Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа компьютерных технологий и информационных систем

Отчёт по лабораторным работам

Дисциплина: Телекоммуникационные технологии.

Выполнил студент гр. 5130901/10101 _____ Д.Л. Симоновский
(подпись)

Руководитель _____ Н.В. Богач
(подпись)

“30” октябрь 2024 г.

Санкт-Петербург
2024

Оглавление

1. Лабораторная работа 1. Сигналы и звуки.

1.1. Упражнение 1.2.

Скачаем с сайта https://freesound.org/ образец звука и различными способами исследуем его. Для удобной работы с сигналами здесь, и в дальнейших работах будем использовать библиотеку thinkdsp.

Откроем скачанный файл, нормализуем и выведем на экран. Код будет выглядеть следующим образом:

```
from thinkdsp import read_wave

wave = read_wave('680840__seth_makes_sounds__homemade.wav')
wave.normalize()
wave.make_audio()
wave.plot()
```

Результат выполнения кода выглядит следующим образом:

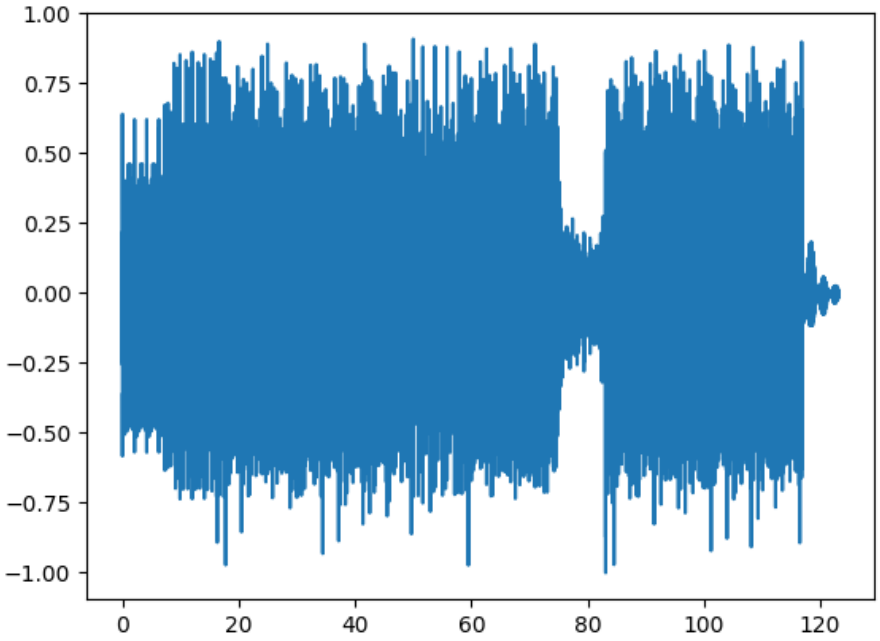

Рис. 1.1. Спектрограмма аудио файла.

Данный отрезок слишком длинный, выделим из него отрезок длинной пол секунды, начиная с 40 секунды аудио файла. Выведем полученный сегмент на экран, используя следующий код:

```
segmet = wave.segment(start=40.0, duration=0.5)
segmet.make_audio()
segmet.plot()
```

Спектрограмма заданного сегмента выглядит следующим образом:

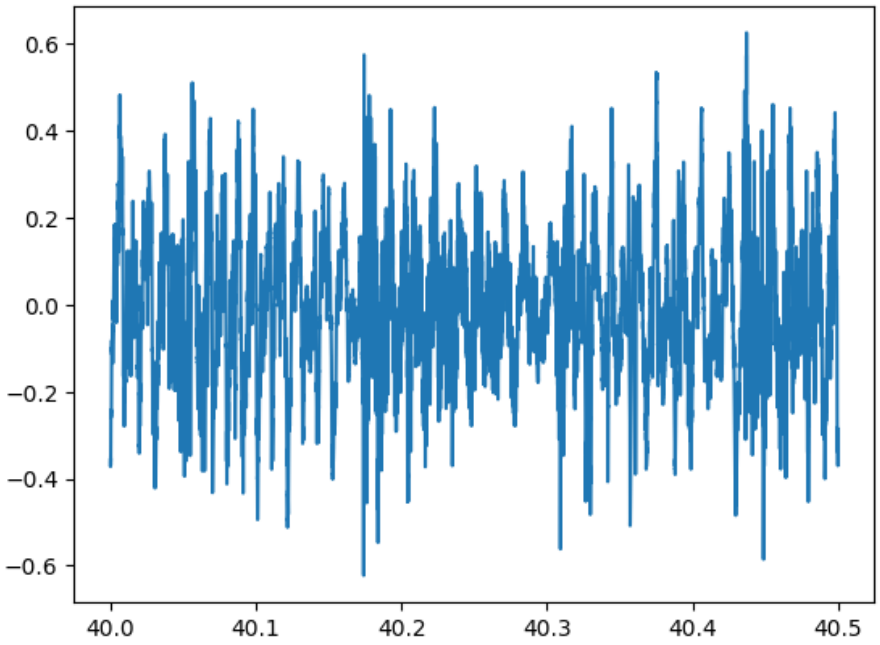

Рис. 1.2. Спектрограмма аудио файла с 40.0 по 40.5 секунды.

Разложим полученный отрезок в спектр и выведем на экран. Код будет выглядеть следующем образом:

```
1  spectrum = segment.make_spectrum()
2  spectrum.plot(high=5000)
```

Этот код выведет спектр до 5000 частоты т.к. далее частоты равны примерно нулю:

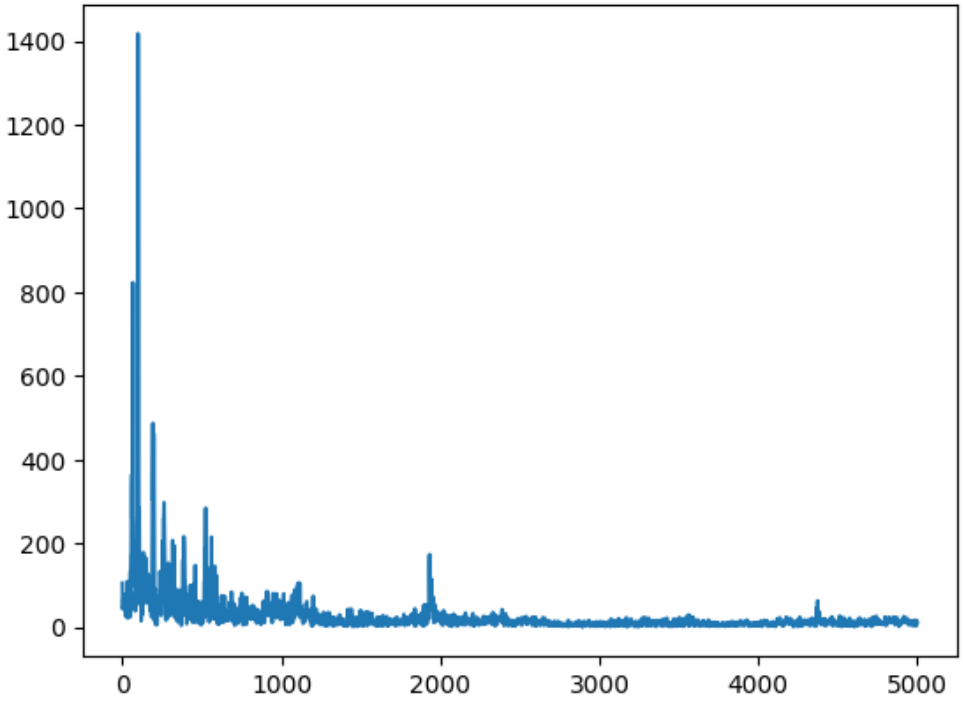

Рис. 1.3. Результат разложения сегмента в спектр.

Доминантной частотой в этом отрывке является 98 Гц.

Теперь поэкспериментируем с функциями high_pass, low_pass и band_stop, которые фильтруют гармоники.

Начнем с low_pass:

```
spectrum.make_wave().make_audio()
spectrum.low_pass(2000)
spectrum.plot(high=5000)
spectrum.make_wave().make_audio()
```

Данный код сохраняет музыкальный фрагмент (для дальнейшего сравнения), после чего применяет функцию low_pass и выводит его спектр на экран, а также опять сохраняет фрагмент. Полученный спектр выглядит следующим образом:

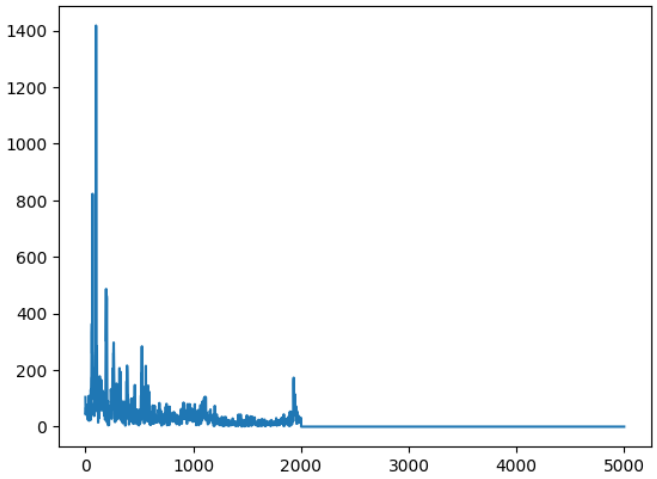

Рис. 1.4. Спектр фрагмента после применения low_pass.

Как видно из рисунка выше, данная функция полностью убрала частоты, выше 2000. Таким образом звук стал более «глухим» и «отдаленным».

Теперь к исходному сегменту применим метод high_pass:

```
spectrum.make_wave().make_audio()
spectrum.high_pass(1000)
spectrum.plot(high=5000)
spectrum.make_wave().make_audio()
```

Полученный спектр имеет следующий вид:

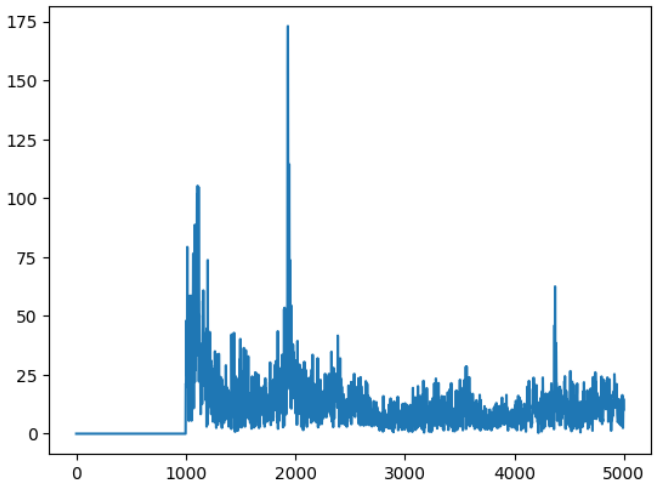

Рис. 1.5. Спектр фрагмента после применения high_pass.

Как видно по спектру, эта функция убирает все частоты ниже заданной. Таким образом звук сильно поменял свое звучание, став более шипящим и менее глубоким.
И последняя функция band_stop:

```
spectrum.make_wave().make_audio()
spectrum.band_stop(low_cutoff=100, high_cutoff=1000)
spectrum.plot(high=5000)
spectrum.make_wave().make_audio()
```

Полученный спектр выглядит следующим образом:

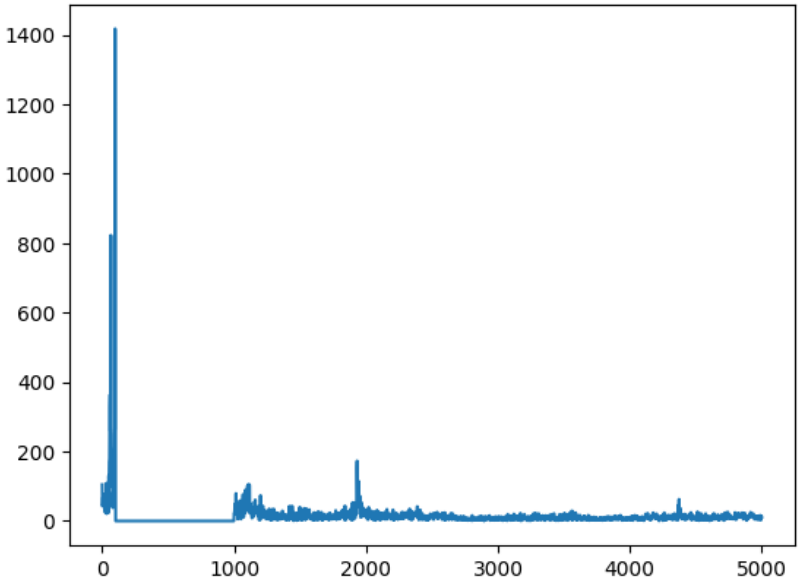

Рис. 1.6. Спектр фрагмента после применения band_stop.

Как мы видим, данная функция убирает частоты из заданного диапазона. Звук фрагмента при удалении частот со 100 Гц до 1000 Гц сильно изменился, в нем практически не слышны ударные.

1.2. Упражнение 1.3.

Создадим сигнал, состоящий из синусов, разной частоты, однако кратных одному числу, например 200:

```
from thinkdsp import SinSignal

signal = (SinSignal(freq=400, amp=1.0) +
          SinSignal(freq=600, amp=1.0) +
          SinSignal(freq=800, amp=1.0))
signal.plot()
```

Полученный сигнал имеет следующий вид:

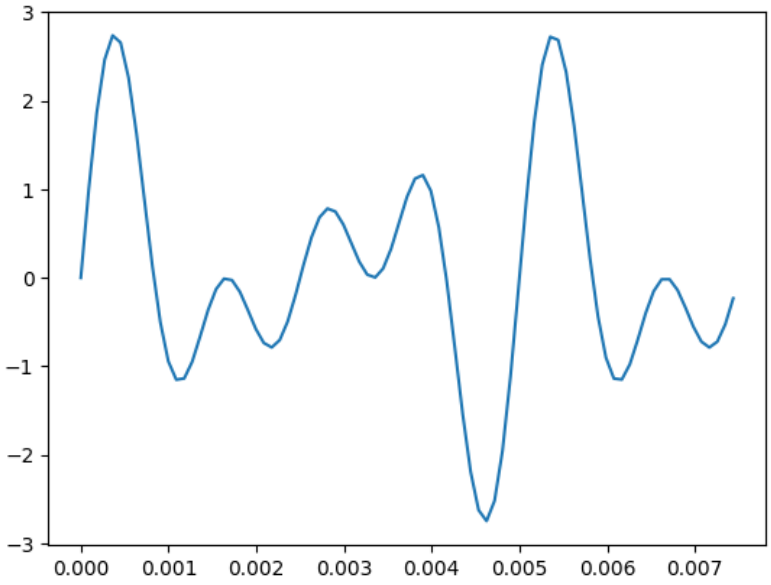

Рис. 1.7. Сигнал, полученный суммой синусов разной частоты.

Создадим файл для прослушивания этого звука, длинной 1 секунда:

```
wave = signal.make_wave(duration=1)
wave.apodize()
wave.make_audio()
```

Полученный звуковой файл является однотонным писком, похожим на звук гудка, но монотонного.

Выведем спектр полученного сигнала:

```
spectrum = wave.make_spectrum()
spectrum.plot(high=2000)
```

Результат выглядит следующим образом:

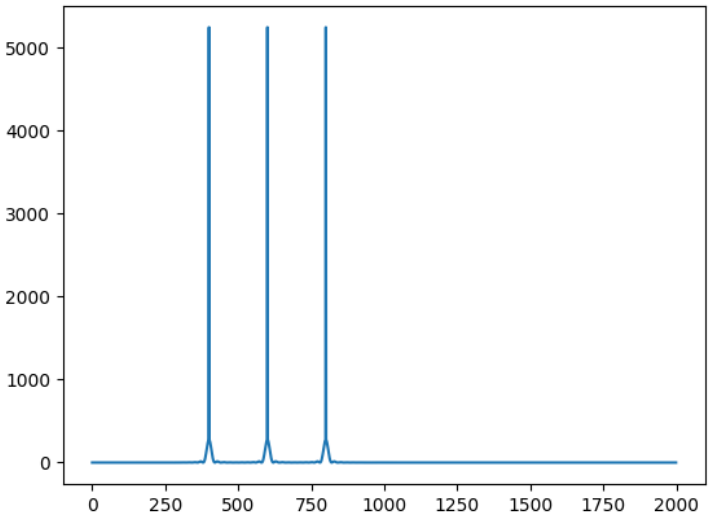

Рис. 1.8. Спектр сигнала, полученного суммой синусов разной частоты.

Как видим, спектр полностью соответствует ожидания, на нем пики находятся именно в тех частотах, которые мы указывали при создании.

Теперь изменим наш сигнал, добавив частоту, не кратную 200:

```
signal += SinSignal(freq=450, amp=1.0)
signal.plot()
signal.make_wave().make_audio()
```

Полученный сигнал имеет следующий вид:

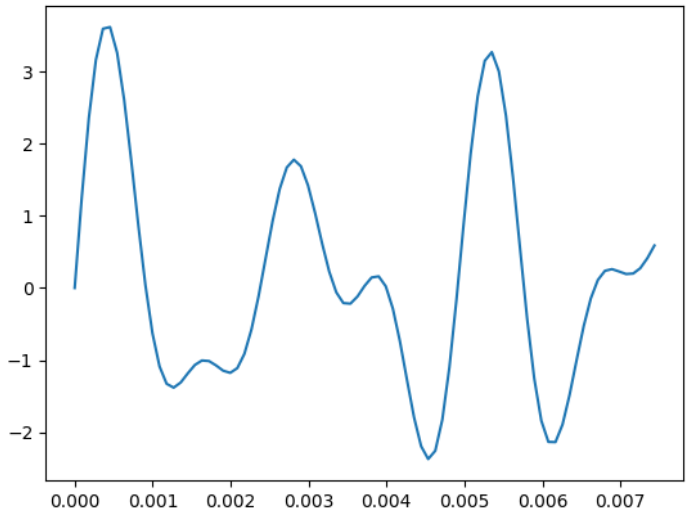

Рис. 1.9. Сигнал, после добавления синуса не кратной частоты.

Полученный сигнал сильно отличается от того, который был ранее. Так же аудио файл тоже чуть-чуть отличается. В монотонном звуке гудка различим какой-то посторонний периодический сигнал.

Выведем спектр полученного сигнала:

```
wave = signal.make_wave(duration=1)
wave.apodize()
spectrum = wave.make_spectrum()
spectrum.plot(high=2000)
```

Полученный спектр имеет следующий вид:

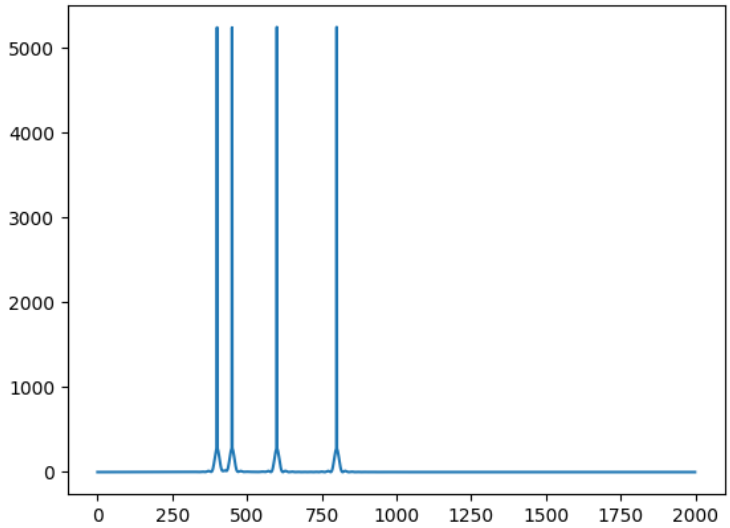

Рис. 1.10. Спектр сигнала, после добавления синуса не кратной частоты.

Как и ожидалось, в спектре появился добавленный ранее сигнал.

1.3. Упражнение 1.4.

Напишем функцию для ускорения и замедления аудио. Для начала прочитаем аудио фрагмент и выведем его на экран:

```
from thinkdsp import read_wave

wave = read_wave('680840__seth_makes_sounds__homemade.wav')
wave.normalize()
wave.plot()
wave.make_audio()
```

Спектрограмма будет выглядеть следующим образом:

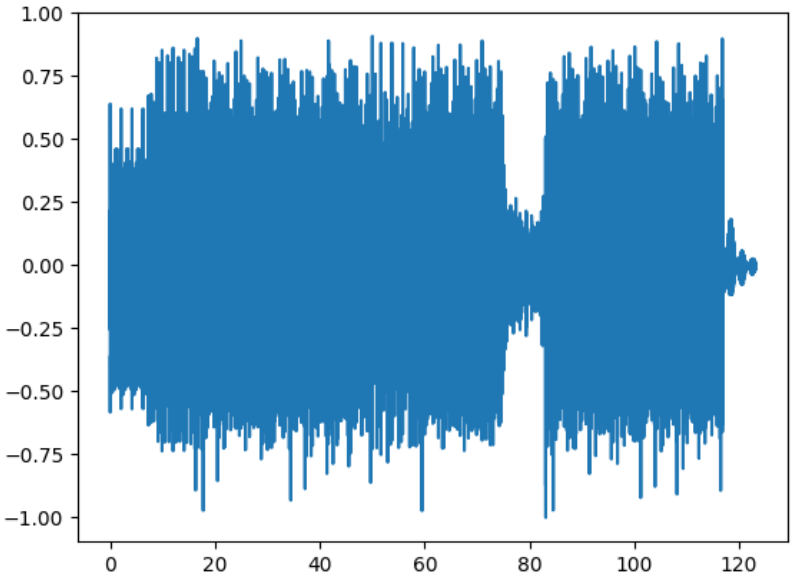

Рис. 1.11. Спектрограмма аудио файла.

Функция для ускорения будет выглядеть следующим образом:

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

Она изменяет ts (которое используется для корректного отображения временной шкалы в plot) и framerate, что, собственно, и ускоряет произведение.

Передадим функции значение 0.5, что эквивалентно ускорению в 2 раза:

```
stretch(wave, 0.5)
wave.plot()
wave.make_audio()
```

После выполнения мы получили аудио файл, который ускорен в 2 раза, как и ожидалось.

Посмотрим на полученную спектрограмму:

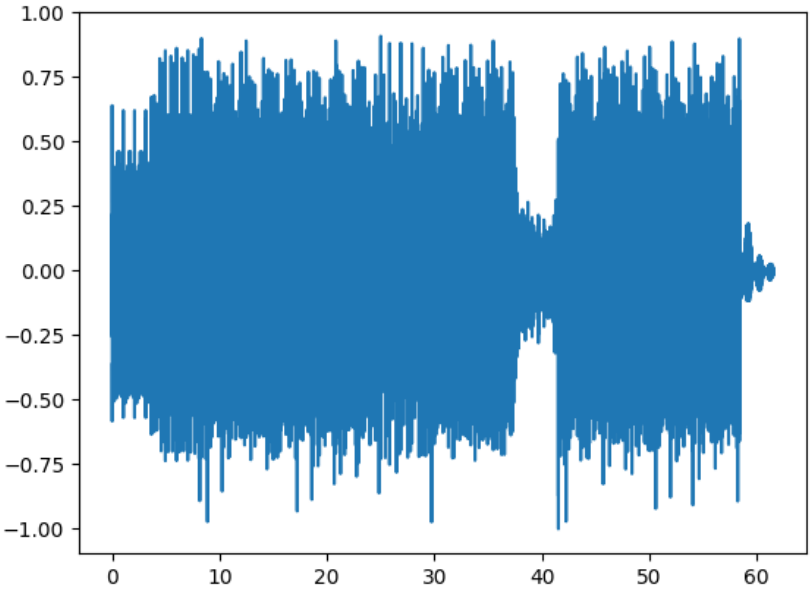

Рис. 1.12. Спектрограмма аудио файла после ускорения.

Как мы видим, полученная спектрограмма не отличается от исходной ничем, кроме длительности аудио фрагмента, он меньше в 2 раза.

2. Приложение:

Ссылка на репозиторий с исходными кодами: https://github.com/DafterT/telecom_labs