Санкт-Петербургский государственный политехнический
университет Петра Великого

**Высшая школа интеллектуальных систем и
суперкомпьютерных технологий**

Лабораторная работа

Шум

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: Env sounds

In this part we need to explore how different real-life environment sounds looks like. To do it rain and sea sound was used.

Firstly, let's check how overall spectrogram of sea looks like.

```
from thinkdsp import *
wave = read_wave('data/sea.wav')
spectrum = wave.make_spectrum()
spectrum.plot()

```

Listing 1: File load and spectrum plotting

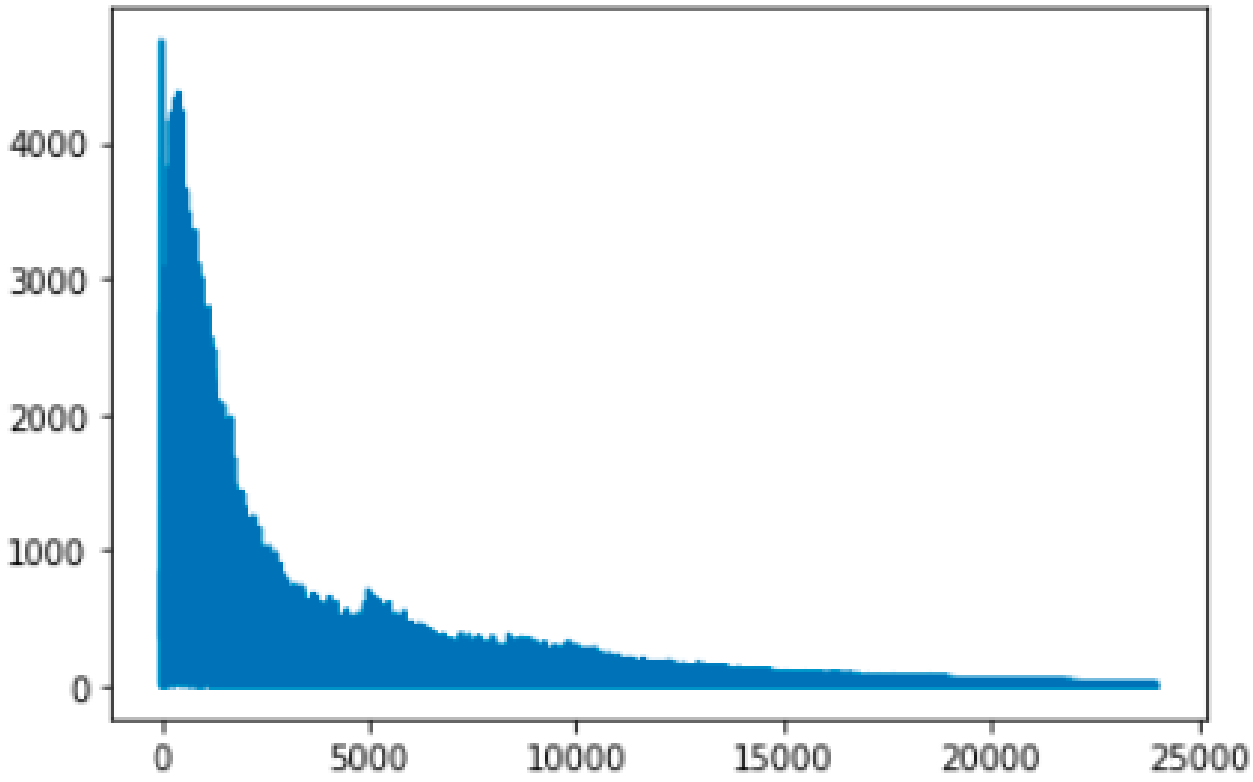

Figure 1: Sea overall spectrum

To make it easier for researching let's plot powers in log-scaled axis. By dominance of lower pitches we can say, that it can be a pink or a red noise.

```
spectrum.plot_power()
loglog = dict(xscale='log', yscale='log')
decorate(xlabel='Frequency (Hz)', **loglog)

```

Listing 2: Log spectrum

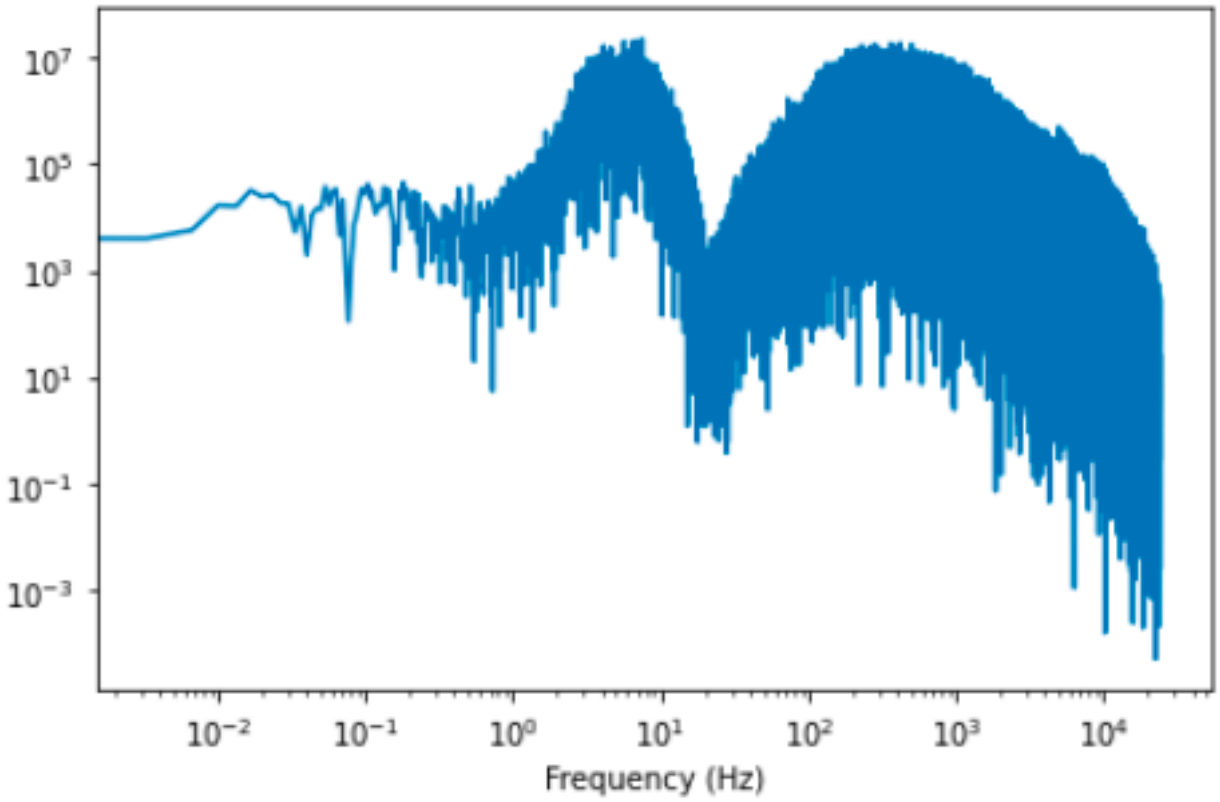

Figure 2: Sea overall log spectrum

Secondly, let's check how signal changes over the time. To do it I've selected 2 segments and plotted an spectrum for them. We can see, that both segments look almost the same.

```
    seg1 = wave.segment(start=10, duration=2)
    seg2 = wave.segment(start=20, duration=2)
    spec1 = seg1.make_spectrum()
    spec2 = seg2.make_spectrum()
    spec2.plot(color='red')
    spec1.plot(color='blue')

```

Listing 3: Segments spectrum

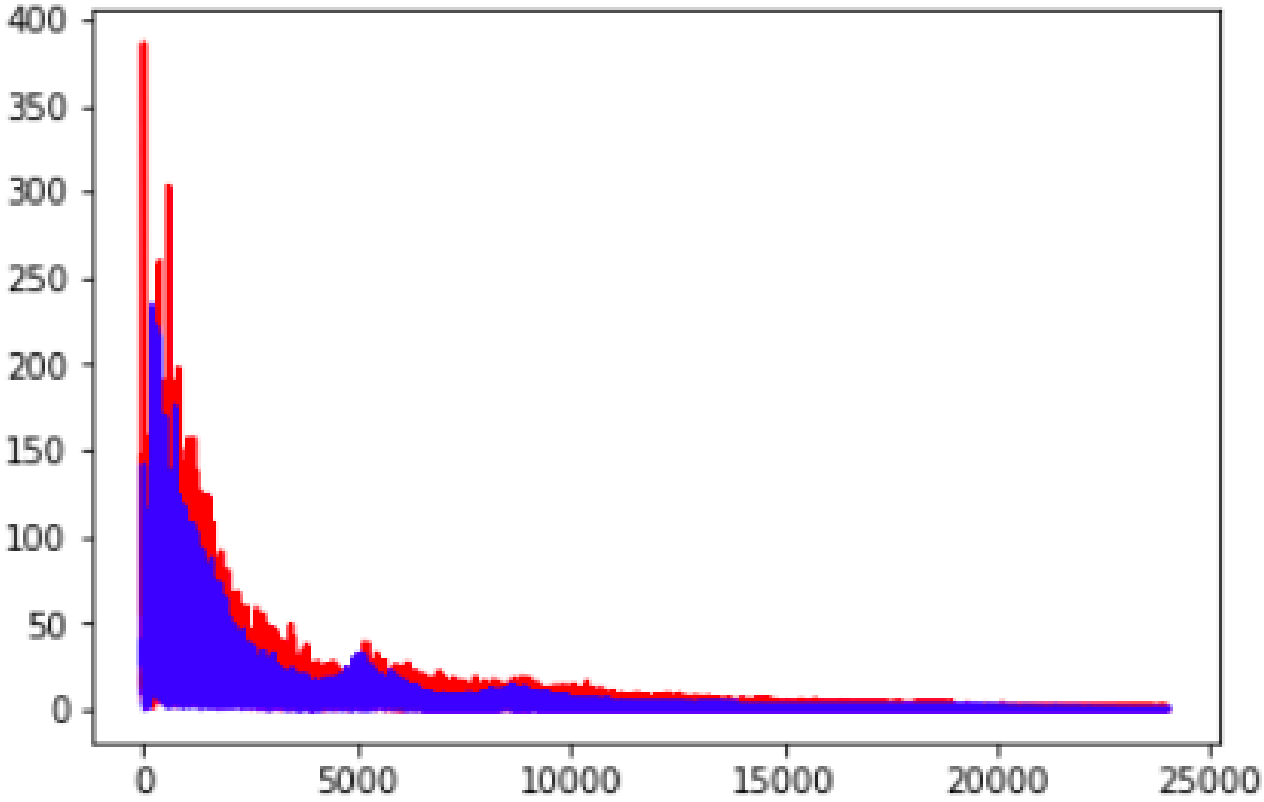

Figure 3: Sea segments log spectrum

```
spec2.plot_power(color='red')
spec1.plot_power(color='blue')
loglog = dict(xscale='log', yscale='log')
decorate(xlabel='Frequency (Hz)', **loglog)

```

Listing 4: Log spectrum of segments

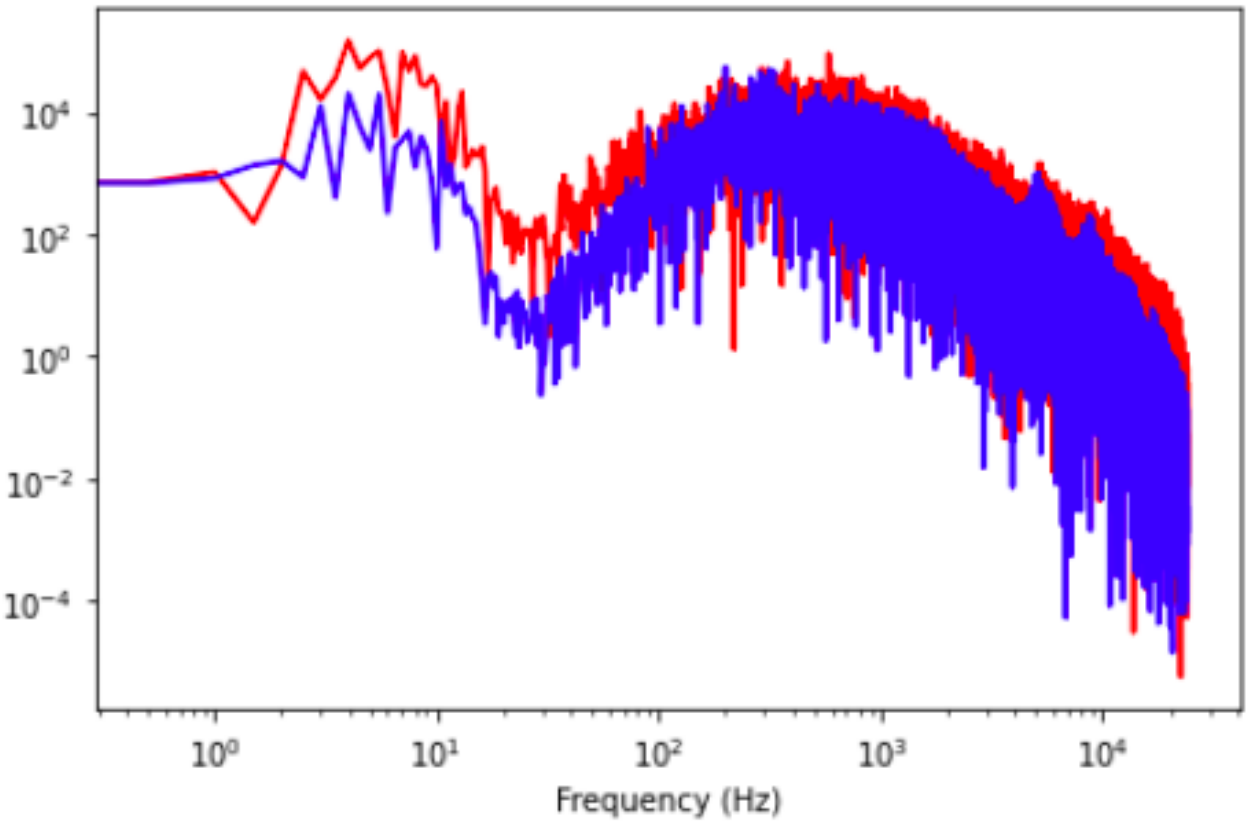

Figure 4: Sea segments spectrum

To make sure, that signal is consistent over the time, let's plot a spectrogram of a signal. We can see, that signal stay almost the same with periodic spikes - crushing waves sound.

```
wave.make_spectrogram(2048).plot(high=5000)
decorate(xlabel='Time(s)', ylabel='Frequency (Hz)')

```

Listing 5: Overall spectrogram

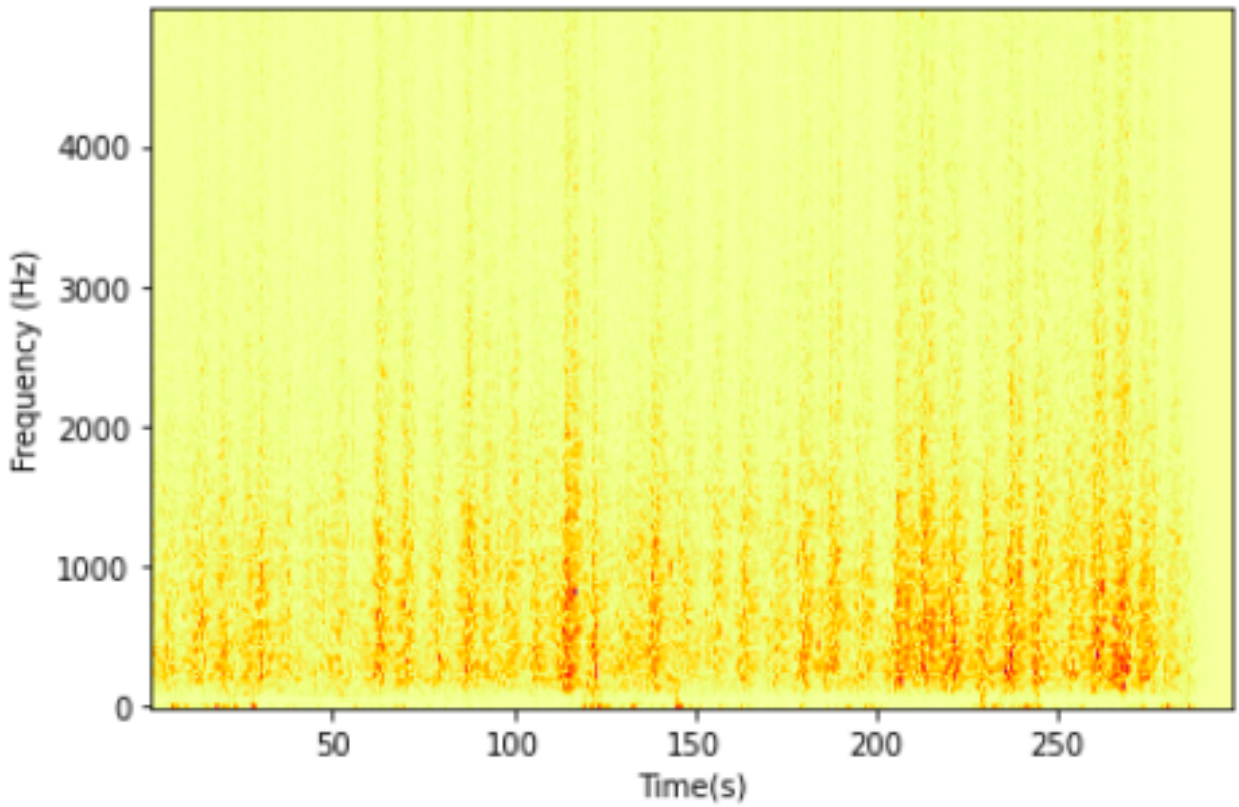

Figure 5: Sea overall spectrogram

Let’s do the same to the other sound - sound of the rain. By looking on its overall spectrum we can say, that it is a pink or a red noise.

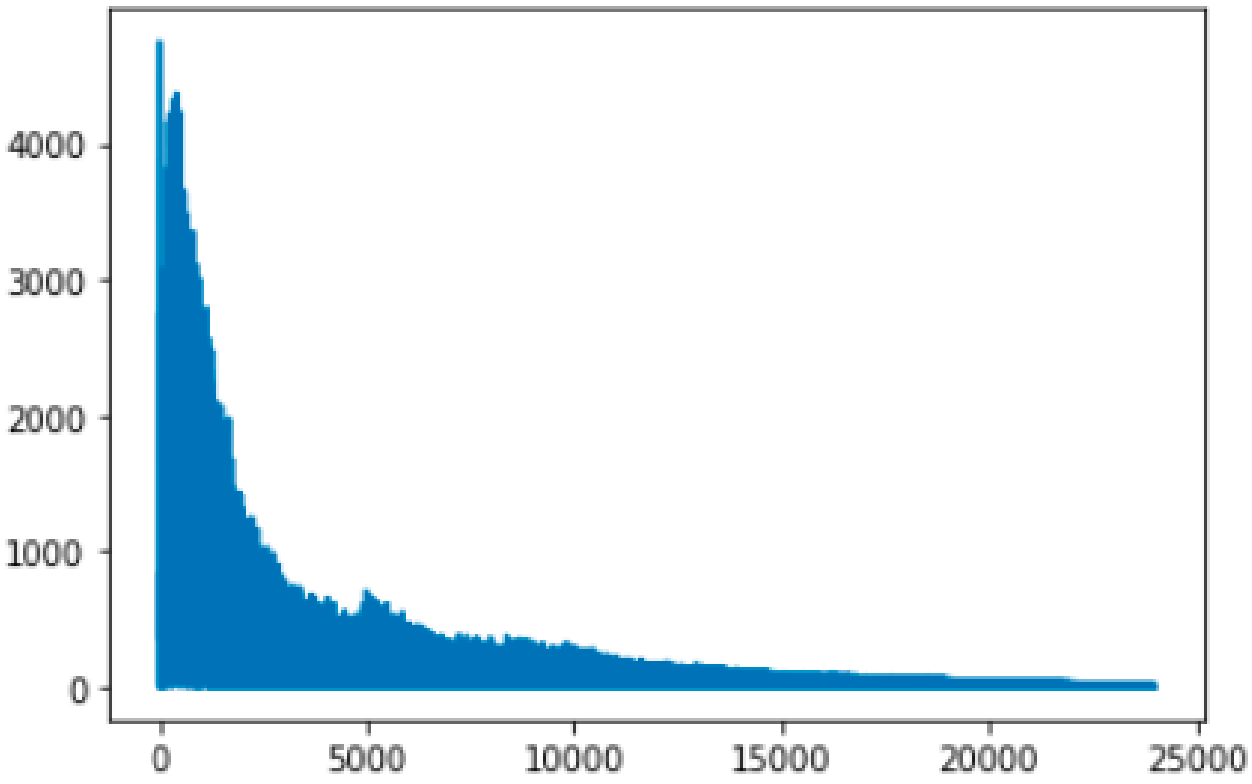

Figure 6: Rain overall spectrum

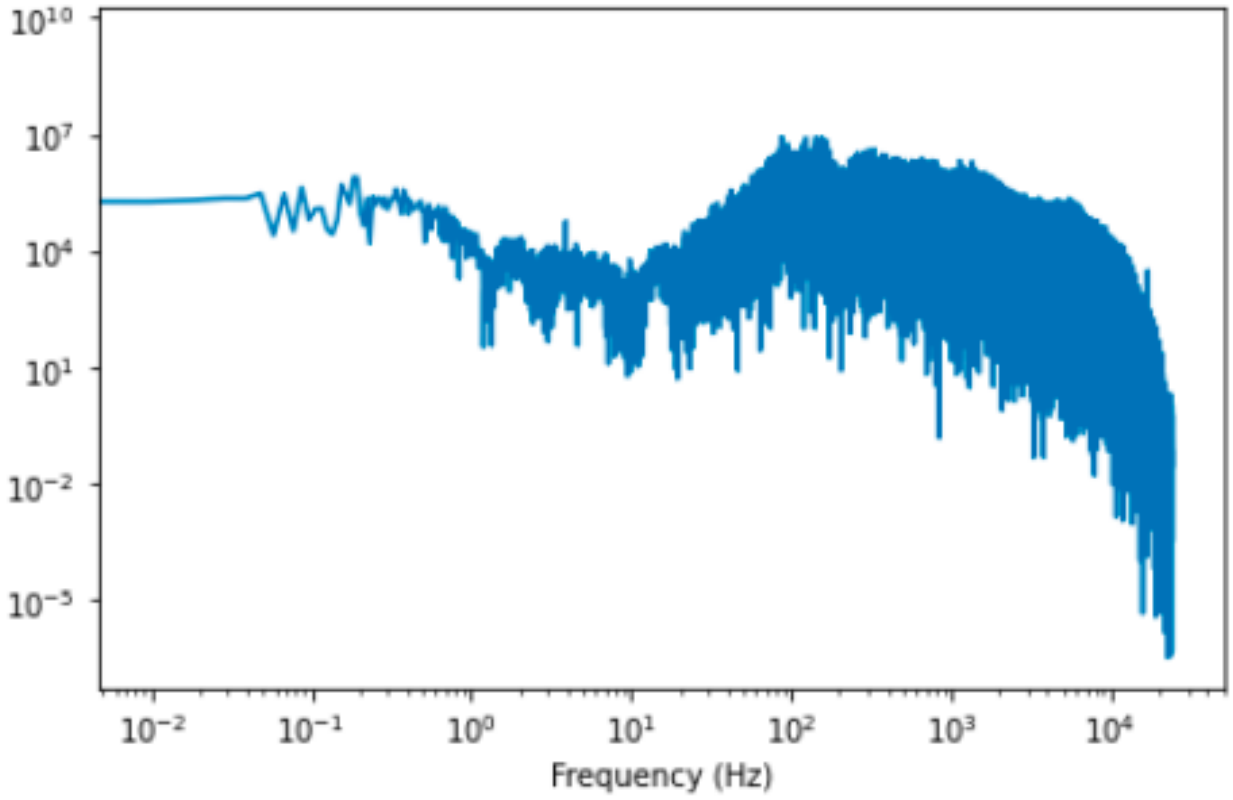

Figure 7: Rain overall log spectrum

By checking its segments we can say, that signal stays consistent over the time.

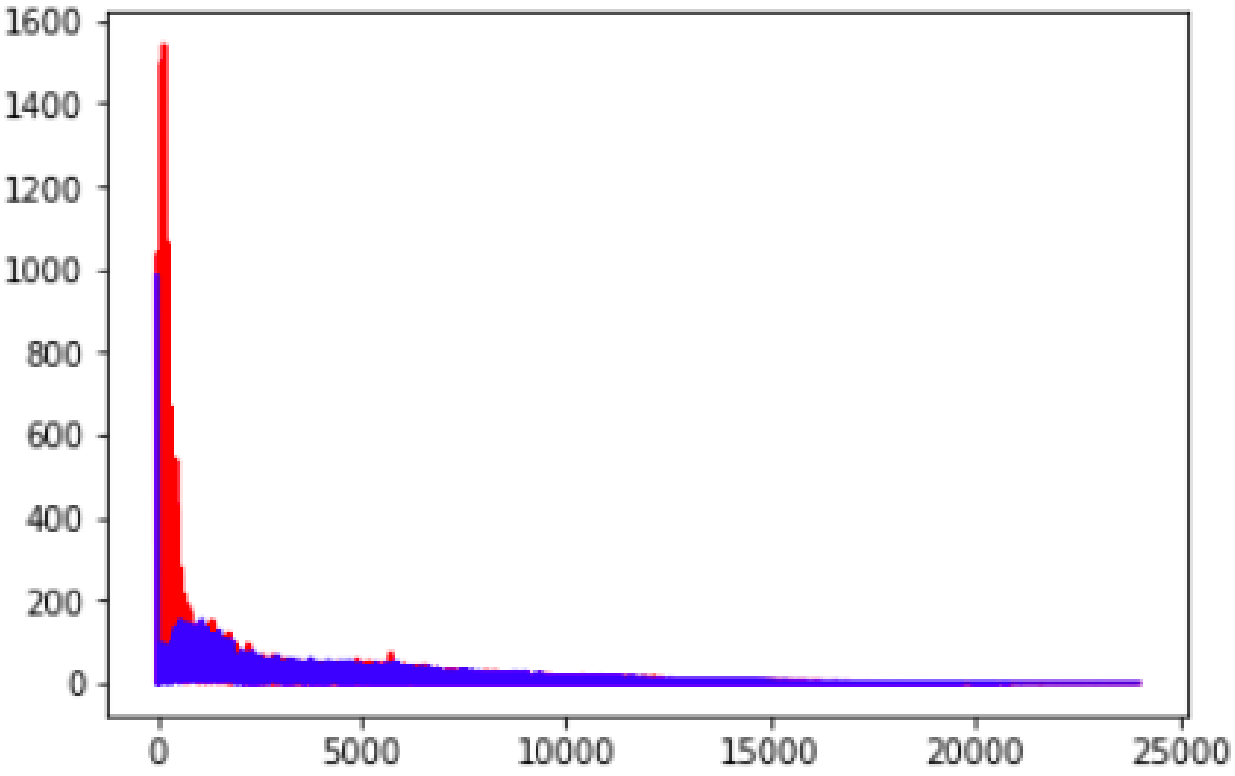

Figure 8: Rain segments log spectrum

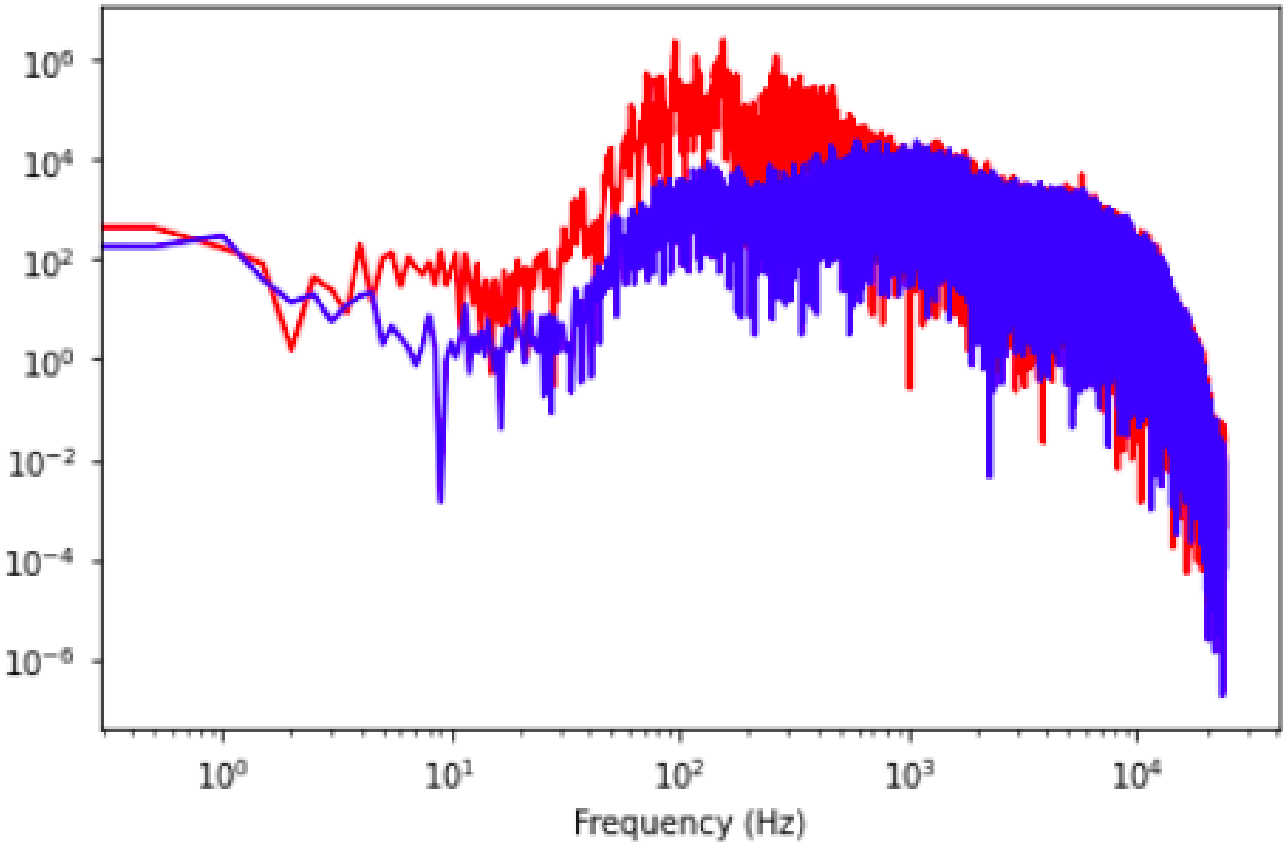

Figure 9: Rain segments spectrum

However, by looking on the overall spectrogram we can say, that there was 2 loud events like something fell or hit something other.

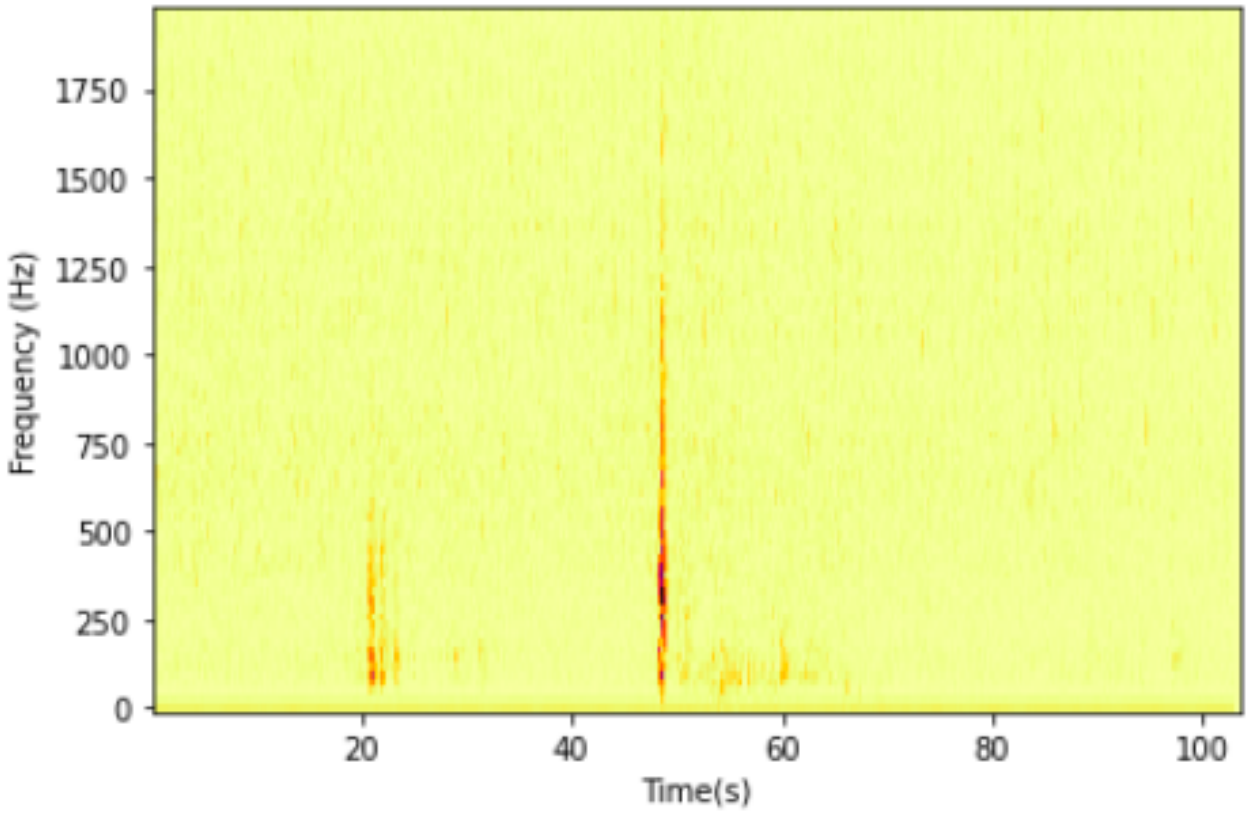

Figure 10: Rain overall spectrogram

A lot of environmental sounds has red/ping nature, and these sounds are not an exception.

2 Part 2: Barrett's method

In this part we need to create Berrett's method to estimate signal's power using its segments.

Let's declare a function:

```
def bartlett_method(wave, seg_length=512):
    spectro = wave.make_spectrogram(seg_length , True)
    spectrums = spectro.spec_map.values()
    pwrs = [spectrum.power for spectrum in spectrums]
    hs = np.sqrt(sum(pwrs) / len(pwrs))
    fs = next(iter(spectrums)).fs
    spectrum = Spectrum(hs, fs, wave.framerate)
    return spectrum

```

Listing 6: Barrett's function definition

Next, let's check how it's work using segments from previous part:

```
p1 = bartlett_method(seg1)
p2 = bartlett_method(seg2)
p1.plot_power(color='blue')
p2.plot_power(color='red')
decorate(xlabel='Frequency (Hz)',ylabel='Power',**loglog)

```

Listing 7: Sawtooth wave plot code

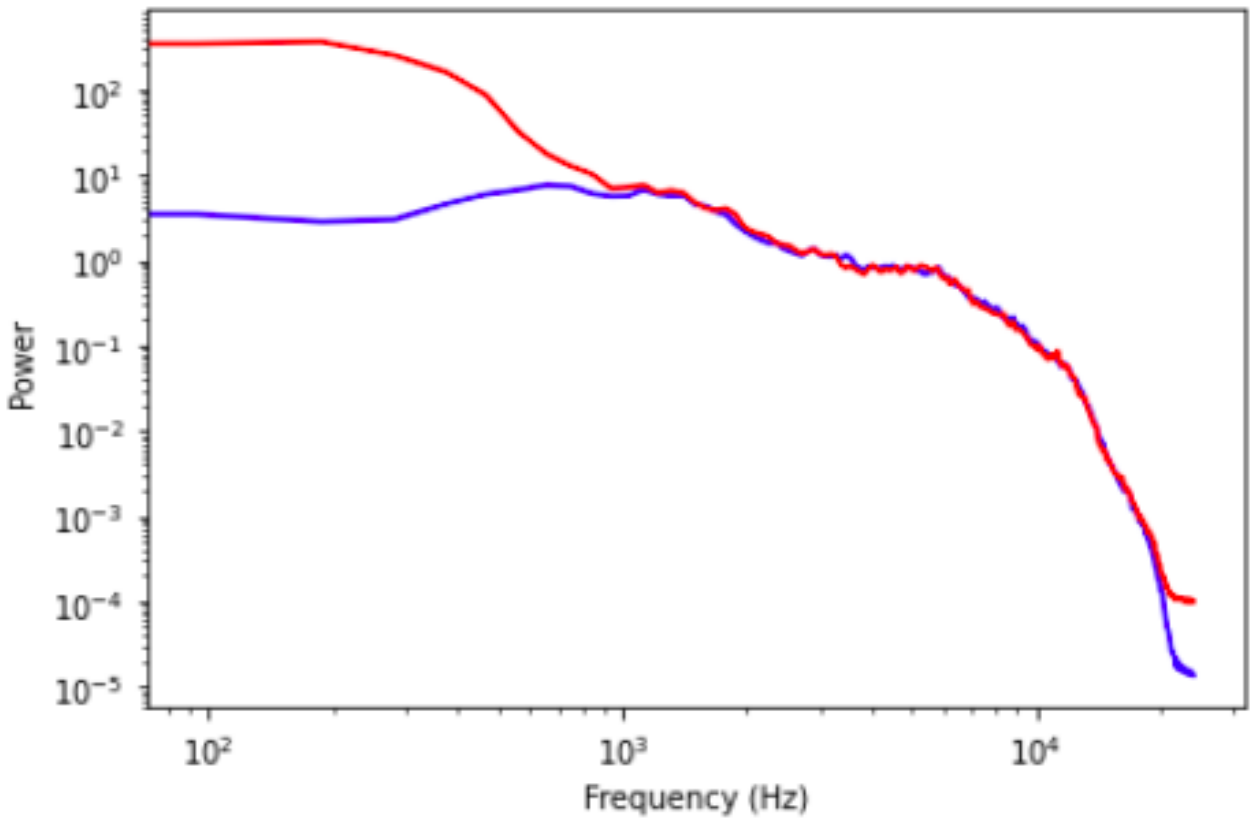

Figure 11: Mean powers by Barrett's method

We can see how close these 2 graphs are. It happens because of noisy nature of the signal - mean energies of the segments represents mean energy of all signal.

3 Part 3: GREATEST CRYPTOCURRENCY OF ALL TIME (aka. bitcoin)

In this part we need to try to use wave analysis tools to the bitcoin chart.

Let's read the bitcoin cost at the end of every day and represent it as a wave.

```
data = pd.read_csv('data/bitcoin.csv')
price = data['Closing Price (USD)']
count = data.index
wave = Wave(price,count,framerate=1)
wave.plot()
decorate(xlabel='Day')

```

Listing 8: Bitcoin to a wave

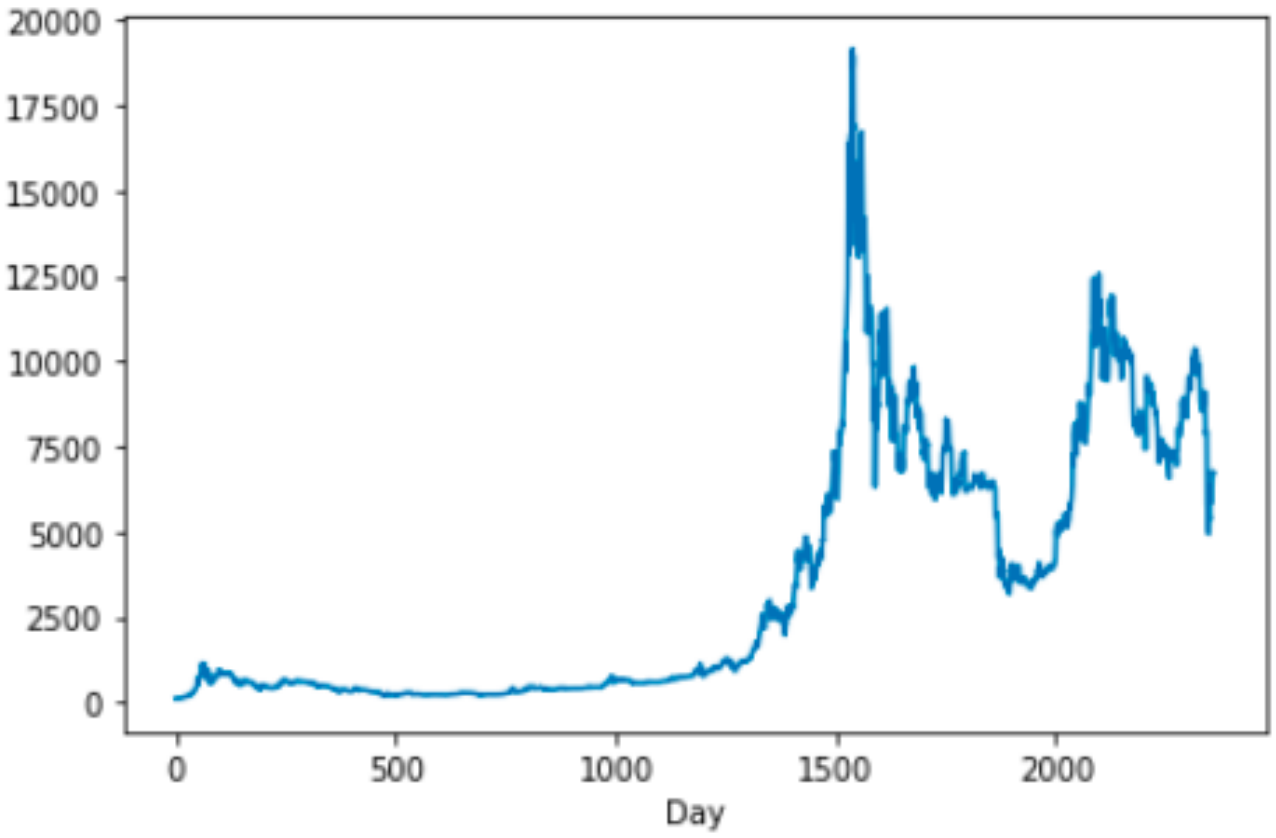

Figure 12: Bitcoin chartt

Let's look, how looks a logarithmic graph of a spectrum's power.

```
spectrum = wave.make_spectrum()
spectrum.plot_power()
decorate(**loglog)
print(spectrum.estimate_slope()[0])

```

Listing 9: Bitcoin to a spectrum's power

```
-1.7332540936758956
```

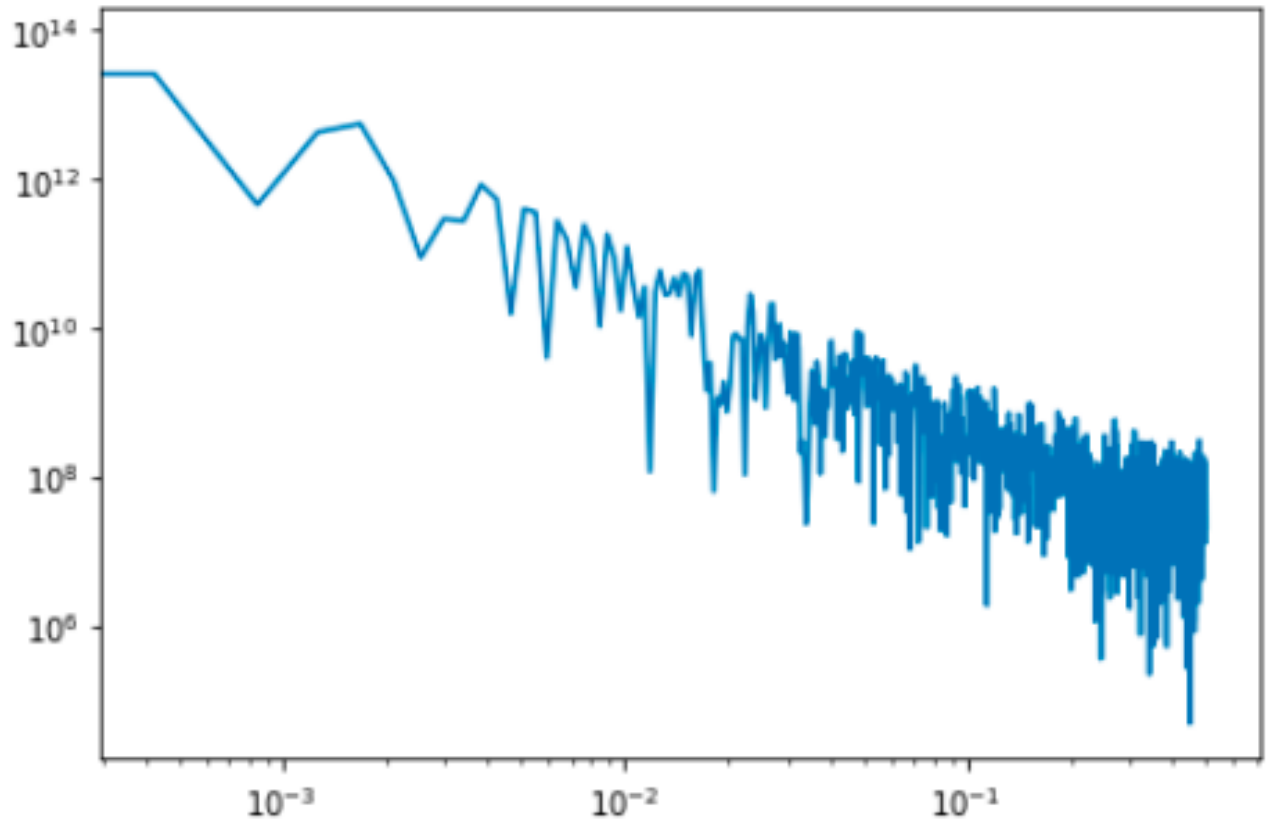

Figure 13: Bitcoin as power

As we can see, bitcoin acts like a pink or a brown noise. By checking its slope rate, that equals ti -1.73 we cannot be sure, what noise it is - red or brown.

4 Part 4: Geiger counter

In this part we need to explore Poisson distribution. To do it I've changed distribution function to `np.random.poisson`.

Code of new noise is next. We can see on the plot, that it has 51 picks, when expected value is 50.

```
class UncorrelatedPoissonNoise(Noise):
    def evaluate(self, ts):
        ys = np.random.poisson(self.amp, len(ts))
        return ys

signal = UncorrelatedPoissonNoise(amp=0.001)
wave = signal.make_wave(duration=5, framerate=10000)
wave.plot()
print(sum(wave.ys))

```

Listing 10: Usage of Poisson

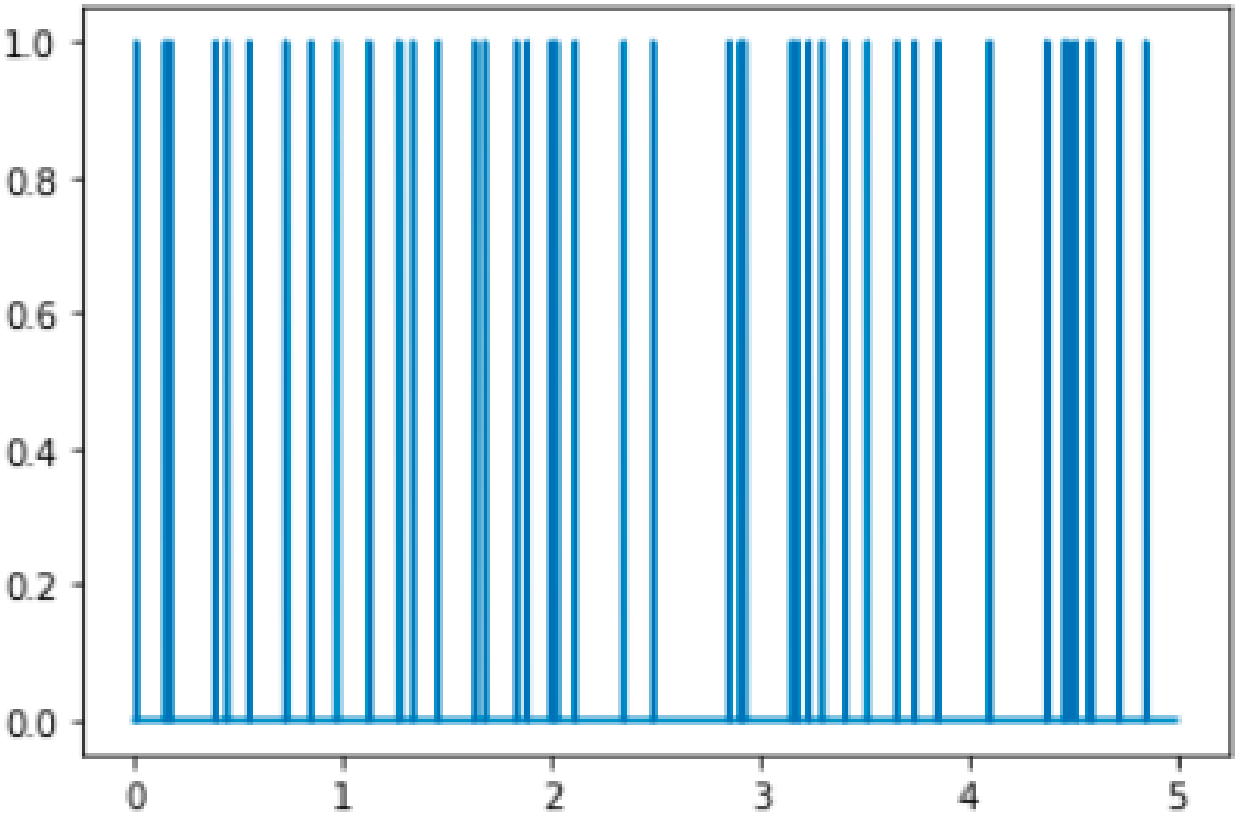

Figure 14: Waveform pf Poisson

Let's create logarithmic power plot. We can see, that it is clear white noise.

```
spectrum = wave.make_spectrum()
spectrum.plot_power()
```

```
    decorate(xlabel='Frequency (Hz)',**loglog)
    print(spectrum.estimate_slope()[0])

```

Listing 11: Usage of Poisson

```
0.008841564134846628
```

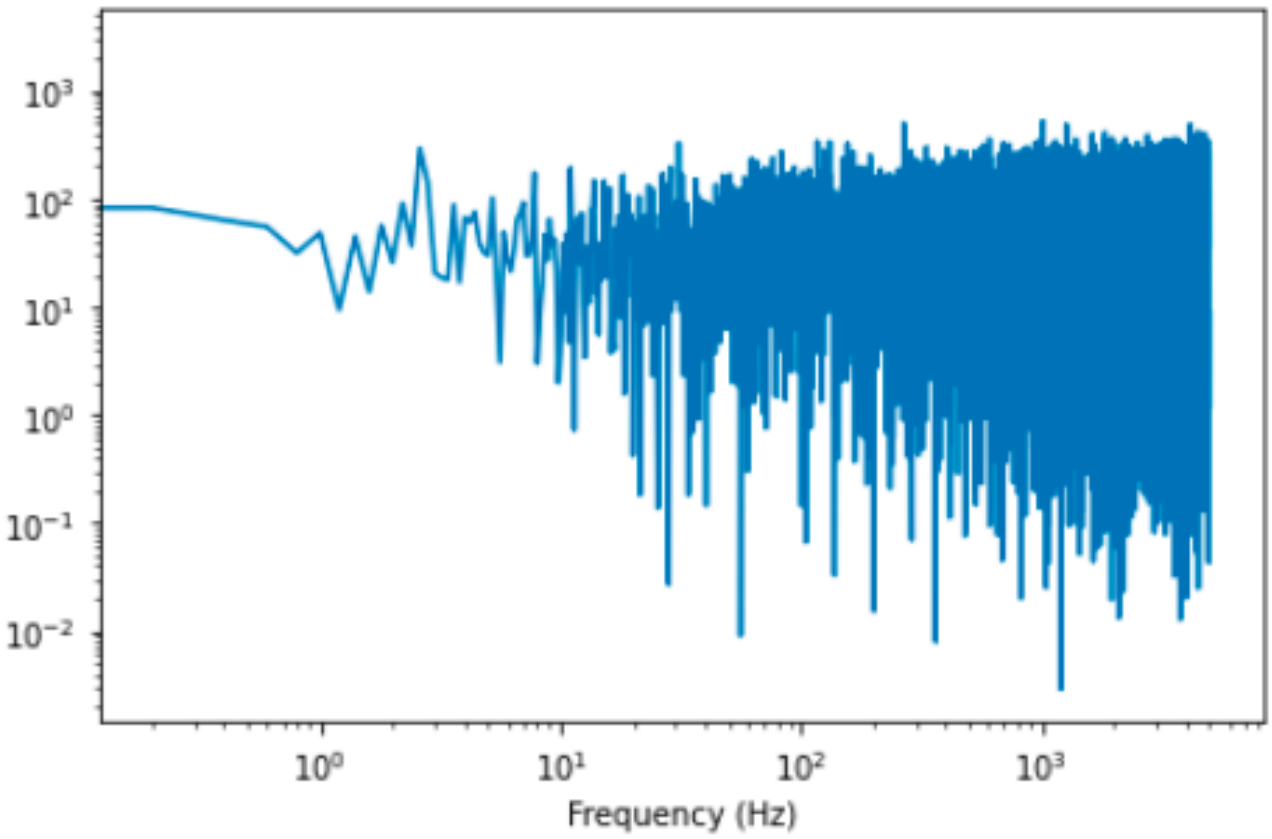

Figure 15: Geiger power plot

5 Part 5: Voss-McCartney algorithm

Algorithm, that creates a pink noise by cutting of higher frequencies is slow and ineffective, that's why new algorithms was created. One of them is Voss-McCartney algorithm.

Code of the class - Listing 12

```
def voss(nrows , ncols=16):
    array = np.empty((nrows , ncols))
    array.fill(np.nan)
    array[0, :] = np.random.random(ncols)
    array[:, 0] = np.random.random(nrows)
    n = nrows
    cols = np.random.geometric(0.5, n)
    cols[cols >= ncols] = 0
    rows = np.random.randint(nrows , size=n)
    array[rows, cols] = np.random.random(n)
    df = pd.DataFrame(array)
    df.fillna(method='ffill', axis=0, inplace=True)
    total = df.sum(axis=1)
    return total.values

```

Listing 12: Voss-McCartney algorithm

Let's create a wave and its plot using this pink-noise generator

```
wave = Wave(voss(10000))
wave.unbias()
wave.normalize()
wave.plot()

```

Listing 13: Wave and plot creation

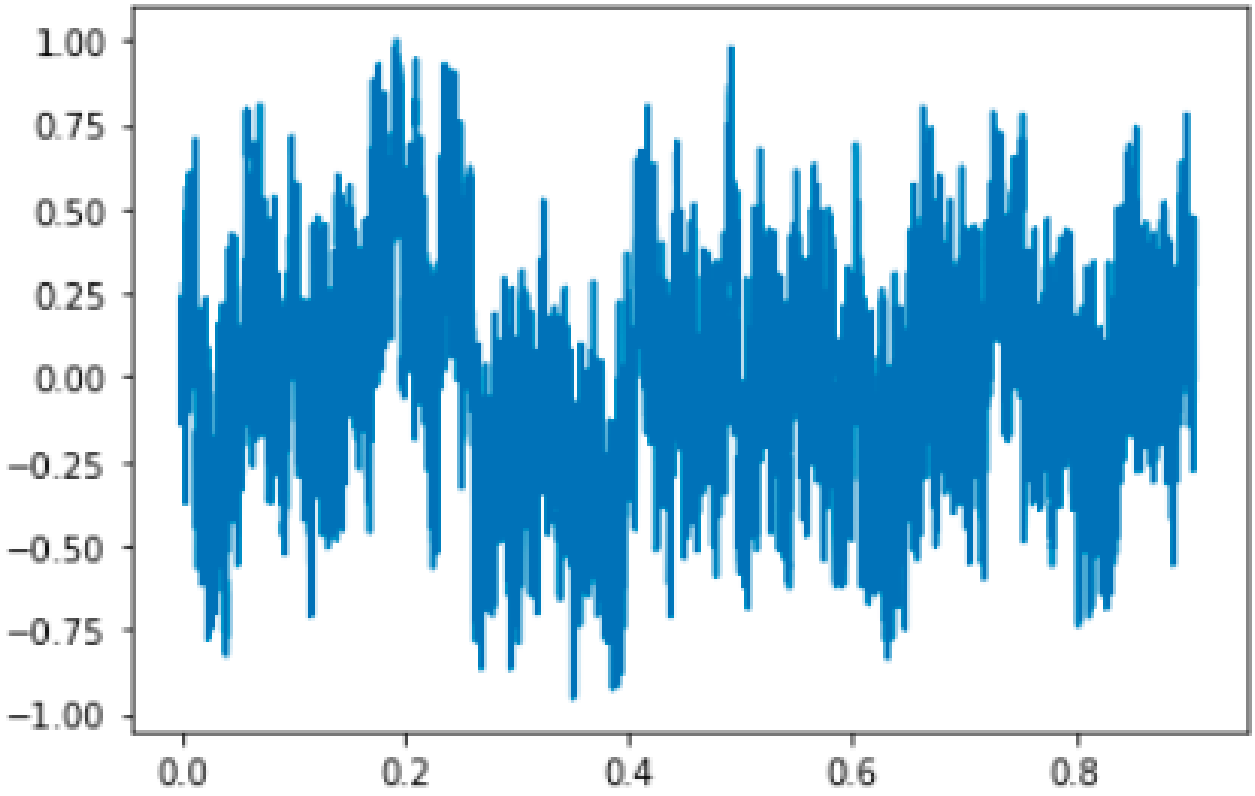

Figure 16: Resulting wave

To be sure, that resulting noise is pink, lets build spectrum's power plot and calculate its slope. It is equals to -1.017 - the noise is pink.

```
    spectrum = wave.make_spectrum()
    # don't break the math's laws
    spectrum.hs[0] = 0
    spectrum.plot_power()
    decorate(xlabel='Frequency (Hz)',**loglog)
    print(spectrum.estimate_slope()[0])

```

Listing 14: Wave and plot creation

-1.017691837950757

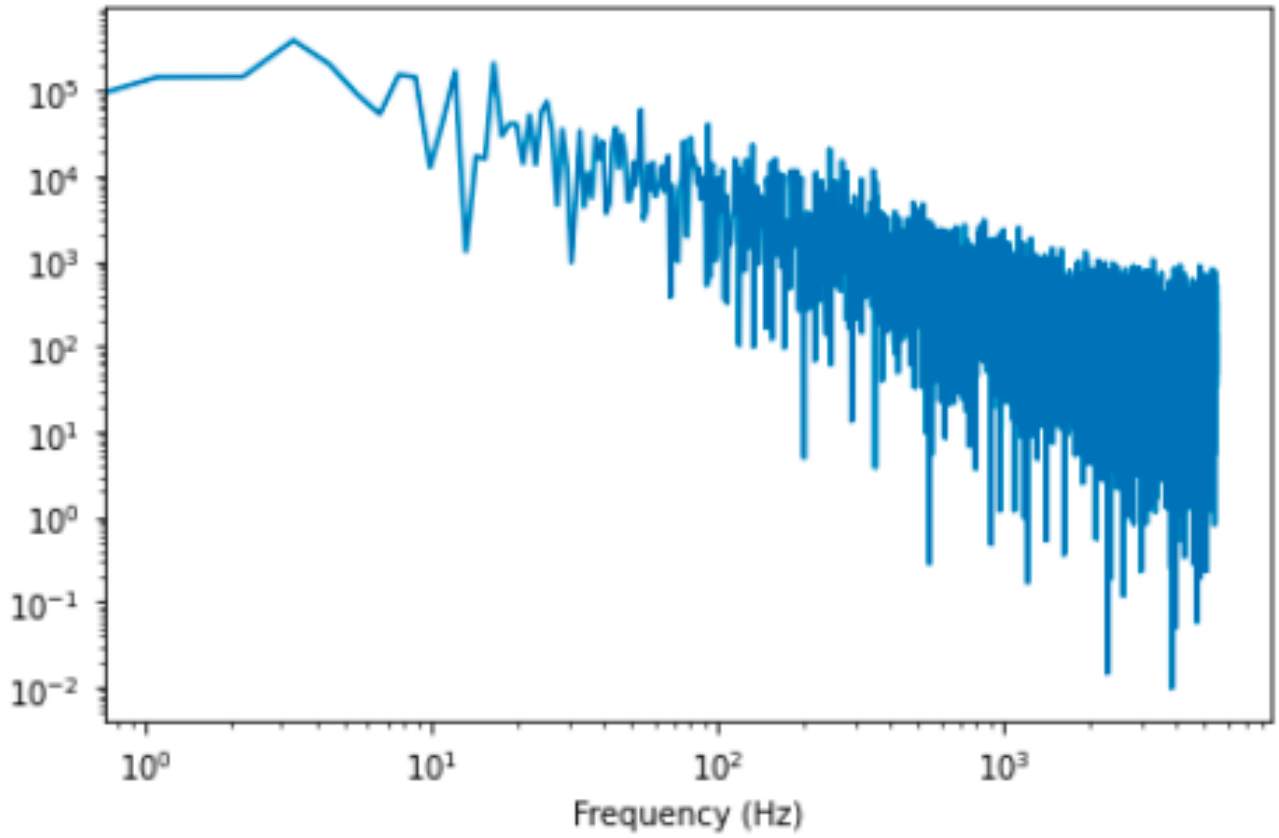

Figure 17: Spectrum's power plot

6 Conclusion

More advanced skills and knowledge of signals, waves spectrum and spectrogram was acquired. Noise - is a random unwanted modification, that affects the signal. Also noise - is signal with a lot of chaotic components, that doesn't have a harmonic structure. There are different types of noise. We've learned some of them - white, pink, red, Gaussian and Poisson. We've learned, how can we generate a noise and analyze it.