# Лабораторная работа №4
# Шум

Кобыжев Александр

7 апреля 2021 г.

# Оглавление

# Список иллюстраций

# Листинги

# Глава 1

# Теоретическая часть о свойствах преобразования Фурье

## 1.1 Общие сведения

Преобразование Фурье функции $f$ вещественной переменной является интегральным и задаётся следующими формулами:

$$\text{Прямое:} \quad F(\nu) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \nu t} dt$$

$$\text{Обратное:} \quad f(t) = \int_{-\infty}^{\infty} F(\nu) e^{2\pi i \nu t} d\nu$$

## 1.2 Свойства

### 1.2.1 Линейность

По определению, для некоторого векторного пространства $(V, K, +, \cdot)$, $a, b \in V$, $\gamma \in K$:

$$f : V \to V \text{ - линейна} \iff \begin{cases} \gamma \cdot f(a) = f(\gamma \cdot a) \\ f(a) + f(b) = f(a + b) \end{cases}$$

Очевидно, что преобразование Фурье (ПФ) удовлетворяет этому условию (как функция на $(\mathbb{R} \to \mathbb{R}, \mathbb{C}, +, \cdot)$), а следовательно:

$$Fourier\left(\sum_i \alpha_i \phi_i(t)\right) = \sum_i \alpha_i \cdot Fourier(\phi_i(t)) = \sum_i \alpha_i \Phi_i(\nu)$$

### 1.2.2 Смещение функции

При смещении функции $\phi(t)$ на $\Delta t$ результат ПФ умножается на $e^{2\pi i\nu\Delta t}$. Пусть $t' = t + \Delta t$, тогда:

$$Fourier\left(\phi(t+\Delta t)\right) = \int_{-\infty}^{\infty} \phi(t+\Delta t) e^{-2\pi i\nu t} dt = \int_{-\infty}^{\infty} \phi(t') e^{-2\pi i\nu(t'-\Delta t)} dt$$

Так как $dt' = d(t+\Delta t) = dt$, то:

$$\int_{-\infty}^{\infty} \phi(t') e^{-2\pi i\nu(t'-\Delta t)} dt' = e^{2\pi i\nu\Delta t} \cdot \int_{-\infty}^{\infty} \phi(t') e^{-2\pi i\nu t'} dt' = e^{2\pi i\nu\Delta t} \cdot F(\nu)$$

### 1.2.3 Масштабирование функции

Пусть $t' = \alpha t$, тогда:

$$Fourier\left(\phi(\alpha t)\right) = \int_{-\infty}^{\infty} \phi(\alpha t) e^{-2\pi i\nu t} dt = \int_{-\infty}^{\infty} \phi(t') e^{-2\pi i\nu \frac{t'}{\alpha}} dt$$

Так как $dt' = \alpha dt$, то для $a > 0$:

$$\int_{-\infty}^{\infty} \phi(t') e^{-2\pi i\nu \frac{t'}{\alpha}} dt = \frac{1}{\alpha} \int_{-\infty}^{\infty} \phi(t') e^{-2\pi i \frac{\nu}{\alpha} t'} dt = \frac{1}{\alpha} \Phi\left(\frac{\nu}{\alpha}\right)$$

Для $a < 0$ получится $dt' < 0$ при $dt > 0$. При этом нужно поменять пределы интегрирования местами, тогда получим результат с отрицательным знаком:

$$-\frac{1}{\alpha} \Phi\left(\frac{\nu}{\alpha}\right)$$

Таким образом, в одной форме это:

$$\frac{1}{|\alpha|}\Phi\left(\frac{\nu}{\alpha}\right)$$

Вывод: при сжатии функции по времени в $\alpha$ раз, её ПФ расширяется по частоте в $\alpha$ раз.

### 1.2.4 Перемножение функции

ПФ произведения двух функций - это свёртка их ПФ.

$$\begin{aligned}
Fourier\left(\phi(t)\xi(t)\right) &= \int_{-\infty}^{\infty}\phi(t)\xi(t)e^{-2\pi i\nu t}dt \\
&= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty}\Phi(k)e^{2\pi ikt}dk\right)\xi(t)e^{-2\pi i\nu t}dt \\
&= \int_{-\infty}^{\infty}\Phi(k)\left(\int_{-\infty}^{\infty}\xi(t)e^{2\pi i(k-\nu)t}dt\right)dk \\
&= \int_{-\infty}^{\infty}\Phi(k)\left(\int_{-\infty}^{\infty}\xi(t)e^{-2\pi i(\nu-k)t}dt\right)dk \\
&= \int_{-\infty}^{\infty}\Phi(k)\Xi(\nu-k)dk \\
&= \left(\Phi * \Xi\right)(\nu)
\end{aligned}$$

### 1.2.5 Свёртывание функции

ПФ свёртки двух функций есть произведение ПФ этих функций. Доказывается аналогично в силу «симметрии» прямого и обратного преобразований Фурье.

### 1.2.6 Дифференцирование функции

При дифференцировании $\phi(t)$ по $t$ её ПФ умножается на $2\pi i\nu$.

$$
\begin{aligned}
Fourier\left(\frac{d\phi(t)}{dt}\right) &= \int_{-\infty}^{\infty} \frac{d\phi(t)}{dt} e^{-2\pi i\nu t} dt \\
&= \int_{-\infty}^{\infty} e^{-2\pi i\nu t} d\phi(t) \\
&= \phi(t)e^{-2\pi\nu t}\Big|_{-\infty}^{\infty} - \int_{-\infty}^{\infty} \phi(t) d\left(e^{-2\pi i\nu t}\right) \\
&= \phi(t)e^{-2\pi\nu t}\Big|_{-\infty}^{\infty} + 2\pi i\nu \int_{-\infty}^{\infty} \phi(t)e^{-2\pi i\nu t} dt \\
&= \phi(t)e^{-2\pi\nu t}\Big|_{-\infty}^{\infty} + 2\pi i\nu \cdot \Phi(\nu)
\end{aligned}
$$

Прямое и обратное преобразование Фурье существует для функций с ограниченной энергией, то есть:

$$\int_{-\infty}^{\infty} |\phi(t)|^2 dt \neq \infty$$

И из этого следует, что первое слагаемое равно 0.

### 1.2.7 Интегрирование функции

При интегрировании ПФ делится на $2\pi i\nu$.

$$
\begin{aligned}
Fourier\left(\int_{-\infty}^{t} \phi(t')dt'\right) &= \int_{-\infty}^{\infty} \left(\int_{-\infty}^{t} \phi(t')dt'\right) e^{-2\pi i\nu t} dt \\
&= -\frac{1}{2\pi i\nu} \cdot \int_{-\infty}^{\infty} \left(\int_{-\infty}^{t} \phi(t')dt'\right) d\left(e^{-2\pi i\nu t}\right) \\
&= -\frac{1}{2\pi i\nu} \cdot \left[e^{-2\pi i\nu t} \int_{-\infty}^{t} \phi(t')dt'\Big|_{-\infty}^{\infty} - \int_{-\infty}^{\infty} e^{-2\pi i\nu t} d\left(\int_{-\infty}^{t} \phi(t')dt'\right)\right] \\
&= -\frac{1}{2\pi i\nu} \cdot \left[e^{-2\pi i\nu t} \int_{-\infty}^{t} \phi(t)dt\Big|_{-\infty}^{\infty} - \int_{-\infty}^{\infty} e^{-2\pi i\nu t}\phi(t)dt\right] \\
&= -\frac{1}{2\pi i\nu} \cdot \left[0 - \int_{-\infty}^{\infty} e^{-2\pi i\nu t}\phi(t)dt\right] \\
&= \frac{1}{2\pi i\nu} \cdot \int_{-\infty}^{\infty} e^{-2\pi i\nu t}\phi(t)dt \\
&= \frac{1}{2\pi i\nu} \cdot \Phi(\nu)
\end{aligned}
$$

0 возникает потому, что $\int_{-\infty}^{\infty} \phi(t')dt' = 0$.

### 1.2.8 Обратимость

Преобразования обратимы, причём обратное преобразование имеет практически такую же форму, как и прямое преобразование.

# Глава 2

# Упражнение 4.1

Для данного упражнения с предложенного сайта я скачал запись обстановки в ресторане.

```
wave =
    thinkdsp.read_wave('173920__matias44__murmullo-restaurant.wav')
wave.make_audio()
```

Листинг 2.1: Прослушивание скачанного шума

Выберем короткий отрезок:

```
segment = wave.segment(start=1.5, duration=1.0)
segment.make_audio()
```

Листинг 2.2: Выбор короткого отрезка

Теперь составим его спектр.

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

Листинг 2.3: Спектр звука

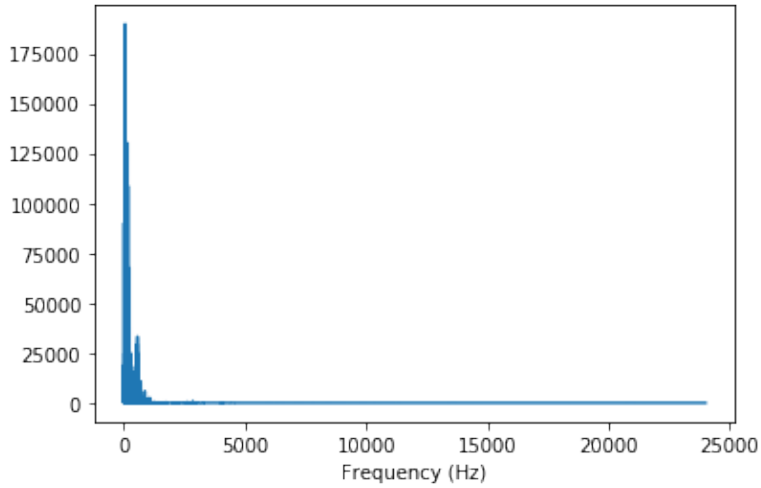


Рис. 2.1: Спектр звука

Амплитуда падает с частотой, поэтому это может быть красный или розовый шум. Мы можем проверить это, посмотрев на спектр мощности в логарифмической шкале.

```
spectrum.plot_power()
thinkplot.config(xlabel='Frequency (Hz)',
                 xscale='log',
                 yscale='log')
```

Листинг 2.4: Спектр мощности звука

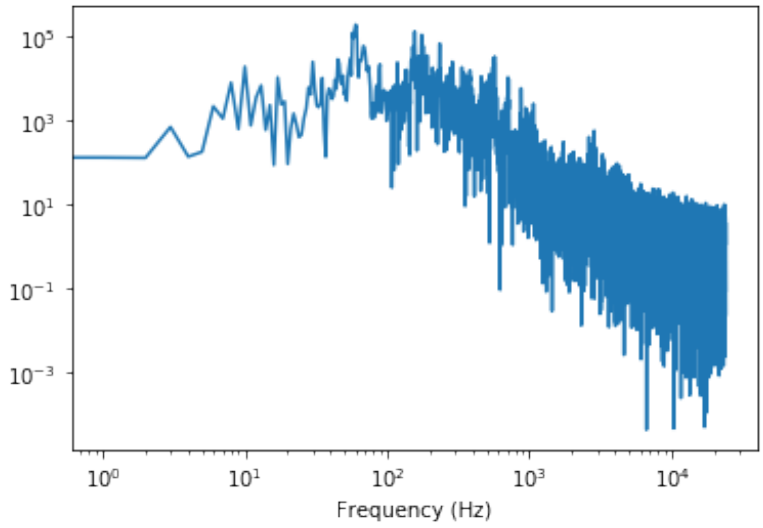


Рис. 2.2: Спектр мощности звука

Эта структура с увеличением, а затем и с уменьшением амплитуды кажется обычным явлением для естественных источников шума. Чтобы увидеть, как спектр меняется с течением времени, я выберу другой сегмент.

```
segment2 = wave.segment(start=2.5, duration=1.0)
segment2.make_audio()
```

Листинг 2.5: Выбор другого сегмента звука

Теперь рассмотрим два спектра:

```
spectrum2 = segment2.make_spectrum()
spectrum.plot_power()
spectrum2.plot_power(color='#beaed4')
thinkplot.config(xlabel='Frequency (Hz)',
                 ylabel='Amplitude')
```

Листинг 2.6: Спектр двух звуков

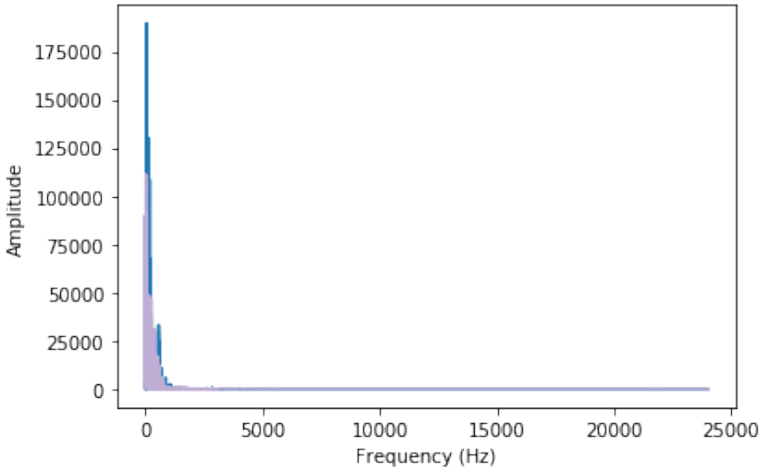


Рис. 2.3: Спектр двух звуков

Теперь рассмотрим график мощности в логарифмическом масштабе.

```
spectrum.plot_power()
spectrum2.plot_power(color='#beaed4')
thinkplot.config(xlabel='Frequency (Hz)',
                 ylabel='Amplitude',
                 xscale='log',
                 yscale='log')
```

Листинг 2.7: Спектр мощности двух звуков

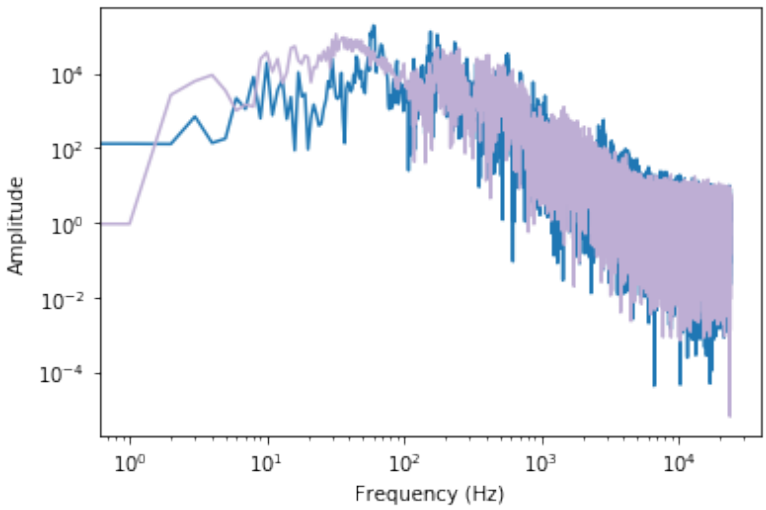


Рис. 2.4: Спектр мощности двух звуков

Таким образом, структура кажется неизменной с течением времени. Мы также можем посмотреть на спектрограмму:

```
segment.make_spectrogram(512).plot(high=5000)
```

Листинг 2.8: Спектрограмма звука

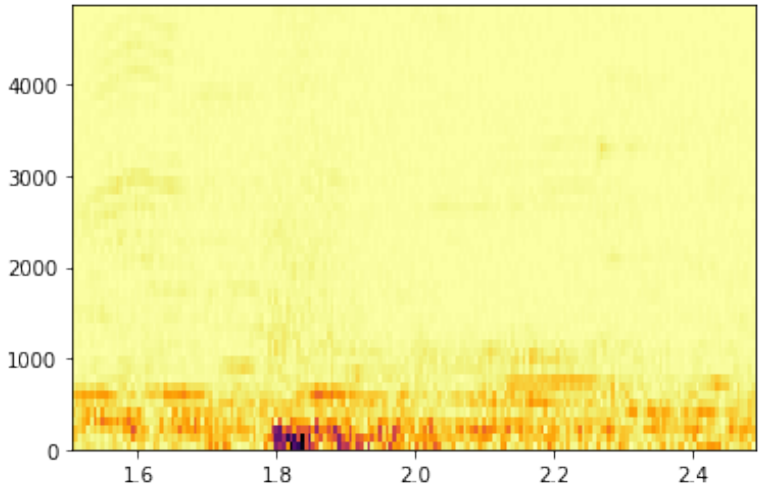


Рис. 2.5: Спектрограмма звука

В этом сегменте общая амплитуда падает, но смесь частот кажется стабильной.

# Глава 3

# Упражнение 4.2

`bartlett_method` создает спектрограмму и извлекает `spec_map`, который отображает время на объекты Spectrum. Он вычисляет PSD для каждого спектра, складывает их и помещает результаты в объект Spectrum.

```
def bartlett_method(wave, seg_length=512, win_flag=True):
    # make a spectrogram and extract the spectrums
    spectro = wave.make_spectrogram(seg_length, win_flag)
    spectrums = spectro.spec_map.values()

    # extract the power array from each spectrum
    psds = [spectrum.power for spectrum in spectrums]

    # compute the root mean power (which is like an amplitude)
    hs = np.sqrt(sum(psds) / len(psds))
    fs = next(iter(spectrums)).fs

    # make a Spectrum with the mean amplitudes
    spectrum = thinkdsp.Spectrum(hs, fs, wave.framerate)
    return spectrum
```

Листинг 3.1: Функция bartlett_method

Построим сегменты:

```
psd = bartlett_method(segment)
psd2 = bartlett_method(segment2)

psd.plot_power()
psd2.plot_power(color='#beaed4')

thinkplot.config(xlabel='Frequency (Hz)',
```

```
8                 ylabel='Power',
9                 xscale='log',
10                yscale='log')
```

Листинг 3.2: Сегменты звуков

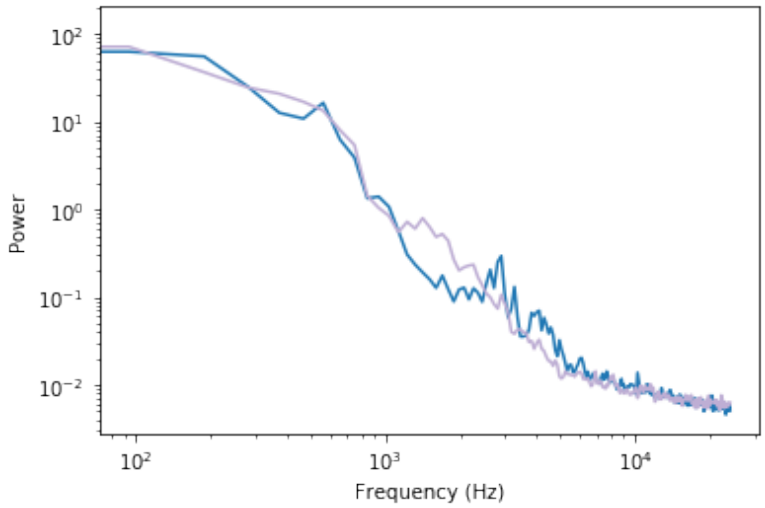


Рис. 3.1: Сегменты звуков

Теперь мы можем более чётко увидеть взаимосвязь между мощностью и частотой. Это не простая линейная зависимость, но она одинакова для разных сегментов, таких как около 1000 Гц, 6000 Гц и выше 10000 Гц.

# Глава 4

# Упражнение 4.3

На предложенной веб-странице я скачал данные о ежедневной цене BitCoin в течение года.

```
data = pd.read_csv('BTC_USD_2020-04-08_2021-04-07-CoinDesk.csv')
data
```

Листинг 4.1: Таблица данных

| | Currency | Date | Closing Price (USD) | 24h Open (USD) | 24h High (USD) | 24h Low (USD) |
|---|---|---|---|---|---|---|
| **0** | BTC | 2020-04-08 | 7175.667477 | 7277.704282 | 7464.732245 | 7081.639209 |
| **1** | BTC | 2020-04-09 | 7367.293398 | 7175.669418 | 7424.743721 | 7155.211053 |
| **2** | BTC | 2020-04-10 | 7321.816614 | 7366.900961 | 7399.469133 | 7125.775519 |
| **3** | BTC | 2020-04-11 | 6866.398189 | 7321.815746 | 7325.324778 | 6752.593664 |
| **4** | BTC | 2020-04-12 | 6873.848495 | 6872.137266 | 6949.788875 | 6777.889694 |
| **...** | ... | ... | ... | ... | ... | ... |
| **360** | BTC | 2021-04-03 | 58821.626994 | 58726.084566 | 60101.752326 | 58478.598349 |
| **361** | BTC | 2021-04-04 | 57517.798773 | 58958.428985 | 59713.210136 | 57185.768006 |
| **362** | BTC | 2021-04-05 | 58177.402764 | 57134.860051 | 58540.984706 | 56552.222275 |
| **363** | BTC | 2021-04-06 | 58843.559540 | 58230.675538 | 59243.036175 | 56846.969047 |
| **364** | BTC | 2021-04-07 | 58040.187602 | 59133.655740 | 59484.199475 | 57421.853085 |

Рис. 4.1: Таблица данных

Визуализируем скачанные данные.

```
wave = thinkdsp.Wave(data['Closing Price (USD)'], data.index,
    framerate=1)
wave.plot()
thinkplot.config(xlabel='Time (days)')
```

Листинг 4.2: Визуализация данных

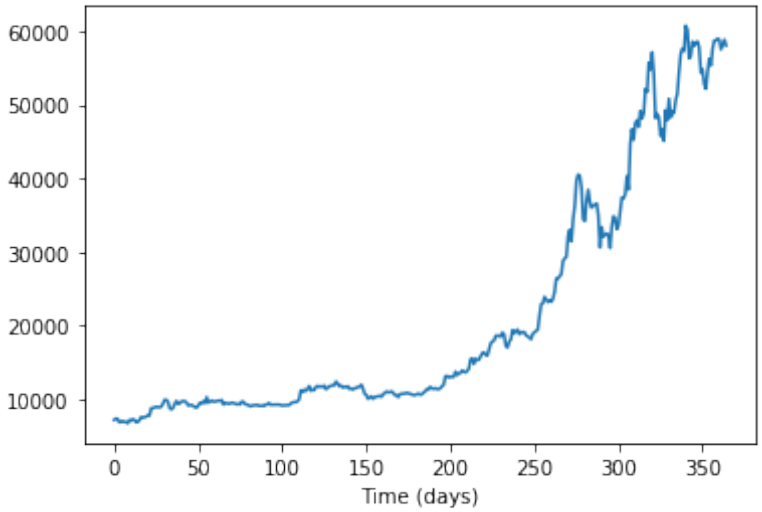


Рис. 4.2: Визуализация данных

Построим спектр искусственно созданного звука, где частотой будет выступать 1/дни.

```
spectrum = wave.make_spectrum()
spectrum.plot_power()
thinkplot.config(xlabel='Frequency (1/days)',
                 xscale='log', yscale='log')
```

Листинг 4.3: Спектр искусственного звука

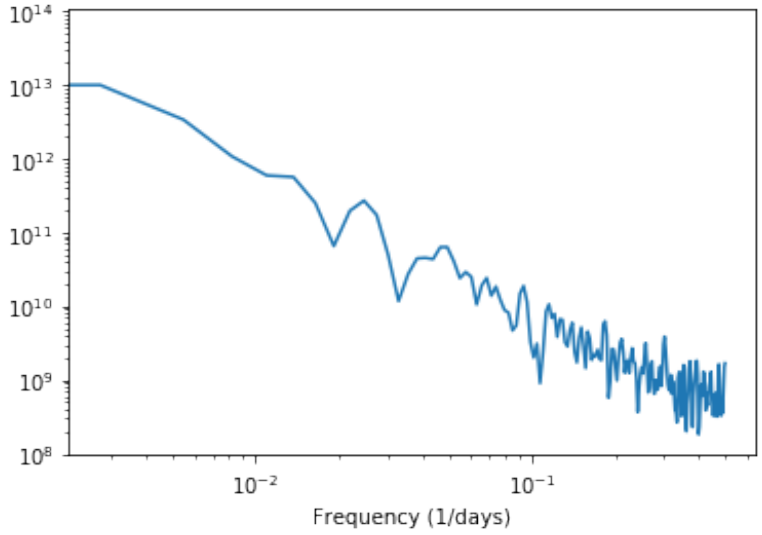


Рис. 4.3: Спектр искусственного звука

Спектр сход с прямой линией, поэтому можно предположить, что это "красный"или "розовый"шум. Проверим это, узнав наклон прямой.

```
spectrum.estimate_slope()[0]
```

Листинг 4.4: Наклон прямой

Наклон составляет `-1.8216406076974851`, что похоже на красный шум (который должен иметь наклон -2).

# Глава 5

# Упражнение 4.4

Созданный класс `UncorrelatedPoissonNoise` представляет некоррелированный пуассоновский шум. Оценивает сигнал в заданное время.

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

Листинг 5.1: Созданный класс UncorrelatedPoissonNoise

Рассмотрим как это звучит при низких уровнях «радиации».

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

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

Листинг 5.2: Создание звука

Звук действительно похож на звуки от счётчика Гейгера, будто бы находишься где-то в Припяти. Чтобы убедиться, что все работает, мы сравниваем ожидаемое количество частиц и фактическое количество:

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

Листинг 5.3: Создание звука

Количество частиц в обоих случаях равняется 10. Теперь рассмотрим

полученный звук:

```
wave.plot()
```

Листинг 5.4: Визуализация звука

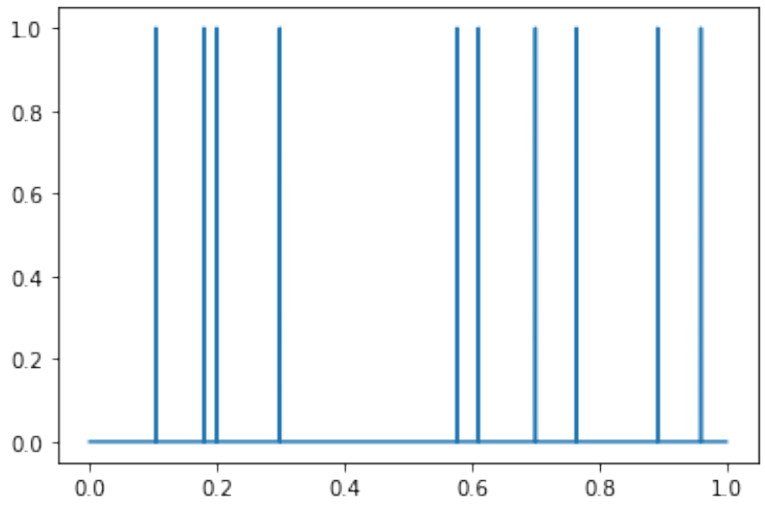


Рис. 5.1: Визуализация звука

Рассмотрим спектр мощности в логарифмическом масштабе:

```
spectrum = wave.make_spectrum()
spectrum.plot_power()
thinkplot.config(xlabel='Frequency (Hz)',
                 ylabel='Power',
                 xscale='log',
                 yscale='log')
```

Листинг 5.5: Спектр мощности звука

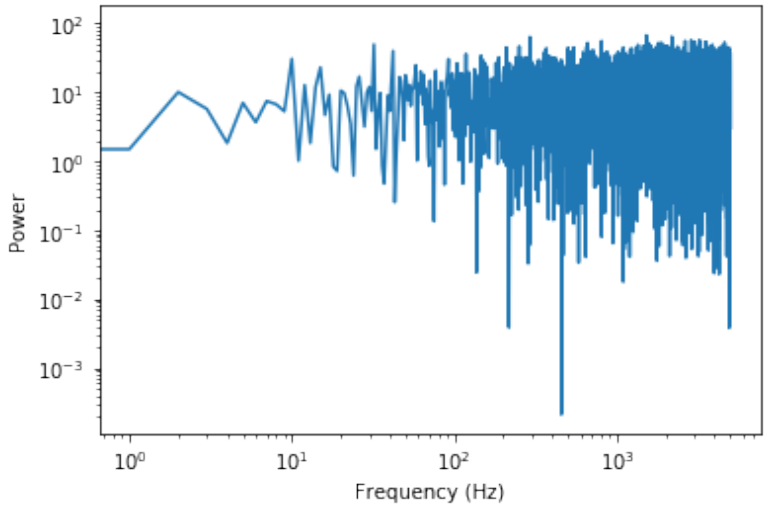


Рис. 5.2: Спектр мощности звука

Рассмотрим наклон:

```
spectrum.estimate_slope().slope
```

Листинг 5.6: Наклон прямой

Похоже на белый шум, а крутизна близка к 0 (`-0.0006437981022653128`).

При более высокой скорости поступления это больше похоже на белый шум:

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

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

Листинг 5.7: Создание нового звука

Построим его график.

```
wave.plot()
```

Листинг 5.8: Визуализация нового звука

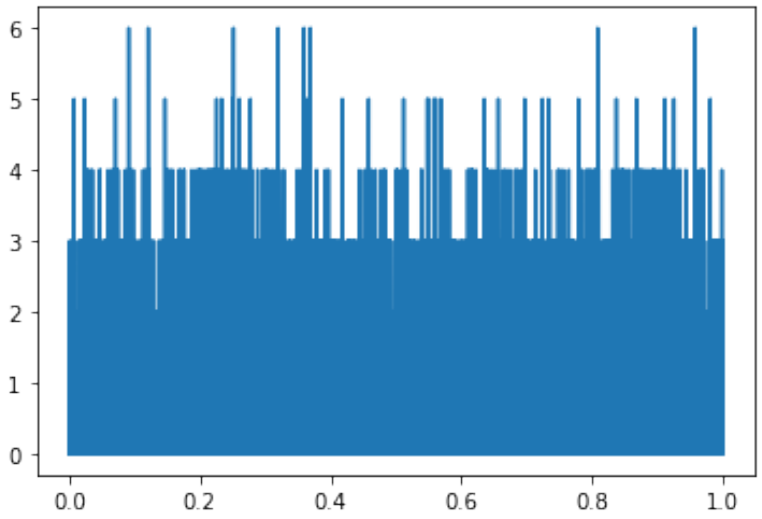


Рис. 5.3: Визуализация нового звука

И спектр сходится на гауссовском шуме.

```
spectrum = wave.make_spectrum()
spectrum.hs[0] = 0

thinkplot.preplot(2, cols=2)
thinkstats2.NormalProbabilityPlot(spectrum.real, label='real')
thinkplot.config(xlabel='Normal sample',
                 ylabel='Power',
                 legend=True,
                 loc='lower right')

thinkplot.subplot(2)
thinkstats2.NormalProbabilityPlot(spectrum.imag, label='imag')
thinkplot.config(xlabel='Normal sample',
                     loc='lower right')
```

Листинг 5.9: Сравнение спектров

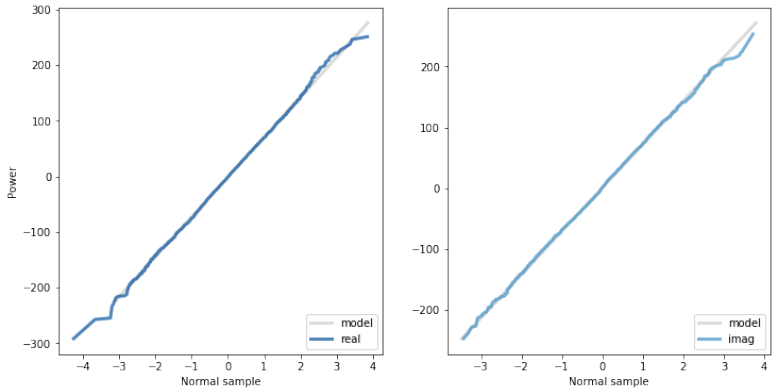


Рис. 5.4: Сравнение спектров

# Глава 6

# Упражнение 4.5

Вот весь процесс в функции: Создает розовый шум с помощью 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 6.1: Создание функции

Чтобы проверить это, я сгенерирую 12005 значений:

```
ys = voss(12005)
ys
```

Листинг 6.2: Генерация значений

Теперь создадим из них звук:

```
wave = thinkdsp.Wave(ys)
wave.unbias()
wave.normalize()
```

Листинг 6.3: Создание звука

Теперь посмотрим на его визуализацию.

```
wave.plot()
```

Листинг 6.4: Визуализация звука

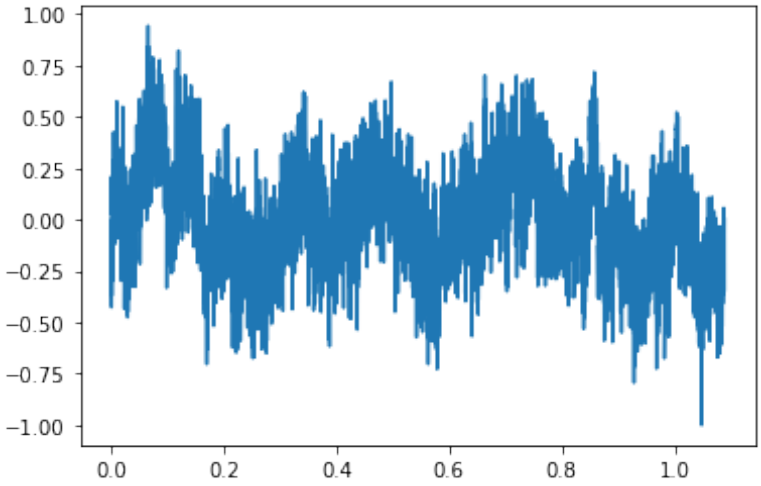


Рис. 6.1: Визуализация звука

Как и ожидалось, это больше похоже на случайное блуждание, чем на белый шум, но более случайное, чем на красный шум. Теперь рассмотрим спектр мощности:

```
spectrum = wave.make_spectrum()
spectrum.hs[0] = 0
spectrum.plot_power()
thinkplot.config(xlabel='Frequency (Hz)',
                 xscale='log',
                 yscale='log')
```

Листинг 6.5: Спектр мощности звука

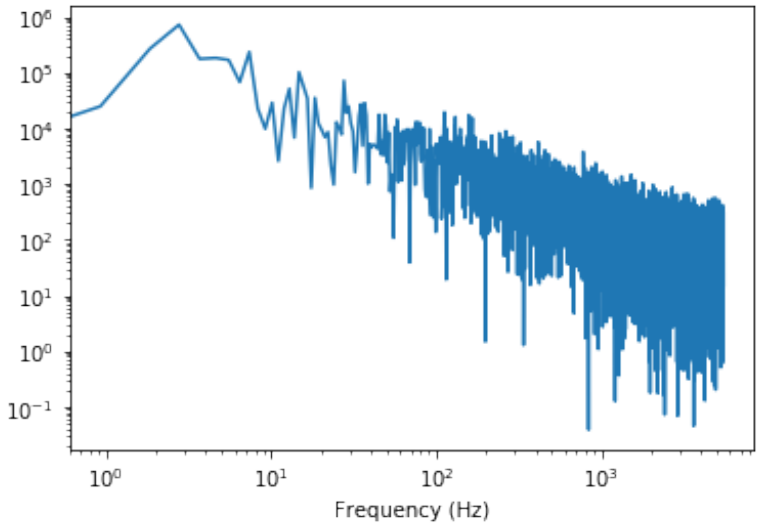


Рис. 6.2: Спектр мощности звука

Посмотрим на наклон:

```
spectrum.estimate_slope().slope
```

Листинг 6.6: Наклон прямой

Расчетный наклон близок к -1 (`-1.0129573459835064`).

Мы можем лучше понять средний спектр мощности, сгенерировав более длинную выборку:

```
seg_length = 40 * 124
iters = 100
wave = thinkdsp.Wave(voss(seg_length * iters))
len(wave)
```

Листинг 6.7: Генерация более длинной выборки

И используя метод Барлетта для вычисления среднего.

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

Листинг 6.8: Использование метода Барлетта

Это довольно близко к прямой линии с некоторой кривизной на самых высоких частотах, если рассматривать спектр мощности звука.

```
spectrum.plot_power()
thinkplot.config(xlabel='Frequency (Hz)',
                 xscale='log',
                 yscale='log')
```

Листинг 6.9: Спектр мощности звука

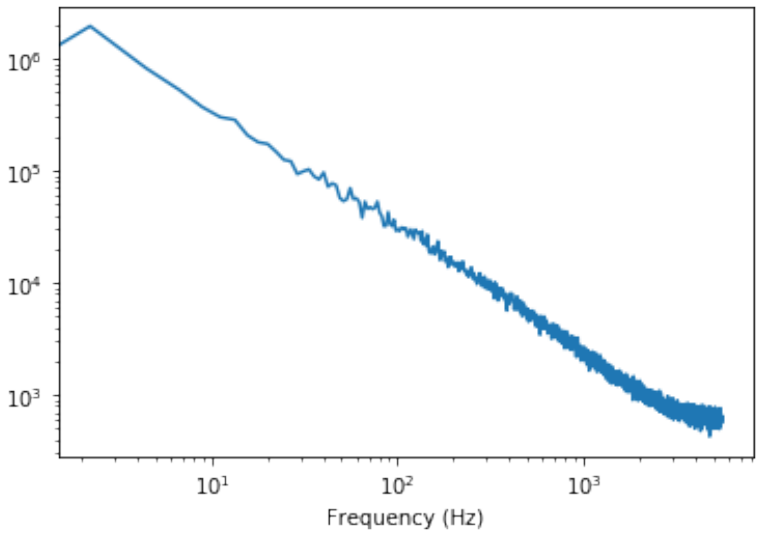


Рис. 6.3: Спектр мощности звука

Посмотрим на наклон:

```
spectrum.estimate_slope().slope
```

Листинг 6.10: Наклон прямой

Наклон теперь более близок к -1 (`-1.0048368551745679`).

# Глава 7

# Выводы

Во время выполнения лабораторной работы получены навыки работы с различными видами шумов. Также получены навыки создания этих шумов через различные данные.