ГУАП

КАФЕДРА № 41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ 10/10
ПРЕПОДАВАТЕЛЬ

ассистент	31.10.2020	Б. К. Акопян
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

ИЗУЧЕНИЕ ЦИФРОВОГО ФИЛЬТРА НА ОСНОВЕ МЕТОДА НАИМЕНЬШИХ КВАДРАТОВ

по курсу: методы и устройства цифровой обработки сигналов

Вариант 3

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР 4711		Хасанов Б.Р.
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2020

1 Цель работы

Изучить метод цифровой обработки сигналов с использованием фильтра на основе метода наименьших квадратов (МНК-фильтра).

2 Краткие теоретические сведения

Пусть x — набор n неизвестных переменных (параметров), $f_i(x)$, i = 1,…, m, m > n — совокупность функций от этого набора переменных. Задача заключается в подборе таких значений x чтобы значения этих функций были максимально близки к некоторым значениям y_i. По существу речь идет о «решении» переопределенной системы уравнений $f_i(x) = y_i$, i = 1, …, m в указанном смысле максимальной близости левой и правой частей системы. Суть МНК заключается в выборе в качестве «меры близости» суммы квадратов отклонений левых и правых частей $|f_i(x) — y|$. Таким образом, сущность МНК может быть выражена следующим образом:

$$\sum_i e_i^2 = \sum_i (y_i - f_i(x))^2 \rightarrow \min_x$$

В случае, если система уравнений имеет решение, то наименьшее значение суммы квадратов будет равно нулю, и могут быть найдены точные решения системы уравнений аналитически или, например, различными численными методами оптимизации. Если система переопределена, то есть, говоря нестрого, количество независимых уравнений больше количества искомых переменных, то система не имеет точного решения и метод наименьших квадратов позволяет найти некоторый «оптимальный» вектор x в смысле максимальной близости векторов y и f(x) или максимальной близости вектора отклонений e к нулю.

3 Ход работы

Программа написана на языке программирования python 3

Стандартно в начале главной функции импортируются нужные библиотеки

```
import numpy as np
import matplotlib.pyplot as plt
from statistics import stdev
from numpy import log, pi, cos
```

Затем, генерируем сигнал данный нам по заданию. Результат генерации сигнала на рисунке 3.1.

```
main_signal_length = 3000
time = np.linspace(0, 2, main_signal_length)
generated_signal = log(2 - cos(2 * pi * 3 * time))
```

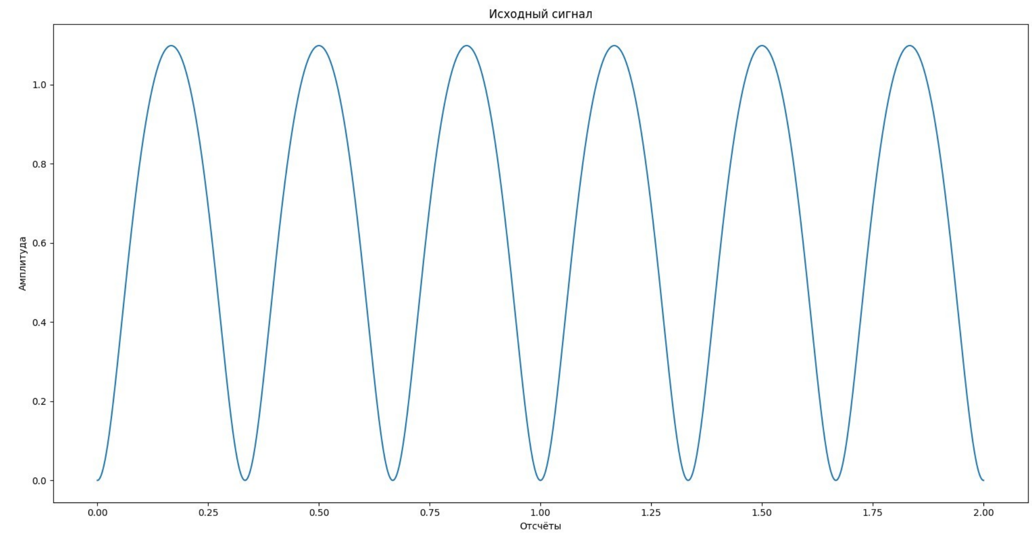

Рисунок 3.1 — Исходный сигнал

Далее, генерируется гауссовский шум с дисперсией 1 и объединяется с сигналом с рисунка 3.1. Результат обозначен на рисунке 3.2 как “Зашумлённый сигнал”:

```
gaussian_noise_signal = (generated_signal
                         + np.random.normal(0, 1, main_signal_length))
```

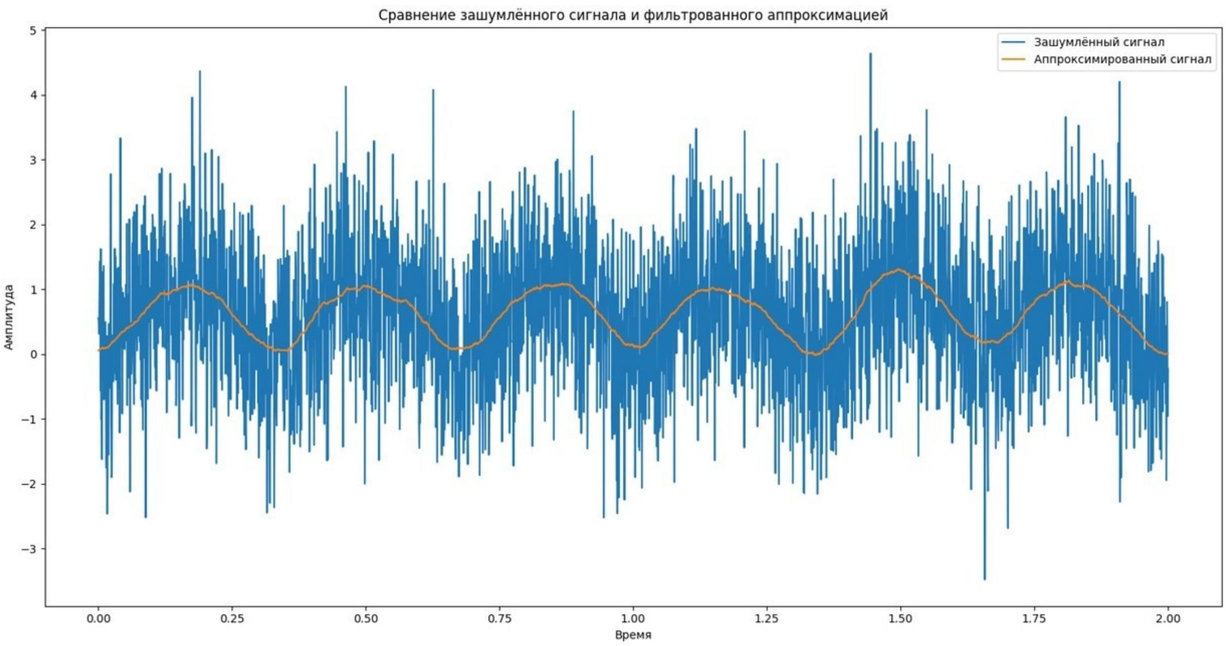

Рисунок 3.2 – Сравнение зашумлённого сигнала и сигнала фильтрованного аппроксимацией

Генерируем константы для степенного полинома. Порядок фильтра 3 – дан по заданию. Размер окна взят с запасом, чтобы можно было определить оптимальный.

```
order_max = 3
order_range = np.linspace(0, order_max - 1, order_max)
moving_window_max = 380
```

Следующие строчки находят зависимость среднеквадратической ошибки от длинны окна и вычисляют индекс минимальной среднеквадратической ошибки.

```
stdev_array = []
plot_start = 280
window_range = range(plot_start, moving_window_max)
for window_number in window_range:
    # Вычисляем полином для конкретного размера окна
    polinominal = least_squares_generate(window_number, order_range)

    # Аппроксимируем сигнал полученным полиномом
    approximated_signal = approximate_signal_with_basis(
            window_number, gaussian_noise_signal, polinominal)

    # Вычисляем среднеквадратическую ошибку
    signal_delta = generated_signal - approximated_signal
    signal_stdev = stdev(signal_delta)/stdev(generated_signal)
    stdev_array.append(signal_stdev)

index_of_minimum_stdev = stdev_array.index(min(stdev_array)) + plot_start
```

Функция least_squares_generate генерирует полином с заданной длиной окна и порядком базиса по формуле $\varphi_{k,m} = \left(\frac{k}{\sqrt{L}}\right)^m$, где k = 0, 1..L — 1; L – длинна окна, m – порядок фильтра. Вот её содержание:

```
def least_squares_polinomial_generate(window_length, basis_order_range):
    """Генерирует мнк полином с заданной длинной окна и порядком базиса"""
    import numpy as np

    moving_window_range = np.linspace(0, window_length - 1, window_length)

    least_squares_array = []
    for order in basis_order_range:
        least_squares_array.append(
            (moving_window_range/np.sqrt(window_length))**order)

    return np.array(least_squares_array).transpose()
```

Функция approximate_signal_with_basis сглаживает сигнал с помощью сгенерированного выше полинома. Вот её содержание:

```
def approximate_signal_with_basis(window_length, signal_for_smooth, basis):
    """
    Сглаживает сигнал методом наименьших квадратов с помощью скользящего окна

    :window_length: Длинна скользящего окна
    :signal_for_smooth: Сигнал для сглаживания
    :basis: базис
    :return: фильтрованный сигнал
    """
    import numpy as np
```

```
    approximated_signal = []
    for number in range(0, len(signal_for_smooth)):

        # Двигаем окно
        if number < int(window_length/2):
            cutted_signal = np.zeros(window_length)

            for signal_number in range(int(window_length/2) - number,
                                       window_length):
                cutted_signal[signal_number] = signal_for_smooth[
                        signal_number - int(window_length/2) + number
                        ]

        elif number > (len(signal_for_smooth) - int(window_length/2)
                       - (window_length % 2)):
            cutted_signal = np.zeros(window_length)
            for signal_number in range(0, int(window_length/2)
                                       + (len(signal_for_smooth) - number)):
                cutted_signal[signal_number] = signal_for_smooth[
                        signal_number + number - int(window_length/2)]
        else:
            cutted_signal = signal_for_smooth[
                    number - int(window_length/2):
                    number + int(window_length/2) + window_length % 2]
        cutted_signal = np.array(cutted_signal)

        # Находим коэффициент разложения
        transpose_basis = basis.transpose()
        coefficient_of_decomposition = np.matmul(np.linalg.inv(np.matmul(
            transpose_basis, basis)),
            np.matmul(transpose_basis, cutted_signal))

        # Осуществляем аппроксимацию на основе базисной функции
        approximated_signal.append(sum(
            np.multiply(coefficient_of_decomposition,
                        basis[int(window_length/2)
                              + window_length % 2])))

    return np.array(approximated_signal)
```

Блок кода под комментарием «Двигаем окно» осуществляет сдвиг окна, каждый отсчёт сигнала «вырезая» выборку заданной длинной таким образом, чтобы текущий отсчёт был в середине вырезанной последовательности (или слева от середины, если длина окна чётная). Если отсчёты находятся по краям выборки, то недостающие отсчёты заменяются нулями.

Блок кода под комментарием «Находим коэффициент разложения» находит этот коэффициент по формуле $\vec{a}=(\varphi^T\varphi)^{-1}\varphi^T\vec{x}$, где x – сигнал для аппроксимации.

Затем в функции осуществляется непосредственно аппроксимация с помощью формулы $y_n = \sum_{m=0}^{M-1} a_m \varphi_{\frac{L}{2}, m}$

Возвращаемся в main(), там с помощью строчек ниже рисуем график зависимости среднеквадратической ошибки от длинны окна для фиксированного зашумлённого сигнала и там же пишем наименьшее значение среднеквадратической ошибки и размер окна для не ё, чтобы воспроизвести этот результат далее. График показан на рисунке 3.3

```
plt.figure()
plt.xlabel("Порядки базиса")
plt.ylabel("Величина среднеквадратической ошибки")
plt.plot(window_range, stdev_array)
plt.text(plot_start, min(stdev_array), "Наименьшее значение " +
        "среднеквадратической ошибки = " +
        "{}".format(round(min(stdev_array), 3)), fontsize=12)
plt.text(plot_start, min(stdev_array) + 0.001, "Порядок наименьшего" +
        "значения среднеквадратической ошибки = " +
        "{}".format(index_of_minimum_stdev), fontsize=12)
```

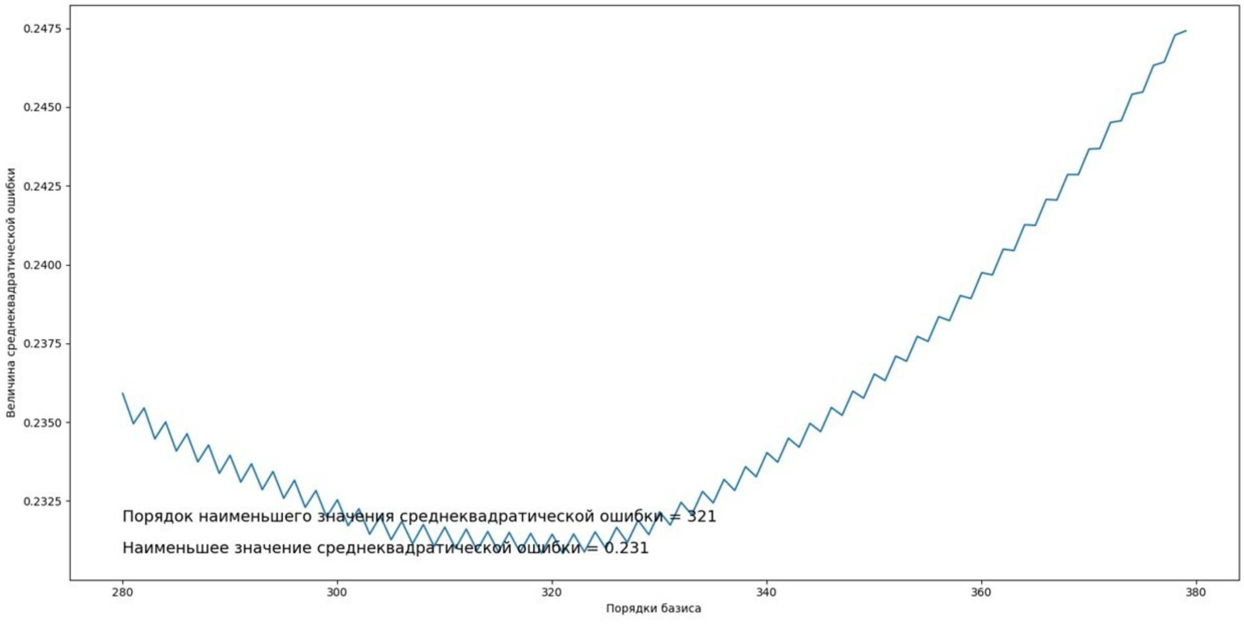

Рисунок 3.3 – Зависимость величины среднеквадратической ошибки от размера окна

С помощью следующих строчек воспроизводим только что найденный лучший результат. Повторяем все действия для него: генерируем полином, аппроксимируем сигнал с помощью этого полинома, рассчитываем среднеквадратическую ошибку

```
polinominal = least_squares_polinomial_generate(index_of_minimum_stdev,
                                                order_range)

# Аппроксимируем
approximated_signal = approximate_signal_with_basis(index_of_minimum_stdev,
```

```
gaussian_noise_signal,
polinominal)
signal_delta = generated_signal - approximated_signal
signal_stdev = stdev(signal_delta)/stdev(generated_signal)
```

Рисуем полученный лучший сигнал с помощью следующих строчек. Результат на рисунке 3.2 помечен как «аппроксимированный сигнал»

```
plt.figure()
plt.title("Сравнение зашумлённого сигнала и фильтрованного аппроксимацией")
plt.xlabel("Время")
plt.ylabel("Амплитуда")
plt.plot(time, gaussian_noise_signal, label="Зашумлённый сигнал")
plt.plot(time, approximated_signal, label="Аппроксимированный сигнал")
plt.legend()
```

С помощь функции plot_signal_difference выводим разность сигнала и его среднеквадратическую ошибку (она совпадает с той, что написана на рисунке 3.3). Сама функция не делает ничего особенного, её содержание можно посмотреть в приложении А. Результат на рисунке 3.4

```
plot_signal_difference(signal_delta, signal_stdev)
```

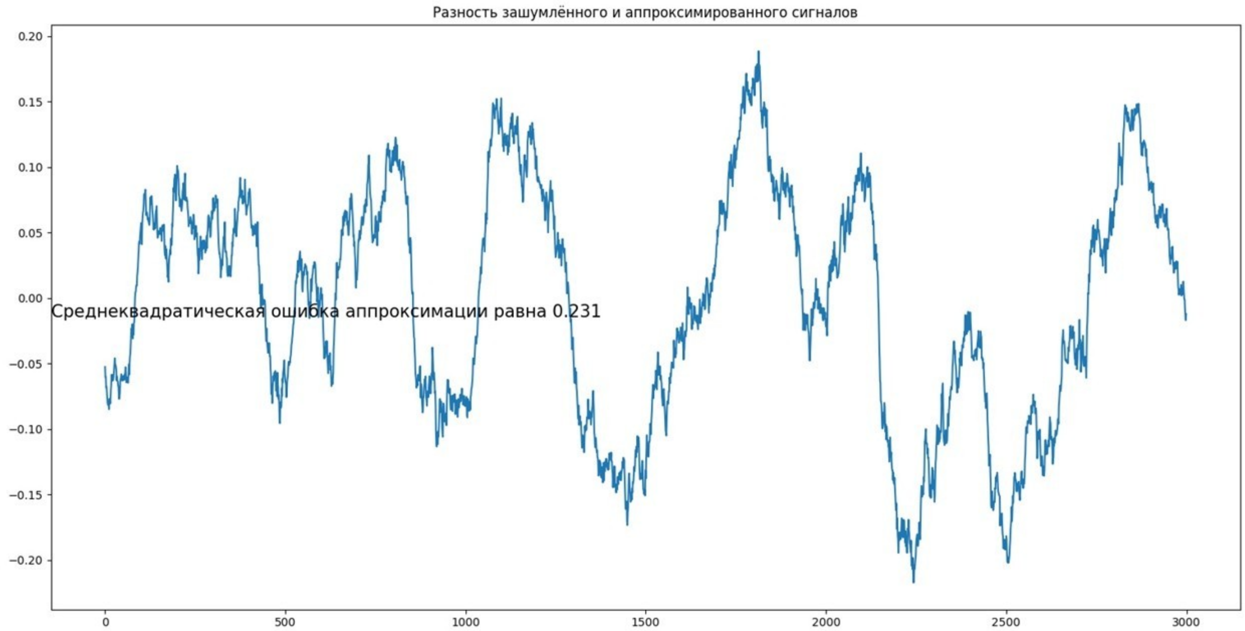

Рисунок 3.4 – Зависимость среднеквадратической ошибки от порядка базиса

4 Выводы

В рамках данной работы мной был профильтрован сигнал на основе метода наименьших квадратов. Для этого был использован степенной полином. На практике убедился в том, что данный метод вполне способен фильтровать сигнал. Справляется он с сигналом не идеально. Это видно на графике 3.2, при этом если сравнивать аппроксимированный сигнал с рисунка 3.2 и чистый сигнал на рисунке 3.1, то форма исходного сигнала вполне угадывается.

Так же, в рамках работы я определил оптимальную длину окна для заданного порядка полинома. При этой длине окна можно наблюдать наименьшую среднеквадратическую ошибку.

Список источников

1. Цифровая обработка сигналов: учебное пособие / В.А. Сериков, В.Р. Луцив; С.-Петерб. гос. ун-т аэрокосм. приборостроения. - СПб: Изд-во ГУАП, 2014. – 110 с. [библиотечный шифр 621.391 С32]

2. Лекция 10: методы аппроксимации и их применение в ЦОС. О.О. Жаринов, ГУАП, 30 сентября 2020г. [Электронный ресурс]. – Режим доступа: https://bbb4.guap.ru/playback/presentation/2.0/playback.html?meetingId=e635832d70e6ac16001b57f22fa329d56c641a9b-1601467330609 – Загл. с экрана. (Дата обращения 3.11.2020г.).

3. Метод наименьших квадратов. [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BD%D0%B0%D0%B8%D0%BC%D0%B5%D0%BD%D1%8C%D1%88%D0%B8%D1%85_%D0%BA%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BE%D0%B2 – Загл. с экрана. (Дата обращения 3.11.2020г.).

Приложение А – Программа

```
def plot_signal_difference(signal_delta, signal_stdev):
    """
    Функция выводит график разности между сигналами и среднеквадратическую
    ошибку фильтрации

    :signal_delta: Разность сигнала
    :signal_stdev: Среднеквадратическая ошибка сигнала
    """
    import matplotlib.pyplot as plt

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(signal_delta)
    ax.set_title("Разность зашумлённого и аппроксимированного сигналов")
    ax.text(0, 0.5, "Среднеквадратическая ошибка аппроксимации равна " +
            "{}".format(round(signal_stdev, 3)), transform=ax.transAxes,
            fontsize=14)


def approximate_signal_with_basis(window_length, signal_for_smooth, basis):
    """
    Сглаживает сигнал методом наименьших квадратов с помощью скользящего окна

    :window_length: Длинна скользящего окна
    :signal_for_smooth: Сигнал для сглаживания
    :basis: базис
    :return: фильтрованный сигнал
    """
    import numpy as np

    approximated_signal = []
    for number in range(0, len(signal_for_smooth)):

        # Двигаем окно
        if number < int(window_length/2):
            cutted_signal = np.zeros(window_length)

            for signal_number in range(int(window_length/2) - number,
                                       window_length):
                cutted_signal[signal_number] = signal_for_smooth[
                        signal_number - int(window_length/2) + number
                        ]

        elif number > (len(signal_for_smooth) - int(window_length/2)
                       - (window_length % 2)):
            cutted_signal = np.zeros(window_length)
            for signal_number in range(0, int(window_length/2)
                                       + (len(signal_for_smooth) - number)):
                cutted_signal[signal_number] = signal_for_smooth[
                        signal_number + number - int(window_length/2)]
        else:
```

```
        cutted_signal = signal_for_smooth[
            number - int(window_length/2):
            number + int(window_length/2) + window_length % 2]
        cutted_signal = np.array(cutted_signal)

        # Находим коэффициент разложения
        transpose_basis = basis.transpose()
        coefficient_of_decomposition = np.matmul(np.linalg.inv(np.matmul(
            transpose_basis, basis)),
            np.matmul(transpose_basis, cutted_signal))

        # Осуществляем аппроксимацию на основе базисной функции
        approximated_signal.append(sum(
            np.multiply(coefficient_of_decomposition,
                        basis[int(window_length/2)
                              + window_length % 2])))

    return np.array(approximated_signal)


def least_squares_generate(window_length, basis_order_range):
    """Генерирует мнк полином с заданной длинной окна и порядком базиса"""
    import numpy as np

    moving_window_range = np.linspace(0, window_length - 1, window_length)

    least_squares_array = []
    for order in basis_order_range:
        least_squares_array.append(
                (moving_window_range/np.sqrt(window_length))**order)

    return np.array(least_squares_array).transpose()


def main():
    import numpy as np
    import matplotlib.pyplot as plt
    from statistics import stdev
    from numpy import log, pi, cos

    # Генерируем сигнал
    main_signal_length = 3000
    time = np.linspace(0, 2, main_signal_length)
    generated_signal = log(2 - cos(2 * pi * 3 * time))

    # Генерируем шумы
    gaussian_noise_signal = (generated_signal
                             + np.random.normal(0, 1, main_signal_length))

    # Задаём константные значения для базисной функции
    order_max = 3
    order_range = np.linspace(0, order_max - 1, order_max)
```

```
moving_window_max = 380

# Находим зависимость среднеквадратической ошибки от длинны окна и
# вычисляем индекс минимальной среднеквадратической ошибки.
stdev_array = []
plot_start = 280
window_range = range(plot_start, moving_window_max)
for window_number in window_range:
    # Вычисляем полином для конкретного размера окна
    polinominal = least_squares_generate(window_number, order_range)

    # Аппроксимируем сигнал полученным полиномом
    approximated_signal = approximate_signal_with_basis(
            window_number, gaussian_noise_signal, polinominal)

    # Вычисляем среднеквадратическую ошибку
    signal_delta = generated_signal - approximated_signal
    signal_stdev = stdev(signal_delta)/stdev(generated_signal)
    stdev_array.append(signal_stdev)

index_of_minimum_stdev = stdev_array.index(min(stdev_array)) + plot_start

# Рисуем график зависимости
plt.figure()
plt.xlabel("Порядки базиса")
plt.ylabel("Величина среднеквадратической ошибки")
plt.plot(window_range, stdev_array)
plt.text(plot_start, min(stdev_array), "Наименьшее значение " +
         "среднеквадратической ошибки = " +
         "{}".format(round(min(stdev_array), 3)), fontsize=12)
plt.text(plot_start, min(stdev_array) + 0.001, "Порядок наименьшего" +
         "значения среднеквадратической ошибки = " +
         "{}".format(index_of_minimum_stdev), fontsize=12)

# Аппроксимируем наилучшей длинной окна, которую мы узнали ранее
# Снова формируем полином
polinominal = least_squares_generate(index_of_minimum_stdev, order_range)

# Аппроксимируем
approximated_signal = approximate_signal_with_basis(index_of_minimum_stdev,
                                                    gaussian_noise_signal,
                                                    polinominal)
signal_delta = generated_signal - approximated_signal
signal_stdev = stdev(signal_delta)/stdev(generated_signal)

# Рисуем графики сигналов
plt.figure()
plt.title("Сравнение зашумлённого сигнала и фильтрованного аппроксимацией")
plt.xlabel("Время")
plt.ylabel("Амплитуда")
plt.plot(time, gaussian_noise_signal, label="Зашумлённый сигнал")
plt.plot(time, approximated_signal, label="Аппроксимированный сигнал")
```

```
    plt.legend()

    # Выводим график разности сигналов
    plot_signal_difference(signal_delta, signal_stdev)

    plt.show()


if __name__ == "__main__":
    main()
```