Лабораторная работа № 1. Звуки и сигналы.

3530901/80201, Шелаев Н. Р.

30 мая 2021 г.

Оглавление

Список иллюстраций

Листинги

Глава 1

Периодические сигналы

Строим периодические сигналы.

```
from thinkdsp import CosSignal, SinSignal
cos_sig = CosSignal(freq=440, amp=1.0, offset=0)
sin_sig = SinSignal(freq=880, amp=0.5, offset=0)
mix = sin_sig + cos_sig

```

Листинг 1.1: Построение сигнала `Cos` и `Sin` и их суммы

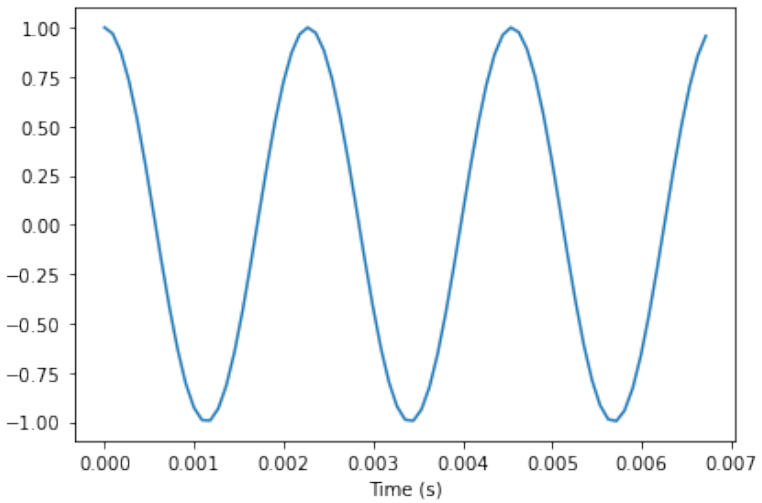

Рис. 1.1: Косинусоида

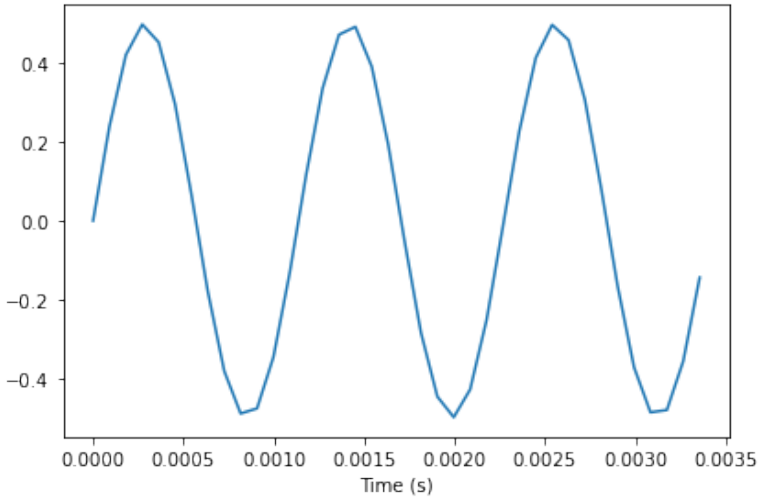

Рис. 1.2: Синусоида

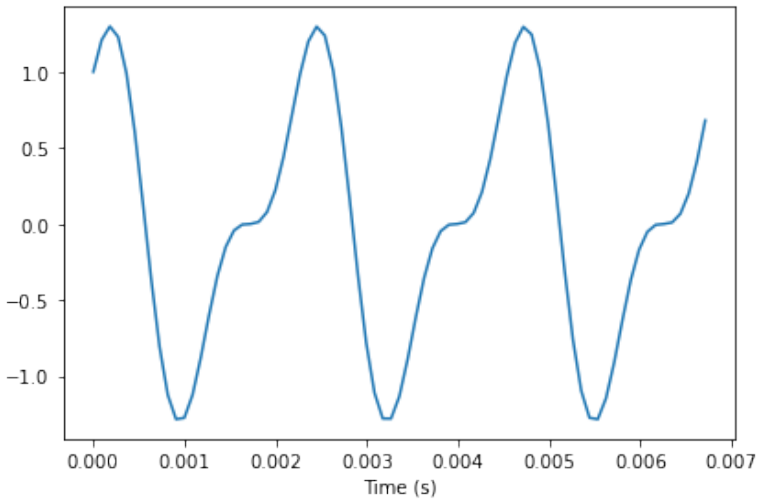

Рис. 1.3: Сумма двух сигналов

Глава 2

Получение звука

Знакомимся с некоторыми функциями предоставленного нам класса `wave` для получения сигналов и их первичной обработки.

```
1         wave = mix.make_wave(duration = 0.5, start = 0,
    framerate = 11025)
2         period = mix.period
3         segment = wave.segment(start=0, duration=period * 3)
4         segment.plot()
5         wave.normalize()
6         wave.apodize()
7         wave.plot()
8
```

Листинг 2.1: Получение сигнала

Результаты для суммы `Cos` и `Sin`:

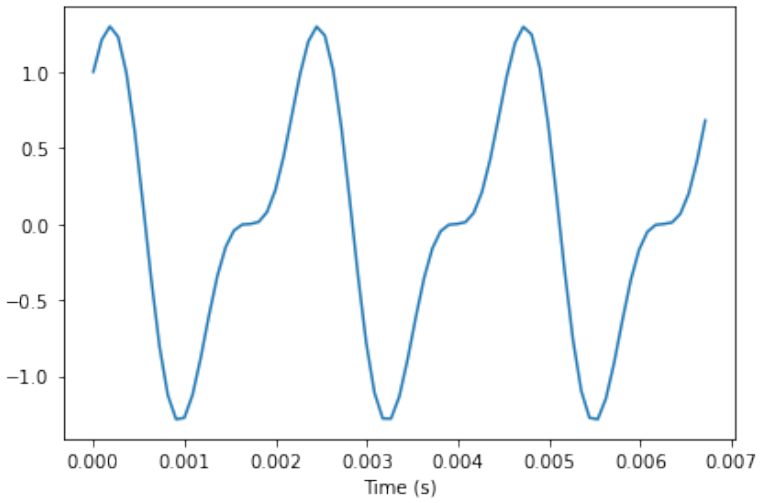

Рис. 2.1: Небольшой сегмент сигнала (3 периода)

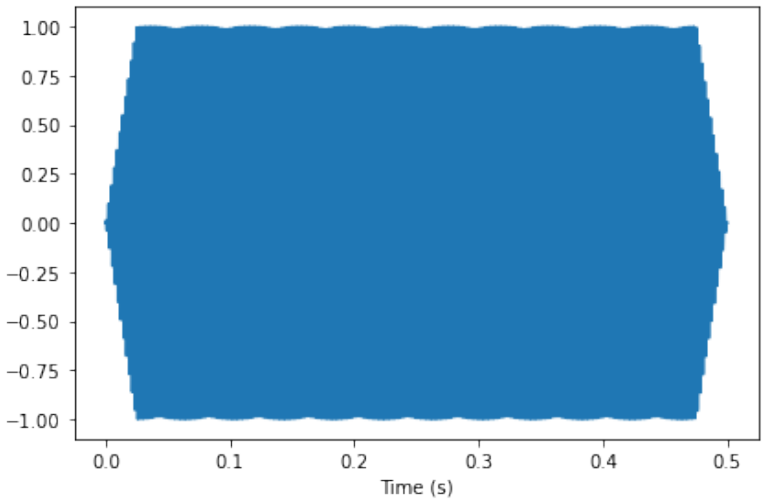

Рис. 2.2: Нормализованный сигнал

Также мы можем сигнал из WAV файлов.

```
1       from thinkdsp import read_wave
2       wave = read_wave('92002__jcveliz__violin-origional.
    wav')
3       wave.make_audio()
4       segment = wave.segment(start = 1.2, duration = 0.6)
5       segment.plot()
```

6

Листинг 2.2: Получение сигнала

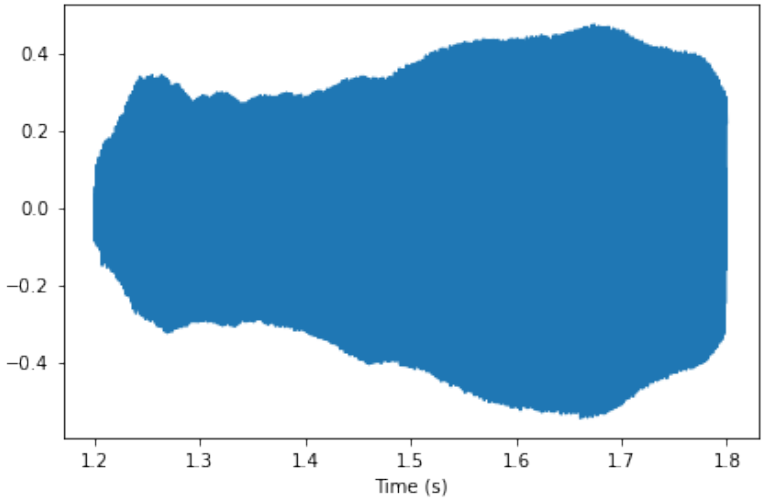

Рис. 2.3: Сегмент сигнала из WAV файла

Глава 3

Спектр сигнала

Строим спектр сигнал.

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

Листинг 3.1: Спектр сигнала

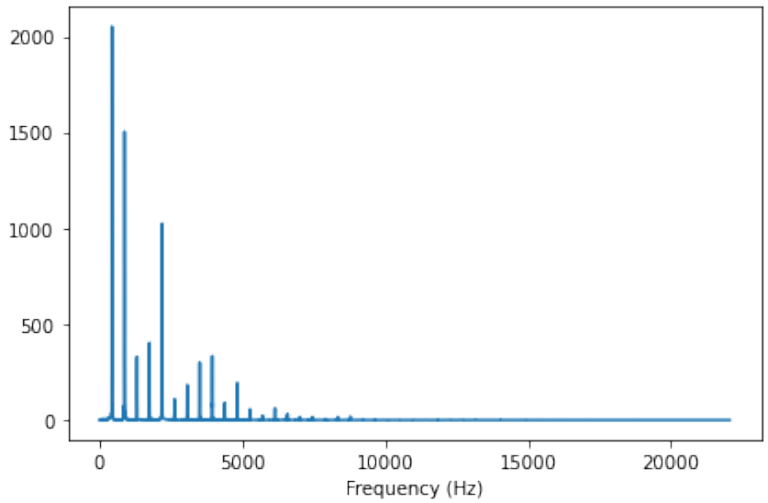

Рис. 3.1: Полученный спектр сигнала

Частотные составляющие выше 10 кГц достаточно малы, поэтому мы можем лучше рассмотреть нижние частоты.

```
spectrum.plot(high = 10000)
```

Листинг 3.2: Улучшаем масштаб

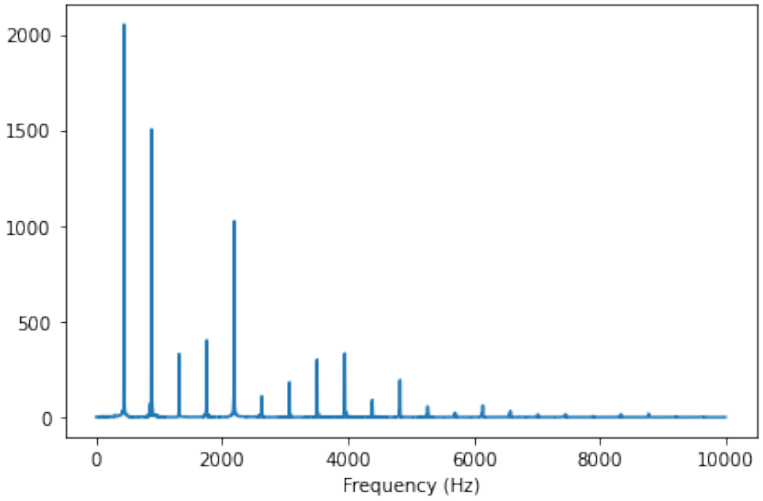

Рис. 3.2: Так уже лучше

Глава 4

Фильтрация

Исследуем работу различных фильтров.

```
spectrum.low_pass(3000)
spectrum.plot(high = 10000)
filtered = spectrum.make_wave()
filtered.normalize()
filtered.apodize()
filtered.plot()
segment.normalize()
segment.apodize()
segment.plot()

```

Листинг 4.1: Функция для фильтра нижних частот

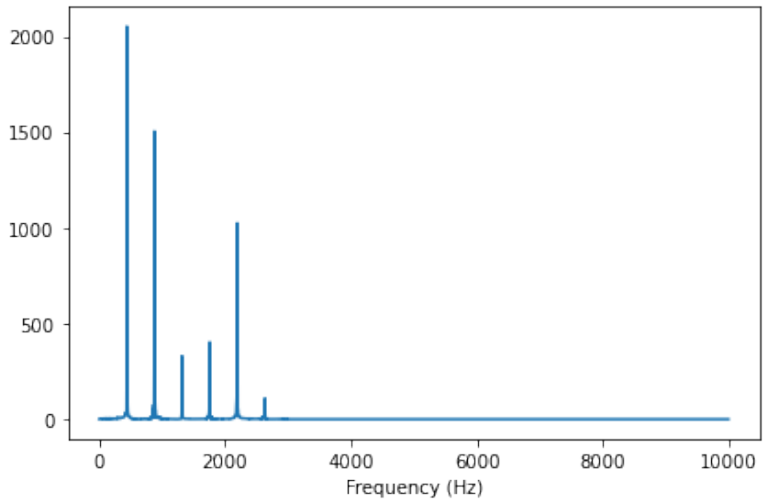

Рис. 4.1: Спектр сигнала с применением фильтра нижних частот

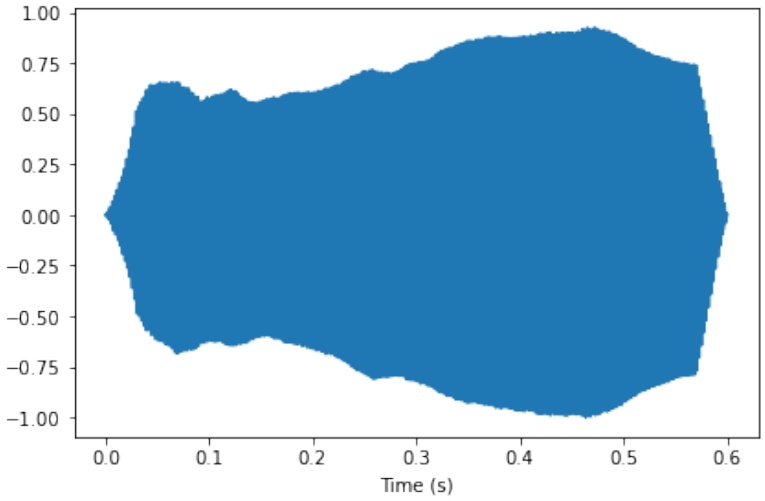

Рис. 4.2: Отфильтрованный сигнал

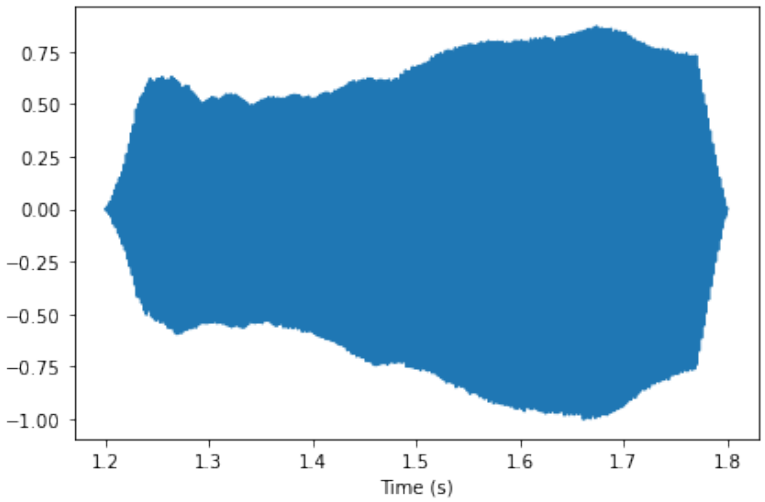

Рис. 4.3: Оригинальный сигнал

При их прослушивании отфильтрованная версия сигнала звучит более однотонно и с приглушенным качеством. Этот сигнал слышится как оригинальный, как если бы его воспроизводили по телефону или через стену.

Глава 5

Упражнения

5.1 Задание 2

Применяем различные фильтры для сложного сигнала.

```
wave = read_wave('170255__dublie__trumpet.wav')
wave.normalize()
wave.plot()

```

Листинг 5.1: Получение сложного сигнала

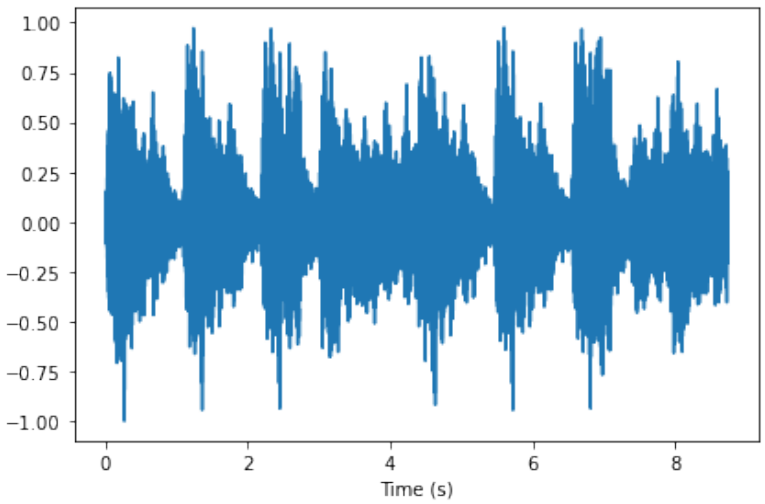

Рис. 5.1: Сложный сигнал

```
segment = wave.segment(start = 2.2, duration = 0.4)
segment.plot()
```

Листинг 5.2: Сегмент сигнала

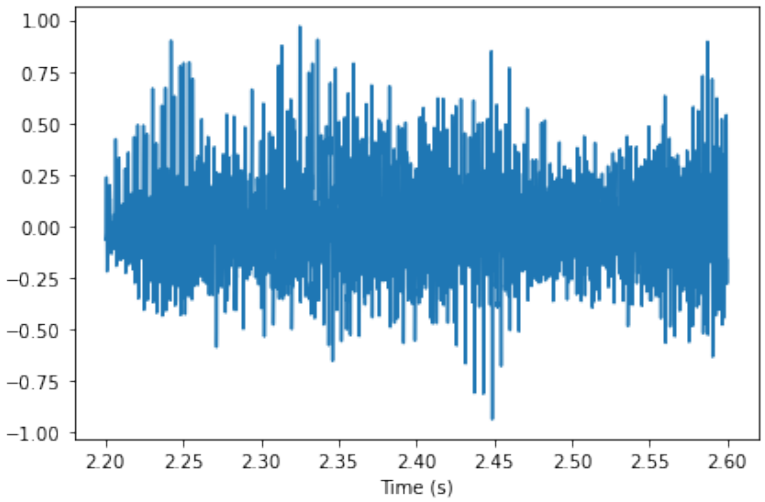

Рис. 5.2: Взяли небольшой сегмент сигнала

```
segment = wave.segment(start = 2.2, duration = 0.4)
segment.plot()
```

Листинг 5.3: Получение спектра сигнала

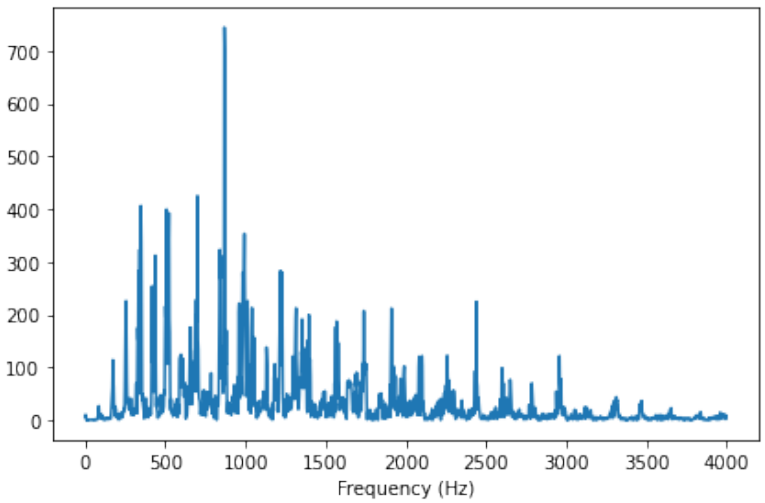

Рис. 5.3: Спектр сигнала

```
spectrum = segment.make_spectrum()
spectrum.plot(high = 1000)

```

Листинг 5.4: Улучшаем масштаб спектра

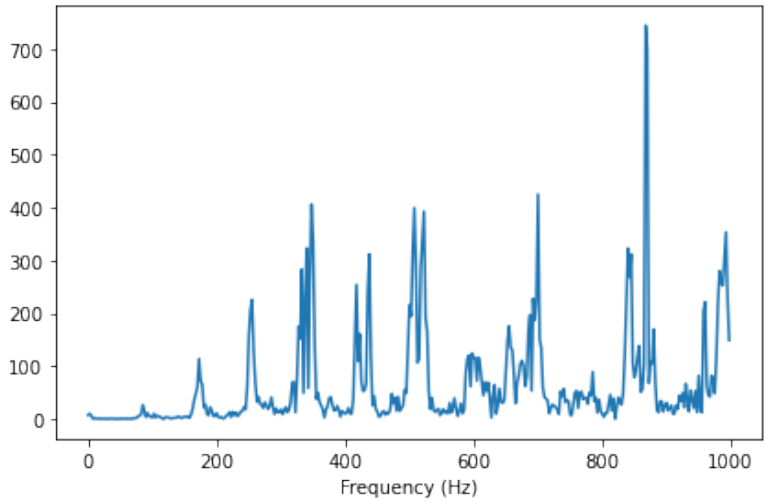

Рис. 5.4: Так лучше видны пики спектра

```
spectrum = segment.make_spectrum()
spectrum.plot(high = 100)

```

Листинг 5.5: Находим основную частоту сигнала (основной пик спектра)

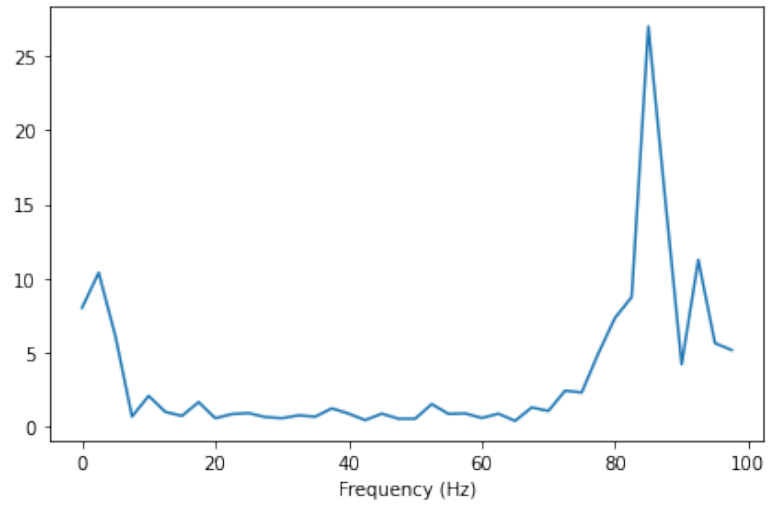

Рис. 5.5: Основная частота сигнала

Доминирующий пик находится на частоте 867,5 Гц. Основной пик составляет примерно 85,0 Гц с гармониками 170, 255, 340, 425, 510, ... Гц. Высота тона сигнала, которую мы воспринимаем, обычно является фундаментальной, даже если она не является доминантной.

```
        from ipywidgets import interact, fixed

        def filter_wave_low(wave, low, factor):
                segment = wave.segment(2.2, 0.4)
            spectrum = segment.make_spectrum()
                spectrum.plot(high = 2500, color = '0.7')
                spectrum.low_pass(low, factor)
                spectrum.plot(high = 2500, color = '#045a8d')
                decorate(xlabel = 'Frequency (Hz)')
                audio = spectrum.make_wave().make_audio()
                display(audio)

        interact(filter_wave_low, wave = fixed(wave), low =
    (0, 2500, 250), factor = (0, 1, 0.05));

```

Листинг 5.6: Функция фильтра НЧ

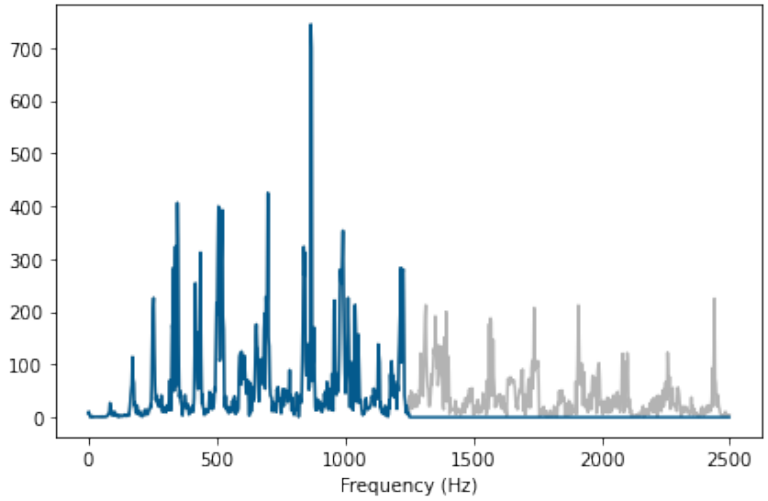

Рис. 5.6: Применение фильтра НЧ

Функции для фильтра верхних частот (ФВЧ) и полосо-заграждающего фильтра (ФПЗ) очень похожи на функцию для ФНЧ и поэтому здесь не приводятся.

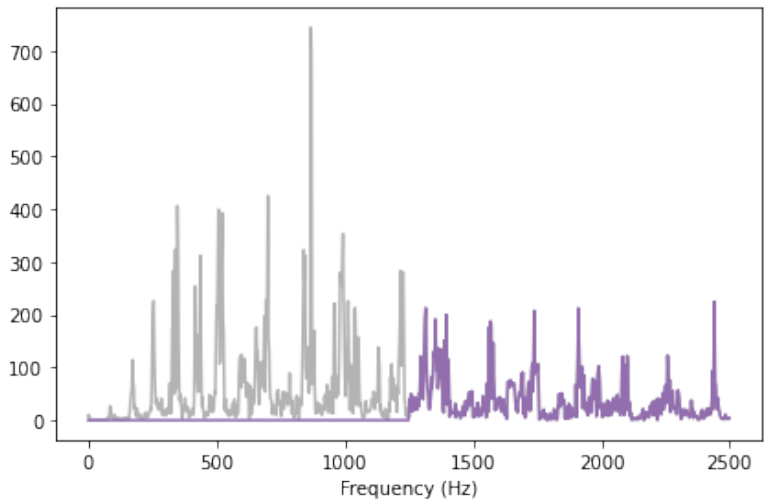

Рис. 5.7: Применение ФВЧ

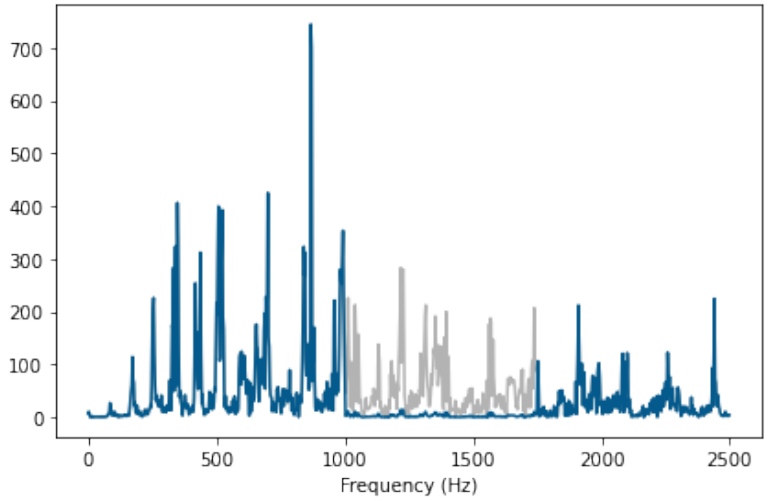

Рис. 5.8: Применение ФПЗ

5.2 Задание 3

Создаём сложный сигнал и исследуем его.

```
1         import math
2         signal = (SinSignal(freq = 600, amp = 0.4, offset =
    math.pi / 2) + CosSignal(freq = 200, amp = 0.6, offset =
    math.pi / 3) + SinSignal(freq = 800, amp = 0.8, offset =
    math.pi / 4) + CosSignal(freq = 400, amp = 1.0, offset =
    math.pi / 5))
3         signal.plot()
4
```

Листинг 5.7: Получение сложного сигнала через сумму других сигналов

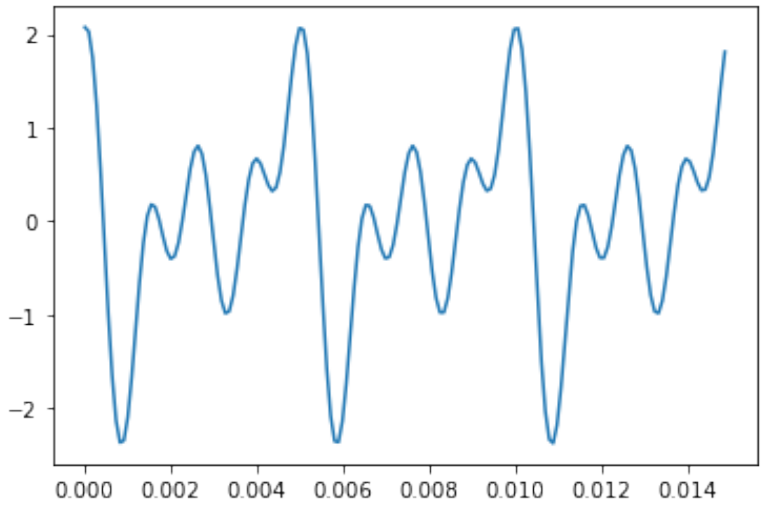

Рис. 5.9: Полученный сложный сигнал

```
wave = signal.make_wave(duration = 2.5)
wave.apodize()
spectrum = wave.make_spectrum()
spectrum.plot(high = 1000)

```

Листинг 5.8: Спектр этого сигнала

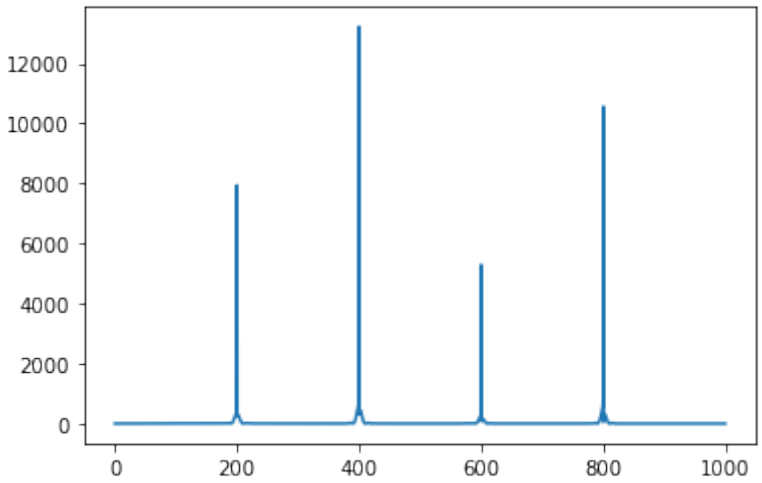

Рис. 5.10: Спектр сигнала с кратной частотой

Здесь использовались сигналы с частотой, кратной основным (что видно

на спектре). Теперь добавим частотные компоненты, не кратные основным.

```
signal += SinSignal(freq = 500, amp = 0.7)
wave = signal.make_wave(duration = 2.5)
wave.apodize()
spectrum = wave.make_spectrum()
spectrum.plot(high = 1000)

```

Листинг 5.9: Добавление некратной частоты

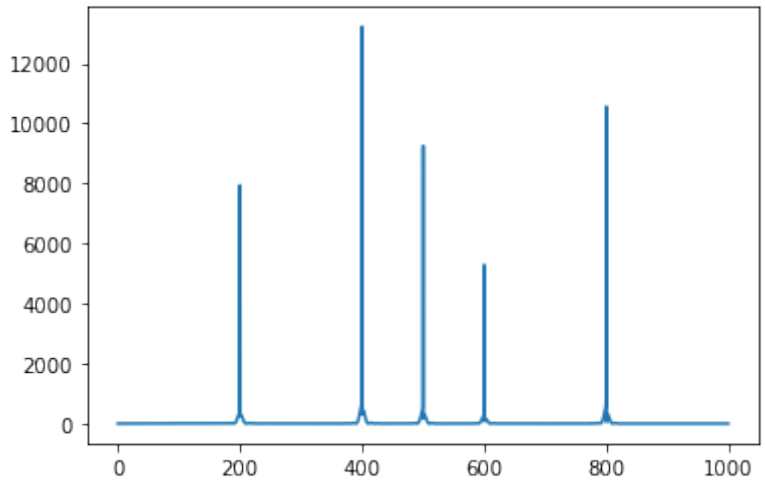

Рис. 5.11: Спектр с некратной частотой

Звук исходного сигнала изменился.

5.3 Задание 4

Функция `scretch` для ускорения и замедления сигнала.

```
wave = read_wave('170255__dublie__trumpet.wav')
wave.normalize()

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

stretch(wave, 2)
wave.plot()

```

Листинг 5.10: Функция для ускорения / замедления сигнала

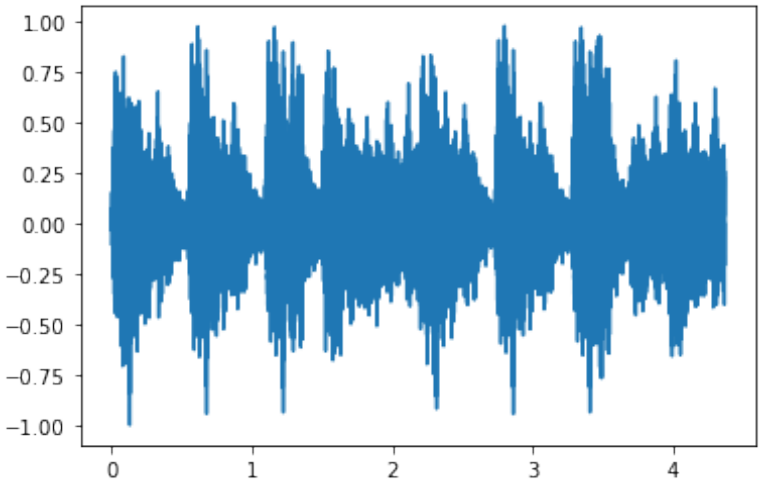

Рис. 5.12: Ускорение сигнала в 2 раза

Глава 6

Вывод

В данной работе мы познакомились с функциями для первичной обработки сигналов. Построили сложный сигнал с некратной частотой и нашли его спектр. Также провели исследование действия различных фильтров на сигнал.