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);
```