

Санкт-Петербургский государственный политехнический
университет Петра Великого

**Высшая школа интеллектуальных систем и
суперкомпьютерных технологий**

Лабораторная работа №2

Гармоники

Выполнил студент 3-го курса
группа 3530901/80201
Матвеец Андрей Вадимович

Преподаватель:
Богач Наталья Владимировна

Санкт-Петербург

2021

Содержание

1	Часть №1: Запуск примеров из chap02.ipynb	6
2	Часть №2: Создание SawtoothSignal	7
3	Часть №3: Квадратный сигнал	12
4	Часть №4: Нулевая частота	15
5	Часть №5: Деление амплитуды на частоту	19
6	Часть №6: Нахождение сигнала	21
7	Выводы	29

Список иллюстраций

1	Проверка работоспособности	6
2	Проверка написанного класса <code>SawtoothSignal</code>	8
3	Получение аудиодорожки из написанного класса <code>SawtoothSignal</code>	8
4	Построение спектра	9
5	Построение прямоугольной волны	10
6	Получение аудио из прямоугольного сигнала	10
7	Построение треугольной волны	11
8	Получение аудио из треугольного сигнала	11
9	Построенный спектр	12
10	Аудио из прямоугольного сигнала	13
11	Построенный спектр	13
12	Аудио из синусоидального сигнала	14
13	Построенный треугольный сигнал	16
14	Спектр треугольного сигнала	17
15	<code>spectrum.hs[0]</code>	17
16	<code>spectrum.hs[0] = 100</code>	18
17	Прямоугольный сигнал	19
18	Вывод спектров на экран	20
19	Полученный сигнал в аудио	20
20	Пилообразный сигнал в аудио	21
21	Вывод пилообразного сигнала на экран	22
22	Вывод спектра на экран	23

23	Вывод нового спектра на экран	24
24	Перевод полученного сигнала в аудио	24
25	Вывод нового спектра на экран	25
26	Вывод нового спектра на экран	26
27	Преобразование нового спектра в аудио	26
28	Отображение сегментов нового сигнала	27
29	Создание сигнала через <code>ParabolicSignal</code>	27
30	Отображение сегментов нового сигнала <code>ParabolicSignal</code>	28

Листинги

1	Создание класса <code>SawtoothSignal</code>	7
2	Тестирование класса <code>SawtoothSignal</code>	7
3	Построение спектра	8
4	Построение прямоугольной волны	9
5	Построение треугольной волны	10
6	Построение прямоугольного сигнала	12
7	Построение спектра	12
8	Построение синусоидального сигнала	13
9	Построение спектра	13
10	Построение треугольного сигнала	15
11	Спектр треугольного сигнала	16
12	<code>spectrum.hs</code> [0]	17
13	<code>filter-spectrum</code>	19
14	Работа с сигналами	19
15	Вывод пилообразного сигнала на экран	21
16	Вывод спектра на экран	22
17	Вызов функции и вывод нового спектра на экран	23
18	Преобразование спектра в сегменты и вывод на экран	24
19	Другой подход	25
20	Отображение спектра нового сигнала	25
21	Отображение сегментов нового сигнала	26
22	Отображение сегментов нового сигнала <code>ParabolicSignal</code>	27

1 Часть №1: Запуск примеров из chap02.ipynb

В первом пункте лабораторной работы нам необходимо выполнить программы из файла chap02.ipynb.

```
[27] def view_harmonics(freq, framerate):  
    """Plot the spectrum of a sawtooth signal.  
  
    freq: frequency in Hz  
    framerate: in frames/second  
    """  
  
    signal = SawtoothSignal(freq)  
    wave = signal.make_wave(duration=0.5, framerate=framerate)  
    spectrum = wave.make_spectrum()  
    spectrum.plot(color='C0')  
    decorate(xlabel='Frequency (Hz)', ylabel='Amplitude')  
    display(wave.make_audio())  
  
from ipywidgets import interact, interactive, fixed  
import ipywidgets as widgets  
  
slider1 = widgets.FloatSlider(min=100, max=10000, value=100, step=100)  
slider2 = widgets.FloatSlider(min=5000, max=40000, value=10000, step=1000)  
interact(view_harmonics, freq=slider1, framerate=slider2);
```

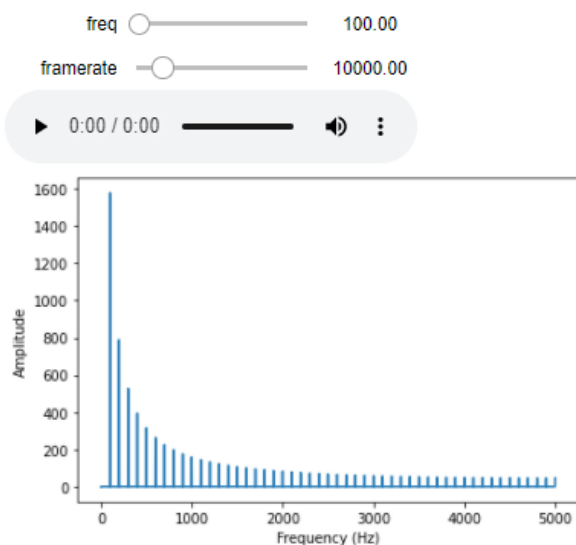


Рис. 1: Проверка работоспособности

В результате выполнения мы увидели, что все запускается как ожидалось, никаких проблем с запуском не появилось и можно переходить к выполнению следующего пункта.

2 Часть №2: Создание SawtoothSignal

Во втором пункте лабораторной работы нам необходимо создать 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 создания класса `SawtoothSignal`

```
1  from thinkdsp import decorate, Sinusoid, normalize, unbias, TriangleSignal,
    SquareSignal, SinSignal, CosSignal, ParabolicSignal
2
3  import numpy as np
4
5  class SawtoothSignal(Sinusoid):
6      def evaluate(self, ts):
7          cycles = self.freq * ts + self.offset / np.pi / 2
8
9          frac, _ = np.modf(cycles)
10         ys = normalize(unbias(frac), self.amp)
11         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.

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

Листинг 2: Тестирование класса `SawtoothSignal`

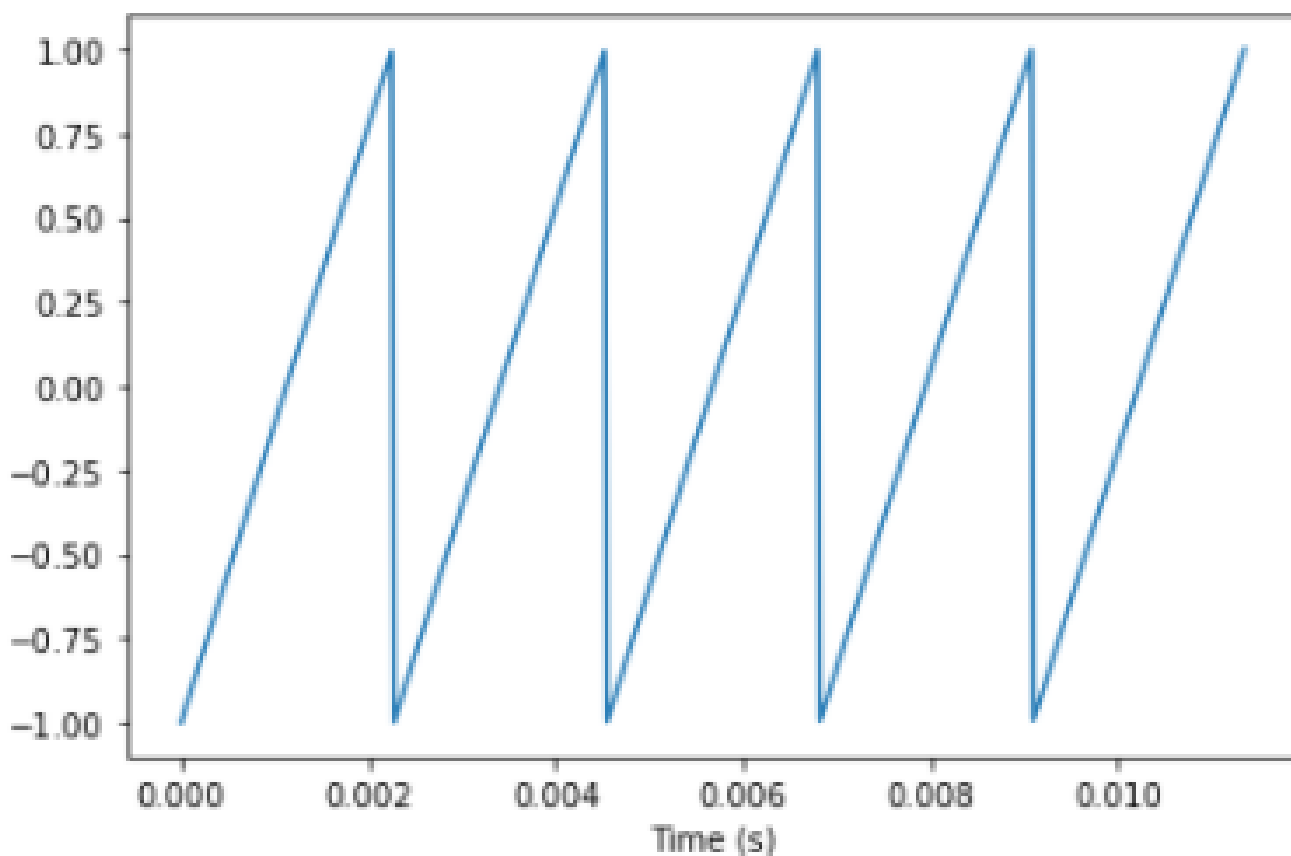


Рис. 2: Проверка написанного класса `SawtoothSignal`

После этого получим аудио дорожку, обращаясь к классу:

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

Out[3]:

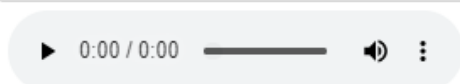


Рис. 3: Получение аудиодорожки из написанного класса `SawtoothSignal`

Теперь построим спектр:

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

Листинг 3: Построение спектра

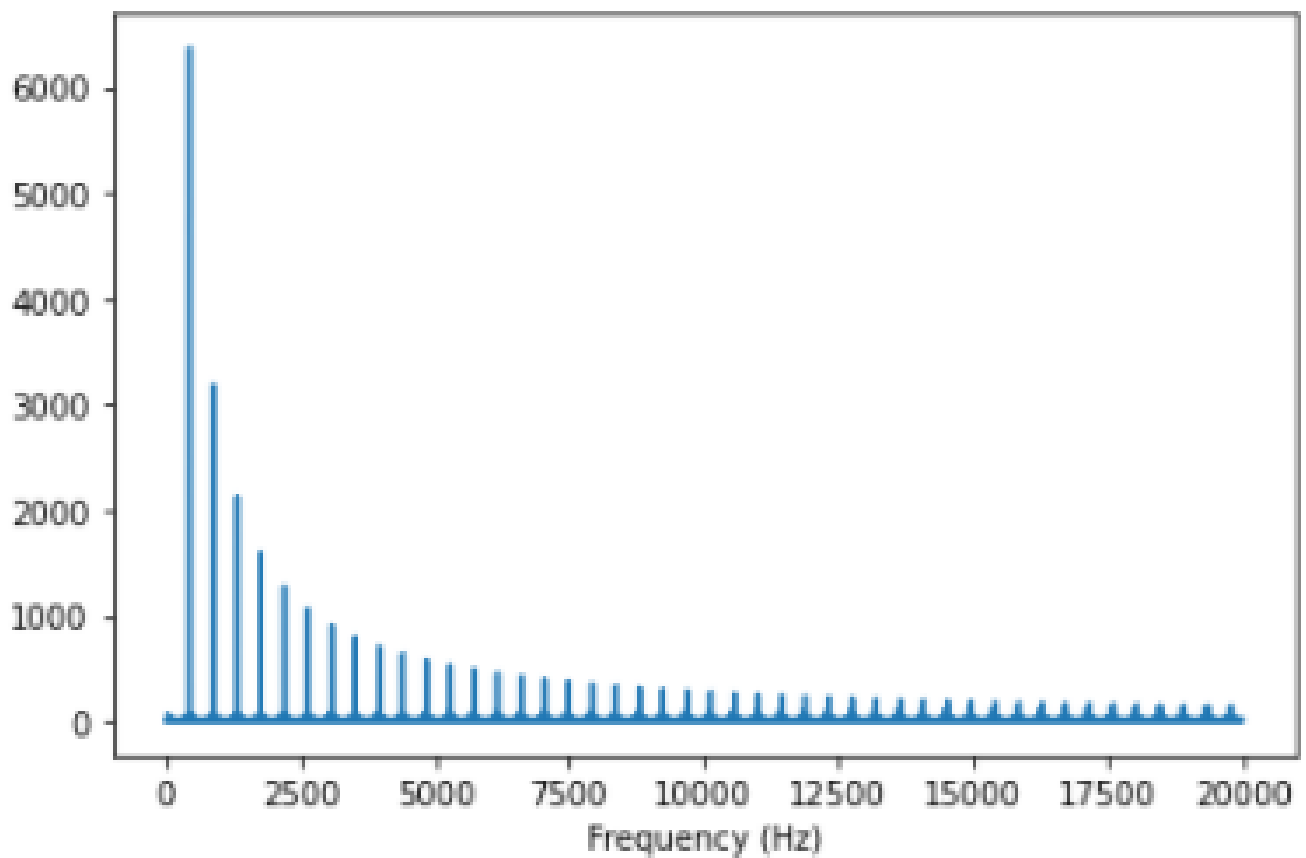


Рис. 4: Построение спектра

Также сравним наш пилообразный сигнал сначала с прямоугольной волной:

```

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

Листинг 4: Построение прямоугольной волны

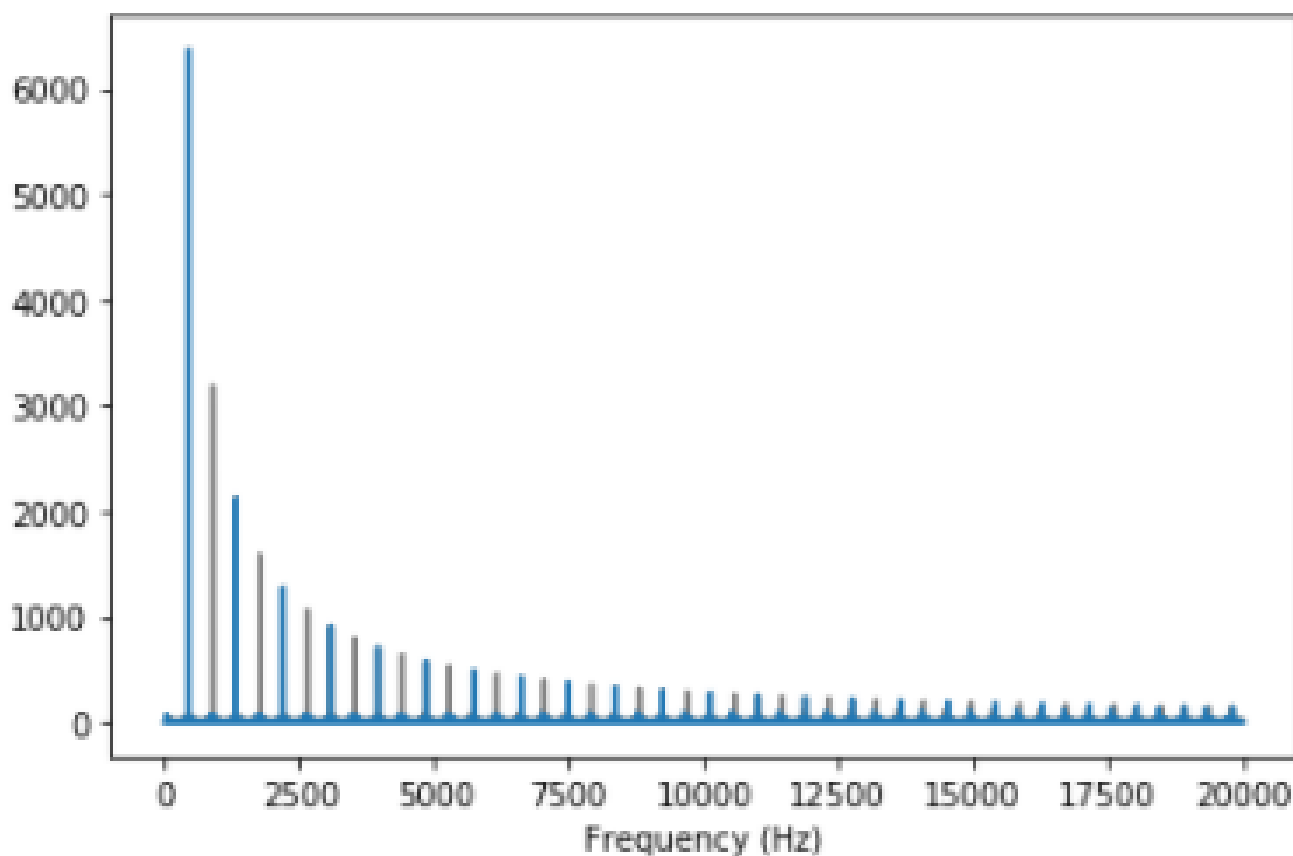


Рис. 5: Построение прямоугольной волны

Также для сравнения получим аудио из прямоугольного сигнала:

```
B [6]: square.make_audio()
```

Out[6]:

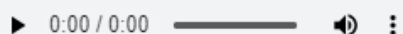


Рис. 6: Получение аудио из прямоугольного сигнала

Теперь сравним наш пилообразный сигнал с треугольным сигналом:

```
1 sawtooth.make_spectrum().plot(color='gray')
2 triangle = TriangleSignal(amp=0.79).make_wave(duration=0.5, framerate
3 =40000)
4 triangle.make_spectrum().plot()
5 decorate(xlabel='Frequency (Hz)')
```

Листинг 5: Построение треугольной волны

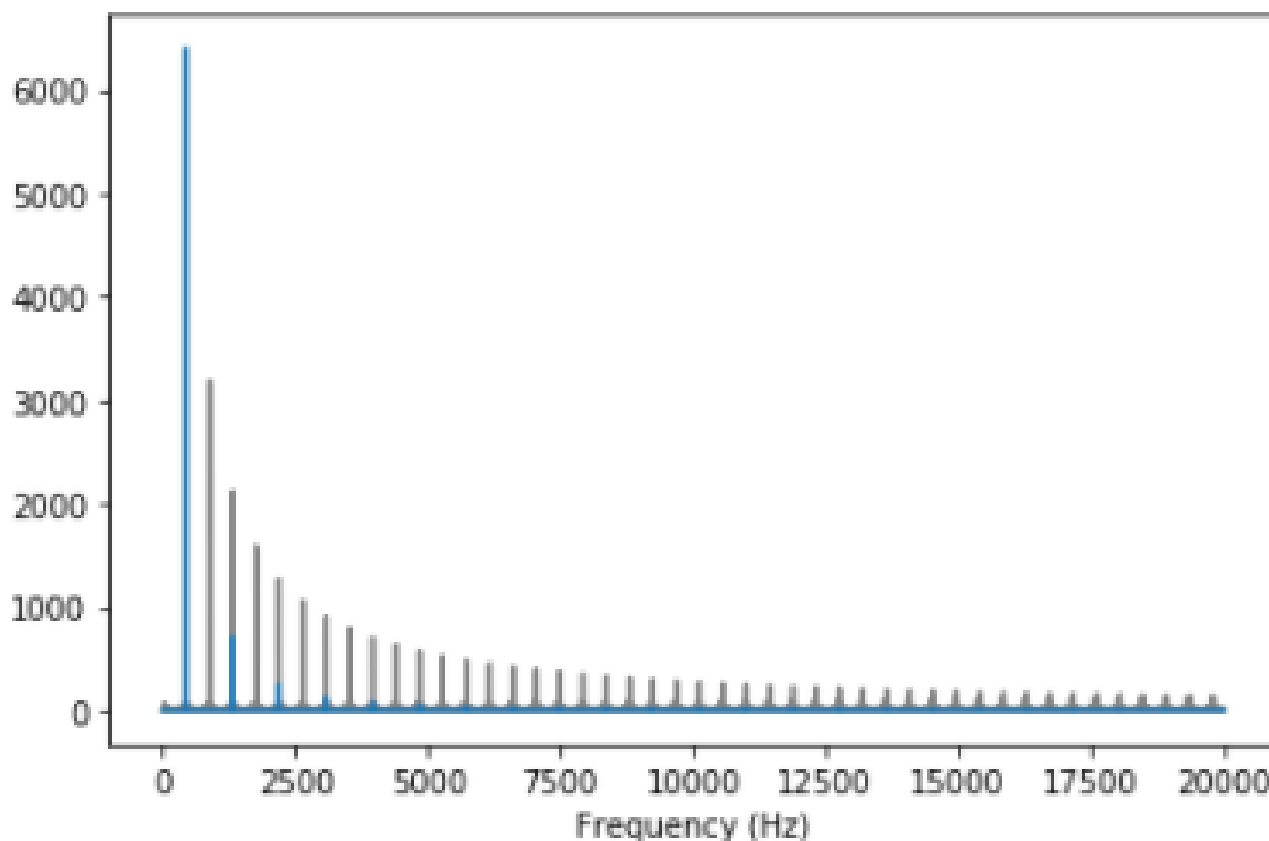


Рис. 7: Построение треугольной волны

Также для сравнения получим аудио из треугольного сигнала:

```
In [8]: triangle.make_audio()
```

Out[8]:

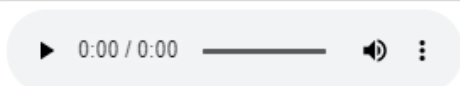


Рис. 8: Получение аудио из треугольного сигнала

В результате выполнения данного пункта можно сделать вывод о том, что в нашем пилообразном сигнале амплитуды частот уменьшаются пропорционально самой частоте, а 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 лабораторной работы нам необходимо создать треугольный 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:

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

Листинг 6: Построение прямоугольного сигнала

После этого построим спектр для этого сигнала:

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

Листинг 7: Построение спектра

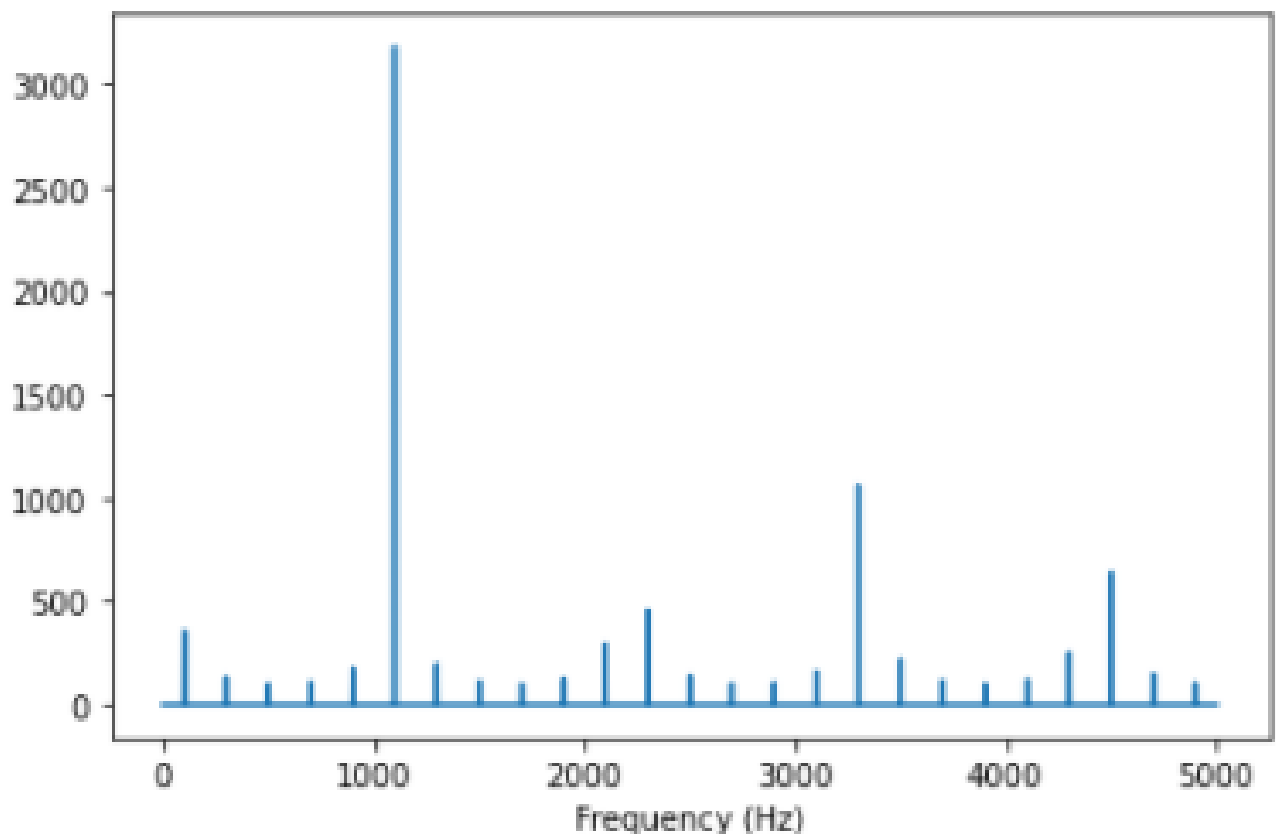


Рис. 9: Построенный спектр

Также создадим из данного сигнала аудиодорожку:

```
In [11]: wave.make_audio()
```

Out[11]:

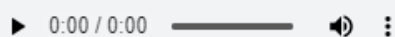


Рис. 10: Аудио из прямоугольного сигнала

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

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

Листинг 8: Построение синусоидального сигнала

Теперь так же создадим для него спектр:

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

Листинг 9: Построение спектра

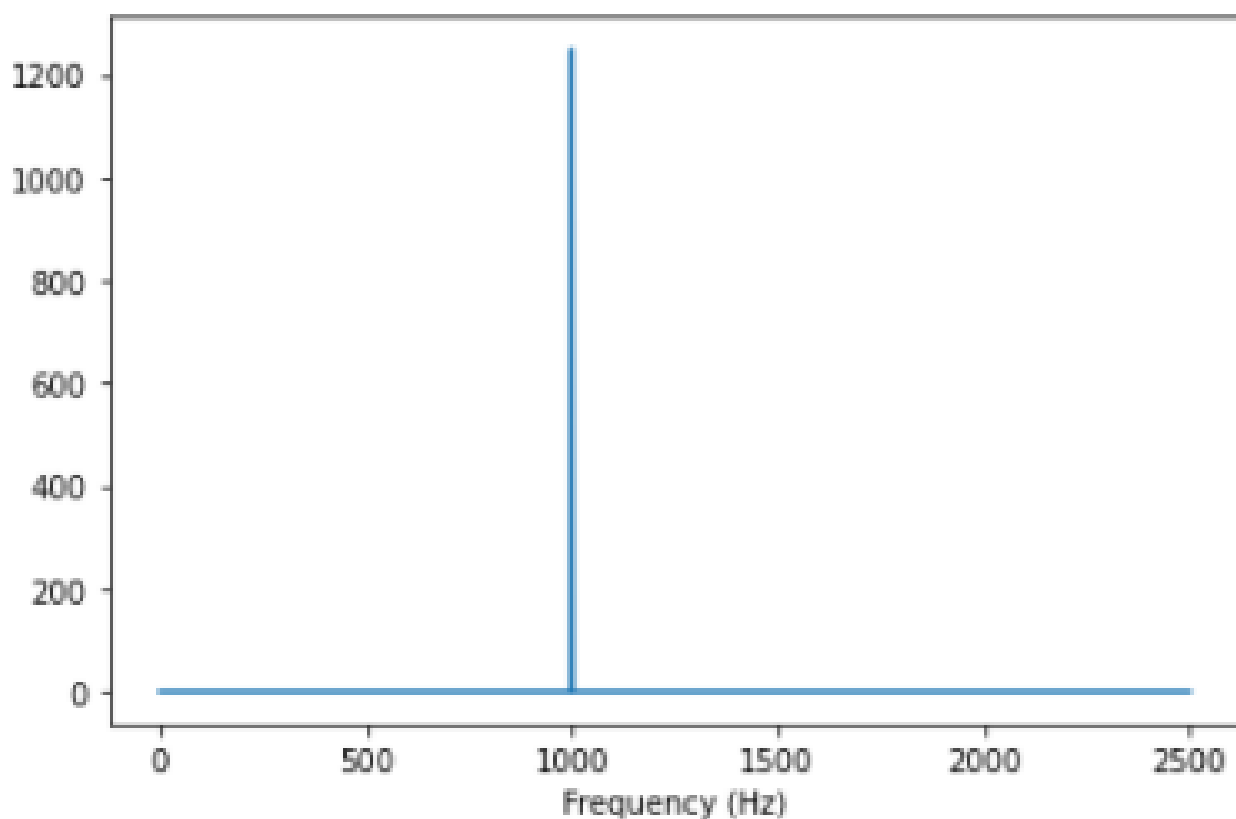


Рис. 11: Построенный спектр

Также создадим из данного сигнала аудиодорожку:

```
In [14]: wave2.make_audio()
```

Out[14]:

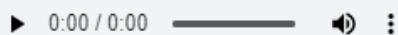


Рис. 12: Аудио из синусоидального сигнала

В результате выполнения данного пункта 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. Так как эти два аудио слышатся по-разному, значит мы можем расслышать «**aliased**-частоты»

4 Часть №4: Нулевая частота

В четвертой части 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]`. Определить какая амплитуда и фаза у компонента.
3. Установить `spectrum.hs[0] = 100` и определить как данная операция влияет на сигнал.

Для начала создадим треугольный сигнал согласно заданию:

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

Листинг 10: Построение треугольного сигнала

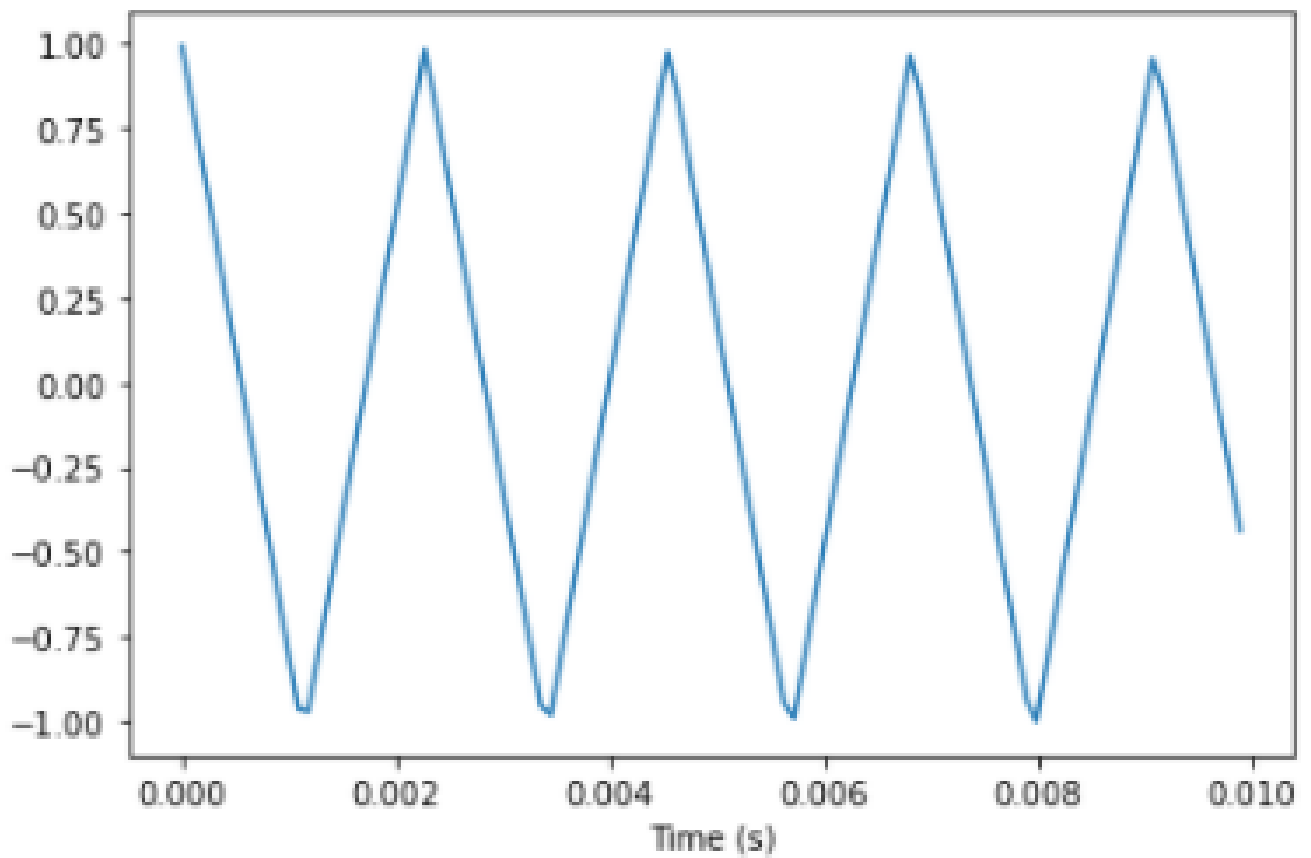


Рис. 13: Построенный треугольный сигнал

После этого создадим спектр для данного сигнала:

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

Листинг 11: Спектр треугольного сигнала

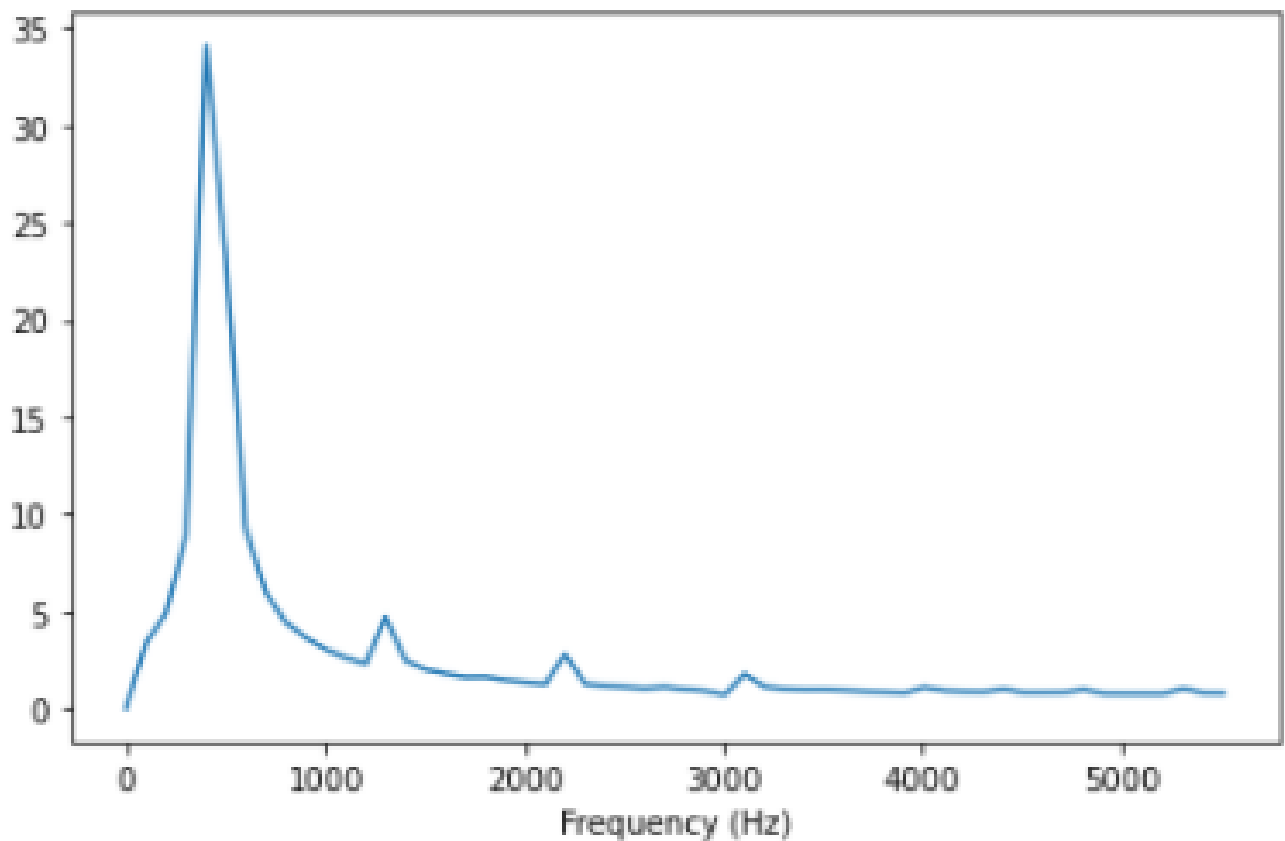


Рис. 14: Спектр треугольного сигнала

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 амплитуду нулевой компоненты разложения, а угол с `Re` описывает частоту.

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

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

Листинг 12: `spectrum.hs[0]`

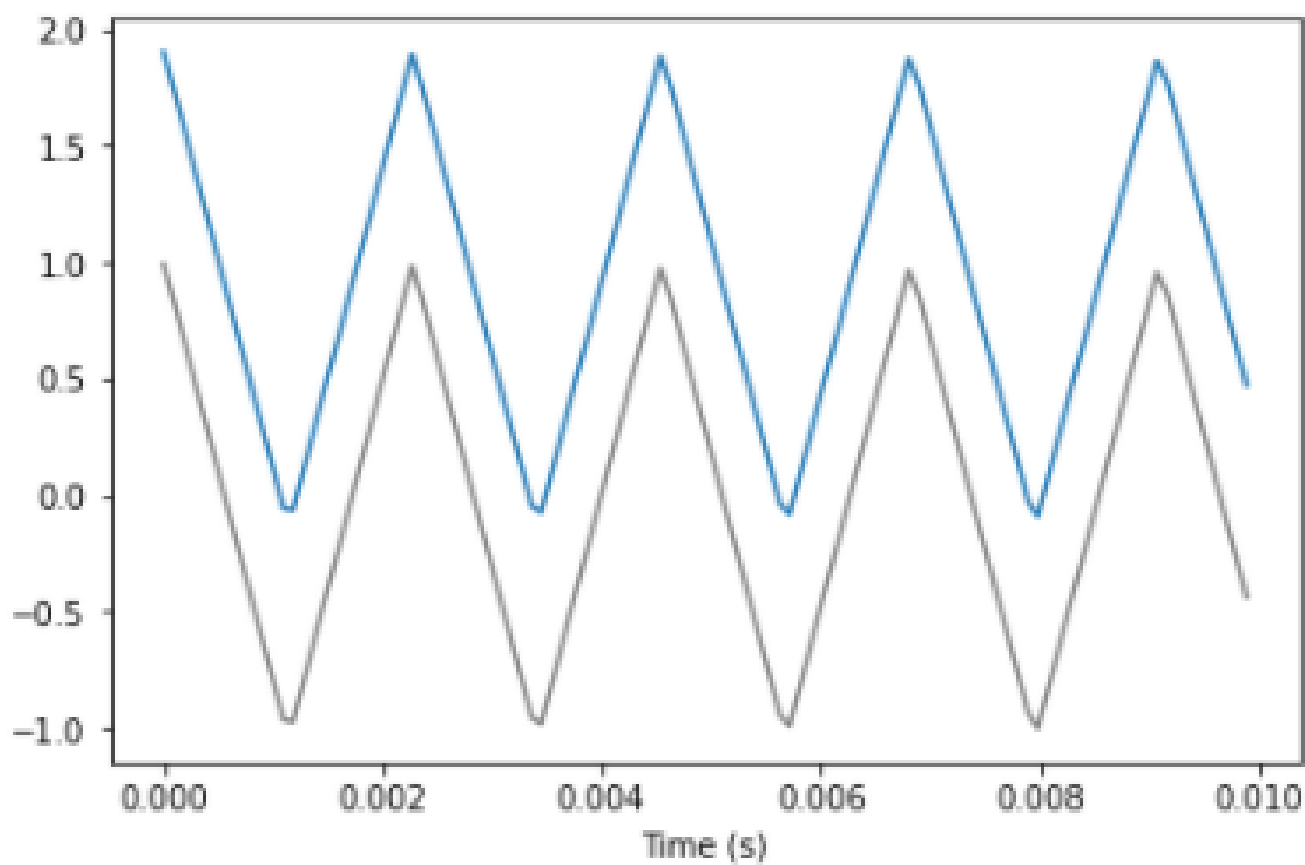


Рис. 16: `spectrum.hs[0] = 100`

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

5 Часть №5: Деление амплитуды на частоту

В пятом пункте лабораторной работы нам необходимо создать функцию, принимающую `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`, используя нашу функцию, и распечатать его.
3. Использовать `spectrum.make-wave`, чтобы сделать `wave` из измененного `spectrum` и прослушать его. Как изменился сигнал?

Для начала напомним нашу функцию `filter-spectrum`:

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

Листинг 13: filter-spectrum

После этого создадим прямоугольный сигнал и сразу переведем его в аудио

```
In [38]: signal = SquareSignal(440)
         wave = signal.make_wave(duration=0.5)
         wave.make_audio()
```

Out[38]:

▶ 0:00 / 0:00 🔊 ⋮

Рис. 17: Прямоугольный сигнал

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

```
1 spectrum = wave.make_spectrum()
2 spectrum.plot(high=10000, color='gray')
3 filter_spectrum(spectrum)
4 spectrum.scale(440)
5 spectrum.plot(high=10000)
6 decorate(xlabel='Frequency (Hz)')
```

Листинг 14: Работа с сигналами

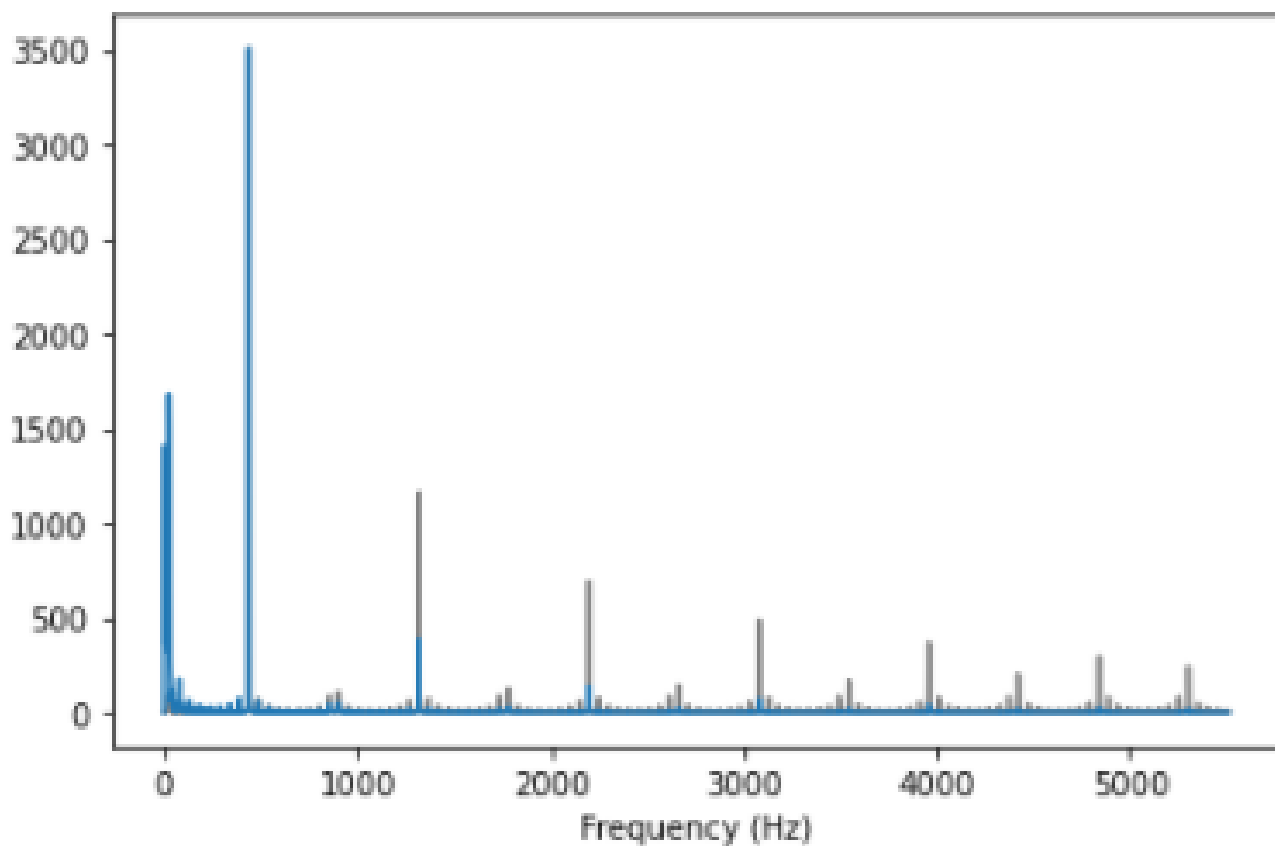


Рис. 18: Вывод спектров на экран

Теперь так же переведем полученный сигнал в аудио:

```
In [40]: filtered = spectrum.make_wave()  
         filtered.make_audio()
```

Out[40]:

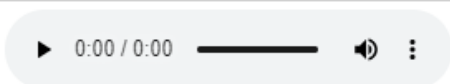


Рис. 19: Полученный сигнал в аудио

В результате 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 четных и нечетных гармоник, спадающих 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

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

Out[23]:

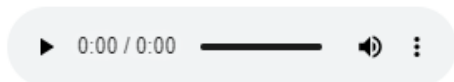


Рис. 20: Пилообразный сигнал в аудио

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

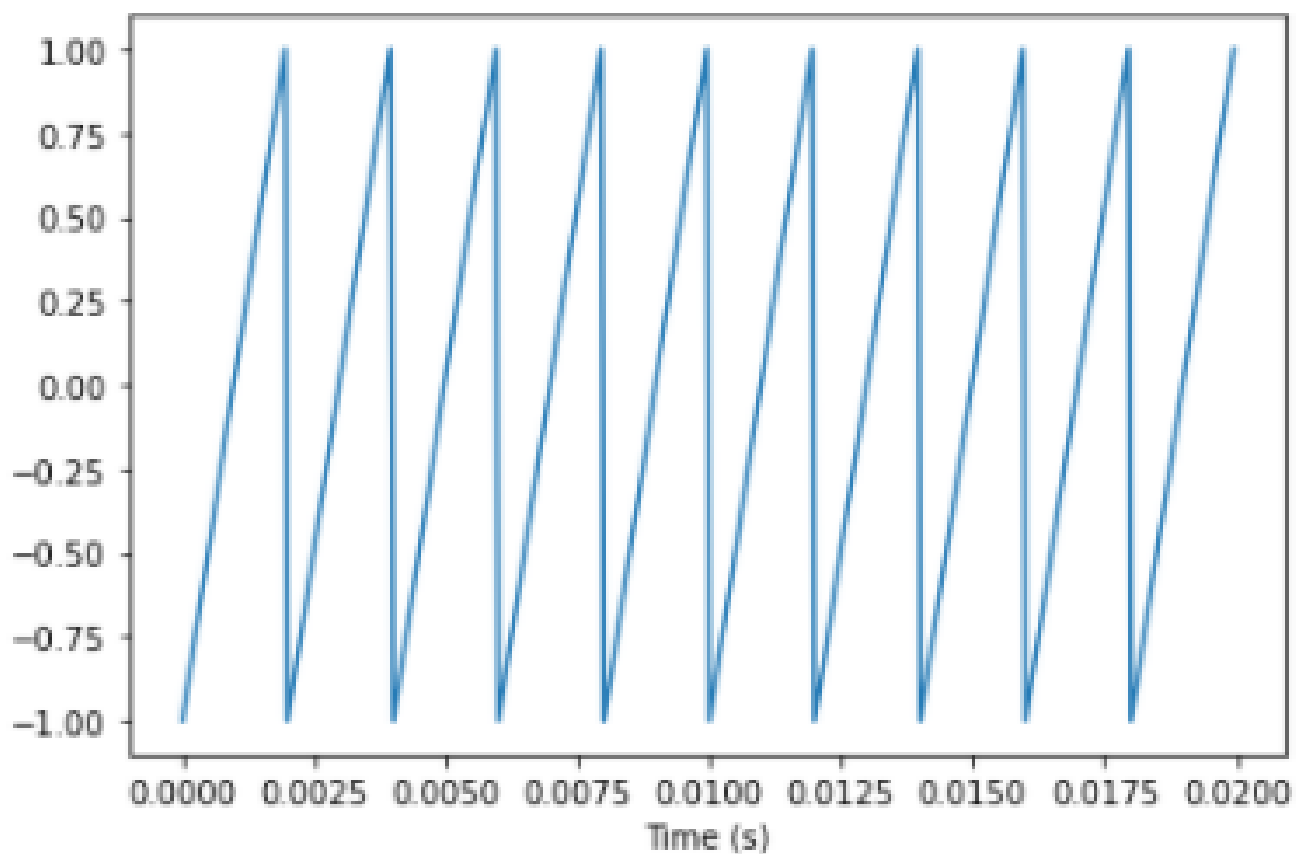


Рис. 21: Вывод пилообразного сигнала на экран

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

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

Листинг 16: Вывод спектра на экран

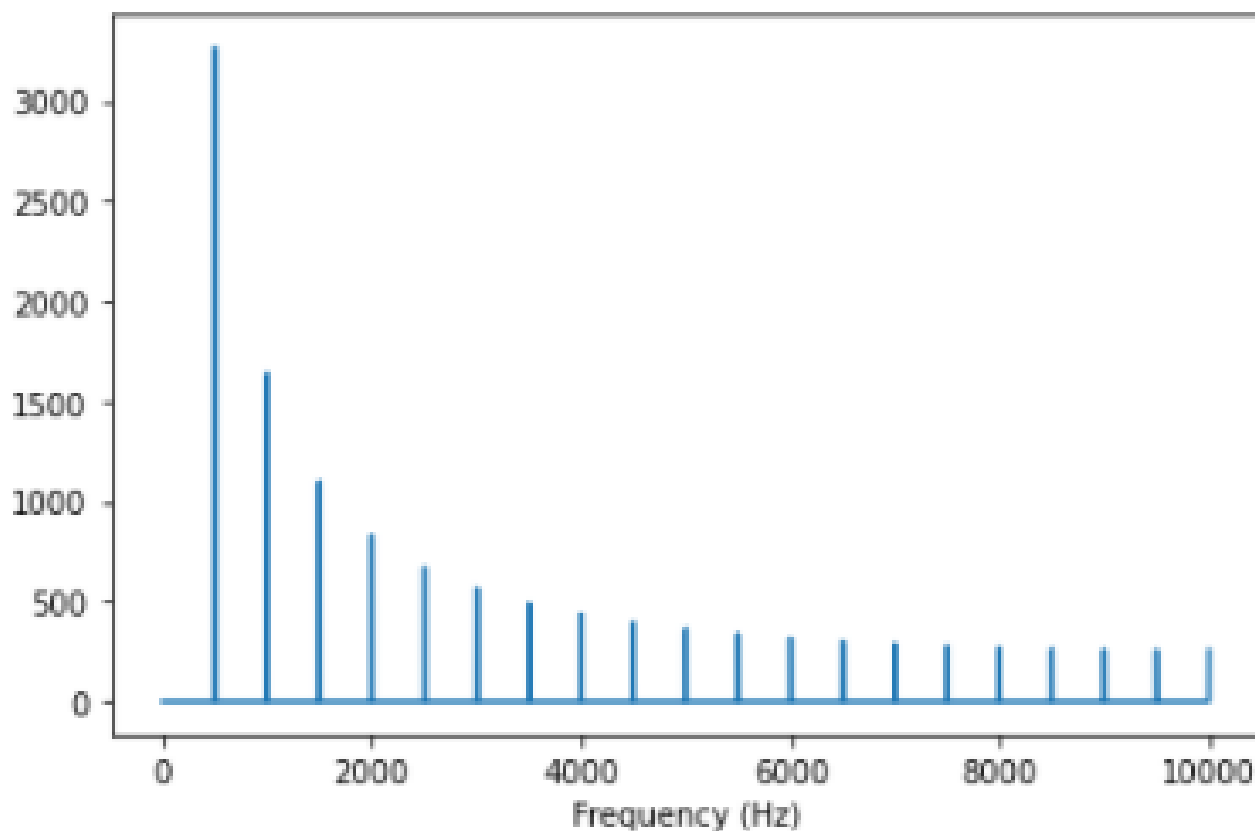


Рис. 22: Вывод спектра на экран

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:

```

1 spectrum.plot(color='gray')
2 filter_spectrum(spectrum)
3 spectrum.scale(freq)
4 spectrum.plot()
5 decorate(xlabel='Frequency (Hz)')
```

Листинг 17: Вызов функции и вывод нового спектра на экран

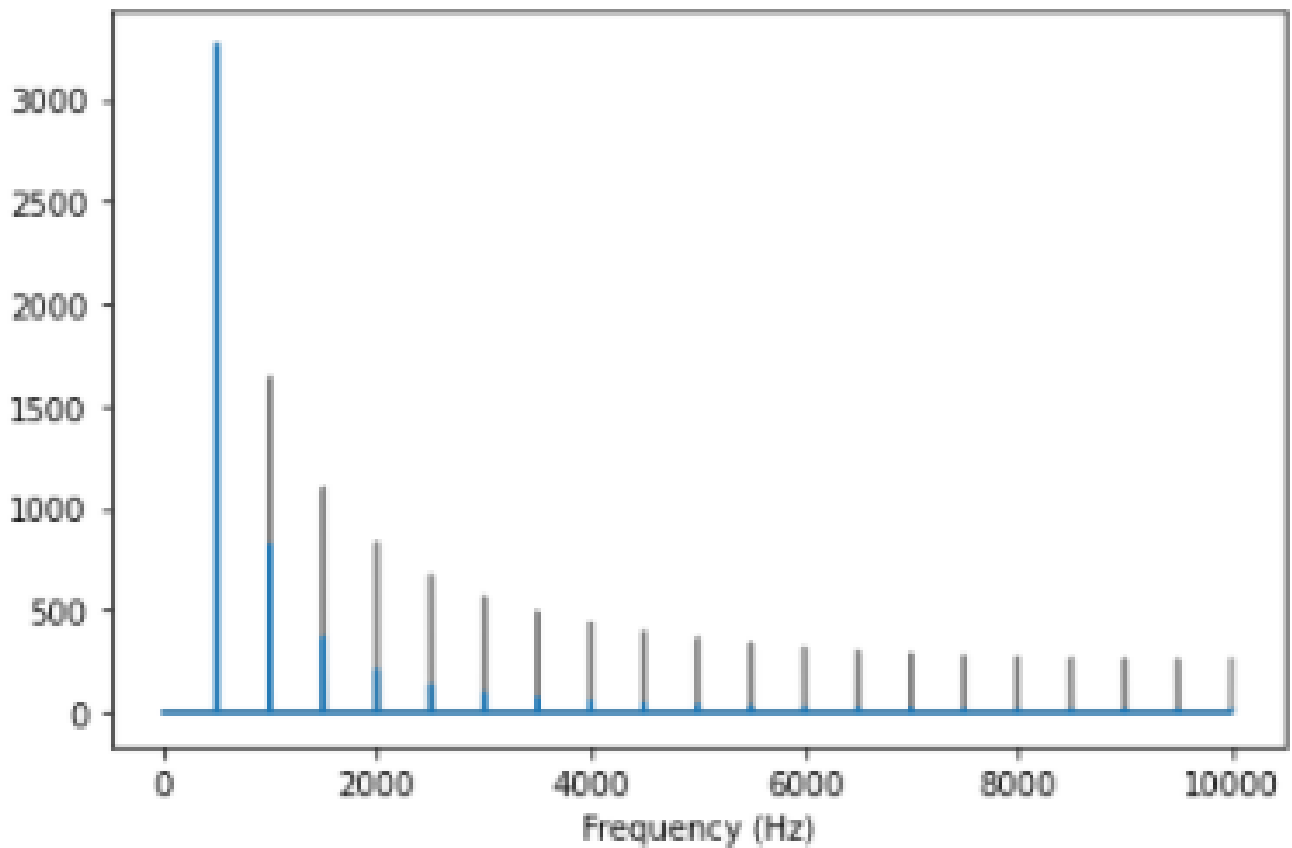


Рис. 23: Вывод нового спектра на экран

Наконец переведем полученный сигнал в аудио:

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

Out[27]:



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

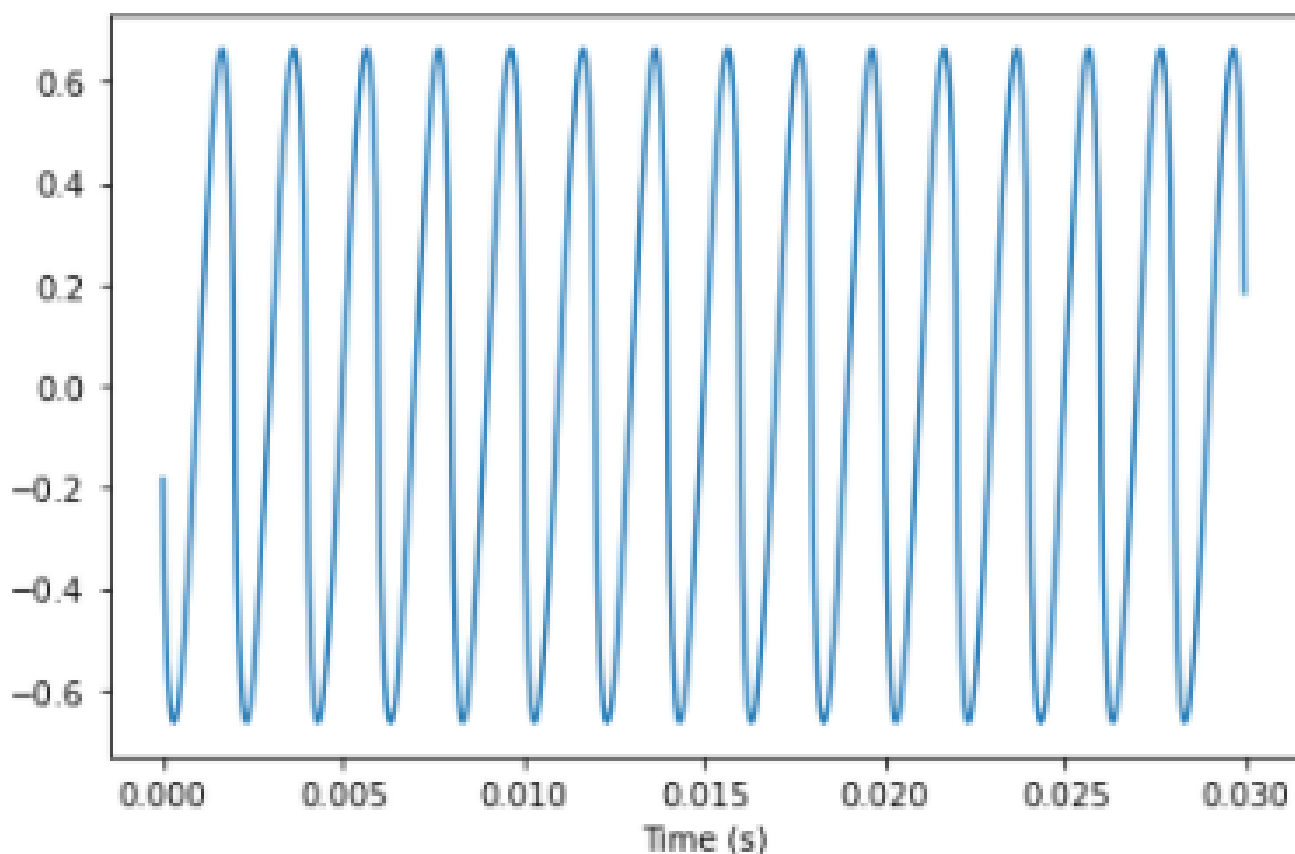


Рис. 25: Вывод нового спектра на экран

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

```

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

```

Листинг 19: Другой подход

Отобразим спектр полученного сигнала:

```

1     spectrum = wave.make_spectrum()
2     spectrum.plot()
3     decorate(xlabel='Frequency (Hz)')

```

Листинг 20: Отображение спектра нового сигнала

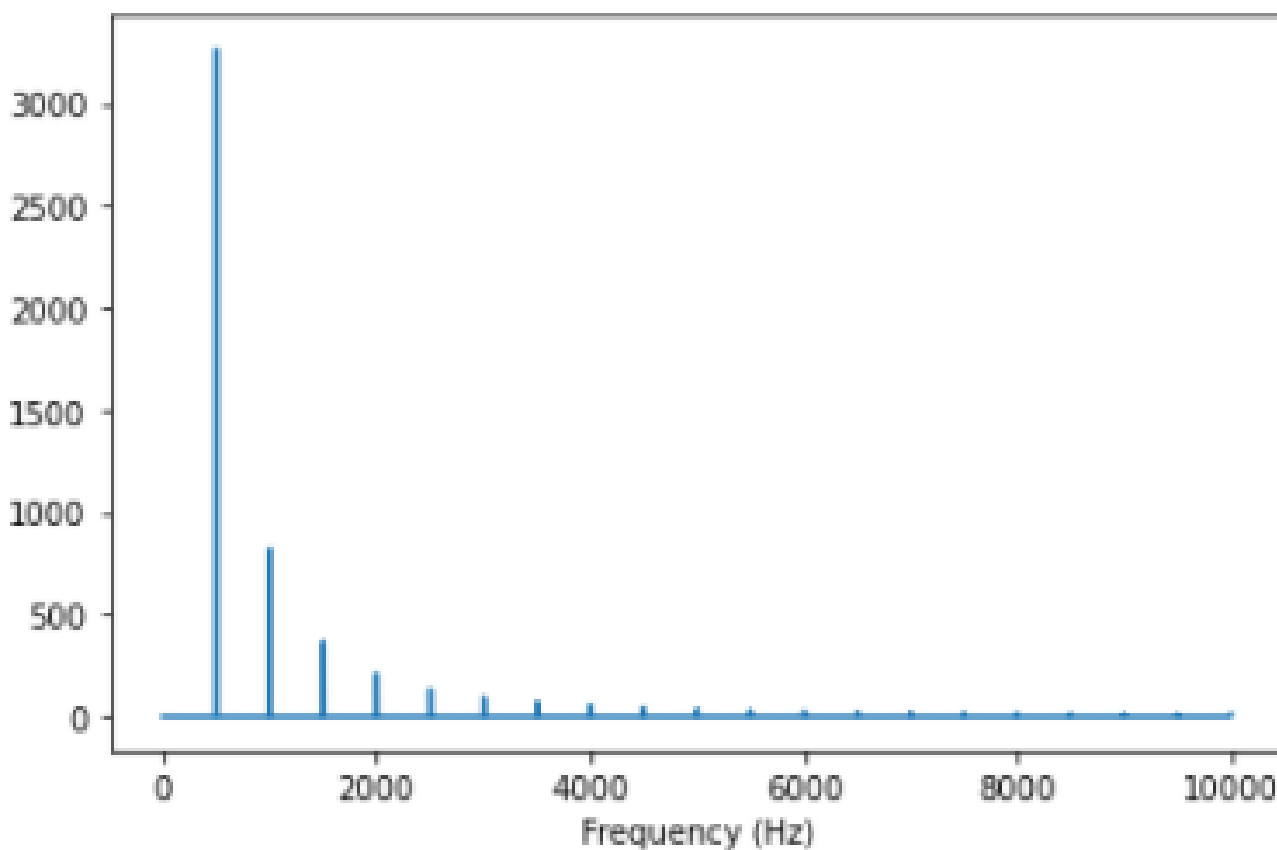


Рис. 26: Вывод нового спектра на экран

Преобразуем новый сигнал в файл:

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

Out[31]:



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

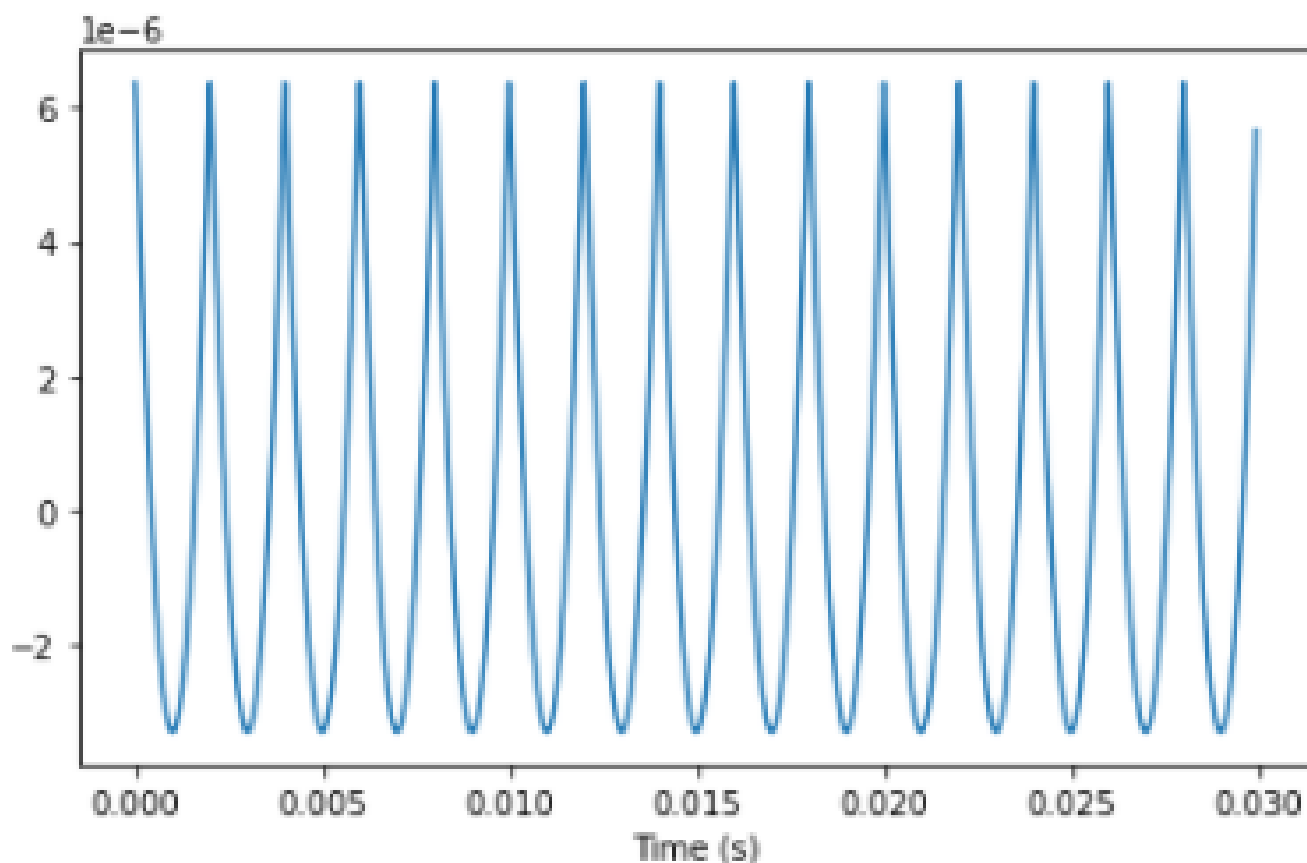


Рис. 28: Отображение сегментов нового сигнала

Как можно увидеть, полученный нами результат похож на параболы, что является правдой, от части. Этот же результат можно добиться, используя 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]:

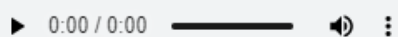


Рис. 29: Создание сигнала через `ParabolicSignal`

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

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

Листинг 22: Отображение сегментов нового сигнала `ParabolicSignal`

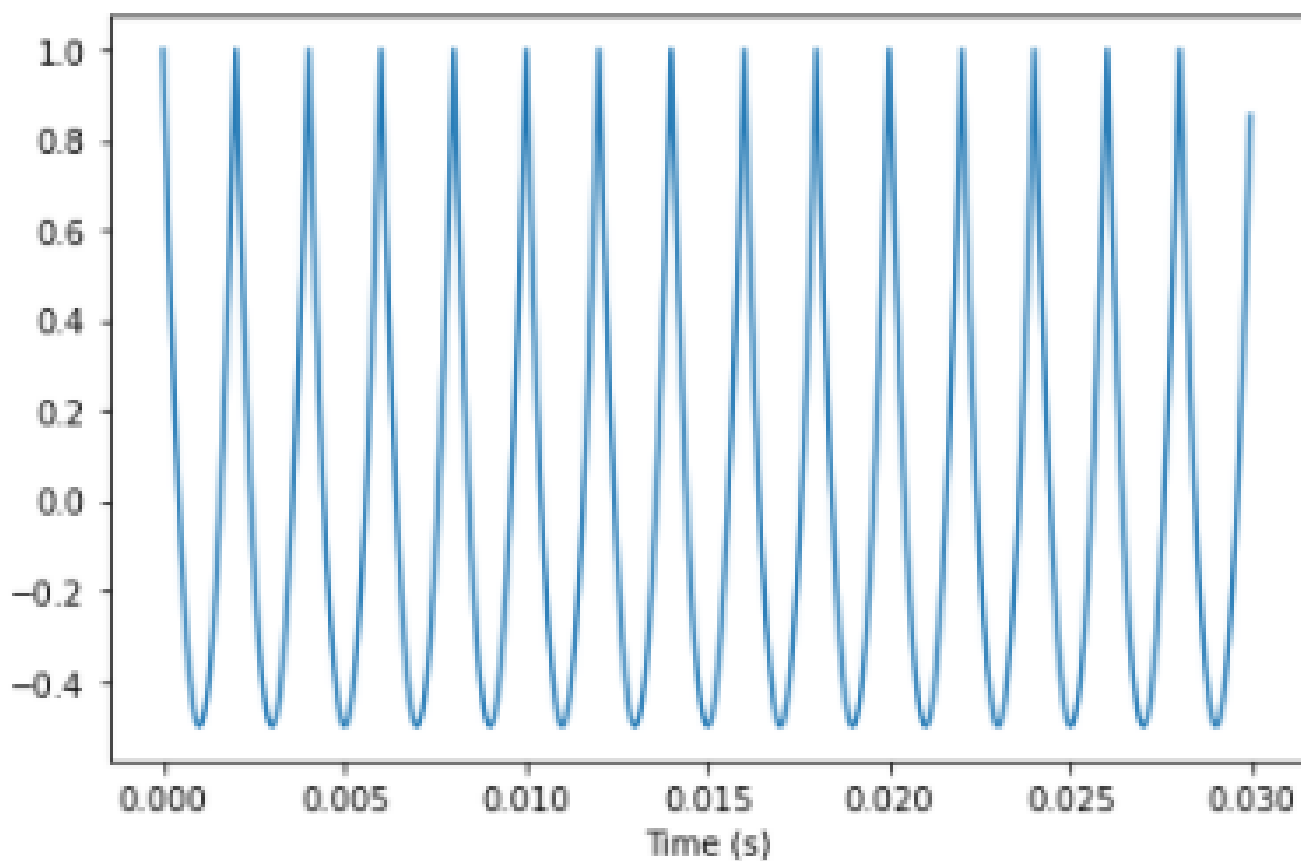


Рис. 30: Отображение сегментов нового сигнала `ParabolicSignal`

В результате выполнения данного пункта можно сказать, 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 способы работы с гармониками, их обработки, смены параметров и т.д. Кроме того был 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**.