Санкт-Петербургский государственный политехнический университет Петра Великого

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Лабораторная работа

Апериодические сигналы

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 ”chap3” file. By executing it we can take a brief look on the chirp signals and spectrograms and windows.

After executing every input in ”chap3” file no problems was found. Different windows was tested, they close to each other, but has different impact on the signal. (Listing 1, Figure 1).

```
    wave = signal.make_wave(duration)
    wave.window(np.bartlett(len(wave)))
    spectrum = wave.make_spectrum()
    spectrum.plot(high=880, color='red')
    decorate(xlabel='Frequency (Hz)')

    wave = signal.make_wave(duration)
    wave.window(np.blackman(len(wave)))
    spectrum = wave.make_spectrum()
    spectrum.plot( high=880, color ='green')
    decorate(xlabel ='Frequency (Hz)')

    wave = signal.make_wave(duration)
    wave.window(np.kaiser(len(wave), 10))
    spectrum = wave.make_spectrum()
    spectrum.plot(high =880, color ='blue')
    decorate(xlabel ='Frequency (Hz)')

    wave = signal.make_wave(duration)
    wave.window(np.hanning(len(wave)))
    spectrum = wave.make_spectrum()
    spectrum.plot(high =880, color ='#FF00FF')
    decorate(xlabel ='Frequency (Hz)')

```

Listing 1: Windows computation

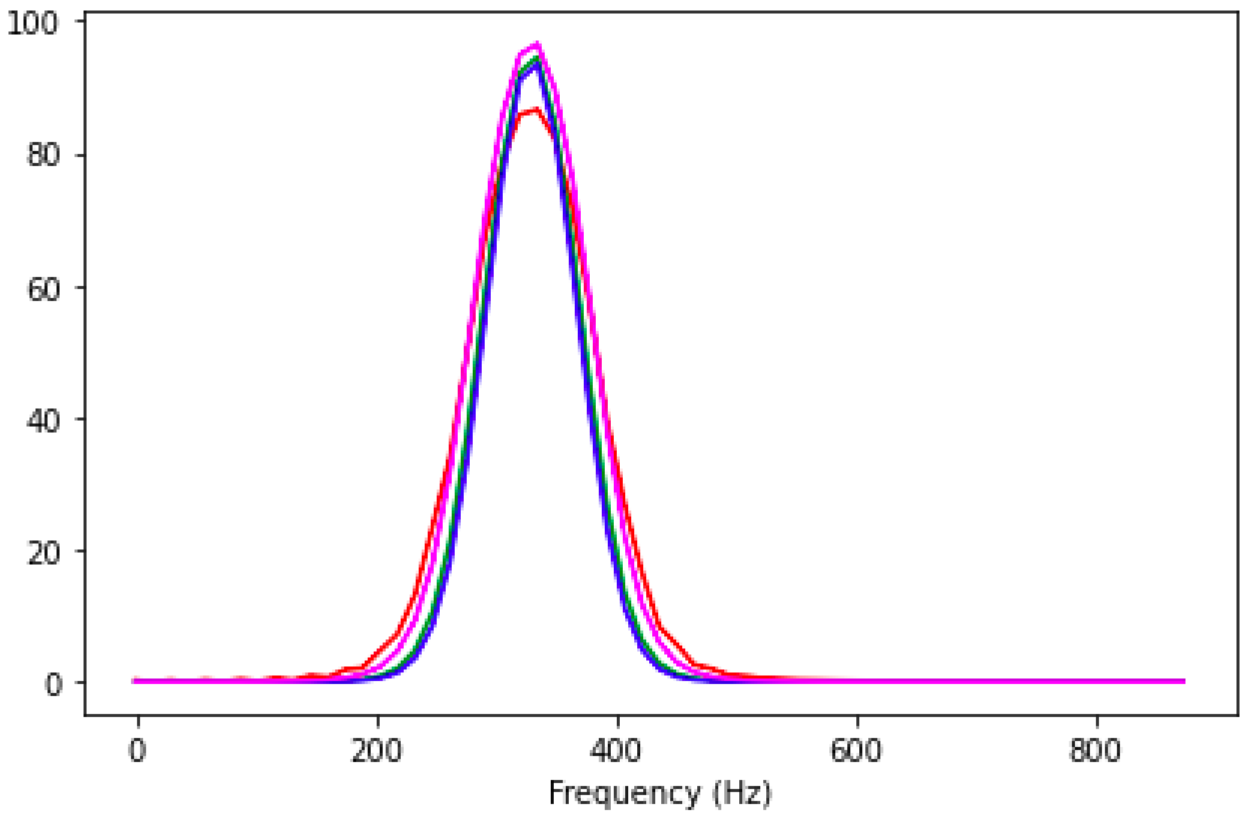

Figure 1: Different windows

2 Part 2: SawtoothChirp

In this part we need to create a `SawtoothChirp` class, that extends the `Chirp` class and provides an `evaluate` function. After it's done we can create its spectrogram.

Let's declare a class (Listing 2)

```python
from thinkdsp import *
import numpy as np

class SawtoothChirp(Chirp):
    def evaluate(self, ts):
        # freq change
        freqs = np.linspace(self.start, self.end, len(ts) - 1)
        # steps
        dts = np.diff(ts)
        dphis = PI2 * freqs * dts
        phases = np.cumsum(dphis)
        # apply default sawtooth code
        cycles = phases / PI2
        frac, _ = np.modf(cycles)
        ys = normalize(unbias(frac), self.amp)
        return ys

```

Listing 2: Sawtooth class definition

Next, we can check is our class works correctly (Listing 3, Figure 2).

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

Listing 3: Sawtooth wave plot code

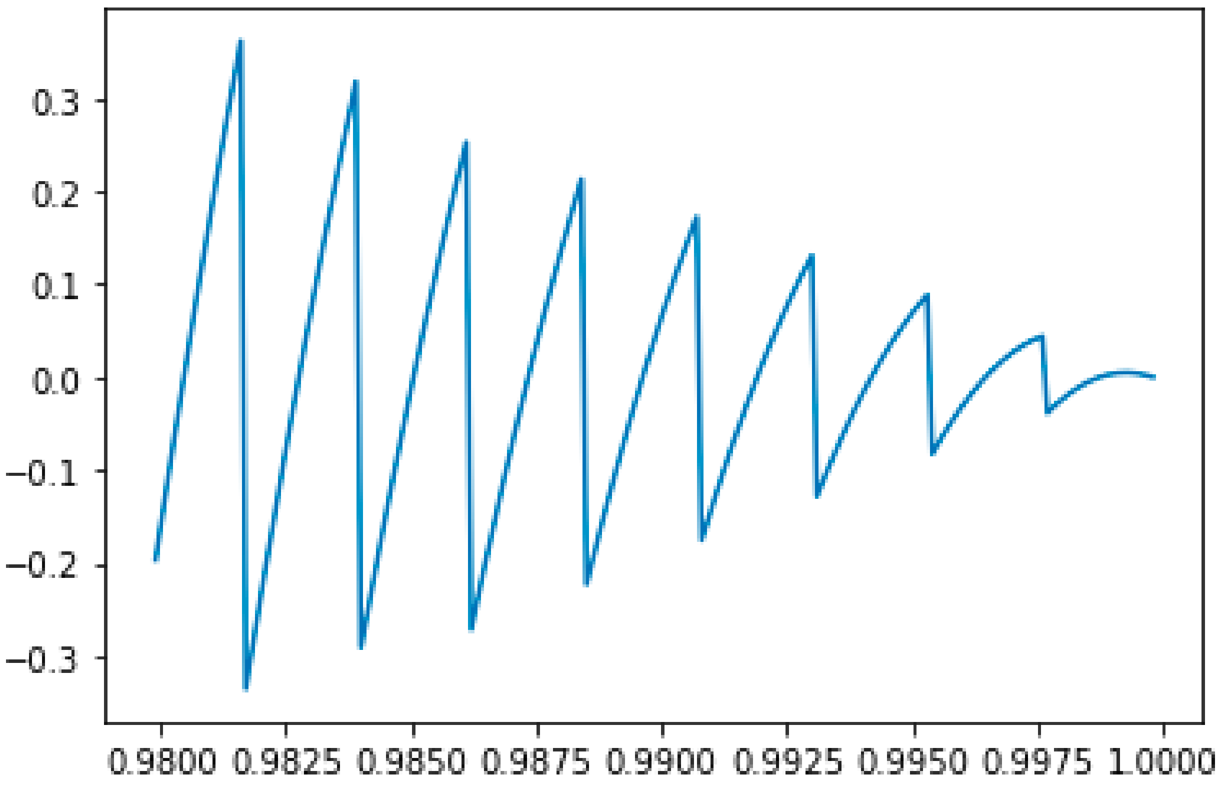

Figure 2: Sawtooth's wave's plot

Next, spectrogram for created signal was created (Listing 4, Figure 3).

```
s = wave.make_spectrogram(256)
s.plot()
decorate(xlabel='Time (s)', ylabel='Frequency (Hz)')
```

Listing 4: Sawtooth's spectrum computation

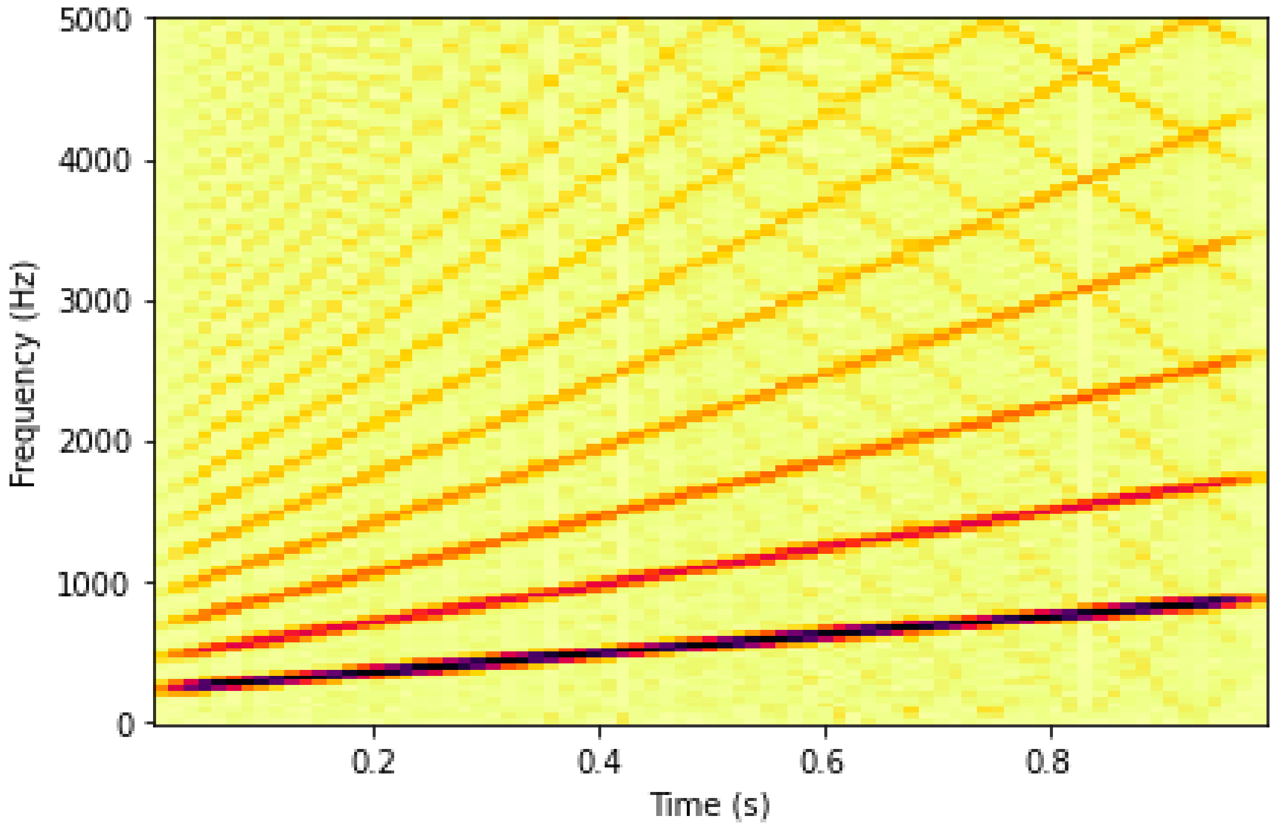

Figure 3: Sawtooth's spectrogram

3 Part 3: Changing Sawtooth

In this part we need to explore the aliasing effect on the Sawtooth chirp.

To reproduce this effect let’s create a sawtooth chirp of frequency 2500 Hz to 3000 Hz and framerate of 20000 Hz (Listing 5, Figure 4). We can clearly see wide main frequency components and a lot of noise. It happens because main components and aliase components are ”drifting” to the higher frequency with the time, and spectrum shows overall spectrum of the sound. We can make spectrogram clearer, by increasing the wave’s framerate (Figure 5).

```
    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

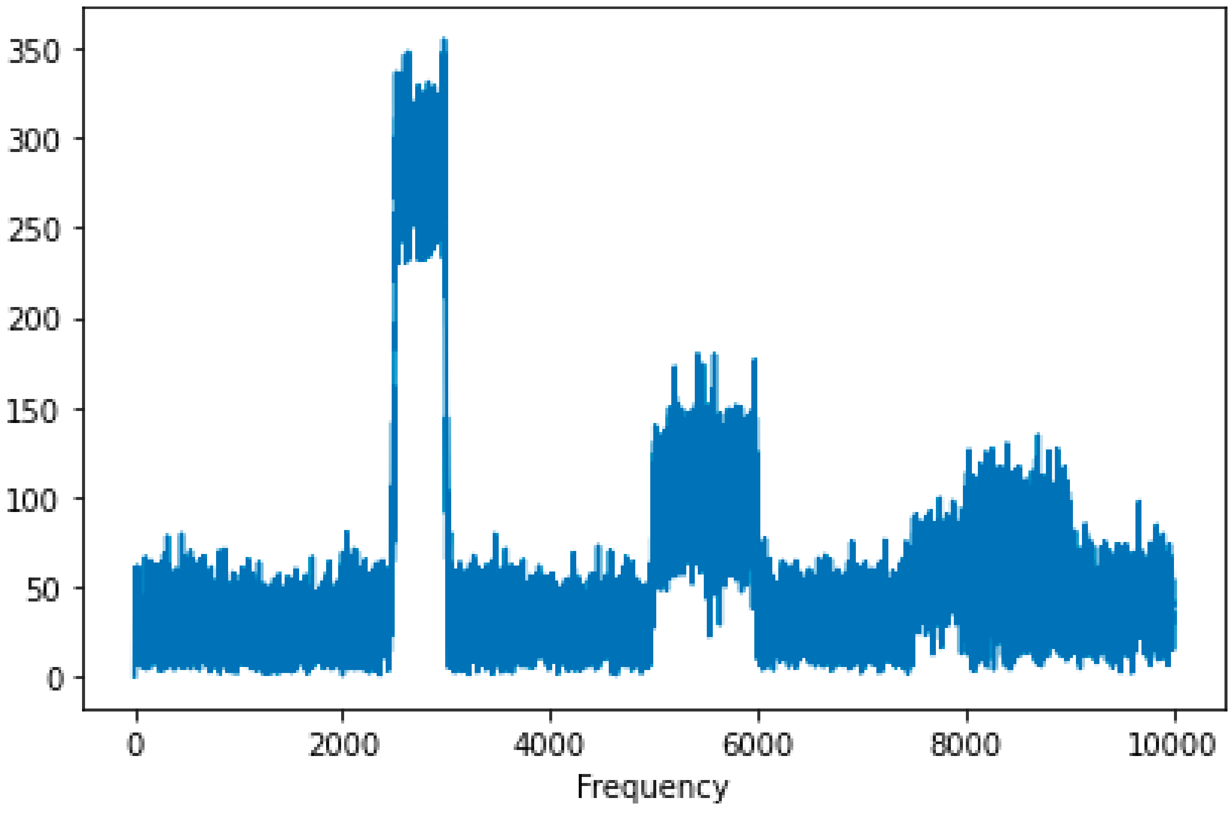

Figure 4: Aliasing effect

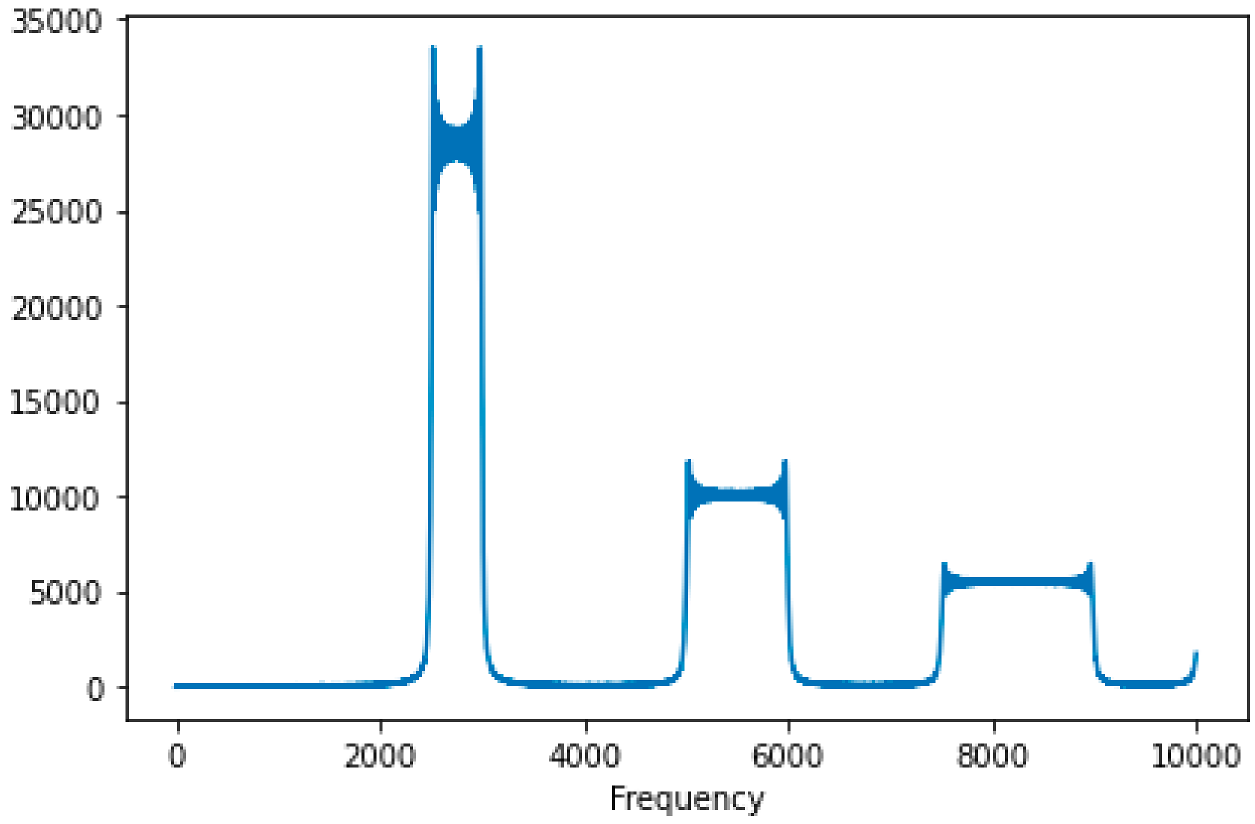

Figure 5: Aliasing effect comparison

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

4 Part 4: Glissando

In this part we need to explore glissando - transition between 2 frequencies. As it was in the Lab1, Halzion track was used.

I’ve selected a vowel ”a” transition segment on 73 second of the record.D

Code of this part - Listing 6

```
wave = read_wave('sound/halzion.wav')
segment = wave.segment(start=73.5, duration=1.5)
#segment.plot()
spec = segment.make_spectrum()
spec.plot(high = 5000)
segment.make_spectrogram(1024).plot(high=5000)
decorate(xlabel='Time (s)', ylabel='Frequency (Hz)')

```

Listing 6: HS operations

Spectrum of the signal looks fine (Figure **??**), as well as spectrogram does (Figure 7. We can clearly see transition of the pitch.

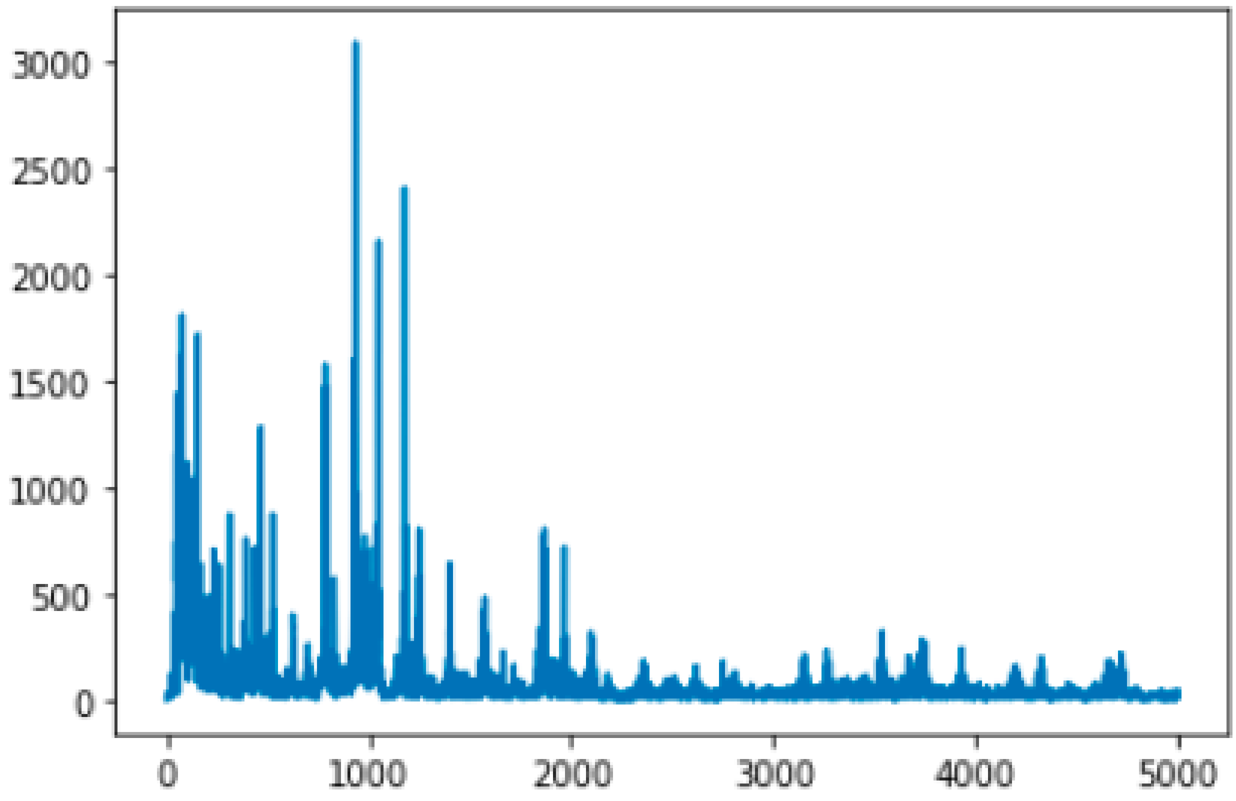

Figure 6: Spectrum of a segment

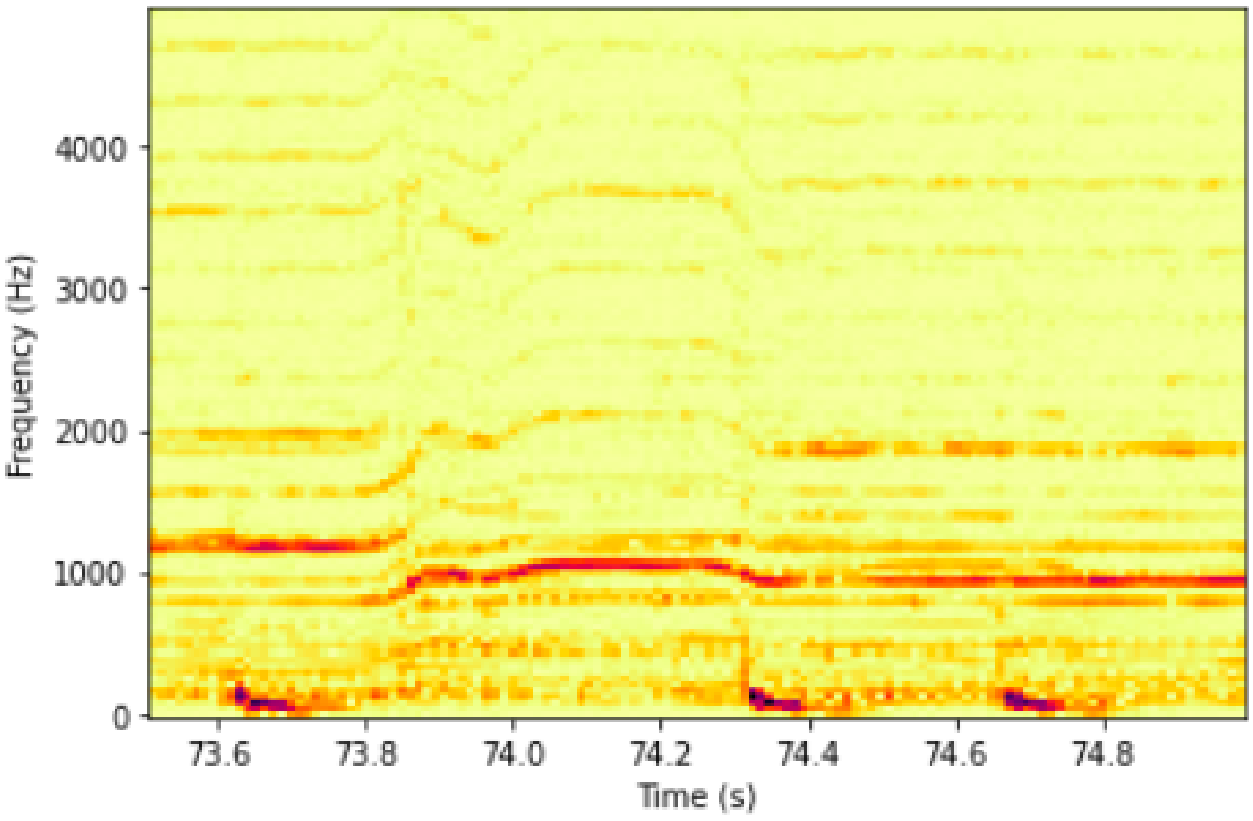

Figure 7: Spectrogram of the segment

5 Part 5: TromboneGliss

In this part we need to create a class, that provides trombone behavior. It should provide evaluate function, that behaves like trombone would behave by sliding its tube.

Code of the class - Listing 7

```python
class TromboneGliss(Chirp):
    def evaluate(self, ts):
        l1, l2 = 1.0 / self.start, 1.0 / self.end
        lengths = np.linspace(l1, l2, len(ts) - 1)
        freqs = 1 / lengths

        dts = np.diff(ts)
        dphis = PI2 * freqs * dts
        phases = np.cumsum(dphis)
        ys = self.amp * np.cos(phases)
        return ys

```

Listing 7: Definition of the class

Code of plotting: Listing 8.

```python
wave1 = MyTromboneGliss(262, 349).make_wave(duration=1)
wave2 = MyTromboneGliss(349, 262).make_wave(duration=1)
wave1.apodize(); wave2.apodize()
wave = wave1 | wave2
wave.make_spectrogram(1024).plot(high=600)
decorate(xlabel='Time (s)', ylabel='Frequency (Hz)')
wave.make_audio()

```

Listing 8: Class usage

As we can see, spectrogram is correct and it looks like exponential one.

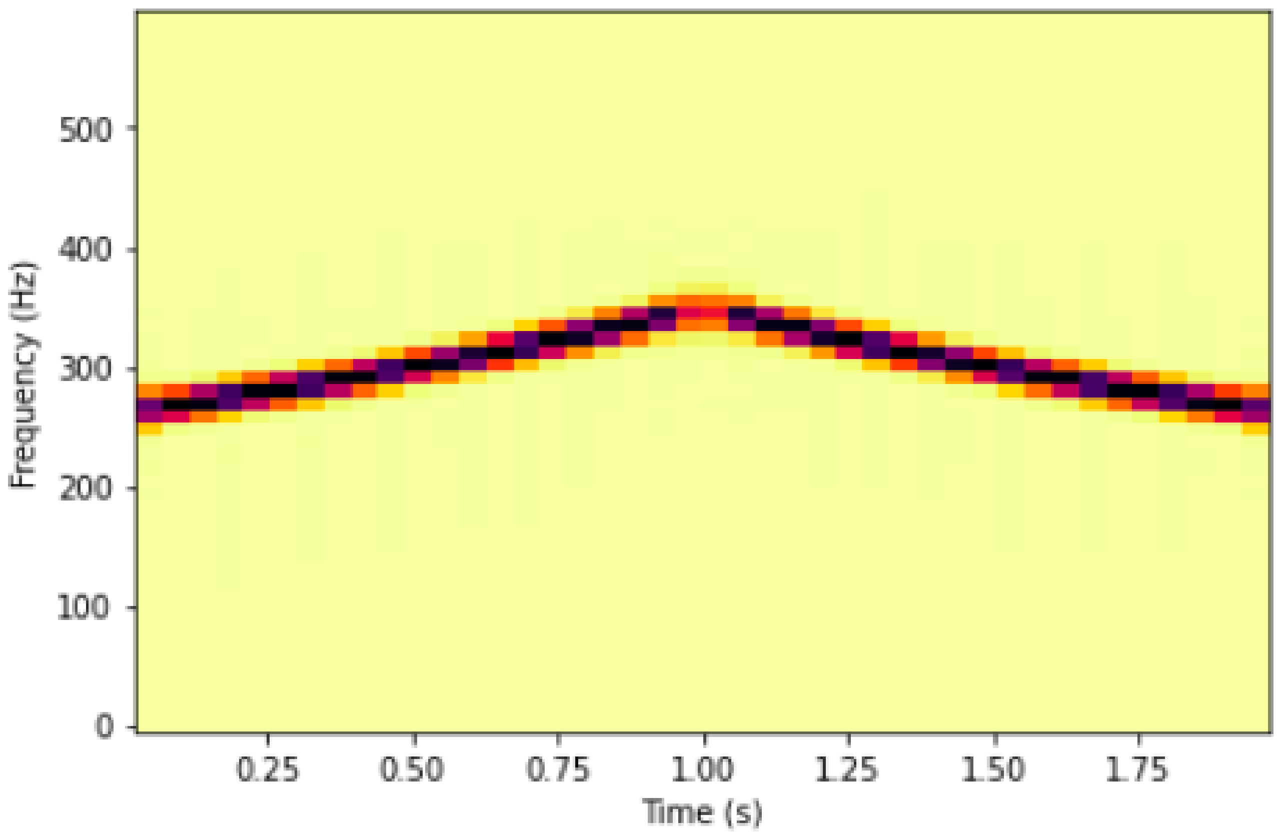

Figure 8: Trombone’s spectrogram

6 Part 6: Vowels analyze

In this part we need to make an analysis of vowels. To do it, I used Japanese vowels A I U E O - all their vowels.

Code of spectrogram plotting can be observed on the Listing 9.

```
wave = read_wave('sound/aiueo.wav')
segment = wave.segment(start=0, duration=6)
segment.make_spectrogram(3000).plot(high=1500)
decorate(xlabel='Time (s)', ylabel='Frequency (Hz)')

```

Listing 9: Signal and spectrum creation

As we can see on the Figure 9, Japanese vowels are differs a lot an we can easily say what the sound it is by looking on the spectrogram.

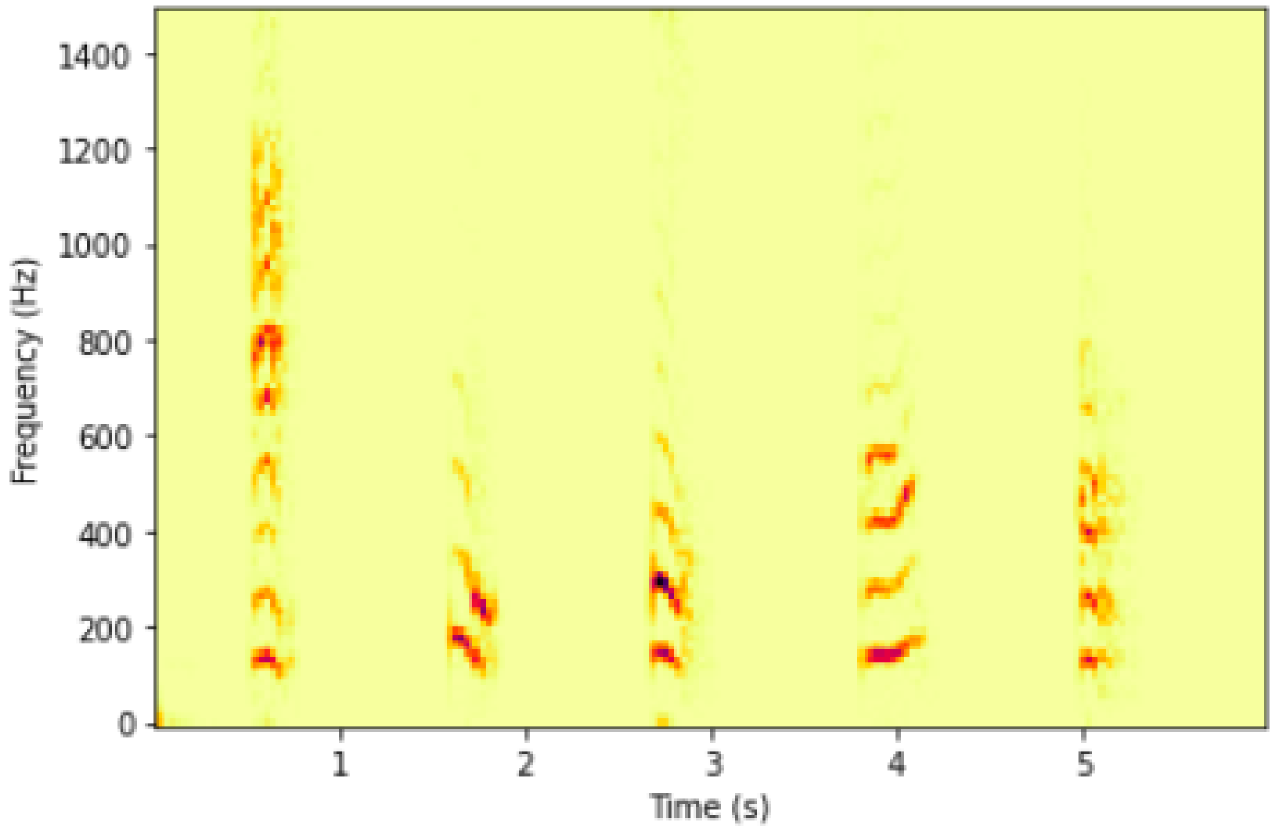

Figure 9: Spectrum of the signal

7 Conclusion

More advanced skills and knowledge of signals, waves spectrum and spectrogram was acquired. Spectrum is a good way to visualize a periodic signal, but we cannot use it for the non-periodic signals. But we can split the non-periodic signal into the set of small segments, which we can considered to be a periodic signals (just like it is done with integrals). That allows us to create spectrogram - a set of spectrums at any of the sound. However, we cannot achieve a good time resolution and a good frequency resolution on the same time.