Xi'an Jiaotong-Liverpool University

西交利物浦大學

EEE204 Continuous and Discrete Time Signals and Systems II

Lab 1

Name: Kai-Yu Lu

ID Number: 1614649

Time: 2019.5.20

Fourier Analysis

Experimental Procedure

1. **Plot a graph of the signal x(t) with T = 100 ms, A = 1, sampling interval of 0.1 ms, and the observation window is (Wb, We) = (-0:2; 0:2) s, using the functions** "square_wave_fun.m" **and** "triangular_wave_fun.m"**.**

```
function [x, T_s_vct] = square_wave_fun(T, A, T_s, W_b, W_e)
% T_s_vct (output) is a vector representing sampling time
% x (output) is a vector containing the sampled x(t) signal
% T is the signal period
% A is the signal amplitude
% T_s is sampling period
% W_b is the starting time of the observation window
% W_e is the ending time of the observation window
%

T_s_vct = [W_b : T_s : W_e];

x_T = -1 * ones(1, length(T_s_vct));

for ii = floor(W_b/T) : ceil(W_e/T)
    x_T(( -T/4 + ii* T < T_s_vct ) & ( T_s_vct < T/4 + ii* T ) ) = 1;
end


x = x_T;
return
```

Figure 1.1.1: The code for "square_wave_fun.m"

```
function [x, T_s_vct] = triangular_wave_fun(T, A, T_s, W_b, W_e)
% T_s_vct (output) is a vector representing sampling time
% x (output) is a vector containing the sampled x(t) signal
% T is the signal period
% A is the signal amplitude
% T_s is sampling period
% W_b is the starting time of the observation window
% W_e is the ending time of the observation window
%

xL = linspace(-A, A, T/T_s + 1);
xL = xL(2:end);

T_s_vctL = linspace(-T/2, 0, T/T_s + 1);
T_s_vctL = T_s_vctL(2:end);

xR = linspace(A, -A, T/T_s + 1);
xR = xR(2:end);
x = [xL xR];

T_s_vctR = linspace(0, T/2, T/T_s + 1);
T_s_vctR = T_s_vctR(2:end);

T_s_vct = [T_s_vctL T_s_vctR];

for ii = -1 : -1 : floor(W_b/T)
    x = [[xL xR] x ];
    T_s_vct = [[T_s_vctL T_s_vctR]+T*ii T_s_vct];
end

for ii = 1 : floor(W_e/T)
    x = [x [xL xR] ];
    T_s_vct = [ T_s_vct  [T_s_vctL T_s_vctR]+T*ii];
end

ind = ( (W_b <= T_s_vct) & (T_s_vct <= W_e) );
T_s_vct = T_s_vct(ind);
x = x(ind);

return
```

Figure 1.1.2: The code for "triangular_wave_fun.m"

```
close all;
clear all;
clc

%%
A =1;
T = 100e-3;
T_s = 0.1e-3; %s
W_b = -0.2;
W_e=0.2;
[x, T_s_vct] = square_wave_fun(T, A, T_s, W_b, W_e);

plot(T_s_vct,x), xlabel('time'),ylabel('magnitute'),
grid on
ylim([-1.5,1.5]);
```

Figure 1.1.3: The code for plotting a graph of x(t) in square wave.

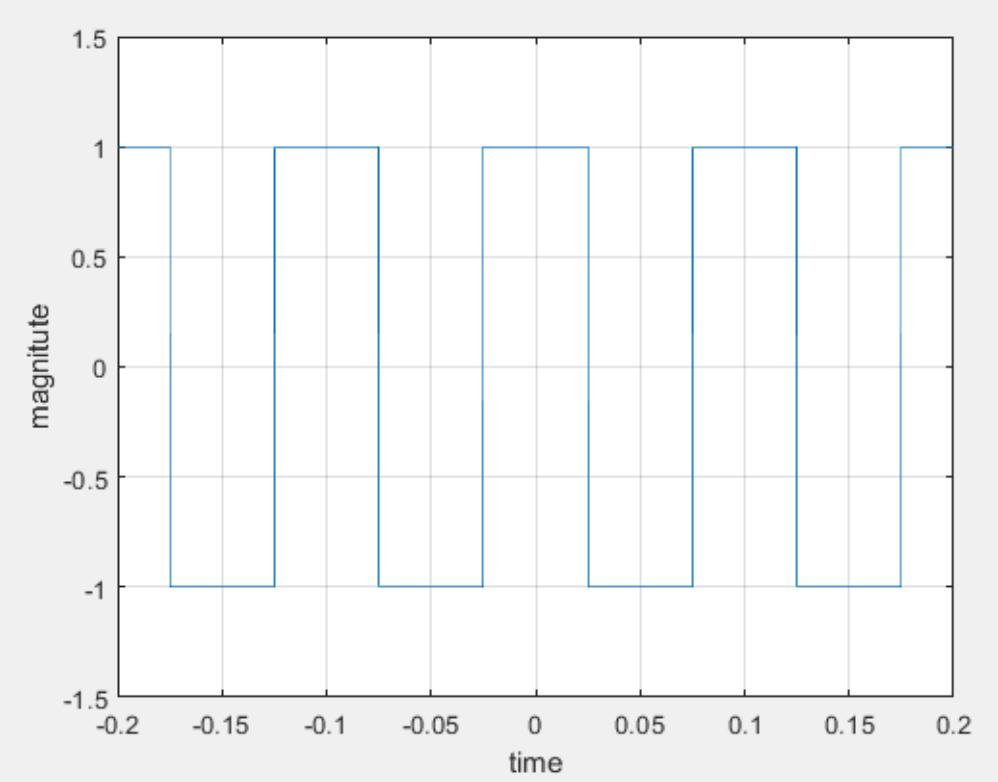

Figure 1.1.4: The result of plotting a graph of x(t) in square wave.

```
close all
clear all
clc

%%
A =1;
T = 100e-3;
T_s = 0.1e-3; %s

W_b = -0.2;
W_e=0.2;
[x, T_s_vct] = triangular_wave_fun(T, A, T_s, W_b, W_e);

plot(T_s_vct,x), xlabel('time'),ylabel('magnitute'),
grid on
ylim([-1.5,1.5]);
```

Figure 1.1.5: The code for plotting a graph of x(t) in triangular wave.

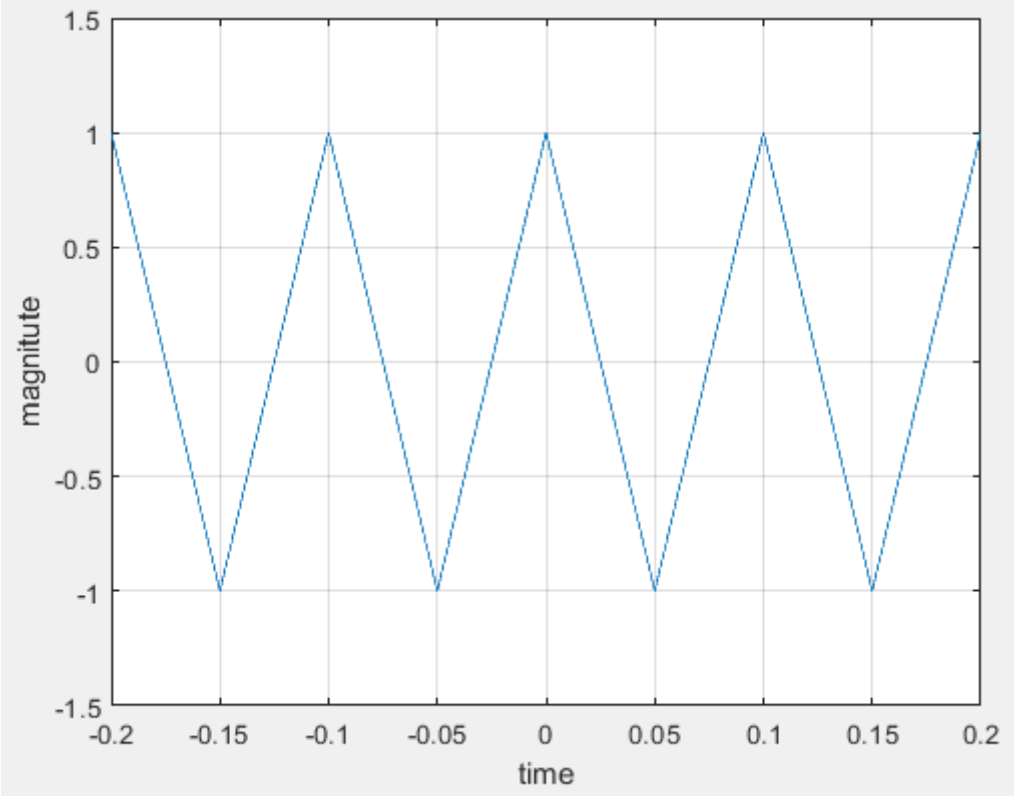

Figure 1.1.6: The result of plotting a graph of x(t) in triangular wave.

From the resulting graph, we can observe that Figure 1.1.4 is a periodic square wave whose period is 0.1s and the magnitude is 1 and Figure 1.1.6 is a periodic triangular wave whose period is 0.1s and the magnitude is 1. However, these two signals have slight zigzag in the lines due to the factor of the sampling time.

2. **Plot a graph of the N-th partial sum $\tilde{x}(t)$ with T = 100 ms, A = 1, and for N = [3; 4; 5; 11], using the functions** "square_wave_PFS_fun.m" **and** "triangular_wave_PFS_fun.m"**. Print the resulting waveforms and compare with the original signal x(t), what do you observe about the behavior of the partial sum when N gets increased? what do you observe about the behavior of the partial sum on the discontinuities of x(t)? (compare the behavior of the triangular and square waveform).**

```
function [pfs_X, T_s_vct] = square_wave_PFS_fun(N, T, A, T_s, W_b, W_e)
% T_s_vct (output) is a vector representing sampling time
% pfs_X (output) is a vector containing the sampled x(t) signal
% N is the number of harmonics
% T is the signal period
% A is the signal amplitude
% T_s is sampling period
% W_b is the starting time of the observation window
% W_e is the ending time of the observation window

Tau = 0;

T_s_vct = [W_b : T_s : W_e];

p_sum = 0;

for nn = -N : 1 : N
    if mod(abs(nn), 2)
        c_n = (-1)^((nn-1)/2)* 2/(nn*pi);
    else
        c_n = 0;
    end
    h_n = c_n * exp(-i*2*pi* nn * (T_s_vct-Tau)/T);
    p_sum = p_sum + h_n;

end

pfs_X = p_sum;
return



nn = -N : 1 : N;
c_n = mod(abs(nn), 2) .* (-1).^((nn-1)/2)* 2./(nn*pi);
stem(nn*1/T, abs(c_n), 'r')
```

Figure 1.2.1: The code for "square_wave_PFS_fun.m"

```
function [pfs_X, T_s_vct] = triangular_wave_PFS_fun(N, T, A, T_s, W_b, W_e)
% T_s_vct (output) is a vector representing sampling time
% pfs_X (output) is a vector containing the sampled x(t) signal
% N is the number of harmonics
% T is the signal period
% A is the signal amplitude
% T_s is sampling period
% W_b is the starting time of the observation window
% W_e is the ending time of the observation window
%
T_s_vct = [W_b : T_s : W_e];

p_sum = 0;

for nn = -N : 1 : N
    if mod(abs(nn), 2)
        c_n = 4/(nn*pi)^2;
    else
        c_n = 0;
    end

    p_sum = p_sum + c_n * exp(-i*2*pi* nn * T_s_vct/T);

end

pfs_X = p_sum;
return


nn = -N : 1 : N;
c_n = 1.6*mod(abs(nn), 2) .* 4./(nn*pi).^2;
stem(nn*1/T, abs(c_n))
```

Figure 1.2.2: The code for "triangular_wave_PFS_fun.m"

```
close all
clear all
clc

%%
A =1;
T = 100e-3;
T_s = 0.1e-3; %s
N = [3,4,5,11];
W_b=-0.2 ;
W_e=0.2;
count = 1;
for i = N
    [pfs_X, T_s_vct] = square_wave_PFS_fun(i, T, A, T_s, W_b, W_e);
    subplot(2,2,count);
    plot(T_s_vct,pfs_X), xlabel('time'),ylabel('magnitute'),title(['when N = ',num2str(i)]);
    grid on
    count = count+1;
end
```

Figure 1.2.3: The code for plotting a graph of x(t) in square wave PFS.

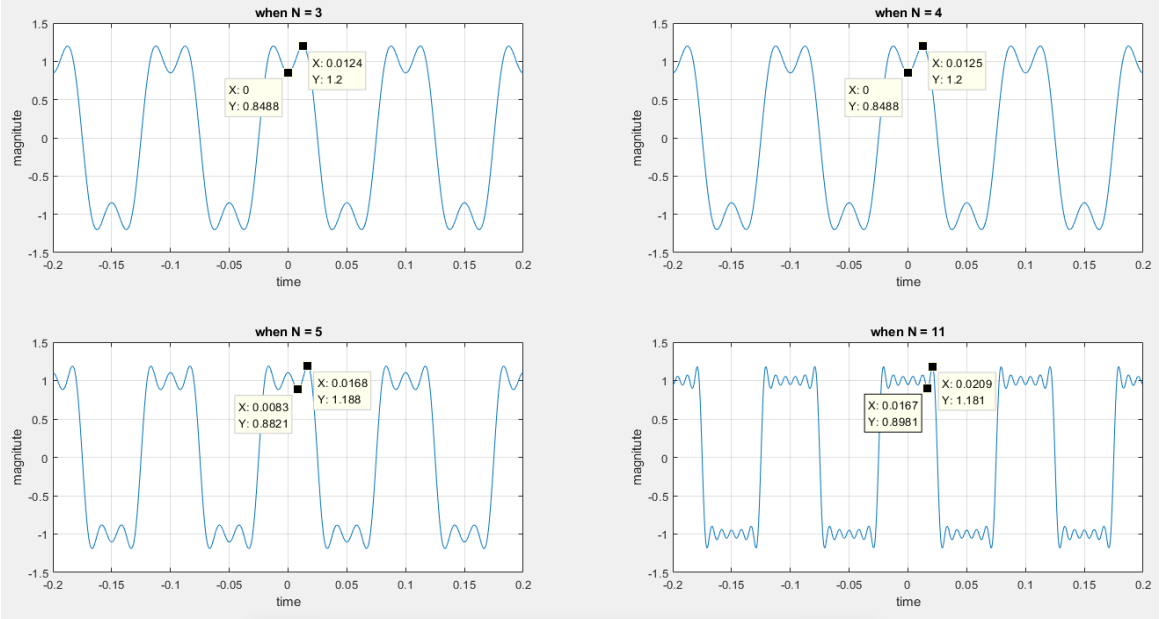

Figure 1.2.4: The result of plotting a graph of the N-th partial sum $\tilde{x}(t)$ in square wave.

```
close all
clear all
clc

%%
A =1;
T = 100e-3;
T_s = 0.1e-3; %s
N = [3,4,5,11];
W_b=-0.2 ;
W_e=0.2;
count = 1;
for i = N
    [pfs_X, T_s_vct] = triangular_wave_PFS_fun(i, T, A, T_s, W_b, W_e);
    subplot(2,2,count)
    plot(T_s_vct,pfs_X), xlabel('time'),ylabel('magnitute'),title(['when N = ',num2str(i)]);
    grid on
    count = count+1;
end
```

Figure 1.2.5: The code for plotting a graph of x(t) in triangular wave PFS.

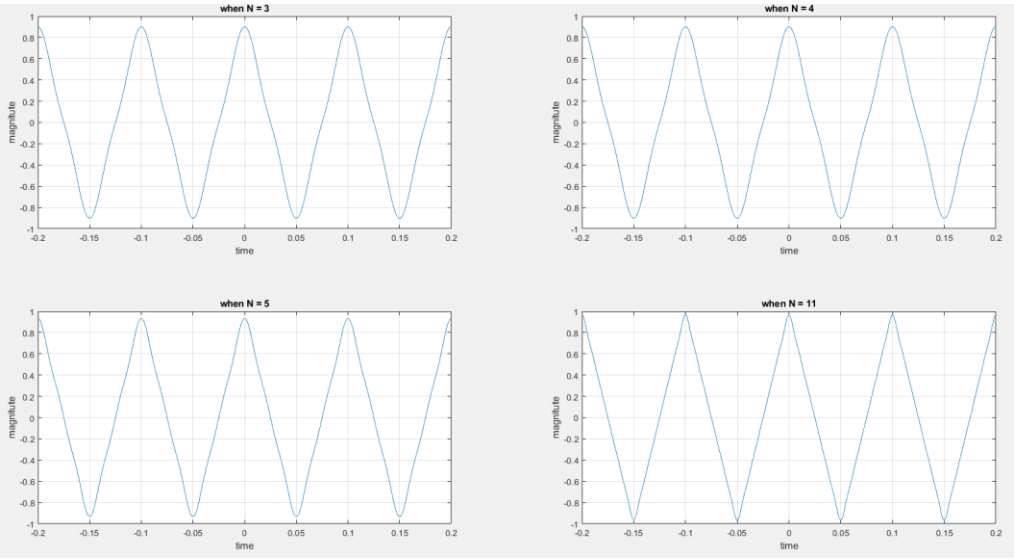

Figure 1.2.6: The result of plotting a graph of the N-th partial sum $\tilde{x}(t)$ in triangular wave.

From Figure 1.2.4 and Figure 1.2.6, it could be observed that the wave would be more close and smooth to original the square/triangular wave according to the increment of N and less N will cause certain distortion in the simulated waves.

Comparing the behavior of the partial sum of the discontinuous x(t) with the behavior of the triangular/square waveform, the partial sum of the discontinuous x(t) has certain overshoots and if N is less, it will cause certain distortion in the simulated waves. However, the triangular/square waveform in previous section does not have any overshoot and have less distortion.

3. **Plot the signal y(t) obtained by filtering the signal x(t) (T = 100 ms and A = 1) using the low-pass filter represented in Fig. 4. Print the resulting waveform and observe the large oscillations near the jump discontinuity (what is the name of this phenomena?). Evaluate the overshoot for the square waveform**

as $a = \tilde{x}(t_0^+) - \tilde{x}(t_0^-)$, with $\tilde{x}(t_0^+)$ $(\widetilde{x}(t_0^-))$ is the first maximum (minimum) value of ~x to occur near the jump discontinuity point. Compare the evaluated overshoot value with the theoretical one.

```
close all
clear all
clc

%%
N=125;
A =1;
T = 100e-3;
T_s = 0.1e-3; %s
W_b=-0.2;
W_e=0.2;

[pfs_X, T_s_vct] = square_wave_PFS_fun(N, T, A, T_s, W_b, W_e);
y_max = abs(max(pfs_X));
plot(T_s_vct,pfs_X), xlabel('time'),ylabel('magnitute')
grid on
```

Figure 1.3.1: The code for plotting signal y(t) in square wave.

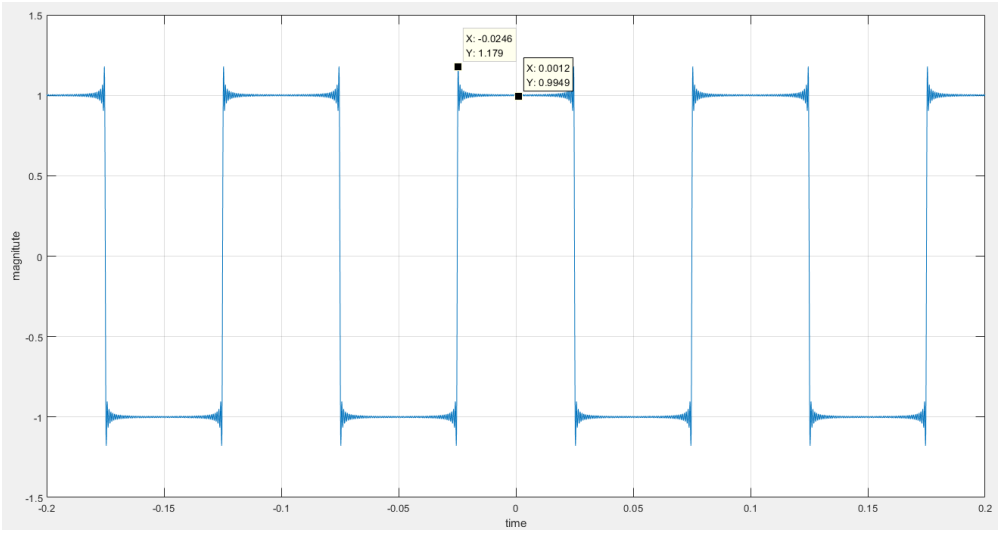

Figure 1.3.2: Plotted signal y(t) in square wave.

```
close all
clear all
clc

%%
N=125;
A =1;
T = 100e-3;
T_s = 0.1e-3; %s
W_b=-0.2 ;
W_e=0.2;

[pfs_X, T_s_vct] = triangular_wave_PFS_fun(N, T, A, T_s, W_b, W_e);
y_max = abs(max(pfs_X));
plot(T_s_vct,pfs_X), xlabel('time'),ylabel('magnitute')
grid on
```

Figure 1.3.3: The code for plotting signal y(t) in square wave.

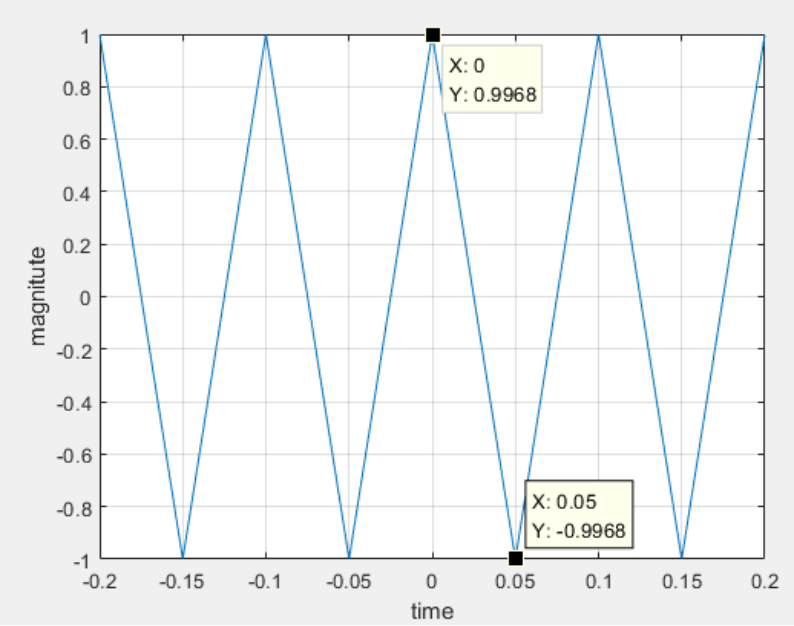

Figure 1.3.4: Plotted signal y(t) in triangular wave.

From Figure 1.3.2, we can observe that Fourier sums of the square wave has certain overshoots near the jump discontinuity and this situation does not disappear when the terms increase. This phenomenon is called Gibbs phenomenon [1]. The overshoot value of the square wave is 1.179-0.9949=0.1841. In theory, the overshoot of the square wave is $\frac{1}{2}\int_0^{\pi}\frac{sint}{t}dt-\frac{\pi}{4}-1=0.14$ [2].

Comparing the evaluated overshoot value (0.18) with the theoretical one (0.14), they are approximately equal. It is therefore we can conclude that the simulated result is correct.

However, from Figure 1.3.4, we discover that the triangular wave does not have the large oscillations, it is therefore Gibbs phenomenon does not exist in triangular wave.

4. **Plot the amplitude and phase of the harmonics of x(t), in the frequency range (-1250; 1250) Hz, with T = 100 ms A = 1. Print the resulting spectrum. How many harmonics are there in the plotted range of frequency? Note the higher frequency content of the square waveform.**

```
close all
clear all
clc

%%
N=125;
A =1;
T = 100e-3;
T_s = 0.1e-3; %s
W_b=-0.2;
W_e=0.2;
FS=2500;
[pfs_X, T_s_vct] = square_wave_PFS_fun(N, T, A, T_s, W_b, W_e);
getSpectrum(fft(pfs_X), FS)
```

Figure 1.4.1: The code for plotting amplitude and phase of the harmonics of x(t) in square wave.

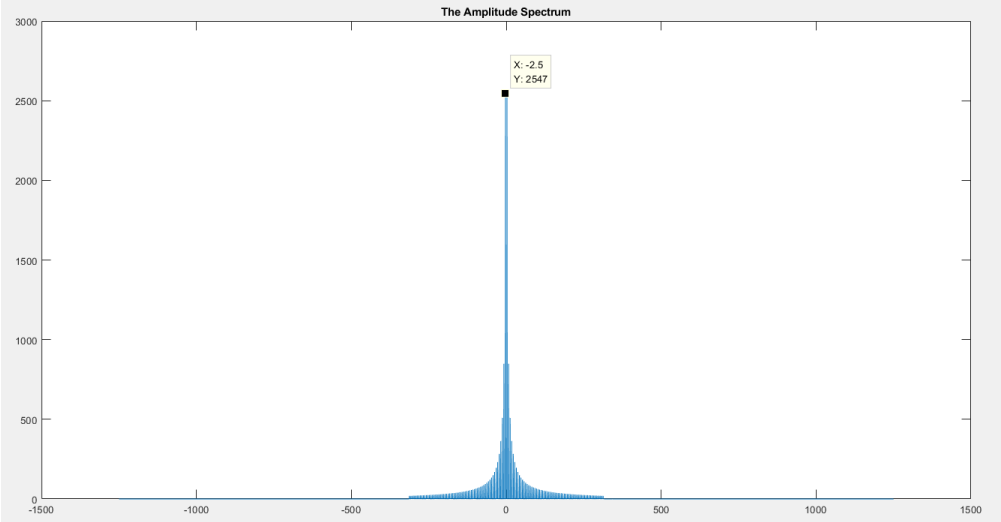

Figure 1.4.2: The amplitude the harmonics of x(t) in square wave.

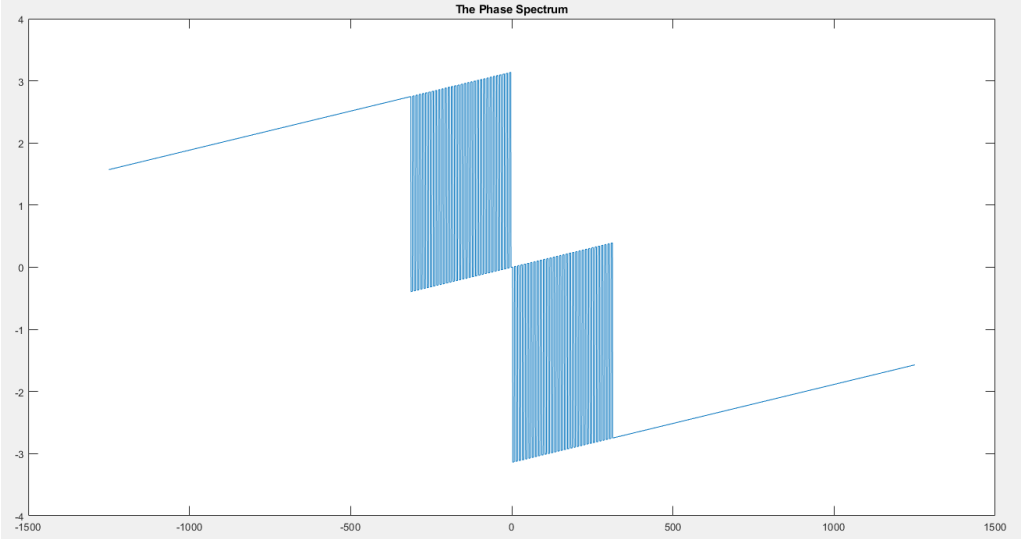

Figure 1.4.3: The phase the harmonics of x(t) in square wave.

In order to find the harmonics of x(t), we can use the formula:

$$x(t) = \sum_{n=-\infty}^{\infty} h_n(t)$$

$h_n(t)$ is the harmonic components of $x(t)$. Additionally, components at frequency $\pm\frac{1}{T}, \pm\frac{2}{T}, \pm\frac{3}{T}$ are the ordered harmonics. In this section, T = 100ms and it is the frequency 10Hz. Due to the fact that there is a frequency in zero

point, the number of harmonics is $\frac{1250}{10} + 1 = 126$.

```
close all
clear all
clc

%%
N=125;
A =1;
T = 100e-3;
T_s = 0.1e-3; %s
W_b=-0.2;
W_e=0.2;
FS=2500;
[pfs_X, T_s_vct] = triangular_wave_PFS_fun(N, T, A, T_s, W_b, W_e);
getSpectrum(fft(pfs_X), FS)
```

Figure 1.4.4 Figure 1.4.1: The code for plotting amplitude and phase of the harmonics of x(t) in triangular wave.

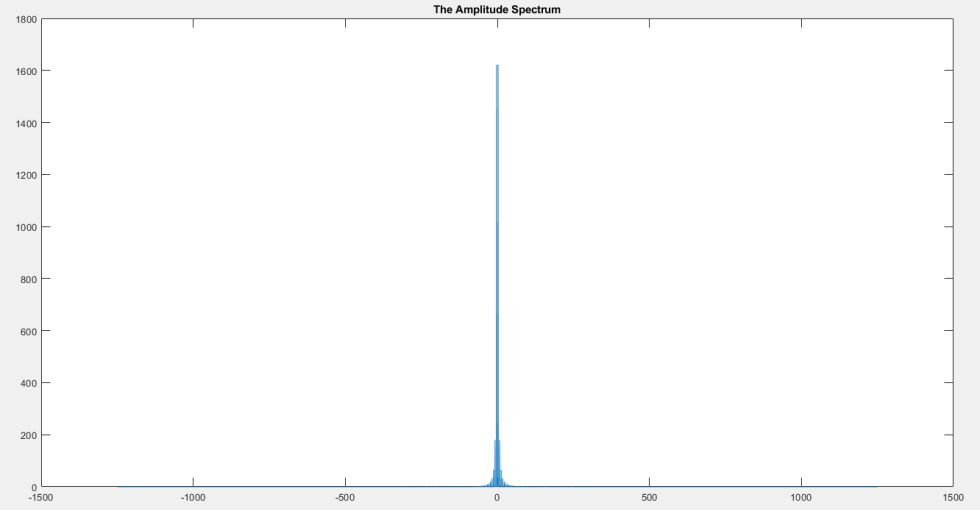

Figure 1.4.5: The amplitude the harmonics of x(t) in triangular wave.

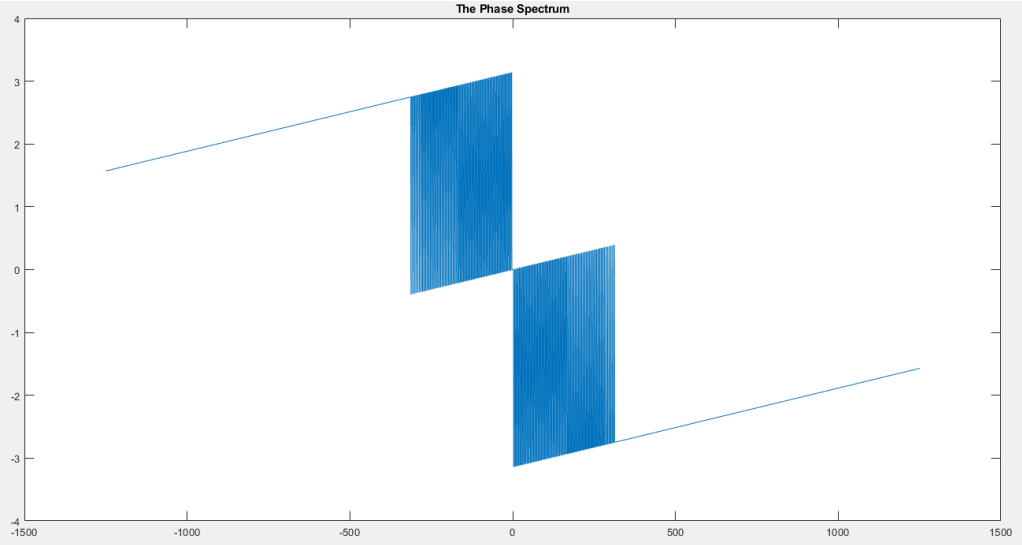

Figure 1.4.6: The phase the harmonics of x(t) in triangular wave.

Discrete Fourier Transform

Experimental Procedure

1. **Use the command audioplayer and play to play the variable $x_{20}[n]$.**

```
[y,Fs]=audioread('music.wav');
x_20=20*Fs;
[y_1,Fs]=audioread('music.wav',[1 x_20]);

playerObj = audioplayer(y_1,Fs);
play(playerObj)
```

Figure 2.1.1: Code for playing the variable $x_{20}[n]$

2. **Use the command fft to evaluate the DFT of the signal $x_{20}[n]$, plot the amplitude and phase spectrum using the command plotSpectrum(estimator) or the provided function getSpectrum(f,**

FS). Print the amplitude spectrum of $x_{20}[n]$ and note that the low frequency components (up to 4000 Hz) carry most of the signal energy.

```
1 -     [y,Fs]=audioread('music.wav');
2 -     x_20=20*Fs;
3 -     [y_2,Fs]=audioread('music.wav',[1 x_20]);
4
5 -     Y_f_2=fft(y_2);
6 -     getSpectrum(Y_f_2,Fs);
```

Figure 2.2.1: The code for plotting the amplitude and phase spectrum of $x_{20}[n]$

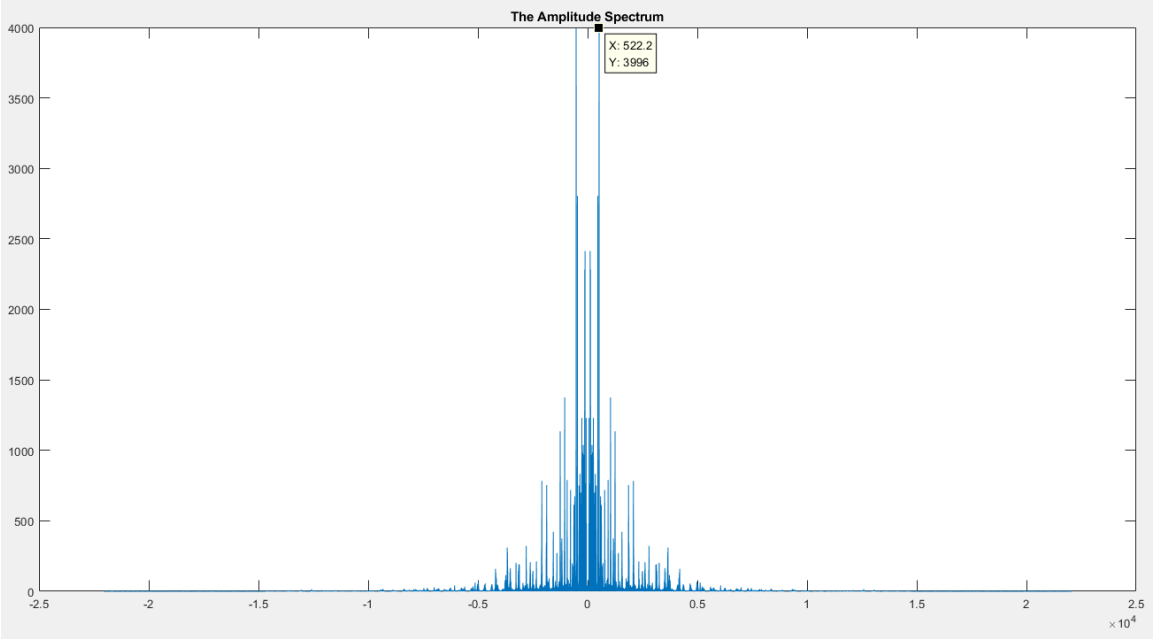

Figure 2.2.2: Amplitude spectrum of $x_{20}[n]$

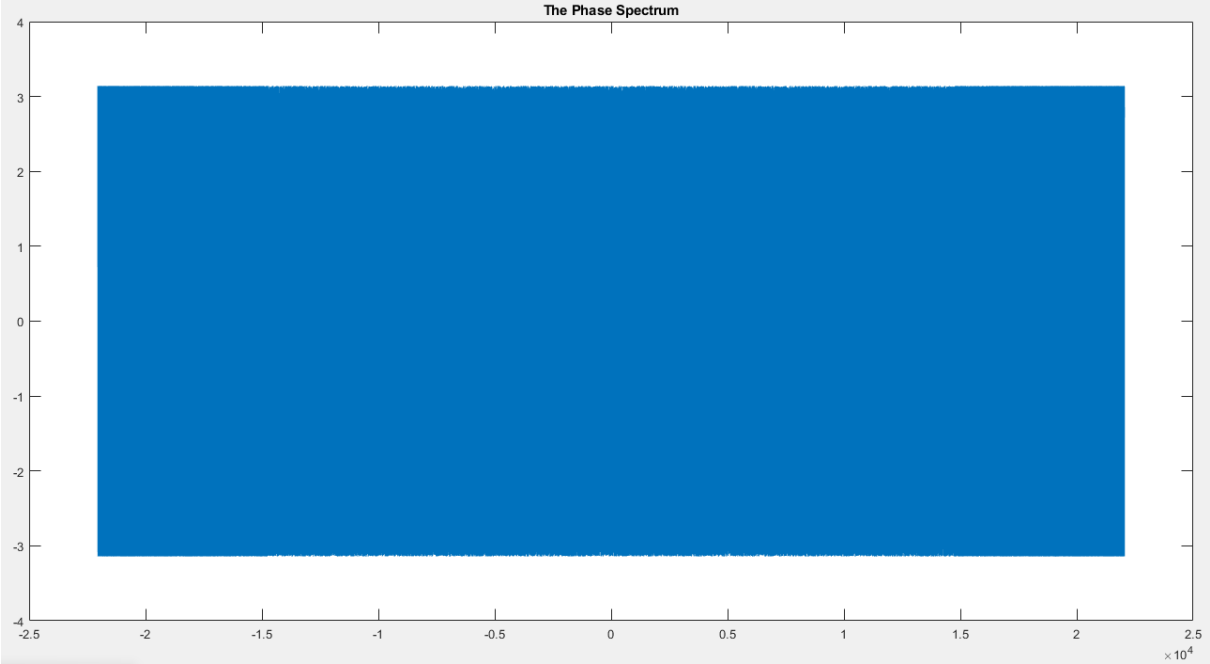

Figure 2.2.3: Phase spectrum of $x_{20}[n]$

As we can see in Figure 2.2.2, when the low frequency is equal to 522.2 Hz, it carried the most of the signal energy because this point has the largest amplitude.

3. **Scaling property (contracting and stretching signal): Generate a vector $x_d[n]$ by down-sampling the vector $x_{20}[n]$ by a factor of four. Using the original value of the sampling frequency, play the content of x_d by the command play, what did you notice? Use the command fft to evaluate the DFT of $x_d[n]$, plot the amplitude and phase spectrum using the command plotSpectrum(estimator) or getSpectrum(f, FS). Print the amplitude spectrum of $x_d[n]$ and compare it with the amplitude spectrum of the original signal. What will happen to the signal spectrum when it is played faster?**

```
[y,Fs]=audioread('music.wav');
x_20=20*Fs;
[y_1,Fs]=audioread('music.wav',[1 x_20]);

factor = 2;
y_3= downsample(y_1,factor);
Y_f_3 =fft(y_3);
playerObj = audioplayer(y_3,Fs);
play(playerObj);
getSpectrum(Y_f_3,Fs);
```

Figure 2.3.1: The code for the problem 3.

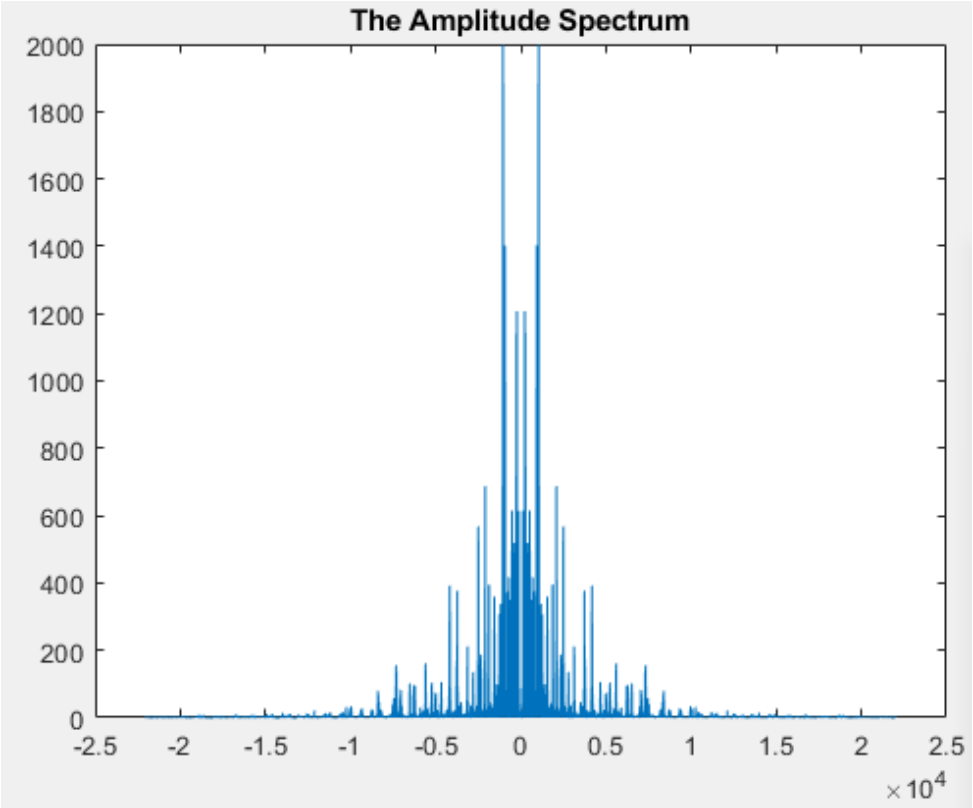

Figure 2.3.2: Amplitude spectrum of $x_d[n]$

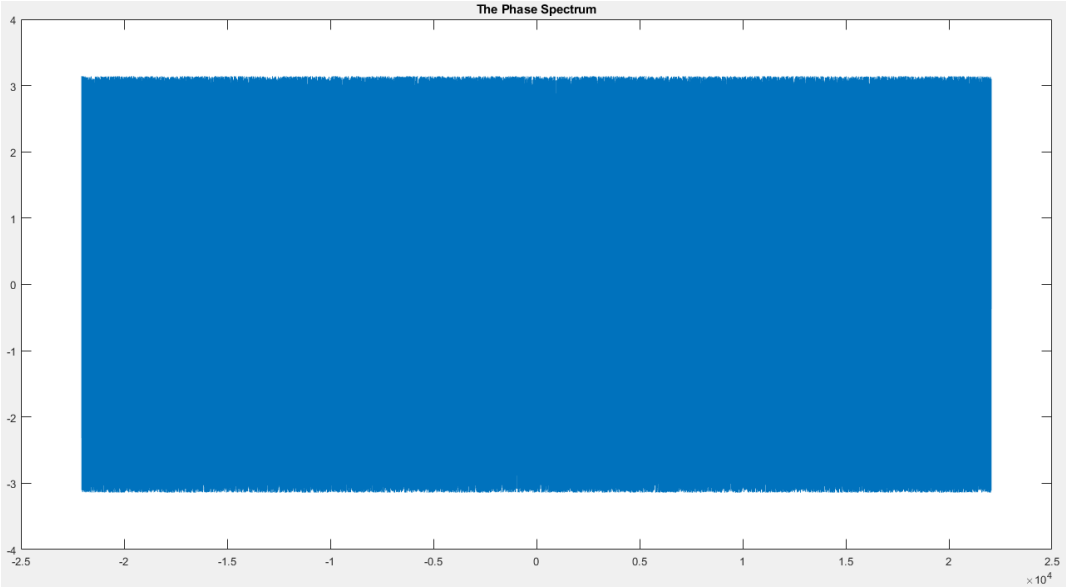

Figure 2.3.3: Amplitude spectrum of $x_d[n]$ when factor increases

When playing the content of $\boldsymbol{x_d}$, $\boldsymbol{x_d}$ is faster than $x_{20}[n]$ but has more and obvious distortion in sound.

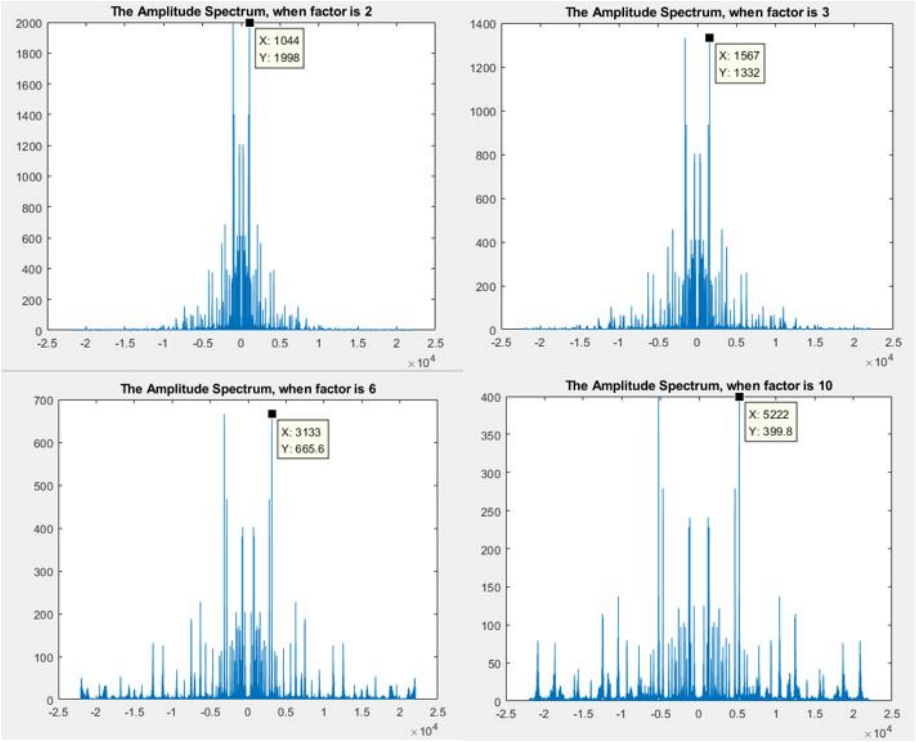

Figure 2.3.4: Phase spectrum of $x_d[n]$

The parameter factor controls the speed of playing and Figure 2.3.3 show the Amplitude spectrum of $x_d[n]$ when factor increases. We can observe that the frequency would be more distributed when the factor increases and the frequency carries most of the signal energy will decrease. When it played faster (n increases), the music will have more and obvious distortion.

Reference

[1] K. Raeen. (2018, Aug.). *A Study of The Gibbs Phenomenon in Fourier Series and Wavelets* [Online]. Available: https://math.unm.edu/~crisp/students/kouroshMSthesis.pdf

[2] Wikipedia. (2019, May.11). *Gibbs phenomenon* [Online]. Available: https://en.wikipedia.org/wiki/Gibbs_phenomenon