DSP HW3

Dmitrii Masnyi

December 2021

Task 1

Using at least 3 different windows, implement frequency domain approximations of an ideal bandpass filter to pass a signal within frequencies 6MHz and 8MHz with the attenuation outside (<5.5MHz and >8.5MHz) 60dB, and the ripple<=0.5dB within the passband. Find a minimal size of each window (min filter order). Sampling frequency is Fs=30MHz. Provide code.

Solution:
I went to the documentation page of the fir functions in Matlab. It is recommended to use fir1 function to design windows-based bandpass filters. Also, I found out that there are a lot of built in matlab functions for windows link to documentation. So I used three of them. I just tuned the number of taps till the filter does not fit requirements.

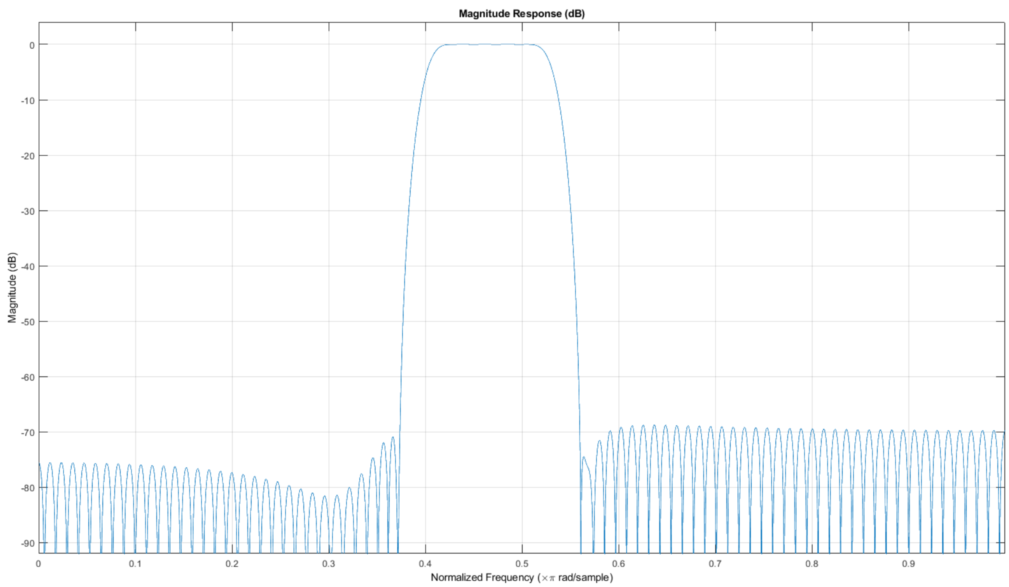

Figure 1: Chebyshev window, n = 170 digital filters.

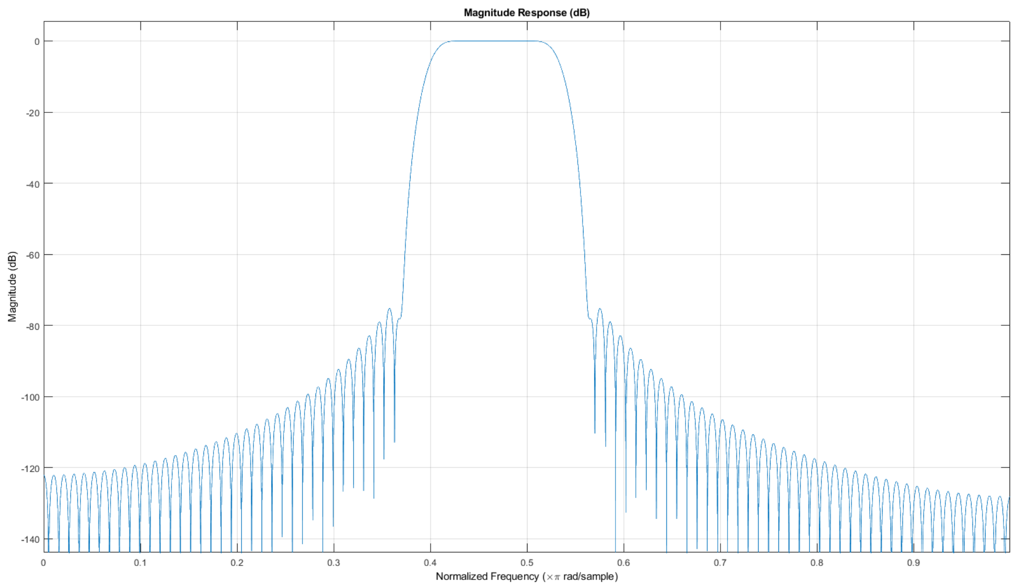

Figure 2: Blackman window, n = 190.

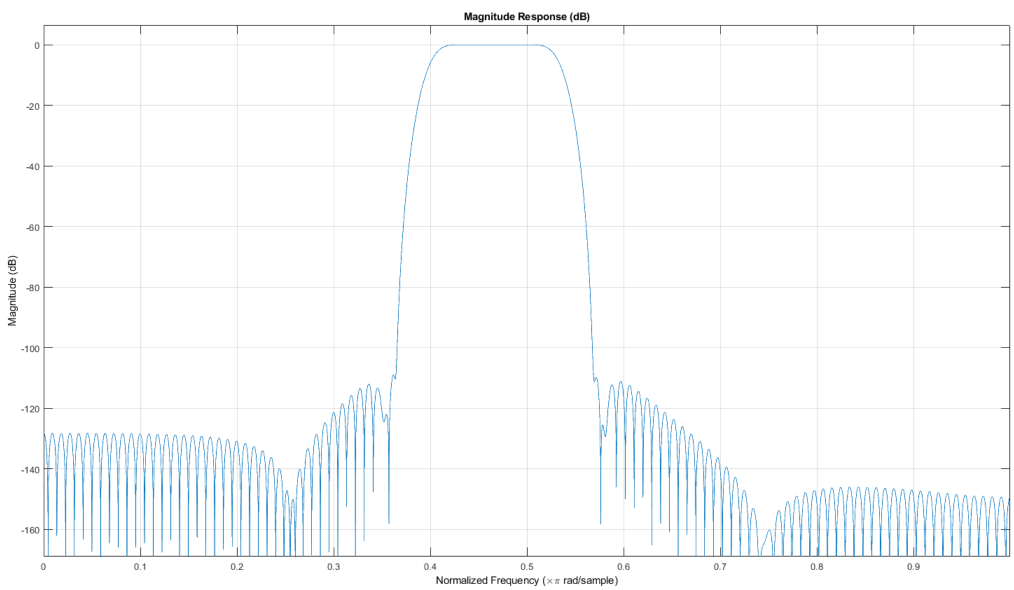

Figure 3: Blackman-Harris window, n = 220.

```
Fpass_left = 6e6;
Fstop_lest = 5.5e6;
Fpass_right = 8e6;
Fstop_right = 8.5e6;
Ripple = 0.5;
Attenuation = 60;
Fs = 30e6;
% chebyshev window window
n = 170;
f1 = fir1(n, [2*Fpass_left/Fs 2*Fpass_right/Fs], 'bandpass', chebwin(n+1, Attenuation));
%fvtool(f1)
% blackman window
n = 190;
f2 = fir1(n, [2*Fpass_left/Fs 2*Fpass_right/Fs], 'bandpass', blackman(n+1));
%fvtool(f2)
% blackmanharris window
n = 220;
f3 = fir1(n, [2*Fpass_left/Fs 2*Fpass_right/Fs], 'bandpass', blackmanharris(n+1));
fvtool(f3)
```

Filter designed using Chebyshev window has the lowest complexity among other designed filters (n = 170), so it is preferable to use it.

Task 2

Approximate a filter with the frequency response:

$$H(f) = \begin{cases} e^{\frac{|f|}{10^7}}, & \text{|f|} < 5 \text{ MHz} \\ 0, & \text{|f|} > 6 \text{ MHz} \end{cases}$$

Let the sampling frequency be Fs = 25 MHz, and the attenuation in the stopband be 60dB. Determine the impulse response of a FIR filter, which approximates this frequency response. Plot the frequency response in terms of magnitude and phase to verify that the approximation holds. Provide code.

Solution:
First of all, I am going utilize embedded Matlab function for window creation. For this reason I need to normalize given frequencies and convert it to the radians per seconds (otherwise this functions will not work).
We have a proportion:

$$Fs = 2\pi$$
$$f = \omega$$
$$\rightarrow f = \frac{\omega Fs}{2\pi}$$

$$|f| < 5 \rightarrow \frac{-10\pi}{Fs} < \omega < \frac{10\pi}{Fs} \rightarrow -0.4\pi < \omega < 0.4\pi$$

$$|f| > 6 \rightarrow \left[\begin{array}{l} \frac{-12\pi}{Fs} > \omega \\ \frac{12\pi}{Fs} < \omega \end{array}\right. \rightarrow \left[\begin{array}{l} -0.48\pi > \omega \\ 0.48\pi < \omega \end{array}\right.$$

Then we can rewrite initial frequency response with recalculated limits:

$$H(\omega) = \begin{cases} e^{\frac{-Fs|\omega|}{2\pi 10^7}}, -0.4\pi < \omega < 0.4\pi \\ 0, \left[\begin{array}{l} -0.48\pi > \omega \\ 0.48\pi < \omega \end{array}\right. \end{cases}$$

Finally:

$$H(\omega) = \begin{cases} e^{\frac{-1.25|\omega|}{\pi}}, -0.4\pi < \omega < 0.4\pi \\ 0, \left[\begin{array}{l} -0.48\pi > \omega \\ 0.48\pi < \omega \end{array}\right. \end{cases}$$

For the simplicity of the further calculation let's assume that in the regions (-0.48π; -0.4π) and (0.4π; 0.48π) $H(w) = 0$ (like in the case of ideal filter). Then I calculate impulse response of the filter by calculating the IDTFT of $H(\omega)$

$$h(n) = \frac{1}{2\pi}\int_{-0.4\pi}^{0.4\pi} e^{\frac{-1.25|\omega|}{\pi}} e^{j\omega n} d\omega = \frac{1}{2\pi}\int_{-0.4\pi}^{0} e^{\omega(\frac{1.25}{\pi}+jn)} d\omega + \frac{1}{2\pi}\int_{0}^{0.4\pi} e^{\omega(\frac{-1.25}{\pi}+jn)} d\omega = I_1 + I_2$$

$$I_1 = \frac{1}{2\pi}\int_{-0.4\pi}^{0} e^{\omega(\frac{1.25}{\pi}+jn)} d\omega = \frac{1}{2\pi}\cdot\frac{\pi(1-e^{-0.5-0.4j\pi n})}{1.25+j\pi n} = \frac{1-e^{-0.5-0.4j\pi n}}{2.5+2j\pi n}$$

$$I_2 = \frac{1}{2\pi}\int_{0}^{0.4\pi} e^{\omega(\frac{-1.25}{\pi}+jn)} d\omega = \frac{1}{2\pi}\cdot\frac{\pi(e^{-0.5+0.4j\pi n}-1)}{-1.25+j\pi n} = \frac{e^{-0.5+0.4j\pi n}-1}{-2.5+2j\pi n}$$

Finally:

$$h(n) = I_1 + I_2 = \frac{1 - e^{-0.5-0.4j\pi n}}{2.5 + 2j\pi n} + \frac{e^{-0.5+0.4j\pi n} - 1}{-2.5 + 2j\pi n}$$

Then I chose window from the previous task (Blackman window) and find n = 94 with which filter satisfies requirements.

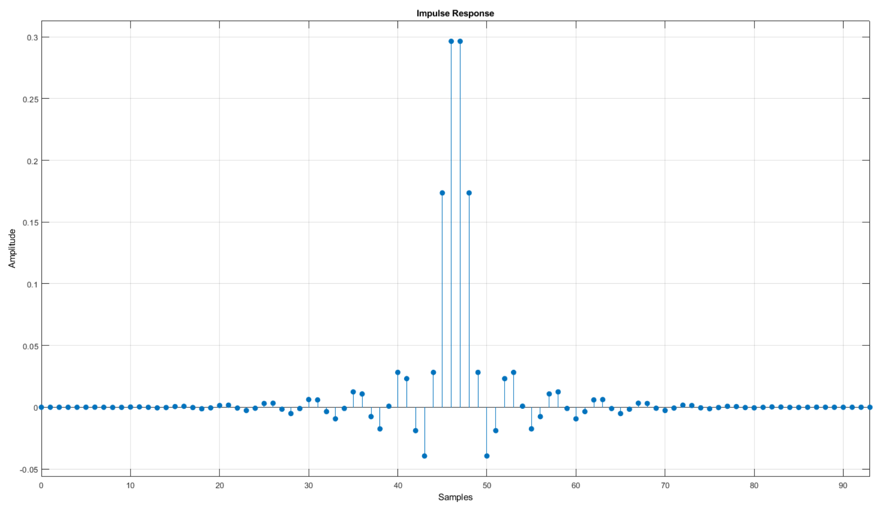

Figure 4: Impulse response.

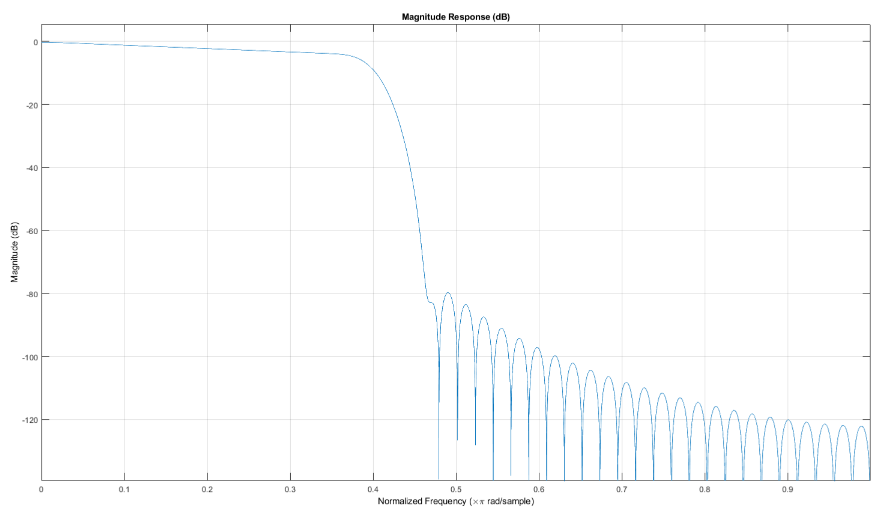

Figure 5: Magnitude response.

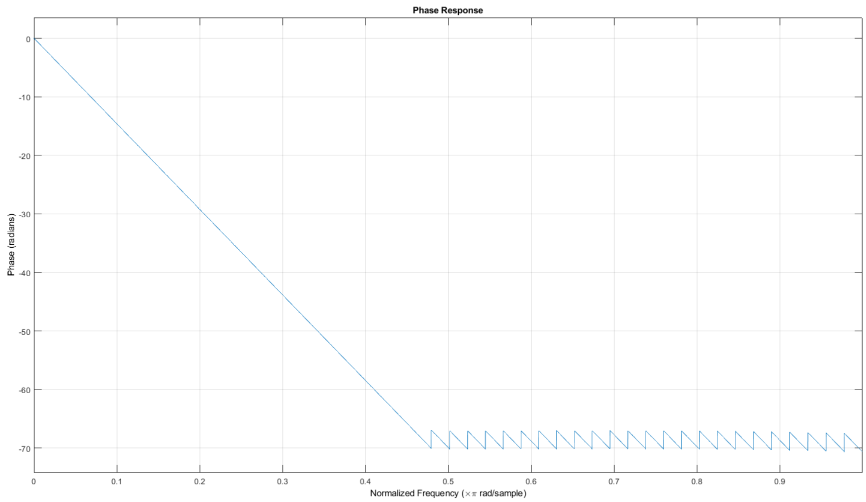

Figure 6: Phase response.

According to figure 5 (magnitude response) and figure 6 (phase response) the approximation holds, because the phase in passband is linear and attenuation in stopband is more then 60dB.

```
n = linspace(-47, 47, 94);
h_id  = (1-exp(-0.5-0.4.*pi.*n.*j))./(2.5+2.*j.*pi.*n) +
(exp(-0.5+0.4.*pi.*n.*j)-1)./(-2.5+2.*j.*pi.*n);
win = blackman(length(n));
h = real(h_id) .* transpose(win);
fvtool(h)
```

Task 3

Design a low pass filter with passband Fpass=4 MHz, stopband Fstop=6MHz, attenuation of at least 40dB,sampling frequency Fs=20MHz. Design at least 3 versions of multiplier-free FIR filters with the least order. You can combine low-order filters to solve the problem. Plot impulse and frequency responses. Compare with a common FIR filter. Provide code.

Solution:

The first approach to design multiplierless filter is to decompose coefficients of the designed filter to the summands, which are w in the power k, where k lies in Z. This method will remove multiplicators from the scheme, because multiplication by the power of 2 equals to the shift of the number in the binary representation. For this approach I found an article with implementation of this method. So I use it as my solution. Important to note, that I slightly tuned Matlab code that was shown in the article in order to use it for different filters.

My solution is quite strange:

- Firstly, I design a filter and plot its magnitude response (to show that it satisfies requirements).
- Secondly, I use the function from article and decompose coefficients of the designed filter in the way I described above.
- Then I normalize approximated coefficients (from previous step) and coefficients of the initial filter. I do it because of some artifacts of functions from paper (it plots shifted by the Y-axis magnitude response). Then I plot magnitude response of the initial filter and of the filter with approximated coefficients. I can say that normalization is not a problem, because the aim of this step is to show that the approximation of the filter's coefficients is quite good and if the curves of the magnitude response coincide it means that everything is good. Hope that my explanation is clear.

Filter 1: fir1 with Hamming window

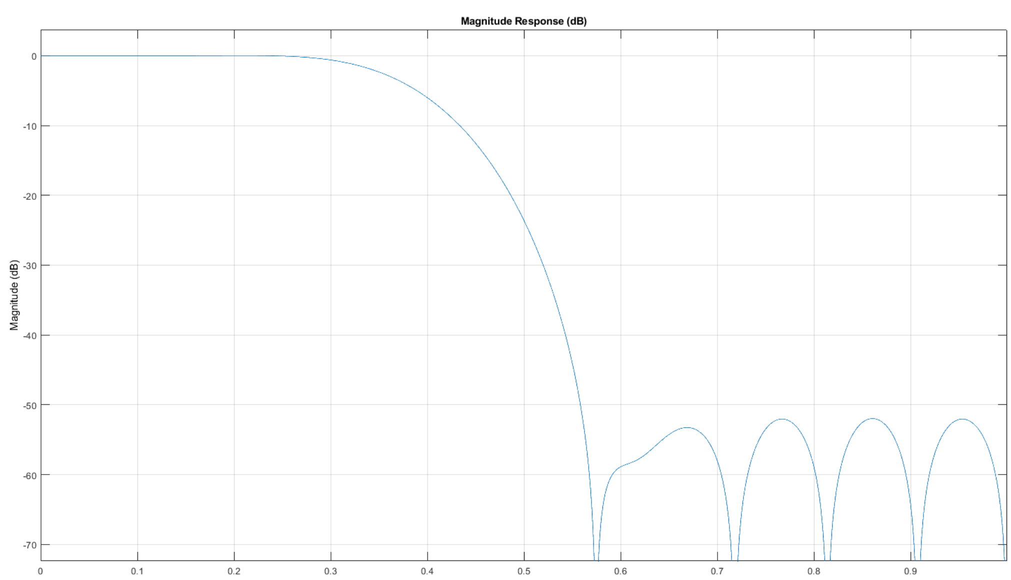

Figure 7: Hamming window, n = 21.

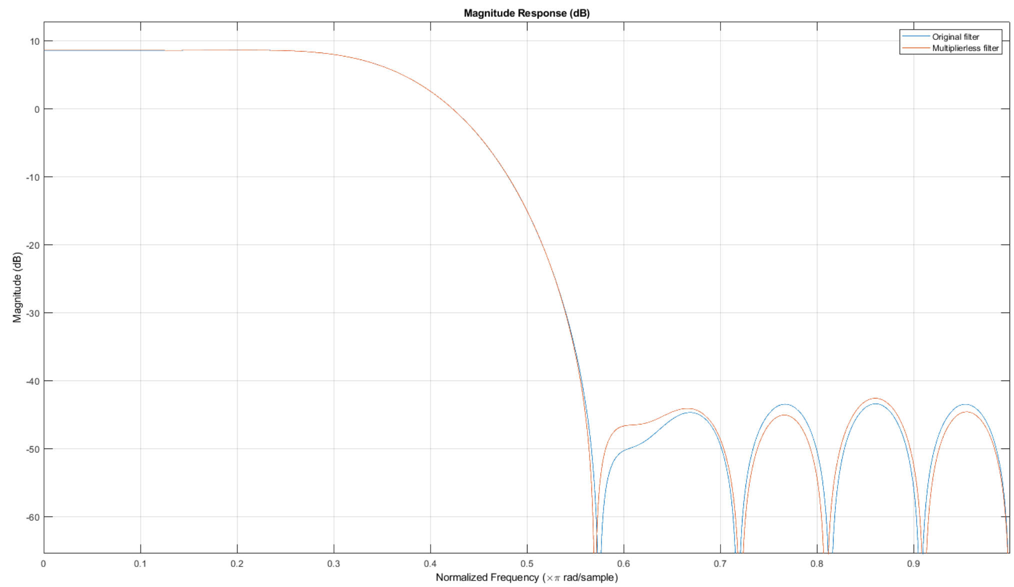

Figure 8: Hamming window, n = 21, nbits = 11.

Filter 2: fir1 with Chebyshev window

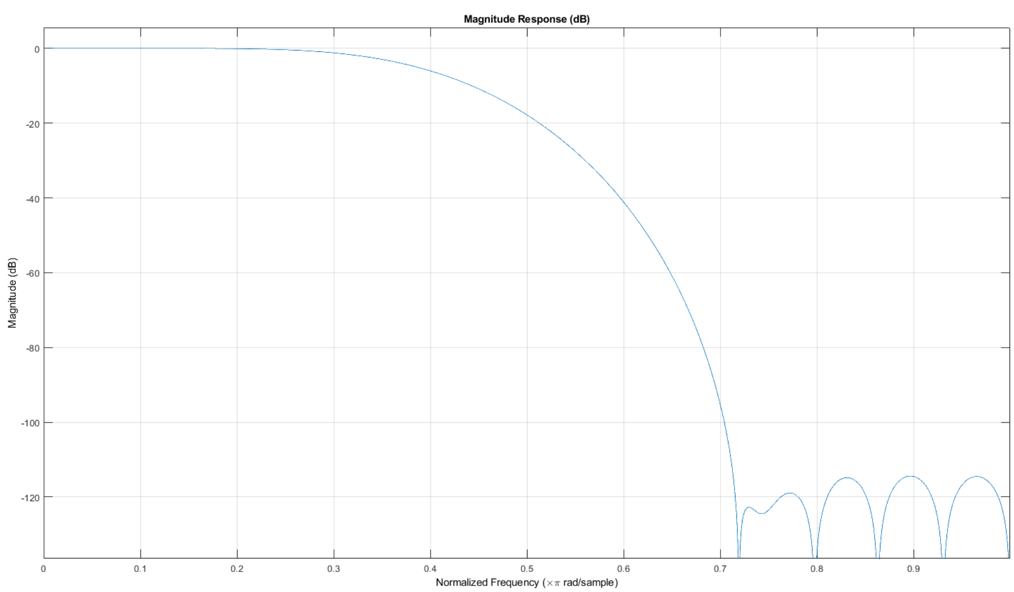

Figure 9: Chebyshev window, n = 23.

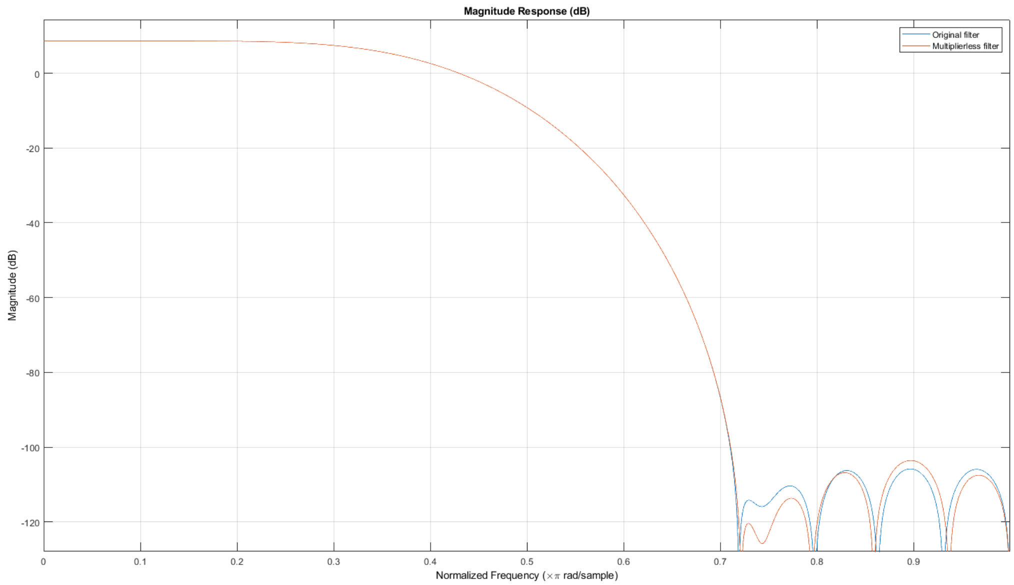

Figure 10: Chebyshev window, n = 23, nbits = 20.

Filter 3: fir1 with Blackmanharris window

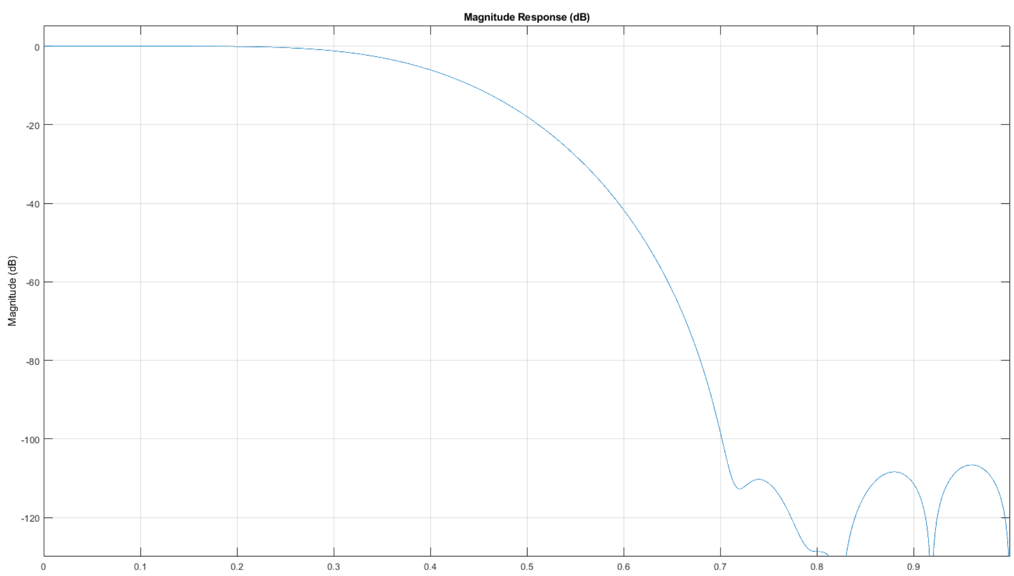

Figure 11: Blackmanharris window, n = 25.

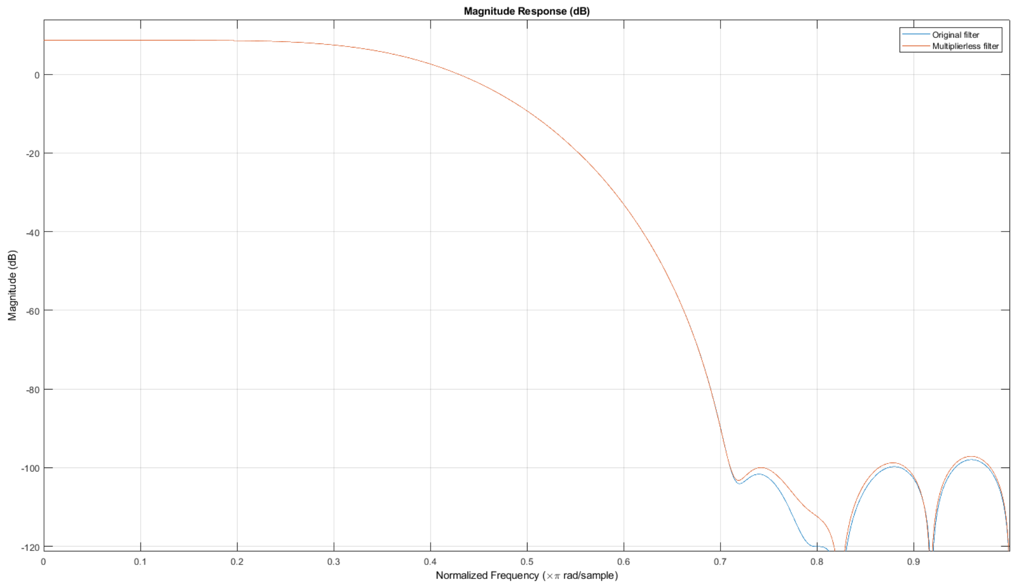

Figure 12: Blackmanharris window, n = 25, nbits = 20.

So nbits should be taken into account to, because the more this parameter the more accurate is the approximation, but the higher the complexity as well (increase the number of summators in the scheme). Overall, the best filter for this task is the first one, because it has the lowest n and nbits.

```
Fpass = 4e6;
Fstop = 6e6;
Fs = 20e6;
Attenuation = 40;
%filter 1
nbits = 11;% defines the quality of approximation
n = 21;
f1 = fir1(n,2*Fpass/Fs, 'low');
%fvtool(f1)
approx_coeff1 = csd_lowpass(f1, n, nbits);
f1 = f1/max(f1); % normalization
approx_coeff1 = approx_coeff1/max(approx_coeff1); % normalization
%fvtool(f1, 1, approx_coeff1, 1); %orange one is approximated
%legend('Original filter', 'Multiplierless filter')
%filter 2
nbits = 20;% defines the quality of approximation
n = 23;
f2 = fir1(n,2*Fpass/Fs, 'low', chebwin(n+1));
%fvtool(f2)
approx_coeff2 = csd_lowpass(f2, n, nbits);
f2 = f2/max(f2); % normalization
approx_coeff2 = approx_coeff2/max(approx_coeff2); % normalization
%fvtool(f2, 1, approx_coeff2, 1); %orange one is approximated
%legend('Original filter', 'Multiplierless filter')
% filter 3
nbits = 20;% defines the quality of approximation
n = 25;
```

```
f3 = fir1(n,2*Fpass/Fs, 'low', blackmanharris(n+1));
%fvtool(f3)
approx_coeff3 = csd_lowpass(f3, n, nbits);
f3 = f3/max(f3); % normalization
approx_coeff3 = approx_coeff3/max(approx_coeff3); % normalization
fvtool(f3, 1, approx_coeff3, 1); %orange one is approximated
legend('Original filter', 'Multiplierless filter')
%%% code from paper
%function [b_opt,B]= csd_lowpass(ntaps,nbits,fpass,fstop,fs)
% 10/30/2016 Neil Robertson
%
% Synthesize FIR LPF with CSD coeffs
%
% ntaps      number of FIR coeffs
% nbits      number of bits per coeff
% fpass      passband edge freq, Hz, kHz, or MHz
% fstop      stopband edge freq, Hz, kHz, or MHz
% fs         sample freq, Hz, kHz, or MHz
%
% b_opt      decimal integer coefficients of LPF
% B          CSD coefficients of LPF (exactly equivalent to b_opt)
%
% Examples
% csd_lowpass(17,9,10,30,100);
% csd_lowpass(27,11,10,25,100);
%
function [b_opt,B]= csd_lowpass(b, ntaps, nbits)

% 2. SEARCH for CSD coeffs with lowest number of signed digits (nsd)
b= b/max(b);                    % make maintap= 1
nsd_thresh= 2;                  % threshold used to compute error
if nbits > 10
   nsd_thresh=3;
end
stop= fix(2/3*2^nbits);         % max allowed CSD value for coeff of length nbit
start= max(2^(nbits-1),stop-600); % starting maintap value. start > stop -600
emin= 999999;
for maintap= start:stop
   b_int=round(b*maintap);       % decimal integer coefficients
   Y= dec2csd1(b_int,nbits);      % compute CSD representation of b_int
   for i=1:ntaps
      nsd(i)= sum(abs(Y(i,:)));      % number of signed digits in coeff i
   end
   m= find(nsd> nsd_thresh);      %find indeces of coeffs that have nsd > nsd_thresh
   e= sum(nsd(m));                % sum of nsd's for those coeffs
   if e <=emin
      emin= e;
      Yopt= Y;                    % CSD coeffs with least nsd's.
      b_opt= b_int;               % integer version of above
   end
end
%
% 3. Compute nsd of coeffs and external gain value
for i= 1:ntaps
```

```
   nsd(i)= sum(abs(Yopt(i,:)));        % number of signed digits in coeff i
end
gain_ext= 2^(nbits+1)/sum(b_opt);    % gain to make overall dc gain = 1
gain_approx= round(gain_ext*16)/16;    % approx gain
gain_rat= rats(gain_approx);
disp(' ')
disp(['coeff denominator = ',num2str(2^(nbits+1))])
disp(['external gain for unity overall gain: ',num2str(gain_ext)])
disp(['approximate external gain =',num2str(gain_rat)])
% 4.  List coeff values in decimal and CSD formats
disp(' ')
disp('fixed-point coeff values')
disp(b_opt')
B = [fliplr(Yopt)];
disp('CSD coeffs, MSB on left;     nsd')
disp(' ')
for i= 1:ntaps
    disp([num2str(B(i,:)),'       ',num2str(nsd(i))])
end
%%% second file from article
% Y= dec2csd1(b_int,nbits)           10/25/16   Neil Robertson
%
% Convert signed decimal integers to binary CSD representation
% See Ruiz and Granda, "Efficient Canonic Signed DigitRecoding",
%      Microelectronics Journal 42, p 1090-1097, 2011
%
% b_int = vector of decimal integer coefficients
% nbits = number of bits in b_int coeffs
% Y = matrix of CSD coeffs
% A = matrix of binary coeffs
%
%                     -- j= 1:nbits --
%                    _                _
%                   | ---- coeff 1 ---- |
%                   | ---- coeff 2 ---- |
%                   |  .       .      . |
%        Y,A =      | ---- coeff i ---- |
%                   |  .       .      . |
%                   | -- coeff ntaps -- |
%                    -                -
% 1.  convert decimal integers to binary integers
function Y= dec2csd1(b_int,nbits)
ntaps= length(b_int);        % number of coeffs
for i= 1:ntaps               % coeff index (row index)
   u= abs(b_int(i));
   for j= 1:nbits            % binary digit index (column index)
   A(i,j)= mod(u,2);      % coeff magnitudes note:  MSB is on right.
   u= fix(u/2);
   end
end
% 2.  convert binary integers to CSD
s= sign(b_int);              % signs of coeffs
z= zeros(ntaps,1);
x= [A z];                    % MSB is on right. append 0 as MSB
```

```
for i= 1:ntaps                    % coeff index (row index)
    c= 0;
    for j= 1:nbits            % binary digit index (column index)
    d= xor(x(i,j),c);
    ys= x(i,j+1)&d;                  % sign bit    0 == pos, 1 == negative
    yd= ~x(i,j+1)&d;                 % data bit
    Y(i,j)= yd - ys;                 % signed digit
    c_next = (x(i,j)&x(i,j+1)) | ((x(i,j)|x(i,j+1))&c);       % carry
    c= c_next;
    end
Y(i,:) = Y(i,:)*s(i);                % multiply CSD coeff magnitude by coeff sign
end
```

Another approach is to design a CIC filter, but I received very bad results. I used function filterBuilder in command line ,then I specified required parameters:

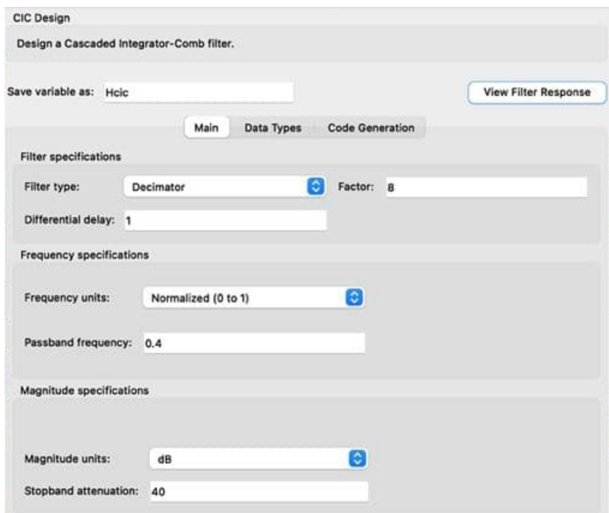

Figure 13: CIC filter parameters

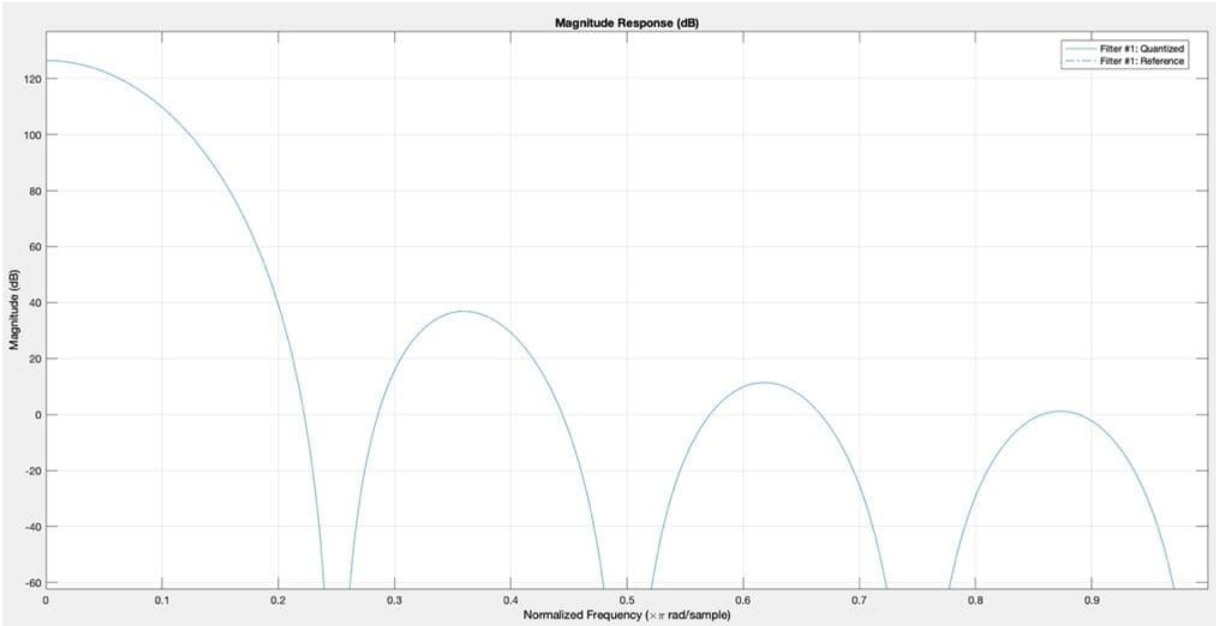

Figure 14: Magnitude response of CIC filter.

```
Discrete-Time FIR Multirate Filter (real)
-----------------------------------------
Filter Structure    : Cascaded Integrator-Comb Decimator
Decimation Factor   : 8
Differential Delay  : 1
Number of Sections  : 7
Stable              : Yes
Linear Phase        : Yes (Type 2)


Design Options
SystemObject : true

Implementation Cost
Number of Multipliers            : 0
Number of Adders                 : 14
Number of States                 : 14
Multiplications per Input Sample : 0
Additions per Input Sample       : 7.875
```

Figure 15: Parameters of CIC filter.

Despite the fact that this filter has no multiplier it does not satisfy the requirements. If we decrease the factor of the filter in the initial setup to 1 it will give us the first lobe of sinc-like function. This filter requires pre-equalization filter that will have the multipliers, so CIC filter is not suitable for solving this task.

Task 4

A signal is presented by ongoing samples in the time domain (online signal). Compare complexity of 2 filtration approaches: in the time domain (using convolution) and in the frequency domain (using FFT). The result should be in the time domain as well.
a) How to realize filtration in both cases (give a detailed answer).
b) Describe advantages and disadvantages of each approach.
c) Compare complexity of methods. When the freq. domain is preferable? Give an example.
Solution:

- a) If we filter signal in the time domain, we calculate the convolution of the signal with the impulse response of the filter: $y(t) = x(t) * h(t)$. For the filtration if the frequency domain I need to calculate the FT of the signal $x(t) \to X(\omega)$ then multiply it by transfer function of the filter $H(\omega)$ and then compute the IFT $X(\omega)H(\omega) \to x_{filt}(t)$
- b) In case of filtration in time domain we can calculate the convolution immediately without any delays,but this approach has complexity $O(N^2)$, where N is the length of the ongoing signal and impulse response (just for simplicity N is equal).
  In case of the filtration in the frequency domain we need to compute FFT at first, for this reason we need to accumulate the batch for this computation, so the filtration has the delay, but the complexity of this approach is less (FFT + multiplication + IFFT) which is approximately $O(Nlog(N))$.
- c) As I write above, the complexity of the frequency domain method is less, but it has a delay. Also, less complexity means less power consumption. But anyway, if the length of the sampled signal is high, the second approach is preferable. If the system has limits on delay (filtration should be almost simultaneous), filtration in time domain is preferable.

Task 5

In the DAC we want to use the linear interpolation between samples instead of the Sample and Hold, as shown in the figure below. This is a First Order Hold reconstruction.
a) Show that $x(t) = \sum_{n=-\infty}^{\infty} x[n]g(t - nT_s)$, where g(t) - triangle pulse.
b) Calculate frequency response of DAC output, considering that x(t) is the band-limited white noise with bandwidth 4 times lower than Fs.
c) Show the difference between Sample and Hold and First Order Hold.Plot impulse and frequency responses. Provide code.
Solution:

- a) By definition of the sampled signal it can be represented as the multiplication of the continuous time signal by the impulse train:

  $$x_s(t) = x(t) \sum_{n=-\infty}^{\infty} \delta(t - nT_s) = \sum_{n=-\infty}^{\infty} x(nT_s)\delta(t - nT_s)$$

  To restore an analog signal by its discrete samples we need to calculate convolution of the sampled signal and the triangle pulse in the time domain (Idea is that the DAC can be treated as a low-pass filter, I mean that if we do signal restoration in the frequency domain we cut the aliased copies of the signal that is indeed low-pass filtration):

  $$x(t) = g(t) * x_s(t) = g(t) * \sum_{n=-\infty}^{\infty} x(nT_s)\delta(t - nT_s) = \sum_{n=-\infty}^{\infty} x(nT_s)g(t) * \delta(t - nT_s)$$

By the property of the convolution with delta function:

$$f(t) * \delta(t-a) = f(t-a)$$

And note that:

$$x[n] = x(nTs)$$

We receive:

$$x(t) = g(t) * x_s(t) = g(t) * \sum_{n=-\infty}^{\infty} x(nT_s)\delta(t-nT_s) = \sum_{n=-\infty}^{\infty} x(nT_s)g(t) * \delta(t-nT_s) = \sum_{n=-\infty}^{\infty} x[n]g(t-nTs)$$

- b) Solution is quite simple, I generated white noise through built-in Matlab function. Then I limited its bandwidth and calculated the convolution with the triangle pulse. This procedure gave me output of white noise fed to the FOH. Then I used fvtool() function to plot it.

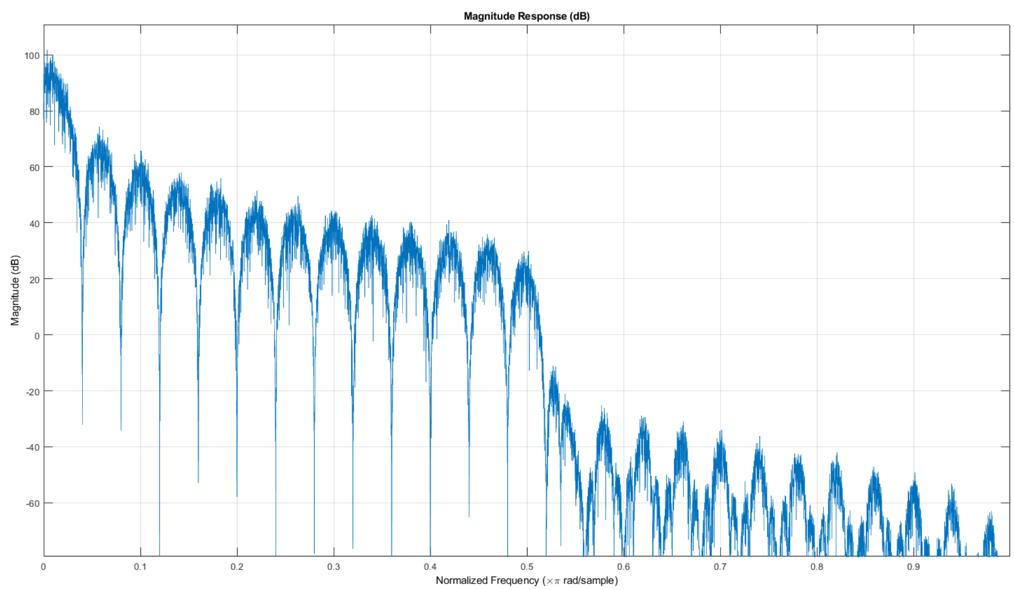

Figure 16: Magnitude response of FOH fed with band limited white noise.

- c) FOH can be represented as triangular window in time domain and SH can be represented as rectangular window in the time domain. So I just utilized fvtool() function to plot impulse and frequency responses of these two windows.

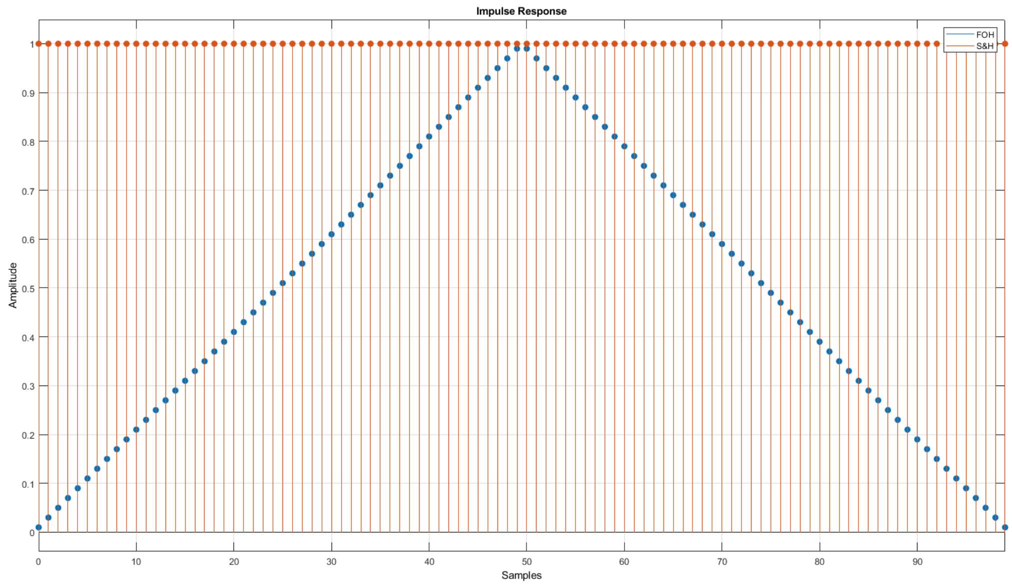

Figure 17: Impulse response of FOH and S&H.

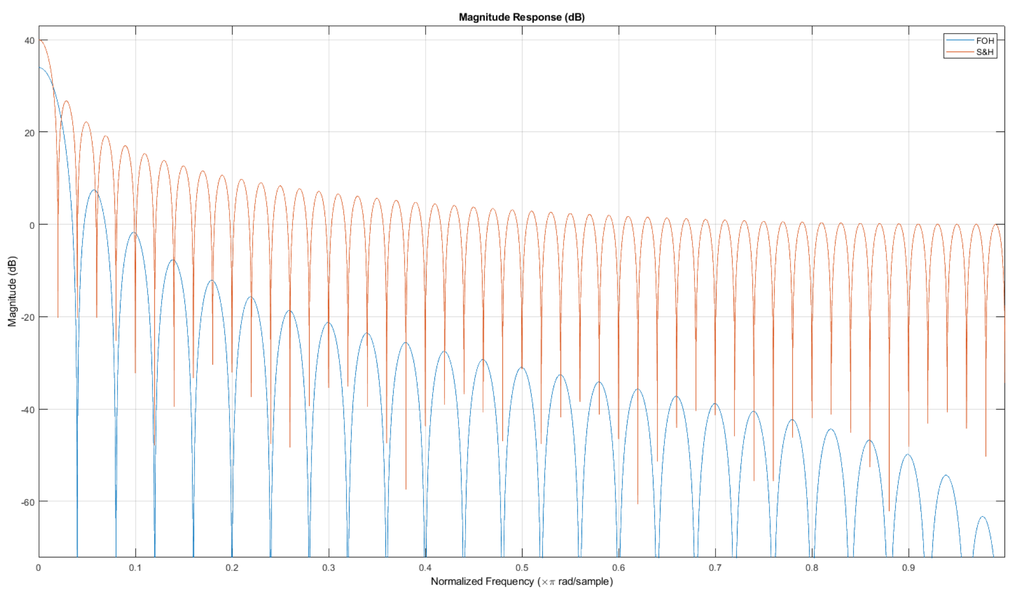

Figure 18: Frequency response of FOH and S&H.

```
N = 10e5;
x = randn(1, N); % input - white noise
n = 100;
Fpass = 4e6;
Fs = 4*Fpass;
filter = fir1(n,2*Fpass/Fs,'low');
x_filtered = conv(x, filter); %filtering white noise
tr = triang(100);
```

```
y = conv(x_filtered, tr);
%fvtool(y); % task b - plotting freq resp of filtered white noise
fvtool(tr, 1, rectwin(100), 1) % plotting imp resp and freq resp of FOH and S&H
legend('FOH','S&H')
```