CmpE 362
Spring 2019
Project 2

Gurkan Demir
2015400177

April 19, 2019

Contents

1 Introduction

1.1 Project Description

In this homework, I am expected to implement some simple time domain exercises using MATLAB. We have 3 questions in this homework, named **Advanced Peak Finder, Frequency (Pitch) of the Sound, N-tap Filter**, respectively.

In the first question, I am expected to plot number of peaks I found versus N.

In the second question, I am expected to follow instructions in **waveexaple.m** on sample file and explain differences between applications.

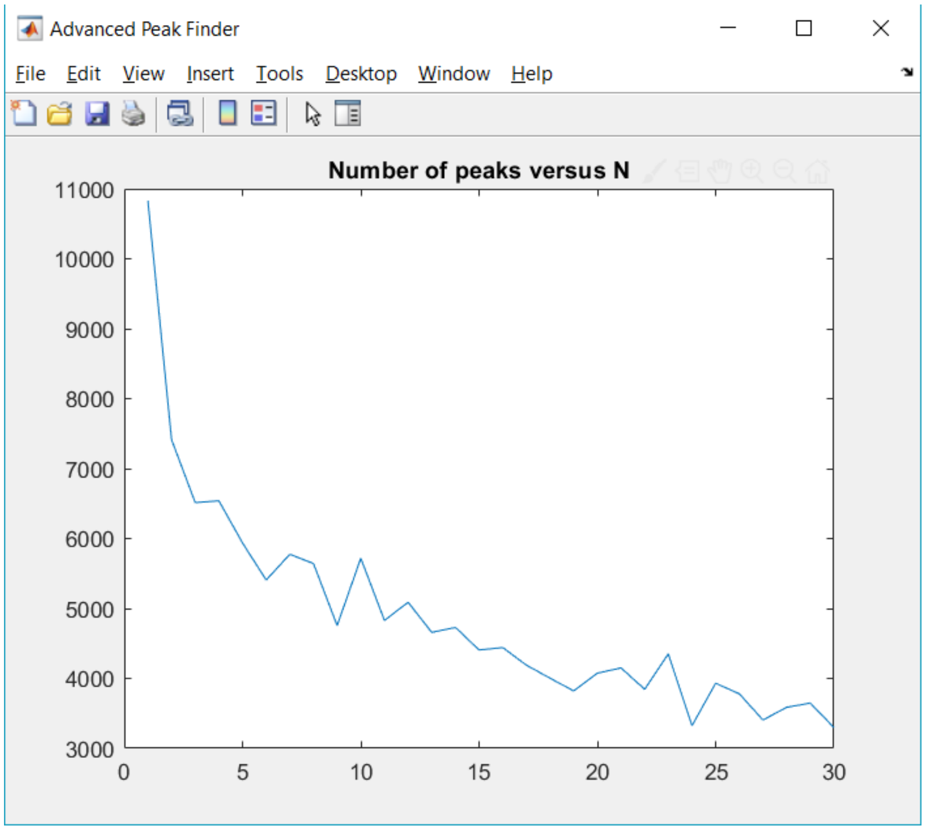
In the last question, I am expected to design N-tap-filter in order to alleviate the effects of delayed versions of sound.

2 Advanced Peak Finder

2.1 Question Description

In this part, I am expected to improve my peak detection algorithm that I developed in the previous homework. I am designing a moving average filter. For this filter, I am expected to change the number of samples used for average calculation, N, from 2 to 30. Additionally, I am expected to plot the number of peaks I found versus N.

2.2 Output



x axis is N, y axis is number of peaks that I found. For N = 1, it shows the number of peaks without moving average.

2.3 Result

As we can see from the figure above, general trend is number of peaks decreases as N increases. Of course there are some exceptions such as when N = 5, 10. Some peaks are vanished due to the fact that when we take the average of more number of data, they become closer to each other.

Additionally, we can conclude that as number samples used for average calculation increases, number of peaks are decreases so fluctuations are smoothed out which causes us to get more accurate spectrum of data.

2.4 Code

Algorithm 1 Advanced Peak Finder

```
clc; clear;
f = figure('Name', 'Advanced Peak Finder', 'NumberTitle', 'off');
figure(f);
fileID = fopen('exampleSignal.csv', 'r');

A = fscanf(fileID, '
A = A(4:length(A));

samples = 1:30;
numOfPeaks = zeros(1, 30);

peaks = findpeaks(A);
numOfPeaks(1) = length(peaks);
for i = 2:30 do
    B = (1/i)*ones(i,1);
    out = filter(B, 1, A);
    peaks = findpeaks(out);
    numOfPeaks(i) = length(peaks);
end for

plot(samples, numOfPeaks);
title('Number of peaks versus N');
```

3 Frequency (Pitch) of the Sound

3.1 Question Description

In this part, I am expected to follow the instructions in **waveexample.m** on **laughter.wav** sample file and explain differences between the applications.

3.2 Exercise 1

In this exercise, I am expected to re-arrange the data so that the frequency is quadrupled.

In this context, I designed new data named **newS** which has length(y)/4 elements. Its elements are the elements of y with index 4*k-3, k less than or equal to length(y).

As I have guessed, resulting sound in this application differs from the original sound in terms of its speed, this one is 4 times faster than original one, since it has higher pitch as a result of I re-arranged the data so that the frequency is quadrupled. As a result data squeezes and sound play so fast which is not understandable.

3.3 Code

Algorithm 2 Exercise 1

```
sizeOfY = size(y,1);
newS = zeros(13159, 1);
i = 1;
while i less than or equal to 13159 do
    newS(i) = y(4*i-3);
    i = i + 1;
end while
sound(newS, Fs);
pause(duration/4 + 2);
```

3.4 Exercise 2

In this exercise, I am expected to re-arrange the data so that the frequency is quadrupled.

In this context, I designed new data named **newSound** which has 2*length(y) elements. For each k, new sound's element with index 2*k-1 is k'th element of y and new sound's element with index 2*k is average of k'th and k+1'th elements of y, k less than length(y).

As I have guessed, resulting sound in this application differs from the original sound in terms of its speed, this one is slower than original one, since it has lower pitch as a result of I re-arranged the data so that the frequency is quadrupled.

3.5 Code

Algorithm 3 Exercise 2

```
newSound = zeros(sizeOfY * 2, 1);
i = 1;
while i less than sizeOfY do
    newSound(2*i-1) = y(i);
    newSound(2*i) = (y(i)+y(i+1))/2;
    i = i + 1;
end while
newSound(2*i-1) = y(i);
newSound(2*i) = y(i);
sound(newS, Fs);
pause(2*duration + 2);
```

3.6 Exercise 3

In this exercise, I am expected to double Fs.

In this context, I only played the sound with frequency 2*Fs. This one differs from the original sound in terms of speed, this one is faster than original one, since I doubled the Fs. We are getting double number of samples in a given time, so sound plays faster.

3.7 Code

Algorithm 4 Exercise 3

```
sound(y, 2*Fs);
pause(duration/2 + 2);
```

3.8 Exercise 4

In this exercise, I am expected yo divide Fs by two.

In this context, I only played the sound with frequency Fs/2. This one differs from the original sound in terms of speed, this one is slower than original one, since I divided the Fs by 2. Additionally, it behaves in a similar manner with the one **Exercise 2.**

3.9 Code

Algorithm 5 Exercise 4

```
1: sound(y, Fs/2);
2: pause(2*duration + 2);
```

4 N-tap Filter

4.1 Question Description

In the last part, I am expected to design an N-tap-filter for alleviating the effects of delayed versions of the sound. I am asked to write a MATLAB script that combines **mike.vaw** and delayed versions of it with **K(100ms).**

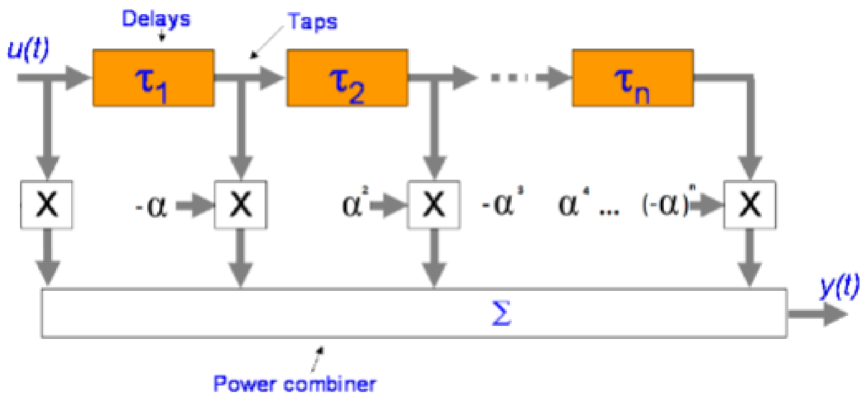


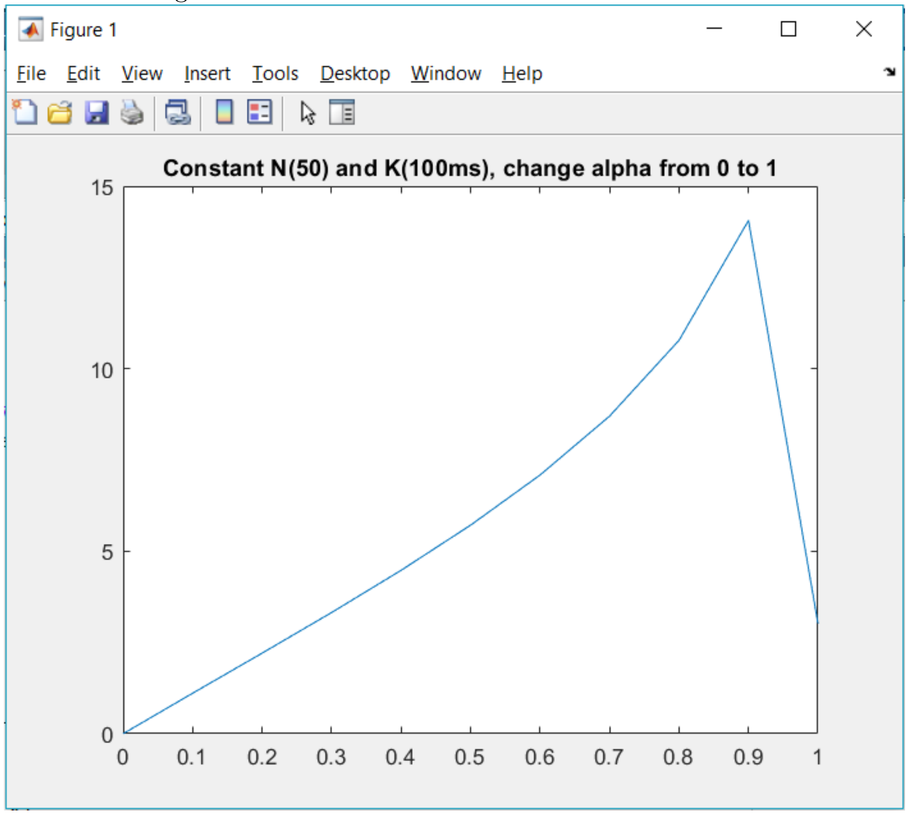
Figure 1: General description of an N-tap filter

Signal to Noise Ratio(SNR) is used as an objective measure for the metric of imperceptibility. Signal to Noise Ratio is a difference metric that is used to calculate the similarity between the original audio signal and the recovered audio signal. It can be calculated using the formula below:

$$SNR(dB) = 10\log \frac{\sum_n I_n^2}{\sum_n (E_n - I_n)^2}$$

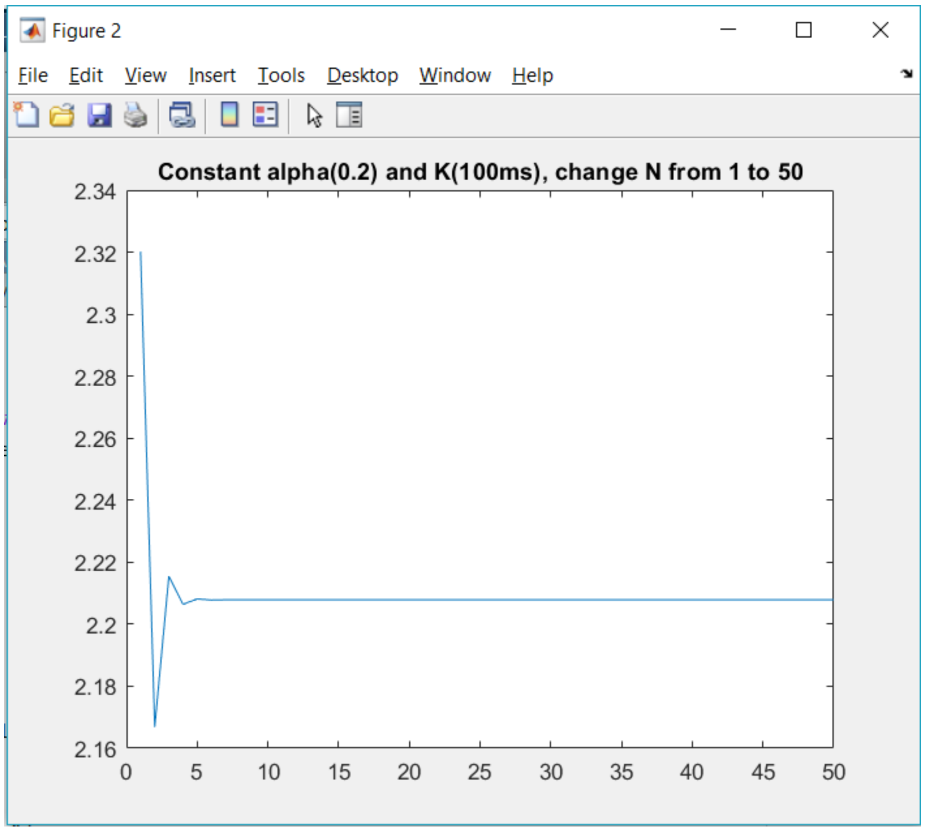
4.2 Output

1. Use constant N and K, change alpha from 0 to 1 and plot SNR of mike.wav and recovered signal.



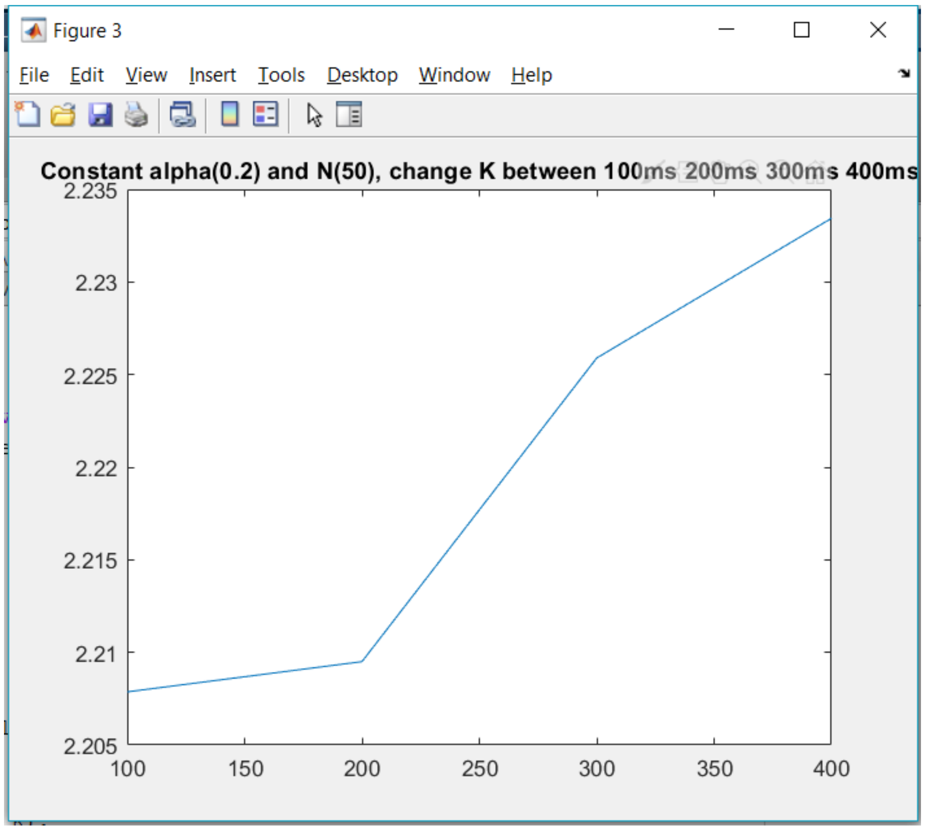
I used N = 50, K = 100ms, and incremented alpha by 0.1.

2. Use constant alpha and K, change N from 1 to 50 and plot SNR of mike.wav and recovered signal.



I used K = 100ms, alpha = 0.2, and incremented N by 1.

3. Use constant alpha and N, change K between 100,200,300,400 milliseconds and plot SNR of mike.wav and recovered signal.



I used N = 50, alpha = 0.2, and incremented K by 100.

4.3 Result

In the first part, which we keep N and K constant and change alpha between 0, 1 increasing by 0.1. For N = 50 and K = 100ms, we can observe that ratio increases as alpha increases until alpha reaches around 0.9s. After that point, as alpha increases ratio decreases simultaneously.

In the second part, which we keep alpha and K constant and change N between 1, 50 increasing by 1. For alpha = 0.2 and K = 100ms, we can observe that ratio oscillates until N reaches around 6. After that point ratio converges to a specific point, which means for those values, increasing N more than 6 has no effect on ratio. So system makes unnecessary computations for N greater than 6, in other words we have unnecessary delay units in our filter.

In the third part, which we keep alpha and N constant and change K between 100, 400 increasing by 100. For alpha = 0.2 and N = 50, we can observe that ratio increases as K increases. As a result we can conclude that, tuning of a signal is more precise with small K values.

From my point of view, SNR values are different for various K, N and alpha values, which means I may get different plots using different constants. Additionally, especially last parts of plots depend on how we implement delay function. There are two main types for implementing delay, one is keeping the array fixed size and appends 0 to first (K * Fs / 1000) indexes, other one is enlarges the array and appends 0 to first (K * Fs / 1000) indexes. I implemented the one that I said firstly.

4.4 Code

```
clc; clear;

hfile = 'mike.wav';
[y, Fs] = audioread(hfile);

K = 100;

N = 50;
alpha = 0:0.1:1;
result1 = zeros(length(alpha), 1);

i = 1;
while i <= length(alpha)
    result1(i) = totalCalculator(y, N, Fs, K, alpha(i));
    i = i + 1;
end

figure;
plot(alpha, result1);
title('Constant N(50) and K(100ms), change alpha from 0 to 1');

alpha = 0.2;
K = 100;
result2 = zeros(50, 1);

i = 1;
while i <= 50
    result2(i) = totalCalculator(y, i, Fs, K, alpha);
    i = i + 1;
end

figure;
plot(1:50, result2);
```

```
title('Constant alpha(0.2) and K(100ms), change N from 1 to 50');

N = 50;
alpha = 0.2;
result3 = zeros(4, 1);

i = 1;
while i <= 4
    result3(i) = totalCalculator(y, N, Fs, 100*i, alpha);
    i = i + 1;
end

figure;
plot(100:100:400, result3);
title('Constant alpha(0.2) and N(50), change K between 100ms 200ms 300ms 400ms')


function res = totalCalculator(original, N, Fs, K, alpha)
    shifted = delay(original, Fs, K);
    with_delay = original + shifted;
    filt = nTapFilt(with_delay, N, Fs, K, alpha);
    SNR = snrCalculator(original, filt);
    res = SNR;
end

function val = snrCalculator(original, recovered)
    x = 0;
    y = 0;
    i = 1;
    while i<=length(original)
       x = x + original(i) * original(i);
       y = y + (recovered(i) - original(i)) * (recovered(i) - original(i));
       i = i + 1;
    end

    val = 10 * log10(x/y);
end

function filt = nTapFilt(signal, N, Fs, K, alpha)
    filt = signal;
    shifted = signal;
    i = 1;
    while i <= N
        shifted = delay(shifted, Fs, K);
        filt = filt + shifted.*((-1)*alpha)^i;
        i = i + 1;
    end
end

function shifted = delay(signal, Fs, K)
    first = (Fs * K) / 1000;
    shifted = [zeros(first, 1); signal(1:length(signal)-first)];
end
```

5 Conclusion

In this homework, I implemented some simple time domain exercises using MATLAB. I was expected to answer 3 questions, named Advanced Peak Finder, Frequency (Pitch) of the Sound, N-tap Filter, respectively. I implemented each questions to separate MATLAB scripts.

Thanks to this project, I learned how to code some simple functionalities in MATLAB like if, for, while etc. Beside of those, I gained experience in implementing time domain exercises using computer. Additionally, due to the this project, now I have more knowledge about filtering since I had a opportunity to have hands-on experience about the facts that I learned during lectures.