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 №3

«Интерполирование функции многочленом Ньютона и многочленом

Лагранжа»

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](#_Toc68250916)

[Постановка задачи 4](#_Toc68250917)

[Теоретические сведения 5](#_Toc68250918)

[Многочлен ньютона 5](#_Toc68250919)

[Многочлен Лагранжа для неравноотстоящих узлов 7](#_Toc68250920)

[Многочлен Лагранжа для равноотстоящих узлов 9](#_Toc68250921)

[Расчетные данные 11](#_Toc68250922)

[Задание № 1 11](#_Toc68250923)

[Задание № 2 11](#_Toc68250924)

[Листинг разработанной программы 13](#_Toc68250925)

[Value\_function\_table.h 13](#_Toc68250926)

[given\_points.h 14](#_Toc68250927)

[Newton\_Interpolation.h 16](#_Toc68250928)

[Lagrange\_ravn.h 19](#_Toc68250929)

[Lagrange.h 20](#_Toc68250930)

[Main.cpp 21](#_Toc68250931)

[Результаты работы программы 22](#_Toc68250932)

[Вывод 25](#_Toc68250933)

# Цель

Закрепление знаний и умений по интерполированию функций с помощью многочленов Ньютона и Лагранжа

# Постановка задачи

* 1. Вычислить значение функции при данных значениях аргумента,

оценить погрешность:

а) используя первую или вторую интерполяционную формулу Ньютона, в зависимости от значения аргумента;

б) с помощью интерполяционного многочлена Лагранжа, используя формулу для равноотстоящих узлов.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| x | y | № Варианта | Значения аргумента | | | | |
| Х1 | Х2 | Х3 | Х4 | Х5 |
| 0,04 | 6,942768 | 19 | 2,44 | 0,02 | 2,55 | 0,3 | 1,7 |
| 0,24 | 6,663545 |
| 0,44 | 6,39555 |
| 0,64 | 6,138335 |
| 0,84 | 5,891466 |
| 1,04 | 5,654523 |
| 1,24 | 5,427107 |
| 1,44 | 5,20884 |
| 1,64 | 4,999351 |
| 1,84 | 4,79829 |
| 2,04 | 4,605314 |
| 2,24 | 4,420094 |
| 2,44 | 4,278694 |

2. Найти приближенное значение функции при данных значениях аргумента с помощью интерполяционного многочлена Лагранжа, если функция задана в неравноостоящих узлах таблицы, оценить погрешность

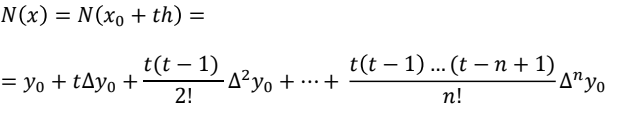
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | y | № Варианта | Х1 | Х2 |
| 0,43 | 1,63597 | 19 | 0,736 | 0,732 |
| 0,48 | 1,73234 |
| 0,55 | 1,87686 |
| 0,62 | 2,03345 |
| 0,70 | 2,22846 |
| 0,75 | 2,35976 |

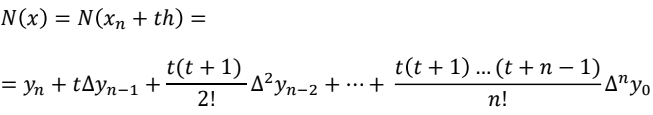
# Теоретические сведения

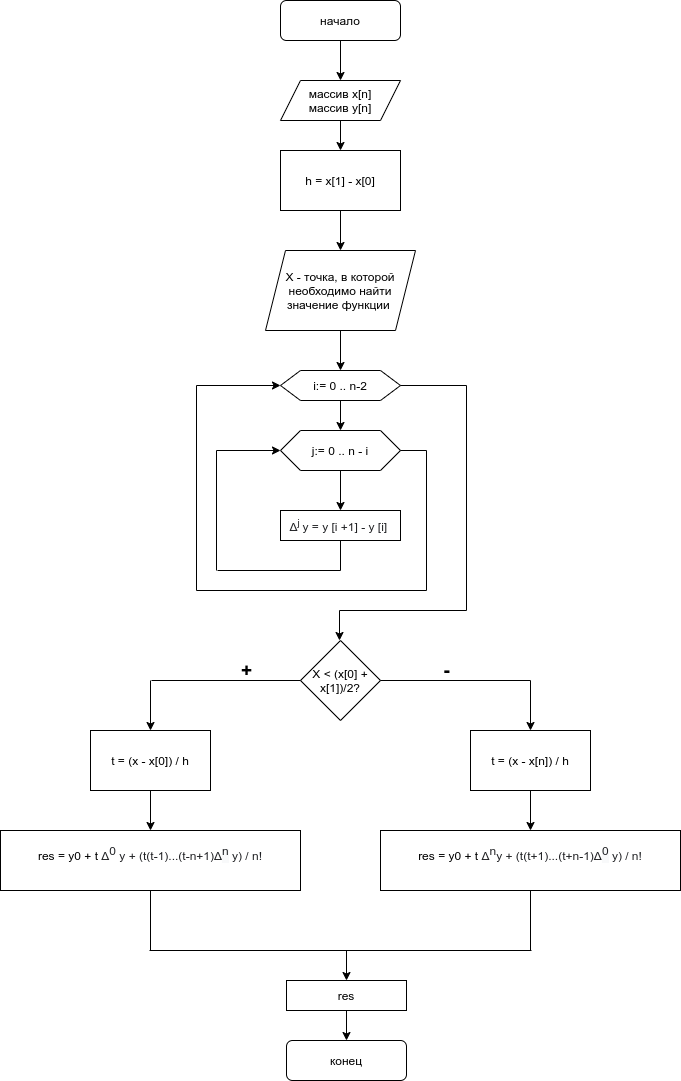
## Многочлен ньютона

Если узлы интерполяции, равноотстоящие и упорядочены по величине, так что , т.е. , то интерполяционный многочлен можно записать в форме Ньютона.

Интерполяционные полиномы в форме Ньютона удобно использовать, если точка интерполирования находится вблизи начала (прямая формула ньютона) или конца таблицы (обратная формула Ньютона).







## Многочлен Лагранжа для неравноотстоящих узлов

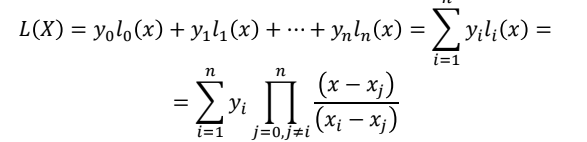
Это многочлен минимальной степени, принимающий данные значения в данном наборе точек. Для пар чисел , где все различны, существует единственный многочлен степени не более n, для которого .

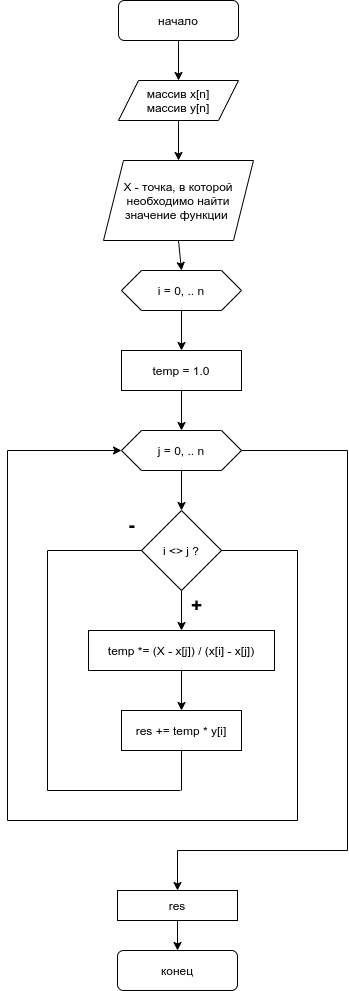
Лагранж предложил способ вычисления таких многочленов:

, где базисные полиномы определяются по формуле:

обладают следующими свойствами:

* Являются многочленами степени n
* при i ≠ j





## Многочлен Лагранжа для равноотстоящих узлов

В случае равномерного распределения узлов интерполяции выражаются через расстояние между узлами интерполяции h и начальную точку :

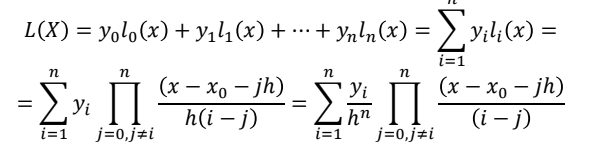
,

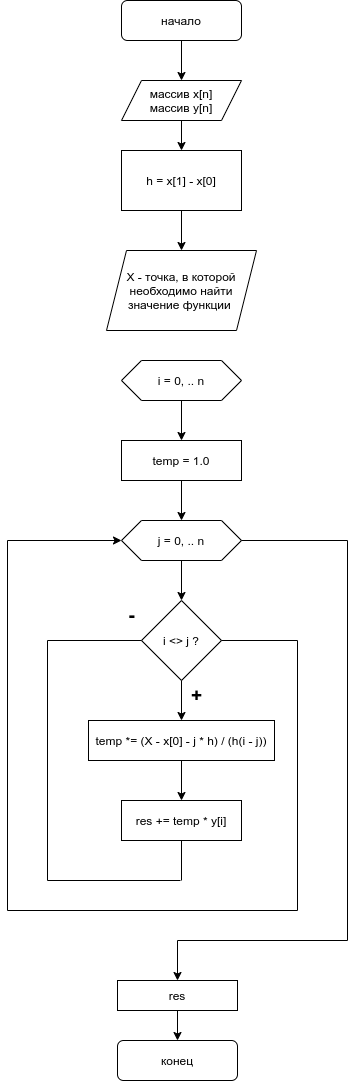
и, следовательно

Подставив эти значения выражения в формулу базисного полинома и вынося h за знакиперемноежения в числителе и знаменателе, получим:

Теперь можно ввести замену переменной:

И получить полином от y, который строится с использованием только целочисленной арифметики. Недостатком данного подхода является факториальная сложность числителя и знаменателя, что требует использование длинной арифметики.



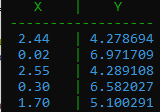


# Расчетные данные

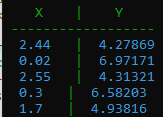
## Задание № 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| x | y | № Варианта | Значения аргумента | | | | |
| Х1 | Х2 | Х3 | Х4 | Х5 |
| 0,04 | 6,942768 | 19 | 2,44 | 0,02 | 2,55 | 0,3 | 1,7 |
| 0,24 | 6,663545 |
| 0,44 | 6,39555 |
| 0,64 | 6,138335 |
| 0,84 | 5,891466 |
| 1,04 | 5,654523 |
| 1,24 | 5,427107 |
| 1,44 | 5,20884 |
| 1,64 | 4,999351 |
| 1,84 | 4,79829 |
| 2,04 | 4,605314 |
| 2,24 | 4,420094 |
| 2,44 | 4,278694 |

Значения, полученные при помощи многочлена Ньютона для равноотстоящих узлов:



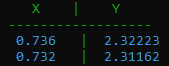
Значения, полученные при помощи многочлена Лагранжа для равноотстоящих узлов:



## Задание № 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | y | № Варианта | Х1 | Х2 |
| 0,43 | 1,63597 | 19 | 0,736 | 0,732 |
| 0,48 | 1,73234 |
| 0,55 | 1,87686 |
| 0,62 | 2,03345 |
| 0,70 | 2,22846 |
| 0,75 | 2,35976 |

Значения, полученные при помощи многочлена Лагранжа для неравноотстоящих узлов:



# Листинг разработанной 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() {

//Включение русского языка в консоли

setlocale(LC\_ALL, "Russian");

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 << Yellow << "\n\tВведите"

<< "координату у " << 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 < x.size(); i++) {

cout << Blue << setprecision(5) << " "

<< this->x[i] << " " << Green

<< " |" << Blue << setw(9) << setprecision(6)

<< this->y[i] << "\n";

}

}

};

## given\_points.h

#ifndef \_given\_points\_

#define \_given\_points\_

#include<vector>

#include<iostream>

#include<fstream>

#include"colors.h"

using namespace std;

class points {

public:

vector<double> x;

size\_t n;

void set\_points() {

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 ";

int k;

double x\_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;

this->x.push\_back(x\_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;

in >> k;

this->n = k;

for (size\_t i = 1; i <= this->n; i++) {

in >> x\_val;

this->x.push\_back(x\_val);

}

is\_readed = true;

}

}

}

//Вывод считанных точек в консоль

cout << Green << "Cчитанные точки\n";

for (size\_t i = 0; i < this->n; i++){

cout << Blue << x[i] << "\n";

}

}

};#endif

## Newton\_Interpolation.h

#ifndef \_Newton\_Interpolation\_

#define \_Newton\_Interpolation\_

#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(10)<<fixed

<< setprecision(6)

<< 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

при помощи интерполяционной формулы Ньютона

Параметры:

1)t - таблица значений функции

2)values - значения х, в которых нужно найти значение функции

\*/

vector<double> Newton(Value\_function\_table table, vector<double> values) {

vector<double> res;

//Нахождение конечных разностей

vector<vector<double>> finiteDifferences;

finiteDifferences = get\_finite\_differences(table);

print\_finite\_differences(finiteDifferences);

//Вычисление середины отрезка переданных X

double mid = (table .x[0] + table.x[(table.n) - 1]) / 2;

//Вычисление шага h

double h = table.x[0] + table.x[1];

//Перемнная для хранения параметра t

double t;

//Нахождение значения функции в каждой переданной точке

for (size\_t k = 0; k < values.size(); k++){

//Переменная для хранения результата

double r = 0;

//Если Xi лежит в промежутке левее середины

//То значение функции вычисляется методом интерполяции

//вперед

if (values[k] < mid){

//t вычисляется как (x - x0)/h

t = (values[k] - table.x[0]) / h;

//К результату прибавляются Y0 + t\*ΔY0

r += table.y[0];

r += t \* finiteDifferences[0][0];

//Вычисление членов (t(t-1)..(t-n+1) \* ΔnY0)/n!

for (int i = 1; i < finiteDifferences.size(); i++){

double temp = 1;

//Промежуточные вычисления числителя

//(t - 1)..(t - n+1)

for (int j = 1; j <= i; j++){

temp \*= (t - j);

}

//К текущему результату добавляем член вида:

//(t(t-1)..(t-n+1) \* ΔnY0)/n!

r += temp \* t \* finiteDifferences[i][0] /

getFact(i+1);

}

//В вектор ответов заносим значение полученное при

//интерполяции вперед

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

//К результату прибавляются Yn + t\*ΔY(n-1)

r+= table.y[table.n-1];

r += t \* finiteDifferences[0]

[finiteDifferences[0].size() -2];

//Вычисление членов

//(t(t+1)..(t+n-1) \* ΔnY(n-1))/n!

for (int i = 1; i < finiteDifferences.size(); i++){

double temp = 1;

//Промежуточные вычисления числителя

//(t + 1)..(t + n-1)

for (int j = 1; j <=i; j++){

temp \*= (t + j);

}

//К текущему результату добавляем член вида:

//(t(t+1)..(t+n-1) \* ΔnY(n-1))/n!

r += t \* temp \* finiteDifferences[i]

[finiteDifferences[i].size()-1]

/ getFact(i+1);

}

//В вектор ответов заносим значение полученное при

//интерполяции назад

res.push\_back(r);

}

}

//Вывод результатов в консоль

cout << Green << "\n X | Y\n"

<<" ------------------\n";

for (int i = 0; i < res.size(); i++){

cout << Blue << setprecision(2)<<" "<< values[i]

<<" "<< Green" |" << Blue << setw(9)

<< setprecision(6) << res[i] << "\n";

}

return res;

}

#endif

## Lagrange\_ravn.h

#ifndef \_Lagrange\_ranv\_

#define \_Lagrange\_ranv\_

#include <vector>

#include <iostream>

#include <iomanip>

#include "Colors.h"

#include "Value\_function\_table.h"

/\*Метод для приближенного вычисления значений

при помощи многочлена Лагранжа для равноостоящих узлов

Параметры:

1)t - таблица значений функции

2)values - значения х, в которых нужно найти значение функции

\*/

vector<double> Lagrange\_ranv(Value\_function\_table table, vector<double> values) {

//Вектор для хранения ответов

vector<double>res;

//Вычисление шага между точками

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

for (size\_t k = 0; k < values.size(); k++) {

//Перемнная для хранения промежуточного результата

double r = 0;

for (int i = 0; i < table.n; i++) {

//Временная переменная для хранения результатов вычислений

double temp = 1;

for (int j = 0; j < table.n; j++) {

//если i = j, то шаг пропускатеся

if (i != j) {

//Вычисление членов произведения вида

//(Х-X0-j\*h)/(h\*(i-j)))

temp \*= (values[k] - table.x[0]-j\*h) /

(double(h \* (i - j)));

}

}

//Полученное произведение умножаем на Yi и добавляем к

//ответу

r += temp \* table.y[i];

}

//Заносим ответ в вектор ответов

res.push\_back(r);

}

//Вывод результатов в консоль

cout << Green << "\n X | Y\n"

<< " ------------------\n";

for (int i = 0; i < res.size(); i++) {

cout << Blue << setprecision(5) << " " << values[i]

<< " " << Green<< " |" << Blue << setw(9)

<< setprecision(6) << 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"

/\*Метод для приближенного вычисления значений

при помощи многочлена Лагранжа для равноостоящих узлов

Параметры:

1)t - таблица значений функции

2)values - значения х, в которых нужно найти значение функции

\*/

vector<double> Lagrange(Value\_function\_table table, vector<double> values) {

//Вектор для хранения ответов

vector<double>res;

//Вычисление шага между точками

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

for (size\_t k = 0; k < values.size(); k++) {

//Перемнная для хранения промежуточного результата

double r = 0;

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

//Временная переменная для хранения результатов вычислений

double temp = 1;

for (size\_t j = 0; j < table.n; j++) {

//если i = j, то шаг пропускатеся

if (i != j) {

//Вычисление членов произведения вида (Хj-X)/(Xj-Xi))

temp \*= (table.x[j]-values[k]) / (table.x[j] –

table.x[i]);

}

}

//Полученное произвдевение умножаем на Yi и добавляем к

//ответу

r += temp \* table.y[i];

}

//Заносим ответ в вектор ответов

res.push\_back(r);

}

//Вывод результатов в консоль

cout << Green << "\n X | Y\n"

<< " ------------------\n";

for (int i = 0; i < res.size(); i++) {

cout << Blue << setprecision(4) << " " << values[i]

<< " " << Green<< " |" << Blue << setw(9)

<< setprecision(6) << res[i] << "\n";

}

return res;

}

#endif

## Main.cpp

#include<iostream>

#include "Value\_function\_table.h"

#include "Newton\_Interpolation.h"

#include "given\_points.h"

#include "Lagrange\_ranv.h"

#include "Lagrange.h"

using namespace std;

int main() {

//Включение русского языка в консоли

setlocale(LC\_ALL, "Russian");

while (true) {

cout << Green << "Для интерполяции функции введите i,"

<< "для завершения программы введите q \n";

char c,m;

cin >> c;

if (c != 'q') {

cout<<Yellow << "Ввод данных таблицы\n";

Value\_function\_table table;

table.set\_value();

points p;

vector<double> r;

cout << Yellow << "Ввод точек, в которых нужно"

<< "найти значение\n";

p.set\_points();

cout<<Green<<"\nВыберите способ интерполяции\n"

<< "\t{1} - формула Ньютона\n"

<< "\t{2} - интерполяционный многочлен"

<< "Лагранжа для равноотстоящих узлов\n"

<< "\t{3} - интерполяционный многочлен"

<< "Лагранжа для неравноотстоящих узлов\n";

cin >> m;

switch (m){

case'1':

r = Newton(table, p.x);

break;

case'2':

r= Lagrange\_ranv(table, p.x);

break;

case'3':

r = Lagrange(table, p.x);

break;

}

}

else {

break;

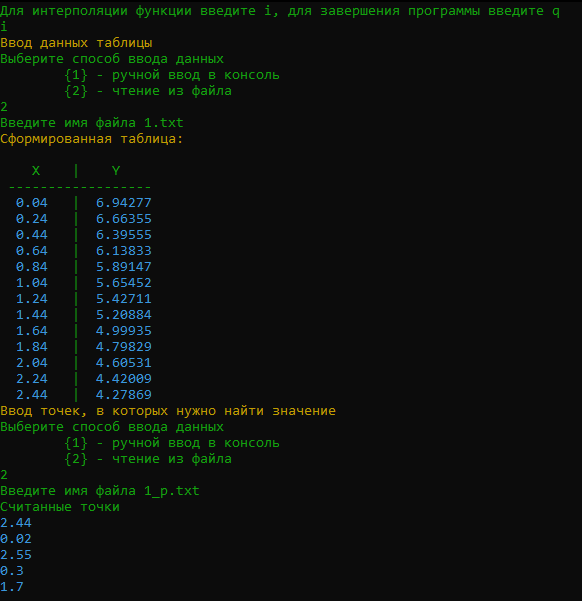
}

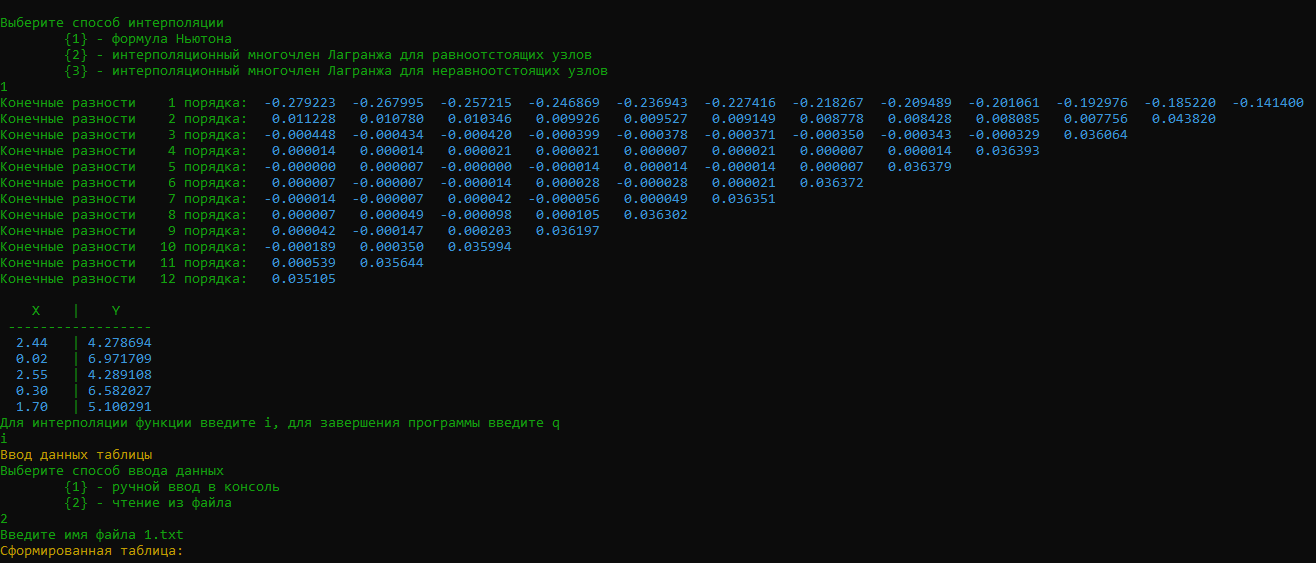
}

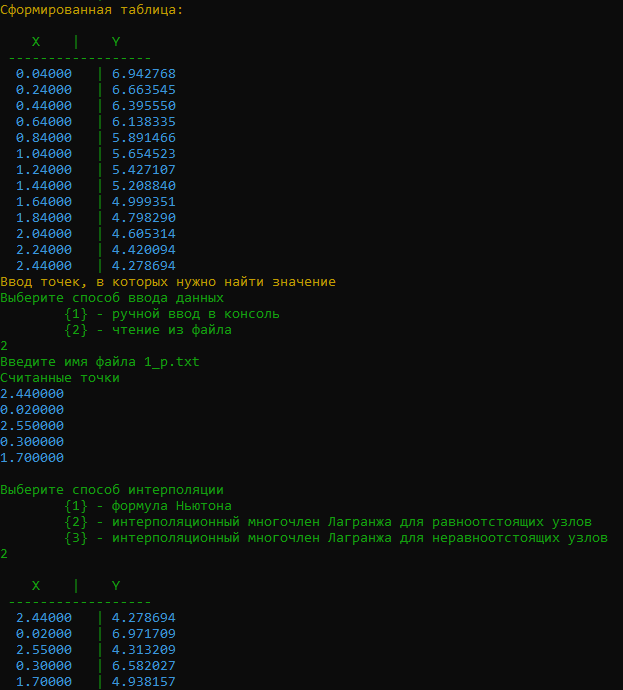
return 0;

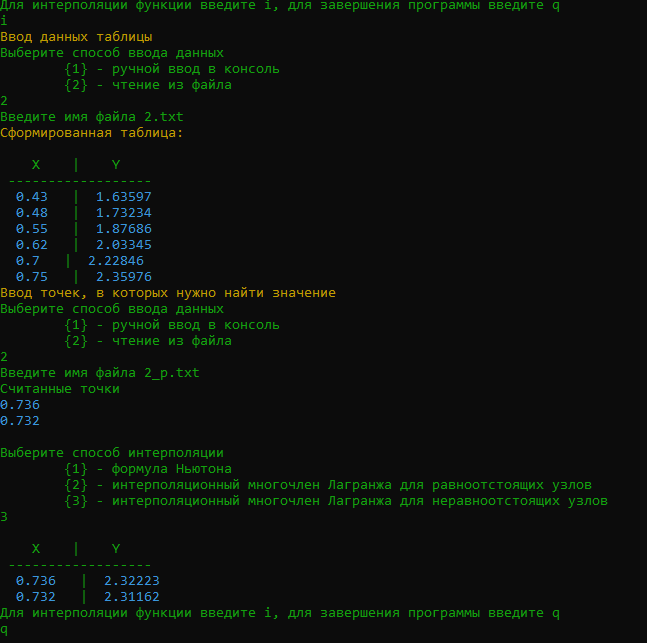
}

# Результаты работы программы









# Вывод

В ходе данной работы были закреплены знания и умения по интерполированию функции с помощью многочленов Ньютона и Лагранжа.