ГУАП

КАФЕДРА № 41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

ИЗУЧЕНИЕ МЕТОДОВ СГЛАЖИВАЮЩЕЙ ФИЛЬТРАЦИИ НА ОСНОВЕ АППРОКСИМАЦИИ

по курсу: методы и устройства цифровой обработки сигналов

Вариант 3

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР 4711	22.10.2020	Хасанов Б.Р.
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2020

1 Цель работы

Изучить основы метода сглаживающей фильтрации зашумленных сигналов на основе методов аппроксимации.

2 Краткие теоретические сведения

Задача сглаживания — это, по сути, задача фильтрации сигнала от скачкообразных (ступенчатых) изменений. Считается, что полезный сигнал их не содержит. Ступенчатый сигнал за счёт множества резких, но небольших по амплитуде, перепадов уровня содержит высокочастотные составляющие, которых нет в сглаженном сигнале.[3]

В рамках работы аппроксимация осуществляется с помощью базиса лежандра. Полином Лежандра (Legendre polynomials) — многочлен, который в наименьшей степени отклоняется от нуля в смысле среднего квадратического. Образует ортогональную систему многочленов на отрезке [-1;1] и рассчитывается по формуле

$$P_k(t) = \frac{1}{2^k k!}\frac{d^k}{dt^k}\left(t^2 - 1\right)^k \quad (3)$$

где t выражается через n по формуле:

$$t = \frac{2n}{N-1} - 1.$$

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 scipy.special import legendre
from pprint import pprint
```

Затем, генерируем сигнал данный нам по заданию с помощью функции signal_generate, она выполняет простейшую операцию, её полный код можно увидеть в приложении А. Результат генерации сигнала на рисунке 3.1.

```
n = np.linspace(0, 2, N)
signal_function = 'log(2 - cos(2 * pi * 3 * time))'
generated_signal = signal_generate(n, signal_function)
```

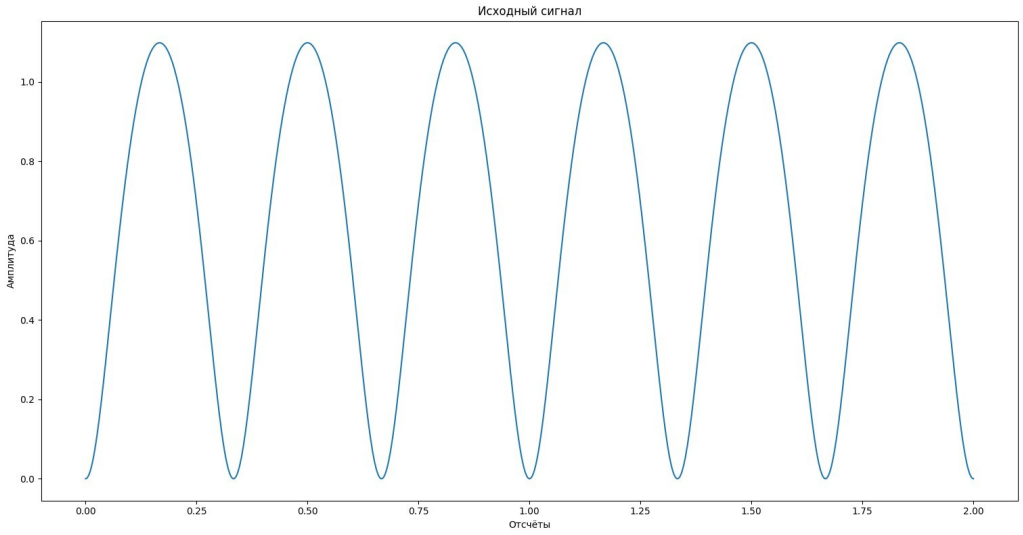

Рисунок 3.1 — Исходный сигнал

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, N)

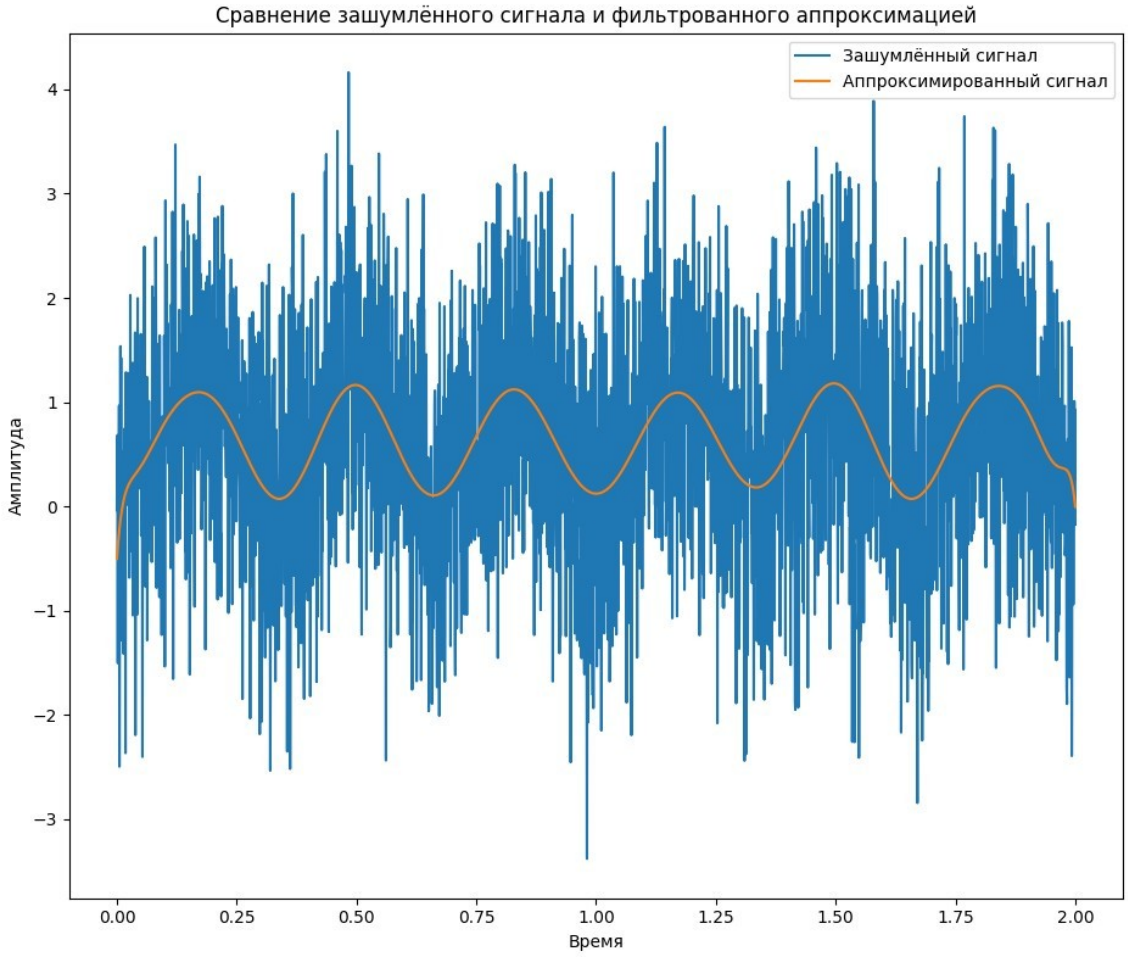

Рисунок 3.2 – Сравнение зашумлённого сигнала и сигнала фильтрованного аппроксимацией, при порядке базиса равному 21

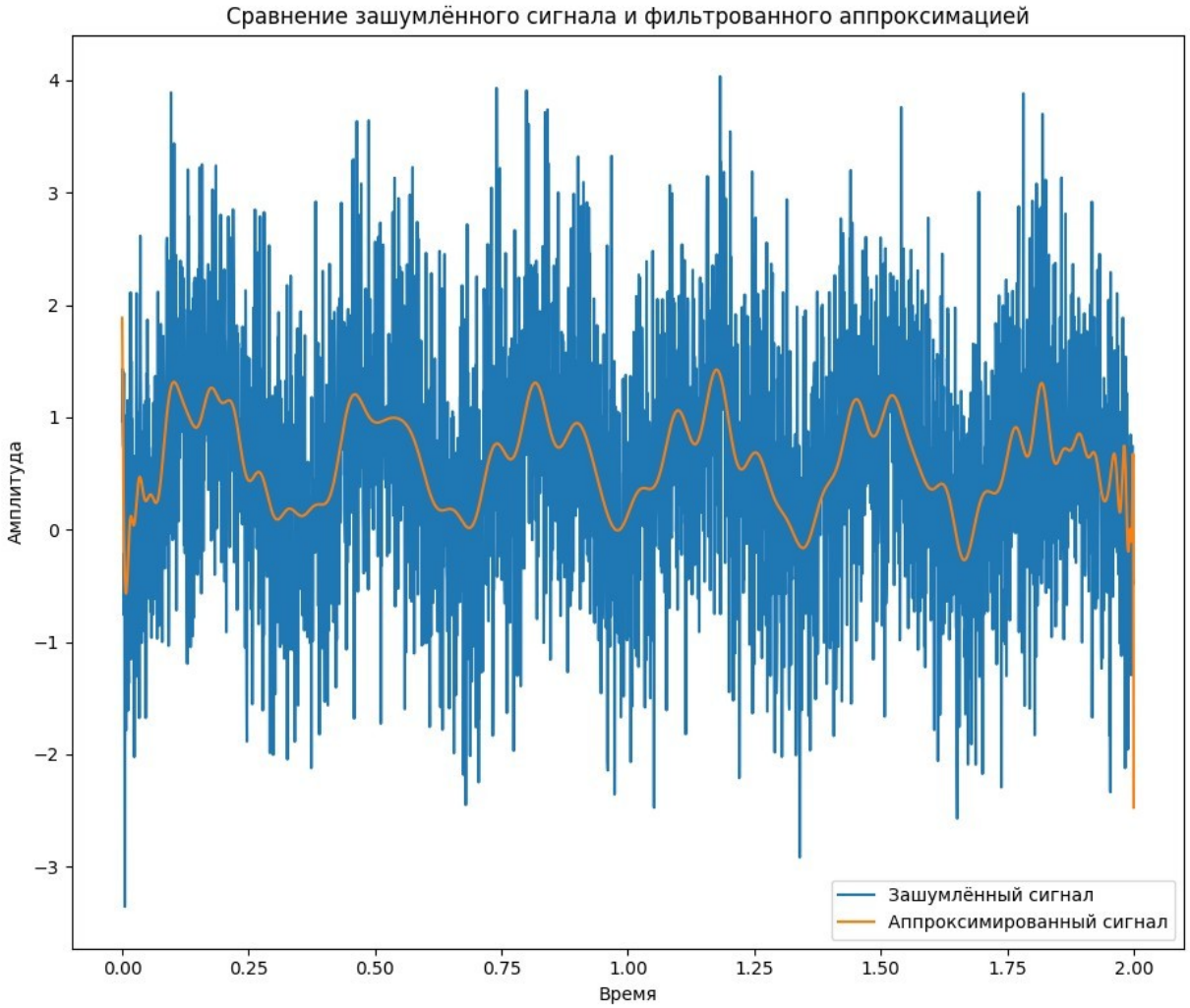

Рисунок 3.3 – Сравнение зашумлённого сигнала и сигнала фильтрованного аппроксимацией, при порядке базиса равному 90

Генерируем полином лежандра порядка M. Берём порядок с запасом, чтобы далее вычислить порядок с наименьшим среднеквадратическим отклонением. При генерации используется функция legandre из библиотеки scipy.special.

```
M = int(N/30)
numbers_range = np.linspace(-1, 1, N)
generated_legendre = []
for order in range(1, M):
    legendre_polynomial = legendre(order - 1)
    generated_legendre.append(legendre_polynomial(numbers_range))
generated_legendre = np.array(generated_legendre)
```

Следующие строчки рисуют графики базисных функций порядками от 1 до 7. Это делается для того, чтобы визуально оценить правильность базисной функции. Результат на графике 3.4

```
plt.figure()
for order_number in range(0, 6):
```

```
    plt.plot(numbers_range, generated_legendre[order_number][:], label="P{}
(x)".format(order_number))
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("Pn(x)")
    plt.title("Полином лежандра")
```

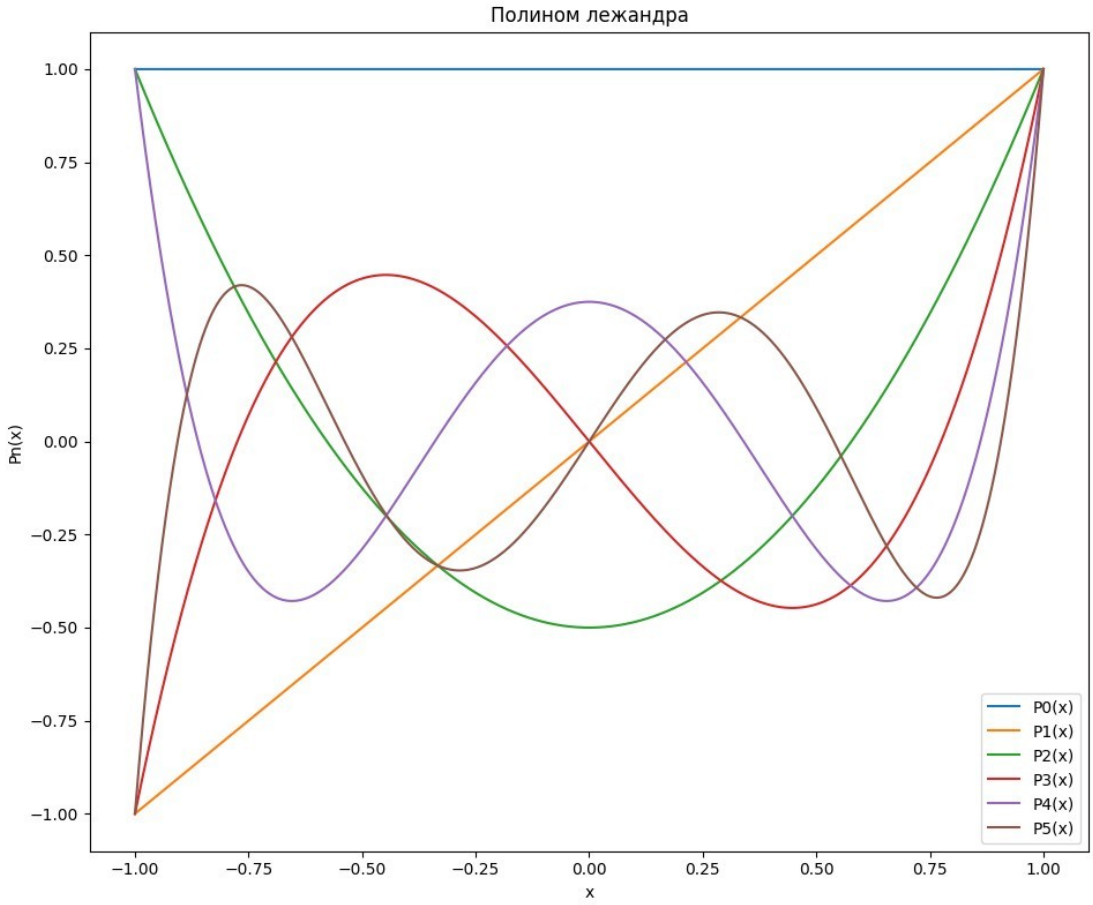

Рисунок 3.4 – Полином лежандра

Проверяем условие ортогональности для базиса Лежандра следующими строчками. Результат на рисунке 3.5. При проверке используется функция транспонирования матрицы transponse библиотеки numpy и matrix_multiplay, задача которого довольно заурядная – перемножить матрицы. Полный текст последней можно найти в приложении.

```
    transponated_generated_legendre = np.transpose(generated_legendre[:22])
                orthogonality_check = matrix_multiply(generated_legendre[:22],
transponated_generated_legendre)
    print(orthogonality_check)
```

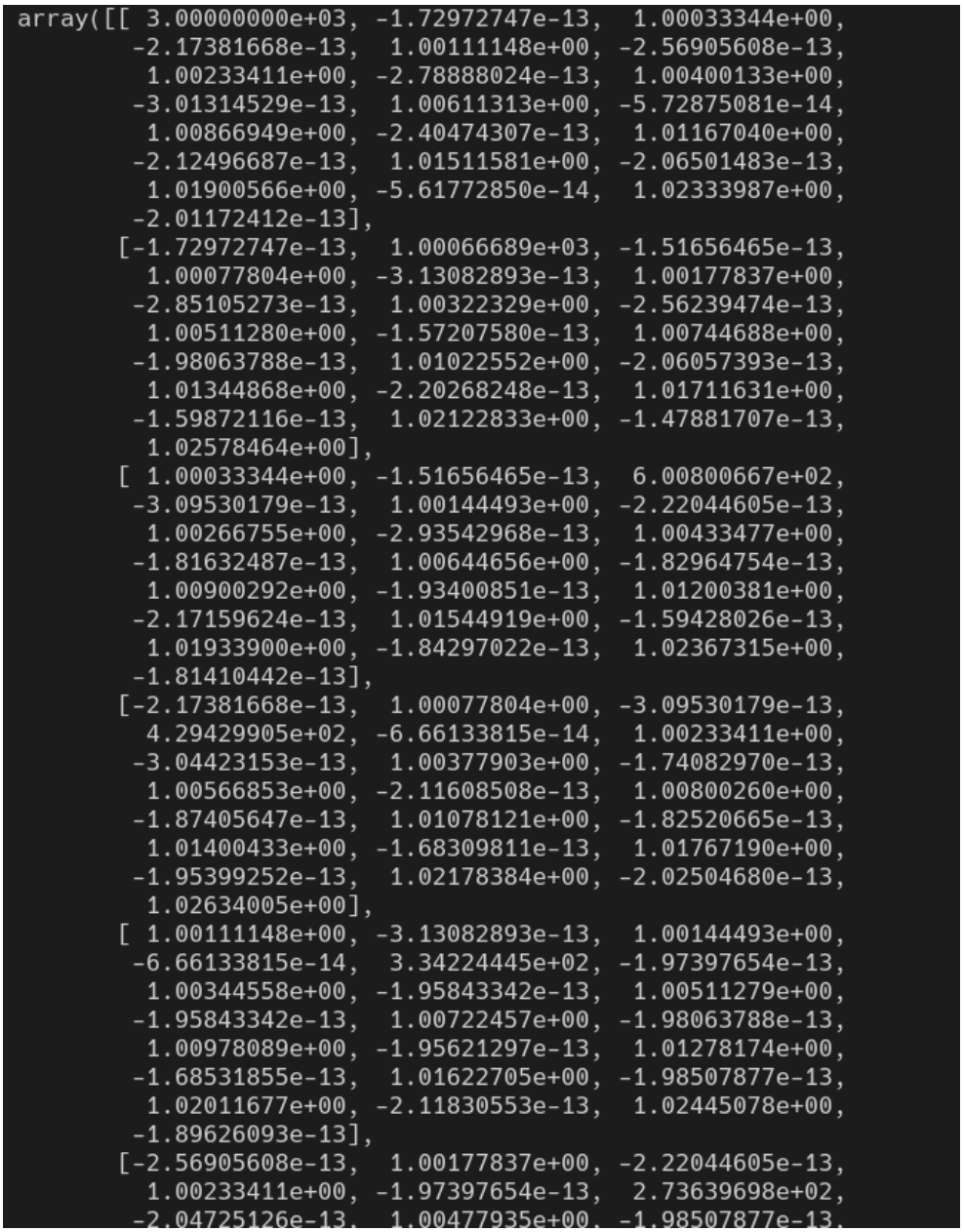

Рисунок 3.5 – Результат проверки на ортогональность

Рассчитываем массив среднеквадратических отклонений при порядках базиса от 10 до 300 и находим порядок с наименьшим среднеквадратическим отклонением.

```
stdev_array = []
plot_start = 10
orders_range = range(plot_start, len(generated_legendre))
for order_number in orders_range:
    approximated_signal = approximate_signal_with_basis(order_number, gaussian_noise_signal, generated_legendre)
    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
```

При этом используется функция approximate_signal_with_basis. Которая содержит в себе следующие строчки:

```
import numpy as np

coefficient_of_decomposition = []
for order_number in range(0, approximate_order):
                                    coefficient_of_decomposition.append(sum(signal_for_smooth * non_orthogonal_basis[order_number])/ \
                                                                        sum(non_orthogonal_basis[order_number] * non_orthogonal_basis[order_number]))

# Осуществляем аппроксимацию на основе базисной функции
approximated_signal = []
for number in range(0, N):
                                  approximated_signal.append(sum(coefficient_of_decomposition * non_orthogonal_basis[:approximate_order,number]))

return np.array(approximated_signal)
```

Первым блоком кода находиться коэффициенты декомпозиции, а вторым уже осуществляется аппроксимация на основе базисной функции.

Выводим график зависимости среднеквадратической ошибки от порядка базиса, на этом же графике выводим порядок базиса с наименьшим среднеквадратическим значением и само наименьшее среднеквадратическое значение. Результат на рисунке 3.6

```
plt.figure()
plt.xlabel("Порядки базиса")
plt.ylabel("Величина среднеквадратической ошибки")
plt.plot(orders_range, stdev_array)
plt.text(plot_start, min(stdev_array), "Наименьшее значение среднеквадратической ошибки = {}".format(round(min(stdev_array), 3)), fontsize=14)
        plt.text(plot_start, min(stdev_array) + 0.1, "Порядок наименьшего значения среднеквадратической ошибки = {}".format(index_of_minimum_stdev), fontsize=14)
```

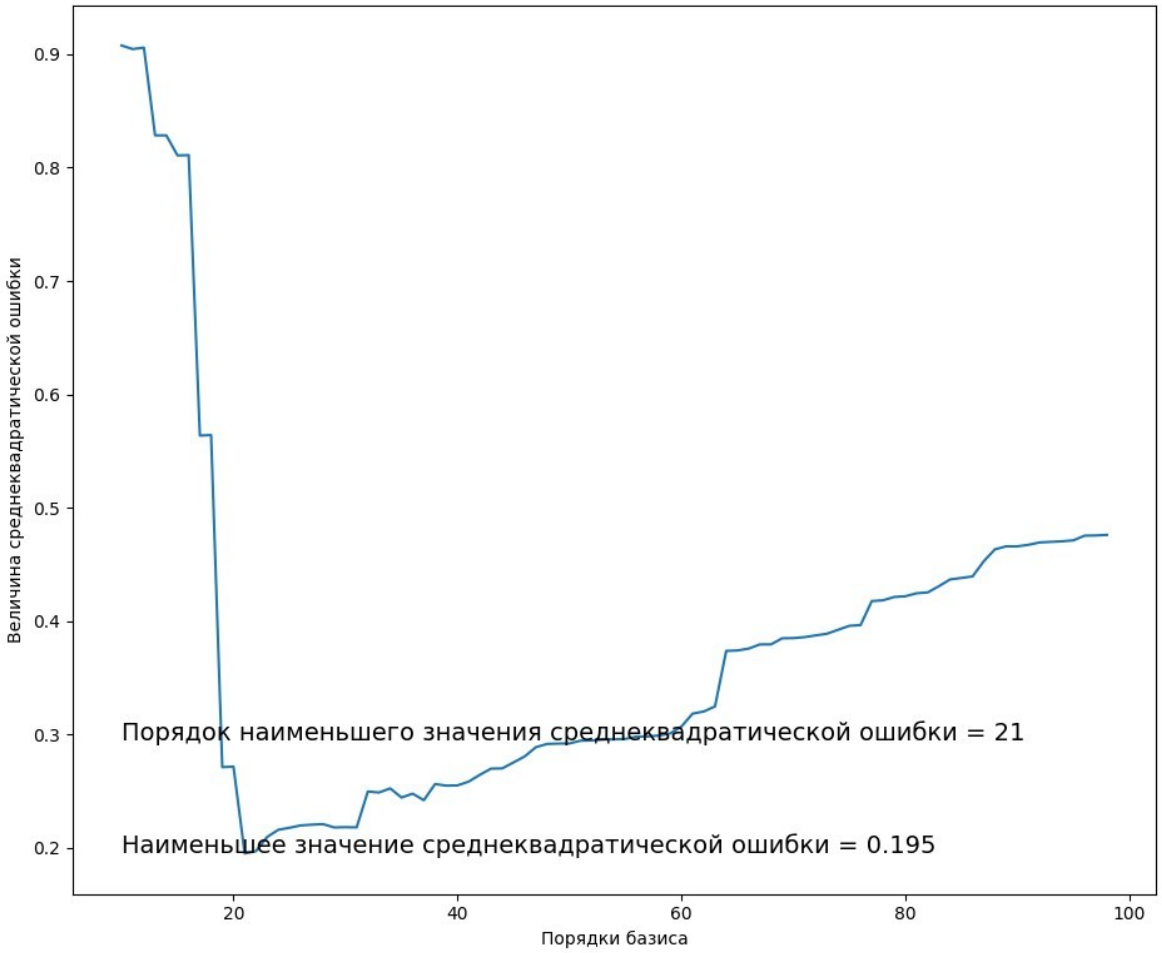

Рисунок 3.6 – Зависимость среднеквадратической ошибки от порядка базиса

Аппроксимируем зашумлённый сигнал базисом с оптимальным порядком, который мы нашли ранее. Результат аппроксимации обозначен на рисунке 3.2 как «аппроксимированный сигнал». Так же, находим среднеквадратическую ошибку сигнала.

```
approximated_signal = approximate_signal_with_basis(index_of_minimum_stdev, gaussian_noise_signal, generated_legendre)
signal_delta = generated_signal - approximated_signal
signal_stdev = stdev(signal_delta)/stdev(generated_signal)
```

Выводим сравнение зашумлённого и аппроксимированного сигнала. Результат на рисунке 3.2

```
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()
```

Следующей строчкой выводим разность между сгенерированным сигналом и аппроксимированным. Результат на рисунке 3.7

plot_signal_difference(signal_delta, signal_stdev)

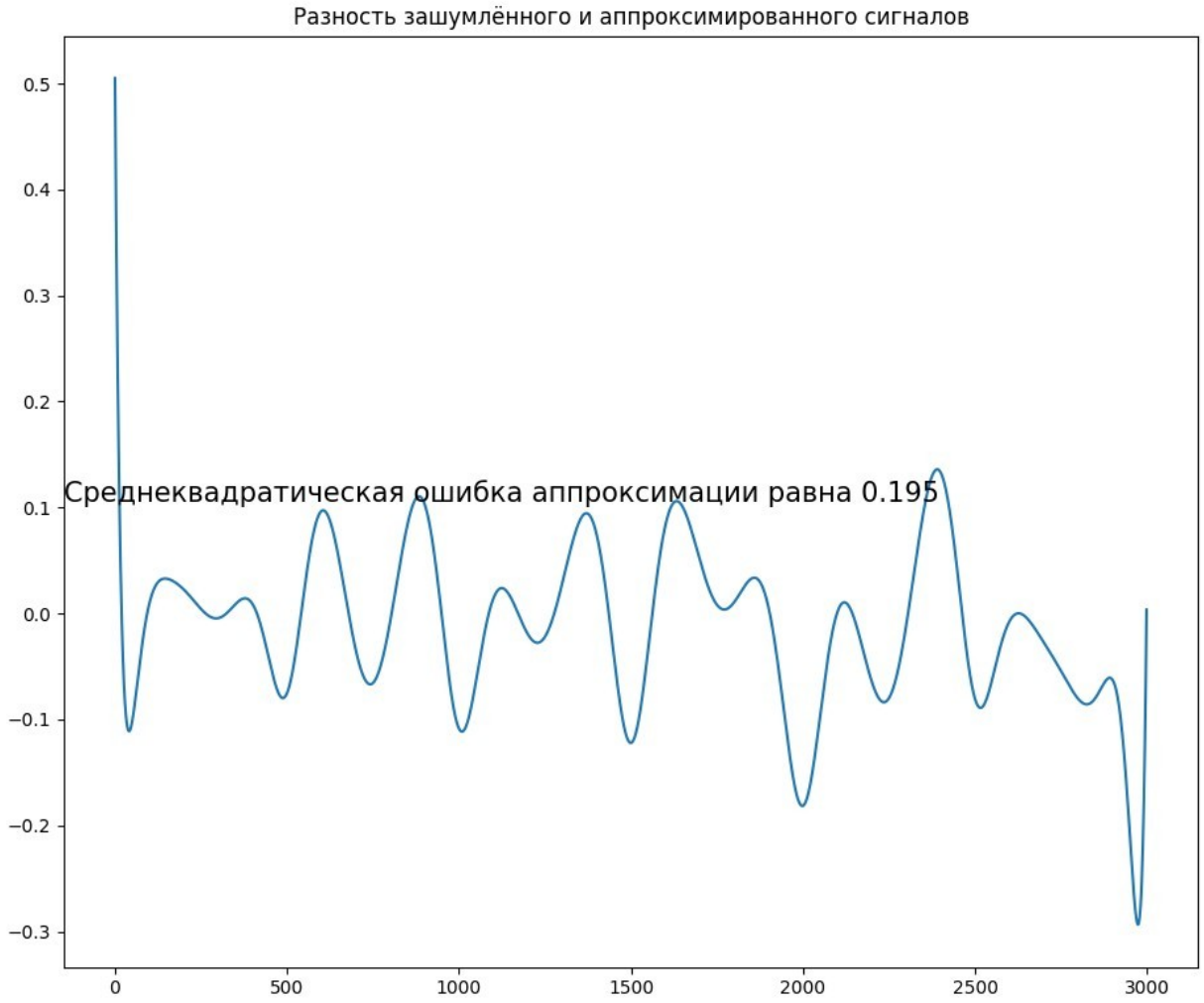

Рисунок 3.7 – Разность зашумлённого и аппроксимированного сигналов при порядке базиса равному 21

Функция plot_signal_difference просто выводит график разности и среднеквадратическую ошибку аппроксимированного сигнала. Её полный код приведён в приложении.

4 Выводы

В рамках данной работы мною было проведена сглаживающая фильтрация зашумлённого сигнала с помощью базиса Лежандра. На практике убедился в том, что данный метод фильтрации относительно хорошо сглаживает сигнал даже при дисперсии равной 1. После аппроксимации базисом Лежандра с оптимальным порядком сигнал прекрасно угадывается, это видно, если сравнить график на рисунке 3.1 и график обозначенный как «аппроксимированный сигнал» на рисунке 3.2. Так же, выяснил, что чем выше порядок базисной функции Лежанда, тем больше аппроксимированный сигнал походит на сигнал с шумом. Это подтверждается тем, что на рисунке 3.6 видно как растёт среднеквадратическая ошибка, а если сравнивать рисунки 3.2 и 3.3, то можно заметить, что аппроксимированный сигнал полученный с помощью сглаживания базисом 90-ого порядка явно больше походит на шум, чем сигнал аппроксимированный базисом 21-ого порядка.

Убедился в том, что базис Лежандра не ортогонален, с помощью перемножения базиса Лежандра на транспонированный базис Лежандра, результат показан на рисунке 3.4. Ортогональный базис имел бы единицы по диагонали массива, но на рисунке видно, что это не так, следовательно, базис не ортогонален.

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 24.10.2020г.).

3. Сглаживание цифровых сигналов. [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/184728/ – Загл. с экрана. (Дата обращения 24.10.2020г.).

Приложение А – Программа

```
# Const
N = 3000

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=15)


def signal_generate(time, function):
    """
    Генерирует сигнал с заданными параметрами

    :time: Отсчёты сигнала
    :function: Функция по которой строиться график
    :returns: Сигнал с заданными параметрами
    """
    from numpy import log, pi, cos

    generated_signal = eval(function)

    return generated_signal


def matrix_multiply(first_matrix, second_matrix):
    """
    Перемножает матрицы

    :first_matrix: первая матрица
    :second_matrix: вторая матрица
    :return: Результат перемножения
    """
    import numpy as np

    multiply_result = np.zeros((len(first_matrix), len(second_matrix[0])))
    for row in range(0, len(first_matrix)):
        for column in range(0, len(second_matrix[0])):
            for counter in range(0, len(second_matrix)):
```

```
            multiply_result[row][column] += first_matrix[row][counter] * second_matrix[counter]
[column]

    return multiply_result


def approximate_signal_with_basis(approximate_order, signal_for_smooth, non_orthogonal_basis):
    """
    Сглаживает сигнал неортогональным базисом определённого порядка

    :approximate_order: Порядок базиса
    :signal_for_smooth: Сигнал для сглаживания
    :non_orthogonal_basis: Неортогональный базис
    :return: фильтрованный сигнал
    """
    import numpy as np

    coefficient_of_decomposition = []
    for order_number in range(0, approximate_order):
        coefficient_of_decomposition.append(sum(signal_for_smooth *
non_orthogonal_basis[order_number])/ \
                                    sum(non_orthogonal_basis[order_number] *
non_orthogonal_basis[order_number]))

    # Осуществляем аппроксимацию на основе базисной функции
    approximated_signal = []
    for number in range(0, N):
        approximated_signal.append(sum(coefficient_of_decomposition *
non_orthogonal_basis[:approximate_order,number]))

    return np.array(approximated_signal)


def main():
    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.special import legendre
    from pprint import pprint
    from statistics import stdev

    # Генерируем сигнал
    time = np.linspace(0, 2, N)
    signal_function = 'log(2 - cos(2 * pi * 3 * time))'
    generated_signal = signal_generate(time, signal_function)

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

    # Генерируем полином Лежандра
    M = int(N/30)
    numbers_range = np.linspace(-1, 1, N)
    generated_legendre = []
```

```
    for order in range(1, M):
        legendre_polynomial = legendre(order - 1)
        generated_legendre.append(legendre_polynomial(numbers_range))
    generated_legendre = np.array(generated_legendre)

    # Выводим на график несколько базисных функций
    plt.figure()
    for order_number in range(0, 6):
        plt.plot(numbers_range, generated_legendre[order_number][:], label="P{}
(x)".format(order_number))
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("Pn(x)")
    plt.title("Полином лежандра")

    # Проверяем условие ортоганальности
    transponated_generated_legendre = np.transpose(generated_legendre[:22])
    orthogonality_check = matrix_multiply(generated_legendre[:22],
transponated_generated_legendre)
    print(orthogonality_check)

    # Находим зависимость среднеквадратической ошибки от порядка базиса и вычисляем
индекс минимальной
    # среднеквадратической ошибки
    stdev_array = []
    plot_start = 10
    orders_range = range(plot_start, len(generated_legendre))
    for order_number in orders_range:
        approximated_signal = approximate_signal_with_basis(order_number, gaussian_noise_signal,
generated_legendre)
        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(orders_range, stdev_array)
    plt.text(plot_start, min(stdev_array), "Наименьшее значение среднеквадратической ошибки =
{}".format(round(min(stdev_array), 3)), fontsize=14)
    plt.text(plot_start, min(stdev_array) + 0.1, "Порядок наименьшего значения
среднеквадратической ошибки = {}".format(index_of_minimum_stdev), fontsize=14)

    # Т. к. матрица не ортогональна, то коэф. параметрической модели сигнала будем искать
соответственно
    approximated_signal = approximate_signal_with_basis(index_of_minimum_stdev,
gaussian_noise_signal, generated_legendre)
    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 и находим среднеквадратическую ошибку
    plot_signal_difference(signal_delta, signal_stdev)

    plt.show()


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