1. Вычисляем узлы по формуле:

$$x_i = a + i * h$$

2. Считаем знач. ф-ции в узлах:

$$f = \sin 2x \cdot \ln(x+5)$$

3. Формируем расширенную матрицу M по формулам:

$$\frac{h_1}{3} M_0 + \frac{h_1}{6} M_1 = \frac{f_1 - f_0}{h_1} - f_0'$$

$$\frac{h_i}{6} M_{i-1} + \frac{h_i + h_{i+1}}{3} M_i + \frac{h_{i+1}}{6} M_{i+1} =$$

$$= \frac{f_{i+1} - f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i}, \quad i = \overline{1, N-1}$$

$$\frac{h_N}{6} M_{N-1} + \frac{h_N}{3} M_N = f_N' - \frac{f_N - f_{N-1}}{h_N}$$

4. Решаем методом левой прогонки

5. Считаем кубический сплайн по формуле:

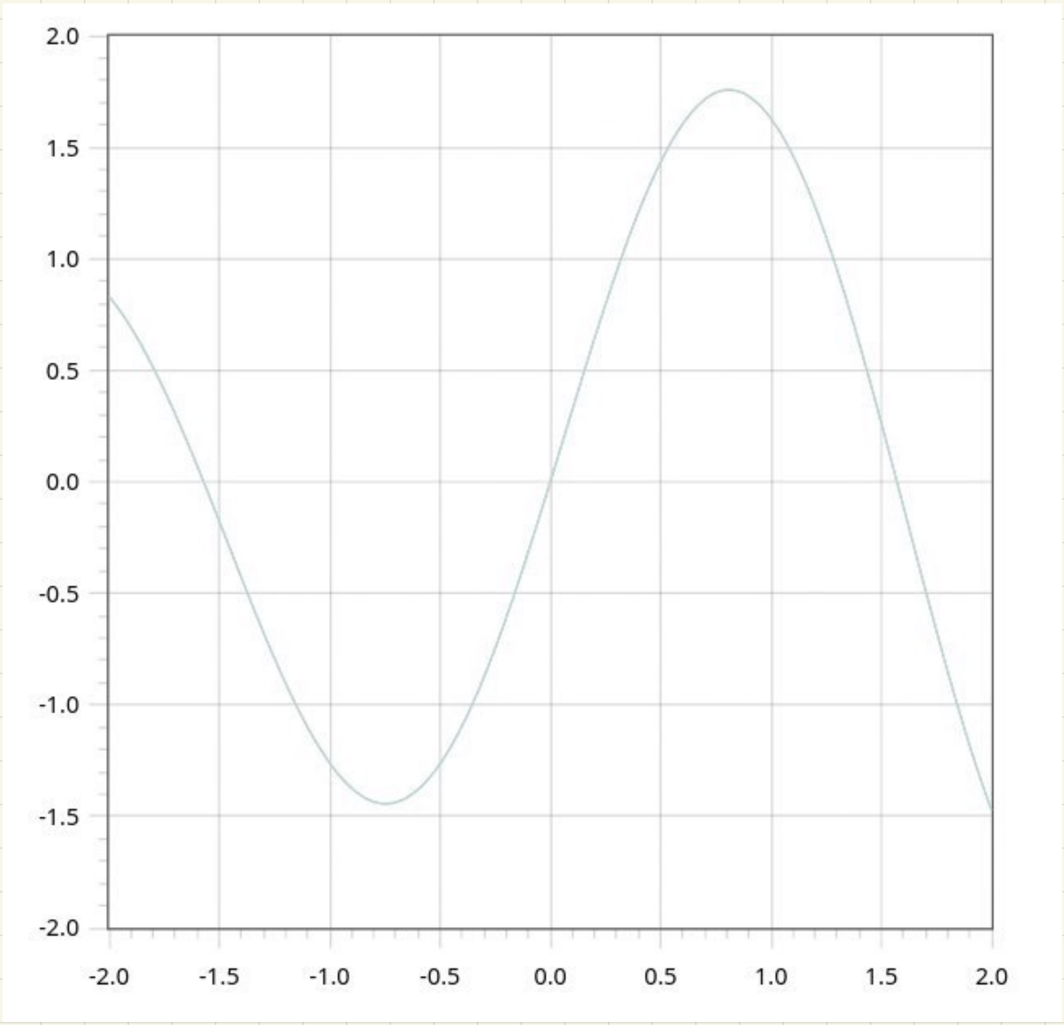
$$S_3(x) = \left\{ P_{i,3} = M_{i-1} \frac{(x_i - x)^3}{6h_i} + M_i \cdot \frac{(x - x_{i-1})^3}{6h_i} + \right.$$

$$+\left(f_{i-1}-\frac{h_i^2}{6}\cdot M_{i-1}\right)\frac{x_i-x}{h_i}+\left(f_i-\frac{h_i^2}{6}\cdot M_i\right)\cdot\frac{x-x_{i-1}}{h_i}\Bigg|$$

$$x\in[x_{i-1};x_i],\quad i=\overline{1,N}\}$$

6. И находим $\max\limits_{i=\overline{1,100}}\left|S_3(x_i)-f(x_i)\right| =$

$= 0.0022349241984065404$



Код:

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

import java.util.stream.IntStream;

import Jama.Matrix;

public class Main {
	public static final int a = -2;
	public static final int b = 2;
	public static final int N = 10;
	public static final double h = (double)(b - a) / N;

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

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

	public static double coef1(double M) {
		return M / (6*h);
	}

	public static double coef2(double f, double M) {
		return f/h - M*h/6;
	}

	public static Matrix fillM() {
		Matrix temp = new Matrix(N+1, N+1);

		//fill first row
		temp.set(0, 0, h/3);
		temp.set(0, 1, h/6);

		IntStream.range(1, N)
				.parallel()
				.forEach(i -> {
					temp.set(i, i-1, h/6);
					temp.set(i, i, 2*h/3);
					temp.set(i, i+1, h/6);
				});

		//fill last row
		temp.set(N, N-1, h/6);
		temp.set(N, N, h/3);

		return temp;
	}

	public static Matrix fillF(double[] f) {
		Matrix temp = new Matrix(N+1, 1);

		//fill first
		temp.set(0, 0, (f[1] - f[0]) / h - derivativeF((double)a));

		IntStream.range(1, N)
```

```
                        .parallel()
                        .forEach(i -> temp.set(i, 0, (f[i+1] - 2*f[i] + f[i-1]) / h));

        //fill last
        temp.set(N, 0, derivativeF((double)b) - (f[N] - f[N-1]) / h);

        return temp;
    }

    public static double countP(int i, double x, double[] interNodes, double[] f, double[] M) {
        return coef1(M[i-1]) * pow(interNodes[i] - x, 3)
                    +coef1(M[i]) * pow(x - interNodes[i-1], 3)
                    +coef2(f[i-1], M[i-1]) * (interNodes[i] - x)
                    +coef2(f[i], M[i]) * (x - interNodes[i-1]);
    }

    public static double countS(double x, double[] interNodes, double[] f, double[] M) {
        int i = 1;
        while(x > interNodes[i]) {
            i++;
        }

        return countP(i, x, interNodes, f, M);
    }

    public static void main(String[] args) {
        double[] interNodes = IntStream.rangeClosed(0, N)
                                        .mapToDouble(i -> a + i*h)
                                        .toArray();

        double[] funcValue = IntStream.rangeClosed(0, N)
                                        .mapToDouble(i -> a + i*h)
                                        .map(Main::f)
                                        .toArray();

        Matrix M = fillM();
        Matrix f = fillF(funcValue);
        double[] moments = M.leftSweet.getColumnPackedCopy();

        double max = IntStream.rangeClosed(0, 100)
                        .mapToDouble(i -> a + i * (double)(b - a) / 100)
                        .map(x -> abs(countS(x, interNodes, funcValue, moments) - f(x)))
                        .max()
                        .getAsDouble();
        System.out.println(max);
    }
}
```