

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет Отдел магистратуры

Кафедра Математика и информатика

Предмет Методы обработки сигналов и изображений

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

**Тема «ДИСКРЕТНЫЕ ПРЕОБРАЗОВАНИЯ И АЛГОРИТМЫ ЦОС»**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Автор |  | | |  | Дроботов С.В. | | | |
|  | (подпись, дата) | | |  |  | | | |
| Обозначение | ОМ.25 0000.000 | | Группа | | | | МСК11 |
| Направление подготовки | | 09.04.02 Информационные системы и технологии | | | | | |
| Профиль | Искусственный интеллект, математическое моделирование и суперкомпьютерные технологии в разработке информационных систем | | | | | | |
| Проверил |  | | |  | | Литвинов В.Н. | |
|  | (подпись, дата) | | |  | |  | |

Ростов-на-Дону

2025г.

Цель работы

Сгенерировать сигнал на основе заданных гармоник и после чего восстановить их с помощью быстрых преобразований Фурье.

Выполнение работы

В таблице ниже приведены использованные: амплитуда, частота, начальная фаза.

|  |  |  |
| --- | --- | --- |
| **Амплитуда** | **Частота (Гц)** | **Начальная фаза (рад)** |
| 2.0 | 50.0 | 0.0 |
| 1.5 | 100.0 | 0.7854 (π/4) |
| 1.0 | 200.0 | 1.5708 (π/2) |
| 0.8 | 300.0 | 2.3562 (3π/4) |
| 0.6 | 400.0 | 3.1416 (π) |
| 0.5 | 500.0 | 0.5236 (π/6) |
| 0.4 | 600.0 | 1.0472 (π/3) |
| 0.3 | 700.0 | 2.0944 (2π/3) |
| 0.2 | 800.0 | 2.6179 (5π/6) |
| 0.1 | 900.0 | 0.3927 (π/8) |

**Определение гармоник**

На рисунке 1 приведен пример реализации класса, содержащего данные гармоник.

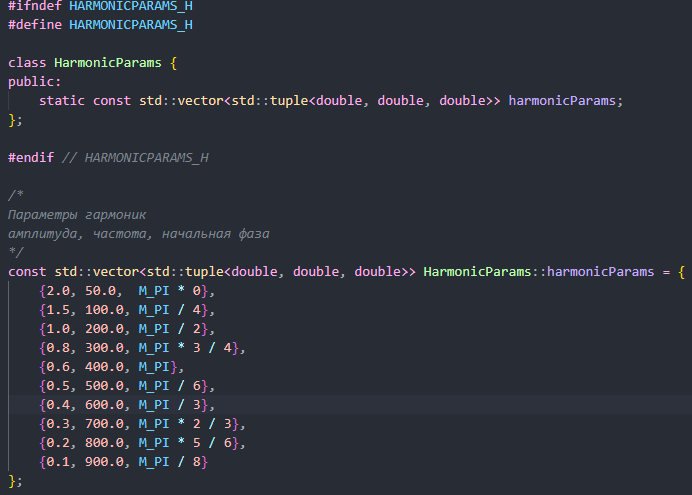


Рис 1

Данный код на C++ реализует класс HarmonicParams, предназначенный для хранения параметров гармоник (амплитуды, частоты и начальной фазы) в виде статического контейнера. Код разделен на заголовочный файл (HarmonicParams.h) и файл с инициализацией статических данных.

**Хранение данных**

Код на C++, приведенный на рисунке 2, реализует структуру SignalData, предназначенную для хранения временных и частотных данных сигнала.

Далее он будет использован для хранения данных импортов и экспортов из файла.

Код представлен в виде заголовочного файла, что позволяет легко подключать его в других модулях.

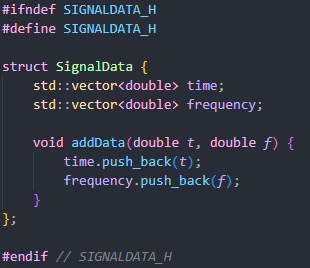


Рис 2

**Операции с гармониками**

Класс HarmonicUtils предоставляет набор статических методов для работы с гармоническими сигналами, включая генерацию, обработку, экспорт и импорт данных. Код разделен на логические блоки.

1. generateHarmonic
   * Генерирует гармонический сигнал.
   * Особенности:
     + Формула: amplitude⋅sin(2π⋅frequency⋅t+phase).
     + Возвращает вектор значений сигнала.
   * Пример реализации метода приведен на рисунке 3.
2. processSignal
   * Обрабатывает сигнал с помощью FFT, фильтрует компоненты и восстанавливает сигнал.
   * Этапы работы:
     + FFT (прямое преобразование Фурье):
       - Используется библиотека FFTW3 для преобразования сигнала в частотную область.
       - План FFT создается с помощью fftw\_plan\_dft\_r2c\_1d.
     + Расчет амплитуд и частот:
       - Амплитуды вычисляются как модуль комплексных значений FFT.
       - Частоты рассчитываются по формуле
     + Фильтрация компонент:
       - Компоненты сортируются по убыванию амплитуды.
       - Отбираются компоненты до достижения целевой энергии, определяемой targetDistortion.
     + Обратное FFT:
       - Восстановление сигнала из отфильтрованных компонент.
   * Пример реализации метода приведен на рисунке 4.
3. exportData

* Экспортирует данные сигнала в файл.
* Пример реализации метода приведен на рисунке 5.

1. importData

* Импортирует данные сигнала из файла.
* Пример реализации метода приведен на рисунке 6.

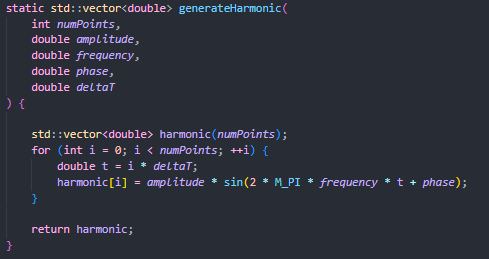


Рис 3

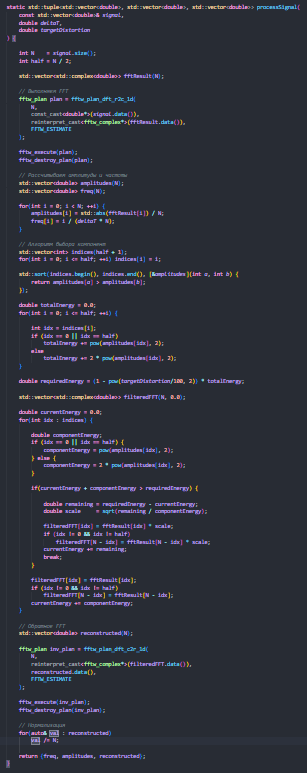


Рис 4

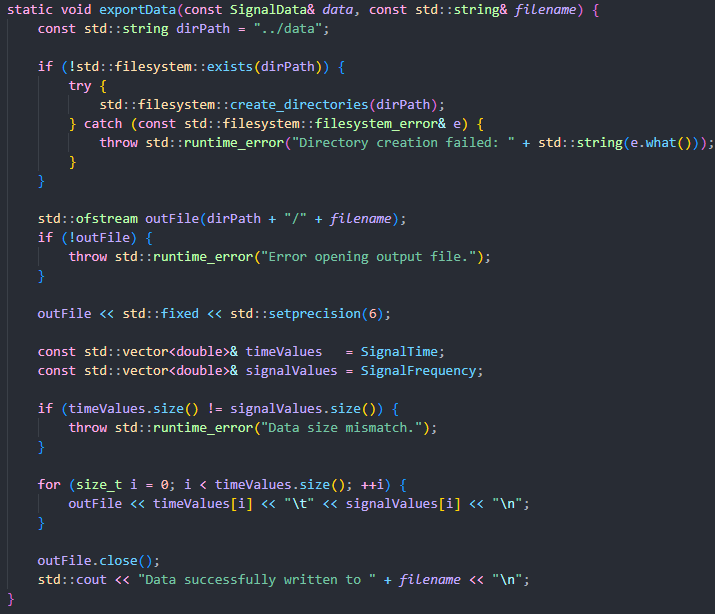


Рис 5

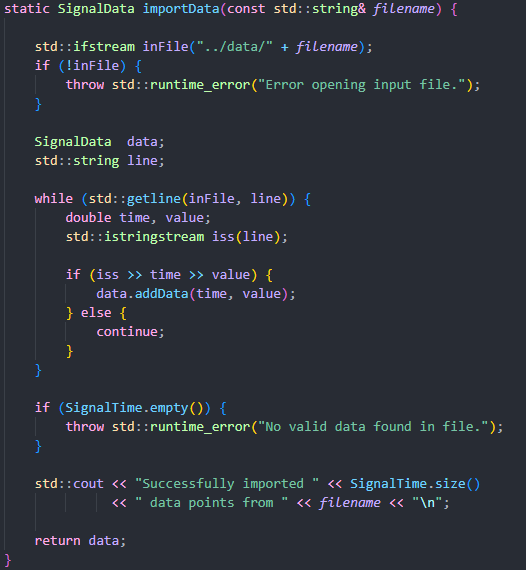


Рис 6

**Генерация данных**

Данная программа генерирует сложный сигнал, складывая несколько гармонических компонент, и сохраняет результат в файл.  
Код демонстрирует работу с классами SignalData, HarmonicUtils и HarmonicParams.

1. **Основная функция main():**
   * Создает объект data для хранения данных сигнала.
   * Задает параметры генерации:

const int numOfPoints = 10000; // Количество точек сигнала

const double deltaT = 1e-4; // Шаг времени (0.0001 сек)

* + Инициализирует векторы:

std::vector<double> resultSignal(numOfPoints, 0.0); // Результирующий сигнал

std::vector<double> time(numOfPoints); // Временные метки

1. **Генерация гармоник**:
   * Для каждой гармоники из HarmonicParams::harmonicParams:
     + Извлекает параметры: амплитуду, частоту, фазу.
     + Генерирует гармонический сигнал:

std::vector<double> harmonic = HarmonicUtils::generateHarmonic(

numOfPoints, amplitude, frequency, phase, deltaT

);

* + - Суммирует гармонику с результирующим сигналом:

for(int i = 0; i < numOfPoints; ++i) {

resultSignal[i] += harmonic[i];

time[i] = i \* deltaT; // Заполнение времени

}

1. **Сохранение данных**:
   * Добавляет данные в объект SignalData:

for(int i = 0; i < numOfPoints; ++i) {

data.addData(time[i], resultSignal[i]);

}

* + Экспортирует данные в файл:

HarmonicUtils::exportData(data, "merged\_signal\_data.txt");

После выполнения будет сгенерирован текстовый файл, содержащий информацию о времени и частоте.

**Построение сгенерированных данных**

1. **Загрузка данных**:
   * Импортирует данные сигнала из файла merged\_signal\_data.txt с помощью метода HarmonicUtils::importData.
   * Данные сохраняются в объект data класса SignalData, который хранит два вектора:
     + time (время),
     + frequency (амплитуда сигнала).
2. **Настройка графика**:
   * Устанавливает размер окна графика: 1200x600 пикселей.
   * Строит график сигнала, используя данные из векторов time и frequency.
   * Добавляет подписи осей (xlabel, ylabel), заголовок (title), легенду (legend) и сетку (grid).
3. **Отображение графика**:
   * Выводит график на экран через plt::show().

Пример кода для построения графика и его результат приведены на рисунке 7 и рисунке 8.

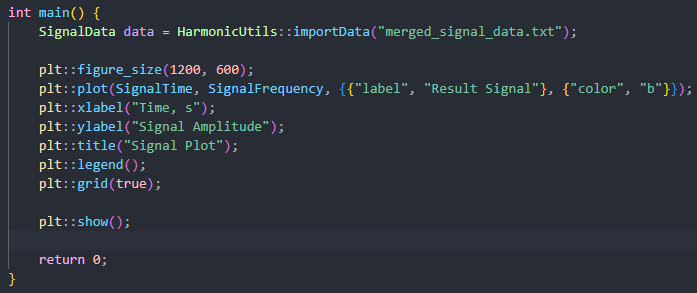


Рис 7

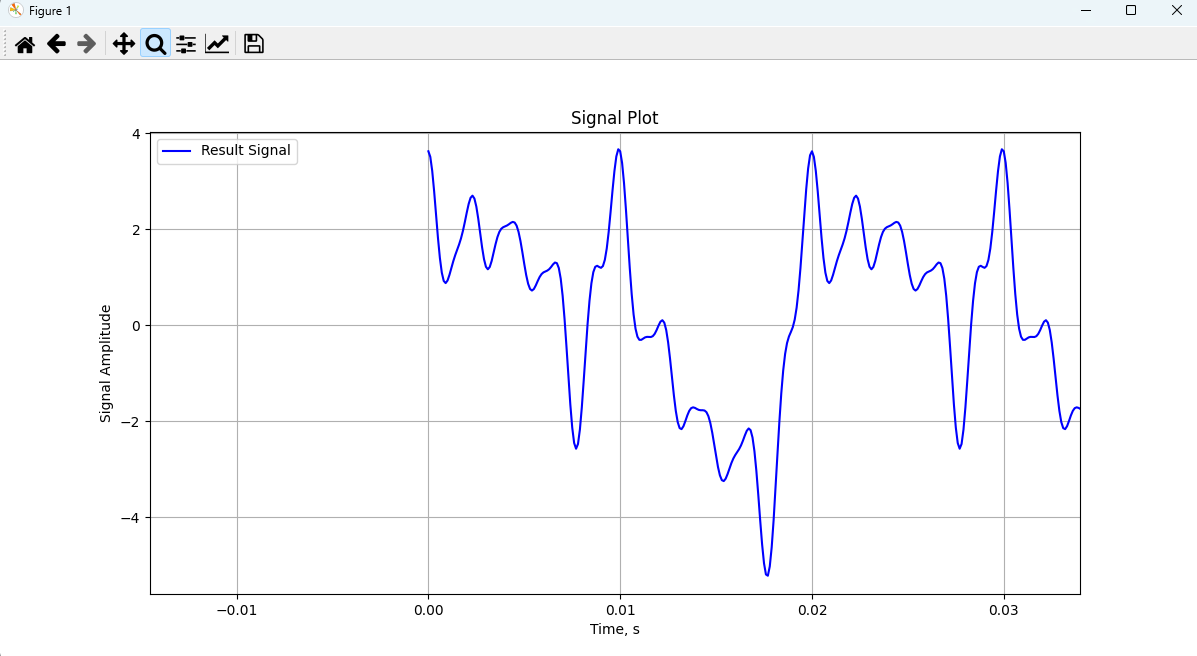


Рис 8

**Построение восстановленных данных**

**1. Загрузка данных**

* Загружает данные сигнала из файла merged\_signal\_data.txt, который содержит временные метки и амплитуды сигнала.

SignalData data = HarmonicUtils::importData("merged\_signal\_data.txt");

**2. Параметры обработки**

* Устанавливает параметры для обработки сигнала: количество точек, шаг времени и допустимый уровень искажений.

const int numOfPoints = 10000;

const double deltaT = 1e-4;

const double targetDistortion = 5.0;

**3. Подготовка данных**

std::vector<double> resultSignal = SignalFrequency;

std::vector<double> time = SignalTime;

**4. Обработка сигнала**

* Вызывает метод processSignal, который:

auto [freq, amplitudes, reconstructed] = HarmonicUtils::processSignal(

resultSignal, deltaT, targetDistortion

);

* + Выполняет FFT-преобразование сигнала.
  + Фильтрует частотные компоненты, чтобы достичь целевого уровня искажений.
  + Восстанавливает сигнал обратным FFT.

**5. Расчет искажений**

* Рассчитывает искажение сигнала как отношение среднеквадратичной ошибки к мощности исходного сигнала.

double mse = 0.0, power = 0.0;

for(int i = 0; i < numOfPoints; ++i) {

mse += pow(resultSignal[i] - reconstructed[i], 2);

power += pow(resultSignal[i], 2);

}

double distortion = sqrt(mse / power) \* 100;

**6. Визуализация**

**График 1: Сравнение сигналов во временной области**

* Строит два графика:
  + Исходный сигнал (синий, полупрозрачный).
  + Восстановленный сигнал (красный, пунктирный).

plt::figure\_size(1200, 600);

plt::plot(time, resultSignal, {{"label", "Original"}, {"alpha", "0.7"}});

plt::plot(time, reconstructed, {{"label", "Reconstructed"}, {"linestyle", "--"}, {"color", "r"}});

plt::title("Signal Comparison (Distortion: " + std::to\_string(distortion) + "%)");

plt::xlabel("Time, s");

plt::ylabel("Amplitude");

plt::legend();

plt::grid(true);

plt::show();

**График 2: Амплитудный спектр**

* Отображает амплитуды частотных компонентов до половины частоты Найквиста (учет симметрии FFT).

std::vector<double> freqHalf(freq.begin(), freq.begin() + numOfPoints / 2);

std::vector<double> ampsHalf(amplitudes.begin(), amplitudes.begin() + numOfPoints / 2);

for(double& amp : ampsHalf) amp \*= 2;

plt::stem(freqHalf, ampsHalf, {{"basefmt", " "}, {"linefmt", "b-"}, {"markerfmt", "bo"}});

plt::xlabel("Frequency, Hz");

plt::ylabel("Amplitude");

plt::title("Amplitude Spectrum");

plt::grid(true);

plt::show();

Пример кода для построения графика и его результат приведен на рисунке 9 рисунке 10 и рисунке 11.



Рис 9

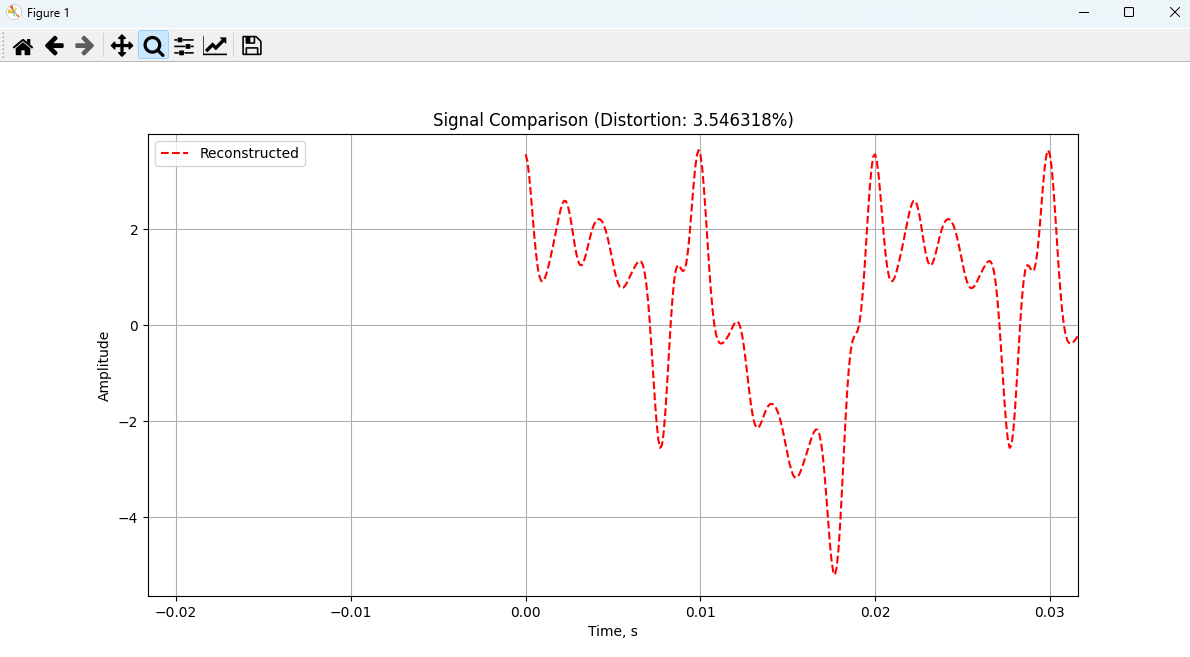


Рис 10

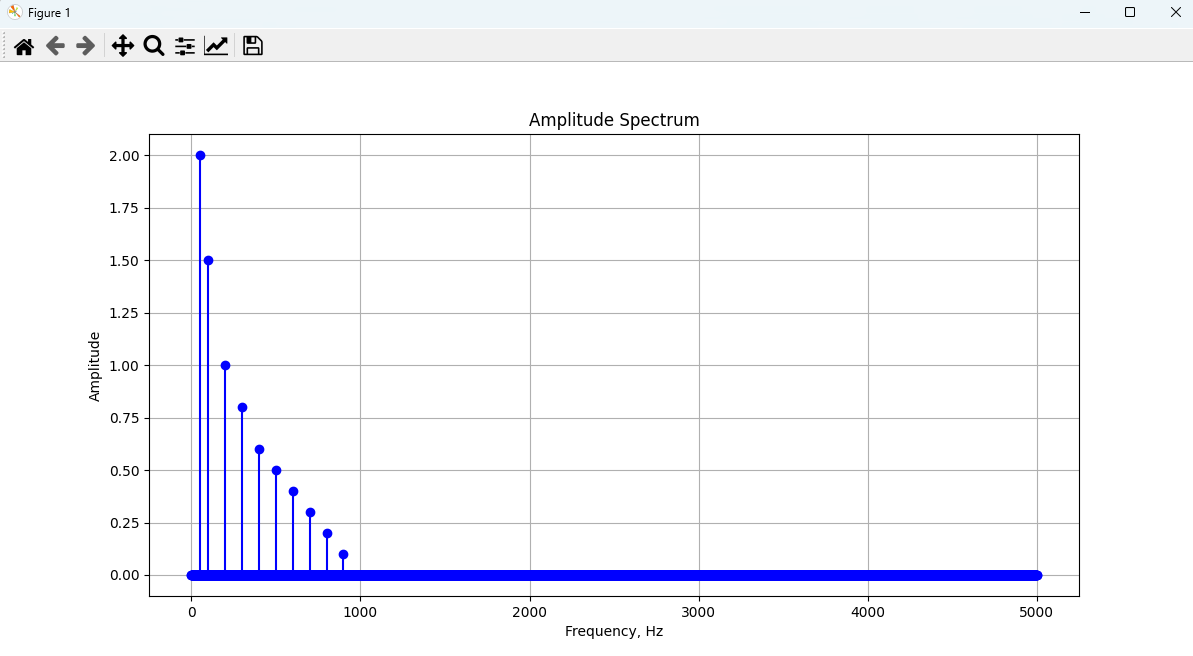


Рис 11

Вывод

На основе проведенной работы была реализована генерация гармоник и восстановление исходных данных с помощью преобразования Фурье. Данные были восстановлены с точностью 3,546318%, что укладывается в производственные стандарты (до 5%). Так же удалось восстановить все исходные гармоники.