Cmpe362 Hw2 Report

Sadullah Gültekin

Spring 2019

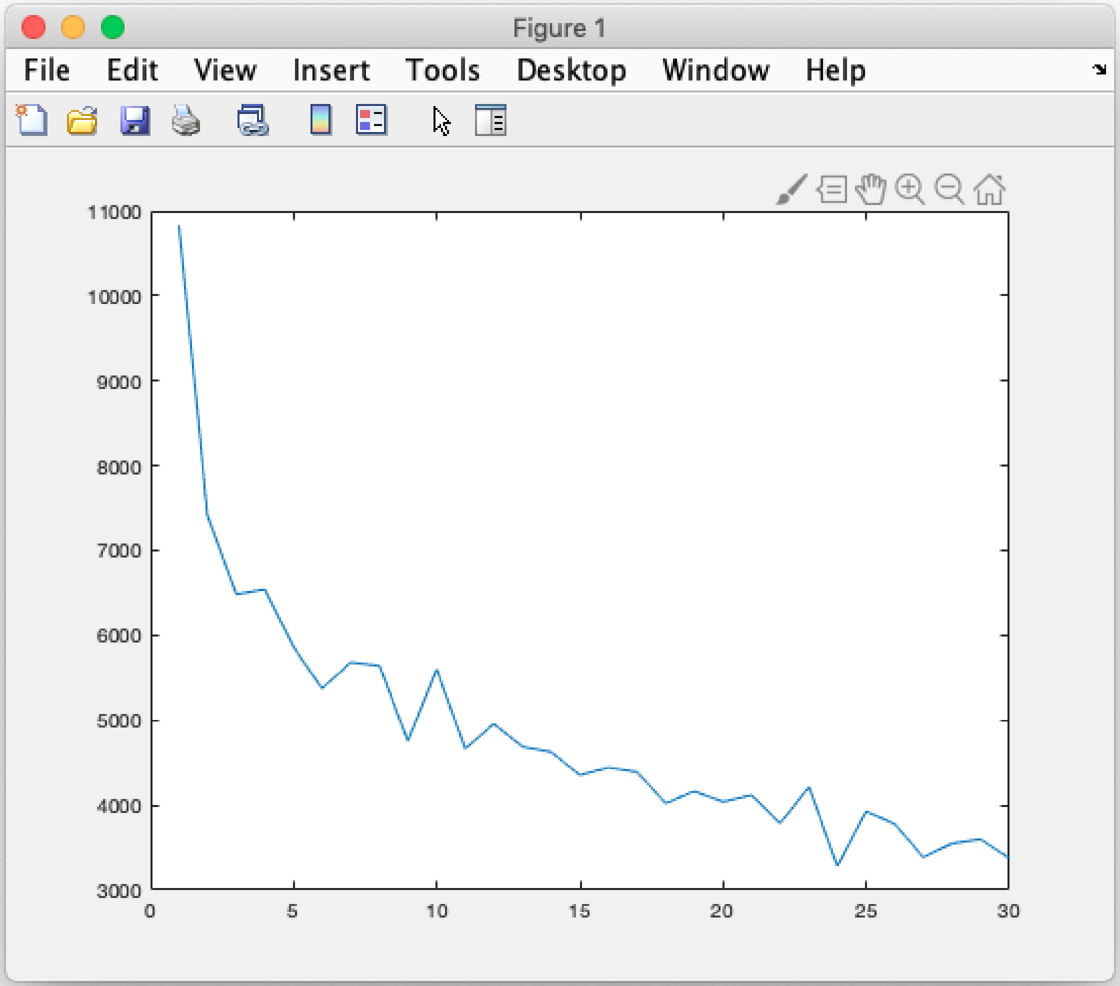
1 Question1

Code Explanation

```
data = csvread('exampleSignal.csv');
A = [];

for i = 1:30
    peaks = findpeaks(MovingAverageFilter(data,i));
    A = [A numel(peaks)];
end

figure(1)
plot(A)

function [filteredNumbers] = MovingAverageFilter(originalNumbers, N)
    filteredNumbers = originalNumbers;
    for i = 1:length(originalNumbers)
        a = 0;
        for j = i-N+1:i
            if j > 0
                a = a + originalNumbers(j);
            end
        end
        filteredNumbers(i) = a/N;
    end
end
```

I read the example signal into the data variable. Then started to a loop that counts until 30. In each iteration I applied a moving average filter to the data. Each time I used a different N value. Then found the peaks using built-in findpeaks function. I collected the number of peaks in each iteration then plot them. In matlab there are built in function that can be used as mooving average filter, but at first, I didn't know them, and wrote the moving average filter function. After a time I realized the built- in function, but decided not to use them.

Code Result



As we can see from the figure, when we increase the N, the signal becomes more smooth and the number of peaks decreases.

2 Question2

Code Explanation

```
hfile = 'laughter.wav';
clear y Fs

[y, Fs] = audioread(hfile);
duration = numel(y) / Fs;

% EXERCISE I

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

% EXERCISE II

x = zeros(1,length(y)*2);
for i = 1:length(x)
    if mod(i,2) == 1
        x(1,i) = y(floor(i/2)+1);
    else
        x(1,i) = y(floor(i/2));
    end
end

sound(x, Fs);
pause(duration + 7)

% EXERCISE III

newFs = Fs * 2;
sound(y, newFs);
pause(duration)

% EXERCISE IV

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

I took the input using audioread function, which gives the input signal and frequency of it. There are 4 exercises that needed to be implemented. The first one is to increase the frequency by 4 doing by manipulating the data. To achieve that, I took one of every four element and deleted the rest. In the second exercise, I needed to do the same thing as exercise 1, but this time, instead of increasing it, I decreased it by two. To do that I duplicated every item in the signal function. As a result, for example if I had a signal like "signal", now I have "ssiiggnnaall". Which looks like i doubled down the frequency. In the thirt and forth excersize, I just changed the value of the frequency, which is just an integer.

Code Result

Excersize 1 became -¿ 4 times faster
Excersize 1 became -¿ 2 times slower
Excersize 1 became -¿ 2 times faster
Excersize 1 became -¿ 2 times slower

3 Question3

Code Explanation

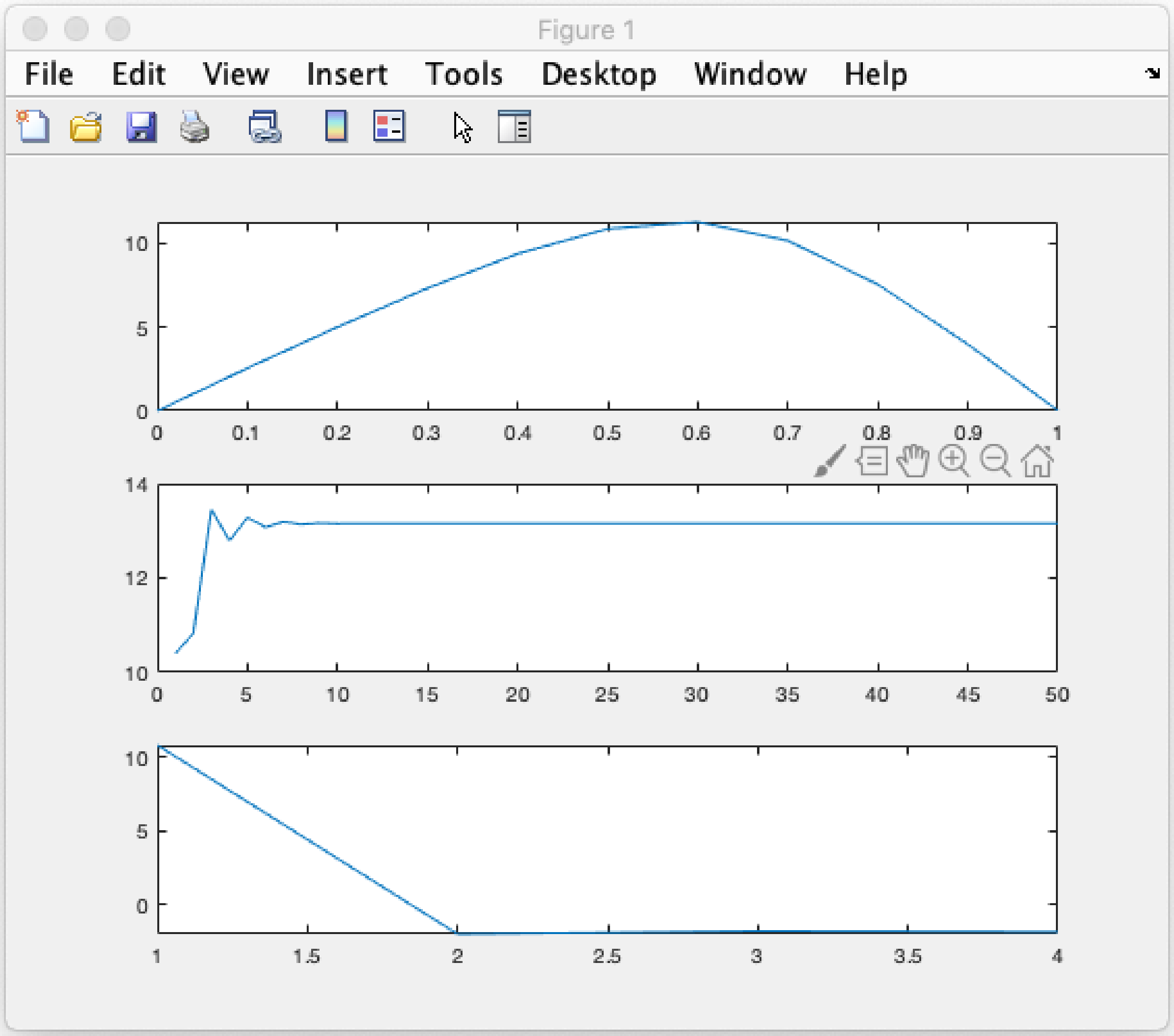
```
clear
hfile = 'mike.wav';

[signal, Fs] = audioread(hfile);
K = 0.1;
signal = [signal ; zeros((Fs*K),1)];
originalSignal = signal;
signal = signal + delayseq(signal,K,Fs);

%%{
% 1 %
allSNRs = [];
N = 2;
K = 0.1;
for alpha = 0:0.1:1
    allSNRs = [allSNRs snr(originalSignal,nTapFilter(signal,alpha,K,N,Fs))];
end
subplot(3,1,1)
plot(0:0.1:1,allSNRs)
%}


%%{
% 2 %
allSNRs = [];
alpha = 0.5;
K = 0.1;
for N = 1:50
    allSNRs = [allSNRs snr(originalSignal,nTapFilter(signal,alpha,K,N,Fs))];
end
subplot(3,1,2)
plot(1:50,allSNRs)
%}
```

```
%%{
% 3 %
allSNRs = [];
alpha = 0.5;
N = 2;
for K = 0.1:0.1:0.4
    allSNRs = [allSNRs snr(originalSignal,nTapFilter(signal,alpha,K,N,Fs))];
end
subplot(3,1,3)
plot(1:4,allSNRs)
%}

function result = nTapFilter(signal,alpha,K,N,fs)
    result = signal;
    for i=1:N
        signal = delayseq(signal,fs,K);
        result = result + ((-1)*alpha)^i * signal;
    end
end

function snrValue = snr(I, E)
    snrValue = 10*log(sum(I.^2)/sum((E-I).^2));
end
```

,

In this question I read the input again with audioread function. Then put it a delay by delayseq function. When I did that, I lost some portion of the data. To prevent the data lost, I enlarged the original function with zeros. Next I used N-tap-filter with different values. At first I made N = 2 and K = 0.1 constant and changed alpha from 0 to 1. Each time I run the n-tap-filter, I calculated a SNR value, and collected them in an array. I made the same process to see the effect of the K and N in the filtering. I took alpha = 0.5 and K = 0.1 as constant, and iterated N from 1 to 50 Finally, I took alpha = 0.5 and N = 2 as constant and iterated K for 0.1 to 0.4. In the very end, I plot them to the same figure, to see them at the same time
I also wrote two helper function, one for nTapFilter and one for SNR value calculation.

Code Result



As we can see from the figure, changing alpha can effect the result considerably. As we increase the alpha, the recovery level of sound will increase to some point. But when we make it to 1, the recovery will be worse.

When it comes to the changing N, after a point it doesn't effect the result. But keeping it in a reasonable level would be wiser, because it takes time to make the computations. So, choosing N like 6 or 7 would be enough to take a good result.

In the third figure, we can observe that, when there is a delay more than a some point, the recovery becomes difficult, or even impossible. In my case, when we apply a delay more than 0.2 sec, it is impossible to recover the original sound.