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 № 5

Численное дифференцирование функций

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……………………………………………………………………………..3
2. Постановка задачи……………………………………………………………..4
3. Теоретические сведения……………………………………………………….5
   1. Интерполяционный многочлен Ньютона……………………………...5
   2. Формулы Лагранжа……………………………………………………..7
4. Расчётные данные…………………...…………………………………………9
5. Листинг разработанной программы…………………………………………11
6. Результаты работы программы………………………………………………21
7. Вывод………………………………………………………………………….23
8. **Цель**

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

1. **Постановка задачи**

Вычислить первую и вторую производные функции в точках х, заданные таблицей.

15

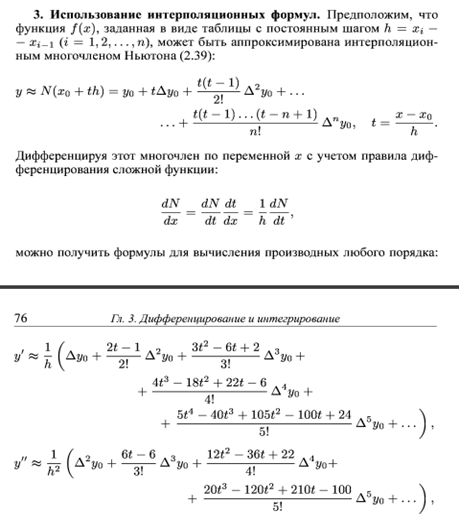
|  |  |
| --- | --- |
| **x** | **y** |
| 3.50 | 33.1154 |
| 3.55 | 34.8133 |
| 3.60 | 36.5982 |
| 3.65 | 38.4747 |
| 3.70 | 40.4473 |
| 3.75 | 42.5211 |
| 3.80 | 44.7012 |
| 3.85 | 46.9931 |
| 3.90 | 49.4024 |
| 3.95 | 51.9354 |
| 4.00 | 54.5982 |
| 4.05 | 57.3975 |
| 4.10 | 60.3403 |
| 4.15 | 63.4340 |
| 4.20 | 66.6863 |

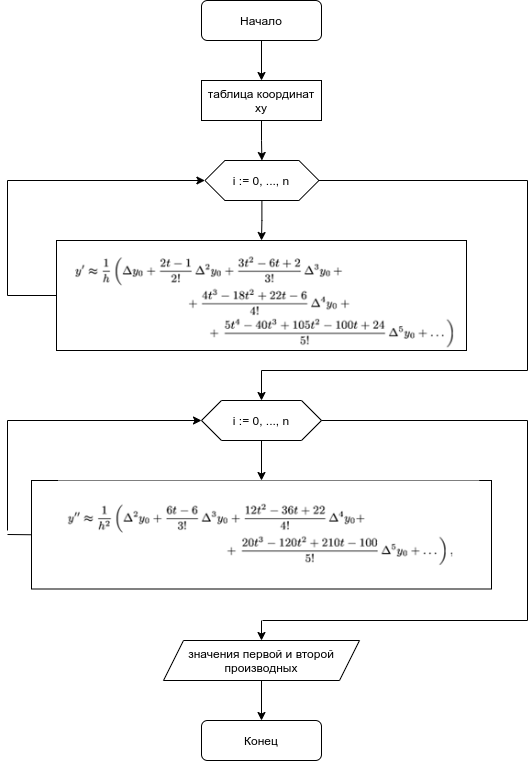
1. **Теоретические сведения**
   1. **Интерполяционный многочлен Ньютона**

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:

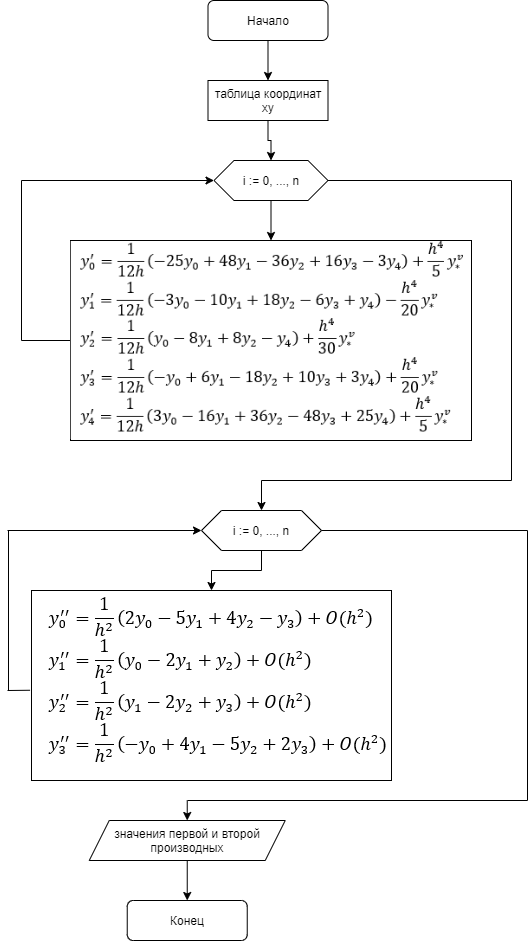




* 1. **Формулы Лагранжа**

Предположим, что функция , заданная в виде таблицы с постоянным шагом может быть аппроксимирована при помощи формул Лагранжа:

Дифференцируя этот многочлен по переменной x, можно получить формулы для вычисления производных любого порядка:



1. **Расчетные данные  
   Многочлен Ньютона**

|  |  |
| --- | --- |
| **x** |  |
| 3.50 | 33.1220 |
| 3.55 | 34.8088 |
| 3.60 | 36.5873 |
| 3.65 | 38.4485 |
| 3.70 | 40.3973 |
| 3.75 | 42.4528 |
| 3.80 | 44.7102 |
| 3.85 | 46.9839 |
| 3.90 | 49.3789 |
| 3.95 | 51.9353 |
| 4.00 | 54.6104 |
| 4.05 | 57.4449 |
| 4.10 | 60.4478 |
| 4.15 | 63.6259 |
| 4.20 | 66.9844 |

|  |  |
| --- | --- |
| **x** |  |
| 3.50 | 32.6900 |
| 3.55 | 34.8267 |
| 3.60 | 36.6433 |
| 3.65 | 38.4200 |
| 3.70 | 40.4367 |
| 3.75 | 42.9733 |
| 3.80 | 44.1000 |
| 3.85 | 46.9933 |
| 3.90 | 49.4867 |
| 3.95 | 51.9880 |
| 4.00 | 54.5833 |
| 4.05 | 57.3867 |
| 4.10 | 60.3500 |
| 4.15 | 63.4333 |
| 4.20 | 66.5967 |

**Формулы Лагранжа**

|  |  |
| --- | --- |
| **x** |  |
| 3.50 | 33.1192 |
| 3.55 | 34.8125 |
| 3.60 | 36.5987 |
| 3.65 | 38.4762 |
| 3.70 | 40.4425 |
| 3.75 | 42.5178 |
| 3.80 | 44.7825 |
| 3.85 | 46.9933 |
| 3.90 | 49.4008 |
| 3.95 | 51.9425 |
| 4.00 | 54.5988 |
| 4.05 | 57.3968 |
| 4.10 | 60.3403 |
| 4.15 | 63.4338 |
| 4.20 | 6636848 |

|  |  |
| --- | --- |
| **x** |  |
| 3.50 | 32.9600 |
| 3.55 | 34.8000 |
| 3.60 | 36.6400 |
| 3.65 | 38.4800 |
| 3.70 | 40.3200 |
| 3.75 | 42.5200 |
| 3.80 | 44.7200 |
| 3.85 | 46.9200 |
| 3.90 | 49.2400 |
| 3.95 | 51.9200 |
| 4.00 | 54.6000 |
| 4.05 | 57.2800 |
| 4.10 | 60.3600 |
| 4.15 | 63.4400 |
| 4.20 | 66.5200 |

1. **Листинг разработанной программы**

**Main.java**

|  |
| --- |
| import solution\_strategy.\*; |
| import java.util.Scanner; |
|  |
| /\*\* |
| \* Класс, содержащий точку входа в программу - метод main. |
| \* Язык: java |
| \* |
| \* Реализация пятой лабораторной работы по диспилине: Вычислительная математика |
| \* Вариант №15 |
| \* |
| \* Текст задания: |
| \* 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). |
| \* |
| \* x y |
| \* 3.50 33.1154 |
| \* 3.55 34.8133 |
| \* 3.60 36.5982 |
| \* 3.65 38.4747 |
| \* 3.70 40.4473 |
| \* 3.75 42.5211 |
| \* 3.80 44.7012 |
| \* 3.85 46.9931 |
| \* 3.90 49.4024 |
| \* 3.95 51.9354 |
| \* 4.00 54.5982 |
| \* 4.05 57.3975 |
| \* 4.10 60.3403 |
| \* 4.15 63.4340 |
| \* 4.20 66.6863 |
| \* |
| \* |
| \* @release: - 03.05.21 |
| \* @last\_update: - 03.05.21 |
| \* |
| \* @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 PURPLE = "\u001B[35m"; |
| public static final String RED = "\u001B[31m"; |
|  |
| /\*\* |
| \* Точка входа в программу |
| \* \*/ |
| public static void main(String[] args) |
| { |
| System.out.println("\t\t\t\tЛабораторная работа №5 <<" + PURPLE + "Численное дифференцирование функций" + |
| " Ньютона и многочленом Лагранжа" + RESET + ">>"); |
|  |
| //Открытие потока ввода |
| Scanner scanner = new Scanner(System.in); |
|  |
| //Таблица значений Вариант №15 |
| double[][] coordinates = {{3.50, 33.1154}, |
| {3.55, 34.8133}, |
| {3.60, 36.5982}, |
| {3.65, 38.4747}, |
| {3.70, 40.4473}, |
| {3.75, 42.5211}, |
| {3.80, 44.7012}, |
| {3.85, 46.9931}, |
| {3.90, 49.4024}, |
| {3.95, 51.9354}, |
| {4.00, 54.5982}, |
| {4.05, 57.3975}, |
| {4.10, 60.3403}, |
| {4.15, 63.4340}, |
| {4.20, 66.6863}}; |
|  |
| //Создание ссылки на объект, реализующий интерфейс |
| //SolutionStrategy |
| SolutionStrategy strategy = null; |
|  |
| //Переменная для хранения результата ввода |
| String ch = ""; |
|  |
| double[][] firstDerivative; |
| double[][] secondDerivative; |
|  |
| //Выбор стратегии решения |
| while (!ch.equals("q")) |
| { |
| System.out.println("Выберите способ нахождения производной:"); |
| System.out.println("\t1. При помощи многочлена Ньютона"); |
| System.out.println("\t2. При помощи многочлена Лагранжа"); |
| System.out.println(); |
| System.out.println("\tВведите q для выхода"); |
| System.out.print("Ввод: "); |
| ch = scanner.nextLine(); |
| System.out.println(); |
|  |
| //Ввод с повторением |
| switch (ch) |
| { |
| case ("1") -> strategy = new NewtonSolution(); |
| case ("2") -> strategy = new LagrangianSolution(); |
| case ("3") -> strategy = new DifferenceForm(); |
| case ("q") -> |
| { |
| System.out.println(RED + "Завершение работы..." + RESET); |
| System.exit(0); |
| } |
| default -> System.out.println(RED + "Неверный ввод!" + RESET); |
| } |
|  |
| assert strategy != null; |
| firstDerivative = strategy.getFirstDerivative(coordinates); |
| secondDerivative = strategy.getSecondDerivative(coordinates); |
|  |
| System.out.println("Значения первой производной: "); |
| for (double[] doubles : firstDerivative) |
| { |
| System.out.printf("%.2f", doubles[0]); |
| System.out.print(" "); |
| System.out.printf("%.4f", doubles[1]); |
| System.out.println(); |
| } |
|  |
| System.out.println(); |
| System.out.println(); |
|  |
| System.out.println("Значения второй производной: "); |
| for (double[] doubles : secondDerivative) |
| { |
| System.out.printf("%.2f", doubles[0]); |
| System.out.print(" "); |
| System.out.printf("%.4f", doubles[1]); |
| System.out.println(); |
| } |
| } |
| } |
| } |

**solution\_strategy/SolutionStrategy.java**

|  |
| --- |
| package solution\_strategy; |
|  |
| /\*\* |
| \* Общий интерфейс всех стратегий решения. |
| \* |
| \* @see DifferenceForm |
| \* @see LagrangianSolution |
| \* @see NewtonSolution |
| \* @author Vladislav Sapozhnikov 19-IVT-3 |
| \* \*/ |
| public interface SolutionStrategy |
| { |
| /\*\* |
| \* Метод для получения первой производной различными способами. |
| \* |
| \* @param coordinates - двумерный массив значений [0][Xi] [1][Yi] |
| \* @return список значений производной в заданных точках. |
| \* \*/ |
| double[][] getFirstDerivative(double[][] coordinates); |
|  |
| /\*\* |
| \* Метод для получения второй производной различными способами. |
| \* |
| \* @param coordinates - двумерный массив значений [0][Xi] [1][Yi] |
| \* @return список значений производной в заданных точках. |
| \* \*/ |
| double[][] getSecondDerivative(double[][] coordinates); |
| } |

**solution\_strategy/NewtonSolution.java**

Начало формы

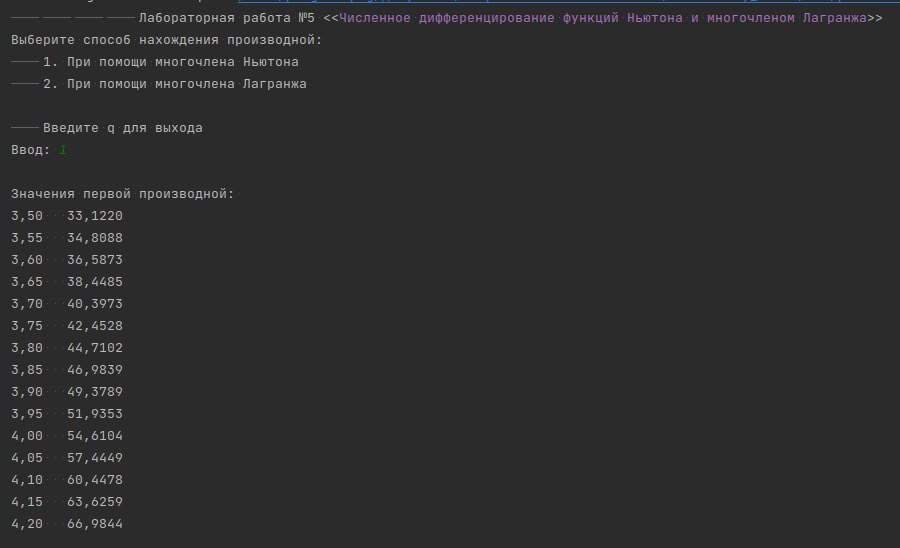
Конец формы

|  |
| --- |
| package solution\_strategy; |
|  |
| import java.util.ArrayList; |
| import java.util.List; |
|  |
| /\*\* |
| \* Класс, в котором реализованы методы нахождения |
| \* первых и вторых произовдных при помощи многочлена Ньютона |
| \* |
| \* @see SolutionStrategy |
| \* \*/ |
| public class NewtonSolution implements SolutionStrategy |
| { |
| /\*\* |
| \* Метод для нахождения конечных приращений |
| \* |
| \* @param coordinates - массив координат для которого необходимо найти |
| \* конечные приращения |
| \* @return список списокв конечных приращений |
| \* \*/ |
| private List<List<Double>> getFiniteDifferences(double[][] coordinates) |
| { |
| //Создание списка списков для хранения значений конченых разностей |
| List<List<Double>> finiteDifferences = new ArrayList<>(); |
|  |
| //Ссылка на временный список для хранения промежуточных значений |
| //конечных разностей |
| List<Double> tempList; |
|  |
| //В промежуточный список заносятся конечные разности 1-ого порядка |
| tempList = new ArrayList<>(); |
| 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 ArrayList<>(); //инициализация промежуточного списка |
| 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); |
| } |
|  |
| return finiteDifferences; |
| } |
|  |
| /\*\* |
| \* 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 получения первых производных при помощи |
| \* многочлена Ньютона |
| \* |
| \* @param coordinates - массив координат точек, в которых |
| \* необходимо найти производные |
| \* @return массив значений первых производных |
| \* \*/ |
| @Override |
| public double[][] getFirstDerivative(double[][] coordinates) |
| { |
| //Вычисление шага |
| double h = coordinates[1][0] - coordinates[0][0]; |
| double[][] resArr = new double[15][2]; |
|  |
| //Нахождение производных для первых 6ти членов |
| //выделяем координаты необходимых точек |
| double[][] tempArr = new double[6][2]; |
| for (int i = 0; i < 6; i++) |
| { |
| tempArr[i][0] = coordinates[i][0]; |
| tempArr[i][1] = coordinates[i][1]; |
| } |
| //Получаем список конечных приращений |
| List<List<Double>> finiteDifference = getFiniteDifferences(tempArr); |
|  |
| //Расчет производных по формуле |
| for (int i = 0; i < 6; i++) |
| { |
| double t = (coordinates[i][0] - coordinates[0][0]) / h; |
|  |
| resArr[i][0] = coordinates[i][0]; |
| resArr[i][1] = (finiteDifference.get(0).get(0) |
| + ((2.0 \* t - 1) \* finiteDifference.get(1).get(0) / getFact(2)) |
| + ((3.0\*t\*t - 6.0\*t + 2) \* finiteDifference.get(2).get(0) / getFact(3) |
| + ((4.0\*t\*t\*t + 18.0\*t\*t + 22.0\*t - 6.0) \*  finiteDifference.get(3).get(0)) / getFact(4)) |
| + ((5.0\*t\*t\*t\*t - 40.0\*t\*t\*t + 105.0\*t\*t -100.0\*t +24.0) \*  finiteDifference.get(4).get(0) / getFact(5))) / h; |
|  |
| } |
|  |
| //Нахождение производных для следующих 3 членов |
| //выделяем координаты необходимых точек |
| tempArr = new double[6][2]; |
| for (int i = 6, j = 0; i < 12; i++, j++) |
| { |
| tempArr[j][0] = coordinates[i][0]; |
| tempArr[j][1] = coordinates[i][1]; |
| } |
| //Получаем список конечных приращений |
| finiteDifference = getFiniteDifferences(tempArr); |
|  |
| //Расчет производных по формуле |
| for (int i = 6; i < 10; i++) |
| { |
| double t = (coordinates[i][0] - coordinates[6][0]) / h; |
|  |
| resArr[i][0] = coordinates[i][0]; |
| resArr[i][1] = (finiteDifference.get(0).get(0) |
| + ((2.0 \* t - 1) \* finiteDifference.get(1).get(0) / getFact(2)) |
| + ((3.0\*t\*t - 6.0\*t + 2) \* finiteDifference.get(2).get(0) / getFact(3) |
| + ((4.0\*t\*t\*t + 18.0\*t\*t + 22.0\*t - 6.0) \*  finiteDifference.get(3).get(0)) / getFact(4)) |
| + ((5.0\*t\*t\*t\*t - 40.0\*t\*t\*t + 105.0\*t\*t -100.0\*t +24.0) \*  finiteDifference.get(4).get(0) / getFact(5))) / h; |
|  |
| } |
|  |
| //Нахождение производных для последних 6ти членов |
| //выделяем координаты необходимых точек |
| tempArr = new double[6][2]; |
| for (int i = 9, j = 0; i < 15; i++, j++) |
| { |
| tempArr[j][0] = coordinates[i][0]; |
| tempArr[j][1] = coordinates[i][1]; |
| } |
| //Получаем список конечных приращений |
| finiteDifference = getFiniteDifferences(tempArr); |
|  |
|  |
| //Расчет производных по формуле |
| for (int i = 9; i < 15; i++) |
| { |
| double t = (coordinates[i][0] - coordinates[9][0]) / h; |
|  |
| resArr[i][0] = coordinates[i][0]; |
| resArr[i][1] = (finiteDifference.get(0).get(0) |
| + ((2.0 \* t - 1) \* finiteDifference.get(1).get(0) / getFact(2)) |
| + ((3.0\*t\*t - 6.0\*t + 2) \* finiteDifference.get(2).get(0) / getFact(3) |
| + ((4.0\*t\*t\*t + 18.0\*t\*t + 22.0\*t - 6.0) \*  finiteDifference.get(3).get(0)) / getFact(4)) |
| + ((5.0\*t\*t\*t\*t - 40.0\*t\*t\*t + 105.0\*t\*t -100.0\*t +24.0) \*  finiteDifference.get(4).get(0) / getFact(5))) / h; |
|  |
| } |
|  |
| return resArr; |
| } |
|  |
| /\*\* |
| \* Метод для получения вторых производных при помощи |
| \* многочлена Ньютона |
| \* |
| \* @param coordinates - массив координат точек, в которых |
| \* необходимо найти производные |
| \* @return массив значений вторых производных |
| \* \*/ |
| @Override |
| public double[][] getSecondDerivative(double[][] coordinates) |
| { |
| //Вычисление шага |
| double h = coordinates[1][0] - coordinates[0][0]; |
| double[][] resArr = new double[15][2]; |
|  |
| //Нахождение производных для первых 6ти членов |
| //выделяем координаты необходимых точек |
| double[][] tempArr = new double[6][2]; |
| for (int i = 0; i < 6; i++) |
| { |
| tempArr[i][0] = coordinates[i][0]; |
| tempArr[i][1] = coordinates[i][1]; |
| } |
| //Получаем список конечных приращений |
| List<List<Double>> finiteDifference = getFiniteDifferences(tempArr); |
|  |
| //Расчет производных по формуле |
| for (int i = 0; i < 6; i++) |
| { |
| double t = (coordinates[i][0] - coordinates[0][0]) / h; |
|  |
| resArr[i][0] = coordinates[i][0]; |
| resArr[i][1] = (finiteDifference.get(1).get(0) |
| + ((6.0 \* t - 6.0) \* finiteDifference.get(2).get(0) / getFact(3)) |
| + ((12.0\*t\*t - 36.0\*t + 22.0) \* finiteDifference.get(3).get(0)  / getFact(4) |
| + ((20.0\*t\*t\*t - 120.0\*t\*t + 210.0\*t - 100.0) \*  finiteDifference.get(4).get(0)) / getFact(5))) / (h\*h); |
|  |
| } |
|  |
|  |
| //Нахождение производных для следующих 3 членов |
| //выделяем координаты необходимых точек |
| tempArr = new double[6][2]; |
| for (int i = 6, j = 0; i < 12; i++, j++) |
| { |
| tempArr[j][0] = coordinates[i][0]; |
| tempArr[j][1] = coordinates[i][1]; |
| } |
| //Получаем список конечных приращений |
| finiteDifference = getFiniteDifferences(tempArr); |
|  |
| //Расчет производных по формуле |
| for (int i = 6; i < 10; i++) |
| { |
| double t = (coordinates[i][0] - coordinates[6][0]) / h; |
|  |
| resArr[i][0] = coordinates[i][0]; |
| resArr[i][1] = (finiteDifference.get(1).get(0) |
| + ((6.0 \* t - 6.0) \* finiteDifference.get(2).get(0) / getFact(3)) |
| + ((12.0\*t\*t - 36.0\*t + 22.0) \* finiteDifference.get(3).get(0)  / getFact(4) |
| + ((20.0\*t\*t\*t - 120.0\*t\*t + 210.0\*t - 100.0) \*  finiteDifference.get(4).get(0)) / getFact(5))) / (h\*h); |
|  |
| } |
|  |
| //Нахождение производных для последних 6ти членов |
| //выделяем координаты необходимых точек |
| tempArr = new double[6][2]; |
| for (int i = 9, j = 0; i < 15; i++, j++) |
| { |
| tempArr[j][0] = coordinates[i][0]; |
| tempArr[j][1] = coordinates[i][1]; |
| } |
| //Получаем список конечных приращений |
| finiteDifference = getFiniteDifferences(tempArr); |
|  |
| //Расчет производных по формуле |
| for (int i = 9; i < 15; i++) |
| { |
| double t = (coordinates[i][0] - coordinates[9][0]) / h; |
|  |
| resArr[i][0] = coordinates[i][0]; |
| resArr[i][1] = (finiteDifference.get(1).get(0) |
| + ((6.0 \* t - 6.0) \* finiteDifference.get(2).get(0) / getFact(3)) |
| + ((12.0\*t\*t - 36.0\*t + 22.0) \* finiteDifference.get(3).get(0)  / getFact(4) |
| + ((20.0\*t\*t\*t - 120.0\*t\*t + 210.0\*t - 100.0) \*  finiteDifference.get(4).get(0)) / getFact(5))) / (h\*h); |
|  |
| } |
|  |
| return resArr; |
| } |
| } |

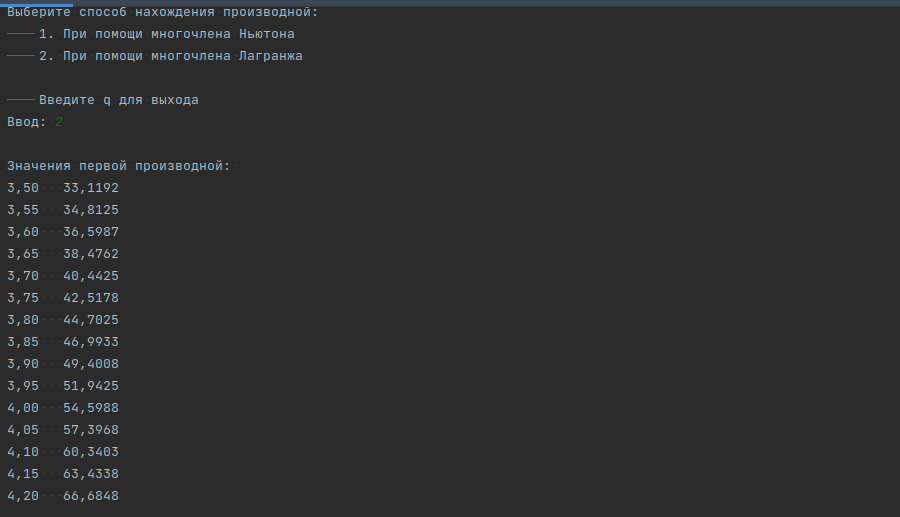
Начало формы

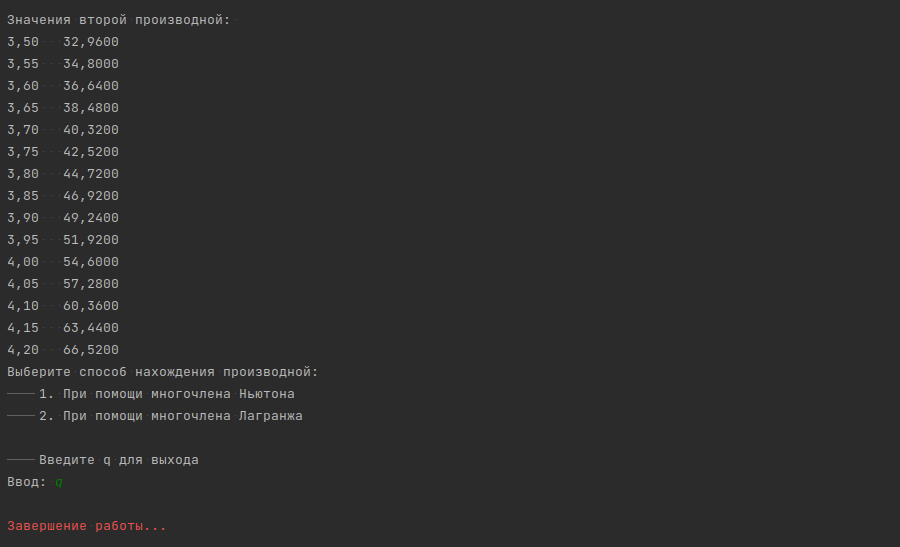
Конец формы

1. **Результаты работы программы**









1. **Вывод**

В ходе данной работы были закреплены знания и умения по вычислению производных первого и второго порядка при помощи интерполяционного многочлена Ньютона формул Лагранжа.