Лабораторная работа № 9.
Дифференцирование и интегрирование.

3530901/80201, Шелаев Н. Р.

31 мая 2021 г.

Оглавление

Список иллюстраций

Листинги

Глава 1

Конечные разности

Проверим, что вычисление разницы между соседними элементами во временной области соответствуют фильтру ВЧ.

```
1       import pandas as pd
2       from thinkdsp import Wave
3
4       df = pd.read_csv('FB_2.csv', header=0, parse_dates=[0])
5       ys = df['Close']
6       if len(ys) % 2: ys = ys[:-1]
7       close = Wave(ys, framerate = 1)
8       close.plot()
9       close_spectrum = close.make_spectrum()
10      close_spectrum.plot()
11
```

Листинг 1.1: И снова акции Facebook

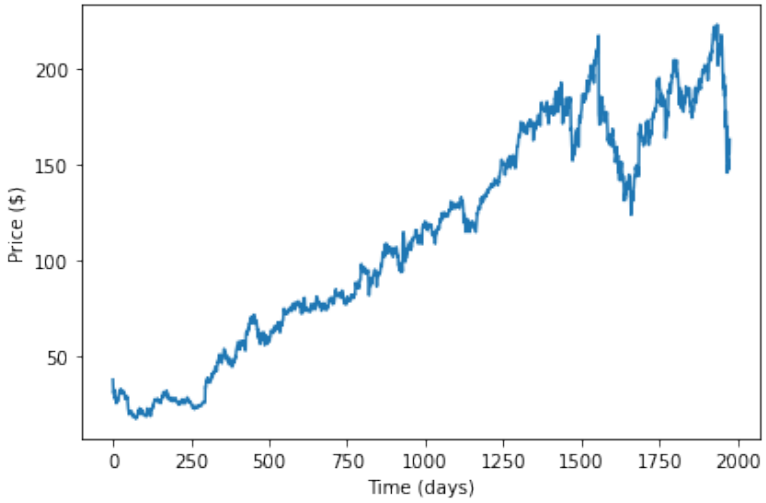

Рис. 1.1: Сам график

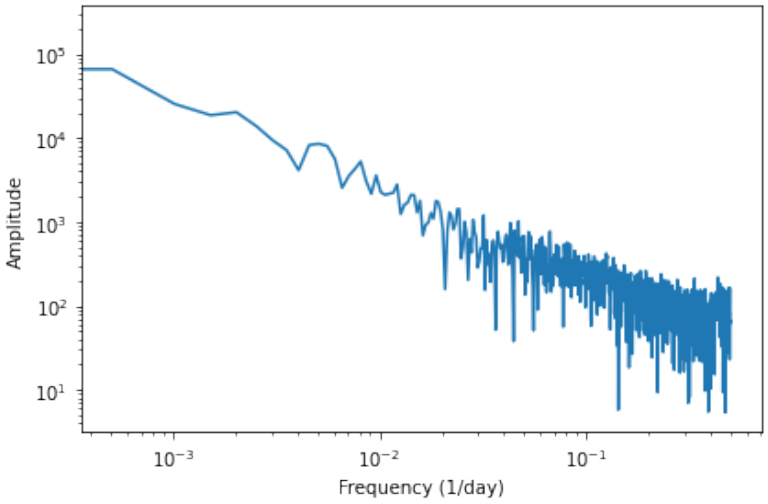

Рис. 1.2: График в логарифмическом масштабе

Наклон линии мощности составляет примерно -1.8, что очень близко к «Красному» шуму (-2).

```
1       change = Wave(np.diff(close.ys), framerate = 1)
2       change.plot()
3       change_spectrum = change.make_spectrum()
4       change_spectrum.plot()
5       decorate(xlabel = 'Frequency (1/day)', ylabel = '
    Amplitude')
```

```
6
7         change_spectrum.plot()
8         decorate(xlabel='Frequency (1/day)', ylabel='Amplitude'
      , xscale='log', yscale='log')
9
```

Листинг 1.2: Считаем разницу между последовательными элементами

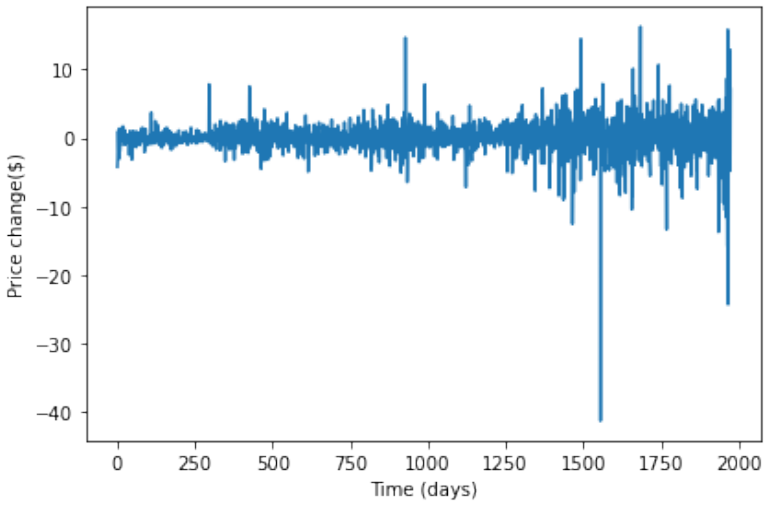

Рис. 1.3: Разница между ежедневными значениями акций

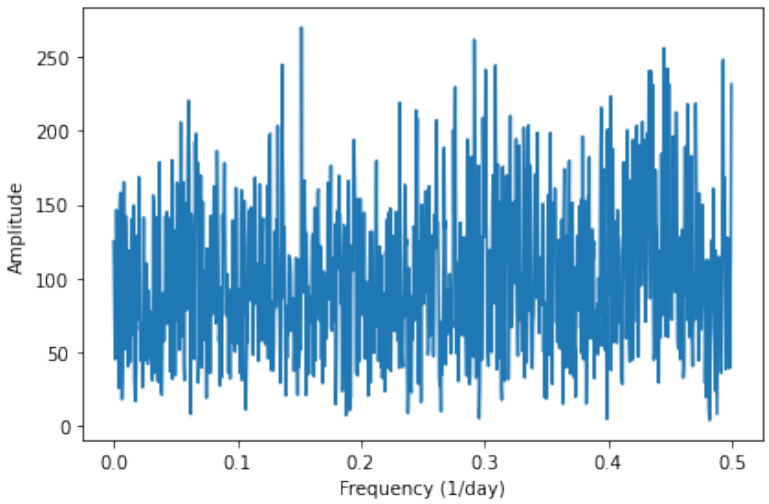

Рис. 1.4: Спектр ежедневных изменений

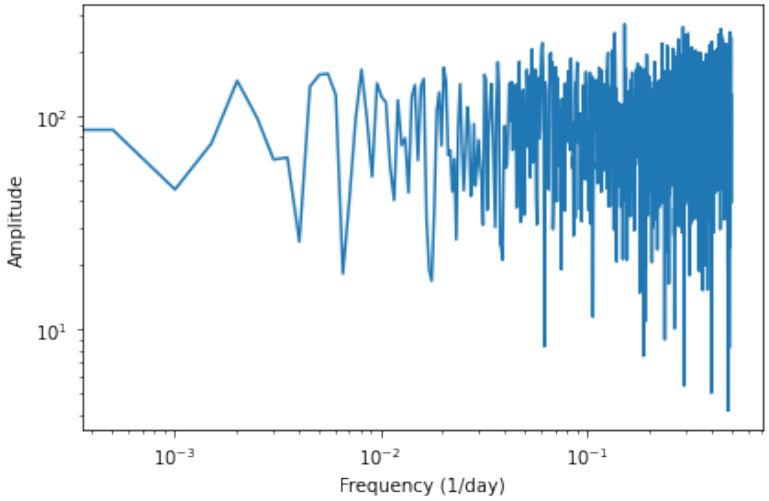

Рис. 1.5: Спектр в логарифмическом масштабе

Наклон прямой близок к 0, поэтому можно сделать вывод, что это «Белый» шум.

```
    from thinkdsp import zero_pad

    def make_filter(window, wave):
        padded = zero_pad(window, len(wave))
        window_wave = Wave(padded, framerate = wave.
framerate)
        window_spectrum = window_wave.make_spectrum()
        return window_spectrum

    diff_window = np.array([1.0, -1.0])
    diff_filter = make_filter(diff_window, close)
    diff_filter.plot()
    plt.plot(diff_filter.angles)

```

Листинг 1.3: Умножение на фильтр - свертка с окном

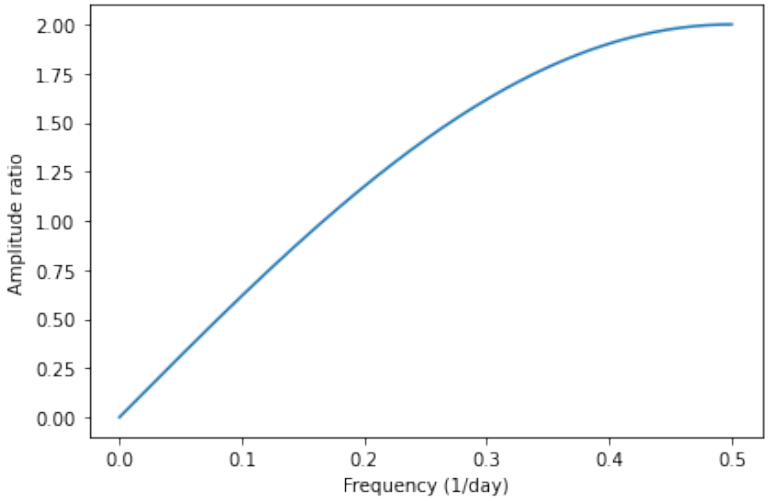

Рис. 1.6: Фильтр для окна

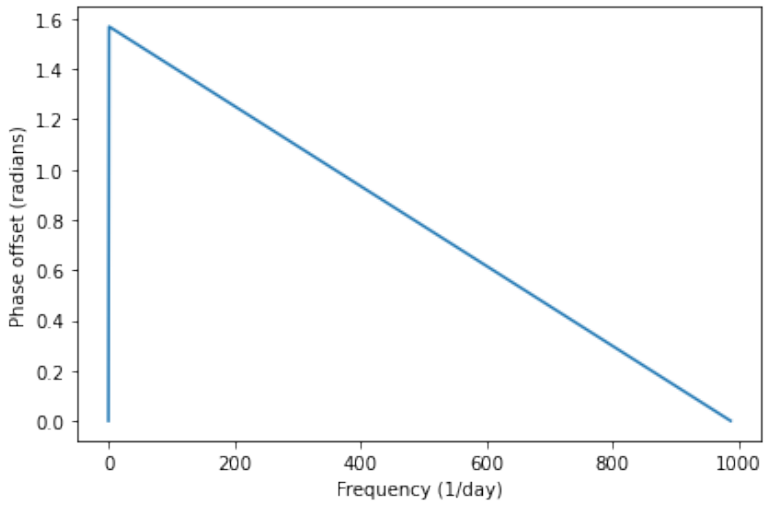

Рис. 1.7: Углы фильтра

```
change_spectrum2 = close_spectrum * diff_filter
change_spectrum2.plot()
change2 = change_spectrum2.make_wave()
change2.ys = change2.ys[1:]
change2.ts = change2.ts[1:]

```

Листинг 1.4: Умножаем спектр цен на полученный фильтр

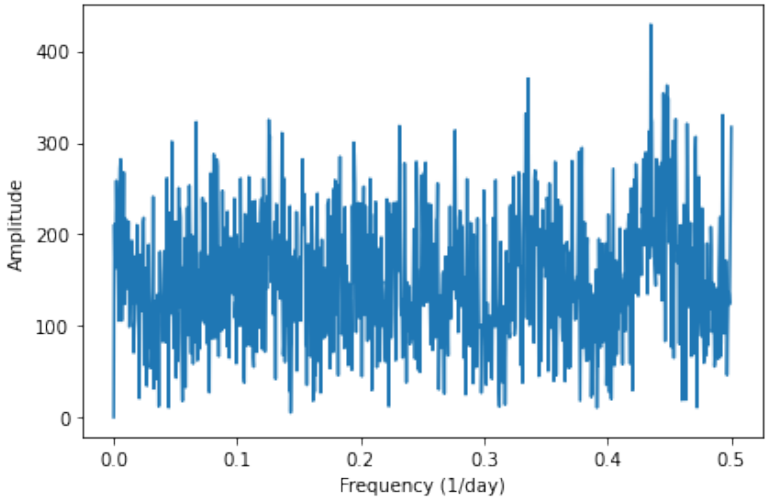

Рис. 1.8: Изменённый спектр цен

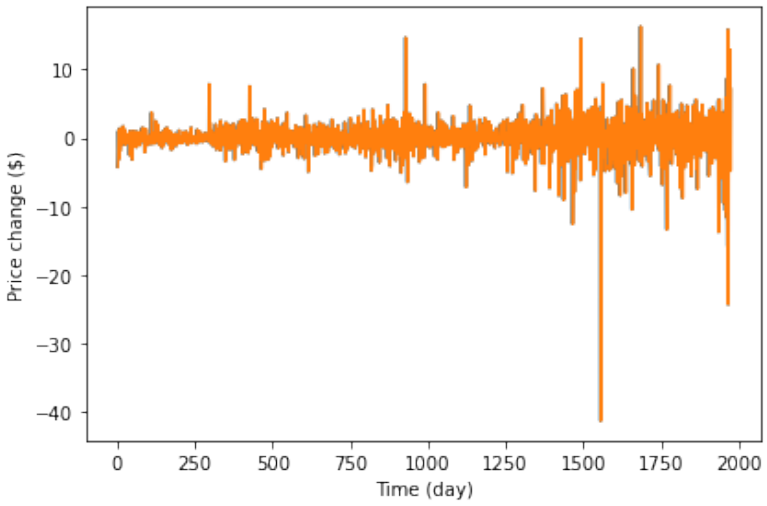

Рис. 1.9: Сравнение двух ранее полученных методов

Результаты этих двух методов совпадают.

Глава 2

Дифференцирование

Дифференцирование во временной области соответствует простому фильтру в частотной области.

```
PI2 = np.pi * 2
deriv_filter = close.make_spectrum()
deriv_filter.hs = PI2 * 1j * deriv_filter.fs
deriv_filter.plot()
deriv_spectrum = close.make_spectrum().differentiate()
deriv_spectrum.plot()

```

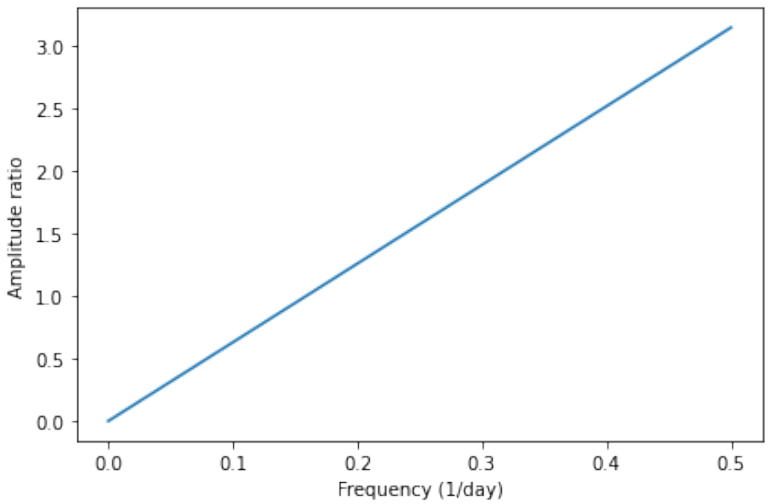

Рис. 2.1: Фильтр для дифференцирования

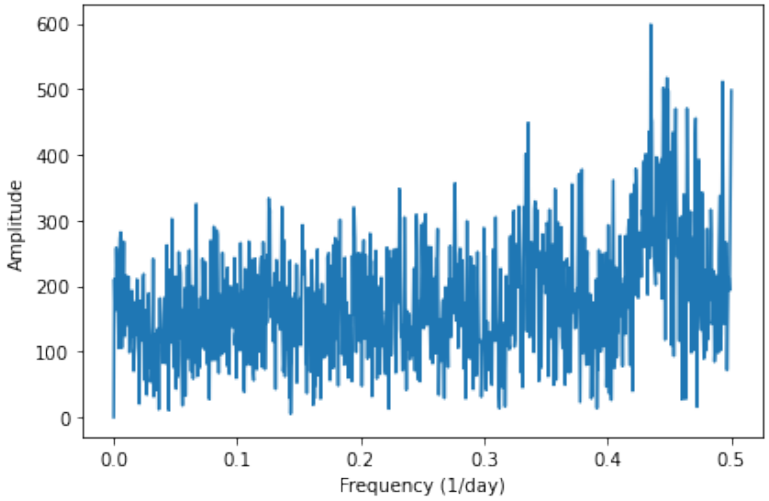

Рис. 2.2: Результат применения фильтра

```
deriv = deriv_spectrum.make_wave()
deriv = deriv_spectrum.make_wave()
change.plot(alpha = 0.5)
deriv.plot(alpha = 0.5)

```

Листинг 2.1: Сравнение двух методов

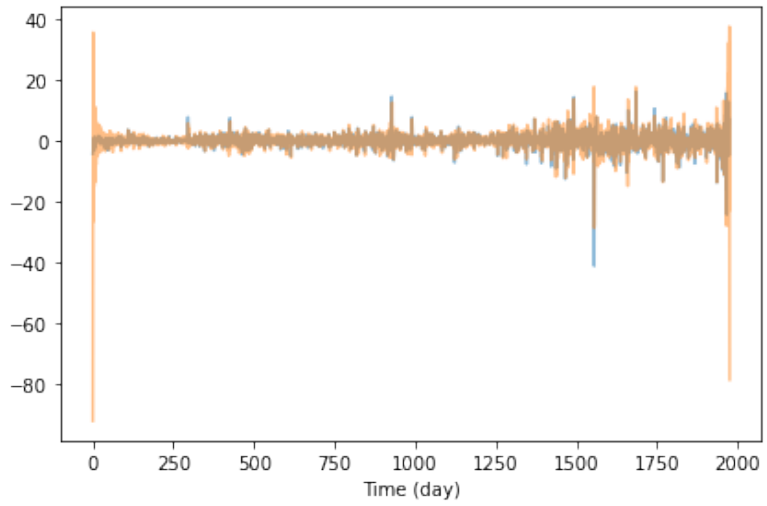

Рис. 2.3: Результаты этих методов похожи

Полученные результаты совпадают с `np.diff`, но с некоторой погрешностью: окно разности является лишь грубым приближением производной

(особенно, на высоких частотах), и спектральная производная основана на предположении, что сигнал является периодическим (а это не так), поэтому в начале и в конце окна поведение отличается.

```
low, high = 0, 50
plt.plot(change.ys[low:high], label = 'diff')
plt.plot(deriv.ys[low:high], label = 'deriv')
deriv_filter.plot()
diff_filter.plot()

```

Листинг 2.2: Увеличим масштаб

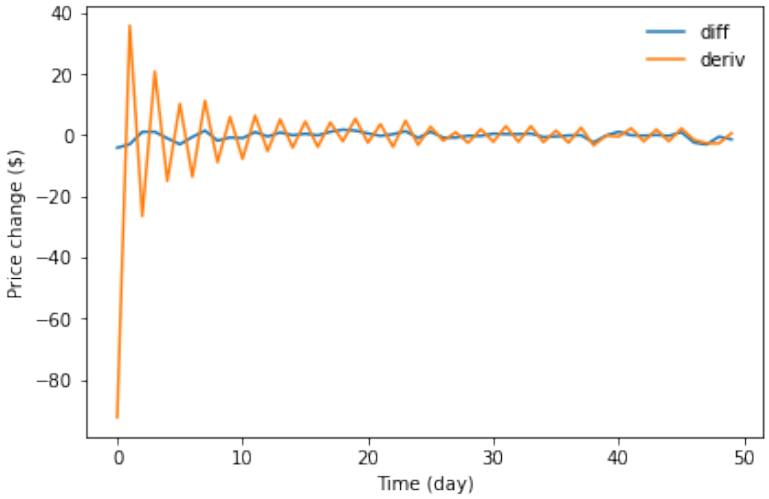

Рис. 2.4: Теперь лучше видны различия

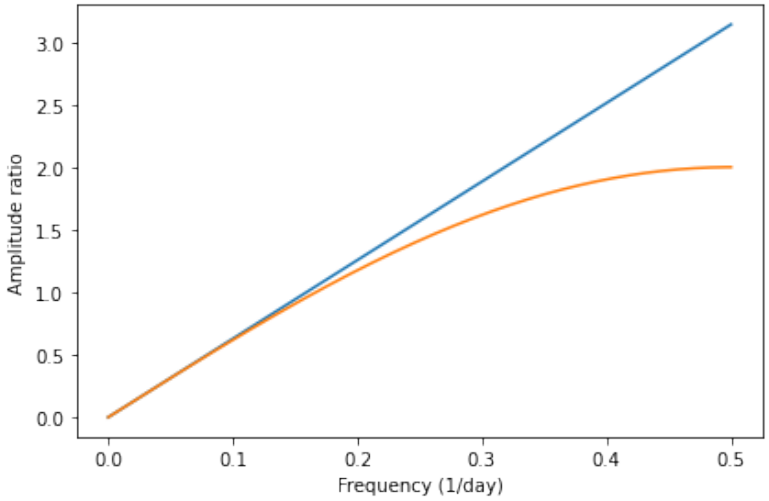

Рис. 2.5: Разница между фильтром для дифференцирования и разностным фильтром

Разностный фильтр не так сильно усиливает высокие частоты.

Глава 3

Интегрирование

Поскольку интегрирование обратно дифференцированию, оно также соответствует простому фильтру.

```
integ_filter = deriv_filter.copy()
integ_filter.hs[1:] = 1/(PI2*1j*integ_filter.fs[1:])
integ_filter.hs[0] = np.inf
integ_filter.plot()

```

Листинг 3.1: Фильтр для интегрирования

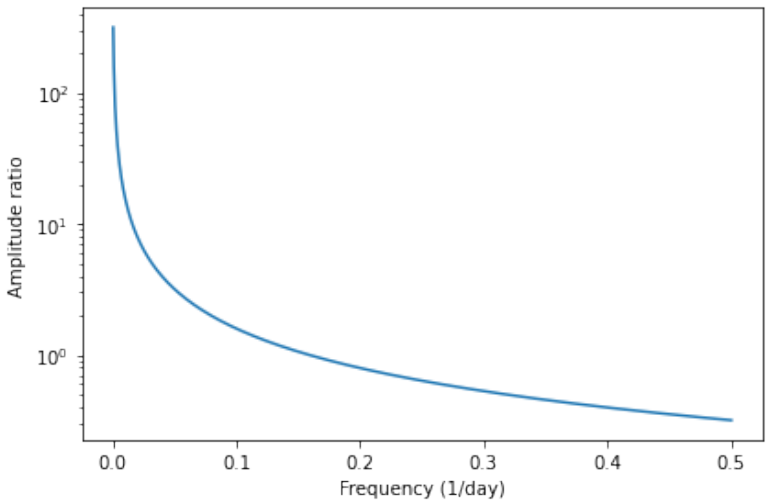

Рис. 3.1: Вид полученного фильтра

```
integ_spectrum = deriv_spectrum.copy().integrate()
integ_spectrum.plot()
close.plot(label = 'closing prices', alpha = 0.7)
```

```
4         integ_spectrum.hs[0] = 0
5         integ_wave = integ_spectrum.make_wave()
6         integ_wave.plot(label = 'integrated derivative', alpha
      = 0.7)
7
```

Листинг 3.2: Фильтр для интегрирования

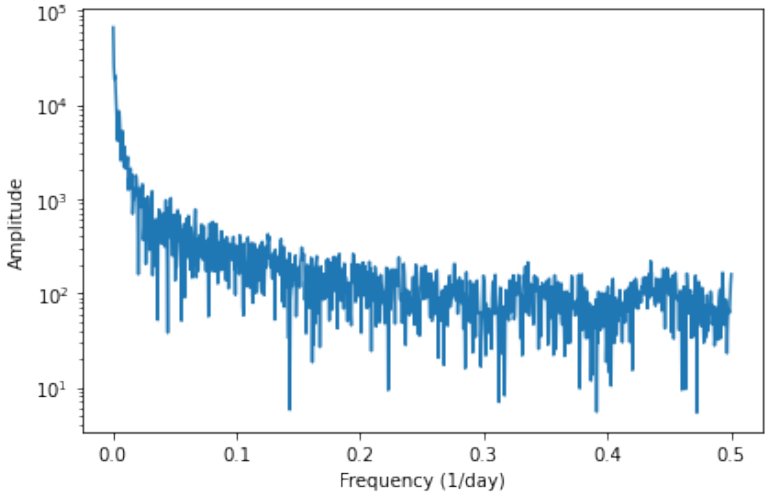

Рис. 3.2: Применим фильтр для спектра производной

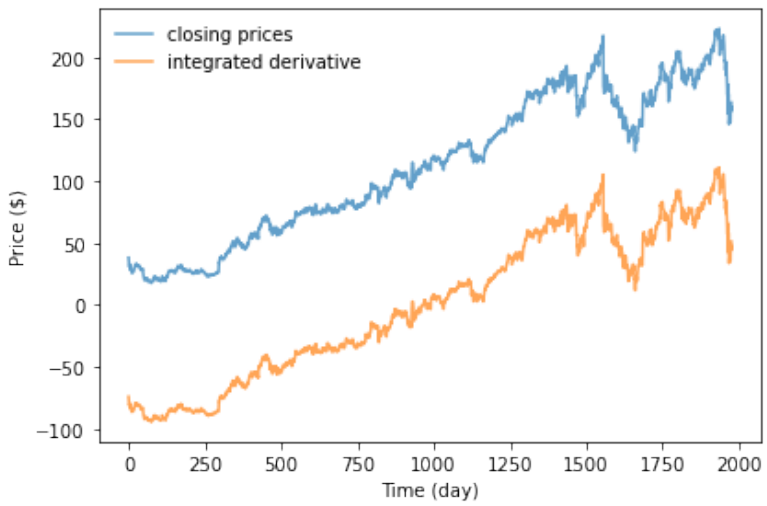

Рис. 3.3: Сравнение исходного сигнала и сигнала после дифференцирования и интегрирования

Результат применения фильтров совпал с исходным графиком, но сдвинут таким образом, что среднее значение равно 0. Производная выбирает первый элемент спектра, который является смещением, а интегрирование не может его восстановить. Поэтому, полученный результат имеет неопределенную константу интегрирования. Если сдвинуть полученный результат к исходному, то разница между ними близка к 0.

Глава 4

Нарастающая сумма

Оператор `diff` аппроксимирует дифференцирование, а нарастающая сумма - интегрирование.

```
1       from thinkdsp import SawtoothSignal
2
3       in_wave = SawtoothSignal(freq = 50).make_wave(duration
    =0.1, framerate=44100)
4       in_wave.unbias()
5       in_wave.plot()
6       in_spectrum = in_wave.make_spectrum()
7       in_spectrum.plot()
8
```

Листинг 4.1: Наш любимый пилообразный сигнал

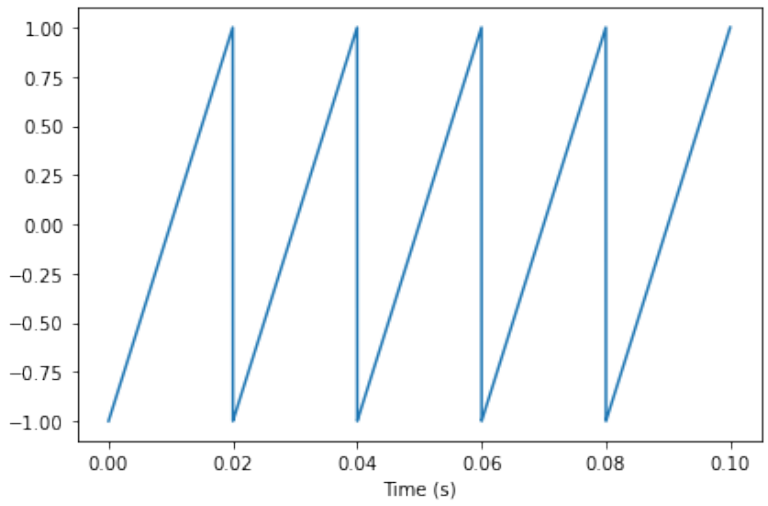

Рис. 4.1: Вид пилообразного сигнала

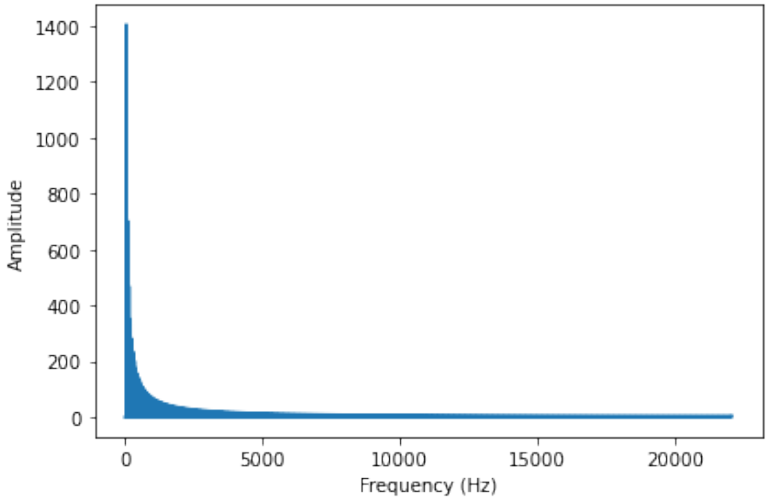

Рис. 4.2: И его спектр

```
1        out_wave = in_wave.cumsum()
2        out_wave.unbias()
3        out_wave.plot()
4        out_spectrum = out_wave.make_spectrum()
5        out_spectrum.plot()
6        ratio_spectrum = out_spectrum.ratio(in_spectrum, thresh
    =1)
7        ratio_spectrum.plot(marker = '.', ms = 4, ls = '')
8
```

Листинг 4.2: Посчитаем сумму

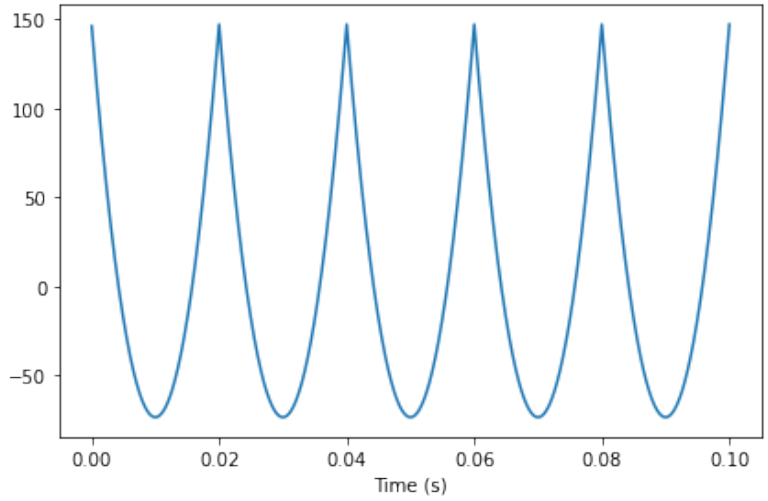

Рис. 4.3: Получили вот этот сигнал

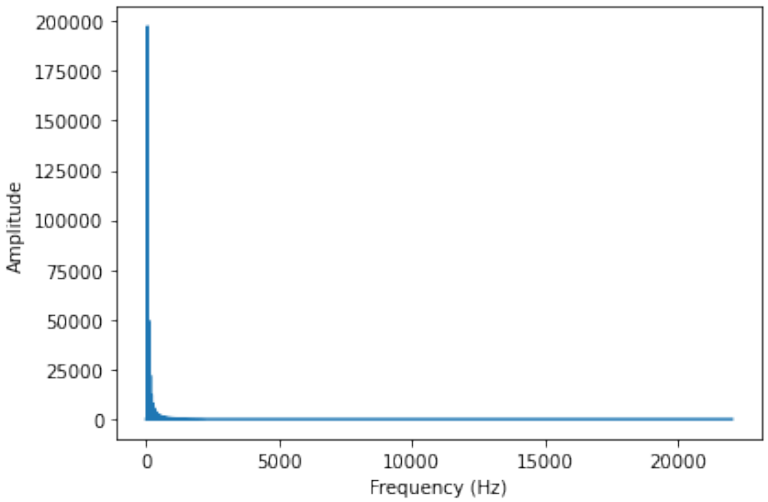

Рис. 4.4: С таким спектром

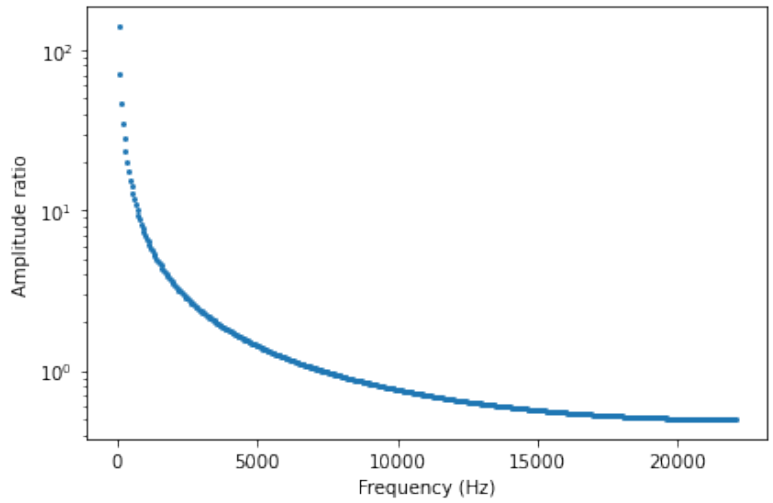

Рис. 4.5: Отношение значения выхода к значению входа

```
    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

```

Листинг 4.3: Получение фильтра для суммирования

```
    integ_filter = cumsum_filter.copy()
    integ_filter.hs[1:] = integ_filter.framerate / (PI2 * 1
j * integ_filter.fs[1:])
    integ_filter.hs[0] = np.inf
    cumsum_filter.plot(label='cumsum filter', alpha=0.7)
    integ_filter.plot(label='integral filter', alpha=0.7)
    cumsum_filter.plot(label = 'cumsum filter')
    ratio_spectrum.plot(label='ratio', marker='.', ms=4, ls
='')

```

Листинг 4.4: Сравнение суммирования и интегрирования

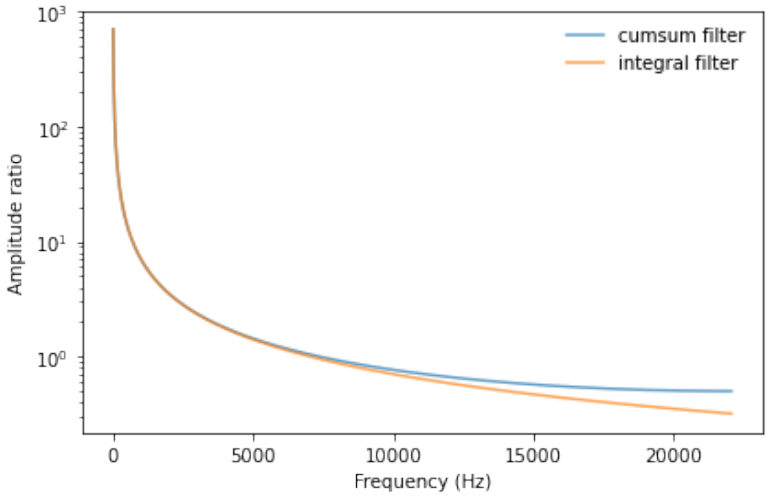

Рис. 4.6: Результат сравнения

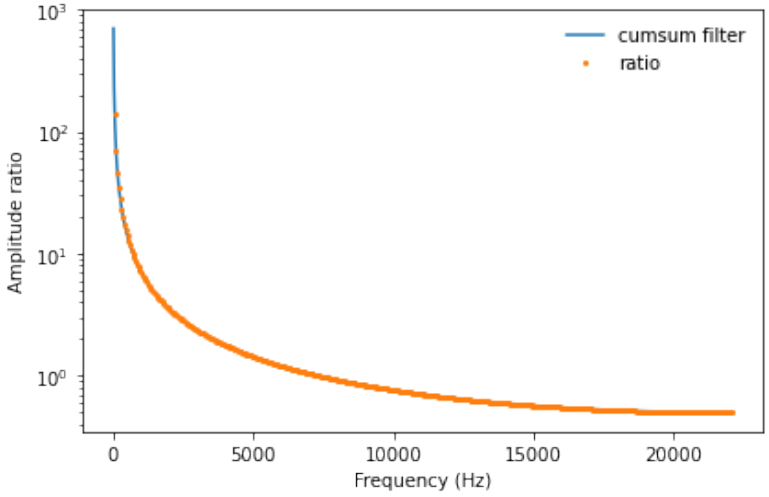

Рис. 4.7: Сравнение вычисленных соотношений с полученным фильтром

Они совпадают, подтверждая, что фильтр для суммирования является обратным разностному фильтру.

```
out_wave.plot(label = 'summed', alpha = 0.7)
cumsum_filter.hs[0] = 0
out_wave2 = (in_spectrum * cumsum_filter).make_wave()
out_wave2.plot(label = 'filtered', alpha = 0.7)

```

Листинг 4.5: Вычисляем выходной сигнал с помощью теоремы о свертке

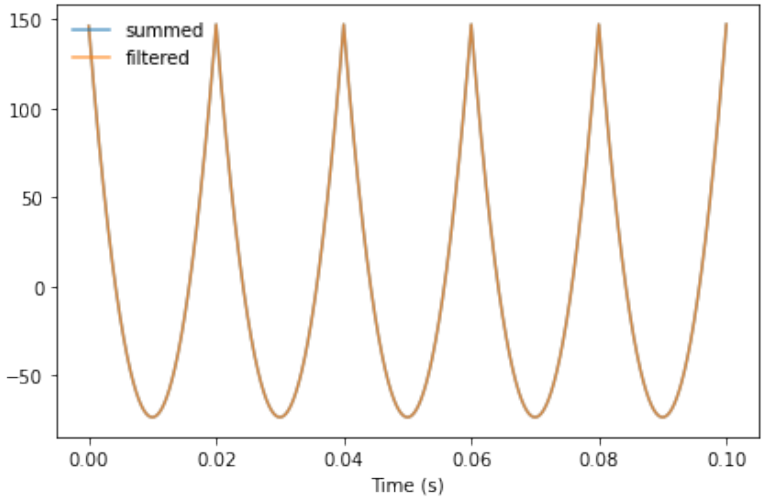

Рис. 4.8: Сравниваем результаты с спектром исходного сигнала

Разница между ними - примерно 0.

Глава 5

Упражнения

5.1 Задание 1

Проверяем работу фильтра для суммирования с апериодическими сигналами.

```
1       in_wave = close
2       in_wave.unbias()
3       in_wave.plot()
4       in_spectrum = in_wave.make_spectrum()
5       in_spectrum.plot()
6       ...
7
```

Листинг 5.1: Возьмем акции Facebook и повторим все действия из предыдущей главы

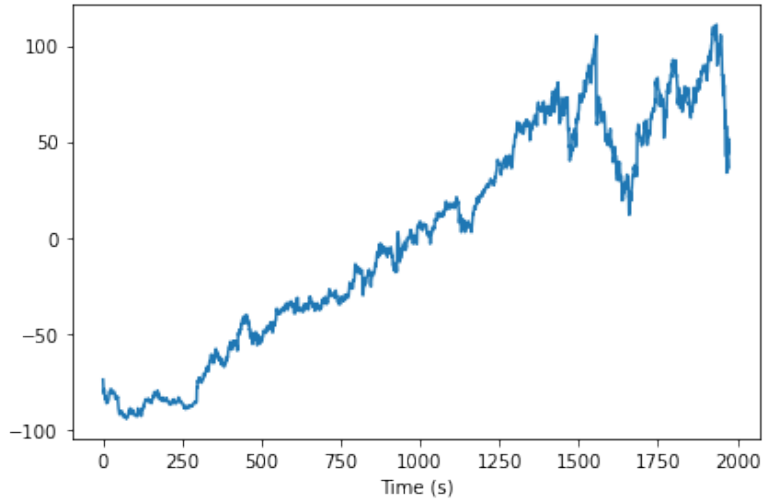

Рис. 5.1: График цен

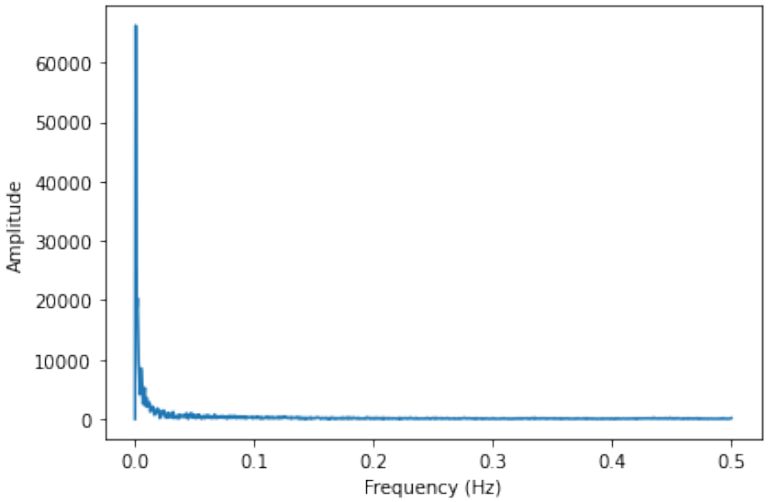

Рис. 5.2: Спектр цен

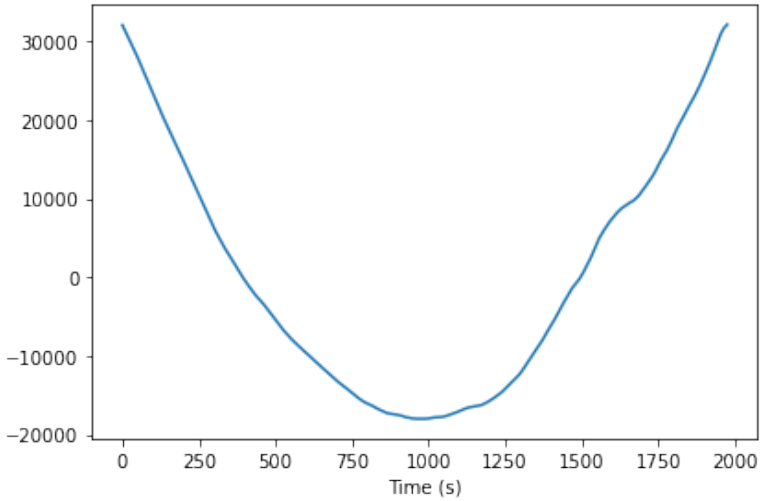

Рис. 5.3: Посчитали нарастающую сумму

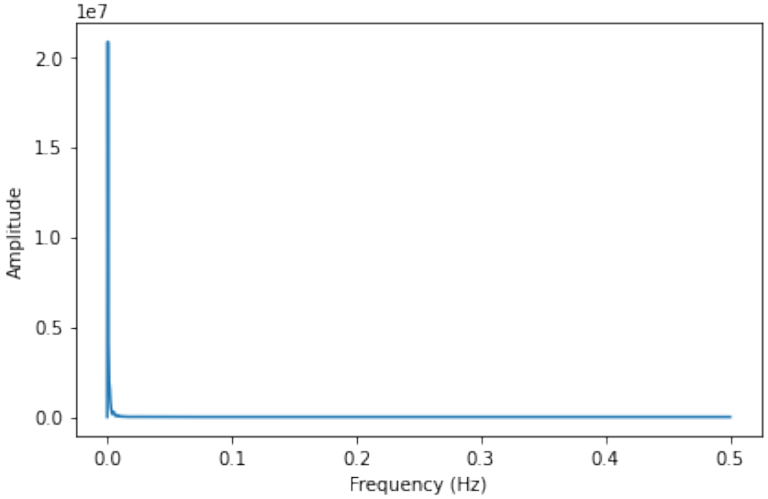

Рис. 5.4: И получили вот такой спектр

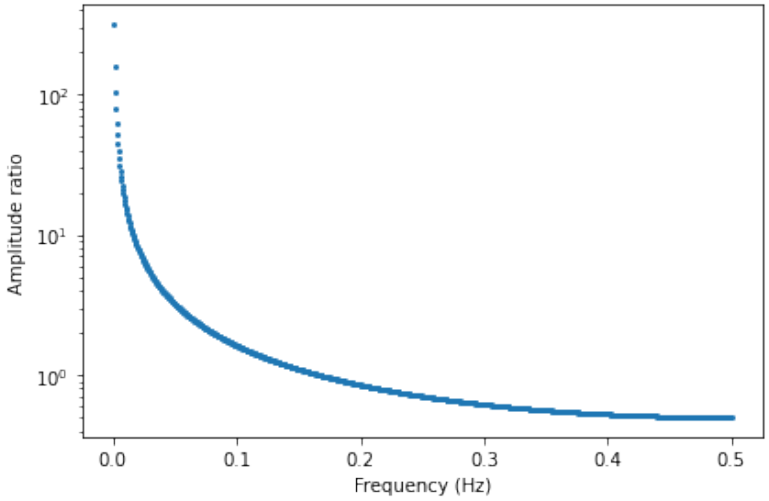

Рис. 5.5: Отношение значения выхода к значению входа

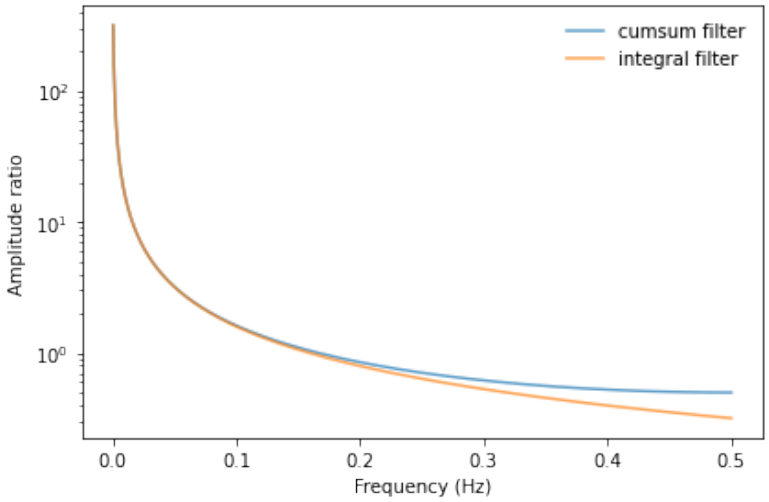

Рис. 5.6: Результат сравнения двух фильтров

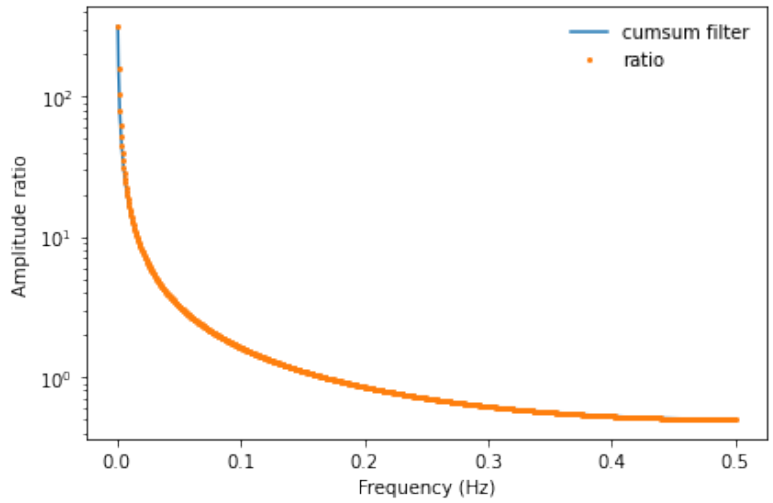

Рис. 5.7: Сравнение вычисленных соотношений с результатом применения фильтра

Они совпадают.

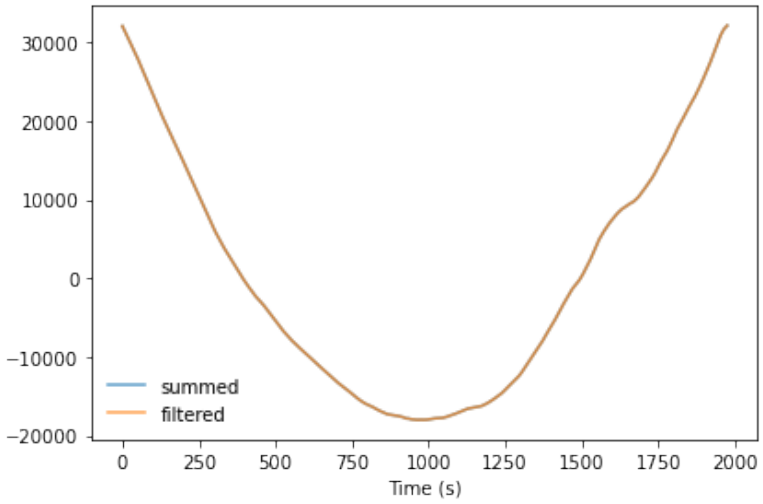

Рис. 5.8: Сравниваем полученные результаты с спектром исходного сигнала

Разница между ними близка к 0. Можно сделать вывод, что с апериодическими сигналами данный фильтр работает так же, как и с периодическими.

5.2 Задание 2

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

```
1       from thinkdsp import TriangleSignal
2
3       in_wave = TriangleSignal(freq = 50).make_wave(duration
    = 0.1, framerate = 44100)
4       in_wave.plot()
5       out_wave = in_wave.diff()
6       out_wave.plot()
7
```

Листинг 5.2: И снова треугольный сигнал

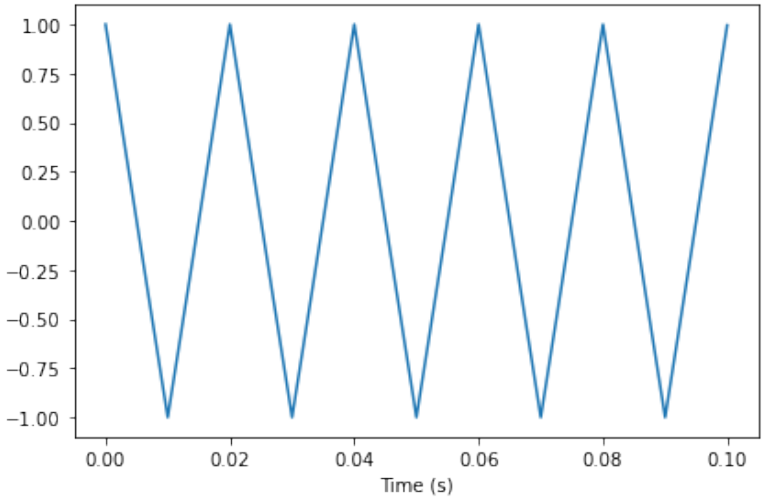

Рис. 5.9: Треугольный сигнал (если вдруг кто забыл, как он выглядит)

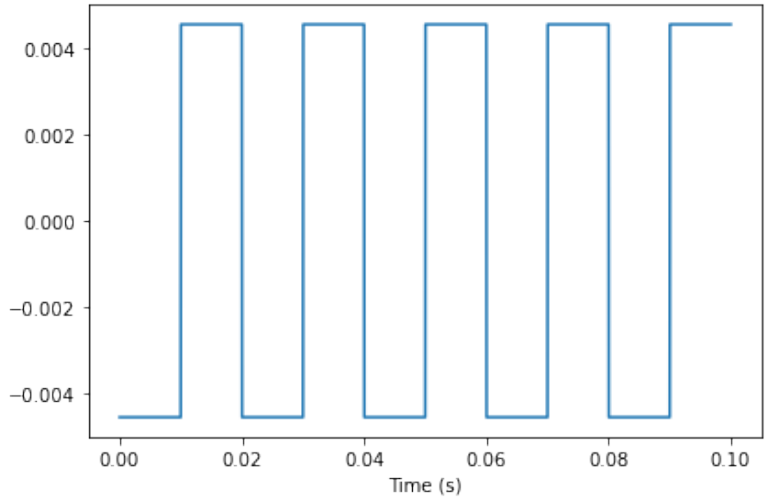

Рис. 5.10: Применяем разностный фильтр

Получился прямоугольный сигнал (немного неожиданно).

```
out_result = in_wave.make_spectrum().differentiate()
out_result.plot()
out_wave2 = out_result.make_wave()
out_wave2.plot()

```

Листинг 5.3: Применение фильтра для дифференцирования

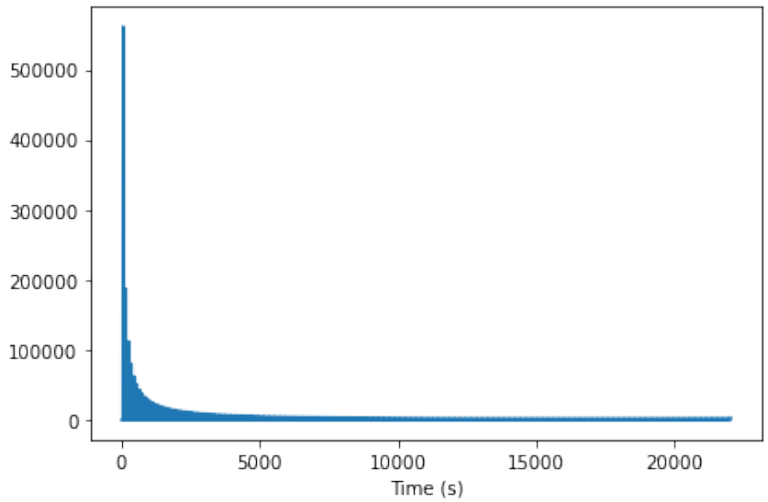

Рис. 5.11: Полученный спектр сигнала

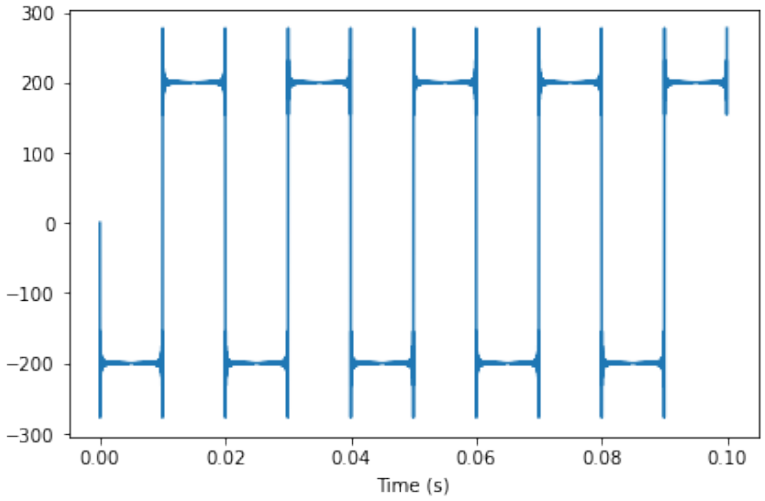

Рис. 5.12: Сигнал после фильтра

Тоже получили прямоугольный сигнал, но с более низким качеством. Проблема заключается в том, что производная не определена в вершинах треугольного сигнала.

5.3 Задание 3

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

```
1       from thinkdsp import SquareSignal
2
3       in_wave = SquareSignal(freq = 50).make_wave(duration
    =0.1, framerate=44100)
4       in_wave.plot()
5
```

Листинг 5.4: Прямоугольный сигнал

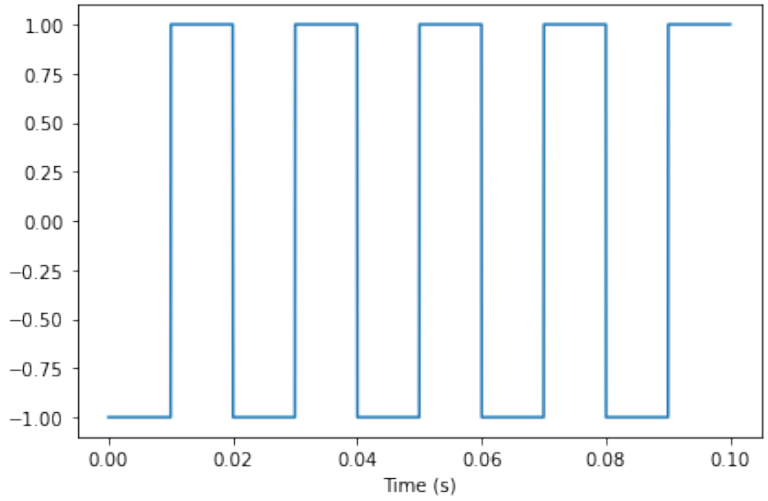

Рис. 5.13: Прямоугольный сигнал

```
out_wave = in_wave.cumsum()
spectrum = in_wave.make_spectrum().integrate()
spectrum.hs[0] = 0
out_wave2 = spectrum.make_wave()
out_wave.unbias()
out_wave.normalize()
out_wave2.normalize()
out_wave.plot()
out_wave2.plot()

```

Листинг 5.5: Сравнение двух фильтров

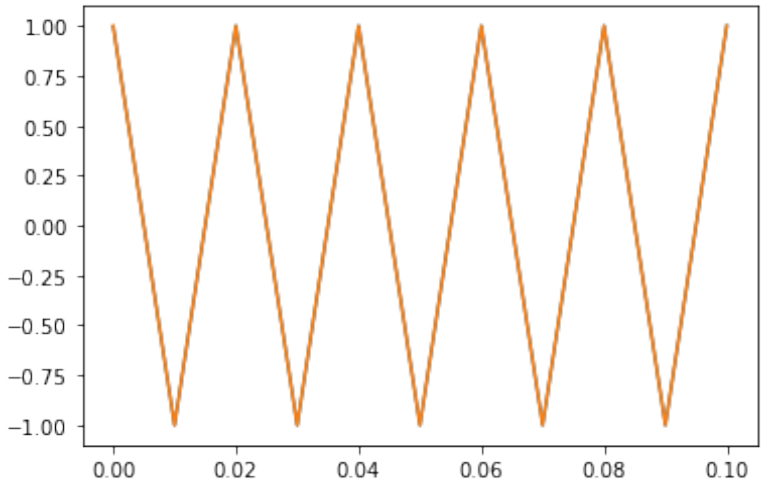

Рис. 5.14: Сравнение результата работы двух фильтров

Получился треугольный сигнал. Результаты этих двух фильтров совпадают небольшой точностью (всего 3 знака после запятой).

5.4 Задание 4

Применение двойного интегрирования на примере пилообразного сигнала.

```
1       from thinkdsp import SawtoothSignal
2
3       in_wave = SawtoothSignal(freq = 50).make_wave(duration
    =0.1, framerate=44100)
4       in_wave.plot()
5
```

Листинг 5.6: Пилообразный сигнал

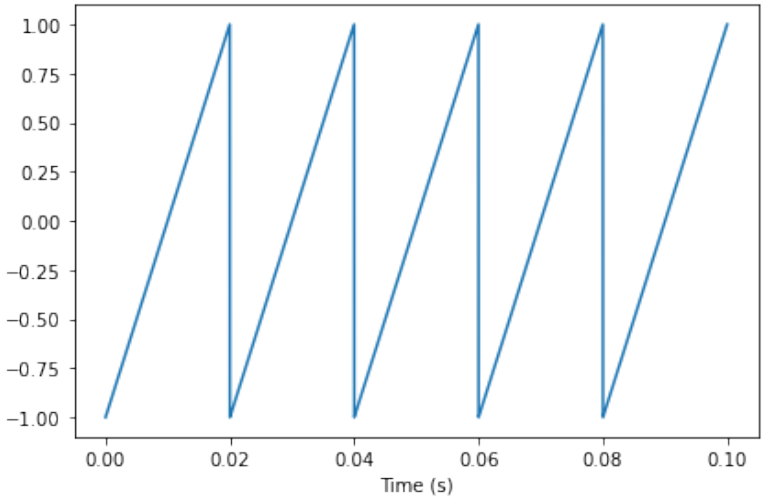

Рис. 5.15: Это пилообразный сигнал

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

```

Листинг 5.7: Два раза применили фильтр для суммирования

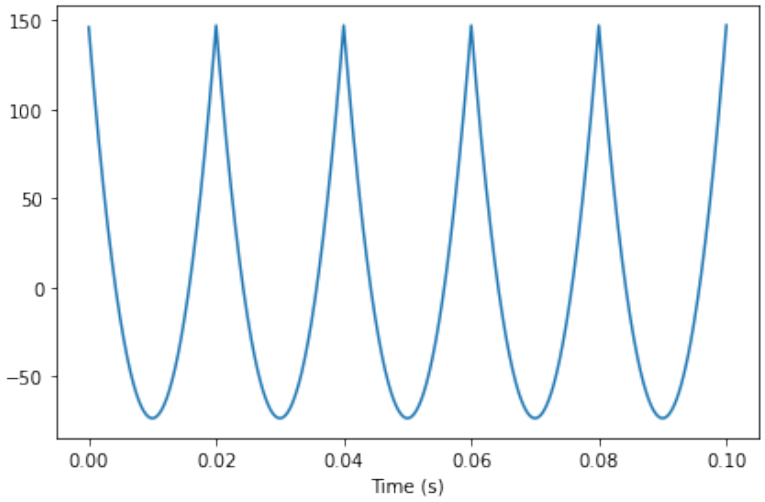

Рис. 5.16: Первый раз - получилась парабола

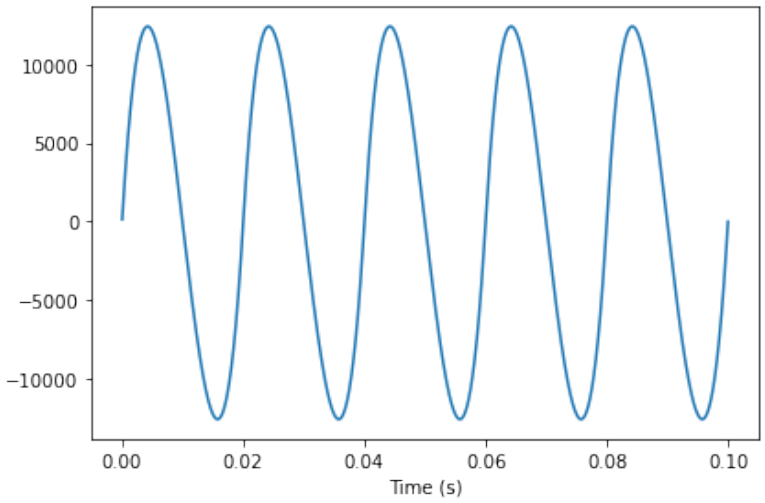

Рис. 5.17: Второй раз - получилась кубическая кривая

```
1        spectrum = (in_wave.make_spectrum().integrate()).
    integrate()
2        spectrum.hs[0] = 0
3        out_wave2 = spectrum.make_wave()
4        out_wave2.plot()
5        out_wave2.make_spectrum().plot(high = 500)
6
```

Листинг 5.8: Два раза интегрируем и строим спектр

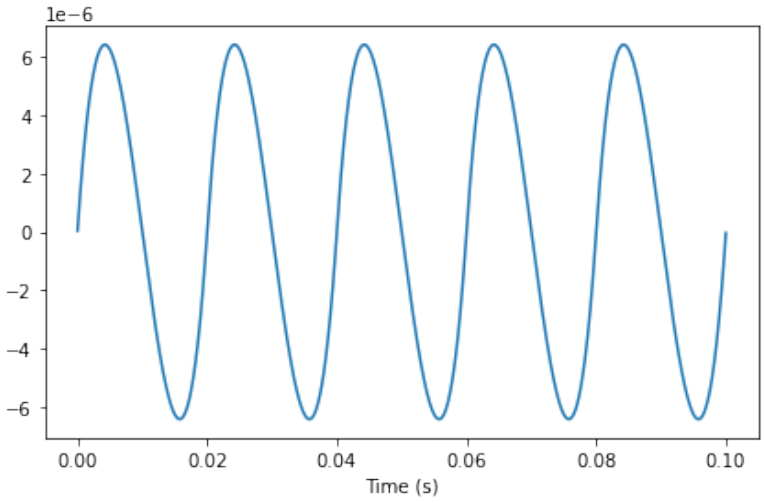

Рис. 5.18: Также получили кубическую кривую

Результат напоминает синусоиду, но это не она.

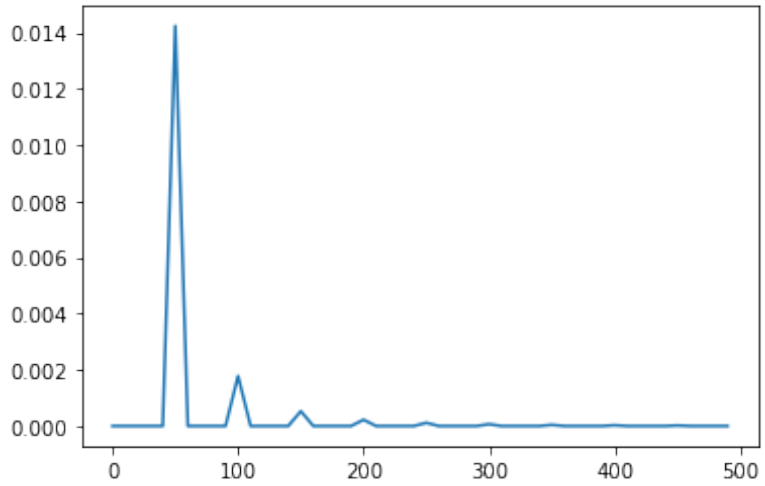

Рис. 5.19: Спектр полученного сигнала

Интегрирование действует как ФНЧ. После двух интегрирований мы отфильтровали почти все гармоники спектра, кроме фундаментальной.

5.5 Задание 5

Два раза применить разностный фильтр и фильтр для дифференцирования на примере кубического сигнала.

```
1       from thinkdsp import CubicSignal
2
3       in_wave = CubicSignal(freq = 0.0005).make_wave(duration
    =10000, framerate=1)
4       in_wave.plot()
5
```

Листинг 5.9: Кубический сигнал

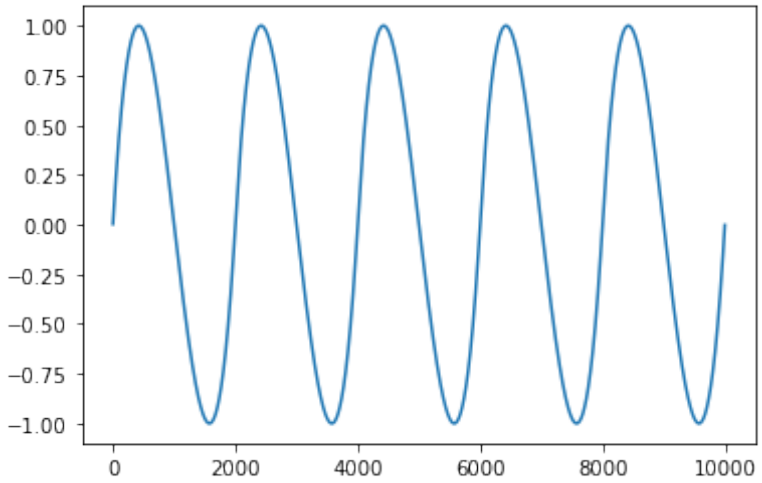

Рис. 5.20: Кубический сигнал

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

```

Листинг 5.10: Применение разностного фильтра

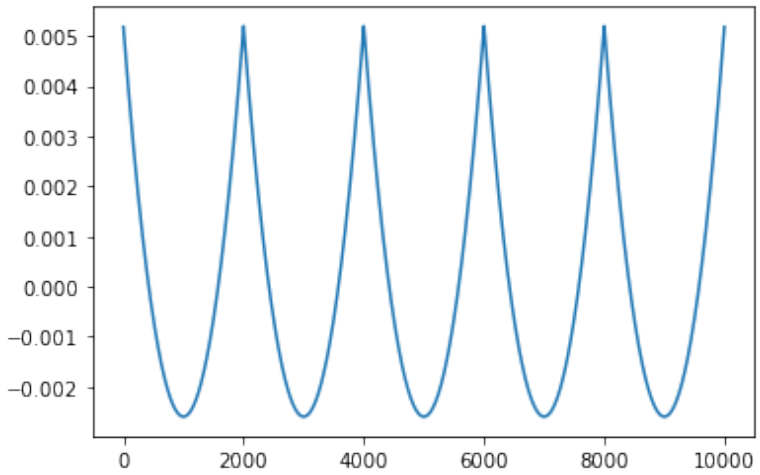

Рис. 5.21: Первый раз - парабола

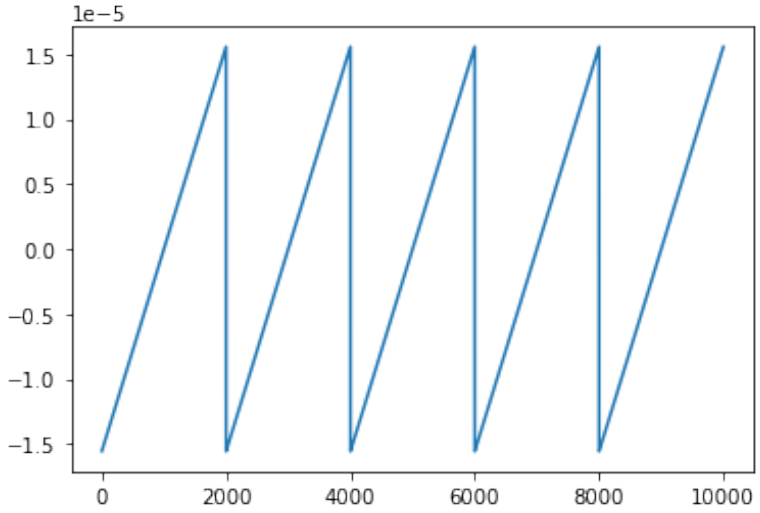

Рис. 5.22: Второй раз - треугольный сигнал

```
    spectrum = in_wave.make_spectrum().differentiate().
differentiate()
    out_wave2 = spectrum.make_wave()
    out_wave2.plot()

```

Листинг 5.11: Применение фильтра для дифференцирования

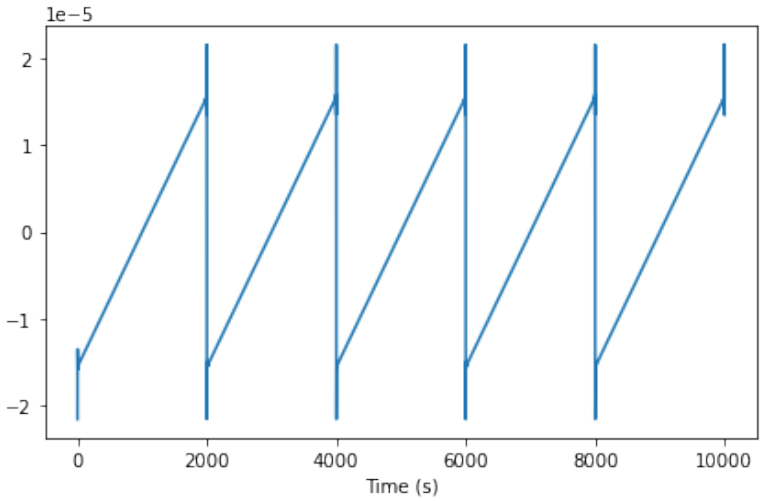

Рис. 5.23: Результат двойного применения фильтра

Получился пилообразный сигнал, но низкого качества. Производная па-

раболического сигнала (после первого раза применения фильтра) в вершинах не определена.

```
from thinkdsp import zero_pad, Wave

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()
deriv_filter = in_wave.make_spectrum()
deriv_filter.hs = (np.pi*2*1j*deriv_filter.fs)**2
diff_filter.plot(label = '2nd diff')
deriv_filter.plot(label = '2nd deriv')

```

Листинг 5.12: Сравнение результатов работы фильтров

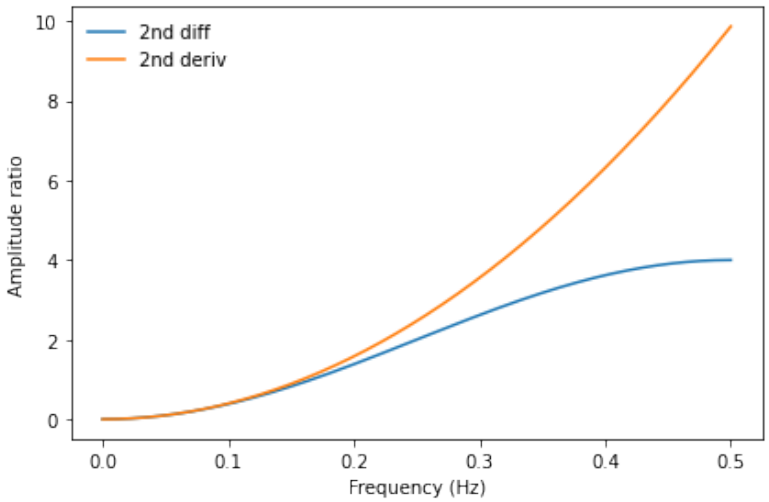

Рис. 5.24: Результат сравнения

Оба фильтра являются ФВЧ. 2-ая производная является параболической и хорошо усиливает высокие частоты. 2-разность является приближением 2-й производной только на низких частотах, а на высоких частотах она существенно отклоняется.

Глава 6

Вывод

В данной работе мы познакомились с методами дифференцирования (разности) и интегрирования (суммы) различных сигналов. Также посмотрели их работу с некоторыми сигналами.