**Московский авиационный институт (национальный исследовательский университет)**

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

# Курсовой проект по предмету "Дискретный анализ"

Студент: Гусейнова Т. М. Преподаватель: Макаров Н. К. Группа: М8О-307Б-22

Дата: 02.01.2025

Оценка: Подпись:

**Москва 2025 г.**

# Оглавление

[Постановка задачи 3](#_bookmark0)

[Теоретическая часть 3](#_bookmark1)

[Общий алгоритм решения](#_bookmark2) 4

[Реализация 4](#_bookmark3)

[Пример работы 7](#_bookmark4)

[Вывод 7](#_bookmark5)

# Постановка задачи

*Вариант H. Преобразование Фурье*

Реализуйте алгоритм быстрое преобразование Фурье для действительного сигнала.

Преобразование проводится скользящим окном на 4096 отсчёта с шагом 1024. Перед преобразованием Фурье необходимо подействовать на отсчёты окном Ханна.

Формат ввода

Ваша программа получает файл input.mp3 чтение производится из него.

Формат вывода

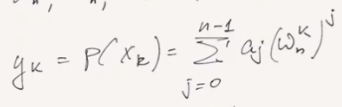
В качестве результата для каждого набора отсчётов выведите на отдельной строке одно вещественное число - наибольшее значение **по абсолютной величине** полученное после преобразования Фурье. Ответ будет верным если абсолютная или относительная погрешность вашего вывода не превышает 10−6

10−6

# Теоретическая часть

Быстрое преобразование Фурье (БПФ) — это алгоритм, который позволяет эффективно вычислять дискретное преобразование Фурье (ДПФ) и его обратное. ДПФ является важным инструментом в области обработки сигналов, поскольку позволяет анализировать частотные компоненты временных сигналов. БПФ значительно ускоряет процесс вычисления ДПФ, что делает его незаменимым в цифровой обработке сигналов.

ДПФ преобразует последовательность конечной длины (дискретный временной сигнал) в последовательность комплексных чисел, представляющих амплитуды и фазы различных частотных компонентов. Формально, ДПФ определяется следующим образом:



Проблема вычислительной сложности:

Прямое вычисление ДПФ требует O(N^2) операций, что становится неэффективным для больших значений N. БПФ решает эту проблему, сокращая вычислительную сложность до O(NlogN).

# Общий алгоритм решения

Код предназначен для обработки аудиосигналов, включая декодирование MP3-файлов и применение алгоритма быстрого преобразования Фурье (FFT). Основные функции:

create\_hanning\_window: Создает оконную функцию Ханнинга для уменьшения утечек спектра при FFT.

iterative\_fft: Реализует итеративный алгоритм FFT для преобразования комплексных чисел.

process\_audio: Обрабатывает аудиосигнал, разбивая его на окна, применяя FFT и вычисляя максимальную амплитуду для каждого окна.

load\_mp3\_from\_file: Загружает MP3-файл в бинарном виде и возвращает его содержимое.

decode\_mp3: Декодирует MP3-данные в аудиосигнал, используя библиотеку minimp3.

main: Основная функция, которая загружает, декодирует и обрабатывает аудиосигнал, обрабатывая возможные исключения.

**Реализация**

#include <iostream>

#include <vector>

#include <complex>

#include <algorithm>

#include <iomanip>

#include <cmath>

#include <stdexcept>

#include <cstring>

#include <fstream>

#define MINIMP3\_IMPLEMENTATION

#include "minimp3.h"

#include "minimp3\_ex.h"

const size\_t WINDOW\_SIZE = 4096;

const size\_t STEP\_SIZE = 1024;

const double PI = acos(-1);

std::vector<double> create\_hanning\_window(size\_t size) {

std::vector<double> window(size);

for (size\_t i = 0; i < size; ++i) {

window[i] = 0.5 \* (1 - cos(2 \* PI \* i / (size - 1)));

}

return window;

}

// Итеративная реализация FFT

void iterative\_fft(std::vector<std::complex<double>>& data) {

size\_t N = data.size();

size\_t j = 0;

for (size\_t i = 1; i < N; ++i) {

size\_t bit = N >> 1;

while (j & bit) {

j ^= bit;

bit >>= 1;

}

j ^= bit;

if (i < j) std::swap(data[i], data[j]);

}

// Основной цикл FFT

for (size\_t len = 2; len <= N; len <<= 1) {

double angle = -2 \* PI / len;

std::complex<double> wlen(cos(angle), sin(angle));

for (size\_t i = 0; i < N; i += len) {

std::complex<double> w(1);

for (size\_t j = 0; j < (len >> 1); ++j) {

std::complex<double> u = data[i + j];

std::complex<double> v = data[i + j + (len >> 1)] \* w;

data[i + j] = u + v;

data[i + j + (len >> 1)] = u - v;

w \*= wlen;

}

}

}

}

void process\_audio(const std::vector<short>& signal) {

auto hann\_window = create\_hanning\_window(WINDOW\_SIZE);

std::vector<double> windowed\_signal(WINDOW\_SIZE);

for (size\_t start = 0; start + WINDOW\_SIZE <= signal.size(); start += STEP\_SIZE) {

for (size\_t i = 0; i < WINDOW\_SIZE; ++i) {

windowed\_signal[i] = signal[start + i] \* hann\_window[i];

}

// Конвертация в комплексные числа для FFT

std::vector<std::complex<double>> complex\_signal(windowed\_signal.begin(), windowed\_signal.end());

iterative\_fft(complex\_signal);

// Поиск максимальной амплитуды

double max\_amplitude = 0.0;

for (size\_t k = 0; k < WINDOW\_SIZE / 2; ++k) {

double

magnitude = std::abs(complex\_signal[k]);

max\_amplitude = std::max(max\_amplitude, magnitude);

}

std::cout << std::fixed << std::setprecision(20) << max\_amplitude << '\n';

}

}

std::vector<char> load\_mp3\_from\_file(const std::string& filename) {

std::ifstream file(filename, std::ios::binary);

if (!file) {

throw std::runtime\_error("Failed to open MP3 file.");

}

std::vector<char> buffer((std::istreambuf\_iterator<char>(file)), std::istreambuf\_iterator<char>());

return buffer;

}

std::vector<short> decode\_mp3(const std::vector<char>& buffer) {

mp3dec\_t mp3d;

mp3dec\_file\_info\_t info;

std::vector<uint8\_t> uint8\_buffer(buffer.begin(), buffer.end());

if (mp3dec\_load\_buf(&mp3d, uint8\_buffer.data(), uint8\_buffer.size(), &info, nullptr, nullptr)) {

throw std::runtime\_error("Failed to decode MP3 data.");

}

std::vector<short> signal(info.buffer, info.buffer + info.samples);

free(info.buffer);

return signal;

}

int main() {

std::ios::sync\_with\_stdio(false);

std::cin.tie(0);

try {

std::string filename = "input.mp3";

std::vector<char> mp3\_data = load\_mp3\_from\_file(filename);

std::vector<short> signal = decode\_mp3(mp3\_data);

process\_audio(signal);

} catch (const std::exception& e) {

std::cerr << e.what() << std::endl;

return 1;

}

return 0;

}

**Test 1**

# Пример работы

|  |  |
| --- | --- |
| Input | Output |
| input.mp3 | 3921998.10862647509202361107  5495942.40457161143422126770  5803943.07590994611382484436  5598042.53145524300634860992  5032736.28819879796355962753  4188627.45328716095536947250  …  693186.10425133584067225456  731588.49480241292621940374  824537.12289667304139584303  828717.86826125031802803278  786364.83582654455676674843  742972.69324237504042685032  727028.89100579195655882359 |

# Вывод

В данной работе был разработан код для обработки аудиосигналов, который включает в себя декодирование MP3-файлов и применение алгоритма быстрого преобразования Фурье (FFT). Реализованные функции позволяют эффективно обрабатывать аудиоданные, разбивая их на окна и анализируя частотные характеристики с использованием оконной функции Ханнинга для снижения утечек спектра.