

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет «Информатика и вычислительная техника»

Кафедра «Математика и информатика»

**Лабораторная работ**

**по дисциплине « Методы обработки сигналов и изображений»**

Выполнил:

Студент Гантсу С.Ж.

Группа МСК11

Проверил:

Литвинов В.Н.

**Цель работы**

Разработка программы на C++, которая:

1. Генерирует периодический несинусоидальный сигнал на основе заданных гармоник.
2. Позволяет вводить параметры гармоник (амплитуду, частоту, фазу) вручную или из файла.
3. Квантует сигнал и сохраняет данные для визуализации.
4. Анализирует сигнал с помощью быстрого преобразования Фурье (БПФ) в Python.

**Общее описание программы**

Данная программа предназначена для:

1. Генерации сигнала, состоящего из суммы гармонических составляющих
2. Квантования полученного сигнала с заданным числом уровней
3. Сохранения результатов в CSV-файл для дальнейшего анализа

**Структура программы**

Программа состоит из двух частей:

C++-код (main.cpp) — генерирует сигнал, квантует его и сохраняет в CSV.

Python-скрипт (visualize.py) — строит графики сигнала и его спектра.

**Функциональные модули**

1. **Генерация сигнала** (generateSignal):
   * Создает временной ряд на основе суммы гармонических составляющих
   * Использует параметры: список гармоник, частоту дискретизации, длительность сигнала
2. **Квантование сигнала** (quantizeSignal):
   * Преобразует непрерывный сигнал в дискретные уровни
   * Равномерное квантование в диапазоне [-maxAmplitude, maxAmplitude]
3. **Работа с файлами**:
   * readHarmonicsFromFile - чтение параметров гармоник из файла
   * writeToFile - запись результатов (время, сигнал, квантованный сигнал) в CSV

Пользовательский интерфейс

Программа взаимодействует с пользователем через консоль, запрашивая:

1. Способ ввода гармоник (из файла или вручную)
2. Параметры каждой гармоники (амплитуда, частота, фаза)
3. Параметры дискретизации (частота и длительность)
4. Число уровней квантования

Входные и выходные данные

**Входные данные:**

* Параметры гармонических составляющих
* Частота дискретизации (Гц)
* Длительность сигнала (секунды)
* Число уровней квантования

**Выходные данные:**

* CSV-файл signal\_data.csv с колонками:
  + Time - временная метка
  + Signal - значение сигнала
  + Quantized - квантованное значение

Алгоритмы и методы

1. **Генерация сигнала**: линейная комбинация гармонических функций
2. **Квантование**: равномерное квантование с округлением до ближайшего уровня
3. **Дискретизация**: равномерная по времени с заданной частотой

**Ограничения**

1. Предполагается, что сумма амплитуд гармоник не превышает максимальный уровень квантования
2. Используется равномерное квантование, что может быть неоптимально для некоторых типов сигналов
3. Фаза гармоник задается в радианах

Возможные улучшения

1. Добавление разных типов квантования (неравномерное, адаптивное)
2. Визуализация результатов непосредственно в программе
3. Поддержка других форматов выходных файлов
4. Добавление фильтрации сигнала перед квантованием

#include <iostream>

#include <fstream>

#include <vector>

#include <cmath>

#include <algorithm>

#include <string>

using namespace std;

struct Harmonic {

    double amplitude;

    double frequency;

    double phase;

};

// Функция для генерации сигнала

vector<double> generateSignal(const vector<Harmonic>& harmonics,

    double samplingRate,

    double duration,

    int& numPoints) {

    double deltaT = 1.0 / samplingRate;

    numPoints = static\_cast<int>(duration \* samplingRate);

    vector<double> signal(numPoints, 0.0);

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

        double t = i \* deltaT;

        for (const auto& h : harmonics) {

            signal[i] += h.amplitude \* sin(2 \* M\_PI \* h.frequency \* t + h.phase);

        }

    }

    return signal;

}

// Функция для квантования сигнала

vector<int> quantizeSignal(const vector<double>& signal, int levels, double maxAmplitude) {

    vector<int> quantized(signal.size(), 0);

    double step = (2 \* maxAmplitude) / levels;

    for (size\_t i = 0; i < signal.size(); ++i) {

        quantized[i] = static\_cast<int>((signal[i] + maxAmplitude) / step);

    }

    return quantized;

}

// Функция для записи данных в файл

void writeToFile(const string& filename,

    const vector<double>& time,

    const vector<double>& signal,

    const vector<int>& quantized = vector<int>()) {

    ofstream out(filename);

    if (!out.is\_open()) {

        cerr << "Error opening file: " << filename << endl;

        return;

    }

    out << "Time,Signal";

    if (!quantized.empty()) {

        out << ",Quantized";

    }

    out << "\n";

    for (size\_t i = 0; i < time.size(); ++i) {

        out << time[i] << "," << signal[i];

        if (!quantized.empty() && i < quantized.size()) {

            out << "," << quantized[i];

        }

        out << "\n";

    }

    out.close();

}

// Функция для чтения гармоник из файла

vector<Harmonic> readHarmonicsFromFile(const string& filename) {

    vector<Harmonic> harmonics;

    ifstream in(filename);

    if (!in.is\_open()) {

        cerr << "Error opening file: " << filename << endl;

        return harmonics;

    }

    Harmonic h;

    while (in >> h.amplitude >> h.frequency >> h.phase) {

        harmonics.push\_back(h);

    }

    in.close();

    return harmonics;

}

int main() {

    vector<Harmonic> harmonics;

    char choice;

    cout << "Read harmonics from file? (y/n): ";

    cin >> choice;

    if (choice == 'y' || choice == 'Y') {

        string filename;

        cout << "Enter filename: ";

        cin >> filename;

        harmonics = readHarmonicsFromFile(filename);

    }

    else {

        int numHarmonics;

        cout << "Enter number of harmonics: ";

        cin >> numHarmonics;

        harmonics.resize(numHarmonics);

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

            cout << "Harmonic " << i + 1 << ":\n";

            cout << "  Amplitude: ";

            cin >> harmonics[i].amplitude;

            cout << "  Frequency (Hz): ";

            cin >> harmonics[i].frequency;

            cout << "  Phase (rad): ";

            cin >> harmonics[i].phase;

        }

    }

    double samplingRate, duration;

    cout << "Enter sampling rate (Hz): ";

    cin >> samplingRate;

    cout << "Enter duration (s): ";

    cin >> duration;

    int numPoints;

    auto signal = generateSignal(harmonics, samplingRate, duration, numPoints);

    // Генерируем временную ось

    vector<double> time(numPoints);

    double deltaT = 1.0 / samplingRate;

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

        time[i] = i \* deltaT;

    }

    // Находим максимальную амплитуду для квантования

    double maxAmplitude = 0;

    for (const auto& h : harmonics) {

        maxAmplitude += h.amplitude;

    }

    // Квантование сигнала

    int quantizationLevels;

    cout << "Enter number of quantization levels: ";

    cin >> quantizationLevels;

    auto quantized = quantizeSignal(signal, quantizationLevels, maxAmplitude);

    // Запись результатов в файл

    writeToFile("signal\_data.csv", time, signal, quantized);

    cout << "Data written to signal\_data.csv\n";

    return 0;

}

Результат

