Full Name: Orkun İbrahim Kök

Department: EEE

Course Code: EEE321

Section: 02

Experiment Number: 06

Date: 12.05.2024

Lab 6 Report

Introduction:

In this lab experiment, Discrete-Time Fourier Transform (DTFT), Inverse Discrete-Time Fourier Transform (IDTFT), and convolution operations will be used on different type of filters.

Analysis:

Part 1

Derivations for this part can be found at the end of this report.

Part 2.1 Implementation of DTFT and IDTFT

In this part, two MATLAB codes were written for DTFT and IDTFT without using the built-in functions. The norm square difference is calculated as 31.5625, which is nonzero. The reason may be while running DTFT and IDFTF codes, if these functions do not cover the entire frequency range, norm square difference can be a nonzero value. Codes can be seen at Appendix.

>> part2_test

Norm square difference (E): 31.5625

Part 2.2 Implementation of Convolution

In this part, a code is written which calculates the convolution using derived DTFT and IDTFT operations. Code can be seen in Appendix.

Part 2.3 Another Approach for Convolution

In this part, again convolution is calculated. However, this time convolıtion result is achieved by matrix multiplication. Code can be seen in Appendix.

Part 2.4 Testing the Convolution Function

In this part, functions written in parts 2.2 and 2.3 are tested, whether they give the same result with built-in conv function in MATLAB or not. Also, elapsed time for each function is calculated. Code can be seen in Appendix.

Built-in conv() output: 2 8 16 22 25 22 16 13 10 7 4 1 -1 -1

ConvFUNC output: 2 8 16 22 25 22 16 13 10 7 4 1 -1 -1

ConvFUNC_M output: 2 8 16 22 25 22 16 13 10 7 4 1 -1 -1

Error between ConvFUNC and conv(): 2.9286e-29

Error between ConvFUNC_M and conv(): 0

Error between ConvFUNC and ConvFUNC_M: 2.9286e-29

Time for built-in conv(): 0.0005162 seconds

Time for ConvFUNC: 0.0003299 seconds

Time for ConvFUNC_M: 0.0004611 seconds

Part 3

In this part, two different types of moving average filters to be tested on an anechoic horn recording with additive white noise. hGMAV decreases the background noise. hSMAV decreases the sound of the recording. Code can be seen in Appendix.

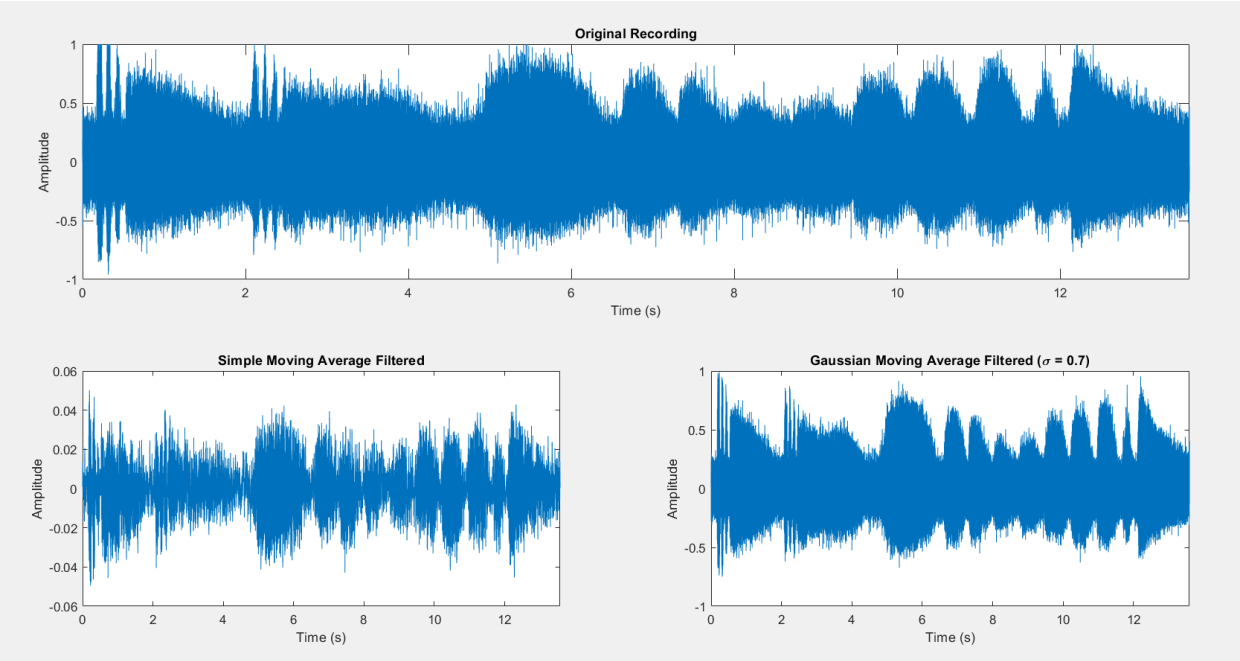

Fig. 1: Plot of the recording and applied filters

Part 4

In this part, a basic equalizer is designed that can attenuate and amplify certain frequencies using basic ideal filter arguments. Figure 2 shows the plots of bassoon, cello, flute, and trumpet frequency spectrum.

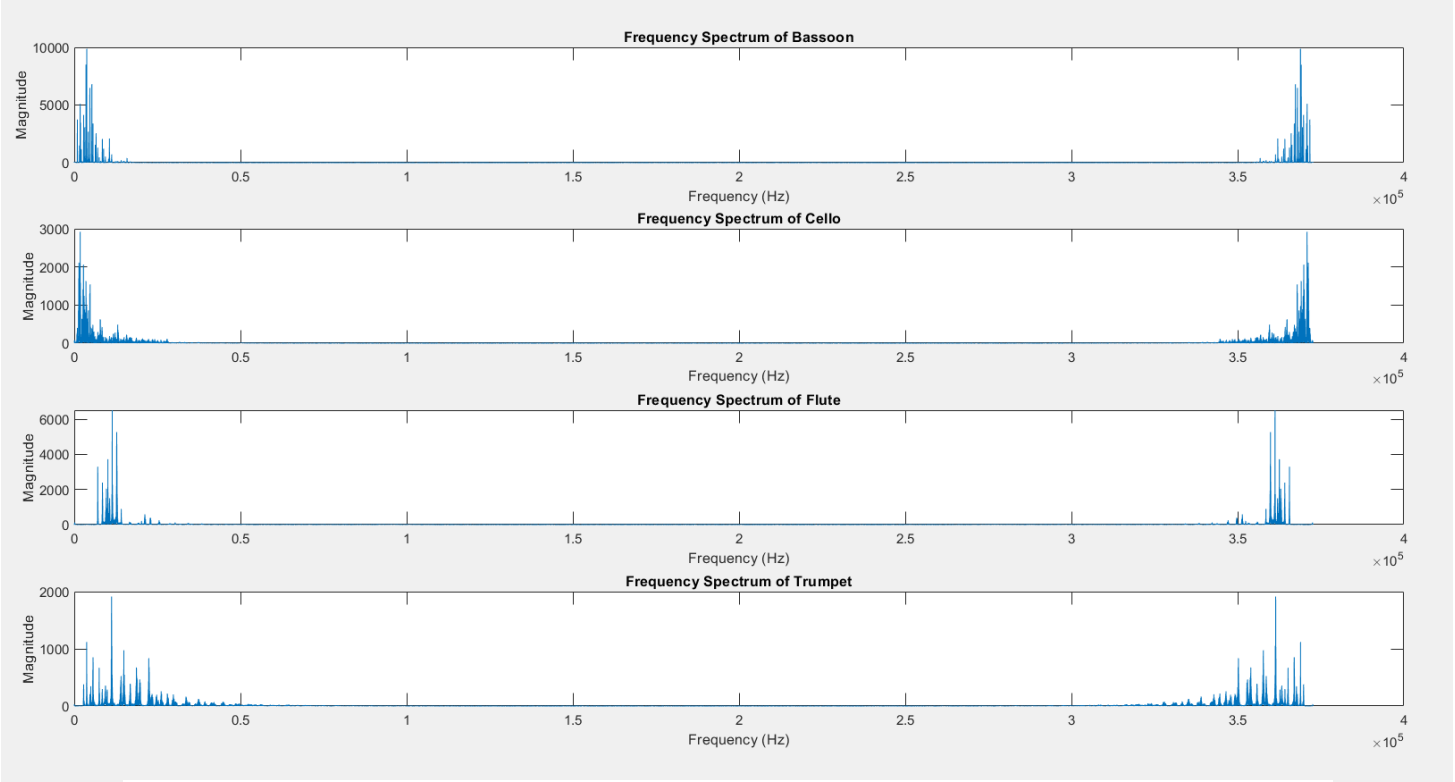

Fig. 2: Frequency spectrum

From the frequency spectrum plots, flute and trumpet are high frequnecy instruments, and bassoon and cello instruments are low frequency instruments.

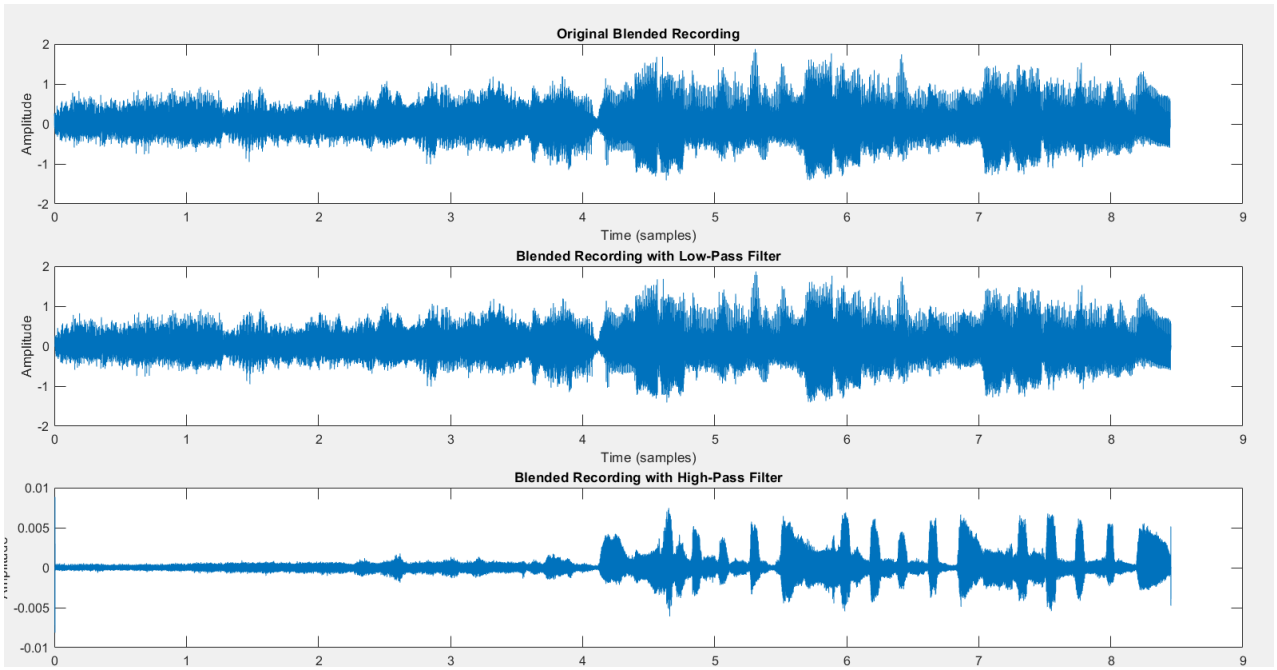

Fig. 3: Original recording, LPF, and HPF plots

As seen from Figure 3, Low-pass filter is nearly the same with original blended recording, however, high-pass filter is different. In high pass blended record, volume is significantly reduced compared to both original blended recording and low-pass filter blended recording. Also, in high-pass filter, cello and bassoon are hard to hear insturments.

Appendix

Part 2.1

```
function [X] = DTFT(x, n)
    X=sum(x.*exp((-1j*2*pi/n*(0:n-1)).*(0:n-1)'),2);
end

function [x] = IDTFT(X, n)
    x = sum(X.*exp((1j*2*pi/n*(0:n-1)).*(0:n-1)'),2) / (4);
end

% Define a simple finite-length signal
n = 0:4; % Time indices
phi1 = ones(size(n)); % Signal phi1[n] = 1 for 0 <= n < 5

% Define the frequency array for the DTFT
w = linspace(-pi, pi, 1000); % Frequency variable from -pi to pi

% Compute the DTFT of the signal
X_phi1 = DTFT(phi1, length(phi1));

% Compute the IDTFT to recover the time-domain signal
phi2 = IDTFT(X_phi1, length(X_phi1));

% Calculate the norm square difference between phi1 and phi2
E = sum(abs(phi1 - phi2).^2);

% Display the error
disp(['Norm square difference (E): ', num2str(E(1))]);
```

Part 2.2

```
function [y] = ConvFUNC(x, h, nx, nh, ny)
    y = IDTFT(DTFT(x, nx) .* DTFT(h, nh), ny);
end
```

Part 2.3

```
function y = ConvFUNC_M(x, h)
    t = x.*h(:);
    y = sum(spdiags(rot90(t)));
end
```

Part 2.4

```
clear;
close all;

x1 = [2, 4, 6, 8, 7, 6, 5, 4, 3, 2, 1];
x2 = [1, 2, 1, -1];
x11 = [x1 zeros(1,3)];
x22 = [x2 zeros(1,10)];
```

```
tic;
y3 = conv(x1, x2);
t3 = toc;

tic;
y1 = ConvFUNC(x11, x22);
t1 = toc;


tic;
y2 = ConvFUNC_M(x1, x2);
t2 = toc;

disp(['Built-in conv() output: ', num2str(y3)]);
disp(['ConvFUNC output: ', num2str(y1)]);
disp(['ConvFUNC_M output: ', num2str(y2)]);

e1 = sum((y1 - y3).^2);
e2 = sum((y2 - y3).^2);
e3 = sum((y1 - y2).^2);

disp(['Error between ConvFUNC and conv(): ', num2str(e1)]);
disp(['Error between ConvFUNC_M and conv(): ', num2str(e2)]);
disp(['Error between ConvFUNC and ConvFUNC_M: ', num2str(e3)]);


disp(['Time for built-in conv(): ', num2str(t3), ' seconds']);
disp(['Time for ConvFUNC: ', num2str(t1), ' seconds']);
disp(['Time for ConvFUNC_M: ', num2str(t2), ' seconds']);
```

Part 3

```
 [data, Fs] = audioread('Part3_recording.flac');

N = round(0.01 * Fs);

hSMAV = ones(1, N) / N;

sigma = 0.7;
mu = 0;
window = -5:5;
hGMAV = exp(-(window - mu).^2 / (2 * sigma^2)) / (sigma * sqrt(2 * pi));
hGMAV = hGMAV / sum(hGMAV);
 % Ensure the length of hGMAV is N
    if length(hGMAV) < N
        % Zero-pad hGMAV if its length is less than N
        hGMAV = [zeros(1, floor((N - length(hGMAV)) / 2)), hGMAV, zeros(1, ceil((N
- length(hGMAV)) / 2))];
    elseif length(hGMAV) > N
        % Truncate hGMAV if its length is greater than N
        hGMAV = hGMAV(1:N);
    end


filteredSMA = ConvFUNC_M(data, hSMAV);
filteredGMA = ConvFUNC_M(data, hGMAV);

time = (0:length(data)-1) / Fs;
```

```
figure;
subplot(2,2,[1,2]);
plot(time, data);
title('Original Recording');
xlabel('Time (s)');
ylabel('Amplitude');
xlim([0, max(time)]);

subplot(223);
plot(time, filteredSMA);
title('Simple Moving Average Filtered');
xlabel('Time (s)');
ylabel('Amplitude');
xlim([0, max(time)]);

subplot(224);
plot(time, filteredGMA);
title(['Gaussian Moving Average Filtered (\sigma = ', num2str(sigma), ')']);
xlabel('Time (s)');
ylabel('Amplitude');
xlim([0, max(time)]);

disp('Playing original recording...');
sound(data, Fs);
pause(length(data)/Fs + 2);

disp('Playing SMA filtered recording...');
sound(filteredSMA, Fs);
pause(length(filteredSMA)/Fs);

disp('Playing GMA filtered recording (\sigma = 0.7)...');
sound(filteredGMA, Fs);
pause(length(filteredGMA)/Fs);
```

Part 4

```
clear;
close all;

[bassoon, Fs] = audioread('bassoon.flac');
[cello, ~] = audioread('cello.flac');
[flute, ~] = audioread('flute.flac');
[trumpet, ~] = audioread('trumpet.flac');


f = linspace(0, Fs/2); % Frequency vector

Y_bassoon = DTFT(bassoon/Fs, length(bassoon/Fs));
Y_cello = DTFT(cello/Fs, length(cello/Fs));
Y_flute = DTFT(flute/Fs, length(flute/Fs));
Y_trumpet = DTFT(trumpet/Fs, length(trumpet/Fs));

figure;
subplot(4,1,1);
plot(abs(Y_bassoon));
title('Frequency Spectrum of Bassoon');
```

```
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(4,1,2);
plot(abs(Y_cello));
title('Frequency Spectrum of Cello');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(4,1,3);
plot(abs(Y_flute));
title('Frequency Spectrum of Flute');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(4,1,4);
plot(abs(Y_trumpet));
title('Frequency Spectrum of Trumpet');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


N = round(0.01 * Fs);
sigmaLPF = 0.4;
sigmaHPF = 0.02;
range = -floor(N/2):floor(N/2);

hLPF = exp(-0.5 * (range / sigmaLPF).^2);
hLPF = hLPF / sum(hLPF); % Normalize
hHPF = -hLPF;
hHPF(floor(length(hHPF)/2)+1) = hHPF(floor(length(hHPF)/2)+1) + 1;


blended = (bassoon + cello + flute + trumpet);

blendedLPF = ConvFUNC_M(blended, hLPF);
blendedHPF = ConvFUNC_M(blended, hHPF);

time1 = (0:length(blended)-1) / Fs;
figure;
subplot(3,1,1);
plot(time1, blended);
title('Original Blended Recording');
xlabel('Time (samples)');
ylabel('Amplitude');

time2 = (0:length(blendedLPF)-1) / Fs;
subplot(3,1,2);
plot(time2, blendedLPF);
title('Blended Recording with Low-Pass Filter');
xlabel('Time (samples)');
ylabel('Amplitude');

time3 = (0:length(blendedHPF)-1) / Fs;
subplot(3,1,3);
plot(time3, blendedHPF);
title('Blended Recording with High-Pass Filter');
xlabel('Time (samples)');
ylabel('Amplitude');
```

```
disp('Playing original blended recording...');
sound(blended, Fs);
pause(length(blended)/Fs + 2);

disp('Playing blended recording with Low-Pass Filter...');
sound(blendedLPF, Fs);
pause(length(blendedLPF)/Fs + 2);

disp('Playing blended recording with High-Pass Filter...');
sound(blendedHPF, Fs);
```

Orhun Ibrahim Kılıç

Part 1

a) $\bar{x}_{N\times 1} = \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[n-1] \end{bmatrix}$

Fourier matrix:

$$\bar{F}_{N\times N} = \begin{bmatrix} F_{0,0} & F_{0,1} & F_{0,2} & \cdots & F_{0,N-1} \\ \vdots & & & & \\ F_{N-1,0} & \cdots & & & F_{N-1,N-1} \end{bmatrix} \quad F_{k,n} = e^{-j\frac{2\pi kn}{N}}$$

$k, n = 0, 1, 2, \cdots, N-1$

matrix multiplication:

$X = F_x \Rightarrow X$ is the resulting frequency domain vector

matrix rep. of IDTFT:

→ Inverse Fourier Transform:

$$\bar{F}_{n,k} = \frac{1}{N} e^{j\frac{2\pi kn}{N}} \quad \text{for} \quad k, n = 0, 1, 2, \cdots, N-1$$

$$X = \bar{F}x$$

---

b) $x_{N\times 1} = \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix}$ $h_{m\times 1} = \begin{bmatrix} h[0] \\ h[1] \\ \vdots \\ h[m-1] \end{bmatrix}$ $\Rightarrow$ resulting convolution length = $\boxed{N+M-1}$

$$T = \begin{bmatrix} h[0] & 0 & 0 & \cdots & 0 \\ h[1] & h[0] & 0 & \cdots & 0 \\ \vdots & \vdots & & & \vdots \\ h[m-1] & h[m-2] & 0 & \cdots & h[0] \\ 0 & h[m-1] & 0 & \cdots & h[1] \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & h[m-1] \end{bmatrix}_{(N+M-1)\times N}$$

Each row corresponds to a shifted version of $h[k]$

$$y = Tx$$

c) define two sequences $x[n]$ and $h[n]$

$$X(e^{jw}) = \sum_{n=-\infty}^{\infty} x[n] e^{-jwn}$$

$$H(e^{jw}) = \sum_{n=-\infty}^{\infty} h[n] e^{-jwn}$$

time domain convolution:

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

DTFT convolution:

$$Y(e^{jw}) = \sum_{n=-\infty}^{\infty} \left(\sum_{k=-\infty}^{\infty} x[k]\, h[n-k] \right) e^{-jwn}$$

$$= \sum_{k=-\infty}^{\infty} x[k] \sum_{n=-\infty}^{\infty} h[n-k]\, e^{-jwn}$$

$$= \left(\sum_{n-k=-\infty}^{\infty} h[n-k]\, e^{-jw(n-k)} \right) \left(\sum_{k=-\infty}^{\infty} x[k]\, e^{-jwk} \right)$$

$$\Rightarrow Y(e^{jw}) = X(e^{jw}) \cdot H(e^{jw})$$

$$\Rightarrow y[n] = \text{IDTFT}\{ Y(e^{jw}) \}$$

$$y[n] = \text{IDTFT}\{ X(e^{jw}) H(e^{jw}) \}$$