**Experiment 9: Linear Phase F I R Filter Design using Frequency sampling method**

|  |  |
| --- | --- |
| **Name** | Prathamesh Mane |
| **UID no. & Branch** | **2022200078 (B1)** |
| **Experiment No.** | 9 |

Code :

clear;

clc;

% User Input

disp('Designing Low Pass Filter (LPF)');

Fs = input('Enter Sampling Frequency (Hz): ');

if Fs <= 0

error('Sampling Frequency (Fs) must be greater than 0.');

end

Ap = input('Enter Pass Band Attenuation (Ap) in dB: ');

As = input('Enter Stop Band Attenuation (As) in dB (> 40): ');

if As <= 40

error('Stop Band Attenuation must be greater than 40 dB.');

end

Fp = input('Enter Pass Band Frequency (Fp) in Hz: ');

Fs\_stop = input('Enter Stop Band Frequency (Fs) in Hz: ');

if Fp <= 0 || Fp >= Fs/2 || Fs\_stop <= 0 || Fs\_stop >= Fs/2 || Fp >= Fs\_stop

error('Frequencies must satisfy: 0 < Fp < Fs\_stop < Fs/2.');

end

N = input('Enter filter Order (N): ');

if mod(N, 2) ~= 0

warning('Filter order N is recommended to be even for symmetric FIR filters.');

end

% Frequency Sampling Method

f = (0:N) / N; % Normalized frequency points

H = zeros(size(f)); % Initialize Frequency Response

Wp = Fp / (Fs/2); % Normalize Pass Band Frequency

H(f <= Wp) = 10^(-Ap/20); % Pass Band Attenuation

H(f > Wp & f < 1) = 10^(-As/20); % Stop Band Attenuation

% Compute Linear Phase Impulse Response

h\_linear = real(ifft(H, 'symmetric'));

h\_linear = h\_linear(1:N+1);

% Frequency Response

[H\_resp\_linear, f\_resp] = freqz(h\_linear, 1, 1024, Fs);

% Plot Magnitude Spectrum

subplot(2, 1, 1);

plot(f\_resp, 20 \* log10(abs(H\_resp\_linear)), 'b', 'LineWidth', 1.5);

grid on;

title('Magnitude Spectrum (LPF - Linear Phase)');

xlabel('Frequency (Hz)');

ylabel('Magnitude (dB)');

ylim([-100 5]);

% Plot Phase Spectrum

subplot(2, 1, 2);

plot(f\_resp, angle(H\_resp\_linear) \* 180/pi, 'b', 'LineWidth', 1.5);

grid on;

title('Phase Spectrum (LPF - Linear Phase)');

xlabel('Frequency (Hz)');

ylabel('Phase (Degrees)');

[original\_signal, Fs\_audio] = audioread('prathamesh\_rec.wav');

if size(original\_signal, 2) > 1

original\_signal = mean(original\_signal, 2); % Convert to mono if stereo

end

% Ensure consistency in sampling frequency

Fs = Fs\_audio; % Use audio sampling frequency

% Truncate or loop signal to 12 seconds

duration = 12; % seconds

num\_samples = Fs \* duration;

if length(original\_signal) < num\_samples

error('The audio file must be at least 12 seconds long.');

else

original\_signal = original\_signal(1:num\_samples);

end

% Generate 7.8 kHz noise

t = (0:num\_samples-1) / Fs;

noise = 0.1 \* sin(2 \* pi \* 7800 \* t)'; % Adjust amplitude of noise as needed

% Add noise to the original signal

noisy\_signal = original\_signal + noise;

% Apply the filter to the noisy signal

filtered\_signal = filter(h\_linear, 1, noisy\_signal);

disp('Playing noisy signal...');

sound(noisy\_signal, Fs);

pause(duration);

disp('Playing filtered signal...');

sound(filtered\_signal, Fs);

pause(duration);

% Plot the signals

time\_axis = (0:num\_samples-1) / Fs;

figure;

subplot(3, 1, 1);

plot(time\_axis, original\_signal, 'b');

grid on;

title('Original Signal');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3, 1, 2);

plot(time\_axis, noisy\_signal, 'r');

grid on;

title('Noisy Signal (Original + 7.8 kHz Noise)');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3, 1, 3);

plot(time\_axis, filtered\_signal, 'g');

grid on;

title('Filtered Signal');

xlabel('Time (s)');

ylabel('Amplitude');

% Save the processed audio files (optional)

audiowrite('noisy\_signal.wav', noisy\_signal, Fs);

audiowrite('filtered\_signal.wav', filtered\_signal, Fs);