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)

“23” февраля 2024 г.

Санкт-Петербург

2024

Оглавление

[1. Лабораторная работа 1. Сигналы и звуки. 3](#_Toc159159916)

[1.1. Упражнение 1.2. 3](#_Toc159159917)

[1.2. Упражнение 1.3. 6](#_Toc159159918)

[1.3. Упражнение 1.4. 9](#_Toc159159919)

[2. Лабораторная работа 2. Гармоники. 10](#_Toc159159920)

[2.1. Упражнение 2.2. 10](#_Toc159159921)

[2.2. Упражнение 2.3. 14](#_Toc159159922)

[2.3. Упражнение 2.4. 15](#_Toc159159923)

[2.4. Упражнение 2.5. 17](#_Toc159159924)

[2.5. Упражнение 2.6. 18](#_Toc159159925)

[3. Лабораторная работа 3. Апериодические сигналы. 20](#_Toc159159926)

[3.1. Упражнение 3.1. 20](#_Toc159159927)

[3.2. Упражнение 3.2. 21](#_Toc159159928)

[3.3. Упражнение 3.3. 23](#_Toc159159929)

[3.4. Упражнение 3.4. 23](#_Toc159159930)

[3.5. Упражнение 3.5. 24](#_Toc159159931)

[3.6. Упражнение 3.6. 25](#_Toc159159932)

[4. Лабораторная работа 4. Шум. 28](#_Toc159159933)

[4.1. Упражнение 4.1. 28](#_Toc159159934)

[4.2. Упражнение 4.2. 31](#_Toc159159935)

[4.3. Упражнение 4.3. 32](#_Toc159159936)

[4.4. Упражнение 4.4. 34](#_Toc159159937)

[4.5. Упражнение 4.5. 35](#_Toc159159938)

[5. Лабораторная работа 5. Автокорреляция. 38](#_Toc159159939)

[5.1. Упражнение 5.2. 38](#_Toc159159940)

[5.2. Упражнение 5.3. 41](#_Toc159159941)

[5.3. Упражнение 5.4. 42](#_Toc159159942)

[6. Приложение: 48](#_Toc159159943)

# Лабораторная работа 1. Сигналы и звуки.

## Упражнение 1.2.

Скачаем с сайта [https://freesound.org/](https://freesound.org/%20) 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:



Результат выполнения кода выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, График, диаграмма

Автоматически созданное описание

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:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

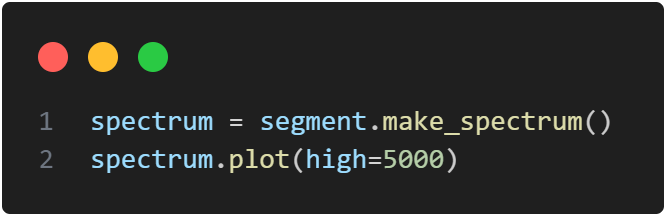
Спектрограмма заданного сегмента выглядит следующим образом:

Изображение выглядит как снимок экрана, текст, График, диаграмма

Автоматически созданное описание

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:



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

Изображение выглядит как снимок экрана, График, диаграмма, линия

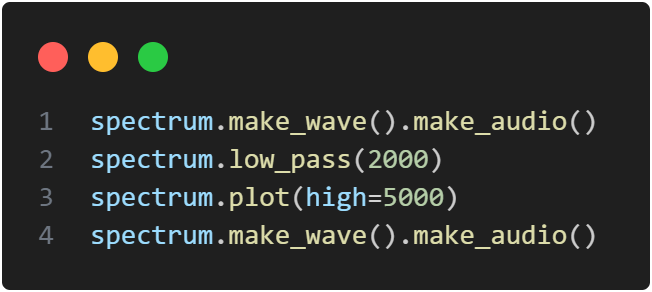
Автоматически созданное описание

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:



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:

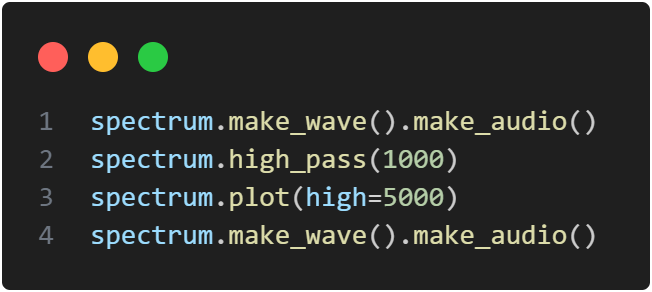
Изображение выглядит как снимок экрана, текст, График, диаграмма

Автоматически созданное описание

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:



Полученный спектр имеет следующий вид:

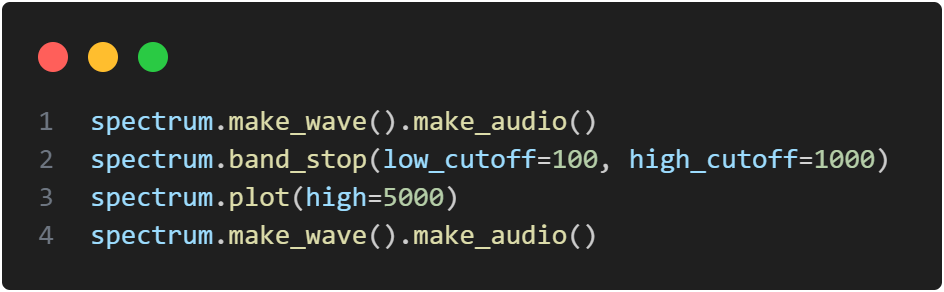
Изображение выглядит как снимок экрана, текст, График, диаграмма

Автоматически созданное описание

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:



Полученный спектр выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, График, диаграмма

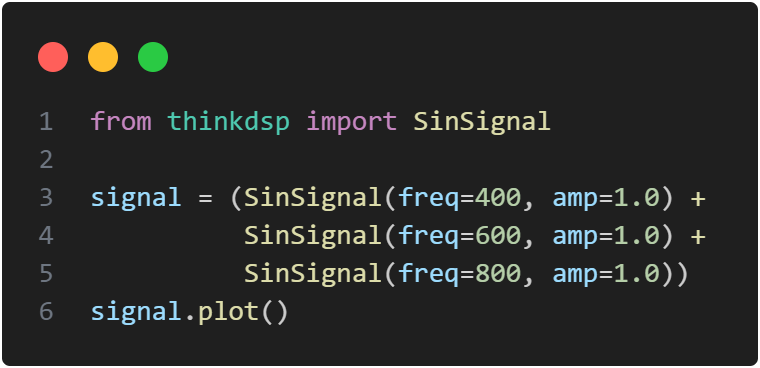
Автоматически созданное описание

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.

## Упражнение 1.3.

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



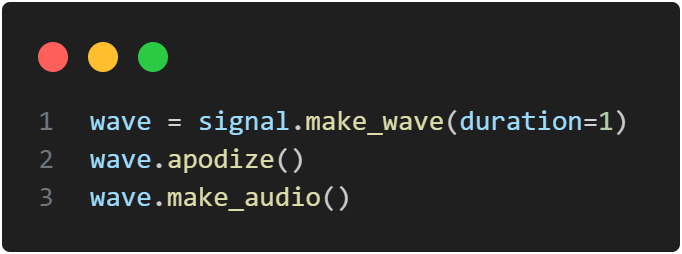
Полученный сигнал имеет следующий вид:

Изображение выглядит как График, линия, диаграмма, текст

Автоматически созданное описание

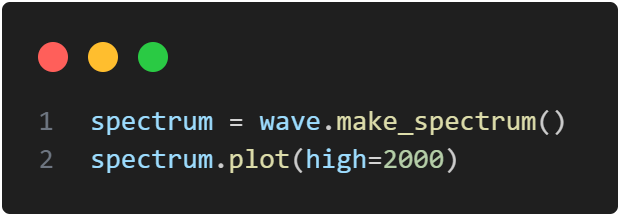
Рис. 1.7. Сигнал, полученный суммой синусов разной частоты.

Создадим файл для прослушивания этого звука, длинной 1 секунда:



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

Выведем спектр полученного сигнала:



Результат выглядит следующим образом:

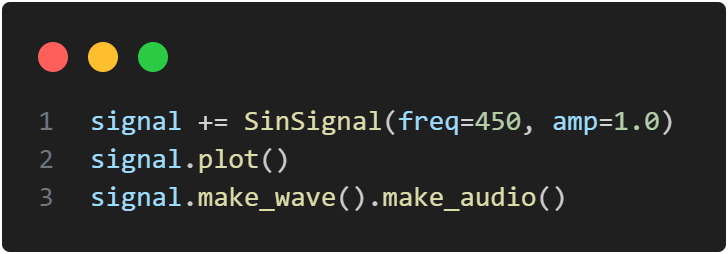
Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание

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:



Полученный сигнал имеет следующий вид:

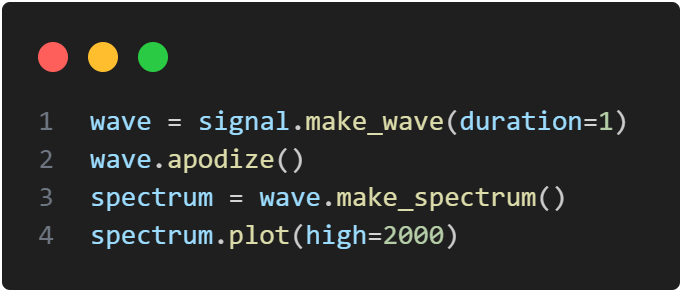
Изображение выглядит как График, диаграмма, линия, Шрифт

Автоматически созданное описание

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:



Полученный спектр имеет следующий вид:

Изображение выглядит как текст, снимок экрана, линия, Прямоугольник

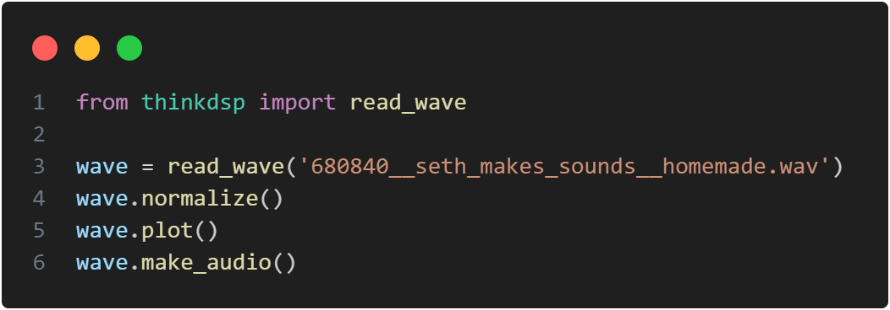
Автоматически созданное описание

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

Как и ожидалось, в спектре появился добавленный ранее сигнал.

## Упражнение 1.4.

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



Спектрограмма будет выглядеть следующим образом:

Изображение выглядит как текст, снимок экрана, График, диаграмма

Автоматически созданное описание

Рис. 1.11. Спектрограмма аудио файла.

Функция для ускорения будет выглядеть следующим образом:



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:



После выполнения мы получили аудио файл, который ускорен в 2 раза, как и ожидалось. Посмотрим на полученную спектрограмму:

Изображение выглядит как текст, снимок экрана, График, диаграмма

Автоматически созданное описание

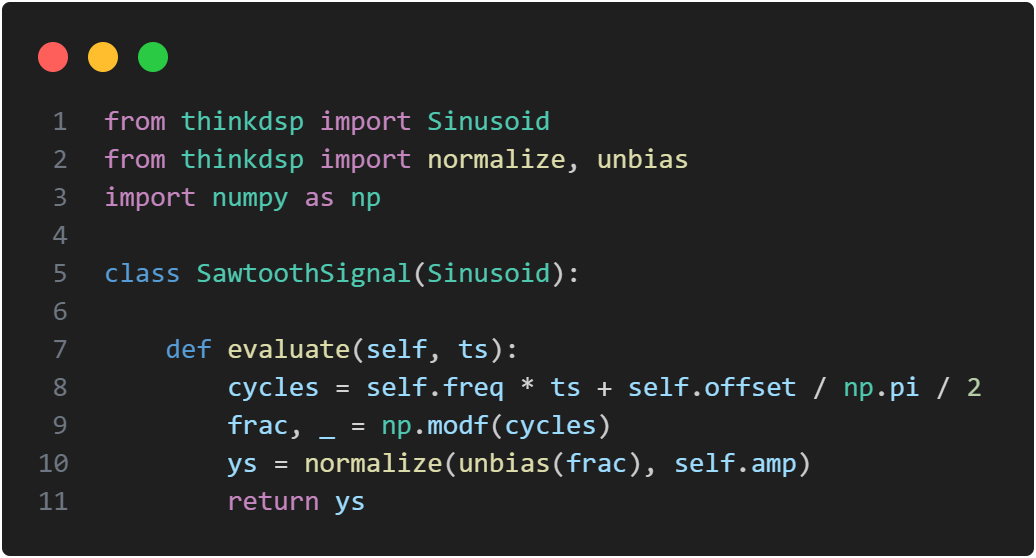
Рис. 1.12. Спектрограмма аудио файла после ускорения.

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

# Лабораторная работа 2. Гармоники.

## Упражнение 2.2.

Разработаем класс, который бы наследовался от Sinusoid из thinkdsp, который позволял бы строить пилообразный сигнал (нарастает от -1 до 1, а затем резко падает до -1). Переопределить необходимо только функцию evaluate:



Здесь:

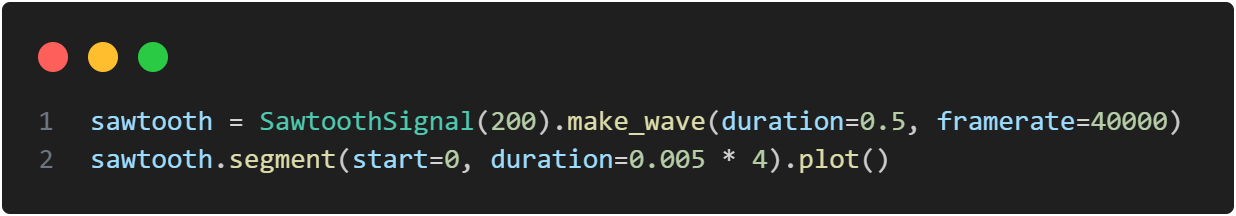
cycles – число циклов со времени старта.

frac – дробная часть, растущая от 0 до 1 за период.

unbias – сдвигает frac так, что он растет от до .

normalize – нормализует функцию, чтоб она росла от self.amp до self.amp.

Создадим экземпляр этого класса и сразу же получим из него Wave, после чего выведем его часть на экран, дабы проверить, что функция реализованная корректно:



Результат запуска выглядит следующим образом:

Изображение выглядит как текст, линия, снимок экрана, График

Автоматически созданное описание

Рис. 2.1. Пилообразный сигнал.

Создадим спектр этого сигнала, выведем на экран, а также посмотрим наибольшие 10 пиков, дабы изучить каким образом частота зависит от амплитуды:

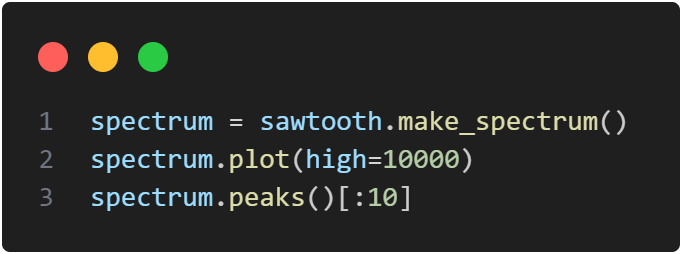


График будет выглядеть таким образом:

Изображение выглядит как текст, снимок экрана, График, линия

Автоматически созданное описание

Рис. 2.2. Спектр пилообразного сигнала.

А также мы получаем следующий массив пиков:

[(6336.586158412468, 200.0),

(3168.547531644226, 400.0),

(2112.647887262727, 600.0),

(1584.783147437211, 800.0),

(1268.132547579567, 1000.0),

(1057.089208734752, 1200.0),

(906.3930765164864, 1400.0),

(793.4141558611690, 1600.0),

(705.5802559636873, 1800.0),

(635.3480882678591, 2000.0)]

Как можно заметить, сигнал содержит как четные, так и нечетные гармоники, а также они уменьшаются пропорционально .

Сравним полученный спектр пилообразного сигнала с прямоугольным:

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Стоит отметить, что прямоугольный сигнал создается с amp=0.5, чтоб выровнять спектрограмму для сравнения её с пилообразным сигналом.

Полученный график выглядит так:

Изображение выглядит как текст, снимок экрана, дисплей, линия

Автоматически созданное описание

Рис. 2.3. Спектрограммы пилообразного и прямоугольного сигналов.

Для удобства сравнения так же проанализируем первые 10 пиков прямоугольного сигнала:

[(6366.41311530820, 200.0),

(2122.71230545574, 600.0),

(1274.31761659952, 1000.0),

(910.967679013610, 1400.0),

(709.300517302879, 1800.0),

(581.126830719483, 2200.0),

(492.527938505399, 2600.0),

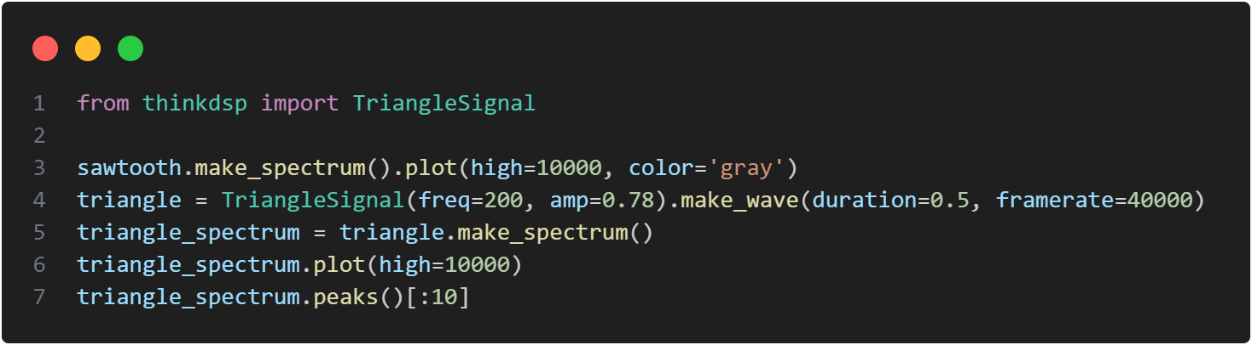
(427.675369582061, 3000.0),

(378.189565961787, 3400.0),

(339.219405574489, 3800.0)]

Стоит обратить внимание, что в отличии от пилообразного сигнала, прямоугольный сигнал имеет только нечетные гармоники, а вот зависимость падения от частоты сохраняется и пропорционально .

Так же выполним аналогичное сравнение с треугольным сигналом:



Аналогично прямоугольному сигналу необходимо изменить amp, однако для треугольного сигнала это значение равно .

График выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, линия, дисплей

Автоматически созданное описание

Рис. 2.4. Спектрограммы пилообразного и треугольного сигналов.

Для удобства сравнения так же проанализируем первые 10 пиков треугольного сигнала:

[(6322.961884943854, 200.0),

(703.0137709503831, 600.0),

(253.4183165242379, 1000.0),

(129.5506855043607, 1400.0),

(78.57692291999746, 1800.0),

(52.77470536760347, 2200.0),

(37.93526415250758, 2600.0),

(28.62556659255658, 3000.0),

(22.40446225716876, 3400.0),

(18.04308559845682, 3800.0)]

Как мы видим, этот сигнал ведет себя совершенно отлично, от пилообразного. В первую очередь мы видим, что в нем присутствуют только нечетные гармоники, а также зависимость падения от частоты пропорциональна .

## Упражнение 2.3.

Создадим прямоугольный сигнал с частотой 1100 Гц и выборкой 10000 кадров в секунду. Отобразим получившийся спектр, а также первые 10 пиков:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Получившийся спектр выглядит таким образом:

Изображение выглядит как текст, снимок экрана, График, линия

Автоматически созданное описание

Рис. 2.5. Спектрограмма прямоугольного сигнала с частотой 1100 Гц и выборкой 10000 кадров в секунду.

А также для удобства анализа приведем старшие 10 пиков:

[(3183.622520909762, 1100.0),

(1062.605379628311, 3300.0),

(639.2453221499661, 4500.0),

(458.4143857027373, 2300.0),

(358.4343652372162, 100.0),

(295.2134792809340, 2100.0),

(251.7953698310349, 4300.0),

(220.2689264585266, 3500.0),

(196.4476698867248, 1300.0),

(177.9095485479867, 900.0)]

Как мы помним, прямоугольный сигнал имеет только нечетные гармоники, а зависимость падения амплитуды от частоты пропорциональна .

Ожидается, что гармоники будут на 3300, 5500, 7700 и 9900 Гц. Как мы видим, пики есть на 1100 и 3300 Гц, однако дальше наблюдается эффект биения, поэтому вместо 5500 мы получаем лишь 4500 (10000 – 5500), а следующая гармоника вместо 7700 получается 2300 (10000 – 7700). Из-за этого сигнал звучит совершенно по-другому, а именно появляются лишние низкие частоты (например, 100), а также посторонние не кратные частоты (2300), которые сильно выбиваются.

Продемонстрируем это, создав аудио файл, исходного прямоугольного сигнала, а также две синусоиды на 2300 и 100 Гц и убедимся, что они достаточно заметны:

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

При прослушивании этих записей действительно заметно, что они сильно выбиваются из исходного сигнала.

## Упражнение 2.4.

Создадим треугольный сигнал и выведем его график на экран:

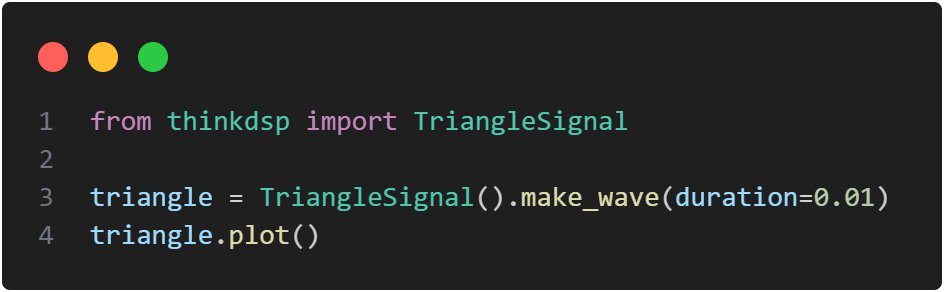


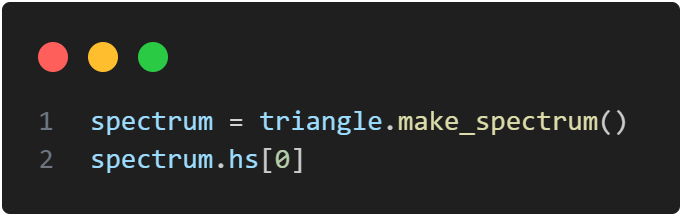
График выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, линия, График

Автоматически созданное описание

Рис. 2.6. График треугольного сигнала.

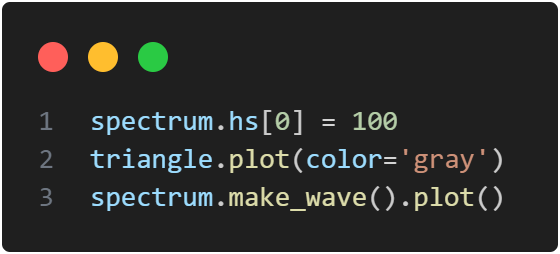
Теперь получим спектр этого сигнала и выведем первый элемент массива hs, который является результатом БПФ:



Результат примерно равен нулю:

(1.0436096431476471e-14+0j)

Изменим его значение на 100 и посмотрим на результат:



В результате получаем следующий график:

Изображение выглядит как линия, текст, График, диаграмма

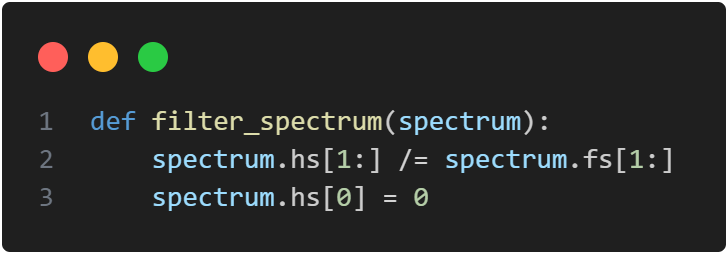
Автоматически созданное описание

Рис. 2.7. График треугольного сигнала, после изменения hs.

Как можно заметить, получившийся сигнал отличается от исходного только вертикального смещения.

## Упражнение 2.5.

Напишем функцию filter\_spectrum, которая принимает спектр и изменяет его, выполняя деление каждый элемент hs, на соответствующую частоты из fs:



Создадим треугольный сигнал, выведем его в виде аудио для дальнейшего сравнения, после чего вызовем нашу функцию фильтрации. Выведем спектрограмму до и после, а также аудио файл после использования функции:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Получившийся график выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Рис. 2.8. Спектрограмма до и после использования функции.

Как можно заметить, чем больше частота, тем меньше становится пик после использования функции. Это логично, ведь деление происходит именно на частоту. Результат похож на low\_pass фильтр.

Так же при прослушивании полученных аудио, мы получаем схожий результат.

## Упражнение 2.6.

Создадим сигнал, в котором есть как четные, так и нечетные гармоники, которые спадают пропорционально . Сигнал будем собирать, используя несколько синусоид:



Как мы видим. Частоты будут изменяться от 500 до 9000 с шагом 500. Амплитуда вычисляется, путем деления , как это требует задание.

График сигнала будет выглядеть следующим образом:

Изображение выглядит как текст, снимок экрана, График, линия

Автоматически созданное описание

Рис. 2.9. Сигнал с четными и нечетными гармониками и амплитудой пропорциональной

Убедимся, что полученный сигнал соответствует требованиям, выведем его спектрограмму:

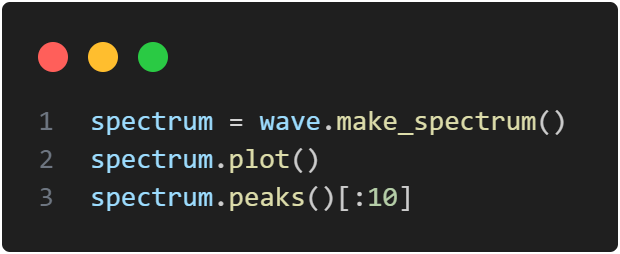


График выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, дисплей, График

Автоматически созданное описание

Рис. 2.10. Спектрограмма сигнала.

Для удобства анализа посмотрим на первые 10 пиков сигнала:

[(2000.0, 500.0),

(500.00, 1000.0),

(222.22, 1500.0),

(125.00, 2000.0),

(80.000, 2500.0),

(55.555, 3000.0),

(40.816, 3500.0),

(31.250, 4000.0),

(24.691, 4500.0),

(20.000, 5000.0)]

Как мы видим, спектрограмма соответствует требованиям задания.

# Лабораторная работа 3. Апериодические сигналы.

## Упражнение 3.1.

Выполним сравнение различных оконных функций, а именно стандартной hamming и bartlett, blackman, hanning.

Создадим сначала синусоидальный сигнал, с частотой 440 и сделаем, чтоб сигнал начинался с 0, а заканчивался в 1. Выведем его спектрограмму:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Результат выглядит следующим образом:

Изображение выглядит как диаграмма, текст, линия, снимок экрана

Автоматически созданное описание

Рис. 3.1. Спектрограмма сигнала.

А теперь применим к этой спектрограмме различные оконные функции:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Результат выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, диаграмма, График

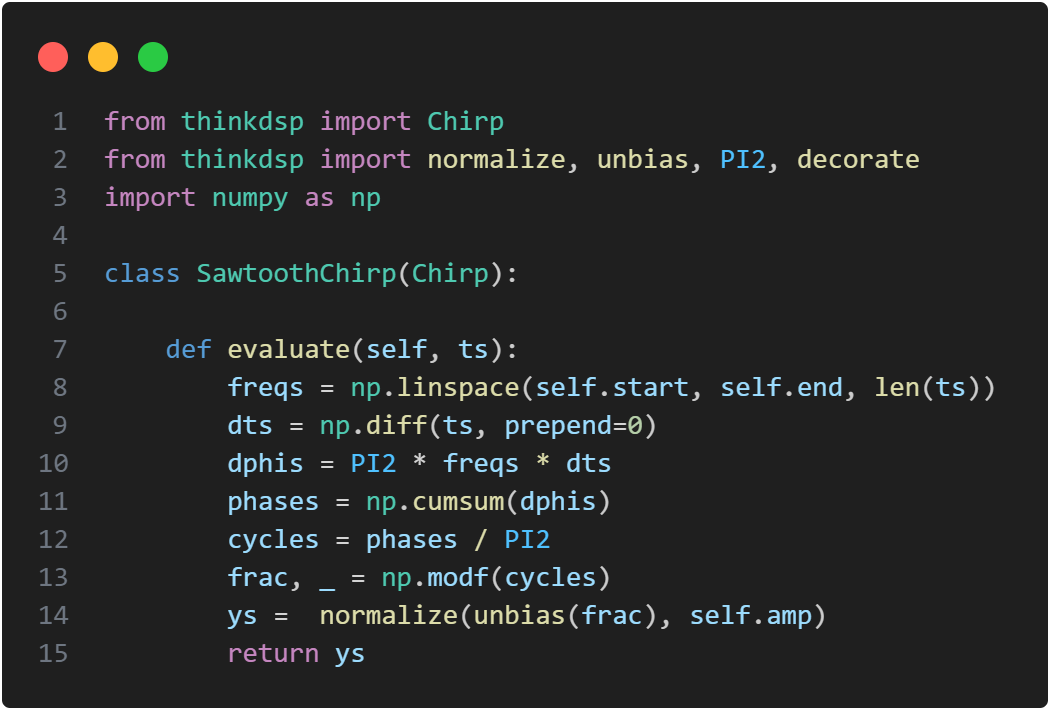
Автоматически созданное описание

Рис. 3.2. Спектрограммы после применения оконных функций.

Как мы видим, функция Хемминга показывает себя лучше всего, именно поэтому она выбрана как универсальный вариант.

## Упражнение 3.2.

Создадим класс, расширяющий Chirp для создания увеличивающегося пилообразного сигнала:



Эта функция является совмещением функции, созданной в 2.1 и функции для создания Chirp.

Создадим экземпляр этого класса с начальной частотой 220 и конечной 880:

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Убедимся в том, что полученная запись соответствует ожиданиям:

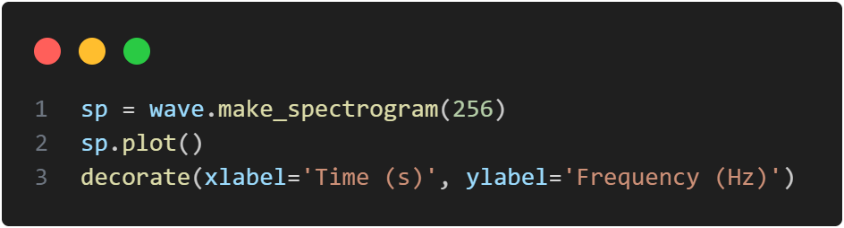
Изображение выглядит как текст, снимок экрана, Шрифт, График

Автоматически созданное описание

Рис. 3.3. Пилообразный чирп.

Как мы видим, это действительно пилообразный чирп, так же это слышно по аудио записи.

Создадим спектрограмму заданного сигнала:



Получившаяся спектрограмма выглядит следующим образом:

Изображение выглядит как снимок экрана, желтый, шаблон, линия

Автоматически созданное описание

Рис. 3.4. Спектрограмма пилообразного сигнала.

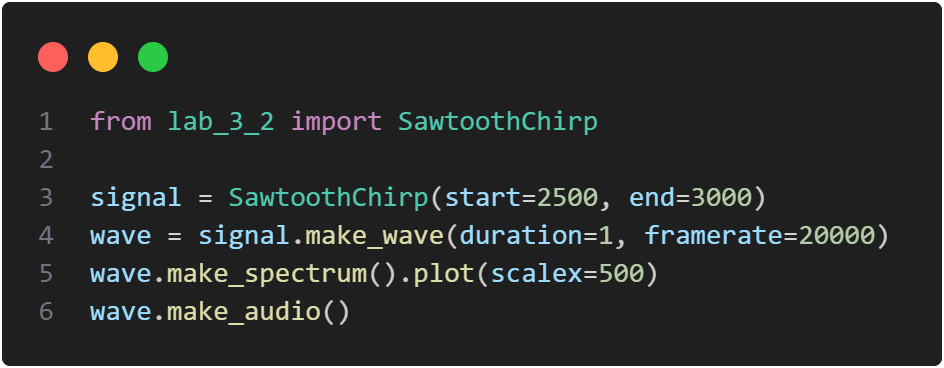
Здесь стоит отметить, что в связи с маленьким фреймрейтом мы получаем достаточно шумный спектр из-за заворота (биения), этот эффект хорошо заметен, как «отскакивающие» от верхней границы темные участки.

## Упражнение 3.3.

Попытаемся предположить, какой будет иметь вид спектр пилообразного чирпа с начальной частотой 2500, конечной 3000 и фреймрейтом 20000.

Поскольку пилообразный сигнал содержит как четные, так и нечетные гармоники, а также они уменьшаются пропорционально можно предположить, что первый пик будет размазан между начальной и конечной частотой, второй будет в 2 раза меньше на частотах с 5000 до 6000, а третий будет на частотах с 7500 до 9000 Гц и будет в 3 раза меньше пика с 2500 до 3000.

Убедимся в этом, создав необходимый чирп:



Получившаяся спектрограмма выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, График, диаграмма

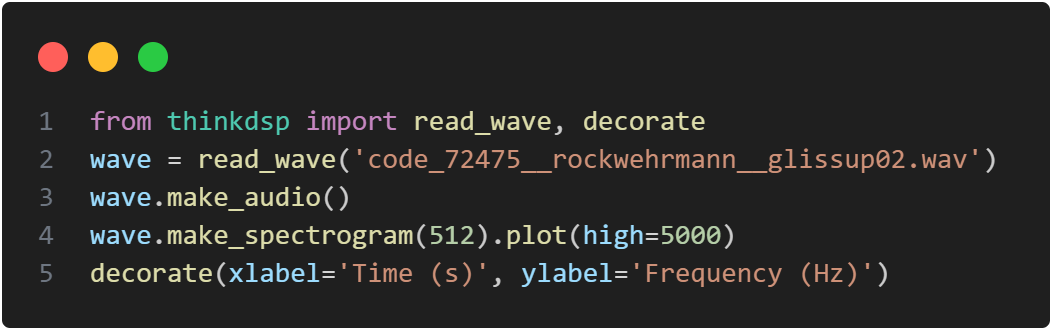
Автоматически созданное описание

Рис. 3.5. Спектрограмма пилообразного чирпа.

Как мы и ожидали спектрограмма полностью соответствует ожиданиям.

## Упражнение 3.4.

Скачаем глиссандо (нота, меняющаяся от одной высоты к другой) и создадим его спектрограмму:



Результирующая спектрограмма выглядит следующим образом:

Изображение выглядит как снимок экрана, желтый, текст, линия

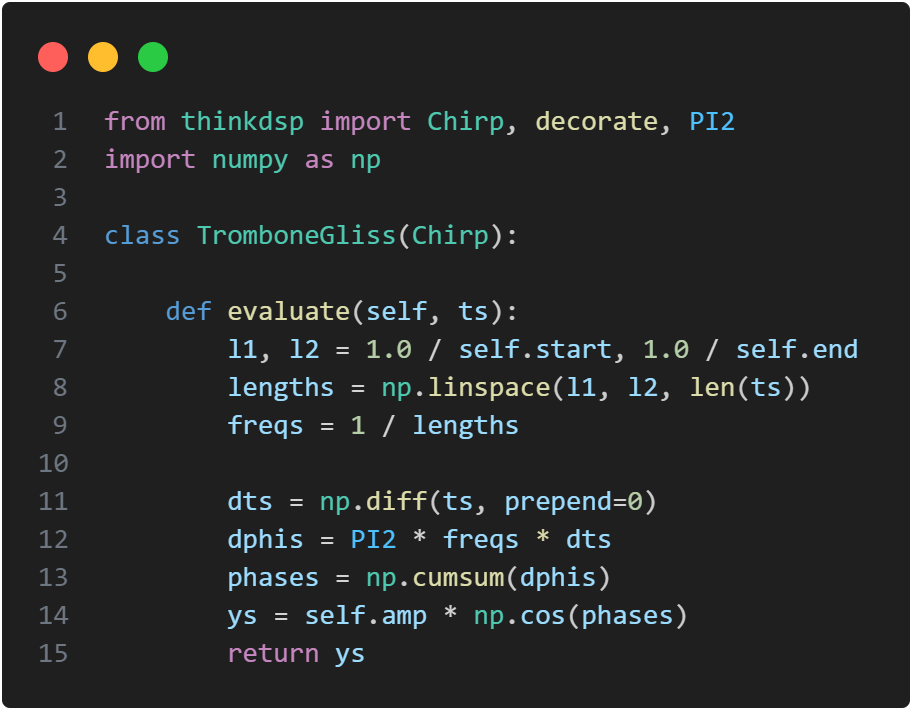
Автоматически созданное описание

Рис. 3.6. Спектрограмма глиссандо.

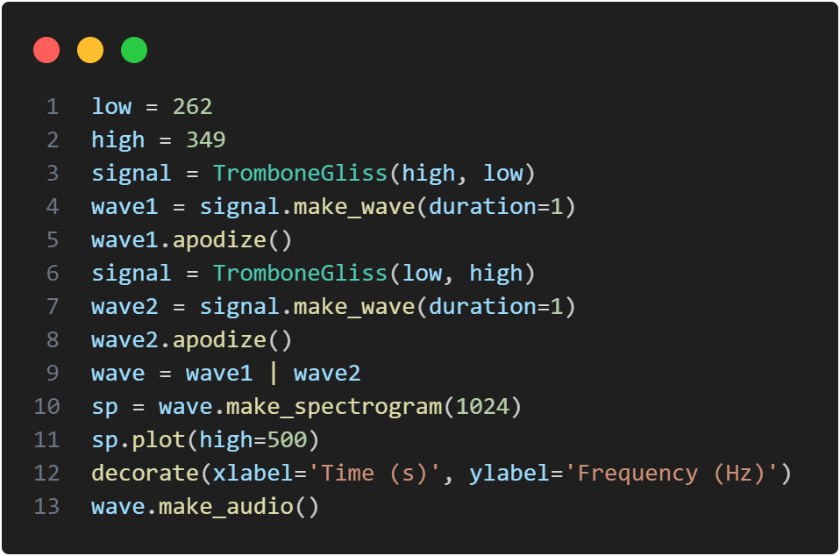
Как можно заметить, частота меняется вместе с изменением высоты ноты (под конец в произведении нота почти не менялась, поэтому и спектрограмма сохраняла свое состояние).

## Упражнение 3.5.

Создадим класс, который будет имитировать глиссандо на тромбоне, при постоянной скорости изменения трубы, если частота звука обратно пропорциональна длине:



В строках с 7 по 9 имитируется изменение длины кулисы тромбона, а далее стандартное объявление функции evaluate.



Получившаяся спектрограмма выглядит следующим образом:

Изображение выглядит как снимок экрана, линия, желтый, Красочность

Автоматически созданное описание

Рис. 3.7. Спектрограмма звука тромбона от С3 до F3 и обратно.

Кажется, что этот сигнал ближе к линейному, однако увеличим разброс до с 1000 Гц до 263:

Изображение выглядит как снимок экрана, График, линия

Автоматически созданное описание

Рис. 3.8. Спектрограмма звука тромбона от 1000 до F3 и обратно.

Как мы видим, теперь сигнал ближе к экспоненциальному, нежели линейному т.к. функция обратно пропорциональна длине, а эта функция степенная, а не линейная.

## Упражнение 3.6.

Далее возьмем аудио с гласными звуками и посмотрим на их спектрограммы, попробуем понять, как они отличаются.

Для начала загрузим аудио и посмотрим на полную спектрограмму:

Изображение выглядит как снимок экрана, Красочность, желтый

Автоматически созданное описание

Рис. 3.9. Спектрограмма гласных звуков.

Теперь обрежем первый звук ‘a’ и посмотрим на его спектрограмму:

Изображение выглядит как текст, снимок экрана, Шрифт, График

Автоматически созданное описание

Рис. 3.10. Спектрограмма звука 'a'.

Теперь обрежем второй звук ‘э’ и посмотрим на его спектрограмму:

Изображение выглядит как текст, снимок экрана, График, линия

Автоматически созданное описание

Рис. 3.11. Спектрограмма звука 'э'.

Теперь обрежем второй звук ‘и’ и посмотрим на его спектрограмму:

Изображение выглядит как текст, снимок экрана, График, линия

Автоматически созданное описание

Рис. 3.12. Спектрограмма звука 'и'.

Теперь обрежем второй звук ‘о’ и посмотрим на его спектрограмму:

Изображение выглядит как текст, снимок экрана, Шрифт, График

Автоматически созданное описание

Рис. 3.13. Спектрограмма звука 'о'.

Теперь обрежем второй звук ‘у’ и посмотрим на его спектрограмму:

Изображение выглядит как текст, снимок экрана, График, линия

Автоматически созданное описание

Рис. 3.14. Спектрограмма звука 'у'

Как можно заметить несмотря на то, что звуки звучат достаточно похоже их спектрограммы сильно отличаются и дают возможность понять, какой действительно звук был произнесен. Вероятно, именно этот метод позволяет распознавать речь и отдельные звуки в современных голосовых помощниках.

# Лабораторная работа 4. Шум.

## Упражнение 4.1.

На сайте <http://asoftmurmur.com/about/> выполним скачивание шумов природы, например звук северного моря. Обрежем его первые 1.5 секунды и выведем его спектр:



В результате получим следующий спектр:

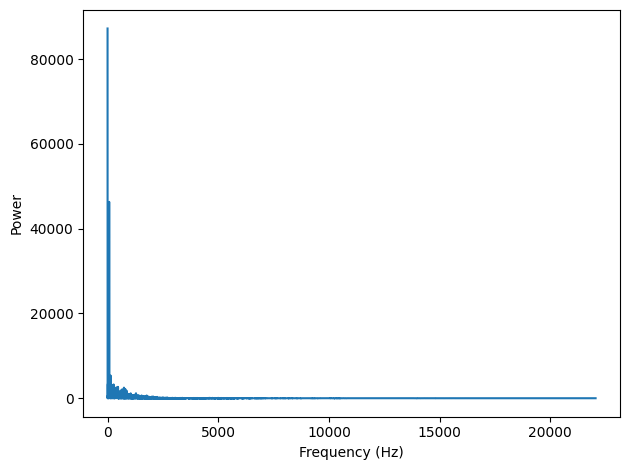
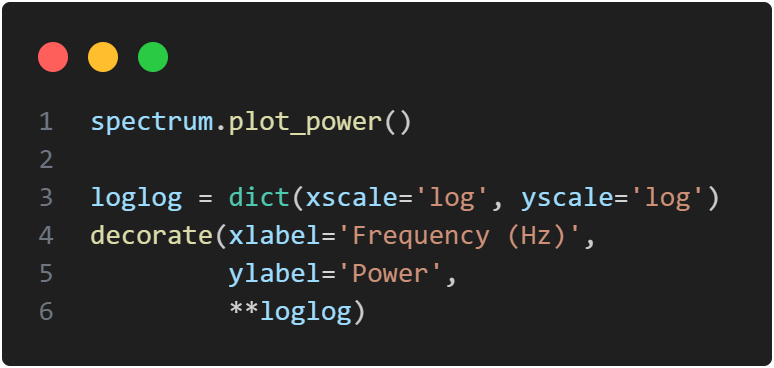


Рис. 4.1. Спектр шума Северного океана

Как мы видим, у нас сильный пик в районе низких частот, а остальные частоты не столь заметны. Этот шум сильно похож на розовый или красный. Проверим это, взглянув на спектр мощности в логарифмическом масштабе:



В результате получаем следующую логарифмическую зависимость:

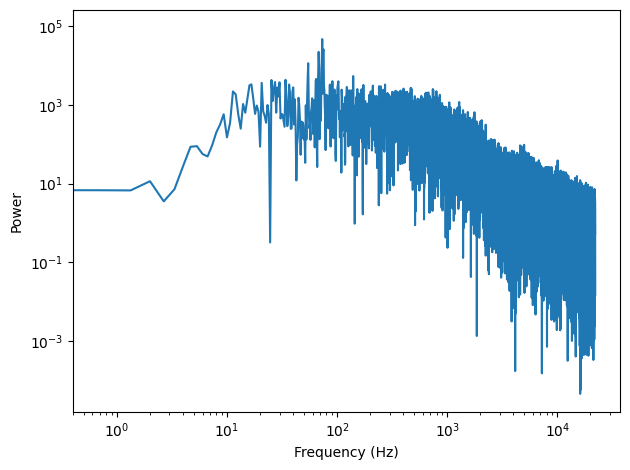
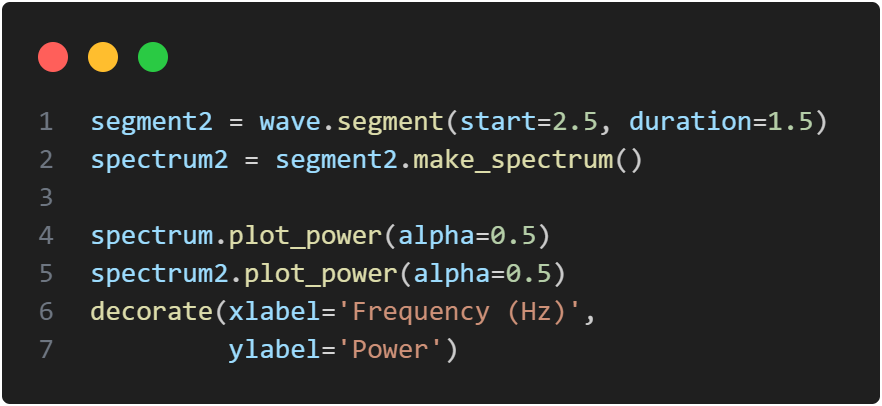


Рис. 4.2. Логарифмическая зависимость мощности от частоты.

Рассматриваемая зависимость необычна т.к. все зависимости, рассмотренные ранее, не возрастали, как эта.

Рассмотрим, как спектрограмма изменяется с течением времени, для этого выберем другой звуковой фрагмент:



Получившийся спектр имеет следующий вид:

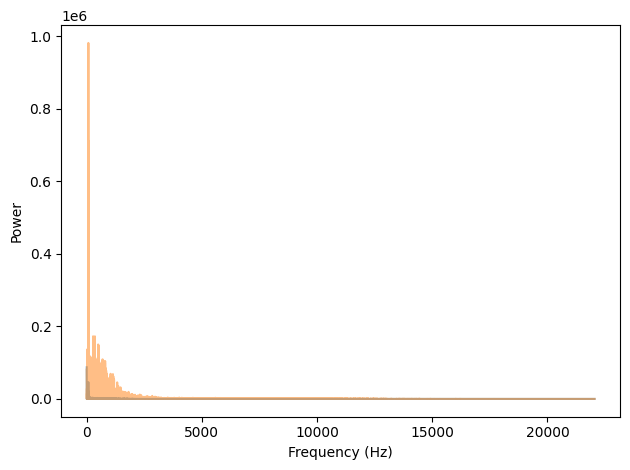
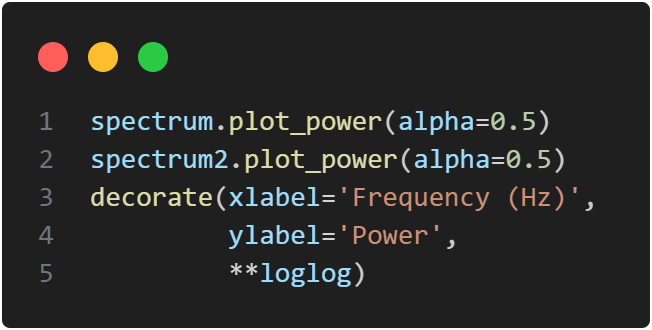


Рис. 4.3. Спектр двух звуковых отрезков.

Как мы видим, мощность второго сигнала имеет совершенно другой порядок, однако и в ней заметно, что преобладают именно низкочастотные сигналы.

Теперь выведем эти сигналы в логарифмической системе:



Получившаяся зависимость приведена ниже:

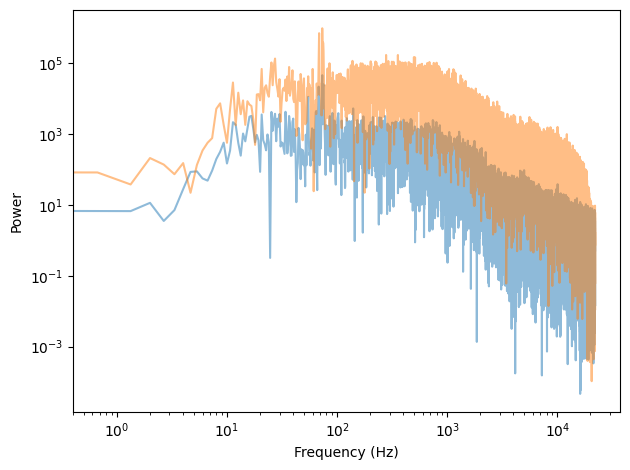
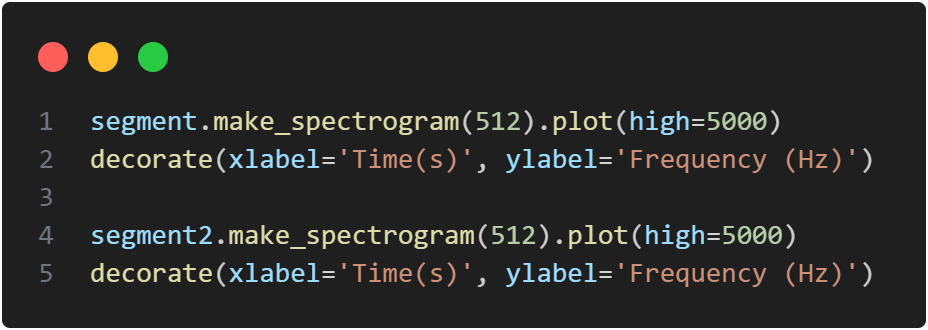


Рис. 4.4. Спектр двух отрезков в логарифмической шкале.

Как мы видим сигналы ведут себя схожим образом. Вероятно их отличие связано с тем, что первый отрывок брался с самого начала, где какое-то время тишина, а лишь потом слышна волна.

Для иллюстрации выведем спектрограмму первого и второго сегментов:



Получившиеся спектрограммы приведены ниже:

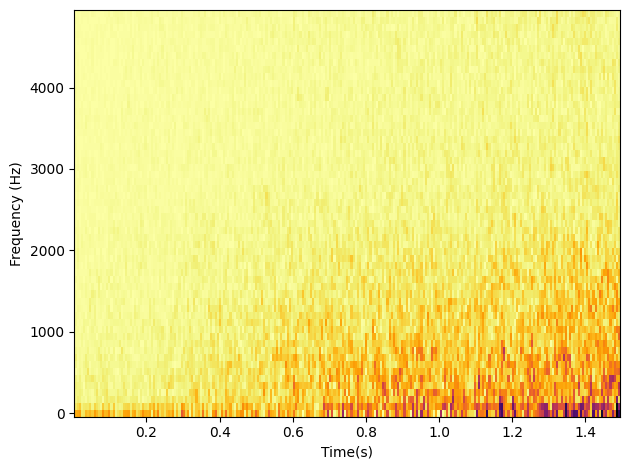
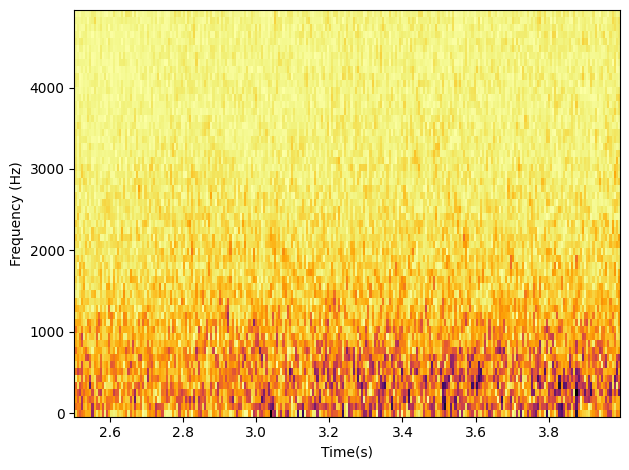
 

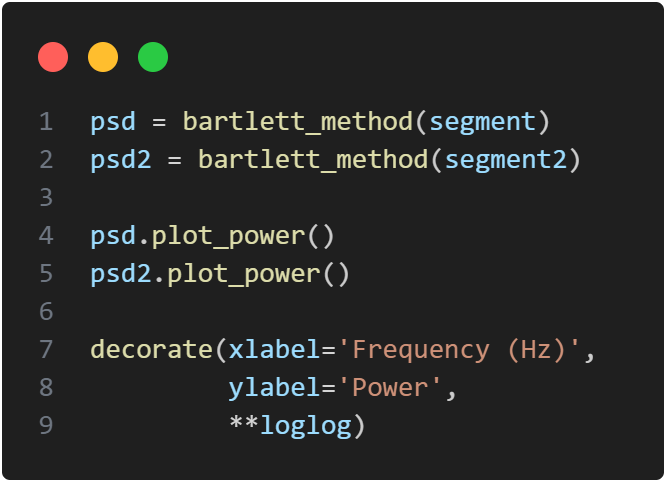
Рис. 4.5. Спектрограмма первого и второго сегментов.

## Упражнение 4.2.

Реализуем метод Бартлетта, который позволит лучше проанализировать спектры мощностей для созданных ранее сегментов:



Воспользуемся созданной функцией:



После выполнения кода получим следующий результат:

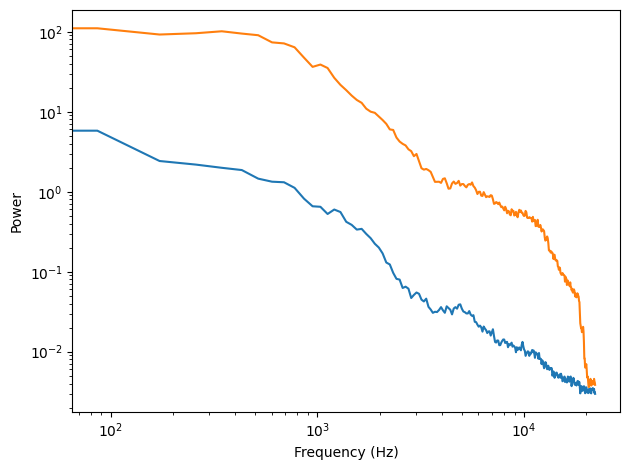
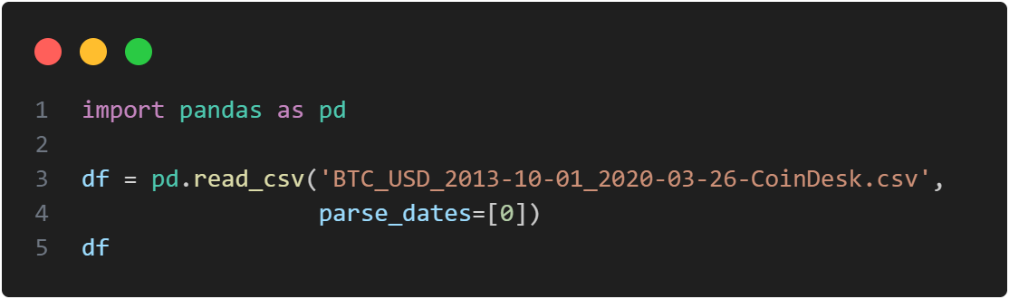


Рис. 4.6. Зависимость мощности от частоты после метода Бартлетта.

Как можно заметить эти два графика не похожи друг на друга, как ожидалось. Как и было сказано ранее, вероятнее всего это связано с тем, что в первом сегменте большую часть занимает лишь накатывающая волна, в отличии от второго.

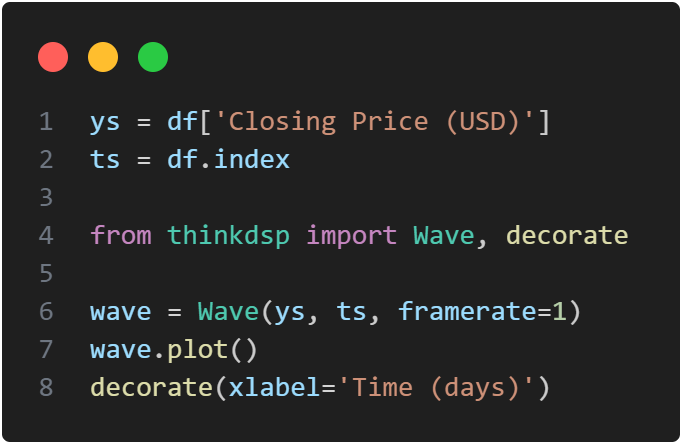
## Упражнение 4.3.

Для упражнения скачаем csv таблицу с данными о ежедневной цене криптовалюты bitcoin (на сайте coindesk кнопки для получения этих данных найдено не было, поэтому csv файл взят из методички) и откроем их в python:



В результате получаем таблицу с необходимыми данными, где есть цена в начале торгов, в конце и наивысшая за этот период.

Построим график закрывающей цены за период, приведенный в таблице:



Мы получаем значение цены биткоина, при закрытии торгов, а также число измерений. Все это передаем Wave, тем самым создав волну. Получившийся рисунок выглядит следующим образом:

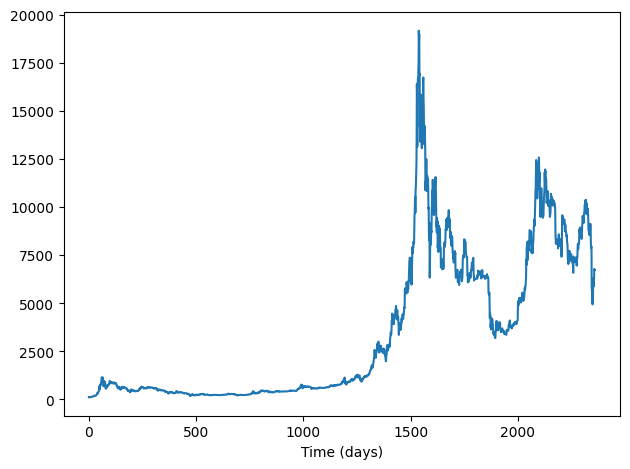
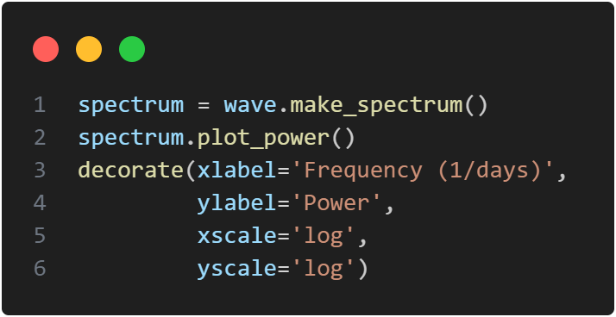


Рис. 4.7. Колебания цены биткоина.

Создадим спектр данного графика и выведем его график мощности на логарифмической шкале:



Получившийся график имеет следующий вид:

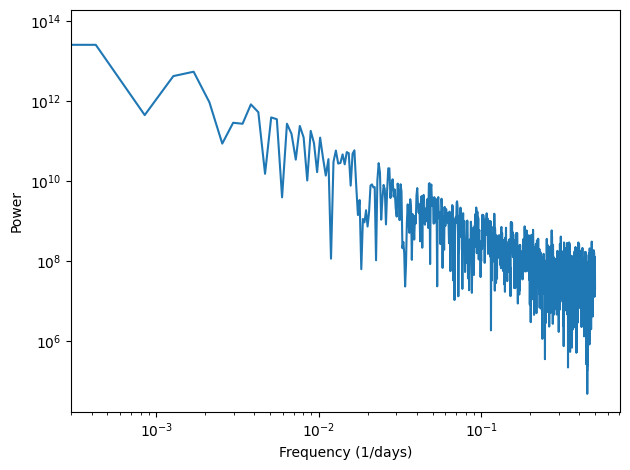
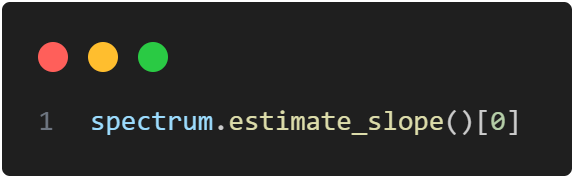


Рис. 4.8. Спектр биткоина.

Данный спектр похож как на красный, так и на розовый. Для ясности выведем значение наклона:



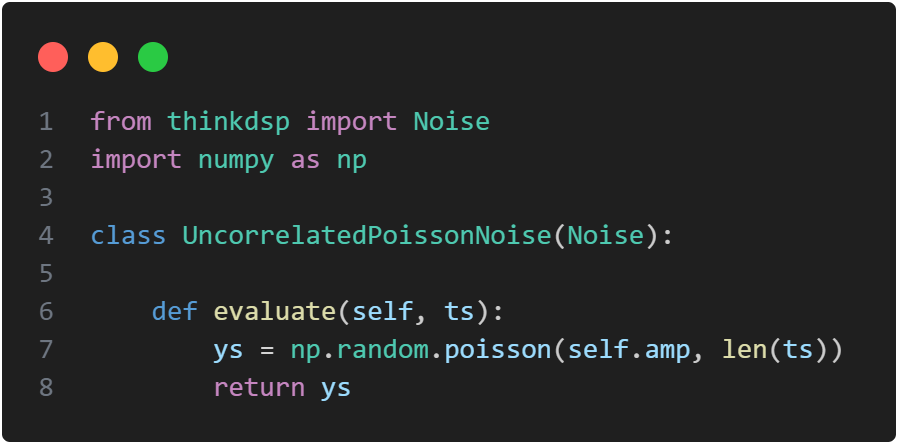
Полученное значение равно:

-1.7332540936758942

Это значение близко к значению красного шума (-2), но все же меньше его. Можно сказать, что этот шум все же является розовым.

## Упражнение 4.4.

Выполним моделирование счетчика Гейгера, используя некоррелированный пуассоновый шум. Для этого создадим новый класс, наследующийся от Noise:

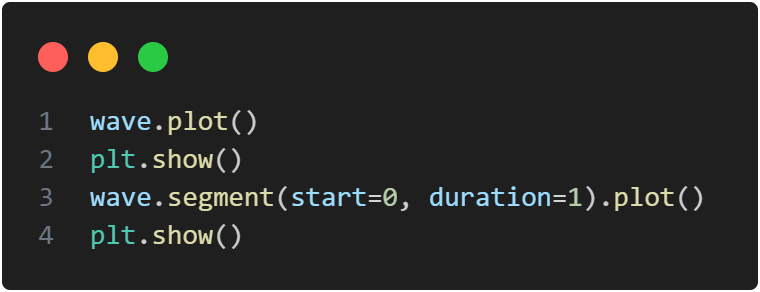


Далее создадим экземпляр этого класса, задав amp = 0.001, а частоту 10 кГц (получится 10 «щелчков» в минуту, что будет сильно похоже на звук счетчика Гейгера):



Данный звук действительно схож с звуком счетчика.

Посмотрим на график этого звука:



Полученные графики выглядят следующим образом:

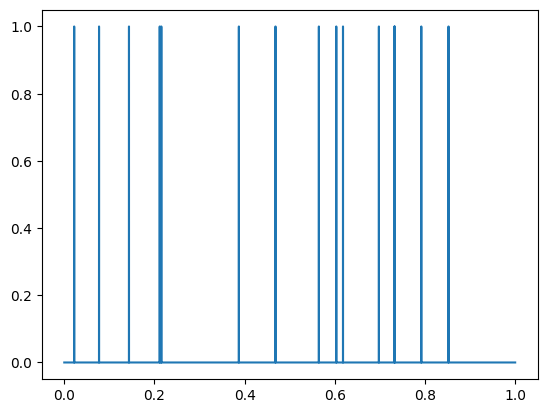
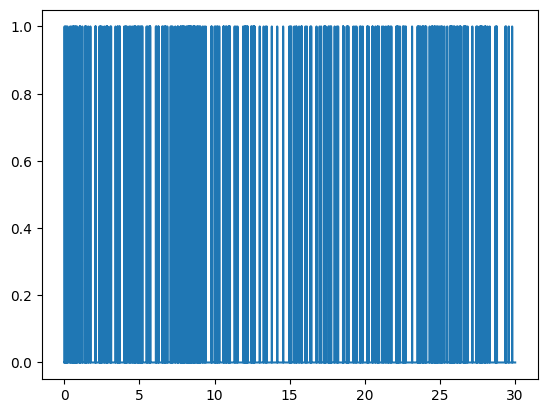
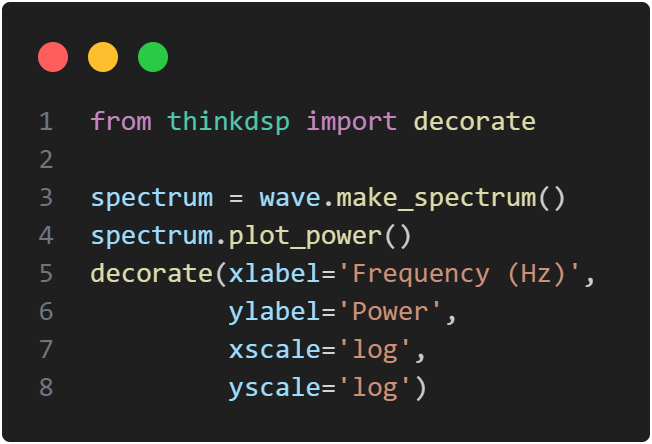


Рис. 4.9. Графики получившегося звука (30 сек и 1 сек).

Посмотрим на спектр мощности получившегося звука в логарифмической шкале:



Полученный график выглядит следующим образом:

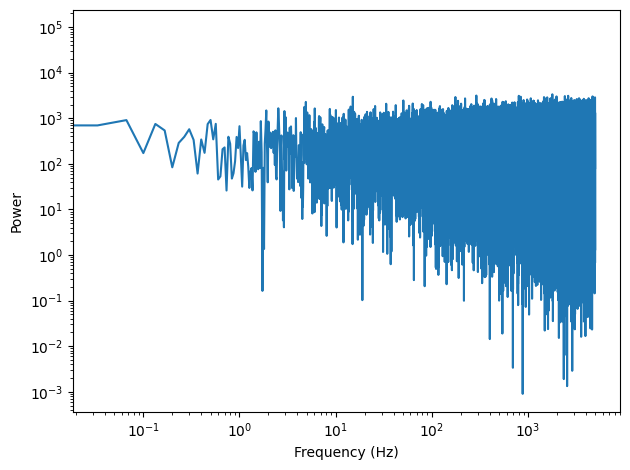
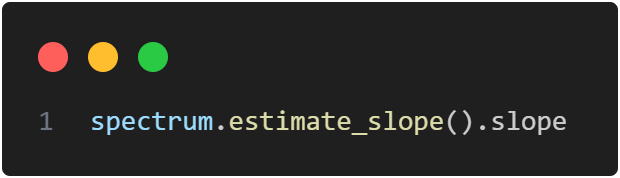


Рис. 4.10. Спектр мощности полученного звука.

Как видно по графику, он сильно похож на график белового шума, убедимся в этом, выведя значение наклона:



Оно равно:

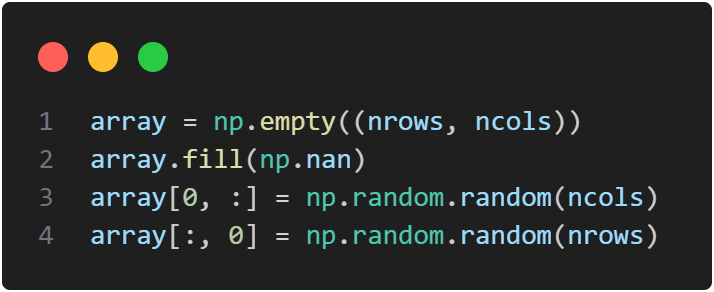
0.00289475039409401

Как можно заметить, полученное число близко к нулю, что свидетельствует о том, что перед нами белый шум.

## Упражнение 4.5.

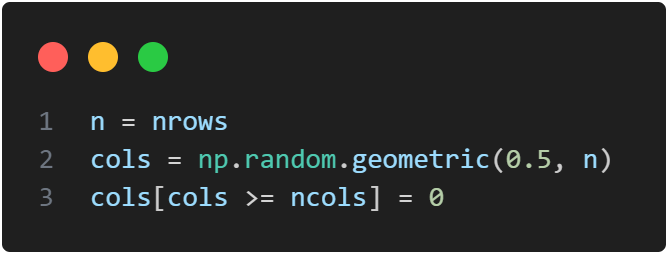
Реализуем на языке python алгоритм Voss-McCartney, который предлагает более эффективный способ генерации розового шума. Его главная идея состоит в суммировании нескольких последовательностей случайных чисел, которые обновляются с разной частотой дискретизации. Первый источник должен обновляться на каждом временном шаге; второй источник - на каждом втором временном шаге, третий источник - на каждом четвертом шаге и так далее. Есть вариант со случайно распределенной частотой обновлений (именно этот вариант и будет реализован нами).

Первым этапом алгоритма будет генерация двумерного массива. Первое измерение – количество точек, для которых будет создан итоговый розовый шум, второе – количество источников для итогового суммирования. Заполним первый столбец и строку случайными числами:

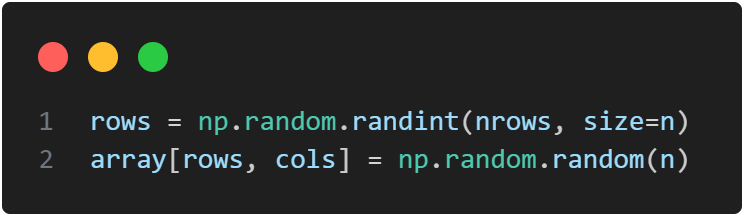


Далее необходимо сгенерировать в каких столбцах произойдут обновления. Для этого очень удобно использовать функцию, генерирующую n-ое число испытаний Бернулли.

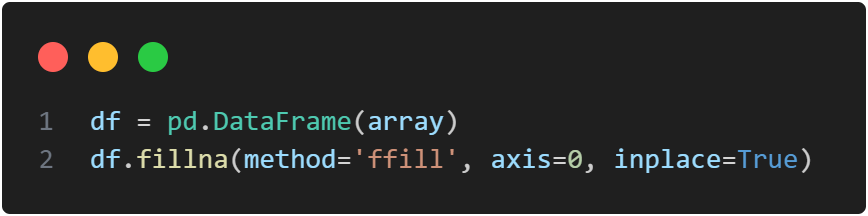
Результаты, которые превышают число столбцов – обнуляем:



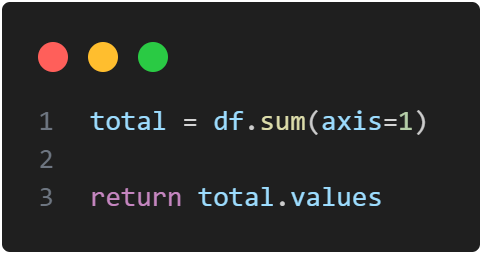
Далее создадим строки, в которых будут соответствующие обновления и обновим значения в соответствующих парах строк-столбцов, записав туда случайное число от 0 до 1:



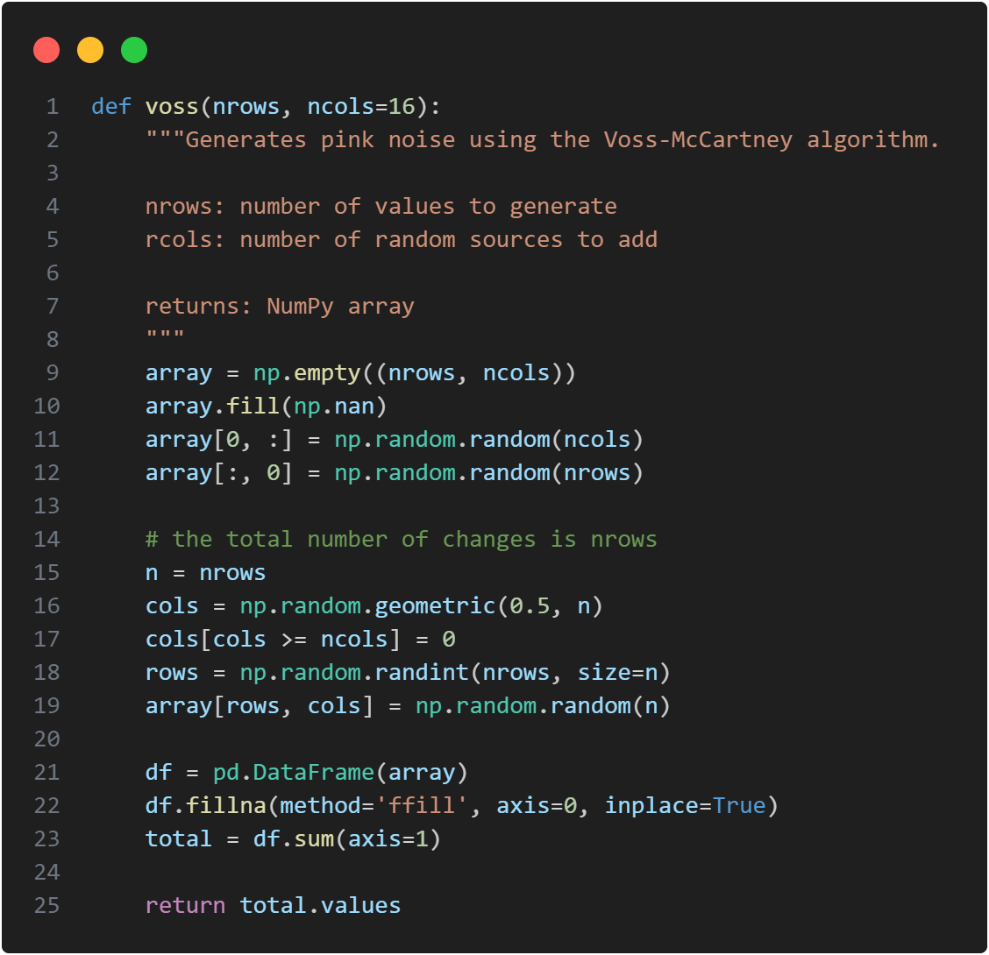
Далее необходимо избавиться от оставшихся nan, записав вместо них предыдущее не nan значение в соответствующем столбце. Для этого воспользуемся библиотекой pandas и функцией fill:



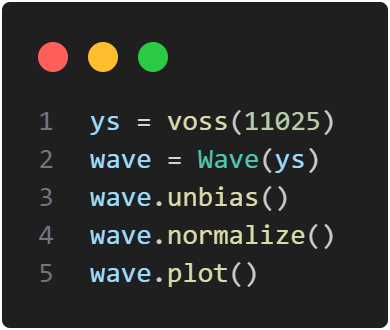
После чего выполним суммирование в соответствующей строке и получим итоговый розовый шум:



Итоговая функция для генерации имеет следующий вид:



Создадим волну, используя эту функцию и проверим, действительно ли получился розовый шум, как требовалось:



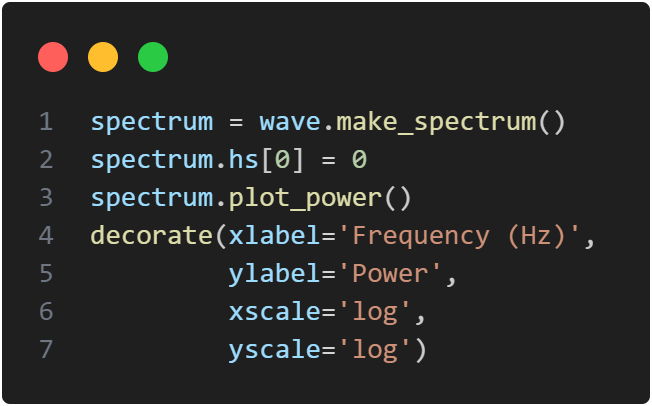
Получился следующий график:

Изображение выглядит как снимок экрана, График, текст, диаграмма

Автоматически созданное описание

Рис. 4.11. График сгенерированного сигнала.

По этому графику видно, что зависимость какая-то присутствует, однако хорошо она не проглядывается. Лучше будет заметно по спектру мощности в логарифмической шкале:



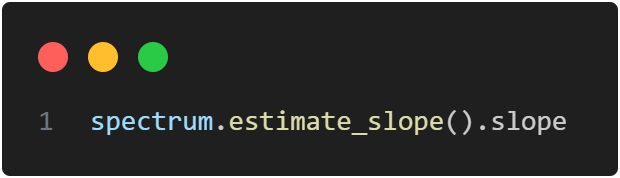
Получившийся спектр выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, График, диаграмма

Автоматически созданное описание

Рис. 4.12. Спектр сгенерированного шума.

Как видим, спектр сильно похож на аналогичный у розового шума, однако, чтоб быть уверенным наверняка, выведем наклон:



Получено следующее значение:

-1.0042795267007751

Это значение очень близко к 1, можно с большой уверенностью сказать, что получившийся шум действительно является розовым, как и требовалось.

# Лабораторная работа 5. Автокорреляция.

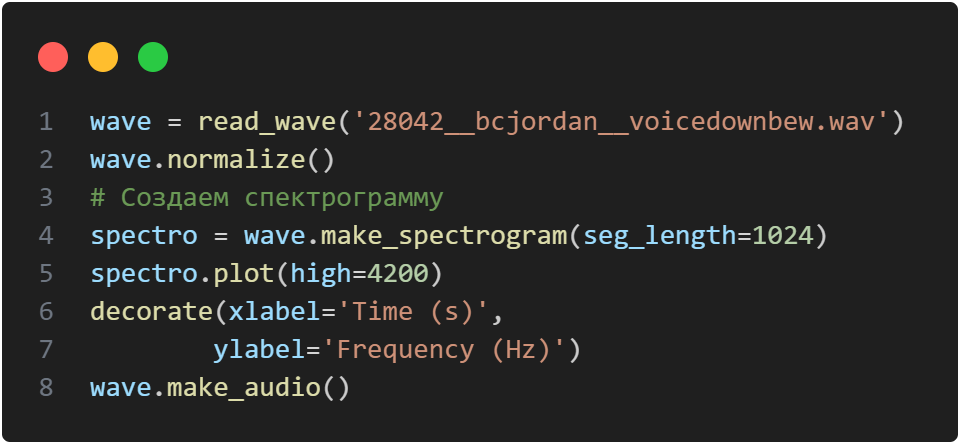
## Упражнение 5.2.

Создадим функцию estimate\_fundamental которая будет принимать волну, начало записи, длину для анализа и промежуток, в котором происходит поиск пиковой частоты:



Эта функция использует функцию autocorr, которую написал автор учебника. На полученных значениях мы выбираем заданный промежуток и получаем максимальное значение, после чего переводим lag в частоту и возвращаем её.

Возьмем запись вокального чирпа и выведем её спектрограмму:



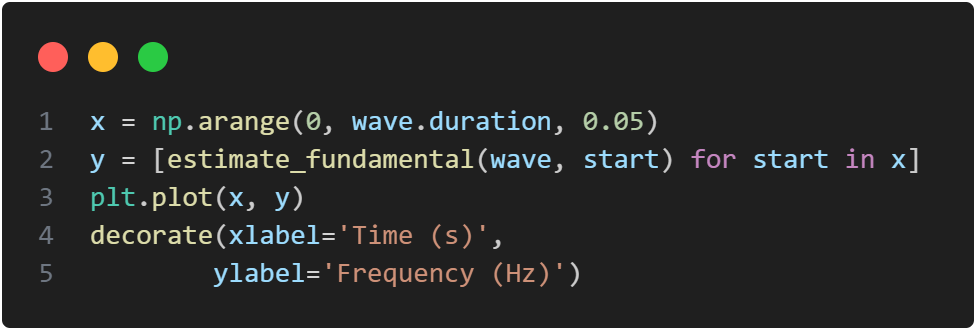
Полученная спектрограмма выглядит следующим образом:

Изображение выглядит как снимок экрана, желтый, текст

Автоматически созданное описание

Рис. 5.1. Спектрограмма вокального чирпа.

Теперь воспользуемся разработанной функцией и создадим график изменения основной частоты этого чирпа:



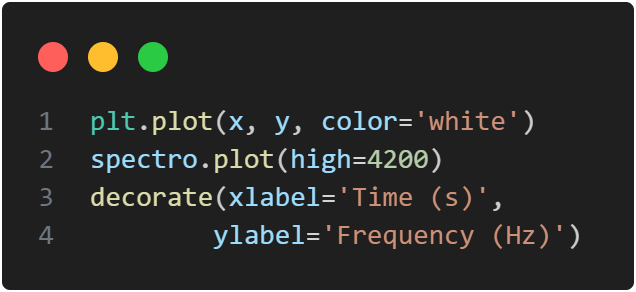
Мы создаем сетку времени. В этих точках вычисляем значение частоты и отрисовываем это на график:

Изображение выглядит как текст, диаграмма, График, линия

Автоматически созданное описание

Рис. 5.2. Изменение частоты вокального чирпа.

Как мы видим, частота чирпа меняется с 530 до 300. Проверим корректность наших расчетов, наложив этот график на спектрограмму, полученную ранее:



Получим следующий результат:

Изображение выглядит как снимок экрана, текст, желтый

Автоматически созданное описание

Рис. 5.3. Спектрограмма вокального чирпа с наложенной частотой.

Как мы видим, график накладывается идеально в центр темной зоны спектрограмма, т. е. на самую высокую частоту, как и ожидалось.

## Упражнение 5.3.

Продолжим анализ цен биткоина, выведем график этих изменений:



Этот график выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, График, линия

Автоматически созданное описание

Рис. 5.4. График изменений цен на биткоин.

В прошлый раз анализ производился, сравнивая изменение цен с шумом, и мы пришли к выводу, что это розовый шум. Повторим анализ, на этот раз используя автокорреляцию:



Получившийся график выглядит таким образом:

Изображение выглядит как диаграмма, График, текст, линия

Автоматически созданное описание

Рис. 5.5. График корреляции цен на биткоин.

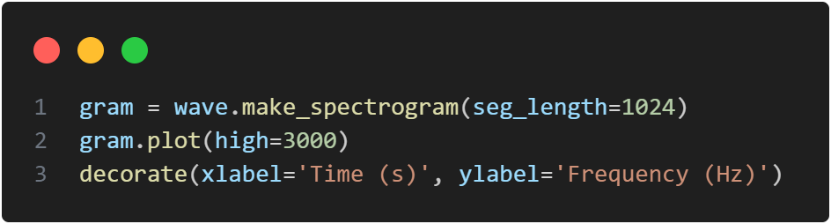
Как мы видим, данный процесс не похож на периодический, а скорее напоминает обычный розовый шум, при близкой к 1.7, что мы и получали в прошлых пунктах.

## Упражнение 5.4.

Выполним анализ звука саксофона для получения информации о явлении, называемом «подавленная основная». Для начала откроем запись саксофона:



Теперь создадим спектрограмму этого звука:



Эта спектрограмма имеет следующий вид:

Изображение выглядит как снимок экрана, желтый, диаграмма, линия

Автоматически созданное описание

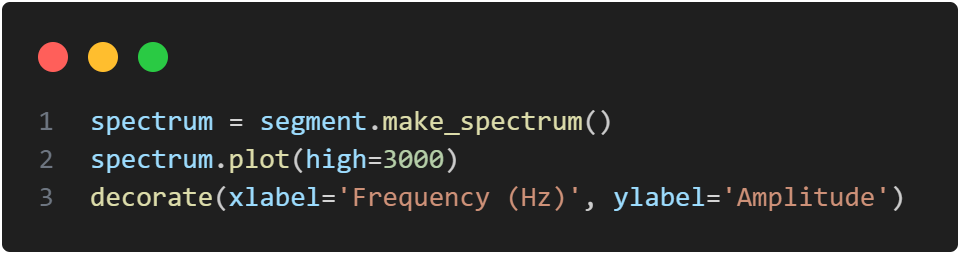
Рис. 5.6. Спектрограмма звука саксофона.

Возьмем отрезок этого звука, например с 4 до 4.5 секунд:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Посмотрим на спектр данного отрывка:



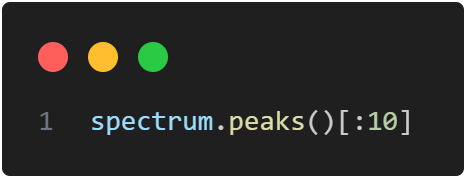
Получившийся график имеет следующий вид:

Изображение выглядит как текст, диаграмма, линия, График

Автоматически созданное описание

Рис. 5.7. Спектр отрезка звука саксофона.

Для удобства выведем значения частот в виде текста:



Результат выглядит следующим образом:

[(3992.340463145426, 1244.0),

(1870.050347074950, 414.0),

(990.3942903418166, 830.0),

(948.6612140134522, 1242.0),

(784.1921786995281, 416.0),

(730.5614131345403, 1658.0),

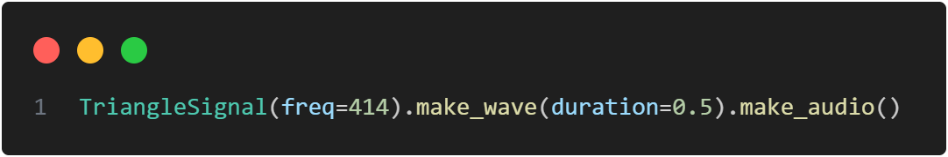
(691.7629429880425, 828.0),

(667.6068421265664, 1246.0),

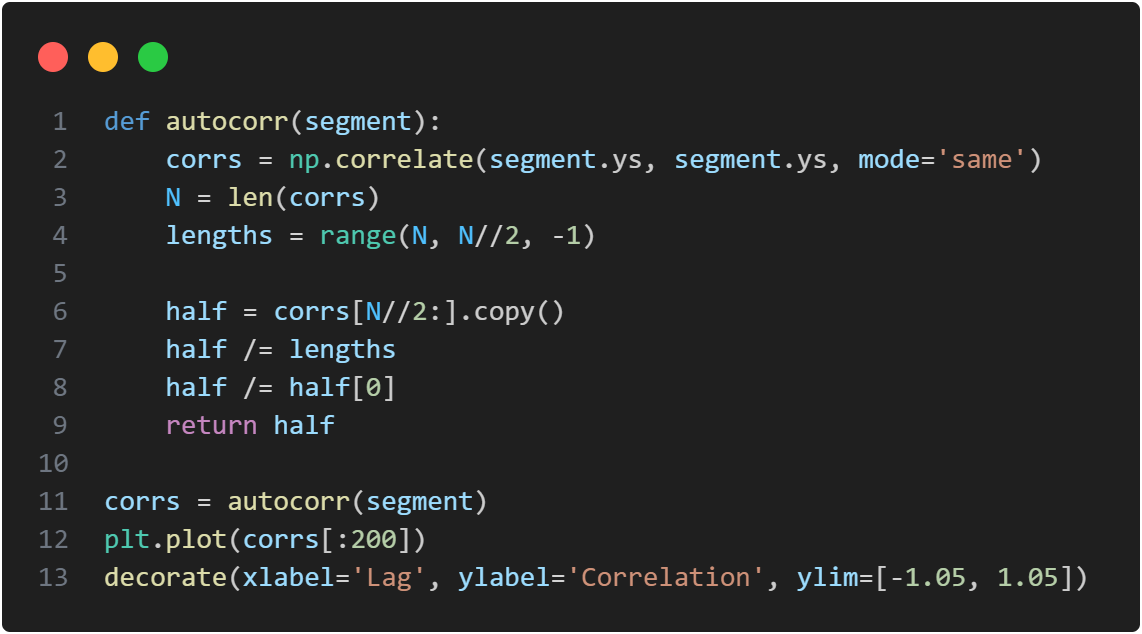
(437.4682504156277, 412.0),

(419.3995359681211, 1240.0)]

Как мы видим, основной пик находится на 1244, однако частота, которую мы слышим – это 414, убедимся в этом, прослушав треугольный сигнал этой частоты:



При прослушивании слышно, что мы слышим именно эту частоту. Чтоб понять, почему это происходи, выведем корреляцию заданного сегмента:



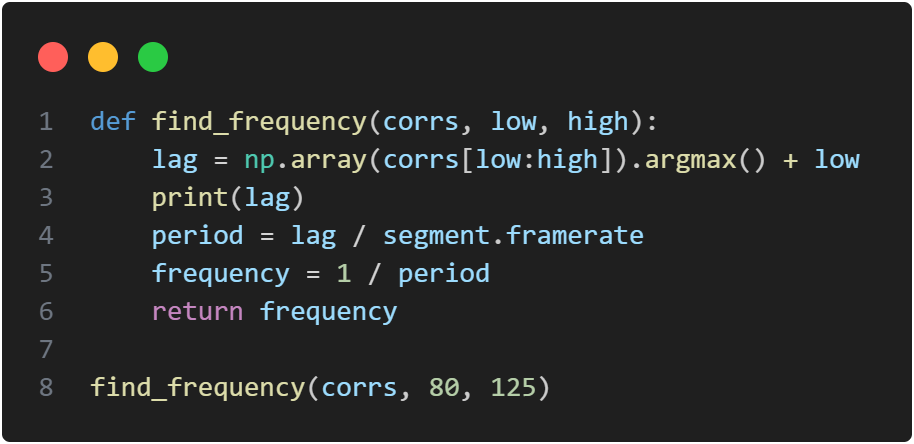
Полученный график выглядит следующим образом:

Изображение выглядит как текст, График, линия, диаграмма

Автоматически созданное описание

Рис. 5.8. График корреляции звукового отрывка.

Как мы видим, пиковая lag находится около 100, для получения точного значения частоты напишем функцию и выполним её:



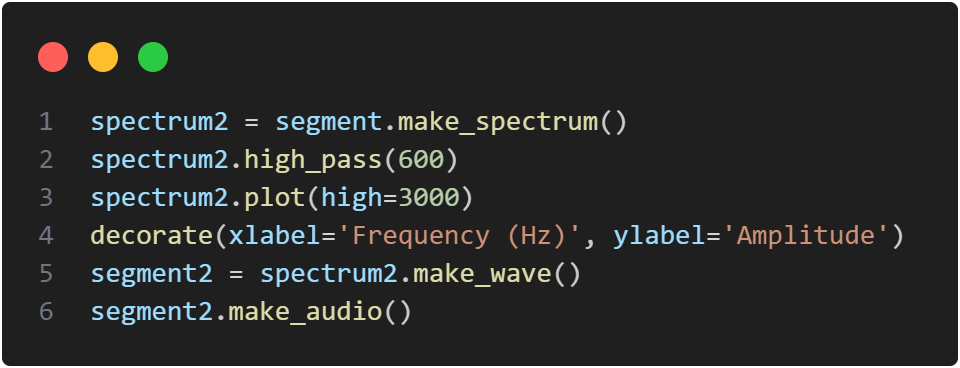
В результате получим следующее значение частоты:

416.0377358490566

Эта частота близка к пику в 414, отличие на 2 Гц не является существенной и может быть результатом погрешности.

Из этого можно сделать вывод, что основная частота, которую мы слышим, зависит не от амплитуды на спектре, а от корреляции.

Теперь попробуем избавиться от этого пика в 414, для этого применим фильтр низких частот:



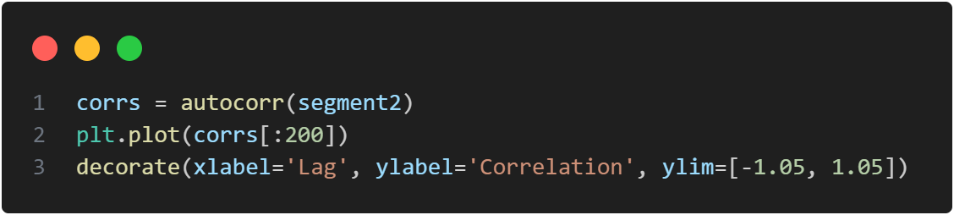
Теперь график амплитуды будет выглядеть следующим образом:

Изображение выглядит как текст, диаграмма, линия, График

Автоматически созданное описание

Рис. 5.9. Спектр отрезка после фильтра низких частот.

После прослушивания становится заметно, что звук стал более высоким, но сам по себе особо не изменился, попробуем понять, с чем это связано, воспользовавшись функцией корреляции:



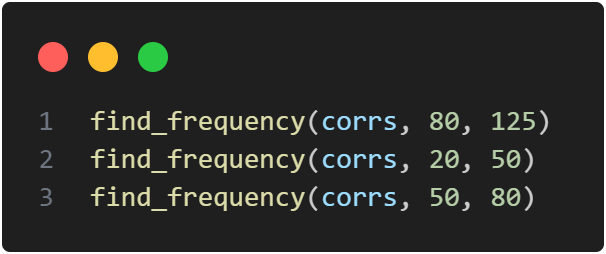
Теперь график корреляции выглядит следующим образом:

Изображение выглядит как текст, линия, График, снимок экрана

Автоматически созданное описание

Рис. 5.10. График корреляции отрывка посоле фильтра низких частот.

Как можно заметить, пик в районе 105 остался. Найдем его точное значение, используя функцию, написанную ранее, а также два предыдущих пика:



Их значения приведены ниже:

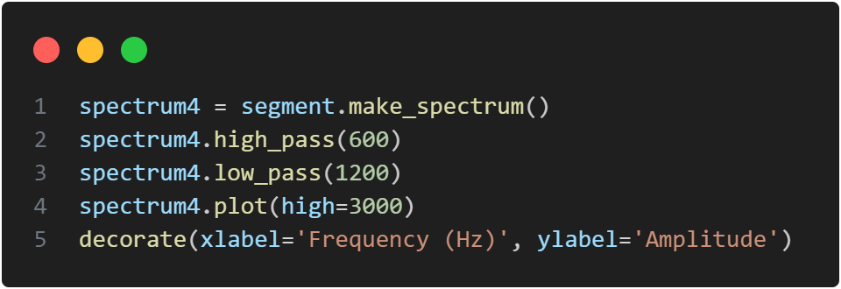
416.03

1225.0

621.12

И несмотря на то, что в спектре мы удалили пик на 414, все еще самая большая частота на графике корреляции находится на 414 т.к. на высоких частотах, преимущественно присутствуют гармоники именно этой частоты, а не 1225 или 621 Гц.

Убедимся, что причина именно в этом, применим как фильтр низких, так и высоких частот:



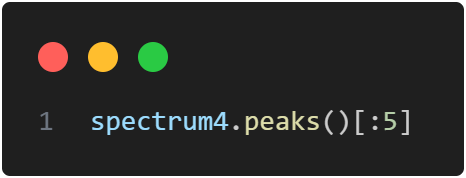
Получившийся спектр имеет следующий вид:

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Рис. 5.11. Спектр звука после фильтра верхних и нижних частот.

Посмотрим значение пика:



[(990.3942903418166, 830.0),

(691.7629429880425, 828.0),

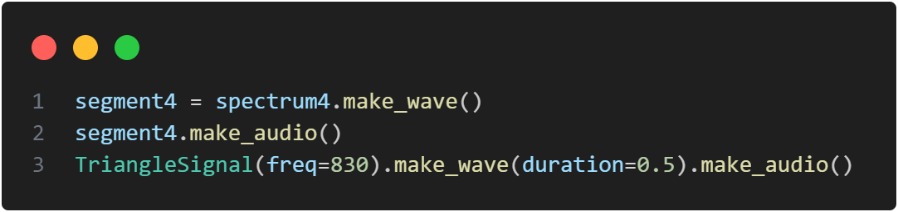
(299.6393588940738, 832.0),

(278.3115019596218, 826.0),

(164.0578985479631, 834.0)]

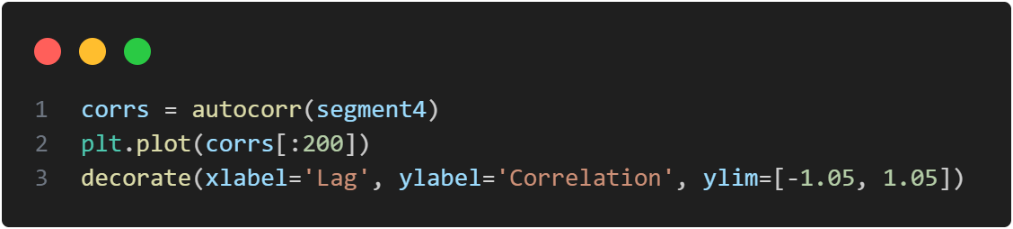
Как можно заметить, пик приходится на частоту 828–830 Гц.

Прослушаем получившуюся запись и сравним с треугольным сигналом на той же частоте:



Мы слышим, что звук действительно сильно изменился и стал больше походить на 830 Гц, нежели на полученные ранее 414 Гц.

Рассмотрим график корреляции полученного отрезка:



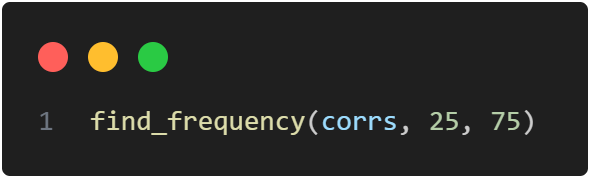
Он выглядит следующим образом:

Изображение выглядит как текст, График, линия, диаграмма

Автоматически созданное описание

Рис. 5.12. График корреляции после фильтра низких и высоких частот.

Посмотрим на значение частоты первого пика:



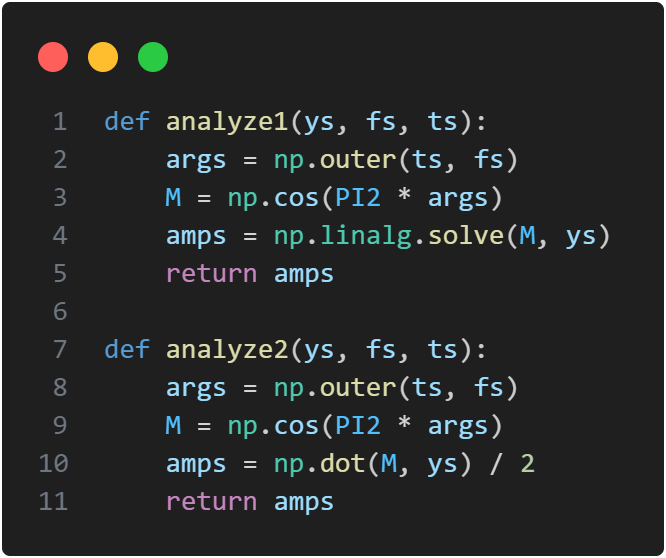
832.0754716981132

Как можно заметить, оно близко к пику, который мы видим на спектре, что подтверждает, что слышимый нами звук является именно 830 Гц.

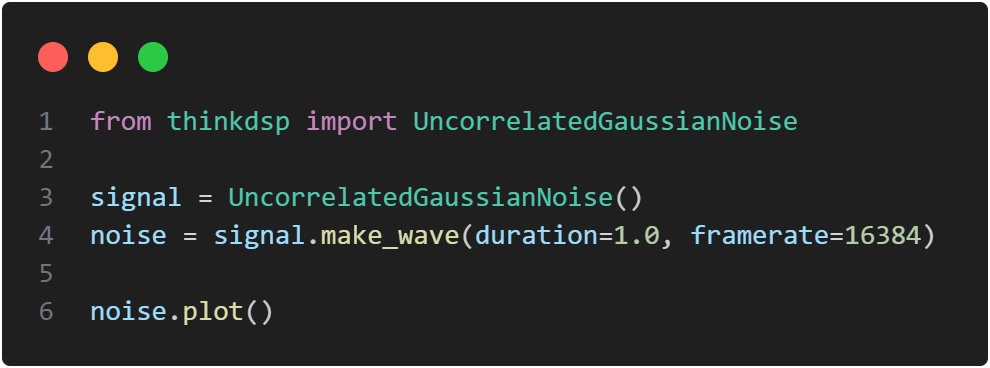
# Лабораторная работа 6. Дискретное косинусное преобразование.

## Упражнение 6.1.

В главе методического материала было приведено 2 функции для получения амплитуд:



Утверждается, что первая функция работает за n3 (из-за np.linalg.solve), а вторая за n2, такая сложность у произведения матриц. Однако проверим, что у алгоритмов именно такая сложность, для этого создадим некоррелированный Гауссов шум (его будем анализировать):



Получившийся шум имеет следующий вид:

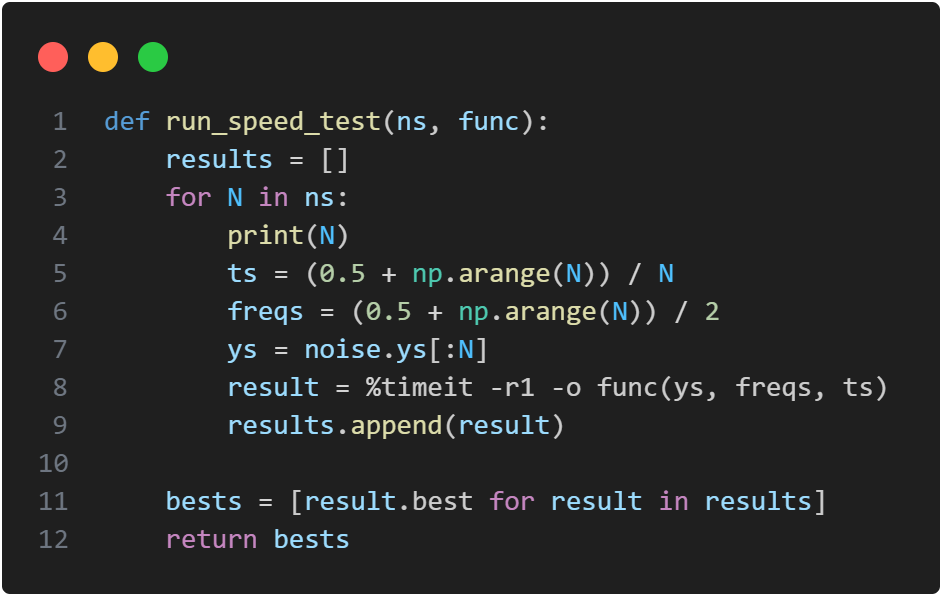
Изображение выглядит как снимок экрана, График

Автоматически созданное описание

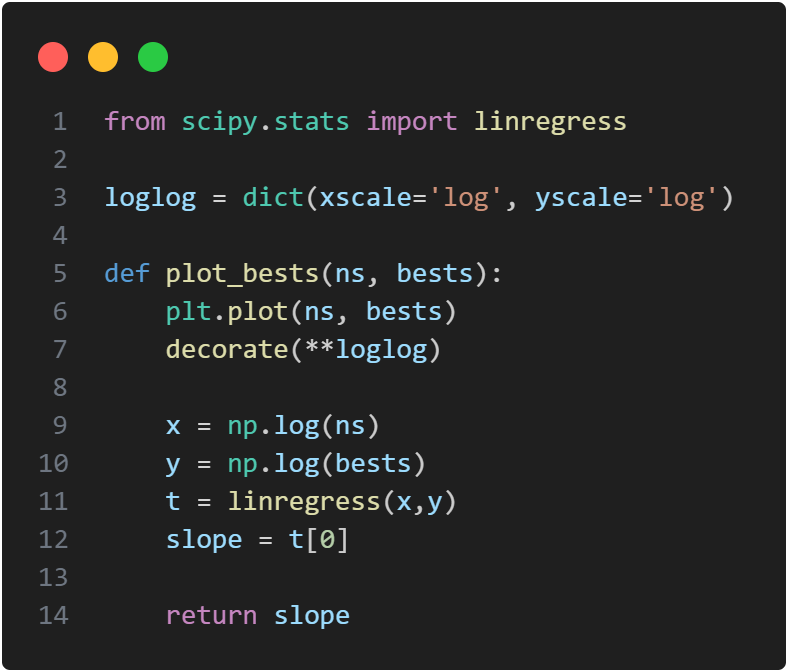
Рис. 6.1. Некоррелированный Гауссов шум.

Хотя, конечно, форма волны никак не будет влиять на расчеты.

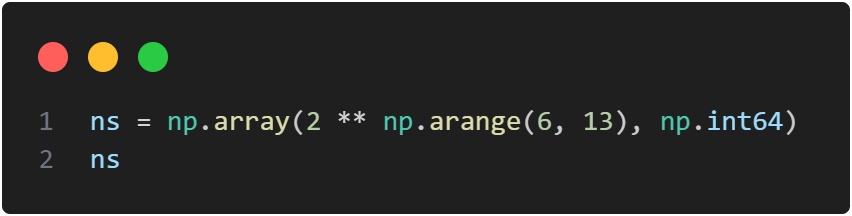
Используя магическую функцию блокнота Jupiter %timeit разработаем функцию, для замеров времени:



Также разработаем функцию для отрисовки результатов в логарифмической шкале, а также возвращающую наклон этой прямой:



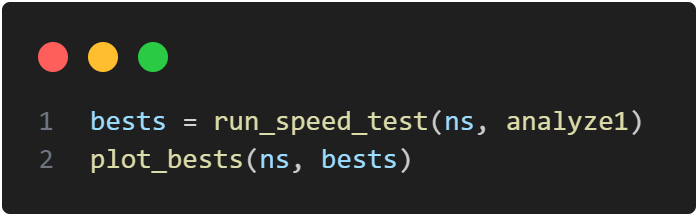
Создадим массив, который будем подавать в качестве количества точек с измерениями:



Его значения выглядят следующим образом:

array([ 64, 128, 256, 512, 1024, 2048, 4096], dtype=int64)

Пора перейти к тестированию. Начнем с функции analyze1:



Получившийся график выглядит следующим образом:

Изображение выглядит как линия, диаграмма, График, снимок экрана

Автоматически созданное описание

Рис. 6.2. Измерения времени работы для analyze1.

Также мы можем посмотреть результаты и в текстовом виде:

64

1.28 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1,000 loops each)

128

5.65 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)

256

7.53 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)

512

18.3 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)

1024

47.1 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)

2048

165 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)

4096

735 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

1.424703446328661

Само время выполнения нас не особо интересует, а вот последнее значение – наклон графика – достаточно интересная величина. Ожидалось, что она будет ближе к 3, однако в ходе наших измерений видно, что она даже меньше двух.

Связано это может быть со множеством факторов, однако вероятнее всего все дело в np.linalg.solve, в котором вполне возможно присутствует ряд оптимизаций, которые ускоряют работу функции на каком-то размере выборки. Вероятнее всего, при увеличении n, мы достигнем ожидаемой сложности, однако это потребует больше ресурсов т.к. даже при 4096 точках, время ожидания составляет порядка секунды.

Теперь выполним анализ функции analyze2:

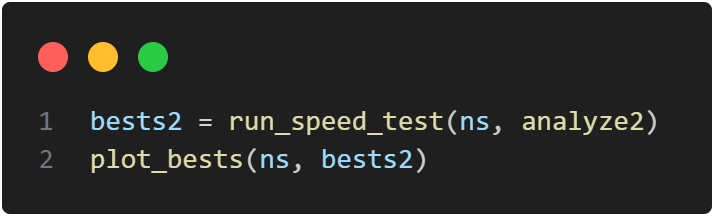


График времени работы этой функции приведен ниже:

Изображение выглядит как линия, текст, График, диаграмма

Автоматически созданное описание

Рис. 6.3. Измерения времени работы для analyze2.

А результаты в текстовом виде приведены ниже:

64

42.2 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 10,000 loops each)

128

174 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 10,000 loops each)

256

524 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 1,000 loops each)

512

3.17 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)

1024

14.6 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)

2048

54.2 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)

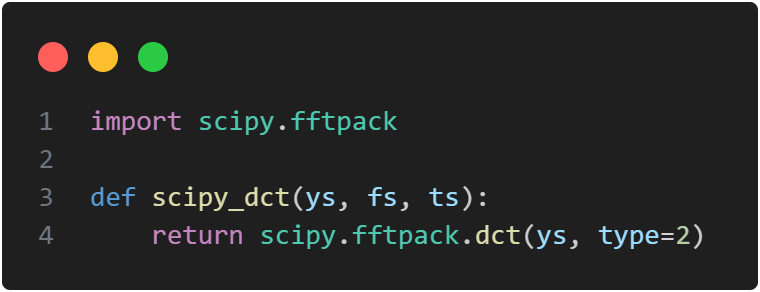
4096

215 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

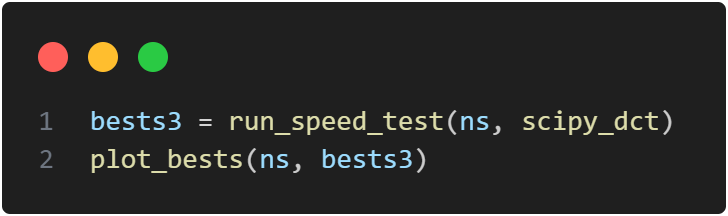
2.082536360735938

Как мы видим, здесь наклон действительно близок к 2, как и ожидалось. Так же очень стоит отметить, что время выполнения значительно ниже, даже несмотря на то, что наклон выше, это также стоит учитывать, однако на это мы сможем обратить внимание позже.

Также повторим эти расчеты с функцией dct из scipy.fftpack:



Выполним измерения, которые выполняли для других функций:



Получившийся график выглядит следующим образом:

Изображение выглядит как линия, диаграмма, График, снимок экрана

Автоматически созданное описание

Рис. 6.4. Измерения времени работы для scipy\_dct.

В текстовом виде значения приведены ниже:

64

4.64 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100,000 loops each)

128

5 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100,000 loops each)

256

5.52 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100,000 loops each)

512

6.46 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100,000 loops each)

1024

8.27 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100,000 loops each)

2048

13.1 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100,000 loops each)

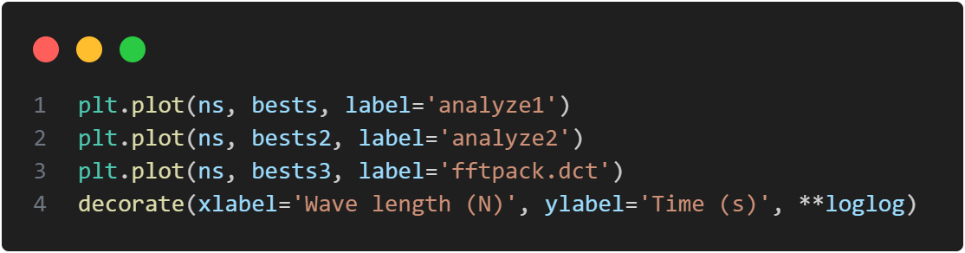
4096

23.6 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 10,000 loops each)

0.37139579643326887

В первую очередь отметим скорость выполнения, которая во много раз меньше, чем у предыдущих функций. Так же обратим внимание на странный угол наклона у нашей функции. Это значение свидетельствует о сложности , о чем можно узнать из документации к методу.

Выведем все три графика на одном листе, чтоб было нагляднее видно разницу в скорости выполнения, а также роста:



Получившийся график приведен ниже:

Изображение выглядит как текст, снимок экрана, линия, График

Автоматически созданное описание

Рис. 6.5. Измерения скорости работы для трех функций.

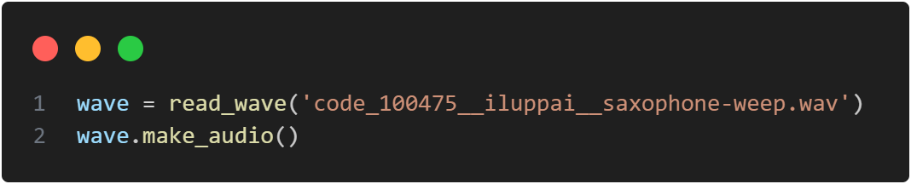
## Упражнение 6.2.

Одно из основных применений ДКП – сжатие звука и изображений. В простейшей форме ДКП при сжатии работает следующим образом:

1. Разбивает длинный сигнал на сегменты.
2. Вычисляет ДКП каждого сегмента.
3. Определяет частотные компоненты с такой амплитудой, что их не слышно, и удаляет их, сохраняя только оставшиеся частоты и амплитуды.
4. При воспроизведении согнала загружает частоты и амплитуды каждого сегмента и применяет обратное ДКП.

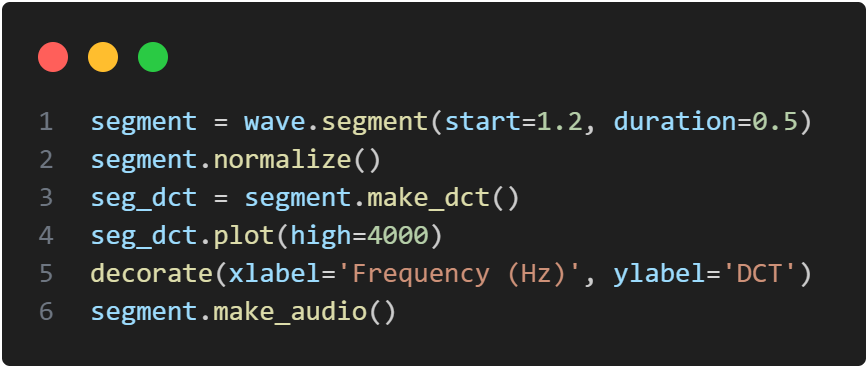
Реализуем этот алгоритм для записи саксофона.

Первым делом прочитаем её:



Для начала выполним этот алгоритм для одного сегмента, а потом напишем функцию, которая будет масштабировать наш подход.

Поэтому получим сегмент нашего произведения, посчитаем её ДКП и выведем на экран:



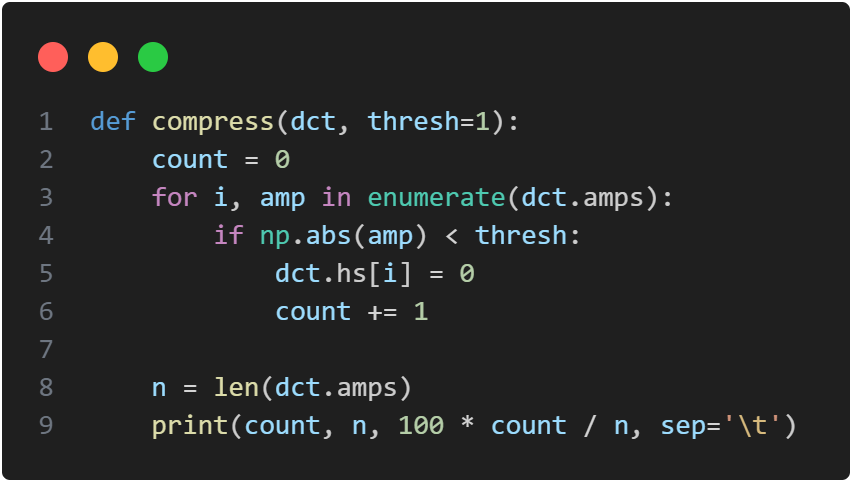
Получившийся график выглядит следующим образом:

Изображение выглядит как текст, диаграмма, линия, График

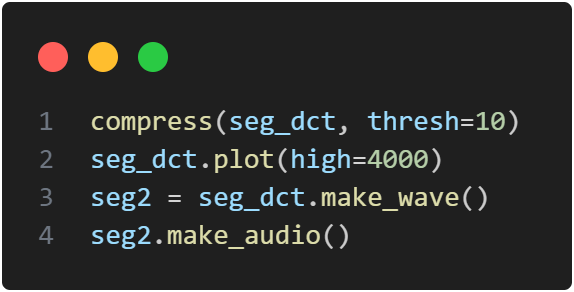
Автоматически созданное описание

Рис. 6.6. ДКП сегмента.

Напишем функцию для третьего пункта, которая будет удалять все частотные компоненты с маленькой амплитудой и удалять их. Также дополнительно она будет выводить сколько значений было удалено и сколько процентов от общего числа:



Выполним эту функцию для нашего отрывка:



В результате получаем следующий график ДКП:

Изображение выглядит как текст, линия, диаграмма, График

Автоматически созданное описание

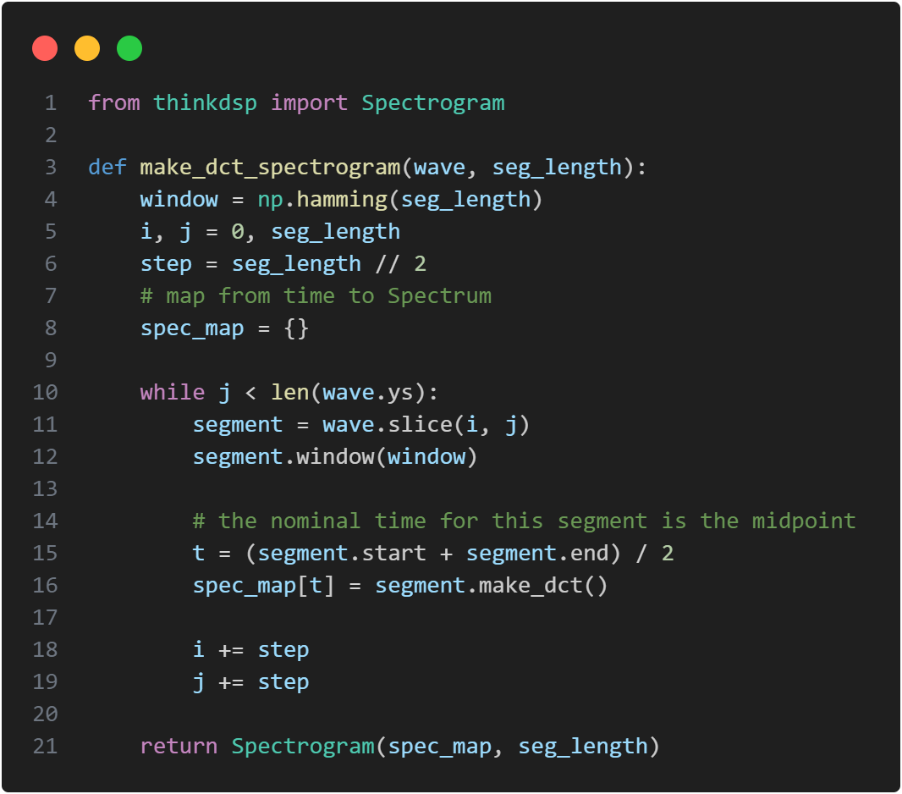
Рис. 6.7. ДКП сегмента после фильтрации.

По графику кажется, что ничего не изменилось (по аудио создается такое же впечатление), однако посмотрим на результат, который вывела наша функция:

20457 22050 92.77551020408163

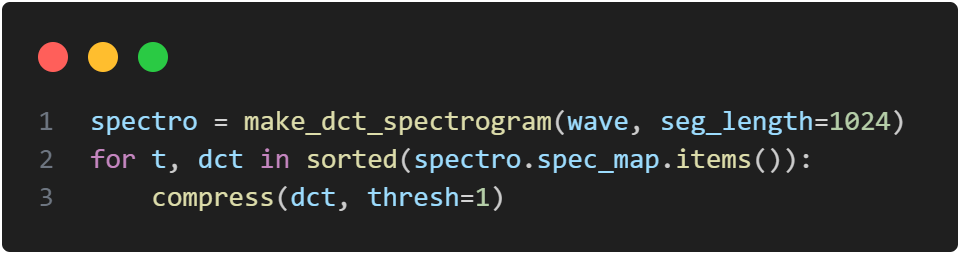
В результате выполнения было удалено 92.8% сегментов. Именно поэтому этот метод является таким эффективным т.к. практически не заметен для уха, однако сильно сохраняет информацию.

Теперь необходимо реализовать алгоритм, который будет работать на сегменты большей длинны. Первым шагом необходимо выполнить разбиение на сегменты с ДКП:

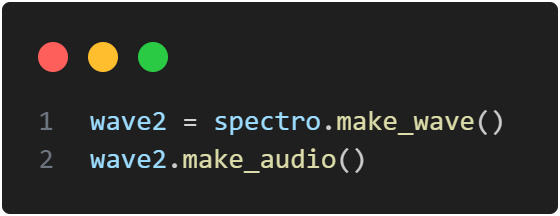


Здесь необходимо отметить необходимость использовать окно Хемминга, которое рассматривалось в предыдущих лабораторных.

После того, как мы разбиваем аудио на сегменты, необходимо применить функцию «сжатия» на каждый из этих сегментов:



Теперь прослушаем получившуюся волну:



Аудио не поменяло звучание, однако было удалено практически 95% значений (судя по логам в ходе вызовов функции), что сильно упрощает хранение этой информации.

В этой функции главное правильно найти баланс между звучанием и удалением информации, например при thresh = 10 разница уже становится сильно заметной, однако к этому моменту удаляется более 98% информации.

## Упражнение 6.3.

# Приложение:

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