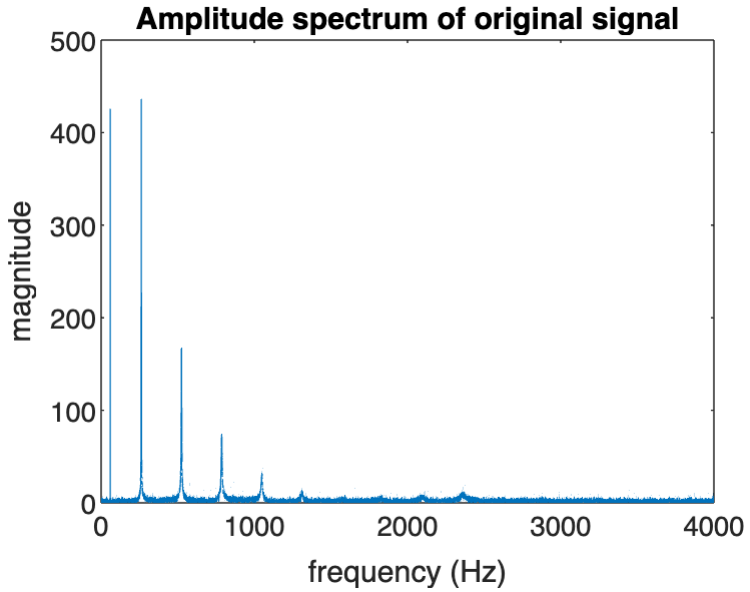
Item 4

```
close all; clear all; clc;

%load and read the audio file
[sig, Fs] = audioread('note_noise.wav');

% Compute the Fourier transform of the audio signal
N = length(sig); %length of the signal
Y = fft(sig); %Compute the Fourier transform
f = (0:N-1)*(Fs/N); %frequency axis

%plot the magnitude spectrum
plot(f(1:N/2), abs(Y(1:N/2)));
%In the plot, only half the length is used to avoid mirroring
%of the spikes. Only the positive frequencies of the
%Fourier transform are considered.
title('Amplitude spectrum of original signal');
xlabel('frequency (Hz)');
ylabel('magnitude');
```




```
% The fundamental freqeuency of the synthesized note is 262 Hz.
% A bandpass filter is needed to remove the noise on both sides of
% the fundamental frequency. The peak of the 262 Hz signal is 436.
% 20log(436) = 53
% A Hamming window will be used since it provides a stopband attenuation of
53 dB.
```

FIR filter

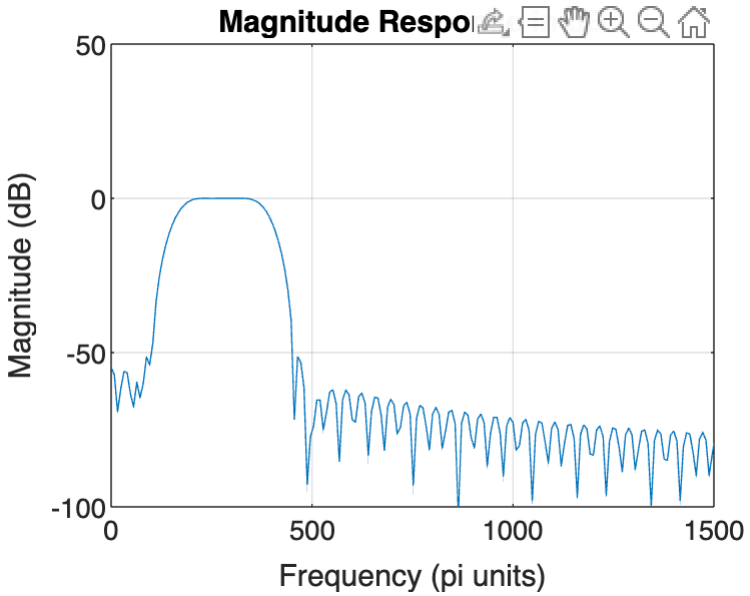
```
% Define the frequency values
ws1 = 60 * 2 * pi / Fs;
wp1 = 262 * 2 * pi / Fs;
wp2 = 262 * 2 * pi / Fs;
ws2 = 526 * 2 * pi / Fs;
As = 50; % minimum stopband attenuation in dB
Rp = 2; % maximum passband ripple in dB

% Compute the minimum transition width between stopband and passband
frequencies
tr_width = min((wp1-ws1), (ws2-wp2));

% Compute the filter order (M) by taking 11 times the ratio of pi to the
transition width (tr_width),
% rounding up to the nearest odd integer
M = ceil(11 * pi / tr_width) + 1; % filter order
disp('The lowest order FIR filter that will meet the desired specifications
is a Hamming window of order:');
```

```
The lowest order FIR filter that will meet the desired specifications is a Hamming window of order:
```

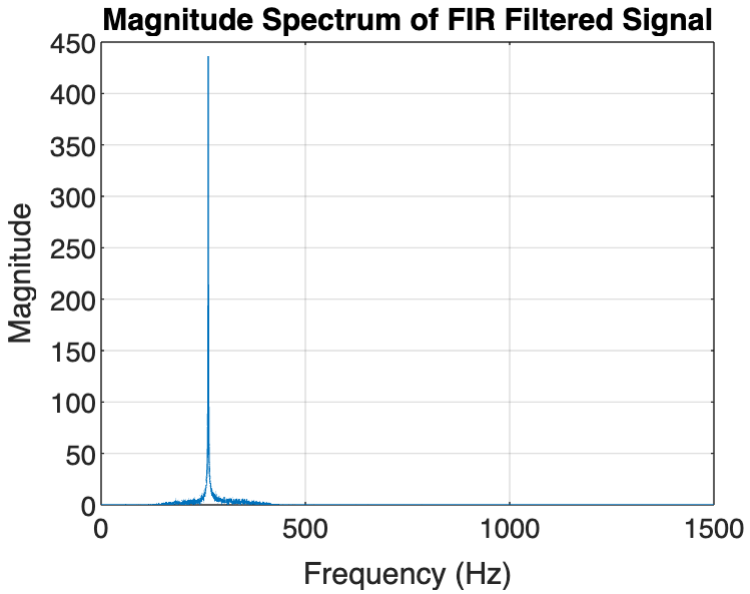
```
disp(M);
```

```
   219
```

```


% Calculate the Hamming windowed impulse response
n = 0:M-1;
wc1 = (ws1 + wp1)/2;
wc2 = (wp2 + ws2)/2;
hd = ideal_lp(wc2, M) - ideal_lp(wc1, M);
whamming = hamming(M)';
h = hd .* whamming;

% Compute numerator coefficients
bhamming = impz(h, 1);

% Calculate frequency response
[dbwin, mag, pha, grd, wwin] = freqz_m(bhamming, 1);

% Plot the magnitude response
figure();
plot(wwin*Fs/(2*pi), dbwin);
title('Magnitude Response in dB');
grid on;
xlabel('Frequency (pi units)');
xlim([0,1500]);
ylabel('Magnitude (dB)');
ylim([-100, 50]);
```




```
% Apply the filter to the audio signal
filtered_signal = filter(bhamming, 1, sig);

% Compute the magnitude spectrum of the filtered signal
NFIR = length(filtered_signal);
freq_bins = 0:Fs/NFIR:Fs - Fs/NFIR;
magnitude_spectrum = abs(fft(filtered_signal));

% Plot the magnitude spectrum
figure();
plot(freq_bins, (magnitude_spectrum));
title('Magnitude Spectrum of FIR Filtered Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
xlim([0, 1500]);
grid on;
```

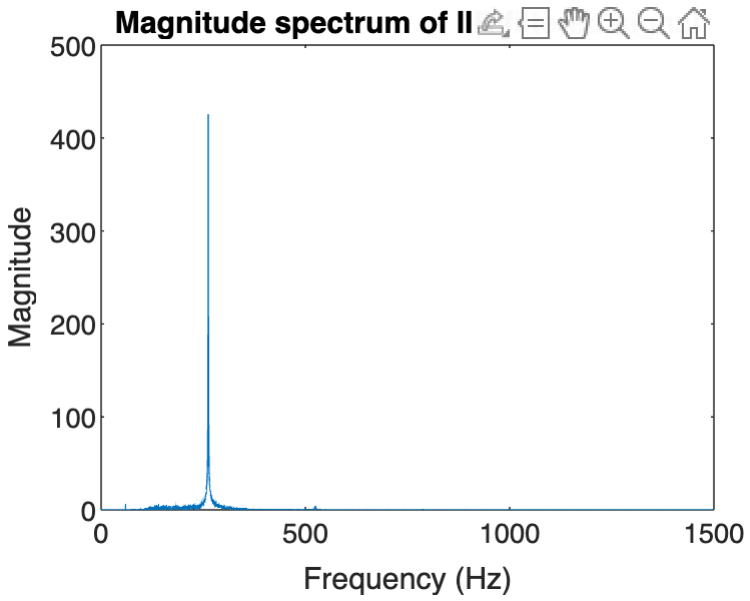



```
% The magnitude spectrum shows suppression of the-
% quasi-periodic noise with minimal attenuation of the desired signal.
% Save the filtered sound as a WAV file
output_filename = 'remoto_noteFIR.wav';  % Desired filename for the filtered
sound
audiowrite(output_filename, filtered_signal, Fs);
```

IIR filter

```
% The bandpass filter is recreated using a Butterworth filter

% Define the filter parameters
fband = [120 300];  % Passband frequency range in Hz
buttorder = 4;  % Filter order

% Normalize the passband frequencies
fnband = fband / (Fs/2);

% Design the Butterworth filter
[bbutter, abutter] = butter(buttorder, fnband, 'bandpass');

% Apply the filter to the audio signal
buttbandpass = filter(bbutter, abutter, sig);
bandpasssig = fft(buttbandpass);

figure()
plot(f(1:N/2), abs(bandpasssig(1:N/2)));
xlim([0 1500]); % limits the plot to its first 1000 points
title('Magnitude spectrum of IIR filtered signal');
xlabel('Frequency (Hz)');
```

```
ylabel('Magnitude');
```




```
% The magnitude spectrum shows suppression of the-
% quasi-periodic noise with minimal attenuation of the desired signal.
% Save the filtered sound as a WAV file
output_filename = 'remoto_noteIIR.wav';  % Desired filename for the filtered
sound
audiowrite(output_filename, buttbandpass, Fs);
```

Functions

```
function h_d = compHd(w_c, M)
    % n is an array from 0 to M-1
    n = 0:M-1;

    % Compute the filter coefficients using the discrete-time impulse
response equation for a low-pass filter (for even values of M)
    h_d = sin(w_c*(n-(M-1)/2))./(pi*(n-(M-1)/2));

    % Set the middle coefficient separately to avoid divide-by-zero error
(for odd values of M)
    h_d((M+1)/2) = w_c/pi;
end

function [db, W] = freqzM(num, den)
    % Compute frequency response of digital filter
    [H, W] = freqz(num, den, 1000, 'whole');

    % Extract magnitude and frequency response from 0 rad to Nyquist
frequency (half of sampling rate)
```

```
    mag = abs(H(1:1:501));
    W = W(1:1:501);

    % Convert magnitude response to dB scale with maximum value normalization
    db = 20*log10((mag+eps)/max(mag));

end

function hd = ideal_lp(wc,M)
% Ideal LowPass filter computation
% --------------------------------
% [hd] = ideal_lp(wc,M)
% hd = ideal impulse response between 0 to M-1
% wc = cutoff frequency in radians
% M = length of the ideal filter
%
alpha = (M-1)/2; n = [0:1:(M-1)];
m=n- alpha; fc = wc/pi; hd = fc*sinc(fc*m);
end

function [db,mag,pha,grd,w] = freqz_m(b,a);
% Modified version of freqz subroutine
% ------------------------------------
% [db,mag,pha,grd,w] = freqz_m(b,a);
% db = Relative magnitude in dB computed over 0 to pi radians
% mag = absolute magnitude computed over 0 to pi radians
% pha = Phase response in radians over 0 to pi radians
% grd = Group delay over 0 to pi radians
% w = 501 frequency samples between 0 to pi radians
% b = numerator polynomial of H(z) (for FIR: b=h)
% a = denominator polynomial of H(z) (for FIR: a=[1])
%
[H,w] = freqz(b,a,1000,'whole');
H = (H(1:1:501))'; w = (w(1:1:501))';
mag = abs(H); db = 20*log10((mag+eps)/max(mag));
pha = angle(H); grd = grpdelay(b,a,w);
end
```