Furkan Büyüksarıkulak

22002097

16.04.2025

EEE 321

Signals and Systems

Spring 2024-2025

Lab Assignment 5

In this lab task we observed the definition of Fourier Series and itc Convregence. We examined how to identify the frequency of the signal. We also decompose music into notes and add corruption to signal. Finally we filter the sound signal by using FIR filter.

Part 1: Fourier Series and its Convergence

In this part we apply the one of the Dirchlet conditions fort he given functions. The condition is for any finite interval of time, the number of discontinioutiesof x(t) must be finite. Even if this signal has discontiniouties iy should be expressed as weihghted sum of harmonics so it must be countinious wrt. t

First we compute Fourier series expansion for

$$x(t) = \cos\left(2t + \frac{\pi}{4}\right)$$

Then we plot the expansion with the original signal in the same plot and we get the result as

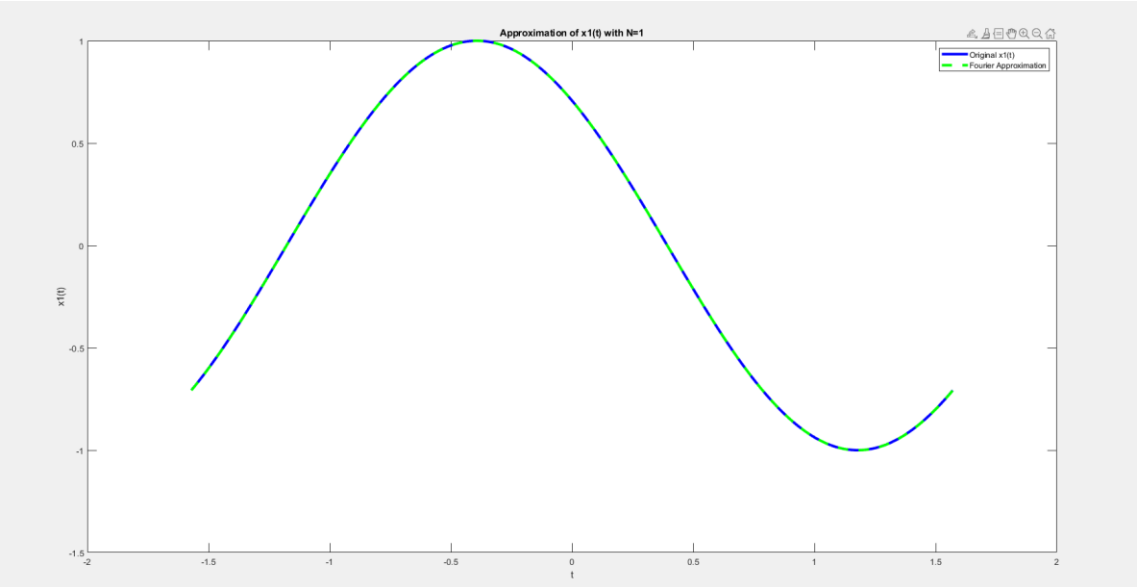


figure 1 : Approximation of x1(t) with N=1

We observe that Fourier approxiamtion overlap with the original signal. Secondly,we have the signal of

$$x(t) = \begin{cases} 1, & \text{for } t \in [2n, 2n+1] \text{ and } n \in \mathbb{Z} \\ 0, & \text{otherwise.} \end{cases}$$

Then we calculat the Fourier series expansion of that signal for n = (“1,10,100”). Then we compare the values of 10 and 100 with original signal. We get the result and plot of the signal as.

```
Max error between x(t) and x_{10}(t): 0.4955
Max error between x(t) and x_{100}(t): 0.4505
```

figure 2 : Max error results

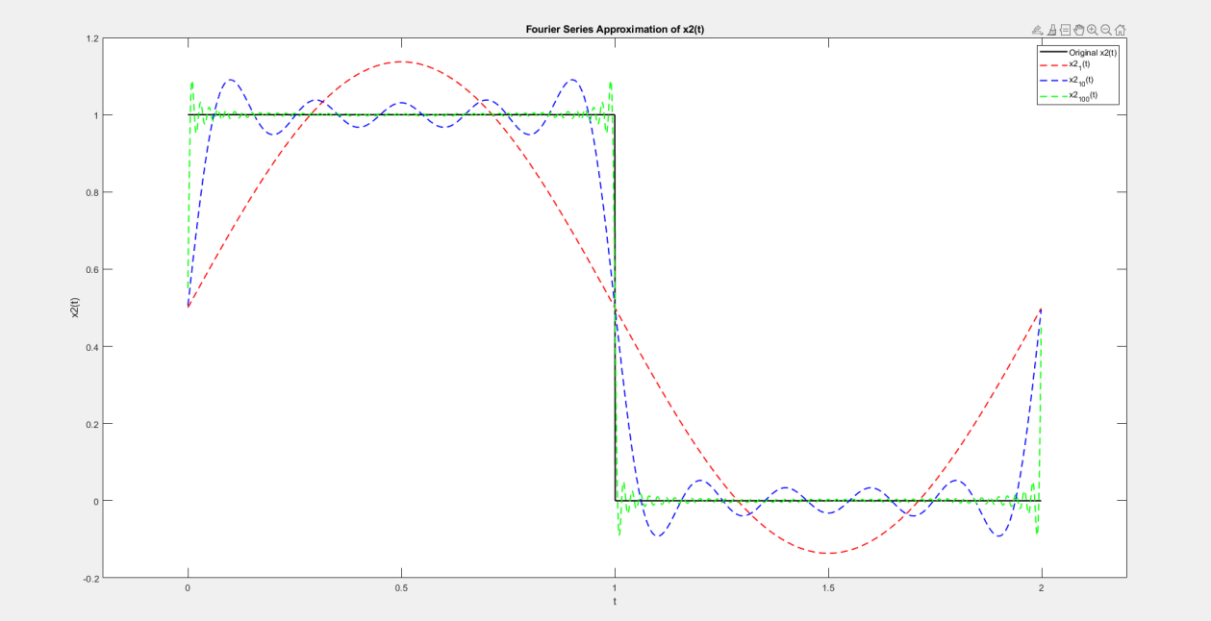


figure 3 : x1(t) Fourier Series Approximation of x2(t)

For the solid black line we have original signal and fort he red line we have low order approximation, we observe smoothness fort his signal and it couldnt capture the sharp transitions. Fort the blue lne which is using 10 harmonics. İt starts to approxiamte the original signal but ripples are highly noticable. Then we have 100 harmonics signal as green and it is closest match but still its ripples near discontiniouties remain.

Part 2 : Identifying Frequencies

In this part we have the function of

$$x(t) = \sin(2\pi f_1 t) + \sin(2\pi f_2 t)$$

And we have

$$f_1 \epsilon \{2,4,6\}, \qquad f_2 \epsilon \{60,90,120\}$$

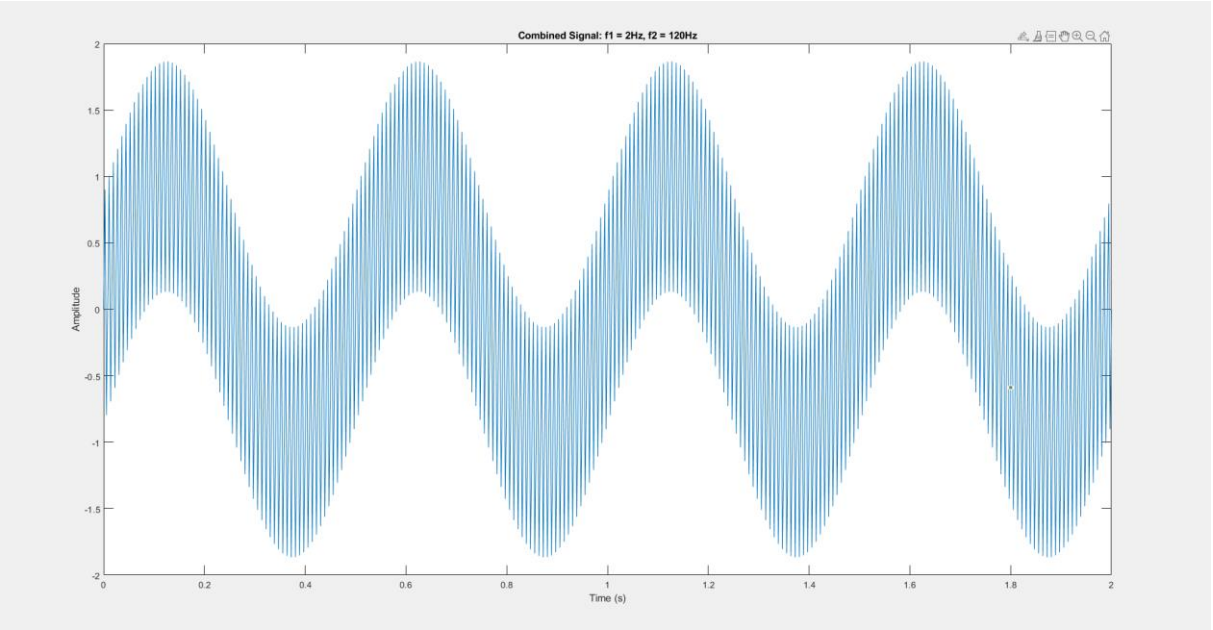
To identify the frequencies of signals we use orthononality of sinusoids.

$$\int \sin(2\pi fit)\sin(2\pi fjt)dt = 0$$

To estimate the f1 and f2 we use the following:

$$\hat{f}_1 = \underset{f\in\{2,4,6\}}{\arg\max} \left| \int_{T_o} x(t)\sin(2\pi f t)\,dt \right| \quad \text{and} \quad \hat{f}_2 = \underset{f\in\{60,90,120\}}{\arg\max} \left| \int_{T_o} x(t)\sin(2\pi f t)\,dt \right|$$

First we set sampling rate as 360 Hz and let f1 2Hz and f2 120Hz the plot of x(t) can be observed as follows and try to identify the frequency as follows.




```
estimated_f1 =

     2


estimated_f2 =

   120
```

figure 4 : x2(t) Combined signal 2Hz and 120Hz and its estimation result

Secondly we assume there is an error in the estimation function with T1 = 19To/20 instead of T0. With this value normally we expect and error fort his situation nevertheless ve get the same result with the previous part. The reason why we couldt see this error is the error is now big enough to observe. The result of faulty estimation is as follows.

```
faulty_estimate_f1 =

     2


faulty_estimate_f2 =

   120
```

figure 5 : Faulty estimation with 19T0/20

Then we add the noise to the our original signal by using this MATLAB code

$$x = x + \text{sigma} * \text{randn}(1;\ \text{length}(x));$$

with the vaues of sigma as 1 and 10 we try to estimate frequencies and we get the result as follows.

```
guess_f1_noise1 =

     2


guess_f2_noise1 =

   120


guess_f1_noise10 =

     2


guess_f2_noise10 =

    60
```

figure 6 : Noisy Guess with sigma 1 and 10

After that we take the values of sigma as 1,10,50,100 abd for each sigma value we perform 10^5 trials and for each trial we get the f1 and f2 values randomly from the array as we say previously. Fort he errorenous estimates we normalize the count with 2x10^5. And plot the stored values. We observe from this plot the error rate is significantly increase between 1 and 10 and there is slight difference between 10 and 50 but there are no observable difference between 50 and 100. The reason why is when we already has significant noise for signal we see the signal as corrupted and then even if ve increase the noise the signal still be a corrupted signal and the error rate do not show significant change. But fort he 1 and 10 noise level ve see that almost clear signal adn there is corruption in 10 and so we see the increase in error rate.

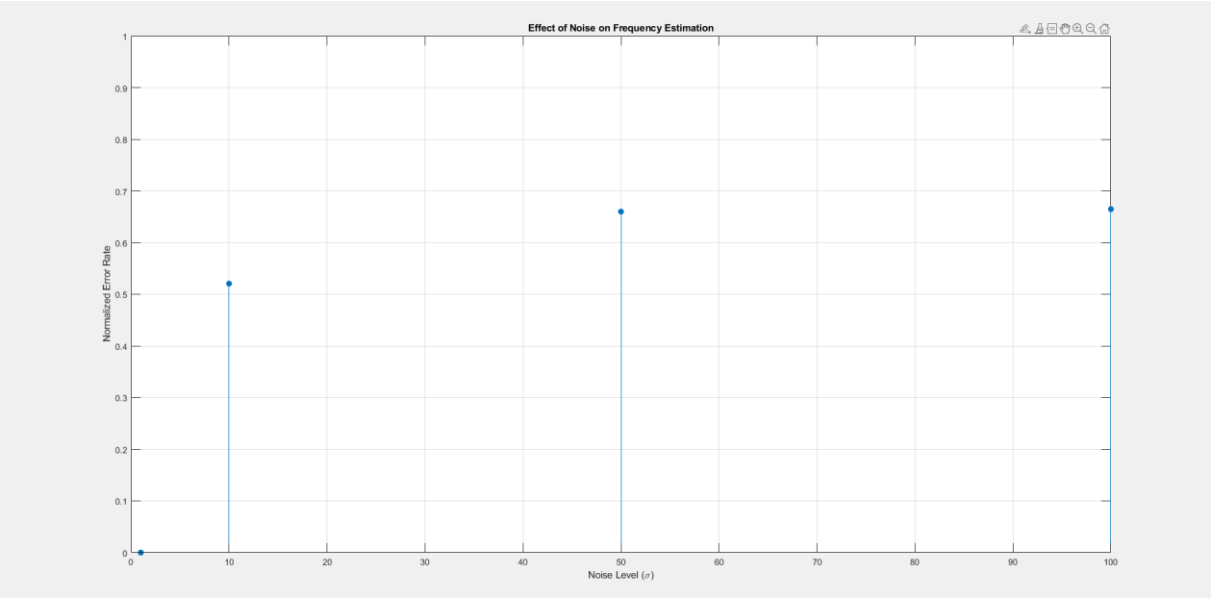


figure 7 : Effect of Noise Frequency Estimation

Part 3: Decomposing Music into Notes

In this part we decompose the given song into musical notes. Wehave the function as:

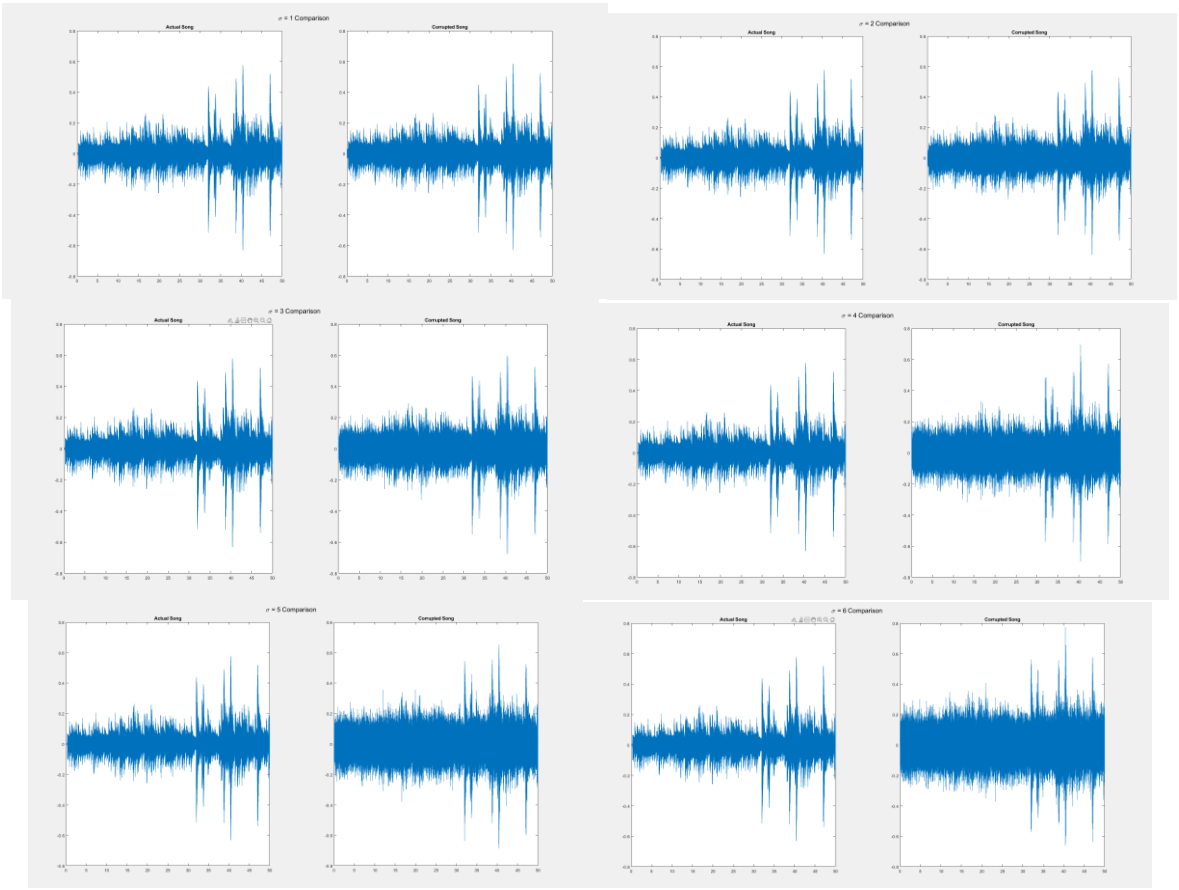
$$\emptyset_k(t) = sin(2\pi f_k t);$$

Where we use the fk as 440 x 2^[(k-1)/12] Hz and k is from 1 to 12 and for each note segments lasts T = 0.25 seconds. In this process we segment the song into intervals and take DFT of the segments. Then we find matched peak freq to the fk and reconstruct the song by using this estimated fk note frequency. Then we get the qsong which is reconstructed song. The peak in each part correctly show the played note and notes are seperated in frequnecy domain. This task uses the frequency domain peak detection it is similar to pitch detection. With this algorith we accurately indentify the music noted with their spectral peak. As for relation with part 2 is both parts detect frequnencies in time domain, and both uses orthogonality and best matches for signals. The differences are ın part two we are simply use signal as sums of sinusoids but for part 3 it requires segment wise processing and we consider temporal structure of signal. But the analysis we use in part 3 requires short time analysis and high resolution.

After that we add noise to the signal by using the following:

$$songnote = songnote + sigma * randn(length(songnote), 1);$$

Then we plot the corrupted signals for sigma from 1 to 10. We observe for lower sigma lower corruption and high noise and corruption on higher sigma values. The plots are as follows.



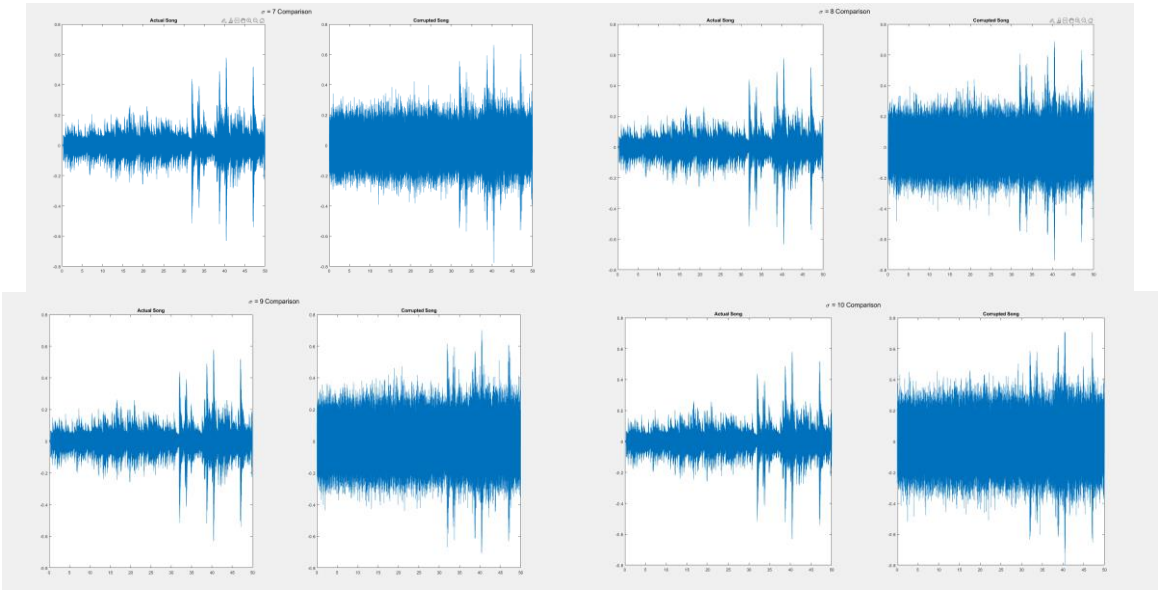


figure 8 : Signal corruption with sigma value from 1 to 10

Part 4: Filtering the Sound Signal

In this part we get the song data and corrupt the signal with noise by using following code and we get the song and corrupted song on the same plot as follows.

$$Y = songdata + 0.05 * randn(length(songdata), 1);$$

,

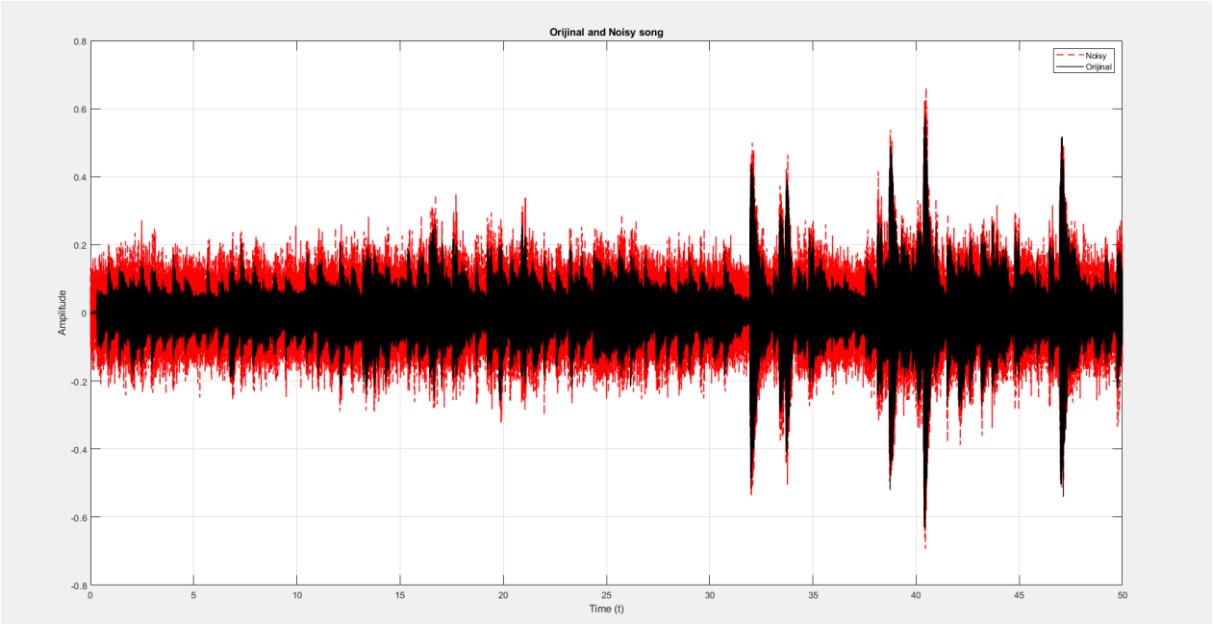


figure 9 : x3(t) Original and Noisy Song

Then we use FIR (fifth order finite impulse response) filter to denoise the corrupted signal. The expression for FIR filter and plot of its impulse response as follows.

$$Z[n] = \begin{cases} \frac{1}{5}\sum_{k=0}^{4} Y[n-k], & \text{if} \quad n \geq 5, \\ Y[n], & \text{otherwise.} \end{cases}$$

$$h[n] = \begin{cases} \frac{1}{5}, & 0 \leq n \leq 4 \\ 0, & \text{otherwise} \end{cases}$$

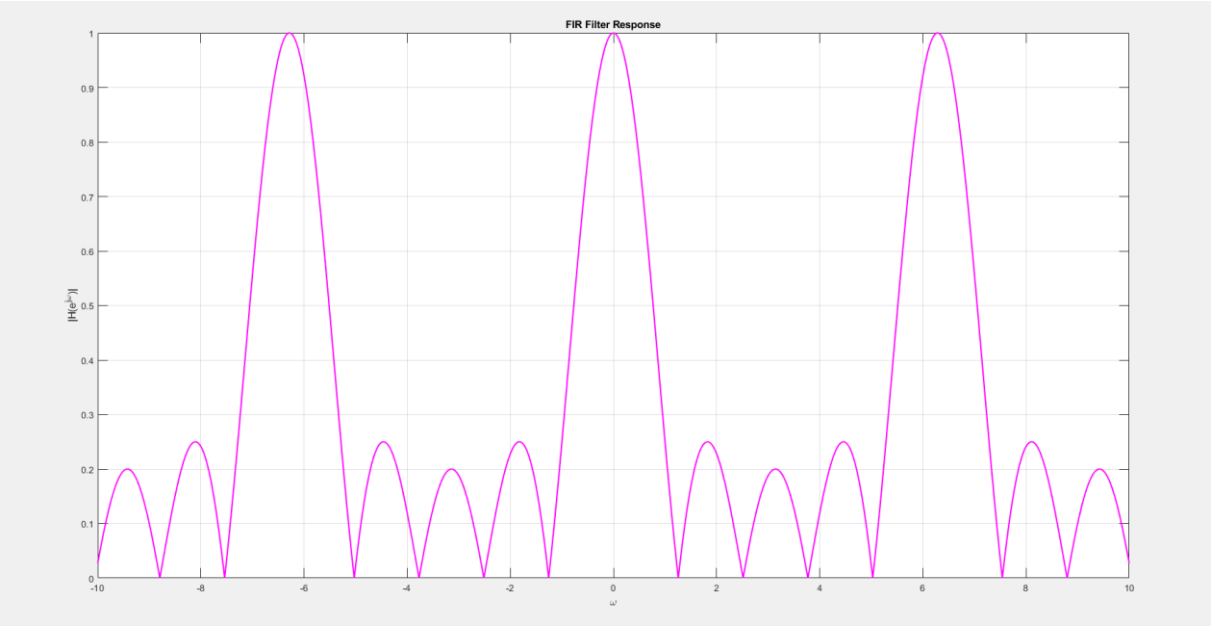


figure 10 : FIR Filter Response

Then we plot the original corrupted and filtered signal at the same plot and we observe as follows. When we listen the sould we can obviously listen the filtering effect on the sound by FIR filter.

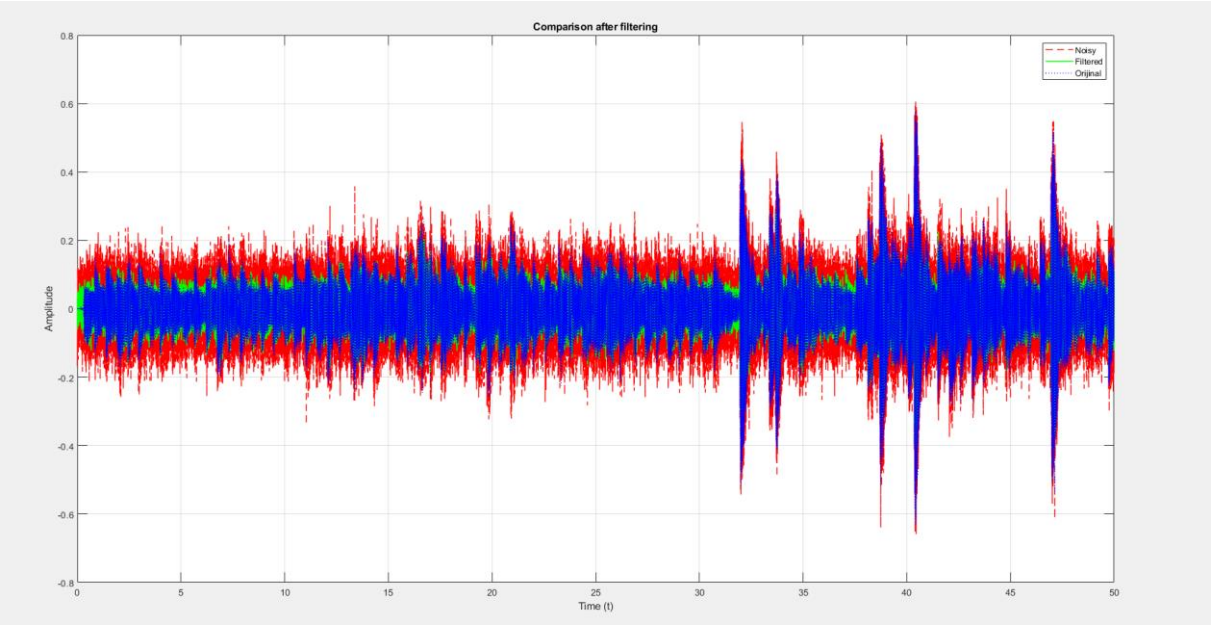


figure 11 : Comparison after filtering

Conclusion

In this lab, we observe the the Fundamentals of Fourier analysis and noise filtering. Firstly we analyse how to harmonic components of signals effect the Fourier approximation. We analyse this approximation for sinusoid and nonsinusoid signals. And we observe how the create noise signal and corrupt the signal then we see how to effect the sigma values on the noise. Finaly we learn how to create FIR filter and filter the noisy signal and observe its effects. In general we achieve required specifications and analyse all the parts with Fourier theory and filtering of signals.

Codes

FSAnalysis

```
function [fsCoeffs] = FSAnalysis(x,k)
% fsCoeffs: An array that contains the Fourier series coefficients of your signal
from -k to k
% x :  One complete period of the sampled continuous-time signal
% k: The number of two-sided Fourier series coefficients that you will estimate

N = length(x); %total number of sample in single period
n = 0:1:N-1; %sample index
fsCoeffs = zeros(1,2*k+1);
for i = -k:1:k
```

```
fsCoeffs(1,i+k+1) = (1/N)*sum(x.*exp(-1j*i*(2*pi/N)*n));
end
```

Part 1

```
clear; clc;

Ts = 0.001;  % time resolution
N1 = 1;      % hormonic limit for x1

% first func
t1 = 0:Ts:pi-Ts;
signal1 = @(t) cos(2*t + pi/4);
x1_vals = signal1(t1);
a1_coeffs = FSAnalysis(x1_vals, N1);

plot_range1 = -pi/2:Ts:pi/2;
x1_approx = zeros(1, length(plot_range1));

for k = -N1:N1
    x1_approx = x1_approx + a1_coeffs(N1 + k + 1) * exp(1j * 2 * pi / pi * k *
plot_range1);
end

figure;
plot(plot_range1, signal1(plot_range1), 'b', 'LineWidth', 3);
hold on;
plot(plot_range1, real(x1_approx), 'g--', 'LineWidth', 3);
title('Approximation of x1(t) with N=1');
xlabel('t'); ylabel('x1(t)');
legend('Original x1(t)', 'Fourier Approximation');
hold off;

%second func
x2 = @(t) mod(t, 2) <= 1;

t2 = 0:Ts:2-Ts;
N2 = 100;
x2_vals = x2(t2);
a2_coeffs = FSAnalysis(x2_vals, N2);

% N = 1, approximations for 10 and 100
approx_x2 = @(N) arrayfun(@(t) ...
    sum(arrayfun(@(k) a2_coeffs(N2 + k + 1) * exp(1j * 2 * pi / 2 * k * t), -
N:N)), t2);

x2_1 = approx_x2(1);
x2_10 = approx_x2(10);
x2_100 = approx_x2(100);

% plots
figure;
plot(t2, x2(t2), 'k', 'LineWidth', 1.5); hold on;
plot(t2, real(x2_1), 'r--', 'LineWidth', 1.3);
plot(t2, real(x2_10), 'b--', 'LineWidth', 1.3);
plot(t2, real(x2_100), 'g--', 'LineWidth', 1.3);
xlim([-0.2 2.2]);
title('Fourier Series Approximation of x2(t)');
xlabel('t'); ylabel('x2(t)');
```

```
legend('Original x2(t)', 'x2_{1}(t)', 'x2_{10}(t)', 'x2_{100}(t)');
hold off;

% Max error comp
max_err_10 = max(abs(x2(t2) - real(x2_10)));
max_err_100 = max(abs(x2(t2) - real(x2_100)));

fprintf('Max error between x(t) and x_{10}(t): %.4f\n', max_err_10);
fprintf('Max error between x(t) and x_{100}(t): %.4f\n', max_err_100);
```

Part 2

```
clear; close all; clc;

% Sampling parameters
Fs = 360;
dt = 1/Fs;

% Signal definition
signal_fn = @(t) sin(2*pi*2*t) + sin(2*pi*120*t);

% Time vector for plotting
t_plot = 0:dt:2;
plot(t_plot, signal_fn(t_plot));
title('Combined Signal: f1 = 2Hz, f2 = 120Hz');
xlabel('Time (s)');
ylabel('Amplitude');

% Frequency sets
f1_set = [2 4 6];
f2_set = [60 90 120];

% Integration time
t_segment = 0:dt:0.5;

% Frequency estimation using T0
projection_f1 = zeros(1,3);
projection_f2 = zeros(1,3);
for k = 1:3
    projection_f1(k) = abs(sum(dt * signal_fn(t_segment) .*
sin(2*pi*f1_set(k)*t_segment)));
    projection_f2(k) = abs(sum(dt * signal_fn(t_segment) .*
sin(2*pi*f2_set(k)*t_segment)));
end

[~, idx_f1] = max(projection_f1);
[~, idx_f2] = max(projection_f2);
estimated_f1 = f1_set(idx_f1)
estimated_f2 = f2_set(idx_f2)

% Estimation with wrong T0
t_faulty = 0:dt:(0.5*19/20);
faulty_proj_f1 = zeros(1,3);
faulty_proj_f2 = zeros(1,3);
for k = 1:3
    faulty_proj_f1(k) = abs(sum(dt * signal_fn(t_faulty) .*
sin(2*pi*f1_set(k)*t_faulty)));
```

```
    faulty_proj_f2(k) = abs(sum(dt * signal_fn(t_faulty) .*
sin(2*pi*f2_set(k)*t_faulty)));
end

[~, faulty_idx_f1] = max(faulty_proj_f1);
[~, faulty_idx_f2] = max(faulty_proj_f2);
faulty_estimate_f1 = f1_set(faulty_idx_f1)
faulty_estimate_f2 = f2_set(faulty_idx_f2)

% Noise effect: sigma = 1 and sigma = 10
clean_signal = signal_fn(t_segment);
signal_noisy_1 = clean_signal + 1*randn(1, length(clean_signal));
signal_noisy_10 = clean_signal + 10*randn(1, length(clean_signal));

% Frequency estimation with sigma = 1
est_f1_noise1 = zeros(1,3);
est_f2_noise1 = zeros(1,3);
for k = 1:3
    est_f1_noise1(k) = abs(sum(dt * signal_noisy_1 .*
sin(2*pi*f1_set(k)*t_segment)));
    est_f2_noise1(k) = abs(sum(dt * signal_noisy_1 .*
sin(2*pi*f2_set(k)*t_segment)));
end
[~, f1_idx_noise1] = max(est_f1_noise1);
[~, f2_idx_noise1] = max(est_f2_noise1);
guess_f1_noise1 = f1_set(f1_idx_noise1)
guess_f2_noise1 = f2_set(f2_idx_noise1)

% Frequency estimation with sigma = 10
est_f1_noise10 = zeros(1,3);
est_f2_noise10 = zeros(1,3);
for k = 1:3
    est_f1_noise10(k) = abs(sum(dt * signal_noisy_10 .*
sin(2*pi*f1_set(k)*t_segment)));
    est_f2_noise10(k) = abs(sum(dt * signal_noisy_10 .*
sin(2*pi*f2_set(k)*t_segment)));
end
[~, f1_idx_noise10] = max(est_f1_noise10);
[~, f2_idx_noise10] = max(est_f2_noise10);
guess_f1_noise10 = f1_set(f1_idx_noise10)
guess_f2_noise10 = f2_set(f2_idx_noise10)

% simulation for different sigma
sigma_vals = [1 10 50 100];
error_rate = zeros(1, length(sigma_vals));
total_trials = 100000;

for s = 1:length(sigma_vals)
    errors = 0;
    sigma = sigma_vals(s);
    for trial = 1:total_trials
        % random f1 and f2
        true_f1 = f1_set(randi(3));
        true_f2 = f2_set(randi(3));
        test_signal = sin(2*pi*true_f1*t_segment) + sin(2*pi*true_f2*t_segment);
        test_noisy = test_signal + sigma*randn(1, length(t_segment));

        resp_f1 = zeros(1,3);
        resp_f2 = zeros(1,3);
```

```
        for k = 1:3
            resp_f1(k) = abs(sum(dt * test_noisy .*
sin(2*pi*f1_set(k)*t_segment)));
            resp_f2(k) = abs(sum(dt * test_noisy .*
sin(2*pi*f2_set(k)*t_segment)));
        end

        [~, pred_idx_f1] = max(resp_f1);
        [~, pred_idx_f2] = max(resp_f2);
        pred_f1 = f1_set(pred_idx_f1);
        pred_f2 = f2_set(pred_idx_f2);

        if pred_f1 ~= true_f1
            errors = errors + 1;
        end
        if pred_f2 ~= true_f2
            errors = errors + 1;
        end
    end
    error_rate(s) = errors / (2 * total_trials);
end

% plot error vs sigma
figure;
stem(sigma_vals, error_rate, 'filled');
xlabel('Noise Level (\sigma)');
ylabel('Normalized Error Rate');
title('Effect of Noise on Frequency Estimation');
grid on;
ylim([0 1]);
```

Part 3

```
clear

T_duration = 0.5;
sampling_rate = 4000;
Ts = 1 / sampling_rate;

note_frequencies = zeros(1,12);
for idx = 1:12
    note_frequencies(idx) = 440 * 2^((idx - 1) / 12);
end

detected_notes = zeros(1,50/T_duration);
time_vector = 0:Ts:T_duration - Ts;

load MA2_songdata.mat

for seg = 1:50/T_duration
    segment = transpose(songdata((seg - 1)*T_duration*sampling_rate + 1 :
seg*T_duration*sampling_rate));
    projection = zeros(1,12);
    for nf = 1:12
        projection(nf) = abs(sum(Ts * segment .* sin(2 * pi * note_frequencies(nf)
* time_vector)));
    end
```

```
    [~, best_match] = max(projection);
    detected_notes(seg) = note_frequencies(best_match);
end

qsong = zeros(200000,1);
for seg = 1:50/T_duration
    note_waveform = sin(2 * pi * detected_notes(seg) * time_vector);
    qsong((seg - 1)*T_duration*sampling_rate + 1 : seg*T_duration*sampling_rate) =
transpose(note_waveform);
end

%sound(qsong, sampling_rate)


sigma_values = 1:10;
for sigma = sigma_values
    figure()
    noisy_song = songdata + sigma * randn(length(songdata),1) / 100;
    time_range = transpose(0:Ts:50-Ts);

    subplot(1,2,1)
    plot(time_range, songdata)
    xlim([0 50])
    ylim([-0.8 0.8])
    title("Actual Song")

    subplot(1,2,2)
    plot(time_range, noisy_song)
    xlim([0 50])
    ylim([-0.8 0.8])
    title("Corrupted Song")

    sgtitle("\sigma = " + sigma + " Comparison")
end



chosen_sigma = 7;
noisy_song = songdata + chosen_sigma * randn(length(songdata),1) / 100;
detected_noisy_notes = zeros(1,50/T_duration);

for seg = 1:50/T_duration
    segment = transpose(noisy_song((seg - 1)*T_duration*sampling_rate + 1 :
seg*T_duration*sampling_rate));
    projection = zeros(1,12);
    for nf = 1:12
        projection(nf) = abs(sum(Ts * segment .* sin(2 * pi * note_frequencies(nf)
* time_vector)));
    end
    [~, best_match] = max(projection);
    detected_noisy_notes(seg) = note_frequencies(best_match);
end

reconstructed_song = zeros(200000,1);
for seg = 1:50/T_duration
    note_waveform = sin(2 * pi * detected_noisy_notes(seg) * time_vector);
    reconstructed_song((seg - 1)*T_duration*sampling_rate + 1 :
seg*T_duration*sampling_rate) = transpose(note_waveform);
end
```

```
%sound(reconstructed_song, sampling_rate)
```

Part 4

```
clear

load MA2_songdata.mat

% parameters
sampling_rate = 4000;
Ts = 1/sampling_rate;
time_vector = 0:Ts:50-Ts;

% noisiy signal
corrupted_wave = songdata + 0.05 * randn(length(songdata),1);

% original and noisy comparison
figure()
plot(time_vector, corrupted_wave, 'r--', 'LineWidth', 1.2)
hold on
plot(time_vector, songdata, 'k', 'LineWidth', 1)
hold off
legend('Noisy', 'Orijinal')
xlabel('Time (t)')
ylabel('Amplitude')
title('Original and Noisy song')
grid on

% 5th order finite impulse response
filtered_wave = corrupted_wave;
for n = 5:length(corrupted_wave)
    filtered_wave(n) = (corrupted_wave(n) + corrupted_wave(n-1) +
corrupted_wave(n-2) + corrupted_wave(n-3) + corrupted_wave(n-4)) / 5;
end

% Impuls response
impulse_response = [1/5 1/5 1/5 1/5 1/5];

% Freq response
freq_vals = -10:Ts:10;
Hw = @(w) 1/5 * (1 + exp(-1j*w) + exp(-1j*2*w) + exp(-1j*3*w) + exp(-1j*4*w));

% Freq resp plot
figure()
plot(freq_vals, abs(Hw(freq_vals)), 'm', 'LineWidth', 1.5)
xlabel('\omega')
ylabel('|H(e^{j\omega})|')
title('FIR Filter Response')
grid on

% noisy,filtered,original
figure()
plot(time_vector, corrupted_wave, 'r--', 'LineWidth', 1)
hold on
plot(time_vector, filtered_wave, 'g', 'LineWidth', 1.2)
plot(time_vector, songdata, 'b:', 'LineWidth', 1)
hold off
legend('Noisy', 'Filtered', 'Orijinal')
```

```
xlabel('Time (t)')
ylabel('Amplitude')
title('Comparison after filtering')
grid on

%sound(filtered_wave, sampling_rate)
```