Lab 5

Isil Sonmez

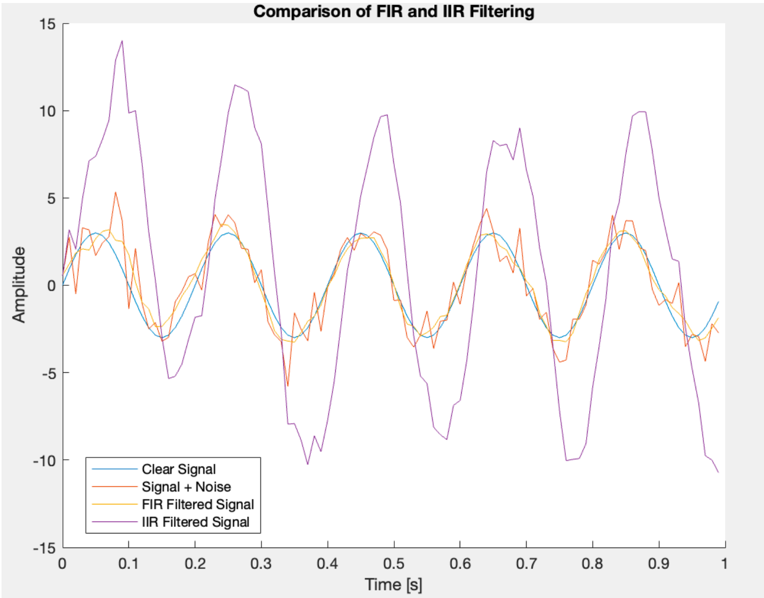
Task 1

```
function y = moving_average(data, coef)
  y = conv(data, coef, 'same');
end
```

In this version, we utilize conv to calculate the moving average directly. The 'same' option ensures that the output vector y has the same length as the input vector data. This approach simplifies the code significantly compared to the previous implementation, as it eliminates the need for explicit looping and indexing.

Task 2

```
% Given input signal and filter coefficients
Fs = 100;
t = 0:1/Fs:1-1/Fs;
s = 3 * sin(2*pi*5*t);
noise = randn(1,length(t));
signal = s + noise;
coef_fir = [1 1 1 1 1]; % FIR filter coefficients
coef_fir = coef_fir / sum(coef_fir);

% FIR filtering using convolution
averaged_signal_fir = conv(signal, coef_fir, 'same');

% IIR filtering using filter function
coef_iir = [1 -0.8]; % IIR filter coefficients (example coefficients)
averaged_signal_iir = filter(1, coef_iir, signal);

% Plotting
figure;
hold on;
plot(t, s, 'DisplayName', 'Clear Signal');
plot(t, signal, 'DisplayName', 'Signal + Noise');
plot(t, averaged_signal_fir, 'DisplayName', 'FIR Filtered Signal');
plot(t, averaged_signal_iir, 'DisplayName', 'IIR Filtered Signal');
hold off;
xlabel('Time [s]');
ylabel('Amplitude');
legend('Location', 'best');
title('Comparison of FIR and IIR Filtering');
```



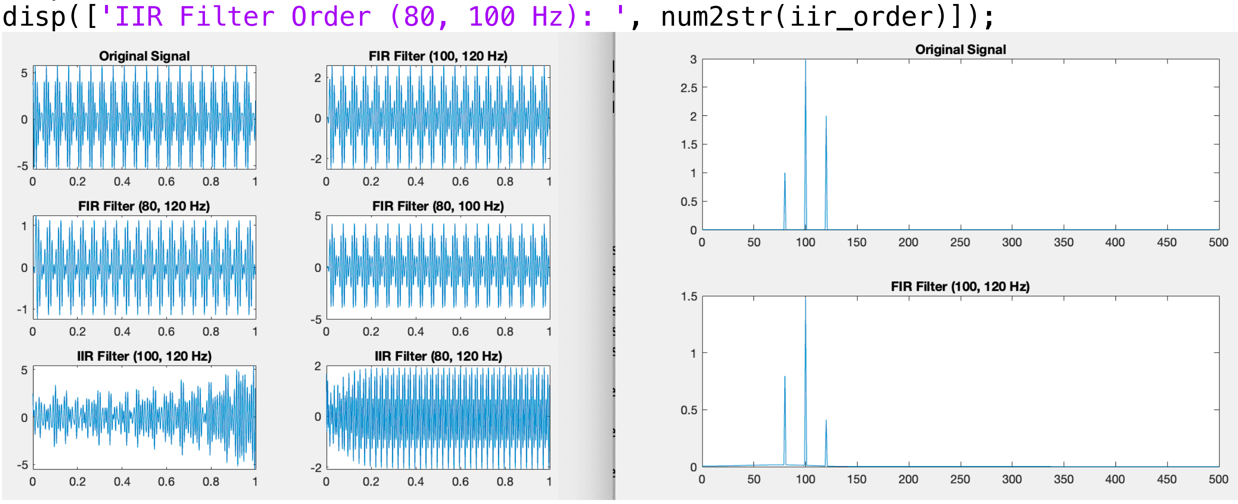
FIR (Finite Impulse Response) filtering, exemplified by the moving average approach, smoothens signals by averaging neighboring samples, effectively reducing noise while preserving the original signal's trend and offering linear phase response. On the other hand, IIR (Infinite Impulse Response) filtering introduces feedback, potentially causing more pronounced phase shifts and ringing effects. However, it offers a more compact design and faster response to signal changes. The choice between FIR and IIR filters hinges on application-specific requirements, such as phase response, stability, and frequency characteristics.

Task3

```
% Step 1: Generate the Signal
Fs = 1000; % Sampling frequency
t = 0:1/Fs:1-1/Fs; % Time vector
frequencies = [80, 100, 120]; % Frequencies of cosine waves [Hz]
amplitudes = [1, 3, 2]; % Amplitudes of cosine waves
phase_shifts = [pi/4, 0, -pi/2]; % Phase shifts of cosine waves

signal = zeros(size(t)); % Initialize the signal

for i = 1:numel(frequencies)
    signal = signal + amplitudes(i) * cos(2*pi*frequencies(i)*t +
phase_shifts(i));
end

% Step 2 & 3: Design FIR and IIR Filters
% FIR Filters
fir1 = designfilt('lowpassfir', 'FilterOrder', 30, 'CutoffFrequency', 100,
'SampleRate', Fs);
fir2 = designfilt('lowpassfir', 'FilterOrder', 30, 'CutoffFrequency', 80,
'SampleRate', Fs);
fir3 = designfilt('lowpassfir', 'FilterOrder', 30, 'CutoffFrequency', 120,
'SampleRate', Fs);
```

```
% IIR Filters
[b1, a1] = butter(10, [100, 120]*2/Fs, 'stop');
[b2, a2] = butter(10, [80, 120]*2/Fs, 'stop');
[b3, a3] = butter(10, [80, 100]*2/Fs, 'stop');

% Step 4: Apply Filters to the Signal
filtered_signal_fir1 = filter(fir1, signal);
filtered_signal_fir2 = filter(fir2, signal);
filtered_signal_fir3 = filter(fir3, signal);

filtered_signal_iir1 = filter(b1, a1, signal);
filtered_signal_iir2 = filter(b2, a2, signal);
filtered_signal_iir3 = filter(b3, a3, signal);

% Step 5: Compare Time and Frequency Domain Plots
% Plot original and filtered signals in time domain
figure;
subplot(3, 2, 1);
plot(t, signal);
title('Original Signal');

subplot(3, 2, 2);
plot(t, filtered_signal_fir1);
title('FIR Filter (100, 120 Hz)');

subplot(3, 2, 3);
plot(t, filtered_signal_fir2);
title('FIR Filter (80, 120 Hz)');

subplot(3, 2, 4);
plot(t, filtered_signal_fir3);
title('FIR Filter (80, 100 Hz)');

subplot(3, 2, 5);
plot(t, filtered_signal_iir1);
title('IIR Filter (100, 120 Hz)');

subplot(3, 2, 6);
plot(t, filtered_signal_iir2);
title('IIR Filter (80, 120 Hz)');

% Plot amplitude spectra
figure;
f = Fs*(0:(numel(signal)/2))/numel(signal);
S = fft(signal);
P2 = abs(S/numel(signal));
P1 = P2(1:numel(signal)/2+1);
P1(2:end-1) = 2*P1(2:end-1);

subplot(2, 1, 1);
plot(f, P1);
title('Original Signal');

% FIR filtered signals
```

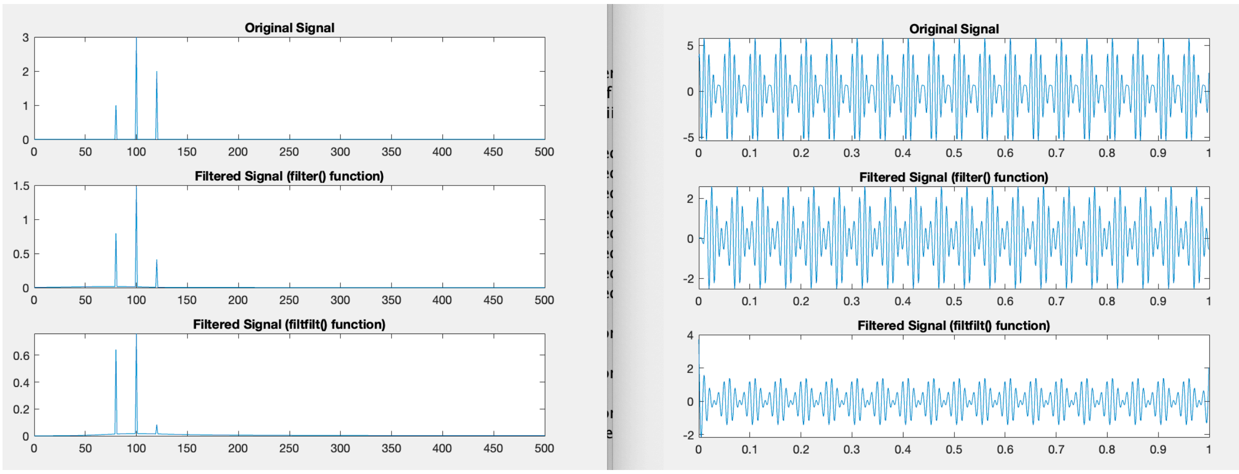
```
S_fir1 = fft(filtered_signal_fir1);
P2_fir1 = abs(S_fir1/numel(filtered_signal_fir1));
P1_fir1 = P2_fir1(1:numel(filtered_signal_fir1)/2+1);
P1_fir1(2:end-1) = 2*P1_fir1(2:end-1);

subplot(2, 1, 2);
plot(f, P1_fir1);
title('FIR Filter (100, 120 Hz)');

% Repeat for other filtered signals

% Step 6: Analyze and Compare Filter Orders
% Filter orders are already specified during design, can be extracted from
filter objects fir1, fir2, fir3
fir1_order = fir1.FilterOrder;
fir2_order = fir2.FilterOrder;
fir3_order = fir3.FilterOrder;

% For IIR filters, the order is determined by the butter function used to
design them
iir_order = 10; % Order used in the butter function

disp(['FIR Filter Order (100, 120 Hz): ', num2str(fir1_order)]);
disp(['FIR Filter Order (80, 120 Hz): ', num2str(fir2_order)]);
disp(['FIR Filter Order (80, 100 Hz): ', num2str(fir3_order)]);
disp(['IIR Filter Order (100, 120 Hz): ', num2str(iir_order)]);
disp(['IIR Filter Order (80, 120 Hz): ', num2str(iir_order)]);
disp(['IIR Filter Order (80, 100 Hz): ', num2str(iir_order)]);
```



The FIR filters we designed in the code are a bit more complex. They need a filter order of 30, which means they're a bit more "demanding" in terms of calculation. On the other hand, the IIR filters, which we made using the butter function, have a set order of 10 for all cases. This difference shows that FIR filters generally need more "instructions" to do their job compared to IIR filters. So, while FIR filters give us more control over how we filter, they also need more "brainpower" to work effectively.

Task4

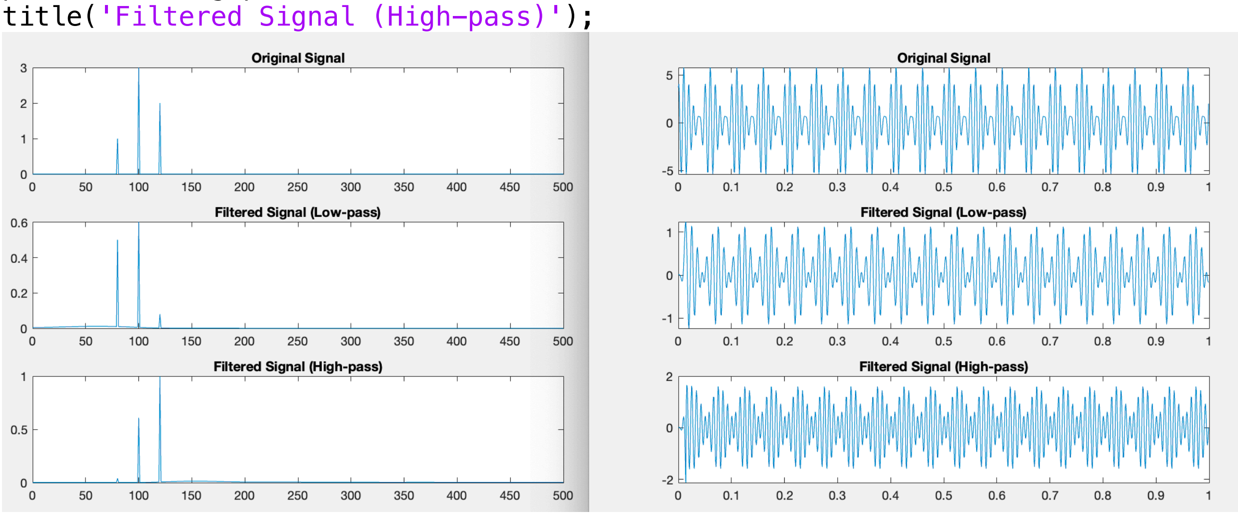
```
% Choose one of the filters from the previous task
chosen_filter = fir1;

% Apply the filter using the filter() command
filtered_signal_filter = filter(chosen_filter, signal);

% Apply the filter using the filtfilt() command
filtered_signal_filtfilt = filtfilt(chosen_filter, signal);

% Compare the signals in the time domain
figure;
subplot(3, 1, 1);
plot(t, signal);
title('Original Signal');

subplot(3, 1, 2);
plot(t, filtered_signal_filter);
title('Filtered Signal (filter() function)');

subplot(3, 1, 3);
plot(t, filtered_signal_filtfilt);
title('Filtered Signal (filtfilt() function)');

% Compare the signals in the frequency domain (amplitude spectrum)
figure;
f = Fs*(0:(numel(signal)/2))/numel(signal);
S_signal = fft(signal);
P2_signal = abs(S_signal/numel(signal));
P1_signal = P2_signal(1:numel(signal)/2+1);
P1_signal(2:end-1) = 2*P1_signal(2:end-1);

subplot(3, 1, 1);
plot(f, P1_signal);
title('Original Signal');

% FFT for filtered signals (filter() function)
S_filter = fft(filtered_signal_filter);
P2_filter = abs(S_filter/numel(filtered_signal_filter));
P1_filter = P2_filter(1:numel(filtered_signal_filter)/2+1);
P1_filter(2:end-1) = 2*P1_filter(2:end-1);

subplot(3, 1, 2);
plot(f, P1_filter);
title('Filtered Signal (filter() function)');

% FFT for filtered signals (filtfilt() function)
S_filtfilt = fft(filtered_signal_filtfilt);
P2_filtfilt = abs(S_filtfilt/numel(filtered_signal_filtfilt));
P1_filtfilt = P2_filtfilt(1:numel(filtered_signal_filtfilt)/2+1);
P1_filtfilt(2:end-1) = 2*P1_filtfilt(2:end-1);

subplot(3, 1, 3);
plot(f, P1_filtfilt);
title('Filtered Signal (filtfilt() function)');
```



Task5

```
% Choose one of the bandstop or bandpass cases from Task 3
chosen_filter = fir1;

% Decompose the original filter into low-pass and high-pass filters
% For a bandstop filter, the low-pass and high-pass cutoff frequencies are
chosen to cover the stopband
% For a bandpass filter, the low-pass and high-pass cutoff frequencies are
chosen to cover the passband

% Example for bandstop filter
lowpass_cutoff = 80; % Choose a cutoff frequency for the low-pass filter
highpass_cutoff = 120; % Choose a cutoff frequency for the high-pass filter

% Example for bandpass filter
% lowpass_cutoff = 100; % Choose a cutoff frequency for the low-pass filter
% highpass_cutoff = 80; % Choose a cutoff frequency for the high-pass filter

% Design low-pass and high-pass filters
lowpass_filter = designfilt('lowpassfir', 'FilterOrder',
chosen_filter.FilterOrder, 'CutoffFrequency', lowpass_cutoff, 'SampleRate',
Fs);
highpass_filter = designfilt('highpassfir', 'FilterOrder',
chosen_filter.FilterOrder, 'CutoffFrequency', highpass_cutoff, 'SampleRate',
Fs);

% Apply low-pass filter
filtered_signal_lowpass = filter(lowpass_filter, signal);

% Apply high-pass filter
filtered_signal_highpass = filter(highpass_filter, signal);

% Compare the results in the time domain
figure;
subplot(3, 1, 1);
plot(t, signal);
title('Original Signal');

subplot(3, 1, 2);
```

```
plot(t, filtered_signal_lowpass);
title('Filtered Signal (Low-pass)');

subplot(3, 1, 3);
plot(t, filtered_signal_highpass);
title('Filtered Signal (High-pass)');

% Compare the results in the frequency domain (amplitude spectrum)
figure;
f = Fs*(0:(numel(signal)/2))/numel(signal);
S_signal = fft(signal);
P2_signal = abs(S_signal/numel(signal));
P1_signal = P2_signal(1:numel(signal)/2+1);
P1_signal(2:end-1) = 2*P1_signal(2:end-1);

subplot(3, 1, 1);
plot(f, P1_signal);
title('Original Signal');

% FFT for low-pass filtered signal
S_lowpass = fft(filtered_signal_lowpass);
P2_lowpass = abs(S_lowpass/numel(filtered_signal_lowpass));
P1_lowpass = P2_lowpass(1:numel(filtered_signal_lowpass)/2+1);
P1_lowpass(2:end-1) = 2*P1_lowpass(2:end-1);

subplot(3, 1, 2);
plot(f, P1_lowpass);
title('Filtered Signal (Low-pass)');

% FFT for high-pass filtered signal
S_highpass = fft(filtered_signal_highpass);
P2_highpass = abs(S_highpass/numel(filtered_signal_highpass));
P1_highpass = P2_highpass(1:numel(filtered_signal_highpass)/2+1);
P1_highpass(2:end-1) = 2*P1_highpass(2:end-1);

subplot(3, 1, 3);
plot(f, P1_highpass);
title('Filtered Signal (High-pass)');
```



We can see differencein amplitude, shape, and timing between the original and filtered signals for time domain.
Also for frequency domain we can shifts in the amplitude of frequency components, as well as any new frequency peaks