Санкт-Петербургский государственный политехнический
университет Петра Великого

**Высшая школа интеллектуальных систем и
суперкомпьютерных технологий**

Лабораторная работа

# Гармоники

Работу выполнил студент
3-го курса, группа 3530901/80201
Солянкин Илья Андреевич

Преподаватель:
Богач Наталья Владимировна

Санкт-Петербург 2021

# Содержание

# Список иллюстраций

# Листинги

# 1 Часть №1: Запуск примеров из `chap02.ipynb`

В первом пункте второй лабораторной работы нам необходимо пройтись по всем программам из файла `chap02.ipynb`.

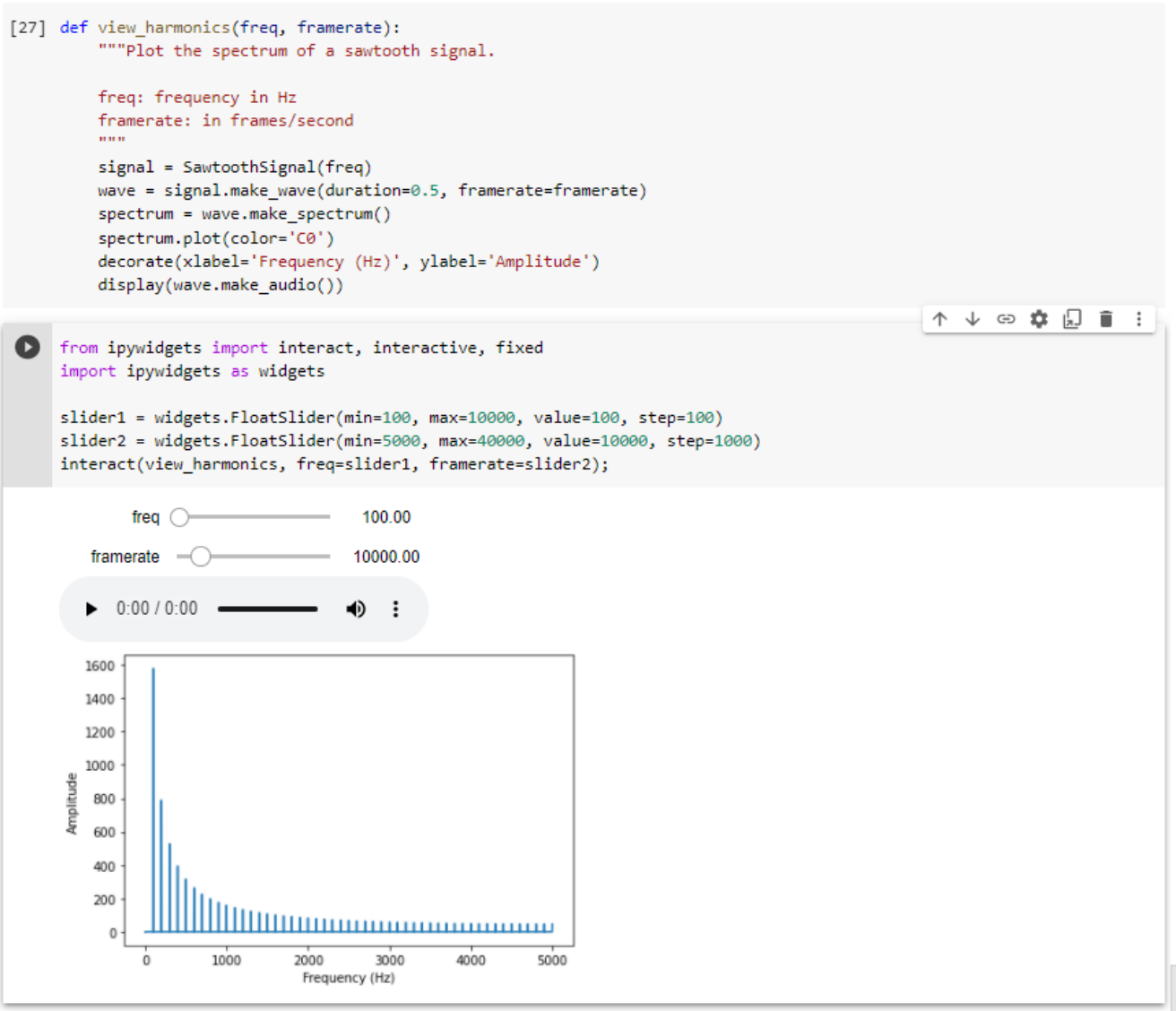


Рис. 1: Проверка, что все работает

В процессе выполнения стало понятно, что все запускается как надо, никаких проблем с запуском не появилось, а значит можно переходить к выполнению следующего пункта.

# 2 Часть №2: Создание SawtoothSignal

Во втором пункте второй лабораторной работы нам необходимо написать класс SawtoothSignal, расширяющий signal и предоставляющий evaluate для оценки пилообразного сигнала. Необходимо так же вычислить спектр пилообразного сигнала.

Начнем с подключения необходимых нам библиотек и написания класса SawtoothSignal

```
from thinkdsp import decorate, Sinusoid, normalize, unbias, TriangleSignal,
SquareSignal, SinSignal, CosSignal, ParabolicSignal

import numpy as np

class SawtoothSignal(Sinusoid):
    def evaluate(self, ts):
        cycles = self.freq * ts + self.offset / np.pi / 2

        frac, _ = np.modf(cycles)
        ys = normalize(unbias(frac), self.amp)
        return ys
```

Листинг 1: Создание класса SawtoothSignal

Теперь проверим, что написанный класс SawtoothSignal работает корректно, для этого обратимся к данному классу и преобразуем все в волну, после чего выведем на экран.

```
test_saw = SawtoothSignal()
test_wave = test_saw.make_wave(test_saw.period * 5, framerate=40000)
test_wave.plot()
decorate(xlabel='Time (s)')
```

Листинг 2: Тестирование класса SawtoothSignal

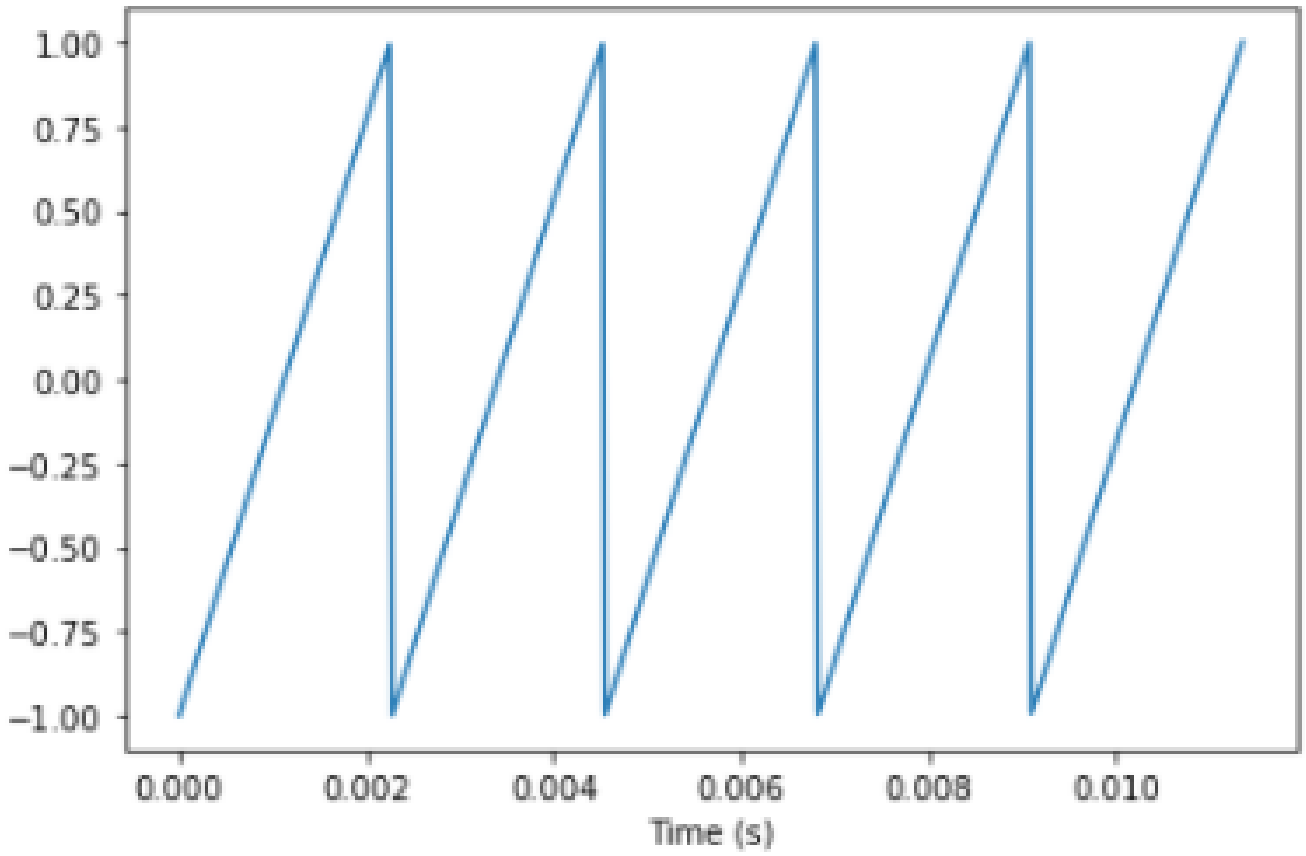


Рис. 2: Проверка написанного класса `SawtoothSignal`

После этого получим аудио дорожку, обращаясь к классу:

```
B [3]: sawtooth = SawtoothSignal().make_wave(duration=0.5, framerate=40000)
       sawtooth.make_audio()
```

Out[3]: ▶ 0:00 / 0:00

Рис. 3: Получение аудиодорожки из написанного класса `SawtoothSignal`

Теперь построим спектр:

```
1  sawtooth.make_spectrum().plot()
2  decorate(xlabel='Frequency (Hz)')
```

Листинг 3: Построение спектра

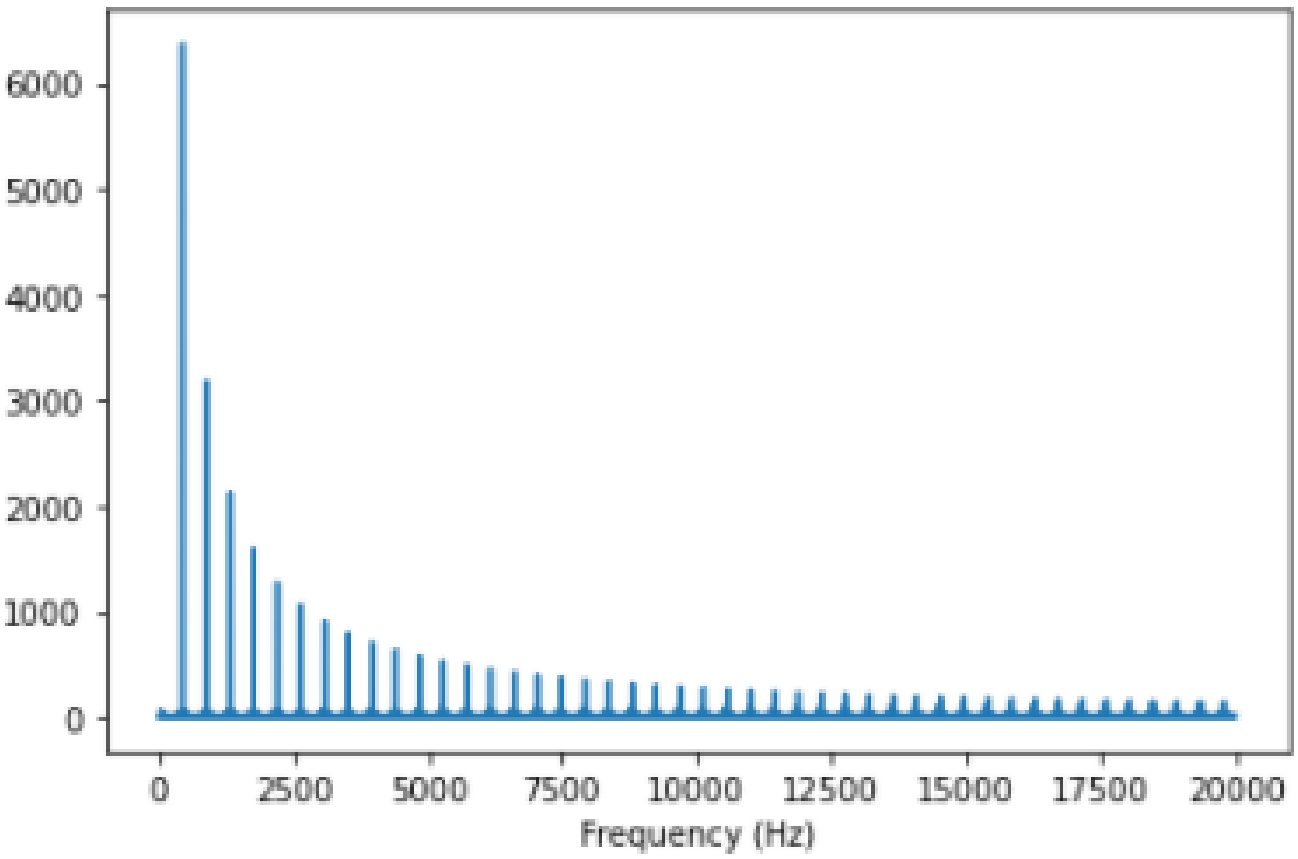


Рис. 4: Построение спектра

Также сравним наш пилообразный сигнал сначала с прямоугольной волной:

```
sawtooth.make_spectrum().plot(color='gray')
square = SquareSignal(amp=0.5).make_wave(duration=0.5, framerate=40000)
square.make_spectrum().plot()
decorate(xlabel='Frequency (Hz)')
```

Листинг 4: Построение прямоугольной волны

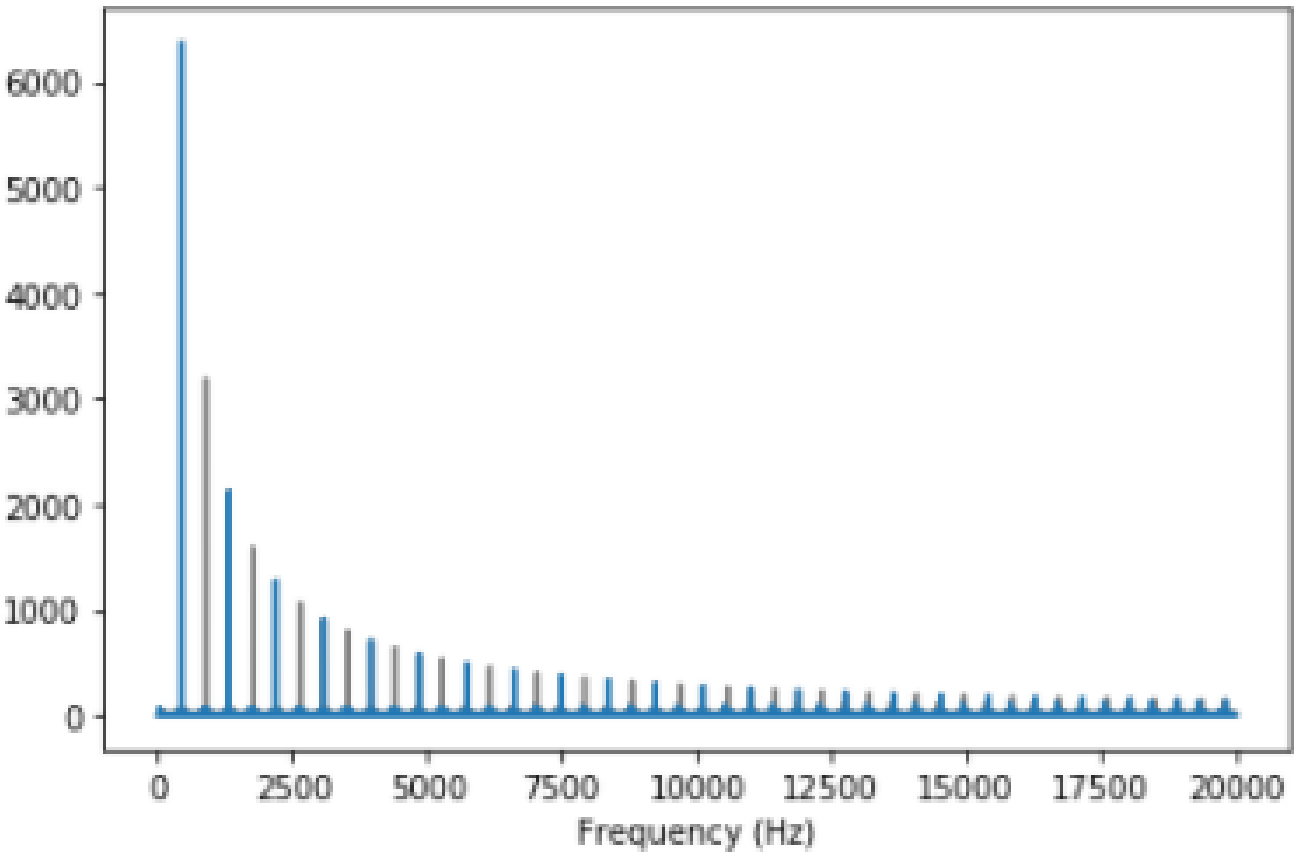


Рис. 5: Построение прямоугольной волны

Также для сравнения получим аудио из прямоугольного сигнала:

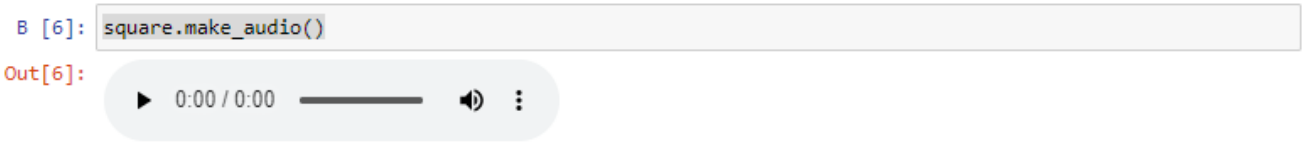


Рис. 6: Получение аудио из прямоугольного сигнала

Теперь сравним наш пилообразный сигнал с треугольным сигналом:

```
sawtooth.make_spectrum().plot(color='gray')
triangle = TriangleSignal(amp=0.79).make_wave(duration=0.5, framerate=40000)
triangle.make_spectrum().plot()
decorate(xlabel='Frequency (Hz)')
```

Листинг 5: Построение треугольной волны

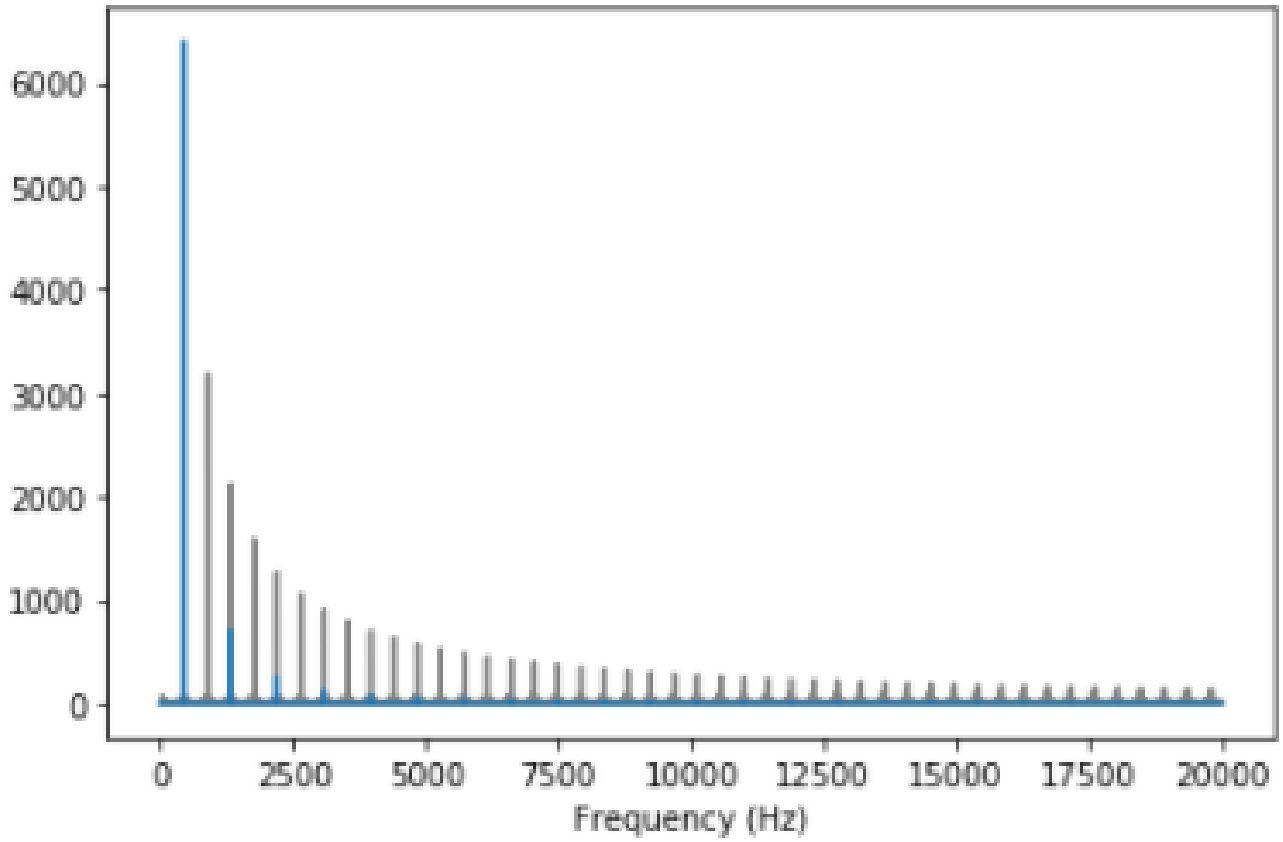


Рис. 7: Построение треугольгой волны

Также для сравнения получим аудио из треугольного сигнала:

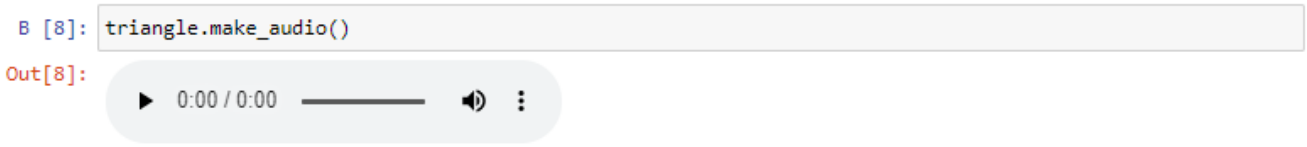


Рис. 8: Получение аудио из треугольного сигнала

В результате выполения данного пункта можно сделать вывод о том, что в нашем пилообразном сигнале амплитуды частот уменьшабтся пропорционально самой частоте, но не ее квадрату. Кроме того при сравнении аудио полученных прямоугольных и треугольных сигналов можно сказать, что квадратный звучит намного реже пилообразного, в то время как треугольный звучит тише.

# 3 Часть №3: Квадратный сигнал

В третьей части второй лабораторной работы нам необходимо создать реугольный сигнал `1100 Hz` и вычислить `wave` с выборками 10000 кадров в секунду. ТАкже необходимо построить спектр и убедиться, что большинство гармоник ”завернуты”из-за биений.

Для начала создадим прямоугольный сигнал:

```
signal = SquareSignal(1100)
wave = signal.make_wave(duration=0.5, framerate=10000)
```

Листинг 6: Построение прямоугольного сигнала

После этого посмтроим спектр для этого сигнала:

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

Листинг 7: Построение спектра

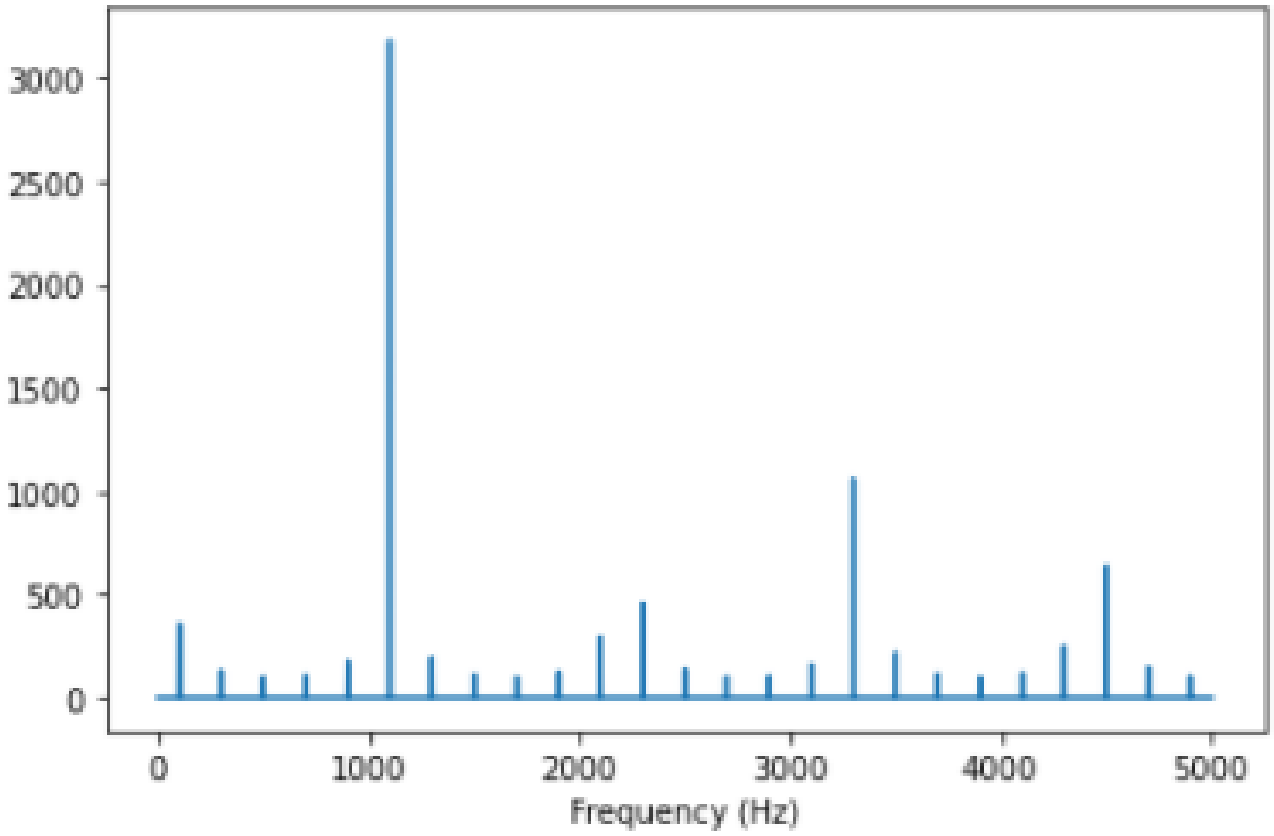


Рис. 9: Построеный спектр

Также создадим из данного сигнала аудиодорожку:

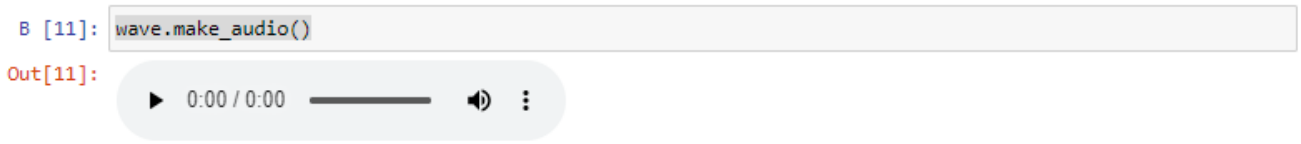


Рис. 10: Аудио из прямоугольного сигала

Для сравнения, чтобы понять, что мы можем услышать «aliased-частоты» создадим еще один сигнал, в нашем случаем синусоидальный

```
signal2 = SinSignal(1000)
wave2 = signal2.make_wave(duration=0.5, framerate=5000)
```

Листинг 8: Построение синусоидального сигнала

Теперь так же создадим для него спектр:

```
spectr2 = wave2.make_spectrum()
spectr2.plot()
decorate(xlabel='Frequency (Hz)')
```

Листинг 9: Построение спектра

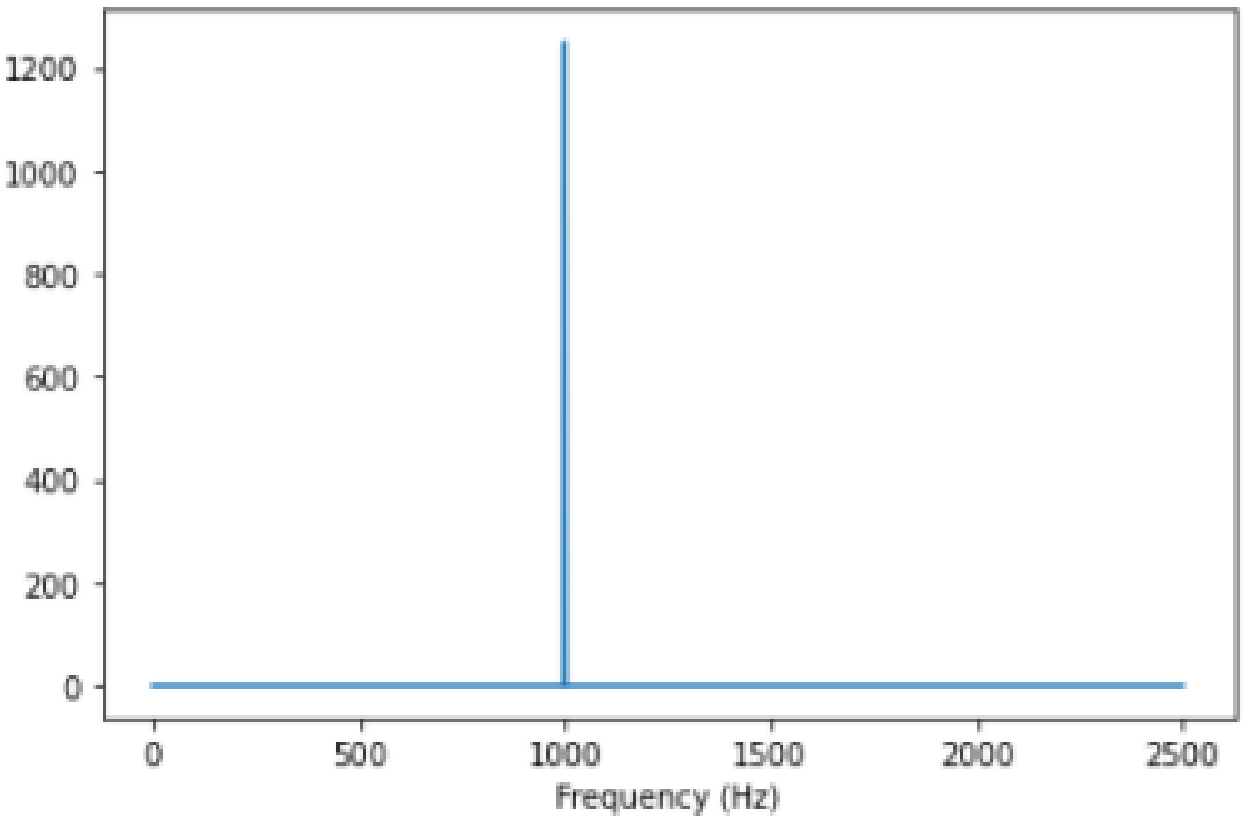


Рис. 11: Построеный спектр

Также создадим из данного сигнала аудиодорожку:

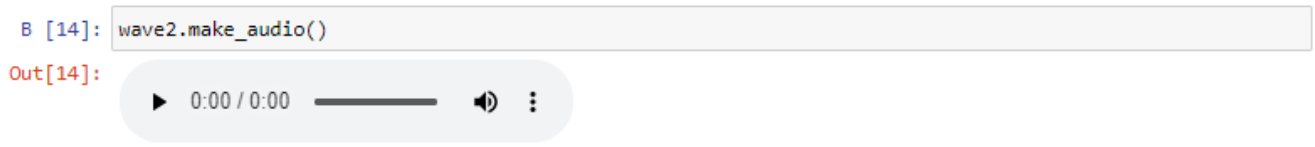


Рис. 12: Аудио из синусоидального сигала

Наконец, в результате выполнения данного поункта мы можем сравнить эти 2 сигнала. На слух они очень сильно отличаются друг от друга, первый сигнал очень сильно режет слух и присутствует некая пульсация. Т.к. эти два аудио слышатся по разному, значить мы можем расслышать «aliased-частоты»

# 4 Часть №4: Нулевая частота

В четвертой части второй лабораторной работы нам необходимо взять объект `spectrum` и распечатать несколько первых значений `spectrum.hs`. Необходимо также убедиться, что они начинаются с нуля, т.е. `spectrum.hs[0]` - амплитуда компоненты с частотой 0.

1. Создать треугольный сигнал с частотой `440 Hz` и `wave` длительностью 0.01 с. после чего распечатать сигнал.
2. Создать объект `spectrum` и распечатать `spectrum.hs[0]`. Опреелить какая амплитуда и фаза у компонента.
3. Усталовить `spectrum.hs[0]` = 100 и определить как данная операция влияет на сигнал.

Для начала создадим треугольный сигнал согласно заданию:

```
triangle = TriangleSignal(440)
wave = triangle.make_wave(duration=0.01)
wave.plot()
decorate(xlabel='Time (s)')
```

Листинг 10: Построение треугольного сигнала

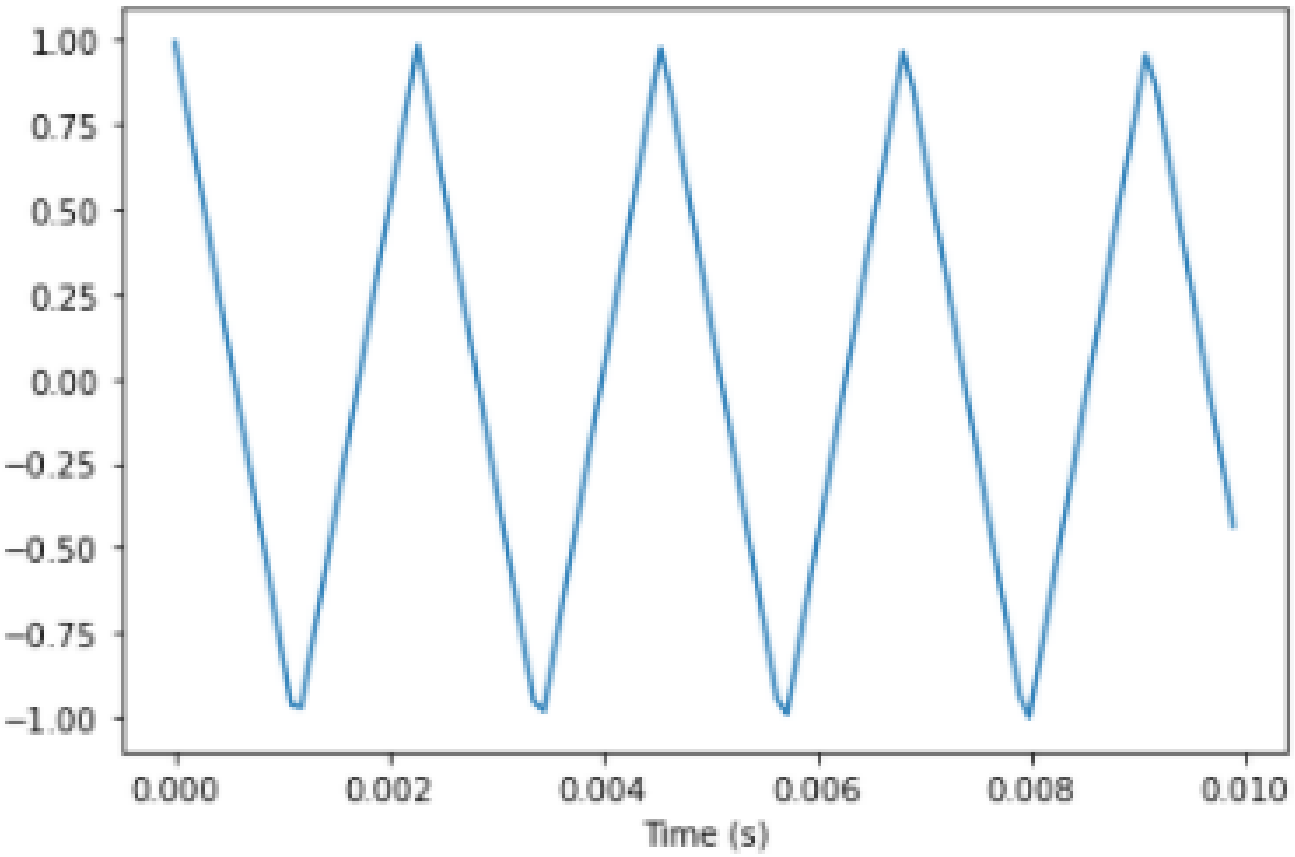


Рис. 13: Построеный треугольный сигнал

После этого создадим спектр для данного сигнала:

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

Листинг 11: Спектр треугольного сигнала

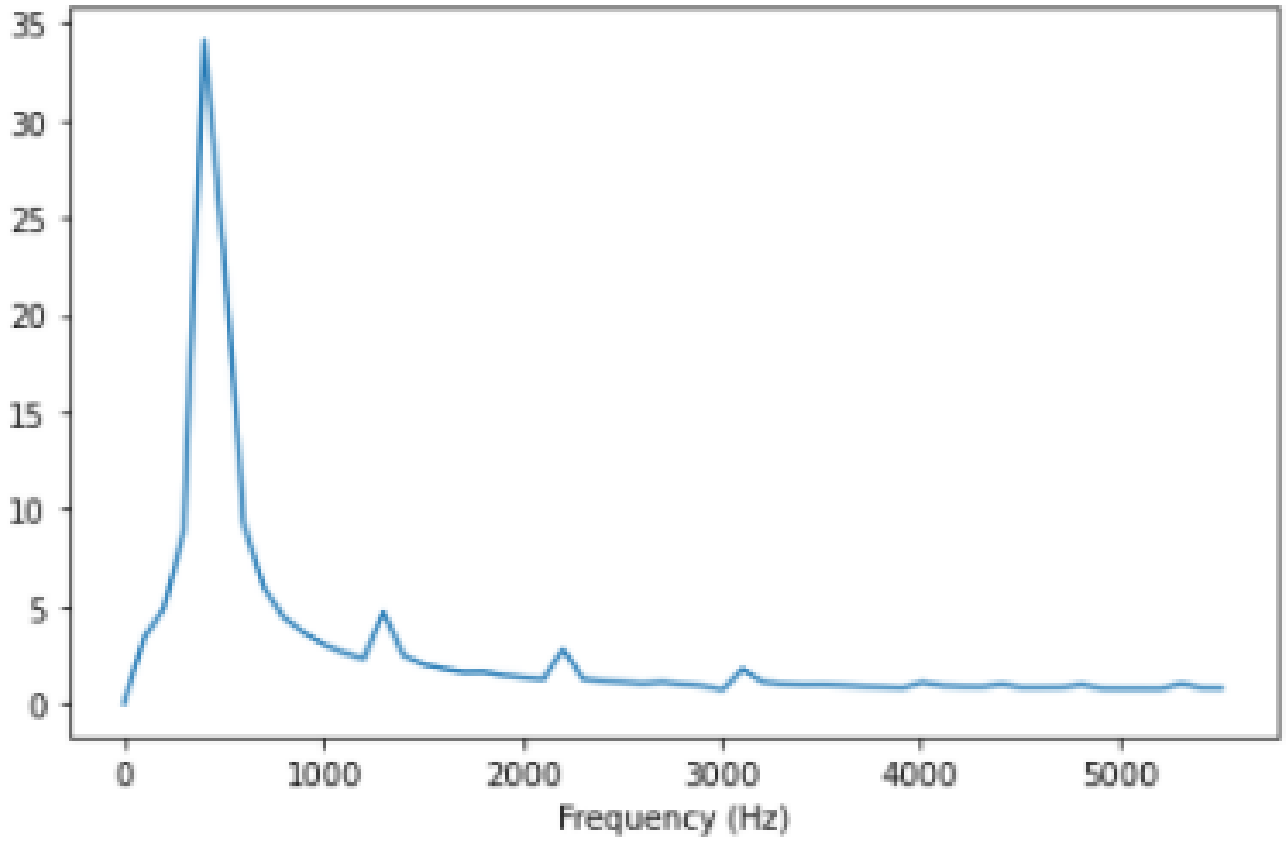


Рис. 14: Спектр треугольного сигнала

Теперь посмотрим на `spectrum.hs[0]`

```
In [17]: spectrum.hs[0]
Out[17]: (1.0436096431476471e-14+0j)
```

Рис. 15: `spectrum.hs[0]`

Как можно увидеть, `spectrum.hs[0]` равен `1.0436096431476471e-14+0j`. ”Длина”данного числа описывает амплитуту нуе=левой компоненты разложения, а угол с `Re` описывает частоту.

Наконец, установим `spectrum.hs[0]` = 100 и выведем полученный сигнал выведем на экран

```
spectrum.hs[0] = 100
wave.plot(color='gray')
spectrum.make_wave().plot()
decorate(xlabel='Time (s)')
```

Листинг 12: `spectrum.hs[0]`

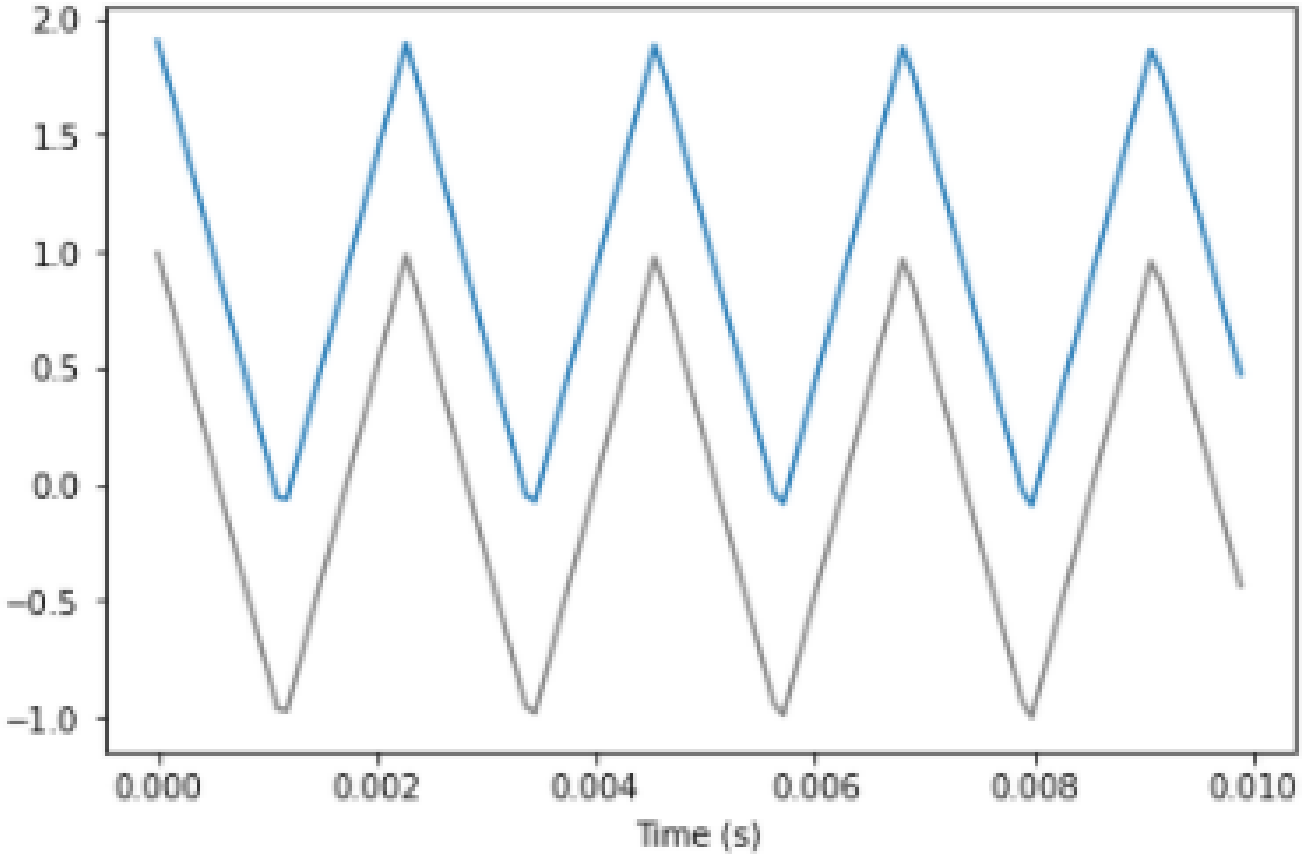


Рис. 16: `spectrum.hs[0]` = 100

Как пожно видеть в результате выполнения данного пункта, разницы между полученными сигналами нет совершенно никакой.

# 5 Часть №5: Деление амплитуды на частоту

В пятом пункте второй лабораторной работы нам необходимо написать функцию, приннимающую `spectrum` как параметр и изменяющую его делением каждого элемента `hs` на соответствующую частоту из `fs`. Необходимо также проверить данную функцию на прямоугольном, треугольном или пилообразном сигнале.

1. Вычислить `spectrum` и распечатать его
2. Изменить `spectrum`, использую нашу функцию, и распечатать его.
3. Использовать `spectrum.make-wave`, чтобы сделать `wave` из измененного `spectrum` и просшулать его. Как изменился сигнал?

Для начала напишем нашу функцию `filter-spectrum`:

```
def filter_spectrum(spectrum):
spectrum.hs[1:] /= spectrum.fs[1:]
spectrum.hs[0] = 0
```

Листинг 13: filter-spectrum

После этого создадим прямоугольный сигнал и сразу переведем его в аудио

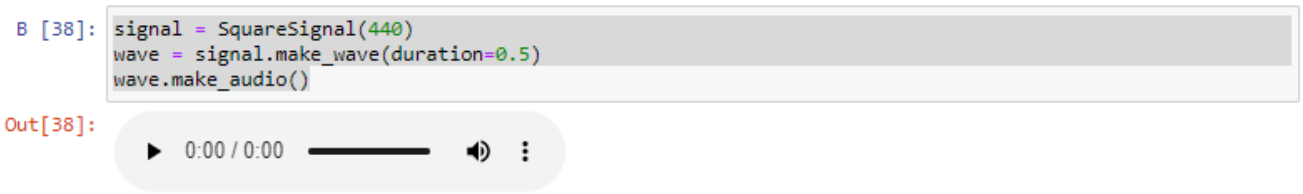


Рис. 17: Прямоугольный сигнал

После этого сразу выведем на экран спектр исходного сигнала, после чего отдадим его на вход нашуй функции. После этого так же выведем полученный спектр на экран:

```
spectrum = wave.make_spectrum()
spectrum.plot(high=10000, color='gray')
filter_spectrum(spectrum)
spectrum.scale(440)
spectrum.plot(high=10000)
decorate(xlabel='Frequency (Hz)')
```

Листинг 14: Работа с сигналами

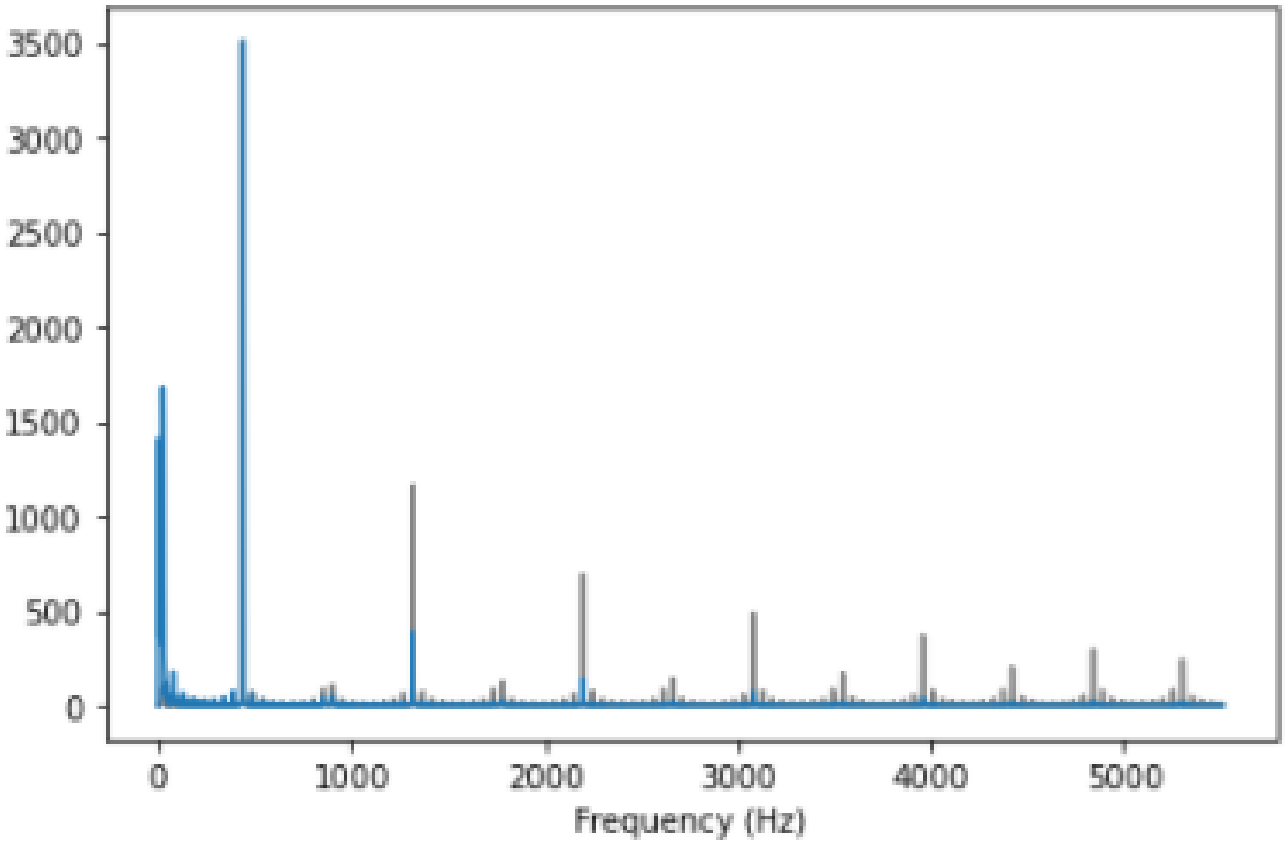


Рис. 18: Вывод спектров на экран

Теперь так же переведем полученный сигнал в аудио:

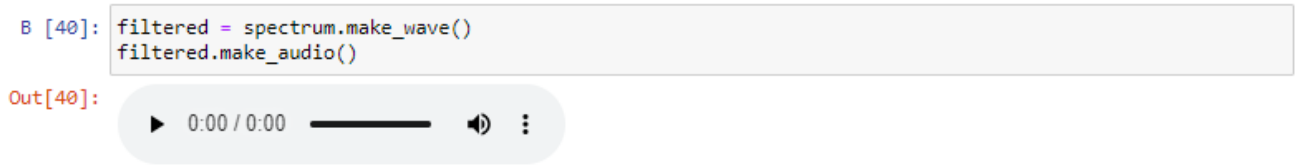


Рис. 19: Полученный сигнал в аудио

В реузльтате, после выполнения данного пункта можно сказать о том, что полученный сигнал звучит тише, чем изначальный сигнал. Он уже не режет уши, как оригинальный сигнал.

# 6 Часть №6: Нахождение сигнала

В шестой части второй лабораторной работы нам необходимо определить, можно ли найти сигнал, состоящий из четныз и нечетных гармоник, спадабщих пропорционально $1/f^2$ ?

Сам текст программы: У треугольных и прямоугольных сигналов есть только нечетные гармоники; в пилообразном сигнале есть и четные, и нечетные гармоники. Гармоники прямоугольных и пилообразных сигналов уменьшаются пропорционально $1/f^2$ гармоники треугольных сигналов — пропорционально $1/f^2$. Можно ли найти сигнал, состоящий из четных и нечетных гармоник, спадающих пропорционально $1/f^2$?

Для начала создадим пилообразный сигнал и представим его в виде аудио

```
B [23]: freq = 500
        signal = SawtoothSignal(freq=freq)
        wave = signal.make_wave(duration=0.5, framerate=20000)
        wave.make_audio()
```

Out[23]: 


Рис. 20: Пилообразный сигнал в аудио

После этого выведем его на экран:

```
1  wave_plot = signal.make_wave(duration=signal.period*10, framerate=20000)
2  wave_plot.plot()
3  decorate(xlabel='Time (s)')
```

Листинг 15: Вывод пилообразного сигнала на экран

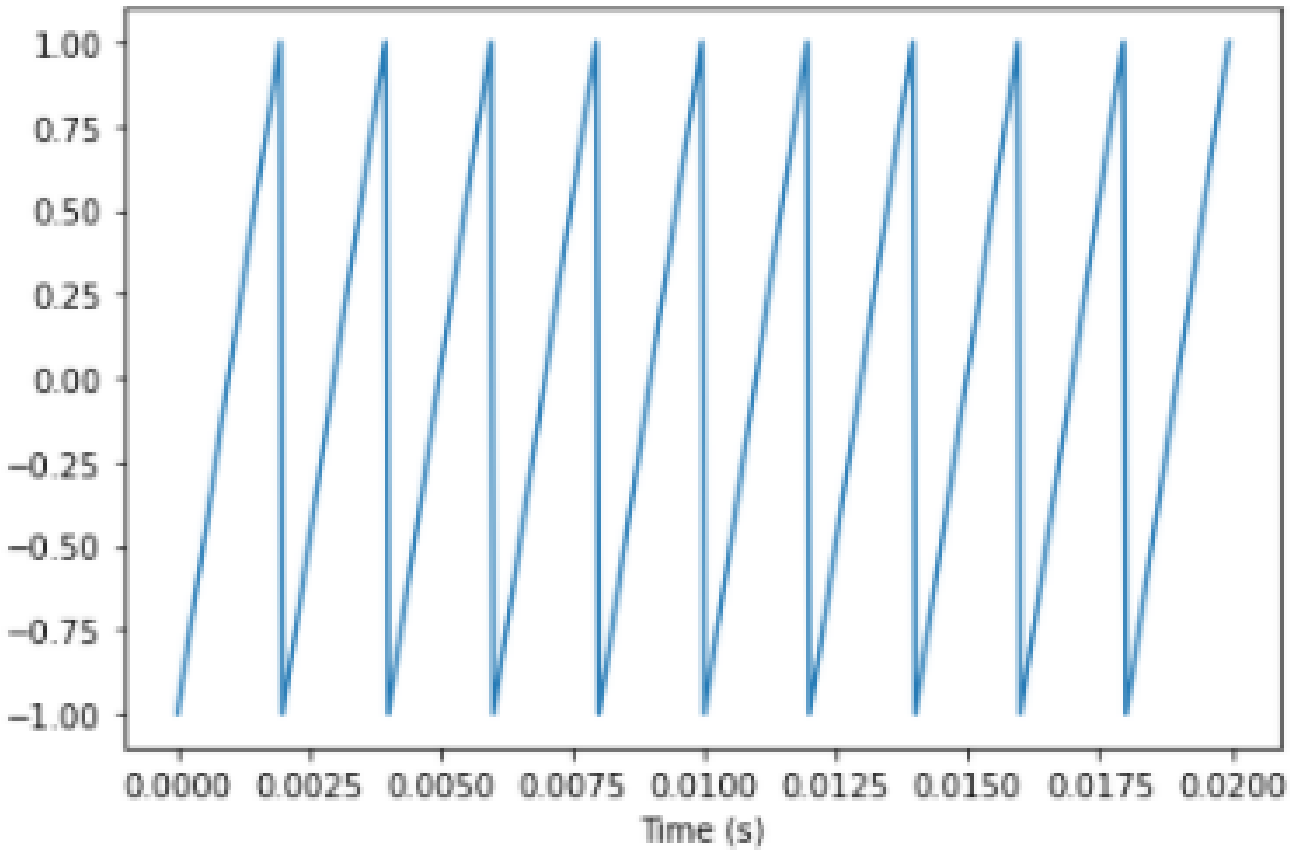


Рис. 21: Вывод пилообразного сигнала на экран

Теперь представим его в виде спектра и так же выведем на экран:

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

Листинг 16: Вывод спектра на экран

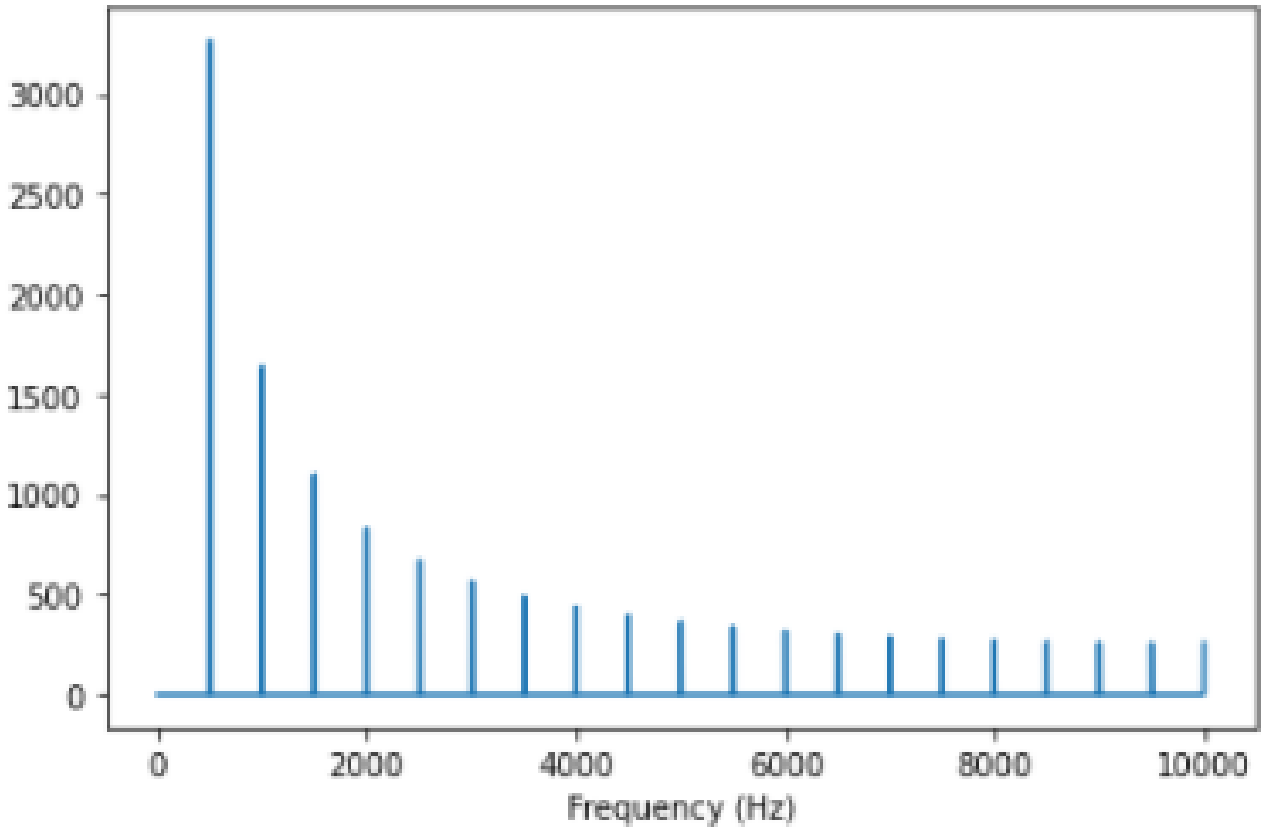


Рис. 22: Вывод спектра на экран

После всего этого нам необходимо вызвать функцию, написанную в прошлом задании, и применить к нашему текущему спектру. Сразу после этого выведем полученный спектр на экран:

```
    spectrum.plot(color='gray')
    filter_spectrum(spectrum)
    spectrum.scale(freq)
    spectrum.plot()
    decorate(xlabel='Frequency (Hz)')
```

Листинг 17: Вызов функции и вывод нового спектра на экран

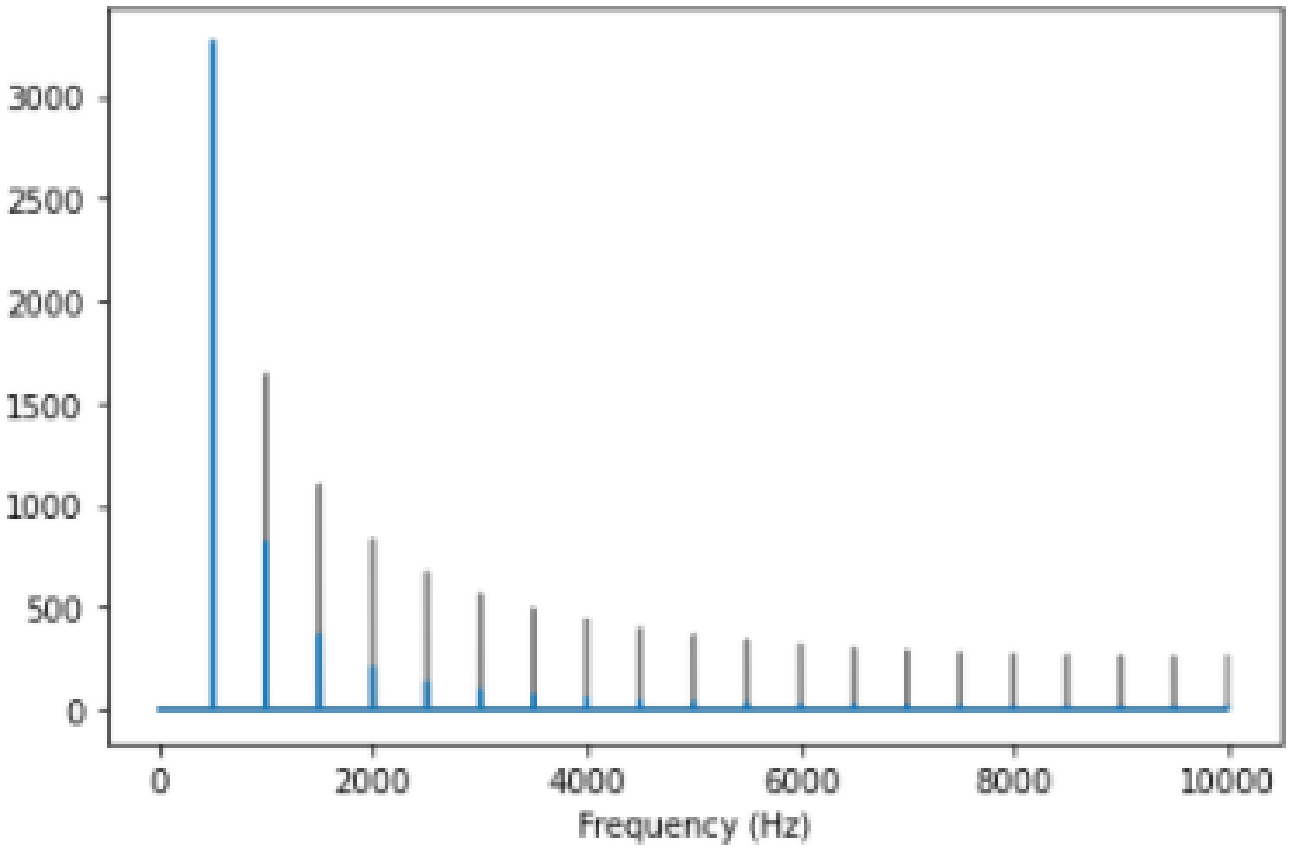


Рис. 23: Вывод нового спектра на экран

Наконец переведем полученный сигнал в аудио:

```
B [27]: wave = spectrum.make_wave()
        wave.make_audio()
```

Out[27]: ▶ 0:00 / 0:00

Рис. 24: Перевод полученного сигнала в аудио

Сразу после этого преобразуем наш спектр в сегменты и выведем так же на экран:

```
1    wave.segment(duration=0.03).plot()
2    decorate(xlabel='Time (s)')
```

Листинг 18: Преобразование спектра в сегменты и вывод на экран

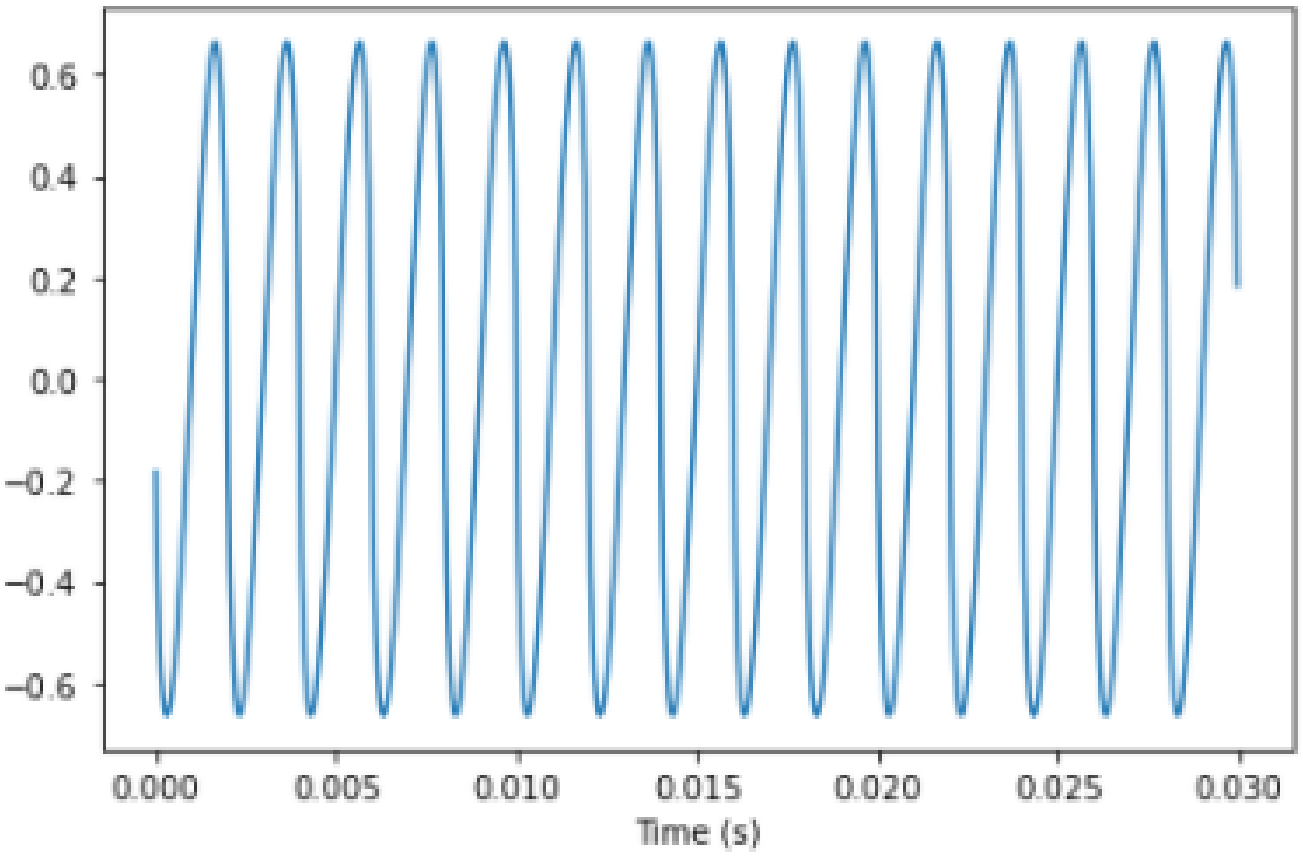


Рис. 25: Вывод нового спектра на экран

Также можно выполнить другой подход, а именно - сложить серию косинусоидальных сигналов с парвильными частотами и амплитудами. Код реализующий это представлен ниже:

```
freqs = np.arange(500, 9500, 500)
amps = 1 / freqs**2
signal = sum(CosSignal(freq, amp) for freq, amp in zip(freqs, amps))
signal
```

Листинг 19: Другой подход

Отобразим спектр полученного сигнала:

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

Листинг 20: Отображение спектра нового сигнала

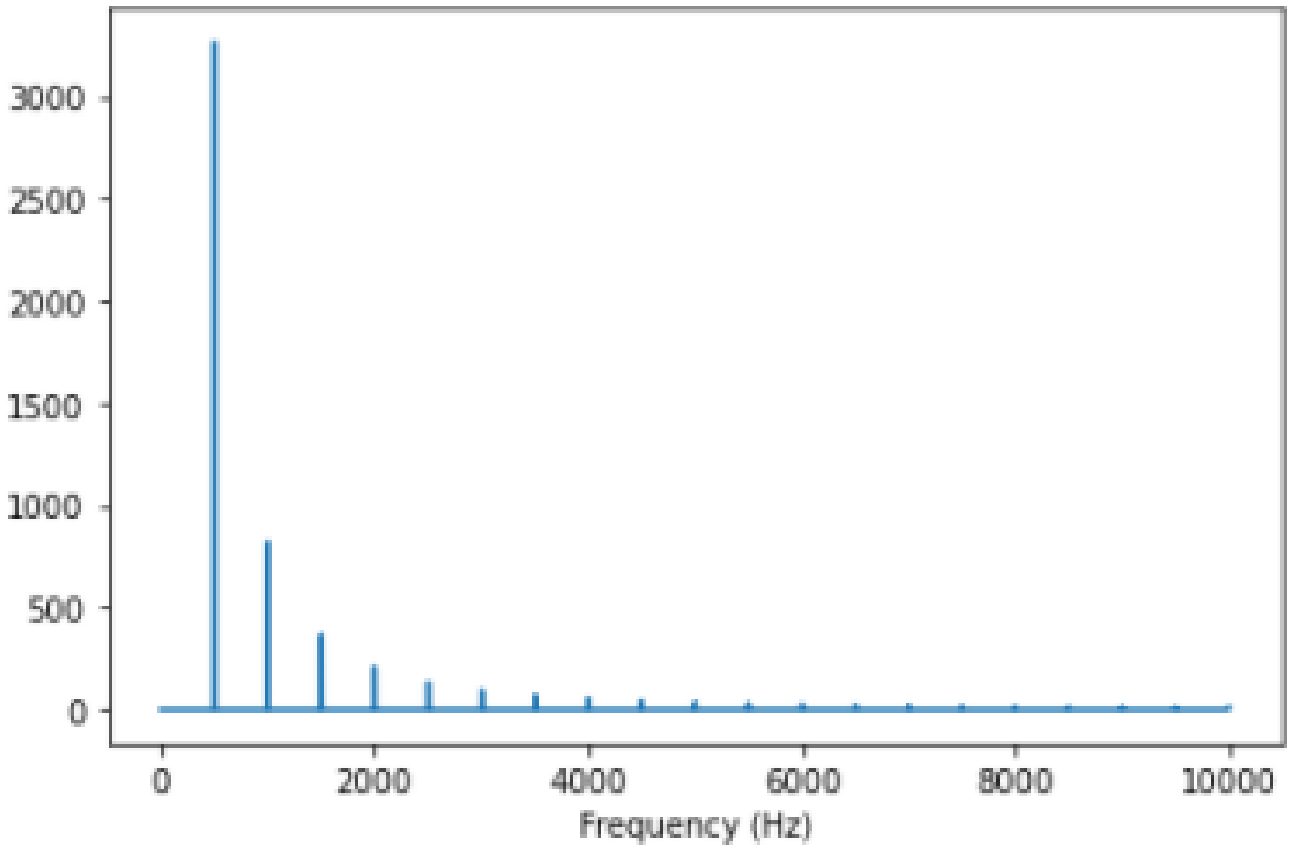


Рис. 26: Вывод нового спектра на экран

Преобразуем новый сигнал в файл:

```
B [31]: wave = signal.make_wave(duration=0.5, framerate=20000)
        wave.make_audio()
```

Out[31]: 0:00 / 0:00

Рис. 27: Преобразование нового спектра в аудио

Наконец, преобразуем наш новый сигнал в сегменты и так же отобразим их на экране:

```
1    wave.segment(duration=0.03).plot()
2    decorate(xlabel='Time (s)')
```

Листинг 21: Отображение сегментов нового сигнала

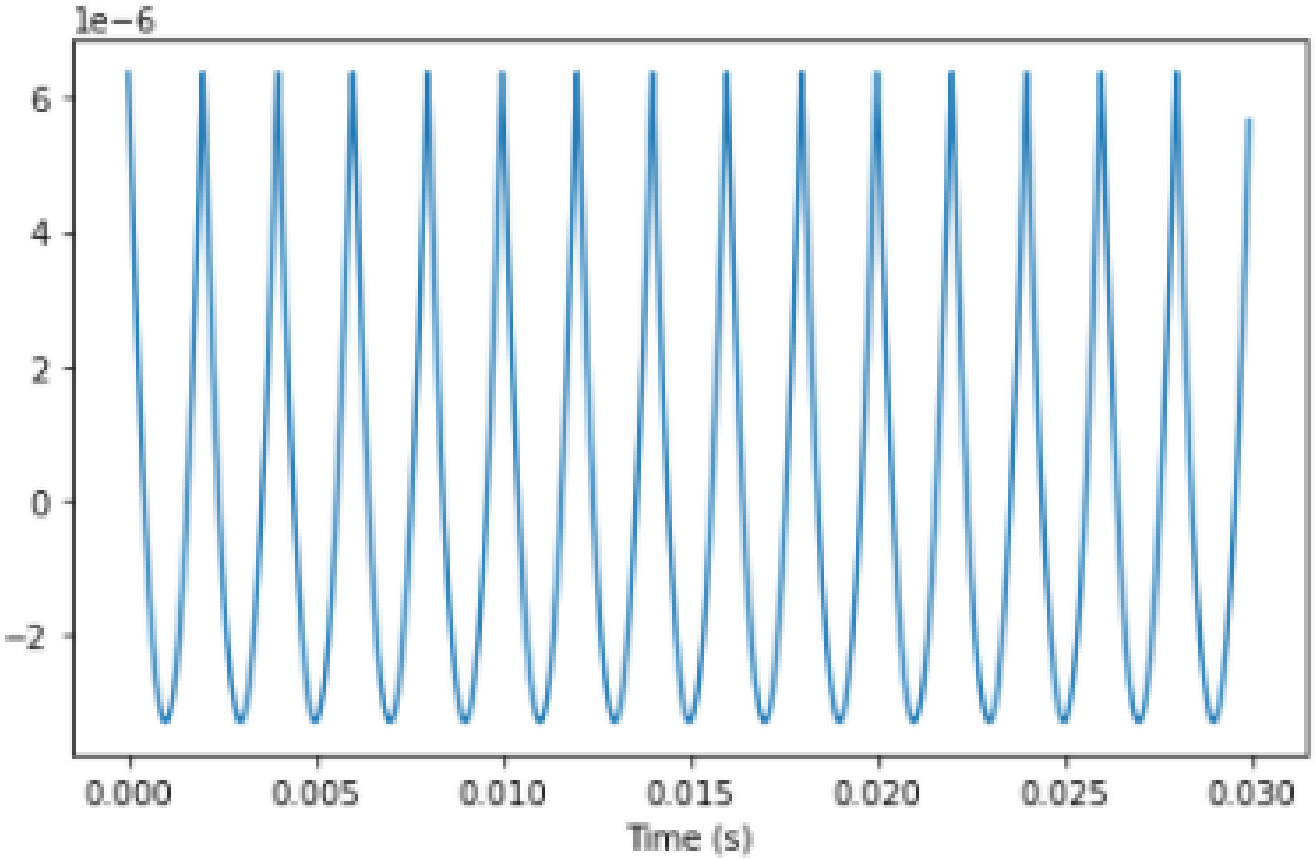


Рис. 28: Отображение сегментов нового сигнала

Как можн оувидеть, полученный нами результат похож на параболы, что так и является, от части. Этот же результат можно добиться, использовав астроенную в `thinkdsp` библиотеку `ParabolicSignal`, вычисляющую параболические формы волны.

Создадим новый сигнал на основе `ParabolicSignal` и переведем его сразу в аудио:

```
B [33]: wave = ParabolicSignal(freq=500).make_wave(duration=0.5, framerate=20000)
        wave.make_audio()
```

Out[33]: ▶ 0:00 / 0:00

Рис. 29: Создание сигнала через `ParabolicSignal`

После этого создадим сегменты из нового сигнала и посмотрим на них:

```
wave.segment(duration=0.03).plot()
decorate(xlabel='Time (s)')
```

Листинг 22: Отображение сегментов нового сигнала `ParabolicSignal`

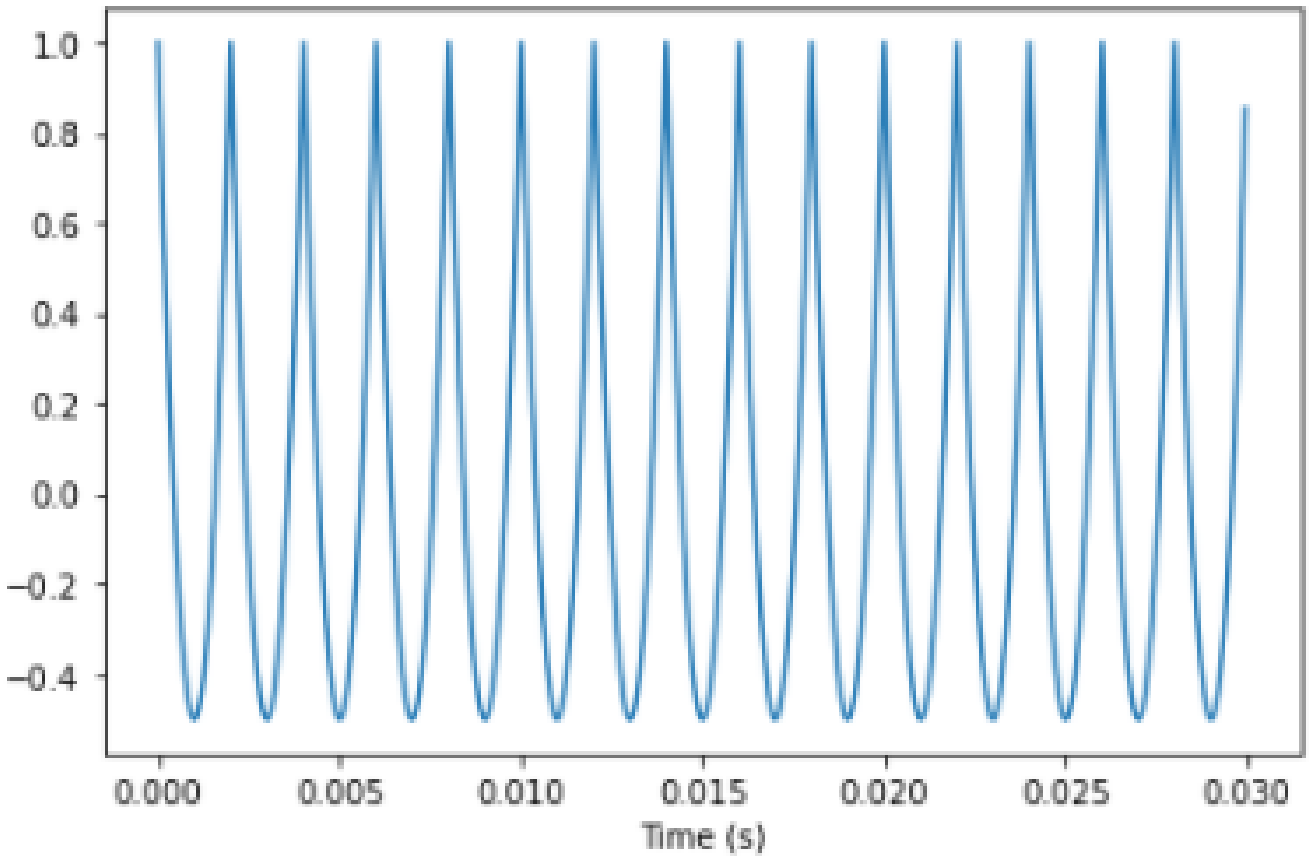


Рис. 30: Отображение сегментов нового сигнала `ParabolicSignal`

В результате выполнения данного пункта можн осказать, что полученные аудиодорожки результата и из `ParabolicSignal` абсолютно идентичны, в результате чего можно сделать вывод, что это один и тот же сигнал.

# 7 Выводы

В результате выполнения данной лабораторной работы мы изучили как надо работать с гармониками, как их обрабатывать, изменять параметры и т.д. Кроме того была реализован и проверен класс для оценки пилообразного сигнала. Также была создана функция для изменения `spectrum` путем изменения его делением каждого элемента `hs` на соответствующую частоту из `fs`.