**IIR FILTER CODE :**

%-----------------------Entering original audio--------------------------------------

%Read filename of audio using a GUI

[filename, pathname] = uigetfile();

filename = strcat(pathname, filename);

% Load the audio file and play it

[inputAudio, Fs] = audioread(filename);

org\_audio\_play = audioplayer(inputAudio, Fs);

play(org\_audio\_play);

pause(5);

stop(org\_audio\_play);

% Display the original audio waveform with respect to time

figure;

subplot(3,1,1);

plot(inputAudio);

title('Original Audio');

%-----------------------Orignal audio treatment--------------------------------------

% Calculate the FFT of the Orignal audio

N = length(inputAudio);

Fourier\_Coeff = fft(inputAudio);

% Compute the single-sided spectrum of the Orignal audio

frequencies = linspace(0, Fs/2, N/2 + 1);

amplitude = abs(Fourier\_Coeff(1:N/2 + 1));

% Plot the single-sided spectrum of Orignal audio

figure;

plot(frequencies, amplitude);

title('Orignal audio single-Sided Frequency Spectrum');

xlabel('Frequency (Hz)');

ylabel('Amplitude');

%-----------------------Noise Synthesis and treatment-----------------------

% Ask the user for the frequency range

prompt = 'Enter the lower bound of the frequency range of the needed noise signal: ';

lowFreq = input(prompt);

prompt = 'Enter the upper bound of the frequency range of the needed noise signal: ';

highFreq = input(prompt);

prompt = 'Enter the needed noise amplitude: ';

noiseAmplitude = input(prompt);

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

[rows, col] = size(inputAudio);

noisySignal = zeros(rows, col);

for freq = lowFreq:highFreq

noisySignal(:, 1) = noisySignal(:, 1) + (cos(2 \* pi \* freq \* t))'; % Channel 1

noisySignal(:, 2) = noisySignal(:, 2) + (sin(2 \* pi \* freq \* t))'; % Channel 2

end

% Calculate the FFT of the Noisy Signal

N = length(noisySignal);

Fourier\_Coeff = fft(noisySignal);

% Compute the single-sided spectrum of the Noisy Signal

frequencies = linspace(0, Fs/2, N/2 + 1);

amplitude = abs(Fourier\_Coeff(1:N/2 + 1));

% Plot the single-sided spectrum of the Noisy Signal

figure;

plot(frequencies, amplitude);

title('Noise Signal single-Sided Frequency Spectrum');

xlabel('Frequency (Hz)');

ylabel('Amplitude');

%-----------------------Add Noisy Signal to original audio-----------------------

% Add noise to the original audio

noisyAudio = inputAudio + noisySignal;

% Display the Noisy audio waveform with respect to time

figure;

subplot(3,1,2);

plot(noisyAudio);

title('Noisy Audio');

%play the noised audio

Filtered\_audio\_play = audioplayer(noisyAudio, Fs);

play(Filtered\_audio\_play);

pause(5);

stop(Filtered\_audio\_play);

%-----------------------Noisy audio treatment--------------------------------------

% Calculate the FFT of the Noisy audio

N = length(noisyAudio);

Fourier\_Coeff = fft(noisyAudio);

% Compute the single-sided spectrum of the Noisy audio

frequencies = linspace(0, Fs/2, N/2 + 1);

amplitude = abs(Fourier\_Coeff(1:N/2 + 1));

% Plot the single-sided spectrum of the Noisy audio

figure;

plot(frequencies, amplitude);

title('Noisy audio single-Sided Frequency Spectrum');

xlabel('Frequency (Hz)');

ylabel('Amplitude');

%---------------------Apply filter designed at matlab "filterDesigner"---------------------

% Apply the IIR filter to the audio signal

filteredAudio = filter(Try\_1, noisyAudio);

% Display the filtered audio waveform with respect to time

figure;

subplot(3,1,3);

plot(filteredAudio);

title('Filtered Audio');

%play the filtered audio

Filtered\_audio\_play = audioplayer(filteredAudio, Fs);

play(Filtered\_audio\_play);

pause(5);

stop(Filtered\_audio\_play);

%-----------------------Filtered Audio treatment--------------------------------------

% Calculate the FFT of the filtered audio

N = length(filteredAudio);

Fourier\_Coeff = fft(filteredAudio);

% Compute the single-sided spectrum

frequencies = linspace(0, Fs/2, N/2 + 1);

amplitude = abs(Fourier\_Coeff(1:N/2 + 1));

% Plot the single-sided spectrum

figure;

plot(frequencies, amplitude);

title('Filtered audio single-Sided Frequency Spectrum');

xlabel('Frequency (Hz)');

ylabel('Amplitude');

%-----------------------Experiment parameters(SNR, PSNR, MSE)--------------------------------------

%SNR:

% Calculate the power of the clean signal

cleanPower = mean(inputAudio.^2);

% Calculate the power of the residual noise

noiseSignal = inputAudio - filteredAudio;

noisePower = mean(noiseSignal.^2);

% Calculate the SNR in decibels

SNR = 10 \* log10(cleanPower / noisePower);

% Display the SNR value

fprintf('Signal-to-Noise Ratio (SNR): %.2f dB\n',SNR);

% Calculate MSE (Mean Squared Error)

mseValue = sum((inputAudio - filteredAudio).^2) / length(inputAudio);

% Calculate PSNR (Peak Signal-to-Noise Ratio)

maxAmplitude = max(abs(inputAudio));

psnrValue = 10 \* log10((maxAmplitude.^2) / mseValue);

% Display the results

fprintf('MSE: %f\n', mseValue);

fprintf('PSNR: %f dB\n', psnrValue);

من أول هنا🔻🔻🔻🔻🔻🔻🔻🔻🔻🔻🔻🔻🔻🔻🔻🔻

%A code to generate the required IIR without the need of the

%"filterDesigner" tool.

% % Design the IIR filter

% filterOrder = input("Enter a filter order : "); % Filter order

% cutoffFrequency = input("Enter a cut-off frequency : "); % Cutoff frequency in Hz

% Filter\_Design = input("Enter the needed filter design : ", 's');%Choose ['butterworth', 'chebyshev1', 'chebyshev2' & 'elliptic']

% % Normalize cutoff frequency

% normalizedCutoff = cutoffFrequency/(Fs/2);

% [b, a] = Filter\_Designer(Filter\_Design, filterOrder, normalizedCutoff)

% Apply the IIR filter to the audio signal

%filteredAudio = filter(b, a, noisyAudio);

% function [b, a] = Filter\_Designer(Filter\_Design, filterOrder, normalizedCutoff)

% switch Filter\_Design

% case 'butterworth'

% [b, a] = butter(filterOrder, normalizedCutoff, 'low');

% case 'chebyshev1'

% % Additional parameters can be added for Chebyshev Type I

% [b, a] = cheby1(filterOrder, 0.5, normalizedCutoff, 'low');

% case 'chebyshev2'

% % Additional parameters can be added for Chebyshev Type II

% [b, a] = cheby2(filterOrder, 20, normalizedCutoff, 'low');

% case 'elliptic'

% % Additional parameters can be added for elliptic filter

% [b, a] = ellip(filterOrder, 0.5, 20, normalizedCutoff, 'low');

% otherwise

% error('Unsupported filter type. Choose from Butterworth, Chebyshev1, Chebyshev2, or Elliptic.');

% end

% end

**FIR FILTER CODE :**

% Load the MP3 audio file using audioread

[filename, pathname] = uigetfile('\*.mp3', 'Select an MP3 audio file');

file\_path = fullfile(pathname, filename);

% Read the MP3 audio file using audioread

[inputAudio, Fs] = audioread(file\_path);

% ----------------------- Original Audio -----------------------

% Display the original audio waveform

figure;

subplot(5, 1, 1);

plot(inputAudio);

title('Original Audio');

xlabel('Time (s)');

ylabel('Amplitude');

% ----------------------- FIR Filter Design and Application -----------------------

% Design the FIR filters

firFilterOrder = 100; % Adjust the filter order as needed

firCutoffFrequency = 2000/Fs; % Adjust the cutoff frequency as needed

% Design FIR filter with Kaiser window

kaiserWindow = kaiser(firFilterOrder+1, 4);

kaiserFilter = fir1(firFilterOrder, firCutoffFrequency, 'low', kaiserWindow);

% Design FIR filter with Hamming window

hammingWindow = hamming(firFilterOrder+1);

hammingFilter = fir1(firFilterOrder, firCutoffFrequency, 'low', hammingWindow);

% Apply the FIR filters to the noisy audio signal

filteredAudioKaiser = filter(kaiserFilter, 1, inputAudio);

filteredAudioHamming = filter(hammingFilter, 1, inputAudio);

% ----------------------- FIR Filtered Audio Waveforms -----------------------

% Display Kaiser filtered audio waveform

subplot(5, 1, 2);

plot(filteredAudioKaiser);

title('Kaiser Filtered Audio');

xlabel('Time (s)');

ylabel('Amplitude');

% Display Hamming filtered audio waveform

subplot(5, 1, 3);

plot(filteredAudioHamming);

title('Hamming Filtered Audio');

xlabel('Time (s)');

ylabel('Amplitude');

% ----------------------- Measure MSE, SNR, and PSNR -----------------------

% Calculate SNR for Kaiser filtered audio

cleanPower\_Kaiser = mean(inputAudio.^2);

noiseSignal\_Kaiser = inputAudio - filteredAudioKaiser;

noisePower\_Kaiser = mean(noiseSignal\_Kaiser.^2);

SNR\_Kaiser = 10 \* log10(cleanPower\_Kaiser / noisePower\_Kaiser);

% Calculate MSE (Mean Squared Error) for Kaiser filtered audio

mseValue\_Kaiser = sum((inputAudio - filteredAudioKaiser).^2) / length(inputAudio);

% Calculate PSNR (Peak Signal-to-Noise Ratio) for Kaiser filtered audio

maxAmplitude\_Kaiser = max(abs(inputAudio));

psnrValue\_Kaiser = 10 \* log10((maxAmplitude\_Kaiser.^2) / mseValue\_Kaiser);

% Display the results for Kaiser filtered audio

fprintf('Kaiser Filter:\n');

fprintf('SNR: %.2f dB\n', SNR\_Kaiser);

fprintf('MSE: %f\n', mseValue\_Kaiser);

fprintf('PSNR: %f dB\n', psnrValue\_Kaiser);

% Calculate SNR for Hamming filtered audio

cleanPower\_Hamming = mean(inputAudio.^2);

noiseSignal\_Hamming = inputAudio - filteredAudioHamming;

noisePower\_Hamming = mean(noiseSignal\_Hamming.^2);

SNR\_Hamming = 10 \* log10(cleanPower\_Hamming / noisePower\_Hamming);

% Calculate MSE (Mean Squared Error) for Hamming filtered audio

mseValue\_Hamming = sum((inputAudio - filteredAudioHamming).^2) / length(inputAudio);

% Calculate PSNR (Peak Signal-to-Noise Ratio) for Hamming filtered audio

maxAmplitude\_Hamming = max(abs(inputAudio));

psnrValue\_Hamming = 10 \* log10((maxAmplitude\_Hamming.^2) / mseValue\_Hamming);

% Display the results for Hamming filtered audio

fprintf('Hamming Filter:\n');

fprintf('SNR: %.2f dB\n', SNR\_Hamming);

fprintf('MSE: %f\n', mseValue\_Hamming);

fprintf('PSNR: %f dB\n', psnrValue\_Hamming);

% Play the original audio

original\_audio\_player = audioplayer(inputAudio, Fs);

play(original\_audio\_player);

pause(5);

stop(original\_audio\_player);

% Play the filtered audio with Kaiser filter

kaiser\_audio\_player = audioplayer(filteredAudioKaiser, Fs);

play(kaiser\_audio\_player);

pause(5);

stop(kaiser\_audio\_player);

% Play the filtered audio with Hamming filter

hamming\_audio\_player = audioplayer(filteredAudioHamming, Fs);

play(hamming\_audio\_player);

pause(5);

stop(hamming\_audio\_player);

**WAVELETE CODE :**

clc

clear

% Load the audio file (replace 'noisy\_audio.wav' with your file)

[y, fs] =audioread("C:\Users\dell\Desktop\Low Frequency.mp3");

% If the audio is stereo, take the average of the channels

if size(y, 2) == 2

y =mean(y, 2);

end

% n=randn(size(y))\*0.02;

% y=y+n;

% Add white Gaussian noise

y = awgn(y,15, 'measured');

% Parameters

wavelet\_name = 'db4'; % Choose a wavelet (e.g., 'db4')

level = 25; % Decomposition level

threshold\_type = 's'; % Soft thresholding ('s') or hard thresholding ('h')

threshold\_value = 0.1; % Adjust this threshold value based on your signal

% Perform wavelet decomposition

[c, l] = wavedec(y, level, wavelet\_name);

% Thresholding

c\_thresh = wthresh(c, threshold\_type, threshold\_value);

% Reconstruct the denoised signal

y\_denoised = waverec(c\_thresh, l, wavelet\_name);

% Plot the original and denoised signals

figure;

subplot(2,1,1);

plot(y);

title('Original Noisy Signal');

subplot(2,1,2);

plot(y\_denoised);

title('Denoised Signal');

K=audioplayer(y, fs);

M=audioplayer(y\_denoised, fs);

% Play the original and denoised signals

play(K);

pause(5); % Pause to allow the first sound to finish

pause(K);

pause(1);

play(M);

pause(5); % Pause to allow the first sound to finish

pause(M);

% Assuming you have the original clean audio signal stored in 'cleanSignal'

% and the denoised signal in 'denoisedSignal'

% Calculate the mean squared error (MSE)

MSE = mean((y - y\_denoised).^2);

fprintf('Mean Square Error (MSE)): %f dB\n',MSE);

% Determine the maximum possible power of the clean signal

maxPower = double(intmax('int16')).^2;

% Calculate the PSNR in decibels

PSNR = 10 \* log10(maxPower / MSE);

% Display the PSNR value

fprintf('Peak Signal-to-Noise Ratio (PSNR): %.2f dB\n',PSNR);