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:

На этапе коррекции используется формула:

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:**

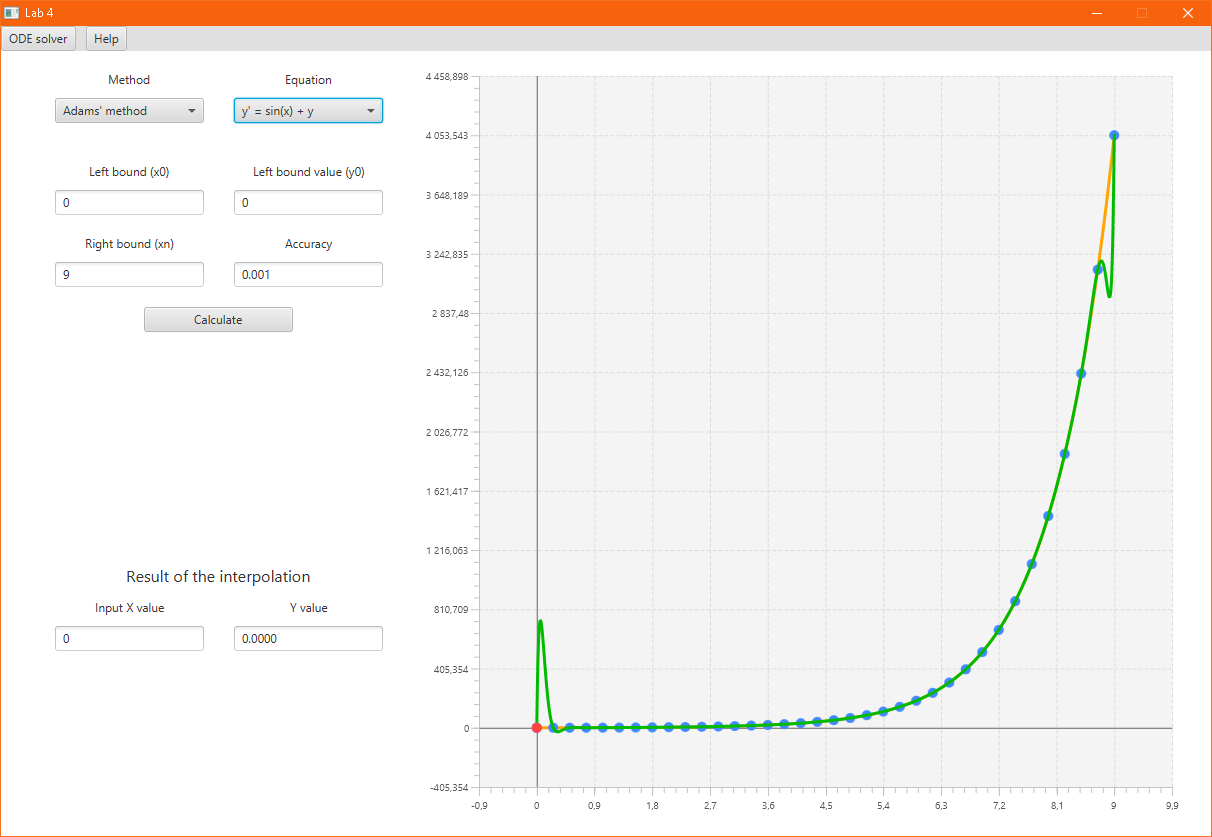
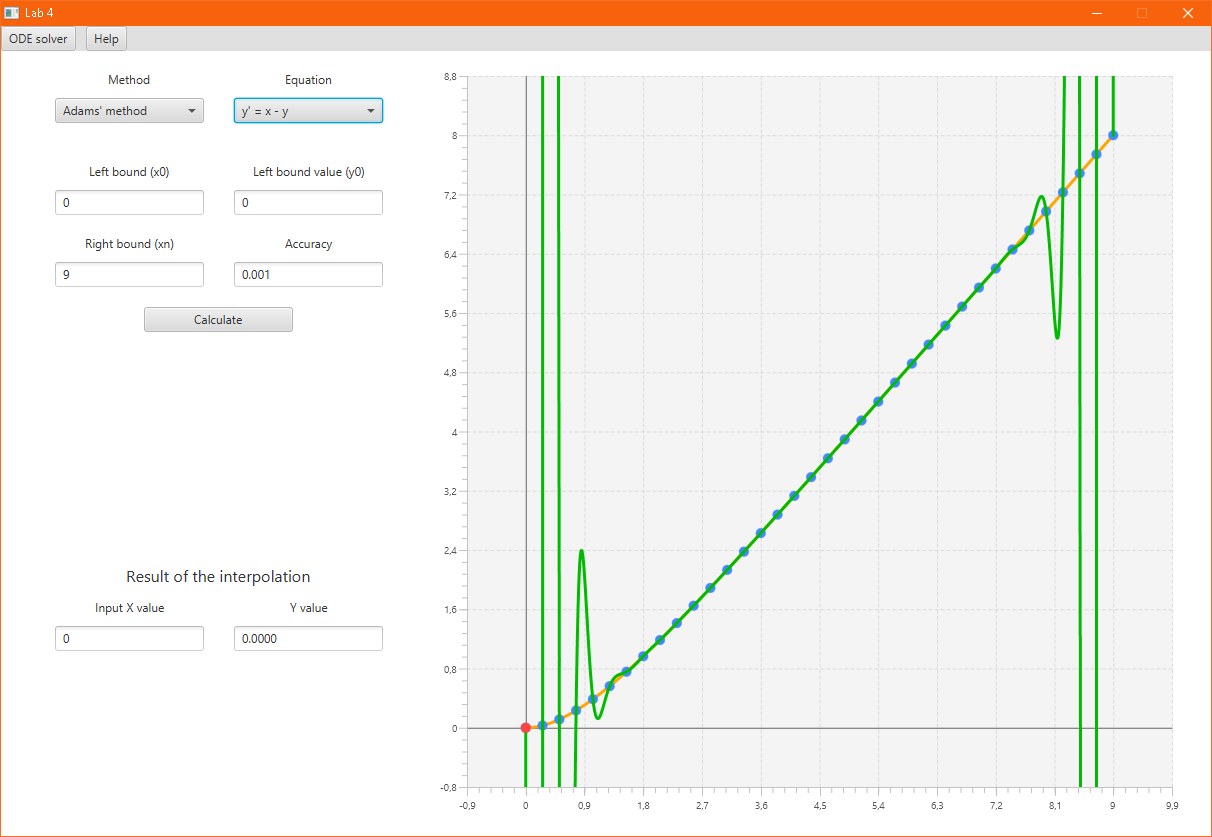
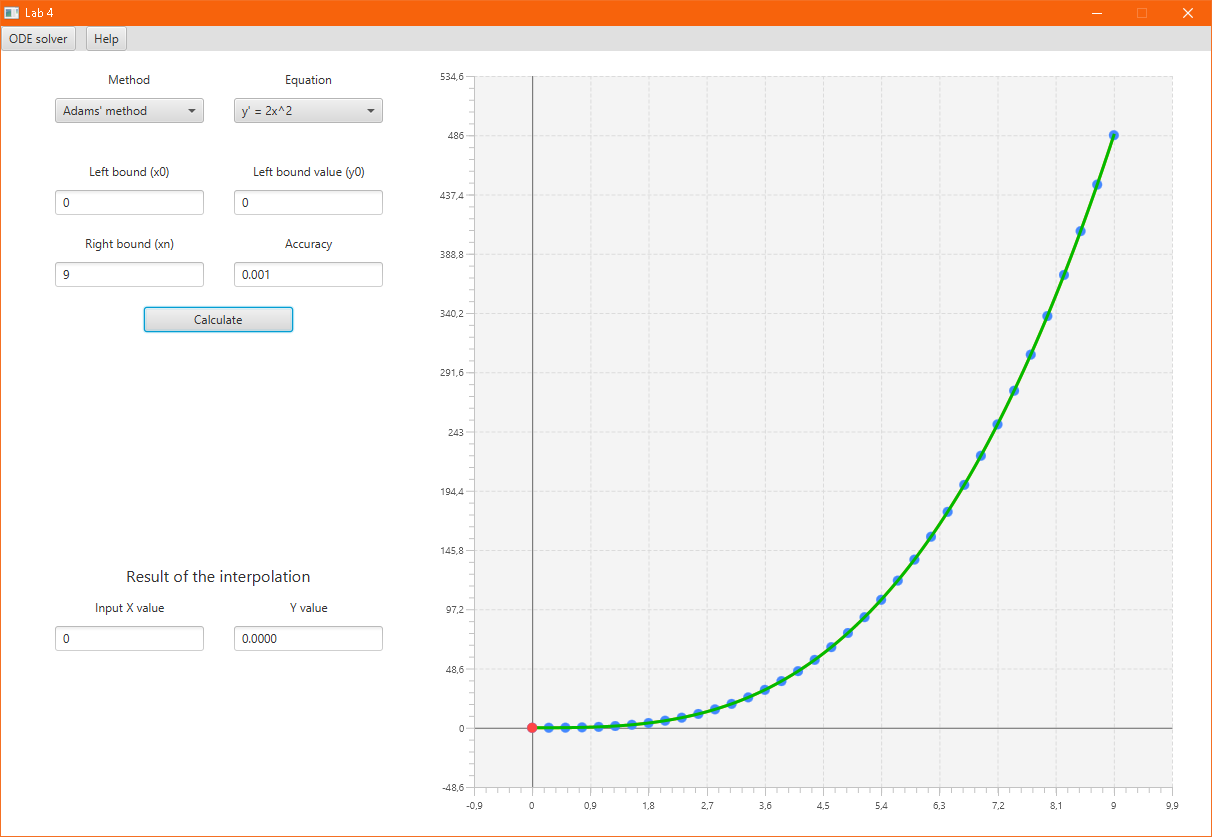
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31    32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155 | **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;  } |

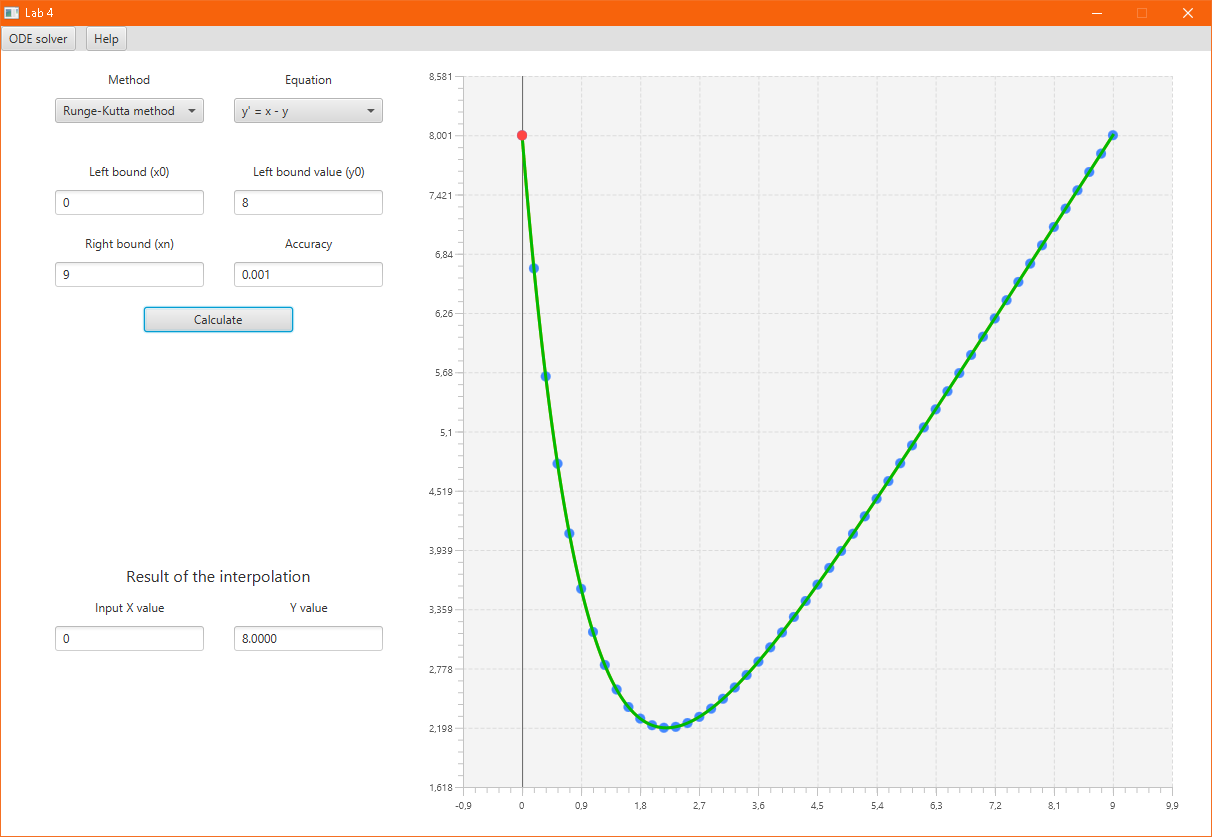
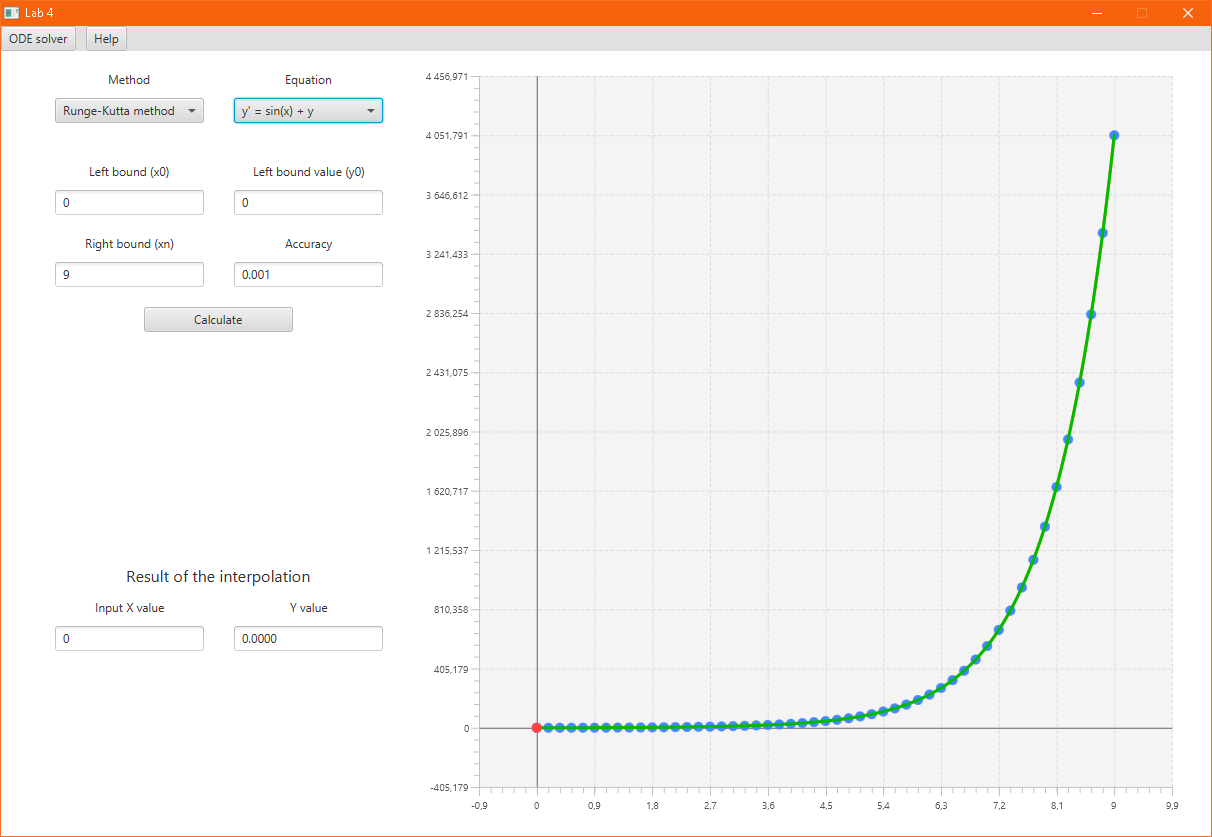
**Изображение выглядит как текст

Автоматически созданное описание**

**Результат работы:**

**Изображение выглядит как снимок экрана

Автоматически созданное описаниеИзображение выглядит как мужчина, стол, белый

Автоматически созданное описание**

**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.

1. Усовершенствованный метод Эйлера:

Второй порядок точности, погрешность накапливается с каждой точкой.

1. 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.

1. 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.