linspace(0,1,k); %the frequencies vector
H_d = zeros (1,k); % a vector that resembles the amplitude of the frequencies
H_d( w_k <= w_cutoff ) = 1; %Making the amplitude of frequencies less than the cutoff
frequency equals 1
%Constructing the F matrix
L_vector= 0:1:L;
Lw_matrix = w_k' * L_vector; %Constructing the Lw_k part which is inside the cosine
F = 2 * cos(Lw_matrix);
F(:,1)= F(:,1)/2; % To make the first column = 1. Or i can simply say --> F(:,1) = 1
H_optimal = F\ H_d';
H_optimal= (H_optimal)'; %The filter coefficients in the time domain
%constructing the full filter coefficients where
H_full = zeros(1,N);
H_full = (fliplr(H_optimal(2:end)));
H_full(L+1 : N) = H_optimal;
fvtool(H_full);
end
```

Weighted least squares

```
function [H_optimal,H_full] = weightedleastSquares(N, k , w_cutoff , weight_passband ,
weight_stopband)
  %N :The length of the filter
  %L :The number of taps in time domain(half the length of the filter -1)
  %k :The number of points in the desired frequency response
```

```
    %w_cutoff  : The cutoff frequency in rad/sec

%     1  2cos(w1)    2cos(2w1)   2cos(3w1) .... 2cos(Lw1)     h(0)       H(w1)
%     1  2cos(w2)    2cos(2w2)   2cos(3w2) .... 2cos(Lw2)     h(1)       H(w2)
%     .                                          .          .
%     .                                        *  .     =    .
%     .                                          .          .
%     1  2cos(wk)    2cos(2wk)   2cos(3wk) .... 2cos(Lwk)     h(L)       H(wk)


    L = (N-1)/2;

    w_k = pi * linspace(0,1,k);  %the frequencies vector
    H_d = zeros (1,k); % a vector that resembles the amplitude of the frequencies
    H_d( w_k <= w_cutoff ) = 1;  %Making the amplitude of frequencies less than the cutoff
frequency equals 1

    %Constructing the Weight matrix
    V=zeros(1,k);
    V( w_k <= w_cutoff ) = weight_passband;  %Making the weight of frequencies less than the
cutoff frequency equals weight_passband
    V( w_cutoff <= w_k ) = weight_stopband;  %Making the weight of frequencies more than the
cutoff frequency equals weight_stopband
    W = diag(V);

    %Constructing the F matrix
    L_vector= 0:1:L;
    Lw_matrix = w_k' * L_vector; %Constructing the Lw_k part which is inside the cosine
    F = 2 * cos(Lw_matrix);
    F(:,1)= F(:,1)/2; % To make the first column = 1. Or i can simply say -->  F(:,1) = 1

    H_optimal =  (W* F)\(W* H_d');
    H_optimal= (H_optimal)';     %The filter coefficients in the time domain

    %constructing the full filter coefficients where
    H_full = zeros(1,N);
    H_full = (fliplr(H_optimal(2:end)));
    H_full(L+1 : N) =  H_optimal;

    fvtool(H_full);

end
```