DSP Project

Part 1: Spectral estimation

1) Code that reads the audio file and store the sampling frequency

```
[audio_data, Fs] = audioread('music_test_fayrouz.mp3');
audio_length = length(audio_data);
```

This code reads the audio file using audioread, determines the number of audio samples and stores the length of the audio.

2) Capture 3 seconds from the middle of the audio file and store in in vector

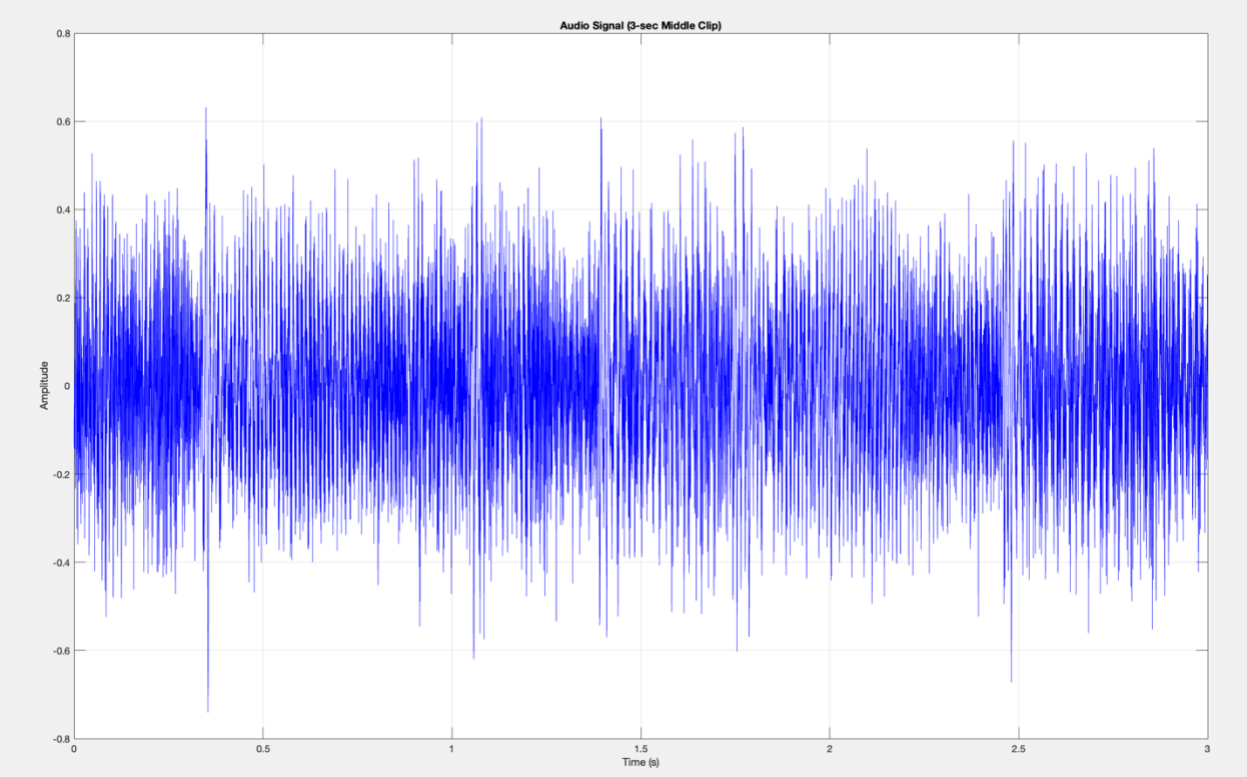
```
% Extract 3 seconds from the middle
mid_index = floor(audio_length / 2);
samples_3s = round(3 * Fs);
start_index = mid_index - floor(samples_3s / 2);
end_index = start_index + samples_3s - 1;
audio_clip = audio_data(start_index:end_index, :);

t = (0:length(audio_clip)-1) / Fs;
```

The code extracts a 3-second segment from the middle of the audio signal by calculating the midpoint and selecting an equal number of samples before and after it. A corresponding time vector t is generated by dividing sample indices by the sampling frequency Fs, enabling time-domain plotting.

3) Plot the audio signal versus time in seconds.

```
figure;
plot(t, audio_clip(:,1), 'b');
xlabel('Time (s)');
ylabel('Amplitude');
title('The 3-sec Audio Signal');
grid on;
```

The code plot the 3-sec audio signal without the interference.



The plot of the 3-sec audio signal without the interference.

4) Generate a sinusoidal interference tone at 15.2 KHz, with the same Fs and of Amplitude 1.8

```
A = 1.8;
f_interf = 15200;
[rows, cols] = size(audio_clip);
interference = A * sin(2 * pi * f_interf * t)';
if cols > 1
    interference = repmat(interference, 1, cols);
end
```

A 15.2 kHz sinusoidal interference signal is generated with an amplitude of 1.8. The interference is created to match the dimensions of the input audio clip so that it can be added directly to the audio without causing dimension mismatch errors. “if cols > 1” checks If the audio is multi-channel, the signal is replicated by repmat function across all channels to maintain compatibility it is essential because all new songs have multiple channels.

5) Add the interference to the audio signal and listen to the audio signal with and without the interference signal.

```
audio_with_interf = audio_clip + interference;
sound(audio_clip, Fs);
pause(4);
sound(audio_with_interf, Fs);
```

The code plays two versions of an audio clip: the original and one with added interference. The first line adds the interference signal to the original audio to create a new variable "audio_with_interf". The sound function is then used to play the original audio, followed by a 4-second pause "pause(4)" to allow it to finish. After the pause, the noisy version "audio_with_interf "is played.

6) Plot the single sided power spectrum of the audio + interference signal using the Welch spectral estimation algorithm and specify the parameters in the Welch spectral estimation function

```
function plot_spectrum(signal, Fs, windowing)
 % This function plots the PSD of a real signal (single sided spectrum)
 Nfft = 2^10; % FFT size used in the spectrum analyzer
 percentage_overlap = 0.5; % 50% overlap between segments
 window_length = Nfft; % length of each segment (can be <= NFFT)

 switch windowing
 case 1
   kaiser_beta = 7;
   window = kaiser(window_length, kaiser_beta);
 case 2
   window = ones(window_length, 1);
 case 3
   window = hann(window_length);
 case 4
   window = hamming(window_length);
 otherwise
   error('Invalid windowing option. Use 1=Kaiser, 2=Rectangular, 3=Hanning, 4=Hamming.');
 end

 PSD = pwelch(signal, window, percentage_overlap * window_length, Nfft, Fs);
 Freq = 0:Fs/Nfft:Fs/2;
 Freq = Freq * 1e-3; % Convert to kHz
 figure; plot(Freq, 10*log10(PSD), 'LineWidth', 1.5);
 grid on;
 xlabel('Frequency (KHz)');
 ylabel('PSD (dB/Hz)');
 switch windowing
 case 1
    title(['PSD with ', 'kaiser window']);
 case 2
   title(['PSD with ', 'rectangular  window']);
 case 3
   title(['PSD with ', 'hanning window']);
 case 4
   title(['PSD with ', 'hamming window']);
 otherwise
   error('Invalid windowing option. Use 1=Kaiser, 2=Rectangular, 3=Hanning, 4=Hamming.');
 end
end
```

Input Parameters:

1) Signal: The input signal you want to analyze
2) Fs: The sampling frequency of the signal
3) Windowing: The windowing number: 1 for kaiser, 2 for rectangular, 3 for hanning and 4 for hamming

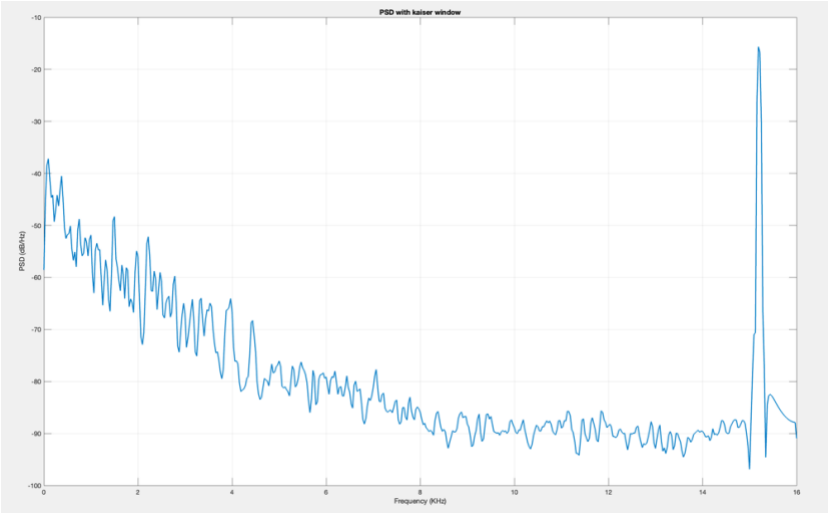
Parameters

a. Nfft (FFT size): Sets the FFT size to 1024 because it is power of 2, gives decent frequency detail without making the segments too long, which would reduce accuracy in time-varying signals and is common standard in audio signals.

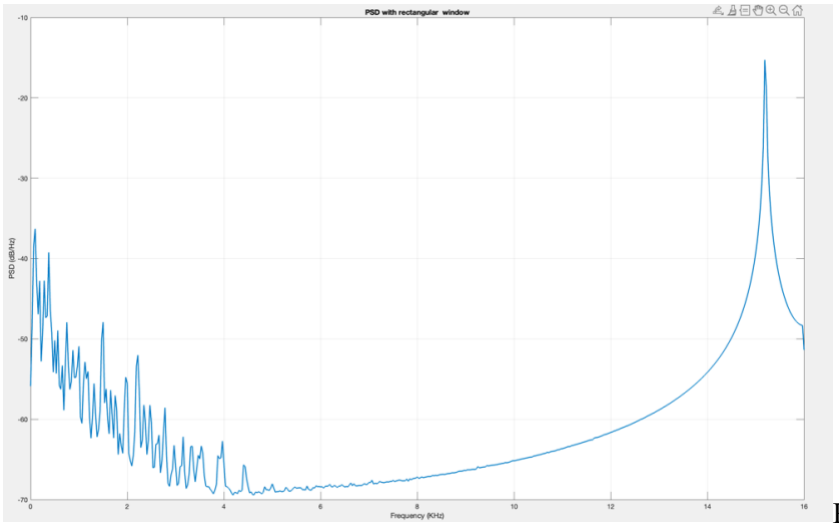
b. Window type:
    1. Kaiser Window with β=7
    2. Rectangular Window
    3. Hanning Window
    4. Hamming Window

c. percentage_overlap (Percentage overlap between the segments): Uses 50% overlap between segments.

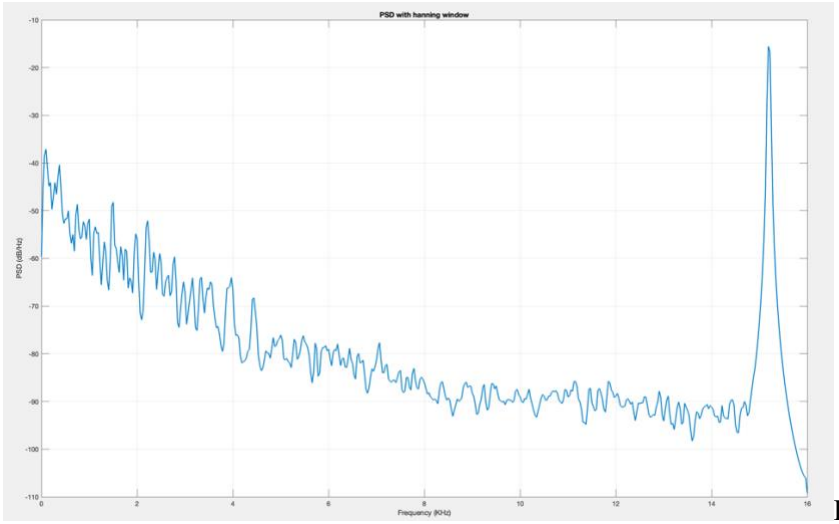
<u>The plot:</u>



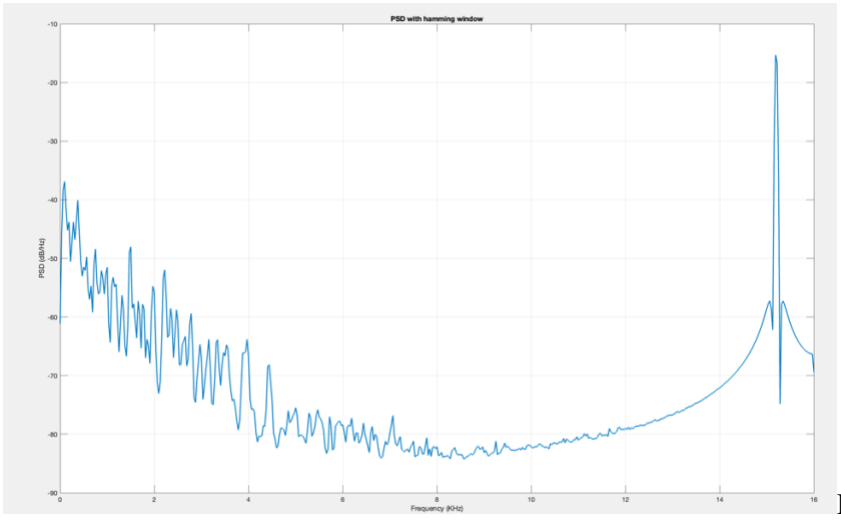
Kaiser Window



Rectangular Window



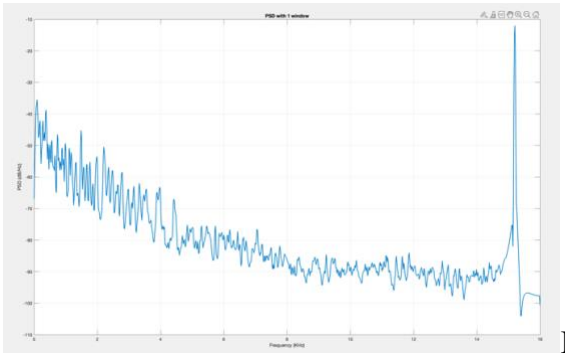
Hanning Window



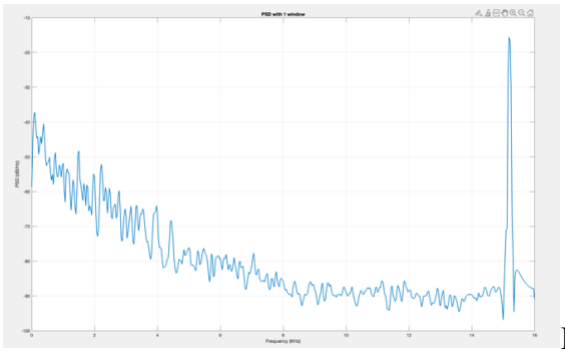
Hamming Window

7) The effect of parameters on the power spectrum

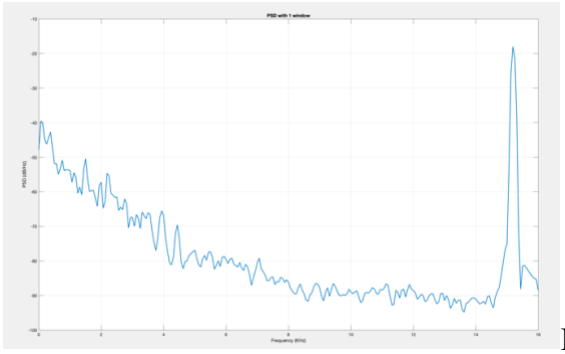
a) FFT size



FFT size 2^11. Kaiser Window



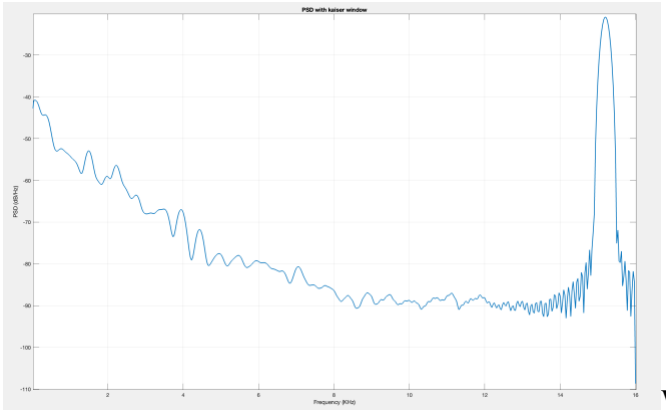
FFT size 2^10. Kaiser Window



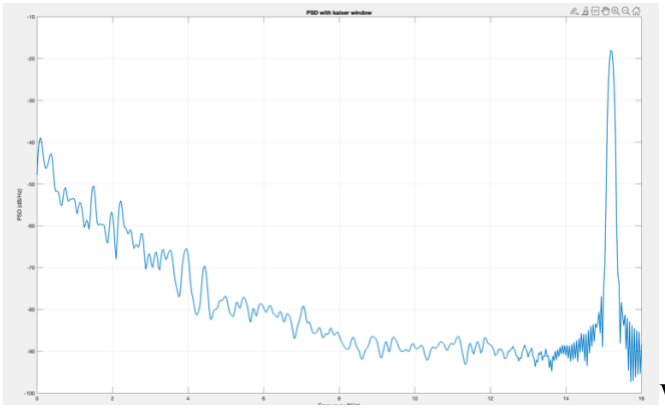
FFT size 2^9. Kaiser Window

d. The higher the FFT size the better frequency resolution and the narrower the peak which makes the spectrum noisier Additionally, when zero-padding is applied, it increases the FFT size by appending zeros to the original signal. This does not change the signal's frequency content or reduce noise, but it results in a denser sampling of the Discrete-Time Fourier Transform (DTFT), offering a smoother and more visually detailed spectrum without improving the actual frequency resolution or signal-to-noise ratio.when you use a smaller window size for the FFT, each segment of the signal that gets analyzed will have fewer data points. This leads to less smoothing and averaging across the signal segments which leads to more spectral leakage.

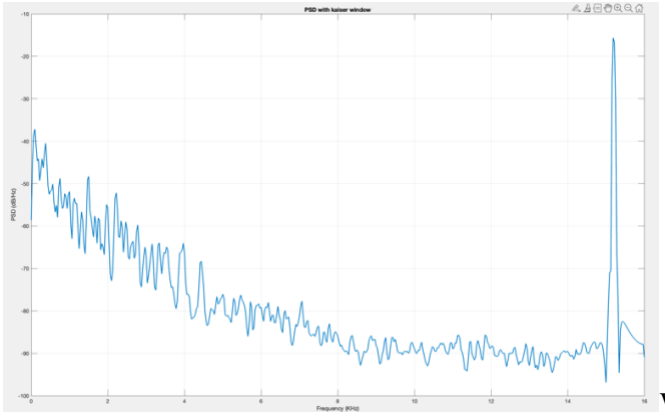
b) Window size



Window size 256.



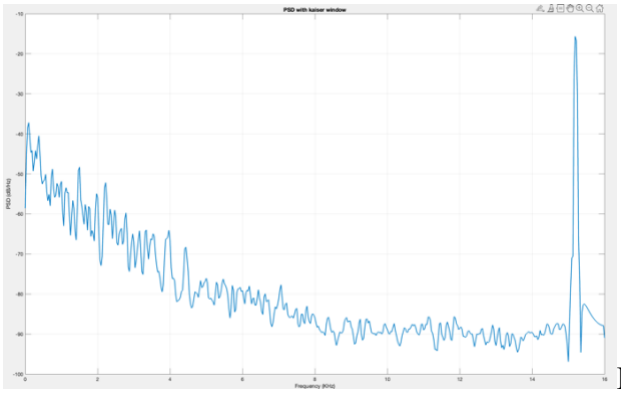
Window size 512.



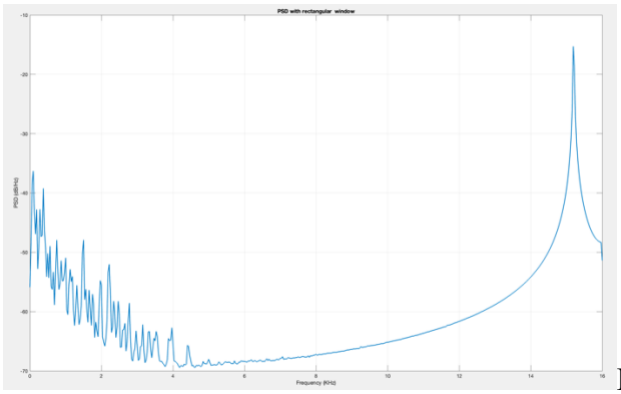
Window size 1024.

If the window size is smaller than FFT size, less averaging and more spectral leakage happen.

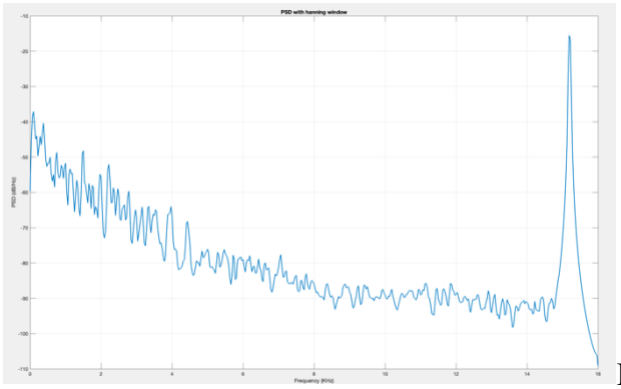
c) Window type



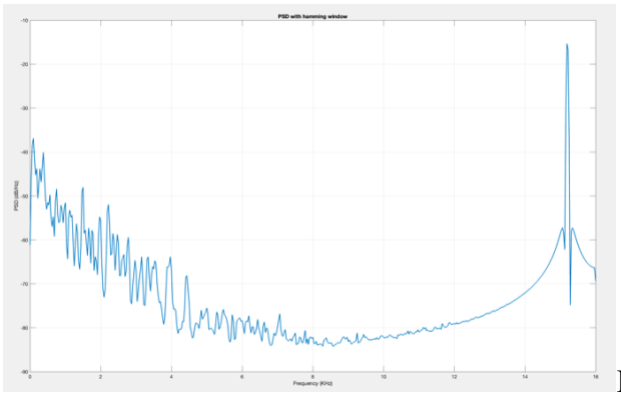
Kaiser window



Rectangular window



Hanning window

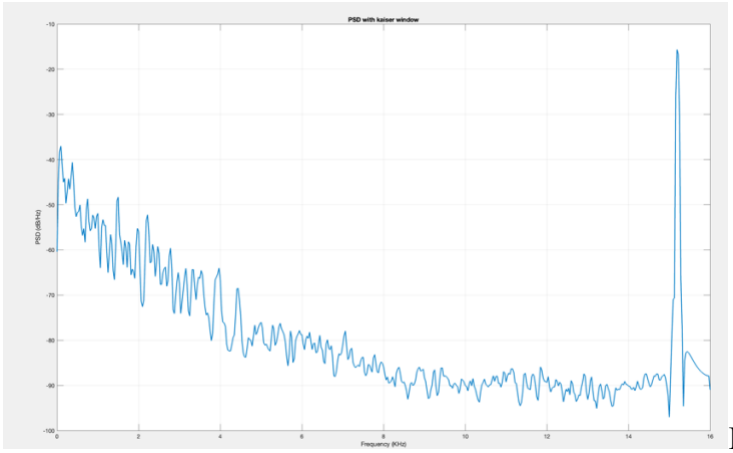


Hamming window

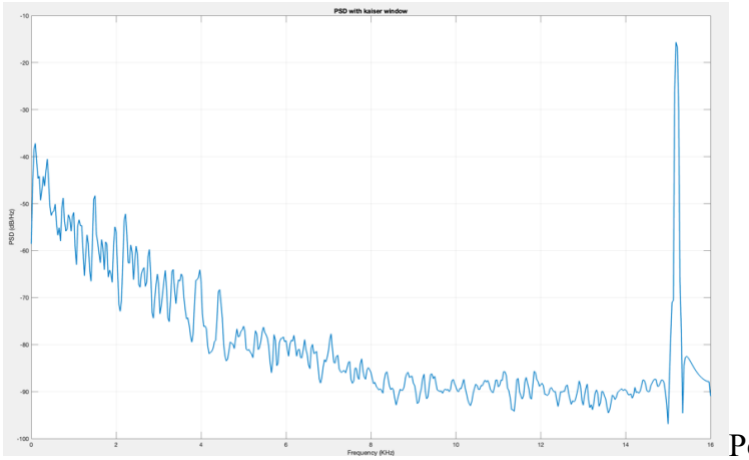
Each window impacts the mainlobe width and sidelobe level in a different ways:

- **Kaiser Window**
  The Kaiser window offers a good balance between resolution and leakage. In the power spectrum, the main peak is narrow and sharp, and the sidelobes are significantly suppressed.
- **Rectangular Window**
  The rectangular window has the narrowest mainlobe, meaning it offers the best theoretical resolution. However, it suffers from very high sidelobes, which results in significant spectral leakage. In the PSD plot, sharp main peak but also many high fluctuating sidelobes which is making it harder to distinguish weak frequency components from noise.
- **Hanning Window**
  The Hanning window provides a good mainlobe width and sidelobe suppression. Its mainlobe is wider than that of the rectangular window, which slightly reduces frequency resolution. However, sidelobes are lower, leading to reduced leakage. The PSD plot shows smoother transitions and better clarity in detecting nearby frequency components.
- **Hamming Window**
  The Hamming window has a narrow mainlobe similar to the Hanning window but higher sidelobes, which are visible in the PSD plot. While it suppresses the first sidelobe better, the following sidelobes are higher in magnitude, leading to more spectral leakage in certain frequency ranges compared to Hanning.

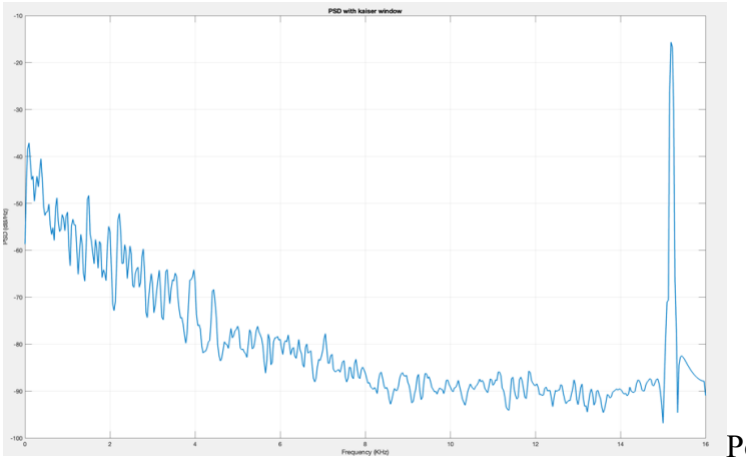
d) Percentage overlap



Percentage overlap 0.25



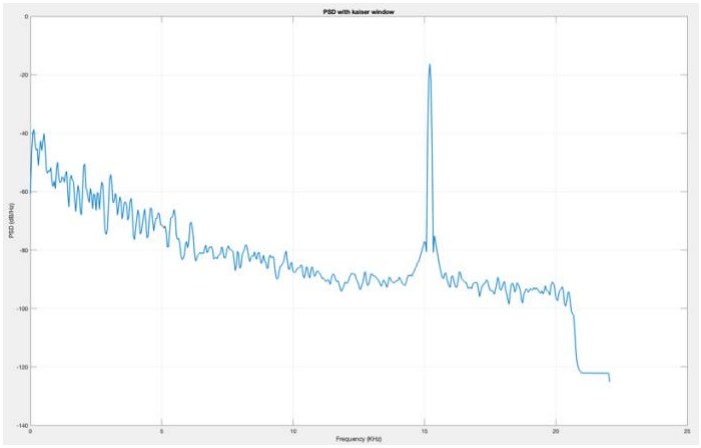
Percentage overlap 0.5



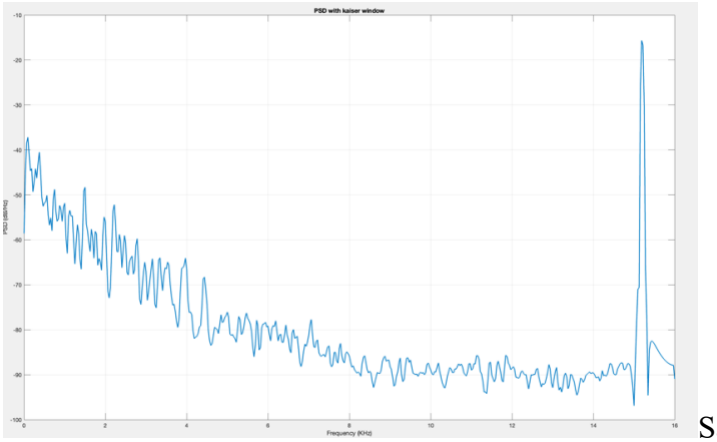
Percentage overlap 0.75

The more overlap the smoother PSD (better averaging), but slower computation.

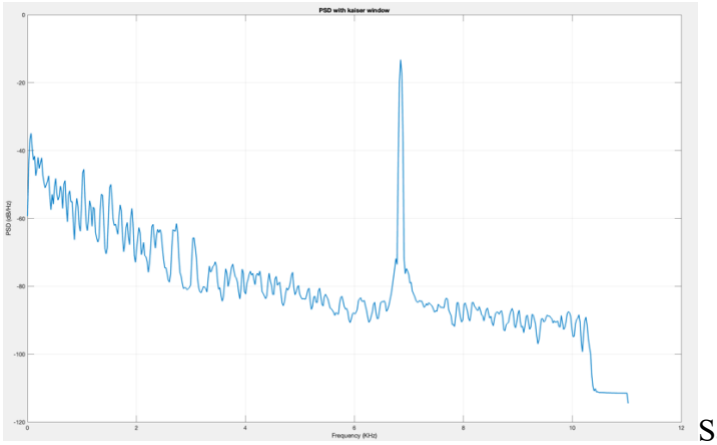
e) Sampling frequency



Sampling Frequency 44100



Sampling Frequency 32000



Sampling Frequency 22050

It sets the frequency range. If Fs is wrong, x-axis labels will be wrong and when the Fs increases the resolution decrease as shown.

Part 2: Digital filter design

1) **Define the digital filtering problem**

   The audio signal is corrupted by a 15.2 kHz sinusoidal interference of amplitude 1.8.

2) **Determine the constraint(s) that is(are) imposed on you in the digital filtering problem**

Parameter	*Constraint*
Computational Cost	1.6 – 6.4 million MACs/s
Latency	< 3.2 ms
Filter Length (N)	100 <= N <= 200
Stopband Attenuation	>= 50 dB
Passband Ripple	<= 0.4 dB

Constraint	Reason
Filter Length (100–200)	Balances memory usage and performance; suitable for real-time use.
Latency < 3.2 ms	Prevents audible delay; important for real-time audio playback: Latency=200/(32000*2)=3.125 ms
MACs/s: 1.6–6.4M	Keeps computation within practical range for embedded DSP systems.
Stopband Attenuation >= 50 dB	Ensures effective suppression of 15.2 kHz interference.
Passband Ripple <= 0.4 dB	Maintains audio fidelity and prevents perceptible distortion.

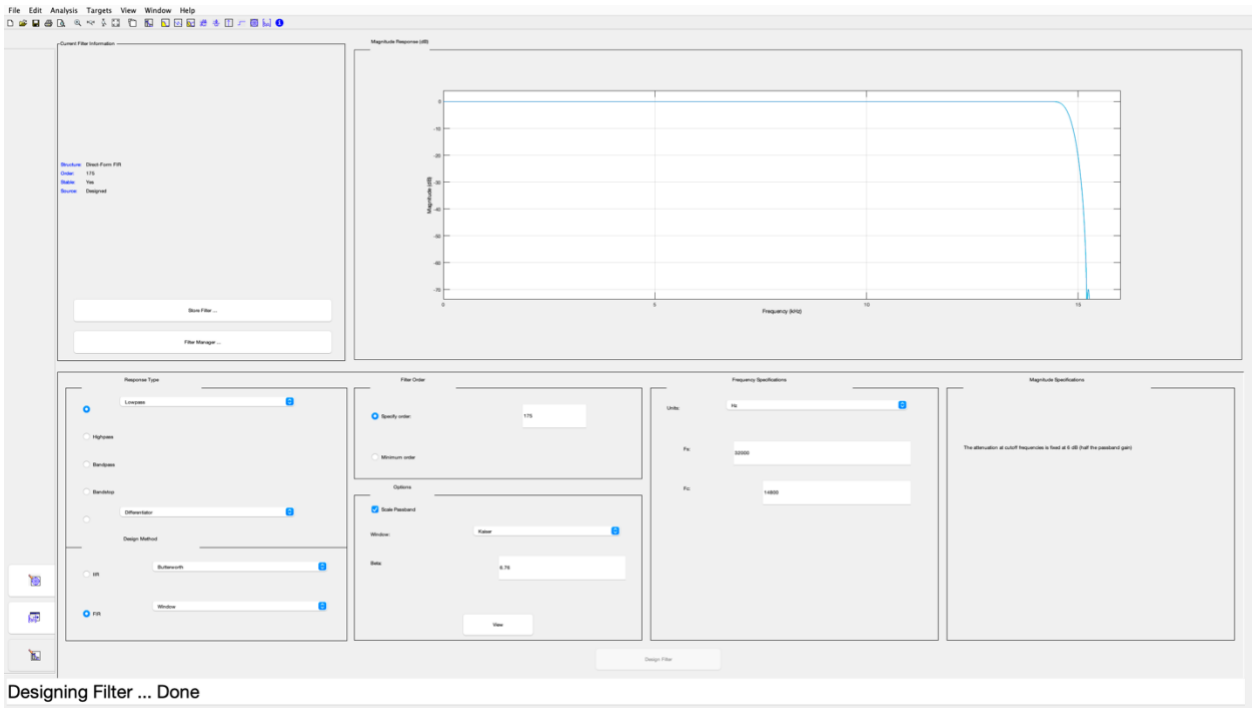
3) **Determine the specifications of the digital filter you plan to design**

Specification	*Value*
Sampling Frequency (Fs)	32 kHz
Filter Type	FIR Filters
Transition Region	14.6 kHz – 15.2 kHz
Stopband Attenuation	>= 60 dB
Passband Ripple	<= 0.1 dB
Transition Bandwidth	About 600 Hz

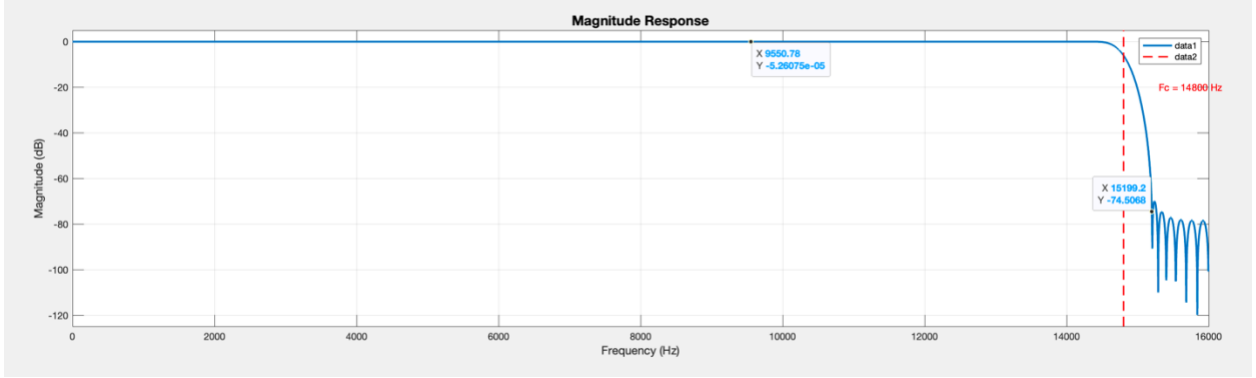
Specification	Reason
Sampling Frequency = 32 kHz	Matches the actual input signal's sample rate.
Transition region = 14.6–15.2 kHz	Targets the known interference range.
Stopband Attenuation >= 60 dB	Provides stronger suppression to make interference inaudible.
Passband Ripple <= 0.1 dB	More strict to preserve signal quality.
Transition Width about 600 Hz	Narrow transition avoids affecting useful neighboring frequencies.
FIR Linear Phase Filter	Prevents phase distortion, which is especially important in audio.

4) Using the filter designer tool in MATLAB, propose multiple solutions

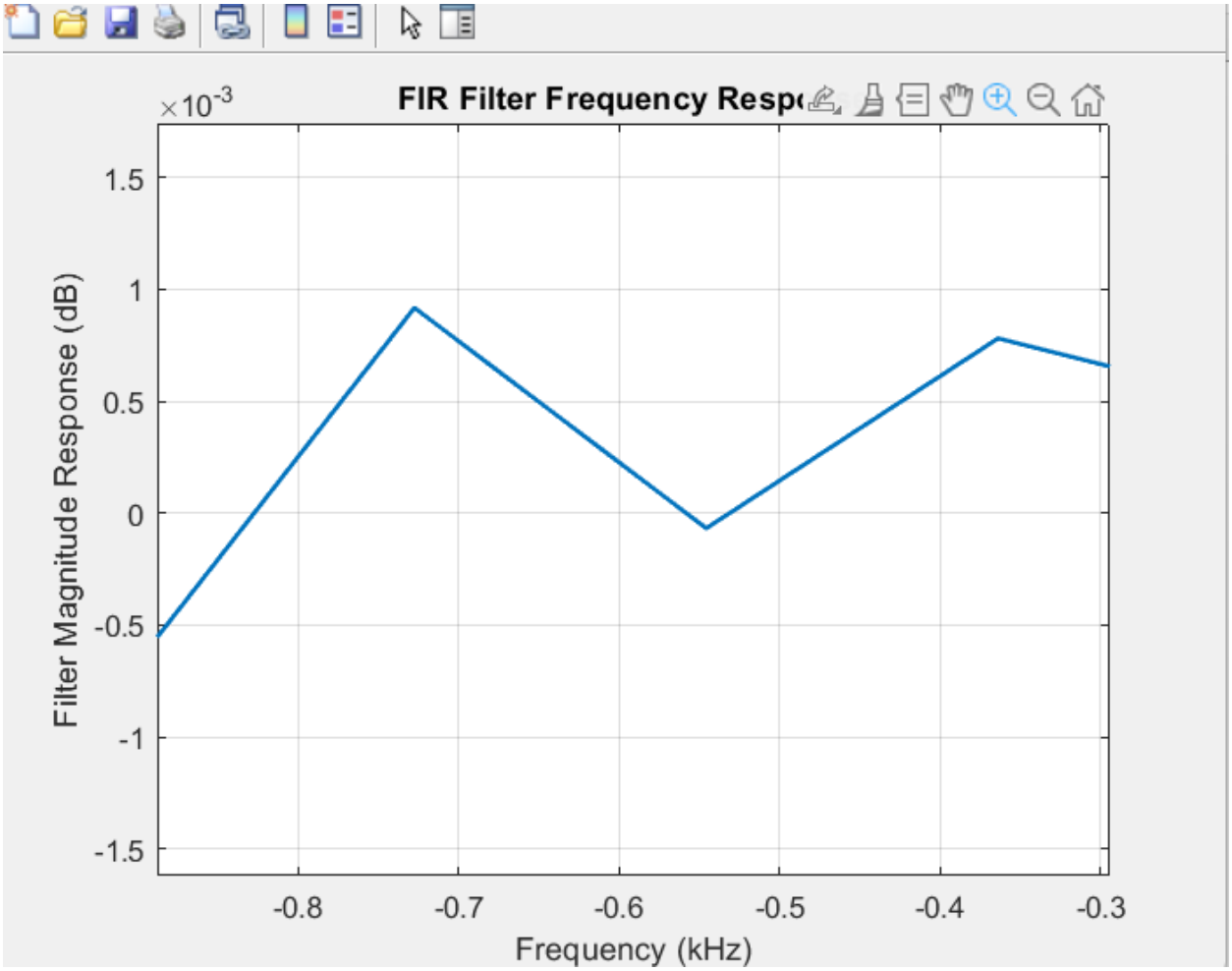
1. Kaiser window with beta 6.76 and order 175



Kaiser Window FIR Lowpass Filter (Order = 175, Beta = 6.76, Fc = 14.8 kHz)



I managed to have a stopband attenuation of 75 dB which is so good and had passband ripple less than 0.001 dB and latency equal 2.73 ms and MACs/s = 5.63 million.

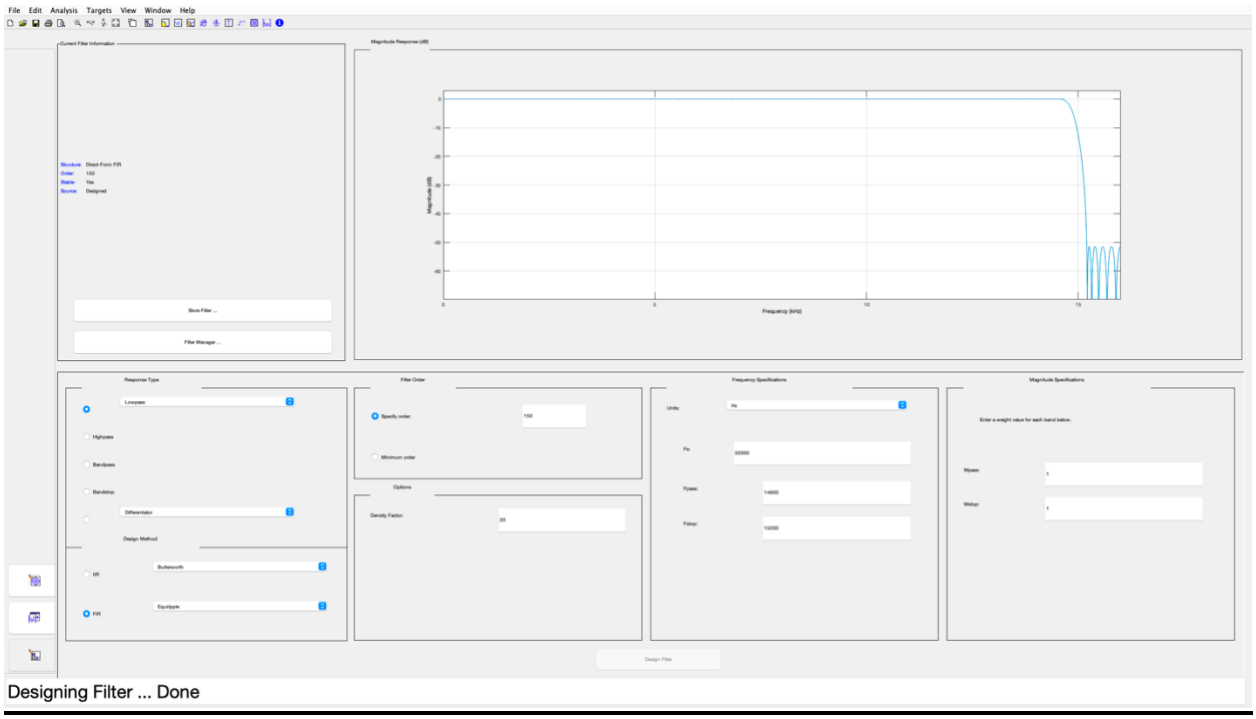


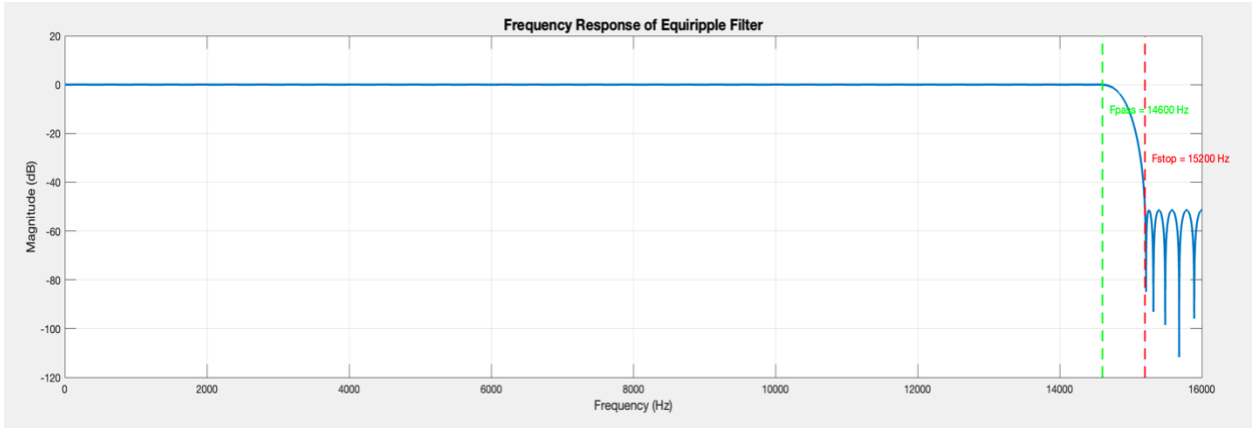

```
function Hd = kaiser_window
%KAISER_WINDOW Returns a discrete-time filter object.

% MATLAB Code
% Generated by MATLAB(R) 24.2 and Signal Processing Toolbox 24.2
% Generated on: 10-May-2025 14:12:15

% FIR Window Lowpass filter designed using the FIR1 function.

% All frequency values are in Hz.
Fs = 32000;  % Sampling Frequency

N    = 175;      % Order
Fc   = 14800;    % Cutoff Frequency
flag = 'scale';  % Sampling Flag
Beta = 6.76;     % Window Parameter

% Create the window vector for the design algorithm.
win = kaiser(N+1, Beta);

% Calculate the coefficients using the FIR1 function.
b  = fir1(N, Fc/(Fs/2), 'low', win, flag);
Hd = dfilt.dffir(b);

% [EOF]
```

Code of the filter

2. Equiripple FIR filter

Equiripple FIR Lowpass Filter (Order = 150, Fp = 14.6 kHz, Fstop = 15.2 kHz)

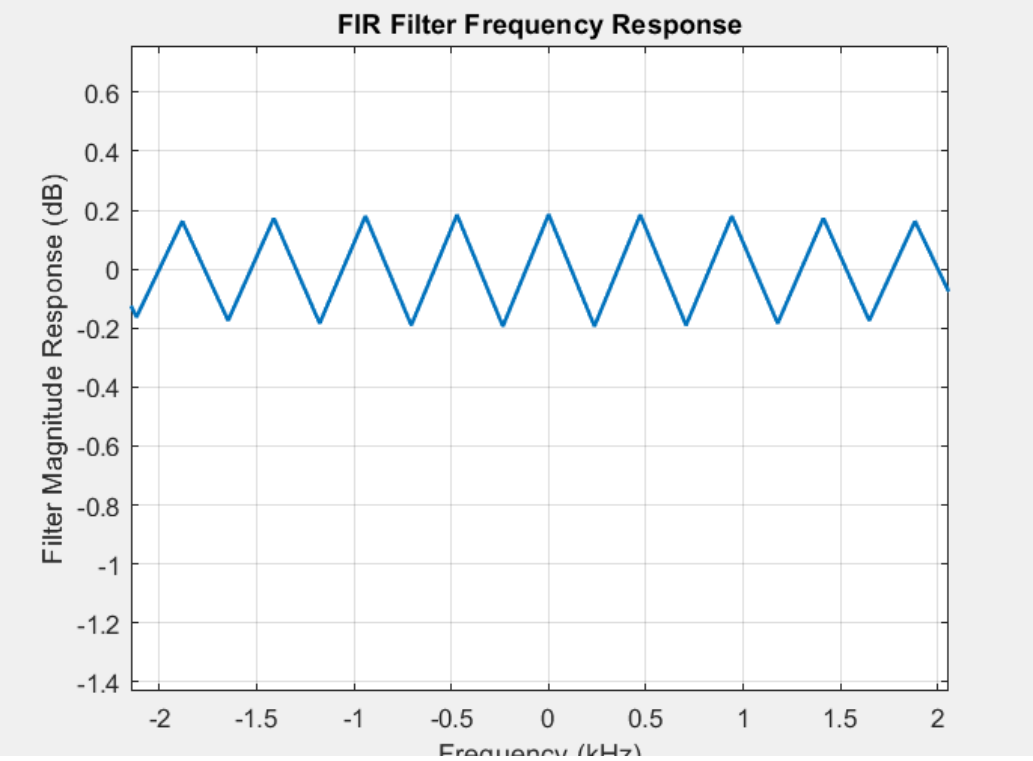




I managed to get attenuation of 51.57 dB and passband ripple less than 0.2 dB and latency equal 2.73 ms and MACs/s = 5.63 million.

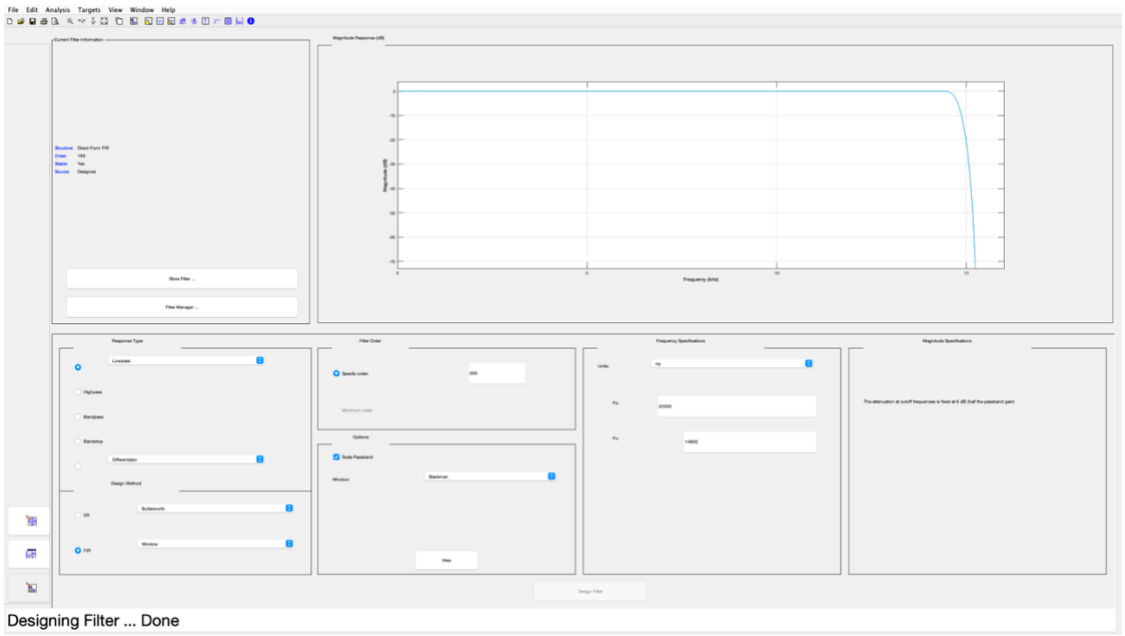
```
function Hd = equiripple
%EQUIRIPPLE Returns a discrete-time filter object.

% MATLAB Code
% Generated by MATLAB(R) 24.2 and Signal Processing Toolbox 24.2.
% Generated on: 10-May-2025 14:48:58

% Equiripple Lowpass filter designed using the FIRPM function.

% All frequency values are in Hz.
Fs = 32000;  % Sampling Frequency

N     = 150;    % Order
Fpass = 14600;  % Passband Frequency
Fstop = 15200;  % Stopband Frequency
Wpass = 1;      % Passband Weight
Wstop = 1;      % Stopband Weight
dens  = 20;     % Density Factor

% Calculate the coefficients using the FIRPM function.
b  = firpm(N, [0 Fpass Fstop Fs/2]/(Fs/2), [1 1 0 0], [Wpass Wstop], ...
           {dens});
Hd = dfilt.dffir(b);

% [EOF]
```

Code of the filter



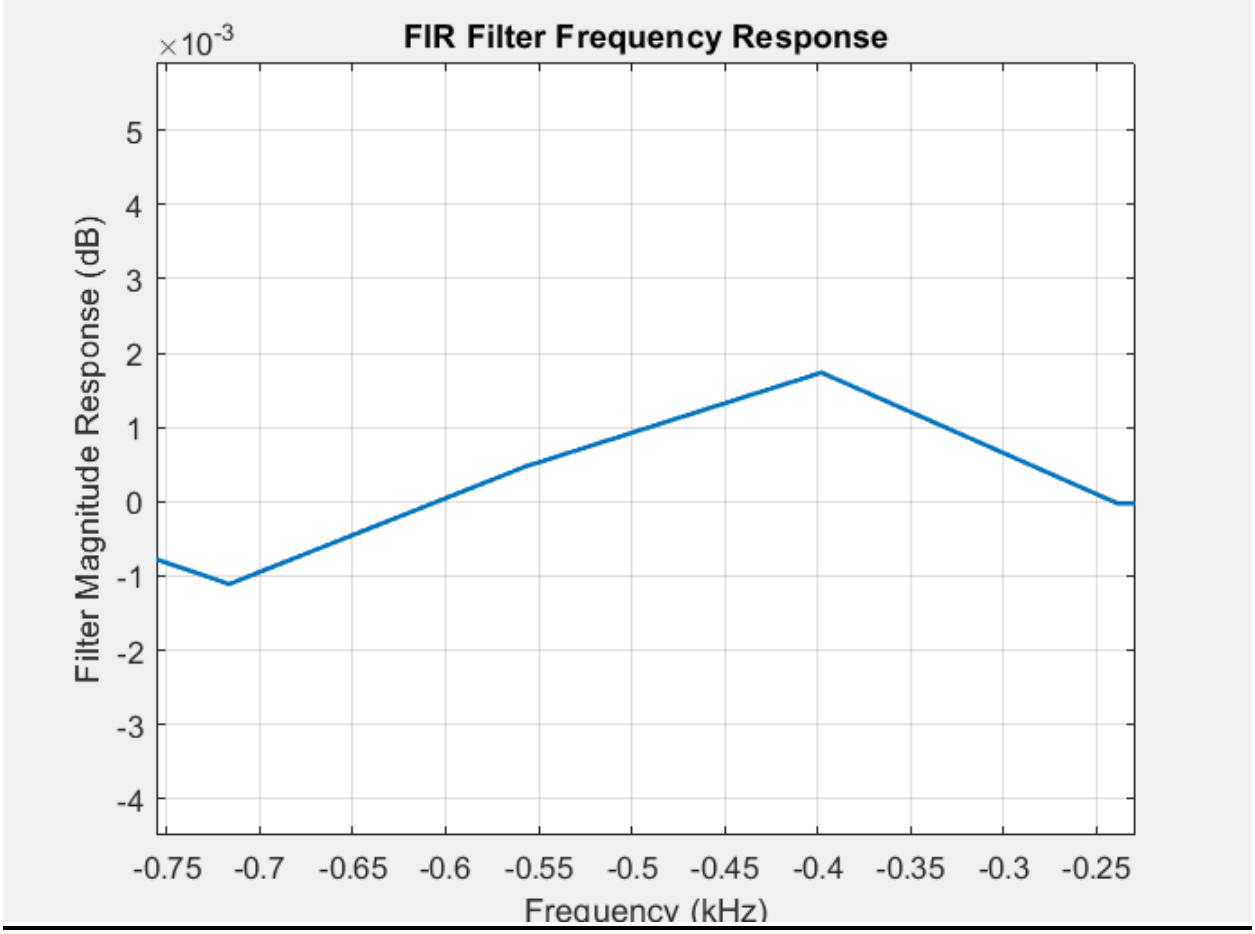
3. Blackman Window Design

Blackman window FIR Lowpass Filter (Order = 200, Fc = 14.8 kHz)



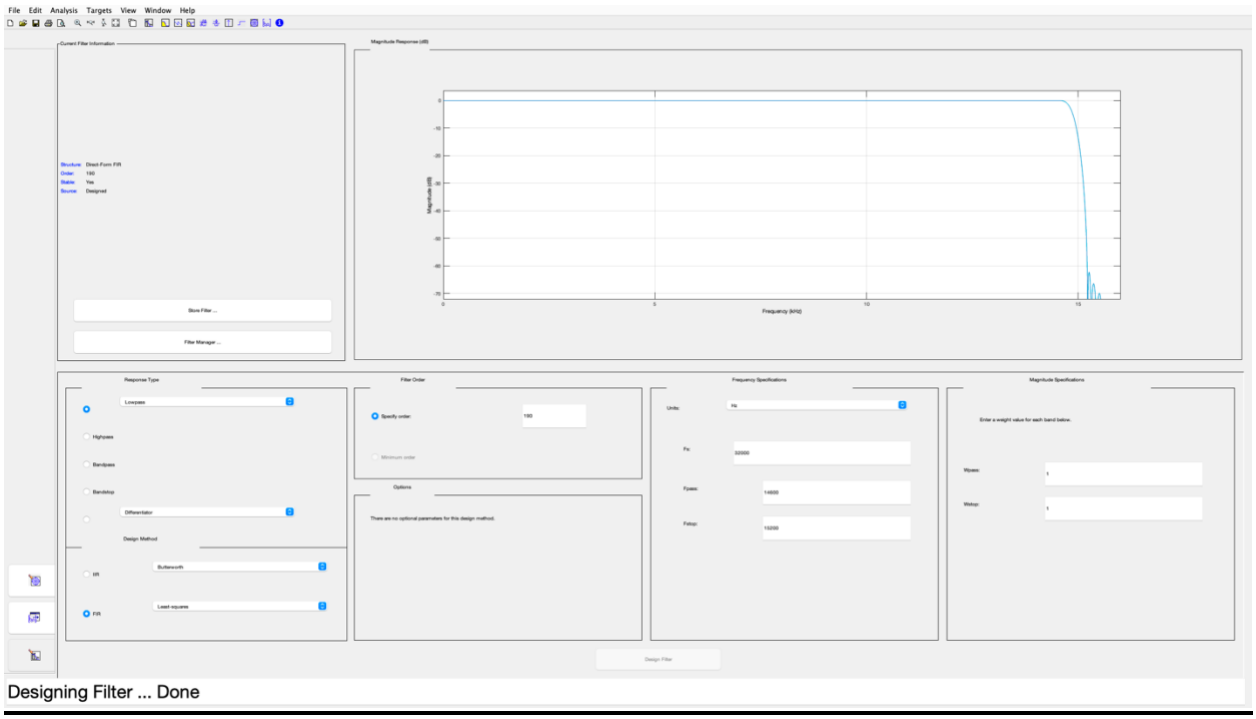
It has latency of 3.125 which is below 3.2 and its MACs/s equals to 6.43 million and a passband ripple less than 0.002 dB.

```
function Hd = blackman
%BLACKMAN Returns a discrete-time filter object.

% MATLAB Code
% Generated by MATLAB(R) 24.2 and Signal Processing Toolbox 24.2.
% Generated on: 10-May-2025 15:06:27

% FIR Window Lowpass filter designed using the FIR1 function.

% All frequency values are in Hz.
Fs = 32000;  % Sampling Frequency

N    = 200;      % Order
Fc   = 14800;    % Cutoff Frequency
flag = 'scale';  % Sampling Flag

% Create the window vector for the design algorithm.
win = blackman(N+1);

% Calculate the coefficients using the FIR1 function.
b  = fir1(N, Fc/(Fs/2), 'low', win, flag);
Hd = dfilt.dffir(b);

% [EOF]
```


FIR Filter Frequency Response
×10-3
5
4
3
2
1
0
-1
-2
-3
-4
Filter Magnitude Response (dB)
-0.75
-0.7
-0.65
-0.6
-0.55
-0.5
-0.45
-0.4
-0.35
-0.3
-0.25
Frequency (kHz)

4. Least Squares Design

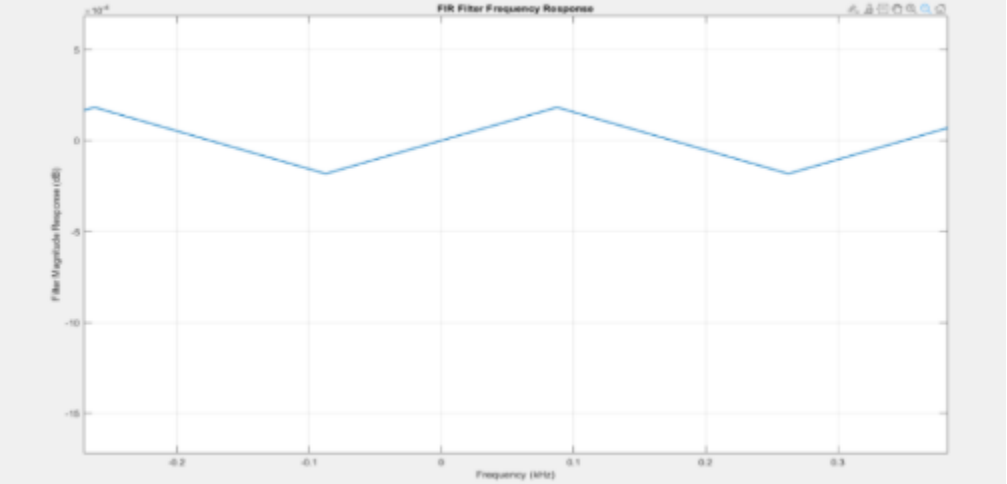
Least square FIR Lowpass Filter (Order = 190, Fp = 14.6 kHz, Fstop = 15.2 kHz)



It has latency of 2.97 which is below 3.2 and its MACs/s equals to 6.11 million and passband ripple less than 2*10^-4 dB

```
function Hd = leastsquares
%LEASTSQUARES Returns a discrete-time filter object.

% MATLAB Code
% Generated by MATLAB(R) 24.2 and Signal Processing Toolbox 24.2.
% Generated on: 10-May-2025 15:22:19

% FIR least-squares Lowpass filter designed using the FIRLS function.

% All frequency values are in Hz.
Fs = 32000;  % Sampling Frequency

N     = 190;    % Order
Fpass = 14600;  % Passband Frequency
Fstop = 15200;  % Stopband Frequency
Wpass = 1;      % Passband Weight
Wstop = 1;      % Stopband Weight

% Calculate the coefficients using the FIRLS function.
b  = firls(N, [0 Fpass Fstop Fs/2]/(Fs/2), [1 1 0 0], [Wpass Wstop]);
Hd = dfilt.dffir(b);

% [EOF]
```

The code for the least square fir filter.



5) **<u>Use decision analysis to compare the different digital filters and narrow them down to one or two solutions based on maximum utility.</u>**

Criterion	***Weight***	***Kaiser***	***Equiripple***	***Blackman***	***Least squares***
Stopband Attenuation (dB)	25%	75	51.57		
Passband Ripple (dB)	20%	<0.001	<0.2		
Latency (ms)	15%	2.73	2.73		
MACs/s (million)	15%	5.63	5.63		
Filter order	15%	175	150		
Transition band	10%	600 Hz	600		
Result	100%				

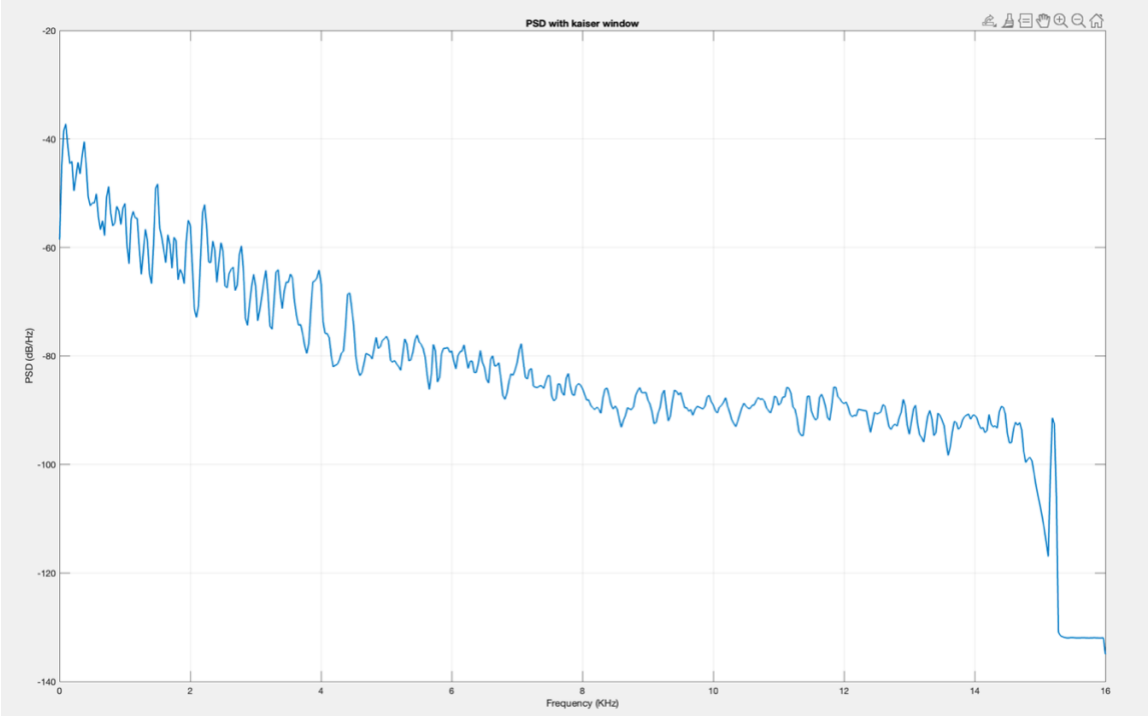
6) **Filter the audio signal + interference**

```
function Hd = kaiser_window
    % FIR Window Lowpass filter using Kaiser window
    Fs = 32000;         % Sampling Frequency
    N = 175;            % Filter order
    Fc = 14800;         % Cutoff Frequency
    Beta = 6.76;        % Kaiser window parameter
    flag = 'scale';     % Normalize gain

    win = kaiser(N+1, Beta);  % Kaiser window
    b = fir1(N, Fc/(Fs/2), 'low', win, flag);  % FIR filter design
    Hd = dfilt.dffir(b);  % Create filter object
end


Hd = kaiser_window();
filtered_audio = filter(Hd, audio_with_interf);
```

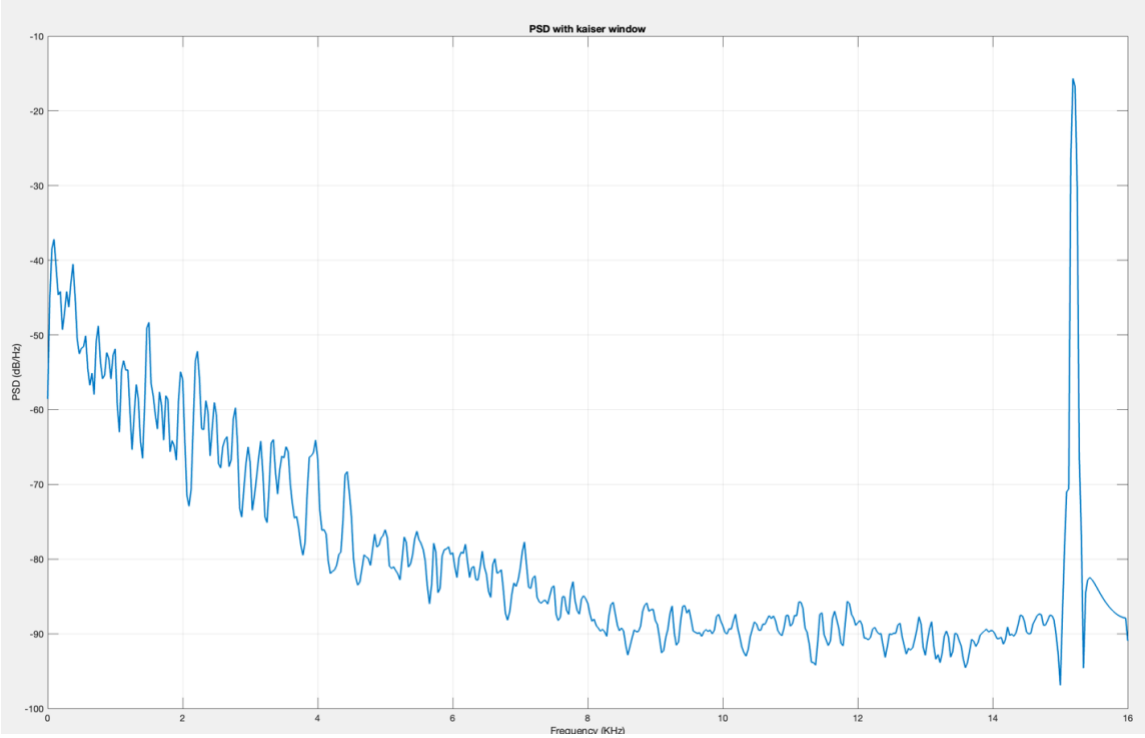
The code here add the signal to the kaiser window fir filter using filter function.

7) **Plot the frequency spectrum of the audio signal after filtering and listen to the filtered audio signal**

```
sound(filtered_audio, Fs);
pause(4);

plot_spectrum(audio_with_interf(:,1), Fs, 1);
plot_spectrum(filtered_audio(:,1), Fs, 1);
```



The spectrum after the filteration.



The spectrum before the filteration.

As we see there is a huge difference between the spectrum before and after the filteration at 15.2 kHz.