Bilge Kaan Ateş

22102758

01.03.2024

**EEE321 SIGNALS AND SYSTEMS**

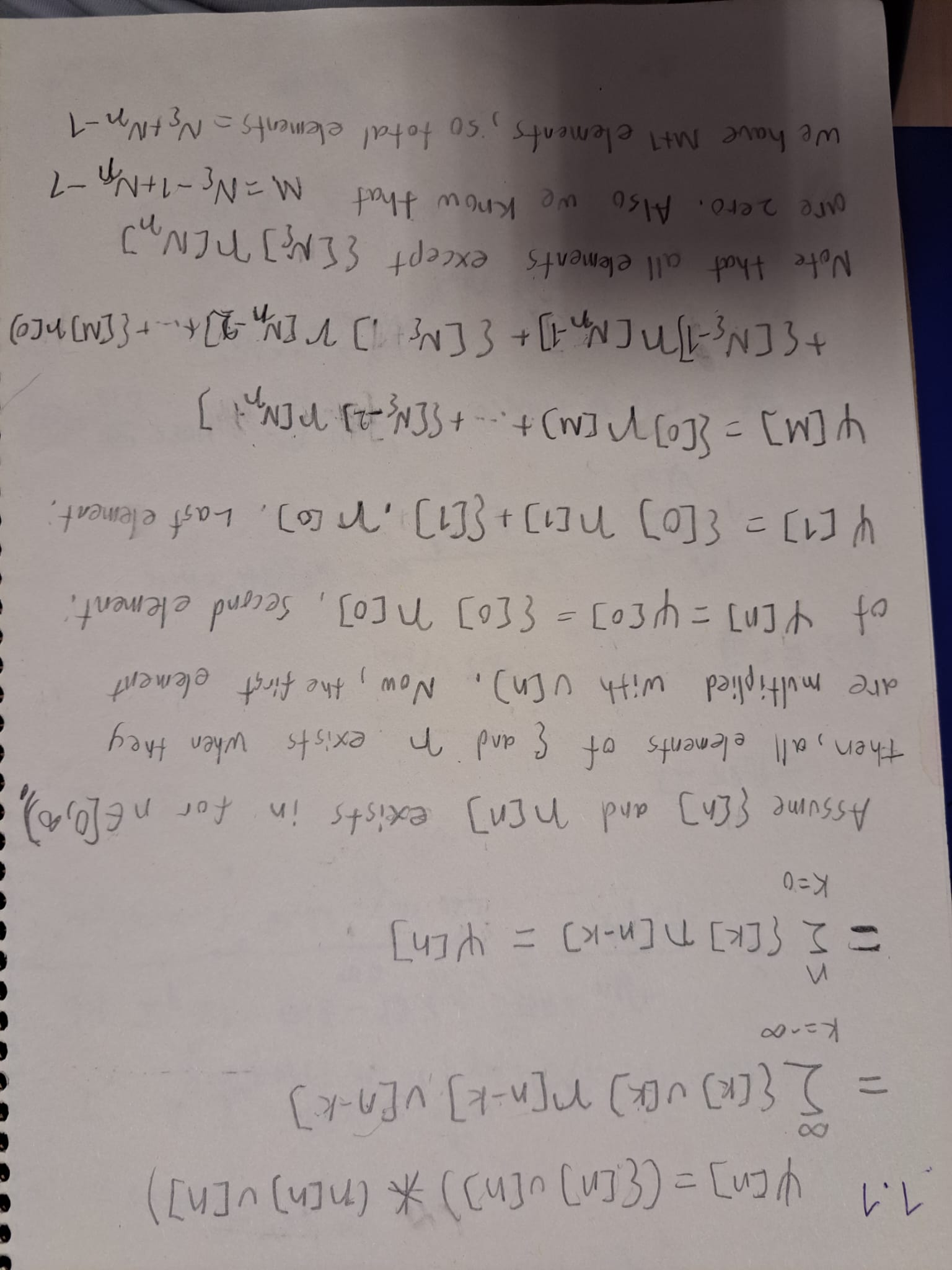
**LAB ASSIGNMENT 2**

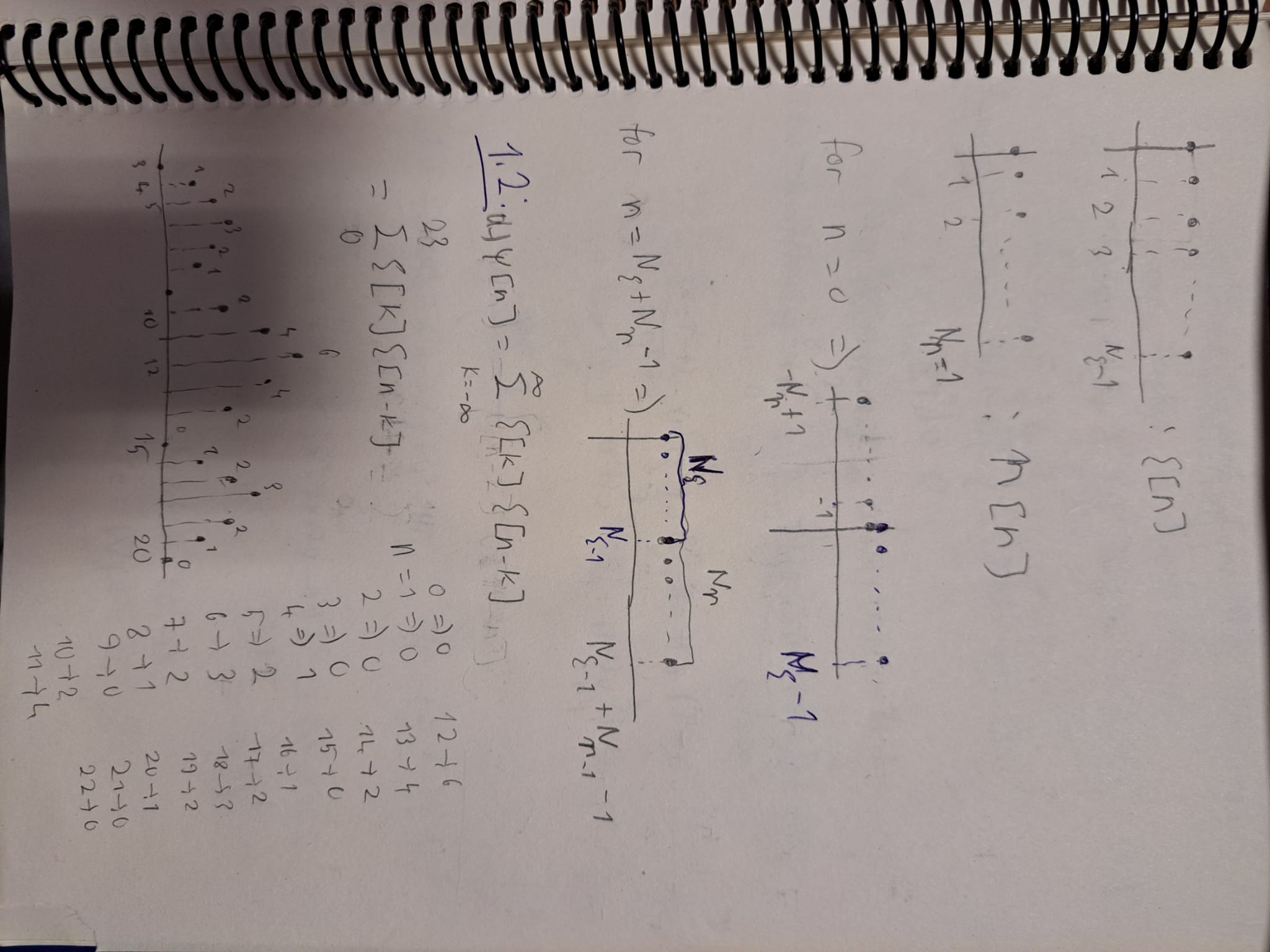
**1) INTRODUCTION:**

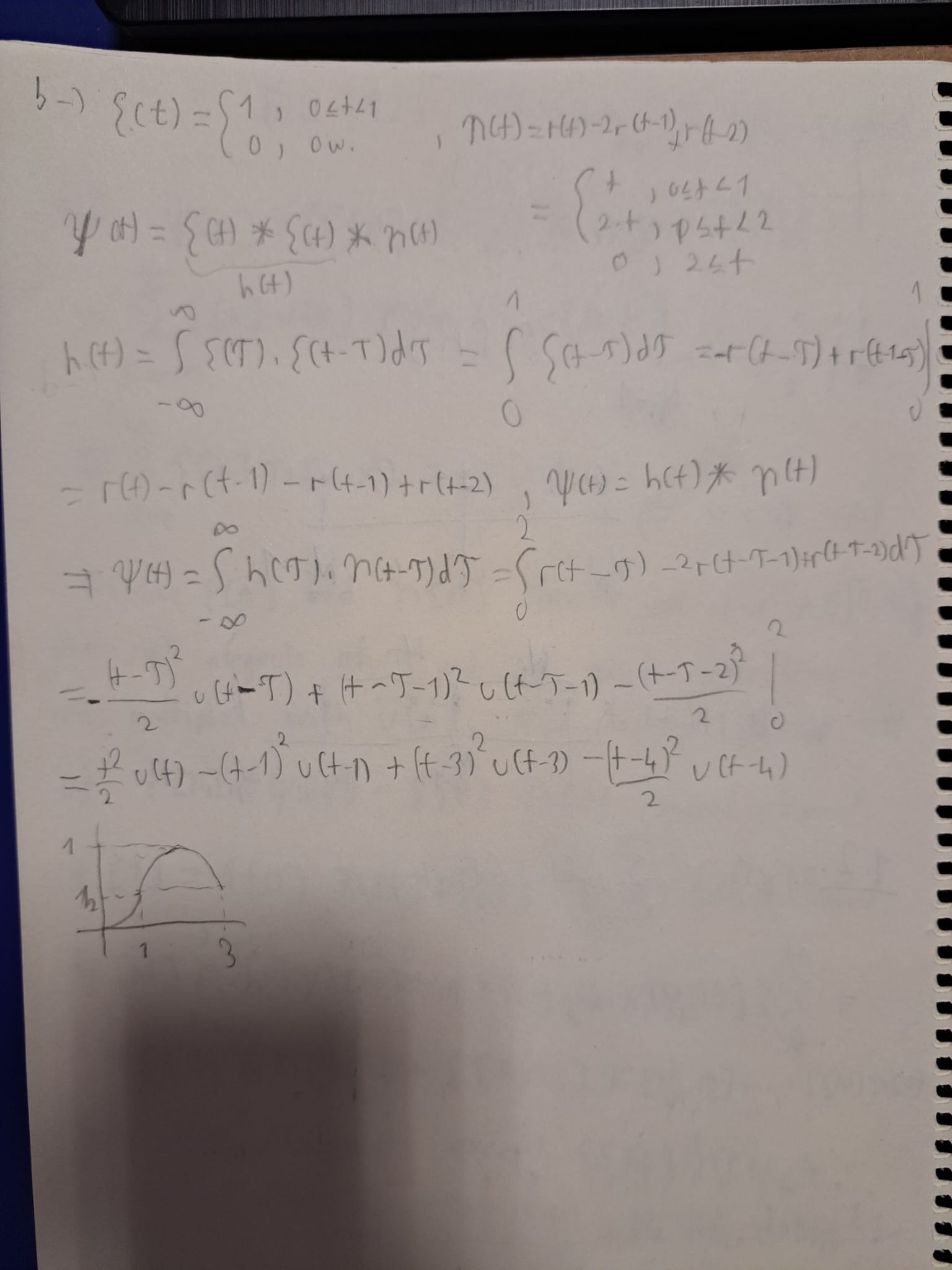
In this lab, I work with convolution and cross-correlation operations on signals. I studied the meaning and visualization of these operations and develop an understanding on the scope of signals and systems perspective. In first part, I develop the proof of convolved signals length and did some operations on the given signals. In the second part, I create a convolution function. In the third part, I created a convolution animation and verified the results of Part 2. In the fourth part, I applied cross-correlation operation onto my speech signal and detect the occurrences of the extracted part of my voice and the whole of my voice via this operation. In the fifth part, I added noise into gTTS generated audio file and observed noise’s effects on the audio signal and cross-correlation operation.

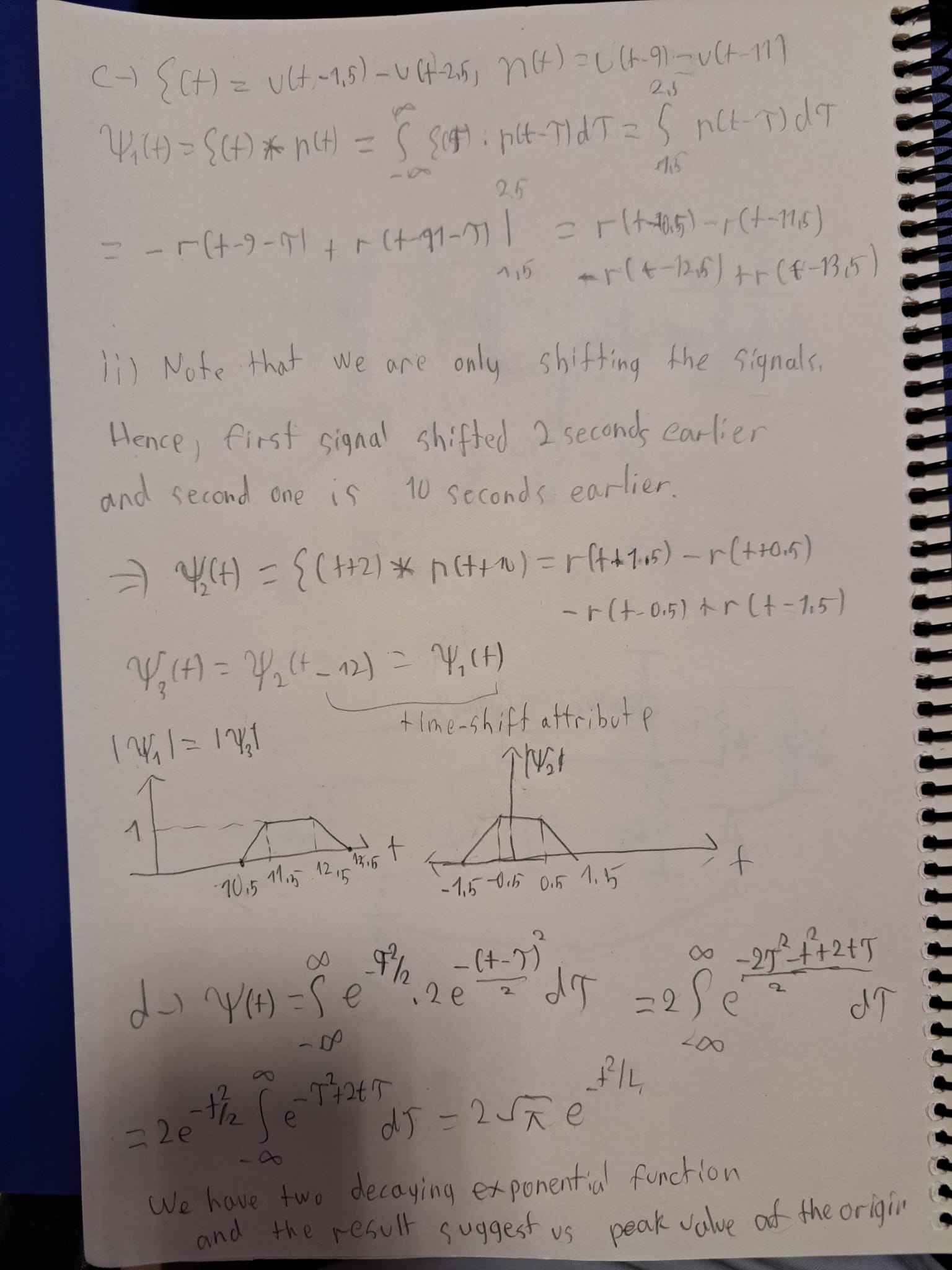
**2) LAB:**

* Part 1:









* Part 2:

1. **Implementing ConvFUNC:**

The input sequences "x" and "h" indicate two discrete signals that are send as inputs to the ConvFUNC function. The variables "Nx," "Nh," and "Ny" stand for the lengths of the "x," "h," and "y" output sequences, respectively. These variables calculated in the first three line of the code for ease. Additionally, zeros are used to initialize the output sequence "y." The input sequence "x" is flipped using the "flip(x)" function so that signals can slides on themselves in a correct fashion. Since we are allowed to use one for loop, I used dot product between the signals to obtain area calculation. In order to do so, I added the zeros to the start and end of the non-flipped signal (x) to extend it to the length of the flipped array (h) to save the initial and final values of convolved function. For every position 'n' in the output sequence 'y,' a corresponding portion of the extended sequence 'X' is selected to have the same length as the input sequence 'x'. The output sequence "y"s element stands in position "a" is labeled with the convolution result. To sum up, convolution takes one input sequence and flips it before adding up all of the signal areas below. Corresponding code is available below:

function [y] = ConvFUNC(x, h, int\_length)

if nargin < 3 || isempty(int\_length)

int\_length = 1;

end

Nx = length(x);

Nh = length(h);

Ny = Nx + Nh - 1;

y = zeros(1, Ny);

concX = zeros(1,Ny-Nx);

X=[concX,x,concX];

for n = 1:Ny

conv\_1 = X(n:n+Nh-1);

conv\_2 = flip(h);

y(n) = int\_length \* dot(conv\_1,conv\_2);

end

end

1. **Testing the ConvFUNC:**

In this section, ConvFUNC is applied to the a) given in Part 1.2, and the verification of the findings is assessed by comparison. First, given signals are defined with the matlab arrays "x" and "h". Next, 'y' is returned from the ConvFUNC function as the convolved array. One can verify the same results obtained with the a) part of Part 1.2 and the below figures last plot.

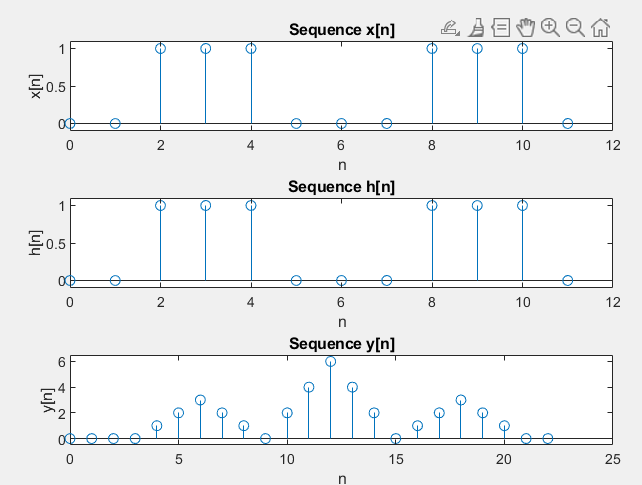
****

Fig. :

Corresponding code exists below:

x = [0,0,1,1,1,0,0,0,1,1,1,0];

h = [0,0,1,1,1,0,0,0,1,1,1,0];

y = ConvFUNC(x,x);

subplot(3, 1, 1);

stem(0:length(x)-1, x);

ylim([-0.1,1.1])

xlabel('n');

ylabel('x[n]');

title('Sequence x[n]');

subplot(3, 1, 2);

stem(0:length(h)-1, h);

ylim([-0.1,1.1])

xlabel('n');

ylabel('h[n]');

title('Sequence h[n]');

subplot(3, 1, 3);

stem(0:length(y)-1, y);

ylim([-0.5,6.5])

xlabel('n');

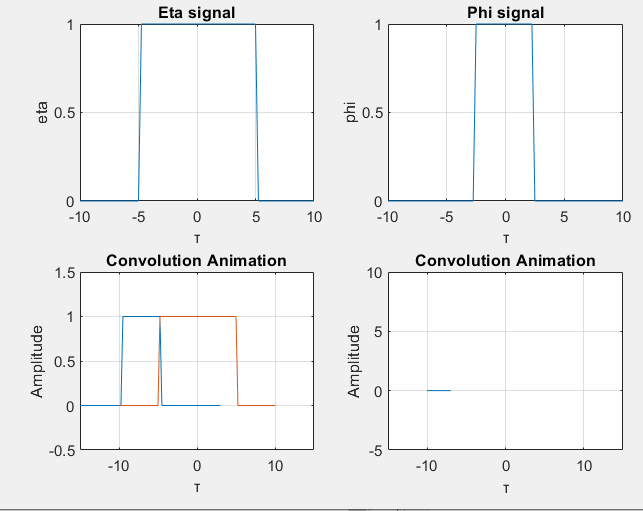
ylabel('y[n]');

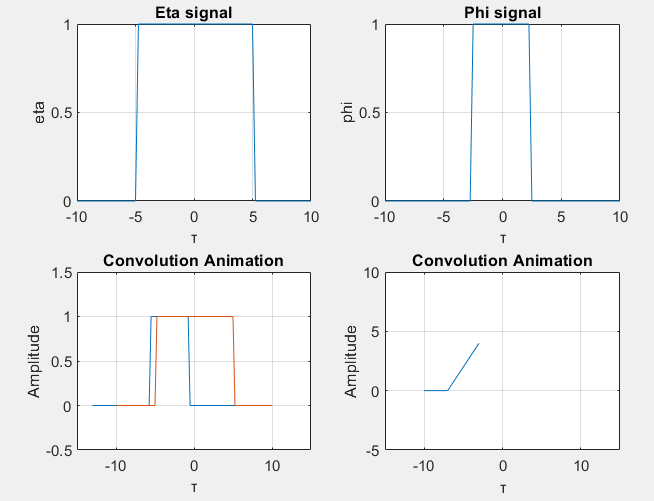
title('Sequence y[n]');

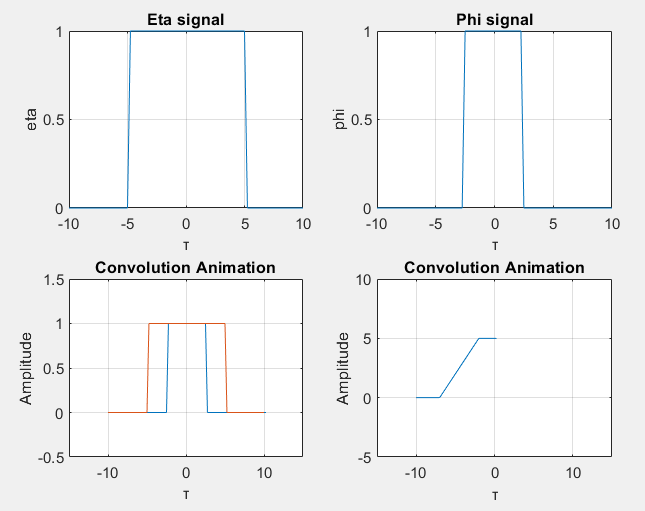
* Part 3:

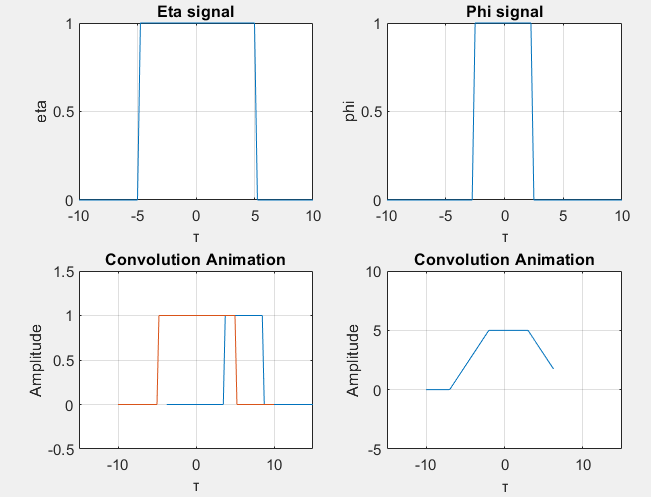
In order to visualize the convolution process of two signals, ξ(t) = u(t+5)−u(t−5) and η(t) = u(t+2.5)−u(t−2.5), this section generates an animated plot. I created an time array "tao" with a sampling interval of 0.25, ranging from -10 to 10 to encapsulate the given signals time length. Next, I created “eta” and “phi” with using logic arguments and converting them from logical arguments to numeric ones with "double" function.

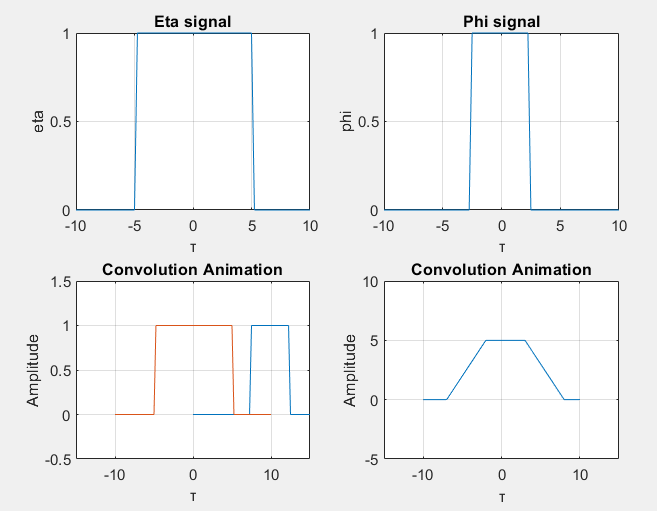
I used the ConvFUNC function to compute the signals' convolution and eliminate the additional extends at the initial and final sections stemmed from additional length occurrence of convolution. The reason behind this is matching the time array “tao” with the convolved signal “omega”. In order to the sliding effect of the animation of the second signal “phi” send to ConvFUNC decreased the "tau" value and increased one interval length all elements of this array per each iteration. For the “omega” signal, I plotted it starting from first element to the “ii”th element to obtain part by part constructing process as “phi” slides on “eta” signal. I defined “x” and “y” ranges so that I obtained stable plot surface.











Corresponding code is shown below:

tao = -10:0.25:10; %samplela çarp

eta = double(tao>-5) - double(tao>5);

phi = double(tao>-2.5) - double(tao>2.5);

omega = ConvFUNC(eta,phi, 0.25);

tao\_new = -length(omega)/8:0.25:(length(omega)-1)/8;

phi\_flipped = fliplr(phi);

omega = omega(double(length(omega)/4):double(length(omega)\*3/4));

tao\_slided = -20:0.25:0;

figure;

for ii = 1:length(omega)

subplot(2, 2, 1);

plot( tao, eta);

xlabel('τ');

ylabel('eta');

title('Eta signal');

grid on;

subplot(2, 2, 2);

plot( tao, phi\_flipped);

xlabel('τ');

ylabel('phi');

title('Phi signal');

grid on;

subplot(2, 2, 4);

plot( tao(1:ii), omega(1:ii));

xlabel('τ');

ylabel('Amplitude');

title('Convolution Animation');

xlim([-15, 15]);

ylim([min(omega)-5, max(omega)+5]);

grid on;

subplot(2, 2, 3);

plot(tao\_slided, phi\_flipped, tao, eta);

title('Convolution Animation');

xlabel('τ');

ylabel('Amplitude');

xlim([-15, 15]);

ylim([min(phi)-0.5, max(phi)+0.5]);

grid on;

% Pause to create animation effect

pause(0.05);

drawnow;

tao\_slided = tao\_slided + 0.25;

end

* Part 4:

1. **Defining Cross-Correlation:**

Cross correlation operation mathematically defined as below:

y[n] = x[n] ★ h[n] =

where indicates the conjugate of .This has similarities with the convolution function. The given function to calculate is below:

★

Open summation form of the given function is as follows;

The unit step function, u[n], is 0 for n<0 and 1 when n≥0. Hence, we can simplify this as below:

The length of the output sequence same as convolution function: = . The proof of this is given in Part 1.Now we can see that cross-correlation operation same as convolution whose first input is conjugated and the second input is time-wise reversed.

★ =

1. **Building a Basic Speech Recognition Algorithm:**

ID = 22102758

, ,

Firstly I set to sampling frequency (fs) 8192 Hz. This represents the quantity of my voice data samples that generated each second. The duration of my voice record is set to ten seconds. Then I initialize the recObj with 16 bits per sample, 8192Hz sampling frequency and 1 channel. Then, I record my voice for 10 second duration with recordblocking function. Then using the getaudiodata function, I obtained corresponding record data and store it in voiceData variable. I saved the voiceData with a sampling frequency of fs = 8192 to my computer as "TotalNumber.flac" file.

Related code with this part given below:

fs = 8192;

recordingDuration = 10;

recObj = audiorecorder(fs, 16, 1);

disp('Start speaking.');

recordblocking(recObj, recordingDuration);

disp('End of recording.');

voiceData = getaudiodata(recObj);

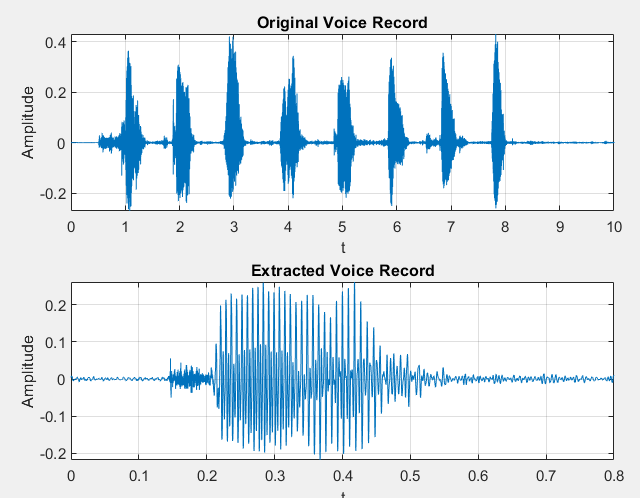
filename = 'TotalNumber.flac';

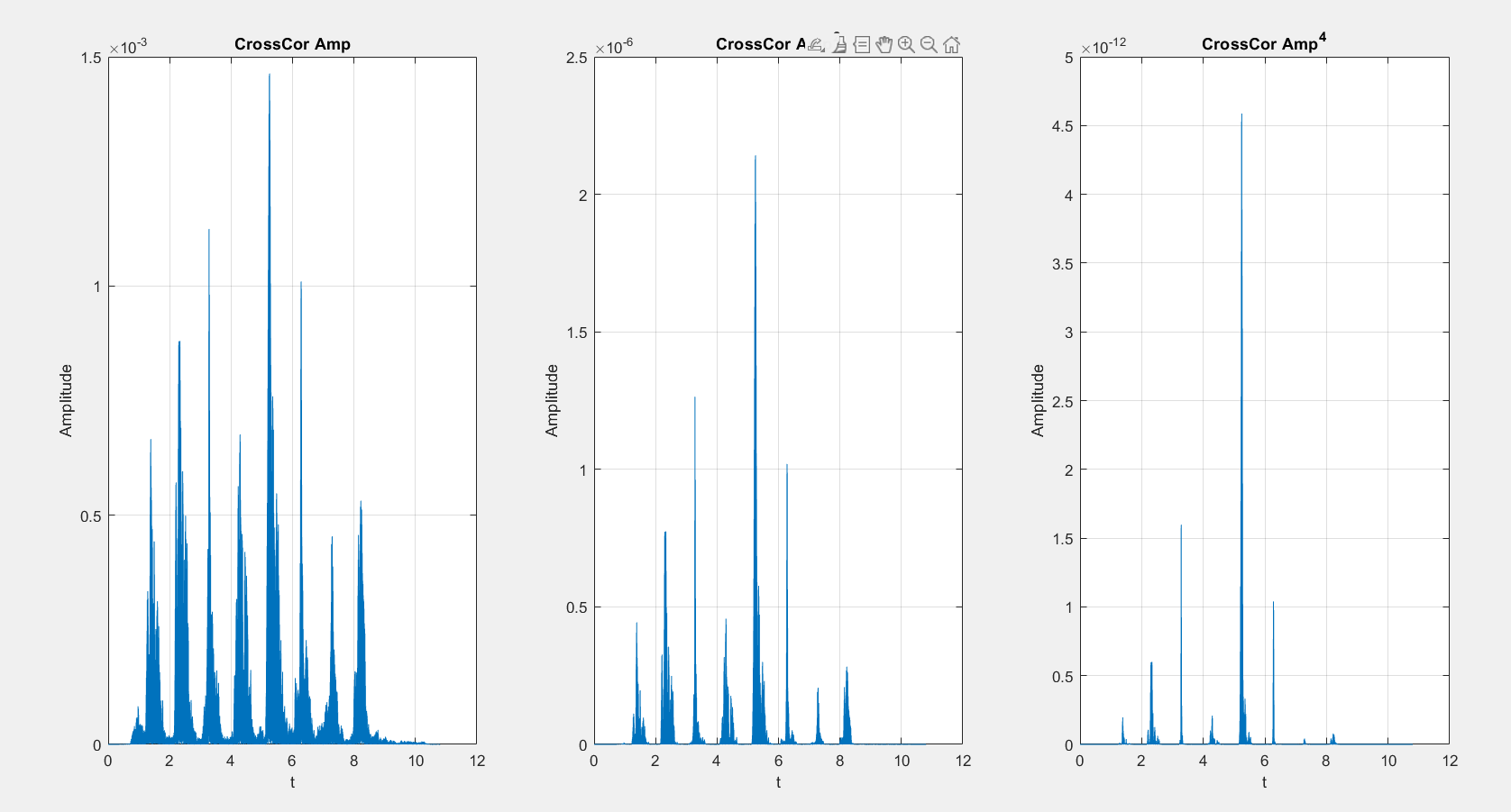
audiowrite(filename,voiceData,fs);

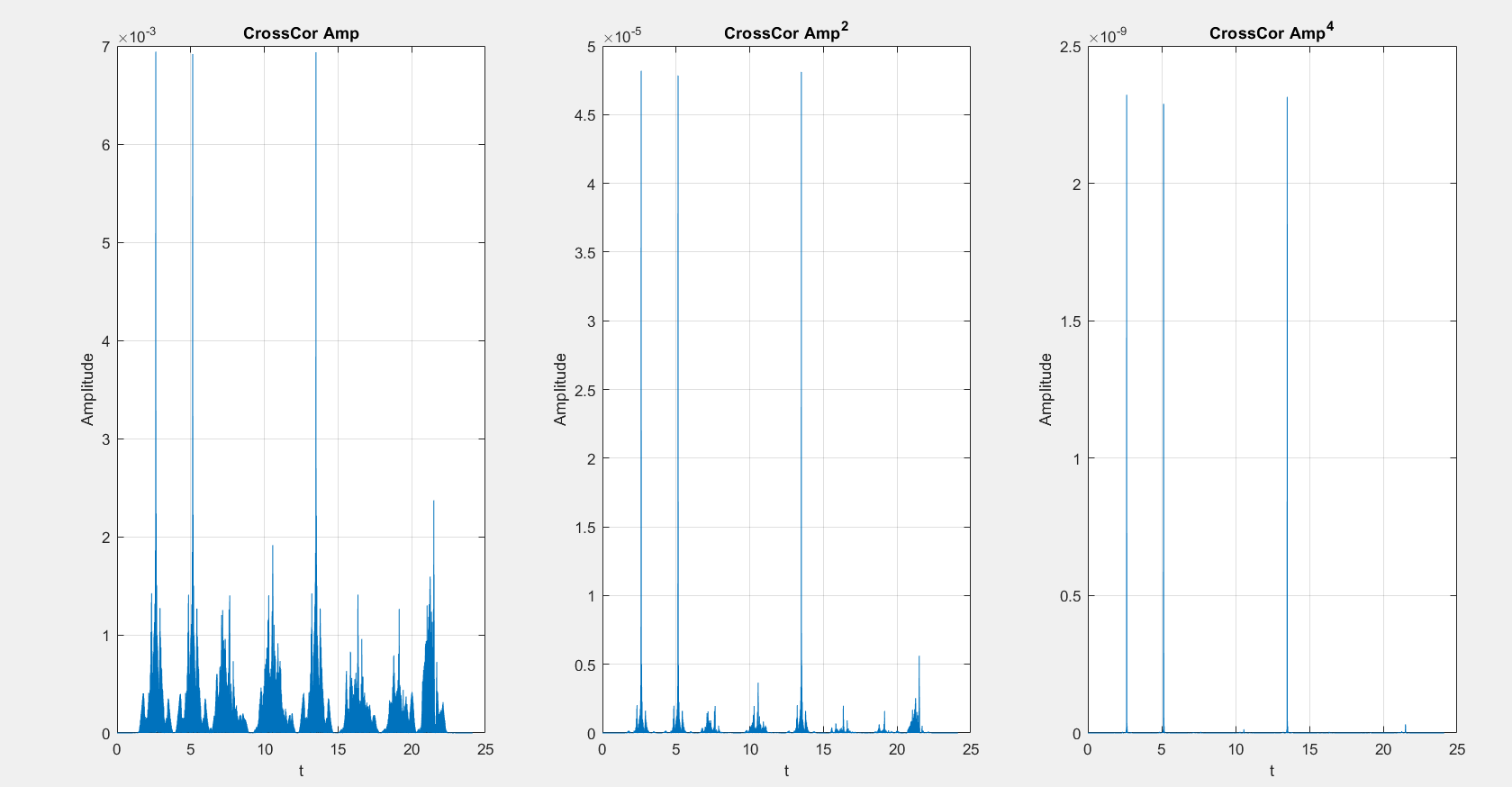
Fort he remaining part, I used Google’s gTTS module to create voice file which is the record of my ID and stored it in “TotalNumber2.flac” file. Then, I stored gTTS and my recorded voice datas in voiceData and voiceData\_my variables respectively. During this operation, I extracted the sampling frequencies in fs and fs\_my variables as well. After that, I made time vectors, plotted the voices, and listened to the sounds. In these plots, I detect the numbers that I voiced and find the starting and ending time of this number. Next, I extracted the value of n1 (2) in my voice record and store it in 'n1.flac' voice file. Then, I take the complex conjugate of the original record and flipped the extracted voice file, and used ConvFUNC which is indeed performed a cross-correlation operation with to given voice data. I stored the output in phi variable and plotted the retrieved signal phi’s amplitude, second power of the amplitude, and fourth amplitude power.

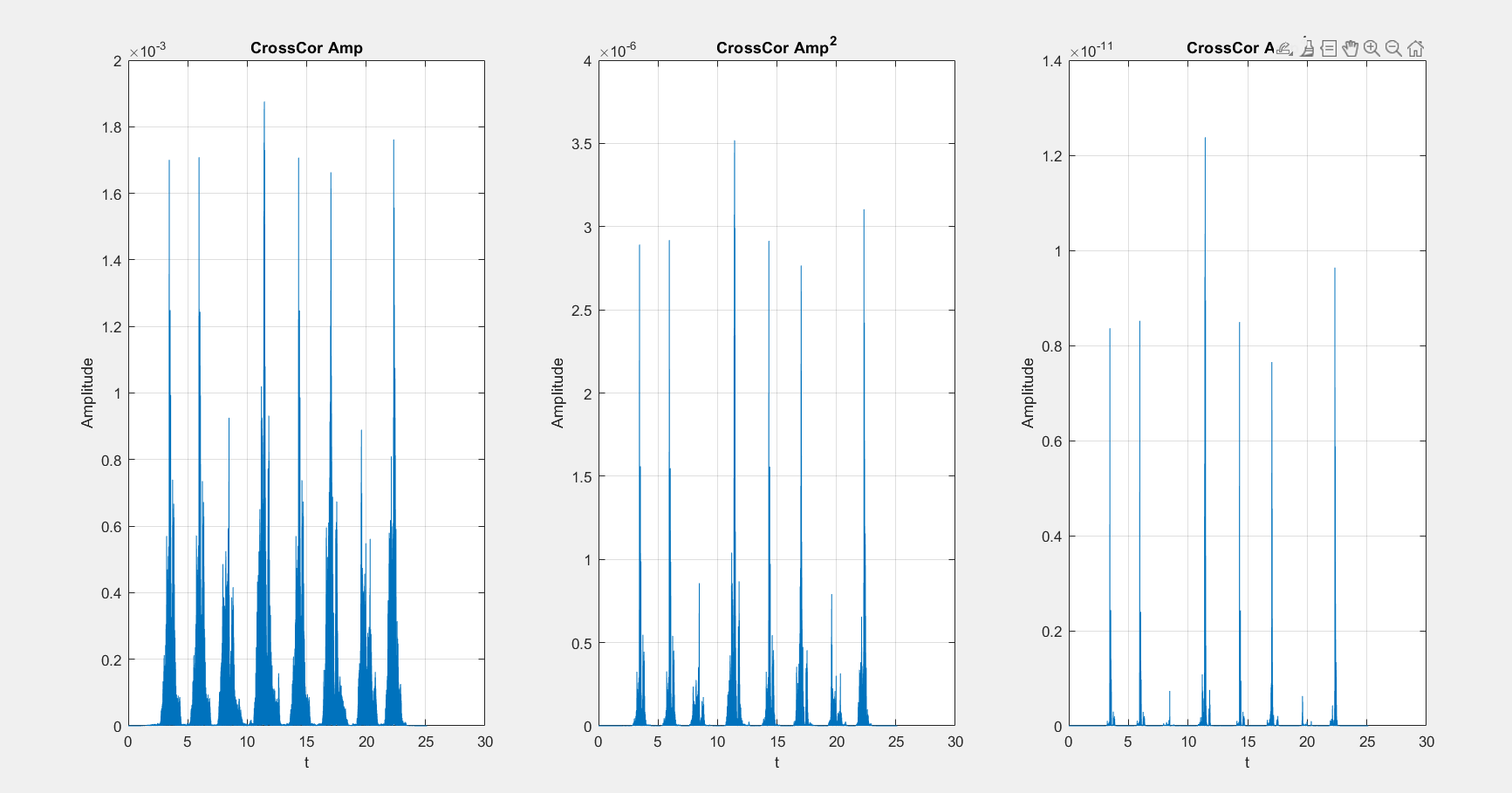
Following figures show the results. Firstly, observing my voice plot, I couldn’t say the n1 (2) in a same fashion. I extracted the third of that number (time-wise: 4.7s – 5.6s). When we observe the amplitudes it can be easily seen that that occurrence caught with high amplitude. However, third index (1) and sixth index (7) has higher amplitude than the other occurrences of n1. When we observe the original signal, it is detectable that this is due to total area covered with n1 and my voice signal. Moreover, different numbers’ plots are different as well. So I can distinguish the different numbers. When I take the square and fourth power of the amplitude, I see that irrelevant numbers’ amplitudes are diminishing. Hence, raising high powers allows us to detect specific occurrences and save it from other noisy errors, such as original n1 occurrence in the amplitude plot.

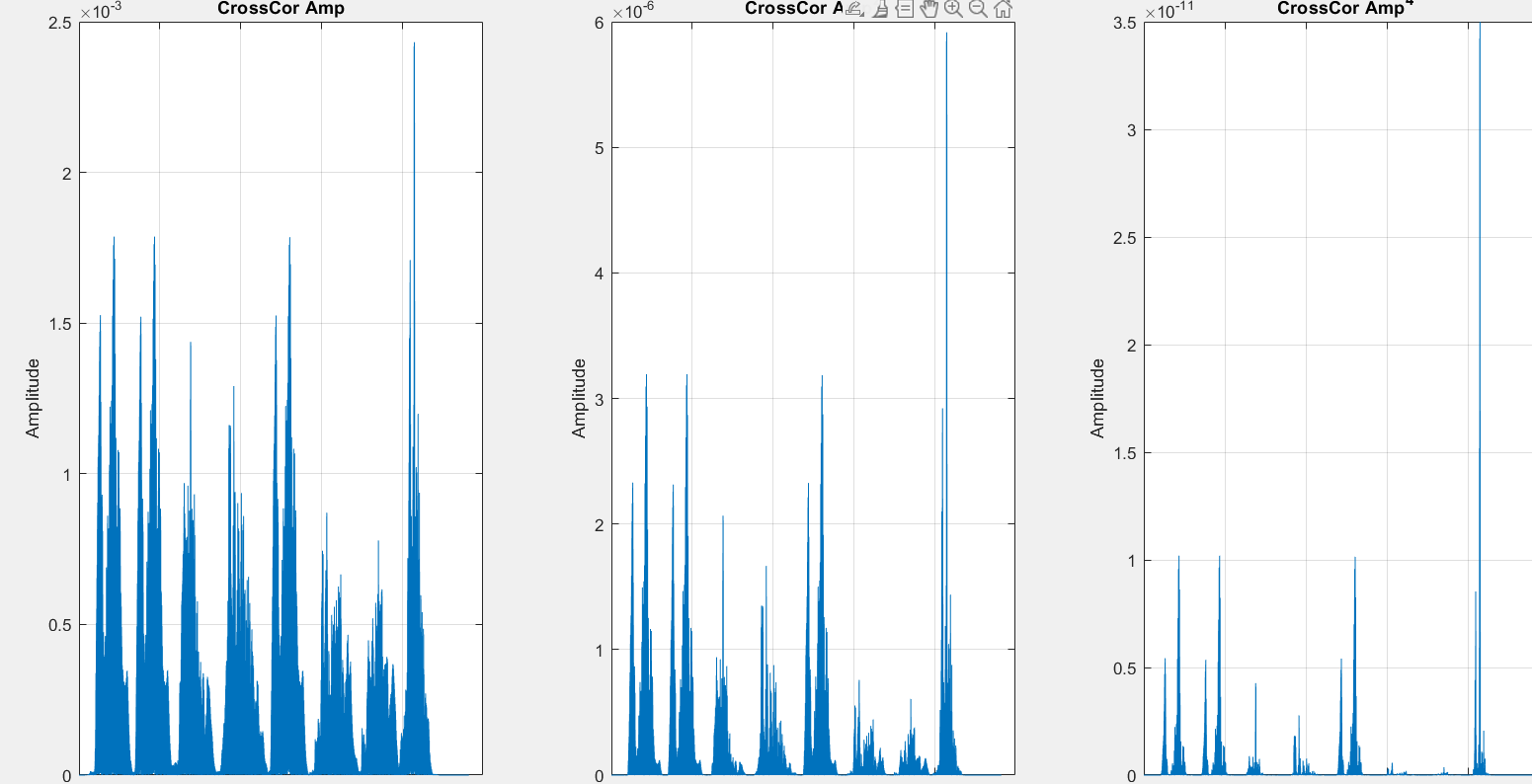
When I apply the same operations on the gTTS voice record I obtained the Fig. … Results show that it is clear to success of cross-correlation in language processing. gTTS has same voice characteristics. Therefore, it is more successful to catch same numbers. However, when I tried to cross-correlation with n2 and gTTS voice record, results are not meaningful and doesn’t suggest any occurrence. While, applying ConvFUNC directly gives meaningless results as well. Since we flipped the n1 signal, reverse matching occurred in only n1 (2) and 8. For the last thing, when I do the cross-correlation with n1 obtained from gTTS and my voice record, it misses the first occurrence of n1 but able to detect second and third occurrence very well.

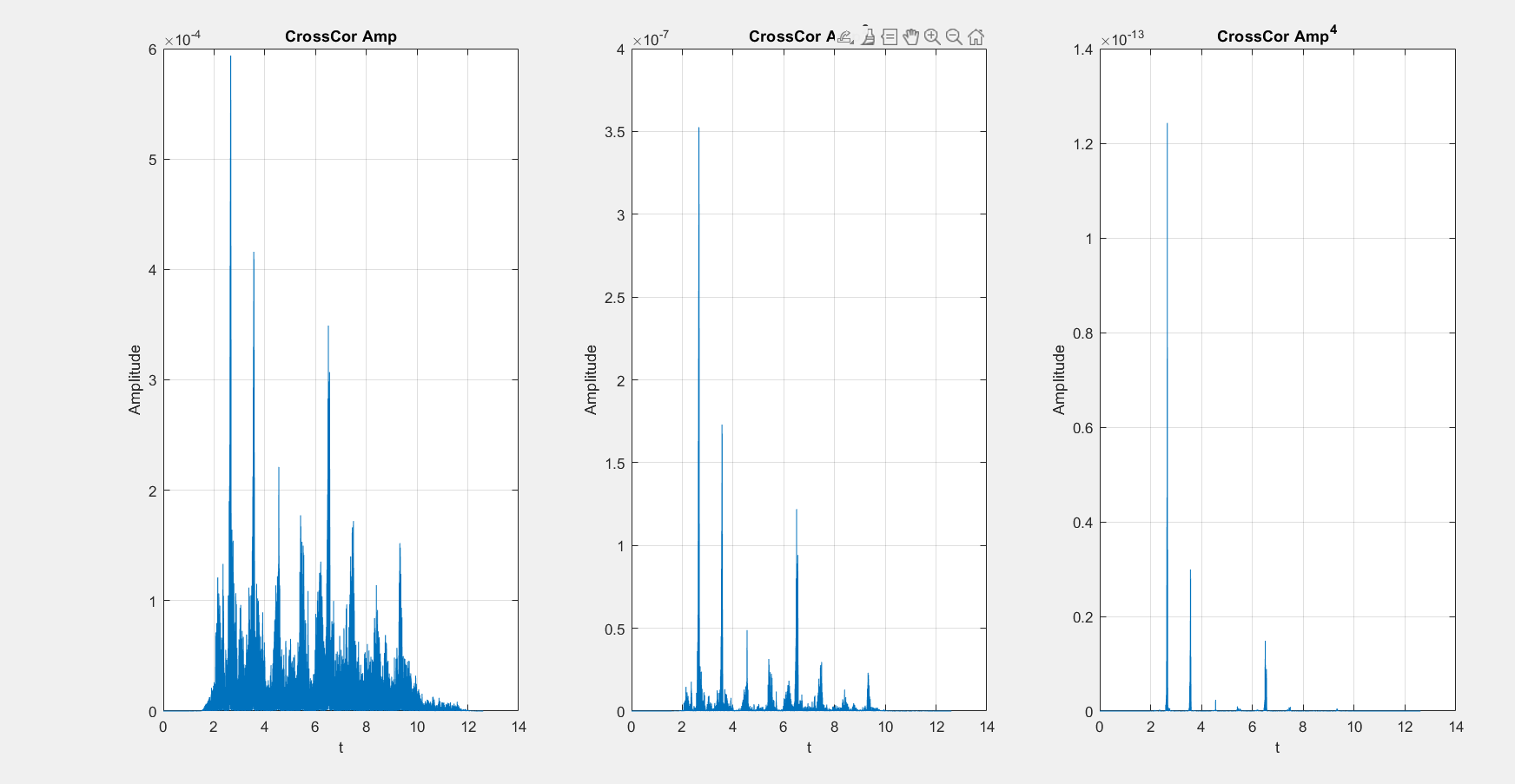












Corresponding codes are below:

[voiceData, fs] = audioread(' TotalNumber2.flac'); %pythondan gelen

[voiceData\_my, fs\_my] = audioread('TotalNumber.flac');

%[n2, fsx] = audioread('2.flac');%kendi kaydettiğim

audio\_len = length(voiceData);

t1 = 1:audio\_len;

t1=t1/fs;

t\_my = linspace(0, 10, 10\*fs\_my);

figure;

%soundsc(voiceData, fs);

%soundsc(voiceData\_my, fs\_my);

subplot(2, 1, 1);

plot( t1, voiceData);

xlabel('t');

ylabel('Amplitude');

title('gTTS Audio');

grid on;

subplot(2, 1, 2);

plot( t\_my, voiceData\_my);

xlabel('t');

ylabel('Amplitude');

title('My Voice');

grid on;

n1=2;

n2=6;

n1\_ext = voiceData\_my(4.7\*fs\_my:5.5\*fs\_my);

%soundsc(n1\_ext,fs\_my);

filename = 'n1.flac';

audiowrite(filename,n1\_ext,fs\_my);

n1\_phil = flip(n1\_ext);

cell\_array = reshape(n1\_ext,1,length(n1\_phil));

voiceData\_my = reshape(voiceData\_my,1,length(voiceData\_my));

phi = ConvFUNC(conj(voiceData\_my),cell\_array,1/fs\_my);

t\_n1 = (0:length(n1\_ext)-1) / fs\_my;

t\_phi = (0:length(phi)-1) / fs\_my;

figure;

subplot(2, 1, 1);

plot( t\_my, voiceData\_my);

xlabel('t');

ylabel('Amplitude');

title("Original Voice Record")

grid on;

subplot(2, 1, 2);

plot( t\_n1 , n1\_ext);

xlabel('t');

ylabel('Amplitude');

title("Extracted Voice Record")

grid on;

phi\_amp = abs(phi);

figure;

subplot(1, 3, 1);

plot(t\_phi , phi\_amp);

xlabel('t');

ylabel('Amplitude');

title("CrossCor Amp")

grid on;

subplot(1, 3, 2);

plot( t\_phi , phi\_amp.^2);

xlabel('t');

ylabel('Amplitude');

title("CrossCor Amp^2")

grid on;

subplot(1, 3, 3);

plot( t\_phi , phi\_amp.^4);

xlabel('t');

ylabel('Amplitude');

title("CrossCor Amp^4")

grid on;

[n1, fsx] = audioread('2.flac');%kendi kaydettiğim

t\_n1 = 1:length(n1);

t\_n1 = t\_n1/fsx;

n1\_phil = flip(n1);

cell\_array = reshape(n1\_phil,1,length(n1\_phil));

voiceData = reshape(voiceData,1,length(voiceData));

phi = ConvFUNC(conj(voiceData\_my),cell\_array,1/fsx);

soundsc (phi,fs\_my);

phi\_amp = abs(phi);

t\_phi = (0:length(phi)-1) / fs\_my;

figure;

subplot(1, 3, 1);

plot(t\_phi , phi\_amp);

xlabel('t');

ylabel('Amplitude');

title("CrossCor Amp")

grid on;

subplot(1, 3, 2);

plot( t\_phi , phi\_amp.^2);

xlabel('t');

ylabel('Amplitude');

title("CrossCor Amp^2")

grid on;

subplot(1, 3, 3);

plot( t\_phi , phi\_amp.^4);

xlabel('t');

ylabel('Amplitude');

title("CrossCor Amp^4")

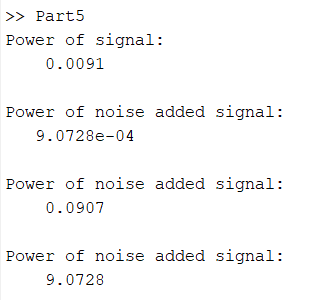
grid on;

* Part 5:

**A. Observing the Effects of SNR:**

I import the audio signal of my ID, into audio\_array. Then, I created the corresponding time vector. I stored its length in audio\_len. Then, I calculated the power of this signal with the below formula:

I used a FOR loop to test the various SNR values and listen to the noisy voices for each one. The algorithm scans an audio file, determines its power, adds noise at various SNR levels, and plays the resulting audio for each condition. The added noise has a rustling effect on the original audio. I was not able to hear number for the SNR = 0.001 value. My . You can find the detailed outputs in the following figure:



Related codes are below:

[audio\_array, fs] = audioread(' TotalNumber2.flac');

audio\_len = length(voiceData);

t1 = 1:audio\_len;

t1=t1/fs;

google\_array = reshape(audio\_array,1,length(audio\_array));

power\_array = reshape(audio\_array,1,length(audio\_array)).^2;

p\_signal = sum(power\_array)/length(power\_array);

disp("Power of signal:")

disp(p\_signal);

snr\_val = [10,0.1,0.001];

rng (5)

for i = snr\_val

p\_noise = p\_signal/i;

disp("Power of noise added signal:")

disp(p\_noise)

awgn = sqrt ( p\_noise ).\* randn ([ audio\_len , 1]);

awgn = reshape(awgn,1,length(awgn));

noisy\_audio = awgn + google\_array;

soundsc(noisy\_audio,fs);

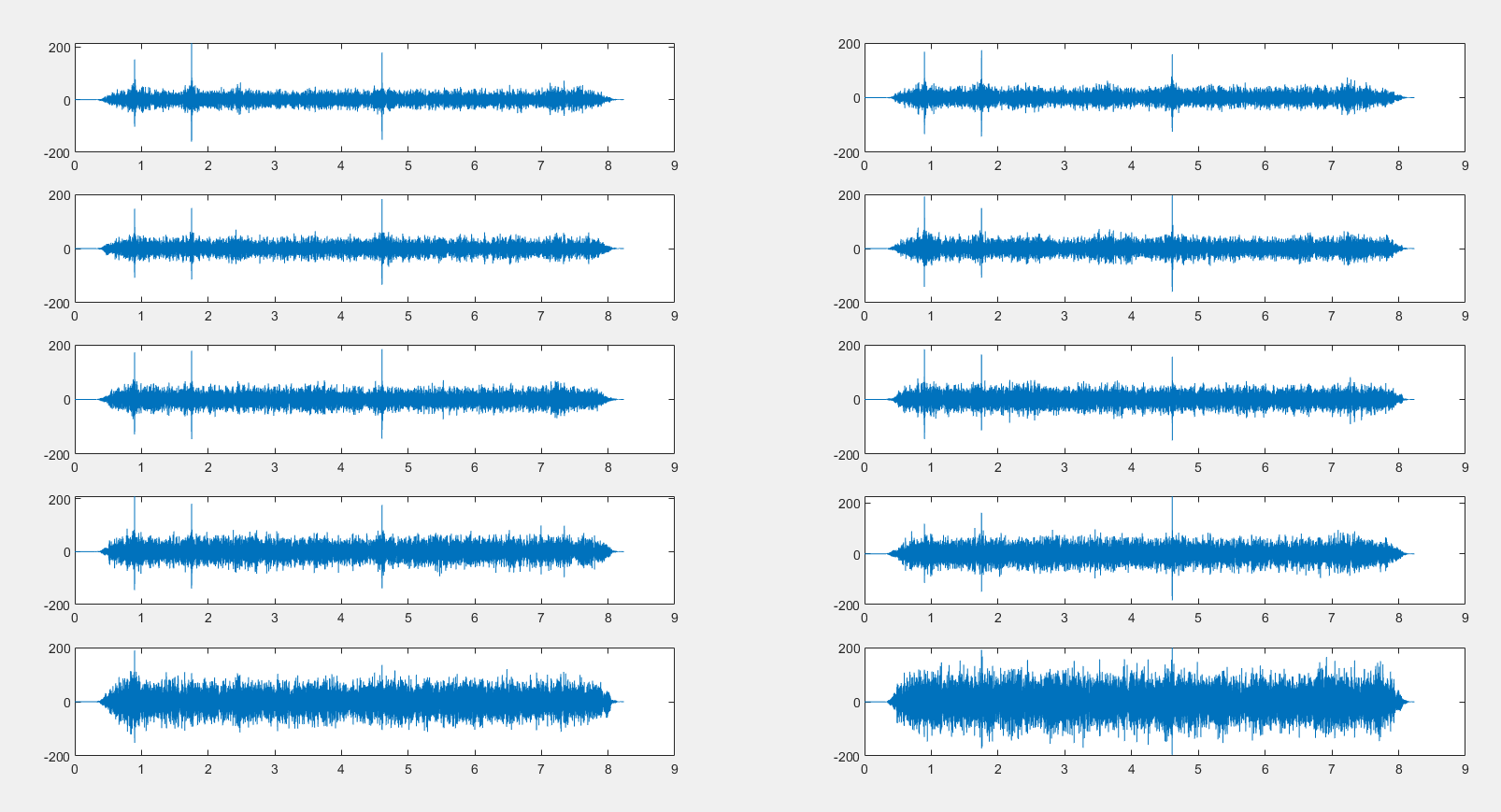
pause(8)

end

**B. Detecting the SNR Limit:**

This section looks at how varying SNR settings affect the cross-correlation results. I began by reading two audio files: Gtts' voice and the n1 value. I find the original signal’s power and utilized cross-correlation with the noise-added signals with various SNR values. Plotting the findings in subplots shows how SNR influences cross-correlation outcomes.

As shown in the plots in the following figure, while the SNR value is closer to 0.01, the algorithm can detect the n1 value in my ID number; however, SNR values below 0.003 the algorithm has problems with detecting them accurately. The signal becomes increasingly noisy, and the system broke for below of 0.003. In the final plot, with SNR = 0.001, n1 detection is not visible. Although one can make guesses, confidence is low.



Following codes give the same results:

[audio\_array, fs] = audioread(' TotalNumber2.flac');

[filter, fs\_n] = audioread('2.flac');

filter = reshape(filter,1,length(filter));

filter = flip(filter);

snr = 0.01:-0.001:0.001;

audio\_len = length(audio\_array);

t1 = 1:audio\_len;

t1=t1/fs;

google\_array = reshape(audio\_array,1,length(audio\_array));

power\_array = reshape(audio\_array,1,length(audio\_array)).^2;

p\_signal = sum(power\_array)/length(power\_array);

rng(5);

figure;

for i = snr

p\_noise = p\_signal/i;

disp(i)

awgn = sqrt ( p\_noise ).\* randn ([ audio\_len , 1]);

awgn = reshape(awgn,1,length(awgn));

noisy\_audio = awgn + google\_array;

output = ConvFUNC(noisy\_audio,filter);

t\_new = (1:length(output))/fs;

index = find(snr == i);

subplot(5,2,index);

plot(t\_new , output);

end