



Федеральное государственное автономное образовательное учреждение высшего образования «Национальный Исследовательский Университет ИТМО»

ЛАБОРАТОРНАЯ РАБОТА №4
ПРЕДМЕТ «ЧАСТОТНЫЕ МЕТОДЫ»
ТЕМА «ЛИНЕЙНАЯ ФИЛЬТРАЦИЯ»

Лектор: Перегудин А. А.
Практик: Пашенко А. В.
Студент: Румянцев А. А.
Поток: ЧАСТ.МЕТ. 1.3

Факультет: СУиР
Группа: R3241

Санкт-Петербург
2024

Содержание

1 Задание 1. Спектральное дифференцирование.	2
1.1 Используемые программы.	5
2 Задание 2. Линейные фильтры.	7
2.1 Фильтр первого порядка.	7
2.2 Специальный фильтр.	13
2.3 Используемые программы.	21
3 Задание 3. Сглаживание биржевых данных.	24
3.1 Используемые программы.	27

1 Задание 1. Спектральное дифференцирование.

Зададим в python список t от -100 до 100 включительно с шагом dt и рассмотрим зашумленный сигнал вида

$$y = \sin(t) + a \cdot (\text{rand}(\text{len}(t)) - 0.5).$$

Построим соответствующий график при переменных $a = 0.2$, $dt = 0.25$. На всех графиках в названии указываются значения используемых параметров для удобства рассматривания различных результатов и последующего сравнения.

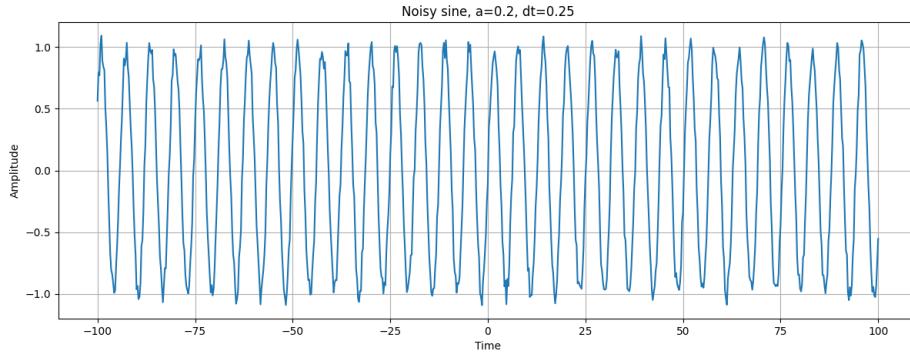


Рис. 1: График зашумленного сигнала.

Найдем численную производную от данного сигнала, используя формулу поэлементного дифференцирования

$$\frac{y(k+1) - y(k)}{dt},$$

после чего построим график.

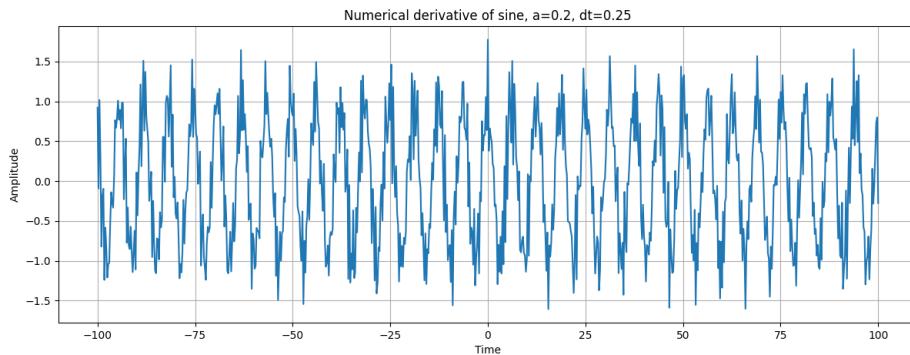


Рис. 2: Численная производная зашумленного сигнала.

Найдем спектральную производную от зашумленного сигнала. Для прямого и обратного преобразования Фурье будем использовать численное интегрирование (trapz). Чтобы превратить Фурье-образ сигнала в Фурье-образ производной, необходимо домножить результат преобразования Фурье на $2\pi i\nu$, где ν – частота (Гц), таким образом получим формулу

$$\mathcal{F} \left\{ \frac{d}{dt} f \right\} = 2\pi i\nu \mathcal{F} \{f\}.$$

Теперь остается только выполнить обратное преобразование Фурье, чтобы получить спектральную производную сигнала. Далее приведены графики вещественной и мнимой компонент Фурье-образа сигнала и его спектральной производной.

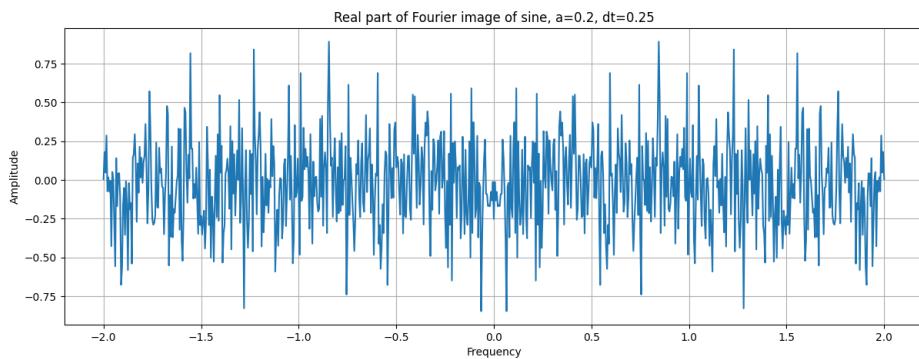


Рис. 3: Вещественная компонента Фурье-образа зашумленного сигнала.

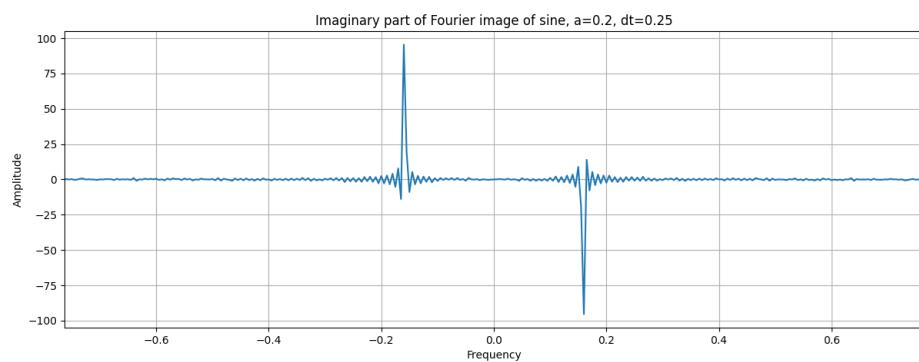


Рис. 4: Мнимая компонента Фурье-образа зашумленного сигнала.

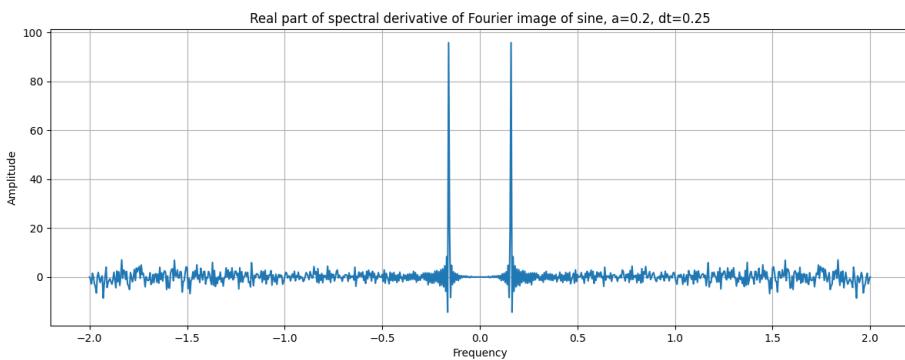


Рис. 5: Вещественная компонента спектральной производной Фурье-образа зашумленного сигнала.

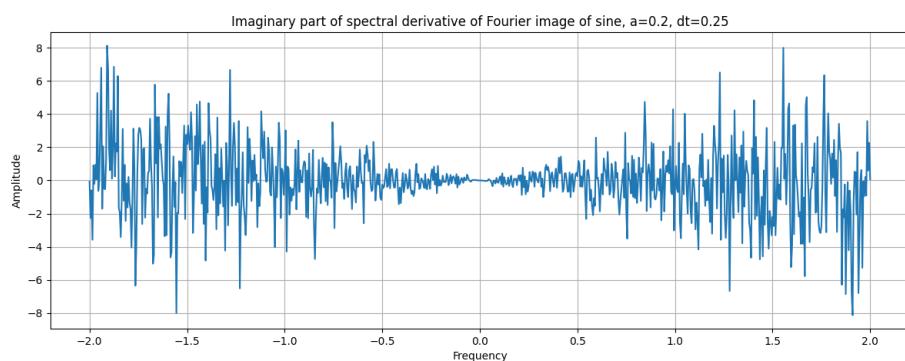


Рис. 6: Мнимая компонента спектральной производной Фурье-образа зашумленного сигнала.

Видим, что вещественная компонента Фурье-образа зашумленного сигнала и его спектральной производной симметричны относительно оси OY , а их мнимые компоненты относительно OX . Подобная симметричность сохраняется в не зависимости от четности исходной функции.

На следующем рисунке приведен график вещественной части спектральной производной зашумленного сигнала, найденной с помощью численного интегрирования. Результат похож на численную производную, но с резкими возрастаниями амплитуд по краям. Данное поведение не зависит от наличия шума в сигнале или выбора шага дискретизации dt .

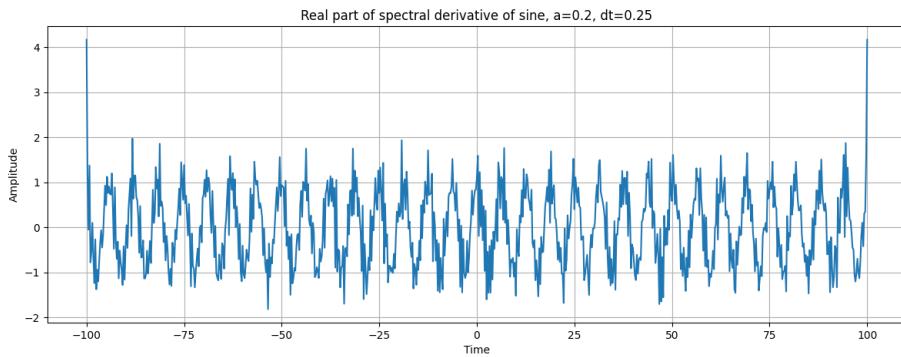


Рис. 7: Вещественная компонента спектральной производной зашумленного сигнала.

Теперь сравним график истинной производной $\cos(t)$ с графиками численной и спектральной производных зашумленного синуса. Оранжевым цветом обозначена спектральная производная, синим численная. Красным цветом выделена производная косинуса.

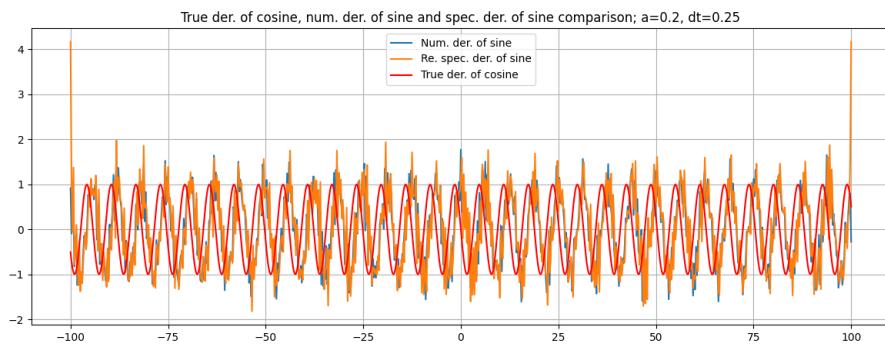


Рис. 8: Сравнительный график производной $\cos(t)$ с численной и спектральной производными зашумленного $\sin(t)$.

Графики численной и спектральной производных похожи друг на друга и на истинную производную косинуса (с разницей в смещении по фазе на $-\pi/2$). Спектральная производная имеет немного больше выбросов по сравнению с численной. Достаточно посмотреть на края графика спектральной производной и на ее амплитуды в точках максимума и минимума каждой волны.

В ходе работы было выяснено, что при маленьком шаге dt или при большом значении a спектральная и численная производные становятся не похожи на истинную производную косинуса. По большей части на графиках видно белый шум. Если исходный сигнал имеет шумы, то его производная будет зашумлена сильнее. Наличие мнимой части у Фурье-образа не зависит от четности исходной функции. Более того, вещественные компоненты

после преобразования Фурье симметричны относительно оси ординат, а мнимые относительно оси абсцисс.

1.1 Используемые программы.

Все графики первого задания строились с помощью языка программирования Python с подключенной библиотекой matplotlib. Далее по ходу работы графики будут строиться той же программой.

```

1 import matplotlib.pyplot as plt
2
3 def build_f(x, y, fz1=16,
4             fz2=5, clr=None, ttl=None,
5             grid=True, legend=False, xlab=None,
6             ylab=None, x11=None, x12=None,
7             y11=None, y12=None, lbl=None,
8             ls='-', ticks=None, rot=None):
9     plt.plot(x, y, color=clr, label=lbl, linestyle=ls)
10    plt.xlabel(xlab)
11    plt.ylabel(ylab)
12    plt.xlim(x11, x12)
13    plt.ylim(y11, y12)
14    plt.xticks(ticks, rotation=rot)
15    plt.title(ttl)
16    plt.gcf().set_size_inches(fz1, fz2)
17    plt.grid(grid)
18    if legend:
19        plt.legend()
20    plt.show()
21
22 def build_fs(x, y: list, colors: list=None,
23              labels: list=None, fz1=16, fz2=5,
24              ttl=None, grid=True, legend=False,
25              xlab=None, ylab=None, x11=None,
26              x12=None, y11=None, y12=None,
27              ls:list=None, ticks=None, rot=None):
28     if (y is None or len(y) <= 0):
29         print('y is None or its len <= 0')
30         return
31     if (colors is None): colors = [None] * len(y)
32     if (labels is None): labels = [None] * len(y)
33     if (ls is None): ls = ['-' ] * len(y)
34
35     for k in range(len(y)):
36         plt.plot(x, y[k], color=colors[k],
37                   label=labels[k], linestyle=ls[k])
38     plt.xlabel(xlab)
39     plt.ylabel(ylab)
40     plt.xlim(x11, x12)
41     plt.ylim(y11, y12)
42     plt.xticks(ticks, rotation=rot)
43     plt.title(ttl)
44     plt.gcf().set_size_inches(fz1, fz2)
45     plt.grid(grid)
46     if legend: plt.legend()
47     plt.show()
```

Листинг 1: Файл с программой для построения графиков.

Для нахождения Фурье-образа и производных использовалась библиотека numpy.

```

1 import numpy as np
2
3 def trapz(y, t, v):
4     Y = []
5     for k in v:
6         Y_k = np.trapz(y * np.exp(-1j * 2 * np.pi * k * t), t)
7         Y.append(Y_k)
8     return Y
9
10 def undo_trapz(Y, t, v):
11     y = []
12     for k in t:
13         y_k = np.trapz(Y * np.exp(1j * 2 * np.pi * k * v), v)
14         y.append(y_k)
15     return y
16
17 def numerical_diff(y, dt):
18     ndiff = []
19     for k in range(len(y) - 1):
20         ndiff_k = (y[k + 1] - y[k]) / dt
21         ndiff.append(ndiff_k)
22     return ndiff
23
24 def spectral_diff(y, t, v):
25     Y = trapz(y, t, v)
26     dY = 2 * np.pi * 1j * v * Y
27     spdiff = undo_trapz(dY, t, v)
28     return spdiff, Y, dY

```

Листинг 2: Программа для вычисления Фурье-образа численным интегрированием и производных.

Программа, где используются написанные функции и задаются необходимые параметры, расположена ниже.

```

1 import numpy as np
2
3 import build_func as bf
4 import fourier_math as fm
5
6 T = 200
7 dt = 0.25
8 t = np.arange(-T / 2, T / 2 + dt, dt)
9 y = np.sin(t)
10
11 a = 0.2
12 y += a * (np.random.rand(len(t)) - 0.5)
13
14 ndsin = fm.numerical_diff(y, dt)
15 ndsin.append(y[-1] / 2)
16
17 V = 1 / dt
18 dv = 1 / T
19 v = np.arange(-V / 2, V / 2 + dv, dv)
20 spdsin, Y, dY = fm.spectral_diff(y, t, v)
21 tdcos = -np.sin(t)
22
23 bf.build_f(t, y, ttl=f'Noisy sine, a={a}, dt={dt}',
24             xlab='Time', ylab='Amplitude')

```

```

25     bf.build_f(v, np.array(Y).real,
26                 ttl=f'Real part of Fourier image of sine, a={a}, dt={dt}',
27                 xlab='Frequency', ylab='Amplitude')
28     bf.build_f(v, np.array(Y).imag,
29                 ttl=f'Imaginary part of Fourier image of sine, a={a}, dt={dt}',
30                 xlab='Frequency', ylab='Amplitude', x11=-0.763, x12=0.763)
31     bf.build_f(v, dY.real,
32                 ttl=f'Real part of spectral derivative of Fourier image of
33                 sine, a={a}, dt={dt}',
34                 xlab='Frequency', ylab='Amplitude')
35     bf.build_f(v, dY.imag,
36                 ttl=f'Imaginary part of spectral derivative of Fourier image
37                 of sine, a={a}, dt={dt}',
38                 xlab='Frequency', ylab='Amplitude')

39     bf.build_f(t, ndsin,
40                 ttl=f'Numerical derivative of sine, a={a}, dt={dt}',
41                 xlab='Time', ylab='Amplitude')
42     bf.build_f(t, np.array(spdsin).real,
43                 ttl=f'Real part of spectral derivative of sine, a={a}, dt={dt}',
44                 xlab='Time', ylab='Amplitude')
45     bf.build_fs(t, y=[ndsin, np.array(spdsin).real, tdcos],
46                  colors=[None, None, 'r'], legend=True,
47                  labels=['Num. der. of sine', 'Re. spec. der. of sine', 'True
48                  der. of cosine'],
49                  ttl=f'True der. of cosine, num. der. of sine and spec. der.
50                  of sine comparison; a={a}, dt={dt}')

```

Листинг 3: Алгоритм, использующий представленные ранее программы.

2 Задание 2. Линейные фильтры.

Рассмотрим сигнал вида

$$u = g + b \cdot (\text{random}(\text{len}(t)) - 0.5) + c \cdot \sin(d \cdot t),$$

который будем пропускать через линейные фильтры.

2.1 Фильтр первого порядка.

Положим $c = 0$, $a = 1$, $b = 0.6$, $d = 0.7$. Будем задавать различные значения постоянной времени $T > 0$ и пропускать сигнал через линейный фильтр первого порядка

$$W_1(p) = \frac{1}{Tp + 1},$$

после чего построим сравнительные графики и исследуем влияние параметров T и a на эффективность фильтрации.

Далее расположены сравнительные графики исходного и фильтрованного сигналов, графики модулей их Фурье-образов, а также АЧХ и ЛАЧХ фильтра. В названии графиков указаны значения используемых параметров. Синим цветом обозначаются функции, относящиеся к исходному сигналу, оранжевым – к фильтрованному. Дополнительно на графике во временной области отрисована исходная функция $g(t)$ для более наглядного рассмотрения работы фильтрации.

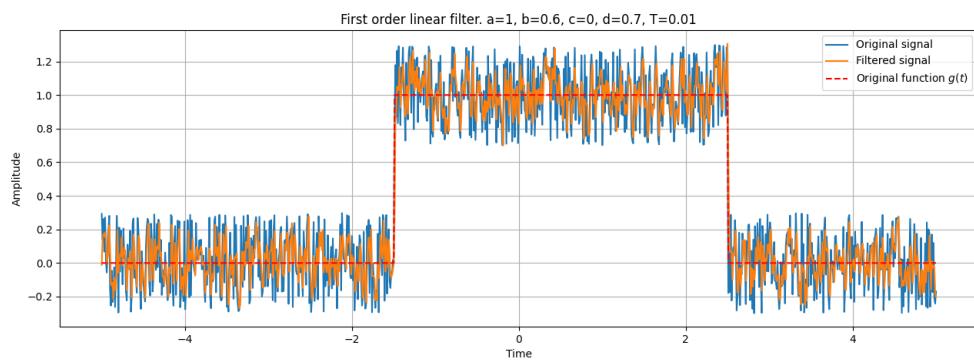


Рис. 9: График исходного и фильтрованного сигналов (1).

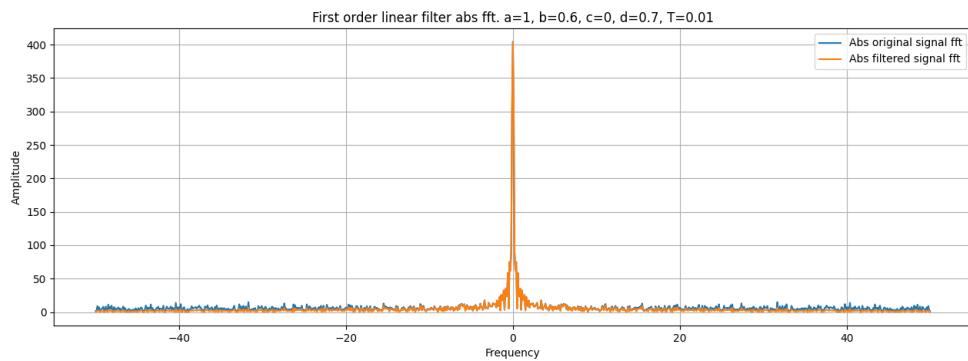


Рис. 10: График модулей Фурье-образа исходного и фильтрованного сигналов (1).

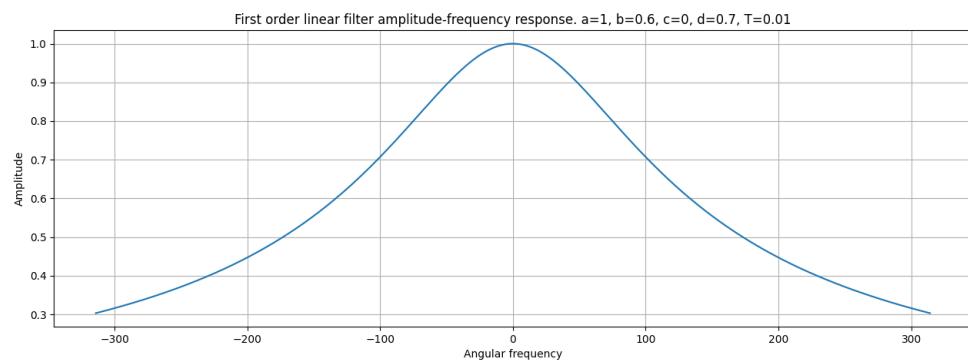


Рис. 11: График амплитудно-частотной характеристики фильтра (1).

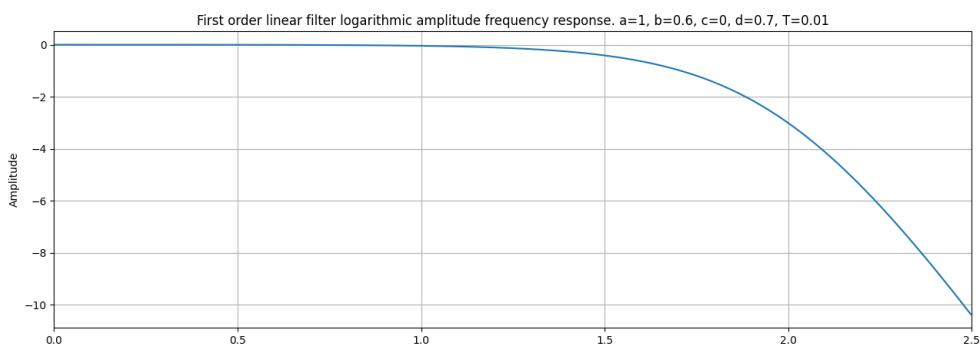


Рис. 12: График логарифмической амплитудно-частотной характеристики фильтра (1).

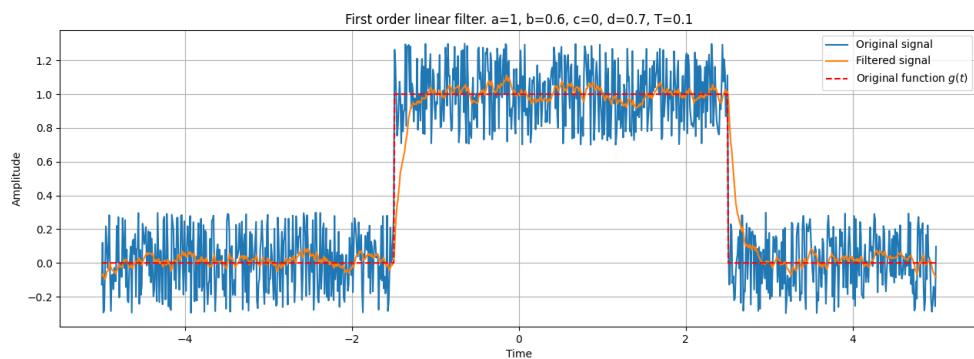


Рис. 13: График исходного и фильтрованного сигналов (2).

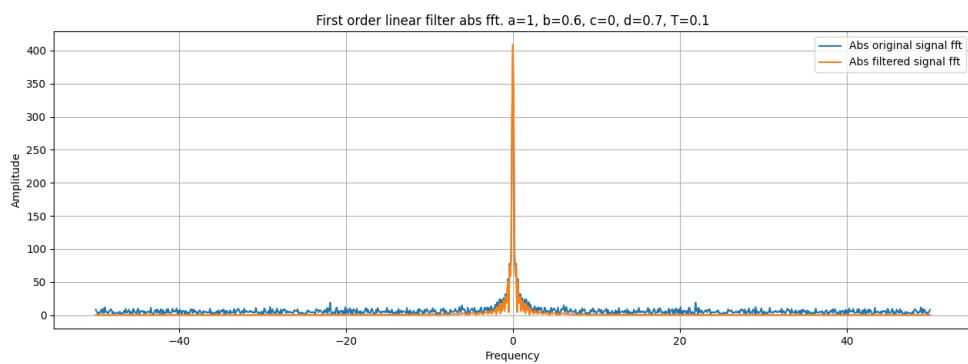


Рис. 14: График модулей Фурье-образа исходного и фильтрованного сигналов (2).

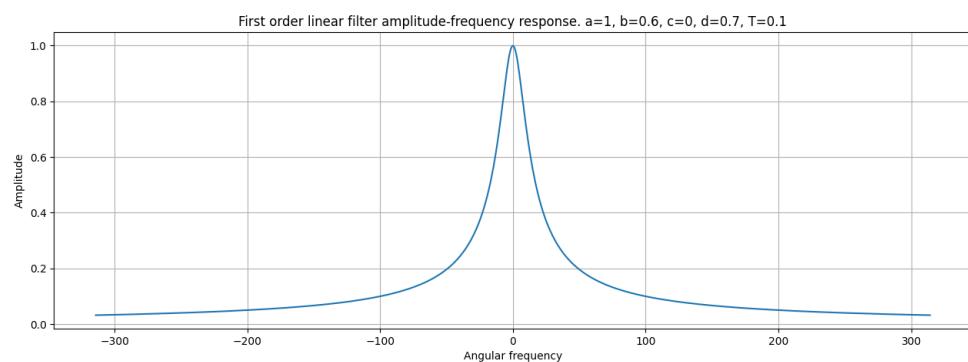


Рис. 15: График амплитудно-частотной характеристики фильтра (2).

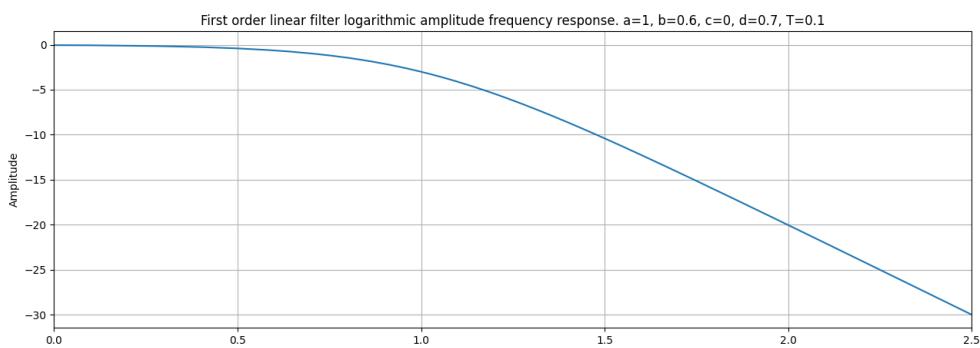


Рис. 16: График логарифмической амплитудно-частотной характеристики фильтра (2).

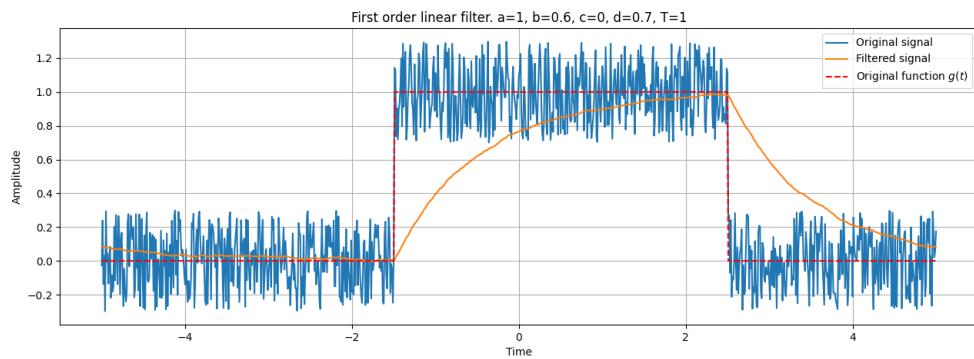


Рис. 17: График исходного и фильтрованного сигналов (3).

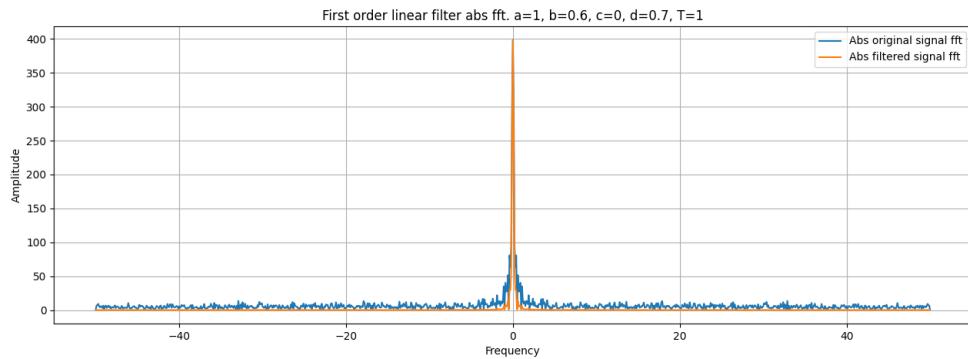


Рис. 18: График модулей Фурье-образа исходного и фильтрованного сигналов (3).

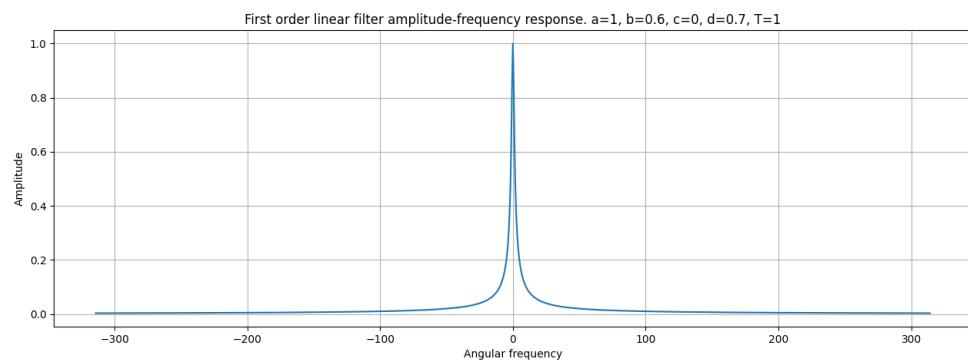


Рис. 19: График амплитудно-частотной характеристики фильтра (3).

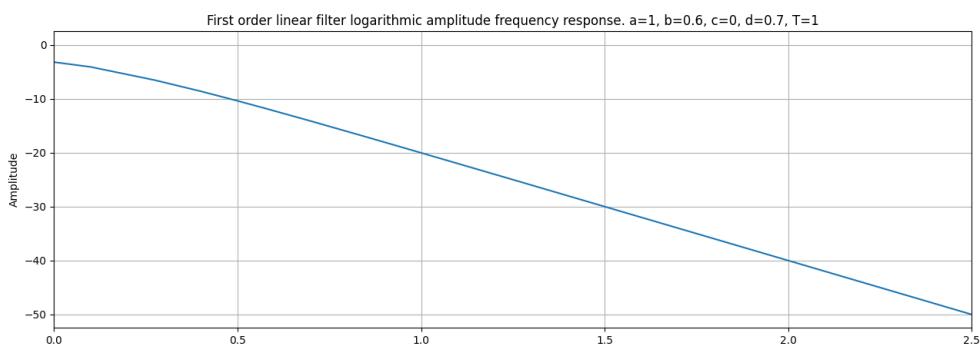


Рис. 20: График логарифмической амплитудно-частотной характеристики фильтра (3).

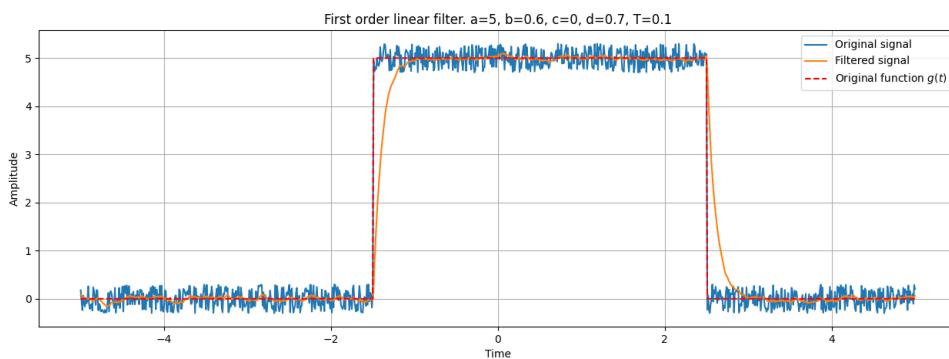


Рис. 21: График исходного и фильтрованного сигналов (4).

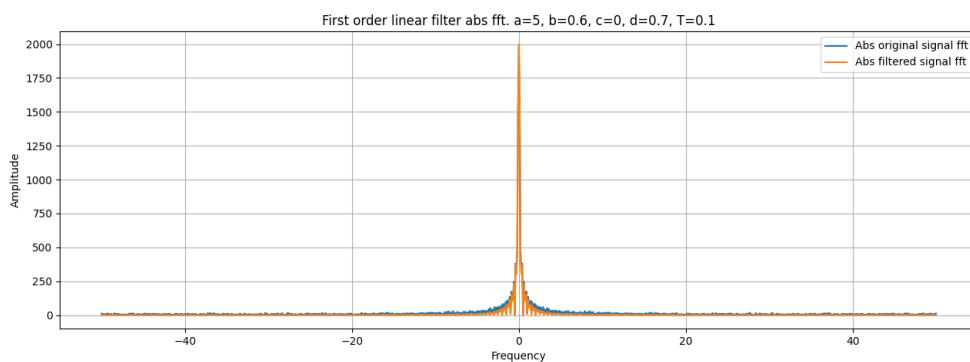


Рис. 22: График модулей Фурье-образа исходного и фильтрованного сигналов (4).

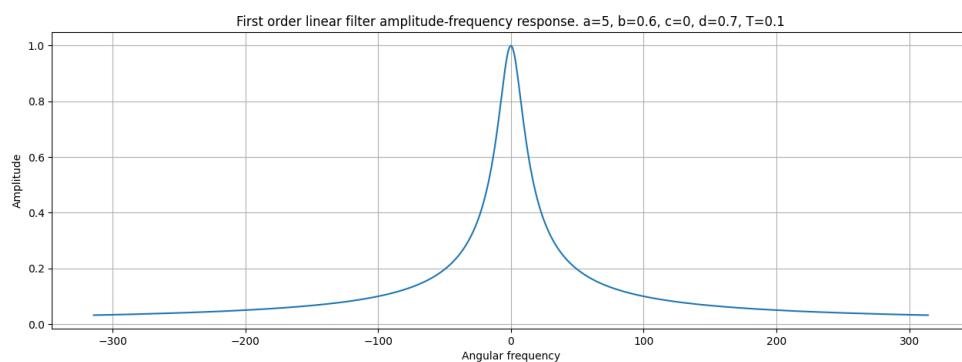


Рис. 23: График амплитудно-частотной характеристики фильтра (4).

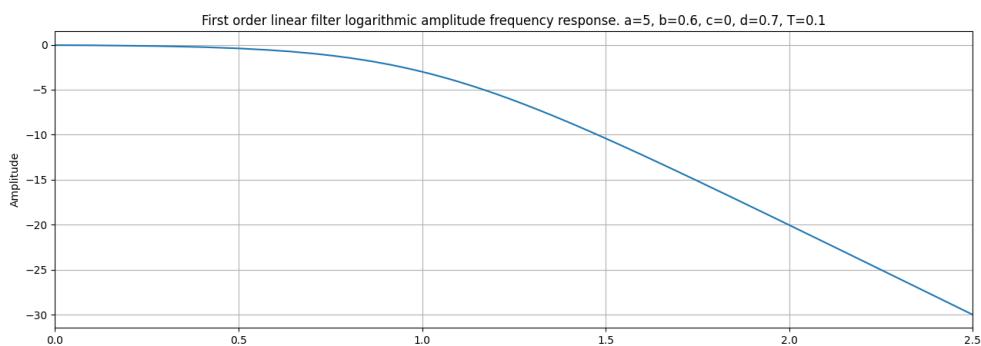


Рис. 24: График логарифмической амплитудно-частотной характеристики фильтра (4).

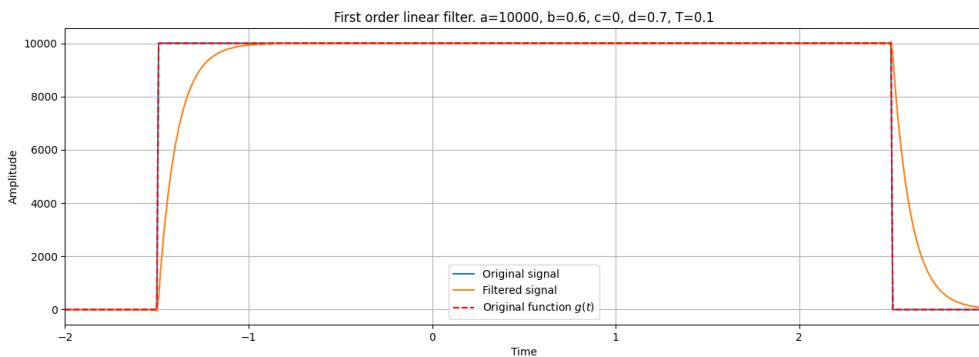


Рис. 25: Более детальное исследование параметра a (Исходный и фильтрованный сигнал).

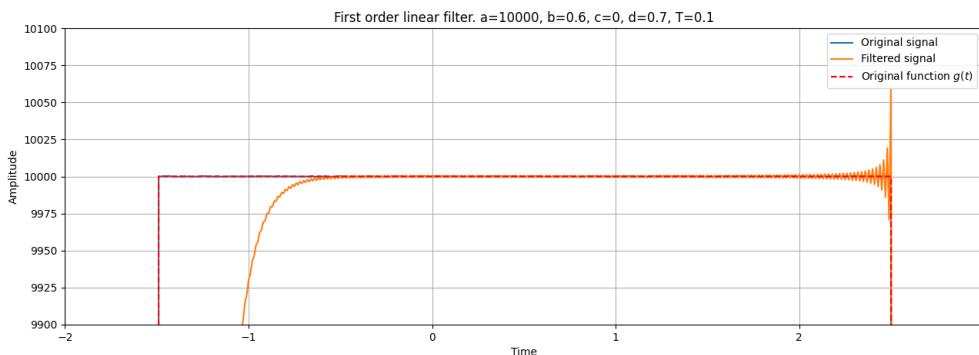


Рис. 26: Более детальное исследование параметра a (Поднятая часть сигналов в приближении).

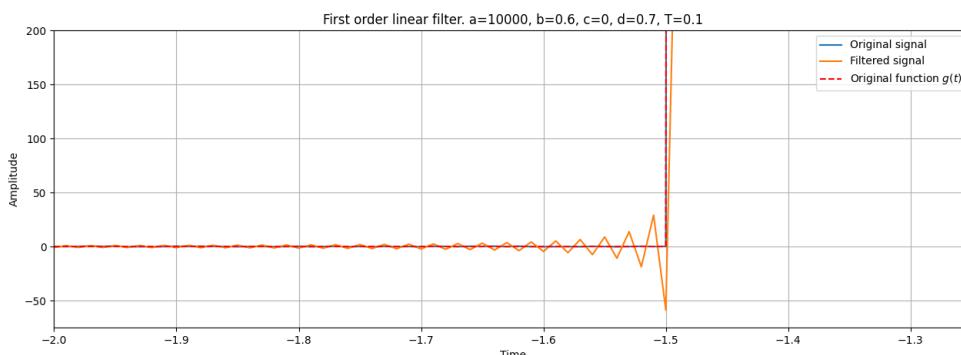


Рис. 27: Более детальное исследование параметра a (Левый разрыв сигналов).

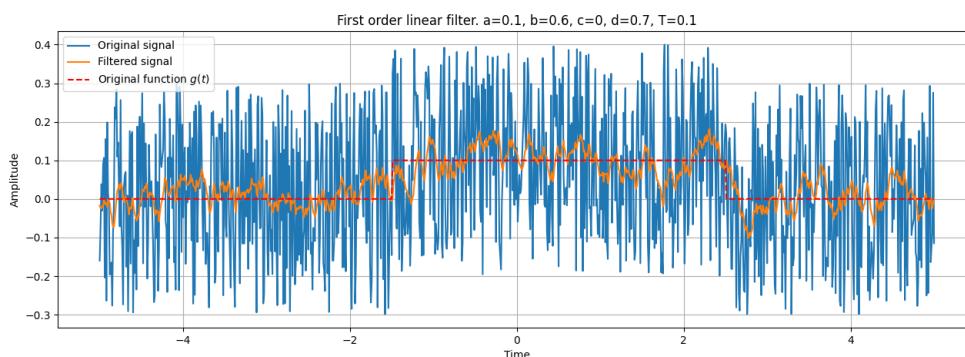


Рис. 28: Более детальное исследование параметра a .

Исходя из графиков можно сделать вывод, что при маленьких и больших значениях постоянной времени T фильтрация хуже, чем при средних значениях. Большие значения приводят к излишней фильтрации сигнала – результирующий сигнал хоть и хорошо сглажен, но приобрел лишний наклон (см. рис. 17). На рис. 18 видно, что в некотором небольшом диапазоне нуля большинство важных частот (имеющих большие амплитуды по сравнению с другими) урезалось фильтром. Еще лучше это видно на графике АЧХ – частота среза по оси абсцисс находится близко к оси ординат, что означает, что фильтр оставил немного амплитуд, соответствующих нижним частотам, нетронутыми, а ниже частоты среза график быстро убывает – происходит резкое ослабление амплитуд соответствующих частот (см. рис. 19). Маленькие значения приводят к недостаточной фильтрации сигнала – хотя фильтрованный сигнал и выглядит лучше исходного, осталось много белого шума (см. рис. 9). На графике модулей Фурье-образов большинство менее значимых частот было оставлено (см. рис. 10). На графике АЧХ рисунка 11 хорошо видно, что частота среза находится примерно в точке $(\omega_0 = 100, A = 1 \div \sqrt{2})$, и, ослабление амплитуд ниже такой частоты среза несильное (график медленно убывает). На графике ЛАЧХ частота среза находится примерно в точке $(2, -3)$ (см. рис. 12). На графиках ЛАЧХ (см. рис. 12, 16, 20) при последовательном увеличении значения T с каждым разом частота среза приближается к нижним частотам, уменьшаясь в 10 раз при переводе из логарифмической шкалы частот (изменение на 1 на логарифмической шкале частот соответствует изменению частоты в 10 раз). Примерно это мы и видим на графиках АЧХ – рис. 11 $\omega_0 \approx 100$, рис. 15 $\omega_0 \approx 10$, рис. 19 $\omega_0 \approx 1$. Хорошо фильтрация получилась на рис. 13 – большая часть белого шума пропала, график не сильно отклонился и неплохо повторяет функцию $g(t)$.

Рассматривая рисунки 13, 21 и 25 кажется, что влияние параметра a на эффективность фильтрации заключается в ее сглаживании – чем больше значение a , тем более гладким становится фильтрованный сигнал. Однако по рис. 26 и 27 можно увидеть, что при большом значении a усиливается эффект Гиббса в местах разрыва сигнала, при этом чем ближе сигнал к месту разрыва, тем постепенно больше становятся амплитуды фильтрованного сигнала – фильтрация явно ухудшается. Также при слишком маленькому значении параметра a фильтрованный сигнал выглядит хуже при том же значении параметра T (см. рис. 28, 13). Следовательно, параметр a так же как и параметр T следует выбирать не маленьким и не большим.

2.2 Специальный фильтр.

Положим $b = 0$. Рассмотрим линейный фильтр вида

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

при этом постараемся подобрать такие $T_1, T_2, T_3 > 0$, чтобы они по возможности хорошо убирали синусоидальную составляющую помехи и не сильно искажали полезный сигнал. Также рассмотрим несколько значений параметра d и найдем подходящие T_1, T_2, T_3 для каждого случая. После построим сравнительные графики и проанализируем влияние параметра c на эффективность фильтрации.

Далее расположены сравнительные графики исходного и фильтрованного сигналов, графики модулей их Фурье-образов и АЧХ и ЛАЧХ фильтра. Синим цветом отрисованы функции, связанные с исходным сигналом, оранжевым – с фильтрованным. Также на графике во временной области отображена исходная функция $g(t)$.

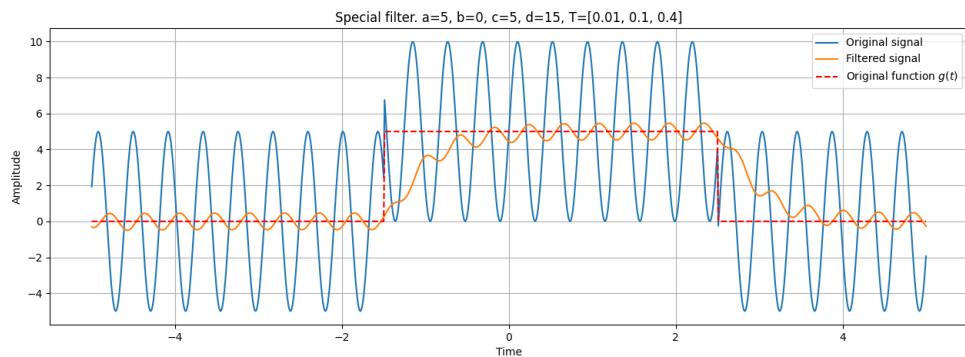


Рис. 29: График исходного и фильтрованного сигналов (1).

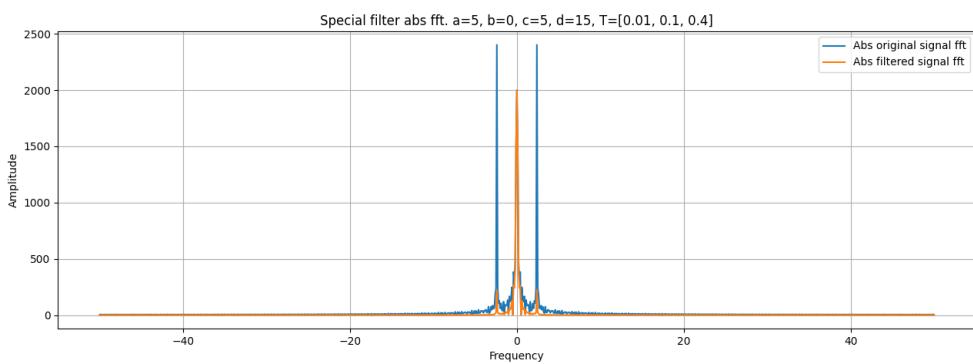


Рис. 30: График модулей Фурье-образа исходного и фильтрованного сигналов (1).

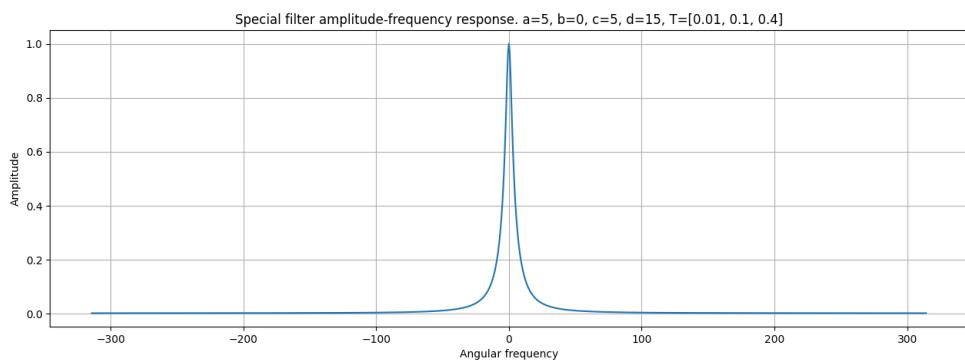


Рис. 31: График амплитудно-частотной характеристики фильтра (1).

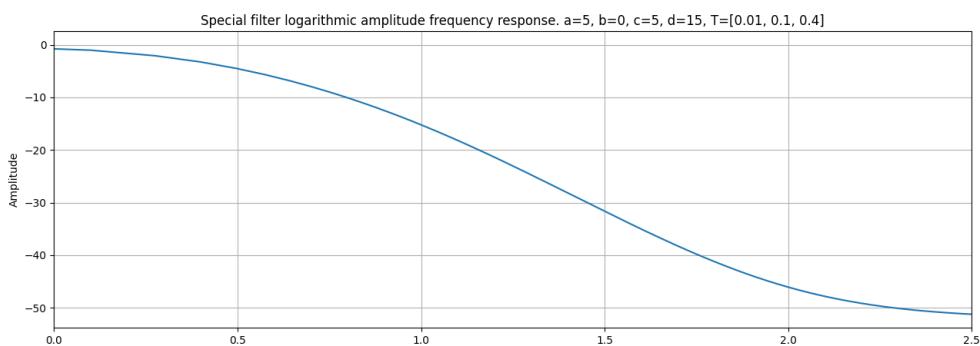


Рис. 32: График логарифмической амплитудно-частотной характеристики фильтра (1).

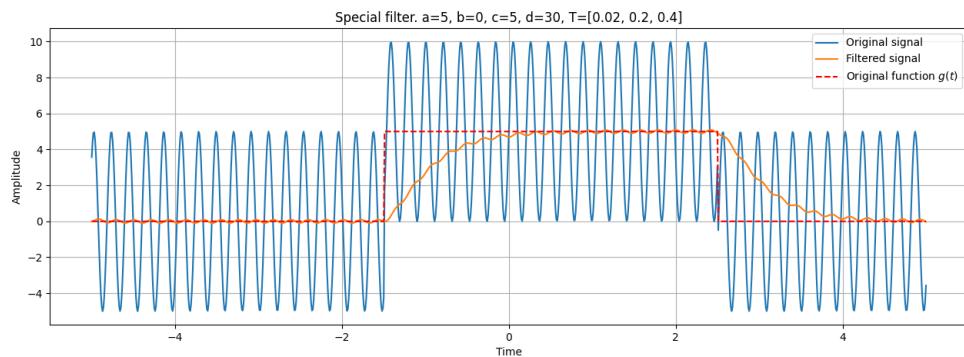


Рис. 33: График исходного и фильтрованного сигналов (2).

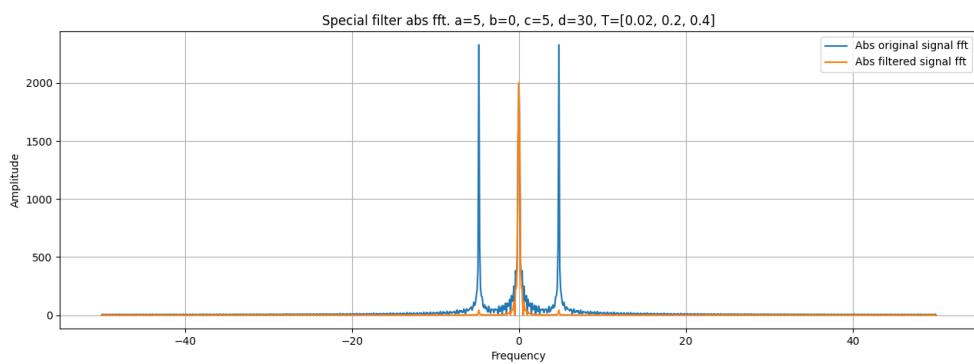


Рис. 34: График модулей Фурье-образа исходного и фильтрованного сигналов (2).

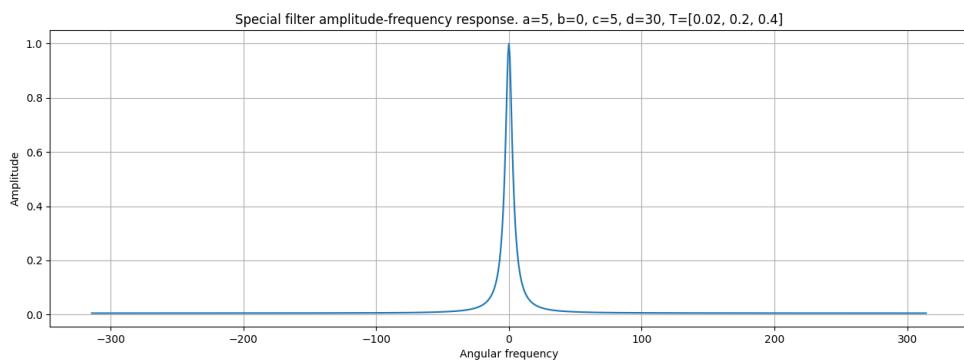


Рис. 35: График амплитудно-частотной характеристики фильтра (2).

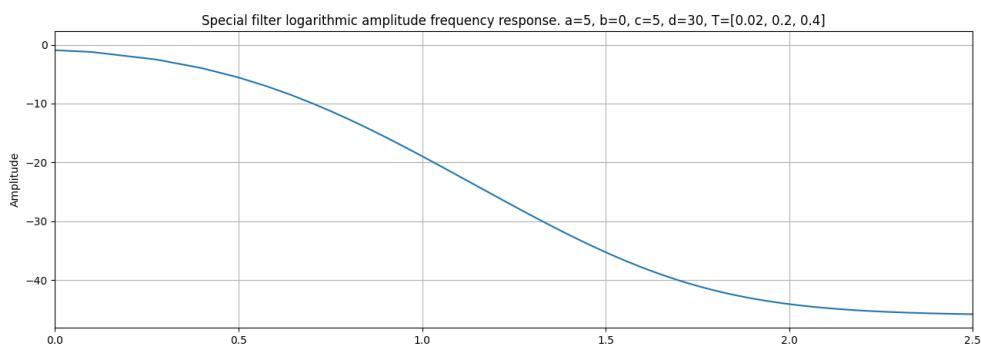


Рис. 36: График логарифмической амплитудно-частотной характеристики фильтра (2).

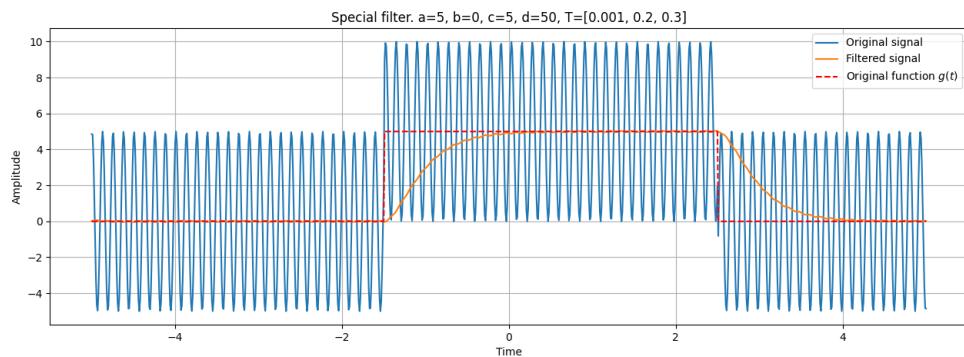


Рис. 37: График исходного и фильтрованного сигналов (3).

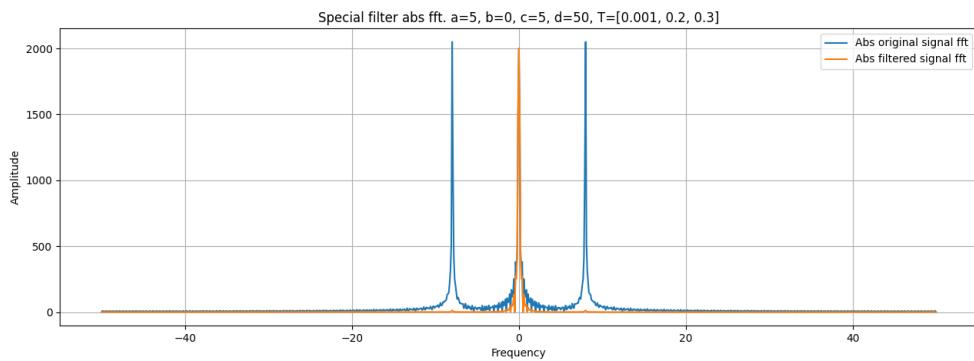


Рис. 38: График модулей Фурье-образа исходного и фильтрованного сигналов (3).

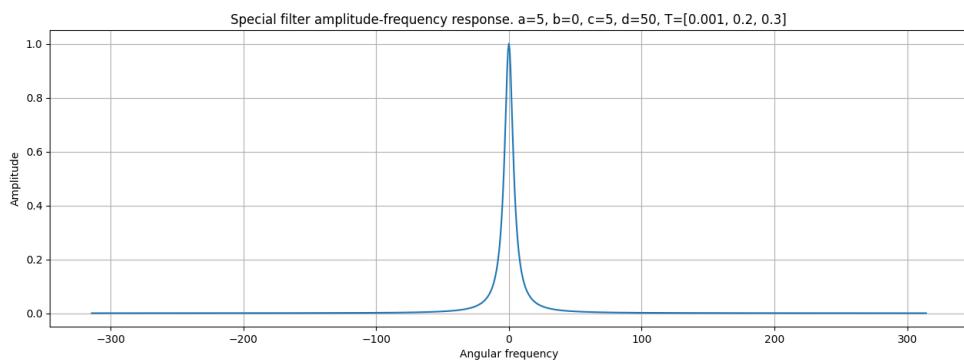


Рис. 39: График амплитудно-частотной характеристики фильтра (3).

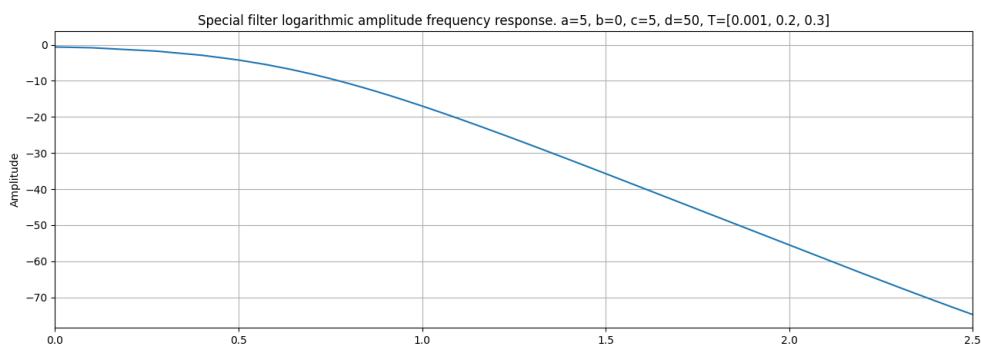


Рис. 40: График логарифмической амплитудно-частотной характеристики фильтра (3).

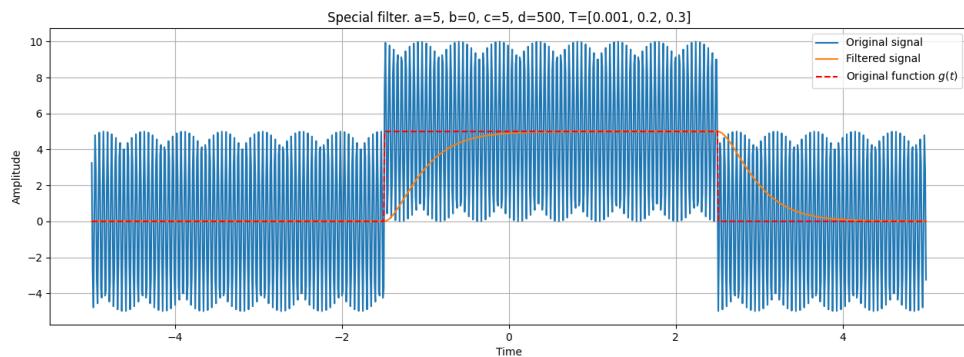


Рис. 41: График исходного и фильтрованного сигналов (4).

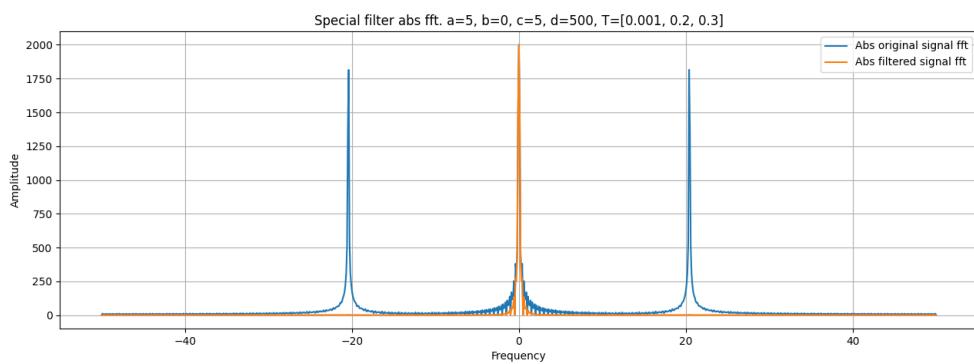


Рис. 42: График модулей Фурье-образа исходного и фильтрованного сигналов (4).

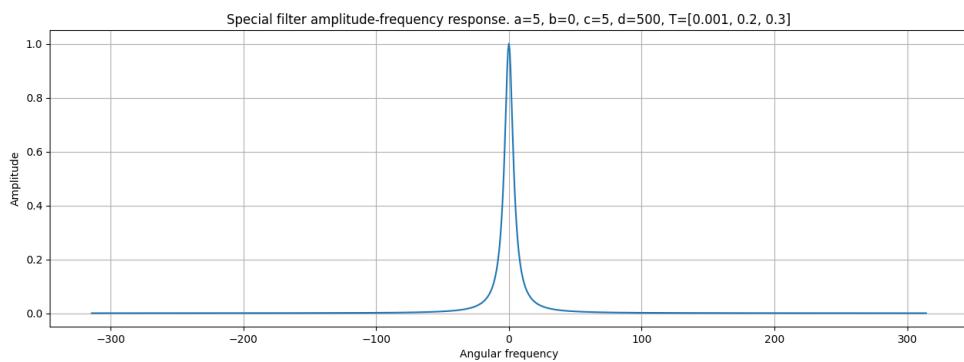


Рис. 43: График амплитудно-частотной характеристики фильтра (4).

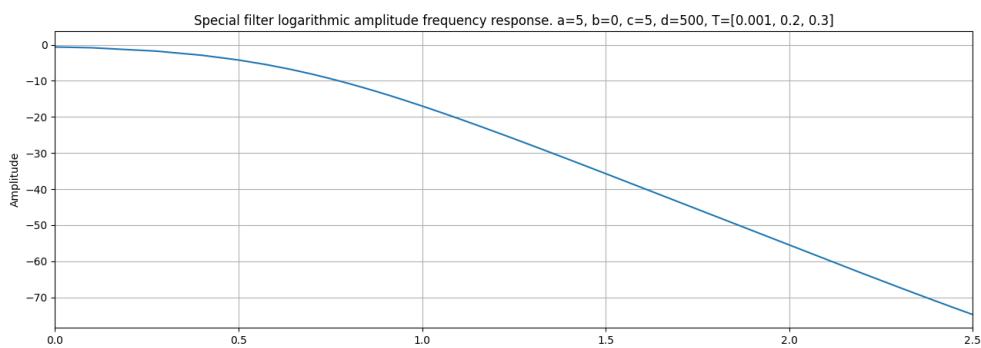


Рис. 44: График логарифмической амплитудно-частотной характеристики фильтра (4).

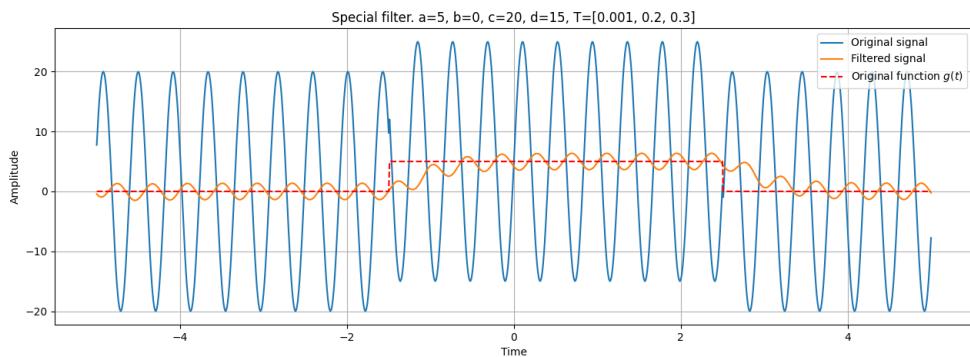


Рис. 45: График исходного и фильтрованного сигналов (5).

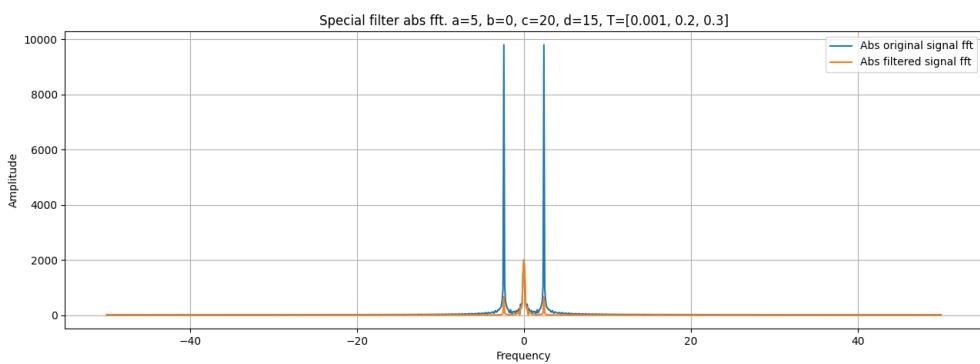


Рис. 46: График модулей Фурье-образа исходного и фильтрованного сигналов (5).

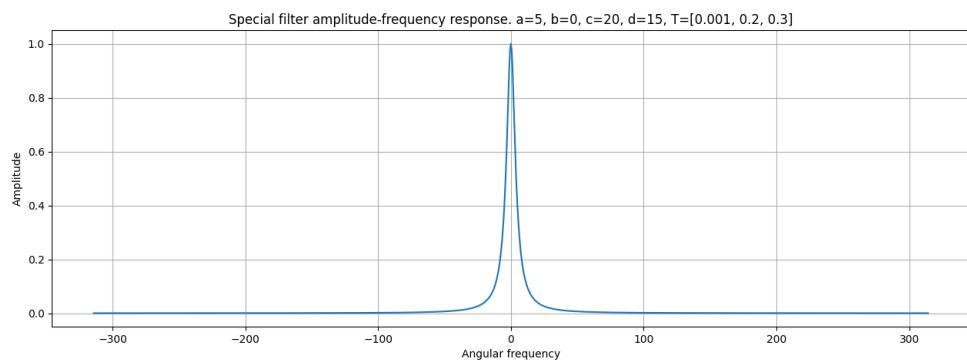


Рис. 47: График амплитудно-частотной характеристики фильтра (5).

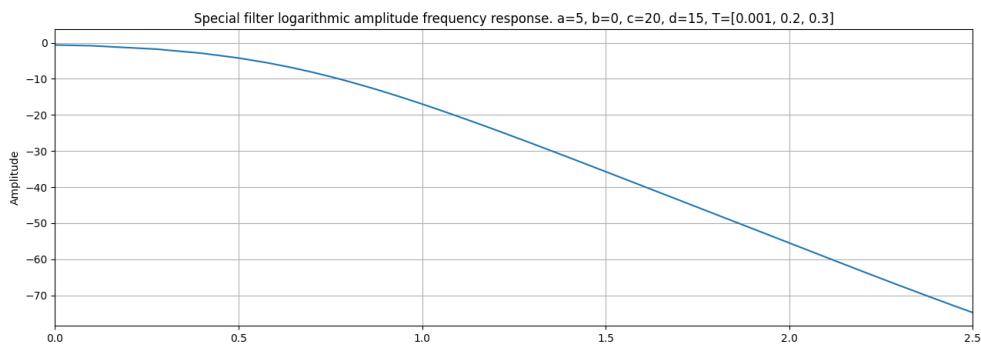


Рис. 48: График логарифмической амплитудно-частотной характеристики фильтра (5).

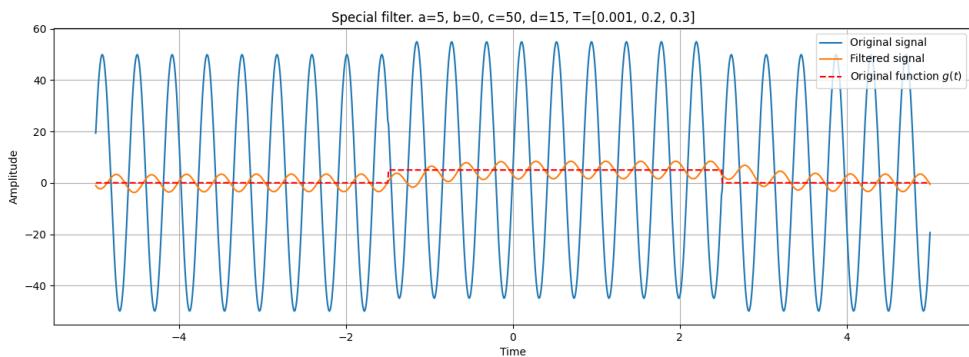


Рис. 49: График исходного и фильтрованного сигналов (6).

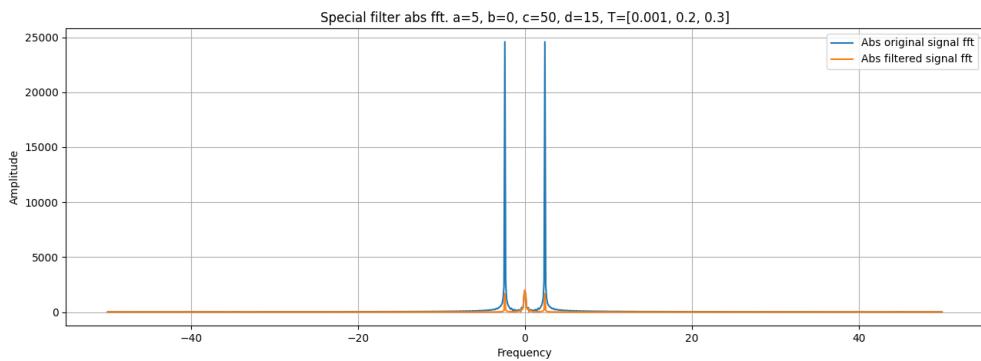


Рис. 50: График модулей Фурье-образа исходного и фильтрованного сигналов (6).

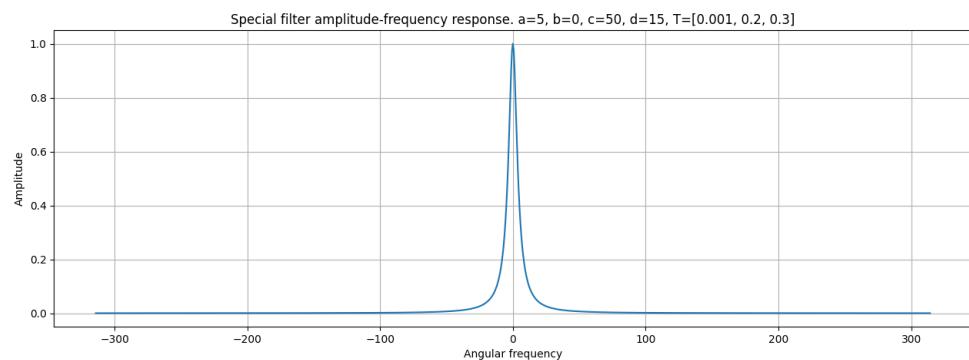


Рис. 51: График амплитудно-частотной характеристики фильтра (6).

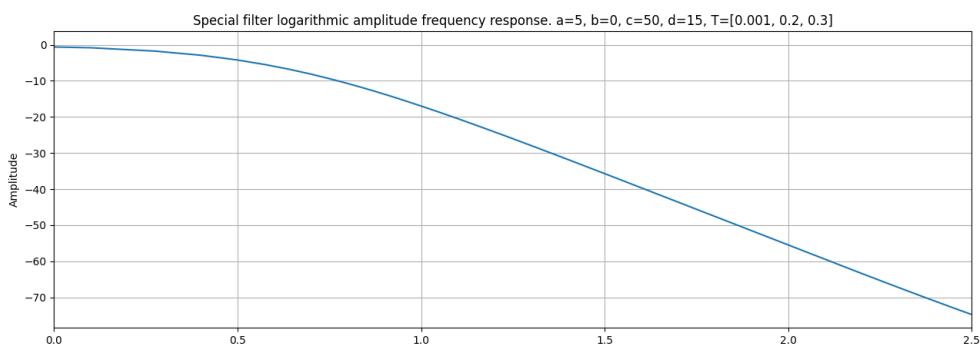


Рис. 52: График логарифмической амплитудно-частотной характеристики фильтра (6).

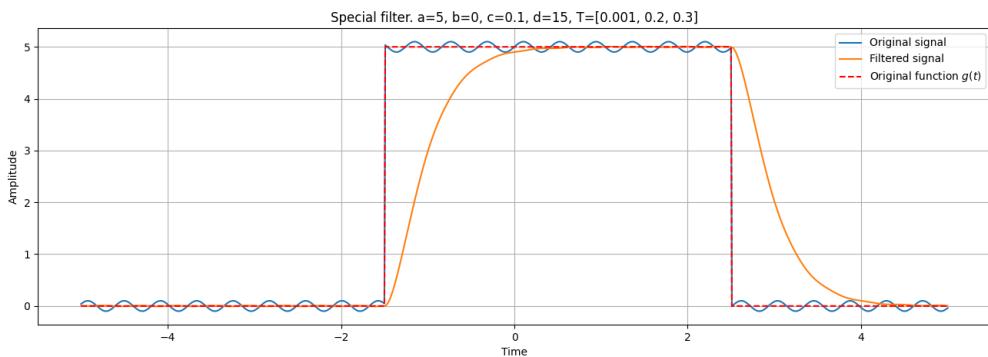


Рис. 53: График исходного и фильтрованного сигналов (7).

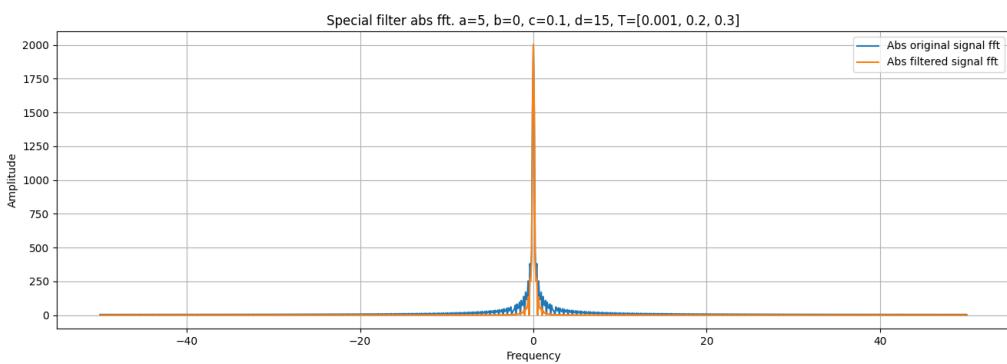


Рис. 54: График модулей Фурье-образа исходного и фильтрованного сигналов (7).

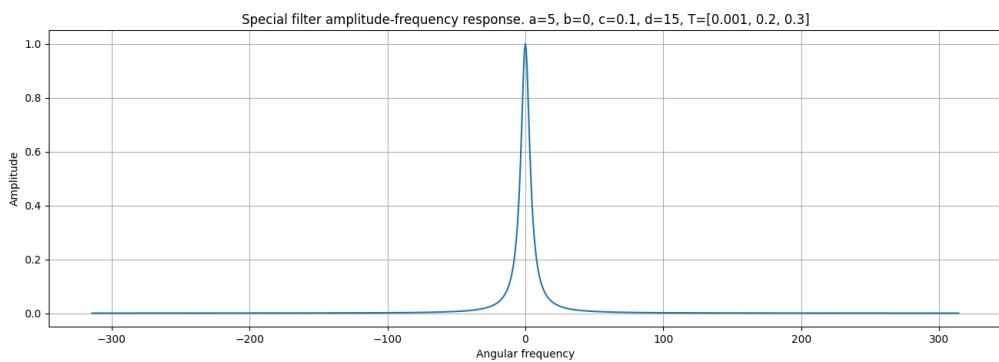


Рис. 55: График амплитудно-частотной характеристики фильтра (7).

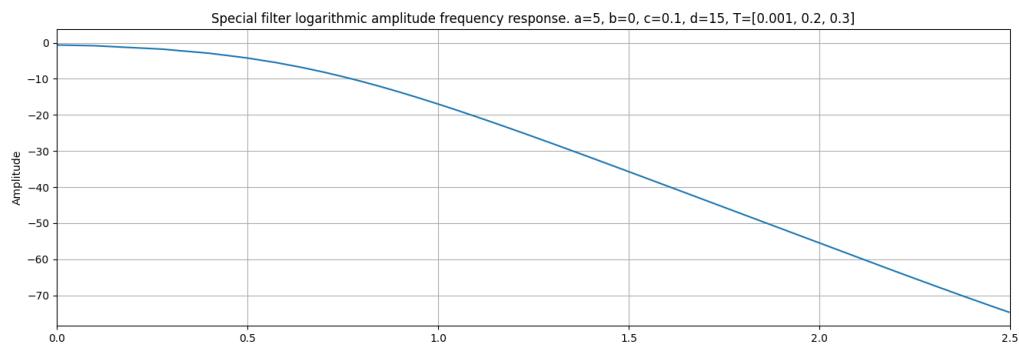


Рис. 56: График логарифмической амплитудно-частотной характеристики фильтра (7).

Исходя из результатов графиков можно сделать вывод, что при увеличении значения параметра d фильтрация улучшается. Основная информация об исходном сигнале хранится на нижних частотах, что мы видим на любом графике модуля Фурье-образа. Чем меньше значение параметра d , тем ближе к нижним частотам расположены гармонический шум, проявляющийся в виде двух высоких возрастаниях амплитуд (см. рис. 30). Из-за этого при маленьком значении параметра d сложно подобрать такие T_1, T_2, T_3 , чтобы минимизировать влияние синусоидального шума и при этом оставить значимую часть сигнала нетронутой, так как сложнее найти ту частоту среза, выше которой фильтр оставит все важные амплитуды и ниже которой полностью уберет амплитуды гармонического шума. В то время как при большом значении параметра d , при условии отсутствия влияния параметра b , выбор коэффициентов, при котором фильтрация будет успешна, более разнообразен (см. рис. 42). В этом случае выбрать частоту среза проще вследствие расширения диапазона ее выбора. В ходе выполнения задания мне удалось получить наилучшую фильтрацию при соотношении $T_1 \ll T_2 \approx T_3$.

Параметр c влияет на амплитуду каждой волны гармонического шума. При увеличении значения параметра c амплитуды увеличиваются как во временном пространстве (см. рис. 29, 45, 49), так и в частотном (см. рис. 30, 46, 50), а при уменьшении значения – уменьшаются. Данное увеличение амплитуд ухудшает эффективность фильтрации. Придется подобрать такие параметры T_1, T_2, T_3 , чтобы амплитуды гармонического шума, сильно большие значимых амплитуд исходного сигнала, ушли под действием фильтра. Однако при выборе такой частоты среза высока вероятность, что фильтр не только не сможет достаточно ослабить амплитуды синусоидального шума, но и затронет нужные для хорошей фильтрации. Данную проблему хорошо видно на рисунке 50. При маленьких значениях параметра c фильтрация получается наилучшей (см. рис. 53).

2.3 Используемые программы.

В ходе выполнения этого задания были использованы некоторые программы, приведенные ранее. В числе новых есть программа, реализующая линейные фильтры, рассмотренные в задании.

```

1 def W_1f(w, T):
2     p = 1j * w
3     return 1 / (T * p + 1)
4
5 def W_2f(w, T_1, T_2, T_3):
6     p = 1j * w
7     return (T_1 * p + 1) ** 2 / ((T_2 * p + 1) * (T_3 * p + 1))

```

Листинг 4: Программа, содержащая реализацию линейных фильтров.

Добавились функции, необходимые для нахождения АЧХ, ЛАЧХ и пропускания сигнала через фильтр. В функции пропуска сигнала через фильтр можно задать параметр `shift`, он будет необходим позже.

```

1 def fft_flt(u, W, shift=True):
2     U = np.fft.fft(u)
3     maybe_shifted_U = U
4     if shift:
5         maybe_shifted_U = np.fft.fftshift(U)
6
7     flt_U = W * maybe_shifted_U
8     maybe_shifted_flt_U = flt_U
9     if shift:

```

```

10         maybe_shifted_flt_U = np.fft.ifftshift(flt_U)
11
12     flt_u = np.fft.ifft(maybe_shifted_flt_U)
13     return flt_u, maybe_shifted_flt_U, U
14
15 def get_AFR(W):
16     return np.abs(W)
17
18 def get_LAFLR(W):
19     return 20 * np.log10(get_AFR(W))

```

Листинг 5: Программа для нахождения АЧХ, ЛАЧХ и пропуска сигнала через фильтр.

Есть вспомогательная программа для создания исходной функции и исходного сигнала с шумом.

```

1 def g_func(t, t_1, t_2, a):
2     if (t_1 <= t <= t_2): return a
3     return 0
4
5 def get_g_f(t: list, t_1, t_2, a):
6     g_f = []
7     for k in range(len(t)):
8         g_f.append(g_func(t[k], t_1, t_2, a))
9     return g_f
10
11 def get_u(g_fs: list, t: list, b, c, d):
12     return np.array(g_fs) + b * (np.random.rand(len(t)) - 0.5) + c * np.
13         sin(d * t)

```

Листинг 6: Программа для создания исходной функции и сигнала с шумом.

Алгоритм, использующий приведенные ранее программы, расположен ниже.

```

1 import numpy as np
2
3 import build_func as bf
4 import lin_filters as lf
5 import fourier_math as fm
6 import helper as hr
7
8 def perform(t, v, w, u, a, b, c, d, T, W, W_LOG, g_f=None,
9             name='Filter', x10=None, x101=None,
10            x11=None, x12=None, x13=None,
11            x14=None, x15=None, x16=None):
12     flt_u, flt_U, U = fm.fft_flt(u, W)
13     AFR, LAFLR = fm.get_AFR(W), fm.get_LAFLR(W_LOG)
14
15     flist1 = [u, flt_u.real]
16     flist2 = [abs(np.fft.fftshift(U)), abs(np.fft.fftshift(flt_U))]
17     llist1 = ['Original signal', 'Filtered signal']
18     llist2 = ['Abs original signal fft', 'Abs filtered signal fft']
19     lslist = ['-', '-']
20     clist = [None, None]
21     if g_f is not None:
22         flist1.append(g_f)
23         llist1.append(r'Original function $g(t)$')
24         lslist.append('--')
25         clist.append('r')
26
27     bf.build_fs(t, y=flist1, labels=llist1, colors=clist, ls=lslist,

```

```

28         ttl=f'{name}. a={a}, b={b}, c={c}, d={d}, T={T}', 
29         xlab='Time', ylab='Amplitude', legend=True, x11=x10, x12=x101)
30     bf.build_fs(v, y=flist2, labels=llist2,
31                 ttl=f'{name} abs fft. a={a}, b={b}, c={c}, d={d}, T={T}', 
32                 xlab='Frequency', ylab='Amplitude', legend=True, x11=x11, 
33                 x12=x12)
34     bf.build_f(w, y=AFR,
35                 ttl=f'{name} amplitude-frequency response. a={a}, b={b}, c={c}
36                 }, d={d}, T={T}', xlab='Angular frequency',
37                 ylab='Amplitude', x11=x13, x12=x14)
38     bf.build_f(np.log10(w[w > 0]), y=LAFR,
39                 ttl=f'{name} logarithmic amplitude frequency response. a={a}
40                 }, b={b}, c={c}, d={d}, T={T}', 
41                 ylab='Amplitude', x11=x15, x12=x16)
42
43
44     T, dt = 10, 0.01
45     V, dv = 1 / dt, 1 / T
46     t_1, t_2 = -1.5, 2.5
47
48
49     v_to_w_coeff = 2 * np.pi
50     t = np.arange(-T / 2, T / 2 + dt, dt)
51     v = np.arange(-V / 2, V / 2 + dv, dv)
52     w = v_to_w_coeff * v
53
54     v_log = np.arange(0, V / 2 + dv, dv)
55     w_log = v_to_w_coeff * v_log
56
57     perform_W_1, perform_W_2 = True, True
58
59     if perform_W_1:
60         a, b, c, d = 1, 0.6, 0, 0.7
61
62         g_fun = hr.get_g_f(t, t_1, t_2, a)
63         u = hr.get_u(g_fun, t, b, c, d)
64
65         T_0 = 0.1
66         W_1 = lf.W_1f(w, T_0)
67         W_1_LOG = lf.W_1f(w_log, T_0)
68
69         perform(t, v, w, u, a, b, c, d, T_0, W_1, W_1_LOG, g_f=g_fun,
70                 name='First order linear filter', x15=0, x16=2.5)
71
72     if perform_W_2:
73         a, b, c, d = 5, 0, 5, 15
74
75         g_fun = hr.get_g_f(t, t_1, t_2, a)
76         u = hr.get_u(g_fun, t, b, c, d)
77
78         T_1, T_2, T_3 = 0.001, 0.2, 0.3
79         W_2 = lf.W_2f(w, T_1, T_2, T_3)
80         W_2_LOG = lf.W_2f(w_log, T_1, T_2, T_3)
81
82         perform(t, v, w, u, a, b, c, d, [T_1, T_2, T_3], W_2, W_2_LOG,
83                 g_f=g_fun, name='Special filter', x15=0, x16=2.5)

```

Листинг 7: Алгоритм, использующий приведенные ранее программы.

3 Задание 3. Сглаживание биржевых данных.

В данном задании представим, что мы разрабатываем инвестиционное приложение, в котором должна присутствовать функция представления сглаженных графиков котировок акций. При этом степень сглаживания должна зависеть рассматриваемого пользователем временного периода.

Скачаем с [данного сайта](#) файл с данными о стоимости акций Сбербанка в период с 21 марта 2018 по 21 марта 2019. Периодичность выберем один день. Загрузим данные в Python и применим к данным линейный фильтр первого порядка. Последовательно возьмем значения постоянной времени T равной 1 день, 1 неделю, 1 месяц, 3 месяца и 1 год. Представим, что во всех месяцах ровно 30 дней, значит за три месяца пройдет 90 дней. Построим красивые сравнительные графики.

Далее расположены результирующие графики. Синим цветом обозначены исходные данные, оранжевым – сглаженные. По оси абсцисс отложены понедельно даты в период за год, по оси ординат цена закрытия акции. В названии графика указаны выбранные параметры шага для частот и временной постоянной.



Рис. 57: График исходных и фильтрованных данных (1).



Рис. 58: График исходных и фильтрованных данных (1).

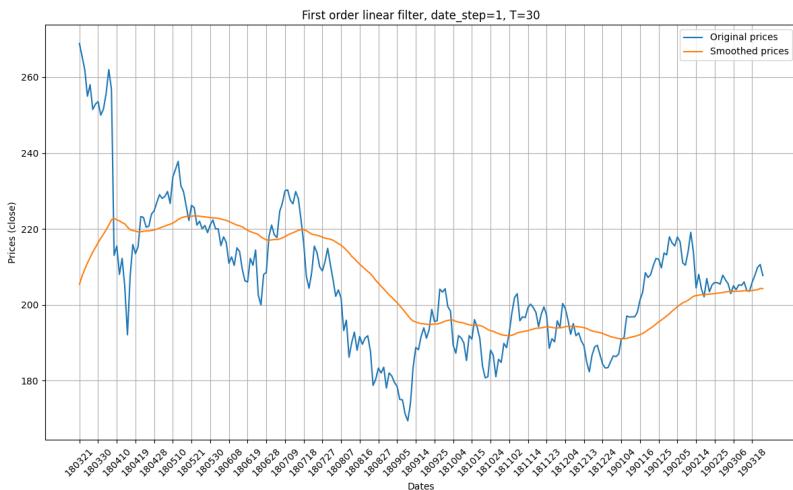


Рис. 59: График исходных и фильтрованных данных (1).



Рис. 60: График исходных и фильтрованных данных (1).

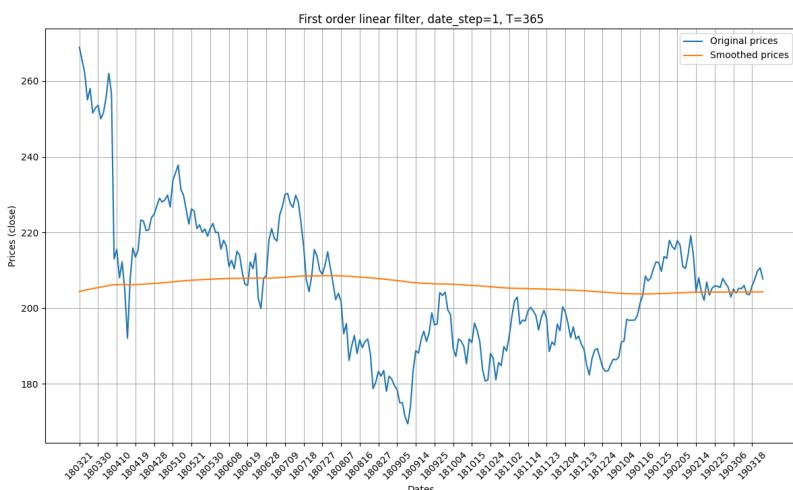


Рис. 61: График исходных и фильтрованных данных (1).



Рис. 62: График исходных и фильтрованных данных (2).



Рис. 63: График исходных и фильтрованных данных (2).



Рис. 64: График исходных и фильтрованных данных (2).



Рис. 65: График исходных и фильтрованных данных (2).

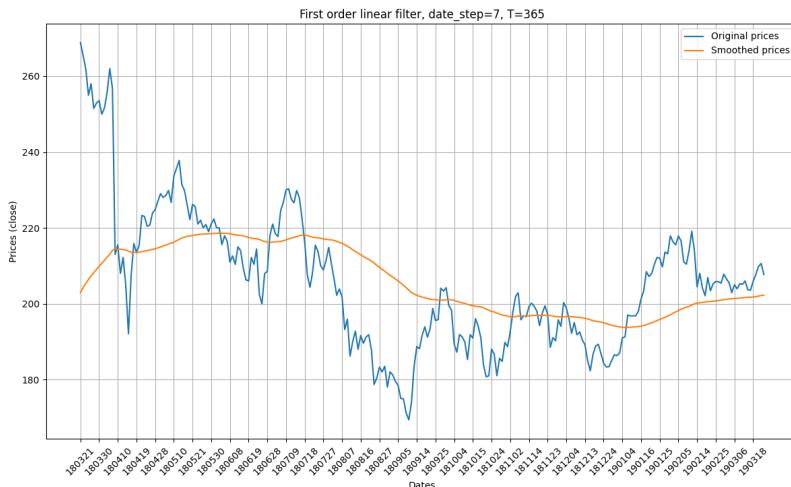


Рис. 66: График исходных и фильтрованных данных (2).

Как видим, при увеличении параметра T график сглаживается, то есть для более длительного временного периода сглаживание становится более склонным к усреднению данных (см. рис. 59, 60, 61, 66). Таким образом сохраняются долгосрочные тренды, а краткосрочные нет. При уменьшении параметра T логика наоборот (см. рис. 57, 58, 62, 63, 64).

Шаг времени для частот влияет на степень сглаживания графика при выборе временного периода. Маленький шаг соответствует более частым измерениям, сглаживание становится более точным, это может быть полезно для краткосрочных трендов. Большой шаг приводит к более грубому сглаживанию, но это может быть полезно для долгосрочных трендов.

3.1 Используемые программы.

Для выполнения данного задания была написана программа на языке Python. В ней использовались некоторые ранее написанные программы. Для нахождения частот была применена функция, реализованная в библиотеке numpy. В эту функцию передается шаг

по времени, в ходе работы были приведены случаи для шагов равных одному дню и одной неделе. Также при использовании моей функции передается параметр shift как ложь, так как из-за особенностей работы функции для поиска частот сдвиг для пропускания данных через фильтр не нужен.

```

1 import csv
2 import numpy as np
3
4 import build_func as bf
5 import lin_filters as lf
6 import fourier_math as fm
7
8 file = 'fm_lab4/data/SBER_180321_190321.csv'
9 find_date = 'DATE'
10 find_price = 'CLOSE'
11 dates = []
12 prices = []
13 with open(file) as csv_file:
14     csv_reader = csv.reader(csv_file, delimiter=';')
15     header = next(csv_reader)
16     date_i = 0
17     price_i = 0
18     for i in range(len(header)):
19         if find_date in header[i]:
20             date_i = i
21         if find_price in header[i]:
22             price_i = i
23
24     for row in csv_reader:
25         dates.append(row[date_i])
26         prices.append(float(row[price_i]))
27
28 date_step = 1
29 n = len(dates)
30 v = np.fft.fftfreq(n, d=date_step)
31 w = 2 * np.pi * v
32
33 T = 1
34 W = lf.W_1f(w, T)
35 flt_u, flt_U, U = fm.fft_flt(prices, W, shift=False)
36
37 bf.build_fs(dates, [prices, flt_u.real], ticks=range(0, len(dates), 7),
38               rot=45, xlab='Dates', ylab=f'Prices ({find_price.lower()})',
39               ttl=f'First order linear filter, date_step={date_step}, T={T}',
40               fz2=8,
41               legend=True, labels=['Original prices', 'Smoothed prices'])

```

Листинг 8: Программа для сглаживания биржевых данных.