Санкт-Петербургский государственный политехнический университет Петра Великого

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Лабораторная работа

Гармоники

Работу выполнил студент
3-го курса, группа 3530901/80201
Сахибгареев Рамис Ринатович

Преподаватель:
Богач Наталья Владимировна

Санкт-Петербург 2021

Contents

List of Figures

Listings

1 Part 1: Examples execution

In this part we need to execute every part of ”chap2” file. By executing it we can take a brief look on the triangle and square signals, alisaing effect.

After executing every input in ”chap2” file no problems was found (Figure 1).

Aliasing interaction

The following interaction explores the effect of aliasing on the harmonics of a sawtooth signal.

In [25]:
```
def view_harmonics(freq, framerate):
    """Plot the spectrum of a sawtooth signal.

    freq: frequency in Hz
    framerate: in frames/second
    """
    signal = SawtoothSignal(freq)
    wave = signal.make_wave(duration=0.5, framerate=framerate)
    spectrum = wave.make_spectrum()
    spectrum.plot(color='C0')
    decorate(xlabel='Frequency (Hz)', ylabel='Amplitude')
    display(wave.make_audio())
```

In [26]:
```
from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets

slider1 = widgets.FloatSlider(min=100, max=10000, value=100, step=100)
slider2 = widgets.FloatSlider(min=5000, max=40000, value=10000, step=1000)
interact(view_harmonics, freq=slider1, framerate=slider2);
```

0:00 / 0:00

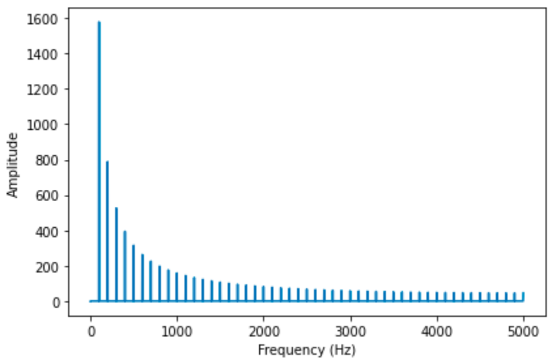

Figure 1: Everything is working fine

2 Part 2: SawtoothSignal

In this part we need to create a `SawtoothSignal` class, that extends the `signal` class and provides an `evaluate` function. After it's done we can create its spectrum and compare it to the spectrum of triangle and square signals.

Firstly we need to import required libraries to the project (Listing 1)

```
from thinkdsp import *
import numpy as np

class SawtoothSignal(Sinusoid):
    def evaluate(self, ts):
        graph = self.freq * ts + self.offset / np.pi / 2

        cutoff, _ = np.modf(graph) # create a cutoff
        ys = normalize(unbias(cutoff), self.amp)
        return ys

```

Listing 1: Sawrooth class definition

Next, we can check is our class works correctly (Listing 2, Figure 2).

```
sawtooth = SawtoothSignal()
sawtooth_wave = sawtooth.make_wave(sawtooth.period * 3, framerate=40000)
sawtooth_wave.plot()
decorate(xlabel='Time (s)')

```

Listing 2: Sawtooth wave plot code

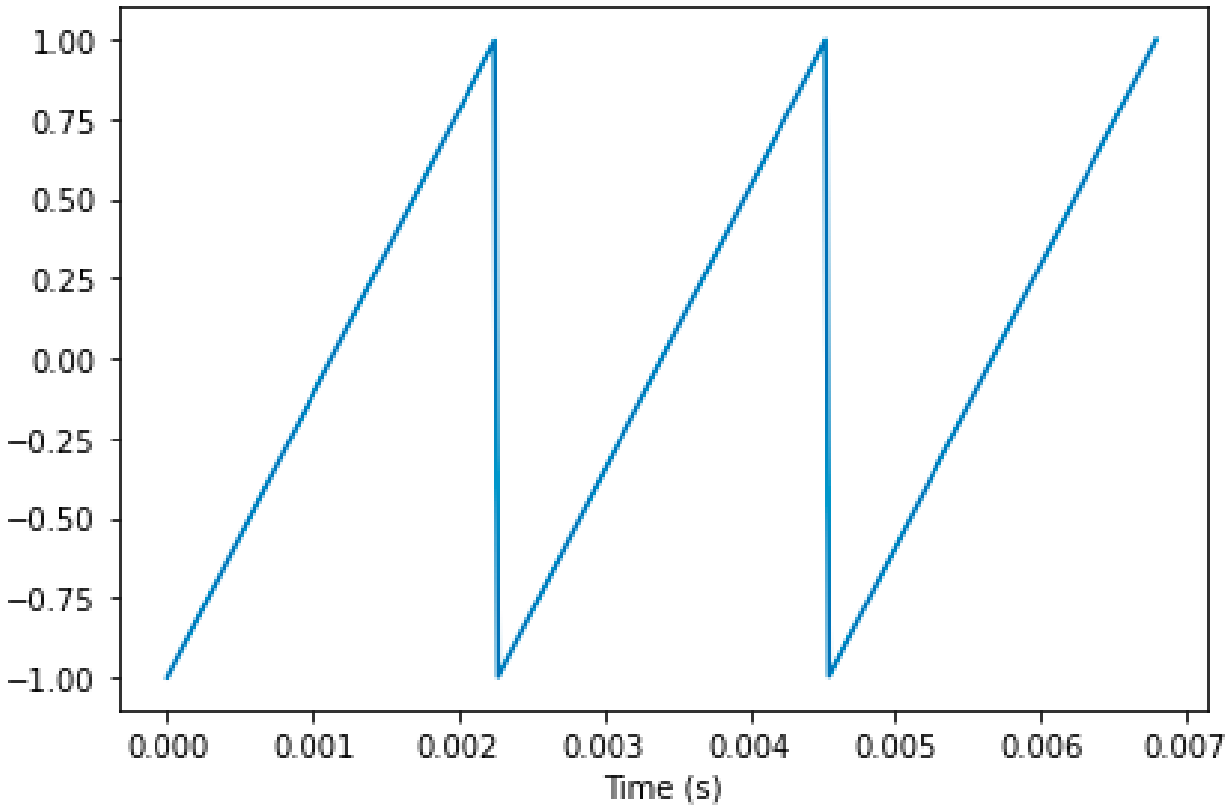

Figure 2: Sawtooth's wave's plot

Next, spectrum of the created signal was created (Listing 3, Figure 3).

```
sawtooth_wave.make_spectrum().plot()
decorate(xlabel='Frequency (Hz)')
```

Listing 3: Sawtooth's spectrum computation

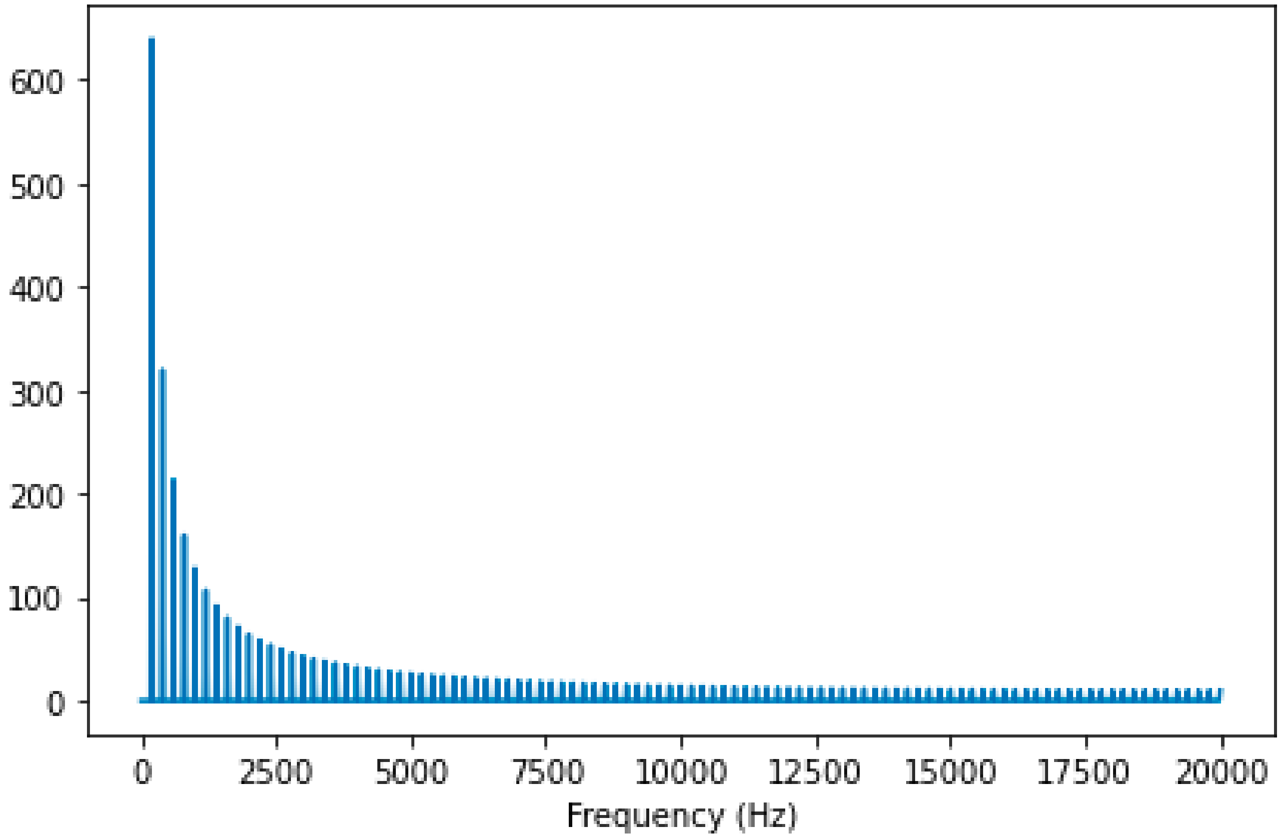

Figure 3: Sawtooth's spectrum

Compared to the triangle signal's spectrum and square signal's spectrum (Listring 4, Figure 4), our sawtooth signal has spikes both on even and odd base frequencies factors, and it decreases linearly from its frequency.

```
SquareSignal(200).make_wave(duration=0.5, framerate=10000).make_spectrum().
plot(color='green')
decorate(xlabel='Frequency (Hz)')
TriangleSignal(200).make_wave(duration=0.5, framerate=10000).make_spectrum().
plot(color='red')
decorate(xlabel='Frequency (Hz)')
sawtooth_wave.make_spectrum().plot(color='blue')
decorate(xlabel='Frequency (Hz)')
```

Listing 4: Spectrum comparison code

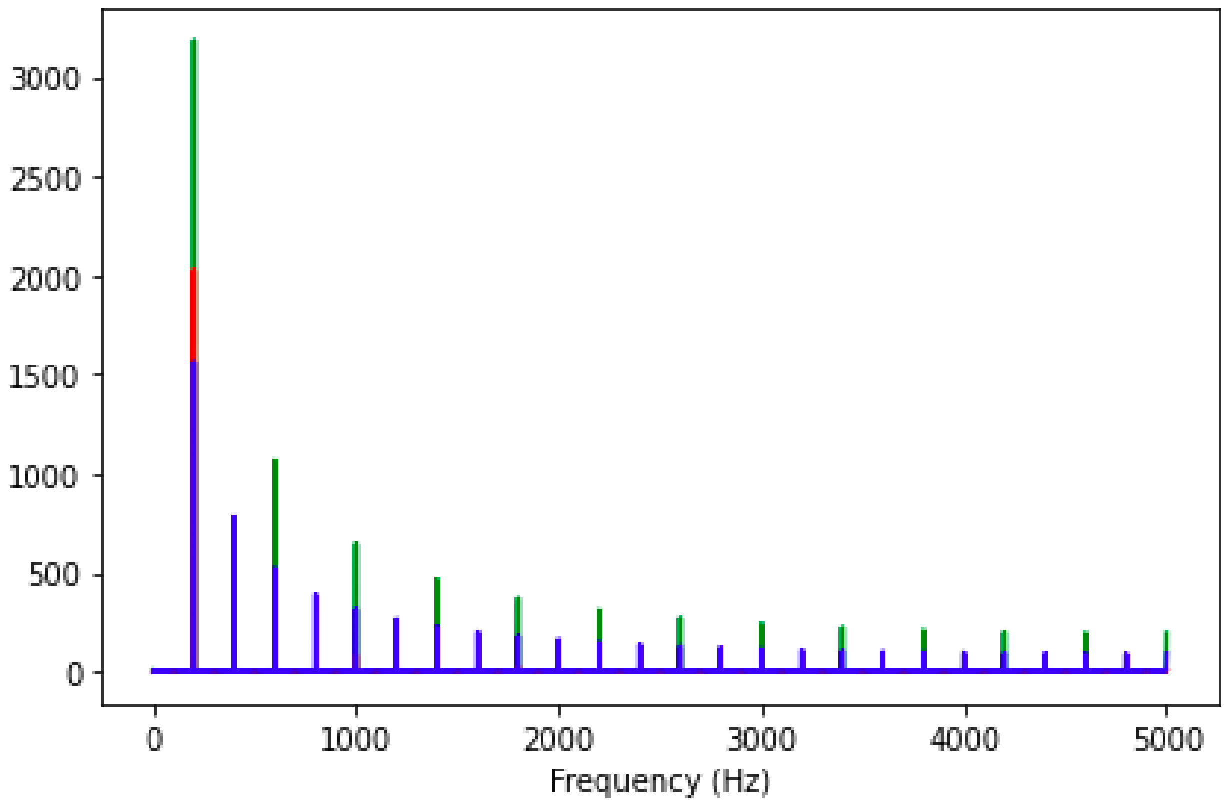

Figure 4: Spectrum comparison

3 Part 3: Aliasing effect

In this part we need to explore the aliasing effect.

To reproduce this effect let's create a wave of frequency 1100 Hz and framerate of 10000 Hz (Listing 5, Figure 5). To comparison, this how this signal looks next to the signal with same frequency, but with framerate of 96000 Hz (Figure 6).

```
wave = SquareSignal(1100).make_wave(duration=9.6, framerate=10000)
wave_clear = SquareSignal(1100).make_wave(duration=1, framerate=96000)
wave.make_spectrum().plot(color='red')
wave_clear.make_spectrum().plot(color='blue')

```

Listing 5: Waves creation

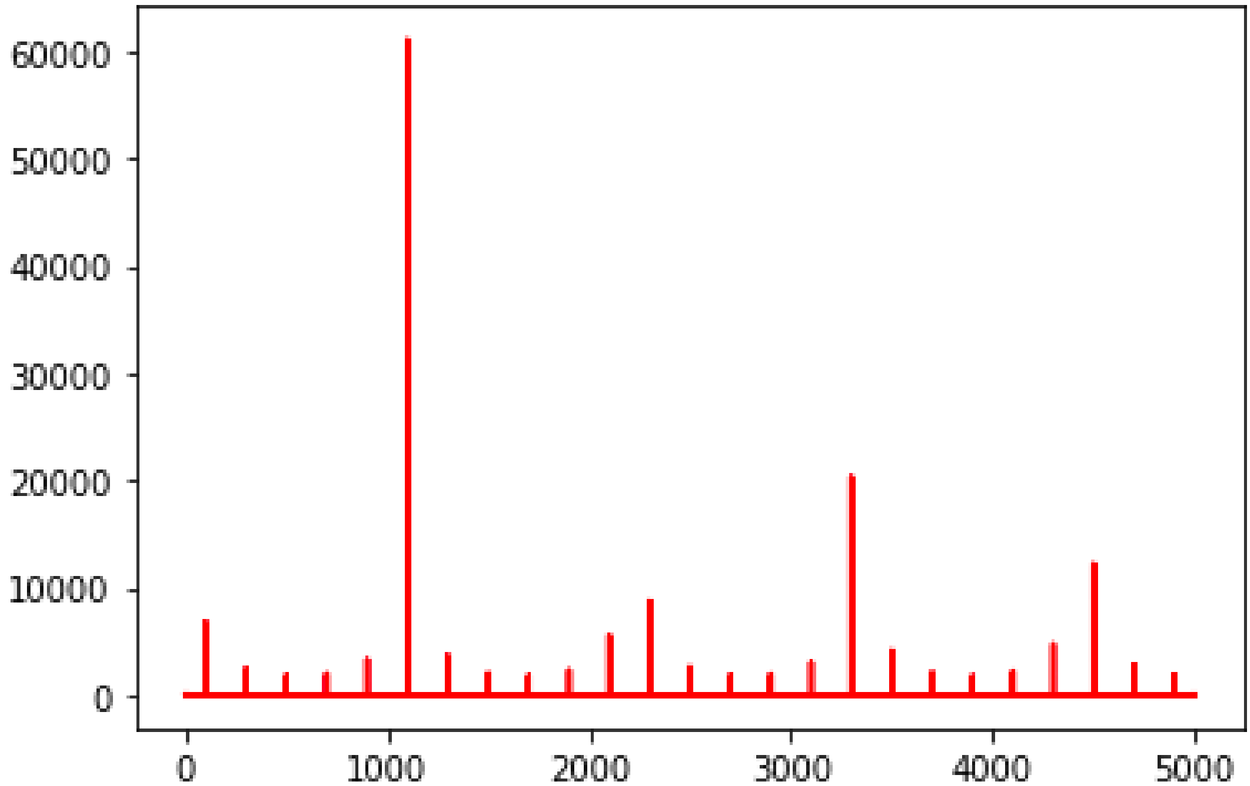

Figure 5: Aliasing effect

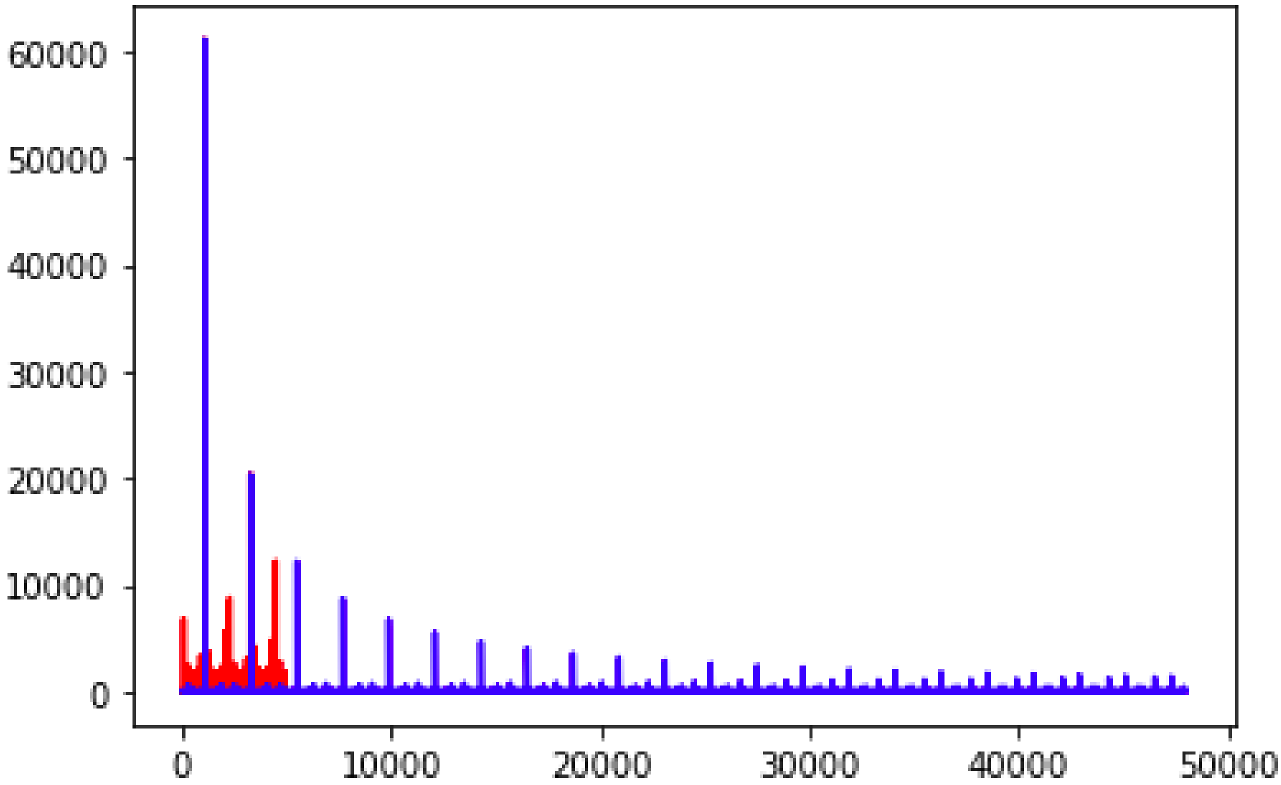

Figure 6: Aliasing effect comparison

By listening them, we can clearly notice the differenct: aliased signal is more ”dirty” and ”noisy”.

4 Part 4: Spectrum's HS

In this part we need to explore, what is the HS values of the spectrum.

To do it, let's create triangle signal with frequency of 440 Hz, length of the signal doesn't matter. Next, let's set its hs[0] = 100 and check, what is the deference.

Code of this part - Listing 6

```
wave = TriangleSignal(440).make_wave(duration= 10 / 440, framerate=48000)
spect = wave.make_spectrum()
print(spect.hs[0])
spect.hs[0] = 100
print(spect.hs[0])
spect_w = spect.make_wave()
spect_w.normalize()
spect_w.plot(color='red')
wave.plot(color='blue')

```

Listing 6: HS operations

`Spectrum.hs[0]=(-9.126033262418787e-14+0j)`, amplitude is a length, angle is a phase.

As we can see, the is no difference between those signals (Figure 7)

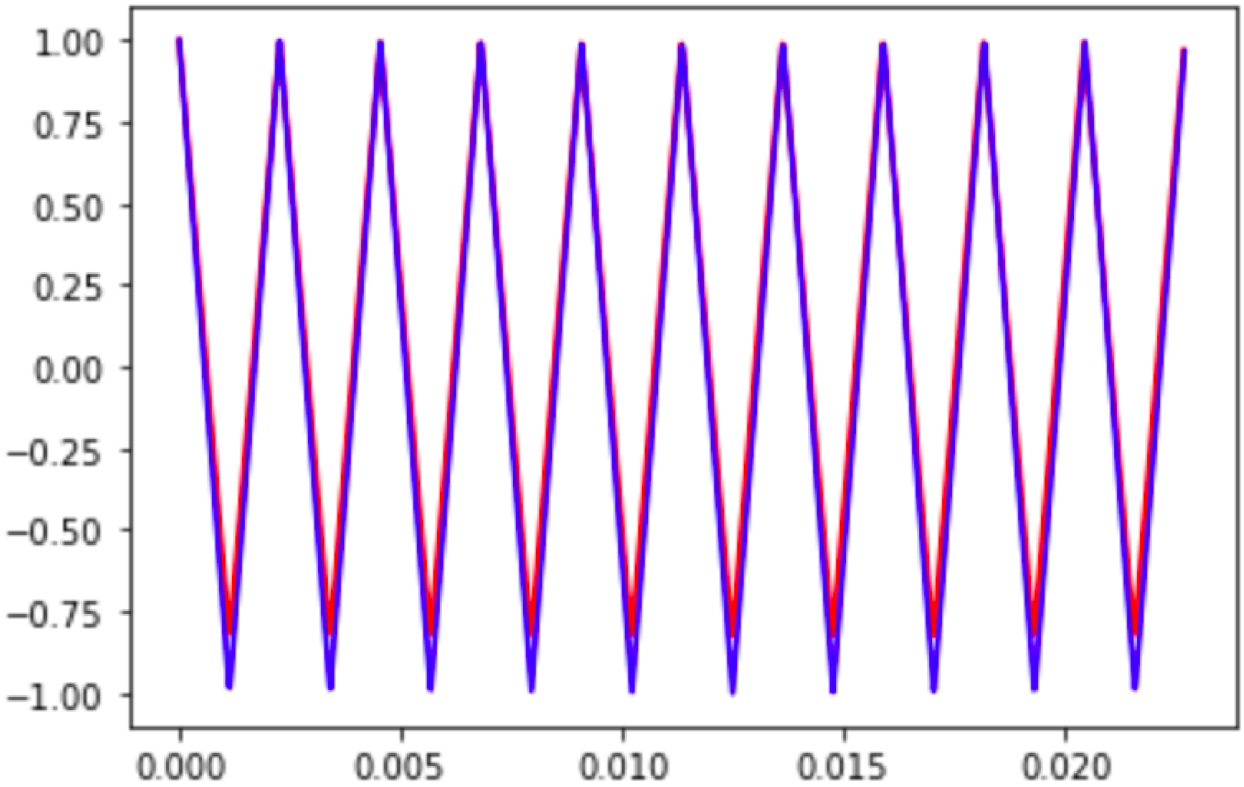

Figure 7: Function call result

5 Part 5: Muffling the signal

In this part we need to create a function, to muffle high frequencies of the wave by dividing hs by the frequency.

Code of this part - Listing 9

```
def spectrum_muffle(spectrum):
    spectrum.hs[0] = 0
    spectrum.hs[1:] /= spectrum.fs[1:]
    return spectrum

```

Listing 7: Definition of the function

Code of plotting: Listing 8.

```
wave = TriangleSignal(100).make_wave(duration=1, framerate=10000)
spec = wave.make_spectrum()
spec.plot(color='red', high=2000)
spectrum_muffle(spec)
spec.scale(100)
spec.plot(color='blue', high=2000)
decorate(xlabel='Frequency (Hz)')

```

Listing 8: Function usage

As we can see, high frequencies are muffled (Figure **??**). After listening we can be sure in it.

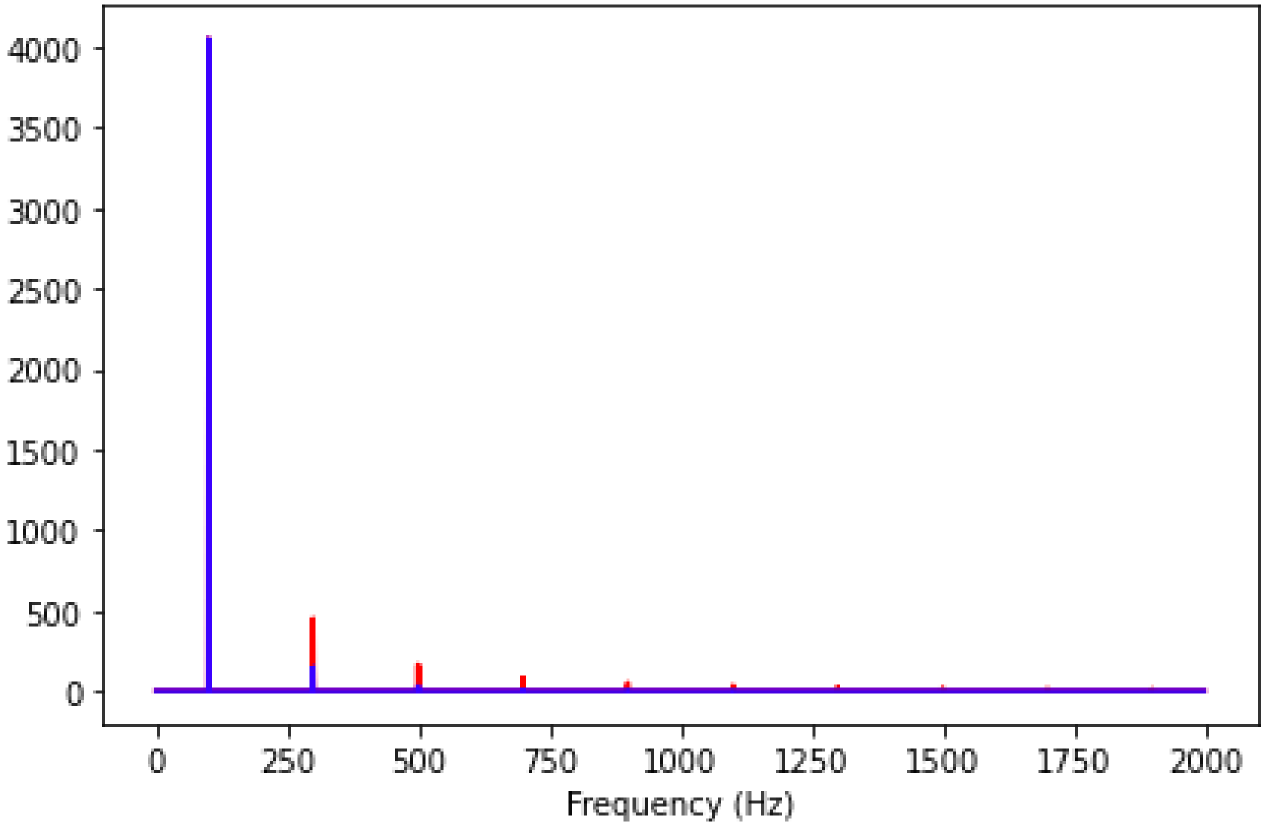

Figure 8: Function call result

6 Part 6: New signal

In this part we need to find a signal, which similar to sawtooth signal, but its amplitude decreases by $1/f^2$ instead of $1/f/$.

We can use sawtooth signal as base, and simply divide it's amplitudes by frequencies once more using function, declared in the previous part. Code is next: Listing **??**. Result is next: Figure **??**. We need to use such big framerate because of aliasing effect.

```
s = SawtoothSignal(200).make_wave(duration=1, framerate=1000000).make_spectrum()
spectrum_muffle(s).plot(high = 5000)

```

Listing 9: Signal and spectrum creation

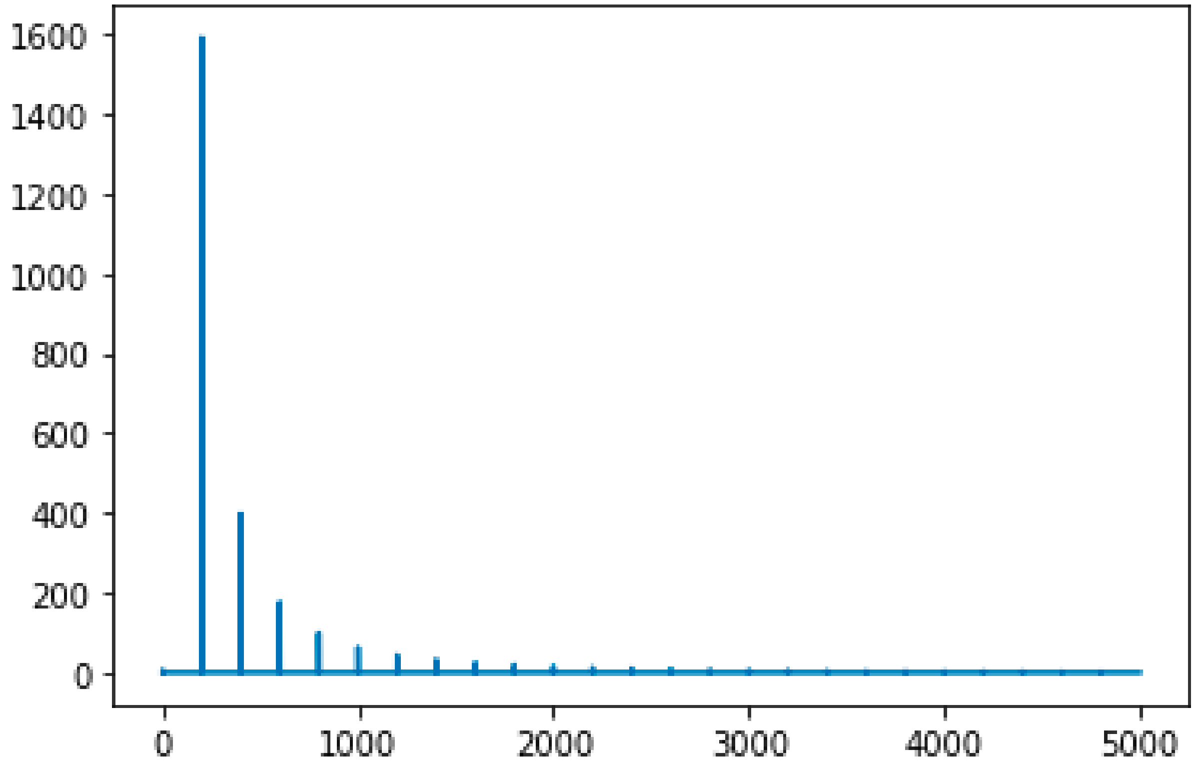

Figure 9: Spectrum of the signal

7 Conclusion

More advanced skills and knowledge of signals, waves and spectrum was acquired. Three more default signal types - triangle, square and sawtooth signals, which has specific features. Aliasing effect was explored. It has a lot of effect on decomposing the signal into the set of sine signals, that's why we need to have a higher framerate to not loss the data. Spectrum components was learned and used to muffle the signal.