ThinkDSP. Лабораторная 4. Шум.

Шерепа Никита

7 мая 2021 г.

Содержание

Список иллюстраций

Листинги

1 Упражнение 4.1

1. **Задание**

   Скачайте несколько файлов с веб-страницы *http://asoftmurmur.com/about/]* и вычислисте спектры каждого сигнала. Похож ли спетр мощности на белый, розовый или броуновский шум? Как спектр меняется во времени?

2. **Ход работы**

   Я скачал несколько звуков. Начнем со звука морских волн. Воспроизведем его, выделим сегмент и построим его спектр.

```
import numpy as np
import matplotlib.pyplot as plt

from thinkdsp import decorate
from thinkdsp import read_wave

wave = read_wave('res/audio/Sea_waves.wav')
wave.make_audio()
segment = wave.segment(start=20, duration=3.0)
segment.make_audio()

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

Листинг 1: Работа со звуком морских волн

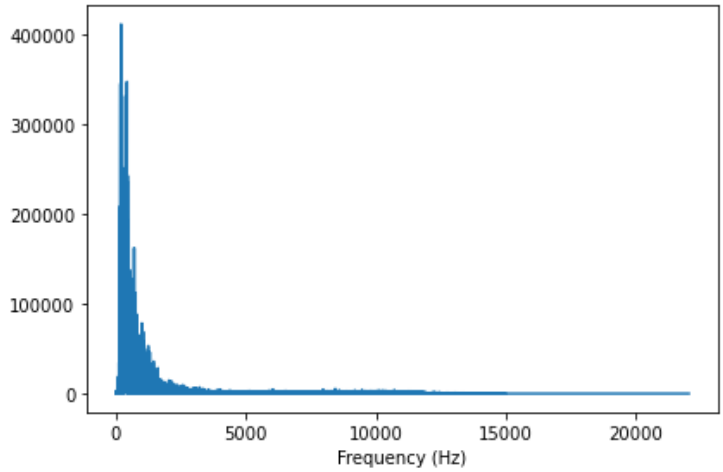

Рис. 1: Спектр сегмента

Видим, что амлитуда убывает с частотой. Поэтому это может быть или броуновский (красный) или розовый шум. Построим спектр мощности на логарифмической шкале.

```
spectrum.plot_power()

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

Листинг 2: Спектр мощности

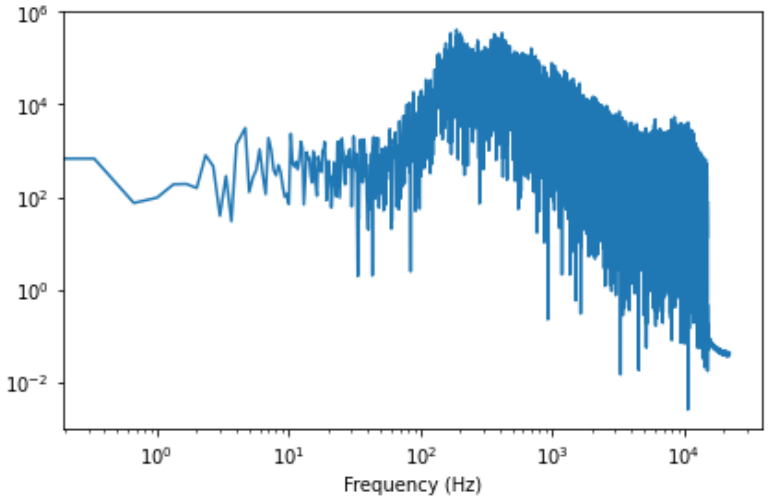

Рис. 2: Спектр мощности

Из-за увеличения и уменьшения амплитуды это все выглядит как стандартный и естественный источник шума.

Выберем другой звук - сверчки. Воспроизведем его, выделим сегмент и построим его спектр.

```
wave = read_wave('res/audio/crickets_texas.wav')
wave.make_audio()
```

Листинг 3: Работа со звуком сверчков

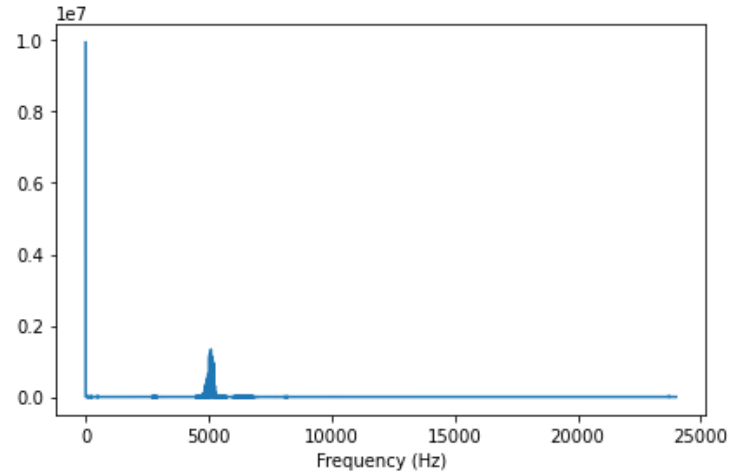

Рис. 3: Спектр сегмента

Немного приблизим

```
spectrum = wave.segment(start = 0, duration =
    5).make_spectrum()
spectrum.high_pass(100)
spectrum.plot()
```

Листинг 4: Приближаем спектр

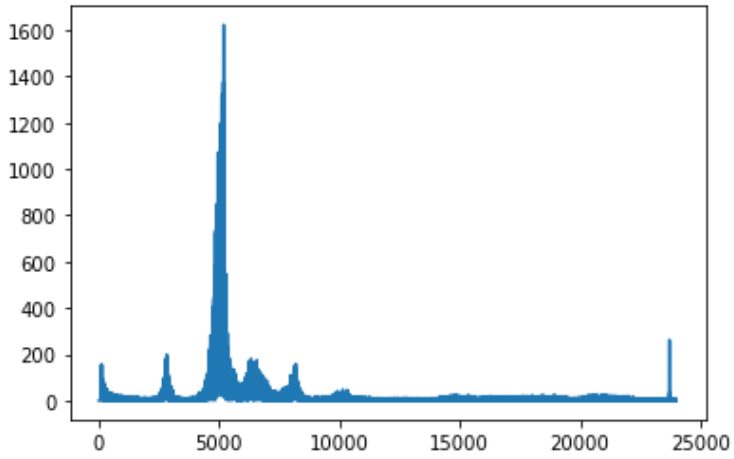

Рис. 4: Приближенный спектр сегмента

Видим, что по характеру изменения частот это не похрже ни на белый шум, ни на розовый

Построим спектр мощности на логарифмической шкале.

```
spectrum.plot_power()

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

Листинг 5: Спектр мощности

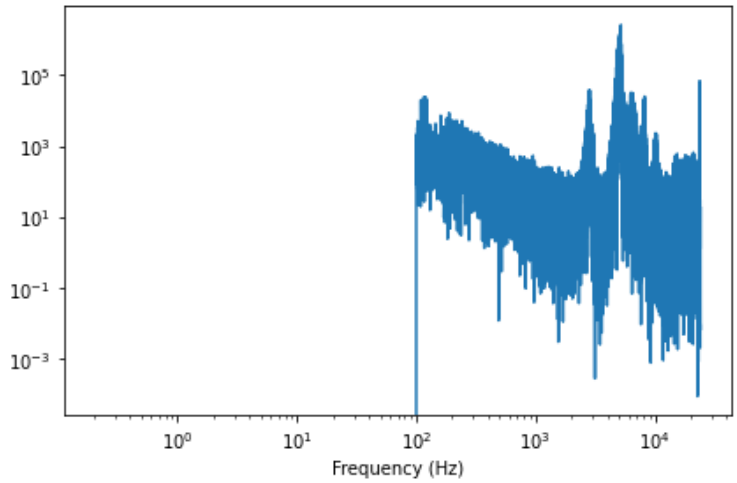

Рис. 5: Спектр мощности

Теперь построим спектрограмму

```
segment.make_spectrogram(512).plot(high=5000)
decorate(xlabel='Time(s)', ylabel='Frequency (Hz)')
```

Листинг 6: Построение спектрограммы звука сверчков

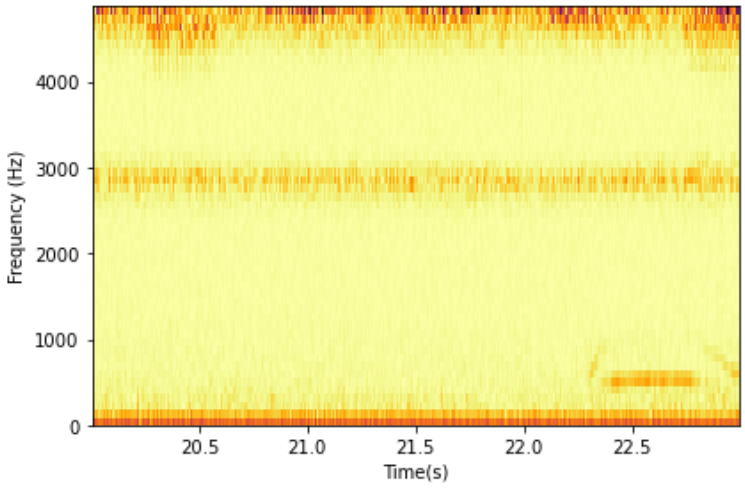

Рис. 6: Спектрограмма звука сверчков

Исходя из спектра мощности и спектрограммы, звук сверчков можно назвать шумом, похожим на белый.

2 Упражнение 4.2

1. **Задание**

   Реализуйте метод Бартлетта и используйте его для оценки спектра мощности шумового сигнала.

2. **Ход работы**

   Реализуем метод Бартлетта

```
from thinkdsp import Spectrum

def bartlett_method(wave, seg_length=512,
    win_flag=True):
spectro = wave.make_spectrogram(seg_length, win_flag)
spectrums = spectro.spec_map.values()

psds = [spectrum.power for spectrum in spectrums]

hs = np.sqrt(sum(psds) / len(psds))
fs = next(iter(spectrums)).fs

spectrum = Spectrum(hs, fs, wave.framerate)
return spectrum
```

Листинг 7: Метод Бартлетта

   Теперь возьмем сегмент звука морской волны и применим к нему метод Бартлетта

```
wave = read_wave('res/audio/Sea_waves.wav')

segment = wave.segment(start=10, duration=3.0)
segment.make_audio()

psd = bartlett_method(segment)

psd.plot_power()

decorate(xlabel='Frequency (Hz)',
ylabel='Power',
**loglog)
```

Листинг 8: Применение метода Бартлетта

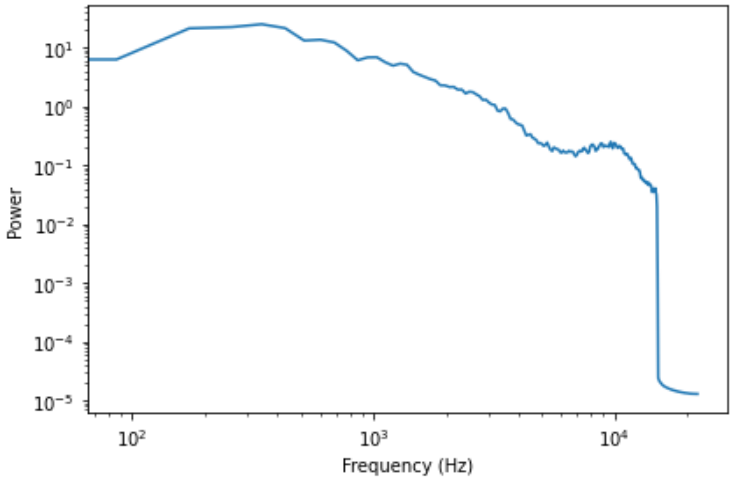

Рис. 7: Спектр сегмента

Видим, что сигнал стабильный. Зависимости более менее одинаковая для разных моментов времени.

3 Упражнение 4.3

1. **Задание**

   Откройте файл с историческими данными о ежедневной цене BitCoin и вычислите спектр цен BitCoin как функцию времени. Похоже ли это на белый, розовый или броуновский шум?

2. **Ход работы**

   Посмотрим динамику цен BitCoin за последние 7 лет.

```
import pandas as pd

df = pd.read_csv('res/BTC_USD_2013-10-01_2021-05-05-CoinDesk.csv',
parse_dates=[0])
df
```

Листинг 9: Считывание динамики цен BitCoin

	Currency	Date	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
0	BTC	2013-10-01	123.654990	124.304660	124.751660	122.563490
1	BTC	2013-10-02	125.455000	123.654990	125.758500	123.633830
2	BTC	2013-10-03	108.584830	125.455000	125.665660	83.328330
3	BTC	2013-10-04	118.674660	108.584830	118.675000	107.058160
4	BTC	2013-10-05	121.338660	118.674660	121.936330	118.005660
...	...	...	...	...	...	...
2768	BTC	2021-05-01	57302.646424	53598.879503	57434.933127	53097.762794
2769	BTC	2021-05-02	57677.975222	57741.020910	58511.256049	57062.700071
2770	BTC	2021-05-03	56427.043125	57824.300187	57925.741567	56123.039508
2771	BTC	2021-05-04	57255.306838	56639.439786	59001.359642	56508.240449
2772	BTC	2021-05-05	53658.843121	57218.805329	57246.891191	53613.595218

2773 rows × 6 columns

Рис. 8: Динамика цен BitCoin

Теперь построим график, для большей наглядности

```
ys = df['Closing Price (USD)']
ts = df.index

from thinkdsp import Wave

wave = Wave(ys, ts, framerate=1)
```

```
7           wave.plot()
8           decorate(xlabel='Time (days)')
```

Листинг 10: Построение графика цен BitCoin

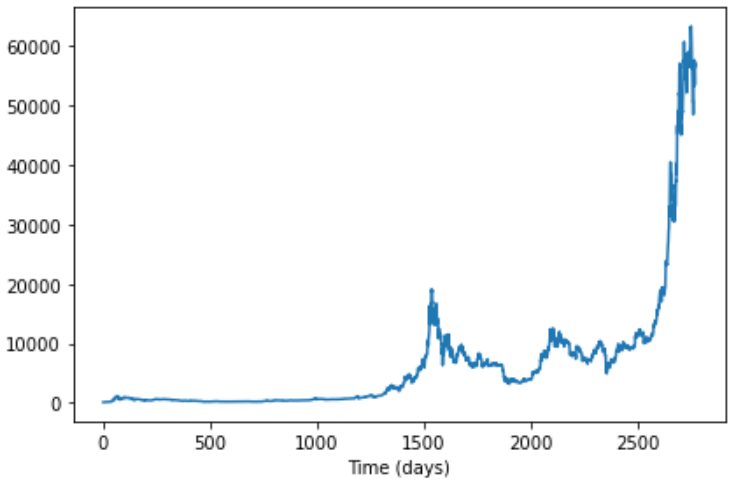

Рис. 9: График цен BitCoin

Построим спектр мощности на логарифмической шкале.

```
1           spectrum = wave.make_spectrum()
2           spectrum.plot_power()
3           decorate(xlabel='Frequency (1/days)', **loglog)
```

Листинг 11: Спектр мощности

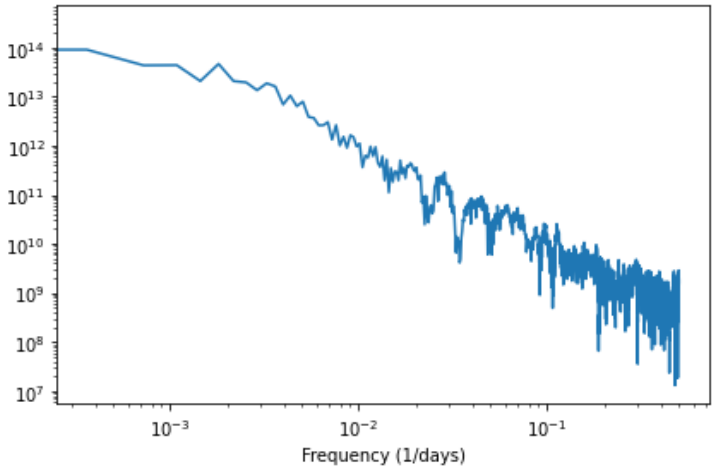

Рис. 10: Спектр мощности

Вычислим наклон прямой

```
spectrum.estimate_slope()[0]

Output
-1.8717417425660792
```

Листинг 12: Спектр мощности

Наклон = -1.8717417425660792.

Как видим, результат похож на розовый шум (наклон лежит в диапазоне от |0| до |2|)

4 Упражнение 4.4

1. **Задание**

   Напишите класс, называемый *UncorrelatedPoissonNoise*, наследующий *thinkdsp._Noise* и предоставляющий *evaluate*. Следует использовать *Np.random.poisson* для генерации случайных величин из распределения Пуассона. Параметр этой функции *lam* - это среднее число частиц за время каждого интервала. Можно использовать атрибут *amp* для определения *lam*. Например, при частоте кадров 10 кГц и *amp* 0.001 получится около 10 "щелчков"(как у счетчика Гейгера) в секунду.

   Сгенерируейте пару секунд UP и прослушайте. Для малых значений *amp*, например 0.001, звук будет как у счетчика Гейгера. При больших значениях он будет похож на белый шум. Вычислите и напечатайте спектр мощности и посмотрите, так ли это.

2. **Ход работы**

   Реализуем класс *UncorrelatedPoissonNoise*

```
from thinkdsp import Noise

class UncorrelatedPoissonNoise(Noise):

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

   Листинг 13: Класс UncorrelatedPoissonNoise

   Теперь создадим и прослушаем UP

```
amp = 0.001
framerate = 10000
duration = 1

signal = UncorrelatedPoissonNoise(amp=amp)
wave = signal.make_wave(duration=duration,
    framerate=framerate)
wave.make_audio()
```

   Листинг 14: Создание и воспроизведение UP

   Звучик как "щелчки"счетчика Гейгера.

Сравним ожидаемое количество частиц с получившимся

```
expected = amp * framerate * duration
actual = sum(wave.ys)
print(expected, actual)


Output
10.0 10
```

Листинг 15: Создание и воспроизведение UP

Видим, что все совпало.

Теперь визуализируем полученный звук

```
wave.plot()
```

Листинг 16: Визуализация щелчков

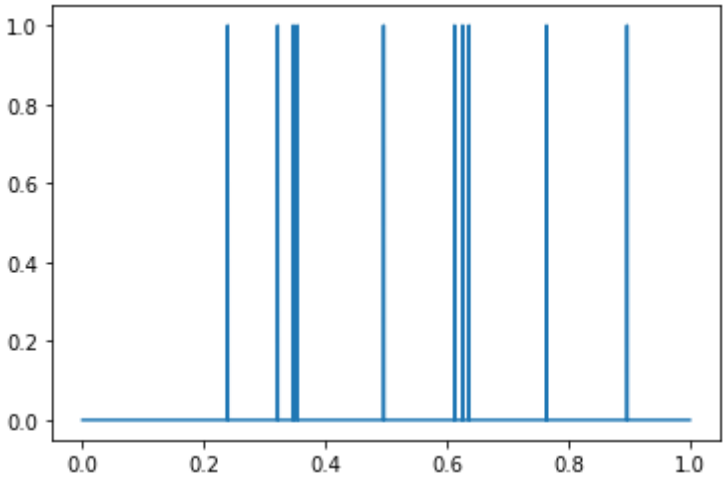

Рис. 11: Визуализация щелчков

Построим спектр мощности на логарифмической шкале.

```
spectrum = wave.make_spectrum()
spectrum.plot_power()
decorate(xlabel='Frequency (1/days)', **loglog)
```

Листинг 17: Спектр мощности

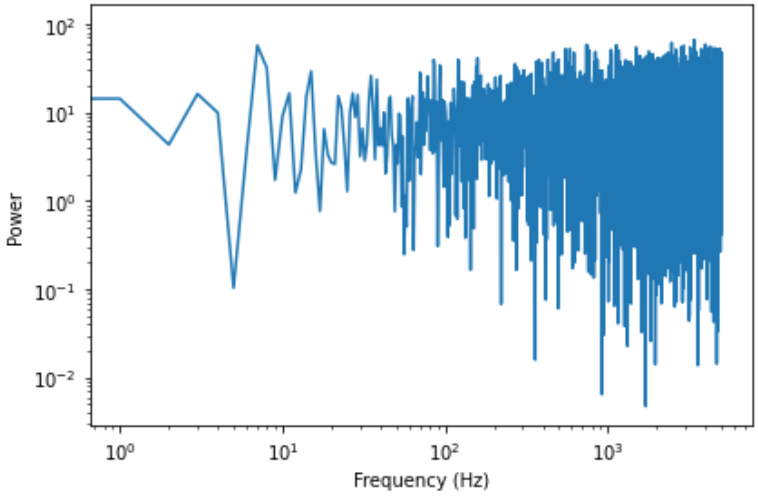

Рис. 12: Спектр мощности

Вычислим наклон.

```
spectrum.estimate_slope().slope

Output
-0.0003900929448715259
```

Листинг 18: Вычисление наклона

Похоже на белый шум.

Теперь увеличим амлитуду сигнала - amp, и посмотрим что будет.

```
amp = 1
framerate = 10000
duration = 1

signal = UncorrelatedPoissonNoise(amp=amp)
wave = signal.make_wave(duration=duration,
    framerate=framerate)
wave.make_audio()
wave.plot()
```

Листинг 19: Создание вопроизведение и построение графика нового звука

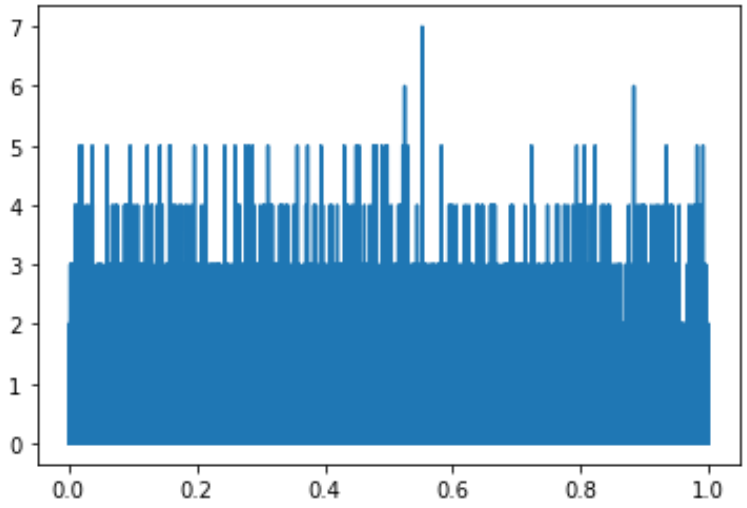

Рис. 13: График нового звука

Спектр сходится на Гауссовском шуме.

Теперь сравним спектры

```
amp = 1
framerate = 10000
duration = 1

signal = UncorrelatedPoissonNoise(amp=amp)
wave = signal.make_wave(duration=duration,
    framerate=framerate)
wave.make_audio()
wave.plot()
```

Листинг 20: Сравнение спектров

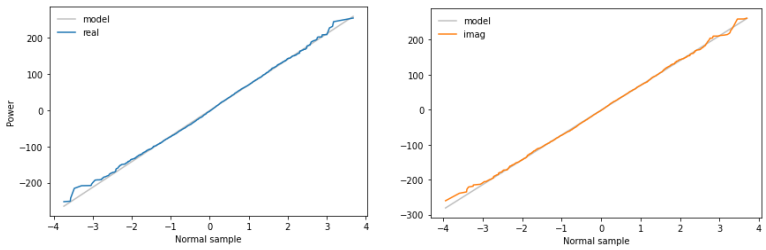

Рис. 14: Сравнение спектров

5 Упражнение 4.5

1. **Задание**

   Изучите алгоритм Восса-МакКартни для генерации розового шума, реализуйте его, вычислите спектр результата и убедитесь, что соотношение между мощностью и частотой соответсвующее.

2. **Ход работы**

   Напишем функцию для реализации алгоритма Восса-МакКартни

```
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)

    # the total number of changes is 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
```

Листинг 21: Алгоритм ВОсса-МакКартни

   Для проверки сгенерируем 12005 значений

```
ys = voss(11025)
ys

Output
array([7.79201163, 7.35145659, 7.20205832, ...,
   9.22659538, 8.72048853,
9.15790613])
```

Листинг 22: Генерация значений

   Теперь создадим из этих значений звук и визуализируем его.

```
ys = voss(11025)
ys

Output
array([7.79201163, 7.35145659, 7.20205832, ...,
    9.22659538, 8.72048853,
9.15790613])
```

Листинг 23: Создание и визуализация звука

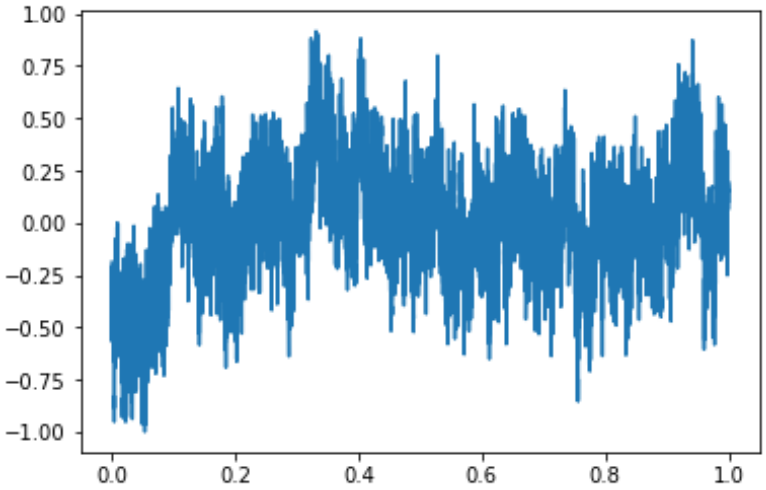

Рис. 15: Визуализация звука

Разброс частот слишком большой для белого шума.

Построим спектр мощности на логарифмической шкале.

```
spectrum = wave.make_spectrum()
spectrum.hs[0] = 0
spectrum.plot_power()
decorate(xlabel='Frequency (Hz)',
**loglog)
```

Листинг 24: Спектр мощности

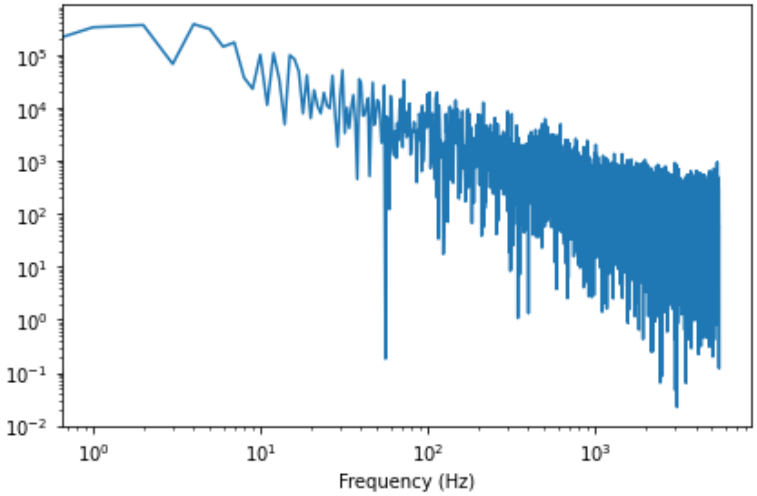

Рис. 16: Спектр мощности

Вычислим наклон

```
spectrum.estimate_slope().slope

Output
-0.9885351381362323
```

Листинг 25: Вычисление наклона

Наклон = -0.9885351381362323

Попробуем получить более точные данные среднего спектра мощности.

Сгенерируем большую выборку, затем с помощью метода Бартлетта вычислим значения и построим график спектра мощности

```
seg_length = 64 * 1024
iters = 100
wave = Wave(voss(seg_length * iters))
len(wave)

spectrum = bartlett_method(wave,
    seg_length=seg_length, win_flag=False)
spectrum.hs[0] = 0
len(spectrum)

spectrum.plot_power()
```

```
11          decorate(xlabel='Frequency (Hz)',
12          **loglog)
```

Листинг 26: Уточнение результата

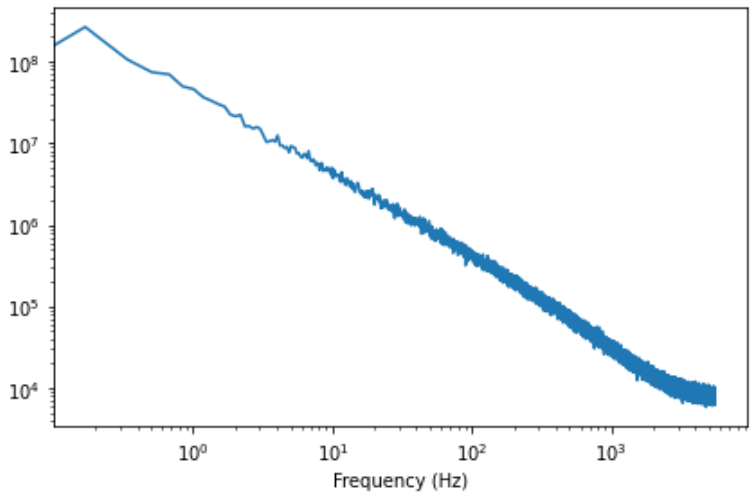

Рис. 17: Более точный спектр мощности

Пересчитаем наклон

```
1           spectrum.estimate_slope().slope
2
3           Output
4           -1.0018254851474162
```

Листинг 27: Пересчитывание наклона

Наклон = -1.0018254851474162

Основываясь на всех полученных результатах выше, можно сказать, что перед нам розовый шум.

6 Вывод

В результате выполнения лабораторной работы получены навыки работы с шумами. Были изучены разные виды шумов: белый, красный, розовый. Также были изучены такие алгоритмы обработки шумов как метот Бартлетта и алгоритм Восса-МакКартни.