Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Инженерная школа информационных технологий и робототехники
Отделение информационных технологий
Направление подготовки 09.04.04 Программная инженерия

Отчёт по лабораторной работе №1

МЕТОДЫ ФИЛЬТРАЦИИ СИГНАЛОВ

по дисциплине Основы теории управления автономными системами

Выполнил студент гр. 8ПМ4Л _____ (Подпись) _____ (Дата) Сокуров Р.Е. (Фамилия И.О.)

Проверил доцент ОАР _____ (Подпись) _____ (Дата) Хожаев И.В. (Фамилия И.О.)

Томск 2024 г.

Цель

Изучить основные методы фильтрации измерительных сигналов.

Задачи

1) Изучить принцип работы следующих фильтров:

– фильтра экспоненциального сглаживания;

– фильтра скользящего среднего;

– медианного фильтра;

– фильтра с ограничением скорости нарастания сигнала;

2) Сгенерировать полезный сигнал и добавить к нему равномерно распределенный шум и всплески большой амплитуды;

3) Реализовать каждый из ранее перечисленных фильтров любым известным способом, проверить работоспособность фильтров на сгенерированном ранее зашумленном сигнале и изучить влияние настроечного параметра фильтра на качество обработки сигнала;

4) Подобрать комбинацию фильтров, отделяющих полезный сигнал от шума обоих типов; сравнить исходный полезный сигнал и отфильтрованный;

5) Оформить отчет.

Ход работы

1. Генерация полезного сигнала и добавление к нему равномерно распределённого шума и всплесков большой амплитуды.

В качестве генерируемого сигнала была выбрана функция $y = \sin(x)$. Данная функция была рассчитана на массиве $x \in [-4\pi; 4\pi]$, состоящем из 800 элементов. Затем, с помощью библиотеки «NumPy» языка «Python» был добавлен равномерно распределённый шум, а также высокоамплитудные случайные всплески (Рисунок 1). Код отображён в файле «code.py» в приложении А.

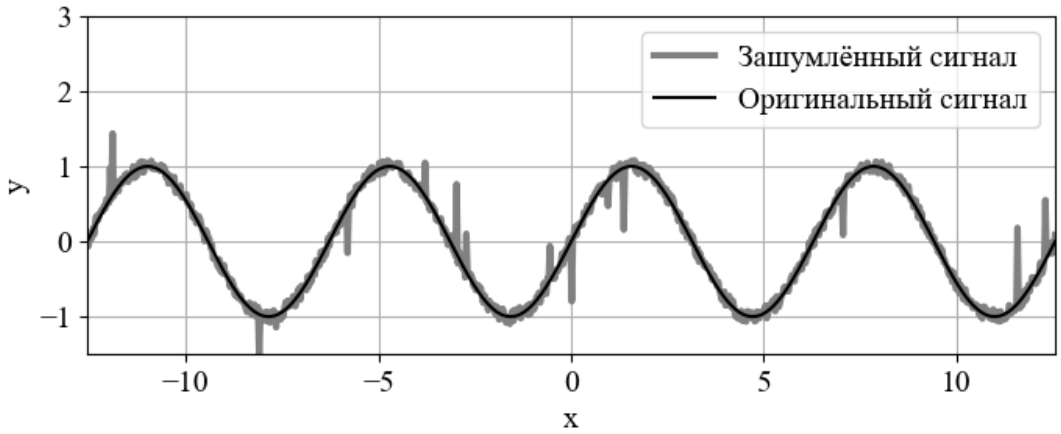

Рисунок 1 – Генерация зашумлённого сигнала

2. Реализация фильтра экспоненциального сглаживания.

Фильтр экспоненциального сглаживания – простейший фильтр низких частот, выходной сигнал которого равен взвешенной сумме последнего измеренного значения сигнала и значения отфильтрованного сигнала на предыдущем шаге:

$$y_n = \alpha \cdot x_n + (1-\alpha) \cdot y_{n-1},$$

где x – измеренные значения (зашумлённый сигнал), y – значения отфильтрованного сигнала, α – настроечный параметр фильтра.

Данный фильтр был реализован в функции «Exponential_smoothing_filter» в файле «code.py» приложения А.

Данная функция была использована с тремя разными значениями коэффициента $\alpha \in \{0.1, 0.25, 0.5\}$:

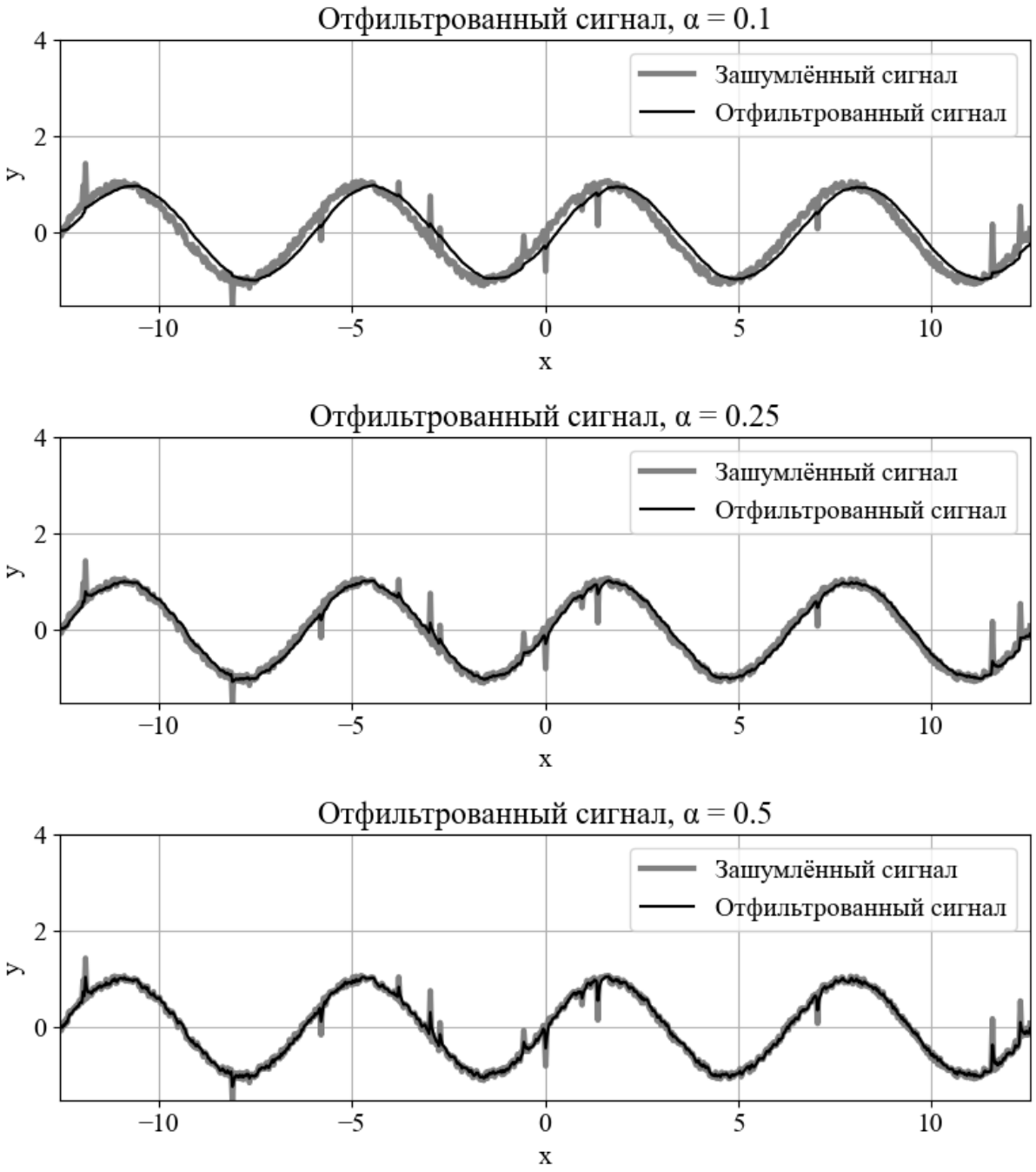

Рисунок 2 – Демонстрация работы фильтра экспоненциального сглаживания

По итогам анализа полученных графиков, оптимальным было принято значение $\alpha = 0.25$, поскольку данное значение значительно сглаживает высокочастотные помехи в сигнале, но ещё не создаёт столь значимое запаздывание, как, например, значение $\alpha = 0.1$.

3. Реализация фильтра скользящего среднего

Фильтр скользящего среднего является фильтром низких частот. Он хранит последние n измерений зашумленного сигнала и выдает на выход их среднее арифметическое.

Данный фильтр был реализован в функции «moving_average_filter» в файле «code.py» приложения А, значение $n \in \{8, 16, 32\}$:

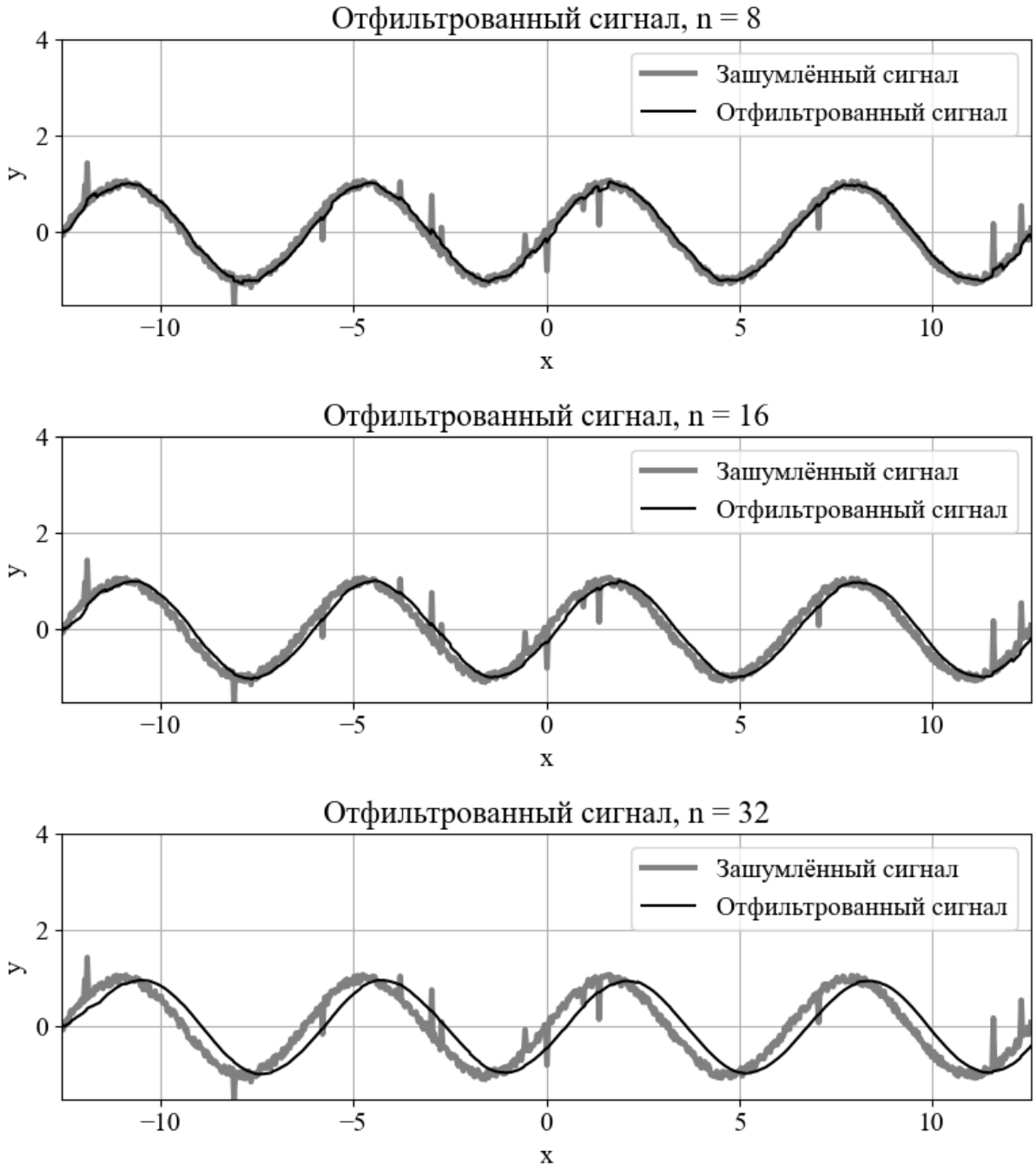

Рисунок 3 – Демонстрация работы фильтра скользящего среднего

Здесь наилучшим значением показал себя размер окна, равный 16, поскольку отставание отфильтрованного сигнала по вазе становится уже слишком большим при $n = 32$, а при $n = 8$ видны участки с резкими переходами.

4. Реализация медианного фильтра

Медианный фильтр является фильтром высоких частот. Его выходной сигнал равен медиане последних n измерений фильтруемого сигнала. Степень сглаживания сигнала также определяется шириной окна n.

Данный фильтр был реализован в функции «median_filter» в файле «code.py» приложения А, значение $n \in \{8, 16, 32\}$:

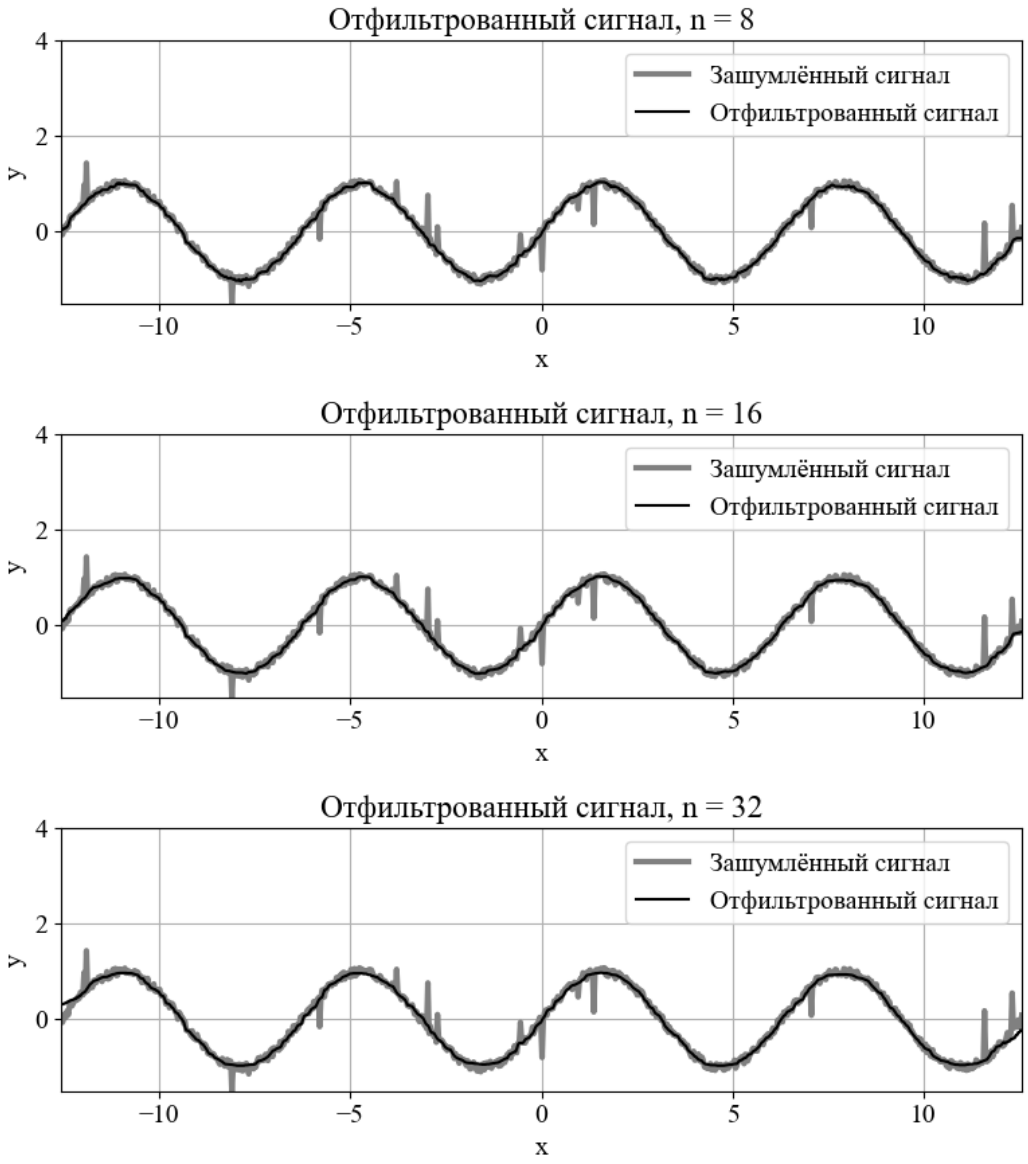

Рисунок 4 – Демонстрация работы медианного фильтра

Здесь, уже при размере окна $n = 8$ фильтр уже справился с низкими частотами (высокоамплитудных всплесков в итоговом сигнале нет), однако лучшим всё же показался размер окна $n = 32$, хотя стоит учесть, что фильтр может выдавать некорректные значения в начале (до того, как заполнится размер окна).

5. Реализация фильтра с ограничением скорости нарастания сигнала

Фильтр с ограничением скорости нарастания сигнала выводит на выход значение входного сигнала без изменений, если приращение между текущим и предыдущим его значением меньше заданного порогового значения. В противном случае значение выходного сигнала фильтра не изменяется.

Данный фильтр был реализован в функции «rate_limit_filter» в файле «code.py» приложения А, значение $threshold \in \{0.05, 0.1, 0.2\}$:

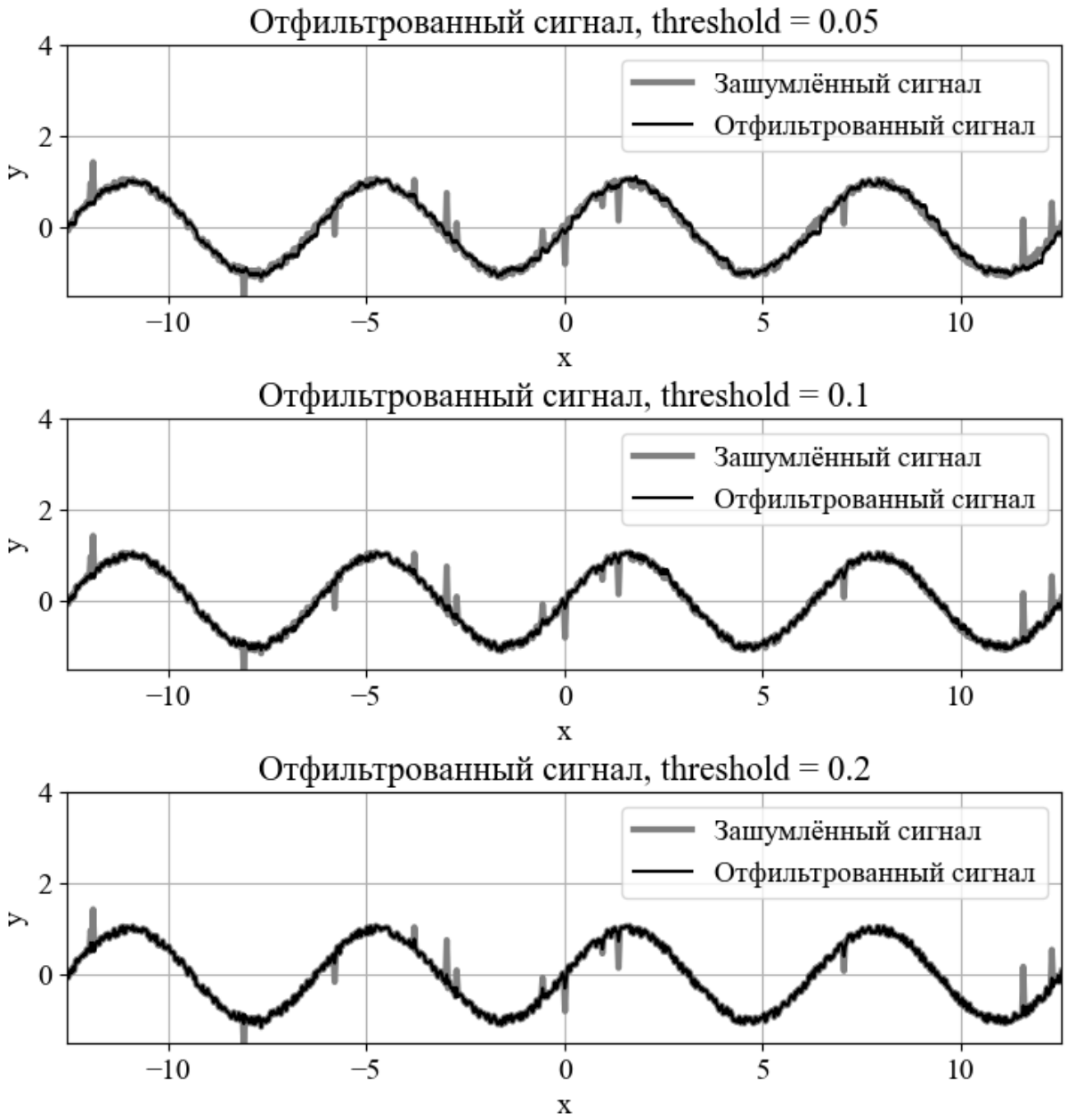

Рисунок 5 – Демонстрация работы фильтра с ограничением скорости нарастания

В данном фильтре, в отличие от всех предыдущих, именно увеличение значения *threshold* ведёт к большей схожести с оригинальным сигналом. Наилучшим значением было принято $threshold = 0.1$, поскольку при $threshold = 0.05$ уже появляются искажение сигнала.

6. Реализация ансамбля (комбинации) фильтров

Для реализации комбинации фильтров, которые лучше всех воспроизводят исходный сигнал из зашумленного, были выбраны медианный фильтр с $n = 32$ и фильтр экспоненциального сглаживания с $\alpha = 0.25$:

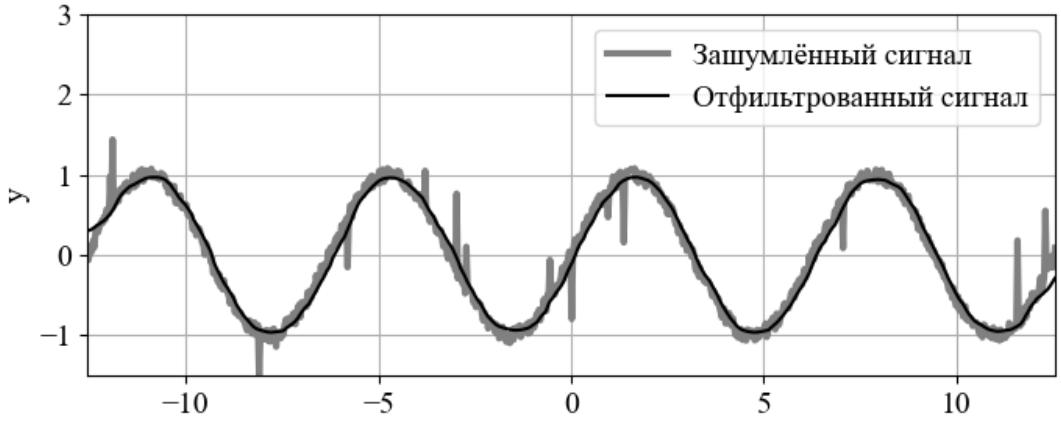

Рисунок 6 – Отфильтрованный двумя фильтрами сигнал

Несмотря на отклонение значений в начале и в конце (из-за особенностей работы медианного фильтра), данный ансамбль отлично справляется со своей задачей фильтрации, при условии, что интересующая нас информация находится не на краях графика.

Заключение

В ходе данной лабораторной работы был составлен полезный сигнал $y = \sin(x)$, из которого, путём искусственного зашумления, получили материал для дальнейшей обработки фильтрами. Были исследованы 4 вида фильтров: фильтр экспоненциального сглаживания, фильтр скользящего среднего, медианный фильтр, фильтр с ограничением скорости нарастания сигнала. Каждый из них рассматривался при различных значениях настраиваемых параметров.

По итогам данных исследований было выбрано два фильтра, комбинация которых использовалась в п.6 работы для максимального приближения зашумлённого сигнала к исходному.

Приложение А

Реализация фильтров

```python
# Подключение необходимых библиотек
import numpy as np
import matplotlib.pyplot as plt

# Установление зерна случайно генерации
# Для возобнавляемости экспериментов
np.random.seed(52)

# Постоянные переменыые
ALPHAS = [0.1, 0.25, 0.5] # Степень влияния прошлых значений для Exponential_smoothing_filter
N = [8, 16, 32] # Используется в скользящем среднем и медианном фильтрах
THRESHOLDS = [0.05, 0.1, 0.2] # Порог скорости нарастания сигнала

# Настройки шрифта для графика
plt.rcParams['font.family'] = 'Times New Roman'
plt.rcParams['font.size'] = 14

# ------------------------------------  Создание функции каждого фильтра  ----------------------------------#
def Exponential_smoothing_filter(data, alpha):
    """
    Функция реализации фильтра экспоненциального сглаживания

    :param a: массив данных, которые нужно отфильтровать
    :param alpha: регулировочный коэффициент
    :return: отфильтрованный массив
    """
    filtered_data = data.copy()
    for i in range(1, filtered_data.size):
        filtered_data[i] = alpha * filtered_data[i] + (1-alpha) * filtered_data[i-1]
    return filtered_data

def moving_average_filter(data, n):
    """
    Функция реализации фильтра скользящего среднего

    :param data: массив данных, которые нужно отфильтровать
    :param n: ширина окна усреднения
    :return: отфильтрованный массив
    """
    # Массив для хранения отфильтрованных значений
    filtered_data = []

    # Массив для хранения текущего окна измерений
    current_values = []

    # Пройдем по всем элементам исходного массива
    for new_value in data:
        # Добавляем текущее новое значение в массив измерений
        current_values.append(new_value)

        # Если количество элементов меньше n, считаем среднее по всем доступным значениям
        if len(current_values) < n:
            avg = np.mean(current_values)
        else:
            # Как только элементов стало n или больше, считаем среднее по последним n значениям
            avg = np.mean(current_values[-n:])

        # Добавляем отфильтрованное значение в результат
        filtered_data.append(avg)

    return filtered_data

def median_filter(data, window_size):
    """
    Функция реализации медианного фильтра

    :param data: массив данных, которые нужно отфильтровать
    :param n: ширина окна расчёта медианы
    :return: отфильтрованный массив
    """
    # Если размер окна меньше 1 или больше длины сигнала, возвращаем исходный сигнал
```

```
    if window_size < 1 or window_size > len(data):
        return data

    # Создаем список для хранения отфильтрованных значений
    filtered_data = []

    half_window = window_size // 2

    # Проходим по каждому элементу сигнала
    for i in range(len(data)):
        # Выбираем окно вокруг текущего элемента
        start_idx = max(0, i - half_window)
        end_idx = min(len(data), i + half_window + 1)

        # Получаем подмассив, сортируем его и находим медиану
        window = sorted(data[start_idx:end_idx])

        # Если окно нечетное, выбираем центральный элемент
        if window_size % 2 == 1:
            median = window[len(window) // 2]
        else:
            # Если окно четное, берем среднее арифметическое двух центральных элементов
            mid1 = window[len(window) // 2 - 1]
            mid2 = window[len(window) // 2]
            median = (mid1 + mid2) / 2.0

        # Добавляем медиану в отфильтрованный сигнал
        filtered_data.append(median)

    return np.array(filtered_data)

def rate_limit_filter(data, threshold):
    """
    Функция реализации функции ограниченной скорости нарастания сигнала

    :param data: массив данных, которые нужно отфильтровать
    :param threshold: пороговое значение скорости нарастания
    :return: отфильтрованный массив
    """

    # Инициализация массива выходного сигнала
    filtered_data = [data[0]]

    # Применение фильтра
    for i in range(1, len(data)):
        delta = data[i] - data[i-1]
        if abs(delta) <= threshold:
            filtered_data.append(data[i])
        else:
            # Ограничение прироста
            if delta > 0:
                filtered_data.append(filtered_data[-1] + threshold)
            else:
                filtered_data.append(filtered_data[-1] - threshold)

    return filtered_data

# ---------------------  Создание чистого и зашумленного сигналов для дальнейших эксериментов ------------------#
x = np.linspace(-4*np.pi, 4*np.pi, 800) # Создание массива иксов
y = np.sin(x) # Функция
y_clean = np.sin(x)
noise = np.random.uniform(-0.1,0.1,800) # Добавляем равномерно распределенный шум
spikes = np.random.choice([0, 1], size=x.shape, p=[0.98, 0.02])  # Случайные всплески (2% вероятность)
amplitude = np.random.uniform(-1, 1, size=x.shape)  # Случайная амплитуда всплесков
y += noise + spikes * amplitude # дальше наш сигнал это зашумлённый сигнал

# ---------------------  Вывод на график результата фильтрации (нужный раскоментить)  ---------------------#

#y_exp = Exponential_smoothing_filter(y)
```

```
# plt.figure(figsize=[8,3])
# plt.plot(x,y, color='gray', linewidth=3, label='Зашумлённый сигнал')
# plt.plot(x,y_clean, color='black', linewidth=1.5, label='Оригинальный сигнал')
# plt.grid()
# ax = plt.gca()
# ax.set_xlim([-4*np.pi,4*np.pi])
# ax.set_ylim([-1.5,3])
# plt.xlabel('x')
# plt.ylabel('y')
# plt.legend()

# ---------------------  Вывод на график результата фильтрации (нужный раскоментить)  ---------
-------------------#

# plt.figure(figsize=[8, 9])  # Размер всей фигуры: 8x9 (3 графика по 8x3)

# for idx, alpha in enumerate(ALPHAS):
#     y_exp = Exponential_smoothing_filter(y, alpha)
#     plt.subplot(len(ALPHAS), 1, idx + 1)
#     plt.plot(x, y, color='gray', linewidth=3, label='Зашумлённый сигнал')
#     plt.plot(x, y_exp, color='black', linewidth=1.5, label=f'Отфильтрованный сигнал')
#     plt.grid()
#     plt.xlim([-4 * np.pi, 4 * np.pi])
#     plt.ylim([-1.5, 4])
#     plt.xlabel('x')
#     plt.ylabel('y')
#     plt.legend()
#     plt.title(f'Отфильтрованный сигнал, α = {alpha}')

# plt.tight_layout()

# ---------------------  Вывод на график результата фильтрации (нужный раскоментить)  ---------
-------------------#

# plt.figure(figsize=[8, 9])  # Размер всей фигуры: 8x9 (3 графика по 8x3)

# for idx, n in enumerate(N):
#     y_exp = moving_average_filter(y, n)
#     plt.subplot(len(N), 1, idx + 1)
#     plt.plot(x, y, color='gray', linewidth=3, label='Зашумлённый сигнал')
#     plt.plot(x, y_exp, color='black', linewidth=1.5, label=f'Отфильтрованный сигнал')
#     plt.grid()
#     plt.xlim([-4 * np.pi, 4 * np.pi])
#     plt.ylim([-1.5, 4])
#     plt.xlabel('x')
#     plt.ylabel('y')
#     plt.legend()
#     plt.title(f'Отфильтрованный сигнал, n = {n}')

# plt.tight_layout()

# ---------------------  Вывод на график результата фильтрации (нужный раскоментить)  ---------
-------------------#

# plt.figure(figsize=[8, 9])  # Размер всей фигуры: 8x9 (3 графика по 8x3)

# for idx, n in enumerate(N):
#     y_exp = median_filter(y, n)
#     plt.subplot(len(N), 1, idx + 1)
#     plt.plot(x, y, color='gray', linewidth=3, label='Зашумлённый сигнал')
#     plt.plot(x, y_exp, color='black', linewidth=1.5, label=f'Отфильтрованный сигнал')
#     plt.grid()
#     plt.xlim([-4 * np.pi, 4 * np.pi])
#     plt.ylim([-1.5, 4])
#     plt.xlabel('x')
#     plt.ylabel('y')
#     plt.legend()
#     plt.title(f'Отфильтрованный сигнал, n = {n}')

# plt.tight_layout()

```

```
209. # ----------------------  Вывод на график результата фильтрации (нужный раскоментить)  ---------
------------------#
210.
211. # plt.figure(figsize=[8, 9])  # Размер всей фигуры: 8x9 (3 графика по 8x3)
212.
213. # for idx, threshold in enumerate(THRESHOLDS):
214. #     y_exp = rate_limit_filter(y, threshold)
215. #     plt.subplot(len(THRESHOLDS), 1, idx + 1)
216. #     plt.plot(x, y, color='gray', linewidth=3, label='Зашумлённый сигнал')
217. #     plt.plot(x, y_exp, color='black', linewidth=1.5, label=f'Отфильтрованный сигнал')
218. #     plt.grid()
219. #     plt.xlim([-4 * np.pi, 4 * np.pi])
220. #     plt.ylim([-1.5, 4])
221. #     plt.xlabel('x')
222. #     plt.ylabel('y')
223. #     plt.legend()
224. #     plt.title(f'Отфильтрованный сигнал, threshold = {threshold}')
225.
226. # plt.tight_layout()
227.
228. # ----------------------  Вывод на график результата фильтрации (нужный раскоментить)  ---------
------------------#
229.
230. y_med = median_filter(y, 32)
231. y_exp = Exponential_smoothing_filter(y_med, 0.25)
232. plt.figure(figsize=[8,3])
233. plt.plot(x,y, color='gray', linewidth=3, label='Зашумлённый сигнал')
234. plt.plot(x,y_exp, color='black', linewidth=1.5, label='Отфильтрованный сигнал')
235. plt.grid()
236. ax = plt.gca()
237. ax.set_xlim([-4*np.pi,4*np.pi])
238. ax.set_ylim([-1.5,3])
239. plt.xlabel('x')
240. plt.ylabel('y')
241. plt.legend()
242.
243. plt.show()
244.
```