



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЕТ  
по лабораторной работе №4

Линейная фильтрация

Автор: Дженжеруха Кирилл

Учебная группа: Р3242

Номер ИСУ: 368103

Преподаватель: Перегудин Алексей Алексеевич

Санкт-Петербург  
2024

## **Содержание**

<b>Задание 1. Спектральное дифференцирование</b>	<b>3</b>
<b>Задание 2. Линейные фильтры</b>	<b>5</b>
<b>Задание 3. Сглаживание биржевых данных</b>	<b>12</b>

Эта лабораторная работа посвящена методам линейной фильтрации данных.

---

## Задание 1. Спектральное дифференцирование

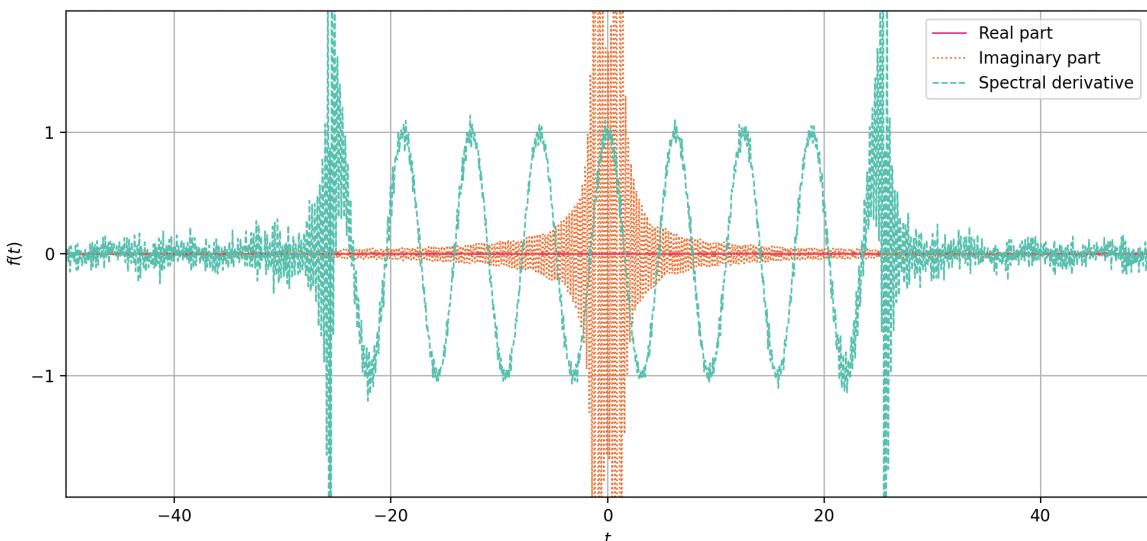
Пусть сигнал передается в виде синусоиды:

$$g(t) = \sin(t).$$

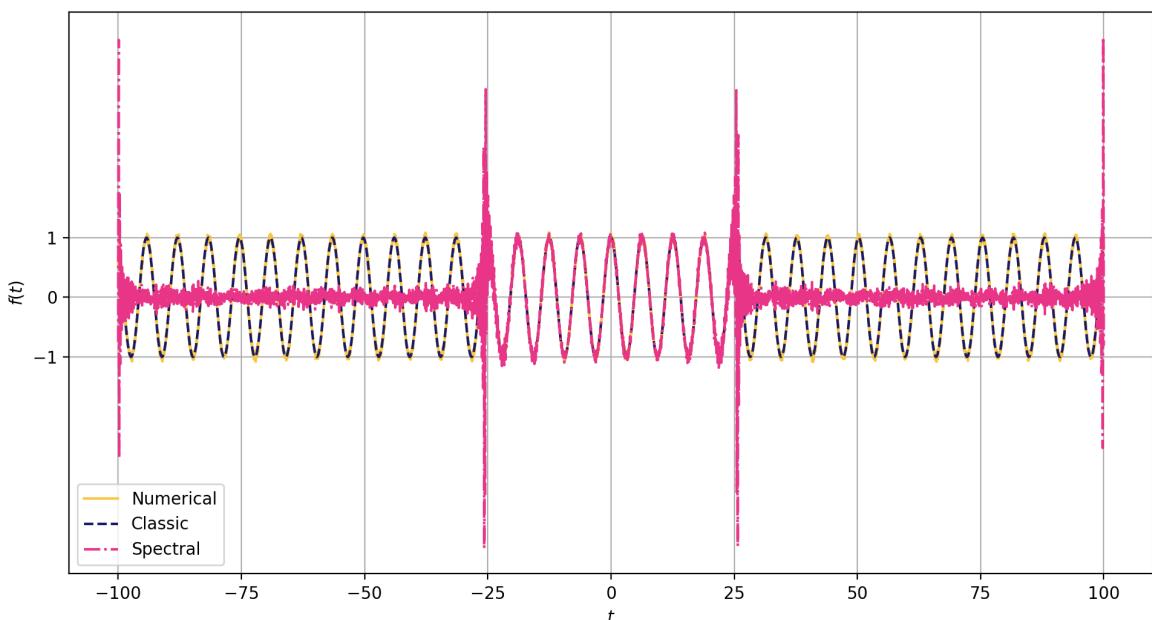
Рассмотрим большой интервал времени  $T = -100 : 100$  с учетом шага дискретизации  $dt$ , а также создадим зашумленную версию сигнала по формуле

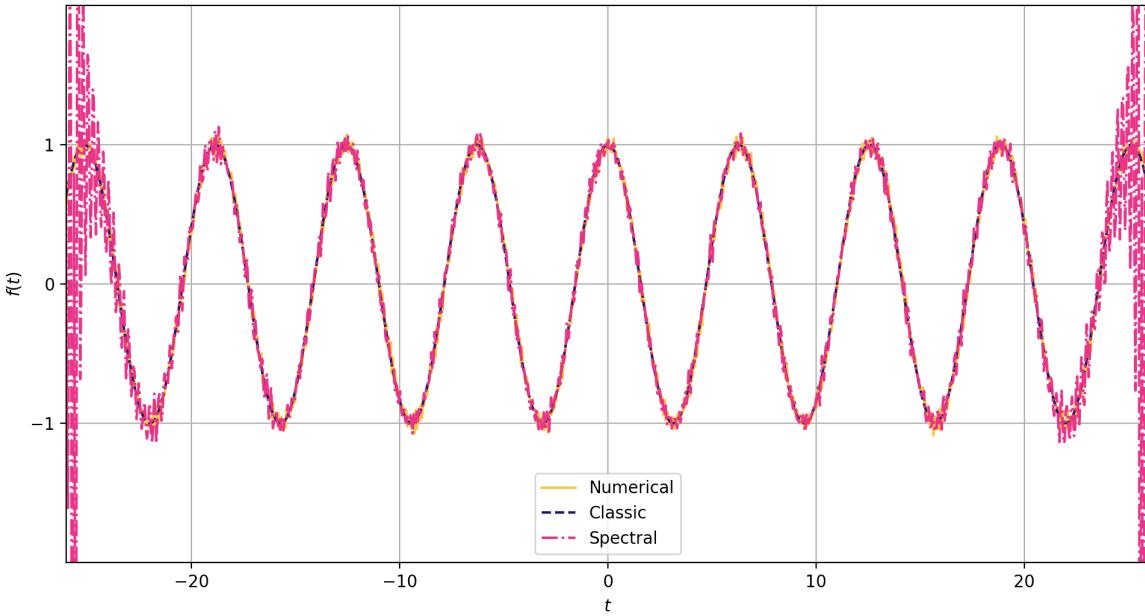
$$u(t) = g + a \cdot (\text{rand}(\text{size}(t)) - 0.5).$$

Найдем численную (дискретную) и спектральную производные зашумленного сигнала и построим некоторые графики:



**Рис. 1** – Графики вещественной и мнимой части образа зашумленного сигнала и график спектральной производной.





**Рис. 2** – Сравнительный график аналитической, дискретной и спектральной производных (сверху) и его увеличенный центральный интервал (снизу).

Сделаем некоторые выводы: пренебрегая погрешностью численного дифференцирования из-за достаточного малого шага дискретизации, мы получаем, что присутствующие искажения сигнала обусловлены добавленной величиной, характеризующей шум. Так как выбор константы задается функцией-рандомайзером для каждого из значений, соответствующих значениям массива времени, искомая добавленная величина является функцией от  $t$  и вносит вклад в искажение производной. Этот вклад так или иначе соразмерен наложенному шуму. Что касается спектральной производной, графики демонстрируют нам, что спектральная производная ведет себя нужным образом только на некотором промежутке времени, зависящем в частности от шага дискретизации.

Приведем листинг кода, необходимого для построения производных:

**Листинг 1.** Код для вычисления аналитической, численной и спектральной производных.

```

1      c = 100
2      dt = 0.05
3      time = np.linspace(-c, c, int(2 * c / dt))
4      freq = np.linspace(-c, c, int(2 * c / dt))
5      a = 0.005
6      u = np.sin(time) + a * (np.random.rand(len(time)) - 0.5)
7
8      def spectral_derivative(u, freq, time):
9          u_image = np.array([])
10         for fr in freq:
11             value = np.trapz(u * np.exp(complex(0, -1) * time * fr), time)
12             u_image = np.append(u_image, value)
13
14         d_u_image = np.array([u_image[i] * freq[i] * complex(0, 1) for i in
15                           range(len(freq))])
16         spec_der = np.array([])
17         for t in time:
18             value = 1 / (2 * np.pi) * np.trapz(d_u_image * np.exp(complex(0,
19                           1) * t * freq), freq)
20             spec_der = np.append(spec_der, value)
21
22     return spec_der

```

```

22
23     def numerical_derivative(array):
24         der = np.array([array[0]])
25         for i in range(len(array) - 1):
26             der = np.append(der, (array[i + 1] - array[i]) / dt)
27         return der
28
29
30     def classic_derivative(t):
31         return np.cos(t)
32
33
34     num_der = numerical_derivative(u)
35     clas_der = classic_derivative(time)
36     spec_der = spectral_derivative(u, freq, time)

```

---

## Задание 2. Линейные фильтры

Пусть сигнал передается в виде квадратной волны, которая задается следующим образом:

$$g(t) = \begin{cases} a, & t \in [t_1, t_2]; \\ 0, & t \in ([t_1, t_2])^C; \end{cases} \quad a = 3, \quad t_1 = -1, \quad t_2 = 2.$$

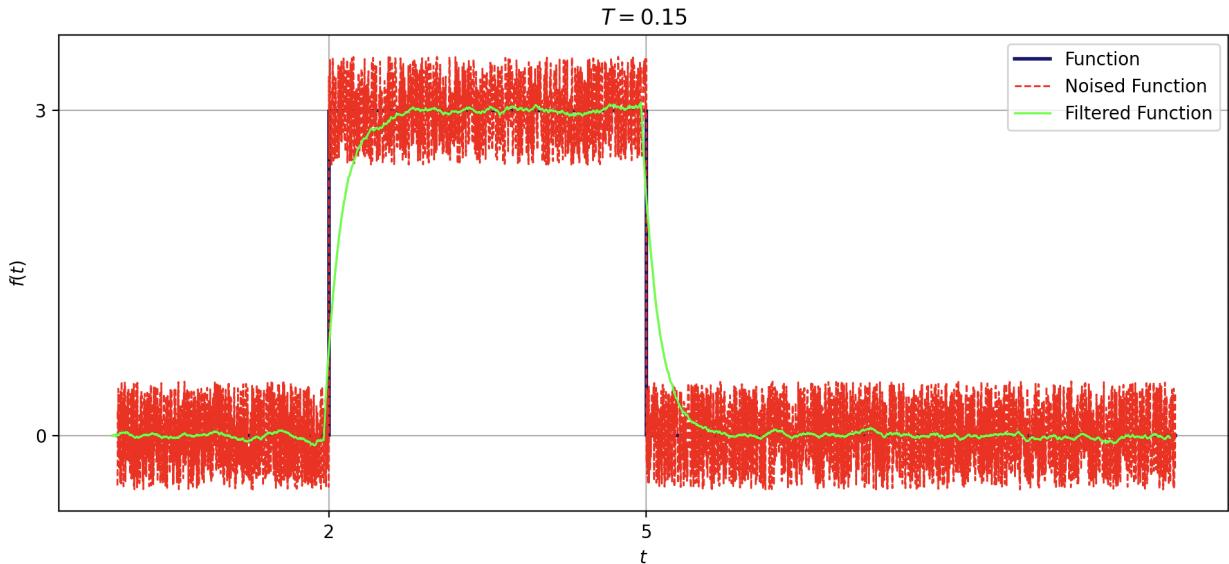
Создадим зашумленную версию сигнала по формуле

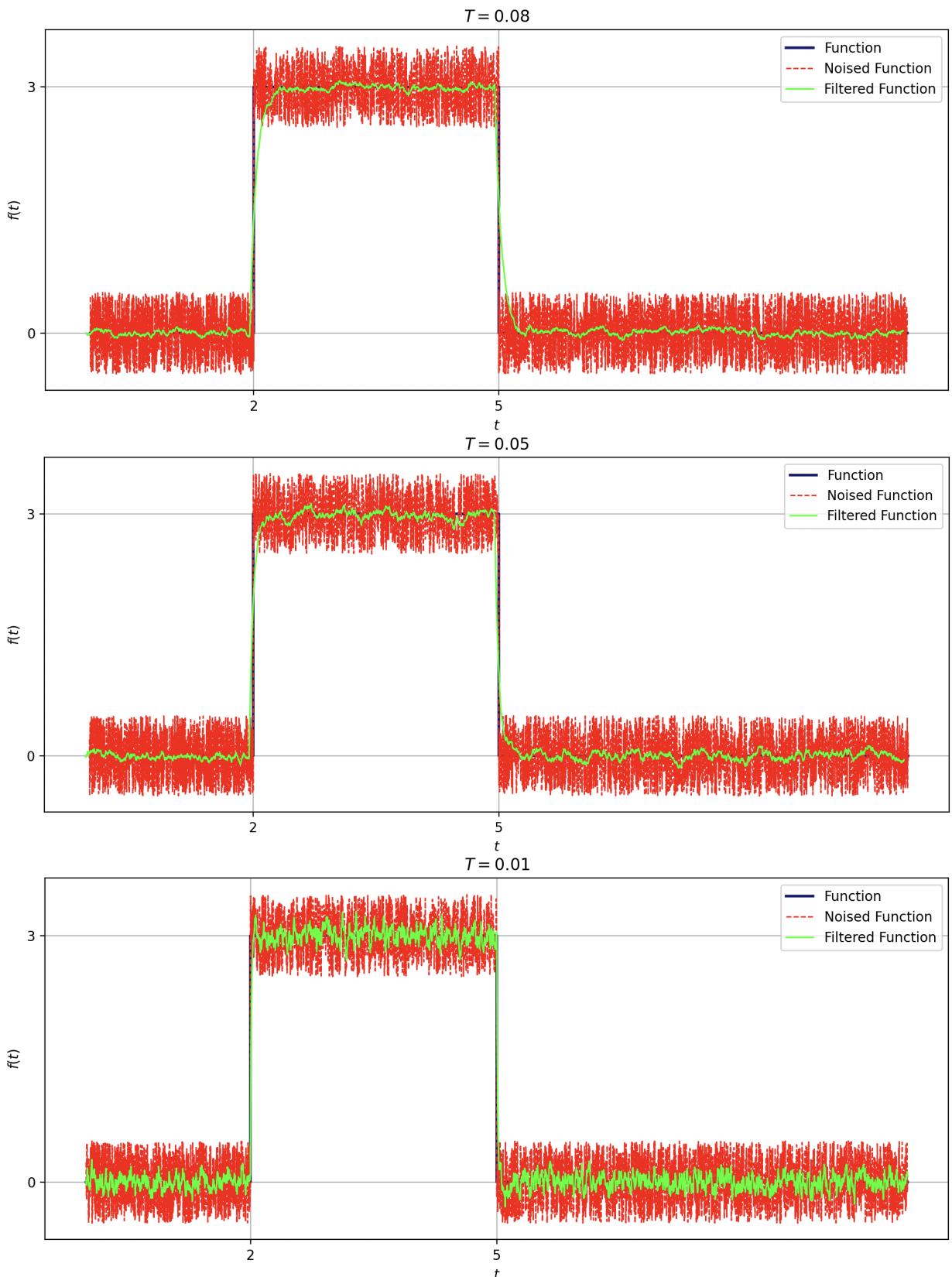
$$u(t) = g + b \cdot (\text{rand}(\text{size}(t)) - 0.5) + c \cdot \sin(d \cdot t).$$

Пропустим зашумленную функцию  $u(t)$  при  $c = 0$  через линейный фильтр первого порядка, заданный следующим образом:

$$W_1(p) = \frac{1}{Tp + 1}, \quad T > 0.$$

Построим графики исходного и фильтрованного сигналов для разных значений  $T$  и  $b$  (при оптимальном значении  $T$ ):

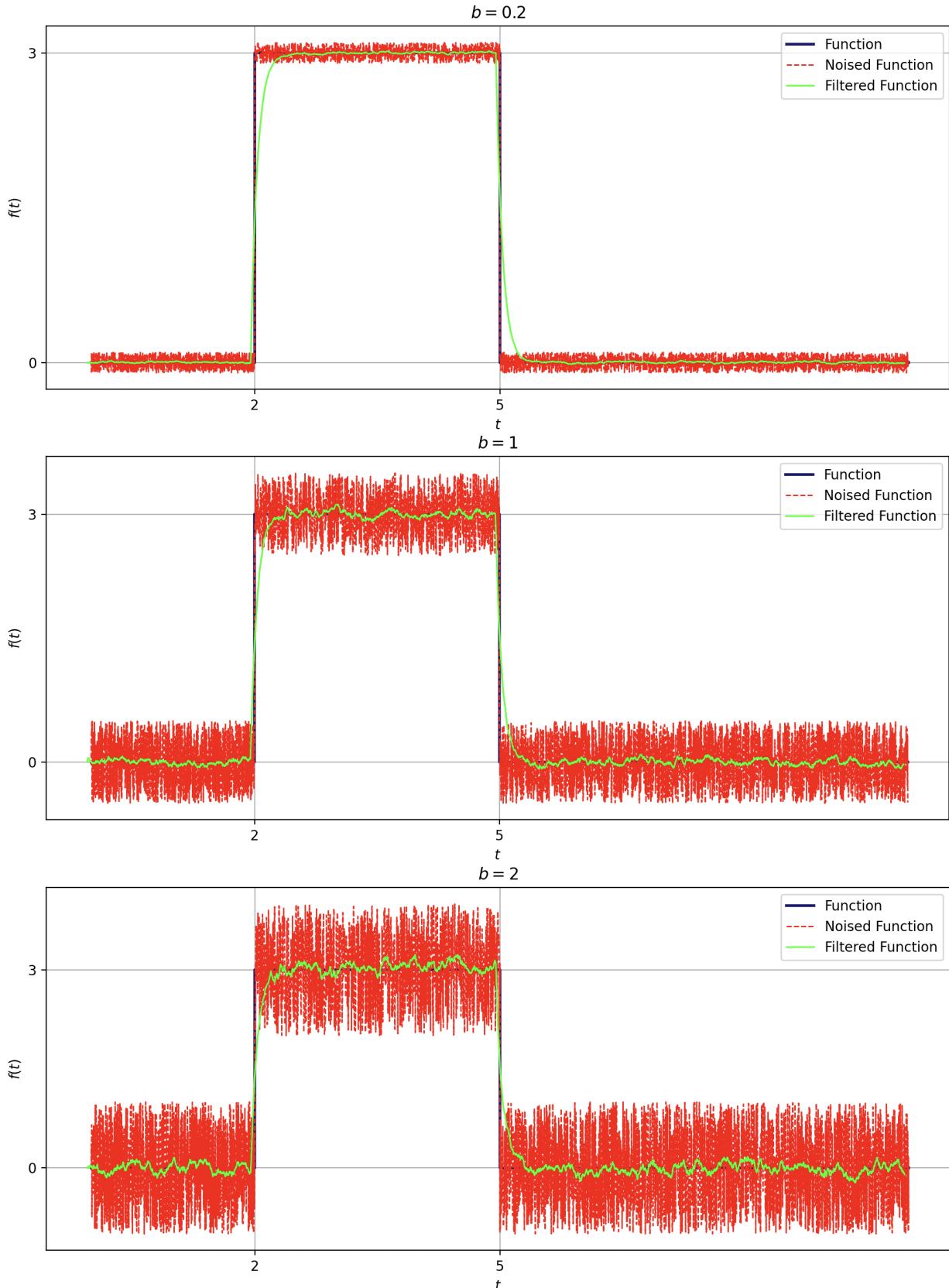


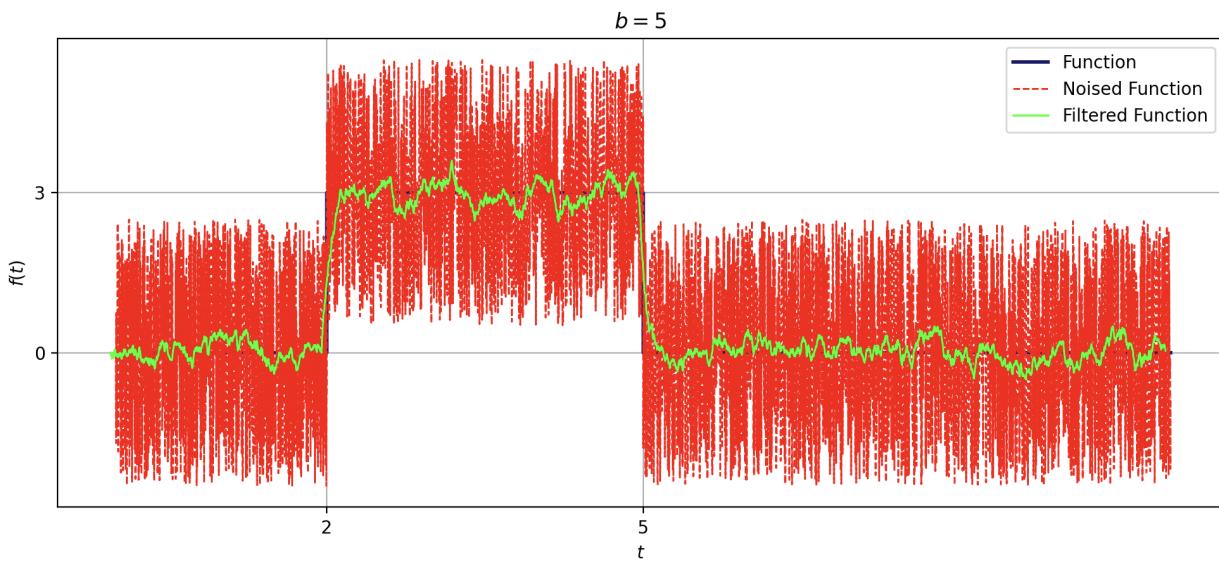


**Рис. 3** – Графики исходного и фильтрованного сигналов для разных значений  $T$ .

Видно, что при значении параметра  $T = 0.08$  наблюдается оптимальное фильтрование зашумленного сигнала. Безусловно, сдвиги и гармоники после фильтрации сохраняются, но линейный фильтр обеспечивает достаточно хорошие исправления искажений.

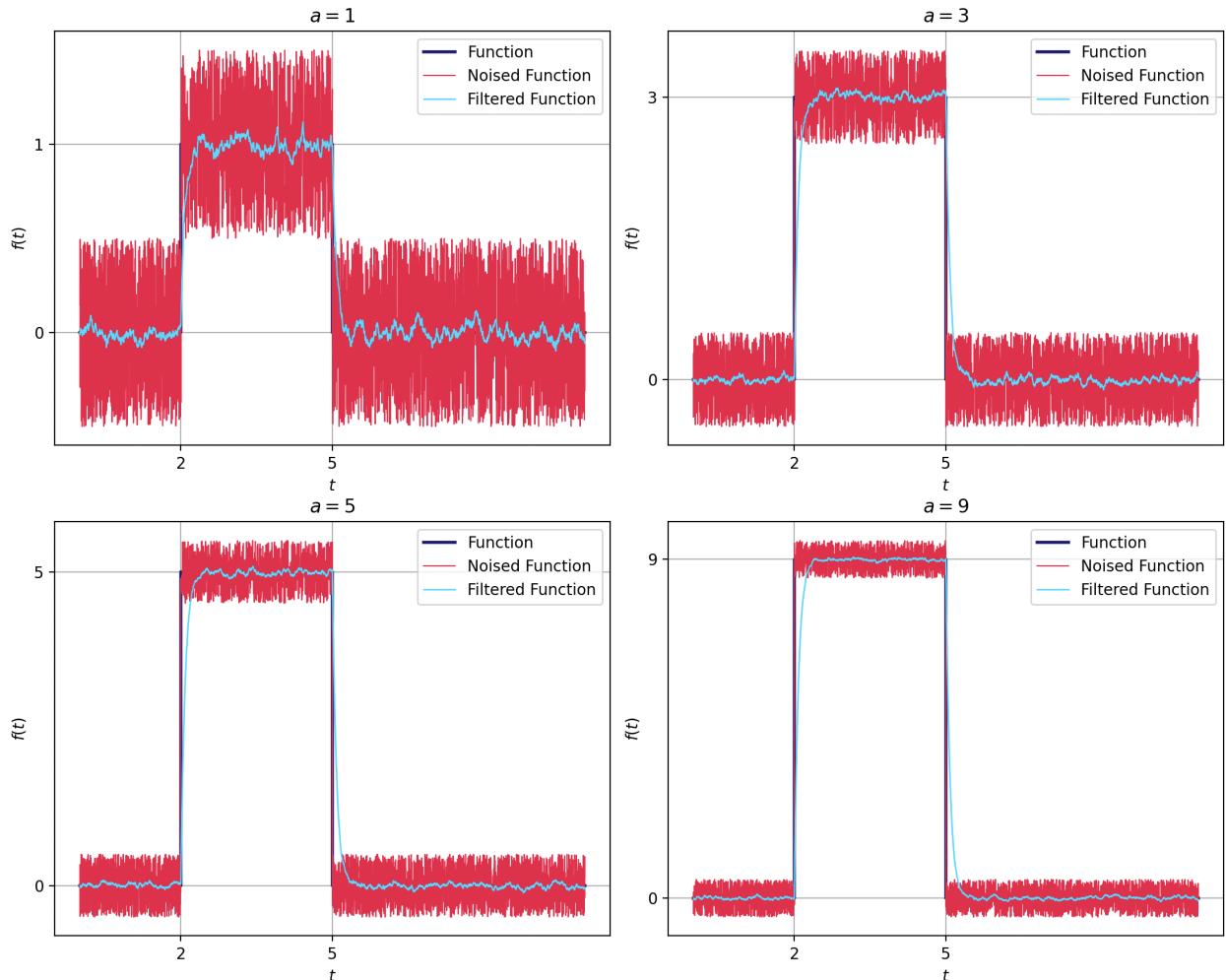
Исследуем качество фильтрации для разных значений коэффициента  $b$  перед зашумляющим слагаемым. Приведем графики данной зависимости:





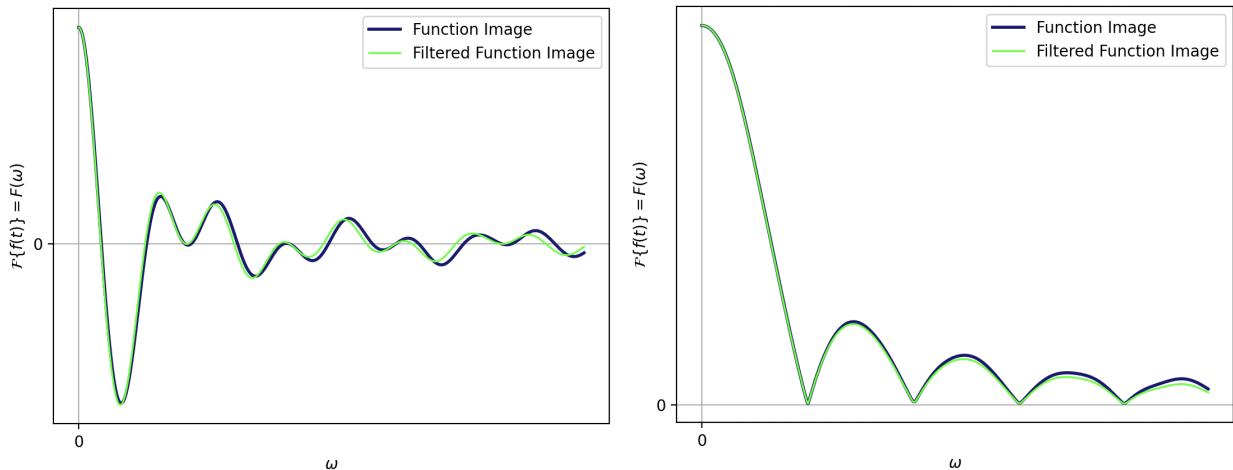
**Рис. 4** – Графики исходного и фильтрованного сигналов для разных значений  $b$  при  $T = 0.08$ .

Кроме того, полезно исследовать поведение фильтра при разных значениях  $a$ , то есть при разном подъеме волны.



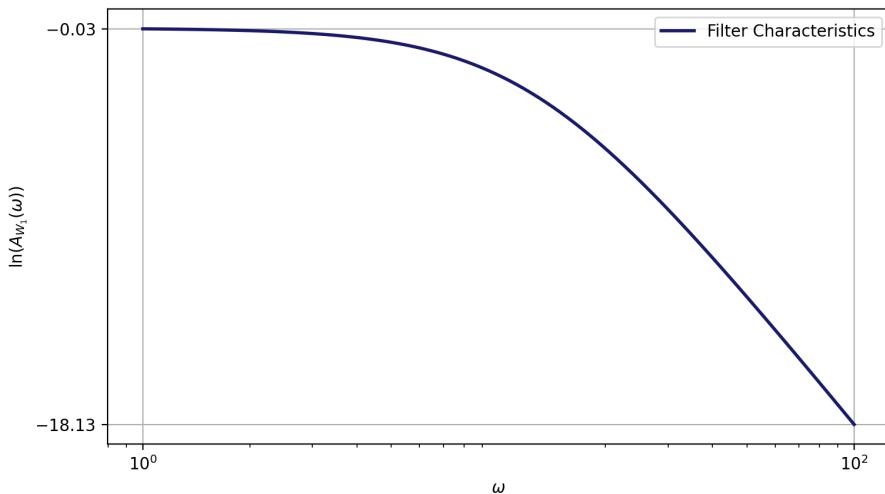
**Рис. 5** – Графики исходного и фильтрованного сигналов для разных значений  $a$  при  $b = 1$  и  $T = 0.08$ .

На графиках Фурье-образов исходные и результирующие искажения выглядят так:



**Рис. 6** – Графики Фурье-образов (слева) и их модулей (справа) исходного и фильтрованного сигналов для  $b = 1$  при  $T = 0.08$ .

Еще один график, передающий достаточно полезную для анализа информацию – график амплитудно-частотной характеристики (АЧХ) примененного линейного фильтра. Построим ее:



**Рис. 7** – График АЧХ примененного фильтра в полулогарифмическом масштабе.

Приведем листинг, содержащий функцию реализации фильтра первого порядка.

**Листинг 2.** Функция реализации линейного фильтра первого порядка.

```

1 def first_order_filter(T1):
2     num = np.array([1])
3     denom = np.array([T1, 1])
4     W = scp.signal.TransferFunction(num, denom)
5
6     return W
7
8 W1 = first_order_filter(T1)
9 T_out, y, x = scp.signal.lsim(W1, u, T)
10 omega, magn, phase = scp.signal.bode(W1)

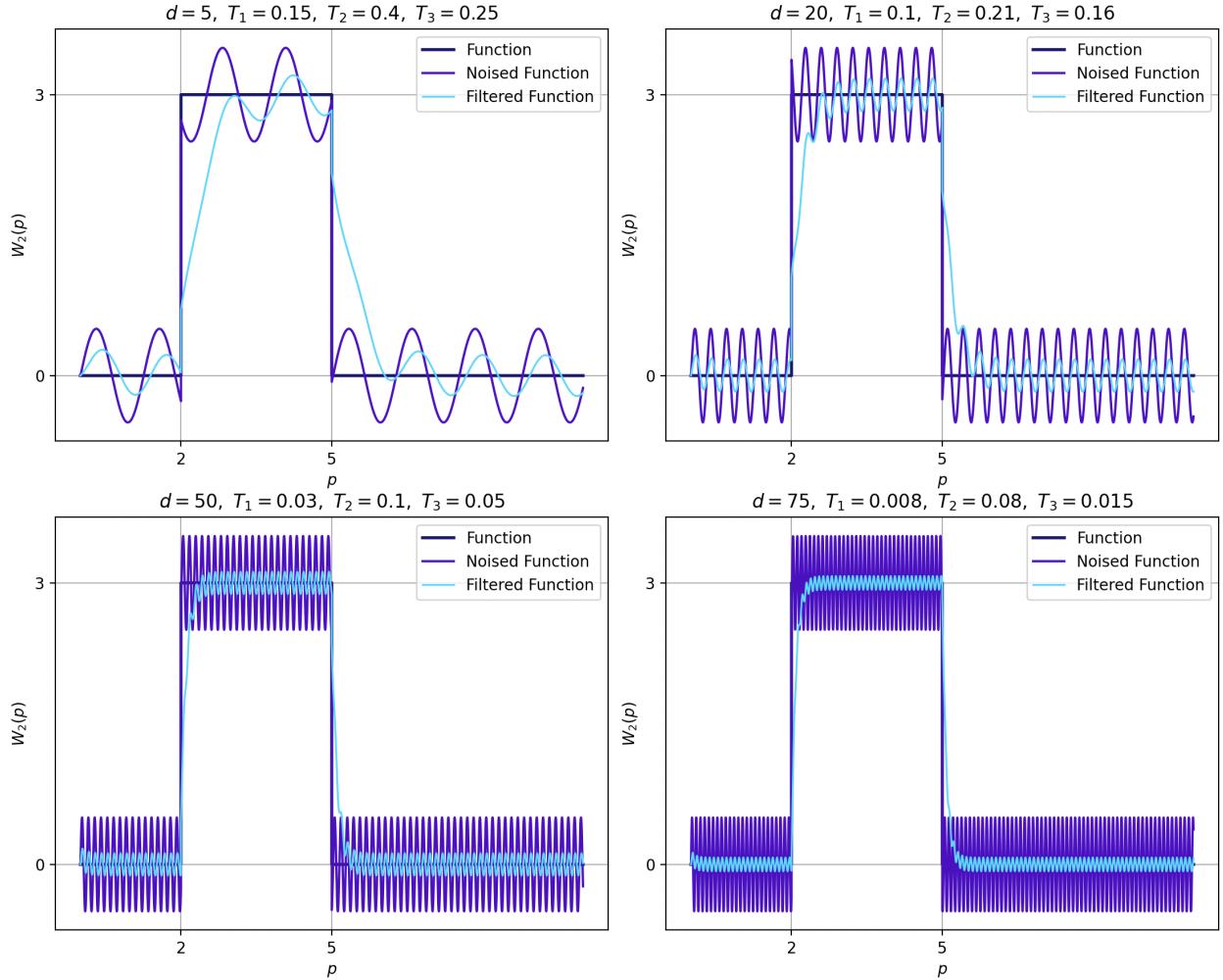
```

Следующим шагом обнулим параметр  $b$  и рассмотрим линейную фильтрацию следующим образом:

$$W_2(p) = \frac{(T_1 p + 1)^2}{(T_2 p + 1)(T_3 p + 1)} = \frac{T_1^2 p^2 + 2T_1 p + 1}{T_2 T_3 p^2 + (T_2 + T_3)p + 1}.$$

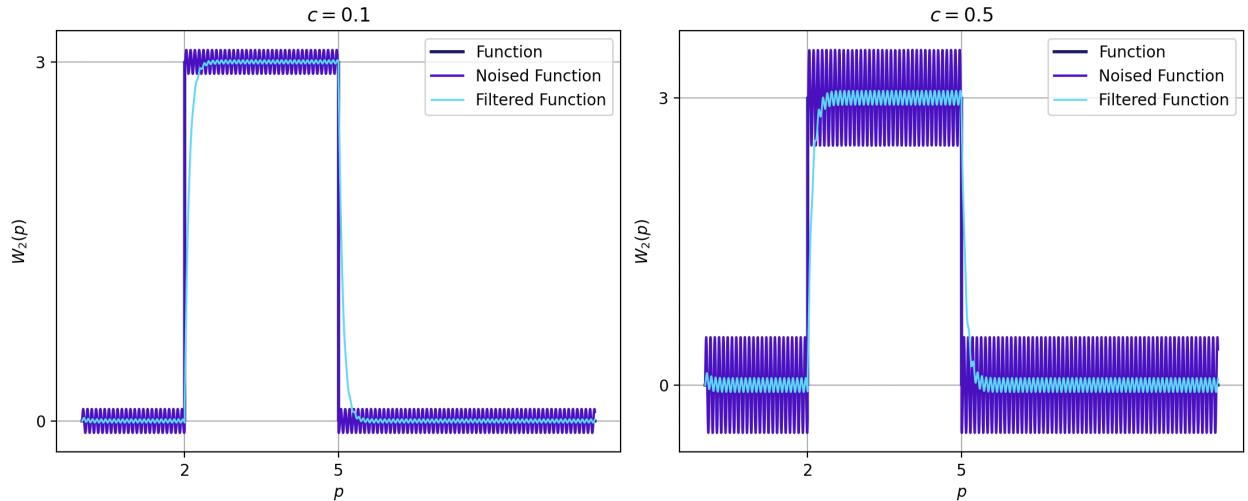
Исследуем влияние выбора параметров  $T_1$ ,  $T_2$ ,  $T_3$  и фильтрации сигнала указанным фильтром при разных значениях частоты синуса – эмпирически подберем значения коэффициентов фильтра таким образом, чтобы синусоидальное искажение убиралось достаточно хорошо при том, что сдвиговое искажение будет не очень значительным.

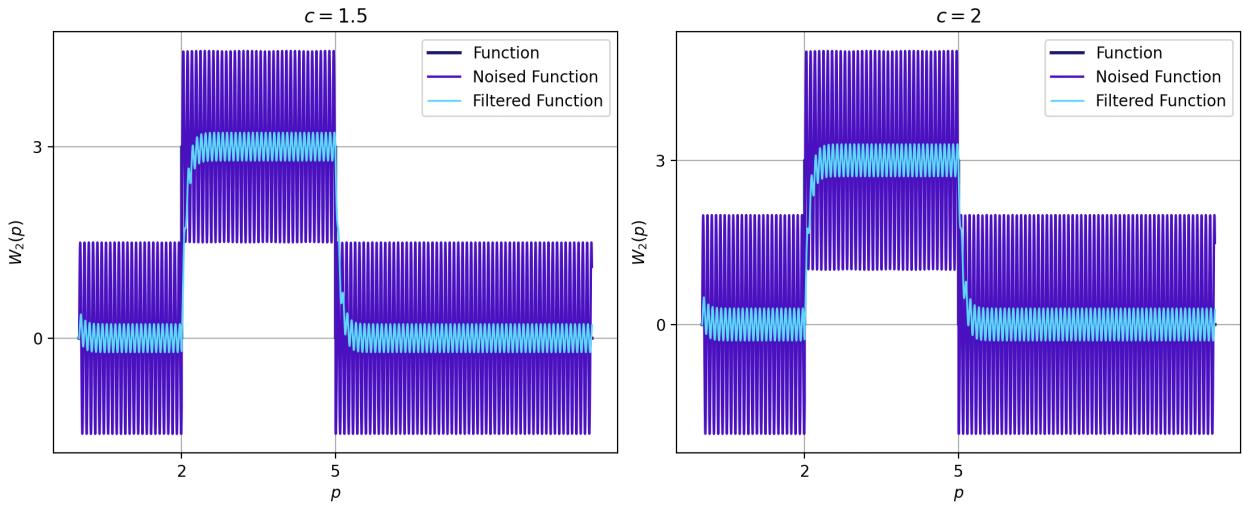
Построим графики исходного и фильтрованного сигнала для разных ситуаций с зафиксированным значением  $c = 0.5$ :



**Рис. 8** – Графики исходного и фильтрованного сигнала для разных значений  $d$ .

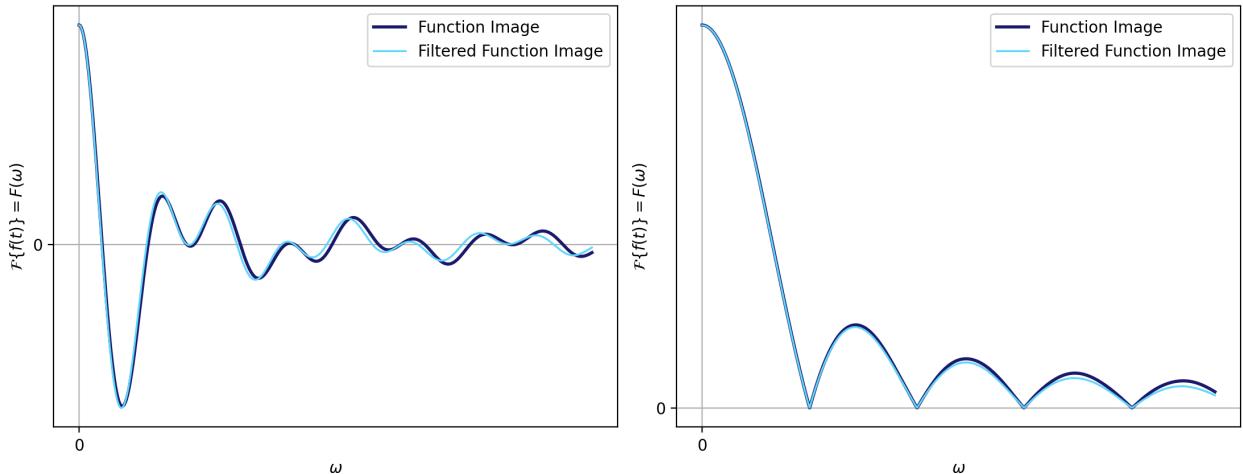
Выберем параметр  $d = 75$  ( $T_1 = 0.008$ ,  $T_2 = 0.08$ ,  $T_3 = 0.015$ ) и построим для данной ситуации графики для разных значений  $c$ :





**Рис. 9** – Графики исходного и фильтрованного сигнала для разных значений  $c$ .

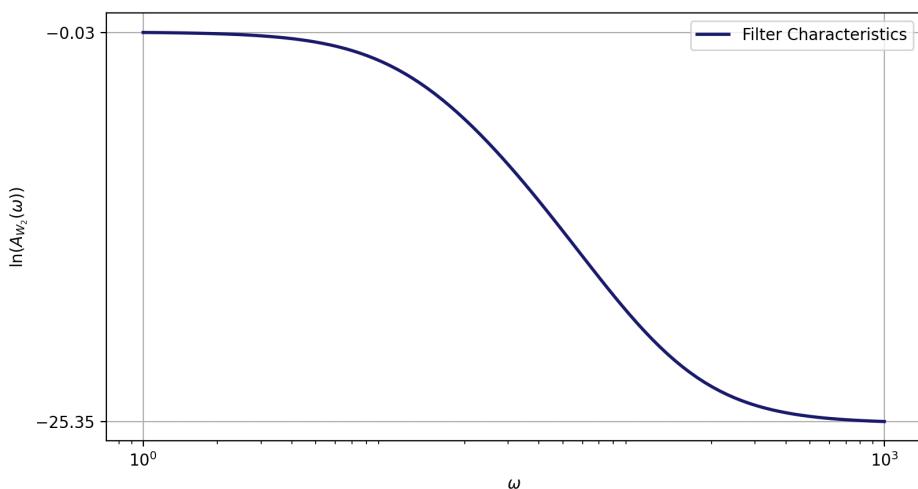
Кроме того, исследуем поведение фильтра при данных фиксированных значениях коэффициентов  $d, T_i$  на графиках их Фурье-образов:



**Рис. 10** – Графики Фурье-образов (слева) и их модулей (справа) исходного и фильтрованного сигнала для  $d = 75$ .

Как несложно заметить, графики Фурье-образов на **Рис. 9** и **Рис. 5** достаточно похожи – они отличаются качественно, поскольку качественно отличается и тип накладываемого шума, но по сути они (в каком-то смысле) соразмерно фильтруют функцию, доводя уровень искажений (как амплитудных, так и временных) до приемлемого уровня.

В заключение, аналогично предыдущему пункту, приведем график амплитудно-частотной характеристики заданного фильтра.



**Рис. 11** – График АЧХ примененного фильтра в полулогарифмическом.

Приведем листинг, содержащий функцию для применения заданного выше фильтра.

**Листинг 3.** Функция реализации специализированного линейного фильтра.

```

1 def specialised_filter(T1, T2, T3):
2     num = np.array([T1 ** 2, 2 * T1, 1])
3     denom = np.array([T2 * T3, T2 + T3, 1])
4     W = scp.signal.TransferFunction(num, denom)
5
6     return W
7
8 W2 = specialised_filter(T1, T2, T3)
9 T_out, y, x = scp.signal.lsim(W2, u, T)
10 omega, magn, phase = scp.signal.bode(W2)
```

---

### Задание 3. Сглаживание биржевых данных

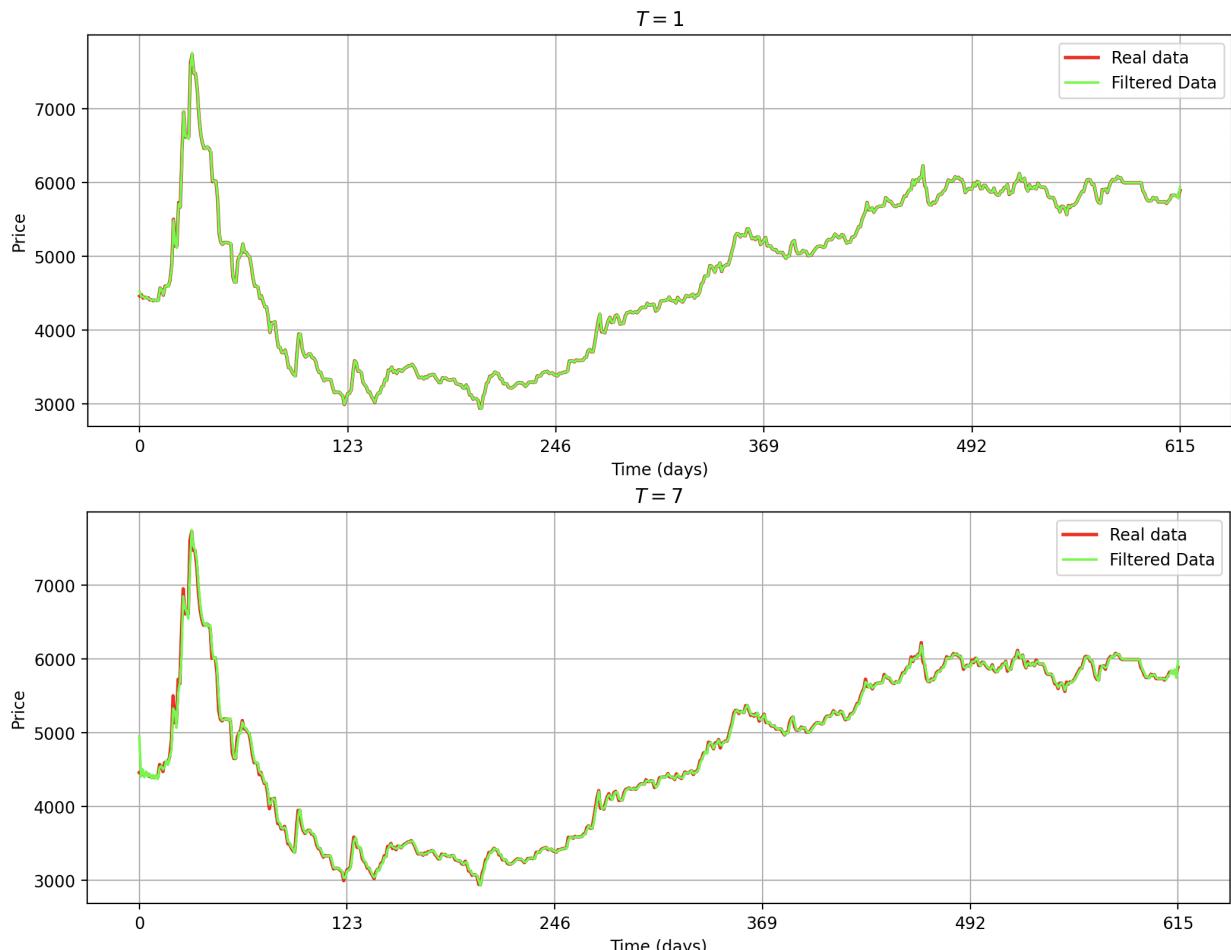
Рассмотрим сведения о стоимости грамма золота в период с 01.02.2022 г. по 01.02.2024 г. с шагом в 1 день и произведем линейную фильтрацию полученных данных с помощью линейного фильтра первого порядка с различными коэффициентами:

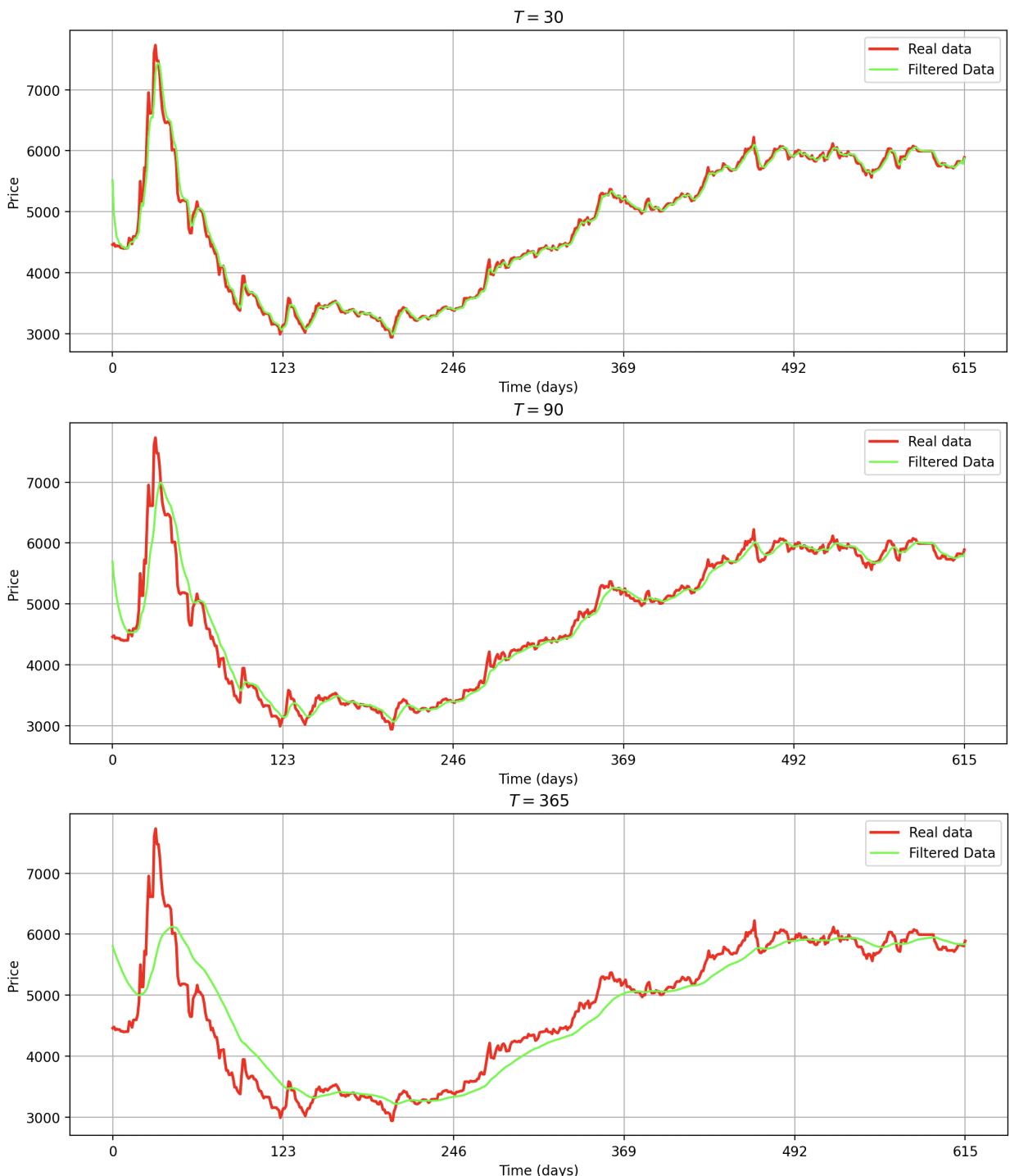
$$W(i\omega) = \frac{1}{i\omega T + 1}, \quad T = \{1, 7, 30, 90, 365\},$$

где  $T$  – количество дней, отображающих временной коэффициент сглаживания.

Глядя на функцию, задающую фильтр, можно предположить, что начальный участок графика с увеличением значения коэффициента  $T$  будет идти “сверху вниз”, подобно гиперболе.

Построим графики исходных и отфильтрованных данных для всех значений  $T$ , принимая во внимание вещественную часть отфильтрованного сигнала:





**Рис. 12** – Графики исходных и фильтрованных значений стоимости золота для разных значений  $T$ .

Таким образом, предположение о поведении графика подтвердилось. Кроме того, при увеличении коэффициента  $T$  мы видим более весомое сглаживание исходных данных. На мой взгляд, наиболее оптимальным стало фильтрование при значении  $T = 90$ , то есть при значении постоянной времени **3 месяца**.

Приведем листинг кода для выполнения фильтрации биржевых данных.

**Листинг 4.** Функция реализации линейного фильтра первого порядка для фильтрации данных.

```

1  with open('filename') as price:
2      reader = csv.reader(price, delimiter=';')
3      data = [row for row in reader]
4

```

```
5     dates = [data[i][0] for i in range(len(data))][:-1]
6     prices = [float(data[i][1]) for i in range(len(data))][:-1]
7
8     dates, prices = np.array(dates), np.array(prices)
9
10    def filter(T, time):
11        return 1 / (T * complex(0, 1) * scp.fft.fftfreq(len(time), 3) + 1)
12
13    def filter_applying(data, filter):
14        return scp.fft.ifft(filter * scp.fft.fft(data))
15
16    time = np.arange(0, len(dates), 1)
17    T = 1
18    filtered_prices = filter_applying(prices, filter(T, time)).real
```