МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра математического анализа

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ ПОДГОТОВКЕ ПО ДИСЦИПЛИНЕ «ГАРМОНИЧЕСКИЙ АНАЛИЗ»

студента 1 курса 118 группы

направления 01.04.02 Прикладная математика и информатика

код и наименование направления

механико-математического факультета

наименование факультета, института, колледжа

Веселова Андрея Сергеевича

фамилия, имя, отчество

Преподаватель:

профессор, д.ф.-м.н.		С. Ф. Лукомский
должность, уч. степень, уч. звание	подпись, дата	инициалы, фамилия

Саратов 2021

СОДЕРЖАНИЕ

Стр.

ВВЕДЕНИЕ

Быстрое преобразование Фурье (БПФ, FFT) — это быстрый алгоритм, $\mathcal{O}(N \log(N))$, для вычисления дискретного преобразования Фурье (ДПФ, DFT), которое напрямую вычисляется за $\mathcal{O}(N^2)$. Основная идея БПФ заключается в разделении вектора коэффициентов на два вектора, рекурсивном вычислении ДПФ для них, и объединении результатов в одно БПФ.

В данной работе рассмотрим теорию по БПФ, алгоритм и программу на $C\#$ 7($.Net$ 4.7.2), которая реализует прямое и обратное БФП.

1 Быстрое преобразование Фурье

Введем несколько вспомогательных определений и теорем.
Пусть $N \in \mathbb{N}$ фиксировано. Определим функции

$$e_n(x) = e^{2\pi i n \frac{j}{2^N}}, \text{ если } x \in \Delta_j^{(N)}, j = \overline{0, 2^N - 1}, n = \overline{0, 2^N - 1}. \tag{1.1}$$

Теорема 1. Функции $e_n(x)$, $n = \overline{0, 2^N - 1}$, определенные в (1.1), образуют ортонормированную систему на $[0, 1)$, состоящую из двоично-ступенчатых функций.

Теорема 2. Любая ступенчатая функция $f(x)$, постоянная на интервалах $\Delta_j^{(N)}$, $x = \overline{0, 2^N - 1}$ единственным образом представима в виде

$$f(x) = \sum_{n=0}^{2^N-1} c_n e_n(x). \tag{1.2}$$

Любая функция $f \in \mathfrak{D}_{2^N}$, полностью определяется своими значениями $f_j = f(\Delta_j^{(N)})$. Поэтому задание функции f равносильно заданию ее вектора значений f_j. В данном случае вектор $(f_0, f_1, \ldots, f_{2^N-1})$ называют сигналом.

Определим функции $e_n(x)$ на дискретном множестве E,
$E = \{0, 1, 2, \ldots, 2^N - 1\}$. Равенство (1.1) примет следующий вид:

$$e_n(j) = \frac{1}{2^{\frac{N}{2}}} e^{2\pi i n \frac{j}{2^N}}, \ j = \overline{0, 2^N - 1} \tag{1.3}$$

и вектор $\big(e(0), e(1), \ldots, e(2^N - 1)\big)$ является сигналом.

Определение 1. Любая функция f, определенная на E, с комплексными значениями $f(j) = \lambda_j^{(N)}$ называется сигналом. Т.о. любой сигнал f представим в виде:

$$f(j) = \sum_{n=0}^{2^N-1} \hat{f}(n) e_n(j). \tag{1.4}$$

2 Алгоритм

Введем следующее преобразование:

$$\begin{aligned} \lambda_{2j} &:= \tfrac{1}{\sqrt{2}}\left(\lambda_j + \lambda_{j+2^{N-1}}\right) \\ \lambda_{2j+1} &:= \tfrac{1}{\sqrt{2}} e^{-\frac{2\pi i j}{2^N}}\left(\lambda_j - \lambda_{j+2^{N-1}}\right) \end{aligned} \tag{2.1}$$

Пусть $\lambda_0, \lambda_1, \ldots, \lambda_j, \ldots, \lambda_{2^n-1}$– значения функции f на полуинтервалах $\Delta_j^{(N)}$. Применим преобразование (2.1) и на первой итерации получим две последовательности:

с четными номерами $(\lambda_0, \lambda_2, \ldots, \lambda_{2^N-2}) = (\lambda_{2n})\,, n = \overline{0, 2^{N-1}-1}$,

с нечетными номерами $(\lambda_1, \lambda_3, \ldots, \lambda_{2^N-1}) = (\lambda_{2n+1})\,, n = \overline{0, 2^{N-1}-1}$.

Применим преобразование (2.1) к $(\lambda_{2n})\,,\ (\lambda_{2n+1})\,, n = \overline{0, 2^N-1}$ (с четными номерами $n = 2j$, с нечетными $- n = 2j+1$) и на второй итерации получим $4(2^2)$ последовательности: $(\lambda_{2^2j}), (\lambda_{2^2j+1}), (\lambda_{2^2j+2}), (\lambda_{2^2j+3}),\ j = \overline{0, 2^{N-2}-1}$.

И т. д. На n-ом шаге имеем получили 2^n последовательностей: $\{(\lambda_{2^nj+\nu})\}_{\nu=0}^{2^n-1}$, $j = \overline{0, 2^{N-n}-1}$.

Преобразование (2.1) запишем в виде:

$$\lambda_j \mapsto \begin{cases} \lambda_{2j} := \tfrac{1}{\sqrt{2}}\left(\lambda_j + \lambda_{j+2^{N-1}}\right) \\ \lambda_{2j+1} := \tfrac{1}{\sqrt{2}} e^{-\frac{2\pi i j}{2^N}}\left(\lambda_j - \lambda_{j+2^{N-1}}\right) \end{cases} \tag{2.2}$$

К каждой последовательности $\{(\lambda_{2^nj+\nu})\}_{\nu=0}^{2^n-1}$, $j = \overline{0, 2^{N-n}-1}$, применим преобразование (2.2), получим:

$$\lambda_{2^nj+\nu} \mapsto \begin{cases} \lambda_{2^n2j+\nu} := \tfrac{1}{\sqrt{2}}\left(\lambda_{2^nj+\nu} + \lambda_{2^n(j+2^{N-n-1})+\nu}\right) \\ \lambda_{2^n(2j+1)} := \tfrac{1}{\sqrt{2}} e^{-\frac{2\pi i j}{2^{N-n}}}\left(\lambda_{2^nj+\nu} - \lambda_{2^n(j+2^{N-n-1})+\nu}\right). \end{cases} \tag{2.3}$$

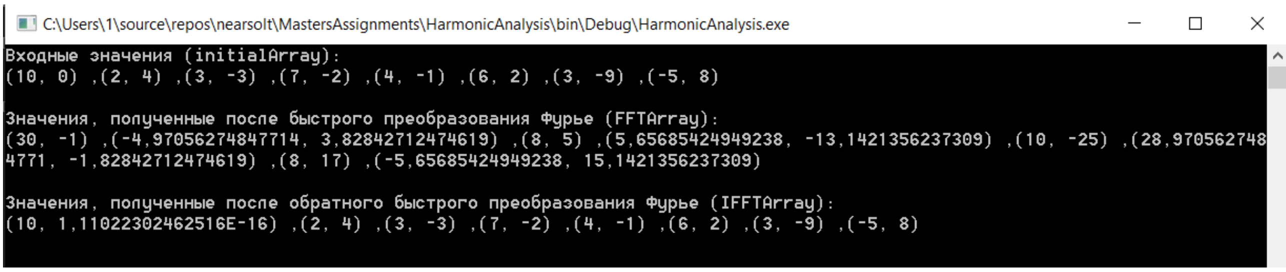
Вычисления начинаем с $n = 0$ и заканчиваем при $n = N-1$. При каждом $n, \nu = \overline{0, 2^N-1}$ применяем формулы (2.3).

3 Реализация алгоритма

В листингах приведены два класса: HarmonicAnalysis.cs – включает в себя точку входа в программу и в методе Main указываем массив значений, который будет передаваться в метод БПФ; Helpers.cs – класс, включающий в себя метод БПФ и вспомогательные для него методы.

В методе Main зададим массив комплексных чисел initialArray: $initialArray = \{10, 2+4i, 3-3i, 7-2i, 4-i, 6+2+, 3-9i, -5+8i\}$. Далее данный массив передаем в метод FFT, у которого вторым параметром указывается булево значение inverse: false - прямое преобразование, true - обратное. Результат записываем в массив FFTArray. Затем передаем в метод FFT с inverse=true для получения обратного преобразования и записываем значения в новый массив IFFTArray.

В результате применения прямого и обратного преобразований получили следующие значения:



ПРИЛОЖЕНИЕ А

Листинг: Быстрое преобразование Фурье

A.1 HarmonicAnalysis.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Numerics;

namespace HarmonicAnalysis {
    class HarmonicAnalysis {

        #region task 1: FFT/IFFT Main
        static void Main(string[] args) {

            Complex[] initialArray = new Complex[] { new Complex(10, 0), new Complex(2, 4), new Complex(3, -3), new Complex(7, -2),
                                                    new Complex(4, -1), new Complex(6, 2), new Complex(3, -9), new Complex(-5, 8) };

            if (!Helpers.IsPowerOfTwo(initialArray.Length)) {
                Console.WriteLine(string.Format("Количество элементов массива {0} равно {1}, что не является степенью двойки, введите корректные данные",
                                                nameof(initialArray), initialArray.Length));
                Console.ReadKey();
                return;
            }
            Console.WriteLine(string.Format("Входные значения ({0}): {1}{2}{1}",
                                            nameof(initialArray), Environment.NewLine, string.Join(" ,", initialArray.ToArray())));

            Complex[] FFTArray = Helpers.FFT(initialArray, initialArray.Length);
            Console.WriteLine(string.Format("Значения, полученные после быстрого преобразования Фурье ({0}): {1}{2}{1}",
                                            nameof(FFTArray), Environment.NewLine, string.Join(" ,", FFTArray.ToArray())));

            Complex[] IFFTArray = Helpers.FFT(FFTArray, FFTArray.Length, true);
```

```
                Console.WriteLine(string.Format("Значения, полученные после обратного быстрого преобразования Фурье ({0}): {1}{2}{1}",
                                                nameof(IFFTArray), Environment.NewLine, string.Join(", ", IFFTArray.ToArray())));

                Console.ReadLine();
            }
            #endregion
    }
}
```

A.2 Helpers.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Numerics;

namespace HarmonicAnalysis {
    class Helpers {

        #region task 1: FFT/IFFT

        /// <summary>
        /// Вычисление поворачивающего модуля e^(-i*2*PI*k/N) при прямом, e^(i*2*PI*k/N) при обратном
        /// </summary>
        /// <param name="kиндекс"> в цикле суммы</param>
        /// <param name="Nразмерность"> массива</param>
        /// <param name="inverseфлаг">: false - прямое преобразование, true - обратное</param>
        /// <returns></returns>
        static Complex W(int k, int N, bool inverse = false) {
            if (k % N == 0) {
                return 1;
            }
            double arg = 2 * Math.PI * k / N * (inverse ? 1 : -1);
            return new Complex(Math.Cos(arg), Math.Sin(arg));
        }

        /// <summary>
        /// Быстрое преобразование Фурье прямое(, если inverse = false; обратное, если inverse = true)
        /// </summary>
```

```
        /// <param name="xмассив"> значений сигнала</param>
        /// <param name="Nразмерность"> массива, должно быть степенью 2</param>
        /// <param name="inverseфлаг">: false - прямое преобразование, true - обратное</param>
        /// <returnsмассив> со значениями спектра сигнала</returns>
        internal static Complex[] FFT(Complex[] x, int N, bool inverse = false) {
            Complex[] X;

            if (N == 2) {
                X = new Complex[2];
                X[0] = x[0] + x[1];
                X[1] = x[0] - x[1];
            } else {
                Complex[] x_even = new Complex[N / 2];
                Complex[] x_odd = new Complex[N / 2];
                for (int i = 0; i < N / 2; i++) {
                    x_even[i] = x[2 * i];
                    x_odd[i] = x[2 * i + 1];
                }
                Complex[] X_even = FFT(x_even, N / 2, inverse);
                Complex[] X_odd = FFT(x_odd, N / 2, inverse);
                X = new Complex[N];
                for (int i = 0; i < N / 2; i++) {
                    X[i] = (X_even[i] + W(i, N, inverse) * X_odd[i]) * (inverse ? 2.0 / N : 1);
                    X[i + N / 2] = (X_even[i] - W(i, N, inverse) * X_odd[i]) * (inverse ? 2.0 / N : 1);
                }
            }
            return X;
        }

        /// <summary>
        /// Центровка массива значений полученных в FFT спектральная( составляющая при нулевой частоте будет в центре массива)
        /// </summary>
        /// <param name="Xмассив"> значений, полученный в FFT</param>
        /// <returns></returns>
        internal static Complex[] NFFT(Complex[] X) {
            int N = X.Length;
            Complex[] X_n = new Complex[N];
            for (int i = 0; i < N / 2; i++) {
                X_n[i] = X[N / 2 + i];
                X_n[N / 2 + i] = X[i];
            }
```

```
            return X_n;
        }

        /// <summary>
        /// Проверка, является ли число степенью двойки (Decrement and Compare)
        /// </summary>
        /// <param name="numвводимое"> число</param>
        /// <returns></returns>
        internal static bool IsPowerOfTwo(int num) {
            return (num != 0) && ((num & (num - 1)) == 0);
        }

        #endregion
    }
}
```