МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «МЭИ»

Институт Радиотехники и электроники им. В.А. Котельникова

Кафедра Физики им. В.А. Фабриканта

ОТЧЁТ

по лабораторной работе № 5

Тема: Основы спектральной фильтрации изображений

Дисциплина: Цифровая фильтрация изображений.

Преподаватель _____ Поройков А.Ю.

(оценка, подпись)

Студент гр. ЭР-03-18 _____ Крючков Н.В.

(подпись)

Москва 2022

Цель работы:

Целью работы является знакомство с процедурой выполнения спектральной фильтрации изображений с помощью двумерного дискретного преобразования Фурье, получения опыта работы с низкочастотными и высокочастотными фильтрами.

Задание:

1. Задать последовательность значений (сигналы для последующей обработки), соответствующих следующим формулам:

$$y(t) = \cos(0{,}5N \cdot t) + N \cdot \sin[t + (2N + 1)t]; \quad (1)$$

$$y(t) = \begin{cases} \frac{1}{20N}, & |t| \le 10N \\ 0, & |t| > 10N \end{cases}, \quad (2)$$

где N – номер студента в журнале, t – аргумент функции.

На рисунке 1 представлен код программы, который задает последовательности, описываемые формулами (1 – 2).

```
6   N = 7
7   t = np.linspace(0, 220, 220)
8   y_1 = np.cos(0.5 * N * t) + (N * np.sin(t + (2*N + 1)*t))
9   y_2 = [0.05 * N if abs(i) <= 10 * N else 0 for i in t]
```

Рисунок 1 – Код программы, описывающий последовательности

2. Реализовать алгоритм прямого и обратного одномерного преобразования Фурье в Python (готовые функции из библиотек *numpy* и подобных не использовать).

Формула для прямого дискретного преобразования Фурье выглядит следующим образом:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn}. \quad (3)$$

Формула для обратного дискретного преобразования Фурье выглядит следующим образом:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn}. \quad (4)$$

На рисунке 2 представлен блок кода, который описывает алгоритм прямого дискретного преобразования Фурье согласно формуле (3).

```
19  def fourier_transformation_forward(centered_image):
20      empty_complex_array = np.zeros(len(centered_image), dtype=complex)
21      centered_image_length = len(centered_image)
22      for k in range(centered_image_length):
23          for n in range(centered_image_length):
24              empty_complex_array[k] += centered_image[n] * np.exp(-2 * np.pi * 1j * k * n / centered_image_length)
25      return np.array(empty_complex_array)
```

Рисунок 3 – Реализация прямого дискретного преобразования Фурье

На рисунке 3 представлен блок кода, который описывает алгоритм обратного дискретного преобразования Фурье согласно формуле (4).

```
28  def fourier_transformation_backward(filtered_array):
29      empty_complex_array = np.zeros(len(filtered_array), dtype=complex)
30      filtered_array_length = len(filtered_array)
31      for k in range(filtered_array_length):
32          for n in range(filtered_array_length):
33              empty_complex_array[k] += filtered_array[n] * np.exp(2 * np.pi * 1j * k * n / filtered_array_length)
34          empty_complex_array[k] *= 1 / filtered_array_length
35      return np.array(empty_complex_array)
```

Рисунок 4 – Реализация обратного дискретного преобразования Фурье

3. Выполнить преобразование Фурье с помощью реализованного алгоритма и с помощью готовых функций из библиотеки *numpy*. Отобразить результаты с помощью графиков. Сравнить результаты

Для того, чтобы реализовать алгоритм нам необходима функция центрирования последовательности, то есть умножение последовательности на $(-1)^{x+y}$, где $x = 0$, так как последовательность представляет собой вектор, и обратная ей функция – децентрирование. Программный код для такой функции представлен на рисунке 5.

```
12  def to_center_func(array_of_values, centered=True):
13      centered_image = []
14      for i in range(len(array_of_values)):
15          if (centered):
16              centered_image.append(array_of_values[i] * ((-1) ** i))
17          else:
18              centered_image.append(array_of_values[i].real * ((-1) ** i))
19      return np.array(centered_image)
```

Рисунок 5 – Программный код функции для центрирования и децентрирования последовательности

На рисунках (6 – 7) представлены результаты использования алгоритма, который был реализован самостоятельно и встроенных в библиотеку *numpy* функции – *np.fft.fft* для разных последовательностей.

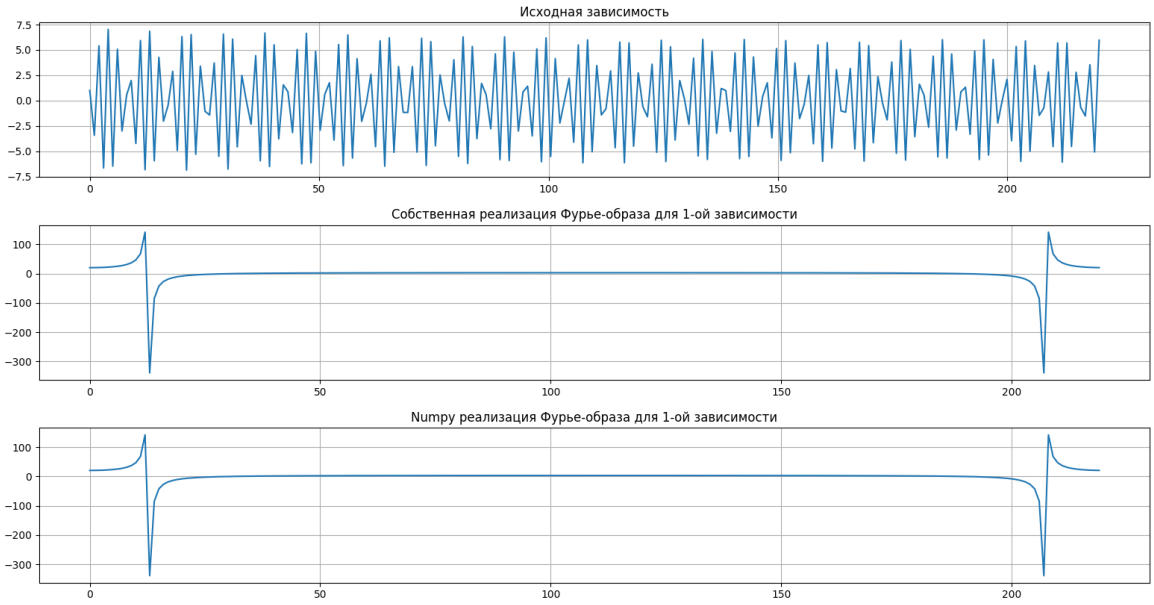

Рисунок 6 – Результат сравнения собственной разработки и встроенных функций в библиотеку *numpy* для первой последовательности

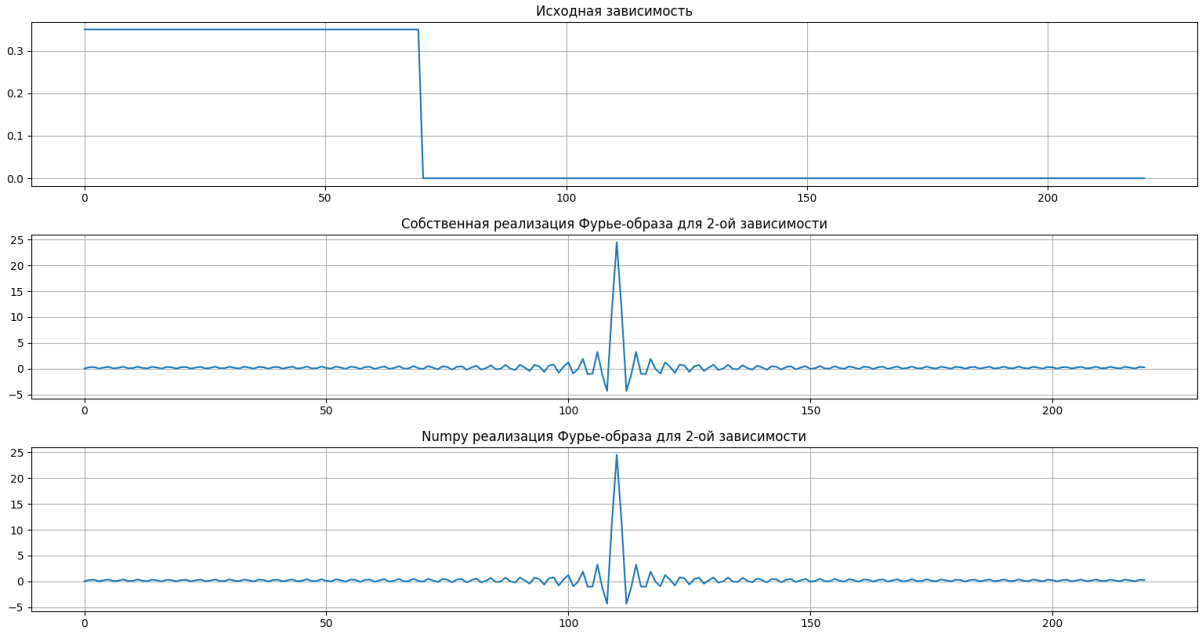

Рисунок 7 – Результат сравнения собственной разработки и встроенных функций в библиотеку *numpy* для второй последовательности

Из рисунка 6 и 7 следует, что Фурье-образ, разработанный самостоятельно, в точности совпадает с Фурье-образом, представленным в библиотеки. Можно заключить, что собственная реализация алгоритма верна.

4. Реализовать алгоритм спектральной фильтрации одномерного сигнала для случая НЧ, ВЧ и полосового фильтра.

На рисунке 8 представлена реализация для НЧ, ВЧ и полосового фильтра для одномерной последовательности.

```python
# Фильтр низких частот
def filter_fourier_lf(transformed_array, shift):
    transformed_array = transformed_array.copy()
    transformed_array_length = transformed_array.size
    transformed_array[:transformed_array_length//2 - shift] = transformed_array[
                                                              :transformed_array_length//2 - shift] * 0
    transformed_array[transformed_array_length//2 + shift + 1:] = transformed_array[
                                                                  transformed_array_length//2 + shift + 1:] * 0
    return transformed_array

# Фильтр высоких частот
def filter_fourier_hf(filtered_array, shift):
    filtered_array = filtered_array.copy()
    filtered_array_length = len(filtered_array)
    filtered_array[filtered_array_length // 2 - shift:
                   filtered_array_length // 2 + shift + 1] = 0 * filtered_array[filtered_array_length // 2- shift:
                                                                                filtered_array_length // 2 + shift + 1]
    return np.array(filtered_array)


# Полосный фильтр
def filter_fourier_pf(filtered_array, shift, freq_offset):
    filtered_array = filtered_array.copy()
    center = len(filtered_array) // 2
    filtered_array[:center - shift - freq_offset] *= 0
    filtered_array[center - shift + freq_offset + 1: center] *= 0
    filtered_array[center: center + shift - freq_offset] *= 0
    filtered_array[center + shift + freq_offset + 1:] *= 0
    return np.array(filtered_array)
```

Рисунок 8 – Реализация фильтров НЧ, ВЧ и полового фильтра для одномерной последовательности

5. Выполнить процедуру фильтрации с сигналами из п. 1. Отобразить результаты в виде графиков.

   Результат наложения фильтров на обе одномерные последовательности представлен на рисунках (9 – 12).

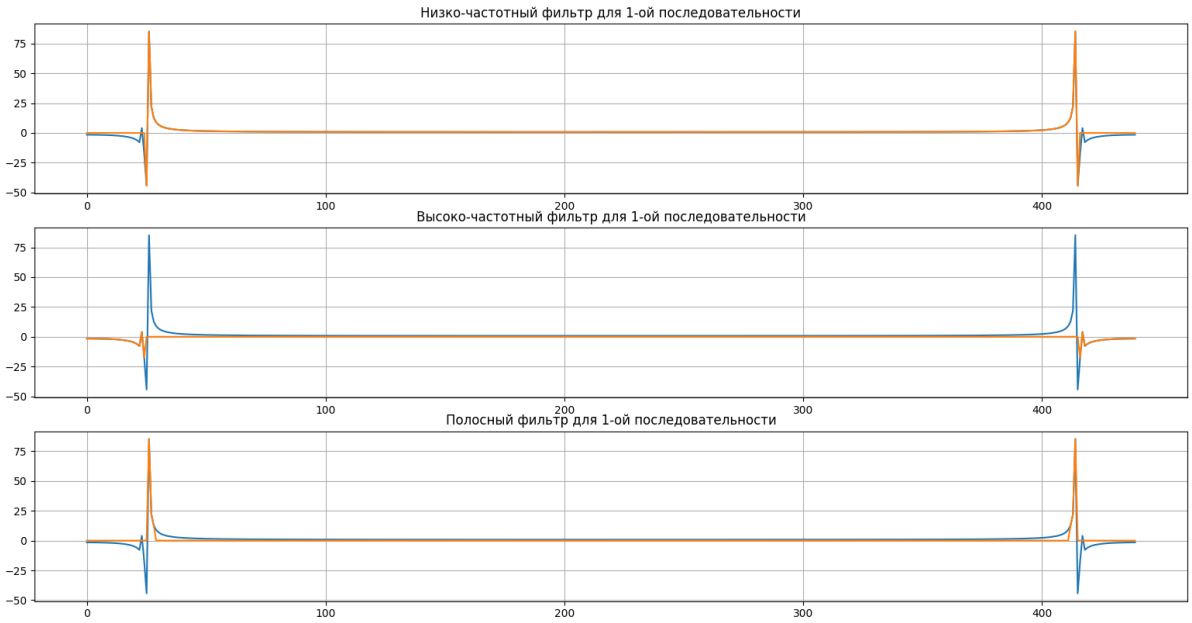

Рисунок 9 – Результат наложения НЧ, ВЧ и полосного фильтров на первого Фурье-образа

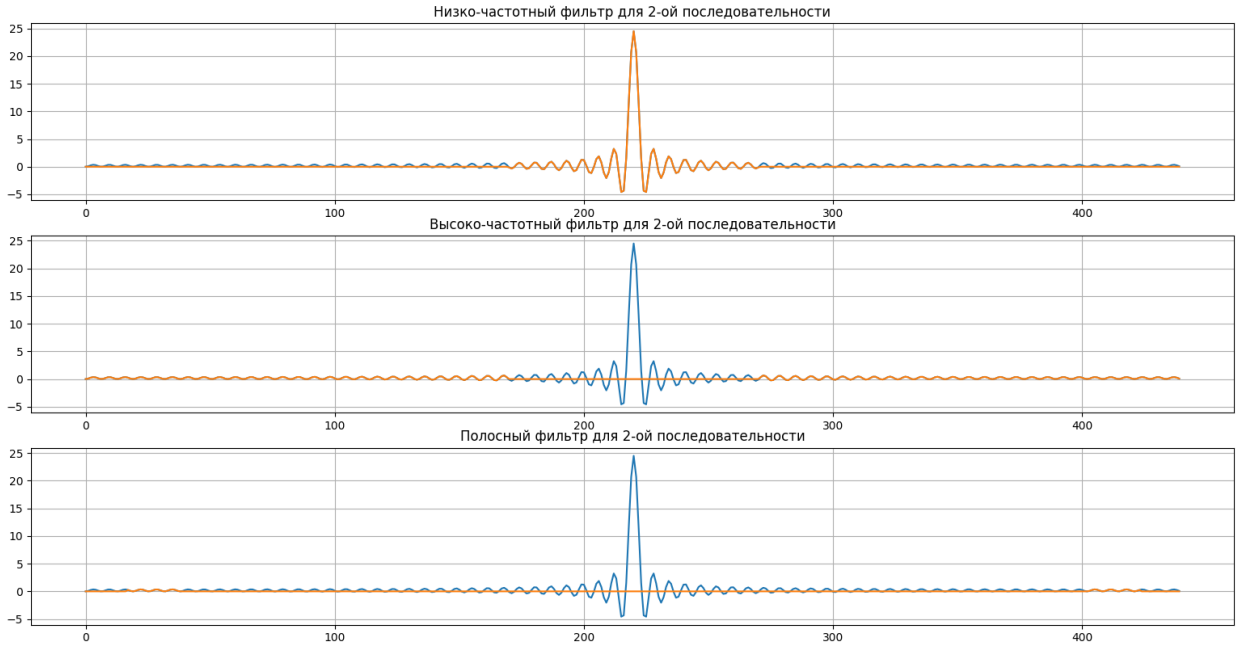

Рисунок 10 – Результат применения НЧ, ВЧ и полосного фильтров для второго Фурье-образа

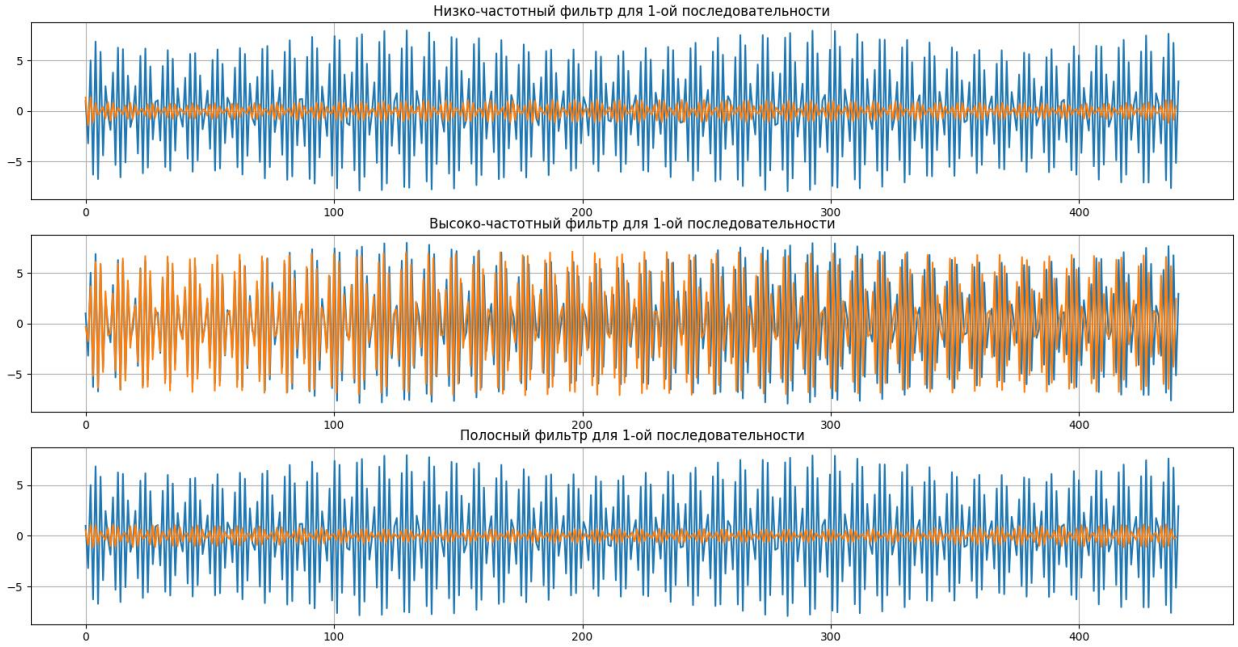

Рисунок 11 – Результат применения НЧ, ВЧ и полосного фильтров для исходной последовательности

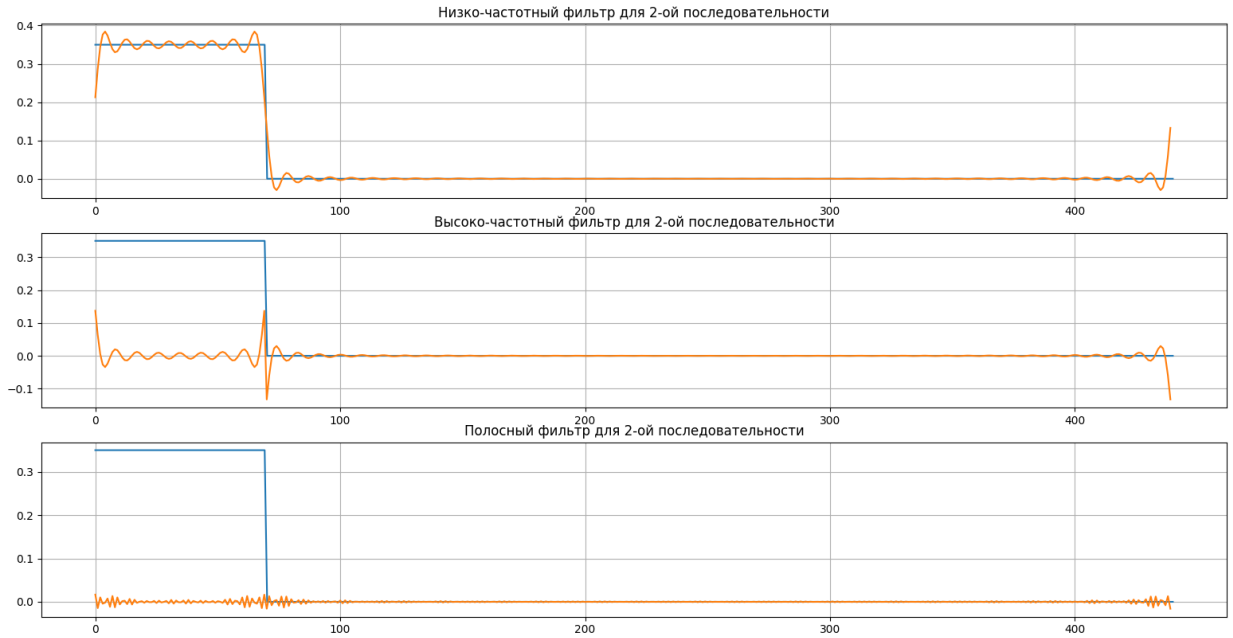

Рисунок 12 – Результат применения НЧ, ВЧ и полосного фильтров для исходной второй последовательности

6. Выполнить двухмерное Фурье преобразования с изображениями, выданными преподавателем. Отобразить полученные Фурье-образы.

Формула для двухмерного прямого преобразования Фурье выглядит следующим образом:

$$X_{kl} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x_{nm} \cdot e^{-2\pi i \left(\frac{kn}{N} + \frac{lm}{M}\right)}. \quad (5)$$

Формула для двухмерного обратного преобразования Фурье выглядит следующим образом:

$$x_{nm} = \frac{1}{NM} \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} X_{kl} \cdot e^{2\pi i \left(\frac{kn}{N} + \frac{lm}{M}\right)}. \quad (6)$$

На рисунке 13 представлена реализация алгоритма двухмерного преобразования Фурье, который был разработан самостоятельно.

```python
def fourier_transform_forward(centered_image):
    centered_image = centered_image.copy()
    (image_height, image_width) = np.shape(centered_image)
    empty_array = np.zeros(shape=(image_height, image_width), dtype=np.complex128)
    for k in range(image_height):
        for l in range(image_width):
            for m in range(image_height):
                for n in range(image_width):
                    empty_array[k][l] += (centered_image[m][n] * (np.exp(-2 * np.pi * 1j * k * m / image_height))
                                          * (np.exp(-2 * np.pi * 1j * l * n / image_width)))
    return np.array(empty_array)
```

Рисунок 13 – Собственная реализация алгоритма двухмерного преобразования Фурье

На рисунке 14 представлен результат работы алгоритма двухмерного Фурье-преобразования, который был разработан самостоятельно.

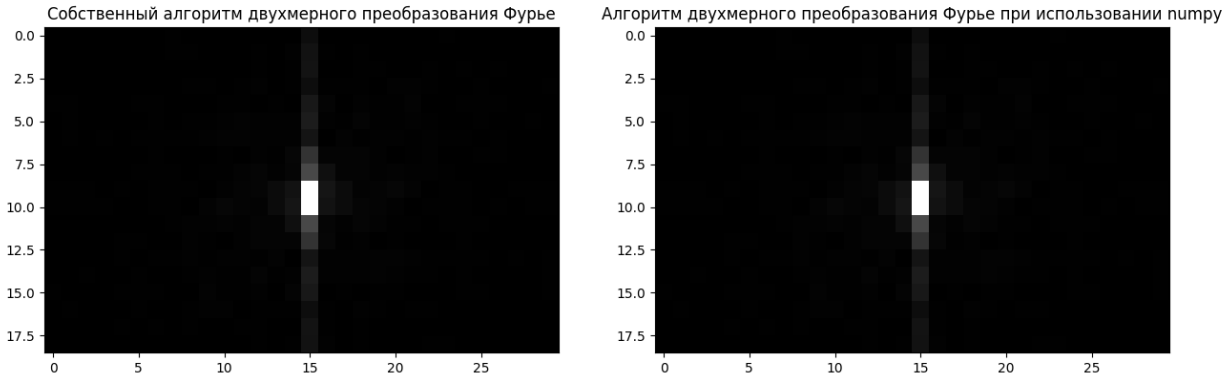

Рисунок 14 – Результат применения двухмерного Фурье преобразования

Как видно из рисунка 14 результат применения собственной реализации алгоритма двухмерного преобразования Фурье и использование встроенной функции в библиотеки *numpy* ничем не отличаются. Отсюда можно утверждать, что собственная реализация была разработана верно. В дальнейшем будем использовать встроенную функцию из-за быстрой скорости работы по сравнению с собственной реализацией.

На 15 рисунке представлено исходное изображение над котором будут проводиться манипуляции по фильтрации.

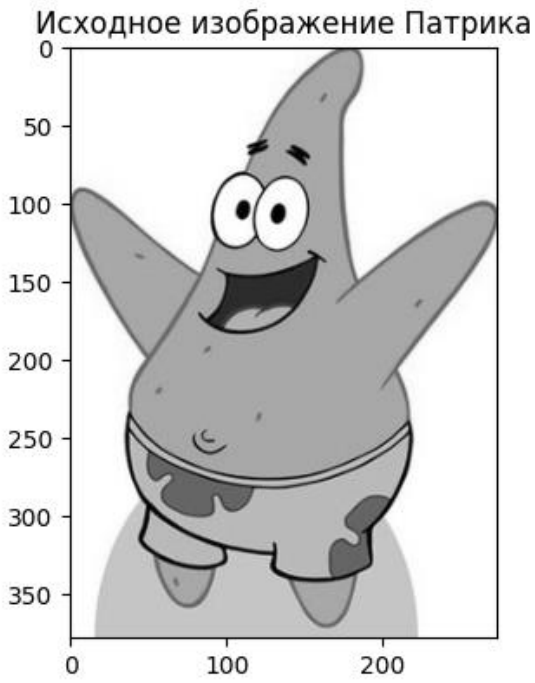

Рисунок 15 – Исходно изображение

На рисунке 16 представлена реализация НЧ, ВЧ и полосового фильтров для двухмерного случая.

```python
def filter_lf_2d(transformed_image, shift):
    transformed_image = transformed_image.copy()
    array_height, array_width = np.shape(transformed_image)
    transformed_image[:array_height // 2 - shift[0], :] *= 0
    transformed_image[array_height // 2 + shift[0] + 1:, :] *= 0
    transformed_image[:, :array_width // 2 - shift[1]] *= 0
    transformed_image[:, array_width // 2 + shift[1] + 1:] *= 0
    return np.array(transformed_image)


def filter_hf_2d(transformed_image, shift):
    transformed_image = transformed_image.copy()
    array_height, array_width = np.shape(transformed_image)
    transformed_image[array_height // 2 - shift[0]: array_height//2 + shift[0] + 1,
        array_width // 2 - shift[1]: array_width // 2 + shift[1] + 1
    ] *= 0

    return np.array(transformed_image)


def filter_pr_2d(transformed_image, *args):
    cut_freq, shift = args
    transformed_image = transformed_image.copy()
    transformed_image[: cut_freq[0] - shift[0], :] *= 0
    transformed_image[cut_freq[0] + shift[0] + 1:, :] *= 0
    transformed_image[:, :cut_freq[1] - shift[1]] *= 0
    transformed_image[:, :cut_freq[1] + shift[1] + 1] *= 0
    return np.array(transformed_image)
```

Рисунок 16 - Реализация НЧ, ВЧ и полосового фильтров для двухмерного случая

Результат применения НЧ, ВЧ и полосного фильтров представлен на рисунке 17.

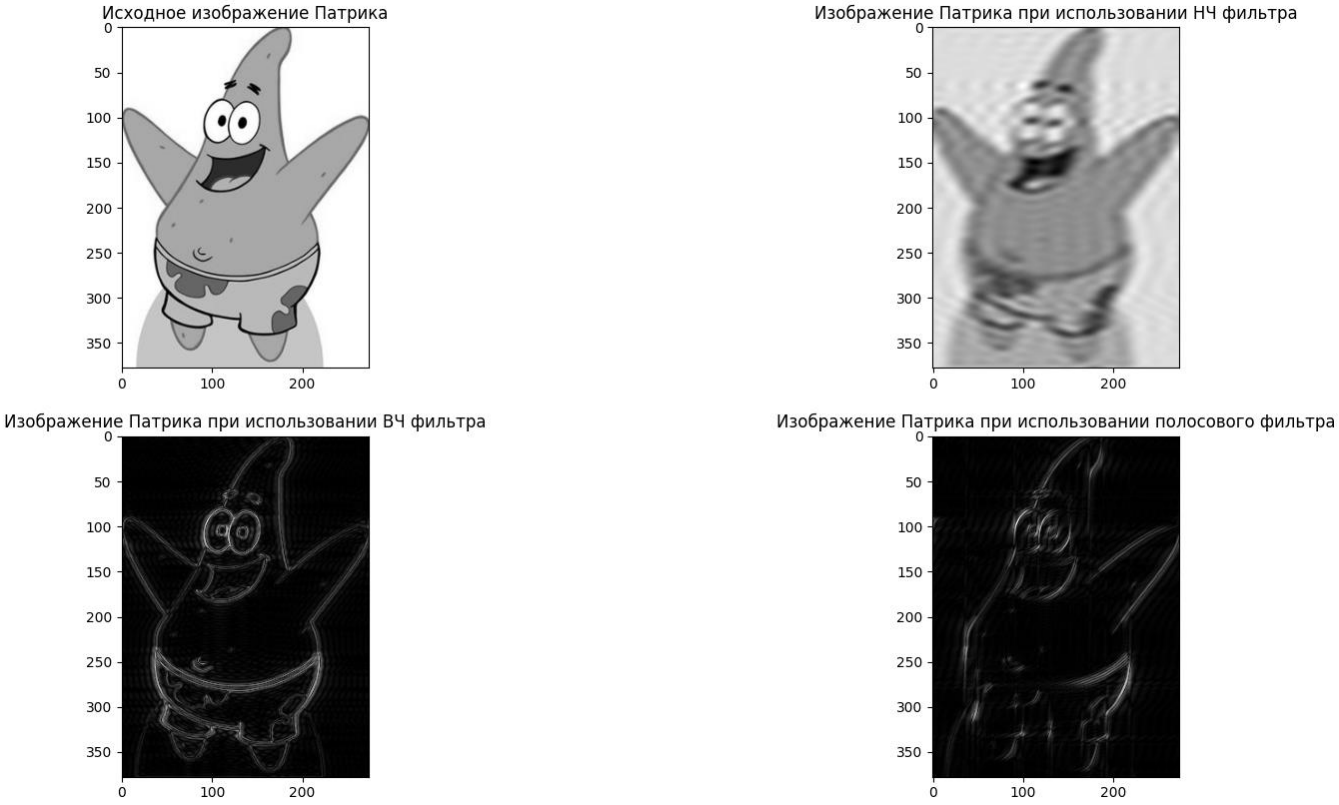

Рисунок 17 – Результат применения фильтрации к изображению

Из результатов применения фильтров, которые представлены на рисунке 17, видно, что после применения НЧ фильтрации изображение стало размываться. Применение ВЧ фильтрация уменьшает яркость гладких областей и выделяет контуры на изображении. Использование полосного фильтра как видно из рисунка 17 выделяет некоторые контуры объектов на определенной частоте.

Вывод:

В данной лабораторной работе были разработаны алгоритмы фильтрации Фурье для одномерной последовательности. К тому же, было произведено сравнение собственной реализации с уже имеющиеся в библиотеке *numpy*. Также была произведена фильтрация двух последовательностей с помощью НЧ, ВЧ и полосового фильтра.

Также был разработан алгоритм фильтрации Фурье для двухмерной последовательности, то есть изображения. Все также было произведен сравнение собственной реализации с уже имеющимися функциями в библиотеке *numpy*. В конечном итоге, были применены фильтры НЧ, ВЧ и полосового фильтра к изображению и приведены результаты с кратким объяснением влияния каждого фильтра.