Indian Institute of Technology, Bombay
EE229 - Signal Processing 1
Computational Assignment 1

Topic - Virtual Reality with Convolution

Name - Krish Rajesh Vedak
Roll No.- 24B3904

Submission Type: Individual

Index

Drive Link:-

https://drive.google.com/drive/folders/16GXXJhHVjuftmhzfUpxUY3r6iXHCSnix?usp=drive_link

Assignment Understanding

Audio, like songs, has a different effect when heard in auditoriums, halls, etc. Also, the quality of audio heard depends on the location of the person in the hall. The best experience is normally observed when the person sits on the midline of the hall, close to the centre, depending on the hall's dimensions. However, we can't have 200 people sitting on that single best seat. Also, any hall has a limited capacity. So, naturally, a question arises: what can be done so that people who attend these concerts online/virtually can get the same experience as the person sitting in the best seat? This is what the assignment is trying to do.

The solution is to create an algorithm that modifies the "dry audio track" into one that is actually heard while sitting in the best seat. Previously, this was done by actually placing a recorder (for individual ears!) in the best seat, and the performers used to perform so that the best audio could be recorded, only to be distributed later to the public. Now, we try to do this using the basic idea of convolution.

The idea is convolution. Audio signals, like songs, are nothing but a series of impulses added together. Specifically, for a discrete signal, x[n], it is given by:

$$x[n] = \sum_{k=-\infty}^{k=\infty} x[k]\delta[n-k]$$

where $\delta[n]$ is the discrete unit impulse function. In our case, we'll have x[n] as the dry song track. This dry sound track now travels through the air, undergoes reflections, power losses, etc, interacts with our outer body, and inner anatomy and finally reaches the left ear and the right ear. Note that the sound reaching the left and right ears is different from when heard together gives us the best experience.

This can be viewed as a system, with the input being the dry sound, and the output being the left and right audio tracks. Here, we use the idea of the response of the Left Ear System [LES] and the Right Ear System [RES] to a unit impulse. Let the output for the LES be $h_{left}[n]$ and the output for the RES be $h_{right}[n]$.

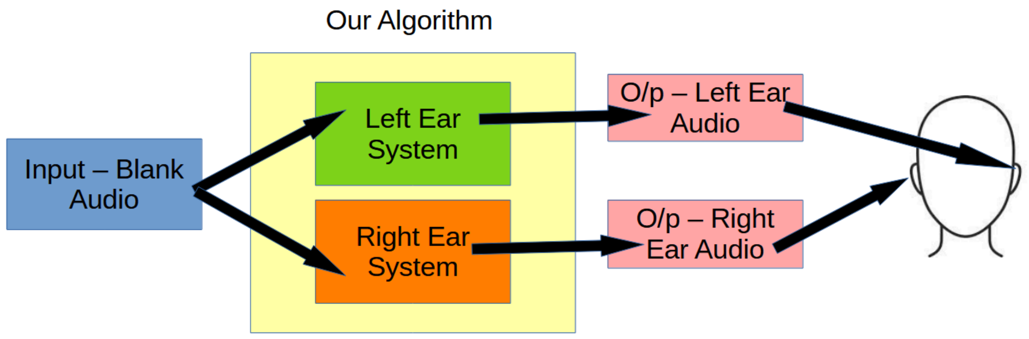


Fig 1. Block diagram of the Hall + Human Body system for Audio Listening in the Hall

Now, an important property comes to the rescue - LTI, i.e. linear time invariance of our audio track. Since our audio track is linear and time invariant, the output for the LES and RES, given the input being our dry audio, which is a summation of impulses, will be the summation of outputs for these individual impulses. Mathematically:

$$y_{left}[n] = \sum_{k=-\infty}^{k=\infty} x[k]h_{left}[n-k]$$

$$y_{right}[n] = \sum_{k=-\infty}^{k=\infty} x[k]h_{right}[n-k]$$

where $y_{left}[n]$ and $y_{right}[n]$ are the outputs of the LES and RES, respectively. The operation done above is called the convolution of x[n] and h[h] and is written as:

$$y_{right}[n] = x[k] * h_{right}[n]$$

$$y_{left}[n] = x[k] * h_{left}[n]$$

In practice, $h_{left}[n]$ and $h_{right}[n]$ are called the left and right channels. Both these channels are packaged together in a 2-channel audio file, which is called the stereo. This stereo file, which has specifically 2 channels for left and right ear impulse response, is called the BRIR - Binaural Room Impulse Response. Although BRIR contains the word “Room”, it also contains the interactions the audio has with the human body shape before we hear it.

We have been given the BRIR .wav file that contains the two channels. Our task is to carry out the convolution of the dry audio with these two channels individually. Another task is to solve the toy problem of implementing the convolution algorithm.

Toy Problem

The toy problem was to write code for the convolution given below:

$$x[n] * h[n]; x[n] = \alpha^n u[n]; h[n] = u[n]; \alpha = 0.4$$

u[n] is the unit step function given by:

$$u[n] = \begin{cases} 1 & \text{if } n \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

We have to evaluate the convolution for n = 1,2,3…, 10.
The Plot for x[n] for the above n is as follows:

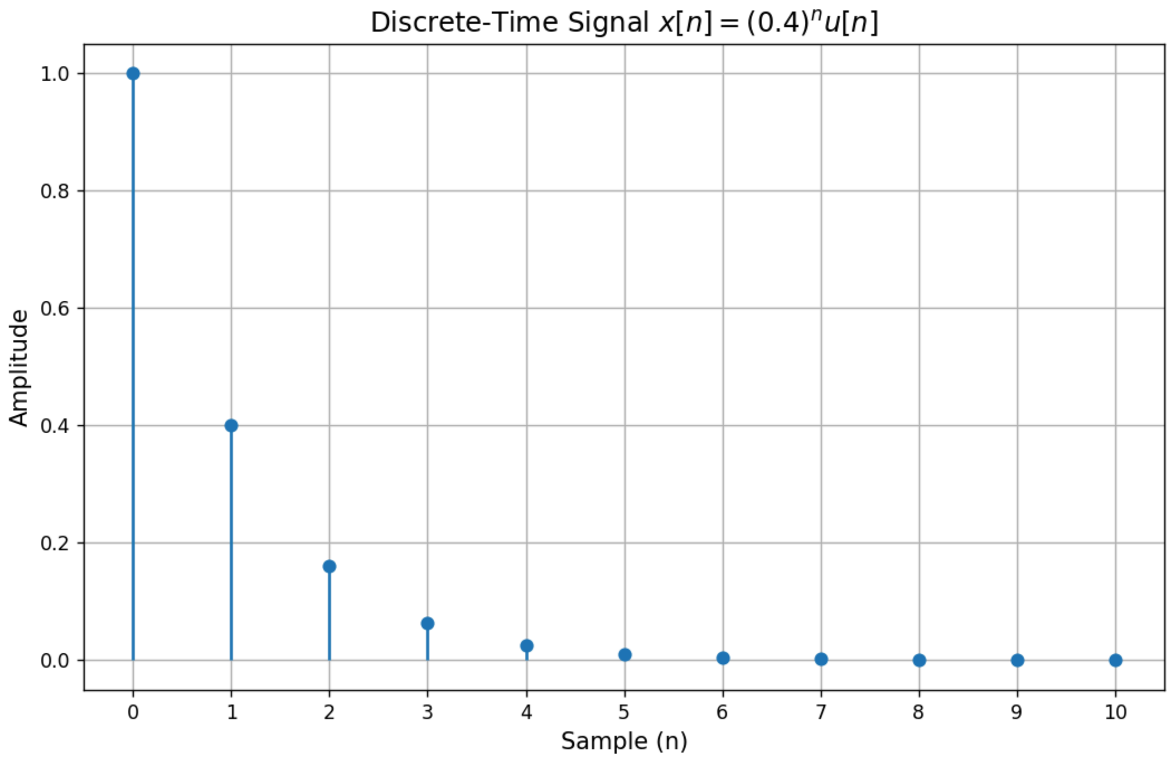


Fig 2. Discrete Time signal x[n]

The convolution for n = 5 is shown below graphically:

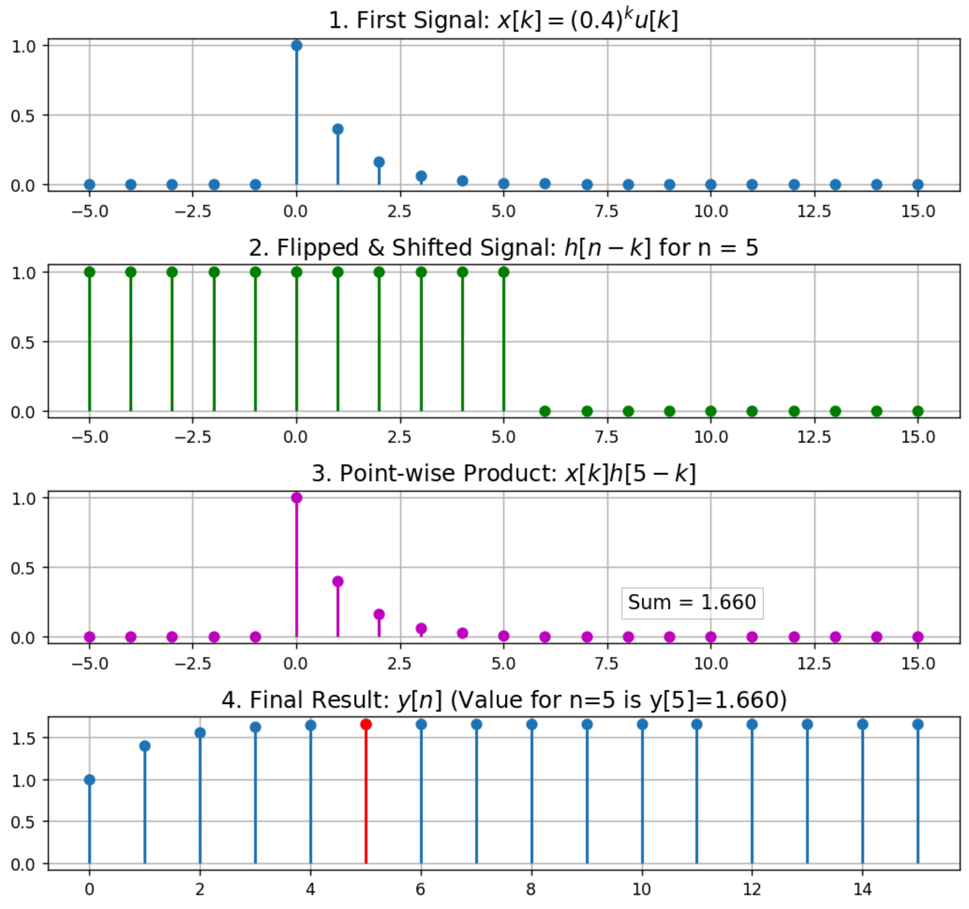


Fig 3. Graphical Convolution for n = 5

Using Python, the convolution results are as follows:

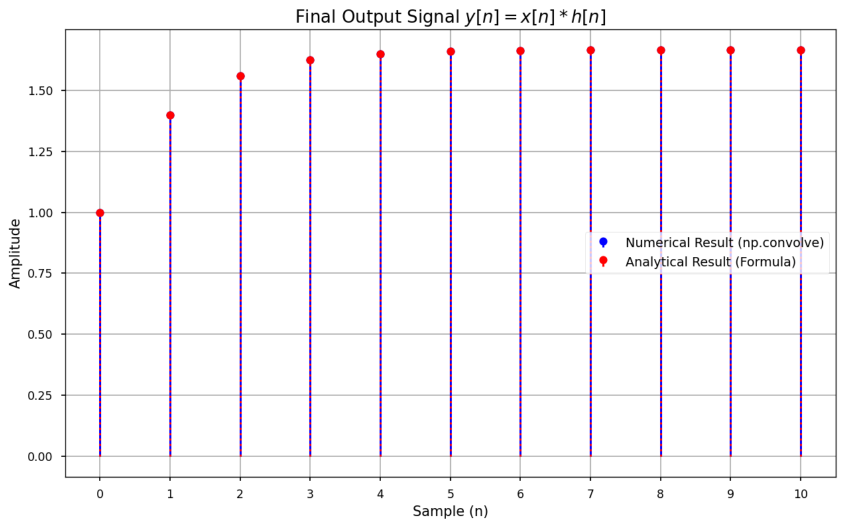
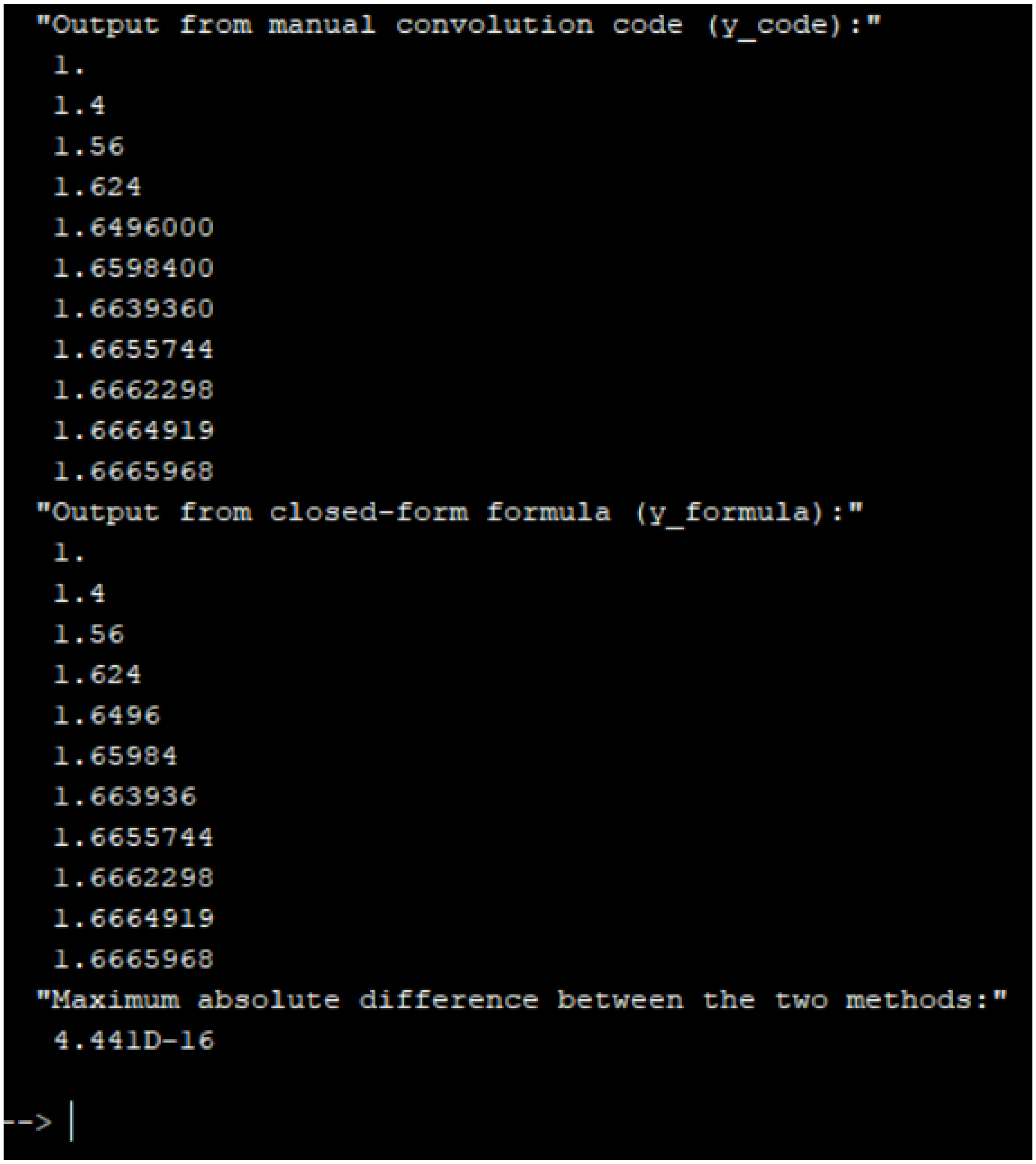


Fig 4. Convoluted signal y[n]

The maximum difference between numerical and analytical results comes out to be approximately 4.4408e-16. This suggests that the convolution was successfully carried out numerically and is identical to the result generated using the built-in method. The numerical values for y[n] were verified by carrying out the convolution in Scilab. The results are as follows:



```
 "Output from manual convolution code (y_code):"
  1.
  1.4
  1.56
  1.624
  1.6496000
  1.6598400
  1.6639360
  1.6655744
  1.6662298
  1.6664919
  1.6665968
 "Output from closed-form formula (y_formula):"
  1.
  1.4
  1.56
  1.624
  1.6496
  1.65984
  1.663936
  1.6655744
  1.6662298
  1.6664919
  1.6665968
 "Maximum absolute difference between the two methods:"
  4.441D-16

--> |
```

Fig 5. Scilab Result for Toy Problem Convolution

The Scilab results also give almost the same maximum absolute difference, suggesting the convolution was correctly carried out by manual summation and for loops.

BRIR Problem

This was solved by creating two separate .sce files - one for manual convolution implementation, and the other for using the built-in method. Both files are submitted. We have carried out the convolution for all the BRIR files.

It is important to note that the convoluted signal is almost identical to the dry signal - it is difficult to pick up the minute differences. So, it is important to be sure that the convolution was actually successful. To illustrate this, we present the convoluted left side signal and compare it with the dry signal for the Five_Columns_Long_16k impulse response.

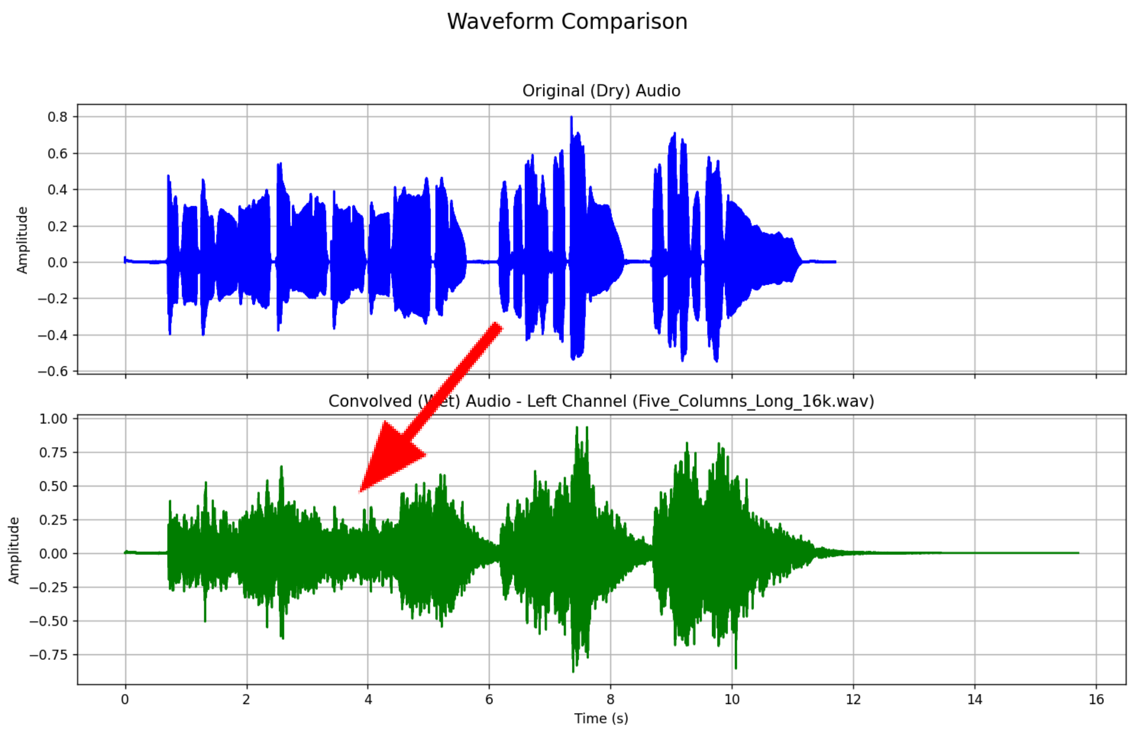


Fig 6. Original and Left Channel Output for Five_Columns_Long_16k.wav

The red arrow indicates the difference. If we observe the region indicated by red arrow, the green plot is actually obtained if we flip the blue plot about the time axis. This is an interesting phenomenon. Why has the flip occurred? Or in other words, why did the phase change? The plot given below shows us the first 0.01 seconds of the corresponding BRIR file.

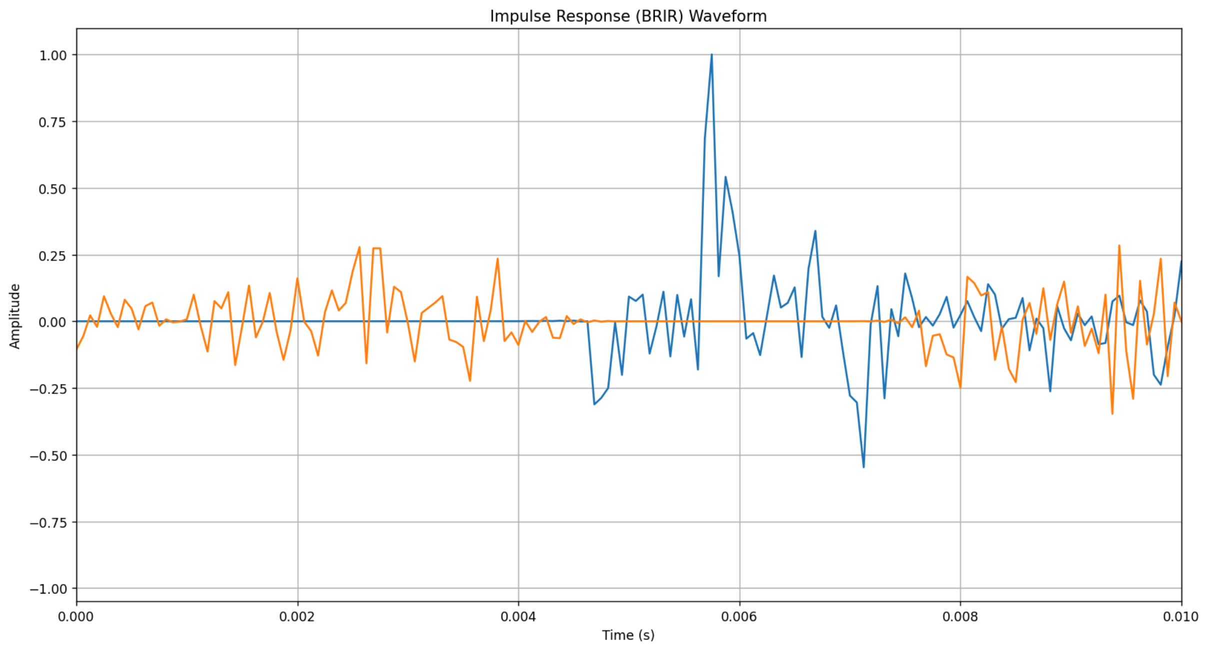


Fig 7. First 0.01 Seconds of the Five_Columns_Long_16k.wav file

The orange and the blue waveforms indicate either the left or the right channels. The point, however, is to note that the first spike for both plots is -ve. The characteristics of the output signal are determined entirely by the impulse response. The reason the output is phase-inverted is that **the BRIR file itself is phase-inverted.**

An impulse response is typically captured by recording a sharp, positive sound pressure event (like a clap). Its waveform should start with a sharp peak going upwards (positive). In this case, the BRIR file likely starts with its main, initial peak going downwards (negative).

When the convolution is calculated, every sample of your original audio is multiplied by the samples of this inverted impulse response. Multiplying by these negative values effectively flips the polarity of the entire original signal.

This can be verified for a second BRIR file - parking_garage_16k.wav.

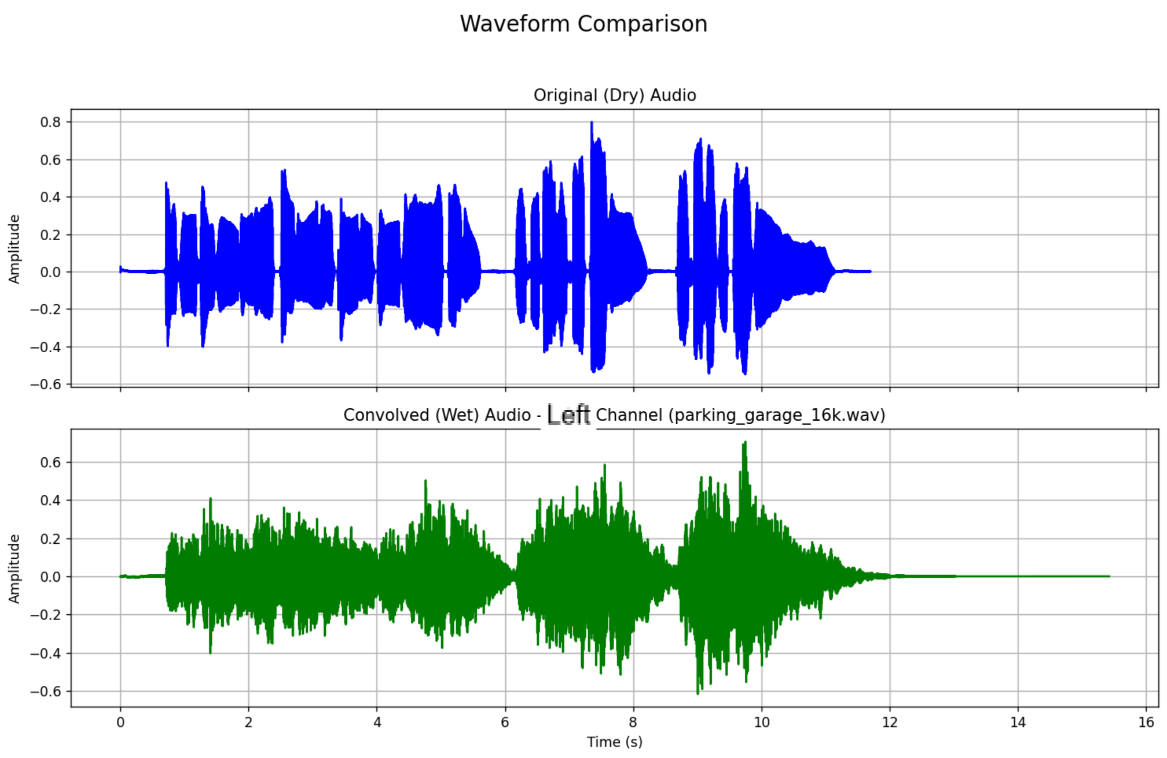


Fig 8. Original and Left Channel Output for parking_garage_16k.wav

As you can see, this time, the plots appear to be in phase. The first 0.001s of the corresponding BRIR file are as follows:

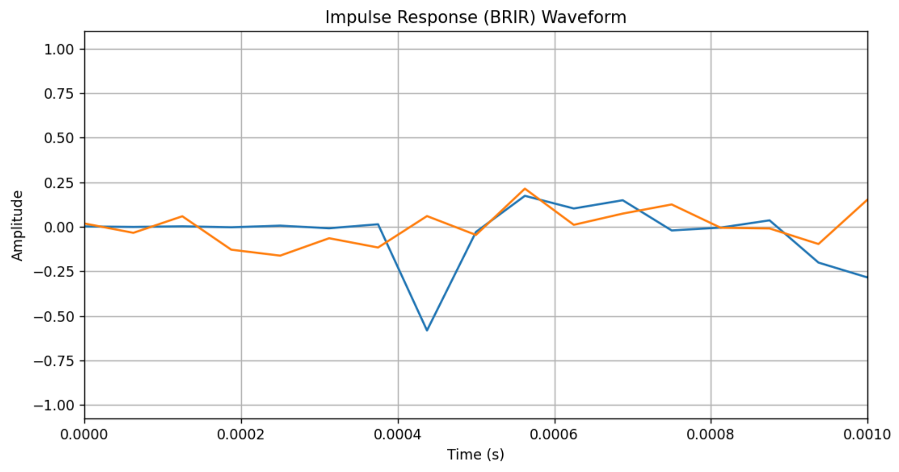


Fig 9. First 0.001 seconds of parking_garage_16k.wav

We suspect the orange waveform corresponds to the left channel since its first peak is +ve. Let's confirm this by comparing the dry sound with the right channel.

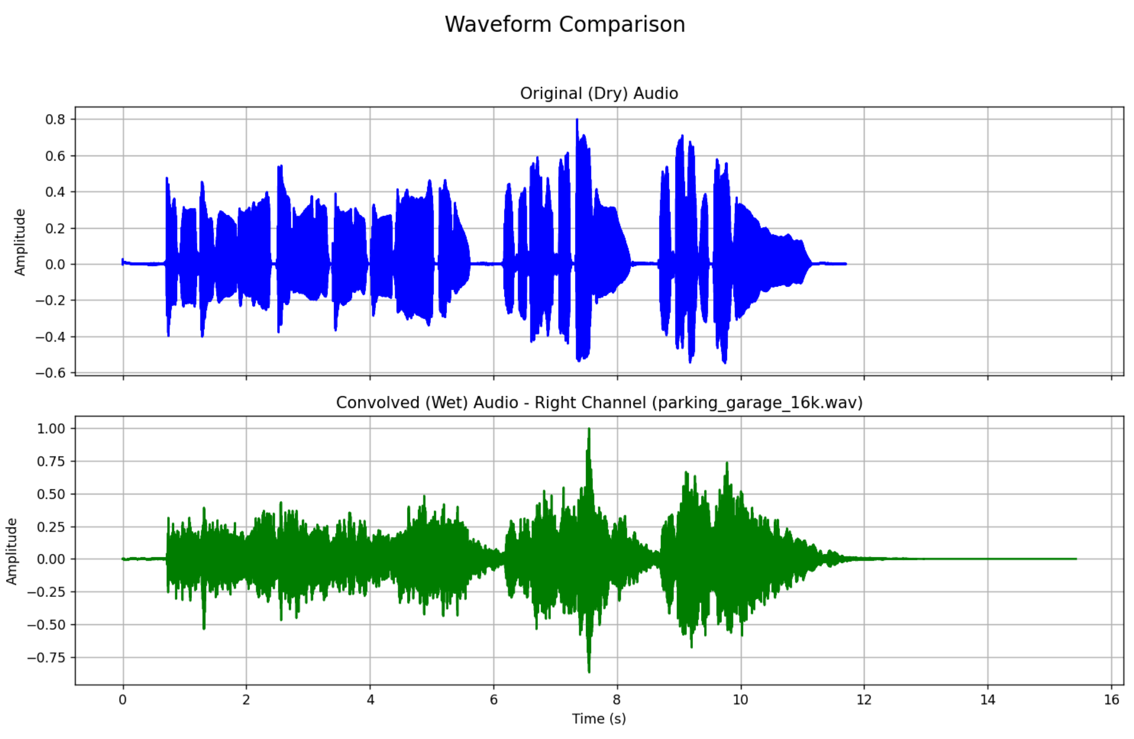


Fig 10. Original and Right Channel Output for parking_garage_16k.wav

As we can see, after choosing the right channel, the plot is inverted - i.e. the blue waveform in the BRIR plot corresponds to the right channel.

We now explore what changes the convolution actually did. This can be seen by calculating the difference between the dry sound and the convoluted sound. The main point is that the convoluted signals are slightly longer, due to the extra trails that are added to mimic the hall or the environment. This is compensated for by adding a padding of zero amplitude sound to the dry sound to make them of equal length. The plot obtained for the BRIR file - parking_garage_16k.wav. is as follows:

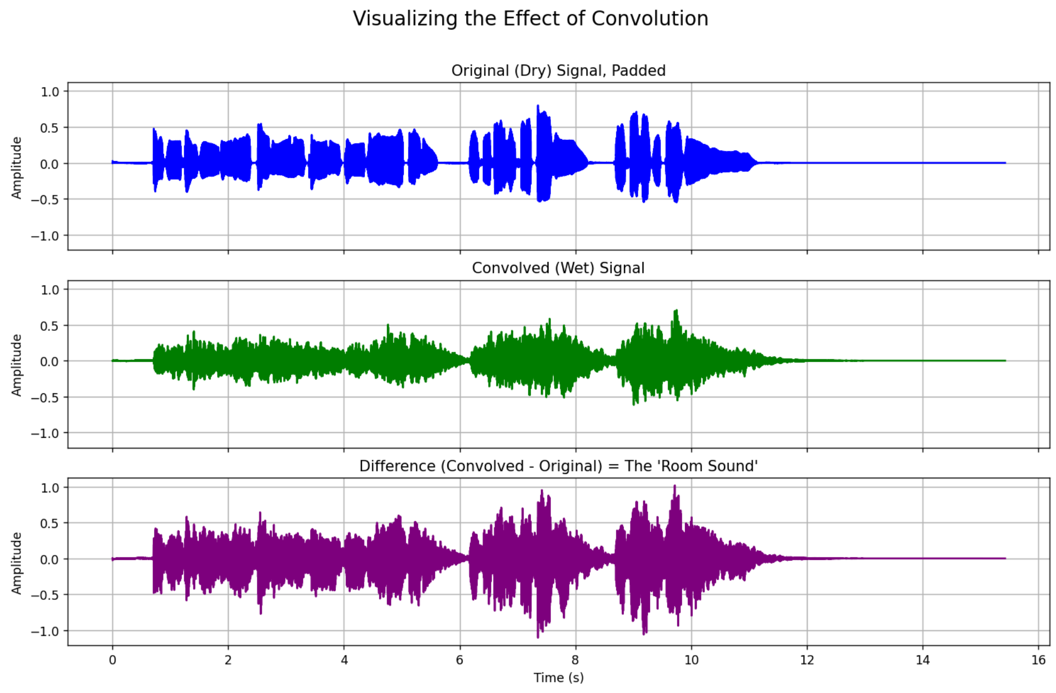


Fig 11. What eactly did convolution do?

Clearly, the purple plot is non-zero. This is what the convolution changed. We can also see that the convolved - left channel amplitude is slightly less than the dry sound, and this makes sense, because parking/garage regions are extremely wide, long and lower in height.

Conclusion

Hence, we have successfully implemented the convolution algorithms on Python and Scilab, carried out the convolution for the dry sound BheegiRegular.wav for all the given BRIR files, analysed the outcomes and visualised the changes convolution exactly did.

Learnings

This assignment was a great learning experience. It introduced me to Scilab and Python's capabilities of handling audio files and convolution. I also learnt about audio file types, audio channels - mono and stereo, what BRIR exactly is, reverberations and how audio signals were then recorded for the faraway audience so that they could get the best experience. I also learnt about the influence of phases on convolution and the fascinating capability of the human ear to not notice changes in absolute phase, making the inverted phase and the original phase sounds appear almost identical to the human brain.

Code Snippets

generate_reverb_data.sce

```
clear; close; clc;

//input wav file
[inp, fs_inp] = wavread("D:/LEVEL UP/IITB Info Related Docs/ACADEMICS/YEAR 2/SEM-3/EE229 -
Signal Processing 1/Assignments/CA1/CA1_wav_sce/BheegiRegular.wav");
[rir, fs_rir] = wavread("D:/LEVEL UP/IITB Info Related Docs/ACADEMICS/YEAR 2/SEM-3/EE229 - Signal
Processing 1/Assignments/CA1/CA1_wav_sce/long_echo_hall_16k.wav");

//SINGLE CHANNEL CONVOLUTION FUNCTION
function [output]=fun_conv(inp1, rir1)
  //PASTE YOUR CODE FOR CONVOLUTION OF TWO SIGNALS
  disp(">>> EXECUTING THE MANUAL FOR-LOOP FUNCTION NOW <<<");
  N = length(inp1);
  M = length(rir1);
  output_len = N + M - 1;
  output = zeros(1, output_len);


  for i = 1:N
    for j = 1:M
      output(i + j - 1) = output(i + j - 1) + inp1(i) * rir1(j);
      disp(i);disp(j);
    end
  end
  //output = conv(inp1,rir1);
endfunction

//obtain RIR for left channel
rir_left = rir(1,:);
//obtain RIR for right channel
rir_right = rir(2,:);

//obtain convolved signal for left channel
out_left = fun_conv(inp,rir_left);
//obtain convolved signal for right channel
out_right = fun_conv(inp,rir_right);

//obtaining stereo sound by combaining two channels
out = [out_left;out_right];
out = out/max(abs(out));

//playing convolved signal
playsnd(out,fs_inp);

//writing convolved signal
wavwrite(out,fs_inp,"D:/LEVEL UP/")
```

toy_problem.sce

```
// Clear console and variables
clc;
clear;

// --- Part 1: Define Signals ---
// Given parameters
alpha = 0.4;
n_vec = 0:10; // Time vector for n = 0, 1, ..., 10

// Define x[n] = alpha^n * u[n]
// Since n >= 0, u[n] is 1, so x[n] = alpha^n
x = alpha.^n_vec;

// Define h[n] = u[n]
// Since n >= 0, u[n] is 1
h = ones(1, length(n_vec));

// --- Part 2: Manual Convolution Code ---
// Initialize the output vector y with zeros
y_code = zeros(1, length(n_vec));

// Outer loop iterates through each output sample 'n'
for n = 0:10
  // Initialize the sum for the current y[n]
  y_n_sum = 0;

  // Inner loop calculates the sum for k from 0 to n
  // y[n] = sum_{k=0 to n} x[k]h[n-k]
  for k = 0:n
    // Scilab uses 1-based indexing, so we use k+1 and (n-k)+1
    term = x(k + 1) * h(n - k + 1);
    y_n_sum = y_n_sum + term;
  end

  // Store the final sum in our output vector y
  // Again, use n+1 for the index
  y_code(n + 1) = y_n_sum;
end

disp("Output from manual convolution code (y_code):");
disp(y_code'); // Transpose (') for a clean column display

// --- Part 3: Compute using Closed-Form Expression and Verify ---
// y[n] = (1 - alpha^(n+1)) / (1 - alpha)
y_formula = (1 - alpha.^(n_vec + 1)) / (1 - alpha);

disp("Output from closed-form formula (y_formula):");
disp(y_formula'); // Transpose for clean display

// --- Verification ---
// Calculate the difference between the two methods
difference = y_code - y_formula;

disp("Maximum absolute difference between the two methods:");
disp(max(abs(difference))); // Should be 0 or a very small floating point error
```

toy.py

```
import numpy as np
import matplotlib.pyplot as plt

# --- 1. Define Parameters and Signals ---
alpha = 0.4
# Define the time vector for n = 0, 1, ..., 10
n = np.arange(11)

# Generate the signal x[n] = alpha^n * u[n]
# Since n is non-negative, u[n] = 1
x_n = alpha**n

# Generate the signal h[n] = u[n]
# Since n is non-negative, h[n] = 1
h_n = np.ones_like(n)

# --- 2. Calculate the Output y[n] in Two Ways ---

# Method A: Numerical Convolution
# Use NumPy's convolve function. We only need the first 11 points of the
output.
y_numerical = np.convolve(x_n, h_n)[:len(n)]

# Method B: Closed-Form Analytical Solution
# From the geometric series formula, y[n] = (1 - alpha^(n+1)) / (1 -
alpha)
y_analytical = (1 - alpha**(n + 1)) / (1 - alpha)


# --- 3. Graphically Verify the Results ---
print("Verification:")
print(f"Max difference between numerical and analytical results:
{np.max(np.abs(y_numerical - y_analytical))}")

# Create a plot to display the results
plt.figure(figsize=(12, 7))
plt.style.use('seaborn-v0_8-notebook')

# Plot the numerical result with blue filled circles
```

```
plt.stem(n, y_numerical, linefmt='b-', markerfmt='bo', basefmt=" ",
label='Numerical Result (np.convolve)')

# Plot the analytical result with red open circles on top to show they
match
plt.stem(n, y_analytical, linefmt='r:', markerfmt='ro', basefmt=" ",
label='Analytical Result (Formula)')

# Add plot titles, labels, and legend for clarity
plt.title(r'Final Output Signal $y[n] = x[n] * h[n]$', fontsize=15)
plt.xlabel('Sample (n)', fontsize=12)
plt.ylabel('Amplitude', fontsize=12)
plt.xticks(n)
plt.legend(fontsize=11)
plt.grid(True)

# Show the plot
plt.show()
```

***