CmpE 362 Introduction to Signals for Computer Engineers

Spring 2020

Homework 2

Emilcan ARICAN

2016400231

Solution 1

Part 1

Definition of $x(t)$ over one period.

$$x(t) = \begin{cases} 2t/T_0 & \text{for} \quad 0 \le t < T_0/2 \\ 2(T_0 - t)/T_0 & \text{for} \quad T_0/2 \le t < T_0 \end{cases}$$

$$a_0 = \frac{1}{T_0} \int_0^{T_0} x(t)\,dt$$

$$a_0 = \frac{1}{T_0}(\text{area}) = \frac{1}{T_0}(T_0)\,\frac{1}{2} = \frac{1}{2}$$

$$a_k = \frac{1}{T_0} \int_0^{T_0} x(t) \,.\, e^{-j(2\pi/T_0)k.t}\,dt$$

$$= \frac{1}{T_0} \int_0^{T_0/2} (2t/T_0).e^{-j(2\pi/T_0)kt}\,dt + \frac{1}{T_0} \int_{T_0/2}^{T_0} (2(T_0-t)/T_0).e^{-j(2\pi/T_0)kt}\,dt$$

$$= \frac{2}{T_0^2} \int_0^{T_0/2} t.e^{-j(2\pi/T_0)kt}\,dt + \frac{2}{T_0^2} \int_{T_0/2}^{T_0} (T_0-t).e^{-j(2\pi/T_0)kt}\,dt$$

$$= \frac{2}{T_0^2}\left[\frac{(-j(2\pi/T_0)kt-1).e^{-j(2\pi/T_0)kt}}{(-j(2\pi/T_0)k)^2}\right]_0^{T_0/2} + \frac{2}{T_0^2}\left[\frac{(-j(2\pi/T_0)k(T_0-t)+1).e^{-j(2\pi/T_0)kt}}{(-j(2\pi/T_0)k)^2}\right]_{T_0/2}^{T_0}$$

$$= \frac{2}{T_0}\left[\frac{(-j\pi k-1).e^{-j\pi k}-(-1)}{(-j(2\pi/T_0)k)^2}\right] + \frac{2}{T_0^2}\left[\frac{1-(-j\pi k+1).e^{-j\pi k}}{(-j(2\pi/T_0)k)^2}\right]$$

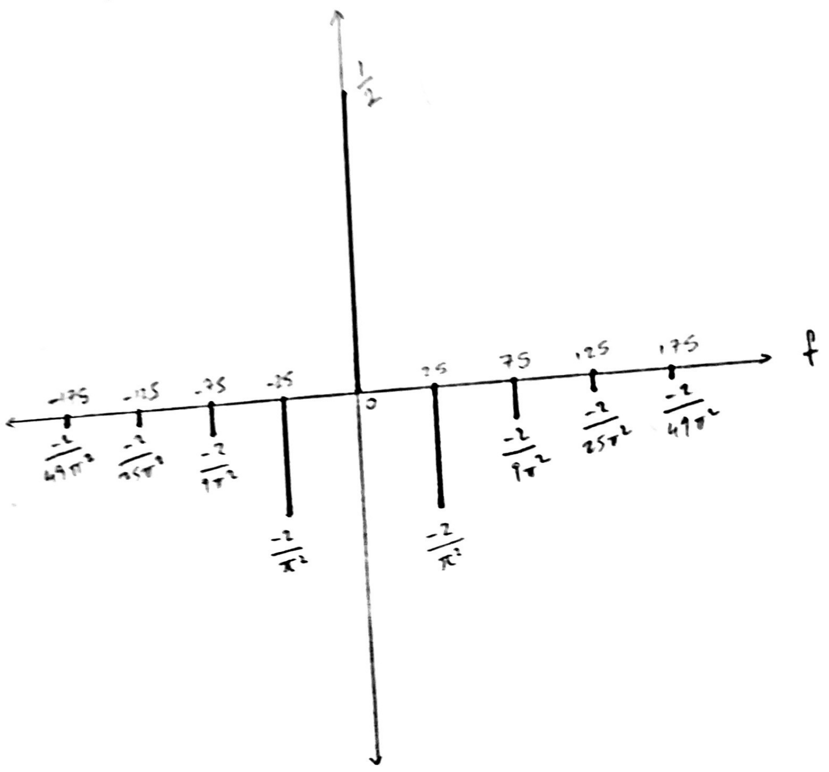
$$= \frac{2}{T_0^2}.\left[\frac{2-2e^{-j\pi k}}{(-j(2\pi/T_0)k)^2}\right] = \frac{\cancel{2}}{T_0^2}.\left[\frac{T_0^2.\cancel{2}.(1-e^{-j\pi k})}{(-1)^2.j^2.2^{\cancel{2}}.(\pi k)^2}\right]$$

$$= \frac{-(1-e^{-j\pi k})}{(\pi k)^2} = \frac{e^{-j\pi k}-1}{(\pi k)^2}$$

$$a_k = \frac{(-1)^k-1}{(\pi k)^2}$$

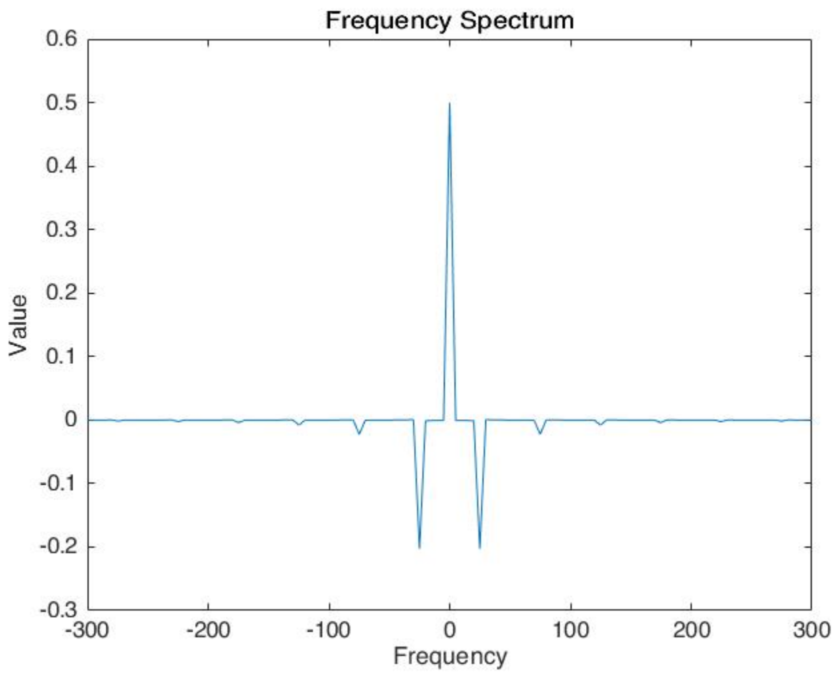
$$a_k = \begin{cases} \frac{-2}{(\pi k)^2} & \text{for odd } k \\ 0 & \text{for even } k \text{ (except zero)} \\ 1/2 & \text{for zero} \end{cases}$$

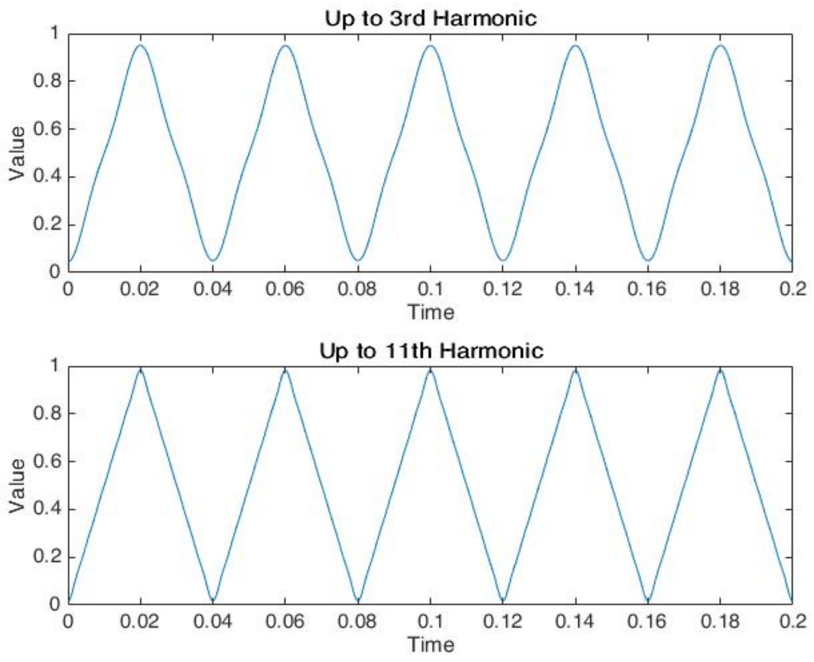
Frequency Spectrum



Part 2

When compared the plot created by matlab, the one that I drew and the one that matlab plotted are ended up being the same frequency spectrums.





Solution 1 - Code

```
%clear old variables
clear; clc;

fs = 4096;                          %frequency
dt = 1/fs;                          %unit time
t = 0:dt:0.2;                       %time array
x = sawtooth(2*pi*25*t,1/2)/2 + 1/2; %triangle signal

Y = fft(x);                         %discrete fourier transform
n = length(x);                      %number of samples
Y0 = fftshift(Y);                   %shift transformed values
f0 = (-n/2:n/2-1)*(fs/n);           %frequency range
Y0 = Y0/n;                          %normalize

H3 = calculate_harmonic(Y0*n,f0,3); %strips the needed part of harmonics
H3 = ifftshift(H3);                 %inverse shift
H3 = ifft(H3);                      %inverse fft

H11 = calculate_harmonic(Y0*n,f0,11); %strips the needed part of harmonics
H11 = ifftshift(H11);               %inverse shift
H11 = ifft(H11);                    %inverse fft

%plot frequency spectrum
figure(1);
plot(f0,Y0),title("Frequency Spectrum"), xlabel('Frequency'), ylabel('Value'), ylim([-0.3
    0.6]), xlim([-300 300]);

%plot harmonics
figure(2);
subplot(2,1,1);
plot(t,H3), title("Up to 3rd Harmonic"),xlabel('Time'), ylabel('Value');
subplot(2,1,2);
plot(t,H11), title("Up to 11th Harmonic"),xlabel('Time'), ylabel('Value');

%strips the harmonics
function y = calculate_harmonic(x, f0, h)
    H = h*25;
    y = x;
    [~,n] = size(x);
    for i = 1:n
        if abs(f0(i)) > H
            y(i) = 0;
        end
    end
end
```

Solution 2

Chosen signal: $\cos(2\pi 0.2\text{t}) + \cos(2\pi 0.7\text{t})$

Ideal sampling Nyquist rate: 1.4

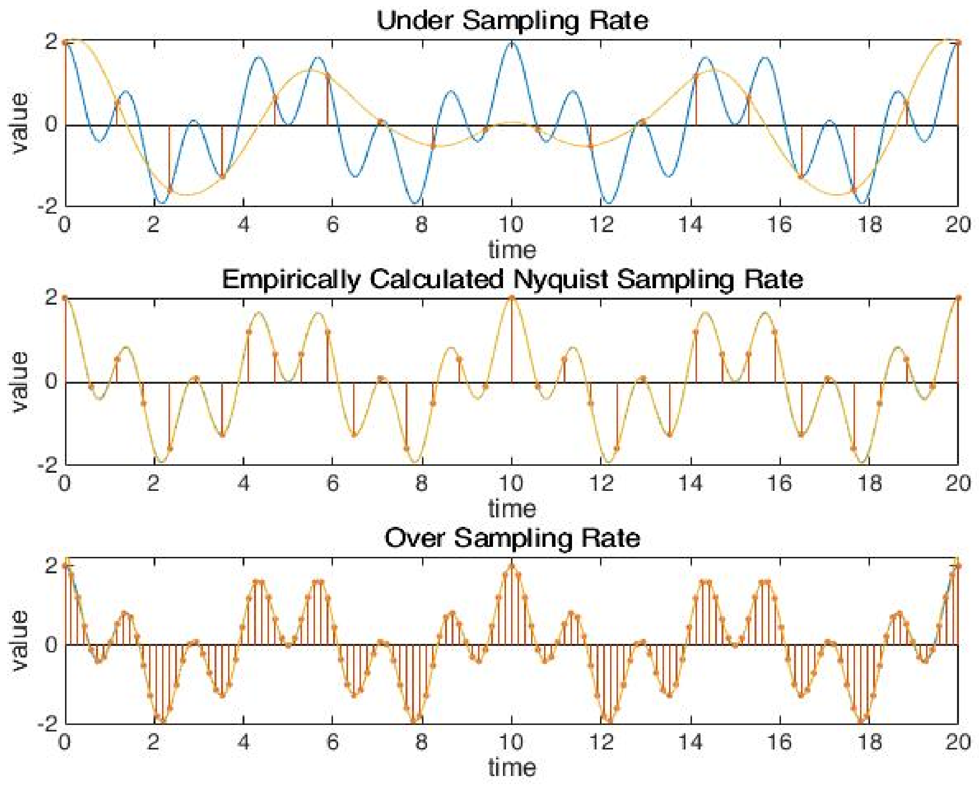
Empirical sampling Nyquist rate: 1.7

Under sampling rate: 0.85

Over sampling rate: 6.8

Calculating Nyquist rate empirically

Basically I started from a small value and started to increase the sampling rate every iteration. In order to keep track of how close my reconstruction to the original one, I have defined a mean absolute error function. When the amount of error falls under some desired value, I concluded that I found a value close to the Nyquist rate. Then multiplied that sampling rate with 4. So I have got my over sampling rate as well. For the under sampling, a similar approach is followed. Divided the empirical Nyquist rate with 2 and that is my under sampling rate. Finally, I have created corresponding plots for each of those.



Solution 2 - Code

```
%clear old variables
clear; clc;

%create time vector and actual signal
[Y, t] = sample(1024);

fs = 0.1;
err = Inf;
%increase sampling frequency until error falls under desired value
while err > 10e-3
    %take samples
    [Ys, ts] = sample(fs);
    if length(ts) > 1
        %reconstruct the signal
        Rs = sinc_interpolation(Ys, ts, t);
        %calculate error
        err = meanAbsoluteError(Y,Rs);
    end
    fs = fs + 0.1;
end

under_fs = fs/2;
%take samples
[under_s, under_ts] = sample(under_fs);
%reconstruct the under sampled signal
under_recons = sinc_interpolation(under_s, under_ts, t);

exact_fs = fs;
%take samples
[exact_s, exact_ts] = sample(exact_fs);
%reconstruct the exact sampled signal
exact_recons = sinc_interpolation(exact_s, exact_ts, t);

over_fs = fs*4;
%take samples
[over_s, over_ts] = sample(over_fs);
%reconstruct the over sampled signal
over_recons = sinc_interpolation(over_s, over_ts, t);

%plot the under sampled signal
subplot(3, 1, 1);
plot(t,Y);
hold on;
stem(under_ts,under_s,'.');
hold on;
plot(t,under_recons);
title('Under Sampling Rate'), xlabel('time'), ylabel('value');
hold off;

%plot the exact sampled signal
subplot(3, 1, 2);
plot(t,Y);
hold on;
```

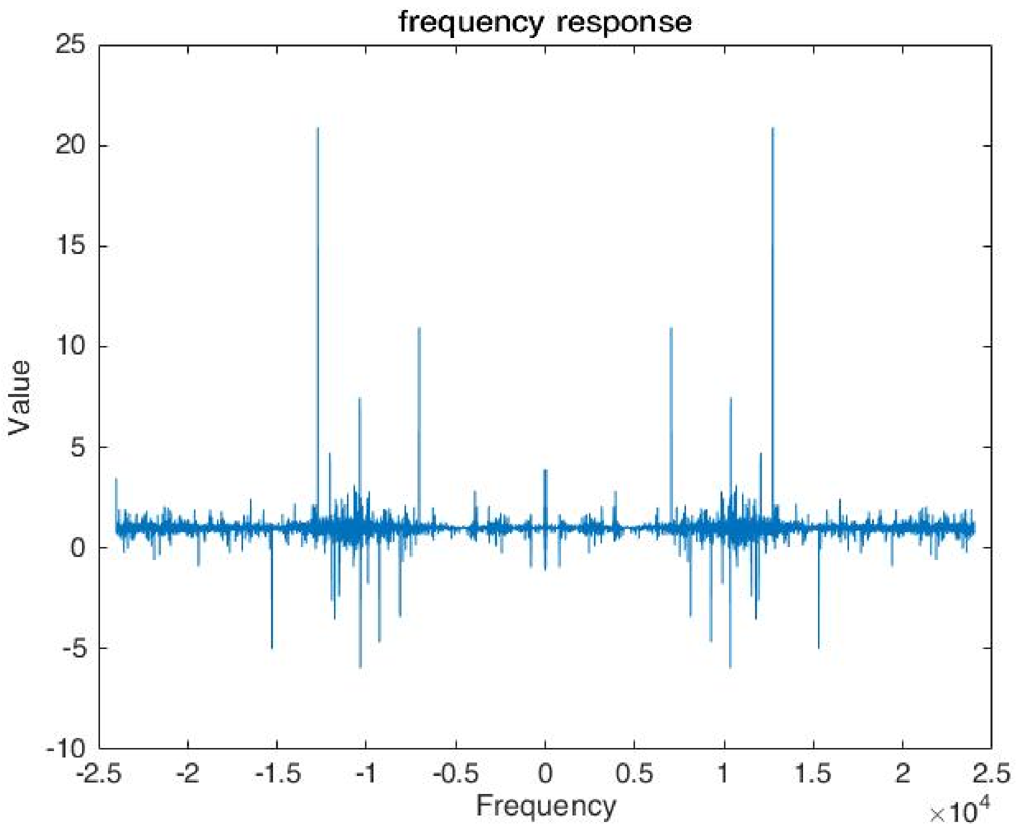
```
stem(exact_ts,exact_s,'.');
hold on;
plot(t,exact_recons);
title('Empirically Calculated Nyquist Sampling Rate'), xlabel('time'), ylabel('value');
hold off;

%plot the over sampled signal
subplot(3, 1, 3);
plot(t,Y);
hold on;
stem(over_ts,over_s,'.');
hold on;
plot(t,over_recons);
title('Over Sampling Rate'), xlabel('time'), ylabel('value');
hold off;

%performs sinc interpolation
function y = sinc_interpolation(sample, ts, t)
dts = ts(1) - ts(2);
[Ts,T] = ndgrid(ts,t);
y = sample*sinc((Ts - T)/dts);
end

%performs sampling
function [Ys,ts] = sample(fs)
dts = 1/fs;
ts = 0:dts:20;
Ys = cos(2*pi*0.2*ts) + cos(2*pi*0.7*ts);
end

%calculates mean absoulute error
function y = meanAbsoluteError(A, B)
y = mean(abs(A-B));
end
```

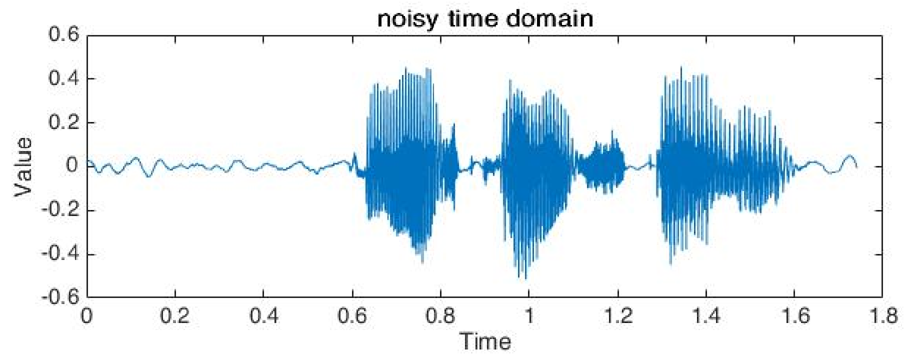
Solution 3

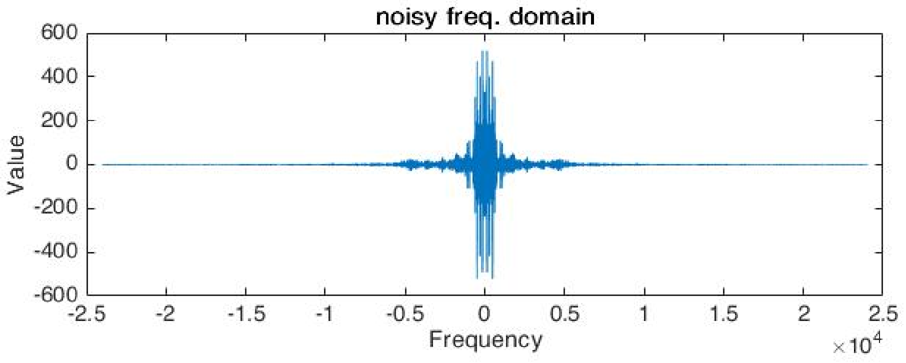
Ideal Filter

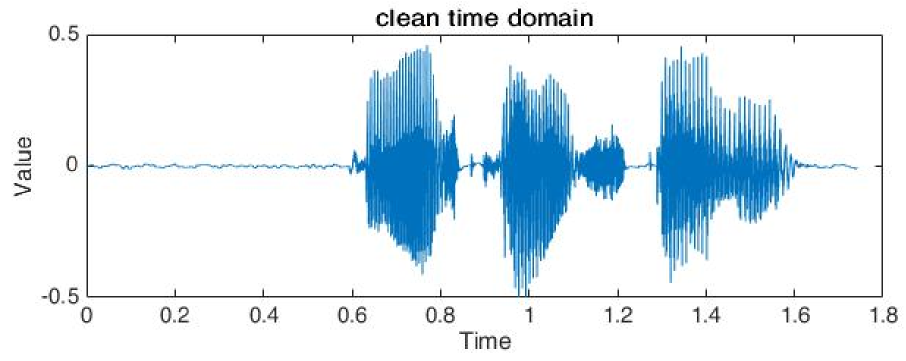
Selected File: p232_001.wav

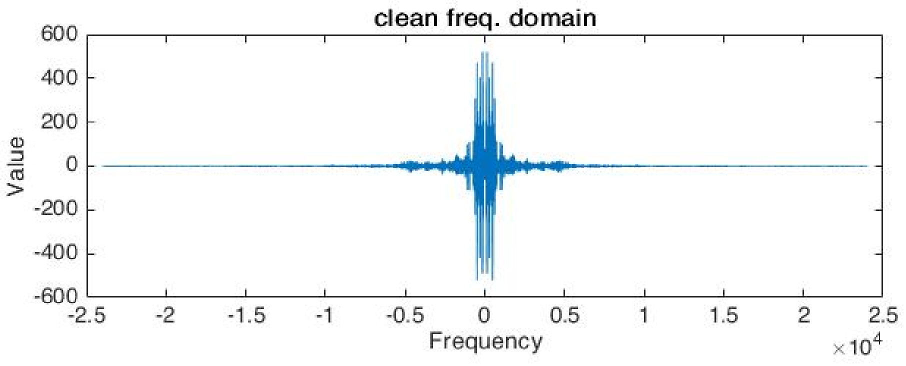
First, fft of the both noisy and the clean signals are taken. Then calculated the frequency response by dividing fft of the clean one to the fft of the noisy one. Then shifted and ploted. After that, frequency response switched to time domain , by using ifft and ifftshift. So that time response is obtained. Finally, noisy signal convoluted with the time response. So clean signal is obtained from the noisy one.

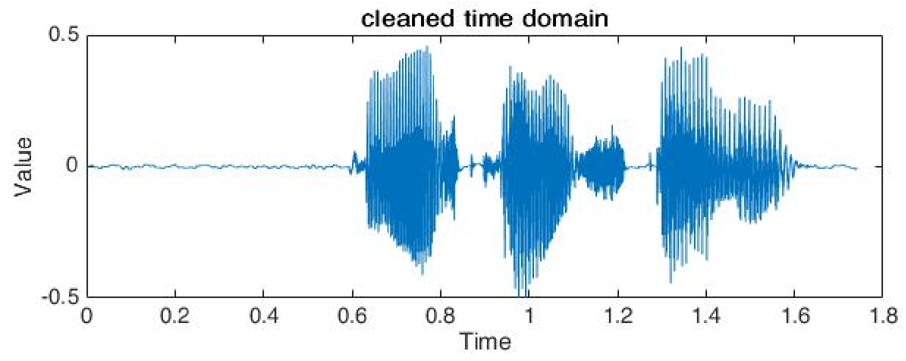


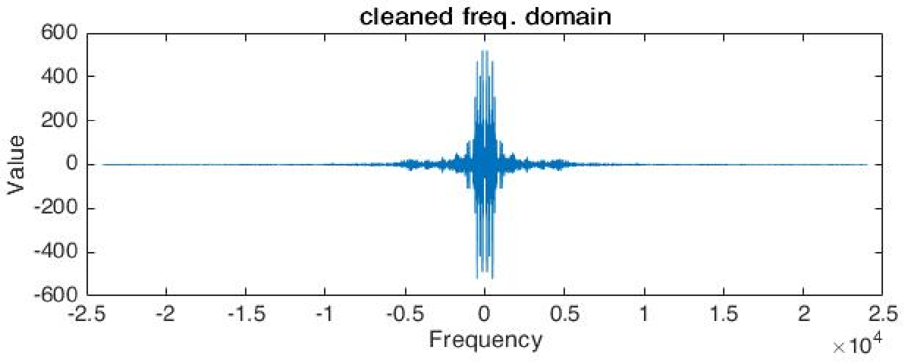

noisy time domain
Value
Time


noisy freq. domain
Value
Frequency
×10^4


clean time domain
Value
Time


clean freq. domain
Value
Frequency
×10^4





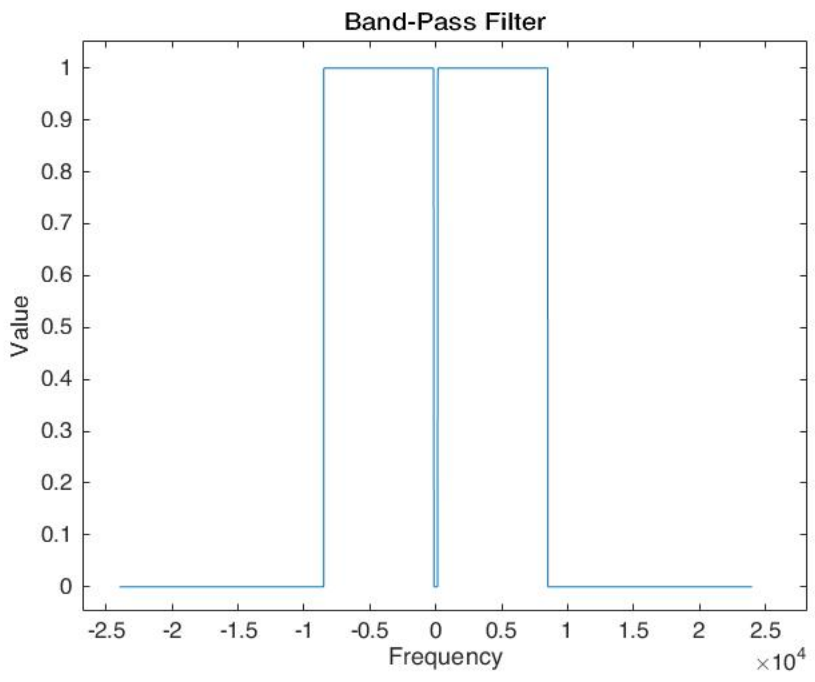
Ideal Filter - Code

```
%clear old variables
clear; clc;

%read the audio files
[noisy,fs] = audioread('noisy/p232_001.wav');
[clean,~] = audioread('clean/p232_001.wav');

[m,~] = size(noisy);                           %get size of the signal
dt = 1/fs;                                     %calculate unit time
t = dt*(0:m-1);                                %initialize time vector
freq_response = fft(clean)./fft(noisy);        %calculate frequency response
shifted_freq_response = fftshift(freq_response); %shift the values
time_response = ifft(freq_response);           %calculate time response
cleaned = cconv(noisy, time_response, m);      %clean noisy signal

[f_noisy, freq_range] = to_freq_dom(noisy, fs); %switch to freq. domain
[f_clean, ~]          = to_freq_dom(clean, fs); %switch to freq. domain
[f_cleaned, ~]        = to_freq_dom(clean, fs); %switch to freq. domain

%plot figures
figure(1);
plot(freq_range,shifted_freq_response), title("frequency response"), xlabel('Frequency'),
    ylabel('Value');
customPlot("noisy", t, noisy, freq_range, f_noisy, 2);
customPlot("clean", t, clean, freq_range, f_clean, 3);
customPlot("cleaned", t, cleaned, freq_range, f_cleaned, 4);

function [shift_freq_dom, freq_range] = to_freq_dom(x, fs)
    [m,~] = size(x);                    %get size of the signal
    freq_dom = fft(x);                  %switch to freq. domain
    shift_freq_dom = fftshift(freq_dom); %shift the values
    freq_range = ((-m/2):(m/2-1))*(fs/m); %calculate frequency range
end

function customPlot(name, t, time, f, freq, fig)
figure(fig);
subplot(2,1,1);
plot(t,time), title(strcat(name, " time domain")), xlabel('Time'), ylabel('Value');
subplot(2,1,2);
plot(f,freq), title(strcat(name, " freq. domain")), xlabel('Frequency'), ylabel('Value');
end
```

Filter Design

Used Filter Types: Band-Pass Filter, Averaging Filter

Since we are clearing noise from the human voices, first thing to do is to make a research about the frequency range of the human voice. As I learned, frequency range for the woman voice is in between 150 and 10000 Hz. On the other hand man voice is in between 100 and 8500 Hz. So I decided to consider these values while designing my filter. I took average of the both ends, and created an interval. Outside of these boundaries can be assumed to be a non-human voice, in other words noise. Designed a low-pass and a high-pass filter and combined them to create a band-pass filter in order to strip the human voice frequency range from the noisy spectrum. Finally in order to get a smoother signal, I have applied an average filter.



Filter Design - Code

```
%clear old variables
clear; clc;

%input and output files
input_file = 'noisy/p232_001.wav';
output_file = 'out.wav';

%read input file aka. noisy signal
[noisy,fs] = audioread(input_file);
[m,~] = size(noisy);
dt = 1/fs;
t = dt*(0:m-1);

[f_noisy, freq_range] = to_freq_dom(noisy, fs);

%prepare high pass filter
hp_filter = zeros(size(freq_range));
for i = 1:m
    if abs(freq_range(i)) < 125
        hp_filter(i) = 0;
    else
        hp_filter(i) = 1;
    end
end

%prepare high pass filter
lp_filter = zeros(size(freq_range));
for i = 1:m
    if abs(freq_range(i)) > 9250
        lp_filter(i) = 0;
    else
        lp_filter(i) = 1;
    end
end

%generate a band pass filter
%with combining low pass and high pass filters
bp_filter = lp_filter.*hp_filter;

%move band pass filter to time domain
unshifted_bp_filter = ifftshift(bp_filter);
t_bp_filter = ifft(unshifted_bp_filter);

%apply band pass filter
cleaned = cconv(noisy,t_bp_filter,m);

%apply average filter to smoothen the signal
for i = 1:m
    if i < m-10
        cleaned(i) = mean(cleaned(i:i+10));
    else
        cleaned(i) = mean(cleaned(i:m));
    end
```

```
end

audiowrite(output_file,cleaned,fs);

function [shift_freq_dom, freq_range] = to_freq_dom(x, fs)
    [m,~] = size(x);                    %get size of the signal
    freq_dom = fft(x);                  %switch to freq. domain
    shift_freq_dom = fftshift(freq_dom); %shift the values
    freq_range = ((-m/2):(m/2-1))*(fs/m); %calculate frequency range
end
```