Университет ИТМО
МФ КТиУ, Ф ПИиКТ

Лабораторная работа №4
Дисциплина «Вычислительная математика»

Решение ОДУ. Задача Коши.

Вариант:
Метод Адамса

Выполнил:
Студент группы P3212
Анищенко Анатолий Алексеевич

Преподаватель:
Перл Ольга Вячеславовна

г. Санкт-Петербург
2020 г.

Цель работы:

Реализовать метод Адамса для решения задачи Коши.

Описание использованного метода:

Метод Адамса – многошаговый метод 4го порядка точности. Используемая в данном методе формула прогноза получена интегрированием обратной интерполяционной формулы Ньютона и имеет вид:

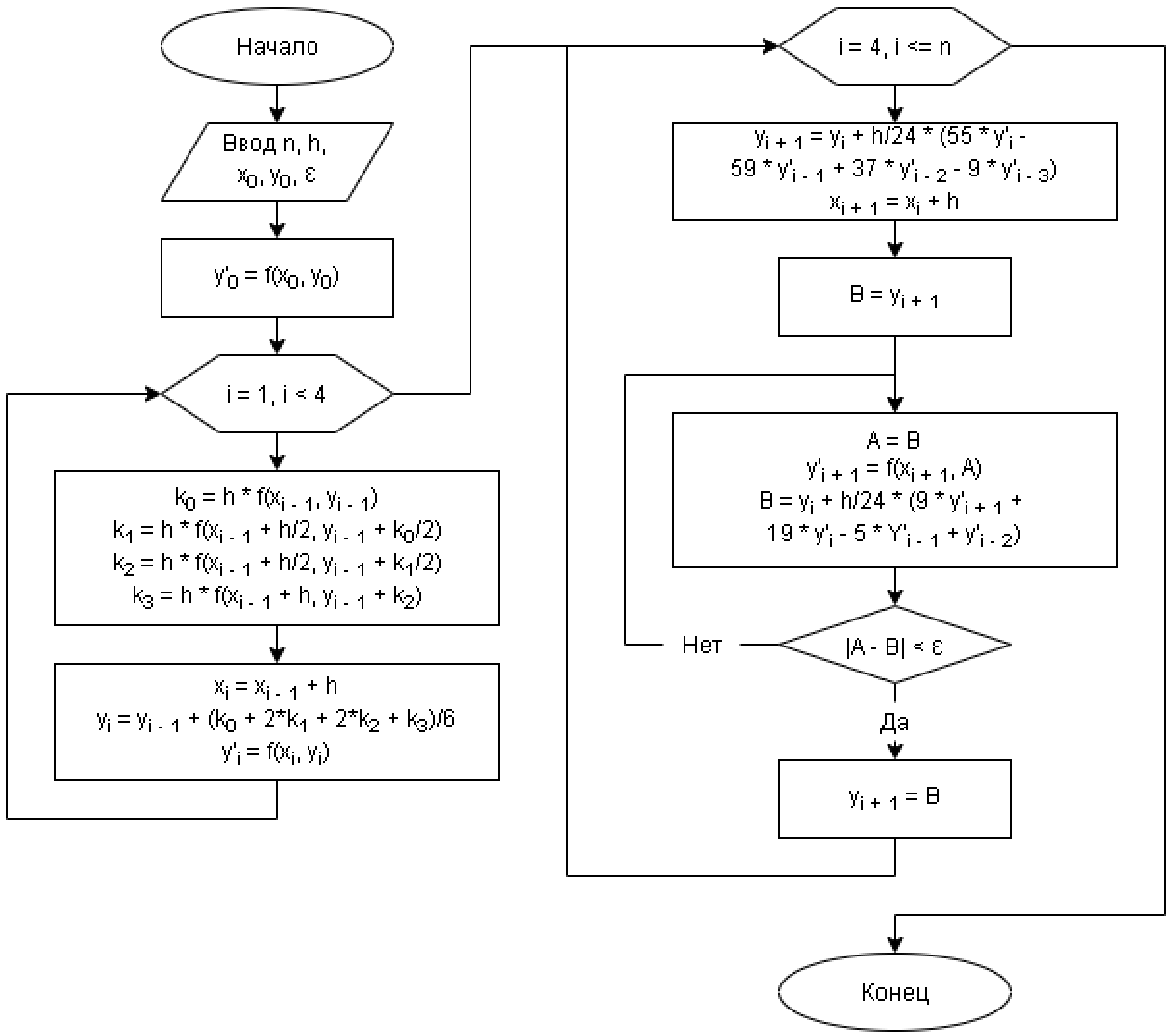
$$y_{i+1} = y_i + \frac{1}{24}h * (55y'_i - 59y'_{i-1} + 37y'_{i-2} - 9y'_{i-3})$$

На этапе коррекции используется формула:

$$y_{i+1} = y_i + \frac{1}{24}h * (9y'_{i+1} + 19y'_i - 5y'_{i-1} + y'_{i-2})$$

Данный метод имеет четвёртый порядок точности, использует в качестве интерполяционного многочлена полином Лагранжа. А также ошибка, полученная на очередном шаге, не имеет тенденцию к экспоненциальному росту.

Листинг численного метода:

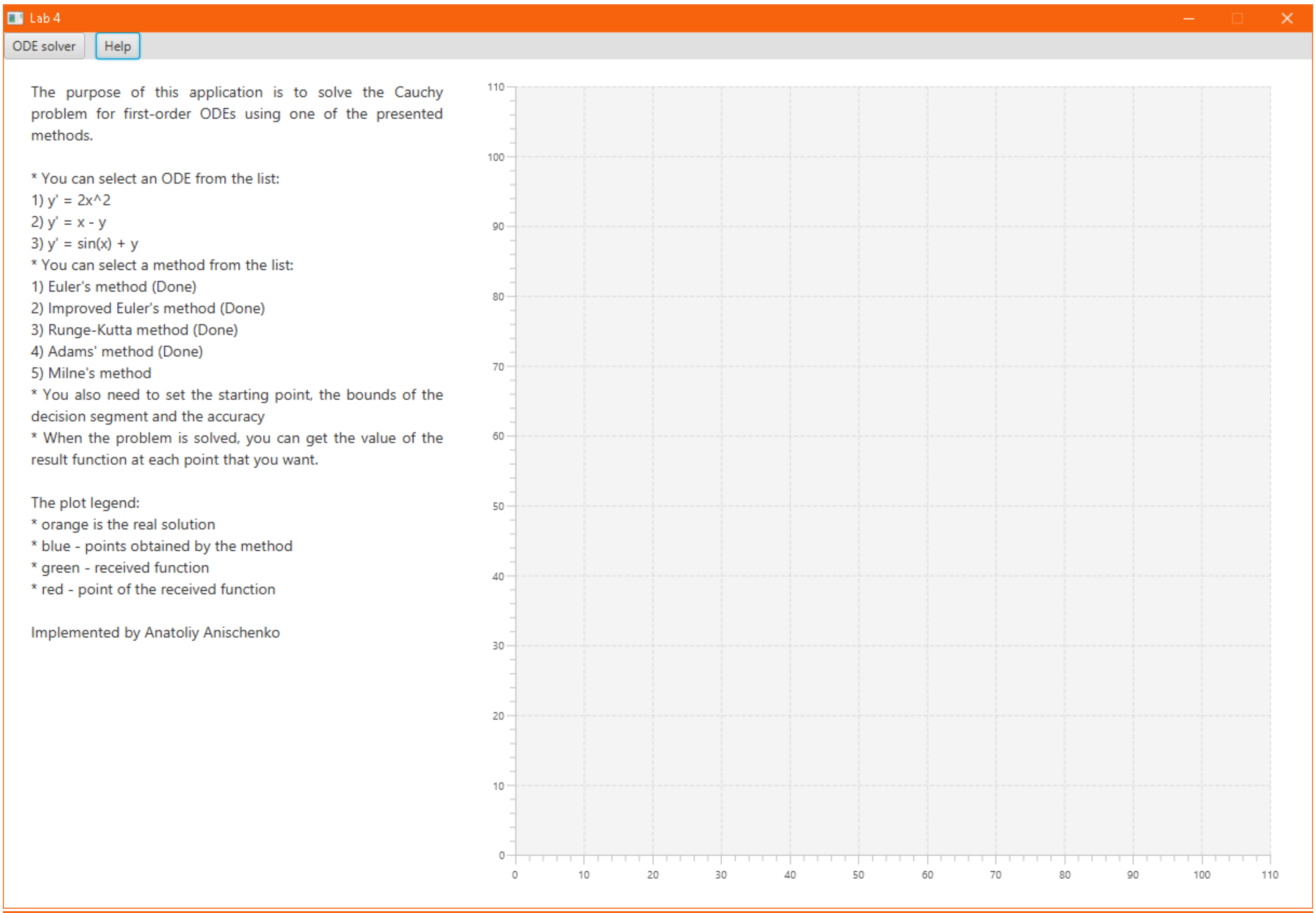


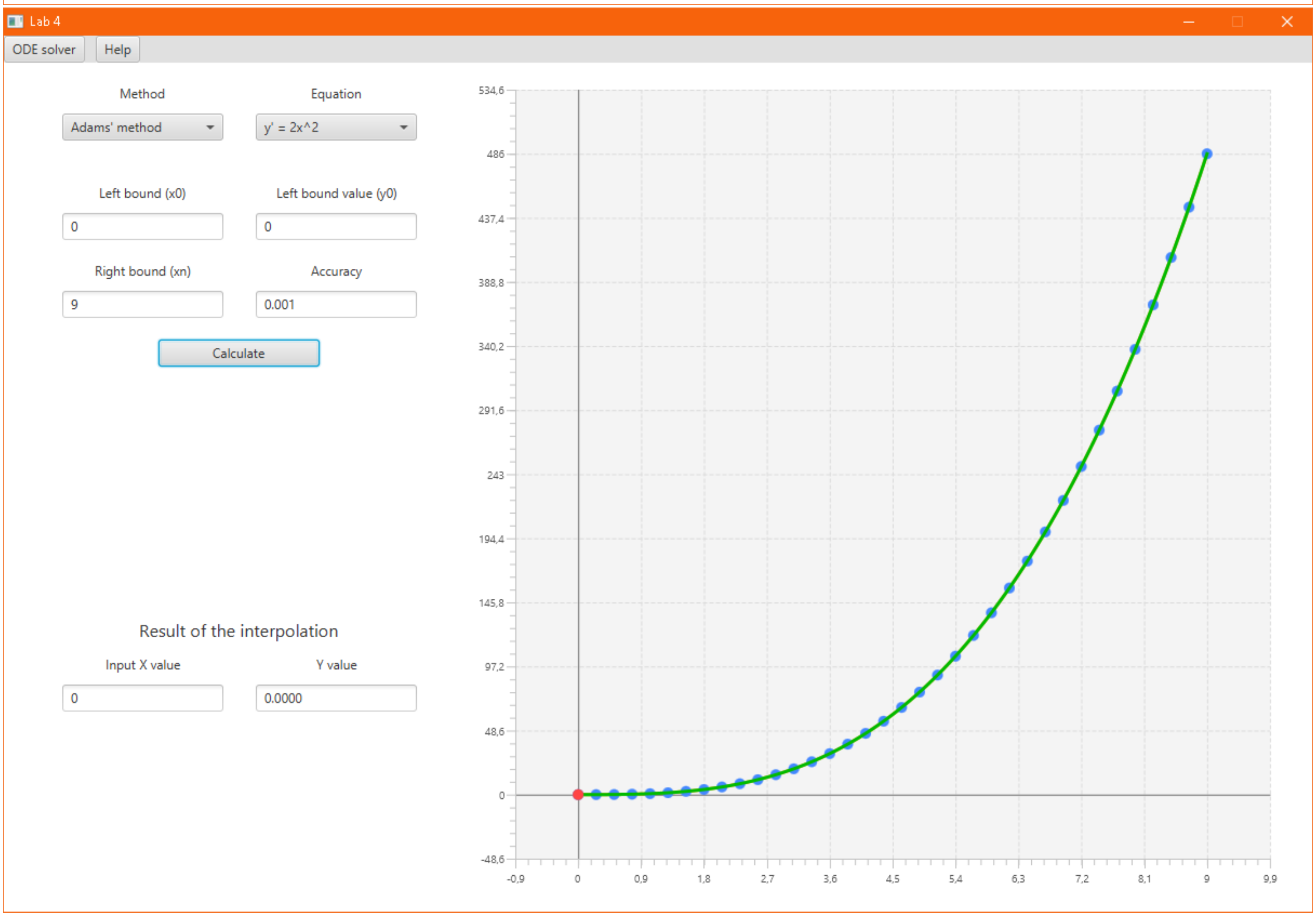
```java
static public OrdinaryDifferentialEquationSolverResult solveODE(
      OrdinaryDifferentialEquationMethodType methodType,
      OrdinaryDifferentialEquation equation,
      Point point0,
      double xn,
      double accuracy
) throws NotImplementedSolutionException, InvalidValueException {
   int count = methodType.getPointsCount(xn - point0.first, accuracy);
   double h = (xn - point0.first) / (count - 1);

   ArrayList<Point> points;
   switch (methodType) {
      case EULER_METHOD:
         points = eulerMethodSolution(equation, point0, h, count);
         break;
      case IMPROVED_EULER_METHOD:
         points = improvedEulerMethodSolution(equation, point0, h, count);
         break;
      case RUNGE_KUTTA_METHOD:
         points = rungeKuttaMethodSolution(equation, point0, h, count);
         break;
      case ADAMS_METHOD:
         points = adamsMethodSolution(equation, point0, h, count, accuracy);
         break;
      case MILNE_METHOD:
      default:
         throw new NotImplementedSolutionException();
   }

   if (count > 80) {
      throw new InvalidValueException("Can't solve ODE on this bounds with this accuracy.\n" +
            "Count of points for interpolation too big (" + count + ")");
   } else {
      return new OrdinaryDifferentialEquationSolverResult(
         InterpolationSolver.solveInterpolation(
            InterpolationMethodType.NEWTON_POLYNOMIAL,
            points
         ),
         points
      );
   }
}

private static ArrayList<Point> eulerMethodSolution(
      OrdinaryDifferentialEquation equation,
      Point point0,
      double h,
      int count
) {
   ArrayList<Point> points = new ArrayList<>();
   points.add(point0);

   double x = point0.first;
   double y = point0.second;

   for (int i = 1; i < count; i++) {
      y = y + h * equation.getValue(x, y);
      x += h;

      points.add(new Point(x, y));
   }
```

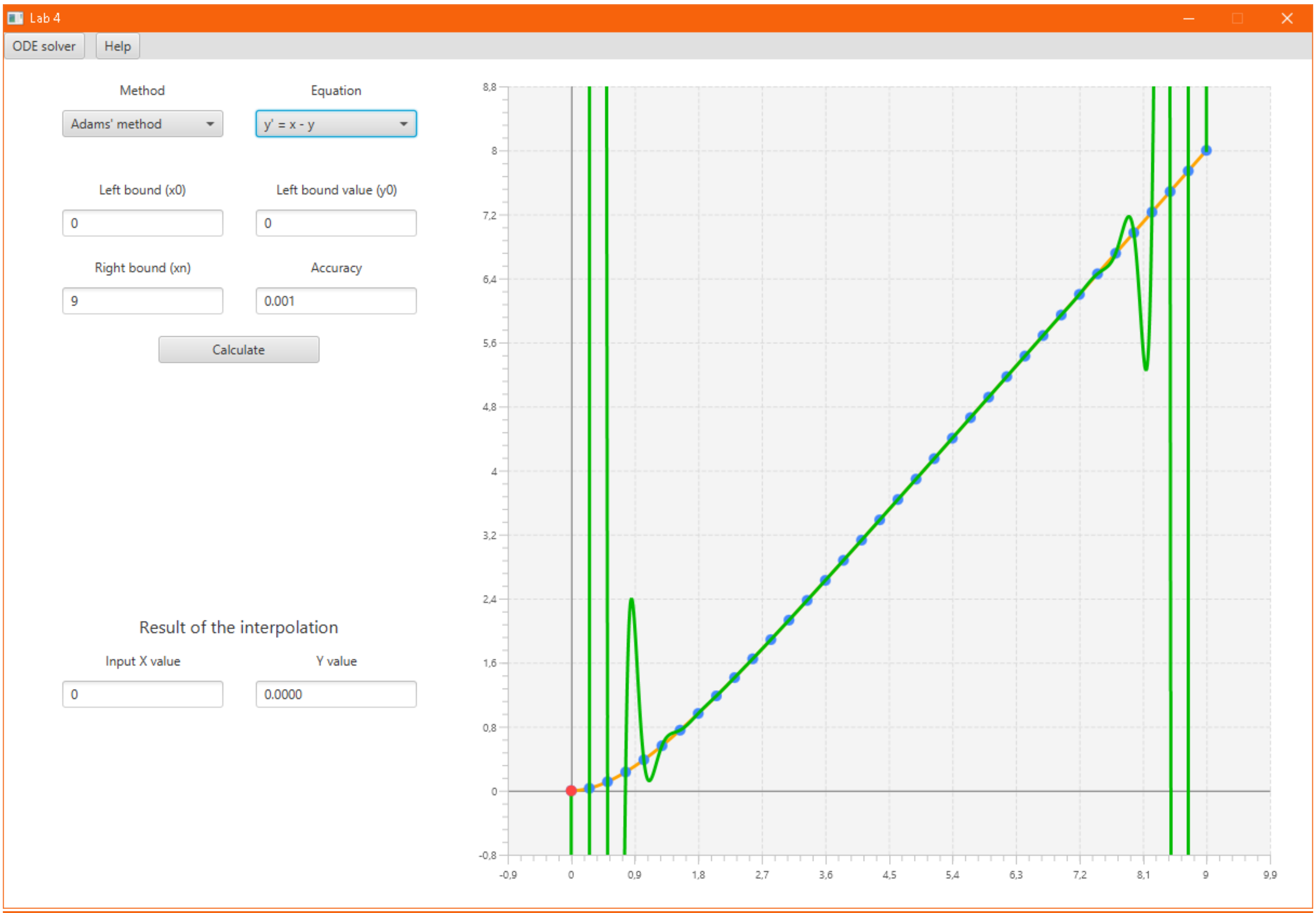
```
    return points;
}

private static ArrayList<Point> improvedEulerMethodSolution(
        OrdinaryDifferentialEquation equation,
        Point point0,
        double h,
        int count
) {
    ArrayList<Point> points = new ArrayList<>();
    points.add(point0);

    double x = point0.first;
    double y = point0.second;

    for (int i = 1; i < count; i++) {
        double stepY = y + h * equation.getValue(x, y);
        y = y + h * (equation.getValue(x, y) + equation.getValue(x + h, stepY)) / 2;
        x += h;

        points.add(new Point(x, y));
    }

    return points;
}

private static ArrayList<Point> rungeKuttaMethodSolution(
        OrdinaryDifferentialEquation equation,
        Point point0,
        double h,
        int count
) {
    ArrayList<Point> points = new ArrayList<>();
    points.add(point0);

    double x = point0.first;
    double y = point0.second;

    for (int i = 1; i < count; i++) {
        double k0 = h * equation.getValue(x, y);
        double k1 = h * equation.getValue(x + h / 2, y + k0 / 2);
        double k2 = h * equation.getValue(x + h / 2, y + k1 / 2);
        double k3 = h * equation.getValue(x + h, y + k2);

        y = y + (k0 + 2 * k1 + 2 * k2 + k3) / 6;
        x += h;

        points.add(new Point(x, y));
    }

    return points;
}

private static ArrayList<Point> adamsMethodSolution(
        OrdinaryDifferentialEquation equation,
        Point point0,
        double h,
        int count,
        double accuracy
) {
```

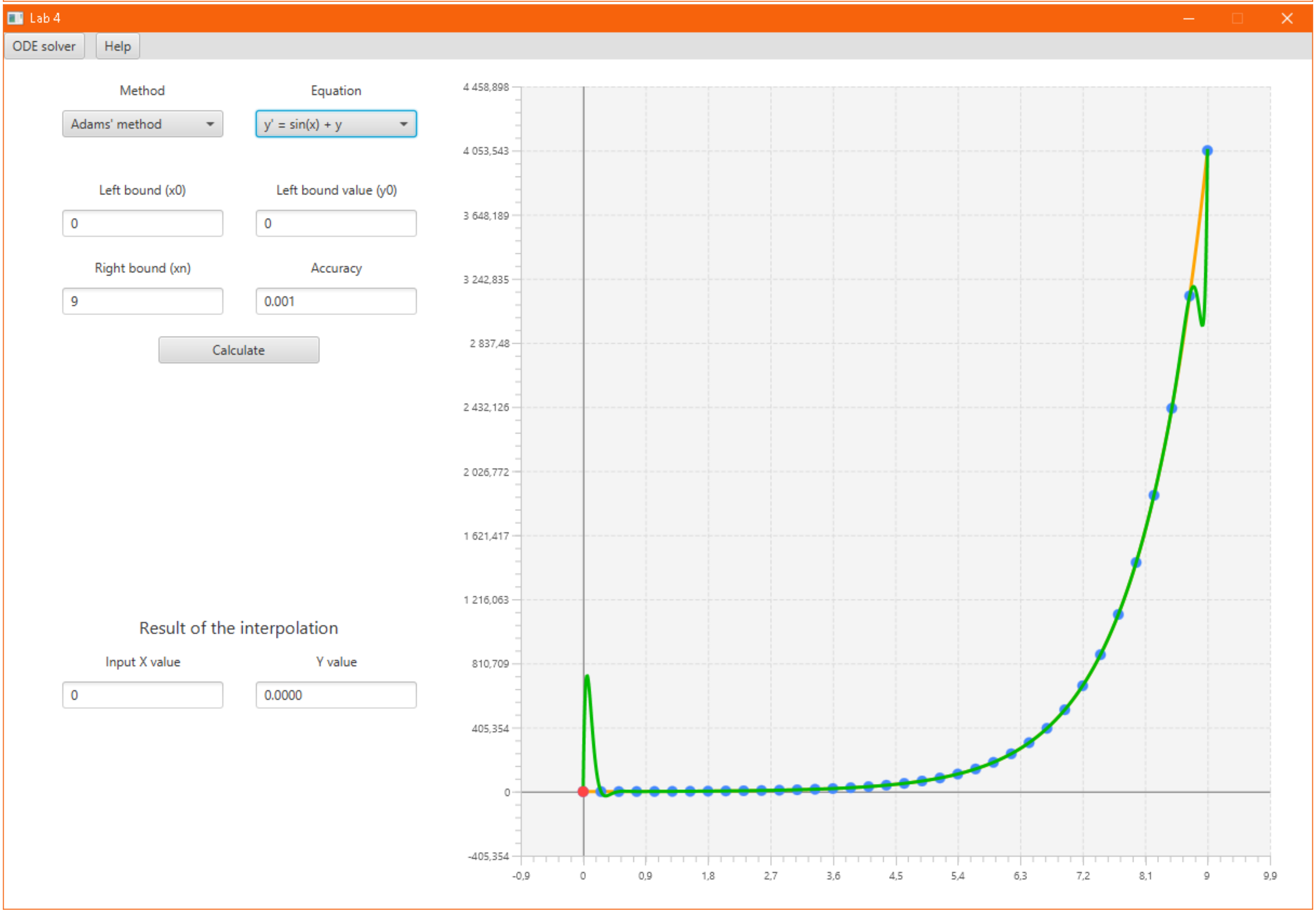
```
    ArrayList<Point> points = new ArrayList<>(rungeKuttaMethodSolution(equation, point0, h, 4));

    double[] yDerivative = new double[count];

    for (int i = 0; i < 4; i++) {
       yDerivative[i] = equation.getValue(points.get(i).first, points.get(i).second);
    }

    double x = points.get(3).first;
    double y = points.get(3).second;

    for (int i = 4; i < count; i++) {
       double yPredicted = y + h * (55 * yDerivative[i - 1] - 59 * yDerivative[i - 2]
             + 37 * yDerivative[i - 3] - 9 * yDerivative[i - 4]) / 24;
       x += h;

       double prevY;
       double newY = yPredicted;

       do {
          prevY = newY;
          yDerivative[i] = equation.getValue(x, prevY);
          newY = y + h * (9 * yDerivative[i] + 19 * yDerivative[i - 1]
                - 5 * yDerivative[i - 2] + yDerivative[i - 3]) / 24;
       } while (Math.abs(prevY - newY) > accuracy);

       y = newY;

       points.add(new Point(x, y));
    }

    return points;
}
```

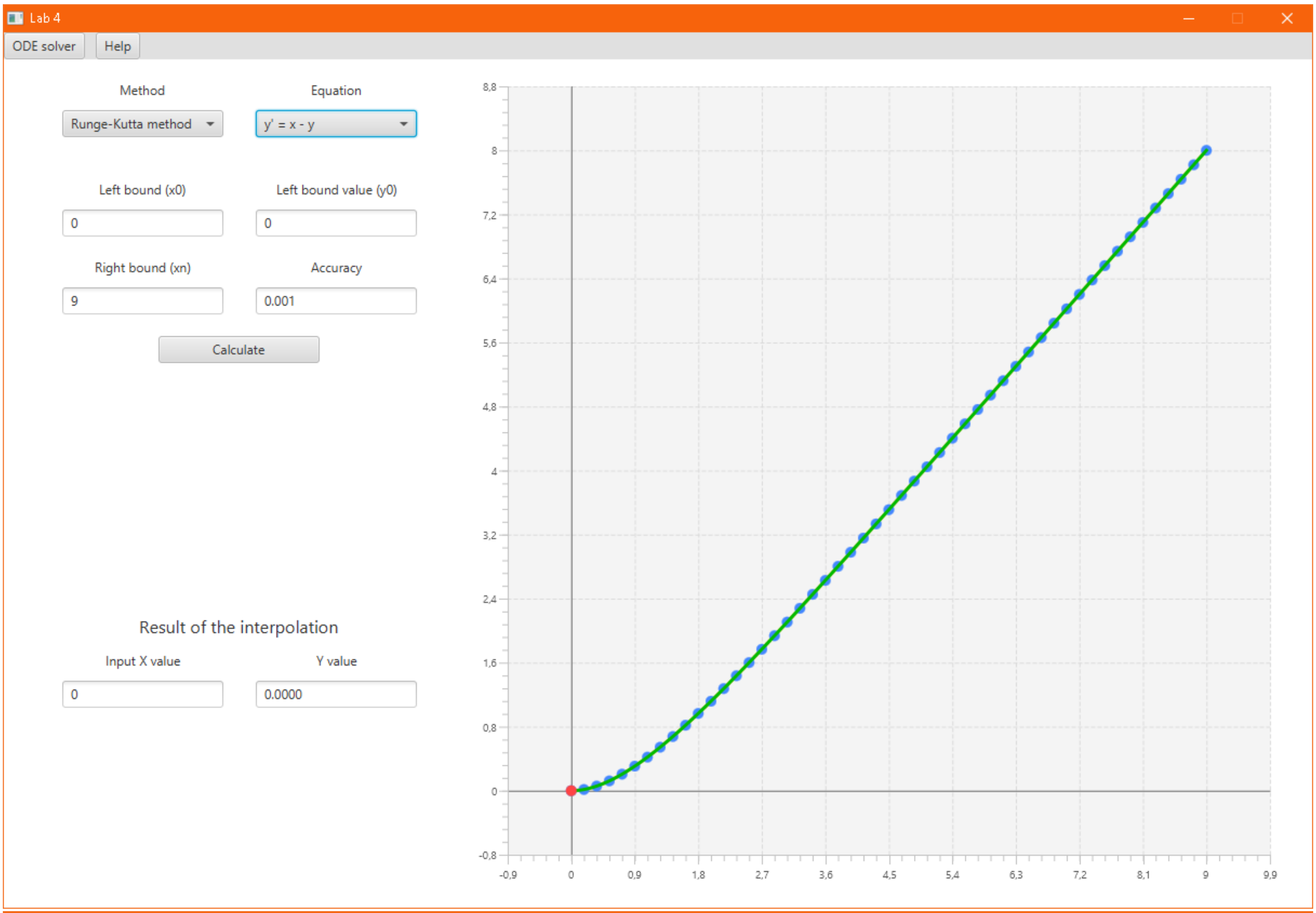
Результат работы:

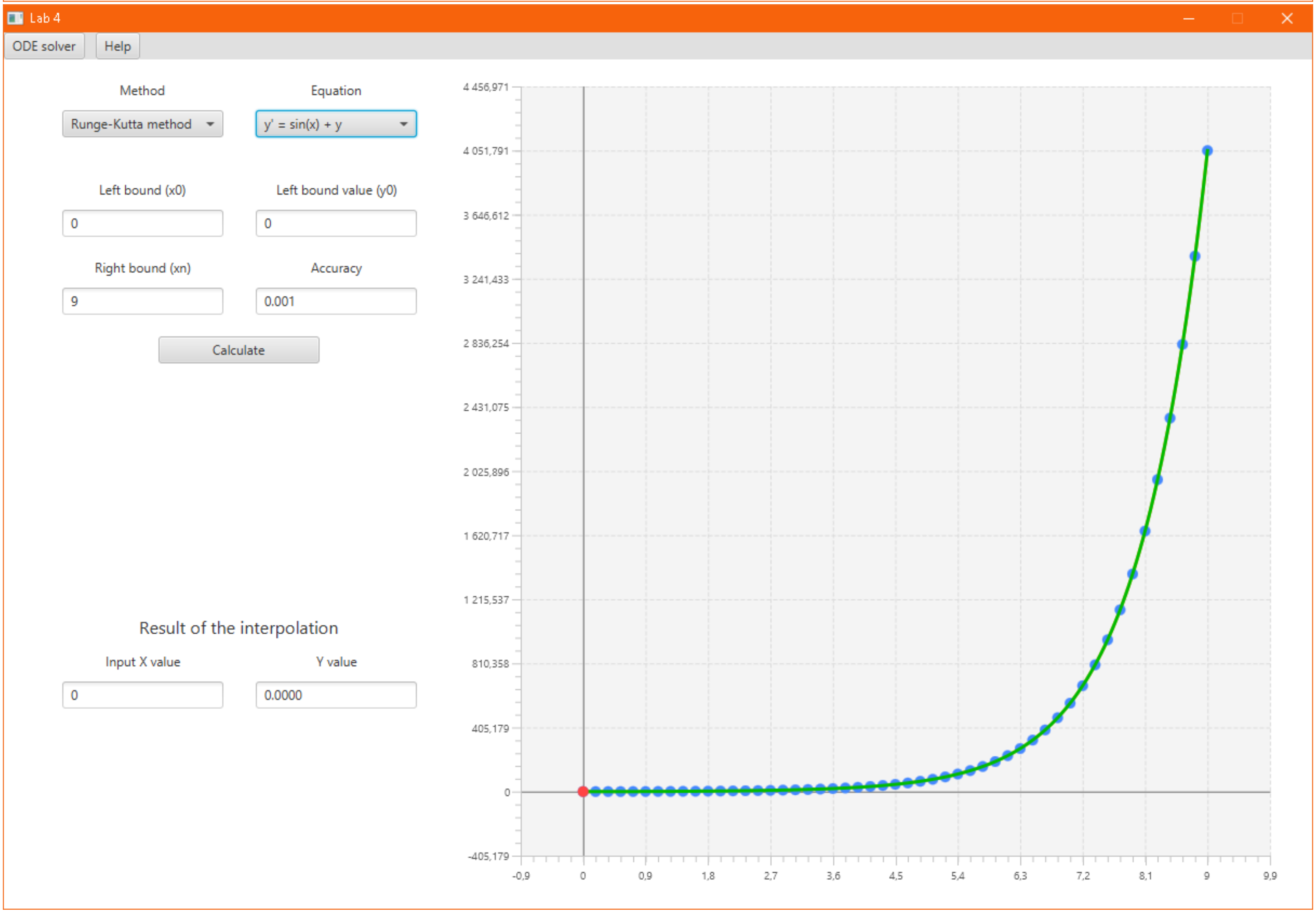


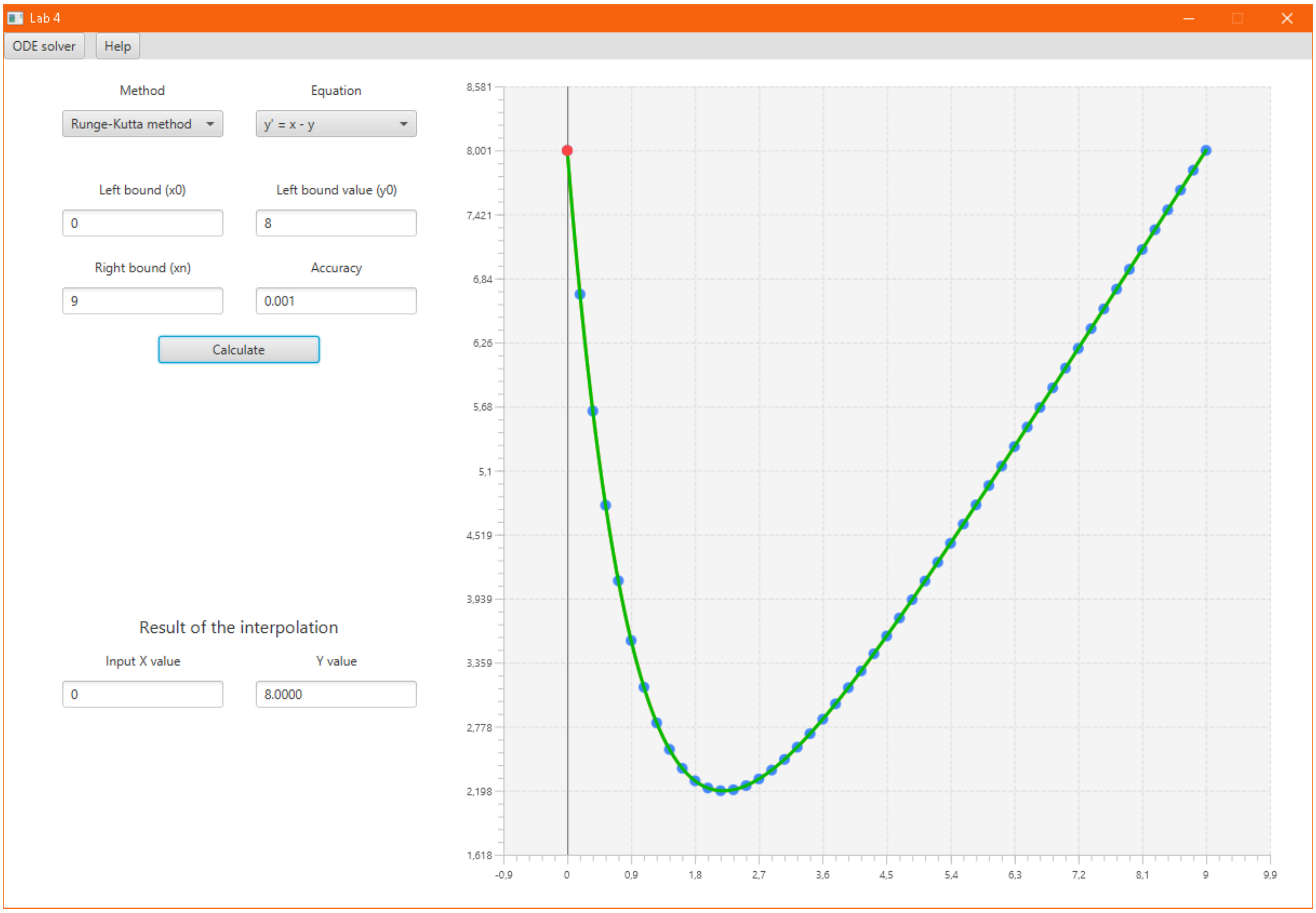



Lab 4
ODE solver
Help
Method
Equation
Adams' method
y' = x - y
Left bound (x0)
Left bound value (y0)
0
0
Right bound (xn)
Accuracy
9
0.001
Calculate
Result of the interpolation
Input X value
Y value
0
0.0000
8,8
8
7,2
6,4
5,6
4,8
4
3,2
2,4
1,6
0,8
0
-0,8
-0,9
0
0,9
1,8
2,7
3,6
4,5
5,4
6,3
7,2
8,1
9
9,9


Lab 4
ODE solver
Help
Method
Equation
Adams' method
y' = sin(x) + y
Left bound (x0)
Left bound value (y0)
0
0
Right bound (xn)
Accuracy
9
0.001
Calculate
Result of the interpolation
Input X value
Y value
0
0.0000
4 458,898
4 053,543
3 648,189
3 242,835
2 837,48
2 432,126
2 026,772
1 621,417
1 216,063
810,709
405,354
0
-405,354
-0,9
0
0,9
1,8
2,7
3,6
4,5
5,4
6,3
7,2
8,1
9
9,9


Lab 4
ODE solver
Help
Method
Runge-Kutta method
Equation
y' = x - y
Left bound (x0)
0
Left bound value (y0)
0
Right bound (xn)
9
Accuracy
0.001
Calculate
Result of the interpolation
Input X value
0
Y value
0.0000
8,8
8
7,2
6,4
5,6
4,8
4
3,2
2,4
1,6
0,8
0
-0,8
-0,9
0
0,9
1,8
2,7
3,6
4,5
5,4
6,3
7,2
8,1
9
9,9


Lab 4
ODE solver
Help
Method
Runge-Kutta method
Equation
y' = sin(x) + y
Left bound (x0)
0
Left bound value (y0)
0
Right bound (xn)
9
Accuracy
0.001
Calculate
Result of the interpolation
Input X value
0
Y value
0.0000
4 456,971
4 051,791
3 646,612
3 241,433
2 836,254
2 431,075
2 025,896
1 620,717
1 215,537
810,358
405,179
0
-405,179
-0,9
0
0,9
1,8
2,7
3,6
4,5
5,4
6,3
7,2
8,1
9
9,9



Выводы:

Существуют одношаговые методы и многошаговы методы для решения задачи Коши. Различие этих типов методов заключается в том, что в первом случае для подсчёта очередной точки мы используем только последнюю полученную. В многошаговых методах используются несколько предыдущих значений. Для многошаговых методов требуется больше памяти и времени, но результат получается точнее. Хотя методы и имеет формально четвёртый порядок точности, на самом деле, точность выше за счёт формул коррекции.

Сравнение одношаговых методов:

1) Метод Эйлера:
   Первый порядок точности, погрешность накапливается с каждой точкой.
2) Усовершенствованный метод Эйлера:
   Второй порядок точности, погрешность накапливается с каждой точкой.
3) Метод Рунге-Кутта 4ого порядка:
   Четвёртый порядок точности, больше вычислений по сравнению с первыми двумя методами.

Сравнение многошаговых методов:

1) Метод Милна:
   Четвёртый порядок точности, использует в качестве интерполяционного многочлена полином Ньютона. Ошибка, полученная на очередном шаге, имеет тенденцию к экспоненциальному росту.

2) Метод Адамса-Башфорта:
   Четвёртый порядок точности, использует в качестве интерполяционного многочлена полином Лагранжа. Ошибка, полученная на очередном шаге, в отличие от предыдущих методов, не имеет тенденцию к экспоненциальному росту.