Министерство образования Республики Беларусь

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОИЭЛЕКТРОНИКИ»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Лабораторная работа № 2
на тему
«Операции свертки и корреляции»

Выполнили: Басько А. С.
Какадей С. В.

Проверил: Третьяков А. Г.

МИНСК 2021

Задание:

1. Ознакомьтесь с теоретической частью.

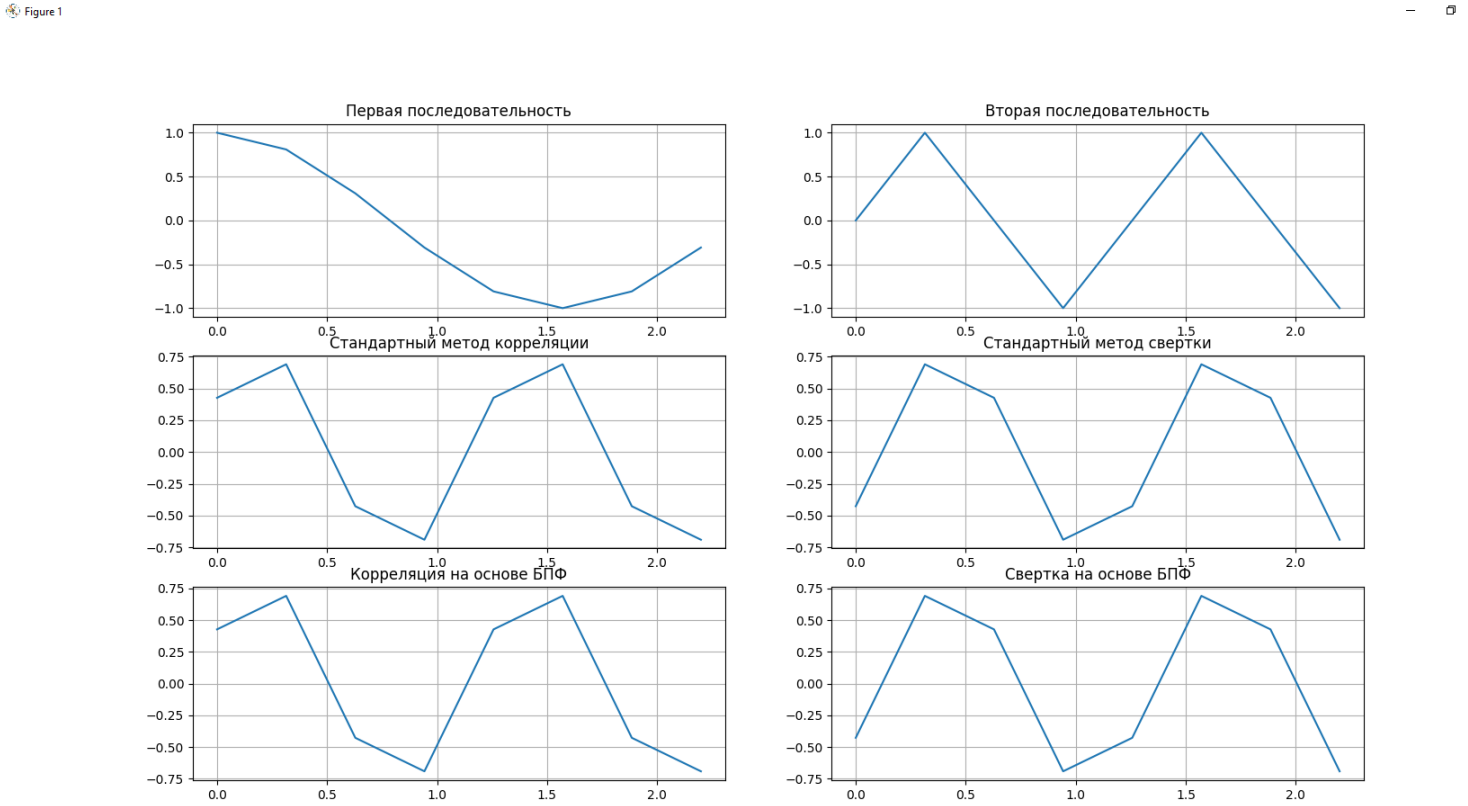
2. Для заданных сигналов реализовать: операцию свертки и операцию корелляции в соответствии с формулами; операцию свертки на основе БПФ и операцию корреляции на основе БПФ. Вывести исходные сигналы, результаты свертки и корреляции

3. Сравните вычислительную сложность вычисления свертки и корреляции предложенными алгоритмами.

Исходные данные:

1. Номер варианта: 3.
2. Заданная функция: y=cos(2x); z=sin(5x)
3. N для БПФ: 8.

Результат работы программы:




```
Сложность обычной корреляции: 64
Сложность обычной свертки: 64
Сложность корреляции на основе БПФ: 44
Сложность свертки на основе БПФ: 44
```

Листинг кода:

(main.py)

```
from operations import Operations
import numpy as np
import matplotlib.pyplot as plt


def main():
    n = 8
    arguments = np.arange(0, n) * np.pi / 10
    function_values_1 = list(map(lambda x: np.cos(2 * x), arguments))
    function_values_2 = list(map(lambda x: np.sin(5 * x), arguments))

    basic_correlation = Operations.basic_function(function_values_1, function_values_2,
1)
    print('Сложность обычной корреляции: {}'.format(Operations.counter))
    basic_convolution = Operations.basic_function(function_values_1, function_values_2, -
1)
    print('Сложность обычной свертки: {}'.format(Operations.counter))

    fft_based_correlation = Operations.fft_based_function(function_values_1,
function_values_2, 1)
    print('Сложность корреляции на основе БПФ: {}'.format(Operations.counter))
    fft_based_convolution = Operations.fft_based_function(function_values_1,
function_values_2, -1)
    print('Сложность свертки на основе БПФ: {}'.format(Operations.counter))

    _, represent = plt.subplots(3, 2)

    represent[0, 0].plot(arguments, function_values_1)
    represent[0, 0].set(title='Первая последовательность')
    represent[0, 0].grid(True)

    represent[0, 1].plot(arguments, function_values_2)
    represent[0, 1].set(title='Вторая последовательность')
    represent[0, 1].grid(True)

    represent[1, 0].plot(arguments, basic_correlation)
    represent[1, 0].set(title='Стандартный метод корреляции')
    represent[1, 0].grid(True)

    represent[1, 1].plot(arguments, basic_convolution)
    represent[1, 1].set(title='Стандартный метод свертки')
    represent[1, 1].grid(True)

    represent[2, 0].plot(arguments, fft_based_correlation)
    represent[2, 0].set(title='Корреляция на основе БПФ')
    represent[2, 0].grid(True)

    represent[2, 1].plot(arguments, fft_based_convolution)
    represent[2, 1].set(title='Свертка на основе БПФ')
    represent[2, 1].grid(True)

    plt.show()


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

(transform.py)

```
import numpy as np


class Transform:
    counter = 0

    @staticmethod
```

```
    def fft(self, in_values, direction):
        self.counter = 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.counter += 1

        return y
```

(operations.py)

```
from transform import Transform
import numpy as np


class Operations:
    counter = 0

    @staticmethod
    def basic_function(first_sequence, second_sequence, operation):
        if operation != 1 and operation != -1:
            raise Exception('Номер операции должен быть равен'
                            '1(корреляция) или -1(свертка). Получено
{}!'.format(operation))

        length = len(first_sequence)

        if length != len(second_sequence):
            raise Exception("Длина последовательностей должна быть одинаковой!")

        Operations.counter = 0

        result = []
```

```
        for i in range(length):
            temp = 0
            for j in range(length):
                k = (i + (j * operation)) % length
                temp += first_sequence[j] * second_sequence[k]

                Operations.counter += 1

            result.append(temp)

        return result

    @staticmethod
    def fft_based_function(first_sequence, second_sequence, operation):
        if operation != 1 and operation != -1:
            raise Exception('Номер операции должен быть равен'
                            '1(корреляция) или -1(свертка). Получено
{}!'.format(operation))

        length = len(first_sequence)

        if length != len(second_sequence):
            raise Exception("Длина последовательностей должна быть одинаковой!")

        Operations.counter = 0

        first_fft = Transform.fft(Transform, first_sequence, 1)
        Operations.counter += Transform.counter

        second_fft = Transform.fft(Transform, second_sequence, 1)
        Operations.counter += Transform.counter

        processed_first_fft = first_fft
        if operation == 1:
            processed_first_fft = np.conj(first_fft)

        temp = np.multiply(processed_first_fft, second_fft)
        Operations.counter += length

        result = Transform.fft(Transform, temp, -1)
        Operations.counter += Transform.counter

        return result
```