Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Численные методы»

Тема: «Аудиопоиск»

Студент: К. М. Воронов
Преподаватель: Д. Л. Ревизников
Группа: М8О-307Б-19

Дата:
Оценка:
Подпись:

Москва, 2022

Постановка задачи

Задача: Реализуйте систему для поиска аудиозаписи по небольшому отрывку. Все файлы будут даны в формате WAV с частотой дискретизации 44100Гц. Входные файлы содержат в себе имена файлов с аудио записями по одному файлу в строке. Результатом ответа на запрос является строка с названием файла, с которым произошло совпадение, либо строка «Не найдено!», если найти совпадение не удалось.

1 Описание

Дискретное преобразование Фурье (ДПФ)

Пусть $A(x)$ — многочлен $n-1$ степени:

$$A(x) = a_0 \cdot + a_1 \cdot x + a_2 \cdot x^2 + a_3 \cdot x^3 + \cdots + a_{n-2} \cdot a^{n-2} + a_{n-1} \cdot a^{n-1}$$

Будем считать, что n является степенью двойки. Если это не так, то мы можем дополнить исходный многочлен недостающими членами с нулевыми коэффициентами.

Обозначим $\omega_{n,k}$ — k-й комплексный корень степени n из единицы. Так как $1 = e^{2\pi i}$, то решая уравнение $z^n = 1 = e^{2\pi i}$ получим $z_k = e^{k \cdot \frac{2\pi i}{n}} = \omega_{n,k}$, где $k \in \{0, 1, 2, \dots n-1\}$. Обозначим $\omega_n = \omega_{n,1}$. По свойству степени $\omega_{n,k} = \omega_n^k$.

Назовём дискретным преобразованием Фурье многочлена $A(x)$ вектор из n чисел:

$$DFT(a_0, a_1, a_2, \dots a_{n-1}) = (A(\omega_n^0), A(\omega_n^1), A(\omega_n^2), \dots A(\omega_n^{n-1}))^T = (y_0, y_1, y_2, \dots y_{n-1})^T$$

Заметим, что для ω_n^k значение многочлена равно:

$$\begin{aligned} A(\omega_n^k) &= a_0 \cdot (\omega_n^k)^0 + a_1 \cdot (\omega_n^k)^1 + a_2 \cdot (\omega_n^k)^2 + a_3 \cdot (\omega_n^k)^3 + \cdots + a_{n-1} \cdot (\omega_n^k)^{n-1} = \\ &= a_0 \cdot \omega_n^0 + a_1 \cdot \omega_n^k + a_2 \cdot \omega_n^{2 \cdot k} + a_3 \cdot \omega_n^{3 \cdot k} + \cdots + a_{n-1} \cdot \omega_n^{(n-1) \cdot k} \end{aligned}$$

Тогда преобразование можно записать в матричном виде:

$$DFT(a_0, a_1, a_2, \dots a_{n-1}) = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \dots \\ y_{n-1} \end{pmatrix} = W \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \dots \\ a_{n-1} \end{pmatrix}$$

Здесь W — матрица Вандермонда:

$$W = \begin{pmatrix} \omega_n^0 & \omega_n^0 & \omega_n^0 & \omega_n^0 & \dots & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \omega_n^2 & \omega_n^3 & \dots & \omega_n^{(n-1)} \\ \omega_n^0 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \dots & \omega_n^{2 \cdot (n-1)} \\ \omega_n^0 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \dots & \omega_n^{3 \cdot (n-1)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \omega_n^0 & \omega_n^{(n-1)} & \omega_n^{2 \cdot (n-1)} & \omega_n^{3 \cdot (n-1)} & \dots & \omega_n^{(n-1)^2} \end{pmatrix}$$

Умножим обе части слева на W^{-1}, получим:

$$W^{-1} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \dots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \dots \\ a_{n-1} \end{pmatrix}$$

W^{-1} имеет следующий вид:

$$W^{-1} = \frac{1}{n} \cdot \begin{pmatrix} \omega_n^0 & \omega_n^0 & \omega_n^0 & \omega_n^0 & \dots & \omega_n^0 \\ \omega_n^0 & \omega_n^{-1} & \omega_n^{-2} & \omega_n^{-3} & \dots & \omega_n^{-(n-1)} \\ \omega_n^0 & \omega_n^{-2} & \omega_n^{-4} & \omega_n^{-6} & \dots & \omega_n^{-2\cdot(n-1)} \\ \omega_n^0 & \omega_n^{-3} & \omega_n^{-6} & \omega_n^{-9} & \dots & \omega_n^{-3\cdot(n-1)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \omega_n^0 & \omega_n^{-(n-1)} & \omega_n^{-2\cdot(n-1)} & \omega_n^{-3\cdot(n-1)} & \dots & \omega_n^{-(n-1)^2} \end{pmatrix}$$

Определим обратное преобразование Фурье для вектора из n чисел:

$$InverseDFT(y_0, y_1, y_2, \dots y_{n-1}) = (a_0, a_1, a_2, \dots a_{n-1})^T$$

Или в матричном виде:

$$InverseDFT(y_0, y_1, y_2, \dots y_{n-1}) = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \dots \\ a_{n-1} \end{pmatrix} = W^{-1} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \dots \\ y_{n-1} \end{pmatrix}$$

Очевидно, что $InverseDFT(DFT(A)) = A$.

Вычисление $DFT(A)$ простым перемножение матрицы на вектор требует $O(n^2)$ операций, однако можно добиться существенного ускорения.

Быстрое преобразование Фурье (БПФ)

Разобьём исходный многочлен $A(x)$ на два многочлена $A_0(x)$ и $A_1(x)$, содержащие чётные и нечётные коэффициенты соответственно:

$$A_0(x) = a_0 + a_2 \cdot x + a_4 \cdot x^2 + \cdots + a_{n-2} \cdot x^{\frac{n}{2}-1}$$

$$A_1(x) = a_1 + a_3 \cdot x + a_5 \cdot x^2 + \cdots + a_{n-1} \cdot x^{\frac{n}{2}-1}$$

Тогда мы можем вычислить значение исходного многочлена следующим образом:

$$A(x) = A_0(x^2) + x \cdot A_1(x^2)$$

Вычислим дискретное преобразование Фурье для многочленов $A_0(x)$ и $A_1(x)$:

$$DFT(A_0) = (y_0^0, y_1^0, y_2^0, \dots y_{\frac{n}{2}-1}^0)^T$$

$$DFT(A_1) = (y_0^1, y_1^1, y_2^1, \dots y_{\frac{n}{2}-1}^1)^T$$

Здесь y^0 и y^1 — значения ДПФ для $A_0(x)$ и $A_1(x)$, а не показатели степени при y.
Теперь вычислим $DFT(A)$, используя $DFT(A_0)$ и $DFT(A_1)$, используя следующие три важных свойства:

$$\omega_{n,k}^2 = (e^{k \cdot \frac{2\pi i}{n}})^2 = e^{2 \cdot k \cdot \frac{2\pi i}{n}} = e^{k \cdot \frac{2\pi i}{\frac{n}{2}}} = \omega_{\frac{n}{2},k}$$

$$\omega_{n,k+\frac{n}{2}} = e^{(k+\frac{n}{2}) \cdot \frac{2\pi i}{n}} = e^{k \cdot \frac{2\pi i}{n}} \cdot e^{\frac{n}{2} \cdot \frac{2\pi i}{n}} = e^{k \cdot \frac{2\pi i}{n}} \cdot e^{\pi i} = e^{k \cdot \frac{2\pi i}{n}} \cdot (-1) = -e^{k \cdot \frac{2\pi i}{n}} = -\omega_{n,k}$$

$$\omega_{n,k+\frac{n}{2}}^2 = (-\omega_{n,k})^2 = \omega_{n,k}^2 = \omega_{\frac{n}{2},k}$$

Пусть $k \in \{0, 1, 2, \dots \frac{n}{2} - 1\}$, тогда, используя первое свойство, получим:

$$y_k = A(\omega_{n,k}) = A_0(\omega_{n,k}^2) + \omega_{n,k} \cdot A_1(\omega_{n,k}^2) = A_0(\omega_{\frac{n}{2},k}) + \omega_{n,k} \cdot A_1(\omega_{\frac{n}{2},k})$$

Степени многочленов $A_0(x)$ и $A_1(x)$ равны $\frac{n}{2}-1$, значит $y_k^0 = A_0(\omega_{\frac{n}{2},k})$ и $y_k^1 = A_1(\omega_{\frac{n}{2},k})$.
Подставим в выражение выше, получим:

$$y_k = y_k^0 + \omega_{n,k} \cdot y_k^1$$

Таким образом, мы вычислили половину значений $DFT(A)$, используя значения $DFT(A_0)$ и $DFT(A_1)$.
Используем второе и третье свойства:

$$y_{k+\frac{n}{2}} = A(\omega_{n,k+\frac{n}{2}}) = A_0(\omega_{n,k+\frac{n}{2}}^2) + \omega_{n,k+\frac{n}{2}} \cdot A_1(\omega_{n,k+\frac{n}{2}}^2) = A_0(\omega_{\frac{n}{2},k}) - \omega_{n,k} \cdot A_1(\omega_{\frac{n}{2},k})$$

Подставим y_k^0 и y_k^1 в выражение:

$$y_{k+\frac{n}{2}} = y_k^0 - \omega_{n,k} \cdot y_k^1$$

Итак, мы получили $DFT(A)$ с помощью значений $DFT(A_0)$ и $DFT(A_1)$. То есть мы разбили исходную задачу на две поздачи вдвое меньшего размера и, решив их, получили решение исходной задачи. Такая схема называется «разделяй-и-властвуй» и имеет известную вычислительную сложность $O(n \cdot \log(n))$.

Всё вышеописанное называется быстрым преобразование Фурье.

Ускорение алгоритма

Заметим, что в приведённой реализации используется $O(n \cdot \log(n))$ памяти, из-за чего константа в асимптотике алгоритма довольно велика. Можно отказаться от рекурсивного выделения памяти, если переупорядочить элементы в исходном векторе.

Рассмотрим самый нижний уровень рекурсии и то, какие коэффициенты исходного многочлена будут использоваться при вычислении.

Изначальный вектор (окружен фигурными скобками)

$$\{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$$

распадается на два вызова (окружены квадратными скобками)

$$\{[a_0, a_2, a_4, a_6], [a_1, a_3, a_5, a_7]\}$$

На следующем уровне рекурсии

$$\{[a_0, a_2, a_4, a_6], [a_1, a_3, a_5, a_7]\}$$

распадается на четыре вызова (окружены круглыми скобками)

$$\{[(a_0, a_4), (a_2, a_6)], [(a_1, a_5), (a_3, a_7)]\}$$

Полученный порядок называется поразрядно обратной перестановкой. Если мы посмотрим на какой позиции стоит тот или иной элемент, окажется, что если мы выполним реверс бит в двоичной записи изначальных позиций, то получим позиции на последнем уровне рекурсии. Например, $6_{10} = 110_2$, выполним реверс бит, получим $011_2 = 3_{10}$, что соответствует индексы элемента a_6 на последнем уровне рекурсии.

Упорядочим элементы вектора a как на последнем слою. Вычислим значения ДПФ для пар, окруженных круглыми скобками и запишем их вместо самих элементов, получим:

$$\{[(y_0^0, y_1^0), (y_0^1, y_0^1)], [(a_1, a_5), (a_3, a_7)]\}$$

Теперь запишем новые значение ДПФ на позиции самих элементов:

$$\{[(y_0^0 + \omega_{4,0} \cdot y_0^1, y_1^0 + \omega_{4,1} \cdot y_1^1), (y_0^0 - \omega_{4,0} \cdot y_0^1, y_1^0 - \omega_{4,1} \cdot y_1^1)], [(a_1, a_5), (a_3, a_7)]\}$$

Проделаем ту же операцию справа, получим:

$$\{[y_0^0, y_1^0, y_2^0, y_3^0], [y_0^1, y_1^1, y_2^1, y_3^1]\}$$

Видно, что элементы снова стоят подходящим образом, можно записывать значения ДПФ на позиции самих элементов:

$$\{[y_0^0 + \omega_{8,0} \cdot y_0^1, y_1^0, y_2^0, y_3^0], [y_0^0 - \omega_{8,0} \cdot y_0^1, y_1^1, y_2^1, y_3^1]\}$$

После всех вычислений получим:

$$\{y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7\}$$

Таким образом мы смогли вычислить значения ДПФ без использования дополнительной памяти. Такой подход уменьшает константу в асимптотике. Временная сложность по-прежнему $O(n \cdot \log(n))$, а вот пространственная $O(1)$.

Листинг БПФ

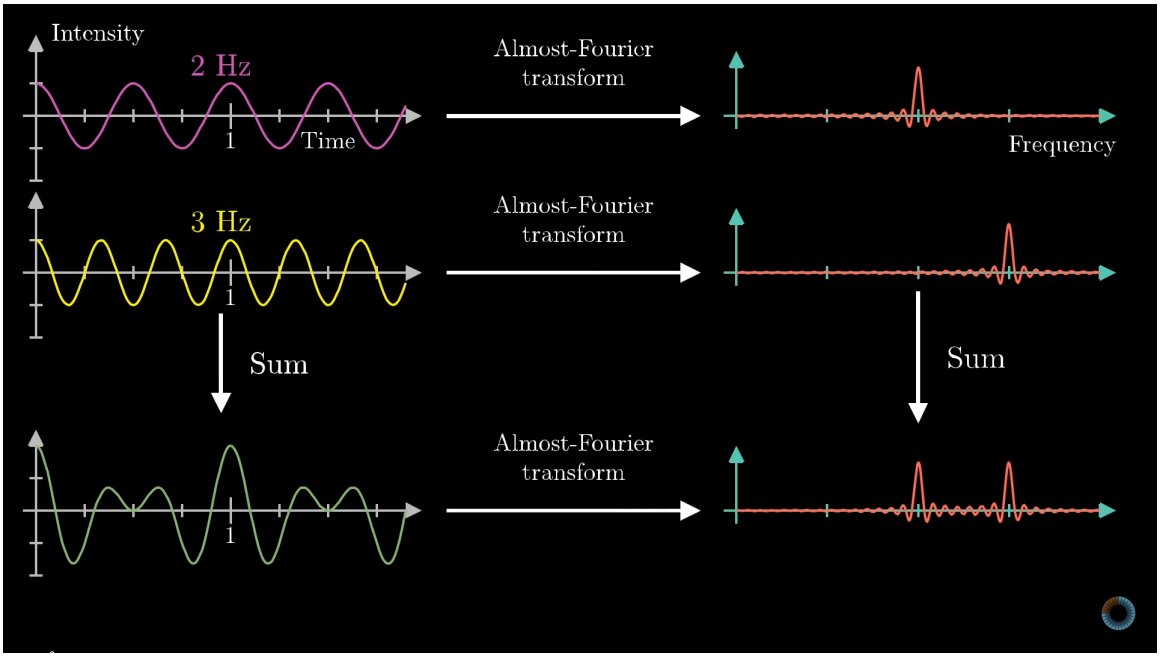
Ниже приведена реализация БПФ.

```
int Reverse(int value, int lg_n) {
    int res = 0;
    while(lg_n > 0) {
        res = res << 1;
        res = res | (value & 1);
        value = value >> 1;
        lg_n -= 1;
    }
    return res;
}

void FFT(vector<complex<double>> & a, const int &n) {
    int lg_n = 0;
    while ((1 << lg_n) < n) {
        ++lg_n;
    }

    for (int i = 0; i < n; ++i) {
        int fr = Reverse(i, lg_n);
        if (i < fr) {
            swap (a[i], a[fr]);
        }
    }

    for (int k = 1; k <= lg_n; ++k) {
        int len = 1 << k;
        for (int i = 0; i < n; i += len) {

            complex<double> wz(cos(2 * PI / len), sin(2 * PI / len));
            complex<double> w(1, 0);

            for (int j = 0; j < len / 2; j++) {
                complex<double> u = a[i + j];
                complex<double> v = a[i + j + len / 2] * w;
                a[i + j] = u + v;
                a[i + j + len / 2] = u - v;
                w *= wz;
```

```
38                }
39            }
40        }
41    }
```

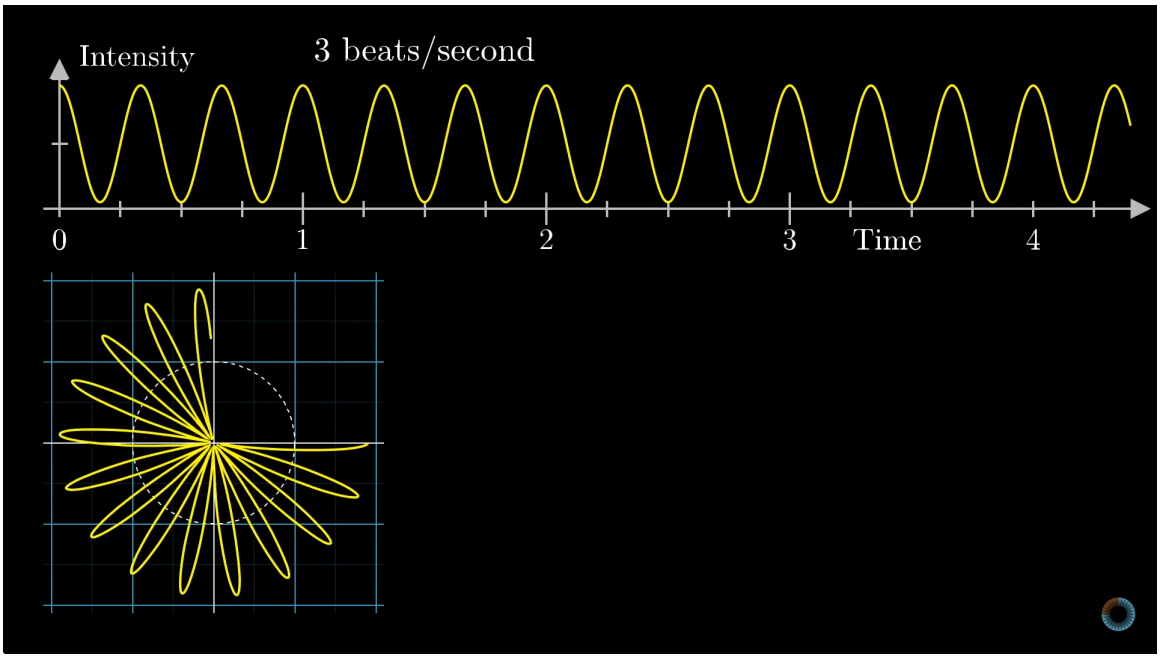
Преобразование Фурье для звукового сигнала

Преобразование Фурье позволяет разложить исходный сигнал на гармонические составляющие. Фактически, можно получить амплитудный спектр сигнала, то есть значения частот, из которых он состоит. После применения преобразования Фурье над «замерами» сигнала получается массив комплексных чисел, модуль которых будет являться значением амплитуды. Для корректного отображения нужно идти с шагом $\frac{D}{size}$, где D - частота дискретизации, а $size$ - количество «замеров» сигнала.

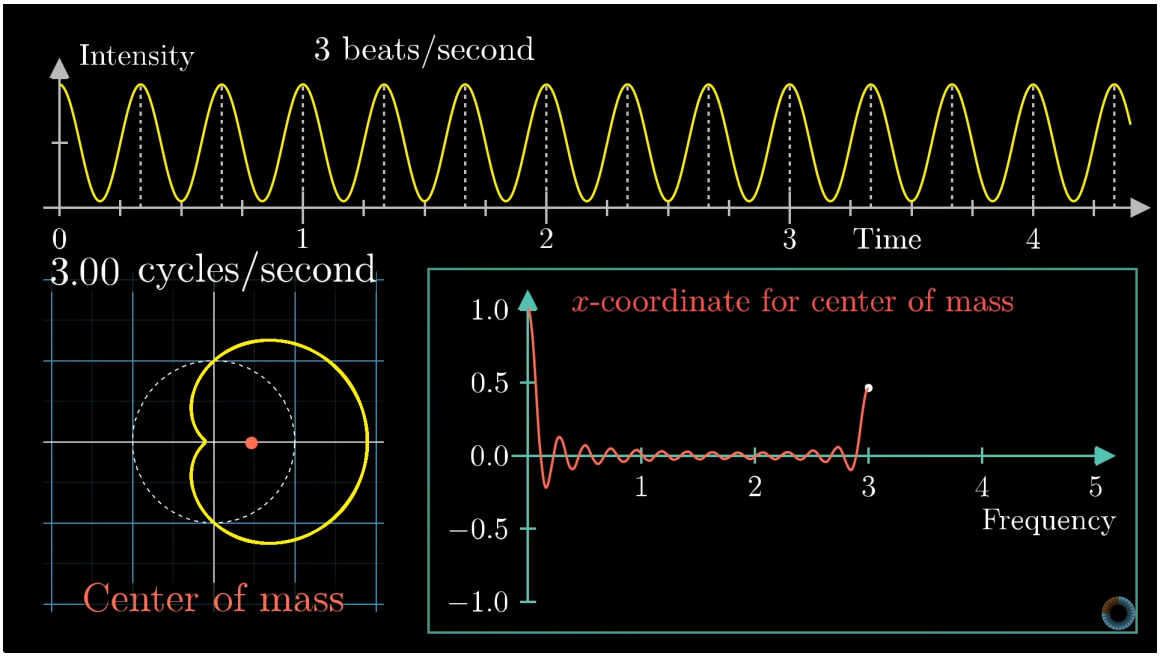


Геометрическая интерпретация

Отобразим исходный график на комплексную плоскость. Для этого введём радиус-вектор, который равномерно вращается по часовой стрелке. Его длина в каждый момент времени равна модулю значения сигнала, а частота вращения выбирается произвольным образом.



Фактически, мы получаем две частоты, одна частота - частота сигнала, а вторая - частота вращения. Вторую частоту, то есть частоту вращения, мы можем менять путем наматывания графика. Соответственно её выбор будет влиять на то, как выглядит нижняя диаграмма. Когда эти частоты совпадут, получится интересная картина. Если мы введём некоторый центр масс, то при совпадении частот он сместится вправо от центра координат, когда как в остальных случаях он будет у нуля. Если постить график зависимочти координанты x этого центра масс от времени, мы как раз получим резкий скачок на нужной нам частоте.



Таким образом, с помощью частот, входящих в какую-либо песню, казалось бы, мы можем с легкостью найти ее отрывок. Но на практике все намного сложнее.

Оконное преобразование Фурье

На практике музыкальные произведения не однородны, то есть их частотный состав меняется каждую секунду. Какие-то звуки появляются, какие-то пропадают, какие-то длятся долго, а какие-то доли секунды. Поэтому имеет смысл анализировать не все произведение сразу, а последовательно, по кусочкам. Для этого применяют оконное преобразование Фурье. Если применять преобразование Фурье каждые n отсчётов, мы получим самое простое окно - прямоугольное. Однако существуют и другие, которые помогают убавить количество шумов. Реализация некоторых окон.

```
void WindwoHann(vector<complex<double>> &signal, int size) {
    for (int i = 0; i < size; i++) {
        double multiplier = 0.5 * (1 - cos(2* PI * i / (size- 1)));
        signal[i] = multiplier * signal[i];
    }
}

void Gausse(vector<complex<double>> &signal, int size) {
    for (int i = 0; i < size; i++) {
        int a = (size - 1) / 2;
        double t = (i - a) / (0.5 * a);
        t = t*t;
        signal[i] *= exp(-t/2);
    }
}
```

Выглядят они следующим образом.

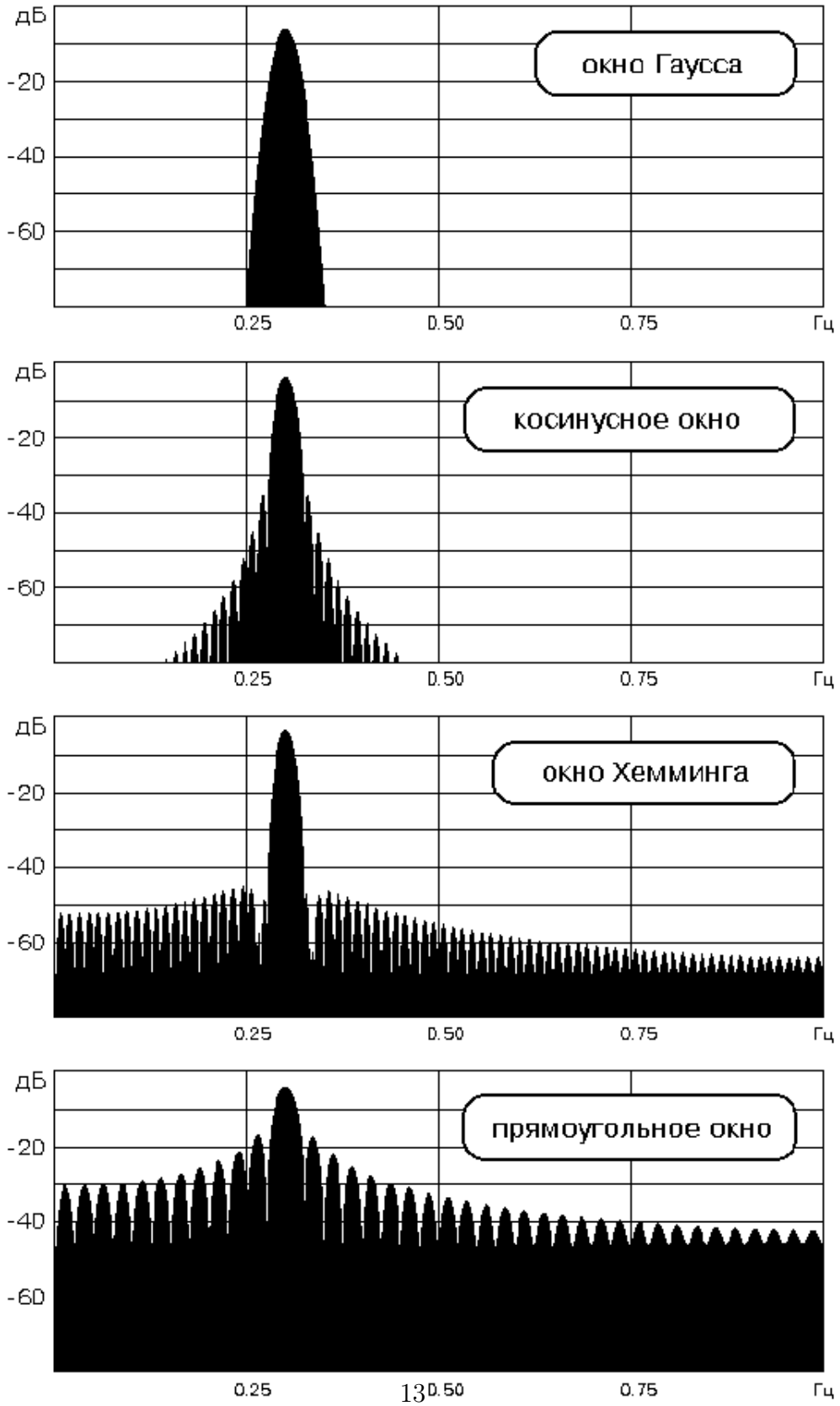


Рис. 12.7. Артефакты спектра одиночного гармонического сигнала (12.7) при использовании разных типов окно.

Алгоритм аудиопоиска

Чтобы находить нужную мне музыкальную композицию, я получал частотные спектры анализируемого файла и примера. После этого находил максимумы на определенных интервалах частот, а именно от 0 до 40, от 40 до 80, от 80 до 120, от 120 до 180 и от 180 до 300, так как обычно именно в этих диапазонах находятся важные элементы музыкальных композиций. Так как результаты получались с погрешностями(например, могло получится $2 * 10^6 + 5.4$ и $2 * 10^6 + 6.2$. Очевидно, скачок и там, и там), я брал от этих максимумов логарифм по основанию 2. Далее непосредственно сравнивал каждое окно с каждым и выводил тот результат, где было больше всего совпадений. Если совпадало меньше половины, то считал, что эта композиция не подходит. Однако минус моего алгоритма в том, что из-за большой погрешности в виде взятия логарифма иногда находятся песни, которых нет в базе данных. Но это некритично, так как лучше найти произведение, которого нет, но оно похоже, чем не найти, которое есть.

2 Исходный код

Реализация основных функций.

Применение преобразования Фурье к аудиофайлу, на выходе массив с частотами для каждого окна.

```
vector<vector<complex<double>>> fileWindiowFFT(FILE * file, FILE *gr, FILE *gr_f, bool
    write_graphics) {

    TWaveHeader header;
    fread(&header, sizeof(TWaveHeader), 1, file);

    Chunk chunk;
    while (true) {
        fread(&chunk, sizeof(chunk), 1, file);
        if (*(unsigned int*)&chunk.ID == 0x61746164) { //data
            break;
        }
        fseek(file, chunk.size, SEEK_CUR);
    }

    short point;
    vector<complex<double>> signal;
    vector<double> graphic;

    int k = 0;
    double mean = 0;
    int window = 0;

    vector<vector<complex<double>>> analysis;

    while (chunk.size > 0) {
        k += 1;
        fread(&point, sizeof(uint16_t), 1, file);
        double p = double(point);

        chunk.size -= sizeof(uint16_t);

        if (header.NChannels == 2) {
            if (k % 2 == 1) {
                mean = p;
                continue;
            }
            p = (p + mean) / 2;
        }

        window += 1;

        signal.push_back(p);
```

```
43
44          if (window == WINSIZE) {
45              window = 0;
46              Gausse(signal, WINSIZE);
47              analysis.push_back(signal);
48              signal.clear();
49          }
50
51          if (write_graphics) {
52              graphic.push_back(p);
53              string name = to_string(graphic[graphic.size() - 1]) + "\n";
54              fprintf(gr, name.c_str());
55          }
56      }
57
58      while(analysis[analysis.size() - 1].size() < (uint)WINSIZE) {
59          analysis[analysis.size() - 1].push_back(complex<double>(0));
60      }
61
62      for (uint i = 0; i < analysis.size(); ++i) {
63          FFT(analysis[i], analysis[i].size());
64      }
65
66      if(write_graphics) {
67          writeGraphicsFFT(gr_f, analysis);
68      }
69
70      return analysis;
71  }
```

Подсчёт максимумов, взятие логарифма.

```
1   int getIndex(double curfreq) {
2
3       if (curfreq < 30 || curfreq > RANGE_FREQ[RANGE_FREQ.size() - 1]) {
4           return -1;
5       }
6
7       for (uint i = 0; i < RANGE_FREQ.size(); ++i) {
8           if (curfreq < RANGE_FREQ[i]) {
9               return (int)i;
10          }
11      }
12
13      return -1;
14  }
15
16  vector<vector<double>> analysisOfFile(vector<vector<complex<double>>> analysis, int
        size) {
17
```

```
    vector<vector<double>> freqs(analysis.size(), vector<double>(RANGE_FREQ.size()));
    for (uint i = 0; i < analysis.size(); ++i) {
        double curfreq = 0;
        vector<double> max(RANGE_FREQ.size());
        for (uint j = 0; j < analysis[i].size() / 2; ++j) {

            curfreq += 44100 / size;
            int ind = getIndex(curfreq);
            if (ind == -1) {
                continue;
            }

            if (floor(log2(abs(analysis[i][j])) > max[ind])) {
                freqs[i][ind] = floor(log2(abs(analysis[i][j])));
            }
        }
    }

    return freqs;
}
```

Сравнение

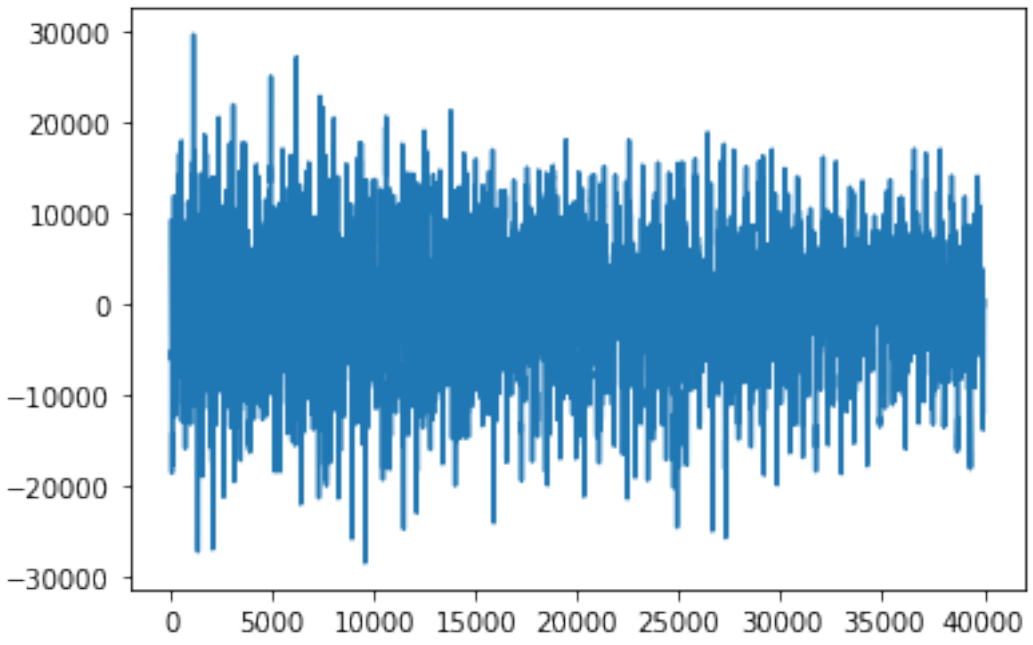
```
int Search(vector<vector<double>> &freqsSearch, vector<vector<double>> &freqsVariant)
    {
    int ans = 0;

    for (uint i = 0; i < freqsVariant.size(); ++i) {
        for (uint k = 0; k < freqsSearch.size(); ++k) {

            bool flag = true;

            for (uint j = 0; j < freqsVariant[i].size(); ++j) {

                if(freqsVariant[i][j] != freqsSearch[k][j]) {
                    flag = false;
                    break;
                }
            }

            if (flag) {
                ans += 1;
            }
        }
    }

    return ans;
}
```

Также я добавил визуализацию графиков в python, путем записи и чтения из файлов.

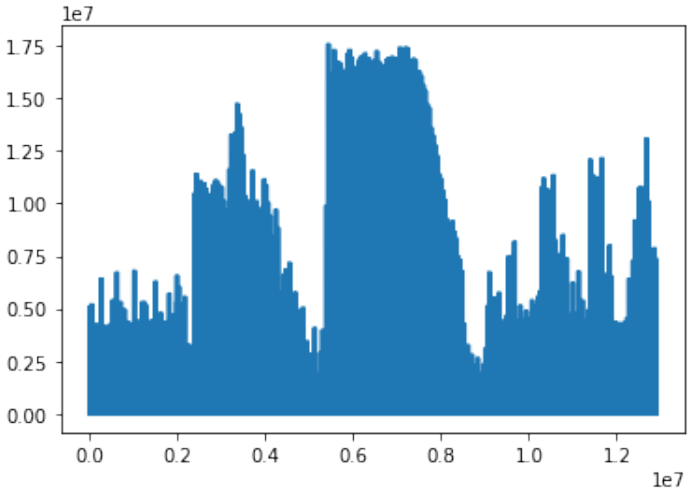
Функция для записи

```
void writeGraphicsFFT(FILE *gr_f, vector<vector<complex<double>>> &analysis) {
    vector<double> magnitude;
    vector<double> phase;

    for (uint i = 0; i < analysis.size(); ++i) {
        for (uint j = 0; j < analysis[i].size(); ++j) {
            magnitude.push_back(sqrt(analysis[i][j].real() * analysis[i][j].real() +
                analysis[i][j].imag() * analysis[i][j].imag()));
            phase.push_back(atan2(analysis[i][j].imag(), analysis[i][j].real()));
        }
    }

    for (uint i = 0; i < magnitude.size(); ++i) {

        string name = to_string(magnitude[i]) + "\n";
        fprintf(gr_f, name.c_str());

        name = to_string(phase[i]) + "\n";
        fprintf(gr_f, name.c_str());
    }
}
```

Визуализация одного из сигналов



Его частотный спектр(все окна подряд)



3 Консоль

На вход программе подается название отрывка и файл с названиями песен, среди которых ищем.

Содержание names.txt

```
Blizzard.wav
Decade_Dance.wav
Silver_Lights.wav
Voyager.wav
Dust.wav
Simma_Hem.wav
Divide.wav
Hollywood_Heights.wav
Interlude.wav
Around.wav
Java.wav
Rust.wav
Roller_Mobster.wav
Run.wav
```

Примеры работы программы

```
kirill@kirill-G3-3779:~/FFT$ ./s* pat1_Silver_Lights.wav names.txt
Анализ файла: Blizzard.wav
Количество совпадений: 27
Анализ файла: Decade_Dance.wav
Количество совпадений: 4
Анализ файла: Silver_Lights.wav
Количество совпадений: 620
Анализ файла: Voyager.wav
Количество совпадений: 36
Анализ файла: Dust.wav
Количество совпадений: 2
Анализ файла: Simma_Hem.wav
Количество совпадений: 33
Анализ файла: Divide.wav
Количество совпадений: 14
Анализ файла: Hollywood_Heights.wav
Количество совпадений: 38
Анализ файла: Interlude.wav
Количество совпадений: 40
Анализ файла: Around.wav
Количество совпадений: 18
Анализ файла: Java.wav
```

```
Количество совпадений: 0
Анализ файла: Rust.wav
Количество совпадений: 23
Анализ файла: Roller_Mobster.wav
Количество совпадений: 4
Анализ файла: Run.wav
Количество совпадений: 79
Silver_Lights.wav

kirill@kirill-G3-3779:~/FFT$ ./s* pat2_Decade_Dance.wav names.txt
Анализ файла: Blizzard.wav
Количество совпадений: 1
Анализ файла: Decade_Dance.wav
Количество совпадений: 920
Анализ файла: Silver_Lights.wav
Количество совпадений: 32
Анализ файла: Voyager.wav
Количество совпадений: 99
Анализ файла: Dust.wav
Количество совпадений: 79
Анализ файла: Simma_Hem.wav
Количество совпадений: 17
Анализ файла: Divide.wav
Количество совпадений: 45
Анализ файла: Hollywood_Heights.wav
Количество совпадений: 6
Анализ файла: Interlude.wav
Количество совпадений: 15
Анализ файла: Around.wav
Количество совпадений: 9
Анализ файла: Java.wav
Количество совпадений: 2
Анализ файла: Rust.wav
Количество совпадений: 8
Анализ файла: Roller_Mobster.wav
Количество совпадений: 108
Анализ файла: Run.wav
Количество совпадений: 21
Decade_Dance.wav

kirill@kirill-G3-3779:~/FFT$ ./s* pat6_None.wav names.txt
```

```
Анализ файла: Blizzard.wav
Количество совпадений: 15
Анализ файла: Decade_Dance.wav
Количество совпадений: 15
Анализ файла: Silver_Lights.wav
Количество совпадений: 45
Анализ файла: Voyager.wav
Количество совпадений: 40
Анализ файла: Dust.wav
Количество совпадений: 42
Анализ файла: Simma_Hem.wav
Количество совпадений: 38
Анализ файла: Divide.wav
Количество совпадений: 38
Анализ файла: Hollywood_Heights.wav
Количество совпадений: 19
Анализ файла: Interlude.wav
Количество совпадений: 34
Анализ файла: Around.wav
Количество совпадений: 12
Анализ файла: Java.wav
Количество совпадений: 1
Анализ файла: Rust.wav
Количество совпадений: 12
Анализ файла: Roller_Mobster.wav
Количество совпадений: 16
Анализ файла: Run.wav
Количество совпадений: 44
Не найдено!
```

4 Выводы

В ходе выполнения курсового проекта я изучил алгоритм быстрого вычисления дискретного преобразования Фурье — быстрое преобразование Фурье, а также реализовал алгоритм аудиопоиска. Уже давно я пытался сделать такого рода программу, но никак не получалось. Хоть информации в интернете на эту тему достаточно, но нет ответов на некоторые вопросы, которые у меня возникали, и из-за которых я не мог продвинуться дальше. Я очень рад, что наконец-то добился своей цели и во всем разобрался. Думаю, такого рода опыт мне не помешает. Но я не только узнал много нового, но и получил удовольствие от процесса.

Список литературы

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

[2] *Формат звуковых файлов WAV*
URL: `https://radioprog.ru/post/1025`).

[3] *Что такое преобразование Фурье? Иллюстрированное введение*
URL: `https://www.youtube.com/watch?v=spUNpyF58BY`.

[4] *Лекция 10. Быстрое преобразование Фурье*
URL: `https://www.youtube.com/watch?v=9Lr4qc0T1oI`.

[5] *Преобразование Фурье в действии: точное определение частоты сигнала и выделение нот*
URL: `https://habr.com/ru/post/247385/`.