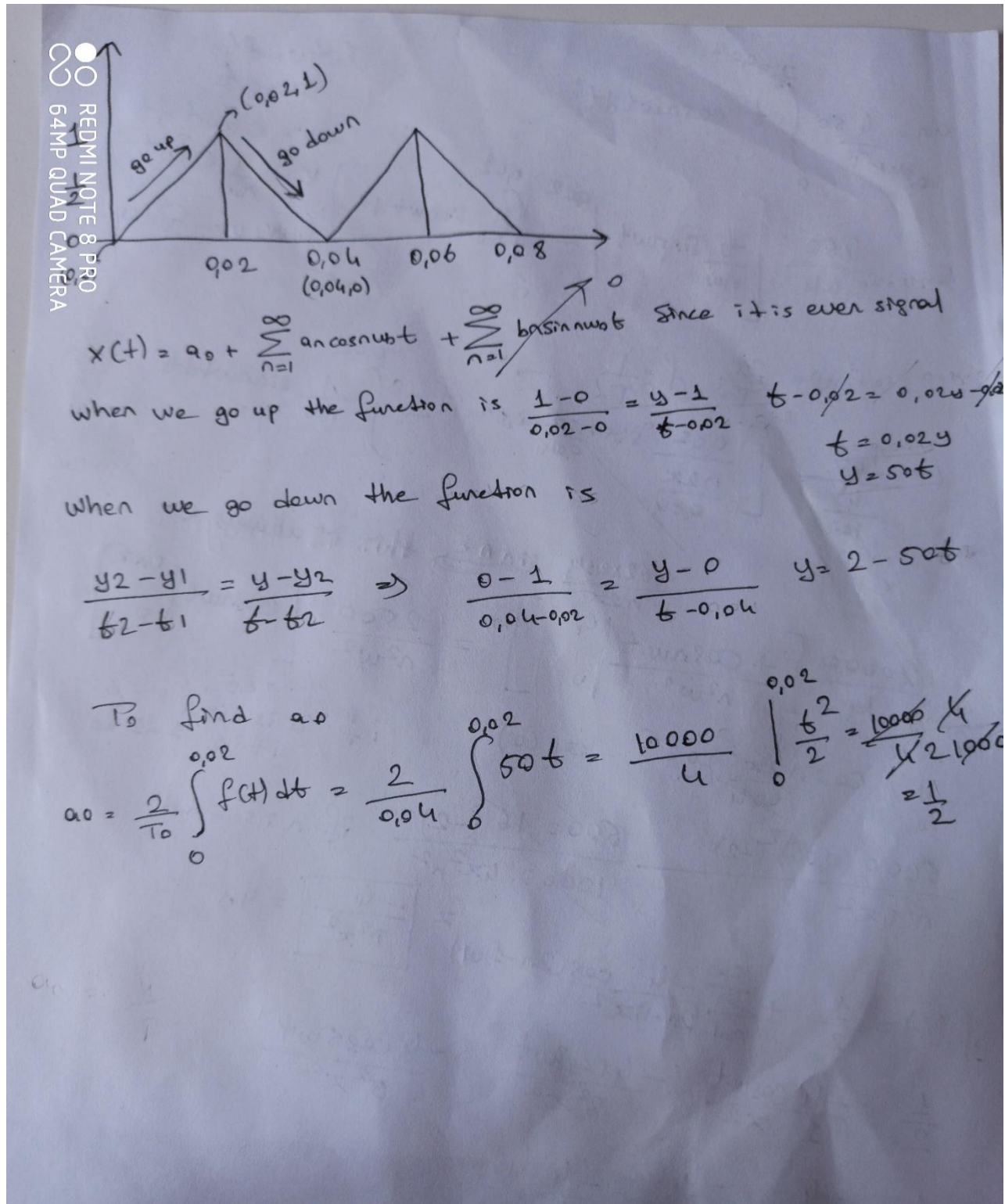


Question 1)

- Part A



$$a_n = \frac{4.50}{0.04} \int_0^{0.02} t \cos n\omega t dt$$

$$\frac{4.50}{0.04} \left[\frac{t}{n\omega} \sin n\omega t \right]_0^{0.02} - \int_0^{0.02} \frac{1}{n\omega} \sin n\omega t dt \quad \left[\begin{array}{l} \text{let } u = t \\ du = dt \\ dv = \cos n\omega t dt \\ v = \frac{1}{n\omega} \sin n\omega t \\ \int u dv = uv - \int v du \end{array} \right]$$

$$\omega = \frac{2\pi}{T} \quad f = \frac{1}{T}$$

$$\frac{200}{\frac{4}{100}} \left[\frac{0.02}{\frac{n2\pi}{0.04}} \sin \frac{n2\pi}{0.04} \cdot 0.02 - \int_0^{0.02} \frac{1}{n\omega} \sin n\omega t dt \right]$$

$$\frac{20000}{4} \left[\frac{0.02 \times 0.04 \sin n\pi}{n2\pi} \right]_{0.02}^{0.02} - \int_0^{0.02} \frac{1}{n\omega} \sin n\omega t dt$$

this is always 0

$$= \frac{2500}{n^2 \omega^2} \left[\cos n\omega t \right]_0^{0.02}$$

$$\cos \frac{n2\pi}{0.04} \cdot 0.02 - \cos(0) \quad \text{if } n \text{ is even it is } 1-1=0$$

$$\frac{5000 (0.04)^2 (-2)}{n^2 4\pi^2} = \frac{5000 \times 16 (-2)}{10000 4\pi^2 n^2} \quad \text{if } n \text{ is odd}$$

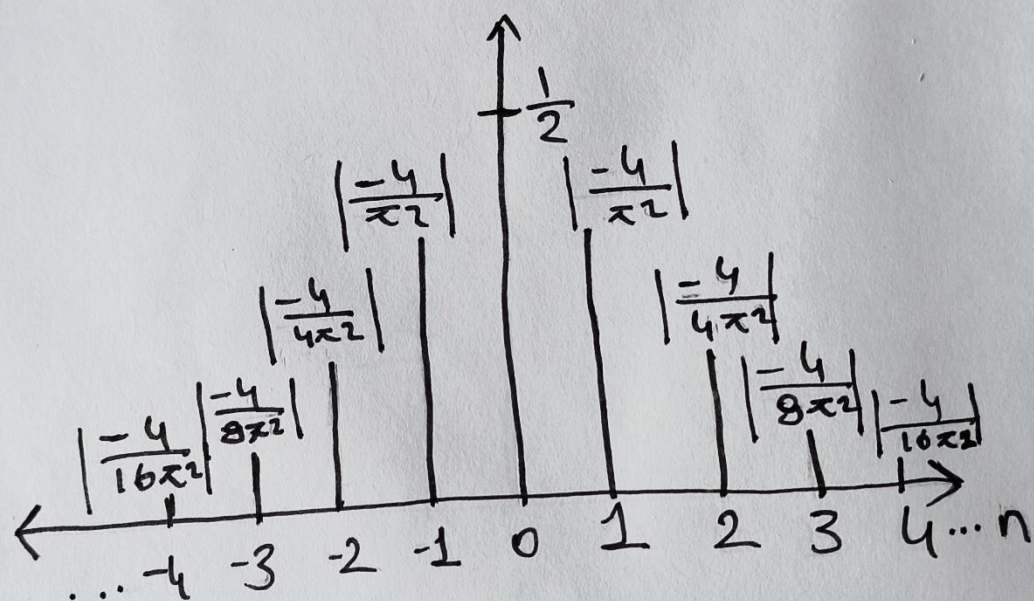
$$f(t) = \frac{1}{2} + \sum_{n=1}^{\infty} \frac{-4 \cos(2n-1)\omega t}{(2n-1)^2 \pi^2} = \boxed{\frac{-4}{n^2 \pi^2}} = a_n$$

$$\frac{1}{2} \quad \frac{-4 \cos \omega t}{1 \pi^2} \quad \frac{-4 \cos 3\omega t}{3^2 \pi^2} \quad \frac{-4 \cos 5\omega t}{5^2 \pi^2}$$

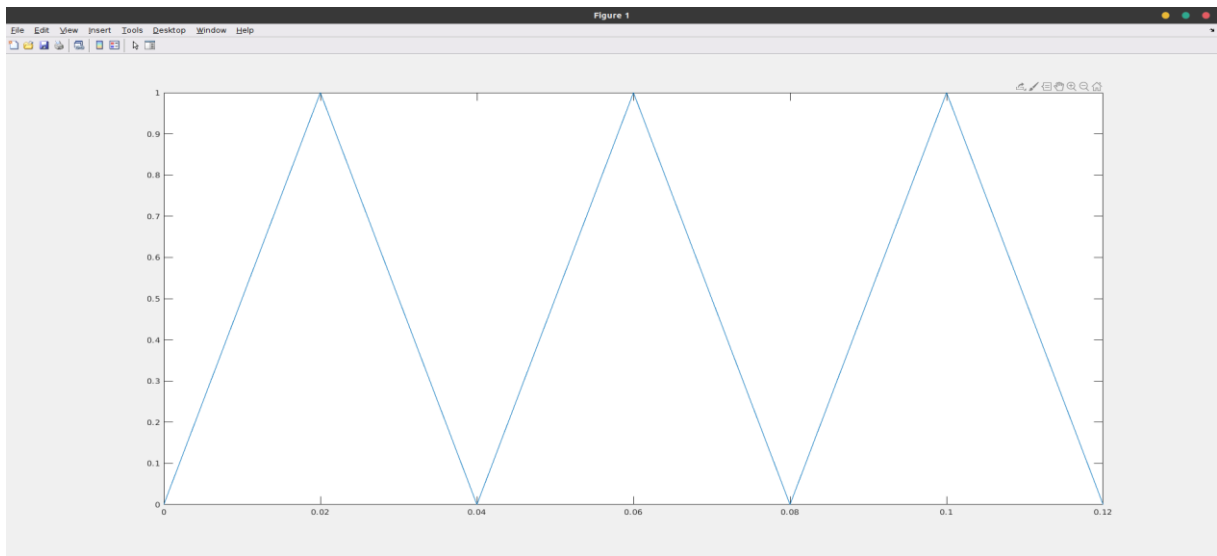
$$\frac{1}{2} \quad \frac{-4}{\pi^2} \quad \frac{-4}{9\pi^2} \quad \frac{-4}{25\pi^2}$$

As we can see from the photos of my solution, the coefficients of the Fourier series are

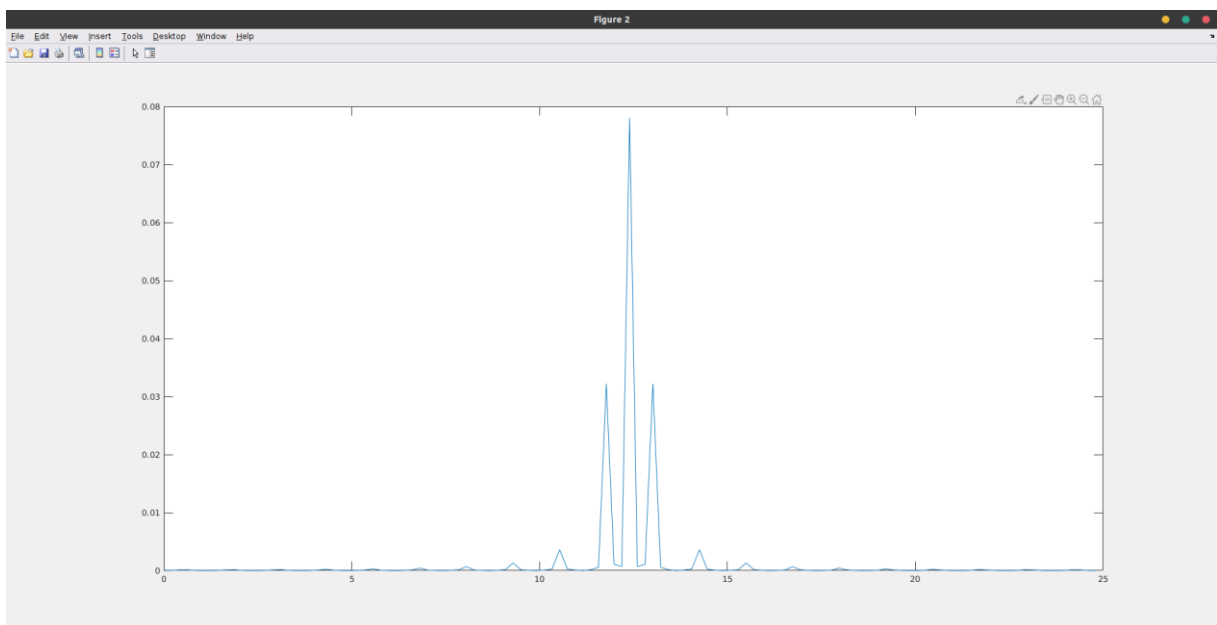
$$a_k = \begin{cases} \frac{1}{2} & \text{if } k=0 \\ 0 & \text{if } k \text{ is even but } \neq 0 \\ -\frac{4}{\pi^2 k^2} & \text{if } k \text{ is odd} \end{cases}$$



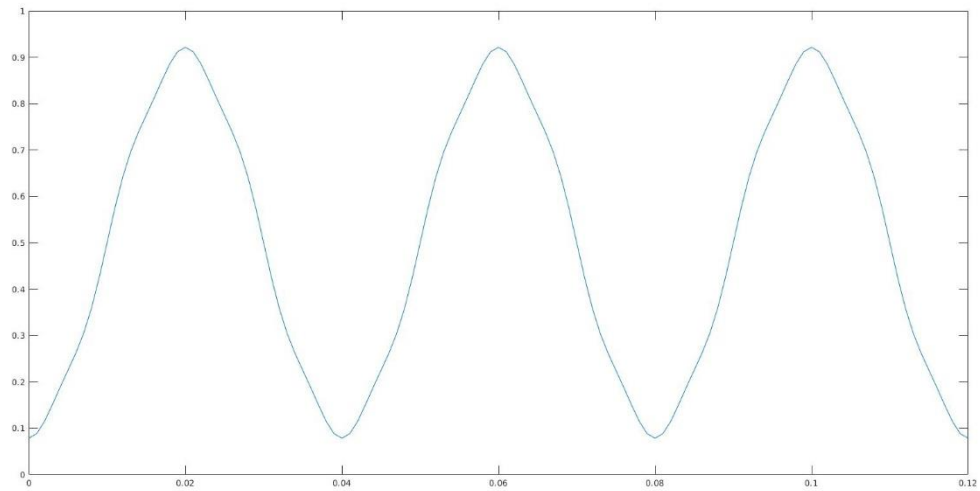
- **Part B**



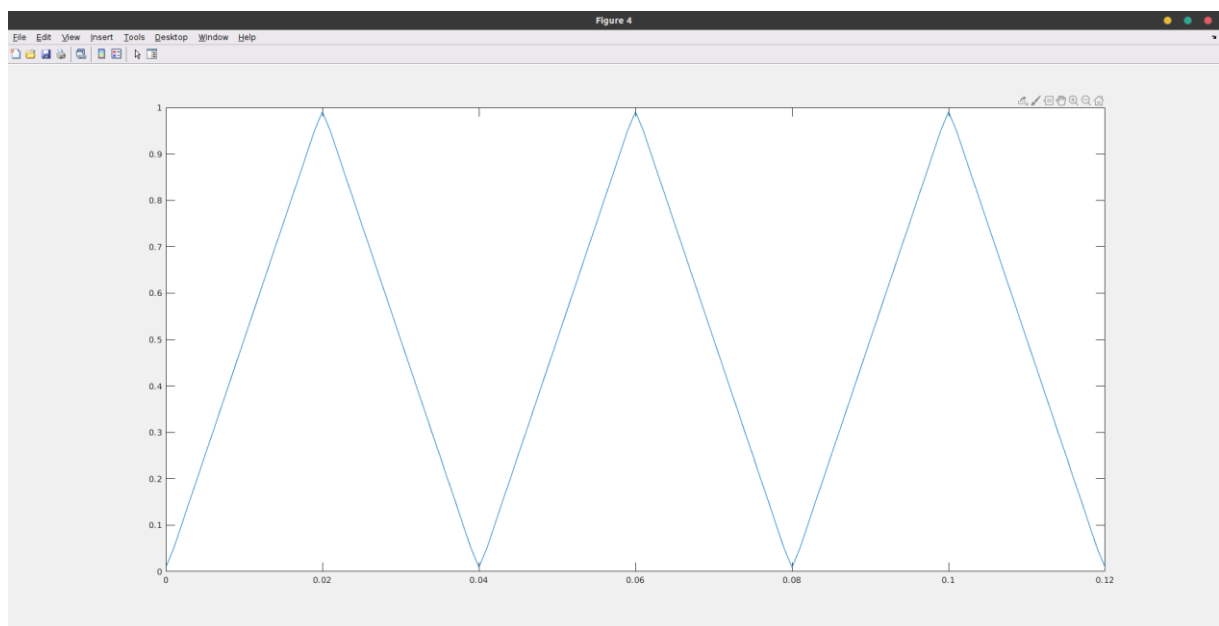
This figure above is constructed by built-in sawtooth function. The formula is clear. The period is 0.04 and we can find the frequency by $1/T$ and put the main formula in sawtooth as $(\text{sawtooth}(f_s * 2 * \pi * t, 0.5) + 1) / 2$ to get positive values and plot the results to take this figure.



This figure above was constructed by operating FFT, normalization, and FFTShift on the signal above respectively. As we can from the figure the coefficients are absolutely matching with the ones, we found in part a.



This figure above was found by summing the 1st and 3rd harmonic series and we can see that the figure resembles the original one little.



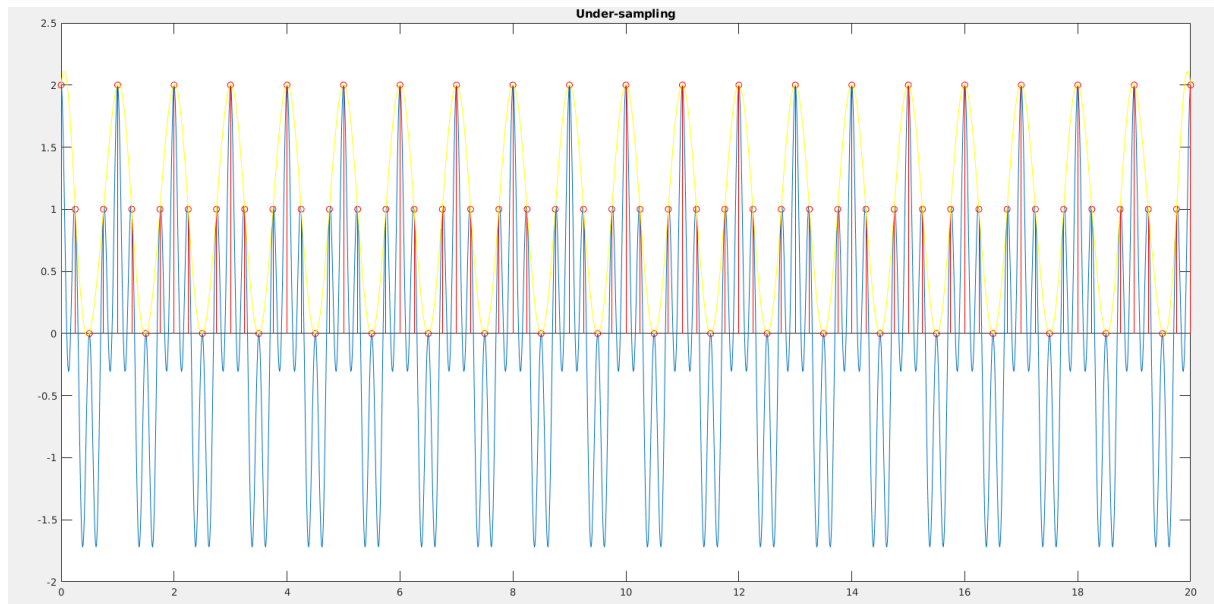
This figure above was found by summing the 1st through 11th harmonic series and we can see that the figure resembles the original one a lot.

The code I implemented is below

```
1  %Here we are defining our figures
2  f1 = figure;
3  f2 = figure;
4  f3 = figure;
5  f4 = figure;
6  %call the first figure
7  figure(f1);
8  %we define the time interval according to the one in the pdf
9  t = 0:0.001:.12;
10 T0 = 0.04; %Period
11 fs = 1 / T0; %this is frequency
12 y1 = (sawtooth(fs * 2 * pi * t, 0.5) + 1) / 2; % this return us the function we want.
13 plot(t, y1); % and plot it
14 %call second figure
15 figure(f2);
16 y = normalize(fft(y1)); %discrete fourier transform and normalize as you mentioned
17 shift_y = fftshift(y);
18 n = length(y1); % to take number of samples
19 f0 = (-n / 2:n / 2 - 1) * (fs / n); % 0-centered frequency range
20 plot(f0, abs(shift_y).^2 / n); % here we get the magnitude.
21 %call third figure
22 figure(f3);
23 %the same process
24 t = 0:0.001:.12;
25 T0 = 0.04;
26 fs = 1 / T0;
27 %Here we find the first and third harmonic series.
28 x1 = (-2 / (pi * pi)) * cos(2 * pi * fs * t);
29 x3 = (-2 / (25 * pi * pi)) * cos(5 * 2 * pi * fs * t);
30 %By adding them up we get the new signal which resamples the original one a little.
31 z = x1 + x3 + 1/2; % we add 1/2 since a0 is 1/2.
32 plot(t, z);
33 %calling the last figure
34 figure(f4);
35 %Here we found the all harmonic series through x1 to x11
36 x1 = (-2 / (pi * pi)) * cos(2 * pi * fs * t);
37 x2 = (-2 / (9 * pi * pi)) * cos(3 * 2 * pi * fs * t);
38 x3 = (-2 / (25 * pi * pi)) * cos(5 * 2 * pi * fs * t);
39 x4 = (-2 / (49 * pi * pi)) * cos(7 * 2 * pi * fs * t);
40 x5 = (-2 / (81 * pi * pi)) * cos(9 * 2 * pi * fs * t);
41 x6 = (-2 / (121 * pi * pi)) * cos(11 * 2 * pi * fs * t);
42 x7 = (-2 / (169 * pi * pi)) * cos(13 * 2 * pi * fs * t);
43 x8 = (-2 / (225 * pi * pi)) * cos(15 * 2 * pi * fs * t);
44 x9 = (-2 / (17 * 17 * pi * pi)) * cos(17 * 2 * pi * fs * t);
45 x10 = (-2 / (19 * 19 * pi * pi)) * cos(19 * 2 * pi * fs * t);
46 x11 = (-2 / (21 * 21 * pi * pi)) * cos(21 * 2 * pi * fs * t);
47 % and summing them up to get the new signal that resamples thhe original one a lot.
48 z = x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10 + x11 + 1/2;
49 plot(t, z);
```

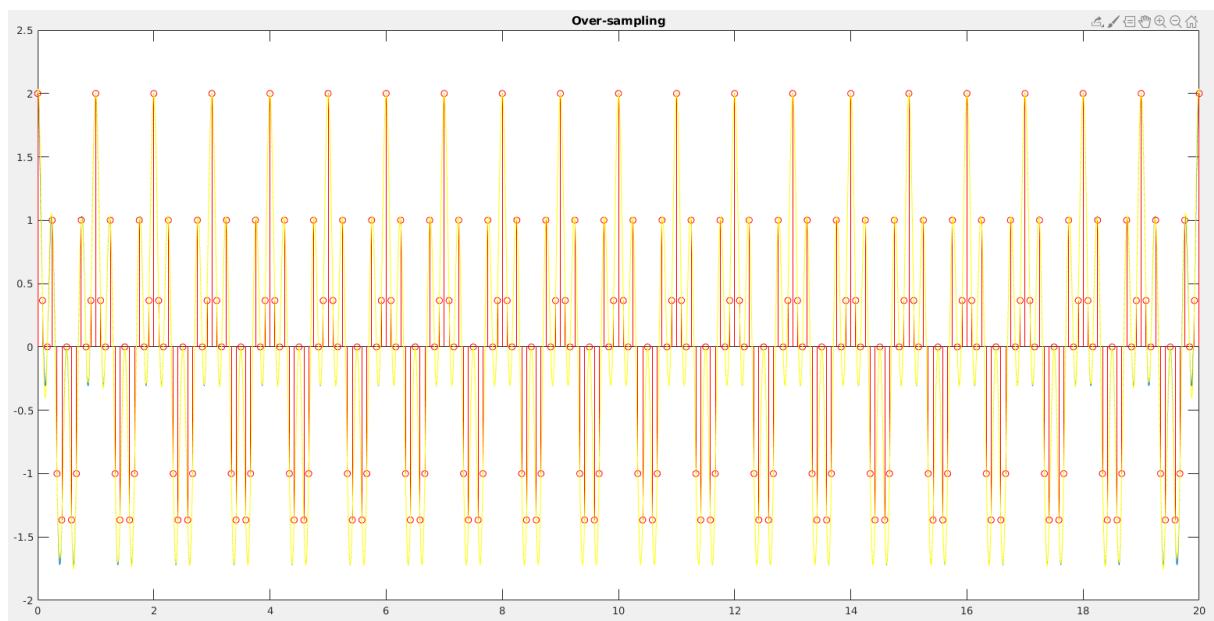
Question 2)

I choose the formula of my signal as $\cos(2 \cdot \pi \cdot 1) + \cos(2 \cdot \pi \cdot 4)$ and take the samples by using some different frequencies and plot the samples by using stem built-in function.



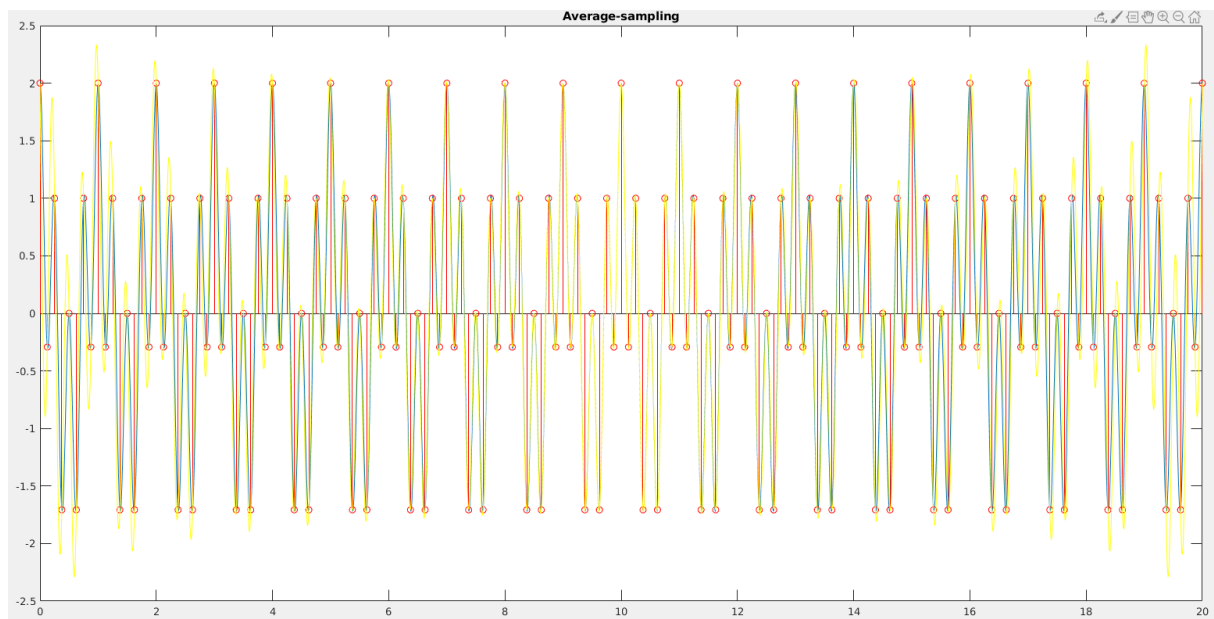
Here I chose the sampling rate as 4 and as you can see the reconstructed signal which is colored yellow does not resemble the original signal.

Then I chose a bigger sampling rate, which is 12.



As you can see the reconstructed signal which colored yellow is perfectly resemble the original signal a lot. But the number of samples is very high.

However, I tried to find the Nyquist sampling rate by taking the average of two number we use $(4+12)/2=8$.



As you can see the new signal reconstructed with sampling rate 8 is resemble the original signal perfect and as we know from the formula of Nyquist sampling rate it should be 8 since the formula is $2 \cdot \max(f_1, f_2)$, f_1 is 1 and f_2 is 4 then the Nyquist sampling rate need to be 8. So, the numbers are matching then our hypothesis is true. 8 is Nyquist sampling rate.

Note: The circles of samples are colored in red since the orange color is not visible a lot.

The code I implemented is below

```
%we define the figures
f1 = figure;
f2 = figure;
f3 = figure;
%calling The first one. This figure shows the undersampling
figure(f1);
F1 = 1; %the first frequency, we will find the second frequency by multipling
it by 4.
Fs = 4; % this is the sampling rate's frequencys. Since we expect the Nyquist
is 8 then we need to select lower value.
tmin = 0; % In pdf the time interval was 0:20 so I define it here like it.
tmax = 20;
t = tmin:0.01:tmax; % and to have continious signal increase 0.01
x1 = cos(2 * pi * F1 * t) + cos(2 * pi * 4 * F1 * t); % we get the original si
gnal here. f1=1 f2=4 (4.f1)
Ts = 1 / Fs; %To get the period.
ts = tmin:Ts:tmax; %Here we get the sampling rate's time interval by increasin
g 1/Fs
```



```

% or we can increase again 0.01 and as constructin the signal we can divide th
e inside of cos by Fs.
x1resampled = cos(2 * pi * F1 * ts) + cos(2 * pi * 4 * F1 * ts); % and we crea
te the sample's signal(The signal look like sticks)
x1reconstructed = zeros(1, length(t)); %Here we create the signal we will reco
nstruct. Initialize it by all zeros with lenght of original one.
samples = length(ts); %then we take the samples of the sampling signal
% here we try to reconstruct the original signal
% first we traverse the first loop with length original one
for index = 1:1:length(t)
    % and we traverse the second loop with length the sample signal.
    for n = 1:1:samples
        % each iteration we add the multiplication of the value of sample sign
al and sinc function. we substruct 1 to eliminate the zero at the beginning.
        x1reconstructed(index) = x1reconstructed(index) + x1resampled(n) * sin
c(((index - 1) * 0.01 - (n - 1) * (1 / Fs)) / (1 / Fs));
    end
end
%after reconstruction we plot the original one then hold on and plot the sampl
e signal then plot the recostrcted signal.
plot(t, x1)
hold on
stem(ts, x1resampled, 'color', 'r')% The clour of samples are red since orange
is not visible a lot.
plot(t, x1reconstructed, '-y')
title("Under-sampling")
%All the logic is the same as above.
figure(f2);
F1 = 1; %the first frequency, we will find the second frequency by multipling
it by 4.
Fs = 12; % this is the sampling rate's frequencys. Since we expect the Nyquist
is 8 then we need to select higher value.
tmin = 0;
tmax = 20;
t = tmin:0.01:tmax;
x1 = cos(2 * pi * F1 * t) + cos(2 * pi * 4 * F1 * t);
Ts = 1 / Fs;
ts = tmin:Ts:tmax;
x1resampled = cos(2 * pi * F1 * ts) + cos(2 * pi * 4 * F1 * ts);
x1reconstructed = zeros(1, length(t));
samples = length(ts);
for index = 1:1:length(t)
    for n = 1:1:samples
        x1reconstructed(index) = x1reconstructed(index) + x1resampled(n) * sin
c(((index - 1) * 0.01 - (n - 1) * (1 / Fs)) / (1 / Fs));
    end
end
plot(t, x1)
hold on

```

```

stem(ts, x1resampled, 'color', [0.9100 0.4100 0.1700])
plot(t, x1reconstructed, '-y')
title("Over-sampling")
%All the logic is the same as above.
figure(f3);
F1 = 1;
Fs = 8;
tmin = 0;
tmax = 20;
t = tmin:0.01:tmax;
x1 = cos(2 * pi * F1 * t) + cos(2 * pi * 4 * F1 * t);
Ts = 1 / Fs;
ts = tmin:Ts:tmax;
x1resampled = cos(2 * pi * F1 * ts) + cos(2 * pi * 4 * F1 * ts);
x1reconstructed = zeros(1, length(t));
samples = length(ts);
for index = 1:1:length(t)
    for n = 1:1:samples
        x1reconstructed(index) = x1reconstructed(index) + x1resampled(n) * sin
c(((index - 1) * 0.01 - (n - 1) * (1 / Fs)) / (1 / Fs));
    end
end
plot(t, x1)
hold on
stem(ts, x1resampled, 'color', [0.9100 0.4100 0.1700])
plot(t, x1reconstructed, '-y')
title("Average-sampling")

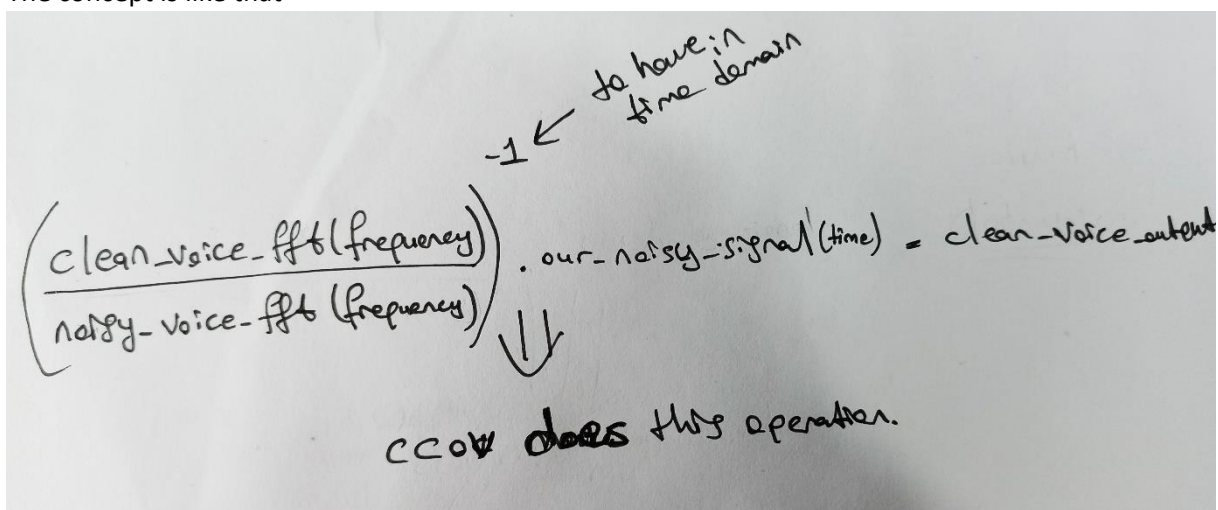
```

Question 3)

The samples I used are c_p232_090.wav and n_p232_090.wav

As explained in pdf we need to apply the operation of discrete Fourier transform to both noisy signal and clean signal and divide the clean ones by noisy one to get the the frequency response of my filter. Actually we don't need to have a filter here since we already can have clean audio's aspects and noisy audio's in both time domain and frequency domain so just take their fft divide them by each other and have the frequency domain then apply ifft to have the aspects in time domain so that we can have the output signal in time domain. The last step is easy one just put them in ccov function as a parameter.

The concept is like that

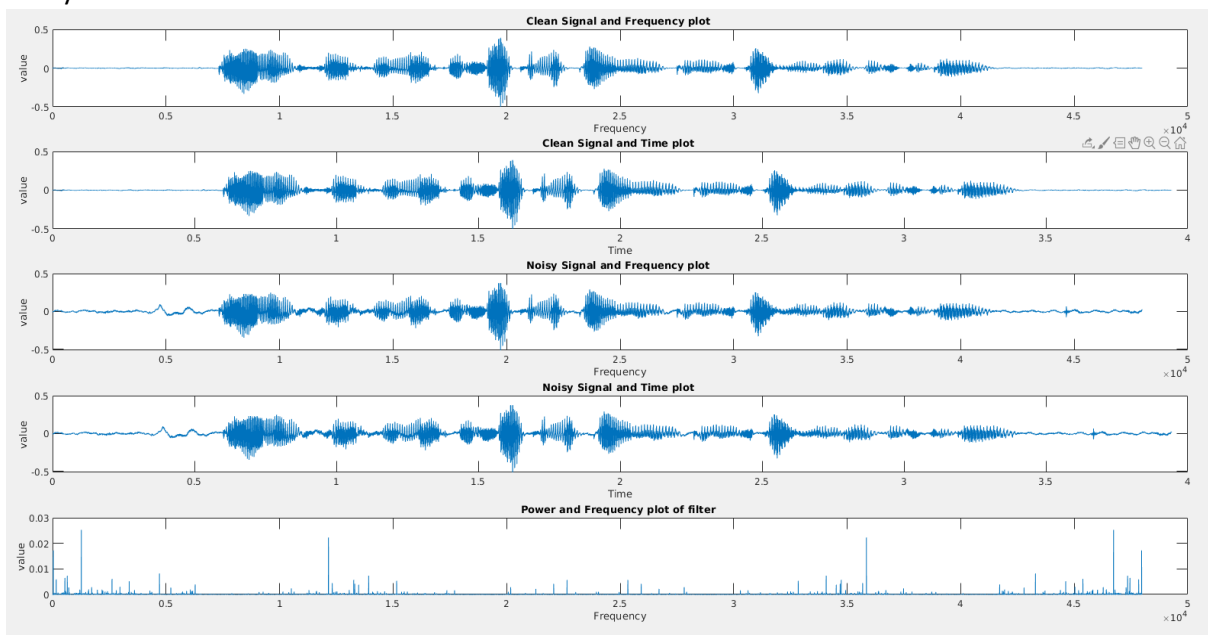


The image shows a handwritten mathematical formula on a piece of paper. The formula is:

$$\left(\frac{\text{clean_voice_fft}(\text{frequency})}{\text{noisy_voice_fft}(\text{frequency})} \right) \cdot \text{our_noisy_signal}(\text{time}) = \text{clean_voice_output}$$

There is a handwritten note above the formula: "-1 ← to have in time domain". Below the formula, there are two downward-pointing arrows. At the bottom, it says "ccov does this operation."

Below I plot the time and frequency spectrum of my clean and noisy signal and frequency spectrum of my filter.



And the first part of the code is below

```
1 %In tthis code we complete the 1/3 part of question3.
2 [our_clean_sample_signal, fs_clean] = audioread('c_p232_090.wav'); % to read the our clean audio
3 our_clean_signal = our_clean_sample_signal(:, 1); % to take the clean audio.
4 taken_clean_fft = fft(our_clean_signal); %
5 Duration_of_our_clean_signal = length(our_clean_signal)
6 t = linspace(0, Duration_of_our_clean_signal / fs_clean, Duration_of_our_clean_signal)
7 f = (0:Duration_of_our_clean_signal - 1) * (fs_clean / Duration_of_our_clean_signal);
8 subplot(5, 1, 1)
9 plot(f, our_clean_signal)
10 title("Clean Signal and Frequency plot ")
11 xlabel('Frequency')
12 ylabel('value')
13 subplot(5, 1, 2)
14 plot(t, our_clean_signal)
15 title("Clean Signal and Time plot ")
16 xlabel('Time')
17 ylabel('value')
18 [our_noisy_sample_signal, fs_noisy] = audioread('n_p232_090.wav'); % to read the nosiy music
19 our_noisy_signal = our_noisy_sample_signal(:, 1); % to take the clean audio.
20 Duration_of_our_noisy_signal = length(our_noisy_signal)% it is important to specify the duration of output signal
21 taken_noisy_fft = fft(our_noisy_signal);
22 t2 = linspace(0, Duration_of_our_noisy_signal / fs_noisy, Duration_of_our_noisy_signal)
23 f2 = (0:Duration_of_our_noisy_signal - 1) * (fs_noisy / Duration_of_our_noisy_signal);
24 figure(f1);
25 subplot(5, 1, 3)
26 plot(f2, our_noisy_signal)
27 title("Noisy Signal and Frequency plot ")
28 xlabel('Frequency')
29 ylabel('value')
30 subplot(5, 1, 4)
31 plot(t2, our_noisy_signal)
32 title("Noisy Signal and Time plot ")
33 xlabel('Time')
34 ylabel('value')
35 our_overall_filter_response = taken_clean_fft ./ taken_noisy_fft% to have frequency response of my ideal lter
36 y0 = fftshift(our_overall_filter_response);
37 power0 = abs(y0).^2 / Duration_of_our_noisy_signal;
38 subplot(5, 1, 5)
39 plot(f, power0)
40 title("Power and Frequency plot of filter ")
41 xlabel('Frequency')
42 ylabel('value')
43 In_Time_Domain_Response = ifft(our_overall_filter_response)% take the inverve fft to chance the domain
44 output = cconv(our_noisy_signal, In_Time_Domain_Response, Duration_of_our_noisy_signal)% and tae convolution to have clean audio
45 audiowrite("newCleanaudio.wav", output, fs_clean)
```

If we want to generalize the filter design, I used averaging filter. The concept is simple we determine how many samples we need to take. According to my general observation I take 225 (Actually 180, 200, 250 are all behaving the same but I choose 225 since I like this number) and create an array with length 225 like full of 1,1,1, 1,.. since all the coefficients of *are the same*. And Divide it with the length of the array which is 225. Then this is our filter time response. I used cconv just like first part and get the clean voice.

This link below shows how I get the idea.

<https://www.youtube.com/watch?v=yargH0L3B68&list=LLrVr8jxbHsx0o7mbyq4kyRA&index=6&t=0s>

The code below is second part of the question3.

```
%In this script we try to design average filter for general cases.
%=== Taking the name of the file that we make clean===
TheNameOFFile = input("Enter The Name Of the sample file : ", 's')
[sample_signalx, Fs] = audioread(TheNameOFFile); % to read the sample noisy audio
sample_signal = sample_signalx(:, 1); % to take the clean audio.
Duration_of_our_signal = length(sample_signal)
arraysize = 225;
sample_TR = ones(1, arraysize) / arraysize;
Cleaned_Signal = cconv(sample_signal, sample_TR, Duration_of_our_signal);
sound(Cleaned_Signal, Fs)% to see the clean audio
taken_clean_ftt = fft(Cleaned_Signal); %
t = linspace(0, Duration_of_our_signal / Fs, Duration_of_our_signal)
f = (0:Duration_of_our_signal - 1) * (Fs / Duration_of_our_signal);
subplot(5, 1, 1)
plot(f, Cleaned_Signal)
title("Clean Signal and Frequency plot ")
xlabel('Frequency')
ylabel('value')
subplot(5, 1, 2)
plot(t, Cleaned_Signal)
title("Clean Signal and Time plot ")
xlabel('Time')
ylabel('value')
taken_noisy_ftt = fft(sample_signal);
subplot(5, 1, 3)
plot(f, sample_signal)
title("Noisy Signal and Frequency plot ")
xlabel('Frequency')
ylabel('value')
subplot(5, 1, 4)
plot(t, sample_signal)
title("Noisy Signal and Time plot ")
xlabel('Time')
ylabel('value')
our_overall_filter_response = taken_clean_ftt ./ taken_noisy_ftt% to have frequency response of my ideal filter
y0 = fftshift(our_overall_filter_response);
power0 = abs(y0).^2 / Duration_of_our_signal;
subplot(5, 1, 5)
plot(f, power0)
title("Power and Frequency plot of filter ")
xlabel('Frequency')
ylabel('value')
```