МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра вычислительные системы и технологии

Лабораторная работа № 3
Интерполирование функции многочленом Ньютона и многочленом Лагранжа
Вариант №15

ОТЧЕТ

по лабораторной работе

по дисциплине

Вычислительная математика

РУКОВОДИТЕЛЬ:

_____ Суркова А.С.

СТУДЕНТ:

_____ Сапожников В.О.

19-ИВТ-3

Работа защищена «___» _____

С оценкой _____

Нижний Новгород 2021

Содержание

1. Цель

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

2. 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
0,101	1,26182	15	0,156	0,12	0,17	0,089	0,144
0,106	1,27644						
0,111	1,29122						
0,116	1,30617						
0,121	1,3213						
0,126	1,3366						
0,131	1,35207						
0,136	1,36773						
0,141	1,38357						
0,146	1,39959						
0,151	1,41579						
0,156	1,43102						
0,161	1,4579						

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

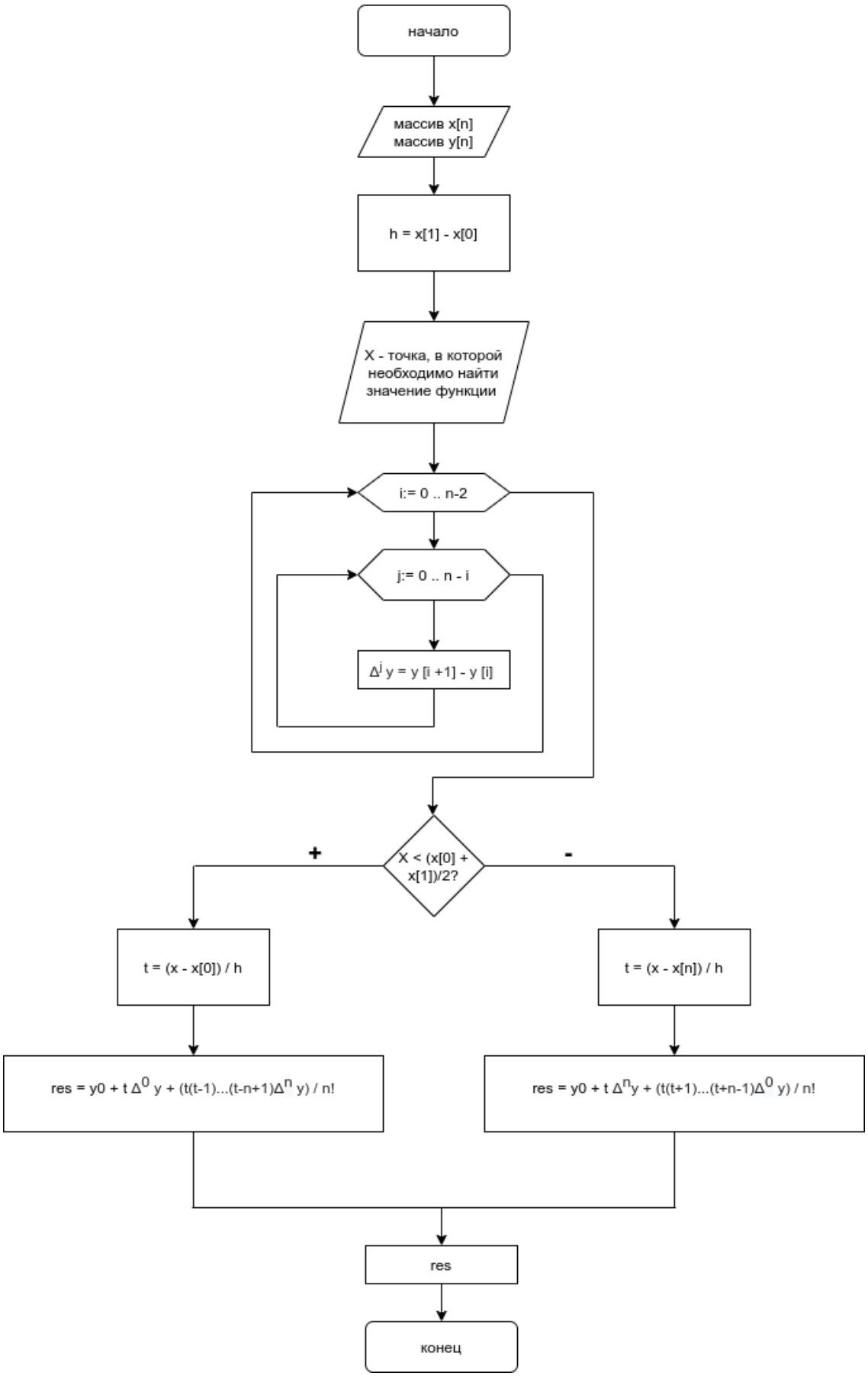
x	y	№ Варианта	X_1	X_2
0,35	2,73951	15	0,482	0,613
0,44	2,30080			
0,47	1,96864			
0,51	1,78776			
0,56	1,59502			
0,64	1,34310			

3. Теоретические сведения

3.1. 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).


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

3.2. 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 $L(x_j) = y_j$.

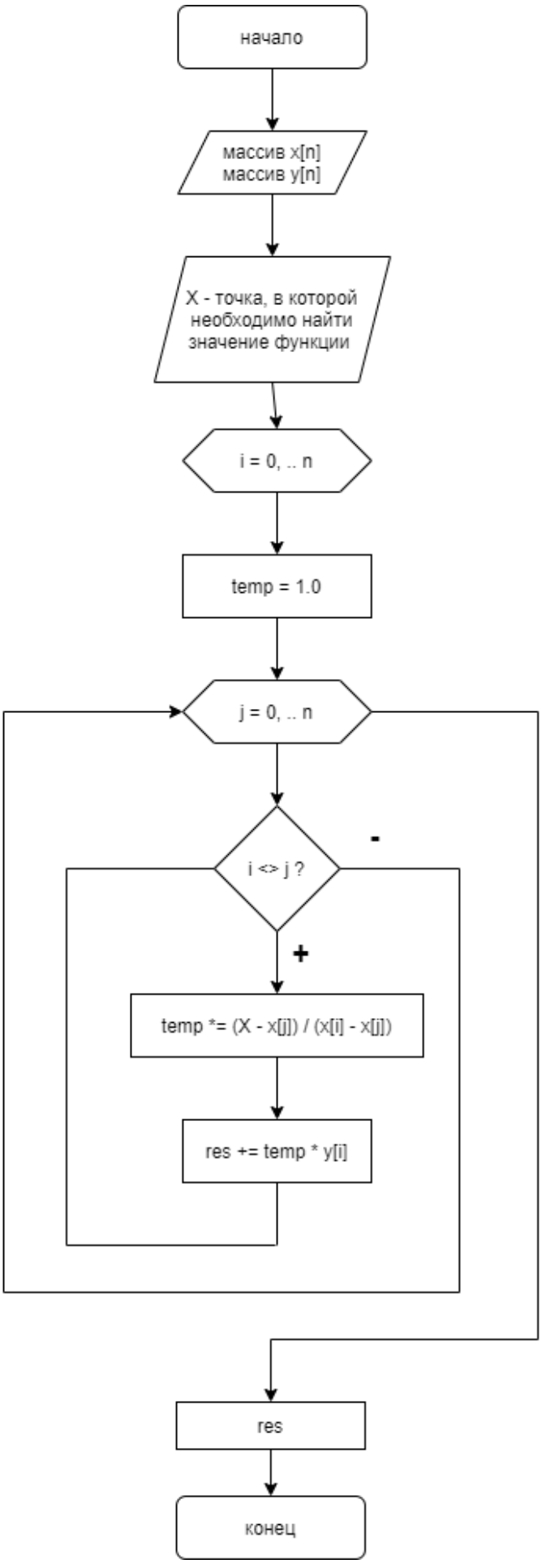
Лагранж предложил способ вычисления таких многочленов:

$L(x) = \sum_{i=0}^{n} y_i l_i(x)$, 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 ≠ 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

3.3. 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

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

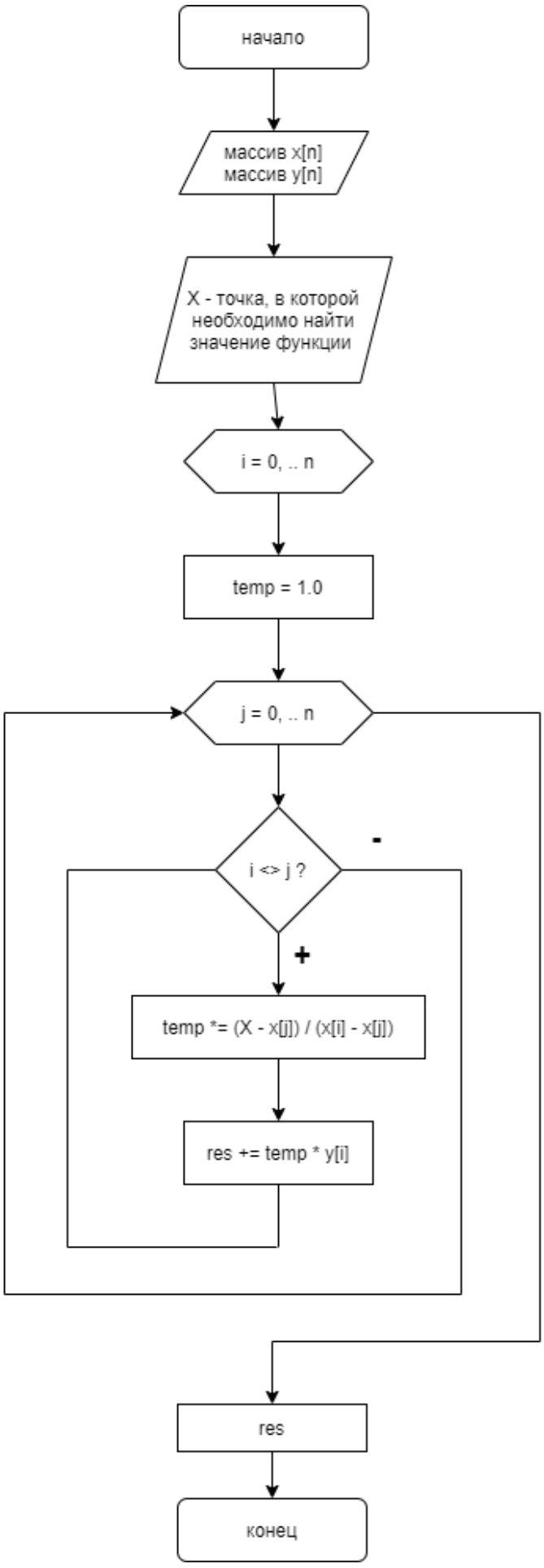
Подставив эти значения выражения в формулу базисного полинома и вынося h за знакиперемноежения в числителе и знаменателе, получим:

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

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.


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

4. Расчетные данные

Задание № 1

x	y	№ Варианта	Значения аргумента				
			X_1	X_2	X_3	X_4	X_5
0,101	1,26182	15	0,156	0,12	0,17	0,089	0,144
0,106	1,27644						
0,111	1,29122						
0,116	1,30617						
0,121	1,3213						
0,126	1,3366						
0,131	1,35207						
0,136	1,36773						
0,141	1,38357						
0,146	1,39959						
0,151	1,41579						
0,156	1,43102						
0,161	1,4579						

Значения, полученные при помощи многочлена Ньютона для равноотстоящих узлов:

X	У
0,156	1,43102
0,12	1,31826
0,17	2,57886
0,089	1,78613
0,144	1,39315

Значения, полученные при помощи многочлена Лагранжа для равноотстоящих узлов:

X	У
0,156	1,43102
0,12	1,31826
0,17	2,57886
0,089	1,78613
0,144	1,39315

Задание № 2

x	y	№ Варианта	X_1	X_2
0,35	2,73951	15	0,482	0,613
0,44	2,30080			
0,47	1,96864			

0,51	1,78776			
0,56	1,59502			
0,64	1,34310			

Значения, полученные при помощи многочлена Лагранжа для неравноотстоящих узлов:

X	У
0,428	2,48481
0,555	1,62586

5. Листинг разработанной программы

Main.java

```
import solution_strategy.*;

import java.util.List;
import java.util.Scanner;

/**
 * Класс, содержащий точку входа в программу - метод main.
 * Язык: java
 *
 * Реализация третьей лабораторной работы по дисциплине: Вычислительная математика
 * Вариант №15
 *
 * Текст задания:
 *  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.
 *
 *                                    Значения аргумента
 *      X    |     Y    | № Варианта |  X1   |  X2   |  X3   | X4    | X5    |
 *    0.101 | 1.26183 |      15    | 0.156 | 0.120 | 0.170 | 0.089 | 0.144 |
 *    0.106 | 1.27644 |            |       |       |       |       |       |
 *    0.111 | 1.29122 |            |       |       |       |       |       |
 *    0.116 | 1.30617 |            |       |       |       |       |       |
 *    0.121 | 1.32130 |            |       |       |       |       |       |
 *    0.126 | 1.33660 |            |       |       |       |       |       |
 *    0.131 | 1.35207 |            |       |       |       |       |       |
 *    0.136 | 1.36773 |            |       |       |       |       |       |
 *    0.141 | 1.38357 |            |       |       |       |       |       |
 *    0.146 | 1.39959 |            |       |       |       |       |       |
 *    0.151 | 1.41579 |            |       |       |       |       |       |
 *    0.156 | 1.43102 |            |       |       |       |       |       |
 *    0.161 | 1.45790 |            |       |       |       |       |       |
 *
 *
 *
 *  2. Найти приближенное значение функции при данных значениях аргумента с помощью
интерполяционного многочлена
 *     Лагранжа, если функция задана в неравноотстоящих узлах таблицы, оценить погрешность
 *
 *                                    Значения аргумента
 *      X  |    Y    | № Варианта |  X1   |  X2   |
```

```
 *    0.35 | 2.73951 |      15    | 0.482 | 0.555 |
 *    0.44 | 2.30080 |            |       |       |
 *    0.47 | 1.96864 |            |       |       |
 *    0.51 | 1.78776 |            |       |       |
 *    0.56 | 1.56502 |            |       |       |
 *    0.64 | 1.34310 |            |       |       |
 *
 *
 * @release: -
 * @last_update: -
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 */
public class Main
{
    //Константы для хранения последовательностей для
    //изменения цвета текста в консоли
    public static final String RESET = "\u001B[0m";
    public static final String RED = "\u001B[31m";
    public static final String PURPLE = "\u001B[35m";
    public static final String CYAN = "\u001B[36m";
    public static final String YELLOW = "\u001B[33m";
    public static final String GREEN = "\u001B[32m";

    /**
     * Точка входа в программу
     * */
    public static void main(String[] args)
    {
        System.out.println("\t\t\t\tЛабораторная работа №3 <<" + PURPLE +"Интерполирование+
                    + функции многочленом Ньютона и многочленом Лагранжа" + RESET + ">>");

        //Открытие потока ввода
        Scanner scanner = new Scanner(System.in);

        //Координаты точек функции задания 1
        double[][] coordinatesTask1 =
                {{0.101, 1.26183},
                {0.106, 1.27644},
                {0.111, 1.29122},
                {0.116, 1.30617},
                {0.121, 1.32130},
                {0.126, 1.33660},
                {0.131, 1.35207},
                {0.136, 1.36773},
                {0.141, 1.38357},
                {0.146, 1.39959},
                {0.151, 1.41579},
```

```
        {0.156, 1.43102},
        {0.161, 1.45790}};

//Xi в которых необходимо найти значения функции для задания 1
double[] desiredValuesTask1 = {0.156, 0.120, 0.170, 0.089, 0.144};


//Координаты точек функции задания 2
double[][] coordinatesTask2 = {{0.35, 2.73951},
                               {0.44, 2.30080},
                               {0.47, 1.96864},
                               {0.51, 1.78776},
                               {0.56, 1.59502},
                               {0.64, 1.34310}};

//Xi в которых необходимо найти значения функции для задания 2
double[] desiredValuesTask2 = {0.428, 0.555};
//Создание ссылки на объект, реализующий интерфейс
//SolutionStrategy
SolutionStrategy strategy;

//Переменная для хранения результата ввода
String ch = "";

List<Double> resList = null;

//Выбор стратегии решения
while (!ch.equals("q"))
{
    System.out.println("Выберите метод:");
    System.out.println("\t1. Интерполяционная формула Ньютона");
    System.out.println("\t2. Многочлен Лагранжа для равноотстоящих узлов");
    System.out.println("\t3. Многочлен Лагранжа для неравноотстоящих узлов");
    System.out.println();
    System.out.println("\tВведите q для выхода");
    System.out.print("Ввод: ");
    ch = scanner.nextLine();
    System.out.println();

    //Засекаем время до начала решения
    double start = System.currentTimeMillis();

    //Ввод с повторением
    switch (ch)
    {
        case ("1") ->
        {
            printConditions(coordinatesTask1, desiredValuesTask1);
```

```
            System.out.println();
            System.out.println("\tВычисление приближенных значений функции при +
                               + помощи CYAN + "\nинтерполяционной формулы Ньютона +
                               + для равноотстоящих узлов\n" + RESET);


            strategy = new NewtonInterpolation();
            resList = strategy.getSolution(coordinatesTask1, desiredValuesTask1);
        }
        case ("2") ->
        {
            printConditions(coordinatesTask1, desiredValuesTask1);
            System.out.println();
            System.out.println("\tВычисление приближенных значений функции при +
                   + помощи " + YELLOW + "\nинтерполяционного многочлена Лагранжа +
                   + для равноотстоящих узлов\n" + RESET);


            strategy = new LagrangianInterpolationEquidistantNodes();
            resList = strategy.getSolution(coordinatesTask1, desiredValuesTask1);
        }
        case ("3") ->
        {
            printConditions(coordinatesTask2, desiredValuesTask2);
            System.out.println();
            System.out.println("\tВычисление приближенных значений функции при +
               + помощи " + GREEN + "\nинтерполяционного многочлена Лагранжа для +
               + неравноотстоящих узлов\n" + RESET);

            strategy = new LagrangianInterpolationUnequallyNodes();
            resList = strategy.getSolution(coordinatesTask2, desiredValuesTask2);
        }
        case ("q") ->
        {
            System.out.println(RED + "Завершение работы..." + RESET);
            System.exit(0);
        }
        default -> System.out.println(RED + "Неверный ввод!" + RESET);
    }

    //Засекаем время после конца решения
    double end = System.currentTimeMillis();

    assert resList != null;
    //Выводим получившиеся ответы
    System.out.println();
    System.out.print(RED + "Ответ: " + RESET);
    for (int i = 0; i < resList.size() - 1; i++) {
```

```
                System.out.printf("y" + (i + 1) + ": %.5f", resList.get(i));
                System.out.print("; ");
            }
            System.out.printf("y" + (resList.size()) + ": %.5f", resList.get(resList.size()
                                                                    - 1));
            System.out.println();

            //Выводим затраченное время для данного решения
            System.out.println("Затраченное время: " + CYAN + (end - start) / 1000.0 +
                                                                + RESET + " секунд");
            System.out.println();
        }
    }

    /**
     * Метод для вывода таблицы заданных точек и значений Xi, в которых
     * необходимо получить значения.
     *
     * @param coordinates   - двумерный массив значений [0][Xi] [1][Yi]
     * @param desiredValues - массив значений Xi, в которых необходимо найти значений
     *                        функций
     * */
    public static void printConditions(double[][] coordinates, double[] desiredValues)
    {
        System.out.println("Заданные точки: ");
        System.out.println("     f(x)");
        System.out.println("  Xi  |    Yi");
        for (double[] coordinate : coordinates)
        {
            System.out.printf("%.3f", coordinate[0]);
            System.out.print(" | ");
            System.out.printf("%.5f", coordinate[1]);
            System.out.println();
        }
        System.out.println();

        System.out.println("Точки в которых необходимо найти значения функции: ");
        for (int i = 0; i < desiredValues.length - 1; i++)
        {
            System.out.print("X" + (i + 1) + ": ");
            System.out.printf("%.5f", desiredValues[i]);
            System.out.print(", ");
        }
        System.out.print("X" + (desiredValues.length) + ": ");
        System.out.printf("%.5f", desiredValues[desiredValues.length - 1]);
        System.out.println();
    }
}
```

solution_strategy/SolutionStrategy.java

```
package solution_strategy;

/**
 * Общий интерфейс всех стратегий решения.
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see LagrangianInterpolationEquidistantNodes
 * @see NewtonInterpolation
 * @see LagrangianInterpolationUnequallyNodes
 * */
public interface SolutionStrategy
{
    /**
     * Метод для вызова той или иной стратегии решения.
     *
     * @param coordinates   - двумерный массив значений [0][Xi] [1][Yi]
     * @param desiredValues - массив значений Xi, в которых необходимо найти значений
функций
     *
     * @return список значений функции в заданных точках.
     * */
    List<Double> getSolution(double[][] coordinates, double[] desiredValues);
}
```

solution_strategy/NewtonInterpolation.java

```
package solution_strategy;

import java.util.LinkedList;
import java.util.List;

/**
 * Класс, реализующий решение при помощи интерполяционной формулы Ньютона
 * для равноотстоящих узлов
 *
 * Реализует интерфейс SolutionStrategy
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SolutionStrategy
 * */
public class NewtonInterpolation implements SolutionStrategy
{
    /**
```

```
 * Вспомогательный метод для вывода конечных разностей i-ого порядка
 * в виде "лестничном виде"
 *
 * @param finiteDifferences - список списков конечных разностей
 * */
private void printFiniteDifferences(List<List<Double>> finiteDifferences)
{
    for (int i = 0; i < finiteDifferences.size(); i++)
    {
        //Если выводимое число будет меньше 10, то задаем дополнительный пробел
        //перед его выводом для равномерного отображения в консоли
        if (i < 9)
        {
            System.out.print("Конечные разности  " + (i+1) + " порядка: ");
        }
        else
        {
            System.out.print("Конечные разности " + (i+1) + " порядка: ");
        }

        for (int j = 0; j < finiteDifferences.get(i).size(); j++)
        {
            //Если число меньше 0 то выводим как оно есть,
            //иначе задаем дополнительный пробле для равномерного вывода в консоль
            if (finiteDifferences.get(i).get(j) < 0)
            {
                System.out.printf("%.5f", finiteDifferences.get(i).get(j));
            }
            else
            {
                System.out.printf(" %.5f", finiteDifferences.get(i).get(j));
            }
            System.out.print(" ");
        }
        System.out.println();
    }
}

/**
 * Метод для вычисления приближенного значения функции при интерполяции вперед
 *
 * @param t                 - значения параметра t, зависящее от координаты X, в
 *                            которой ищется значение
 * @param coordinates       - массив координат заданных точек
 * @param finiteDifferences - список списков конечных разностей
 *
 * @return - искомое значение функции
 * */
```

```
private double directInterpolation(double t, double[][] coordinates,
                                   List<List<Double>> finiteDifferences)
{
    //Переменная для хранения результат
    double res = 0.0;

    //К результату прибавляются Y0 + t*ΔY0
    res += coordinates[0][1];
    res += t * finiteDifferences.get(0).get(0);

    //Вычисление членов (t(t-1)..(t-n+1) * ΔnY0)/n!
    for (int i = 1; i < finiteDifferences.size(); i++)
    {
        double temp = 1.0;

        //Промежуточные вычисления знаменателя (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!
        res += temp * t * finiteDifferences.get(i).get(0)/ getFact((i + 1));
    }

    return res;
}

/**
 * Метод для вычисления приближенного значения функции при интерполяции назад
 *
 * @param t                 - значения параметра t, зависящее от координаты X, в
 *                            которой ищется значение
 * @param coordinates       - массив координат заданных точек
 * @param finiteDifferences - список списков конечных разностей
 *
 * @return - искомое значение функции
 * */
private double reverseInterpolation(double t, double[][] coordinates,
                                    List<List<Double>> finiteDifferences)
{
    //Переменная для хранения результат
    double res = 0.0;

    //К результату прибавляются Yn + t*ΔY(n-1)
    res += coordinates[coordinates.length -1][1];
    res += t * finiteDifferences.get(0).get(finiteDifferences.get(0).size()-1);
```

```
    //Вычисление членов (t(t+1)..(t+n-1) * ΔnY(n-1))/n!
    for (int i = 1; i < finiteDifferences.size(); i++)
    {
        double temp = 1.0;

        //Промежуточные вычисления знаменателя (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!
        res += t * temp * finiteDifferences.get(i).get(finiteDifferences.get(i).size()
                                                              -1)/ getFact((i + 1));
    }

    return res;
}

/**
 * Вспомогательный метод для получения факториала
 *
 * @param n - число, от которого необходимо получить факториал
 *
 * @return - факториал переданного числа
 * */
private 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
 *
 * @param coordinates   - двумерный массив значений [0][Xi] [1][Yi]
 * @param desiredValues - массив значений Xi, в которых необходимо найти значений
 *                        функций
 *
 * @return список значений функции в заданных точках.
```

```
 * */
@Override
public List<Double> getSolution(double[][] coordinates, double[] desiredValues)
{
    List<Double> resList = new LinkedList<>();

    //Создание списка списков для хранения значений конченых разностей
    List<List<Double>> finiteDifferences = new LinkedList<>();

    //Ссылка на временный список для хранения промежуточных значений
    //конечных разностей
    List<Double> tempList;

    //В промежуточный список заносятся конечные разности 1-ого порядка
    tempList = new LinkedList<>();
    for (int i = 0; i < coordinates.length - 1; i++)
    {
        //Вычисление конечных разностей
        tempList.add(coordinates[i + 1][1] - coordinates[i][1]);
    }
    //промежуточный список ханосится в список списков кончеых разностей
    finiteDifferences.add(tempList);

    //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 (int i = 0; i < coordinates.length-2; i++)
    {
        tempList = new LinkedList<>();    //инициализация промежуточного
                                          //списка пустым списком
        for (int j = 0; j < finiteDifferences.get(i).size() - 1; j++)
        {
            //Вычисление конечных разностей
            tempList.add(finiteDifferences.get(i).get(j + 1)
                         - finiteDifferences.get(i).get(j));
        }
        finiteDifferences.add(tempList);
    }

    //Вывод "лестницы" конечных разностей
    printFiniteDifferences(finiteDifferences);

    //Вычисление середины отрезка переданных X
    double midpoint = (coordinates[0][0] + coordinates[coordinates.length - 1][0]) / 2;

    //Вычисление шага h
    double h = coordinates[1][0] - coordinates[0][0];
```

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

        //Для каждого числа из массива Xi, в которых необходимо найти значения функции
        //находим значения функции
        for (double desiredValue : desiredValues)
        {
            //Если Xi лежит в промежутке [x0; (x0 + xn) / 2]
            if (desiredValue < midpoint)
            {
                //t вычисляется как (x - x0)/h
                t = (desiredValue - coordinates[0][0]) / h;

                //В список ответов заносим значение полученное при интерполяции вперед
                resList.add(directInterpolation(t, coordinates, finiteDifferences));
            }
            //Иначе Xi лежит в промежутке [(x0 - xn)/2; xn]
            else
            {
                //t вычисляется как (x - xn)/h
                t = (desiredValue - coordinates[coordinates.length - 1][0]) / h;

                //В список ответов заносим значение полученное при интерполяции назад
                resList.add(reverseInterpolation(t, coordinates, finiteDifferences));
            }
        }

        return resList;
    }

    /**
     * Конструктор без параметров.
     * */
    public NewtonInterpolation()
    {
    }
}
```

solution_strategy/LangraniabInterpolationEquidistantNodes.java

```
package solution_strategy;

import java.util.LinkedList;
import java.util.List;

/**
 * Класс, реализующий решение методом Гаусса.
 * Реализует интерфейс SolutionStrategy
```

```
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SolutionStrategy
 * */
public class LagrangianInterpolationEquidistantNodes implements SolutionStrategy
{
    /**
     * Метод для приближенного вычисления значений
     * при помощи многочлена Лагранжа для равностоящих узлов
     *
     * @param coordinates   - двумерный массив значений [0][Xi] [1][Yi]
     * @param desiredValues - массив значений Xi, в которых необходимо найти значений
     *                        функций
     *
     * @return список значений функции в заданных точках.
     * */
    @Override
    public List<Double> getSolution(double[][] coordinates, double[] desiredValues) {
        //Список для хранения ответов
        List<Double> resList = new LinkedList<>();

        //Вычисление шага между точками
        double h = coordinates[1][0] - coordinates[0][0];

        //Проход по каждому Xi в котором необходимо найти значение функции
        for (double desiredValue : desiredValues)
        {
            //Переменная для хранения промежуточного результата
            double res = 0.0;

            for (int i = 0; i < coordinates.length; i++)
            {
                //Временная переменная для хранения результатов вычислений
                double temp = 1.0;

                for (int j = 0; j < coordinates.length; j++)
                {
                    //если i = j, то шаг пропускается
                    if (i != j)
                    {
                        //Вычисление членов произведения вида (X - Xi -j*h)/(h(i-j))
                        temp *= (desiredValue - coordinates[0][0] - j * h) / (h * (i - j));
                    }
                }
                //Полученное произведение умножаем на Yi и добавляем к ответу
                res += temp * coordinates[i][1];
            }
            //Заносим ответ в список ответов
```

```
            resList.add(res);
        }
        return resList;
    }

    /**
     * Конструктор без параметров.
     * */
    public LagrangianInterpolationEquidistantNodes()
    {
    }
}
```

solution_strategy/ LangraniabInterpolationUnequidistantNodes.java

```
package solution_strategy;

import java.util.LinkedList;
import java.util.List;

/**
 * Класс реализующий решение при помощи многочлена Лагранжа
 * для неравноотстоящих узлов
 *
 * Реализует интерфейс SolutionStrategy
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SolutionStrategy
 * */
public class LagrangianInterpolationUnequallyNodes implements SolutionStrategy
{
    /**
     * Метод для приближенного вычисления значений
     * при помощи многочлена Лагранжа для неравноотстоящих узлов
     *
     * @param coordinates   - двумерный массив значений [0][Xi] [1][Yi]
     * @param desiredValues - массив значений Xi, в которых необходимо найти значений
     *                        функций
     *
     * @return список значений функции в заданных точках.
     * */
    @Override
    public List<Double> getSolution(double[][] coordinates, double[] desiredValues)
    {
        //Список для хранения ответов
        List<Double> resList = new LinkedList<>();
```

```
        //Проход по каждому Xi, в котором необходимо найти значение функции
        for (double desiredValue : desiredValues)
        {
            //Перемнная для хранения промежуточного резлуьтат
            double res = 0.0;

            for (int i = 0; i < coordinates.length; i++)
            {
                //Временная переменная для хранения результатов вычислений
                double temp = 1.0;

                for (int j = 0; j < coordinates.length; j++)
                {
                    //если i = j, то шаг пропускатеся
                    if (i != j)
                    {
                        //Вычисление членов произведения вида (X-Xj)/(Xi-Xj)
                        temp *= (desiredValue - coordinates[j][0]) / (coordinates[i][0] -
                                                    coordinates[j][0]);
                    }
                }

                //Полученное произведение умножаем на Yi и добавляем к ответу
                res += temp * coordinates[i][1];
            }
            //Заносим ответ в список ответов
            resList.add(res);
        }
        return resList;
    }

    /**
     * Конструктор без параметров.
     * */
    public LagrangianInterpolationUnequallyNodes()
    {
    }
}
```

6. Результаты работы программы

```
——— ——— ——— ——— Лабораторная работа №3 <<Интерполирование функции многочленом Ньютона и многочленом Лагранжа>>
Выберите метод:
——— 1. Интерполяционная формула Ньютона
——— 2. Многочлен Лагранжа для равноотстоящих узлов
——— 3. Многочлен Лагранжа для неравноотстоящих узлов

——— Введите q для выхода
Ввод: 1

Заданные точки:
      f(x)
  Xi  |    Yi
0,101 | 1,26183
0,106 | 1,27644
0,111 | 1,29122
0,116 | 1,30617
0,121 | 1,32130
0,126 | 1,33660
0,131 | 1,35207
0,136 | 1,36773
0,141 | 1,38357
0,146 | 1,39959
0,151 | 1,41579
0,156 | 1,43102
0,161 | 1,45790

Точки в которых необходимо найти значения функции:
X1: 0,15600, X2: 0,12000, X3: 0,17000, X4: 0,08900, X5: 0,14400

——— Вычисление приближенных значений функции при помощи
интерполяционной формулы Ньютона для равноотстоящих узлов

Конечные разности  1 порядка:  0,01461  0,01478  0,01495  0,01513  0,01530  0,01547  0,01566  0,01584  0,01602  0,01620  0,01523  0,02688
Конечные разности  2 порядка:  0,00017  0,00017  0,00018  0,00017  0,00017  0,00019  0,00018  0,00018  0,00018 -0,00097  0,01165
Конечные разности  3 порядка:  0,00000  0,00001 -0,00001 -0,00000  0,00002 -0,00001 -0,00000  0,00000 -0,00115  0,01262
Конечные разности  4 порядка:  0,00001 -0,00002  0,00001  0,00002 -0,00003  0,00001  0,00000 -0,00115  0,01377
Конечные разности  4 порядка:  0,00001 -0,00002  0,00001  0,00002 -0,00003  0,00001  0,00000 -0,00115  0,01377
Конечные разности  5 порядка: -0,00003  0,00003  0,00001 -0,00005  0,00004 -0,00001 -0,00115  0,01492
Конечные разности  6 порядка:  0,00006 -0,00002 -0,00006  0,00009 -0,00005 -0,00114  0,01607
Конечные разности  7 порядка: -0,00008 -0,00004  0,00015 -0,00014 -0,00109  0,01721
Конечные разности  8 порядка:  0,00004  0,00019 -0,00029 -0,00095  0,01830
Конечные разности  9 порядка:  0,00015 -0,00048 -0,00066  0,01925
Конечные разности 10 порядка: -0,00063 -0,00018  0,01991
Конечные разности 11 порядка:  0,00045  0,02009
Конечные разности 12 порядка:  0,01964

Ответ: y1: 1,43102; y2: 1,31826; y3: 2,57886; y4: 1,78613; y5: 1,39315
Затраченное время: 0.034 секунд

Выберите метод:
——— 1. Интерполяционная формула Ньютона
——— 2. Многочлен Лагранжа для равноотстоящих узлов
——— 3. Многочлен Лагранжа для неравноотстоящих узлов

——— Введите q для выхода
Ввод: 2

Заданные точки:
      f(x)
  Xi  |    Yi
0,101 | 1,26183
0,106 | 1,27644
0,111 | 1,29122
0,116 | 1,30617
0,121 | 1,32130
0,126 | 1,33660
0,131 | 1,35207
0,136 | 1,36773
0,141 | 1,38357
0,146 | 1,39959
0,151 | 1,41579
```

```
0,146 | 1,39959
0,151 | 1,41579
0,156 | 1,43102
0,161 | 1,45790

Точки в которых необходимо найти значения функции:
X1: 0,15600, X2: 0,12000, X3: 0,17000, X4: 0,08900, X5: 0,14400

—— Вычисление приближенных значений функции при помощи
интерполяционного многочлена Лагранжа для равноотстоящих узлов


Ответ: y1: 1,43102; y2: 1,31826; y3: 2,57886; y4: 1,78613; y5: 1,39315
Затраченное время: 0.022 секунд

Выберите метод:
—— 1. Интерполяционная формула Ньютона
—— 2. Многочлен Лагранжа для равноотстоящих узлов
—— 3. Многочлен Лагранжа для неравноотстоящих узлов

—— Введите q для выхода
Ввод: 3

Заданные точки:
     f(x)
  Xi  |    Yi
0,350 | 2,73951
0,440 | 2,30080
0,470 | 1,96864
0,510 | 1,78776
0,560 | 1,59502
0,640 | 1,34310

Точки в которых необходимо найти значения функции:
X1: 0,42800, X2: 0,55500
0,640 | 1,34310

Точки в которых необходимо найти значения функции:
X1: 0,42800, X2: 0,55500

—— Вычисление приближенных значений функции при помощи
интерполяционного многочлена Лагранжа для неравноотстоящих узлов


Ответ: y1: 2,48481; y2: 1,62586
Затраченное время: 0.027 секунд

Выберите метод:
—— 1. Интерполяционная формула Ньютона
—— 2. Многочлен Лагранжа для равноотстоящих узлов
—— 3. Многочлен Лагранжа для неравноотстоящих узлов

—— Введите q для выхода
Ввод: 0

Неверный ввод!

Ответ: y1: 2,48481; y2: 1,62586
Затраченное время: 0.0 секунд

Выберите метод:
—— 1. Интерполяционная формула Ньютона
—— 2. Многочлен Лагранжа для равноотстоящих узлов
—— 3. Многочлен Лагранжа для неравноотстоящих узлов

—— Введите q для выхода
Ввод:
Ответ: y1: 2,48481; y2: 1,62586
Затраченное время: 0.0 секунд

Выберите метод:
—— 1. Интерполяционная формула Ньютона
—— 2. Многочлен Лагранжа для равноотстоящих узлов
—— 3. Многочлен Лагранжа для неравноотстоящих узлов

—— Введите q для выхода
Ввод: q

Завершение работы...

Process finished with exit code 0
```

7. Вывод

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