Лабораторная работа 9

Луняк Николай

20 апреля 2021 г.

Оглавление

Список иллюстраций

Листинги

Глава 1

Багаж импортов

Разные штуки, которые часто оказываются необходимыми.

```
from thinkdsp import Signal, Sinusoid, SquareSignal,
    TriangleSignal, SawtoothSignal, ParabolicSignal
from thinkdsp import normalize, unbias, PI2, decorate
from thinkdsp import Chirp
from thinkdsp import read_wave
from thinkdsp import Spectrum, Wave,
    UncorrelatedGaussianNoise, Spectrogram
from thinkdsp import Noise

import numpy as np
import pandas as pd

from matplotlib import pyplot as plt

import thinkstats2

from scipy.stats import linregress

import scipy
import scipy.fftpack

import scipy.signal

from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets

loglog = dict(xscale='log', yscale='log')

PI2 = np.pi * 2
```

Листинг 1.1: Импорты

Глава 2

Сравнение `diff` и `differentiate` на примере треугольного сигнала

Создадим треугольный сигнал:

```
in_wave = TriangleSignal(freq=50).make_wave(duration=0.1,
    framerate=44100)
in_wave.plot()
decorate(xlabel='Time (s)')
```

Листинг 2.1: Создаем треугольный сигнал

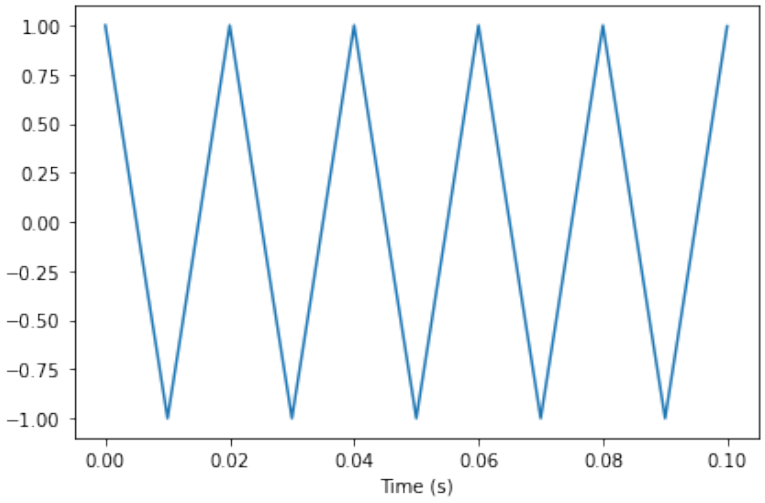

Рис. 2.1: Исходный сигнал

Посчитаем его производную численно:

```
out_wave = in_wave.diff()
out_wave.plot()
decorate(xlabel='Time (s)')
```

Листинг 2.2: Делаем `diff`

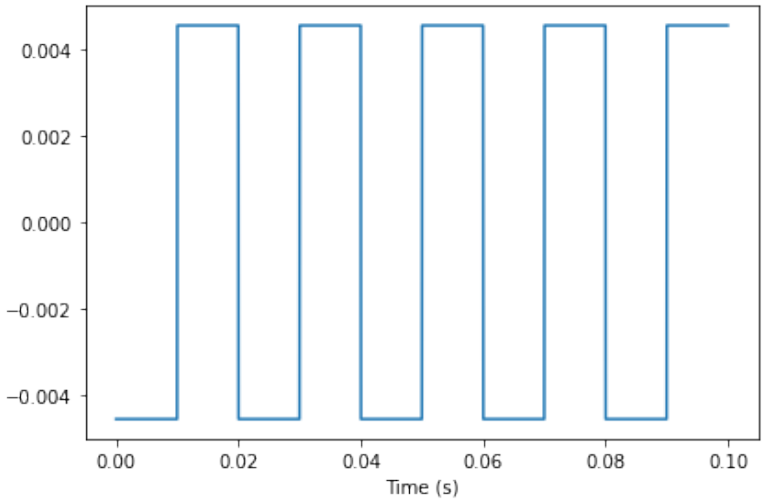

Рис. 2.2: После `diff`

Получился прямоугольный сигнал. Теперь возьмем спектр:

```
in_spectrum = in_wave.make_spectrum()
in_spectrum.plot(high=2000)
decorate(xlabel='Frequency (1/s)')
```

Листинг 2.3: Спектр

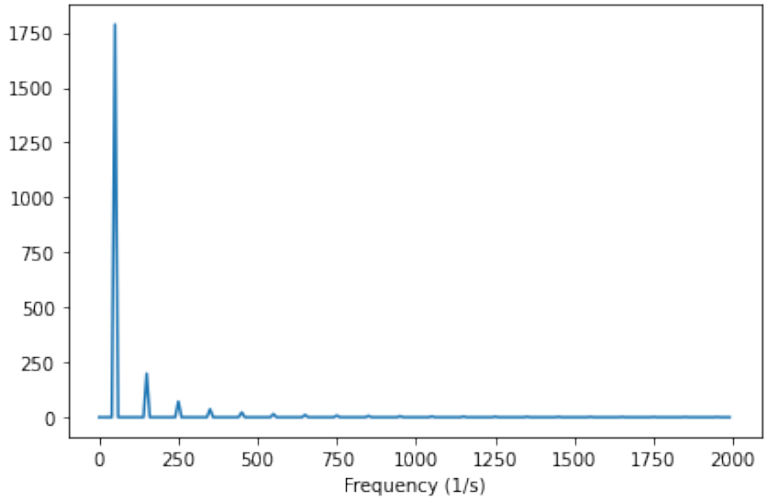

Рис. 2.3: Спектр

Получим его производную:

```
out_spectrum = in_spectrum.differentiate()
out_spectrum.plot(high=2000)
decorate(xlabel='Frequency (1/s)')
```

Листинг 2.4: Продифференциированный спектр

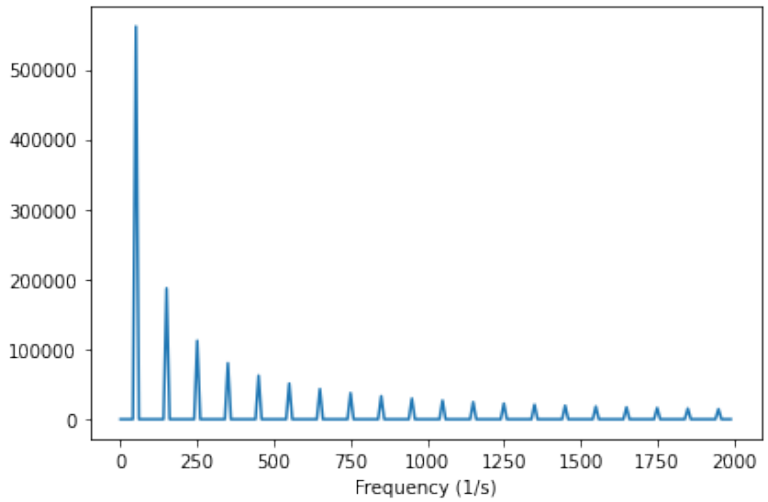

Рис. 2.4: Продифференциированный спектр

А теперь превратим обратно в сигнал во времени;

```
out_wave2 = out_spectrum.make_wave()
out_wave2.plot()
decorate(xlabel='Time (s)')
```

Листинг 2.5: Новый сигнал

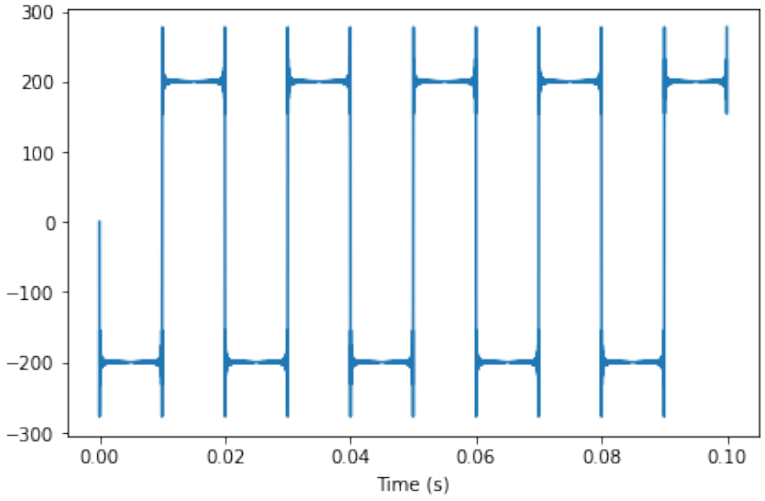

Рис. 2.5: Новый сигнал

И мы снова получили прямоугольный сигнал, правда в этот раз заметно более «плохого качества» (края плохо аппроксимируются конечным числом синусоид).

Глава 3

Сравнение `cumsum` и `integrate` на примере прямоугольного сигнала

Уже по названию напрашивается аналогия с прошлым разделом, но убедимся в этмо более формально.

```
in_wave = SquareSignal(freq=50).make_wave(duration=0.1,
    framerate=44100)
in_wave.plot()
decorate(xlabel='Time (s)')
```

Листинг 3.1: Прямоугольный сигнал

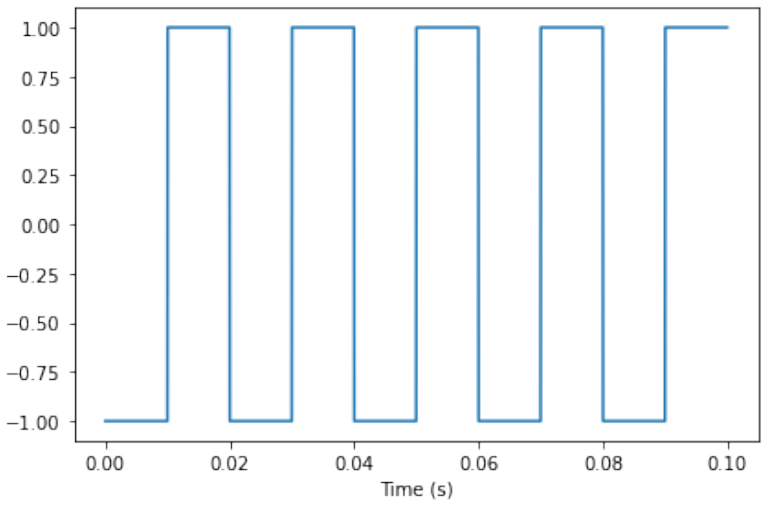

Рис. 3.1: Прямоугольный сигнал

Считаем `cumsum`:

```
out_wave = in_wave.cumsum()
out_wave.plot()
decorate(xlabel='Time (s)')
```

Листинг 3.2: «Просуммированный»

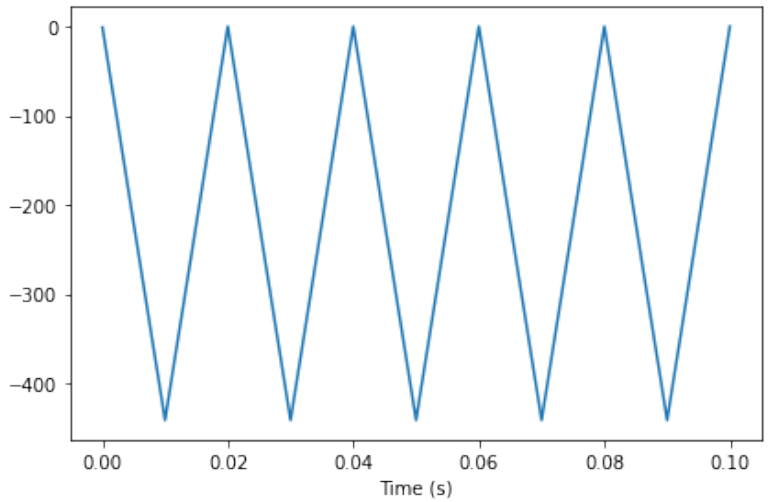

Рис. 3.2: «Просуммированный»

Теперь возьмем спектр:

```
in_spectrum = in_wave.make_spectrum()
in_spectrum.hs[0] = 0
in_spectrum.plot(high=5000)
decorate(xlabel='Frequency (1/s)')
```

Листинг 3.3: Спектр

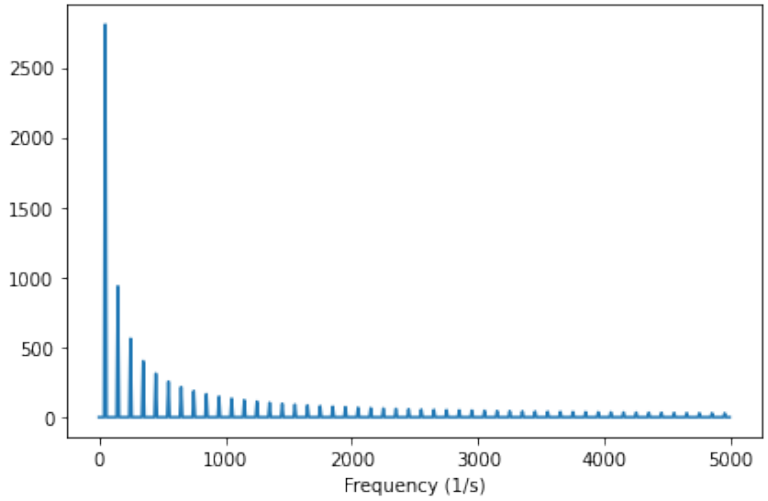

Рис. 3.3: Спектр

Интегрируем:

```
out_spectrum = in_spectrum.integrate()
out_spectrum.hs[0] = 0
out_spectrum.plot(high=2000)
decorate(xlabel='Frequency (1/s)')
```

Листинг 3.4: Проинтегрированный сигнал

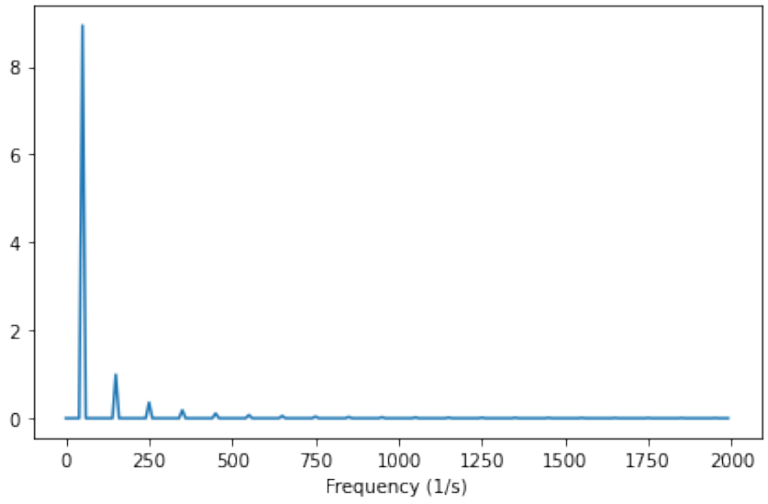

Рис. 3.4: Проинтегрированный сигнал

И превращаем в сигнал:

```
out_wave2 = out_spectrum.make_wave()
out_wave2.plot()
decorate(xlabel='Time (s)')
```

Листинг 3.5: Новый сигнал

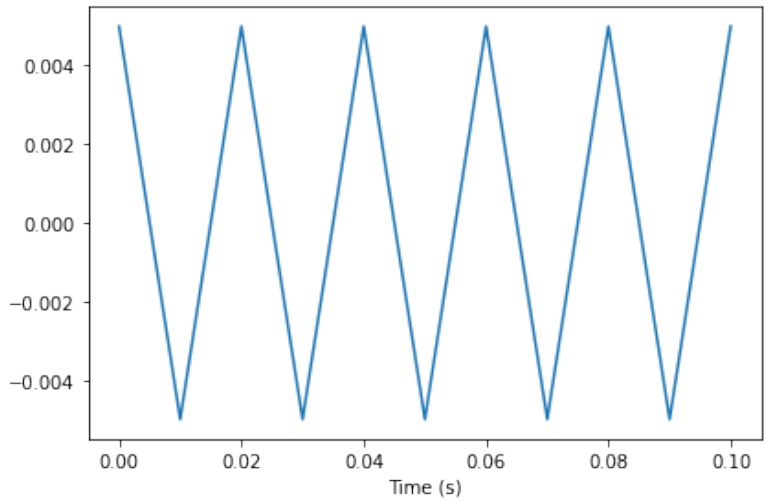

Рис. 3.5: Новый сигнал

А вот тут уже сигналы кажутся полностью одинаковыми, хотя и с точностью до линейной трансформации.

Глава 4

Двойное интегрирование на примере пилообразного сигнала

```
in_wave = SawtoothSignal(freq=50).make_wave(duration=0.1,
    framerate=44100)
in_wave.plot()
decorate(xlabel='Time (s)')
```

Листинг 4.1: Пилообразный сигнал

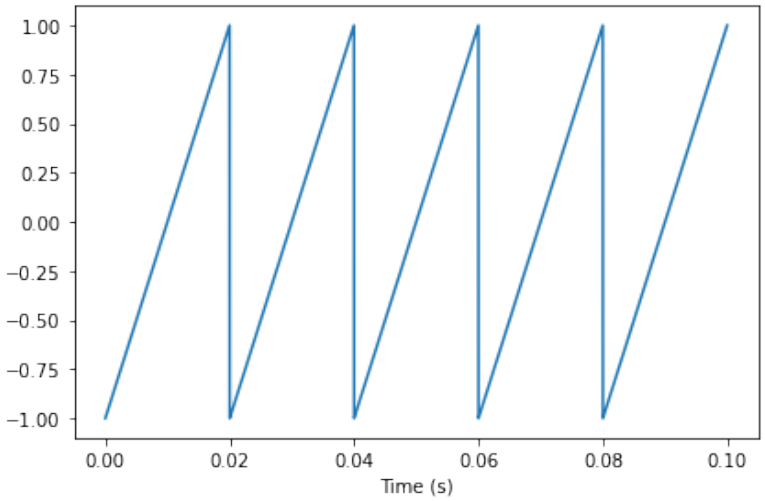

Рис. 4.1: Пилообразный сигнал

Считаем `cumsum`:

```
out_wave = in_wave.cumsum()
out_wave.unbias()
```

```
out_wave.plot()
decorate(xlabel='Time (s)')
```

Листинг 4.2: «Просуммированный»

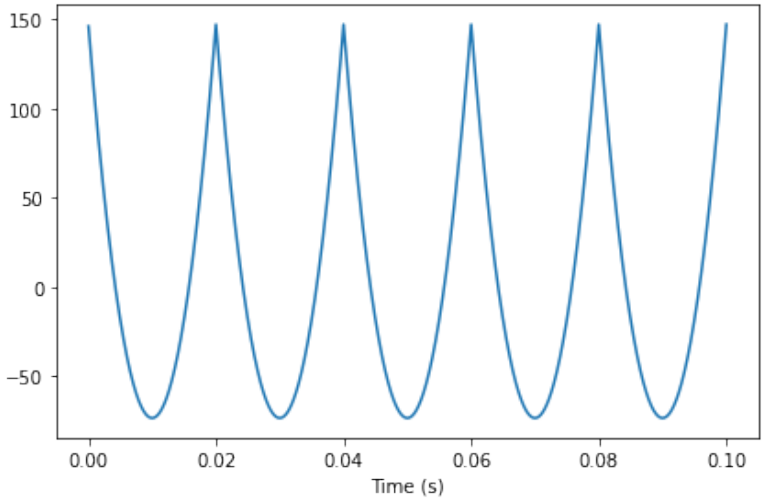

Рис. 4.2: «Просуммированный»

Еще считаем `cumsum`:

```
out_wave = out_wave.cumsum()
out_wave.plot()
decorate(xlabel='Time (s)')
```

Листинг 4.3: «Просуммированный x2»

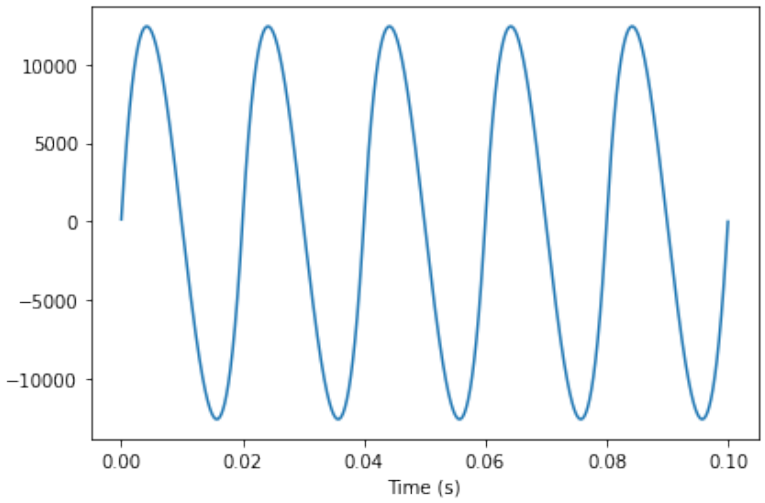

Рис. 4.3: «Просуммированный x2»

Теперь возьмем спектр:

```
in_spectrum = in_wave.make_spectrum()
in_spectrum.hs[0] = 0
in_spectrum.plot(high=5000)
decorate(xlabel='Frequency (1/s)')
```

Листинг 4.4: Спектр

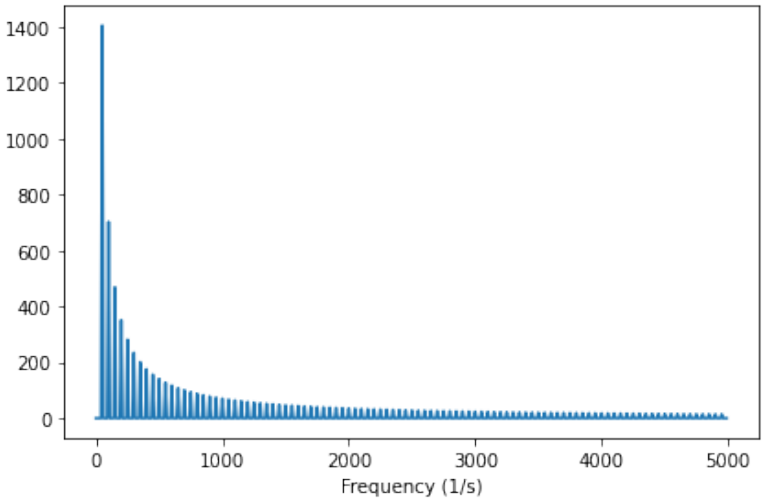

Рис. 4.4: Спектр

Интегрируем:

```
out_spectrum = in_spectrum.integrate()
out_spectrum.hs[0] = 0
out_spectrum.plot(high=2000)
decorate(xlabel='Frequency (1/s)')
```

Листинг 4.5: Проинтегрированный спектр

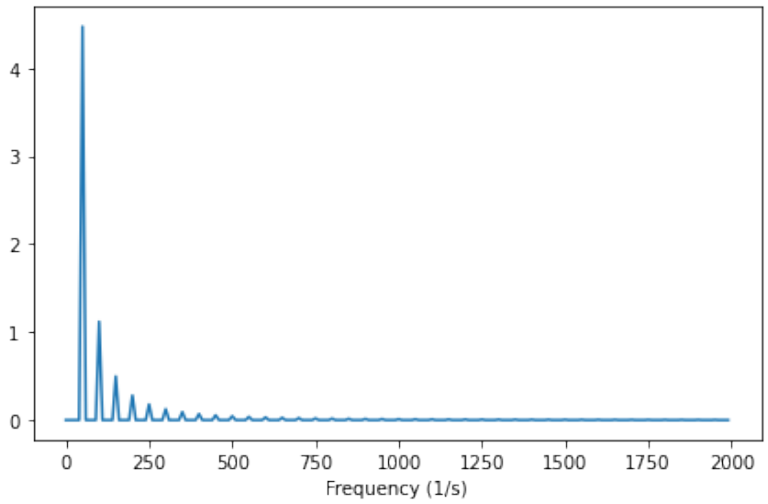

Рис. 4.5: Проинтегрированный спектр

Снова интегрируем:

```
out_spectrum = out_spectrum.integrate()
out_spectrum.hs[0] = 0
out_spectrum.plot(high=500)
decorate(xlabel='Frequency (1/s)')
```

Листинг 4.6: Проинтегрированный спектр x2

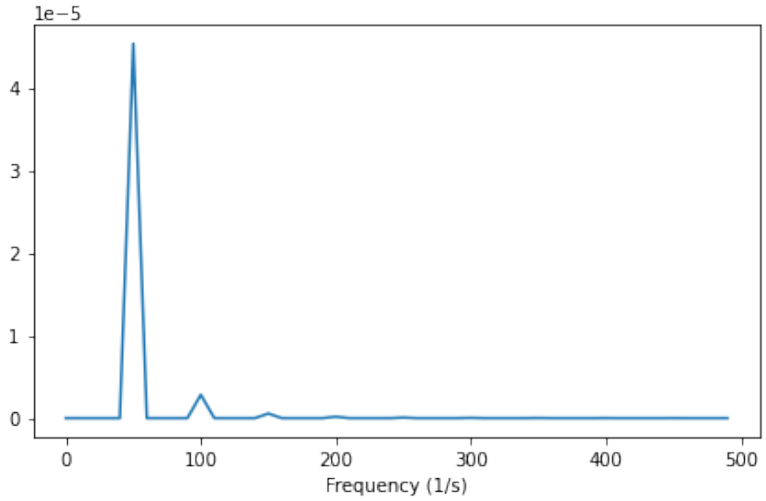

Рис. 4.6: Проинтегрированный спектр x2

И превращаем в сигнал:

```
out_wave2 = out_spectrum.make_wave()
out_wave2.plot()
decorate(xlabel='Time (s)')
```

Листинг 4.7: Новый сигнал

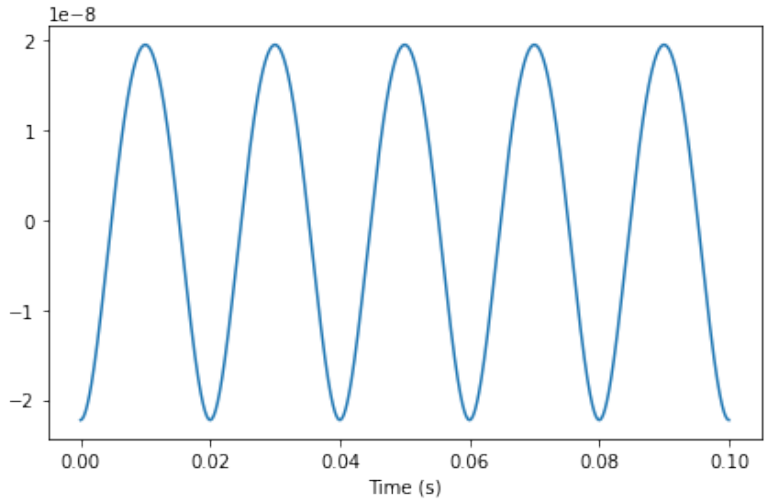

Рис. 4.7: Новый сигнал

Тут «вторая версия» более похожа на синусоиду, чем первая. Можно вот так вот сравнить еще:

```
out_wave.normalize()
out_wave.unbias()
out_wave.plot()

out_wave2.normalize()
out_wave2.unbias()
out_wave2.plot()

decorate(xlabel='Time (s)')
```

Листинг 4.8: Сравнение

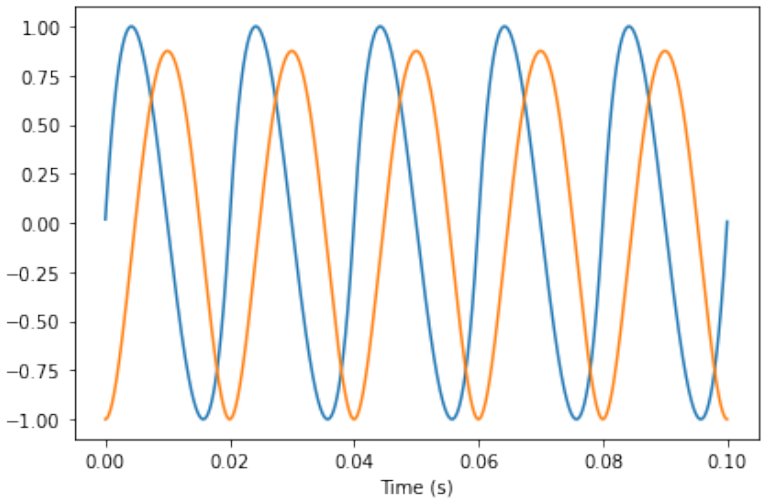

Рис. 4.8: Сравнение

Можно видеть, что первая чуть сильнее наклоняется влево. Впрочем, ни то, ни то не есть синусоида, просто большие частоты подавляются, но не полностью идеально, поэтому нам может казаться, что это синусоиды.

Глава 5

Двойное дифференциирование на примере кубического сигнала

```
from thinkdsp import CubicSignal
in_wave = CubicSignal(freq=0.0005).make_wave(duration=10000,
    framerate=1)
in_wave.plot()
```

Листинг 5.1: Кубический сигнал

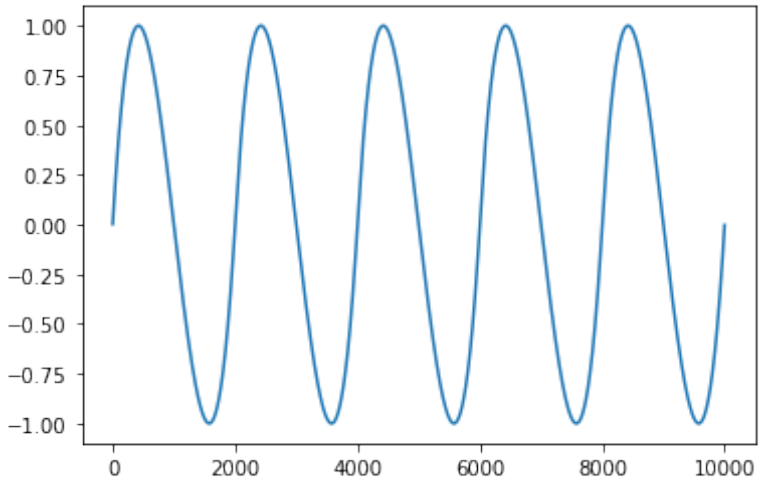

Рис. 5.1: Кубический сигнал

Перед тем, как продолжить, хотелось бы сразу посмотреть на спектр:

```
in_wave.make_spectrum().plot(high=0.006)
```

Листинг 5.2: Спектр

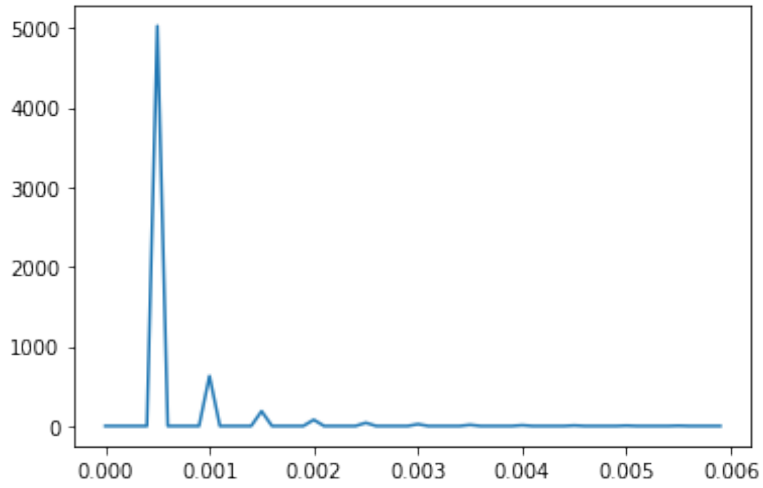

Рис. 5.2: Спектр

Ага, это уже больше похоже на результаты прошлого раздела. У нас есть некоторая значимая низкая частота, и «что-то там еще» после нее, и в итоге сигнал кажется наклоненным. Впрочем, вот так вот «на глаз» нельзя говорить, что мы получили в прошлом разделе именно кубический сигнал, кривизну нужно доказывать.

Посчитаем `diff`:

```
1 out_wave = in_wave.diff()
2 out_wave.plot()
```

Листинг 5.3: Применили `diff`

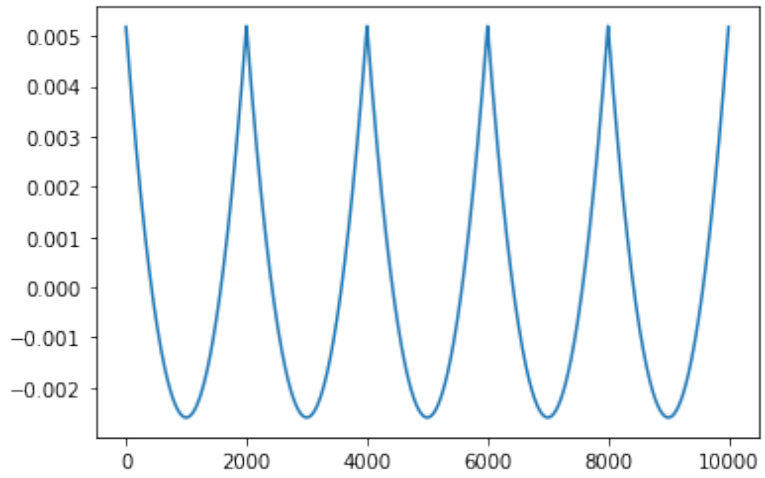

Рис. 5.3: Применили `diff`

Еще считаем `diff`:

```
out_wave = out_wave.diff()
out_wave.plot()
```

Листинг 5.4: Применили `diff` x2

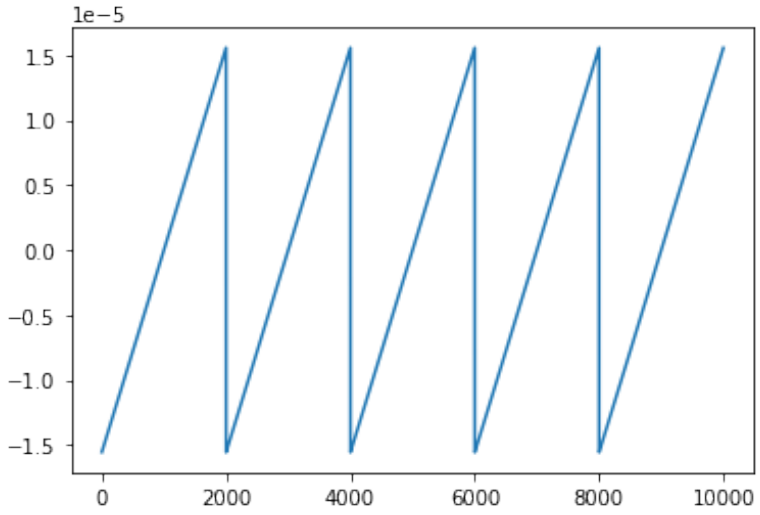

Рис. 5.4: Применили `diff` x2

Теперь возьмем спектр:

```
in_spectrum = in_wave.make_spectrum()
in_spectrum.plot(high=0.01)
decorate(xlabel='Frequency (1/s)')
```

Листинг 5.5: Спектр

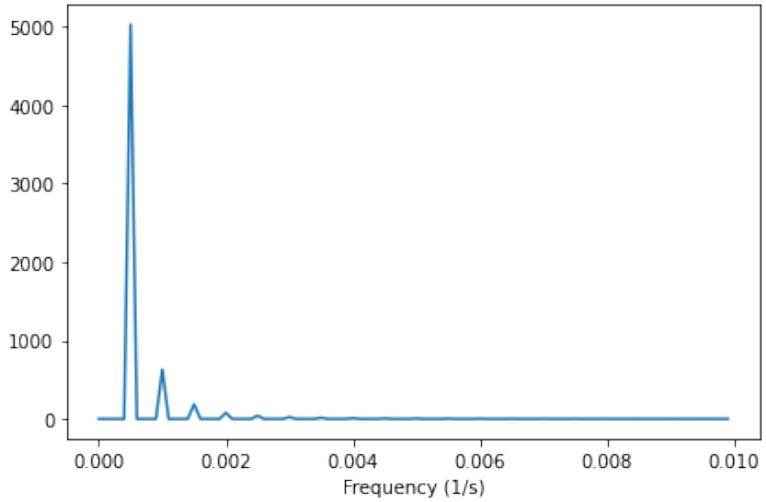

Рис. 5.5: Спектр

Дифференциируем:

```
out_spectrum = in_spectrum.differentiate()
out_spectrum.plot(high=0.01)
decorate(xlabel='Frequency (1/s)')
```

Листинг 5.6: Продифференциированный спектр

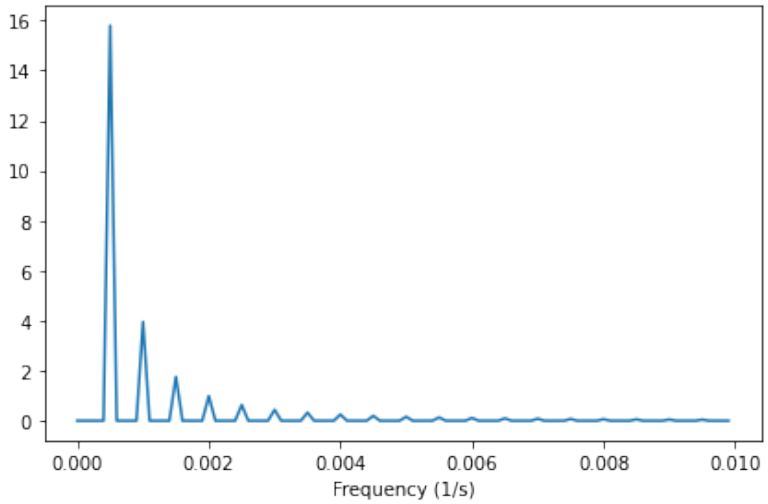

Рис. 5.6: Продифференциированный спектр

Снова дифференциируем:

```
out_spectrum = out_spectrum.differentiate()
out_spectrum.plot(high=0.04)
decorate(xlabel='Frequency (1/s)')
```

Листинг 5.7: Продифференциированный спектр x2

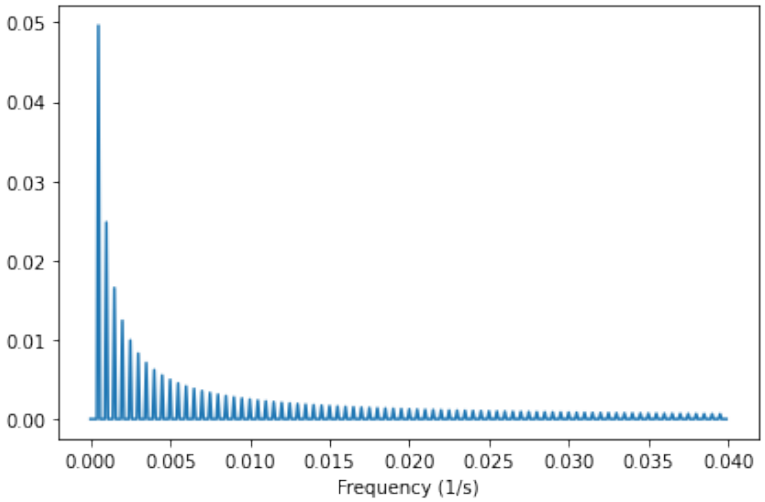

Рис. 5.7: Продифференциированный спектр x2

И превращаем в сигнал:

```
out_wave2 = out_spectrum.make_wave()
out_wave2.plot()
decorate(xlabel='Time (s)')
```

Листинг 5.8: Новый сигнал

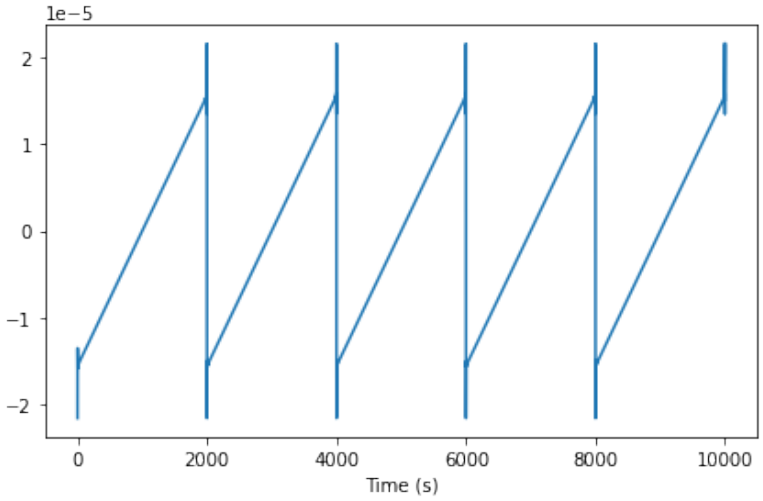

Рис. 5.8: Новый сигнал

Да, пожалуй, это тоже пилообразный сигнал, хотя мы наблюдаем эффект, аналогичный первому разделу с неточной аппроксимацией.

Рассмотрим соответствующие фильтры. Из Вычислительной математики знаем окно, соответствующее второй численной производной, так что просто берем ее DFT.

```
from thinkdsp import zero_pad

diff_window = np.array([-1.0, 2.0, -1.0])
padded = zero_pad(diff_window, len(in_wave))
diff_wave = Wave(padded, framerate=in_wave.framerate)

diff_filter = diff_wave.make_spectrum()
diff_filter.plot(label='2nd diff')
decorate(xlabel='Frequency (Hz)', ylabel='Amplitude ratio')
```

Листинг 5.9: Фильтр для `difference` x2

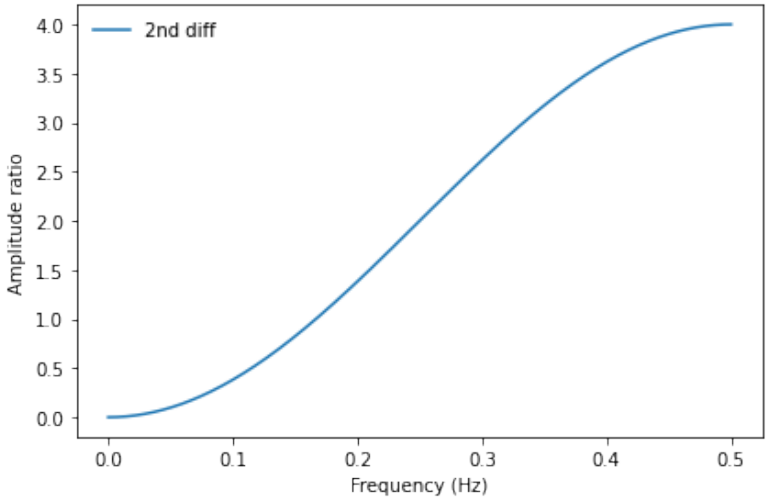

Рис. 5.9: Фильтр для `difference` x2

Фильтр для второй производной есть квадрат фильтра первой.

```
deriv_filter = in_wave.make_spectrum()
deriv_filter.hs = (PI2 * 1j * deriv_filter.fs)**2
deriv_filter.plot(label='2nd deriv')
decorate(xlabel='Frequency (Hz)', ylabel='Amplitude ratio')
```

Листинг 5.10: Фильтр для `differentiate` x2

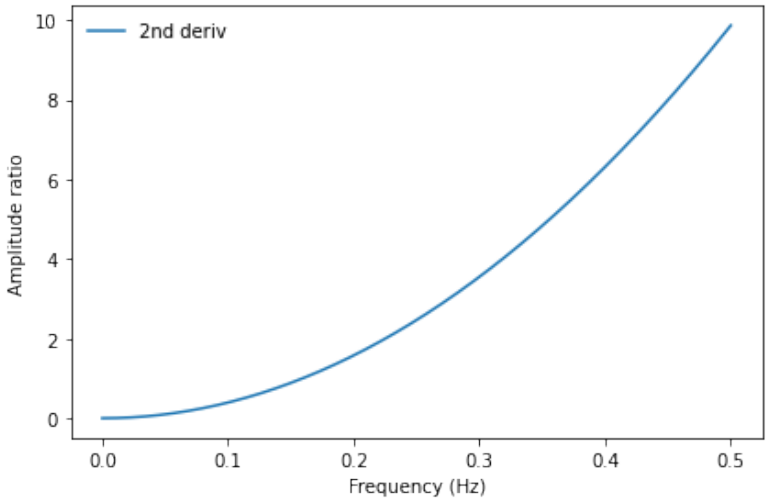

Рис. 5.10: Фильтр для `differentiate` x2

Удобно смотреть вместе на оба фильтра.

```
diff_filter.plot(label='2nd diff')
deriv_filter.plot(label='2nd deriv')
decorate(xlabel='Frequency (Hz)', ylabel='Amplitude ratio')
```

Листинг 5.11: Сравнение

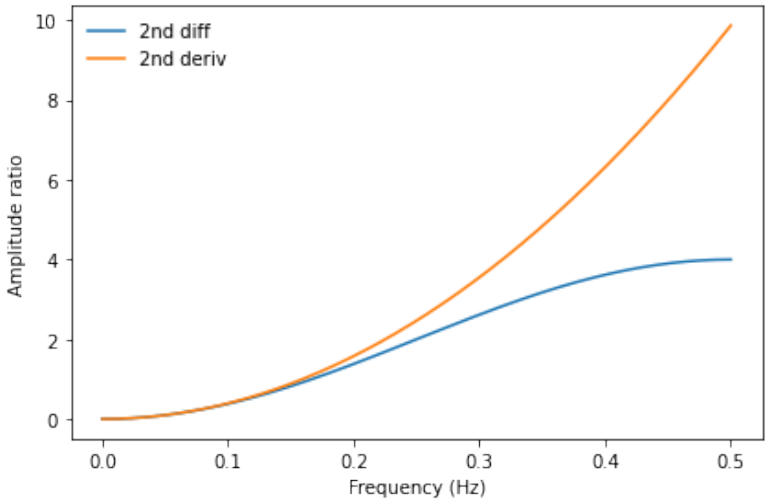

Рис. 5.11: Сравнение

Поначалу они очень похожи, но вот для больших частот начинают сильно расходиться.