Bilkent University

Fall 2024-2025

EEE 391 Basics of Signals and Systems MATLAB Assignment 1

Musical Signal Analysis and Synthesis

Deniz Tuna ONGUNER

Department of Computer Engineering

Student ID: 22001788

Section: 3

November 15, 2024

List of Figures

Contents

Introduction

The objective of this report is to document, demonstrate, and analyze the use of MATLAB in creating, manipulating, and enriching musical signals through sinusoidal synthesis and harmonic addition. Throughout the assignment, I have worked with the notes **F#**, **A**, and **C#**, in 4th octave, to construct the **F#m** chord, as these notes were selected based on my Bilkent ID number, as stated in the assignment document, which is 22001788. The MATLAB code used for this project is provided under Appendix A, and the plots generated during the analysis can be found in Appendix B. It is important to note that I have no training or background in music, and my approach to this task is grounded purely in the application of signal processing techniques rather than musical expertise.

Implementation

Part 1: Generating the F#m Chord

In the first part of the assignment, I have generated the F#m chord with the given amplitude of 1 and phase of 0. The frequencies for the notes F#, A, and C# in the 4th octave were used to create the individual sinusoidal signals. These signals were then combined to form the F#m chord, which was played using the MATLAB's sound function. The individual notes and the chord were plotted using the plot function. The code for this part can be found in Listing 1 of Appendix A, and the generated plots are shown in Figure 1.

Part 2: Customizing the Chord

In the second part of the assignment, I have customized the F#m chord by adjusting the amplitude and phase values based on my Bilkent ID number, which is 22001788. Hence, the values 7, 8, and 8 were used as the new amplitudes, respectively. The phase values were also determined based on the ID digits as 17, 178, and 788 degrees, respectively. The custom notes were then combined to form the custom F#m chord. The code for this part can be found in Listing 2 of Appendix A, and the generated plots are shown in Figure 2.

Part 3: Adding Harmonics to the Chord

In the third part of the assignment, I have added harmonics to the F#m chord. The fundamental frequency of each note was combined with the 2nd, 3rd, and 4th harmonics to enrich the sound. The code for this part can be found in Listing 3 of Appendix A, and the generated plots are shown in Figure 3.

Part 4: Fourier Transform of the Enriched Chord

In the fourth part of the assignment, I have computed the Fourier Transform of the enriched F#m chord with harmonics. The magnitude spectrum of the chord was plotted to visualize the frequency components present in the signal. The code for this part can be found in Listing 4 of Appendix A, and the generated plot is shown in Figure 4.

Part 5: Reducing Sampling Frequency and Interpolating the Signal

In the fifth part of the assignment, I have reduced the sampling frequency of the enriched F#m chord to a value lower than or equal to two times the frequency of the highest note, which is A with a frequency of 440 Hz. I have chosen a reduced sampling frequency of 800 Hz for simplicity. The chord was then generated at the reduced sampling frequency and upsampled to the original sampling rate. The code for this part can be found in Listing 5 of Appendix A, and the generated plots are shown in Figure 5.

Results and Discussion

In the first part of the assignment, although the individual notes sound good, the chord sounds like a horn. This is because all the notes are played at the same time, and the frequencies are not harmonically related. It is also important to note that MATLAB's sound function does not normalize audio by default [1].

In the second part, the custom F#m chord sounds even worse than the original one. This might be due to the fact that the amplitudes and phases were chosen arbitrarily based on the ID number, which does not guarantee a harmonious sound, and the last three digits of my ID number are either the same or too close to one other; 7, 8, and 8, respectively.

In the third part, however, the F#m chord with added harmonics sounds better than the previous versions, and it reflects the expected sound of a F#m chord more closely. This is because the harmonics enrich the sound and make it more close to a real musical chord.

In the fourth part, the Fourier Transform of the enriched F#m chord with harmonics shows the frequency components present in the signal (Refer to Appendix B, Figure 4). The peaks in the spectrum correspond to the fundamental frequencies and harmonics of the notes in the chord.

In the fifth part, the enriched F#m chord was downsampled to a reduced sampling frequency of 800 Hz and then upsampled back to the original sampling rate of 8000 Hz. The sound of the downsampled chord is aliased due to the reduced sampling frequency, and the upsampled chord sounds similar to the original chord. The magnitude spectrum of the downsampled chord shows the aliasing effects, while the upsampled chord retains the original frequency components. The plots for the downsampled and upsampled chords can be found in Figure 5 of Appendix B.

In conclusion, this very assignment has provided a hands-on experience with musical signal analysis and synthesis using MATLAB. The results demonstrate the importance of harmonics in creating a rich and harmonious sound and the effects of sampling frequency on the quality of the audio signals, specifically in terms of notes and chords.

References

1. Play Audio Data at default sample rate (no date) MathWorks. Available at: https://www.mathworks.com/help/matlab/ref/sound.html (Accessed: 15 November 2024).

Appendix A

Listing 1: Generating F#m Chord and Plotting Notes

```matlab
% @author: Deniz Tuna ONGUNER
% MATLAB version: R2024b
% This code is written for the first MATLAB assignment of EEE 391 course.
% Part 1: Generating the F#m chord with amplitude=1 and phase=0
FREQ_SAMPLING = 8000; % as given in the assignment document
t = 0:1/FREQ_SAMPLING:1; % 1 second duration, also as given

% Frequencies for F#, A, and C# in the 4th octave
FREQ_F_SHARP = 370; % F#
FREQ_A = 440; % A
FREQ_C_SHARP = 277; % C#

% Declaring initial sinusoidal signals
AMP = 1; % as recommended in the assignment document
PHI_deg = 0;

% Generating individual signals with initial amplitude and phase
note1 = AMP * sin(2 * pi * FREQ_F_SHARP * t + deg2rad(PHI_deg)); % F#
note2 = AMP * sin(2 * pi * FREQ_A * t + deg2rad(PHI_deg)); % A
note3 = AMP * sin(2 * pi * FREQ_C_SHARP * t + deg2rad(PHI_deg)); % C#

% Playing individual notes
sound(note1, FREQ_SAMPLING); pause(1.5);
sound(note2, FREQ_SAMPLING); pause(1.5);
sound(note3, FREQ_SAMPLING); pause(1.5);

% Combining the notes to form the chord F#m, and playing it
chord = note1 + note2 + note3;
sound(chord, FREQ_SAMPLING);

% Plotting individual notes and the chord
figure;

subplot(4,1,1); plot(t, note1); % Plotting the F# note
title('F# Note'); xlim([0 0.02]);

subplot(4,1,2); plot(t, note2); % Plotting the A note
title('A Note'); xlim([0 0.02]);

subplot(4,1,3); plot(t, note3); % Plotting the C# note
title('C# Note'); xlim([0 0.02]);

subplot(4,1,4); plot(t, chord); % Plotting the F#m chord
title('F#m Chord'); xlim([0 0.02]);
```

Listing 2: Generating Custom F#m Chord and Plotting Notes

```matlab
% @author: Deniz Tuna ONGUNER
% Part 2: Customizing the chord with amplitude and phase values by ID digits
FREQ_SAMPLING = 8000;
t = 0:1/FREQ_SAMPLING:1;

% Frequencies for F#, A, and C# in the 4th octave
FREQ_F_SHARP = 370; % F#
FREQ_A = 440; % A
FREQ_C_SHARP = 277; % C#

% Amplitudes from ID digits, my ID is 22001788
A1 = 7 ; A2 = 8 ; A3 = 8 ;

% Phase values from ID digits in degrees
% d4d5d6 = 017, d5d6d7 = 178, d6d7d8 = 788
phi_d4d5d6_deg = 017;
phi_d5d6d7_deg = 178;
phi_d6d7d8_deg = 788;

% Converting degrees to radians for the sine function
phi_d4d5d6_rad = deg2rad(phi_d4d5d6_deg);
phi_d5d6d7_rad = deg2rad(phi_d5d6d7_deg);
phi_d6d7d8_rad = deg2rad(phi_d6d7d8_deg);

% Generating signals with custom amplitudes and phases
note1_prime = A1 * sin(2 * pi * FREQ_F_SHARP * t + phi_d4d5d6_rad);
note2_prime = A2 * sin(2 * pi * FREQ_A * t + phi_d5d6d7_rad);
note3_prime = A3 * sin(2 * pi * FREQ_C_SHARP * t + phi_d6d7d8_rad);

% Combined chord with custom settings
chord_prime = note1_prime + note2_prime + note3_prime;
sound(chord_prime, FREQ_SAMPLING);

% Plotting custom individual notes and the chord
figure;

subplot(4,1,1); plot(t, note1_prime); % Plotting the custom F# note
title('Custom F# Note'); xlim([0 0.02]);

subplot(4,1,2); plot(t, note2_prime); % Plotting the custom A note
title('Custom A Note'); xlim([0 0.02]);

subplot(4,1,3); plot(t, note3_prime); % Plotting the custom C# note
title('Custom C# Note'); xlim([0 0.02]);

subplot(4,1,4); plot(t, chord_prime); % Plotting the custom F#m chord
title('Custom F#m Chord'); xlim([0 0.02]);
```

Listing 3: Generating Custom F#m Chord and Plotting Notes

```matlab
% @author: Deniz Tuna ONGUNER
% Part 3: Adding harmonics to the chord
FREQ_SAMPLING = 8000; % Hz
t = 0:1/FREQ_SAMPLING:1; % 1 second duration

% Frequencies for F#, A, and C# in the 4th octave
FREQ_F_SHARP = 370; % F#
FREQ_A = 440;       % A
FREQ_C_SHARP = 277; % C#

% Amplitude weights for harmonics
AMP = 1; % Fundamental amplitude
AMP_HARMONICS = [0.5, 0.25, 0.125]; % Amplitudes for 2f, 3f, 4f harmonics

% Function to generate a signal with fundamental and harmonic components
% Inputs:
%   signal - Fundamental frequency of the signal (in Hz).
%   AMP - Amplitude of the fundamental frequency component.
%   AMP_HARMONICS - Array containing amplitudes for the harmonics.
%   t - Time vector for signal generation.
% Output:
%   Signal with fundamental and specified harmonics.
add_harmonics = @(signal, t) AMP * sin(2 * pi * signal * t) + ...
                      AMP_HARMONICS(1) * sin(2 * pi * 2 * signal * t) + ...
                      AMP_HARMONICS(2) * sin(2 * pi * 3 * signal * t) + ...
                      AMP_HARMONICS(3) * sin(2 * pi * 4 * signal * t);

% Generate signals with harmonics
note1_with_harmonics = add_harmonics(FREQ_F_SHARP, t);
note2_with_harmonics = add_harmonics(FREQ_A, t);
note3_with_harmonics = add_harmonics(FREQ_C_SHARP, t);

% Combine harmonics to form the chord
chord_with_harmonics = note1_with_harmonics + ...
                     note2_with_harmonics + ...
                     note3_with_harmonics;

% Play the notes and the chord with harmonics
sound(note1_with_harmonics, FREQ_SAMPLING); pause(1.5);
sound(note2_with_harmonics, FREQ_SAMPLING); pause(1.5);
sound(note3_with_harmonics, FREQ_SAMPLING); pause(1.5);
sound(chord_with_harmonics, FREQ_SAMPLING);

% The plotting implementation has been omitted for the sake of brevity.
```

Listing 4: Fourier Transform of the Enriched Chord of F#m

```matlab
% This code below is the continuation of the previous code
%   snippet in Listing 3 of Appendix A (see on the previous page).
% Which is why the code is not repeated here; please,
%   assume that all the necessary variables/constants are defined.

% @author: Deniz Tuna ONGUNER
% Part 4:

% Fourier Transform of the chord with harmonics
chord_length = length(chord_with_harmonics); % Number of samples
fft_chord = fft(chord_with_harmonics); % Compute Fourier Transform
magnitudes = abs(fft_chord/chord_length); % Normalize the magnitude

% Frequency vector for plotting
freq_p = (0:chord_length/2-1) * (FREQ_SAMPLING/chord_length);
magnitudes = magnitudes(1:chord_length/2);

% Plot the spectrum
figure;

plot(freq_p, magnitudes);
title('Magnitude Spectrum of F#m Chord with Harmonics');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;
```

Listing 5: Reducing Sampling Frequency and Interpolating the Signal

```matlab
% This code below is the continuation of the previous code
%   snippet in Listing 4 of Appendix A (see on the previous page).
% Which is why the code is not repeated here; please,
%   assume that all the necessary variables/constants are defined.

% @author: Deniz Tuna ONGUNER
% Part 5:

% Highest frequency, in 4th octave, is A with the value of 440 Hz
% Hence, the reduced sampling shall be lower than or equal to two times A
% Frequency Sampling Reduced <= 2 * 440 = 880
REDUCED_FREQ_SAMPLING = 800; % I have chosen 800 for simplicity
t_low = 0:1/REDUCED_FREQ_SAMPLING:1;

% Generate the chord at the reduced sampling frequency
note1_low = add_harmonics(FREQ_F_SHARP, t_low);
note2_low = add_harmonics(FREQ_A, t_low);
note3_low = add_harmonics(FREQ_C_SHARP, t_low);
chord_low = note1_low + note2_low + note3_low;

% Upsample the aliased signal to the original sampling rate
t_high = 0:1/FREQ_SAMPLING:1;
chord_upsampled = interp1(t_low, chord_low, t_high, 'linear');

% Aliased chord spectrum
chord_length_fs_low = length(chord_low);
fft_chord_low = fft(chord_low);
magnitudes_low = abs(fft_chord_low/chord_length_fs_low);
freq_low = (0:chord_length_fs_low/2-1) * ...
    (REDUCED_FREQ_SAMPLING/chord_length_fs_low);
magnitudes_low = magnitudes_low(1:chord_length_fs_low/2);

sound(chord_low, FREQ_SAMPLING); pause(1.5);

% Upsampled chord spectrum
chord_length_fs_high = length(chord_upsampled);
fft_chord_high = fft(chord_upsampled);
magnitudes_high = abs(fft_chord_high/chord_length_fs_high);
freq_high = (0:chord_length_fs_high/2-1) * ...
    (FREQ_SAMPLING/chord_length_fs_high);
magnitudes_high = magnitudes_high(1:chord_length_fs_high/2);

% Listen to the upsampled aliased signal
sound(chord_upsampled, FREQ_SAMPLING);

% The plotting implementation has been omitted for the sake of brevity.
% Be aware that MATLAB will throw a warning as follows:
% Warning: Integer operands are required for colon operator when used as index.
```

Appendix B

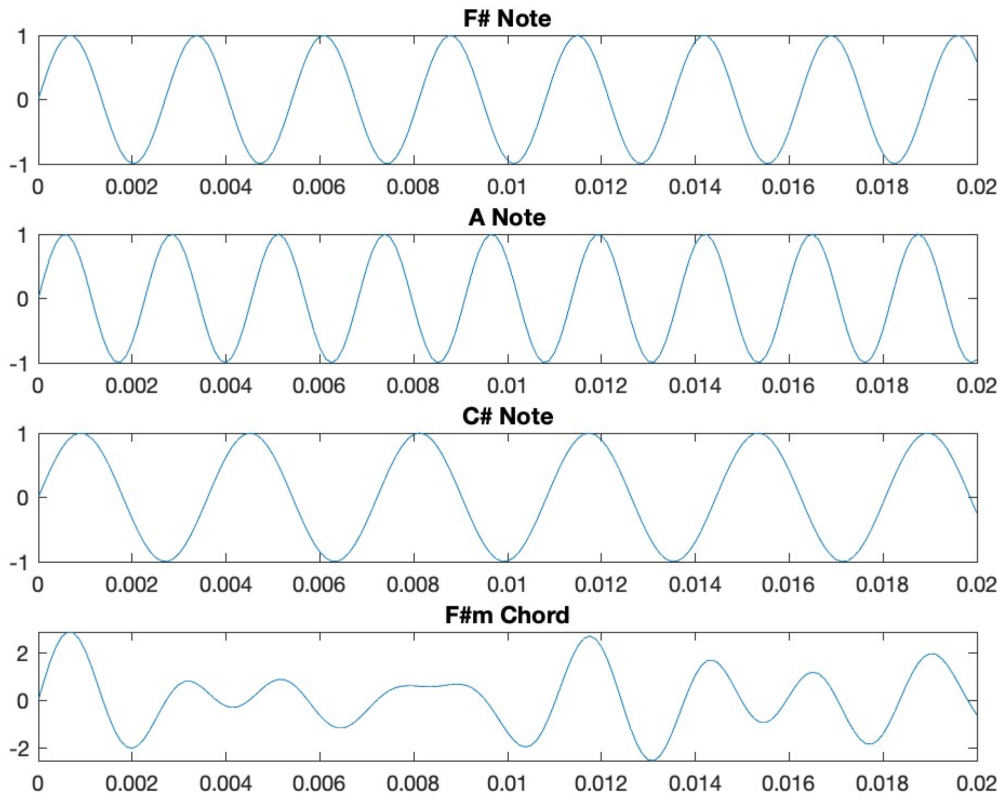

Figure 1: Generated signals for F#, A, C# notes and F#m chord

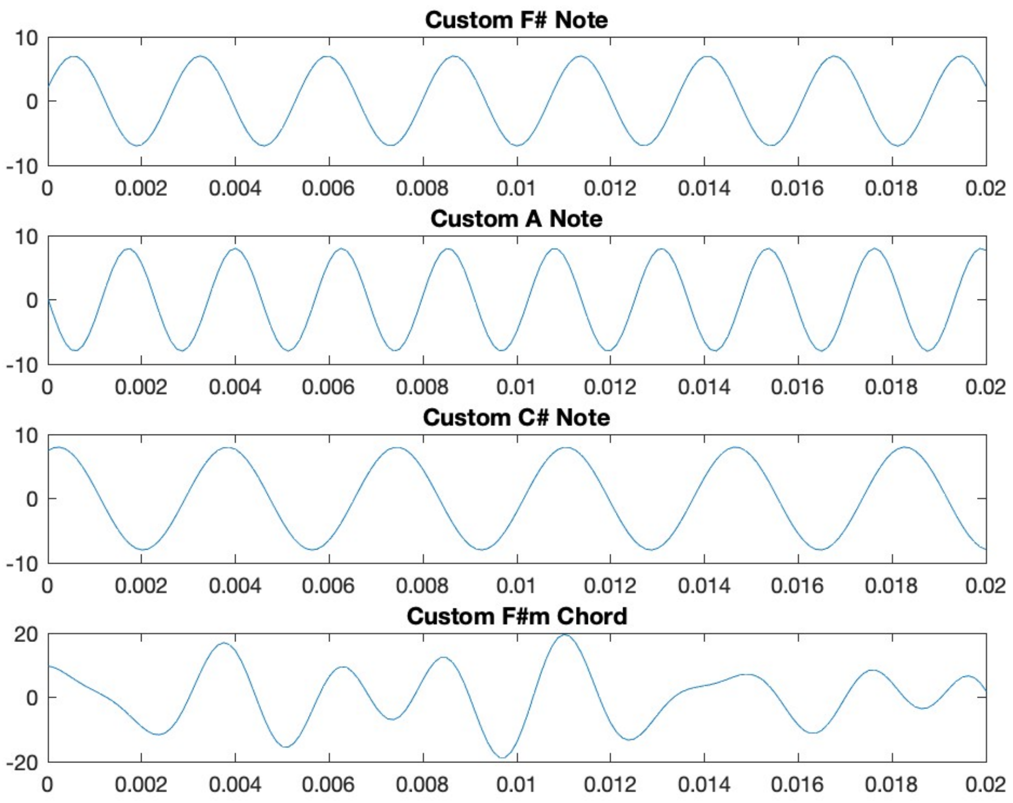

Figure 2: Generated signals for F#, A, C# notes and F#m with adjusted amplitudes

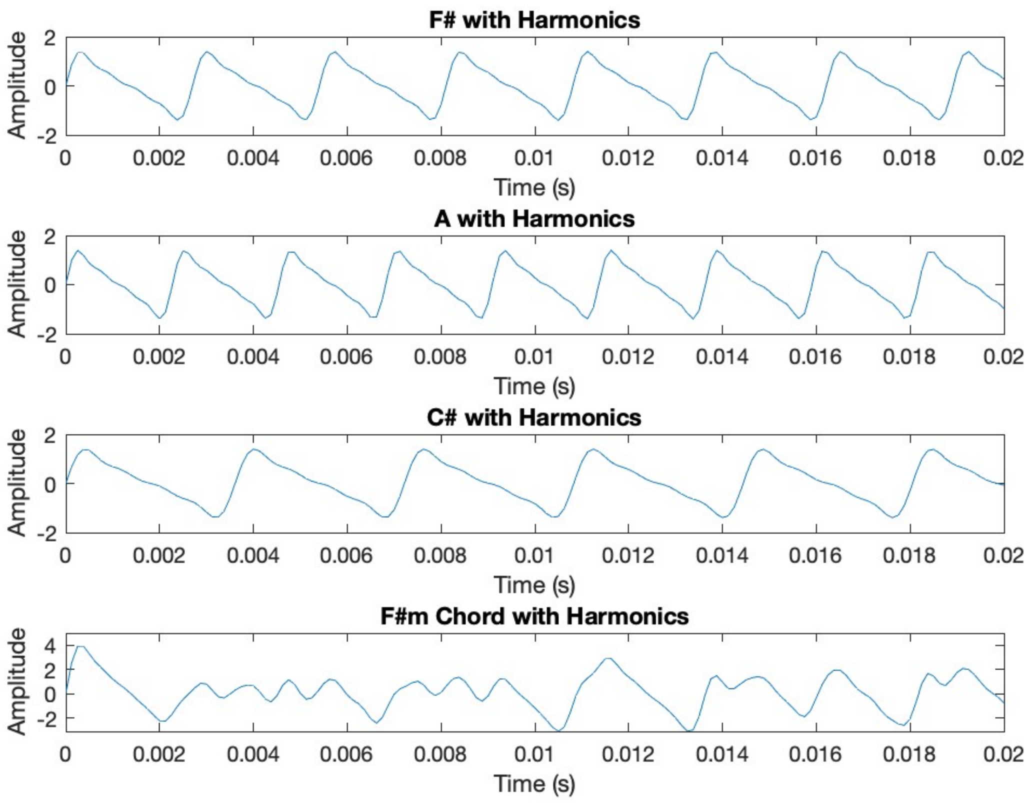

Figure 3: Generated signals for F#, A, C# notes and F#m with added harmonics

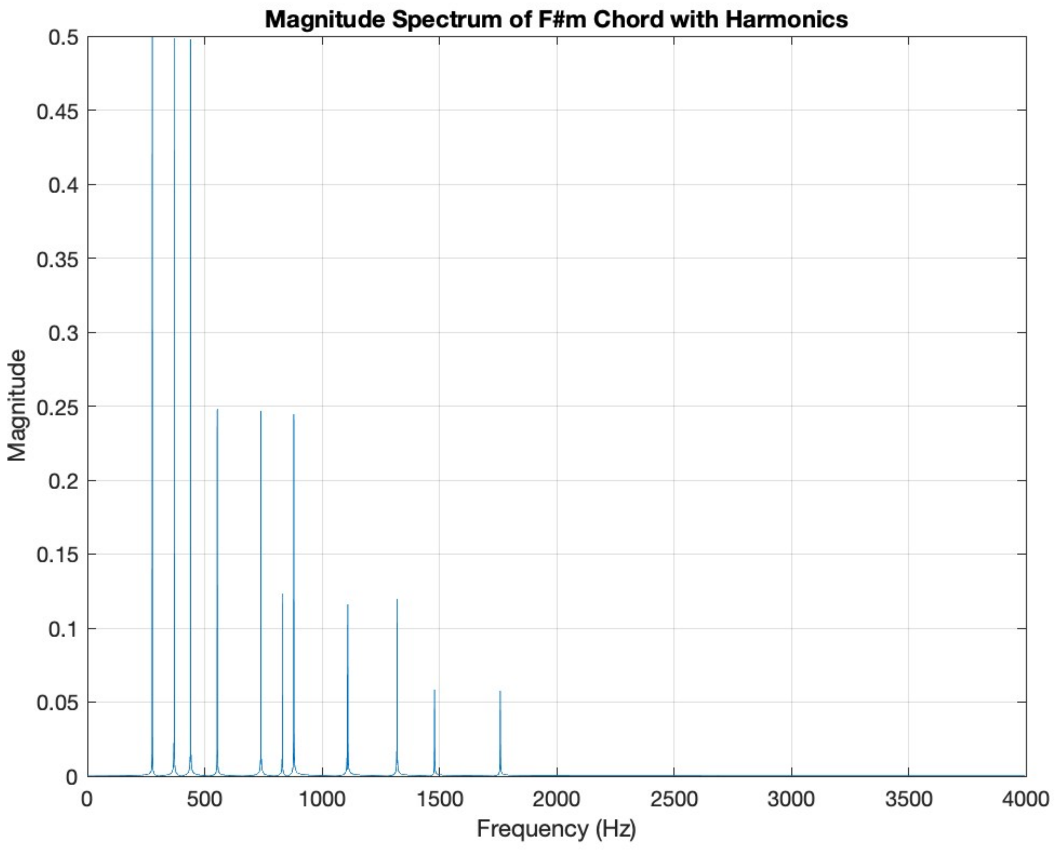

Figure 4: Magnitude spectrum of the enriched F#m chord with harmonics

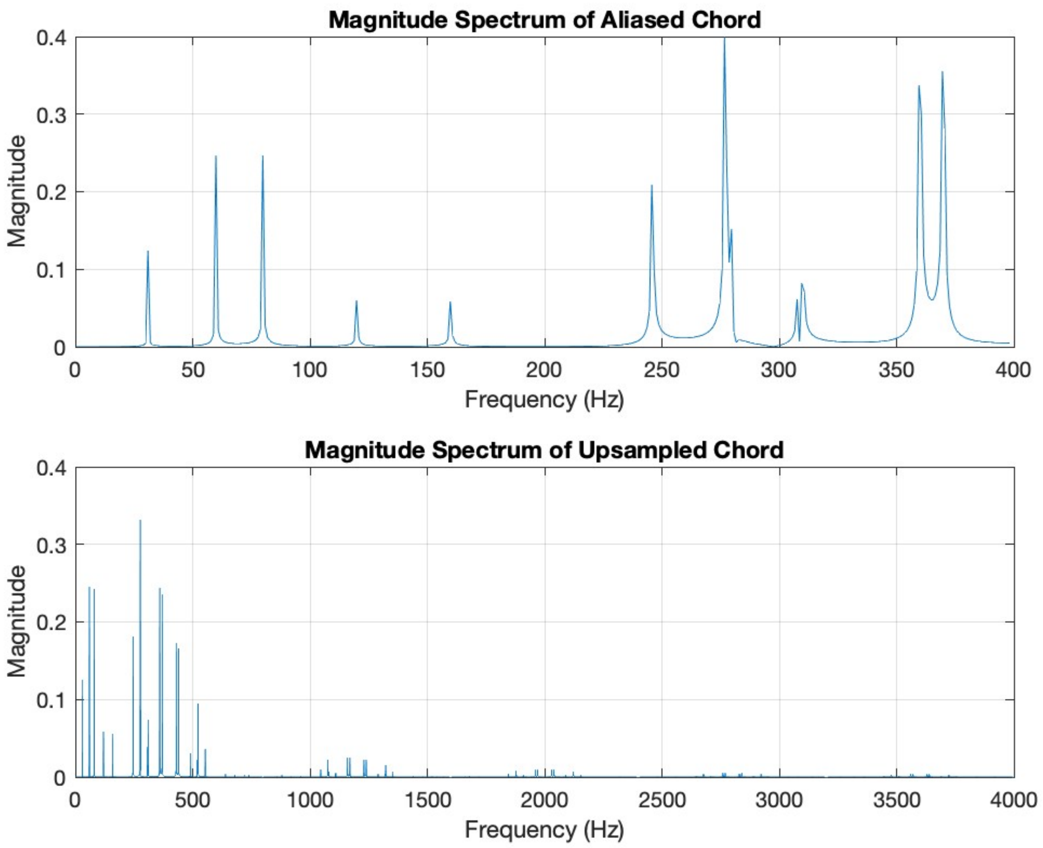

Figure 5: Magnitude spectrums of the enriched F#m chord with reduced sampling frequency and the upsampled signal