ГУАП

КАФЕДРА № 41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

ИЗУЧЕНИЕ МЕТОДОВ ФИЛЬТРАЦИИ НА ОСНОВЕ ИНТЕРПОЛЯЦИИ

по курсу: методы и устройства цифровой обработки сигналов

Вариант 4

РАБОТУ ВЫПОЛНИЛ

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

Санкт-Петербург 2020

1 Цель работы

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

2 Краткие теоретические сведения

Интерполяцией называется увеличение количества отсчётов сигнала в единицу времени. Отношение количества новых отсчётов сигнала в единицу времени по отношению к предыдущему количеству отсчётов сигнала называется коэффициентом интерполяции. Обычно это отношение выбирается целым числом.

При увеличении частоты отсчётов сигнала в соответствии с теоремой Котельникова расширяется полоса частот, описываемых этими отсчетами. Это означает, что в новую полосу частот попадает несколько частотных образов первоначального варианта сигнала. При интерполяции необходимо выбрать нужный частотный образ. Обычно выбирается полоса частот от 0 до fв.

Задача выбора необходимого частотного образа решается при помощи цифрового фильтра. Такой фильтр называется интерполирующим. Именно этот фильтр вычисляет значения сигнала в точках между первоначальными отсчётами сигнала.[3]

3 Программа, в которой представлена последовательность и результаты обработки сигналов, с необходимыми комментариями

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.interpolate import CubicSpline
```

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 4.1.

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

Далее, генерируются гауссовский шум и негауссовский шум для объединения с сигналом с рисунка 4.1. В качестве негауссовского шума, по заданию, был выбран шум тюки с ε = 0.05. Результат обозначен на рисунках 4.2 и 4.4, как “Исходный сигнал”:

```
gaussian_noise_signal = generated_signal + np.random.normal(0, 1, N)
tuke_noise_signal = generated_signal + tuke_noise_generate(0.05, 0.25, 4.5, 0)
```

Причём, для шума тюки используется самописная функция tuke_noise_generate. Вот её содержание:

```
def tuke_noise_generate(epsilon, dispersion1, dispersion2, mean):
    """
    Генерирует шум по модели тюки

    :epsilon: Вероятность зашумления
    :dispersion1: Дисперсия несущего сигнала
    :dispersion2: Дисперсия зашумления
    :mean: Мат. ожидание
    :return: Массив значений по модели тюки
    """
    import numpy as np

    first_noise_signal = np.random.normal(mean, dispersion1, N)
    second_noise_signal = np.random.normal(mean, dispersion2, N)

    noise_probability = np.random.uniform(0, 1, N)

    noise = []
    for number in range(0, N):
        if noise_probability[number] <= (1 - epsilon):
            noise.append(first_noise_signal[number])
        else:
            noise.append(second_noise_signal[number])

    return np.array(noise)
```

Возвращаемся обратно в main(). Вычисляем положение M узлов для интерполяции с помощью функцию get_points, полученные точки отмечены на рисунках 4.2 и 4.4, как узловые точки исходного сигнала. Количество узлов M нам нужно будет менять по заданию, чтобы получить нименьшее значение среднеквадратической ошибки интерполяции:

```
    M = 120
    points_time = np.linspace(0, 2, M)
    window_length = int(N/M)
    points_of_tuke_noise_signal = get_points(tuke_noise_signal, M, window_length)
    points_of_gaussian_noise_signal = get_points(gaussian_noise_signal, M, window_length)
```

Функция get_points выглядит следующим образом:

```
def get_points(input_signal, points_number, step):
    """
    Получить узловые точки из сигнала с заданным шагом

    :input_signal: Сигнал для которого нужно выделить узловые точки
    :points_number: Количество точек
    :step: Шаг с которым будут браться точки
    :return: Массив узловых точек
    """
    import numpy as np
```

```
    calculated_points = []
    if len(input_signal) % step != 0:
        for segment in range(0, points_number - 1):
            calculated_points.append(np.median(input_signal[segment * step:(segment + 1) * step]))
        calculated_points.append(np.median(input_signal[(points_number - 1) * step:]))
    else:
        for segment in range(0, points_number):
            calculated_points.append(np.median(input_signal[segment * step:(segment + 1) * step]))

    return np.array(calculated_points)
```

У алгоритма вычисления точек есть один нюанс: если количество отсчётов сигнала делиться на количество узловых точек без остатка, то мы просто последовательно вычисляем медианы на заданных диапазонах отсчётов сигнала, но вот если остаток есть, то вычисляются медианы для всех диапазонов кроме последнего, в качестве последнего диапазона берётся оставшееся количество отсчётов.

В main() следующими строчками проводим интерполяцию сигнала с гауссовским шумом с помощью сторонней функции CubicSpline, выводим интерполированный сигнал с помощью функции interpolation_signal_plot (результат работы которой показан на рисунке 4.2), вычисляем среднеквадратическую ошибку фильтрации и выводим её вместе с графиком разности между сигналами с помощью функции plot_signal_difference (результат на рисунке 4.8):

```
gaussian_noise_signal_interpolation = CubicSpline(points_time, points_of_gaussian_noise_signal)
    interpolation_signal_plot(gaussian_noise_signal, time, points_of_gaussian_noise_signal, points_time, gaussian_noise_signal_interpolation(time), "Гауссовского")
    plot_signal_difference(generated_signal, gaussian_noise_signal_interpolation(time), time)
```

Функция interpolation_signal_plot просто выводит графики, её содержание можно найти в приложении А, а вот функция plot_signal_difference более затейлива:

```
def plot_signal_difference(main_signal, main_like_signal, counts):
    """
    Функция выводит график разности между сигналами и среднеквадратическую ошибку фильтрации

    :main_signal: Главный сигнал
    :main_like_signal: Сигнал, который хочет быть похожим на главный
    :counts: Отсчёты
    """
    import matplotlib.pyplot as plt
    from statistics import stdev

    signal_delta = main_signal - main_like_signal
    epsilon_spline = stdev(signal_delta)/stdev(main_signal)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(signal_delta)
```

ax.text(0, 0.5, "Среднеквадратическая ошибка интерполяции равна {}".format(epsilon_spline), transform=ax.transAxes, fontsize=15)

Она сначала выводит график разности, а затем рассчитывает среднеквадратическую ошибку и вставляет её в график

Проделываем всё то же самое для сигнала с шумом тюки и получаем результаты на рисунках 4.4 и 4.10

```
tuke_noise_signal_interpolation = CubicSpline(points_time, points_of_tuke_noise_signal)
    interpolation_signal_plot(tuke_noise_signal, time, points_of_tuke_noise_signal, points_time, tuke_noise_signal_interpolation(time), "тюки")
plot_signal_difference(generated_signal, tuke_noise_signal_interpolation(time), time)
```

4 Полученные графики

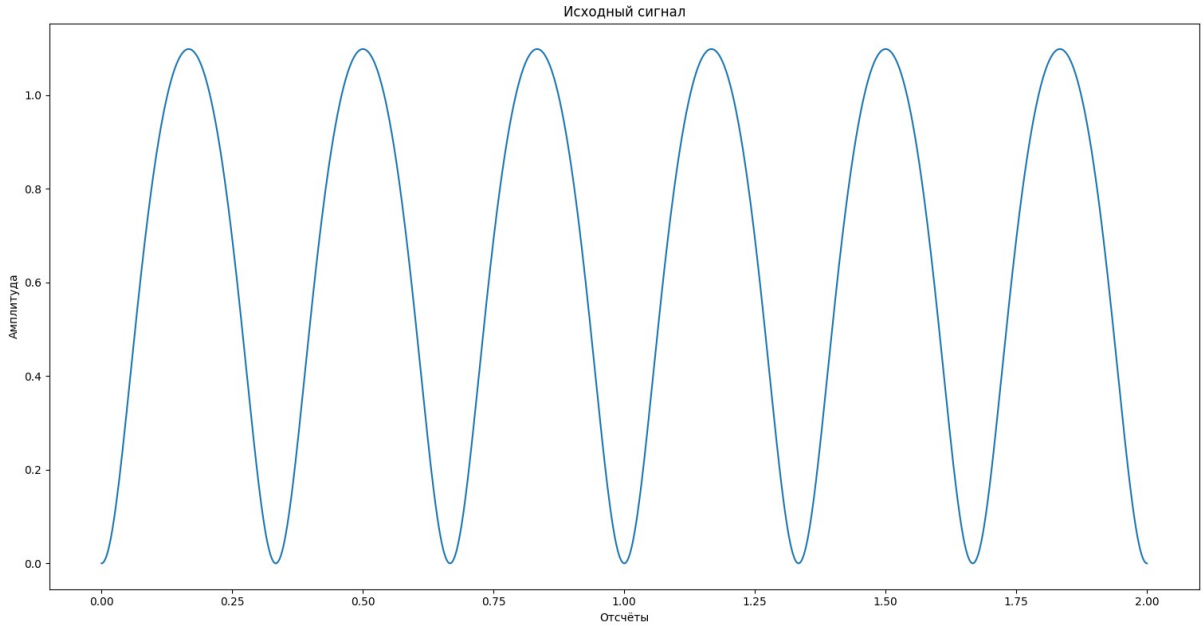

Рисунок 4.1 — Исходный сигнал

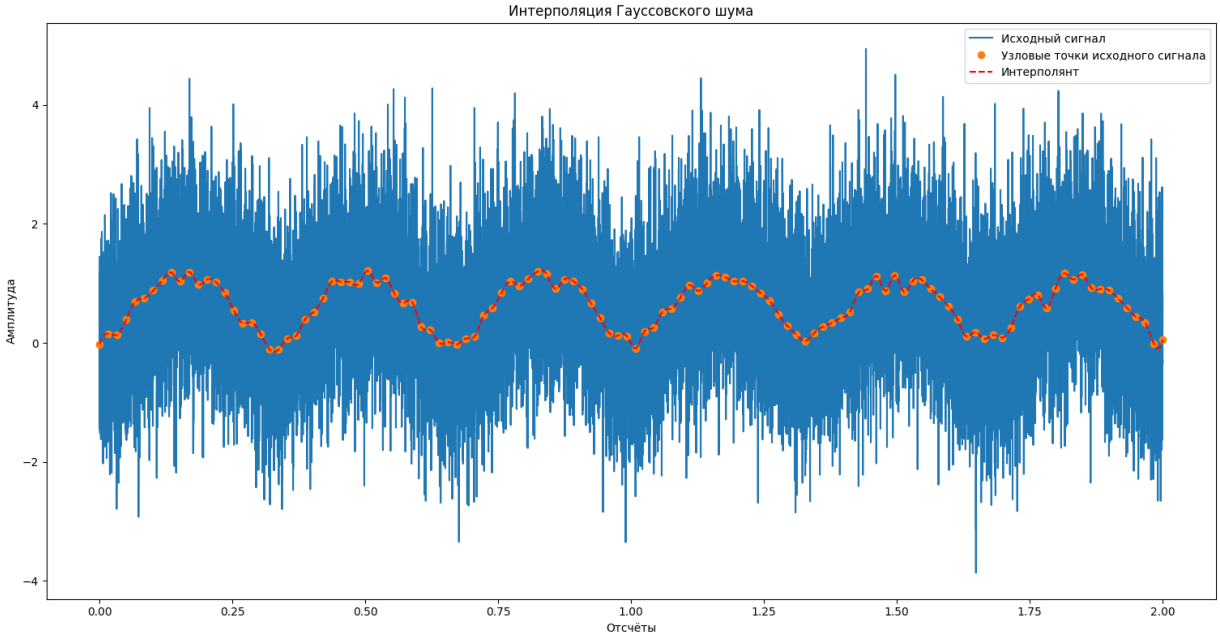

Рисунок 4.2 — Результат интерполяционной фильтрации сигнала с гауссовским шумом, с дисперсией 1 и количеством узлов 120 из 18000 отсчётов

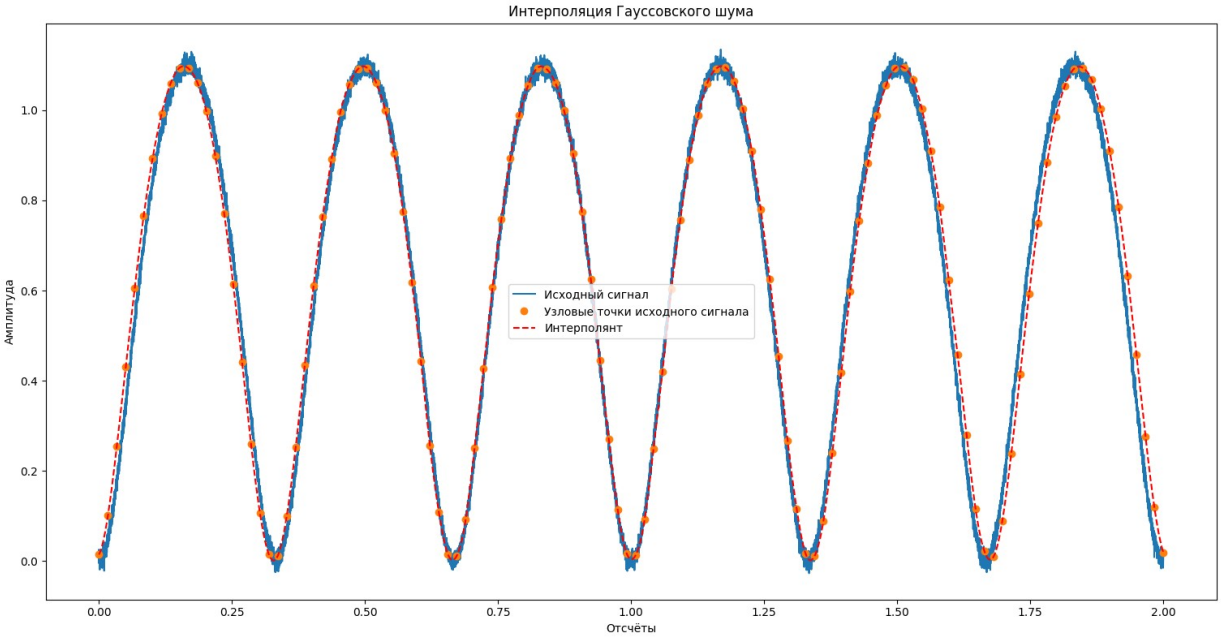

Рисунок 4.3 — Результат интерполяционной фильтрации сигнала с гауссовским шумом, с дисперсией 0.01 и количеством узлов 120 из 18000 отсчётов

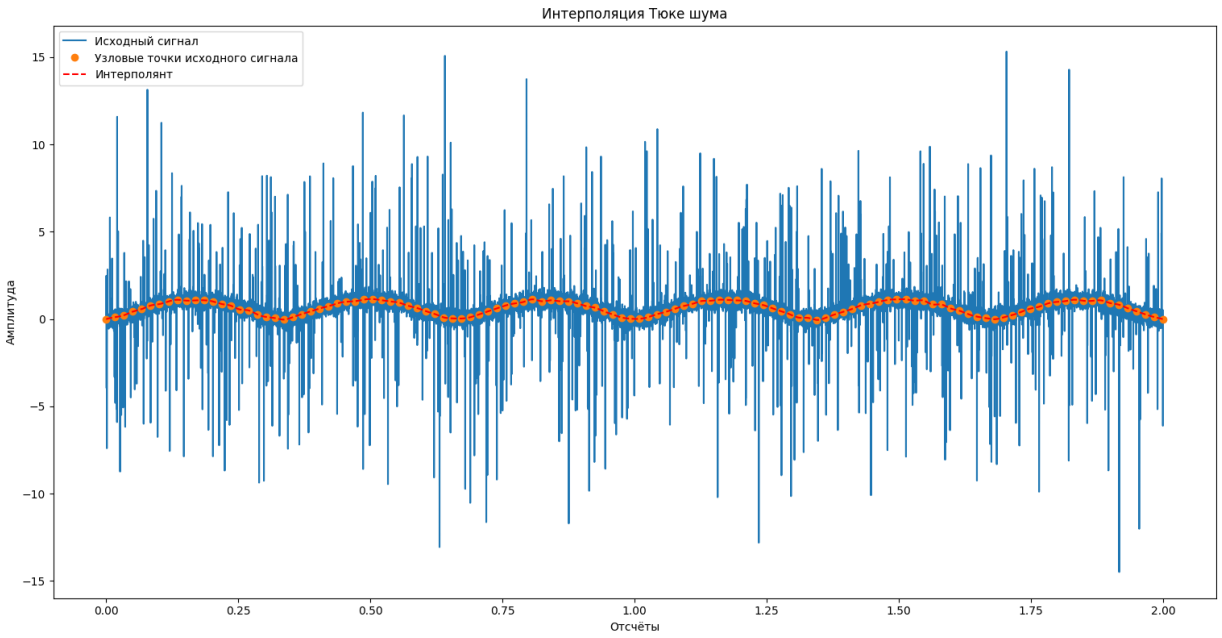

Рисунок 4.4 – Результат интерполяционной фильтрации сигнала с гауссовским шумом, с дисперсией 1 и количеством узлов 120 из 18000 отсчётов

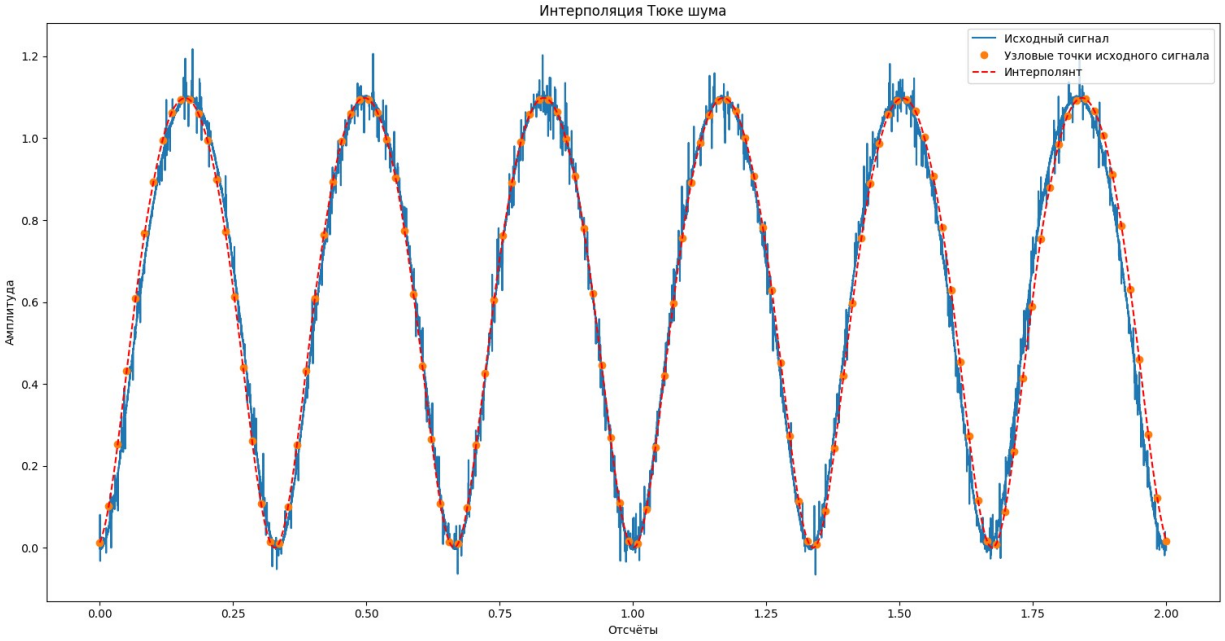

Рисунок 4.5 – Результат интерполяционной фильтрации сигнала с шумом тюки, с дисперсией 0.01 и количеством узлов 120 из 18000 отсчётов

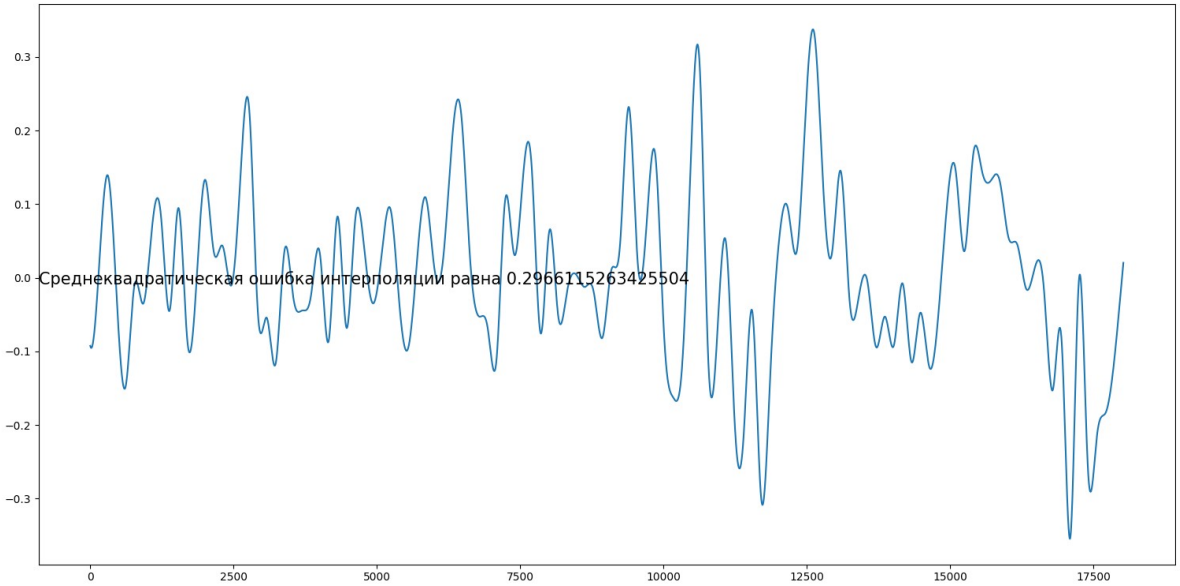

Рисунок 4.6 – График разности между заданной функцией и интерполянтом сигнала с гауссовским шумом, при количестве точек 118. Слева по середине выведена среднеквадратическая ошибка интерполяции

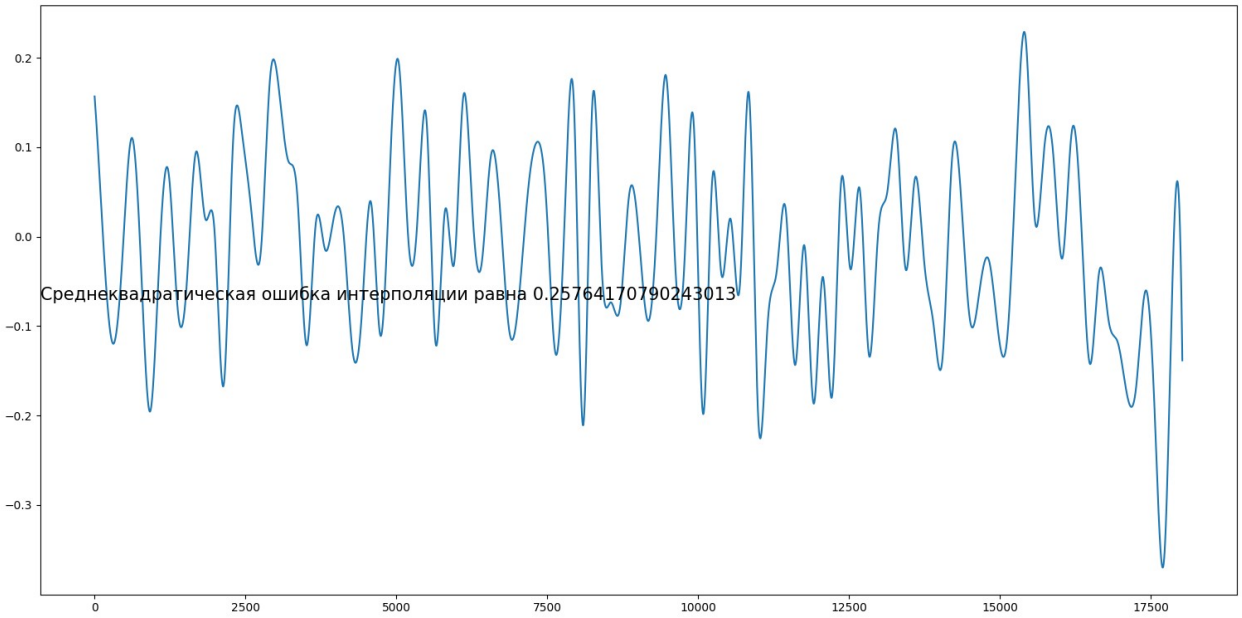

Рисунок 4.7 – График разности между заданной функцией и интерполянтом сигнала с гауссовским шумом, при количестве точек 119. Слева по середине выведена среднеквадратическая ошибка интерполяции

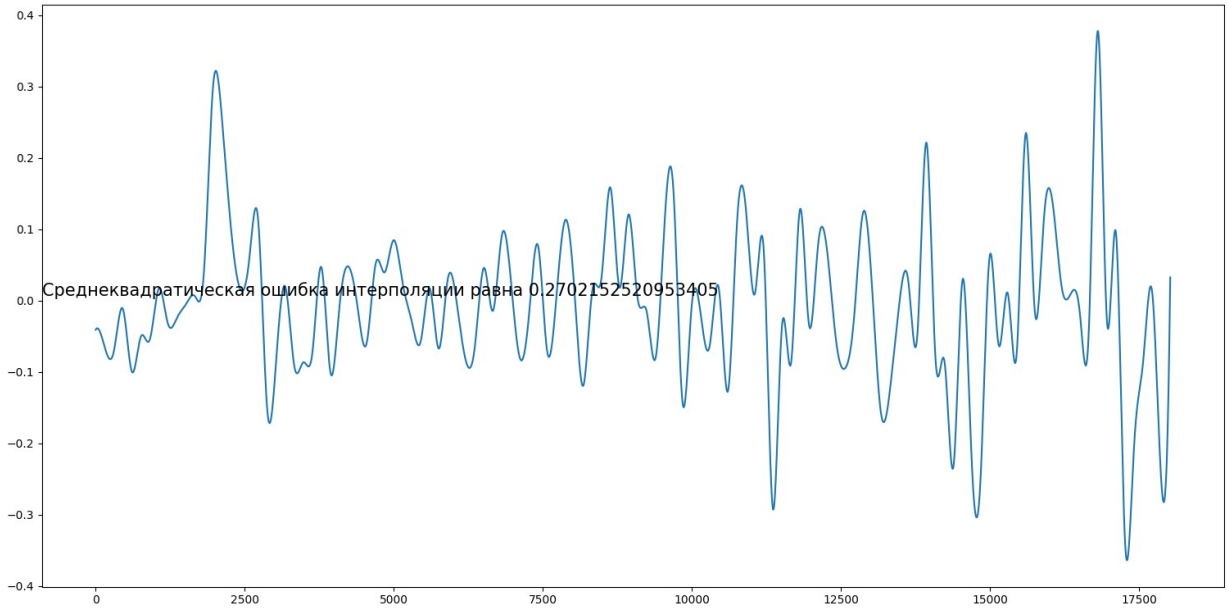

Рисунок 4.8 – График разности между заданной функцией и интерполянтом сигнала с гауссовским шумом, при количестве точек 120. Слева по середине выведена среднеквадратическая ошибка интерполяции

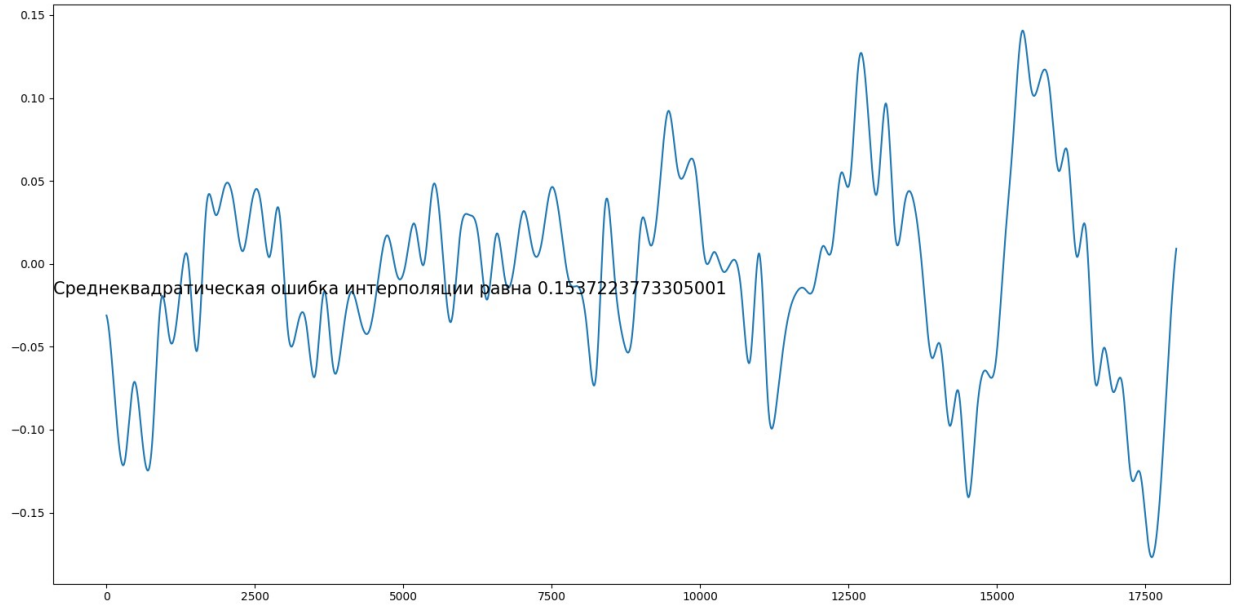

Рисунок 4.9 – График разности между заданной функцией и интерполянтом сигнала с шумом тюки, при количестве точек 119. Слева по середине выведена среднеквадратическая ошибка интерполяции

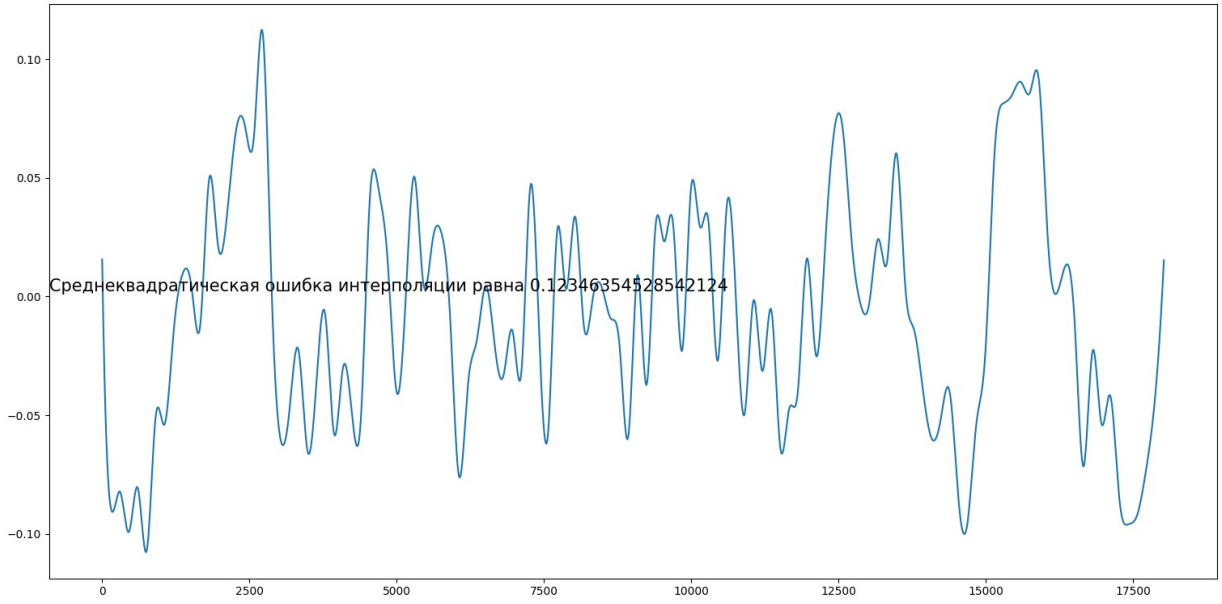

Рисунок 4.10 – График разности между заданной функцией и интерполянтом сигнала с шумом тюки, при количестве точек 120. Слева по середине выведена среднеквадратическая ошибка интерполяции

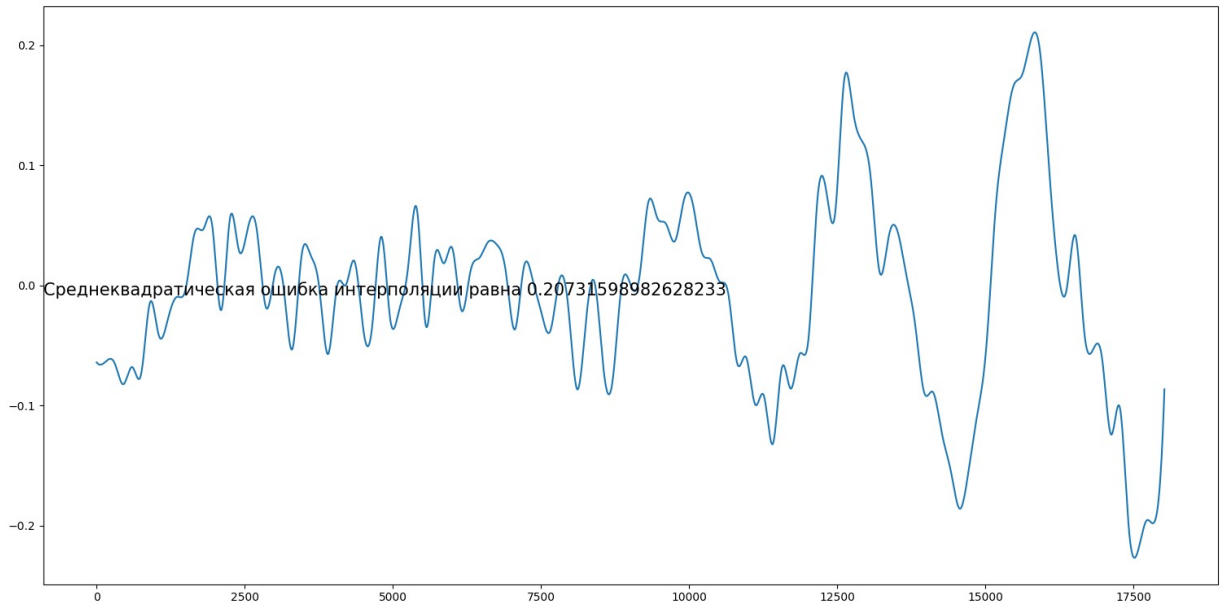

Рисунок 4.11 – График разности между заданной функцией и интерполянтом сигнала с шумом тюки, при количестве точек 121. Слева по середине выведена среднеквадратическая ошибка интерполяции

4 Выводы

В рамках данной работы мной была проведена фильтрация зашумлённого гауссовской и негауссовской помехой сигнала путём кубической интерполяции по узлам. Узлы интерполяции были выбраны с помощью вычисления медианы равных, по количеству отсчётов, выборок зашумлённых сигналов. На практике убедился в возможности фильтрации сигнала данным способом, получая при этом относительно небольшую среднеквадратическую ошибку (~0.3 для сигнала с гауссовским шумом и ~0.12 для негауссовского). При дисперсии шума равной единице сигнал, хоть и угадывался, но не был отфильтрован хорошо, это видно если сравнивать интерполянт на рисунке 4.2 и исходный сигнал на рисунке 4.1. При уменьшении дисперсии до 0.01 сигнал фильтруется очень хорошо, при этом наблюдается небольшое растяжение интерполянта сигнала с гауссовским шумом в стороны, относительно исходного сигнала, это показано на рисунке 4.3, такой же эффект наблюдается и у негауссовского шума относительно исходного сигнала, это показано на рисунке 4.5.

Было найдено наиболее оптимальное количество узловых точек для интерполяции по сигналу с 18000 отсчётами – оно составило 119 точек для сигнала с гауссовским шумом (результаты для 118, 119, 120 точек показаны на рисунках 4.6, 4.7, 4.8 соответственно) и 120 точек для сигнала с негауссовским шумом (результаты для 119, 120 и 121 точек показаны на рисунках 4.9, 4.10, 4.11 соответственно)

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 9: Методы интерполяции и их применение в ЦОС. О.О. Жаринов, ГУАП, 9 сентября 2020г. [Электронный ресурс]. – Режим доступа: https://bbb4.guap.ru/playback/presentation/2.0/playback.html?meetingId=52ed67f6ad8cd06c8d3c56a487d54eb4466bbaa8-1599652620665. – Загл. с экрана. (Дата обращения 02.10.2020г.).

3. Интерполирующие цифровые фильтры. [Электронный ресурс]. – Режим доступа: https://digteh.ru/digital/Intrpltr.php – Загл. с экрана. (Дата обращения 02.10.2020г.).

Приложение А – Программа

```
# Const
N = 18000

def plot_signal_difference(main_signal, main_like_signal, counts):
    """
    Функция выводит график разности между сигналами и среднеквадратическую ошибку фильтрации

    :main_signal: Главный сигнал
    :main_like_signal: Сигнал, который хочет быть похожим на главный
    :counts: Отсчёты
    """
    import matplotlib.pyplot as plt
    from statistics import stdev

    signal_delta = main_signal - main_like_signal
    epsilon_spline = stdev(signal_delta)/stdev(main_signal)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(signal_delta)
    ax.text(0, 0.5, "Среднеквадратическая ошибка интерполяции равна {}".format(epsilon_spline), transform=ax.transAxes, fontsize=15)

def get_points(input_signal, points_number, step):
    """
    Получить узловые точки из сигнала с заданным шагом

    :input_signal: Сигнал для которого нужно выделить узловые точки
    :points_number: Количество точек
    :step: Шаг с которым будут браться точки
    :return: Массив узловых точек
    """
    import numpy as np

    calculated_points = []
    if len(input_signal) % step != 0:
        for segment in range(0, points_number - 1):
            calculated_points.append(np.median(input_signal[segment * step:(segment + 1) * step]))
        calculated_points.append(np.median(input_signal[(points_number - 1) * step:]))
    else:
        for segment in range(0, points_number):
            calculated_points.append(np.median(input_signal[segment * step:(segment + 1) * step]))

    return np.array(calculated_points)


def tuke_noise_generate(epsilon, dispersion1, dispersion2, mean):
    """
    Генерирует шум по модели тюки

    :epsilon: Вероятность зашумления
```

```
    :dispersion1: Дисперсия несущего сигнала
    :dispersion2: Дисперсия зашумления
    :mean: Мат. ожидание
    :return: Массив значений по модели тюки
    """
    import numpy as np

    first_noise_signal = np.random.normal(mean, dispersion1, N)
    second_noise_signal = np.random.normal(mean, dispersion2, N)

    noise_probability = np.random.uniform(0, 1, N)

    noise = []
    for number in range(0, N):
        if noise_probability[number] <= (1 - epsilon):
            noise.append(first_noise_signal[number])
        else:
            noise.append(second_noise_signal[number])

    return np.array(noise)


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 interpolation_signal_plot(high_discretization_signal, high_discretization_numbers,
low_discretization_signal,
                              low_discretization_numbers, interpolated_signal, noise_type):
    """
    Строит три графика с заданной дискретизацией

    :high_discretization_signal: Сигнал с высокой частотой дискретизации
    :high_discretization_numbers: Отсчёты сигнала с высокой частотой дискретизации
    :low_discretization_signal: Сигнал с низкой частотой дискретизации
    :low_discretization_numbers: Отсчёты сигнала с низкой частотой дискретизации
    :input_signal: Сигнал с низкой частотой дискретизации прошедший интерполяцию
    :noise_type: Тип шума
    """
    import matplotlib.pyplot as plt
    import numpy as np
```

```
    plt.figure()
    plt.title("Интерполяция {} шума".format(noise_type))
    plt.xlabel('Отсчёты')
    plt.ylabel('Амплитуда')
    plt.plot(high_discretization_numbers, high_discretization_signal, label="Исходный сигнал")
    plt.plot(low_discretization_numbers, low_discretization_signal, "o", label="Узловые точки
исходного сигнала")
    plt.plot(high_discretization_numbers, interpolated_signal, "--",color="r", label="Интерполянт")
    plt.legend()


def main():
    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.interpolate import CubicSpline

    # 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)
    tuke_noise_signal = generated_signal + tuke_noise_generate(0.05, 0.25, 4.5, 0)

    # Генерируем массив узловых точек из медианных значений окон с шагом
    M = 120
    points_time = np.linspace(0, 2, M)
    window_length = int(N/M)
    points_of_tuke_noise_signal = get_points(tuke_noise_signal, M, window_length)
    points_of_gaussian_noise_signal = get_points(gaussian_noise_signal, M, window_length)

    # Кубическая интерполяция для сигнала с гауссовским шумом
    gaussian_noise_signal_interpolation = CubicSpline(points_time,
points_of_gaussian_noise_signal)
    interpolation_signal_plot(gaussian_noise_signal, time, points_of_gaussian_noise_signal,
points_time, gaussian_noise_signal_interpolation(time), "Гауссовского")
    plot_signal_difference(generated_signal, gaussian_noise_signal_interpolation(time), time)

    # Кубическая интерполяция для сигнала с шумом тюки
    tuke_noise_signal_interpolation = CubicSpline(points_time, points_of_tuke_noise_signal)
    interpolation_signal_plot(tuke_noise_signal, time, points_of_tuke_noise_signal, points_time,
tuke_noise_signal_interpolation(time), "тюки")
    plot_signal_difference(generated_signal, tuke_noise_signal_interpolation(time), time)

    plt.show()


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