Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа № 5

«Интерполяция функции»

По дисциплине «Вычислительная математика»

Вариант 12

Выполнила:

Студентка группы P3217

Русакова Е.Д.

Преподаватель:

Малышева Т.А.

Санкт-Петербург

2024

Оглавление

Цель работы:

Решить задачу интерполяции, найти значения функции при заданных значениях аргумента, отличных от узловых точек.

Задание:

<u>Обязательное задание (до 80 баллов)</u>

Вычислительная реализация задачи:

1. Выбрать из табл. 1 заданную по варианту таблицу $y = f(x)$ (таблица 1.1 – таблица 1.5);
2. Построить таблицу конечных разностей для заданной таблицы. Таблицу отразить в отчете;
3. Вычислить значения функции для аргумента X_1 (см. табл.1), используя первую или вторую интерполяционную формулу Ньютона. Обратить внимание какой конкретно формулой необходимо воспользоваться;
4. Вычислить значения функции для аргумента X_2 (см. табл. 1), используя первую или вторую интерполяционную формулу Гаусса. Обратить внимание какой конкретно формулой необходимо воспользоваться;
5. ***Подробные вычисления привести в отчете.***

5. Программа должна быть протестирована на различных наборах данных, в том числе и некорректных.
6. Проанализировать результаты работы программы.

<u>Необязательное задание (до 20 баллов)</u>

1. Реализовать в программе вычисление значения функции для заданного значения аргумента, введенного с клавиатуры, используя схемы Стирлинга;
2. Реализовать в программе вычисление значения функции для заданного значения аргумента, введенного с клавиатуры, используя схемы Бесселя.

Программная реализация задачи:

1. Исходные данные задаются тремя способами:

a) в виде набора данных (таблицы x,y), пользователь вводит значения с клавиатуры;

b) в виде сформированных в файле данных (подготовить не менее трех тестовых вариантов);

c) на основе выбранной функции, из тех, которые предлагает программа, например, $\sin x$. Пользователь выбирает уравнение, исследуемый интервал и количество точек на интервале (не менее двух функций).

2. Сформировать и вывести таблицу конечных разностей;
3. Вычислить приближенное значение функции для заданного значения аргумента, введенного с клавиатуры, указанными методами (см. табл. 2). Сравнить полученные значения;
4. Построить графики заданной функции с отмеченными узлами интерполяции и интерполяционного многочлена Ньютона/Гаусса (разными цветами);

Задание для варианта 12:

Для вычислительной реализации задачи:

$x_1 = 0{,}523 \;\; x_2 = 0{,}639$

x	y
0,5	1,532
0,55	2,5356
0,6	3,5406
0,65	4,5462
0,7	5,5504
0,75	6,5559
0,8	7,5594

Рабочие формулы:

Многочлен Лагранжа:

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

Разделенные разности k-го порядка:

$$f(x_i, x_{i+1}, \dots, x_{i+k}) = \frac{f(x_{i+1}, \dots, x_{i+k}) - f(x_i, x_{i+1})}{x_{i+k} - x_i}$$

Многочлен Ньютона с разделенными разностями:

$$N_n(x) = f(x_0) + \sum_{k=1}^{n} f(x_0, x_1, \dots, x_k) * \prod_{j=0}^{k-1} (x - x_j)$$

Конечные разности k-го порядка:

$$\Delta_{y_i}^{k} = \Delta_{y_{i+1}}^{k-1} - \Delta_{y_i}^{k-1}$$

Многочлен Ньютона с конечными разностями для интерполирования вперед:

$$x_i < x < x_{i+1}, \qquad t = \frac{x - x_i}{h}$$

$$N_n(x) = \sum_{k=0}^{n} \frac{\Delta y_i^k}{k!} * \prod_{m=0}^{k-1} (t - m)$$

Многочлен Ньютона с конечными разностями для интерполирования назад:

$$t = \frac{x - x_n}{h}$$

$$N_n(x) = \sum_{k=0}^{n} \frac{\Delta y_{n-k}^k}{k!} * \prod_{m=0}^{k-1}(t+m)$$

Замена переменной:

$$t = \frac{x - x_0}{h} = \frac{x - a}{h}, \text{где } a = x_0 - \text{центральная точка}$$

Многочлен Гаусса: $x > a$

$$P_n(x) = y_0 + \sum_{k=1}^{\frac{n}{2}} \left(\frac{\Delta y_{-(k-1)}^{2k-1}}{(2k-1)!} * \prod_{m=-(k-1)}^{k-1} (t-m) + \frac{\Delta y_{-k}^{2k}}{(2k)!} * \prod_{m=-(k-1)}^{k} (t-m) \right)$$

Многочлен Гаусса: $x < a$

$$P_n(x) = y_0 + \sum_{k=1}^{\frac{n}{2}} \left(\frac{\Delta y_{-k}^{2k-1}}{(2k-1)!} * \prod_{m=-(k-1)}^{k-1} (t-m) + \frac{\Delta y_{-k}^{2k}}{(2k)!} * \prod_{m=-(k-1)}^{k} (t+m) \right)$$

Многочлен Стирлинга: для нечетного числа узлов

$$P_n(x) = y_0 + \sum_{k=1}^{\frac{n}{2}} \left(\prod_{m=1}^{k} (t^2 - m^2) * \left(\frac{t}{(2k-1)!} * \frac{\Delta y_{-k}^{2k-1} + \Delta y_{-(k-1)}^{2k-1}}{2} + \frac{t^2}{(2k)!} * \Delta y_{-k}^{2k} \right) \right)$$

Многочлен Бесселя: для четного числа узлов

$$P_n(x) = \frac{y_0 + y_1}{2} + \left(t - \frac{1}{2}\right) \Delta y_0 + \sum_{k=1}^{\frac{n}{2}-1} \left(\prod_{m=0}^{k} (t-m) * \prod_{m=1}^{k-1} (t+m) * \left(\frac{\Delta y_{-k}^{2k} + \Delta y_{-(k-1)}^{2k}}{2 * (2k)!} + \frac{\left(t - \frac{1}{2}\right) * \Delta y_{-k}^{2k+1}}{(2k+1)!} \right) \right)$$

Вычислительная часть:

Таблица конечных разностей:

i	i для методов Гаусса, Стирлинга, Бесселя	x	y	Δy_i	Δy_i^2	Δy_i^3	Δy_i^4	Δy_i^5	Δy_i^6
0	-3	0,5	1,532	1,0036	0,0014	-0,0008	-0,0012	0,0059	-0,0166
1	-2	0,55	2,5356	1,005	0,0006	-0,002	0,0047	-0,0107	
2	-1	0,6	3,5406	1,0056	-0,0014	0,0027	-0,006		
3	0	0,65	4,5462	1,0042	0,0013	-0,0033			
4	1	0,7	5,5504	1,0055	-0,002				
5	2	0,75	6,5559	1,0035					
6	3	0,8	7,5594						

Многочлен Ньютона

$$x = 0{,}523$$

$$x_0 < x < x_1$$

$$t = \frac{x - x_0}{h} = \frac{0{,}523 - 0{,}5}{0{,}05} = 0{,}46$$

$$N_6(x) = \sum_{k=0}^{n} \frac{\Delta y_i^k}{k!} * \prod_{m=0}^{k-1} (t - m) = \sum_{k=0}^{6} \frac{\Delta y_0^k}{k!} * \prod_{m=0}^{k-1} (0{,}46 - m)$$

$$= y_0 + \frac{\Delta y_0^1}{1!} * t + \frac{\Delta y_0^2}{2!} * t(t-1) + \frac{\Delta y_0^3}{3!} * t(t-1)(t-2)$$

$$+ \frac{\Delta y_0^4}{4!} * t(t-1)(t-2)(t-3) + \frac{\Delta y_0^5}{5!} * t(t-1)(t-2)(t-3)(t-4)$$

$$+ \frac{\Delta y_0^6}{6!} * t(t-1)(t-2)(t-3)(t-4)(t-5)$$

$$= 1{,}532 + \frac{1{,}0036}{1} * 0{,}46 + \frac{0{,}0014}{2} * 0{,}46(0{,}46 - 1) + \frac{-0{,}0008}{6}$$
$$* 0{,}46(0{,}46 - 1)(0{,}46 - 2) + \frac{-0{,}0012}{24} * 0{,}46(0{,}46 - 1)(0{,}46 - 2)(0{,}46 - 3)$$
$$+ \frac{0{,}0059}{120} * 0{,}46(0{,}46 - 1)(0{,}46 - 2)(0{,}46 - 3)(0{,}46 - 4) + \frac{-0{,}0166}{720}$$
$$* 0{,}46(0{,}46 - 1)(0{,}46 - 2)(0{,}46 - 3)(0{,}46 - 4)(0{,}46 - 5)$$
$$= 1{,}532 + 1{,}0036 * 0{,}46 + 0{,}0007 * 0{,}46(-0{,}54) - 0{,}00013 * 0{,}46(-0{,}54)(-1{,}54)$$
$$- 0{,}00005 * 0{,}46(-0{,}54)(-1{,}54)(-2{,}54) + 0{,}00004917$$
$$* 0{,}46(-0{,}54)(-1{,}54)(-2{,}54)(-3{,}54) - 0{,}00002306$$
$$* 0{,}46(0 - 0{,}54)(-1{,}54)(-2{,}54)(-3{,}54)(-4{,}54)$$
$$= 1{,}532 + 1{,}0036 * 0{,}46 + 0{,}0007 * (-0{,}2484) - 0{,}00013 * 0{,}382536 - 0{,}00005$$
$$* (-0{,}97164) + 0{,}00004917 * 3{,}43961 - 0{,}00002306 * (-15{,}61583) =$$
$$= 1{,}532 + 0{,}461656 - 0{,}0001739 - 0{,}0000497 + 0{,}0000486 + 0{,}0001691$$
$$+ 0{,}0003601 = 1{,}9940102$$

Многочлен Гаусса:

$$a = x_0 = 0{,}65$$

$$x = X_2 = 0{,}639$$

$x < a$ ⇒ используем вторую интерполяционную функцию

$$t = \frac{x - a}{h} = \frac{0{,}639 - 0{,}65}{0{,}05} = -\frac{0{,}011}{0{,}05} = -0{,}22$$

$$P_n(x) = y_0 + \sum_{k=1}^{\frac{n}{2}} \left(\frac{\Delta y_{-k}^{2k-1}}{(2k-1)!} * \prod_{m=-(k-1)}^{k-1} (t-m) + \frac{\Delta y_{-k}^{2k}}{(2k)!} * \prod_{m=-(k-1)}^{k} (t+m) \right)$$

$$= y_0 + \Delta y_{-1} * t + \frac{\Delta y_{-1}^2}{2!} * t(t+1) + \frac{\Delta y_{-2}^3}{3!} * (t+1)t(t-1) + \frac{\Delta y_{-2}^4}{4!}$$
$$* (t+2)(t+1)t(t-1) + \frac{\Delta y_{-3}^5}{5!} * (t+2)(t+1)t(t-1)(t-2) + \frac{\Delta y_{-3}^6}{6!}$$
$$* (t+3)(t+2)(t+1)t(t-1)(t-2)$$
$$= 4{,}5462 + 1{,}0056 * (-0{,}22) + \frac{-0{,}0014}{2} * (-0{,}22)(-0{,}22+1) + \frac{-0{,}002}{6}$$
$$* (-0{,}22+1)(-0{,}22)(-0{,}22-1) + \frac{0{,}0047}{24}$$
$$* (-0{,}22+2)(-0{,}22+1)(-0{,}22)(-0{,}22-1) + \frac{0{,}0059}{120}$$
$$* (-0{,}22+2)(-0{,}22+1)(-0{,}22)(-0{,}22-1)(-0{,}22-2) + \frac{-0{,}0166}{720}$$
$$* (-0{,}22+3)(-0{,}22+2)(-0{,}22+1)(-0{,}22)(-0{,}22-1)(-0{,}22-2)$$
$$= 4{,}5462 + 1{,}0056 * (-0{,}22) - 0{,}0007 * (-0{,}22)(0{,}78) - 0{,}003333$$
$$* (0{,}78)(-0{,}22)(-1{,}22) + 0{,}000196 * (1{,}78)(0{,}78)(-0{,}22)(-1{,}22) + 0{,}0000492$$
$$* (1{,}78)(0{,}78)(-0{,}22)(-1{,}22)(-2{,}22) - 0{,}000023056$$
$$* (2{,}78)(1{,}78)(0{,}78)(-0{,}22)(-1{,}22)(-2{,}22)$$
$$= 4{,}5462 + 1{,}0056 * (-0{,}22) - 0{,}0007 * (-0{,}1716) - 0{,}003333 * 0{,}209352$$
$$+ 0{,}000196 * 0{,}37264656 + 0{,}0000492 * (-0{,}8272753632) - 0{,}000023056$$
$$* (-2{,}299825509696)$$
$$= 4{,}5462 - 0{,}221232 + 0{,}00012012 - 0{,}0006978 + 0{,}000073 - 0{,}0000407$$
$$+ 0{,}000053 = 4{,}32447562$$

Программная реализация:

Описание разработанной программы:

Разработанная программа позволяет решить задачу интерполяции разными методами. Пользователь вводит исходные значения точек с клавиатуры или из файла, или выбирает функцию. Затем пользователь выбирает метод, из тех, которые подходят для исходных данных. Программа находит интерполяционную(ые) функцию(ии). Затем пользователь вводит точки, в которых нужно найти значение и программа находит эти значения.Также программа строит графики полученных функций и исходных данных.

Исходный код программы:

Полный код программы выложен на Github и доступен по ссылке [lenapochemy/comp-math-lab5: вычмат лаба 5 интерполяция (github.com)]

Далее приведен код классов, которые отвечают за решение задачи интерполяции.

LagrangePolynomial.java – метод Лагранжа

```
package polynomials;

import utils.Helper;

import java.util.Vector;
```

```java
import java.util.function.DoubleFunction;

public class LagrangePolynomial {

    private final double[] x ,y ;
    private final int n;
    private final Vector<Double> lagrangeX, lagrangeValue;
    private final DoubleFunction<Double> lagrangeFunc;
    double minX, maxX;

    public LagrangePolynomial(double[] x, double[] y){
        this.x = x;
        this.y = y;
        this.n = x.length;
        this.lagrangeFunc = solve();

        minX = x[0];
        maxX = x[n-1];
        lagrangeX = new Vector<>();
        lagrangeValue = new Vector<>();
        double step = 0.1;
        double minH = x[0];
        for(int i = 0; i < n; i++){
            if(minH < x[i]){
                minH = x[i];
            }
        }
        if(minH < step){
            step = minH/2;
        }
        for(double i = minX - 0.1; i < maxX + 0.2; i+= step){
//            System.out.println("x = " + rounding(i) + " y = " +
rounding(lagrangeFunc.apply(i)));
            lagrangeX.add(Helper.rounding(i));
            lagrangeValue.add(Helper.rounding(lagrangeFunc.apply(i)));
        }

    }



    private DoubleFunction<Double> solve(){
        DoubleFunction<Double> interpolationFunc = x -> 0.0;
        for(int i = 0; i < n; i++){
            DoubleFunction<Double> termFunc = x -> 1.0;
            double x_i = x[i];
            for(int j = 0; j < n; j++){
                if(j != i) {
                    double x_j = x[j];
                    DoubleFunction<Double> prevFunc = termFunc;
                    termFunc = x -> prevFunc.apply(x) * ((x - x_j) / (x_i -
x_j));
                }
            }
            DoubleFunction<Double> prevFunc = interpolationFunc;
            double y_i = y[i];
            DoubleFunction<Double> finalTermFunc = termFunc;
            interpolationFunc = x -> prevFunc.apply(x) + y_i *
finalTermFunc.apply(x);
```

```
        }

        return interpolationFunc;
    }


    public Vector<Double> getLagrangeValue() {
        return lagrangeValue;
    }
    public Vector<Double> getLagrangeX(){
        return lagrangeX;
    }

    public DoubleFunction<Double> getLagrangeFunc(){
        return lagrangeFunc;
    }
}
```

NewtonPolynomial.java – метод Ньютона с разделенными разностями

```
package polynomials;

import utils.Helper;

import java.util.Vector;
import java.util.function.DoubleFunction;

public class LagrangePolynomial {

    private final double[] x ,y ;
    private final int n;
    private final Vector<Double> lagrangeX, lagrangeValue;
    private final DoubleFunction<Double> lagrangeFunc;
    double minX, maxX;

    public LagrangePolynomial(double[] x, double[] y){
        this.x = x;
        this.y = y;
        this.n = x.length;
        this.lagrangeFunc = solve();

        minX = x[0];
        maxX = x[n-1];
        lagrangeX = new Vector<>();
        lagrangeValue = new Vector<>();
        double step = 0.1;
        double minH = x[0];
        for(int i = 0; i < n; i++){
            if(minH < x[i]){
                minH = x[i];
            }
        }
        if(minH < step){
            step = minH/2;
        }
        for(double i = minX - 0.1; i < maxX + 0.2; i+= step){
//              System.out.println("x = " + rounding(i) + " y = " +
rounding(lagrangeFunc.apply(i)));
            lagrangeX.add(Helper.rounding(i));
```

```
            lagrangeValue.add(Helper.rounding(lagrangeFunc.apply(i)));
        }

    }



    private DoubleFunction<Double> solve(){
        DoubleFunction<Double> interpolationFunc = x -> 0.0;
        for(int i = 0; i < n; i++){
            DoubleFunction<Double> termFunc = x -> 1.0;
            double x_i = x[i];
            for(int j = 0; j < n; j++){
                if(j != i) {
                    double x_j = x[j];
                    DoubleFunction<Double> prevFunc = termFunc;
                    termFunc = x -> prevFunc.apply(x) * ((x - x_j) / (x_i -
x_j));
                }
            }
            DoubleFunction<Double> prevFunc = interpolationFunc;
            double y_i = y[i];
            DoubleFunction<Double> finalTermFunc = termFunc;
            interpolationFunc = x -> prevFunc.apply(x) + y_i *
finalTermFunc.apply(x);
        }

        return interpolationFunc;
    }


    public Vector<Double> getLagrangeValue() {
        return lagrangeValue;
    }
    public Vector<Double> getLagrangeX(){
        return lagrangeX;
    }

    public DoubleFunction<Double> getLagrangeFunc(){
        return lagrangeFunc;
    }
}
```

GaussPolynomial.java – метод Гаусса

```
package polynomials;

import utils.Helper;
import java.util.Vector;
import java.util.function.DoubleFunction;

public class GaussPolynomial {

    private final double[] x ,y ;
    private final int n;
    private final double a, h;
    private final Vector<Double> gaussY, gaussX;
    private final DoubleFunction<Double> funcXMoreA, funcXLessA;
    private final double[][] finDiff;
```

```
    public GaussPolynomial(double[] x, double[] y, double h, double[][]
finDiff) {
        this.x = x;
        this.y = y;
        this.n = x.length;
        this.h = h;
        this.a = x[n/2];

        this.finDiff = finDiff;

        this.funcXMoreA = solveForXMoreA();
        this.funcXLessA = solveForXLessA();

        gaussY = new Vector<>();
        gaussX = new Vector<>();
        for(double i = x[0] - 0.1; i < x[n-1] + 0.2; i+= 0.1){
            gaussX.add(i);
            if(i < a){
                gaussY.add(funcXLessA.apply(i));
            } else {
                gaussY.add(funcXMoreA.apply(i));
            }
        }

    }

    public double gaussFunc(double num){
        if(num > a){
            System.out.println("Используется первая итнерполяционная формула
Гаусса (x>a)");
            return funcXMoreA.apply(num);
        } else {
            System.out.println("Используется вторая итнерполяционная формула
Гаусса (x>a)");
            return funcXLessA.apply(num);
        }
    }


    private DoubleFunction<Double> solveForXMoreA(){
        DoubleFunction<Double> t = x -> (x - a) / h;

        DoubleFunction<Double> func = x -> y[n/2];
        int count;
        if(n % 2 == 0){
            count = n /2;
        } else {
            count = n/2 + 1;
        }
        for(int k = 1; k < count; k++){
            DoubleFunction<Double> tempFunc = x -> 1.0;

            for(int m = 1-k; m <= k-1; m++){
                DoubleFunction<Double> prevTempFunc = tempFunc;
                int finalM = m;
                tempFunc = x -> prevTempFunc.apply(x) * (t.apply(x) -
finalM);
            }
```

```
            DoubleFunction<Double> prevFunc = func;
            DoubleFunction<Double> finalTempFunc = tempFunc;
            int finalK = k;
            int finalN = n / 2;
            func = x -> prevFunc.apply(x) +
                    finalTempFunc.apply(x) * finDiff[-(finalK-1) +
finalN][2*finalK] / Helper.factorial(2 * finalK -1) +
                    finalTempFunc.apply(x) * (t.apply(x) - finalK) *
finDiff[-finalK + finalN][2 * finalK +1] / Helper.factorial(2 * finalK);

        }
        return func;
    }


    private DoubleFunction<Double> solveForXLessA(){
        DoubleFunction<Double> t = x -> (x - a) / h;

        DoubleFunction<Double> func = x -> y[n/2];
        int count;
        if(n % 2 == 0){
            count = n /2;
        } else {
            count = n/2 + 1;
        }

        for(int k = 1; k < count; k++){
            DoubleFunction<Double> tempFunc1 = x -> 1.0;
            for(int m = 1-k; m <= k-1; m++){
                DoubleFunction<Double> prevTempFunc = tempFunc1;
                int finalM = m;
                tempFunc1 = x -> prevTempFunc.apply(x) * (t.apply(x) -
finalM);
            }
            DoubleFunction<Double> tempFunc2 = x -> 1.0;
            for(int m = 1-k; m <= k; m++){
                DoubleFunction<Double> prevTempFunc = tempFunc2;
                int finalM = m;
                tempFunc2 = x -> prevTempFunc.apply(x) * (t.apply(x) +
finalM);
            }

            DoubleFunction<Double> prevFunc = func;
            DoubleFunction<Double> finalTempFunc1 = tempFunc1;
            DoubleFunction<Double> finalTempFunc2 = tempFunc2;
            int finalK = k;
            int finalN = n / 2;
            func = x -> prevFunc.apply(x) +
                    finalTempFunc1.apply(x) * finDiff[-finalK +
finalN][2*finalK] / Helper.factorial(2 * finalK -1) +
                    finalTempFunc2.apply(x) * finDiff[-finalK + finalN][2 *
finalK +1] / Helper.factorial(2 * finalK);
        }

        return func;
    }

    public Vector<Double> getGaussX() {
        return gaussX;
```

```
    }

    public Vector<Double> getGaussY() {
        return gaussY;
    }

}
```

StirlingPolynomial.java – метод Стирлинга

```
package polynomials;


import utils.Helper;

import java.util.Vector;
import java.util.function.DoubleFunction;

import static java.lang.Math.abs;

public class StirlingPolynomial {

    private final double[] x ,y ;
    private final int n;
    private final double a, h;
    private final double[][] finDiff;
    public final DoubleFunction<Double> stirlingFunc;
    public final Vector<Double> stirlX, stirlY;


    public StirlingPolynomial(double[] x, double[] y, double h, double[][]
finDiff) {
        this.x = x;
        this.y = y;
        this.n = x.length;
        this.h = h;
        this.a = x[n/2];
        this.finDiff = finDiff;
        this.stirlingFunc = solve();

        stirlX = new Vector<>();
        stirlY = new Vector<>();
        double step = 0.1;
        if(h < step){
            step = h/2;
        }
        for(double i = x[0] - 0.1; i < x[n-1] + 0.2; i+= step){
            stirlX.add(i);
            stirlY.add(stirlingFunc.apply(i));
        }
    }

    public boolean checkT(double x){
        double t = (x - a) / h;
        return abs(t) <= 0.25;
    }
```

```
    public DoubleFunction<Double> solve(){
        DoubleFunction<Double> t = x -> (x - a) / h;
        DoubleFunction<Double> func = x -> y[n/2];

        for(int k = 1; k < n/2+1; k++){
            DoubleFunction<Double> tempFunc = x -> 1.0;

            for(int m = 1; m <= k-1; m++){
                DoubleFunction<Double> prevTempFunc = tempFunc;
                int finalM = m;
                tempFunc = x -> prevTempFunc.apply(x) * (Math.pow(t.apply(x),
2) - Math.pow(finalM, 2));
            }

            DoubleFunction<Double> prevFunc = func;
            DoubleFunction<Double> finalTempFunc = tempFunc;
            int finalK = k;
            int finalN = n / 2;
            func = x -> prevFunc.apply(x) +
                    finalTempFunc.apply(x) * (t.apply(x) * (finDiff[-finalK +
finalN][2*finalK] + finDiff[-(finalK-1) + finalN][2*finalK]) / (2 *
Helper.factorial(2 * finalK - 1 ) ) +
                     Math.pow(t.apply(x), 2) * finDiff[-finalK +
finalN][2*finalK + 1] / Helper.factorial(2 * finalK));
        }

        return func;
    }


}
```

BesselPolynomial.java – метод Бесселя

```
package polynomials;

import utils.Helper;

import java.util.Vector;
import java.util.function.DoubleFunction;

import static java.lang.Math.abs;

public class BesselPolynomial {

    private final double[] x ,y ;
    private final int n;
    private final double a, h;
    private final double[][] finDiff;
    public final DoubleFunction<Double> besselFunc;
    public final Vector<Double> besselX, besselY;


    public BesselPolynomial(double[] x, double[] y, double h, double[][]
finDiff) {
        this.x = x;
        this.y = y;
        this.n = x.length;
        this.h = h;
        this.a = x[n/2 - 1];
```

```
        this.finDiff = finDiff;
        this.besselFunc = solve();

        double step = 0.1;
        if(h < step){
            step = h/2;
        }
        besselX = new Vector<>();
        besselY = new Vector<>();
        for(double i = x[0] - 0.1; i < (x[n-1] + 0.2); i+= step){
            besselX.add(i);
            besselY.add(besselFunc.apply(i));
        }
    }

    public boolean checkT(double x){
        double t = (x - a) / h;
        return abs(t) >= 0.25 && abs(t) <= 0.75;
    }

    public DoubleFunction<Double> solve(){
        DoubleFunction<Double> t = x -> (x - a) / h;
        DoubleFunction<Double> func = x -> (y[n/2 - 1] + y[n/2])/2 +
(t.apply(x) - 0.5) * finDiff[n/2-1][2];

        for(int k = 1; k < n/2; k++){
            DoubleFunction<Double> tempFunc = x -> 1.0;

            for(int m = 0; m <= k; m++){
                DoubleFunction<Double> prevTempFunc = tempFunc;
                int finalM = m;
                tempFunc = x -> prevTempFunc.apply(x) * (t.apply(x) -
finalM);
            }
            for(int m = 1; m <= k - 1; m++){
                DoubleFunction<Double> prevTempFunc = tempFunc;
                int finalM = m;
                tempFunc = x -> prevTempFunc.apply(x) * (t.apply(x) +
finalM);
            }

            DoubleFunction<Double> prevFunc = func;
            DoubleFunction<Double> finalTempFunc = tempFunc;
            int finalK = k;
            int finalN = n / 2 - 1;

            func = x -> prevFunc.apply(x) +
                    finalTempFunc.apply(x) * ( (finDiff[-finalK +
finalN][2*finalK + 1] + finDiff[-(finalK-1) + finalN][2*finalK + 1]) / (2 *
Helper.factorial(2 * finalK) ) +
                            (t.apply(x) - 0.5) * finDiff[-finalK +
finalN][2*finalK + 2] / Helper.factorial(2 * finalK + 1) );
        }

        return func;
    }
```

```
}
```

Helper.java – класс с расчетом таблицы конечных разностей

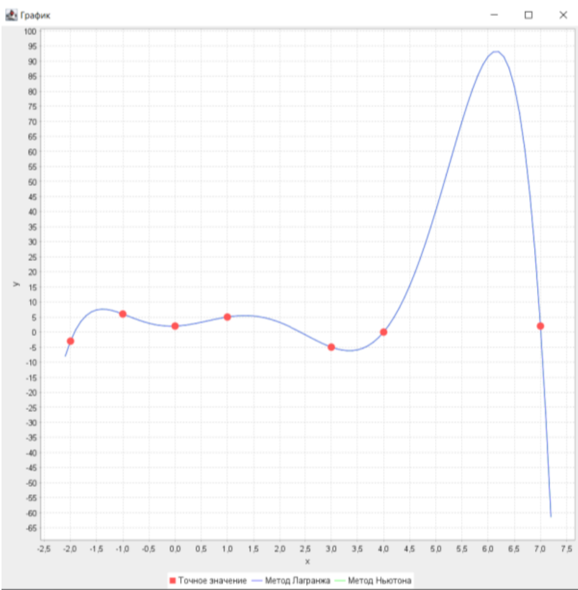
```
package utils;

import java.math.BigDecimal;
import java.math.RoundingMode;

public class Helper {

    public static double checkDiffInterval(double[] x){
        int n = x.length;
        double diff = rounding(Math.abs(x[0] - x[1]));
        for(int i = 1; i < n-1; i++){
            if(rounding(Math.abs(x[i] - x[i+1])) != diff){
                return -1;
            }
        }
        return diff;
    }


    public static double[][] finiteDiffs(double[] x, double[] y){
        int n = x.length;
        double[][] finDiff = new double[n][n+1];
        for(int i = 0; i < n; i++){
            finDiff[i][0] = x[i];
            finDiff[i][1] = y[i];
        }
        int m = n-1;
        for(int j = 2; j < n+1; j++, m--){
            for(int i = 0; i < m; i++){
                finDiff[i][j] =  finDiff[i+1][j-1] - finDiff[i][j-1];
//                  System.out.println("ij = " + finDiff[i][j] + " i1j_1 = " +
finDiff[i+1][j-1] + " ij_1 = " + finDiff[i][j-1]);
            }
        }
        printFiniteDiffs(finDiff, n);

        return finDiff;
    }

    private static void printFiniteDiffs(double[][] finDiff, int n){
        System.out.println("Таблица конечных разностей");
        String res =  String.format(" %-3s|", "i");
        res += String.format(" %-3s|", " ");
        res += String.format(" %-10s|", "x");
        res += String.format(" %-10s|", "y");
        for(int i = 2; i < n; i++){
            if(i == 2){
                res += String.format(" %-10s|" , ("Δy_i"));
            }
            res += String.format(" %-10s|" , ("Δy^" + (i) + " _i"));
        }

        System.out.println(res);
        int m = n+1;
        int s;
```

```
        if(n % 2 == 0){
            s = n/2 - 1;
        } else s = n/2;
        for(int i = 0; i < n; i++, m--){
            res = String.format(" %-3s|", i);
            res += String.format(" %-3s|", i - s);
            for(int j = 0; j < m; j++){
                String str;
                if(rounding(finDiff[i][j]) >= 0) {
                    str = " " + rounding(finDiff[i][j]);
                } else str = Double.toString(rounding(finDiff[i][j]));
                res += String.format(" %-10s|", str);
//                System.out.print(finDiff[i][j] + " ");
            }
            System.out.println(res);
        }

    }

    public static double rounding(double number){
        BigDecimal help = new BigDecimal(number);
        help = help.setScale(6, RoundingMode.HALF_UP);
        return help.doubleValue();
    }



    public static int factorial(int num){
        if(num <= 1) return 1;
        return num * factorial(num - 1);
    }


}
```

Примеры работы программы:

Пример 1

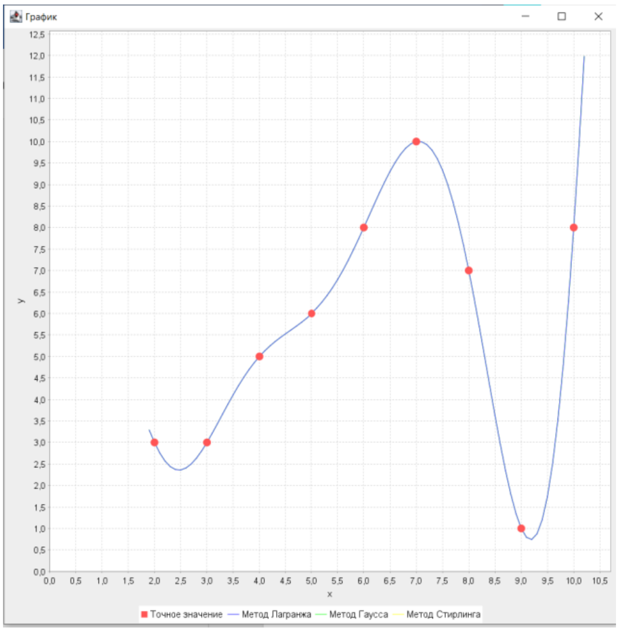
```
Выберите один вариант:
    1. Ввод данных точками
    2. Ввод данных точками через интервалами
    3. Ввод данных через функцию
1
Выберите один вариант:
    1. Ввод данных из файла
    2. Ввод данных с клавиатуры
1
Введите путь к файлу:
C:\Users\Elena\IdeaProjects\compMath\lab5\src\files\data1
Введите количество точек: 7
Введите исходные точки парами (x, y) через пробел
-2 -3
-1 6
0 2
1 5
7 2
3 -5
4 0
Выберите метод
    1. Многочлен Лагранжа
    2. Многочлен Ньютона с разделенными разностями
    3. Все методы
3
```



```
Введите значение числа для вычислений : 4.332
Метод Лагранжа - вычисленное значение: 9.112893715594915
Метод Ньютона с разделенными - вычисленное значение: 9.112893715594966
Введите значение числа для вычислений : 1.34
Метод Лагранжа - вычисленное значение: 5.473071743063243
Метод Ньютона с разделенными - вычисленное значение: 5.47307174306325
Введите значение числа для вычислений : 7.56
Метод Лагранжа - вычисленное значение: -240.70051968081893
Метод Ньютона с разделенными - вычисленное значение: -240.70051968081907
Введите значение числа для вычислений : exit

Process finished with exit code 0
```

Пример 2

```
Выберите один вариант:
    1. Ввод данных точками
    2. Ввод данных точками через интервалами
    3. Ввод данных через функцию
1
Выберите один вариант:
    1. Ввод данных из файла
    2. Ввод данных с клавиатуры
1
Введите путь к файлу:
C:\Users\Elena\IdeaProjects\compMath\lab5\src\files\data8
Введите количество точек: 9
Введите исходные точки парами (x, y) через пробел
2 3
5 6
9 1
4 5
3 3
6 8
7 10
8 7
10 8
```




```
Выберите метод
    1. Многочлен Лагранжа
    2. Многочлен Гаусса
    3. Многочлен Стрилинга
    4. Все методы
4
Таблица конечных разностей
 i  |    | x         | y         | Δy_i      | Δy^2 _i   | Δy^3 _i   | Δy^4 _i   | Δy^5 _i   | Δy^6 _i   | Δy^7 _i   | Δy^8 _i   |
 0  | -4 |  2.0      |  3.0      |  0.0      |  2.0      | -3.0      |  5.0      | -8.0      |  7.0      |  5.0      | -21.0     |
 1  | -3 |  3.0      |  3.0      |  2.0      | -1.0      |  2.0      | -3.0      | -1.0      |  12.0     | -16.0     |
 2  | -2 |  4.0      |  5.0      |  1.0      |  1.0      | -1.0      | -4.0      |  11.0     | -4.0      |
 3  | -1 |  5.0      |  6.0      |  2.0      |  0.0      | -5.0      |  7.0      |  7.0      |
 4  | 0  |  6.0      |  8.0      |  2.0      | -5.0      |  2.0      |  14.0     |
 5  | 1  |  7.0      |  10.0     | -3.0      | -3.0      |  16.0     |
 6  | 2  |  8.0      |  7.0      | -6.0      |  13.0     |
 7  | 3  |  9.0      |  1.0      |  7.0      |
 8  | 4  |  10.0     |  8.0      |
```

```
Введите значение числа для вычислений : 6.13356
Метод Лагранжа - вычисленное значение: 8.364003628486007
Используется первая итнерполяционная формула Гаусса (x>a)
Метод Гаусса - вычисленное значение: 8.364003628486003
Метод Стирлинга - вычисленное значение: 8.364003628486005
Введите значение числа для вычислений : 7.34
Метод Лагранжа - вычисленное значение: 9.728150703622529
Используется первая итнерполяционная формула Гаусса (x>a)
Метод Гаусса - вычисленное значение: 9.72815070362253
Для данного значения метод Стирлинга не применяется
Введите значение числа для вычислений : 2.785
Метод Лагранжа - вычисленное значение: 2.612620542333582
Используется вторая итнерполяционная формула Гаусса (x<a)
Метод Гаусса - вычисленное значение: 2.612620542333581
Для данного значения метод Стирлинга не применяется
Введите значение числа для вычислений :
```

Пример 3

```
Выберите один вариант:
    1. Ввод данных точками
    2. Ввод данных точками через интервалами
    3. Ввод данных через функцию
3
Выберите один вариант:
    1. Ввод данных из файла
    2. Ввод данных с клавиатуры
2
Выберите функцию для решения:
    1. sin(x)
    2. x²+3x-4
    3. x*cos(x)-3
2
Введите значение левой границы интервала : 1
Введите значение правой границы интервала : 3
Введите количество точек: 15
Выберите метод
    1. Многочлен Лагранжа
    2. Многочлен Гаусса
    3. Многочлен Стрилинга
    4. Все методы
4
```

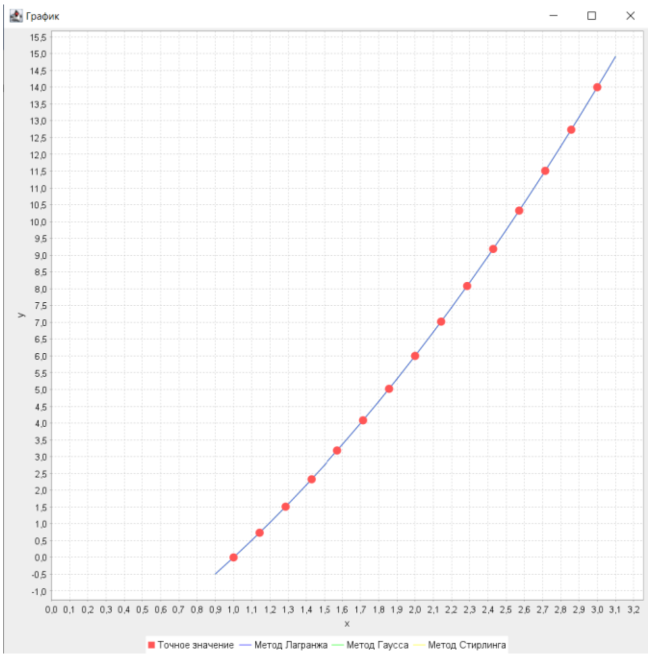


Таблица конечных разностей

i		x	y	Δy_i	Δy^2 _i	Δy^3 _i	Δy^4 _i	Δy^5 _i	Δy^6 _i	Δy^7 _i	Δy^8 _i	Δy^9 _i	Δy^10 _i	Δy^11 _i	Δy^12 _i	Δy^13 _i	Δy^14 _i
0	-7	1.0	0.0	0.734694	0.040816	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	-6	1.142857	0.734694	0.77551	0.040816	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	-5	1.285714	1.510204	0.816327	0.040816	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
3	-4	1.428571	2.326531	0.857143	0.040816	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0			
4	-3	1.571429	3.183673	0.897959	0.040816	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0				
5	-2	1.714286	4.081633	0.938776	0.040816	0.0	0.0	0.0	0.0	0.0	0.0	0.0					
6	-1	1.857143	5.020408	0.979592	0.040816	0.0	0.0	0.0	0.0	0.0	0.0						
7	0	2.0	6.0	1.020408	0.040816	0.0	0.0	0.0	0.0	0.0							
8	1	2.142857	7.020408	1.061224	0.040816	0.0	0.0	0.0	0.0								
9	2	2.285714	8.081633	1.102041	0.040816	0.0	0.0	0.0									
10	3	2.428571	9.183673	1.142857	0.040816	0.0	0.0										
11	4	2.571429	10.326531	1.183673	0.040816	0.0											
12	5	2.714286	11.510204	1.22449	0.040816												
13	6	2.857143	12.734694	1.265306													
14	7	3.0	14.0														

```
Введите значение числа для вычислений : 5.4
Точное значение: 41.36000000000001
Метод Лагранжа - вычисленное значение: 41.357550621032715
Используется первая итнерполяционная формула Гаусса (x>a)
Метод Гаусса - вычисленное значение: 41.316273685686994
Для данного значения метод Стирлинга не применяется
Введите значение числа для вычислений : 2.222
Точное значение: 7.603284
Метод Лагранжа - вычисленное значение: 7.6032839999999995
Используется первая итнерполяционная формула Гаусса (x>a)
Метод Гаусса - вычисленное значение: 7.603283999999992
Для данного значения метод Стирлинга не применяется
Введите значение числа для вычислений : 2.0004
Точное значение: 6.00280016
Метод Лагранжа - вычисленное значение: 6.00280016
Используется первая итнерполяционная формула Гаусса (x>a)
Метод Гаусса - вычисленное значение: 6.00280016
Метод Стирлинга - вычисленное значение: 6.00280016
Введите значение числа для вычислений : -12
Точное значение: 104.0
Метод Лагранжа - вычисленное значение: -98816.0
Используется вторая итнерполяционная формула Гаусса (x<a)
Метод Гаусса - вычисленное значение: -2.6638805996550217E7
Для данного значения метод Стирлинга не применяется
Введите значение числа для вычислений :
```

Вывод:

При выполнении лабораторной работы я познакомилась с различными методами интерполяции и выполнила программную реализацию некоторых из них.