Отчёт по лабораторной работе №1

Денисов-Элерс Э.Ф, 5130904/40003, вариант: 34

1. Задание

Для функции $f(x) = \sin(x^2)$ по узлам $x_k = 0.2k$ $(k = 0,1, \dots 6)$ построить полином Лагранжа $L(x)$ 6-й степени и сплайн-функцию $S(x)$. Затем сравнить значения трёх интегралов:

$$\int_0^{1.2} f(x)dx \quad \int_0^{1.2} S(x)dx \quad \int_0^{1.2} L(x)dx$$

Для вычисления первого и последнего интегралов использовать программу **QUANC8**.

2. Описание решения

Язык: Java

Использованные методы из библиотеки:

- QUANC8 – адаптивный метод численного интегрирования (ABSERR = 1e-6, RELERR = 1e-6)
- SPLINE – метод интерполяции, основанный на аппроксимации кубическими функциями (сплайнами)

3. Результаты

```
================================================================================================================
f(x) = sin(x^2)
integral from 0.0 to 1.2
+---+----------+----------------------+-----------------------+-----------------------+-------+-----------------+
| N | Method   | Result               | ERREST                | Deviation             | Count | Additional info |
+---+----------+----------------------+-----------------------+-----------------------+-------+-----------------+
| 1 | Function | 0,49611578848272170  | 5,982768858364244E-13 | 0,000000000000000E+00 | 33    | QUANC8 flag=0.0 |
| 2 | Spline   | 0,49599621016821316  | -                     | 1,195783145085394E-04 | -     | iflag=0         |
| 3 | Lagrange | 0,49612811589267347  | 4,861973753979180E-17 | 1,232740995177339E-05 | 33    | QUANC8 flag=0.0 |
+---+----------+----------------------+-----------------------+-----------------------+-------+-----------------+


================================================================================================================
```

$$\int_0^{1.2} f(x)dx = 0{,}49611578848272170 \ (1)$$

$$\int_0^{1.2} S(x)dx = 0{,}49599621016821316 \ (2)$$

$$\int_0^{1.2} L(x)dx = 0{,}49612811589267347 \ (3)$$

4. Анализ результатов

Примем за эталонное значение результат вычисления интеграла (1). Это допустимо, поскольку малое значение $ERREST \approx 5{,}98 \cdot 10^{-13}$ свидетельствует о малой погрешности при вычислении. Нулевой флаг подтверждает успешное завершение алгоритма без признаков численной нестабильности для данного случая.

Сравним с эталонным результаты вычисления с помощью аппроксимации сплайнами (2) и полиномом Лагранжа 6-й степени (3). Отклонения от (1) составили $1{,}19 \cdot 10^{-4}$ и $1{,}23 \cdot 10^{-5}$ соответственно. В данном случае интерполяционный многочлен Лагранжа степени 6 даёт более точное значение интеграла, чем кубический сплайн. Если погрешность некритична, то использование сплайна более предпочтительно, так как он представляет собой полином третьей степени, а не шестой, как в случае полинома Лагранжа.

Малое значение $ERREST \approx 4{,}86 \cdot 10^{-17}$ для интеграла (3) показывает, что программа QUANC8 отработала с высокой точностью. В случае интеграла (2) точность вычисления обеспечивается аналитическим интегрированием построенного сплайна. Таким образом, основной вклад в различие результатов интегрирования вносит не погрешность интегратора, а ошибка аппроксимации функции.

Приложение 1. Код программы

Lab1

```
package com.edwin.lab1;

import com.edwin.sources.quanc8.Quanc8;
import com.edwin.sources.splineSeval.Spline;

import java.util.function.Function;

import static
com.edwin.sources.lagrangeInterpolation.LagrangeInterpolation.calculateY;
import static
com.edwin.sources.lagrangeInterpolation.LagrangeInterpolation.generateX;
import static
com.edwin.sources.lagrangeInterpolation.LagrangeInterpolation.interpolate;

public class Lab1 {
    final static double A = 0;
    final static double B = 1.2;
    final static double ABSERR = 1e-6;
    final static double RELERR = 1e-6;
    final static Function<Double, Double> F = x -> Math.sin(x * x);

    public static void main(String[] args) {
        double[] x = generateX(0, 6, 0.2);
        double[] y = calculateY(x, F);
        int n = x.length;

        // just a function
        Quanc8 fQuanc8 = new Quanc8(F, A, B, ABSERR, RELERR);
        fQuanc8.calculate();

        // spline
        Spline spline = new Spline(n, x, y, new double[n], new double[n], new
double[n], 0);
        spline.spline();
        double splineResult = calculateSplineIntegral(A, B, x, y,
spline.getB(), spline.getC(), spline.getD());
        double splineDeviation = Math.abs(fQuanc8.getRESULT() -
splineResult);

        // lagrange
        Function<Double, Double> L = x0 -> interpolate(x, y, x0);
        Quanc8 lQuanc8 = new Quanc8(L, A, B, ABSERR, RELERR);
        lQuanc8.calculate();
        double lagrangeDeviation = Math.abs(fQuanc8.getRESULT() -
lQuanc8.getRESULT());

        // format results
        formatResults(fQuanc8, lQuanc8, spline, splineResult,
splineDeviation, lagrangeDeviation);
    }

    private static double calculateSplineIntegral(double start, double end,
                                                  double[] x, double[] y,
                                                  double[] b, double[] c,
double[] d) {
        int n = x.length;
        double result = 0;
        for (int i = 0; i < n - 1; i++) {
```

```
            double segL = Math.max(start, x[i]);
            double segR = Math.min(end, x[i + 1]);

            if (segL >= segR) {
                continue;
            }

            double[] coefficients = new double[]{y[i], b[i], c[i], d[i]};
            result += calculatePolynomialIntegral(segL, segR, x[i],
coefficients);
        }

        return result;
    }

    private static double calculatePolynomialIntegral(double start, double
end, double xk, double[] coefficients) {
        int n = coefficients.length;
        double result = 0;
        for (int i = 0; i < n; i++) {
            result += coefficients[i] * ((Math.pow((end - xk), i + 1) -
Math.pow((start - xk), i + 1)) / (i + 1));
        }

        return result;
    }

    private static void formatResults(Quanc8 fQuanc8, Quanc8 lQuanc8, Spline
spline, double splineResult,
                                      double splineDeviation, double
lagrangeDeviation) {
        System.out.println("=".repeat(115));
        System.out.println("f(x) = sin(x^2)");
        System.out.printf("integral from %s to %s%n", A, B);

        String fmt = "| %-1s | %-8s | %-20s | %-22s | %-22s | %-5s | %-15s
|%n";
        String fmtData = "| %-1s | %-8s | %-20.17f | %-22s | %-22s | %-5s |
%-15s |%n";
        String line = "+---+----------+----------------------+--------------
---------+" +
                "------------------------+-------+-----------------+";

        System.out.println(line);
        System.out.printf(fmt, "N", "Method", "Result", "ERREST",
"Deviation", "Count", "Additional info");
        System.out.println(line);
        System.out.printf(fmtData, '1', "Function",
                fQuanc8.getRESULT(),
                String.format("%.15E", fQuanc8.getERREST()),
                String.format("%.15E", 0.0),
                fQuanc8.getNOFUN(),
                "QUANC8 flag=" + fQuanc8.getFLAG());

        System.out.printf(fmtData, '2', "Spline",
                splineResult,
                "-",
                String.format("%.15E", splineDeviation),
                "-",
                "iflag=" + spline.getIflag());

        System.out.printf(fmtData, '3', "Lagrange",
                lQuanc8.getRESULT(),
                String.format("%.15E", lQuanc8.getERREST()),
```

```
                String.format("%.15E", lagrangeDeviation),
                lQuanc8.getNOFUN(),
                "QUANC8 flag=" + lQuanc8.getFLAG());
        System.out.println(line);
        System.out.println();
        System.out.println("=".repeat(115));
    }
}
```

LagrangeInterpolation

```
package com.edwin.sources.lagrangeInterpolation;

import java.util.function.Function;

public class LagrangeInterpolation {
    public static double interpolate(double[] x, double[] y, double x0) {
        double result = 0;
        int n = x.length;

        for (int i = 0; i < n; i++) {
            double term = y[i];
            for (int j = 0; j < i; j++) {
                term *= (x0 - x[j]) / (x[i] - x[j]);
            }
            for (int j = i + 1; j < n; j++) {
                term *= (x0 - x[j]) / (x[i] - x[j]);
            }

            result += term;
        }

        return result;
    }

    public static double[] generateX(int start, int end, double multiplier) {
        double[] vertexes = new double[end - start + 1];
        for (int i = start; i <= end; i++) {
            vertexes[i - start] = multiplier * i;
        }

        return vertexes;
    }

    public static double[] calculateY(double[] x, Function<Double, Double> f)
{
        int n = x.length;
        double[] y = new double[n];
        for (int i = 0; i < n; i++) {
            y[i] = f.apply(x[i]);
        }

        return y;
    }
}
```