Министерство образования Республики Беларусь

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОИЭЛЕКТРОНИКИ»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Лабораторная работа № 1
на тему
«Преобразование Фурье»

Выполнили: Довголёнок Д. А.
Миранович

Проверил: Третьяков А. Г.

МИНСК 2021

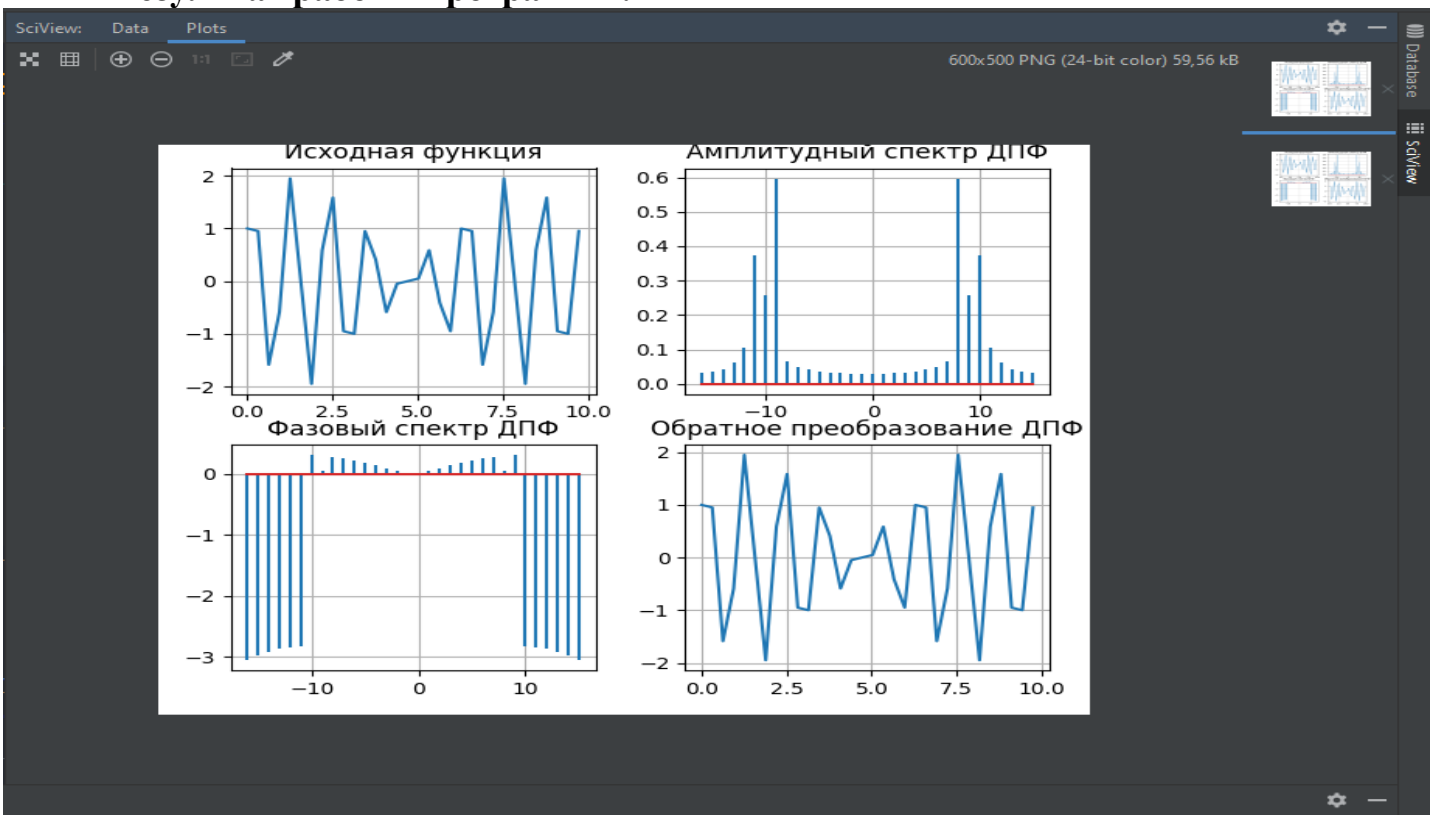
Задание:

1. Ознакомиться с теоретической частью.
2. Для заданного сигнала реализовать ДПФ и алгоритм быстрого преобразования Фурье (БПФ) (прямое и обратное преобразования). Результат работы программы – амплитудный и фазовый спектр сигнала.
3. Сравнить БПФ с методом ДПФ по вычислительной сложности (количество операций сложения и умножения).
4. Оформить отчет

Исходные данные:

1. Номер варианта: 7.
2. Функция сигнала: y=cos(5x) + sin(6x).
3. Алгоритм БПФ: БПФ с прореживанием по времени.
4. Число отсчетов N: 32.

Результат работы программы:




```
ДПФ число операций: 1024
БПФ число операций: 80
```

Данные результаты обусловлены тем, что алгоритм быстрого преобразования Фурье является ускоренным алгоритмом для вычисления дискретного преобразования Фурье за счет сокращения количества операций сложения и умножения. Это, в свою очередь, достигается благодаря использованию свойств симметрии и периодичности коэффициентов поворота

Листинг кода:

(main.py)

```
from transform import FourierTransform
import numpy as np
import matplotlib.pyplot as plt
import cmath


def main():
    n = 32
    arguments = np.arange(0, n) * np.pi / 10
    fun_values = FourierTransform.function_values(arguments)
    frequencies = [x for x in range(-int(n / 2), int(n / 2), 1)]

    dft_values = FourierTransform.dft(FourierTransform, fun_values, 1)
    i_dft_values = FourierTransform.dft(FourierTransform, dft_values, -1)

    for i in range(n):
        dft_values[i] /= n

    shifted_dft = FourierTransform.shift(dft_values)

    dft_pol = list(map(lambda x: cmath.polar(x), shifted_dft))

    dft_ph_value = []
    dft_amp_value = []

    for i in range(len(shifted_dft)):
        dft_amp_value.append(dft_pol[i][0])
        dft_ph_value.append(dft_pol[i][1])

    print('СПФ число операций:', FourierTransform.count)

    fft_values = FourierTransform.fft(FourierTransform, fun_values, 1)
    i_fft_values = FourierTransform.fft(FourierTransform, fft_values, -1)

    for i in range(n):
        fft_values[i] /= n

    shifted_fft = FourierTransform.shift(fft_values)

    fft_pol = list(map(lambda x: cmath.polar(x), shifted_fft))

    fft_ph_value = []
    fft_amp_value = []

    for i in range(len(shifted_fft)):
        fft_amp_value.append(fft_pol[i][0])
        fft_ph_value.append(fft_pol[i][1])

    print('БПФ число операций:', FourierTransform.count)

    first, dff_x_arr = plt.subplots(2, 2, figsize=(6, 5))
    plt.tight_layout()

    dff_x_arr[0, 0].plot(arguments, fun_values)
    dff_x_arr[0, 0].set_title('Исходная функция')
    dff_x_arr[0, 0].grid(True)

    dff_x_arr[0, 1].stem(frequencies, dft_amp_value, markerfmt=' ')
    dff_x_arr[0, 1].set_title('Амплитудный спектр')
    dff_x_arr[0, 1].grid(True)

    dff_x_arr[1, 0].stem(frequencies, dft_ph_value, markerfmt=' ')
    dff_x_arr[1, 0].set_title('Фазовый спектр')
```

```
    dff_x_arr[1, 0].grid(True)

    dff_x_arr[1, 1].plot(arguments, i_dft_values)
    dff_x_arr[1, 1].set_title('Обратное преобразование')
    dff_x_arr[1, 1].grid(True)

    second, fft_x_arr = plt.subplots(2, 2, figsize=(6, 5))
    plt.tight_layout()

    fft_x_arr[0, 0].plot(arguments, fun_values)
    fft_x_arr[0, 0].set_title('Исходная функция')
    fft_x_arr[0, 0].grid(True)

    fft_x_arr[0, 1].stem(frequencies, fft_amp_value, markerfmt=' ')
    fft_x_arr[0, 1].set_title('Амплитудный спектр')
    fft_x_arr[0, 1].grid(True)

    fft_x_arr[1, 0].stem(frequencies, fft_ph_value, markerfmt=' ')
    fft_x_arr[1, 0].set_title('Фазовый спектр')
    fft_x_arr[1, 0].grid(True)

    fft_x_arr[1, 1].plot(arguments, i_fft_values)
    fft_x_arr[1, 1].set_title('Обратное преобразование')
    fft_x_arr[1, 1].grid(True)

    plt.show()


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

(transform.py)

```
import numpy as np


class FourierTransform:
    count = 0

    @staticmethod
    def function_values(arguments):
        function_values = list(map(lambda x: np.cos(2*x) + np.sin(5*x), arguments))

        return function_values

    @staticmethod
    def dft(self, values, direction):
        self.count = 0

        n = len(values)
        out_values = [complex(0, 0)] * n

        for m in range(n):
            for k in range(n):
                out_values[m] += values[k] * np.exp(direction * complex(0, -1) * 2 *
np.pi * m * k / n)
                self.count += 1
            if direction == -1:
                out_values[m] /= n

        return out_values

    @staticmethod
    def fft(self, in_values, direction):
        self.count = 0
        out_values = self.fft_realise(self, in_values, direction)
```

```
        if direction == -1:
            n = len(in_values)
            for i in range(n):
                out_values[i] /= n

        return out_values

    @staticmethod
    def fft_realise(self, values, direction):
        n = len(values)

        if n == 1:
            return values

        values_even = [complex(0, 0)] * (int(n / 2))
        values_odd = [complex(0, 0)] * (int(n / 2))

        for i in range(n):
            if i % 2 == 0:
                values_even[int(i / 2)] = values[i]
            else:
                values_odd[int(i / 2)] = values[i]

        b_even = self.fft_realise(self, values_even, direction)
        b_odd = self.fft_realise(self, values_odd, direction)

        w_n = complex(np.exp(direction * 2 * np.pi * complex(0, -1) / n))
        w = 1
        y = [complex(0, 0)] * n

        for i in range(int(n / 2)):
            y[i] = b_even[i] + b_odd[i] * w
            y[i + int(n / 2)] = b_even[i] - b_odd[i] * w
            w = w * w_n
            self.count += 1

        return y

    @staticmethod
    def shift(values):
        length = len(values)

        first_half = values[0:int(length / 2)]
        second_half = first_half[:]
        first_half.reverse()
        shifted_result = first_half + second_half

        return shifted_result
```