

Санкт-Петербургский государственный политехнический  
университет Петра Великого.

**Высшая школа интеллектуальных систем и  
суперкомпьютерных технологий**

ЛАБОРАТОРНАЯ РАБОТА

Шум

Работу выполнила студентка:

\_\_\_\_\_ А. И. Луцкевич  
« \_\_\_\_ » \_\_\_\_\_ 2021 г.

Преподаватель лабораторных  
работ:

\_\_\_\_\_ Н. В. Богач  
« \_\_\_\_ » \_\_\_\_\_ 2021 г.

Санкт-Петербург, 2021 г.

## Суть работы 4.1:

В первой части четвертой лабораторной работы нам необходимо скачать файлы с шумом природы, например, дождь или морские волны. Выделить из этих сигналов спектры и установить, на какой шум похож каждый сигнал.

Я выбрала два звука - звук дождя и звук морских волн. Первым для анализа будет звук дождя.

Мною был скачан музыкальный файл с сайта и помещен в рабочую папку. В коде сначала был прочитан файл и записан в переменную `wave`.

```
In [7]: from thinkdsp import read_wave  
  
wave = read_wave('193335__soundman9826__rain-and-thunder.wav')  
wave.make_audio()
```

Out[7]:



Звук получился аж на 17 минут. Для моей работы это излишне, поэтому выберем секундный отрезок, начиная с 5ой секунды:

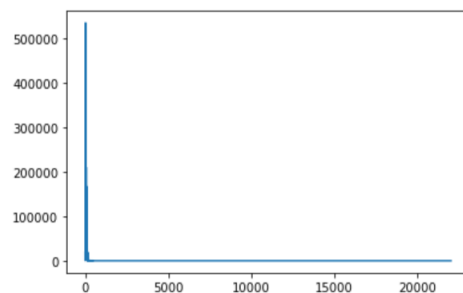
```
In [9]: segment = wave.segment(start=5, duration=1.0)  
segment.make_audio()
```

Out[9]:



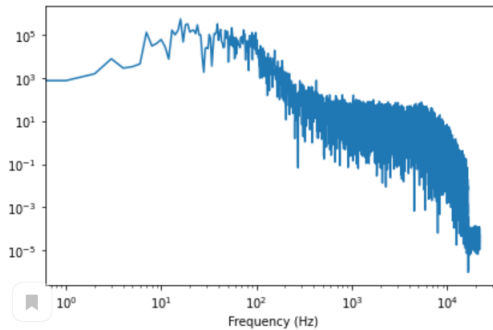
Для анализа был построен спектр этого секундного отрезка:

```
In [10]: spectrum = segment.make_spectrum()  
spectrum.plot_power()
```



По спектру видно, что большей амплитуде соответствует меньшая частота. Из этого следует, что перед нами либо красный, либо же розовый шум. Для более детального анализа построим спектр в логарифмическом масштабе.

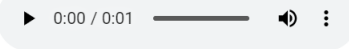
```
In [12]: spectrum.plot_power()
loglog = dict(xscale='log', yscale='log')
decorate(xlabel='Frequency (Hz)', **loglog)
```



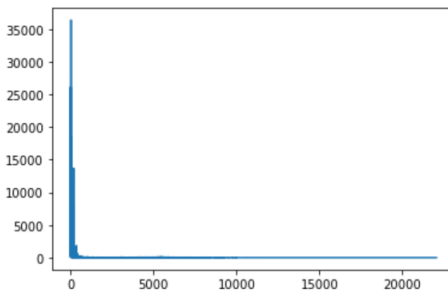
Для сравнения мною был взят еще один отрезок, начиная с 62ой секунды и построен спектр нового отрезка.

```
In [13]: segment = wave.segment(start=62, duration=1.0)
segment.make_audio()
```

Out[13]:

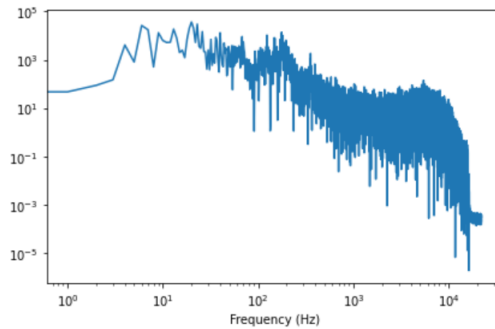


```
In [14]: spectrum = segment.make_spectrum()
spectrum.plot_power()
```



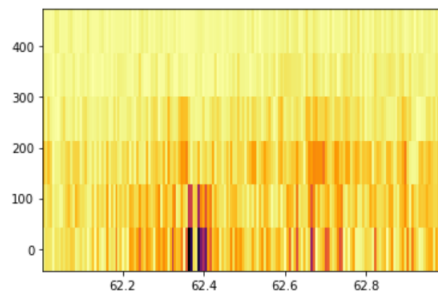
По спектру также видно, что это либо красный, либо же розовый шум. Видно, что спектры двух отрезков похожи, из-за этого делаем вывод, что сигнал не сильно меняется на всем своем протяжении. Построим еще спектр в логарифмическом масштабе.

```
In [15]: spectrum.plot_power()
loglog = dict(xscale='log', yscale='log')
decorate(xlabel='Frequency (Hz)', **loglog)
```



Также последним шагом построим спектограмму звука дождя:

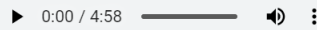
```
In [17]: segment.make_spectrogram(512).plot(high=500)
```



Теперь рассмотрим второй файл, где слышен звук морской волны. Прделаем все те же действия: сохраним в wave, выведем аудиодорожку, обрежем секундный отрезок (со 2ой секунды).

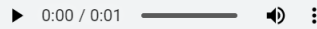
```
In [19]: wave = read_wave('13793__soarer__north-sea.wav')
wave.make_audio()
```

Out[19]:



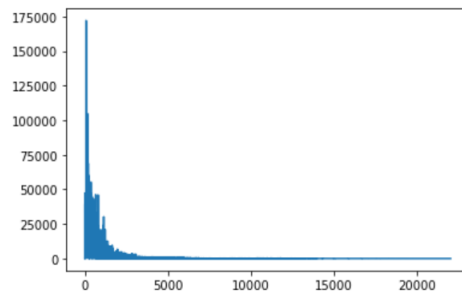
```
In [33]: segment = wave.segment(start=2, duration=1.0)
segment.make_audio()
```

Out[33]:



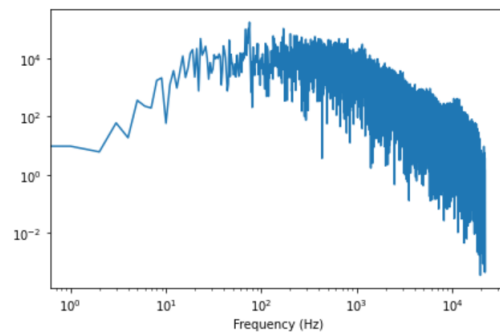
Посмотрим спектр данного сегмента.

```
In [36]: spectrum = segment.make_spectrum()
spectrum.plot_power()
```



По спектру видно, что большей амплитуде соответствует меньшая частота. Из этого следует, что перед нами либо красный, либо же розовый шум. Построим еще спектр в логарифмическом масштабе.

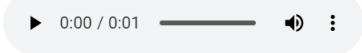
```
In [37]: spectrum.plot_power()
loglog = dict(xscale='log', yscale='log')
decorate(xlabel='Frequency (Hz)', **loglog)
```



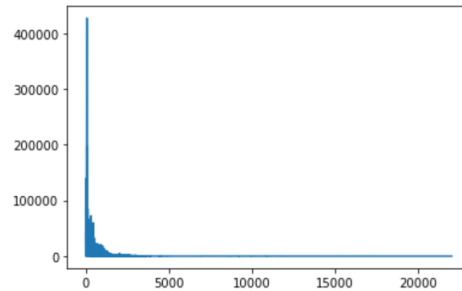
Для сравнения мною был взят еще один отрезок, начиная с 62ой секунды и построен спектр нового отрезка.

```
In [32]: segment2 = wave.segment(start=62, duration=1.0)
segment2.make_audio()
```

Out[32]:



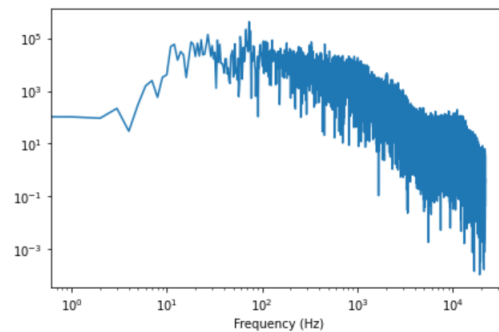
```
In [38]: spectrum = segment2.make_spectrum()
spectrum.plot_power()
```



По спектру также видно, что это либо красный, либо же розовый шум. Видно, что спектры двух отрезков похожи, из-за этого делаем вывод, что сигнал не сильно меняется на всем своем протяжении.

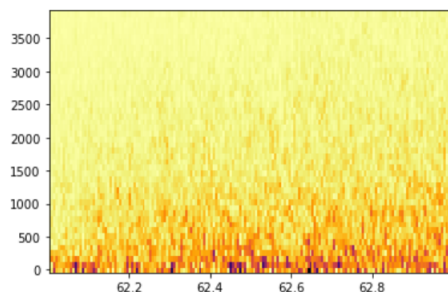
Построим еще спектр в логарифмическом масштабе.

```
In [39]: spectrum.plot_power()
loglog = dict(xscale='log', yscale='log')
decorate(xlabel='Frequency (Hz)', **loglog)
```



Также последним шагом построим спектограмму звука моря:

```
In [40]: segment2.make_spectrogram(512).plot(high=4000)
```



## Суть работы 4.2:

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

Напишем метод Барлетта: сделаем спектрограмму, извлечем спектр, извлечем массив мощности из каждого спектра, вычислим среднюю мощность (как амплитуда) и сделаем спектр со средними амплитудами.

```
In [43]: 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]

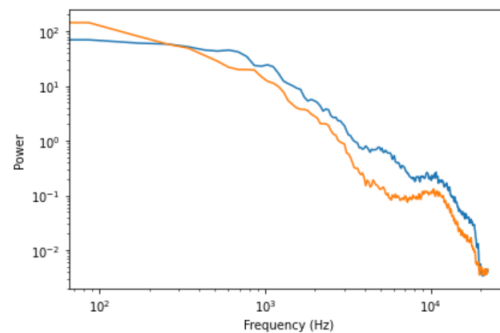
    hs = np.sqrt(sum(psds) / len(psds))
    fs = next(iter(spectrums)).fs

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

Теперь необходимо проверить данный метод. Проверять будем на двух последних отрезках, которые были получены в прошлом пункте (звук морской волны со 2ой и с 62ой секунды).

```
In [44]: from thinkdsp import Spectrum

psd = bartlett_method(segment)
psd2 = bartlett_method(segment2)
psd.plot_power()
psd2.plot_power()
decorate(xlabel='Frequency (Hz)', ylabel='Power', **loglog)
```



Из полученных спектров видно, что есть связь между частотой и амплитудой, но зависимость нелинейная.

## Суть работы 4.3:

Необходимо скачать CSV файл с историческими данными цен на BitCoin. Необходимо вычислить спектр цен BitCoin как функцию времени и установить, на какой шум он больше похож.

Скачаем сначала CSV файл с данными цен BitCoin за последний год.

```
In [8]: import pandas as pd

dataB = pd.read_csv('BTC_USD_2020-04-28_2021-04-27-CoinDesk.csv')
dataB
```

Out[8]:

	Currency	Date	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
0	BTC	2020-04-28	7776.507543	7624.854338	7798.276656	7621.487082
1	BTC	2020-04-29	7761.758784	7788.574229	7793.636018	7677.178774
2	BTC	2020-04-30	8773.106488	7761.758619	8973.079277	7725.542654
3	BTC	2020-05-01	8767.672623	8768.047180	9469.078423	8415.474740
4	BTC	2020-05-02	8853.774484	8767.672143	9073.817530	8593.380363
...	...	...	...	...	...	...
360	BTC	2021-04-23	51965.059559	53830.823864	55471.076372	50500.731862
361	BTC	2021-04-24	50669.144382	51714.073970	52111.185068	47467.912032
362	BTC	2021-04-25	50733.769504	51217.172330	51253.442948	48932.158814
363	BTC	2021-04-26	48542.952203	50177.237403	50668.659025	47105.593068
364	BTC	2021-04-27	53558.707845	49139.230955	54398.592788	48854.581374

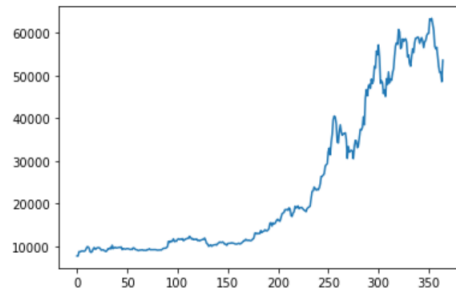
365 rows × 6 columns



Считаем необходимые для нас данные в отдельный список и построим wave по этим данным.

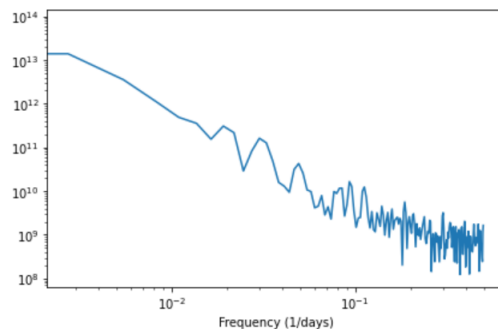
```
In [19]: data = dataB['Closing Price (USD)']  
di = dataB.index
```

```
In [20]: from thinkdsp import Wave  
wave = Wave(data, di, framerate=1)  
wave.plot()
```



Построим логарифмический спектр по предоставленным данным.

```
In [21]: spectrum = wave.make_spectrum()  
spectrum.plot_power()  
decorate(xlabel='Frequency (1/days)', **loglog)
```



Вычислим наклон полученного спектра.

```
In [55]: spectrum.estimate_slope()[0]
```

```
Out[55]: -1.7260415653227166
```

Наклон получился равным -1,726, поэтому можно сделать вывод, что перед нами розовый шум, т.к. находится между 0 и -2.

## Суть работы 4.4:

Реализовать класс `UncorrelatedPoissonNoise`, наследующий `Noise` и предоставляющий `evaluate`. Необходимо сгенерировать случайные величины из распределения Пуассона, а так же пару секунд UP и прослушать. При ма-

лых значениях `amp` звук будет похож на счетчик Гейгера, а при больших на белый шум. Вычислить и напечатать спектр для этих сигналов.

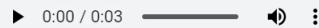
Напишем класс `UncorrelatedPoissonNoise`.

```
In [56]: from thinkdsp import Noise
class UncorrelatedPoissonNoise(Noise):
    def evaluate(self, ts):
        ys = np.random.poisson(self.amp, len(ts))
        return ys
```

Далее создадим сигнал с `amp = 0.001`, переведем в аудио представление:

```
In [58]: signal = UncorrelatedPoissonNoise(amp=0.001)
wave = signal.make_wave(duration=3, framerate=10000)
wave.make_audio()
```

Out[58]:



Что у нас получился звук очень похожий на счетчик Гейгера.

Необходимо сверить получившиеся число количество частиц с ожидаемым.

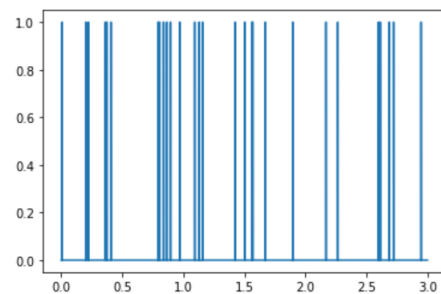
```
In [59]: expected = 0.001 * 10000 * 3
actual = sum(wave.ys)
print(expected, actual)
```

30.0 28

Числа находятся блико друг к другу.

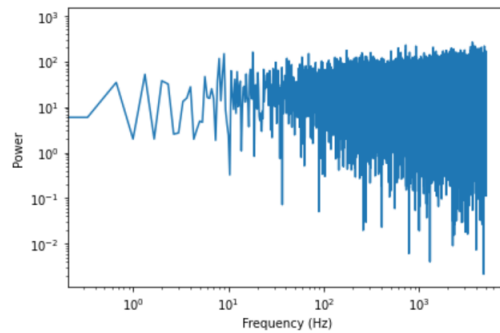
Построим полученные значения.

```
In [60]: wave.plot()
```



Узнаем мощность спектра и посмотрим на наклон полученного спектра.

```
In [27]: spectrum = wave.make_spectrum()
spectrum.plot_power()
decorate(xlabel='Frequency (Hz)',ylabel='Power',**loglog)
```



```
In [28]: spectrum.estimate_slope()[0]
```

```
Out[28]: -2.817769741363608e-05
```

Так как число очень маленькое, можно сделать вывод, что перед нами белый шум.

## Суть работы 4.5:

Необходимо реализовать алгоритм Восса-МакКартни, вычислить спектр и убедиться, что соотношение между мощностью и частотой соответствующие.

Реализуем метод `voss`, где `nrows` - количество генерируемых значений, `ncols` - количество случайных источников для добавления, а возвращает массив NumPy.

```
In [1]: 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
```

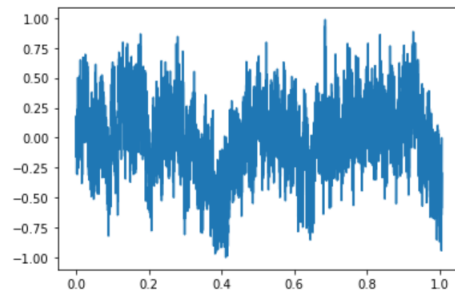
Необходимо протестировать данный момент. Для этого создадим аудио из этого сигнала и построим его.

```
In [11]: wave = Wave(voss(11100))
         wave.unbias()
         wave.normalize()
         wave.make_audio()
```

Out[11]:

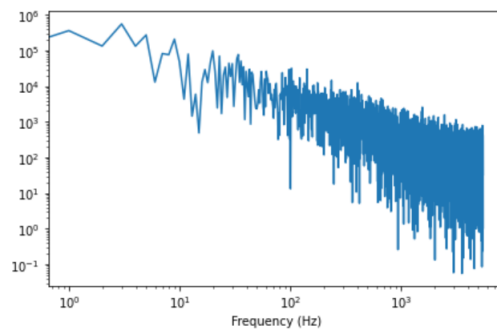


```
In [12]: wave.plot()
```



Послушав и посмотрев, можно сказать, что это простой шум. Также посмотрим на спектр мощности, чтобы убедиться в зависимости частоты от амплитуды.

```
In [15]: spectrum = wave.make_spectrum()
         spectrum.hs[0] = 0
         spectrum.plot_power()
         loglog = dict(xscale='log', yscale='log')
         decorate(xlabel='Frequency (Hz)', **loglog)
```



Посмотрим на наклон полученного спектра.

```
In [16]: spectrum.estimate_slope().slope
```

Out[16]: -0.9694775878926188

Полученный сигнал является розовым шумом, так как значение близко к -1.

## **Заключение:**

По итогу выполнения данной лабораторной работы я изучила понятие шума и его виды. Также научилась строить спектры мощности шумов.

В качестве практики были преобразованы данные по ценам BitCoin и представлены в виде шума. Также был реализован алгоритм Восса-МакКартни и протестирован.