Лабораторная работа 3

Луняк Николай

1 марта 2021 г.

Оглавление

Список иллюстраций

Листинги

Глава 1

Пробуем примеры из `chap03.ipynb`

Тут надо просто посмотреть на примеры из третьей главы, послушать всякие звуки и убедиться, что все работает.

```
In [19]: def eye_of_sauron(start, end):
             """Plots the spectrum of a chirp.

             start: initial frequency
             end: final frequency
             """
             signal =  Chirp(start=start, end=end)
             wave = signal.make_wave(duration=0.5)
             spectrum = wave.make_spectrum()

             spectrum.plot(high=1200)
             decorate(xlabel='Frequency (Hz)', ylabel='Amplitude')
```

```
In [20]: slider1 = widgets.FloatSlider(min=100, max=1000, value=100, step=50)
         slider2 = widgets.FloatSlider(min=100, max=1000, value=200, step=50)
         interact(eye_of_sauron, start=slider1, end=slider2);
```

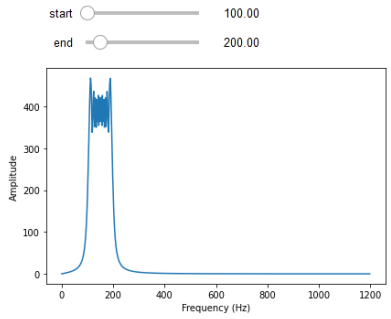

Рис. 1.1: Работает

Проверять разные окна будем вот так.

```
wave = signal.make_wave(duration)
wave.window(np.bartlett(len(wave)))
spectrum = wave.make_spectrum()
spectrum.plot(high=880)
```

```
decorate(xlabel='Frequency (Hz)')

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

wave = signal.make_wave(duration)
wave.window(np.hanning(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='orange')
decorate(xlabel='Frequency (Hz)')
```

Листинг 1.1: Проверка окон

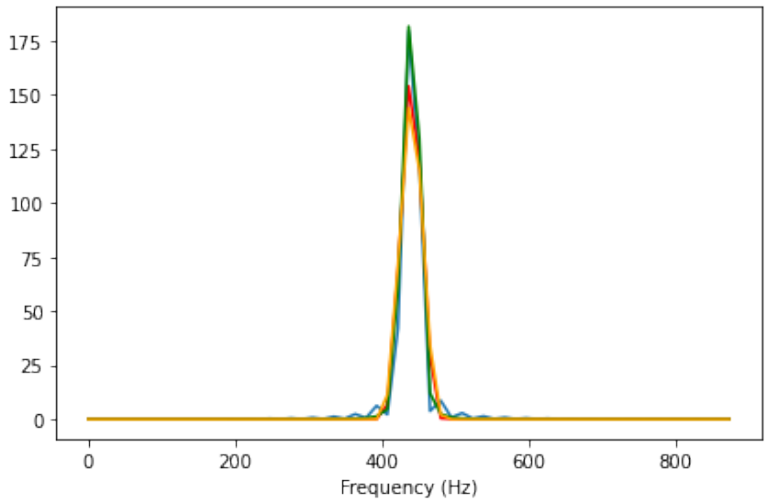

Рис. 1.2: Все вместе

Глава 2

Делаем SawtoothChirp

2.1 Основной код

В задании просят за основу взять Chirp.

```
class MySawtoothChirp(Chirp):
    def evaluate(self, ts):
        freqs = np.linspace(self.start, self.end, len(ts) -
    1)

        dts = np.diff(ts)
        dphis = PI2 * freqs * dts

        phases = np.cumsum(dphis)
        phases = np.insert(phases, 0, 0)

        cycles = phases / PI2
        frac, _ = np.modf(cycles)

        ys = self.amp * frac
        return ys
```

Листинг 2.1: Реализация своего пилообразного Chirp'а

Деление на PI2 я сделал по сути «для красоты», чтобы период был как бы 2π.

2.2 Проверка

Проверим сегменты в секундном Wave'е (начало и конец).

```
test_saw = MySawtoothChirp(start=220, end=440)
test_wave = test_saw.make_wave(duration=1, framerate=11025)
test_wave.segment(start=0, duration=0.02).plot()
```

```
decorate(xlabel='Time (s)')
```

Листинг 2.2: Начало

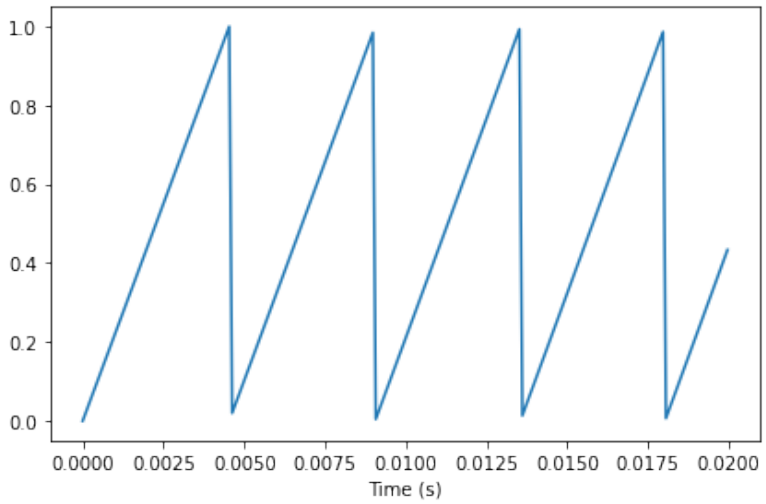

Рис. 2.1: Начало

```
test_wave.segment(start=1-0.02, duration=0.02).plot()
decorate(xlabel='Time (s)')
```

Листинг 2.3: Конец

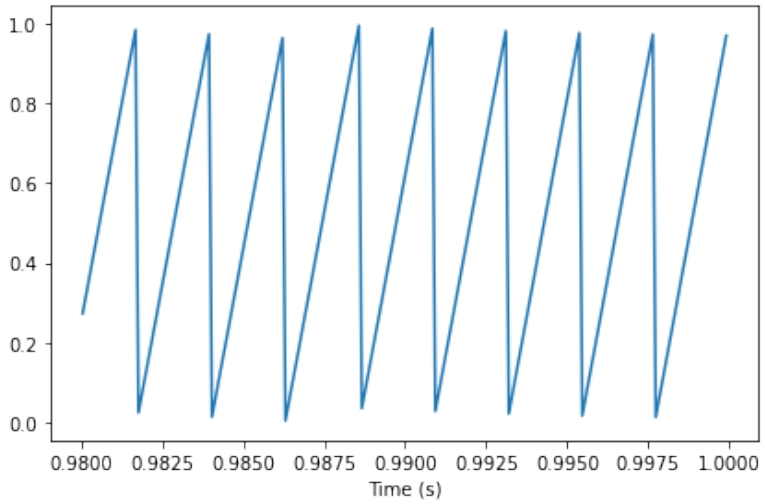

Рис. 2.2: Конец

Глава 3

Строим специфический `SawtoothChirp`

Сделаем ровно так, как нас просят в задании.

```
signal = MySawtoothChirp(start=2500, end=3000)
wave = signal.make_wave(duration=1, framerate=20_000)
```

Листинг 3.1: Создаем сигнал и ...

Предлагаю сразу кое-что проверить.

```
wave.segment(start=0.9, duration=0.02).plot()
decorate(xlabel='Time')
```

Листинг 3.2: Проверяем его

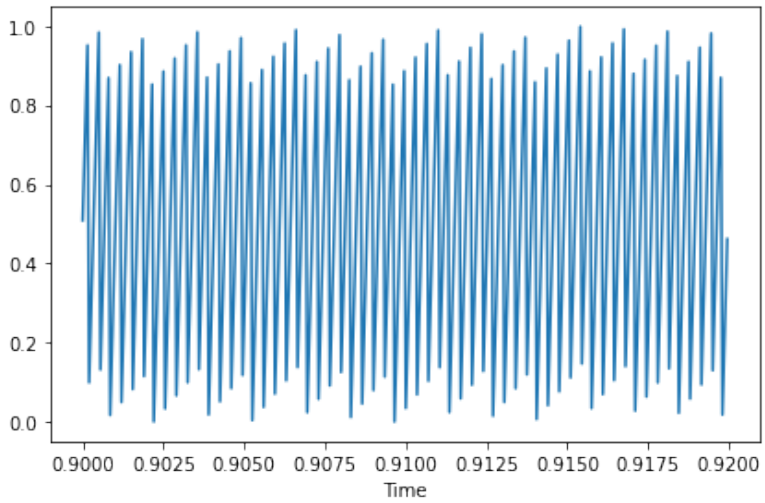

Рис. 3.1: Конец интервала

Кажется, будто бы наш сигнал начинает «скакать» вверх и вниз. Это можно исправить, увеличив `framerate`, но оставим пока так.

Посмотрим, на спектр.

```
spectrum = wave.make_spectrum()
spectrum.plot()
decorate(xlabel='Frequency')
```

Листинг 3.3: Спектр

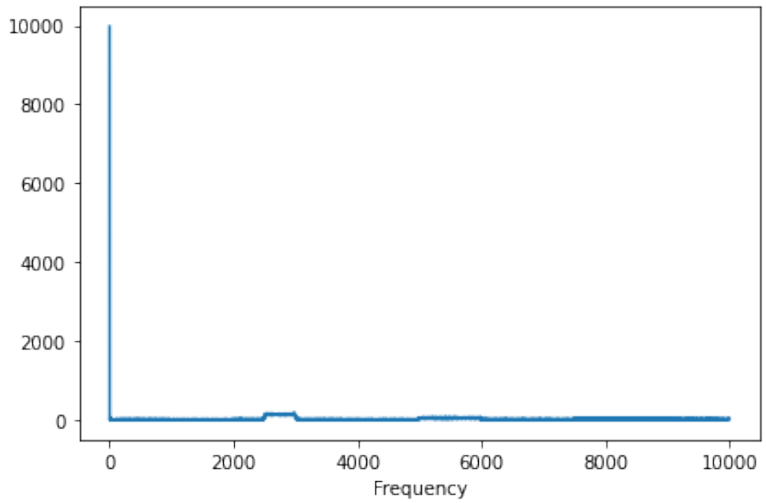

Рис. 3.2: Спектр

И уберем частоту в самом начале, чтобы лучше видеть форму.

```
spectrum = wave.make_spectrum()
spectrum.high_pass(10)
spectrum.plot()
decorate(xlabel='Frequency')
```

Листинг 3.4: Еще спектр

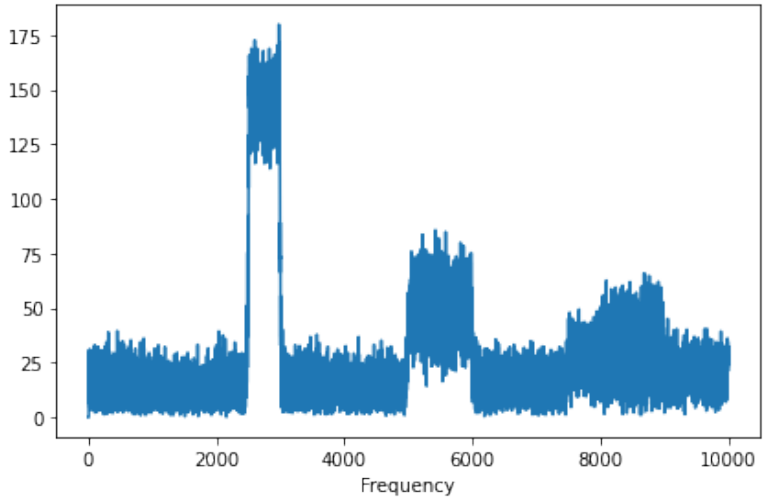

Рис. 3.3: Еще спектр

Похоже, тут есть некоторая уменбшающаяся величина, но при этом все как будто слишком «зашумленно». Вот, что получается с `framerate`, увеличенным в 16 раз.

```
signal16 = MySawtoothChirp(start=2500, end=3000)
wave16 = signal.make_wave(duration=1, framerate=20_000*16)
wave16.segment(start=0.9, duration=0.02).plot()
decorate(xlabel='Time')
```

Листинг 3.5: Новый сигнал

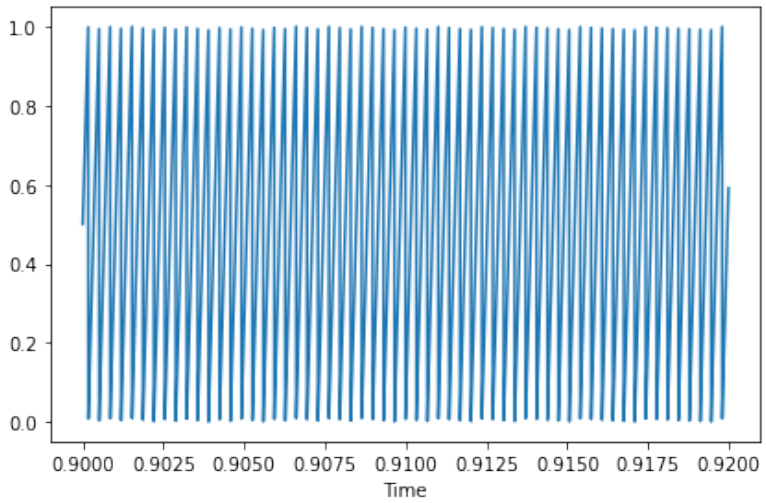

Рис. 3.4: Еще спектр

```
spectrum16 = wave16.make_spectrum()
spectrum16.high_pass(10)
spectrum16.plot()
decorate(xlabel='Frequency')
```

Листинг 3.6: Новый спектр

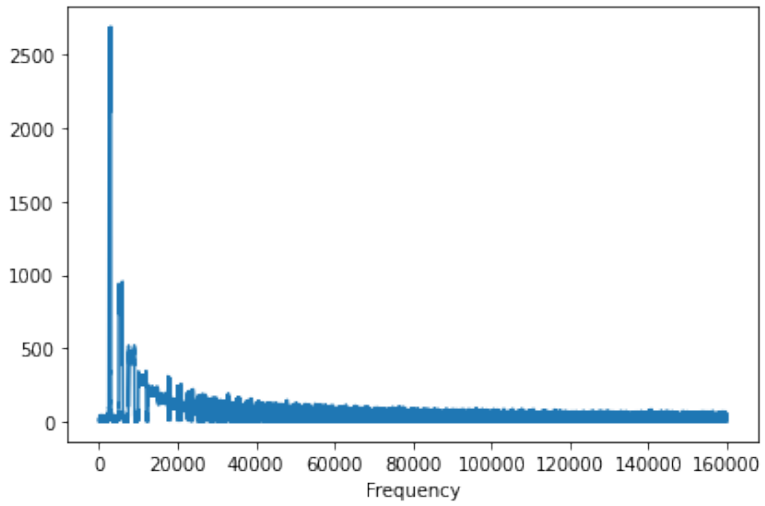

Рис. 3.5: Еще спектр

Теперь уже можно примерно видеть, к какой кривой стремится график.

Глава 4

Glissando

Начало возьму из «Rhapsody in Blue» (George Gershwin), как это советуют в ThinkDSP.

```
from thinkdsp import read_wave
wave = read_wave('Sounds/rhapblue11924.wav')
segment = wave.segment(start=1.35, duration=1.8-1.35)
segment.plot()
```

Листинг 4.1: Загрузка

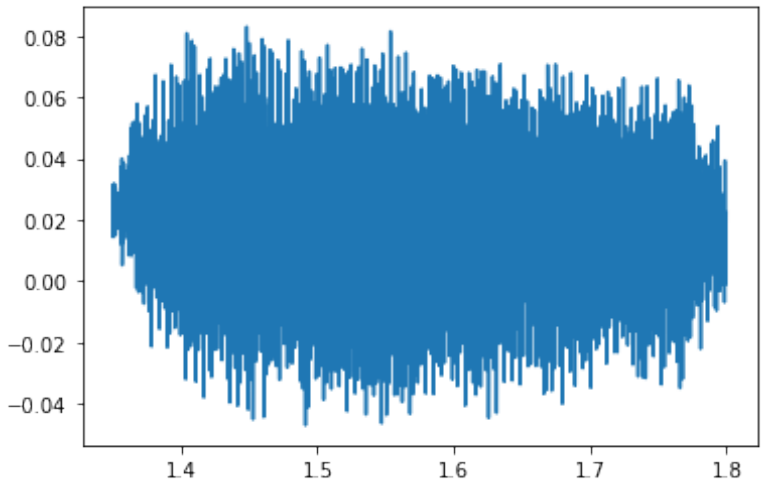

Рис. 4.1: Визуалиация

```
spectrum = segment.make_spectrum()
spectrum.plot()
```

Листинг 4.2: Спектр

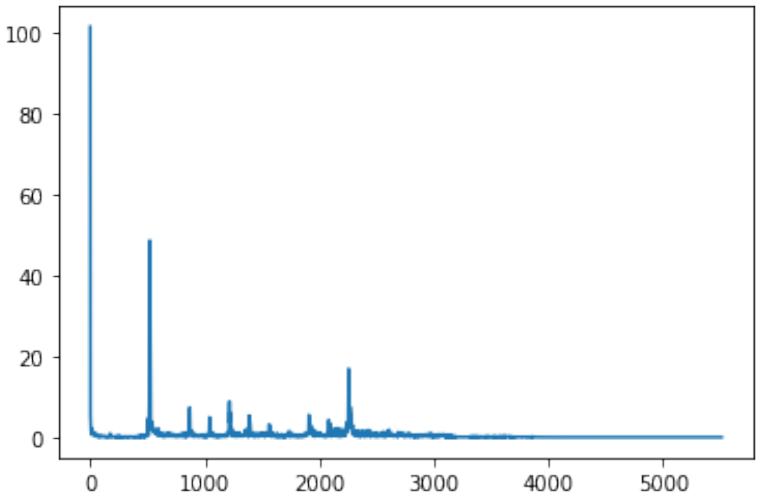

Рис. 4.2: Спектр

Глава 5

Делаем `TromboneGliss`

5.1 Рассуждения

Если X - некоторая частота (нота), а l_X - величина, означающаяся, насколько выдвинута кулиса, то по условию:

$$X \sim \frac{1}{l_X}$$
$$X = k \cdot \frac{1}{l_X}$$

Тогда C_3 (262Hz) и F_3 (349Hz) соответствуют некие l_{C_3} и l_{F_3} (причем, $l_{C_3} > l_{F_3}$).

Так как:

$$\begin{cases} C_3 = k \cdot \dfrac{1}{l_{C_3}} \\ F_3 = k \cdot \dfrac{1}{l_{F_3}} \end{cases} \Rightarrow F_3 \cdot l_{F_3} = C_3 \cdot l_{C_3} \Rightarrow X = \frac{F_3 \cdot l_{F_3}}{l_X}$$

Или:

$$
\begin{aligned}
X &= \frac{F_3 \cdot l_{F_3}}{l_{F_3} + \delta l} \\
&= \frac{F_3 \cdot l_{F_3}}{l_{F_3} + t(l_{C_3} - l_{F_3})} \\
&= \frac{F_3}{1 + t\left(\frac{l_{C_3} - l_{F_3}}{l_{F_3}}\right)} \\
&= \frac{F_3}{1 + t\left(\frac{l_{C_3}}{l_{F_3}} - 1\right)} \\
&= \frac{F_3}{1 + t\left(\frac{F_3}{C_3} - 1\right)}
\end{aligned}
$$

Где $t \in [0, 1]$ - некоторая величина, принимающая значение 0 на концах периода, а 1 - в его середине и меняющаяся линейно.

Если l_X меняется линейно, то X будет меняться обратно пропорционально квадрату l_X:

$$
\frac{dX}{dl_X} = -\frac{k}{l_X^2}
$$

5.2 Реализация

```
class MyTromboneGliss(Chirp):
    def _get_frequency(self, ts):
        S, E = self.start, self.end

        # ts are normalized to [0,1]
        return E / (1 + ts * (E/S - 1))

    def evaluate(self, ts):
        l_C3, l_F3 = 2, 1

        freqs = self._get_frequency(1 - 2 * np.abs(ts[:-1] - 0.5))

        dts = np.diff(ts)
        dphis = PI2 * freqs * dts

        phases = np.cumsum(dphis)
```

```
17          phases = np.insert(phases, 0, 0)
18
19          ys = self.amp * np.cos(phases)
20          return ys
```

Листинг 5.1: Код `MyTromboneChirp`

Создадим сигнал.

```
signal = MyTromboneGliss(start=262, end=349)
wave = signal.make_wave(duration=1, framerate=11025)
```

Листинг 5.2: Искомый сигнал

В `.ipynb`-файле я, конечно, посмотрел на разные участки сигнала, но с данными частотами сложно разглядеть различие. Вот если брать 1000 и 100, то там будет видно, что она меняется. Ввиду незаметности для глаза, не буду вставлять сюда иллюстрации.

Посмотрим на спектрограмму.

```
wave.make_spectrogram(512).plot(high=600)
```

Листинг 5.3: Спектрограмма

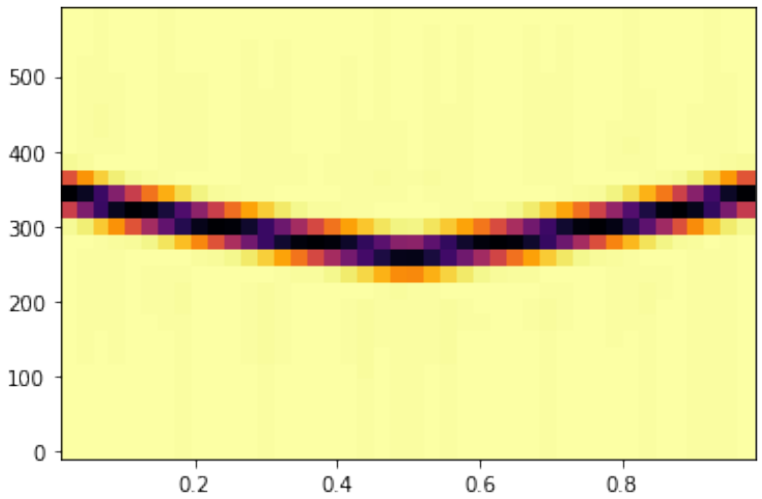

Рис. 5.1: Спектрограмма

На глаз она кажется кусочно-линейной, но из рассуждений выше следует, что это лишь иллюзия.

Глава 6

Гласные

Скачаем какие-нибудь звуки гласных из сети и посмотрим на них.

```
wave = read_wave(
    'Sounds/523055__cbelloso__vocales-vowels-man-woman-kid-girl.wav'
)
segment = wave.segment(start=0.2, duration=3.9)
segment.plot()
```

Листинг 6.1: Выбираем участок

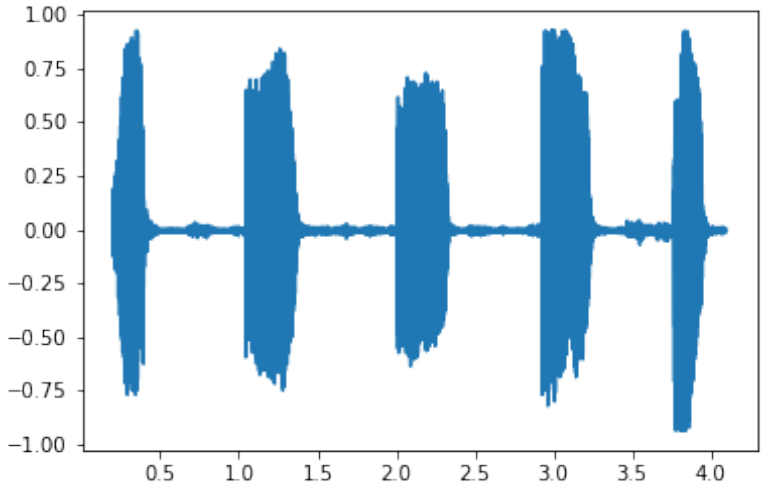

Рис. 6.1: Участок

```
segment.make_spectrogram(512).plot()
```

Листинг 6.2: Спектрограмма

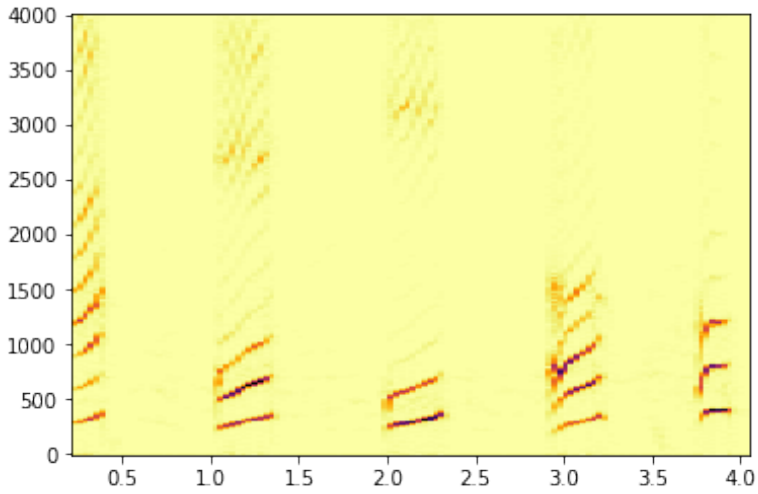

Рис. 6.2: Спектрограмма

Участки не горизонтальны потому, что так звуки произноятся на записи. Нам важно то, что некоторые из участков темнее, а некоторые светлее (в рамках одного столбца), что соответствует спектрам соответствующих гласных.

В `.ipynb`-файле этой лабораторной работы можно посмотреть на сами спектры гласных. Я не стал вставлять их сюда, потому что это утомительно.