Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ»

Студент: А. А. Кабанов
Преподаватель: Н. К. Макаров
Группа: М8О-301Б-22

Дата:
Оценка:
Подпись:

Москва, 2025

Курсовой проект

Вариант: Преобразование Фурье.

Задача: Реализовать алгоритм быстрого преобразования 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: Программа получает файл input.mp3, чтение производится из него.

Формат вывода: В качестве результата для каждого набора отсчётов выводится 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}.

1 Описание

Требуется реализовать преобразование Фурье, применимое к действительному сигналу, на C++.

Аудиосигнал декодируется из файла, после чего к нему применяется скользящее окно с последующим преобразованием Фурье. Это позволяет получить частотный спектр сигнала в каждый момент времени.

Аудиофайл в формате .mp3 декодируется в массив отсчётов (сэмплов) с помощью библиотеки minimp3.

Полученные отсчёты представляют собой временной ряд, который описывает амплитуду звукового сигнала в каждый момент времени.

Перед применением FFT используется оконная функция Ханна (Hann window). Это необходимо для уменьшения эффектов, связанных с разрывами на границах окна (спектральные утечки).

Окно — это функция, которая применяется к отрезку сигнала (обычно фиксированного размера) перед выполнением каких-либо операций. Оно "вырезает" часть сигнала, и его значение постепенно уменьшается к краям этого отрезка.

Окно Ханна задается формулой: $\omega(n) = \frac{1}{2}(1 - cos(\frac{2\pi n}{N-1}))$, где N - ширина окна, n - индекс отсчета в окне. Сложность - $O(N)$.

Далее приведена реализация итеративного быстрого преобразования Фурье, состоящего из бит-реверсной перестановки и алгоритма Кули-Тьюки (с "бабочками").

Отличие рекурсивного и итеративного алгоритма заключается в том, что можно расположить элементы вектора таким же образом, как они лежат в листьях, когда происходят вызовы рекурсивного алгоритма (то есть применить восходящий подход).

Далее реализован основной цикл БПФ - вычисления спектра с использованием "бабочек операций умножения на поворотный множитель и сложения и вычитания результатов, что ускоряет работу преобразования Фурье с $O(n^2)$ до $O(nlogn)$.

```
1  #include <bits/stdc++.h>
2  #include <cmath>
3  #include <iomanip>
4
5  #define MINIMP3_IMPLEMENTATION
6
7  #include "minimp3.h"
8  #include "minimp3_ex.h"
9
10 std::vector<short> decoder() {
11   mp3dec_t mp3d;
12   mp3dec_file_info_t info;
13   if (mp3dec_load(&mp3d, "input.mp3", &info, NULL, NULL)) {
14     throw std::runtime_error("Decode error");
```

```
  }
  std::vector<short> res(info.buffer, info.buffer + info.samples);
  free(info.buffer);
  return res;
}

std::vector<double> HannWindow(size_t sz) {
  std::vector<double> res(sz);
  for (size_t i = 0; i < sz; ++i) {
    res[i] = 0.5 * (1 - cos((2 * M_PI * static_cast<int>(i)) / (sz - 1)));
  }
  return res;
}

void FFT(std::vector<std::complex<double>> &A) {
  size_t n = A.size();
  size_t k = std::log2(n);
  for (size_t i = 0, j = 0; i < n; ++i) {
    if (i < j) {
      std::swap(A[i], A[j]);
    }
    int bit = n >> 1;
    while (j & bit) {
      j ^= bit;
      bit >>= 1;
    }
    j ^= bit;
  }
  for (uint64_t s = 1; s <= k; ++s) {
    uint64_t m = 1 << s; // 2^s
    std::complex<double> w_m = std::polar(1.0, -2 * M_PI / m);
    for (uint64_t k = 0; k < n; k += m) {
      std::complex<double> w = 1;
      for (uint64_t j = 0; j < m / 2; ++j) {
        std::complex<double> t = w * A[k + j + m / 2];
        std::complex<double> u = A[k + j];
        A[k + j] = u + t;
        A[k + j + m / 2] = u - t;
        w = w * w_m;
      }
    }
  }
}

int main() {
  std::ios_base::sync_with_stdio(false);
  std::cin.tie(nullptr);

  std::vector<short> data = decoder();
```

```
64      std::size_t win_size = 4096;
65      std::size_t step = 1024;
66      std::size_t data_size = data.size();
67      std::vector<double> hw = HannWindow(win_size);
68
69      for (uint64_t i = 0; i < data_size - win_size; i += step) {
70        std::vector<std::complex<double>> a(win_size);
71        for (std::size_t j = 0; j < win_size; ++j) {
72          a[j] = data[i + j] * hw[j];
73        }
74        FFT(a);
75        double m = 0.0;
76        for (auto& num : a) {
77          m = std::max(m, std::abs(num));
78        }
79        std::cout << std::fixed << std::setprecision(20) << m << "\n";
80      }
81    }
```

2 Консоль

```
[anton@home course-project-FFT]$ make
g++ -O2 -lm -fno-stack-limit -std=c++20 -x c++ solution.cpp -o executable
[anton@home course-project-FFT]$ make run
./executable
3921998.10862647509202361107
...
693186.10425133584067225456
731588.49480241292621940374
824537.12289667304139584303
828717.86826125031802803278
786364.83582654455676674843
742972.69324237504042685032
727028.89100579195655882359
```

3 Тест производительности

Тест производительности представляет из себя следующее: запуск файлов формата mp3 (V0 и 320 кбит/с) с отключенным выводом полученных значений в консоль/-файл и использованием std::chrono для замера времени выполнения FFT.

```
[anton@home course-project-FFT]$ make
g++ -O2 -lm -fno-stack-limit -std=c++20 -x c++ solution.cpp -o executable
[anton@home course-project-FFT]$ time make run
./executable
[FFT_BENCH] Done in 15935 microseconds.
[anton@home course-project-FFT]$ make run
./executable
[FFT_BENCH] Done in 1299263 microseconds.
[anton@home course-project-FFT]$ make run
./executable
[FFT_BENCH] Done in 8175367 microseconds.
[anton@home course-project-FFT]$ make run
./executable
[FFT_BENCH] Done in 1306724 microseconds.
[anton@home course-project-FFT]$ make run
./executable
[FFT_BENCH] Done in 8345970 microseconds.
```

Файл	**Размер (МБ)**	**Время выполнения (с)**
Тестовый файл	0.05	0.02
Файл 1 (V0)	8.4	1.3
Файл 2 (V0)	47.3	8.1
Файл 1 (320)	10.8	1.3
Файл 2 (320)	70.4	8.3

4 Выводы

Я выполнил задание курсового проекта по дискретному анализу.

В ходе выполнения работы был реализован алгоритм быстрого преобразования Фурье (FFT) для анализа аудиосигнала, декодированного из файла формата .mp3. Использовалось скользящее окно размером 4096 отсчётов с шагом 1024, а также оконная функция Ханна для уменьшения спектральных утечек. Бит-реверсивная перестановка и операция "бабочки"обеспечили эффективное вычисление спектра сигнала за время $O(nlogn)$.

Данный подход может быть использован для решения задач анализа звука, таких как распознавание речи, обработка музыки или выделение характерных частотных признаков.

Он отличается, по моему мнению, достаточно сложной для понимания математической базы, но это целиком и полностью оправдано, так как он ускоряет анализ действительных сигналов.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 3-е издание.* — Издательский дом «Вильямс», 2013. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 978-5-8459-1794-2 (рус.))

[2] *MAXimal :: algo :: Быстрое преобразование Фурье за O (N log N). Применение к умножению двух полиномов или длинных чисел*
URL: `http://e-maxx.ru/algo/fft_multiply`