**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

Факультет прикладной математики и информатики

Лабораторная работа №1

**Преобразование Фурье**

Вариант №2

Тишков Дмитрий Александрович

Минск, 2025

**Исходные данные**

Функция: y = sin(3x) + cos(x)

Количество точек: N = 16

Диапазон x: [0, 2π]

**Рекурсивный алгоритм БПФ**:

* Разделение входного массива на четные и нечетные элементы
* Рекурсивное применение БПФ к подмассивам
* Объединение результатов с использованием поворотных множителей (корней из единицы)

**Прямое БПФ**:

* Вызывается с параметром invert = false
* Вычисляет комплексный спектр сигнала

**Обратное БПФ**:

* Вызывается с параметром invert = true
* Включает дополнительное деление на 2 на каждом уровне рекурсии
* Восстанавливает исходный сигнал из частотных компонентов

**Исходный код**

#include <iostream>

#include <vector>

#include <cmath>

#include <fstream>

#include <complex>

const int N = 16;

const double PI = 3.14159265358979323846;

typedef std::complex<double> base;

void fft(std::vector<base> &a, bool invert) {

int n = (int)a.size();

if (n == 1) return;

std::vector<base> a0(n/2), a1(n/2);

for (int i=0, j=0; i<n; i+=2, ++j) {

a0[j] = a[i];

a1[j] = a[i+1];

}

fft(a0, invert);

fft(a1, invert);

double ang = 2\*PI/n \* (invert ? -1 : 1);

base w(1), wn(cos(ang), sin(ang));

for (int i=0; i<n/2; ++i) {

a[i] = a0[i] + w \* a1[i];

a[i+n/2] = a0[i] - w \* a1[i];

if (invert) {

a[i] /= 2;

a[i+n/2] /= 2;

}

w \*= wn;

}

}

void saveToFile(const std::vector<double>& data, const std::string& filename) {

std::ofstream out(filename);

for (const auto& val : data) {

out << val << "\n";

}

out.close();

}

void saveComplexToFile(const std::vector<base>& data, const std::string& filename) {

std::ofstream out(filename);

for (const auto& val : data) {

out << std::abs(val) << "\n";

}

out.close();

}

int main() {

std::vector<double> input(N);

std::vector<double> x\_values(N);

for (int i = 0; i < N; ++i) {

x\_values[i] = 2 \* PI \* i / N;

input[i] = sin(3 \* x\_values[i]) + cos(x\_values[i]);

}

saveToFile(input, "original.txt");

saveToFile(x\_values, "x\_values.txt");

std::vector<base> fft\_input(N);

for (int i = 0; i < N; ++i) {

fft\_input[i] = base(input[i], 0);

}

fft(fft\_input, false);

saveComplexToFile(fft\_input, "fft.txt");

fft(fft\_input, true);

std::vector<double> inverse(N);

for (int i = 0; i < N; ++i) {

inverse[i] = fft\_input[i].real();

}

saveToFile(inverse, "inverse.txt");

return 0;

}

Визуализация результатов  
import numpy as np

import matplotlib.pyplot as plt

original = np.loadtxt('original.txt')

x\_values = np.loadtxt('x\_values.txt')

fft\_mag = np.loadtxt('fft.txt')

inverse = np.loadtxt('inverse.txt')

plt.figure(figsize=(12, 8))

plt.subplot(3, 1, 1)

plt.plot(x\_values, original, 'b-o')

plt.title('Original Signal: y = sin(3x) + cos(x)')

plt.xlabel('x')

plt.ylabel('y')

plt.grid()

plt.subplot(3, 1, 2)

plt.stem(np.arange(16), fft\_mag, 'r')

plt.title('FFT Magnitude')

plt.xlabel('Frequency bin')

plt.ylabel('Magnitude')

plt.grid()

plt.subplot(3, 1, 3)

plt.plot(x\_values, original, 'b-o', label='Original')

plt.plot(x\_values, inverse, 'r--x', label='Reconstructed')

plt.title('Comparison')

plt.xlabel('x')

plt.ylabel('y')

plt.legend()

plt.grid()

plt.tight\_layout()

plt.show()

## **Результаты и анализ**

### График исходной функции

График показывает сумму двух гармонических компонент с частотами 1 и 3. Форма сигнала сложнее простой синусоиды из-за интерференции этих компонент.

### График прямого БПФ

На спектре четко видны:

* Пик на частоте 1 (соответствует cos(x))
* Пик на частоте 3 (соответствует sin(3x))
* Зеркальные пики на частотах N-1 и N-3 (свойство БПФ вещественных сигналов)

Амплитуды основных пиков составляют N/2 = 8, что соответствует теоретическим ожиданиям.

### График обратного БПФ

Восстановленный сигнал полностью совпадает с исходным, за исключением незначительных ошибок округления (порядка 1e-15), что подтверждает корректность реализации.

## **Выводы**

1. Реализованный рекурсивный алгоритм БПФ корректно выполняет прямое и обратное преобразования Фурье.
2. Алгоритм правильно идентифицирует частотные компоненты сложного сигнала.
3. Собственная реализация дает результаты, идентичные MATLAB, что подтверждает ее правильность.
4. Рекурсивный подход обеспечивает ясность реализации, хотя может быть менее эффективен по памяти, чем итеративные версии.
5. Полученные результаты полностью соответствуют теоретическим ожиданиям для алгоритма БПФ.