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

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				
			X_1	X_2	X_3	X_4	X_5
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	X_1	X_2
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 $x_{i+1} - x_i = h = const$, т.е. $x_i = x_0 + ih$, то интерполяционный многочлен можно записать в форме Ньютона.

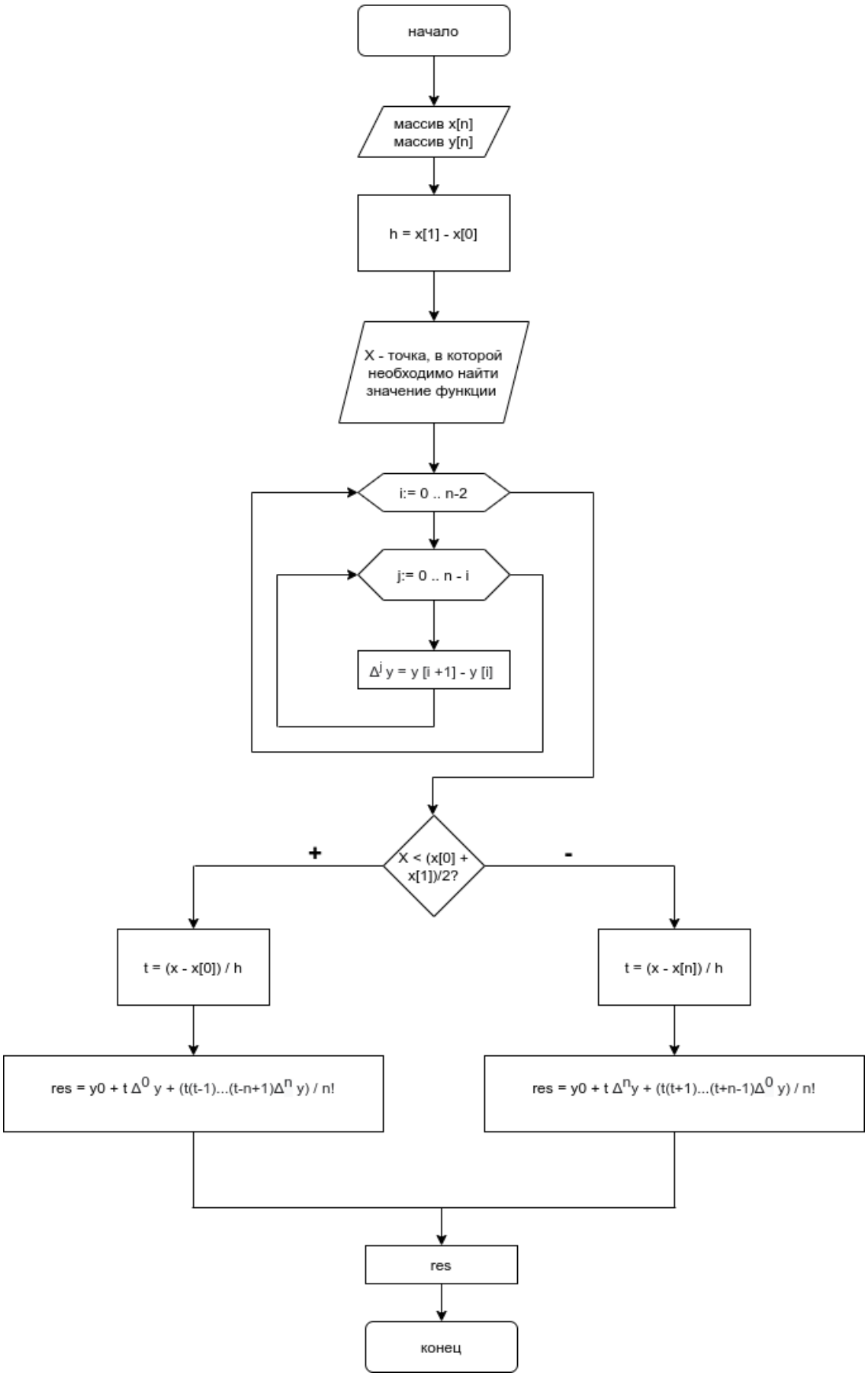
Интерполяционные полиномы в форме Ньютона удобно использовать, если точка интерполирования находится вблизи начала (прямая формула ньютона) или конца таблицы (обратная формула Ньютона).

$$N(x) = N(x_0 + th) =$$

$$= y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_0 + \cdots + \frac{t(t-1)\ldots(t-n+1)}{n!}\Delta^n y_0$$

$$N(x) = N(x_n + th) =$$

$$= y_n + t\Delta y_{n-1} + \frac{t(t+1)}{2!}\Delta^2 y_{n-2} + \cdots + \frac{t(t+1)\ldots(t+n-1)}{n!}\Delta^n y_0$$


начало
массив x[n]
массив y[n]
h = x[1] - x[0]
X - точка, в которой необходимо найти значение функции
i:= 0 .. n-2
j:= 0 .. n - i
Δ^j y = y [i +1] - y [i]
X < (x[0] + x[1])/2?
+
-
t = (x - x[0]) / h
t = (x - x[n]) / h
res = y0 + t Δ^0 y + (t(t-1)...(t-n+1)Δ^n y) / n!
res = y0 + t Δ^n y + (t(t+1)...(t+n-1)Δ^0 y) / n!
res
конец

Многочлен Лагранжа для неравноотстоящих узлов

Это многочлен минимальной степени, принимающий данные значения в данном наборе точек. Для $n+1$ пар чисел $(x_0;\ y_0), (x_1;\ y_1), \dots, (x_n;\ y_n)$, где все x_j различны, существует единственный многочлен $L(x)$ степени не более n, для которого $L(x_j) = y_j$.

Лагранж предложил способ вычисления таких многочленов:

$L(x) = \sum_{i=0}^{n} y_i l_i(x)$, где базисные полиномы определяются по формуле:

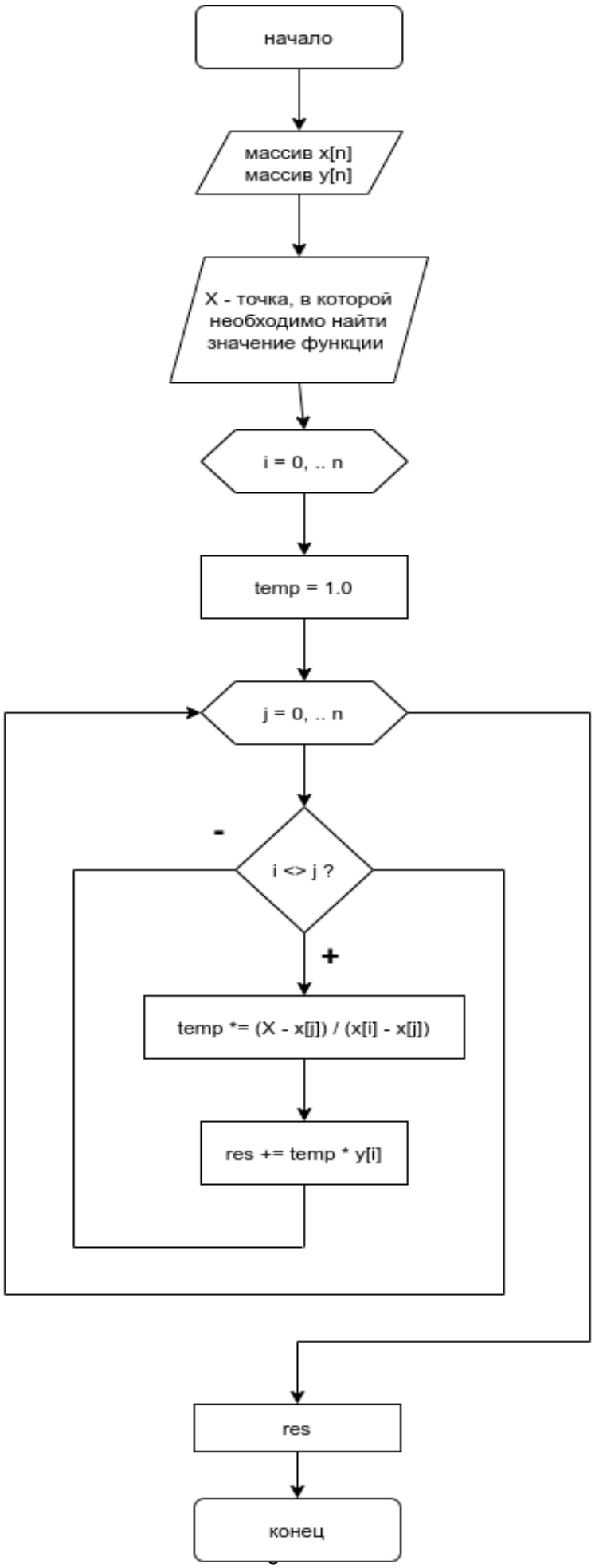
$$l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{(x - x_i)}{(x_i - x_j)} = \frac{(x - x_0)}{(x_i - x_0)} \dots \frac{(x - x_{j-1})}{(x_i - x_{j-1})} \dots \frac{(x - x_n)}{(x_i - x_n)}$$

$l_i(x)$ обладают следующими свойствами:

- Являются многочленами степени n
- $l_i(x_i) = 1$
- $l_i(x_i) = 0$ при $i \neq j$

$$L(X) = y_0 l_0(x) + y_1 l_1(x) + \dots + y_n l_n(x) = \sum_{i=1}^{n} y_i l_i(x) =$$

$$= \sum_{i=1}^{n} y_i \prod_{j=0, j\neq i}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$


начало
массив x[n]
массив y[n]
X - точка, в которой необходимо найти значение функции
i = 0, .. n
temp = 1.0
j = 0, .. n
i <> j ?
-
+
temp *= (X - x[j]) / (x[i] - x[j])
res += temp * y[i]
res
конец

Многочлен Лагранжа для равноотстоящих узлов

В случае равномерного распределения узлов интерполяции x_i выражаются через расстояние между узлами интерполяции h и начальную точку x_0:

$$x_i = x_0 + ih,$$

и, следовательно

$$x_j - x_i = (j - i)h.$$

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

$$l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{(x - x_i)}{(x_i - x_j)} = \frac{\prod_{i=0, i\neq j}^{n}(x - x_0 - ih)}{h^{n-1}\prod_{i=0, i\neq j}^{n}(j - i)}$$

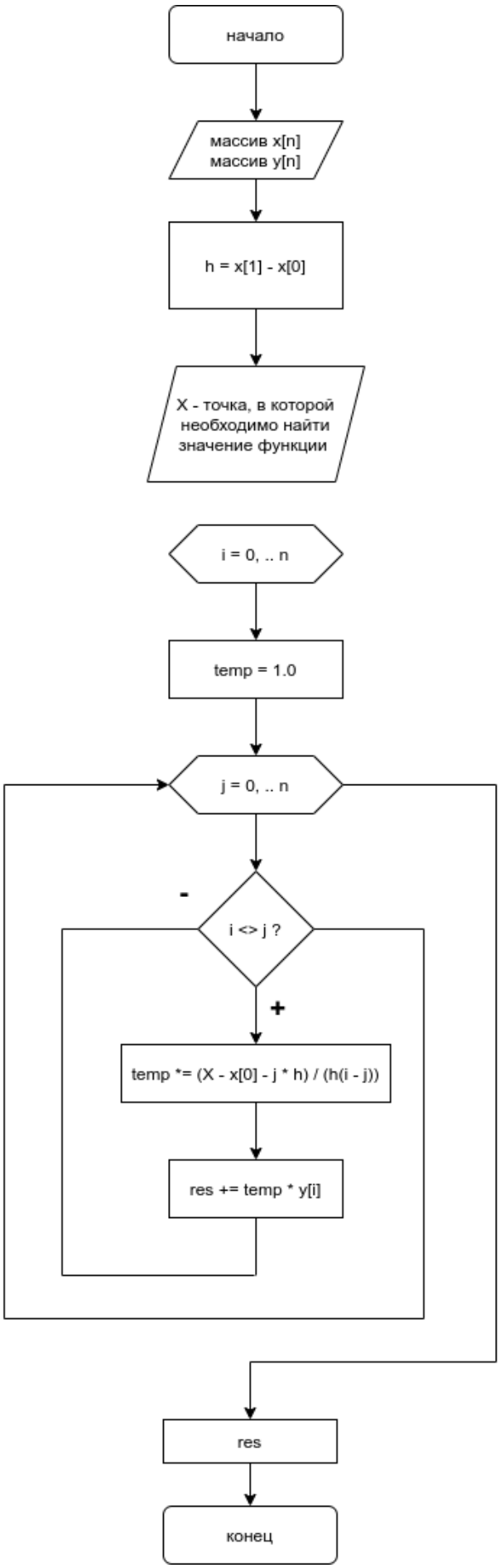
Теперь можно ввести замену переменной:

$$y = \frac{x - x_0}{h}$$

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.

$$L(X) = y_0 l_0(x) + y_1 l_1(x) + \cdots + y_n l_n(x) = \sum_{i=1}^{n} y_i l_i(x) =$$

$$= \sum_{i=1}^{n} y_i \prod_{j=0, j\neq i}^{n} \frac{(x - x_0 - jh)}{h(i - j)} = \sum_{i=1}^{n} \frac{y_i}{h^n} \prod_{j=0, j\neq i}^{n} \frac{(x - x_0 - jh)}{(i - j)}$$


начало
массив x[n]
массив y[n]
h = x[1] - x[0]
X - точка, в которой необходимо найти значение функции
i = 0, .. n
temp = 1.0
j = 0, .. n
-
i <> j ?
+
temp *= (X - x[0] - j * h) / (h(i - j))
res += temp * y[i]
res
конец

Расчетные данные

Задание № 1

x	y	№ Варианта	Значения аргумента				
			X_1	X_2	X_3	X_4	X_5
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:

X	Y
2.44	4.278694
0.02	6.971709
2.55	4.289108
0.30	6.582027
1.70	5.100291

Значения, полученные при помощи многочлена Лагранжа для равноотстоящих узлов:

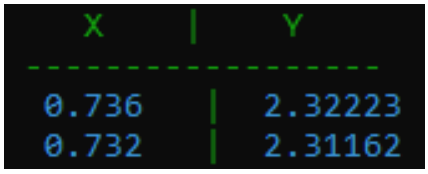
X	Y
2.44	4.27869
0.02	6.97171
2.55	4.31321
0.3	6.58203
1.7	4.93816

Задание № 2

x	y	№ Варианта	X_1	X_2
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 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 < 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 << "Считанные 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 разностей
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 x, 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 x, 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
          //(X-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 x, 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 (Xj-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

```
Для интерполяции функции введите i, для завершения программы введите q
i
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
 ------------------
  0.04   |  6.94277
  0.24   |  6.66355
  0.44   |  6.39555
  0.64   |  6.13833
  0.84   |  5.89147
  1.04   |  5.65452
  1.24   |  5.42711
  1.44   |  5.20884
  1.64   |  4.99935
  1.84   |  4.79829
  2.04   |  4.60531
  2.24   |  4.42009
  2.44   |  4.27869
Ввод точек, в которых нужно найти значение
Выберите способ ввода данных
        {1} - ручной ввод в консоль
        {2} - чтение из файла
2
Введите имя файла 1_p.txt
Считанные точки
2.44
0.02
2.55
0.3
1.7
```

Сформированная таблица:

X	Y
0.04000	6.942768
0.24000	6.663545
0.44000	6.395550
0.64000	6.138335
0.84000	5.891466
1.04000	5.654523
1.24000	5.427107
1.44000	5.208840
1.64000	4.999351
1.84000	4.798290
2.04000	4.605314
2.24000	4.420094
2.44000	4.278694

Ввод точек, в которых нужно найти значение
Выберите способ ввода данных
{1} - ручной ввод в консоль
{2} - чтение из файла
2
Введите имя файла 1_p.txt
Считанные точки
2.440000
0.020000
2.550000
0.300000
1.700000

Выберите способ интерполяции
{1} - формула Ньютона
{2} - интерполяционный многочлен Лагранжа для равноотстоящих узлов
{3} - интерполяционный многочлен Лагранжа для неравноотстоящих узлов
2

X	Y
2.44000	4.278694
0.02000	6.971709
2.55000	4.313209
0.30000	6.582027
1.70000	4.938157

```
Для интерполяции функции введите i, для завершения программы введите q
i
Ввод данных таблицы
Выберите способ ввода данных
        {1} - ручной ввод в консоль
        {2} - чтение из файла
2
Введите имя файла 2.txt
Сформированная таблица:

    X    |    Y
 ------------------
  0.43   |  1.63597
  0.48   |  1.73234
  0.55   |  1.87686
  0.62   |  2.03345
  0.7    |  2.22846
  0.75   |  2.35976
Ввод точек, в которых нужно найти значение
Выберите способ ввода данных
        {1} - ручной ввод в консоль
        {2} - чтение из файла
2
Введите имя файла 2_p.txt
Считанные точки
0.736
0.732

Выберите способ интерполяции
        {1} - формула Ньютона
        {2} - интерполяционный многочлен Лагранжа для равноотстоящих узлов
        {3} - интерполяционный многочлен Лагранжа для неравноотстоящих узлов
3

    X    |    Y
 ------------------
  0.736  |  2.32223
  0.732  |  2.31162
Для интерполяции функции введите i, для завершения программы введите q
q
```

Вывод

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