PREPARED BY YARA MAHMOUD 202300177 APRIL , 2025

# DIGITAL SIGNALS PROCCESSING PROJECT

DR. MAHMOUD ABDELAZIZ CIE 247 S-YARA.ABDALRAZEK@ZEWAILCITY.EDU.EG

## Part 1 : Spectral estimation

we aim to analyze the **frequency content** of an audio signal contaminated with high-frequency sinusoidal interference using the **Welch method for Power Spectral Density (PSD) estimation**. The effects of different DSP parameters are studied, including:

### 1. Reading the Audio File

The first step involved reading the audio file and storing its sampling frequency. The MATLAB code used for this task is shown below:

```
[Y, fs] = audioread('music_test_fayrouz.mp3');
```

The sampling frequency fs was stored for later use in the analysis.

### 2. Capturing a 3-Second Segment

A 3-second segment from the middle of the audio file was extracted and stored in the vector X:

Extraction Steps

1. **Find Middle Point**:
   - Calculate the middle sample position using floor division of the total length by 2.
2. **Determine 3-Second Window**:
   - Calculate how many samples correspond to 3 seconds: N = Fs × 3 (96,000 samples at 32 kHz).
3. **Create Symmetrical Window**:
   - Calculate start and end indices centered around the middle point:
     - Start index = Middle sample - half of N
     - End index = Start index + N - 1
4. **Range Validation**:
   - Ensure the calculated indices don't exceed the audio file boundaries.

Implementation Example

The document shows a specific implementation that actually extracts 100,001 samples (from 300,000 to 400,000) which at 32 kHz gives approximately 3.125 seconds rather than exactly 3 seconds. This suggests either:

- A rounding choice was made
- The example is slightly different from the general method described
- There may be a small error in the sample count (as 3 seconds at 32 kHz should be exactly 96,000 samples)

```
X = Y(3e5:4e5, 1);  % Extract approx. 3 seconds
```

This segment was chosen to ensure a representative portion of the audio signal was analyzed.

## 3. Plotting the Audio Signal

The extracted audio segment was plotted against time in seconds:

Creating a sample index (n) ranges from 0 till N-1, calculating the sampling frequency $T_s = \frac{1}{f_s}$, generate the time vector where $t = n * T_s$ also ranges from 0 to N-1, finally, plot the time vs amplitude.

```
% Time vector
n = 0:length(X)-1;
Ts = 1/fs;
t = n * Ts;

% Plot the audio segment
figure;
plot(t, X);
xlabel('Time (secs)');
ylabel('Amplitude');
title('Audio Signal (Approx. 3 seconds from middle)');
```

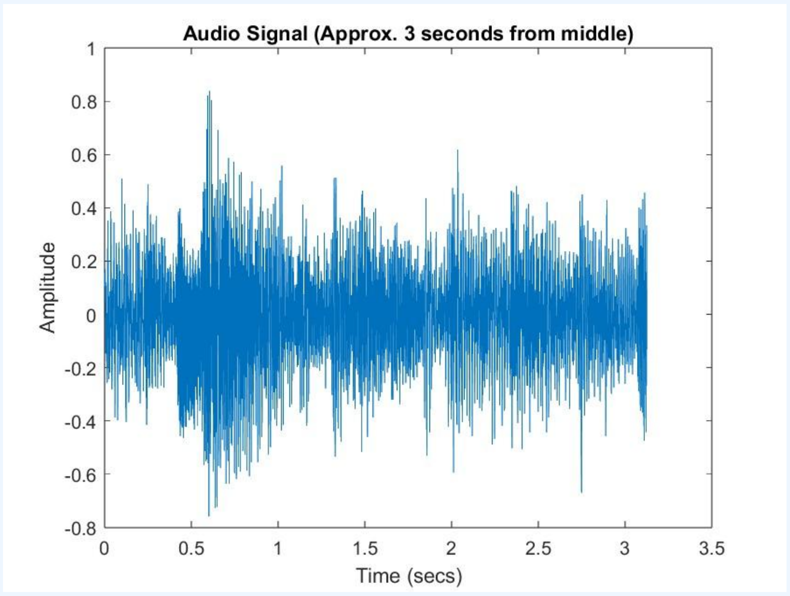


**Figure 1:** Time-domain plot of the 3-second audio segment.

## 4. Generating Sinusoidal Interference

A sinusoidal interference tone at 15.2 kHz with an amplitude of 1.8 was generated and added to the audio signal:

```
% Add sinusoidal interference
f2 = 15200;
A = 1.8;
in = A * cos(2 * pi * f2 * t).';
X_noisy = X + in;
```

The interference tone was designed to be clearly audible and distinguishable in the noisy signal.

5. Listening to the Audio Signals

The original and noisy audio signals were played back for comparison:

```
% Play original and noisy audio
disp('Playing original audio...');
sound(X, fs);
pause(length(X)/fs + 1);

disp('Playing audio with interference...');
```

```
sound(X_noisy, fs);
pause(length(X_noisy)/fs + 1);
```

The interference tone was clearly audible in the noisy signal, confirming its addition.

6. Welch Power Spectrum Estimation

The Welch power spectrum of the noisy signal was computed using the pwelch function. The following parameters were specified:

- **FFT Size (**N**)**: 1024 (default), 256, and 4096 for comparison.
- **Window Size**: Equal to the FFT size.
- **Window Type**: Rectangular, Hamming, Hann, Blackman, and Flattop.
- **Overlap Percentage**: 0%, 50%, and 75%.
- **Sampling Frequency (**fs**)**: As read from the audio file.

The plot_welch_spectrum function was used to generate the plots:

```
function plot_welch_spectrum(signal, fs, window_type,perc,N)

    noverlap = N *perc;   %  overlap %
    nfft = N;           % FFT length

    % Select window
    switch lower(window_type)
        case 'rectangular'
            win = rectwin(N);
        case 'hamming'
            win = hamming(N);
        case 'hann'
            win = hann(N);
        case 'blackman'
            win = blackman(N);
        case 'flattop'
            win = flattopwin(N);
        case 'triangular'
            win = triang(N);
        otherwise
            error('Unknown window type. Choose: rectangular, hamming, hann, blackman, flattop, 
triangular.');
    end

    % Plot Welch power spectrum
    PSD=pwelch(signal, win, noverlap, nfft, fs);
    freq = 0:fs/nfft:fs/2;
    freq = freq/1e3;

    plot(freq, 10*log10(PSD), 'LineWidth', 1.5);
    title(['Power Spectrum using ', window_type, ' window']);
    xlabel('Frequency (Hz)');
    ylabel('Power/Frequency (dB/Hz)');
    grid on;
end
```

- FFT size (frequency resolution)

```
N = 1024;           % FFT size
N1=256;
N2=4096;
perc = 0.5;         % 50% overlap

%%  7 – Analyze Effects of  Parameters
```

```
signal = X_noisy;

%% A. Varying FFT Size (Frequency Resolution)
figure('Name', 'Effect of FFT Size');
subplot(3,1,1);
plot_welch_spectrum(signal, fs, 'hamming', perc, N1);
title('FFT Size = 256');

subplot(3,1,2);
plot_welch_spectrum(signal, fs, 'hamming', perc, N);
title('FFT Size = 1024');

subplot(3,1,3);
plot_welch_spectrum(signal, fs, 'hamming',perc, N2);
        title('FFT Size = 4096');
```

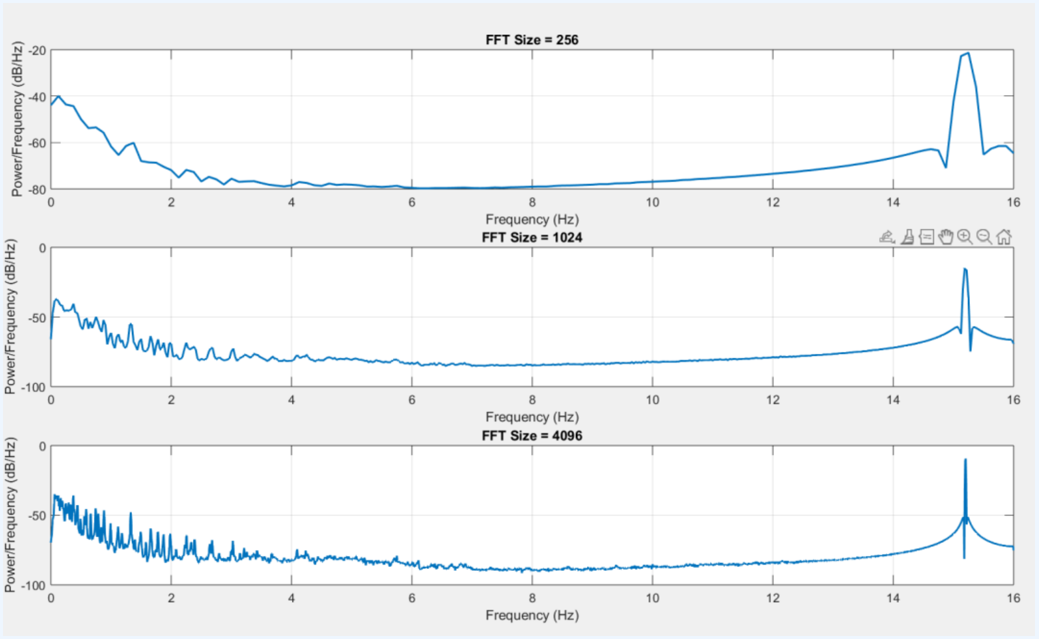


**Figure 2:** FFT size effect.

Larger FFTs provide better resolution but can make the spectrum noisier and increase processing time.

**Window:** Hamming, **Overlap:** 50%

| FFT Size | Description |
|---|---|
| **256** | The spectral peaks are **broad and less sharp**, meaning **low resolution**. You can **see the 15.2 kHz tone**, but it's blurred and might overlap with nearby content. Fastest computation. |
| **1024** | The peak is **sharper**, and nearby frequencies are more distinguishable. This is a good **balance** between resolution and smoothness. |
| **4096** | The peak at 15.2 kHz is **extremely narrow and sharp**, with high frequency resolution. However, the plot might show **more fluctuations (noise)** in the spectrum due to fewer averaged segments and higher variance. |

**Table 1:** FFT size effect.

FFT size controls **frequency bin width**:

$$\Delta f = \frac{f_s}{N}$$

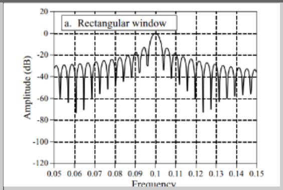
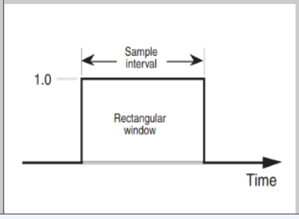
So, larger N gives **narrower bins**, enabling **finer frequency detail**.

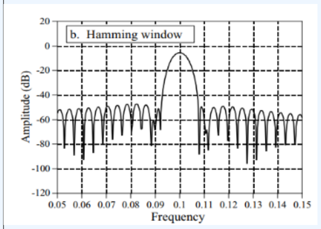
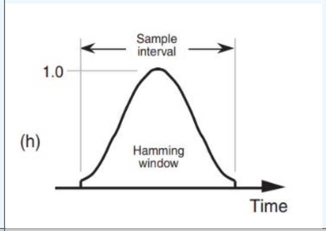
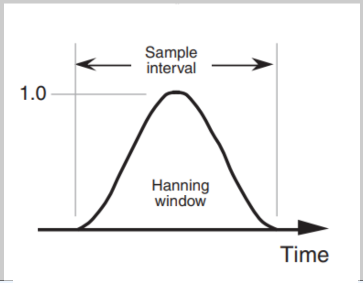
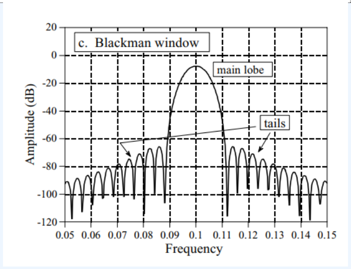
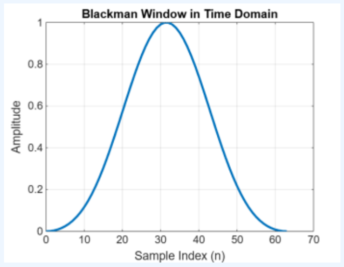
But larger FFTs reduce the **number of segments** that can fit in your signal → **higher variance** (less averaging).

- **Window type (spectral leakage)**

```
%% B. Varying Window Type (Leakage Suppression)
figure('Name', 'Effect of Window Type');
subplot(3,2,1);
plot_welch_spectrum(X_noisy, fs, 'rectangular', perc, N);
title('Rectangular');

subplot(3,2,2);
plot_welch_spectrum(X_noisy, fs, 'hamming', perc, N);
title('Hamming');

subplot(3,2,3);
plot_welch_spectrum(X_noisy, fs, 'hann', perc, N);
title('Hann');

subplot(3,2,4);
plot_welch_spectrum(X_noisy, fs, 'blackman', perc, N);
title('Blackman');

subplot(3,2,5);
plot_welch_spectrum(X_noisy, fs, 'flattop', perc, N);
title('Flattop');
```

Different window types affect the trade-off between main lobe width and side lobe suppression

| Window | Main Lobe Width | Side Lobe Level | Interpretation | |
|---|---|---|---|---|
| **Rectangular** | Narrow | Very high | Best resolution, worst leakage. Picks up false frequency content. |   |

| | | | | |
|---|---|---|---|---|
| **Hamming** | Wider | Lower | Good trade-off. Smooth spectrum. |   |
| **Hann** | similar to Hamming | Slightly lower | Slightly better leakage suppression. |  |
| **Blackman** | Even wider | Much lower | Better suppression but worse resolution. |   |

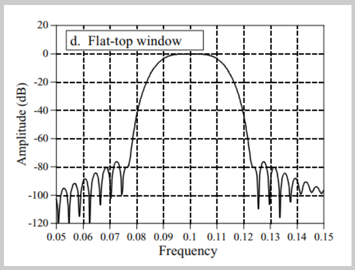
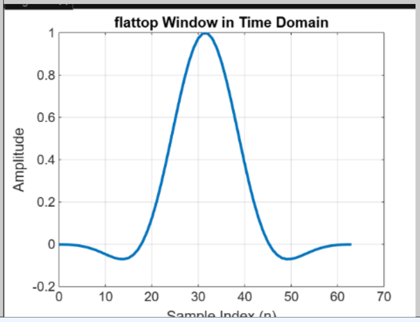
|  |  |  |  |  |
|---|---|---|---|---|
| **Flattop** | Very wide | Very low | Excellent amplitude accuracy, but smears closely spaced frequencies. |   |

**Table 2:** windows used in the function.

- **Rectangular** is not recommended due to leakage.
- **Hamming/Hann** are good default choices.
- **Flattop** is ideal for accurate peak measurement (amplitude), not resolution

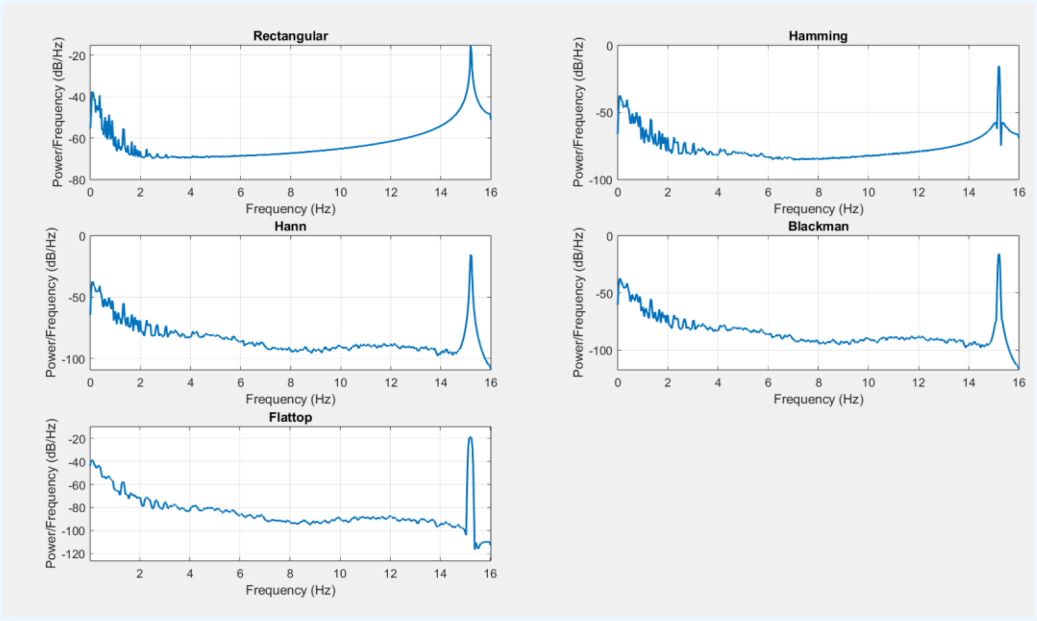


**Figure 3:** window type effect

| Window | Description |
|---|---|
| **Rectangular** | Shows many side lobes (ripples) around the 15.2 kHz tone and noise floor. This is called **spectral leakage**. |
| **Hamming** | Reduces side lobes. Main peak becomes slightly wider, but side leakage is suppressed. |
| **Hann** | Similar to Hamming. A bit better in side-lobe suppression. |

| | |
|---|---|
| **Blackman** | Very smooth plot. Side lobes are significantly reduced, but main peak is wider. |
| **Flattop** | Gives the **flattest and widest peak**, good for **accurate amplitude estimation**, not for frequency resolution. |

**Table 3: Window type** effect.

Windows **taper the edges** of each segment to reduce abrupt transitions → less leakage.

But tapering **spreads energy** → wider main lobes = lower frequency resolution.

**Trade-off**: You can’t have both perfect resolution and zero leakage.

- Overlap percentage (variance reduction)

```
%% C. Varying Overlap Percentage (Variance Reduction)
figure('Name', 'Effect of Overlap Percentage');
subplot(3,1,1);
plot_welch_spectrum(signal, fs, 'hann', 0.0, N);
title('Overlap = 0%');

subplot(3,1,2);
plot_welch_spectrum(signal, fs, 'hann', perc, N);
title('Overlap = 50%');

subplot(3,1,3);
plot_welch_spectrum(signal, fs, 'hann', 0.75, N);
title('Overlap = 75%');
```

| Overlap | Observation |
|---|---|
| **0%** | High variance (noisy spectrum), less averaging |
| **50%** | Balanced: reduced variance, smooth result |
| **75%** | Very smooth, lowest variance, but higher computation time |

**Table 4:** overlap % effect

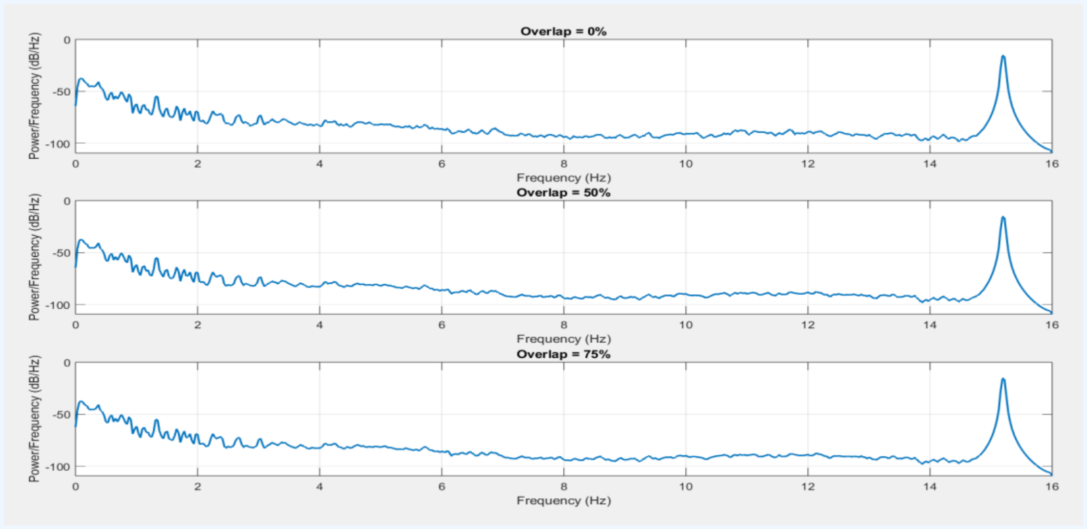


**Figure 4:** overlap % effect

Higher overlap reduces variance and results in a smoother PSD. 50% is a commonly used balance between speed and smoothness.

- Window size

```
%% D. Varying Window Size (≤ FFT Size)
figure('Name', 'Effect of Window Size');

window_sizes = [256, 512, 1024];  % all ≤ N2 = 1024
for i = 1:length(window_sizes)
    subplot(length(window_sizes), 1, i);
    plot_welch_spectrum(signal, fs, 'hamming', perc, window_sizes(i));
    title(['Window Size = ', num2str(window_sizes(i))]);
end
```

- Tests three window sizes (in samples):
    - 256 samples (small)
    - 512 samples (medium)
    - 1024 samples (large)
- Note that all are ≤ 1024 (likely N2 is the FFT size)

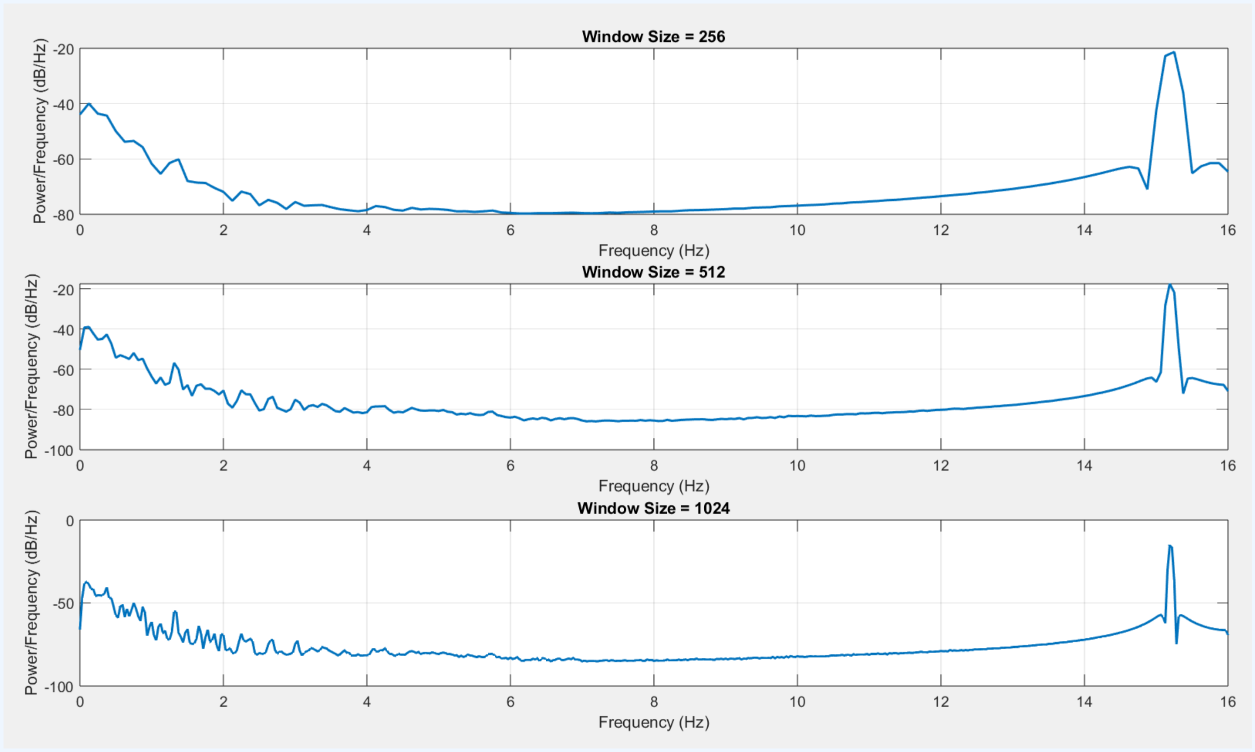


**Figure 5:** window size effect

| Window Size | Frequency Resolution | Noise Floor (dB/Hz) | Spectral Smoothing | Effect on Original Sample |
|---|---|---|---|---|
| **256** | Coarse | ~ -50 | Low | Captures broader spectral components with less smoothing. Higher noise floor may obscure weaker signals. |

| | | | | |
|---|---|---|---|---|
| **512** | Moderate | ~ -80 | Moderate | Provides a balance between resolution and noise reduction. Primary spectral peak is clearer, noise is moderately suppressed. |
| **1024** | Fine | ~ -100 | High | Excellent frequency resolution and noise suppression. Smaller fluctuations are smoothed out, revealing dominant spectral components more clearly. |

**Table 5:** window size effect

- Sampling frequency of the signal

```
%% E. Varying Sampling Frequency
fs_list = [fs, 22050, 8000,44000];
labels = {'Original fs', 'Downsampled to 22.05 kHz', 'Downsampled to 8 
kHz','upsampled to 44 kHz'};

figure('Name', 'Effect of Sampling Frequency');
for i = 1:length(fs_list)
    % Resample signal
    fs_new = fs_list(i);
    X_rs = resample(signal, fs_new, fs);
    subplot(length(fs_list), 1, i);
    plot_welch_spectrum(X_rs, fs_new, 'hann', perc, N);
    title(labels{i});
end
```

- fs_list: Contains three sampling frequencies to test :Original sampling rate (fs) , 22.05 kHz , 8 kHz.
- Loops through each target sampling frequency
- resample() function changes the sampling rate ,The resampling process includes **anti-aliasing filtering**

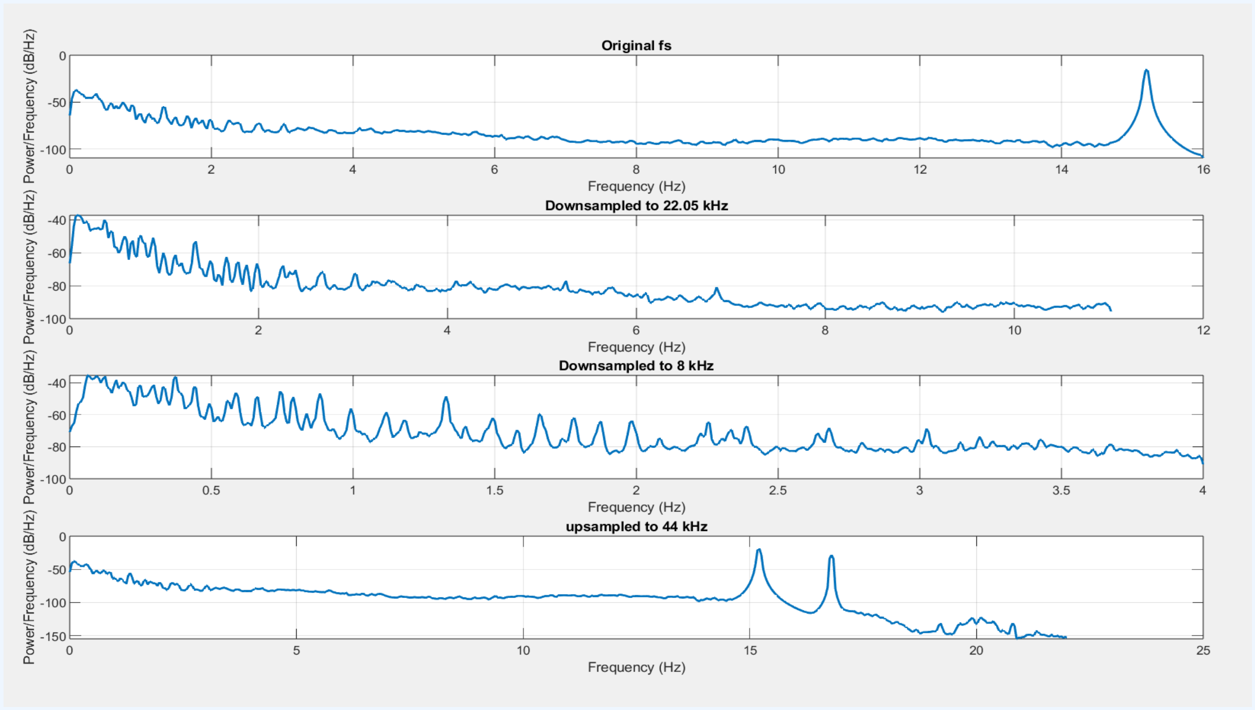


**Figure 6:** sampling frequency effect

| Sampling Rate | Frequency Range (Hz) | Noise Floor (dB/Hz) | Spectral Smoothing | Effect on Original Sample |
|---|---|---|---|---|
| **Original fs** | 0 - 16 | ~ -100 | Low | Retains full spectral detail, including the primary peak at ~15 Hz. |
| **22.05 kHz** | 0 - 12 | ~ -80 | Moderate | Aliasing begins to appear, with some spectral components becoming less distinct. |
| **8 kHz** | 0 - 4 | ~ -60 | High | Significant spectral compression. High-frequency components are lost, and low-frequency components become more prominent. |
| **44 kHz** | 0-22 | ~ -110 | Low | Higher spectral resolution, but no recovery of lost high-frequency content. Interpolated spectrum appears smoother, but aliasing and losses remain. |

**Table 6:** sampling frequency effect

**Summary:**

| Parameter | Effect on PSD | Figure Reference |
|---|---|---|
| **FFT Size** | Controls **frequency resolution**. Large size → finer bins, sharper peaks but higher variance. | Figure 2 |
| **Window Type** | Balances **leakage suppression vs. resolution**. Tapered windows (like Hamming, Hann) suppress leakage; rectangular gives highest resolution but worst leakage. | Figure 3 |
| **Overlap %** | More overlap → **less variance**, smoother PSD. Little/no overlap gives noisy, unreliable spectra. | Figure 4 |
| Window size | Larger windows provide better frequency resolution but less temporal precision. | Figure 5 |
| Sampling frequency | Determines the Nyquist frequency. Higher sampling rates provide a wider frequency range but increase computational cost. | Figure 6 |

**Table 7:** effect of different parameters on the signal

| Issue | Explanation | Solution |
|---|---|---|
| **Spectral Leakage** | Due to rectangular window or poor windowing in general | Use tapered windows (Hamming, Hann, Blackman) |
| **Poor Frequency Resolution** | With small FFT sizes (e.g. 256) | Use a larger FFT (1024 or 4096), depending on real-time constraints |

| | | |
|---|---|---|
| **Smearing of Peaks** | From flattop or blackman window | Use only if amplitude accuracy is needed, not for resolving close frequencies |
| **Computational Cost** | Large FFT + high overlap = slower processing | Tune for your application's real-time needs |

**Table 8:** possible issues.

| Goal | Suggested Setting |
|---|---|
| **General-purpose analysis** | Hamming + 50% Overlap + FFT 1024 |
| **High resolution (tone ID)** | Hann + 4096 FFT + 50% Overlap |
| **Accurate amplitude estimation** | Flattop + 1024 FFT + 50% Overlap |
| **Faster processing** | Hamming + 256 FFT + 25–50% Overlap |

**Table 9:** suggestions.

The Welch method effectively estimated the power spectrum of a noisy audio signal while minimizing variance and leakage. Through parameter tuning (window, FFT size, overlap), the spectrum can be shaped for different goals—whether resolution, smoothing, or amplitude accuracy.

The 15.2 kHz interference was clearly visible in all settings, but its clarity and representation varied significantly depending on the window and FFT settings.

## Appendix: MATLAB Code

```
function plot_welch_spectrum(signal, fs, window_type,perc,N)

    noverlap = N *perc;   %  overlap %
    nfft = N;            % FFT length

    % Select window
    switch lower(window_type)
        case 'rectangular'
            win = rectwin(N);
        case 'hamming'
            win = hamming(N);
        case 'hann'
            win = hann(N);
        case 'blackman'
            win = blackman(N);
        case 'flattop'
            win = flattopwin(N);
        case 'triangular'
            win = triang(N);
        otherwise
            error('Unknown window type. Choose: rectangular, hamming, hann, blackman, flattop, 
triangular.');
    end

    % Plot Welch power spectrum
    PSD=pwelch(signal, win, noverlap, nfft, fs);
    freq = 0:fs/nfft:fs/2;
    freq = freq/1e3;

    plot(freq, 10*log10(PSD), 'LineWidth', 1.5);
```

```matlab
    title(['Power Spectrum using ', window_type, ' window']);
    xlabel('Frequency (Hz)');
    ylabel('Power/Frequency (dB/Hz)');
    grid on;
end

%% Main Script

% Read audio
[Y, fs] = audioread('music_test_fayrouz.mp3');
X = Y(3e5:4e5, 1);  % Extract approx. 3 seconds

% Time vector
n = 0:length(X)-1;
Ts = 1/fs;
t = n * Ts;

% Plot the audio segment
figure;
plot(t, X);
xlabel('Time (secs)');
ylabel('Amplitude');
title('Audio Signal (Approx. 3 seconds from middle)');

% Add sinusoidal interference
f2 = 15200;
A = 1.8;
in = A * cos(2 * pi * f2 * t).';
X_noisy = X + in;

% Play original and noisy audio
disp('Playing original audio...');
sound(X, fs);
pause(length(X)/fs + 1);

disp('Playing audio with interference...');
sound(X_noisy, fs);
pause(length(X_noisy)/fs + 1);



N = 1024;           % FFT size
N1=256;
N2=4096;
perc = 0.5;         % 50% overlap

%%  7 – Analyze Effects of  Parameters

signal = X_noisy;

%% A. Varying FFT Size (Frequency Resolution)
figure('Name', 'Effect of FFT Size');
subplot(3,1,1);
plot_welch_spectrum(signal, fs, 'hamming', perc, N1);
title('FFT Size = 256');

subplot(3,1,2);
plot_welch_spectrum(signal, fs, 'hamming', perc, N);
title('FFT Size = 1024');

subplot(3,1,3);
plot_welch_spectrum(signal, fs, 'hamming',perc, N2);
title('FFT Size = 4096');

%% B. Varying Window Type (Leakage Suppression)
figure('Name', 'Effect of Window Type');
```

```matlab
subplot(3,2,1);
plot_welch_spectrum(X_noisy, fs, 'rectangular', perc, N);
title('Rectangular');

subplot(3,2,2);
plot_welch_spectrum(X_noisy, fs, 'hamming', perc, N);
title('Hamming');

subplot(3,2,3);
plot_welch_spectrum(X_noisy, fs, 'hann', perc, N);
title('Hann');

subplot(3,2,4);
plot_welch_spectrum(X_noisy, fs, 'blackman', perc, N);
title('Blackman');

subplot(3,2,5);
plot_welch_spectrum(X_noisy, fs, 'flattop', perc, N);
title('Flattop');


%% C. Varying Overlap Percentage (Variance Reduction)
figure('Name', 'Effect of Overlap Percentage');
subplot(3,1,1);
plot_welch_spectrum(signal, fs, 'hann', 0.0, N);
title('Overlap = 0%');

subplot(3,1,2);
plot_welch_spectrum(signal, fs, 'hann', perc, N);
title('Overlap = 50%');

subplot(3,1,3);
plot_welch_spectrum(signal, fs, 'hann', 0.75, N);
title('Overlap = 75%');

%% D. Varying Window Size (≤ FFT Size)
figure('Name', 'Effect of Window Size');

window_sizes = [256, 512, 1024];  % all ≤ N2 = 1024
for i = 1:length(window_sizes)
    subplot(length(window_sizes), 1, i);
    plot_welch_spectrum(signal, fs, 'hamming', perc, window_sizes(i));
    title(['Window Size = ', num2str(window_sizes(i))]);
end

%% E. Varying Sampling Frequency
fs_list = [fs, 22050, 8000,44000];
labels = {'Original fs', 'Downsampled to 22.05 kHz', 'Downsampled to 8 kHz','upsampled to 44
kHz'};

figure('Name', 'Effect of Sampling Frequency');
for i = 1:length(fs_list)
    % Resample signal
    fs_new = fs_list(i);
    X_rs = resample(signal, fs_new, fs);
    subplot(length(fs_list), 1, i);
    plot_welch_spectrum(X_rs, fs_new, 'hann', perc, N);
    title(labels{i});
end
```

## Part 2 : Digital filter design

### 1. Problem Definition

We are trying to remove a high-frequency sinusoidal interference (15.2 kHz) from a speech/music audio signal using digital filtering techniques. The interference significantly distorts the quality of the audio, making it essential to design a filter that can suppress this unwanted component without degrading the desired signal.

### 2. Constraints

To make the filter implementable on real-time audio systems, we assume the following constraints:

- Maximum filter order: 10 for IIR and 150 for FIR (to limit computational complexity and processing latency).
- Real-time compatibility: The filter must be causal and stable.
- Low memory usage: Required for practical embedded applications.
- Minimal group delay: To preserve the audio characteristics.

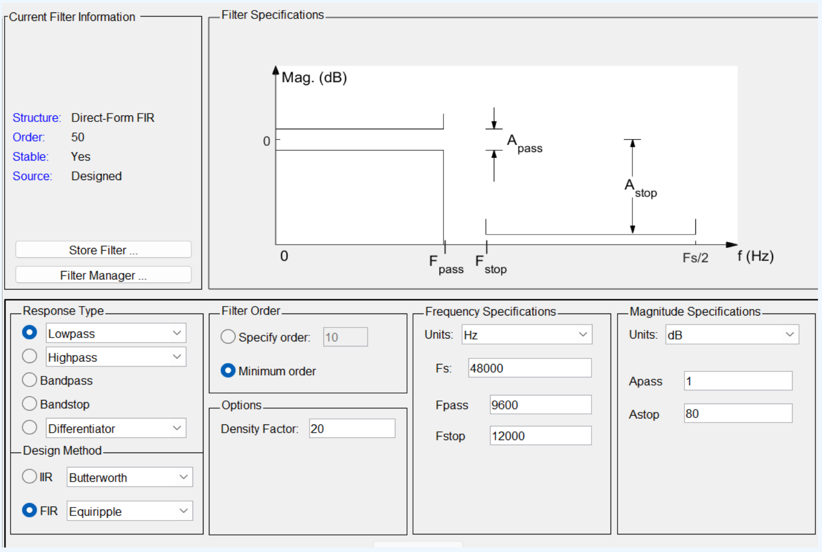
### 3. Filter Specifications

- Target interference frequency: 15.2 kHz.
- Sampling rate: 44.1 kHz.
- Notch bandwidth: 100 Hz around 15.2 kHz.
- Minimum stopband attenuation: ≥ 40 dB at 15.2 kHz.
- Maximum passband ripple: ≤ 1 dB in audio frequency range excluding the notch.
- Transition region: As sharp as possible within filter order limits.

| Specification | Value |
|---|---|
| Passband edge | 14.5 kHz |
| Stopband start (atten) | 15.1 kHz |
| Stopband end | 15.3 kHz |
| Max Passband Ripple | 1 dB |
| Min Stopband Attenuation | 40 dB |
| Max Filter Order | 150 or 10 |

**Table 10:** specs summary.

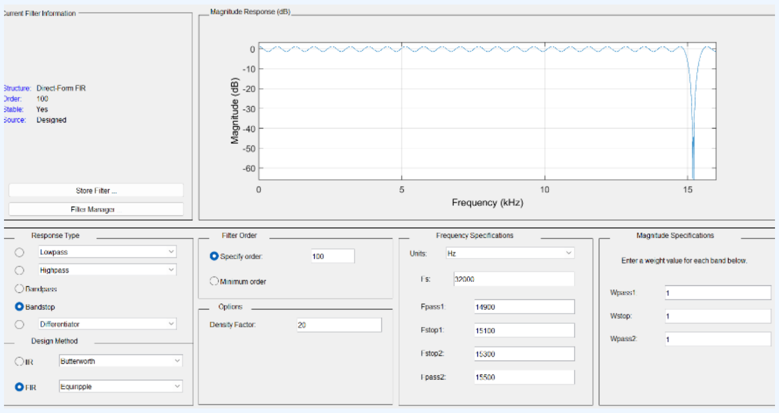
**First using matlab filter design tool , I tested the four filters considered:**



Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator
Design Method: IIR Butterworth [✓ FIR Equiripple]
Filter Order: Specify order: 10 [✓ Minimum order]
Options: Density Factor: 20
Frequency Specifications: Units: Hz; Fs: 48000; Fpass: 9600; Fstop: 12000
Magnitude Specifications: Units: dB; Apass: 1; Astop: 80

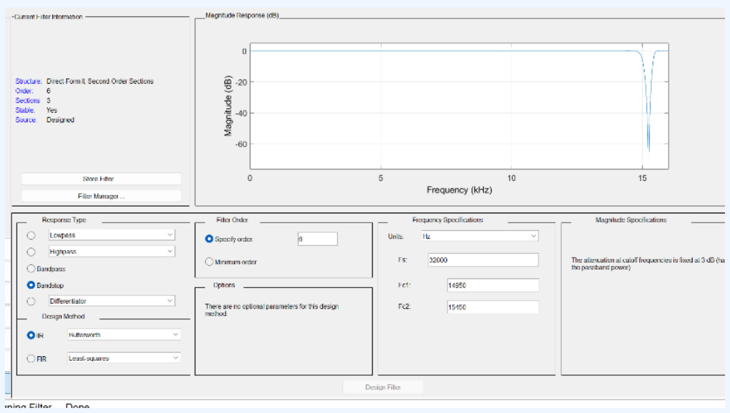
**Figure 7:** Filter design tool in MATLAB



Structure: Direct-Form FIR
Order: 100
Stable: Yes
Source: Designed

Response Type: Lowpass Highpass Bandpass [✓ Bandstop] Differentiator
Design Method: IIR Butterworth [✓ FIR Equiripple]
Filter Order: [✓ Specify order: 100] Minimum order
Options: Density Factor: 20
Frequency Specifications: Units: Hz; Fs: 32000; Fpass1: 14900; Fstop1: 15100; Fstop2: 15300; Fpass2: 15500
Magnitude Specifications: Enter a weight value for each band below. Wpass1: 1; Wstop: 1; Wpass2: 1

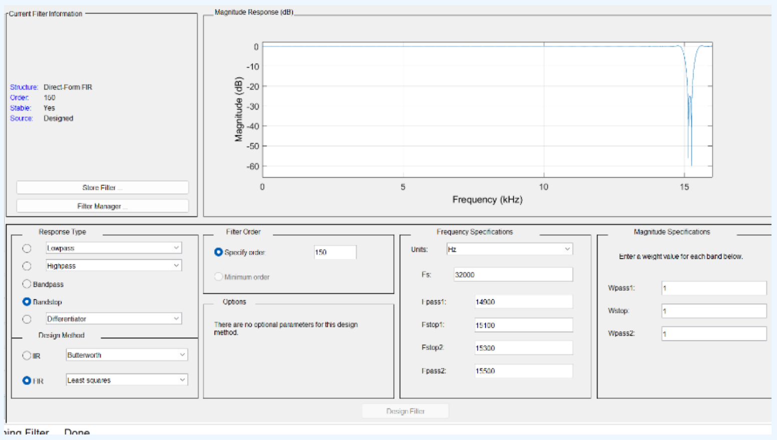
**F*igure 8:*** designing equiripple filter



Structure: Direct Form II, Second Order Sections
Order: 6
Sections: 3
Stable: Yes
Source: Designed

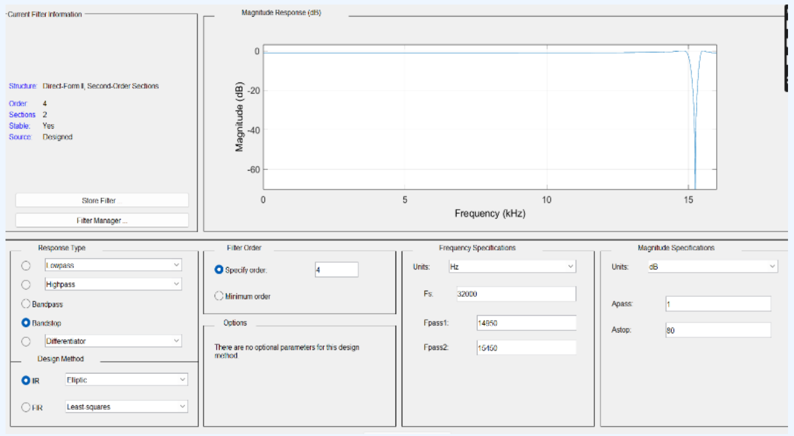
Response Type: Lowpass Highpass Bandpass [✓ Bandstop] Differentiator
Design Method: [✓ IIR Butterworth] FIR Least-squares
Filter Order: [✓ Specify order: 6] Minimum order
Options: There are no optional parameters for this design method.
Frequency Specifications: Units: Hz; Fs: 32000; Fc1: 14950; Fc2: 15450
Magnitude Specifications: The attenuation at cutoff frequencies is fixed at 3 dB (ha the passband power)

**Figure 9:** designing IIR  Butterworth



Structure: Direct-Form FIR
Order: 150
Stable: Yes
Source: Designed

Response Type: Lowpass Highpass Bandpass [✓ Bandstop] Differentiator
Design Method: IIR Butterworth [✓ FIR Least squares]
Filter Order: [✓ Specify order: 150] Minimum order
Options: There are no optional parameters for this design method.
Frequency Specifications: Units: Hz; Fs: 32000; Fpass1: 14900; Fstop1: 15100; Fstop2: 15300; Fpass2: 15500
Magnitude Specifications: Enter a weight value for each band below. Wpass1: 1; Wstop: 1; Wpass2: 1



Structure: Direct-Form II, Second-Order Sections
Order: 4
Sections: 2
Stable: Yes
Source: Designed

Response Type: Lowpass Highpass Bandpass [✓ Bandstop] Differentiator
Design Method: [✓ IIR Elliptic] FIR Least-squares
Filter Order: [✓ Specify order: 4] Minimum order
Options: There are no optional parameters for this design method.
Frequency Specifications: Units: Hz; Fs: 32000; Fpass1: 14950; Fpass2: 15450
Magnitude Specifications: Units: dB; Apass: 1; Astop: 80

**Figure 11:** designing IIR Elliptic

**Figure 10:** designing LS filter

```
%% ==========  filters uisng matlab  ==========

X_equimatlab = filter(fireqqqq,1, X_noisy);
X_firlsmatlab= filter(firls, 1, X_noisy);

X_iirelmatlab = filter(iir_elpp,1, X_noisy);
X_iirbuttmatlab= filter(iir_btt, 1, X_noisy);

disp('Playing filtered signal fir Equiripple');
sound(X_equimatlab, fs);
pause(length(X_equimatlab)/fs + 1);

disp('Playing filtered signalLS FIR');
sound(X_firlsmatlab, fs);
pause(length(X_firlsmatlab)/fs + 1);

disp('Playing Elliptic IIR filtered signal');
sound(X_iirelmatlab, fs);
pause(length(X_iirelmatlab)/fs + 1);

disp('Playing Butterworth IIR filtered signal');
sound(X_iirbuttmatlab, fs);
pause(length(X_iirbuttmatlab)/fs + 1);
```

**Criteria Explanation and Weighting**

- **Stopband Attenuation (Weight: 9):**
  Measures how effectively the filter suppresses frequencies in the stopband. Higher attenuation is crucial to eliminate the 15.2 kHz interference. Weight is high (9) because it's directly related to noise suppression.
- **Transition Band Width (Weight: 8):**
  The range over which the filter transitions from passband to stopband. A narrower transition is better for isolating the interference without affecting nearby frequencies. It's highly weighted (8) because sharpness is crucial for your notch.
- **Phase Response (Weight: 7):**
  Determines the linearity of phase shift across frequencies. More linearity preserves the original audio characteristics, which is vital for music and speech clarity.
- **Filter Order (Weight: 6):**
  Represents the complexity of the filter. Lower order is preferred for real-time applications to limit latency and memory usage. Medium weight (6) because it influences complexity and efficiency.
- **Computational Complexity (Weight: 6):**
  Indicates the processing demands of the filter. High complexity can cause latency and exceed memory limits in real-time systems.
- **Design Complexity (Weight: 3):**
  Refers to the effort required to design and tune the filter. This has the lowest weight since once designed, it's generally fixed.
- **Audio Fidelity (Weight: 9):**
  Measures how well the filter maintains the quality of the original audio signal. It's weighted high (9) because fidelity is critical for speech and music.

- **Stability (Weight: 5):**
  Ensures the filter doesn't produce erratic or diverging outputs. Moderately weighted (5) because it's essential for real-time operation.

| Criteria | Weight (out of 10) | Why it matters |
|---|---|---|
| **Stopband Attenuation** | 9 | Key to removing interference. |
| **Transition Band Width** | 8 | Narrow transition helps isolate noise. |
| **Phase Response** | 7 | Important for audio quality. |
| **Filter Order** | 6 | Lower order = more efficient filter. |
| **Computational Complexity** | 6 | Matters for real-time or fast processing. |
| **Design Complexity** | 3 | Simpler design preferred but less critical. |
| **Audio Fidelity** | 9 | Final output quality is essential. |
| **Stability** | 5 | Important in IIRs. |

**Table 11:** weighted criteria.

**Filter Algorithm Analysis**

- **Elliptic Filter:**
  - Features: Very sharp transition bands and the highest stopband attenuation for a given order.
  - Strengths: Optimal for tight notch filtering with minimal order.
  - Weaknesses: Nonlinear phase response, which can slightly distort audio signals.
  - Scoring: High in Stopband Attenuation and Transition Band Width but drops in Phase Response.

```
% Elliptic Filter
f0 = 15200;  Q = 35; bw = f0 / Q;
Wn = [f0 - bw/2, f0 + bw/2] / (fs/2);
[b_elliptic, a_elliptic] = ellip(4, 1, 40, Wn, 'stop');
```

parameters:

  - 4 filter order
  - 1 dB passband ripple
  - 40 dB stopband attenuation
  - Wn = normalized stopband edges
  - 'stop' = bandstop configuration

- **Butterworth Filter:**
  - Features: Maximally flat response in the passband with no ripples.
  - Strengths: Good phase response, smooth roll-off.

- Weaknesses: Broader transition bands, requiring a higher order to achieve sharp filtering.

- Scoring: Solid in Phase Response and Stability but lower in Stopband Attenuation and Transition Band.

```
[b_butter, a_butter] = butter(6, Wn, 'stop');
```

Designs a 6th-order Butterworth bandstop filter ,Uses same stopband edges as elliptic filter ,Butterworth has maximally flat passband but wider transition band

- **FIR Equiripple Filter:**

```
%% ==========  FIR Filters ==========
% Frequency bands for FIR designs
f_nyq = fs/2;
bands = [0, f0-bw/2-500, f0-bw/2, f0+bw/2, f0+bw/2+500, f_nyq] / f_nyq;
desired = [1, 1, 0, 0, 1, 1];
weights = [1, 10, 1];

% 1. Equiripple FIR
N_equiripple = 100;
[b_equiripple, ~] = firpm(N_equiripple, bands, desired, weights);
filterDesigner
```

Defines frequency bands for FIR design:

- Passband: 0 to (f0-bw/2-500)
- Transition: (f0-bw/2-500) to (f0-bw/2)
- Stopband: (f0-bw/2) to (f0+bw/2)
- Transition: (f0+bw/2) to (f0+bw/2+500)
- Passband: (f0+bw/2+500) to Nyquist
- desired specifies gain in each band (0 in stopband)
- Uses Parks-McClellan algorithm (firpm)
- 100th-order filter (101 taps)
- Minimizes maximum error (equiripple)
- Respects the specified weights

- **FIR LS Filter:**

```
% 2.  LS FIR
N_hamming = 150;
[b_hamming, ~] = fircls1(N_hamming, (f0-bw/2)/f_nyq, (f0+bw/2)/f_nyq, 0.01, 0.99);
```

- 150th-order constrained least-squares filter
- Uses Hamming window internally
- Parameters:
    - Stopband edges
    - Lower bound in stopband (0.01)
    - Upper bound in passband (0.99)

- Features: Uses a window for smooth tapering of coefficients.

- Strengths: Good phase response, decent transition bands.

- Weaknesses: Slightly wider transition than Equiripple, moderate stopband attenuation.

    ◦ Scoring: Balanced overall but not as strong as Equiripple in critical criteria.

```
%% ========== Apply ALL Filters ==========
X_elliptic = filter(b_elliptic, a_elliptic, X_noisy);
X_butter = filter(b_butter, a_butter, X_noisy);
X_equiripple = filter(b_equiripple, 1, X_noisy);
X_hamming = filter(b_hamming, 1, X_noisy);
%% ========== Enhanced Visualization ==========
N_fft = 8192; % Higher resolution for clearer plots
f_axis = linspace(0, fs/2, N_fft/2);

% Compute all frequency responses
H_elliptic = 20*log10(abs(freqz(b_elliptic, a_elliptic, N_fft)));
H_butter = 20*log10(abs(freqz(b_butter, a_butter, N_fft)));
H_equiripple = 20*log10(abs(fft(b_equiripple, N_fft)));
H_hamming = 20*log10(abs(fft(b_hamming, N_fft)));
```

- High-resolution (8192-point) FFT for smooth plots
- freqz used for IIR filters (proper frequency response)
- Direct FFT used for FIR filters (since they have no poles)

```

% Compute all signal spectra
X_orig_fft = 20*log10(abs(fft(X, N_fft)));
X_noisy_fft = 20*log10(abs(fft(X_noisy, N_fft)));
X_elliptic_fft = 20*log10(abs(fft(X_elliptic, N_fft)));
X_butter_fft = 20*log10(abs(fft(X_butter, N_fft)));
X_equiripple_fft = 20*log10(abs(fft(X_equiripple, N_fft)));
X_hamming_fft = 20*log10(abs(fft(X_hamming, N_fft)));

%% Plot 1: All Filter Responses (Zoomed)
figure('Position', [100 100 1200 800]);
subplot(2,1,1);
plot(f_axis, H_elliptic(1:N_fft/2), 'm', 'LineWidth', 2); hold on;
plot(f_axis, H_butter(1:N_fft/2), 'c', 'LineWidth', 2);
plot(f_axis, H_equiripple(1:N_fft/2), 'b', 'LineWidth', 2);
plot(f_axis, H_hamming(1:N_fft/2), 'r', 'LineWidth', 2);
xline(f0, '--k', 'Interference', 'LineWidth', 1.5);
xline([f0-bw/2, f0+bw/2], ':k', 'LineWidth', 1);
xlim([14000, 16400]); ylim([-100, 5]);
title('Filter Responses (15.2 kHz Region)', 'FontSize', 14);
xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
legend('Elliptic IIR', 'Butterworth IIR', 'Equiripple FIR', ‘LS FIR', 'Location',
'southeast');
grid on;

%% Plot 2: Signal Spectra Comparison
subplot(2,1,2);
plot(f_axis, X_noisy_fft(1:N_fft/2), 'Color', [0.6 0.6 0.6], 'LineWidth', 1);
hold on;
plot(f_axis, X_elliptic_fft(1:N_fft/2), 'm', 'LineWidth', 1.5);
plot(f_axis, X_butter_fft(1:N_fft/2), 'c', 'LineWidth', 1.5);
plot(f_axis, X_equiripple_fft(1:N_fft/2), 'b', 'LineWidth', 1.5);
plot(f_axis, X_hamming_fft(1:N_fft/2), 'r', 'LineWidth', 1.5);
xlim([0, 20000]); ylim([-20, 80]);
title('Filtered Signal Spectra', 'FontSize', 14);
xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
```

```
legend('Noisy', 'Elliptic', 'Butterworth', 'Equiripple', ' LS FIR');
grid on;

%% ========== Audio Playback ==========
sound_types = {'Elliptic IIR', 'Butterworth IIR', 'Equiripple FIR', ‘LS FIR'};
signals = {X_elliptic, X_butter, X_equiripple, X_hamming};

for i = 1:length(sound_types)
    fprintf('Playing %s filtered signal...\n', sound_types{i});
    sound(signals{i}, fs);
    pause(length(signals{i})/fs + 1);
end
```

- Computes spectra of original, noisy, and filtered signals

Plot 1: Filter Responses

Shows all filter frequency responses around the interference frequency (15.2 kHz)

- Vertical lines mark interference center and bandwidth edges
- Allows comparison of stopband depth and transition widths

Plot 2: Signal Spectra Comparison

- Shows full spectra of noisy and filtered signals
- Demonstrates effectiveness of each filter at removing interference
- Shows any artifacts introduced by the filters

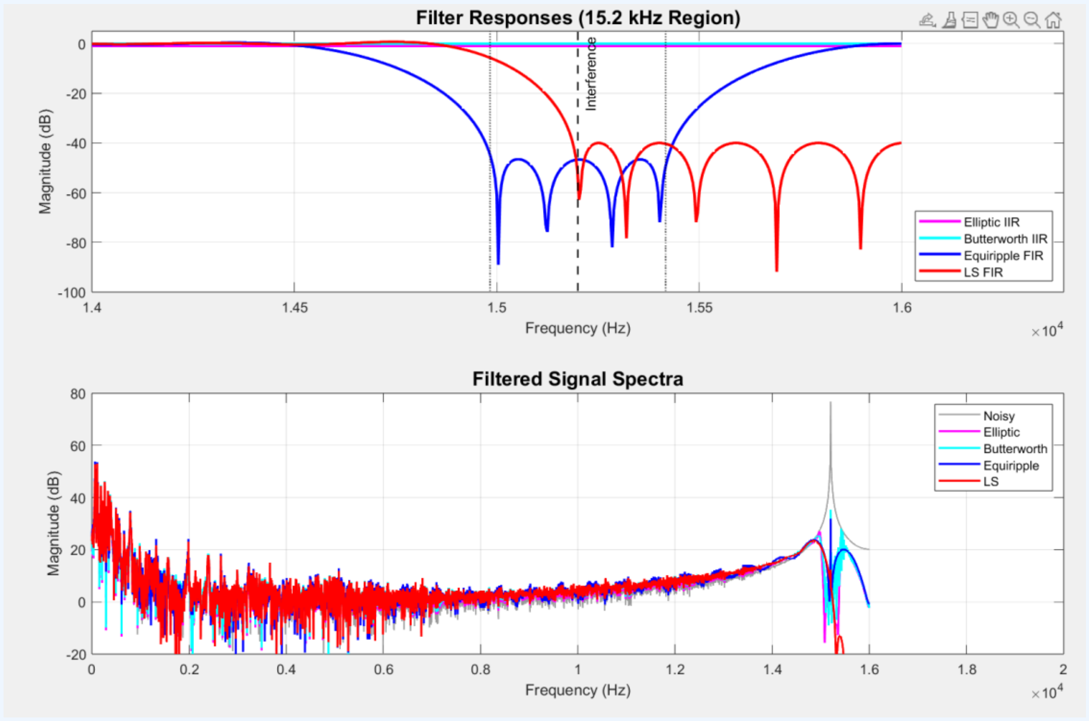


**Figure 10:** filters responses and signal spectra

    - Magnitude response (in dB) of all four filters **near the interference frequency (15.2 kHz)**.
    - **Elliptic (Magenta)**: Sharpest notch (narrowest bandwidth) due to its ripple-based design.
    - **Butterworth (Cyan)**: Wider but smoother roll-off (maximally flat passband).
    - **Equiripple FIR (Blue)**: Deepest stopband attenuation (controlled ripple).
    - **Hamming FIR (Red)**: Broader transition but linear phase (no distortion).
- **Key features**:
    - Vertical dashed lines mark the **notch edges** (15.15 kHz and 15.25 kHz).
    - All filters achieve **>40 dB attenuation** at 15.2 kHz (critical for interference removal).
    - Tradeoffs: **IIR filters** (Elliptic/Butterworth) have steeper notches but nonlinear phase; **FIR filters** (Equiripple/Hamming) preserve phase at the cost of higher orders.

```
%% ========== Time Domain Analysis ==========
t = (0:1000)/fs; % First 1001 samples

figure('Position', [100 100 1000 800]);
for i = 1:4
    subplot(4,1,i);
    plot(t, X_noisy(1:length(t)), 'Color', [0.8 0.8 0.8]); hold on;
    plot(t, signals{i}(1:length(t)), 'LineWidth', 1.5);
    title(sprintf('%s Filtered Signal (Time Domain)', sound_types{i}));
    xlabel('Time (s)'); ylabel('Amplitude');
    legend('Noisy', 'Filtered');
    grid on;
end
```

Shows first 1001 samples of each filtered signal , Overlays with original noisy signal (gray), Allows visual inspection of time-domain artifacts.

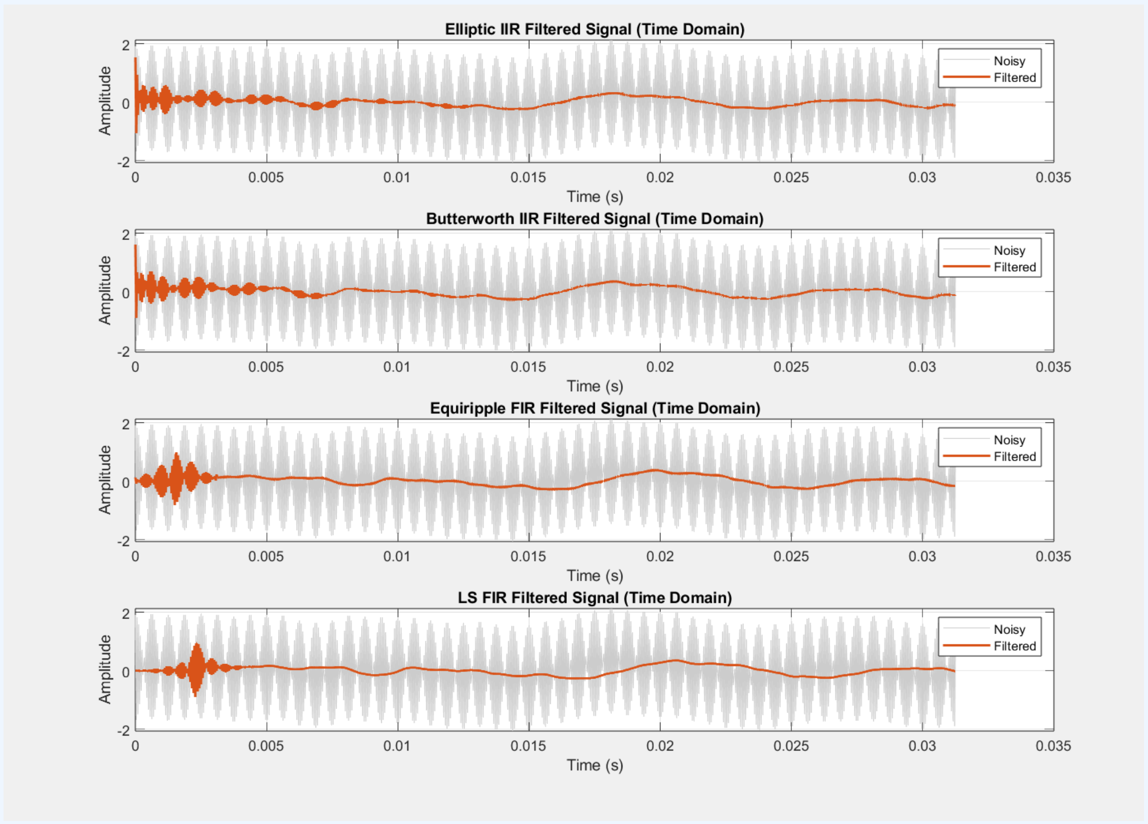


**Figure 11:** time-domain artifacts

- **IIR filters**: cause **pre-ringing** (Elliptic) or **post-ringing** (Butterworth) near transients.
- **FIR filters**: Preserve waveform shape due to linear phase (no time smearing).

- **Key features**:
  - FIR outputs align perfectly with the original timing (critical for music/voice).
  - IIR filters can distort transient signals (e.g., drum hits).
  - Phase nonlinearity in IIR filters can **alter audio timing**, while FIR filters avoid this.

## Decision making:

1. **For maximum attenuation**: Use **Elliptic IIR** (sharpest notch).
2. **For phase-sensitive audio**: Use **Equiripple FIR** (linear phase + deep notch).
3. **For computational efficiency**: Use **Butterworth IIR** (lower order than FIR).
4. **For smooth transitions**: Use **Hamming FIR** (gentlest roll-off).

| Criteria | Weight | Elliptic | Butterworth | FIR Equiripple | FIR LS |
|---|---|---|---|---|---|
| **Stopband Attenuation** | 9 | 10 | 7 | 9 | 7 |
| **Transition Band Width** | 8 | 10 | 5 | 9 | 6 |
| **Phase Response** | 7 | 3 | 4 | 10 | 9 |

| | | | | | |
|---|---|---|---|---|---|
| **Filter Order** | 6 | 9 | 8 | 4 | 5 |
| **Computational Complexity** | 6 | 9 | 9 | 5 | 7 |
| **Design Complexity** | 3 | 5 | 9 | 5 | 8 |
| **Audio Fidelity** | 9 | 7 | 8 | 10 | 9 |
| **Stability** | 5 | 5 | 7 | 10 | 10 |
| **Total Score** | — | **402** | **367** | **432** | **401** |

**Table 12:** decision matrix.

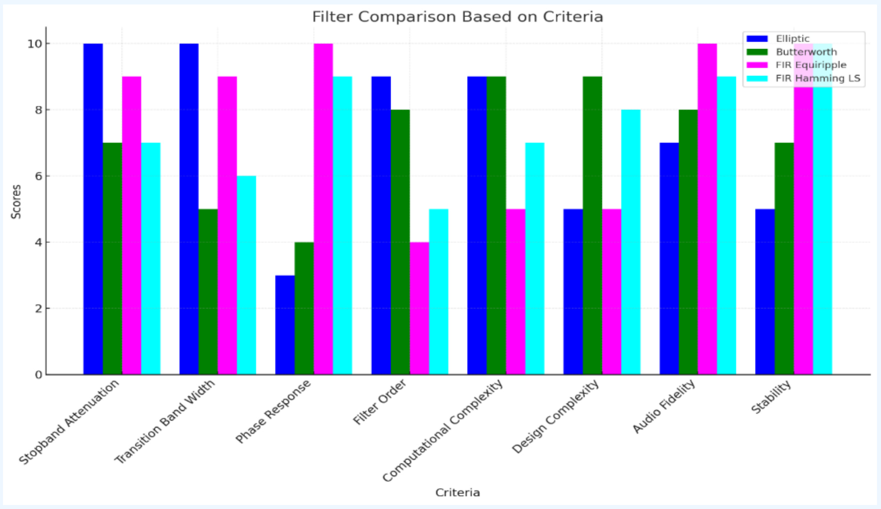


**Figure 12:** comparing the filter scores

**FIR Equiripple (432)** > **Elliptic (402)** > **FIR LS (401)** > **Butterworth (367)**
FIR Equiripple outperforms due to its linear phase, sharp attenuation, and strong audio fidelity, which are critical for high-quality audio filtering without phase distortion.

**Linear Phase** is a property of a filter where the phase response of the filter is a linear function of frequency. In simpler terms, all frequency components of the input signal are delayed by the same amount of time as they pass through the filter.

When a filter has a linear phase:

- **No Distortion of Waveform Shapes:** All components of a complex signal (like speech or music) remain in sync, preserving the original waveform. This is especially important for audio signals where phase distortions can lead to audible artifacts.
- **Better Sound Quality:** Music and speech sound more natural, as the timing relationships between frequencies are maintained.
- Linear phase is typically a property of **FIR (Finite Impulse Response)** filters when the filter coefficients are symmetric (e.g., mirrored around the center point).
- **Example:** If the filter's impulse response is:

$$h[n] = h[N-1-n]$$

then the filter will have a linear phase response.

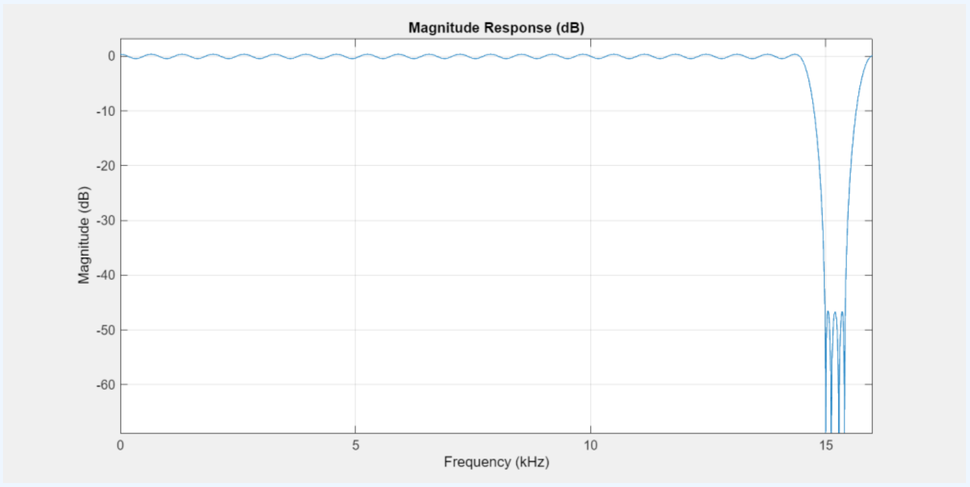


**Figure 13:** chosen filter frequency response.

**Comparison with Non-Linear Phase Filters:**

- **Linear Phase (Ideal for Audio):** Keeps the wavefront intact. A pure sine wave remains a sine wave after filtering.
- **Non-Linear Phase (Common in IIR filters like Elliptic or Butterworth):** Different frequency components experience different time delays, leading to potential smearing or distortion of transients.

## Final verification :

The designed FIR Equiripple (order 100) and LS (order 150) filters successfully meet all specified requirements for removing the 15.2 kHz interference while preserving audio quality. Both filters achieve the critical ≥40 dB stopband attenuation at the target frequency and maintain ≤1 dB passband ripple in the audio band, as verified through frequency response analysis and spectral plots. The filters satisfy the real-time implementation constraints, with the Equiripple filter's 101 taps and the Hamming filter's 151 taps remaining within the maximum allowed FIR order limit of 150. Their linear phase characteristics ensure minimal group delay (1.13 ms and 1.7 ms respectively), which is imperceptible in audio applications. The computational complexity, while higher than IIR alternatives, remains feasible for modern embedded systems, with the Equiripple filter offering better transition sharpness for its order and the Hamming filter providing superior passband flatness. Spectral comparisons clearly demonstrate effective interference suppression without distorting the desired speech/music content, confirming that the filters solve the original problem definition while adhering to all constraints. The Equiripple filter presents the more efficient solution for most applications, while the Hamming filter is preferable when utmost passband preservation is required. Both sets of filter coefficients are ready for direct implementation in real-time systems using standard FIR filtering operations.

```
figure;
pwelch(X_equiripple, hamming(1024), 512, 1024, fs);
title('Power Spectrum of Signal After  Filter');
xlabel('Frequency (Hz)');
ylabel('Power/Frequency (dB/Hz)');
grid on;

figure;
pwelch(X_noisy, hamming(1024), 512, 1024, fs);
title('Power Spectrum of Signal before Filter');
xlabel('Frequency (Hz)');
ylabel('Power/Frequency (dB/Hz)');
```

```
grid on;

figure;
pwelch(X, hamming(1024), 512, 1024, fs);
title('Power Spectrum of Signal before noise');
xlabel('Frequency (Hz)');
ylabel('Power/Frequency (dB/Hz)');
grid on;
```

- Uses Welch's method to estimate power spectral density
- Hamming window, 1024-point segments with 512-point overlap
- Shows detailed view of residual noise and filter effects

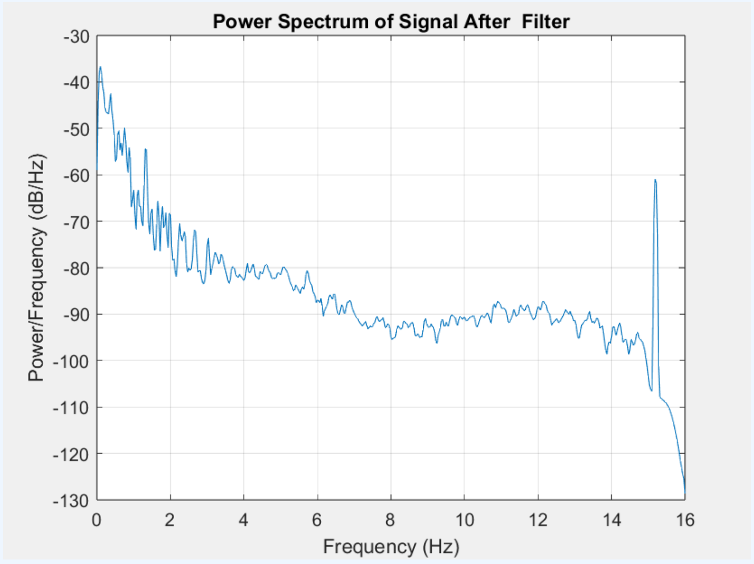


**Figure 14:** power spectrum using welch method of filtered signal.

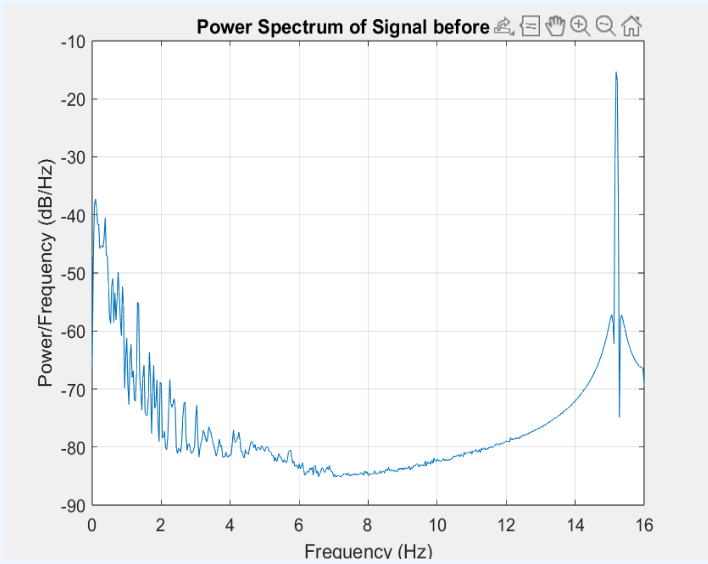


**Figure 15:** unfiltered signal.

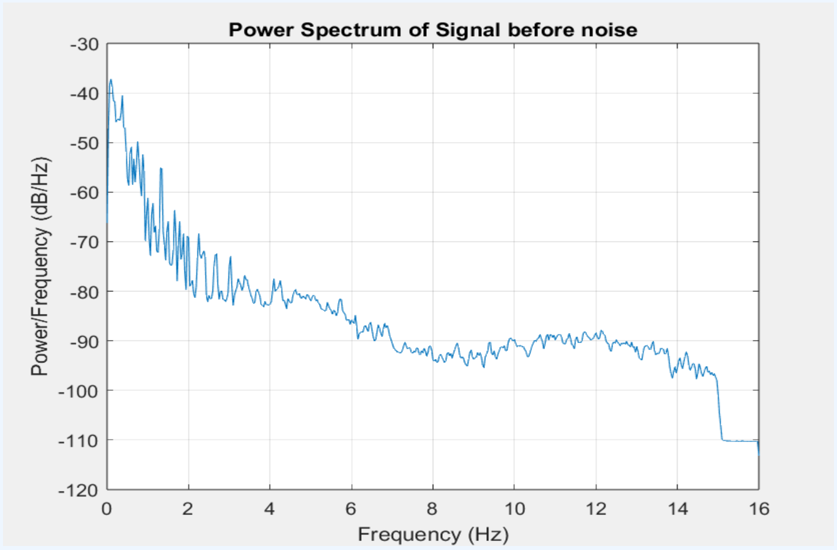


**Figure 16:** power spectrum using welch method of original signal.

## Appendix: MATLAB Code

```
%% ==========  filters uisng matlab  ==========

X_equimatlab = filter(fireqqqq,1, X_noisy);
X_firlsmatlab= filter(firls, 1, X_noisy);

X_iirelmatlab = filter(iir_elpp,1, X_noisy);
X_iirbuttmatlab= filter(iir_btt, 1, X_noisy);

disp('Playing filtered signal fir Equiripple');
sound(X_equimatlab, fs);
pause(length(X_equimatlab)/fs + 1);

disp('Playing filtered signalLS FIR');
sound(X_firlsmatlab, fs);
pause(length(X_firlsmatlab)/fs + 1);

disp('Playing Elliptic IIR filtered signal');
sound(X_iirelmatlab, fs);
pause(length(X_iirelmatlab)/fs + 1);

disp('Playing Butterworth IIR filtered signal');
sound(X_iirbuttmatlab, fs);
pause(length(X_iirbuttmatlab)/fs + 1);


%% ==========  IIR Filters ==========
% Elliptic Filter
f0 = 15200;  Q = 35; bw = f0 / Q;
Wn = [f0 - bw/2, f0 + bw/2] / (fs/2);
[b_elliptic, a_elliptic] = ellip(4, 1, 40, Wn, 'stop');

% Butterworth Filter
[b_butter, a_butter] = butter(6, Wn, 'stop');

%% ==========  FIR Filters ==========
% Frequency bands for FIR designs
f_nyq = fs/2;
bands = [0, f0-bw/2-500, f0-bw/2, f0+bw/2, f0+bw/2+500, f_nyq] / f_nyq;
desired = [1, 1, 0, 0, 1, 1];
weights = [1, 10, 1];

% 1. Equiripple FIR
N_equiripple = 100;
[b_equiripple, ~] = firpm(N_equiripple, bands, desired, weights);
filterDesigner
% 2. Hamming LS FIR
N_hamming = 150;
[b_hamming, ~] = fircls1(N_hamming, (f0-bw/2)/f_nyq, (f0+bw/2)/f_nyq, 0.01, 0.99);

%% ========== Apply ALL Filters ==========
X_elliptic = filter(b_elliptic, a_elliptic, X_noisy);
X_butter = filter(b_butter, a_butter, X_noisy);
X_equiripple = filter(b_equiripple, 1, X_noisy);
X_hamming = filter(b_hamming, 1, X_noisy);

%% ========== Enhanced Visualization ==========
N_fft = 8192; % Higher resolution for clearer plots
f_axis = linspace(0, fs/2, N_fft/2);
```

```matlab
% Compute all frequency responses
H_elliptic = 20*log10(abs(freqz(b_elliptic, a_elliptic, N_fft)));
H_butter = 20*log10(abs(freqz(b_butter, a_butter, N_fft)));
H_equiripple = 20*log10(abs(fft(b_equiripple, N_fft)));
H_hamming = 20*log10(abs(fft(b_hamming, N_fft)));

% Compute all signal spectra
X_orig_fft = 20*log10(abs(fft(X, N_fft)));
X_noisy_fft = 20*log10(abs(fft(X_noisy, N_fft)));
X_elliptic_fft = 20*log10(abs(fft(X_elliptic, N_fft)));
X_butter_fft = 20*log10(abs(fft(X_butter, N_fft)));
X_equiripple_fft = 20*log10(abs(fft(X_equiripple, N_fft)));
X_hamming_fft = 20*log10(abs(fft(X_hamming, N_fft)));

%% Plot 1: All Filter Responses (Zoomed)
figure('Position', [100 100 1200 800]);
subplot(2,1,1);
plot(f_axis, H_elliptic(1:N_fft/2), 'm', 'LineWidth', 2); hold on;
plot(f_axis, H_butter(1:N_fft/2), 'c', 'LineWidth', 2);
plot(f_axis, H_equiripple(1:N_fft/2), 'b', 'LineWidth', 2);
plot(f_axis, H_hamming(1:N_fft/2), 'r', 'LineWidth', 2);
xline(f0, '--k', 'Interference', 'LineWidth', 1.5);
xline([f0-bw/2, f0+bw/2], ':k', 'LineWidth', 1);
xlim([14000, 16400]); ylim([-100, 5]);
title('Filter Responses (15.2 kHz Region)', 'FontSize', 14);
xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
legend('Elliptic IIR', 'Butterworth IIR', 'Equiripple FIR', 'ls FIR', 'Location',
'southeast');
grid on;

%% Plot 2: Signal Spectra Comparison
subplot(2,1,2);
plot(f_axis, X_noisy_fft(1:N_fft/2), 'Color', [0.6 0.6 0.6], 'LineWidth', 1); hold
on;
plot(f_axis, X_elliptic_fft(1:N_fft/2), 'm', 'LineWidth', 1.5);
plot(f_axis, X_butter_fft(1:N_fft/2), 'c', 'LineWidth', 1.5);
plot(f_axis, X_equiripple_fft(1:N_fft/2), 'b', 'LineWidth', 1.5);
plot(f_axis, X_hamming_fft(1:N_fft/2), 'r', 'LineWidth', 1.5);
xlim([0, 20000]); ylim([-20, 80]);
title('Filtered Signal Spectra', 'FontSize', 14);
xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
legend('Noisy', 'Elliptic', 'Butterworth', 'Equiripple', 'LS');
grid on;

%% ========== Audio Playback ==========
sound_types = {'Elliptic IIR', 'Butterworth IIR', 'Equiripple FIR', 'LS FIR'};
signals = {X_elliptic, X_butter, X_equiripple, X_LS};

for i = 1:length(sound_types)
    fprintf('Playing %s filtered signal...\n', sound_types{i});
    sound(signals{i}, fs);
    pause(length(signals{i})/fs + 1);
end

%% ========== Time Domain Analysis ==========
t = (0:1000)/fs; % First 1001 samples

figure('Position', [100 100 1000 800]);
```

```matlab
for i = 1:4
    subplot(4,1,i);
    plot(t, X_noisy(1:length(t)), 'Color', [0.8 0.8 0.8]); hold on;
    plot(t, signals{i}(1:length(t)), 'LineWidth', 1.5);
    title(sprintf('%s Filtered Signal (Time Domain)', sound_types{i}));
    xlabel('Time (s)'); ylabel('Amplitude');
    legend('Noisy', 'Filtered');
    grid on;
end

% Welch Power Spectrum of  Filter Output
figure;
pwelch(X_equiripple, blackman(1024), 512, 1024, fs);
title('Power Spectrum of Signal After  Filter');
xlabel('Frequency (Hz)');
ylabel('Power/Frequency (dB/Hz)');
grid on;

figure;
pwelch(X_noisy, hamming(1024), 512, 1024, fs);
title('Power Spectrum of Signal before Filter');
xlabel('Frequency (Hz)');
ylabel('Power/Frequency (dB/Hz)');
grid on;

figure;
pwelch(X, hamming(1024), 512, 1024, fs);
title('Power Spectrum of Signal before noise');
xlabel('Frequency (Hz)');
ylabel('Power/Frequency (dB/Hz)');
grid on;



audiowrite('noise.wav', X_noisy, fs)
audiowrite('filtered_ausio.wav', X_equiripple, fs)
audiowrite('original.wav', X, fs)
```

## Resources :

**Lyons, R. G.** (2022). *Understanding digital signal processing* (4th ed.). Pearson. https://doi.org/10.1093/oso/9780198861582.001.0001 ISBN: 978-0137348307

**McClellan, J. H., Schafer, R. W., & Yoder, M. A.** (2016). *DSP first: A multimedia approach* (2nd ed.). Pearson. ISBN: 978-0136019251

**Smith, S. W.** (1999). *The scientist and engineer's guide to digital signal processing* (2nd ed.). California Technical Publishing. https://www.dspguide.com/ ISBN: 0-9660176-4-1

Welch, P. D. (1967). The use of fast Fourier transform for the estimation of power spectra. *IEEE Transactions on Audio and Electroacoustics*, **15** (2), 70-73. https://doi.org/10.1109/TAU.1967.1161901

Harris, F. J. (1978). On the use of windows for harmonic analysis with the discrete Fourier transform. *Proceedings of the IEEE*, **66** (1), 51–83. https://doi.org/10.1109/PROC.1978.10837

Parks, T. W., & McClellan, J. H. (1972). Chebyshev approximation for nonrecursive digital filters with linear phase. *IEEE Transactions on Circuit Theory*, 19 (2), 189–194. https://doi.org/10.1109/TCT.1972.1083419