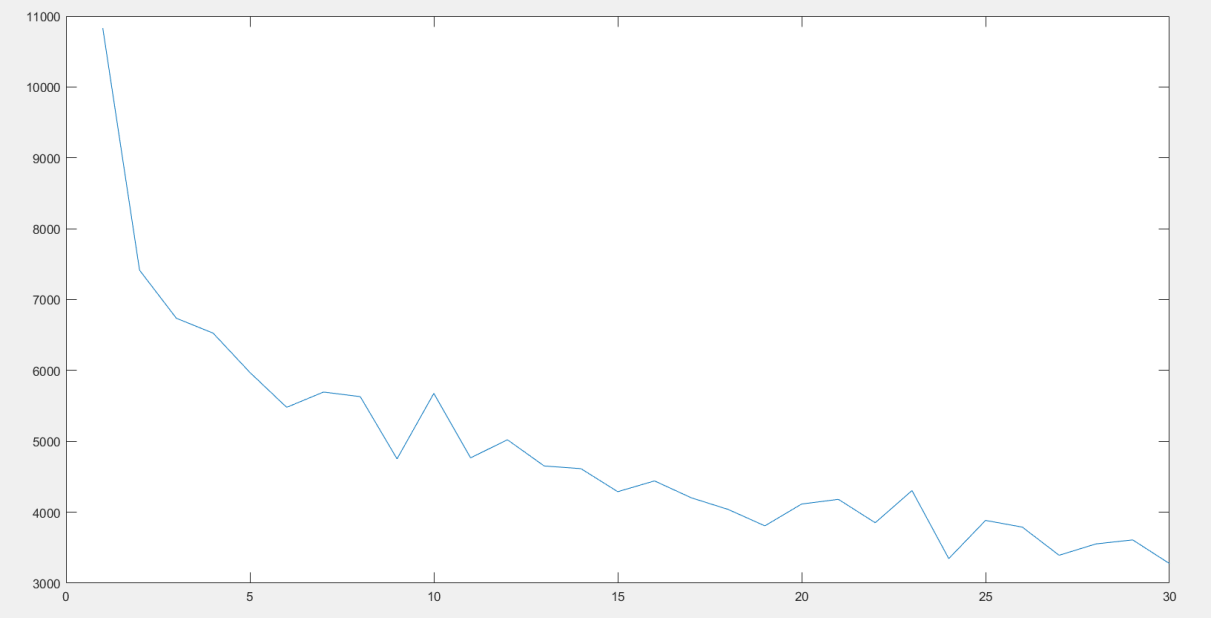
CmpE 362- Homework 2
BARAN DENİZ KORKMAZ
2015400183

PROBLEM 1

```
%{
    Question 1
%}
%{
   Clears old variables from console and workspace to avoid some
   possible errors.
%}
clear;clc;

% Sets the file name in order to be able to read it via csvread command.
file='exampleSignal.csv';

% Reads the csv file. PS: csvread function caused some numerical
% anormalities, therefore in the project, the load function has been used.
mySignal=load(file);

% y1 is an alias of the signal.
y1=mySignal;

% Below are the filter coefficients for the moving average filters.
b2=[1/2 1/2];
b3=[1/3 1/3 1/3];
b4=[1/4 1/4 1/4 1/4];
b5=[1/5 1/5 1/5 1/5 1/5];
b6=[1/6 1/6 1/6 1/6 1/6 1/6];
b7=[1/7 1/7 1/7 1/7 1/7 1/7 1/7];
b8=[1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8];
b9=[1/9 1/9 1/9 1/9 1/9 1/9 1/9 1/9 1/9];
b10=[1/10 1/10 1/10 1/10 1/10 1/10 1/10 1/10 1/10 1/10];
b11=[1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11];
b12=[1/12 1/12 1/12 1/12 1/12 1/12 1/12 1/12 1/12 1/12 1/12 1/12];
b13=[1/13 1/13 1/13 1/13 1/13 1/13 1/13 1/13 1/13 1/13 1/13 1/13 1/13];
b14=[1/14 1/14 1/14 1/14 1/14 1/14 1/14 1/14 1/14 1/14 1/14 1/14 1/14
1/14];
b15=[1/15 1/15 1/15 1/15 1/15 1/15 1/15 1/15 1/15 1/15 1/15 1/15 1/15 1/15
1/15];
b16=[1/16 1/16 1/16 1/16 1/16 1/16 1/16 1/16 1/16 1/16 1/16 1/16 1/16 1/16
1/16 1/16];
b17=[1/17 1/17 1/17 1/17 1/17 1/17 1/17 1/17 1/17 1/17 1/17 1/17 1/17 1/17
1/17 1/17 1/17];
b18=[1/18 1/18 1/18 1/18 1/18 1/18 1/18 1/18 1/18 1/18 1/18 1/18 1/18 1/18
1/18 1/18 1/18 1/18];
b19=[1/19 1/19 1/19 1/19 1/19 1/19 1/19 1/19 1/19 1/19 1/19 1/19 1/19 1/19
1/19 1/19 1/19 1/19 1/19];
b20=[1/20 1/20 1/20 1/20 1/20 1/20 1/20 1/20 1/20 1/20 1/20 1/20 1/20 1/20
1/20 1/20 1/20 1/20 1/20 1/20];
b21=[1/21 1/21 1/21 1/21 1/21 1/21 1/21 1/21 1/21 1/21 1/21 1/21 1/21 1/21
1/21 1/21 1/21 1/21 1/21 1/21 1/21];
b22=[1/22 1/22 1/22 1/22 1/22 1/22 1/22 1/22 1/22 1/22 1/22 1/22 1/22 1/22
1/22 1/22 1/22 1/22 1/22 1/22 1/22 1/22];
```

```
b23=[1/23 1/23 1/23 1/23 1/23 1/23 1/23 1/23 1/23 1/23 1/23 1/23 1/23 1/23
1/23 1/23 1/23 1/23 1/23 1/23 1/23 1/23 1/23 1/23 1/23];
b24=[1/24 1/24 1/24 1/24 1/24 1/24 1/24 1/24 1/24 1/24 1/24 1/24 1/24 1/24
1/24 1/24 1/24 1/24 1/24 1/24 1/24 1/24 1/24 1/24 1/24];
b25=[1/25 1/25 1/25 1/25 1/25 1/25 1/25 1/25 1/25 1/25 1/25 1/25 1/25 1/25
1/25 1/25 1/25 1/25 1/25 1/25 1/25 1/25 1/25 1/25 1/25];
b26=[1/26 1/26 1/26 1/26 1/26 1/26 1/26 1/26 1/26 1/26 1/26 1/26 1/26 1/26
1/26 1/26 1/26 1/26 1/26 1/26 1/26 1/26 1/26 1/26 1/26];
b27=[1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27
1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27 1/27];
b28=[1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28
1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28 1/28];
b29=[1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29
1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29 1/29
1/29];
b30=[1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30
1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30 1/30
1/30];

% Below are the resulting signals after each moving average filter
% operation.
y2=conv(b2, mySignal);
y3=conv(b3, mySignal);
y4=conv(b4, mySignal);
y5=conv(b5, mySignal);
y6=conv(b6, mySignal);
y7=conv(b7, mySignal);
y8=conv(b8, mySignal);
y9=conv(b9, mySignal);
y10=conv(b10, mySignal);
y11=conv(b11, mySignal);
y12=conv(b12, mySignal);
y13=conv(b13, mySignal);
y14=conv(b14, mySignal);
y15=conv(b15, mySignal);
y16=conv(b16, mySignal);
y17=conv(b17, mySignal);
y18=conv(b18, mySignal);
y19=conv(b19, mySignal);
y20=conv(b20, mySignal);
y21=conv(b21, mySignal);
y22=conv(b22, mySignal);
y23=conv(b23, mySignal);
y24=conv(b24, mySignal);
y25=conv(b25, mySignal);
y26=conv(b26, mySignal);
y27=conv(b27, mySignal);
y28=conv(b28, mySignal);
y29=conv(b29, mySignal);
y30=conv(b30, mySignal);

%The following part calculates the number of peaks for each signal.
numOfPeaks1=numel(findpeaks(y1));
numOfPeaks2=numel(findpeaks(y2));
numOfPeaks3=numel(findpeaks(y3));
numOfPeaks4=numel(findpeaks(y4));
numOfPeaks5=numel(findpeaks(y5));
numOfPeaks6=numel(findpeaks(y6));
numOfPeaks7=numel(findpeaks(y7));
numOfPeaks8=numel(findpeaks(y8));
numOfPeaks9=numel(findpeaks(y9));
```

```
numOfPeaks10=numel(findpeaks(y10));
numOfPeaks11=numel(findpeaks(y11));
numOfPeaks12=numel(findpeaks(y12));
numOfPeaks13=numel(findpeaks(y13));
numOfPeaks14=numel(findpeaks(y14));
numOfPeaks15=numel(findpeaks(y15));
numOfPeaks16=numel(findpeaks(y16));
numOfPeaks17=numel(findpeaks(y17));
numOfPeaks18=numel(findpeaks(y18));
numOfPeaks19=numel(findpeaks(y19));
numOfPeaks20=numel(findpeaks(y20));
numOfPeaks21=numel(findpeaks(y21));
numOfPeaks22=numel(findpeaks(y22));
numOfPeaks23=numel(findpeaks(y23));
numOfPeaks24=numel(findpeaks(y24));
numOfPeaks25=numel(findpeaks(y25));
numOfPeaks26=numel(findpeaks(y26));
numOfPeaks27=numel(findpeaks(y27));
numOfPeaks28=numel(findpeaks(y28));
numOfPeaks29=numel(findpeaks(y29));
numOfPeaks30=numel(findpeaks(y30));

% The matrix a consists of the numerical values of the peaks for each
% signal.
a=[numOfPeaks1 numOfPeaks2 numOfPeaks3 numOfPeaks4 numOfPeaks5 numOfPeaks6
numOfPeaks7 numOfPeaks8 numOfPeaks9 numOfPeaks10 numOfPeaks11 numOfPeaks12
numOfPeaks13 numOfPeaks14 numOfPeaks15 numOfPeaks16 numOfPeaks17
numOfPeaks18 numOfPeaks19 numOfPeaks20 numOfPeaks21 numOfPeaks22
numOfPeaks23 numOfPeaks24 numOfPeaks25 numOfPeaks26 numOfPeaks27
numOfPeaks28 numOfPeaks29 numOfPeaks30];

% The matrix n denotes the length of the moving average filter.For example,
%  n=1 denotes the original signal.
n=[1:30]

% Plots the number of peaks for the corresponding length of the moving
% average filter.
plot(n,a);
```



On Problem 1

In the code, i simply created the corresponding coefficients for each requested length of the moving average filter, then i implemented the convolution by the method conv for each corresponding coefficients. By the numel function, i could have taken the number of peaks for each signal. Then i plotted the number of peaks vs the length of the moving average filter.

As the length of the moving average filter, the graph of the signal becomes flatter so that we observe much less the number of peaks as the length increases. The fact can easily be observed from the plot as expected.

QUESTION 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

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

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

% In order to be able to quadruple the frequency that is getting a
% frequenct multiplied by four, the following code piece simply changes the
% pitch. Then the duration of the pause should be determined by the
% one-forth of the duration of the normal signal since the frequency is
% quadrupled.
yQuadrupled=y(1:4:end);
sound(yQuadrupled, Fs);
pause(duration/4+2)


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

%In order to halve the frequency, the following desing enables us to expand
%the signal in a way that the frequency becomes half of the old value. The
%design simply creates a new vector of size of the double of the length of
%the signal, and it puts the first index value of the signal into the first
%two index of the new signal. The process goes on until the all n elements
```

```
%are distributed into 2n indexes of the new signal. The new signal is a
%good model for the signal with a frequency that is halved. Since the
%frequency is halved, the pause duration will be determined by the double
%of the old duration.

col=2*size(y,1);
yHalved = ones(col,1);

for index = 1 : size(y, 1)
    yHalved(2 * index) = y(index);
    yHalved(2 * index - 1) = y(index);
end

sound(yHalved, Fs);
pause(2*duration + 2)

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

%The following code piece multiplies Fs by 2 and sounds it. The duration is
%arranged to adapt into the new frequency.
FsDoubled=Fs*2;
sound(y,FsDoubled);
pause(duration/2 + 2)

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

%The following code piece divides Fs by 2 and sounds it. The duration is
%arranged to adapt into the new frequency.
FsHalved=Fs/2;
sound(y,FsHalved);
```

On Problem 2

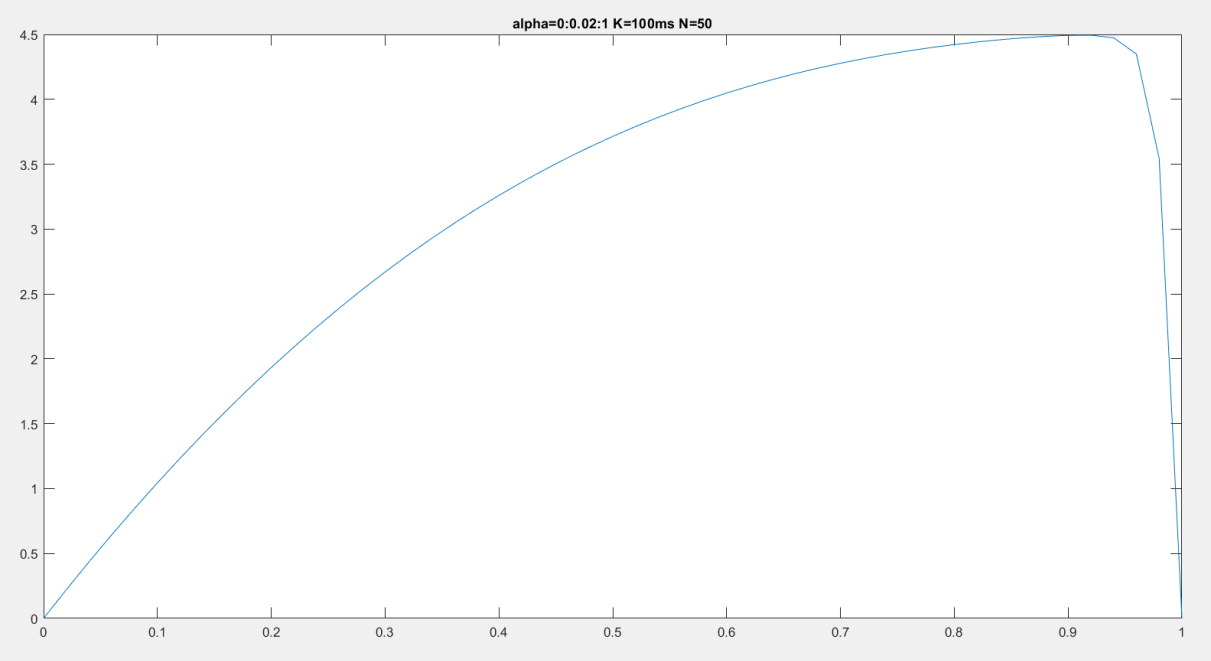
In the problem 2, the operation of the quadrupling the frequency is just the same as changing the pitch. The second part requires much more attention and the way the second part is implemented is explained by comments in detail. The third and fourth parts of the question is carried out by changing the value of Fs arithmetically. The resulting sounds behave as expected. In the first part, we have a 4x playing record, besides in the second part it's speed is half of the original signal. In the third and fourth parts, since we changed the value of Fs, not the signals themselves, we still can analyze the effect of the frequency by the sound method.

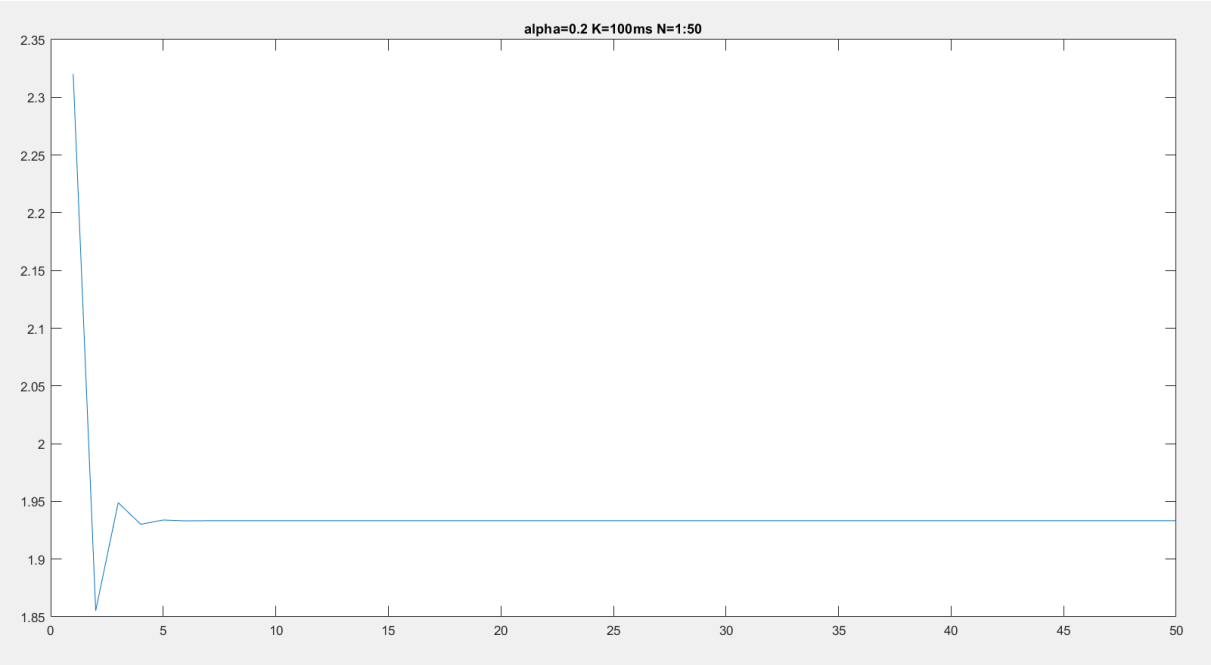
QUESTION 3

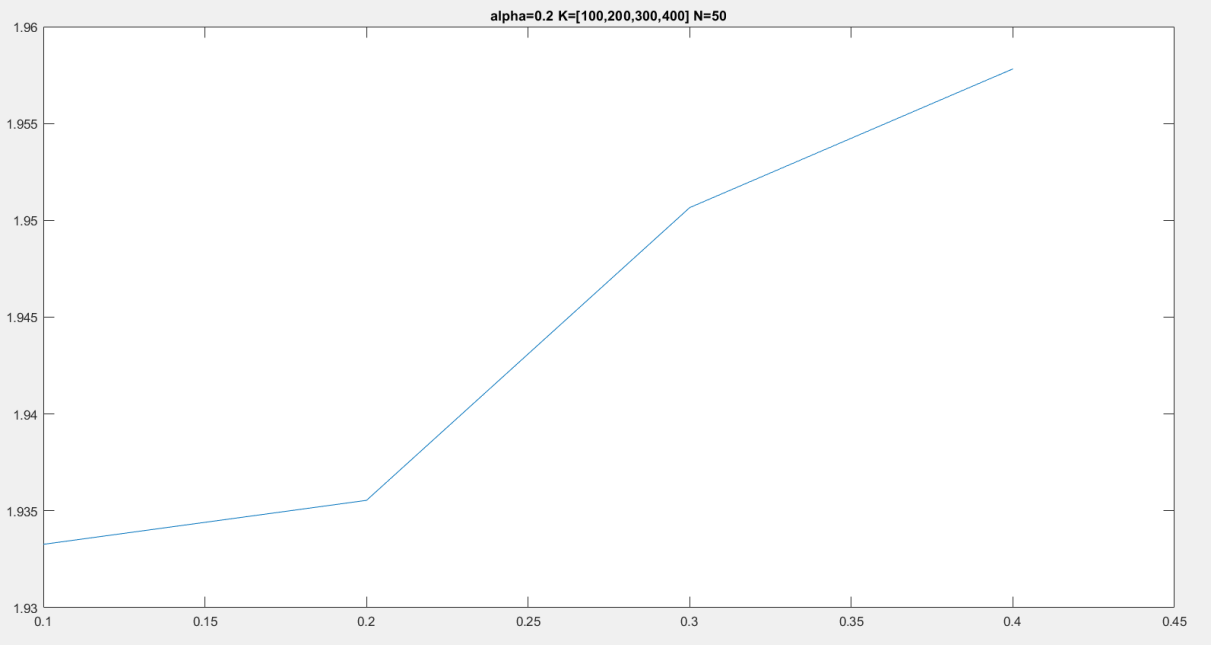
```
%{
    Question 3
%}

%{
   Clears old variables from console and workspace to avoid some
   possible errors.
%}
clear; clc;
clc; clear;

hfile = 'mike.wav'; % This is a string, corresponding to the filename

% Clears the variables of y and Fs to avoid possible errors.
clear y Fs

% The audioread function reads the audio file and returns the signal and
the
% frequency of the signal.
[y, Fs] = audioread(hfile);

% The default value of the K is determined as 100 ms (0.1 s).
K=0.1;

% The combined mike.wav and the delayed version of it.
yCombined=y+delayseq(y,Fs*K);

%OUTPUT1
%N and K are constants, a is changed from 0 to 1. Since N-Tap Filter will
be
%implemented for 50 times, the range between two alpha values of 0.02
%is appropriate. The constant value of N is determined as 50 and the
constant value of K will
%be taken as 100 ms.
N=50;
alpha=(0:0.02:1);

% Since the filter will be operated for each alpha value, the length of the
% array that holds the corresponding snr values is equal to the length of
% the alpha array.
sizeOfAlphaArray=length(alpha);
snrArray1=zeros(sizeOfAlphaArray,1);

% For each alpha values, nTapFilter operates and the calculated snr value
% of the original and the recovered signal is added into the corresponding
% index at the snr array.
for i=1:sizeOfAlphaArray
    filterResult1=nTapFilter(yCombined,Fs,K,N,alpha(i));
    snrArray1(i)=calculateSNR(y,filterResult1);
end

% Since we have the snr values for the corresponding alpha values, we are
% ready to plot.
figure('Name','Output 1')
plot(alpha,snrArray1);
title('alpha=0:0.02:1 K=100ms N=50');

% Plays the original sound.
```

```
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

% Plays the last signal that is created in the loop (for alpha=1) in order
% to be able to compare the recovered sound to the original sound.
sound(filterResult1,Fs);
pause(duration+2);

%OUTPUT 2
%Alpha and K are constants, N is changed from 1 to 50. The alpha value is
%taken as 0.2 and the K value is taken as 100 ms.
alphaValue2=0.2;
snrArray2=zeros(50,1);
%Since N is changed from 1 to 50, the following line creates the array of N
%values.
NArray=1:50;

%We'll be moving one by one for each element of the NArray and carry out
%the necessary operations in the following for loop. In the loop, first, we
%get the filtered signal and then the signal generated by the n-tap filter
%and the original signal will be compared in the calculate SNR method.
for N=1:50
    filterResult2=nTapFilter(yCombined,Fs,K,N,alphaValue2);
    snrArray2(N)=calculateSNR(y,filterResult2);
end

% Since we have the snr values for the corresponding N values, we are
% ready to plot.
figure('Name','Output 2')
plot(NArray,snrArray2);
title('alpha=0.2 K=100ms N=1:50');

% Plays the last signal that is created in the loop (for N=50) in order
% to be able to compare the recovered sound to the original sound.
sound(filterResult2, Fs);
pause(duration + 2);

%OUTPUT 3
%Alpha and N are constants, K is changed between 100,200,300,400 ms. The
%constant alpha value is taken as 0.2, and the constant N value is 50.
alphaValue3=0.2;
snrArray3=zeros(4,1);
N=50;
KArray=[0.1 0.2 0.3 0.4];

%For each K values, the delayseq method creates the delayed version of the
%original signal, then the combined version will be operated in the filter
%with the corresponding K value. Finally the SNR value for the original
%signal and the recovered signal will be calculated.
for i=1:length(KArray)
    delayedSignal3=delayseq(y,Fs*KArray(i));
    yCombinedCase3=delayedSignal3+y;
    filterResult3=nTapFilter(yCombinedCase3,Fs,KArray(i),N,alphaValue3);
    snrArray3(i)=calculateSNR(y,filterResult3);
end

% Since we have the snr values for the corresponding K values, we are
% ready to plot.
```

```
figure('Name','Output 3');
plot(KArray,snrArray3);
title('alpha=0.2 K=[100,200,300,400] N=50');

% Plays the last signal that is created in the loop (for K=400 ms) in order
% to be able to compare the recovered sound to the original sound.
sound(filterResult3, Fs);
pause(duration + 2)

%N-Tap Filter will be implemented by the following method. The method
%simply adds the original signal with each delayed version of it with
%a multiplication of alpha according to the given index value (i). Then
%finally returns the filteredSignal.
function filteredSignal=nTapFilter(y,Fs,K,N,alpha)
    newSignal=y;
    for i=1:N
        delayedY=delayseq(y,Fs*K);
        newSignal=newSignal + (alpha*(-1))^i * delayedY;
    end
    filteredSignal=newSignal;
end

%calculateSNR function simply returns the snr value of the original audio
%and the recovered audio.
function snrValue=calculateSNR(originalAudio,recoveredAudio)
    snrValue=10*log10(sum(originalAudio.^2) ./ sum((recoveredAudio-
originalAudio).^2));
end
```







On Problem 3

In the problem 3, we built an N-Tap Filter that processes the given signal with parameters alpha, K, and N. alpha denotes the value of the coefficient at each step to be multiplied with the given signal. K denotes the delay amount. N denotes the number of delay operations. The way the filter is implemented and the way requested outputs generated are explained by the comments in detail. In the plots, we graphed the values of SNR according to the alpha,K, and N values at each output. Since the higher SNR value denotes a better processes signal we have, we can observe that alpha value is the main element of the precision for the recovered signal. The second graph shows the relationship between SNR value and the N values. Since the SNR values deviates between 1.85 and 2.35, we can conclude that the value of alpha has a larger impact to get a larger SNR value. Finally, the relationship

between the value of K and the SNR values is plotted in the third graph. As the delay amount increases, we get a higher SNR value. Therefore a better processed recovered signal we have.