```
import static java.lang.Math.*;

import java.util.stream.*;

import java.text.DecimalFormat;

import Jama.Matrix;

public class Solution {
	public static final int a = -2;
	public static final int b = 2;
	public static final double h = 0.1;
	public static final int n = 3;

	public static double f(double x) {
		return sin(2*x) * log(x+5);
	}

	public static double s(int i, double[] x) {
		return IntStream.range(0, x.length)
					.parallel()
					.mapToDouble(k -> pow(x[k], i))
					.sum();
	}

	public static double m(int i, double[] x, double f[]) {
		return IntStream.range(0, x.length)
					.parallel()
					.mapToDouble(k -> f[k] * pow(x[k], i))
					.sum();
	}

	public static Matrix fillS(double[] x) {
		double[] s = new double[2*n + 1];
		IntStream.range(0, 2*n+1)
				.parallel()
				.forEach(i -> s[i] = s(i, x));

		Matrix temp = new Matrix(n+1, n+1);
		IntStream.range(0, n+1)
				.parallel()
				.forEach(i -> IntStream.range(0, n+1)
							.parallel()
							.forEach(j -> temp.set(i, j, s[i+j])));
		return temp;
	}

	public static Matrix fillM(double[] x, double[] f) {
		Matrix temp = new Matrix(n+1, 1);
		IntStream.range(0, n+1)
				.parallel()
				.forEach(i -> temp.set(i, 0, m(i, x, f)));
		return temp;
	}

	public static void showQ(double[] c) {
		DecimalFormat df = new DecimalFormat();
```

```
        df.setPositivePrefix("+ ");
        df.setNegativePrefix("- ");
        df.setMaximumFractionDigits(9);

        System.out.print(c[0] + " ");
        IntStream.range(1, n+1)
                        .forEach(i -> System.out.print(df.format(c[i]) + "*x^" + i + " "));
        System.out.println();
    }

    public static double countQ(double x, double[] c) {
        return IntStream.range(0, n+1)
                                    .parallel()
                                    .mapToDouble(i -> c[i] * pow(x, i))
                                    .sum();
    }


    public static void main(String[] args) {
        double[] x = DoubleStream.iterate(a, i -> i + h)
                                                    .takeWhile(i -> i < b + h)
                                                    .toArray();

        double[] f = DoubleStream.of(x)
                                                    .map(Solution::f)
                                                    .toArray();

        Matrix s = fillS(x);
        Matrix m = fillM(x, f);
        double[] c = s.solve(m).getColumnPackedCopy();

        showQ(c);

        double deltaF = DoubleStream.of(x)
                                                        .map(value -> pow(f(value) - countQ(value, c), 2))
                                                        .sum();
        System.out.println(deltaF);
    }
}
```