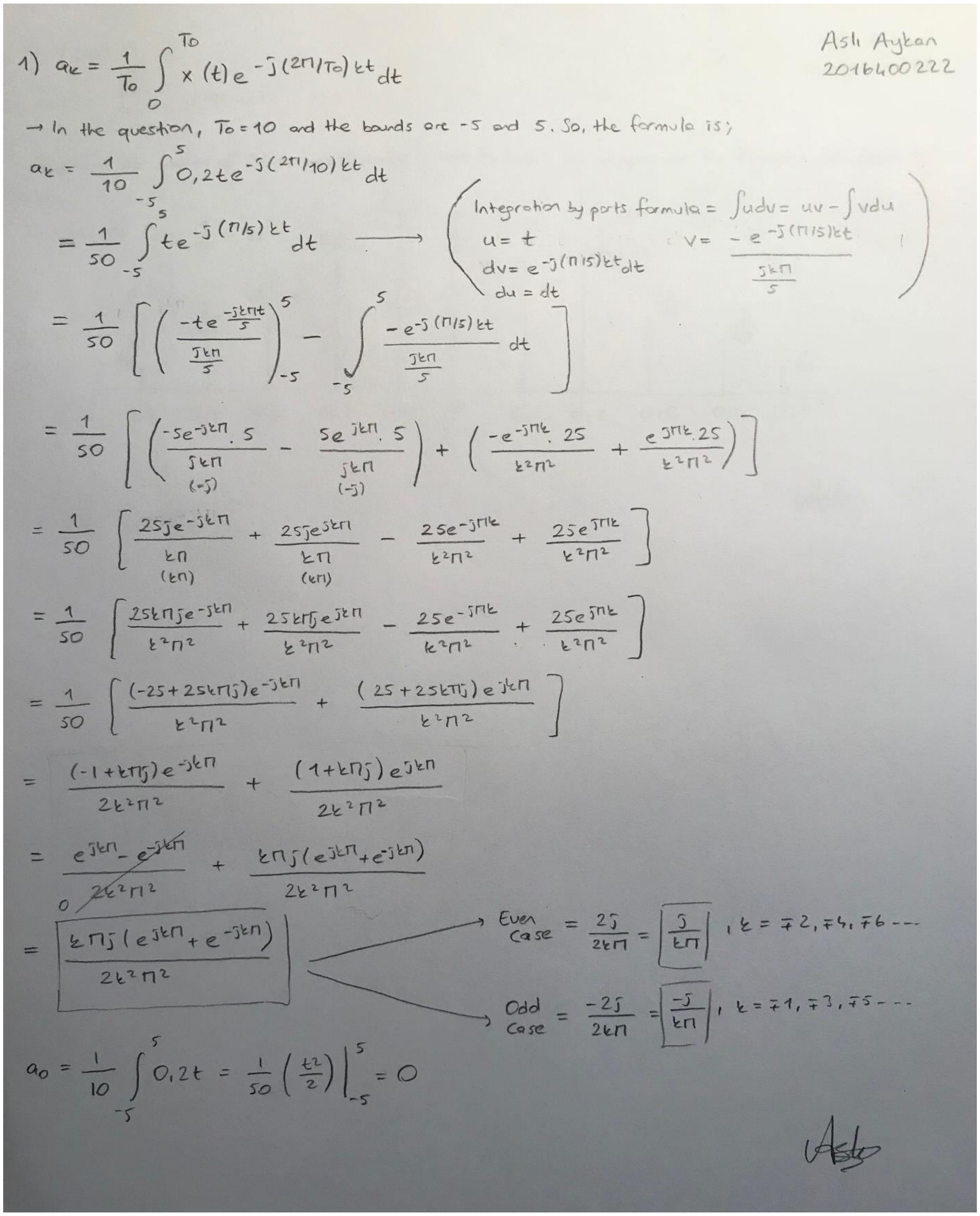
Aslı Aykan
2016400222

CMPE 362 | HW 1

1.1)



Aslı Aykan
2016400222

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

→ In the question, $T_0 = 10$ and the bounds are -5 and 5. So, the formula is;

$$a_k = \frac{1}{10}\int_{-5}^{5} 0{,}2te^{-j(2\pi/10)kt}dt$$

$$= \frac{1}{50}\int_{-5}^{5} te^{-j(\pi/5)kt}dt \longrightarrow \left(\begin{array}{ll} \text{Integration by parts formula} = \int u\,dv = uv - \int v\,du & \\ u = t & v = \frac{-e^{-j(\pi/5)kt}}{\frac{jk\pi}{5}} \\ dv = e^{-j(\pi/5)kt}dt & \\ du = dt & \end{array}\right)$$

$$= \frac{1}{50}\left[\left(\frac{-te^{\frac{-jk\pi t}{5}}}{\frac{jk\pi}{5}}\right)\Bigg|_{-5}^{5} - \int_{-5}^{5}\frac{-e^{-j(\pi/5)kt}}{\frac{jk\pi}{5}}dt\right]$$

$$= \frac{1}{50}\left[\left(\frac{-5e^{-jk\pi}.5}{\underset{(-j)}{jk\pi}} - \frac{5e^{jk\pi}.5}{\underset{(-j)}{jk\pi}}\right) + \left(\frac{-e^{-j\pi k}.25}{k^2\pi^2} + \frac{e^{j\pi k}.25}{k^2\pi^2}\right)\right]$$

$$= \frac{1}{50}\left[\frac{25je^{-jk\pi}}{\underset{(k\pi)}{k\pi}} + \frac{25je^{jk\pi}}{\underset{(k\pi)}{k\pi}} - \frac{25e^{-j\pi k}}{k^2\pi^2} + \frac{25e^{j\pi k}}{k^2\pi^2}\right]$$

$$= \frac{1}{50}\left[\frac{25k\pi je^{-jk\pi}}{k^2\pi^2} + \frac{25k\pi je^{jk\pi}}{k^2\pi^2} - \frac{25e^{-j\pi k}}{k^2\pi^2} + \frac{25e^{j\pi k}}{k^2\pi^2}\right]$$

$$= \frac{1}{50}\left[\frac{(-25+25k\pi j)e^{-jk\pi}}{k^2\pi^2} + \frac{(25+25k\pi j)e^{jk\pi}}{k^2\pi^2}\right]$$

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

$$= \underset{0}{\cancel{\frac{e^{jk\pi} - e^{-jk\pi}}{2k^2\pi^2}}} + \frac{k\pi j(e^{jk\pi}+e^{-jk\pi})}{2k^2\pi^2}$$

$$= \boxed{\frac{k\pi j(e^{jk\pi}+e^{-jk\pi})}{2k^2\pi^2}}$$

→ Even Case $= \frac{2j}{2k\pi} = \boxed{\frac{j}{k\pi}}$, $k = \mp 2, \mp 4, \mp 6 \ldots$

→ Odd Case $= \frac{-2j}{2k\pi} = \boxed{\frac{-j}{k\pi}}$, $k = \mp 1, \mp 3, \mp 5 \ldots$

$$a_0 = \frac{1}{10}\int_{-5}^{5} 0{,}2t = \frac{1}{50}\left(\frac{t^2}{2}\right)\Bigg|_{-5}^{5} = 0$$

[illegible]

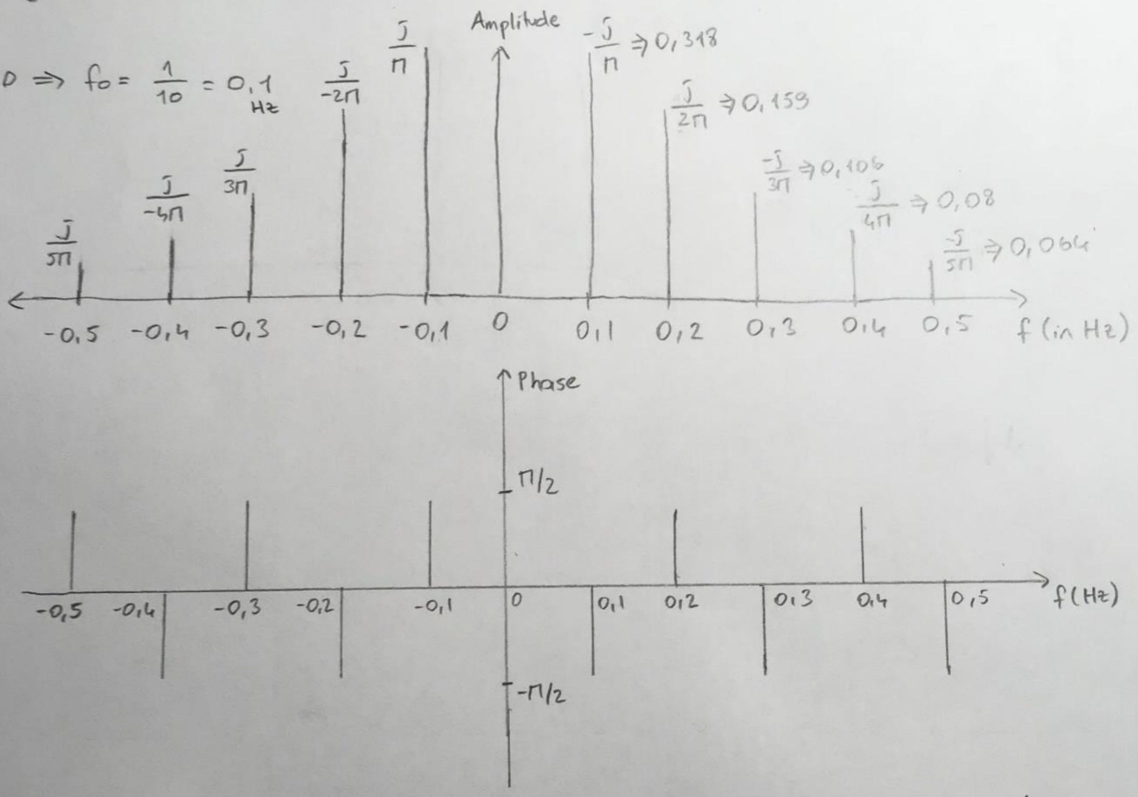
1) (cont.)

Aslı Aykan
2016400222

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{j2\pi k f_0 t}$$

By putting the values of Fourier coefficients to this formula, we can generate the formulas for harmonic sums.

$T_0 = 10 \Rightarrow f_0 = \frac{1}{10} = 0{,}1$ Hz



[illegible]

I checked the correctness of my resulting spectrums with the help of in fseriesdemo library, as suggested in the question.

1.2)

Code & Comments:

```
fs = 100;  % fs is given
t = -10:1/fs:10-1/fs;  % time vector is given

y = mod(0.2*t+1,2)-1;  % the given signal function (written in periodic way)
% 3 Harmonic Sum
y1 = (1i/pi)*exp(-1i*pi*t/5) + (-1i/pi)*exp(1i*pi*t/5)...
   + (1i/(2*pi))*exp(1i*2*pi*t/5)+ (-1i/(2*pi))*exp(-1i*2*pi*t/5)...
   + (1i/(3*pi))*exp(-1i*pi*3*t/5) + (-1i/(3*pi))*exp(1i*pi*3*t/5);
```

```
% 5 Harmonic Sum
y2 = (1i/pi)*exp(-1i*pi*t/5) + (-1i/pi)*exp(1i*pi*t/5)...
   + (1i/(2*pi))*exp(1i*2*pi*t/5)+ (-1i/(2*pi))*exp(-1i*2*pi*t/5)...
   + (1i/(3*pi))*exp(-1i*pi*3*t/5) + (-1i/(3*pi))*exp(1i*pi*3*t/5)...
   + (1i/(4*pi))*exp(1i*pi*4*t/5) + (-1i/(4*pi))*exp(-1i*pi*4*t/5)...
   + (1i/(5*pi))*exp(-1i*pi*5*t/5) + (-1i/(5*pi))*exp(1i*pi*5*t/5);
% 15 Harmonic Sum
y3 = (1i/pi)*exp(-1i*pi*t/5) + (-1i/pi)*exp(1i*pi*t/5)...
   + (1i/(2*pi))*exp(1i*2*pi*t/5)+ (-1i/(2*pi))*exp(-1i*2*pi*t/5)...
   + (1i/(3*pi))*exp(-1i*pi*3*t/5) + (-1i/(3*pi))*exp(1i*pi*3*t/5)...
   + (1i/(4*pi))*exp(1i*pi*4*t/5) + (-1i/(4*pi))*exp(-1i*pi*4*t/5)...
   + (1i/(5*pi))*exp(-1i*pi*5*t/5) + (-1i/(5*pi))*exp(1i*pi*5*t/5)...
   + (1i/(6*pi))*exp(1i*pi*6*t/5) + (-1i/(6*pi))*exp(-1i*pi*6*t/5)...
   + (1i/(7*pi))*exp(-1i*pi*7*t/5) + (-1i/(7*pi))*exp(1i*pi*7*t/5)...
   + (1i/(8*pi))*exp(1i*pi*8*t/5) + (-1i/(8*pi))*exp(-1i*pi*8*t/5)...
   + (1i/(9*pi))*exp(-1i*pi*9*t/5) + (-1i/(9*pi))*exp(1i*pi*9*t/5)...
   + (1i/(10*pi))*exp(1i*pi*10*t/5) + (-1i/(10*pi))*exp(-1i*pi*10*t/5)...
   + (1i/(11*pi))*exp(-1i*pi*11*t/5) + (-1i/(11*pi))*exp(1i*pi*11*t/5)...
   + (1i/(12*pi))*exp(1i*pi*12*t/5) + (-1i/(12*pi))*exp(-1i*pi*12*t/5)...
   + (1i/(13*pi))*exp(-1i*pi*13*t/5) + (-1i/(13*pi))*exp(1i*pi*13*t/5)...
   + (1i/(14*pi))*exp(1i*pi*14*t/5) + (-1i/(14*pi))*exp(-1i*pi*14*t/5)...
   + (1i/(15*pi))*exp(-1i*pi*15*t/5) + (-1i/(15*pi))*exp(1i*pi*15*t/5);



hold on
plot(t, y,'DisplayName','Original'), xlim([-10 10]), ylim([-1.5 1.5]);
plot(t, y1,'DisplayName','Sum of 3 Harmonics'), xlim([-10 10]), ylim([-1.5 1.5]);
plot(t, y2,'DisplayName','Sum of 5 Harmonics'), xlim([-10 10]), ylim([-1.5 1.5]);
plot(t, y3,'DisplayName','Sum of 15 Harmonics'), xlim([-10 10]), ylim([-1.5 1.5]);
set(legend,'fontsize',5);
xlabel('time')
ylabel('amplitude')
title('Signal and Harmonics Sum')
hold off
```

Explanation:

To write the harmonics sum formulas, I used the general x(t) formula that we have learned in class, which is x(t) = $\sum_{-\infty}^{\infty} a_k e^{j2\pi k f_0 t}$. Since T_0 is 10, f_0 is 1/10 = 0.1. For a_k values, I used the fourier coefficient values that I derived in the first part of the question.

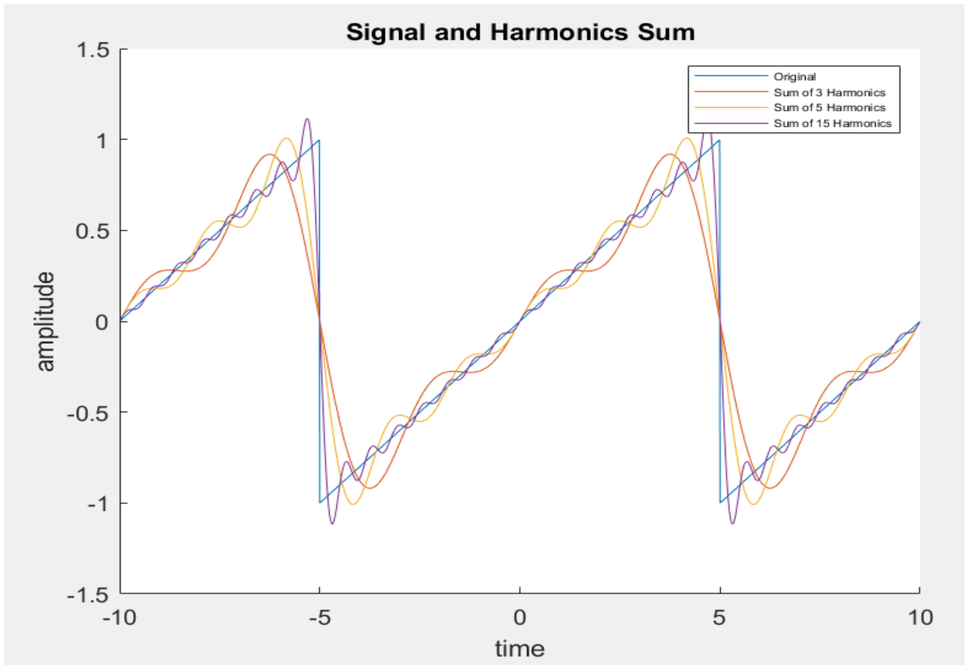
For 3 harmonics sum, -3<=k<=3

For 5 harmonics sum, -5<=k<=5

For 15 harmonics sum, -15<=k<=15

Figure:

(on the next page)



2)

Code & Comments:

```
[y,fs] = audioread('.\faultyphone.wav'); % sample data and sample rate

len = length(y); % number of samples
ff = fft(y,len); % computes DFT of x using fast fourier transform
x1 = fftshift(abs(ff)); % shift the magnitude of fft values
f0 = (-len/2:len/2-1)*(fs/len); % 0-centered frequency range

plot(f0,x1);
[maxvalues] = find(x1==max(x1));  % finding the index of la sound in frequency domain

filter = ones(len,1);
max1 = maxvalues(1,1);
max2 = maxvalues(2,1);

% making the necessary indexes of filter equal to 0 to ignore the la sound
% frequency
filter(max1,1) = 0;
filter(max2,1) = 0;

% filter value in time domain
fresult = ifft(ifftshift(filter)) ;

% convolution operation
convval = cconv(fresult,y, len);
```

```
len2 = length(convval); % number of samples
ff2 = fft(convval,len); % computes DFT of x using fast fourier transform
x2 = fftshift(abs(ff2)); % shift the magnitude of fft values
f0 = (-len2/2:len2/2-1)*(fs/len2); % 0-centered frequency range

% plotting frequency after filtering
%plot(f0,x2);

% listening filtered sound in matlab
soundsc(convval,fs);

% creating sound file
filename = 'filtered.wav';
audiowrite(filename,convval,fs);
```

Explanation:

Firstly, I moved to the frequency domain to see the frequency spectrum with the help of *fft* and *fftshift* functions. I saw the frequency of la sound(approximately 440 Hz) is the highest one in the frequency plot, and because the spectrum is 0-centered, there were two of them. So, I took the indexes of maximum values in the frequency spectrum with the help of *find* and *max* functions. After getting indexes, as explained in the question, I used 1 and 0 values for the filter, in which all the values are 1 except the indexes that correspond to la sound. These values should be 0 to ignore the la sound.

To check the correctness of my filter, I moved it to the time domain and used *cconv* function to make a convolution of filter and the initial signal, then listened and saved it to the file *filtered.wav.*

I used *cconv* function instead of *conv* (so, I used circular convolution instead of linear convolution), because the *conv* function did not give the correct sound. I searched the reason and I realized that the linear convolution works as half rotation, whereas the circular convolution works as full rotation. I think it is the reason of having a more accurate result with *cconv* function.

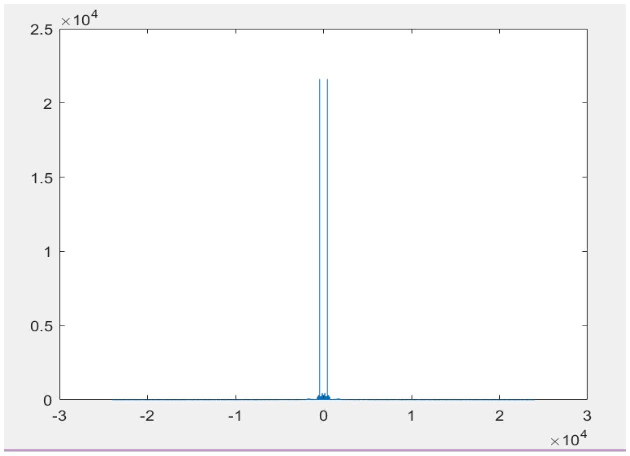
As can be seen from the before filtering and after filtering figures, the biggest amplitude values are gone after filtering.

Figures:

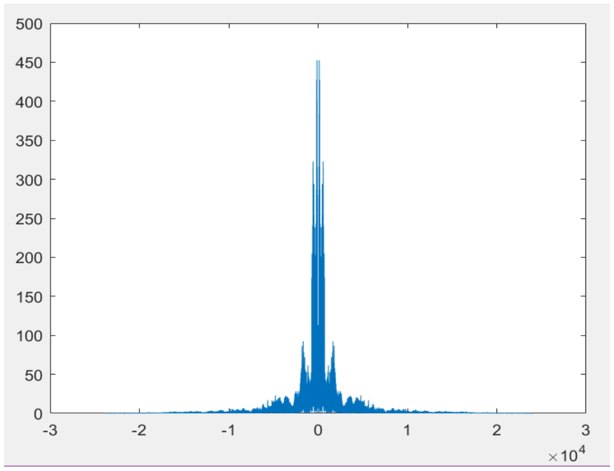
(Figures are on next page)

Aslı Aykan
2016400222

Before Filtering:



After Filtering:



3)

Code & Comments:

```
[y,fs] = audioread('.\enginesound.m4a'); % sample data and sample rate
size = length(y)/fs;  % time span signal
t = 0:1/fs:size-1/fs; % span time vector

avgsignal = y;

for index = 1:length(y)
   sum = 0;
   for aindex = 1:50  % summing 50 sample for each element in avg. filter
     if index+aindex<length(y)
       sum = sum + y(index+aindex);
     else
```

```
            sum = sum + 0;
        end
    end
    sum = sum /50;   % sum of 50 samples should be divided by 5
    avgsignal(index)=sum;
end


%spectrogram(avgsignal);


subplot(2,1,1), plot(t, y), title('initial');
subplot(2,1,2), plot(t, avgsignal), title('50-point average');
```

Explanation:

As we learned in class, the point average filters can be created with 3 different options: by using past values, present values, or future values. I decided to use future values and wrote my code accordingly.

Since the filter length is the same with the length of initial signal, I created a for loop of length y(initial signal, which is derived by *audioread* function) to calculate each element in the filter. Since it is a future value point average, any element in filter is the sum of next n elements in the initial signal divided by n.

filter(a) = (ins(a) + ins(a+1) + …… + ins(a+50))/50   - ins is the initial signal function –

So, I created an inner loop which sums the next 50 elements in the initial signal, then divided the sum by 50. This loop runs for each element in the filter. For the last elements of the filter (if we do not have 50 next elements in the original signal), I checked the length and add 0 if there is no element left for the future elements.

I plotted the initial signal and its 50-point average form as subplots to compare them easily. As can be seen from the figure below, the averaging filter reduces the noise in the signal. I also tried it with more than 50 points and realized that the noise reduces as the number of points in the averaging filter increases. I searched its reason and learned that since the noise is random, it is not easy to find the indexes of the noise and reduce them by giving them special coefficients. By filtering the signal with averaging filter, without giving special attention to some indexes, it is very easy and fast to reduce the noise in the signal. We can also say that the signal becomes smoother, and in data smoothing, it is natural to have decrease in amplitude. As can be seen from the figure, the amplitude of filtered signal is lower.

As can be seen from the spectrogram of the filtered signal, there exists three yellow lines on the left (with high amplitude and low frequency), they represent the fundamental frequency. To find its value, we need to zoom into the spectrogram.

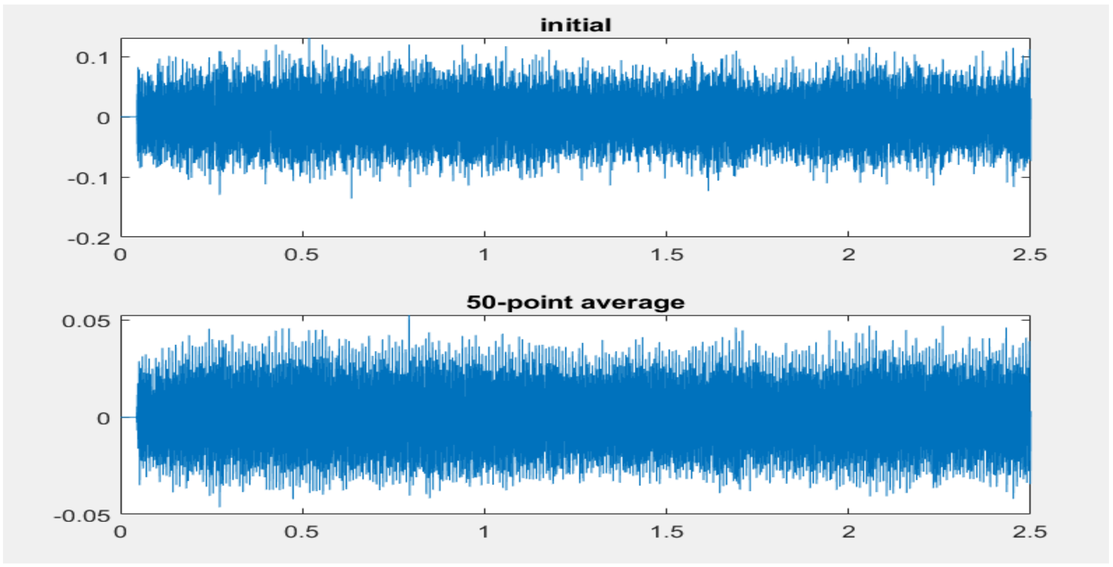
As can be seen from the zoomed versions of the spectrogram, 0.009 value is the value that has the highest amplitude from start, but the value is in the form of normalized frequency. The formula for frequency is:

Aslı Aykan
2016400222

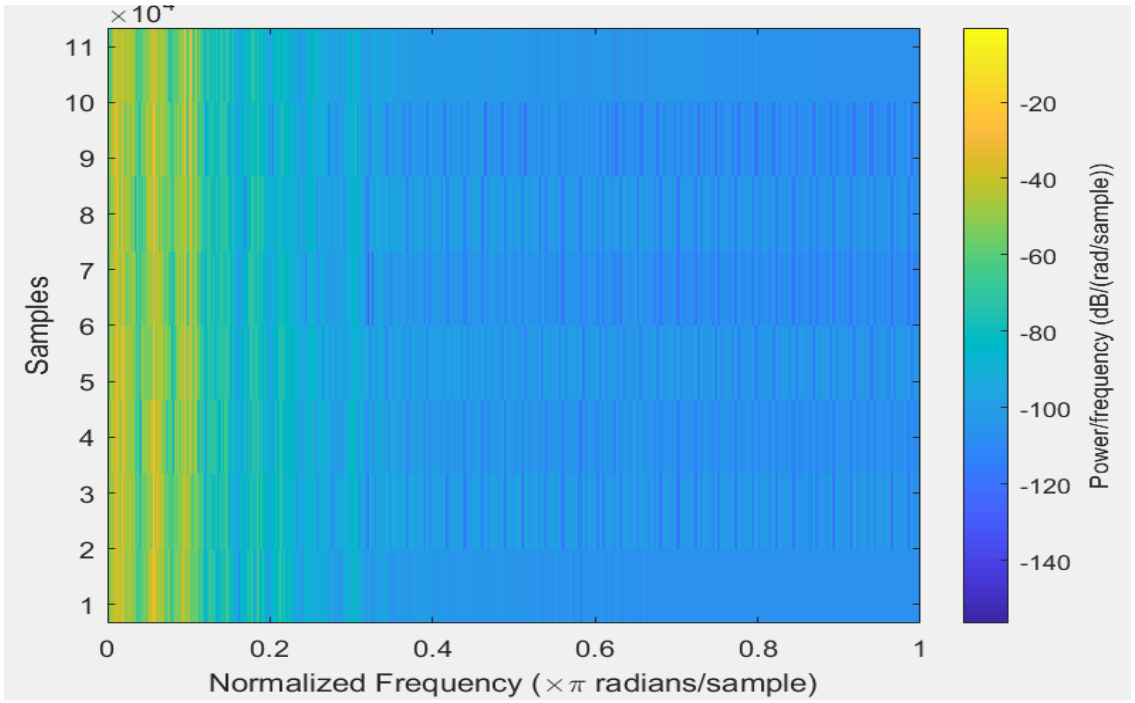
f = fs x fn / 2 = 48000 x 0.009 / 2 = 216 Hz is the fundamental frequency. (fs=48000 value is found with the help of *audioread* function)

So, the N_{RPM} = (1/2)(1)(60)(216) = 6480

Figure:

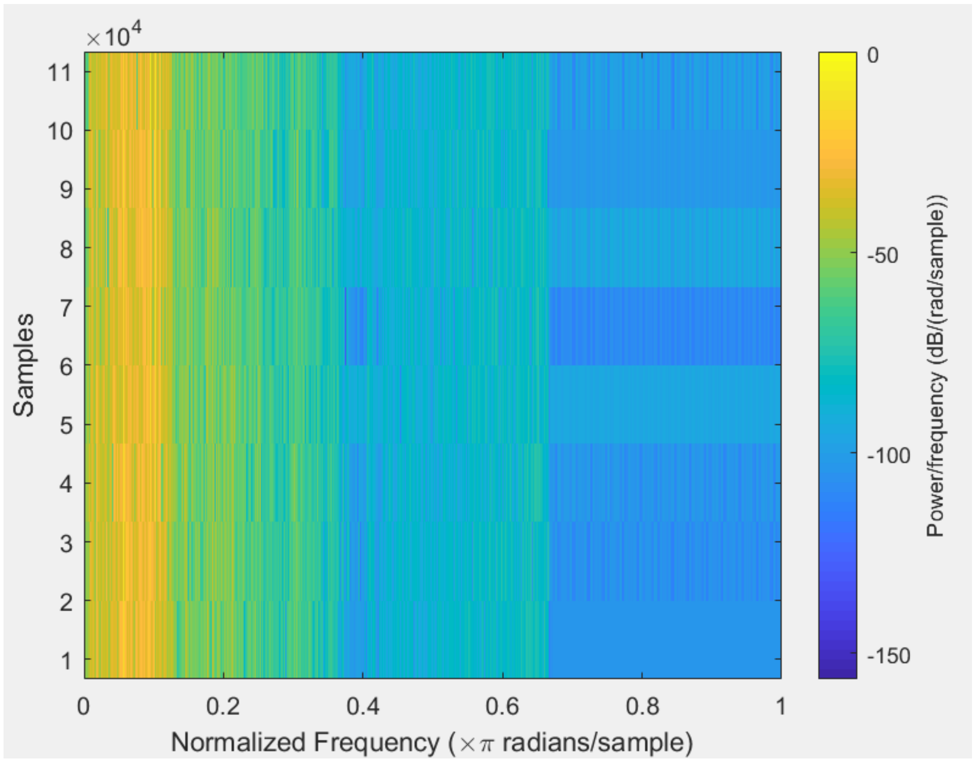


Spectrogram of the filtered signal:



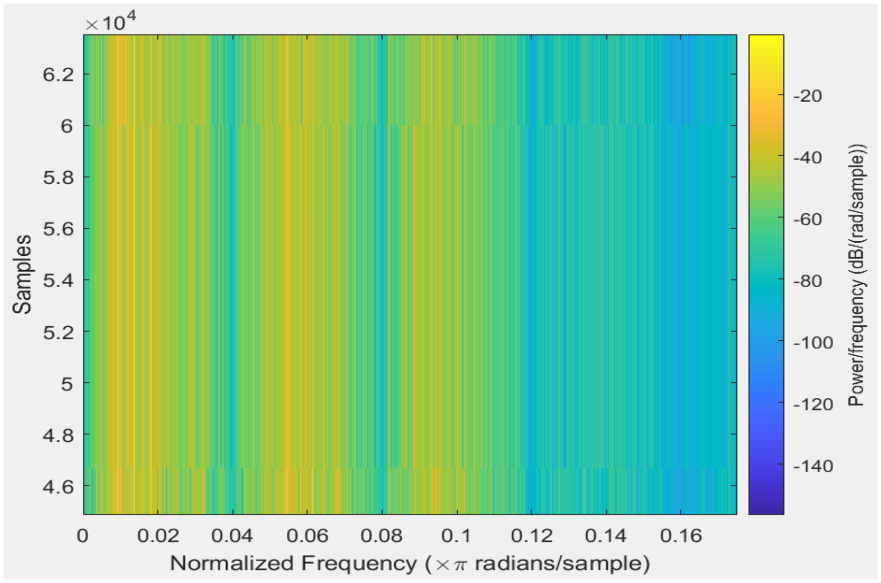
Aslı Aykan
2016400222

Spectrogram of the initial signal:



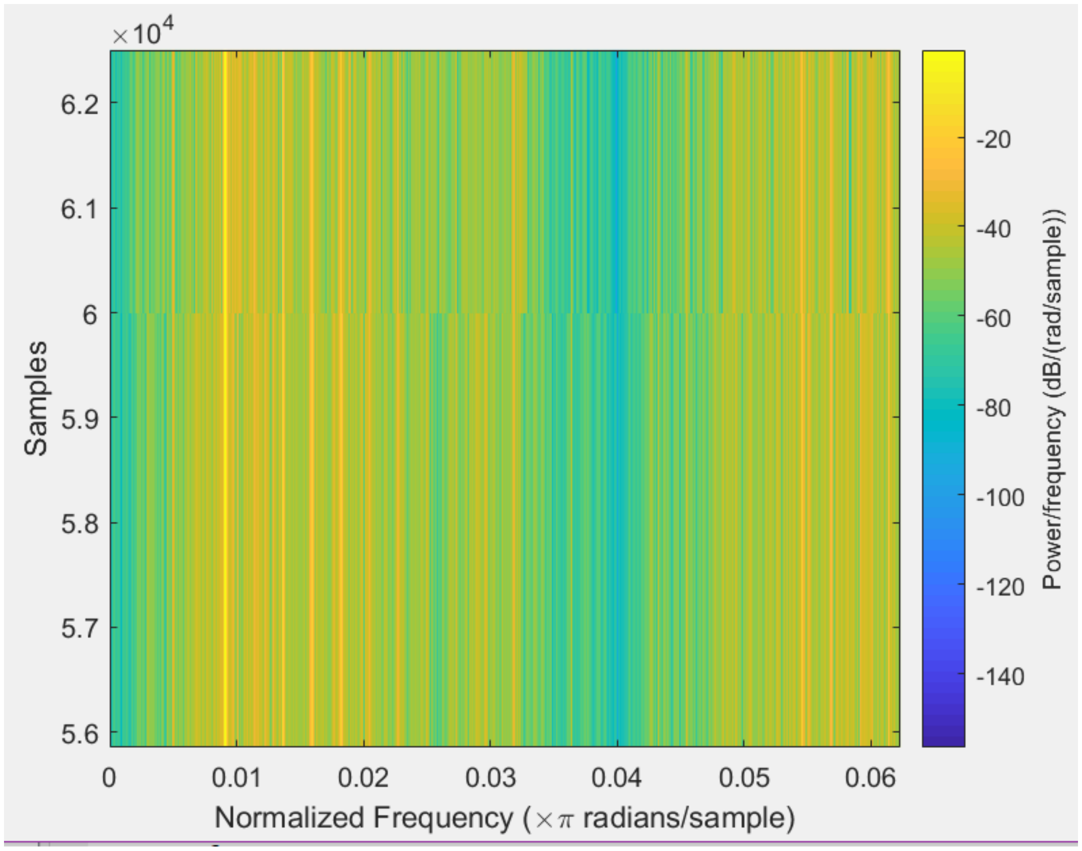
Zoomed Versions of Spectrogram (for filtered signal):

First step:

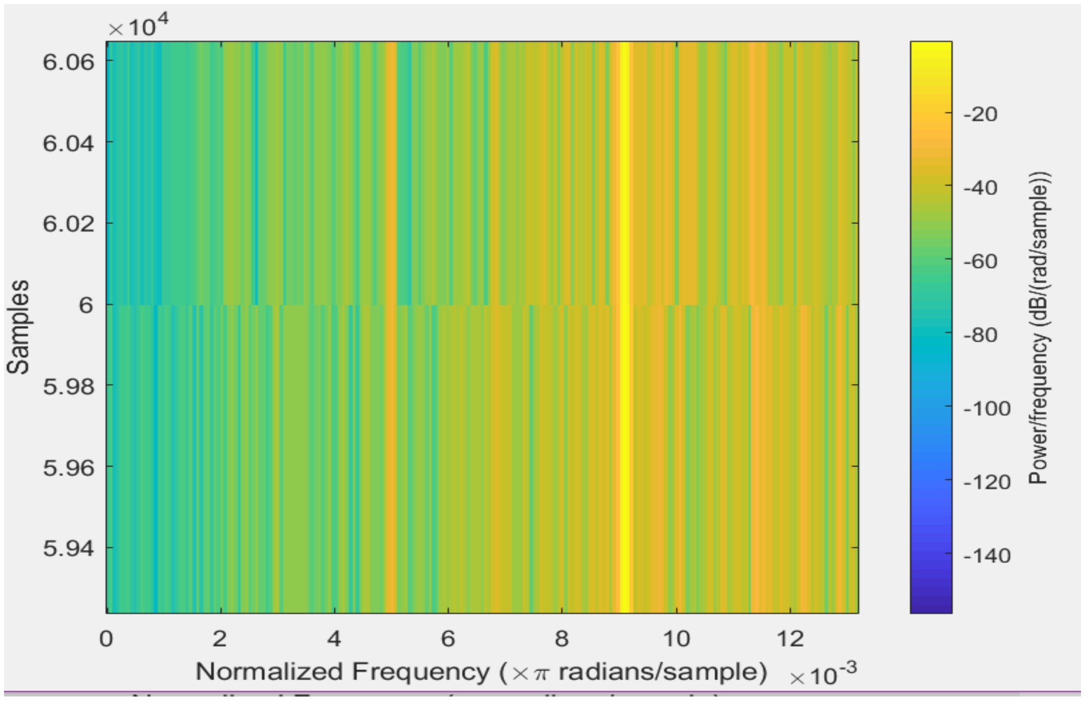


Aslı Aykan
2016400222

Second Step:



Third Step:


$\times 10^4$
Samples
Normalized Frequency ($\times\pi$ radians/sample)
$\times 10^{-3}$
Power/frequency (dB/(rad/sample))