МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра вычислительные системы и технологии

Лабораторная работа № 6

Приближенное решение обыкновенных дифференциальных

уравнений

Вариант №15

ОТЧЕТ

по лабораторной работе

по дисциплине

Вычислительная математика

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Суркова А.С.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сапожников В.О.

19-ИВТ-3

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2021

**Содержание**

1. Цель……………………………………………………………………………..3
2. Постановка задачи……………………………………………………………..4
3. Теоретические сведения……………………………………………………….5
   1. Задача Коши……………………………...……………………………...5
   2. Метод Эйлера и метод Эйлера с пересчетом………………………….5
   3. Метод Рунге-Кутты……………………………………………………..8
   4. Метод Адамса………………………………………………………….10
4. Расчётные данные…………………...………………………………………..12
5. Листинг разработанной программы…………………………………………14
6. Результаты работы программы………………………………………………24
7. Вывод………………………………………………………………………….27
8. **Цель**

Закрепление знаний и умений по численному решению обыкновенных дифференциальных уравнений методом Эйлера, методом Эйлера с пересчетом, методом Рунге-Кутты и методом Адамса.

1. **Постановка задачи**

*Задание 1.*

Используя метод Эйлера и метод Эйлера с пересчетом, составить таблицу приближенных значений интеграла дифференциального уравнения , удовлетворяющих начальным условиям на отрезке ; шаг . Все вычисление вести с четырехзначными знаками. Проверить полученные значения, использую метод Рунге-Кутты 4 порядка.

*Задание 2.*

Используя метод Адамса с третьими разностями составить таблицу приближенных значений интеграла дифференциального уравнения , удовлетворяющих начальным условиям на отрезке ; шаг . Все вычисление вести с четырехзначными знаками. Начальный отрезок определить методом Рунге-Кутты. Проверить полученные значения, используя метод Эйлера с пересчетом.

1. **Теоретические сведения**
   1. **Задачи Коши**

Требуется найти функцию , удовлетворяющую уравнению и принимающую при заданное значение ; . Считаем, что решение нужно найти для значения . Производной функции называется передел отношения приращения функции к приращению аргумента при стремлении последнего к нулю:

в нашем случае.

* 1. **Метод Эйлера и метод Эйлера с пересчетом**

Рассмотрим уравнение в окрестности узлов и заменим производную правой разностью . Полученная аппроксимация дифференциального уравнения имеет 1-ый порядок, поскольку при замене допускается погрешность Выразим По этой формуле могут быть найдены значения сеточной функции в узлах. Построенный алгоритм называется методом Эйлера.

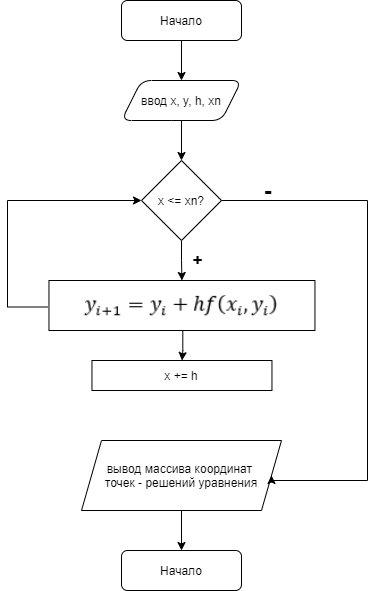
Погрешность.

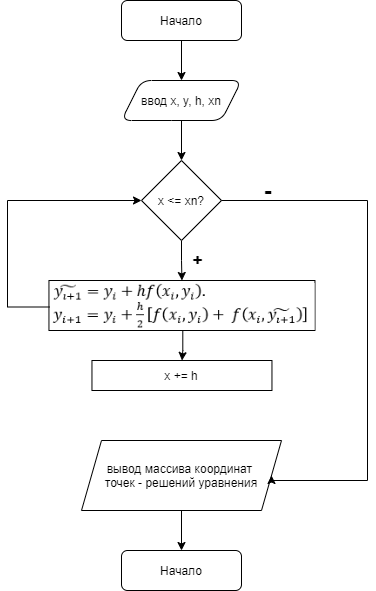
Погрешность на каждом шаге увеличивается на .  
Для повышения точности вычислений используют методом Эйлера с пересчетом. Однако для его реализации необходимо дважды вычислять правую часть функции. Заметим, что метод Эйлера с пересчетом представляет собой разновидность методом Рунге-Кутты.

Первым действием найдем

Вторым действием найдем

Метод Эйлера



Метод Эйлера с пересчетом  


* 1. **Метод Рунге-Кутты**

Метод Рунге — Кутты четвёртого порядка при вычислениях с постоянным шагом интегрирования столь широко распространён, что его часто называют просто методом Рунге — Кутты.

Рассмотрим задачу Коши для системы обыкновенных дифференциальных уравнений первого порядка.

{\displaystyle {\textbf {y}}'={\textbf {f}}(x,{\textbf {y}}),\quad {\textbf {y}}(x\_{0})={\textbf {y}}\_{0}.}

Тогда приближенное значение в последующих точках вычисляется по итерационной формуле:

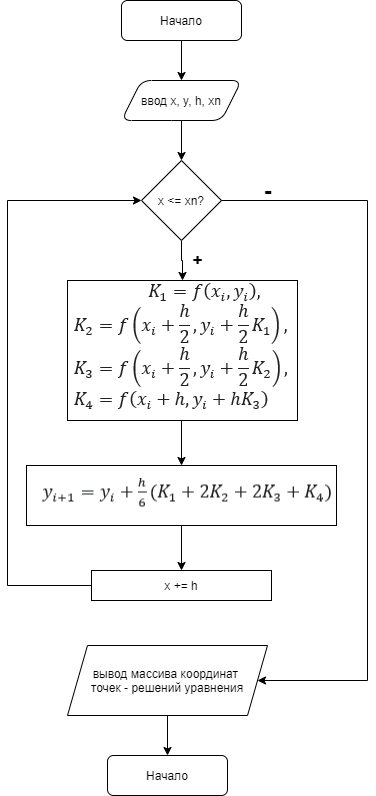
{\displaystyle {\textbf {y}}\_{n+1}={\textbf {y}}\_{n}+{h \over 6}({\textbf {k}}\_{1}+2{\textbf {k}}\_{2}+2{\textbf {k}}\_{3}+{\textbf {k}}\_{4})}

Вычисление нового значения проходит в четыре стадии:

{\displaystyle {\textbf {k}}\_{1}={\textbf {f}}\left(x\_{n},{\textbf {y}}\_{n}\right),}

где {\displaystyle h} — величина шага сетки по {\displaystyle x}xx.

Этот метод имеет четвёртый порядок точности. Это значит, что ошибка на одном шаге имеет порядок {\displaystyle O(h^{5})}, а суммарная ошибка на конечном интервале интегрирования имеет порядок {\displaystyle O(h^{4})}.

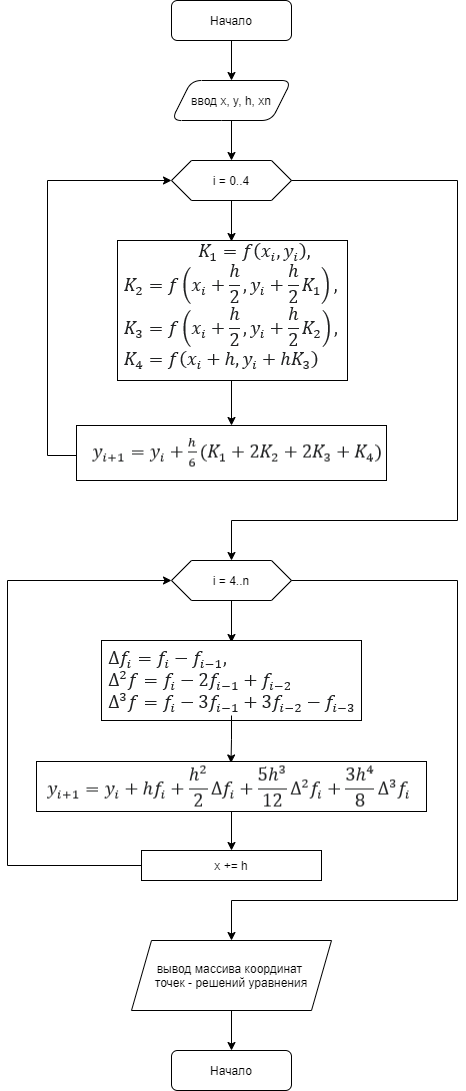


* 1. **Метод Адамса**

Запишем исходное уравнение в виде .

Проинтегрируем обе части этого уравнения по на отрезке . Интеграл от левой части легко вычисляется:   
Для вычисления интеграла от правой части уравнения строится сначала интерполяционный многочлен степени для аппроксимации функции на отрезке по значениям После этого можно написать:  
Приравнивая полученные выражение, можно получить формулу для определения неизвестного значения сеточной функции в узле :  
На основе этой формулы можно строить различные многошаговые методы любого порядка точности, который зависит от степени интерполяционного многочлена . Методы Адамса широко распространены и при k=1 – получается метод Эйлера первого порядка. По условию нам нужен метод Адамса четвертого порядка, с третьими разностями. Для него необходимо 4 значения y и функции предыдущих шагов. В качестве интерполяционного многочлена возьмем многочлен Ньютона. Конченые разности с учетом

Тогда разностную схему четверного порядка метода Адамса можно записать после необходимых преобразований в следующем виде:



1. **Расчетные данные  
   Задание 1**Метод Эйлера

|  |  |
| --- | --- |
| X | Y |
| 1.7000 | 5.3000 |
| 1.8000 | 5.5693 |
| 1.9000 | 5.8473 |
| 2.0000 | 6.1331 |
| 2.1000 | 6.4259 |
| 2.2000 | 6.7249 |
| 2.3000 | 7.0291 |
| 2.4000 | 7.3377 |
| 2.5000 | 7.6498 |
| 2.6000 | 7.9647 |
| 2.7000 | 8.2817 |

Метод Эйлера c пересчетом

|  |  |
| --- | --- |
| X | Y |
| 1.7000 | 5.3000 |
| 1.8000 | 5.5686 |
| 1.9000 | 5.8455 |
| 2.0000 | 6.1299 |
| 2.1000 | 6.4288 |
| 2.2000 | 6.7174 |
| 2.3000 | 7.0190 |
| 2.4000 | 7.3246 |
| 2.5000 | 7.6334 |
| 2.6000 | 7.9448 |
| 2.7000 | 8.2580 |

Метод Рунге-Кутты

|  |  |
| --- | --- |
| X | Y |
| 1.7000 | 5.3000 |
| 1.8000 | 5.5730 |
| 1.9000 | 5.8542 |
| 2.0000 | 6.1429 |
| 2.1000 | 6.4382 |
| 2.2000 | 6.7392 |
| 2.3000 | 7.0450 |
| 2.4000 | 7.3548 |
| 2.5000 | 7.6678 |
| 2.6000 | 7.9831 |
| 2.7000 | 8.3003 |

**Задание 2**Метод Адамса

|  |  |
| --- | --- |
| X | Y |
| 0.0000 | 0.0000 |
| 0.1000 | 0.0992 |
| 0.2000 | 0.1933 |
| 0.3000 | 0.2779 |
| 0.4000 | 0.3544 |
| 0.5000 | 0.4171 |
| 0.6000 | 0.4665 |
| 0.7000 | 0.5043 |
| 0.8000 | 0.5325 |
| 0.9000 | 0.5537 |
| 1.0000 | 0.5699 |

Метод Эйлера с пересчетом

|  |  |
| --- | --- |
| X | Y |
| 0.0000 | 0.0000 |
| 0.1000 | 0.0923 |
| 0.2000 | 0.1817 |
| 0.3000 | 0.2639 |
| 0.4000 | 0.3361 |
| 0.5000 | 0.3969 |
| 0.6000 | 0.4467 |
| 0.7000 | 0.4863 |
| 0.8000 | 0.5174 |
| 0.9000 | 0.5417 |
| 1.0000 | 0.5612 |

1. **Листинг разработанной программы**

**Main.java**

|  |
| --- |
| import function.Function; |
| import function.FirstFunction; |
| import function.SecondFunction; |
| import solution\_strategy.\*; |
|  |
| import java.util.Scanner; |
|  |
| /\*\* |
| \* Класс, содержащий точку входа в программу - метод main. |
| \* Язык: java |
| \* |
| \* Реализация шестой лабораторной работы по дисциплине: Вычислительная математика |
| \* Вариант №15 |
| \* |
| \* Текст задания: |
| \* Задание 1 |
| \* Используя метод Эйлера и метод Эйлера с пересчетом, составить |
| \* таблицу приближенных значений интеграла дифференциального уравнения |
| \* y΄=f(x,y), удовлетворяющего начальным условиям y( на отрезке [a,b]; |
| \* шаг h=0.1. Все вычисления вести с четырехзначными знаками. Проверить |
| \* полученные значения, используя метод Рунге-Кутты 4 порядка |
| \* |
| \* Функция: |
| \* y'=x+sin(y/pi) X ∈ [1.7; 2.7] y0(1.7) = 5.3 |
| \* |
| \* @release: - 03.05.21 |
| \* @last\_update: - 03.05.21 |
| \* |
| \* @author Vladislav Sapozhnikov 19-IVT-3 |
| \*/ |
| public class Main |
| { |
| //Константы для хранения последовательностей для |
| //изменения цвета текста в консоли |
| public static final String RESET = "\u001B[0m"; |
| public static final String PURPLE = "\u001B[35m"; |
| public static final String RED = "\u001B[31m"; |
|  |
| /\*\* |
| \* Точка входа в программу |
| \* \*/ |
| public static void main(String[] args) |
| { |
| System.out.println("\t\t\t\tЛабораторная работа №5 <<" + PURPLE + "Численное  дифференцирование функций Ньютона и многочленом Лагранжа" + RESET + ">>"); |
|  |
|  |
| //Открытие потока ввода |
| Scanner scanner = new Scanner(System.in); |
|  |
| //Создание ссылки на объект, реализующий интерфейс |
| //SolutionStrategy |
| SolutionStrategy strategy = null; |
|  |
| //Переменная для хранения результата ввода |
| String ch = ""; |
|  |
| double x = 0.0; |
| double xn = 0.0; |
| double y = 0.0; |
|  |
| Function function = null; |
|  |
| //Выбор стратегии решения |
| while (!ch.equals("q")) |
| { |
| System.out.println("Выберите способ:"); |
| System.out.println("\t1. Задание1. Метод Эйлера"); |
| System.out.println("\t2. Задание1. Метод Эйлера с пересчетом"); |
| System.out.println("\t3. Задание1. Метод Рунге-Кутта 4 порядка"); |
| System.out.println(); |
| System.out.println("\t4. Задание2. Метод Адамса"); |
| System.out.println("\t5. Задание2. Метод Эйлера с пересчетом"); |
| System.out.println(); |
| System.out.println("\tВведите q для выхода"); |
| System.out.print("Ввод: "); |
| ch = scanner.nextLine(); |
| System.out.println(); |
|  |
| //Ввод с повторением |
| switch (ch) |
| { |
| case ("1"): |
| { |
| strategy = new EulerSolution(); |
| function = new FirstFunction(); |
| x = 1.7; |
| xn = 2.7; |
| y = 5.3; |
| break; |
| } |
| case ("2"): |
| { |
| strategy = new EulerRecalculationSolution(); |
| function = new FirstFunction(); |
| x = 1.7; |
| xn = 2.7; |
| y = 5.3; |
| break; |
| } |
| case ("3"): |
| { |
| strategy = new RungeKuttSolution(); |
| function = new FirstFunction(); |
| x = 1.7; |
| xn = 2.7; |
| y = 5.3; |
| break; |
| } |
| case ("4"): |
| { |
| strategy = new AdamsSolution(); |
| function = new SecondFunction(); |
| x = 0.0; |
| xn = 1.0; |
| y = 0; |
| break; |
| } |
| case ("5"): |
| { |
| strategy = new EulerRecalculationSolution(); |
| function = new SecondFunction(); |
| x = 0.0; |
| xn = 1.0; |
| y = 0; |
| break; |
| } |
| case ("q"): |
| { |
| System.out.println(RED + "Завершение работы..." + RESET); |
| System.exit(0); |
| } |
| default: |
| { |
| System.out.println(RED + "Неверный ввод!" + RESET); |
| break; |
| } |
| } |
|  |
| assert strategy != null; |
| strategy.getSolution(function, x, xn, y, 0.1); |
| System.out.println(); |
| System.out.println(); |
| } |
| } |
| } |

**function/Function.java**

|  |
| --- |
| package function; |
|  |
| /\*\* |
| \* Интерфейс, содержащий основные методы функций. |
| \* \*/ |
| public interface Function |
| { |
| /\*\* |
| \* Получение значения функции при заданных координатах. |
| \* |
| \* @param x - координата по х |
| \* @param y - координата по у |
| \* @return значение функции |
| \* \*/ |
| double getValue(double x, double y); |
| } |

**function/FirstFunction.java**

|  |
| --- |
| package function; |
|  |
| /\*\* |
| \* Класс - первая функция Варианта №15 |
| \* |
| \* @see Function |
| \* \*/ |
| public class FirstFunction implements Function |
| { |
| /\*\* |
| \* Получение значения функции при заданных координатах. |
| \* |
| \* @param x - координата по х |
| \* @param y - координата по у |
| \* @return значение функции |
| \* \*/ |
| @Override |
| public double getValue(double x, double y) |
| { |
| return x + Math.sin(y / Math.PI); |
| } |
| } |

**function/SecondFunction.java**

|  |
| --- |
| package function; |
| /\*\* |
| \* Класс - вторая функция Варианта №15 |
| \* |
| \* @see Function |
| \* \*/ |
| public class SecondFunction implements Function |
| { |
| /\*\* |
| \* Получение значения функции при заданных координатах. |
| \* |
| \* @param x - координата по х |
| \* @param y - координата по у |
| \* @return значение функции |
| \* \*/ |
| @Override |
| public double getValue(double x, double y) |
| { |
| return Math.cos(1.5 \* x + y) + 1.5 \* (x - y); |
| } |
| } |

**solution\_strategy/SolutionStrategy.java**

|  |
| --- |
| package solution\_strategy; |
|  |
| import function.Function; |
|  |
| /\*\* |
| \* Общий интерфейс всех стратегий решения. |
| \* \*/ |
| public interface SolutionStrategy |
| { |
| /\*\* |
| \* Метод для получения массива точек являющихся решением |
| \* дифференциальных уравнений. |
| \* \*/ |
| void getSolution(Function function, double x, double xn, double y, double h); |
| } |

**solution\_strategy/EulerSolution.java**

Начало формы

Конец формы

|  |
| --- |
| package solution\_strategy; |
| import function.Function; |
|  |
| /\*\* |
| \* Класс, реализующий решение простого дифференциального |
| \* уравнения методом Эйлера. |
| \* |
| \* @see SolutionStrategy |
| \* \*/ |
| public class EulerSolution implements SolutionStrategy |
| { |
| /\*\* |
| \* Метод для получения массива точек, являющихся решением |
| \* дифференциальных уравнений при помощи метода Эйлера. |
| \* |
| \* @param function - уравнение значение которого необходимо  \* получить |
| \* @param x - начальное значение х |
| \* @param xn - конченое значение х |
| \* @param y - начальное значение y |
| \* @param h - величина шага |
| \* \*/ |
| @Override |
| public void getSolution(Function function, double x, double xn,  double y, double h) |
| { |
| //Добавляем в массив начальные условия |
| System.out.printf("%.4f", x); |
| System.out.print(" "); |
| System.out.printf("%.4f", y); |
| System.out.println(); |
|  |
| //по формуле высчитываем все остальные точки |
| while (x <= xn) |
| { |
| y = y + h \* function.getValue(x, y); |
| x += h; |
|  |
| System.out.printf("%.4f", x); |
| System.out.print(" "); |
| System.out.printf("%.4f", y); |
| System.out.println(); |
| } |
| } |
| } |

**solution\_strategy/EulerRecalculationSolution.java**

|  |
| --- |
| package solution\_strategy; |
|  |
| import function.Function; |
|  |
| /\*\* |
| \* Класс реализующий решение простого дифференциального |
| \* уравнения методом Эйлера с пересчетом. |
| \* |
| \* @see SolutionStrategy |
| \* \*/ |
| public class EulerRecalculationSolution implements SolutionStrategy |
| { |
| /\*\* |
| \* Метод для получения массива точек, являющихся решением |
| \* дифференциальных уравнений при помощи метода Эйлера с пересчетом. |
| \* |
| \* @param function - уравнение значение которого необходимо получить |
| \* @param x - начальное значение х |
| \* @param xn - конченое значение х |
| \* @param y - начальное значение y |
| \* @param h - величина шага |
| \* \*/ |
| @Override |
| public void getSolution(Function function, double x, double xn, double y, double h) |
| { |
| double recalculationY; |
|  |
| //Добавляем в массив начальные условия |
| System.out.printf("%.4f", x); |
| System.out.print(" "); |
| System.out.printf("%.4f", y); |
| System.out.println(); |
|  |
| //по формуле высчитываем все остальные точки |
| int i = 1; |
| while (x <= xn) |
| { |
| recalculationY = y + h \* function.getValue(x, y); |
| y = y + 0.5 \* h \* (function.getValue(x, y) + function.getValue(x, recalculationY)); |
| x += h; |
|  |
| System.out.printf("%.4f", x); |
| System.out.print(" "); |
| System.out.printf("%.4f", y); |
| System.out.println(); |
|  |
| i++; |
| } |
| } |
| } |

**solution\_strategy/RungekuttSolution.java**

|  |
| --- |
| package solution\_strategy; |
|  |
| import function.Function; |
|  |
| /\*\* |
| \* Класс реализующий решение простого дифференциального |
| \* уравнения методом Рунге-Кутта. |
| \* |
| \* @see SolutionStrategy |
| \* \*/ |
| public class RungeKuttSolution implements SolutionStrategy |
| { |
| /\*\* |
| \* Метод для получения массива точек, являющихся решением |
| \* дифференциальных уравнений при помощи метода Рунге-Кутта. |
| \* |
| \* @param function - уравнение значение которого необходимо получить |
| \* @param x - начальное значение х |
| \* @param xn - конченое значение х |
| \* @param y - начальное значение y |
| \* @param h - величина шага |
| \* \*/ |
| @Override |
| public void getSolution(Function function, double x, double xn, double y, double h) |
| { |
| //Добавляем в массив начальные условия |
| System.out.printf("%.4f", x); |
| System.out.print(" "); |
| System.out.printf("%.4f", y); |
| System.out.println(); |
|  |
| double K1; |
| double K2; |
| double K3; |
| double K4; |
|  |
| //по формуле высчитываем все остальные точки |
| while (x <= xn) |
| { |
| K1 = function.getValue(x , y); |
| K2 = function.getValue(x + h / 4.0, y + (h / 4.0) \* K1); |
| K3 = function.getValue(x + h / 2.0, y + (h / 2.0) \* K2); |
| K4 = function.getValue(x + h, y + h \* K1 - 2.0 \* h \* K2 + 2.0 \* h \* K3); |
|  |
| y = y + (h \* (K1 + 2.0 \* K2 + 2.0 \* K3 + K4)) / 6.0; |
| x += h; |
|  |
| System.out.printf("%.4f", x); |
| System.out.print(" "); |
| System.out.printf("%.4f", y); |
| System.out.println(); |
| } |
| } |
| } |

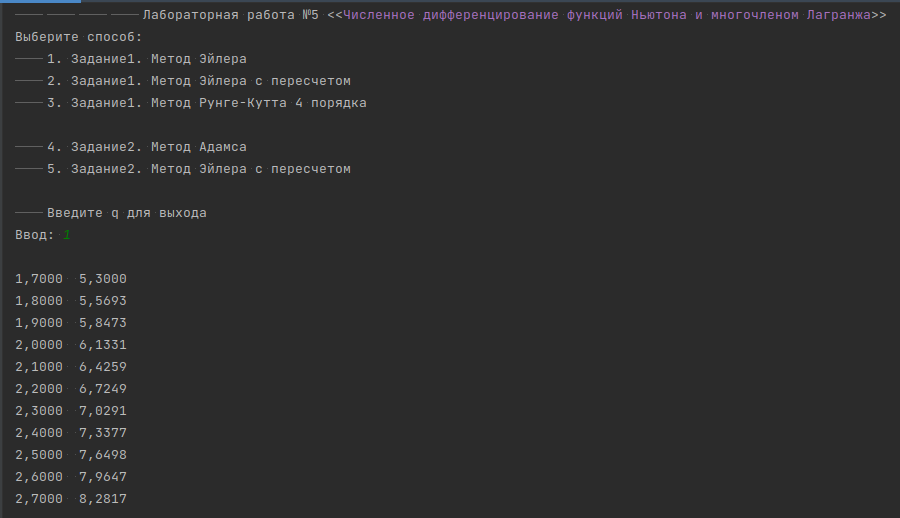
**solution\_strategy/AdamsSolution.java**

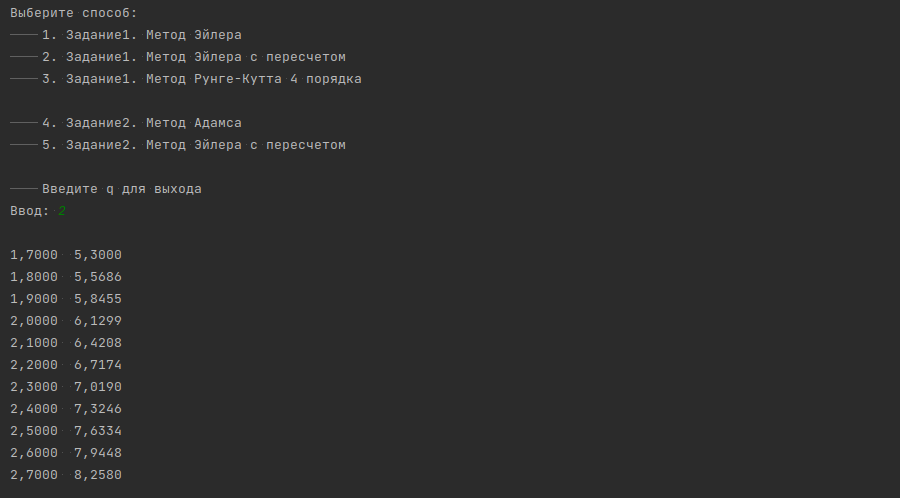
|  |
| --- |
| package solution\_strategy; |
| import function.Function; |
| /\*\* |
| \* Класс реализующий решение простого дифференциального |
| \* уравнения методом Адамса. |
| \* |
| \* @see SolutionStrategy |
| \* \*/ |
| public class AdamsSolution implements SolutionStrategy |
| { |
| /\*\* |
| \* Метод для получения массива точек являющихся решением |
| \* дифференциальных уравнений при помощи метода Адамса. |
| \* |
| \* @param function - уравнение значение которго необходимо получить |
| \* @param x - начальное значение х |
| \* @param xn - конченое значение х |
| \* @param y - начальное значение y |
| \* @param h - величина шага |
| \* \*/ |
| @Override |
| public void getSolution(Function function, double x, double xn, double y, double h) |
| { |
| //Массив для хранения результатов |
| double[][] res = new double[11][2]; |
|  |
| //Добавляем в массив начальные условия |
| res[0][0] = x; |
| res[0][1] = y; |
|  |
| //Переменные для расчета методом Рунге-Кутта |
| double K1; |
| double K2; |
| double K3; |
| double K4; |
|  |
| //Вычисляем начальный отрезок методом Рунге-Кутта |
| for (int i = 1; i < 4; i++) |
| { |
| K1 = function.getValue(x , y); |
| K2 = function.getValue(x + h / 4.0, y + (h / 4.0) \* K1); |
| K3 = function.getValue(x + h / 2.0, y + (h / 2.0) \* K2); |
| K4 = function.getValue(x + h, y + h \* K1 - 2.0 \* h \* K2 + 2.0 \* h \* K3); |
|  |
| y = y + (h \* (K1 + 2.0 \* K2 + 2.0 \* K3 + K4)) / 6.0; |
| x += h; |
|  |
| res[i][0] = x; |
| res[i][1] = y; |
|  |
| System.out.printf("%.4f", res[i][0]); |
| System.out.print(" "); |
| System.out.printf("%.4f", res[i][1]); |
| System.out.println(); |
| } |
|  |
| //Вычисляем все остальные значения при помощи метода Адамса |
| double[] df = new double[3]; |
| for (int i = 4; i < 11; i++) |
| { |
| df[0] = res[i][1] - res[i - 1][1]; |
| df[1] = res[i][1] - 2.0 \* res[i - 1][1] + res[i - 2][1]; |
| df[1] = res[i][1] - 3.0 \* res[i - 1][1] + 3.0 \* res[i - 2][1] - res[i - 3][1]; |
|  |
| y = y + h \* function.getValue(x, y) + (df[0] \* h \* h / 2.0) + 5.0 \*  (df[1] \* h \* h \* h / 12.0) + 3.0 \* (df[2] \* h \* h \* h \* h / 8.0); |
|  |
| x += h; |
|  |
| res[i][0] = x; |
| res[i][1] = y; |
|  |
| System.out.printf("%.4f", res[i][0]); |
| System.out.print(" "); |
| System.out.printf("%.4f", res[i][1]); |
| System.out.println(); |
| } |
| } |
| } |

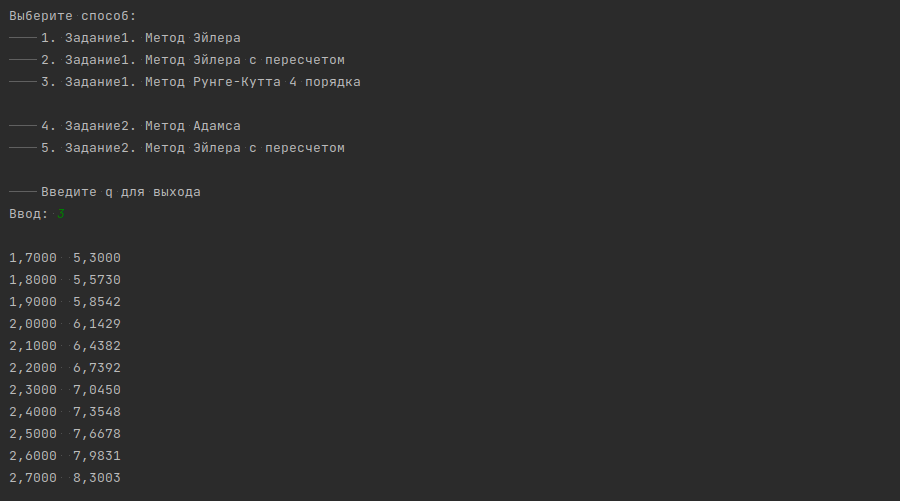
Начало формы

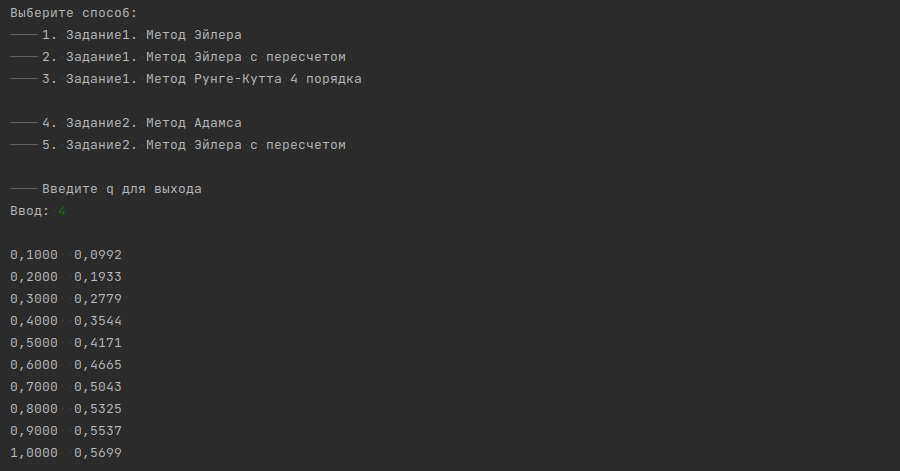
Конец формы

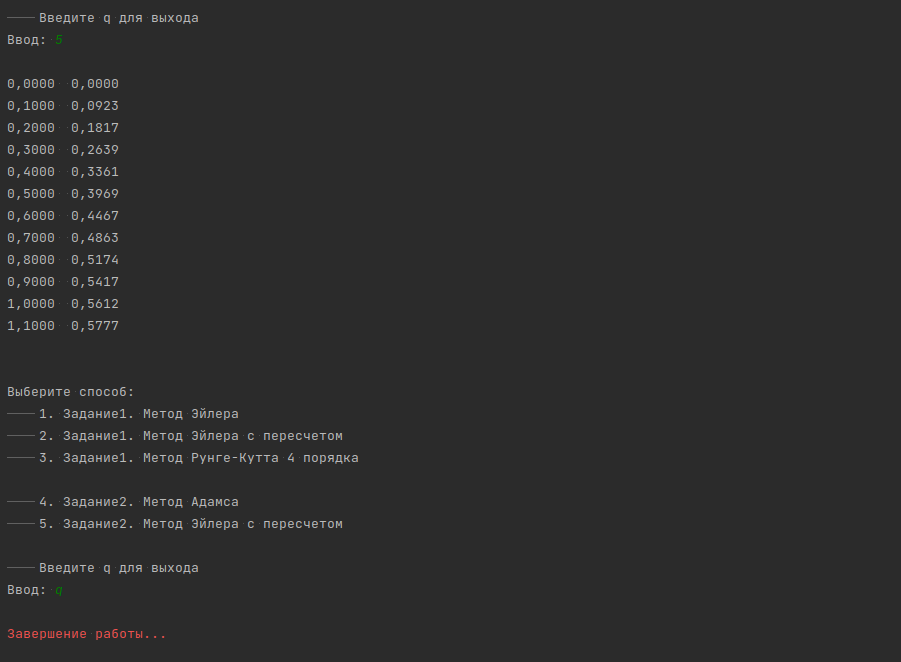
1. **Результаты работы программы**











1. **Вывод**

В ходе данной работы были закреплены знания и умения по вычислению приближенных значений интеграла дифференциального уравнения при помощи методом Эйлера, Эйлера с пересчетом, Рунге-Кутты и Адамса.