#include <iostream>

#include <vector>

#include <cmath>

#include <fstream>

#include <algorithm>

using namespace std;

// Определяем константу PI, если M\_PI недоступна

#ifndef M\_PI

#define M\_PI 3.14159265358979323846

#endif

struct Harmonic {

double amplitude;

double frequency;

double phase\_shift;

};

vector<double> generate\_signal(const vector<Harmonic>& harmonics,

double sampling\_rate,

double duration,

int& num\_points) {

double delta\_t = 1.0 / sampling\_rate;

num\_points = static\_cast<int>(duration \* sampling\_rate);

vector<double> signal(num\_points, 0.0);

for (int i = 0; i < num\_points; ++i) {

double t = i \* delta\_t;

for (const auto& h : harmonics) {

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

}

}

return signal;

}

void save\_to\_csv(const vector<double>& signal,

double sampling\_rate,

const string& filename) {

ofstream out(filename);

out << "time,value\n";

double delta\_t = 1.0 / sampling\_rate;

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

out << i \* delta\_t << "," << signal[i] << "\n";

}

}

void visualize\_in\_python(const string& csv\_file) {

string command = "python visualize.py " + csv\_file; // Изменено с python3 на python для лучшей совместимости

system(command.c\_str());

}

int main() {

vector<Harmonic> harmonics;

int choice;

cout << "1. Enter harmonics manually\n2. Read from file\n";

cin >> choice;

if (choice == 1) {

int num\_harmonics;

cout << "Number of harmonics: ";

cin >> num\_harmonics;

for (int i = 0; i < num\_harmonics; ++i) {

Harmonic h;

cout << "Harmonic " << i + 1 << " (amplitude frequency phase\_shift in radians): ";

cin >> h.amplitude >> h.frequency >> h.phase\_shift;

harmonics.push\_back(h);

}

}

else {

string filename;

cout << "Filename: ";

cin >> filename;

ifstream in(filename);

if (!in.is\_open()) {

cerr << "Error: Could not open file " << filename << endl;

return 1;

}

Harmonic h;

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

harmonics.push\_back(h);

}

}

double sampling\_rate, duration;

cout << "Sampling rate (Hz): ";

cin >> sampling\_rate;

cout << "Duration (s): ";

cin >> duration;

if (sampling\_rate <= 0 || duration <= 0) {

cerr << "Error: Sampling rate and duration must be positive" << endl;

return 1;

}

int num\_points;

auto signal = generate\_signal(harmonics, sampling\_rate, duration, num\_points);

// Find maximum amplitude for scaling

double max\_amp = \*max\_element(signal.begin(), signal.end(),

[](double a, double b) { return abs(a) < abs(b); });

cout << "Signal generated with " << num\_points << " points.\n";

cout << "Maximum amplitude: " << max\_amp << endl;

string outfile = "signal.csv";

save\_to\_csv(signal, sampling\_rate, outfile);

cout << "Data saved to " << outfile << endl;

visualize\_in\_python(outfile);

return 0;

}