Санкт-Петербургский государственный политехнический
университет Петра Великого

**Высшая школа интеллектуальных систем и
суперкомпьютерных технологий**

Лабораторная работа

Сигналы и звуки

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: Check

In this part we need to make sure, that every listing is able to be executed. To do it, we can simply try to execute every example.

After executing every input in ”chap1” file no problems was found (Figure 1).

ThinkDSP

This notebook contains code examples from Chapter 1: Sounds and Signals

Copyright 2015 Allen Downey

License: Creative Commons Attribution 4.0 International

Think DSP module

`thinkdsp` is a module that accompanies *Think DSP* and provides classes and functions for working with signals.

Documentation of the thinkdsp module is here.

In [7]:
```
# Get thinkdsp.py

import os

if not os.path.exists('thinkdsp.py'):
    !wget https://github.com/AllenDowney/ThinkDSP/raw/master/code/thinkdsp.py
```

Signals

Instantiate cosine and sine signals.

In [8]:
```
from thinkdsp import CosSignal, SinSignal

cos_sig = CosSignal(freq=440, amp=1.0, offset=0)
sin_sig = SinSignal(freq=880, amp=0.5, offset=0)
```

Plot the sine and cosine signals. By default, `plot` plots three periods.

In [9]:
```
from thinkdsp import decorate

cos_sig.plot()
decorate(xlabel='Time (s)')
```

Figure 1: Everything is working fine

2 Part 2: Processing

In this part we need to download any sound, that contains a clear pitch. Halzion by YOASOBI was selected. We need to select a half second interval with consistent pitch level, compute its spectrum. After harmonic filtration should be performed and spectrum should be converted back to the wave.

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

```
            import os
            if not os.path.exists('thinkdsp.py'):
                !wget https://github.com/AllenDowney/ThinkDSP/raw/master/code/
    thinkdsp.py
            from thinkdsp import read_wave
            from thinkdsp import decorate

```

Listing 1: Libraries import

After imports done we can easily create a wave of the .wav file (Listing 2). Plot of the wave looks in the next way (Figure 2).

```
            wave = read_wave('sound/halzion.wav')
            wave.normalize()
            wave.make_audio()

```

Listing 2: Wav file conversion to wave instance

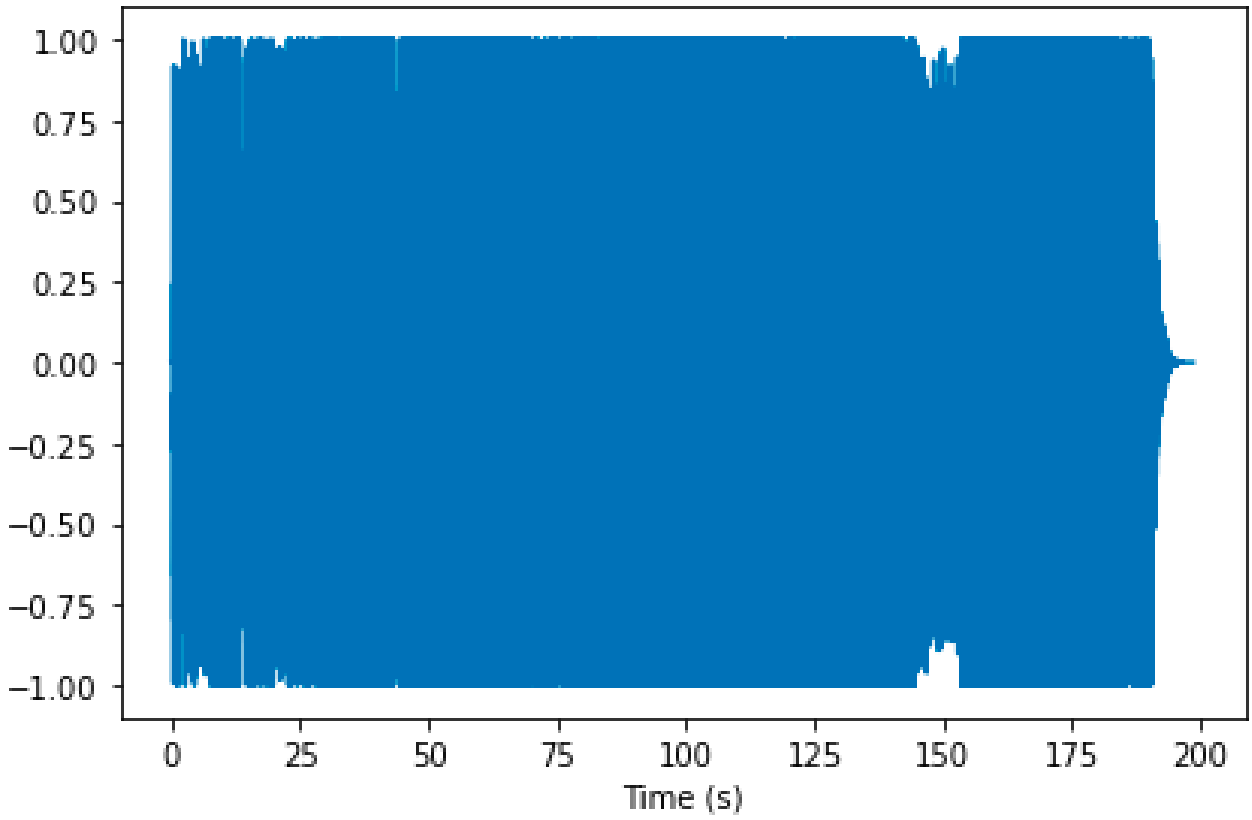

Figure 2: Halzion's wave's plot

Selected half second segment starts at 1:42 (Listing 3).

```
segment = wave.segment(start=102, duration=0.5)
segment.make_audio()
```

Listing 3: Segment extraction

Using `segment.plot` next plot was acquired (Figure 3). Spectrum was acquired using next functions (Listing 4). Spectrum of the segment can be viewed on the Figure 4.

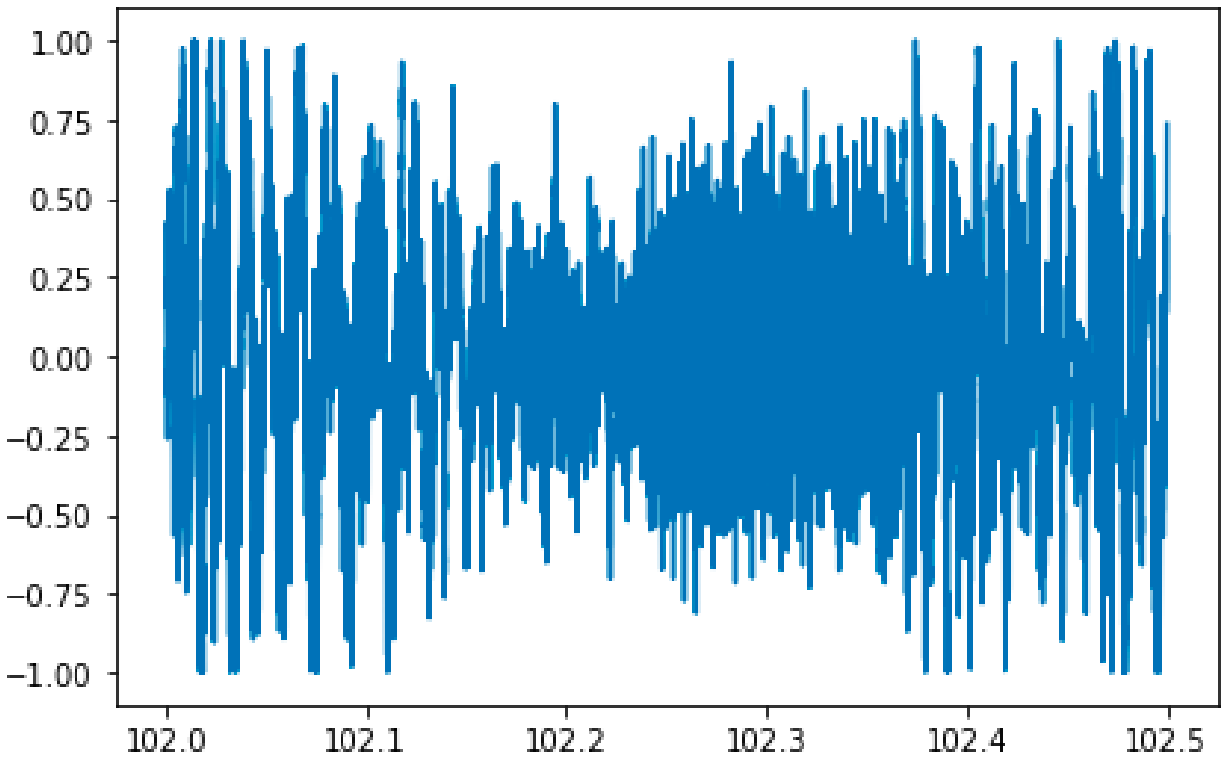

Figure 3: Segment's plot

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

```

Listing 4: Segment spectrum calculation

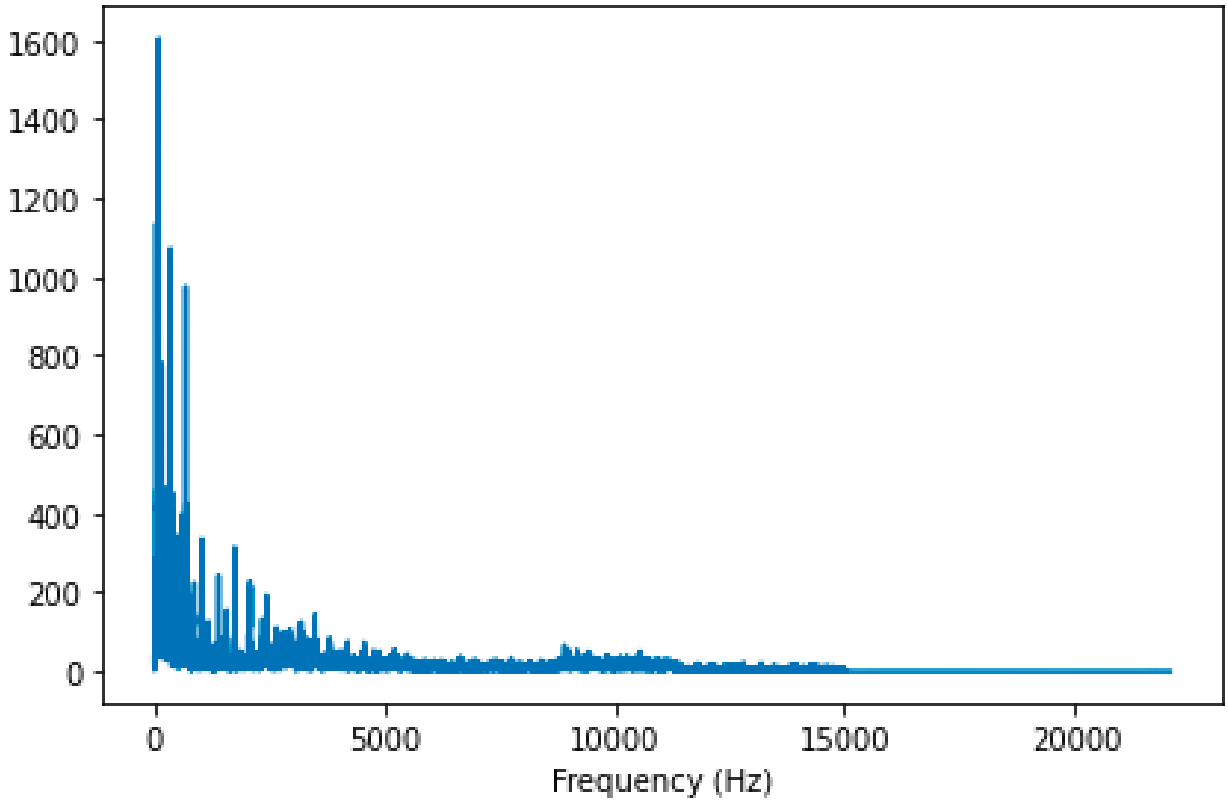

Figure 4: Segment's spectrum

We can see, that higher frequencies are unused, so we can zoom in more dense area between 0 Hz and 5000 Hz (Figure 5). We can clearly see spikes on the plot.

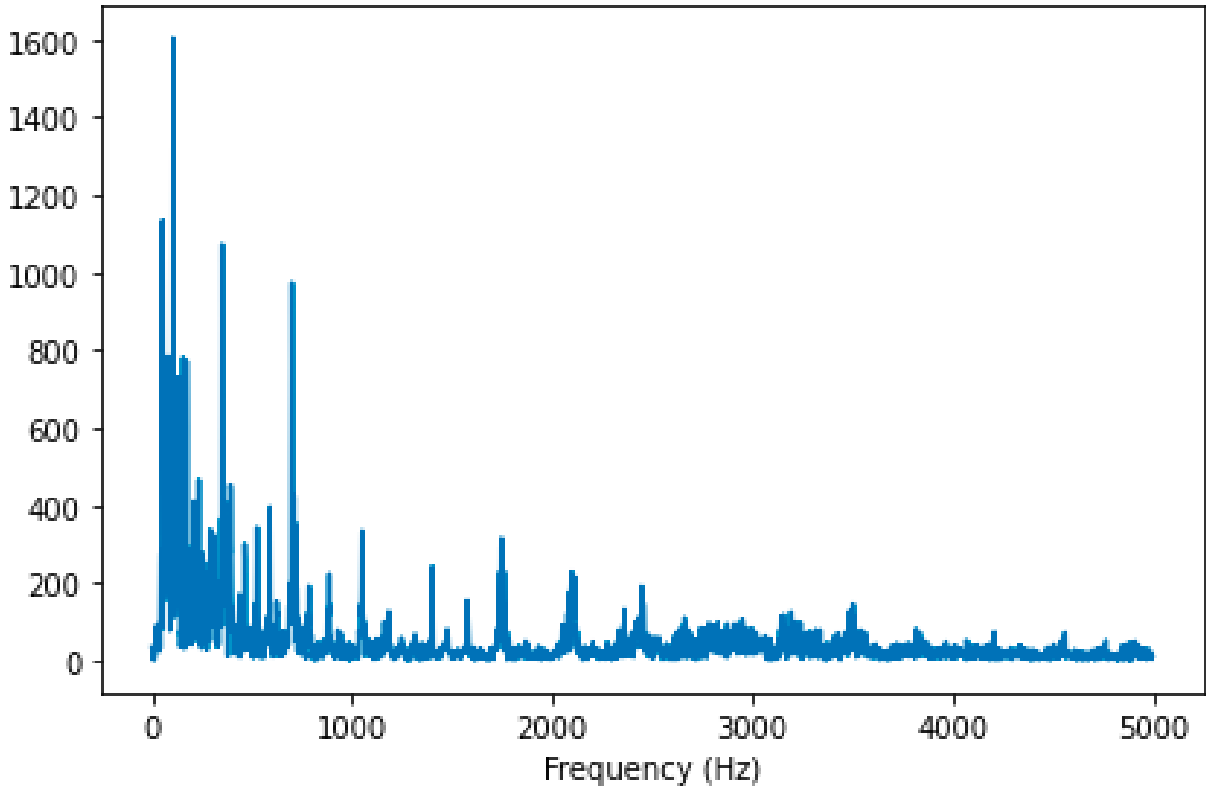

Figure 5: Spectrum zoom in

To filtrate spectrum next code Listing 5 was used. Result can be observed on the Figure 6.

```
filtered = spectrum.high_pass(cutoff=500, factor=0.5)
filtered = spectrum.make_wave()
filtered.normalize()
filtered.plot()
decorate(xlabel='Time (s)')

```

Listing 5: Spectrum filtering

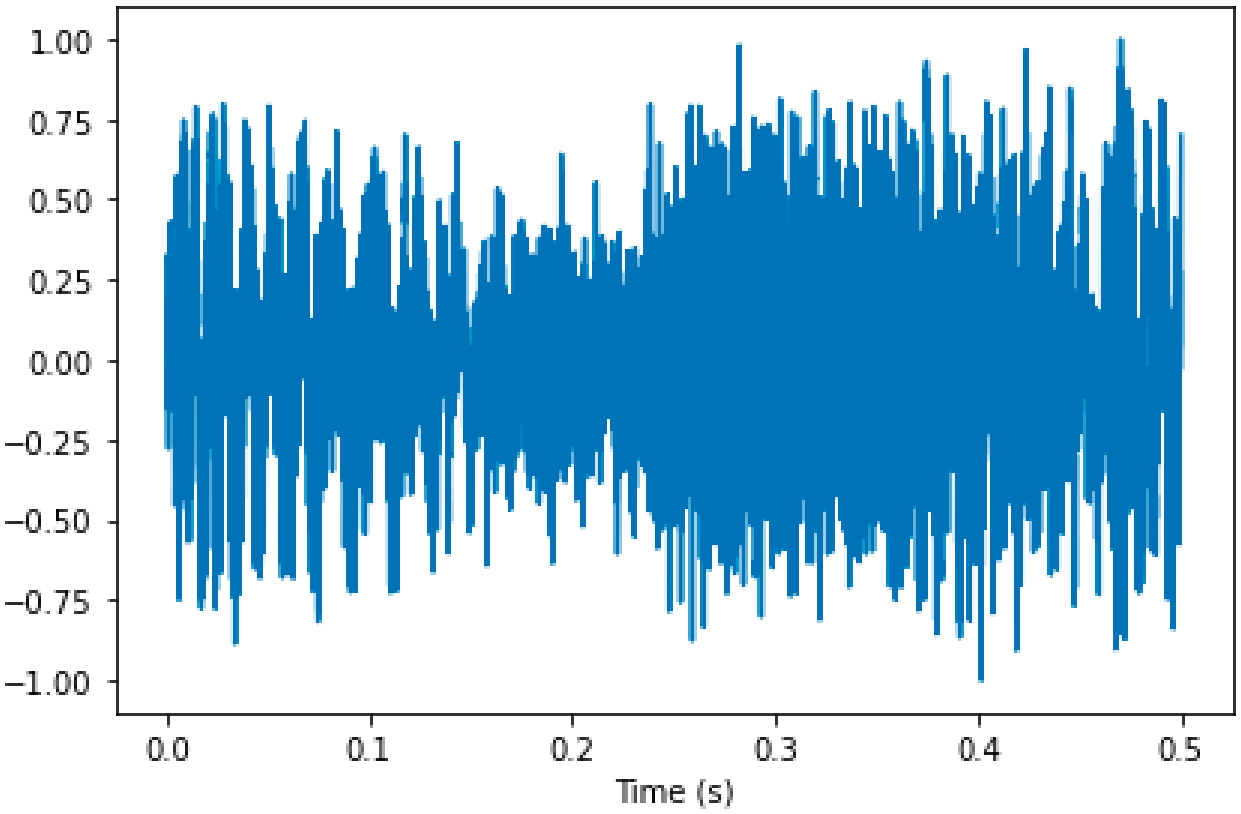

Figure 6: Filtered spectrum's wave plot

After listening filtered sound no differences was found, but background sounds became quieter.

3 Part 3: Combining

In this part we need to create 2 signals by summing a sine wave signal and a cosine wave signal. This signal should be converted to an audio and to a spectrum.

Two signals was created. One with frequency of 555 Hz and other with frequency of 742 Hz (Listing 6). Result can be observed on the Figure 7.

```
from thinkdsp import CosSignal , SinSignal
cos_sig = CosSignal(freq=555, amp=0.82)
sin_sig = SinSignal(freq=742, amp=0.42)
cos_sig.plot()
sin_sig.plot()
decorate(xlabel='Time (s)')

```

Listing 6: Signal creation

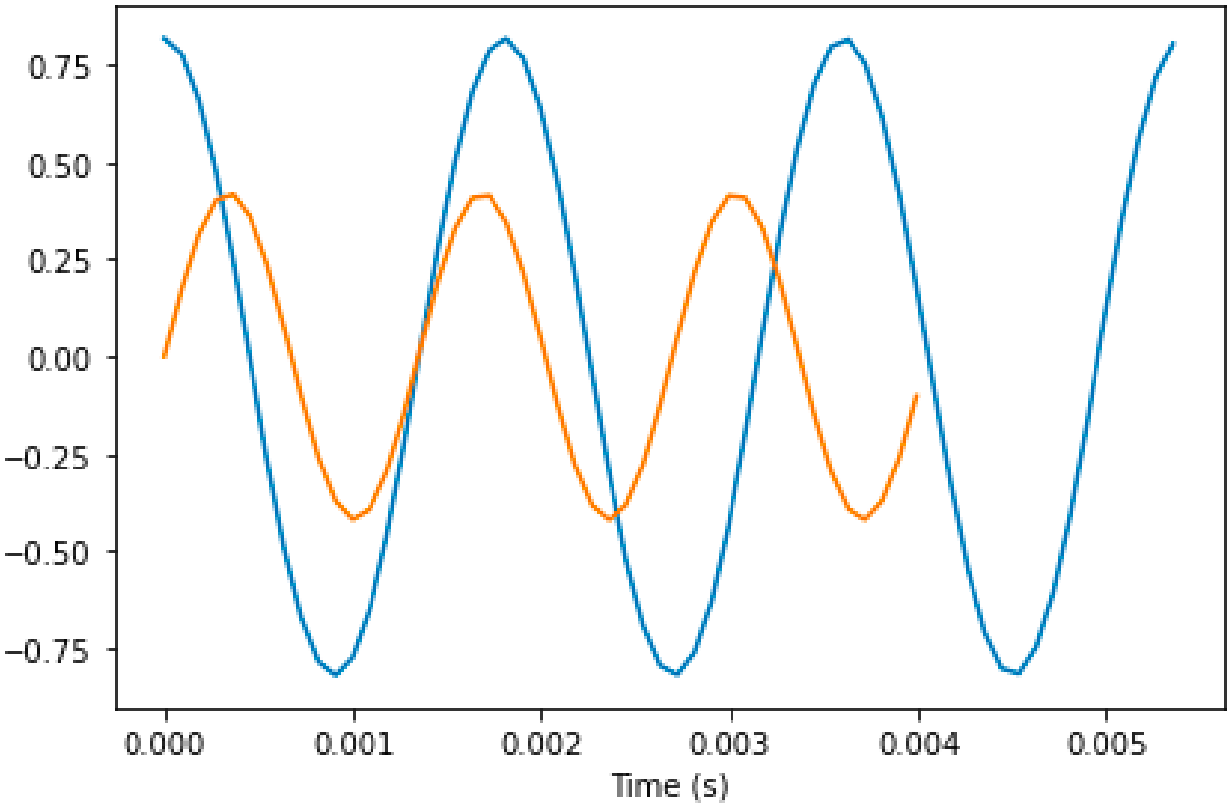

Figure 7: Created signals

By summing them (Listing 7) next result was achieved (Figure 8).

```
signal = cos_sig + sin_sig
signal.plot()
decorate(xlabel='Time (s)')

```

Listing 7: Summing the signals

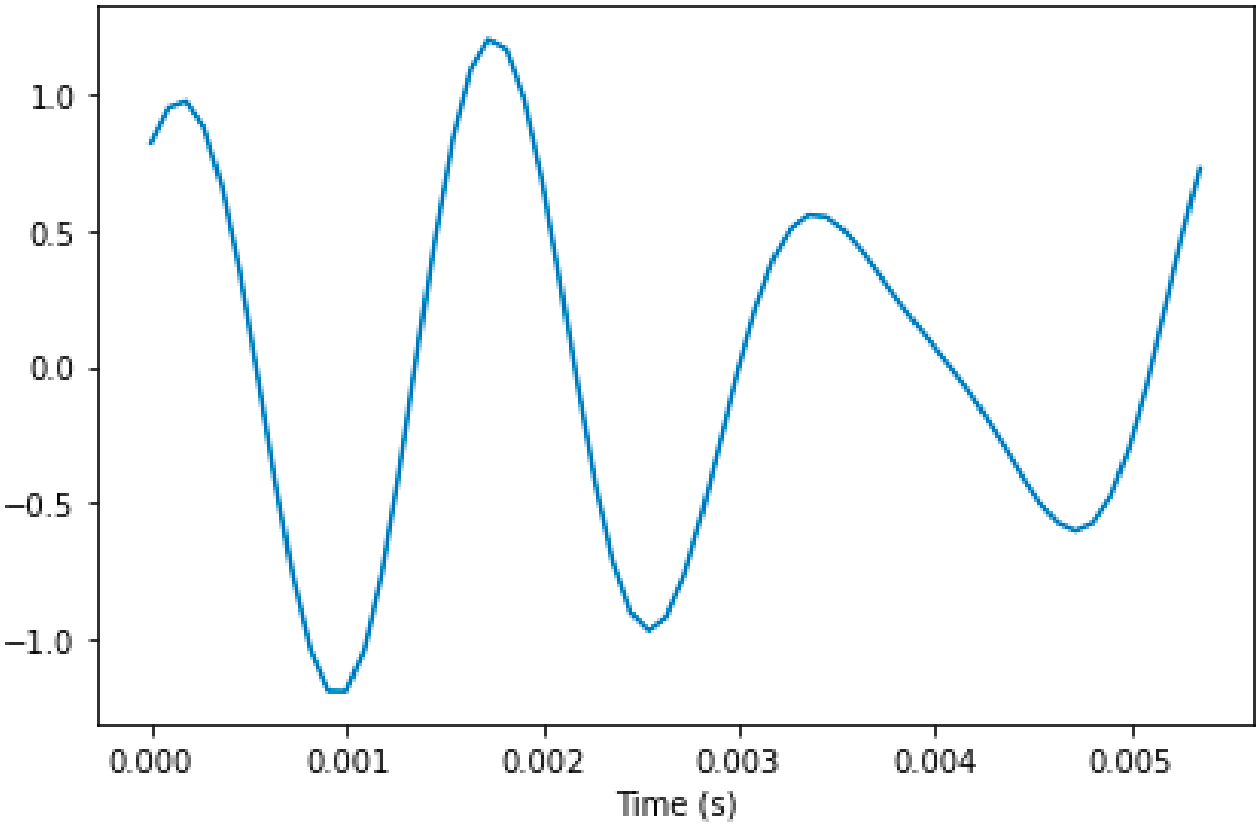

Figure 8: Summing the signals

Wave and spectrum was created out of the signal using next code Listing 8. Computed spectrum is next: Figure 9. We can clearly see 2 spikes with the frequencies of 555 and 742 Hz.

```
1           sig_wave = signal.make_wave(duration = 10, start = 0, framerate =
    400000)
2           sig_spectrum = sig_wave.make_spectrum()
3           sig_spectrum.plot(high=1000)
4           decorate(xlabel='Frequency (Hz)')
5
```

Listing 8: Creating a wave and a spectrum

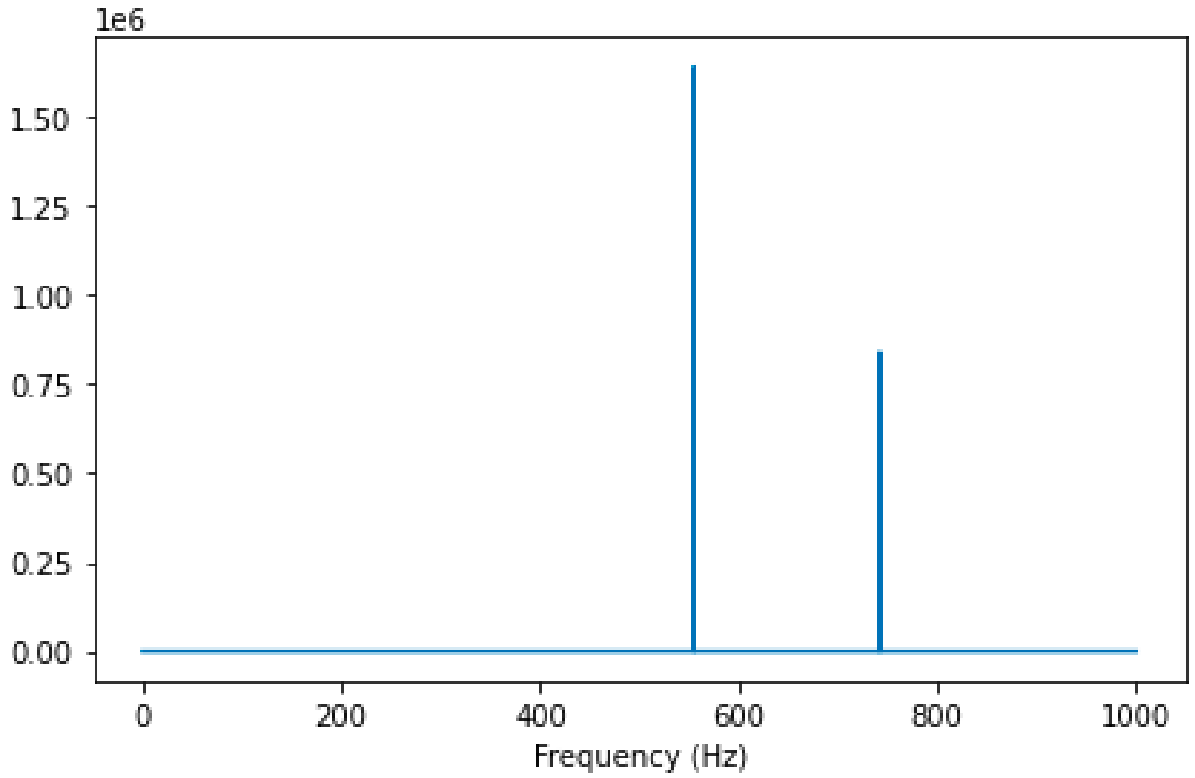

Figure 9: Spectrum of combined signals

By adding another signal to the existing one we are adding a new spile to the spectrum: Figure 10

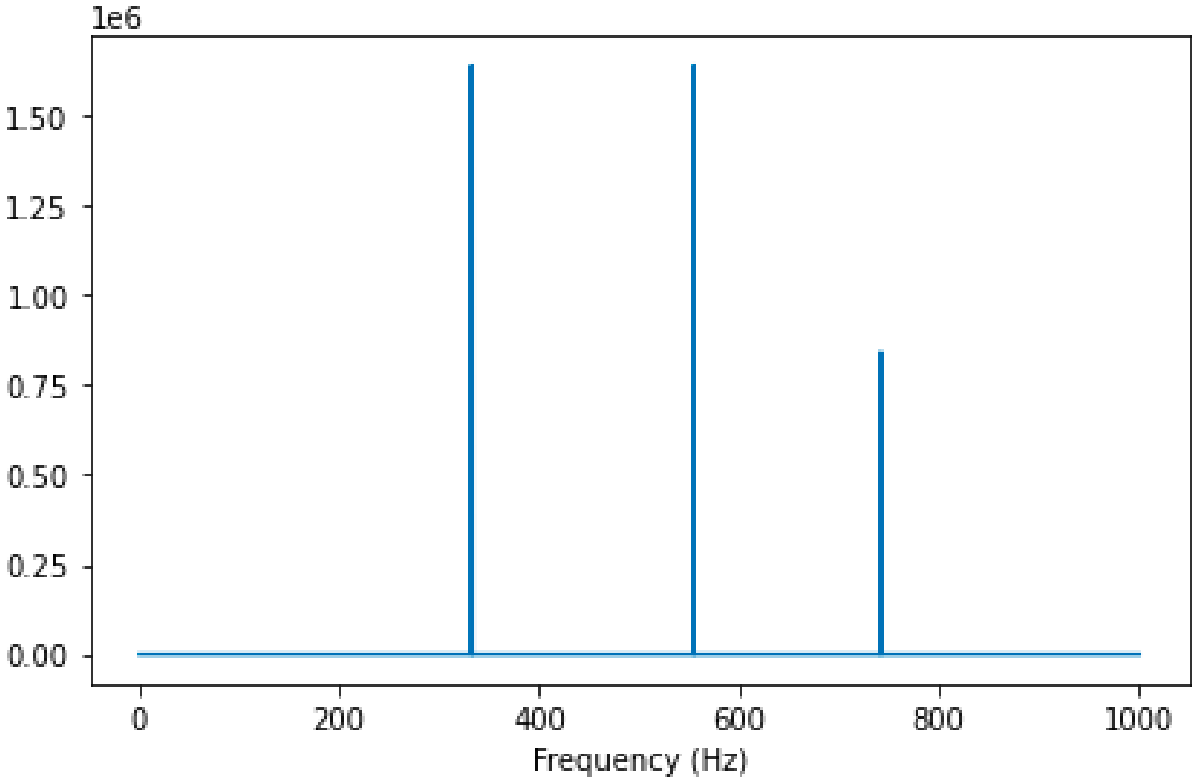

Figure 10: Spectrum with one more added signal

By adding low frequency signal, the sound became more flat and pleasant.

4 Part 4: Stretch

In this part we need to create function `stretch` to slow down or speed up the input wave by changing its `ts` and `framerate`.

Code of the function is next: Listing 9

```
def stretch(wave, factor):
wave.ts *= factor
wave.framerate /= factor

```

Listing 9: Definition of the function

Let's speed up YOASOBI's song by 20 percent (Figure 11). Original sound's length is 3:18, after stretching we've got 2:38.

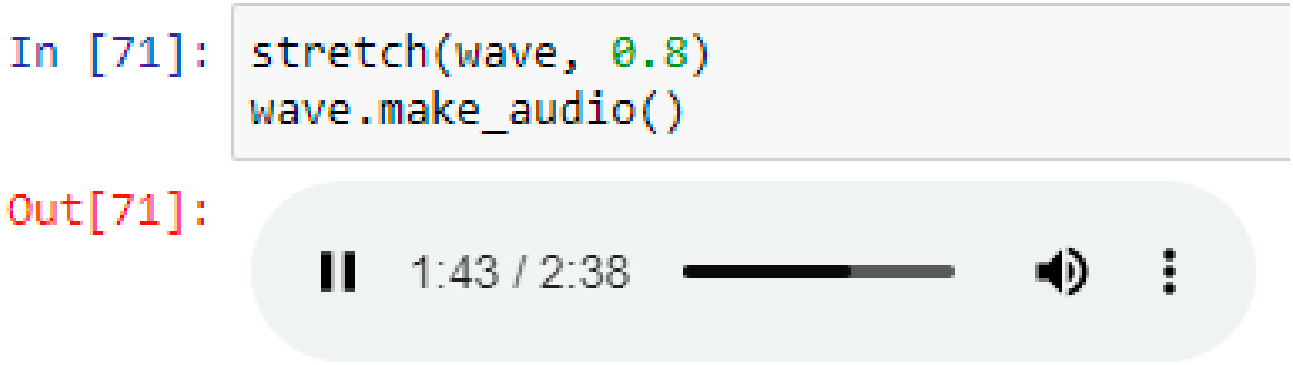

Figure 11: Function call result

5 Conclusion

Basic skills and knowledge of signals, waves and spectrum was acquired. Every signal can be decomposed into the set of frequencies and amplitudes of sine signal, that represents input signal. That allows to perform different operations on the signal, transport and store it.