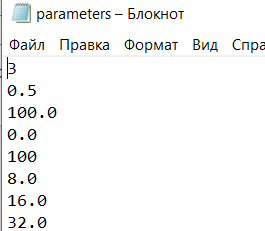
**Лабораторная работа №1**

Исходные данные 

1) это количество гармоник

2) это базовая частота

3) это частота дискретизации

4) это начальная частота сигнала

5-8 переменная величины гармоник (должны быть кратные)

Далее приведён код программы, которая генерирует исходный сигнал, а потом квантованный, данные импортируется из файла, а далее выводятся в каждый из файлов: исходный сигнал, квантованный сигнал, наложенные друг на друга сигналы.

## **Код программы**

#include <iostream>

#include <fstream>

#include <vector>

#include <cmath>

#define \_USE\_MATH\_DEFINES

#include <math.h>

struct Harmonic {

double amplitude;

double frequency;

};

struct Sample {

double time;

double value;

};

int main() {

std::string filename = "parameters.txt";

std::ifstream file(filename);

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

std::cerr << "File opening error " << filename << std::endl;

return 1;

}

int numHarmonics;

double baseFrequency;

double samplingFrequency;

double initialPhase;

file >> numHarmonics;

file >> baseFrequency;

file >> samplingFrequency;

file >> initialPhase;

std::vector<Harmonic> harmonics(numHarmonics);

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

file >> harmonics[i].amplitude;

harmonics[i].frequency = baseFrequency \* (i + 1);

}

file.close();

int numSamples = static\_cast<int>(samplingFrequency / baseFrequency) \* 10;

double period = 1.0 / baseFrequency;

double dt = period / numSamples;

std::vector<double> signal(numSamples);

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

double t = i \* dt;

double signalValue = 0.0;

for (const auto& harmonic : harmonics) {

signalValue += harmonic.amplitude \* std::sin(2 \* M\_PI \* harmonic.frequency \* t + initialPhase);

}

signal[i] = signalValue;

}

std::ofstream signalOutFile("signal.txt");

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

std::cerr << "Error opening the file for recording the signal." << std::endl;

return 1;

}

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

double t = i \* dt;

signalOutFile << t << " " << signal[i] << std::endl;

}

signalOutFile.close();

std::cout << "The signal is recorded in a file signal.txt" << std::endl;

std::string inputFilename = "signal.txt";

std::string outputFilename = "quantized\_signal.txt";

std::ifstream inputFile(inputFilename);

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

std::cerr << "File opening error " << inputFilename << std::endl;

return 1;

}

std::vector<Sample> signalSamples;

double time, value;

while (inputFile >> time >> value) {

signalSamples.push\_back({ time, value });

}

inputFile.close();

if (signalSamples.empty()) {

std::cerr << "file " << inputFilename << " empty." << std::endl;

return 1;

}

int numLevels = 16;

double minSignal = signalSamples[0].value;

double maxSignal = signalSamples[0].value;

for (const auto& sample : signalSamples) {

if (sample.value < minSignal) {

minSignal = sample.value;

}

if (sample.value > maxSignal) {

maxSignal = sample.value;

}

}

std::ofstream outputFile(outputFilename);

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

std::cerr << "File opening error " << outputFilename << std::endl;

return 1;

}

std::vector<Sample> quantizedSignal;

for (const auto& sample : signalSamples) {

double normalizedValue = (sample.value - (maxSignal + minSignal) / 2.0) / ((maxSignal - minSignal) / 2.0);

double quantizedNormalized = std::round(normalizedValue \* (numLevels - 1)) / (numLevels - 1);

double quantizedValue = quantizedNormalized \* ((maxSignal - minSignal) / 2.0) + (maxSignal + minSignal) / 2.0;

outputFile << sample.time << " " << quantizedValue << std::endl;

quantizedSignal.push\_back({ sample.time, quantizedValue });

}

outputFile.close();

std::cout << "The quantized signal is recorded in a file " << outputFilename << std::endl;

std::string inputFilename1 = "signal.txt";

std::string inputFilename2 = "quantized\_signal.txt";

std::string outputFilename3 = "overlapped\_signal.txt";

std::vector<Sample> originalSignal;

std::ifstream inputFile1(inputFilename1);

if (!inputFile1.is\_open()) {

std::cerr << "File opening error " << inputFilename1 << std::endl;

return 1;

}

std::ifstream inputFile2(inputFilename2);

if (!inputFile2.is\_open()) {

std::cerr << "File opening error " << inputFilename2 << std::endl;

return 1;

}

signalSamples.clear();

while (inputFile1 >> time >> value) {

originalSignal.push\_back({ time, value });

}

quantizedSignal.clear();

while (inputFile2 >> time >> value) {

quantizedSignal.push\_back({ time, value });

}

inputFile1.close();

inputFile2.close();

if (originalSignal.size() != quantizedSignal.size()) {

std::cerr << "The signals have different lengths." << std::endl;

return 1;

}

std::ofstream outputFile3(outputFilename3);

if (!outputFile3.is\_open()) {

std::cerr << "File opening error " << outputFilename3 << std::endl;

return 1;

}

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

outputFile3 << originalSignal[i].time << " " << originalSignal[i].value << " " << quantizedSignal[i].value << std::endl;

}

outputFile3.close();

std::cout << "Both signals are recorded in a file " << outputFilename3 << std::endl;

double mse = 0.0;

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

double error = originalSignal[i].value - quantizedSignal[i].value;

mse += error \* error;

}

mse /= originalSignal.size();

std::cout << "Average square error (SQR): " << mse << std::endl;

double signalPower = 0.0;

for (const auto& sample : originalSignal) {

signalPower += sample.value \* sample.value;

}

signalPower /= originalSignal.size();

double noisePower = mse;

double snr = 10 \* log10(signalPower / noisePower);

std::cout << "Signal-to-noise ratio (SNR) in decibels: " << snr << std::endl;

return 0;

}

## **Визуализация на Python**

import matplotlib.pyplot as plt

import numpy as np

data = np.loadtxt('signal.txt')

plt.figure(figsize=(10, 6))

plt.plot(data[:, 0], data[:, 1])

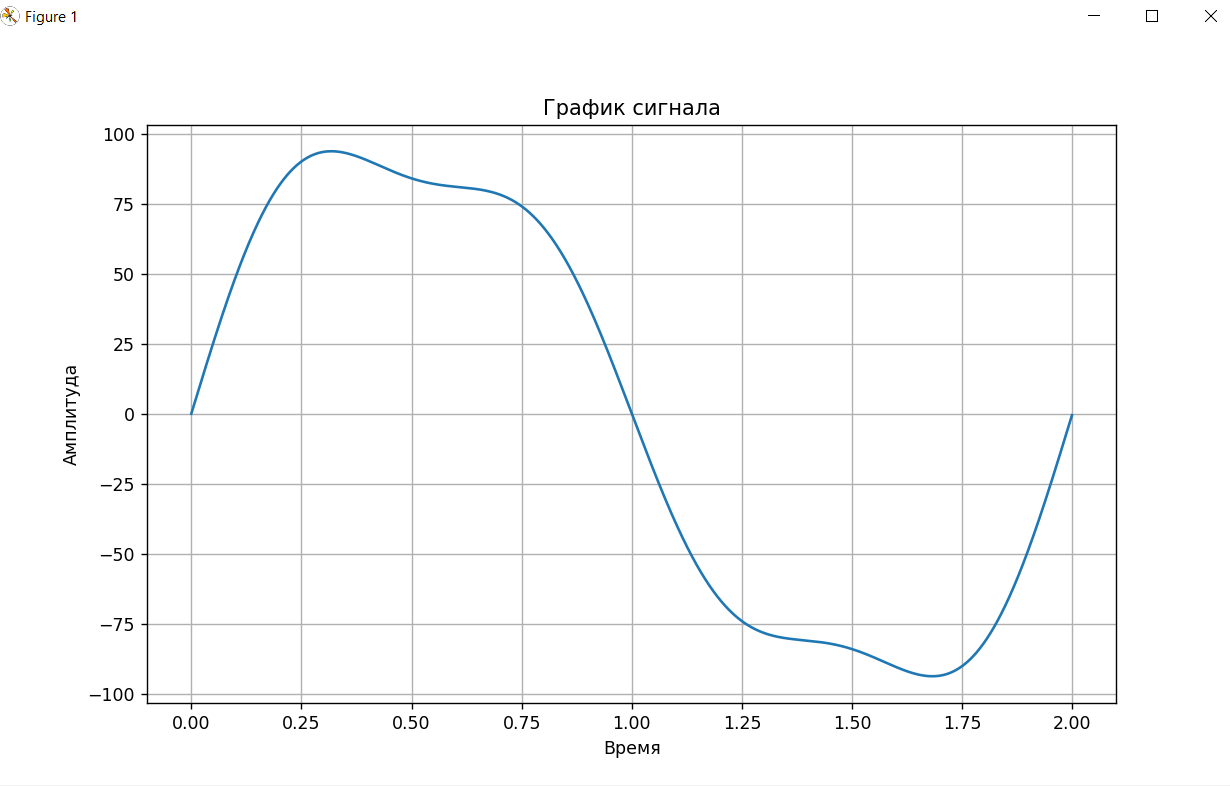
plt.xlabel('Время')

plt.ylabel('Амплитуда')

plt.title('График сигнала')

plt.grid(True)

plt.show()



**Квантованный сигнал**

import matplotlib.pyplot as plt

import numpy as np

data = np.loadtxt('quantized\_signal.txt')

plt.figure(figsize=(10, 6))

plt.plot(data[:, 0], data[:, 1])

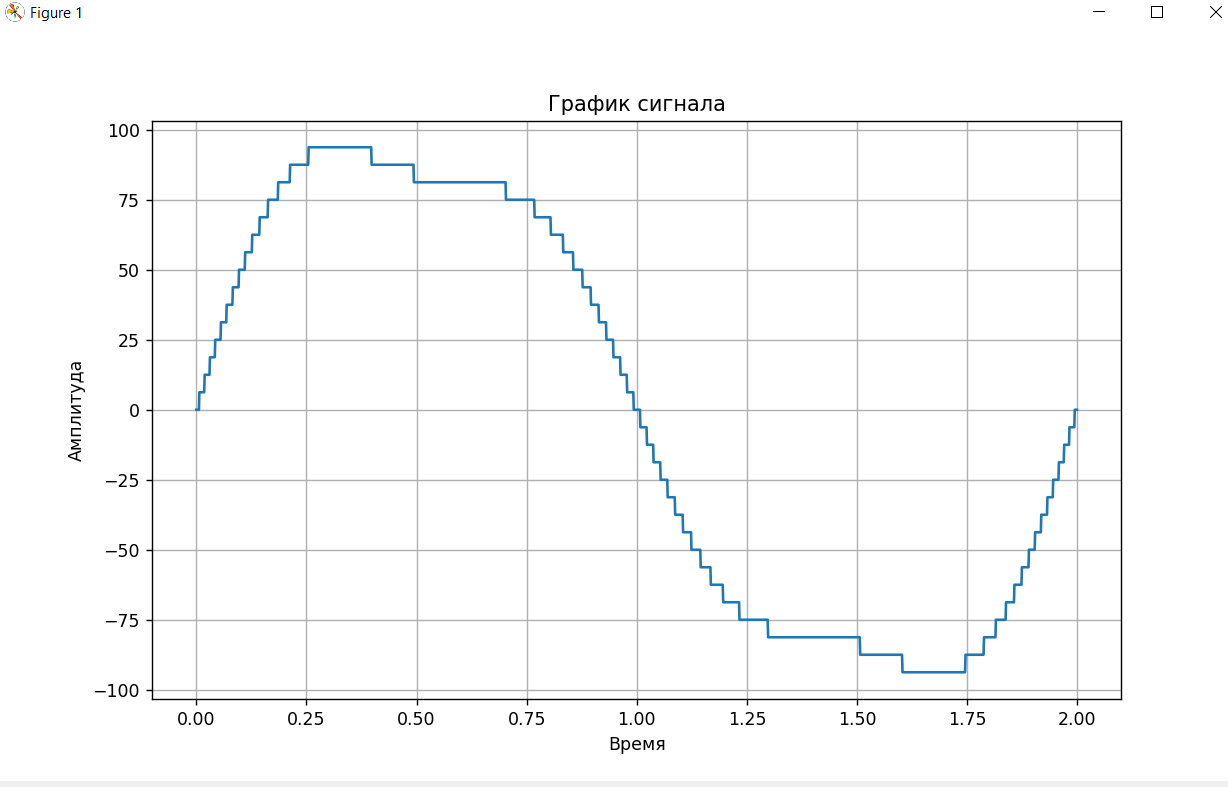
plt.xlabel('Время')

plt.ylabel('Амплитуда')

plt.title('График сигнала')

plt.grid(True)

plt.show()



**Сравнение сигналов**

import matplotlib.pyplot as plt

import numpy as np

data = np.loadtxt('overlapped\_signal.txt')

plt.plot(data[:, 0], data[:, 1], label='Исходный сигнал')

plt.plot(data[:, 0], data[:, 2], label='Квантованный сигнал')

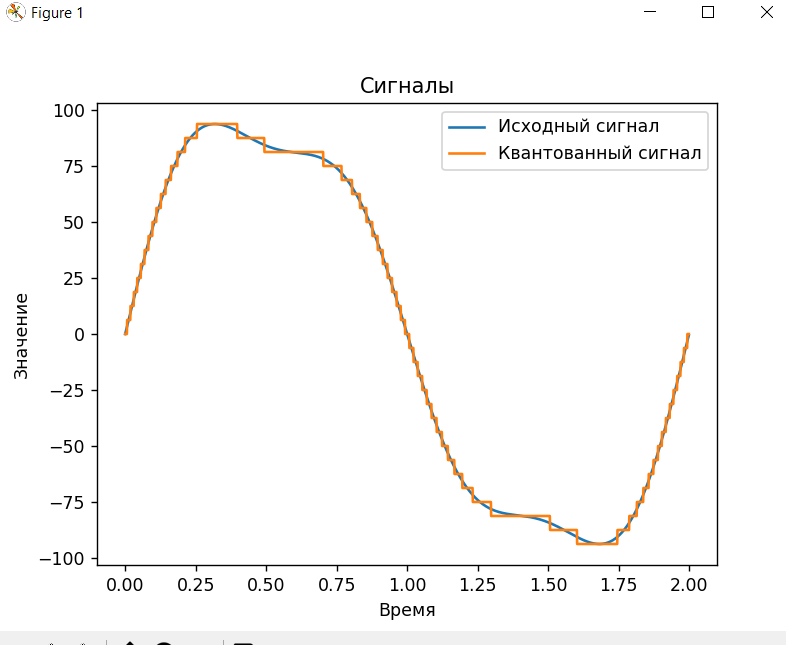
plt.legend()

plt.xlabel('Время')

plt.ylabel('Значение')

plt.title('Сигналы')

plt.show()



## **Заключение**

В ходе лабораторной работы было выполнено проектирование дискретного сигнала, его визуализация, а также проектирование и визуализация квантованного сигнала, чем больше уровней квантования тем ближе к исходному сигналу получится квантованный, так как он становится более плавным на графике.