федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа №6

Вариант 7

Студент: Кулагин Вячеслав Дмитриевич
Преподаватель: Наумова Надежда Александровна

Группа: P3209

Санкт-Петербург
2025

Содержание

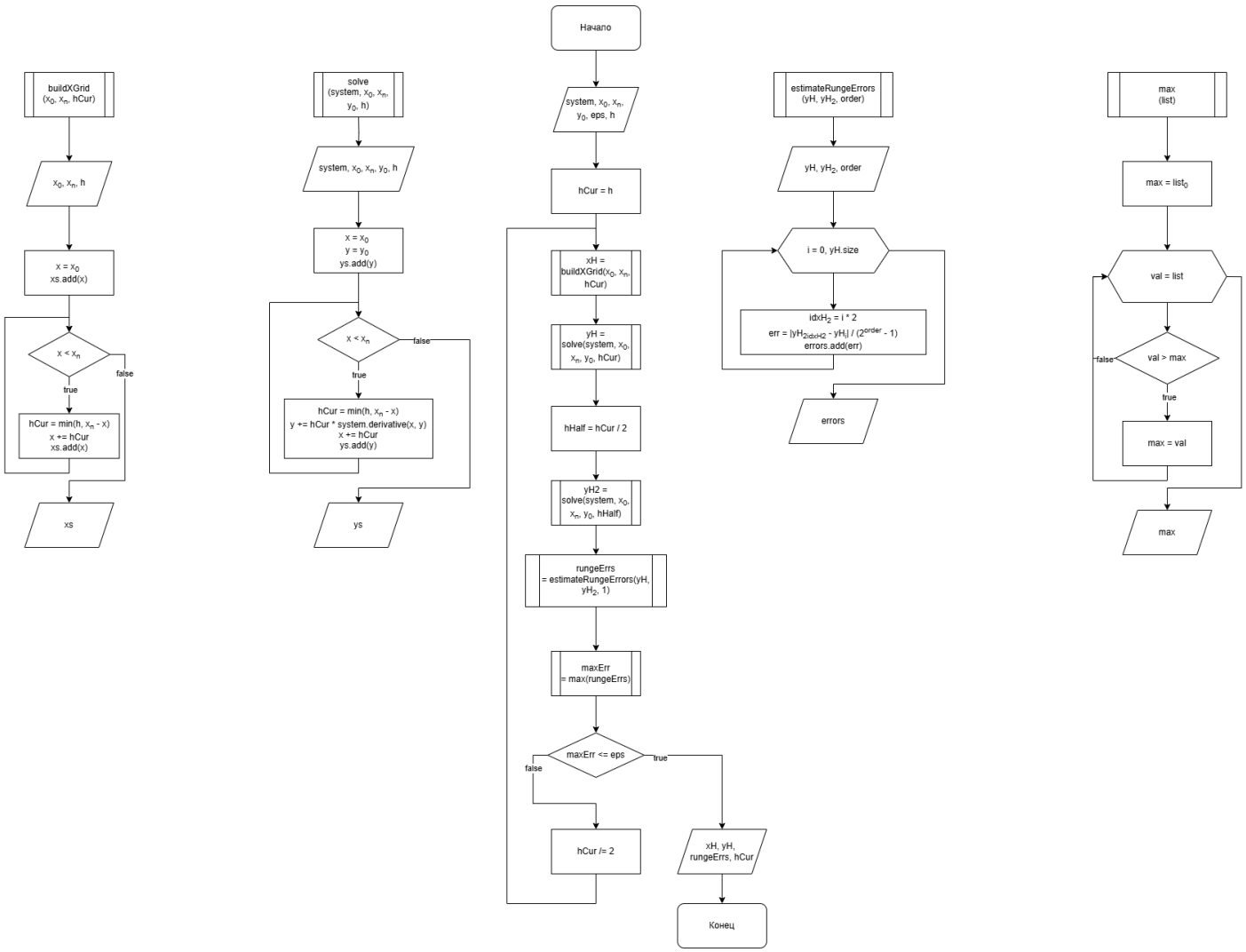
1 Цели работы

решить задачу Коши для обыкновенных дифференциальных уравнений численными методами.

2 Программная реализация

2.1 Метод Эйлера

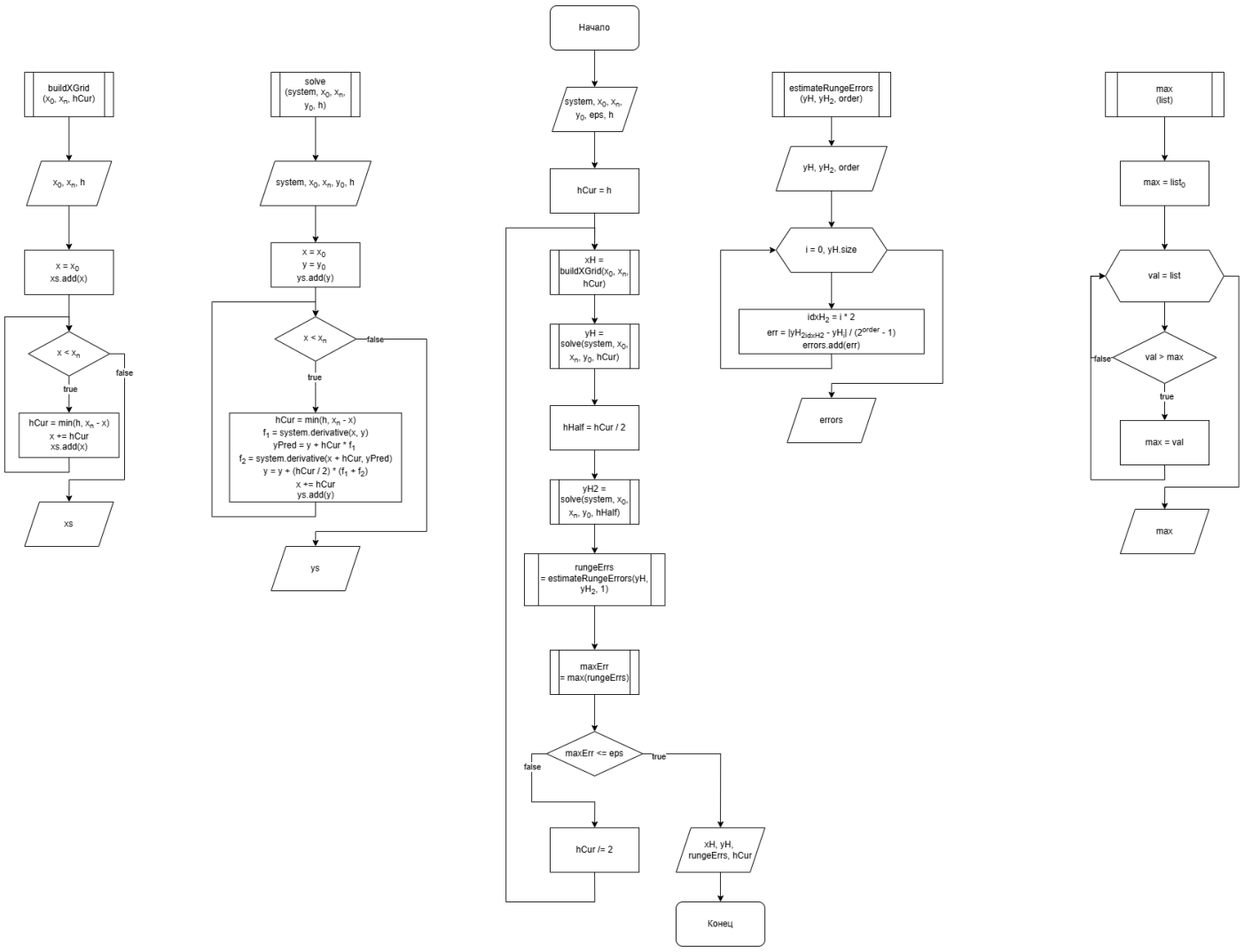
```
public static List<Double> solve(ODESystem system, double x0, double xn, double y0,
    List<Double> ys = new ArrayList<>();
    double x = x0;
    double y = y0;
    ys.add(y);

    while (x < xn - 1e-12) {
        double hCur = Math.min(h, xn - x);
        y += hCur * system.derivative(x, y);
        x += hCur;
        ys.add(y);
    }
    return ys;
}

public static RungeResult solveWithRunge(ODESystem system,
                                         double x0, double xn,
                                         double y0, double eps,
                                         double h) {
    double hCur = h;

    while (true) {
        List<Double> xH = RungeHelper.buildXGrid(x0, xn, hCur);
        List<Double> yH = solve(system, x0, xn, y0, hCur);

        double hHalf = hCur / 2.0;
        List<Double> yH2 = solve(system, x0, xn, y0, hHalf);

        List<Double> rungeErrs = RungeHelper.estimateRungeErrors(yH, yH2, 1);
        double maxErr = RungeHelper.max(rungeErrs);

        if (maxErr <= eps) {
            return new RungeResult(xH, yH, rungeErrs, hCur);
        } else {
            hCur /= 2.0;
        }
    }
}
```



2.2 Модифицированный метод Эйлера

```
public static List<Double> solve(ODESystem system, double x0, double xn, double y0,
    List<Double> ys = new ArrayList<>();
    double x = x0;
    double y = y0;
    ys.add(y);

    while (x < xn - 1e-12) {
        double hCur = Math.min(h, xn - x);

        double f1 = system.derivative(x, y);
        double yPred = y + hCur * f1;
        double f2 = system.derivative(x + hCur, yPred);
        y = y + (hCur / 2.0) * (f1 + f2);

        x += hCur;
        ys.add(y);
    }
    return ys;
```

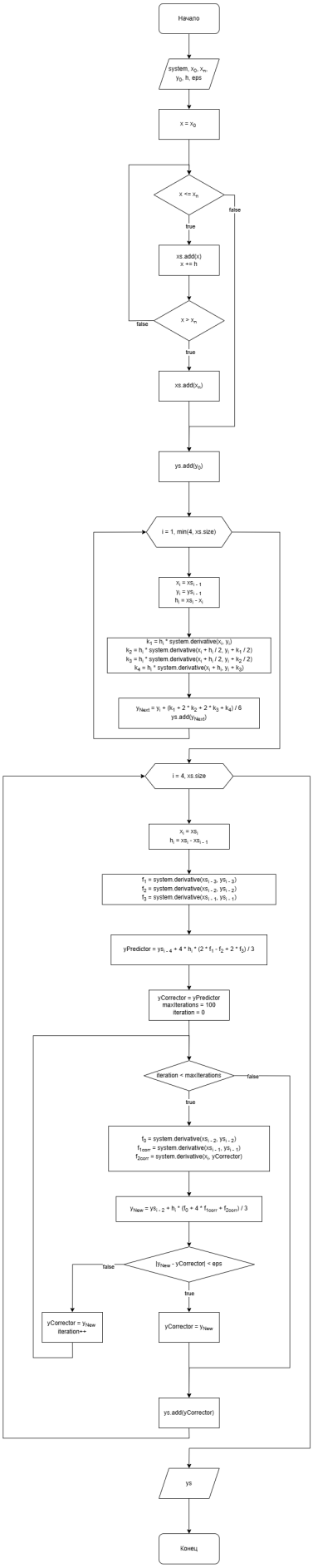
```
}

public static RungeResult solveWithRunge(ODESystem system,
                                         double x0, double xn,
                                         double y0, double eps,
                                         double h) {
    double hCur = h;

    while (true) {
        List<Double> xH = RungeHelper.buildXGrid(x0, xn, hCur);
        List<Double> yH = solve(system, x0, xn, y0, hCur);

        double hHalf = hCur / 2.0;
        List<Double> yH2 = solve(system, x0, xn, y0, hHalf);

        List<Double> rungeErrs = RungeHelper.estimateRungeErrors(yH, yH2, 2);
        double maxErr = RungeHelper.max(rungeErrs);

        if (maxErr <= eps) {
            return new RungeResult(xH, yH, rungeErrs, hCur);
        } else {
            hCur /= 2.0;
        }
    }
}
```



2.3 Метод Милна

```
public static List<Double> solve(ODESystem system, double x0, double xn, double y0,
    List<Double> xs = new ArrayList<>();
    List<Double> ys = new ArrayList<>();

    double x = x0;
    while (x <= xn + 1e-12) {
        xs.add(x);
        x += h;
        if (x > xn + 1e-12) {
            xs.add(xn);
            break;
        }
    }

    ys.add(y0);
    for (int i = 1; i < Math.min(4, xs.size()); i++) {
        double xi = xs.get(i - 1);
        double yi = ys.get(i - 1);
```

```
            double hi = xs.get(i) - xi;

            double k1 = hi * system.derivative(xi, yi);
            double k2 = hi * system.derivative(xi + hi / 2, yi + k1 / 2);
            double k3 = hi * system.derivative(xi + hi / 2, yi + k2 / 2);
            double k4 = hi * system.derivative(xi + hi, yi + k3);

            double yNext = yi + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
            ys.add(yNext);
        }

        for (int i = 4; i < xs.size(); i++) {
            double xi = xs.get(i);
            double hi = xs.get(i) - xs.get(i - 1);

            double f1 = system.derivative(xs.get(i - 3), ys.get(i - 3));
            double f2 = system.derivative(xs.get(i - 2), ys.get(i - 2));
            double f3 = system.derivative(xs.get(i - 1), ys.get(i - 1));

            double yPredictor = ys.get(i - 4) + 4 * hi * (2 * f1 - f2 + 2 * f3) / 3;

            double yCorrector = yPredictor;
            int maxIterations = 100;
            int iteration = 0;

            while (iteration < maxIterations) {
                double f0 = system.derivative(xs.get(i - 2), ys.get(i - 2));
                double f1_corr = system.derivative(xs.get(i - 1), ys.get(i - 1));
                double f2_corr = system.derivative(xi, yCorrector);

                double yNew = ys.get(i - 2) + hi * (f0 + 4 * f1_corr + f2_corr) / 3;

                if (Math.abs(yNew - yCorrector) < eps) {
                    yCorrector = yNew;
                    break;
                }
                yCorrector = yNew;
                iteration++;
            }

            ys.add(yCorrector);
        }

        return ys;
    }
```


Начало
system, x_0, x_n, y_0, h, eps
$x = x_0$
$x <= x_n$
false
true
xs.add(x)
$x += h$
$x > x_n$
false
true
xs.add(x_n)
ys.add(y_0)
i = 1, min(4, xs.size)
$x_i = xs_{i-1}$
$y_i = ys_{i-1}$
$h_i = xs_i - x_i$
$k_1 = h_i * system.derivative(x_i, y_i)$
$k_2 = h_i * system.derivative(x_i + h_i / 2, y_i + k_1 / 2)$
$k_3 = h_i * system.derivative(x_i + h_i / 2, y_i + k_2 / 2)$
$k_4 = h_i * system.derivative(x_i + h_i, y_i + k_3)$
$y_{iNext} = y_i + (k_1 + 2 * k_2 + 2 * k_3 + k_4) / 6$
ys.add(y_{iNext})
i = 4, xs.size
$x_i = xs_i$
$h_i = xs_i - xs_{i-1}$
$f_1 = system.derivative(xs_{i-3}, ys_{i-3})$
$f_2 = system.derivative(xs_{i-2}, ys_{i-2})$
$f_3 = system.derivative(xs_{i-1}, ys_{i-1})$
$yPredictor = ys_{i-4} + 4 * h_i * (2 * f_1 - f_2 + 2 * f_3) / 3$
yCorrector = yPredictor
maxIterations = 100
iteration = 0
iteration < maxIterations
false
true
$f_0 = system.derivative(xs_{i-3}, ys_{i-3})$
$f_{1corr} = system.derivative(xs_{i-1}, ys_{i-1})$
$f_{2corr} = system.derivative(x_i, yCorrector)$
$y_{New} = ys_{i-2} + h_i * (f_0 + 4 * f_{1corr} + f_{2corr}) / 3$
$|y_{New} - yCorrector| < eps$
false
true
yCorrector = y_{New}
iteration++
yCorrector = y_{New}
ys.add(yCorrector)
ys
Конец

3 Вывод

Проведя эту работу, я реализовал решение задач численными методами (методом Эйлера, модифицированным методом Эйлера, методом Милна). Также удалось их реализовать и построить графики для сравнения.