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: **2**

Преподаватель:
Машина Е. А.

Выполнил:
Вальц Мартин
Группа: P3210

Санкт-Петербург, 2024 г.

Цель работы: решить задачу интерполяции, найти значения функции при заданных значениях аргумента, отличных от узловых точек.

1. Вычислительная реализация задачи

1. Выбрать таблицу $y=f(x)$**:**

	x	y	N варианта	X_1	X_2
Таблица 1.4	0.50	1.5320	**2**	0.502	0.645
	0.55	2.5356			
	0.60	3.5406			
	0.65	4.5462			
	0.70	5.5504			
	0.75	6.5559			
	0.80	7.5594			

2. Построить таблицу конечных разностей:

№	x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$	$\Delta^5 y_i$	$\Delta^6 y_i$
0.	0.50	1.5320	1.0036	0.0014	-0.0008	-0.0012	0.0059	-0.0166
1.	0.55	2.5356	1.0050	0.0006	-0.0020	0.0047	-0.0107	
2.	0.60	3.5406	1.0056	-0.0014	0.0027	-0.0060		
3.	0.65	4.5462	1.0042	0.0013	-0.0033			
4.	0.70	5.5504	1.0055	-0.0020				
5.	0.75	6.5559	1.0035					
6.	0.80	7.5594						

3. Вычислить значения функции для аргумента $\boldsymbol{X}_1$, используя первую или вторую интерполяционную формулу **Ньютона**:

Воспользуемся формулой Ньютона для интерполирования **вперед**, так как $X_1 = 0.502$ лежит в левой половине отрезка.

Для $X_1 = 0.502$: $t=\frac{(x-x_n)}{h}=\frac{(0.502-0.500)}{0.05}=0.04$

$$N_6(x)=y_0+t\Delta y_0+\frac{t(t-1)}{2!}\Delta^2 y_0+\frac{t(t-1)(t-2)}{3!}\Delta^3 y_0+\frac{t(t-1)(t-2)(t-3)}{4!}\Delta^4 y_0+\frac{t(t-1)(t-2)(t-3)(t-4)}{5!}\Delta^5$$

$$y(0.502)\approx 1.5320+0.04 * 1.0036+\frac{0.04(0.04-1)}{2} * 0.0014+\frac{0.04(0.04-1)(0.04-2)}{6} * (-0.0008)+\frac{0.04(}{}$$

$$y(0.502)\approx 1.572262 49$$

4. Вычислить значения функции для аргумента X_2, используя первую или вторую интерполяционную формулу **Гаусса**:

Центральная точка $a = 0.65$, $X_2 = 0.645 < 0.65$, то есть $x < a \rightarrow$ используем **вторую** интерполяционную формулу Гаусса.

$$t = \frac{(x - x_0)}{h} = \frac{(0.645 - 0.65)}{0.05} = -0.1$$

$$P_6(x) = y_0 + t\Delta y_{-1} + \frac{t(t+1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-2} + \frac{(t+2)(t+1)t(t-1)}{4!}\Delta^4 y_{-2} + \frac{(t+2)(t+1)t(t-1)(t-2)}{5!}\Delta$$

$$y(0.645) \approx 4.5462 + (-0.1) * 1.0056 + \frac{-0.1(-0.1+1)}{2} * (-0.0014) + \frac{(-0.1+1)(-0.1)(-0.1-1)}{6} * (-0.0020$$

$$y(0.502) \approx 4.4457138257325$$

2. Программная реализация задачи

Bessel.java

```
package lab5.methods;

public class Bessel extends Polynomial{
    @Override
    public double execute() {
        var values = Polynomial.getValues();

        float x[]=new float[10],y[][]=new float[10][10];
        double v = Polynomial.getX(), sum, u;
        int k;

        int n = values.size() - 1;

        int i,j;

        for (i = 1; i<n; i++)
            for (j = 0; j<n - i; j++)
                y[j][i] = y[j + 1][i - 1] - y[j][i - 1];

        for (i = 0; i<n; i++) {
            for (j = 0; j<n - i; j++)
                System.out.print(y[i][j]+"\t");
            System.out.println();
        }

        sum = (y[2][0] + y[3][0]) / 2;

        if (n % 2!=0)
            k = n / 2;
        else
            k = n / 2 - 1;

        u = (v - x[k]) / (x[1] - x[0]);

        for (i = 1; i<n; i++) {
            if (i % 2!=0)
                sum = (float) (sum+((u - 0.5)*cal_u(u, i - 1) * y[k][i]) / factorial(i));
            else
                sum = sum + (cal_u(u, i) * (y[k][i] + y[--k][i]) / (factorial(i) * 2));
        }
        System.out.println(sum);
        return sum;
    }

    int factorial(int n) {
        int fact = 1,i=2;
        for (i = 2; i<= n; i++)
            fact= fact*i;
        return fact;
```

```
    }

    double cal_u(double u, int n) {
        double tmp;
        int i;
        if (n == 0)
            return 1;
        tmp = u;
        for (i = 1; i<= n / 2; i++)
            tmp = tmp * (u - i);
        for (i = 1; i<n / 2; i++)
            tmp = tmp * (u + i);
        return tmp;
    }
}
```

Gauss.java

```
package lab5.methods;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


public class Gauss extends Polynomial {

    private final Map<Integer, List<Double>> deltas = new HashMap<>();
    private final Map<Integer, Double> xes = new HashMap<>();

    @Override
    public double execute() {
        double x = getX();
        ArrayList<Double[]> values = getValues();
        double result;
        double a = values.get((values.size() - 1) / 2)[0];
        int n = values.size();

        for (int j = 0, i = -(n / 2); i <= n / 2; i++, j++) {
            xes.put(i, values.get(j)[0]);

            var temp = new ArrayList<Double>();
            temp.add(values.get(j)[1]);
            deltas.put(i, temp);
        }
        for (int k = 0, i = 0; i < n; i++, k++) {
            for (int j = -(n / 2); j < n / 2 - k; j++) {
                Double fd = deltas.get(j).get(i);
                Double sd = deltas.get(j + 1).get(i);
                deltas.get(j).add(sd - fd);
            }
        }
```

```
    if (x > a) {
        result = firstFormula();
    } else {
        result = secondFormula();
    }

    return result;
}

private double firstFormula() {
    double x = getX();
    ArrayList<Double[]> values = getValues();
    int n = values.size();
    double h = values.get(1)[0] - values.get(0)[0];
    double t = (x - values.get((n + 1) / 2 - 1)[0]) / h;
    double result = 0;
    double tkoeff = 1;
    for (int j = 1, i = 0; j < n - 1; i--, j += 2) {
        double delta1 = deltas.get(i).get(j - 1);
        double delta2 = deltas.get(i).get(j);
        tkoeff = 1;
        for (int k = 0; k < j - 1; k++) {
            tkoeff *= t + (k % 2 == 0 ? k : -k);
        }
        result += delta1 * tkoeff / factorial(j - 1);
        tkoeff *= t + j / 2;
        result += delta2 * tkoeff / factorial(j);
    }
    tkoeff *= t - n / 2;
    result += deltas.get(-(n/2)).get(n-1) * tkoeff / factorial(n - 1);

    return result;
}

private double secondFormula() {
    double x = getX();
    ArrayList<Double[]> values = getValues();
    int n = values.size();
    double h = values.get(1)[0] - values.get(0)[0];
    double t = (x - values.get((n + 1) / 2 - 1)[0]) / h;
    double result = deltas.get(0).get(0);
    double temp = 1;
    for (int j = 2, i = -1, f = 0, s = 1; j <= n - 1; i--, j += 2, f--, s++) {
        double delta1 = deltas.get(i).get(j - 1);
        double delta2 = deltas.get(i).get(j);

        temp *= t + f;
        result += delta1 * temp / factorial(j - 1);
        temp *= t + s;
        result += delta2 * temp / factorial(j);
    }
```

```
        return result;
    }

    private long factorial(int number) {
        long result = 1;

        for (int factor = 2; factor <= number; factor++) {
            result *= factor;
        }

        return result;
    }
}
```

Lagrange.java

```
package lab5.methods;


import java.util.ArrayList;

public class Lagrange extends Polynomial{
    @Override
    public double execute() {
        double x = getX();
        ArrayList<Double[]> values = getValues();
        double result = 0;
        for(int i = 0; i < values.size(); i++){
            double intermediateResult = 1;
            for(int j = 0; j < values.size(); j++){
                if (i == j) continue;
                intermediateResult *= (
                        (x - values.get(j)[0]) /
                                (values.get(i)[0] - values.get(j)[0])
                );
            }
            result += intermediateResult * values.get(i)[1];
        }
        return result;
    }
}
```

Newton.java

```
package lab5.methods;

import java.util.ArrayList;
import java.util.List;

public class Newton extends Polynomial{
    @Override
    public double execute() {
```

```
        double x = getX();
        ArrayList<Double[]> values = getValues();
        double result = values.get(0)[1];
        for(int i = 2; i < values.size() + 1; i++){
            List<Double[]> mas = values.subList(0, i);
            double production = calculateDividedDifference(mas);
            for (int j = 0; j < i - 1; j++){
                production *= (x - values.get(j)[0]);
            }
            result += production;
        }
        return result;
    }

    public double calculateDividedDifference(List<Double[]> mas){
        List<Double[]> mas1 = mas.subList(1, mas.size());
        List<Double[]> mas2 = mas.subList(0, mas.size() - 1);
        if (mas1.size() == 1 && mas2.size() == 1) {
            return (
                    (mas1.get(0)[1] - mas2.get(0)[1]) /
                            (mas1.get(0)[0] - mas2.get(0)[0])
            );
        }
        return (
                (calculateDividedDifference(mas1) - calculateDividedDifference(mas2)) /
                        (mas.get(mas.size() - 1)[0] - mas.get(0)[0])
        );
    }
}
```

Stirling.java

```
package lab5.methods;

import static java.lang.Math.pow;

public class Stirling extends Polynomial{
    @Override
    public double execute() {
        var values = Polynomial.getValues();
        var x1 = Polynomial.getX();
        var n = values.size();
        double h, a, u;
        double y1 = 0, N1 = 1, d = 1,
                N2 = 1, d2 = 1, temp1 = 1,
                temp2 = 1, k = 1, l = 1, delta[][];

        delta = new double[n][n];
        int i, j, s;
        h = values.get(1)[0] - values.get(0)[0];
        s = n / 2;
        a = values.get(s)[0];
        u = (x1 - a) / h;
```

```
        // Preparing the forward difference
        // table
        for (i = 0; i < n - 1; ++i) {
            delta[i][0] = values.get(i + 1)[1] - values.get(i)[1];
        }
        for (i = 1; i < n - 1; ++i) {
            for (j = 0; j < n - i - 1; ++j) {
                delta[j][i] = delta[j + 1][i - 1]
                        - delta[j][i - 1];
            }
        }

        // Calculating f(x) using the Stirling
        // formula
        y1 = values.get(s)[1];

        for (i = 1; i <= n - 1; ++i) {
            if (i % 2 != 0) {
                if (k != 2) {
                    temp1 *= (pow(u, k) -
                            pow((k - 1), 2));
                }
                else {
                    temp1 *= (pow(u, 2) -
                            pow((k - 1), 2));
                }
                ++k;
                d *= i;
                s = (n - i) / 2;
                y1 += (temp1 / (2 * d)) *
                        (delta[s][i - 1] +
                                delta[s - 1][i - 1]);
            }
            else {
                temp2 *= (pow(u, 2) -
                        pow((l - 1), 2));
                ++l;
                d *= i;
                s = (n - i) / 2;
                y1 += (temp2 / (d)) *
                        (delta[s][i - 1]);
            }
        }
        return y1;
    }
}
```

Вывод

В ходе выполнения данной лабораторной работы я рассмотрел и реализовал методы интерполяции Ньютона и Гаусса для заданной таблицы данных. Интерполяция позволяет нам предсказывать значения функции в промежуточных точках на основе имеющихся данных.

С помощью разработанной программы были вычислены приближенные значения функции для заданных аргументов с использованием методов Ньютона и Гаусса. Было проведено сравнение результатов, полученных разными методами.

Результаты показали, что оба метода могут быть эффективно использованы для интерполяции, но их точность может зависеть от конкретной функции и распределения данных. Эта работа подчеркивает важность выбора подходящего метода интерполяции в соответствии с требованиями конкретной задачи.