Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Отчёт по лабораторной работе №1
Дисциплина: Телекоммуникационные технологии
Тема: Звуки и сигналы

Работу выполнил:
Ляшенко В.В.
Группа: 3530901/80201
Преподаватель:
Богач Н.В.

Санкт-Петербург
2021

Оглавление

Список иллюстраций

Листинги

Глава 1

Упражнение 1.1

Создадим копию репозитория ThinkDSP и клонируем его на компьютер. Затем установим дистрибутив Anaconda, который содержит Jupyter Notebook. Теперь можно перейти к выполнению упражнений.

В первом упражнении нам требуется для Jupyter загрузить `chap01.ipynb`, прочитать пояснения и запустить примеры.

Все примеры были успешно запущены. В последнем примере при выставлении более низких значений `cutoff` звук получался более приглушённым (Рис.1.1).

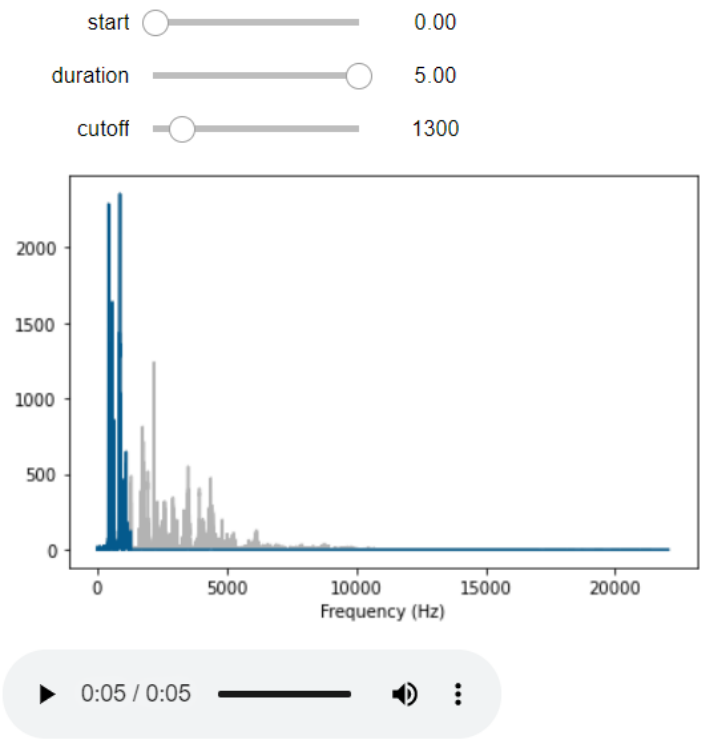

Рис. 1.1: Использование интерактивных виджетов IPython

Глава 2

Упражнение 1.2

2.1 Сегмент

Скачаем с сайта `https://freesound.org` образец звука, имеющий чётко выраженную высоту. С помощью метода `read_wave` прочтём запись.

```
from thinkdsp import read_wave
wave = read_wave('sounds/564358__voxlab__chillout-vox-pad-chord-gb6.wav')
wave.make_audio()
```

Листинг 2.1: Чтение записи

Из этой записи выделим полусекундный сегмент, в котором высота постоянна (Рис.2.1).

```
from thinkdsp import decorate
segment = wave.segment(start=6, duration=0.5)
segment.plot()
decorate(xlabel='Time (s)')
```

Листинг 2.2: Выделение сегмента

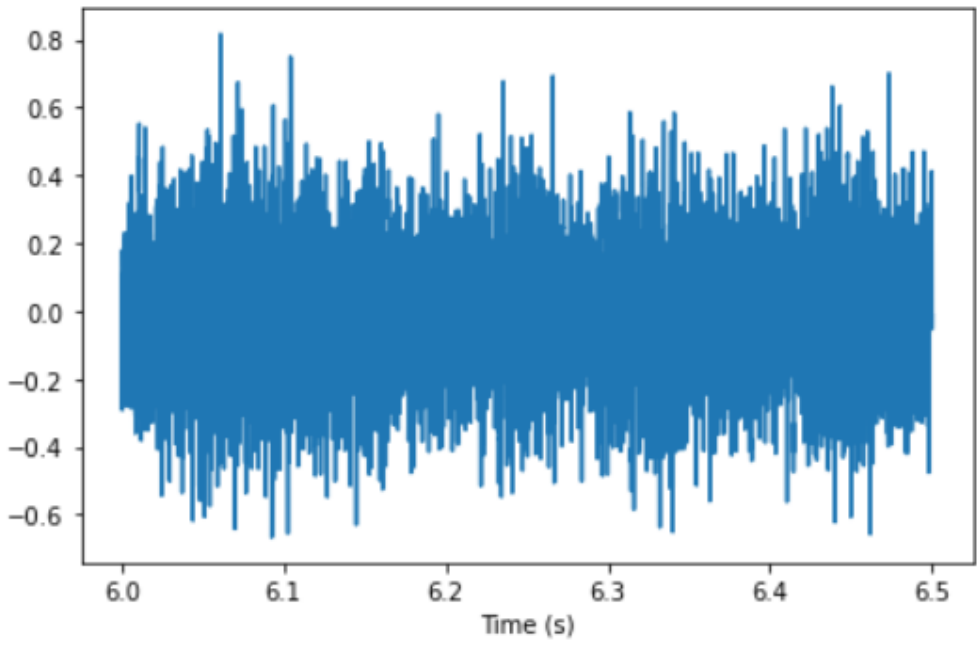

Рис. 2.1: Сегмент

2.2 Спектр

После этого вычислим и распечатаем спектр выбранного сегмента (Рис.2.2).

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

Листинг 2.3: Вычисление спектра

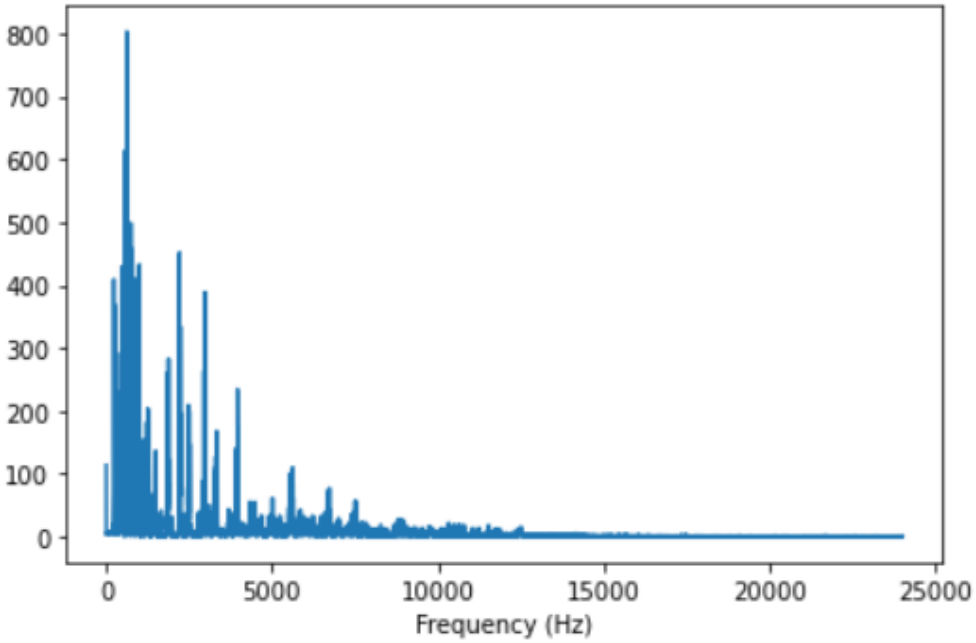

Рис. 2.2: Спектр

Тембр - это характеристика, определяющая восприятие качества звука. Тембр звука зависит от частот в спектре и их интенсивностей. Т.е. чем больше различных частот в спектре, тем богаче и насыщинее будет тембр.

2.3 Фильтрация

Используем `high_pass`, `low_pass` и `band_stop` для фильтрации гармоник.

```
spectrum.high_pass(2000)
spectrum.low_pass(10000)
spectrum.band_stop(2050, 2950)
spectrum.band_stop(3000, 5000)
spectrum.band_stop(5050, 7950)
spectrum.band_stop(8050, 9000)
spectrum.plot(high=20000)
decorate(xlabel='Frequency (Hz)')
```

Листинг 2.4: Выполнение фильтрации

На рис.2.3. видим результат фильтрации гармоник, полученный с помощью предложенных методов.

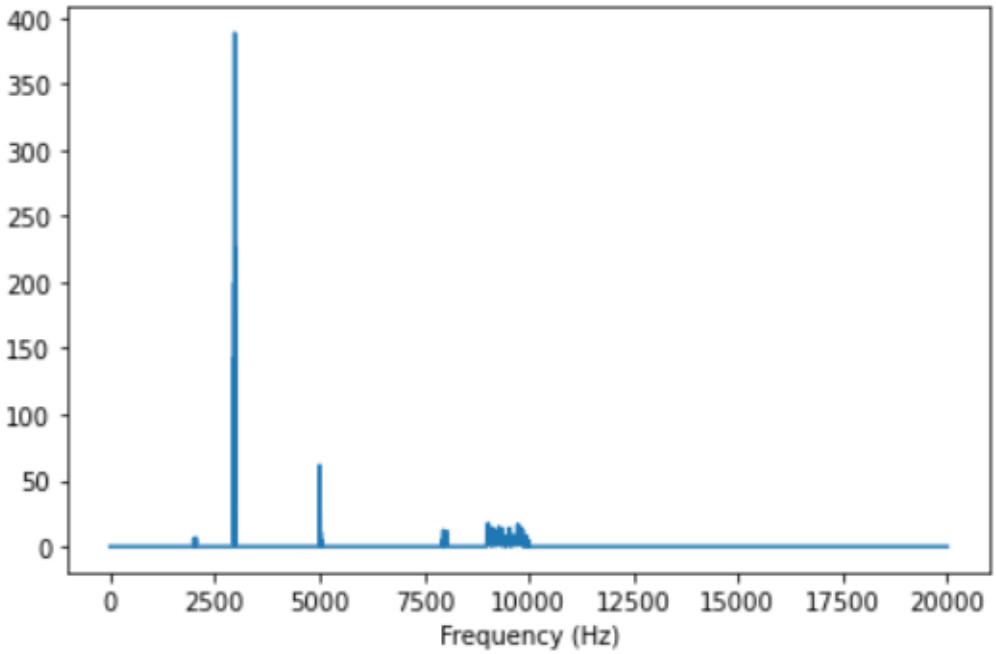

Рис. 2.3: Результат фильтрации

Преобразуем спектр обратно в сигнал и послушаем его.

```
spectrum.make_wave().make_audio()
```

Листинг 2.5: Преобразование спектра в сигнал

В результате фильтрации звук получился более тонким и звенящим.

Глава 3

Упражнение 1.3

3.1 Сложный сигнал

Теперь создадим сложный сигнал из объектов `SinSignal` и `CosSignal`, суммируя их. Полученный сигнал представлен на рис.3.1. Затем обработаем этот сигнал для получения `wave` и прослушаем его.

```
from thinkdsp import CosSignal, SinSignal

cos_sig1 = CosSignal(freq=250, amp=0.7, offset=0)
sin_sig1 = SinSignal(freq=600, amp=0.4, offset=0)
cos_sig2 = CosSignal(freq=350, amp=1.0, offset=0)
sin_sig2 = SinSignal(freq=800, amp=0.8, offset=0)

sum_sig = cos_sig1 + sin_sig1 + cos_sig2 + sin_sig2
sum_sig.plot()
decorate(xlabel='Time (s)')
sum_sig.make_wave().make_audio()
```

Листинг 3.1: Создание сложного сигнала

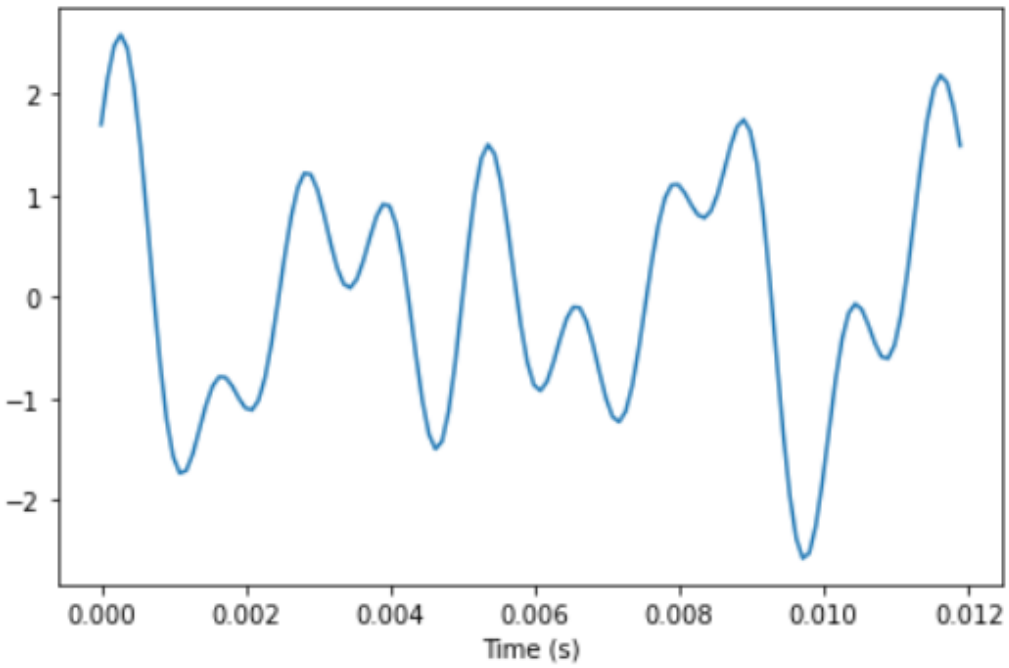

Рис. 3.1: Сложный сигнал

Получившийся сигнал по звучанию похож на телефонный гудок.

3.2 Спектр сложного сигнала

Вычислим спектр полученного сигнала и выведем его (Рис.3.2).

```
spectrum = sum_sig.make_wave().make_spectrum()
spectrum.plot(1000)
decorate(xlabel='Frequency (Hz)')
```

Листинг 3.2: Вычисление спектра сложного сигнала

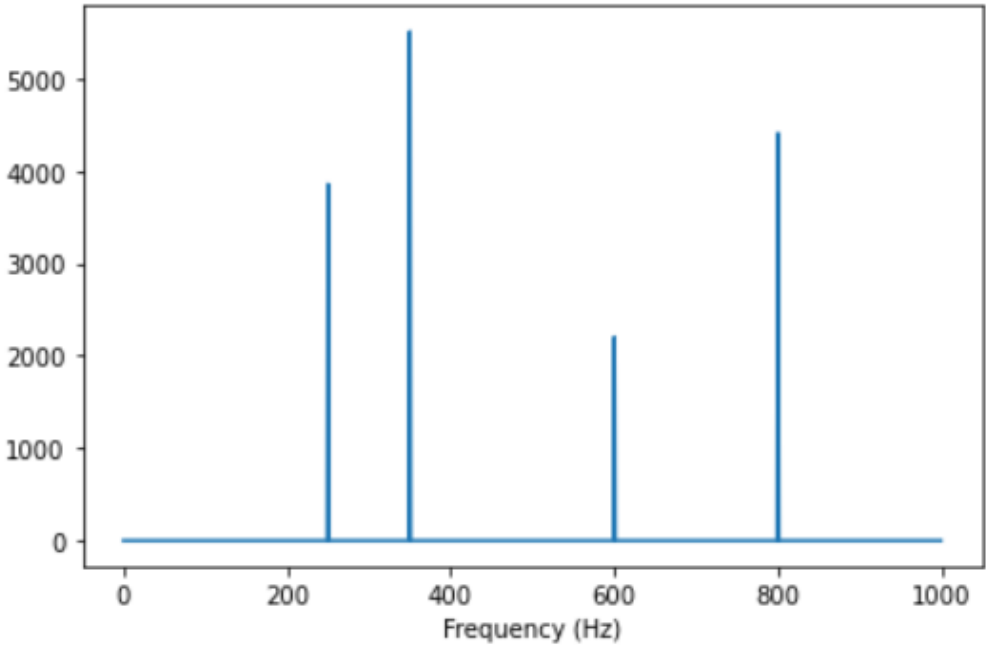

Рис. 3.2: Спектр сложного сигнала

3.3 Некратные компоненты

Добавим частотные компоненты, не кратные основным.

```
cos_sig3 = CosSignal(freq=3397, amp=0.6, offset=0)
sin_sig3 = SinSignal(freq=2221, amp=0.9, offset=0)
sum_sig = sum_sig + sin_sig3 + cos_sig3
sum_sig.plot()
decorate(xlabel='Time (s)')
sum_sig.make_wave().make_audio()
```

Листинг 3.3: Добавление новых компонентов

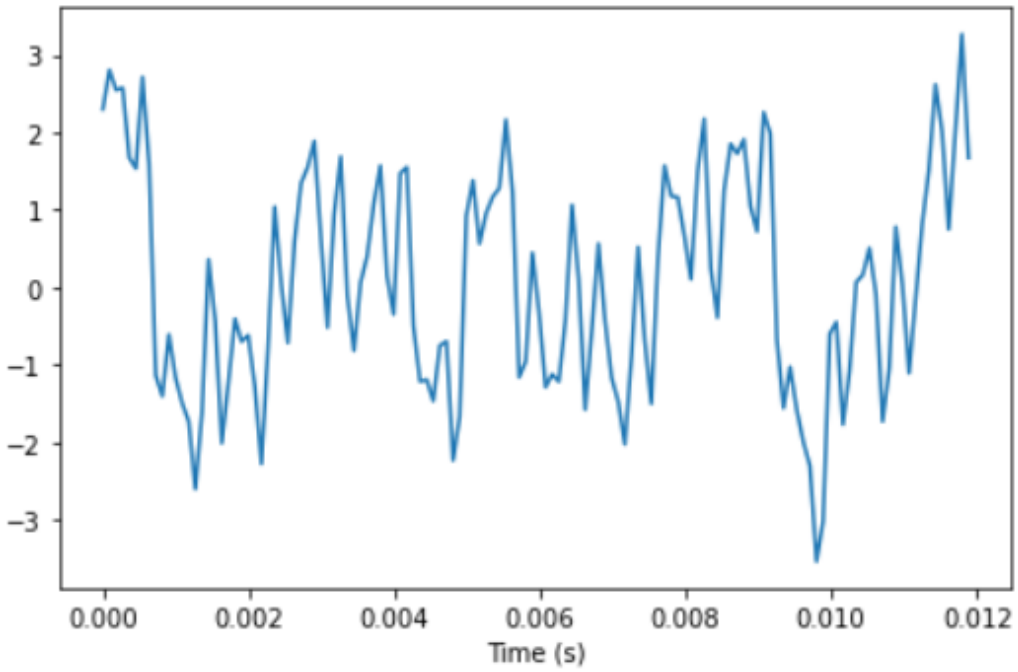

Рис. 3.3: Сложный сигнал с добавленными компонентами

Получим спектр нового сигнала (Рис.3.4).

```
spectrum = sum_sig.make_wave().make_spectrum()
spectrum.plot(4000)
decorate(xlabel='Frequency (Hz)')
```

Листинг 3.4: Вычисление спектра нового сигнала

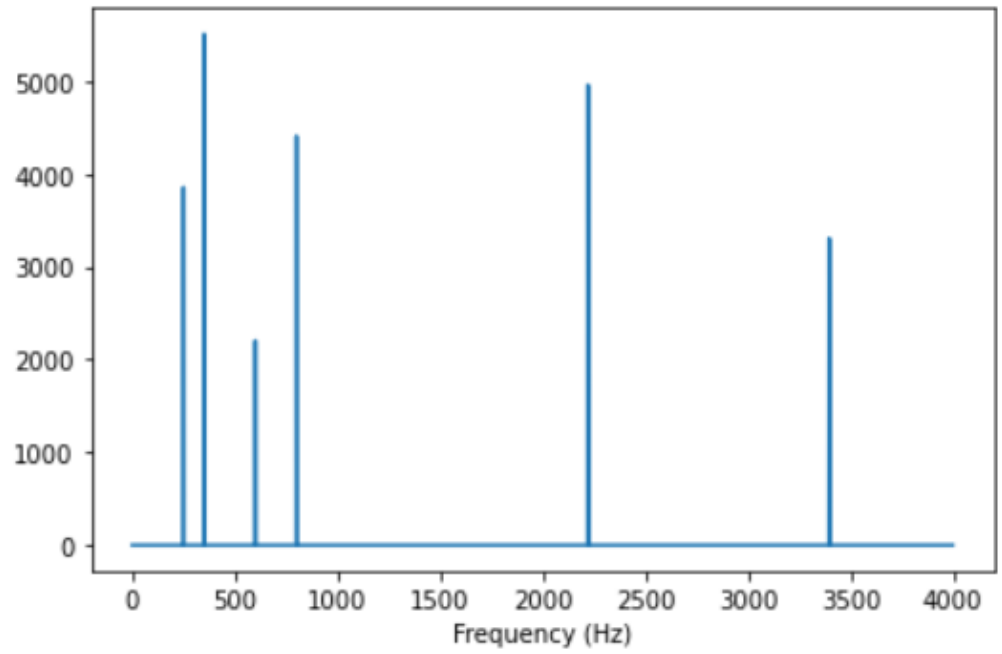

Рис. 3.4: Спектр нового сигнала

Получившийся звук стал более пищащим и неприятным.

Глава 4

Упражнение 1.4

Напишем функцию `stretch`, которая будет ускорять или замедлять сигнал изменением `ts` и `framerate`.

```
def stretch(wave, factor):
wave.ts *= factor
wave.framerate /= factor
```

Листинг 4.1: Функция stretch

Возьмём звук из Упражнения 1.2, создадим на его основе два сигнала и один из них замедлим, а другой ускорим (Рис.4.1-4.2).

```
wave1 = read_wave('sounds/564358__voxlab__chillout-vox-pad-chord-gb6.wav')
wave2 = read_wave('sounds/564358__voxlab__chillout-vox-pad-chord-gb6.wav')
stretch(wave1, 2.0)
wave1.plot()
wave1.make_audio()
```

Листинг 4.2: Замедление сигнала

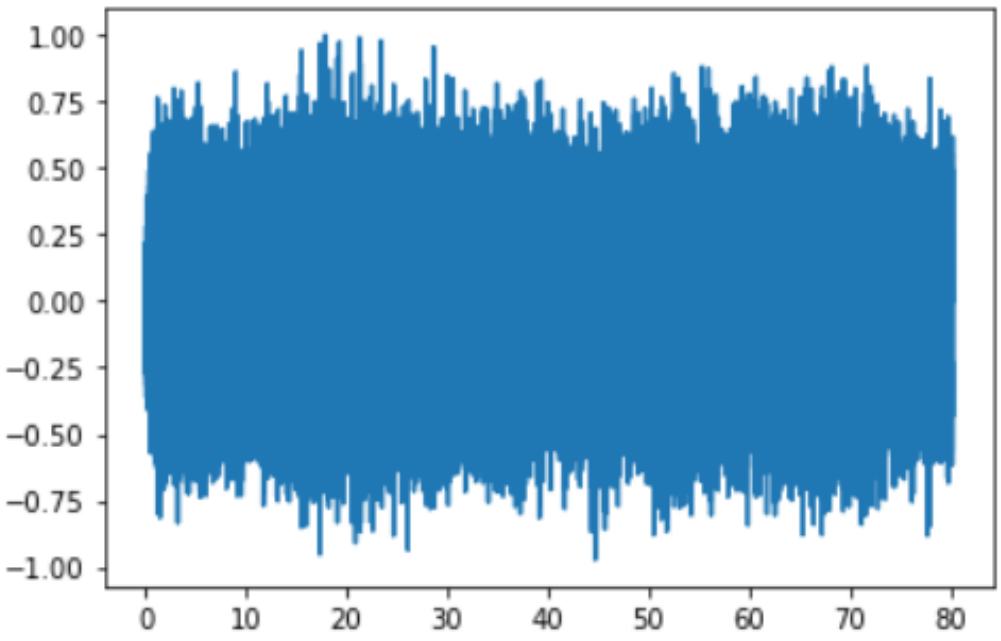

Рис. 4.1: Замедленный сигнал

```
stretch(wave2, 0.2)
wave2.plot()
wave2.make_audio()
```

Листинг 4.3: Ускорение сигнала

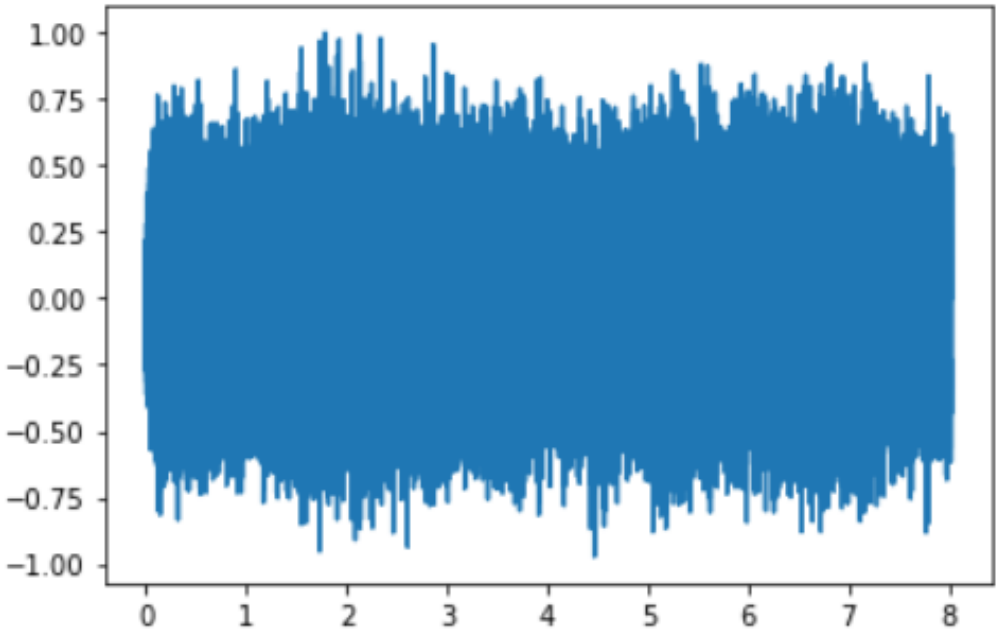

Рис. 4.2: Ускоренный сигнал

В первом случае звук стал более тяжёлым, он словно слышен через толщу воды. Во втором же случае звук наоборт стал писклявым и звенящим.

Глава 5

Выводы

В результате выполнения данной работы мы познакомились с понятиями сигнала и звука, а также понятиями описывающими их. Кроме того, мы научились работать с сигналами, спектрами и звуковыми волнами, применяя различные функции, написанные на языке Python.