

Практикум на ЭВМ. Преобразование Фурье.

Баев А.Ж.

Казахстанский филиал МГУ

22 февраля 2019

- 1 Непрерывное преобразование Фурье
- 2 Дискретное преобразование Фурье
- 3 Обработка звука
- 4 WAV-файлы

Непрерывное преобразование Фурье

Дана комплекснозначная функция от вещественного аргумента (времени) при $t \in [0, 1]$

$$f(t)$$

Разложим его по базису из комплекснозначных функций $\{\varphi_i = e^{2\pi kti}\}$

$$f(t) = c_0 \cdot 1 + c_1 e^{2\pi ti} + c_2 e^{4\pi ti} + c_3 e^{6\pi ti} + \dots$$

Непрерывное преобразование Фурье

Базисных функций бесконечно много, как и значений функций.
Если взять n (конечное число) значений функции, то можно взять конечное число базисных функций!

Дискретное преобразование Фурье

Дан вектор комплексных чисел

$$\vec{f} = (f_0, f_1, \dots, f_{n-1})$$

Разложим его по базису из комплексных векторов $\{\varphi_i\}$

$$\vec{f} = c_0\vec{\varphi}_0 + c_1\vec{\varphi}_1 + c_2\vec{\varphi}_2 + \dots + c_{n-1}\vec{\varphi}_{n-1}$$

Если $\{\varphi_i\}$ — ортогональный базис, то легко найти коэффициенты:

$$c_i = \frac{(f_i, \varphi_i)}{(\varphi_i, \varphi_i)}$$

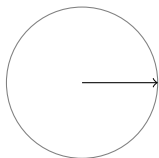
Дискретное преобразование Фурье

Выберем в качестве базисного вектора $\vec{\varphi}_k$ — значения сеточной функции

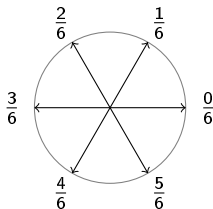
$$e^{2\pi kxi}$$

для значений x на равномерной сетке $[0, 1)$ с шагом $\frac{1}{n}$.

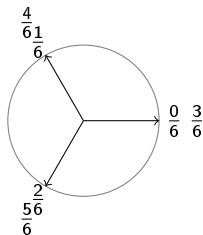
Например, при $n = 6$. Рассмотрим значения при $x = 0, \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}$



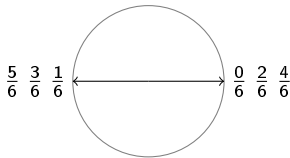
$$\varphi_0(x) = 1$$



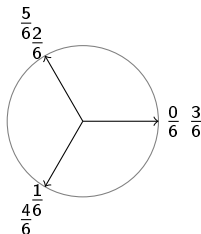
$$\varphi_1(x) = e^{2\pi x}$$



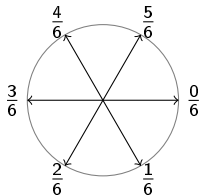
$$\varphi_2(x) = e^{4\pi x}$$



$$\varphi_3(x) = e^{6\pi x}$$



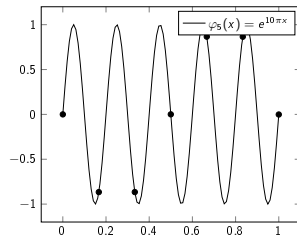
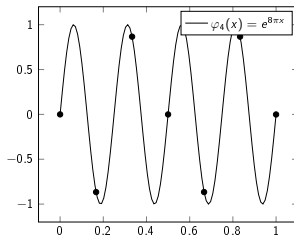
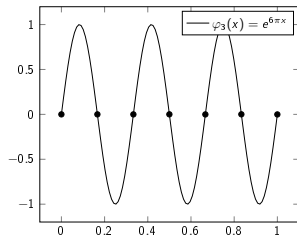
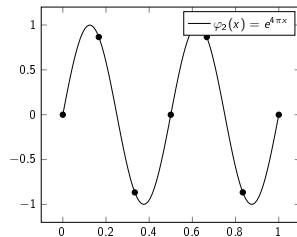
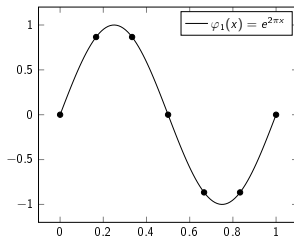
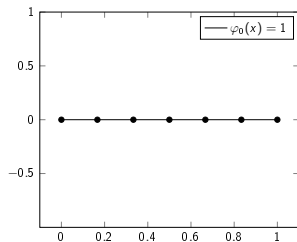
$$\varphi_4(x) = e^{8\pi x}$$



$$\varphi_5(x) = e^{10\pi x}$$

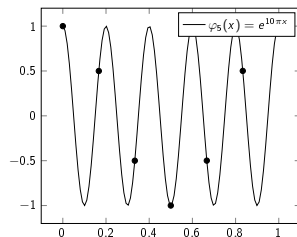
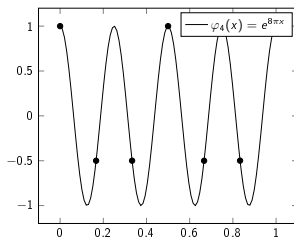
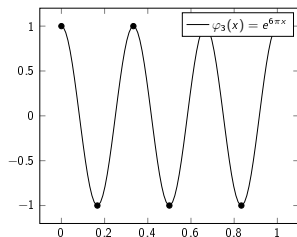
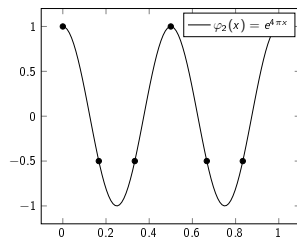
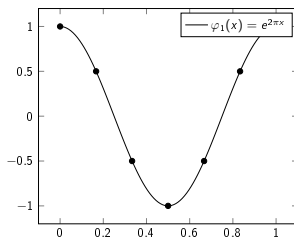
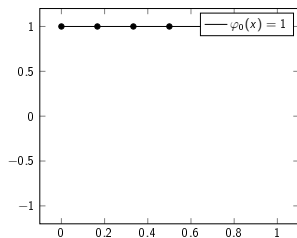
Дискретное преобразование Фурье

Мнимая часть



Дискретное преобразование Фурье

Вещественная часть



Дискретное преобразование Фурье

Обозначим через

$$\lambda = e^{\frac{2\pi}{n}i}$$

Тогда

$$\varphi_k\left(\frac{j}{n}\right) = e^{\frac{2\pi k}{n}i} = \lambda^{kj}$$

Здесь j соответствует шкале времени, k соответствует частоте функции.

Обратное преобразование Фурье

Разложение по базису:

$$f_j = \sum_{k=0}^{n-1} c_k \lambda^{kj} = c_0 + c_1 \lambda^j + c_2 \lambda^{2j} + \dots + c_{n-1} \lambda^{(n-1)j}$$

Прямое преобразование Фурье

Коэффициенты разложения в ряд Фурье:

$$c_k = \frac{1}{n} \sum_{j=0}^{n-1} f_j \lambda^{-kj} = \frac{1}{n} \left(f_0 + f_1 \lambda^{-k} + f_2 \lambda^{-2k} + \dots + f_{n-1} \lambda^{-(n-1)k} \right)$$

Опишем класс комплексных чисел:

```
class Complex {
    double _re, _im;
public:
    Complex();
    Complex(double re, double im = 0);
    double re() const;
    double amp() const;
    double phase() const;
    Complex operator+=(const Complex &z) const;
    Complex operator*(double x) const;
    Complex operator/(double x) const;
    Complex operator^(double alpha) const;
};
```

Значения: амплитуды `amp()` и фазы `phase()`.

Операторы: сложения с комплексным числом, умножения на вещественное, деления на вещественное и поворот на α радиан (то есть умножение на $e^{2\pi i \alpha} = \cos 2\pi \alpha + i \sin 2\pi \alpha$).

Пример сигнала (код)

Создадим вектор из чисел, которые описывают значения исходной функции (сигнала).

```
std::vector<Complex> function({1, 6, 2, 5, 3, 4});
```

Пример сигнала

Сигнал:

$$f_0 = 1$$

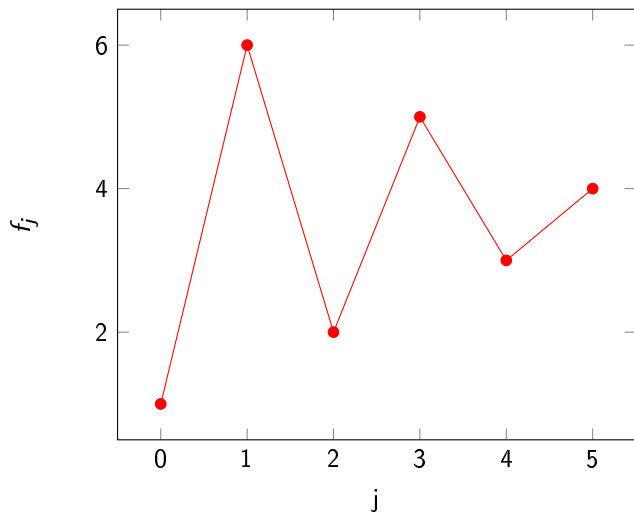
$$f_1 = 6$$

$$f_2 = 2$$

$$f_3 = 5$$

$$f_4 = 3$$

$$f_5 = 4$$



Пример спектра (код)

Создадим вектор из чисел, которые описывают спектр исходной функции.

```
std::vector<Complex> f2s(const std::vector<Complex> &f)
{
    std::vector<Complex> c;
    int n = f.size();
    c.resize(n);
    for (int k = 0; k < n; ++k) {
        for (int j = 0; j < n; ++j)
            c[k] += f[j] ^ ((- 2 * M_PI * k * j) / n);
        c[k] = c[k] / n;
    }
    return c;
}

...
std::vector<Complex> spectr = f2s(function);
```


Пример спектра

Спектр:

$$c_0 = 3.5$$

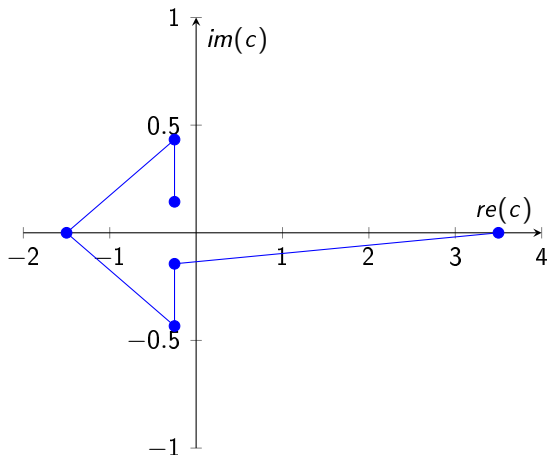
$$c_1 = -0.25 - 0.144i$$

$$c_2 = -0.25 - 0.433i$$

$$c_3 = -1.5$$

$$c_4 = -0.25 + 0.433i$$

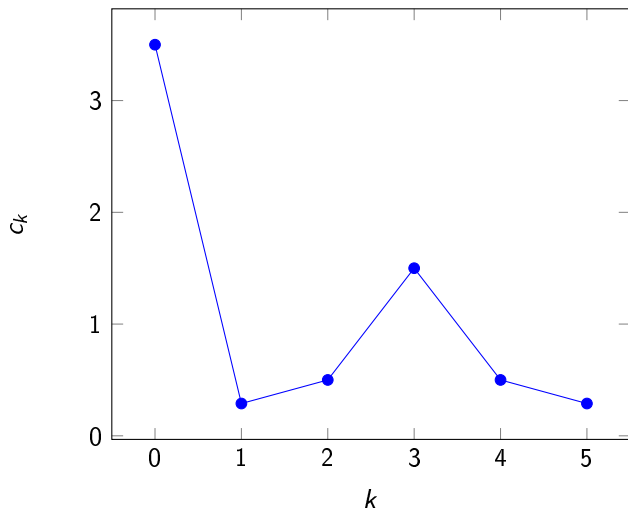
$$c_5 = -0.25 + 0.144i$$



Пример спектра

Амплитуды спек-
тра:

$$\begin{aligned}|c_0| &= 3.5 \\ |c_1| &= 0.289 \\ |c_2| &= 0.5 \\ |c_3| &= 1.5 \\ |c_4| &= 0.5 \\ |c_5| &= 0.289\end{aligned}$$



Пример восстановления функции (код)

```
std::vector<Complex> s2f(const std::vector<Complex> &c)
{
    std::vector<Complex> f;
    int n = c.size();
    f.resize(n);
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < n; ++k)
            f[j] += c[k] ^ ((2 * M_PI * k * j) / n);
    return f;
}

...

std::vector<Complex> function_answer = s2f(spectr );
```

Скачиваем WAV файл

BBC выложил более 16 000 звуковых эффектов в бесплатный доступ для образовательных целей:

<http://bbcsfx.acropolis.org.uk/>

Для примера выберем в категории Telephones «Touch tone phone - quick dial 7 digits.» на 7 секунд.

<http://bbcsfx.acropolis.org.uk/assets/07070057.wav>

Скачиваем WAV файл

Для начала посмотрим свойства файла в терминале:

```
file 07070057.wav
```

Получим:

```
07070057.wav: RIFF (little-endian) data, WAVE audio,  
Microsoft PCM, 16 bit, stereo 44100 Hz
```

Здесь есть полезная информация: битность 16 и частота дискретизации 44100.

Работа с бинарными файлами на C++.

Данные в файле записаны в бинарном виде. Считаем всё побайтово.
Конструктор итератора с потоком

```
std::istreambuf_iterator<char>(input)
```

создает указатель на начало данных в потоке.
Конструктор по умолчанию

```
std::istreambuf_iterator<char>()
```

создает указатель на «конец файла».

Итоговый код, который записывает бинарные данные в вектор buf:

```
std::ifstream input("07070057.wav", std::ios::binary);  
std::istreambuf_iterator<char> start(input);  
std::istreambuf_iterator<char> end;  
std::vector<char> buf(start, end);
```

Извлекаем данные из WAV файла

Подробнее про формат

<https://audiocoding.ru/article/2008/05/22/wav-file-structure.html>

Первая секция находится на 36 байте. Каждая секция состоит из:

- 1 имени из 4 ascii символов (4 байта);
- 2 числа n , которое задает размер секции (4 байта);
- 3 сырые данные (n байт).

```
std::string chunk_name(buf.begin() + chunk_start,
                        buf.begin() + chunk_start + 4);
chunk_start += 4;

int chunk_size = *(int*)(buf.data() + chunk_start);
chunk_start += 4;
```

Извлекаем данные из WAV файла

Если имя секции отлично от «data», то пропустим эту секцию (сдвинемся на $(8 + \text{chunk_size})$).

```
while (chunk_name != "data") {  
    chunk_start += chunk_size;  
    chunk_name = std::string(buf.begin() + chunk_start,  
                             buf.begin() + chunk_start + 4);  
    chunk_start += 4;  
    chunk_size = *(int*)(buf.data() + chunk_start);  
    chunk_start += 4;  
}
```

Например, в приведённом файле 07070057.wav целых 3 секции с метаданными (то есть не относятся к звуку):

name	start	size
sav1	44	42
bext	94	642
pad	744	16
data	768	1159536

Извлекаем данные из WAV файла

Чтобы извлечь данные из файла 07070057.wav надо понимать сколько занимает места один дискретный тик: 2 канала по 16 бит каждый. То есть 4 байта на тик.

На каждую амплитуду приходится один short:

```
*(short *) (buf.data() + i)
```

Забираем только четные амплитуды (один канал).

```
std::vector<Complex> function;  
for (int i = 0; i < chunk_size; i+=4) {  
    short value = *(short *) (buf.data() + i);  
    function.push_back(value);  
}
```

Всего получилось 289 884 амплитуды. Учитывая дискретизацию в 44 100 Гц, можно получить длительность записи: 6.573 секунды.

Смотрим на звук из WAV файла

Поскольку звук начинается не сразу, то все начальные значения амплитуд почти не отличаются от 0. Выведем 4410 амплитуд начиная с 5000-й:

```
std::vector<Complex> function = parse(buf);
int shift = 5000, size = 4410;
std::vector<Complex> function_part(
    function.begin() + shift,
    function.begin() + shift + size
);
```

Результат отправим в файл

```
std::ofstream file_function;
file_function.open("f.txt");
for (int i = 0; i < size; i++)
    file_function << i + shift << ' '
                  << function_part[i].re() << '\n';
file_function.close();
```

Смотрим на звук из WAV файла

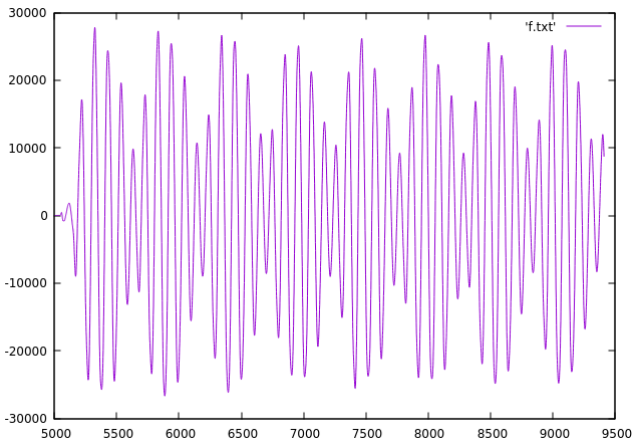
Посмотрим на график из них

```
gnuplot -p -e "plot 'f.txt' with lines"
```

Суффикс `-p` разрешает создание отдельного окна, `-e` выполняет команды `gnuplot` из строки. Команда `plot` строит данные из файла. Параметры `with lines` задают соединение точек.

Смотрим на звук из WAV файла

Частота дискретизации 44100 означает, что на 1 секунду приходится 44100 амплитуд. То есть данная картинка показывает звук длительностью чуть более 0.1 секунды.



Смотрим на звук из WAV файла

Вычислим спектр и для такого отрезка.

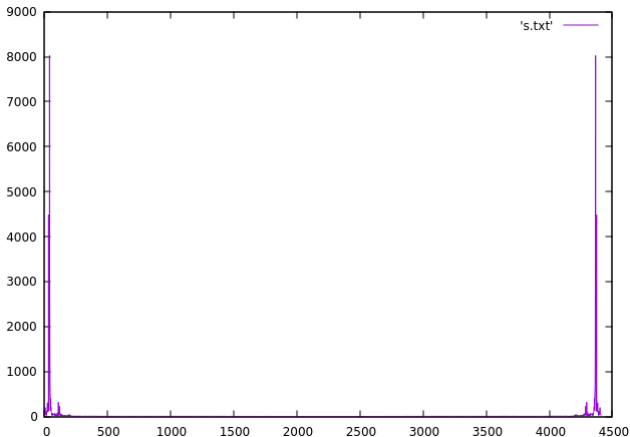
```
std::vector<Complex>   spectr = f2s(function_part);

std::ofstream file_spectr;
file_spectr.open ("s.txt");
for (int i = 0; i < size; i++)
    file_spectr << i << ' ' << spectr[i].amp() << '\n';
file_spectr.close();
```

Обратите внимание, что выводятся модули комплексных чисел!!!

Смотрим на звук из WAV файла

Явно выбивается одна частота

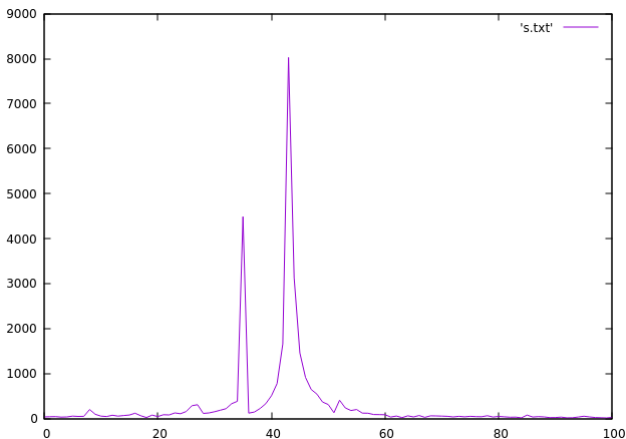


Рассмотрим поближе

```
gnuplot -p -e "plot[0:100] 's.txt' with lines"
```

Смотрим на звук из WAV файла

Примерно 41-й коэффициенты Фурье из 1000.



На 1000 коэффициентов приходится диапазон в 44100 Гц. Значит, шаг дискретизации по частотам — $44100/44100 = 10$ Гц. То есть в данном сегменте явно превалирует звук частота звука 440 Гц.

Смотрим на звук из WAV файла

Подпишем корректно ось времени

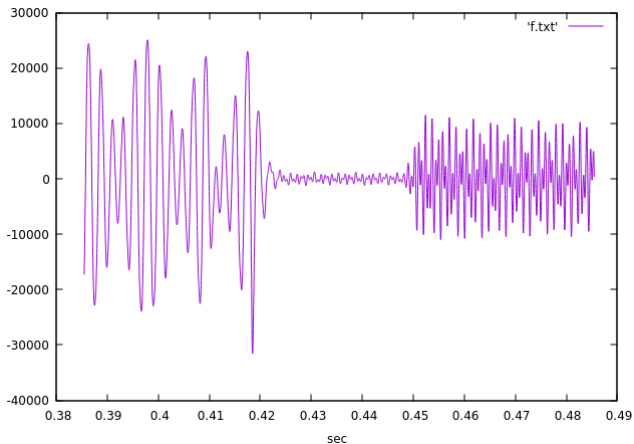
```
for (int i = 0; i < size; i++) {  
    double time = (i + shift) / 44100.0;  
    file_function << time << '␣'  
                  << function_part[i].re() << '\n';  
}
```

Подпишем корректно ось частот

```
for (int i = 0; i < size; i++) {  
    double frequency = 44100.0 / size * i ;  
    file_spectr << frequency << '␣'  
                << spectr[i].amp() << '\n';  
}
```

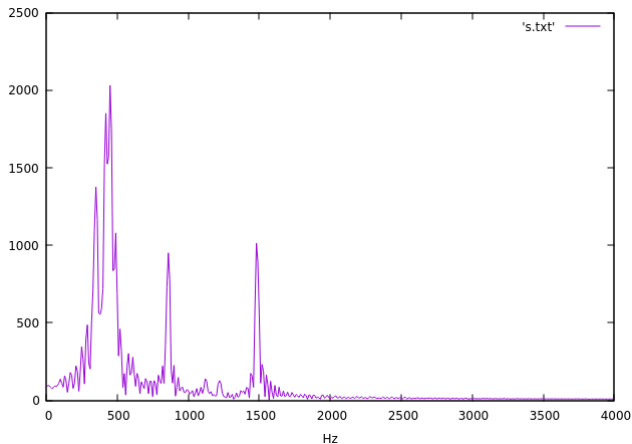

Смотрим на звук из WAV файла

Аналогичные картинки для старта на 1700-й амплитуде и длительностью 4410 амплитуд



Переключается звук.

Смотрим на звук из WAV файла



Смешивается частота 410 Гц и 1500 Гц.