Bilkent University
Department of Electrical and Electronics Engineering

EEE 321: Signals and Systems
Lab Assignment 2

In this lab, you will implement the convolution and cross-correlation operations. Please work on this assignment before coming to the laboratory. At the end of the laboratory session, you must show your completed work for all parts. After the lab session, you will have more time to format your completed work as a report and submit it on Moodle. Some parts will be performed by hand, and others will be done using MATLAB. Please address all the questions asked in the assignment and include all your codes as text and derivations for all parts. Before the submission due date, make sure to upload your work as a readable, well-formatted single PDF file. Note that the MATLAB codes will be tested on MATLAB R2023a. Ensure your code does not raise an error or a warning in earlier versions. Do **NOT** forget to add proper captions, axis labels, and titles for any plot you provide.

1 Part 1

1.1 Defining Convolution

For the continuous-time signals $x(t), h(t)$ and discrete-time sequences $x[n], h[n]$, the continuous and discrete convolution operations can be defined as in Eqns.(1) and (2), respectively.

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)\, h(t-\tau) d\tau \tag{1}$$

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]\, h[n-k] \tag{2}$$

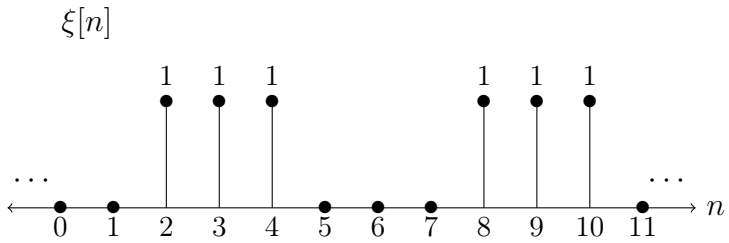
Let the sequences $\xi[n]u[n]$ and $\eta[n]u[n]$ have finite durations with length N_ξ and N_η, respectively, and arbitrary positive real values; where $u[n]$ denotes the unit-step function. Considering these defined signals and discrete convolution defined in Eq.(2), express $\psi[n] = (\xi[n]u[n]) * (\eta[n]u[n])$ explicitly as an open summation of elements of

$\xi[n]$ and $\eta[n]$, for finite positive integer values of N_ξ and N_η. What would be the length of $\psi[n]$? Support your result with graphical representations. How can you explain this discrete convolution operation in plain language as if you explain it to an early high school student?
Hint: Using the graphical representations as supporting tools may be helpful.

1.2 Evaluating Convolution

In this part, you will be evaluating the convolution operation by hand. Please obtain both mathematical solutions and graphical representations. Make sure you present your results neatly so that they are easy to read and follow.

a) For the sequence $\xi[n]$ defined below, evaluate $\psi[n] = \xi[n] * \xi[n]$.



b) Let $\xi(t) = u(t) - u(t-1)$, and $\eta(t) = r(t) - 2r(t-1) + r(t-2)$, where $u(t)$ and $r(t)$ denote the unit-step and ramp signals, respectively. Evaluate $\psi(t) = \xi(t) * \eta(t) * \xi(t)$.

c) Let $\xi(t) = u(t-1.5) - u(t-2.5)$, and $\eta(t) = u(t-9) - u(t-11)$, where $u(t)$ denotes the unit-step signal. Evaluate:

i) $\psi_1(t) = \xi(t) * \eta(t)$.

ii) $\psi_2(t) = \xi(t+2) * \eta(t+10)$, and then find $\psi_3(t) = \psi_2(t-12)$.

Comment on your results.

d) Let $\xi(t) = e^{\frac{-t^2}{2}}$, and $\eta(t) = 2e^{\frac{-t^2}{2}}$. Evaluate $\psi(t) = \xi(t) * \eta(t)$. Comment on the result of the convolution operation with these two signals.

2 Part 2

2.1 Implementing Convolution

Concerning the derivation indicated in Part 1.1, write a MATLAB function that computes the discrete convolution of two sequences. While writing your function, you can use up to **ONE for** loop. Using either another type of loop or more than one **for** loop is **not allowed**. Your function should look like:

function [y] = ConvFUNC(x,h) where

- **x** of size $1 \times N_x$ represents the first sequence to be convolved.
- **h** of size $1 \times N_h$ represents the second sequence to be convolved.

- **y** of size $1\times N_y$ represents the output sequence of the convolution operation.

Save your function as a separate file in the same directory you work in, to be used in the proceeding sections.

2.2 Testing the Convolution Function

Regarding the **ConvFUNC** developed in Part 2.1, to test your function, find the result of a) indicated in Part 1.2 using **ConvFUNC**. Using MATLAB's **subplot** and **stem** functions, plot $x[n], h[n]$ and $y[n]$ with proper indications stated in the introductory paragraph.

3 Part 3

3.1 Creating Convolution Animation

In this part, you will be creating an animation for the convolution operation of two sequences to better observe and learn this operation.

Before starting your animation, you should first properly define the signals to be convolved. Let the two signals to be convolved be defined as $\xi(t) = u(t+5)-u(t-5)$ and $\eta(t) = u(t+2.5)-u(t-2.5)$, respectively, where $u(t)$ refers to the unit-step function. To create these signals, create a time array named τ with a sampling interval of $S_i = 0.25$, for which select proper end points so that the sampled sequences $\xi[n]$ and $\eta[n]$ can properly be displayed within these limits. Utilizing τ, create the sequences $\xi[n] = \xi(nS_i)$ and $\eta[n] = \eta(nS_i)$; and by using **plot** function, check whether you have created $\xi[n]$ and $\eta[n]$ properly (you do not need to provide these plots in your report). After creating $\xi[n]$ and $\eta[n]$ with the use of τ, using **ConvFUNC**, calculate $\psi[n] = \xi[n] * \eta[n]$ and plot $\psi[n]$ to check whether you have created $\psi[n]$ properly (again, you do not need to provide this plot in your report). Considering your explanation of the discrete convolution operation in plain language in Part 1.1, flip one of $\xi[n]$ or $\eta[n]$ and shift this sequence until it is positioned just before the other (assume the flipped and shifted sequence is selected as $\eta[n]$ in the following parts). Then, in a **for** loop sweeping an index —let this index be denoted as ii— from 1 up to the end of the length of $\psi[n]$, create a **subplot**. In this **subplot** plot

- $\xi[n]$ with the corresponding τ values,
- $\eta[n]$ with the corresponding τ values,
- $\eta[n]$ flipped and shifted right before $\xi[n]$ with the corresponding τ values,
- $\psi[n]$ starting from its initial value up to its iith value with the corresponding τ values.

As you sweep ii, update your figure so that the flipped and shifted $\eta[n]$ shifts one unit to the right by a sampling interval amount, and $\psi[n]$ is displayed from its initial value up to its updated iith value.

Note 1: As stated before, your codes will be tested on MATLAB R2023a. Hence, ensure that your code raises neither an error nor a warning in earlier versions.

Note 2: While observing your animated figure, make sure that **xlim** and **ylim** values are adjusted in a proper manner. You can find an example of an animated plot here. Your animation might run slower than the example provided. However, make sure that your animation has nothing less than this exemplified animation and gives the right results for different $\xi[n]$ and $\eta[n]$.

After creating your convolution animation, you can have fun inserting different input sequences and observing how they are convolved.

4 Part 4

4.1 Defining Cross-Correlation

For the discrete-time sequences $x[n]$ and $h[n]$, the discrete cross-correlation operation is defined as

$$y[n] = x[n] \star h[n] = \sum_{k=-\infty}^{\infty} x^*[k]\, h[n+k] \tag{3}$$

where $x^*[k]$ represents the complex conjugate of $x[k]$.

Let the sequences $\xi[n]u[n]$ and $\eta[n]u[n]$ have finite lengths with length N_ξ and N_η, respectively, and arbitrary positive real values; where $u[n]$ denotes the unit-step function. Considering these defined sequences and the discrete cross-correlation operation stated in Eq.(3), express $\psi[n] = (\xi[n]u[n]) \star (\eta[n]u[n])$ in open summation form of elements of $\xi[n]$ and $\eta[n]$, similar to what you have done in Part 1.1. What would be the length of $\psi[n]$? Support your result with graphical representations. Compare and contrast the discrete cross-correlation and discrete convolution operations. For the sequences $\xi[n]u[n]$ and $\eta[n]u[n]$, propose a pre-processing method to be applied on one of these sequences so that using this method and by applying the discrete convolution operation, the discrete cross-correlation operation is derived.

4.2 Building a Basic Speech Recognition Algorithm

In this part, you will be building a basic speech recognition algorithm by utilizing the indications in Part 4.1 and comparing the success of this algorithm in different cases.

The set $\Upsilon = \{0, 1, 2, \ldots, 8, 9\}$ contains the digits that can be used in a student's ID. Your student ID number —denoted as ID— consists of eight digits, some of which might repeat. Let ρ represent the array with length N_ρ, which includes the repetitive numbers in ID. Hence, the number that repeats first is saved as the initial item in this array. Similarly, let λ represent the array with length N_λ, including the digits not used from the set Υ. Next, calculate $\Delta = [(ID) \bmod 7]$, $\Delta_\rho = [(\Delta) \bmod N_\rho] + 1$, $\Delta_\lambda = [(\Delta) \bmod N_\lambda] + 1$. Select the numbers n_1 and n_2 from the sets ρ, λ with the index Δ_ρ and Δ_λ, respectively; where the initial number in a set has the index 1. For instance, assume that Ayşe has the student ID number $ID = 21401088$. Then, for Ayşe, $\rho = \{1, 0, 8\}, N_\rho = 3; \lambda = \{3, 5, 6, 7, 9\}, N_\lambda = 5$. Hence, $\Delta = [(21401088) \bmod 7] = 2$, $\Delta_\rho = [(2) \bmod 3] + 1 = 3$, $\Delta_\lambda = [(2) \bmod 5] + 1 = 3$, which indicates that the numbers

selected are both in the third index of the sets ρ and λ, which corresponds to the numbers $n_1 = 8$, and $n_2 = 6$. Concerning these indications, find n_1, n_2 for ID.

First, using MATLAB, record your voice pronouncing ID by indicating each digit one by one in 10-seconds long .flac file with the properties: 16 bits per sample, one channel, and sampling rate of $f_s = 8192$.

Note: While recording your voice, try to pronounce the same numbers as similarly as possible and minimize the environmental noise as much as possible.

After recording your voice, with the help of the Python code provided below, using a text-to-speech algorithm, record ID by changing *TotalNumber* below with ID. Do **NOT** replace anything else in the code below except the numbers for *TotalNumber*, as the dots and spaces add delays for better visualization in the proceeding parts.

```
from gtts import gTTS

TotalNumber = "2. 1. 4. 0. 1. 0. 8. 8."
Total_audio = gTTS(text=TotalNumber, lang="en", slow=True)
Total_audio.save("TotalNumber.flac")

for num in range(0, 10):
    strnum = ", " + str(num) + "."
    num_audio = gTTS(text=strnum, lang="en", slow=True)
    num_audio.save(str(num)+".flac")
```

After running the code segment above, as you have calculated, only proceed with *TotalNumber.flac*, n_1*.flac*, and n_2*.flac*; where n_1 and n_2 correspond to the numbers you have calculated above. Listen *TotalNumber.flac* in MATLAB using **soundsc** command, and make sure that the recording you have saved your voice is in the same fashion as this text-to-speech output.

After saving these recordings containing the audio signals indicated, load the .flac file containing your voice recording in MATLAB and ensure that both the audio signal and the sampling rate are properly loaded.

Then, create an array that will represent the time axis of your signal, which starts from 0 and lasts up to where your speech signal ends, with the steps of $\frac{1}{f_s}$ where f_s represents the sampling rate of your speech signal. After creating the time axis, to extract n_1 from your speech signal, create a plot —you do not need to provide this plot in your report— with this time axis defined and your speech signal. You should be adequately visualizing each number signal in your plot. Find the limits of one of the instances of n_1 in your speech signal, and note these limits. Using these limits and your sampling rate information, extract one of the instances where you say n_1 from your speech signal into another .flac file.

After completing these steps provided, you should have five different .flac files, where the first is your speech signal where you have recorded ID, as defined above, second is n_1 that you have extracted from your speech signal, and the other three are *TotalNumber*, n_1, and n_2, respectively obtained from the text-to-speech algorithm provided.

As you have created these signals, using the relation you have obtained in Part 4.1, and

using the **ConvFUNC** you have defined as it is, create the output sequence, denoted by $\psi[n]$, of the cross-correlation operation of n_1 (the signal you have extracted from your speech signal) and your whole speech signal. Using the **subplot** command in MATLAB, create two plots in the same figure window, where the first is your whole speech signal and the time array corresponding to this signal, and the second is n_1 and the time array corresponding to this signal. In another figure window, plot $|\psi[n]|$ and the time array corresponding to this signal. Repeat this by re-plotting the second figure with $|\psi[n]|^2$ and the time array corresponding to this signal and with $|\psi[n]|^4$ and the time array corresponding to this signal. What did you observe as you have changed $|\psi[n]|$ to $|\psi[n]|^2$ and $|\psi[n]|^4$? Were you able to detect each occurrence of n_1 in your whole speech signal? Why? If you were to record your voice saying n_1, what might have changed in $|\psi[n]|$?

Repeat the above paragraph by searching for n_1 that you have obtained from the text-to-speech algorithm in *TotalNumber* obtained from the text-to-speech algorithm (You do not need to answer the same questions of the above paragraph). Compare the result of the text-to-speech signal with the result you have obtained in the previous paragraph. If there exists a difference, what might have caused this difference? Instead of finding the cross-correlation operation, what would have changed in $|\psi[n]|$ if you had used **ConvFUNC** directly without using the relation you have obtained in Part 4.1? Support your argument by evaluating the plot for the case if you have used **ConvFUNC** directly instead of the cross-correlation operation. Search for n_2 that you have obtained from the text-to-speech algorithm in *TotalNumber* obtained from the text-to-speech algorithm. Does the result make sense?

After searching the specific n_1s indicated in the preceding paragraphs, search for n_1 that you obtained from the text-to-speech algorithm in your own speech signal and check for $|\psi[n]|^4$. Repeat this by searching for n_1 that you have obtained from your own speech signal in *TotalNumber* obtained from the text-to-speech algorithm. If there exists a difference, which of these searches were more successful? Why? What do you think would make the less successful method better? Indicate your comments.

Note: Unless the reverse is explicitly specified, indicate your plots within your report with specific properties as identified in the introductory paragraph, and specify your comments for the questions raised for each case in your report.

5 Part 5

In real-life applications, we cannot obtain pure signals; most of the time, we work with signals contaminated by noise or interference. Such contamination can result from many factors, such as the signals in the environment we are not interested in, interfering with the signal we are trying to observe. To reduce the effects of the noise, we use various methods, such as spectral filtering, averaging, etc. Although we have tools for coping with the noise, there exists a limit to the noise in our signals for our systems to function properly. For a concrete example, imagine you are trying to distinguish a word from a speech recording. If there is also some construction noise in the background of the recording, you may still be able to understand this word. However, as the construction

noise gets louder and louder, it will get more challenging for you to understand this word. Eventually, there will be a point where you will not be able to understand the word. You can think of this point as the limit of yourself (as a system that detects the words) at which you cannot function properly due to high amounts of noise. In this part, you will try to find this limit for the system you worked with in Part 4.

A common way of expressing the amount of noise in a signal is by using the signal-to-noise ratio (SNR). SNR is the ratio of the power of signal to the power of noise:

$$\text{SNR} = \frac{P_{\text{signal}}}{P_{\text{noise}}} \tag{4}$$

Here, by using this definition, you will report the effects of SNR and the system's limit by adding different amounts of noise to your signal.

5.1 Observing the Effects of SNR

First, load the audio signal of your ID that you generated with the text-to-speech algorithm (also referred to as $TotalNumber$) in Part 4.2 into an array called *audio_array*. Do not forget to load it with its sampling frequency. It is necessary to listen to the signal with the **soundsc** function, as you learned in the first lab assignment. Then, determine the length of the array holding this signal and name the variable containing this value as *audio_len*. This length variable will be needed to generate the noise signal and calculate the power of our signal. Listen to the audio with the **soundsc** function by providing it with the *audio_array* and sampling frequency variables.

Second, determine the power of this audio signal. For this application, we can use definition of the average power as follows,

$$P_{\text{signal}} = \frac{1}{N_x} \sum_{i=1}^{N_x} (x[i])^2 \tag{5}$$

where N_x is the number of samples in our sequence and $x[i]$s are the samples of our sequence. Store this value in a variable called *p_signal* and print its value.

The noise in our signals can be modeled with various distributions. Here, we will assume that the noise in our signal is additive white Gaussian noise (AWGN). We will generate this AWGN by using a normally distributed random number generator. To do this, first, you need to determine the power of our AWGN. For an SNR value of 10, calculate the power of noise using Eq.(4) and the *p_signal* you calculated above. Name this value *p_noise*, and print *p_noise*. Then, we can generate the AWGN with the following script in MATLAB:

```
rng(5)
awgn = sqrt(p_noise).*randn([audio_len, 1]);
```

Here, the first line sets a seed for the random number generator. Setting a seed provides that although the numbers generated by this generator are random, the values generated for you and your friend are the same as long as you use the same seed (seed is

set as 5 in this example). Also, the random-number generation here is not truly random but *pseudo-random*. If you are interested in learning more about random number generation and seeds, this link might be an adequate start. The second line here generates an array of normally distributed random samples with size *audio_len* by 1 and with a standard deviation of $\sqrt{p\_noise}$.

Finally, simply add the noise sequence, *awgn*, to your audio signal, *audio_array*, and equate the result of this summation to a variable named *noisy_audio*. Listen to this noisy audio signal. What do you hear? Repeat this process for the SNR values of 0.1 and 0.001. Listen to your noise-corrupted signals for these SNR values as well. How does the sound change? Can you hear the numbers? Print *p_noise* and report for each case.

5.2 Detecting the SNR Limit

In this part, you will be repeating the detection task you completed in Part 4.2 with the noise-corrupted signals for different SNR values. By doing this, you will identify the SNR value at which your system starts to fail to detect the numbers.

Load the audio signal of your ID, generated with the text-to-speech algorithm in Part 4.2 into an array called *audio_array*. Do not forget to get its sampling frequency. Also, load the audio file of a single number (defined as n_1 previously) that is generated with the text-to-speech algorithm as a variable called *filter*, similar to what you have done in Part 4.2.

Then, write a **for** loop that sweeps the SNR values from 0.01 to 0.001, decreasing the SNR by 0.001 at each step. Inside this **for** loop, you should repeat the steps that you have followed in Part 5.1 to generate a noise sequence and add it to your original audio signal and, again, call this noise-corrupted signal *noisy_audio*. Using the cross-correlation function you have derived and used in Part 4.2, you will try to detect the number in the *filter* signal in the *noisy_audio* signal. This operation should be performed in the **for** loop, as we are trying to find the SNR limit of this detection system. Call the result of the cross-correlation operation *output*. Then, using **subplot** function with five rows and two columns, plot these *output* variables for every SNR you are iterating through with this loop. As stated, you should plot each *output* within one figure using the **subplot** function. Do not forget to add a title to each subplot that indicates the SNR value for each *output*.

By observing the resulting plots, determine if your system can detect the number pronounced in the audio file of your *filter*. At what SNR value does your system start to fail?

6 Final Remarks

Throughout this assignment, you are **NOT** allowed to use symbolic operations in MATLAB. Submit the results of your own work in the form of a well-documented lab report on Moodle. Borrowing full or partial code from your peers or elsewhere is **NOT** allowed

and will be punished. The axes of all plots should be scaled and labeled. To modify the styles of the plots, add labels, and scale the plots, use only MATLAB commands; do **NOT** use the GUI of the figure windows. When your program is executed, the figures must appear exactly the same as you provide in your solution. You need to write your MATLAB codes not only correctly but efficiently as well. Please include all evidence (plots, screen dumps, MATLAB codes, MATLAB command window print-outs, etc.) as needed in your report. Append your MATLAB code at the end of your assignment as text, not as an image, and do **NOT** upload it separately. You can use the "Publish" menu of MATLAB to generate a PDF file from your codes and their outputs and append it to the end of your report. If you do this, please also indicate the part that the code corresponds to with a label. Typing your report instead of handwriting some parts will be better. If you decide to write some parts by hand, please use plain white paper. Please do not upload any photos/images of your report. Your complete report should be uploaded on Moodle as a single good-quality PDF file by the given deadline. Please try to upload several hours before the deadline to avoid last-minute problems that may cause you to miss the deadline. Please DO **NOT** submit files by e-mail or as hard copies.

Please name the PDF file you submit on Moodle as follows using only lower-case English characters for your first name, middle name (if any), and last name. Please use your full name as it appears on the Bilkent system.

LAB2_firstname_middle name_lastname.pdf
filename example for Ayşenur Çiğdem Sürücü:
LAB2_aysenur_cigdem_surucu.pdf