EEE 321

Report: Signals and Systems Lab 6

Yigit Turkmen – 22102518

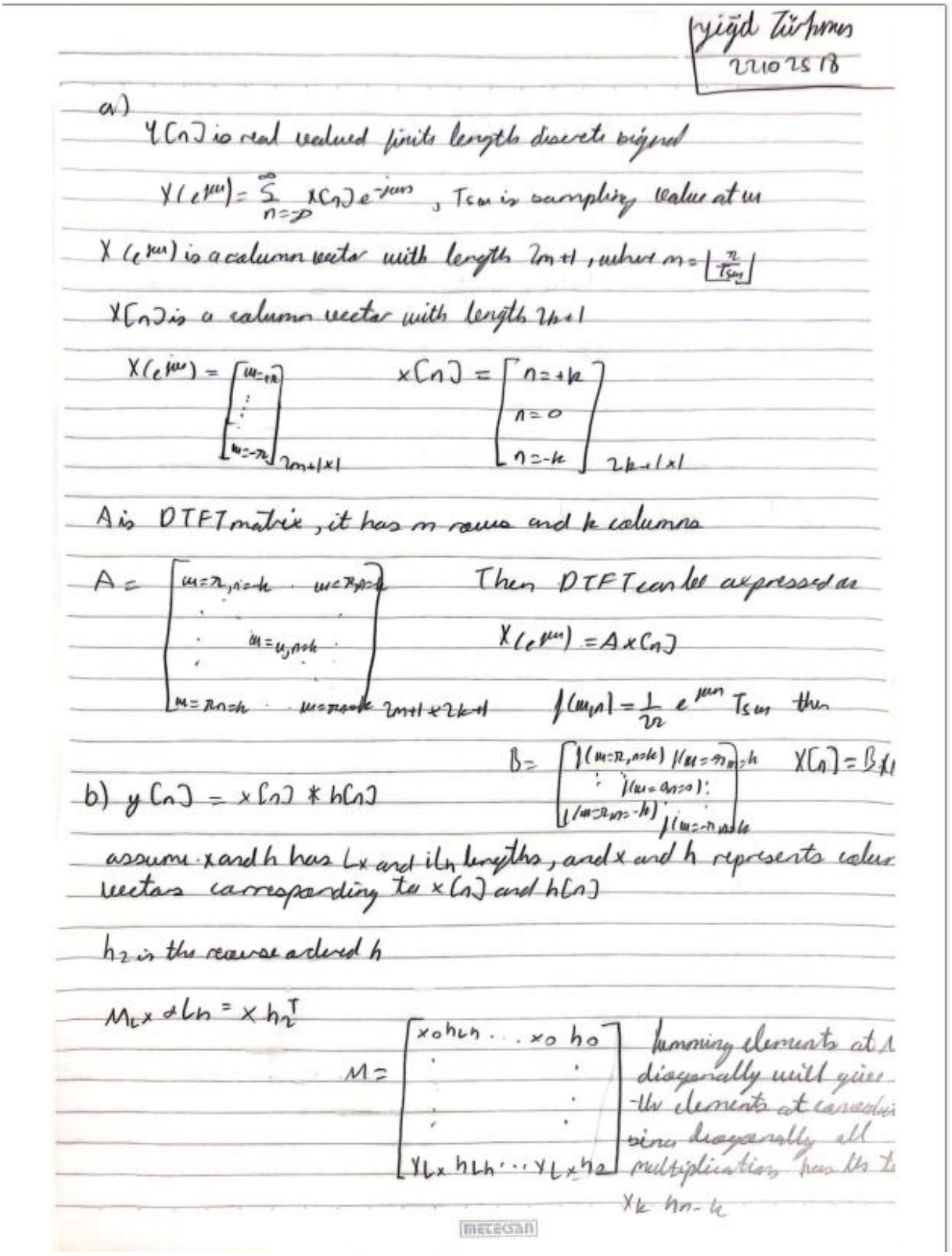
May 11, 2024

Introduction:

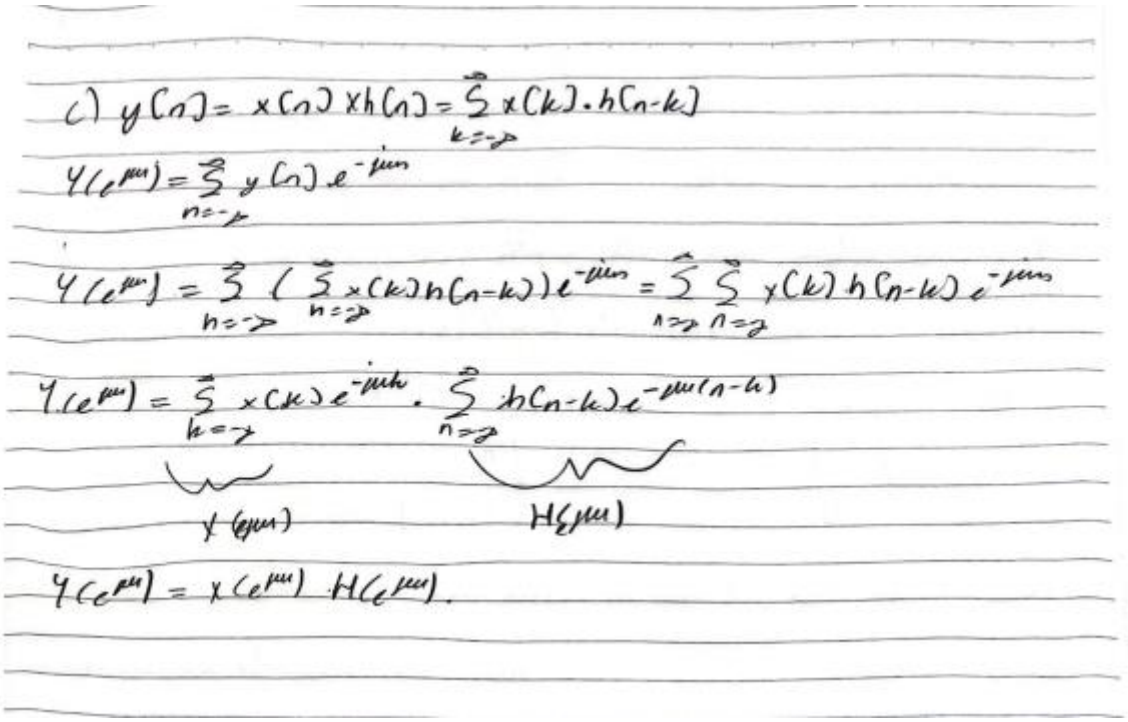
This lab focuses on exploring the Discrete-Time Fourier Transform (DTFT), its inverse (IDTFT), and the convolution process. By applying these techniques to sound signals, we aim to gain a deeper understanding of how different types of filters affect these signals. This practical exploration not only reinforces theoretical concepts discussed in lectures but also provides hands-on experience with MATLAB, enhancing our ability to tackle real-world signal processing challenges. Through this lab, we will develop and test our own implementations of these transformations and convolution without relying on built-in MATLAB functions, pushing the boundaries of our programming skills and theoretical knowledge.

Part 1

See the calculations and derivations for this part below.



Part 1 a and b


c) $y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$
$Y(e^{j\omega}) = \sum_{n=-\infty}^{\infty} y[n] e^{-j\omega n}$
$Y(e^{j\omega}) = \sum_{n=-\infty}^{\infty} \left(\sum_{k=-\infty}^{\infty} x[k] h[n-k] \right) e^{-j\omega n} = \sum_{k=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x[k] h[n-k] e^{-j\omega n}$
$Y(e^{j\omega}) = \sum_{k=-\infty}^{\infty} x[k] e^{-j\omega k} \cdot \sum_{n=-\infty}^{\infty} h[n-k] e^{-j\omega(n-k)}$
$X(e^{j\omega})$
$H(e^{j\omega})$
$Y(e^{j\omega}) = X(e^{j\omega}) H(e^{j\omega})$.

Part 1 c

Part 2

2.1 Implementation of DTFT and IDTFT:

See the related functions code for this part.

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

"DTFT.m"

```
function [x] = IDTFT(X,n,w)
    x = real((0.001/(2*pi)) * (exp(1j*n*w.'))*X);
end
```

"IDTFT.m"

Difference error: 1.7 * 10**-6 which is very small number that is close to zero. Not exactly zero but almost zero; therefore, small numerical errors are generally acceptable and expected in practical computational environments.

```
>> lab_6_part_2_1
Norm square difference between signal_phi_1 and transformed_signal_phi_2 is:
   1.6981e-06
```

"The difference between phi1 and ph2"

2.2 Implementation of Convolution:

See the related function for this part.

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

"ConvFUNC.m"

2.3 Another Approach for Convolution:

See the related function for this part.

```
function [y] = ConvFUNC_M(x,h)
    A = x*flip(h).';
    y = flip(sum(spdiags(A),1).');
end
```

"ConvFUNC.m"

2.4 Testing the Convolution Function:

Elapsed times for y1, y2 and y3 respectively in matlab:

```
Time it takes to execute ConvFUNC:
    0.0039

Time it takes to execute ConvFUNC_M:
    0.0016

Time it takes to execute built-in conv:
   5.7270e-04
```

As expected built in function is the fastest function here followed by Conv_M and Conv.

Differences for y1, y2 and y3 respectively in matlab:

```
Norm square difference between ConvFUNC and ConvFUNC_M is:
   1.2807e-20

Norm square difference between ConvFUNC_M and default conv is:
     0

Norm square difference between ConvFUNC and default conv is:
   1.2807e-20
```

So, as expected results are very close to the zero therefore, results are sensible. Possible error source are computational environments.

Part 3

Comments:

From hearings, Gaussian moving average filter is much better than the simple moving average filter. Why?

Effectiveness of Noise Reduction: Which filter more effectively reduces noise while preserving the original audio characteristics? Typically, the GMA filter, due to its weighted averaging, it preserves more of the signal's details while reducing noise compared to the SMA filter, which applies an unweighted average across the window.

Sound Quality: Which filter maintains the quality of the original recording? The GMA filter introduces less distortion because it smoothly transitions the weights in the averaging process.

SMA Filter is a type of low-pass filter because it averages the signal over a period, effectively smoothing out the high-frequency fluctuations (noise) while allowing low-frequency components (main signal) to pass.

GMA Filter also is a type of low-pass filter, but with a weighted average based on a Gaussian function. This approach gives more importance to the central samples within the window and less to those further away, which can help in preserving signal features while reducing noise.

See the figure below to observe plots related to this part (figure 1).

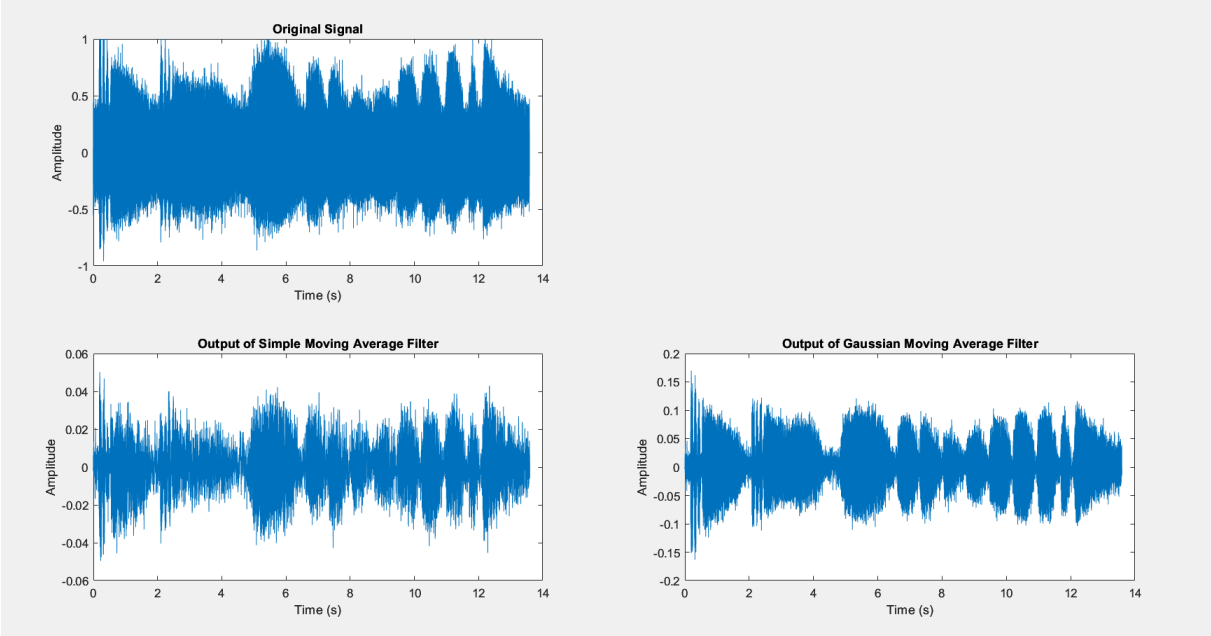


Figure 1: Plots for this part: Original sound, SMA filter and GMA filter

Part 4

See the figure below to observe frequency responses of filters (figure 2).

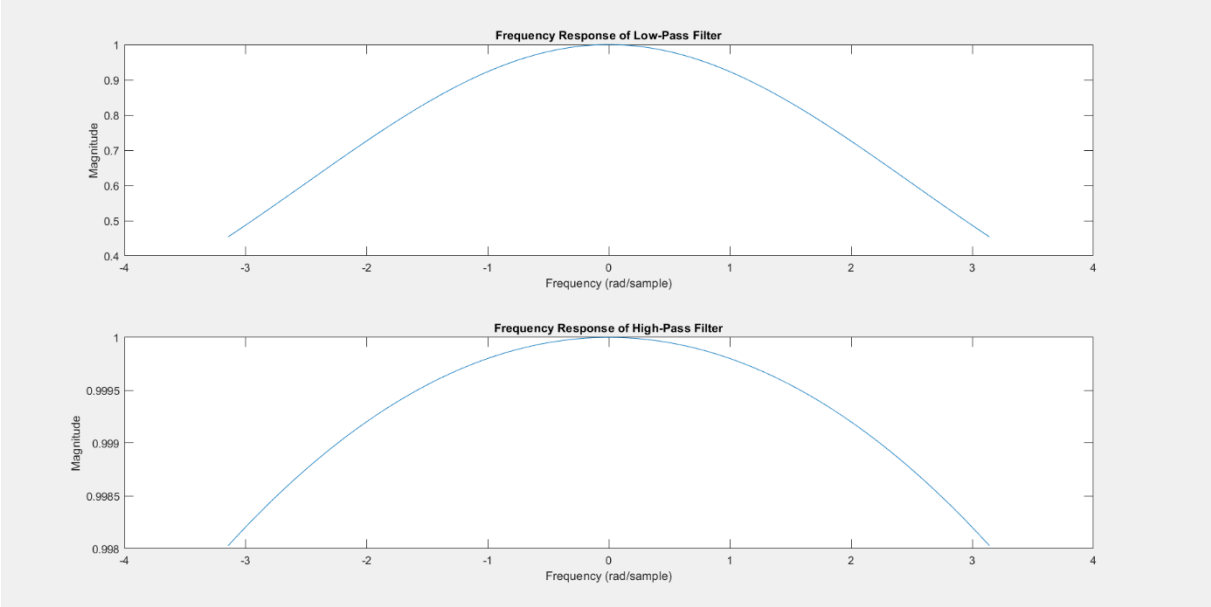


Figure 2: Plots of frequency responses of filters

See the figure below to observe the frequency analysis of the instruments (figure 3).

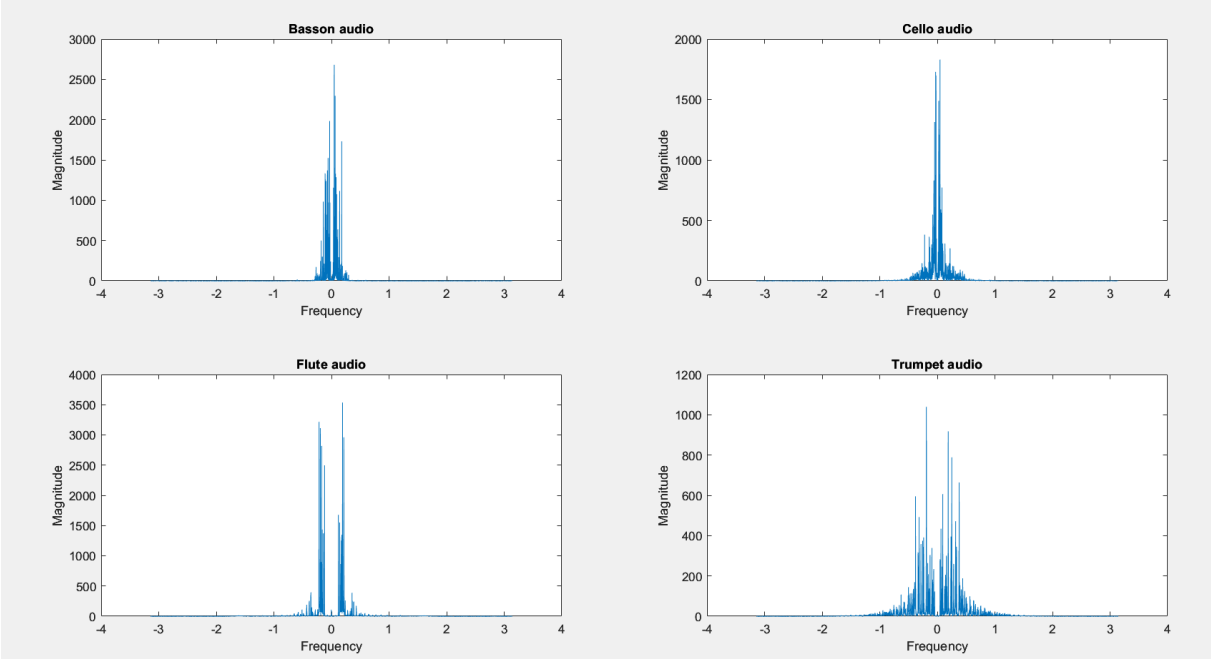


Figure 3: Plots of frequency responses of instruments

Comments on frequency characteristics of the instruments: From the plots it is observed that Basson and Cello are low frequency instruments while Flute and Trumpet are high frequency instruments.

See the figure below to observe the original sound, lpf sound and hpf sound (figure 4).

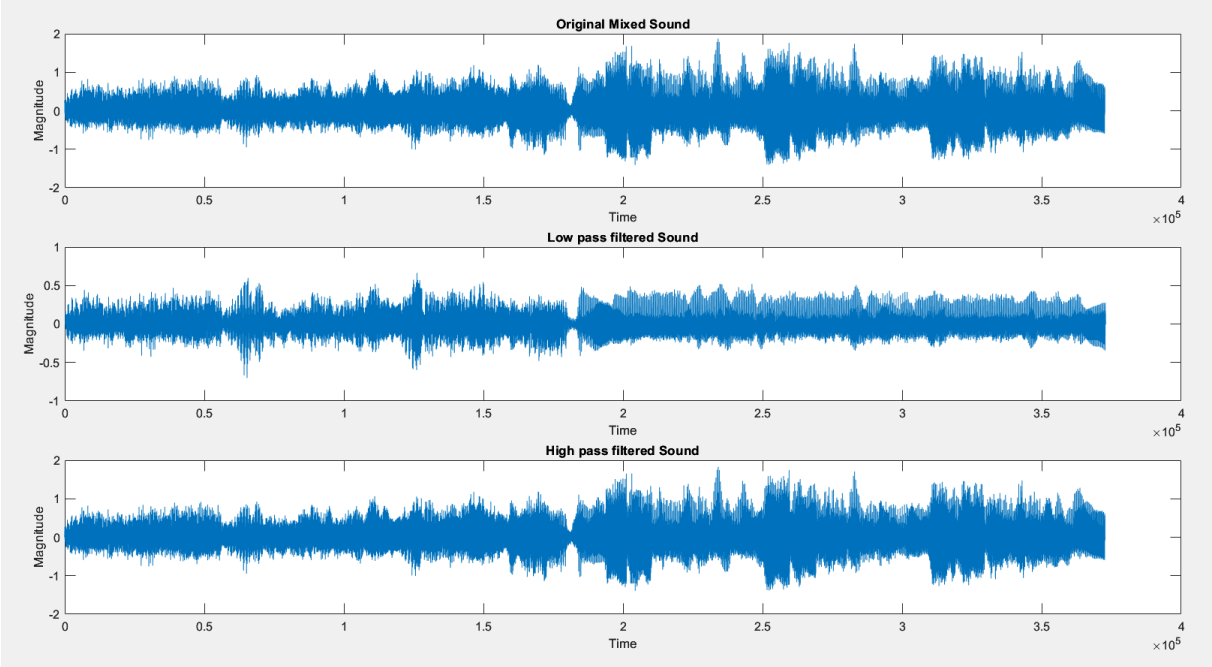


Figure 4: Plots of original sound, lpf and hpf sounds.

Comments on Listen to each result and comment on which instruments are eliminated for each filter. Do the results make sense considering the frequency distribution of the instruments?

In low pass, trumpet and flute are eliminated. In high pass, cello and basson are eliminated whic are the expected results.

Yes, the results make sense considering the frequency distribution of the instruments.

Whole codes used for this assignment:

```
function [y] = ConvFUNC_M(x,h)
    A = x*flip(h).';
    y = flip(sum(spdiags(A),1).');
end
```

"ConvFUNC.m"

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

"DTFT.m"

```
function [x] = IDTFT(X,n,w)
    x = real((0.001/(2*pi)) * (exp(1j*n*w.'))*X);
end
```

"IDTFT.m"

```
max_value = 50;
signal_function = @(n) (n >= 0) & (n < 5);  % Defining the signal function
time_step_width = 0.001;  % Time step width for frequency domain
n_sequence = (-max_value:1:max_value).';  % Sequence of time samples
omega = (-pi:time_step_width:pi).';  % Frequency range from -pi to pi
N = length(n_sequence);  % Length of the sequence

signal_phi_1 = signal_function(n_sequence);  % Generating the signal

transformed_signal_phi_2 = IDTFT(DTFT(signal_phi_1, n_sequence, omega),
n_sequence, omega);  % Applying DTFT and then IDTFT

error_sum = sum(abs(signal_phi_1 - transformed_signal_phi_2).^2);  % Calculating
the norm square difference

disp("Norm square difference between signal_phi_1 and transformed_signal_phi_2 is:
");
disp(error_sum);  % Displaying the error
```

"part2_1.m"

```
% Define input signals
input_signal_1 = [2, 4, 6, 8, 7, 6, 5, 4, 3, 2, 1].';
input_signal_2 = [1, 2, 1, -1].';

% Define frequency domain sampling parameters
frequency_step = 0.001;
omega_range = (-pi:frequency_step:pi).';

% Create time indices for signals
time_indices_1 = (0:length(input_signal_1)-1).';
time_indices_2 = (0:length(input_signal_2)-1).';
result_indices = (0:length(input_signal_1) + length(input_signal_2)-2).';

% Execute ConvFUNC and measure time
tic;
output_y1 = ConvFUNC(input_signal_1, input_signal_2, time_indices_1,
time_indices_2, result_indices, omega_range);
execution_time_1 = toc;
disp("Time it takes to execute ConvFUNC: ")
disp(execution_time_1)

% Execute ConvFUNC_M and measure time
tic;
output_y2 = ConvFUNC_M(input_signal_1, input_signal_2);
execution_time_2 = toc;
disp("Time it takes to execute ConvFUNC_M: ")
disp(execution_time_2)

% Execute built-in conv function and measure time
tic;
output_y3 = conv(input_signal_1, input_signal_2);
execution_time_3 = toc;
disp("Time it takes to execute built-in conv: ")
disp(execution_time_3)

% Calculate and display norm square differences
error_between_y1_and_y2 = sum(abs(output_y1 - output_y2).^2);
error_between_y2_and_y3 = sum(abs(output_y2 - output_y3).^2);
error_between_y1_and_y3 = sum(abs(output_y1 - output_y3).^2);

disp("Norm square difference between ConvFUNC and ConvFUNC_M is: ");
disp(error_between_y1_and_y2);
disp("Norm square difference between ConvFUNC_M and default conv is: ");
disp(error_between_y2_and_y3);
disp("Norm square difference between ConvFUNC and default conv is: ");
disp(error_between_y1_and_y3);
```

“part2_4.m”

```
[audio_signal, sampling_rate] = audioread('Part3_recording.flac');

% Define frequency domain parameters
frequency_step_size = 0.001;
frequency_vector = (-pi:frequency_step_size:pi).';

% Define filter length as 1% of the sampling rate
filter_length = 0.01 * sampling_rate;

% Time vector for filter design
time_vector = (-5:(10.01/filter_length):5).';

% Simple Moving Average (SMA) filter design
simple_moving_average_filter = ones(filter_length, 1) / filter_length;

% Gaussian Moving Average (GMA) filter parameters
standard_deviation = 0.7;
mean = 0;

% GMA filter design
gaussian_moving_average_filter = (1 / (standard_deviation * sqrt(2 * pi))) * exp(-
(time_vector - mean).^2 / (2 * standard_deviation^2));
gaussian_moving_average_filter = gaussian_moving_average_filter /
sum(gaussian_moving_average_filter);

% Apply convolution using the matrix method for SMA filter
filtered_signal_sma = ConvFUNC_M(audio_signal, simple_moving_average_filter);

% Apply convolution using the matrix method for GMA filter
filtered_signal_gma = ConvFUNC_M(audio_signal, gaussian_moving_average_filter);

% Plotting the results
figure;
subplot(2, 2, 1);
plot((1:length(audio_signal)) / sampling_rate, audio_signal);
title('Original Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2, 2, 3);
plot((1:length(filtered_signal_sma)) / sampling_rate, filtered_signal_sma);
title('Output of Simple Moving Average Filter');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2, 2, 4);
plot((1:length(filtered_signal_gma)) / sampling_rate, filtered_signal_gma);
title('Output of Gaussian Moving Average Filter');
xlabel('Time (s)');
ylabel('Amplitude');
```

“part3.m”

```
[basson_audio, freq] = audioread('bassoon.flac');
[cello_audio, freq] = audioread('cello.flac');
[flute_audio, freq] = audioread('flute.flac');
[trumpet_audio, freq] = audioread('trumpet.flac');


mixed_audio = basson_audio+cello_audio+flute_audio+trumpet_audio;

frequency_resolution = 0.001;
omega = (-pi:frequency_resolution:pi).';

N = 0.01 * freq;
time_index = (-5:(10.01/N):5).';


mu = 0;

sigma_low = 0.4;
low_pass_filter = (1/(sigma_low*sqrt(2*pi)))*exp(-
(time_index).^2/(2*sigma_low^2));
low_pass_filter = low_pass_filter / sum(low_pass_filter);

sigma_high = 0.02;
high_pass_filter = (1/(sigma_high*sqrt(2*pi)))*exp(-
(time_index).^2/(2*sigma_high^2));
high_pass_filter = high_pass_filter / sum(high_pass_filter);

dtft_low_pass = DTFT(low_pass_filter, time_index, omega);

dtft_high_pass = DTFT(high_pass_filter, time_index, omega);

figure;
subplot(2, 1, 1);
plot(omega, abs(dtft_low_pass));
title('Frequency Response of Low-Pass Filter');
xlabel('Frequency (rad/sample)');
ylabel('Magnitude');


subplot(2, 1, 2);
plot(omega,abs(dtft_high_pass));
title('Frequency Response of High-Pass Filter');
xlabel('Frequency (rad/sample)');
ylabel('Magnitude');


basson_dtft = DTFT(basson_audio,(0:(length(basson_audio)-1)).',omega);
cello_dtft = DTFT(cello_audio,(0:(length(cello_audio)-1)).',omega);
flute_dtft = DTFT(flute_audio,(0:(length(flute_audio)-1)).',omega);
trumpet_dtft = DTFT(trumpet_audio,(0:(length(trumpet_audio)-1)).',omega);
figure;
subplot(2,2,1);
plot(omega,abs(basson_dtft));
xlabel("f");
ylabel("Magnitude");
title("Basson audio");
subplot(2,2,2);
```

```
plot(omega,abs(cello_dtft));
xlabel("Frequency");
ylabel("Magnitude");
title("Cello audio");
subplot(2,2,3);
plot(omega,abs(flute_dtft));
xlabel("Frequency");
ylabel("Magnitude");
title("Flute");
subplot(2,2,4);
plot(omega,abs(trumpet_dtft));
xlabel("Frequency");
ylabel("Magnitude");
title("Trumpet audio");

mixed_audio_low_pass = ConvFUNC_M(mixed_audio, low_pass_filter);

mixed_audio_high_pass = ConvFUNC_M(mixed_audio, high_pass_filter);

figure;
subplot(3,1,1);
plot(mixed_audio);
xlabel("Time");
ylabel("Magnitude");
title("Original Mixed Sound");
subplot(3,1,2);
plot(mixed_audio_low_pass);
xlabel("Time");
ylabel("Magnitude");
title("Low pass filtered Sound");
subplot(3,1,3);
plot(mixed_audio_high_pass);
xlabel("Time");
ylabel("Magnitude");
title("High pass filtered Sound");
```

“part4.m”