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 $\omega = \{x_i : a = x_0 < x_1 < \cdots < x_n = b\}$ заданы значения функции $y(x)$:

$$y(x_0) = y_0, y(x_1) = y_1, \dots, y(x_n) = y_n.$$

Требуется построить функцию $f(x)$, совпадающую с функцией $y(x)$ в узлах сетки:

$$f(x_i) = y_i, i = 0,1, \dots, n.$$

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:

$$f_i = y_i,$$

$$f'(x_i - 0) = f'(x_i + 0),$$

$$f''(x_i - 0) = f''(x_i + 0), i = 1, 2, \dots, n - 1.$$

На границах при $x = x_0$ и $x = x_n$ ставятся условия:

$$f''(x_0) = 0, f''(x_n) = 0.$$

Кубический полином ищется в виде

$$f(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3, x_{i-1} \le x \le x_i.$$

Из условия $f_i = y_i$ получается:

$$f(x_{i-1}) = a_i = y_{i-1},$$

$$f(x_i) = a_i + b_i h_i + c_i h_i^2 + d_i h_i^3, h_i = x_i - x_{i-1}, i = 1,2, \dots, n-1. \quad (1)$$

Первая и вторая производные равны:

$$f'(x) = b_i + 2c_i(x - x_{i-1}) + 3d_i(x - x_{i-1})^2,$$

$$f''(x) = 2c_i + 6d_i(x - x_{i-1}), x_{i-1} \le x \le x_i.$$

Из условий $f'(x_i - 0) = f'(x_i + 0)$ и $f''(x_i - 0) = f''(x_i + 0)$ получаются равенства:

$$b_{i+1} = b_i + 2c_i h_i + 3d_i h_i^2,$$

$$c_{i+1} = c_i + 3d_i h_i, i = 1,2, \dots, n-1.$$

Из краевых условий $f''(x_0) = 0$ и $f''(x_n) = 0$ следуют равенства:

$$c_1 = 0, c_n + 3d_n h_n = 0.$$

Из приведенных выше равенств получаются выражения для коэффициентов a_i, b_i, d_i:

$$a_i = y_{i-1}, \quad (2)$$

$$b_i = \frac{y_i - y_{i-1}}{h_i} - h_i \frac{c_{i+1} + 2c_i}{3}, i = 1,2, \dots, n-1, \quad (3)$$

$$b_n = \frac{y_n - y_{n-1}}{h_n} - h_n \frac{2c_n}{3}, \quad (4)$$

$$d_i = \frac{c_{i+1} - c_i}{3h_i}, i = 1,2, \dots, n-1, \quad (5)$$

$$d_n = -\frac{c_n}{3h_n}. \quad (6)$$

Для коэффициентов c_i получается система уравнений:

$$h_i c_i + 2(h_i + h_{i+1})c_{i+1} + h_{i+1}c_{i+2} = 3\left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i}\right), i = 1,2, \dots, n-1,$$

$$c_1 = 0, c_{n+1} = 0.$$

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

$$c_i = \xi_{i+1} c_{i+1} + \eta_{i+1}, i = 1,2, \dots, n-1. \quad (7)$$

Подстановкой в систему уравнений для c_i выражений для c_{i+2} получаются выражения для ξ_{i+1} и η_{i+1}:

$$\xi_{i+1} = -\frac{h_i}{h_{i-1}\xi_i + 2(h_{i-1} + h_i)}, \quad (8)$$

$$\eta_{i+1} = \frac{f_i - h_{i-1}\eta_i}{h_{i-1}\xi_i + 2(h_{i-1} + h_i)}. \quad (9)$$

В этих формулах принято обозначение:

$$f_i = 3\left(\frac{y_i - y_{i-1}}{h_i} - \frac{y_{i-1} - y_{i-2}}{h_{i-1}}\right).$$

Из условия $c_1 = 0$ следует $\xi_2 = 0, \eta_2 = 0$.

Алгоритм

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

1. По формулам (8) и (9) вычисляются коэффициенты ξ_{i+1}, η_{i+1} с учетом, что $\xi_2 = 0$, $\eta_2 = 0$.
2. По формуле (7) вычисляются коэффициенты c_i.
3. По формулам (2-6) вычисляются оставшиеся коэффициенты a_i, b_i, d_i.

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.

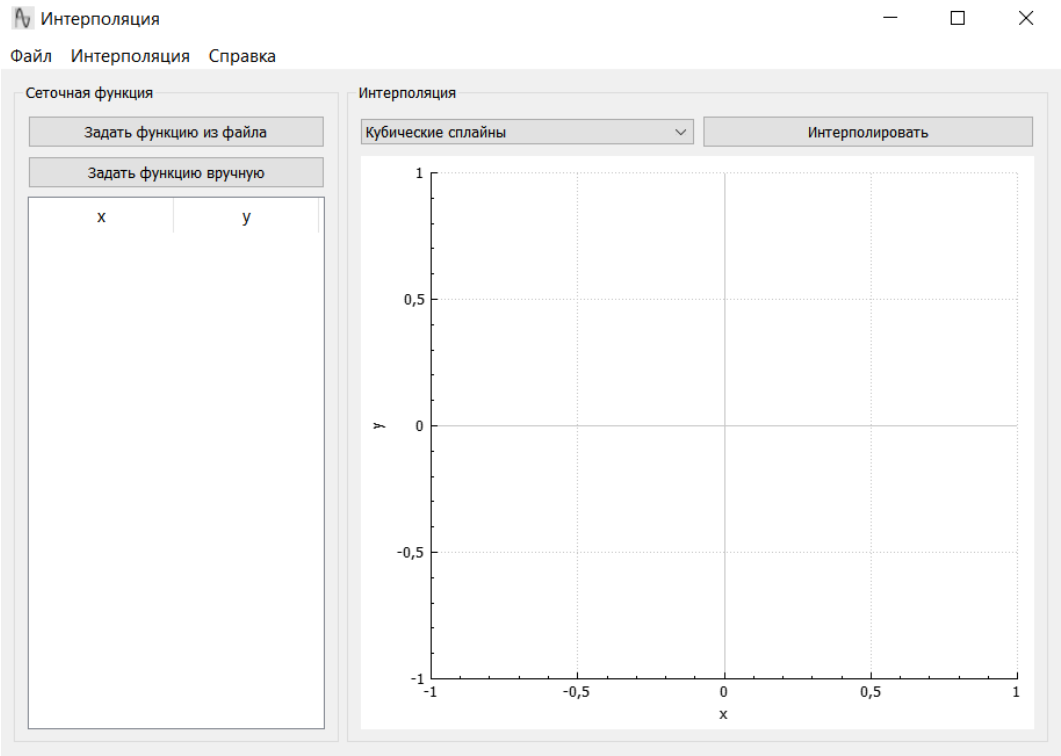


Рис. 1

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.

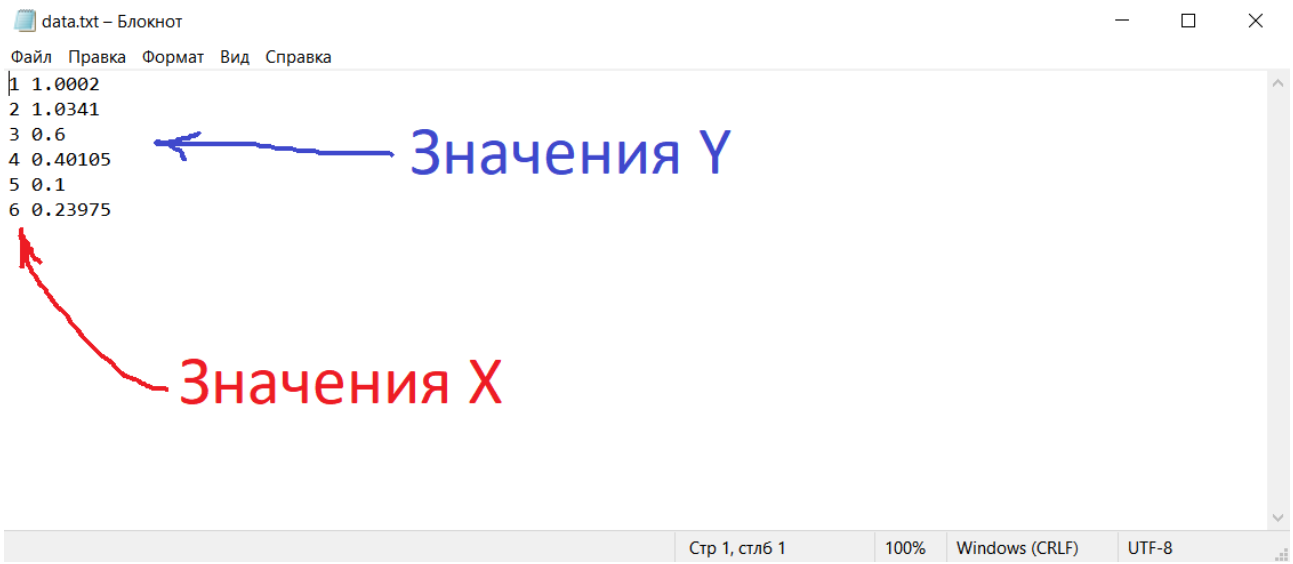


Рис. 2

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

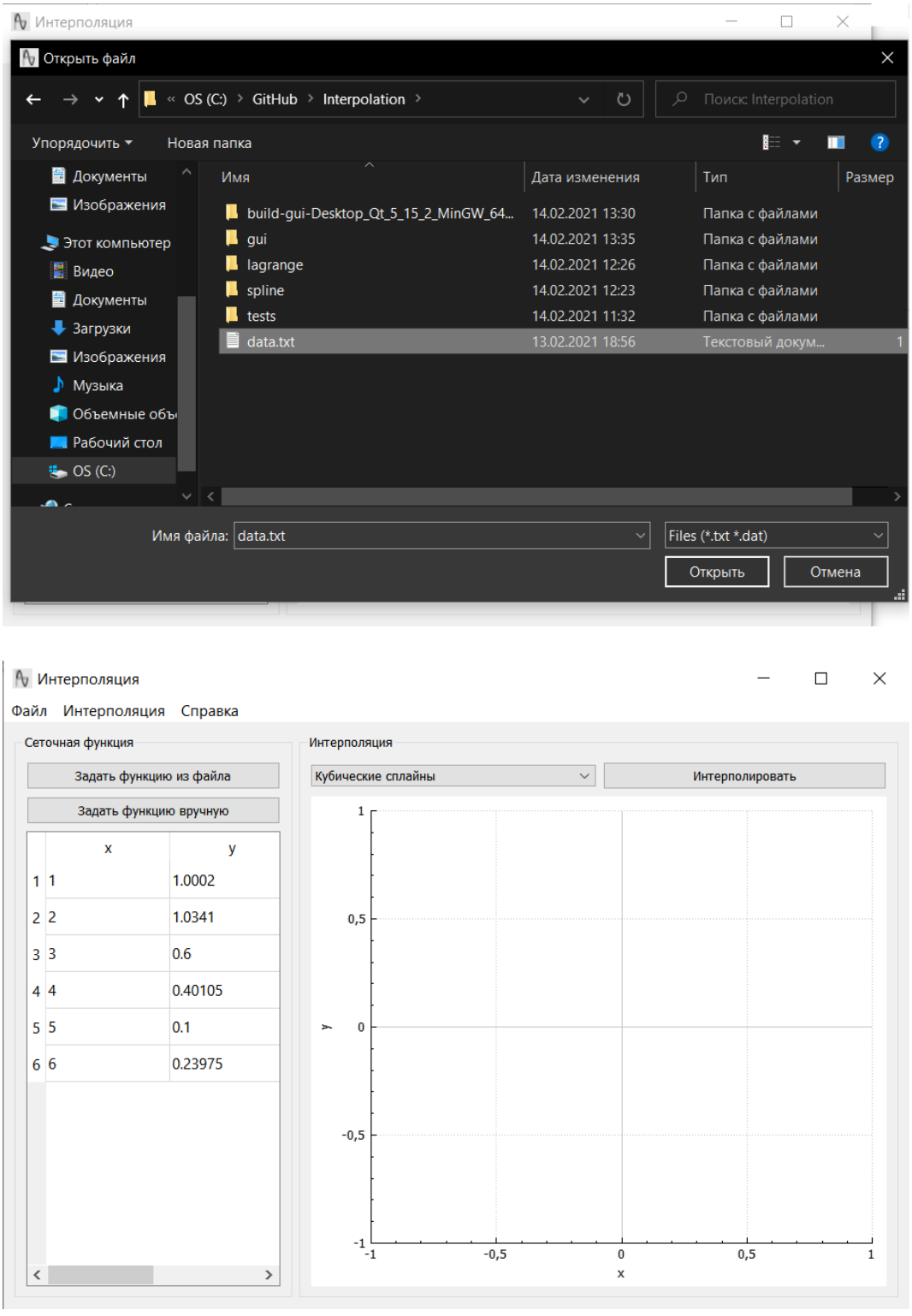


Рис. 3

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

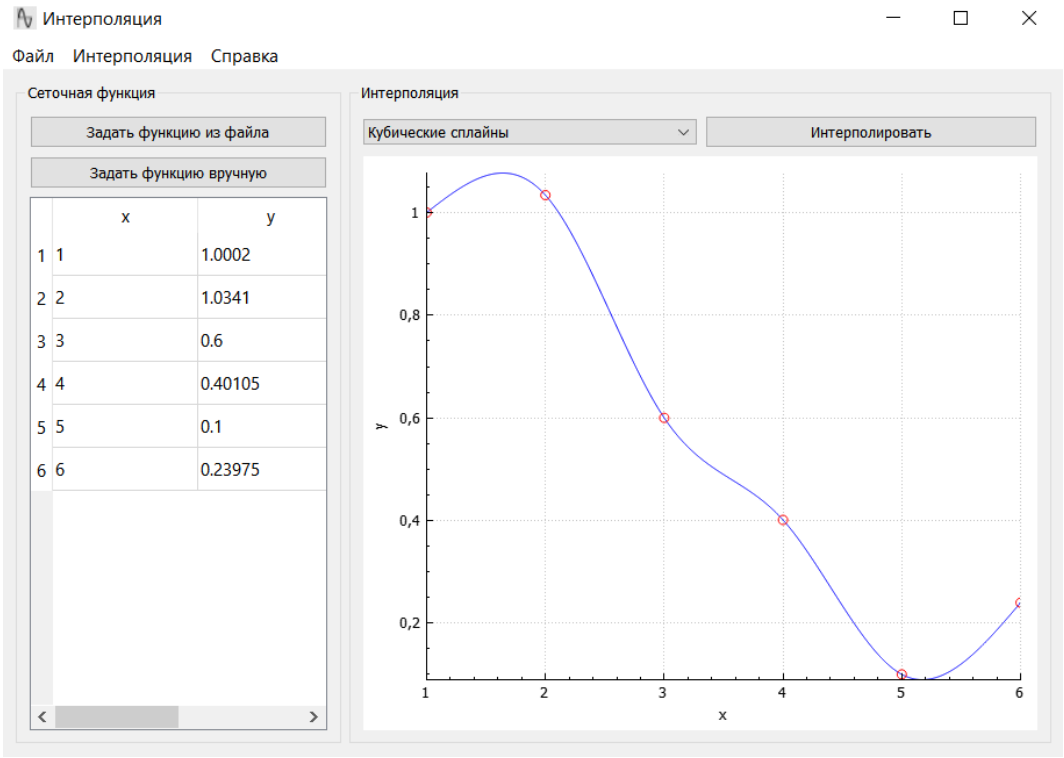
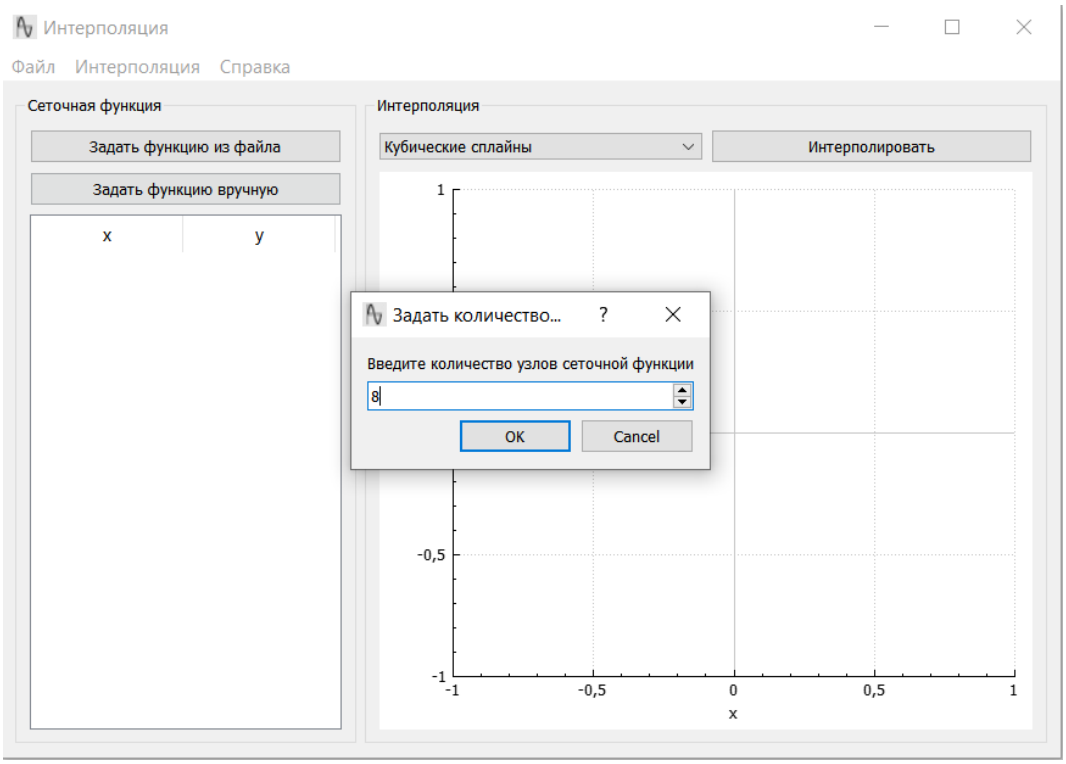


Рис. 4

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 x и значения функции y в этих узлах (рис. 5).



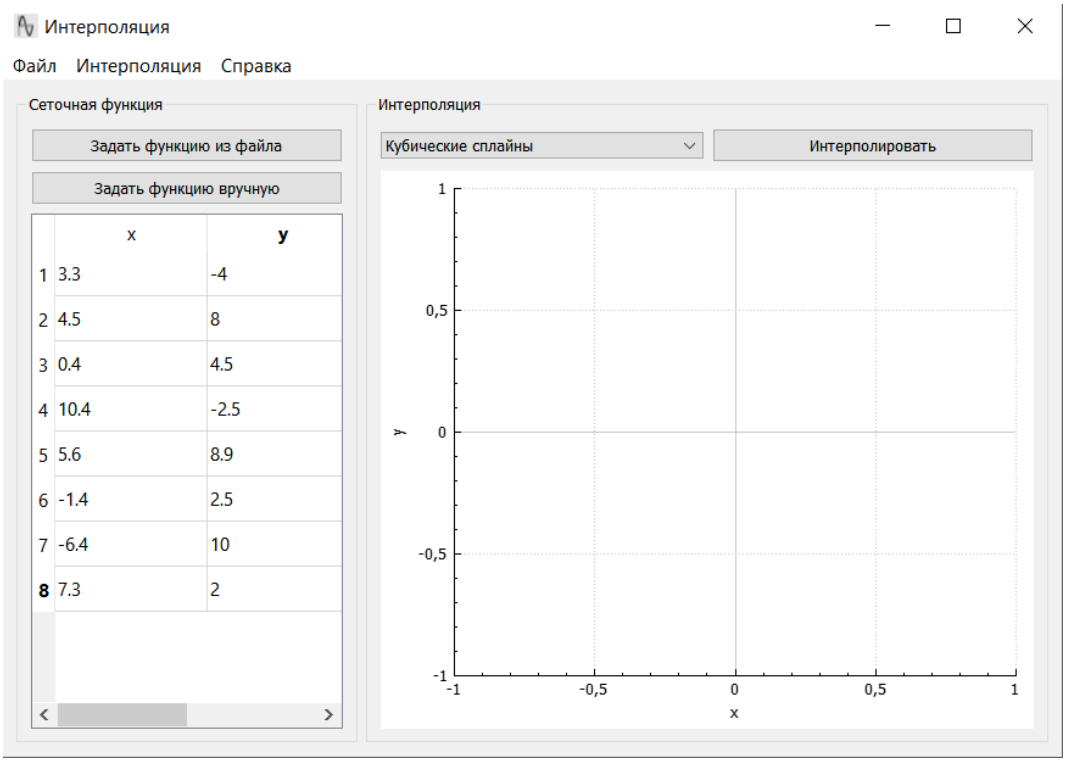


Рис. 5

На рис. 6 показан график интерполированной функции.

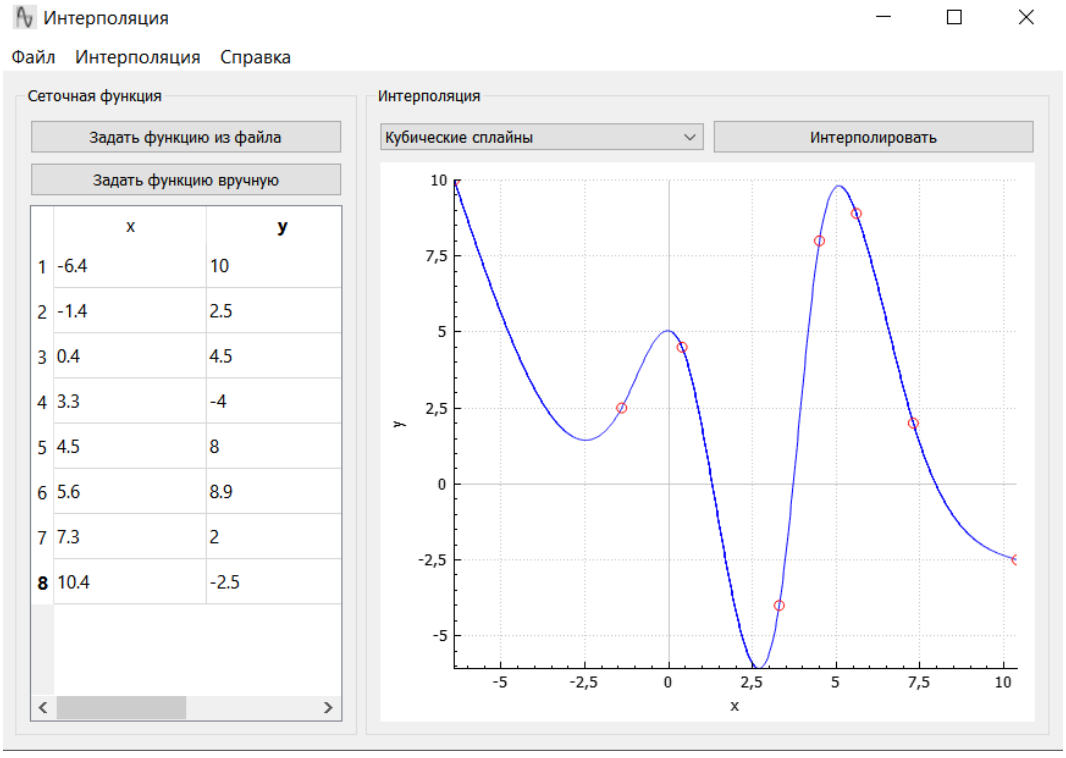


Рис. 6

Список литературы

1. Интерполяция кубическими сплайнами // http://www.machinelearning.ru/wiki/index.php?title=Интерполяция_кубическими_сплайнами

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 с 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();
}
```