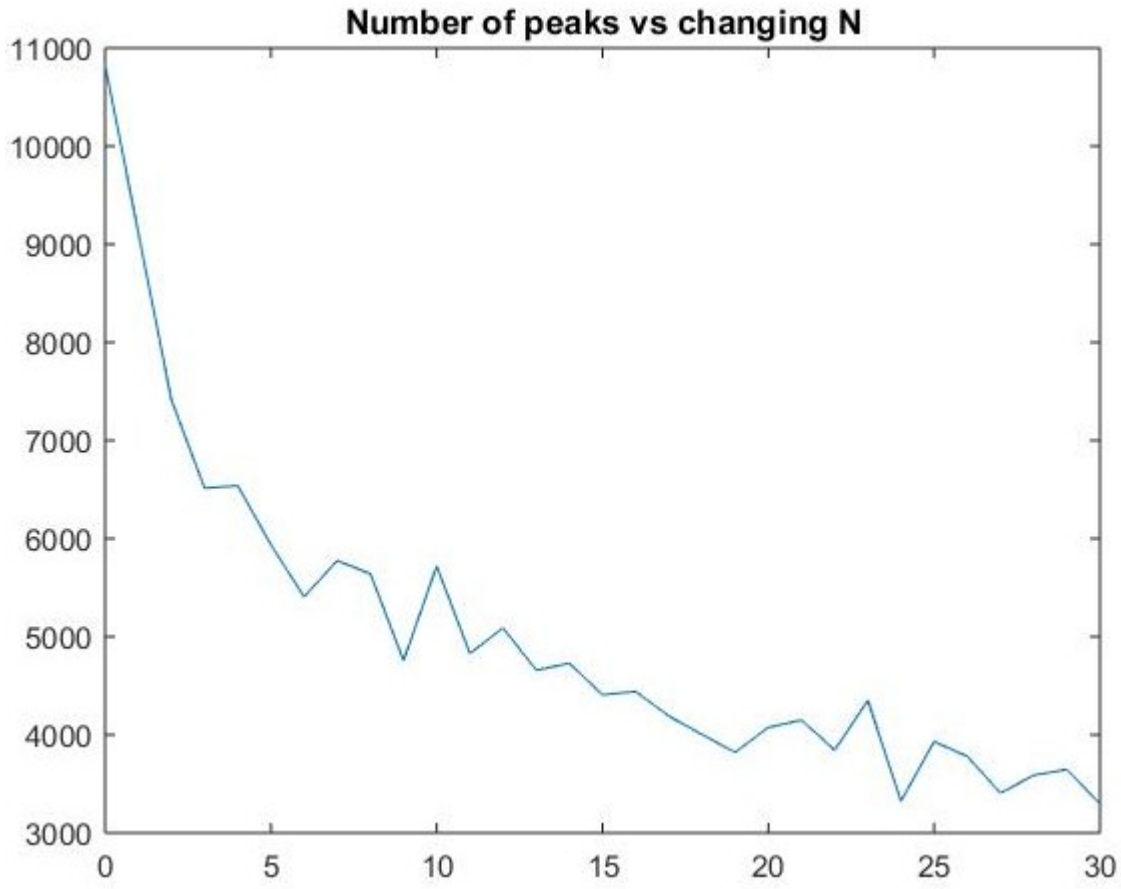
CmpE 362: Homework 2: Time Domain Filtering

Berk Erol 2014400189

Problem 1

```
% Prepare file for reading
file = fopen('exampleSignal.csv', 'r');
% Read the file
v = fscanf(file, '%f');
% Close the file source
fclose(file);
% Discard the first three elements
v1 = v(4:end);

% Initialize the array that holds number of samples
numberOfSamples = [0 2:30];
% Initialize the array that holds number of peaks
numberOfPeaks = ones(1, 30);

% Find number of peaks for no filter option
numberOfPeaks(1) = length(findpeaks(v1));

for i = 2:30
    % Create coefficients for the moving average filter
    b = ones(1, i) / i;
    % Filter the data with moving average coefficients
    filtered = filter(b, 1, v1);
    % Find peaks from moving average filtered data
    peaks = findpeaks(filtered);
    % Save the total number of peaks
    numberOfPeaks(i) = length(peaks);
end

plot(numberOfSamples, numberOfPeaks);
title('Number of peaks vs changing N');
```



In general, number of peaks decreases as number of samples used for average calculation increases. The reason is that if we increase the number of data points then resulting averages will share increasing number of common data points that are used for calculation so they will become more similar with increasing number of shared data points. Of course there are some exceptions like 10 and 23 but common trend is like this.

Problem 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
%   CMPE 362 Homework II-b   %
%%%%%%%%%%%%%%%%%%%%%%%%%%

                                                % Fs is the frequency =
number of samples per second
                                                % y is the actual sound
data
hfile = 'laughter.wav';                         % This is a string,
corresponding to the filename
clear y Fs                                      % Clear unneded variables

%% PLAYING A WAVE FILE

[y, Fs] = audioread(hfile);      % Read the data back into MATLAB, and
listen to audio.
                                                % nbits is number of bits
per sample
sound(y, Fs);                                   % Play the sound & wait
until it finishes

duration = numel(y) / Fs;                       % Calculate the duration
pause(duration + 2)                             % Wait that much + 2
seconds

%% CHANGE THE PITCH

sound(y(1:2:end), Fs);                          % Get rid of even numbered
samples and play the file

pause(duration / 2 + 2)

%% EXERCISE I
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Re-arrange the data so that   %
%   the frequency is quadrupled and play the file   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sound(y(1:4:end), Fs);

pause(duration / 4 + 2);

%% EXERCISE II
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Re-arrange the data so that   %
%   the frequency is halved and play the file  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sizeY = size(y, 1);
```

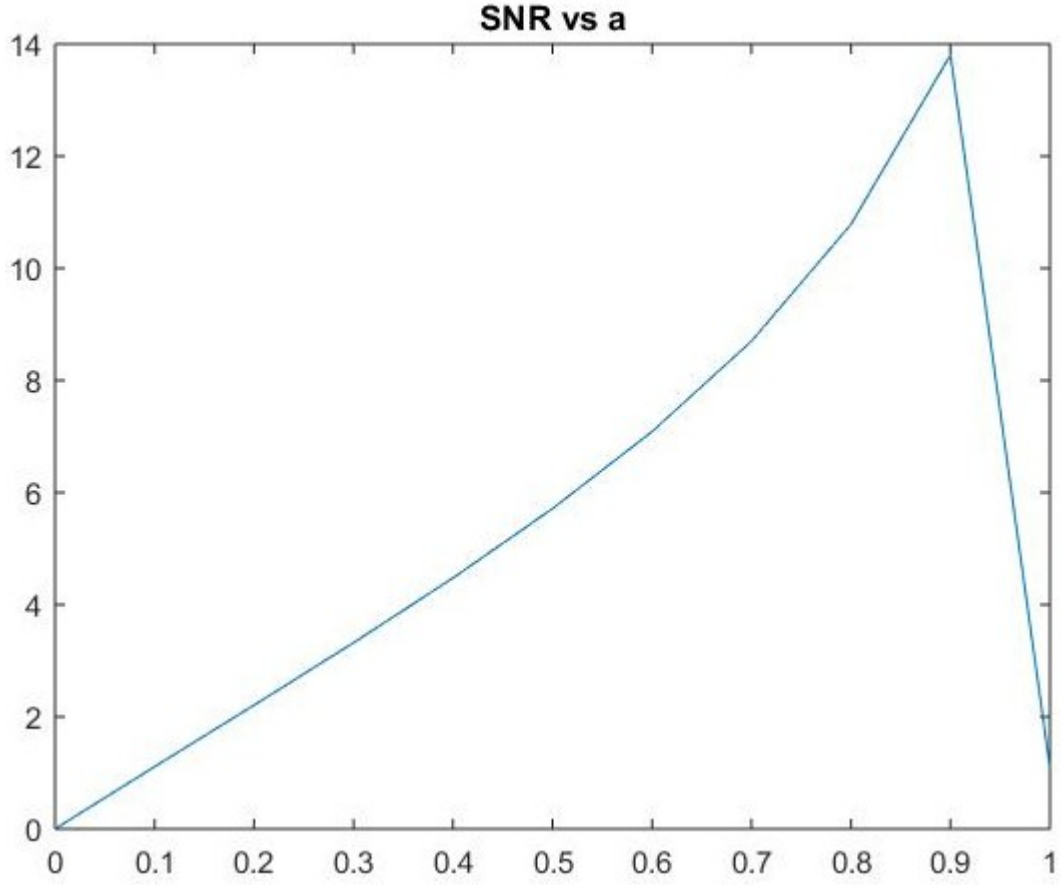
```
newY = ones(sizeY * 2, 1);
for i = 1: sizeY
    newY(2 * i - 1) = y(i);
    newY(2 * i) = y(i);
end

sound(newY, Fs);

pause(duration * 2 + 2);

%% EXERCISE III
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Double Fs and play the sound  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sound(y, Fs * 2);

pause(duration / 2 + 2);

%% EXERCISE IV
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Divide Fs by two and play the sound  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sound(y, Fs / 2);
```

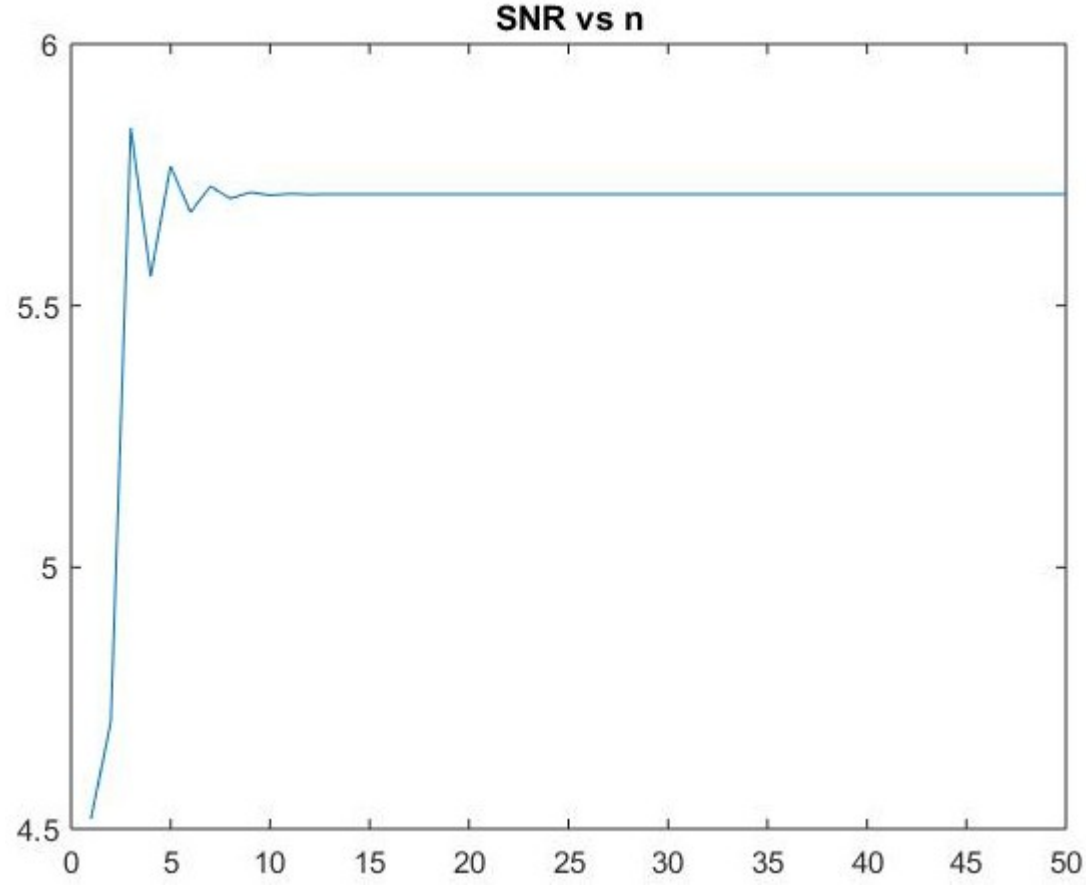
In changing the pitch part, we eliminated half of the data and the effect was the frequency is doubled and the pitch is higher. In exercise 1, we just used quarter of the data and the effect was the frequency is quadrupled and the pitch is even more higher. In exercise 2, we generated double size data by duplicating each data point and the effect was the frequency is halved and the pitch is lower. In exercise 3, we doubled the sampling rate and the effect was similar with changing the pitch part. In exercise 4, we halved the sampling rate and the effect was similar with exercise 2.

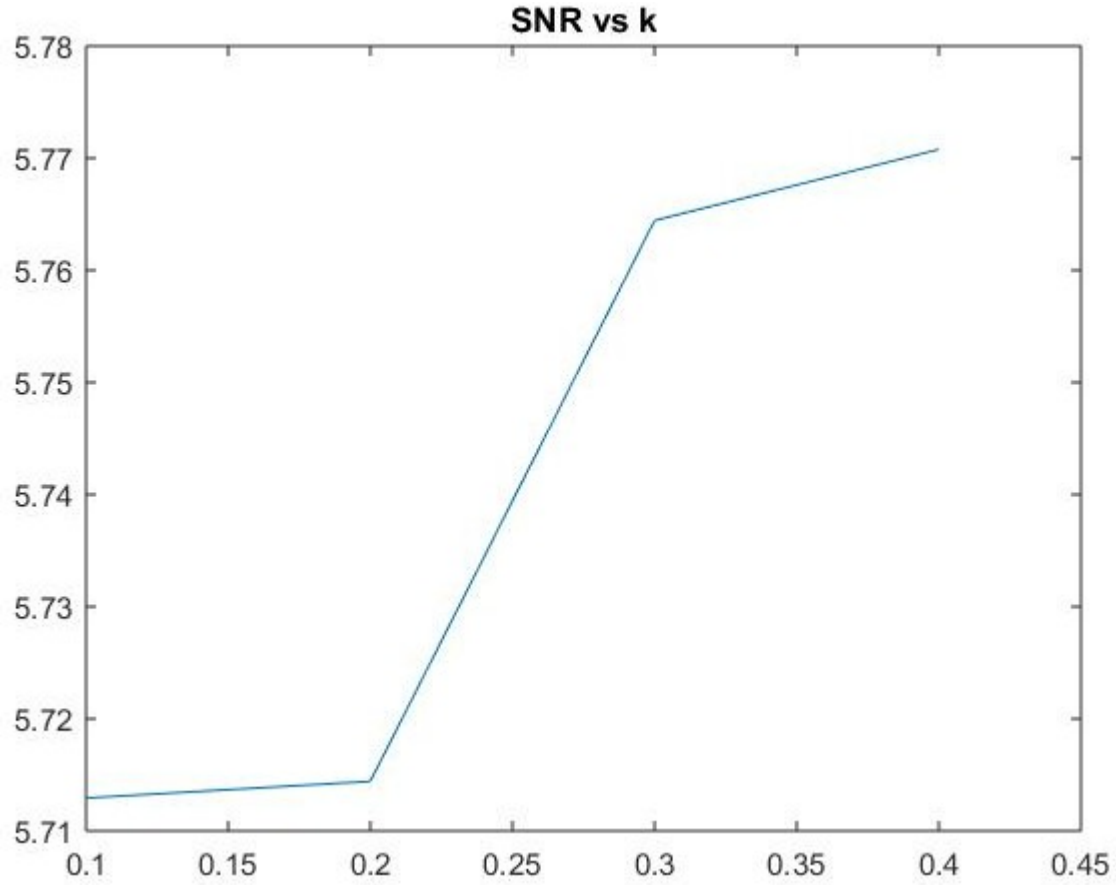
Problem 3

```
% Read audio file
[x, Fs] = audioread('mike.wav');

% Declare constants
A = 0.5;
N = 25;
K = 0.1;

% Change a from 0 to 1
as = 0:0.1:1;
% Initialize array that holds SNRs
snrs = zeros(length(as), 1);
for i = 1:length(as)
    % Calculate SNR of original and recovered signal
    % Recovered signal is N-tap Filter output
    % N-tap Filter input is the combination of original signal and delayed
    % version of it with K seconds
    snrs(i) = SNR(x, ntap(x + delayseq(x, K, Fs), as(i), N, K, Fs));
end
plot(as, snrs);
title('SNR vs a');
figure;

% Change n from 1 to 50
ns = 1:50;
% Initialize array that holds SNRs
snrs = zeros(length(ns), 1);
for i = 1:length(ns)
    % Calculate SNR of original and N-tap filter output
    snrs(i) = SNR(x, ntap(x + delayseq(x, K, Fs), A, ns(i), K, Fs));
end
plot(ns, snrs);
title('SNR vs n');
figure;

% Change k from 0.1 to 0.4
ks = 0.1:0.1:0.4;
% Initialize array that holds SNRs
snrs = zeros(length(ks), 1);
for i = 1:length(ks)
    % Calculate SNR of original and N-tap filter output
    snrs(i) = SNR(x, ntap(x + delayseq(x, ks(i), Fs), A, N, ks(i), Fs));
end
plot(ks, snrs);
title('SNR vs k');

% N-tap filter: multiplies the delayed signals and adds to original
function [y] = ntap(x, a, n, k, Fs)
y = x;
```

```
for i = 1:n
    y = y + (((- a) ^ i) * delayseq(x, k * i, Fs));
end
end

% Calculates SNR of original and recovered signal
function [z] = SNR(x, y)
z = 10 * log10(sum(x .^ 2) / sum((y - x) .^ 2));
end
```


SNR vs a
14
12
10
8
6
4
2
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


SNR vs n
6
5.5
5
4.5
0
5
10
15
20
25
30
35
40
45
50



SNR increases as alpha increases until 0.9. After that SNR decreases. To get a better SNR, we should use alpha close to 0.9. SNR oscillates between 4.5 and 6 until it converges at 5.75 as n increases from 1 to 50. So there is no need to increase n after some point to get a better SNR. SNR increases as k increases. In general, N-tap filter does a fairly good job in cleaning noise from delayed signals.