Отчет о проекте на тему: «Алгоритм и программная реализация одномерной линейной интерполяции сеточной функции кубическими сплайнами»

# Введение

В работе выполняется проект на тему «Алгоритм и программная реализация одномерной линейной интерполяции сеточной функции кубическими сплайнами».

# Метод

Пусть на сетке заданы значения функции :

Требуется построить функцию , совпадающую с функцией в узлах сетки:

Интерполяция кубическими сплайнами является частным случаем кусочно-полиномиальной интерполяции. В этом специальном случае между любыми двумя соседними узлами функция интерполируется кубическим полиномом. Его коэффициенты на каждом интервале определяются из условий:

На границах при и ставятся условия:

Кубический полином ищется в виде

Из условия получается:

Первая и вторая производные равны:

Из условий и получаются равенства:

Из краевых условий и следуют равенства:

Из приведенных выше равенств получаются выражения для коэффициентов , , :

Для коэффициентов получается система уравнений:

Эту систему уравнений можно решить методом прогонки, который основан на предположении, что искомые неизвестные связаны рекуррентным соотношением:

Подстановкой в систему уравнений для выражений для получаются выражения для и :

В этих формулах принято обозначение:

Из условия следует , .

# Алгоритм

Алгоритм решения задачи интерполяции сеточной функции кубическими сплайнами выглядит следующим образом.

1. По формулам (8) и (9) вычисляются коэффициенты , с учетом, что , .
2. По формуле (7) вычисляются коэффициенты .
3. По формулам (2-6) вычисляются оставшиеся коэффициенты , , .

# Тестирование

Для решения поставленной задачи об интерполяции сеточной функции кубическими сплайнами была разработана программа с графическим интерфейсом. Программа написана на языке программирования C++ с применением библиотеки Qt5. Исходный код программы приведен в Приложении.

На рис. 1 показано изображение главного окна приложения.

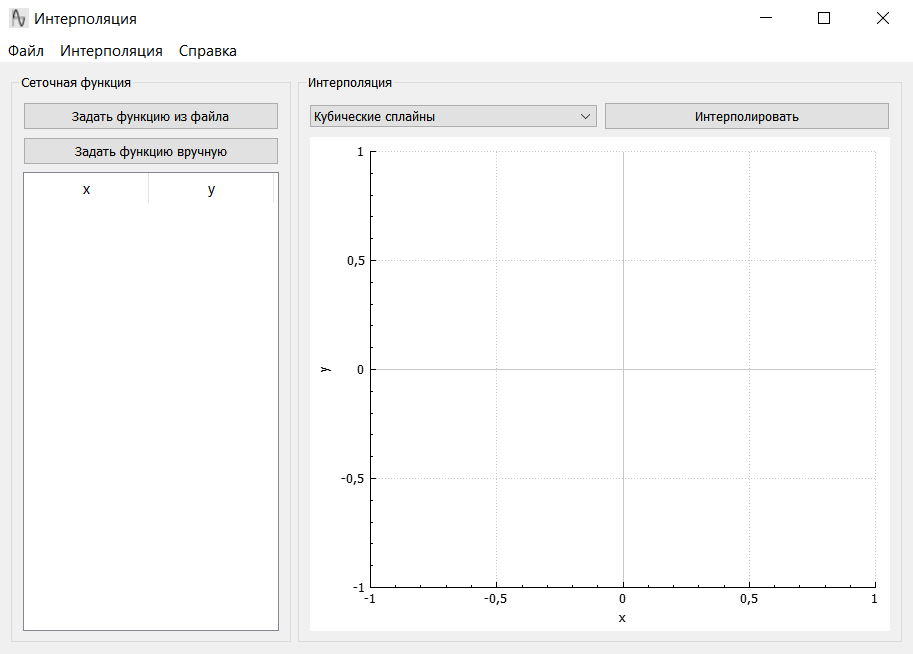


Рис. 1

В приложении предусмотрено два способа задать сеточную функцию:

1. С помощью заранее записанного текстового файла.
2. Вручную записав значения сеточной функции в таблицу.

Чтобы задать сеточную функцию с помощью текстового файла, нужно записать файл, в котором на каждой строчке через пробел записаны координата узла и значение функции в узле. Например, на рис. 2 показано изображение файла data.txt с сеточной функцией.

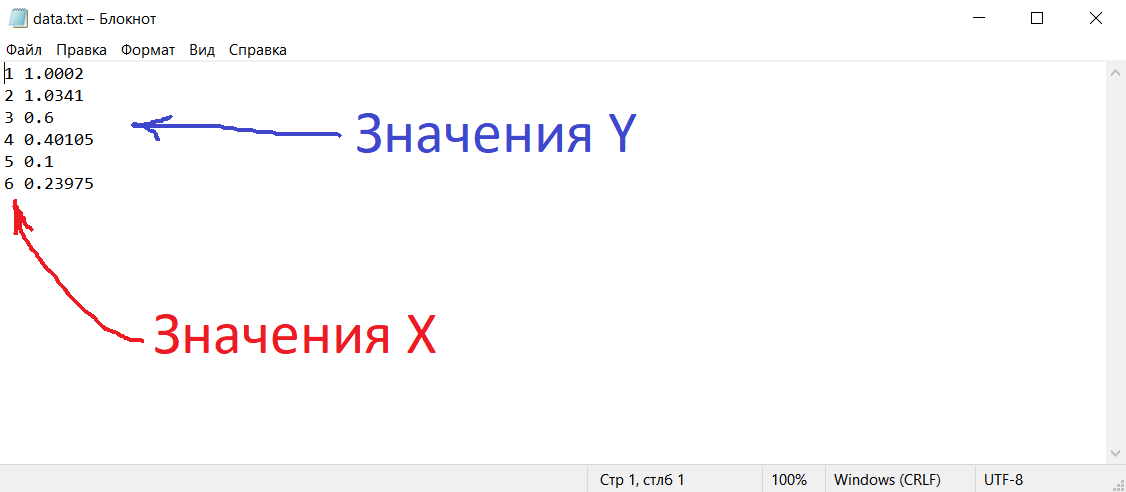
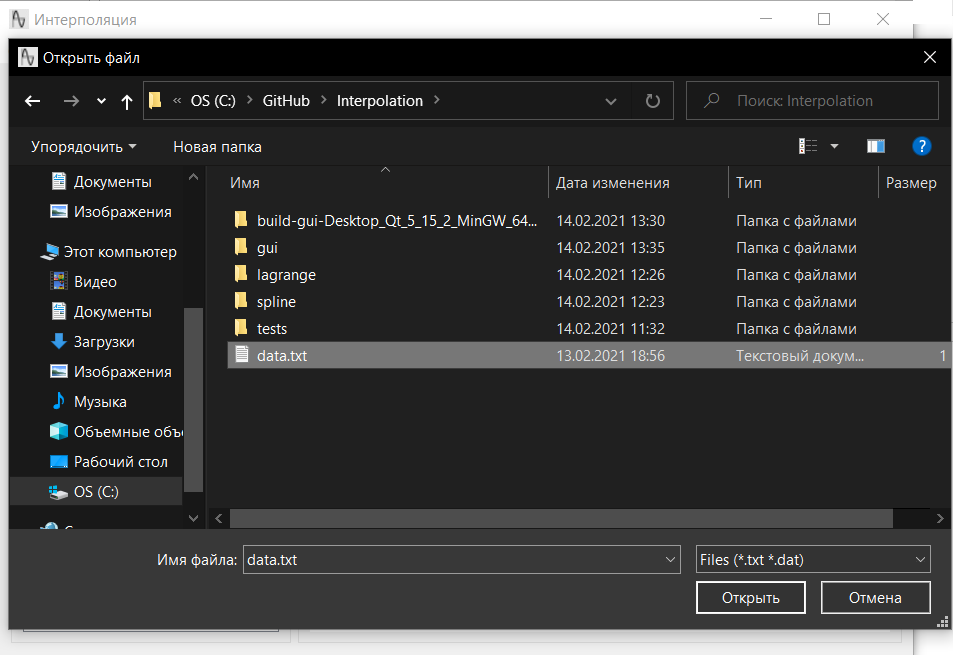


Рис. 2

На рис. 3 показано изображение окна приложения при выборе файла с сеточной функцией.



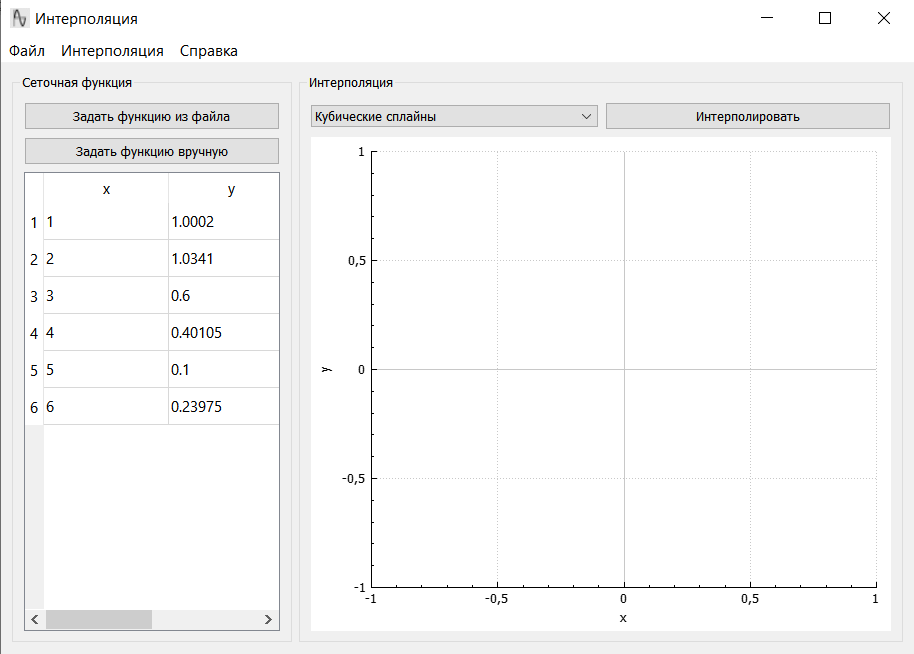


Рис. 3

После этого нужно кликнуть по кнопке «Интерполировать», и в окне появится график функции, интерполированной кубическими сплайнами (рис. 4).

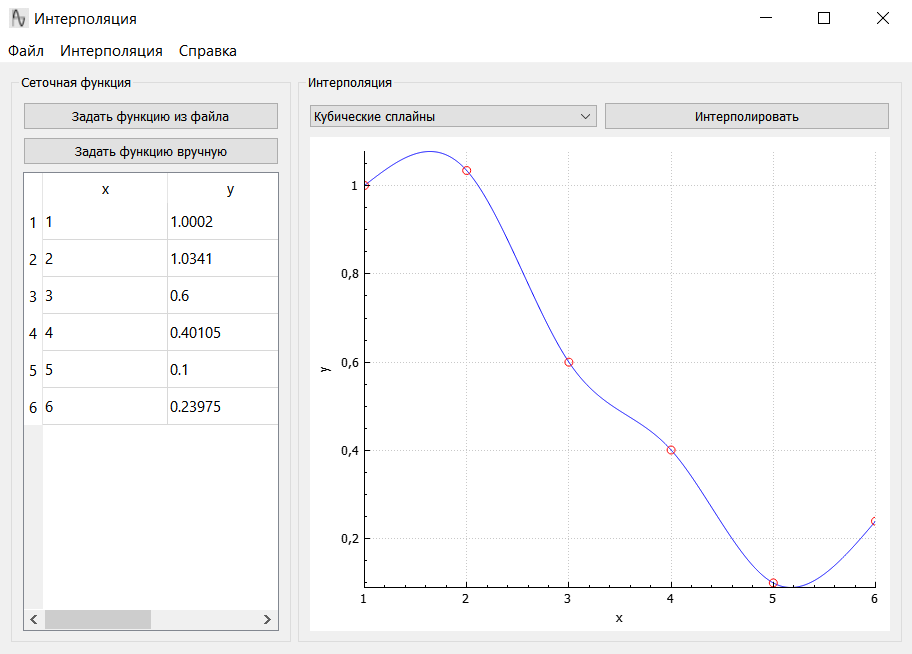
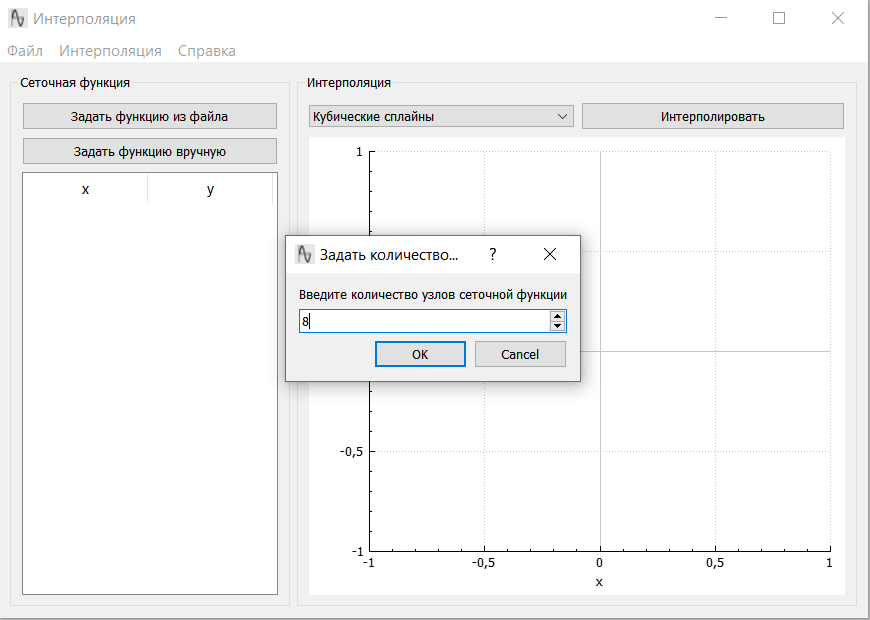


Рис. 4

Как уже упоминалось, в приложении имеется возможность задать сеточную функцию вручную. При клике по кнопке «Задать функцию вручную» появится диалоговое окно, в котором нужно ввести количество узлов сеточной функции (рис. 5). После этого нужно ввести координаты узлов и значения функции в этих узлах (рис. 5).



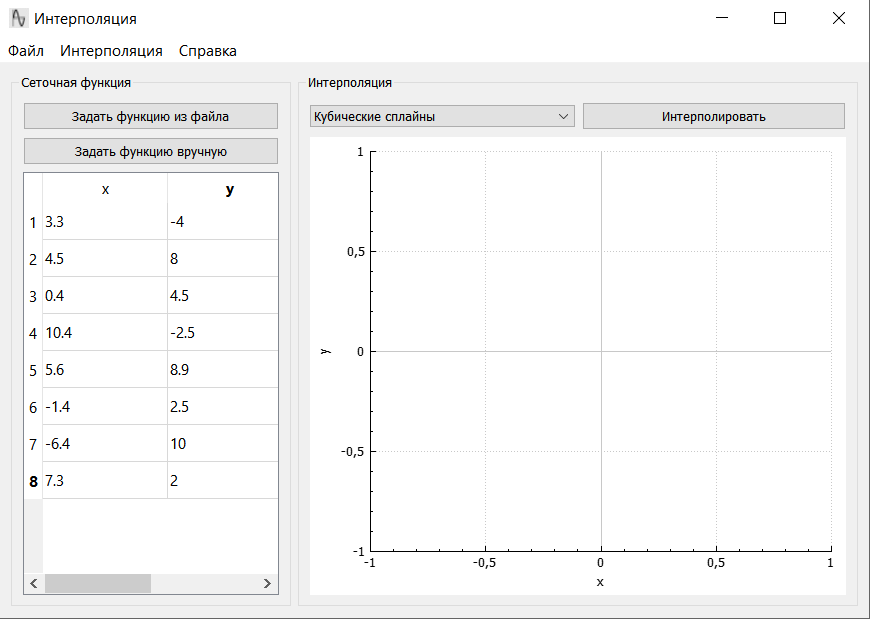


Рис. 5

На рис. 6 показан график интерполированной функции.

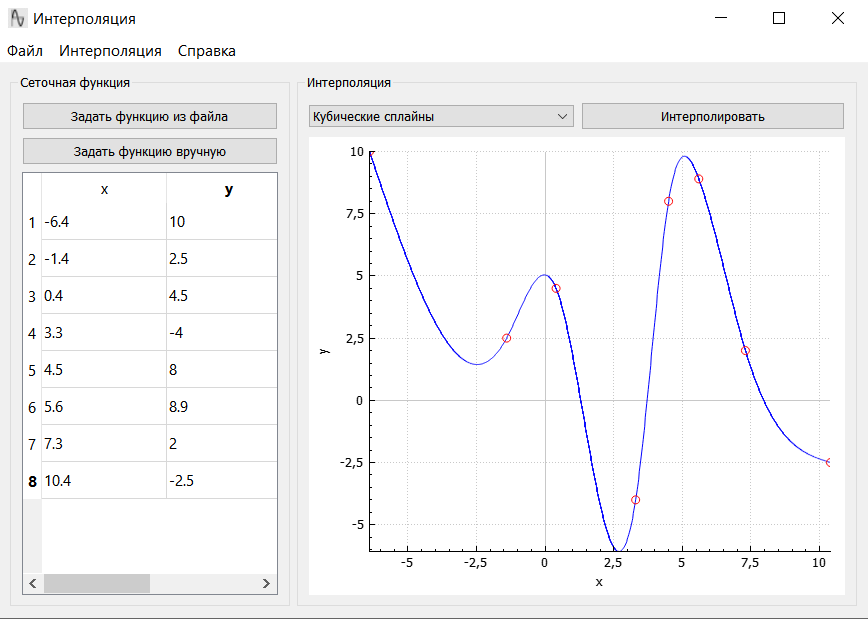


Рис. 6

# Список литературы

1. Интерполяция кубическими сплайнами // [http://www.machinelearning.ru/wiki/index.php?title=Интерполяция\_кубическими\_сплайнами](http://www.machinelearning.ru/wiki/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%D0%BA%D1%83%D0%B1%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%BC%D0%B8_%D1%81%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD%D0%B0%D0%BC%D0%B8)

# Приложение

Листинг файла spline.h.

/\*

Заголовочный файл содержит объявление класса Spline для осуществления

интерполяции одномерной сеточной функции кубическими сплайнами.

\*/

#pragma once

#ifndef SPLINE\_H

#define SPLINE\_H

#include <vector>

/\*\*

\* Класс для интерполяции сеточной функции кубическими сплайнами.

\*/

class Spline

{

public:

// Конструктор по умолчанию

Spline();

// Конструктор копирования

Spline(Spline&);

// Конструктор инициализации

Spline(unsigned int, double\*, double\*);

// Конструктор инициализации

Spline(std::vector<double>&, std::vector<double>&);

// Деструктор

~Spline();

// Метод вычисляет значение функции в точке

double calculate(double);

// Перегрузка оператора присваивания

Spline& operator = (const Spline&);

private:

unsigned int n = 0; // количество узлов сеточной функции

double\* x = nullptr; // массив координат узлов

double\* y = nullptr; // массив значений сеточной функции в узлах

// Коэффициенты интерполяции кубическими сплайнами

double\* a = nullptr;

double\* b = nullptr;

double\* c = nullptr;

double\* d = nullptr;

// Метод удаляет динамические массивы

void delete\_arrays(unsigned int, ...);

// Метод находит индекс наименьшего из двух узлов, между которыми попадает

// координата точки

unsigned int find\_index(double);

// Метод инициализирует сеточную функцию, для которой будет применена

// интерполяция сплайнами

void init(unsigned int, double\*, double\*);

// Метод инициализирует сеточную функцию, для которой будет применена

// интерполяция сплайнами

void init(std::vector<double>&, std::vector<double>&);

// Метод вычисляет коэффициенты для интерполяции сплайнами

void init\_spline();

// Метод инициализирует коэффициенты для интерполяции сплайнами

void init\_spline(double\*, double\*, double\*, double\*);

// Метод вычисляет в обратном ходе коэффициенты c кубических сплайнов

void run\_reverse(double\*, double\*);

// Метод вычисляет в прямом ходе коэффициенты eta, xi

void run\_straight(double\*\*, double\*\*);

};

#endif // !SPLINE\_H

Листинг файла spline.cpp.

/\*

Модуль содержит определение методов класса Spline.

\*/

#include <iostream>

#include <stdarg.h>

#include "spline.h"

/\*\*

\* Конструктор по умолчанию.

\*/

Spline::Spline() {}

/\*\*

\* Конструктор копирования.

\* @param s: копируемый объект.

\*/

Spline::Spline(Spline& s)

{

// Для интерполяции кубическими сплайнами необходимо как минимум 2 узла

if (s.n < 2)

return;

// Инициализируем сеточную функцию

init(s.n, s.x, s.y);

// Инициализируем кубические сплайны

init\_spline(s.a, s.b, s.c, s.d);

}

/\*\*

\* Конструктор инициализации.

\* @param n: количество узлов, в которых определена сеточная функция;

\* @param x: массив координат узлов сеточной функции;

\* @param y: массив значений сеточной функции.

\*/

Spline::Spline(unsigned int n, double\* x, double\* y)

{

// Для интерполяции кубическими сплайнами необходимо как минимум 2 узла

if (n < 2)

return;

// Инициализируем сеточную функцию

init(n, x, y);

// Вычисляем коэффициенты кубических сплайнов

init\_spline();

}

/\*\*

\* Конструктор инициализации.

\* @param x: массив координат узлов сеточной функции;

\* @param y: массив значений сеточной функции.

\*/

Spline::Spline(std::vector<double>& x, std::vector<double>& y)

{

// Для интерполяции кубическими сплайнами необходимо как минимум 2 узла

if (x.size() < 2)

return;

// Инициализируем сеточную функцию

init(x, y);

// Вычисляем коэффициенты кубических сплайнов

init\_spline();

}

/\*\*

\* Деструктор.

\*/

Spline::~Spline()

{

// Очищаем память, выделенную на динамические массивы со значениями

// координат узлов, сеточной функции, коэффициентами кубических сплайнов

delete\_arrays(6, &x, &y, &a, &b, &c, &d);

}

/\*\*

\* Метод вычисляет значение функции в точке.

\* @param x: координата точки.

\* @return: значение интерполированной функции.

\*/

double Spline::calculate(double x)

{

if (n < 2)

return 0;

// Определяем индекс наименьшего из двух узлов, между которыми попадает

// координата x

unsigned int i = find\_index(x);

// Вычисляем значение интерполяции

double dx = x - this->x[i];

return a[i + 1] + dx \* (b[i + 1] + dx \* (c[i + 1] + dx \* d[i + 1]));

}

/\*\*

\* Метод удаляет динамические массивы.

\* @param n: количество удаляемых динамических массивов.

\*/

void Spline::delete\_arrays(unsigned int n, ...)

{

va\_list factor; // указатель на необязательный параметр

va\_start(factor, n); // устанавливаем указатель

for (unsigned int i = 0; i < n; i++)

{

double\*\* array = va\_arg(factor, double\*\*);

if (\*array != nullptr)

delete[] \* array;

}

va\_end(factor);

}

/\*\*

\* Метод находит индекс наименьшего из двух узлов, между которыми попадает

\* координата точки.

\* @param x: координата точки.

\* @return: индекс наименьшего из двух соседних узлов, между которыми попадает

\* точка.

\*/

unsigned int Spline::find\_index(double x)

{

if (x <= this->x[0])

return 0;

if (x >= this->x[n - 1])

return n - 2;

for (unsigned int i = 0; i < n - 1; i++)

{

if (this->x[i] <= x && x <= this->x[i + 1])

return i;

}

return 0;

}

/\*\*

\* Метод инициализирует сеточную функцию, для которой будет применена

\* интерполяция сплайнами.

\* @param n: количество узлов, в которых определена сеточная функция;

\* @param x: массив координат узлов сеточной функции;

\* @param y: массив значений сеточной функции.

\*/

void Spline::init(unsigned int n, double\* x, double\* y)

{

// Удаляем память, выделенную на динамические массивы

delete\_arrays(2, &this->x, &this->y);

// Записываются координаты узлов и значения сеточной функции в узлах

this->n = n;

this->x = new double[this->n];

this->y = new double[this->n];

for (unsigned int i = 0; i < n; i++)

{

this->x[i] = x[i];

this->y[i] = y[i];

}

}

/\*\*

\* Метод инициализирует сеточную функцию, для которой будет применена

\* интерполяция сплайнами.

\* @param x: массив координат узлов сеточной функции;

\* @param y: массив значений сеточной функции.

\*/

void Spline::init(std::vector<double>& x, std::vector<double>& y)

{

// Удаляем память, выделенную на динамические массивы

delete\_arrays(2, &this->x, &this->y);

// Записываются координаты узлов и значения сеточной функции в узлах

this->n = x.size();

this->x = new double[this->n];

this->y = new double[this->n];

for (unsigned int i = 0; i < n; i++)

{

this->x[i] = x[i];

this->y[i] = y[i];

}

}

/\*\*

\* Метод вычисляет коэффициенты для интерполяции сплайнами.

\*/

void Spline::init\_spline()

{

// Прямой ход для вычисления коэффициентов eta, xi

double\* eta = new double[n + 1];

double\* xi = new double[n + 1];

run\_straight(&eta, &xi);

// Выделяем память на массив с коэффициентами кубических сплайнов

a = new double[n];

b = new double[n];

c = new double[n];

d = new double[n];

// Обратный ход для вычисления коэффициентов кубических сплайнов

run\_reverse(eta, xi);

// Удаляем выделенную для eta и xi память

delete\_arrays(2, &eta, &xi);

}

// Метод инициализирует коэффициенты для интерполяции сплайнами

void Spline::init\_spline(double\* a, double\* b, double\* c, double\* d)

{

// Удаляем память, выделенную на динамические массивы

delete\_arrays(4, &this->a, &this->b, &this->c, &this->d);

// Записываются коэффициенты кубических сплайнов

this->a = new double[n];

this->b = new double[n];

this->c = new double[n];

this->d = new double[n];

for (unsigned int i = 0; i < n; i++)

{

this->a[i] = a[i];

this->b[i] = b[i];

this->c[i] = c[i];

this->d[i] = d[i];

}

}

/\*\*

\* Метод вычисляет в обратном ходе коэффициенты кубических сплайнов.

\* @param eta, xi: массивы для коэффициентов eta, xi.

\*/

void Spline::run\_reverse(double\* eta, double\* xi)

{

double h = x[n - 1] - x[n - 2];

a[n - 1] = y[n - 2];

c[n - 1] = eta[n];

b[n - 1] = (y[n - 1] - y[n - 2]) / h - 2 \* h \* c[n - 1] / 3;

d[n - 1] = -c[n - 1] / 3 / h;

for (unsigned int i = n - 2; i > 0; i--)

{

h = x[i] - x[i - 1];

a[i] = y[i - 1];

c[i] = xi[i + 1] \* c[i + 1] + eta[i + 1];

b[i] = (y[i] - y[i - 1]) / h - h \* (c[i + 1] + 2 \* c[i]) / 3;

d[i] = (c[i + 1] - c[i]) / 3 / h;

}

}

/\*\*

\* Метод вычисляет в прямом ходе коэффициенты eta, xi.

\* @param eta, xi: массивы для коэффициентов eta, xi.

\*/

void Spline::run\_straight(double\*\* eta, double\*\* xi)

{

(\*eta)[2] = 0;

(\*xi)[2] = 0;

for (unsigned int i = 2; i < n; i++)

{

double f = 3 \* ((y[i] - y[i - 1]) / (x[i] - x[i - 1]) -

(y[i - 1] - y[i - 2]) / (x[i - 1] - x[i - 2]));

double d = (x[i - 1] - x[i - 2]) \* (\*xi)[i] + 2 \* (x[i] - x[i - 2]);

(\*eta)[i + 1] = (f - (x[i - 1] - x[i - 2]) \* (\*eta)[i]) / d;

(\*xi)[i + 1] = (x[i - 1] - x[i]) / d;

}

}

/\*\*

\* Перегрузка оператора присваивания.

\*/

Spline& Spline::operator = (const Spline& s)

{

// Проверка на самоприсваивание

if (this == &s)

return \*this;

// Удаляем память, выделенную на динамические массивы

delete\_arrays(6, &this->x, &this->y, &this->a, &this->b, &this->c, &this->d);

// Инициализируем сеточную функцию

init(s.n, s.x, s.y);

// Инициализируем кубические сплайны

init\_spline(s.a, s.b, s.c, s.d);

return \*this;

}

Листинг файла functions.h.

/\*

\* Заголовочный файл с прототипами полезных функций.

\*/

#pragma once

#ifndef FUNCTIONS\_H

#define FUNCTIONS\_H

#include <vector>

// Функция разбиения массива для быстрой сортировки

int partition(std::vector<double>&, std::vector<double>&, int, int);

// Функция сортирует два вектора по возрастанию элементов одного из

// векторов

void sort\_quick(std::vector<double>&, std::vector<double>&, int, int);

// Функция меняет местами два элемента массива

void swap(std::vector<double>&, int, int);

#endif // FUNCTIONS\_H

Листинг файла functions.cpp.

/\*

\* Модуль содержит определение полезных функций.

\*/

#include "functions.h"

/\*\*

\* Функция разбиения массива для быстрой сортировки.

\* @param a: массив, который должен быть отсортирован по возрастанию;

\* @param b: массив, который будет отсортирован за компанию;

\* @param left, right: индексы, элементы между которыми нужно отсортировать.

\*/

int partition(std::vector<double>& a, std::vector<double>& b, int left,

int right)

{

// Выбираем средний элемент в качестве опорного

double pivot = a[(left + right) / 2];

int i = left - 1;

int j = right + 1;

while (true)

{

i++;

while (a[i] < pivot)

i++;

j--;

while (a[j] > pivot)

j--;

if (i >= j)

return j;

// Если элемент с индексом i (слева от опорного) больше, чем элемент с

// индексом j (справа от опорного), меняем их местами

swap(a, i, j);

swap(b, i, j);

}

}

/\*\*

\* Функция сортирует два вектора по возрастанию элементов одного из векторов.

\* @param a: массив, который должен быть отсортирован по возрастанию;

\* @param b: массив, который будет отсортирован за компанию;

\* @param left, right: индексы, элементы между которыми нужно отсортировать.

\*/

void sort\_quick(std::vector<double>& a, std::vector<double>& b, int left,

int right)

{

if (left < right)

{

int split\_index = partition(a, b, left, right);

sort\_quick(a, b, left, split\_index);

sort\_quick(a, b, split\_index + 1, right);

}

}

/\*\*

\* Функция меняет местами два элемента массива.

\* @param a: массив;

\* @param i, j: индексы элементов, которые нужно поменять местами.

\*/

void swap(std::vector<double>& a, int i, int j)

{

double tmp = a[i];

a[i] = a[j];

a[j] = tmp;

}

Листинг файла lagrange.h.

/\*

Заголовочный файл с объявлением класса Lagrange для интерполяции сеточной

функции полиномами Лагранжа.

\*/

#pragma once

#ifndef LAGRANGE\_H

#define LAGRANGE\_H

#include <vector>

/\*\*

\* Класс для интерполяции сеточной функции полиномами Лагранжа.

\*/

class Lagrange

{

public:

// Конструктор по умолчанию

Lagrange();

// Конструктор копирования

Lagrange(Lagrange&);

// Конструктор инициализации

Lagrange(const std::vector<double>&, const std::vector<double>&);

// Деструктор

~Lagrange();

// Метод вычисляет значение функции в точке

double calculate(double);

// Перегрузка оператора присваивания

Lagrange& operator = (const Lagrange&);

private:

std::vector<double> x; // массив координат узлов

std::vector<double> y; // массив значений сеточной функции в узлах

// Метод инициализирует сеточную функцию, для которой будет применена

// интерполяция полиномами Лагранжа

void init(const std::vector<double>&, const std::vector<double>&);

};

#endif // !LAGRANGE\_H

Листинг файла lagrange.cpp.

/\*

Модуль содержит определение методов класса Lagrange.

\*/

#include "lagrange.h"

/\*\*

\* Конструктор по умолчанию.

\*/

Lagrange::Lagrange() {}

/\*\*

\* Конструктор копирования.

\* @param l: копируемый объект.

\*/

Lagrange::Lagrange(Lagrange& l)

{

// Для интерполяции полиномами Лагранжа необходимо как минимум 2 узла

if (l.x.size() < 2)

return;

// Инициализируем сеточную функцию

init(l.x, l.y);

}

/\*\*

\* Конструктор инициализации.

\* @param x: массив координат узлов сеточной функции;

\* @param y: массив значений сеточной функции.

\*/

Lagrange::Lagrange(const std::vector<double>& x, const std::vector<double>& y)

{

// Для интерполяции кубическими сплайнами необходимо как минимум 2 узла

if (x.size() < 2)

return;

// Инициализируем сеточную функцию

init(x, y);

}

/\*\*

\* Деструктор.

\*/

Lagrange::~Lagrange() {}

/\*\*

\* Метод вычисляет значение функции в точке.

\* @param x: координата точки.

\* @return: значение интерполированной функции.

\*/

double Lagrange::calculate(double x)

{

double y = 0;

for (unsigned int i = 0; i < this->x.size(); i++)

{

double l\_i = 1;

for (unsigned int j = 0; j < this->x.size(); j++)

{

if (i == j)

continue;

l\_i \*= (x - this->x[j]) / (this->x[i] - this->x[j]);

}

y += l\_i \* this->y[i];

}

return y;

}

/\*\*

\* Метод инициализирует сеточную функцию, для которой будет применена

\* интерполяция полиномами Лагранжа.

\* @param x: массив координат узлов сеточной функции;

\* @param y: массив значений сеточной функции.

\*/

void Lagrange::init(const std::vector<double>& x, const std::vector<double>& y)

{

// Очищаем массивы

this->x.clear();

this->y.clear();

// Записываются координаты узлов и значения сеточной функции в узлах

this->x = x;

this->y = y;

}

/\*\*

\* Перегрузка оператора присваивания.

\*/

Lagrange& Lagrange::operator = (const Lagrange& l)

{

// Проверка на самоприсваивание

if (this == &l)

return \*this;

// Инициализируем сеточную функцию

init(l.x, l.y);

return \*this;

}

Листинг файла mainwindow.h.

/\*

\* Заголовочный файл с классом для главного окна приложения.

\*/

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QAction>

#include <QComboBox>

#include <QMainWindow>

#include <QTableWidget>

#include <QVector>

#include "qcustomplot.h"

#include <vector>

// Постоянные

const QString LAGRANGE = "Полиномы Лагранжа";

const QString SPLINE = "Кубические сплайны";

/\*\*

\* Класс для главного окна приложения.

\*/

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

// Конструктор

MainWindow(QWidget\* parent = nullptr);

// Деструктор

~MainWindow();

private:

const QString ICON = "icon.png"; // путь к иконке

// Пункты меню интерполяции полиномами Лагранжа и сплайнами

QAction\* menu\_lagrange;

QAction\* menu\_spline;

QComboBox\* method; // выпадающий список с методами интерполяции

QCustomPlot\* plot; // область для графика функции

QTableWidget\* tbl; // таблица для сеточной функции

std::vector<double> x; // массив координат узлов

std::vector<double> y; // массив значений функции в узлах

// Метод конвертирует векторы

void convert\_vector(std::vector<double>&, QVector<double>&);

// Метод располагает виджеты на главном окне

void create\_main\_wnd();

// Метод создает меню

void create\_menu();

// Метод собирает значения сеточной функции из таблицы

bool get\_grid\_function();

// Метод для интерполяции полиномами Лагранжа

static void interpolate\_lagrange(

unsigned int, std::vector<double>&, std::vector<double>&,

QVector<double>&, QVector<double>&);

// Метод для интерполяции кубическими сплайнами

static void interpolate\_spline(

unsigned int, std::vector<double>&, std::vector<double>&,

QVector<double>&, QVector<double>&);

// Метод выводит сеточную функцию в таблицу

void show\_grid\_function(int, bool have\_values = false);

// Метод рисует график интерполированной функции

void show\_plot(QVector<double>&, QVector<double>&);

public slots:

// Слот для интерполяции сеточной функции

void interpolate();

// Слот для чтения данных о сеточной функции из файла

void set\_grid\_function\_from\_file();

// Слот для задания данных о сеточной функции вручную

void set\_grid\_function\_manually();

// Слот выводит информацию о приложении

void show\_info();

};

#endif // MAINWINDOW\_H

Листинг файла mainwindow.cpp.

/\*

\* Модуль содержит определения методов класса главного окна.

\*/

#include <QApplication>

#include <QComboBox>

#include <QFile>

#include <QFileDialog>

#include <QGroupBox>

#include <QInputDialog>

#include <QMenu>

#include <QMenuBar>

#include <QPushButton>

#include <QTextStream>

#include <QValidator>

#include <QVBoxLayout>

#include "functions.h"

#include "mainwindow.h"

#include "../lagrange/lagrange.h"

#include "../spline/spline.h"

/\*\*

\* Конструктор.

\*/

MainWindow::MainWindow(QWidget\* parent) : QMainWindow(parent)

{

// Задаем иконку и название главного окна

setWindowIcon(QIcon(ICON));

setWindowTitle("Интерполяция");

// Создаем меню

create\_menu();

// Располагаем виджеты на главном окне

create\_main\_wnd();

}

/\*\*

\* Деструктор.

\*/

MainWindow::~MainWindow() {}

/\*\*

\* Метод конвертирует векторы.

\* @param x: вектор с существующими данными;

\* @param x\_new: новый вектор, в который будут записаны данные.

\*/

void MainWindow::convert\_vector(std::vector<double>& x, QVector<double>& x\_new)

{

for (auto i : x)

x\_new.push\_back(i);

}

/\*\*

\* Метод располагает виджеты на главном окне.

\*/

void MainWindow::create\_main\_wnd()

{

// Область для ввода сеточной функции.

// Кнопка для открытия файла с данными сеточной функции

QVBoxLayout\* vbox = new QVBoxLayout();

QPushButton\* btn = new QPushButton("Задать функцию из файла", this);

connect(btn, &QPushButton::clicked, this,

&MainWindow::set\_grid\_function\_from\_file);

vbox->addWidget(btn);

// Кнопка для задания сеточной функции вручную

btn = new QPushButton("Задать функцию вручную", this);

connect(btn, &QPushButton::clicked, this,

&MainWindow::set\_grid\_function\_manually);

vbox->addWidget(btn);

// Таблица сеточной функции

tbl = new QTableWidget(0, 2, this);

tbl->setHorizontalHeaderLabels({ "x", "y" });

vbox->addWidget(tbl, 1);

QGroupBox\* grid\_fnc\_box = new QGroupBox("Сеточная функция");

grid\_fnc\_box->setLayout(vbox);

// Область для интерполяции функции

QHBoxLayout\* hbox\_inter = new QHBoxLayout();

// Список доступных типов интерполяции

method = new QComboBox();

QStringList types = { SPLINE, LAGRANGE };

method->addItems(types);

hbox\_inter->addWidget(method);

// Кнопка для запуска интерполяции

btn = new QPushButton("Интерполировать", this);

connect(btn, &QPushButton::clicked, this, &MainWindow::interpolate);

hbox\_inter->addWidget(btn);

vbox = new QVBoxLayout();

vbox->addLayout(hbox\_inter);

// Область для графика

plot = new QCustomPlot();

plot->xAxis->setLabel("x");

plot->yAxis->setLabel("y");

plot->xAxis->setRange(-1, 1);

plot->yAxis->setRange(-1, 1);

plot->replot();

vbox->addWidget(plot, 1);

QGroupBox\* interpolation\_box = new QGroupBox("Интерполяция");

interpolation\_box->setLayout(vbox);

QHBoxLayout\* hbox = new QHBoxLayout();

hbox->addWidget(grid\_fnc\_box);

hbox->addWidget(interpolation\_box, 1);

// Размещаем по центру окна

QWidget\* main\_widget = new QWidget();

main\_widget->setLayout(hbox);

setCentralWidget(main\_widget);

}

/\*\*

\* Метод создает меню на главном окне.

\*/

void MainWindow::create\_menu()

{

QMenu\* menu;

// Меню с чтением данных

menu = menuBar()->addMenu("Файл");

// Пунтк меню для чтения файла с данными

QAction\* file = new QAction("Открыть файл", this);

connect(file, &QAction::triggered, this,

&MainWindow::set\_grid\_function\_from\_file);

menu->addAction(file);

// Пункт меню для выхода из приложения

QAction\* quit = new QAction("Выход", this);

connect(quit, &QAction::triggered, qApp, &QApplication::quit);

menu->addSeparator(); // устанавливаем разделитель

menu->addAction(quit); // добавляем действие "Выход"

// Меню с типами интерполяции

menu = menuBar()->addMenu("Интерполяция");

// Пункт меню с интерполяцией кубическими сплайнами

menu\_spline = new QAction(SPLINE, this);

connect(menu\_spline, &QAction::triggered, this, &MainWindow::interpolate);

menu->addAction(menu\_spline);

// Пункт меню с интерполяцией полиномами Лагранжа

menu\_lagrange = new QAction(LAGRANGE, this);

connect(menu\_lagrange, &QAction::triggered, this,

&MainWindow::interpolate);

menu->addAction(menu\_lagrange);

// Меню с информацией о приложении

menu = menuBar()->addMenu("Справка");

QAction\* info = new QAction("О приложении", this);

connect(info, &QAction::triggered, this, &MainWindow::show\_info);

menu->addAction(info);

}

/\*\*

\* Метод собирает значения сеточной функции из таблицы.

\* @return: true, если для полученной сеточной функции можно выполнить

\* интерполяцию, иначе false.

\*/

bool MainWindow::get\_grid\_function()

{

// Очищаем списки с координатами узлов и значениями сеточной функции

x.clear();

y.clear();

// Пробегаем по строкам таблицы

int row = 0;

while (row < tbl->rowCount())

{

double values[2];

try

{

for (int column = 0; column < 2; column++)

{

QLineEdit\* cell = (QLineEdit\*)tbl->cellWidget(row, column);

if (cell->text() == "" || cell->text() == "-")

throw - 1;

values[column] = cell->text().toDouble();

}

}

catch (...)

{

tbl->removeRow(row);

continue;

}

x.push\_back(values[0]);

y.push\_back(values[1]);

row++;

}

return x.size() > 1;

}

/\*\*

\* Слот для интерполяции сеточной функции.

\*/

void MainWindow::interpolate()

{

// Получаем значения сеточной функции

if (!get\_grid\_function())

{

// Сеточная функция не подходит для интерполяции

QMessageBox msgBox;

msgBox.setWindowIcon(QIcon(ICON));

msgBox.setWindowTitle("Ошибка");

msgBox.setText("Интерполяция невозможна.");

msgBox.exec();

return;

}

// Сортируем узлы по возрастанию

sort\_quick(x, y, 0, x.size() - 1);

// Выводим отсортированную сеточную функцию в таблицу

show\_grid\_function(x.size(), true);

// Определяем тип интерполяции

void (\*interpolation)(

unsigned int, std::vector<double>&, std::vector<double>&,

QVector<double>&, QVector<double>&) = nullptr;

if (sender() != menu\_lagrange && sender() != menu\_spline)

{

// Была нажата кнопка 'Интерполировать'

QString interpolation\_type = method->currentText();

if (interpolation\_type == SPLINE)

interpolation = interpolate\_spline;

else if (interpolation\_type == LAGRANGE)

interpolation = interpolate\_lagrange;

}

else

{

// Был выбран один из пунктов меню с типом интерполяции

if (sender() == menu\_spline)

interpolation = interpolate\_spline;

else if (sender() == menu\_lagrange)

interpolation = interpolate\_lagrange;

}

// Интерполируем сеточную функцию и вычисляем значения в новых точках

QVector<double> x\_new;

QVector<double> y\_new;

unsigned int N = 1000;

interpolation(N, x, y, x\_new, y\_new);

// Рисуем график интерполированной функции

show\_plot(x\_new, y\_new);

}

/\*\*

\* Метод для интерполяции сеточной функции полиномами Лагранжа.

\* @param n: количество точек, в которых нужно посчитать значения

\* интерполированной функции;

\* @param x, y: массивы с координатами узлов и значениями сеточной функции;

\* @param x\_new, y\_new: массивы, куда будут записаны координаты и значения

\* интерполированной функции.

\*/

void MainWindow::interpolate\_lagrange(

unsigned int n, std::vector<double>& x, std::vector<double>& y,

QVector<double>& x\_new, QVector<double>& y\_new)

{

Lagrange l(x, y);

double dx = (x[x.size() - 1] - x[0]) / (n - 1);

for (unsigned int i = 0; i < n; i++)

{

x\_new.push\_back(x[0] + dx \* i);

y\_new.push\_back(l.calculate(x\_new[i]));

}

}

/\*\*

\* Метод для интерполяции сеточной функции сплайнами.

\* @param n: количество точек, в которых нужно посчитать значения

\* интерполированной функции;

\* @param x, y: массивы с координатами узлов и значениями сеточной функции;

\* @param x\_new, y\_new: массивы, куда будут записаны координаты и значения

\* интерполированной функции.

\*/

void MainWindow::interpolate\_spline(

unsigned int n, std::vector<double>& x, std::vector<double>& y,

QVector<double>& x\_new, QVector<double>& y\_new)

{

Spline s(x, y);

double dx = (x[x.size() - 1] - x[0]) / (n - 1);

for (unsigned int i = 0; i < n; i++)

{

x\_new.push\_back(x[0] + dx \* i);

y\_new.push\_back(s.calculate(x\_new[i]));

}

}

/\*\*

\* Слот для чтения данных о сеточной функции из файла.

\*/

void MainWindow::set\_grid\_function\_from\_file()

{

// Открываем диалоговое окно для выбора файла

QString filename = QFileDialog::getOpenFileName(

this, "Открыть файл", "/", "Files (\*.txt \*.dat)");

if (filename == "")

return;

// Очищаем массивы с координатами узлов и значениями сеточной функции

x.clear();

y.clear();

// Читаем данные из файла

QFile file(filename);

if (!file.open(QIODevice::ReadOnly | QIODevice::Text))

return;

QTextStream in(&file);

while (!in.atEnd())

{

QString line = in.readLine();

QStringList xy = line.split(" ");

x.push\_back(xy[0].toDouble());

y.push\_back(xy[1].toDouble());

}

// Выводим сеточную функцию в таблицу

show\_grid\_function(x.size(), true);

}

/\*\*

\* Слот для задания данных о сеточной функции вручную.

\*/

void MainWindow::set\_grid\_function\_manually()

{

// Получаем количество узлов сеточной функции

bool ok;

int n = QInputDialog::getInt(

this, "Задать количество узлов",

"Введите количество узлов сеточной функции", 2, 2, 1000, 1, &ok);

if (!ok)

return;

// Выводим таблицу для сеточной функции

show\_grid\_function(n);

}

/\*\*

\* Метод выводит сеточную функцию в таблицу.

\* @param n: количество строк в таблице;

\* @param have\_values: true, если значения сеточной функции уже заданы.

\*/

void MainWindow::show\_grid\_function(int n, bool have\_values)

{

// Валидатор для ячеек таблицы

QRegExpValidator\* validator =

new QRegExpValidator(QRegExp("^(\\-)?[0-9]+[\\.]?[0-9]\*$"));

// Задаем количество строк в таблице

tbl->setRowCount(n);

// Записываем значения построчно

for (int row = 0; row < n; row++)

{

for (int column = 0; column < 2; column++)

{

QString item = "";

if (have\_values && column == 0)

item = tr("%1").arg(x[row]);

else if (have\_values && column == 1)

item = tr("%1").arg(y[row]);

// Устанавливаем значение и валидатор для ячейки

QLineEdit\* cell = new QLineEdit();

cell->setValidator(validator);

cell->setText(item);

cell->setFrame(false);

tbl->setCellWidget(row, column, cell);

}

}

}

/\*\*

\* Слот выводит информацию о приложении.

\*/

void MainWindow::show\_info()

{

QMessageBox msgBox;

msgBox.setWindowIcon(QIcon(ICON));

msgBox.setWindowTitle("Информация");

msgBox.setText("Приложение для интерполяции сеточных функций.");

msgBox.exec();

}

/\*\*

\* Метод рисует график интерполированной функции.

\* @param x\_new, y\_new: координаты x и значения интерполированной функции.

\*/

void MainWindow::show\_plot(QVector<double>& x\_new, QVector<double>& y\_new)

{

plot->clearGraphs();

// График интерполированной функции

plot->addGraph();

plot->graph(0)->setPen(QPen(Qt::blue));

plot->graph(0)->setData(x\_new, y\_new);

plot->graph(0)->rescaleAxes();

// График сеточной функции

QVector<double> x\_grid, y\_grid;

convert\_vector(x, x\_grid);

convert\_vector(y, y\_grid);

plot->addGraph();

plot->graph(1)->setPen(QColor(255, 0, 0, 255));

plot->graph(1)->setLineStyle(QCPGraph::lsNone);

plot->graph(1)->setScatterStyle(QCPScatterStyle(QCPScatterStyle::ssCircle, 8));

plot->graph(1)->setData(x\_grid, y\_grid);

plot->replot();

}

Листинг файла main.cpp.

/\*

\* Модуль для запуска приложения с графическим интерфейсом.

\*/

#include <QApplication>

#include "mainwindow.h"

int main(int argc, char\* argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}