23 Вопрос. Интерполяция сплайнами.

Построение интерполяционного многочлена Лагранжа и Ньютона с использованием большого числа узлов интерполирование на отрезке [a, b] может привести к плохому приближению интерполируемой функции из-за возрастания вычислительной погрешности. Кроме того, построенный многочлен будет иметь высокую степень, что тоже весьма нежелательно.
Этих неприятностей можно избежать, разбив отрезок [a, b] на частичные отрезки и построив на каждом из них многочлен невысокой степени, так или иначе приближенный к заданной функции *f(x)*.

Суть сплайн-интерполяции заключается в определении интерполирующей функции по формулам одного типа для различных подмножеств и в стыковке значений функции и ее производных на границах подмножеств. Наиболее изученным и широко применяемым является вариант, в котором между любыми двумя точками строится многочлен п-й степени. $S(x) = \sum_{k=0}^{n} a_{ik} x^k$, $x_{i-1} \le x \le x_i$. который в узлах интерполяции принимает значения интерполируемой функции и непрерывен вместе со своими (п-1)-й производными. Такой кусочно-непрерывный интерполяционный многочлен называется сплайном.

Принципиальное отличие идеи сплайн-интерполяции от интерполяции полиномом состоит в том, что полином один, а сплайн состоит из нескольких полиномов, а именно их количество равно количеству интервалов, внутри которых мы производим интерполяцию.

Определение кубического сплайна.

Пусть на отрезке [a, b] задана функция $y = f(x)$. Рассмотрим сетку узлов

$$a = x_0 < x_1 < x_2 < \dots < x_n = b \ (1)$$

и обозначим расстояние между смежными узлами:

$$h_i = x_i - x_{i-1}, i = 1, 2, \dots, n \ (2)$$

Назовем кубическим сплайном функции y = f(x), x ∈ [a; b] на сетке (2) функцию $S(x)$, удовлетворяющую условиям:

1. На каждом частичном отрезке $[x_{i-1}, x_i]$ функция S(x) является полиномом третьей степени.

2. Функция, ее первая S'(x) и вторая S''(x) производные непрерывны на сегменте [a; b].

3. В узлах интерполяции сплайн принимает значения интерполируемой функции:

$S(x_i) = f(x_i) = f_i$, $i = 0, 1, 2, \dots, n$.

4. На концах сегмента [a; b] вторая производная функции S''(x) удовлетворяет условиям S''(a)= S''(b)=0.

Построение:

На каждом отрезке $[x_{i-1}, x_i]$, $i = \overline{1, N}$ функция $S(x)$ есть полином третьей степени $S_i(x)$, коэффициенты которого надо определить. Запишем для удобства $S_i(x)$ в виде:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

тогда

$$S_i(x_i) = a_i, \quad S_i'(x_i) = b_i, \quad S_i''(x_i) = 2c_i, \quad i = \overline{1, N}.$$

Условия непрерывности всех производных до второго порядка включительно записываются в виде

$$\begin{aligned} S_i(x_{i-1}) &= S_{i-1}(x_{i-1}), \\ S_i'(x_{i-1}) &= S_{i-1}'(x_{i-1}), \\ S_i''(x_{i-1}) &= S_{i-1}''(x_{i-1}), \end{aligned}$$

где i меняется от 1 до N, а условия интерполяции в виде

$$S_i(x_i) = f(x_i).$$

Обозначим: $h_i = x_i - x_{i-1} \quad (i = \overline{1, N}), \quad f_i = f(x_i) \quad (i = \overline{0, N})$

Отсюда получаем формулы для вычисления коэффициентов "Естественного сплайна":

$$a_i = f(x_i);$$

$$d_i = \frac{c_i - c_{i-1}}{3 \cdot h_i};$$

$$b_i = \frac{a_i - a_{i-1}}{h_i} + \frac{2 \cdot c_i + c_{i-1}}{3} \cdot h_i;$$

$$c_{i-1} \cdot h_i + 2 \cdot c_i \cdot (h_i + h_{i+1}) + c_{i+1} \cdot h_{i+1} = 3 \cdot \left(\frac{a_{i+1} - a_i}{h_{i+1}} - \frac{a_i - a_{i-1}}{h_i} \right),$$

причем $c_N = S''(x_N) = 0$ и $c_1 - 3 \cdot d_1 \cdot h_1 = S''(x_0) = 0$.

Если учесть, что $c_0 = c_N = 0$, то вычисление c можно провести с помощью метода прогонки для трёхдиагональной матрицы.

Погрешность

$$|R_n(x)| \leqslant \frac{5}{2} h^3 \max_{y \in [a,b]} |f^{(3)}(y)|$$

где h – шаг сетки.

Код: (Но без метода main)

```
package lecho.lib.hellocharts.utils;
import java.util.List;
public class SplineInterpolator {
	private final List<Float> mX;
	private final List<Float> mY;
	private final float[] mM;
```

```java
private SplineInterpolator(List<Float> x, List<Float> y, float[] m) {
		mX = x;
		mY = y;
		mM = m;
	}
	public static SplineInterpolator createMonotoneCubicSpline(List<Float> x, 
List<Float> y) {
		if (x == null || y == null || x.size() != y.size() || x.size() < 2) {
			throw new IllegalArgumentException("There must be at least two 
control "
					+ "points and the arrays must be of equal length.");
		}

		final int n = x.size();
		float[] d = new float[n - 1]; // could optimize this out
		float[] m = new float[n];

		// Compute slopes of secant lines between successive points.
		for (int i = 0; i < n - 1; i++) {
			float h = x.get(i + 1) - x.get(i);
			if (h <= 0f) {
				throw new IllegalArgumentException("The control points must 
all "
						+ "have strictly increasing X values.");
			}
			d[i] = (y.get(i + 1) - y.get(i)) / h;
		}

		// Initialize the tangents as the average of the secants.
		m[0] = d[0];
		for (int i = 1; i < n - 1; i++) {
			m[i] = (d[i - 1] + d[i]) * 0.5f;
		}
		m[n - 1] = d[n - 2];

		// Update the tangents to preserve monotonicity.
		for (int i = 0; i < n - 1; i++) {
			if (d[i] == 0f) { // successive Y values are equal
				m[i] = 0f;
				m[i + 1] = 0f;
			} else {
				float a = m[i] / d[i];
				float b = m[i + 1] / d[i];
				float h = (float) Math.hypot(a, b);
				if (h > 9f) {
					float t = 3f / h;
					m[i] = t * a * d[i];
					m[i + 1] = t * b * d[i];
				}
			}
		}
		return new SplineInterpolator(x, y, m);
	}
```

```
	public float interpolate(float x) {
		// Handle the boundary cases.
		final int n = mX.size();
		if (Float.isNaN(x)) {
			return x;
		}
		if (x <= mX.get(0)) {
			return mY.get(0);
		}
		if (x >= mX.get(n - 1)) {
			return mY.get(n - 1);
		}


		int i = 0;
		while (x >= mX.get(i + 1)) {
			i += 1;
			if (x == mX.get(i)) {
				return mY.get(i);
			}
		}
		float h = mX.get(i + 1) - mX.get(i);
		float t = (x - mX.get(i)) / h;
		return (mY.get(i) * (1 + 2 * t) + h * mM[i] * t) * (1 - t) * (1 - t)
				+ (mY.get(i + 1) * (3 - 2 * t) + h * mM[i + 1] * (t - 1)) *
t * t;
	}
	@Override
	public String toString() {
		StringBuilder str = new StringBuilder();
		final int n = mX.size();
		str.append("[");
		for (int i = 0; i < n; i++) {
			if (i != 0) {
				str.append(", ");
			}
			str.append("(").append(mX.get(i));
			str.append(", ").append(mY.get(i));
			str.append(": ").append(mM[i]).append(")");
		}
		str.append("]");
		return str.toString();
	}
```