ГУАП

КАФЕДРА № 41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №5

РАСЧЁТ И МОДЕЛИРОВАНИЕ ЦИФРОВОГО НЕРЕКУРСИВНОГО ФИЛЬТРА ВИННЕРА

по курсу: методы и устройства цифровой обработки сигналов

Вариант 7

РАБОТУ ВЫПОЛНИЛ

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

Санкт-Петербург 2020

1 Цель работы

Изучить методику расчета цифрового нерекурсивного фильтра Винера и произвести моделирование его работы.

2 Краткие теоретические сведения

Цифровой фильтр Винера применяется для решения задачи фильтрации сигнала на фоне аддитивной помехи в случае, когда заданы статистические свойства сигнала и помехи (корреляционные функции или функции спектральной плотности мощности). Возможны различные способы реализации фильтра Винера. Изученный в прошлом семестре подход, предполагающий реализацию алгоритма фильтрации в частотной области, имеет недостаток: требуется предварительная запись полного информационного массива данных $\{x_n\}$, n=0,1...N−1, что затрудняет обработку сигнала в реальном масштабе времени. Нерекурсивный фильтр свободен от этого недостатка, обеспечивая обработку сигналов неограниченной длительности.

Нерекурсивный фильтр, имеющий разностное уравнение вида

$$y_n = \sum_{m=0}^{Q-1} b_m x_{n-m}$$

будет представлять собой фильтр Винера, если коэффициенты $\{b_m\}$, m=0,1...Q−1 рассчитаны как решение матричного уравнения Винера-Хопфа

$$\vec{b} = R_x^{-1} \vec{r}_s$$

где R_x – теплицева матрица размера Q×Q, элементы которой являются отсчетами дискретной автокорреляционной функции смеси полезного сигнала с помехой, при взаимной их независимости справедливо равенство $R_X = R_S + R_V$, в котором R_S - матрица автокорреляций полезного сигнала, R_V - матрица автокорреляций помехи; r_S - вектор-столбец отсчетов корреляционной функции сигнала.

Первое уравнение позволяет организовать обработку на скользящем окне длиной Q отсчетов, обеспечивая возможность работы фильтра по мере поступления новых отсчетов входного сигнала.

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 $\cos(\pi \cdot k \cdot T_{\Delta}/Q) - \cos(3\pi \cdot k \cdot T_{\Delta}/Q)$, данный нам по заданию, задаём дискретизацию сигнала и длину полезного сигнала. Результат генерации сигнала на рисунке 3.1.

```
useful_signal_length = 2**7
signal_discretization = 1/useful_signal_length
useful_signal_range = np.linspace(0, useful_signal_length - 1,
                                  useful_signal_length)
useful_signal = (
    cos(pi * useful_signal_range * signal_discretization /
        useful_signal_length)
    - cos(3 * pi * useful_signal_range * signal_discretization /
          useful_signal_length))
```

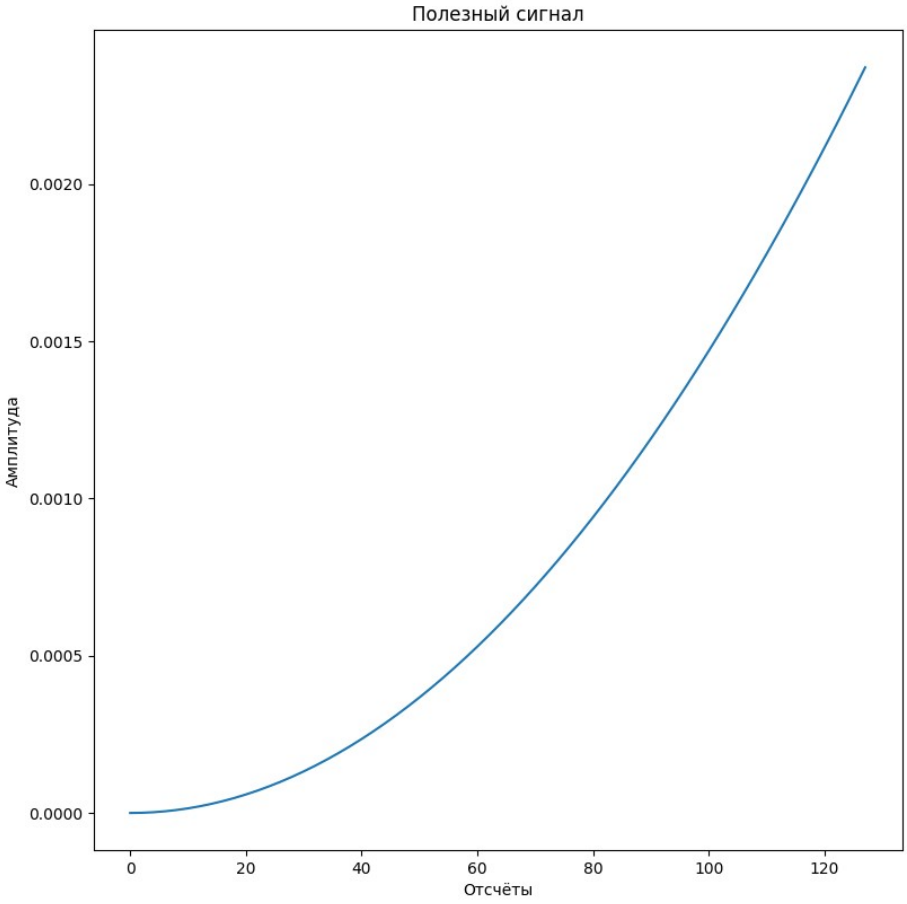

Рисунок 3.1 — Исходный сигнал

Данный сигнал не репрезентативен и едва показывает способности данного фильтра. На графике 3.2 можно увидеть, что белый шум следует за сигналом, что делает фильтрацию сомнительной. На рисунке 3.3 показан сигнал с экспоненциальным шумом, в этот раз сигнал распознать невозможно даже на фильтрованном сигнале. Это при дисперсии равной 0.001, при увеличении этого значения хотя бы на 0.001 – фильтрация будет невозможна.

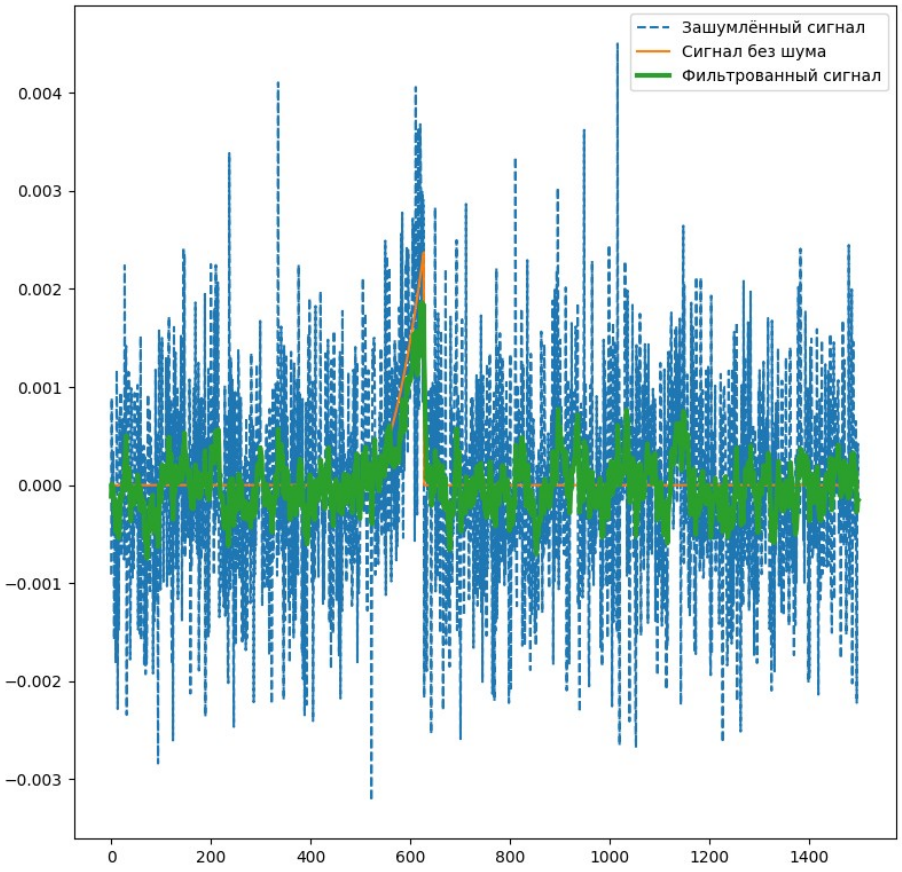

Рисунок 3.2 – Фильтрация сигнала по заданию с БШ

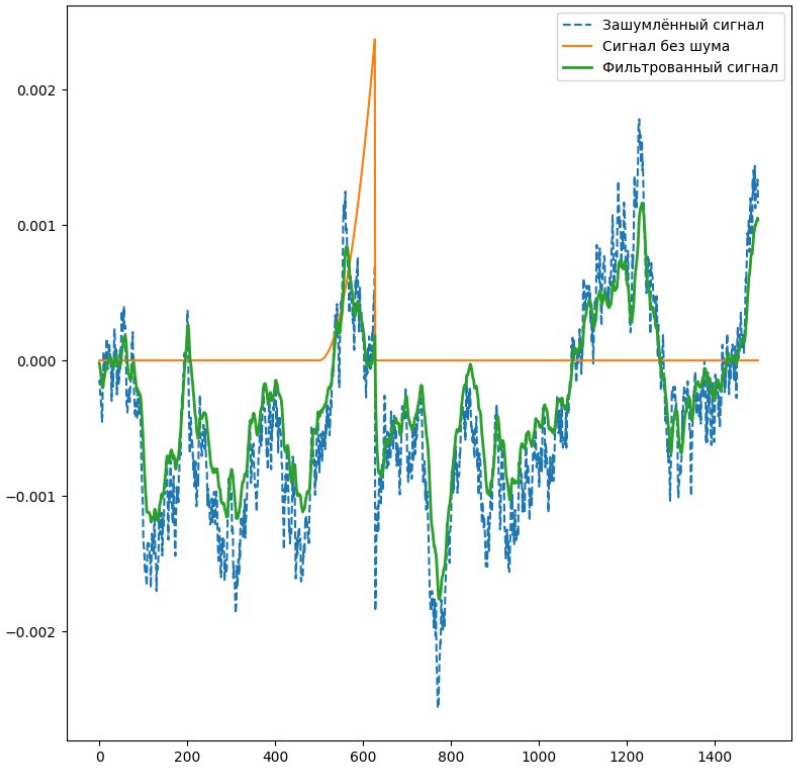

Рисунок 3.3 – Фильтрация сигнала по заданию с экспоненциальным шумом

По этому мной был взят сигнал $\cos\left(2\pi \cdot 3 \cdot k \cdot T_{\Delta}\right)^3$, показанный на рисунке 3.4

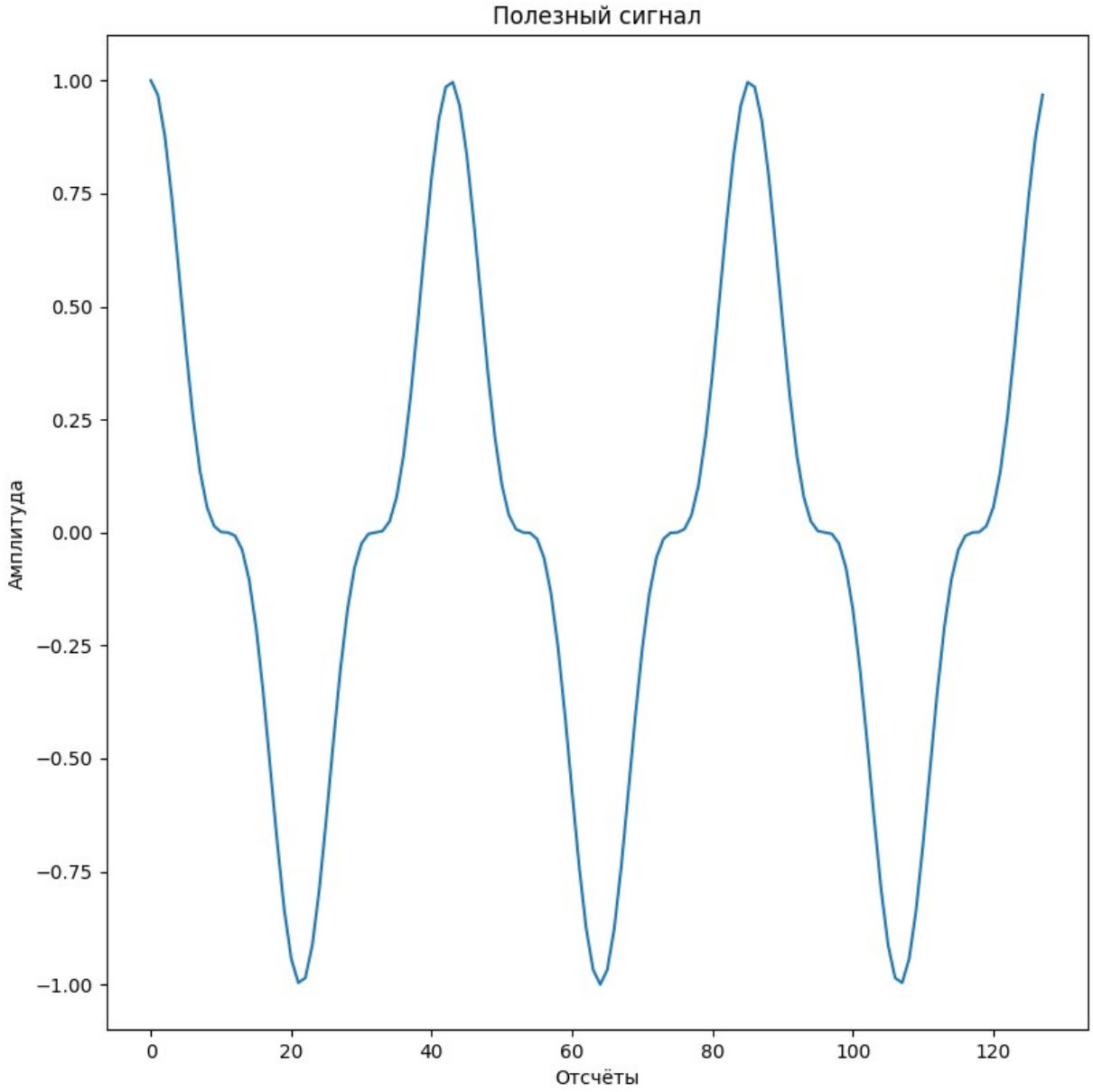

Рисунок 3.4 – Новый сигнал

Далее только что полученный сигнал дополняется нулями с обоих сторон:

```
main_signal_length = 1500
shift_size = 500
main_signal = []
for number in range(0, main_signal_length):
    if shift_size < number and number < shift_size + useful_signal_length:
        main_signal.append(useful_signal[number - shift_size])
```

```
    else:
        main_signal.append(0)
main_signal = np.array(main_signal)
```

После чего прибавляем к полученному сигналу белый шум с дисперсией 0.9, результат обозначен на рисунке 3.5 как :

```
noise_dispersion = 0.9
gaussian_noise = np.random.normal(0, noise_dispersion, main_signal_length)
gaussian_noise_signal = main_signal + gaussian_noise
```

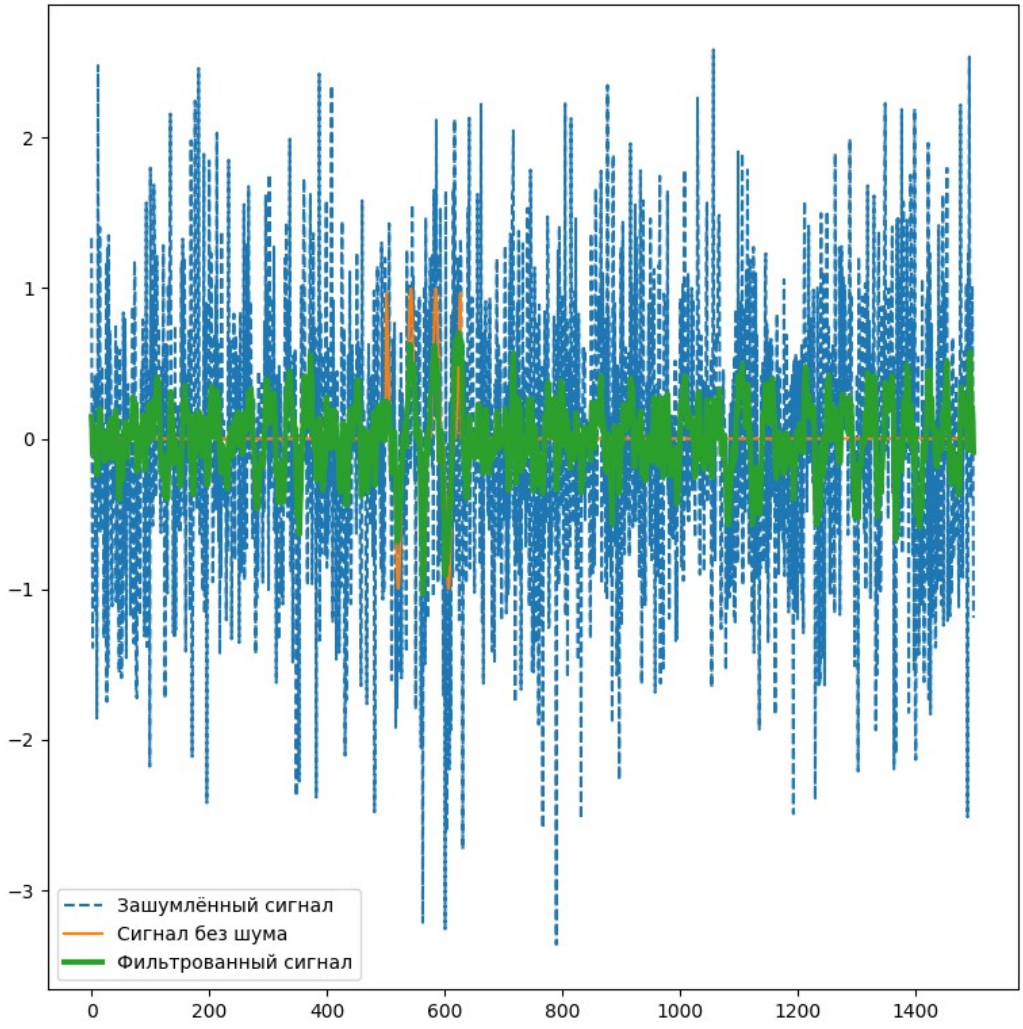

Рисунок 3.5 – Сравнение зашумлённого сигнала и фильтрованного сигнала

Генерируем экспоненциально коррелированный шум с дисперсией 0.9, результат на рисунке 3.6:

```
distribution_of_a_random_variable = 1
exponential_operator = (
    distribution_of_a_random_variable * signal_discretization)
noise_exp = exp(-exponential_operator)
gaussian_noise_for_exp = np.random.normal(0, 1, main_signal_length)
```

```
exponentional_noise = [noise_dispersion * np.sqrt(1 - noise_exp**2) *
                       gaussian_noise_for_exp[number]]
for number in range(1, main_signal_length):
    exponentional_noise.append(
            exponentional_noise[number - 1] * noise_exp
            + gaussian_noise_for_exp[number] * noise_dispersion * np.sqrt(
                1 - noise_exp**2))
exponentional_noise = np.array(exponentional_noise)
exponentional_noise_signal = main_signal + exponentional_noise
```

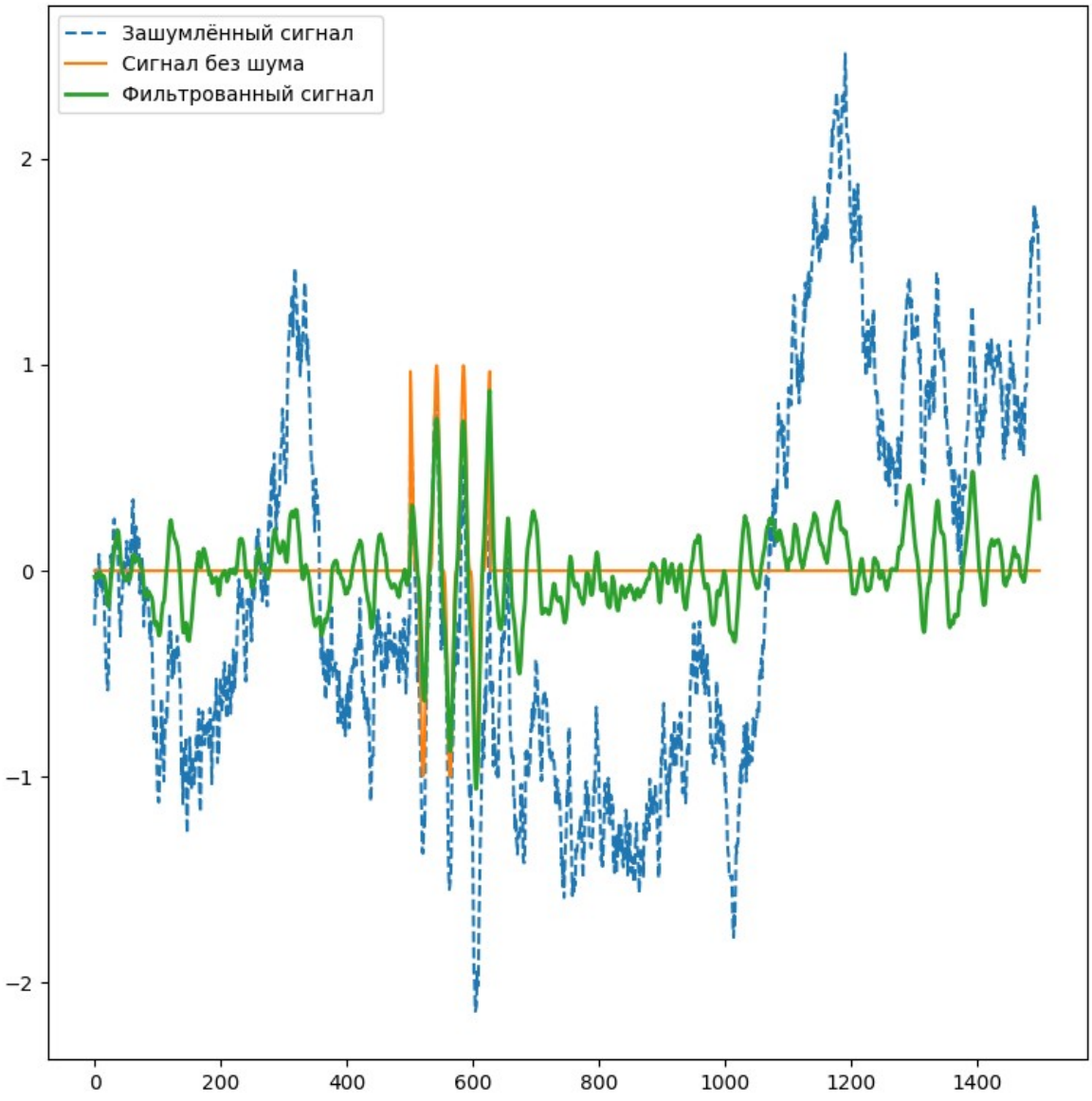

Рисунок 3.6 – Сравнение зашумлённого сигнала и фильтрованного сигнала

Чтобы проверить действительно ли у нас получился экспоненциально коррелированный шум, получаем автокорреляцию полученного шума и теоретическую автокорреляцию шума на строчках ниже. Их сравнение показано на рисунке 3.7

```
exponentional_noise_correlation = np.correlate(
```

```
    exponentional_noise, exponentional_noise, mode="full")\
    / exponentional_noise.size

correlation_shift_range = np.linspace(0, main_signal_length,
                                      main_signal_length)
theoretical_exponentional_noise_correlation = (
    noise_dispersion**2 * np.exp(-distribution_of_a_random_variable
                                 * correlation_shift_range
                                 * signal_discretization))
```

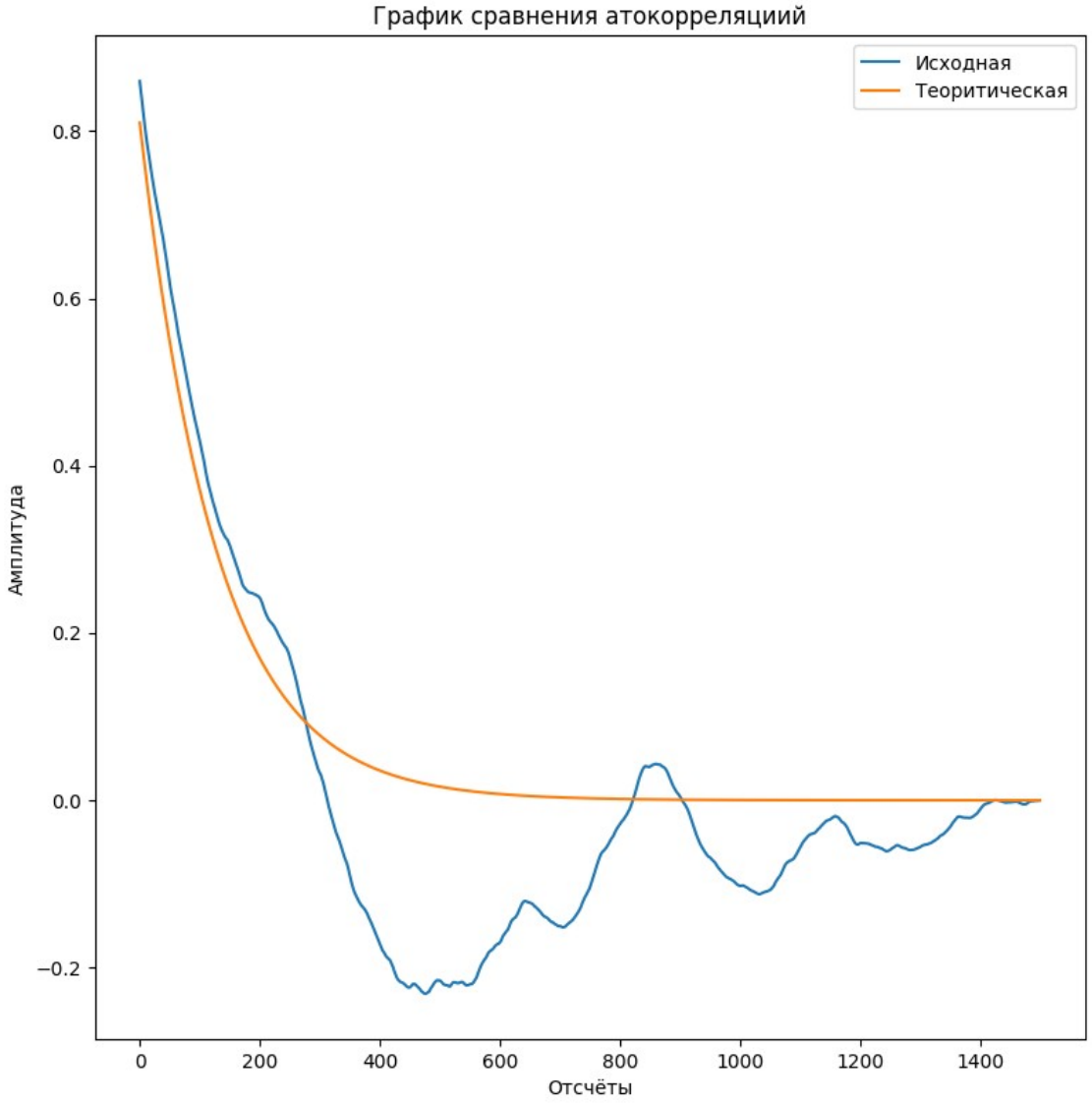

Рисунок 3.7 – График сравнений автокорреляций

Чтобы отфильтровать сигнал нужно найти коэффициенты Виннера. Для начала нужно найти вектор-столбец отсчётов корреляционной функции сигнала. Следующие строчки делают именно это:

```
usful_signal_correlation = []
expected_value = sum(useful_signal) / useful_signal_length
for number in useful_signal_range:
    usful_signal_correlation_step_sum = 0
```

```
    for sum_number in range(0, useful_signal_length - int(number)):
        usful_signal_correlation_step_sum += (((useful_signal[sum_number]
                                                - expected_value)
                                               * (useful_signal[
                                                  sum_number
                                                  + int(number)]
                                                - expected_value)))
    usful_signal_correlation.append(usful_signal_correlation_step_sum
                                    / (useful_signal_length))
usful_signal_correlation = np.array(usful_signal_correlation)
```

Затем находим матрицу автокорреляций полезного сигнала:

```
autocorrelation_useful_signal_matrix = []
for i in useful_signal_range:
    autocorrelation_useful_signal_matrix.append([])
    for j in useful_signal_range:
        autocorrelation_useful_signal_matrix[int(i)].append(
            usful_signal_correlation[abs(int(i) - int(j))])
autocorrelation_useful_signal_matrix = np.array(
    autocorrelation_useful_signal_matrix)
```

После чего находим теоретическую матрицу автокорреляций белого шума:

```
autocorrelation_gaussian_noise_matrix = []
for i in useful_signal_range:
    autocorrelation_gaussian_noise_matrix.append([])
    for j in useful_signal_range:
        if i == j:
            autocorrelation_gaussian_noise_matrix[int(i)].append(
                noise_dispersion**2)
        else:
            autocorrelation_gaussian_noise_matrix[int(i)].append(0)
autocorrelation_gaussian_noise_matrix = np.array(
    autocorrelation_gaussian_noise_matrix)
```

Складываем автокорреляции, находим коэффициенты Винера для БШ и фильтруем сигнал с помощью функции wiener_signal_filter(), результат фильтрации на рисунке 3.8:

```
cross_correlation_matrix = (autocorrelation_gaussian_noise_matrix
                            + autocorrelation_useful_signal_matrix)

gaussian_noise_wiener_coefficients = (np.matmul(np.linalg.inv(
    cross_correlation_matrix), usful_signal_correlation))

gaussian_noise_filtered_signal = wiener_signal_filter(
    main_signal_length, useful_signal_length,
    gaussian_noise_wiener_coefficients, gaussian_noise_signal)
```

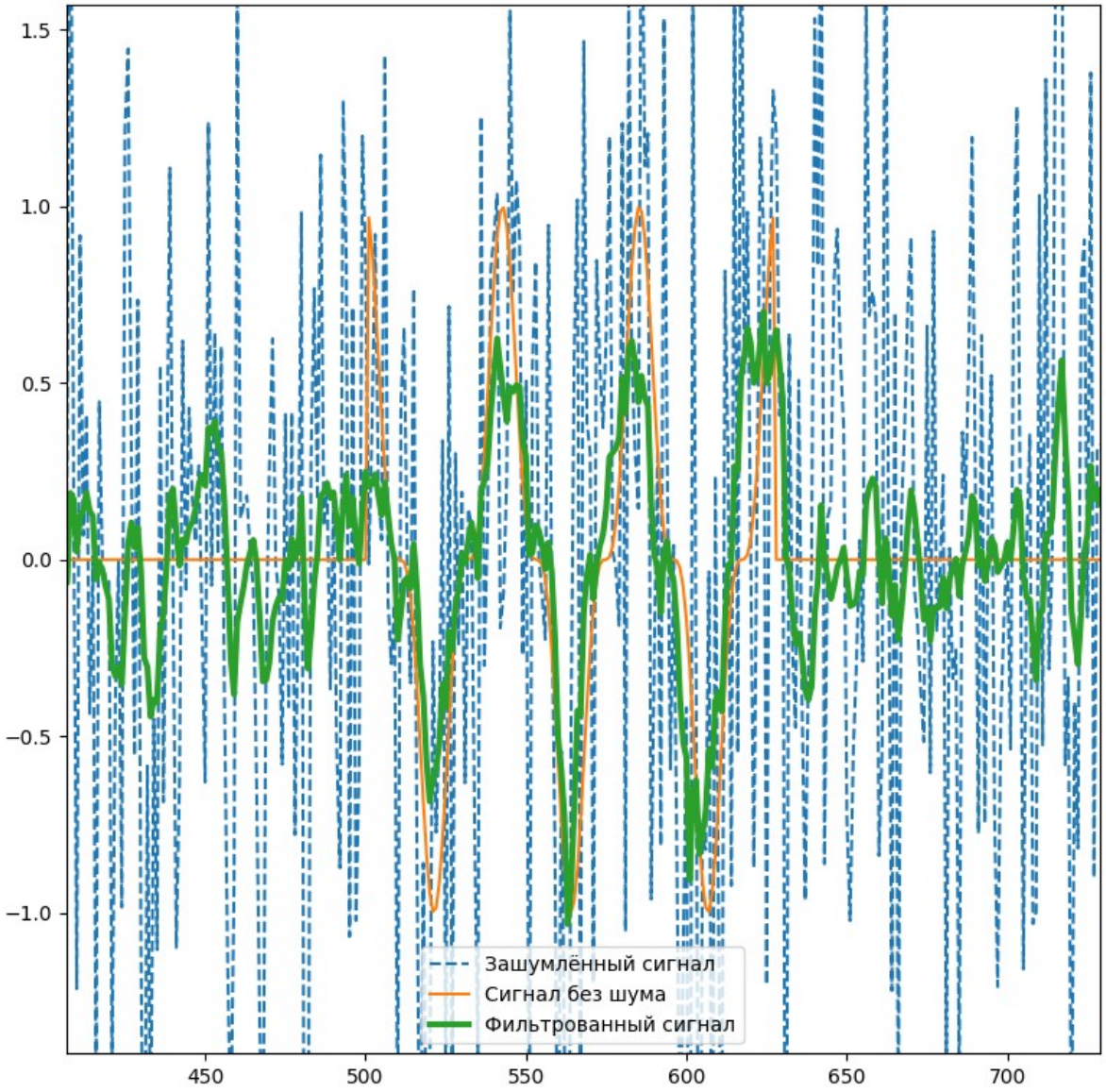

Рисунок 3.8 – Результат фильтрации зашумлённого БШ сигнала

Функция wiener_signal_filter() выполняет фильтрацию по формуле $y_n = \sum_{m=0}^{Q-1} b_m x_{n-m}$ и содержит следующий код:

```
def wiener_signal_filter(signal_length, signal_that_need_to_filter_length,
                         wiener_coefficients, noised_signal):
    """Signal filtration with wiener filter"""
    from numpy import array

    filtered_signal = []
```

```
    for number in range(0, signal_length):
        filtered_signal_step_sum = 0
        for sum_number in range(0, signal_that_need_to_filter_length):
            if number - sum_number >= 0:
                filtered_signal_step_sum += (
                    wiener_coefficients[sum_number]
                    * noised_signal[number - sum_number])
        filtered_signal.append(filtered_signal_step_sum)

    return array(filtered_signal)
```

Возвращаемся в main(), там с помощью строчек ниже находим разность фильтрованного сигнала и заданной функцией дополненной нулями, результат на рисунке 3.9

```
    gaussian_signal_delta = main_signal - gaussian_noise_filtered_signal
    gaussian_signal_stdev = (stdev(gaussian_signal_delta)
                             / stdev(main_signal))
    plot_signal_difference(gaussian_signal_delta, gaussian_signal_stdev)
```

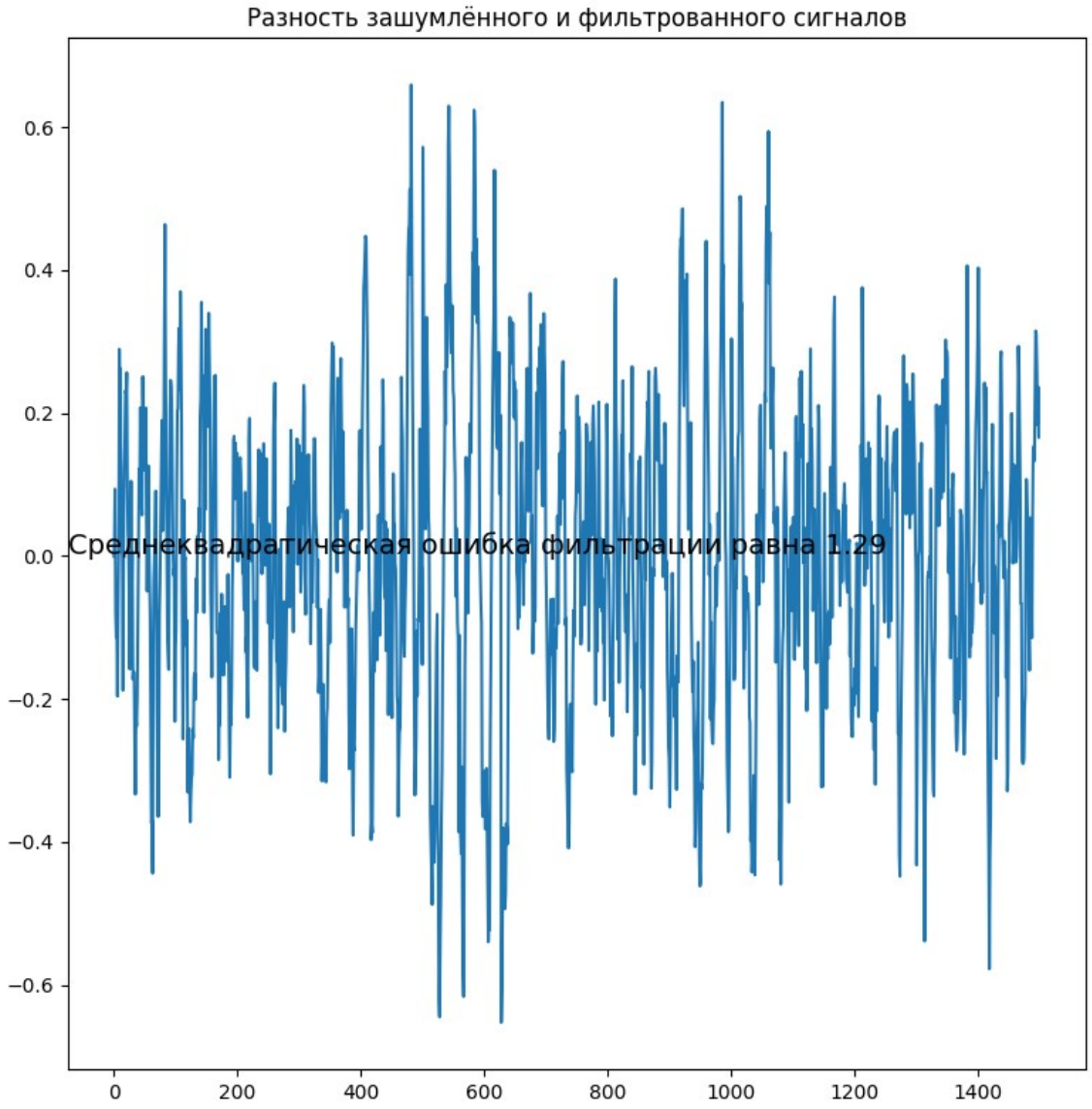

Рисунок 3.9 – Разница исходного и фильтрованного сигнала с БШ, и среднеквадратическая ошибка фильтрации

Для того, чтобы отфильтровать сигнал с экспоненциально коррелированным шумом нужно найти только теоретическую автокорреляцию для экспоненциального шума, затем сложить автокорреляции, найти коэффициенты Винера для сигнала с экспоненциально коррелированным шумом, после чего профильтровать. Это делается строчками ниже. Результат на рисунке 3.10.

```
cross_correlation_matrix = (autocorrelation_exponential_noise_matrix
                            + autocorrelation_useful_signal_matrix)

exponential_noise_wiener_coefficients = (np.matmul(np.linalg.inv(
    cross_correlation_matrix), usful_signal_correlation))
```

```
exponentional_noise_filtered_signal = wiener_signal_filter(
    main_signal_length, useful_signal_length,
    exponential_noise_wiener_coefficients, exponentional_noise_signal)
```

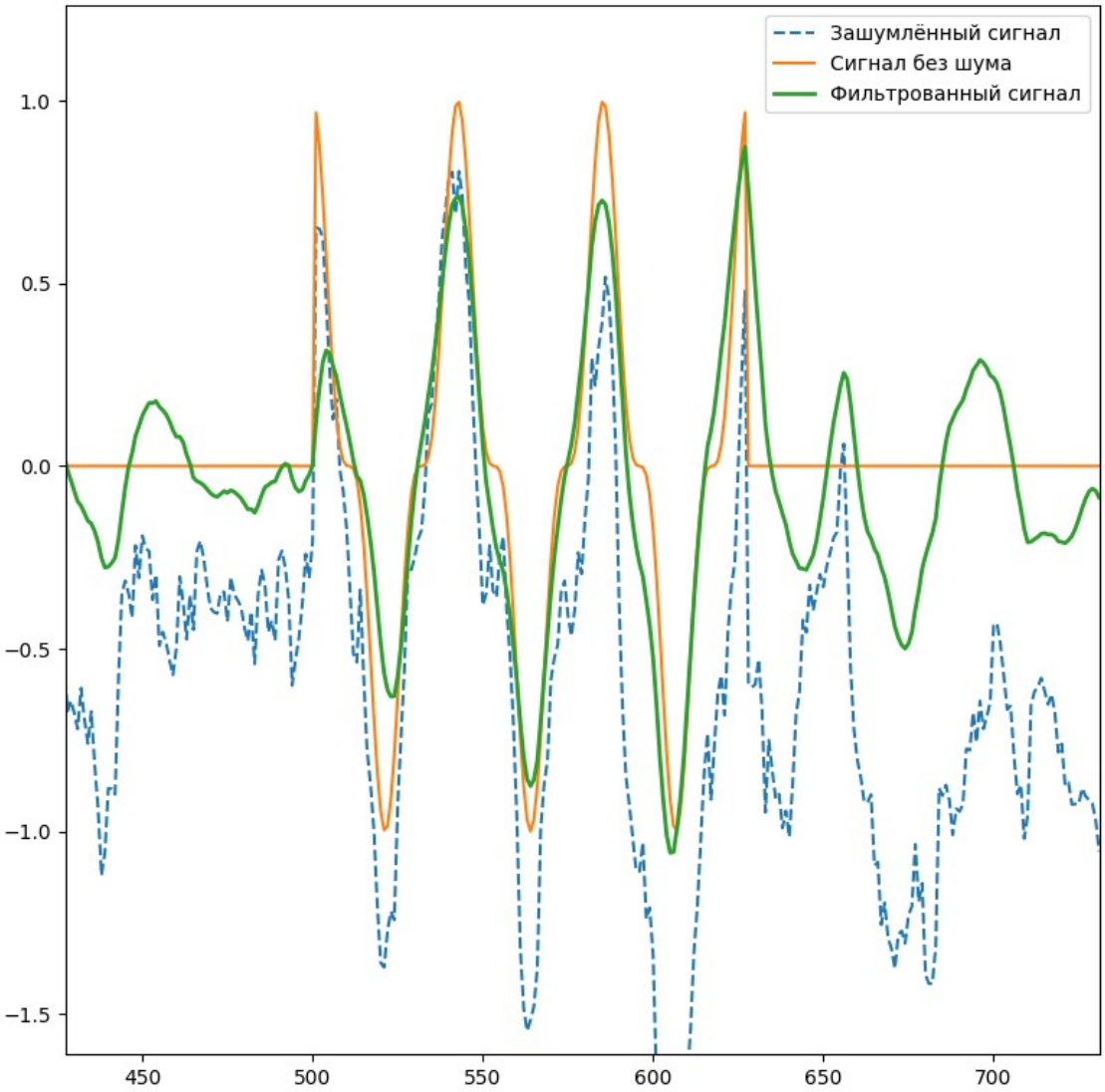

Рисунок 3.10 – Результат фильтрации зашумлённого экспоненциально коррелированным шумом сигнала

После чего находим разность фильтрованного сигнала и заданной функцией дополненной нулями, результат на рисунке 3.11

```
exponential_signal_delta = main_signal - exponentional_noise_signal
exponential_signal_stdev = (stdev(exponential_signal_delta)
                            / stdev(main_signal))
plot_signal_difference(exponential_signal_delta, exponential_signal_stdev)
```

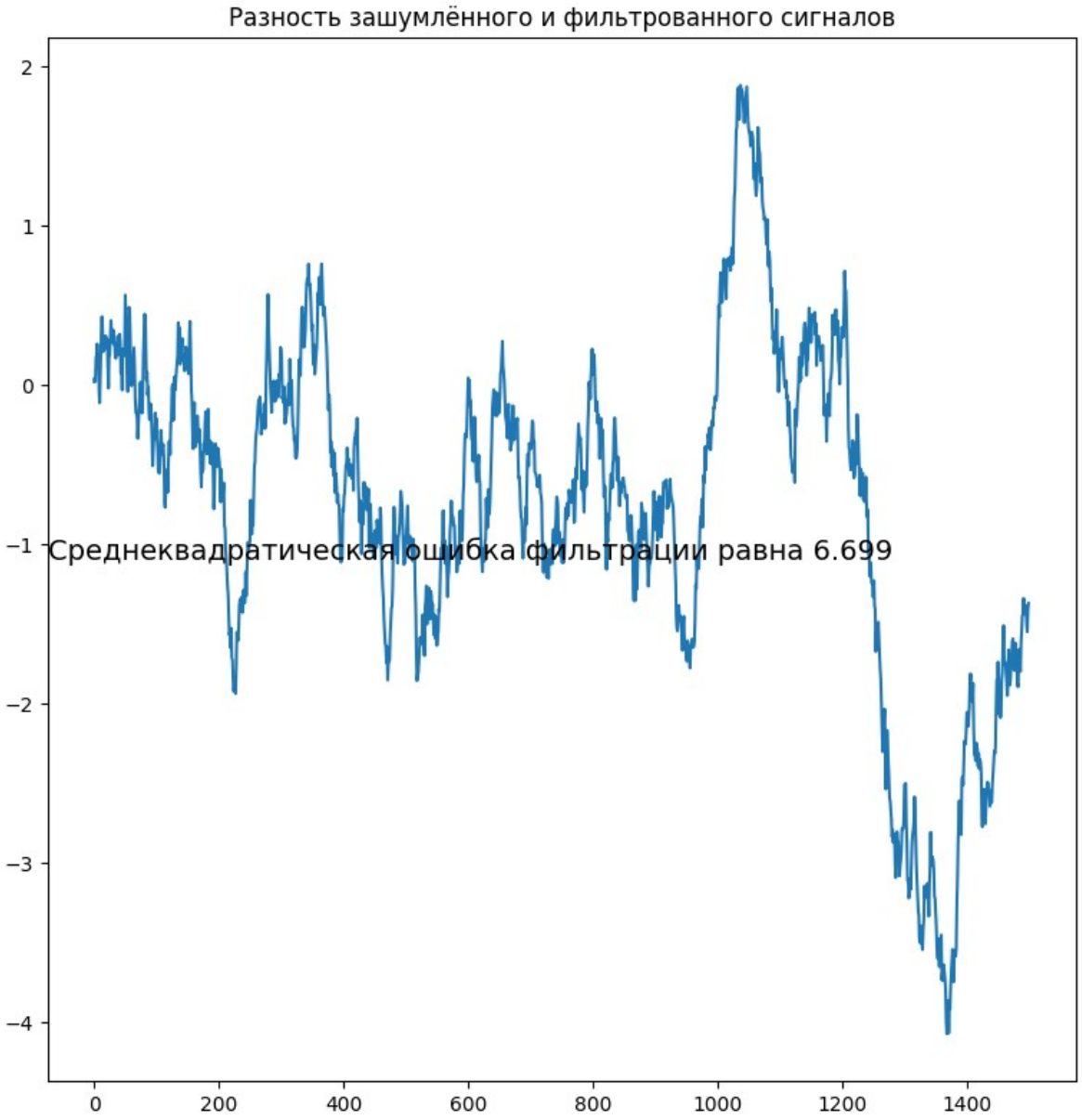

Рисунок 3.11 – Разница исходного и фильтрованного сигнала с экспоненциально коррелированным шумом, и среднеквадратическая ошибка фильтрации

4 Выводы

В рамках данной работы мной был рассчитан и промоделирован нерекурсивный фильтр Винера. С его помощью были профильтрован сигнал с белым шумом и экспоненциально коррелированным шумом. Убедиться, что это был именно экспоненциально коррелированный шум можно посмотрев на рисунок 3.7, на нём видно, что график автокорреляции нашего шума похож на теоретический график экспоненциально коррелированного шума. Результаты фильтрации видны на рисунках 3.8 и 3.10. Чисто визуально можно заключить, что экспоненциально коррелированный шум фильтруется лучше, чем белый шум. С другой стороны, если исходный сигнал не имеет повторяющегося паттерна, то распознать его на фоне экспоненциально коррелированного шума будет невозможно. Это видно на рисунке 3.3

При выполнении работы стал очевиден главный минус фильтрации Виннера – нужно угадывать корреляционную функцию помехи.

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 4: Методы аппроксимации и их применение в ЦОС: сглаживающая фильтрация и мультимедиа-сжатие. О.О. Жаринов, ГУАП, 14 октября 2020г. [Электронный ресурс]. – Режим доступа: https://lms.guap.ru/new/mod/bigbluebuttonbn/view.php?id=26193 – Загл. с экрана. (Дата обращения 20.11.2020г.).

3. Фильтр Винера-Хопфа [Электронный ресурс]. – Режим доступа: https://life-prog.ru/1_40046_filtr-vinera-hopfa.html – Загл. с экрана. (Дата обращения 20.11.2020г.).

Приложение А – Программа

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

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

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


def wiener_signal_filter(signal_length, signal_that_need_to_filter_length,
                         wiener_coefficients, noised_signal):
    """Signal filtration with wiener filter"""
    from numpy import array

    filtered_signal = []
    for number in range(0, signal_length):
        filtered_signal_step_sum = 0
        for sum_number in range(0, signal_that_need_to_filter_length):
            if number - sum_number >= 0:
                filtered_signal_step_sum += (
                    wiener_coefficients[sum_number]
                    * noised_signal[number - sum_number])
        filtered_signal.append(filtered_signal_step_sum)

    return array(filtered_signal)


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

    # Set parameters for useful signal and find him
    useful_signal_length = 2**7
    signal_discretization = 1/useful_signal_length
    useful_signal_range = np.linspace(0, useful_signal_length - 1,
                                      useful_signal_length)
    # useful_signal = (
    #     cos(pi * useful_signal_range * signal_discretization /
    #         useful_signal_length)
```

```
#     - cos(3 * pi * useful_signal_range * signal_discretization /
#           useful_signal_length))
useful_signal = (cos(2 * pi * useful_signal_range * 3
                * signal_discretization))**3

# Plot useful signal
plt.figure()
plt.plot(useful_signal)
plt.title("Полезный сигнал")
plt.xlabel("Отсчёты")
plt.ylabel("Амплитуда")

# Set parametrs for main signal and generate him
main_signal_length = 1500
shift_size = 500
main_signal = []
for number in range(0, main_signal_length):
    if shift_size < number and number < shift_size + useful_signal_length:
        main_signal.append(useful_signal[number - shift_size])
    else:
        main_signal.append(0)
main_signal = np.array(main_signal)

# Generate gaussian noise
noise_dispersion = 0.9
gaussian_noise = np.random.normal(0, noise_dispersion, main_signal_length)
gaussian_noise_signal = main_signal + gaussian_noise

# Generate exponential correlation function noise, which formula
# has been taken from "Быков В.В. Цифровое моделирование в статистич
# радиотехнике" page 104, № 1
distribution_of_a_random_variable = 1
exponential_operator = (
    distribution_of_a_random_variable * signal_discretization)
noise_exp = exp(-exponential_operator)
gaussian_noise_for_exp = np.random.normal(0, 1, main_signal_length)
exponentional_noise = [noise_dispersion * np.sqrt(1 - noise_exp**2) *
                       gaussian_noise_for_exp[number]]
for number in range(1, main_signal_length):
    exponentional_noise.append(
            exponentional_noise[number - 1] * noise_exp
            + gaussian_noise_for_exp[number] * noise_dispersion * np.sqrt(
                1 - noise_exp**2))
exponentional_noise = np.array(exponentional_noise)

exponentional_noise_signal = main_signal + exponentional_noise

# Find correlation exponentional noise and theoretical correlation
exponentional_noise_correlation = np.correlate(
    exponentional_noise, exponentional_noise, mode="full")\
    / exponentional_noise.size
```

```
correlation_shift_range = np.linspace(0, main_signal_length,
                                      main_signal_length)
theoretical_exponentional_noise_correlation = (
    noise_dispersion**2 * np.exp(-distribution_of_a_random_variable
                                 * correlation_shift_range
                                 * signal_discretization))

# Plot theoretical and practical correlation
plt.figure()
plt.title("График сравнения атокорреляциий")
plt.xlabel("Отсчёты")
plt.ylabel("Амплитуда")
plt.plot(exponentional_noise_correlation[
    int(exponentional_noise_correlation.size//2):], label="Исходная")
plt.plot(theoretical_exponentional_noise_correlation,
         label="Теоритическая")
plt.legend()

# Find coeficients for gaussian noise signal filter
usful_signal_correlation = []
expected_value = sum(useful_signal) / useful_signal_length
for number in useful_signal_range:
    usful_signal_correlation_step_sum = 0
    for sum_number in range(0, useful_signal_length - int(number)):
        usful_signal_correlation_step_sum += (((useful_signal[sum_number]
                                                - expected_value)
                                               * (useful_signal[
                                                   sum_number
                                                   + int(number)]
                                                  - expected_value)))
    usful_signal_correlation.append(usful_signal_correlation_step_sum
                                    / (useful_signal_length))
usful_signal_correlation = np.array(usful_signal_correlation)

autocorrelation_useful_signal_matrix = []
for i in useful_signal_range:
    autocorrelation_useful_signal_matrix.append([])
    for j in useful_signal_range:
        autocorrelation_useful_signal_matrix[int(i)].append(
            usful_signal_correlation[abs(int(i) - int(j))])
autocorrelation_useful_signal_matrix = np.array(
    autocorrelation_useful_signal_matrix)

autocorrelation_gaussian_noise_matrix = []
for i in useful_signal_range:
    autocorrelation_gaussian_noise_matrix.append([])
    for j in useful_signal_range:
        if i == j:
            autocorrelation_gaussian_noise_matrix[int(i)].append(
                noise_dispersion**2)
        else:
            autocorrelation_gaussian_noise_matrix[int(i)].append(0)
```

```
autocorrelation_gaussian_noise_matrix = np.array(
   autocorrelation_gaussian_noise_matrix)

cross_correlation_matrix = (autocorrelation_gaussian_noise_matrix
                        + autocorrelation_useful_signal_matrix)

gaussian_noise_wiener_coefficients = (np.matmul(np.linalg.inv(
   cross_correlation_matrix), usful_signal_correlation))

gaussian_noise_filtered_signal = wiener_signal_filter(
   main_signal_length, useful_signal_length,
   gaussian_noise_wiener_coefficients, gaussian_noise_signal)

plt.figure()
plt.plot(gaussian_noise_signal, '--', label="Зашумлённый сигнал")
plt.plot(main_signal, label="Сигнал без шума")
plt.plot(gaussian_noise_filtered_signal, label="Фильтрованный сигнал",
       linewidth=3)
plt.legend()

gaussian_signal_delta = main_signal - gaussian_noise_filtered_signal
gaussian_signal_stdev = (stdev(gaussian_signal_delta)
                     / stdev(main_signal))
plot_signal_difference(gaussian_signal_delta, gaussian_signal_stdev)

# Find coeficients for filter signal with exponential noise
autocorrelation_exponential_noise_matrix = []
for i in useful_signal_range:
   autocorrelation_exponential_noise_matrix.append([])
   for j in useful_signal_range:
      if i == j:
         autocorrelation_exponential_noise_matrix[int(i)].append(
            noise_dispersion**2 * noise_exp)
      else:
         autocorrelation_exponential_noise_matrix[int(i)].append(0)
autocorrelation_exponential_noise_matrix = np.array(
   autocorrelation_exponential_noise_matrix)

cross_correlation_matrix = (autocorrelation_exponential_noise_matrix
                        + autocorrelation_useful_signal_matrix)

exponential_noise_wiener_coefficients = (np.matmul(np.linalg.inv(
   cross_correlation_matrix), usful_signal_correlation))

exponentional_noise_filtered_signal = wiener_signal_filter(
   main_signal_length, useful_signal_length,
   exponential_noise_wiener_coefficients, exponentional_noise_signal)

plt.figure()
plt.plot(exponentional_noise_signal, '--', label="Зашумлённый сигнал")
plt.plot(main_signal, label="Сигнал без шума")
plt.plot(exponentional_noise_filtered_signal, label="Фильтрованный сигнал",
```

```
            linewidth=2)
    plt.legend()

    exponential_signal_delta = main_signal - exponentional_noise_signal
    exponential_signal_stdev = (stdev(exponential_signal_delta)
                                / stdev(main_signal))
    plot_signal_difference(exponential_signal_delta, exponential_signal_stdev)

    plt.show()


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