Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая проект по курсу «Дискретный анализ»

Студент: П.А. Земсков

Преподаватель: Н.К. Макаров

Группа: М8О-301Б

Дата: _____

Оценка: _____

Подпись: _____

Москва, 2025

Содержание

1 Курсовой проект

Задача

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.
Вариант алгоритма: Преобразование Фурье

2 Описание

Алгоритм выполняет преобразование Фурье для действительного сигнала.

2.1 Формат ввода

- Программа получает файл `input.mp3`, из которого производится чтение аудиоданных.

2.2 Формат вывода

- Для каждого окна аудиосигнала выводится одно число — максимальная амплитуда спектра, вычисленная по результатам преобразования Фурье.
- Значения выводятся построчно, в порядке обработки окон.

2.3 Особенности реализации

- Декодирование MP3 в PCM выполняется с использованием библиотеки `minimp3`.
- Используется оконная обработка сигнала — окно Ханна длиной $W = 4096$.
- Обработка выполняется с перекрытием: шаг между окнами $STEP = 1024$.
- Реализовано эффективное быстрое преобразование Фурье (FFT), включающее:
  - битовый реверс индексов,
  - предвычисление корней единицы.
- Анализируется только первая половина спектра $(0 \dots W/2)$ благодаря его симметричности.
- Для каждого окна выводится максимальная спектральная амплитуда с точностью до 20 знаков после запятой.

2.4 Сложность алгоритма

- Декодирование MP3: $O(n)$, где n — количество PCM-сэмплов.
- FFT для одного окна: $O(W \log W)$.
- Общее количество окон: $\approx \frac{n}{STEP}$.
- Итоговая асимптотика:

$$O\left(\frac{n}{STEP} \cdot W \log W\right)$$

При $W = 4096$ и $STEP = 1024$:

$$\approx O(n \log W)$$

3 Исходный код

Общая идея: Программа выполняет локальный спектральный анализ MP3-аудиофайла. Сначала входной MP3-файл декодируется в последовательность PCM-сэмплов с помощью библиотеки minimp3. Далее сигнал анализируется методом, аналогичным STFT: он разбивается на перекрывающиеся окна фиксированной длины (4096 сэмплов, шаг 1024 сэмпла).

Каждое окно предварительно домножается на окно Ханна, что уменьшает искажения на границах сегмента. После этого для окна выполняется быстрое преобразование Фурье (FFT), позволяющее определить распределение энергии по частотам. Из полученного спектра берётся только первая половина и вычисляется максимальная амплитуда.

В результате для каждого окна выводится одно число — максимально сильная частотная составляющая в данном фрагменте аудиосигнала.

Листинг 1: Исходный код

```
#include <bits/stdc++.h>

#define MINIMP3_IMPLEMENTATION
#include "minimp3.h"
#include "minimp3_ex.h"

using namespace std;
constexpr double PI = 3.14159265358979323846264338327950288;


void fft(vector<complex<double>>& a) {
    int n = a.size();
    static vector<int> rev;
    static vector<complex<double>> roots{ {0,0}, {1,0} };

    if ((int)rev.size() != n) {
        int k = __builtin_ctz(n);
        rev.assign(n, 0);
        for (int i = 0; i < n; i++)
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (k - 1));
    }

    if ((int)roots.size() < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            double angle = 2 * PI / (1 << (k + 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots[2 * i] = roots[i];
                double ang = angle * (2 * i + 1 - (1 << k));
                roots[2 * i + 1] = complex<double>(cos(ang), sin(
                    ang));
            }
            k++;
        }
    }

    for (int i = 0; i < n; i++)
        if (i < rev[i]) swap(a[i], a[rev[i]]);

    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                complex<double> z = a[i + j + k] * roots[j + k];
                a[i + j + k] = a[i + j] - z;
                a[i + j] += z;
            }
        }
    }
}
```

```
50
51
52 vector<short> decode_mp3(const char* filename) {
53     mp3dec_t dec;
54     mp3dec_file_info_t info;
55
56     if (mp3dec_load(&dec, filename, &info, NULL, NULL)) {
57         throw runtime_error("mp3␣decode␣error");
58     }
59
60     vector<short> res(info.buffer, info.buffer + info.samples);
61     free(info.buffer);
62     return res;
63 }
64
65
66 int main() {
67     ios::sync_with_stdio(false);
68     cin.tie(nullptr);
69
70     vector<short> pcm = decode_mp3("input.mp3");
71
72     const int W = 4096;
73     const int STEP = 1024;
74
75     static double hann[W];
76     for (int i = 0; i < W; i++) {
77         hann[i] = 0.5 * (1 - cos(2 * PI * i / (W - 1)));
78     }
79
80     int N = pcm.size();
81     for (int start = 0; start + W <= N; start += STEP) {
82         vector<complex<double>> a(W);
83
84         for (int i = 0; i < W; i++) {
85             a[i] = complex<double>(pcm[start + i] * hann[i], 0.0);
86         }
87
88         fft(a);
89
90         double mx = 0;
91         for (int i = 0; i <= W / 2; i++) {
92             mx = max(mx, abs(a[i]));
93         }
94
95         cout << fixed << setprecision(20) << mx << "\n";
96     }
97
98     return 0;
99 }
```

4 Консоль

Пример компиляции и демонстрация работы программы:

```
C:\Users\jocke\Documents\diskran\kp> g++ --std=c++20 main.cpp -o main
C:\Users\jocke\Documents\diskran\kp> ./main
C:\Users\jocke\Documents\diskran\kp> cat output.txt
3921998.10862647509202361107
5495942.40457161143422126770
5803943.07590994611382484436
5598042.53145524300634860992
5032736.28819879796355962753
4188627.45328716095536947250
3592698.56860362365841865540
...
```

5 Тест производительности

Для оценки эффективности работы системы были проведены измерения времени декодирования аудио, вычисления FFT различного размера, а также полного конвейера обработки. Результаты приведены ниже.

5.1 MP3 декодирование

Метрика	Значение
Среднее время декодирования MP3	4.26 ms

Таблица 1: Производительность MP3 декодирования

5.2 Производительность FFT

Размер окна	Время, ms
512	0.005
1024	0.012
2048	0.050
4096	0.092
8192	0.180

Таблица 2: Время вычисления FFT в зависимости от размера окна

5.3 Полный конвейер обработки

Метрика	Значение
Время обработки	40.80 ms
Количество окон	272
Время на окно	0.150 ms

Таблица 3: Производительность полного конвейера (декодирование + обработка)

5.4 Сравнение параметров окон при полном конвейере

Размер окна / шаг	Общее время, ms	Кол-во окон	Время на окно, ms
2048 / 512	13.29	547	0.024
4096 / 1024	22.03	272	0.081
8192 / 2048	22.58	134	0.168

Таблица 4: Сравнение производительности полного конвейера при разных окнах

6 Выводы

Реализованный алгоритм успешно выполняет локальный спектральный анализ аудиосигнала. Использование библиотеки minimp3 обеспечивает быстрое декодирование MP3 в PCM, а применение окна Ханна и эффективной FFT позволяет получать корректные спектральные оценки для каждого сегмента сигнала. Перекрытие окон делает анализ более плавным и чувствительным к кратковременным изменениям.

Проведённые измерения производительности показывают, что даже при размере окна 4096 алгоритм работает достаточно быстро и масштабируется предсказуемо при изменении параметров окна и шага. Итоговая производительность позволяет применять данный подход для обработки аудио в полу-реальном времени или для офлайн-анализа больших файлов.

Таким образом, разработанная программа является эффективным инструментом для оценки спектральной динамики сигнала и может служить основой для более сложных систем анализа аудио.

7 Список литературы

1. *Быстрое преобразование Фурье — Алгоритмика.* `https://ru.algorithmica.org/cs/algebra/fft/`.

2. *ГОСТ Р 7.0.5-2008. Библиографическая ссылка. — 2008.*