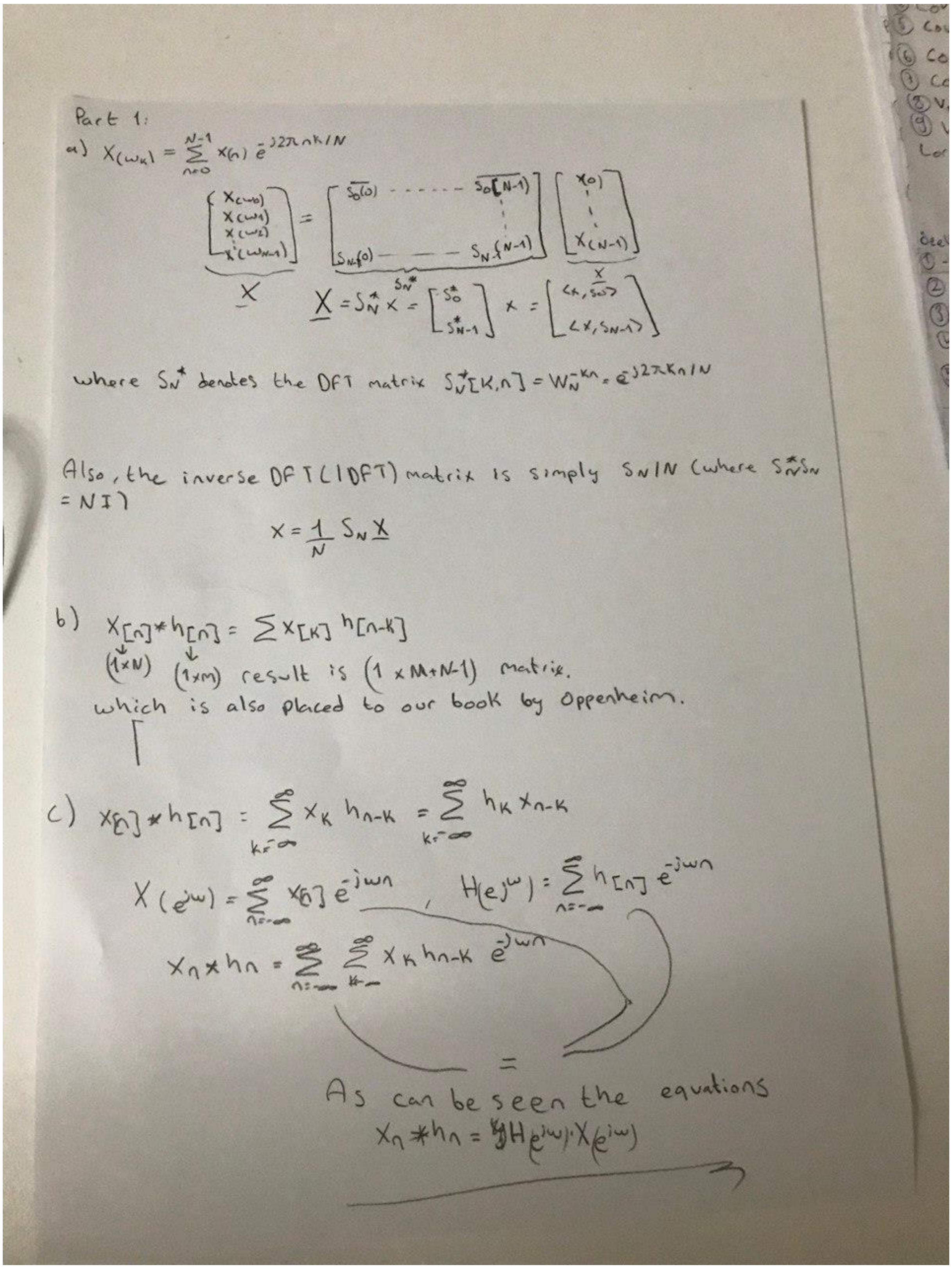
Muhammed Enes Inanc

22001799

EEE-321 Lab 6

Part 1:



Part 1:

a) $X(\omega_k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi nk/N}$

$$\underbrace{\begin{bmatrix} X(\omega_0) \\ X(\omega_1) \\ X(\omega_2) \\ \vdots \\ X(\omega_{N-1}) \end{bmatrix}}_{\underline{X}} = \underbrace{\begin{bmatrix} \overline{S_0(0)} & \cdots & \overline{S_0[N-1)} \\ & & \vdots \\ S_{N-1}(0) & \cdots & S_{N-1}(N-1) \end{bmatrix}}_{S_N^*} \underbrace{\begin{bmatrix} x(0) \\ \vdots \\ x(N-1) \end{bmatrix}}_{x}$$

$$\underline{X} = S_N^* x = \begin{bmatrix} S_0^* \\ \\ S_{N-1}^* \end{bmatrix} x = \begin{bmatrix} \langle x, S_0 \rangle \\ \\ \langle x, S_{N-1} \rangle \end{bmatrix}$$

where S_N^* denotes the DFT matrix $S_N^*[K,n] = W_N^{-Kn} = e^{-j2\pi Kn/N}$

Also, the inverse DFT (IDFT) matrix is simply S_N/N (where $S_N^* S_N = NI$)

$$x = \frac{1}{N} S_N \underline{X}$$

b) $x[n] * h[n] = \sum x[K]\, h[n-K]$

$\underset{(1\times N)}{\downarrow}$ $\underset{(1\times M)}{\downarrow}$ result is $(1 \times M+N-1)$ matrix.

which is also placed to our book by Oppenheim.

c) $x[n] * h[n] = \sum_{k=-\infty}^{\infty} x_K\, h_{n-K} = \sum_{k=-\infty}^{\infty} h_K\, x_{n-K}$

$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-j\omega n}$, $H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h[n]\, e^{-j\omega n}$

$x_n * h_n = \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} x_K\, h_{n-K}\, e^{-j\omega n}$

$=$

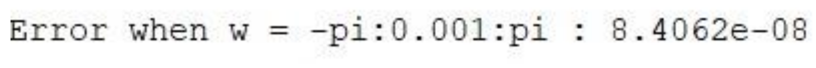
As can be seen the equations

$x_n * h_n = ~~y~~ H(e^{j\omega}) \cdot X(e^{j\omega})$

Part 2:

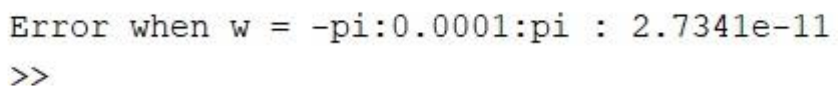
Part 2.1:

In this section, the DTFT and IDTFT functions were implemented as requested. E was computed using the formula given in the question, however, its value was found to be nonzero. Although there could be multiple reasons for this, possible significant reasons are as follows: When transitioning from the time domain to the frequency domain, transformations involve numerical approximation. Another reason is that we may not sample the signal sufficiently (as in continuous time). As seen in the figure1 and figure2, different values of w result in different E values. The two reasons mentioned above are the main reasons for E being nonzero.



```
Error when w = -pi:0.001:pi : 8.4062e-08
```

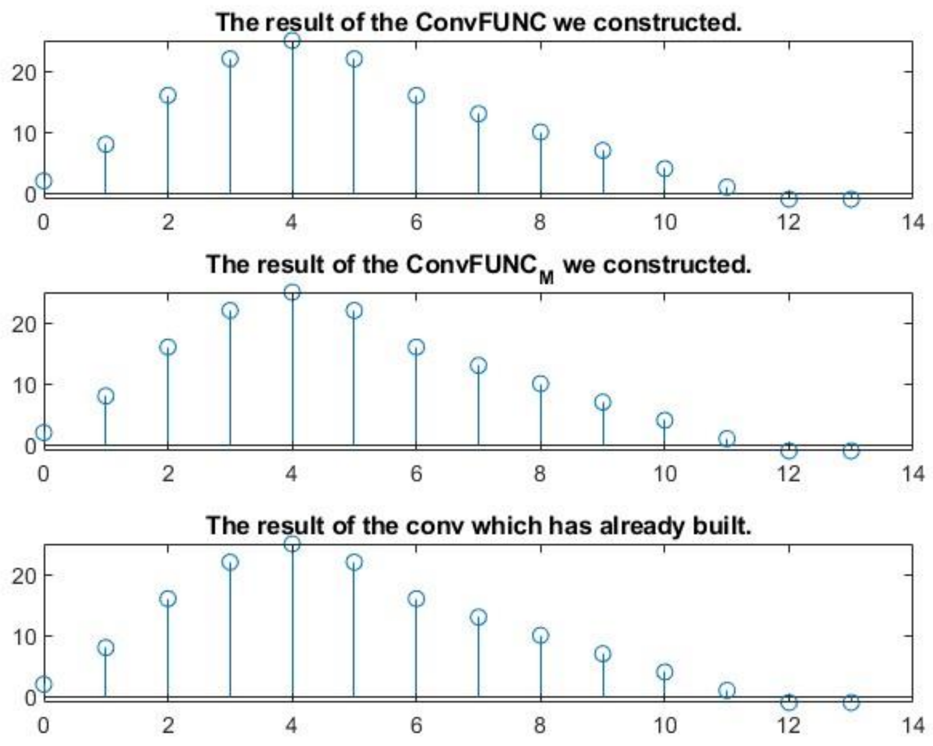
Figure 1 Error for w = -pi:0.001:pi

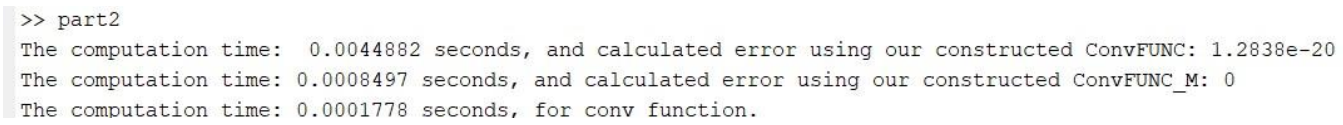


```
Error when w = -pi:0.0001:pi : 2.7341e-11
>>
```

Figure 2 Error for w = -pi:0.001:pi

Part 2.4:

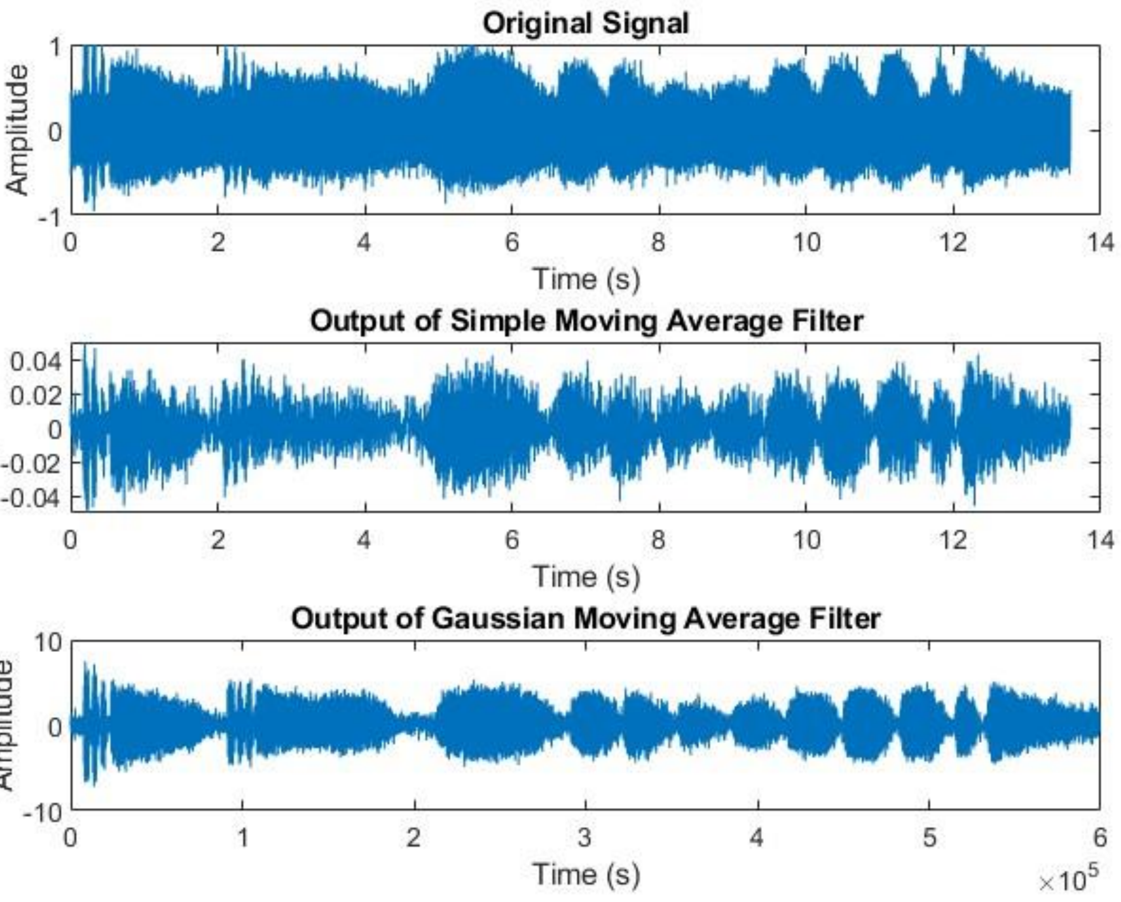


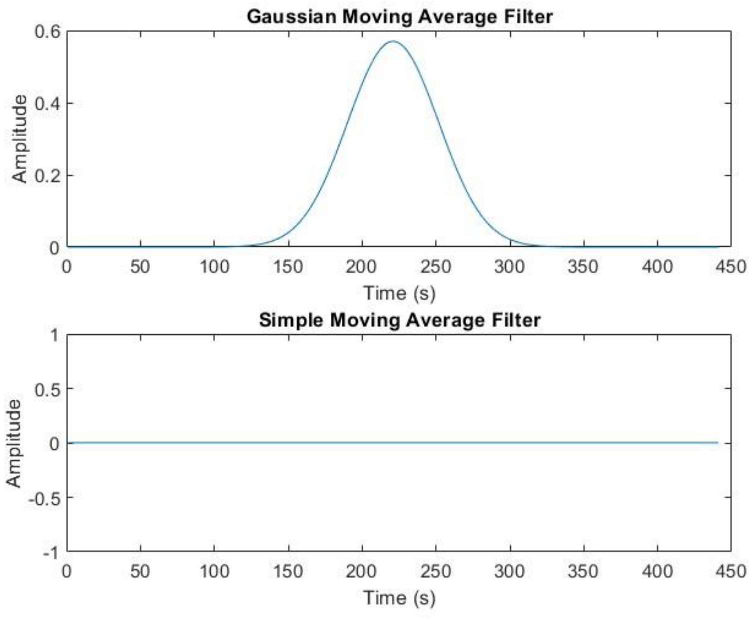


```
>> part2
The computation time:  0.0044882 seconds, and calculated error using our constructed ConvFUNC: 1.2838e-20
The computation time: 0.0008497 seconds, and calculated error using our constructed ConvFUNC_M: 0
The computation time: 0.0001778 seconds, for conv function.
```

Part 3:

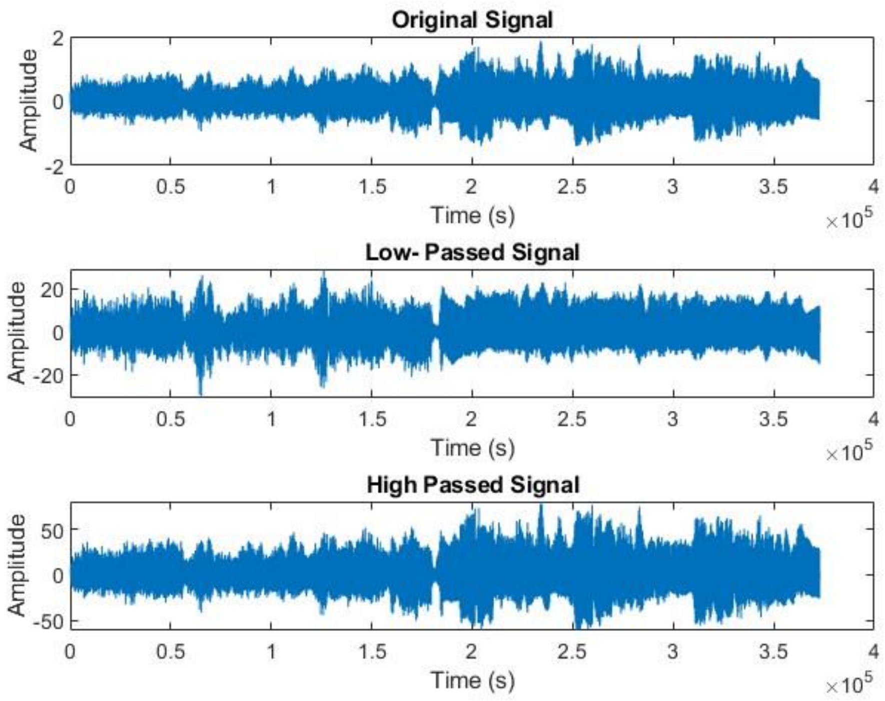
At this stage, two different filters were used. These filters were designed to eliminate the noise in the original sound. The first of these filters is the simple moving average filter, and the other is the Gaussian filter. Gaussian is more suitable for filtering in this sound. Because there are sudden spikes in the sound. Since the Gaussian filter is a nonlinear filter and keeps the weight closer to the center of the kernel, it is more successful in filtering the sound with sudden spikes. However, it should also be noted that the simple moving average filter also performed quite well.

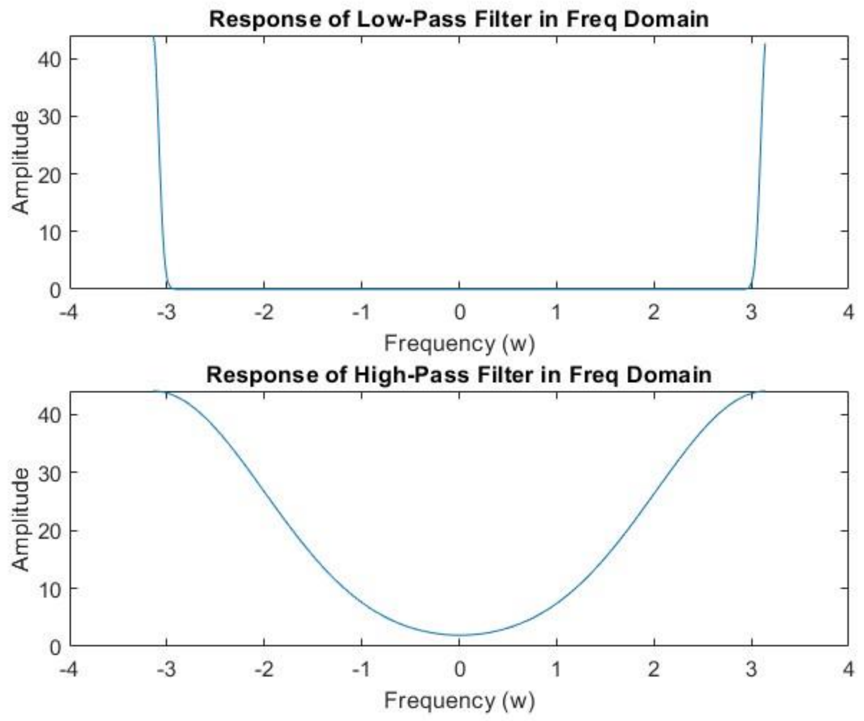

Original Signal
Amplitude
Time (s)
Output of Simple Moving Average Filter
Amplitude
Time (s)
Output of Gaussian Moving Average Filter
Amplitude
Time (s)
$\times 10^5$


Gaussian Moving Average Filter
Amplitude
Time (s)
Simple Moving Average Filter
Amplitude
Time (s)

Part 4:

In this section, a low-pass filter (LPF) and a high-pass filter (HPF) were applied to the orchestra. The filtering process was carried out using the respective impulse responses of each filter and performing convolution. LPF passes frequencies below a certain cutoff frequency, while HPF passes frequencies above it. LPF will attenuate high-frequency instruments such as flute, piccolo, and violin, but low-frequency instruments like double bass, tuba, and bassoon will be less affected. The situation will be the opposite for HPF. When HPF, LPF, and the original sound are mixed, a tone will emerge that is not unpleasant to the ear and where the weight is in the central frequencies, resulting in a pleasing sound to the listener.





Appendices:

Part 2:

```
boarder = 5;
n = 0:4;
firstsig = ones(1, boarder);
w = -pi:0.0001:pi
firstsigdtft = DTFT(firstsig, n, w);
recofirstsig = IDTFT(firstsigdtft, n, w);
recofirstsig = real(recofirstsig);
Errror = sum(abs(firstsig - recofirstsig).^2);

disp(['Error when w = -pi:0.0001:pi : ', num2str(Errror)]);
funcgiven1 = [2, 4, 6, 8, 7, 6, 5, 4, 3, 2, 1];  % These signals x1 and x2 are
given
funcgiven2 = [1, 2, 1, -1];        % and we gonna use to test our function
w = -pi :0.001 :pi;  % This vector also defined in report

tic;  % built function in matlab to control timer
resforilk = ConvFUNC(funcgiven1, funcgiven2, 0:length(funcgiven1)-1,
0:length(funcgiven2)-1, 0:length(funcgiven1)+length(funcgiven2)-2, w);
time1 = toc;  % finding the time1 to compare
tic;
resforiki = ConvFUNC_M(funcgiven1, funcgiven2);
time2 = toc;   % finding the time2 to compare
tic;
resforthree = conv(funcgiven1, funcgiven2);
time3 = toc;   % finding the time3 to compare
errror1 = norm(resforilk - resforthree, 2)^2;  % we are calculating the error for
ConvFUNC by comparing by conv
```

```
errror2 = norm(resforiki - resforthree, 2)^2;  % we are calculating the error for
ConvFUNC_M by comparing by conv
disp(['The computation time:  ', num2str(time1), ' seconds, and calculated error
using our constructed ConvFUNC: ', num2str(errror1)]); % we gonna print the error
and calculation time for ConvFUNC
disp(['The computation time: ', num2str(time2), ' seconds, and calculated error
using our constructed ConvFUNC_M: ', num2str(errror2)]); % we gonna print the
error and calculation time for ConvFUNC_M
disp(['The computation time: ', num2str(time3), ' seconds, for conv function.']);
% we gonna print the calculation time for conv
figure;
subplot(3,1,1);
stem(0:length(resforilk)-1, resforilk);
title('The result of the ConvFUNC we constructed.');
subplot(3,1,2);
stem(0:length(resforiki)-1, resforiki);
title('The result of the ConvFUNC_M we constructed.');
subplot(3,1,3);
stem(0:length(resforthree)-1, resforthree);
title('The result of the conv which has already built.');

function [y] = ConvFUNC(x,h,nx,nh,ny,w)
    y = IDTFT((DTFT(x,nx,w).*DTFT(h,nh,w)),ny,w); % we gonna calculate the
convolution by benefiting from our dtft and idtft (indirectly matrices)

end


function [x] = IDTFT(X,n,w)
    x = (X*(exp(1j*w'*n) )/(pi*2)) *(w(2)-w(1)); % we gonna calculate the reverse
foruier transform by benefiting from matrix

end


function [y] = ConvFUNC_M(x,h)
    y = sum(spdiags(x(end:-1:1)'*h)); % we gonna calculate the convolution by
benefiting from matrix

end


function [X] = DTFT(x,n,w)
    X =  x* exp(-1j*n'*w); % we gonna calculate the fourier transform by
benefiting from matrix

end
```

Part 3:

```
[y, Fs] = audioread('Part3_recording.flac');
% sound(y, Fs);
ts = 1/Fs
% Plot the original signal
t = (0:length(y)-1) / Fs;
figure;
subplot(3, 1, 1);
plot(t, y);
```

```
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Signal');
N = (0.01 * Fs); % Length of the filters
sigma = 0.7; % Standard deviation for Gaussian filter
mu = 0; % Mean for Gaussian filter

% Create simple moving average filter
hSMAV = ones(1, N) / N;

% Create Gaussian moving average filter
t_window = linspace(-5, 5, 441);
n =randn(1,length(t_window));
gaussian_kernel = exp(-t_window.^2 / (2 * sigma^2)) / (sqrt(2 * pi) * sigma);

% Normalize the kernel
% gaussian_kernel = gaussian_kernel / sum(gaussian_kernel);

% Apply filters using convolution

y_sma = conv(y, hSMAV, 'same');
y_gma = conv(y, gaussian_kernel, 'same');
sound(y_sma, Fs);
pause(12);
sound(y_gma,Fs);

% Play the first sound
% sound(y, Fs);
% % Wait for the first sound to finish playing
% pause(length(y)/Fs + 1); % Add extra second as a buffer
%
% % Play the second sound
% sound(y_sma, Fs);
% % Wait for the second sound to finish playing
% pause(length(y_sma)/Fs + 1);

% Play the third sound
% Wait for the third sound to finish playing
% pause(length(y_gma)/Fs + 1);


% Plot filtered signals
subplot(3, 1, 2);
plot(t, y_sma);
xlabel('Time (s)');
ylabel('Amplitude');
title('Output of Simple Moving Average Filter');

subplot(3, 1, 3);
plot( y_gma);
xlabel('Time (s)');
ylabel('Amplitude');
title('Output of Gaussian Moving Average Filter');

figure;
subplot(2,1,1)
plot(gaussian_kernel);
xlabel('Time (s)');
ylabel('Amplitude');
```

```
title('Gaussian Moving Average Filter');

subplot(2,1,2)
plot(hSMAV);
xlabel('Time (s)');
ylabel('Amplitude');
title('Simple Moving Average Filter');
```

Part 4:

```
N = (0.01 * Fs);
sigma = 0.7;
lpsigma = 0.4;
hpsigma = 0.02;
mu = 0;

% hSMAV = ones(1, N) / N;

t_window = linspace(-5, 5, 441);
n =randn(1,length(t_window));
lp_gaussian_kernel = exp(-t_window.^2 / (2 * lpsigma^2)) / (sqrt(2 * pi) *
lpsigma);
hp_gaussian_kernel = exp(-t_window.^2 / (2 * hpsigma^2)) / (sqrt(2 * pi) *
hpsigma);
w = linspace(-pi,pi,441);
H_LPF = fft(lp_gaussian_kernel);
H_HPF = fft(hp_gaussian_kernel);

figure;
subplot(2,1,1);
plot(w, abs(H_LPF));
xlabel('Frequency (w)');
ylabel('Amplitude');
title('Response of Low-Pass Filter in Freq Domain');
subplot(2,1,2);
plot(w, abs(H_HPF));
xlabel('Frequency (w)');
ylabel('Amplitude');
title('Response of High-Pass Filter in Freq Domain');



[k, Fs_bass] = audioread('bassoon.flac');
[m, Fs_flu] = audioread('flute.flac');
[km, Fs_cel] = audioread('cello.flac');
[ka, Fs_trum] = audioread('trumpet.flac');

% her sesi f domainde ciz
ses1 = fft(k);
ses2 = fft(m);
ses3 = fft(km);
ses4 = fft(ka);


ali = k + m + km+ ka;
ali1 = conv(ali,hp_gaussian_kernel);
ali2 = conv(ali,lp_gaussian_kernel);
```

```
 sound(ali, Fs);  % original music
 pause(length(ali)/Fs + 1);
 sound(ali2, Fs);  %  Low-Passed orchestra music
 pause(length(ali2)/Fs + 1);
sound(ali1, Fs);  % High-Passed orchestra music
 pause(length(ali1)/Fs + 1);

figure;
subplot(3,1,1);
plot( ali);
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Signal');
subplot(3,1,2);
plot( ali2);
xlabel('Time (s)');
ylabel('Amplitude');
title('Low- Passed Signal');

subplot(3,1,3);
plot( ali1);
xlabel('Time (s)');
ylabel('Amplitude');
title('High Passed Signal');
```

References:

https://ccrma.stanford.edu/~jos/mdft/Matrix_Formulation_DFT.html