



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЕТ  
по лабораторной работе №3

Жесткая фильтрация

Автор: Дженжеруха Кирилл

Учебная группа: Р3242

Номер ИСУ: 368103

Преподаватель: Перегудин Алексей Алексеевич

Санкт-Петербург  
2024

## **Содержание**

<b>Задание 1. Жесткие фильтры</b>	<b>3</b>
<b>Задание 2. Фильтрация звука</b>	<b>9</b>

*Математический анализ столъ же обширен,  
как и сама природа.  
Жан-Батист Жозеф Фурье*

Эта лабораторная работа посвящена методам жесткой фильтрации данных, содержащих некоторые искажения.

---

## Задание 1. Жесткие фильтры

Жесткая фильтрация подразумевает под собой выявление “дефектов” входящих данных с помощью преобразований Фурье и обнуление тех участков, которые создают помехи при распознавании сигнала.

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

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

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

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

Примем  $c = 0$  (иначе говоря, обнулим вклад синусоидального шума) и построим Фурье-образ зашумленного сигнала. Обозначив за  $\text{sl} = \nu_0$ , где  $\nu_0$  – величина частоты, используемая для обозначения границ интервала ненулевых частот. Построим графики модуля Фурье-образа зашумленного сигнала для различных значений параметра  $\text{sl}$ :

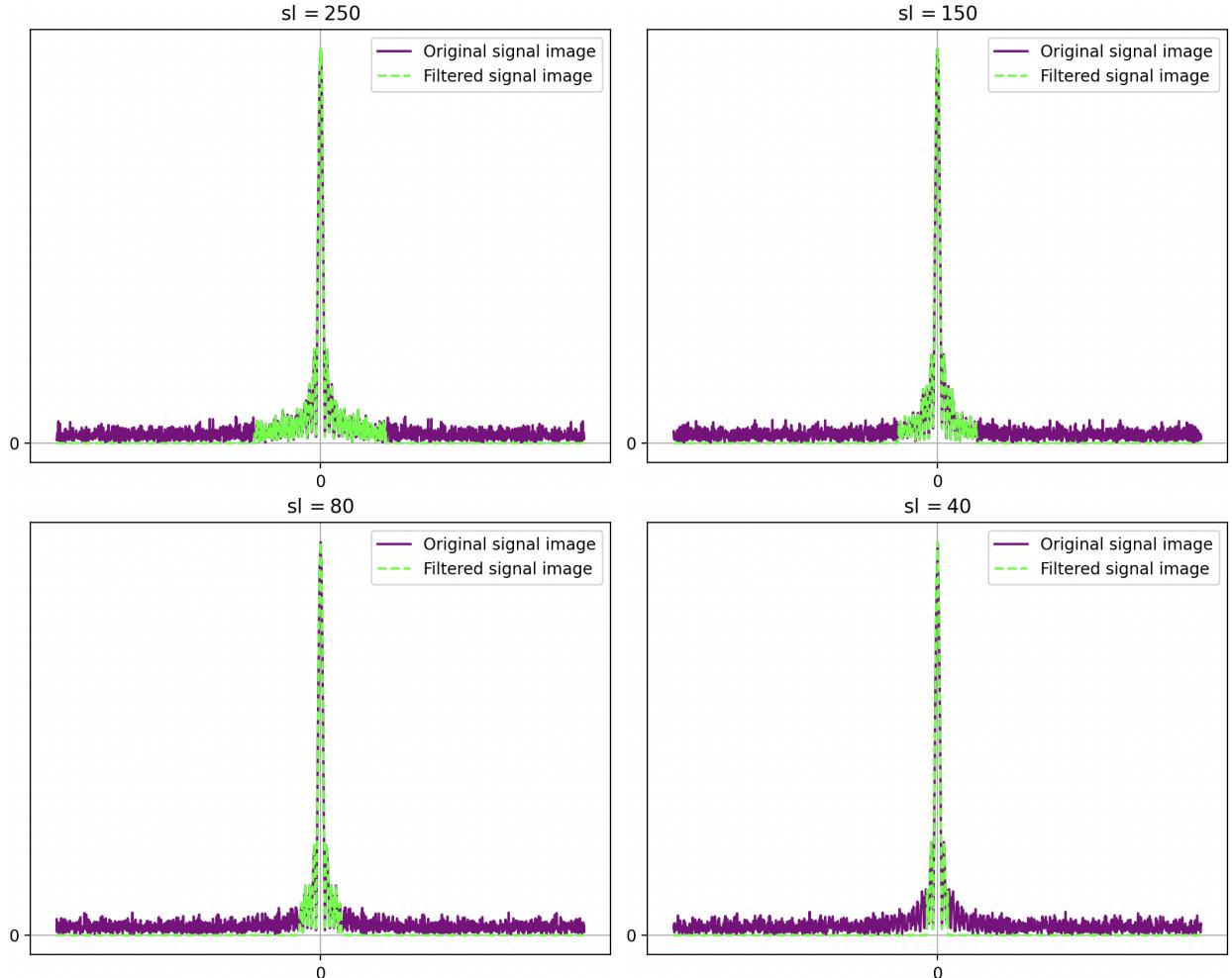
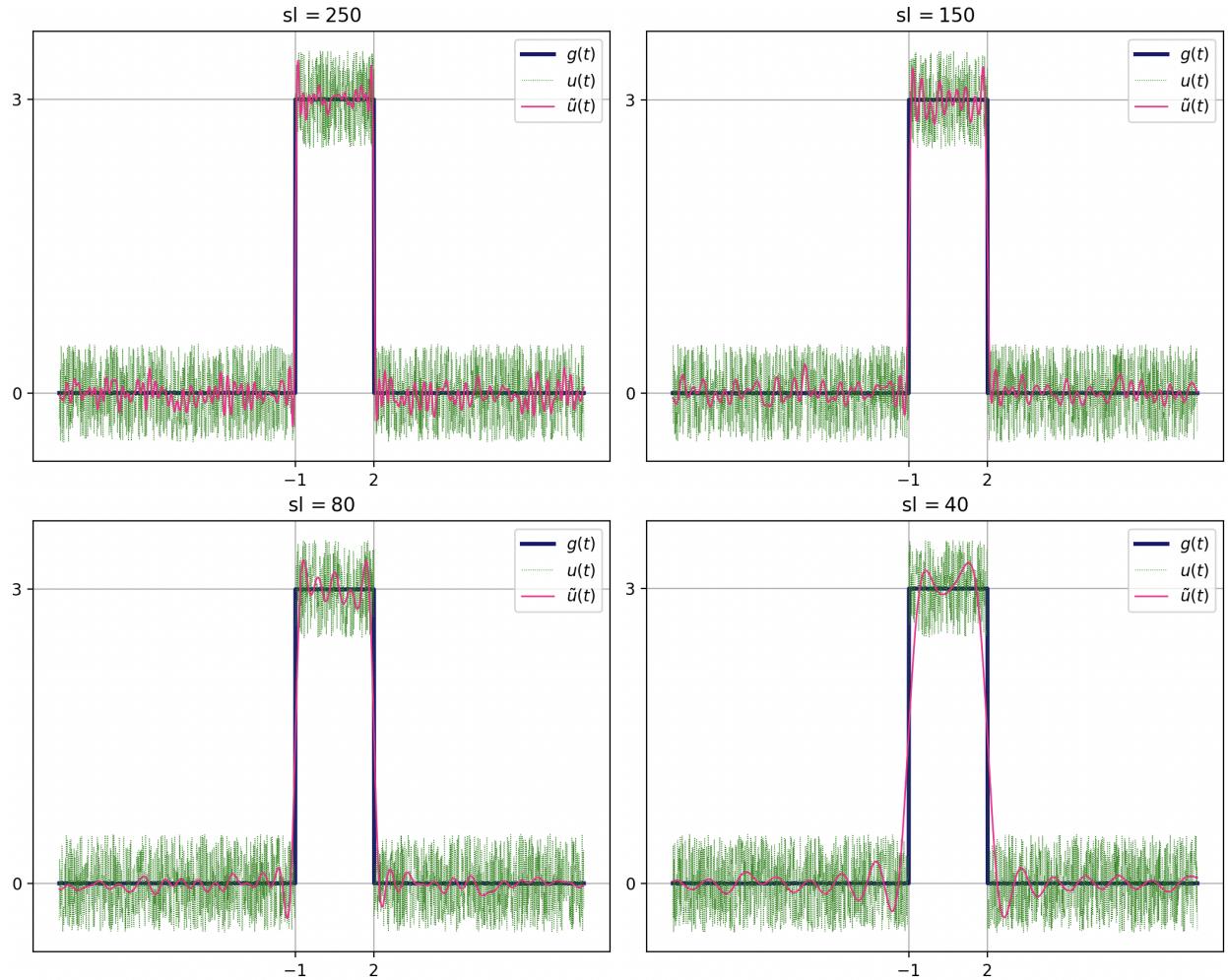


Рис. 1 – Графики модуля Фурье-образа зашумленного сигнала для различных значений  $\text{sl}$ .

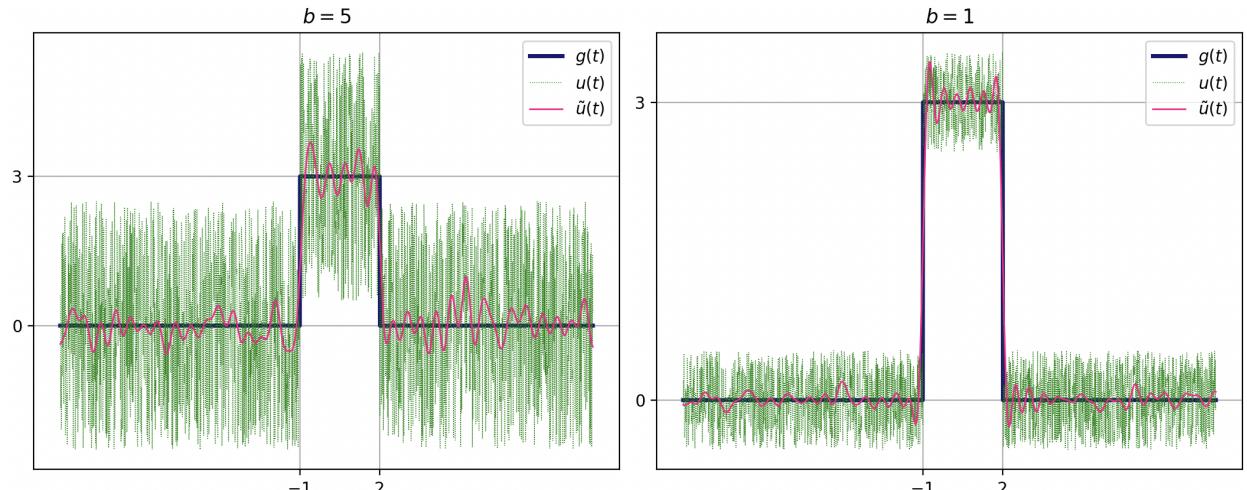
Увидим, что чем меньше интервал выбранных частот, тем, очевидно, меньше помех будет содержать сигнал на выходе. Выполним обратные преобразования для каждого из интервалов и построим сравнительные графики, каждый из которых содержит график исходной функции  $g(t)$ , зашумленного сигнала  $u(t)$  и сигнала после фильтрации  $\tilde{u}(t)$ :

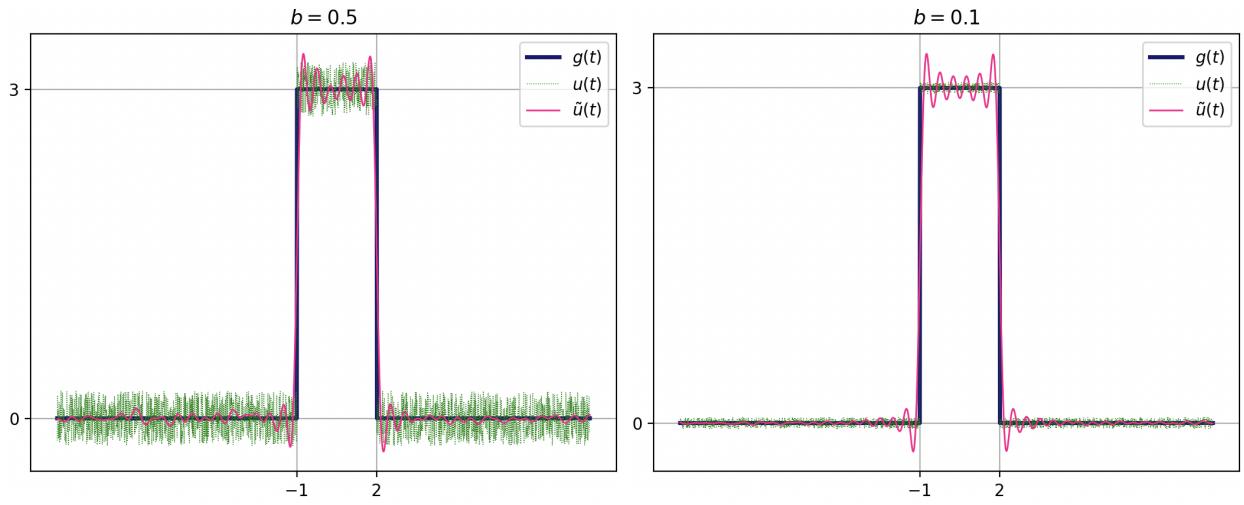


**Рис. 2** – Сравнительные графики после фильтрации сигнала для различных значений  $sl$ .

Как мы видим, сигнал действительно содержит меньше помех – это иллюстрируется амплитудой колебаний фильтрованного сигнала на различных промежутках. Но при уменьшении промежутка уменьшается и точность результирующего сигнала, происходит растяжение.

Кроме того, поведение фильтра зависит от значения параметра  $b$ , задаваемого вручную. Проиллюстрируем работу фильтра для разных значений коэффициента:





**Рис. 3** – Сравнительные графики после фильтрации сигнала для различных значений  $b$ .

Поймем и здесь, что происходит. При большом значении коэффициента происходит сильное зашумление, которое также достаточно сильно подавляется, что говорит о приемлемой работе фильтра. Но при маленьких значениях коэффициента, таких как, к примеру,  $b = 0.1$ , мы видим колебания на границах промежутка  $[t_1, t_2]$  при достаточно малом зашумлении сигнала. Их возникновение вполне естественно из-за работы с гармониками, но не является полезным при фильтрации.

Приведем листинг с функцией, реализующей высокочастотный жесткий фильтр.

#### Листинг 1. Высокочастотная жесткая фильтрация.

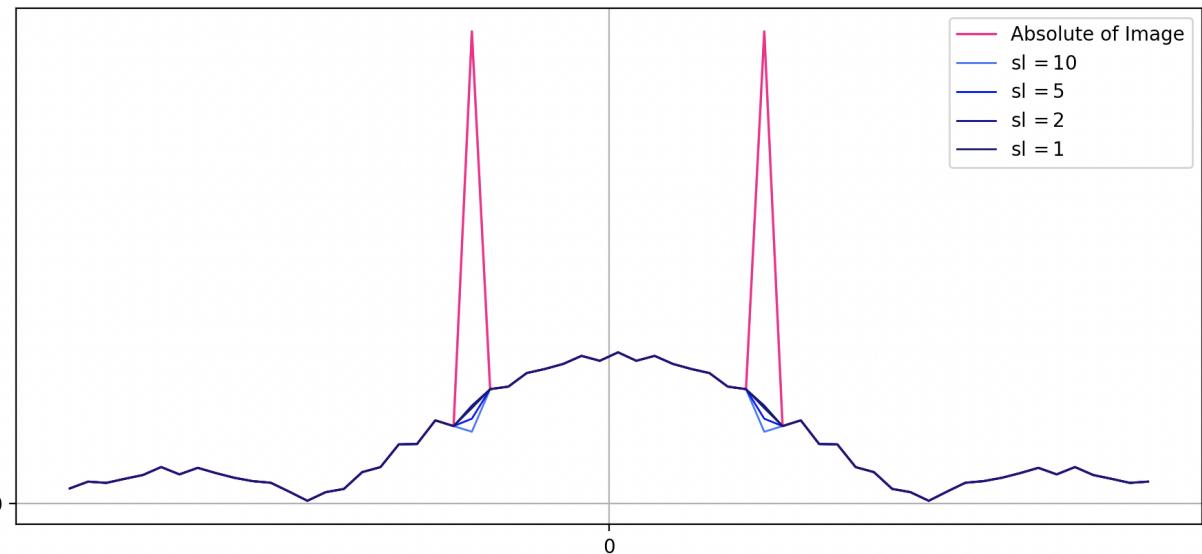
```

1  u_value = g_value + b * (np.random.rand(len(time)) - 0.5) # u function
2  u_transformed = tr.fftshift(tr.fft(u_value)) # fourier transform
3  u_tr = np.concatenate(([0 for i in range(int(u_transformed.shape[0] / 2 - sl))], u_transformed[int(u_transformed.shape[0] / 2 - sl):int(u_transformed.shape[0] / 2 + sl)], [0 for i in range(int(u_transformed.shape[0] / 2 + sl))])) # filtration
4  u_itr = tr.ifft(tr.ifftshift(u_tr)) # return filtered signal

```

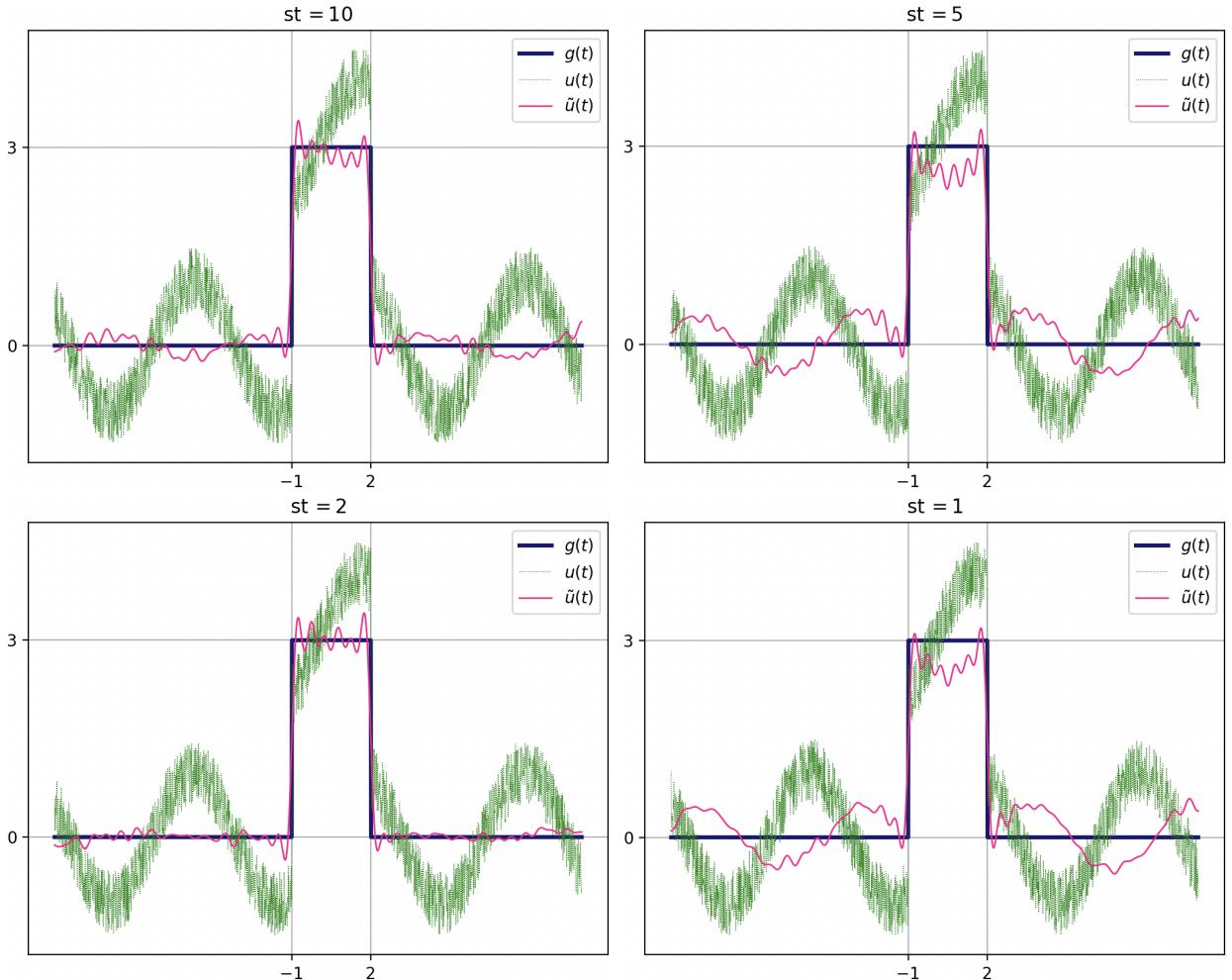
В предыдущем шаге мы работали с фильтром высоких частот – сейчас обработаем специфические частоты. Примем коэффициенты  $c$  и  $d$  ненулевыми (по умолчанию – единичными).

На графике модуля Фурье-образа сигнала можно увидеть два ярко выраженных пика – именно они влияют на наличие специфического зашумления. Удалим их и приведем графики модулей Фурье-образов для разных параметров среза  $sl$ :



**Рис. 4** – Сравнительный график модулей Фурье-образов специфически зашумленного сигнала в зависимости от параметра  $sl$ .

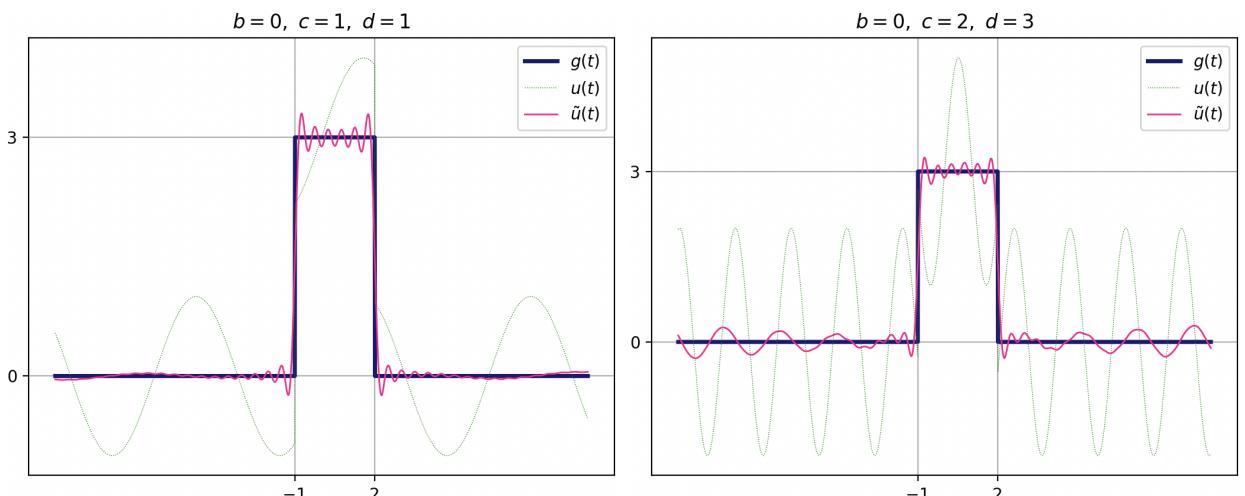
Удаление этих пиков приведет к уменьшению влияния синусоидальной составляющей зашумления, а выполнив дополнительно высокочастотную фильтрацию, уменьшим влияние иного зашумления. Мы знаем, как работает высокочастотная фильтрация, поэтому, выбрав оптимальное значение  $sl = 100$ , проиллюстрируем влияние на фильтрацию сигнала параметра  $st$ , отвечающего за коррекцию пиков:



**Рис. 5** – Сравнительные графики после фильтрации сигнала для различных значений  $st$ .

Сделаем некоторые выводы, опираясь на приведенные графики. Можно говорить о том, что чем “выгоднее” промежуток, на котором происходит удаление пиков (и их окрестностей, вообще говоря), тем ближе к истине фильтрованный сигнал. Если удалить слишком мало или слишком много, наличие синусоидального зашумления не будет исправлено.

Мы видим, что оптимальным значением параметра  $st$  является значение  $st = 2$ . Не изменяя его, посмотрим, что будет меняться при изменении коэффициентов  $b$ ,  $c$  и  $d$ :



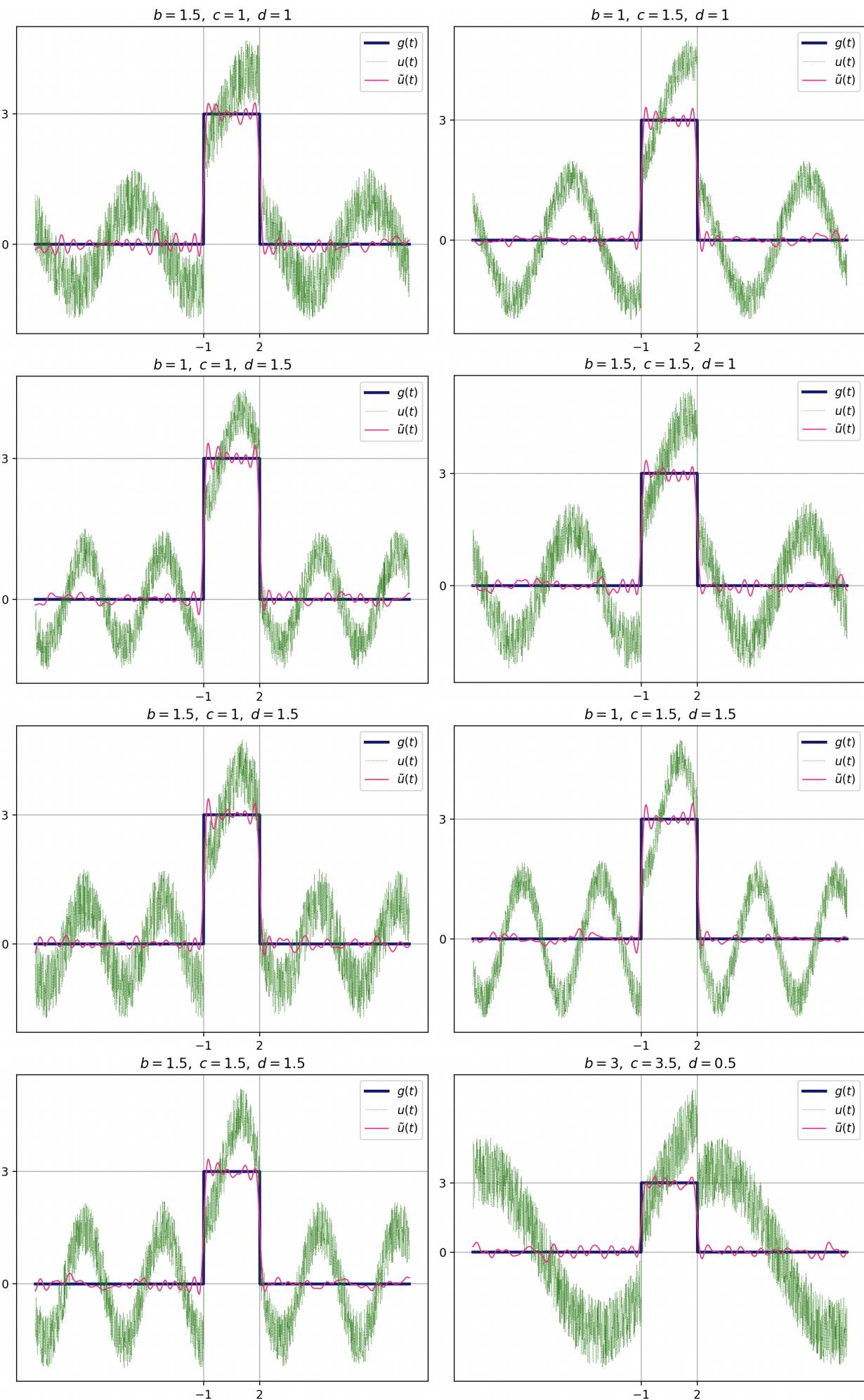


Рис. 6 – Сравнительные графики для различных значений коэффициентов  $b, c$  и  $d$ .

Как мы можем наблюдать, фильтр достаточно неплохо справился с зашумлением сигнала для любых из выставленных параметров  $b$ ,  $c$  и  $d$ . Приведем листинг, содержащий функцию для проведения фильтрации специфических частот.

### Листинг 2. Жесткая фильтрация специфических частот.

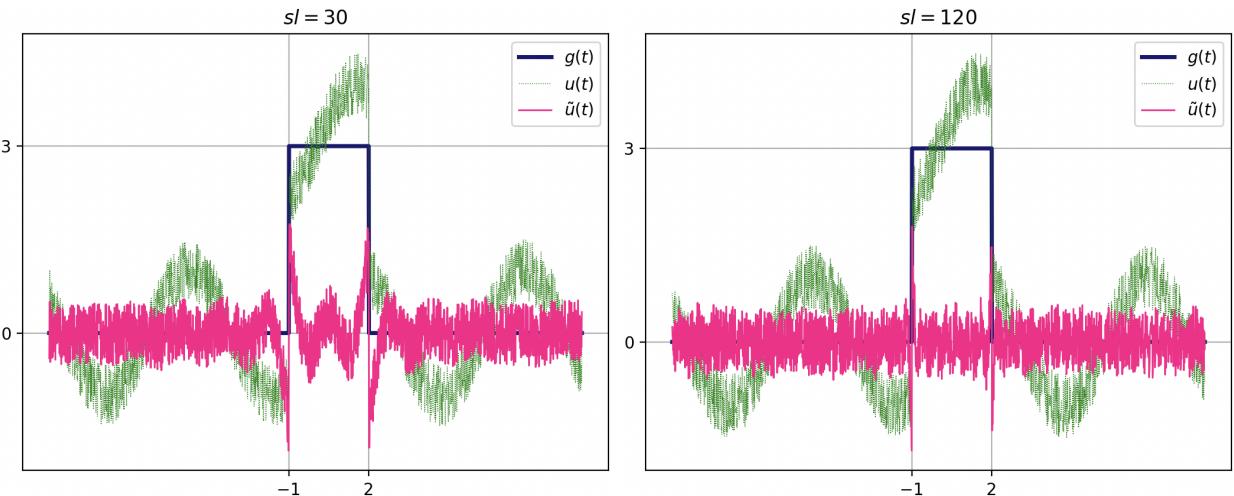
```

1  u_value = g_value + b * (np.random.rand(len(time)) - 0.5) + c * np.sin(d *
2    time)
3  u_transformed = tr.fftshift(tr.fft(u_value))
4  index_max = np.where(abs(u_transformed) == max(abs(u_transformed)))
5  u_tr_1 = np.array([u for u in u_transformed])
6  u_tr_1[index_max[0][0]] = (u_tr_1[index_max[0][0] - st] + u_tr_1[index_max
7    [0][0] + st]) / 2
8  u_tr_1[index_max[0][1]] = (u_tr_1[index_max[0][1] - st] + u_tr_1[index_max
    [0][1] + st]) / 2
9  u_tr = np.concatenate(([0 for i in range(int(u_tr_1.shape[0] / 2 - sl))],
10    u_tr_1[int(u_transformed.shape[0] / 2 - sl):int(u_tr_1.shape[0] / 2 +
11      sl)], [0 for i in range(int(u_tr_1.shape[0] / 2 + sl), u_tr_1.shape[0])]))
12  u_itr = tr.ifft(tr.ifftshift(u_tr))

```

По графикам модулей Фурье-образов ясно, что они являются четными функциями, а значит центрированными относительно оси  $Oy$ . Стоит предположить, что при проведении низкочастотной фильтрации мы получим сомнительный с точки зрения приближения к исходной функции результат, который все же может быть применим для решения некоторых задач, таких как частотный анализ звука (если нужно проанализировать исключительно высокие частоты).

Для примера построим графики для разных значений параметра  $sl$ , определяющего окрестность точки  $\nu = 0$ :



**Рис. 7** – Графики после фильтрации сигнала для различных значений  $sl$ .

Как можно заметить, низкочастотная фильтрация не дала приближения к функции – зашумление осталось, причем учитывая его можно сделать вывод о трудностях анализа сигнала в принципе.

Приведем листинг с функцией реализации низкочастотной фильтрации.

### Листинг 3. Низкочастотная жесткая фильтрация.

```

1  u_value = g_value + b * (np.random.rand(len(time)) - 0.5) + c * np.sin(d *
2    time)
3  u_transformed = tr.fftshift(tr.fft(u_value))

```

```

3     u_tr = np.concatenate((u_transformed[:int(u_transformed.shape[0] / 2 - mu)],
4                             [0 for i in range(int(u_transformed.shape[0] / 2 - mu), int(
5                               u_transformed.shape[0] / 2 + mu))], u_transformed[int(u_transformed.
6                               shape[0] / 2 + mu):]))
7     u_itr = tr.ifft(tr.ifftshift(u_tr))

```

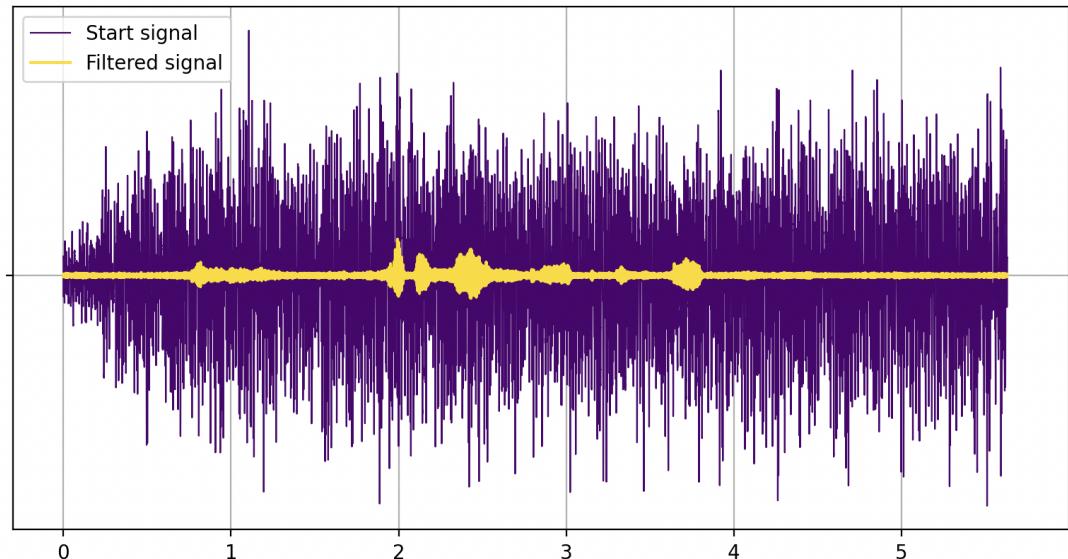
---

## Задание 2. Фильтрация звука

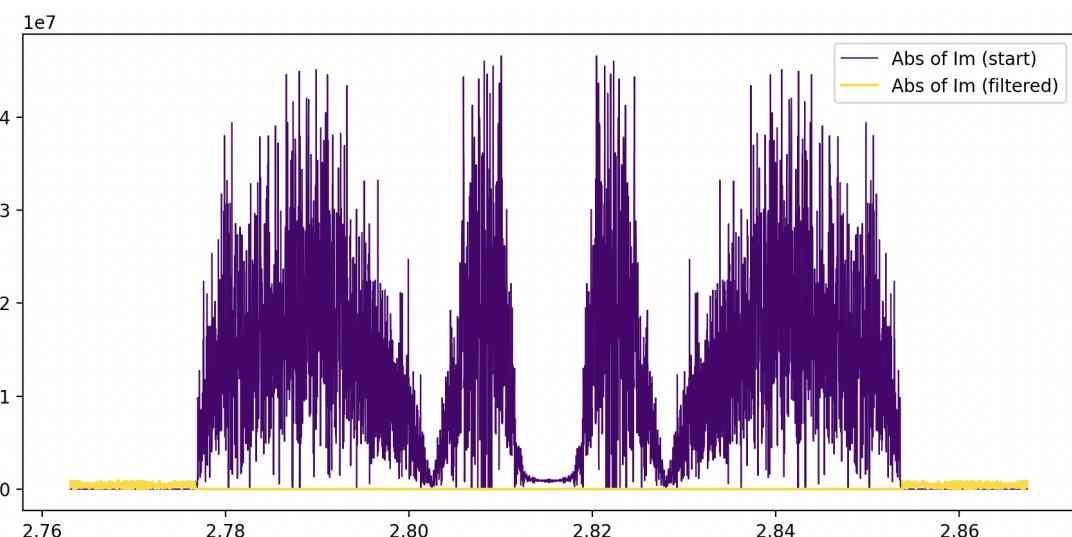
В качестве задания было предложено проанализировать частотно исходный аудиофайл, содержащий фразу "Муха – это маленькая птичка" и посторонний многочастотный шум. Так как на записи преобладает низкочастотный шум, удалить его напрямую не представляется возможность, поэтому алгоритм обработки был следующим:

- Создать копию массива данных исходной записи;
- Удалить из нее высокочастотные пики, соответствующие произнесенным словам;
- Произвести вычитание массивов (иными словами, мы создали маску и инвертировали ее);
- Удалить высокочастотные шумы.

В результате работы программы были получены следующие результаты:



**Рис. 8** – Графики исходного и фильтрованного звукового сигнала.



**Рис. 9** – Графики модулей Фурье-образов исходного и фильтрованного звукового сигнала.

**Листинг 4.** Код для жесткой фильтрации звукового сигнала.

```
1 import scipy.fft as tr
2
3 samplerate, data = wavfile.read(filename)
4 time = np.linspace(0., data.shape[0] / samplerate, data.shape[0])
5 fourier_image = tr.fftshift(tr.fft(data))
6 fourier_image_start = np.array([f for f in fourier_image])
7
8 abs_image = abs(fourier_image)
9 index = np.array([])
10 for i in range(len(abs_image)):
11     if abs_image[i] > 7e7:
12         index = np.append(index, i)
13 for j in index:
14     fourier_image[int(j)] = 0
15 index_2 = np.array([])
16 for i in range(len(abs_image)):
17     if abs_image[i] > 4.7e7:
18         index_2 = np.append(index_2, i)
19 for j in index_2:
20     fourier_image[int(j)] = 0
21
22 n1 = 1700
23 fourier_image = np.concatenate(([0 for i in range(int(len(fourier_image) /
2) - n1)], fourier_image[(int(len(fourier_image) / 2) - n1):(int(len(
fourier_image) / 2) + n1)], [0 for i in range(int(len(fourier_image) /
2) + n1, len(fourier_image))]))
24 voice_image = fourier_image_start - fourier_image
25
26 abs_voice = abs(voice_image)
27 index_3 = np.array([])
28 for i in range(len(abs_voice)):
29     if abs_voice[i] > 0.9e6:
30         index_3 = np.append(index_3, i)
31 for j in index_3:
32     voice_image[int(j)] = 0
33
34 data_filtered = tr.ifft(tr.ifftshift(voice_image))
```