Описание метода, расчетные формулы

Интерполяция – способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений.

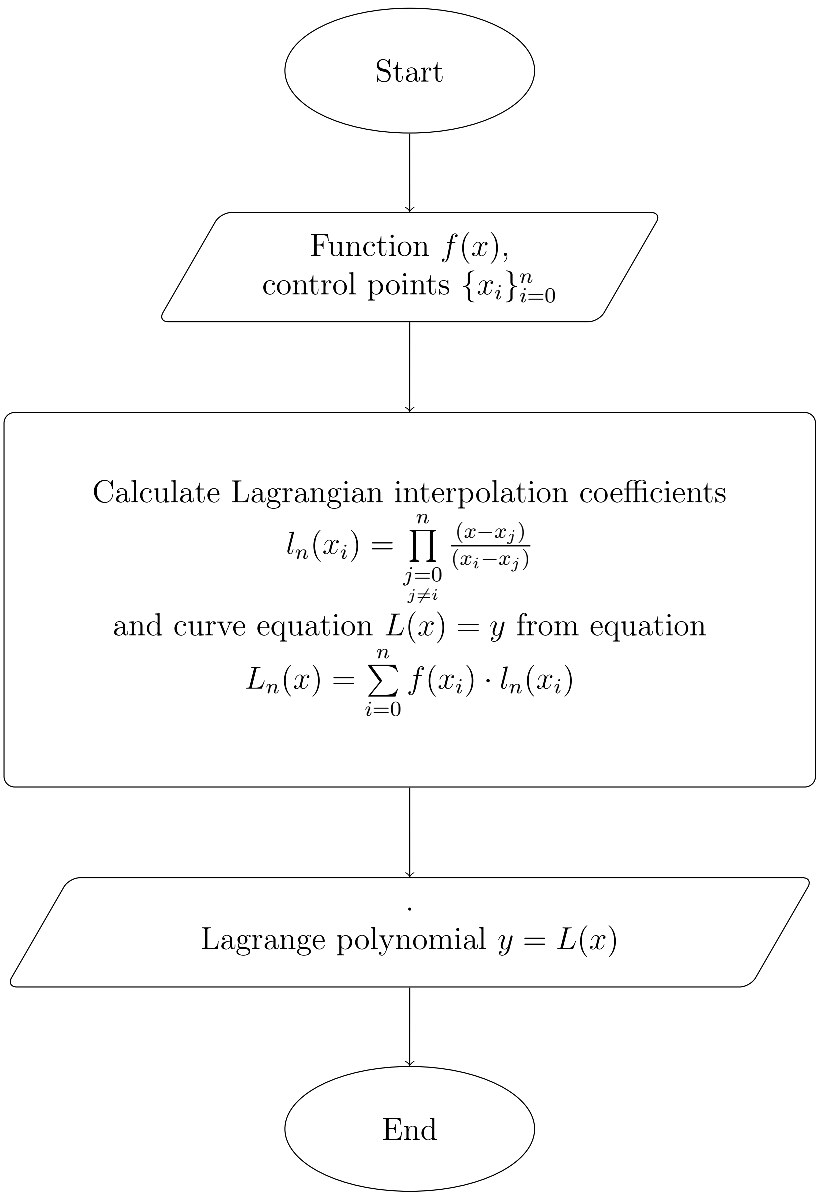
Интерполяционный полином Лагранжа имеет вид:

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

где $l_n(x_i)$ – множитель Лагранжа

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

Блок-схема численного метода



Листинг реализованного численного метода программы

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

public class LagrangePolynomialBuilder {
    private final Function function;
    private final StringBuilder lagrangePolynomial;
    private final List<Double> xData;
    private final List<Double> yData;

    public LagrangePolynomialBuilder(Function function) {
        if (function == null) {
            throw new IllegalArgumentException("Experimental function can not be null");
        }
        this.function = function;
        this.lagrangePolynomial = new StringBuilder();
        this.xData = new ArrayList<>(4);
        this.yData = new ArrayList<>(4);
    }

    public LagrangePolynomialBuilder experimentalData(Double... data) {
        if (data.length < 2) {
            throw new IllegalArgumentException("Experimental points can not be less than
            ↪  2");
        }
        for (Double point : data) {
            this.xData.add(point);
            this.yData.add(function.apply(point));
        }
        return this;
    }

    private StringBuilder lagrangeMultiplier(int i) {
        StringBuilder numerator = new StringBuilder();
        Double dominator = 1.0D;
        for (int j = 0; j < xData.size(); j++) {
            if (i == j) continue;
            /* Create a numerator of lagrange multiplier */
            numerator.append("(x-").append(xData.get(j)).append(")");
            /* Calculate a dominator value */
            dominator *= (xData.get(i) - xData.get(j));

            if (-1E-6 < dominator && dominator < 1E-6) {
                throw new IllegalArgumentException("Too small steps!");
            }
        }
        return numerator.append("/").append(dominator);
    }
```

```java
    public Function build() {
        if (xData.isEmpty() || yData.isEmpty()) {
            throw new IllegalArgumentException("Experimental data is empty!");
        }
        for (int i = 0; i < yData.size(); i++) {
            if (yData.get(i).isNaN() || yData.get(i).isInfinite()) {
                throw new IllegalArgumentException("Experimental function undefined at x=" + xData.get(i));
            }
            lagrangePolynomial.append("+").append(yData.get(i))
                    .append("(")
                    .append(lagrangeMultiplier(i))
                    .append(")");
        }
        return new Function(lagrangePolynomial.toString());
    }
}
```

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

public class Function {
    private final Expression expression;

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

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

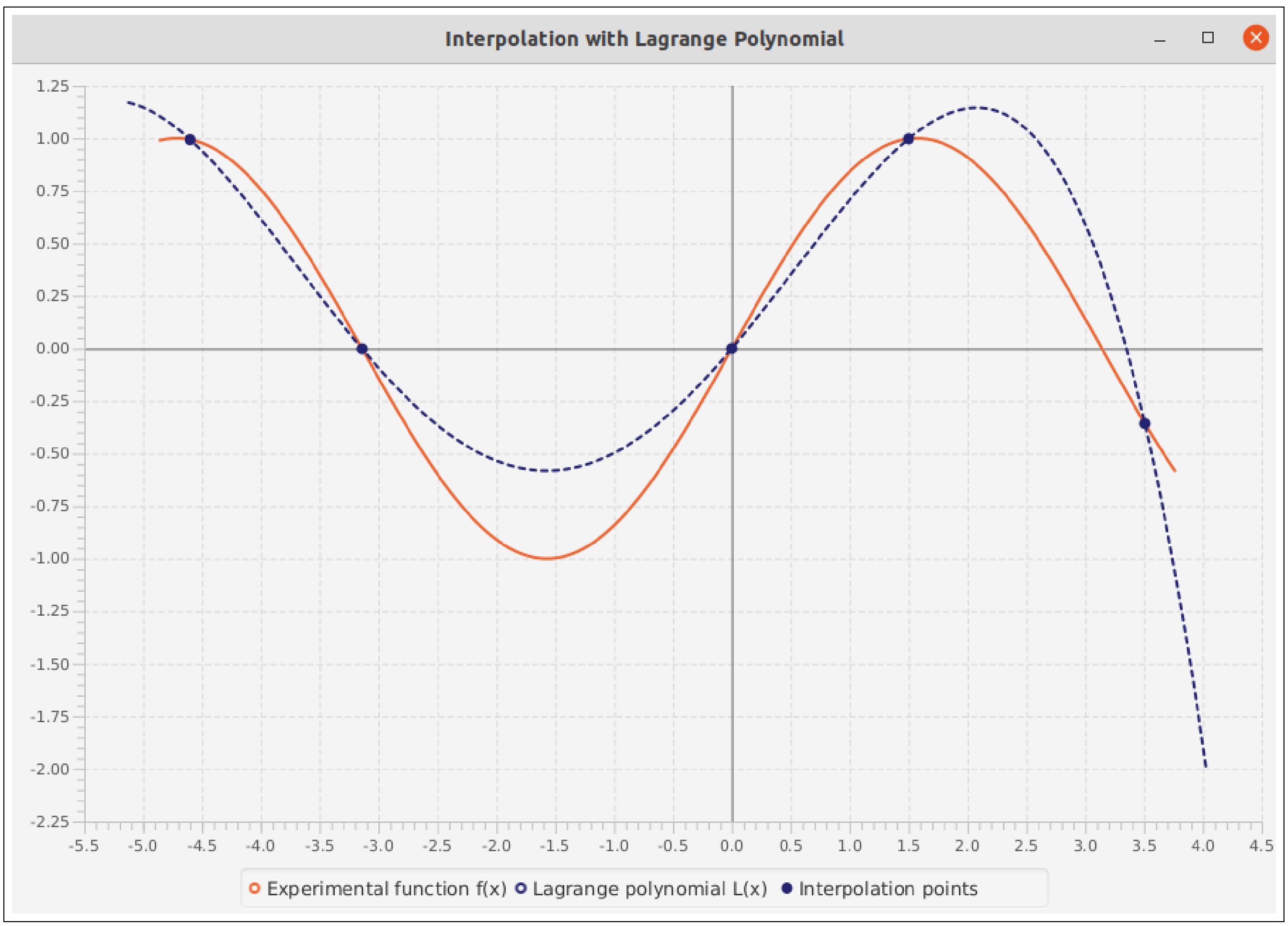
Примеры и результаты работы программы на разных данных

Входные данные

experimental-data.properties

```
# ================================
# = INTERPOLATION DATA
# ================================
# Experimental function and points
experimental.function=sin(x)
experimental.points=[-4.6,3.507,0.0,-3.14,1.5]
# Interpolation points
interpolation.points=[]
# ================================
# = AXIS BOUNDS
# ================================
# Axis lower and upper bounds [lowerbound,upperbound]
bounds.axis.x=[]
bounds.axis.y=[]
```

Результат



Вывод

Интерполирование многочленом Ньютона:

- *Применяется только для таблиц с равноудалёнными узлами.*
- *При добавлении новой точки отсутствует необходимость пересчитывать все коэффициенты заново.*

Интерполирование многочленом Лагранжа:

- *Применяется для таблиц с разноудалёнными узлами.*
- *С изменением числа узлов, приходится проводить все вычисления заново.*

Интерполирование кубическими сплайнами:

- *Применятся только для непрерывных функции.*
- *Степень многочленов не зависит от число узлов сетки.*