Digital Signal Processing Lab

Experiment 2 – Designing Low Pass Filters by Windowing Method

Utkarsh Patel (18EC30048)

Objective

- Designing FIR filters for various orders and cut-off frequencies
- Observing stop-band attenuation by the designed filter
- Observing response of designed FIR filters to the inputs contaminated by the noise

Theory

- Ideal filters having sharp cut-off are not realizable in practice since their impulse responses extend up to infinity.
- In order to realize filters practically, we need to truncate the impulse response after a few samples. Due to this, many undesired features get introduced in the frequency domain.
- By windowing method, the truncated filter is modified to satisfy certain frequency domain characteristics.
- Let $H_d(\omega)$ be the frequency domain representation of ideal low pass filter (which can't be realized practically) and $W(\omega)$ be the frequency domain representation of the window function used, then the frequency domain representation of designed filter is given as

$$H(\omega) = H_d(\omega)W(\omega)$$

Thus, the impulse response of the designed filter is given as

$$h(n) = h_d(n) * w(n)$$

- In this experiment, five window functions have been used in designing the filter.

Window name	***Window Function***
Rectangular	$w_N(n) = \begin{cases} 1, & 0 \le n < N \\ 0, & otherwise \end{cases}$
Triangular	$w_N(n) = \begin{cases} 1 - 2\dfrac{n - \frac{N-1}{2}}{N-1}, & 0 \le n < N \\ 0, & otherwise \end{cases}$
Hanning	$w_N(n) = \begin{cases} 0.5 - 0.5\cos\dfrac{2\pi n}{N-1}, & 0 \le n < N \\ 0, & otherwise \end{cases}$
Hamming	$w_N(n) = \begin{cases} 0.54 - 0.46\cos\dfrac{2\pi n}{N-1}, & 0 \le n < N \\ 0, & otherwise \end{cases}$
Blackman	$w_N(n) = \begin{cases} 0.42 - 0.5\cos\dfrac{2\pi n}{N-1} + 0.08\cos\dfrac{4\pi n}{N-1}, & 0 \le n < N \\ 0, & otherwise \end{cases}$

Results

PART A

Using different window functions, low pass filters are designed and their frequency response is plotted. Different window length $N = 8$, 64 and 512 are used and the response is observed. For each plot, approximate transition width of main lobe, peak of first side lobe and maximum stop band attenuation is computed.

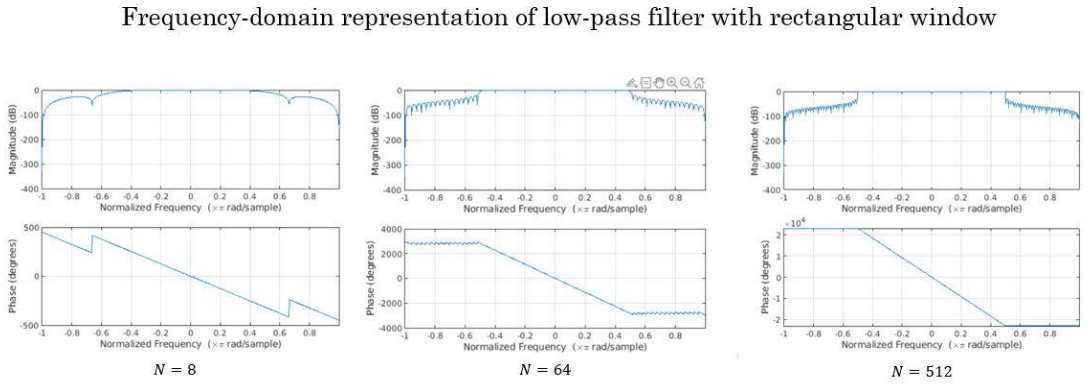

Fig. 1. Frequency-domain representation of low-pass filter with rectangular window

Table 1. *Transition width, first side lobe peak and max. stop attenuation for rectangular window*

Window Length	***Transition Width***	***Peak of 1st Side Lobe***	***Max. Stopband Attenuation***
8	0.19 π	-26.54 dB	170 dB
64	0.021 π	-27.23 dB	331.15 dB
512	0.006 π	-29.1 dB	352.19 dB

- As we can observe that as the window length increases, the filter becomes more and more similar to the ideal low pass filter.
- As the length of window increases, the width of main lobe decreases, thus the transition width between passband and stopband decreases.
- The peak of first side lobe decreases with increase in window length.
- Stopband attenuation increases with increase in window length.

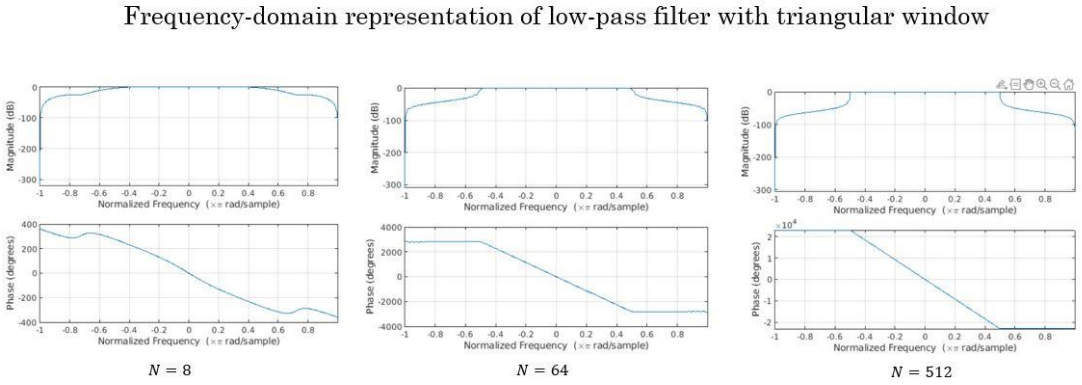

Fig. 2. Frequency-domain representation of low-pass filter with triangular window

Table 2. *Transition width, first side lobe peak and max. stop attenuation for triangular window*

Window Length	***Transition Width***	***Peak of 1st Side Lobe***	***Max. Stopband Attenuation***
8	0.23 π	-26.77 dB	121.48 dB
64	0.05 π	-27.84 dB	122.67 dB
512	0.04 π	-29.64 dB	123.74 dB

- As we can observe that as the window length increases, the filter becomes more and more similar to the ideal low pass filter.
- As the length of window increases, the width of main lobe decreases, thus the transition width between passband and stopband decreases.
- The peak of first side lobe decreases with increase in window length.
- Stopband attenuation increases with increase in window length.

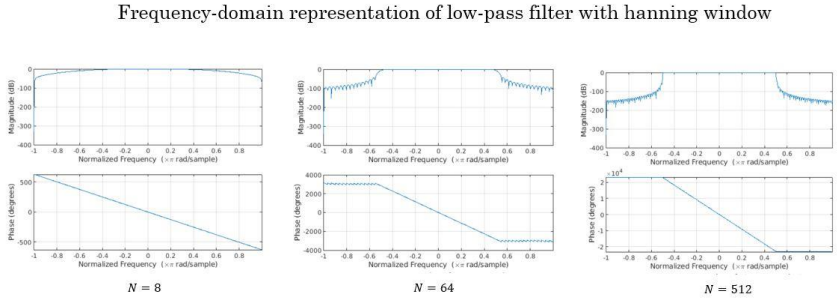

Fig. 3. Frequency-domain representation of low-pass filter with hanning window

Table 3. *Transition width, first side lobe peak and max. stop attenuation for hanning window*

Window Length	***Transition Width***	***Peak of 1st Side Lobe***	***Max. Stopband Attenuation***
8	0.57 π	-43.1 dB	85.23 dB
64	0.05 π	-44.2 dB	124.33 dB
512	0.01 π	-66.24 dB	172.82 dB

- As we can observe that as the window length increases, the filter becomes more and more similar to the ideal low pass filter.
- As the length of window increases, the width of main lobe decreases, thus the transition width between passband and stopband decreases.
- The peak of first side lobe decreases with increase in window length.
- Stopband attenuation increases with increase in window length.

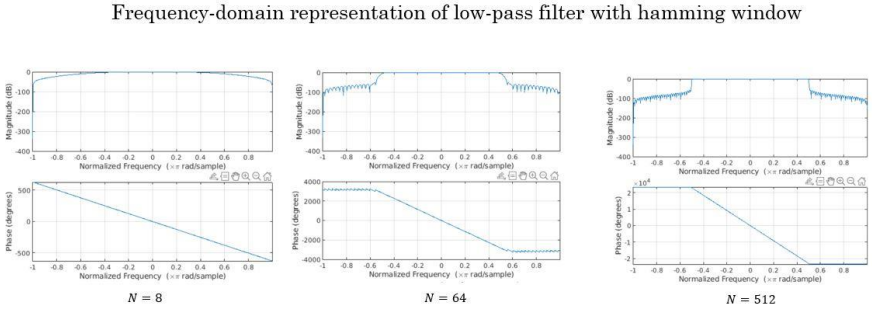

Fig. 4. Frequency-domain representation of low-pass filter with hamming window

Table 4. *Transition width, first side lobe peak and max. stop attenuation for hamming window*

Window Length	***Transition Width***	***Peak of 1st Side Lobe***	***Max. Stopband Attenuation***
8	0.56 π	-45.21 dB	85.91 dB
64	0.062 π	-52.1 dB	125.3 dB
512	0.015 π	-59.58 dB	162.34 dB

- As we can observe that as the window length increases, the filter becomes more and more similar to the ideal low pass filter.
- As the length of window increases, the width of main lobe decreases, thus the transition width between passband and stopband decreases.
- The peak of first side lobe decreases with increase in window length.
- Stopband attenuation increases with increase in window length.

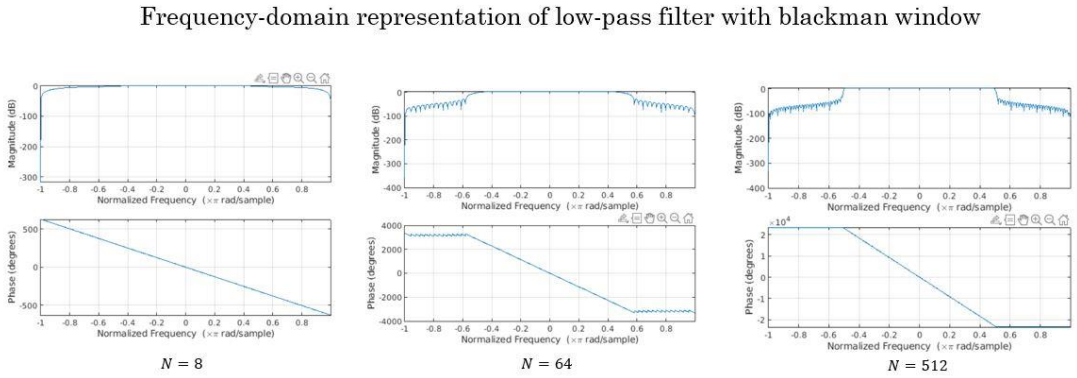

Fig. 5. Frequency-domain representation of low-pass filter with blackman window

Table 5. *Transition width, first side lobe peak and max. stop attenuation for blackman window*

Window Length	***Transition Width***	***Peak of 1st Side Lobe***	***Max. Stopband Attenuation***
8	π	NIL	62 dB
64	0.06 π	-29.84 dB	110.18 dB
512	0.003 π	-36.41 dB	135.14 dB

- As we can observe that as the window length increases, the filter becomes more and more similar to the ideal low pass filter.
- As the length of window increases, the width of main lobe decreases, thus the transition width between passband and stopband decreases.
- The peak of first side lobe decreases with increase in window length.
- Stopband attenuation increases with increase in window length.

PART B

Testing the performance of the filter on an input signal with one frequency in passband while the other in stopband. The filter is designed with $F_s = 10$ KHz and with $\omega_c = 2.5$ KHz. The passband frequency is 2 KHz and the stopband frequency is 4 KHz.

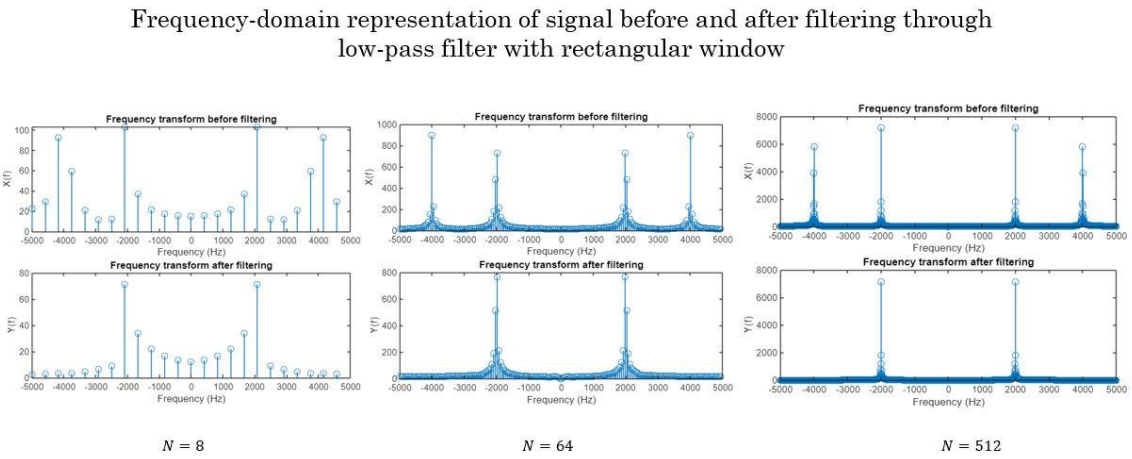

Fig. 6. Frequency-domain representation of signal before and after filtering through low-pass filter with rectangular window

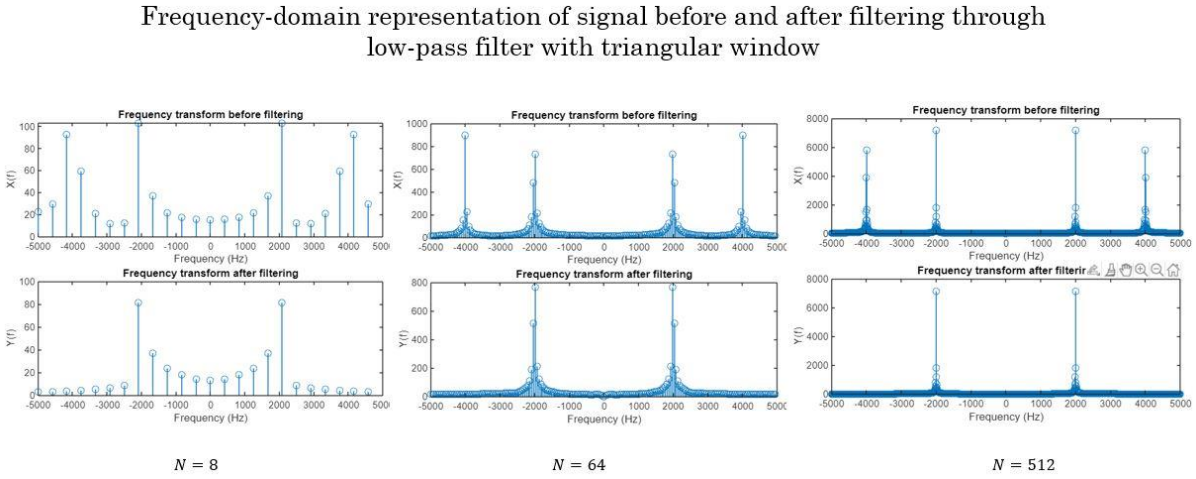

Fig. 7. Frequency-domain representation of signal before and after filtering through low-pass filter with triangular window

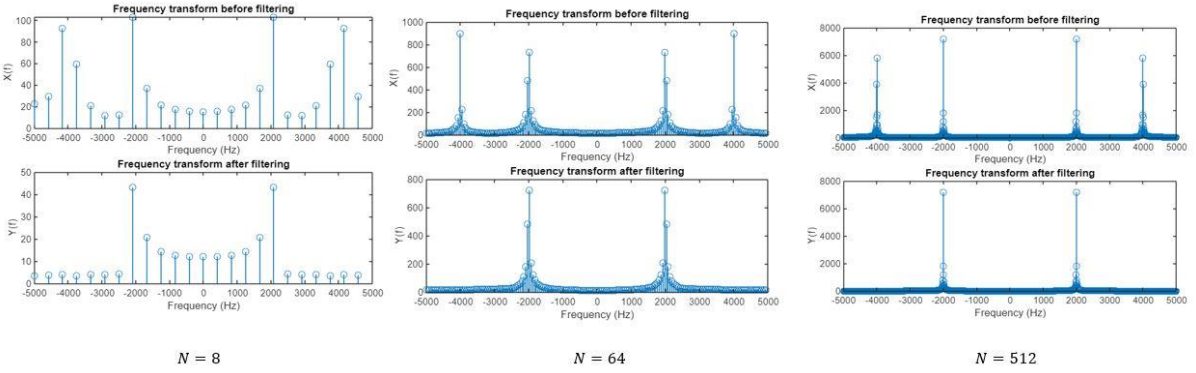

Fig. 8. Frequency-domain representation of signal before and after filtering through low-pass filter with hanning window

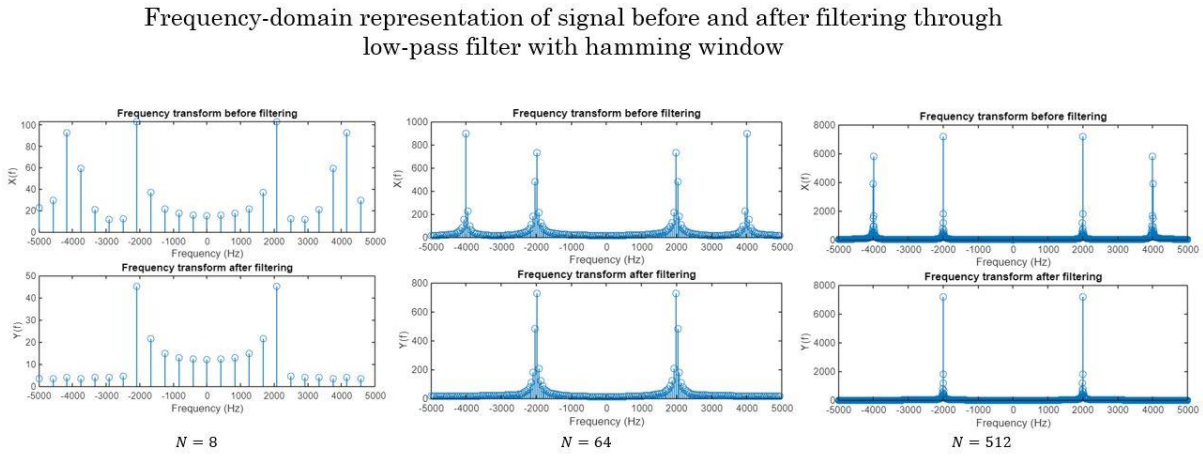

Fig. 9. Frequency-domain representation of signal before and after filtering through low-pass filter with hamming window

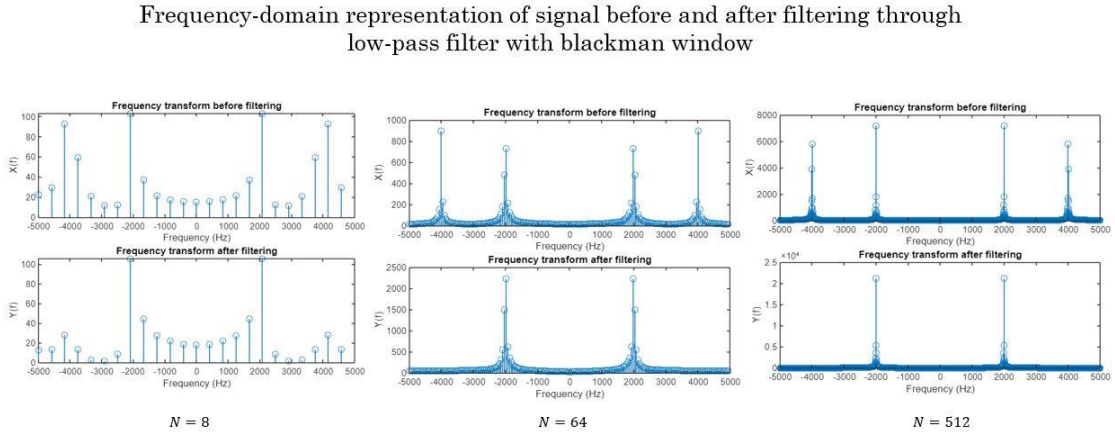

Fig. 10. Frequency-domain representation of signal before and after filtering through low-pass filter with blackman window

PART C

Adding white noise to the signal and testing the performance of different filters on the corrupted signal and computing the SNR value.

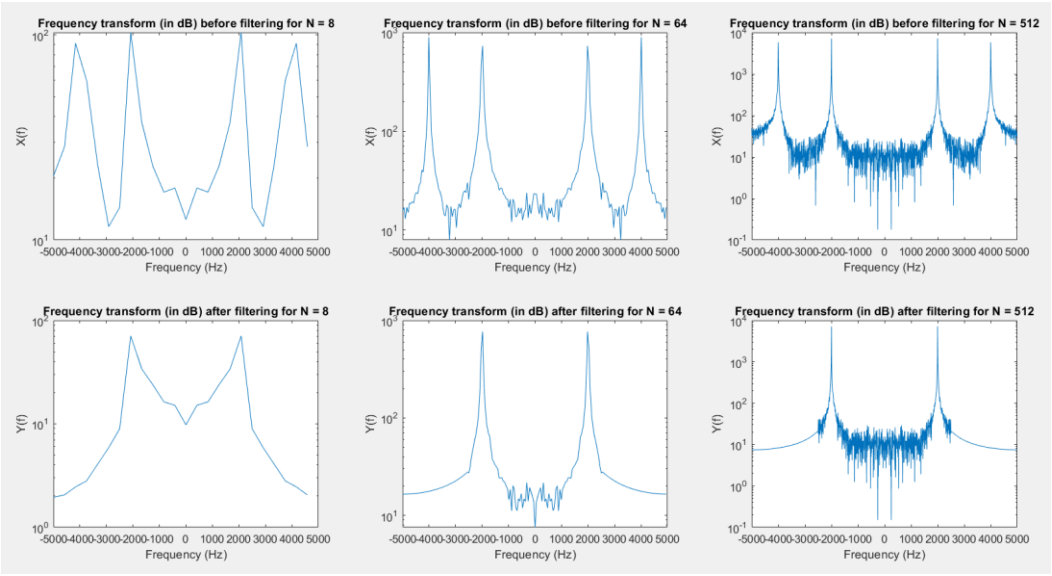

Fig. 11. Frequency-domain representation (in dB) of corrupted signal before and after filtering through low-pass filter with rectangular window

Table 6. *SNR for rectangular window*

Window Length	***Signal Amplitude*** (in dB)	***Noise Amplitude*** (in dB)	***SNR*** (in dB)
8	14.17	-11.37	25.54
64	17.51	-13.31	30.82
512	16.97	-12.64	29.61

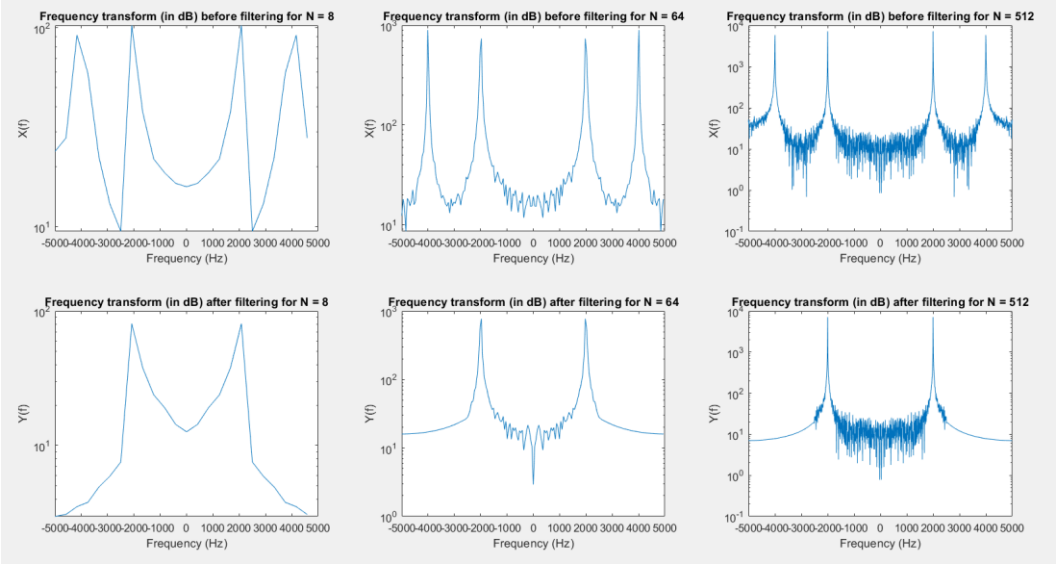

Fig. 12. Frequency-domain representation (in dB) of corrupted signal before and after filtering through low-pass filter with triangular window

Table 7. *SNR for triangular window*

Window Length	***Signal Amplitude*** (in dB)	***Noise Amplitude*** (in dB)	***SNR*** (in dB)
8	15.16	-13.45	28.60
64	17.52	-12.93	30.45
512	16.97	-13.18	30.15

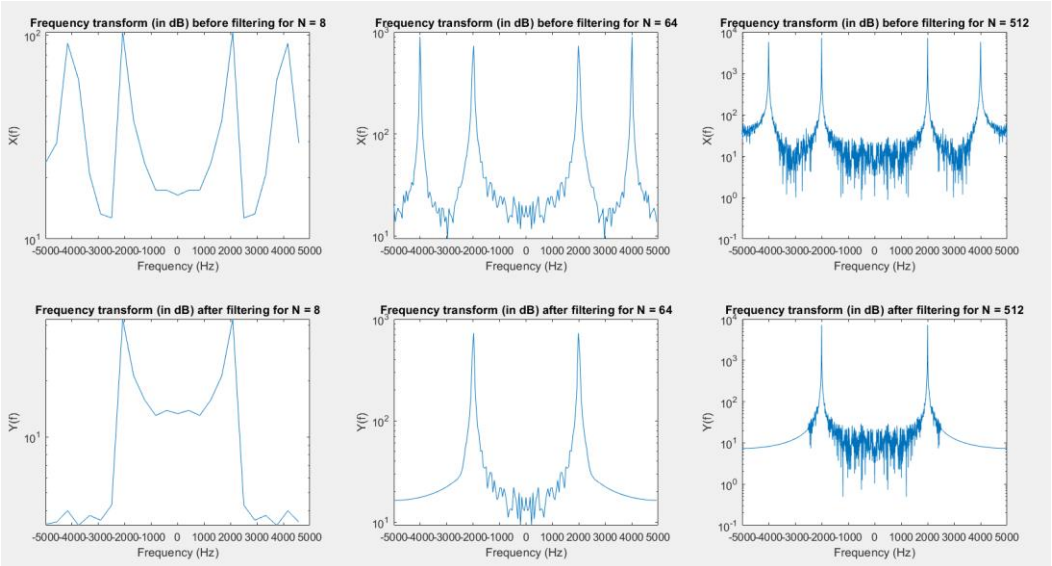

Fig. 13. Frequency-domain representation (in dB) of corrupted signal before and after filtering through low-pass filter with hanning window

Table 8. *SNR for hanning window*

Window Length	***Signal Amplitude*** (in dB)	***Noise Amplitude*** (in dB)	***SNR*** (in dB)
8	10.34	-15.89	26.23
64	17.05	-12.70	29.65
512	17.00	-12.97	29.97

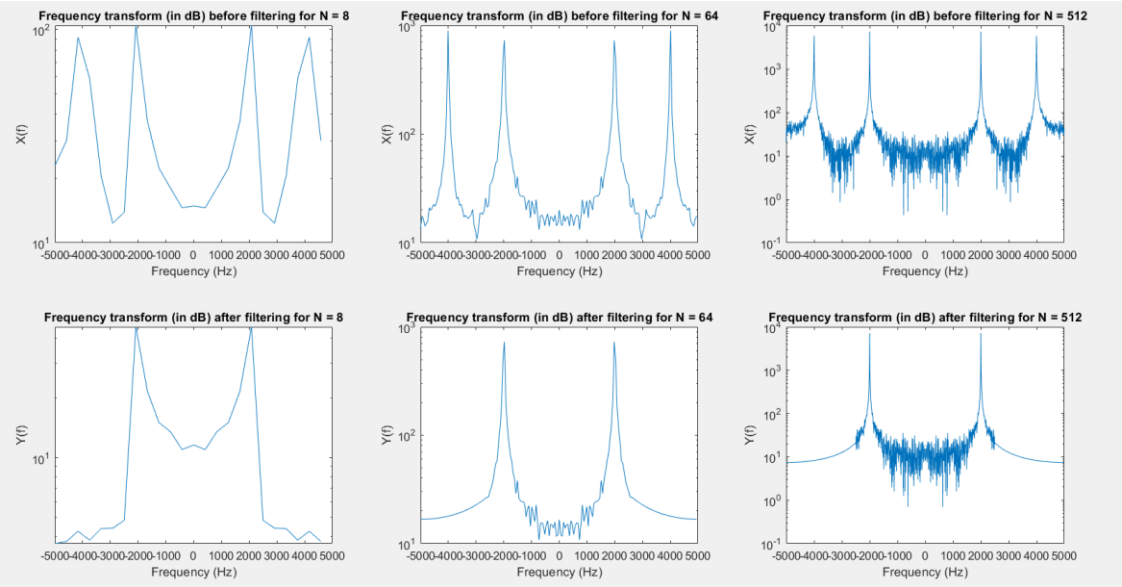

Fig. 14. Frequency-domain representation (in dB) of corrupted signal before and after filtering through low-pass filter with hamming window

Table 9. *SNR for hamming window*

Window Length	***Signal Amplitude*** (in dB)	***Noise Amplitude*** (in dB)	***SNR*** (in dB)
8	10.55	-17.00	27.54
64	17.08	-14.00	31.08
512	17.00	-12.68	29.68

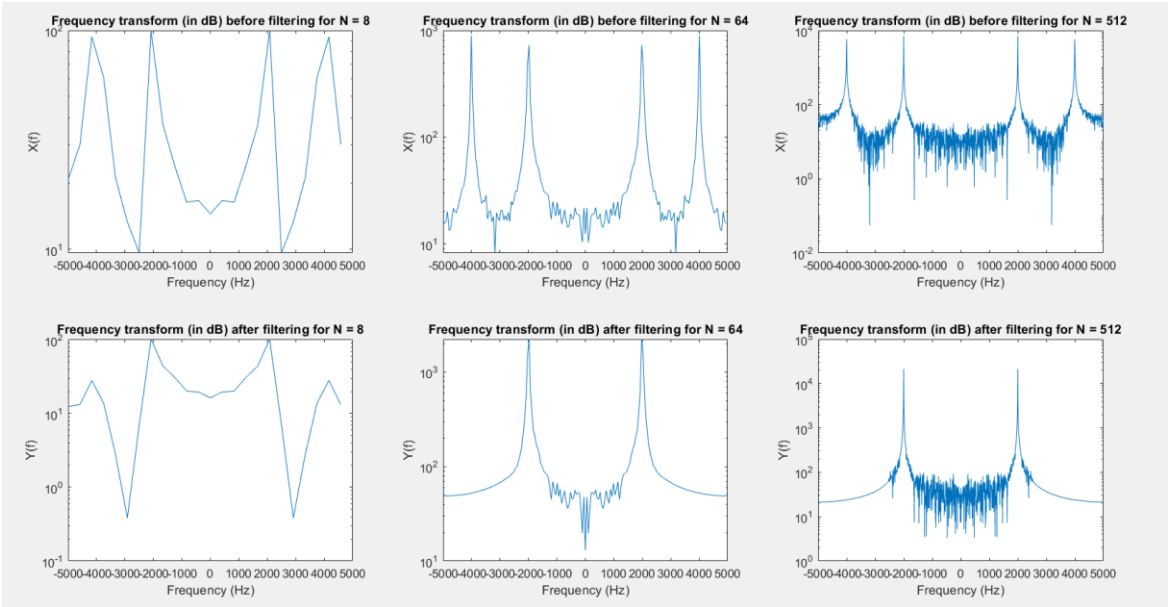

Fig. 15. Frequency-domain representation (in dB) of corrupted signal before and after filtering through low-pass filter with blackman window

Table 10. *SNR for blackman window*

Window Length	***Signal Amplitude*** (in dB)	***Noise Amplitude*** (in dB)	***SNR*** (in dB)
8	17.32	-8.09	25.41
64	26.69	-3.85	30.55
512	26.42	-3.54	29.96

Discussion

- Ideal filters having sharp cut-off are not practically realizable since their impulse response extends up to infinity. In order to realize filters practically, the impulse response is truncated after a few samples. Due to this abrupt transition, many undesired features get introduced in the frequency domain. By windowing method, the truncated filter is modified to satisfy certain frequency domain requirements and perform its task well.
- How well the window is working is mainly depends by the width of the main lobe and the peak of the side lobe.
- Ideally, the main lobe width should be narrow and the side lobe amplitude should be small. However, for a fixed length window, these cannot be minimized independently.
- As the length of the window is increased, the width of the main lobe decreases which results in a decrease in the transition width between passband and stopband. The relation is given as $N\delta_f = c$, where δ_f is the transition width and c is a parameter that depends on the window.
- The peak side lobe amplitude of the window is determined by the shape of window and theoretically it should be independent of the window length, but in some cases, we have observed that as the length of the window increases, the peak decreases.
- We observe that as the window length increases the maximum attenuation increases this is because the filter becomes more similar to the ideal filter response.
- If the window shape is changed to decrease the sidelobe amplitude , the width of the main lobe will generally increase.
- As the size of the window increases then the transition width decreases which is more like the ideal low pass filter and there is a steep drop near the cut-off frequency.
- On passing a signal with two frequency components (one less than the cut-off frequency and the other greater than cut-off frequency) through the designed filters we can see that the greater frequency component gets filtered.
- On adding noise to the signal we can observe that some more frequencies are present in the filtered signal however a lot of noise gets filtered.
- The performance of the filter becomes better on increasing the window length and more noise frequencies gets filtered.
- The noise amplitude was chosen such that the SNR should be 30 dB.
- However, as it can be observed through the plots, as the window length is increased, the observed SNR converges to 30 dB.

Appendix

MATLAB Codes

Code for Figure 1

```
% Author: Utkarsh Patel
% Experiment 2: Part 1.1
% Analysing frequency response of LPF constructed rectangular window

wc = pi / 2;           % Cut-off frequency of the FIR filter
w = -1 * pi: 0.01: pi; % frequency range

N = 8;                 % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Rectangular window with N = " + string(N));
freqz(hn, 1, w);

N = 64;                 % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Rectangular window with N = " + string(N));
freqz(hn, 1, w);

N = 512;                % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Rectangular window with N = " + string(N));
freqz(hn, 1, w);

function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate rectangular window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 1;
    end
end
```

Code for Figure 2

```
% Author: Utkarsh Patel
% Experiment 2: Part 1.2
% Analysing frequency response of LPF constructed triangular window

wc = pi / 2;           % Cut-off frequency of the FIR filter
w = -1 * pi: 0.01: pi; % frequency range

N = 8;                 % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N, K);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Triangular window with N = " + string(N));
freqz(hn, 1, w);

N = 64;                % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N, K);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Triangular window with N = " + string(N));
freqz(hn, 1, w);

N = 512;               % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N, K);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Triangular window with N = " + string(N));
freqz(hn, 1, w);

function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N, K)
    % Generate triangular window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 1 - (2 * ((i - K) / (N - 1)));
    end
end
```

Code for Figure 3

```
% Author: Utkarsh Patel
% Experiment 2: Part 1.3
% Analysing frequency response of LPF constructed by hanning window

wc = pi / 2;           % Cut-off frequency of the FIR filter
w = -1 * pi: 0.01: pi; % frequency range

N = 8;                 % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Hanning window with N = " + string(N));
freqz(hn, 1, w);

N = 64;                 % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Hanning window with N = " + string(N));
freqz(hn, 1, w);

N = 512;                % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Hanning window with N = " + string(N));
freqz(hn, 1, w);

function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate hanning window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 0.5 - (0.5 * cos((2 * pi * i) / (N - 1)));
    end
end
```

Code for Figure 4

```
% Author: Utkarsh Patel
% Experiment 2: Part 1.4
% Analysing frequency response of LPF constructed by hamming window

wc = pi / 2;           % Cut-off frequency of the FIR filter
w = -1 * pi: 0.01: pi; % frequency range

N = 8;                 % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Hamming window with N = " + string(N));
freqz(hn, 1, w);

N = 64;                 % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Hamming window with N = " + string(N));
freqz(hn, 1, w);

N = 512;                % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Hamming window with N = " + string(N));
freqz(hn, 1, w);

function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate hamming window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 0.54 - (0.46 * cos((2 * pi * i) / (N - 1)));
    end
end
```

Code for Figure 5

```
% Author: Utkarsh Patel
% Experiment 2: Part 1.5
% Analysing frequency response of LPF constructed by blackmnan window

wc = pi / 2;           % Cut-off frequency of the FIR filter
w = -1 * pi: 0.01: pi; % frequency range

N = 8;                 % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Blackmnan window with N = " + string(N));
freqz(hn, 1, w);

N = 64;                 % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Blackmnan window with N = " + string(N));
freqz(hn, 1, w);

N = 512;                % window length
K = (N - 1) / 2;

% truncated version of time-domain representation of
% ideal low-pass filter
hd = sincf(N, K, wc);
win = mywin(N);     % my window
hn = hd.* win;         % pratical low-pass filter designed
figure("Name", "Blackmnan window with N = " + string(N));
freqz(hn, 1, w);

function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate black window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 0.42 - (0.5 * cos((2 * pi * i) / (N - 1))) + (0.8 * cos((4 * pi * i) / (N - 1)));
    end
end
```

Code for Figure 6

```
N = 8;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 1);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 4);
stem(f, abs(y_f));
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 64;                 % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 2);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 5);
stem(f, abs(y_f));
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 512;                % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 3);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 6);
stem(f, abs(y_f));
```

```
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');


function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate rectangular window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 1;
    end
end
```

Code for Figure 7

```
N = 8;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 1);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 4);
stem(f, abs(y_f));
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 64;                 % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 2);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 5);
stem(f, abs(y_f));
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 512;                % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 3);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 6);
stem(f, abs(y_f));
```

```
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');


function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate triangular window
    win = zeros(1, N);
    K = (N - 1) / 2;
    for i = 0: N - 1
        win(i + 1) = 1 - (2 * ((i - K) / (N - 1)));
    end
end
```

Code for Figure 8

```
N = 8;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 1);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 4);
stem(f, abs(y_f));
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 64;                 % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 2);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 5);
stem(f, abs(y_f));
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 512;                % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 3);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 6);
stem(f, abs(y_f));
```

```
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');


function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate hanning window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 0.5 - (0.5 * cos((2 * pi * i) / (N - 1)));
    end
end
```

Code for Figure 9

```
N = 8;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 1);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 4);
stem(f, abs(y_f));
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 64;                 % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 2);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 5);
stem(f, abs(y_f));
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 512;                % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 3);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 6);
stem(f, abs(y_f));
```

```
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');


function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate hamming window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 0.54 - (0.46 * cos((2 * pi * i) / (N - 1)));
    end
end
```

Code for Figure 10

```
N = 8;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 1);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 4);
stem(f, abs(y_f));
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 64;                 % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 2);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 5);
stem(f, abs(y_f));
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 512;                % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 3);
stem(f, abs(x_f));
title('Frequency transform before filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 6);
stem(f, abs(y_f));
```

```
title('Frequency transform after filtering, N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');


function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate black window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 0.42 - (0.5 * cos((2 * pi * i) / (N - 1))) + (0.8 * cos((4 * pi * i) / (N - 1)));
    end
end
```

Code for Figure 11

```
N = 8;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal



subplot(2, 3, 1);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 4);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 64;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 2);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 5);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 512;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
```

```
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 3);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 6);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');


function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate rectangular window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 1;
    end
end
```

Code for Figure 12

```
N = 8;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal



subplot(2, 3, 1);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 4);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 64;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 2);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 5);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 512;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
```

```
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 3);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 6);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');


function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate triangular window
    win = zeros(1, N);
    K = (N - 1) / 2;
    for i = 0: N - 1
        win(i + 1) = 1 - (2 * ((i - K) / (N - 1)));
    end
end
```

Code for Figure 13

```
N = 8;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal



subplot(2, 3, 1);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 4);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 64;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 2);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 5);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 512;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
```

```
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 3);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 6);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');


function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate hanning window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 0.5 - (0.5 * cos((2 * pi * i) / (N - 1)));
    end
end
```

Code for Figure 14

```
N = 8;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal



subplot(2, 3, 1);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 4);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 64;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 2);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 5);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 512;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
```

```
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 3);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 6);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');


function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate hamming window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 0.54 - (0.46 * cos((2 * pi * i) / (N - 1)));
    end
end
```

Code for Figure 15

```
N = 8;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal



subplot(2, 3, 1);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 4);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 64;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 2);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 5);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');

N = 512;                  % window length
K = (N - 1) / 2;
Fs = 10000;             % sampling frequency
t = 0: 1 / Fs: (3 * N - 1) / Fs;
wc = pi / 2;            % this implies cut-off freq is Fs / 4
w1 = 2000;              % passband frequency
w2 = 4000;              % stopband frequency
x1 = [10 10] * sin(2 * pi * [w1 w2]' * t); % constructed signal
```

```
SNR = 30;
noizz = randn(size(x1)) * std(x1) / db2mag(SNR);
x = x1 + noizz;
hd = sincf(N, K, wc);   % desired time-domain LPF
win = mywin(N);         % my window
hn = hd.*win;           % practical FIR
f = (-(3 * N) / 2: ((3 * N) / 2 - 1)) * (Fs / (3 * N)); % frequency range for plot
x_f = fftshift(fft(x)); % DFT of constructed signal
y = filtfilt(hn, 1, x); % signal after filtering
z = filtfilt(hn, 1, noizz); % noise after filtering
SNR_ = snr(y, z); % SNR observed
SIG_AMP = 20 * log10(rms(y)); % amplitude of filtered signal
disp('Signal amp: ' + string(SIG_AMP) + ', Noise amp: ' + string(SIG_AMP - SNR_) + ', SNR : ' + string(SNR_));
y_f = fftshift(fft(y)); % DFT of filtered signal

subplot(2, 3, 3);
semilogy(f, abs(x_f));
title('Frequency transform (in dB) before filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('X(f)');
subplot(2, 3, 6);
semilogy(f, abs(y_f));
title('Frequency transform (in dB) after filtering for N = ' + string(N));
xticks(floor(-Fs / 2000) * 1000 : 1000 : ceil(Fs / 2000) * 1000 );
xlabel('Frequency (Hz)');
ylabel('Y(f)');


function hd = sincf(N, K, wc)
    % Generates truncated version of time-domain representation
    % of ideal low-pass filter
    hd = zeros(1, N);
    for i = 0: N - 1
        hd(i + 1) = sin(wc * (i - K)) / (pi * (i - K));
    end
end

function win = mywin(N)
    % Generate black window
    win = zeros(1, N);
    for i = 0: N - 1
        win(i + 1) = 0.42 - (0.5 * cos((2 * pi * i) / (N - 1))) + (0.8 * cos((4 * pi * i) / (N - 1)));
    end
end
```