Описание метода, расчетные формулы

Интерполяционный полином Лагранжа имеет вид:

$$L_n(x) = \sum_{i=0}^{n} f(x_i) \cdot l_n(x_i),$$

где $l_n(x_i)$ – *множитель Лагранжа*

$$l_n(x_i) = \frac{(x - x_0) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)} = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

Под узлами чебышева понимают *корни многочлена Чебышева первого рода*. Они часто исползуется в качестве узлов при *полиномиальной интерполяции*, так как позволяют снизить влияние *феномена Рунге*.

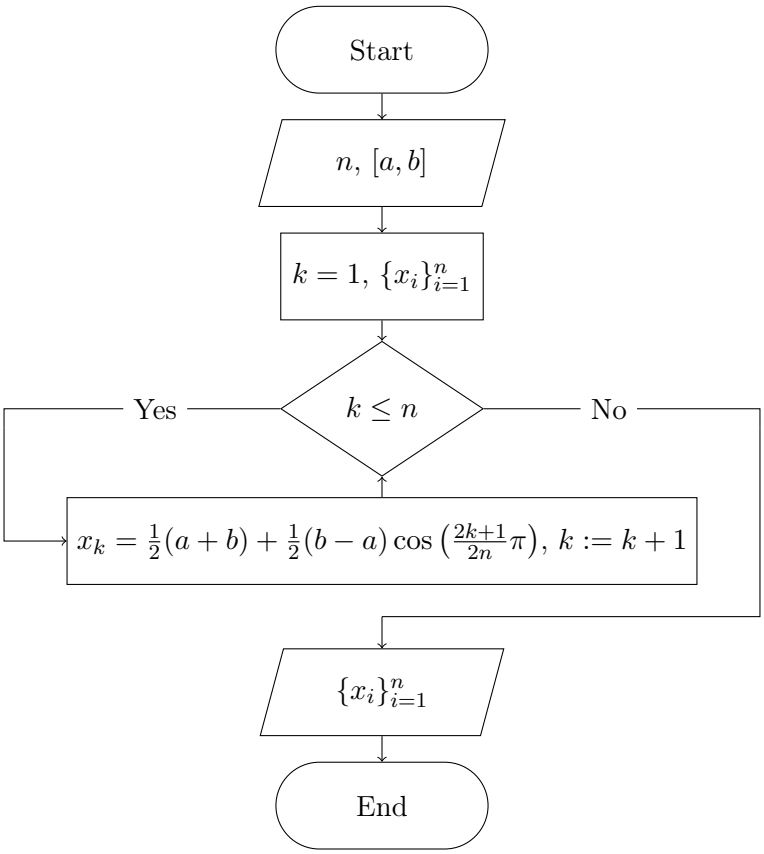
Для натурального числа n узлы Чебышева на отрезке $[-1, 1]$ задаются формулой

$$x_k = \cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, \ldots, n.$$

Это корни многочлена Чебышева первого рода степени n. Для получения узлов на произвольном отрезке $[a, b]$ можно применить *аффинное преобразование* отрезков:

$$x_k = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, \ldots, n.$$

Блок-схема численного метода



Листинг реализованного численного метода программы

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

public class ChebyshevPolynomial {
    private final int order;

    public ChebyshevPolynomial(int order) {
        if (order < 1) {
            throw new IllegalArgumentException("Order can not be less than 1");
        }
        this.order = order;
    }

    public List<Double> nodesAt(double a, double b) {
        List<Double> nodes = new ArrayList<>(order);
        /*
         * Nodes affine transformation over an arbitrary interval [a,b]
         */
        for (Double root : roots()) {
            nodes.add(0.5 * (a + b) + 0.5 * (a - b) * root);
        }
        return nodes;
    }

    private Double root(int i) {
        /*
         * i-th root of Chebyshev polynomial, where i = 1,2,...,n
         */
        return Math.cos(Math.PI * (i - 0.5) / order);
    }

    public List<Double> roots() {
        List<Double> roots = new ArrayList<>(order);
        /*
         * Chebyshev polynomial roots at (-1,1)
         */
        for (int i = 1; i <= order; i++) {
            roots.add(root(i));
        }
        return roots;
    }
}
```

```java
import net.objecthunter.exp4j.Expression;
import net.objecthunter.exp4j.ExpressionBuilder;

public class Function {
    private final Expression expression;

    public Function(String expression) {
        this.expression = new ExpressionBuilder(expression).variable("x").build();
    }

    public Double apply(double x) {
        try {
            return this.expression.setVariable("x", x).evaluate();
        } catch (RuntimeException exception) {
            return Double.NaN;
        }
    }
}
```

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

public final class LagrangianPolynomialBuilder {
    private final Function initialFunction;
    private final StringBuilder lagrangianPolynomial;
    private final List<Double> xData;
    private final List<Double> yData;

    public LagrangianPolynomialBuilder(Function initialFunction) {
        if (initialFunction == null) {
            throw new IllegalArgumentException("Initial function can not be null.");
        }
        this.initialFunction = initialFunction;
        this.lagrangianPolynomial = new StringBuilder();
        this.xData = new ArrayList<>(4);
        this.yData = new ArrayList<>(4);
    }

    public LagrangianPolynomialBuilder setNodes(List<Double> nodes) {
        if (nodes.size() < 2) {
            throw new IllegalArgumentException("Nodes can not be less than 2");
        }
        /* Calculate a data */
        for (Double node : nodes) {
            this.xData.add(node);
            this.yData.add(initialFunction.apply(node));
        }
        return this;
    }

    private StringBuilder lagrangianMultiplier(int i) {
        StringBuilder numerator = new StringBuilder();
        Double dominator = 1.0;

        for (int j = 0; j < xData.size(); j++) {
            if (i == j) continue;
            /* Create a numerator of lagrangian multiplier */
            numerator.append("(x-").append(xData.get(j)).append(")");
            /* Calculate a dominator */
            dominator *= (xData.get(i) - xData.get(j));
        }
        if (-1e-9 < dominator && dominator < 1e-9) {
            throw new IllegalArgumentException("Too small steps!");
        }
        return numerator.append("/").append(dominator);
    }

    public Function build() {
        if (xData.isEmpty() || yData.isEmpty()) {
            throw new IllegalArgumentException("Nodes didn't set!");
        }
        for (int i = 0; i < yData.size(); i++) {
            if (yData.get(i).isNaN() || yData.get(i).isInfinite()) {
                throw new IllegalArgumentException("Initial function undefined at x=" + xData.get(i));
            }
            /* Create a Lagrangian Polynomial */
            lagrangianPolynomial.append("+").append(yData.get(i))
                    .append("(")
                    .append(lagrangianMultiplier(i))
                    .append(")");
        }
        return new Function(lagrangianPolynomial.toString());
    }
}
```

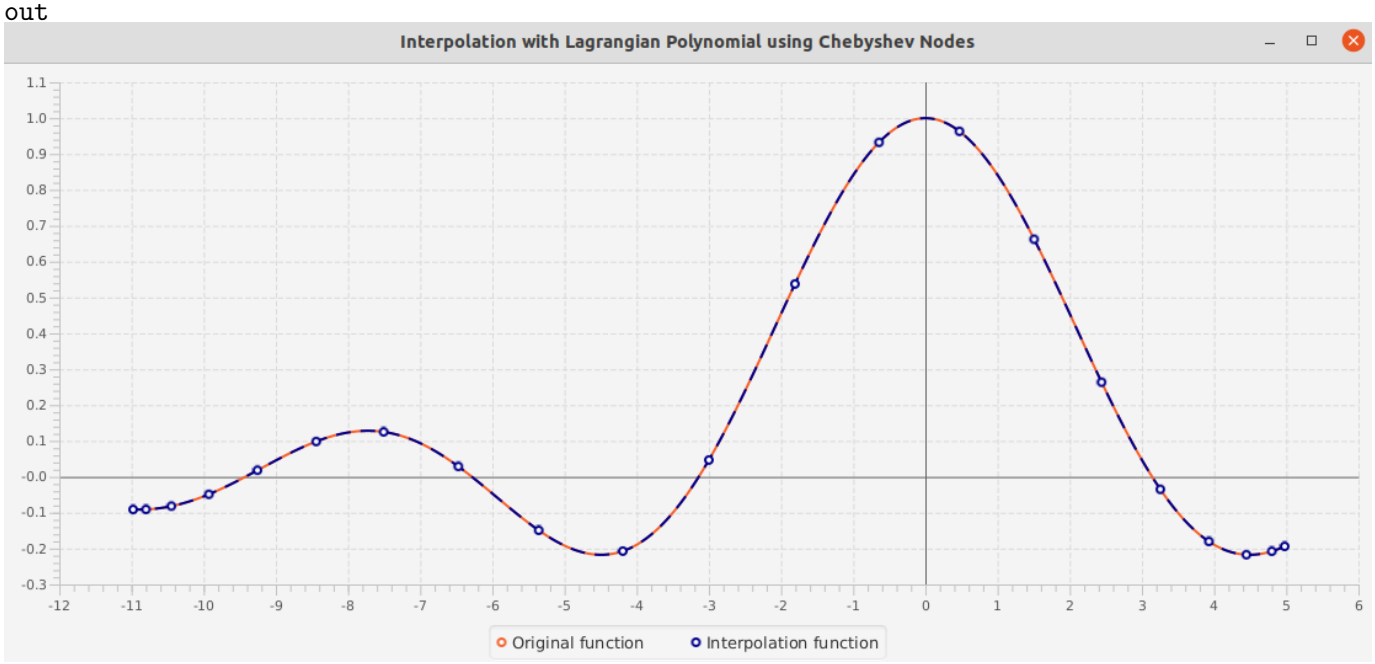
Примеры и результаты работы программы на разных данных

in

application.properties

```
# ================================
# = CHEBYSHEV'S POLYNOMIAL
# ================================
# Initial function
interpolation.initial.function=sin(x)/x
# Interpolation interval [a,b]
interpolation.interval.a=-11
interpolation.interval.b=5
# Number of nodes
interpolation.node=21
# Use Chebyshev's Nodes for interpolation
interpolation.chebyshev=true
```

out



Вывод

Узлы Чебышева позволяется уменшить погрешности интерполяционного многочлена Лагранжа, так как относительно большее количества узлов будут сконцентрированы на концах отрезка интерполирования, на которых обычно и находится самая высокая погрешность.