Furkan Büyüksarıkulak

22002097

16.04.2025

EEE 321

Signals and Systems

Spring 2024-2025

Lab Assignment 6

In this lab task we observed the methods to compute discrete time Fourier transform, iverse discrete time Fourier transform and convolution. We explore three distict methods for computing dt convolution of sequences. Then we designed some filters using these transformation algorithms that we had and tested them on noisy sounds.

Part 1

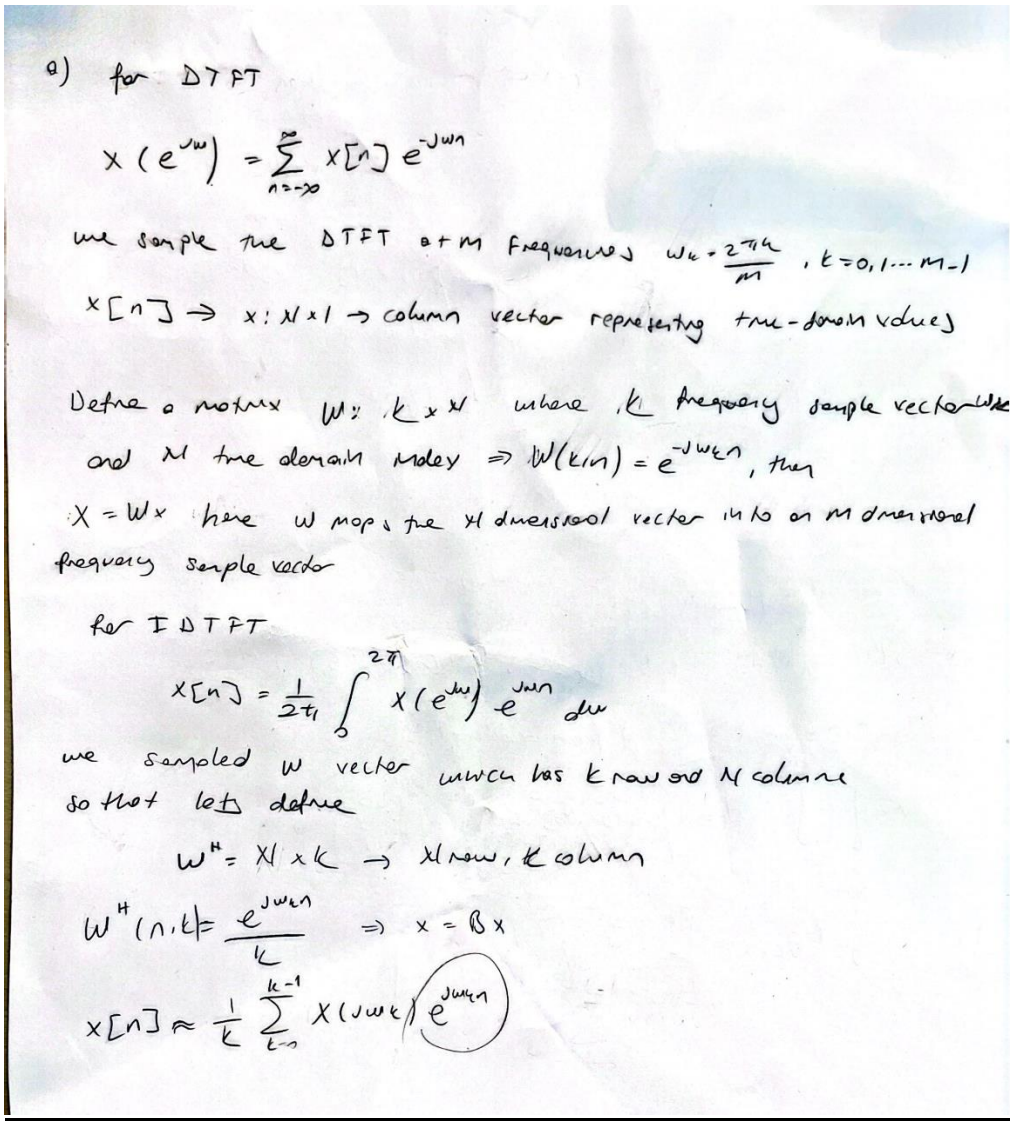

figure 1 : Part 1 paper solution (a)

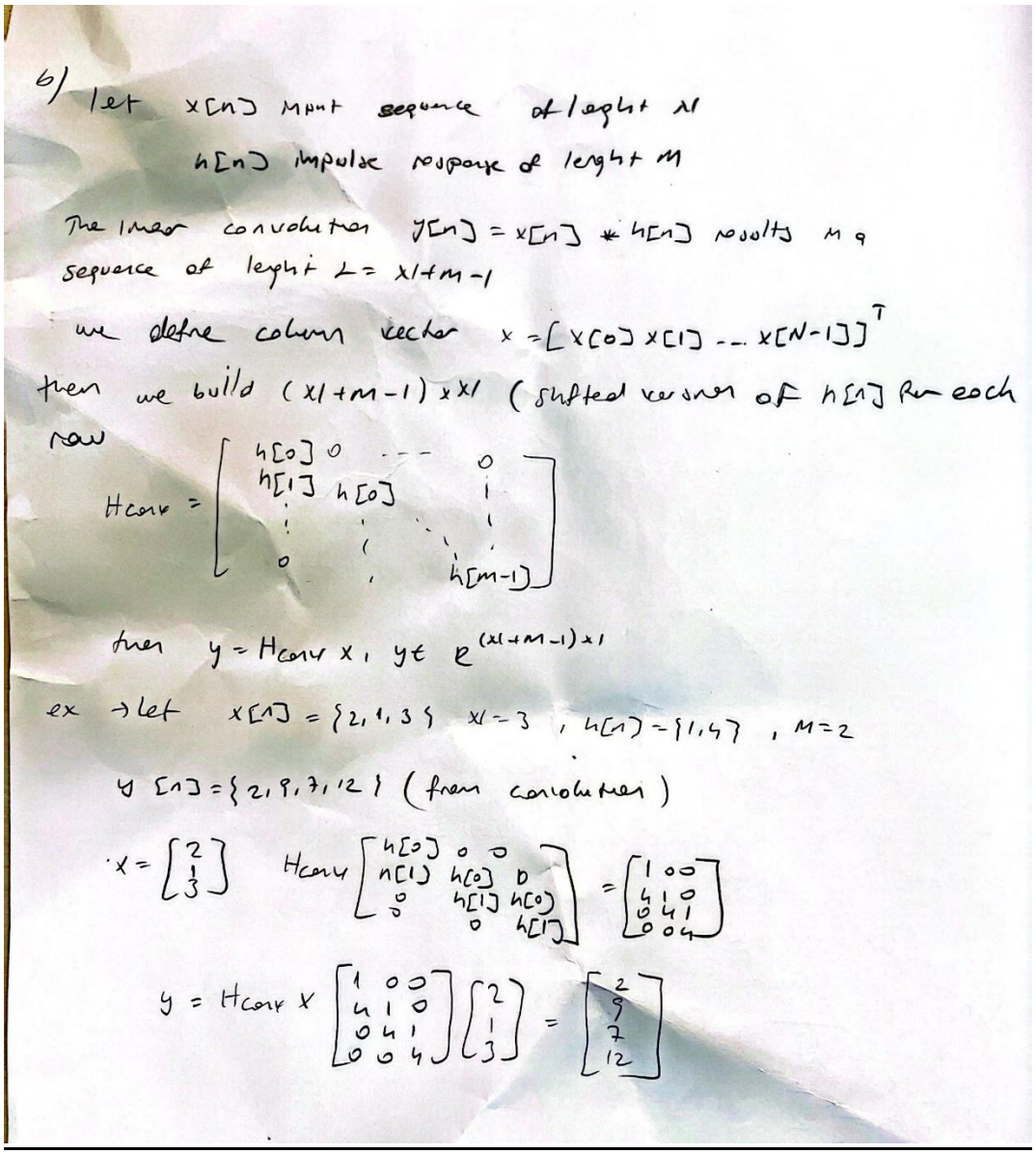

figure 2 : Part 1 paper solution (b)

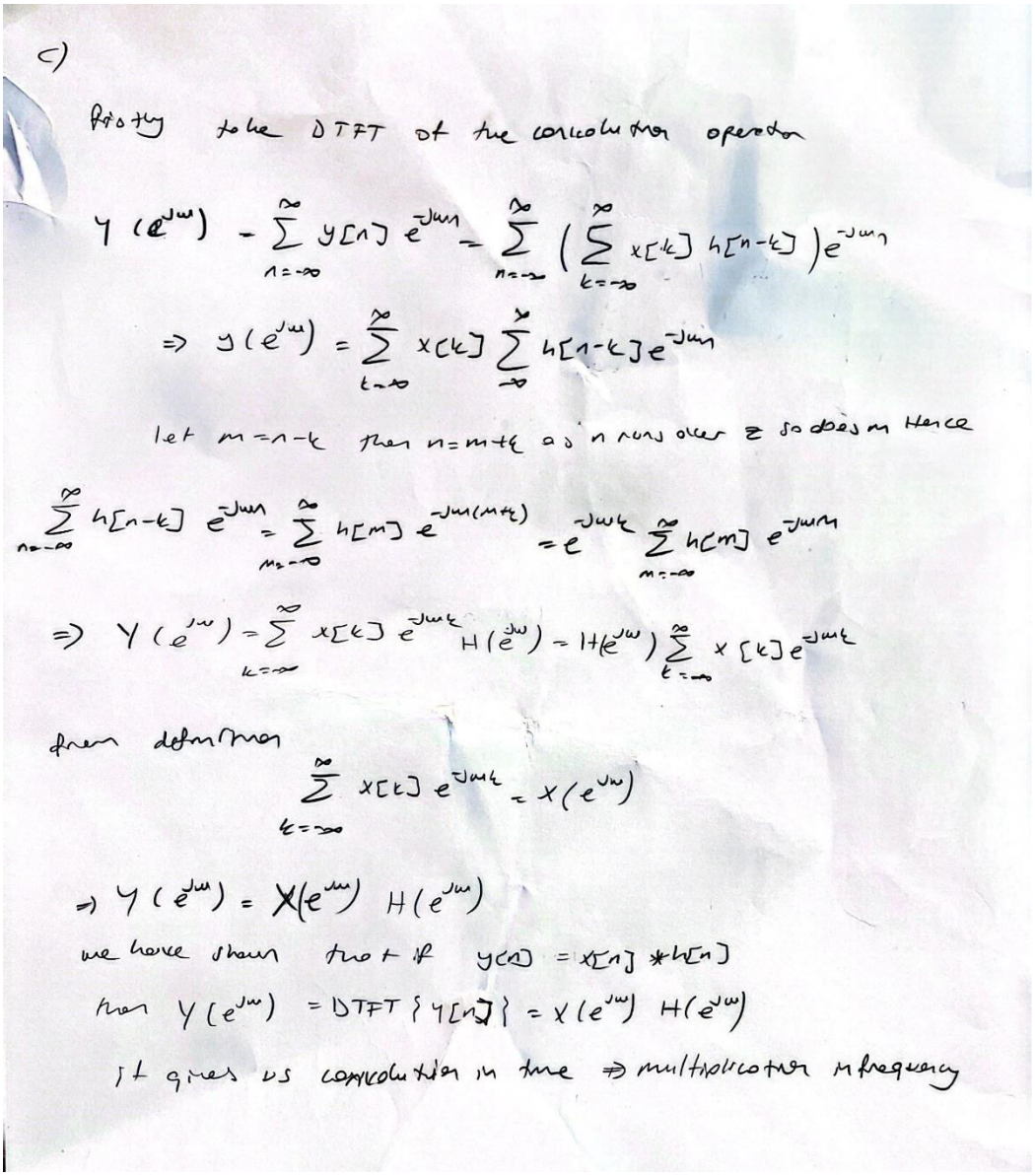

figure 3 : Part1 paper solution (c)

Part 2

Part 2.1: Implementation of DTFT and IDTFT

In this part, we implemented the DTFT and IDTFT transformations that we derived in part 1 in order not to use the ready-made functions of Matlab. Matlab functions created according to the desired template are as follows.

function [X] = DTFT (x, n, w)

function [x] = IDTFT (X, n, w)

- x: finite length discrete-time sequence
- X: frequency-domain signal
- n: the discrete-time variable
- w: the frequency variable

```
function X = DTFT(x,n,w)
  X = exp(-1j*(n(:)*w(:).')).' * x(:);
end
```

```
function x = IDTFT(X,n,w)
  x = real( (exp(1j*(n(:)*w(:).')) * X(:)) / length(w) );
end
```

Also where $\emptyset_1[n] = 1\ for\ 0 \leq n \leq 5\ and\ otherwise = 0$ then we check the norm square difference between $\emptyset_2[n] = IDTFT\{DTFT[\emptyset_1[n]]\}$ which gives us the error;

$$\mathcal{E} = \sum_{n=0}^{N} |\phi_1[n] - \phi_2[n]|^2$$

figure 4 : Error calculation

Then we get a result of E as follows

```
>> lab6_pt2
   1.5127e-07
```

figure 5 : The error result of functions

This is a very small value. There are some alternatives that can cause this margin of error. One of them is MATLAB is using floating point arithmetic. Therefore, these arithmetic calculations that it makes cause small margins of error. Therefore, as the number of samples for the operation we have done increases, the probability of observing this error rate increases. In addition, the sample frequency mistmach made for phi 1 and phi 2 can also cause a margin of error.

Part 2.2: Implementation of Convolution

In this part, we wrote a convolution function using the DTFT and IDTFT functions we created in part 2.1. We express this implemented convolution as linear convolution. The operation we performed here is to calculate the DTFT of the two functions, then transform them to the time domain with IDTFT and perform the multiplication operation. From here, we see that convolution in the time domain corresponds to multiplication in the frequency domain.

function [y] = ConvFUNC(x, h, nx, nh, ny, w)

- x: finite length discrete-time sequence x[n]
- nx: the discrete-time variable for the sequence x[n]
- h: finite length discrete-time sequence h[n]
- nh: the discrete-time variable for the sequence h[n]
- ny: discrete-time variable for the sequence y[n]
- w: the frequency variable
- y: the output of the convolution operation

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

Part 2.3: Another Approach for Convolution

The convolution function we wrote in this section is based on a different method. Here we perform the calculations as matrix based. Here, the matrix we specify as H is the sparse Toelliptz matrix created according to the lengths of x and h. The diagonals of H are filled with shifted h elements. Then we perform matrix multiplication to find the result. The structure of the function is as follows.

function [y] = ConvFUNC_M(x, h)

- x: finite length discrete-time sequence x[n]
- h: finite length discrete-time sequence h[n]
- y: the output of the convolution operation

```
function y = ConvFUNC_M(x,h)
    H = spdiags( ones(length(x)+length(h)-1,1) * h(:).', -(0:length(h)-1),
length(x)+length(h)-1, length(x) );
    y = H * x(:);
end
```

Part 2.4 :Testing the Convolution Function

In this part, we tested ConvFUNC and ConvFUNC_M that we created as custom. Then, we compared the test results we obtained with the results of matlab's built-in function conv. We used the squared norm of the differences in part 2.1 to compare the accuracy of the results. We also tested the calculation times for the functions using tic and toc. According to the results we obtained, we observed that DTFT-based ConvFUNC was the fastest and most effective, while matlab's built-in conv was the slowest. When we look at the accuracy rate, we observe a negligible margin of error in both functions, which shows that the functions are successful.

```
Elapsed times (seconds):
  ConvFUNC (DTFT/IDTFT):     0.0100578
  ConvFUNC_M (matrix):       0.0343742
  MATLAB conv() built-in:    0.0432776

Norm-square differences:
  ||y1 - y3||^2 = 4.2188e-05
  ||y2 - y3||^2 = 0
  ||y1 - y2||^2 = 4.2188e-05
```

figure 6 : The elapsed time of calculations of functions and norm square differences.

Part 3

In this part we design two different types of mobing average filters. These are Simple Moveing Average (SMA) and Gaussian Moving Average (GMA) filters. We use them to reduce the noice of horn corrupted by additive white noise. The sma filter was constructed via averaging across with N = 0.001*fs where fs is smapling frequency. The impulse response of hsmav[n] is a sequence of uniform weights. The GMA is use normalized Gaussian window defined in the range -5 to 5 with μ=0 and σ=0.7 and the result of impulse responses and its plots are as follows. Also both filters are implemented by custom ConvFUNC_M function.

$$h_{\mathrm{SMAV}}[n] = \frac{1}{N} \cdot \underbrace{[1, 1, \cdots, 1, 1]}_{N \text{ times}}$$

$$h_{\mathrm{GMAV}}[n] = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{-\frac{(n-\mu)^2}{2\sigma^2}}$$

figure 7 : SMA and GMA filter algotirhm

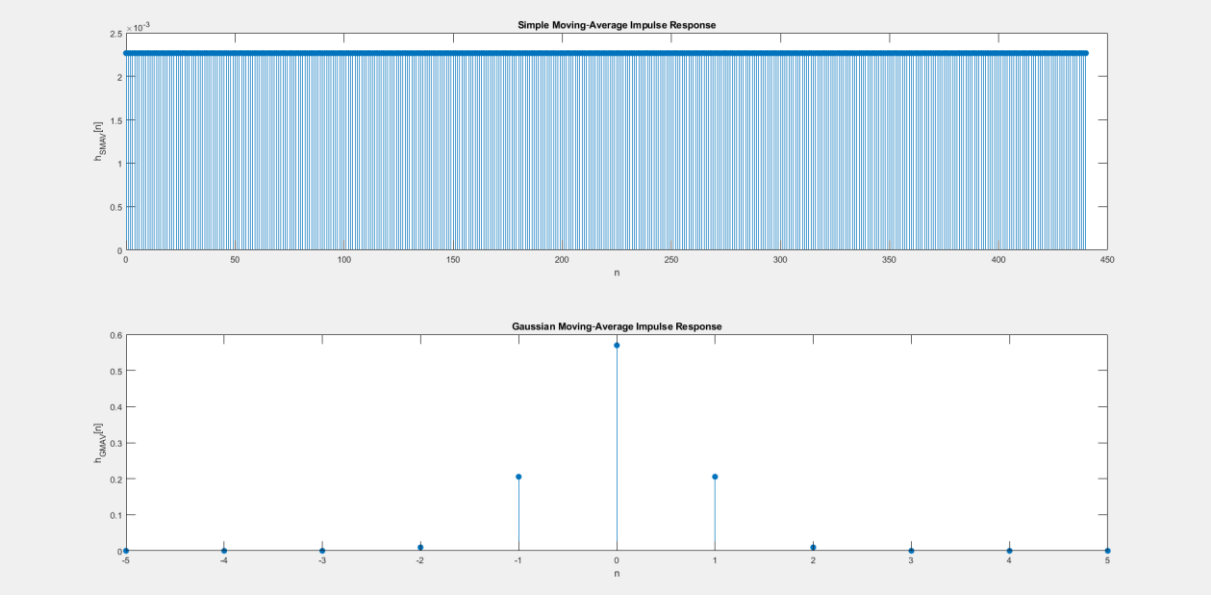

figure 8 : SMA and GMA Impulse Responses

The SMA impulse response appears as a flat rectangular pulse and GMA shows bell-shaped curve concentrated around the center.

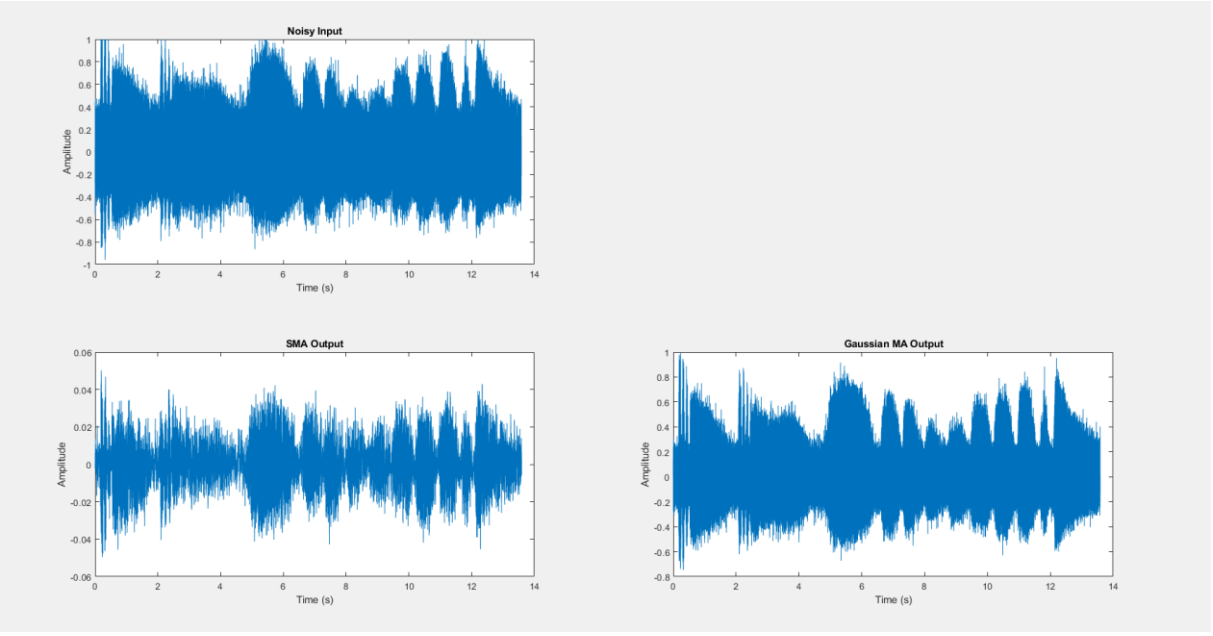

figure 9 : Noisiy signal and filtered output plots with SMA and GMA

As shown in the figure we have noisy signal at the left top and when we use SMA filter we get the output as bottom left and when we use GMA filter we get the result as bottom right. What we observe from these result is both filters successfully reduce the high frequency noise but because of the characteristic of filters, the results smoothing effect is different. The SMA has more uniform smoothing effect but it gives some signal distortion, the reason of this situation is its abrupt cutoff in the time domain. The GMA filter has smoother and bell shaped impulse response , it gives more natural smoothing with less distortion of transient signal components.In general, the Gaussian Moving Average filter was more effective for denoising the audio signal, it provides balance between denoising and less distortion effect.

Part 4

In this part we design a basic equalizer using ideal filters to isolate different frequency bands in an orchestral recording compose of multiple instruments. We achieve this by Gaussian based filter. We design a low pass filter bu using Gaussian with sigma = 0.4 x N where N is the filter lenght. The impulse response was normalized to unit area. Then we design a high pass filter with sigma = 0.02 x N and adding delta fuc at the center. This desgin ensures cancellation of low frequencies. The impulse response of filters can be seen below. Also we import the recordings and analyze DTFT o these recordings. These plots can be observed as follows.

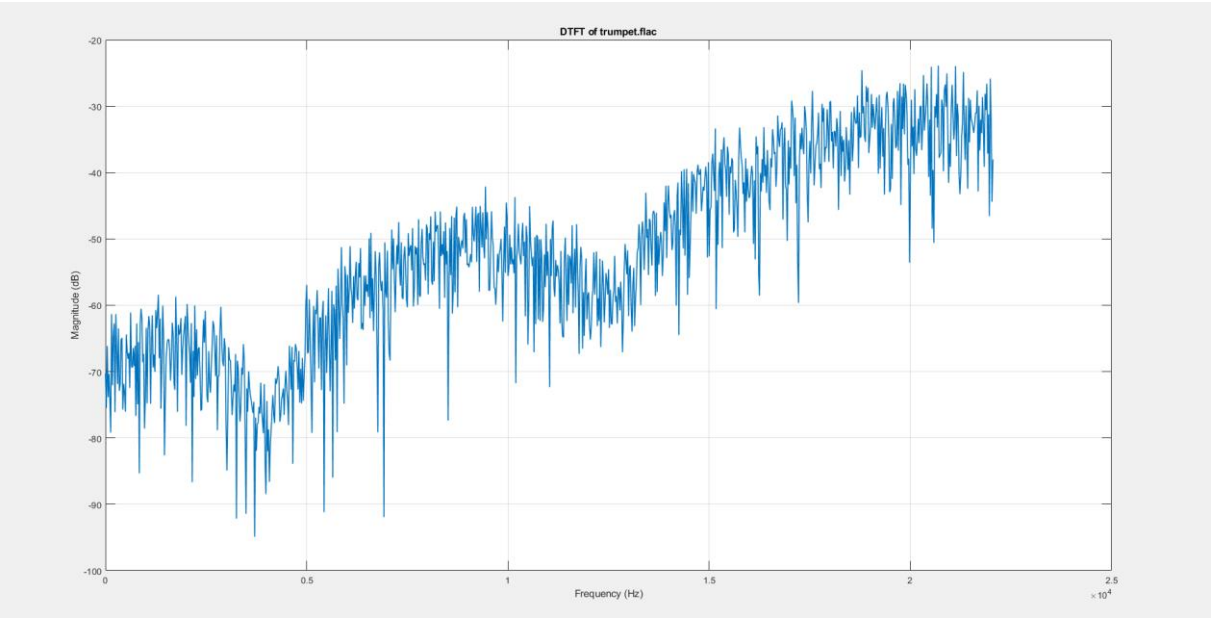

figure 10 : DTFT of Trumped

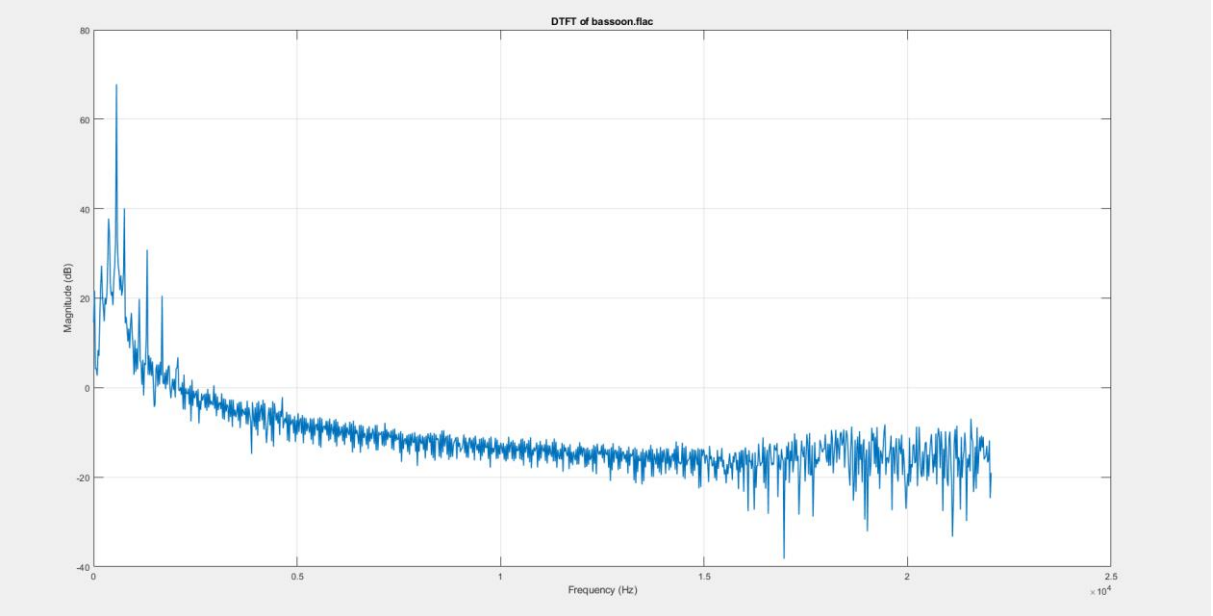

figure 11 : DTFT of Bassoon

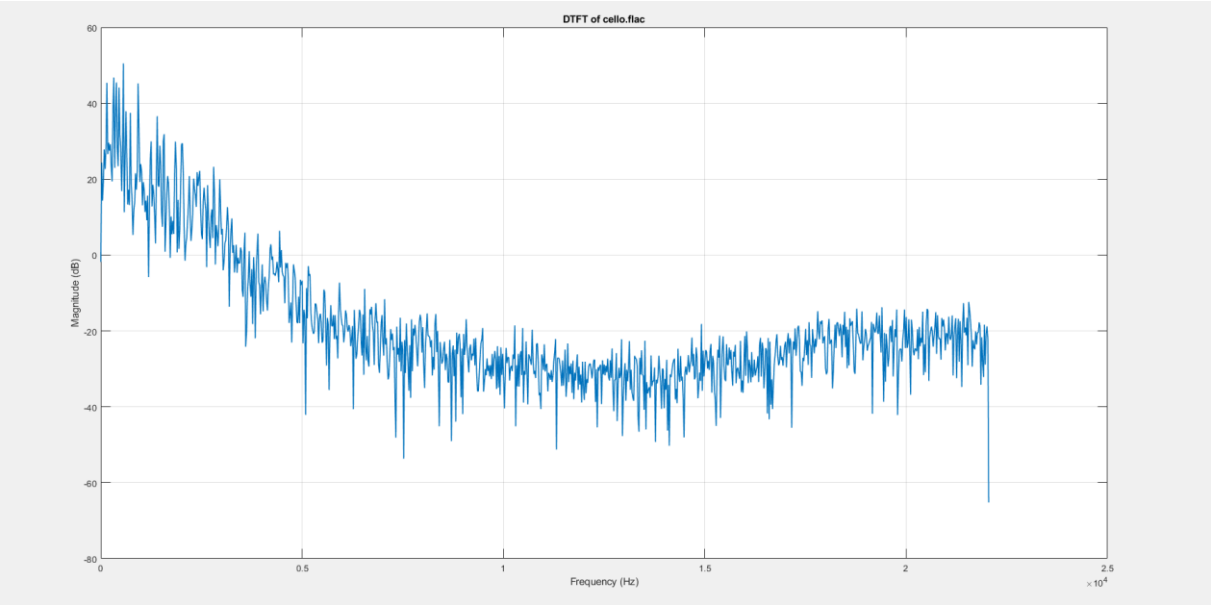

figure 12 : DTFT of cello

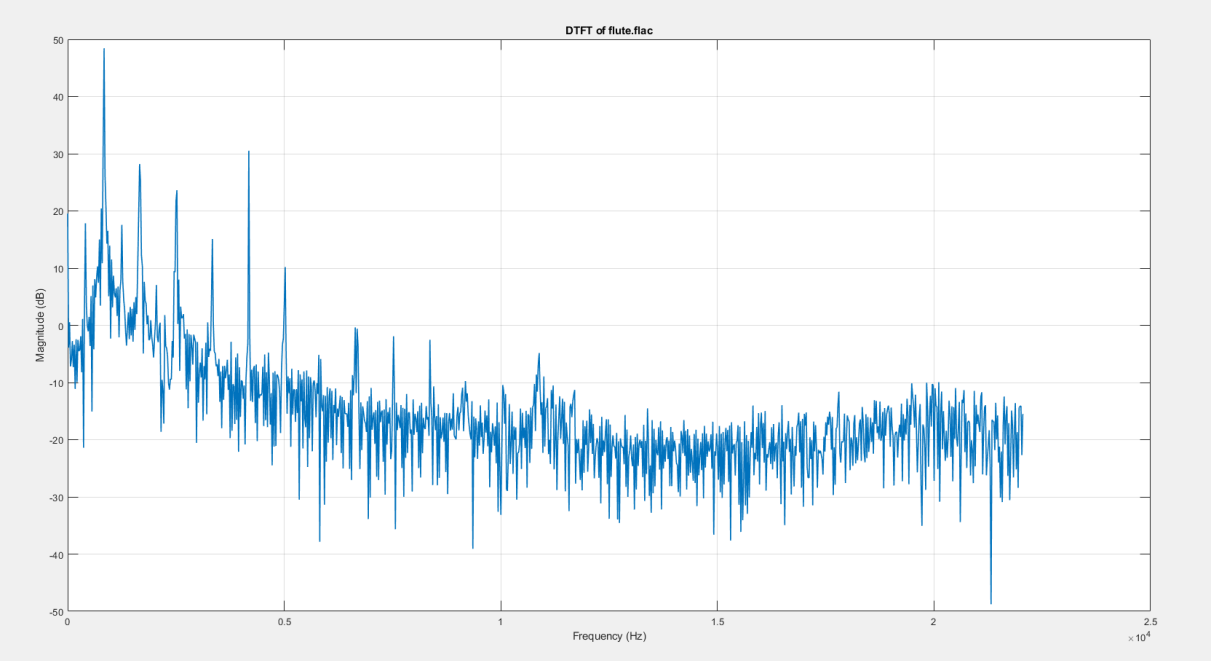

figure 13 : DTFT of flute

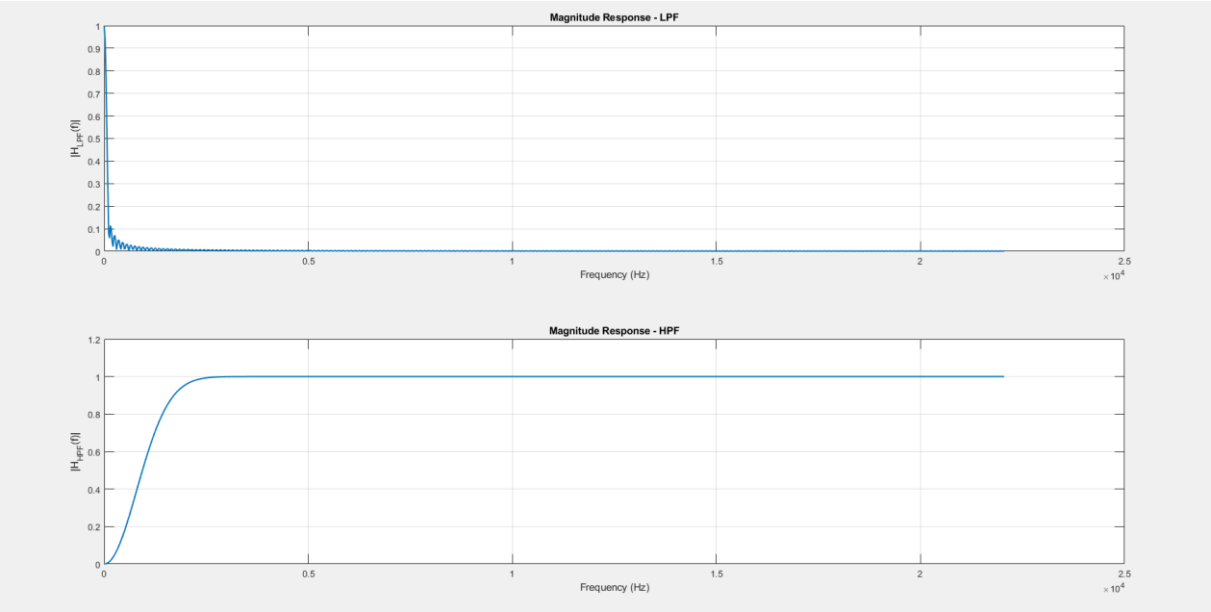

figure 14 : Magnitude response of LPF and HPF

We can clearly see that the trumped produces highest frequency and flute produces mid to high freq and bassoon has the lowest frequency also çello has low to mid frequency. Then we mixed the sounds we had and turned them into an orchestra. And we implemented the LPF and HPF that we created. When we listened and analyzed the sounds, when we listened to the output of the LPF, we hear mostly bassoon and partly cello, in this filter we almost cancel the trumped and flute. When we listen to the output of the HPF, we can listen to it as trumped and flute. The high frequencies of these instruments stand out with the filter performing its duty. In the HPF, we see that the cello and bass are suppressed. It can be seen in the plot below that the filters work correctly.

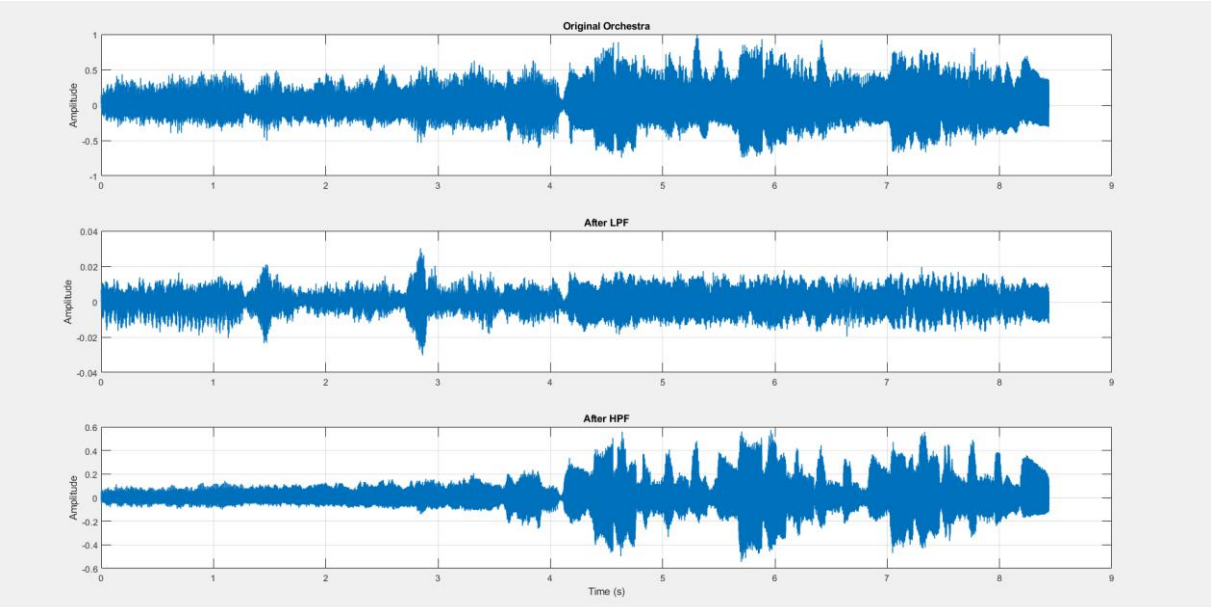

figure 15 : Plot of Original orchestra and filtered form with LPF and HPF

Conclusion

In this lab we analyze fundamental conceept of signal processing like DTFT and IDTFT, and convolution. We learn how DTFT and IDTFT behave and how to construct a convolution fuction by uins Fourier transform. Then we constuct a different conv function by using matrix multiplication algorith then compare them with matlab conv function. Then we learn two different filters as SMA and GMA then compare the outputs of the filter and we decide that Gaussian is more effective and useful. Then we analyze musical instuments and take DTFT of the recordings and analyze them. Then we construct LPF and HPF to use unified orchestra voice filtering. Then we observe the filters are Works very well and we filter the voice as axpected. Overall, we completed all task of this lab and achive required result.

Codes

DTFT

```
function X = DTFT(x,n,w)
  X = exp(-1j*(n(:)*w(:).')).' * x(:);
end
```

IDTFT

```
function x = IDTFT(X,n,w)
  x = real( (exp(1j*(n(:)*w(:).')) * X(:)) / length(w) );
end
```

ConvFUNC

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

ConvFUNC_M

```
function y = ConvFUNC_M(x,h)
    H = spdiags( ones(length(x)+length(h)-1,1) * h(:).', -(0:length(h)-1),
length(x)+length(h)-1, length(x) );
    y = H * x(:);
end
```

Part 2.1

```
% ϕ₁[n] = 1 for n=0:4, zero otherwise
n = 0:5;
phi1 = double(n<5);

% freq grid
w = -pi:0.001:pi;

% DTFT and IDTFT
```

```
X  = DTFT(phi1, n, w);
phi2 = IDTFT(X, n, w);

% error
E = sum((phi1(:)-phi2(:)).^2);
disp(E)
```

Part 2.4

```
% input sequences
x1 = [2, 4, 6, 8, 7, 6, 5, 4, 3, 2, 1];
x2 = [1, 2, 1, -1];

%conv via DTFT/IDTFT (Part 2.2)
tic
y1 = ConvFUNC(x1, x2, 0:length(x1)-1, 0:length(x2)-1, 0:(length(x1)+length(x2)-2),
-pi:0.001:pi);
t1 = toc;

%conv via matrix multiplication (Part 2.3)
tic
y2 = ConvFUNC_M(x1, x2);
t2 = toc;

%MATLAB built-in
tic
y3 = conv(x1, x2);
t3 = toc;

%Compare lengths
L = length(x1) + length(x2) - 1;
assert( all([numel(y1),numel(y2),numel(y3)] == L), 'Output lengths mismatch' );


E12 = sum((y1(:) - y3(:)).^2);
E23 = sum((y2(:) - y3(:)).^2);
E13 = sum((y1(:) - y2(:)).^2);


fprintf('\nElapsed times (seconds):\n');
fprintf('  ConvFUNC (DTFT/IDTFT):     %g\n', t1);
fprintf('  ConvFUNC_M (matrix):       %g\n', t2);
fprintf('  MATLAB conv() built-in:    %g\n', t3);

fprintf('\nNorm-square differences:\n');
fprintf('  ||y1 – y3||^2 = %g\n', E12);
fprintf('  ||y2 – y3||^2 = %g\n', E23);
fprintf('  ||y1 – y2||^2 = %g\n\n', E13);
```

Part 3

```
[x_noisy, fs] = audioread('Part3_recording.flac');
t = (0:length(x_noisy)-1)/fs;
```

```
% Parameters
N = round(0.01 * fs);
h_SMA = (1/N) * ones(1, N);
nG = -5:5;
sigma = 0.7;
h_GMA = (1/(sigma*sqrt(2*pi))) * exp(-((nG-0).^2)/(2*sigma^2));
h_GMA = h_GMA / sum(h_GMA);             % normalize area to 1

% SMA GMA

y_SMA = ConvFUNC_M(x_noisy, h_SMA).';    % note transpose back to row
y_GMA = ConvFUNC_M(x_noisy, h_GMA).';

t_SMA = (0:length(y_SMA)-1)/fs;
t_GMA = (0:length(y_GMA)-1)/fs;

figure;
subplot(2,2,1);
plot(t, x_noisy);
xlabel('Time (s)'); ylabel('Amplitude');
title('Noisy Input');

subplot(2,2,3);
plot(t_SMA, y_SMA);
xlabel('Time (s)'); ylabel('Amplitude');
title('SMA Output');

subplot(2,2,4);
plot(t_GMA, y_GMA);
xlabel('Time (s)'); ylabel('Amplitude');
title('Gaussian MA Output');

soundsc(x_noisy, fs);    disp('Playing: Noisy');    pause(length(x_noisy)/fs + 1);
soundsc(y_SMA,   fs);    disp('Playing: SMA filtered'); pause(length(y_SMA)/fs +
1);
soundsc(y_GMA,   fs);    disp('Playing: Gaussian MA');  % end

figure;
subplot(2,1,1);
stem(0:N-1, h_SMA, 'filled');
xlabel('n'); ylabel('h_{SMAV}[n]');
title('Simple Moving-Average Impulse Response');

subplot(2,1,2);
stem(nG, h_GMA, 'filled');
xlabel('n'); ylabel('h_{GMAV}[n]');
title('Gaussian Moving-Average Impulse Response');
```

Part 4

```
clear; close all;

[in_bass, fs] = audioread('Part4_recordings/bassoon.flac');
[in_cell, ~]  = audioread('Part4_recordings/cello.flac');
[in_flut, ~]  = audioread('Part4_recordings/flute.flac');
[in_trum, ~]  = audioread('Part4_recordings/trumpet.flac');
```

```
L = max([length(in_bass), length(in_cell), length(in_flut), length(in_trum)]);
in_bass(end+1:L) = 0;
in_cell(end+1:L) = 0;
in_flut(end+1:L) = 0;
in_trum(end+1:L) = 0;

% Create orchestra
orch = in_bass + in_cell + in_flut + in_trum;
t = (0:L-1)/fs;

%Design Gaussian LPF and HPF
N = round(0.01 * fs);                    % Filter length
n = -floor(N/2):floor(N/2);             % Time index

sigma_lpf = 0.4 * N;
h_lpf = exp(-(n.^2)/(2*sigma_lpf^2));
h_lpf = h_lpf / sum(h_lpf);

sigma_hpf = 0.02 * N;
h_g = exp(-(n.^2)/(2*sigma_hpf^2));
h_g = h_g / sum(h_g);
h_hpf = -h_g; h_hpf((end+1)/2) = h_hpf((end+1)/2) + 1;

%Plot frequency responses
w = linspace(0, pi, 1024);
H_lpf = DTFT(h_lpf, n, w);
H_hpf = DTFT(h_hpf, n, w);

figure;
subplot(2,1,1);
plot(w/pi*(fs/2), abs(H_lpf), 'LineWidth', 1.5);
title('Magnitude Response - LPF'); xlabel('Frequency (Hz)');
ylabel('|H_{LPF}(f)|'); grid on;

subplot(2,1,2);
plot(w/pi*(fs/2), abs(H_hpf), 'LineWidth', 1.5);
title('Magnitude Response - HPF'); xlabel('Frequency (Hz)');
ylabel('|H_{HPF}(f)|'); grid on;

max_len = fs;
w = linspace(0, pi, 1024);
files = dir('Part4_recordings/*.flac');

for k = 1:numel(files)
    [x, ~] = audioread(fullfile(files(k).folder, files(k).name), [1 max_len]);
    n_x = 0:length(x)-1;
    X = DTFT(x, n_x, w);

    figure;
    plot(w/pi*(fs/2), 20*log10(abs(X) + 1e-6), 'LineWidth', 1.2);
    title(['DTFT of ', files(k).name]);
    xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)'); grid on;
end

% Convolve using ConvFUNC_M
y_lp = ConvFUNC_M(orch, h_lpf)';
y_hp = ConvFUNC_M(orch, h_hpf)';
```

```
y_lp = y_lp(N:end-N+1);
y_hp = y_hp(N:end-N+1);
orch = orch(1:length(y_lp));
t = t(1:length(y_lp));

% Normalize
max_val = max([max(abs(y_lp)), max(abs(y_hp)), max(abs(orch))]);
y_lp = y_lp / max_val * 0.99;
y_hp = y_hp / max_val * 0.99;
orch = orch / max_val * 0.99;

soundsc(orch, fs); disp('Playing: Original'); pause(length(orch)/fs + 1);
soundsc(y_lp, fs);  disp('Playing: LPF');     pause(length(y_lp)/fs + 1);
soundsc(y_hp, fs);  disp('Playing: HPF');     pause(length(y_hp)/fs + 1);

audiowrite('orchestra_LPF.flac', y_lp, fs);
audiowrite('orchestra_HPF.flac', y_hp, fs);

figure;
subplot(3,1,1);
plot(t, orch, 'LineWidth', 1.1); title('Original Orchestra'); ylabel('Amplitude');
grid on;
subplot(3,1,2);
plot(t, y_lp, 'LineWidth', 1.1); title('After LPF'); ylabel('Amplitude'); grid on;
subplot(3,1,3);
plot(t, y_hp, 'LineWidth', 1.1); title('After HPF'); xlabel('Time (s)');
ylabel('Amplitude'); grid on;
```