МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Лабораторная работа №5

«Численное дифференцирование функций»

по дисциплине

Вычислительная математика

РУКОВОДИТЕЛЬ:

________________ Суркова А.С.

СТУДЕНТ:

________________ Сухоруков В.А.

19-ИВТ-3

Работа защищена «___» ____________

С оценкой ________________________

Нижний Новгород 2021

Оглавление

Цель

Закрепление знаний и умений по численному дифференцированию функций с помощью интерполяционного многочлена Ньютона и метода неопределенных коэффициентов.

Постановка задачи

Найти первую и вторую производную функции в точках x, заданных таблицей, используя интерполяционные многочлены Ньютона. Сравнить со значениями производных, вычисленными по формулам, основанным на интерполировании многочленом Лагранжа (вычисление производных через значения функций).

7.

x	y
1.340	4.25562
1.345	4.35325
1.350	4.45522
1.355	4.56184
1.360	4.67344
1.365	4.79038
1.370	4.91306
1.375	5.04192
1.380	5.17744
1.385	5.32016
1.390	5.47069
1.395	5.62968

Теоретические сведения

Метод Ньютона

Предположим, что функция $f(x)$, заданная в виде таблицы с постоянным шагом $h = x_i - x_{i-1}$ может быть аппроксимированная интерполяционным многочленом Ньютона:

$$y \approx N(x_0 + th) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_0 + \cdots +$$
$$+ \frac{t(t-1)\ldots(t-n+1)}{n!}\Delta^n y_0 \quad t = \frac{x - x_0}{h}$$

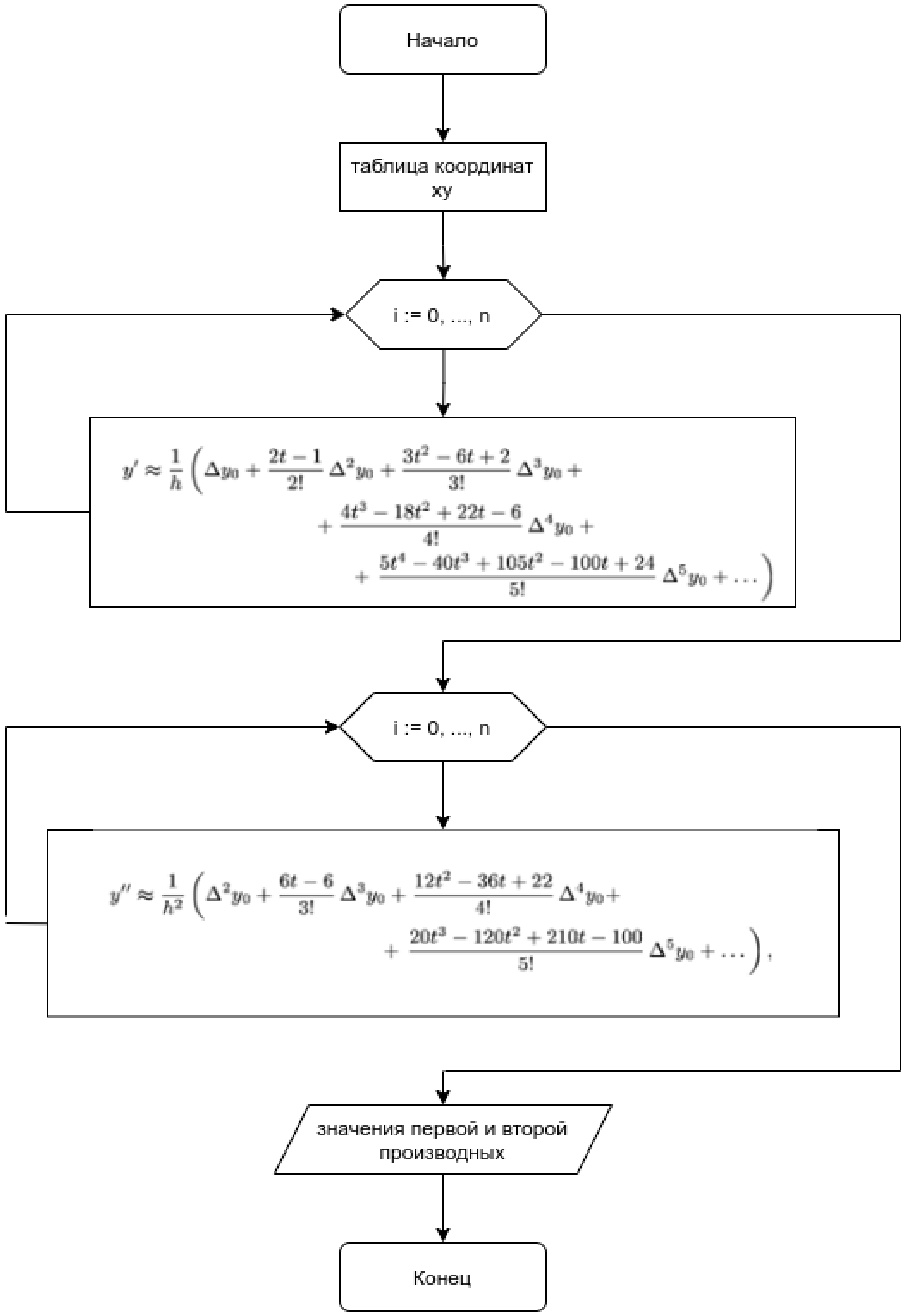
Дифференцируя этот многочлен по переменной x с учетом правила дифференцирования сложной функции:

$$\frac{dN}{dx} = \frac{dN}{dt}\frac{dt}{dx} = \frac{1}{h}\frac{dN}{dt},$$

можно получить формулы для вычисления производных любого порядка:

$$y' \approx \frac{1}{h}\left(\Delta y_0 + \frac{2t-1}{2!}\Delta^2 y_0 + \frac{3t^2-6t+2}{3!}\Delta^3 y_0 + \right.$$
$$+ \frac{4t^3-18t^2+22t-6}{4!}\Delta^4 y_0 +$$
$$\left. + \frac{5t^4-40t^3+105t^2-100t+24}{5!}\Delta^5 y_0 + \ldots\right),$$

$$y'' \approx \frac{1}{h^2}\left(\Delta^2 y_0 + \frac{6t-6}{3!}\Delta^3 y_0 + \frac{12t^2-36t+22}{4!}\Delta^4 y_0 + \right.$$
$$\left. + \frac{20t^3-120t^2+210t-100}{5!}\Delta^5 y_0 + \ldots\right),$$



Метод Лагранжа

Используя формулы интерполяционного многочлена Лагранжа можно получить формулы для производных, выраженные через значение функции.

$$f'(x) = L'(x)$$

$$y_0' = \frac{1}{12h}(-25y_0 + 48y_1 - 36y_2 + 16y_3 - 3y_4)$$

$$y_1' = \frac{1}{12h}(-3y_0 - 10y_1 + 18y_2 - 6y_3 + y_4)$$

$$y_i' = \frac{1}{12h}(y_{i-2} - 8y_{i-1} + 8y_{i+1} - y_{i+2})$$

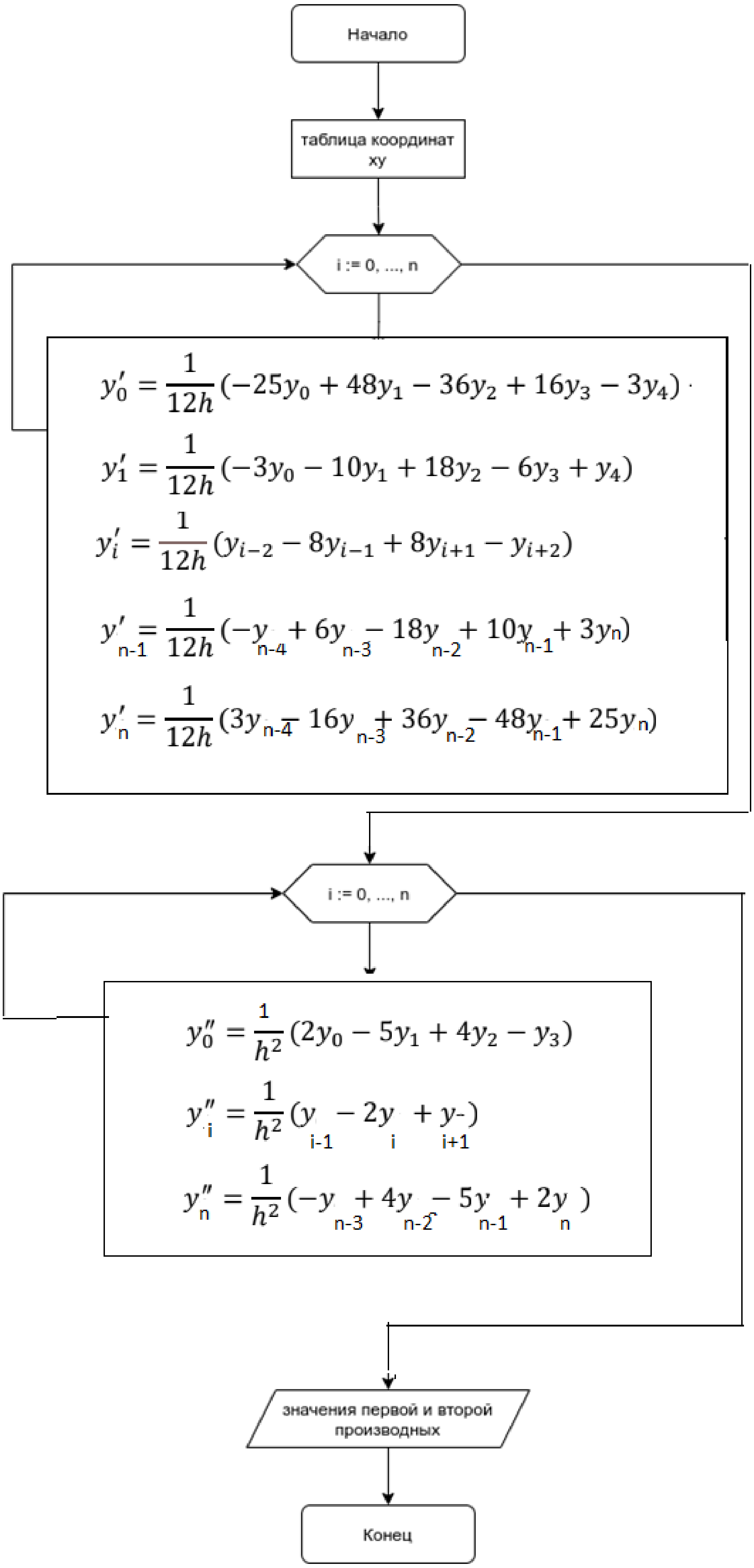
$$y_{n-1}' = \frac{1}{12h}(-y_{n-4} + 6y_{n-3} - 18y_{n-2} + 10y_{n-1} + 3y_n)$$

$$y_n' = \frac{1}{12h}(3y_{n-4} - 16y_{n-3} + 36y_{n-2} - 48y_{n-1} + 25y_n)$$

$$y_0'' = \frac{1}{h^2}(2y_0 - 5y_1 + 4y_2 - y_3)$$

$$y_i'' = \frac{1}{h^2}(y_{i-1} - 2y_i + y_{i+1})$$

$$y_n'' = \frac{1}{h^2}(-y_{n-3} + 4y_{n-2} - 5y_{n-1} + 2y_n)$$



Начало
таблица координат xy
i := 0, ..., n
$y'_0 = \frac{1}{12h}(-25y_0 + 48y_1 - 36y_2 + 16y_3 - 3y_4)$
$y'_1 = \frac{1}{12h}(-3y_0 - 10y_1 + 18y_2 - 6y_3 + y_4)$
$y'_i = \frac{1}{12h}(y_{i-2} - 8y_{i-1} + 8y_{i+1} - y_{i+2})$
$y'_{n-1} = \frac{1}{12h}(-y_{n-4} + 6y_{n-3} - 18y_{n-2} + 10y_{n-1} + 3y_n)$
$y'_n = \frac{1}{12h}(3y_{n-4} - 16y_{n-3} + 36y_{n-2} - 48y_{n-1} + 25y_n)$
i := 0, ..., n
$y''_0 = \frac{1}{h^2}(2y_0 - 5y_1 + 4y_2 - y_3)$
$y''_i = \frac{1}{h^2}(y_{i-1} - 2y_i + y_{i+1})$
$y''_n = \frac{1}{h^2}(-y_{n-3} + 4y_{n-2} - 5y_{n-1} + 2y_n)$
значения первой и второй производных
Конец

Расчетные данные

x	y'
1.340	4.25562
1.345	4.35325
1.350	4.45522
1.355	4.56184
1.360	4.67344
1.365	4.79038
1.370	4.91306
1.375	5.04192
1.380	5.17744
1.385	5.32016
1.390	5.47069
1.395	5.62968
1.340	4.25562
1.345	4.35325
1.350	4.45522

```
Ньютон первая производная
        xi       |        y'i
--------------------------------------
        1.340    |        19.11207
        1.345    |        19.94990
        1.350    |        20.84840
        1.355    |        21.81057
        1.360    |        22.84140
        1.365    |        23.94790
        1.370    |        23.44530
        1.375    |        24.72947
        1.380    |        26.11263
        1.385    |        27.61180
        1.390    |        29.23797
        1.395    |        30.99613

Ньютон вторая производная
        xi       |        y''i
--------------------------------------
        1.340    |        161.60000
        1.345    |        173.56667
        1.350    |        185.93333
        1.355    |        199.10000
        1.360    |        213.46667
        1.365    |        229.43333
        1.370    |        248.36667
        1.375    |        266.06667
        1.380    |        287.76667
        1.385    |        312.26667
        1.390    |        338.36667
        1.395    |        364.86667
```

```
Лагранж первая производная
        xi        |        y'i
------------------------------------
        1.340     |        19.11167
        1.345     |        19.95000
        1.350     |        20.84833
        1.355     |        21.81050
        1.360     |        22.84133
        1.365     |        23.94800
        1.370     |        25.13867
        1.375     |        26.42100
        1.380     |        27.80483
        1.385     |        29.30400
        1.390     |        30.92967
        1.395     |        32.68933

Лагранж вторая производная
        xi        |        y''i
------------------------------------
        1.340     |        161.20000
        1.345     |        173.60000
        1.350     |        186.00000
        1.355     |        199.20000
        1.360     |        213.60000
        1.365     |        229.60000
        1.370     |        247.20000
        1.375     |        266.40000
        1.380     |        288.00000
        1.385     |        312.40000
        1.390     |        338.40000
        1.395     |        364.40000
```

Код программы

Value_function_table.h

```
#pragma once
#include<vector>
#include<iostream>
#include<fstream>
#include<string>
#include"Colors.h"
#include <iomanip>
using namespace std;

/*Класс для описания таблицы значений функции*/
class Value_function_table{
public:
	vector<double>x;                    //Координаты x точек
	vector<double>y;
	//Координаты y точек
	size_t n;                           //Количество точек

	Value_function_table() {
		n = 0;
```

```
    }

    //Функция заполнения таблицы
    void set_value() {
        setlocale(LC_ALL, "Russian");    //Включение русского
языка в консоли

        bool is_readed = false;
        while (is_readed == false) {
            cout<<Green << "Выберите способ ввода данных\n"
                << "\t{1} - ручной ввод в консоль\n"
                << "\t{2} - чтение из файла\n";
            int metod;
            cin >> metod;
            if (metod == 1) {
                cout << Green
                << "Введите количество точек в таблице ";
                int k;
                double x_val, y_val;
                cin >> k;
                this->n = k;
                for (size_t i = 1; i <= n; i++) {
                    cout << Yellow
                         << "\n\tВведите координату x "
                         << i << " точки ";
                    cin >> x_val;
                    cout << "\tВведите координату y "
                         << i << " точки ";
                    cin >> y_val;

                    this->x.push_back(x_val);
                    this->y.push_back(y_val);
                    cout << Reset << "\n";
                }
                is_readed = true;
            }
            else {
                if (metod == 2) {
                    cout << Green << "Введите имя файла ";
                    string file_name;
                    cin >> file_name;
                    ifstream in(file_name);
                    int k;
                    double x_val, y_val;
                    in >> k;
                    this->n = k;
                    for (size_t i = 1; i <= n; i++) {
                        in >> x_val >> y_val;

                        this->x.push_back(x_val);
                        this->y.push_back(y_val);
                    }
                    is_readed = true;
```

```
                }
            }
        }

        //Вывод сформированной таблицы в консоль
        cout << Yellow << "Сформированная таблица:\n"
             << Green << "\n    X    |    Y\n"
             << " ------------------\n";
        for (int i = 0; i < this->x.size(); i++) {
             cout << Blue<< "   " <<fixed<< setprecision(3)
                  << this->x[i] << "   " << Green
                  << " |" << Blue << setw(9) << setprecision(4)
                  << this->y[i] << "\n";
        }

    }

};
```

Newton.h

```
#ifndef _Newton_
#define _Newton_
#include <vector>
#include <iostream>
#include <iomanip>
#include "Colors.h"
#include "Value_function_table.h"
using namespace std;


//Метод для нахождения конечных разнстей
vector<vector<double>>
get_finite_differences(Value_function_table t) {
     vector<vector<double>> res;

     //Вычисления конечные разностей первого порядка
     vector<double> temp;
     for (size_t  i = 1; i < t.n; i++) {
          temp.push_back(t.y[i] - t.y[i - 1]);
     }
     res.push_back(temp);

     //На каждом i-ом шаге вычисляем значения конченых разностей
нового порядка
     //и заносим в промежуточный список.
     //Полученный промежуточный список заносим в список списков
промежуточных разностей
     for (size_t i = 0; i < t.n - 2; i++) {
          //Создание нового вектора конечных разностей
          vector<double>tmp;
          for (size_t j = 0; j < res[i].size() - 1; j++) {
               //Вычисление конечных разностей
```

```
                tmp.push_back(res[i][j + 1] - res[i][j]);
            }
            res.push_back(tmp);
        }
        return res;
}


/*Метод для вывода конечных разностей i - ого
  порядка в "лестничном виде"
*/
void print_finite_differences(vector<vector<double>>
finiteDifferences) {
  for (size_t  i = 0; i < finiteDifferences.size(); i++){
     cout<<Green << "Конечные разности  "<<setw(3)<<(i + 1)
          << " порядка: ";

    for (size_t  j = 0; j < finiteDifferences[i].size(); j++){
                   cout<<Blue << setw(7)<<fixed
                        << setprecision(4)
                        << finiteDifferences[i][j] << " ";
    }
    cout<<Reset << "\n";
  }
     return;
}

//Метод для получения факториала
int getFact(int n){
     int res = 1;
     while (n > 1){
          res *= n;
          n--;
     }
     return res;
}

/*Метод для приближенного вычисления значений первой производной
  при помощи интерполяционной формулы Ньютона
     Параметры:
          1)table - таблица значений функции
*/
vector<double> Newton_first_derivative(Value_function_table
table) {
     vector<double> res;

     //Нахождение конечных разностей
     vector<vector<double>> finiteDifferences;
     finiteDifferences = get_finite_differences(table);
     print_finite_differences(finiteDifferences);

     //Вычисление шага h
     double h = table.x[1] - table.x[0];
```

```
    //Перемнная для хранения параметра t
    double t;

    //Вычисление середины отрезка переданных X
    double mid = (table.x[0] + table.x[(table.n) - 1]) / 2;

    //Нахождение значения функции в каждой переданной точке
    for (size_t k = 0; k < table.x.size(); k++){

        //Переменная для хранения результата
        double r = 0;

        //Если Xi лежит в промежутке левее середины
        //То значение функции вычисляется методом интерполяции
        //вперед
        if (table.x[k] < mid) {
            //t вычисляется как (x - x0)/h
            t = (table.x[k] - table.x[0]) / h;

            //t вычисляется как (x - x0)/h
            t = (table.x[k] - table.x[0]) / h;

            //К результату прибавляются ΔY0
            r += finiteDifferences[0][0];
            //прибавляем к результату последующие слагаемые до
            //Δ^5y0
            r += ((2.0 * t - 1.0) * finiteDifferences[1][0] /
                  getFact(2));

            r += ((3.0 * t * t - 6.0 * t + 2.0) *
                 finiteDifferences[2][0] / getFact(3));

            r += ((4.0 * t * t * t - 18.0 * t * t + 22.0 * t -
             6.0) * finiteDifferences[3][0] / getFact(4));

            r += ((5.0 * t * t * t * t - 40.0 * t * t * t +
                  105.0 * t * t - 100.0 * t + 24.0) *
                  finiteDifferences[4][0] / getFact(5));

            //Делим полученный результат на h
            r = r / h;

            //В вектор ответов заносим полученное значение
            res.push_back(r);
        }
        //Иначе Xi лежит в промежутке правее середины
        //значение функции вычисляется методом интерполяции
назад
        else {
            //t вычисляется как (x - xn)/h
            t = (table.x[k] - table.x[table.n - 1]) / h;
```

```
			//К результату прибавляются ΔY(n-1)
			r += finiteDifferences[0]
				[finiteDifferences[0].size() - 2];
			//прибавляем к результату последующие слагаемые до
ΔY(n-6)
			r += ((2.0 * t + 1.0) *
			   finiteDifferences[1]
			   [finiteDifferences[1].size() - 1]
			    / getFact(2));

			r += ((3.0 * t * t + 6.0 * t + 2.0) *
				finiteDifferences[2]
				[finiteDifferences[2].size() - 1] /
				getFact(3));

			r += ((4.0 * t * t * t + 18.0 * t * t + 22.0 * t +
				 6.0) * finiteDifferences[3]
				[finiteDifferences[3].size() - 1] /
				 getFact(4));

		r += ((5.0 * t * t * t * t + 40.0 * t * t * t +
				 105.0 * t * t + 100.0 * t + 24.0) *
				finiteDifferences[4]
				[finiteDifferences[4].size() - 1] /
				getFact(5));

			//Делим полученный результат на h
			r = r / h;

			//В вектор ответов заносим полученное значение
			res.push_back(r);
			}
	}

	//Вывод результатов в консоль
	cout << Blue << "\nНьютон первая производная\n"
		<< Green << "\txi\t|\ty'i\n"
		<< "-------------------------------------\n";
	for (size_t i = 0; i < res.size(); i++) {
		cout << "\t" << setprecision(3) << table.x[i]
			<< "\t|\t" << setprecision(5) << res[i] << "\n";
	}
	return res;
}


/*Метод для приближенного вычисления значений второй производной
	при помощи интерполяционной формулы Ньютона
	Параметры:
		1)table - таблица значений функции
*/
vector<double> Newton_second_derivative(Value_function_table
table) {
```

```
vector<double> res;

//Нахождение конечных разностей
vector<vector<double>> finiteDifferences;
finiteDifferences = get_finite_differences(table);

//Вычисление шага h
double h = table.x[1] - table.x[0];

//Перемнная для хранения параметра t
double t;

//Вычисление середины отрезка переданных X
double mid = (table.x[0] + table.x[(table.n) - 1]) / 2;

//Нахождение значения функции в каждой переданной точке
for (size_t k = 0; k < table.x.size(); k++) {
     //Переменная для хранения результата
     double r = 0;

     //Если Xi лежит в промежутке левее середины
     //То значение функции вычисляется методом интерполяции
     //вперед
     if (table.x[k] < mid) {
          //t вычисляется как (x - x0)/h
          t = (table.x[k] - table.x[0]) / h;

          //К результату прибавляются Δ2Y0
          r += finiteDifferences[1][0];

          //прибавляем к результату последующие слагаемые до
          //Δ^5y0
          r += ((6.0 * t - 6.0) * finiteDifferences[2][0])
                / getFact(3);

          r += ((12.0 * t * t - 36.0 * t + 22.0) *
               finiteDifferences[3][0]) / getFact(4);

          r += ((20.0 * t * t * t - 120.0 * t * t + 210.0 *
               t - 100.0) * finiteDifferences[4][0] /
               getFact(5));

          //Делим полученный результат на h^2
          r /= (h * h);

          //В вектор ответов заносим полученное значение
          res.push_back(r);
     }
     //Иначе Xi лежит в промежутке правее середины
     //значение функции вычисляется методом интерполяции
назад
     else {
          //t вычисляется как (x - xn)/h
```

```
                t = (table.x[k] - table.x[table.n - 1]) / h;

                //К результату прибавляются Δ2Y(n-2)
                r += finiteDifferences[1]
                     [finiteDifferences[0].size() - 2];

                //прибавляем к результату последующие слагаемые до
                //ΔY(n-5)
                r += ((6.0 * t + 6.0) *
                      finiteDifferences[2]
                     [finiteDifferences[2].size() - 1])
                     / getFact(3);

                r += ((12.0 * t * t + 36.0 * t + 22.0) *
                     finiteDifferences[3]
                     [finiteDifferences[3].size() - 1])
                     / getFact(4);

                r += ((20.0 * t * t * t + 120.0 * t * t + 210.0 *
                     t + 100.0) * finiteDifferences[4]
                     [finiteDifferences[4].size() - 1] /
                      getFact(5));;

                //Делим полученный результат на h^2
                r /= (h * h);

                //В вектор ответов заносим полученное значение
                res.push_back(r);
            }
        }

        //Вывод результатов в консоль
        cout << Blue << "\nНьютон вторая производная\n"
             << Green << "\txi\t|\ty''i\n"
             << "-------------------------------------\n";
        for (size_t i = 0; i < res.size(); i++) {
             cout << "\t"<< setprecision(3) << table.x[i]
                   << "\t|\t" << setprecision(5) << res[i] << "\n";
        }
        return res;
}
#endif
```

Lagrange.h

```
#ifndef _Lagrange_
#define _Lagrange_
#include <vector>
#include <iostream>
#include <iomanip>
#include "Colors.h"
#include "Value_function_table.h"
using namespace std;
```

```
/*Метод для приближенного вычисления значений первой производной
	при помощи интерполяционной формулы Ньютона
	Параметры:
		1)table - таблица значений функции
*/
vector<double> Lagrange_first_derivative(Value_function_table
table) {
	vector<double> res;
	double h = table.x[1] - table.x[0];
	double  r;

	// вычисление производной в начальных точках
	r = (-25 * table.y[0] + 48 * table.y[1] - 36 * table.y[2] +
		16 * table.y[3] - 3 * table.y[4]) / (12 * h);

	res.push_back(r);

	r = (-3 * table.y[0] - 10 * table.y[1] + 18 * table.y[2] -
		 6 *  table.y[3] +  table.y[4]) / (12 * h);

	res.push_back(r);

	// вычисление производной в средних точках
	for (int i = 2; i < 10; i++){
		r = ( table.y[i - 2] - 8 *  table.y[i - 1] + 8 *
		  table.y[i + 1] -  table.y[i + 2]) / (12 * h);

		res.push_back(r);
	}

	// вычисление производной в последних точках
	r = (- table.y[7] + 6 *  table.y[8] - 18 *  table.y[9] +
		 10 *  table.y[10] + 3 *  table.y[11]) / (12 * h);

	res.push_back(r);

	r = (3 *  table.y[7] - 16 *  table.y[8] + 36 *  table.y[9] -
		48 *  table.y[10] + 25 *  table.y[11]) / (12 * h);

	res.push_back(r);


	//Вывод результатов в консоль
	cout << Blue << "\nЛагранж первая производная\n"
		<<Green<< "\txi\t|\ty'i\n"
		<< "--------------------------------------\n";
	for (size_t i = 0; i < res.size(); i++) {
		cout << "\t" << setprecision(3) << table.x[i]
		      << "\t|\t" << setprecision(5) << res[i] << "\n";
	}
	return res;
}
```

```
/*Метод для приближенного вычисления значений первой производной
	при помощи интерполяционной формулы Ньютона
	Параметры:
		1)table - таблица значений функции
*/
vector<double> Lagrange_second_derivative(Value_function_table
table) {
	double h, res ;
	vector<double> r;
	h =  table.x[1] -  table.x[0];

	// вычисление производной в начальной точке
	res = (2.0 *  table.y[0] - 5.0 *  table.y[1] + 4.0 *
		  table.y[2] -  table.y[3]) / (h * h);

	r.push_back(res);

	// вычисление производной в средних точках
	for (int i = 1; i < 11; i++){
		res = ( table.y[i - 1] - 2 *  table.y[i] +
			  table.y[i + 1]) / (h * h);

		r.push_back(res);
	}

	// вычисление производной в конечной точке
	res = (- table.y[8] + 4 *  table.y[9] - 5 *  table.y[10] +
		  2 *  table.y[11]) / (h * h);

	r.push_back(res);

	cout << Blue <<"\nЛагранж вторая производная\n"
		<< Green << "\txi\t|\ty''i\n"
		<< "-------------------------------------\n";
	for (size_t i = 0; i < r.size(); i++) {
		cout << "\t" << setprecision(3)<< table.x[i]
			<< "\t|\t" << setprecision(5) << r[i] << "\n";
	}
	return r;
}

#endif
```

Main.cpp

```
#include<iostream>
#include<fstream>
#include<vector>
#include"Colors.h"
#include"Value_function_table.h"
#include"Newton.h"
#include"Lagrange.h"
```

```cpp
using namespace std;
int main() {
	setlocale(LC_ALL, "Russian");	//Включение русского языка в консоли
	cout << Yellow << "Ввод данных таблицы\n";
	Value_function_table table;
	table.set_value();
	Newton_first_derivative(table);
	Newton_second_derivative(table);
	Lagrange_first_derivative(table);
	Lagrange_second_derivative(table);
	return 0;
}
```

Результат работы программы

```
Ввод данных таблицы
Выберите способ ввода данных
        {1} - ручной ввод в консоль
        {2} - чтение из файла
2
Введите имя файла 1.txt
Сформированная таблица:

    X    |    Y
 -----------------
  1.340   |   4.2556
  1.345   |   4.3533
  1.350   |   4.4552
  1.355   |   4.5618
  1.360   |   4.6734
  1.365   |   4.7904
  1.370   |   4.9131
  1.375   |   5.0419
  1.380   |   5.1774
  1.385   |   5.3202
  1.390   |   5.4707
  1.395   |   5.6297
Конечные разности    1 порядка:  0.0976  0.1020  0.1066  0.1116  0.1169  0.1227  0.1289  0.1355  0.1427  0.1505  0.1590
Конечные разности    2 порядка:  0.0043  0.0047  0.0050  0.0053  0.0057  0.0062  0.0067  0.0072  0.0078  0.0085
Конечные разности    3 порядка:  0.0003  0.0003  0.0004  0.0004  0.0004  0.0005  0.0005  0.0006  0.0006
Конечные разности    4 порядка:  0.0000  0.0000  0.0000  0.0000  0.0000  0.0001  0.0001  0.0000
Конечные разности    5 порядка:  0.0000  0.0000 -0.0000 -0.0000  0.0000  0.0000 -0.0000
Конечные разности    6 порядка: -0.0000 -0.0000  0.0000  0.0000 -0.0000 -0.0000
Конечные разности    7 порядка: -0.0000  0.0000  0.0000 -0.0000 -0.0000
Конечные разности    8 порядка:  0.0000  0.0000 -0.0001  0.0000
Конечные разности    9 порядка: -0.0000 -0.0001  0.0001
Конечные разности   10 порядка: -0.0001  0.0001
Конечные разности   11 порядка:  0.0002
```

```
Ньютон первая производная
          xi        |          y'i
-----------------------------------
          1.340     |          19.11207
          1.345     |          19.94990
          1.350     |          20.84840
          1.355     |          21.81057
          1.360     |          22.84140
          1.365     |          23.94790
          1.370     |          23.44530
          1.375     |          24.72947
          1.380     |          26.11263
          1.385     |          27.61180
          1.390     |          29.23797
          1.395     |          30.99613
```

Ньютон вторая производная

xi	y''i
1.340	161.60000
1.345	173.56667
1.350	185.93333
1.355	199.10000
1.360	213.46667
1.365	229.43333
1.370	248.36667
1.375	266.06667
1.380	287.76667
1.385	312.26667
1.390	338.36667
1.395	364.86667

Лагранж первая производная

xi	y'i
1.340	19.11167
1.345	19.95000
1.350	20.84833
1.355	21.81050
1.360	22.84133
1.365	23.94800
1.370	25.13867
1.375	26.42100
1.380	27.80483
1.385	29.30400
1.390	30.92967
1.395	32.68933

Лагранж вторая производная

xi	y''i
1.340	161.20000
1.345	173.60000
1.350	186.00000
1.355	199.20000
1.360	213.60000
1.365	229.60000
1.370	247.20000
1.375	266.40000
1.380	288.00000
1.385	312.40000
1.390	338.40000
1.395	364.40000

Вывод

В ходе данной работы были закреплены знания и умения по вычислению производных первого и второго порядка при помощи интерполяционных многочленов Ньютона и Лагранжа. Значения, полученные двумя способами совпадают в пределах погрешности.