Моделирование алгоритмов ЦОС и обоснование подходящего для решения поставленной задачи

Для проверки работоспособности выбранного цифрового фильтра нужно промоделировать его работу. Фильтрация производится во временной области по уравнению (1)

$$y_n = \sum_{k=0}^{q} b_k x_{n-k} - \sum_{m=1}^{r} a_m y_{n-m} \qquad (1)$$

где массивы b и a вычисляются по формулам из таблицы 1

Таблица 1 − Формулы сомножителей D к (z), получающиеся при применении формул обобщённого билинейного преобразования к выражению (1)

Тип ЦФ	Формула для $D_K(z)$	Формулы* для коэффициентов $D_K(z)$
ФНЧ с частотой среза $\omega_{ср}$	$\frac{\beta_0 z^2+\beta_1 z+\beta_2}{\alpha_0 z^2+\alpha_1 z+\alpha_2}$	$\beta_0=(\omega_{cp}T_\Delta)^2, \beta_1=2\beta_0, \beta_2=\beta_0,$ $\alpha_0=a_2(\omega_{cp}T_\Delta)^2+2a_1\omega_{cp}T_\Delta+4a_0,$ $\alpha_1=2a_2(\omega_{cp}T_\Delta)^2-8a_0,$ $\alpha_2=a_2(\omega_{cp}T_\Delta)^2-2a_1\omega_{cp}T_\Delta+4a_0$
ФВЧ с частотой среза $\omega_{ср}$	$\frac{\beta_0 z^2+\beta_1 z+\beta_2}{\alpha_0 z^2+\alpha_1 z+\alpha_2}$	$\beta_0=4, \beta_1=-8, \beta_2=4,$ $\alpha_0=a_0(\omega_{cp}T_\Delta)^2+2a_1\omega_{cp}T_\Delta+4a_2,$ $\alpha_1=2a_0(\omega_{cp}T_\Delta)^2-8a_2,$ $\alpha_2=a_0(\omega_{cp}T_\Delta)^2-2a_1\omega_{cp}T_\Delta+4a_2$
ПФ с частотами среза $\omega_н$ и $\omega_в$	$\frac{\beta_0 z^4+\beta_1 z^3+\beta_2 z^2+\beta_3 z+\beta_4}{\alpha_0 z^4+\alpha_1 z^3+\alpha_2 z^2+\alpha_3 z+\alpha_4}$	$\beta_0=1, \beta_1=0, \beta_2=-2, \beta_3=0, \beta_4=1,$ $\alpha_0=\gamma^2 a_0+\gamma a_1+a_2,$ $\alpha_1=-4\gamma^2\zeta a_0-2\gamma\zeta a_1,$ $\alpha_2=4\gamma^2\zeta^2 a_0+2\gamma a_0-2a_2,$ $\alpha_3=-4\gamma^2\zeta a_0+2\gamma\zeta a_1,$ $\alpha_4=\gamma^2 a_2-\gamma a_1+a_0$
РФ с частотами среза $\omega_н$ и $\omega_в$	$\frac{\beta_0 z^4+\beta_1 z^3+\beta_2 z^2+\beta_3 z+\beta_4}{\alpha_0 z^4+\alpha_1 z^3+\alpha_2 z^2+\alpha_3 z+\alpha_4}$	$\beta_0=\gamma^2, \beta_1=-4\gamma^2\zeta, \beta_2=4\gamma^2\zeta^2+2\gamma^2,$ $\beta_3=\beta_1, \beta_4=\beta_0,$ $\alpha_0=\gamma^2 a_2-\gamma a_1+a_2,$ $\alpha_1=4\gamma^2\zeta a_2-2\gamma\zeta a_1,$ $\alpha_2=4\gamma^2\zeta^2 a_2+2\gamma a_2-2a_0,$ $\alpha_3=-4\gamma^2\zeta a_2+2\gamma\zeta a_1,$ $\alpha_4=\gamma^2 a_2-\gamma a_1+a_0$

* Примечание: $\gamma = ctg\left(\frac{T_\Delta}{2}(\omega_в - \omega_н)\right), \zeta = \frac{\cos\left(\frac{T_\Delta}{2}(\omega_в + \omega_н)\right)}{\cos\left(\frac{T_\Delta}{2}(\omega_в - \omega_н)\right)}$

Программа для моделирования написана на языке python 3.7 её полное содержание можно найти в приложении 1, я прокомментирую самые важные части. В данном блоке кода задаются характеристики двух сигналов. Один находиться в середине полосы пропускания и должен пройти через фильтр без изменений, а другой подавиться, т. к. находится вне полосы пропускания фильтра Дискретизация сигнала определена в начале файла и равна 1/9000

```
A1 = 3
A2 = 2

f1 = 200
f2 = 400.33
```

С помощью функции signal_generate генерируем сигналы с заданными характеристиками. Это делается для того, чтобы подтвердить, что подавление сигналов соответствует АЧХ. Формула генерации следующая $s = A \cdot \sin(2\pi f t)$, где s – сигнал, A – амплитуда гармоники, f – частота гармоники, t – время. Получившийся сигналы показаны на рисунке 1 и 2, обозначены как нефильтрованные.

```
signal_in_bandwith = signal_generate(signal_discritisation, {A2:f2})
signal_out_of_bandwith = signal_generate(signal_discritisation, {A1:f1})
```

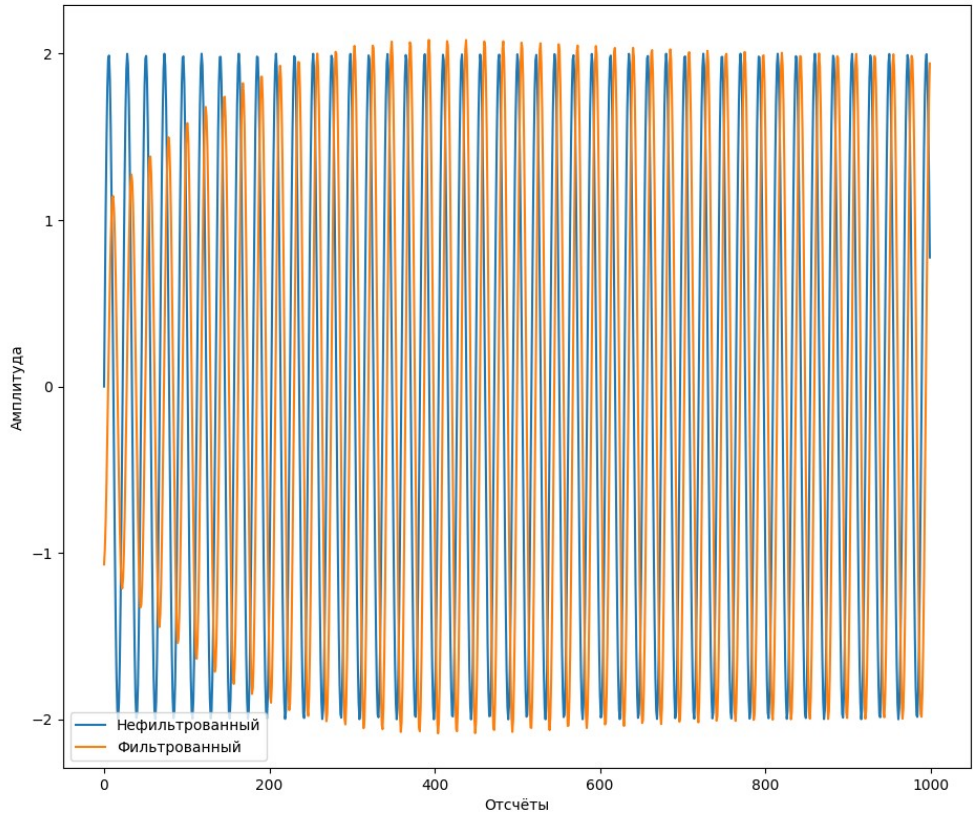

Рисунок 1 – Сравнение фильтрованного и нефильтрованного синусоидального сигнала с амплитудой 2 и частотой 400

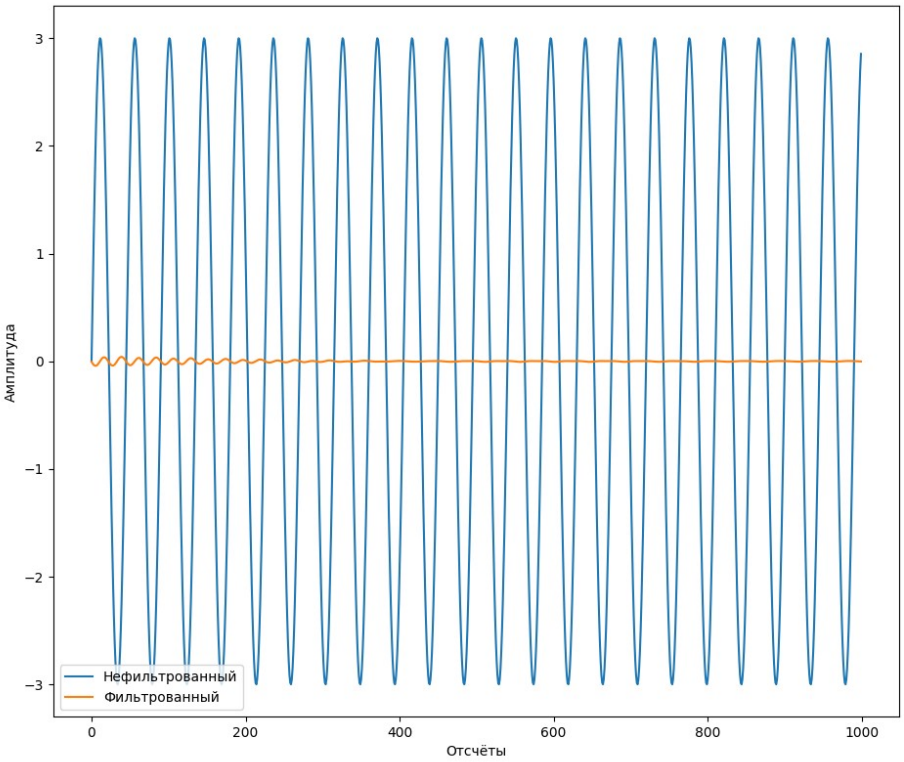

Рисунок 2 – Сравнение фильтрованного и нефильтрованного синусоидального сигнала с амплитудой 3 и частотой 200

Далее с помощь следующей строки добавим к сигналу, находящемуся в полосе пропускания фильтра белый шум с мат. ожиданием 0 и дисперсией 1, чтобы проверить как фильтр справляется непосредственно с задачей фильтрации. Результат на рисунке 3, подписан как нефильтрованный.

gaussian_noise_signal = signal_in_bandwith + lambd * np.random.normal(0, 1, N)

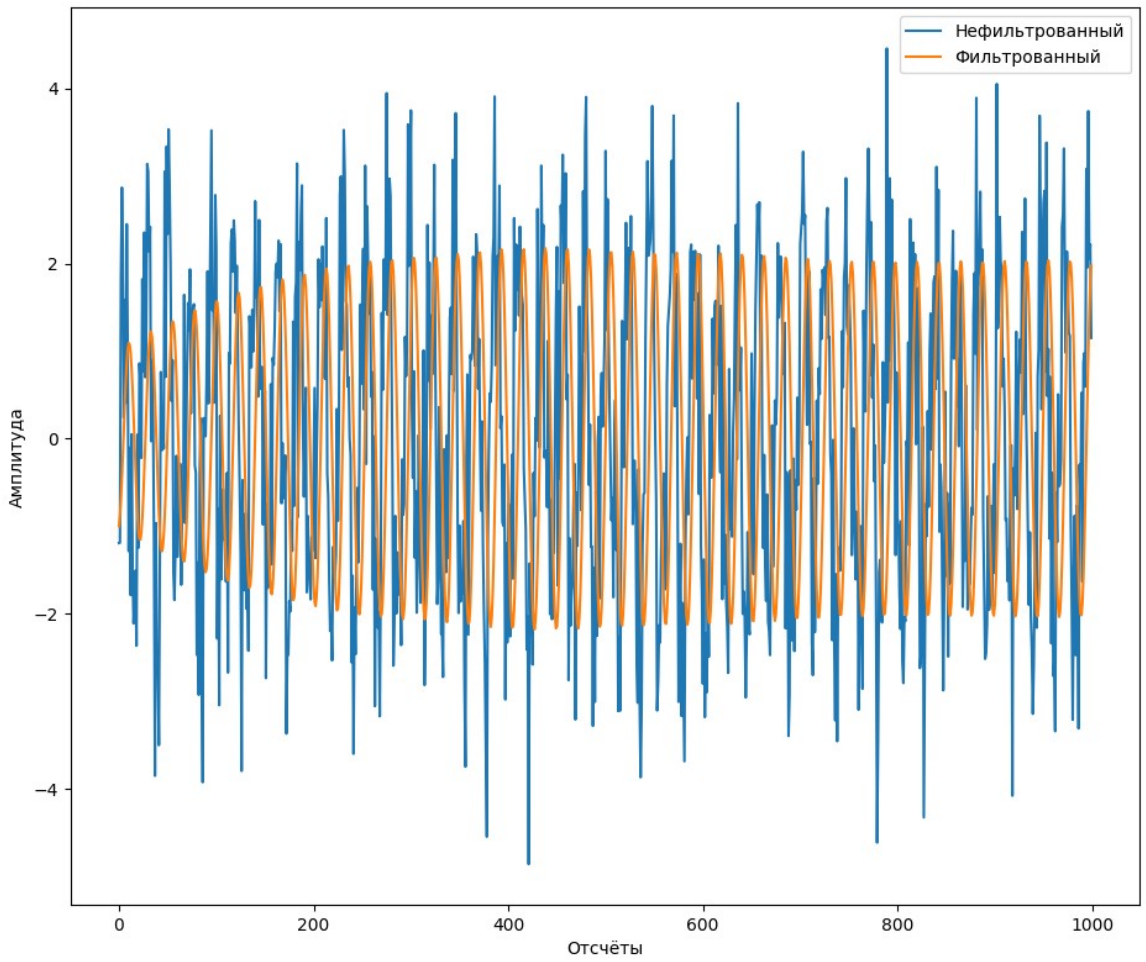

Рисунок 3 – Сравнение фильтрованного и нефильтрованного синусоидального сигнала с амплитудой 2 и частотой 400 и белым шумом

Генерируем АЧХ теоретического рекурсивного фильтра, как в прошлом этапе. Оно показано на рисунке 4.

filter_transfer_function = recursive_transfer_function(np.exp(np.array(coefficient)), fv=410, fn=390)

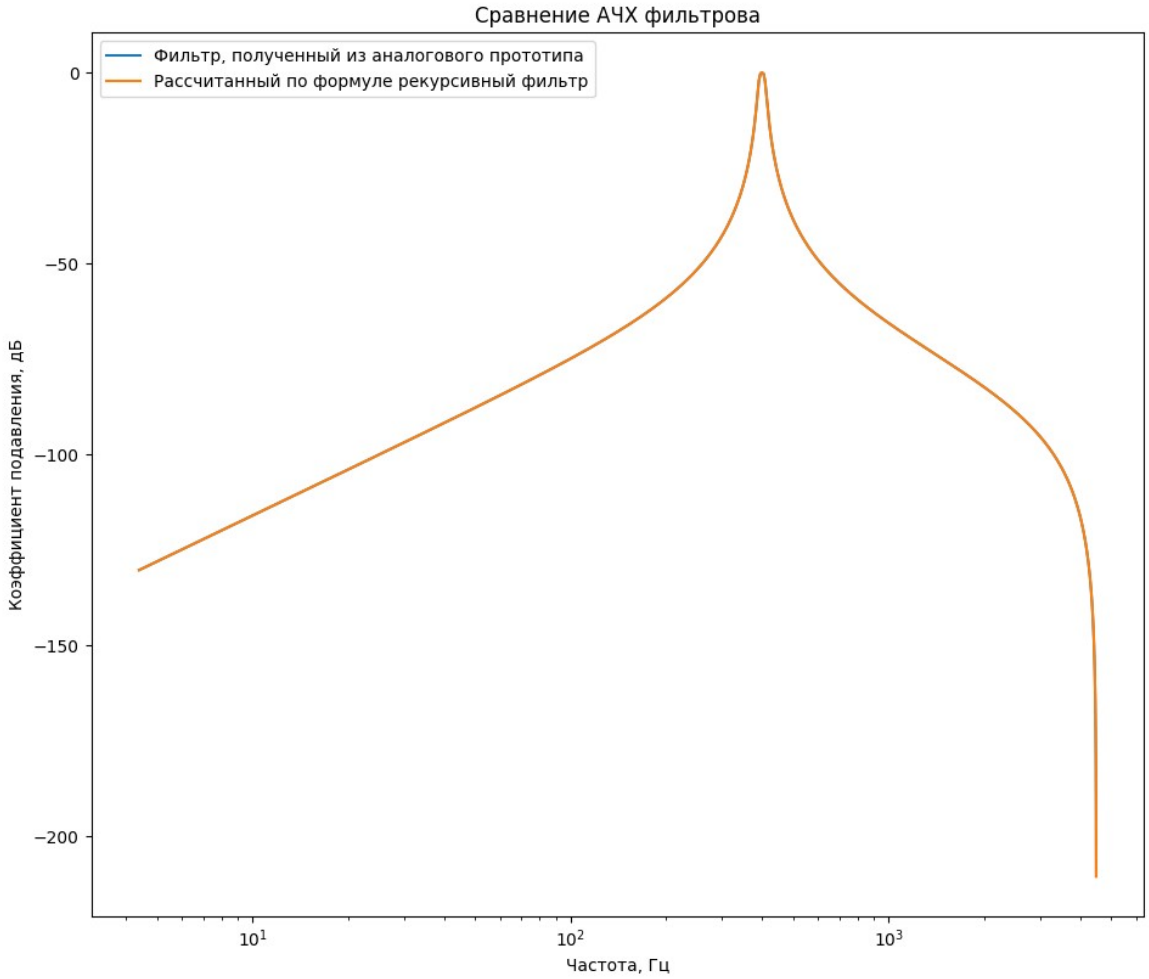

Рисунок 4 – Сравнение двух АЧХ фильтров с одинаковыми параметрами

Затем производится расчёт фильтра для его последующего использования по формулам приведённым в таблице 1, для ПФ. Это делается с помощью функции generate_recursive_filter. Возвращаются как АЧХ, показанное на рисунке 3, так и коэффициенты a и b, используемые при фильтрации.

recursive_filter, A, B = generate_recursive_filter(np.exp(np.array(coefficient)), fv = 410, fn=390)

После чего производиться фильтрация сигнала находящегося в полосе пропускания. Она производится по формуле 1, которая реализована в функции signal_filtration_with_recursive_filter. Результат показан на рисунке 1, обозначен как фильтрованный. Фильтрованный сигнал смещён влево

относительно нефильтрованного на 220 точек, чтобы их удобнее было сравнивать.

filtered_usuall_signal = signal_filtration_with_recursive_filter(signal_in_bandwith, A, B)

Теперь фильтруем сигнал вне полосы пропускания. Результат обозначен как фильтрованный на рисунке 2.

filtered_harmonic_noise_signal = signal_filtration_with_recursive_filter(signal_out_of_bandwith, A, B)

Ну и наконец фильтруем сигнал с белым шумом. Результат на рисунке 3, обозначен как фильрованный.

filtered_gaussian_noise_signal = signal_filtration_with_recursive_filter(gaussian_noise_signal, A, B)

На рисунке 1 видно, что сигнал с частотой входящей в спектр частот в полосе пропускания фильтра минует его без изменений, тогда как сигнал с частотой выходящей за рамки полосы полностью подавляется это видно на 2 рисунке. На 3 рисунке видно, что фильтр так же хорошо справляется с шумами. Из наблюдений можно сделать вывод, что фильтр работает в соответствии с АЧХ

Список литературы:

1. Цифровые фильтры частотной селекции: учебное пособие / О.О. Жаринов, И.О. Жаринов.СПб: Изд-во ГУАП, 2019. – 77 с. [библиотечный шифр 621.372 Ж 34].

ПРИЛОЖЕНИЕ 1

```
# Const
delta_T = 1/9000
N = 2000
q = 2048


def recursive_transfer_function(z, fn=0, fv=1):
    """
        Генерирует теоритически заданный аналоговый прототип ПФ фильтра Баттерворта
    соответственно входным данным

    :z: оператор передаточной функции
    :fn: Нижняя граница частоты среза фильтра
    :fv: Верхняя граница частоты среза фильтра
    :returns: расчитанную передаточную функцию
    """
    import numpy as np

    fn = fn * 2 * np.pi
    fv = fv * 2 * np.pi

    # Вводим константы
    gamma = 1/np.tan((delta_T/2) * (fv - fn))
    zeta = np.cos((delta_T/2) * (fv + fn))/np.cos((delta_T/2) * (fv - fn))

    # Осуществляем замену переменной p из ФНЧ в ПФ
    changed_p = gamma * (((z**2) - 2 * zeta * z + 1)/((z**2) - 1))

    # Записываем передаточную функцию фильтра Баттерворта 2-ого порядка
    calculated_transfer_function = 1/((changed_p**2) + 1.41421 * changed_p + 1)

    return calculated_transfer_function


def generate_recursive_filter(z, fn=0, fv=1):
    """
        Генерирует рекурсивный фильтр с заданными параметрами и его коэффициенты

    :z: оператор передаточной функции
    :fn: Нижняя граница частоты среза фильтра
    :fv: Верхняя граница частоты среза фильтра
```

```
    :returns: Теоритический фильтр по аналоговой передаточной функции
    :returns: Alpha коэффициенты фильтра
    :returns: Beta коэффициенты фильтра
    """
    import numpy as np

    # Преобразовываем частоты для правильной работы
    fn = fn * 2 * np.pi
    fv = fv * 2 * np.pi

    gamma = 1/np.tan((delta_T/2) * (fv - fn))
    zeta = np.cos((delta_T/2) * (fv + fn))/np.cos((delta_T/2) * (fv - fn))

    # переменные используемые по формуле
    filter_order = 2
    order_number = 1

    # Непосредственно генерируем фильтр
    alphas = []
    betas = []
    our_filter = 1
        a_constants = [1, -2 * np.cos(((2 * order_number + filter_order - 1)/(2 * filter_order)) * np.pi), 1]

    current_betta = [1, 0, -2, 0, 1]

        current_alpha = [gamma**2 * a_constants[0] + gamma * a_constants[1] + a_constants[2],
                         -4 * gamma**2 * zeta * a_constants[0] - 2 * gamma * zeta * a_constants[1],
                          4 * gamma**2 * zeta**2 * a_constants[0] + 2 * gamma**2 * a_constants[0] - 2 * a_constants[2],
                         -4 * gamma**2 * zeta * a_constants[0] + 2 * gamma * zeta * a_constants[1],
                gamma**2 * a_constants[0] - gamma * a_constants[1] + a_constants[2]]

        our_filter = our_filter * ((current_betta[0] * z**4 + current_betta[1] * z**3 + current_betta[2] * z**2 + current_betta[3] * z + current_betta[4])/ \
          (current_alpha[0] * z**4 + current_alpha[1] * z**3 + current_alpha[2] * z**2 + current_alpha[3] * z + current_alpha[4]))
    alphas.append(np.array(current_alpha)/current_alpha[0])
    betas.append(np.array(current_betta)/current_alpha[0])

    return np.array(our_filter), np.array(alphas), np.array(betas)
```

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

      :harmonics: гармоники сигнала в виде словаря {амплитуда:частота} для каждой гармоники
    :time: отсчёты по x-оси
    :returns: сигнал с заданными параметрами
    """
    from numpy import sin, pi, zeros

    generated_signal = zeros(len(time))

    for ind, harmonic in enumerate(harmonics):
        generated_signal += harmonic * sin(2 * pi * harmonics[harmonic] * time)

    return generated_signal


def compare_signals(regular_signal, filtered_signal, plot_range, shift_point):
    """
      Строит график сигнала и его АЧХ. Как бы показывает разницу между сигналами на графике

    :regular_signal: обычный сигнал
    :filter_signal: фильтрованный сигнал
      :shift_point: Точка сдвига фильтрованного сигнала. Нужна чтобы убрать задержку и было легче сравнить
    сигналы
    plot_range: отрезок на котором будет построен график
    """
    import matplotlib.pyplot as plt
    import numpy as np

    plt.figure()
    plt.plot(plot_range, regular_signal[plot_range], label='Нефильтрованный')
          plt.plot(plot_range, filtered_signal[np.array(plot_range) + shift_point], label='Фильтрованный')
    plt.xlabel('Отсчёты')
    plt.ylabel('Амплитуда')
    plt.legend()


def signal_plot(input_signal):
```

```
    """
    Строит график сигнала и амплитудного спектра

    :input_signal: входной сигнал
    """
    import matplotlib.pyplot as plt
    import numpy as np

    # Вычисляем частоту сигнала
    k = np.arange(0, int(len(input_signal)/2), 1)
    frequency = k/(int(len(input_signal)) * delta_T)

    # Строим сигнал
    plt.figure()
    plt.xlabel('Отсчёты')
    plt.ylabel('Амплитуда')
    plt.plot(frequency, input_signal[:int(N/2)])


def signal_filtration_with_recursive_filter(signal_to_filter, A_coef, B_coef):
    """
    Фильтрация сигнала с помощью рекурсивного фильтра

    :signal_to_filter: сигнал, который нужно отфильтровать
    :A_coef: Alpha коэффициенты фильтра
    :B_coef: Beta коэффициенты фильтра
    :return: Фильтрованный сигнал
    """
    import numpy as np

    filtered_signal = []
    # Итерация по отсчётам сигнала
    for current_count in range(0, len(signal_to_filter)):
        filtered_signal.append(0)
        # Итарация по коэффициентам beta конкретного каскада
        for current_beta_coefficient in range(0, len(B_coef[0])):
            if current_count - current_beta_coefficient >= 0:
                filtered_signal[current_count] += B_coef[0][current_beta_coefficient] * \
                                    signal_to_filter[current_count - current_beta_coefficient]
        # Итерация по коэффициентам alpha конкретного каскада
        for current_alpha_coefficient in range(1, len(A_coef[0])):
            if current_count - current_alpha_coefficient >= 0:
                filtered_signal[current_count] -= A_coef[0][current_alpha_coefficient] *
\
                                    filtered_signal[current_count - current_alpha_coefficient]
```

```
    return np.array(filtered_signal)


def main():
    import numpy as np
    import matplotlib.pyplot as plt

    # Задаём характерестики сигнала, который будем генерировать
    A1 = 3
    A2 = 2

    f1 = 200
    f2 = 400.33

    signal_discritisation = np.linspace(0, N-1, N) * delta_T

    # Генерируем сигналы
    lambd = 1
    signal_in_bandwith = signal_generate(signal_discritisation, {A2:f2})
    signal_out_of_bandwith = signal_generate(signal_discritisation, {A1:f1})
     gaussian_noise_signal = signal_in_bandwith + lambd * np.random.normal(0, 1,
N)

    # Вычисляем частоту
    k = np.linspace(0, q, q)
    frequency = k/(q * delta_T)

    # Формируем массив коэффициентов
    coefficient = []
    for frequency_number in frequency:
        coefficient.append(complex(0, 2 * np.pi * frequency_number * delta_T))

    # Расчёт теоритического рекурсивного фильтра по передаточной функции
                                                    filter_transfer_function =
recursive_transfer_function(np.exp(np.array(coefficient)), fv=410, fn=390)

    # Генереируем фильтр
     recursive_filter, A, B = generate_recursive_filter(np.exp(np.array(coefficient)),
fv=410, fn=390)

    plt.figure()
    plt.plot(signal_in_bandwith)

        # Сравниваем теоритическое АЧХ рекурсивного фильтра и АЧХ
```

```
рассчитаного фльтра
    plt.figure()
        plt.plot(frequency[:int(q/2)], 20 * np.log10(abs(filter_transfer_function))[:int(q/2)], label='Фильтр, полученный из аналогового прототипа')
        plt.plot(frequency[:int(q/2)], 20 * np.log10(abs(recursive_filter))[:int(q/2)], label='Рассчитанный рекурсивный фильтр')
    plt.title('Сравнение АЧХ фильтрова')
    plt.xlabel('Частота, Гц')
    plt.ylabel('Коэффициент подавления, дБ')
    plt.xscale('log')
    plt.legend()

    # Выставляем точку смещения фильтрованного сигнала и длинну выборки, которая будет выводиться
    shift_point = 220
    range_start = 0
    range_stop = 1000
    plot_range = range(range_start, range_stop)

    # Фильтруем сигнал без помех и строим графики
                                                    filtered_usuall_signal = signal_filtration_with_recursive_filter(signal_in_bandwith, A, B)
        compare_signals(signal_in_bandwith, filtered_usuall_signal, plot_range, shift_point)

    # Фильтруем сигнал с гармонической помехой и строим графики
                                                filtered_harmonic_noise_signal = signal_filtration_with_recursive_filter(signal_out_of_bandwith, A, B)
        compare_signals(signal_out_of_bandwith, filtered_harmonic_noise_signal, plot_range, shift_point)

    # Фильтруем сигнал с гауссовским шумом и строим графики
                                                filtered_gaussian_noise_signal = signal_filtration_with_recursive_filter(gaussian_noise_signal, A, B)
        compare_signals(gaussian_noise_signal, filtered_gaussian_noise_signal, plot_range, shift_point)

    plt.show()


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