Bilge Kaan Ateş

22102758

28.04.2024

**EEE321 SIGNALS AND SYSTEMS**

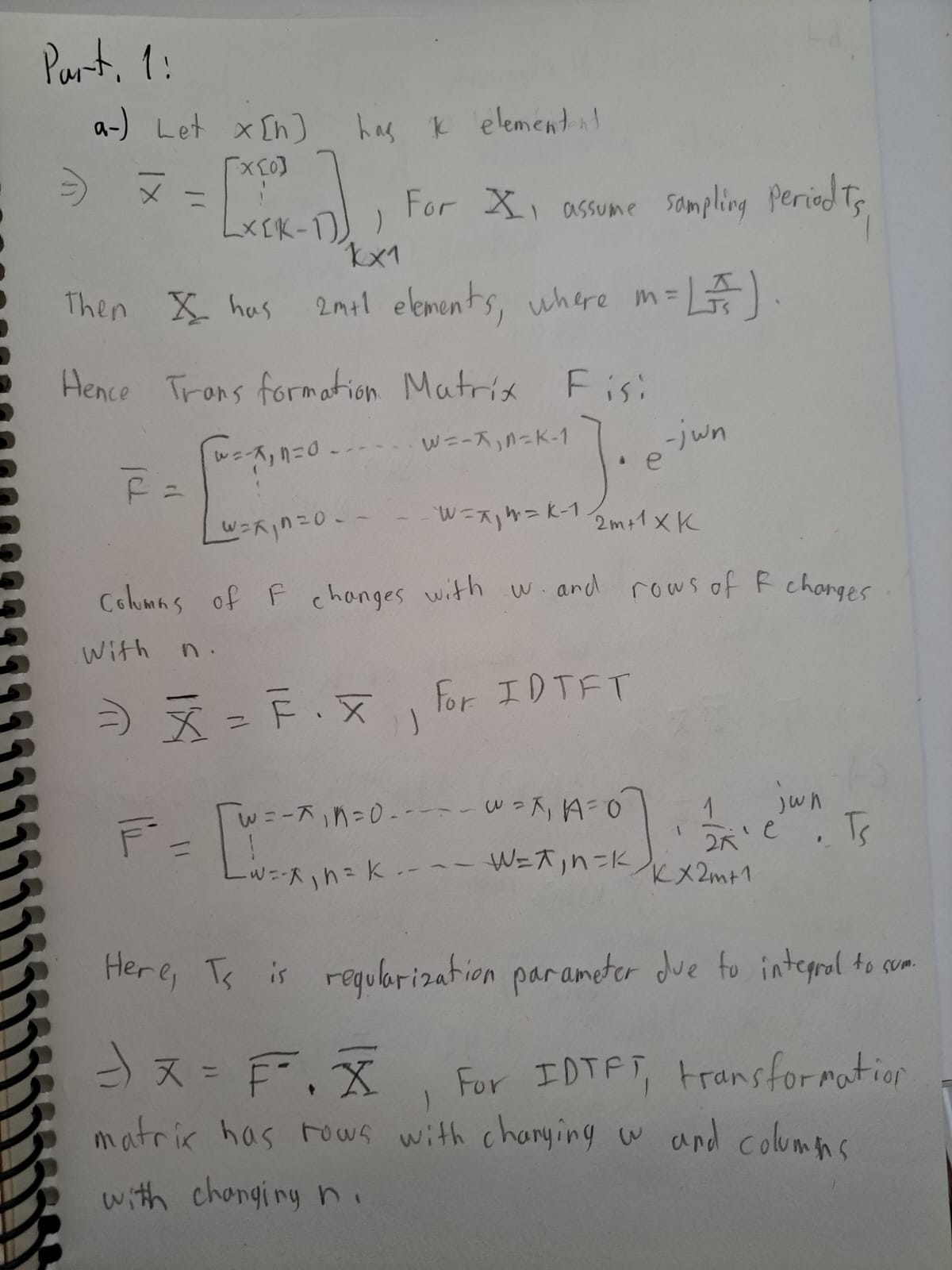
**LAB ASSIGNMENT 4**

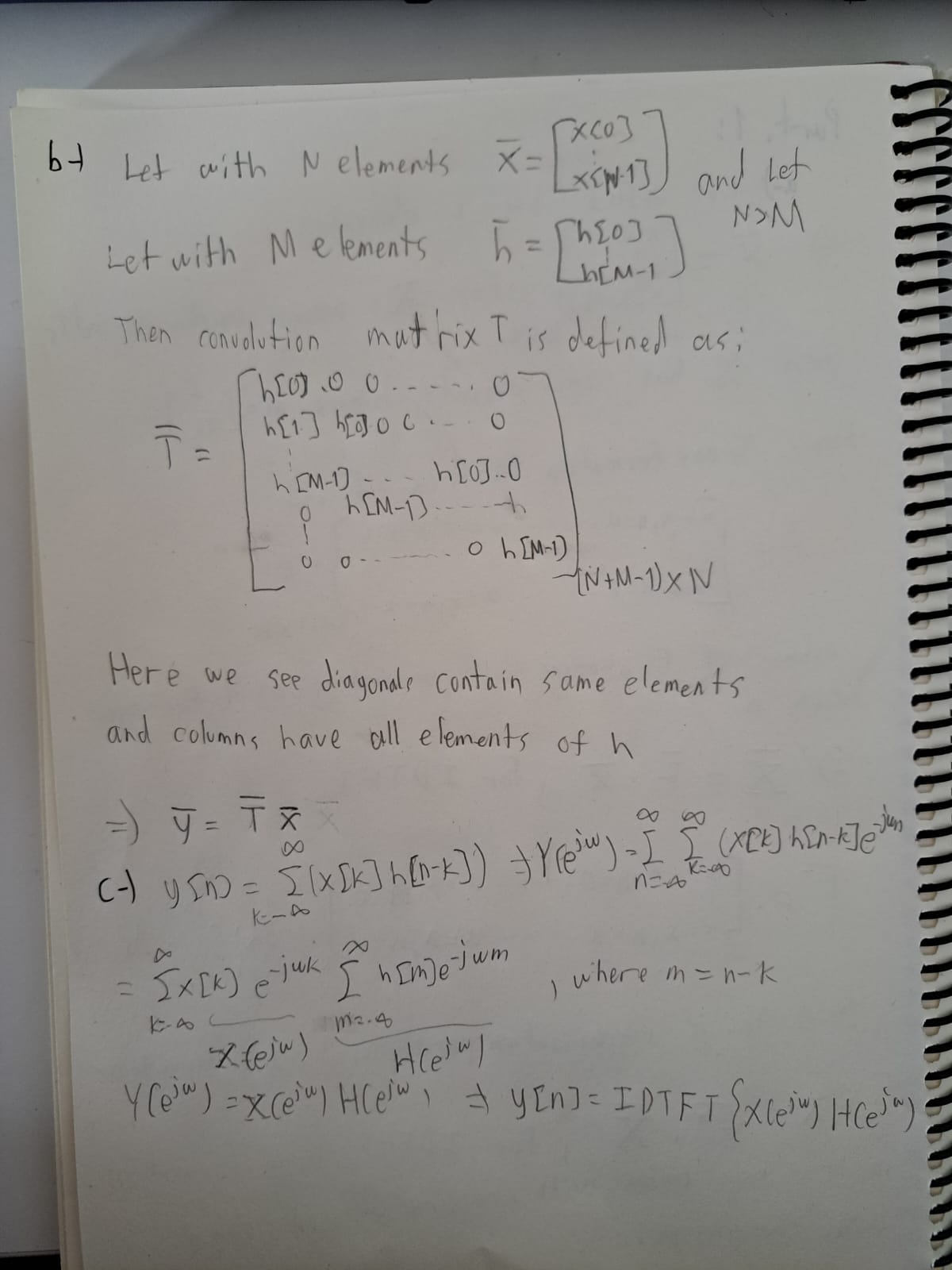
**1) INTRODUCTION:**

In this lab, I work with ideal/imperfect integrator systems and the properties of LTI systems such as linearity, causality, memory etc. Moreover, I analyzed BIBO stability of the systems as well. I studied the impulse and unit step response of the integrator systems. In first part, I manually handled the integrator systems and analyze their LTI properties and test them in MATLAB. In the second part, I write **sumElements()** function and analyzed the systems BIBO stability via Matlab. In the third part, I investigate the differences of ideal and imperfect (exponentially decaying) integrators. In the fourth part, firstly I derived second order difference equation and implement it in Matlab. Then, I found the inverse system of it by manually and implement it in Matlab and observed these systems convolution gives me impulse response.

**2) LAB:**

* Part 1:





* Part 2:

Part 2.1:

In this part, I created **DTFT** and **IDTFT** functions. Both of them contains the time array ***n*** and ***w*** which represents the time array and frequency array respectively. **DTFT** takes ***x*** as input which is time-domain signal array and **IDTFT** takes ***X*** as input which is frequency domain signal array. Their output is an array in frequency domain and time domain respectively. Important thing for these functions is the vector multiplication in the correct order to obtain corresponding array. I added normalization factor in **IDTFT** function since all the signals have .

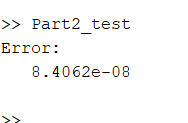


Fig. 1: Testing error

Test error as seen in Fig. 1 which shows the high success of the functions and approximations. I think error stems from and π match since frequency array cannot reach –π and π values due their irrationality.

You can see the corresponding code below:

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

X = (exp(-1j \* w.' \* n) \* x.').';

End

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

x = ((1/(2\*pi)) \* (exp(1j \* w.' \* n)).' \* X.').'/1000;

end

n = 0:4;

phi = double(n>=0 & n<5);

w = -pi:0.001:pi;

Phi = DTFT(phi,n,w);

phi\_ift=IDTFT(Phi,n,w);

E = norm(phi - phi\_ift)^2;

disp(E);

Part 2.2:

In this subsection, I created the specified **ConvFUNC** by obeying the restrictions. This function first converts the time-domain signals to frequency-domain signals and takes inverse transformation of the multiplied array in the frequency domain.

Following lines represent the code:

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

y = IDTFT(DTFT(x, nx, w) .\* DTFT(h, nh, w), ny, w);

end

Part 2.3:

In this part, I implemented the specified **ConvFUNC\_M** by obeying the restrictions. I implemented the derived matrix in the function and multiplied with the signal.

You can see the written code below:

function [y] = ConvFUNC\_M(x,h)

M = toeplitz([h(1) zeros(1, length(x) - 1)], [h zeros(1, length(x) - 1)]);

y = x\* M;

end

Part 2.4:

In this part, I tested my written functions with the built-in Matlab function **conv.** I defined the following arrays:

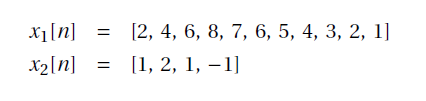


Fig. 2: Defined arrays

I calculated three different convolution results where two of them are written by me (**ConvFUNC** and **ConvFUNC\_M**) and calculated the time spent on during these functions with the **tic** and **toc** commands.

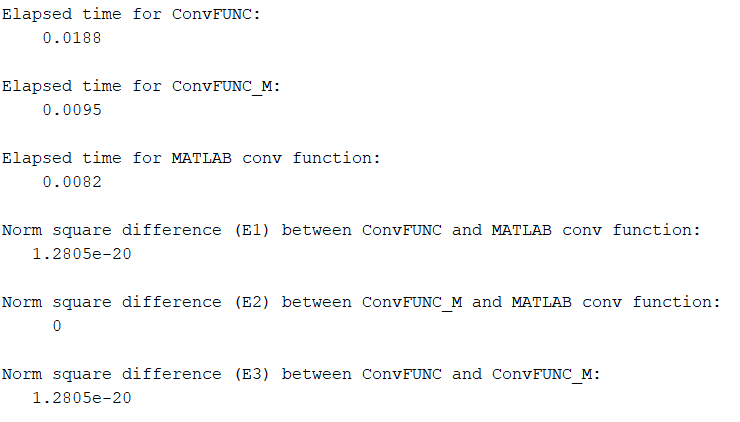


Fig. 3: Results

As seen in Fig. 3, **ConvFUNC\_M** consumes nearly same time with the built-in **conv** function and their results are exactly the same which shows the matrix-calculation efficiency and accuracy. However, there is some small error with the **ConvFUNC** and **conv**. This probably occurs due to frequency array with Ts distances doesn’t perfectly match with the boundaries –π and π. Also two times more time spent on **ConvFUNC**. This probably occurred due to **DTFT** and **IDTFT** operations.

* Part 3:

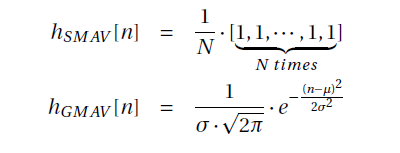


Fig. 4: Filters definition

I designed two different moving average filter and tested them on an noisy and anechoic horn recording in this part. I used the definitions in depicted Fig. 4 for the filters. SMAV filter calculates the average of a signal during definite length past periods. Hence, it removes randomly occurred noise in a simple fashion. While, GMAV filter utilize Gaussian weight for the previous signals to remove noise which means the closest points are more important. This method leads to smoother removal of the noise compared to SMAV. As I listened the filtered signals, I observed the theory on these sounds. GMAV filter maintains signals characteristics better than the SMAV filter. This smoothness can be observed visually in Fig. 5 as well.

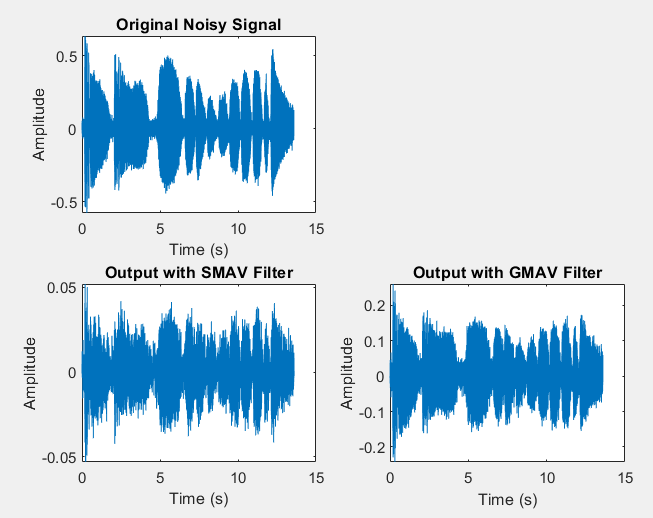


Fig. 5: Plots of the original and filtered signals

* Part 4:

In this part I designed two filters: high-pass and low-pass. I created them in frequency domain with Gaussian distributions. Filters plots can be seen in Fig. 6. Then I take fourier transform of the signals. Their plots are represented in Fig. 7.

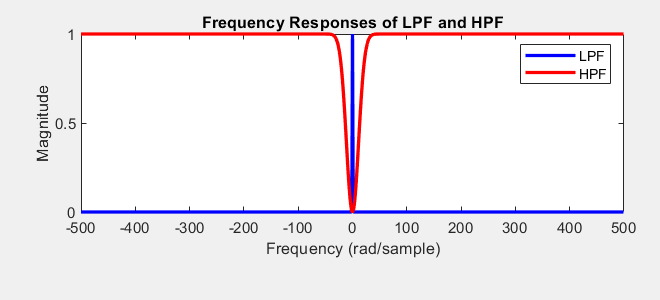


Fig. 6: Filters’ plots

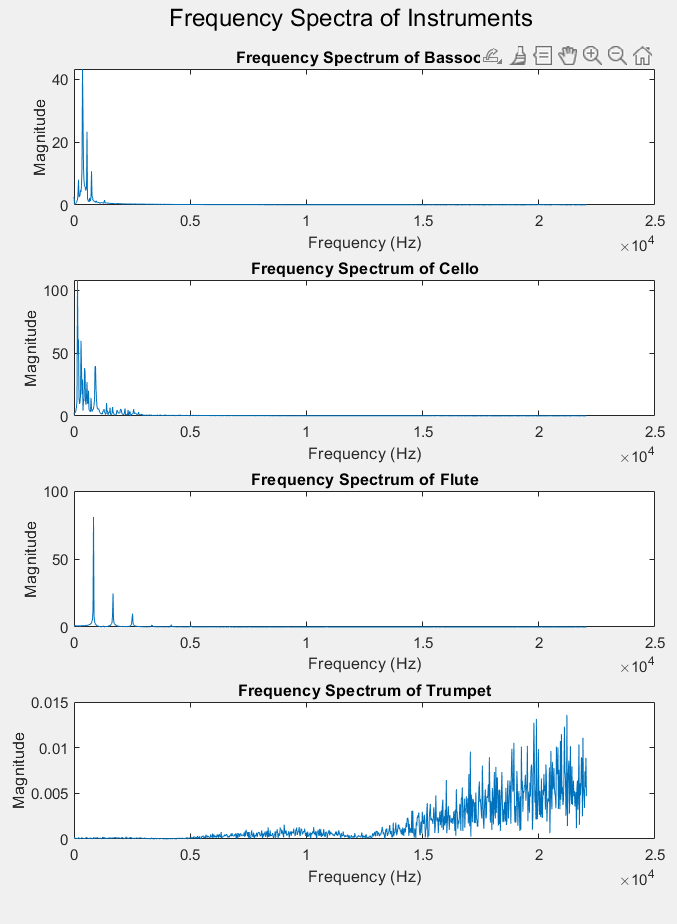


Fig. 6: Frequency Spectrum of the different instruments

Fig. 6 shows that bassoon, cello and flute operates in low frequencies while trumpet has high-frequency operation. After these analysis, I blend them and listened the orchestra sound. Then, I applied my filters on this blended sound array and listened them separately. As expected, with low-pass filter only trumpet voice is eliminated. However, with high-pass filter only trumpet voice is remained. This stems from the nature of instruments and their operating frequencies.

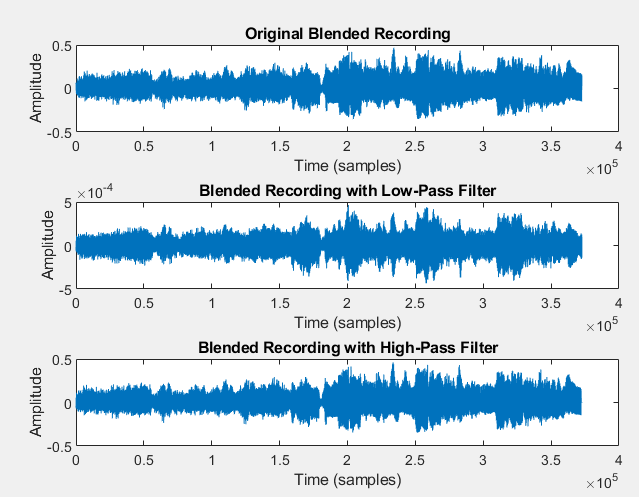


Fig. 7: Original and filtered signals

Related codes are presented below:

% Clear workspace and close all figures

clear;

close all;

% Load instrument recordings

[bassoon, Fs] = audioread('bassoon.flac');

[cello, ~] = audioread('cello.flac');

[flute, ~] = audioread('flute.flac');

[trumpet, ~] = audioread('trumpet.flac');

% Prepare for FFT

N = 2048; % Number of FFT points

f = linspace(0, Fs/2, N/2+1); % Frequency vector

% Calculate FFT for each instrument

Y\_bassoon = fft(bassoon, N);

Y\_cello = fft(cello, N);

Y\_flute = fft(flute, N);

Y\_trumpet = fft(trumpet, N);

% Plot frequency spectra for all instruments

figure;

subplot(4,1,1);

plot(f, abs(Y\_bassoon(1:N/2+1)));

title('Frequency Spectrum of Bassoon');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

subplot(4,1,2);

plot(f, abs(Y\_cello(1:N/2+1)));

title('Frequency Spectrum of Cello');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

subplot(4,1,3);

plot(f, abs(Y\_flute(1:N/2+1)));

title('Frequency Spectrum of Flute');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

subplot(4,1,4);

plot(f, abs(Y\_trumpet(1:N/2+1)));

title('Frequency Spectrum of Trumpet');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

% Optional: Enhance plot appearance

sgtitle('Frequency Spectra of Instruments'); % Super title for all subplots

set(gcf, 'Position', [100, 100, 600, 800]); % Adjust figure size

% Design low-pass and high-pass filters

N = round(0.01 \* Fs); % Filter length

sigmaLPF = 0.4;

sigmaHPF = 0.02;

range = -floor(N/2):floor(N/2);

N = 1000; % Number of points

Fxs = 1000; % Sampling frequency

% Define the frequency range

f = (-Fxs/2):(Fxs/N):(Fxs/2-Fxs/N); % Frequency range

% Compute the Gaussian function in the frequency domain

HLPF = exp(-(pi^2\*sigmaLPF^2\*f.^2));

HHPF = 1 - exp(-(pi^2\*sigmaHPF^2\*f.^2));

figure;

subplot(2, 1, 1);

plot(f, abs(HLPF), 'b', 'LineWidth', 2);

hold on;

plot(f, abs(HHPF), 'r', 'LineWidth', 2);

xlabel('Frequency (rad/sample)');

ylabel('Magnitude');

title('Frequency Responses of LPF and HPF');

legend('LPF', 'HPF');

% Blend the recordings

blended = (bassoon + cello + flute + trumpet) / 4;

HLPF = ifft(HLPF);

HHPF = ifft(HHPF);

% Apply filters

blendedLPF = conv(blended, HLPF, 'same');

blendedHPF = conv(blended, HHPF, 'same');

% Plot original and filtered blended recordings

figure;

subplot(3,1,1);

plot(blended);

title('Original Blended Recording');

xlabel('Time (samples)');

ylabel('Amplitude');

subplot(3,1,2);

plot(blendedLPF);

title('Blended Recording with Low-Pass Filter');

xlabel('Time (samples)');

ylabel('Amplitude');

subplot(3,1,3);

plot(blendedHPF);

title('Blended Recording with High-Pass Filter');

xlabel('Time (samples)');

ylabel('Amplitude');

% Play the original and filtered sounds for auditory comparison

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);