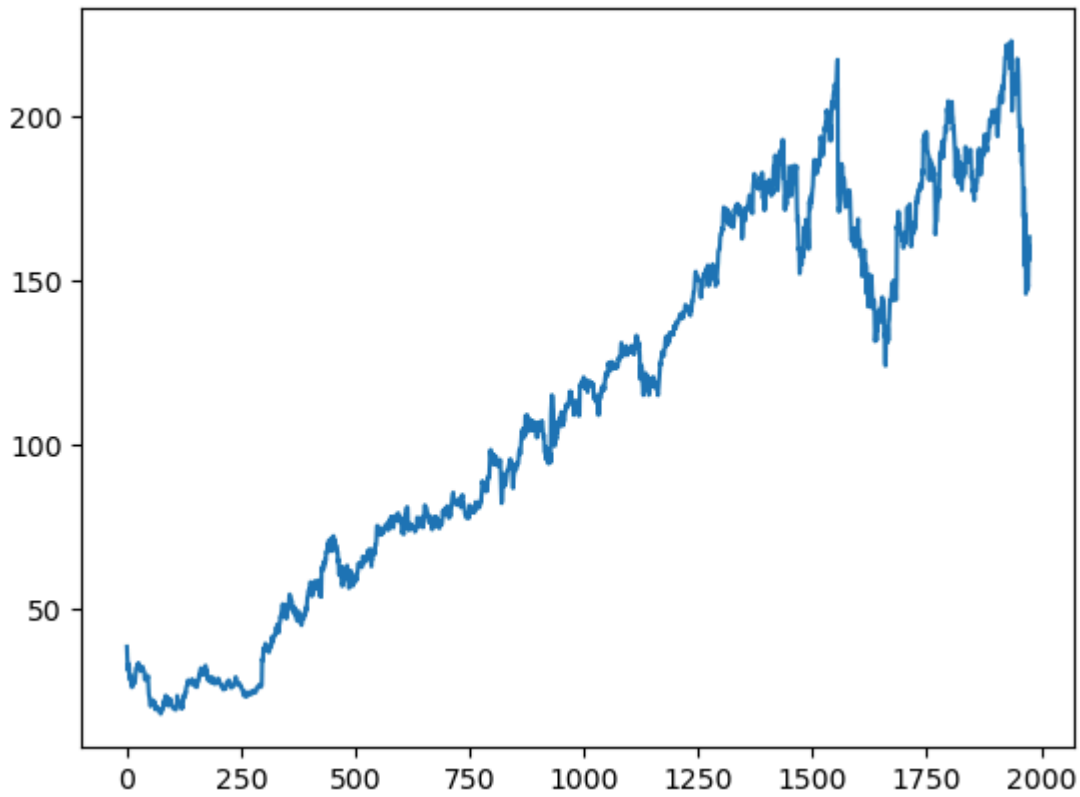
Лабораторная работа 9

Упражнение 9.1

В разделе «Нарастающая сумма» в методических материалах было отмечено, что данные примеры работают только с периодическими сигналами. Проверим это, подав вместо пилообразного сигнала данные с биржи о компании Facebook:

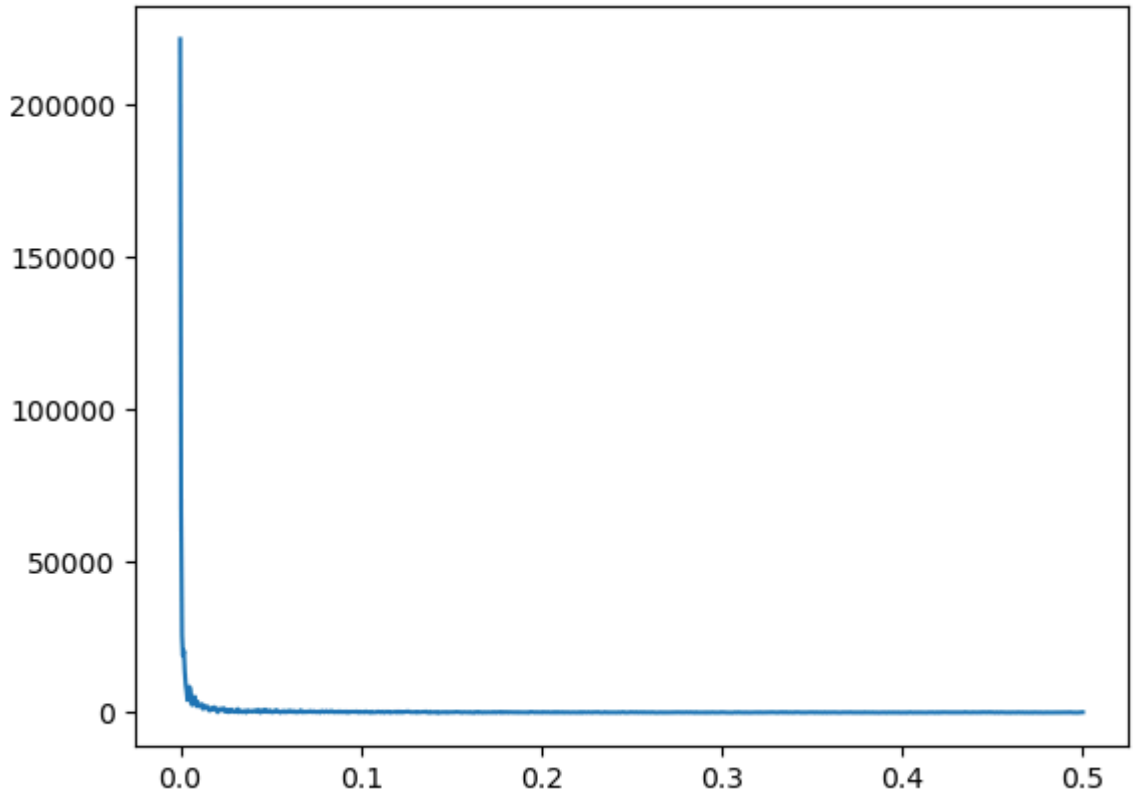
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from thinkdsp import *
```

In [ ]:
```
df = pd.read_csv('FB_2.csv', header=0, parse_dates=[0])
ys = df['Close']
if len(ys) % 2:
    ys = ys[:-1]

in_wave = Wave(ys, framerate=1)
in_wave.plot()
```



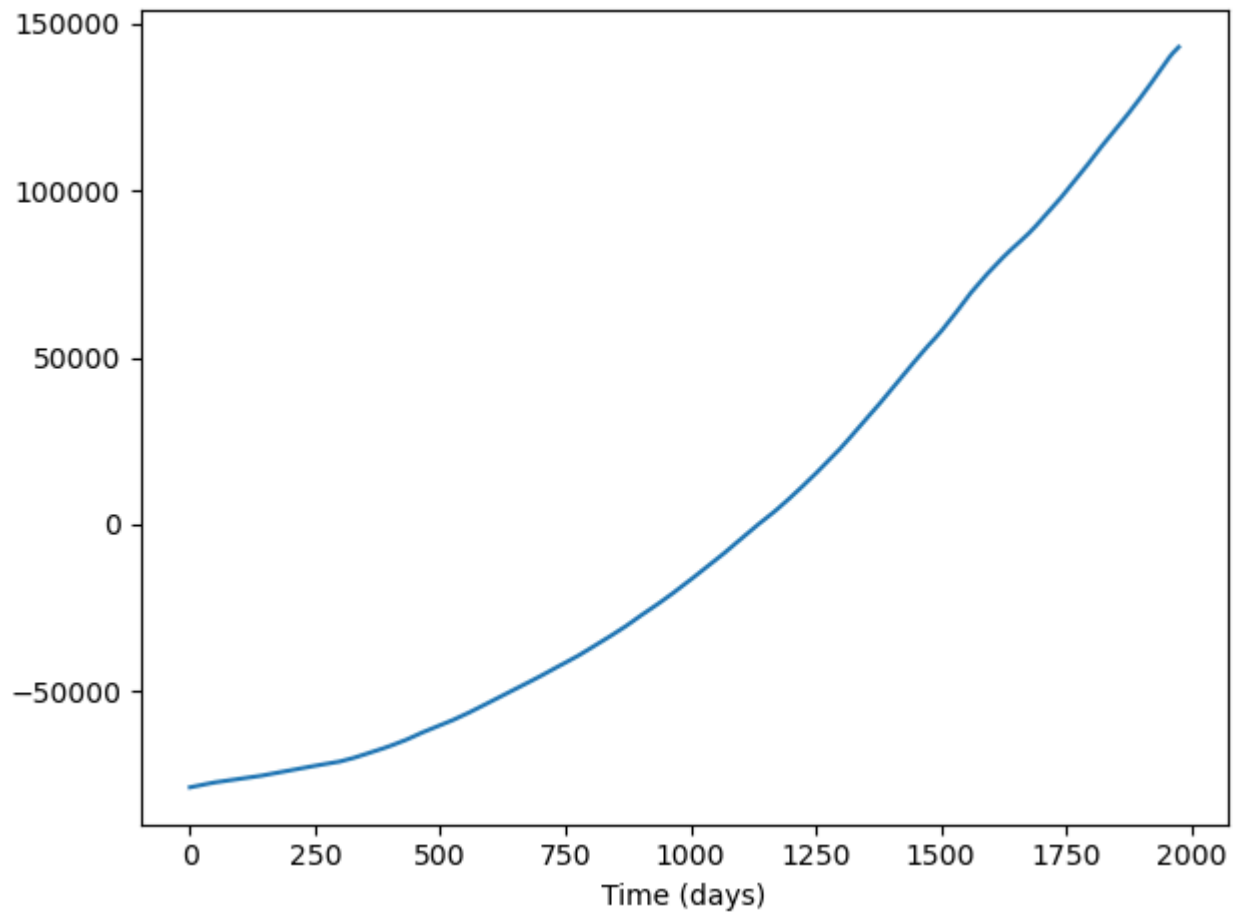
Также построим спектр волны

In [ ]:
```
in_spectrum = in_wave.make_spectrum()
in_spectrum.plot()
```

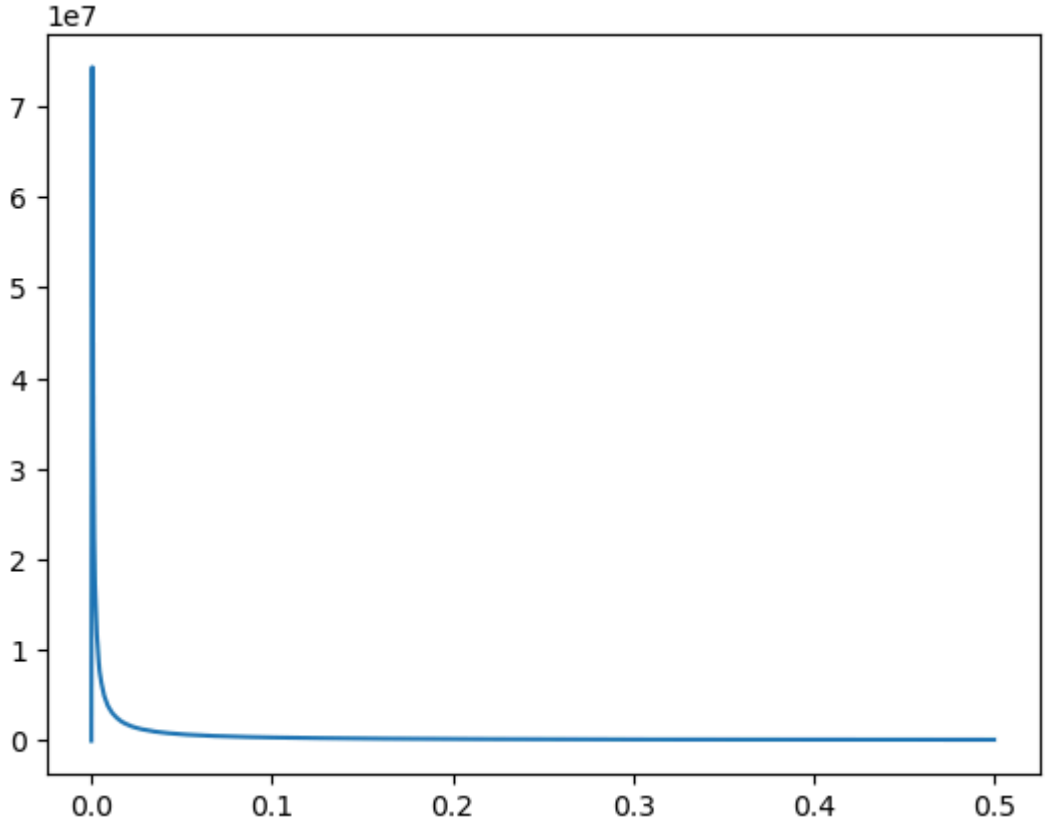


Теперь построим нарастающую сумму сигнала, нарастающая сумма - аппроксимация интегрирования.

In [ ]:
```
out_wave = in_wave.cumsum()
out_wave.unbias()
out_wave.plot()
decorate(xlabel='Time (days)')
```

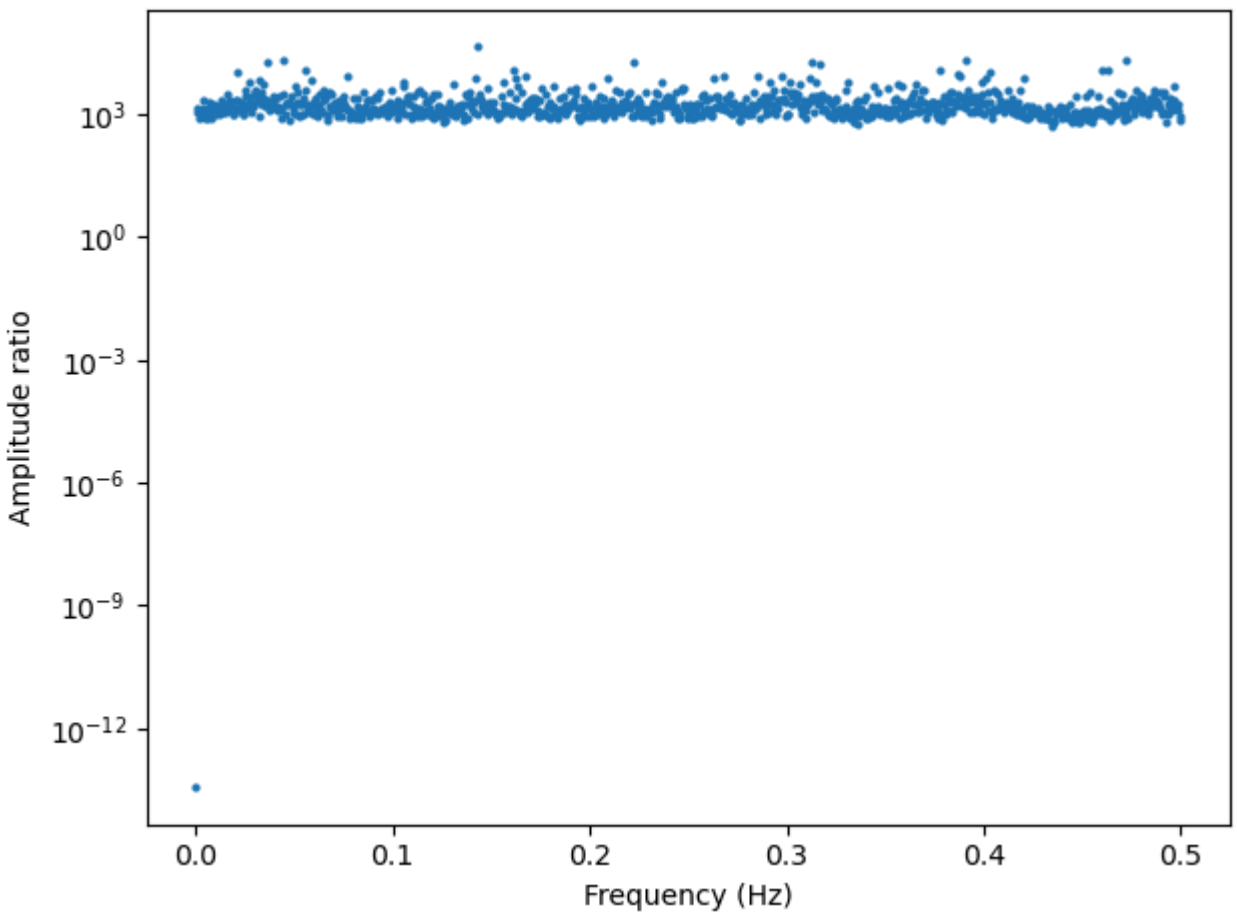


In [ ]:
```
out_spectrum = out_wave.make_spectrum()
out_spectrum.plot()
```



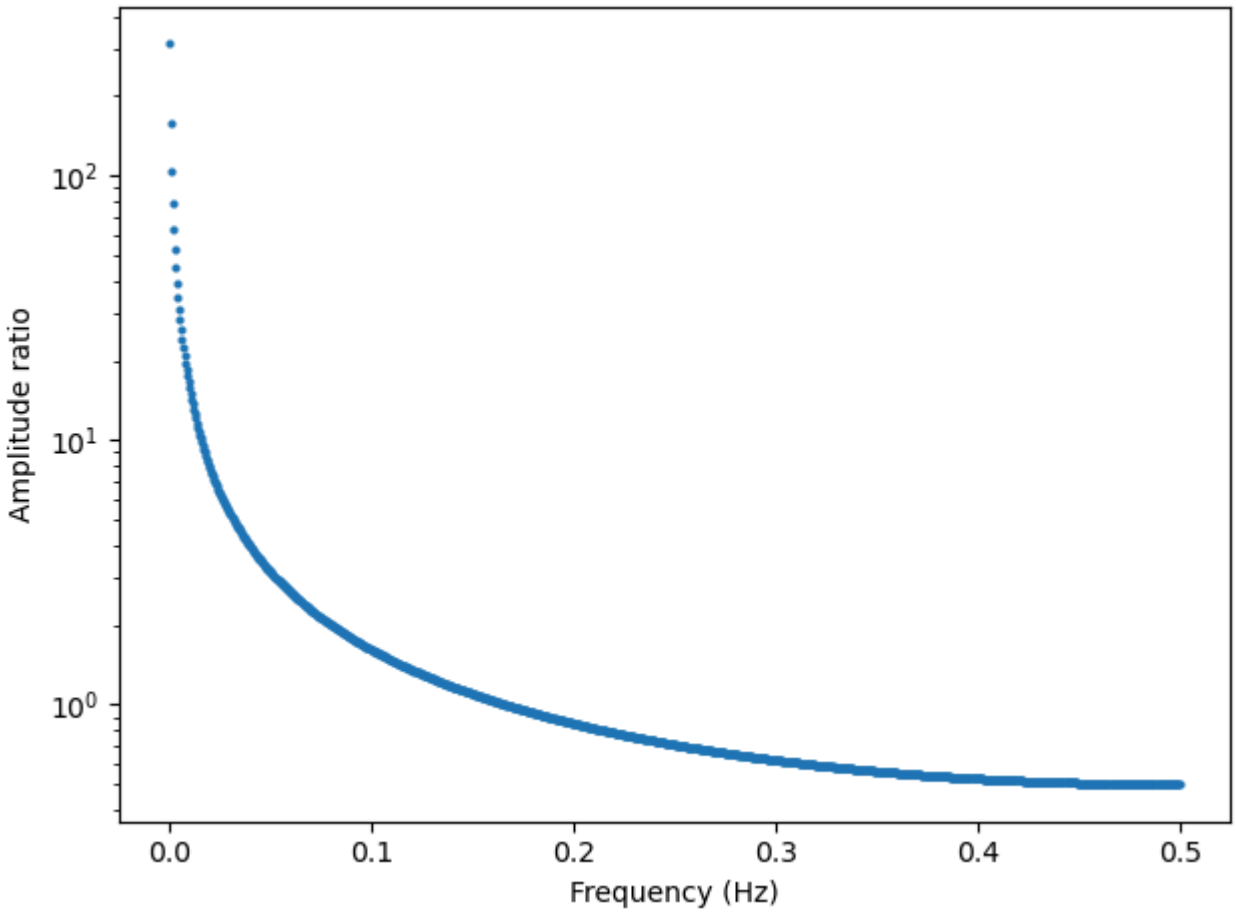
Сравнивая с графиком нарастающей суммы, можем заметить, что график не циклический. Это связано с сигналом - в примере используется циклический пилообразный сигнал.

In [ ]:
```
in_spectrum = in_wave.make_spectrum()
ratio_spectrum = out_spectrum.ratio(in_spectrum, thresh=1)
ratio_spectrum.plot(marker='.', ms=4, ls='')

decorate(xlabel='Frequency (Hz)',
         ylabel='Amplitude ratio',
         yscale='log')
```



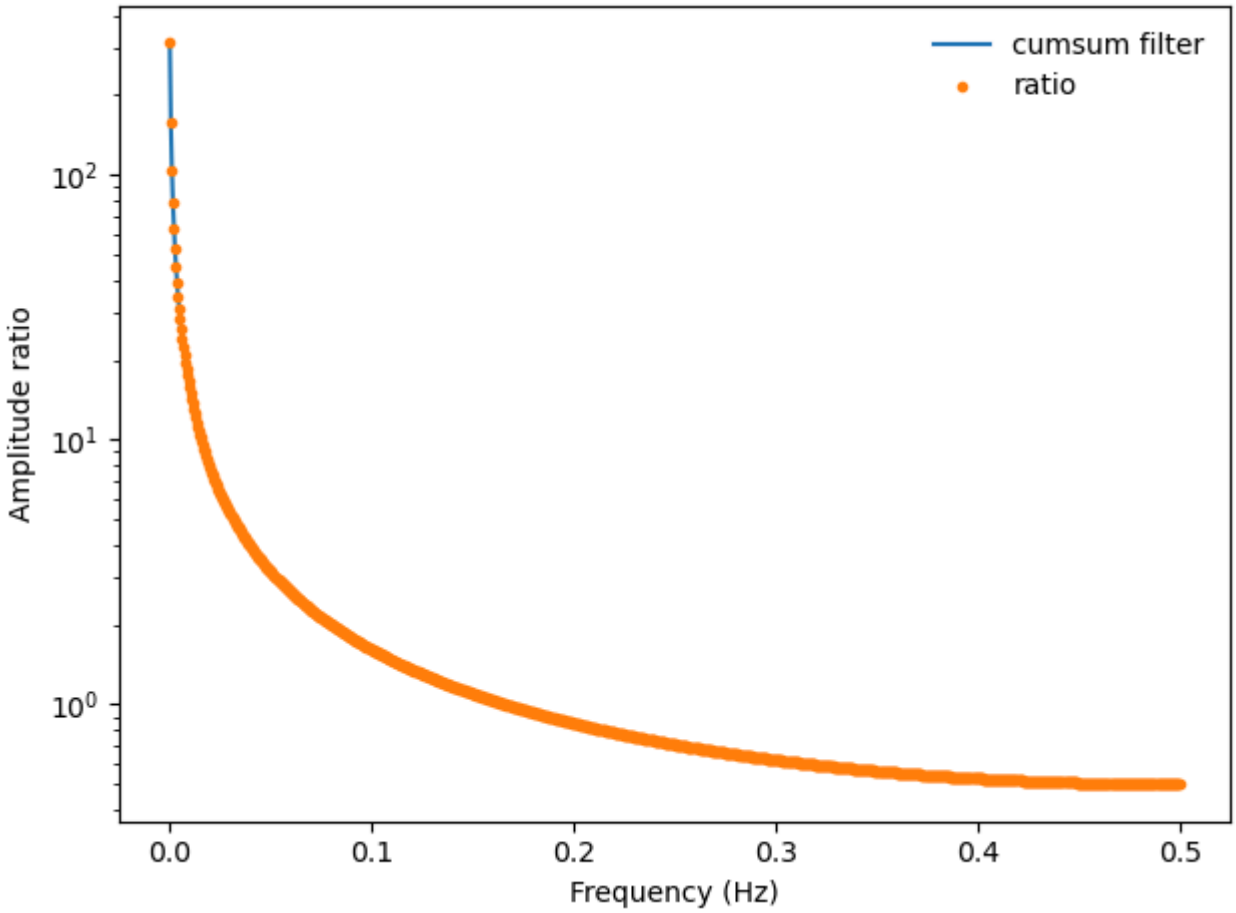
Чтобы наблюдать корректное отношение между спектрами входного и выходного сигналов, нормализуем их.

In [ ]:
```
in_wave = Wave(ys, framerate=1)
in_wave.unbias()
out_wave = in_wave.cumsum()
out_spectrum = out_wave.make_spectrum()
in_spectrum = in_wave.make_spectrum()
ratio_spectrum = out_spectrum.ratio(in_spectrum, thresh=1)

ratio_spectrum.plot(marker='.', ms=4, ls='')

decorate(xlabel='Frequency (Hz)',
         ylabel='Amplitude ratio',
         yscale='log')
```



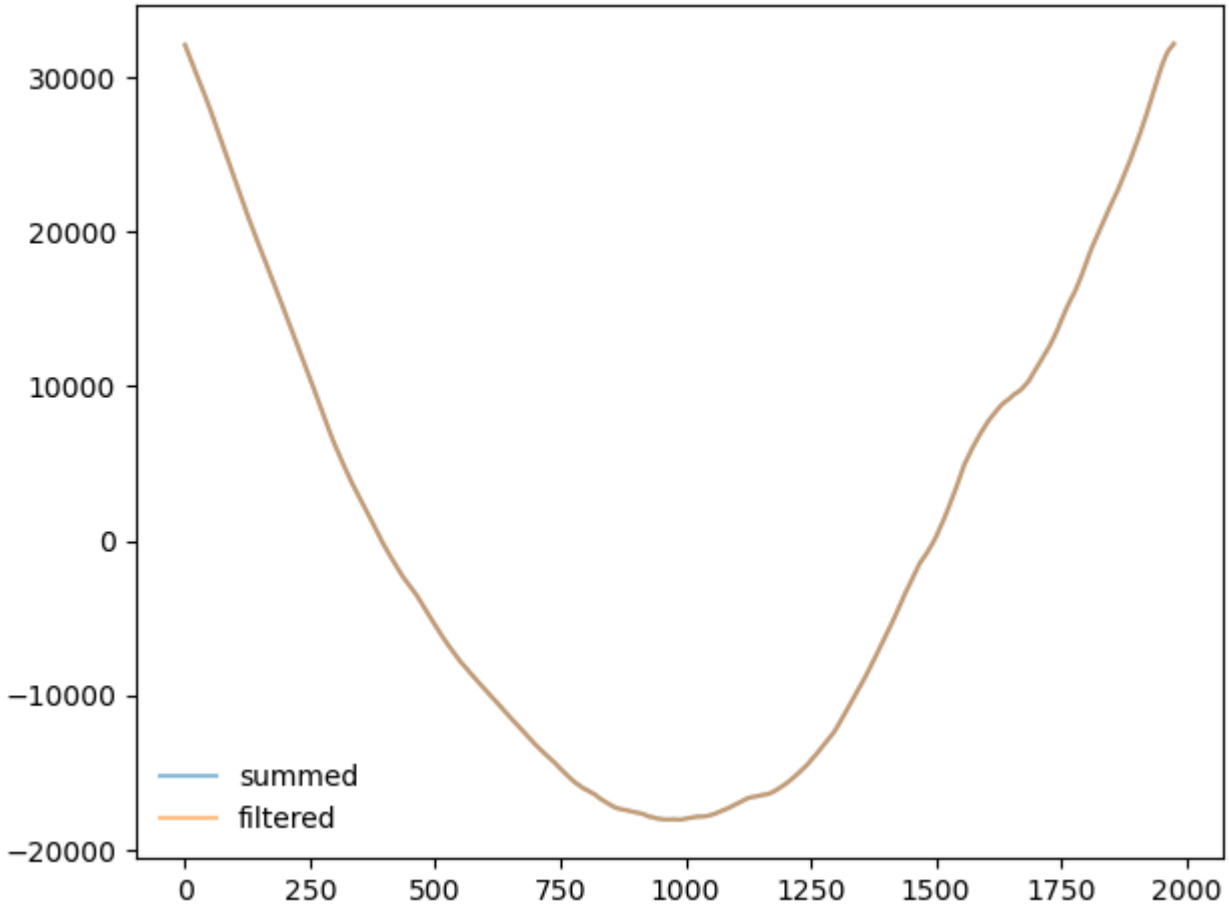
Также мы можем сравнить вычисленные отношения с фильтром накапливающей суммы. Чтобы получить фильтр накапливающей суммы, можем вычислить дифференциальный фильтр и взять обратное от него

In [ ]:
```
diff_window = np.array([1.0, -1.0])
padded = zero_pad(diff_window, len(in_wave))
diff_wave = Wave(padded, framerate=in_wave.framerate)
diff_filter = diff_wave.make_spectrum()

cumsum_filter = diff_filter.copy()
cumsum_filter.hs[1:] = 1 / cumsum_filter.hs[1:]
cumsum_filter.hs[0] = np.inf

cumsum_filter.plot(label='cumsum filter')
ratio_spectrum.plot(marker='.', label='ratio', ls='')

decorate(xlabel='Frequency (Hz)',
         ylabel='Amplitude ratio',
         yscale='log')
```



Также проверим теорему о свертке, применив фильтр в частотной области

In [ ]:
```
in_wave = Wave(ys, framerate=1)
in_wave.unbias()
out_wave = in_wave.cumsum()
out_wave.unbias()

out_wave.plot(label='summed', alpha=0.5)
cumsum_filter.hs[0] = 0
out_wave2 = (in_spectrum * cumsum_filter).make_wave()
out_wave2.unbias()
out_wave2.plot(label='filtered', alpha=0.5)
decorate()
```

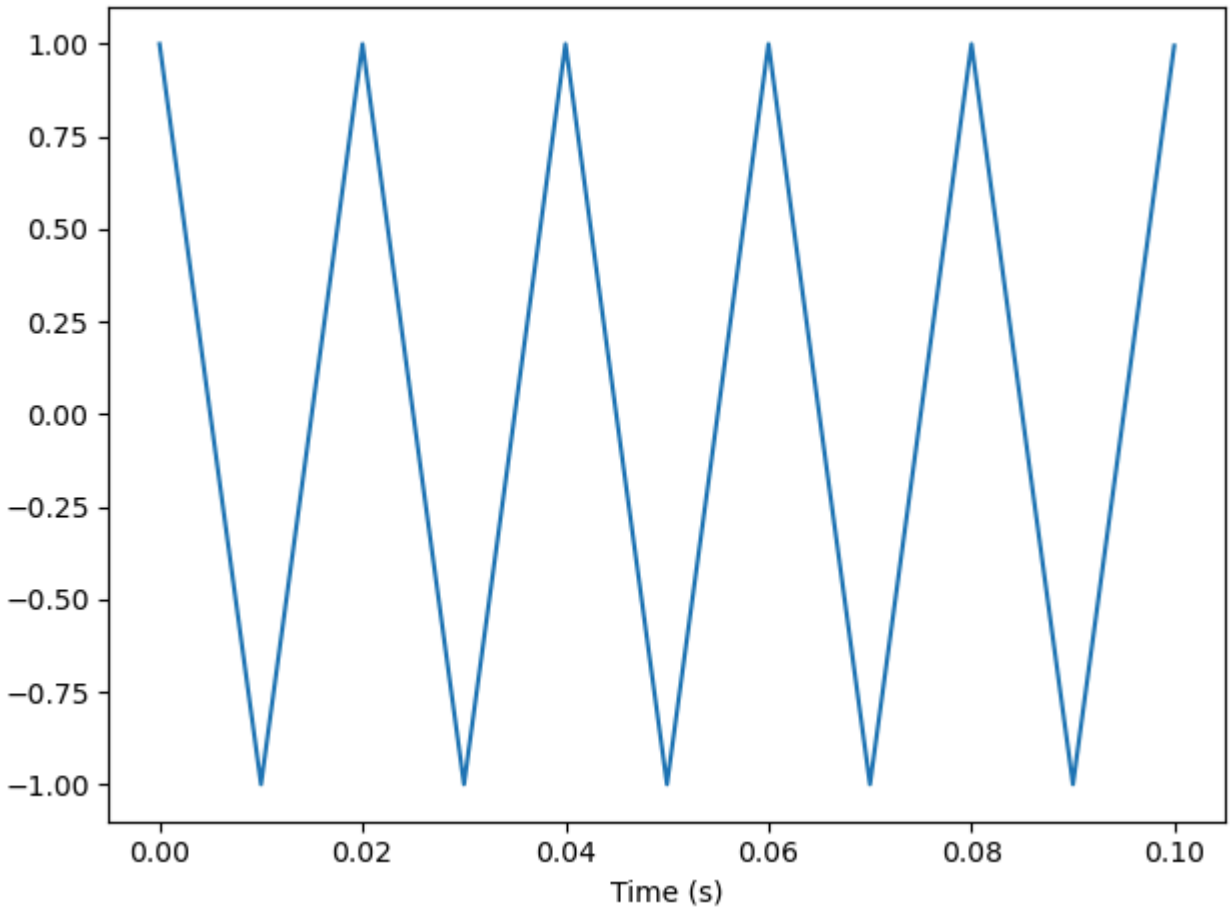


Как мы можем заметить полученные кривые совпадают с точностью до погрешности вычислений. Таким образом, можем сделать вывод, что для реальных сигналов метод работает корректно при условии, что сигналы нормализованы

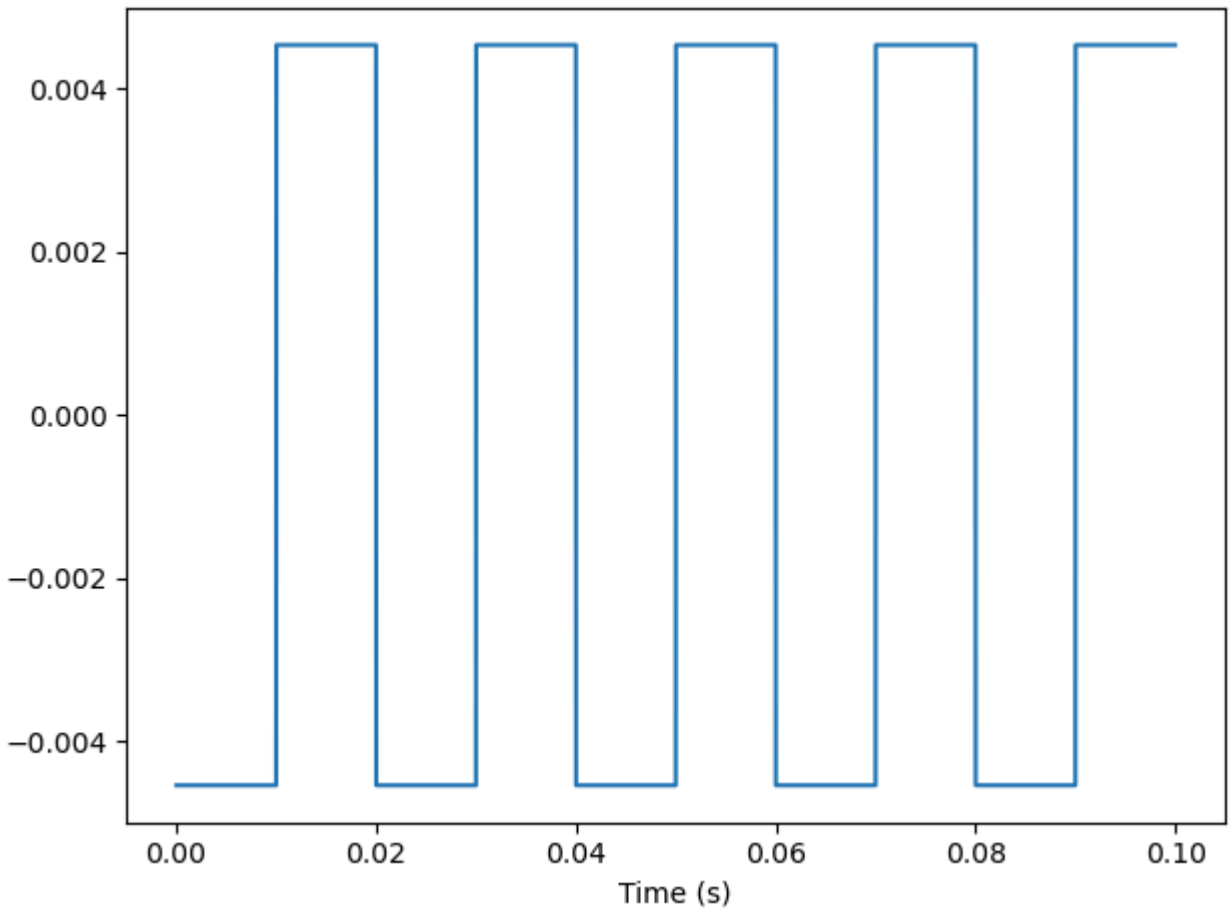
Упражнение 9.2

> В этом упражнении изучается влияние `diff` и `differentiate` на сигнал. Создайте треугольный сигнал и напечатайте его. Примените `diff` к сигналу и напечатайте результат. Вычислите спектр треугольного сигнала, примените `differentiate` и напечатайте результат. Преобразуйте спектр обратно в сигнал и напечатайте его. Есть ли различия в воздействии `cumsum` и `integrate` на этот сигнал

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



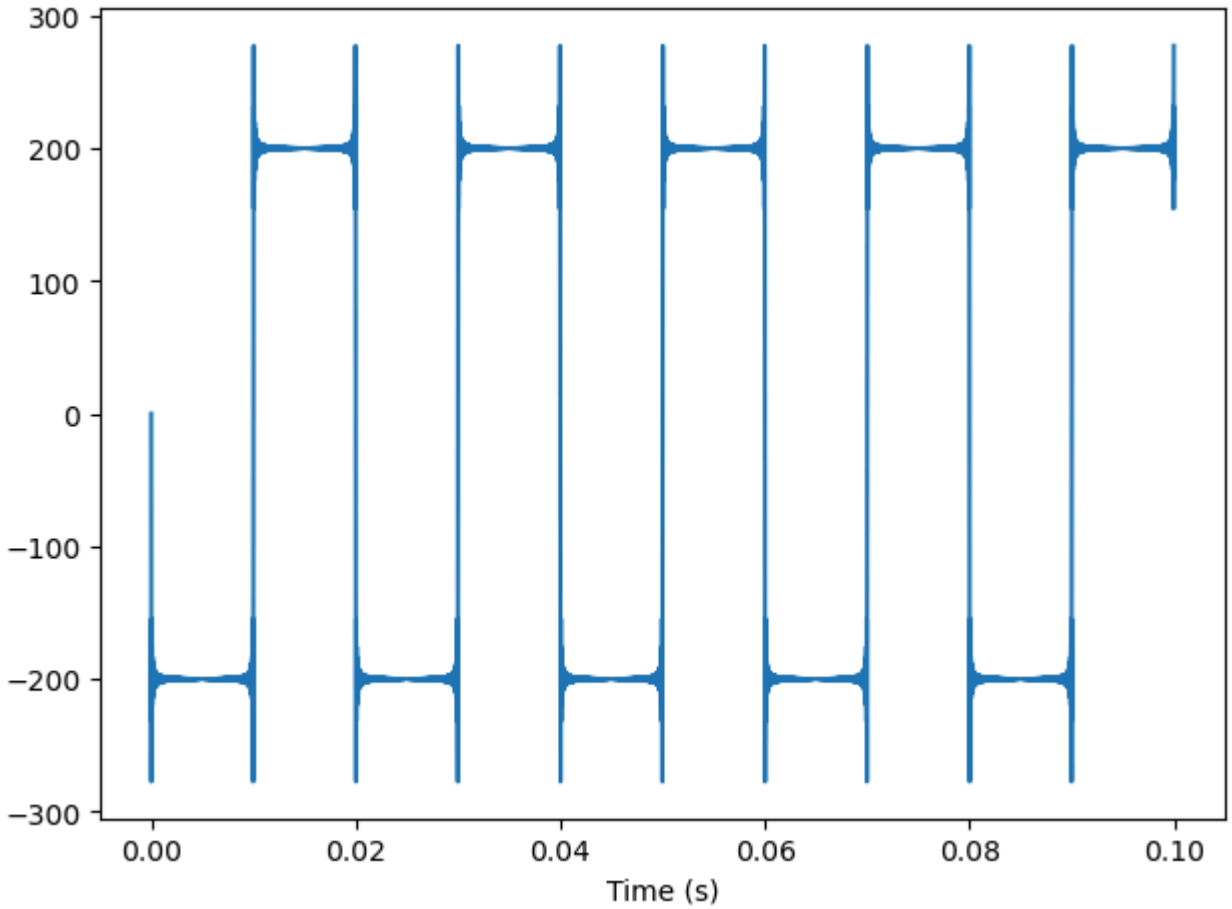
In [ ]:
```
out_wave = in_wave.diff()
out_wave.plot()
decorate(xlabel='Time (s)')
```



Результатом взятия производной от треугльного сигнала является прямоугольный сигнал, так как треугольный сигнал равномерно возрастает и спадает - это соответствует горизонтальным линиям прямоугольного сигнала.

Теперь применим `differentiate` к спектру треугольного сигнала

In [ ]:
```
out_wave2 = in_wave.make_spectrum().differentiate().make_wave()
out_wave2.plot()
decorate(xlabel='Time (s)')
```

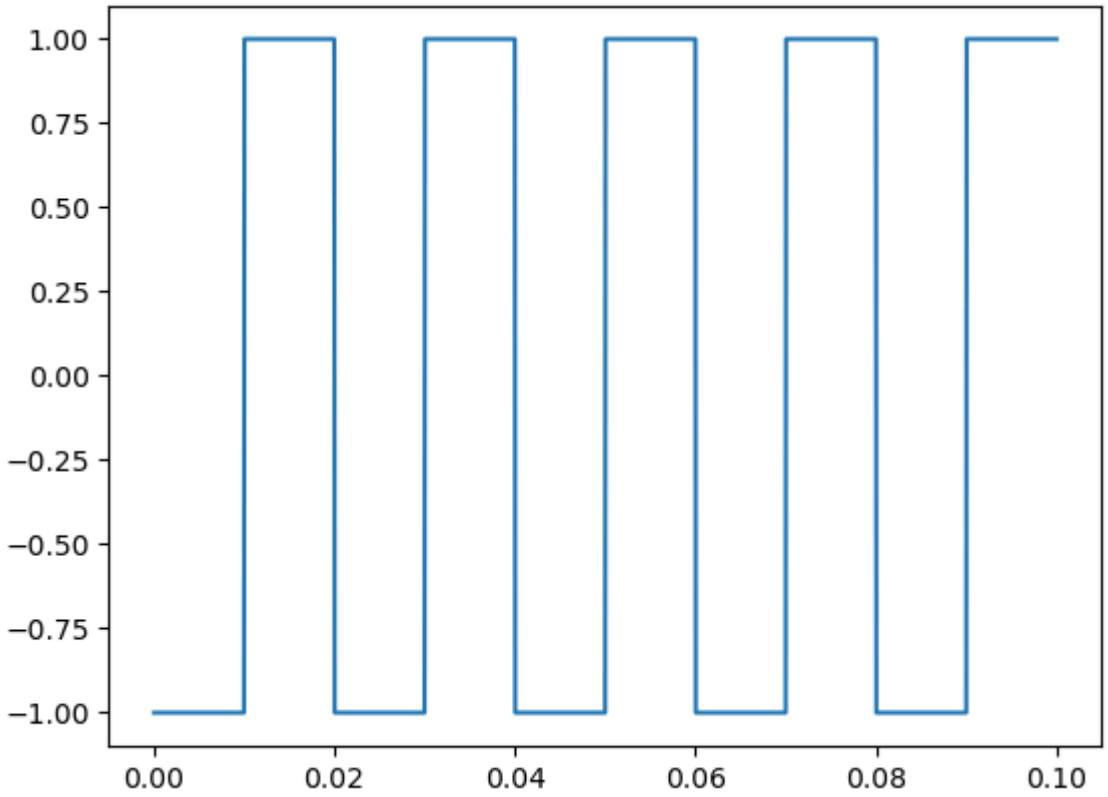


Полученный сигнал совпадает с прямоугольным, но мы получаем звон в местах перехода сигнала с одного уровня на другой. Это связано с тем, что производная треугольного сигнала не определена в местах перегиба

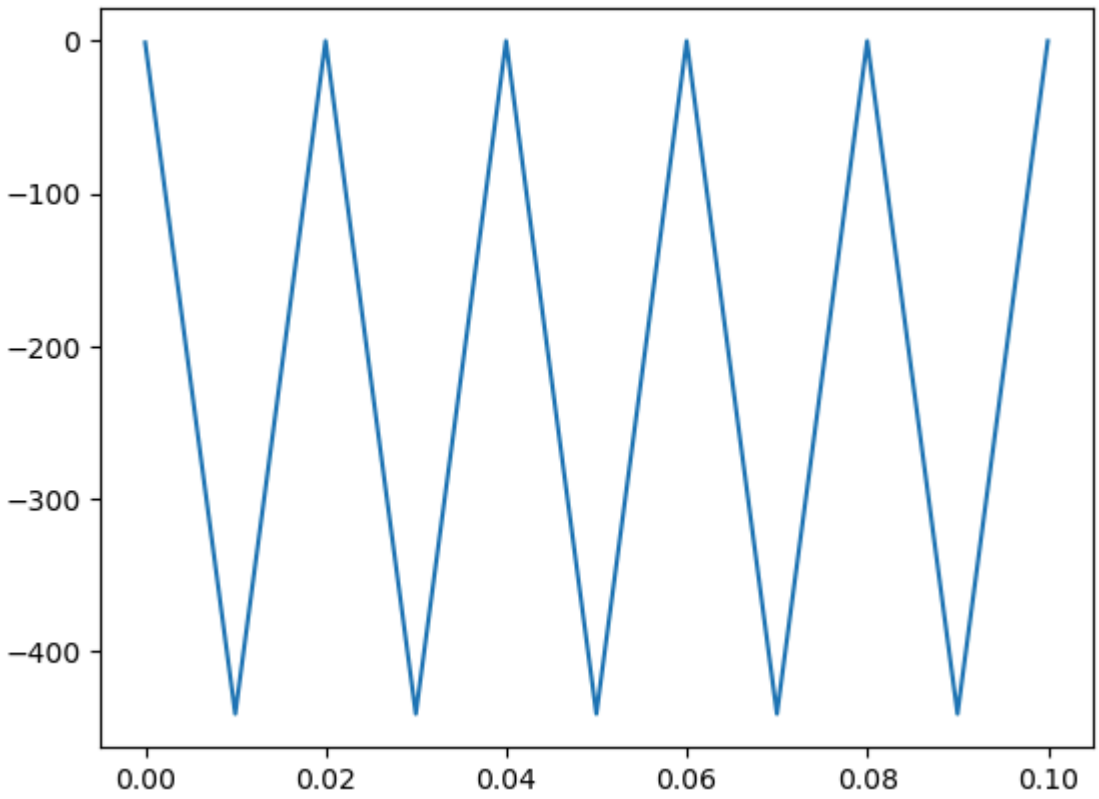
Упражнение 9.3

> В данном упражнении изучается влияние `cumsum` и `integrate` на сигнал. Создайте прямоугоьлный сигнал и напечатайте его. Примените `cumsum` и напечатайте результат. Преобразуйте спектр прямоугольного сигнала, примените `integrate` и напечатайте результат. Преобразуйте спектр обратно в сигнал и напечатайте его. Есть ли различия в воздействии `cumsum` и `integrate` на этот сигнал?

In [ ]:
```
in_wave = SquareSignal(freq=50).make_wave(duration=0.1, framerate=44100)
in_wave.plot()
```

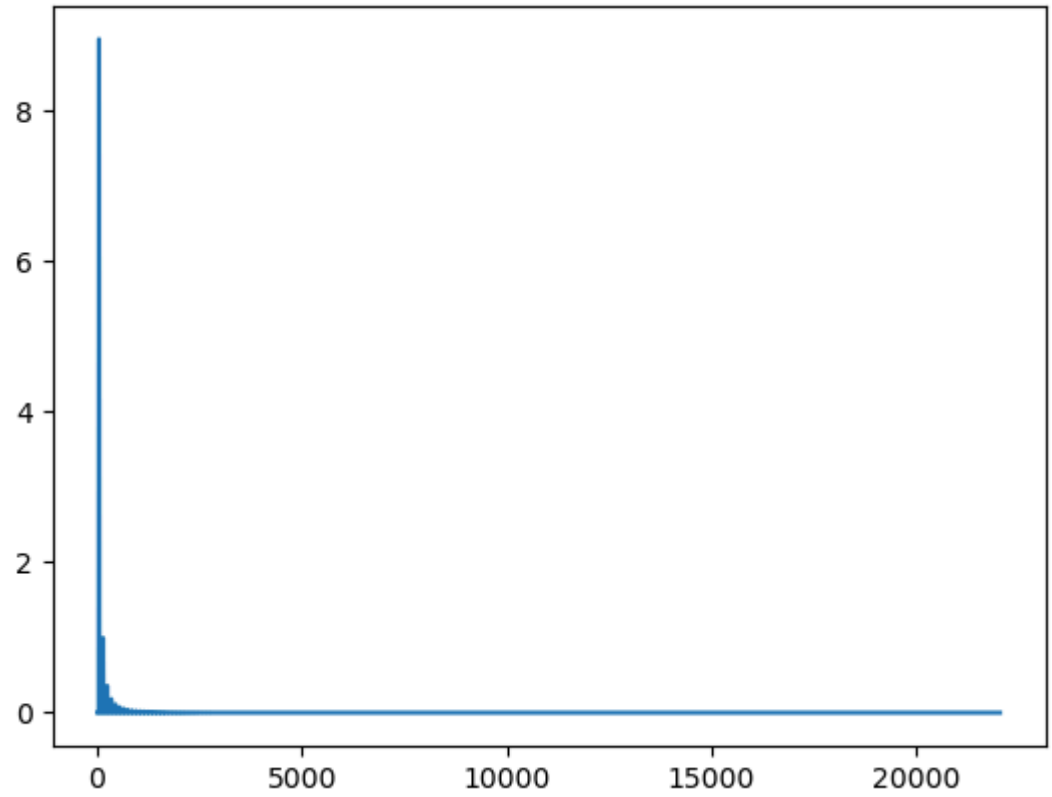


In [ ]:
```
out_wave = in_wave.cumsum()
out_wave.plot()
```

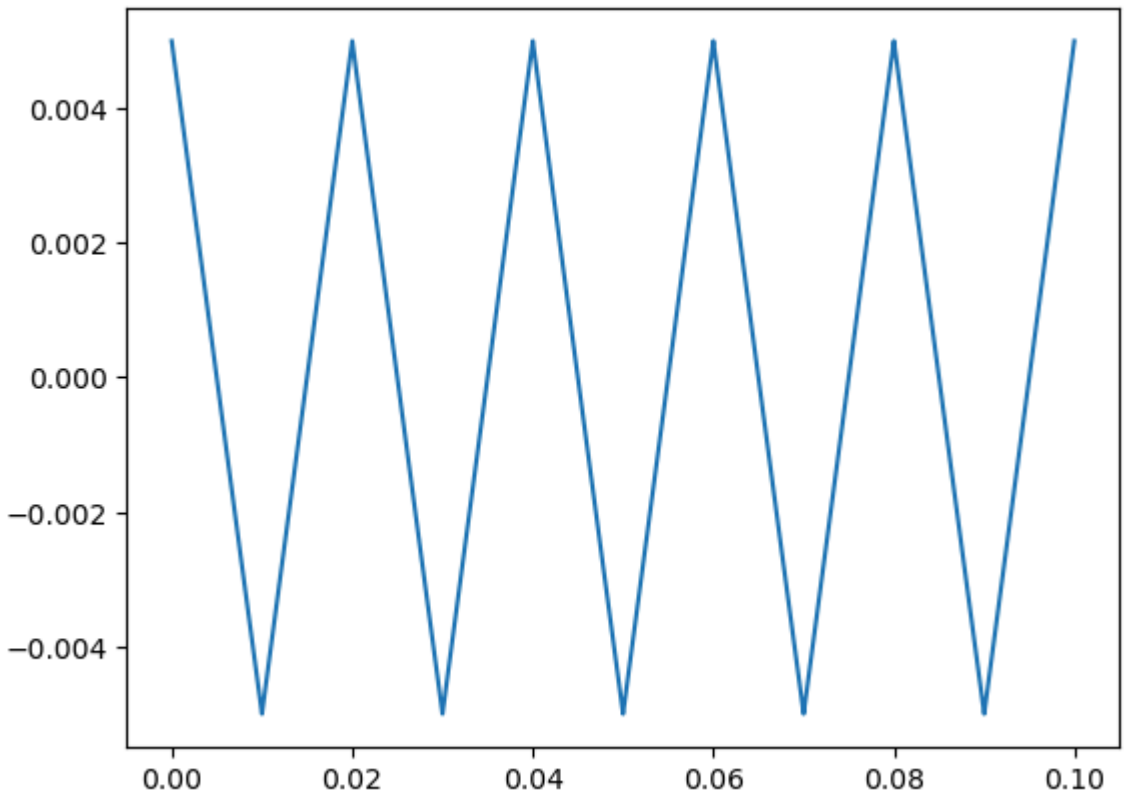


Как и ожидалось при взятии накопленной сумма от прямоугольного сигнала, получаем треугольный. Это было рассмотрено в предыдущем упражнении

In [ ]:
```
sq_spectrum_int = in_wave.make_spectrum().integrate()
sq_spectrum_int.plot()
```



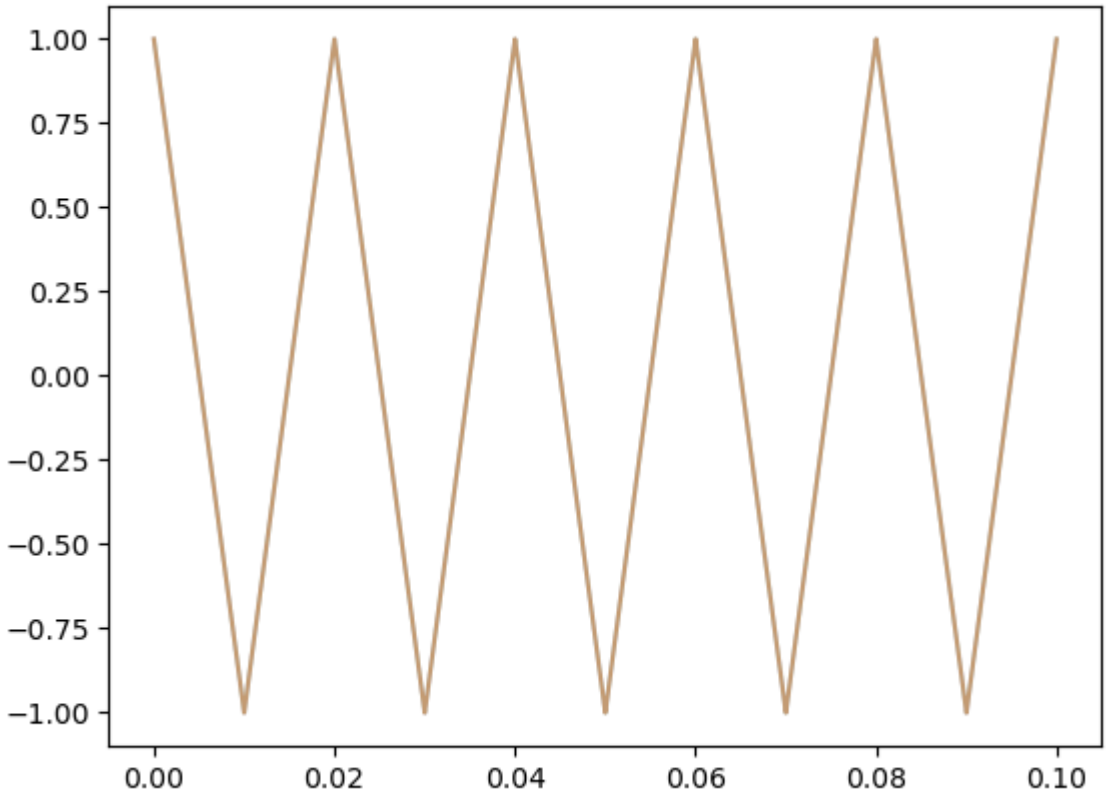
In [ ]:
```
sq_spectrum_int.hs[0] = 0
out_wave2 = sq_spectrum_int.make_wave()
out_wave2.plot()
```



Интегрирование спектра сигнала также дает нам треугольный сигнал, но амплитуды очень сильно отличаются

Для сравнения накапливающей суммы и интегрирования нормализуем оба сигнала

In [ ]:
```
out_wave.unbias()
out_wave.normalize()
out_wave2.normalize()
out_wave.plot(alpha=0.5)
out_wave2.plot(alpha=0.5)
```



Полученные сигналы идентичны с хорошей точностью

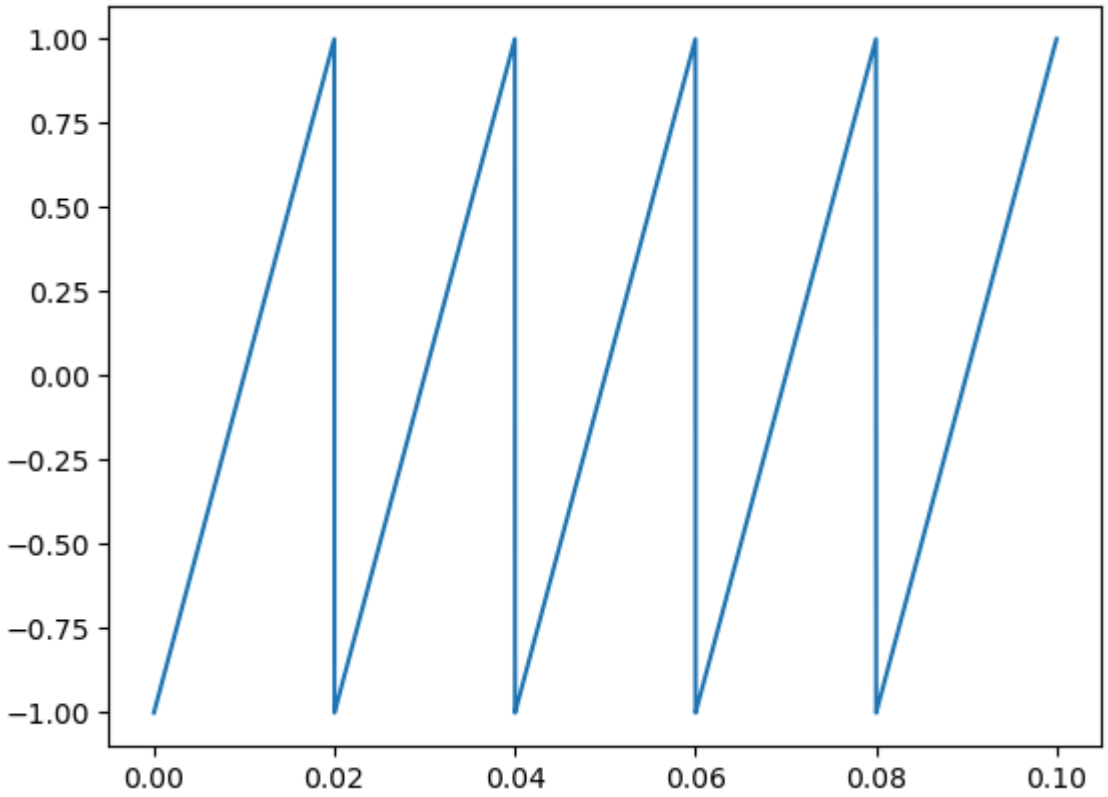
In [ ]:
```
out_wave.max_diff(out_wave2)
```

Out[ ]:
```
2.7755575615628914e-17
```

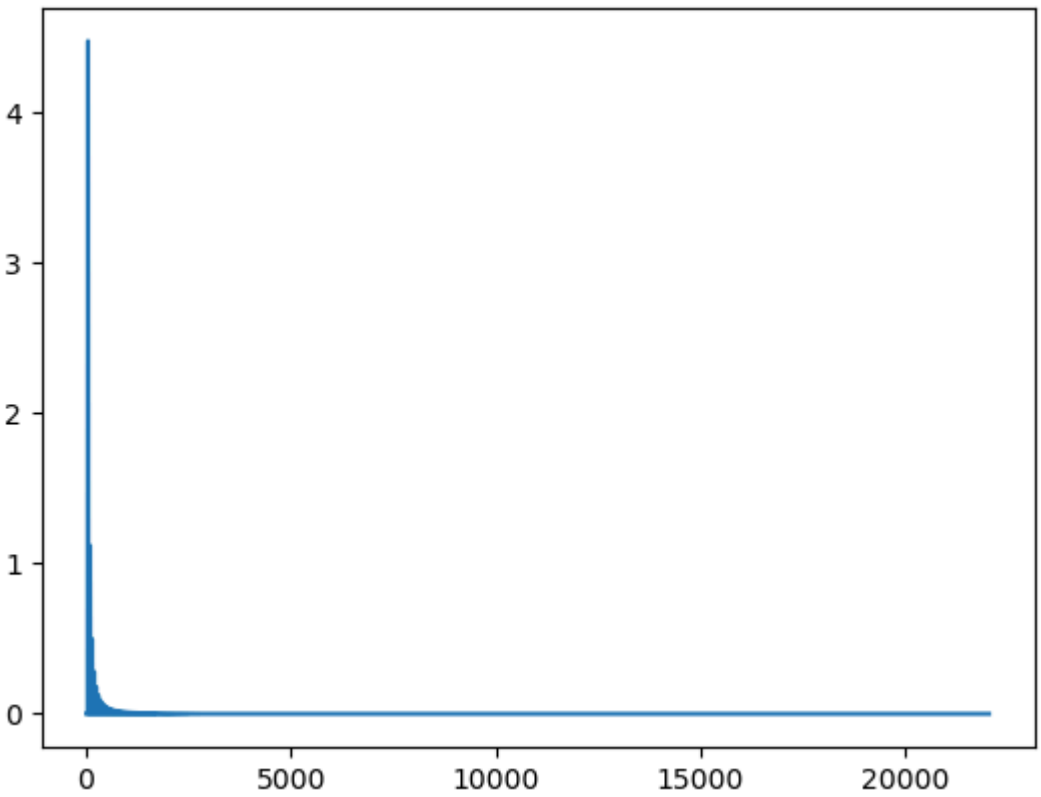
Упражнение 9.4

> Создайте пилообразный сигнал, вычислите его спектр, а затем дважды примените `integrate`. Напечатайте результирующий сигнал и его спектр. Какова математическая форма сигнала? Почему он напоминает синусоиду?

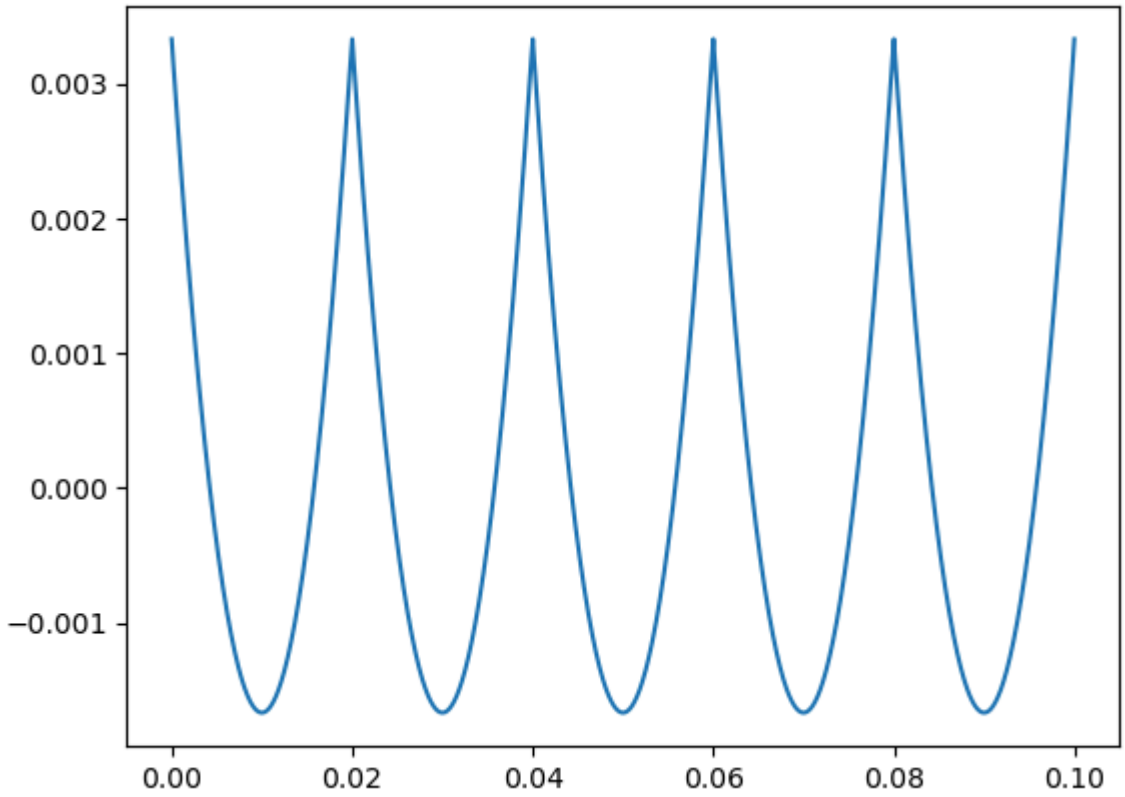
In [ ]:
```
in_wave = SawtoothSignal(freq=50).make_wave(duration=0.1, framerate=44100)
in_wave.plot()
```



In [ ]:
```
in_spectrum = in_wave.make_spectrum().integrate()
in_spectrum.plot()
```



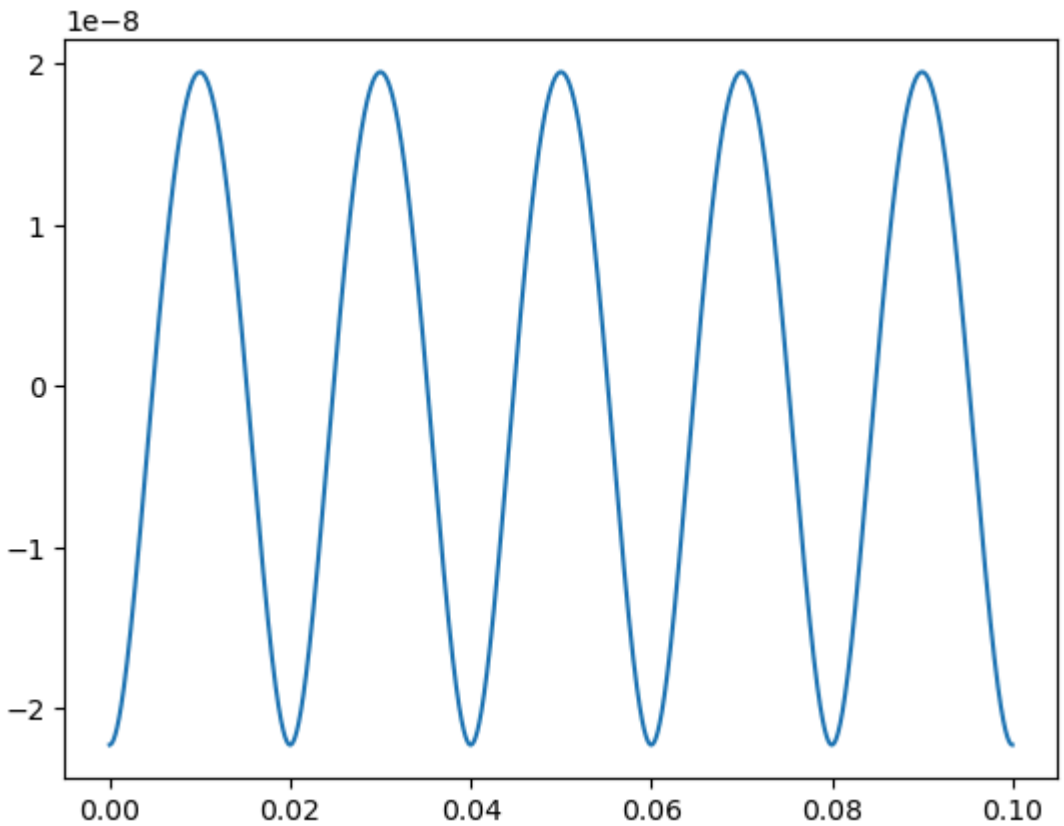
In [ ]:
```
in_spectrum.hs[0] = 0
out_wave = in_spectrum.make_wave()
out_wave.unbias()
out_wave.plot()
```



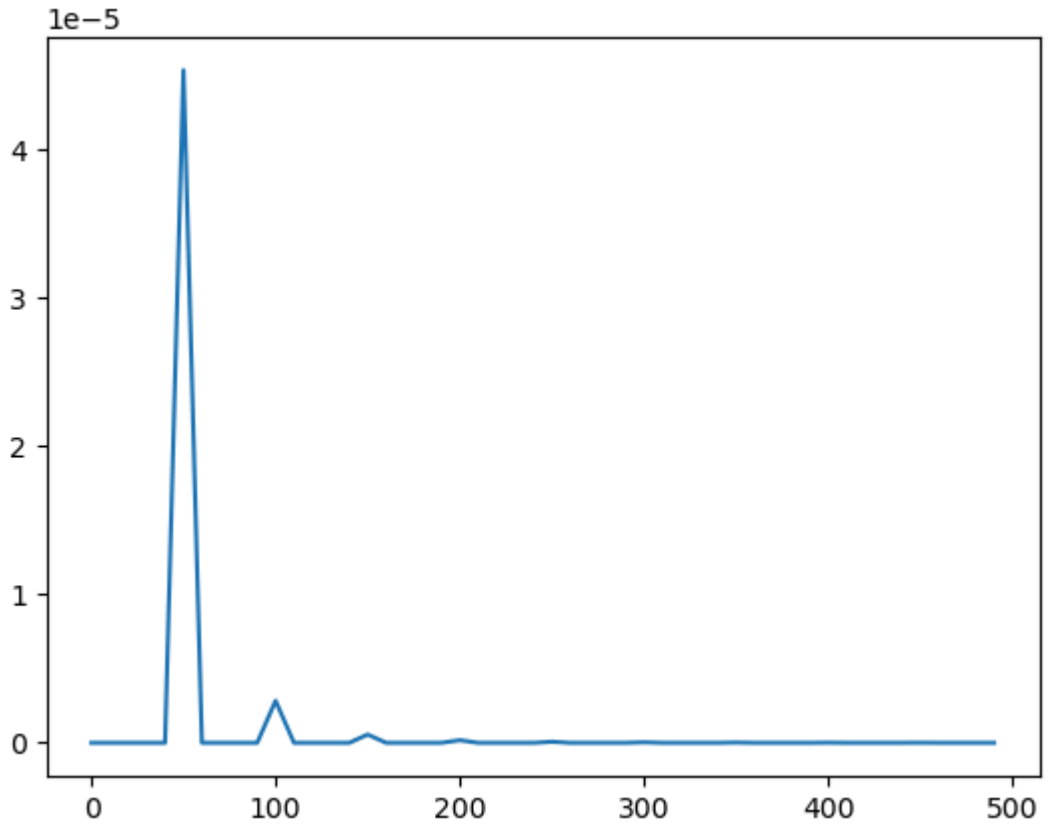
Первый интеграл от пилообразного сигнала дает сигнал, который состоит из парабол. Появление параболы монжо объяснить тем, что пилообразный сигнал - повторяющиеся прямые, которые можно описать уравнением $y = x$, если взять интеграл, то получим $y = \frac{x^2}{2}$. Квадрат дает параболу

При повторном взятии интеграла получим кубическую кривую, ее появление можно также объяснить взятием интегала.

In [ ]:
```
in_spectrum = in_spectrum.integrate()
in_spectrum.hs[0] = 0
out_wave = in_spectrum.make_wave()
out_wave.plot()
```



In [ ]:
```
out_wave.make_spectrum().plot(high=500)
```

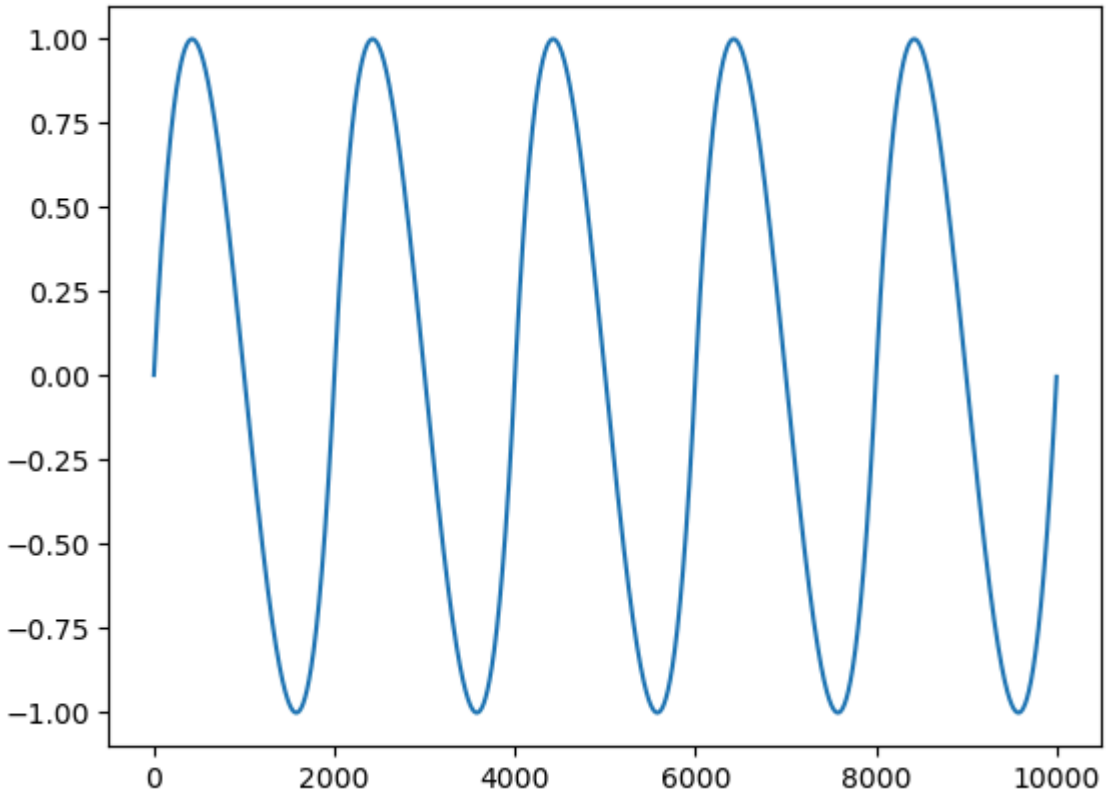


Полученный сигнал, стал похож на синусоиду, так как интегрирование выступает в роли фильтра высоких частот, что можно увидеть на графике спектра. Мы отфильтровали пости все кроме фундаментальной частоты - 50 Гц

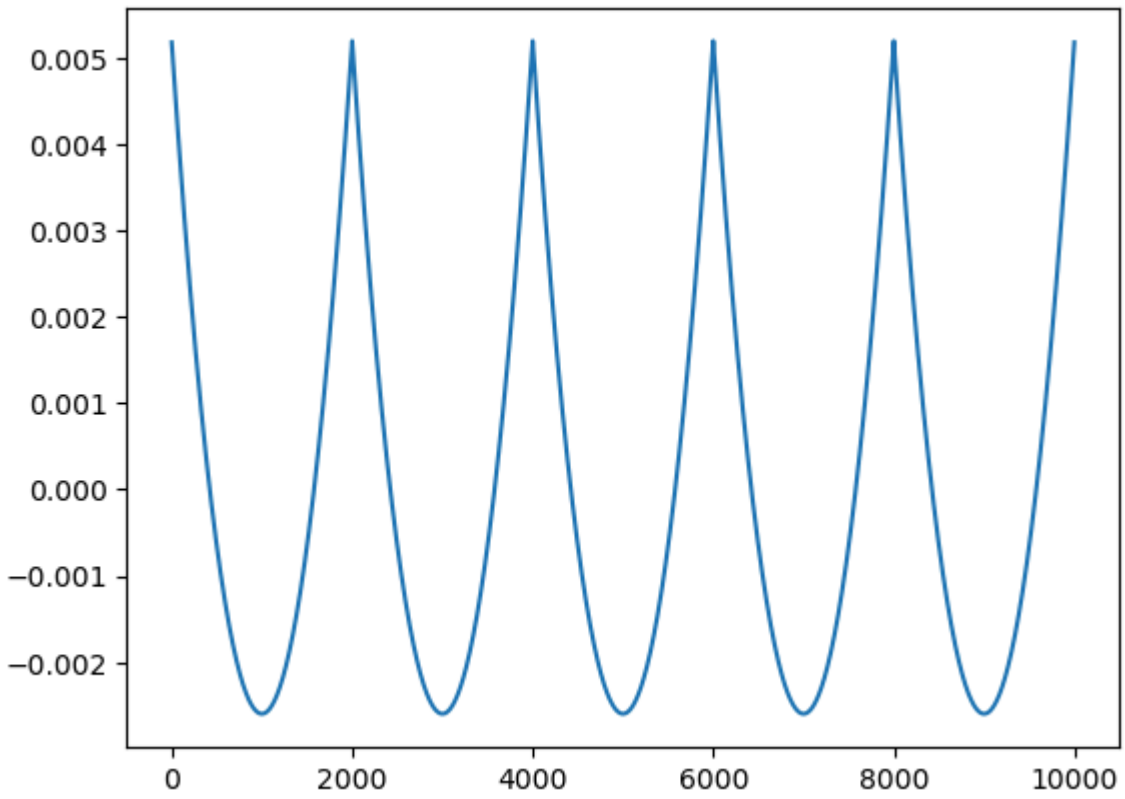
Упражнение 9.5

> Создайте `CubicSignal`, определенный в `thinkdsp`. Вычислите вторую разность, дважды применив `diff`. Как выглядит результат? Вычислите вторую вторую производную, дважды применив `differentiate` к спектру. Похожи ли результаты? Распечатайте фильтры, соответствующте второй разности и второй производной, и сравните их.

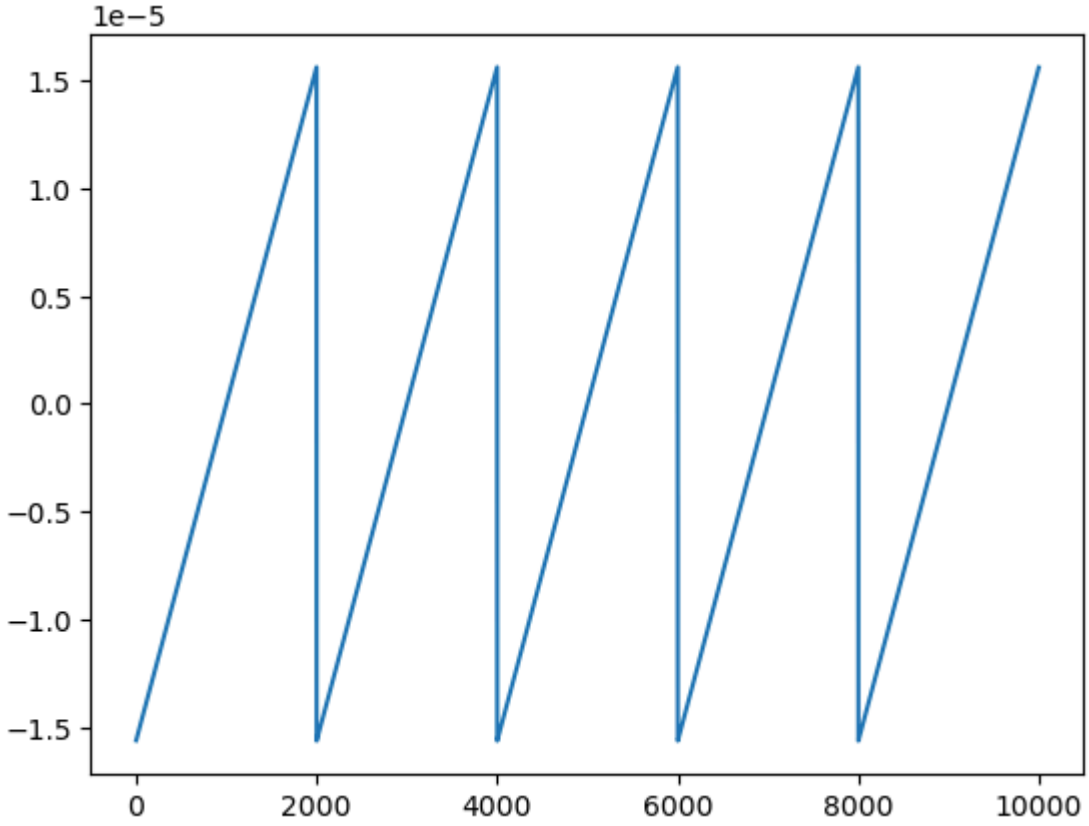
In [ ]:
```
in_wave = CubicSignal(freq=0.0005).make_wave(duration=10000, framerate=1)
in_wave.plot()
```



In [ ]:
```
out_wave = in_wave.diff()
out_wave.plot()
```



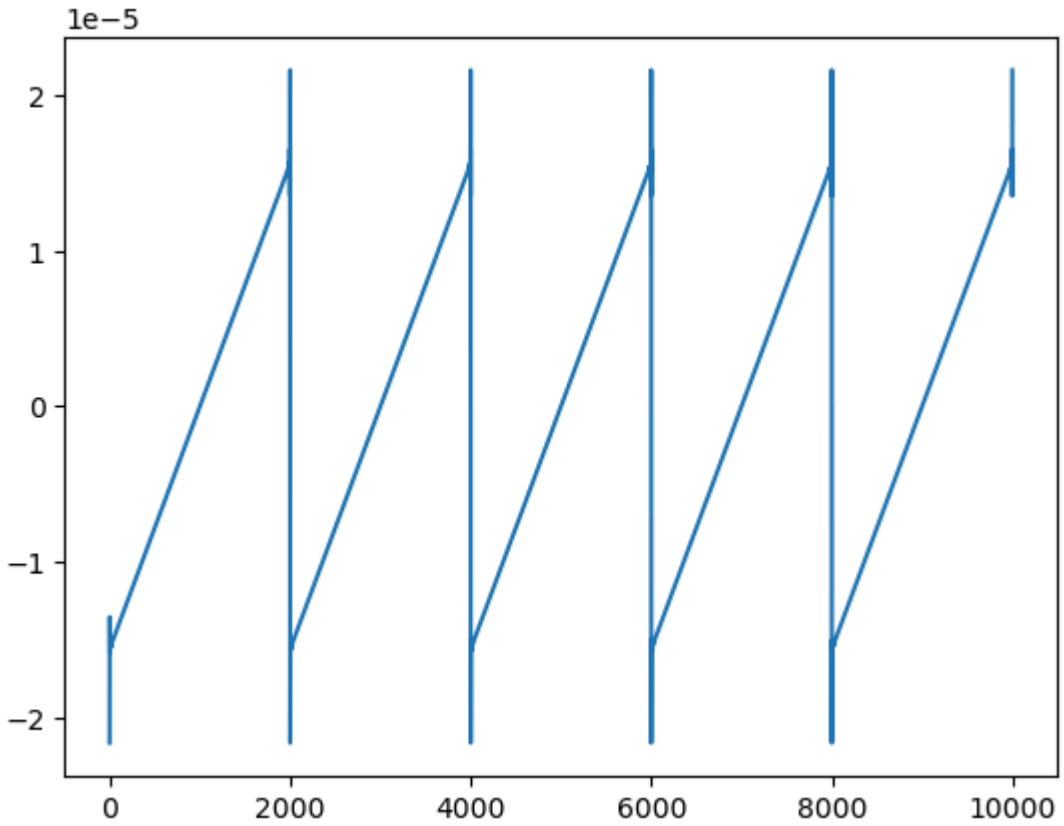
In [ ]:
```
out_wave = out_wave.diff()
out_wave.plot()
```



Полученные результаты подтверждают, что дифференциирование обратно интегрированию. Принцип описан в предыдущем упражнении

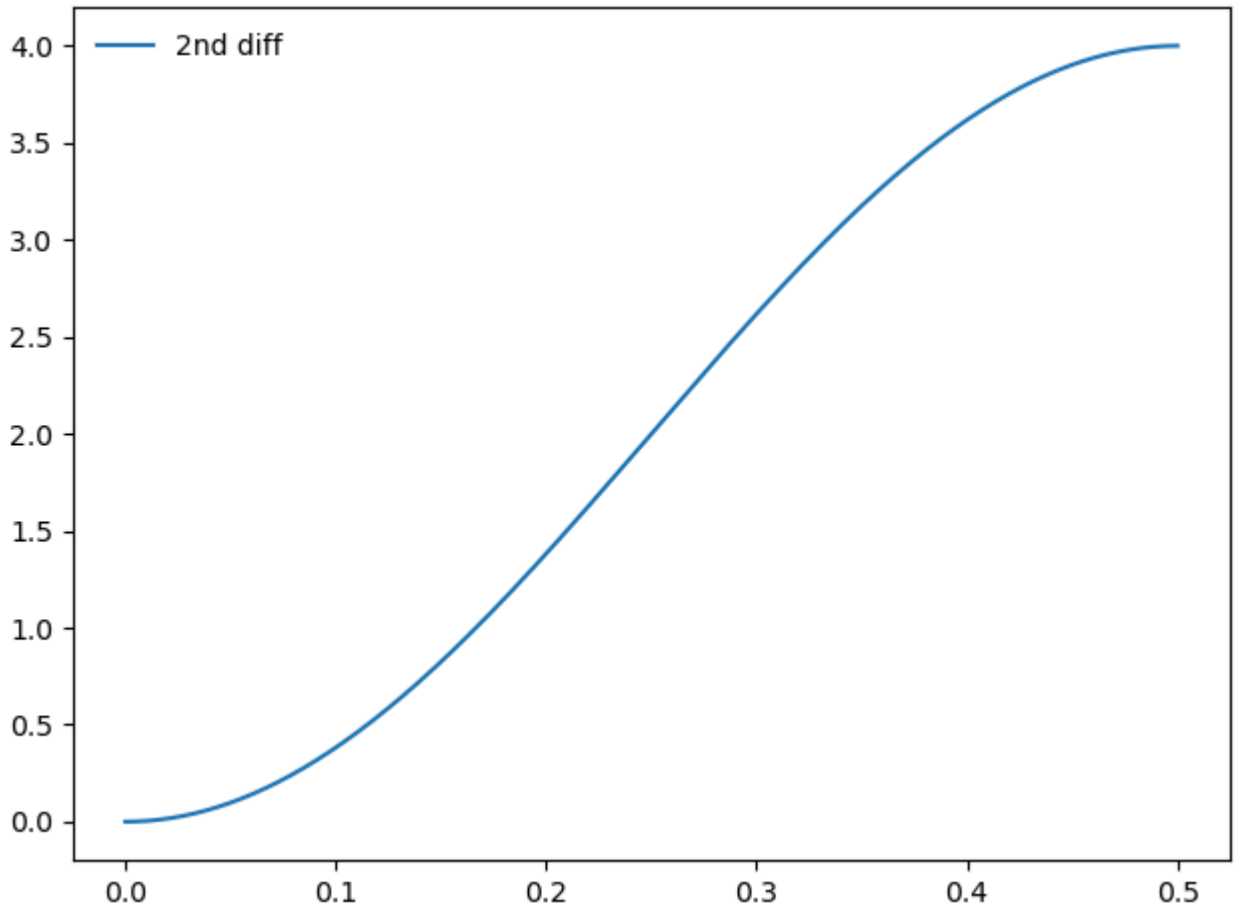
При двойном дифференциировании спектра получаем звон у волны в местах, где производная параболы не определена

In [ ]:
```
spectrum = in_wave.make_spectrum().differentiate().differentiate()
out_wave2 = spectrum.make_wave()
out_wave2.plot()
```



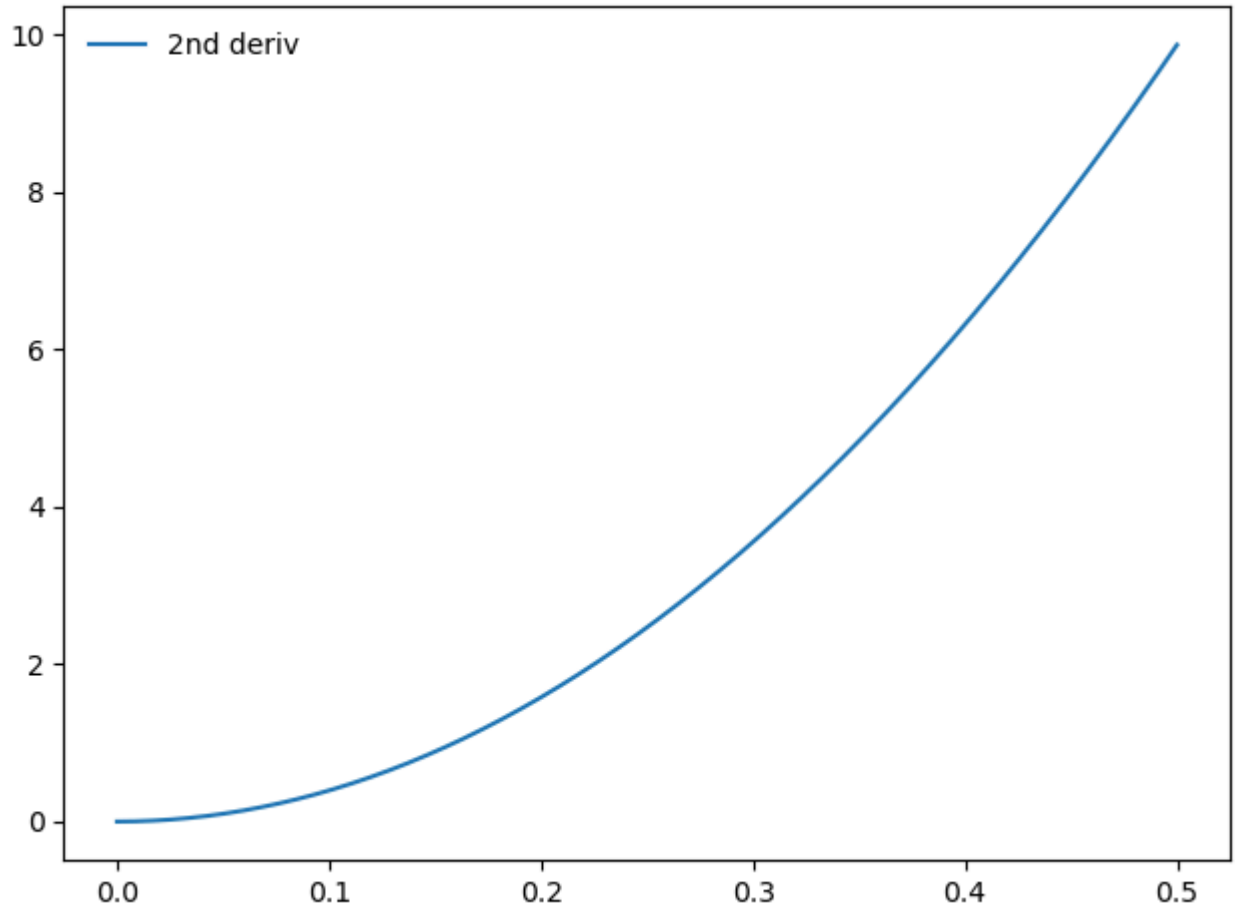
Для нахождения фильтра, который соответствует второй конечной разности, можем вычислить фильтр окна, которое соответствует второй конечной разности

In [ ]:
```
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()
```

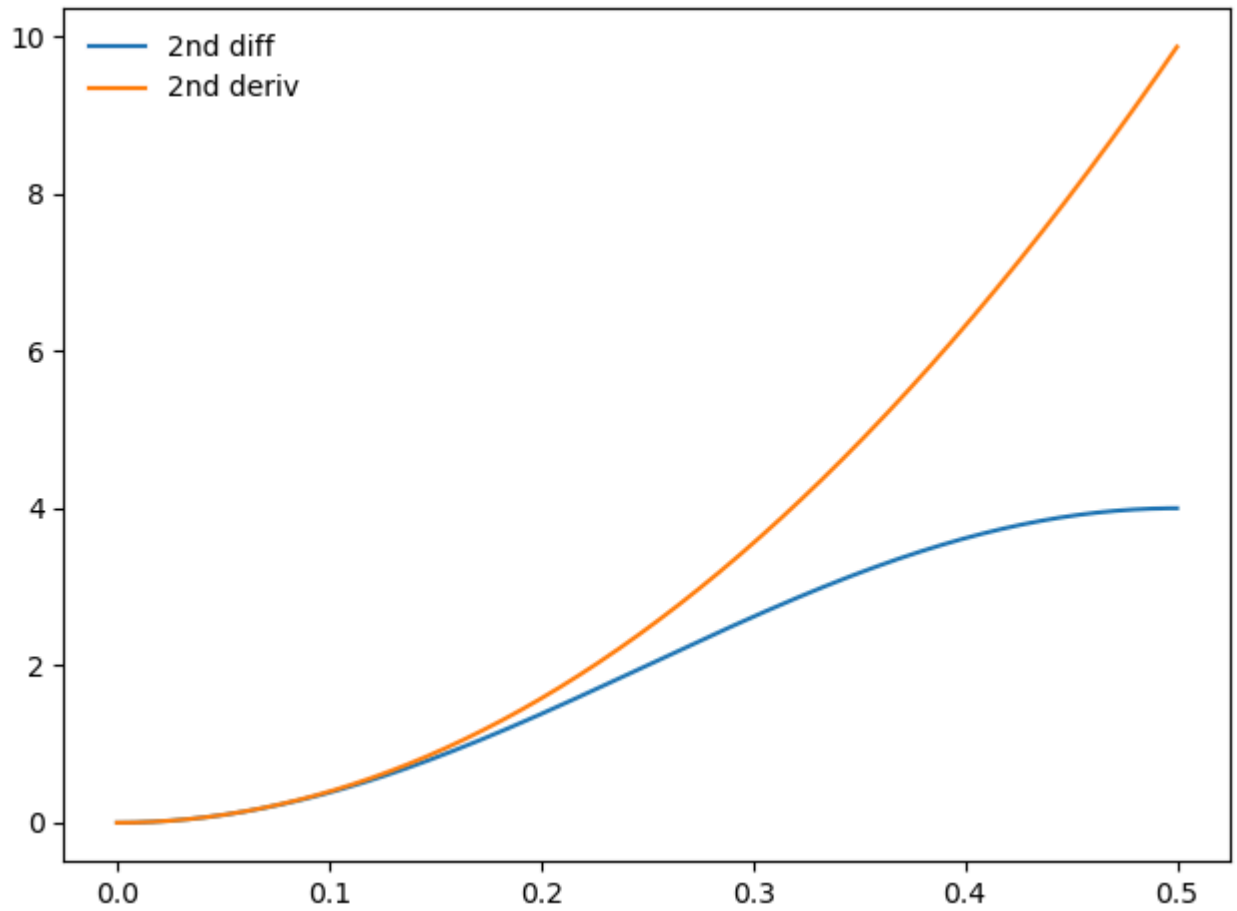


Для нахождения фильтра второй производной, можно рассчитать фильтр первой производной и возвести его в квадрат

In [ ]:
```
deriv_filter = in_wave.make_spectrum()
deriv_filter.hs = (np.pi * 2 * 1j * deriv_filter.fs)**2
deriv_filter.plot(label='2nd deriv')

decorate()
```



In [ ]:
```
diff_filter.plot(label='2nd diff')
deriv_filter.plot(label='2nd deriv')
decorate()
```



Оба полученных фильтра - фильтры низких частот. Фильтр на основе производной сильнее всего усиливает высокие частоты. Вторая конечная разность - аппроксимация второй производной,

которая совпадает со второй производной в области низких частот, но значительно отклоняется при более высоких частотах

In [ ]: