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 3](#_Toc70595868)

[Постановка задачи 4](#_Toc70595869)

[Теоретические сведения 5](#_Toc70595870)

[Метод Ньютона 5](#_Toc70595871)

[Метод Лагранжа 6](#_Toc70595872)

[Расчетные данные 9](#_Toc70595873)

[Код программы 10](#_Toc70595874)

[Value\_function\_table.h 10](#_Toc70595875)

[Newton.h 12](#_Toc70595876)

[Lagrange.h 17](#_Toc70595877)

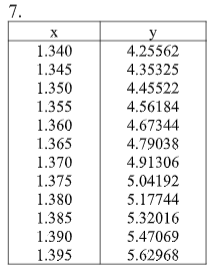
[Вывод 19](#_Toc70595878)

# 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 х, заданных таблицей, используя интерполяционные многочлены Ньютона. Сравнить со значениями производных, вычисленными по формулам, основанным на интерполировании многочленом Лагранжа (вычисление производных через значения функций).



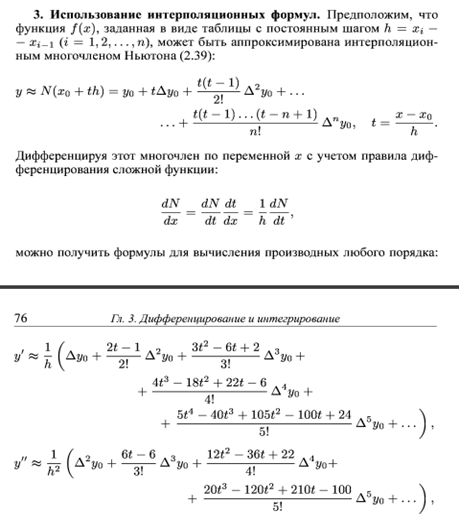
# Теоретические сведения

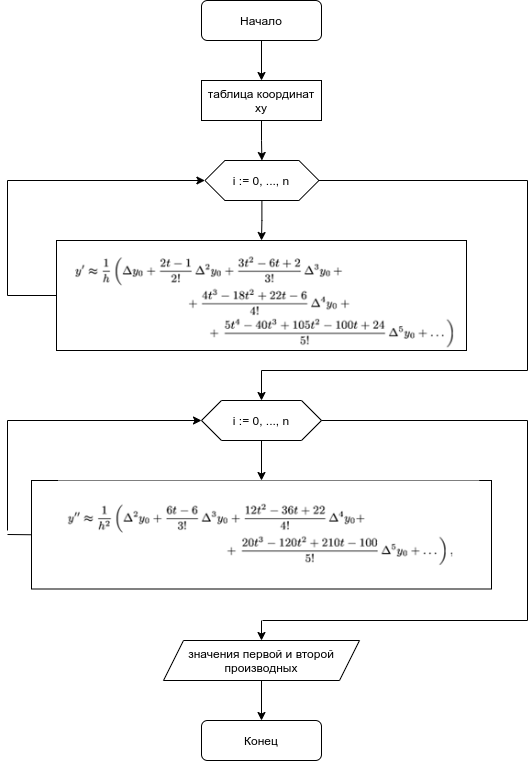
## Метод Ньютона

Предположим, что функция , заданная в виде таблицы с постоянным шагом может быть аппроксимированная интерполяционным многочленом Ньютона:

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

можно получить формулы для вычисления производных любого порядка:

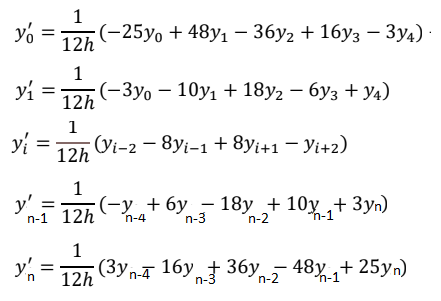


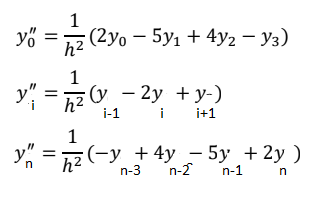


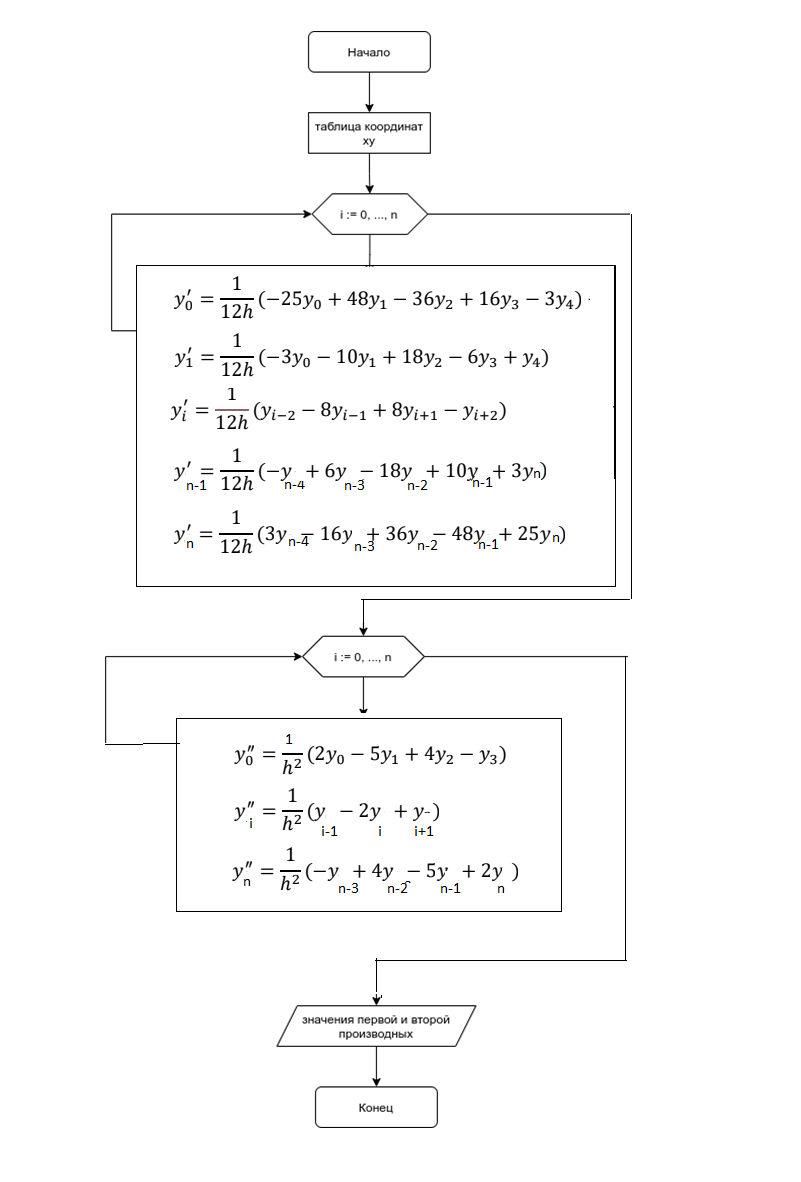
## Метод Лагранжа

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



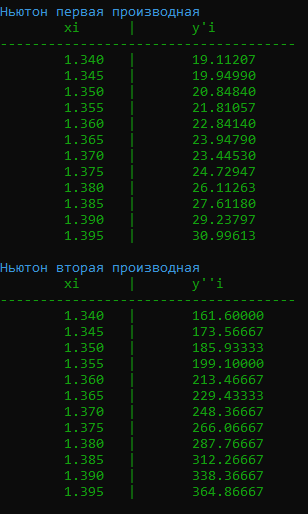


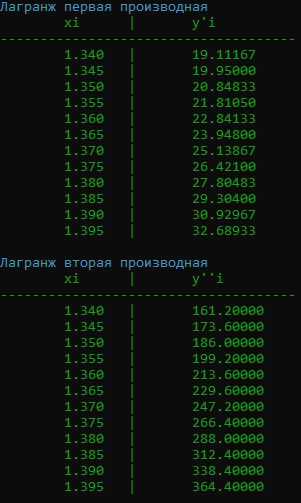




# Расчетные данные

|  |  |
| --- | --- |
| **x** |  |
| 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 |





# 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"

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);

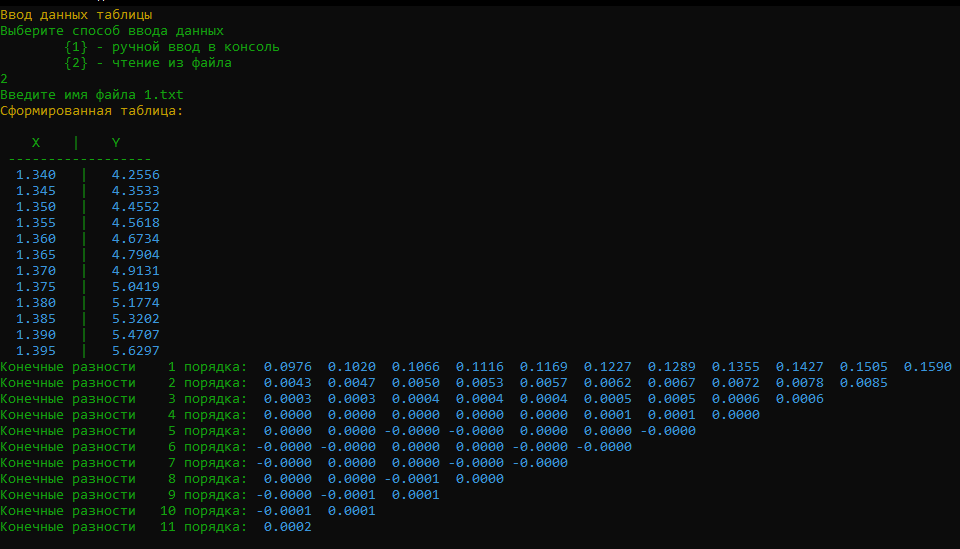
Lagrange\_first\_derivative(table);

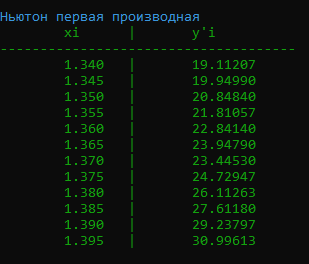
Lagrange\_second\_derivative(table);

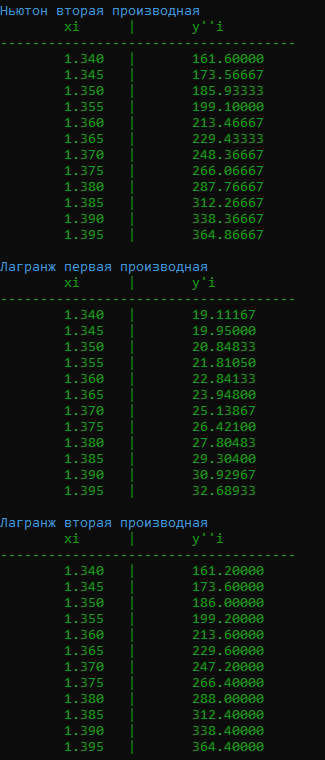
return 0;

}

# Результат работы программы







# 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.