РАБОТА №8. ПОСТРОЕНИЕ НЕРЕКУРСИВНЫХ ЦИФРОВЫХ ФИЛЬТРОВ МЕТОДОМ ЧАСТОТНОЙ ВЫБОРКИ И ИХ ПРИМЕНЕНИЕ

Цель работы: получение навыков проектирования и применения нерекурсивных цифровых фильтров.

Планируемая продолжительность: от 2 до 4 академических часов.

Тип работы:  с использованием компьютерных средств.

Теоретические основы

Цифровым фильтром называют математическое выражение, применив к некоторому сигналу которое можно получить копию данного сигнала, в которой подавлена та или иная область частот, причем и сигнал, и его копия являются цифровыми величинами (массивы дискретных значений). Такие фильтры можно реализовать и аппаратно, так же, как и аналоговые, но из-за растущей сложности структуры с увеличением порядка чаще прибегают к программной реализации фильтра, и уже программу записывают в некое вычислительное устройство, обрабатывающее входной сигнал.

Цифровые фильтры могут реализовывать те же выражения, что и аналоговые (Баттерворта, Кауэра, Бесселя и т.п.), но, вместе с тем, позволяют получить и фильтры, передаточная характеристика которых задана произвольно (буквально можно вручную по точкам задать идеальную для конкретного случая характеристику и получить выражение для ее реализации).

Цифровые фильтры в самой широкой классификации принято делить на рекурсивные (используют обратную связь) и нерекурсивные (не используют обратную связь). Их также называют БИХ (с бесконечной импульсной характеристикой) и КИХ (с конечной импульсной характеристикой) фильтрами.

В рамках данной работы рассмотрим нерекурсивные цифровые фильтры. Основной характеристикой таких фильтров является импульсная характеристика:

$$h_k = [h_0, h_1, \ldots, h_{M-1}],$$

где [] – обозначение массива дискретных значений;

M – число точек в импульсной характеристике.

Применить фильтр с импульсной характеристикой h_k к сигналу s из N точек можно, воспользовавшись следующим алгоритмом работы фильтра:

$$y_m = \sum_{i=0}^{N+2(M-1)} s_i \cdot h_{m-i},$$

который, фактически, является сверткой импульсной характеристики фильтра и входного сигнала.

Проектирование нерекурсивного фильтра в Smath Studio

Для реализации нерекурсивного ЦФ в Smath Studio сперва вновь зададим зависимости и изменим график построения функции из предыдущей работы.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy.fft import fft, fftfreq
%matplotlib widget
```

Рис. 8.1. Импорт зависимостей

```
def plot(*args, y=None, stem=False):
    ax = plt.figure()
    if y is None:
        for y in args:
            x = np.arange(y.size)
            if stem:
                plt.stem(x, y)
            else:
                plt.plot(x, y)
    else:
        if stem:
            plt.stem(args[0], y)
        else:
            plt.plot(args[0], y)
    plt.grid(True)
    plt.ylabel('y')
    plt.xlabel('x')
    plt.show()
```

Рис. 8.2. Функция построения графиков

Так как в данной работе она используется для фильтрации, назовем ее filtr, в отличие от прошлой работы.

```
def filtr(array_1: np.ndarray,
          array_2: np.ndarray) -> np.ndarray:
    return signal.convolve(array_1, array_2, mode="full")
```

Рис.8.3. Задание свертки для фильтрации

Рассмотрим работу фильтра на примере. Зададим сигнал, состоящий из двух частот.

```
def y(x: np.ndarray) -> np.ndarray:
    return 10 * np.sin(9 * 2 * np.pi * x) + 5 * np.sin(2 * 2 * np.pi * x)


N = 33
DX = 1 / N
x = np.arange(start=1, stop=N)
sig = y(x * DX)
plot(sig)
```

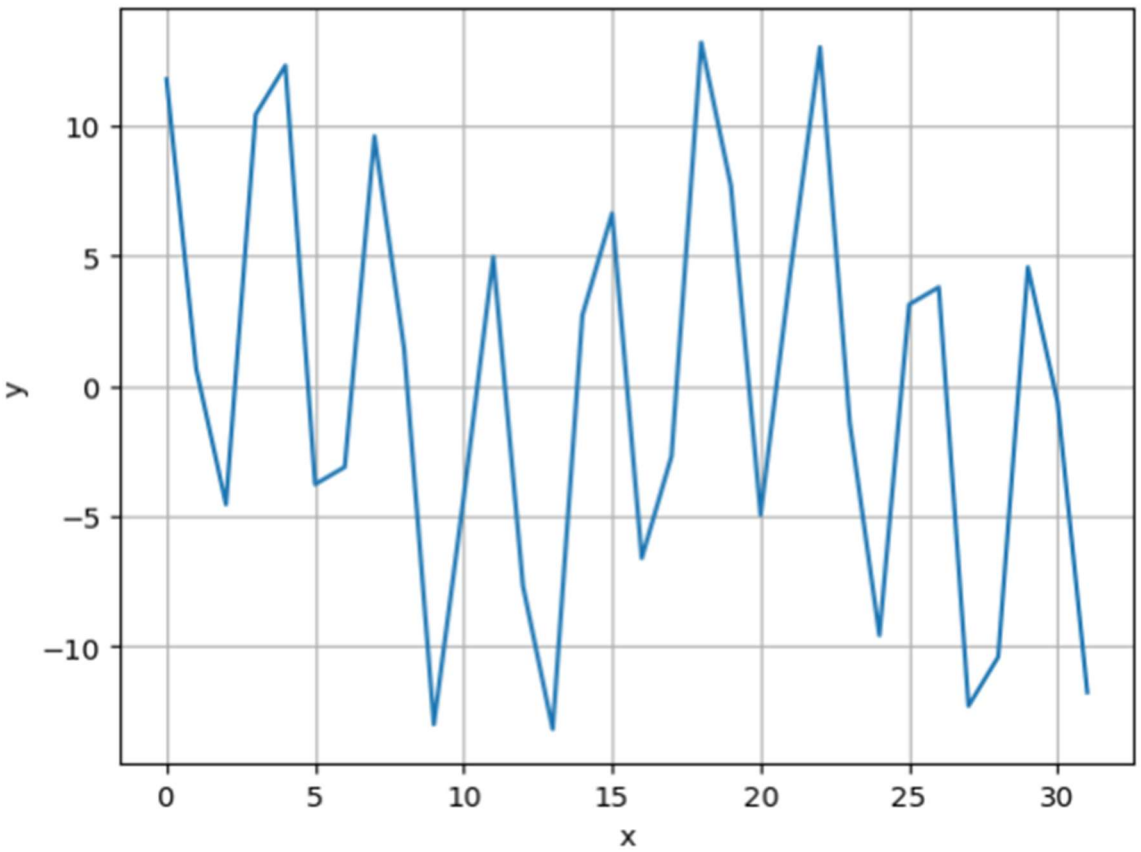

Рис. 8.4. Тестовый сигнал

А также фильтр с импульсной характеристикой, приведенной ниже.

```
IR = np.array([0.4366, 0.3943, 0.1416, -0.0793, -0.0924,
               0.0327, 0.0889, -0.0068, -0.1109, -0.0311, 0.2264])
plot(IR)
```

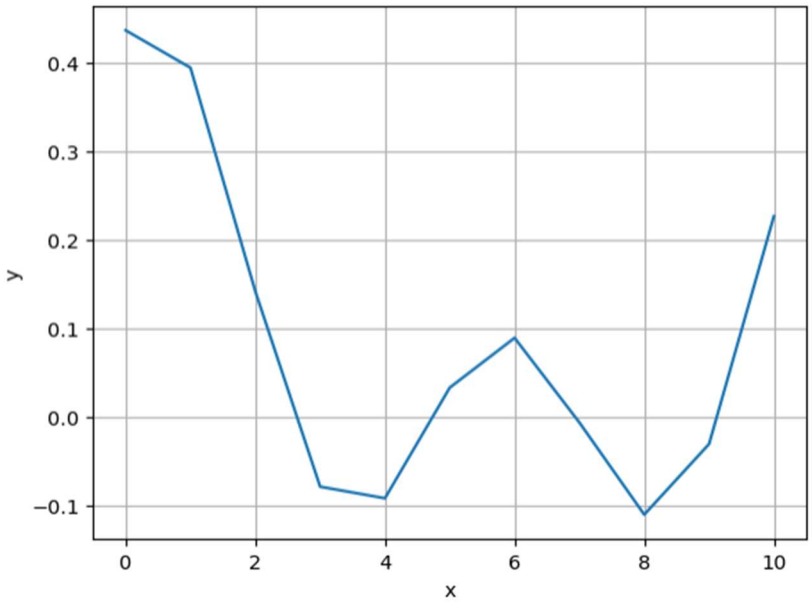

Рис. 8.5. Импульсная характеристика

Теперь построим график сигнала до и после фильтрации.

```
plot(sig, filtr(sig, IR))
```

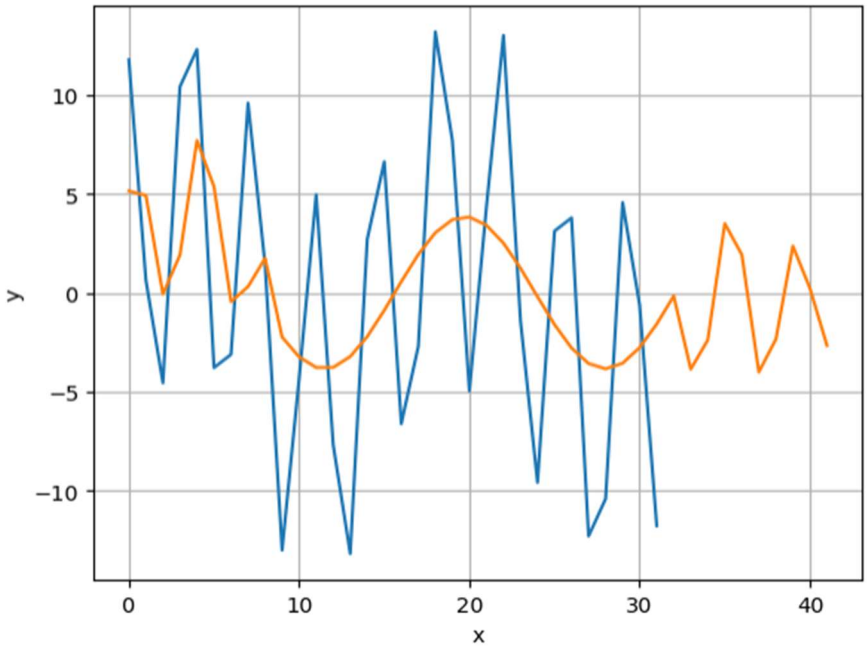

Рис. 8.6. Результат фильтрации

Синим – сигнал до фильтрации, жёлтым – после. Можно видеть, что в средней части сигнал после фильтрации практически идеально гладкий, тогда как по краям имеются существенные искажения. Это обусловлено тем, что по краям не все значения импульсной характеристики фильтра участвуют в свертке, так как они не накладываются на соответствующие значения сигнала, в средней же части участвуют все значения импульсной характеристики.

Зная, как применять фильтр к тому или иному сигналу, логично поставить вопрос о том, как же формировать импульсную характеристику фильтра, чтобы иметь возможность получить подавление на нужной частоте, ведь сама по себе импульсная характеристика не несет информации о частоте среза. Для этого можно поступить достаточно просто: задать желаемую частотную характеристику фильтра и определить на ее основе требуемую импульсную.

Алгоритм работы здесь следующий: задать дискретные отсчеты требуемой амплитудно-частотной (АЧХ) и фазо-частотной характеристики (ФЧХ) фильтра, после чего подставить ФЧХ в качестве аргумента в комплексную экспоненту, перемножить АЧХ на комплексную экспоненту в степени ФЧХ и найти ОДПФ данного произведения. Полученный результат и будет требуемой импульсной характеристикой.

$$h_k = iF_k\left(|H(j\omega)| \cdot e^{j\phi}\right),$$

где h_k - k-ое значение импульсной характеристики фильтра;

$iF_k(\cdots)$ - k-ое значение ОДПФ;

$|H(j\omega)|$ - АЧХ;

ϕ - ФЧХ;

j – мнимая единица.

Введём переменную частоты в отдельную переменную FMAX равную 10. После чего нужно задать две функции, первая будет сигналом, вторая пример отфильтрованной функции, и выведем эти две функции на графике.

```
1  def y(x: np.ndarray) -> np.ndarray:
2      return 3 * np.sin(FMAX * 2 * np.pi * x) + 2 * np.sin((FMAX / 4) * 2 * np.pi * x)
```

Рис. 8.7. Функция для фильтрации

```
1  def y_filterred_reference(x: np.ndarray) -> np.ndarray:
2      return 2 * np.sin((FMAX / 4) * 2 * np.pi * x)
```

Рис. 8.8. Примерный отфильтрованный сигнал

```
1  N = 30
2  T = 1
3  dx = T / N
4  fd = 1 / dx
5  x = np.arange(start=1, stop=N)
6  sig = y(x * dx)
7  filtered_reference = y_filterred_reference(x * dx)
8  plot(sig, filtered_reference)
```

Рис. 8.9. Создание массива сигнала

Желтый сигнал – примерный, отфильтрованный должен стремится к его виду, синий – это сигнал, который требуется очистить от высокочастотной помехи.

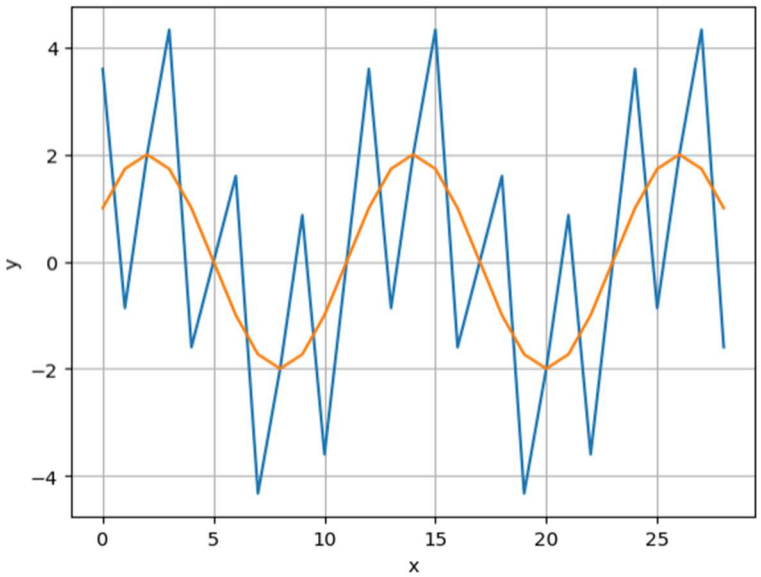

Рис. 8.10. График сигналов

Зададим спектр сигнала sig, позволяющих получить график спектра от 0 до fд/2 частот (fд – частота дискретизации), поскольку зеркальная часть модуля преобразования Фурье физического смысла не несет.

```
1  yf = fft(sig)
2  xf = fftfreq(N, dx)[:N // 2]
3  plot(xf, y=2.0 / N * np.abs(yf[0:N // 2]))
```

Рис. 8.11. Вычисление спектра и его отображение

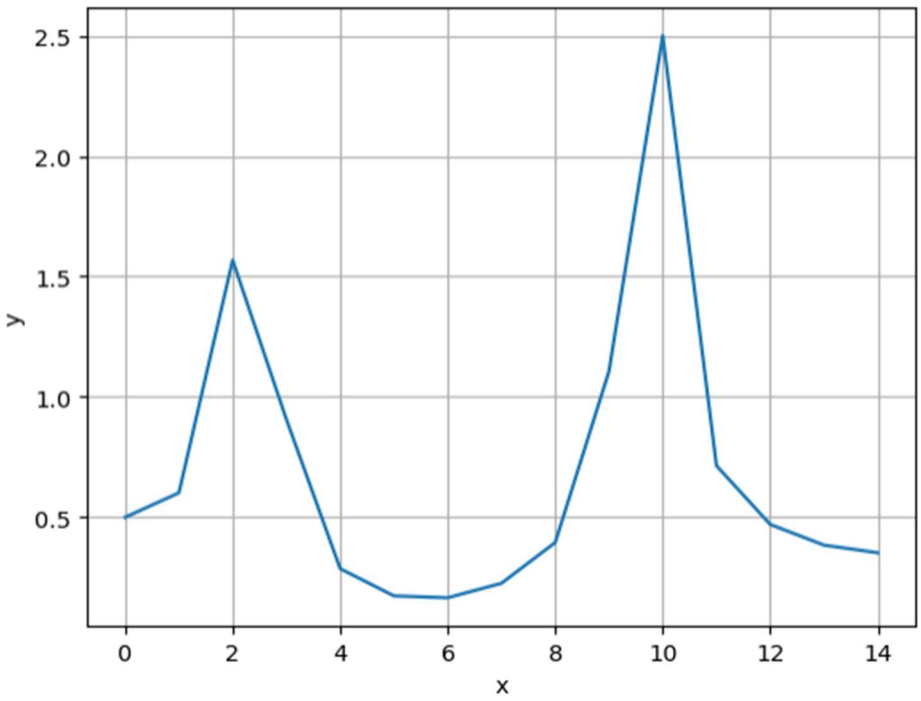

Рис. 8.12. График спектра sig

Сперва зададим функцию, которая формировала бы массив, состоящий из взятых по модулю элементов исходного массива, принимаемого в качестве параметра (это нужно, чтобы найти модуль ДПФ). В дальнейших примерах в рамках данной работы эта функция будет названа absa(ar).

Спроектируем частотную характеристику, соответствующую требованиям. Следует сразу отметить, что при проектировании

характеристики необходимо сохранить зеркальную часть модуля ДПФ, поскольку после умножения на экспоненту в степени ФЧХ результат подвергнется ОДПФ, и если отбросить область, соответствующую отрицательным частотам, то ОДПФ даст неадекватный результат.

```
1  Ha = np.array([1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1])
2  plot(Ha)
```

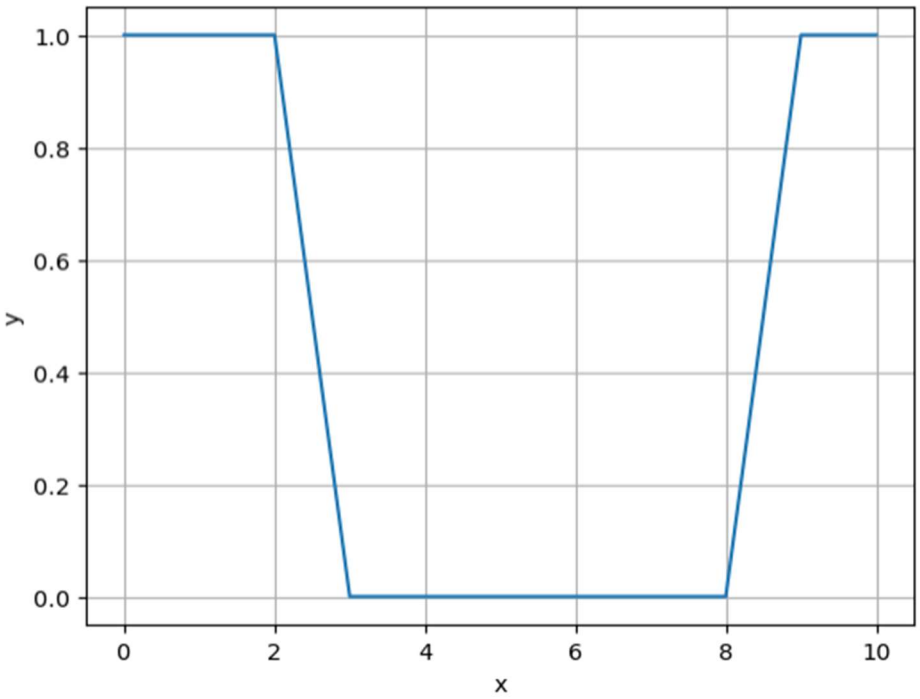

Рис. 8.13. Частотная характеристика фильтра

Еще раз напомним, что приведенный выше график (как и массив, его задающий), характеризует как фильтр будет реагировать на те или иные частоты. Последнему элементу массива (11-му) соответствует fd, первому – 0. Так как в данном примере fd=30 Гц, получим, что расстояние между соседними отсчетами по оси x имеет размер ~2,7 Гц. Число точек АЧХ фильтра равно числу точек его импульсной характеристики и выбирается пользователем (чем больше точек, тем дольше обработка, но тем точнее можно задать частоты на АЧХ). В данном случае 11 точек вполне хватает.

Если возникают проблемы при построении этой характеристики, можно посмотреть подробнее, как был получен данный график. При

известной fd и подавляемой/пропускаемой частоте можно построить АЧХ идеального ФНЧ, который бы устранял 10 Гц, сохраняя 2 Гц. Построим данную теоретическую АЧХ.

```
1  Ha = np.array([1, 0.9, 0.7, 0, 0, 0, 0, 0, 0, 0, 0])
2  plot(Ha)
```

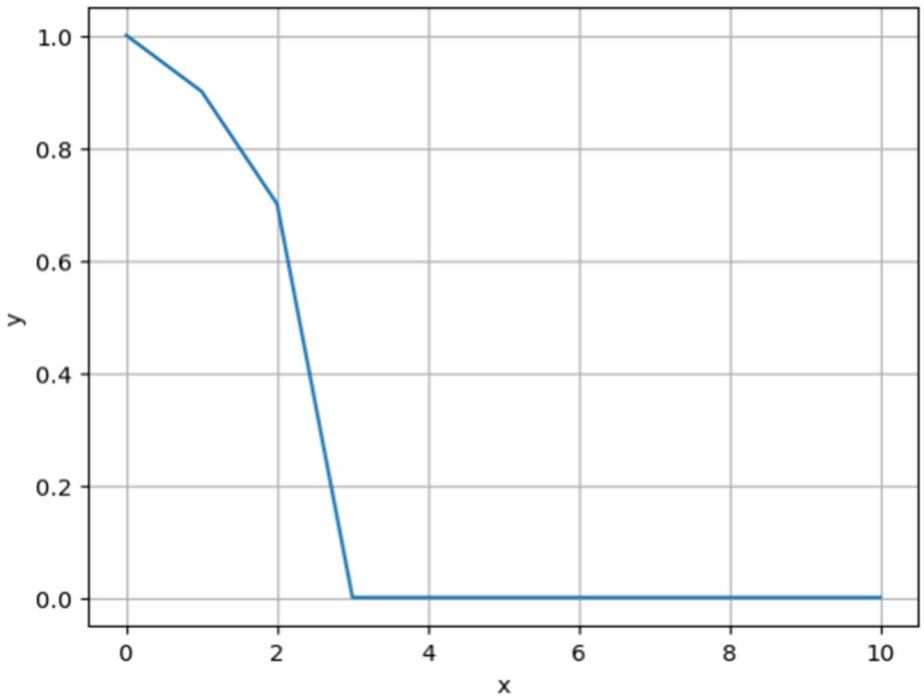

Рис. 8.14. Теоретическая АЧХ

Характеристика обращается в нуль на четвертом отсчете, который соответствует 2,7*3=~8 Гц, причем левее – плавно спадающая область пропускания, правее – ровная область подавления.

Достроим правую часть характеристики, отразив все значения характеристики, кроме первого. Отраженные элементы будут располагаться в следующем порядке – второй станет последним, третий – предпоследним и так далее. Первый элемент не учитывается потому, что он характеризует постоянную составляющую (соответствует нулевой частоте). В результате получим следующую характеристику, идентичную приведенной ранее.

```
Ha = np.array([1, 0.9, 0.7, 0, 0, 0, 0, 0, 0, 0.7, 0.9])
plot(Ha)
```

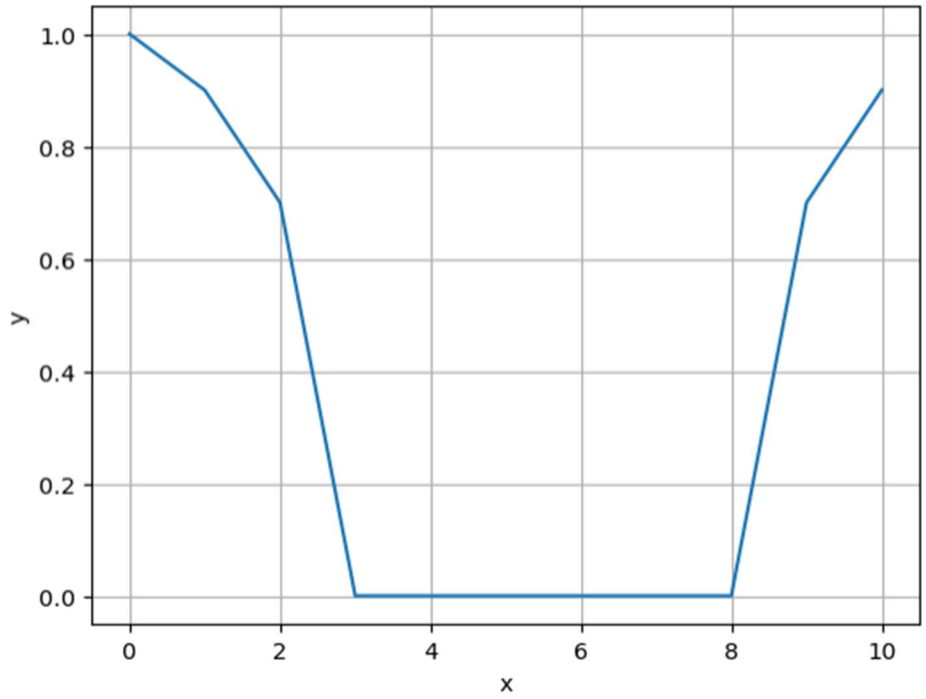

Рис.8.15. Дополненная АЧХ

Теперь зададим ФЧХ. Так как по заданию требований к ней нет, но без нее алгоритм реализован быть не может, поступаем наиболее простым способом – задаем линейную ФЧХ.

```
Fa = np.array([0, -0.2, -0.4, -0.6, -0.8, -1, 1, 0.8, 0.6, 0.4, 0.2])
plot(Fa)
```

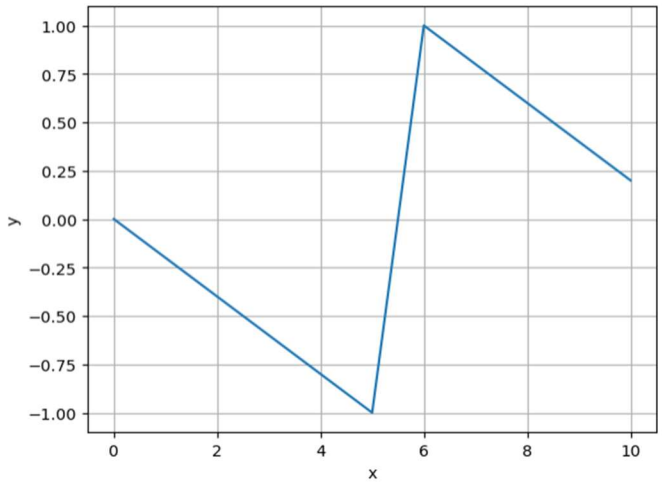

Рис.8.16. Теоретическая ФЧХ

График ФЧХ является не просто зеркальным, по аналогии с АЧХ, но и инвертированным в правой части. Такую ФЧХ можно использовать и при заданном варианте.

Имеем массивы ФЧХ и АЧХ, можем начать восстанавливать импульсную характеристику. Для этого, сперва, сформируем массив, элементы которого равны комплексной экспоненте в степени ФЧХ.

```
1  def arexp(array: np.ndarray) -> np.ndarray:
2      result = np.zeros(array.size)
3      for idx in range(result.size):
4          result[idx] = np.exp(array[idx] * 1j)
5      return result
```

```
1  Fe = arexp(Fa)
```

Рис.8.17. Массив экспоненты

Рассчитаем и отобразим отфильтрованный массив с изначальным и примерным. В данном случае синий сигнал – это фильтруемый массив, желтый – отфильтрованный, зелёный – примерный отфильтрованный

```
1  IR = np.fft.ifft(Ha * Fe)
2  plot(sig, filtr(sig, IR), filtered_reference)
```

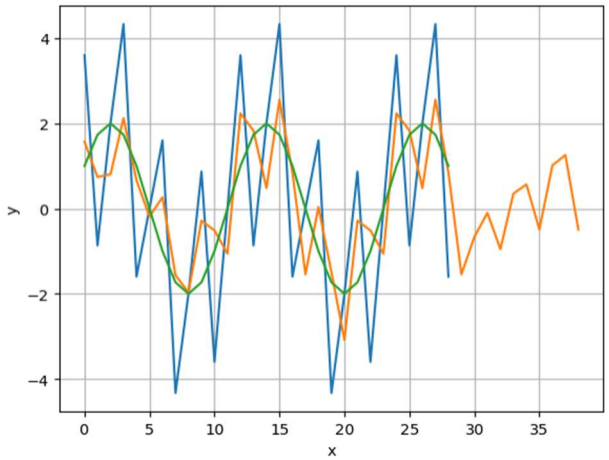

Рис.8.18. Фильтрация массива

Видим, что помеха подавлена, но далеко не полностью. Проверяем АЧХ фильтра.

```
1  plot(np.abs(np.fft.fft(IR)))
```

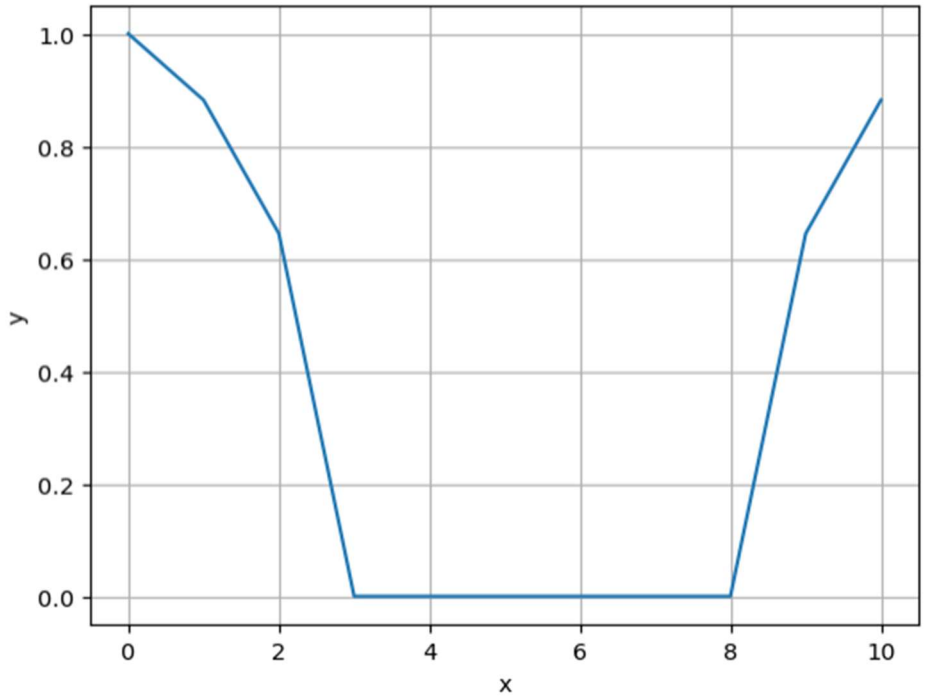

Рис. 8.19. АЧХ фильтра

Она совпадает заданной, значит все выполнено правильно.

Суть данной проблемы (подавления с меньшей амплитудой) в том, что проявляется так называемый эффект Гиббса. АЧХ рассчитанного фильтра (жёлтый график), точно проходит через узлы дискретизации, однако между точками дискретизации сильно отличается от идеальной характеристики (синий график). При этом наблюдается сильная неравномерность АЧХ в полосе пропускания фильтра, и высокий уровень боковых лепестков в полосе заграждения.

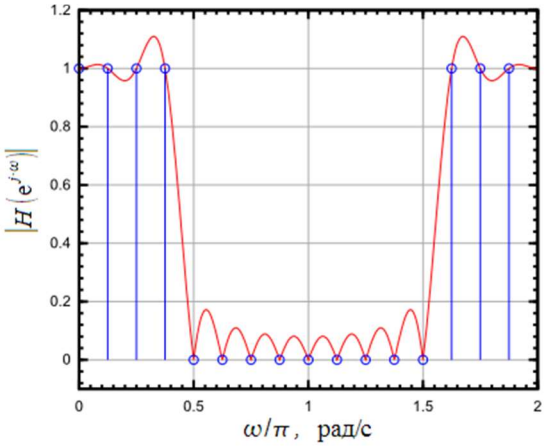

Рис.8.20. Эффект Гиббса

Поскольку 10 Гц не попадает точно в точку, где коэффициент усиления равен нулю, помеха подавляется не полностью. Эта проблема решается либо проектированием фильтра с более плавной АЧХ в местах разрывов, либо увеличением порядка фильтра (тогда пульсации остаются, но их амплитуда во много раз меньше), либо проектированием такой АЧХ, чтобы подавляемая частота точно попадала в одно из значений.

Так как последний способ достаточно просто реализуется, попробуем прибегнуть к нему. Поскольку фильтр состоит из 11 точек, а в той форме, в которой сигнал был задан выше, частота дискретизации равна числу точек в сигнале, чтобы точно попасть на АЧХ в некоторую частоту, следует сделать размер сигнала кратным числу точек фильтра (на практике, когда фильтр и сигнал имеют большее число значений, это не обязательно). Меняем N с 30 на 33.

Чтобы узнать, какой точке АЧХ соответствует та или иная частота, зададим итератор *l* и выражение, для формирования массива частот, соответствующих точкам импульсной характеристики.

```
1  fir = np.ceil(np.arange(IR.size + 1) * (fd / IR.size))
2  plot(np.abs(np.fft.fft(IR)), stem=True)
3  fir
```

Рис. 8.21. Массив частот

Теперь построим АЧХ фильтра и вызовем рядом значения массива частот, чтобы сопоставить данные.

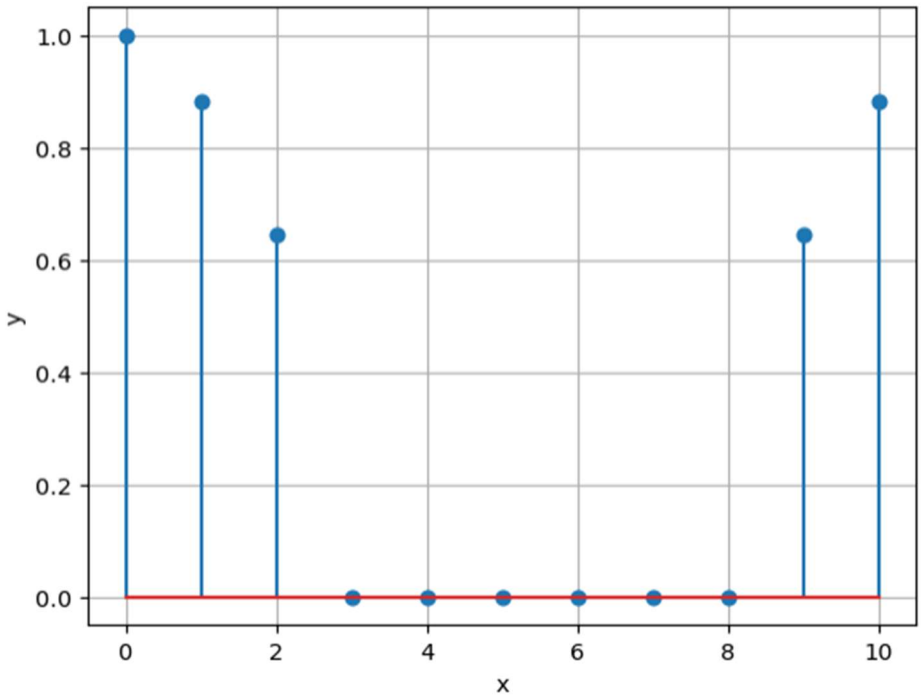

Рис. 8.22. АЧХ фильтра с частотами

По графику видно, что подавление начинается с 4 точки АЧХ (первое значение в нуле). В массиве частот видим, что полностью подавить фильтр с такой АЧХ может частоты 9 Гц, 12 Гц, 15 Гц (на частотах выше располагается зеркальная часть ДПФ).

Проверим работу фильтра. Изменим частоту помехи fmax на 25 и проверим результат.

```
1  N = 33
2  FMAX = 25
3  x = np.arange(start=1, stop=N)
4  sig = y(x * dx)
5  filtered_reference = y_filterred_reference(x * dx)
6  plot(sig, filtr(sig, IR), filtered_reference)
```

Рис. 8.23. Фильтрация данных

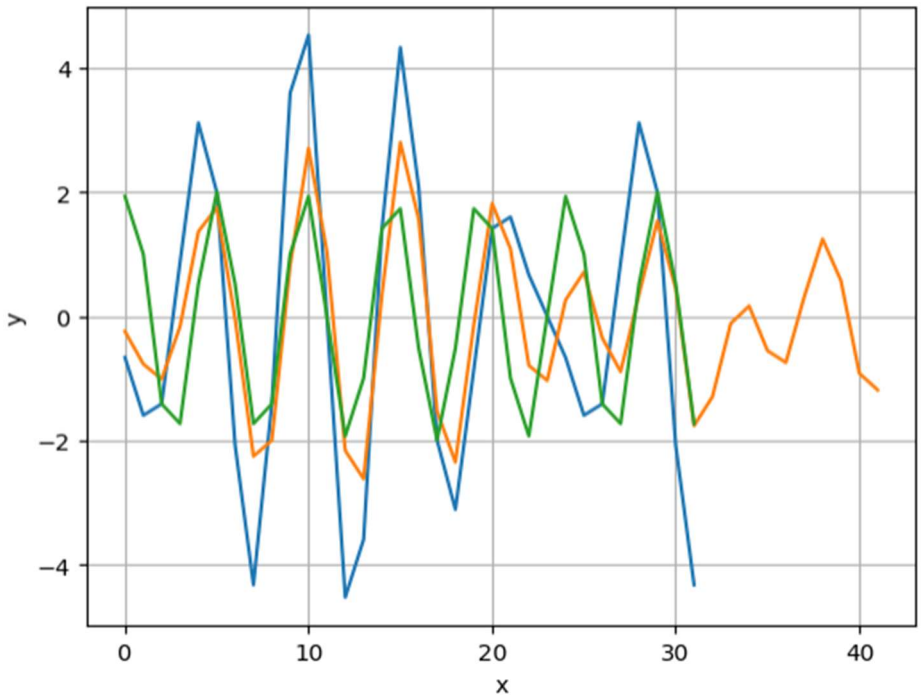

Рис. 8.24. График данных

Видно, что в средней части, где задействованы все элементы импульсной характеристики фильтра, помехи подавляется полностью, так как наш примерный зелёный сигнал и жёлтый отфильтрованный совпадают. Что говорит о том, что правильный подбор числа точек в фильтре на практике позволяет создавать простые и довольно точные алгоритмы устранения помехи.

Задание

1. Повторить приведенный выше пример.
2. Изменить указанные в таблице ниже выражения в соответствии с заданными значениями;
3. Спроектировать фильтр для подавления слагаемого на частоте fmax в сигнале без подавления второго слагаемого.

Таблица 11.1

Варианты заданий

Вариант	Выражения
1,6,11, 16,21,26	$fmax := 12$ $y(x) := 10 \cdot \sin(fmax \cdot 2 \cdot \pi \cdot x) + 2 \cdot \sin(9 \cdot 2 \cdot \pi \cdot x)$
2,7,12, 17,22,27	$fmax := 15$ $y(x) := 10 \cdot \sin(fmax \cdot 2 \cdot \pi \cdot x) + 2 \cdot \sin(12 \cdot 2 \cdot \pi \cdot x)$
3,8,13, 18,23,28	$fmax := 9$ $y(x) := 10 \cdot \sin(fmax \cdot 2 \cdot \pi \cdot x) + 2 \cdot \sin(6 \cdot 2 \cdot \pi \cdot x)$
4,9,14, 19,24,29	$fmax := 6$ $y(x) := 10 \cdot \sin(fmax \cdot 2 \cdot \pi \cdot x) + 2 \cdot \sin(3 \cdot 2 \cdot \pi \cdot x)$
5,10,15, 20,25,30	$fmax := 9$ $y(x) := 10 \cdot \sin(fmax \cdot 2 \cdot \pi \cdot x) + 2 \cdot \sin(3 \cdot 2 \cdot \pi \cdot x)$