**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

Лабораторная №3

**Обыкновенные дифференциальные уравнения**

Вариант 2

**Выполнил:**

Кендысь Алексей Максимович

студент 3 курса, 7 группы,

специальность

“прикладная математика”

**Преподаватель:**

Доцент кафедры вычислительной

математики ФПМИ,

А.М. Будник

Минск, 2022

**Содержание:**

Постановка задачи 2

Предварительные вычисления 2

Неявный метод Эйлера 3

Краткие теоретические сведения 3

Листинг программы 4

Результаты 5

Выводы 5

Метод Рунге-Кутта 6

Краткие теоретические сведения 6

Листинг программы 7

Результаты 9

Выводы 9

Метод последовательного повышения порядка точности 9

Краткие теоретические сведения 9

Листинг программы 11

Результаты 12

Выводы 12

Экстраполяционный метод Адамса 12

Краткие теоретические сведения 13

Листинг программы 14

Результаты 15

Выводы 15

Постановка задачи

Найти приближённое решение задачи Коши

на сетке узлов при 10-ти разбиениях отрезка интегрирования, применяя следующие методы:

1. Неявный метод Эйлера. Для его реализации использовать алгоритм метода Ньютона.
2. Метод Рунге-Кутта, построенный по таблице Бутчера вида:
3. Метод последовательного повышения порядка точности 2-го порядка при .
4. Экстраполяционный метод Адамса 3-го порядка с началом таблицы, построенным по соответствующему методу последовательного повышения порядка точности.

Для проведения анализа полученных результатов необходимо:

1. Используя таблицу приближённых результатов, получить погрешности методов 1-3, используя точное решение уравнения.
2. Исходя из вида главного члена локальной погрешности методов 1-4, объяснить разницу результатов.
3. На основе полученных численных и теоретических результатов сделать вывод о точности каждого метода 1-4.

Предварительные вычисления

Найдём точное решение задачи Коши.

Уравнение является дифференциальным уравнением Бернулли, которое заменой приводится к линейному неоднородному уравнению первого порядка:

Далее находим решение соответствующего однородного уравнения, а потом методом вариации произвольных постоянных само решение линейного уравнения. В итоге для исходного уравнения, делая обратную замену, получаем общее решение:

Подставляя условие , получаем точное решение исходной задачи Коши:

Неявный метод Эйлера

Краткие теоретические сведения

Рассмотрим разбиение отрезка для (количество разбиений). Величина шага Узлы ,

*.*

Неявный метод Эйлера записывается в виде:

, ,

где , , – правая часть уравнения.

Метод является неявным и в таком виде пока не может быть применим, т.к. чтобы найти требуется решить нелинейное уравнение. Для этого используем метод Ньютона, который для уравнения записывается в виде:

В итоге получаем следующий метод:

Здесь .

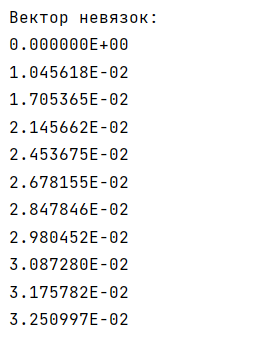
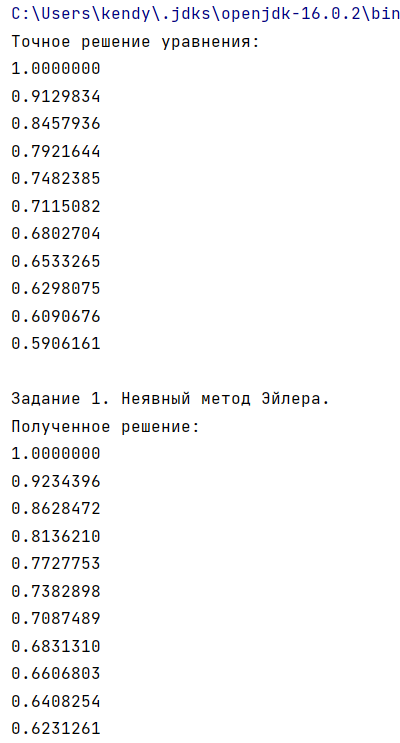
В качестве начального приближения берем значение на предыдущем узле, . Для остановки итерационного процесса используем Значение берется в соответствие с видом локальной погрешности метода, выбирается значение в степени на 1 больше, чем в главном члене погрешности.

Неявный метод Эйлера является методом 1-го порядка, и его локальная погрешность имеет вид:

Листинг программы

import java.util.\*;  
  
class F {  
 public static double getValue(double t, double u) {  
 return (Math.*pow*(u, 2.) \* Math.*log*(t) - u) / t;  
 }  
  
 public static double getDuValue(double t, double u) {  
 return (2. \* u \* Math.*log*(t) - 1) / t;  
 }  
}  
  
class U {  
 public static double getValue(double t) {  
 return 1. / (Math.*log*(t) + 1);  
 }  
}  
  
class BackwardEulerMethod {  
 private final int n;  
 private final double tau;  
 private final double[] t;  
 private final double u0;  
  
 public BackwardEulerMethod(int n, double tau, double[] t, double u0) {  
 this.n = n;  
 this.tau = tau;  
 this.t = t;  
 this.u0 = u0;  
 }  
  
 public double[] getY() {  
 double[] res = new double[this.n + 1];  
 res[0] = this.u0;  
  
 for (int i = 1; i <= this.n; i++) {  
 res[i] = newtonMethod(res[i - 1], t[i]);  
 }  
  
 return res;  
 }  
  
 public double newtonMethod(double yJ, double t) {  
 final double lteDeg = 2.;  
 double eps = Math.*pow*(this.tau, lteDeg + 1.);  
  
 double nwt1 = yJ;  
 double nwt2 = phiNewton(nwt1, yJ, t);  
 while (Math.*abs*(nwt2 - nwt1) > eps) {  
 nwt1 = nwt2;  
 nwt2 = phiNewton(nwt2, yJ, t);  
 }  
  
 return nwt2;  
 }  
  
 private double phiNewton(double y, double yJ, double t) {  
 return y - (y - yJ - this.tau \* F.*getValue*(t, y)) / (1 - this.tau \* F.*getDuValue*(t, y));  
 }  
}  
  
class CauchyProblem {  
 private final static int *N* = 10;  
 private final static double *A* = 1.;  
 private final static double *B* = 2.;  
 private final static double *U0* = 1.;  
 private final double tau;  
 private final double[] t;  
 private final double[] u;  
 private double[] y1;   
 private double[] res1;   
  
  
 public CauchyProblem() {  
 this.tau = (*B* - *A*) / *N*;  
  
 this.t = new double[*N* + 1];  
 this.t[0] = *A*;  
 for (int i = 1; i <= *N*; i++) {  
 this.t[i] = this.t[i - 1] + this.tau;  
 }  
  
 this.u = new double[*N* + 1];  
 for (int i = 0; i <= *N*; i++) {  
 this.u[i] = U.*getValue*(this.t[i]);  
 }  
 }  
  
 public void solve() {  
 BackwardEulerMethod bem = new BackwardEulerMethod(*N*, this.tau, this.t, *U0*);  
 this.y1 = bem.getY();  
 this.res1 = this.getResidual(this.y1);   
 }  
  
 public void outResult() {  
 Formatter fmt = new Formatter();  
  
 fmt.format("Точное решение уравнения:\n");  
 for (int i = 0; i <= *N*; i++) {  
 fmt.format("%.7f\n", this.u[i]);  
 }  
  
 fmt.format("\nЗадание 1. Неявный метод Эйлера.\n");  
 this.outYAndRes(fmt, this.y1, this.res1);   
  
 System.*out*.println(fmt);  
 }  
  
 private void outYAndRes(Formatter fmt, double[] y, double[] res) {  
 fmt.format("Полученное решение:\n");  
 for (int i = 0; i <= *N*; i++) {  
 fmt.format("%.7f\n", y[i]);  
 }  
 fmt.format("Вектор невязок:\n");  
 for (int i = 0; i <= *N*; i++) {  
 fmt.format("%E\n", res[i]);  
 }  
 }  
  
 private double[] getResidual(double[] y) {  
 double[] res = new double[*N* + 1];  
 for (int i = 0; i <= *N*; i++) {  
 res[i] = Math.*abs*(this.u[i] - y[i]);  
 }  
  
 return res;  
 }  
}  
  
public class Main {  
  
 public static void main(String[] args) {  
 CauchyProblem cp = new CauchyProblem();  
 cp.solve();  
 cp.outResult();  
 }  
}

Результаты



Выводы

Из вида вектора невязок (модуль разницы между точным и приближенным значением) видно, что реальная погрешность соответствует виду главного члена локальной погрешности (для первого значения она составила примерно ). Неявный метод Эйлера является методом 1-го порядка, вид локальной погрешности – в нашем случае . Локальная погрешность описывает погрешность на каждом шаге. Под знаком кроется некоторая константа, т.е. , поэтому для остальных значений локальная погрешность уже другая, она чуть меньше. Также заметим, что общая погрешность нарастает с каждым значением.

Т.к. в качестве остановки процесса метода Ньютона берется , где в качестве берется степень на 1 больше, чем в главном члене локальной погрешности, то метод Ньютона не влияет на точность приближения.

Метод Рунге-Кутта

Краткие теоретические сведения

Метод Рунге-Кутта, построенный по таблице Бутчера вида , записывается в следующем виде:

*.*

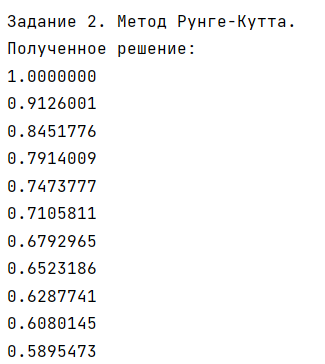
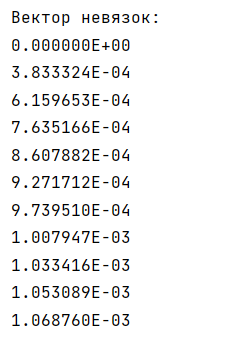
Данный метод базируется на квадратурной формуле трапеции. Он является методом 2-го порядка, и его локальная погрешность имеет вид:

Листинг программы

import java.util.\*;  
  
class F {  
 public static double getValue(double t, double u) {  
 return (Math.*pow*(u, 2.) \* Math.*log*(t) - u) / t;  
 }  
  
 public static double getDuValue(double t, double u) {  
 return (2. \* u \* Math.*log*(t) - 1) / t;  
 }  
}  
  
class U {  
 public static double getValue(double t) {  
 return 1. / (Math.*log*(t) + 1);  
 }  
}

class RungeKuttaMethod {  
 public static double[] getY(int n, double tau, double[] t, double u0) {  
 double[] res = new double[n + 1];  
 res[0] = u0;  
  
 double k1, k2;  
 for (int i = 1; i <= n; i++) {  
 k1 = F.*getValue*(t[i - 1], res[i - 1]);  
 k2 = F.*getValue*(t[i], res[i - 1] + tau \* k1);  
 res[i] = res[i - 1] + 0.5 \* tau \* (k1 + k2);  
 }  
  
 return res;  
 }  
}  
  
class CauchyProblem {  
 private final static int *N* = 10;  
 private final static double *A* = 1.;  
 private final static double *B* = 2.;  
 private final static double *U0* = 1.;  
 private final double tau;  
 private final double[] t;  
 private final double[] u;   
 private double[] y2;   
 private double[] res2;   
  
  
 public CauchyProblem() {  
 this.tau = (*B* - *A*) / *N*;  
  
 this.t = new double[*N* + 1];  
 this.t[0] = *A*;  
 for (int i = 1; i <= *N*; i++) {  
 this.t[i] = this.t[i - 1] + this.tau;  
 }  
  
 this.u = new double[*N* + 1];  
 for (int i = 0; i <= *N*; i++) {  
 this.u[i] = U.*getValue*(this.t[i]);  
 }  
 }  
  
 public void solve() {  
 this.y2 = RungeKuttaMethod.*getY*(*N*, this.tau, this.t, *U0*);  
 this.res2 = this.getResidual(this.y2);   
 }  
  
 public void outResult() {  
 Formatter fmt = new Formatter();  
  
 fmt.format("Точное решение уравнения:\n");  
 for (int i = 0; i <= *N*; i++) {  
 fmt.format("%.7f\n", this.u[i]);  
 }  
  
 fmt.format("\nЗадание 2. Метод Рунге-Кутта.\n");  
 this.outYAndRes(fmt, this.y2, this.res2);   
  
 System.*out*.println(fmt);  
 }  
  
 private void outYAndRes(Formatter fmt, double[] y, double[] res) {  
 fmt.format("Полученное решение:\n");  
 for (int i = 0; i <= *N*; i++) {  
 fmt.format("%.7f\n", y[i]);  
 }  
 fmt.format("Вектор невязок:\n");  
 for (int i = 0; i <= *N*; i++) {  
 fmt.format("%E\n", res[i]);  
 }  
 }  
  
 private double[] getResidual(double[] y) {  
 double[] res = new double[*N* + 1];  
 for (int i = 0; i <= *N*; i++) {  
 res[i] = Math.*abs*(this.u[i] - y[i]);  
 }  
  
 return res;  
 }  
}  
  
public class Main {  
  
 public static void main(String[] args) {  
 CauchyProblem cp = new CauchyProblem();  
 cp.solve();  
 cp.outResult();  
 }  
}

Результаты

Выводы

Как и для прошлого метода, реальная погрешность примерно соответствует виду главного члена локальной погрешности (для первого значения она составила примерно ). Используемый метод Рунге-Кутта является методом 2-го порядка, вид локальной погрешности – . Для данного метода невязка вышла даже чуть меньше ожидаемого, константа в главном члене погрешности понизила её еще на порядок.

Используемый метод Рунге-Кутта вышел точнее неявного метода Эйлера за счёт разницы в порядках методов.

Метод последовательного повышения порядка точности

Краткие теоретические сведения

Метод последовательного повышения порядка точности 2-го порядка при имеет вид:

*.*

Здесь в квадратных скобках указана погрешность вычисляемого значения, а именно степень в главном члене локальной погрешности метода.

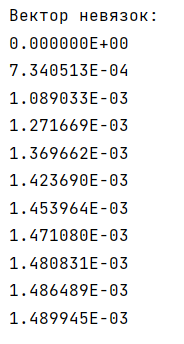
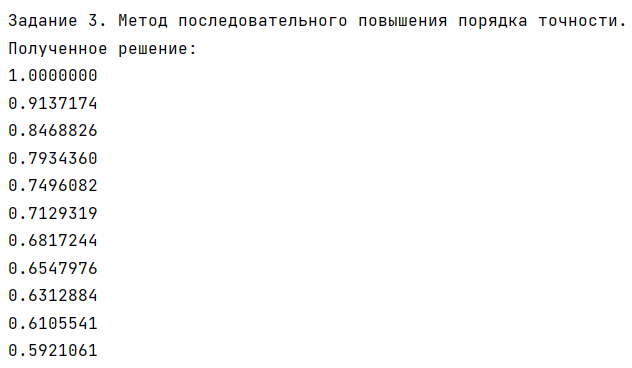
Сама формула последовательного повышения порядка – это первая строчка метода. Для 2-го порядка и она определяется единственным образом. Вторая же строчка – это способ получения значения . Для него достаточно использовать метод на один порядок ниже, т.е. метод 1-го порядка. В данном случае был выбран явный метод Эйлера (только тут шаг ).

Данный метод базируется на квадратурной формуле средних прямоугольников. Он является методом 2-го порядка, и его локальная погрешность имеет вид:

Листинг программы

import java.util.\*;  
  
class F {  
 public static double getValue(double t, double u) {  
 return (Math.*pow*(u, 2.) \* Math.*log*(t) - u) / t;  
 }  
  
 public static double getDuValue(double t, double u) {  
 return (2. \* u \* Math.*log*(t) - 1) / t;  
 }  
}  
  
class U {  
 public static double getValue(double t) {  
 return 1. / (Math.*log*(t) + 1);  
 }  
}  
  
class IncreasingAccuracyMethod {  
 public static double[] getY(int n, double tau, double[] t, double u0) {  
 double[] res = new double[n + 1];  
 res[0] = u0;  
  
 double y12;  
 for (int i = 1; i <= n; i++) {  
 y12 = res[i - 1] + 0.5 \* tau \* F.*getValue*(t[i - 1], res[i - 1]);  
 res[i] = res[i - 1] + tau \* F.*getValue*(t[i - 1] + 0.5 \* tau, y12);  
 }  
  
 return res;  
 }  
}  
  
class CauchyProblem {  
 private final static int *N* = 10;  
 private final static double *A* = 1.;  
 private final static double *B* = 2.;  
 private final static double *U0* = 1.;  
 private final double tau;  
 private final double[] t;  
 private final double[] u;   
 private double[] y3;   
 private double[] res3;   
  
  
 public CauchyProblem() {  
 this.tau = (*B* - *A*) / *N*;  
  
 this.t = new double[*N* + 1];  
 this.t[0] = *A*;  
 for (int i = 1; i <= *N*; i++) {  
 this.t[i] = this.t[i - 1] + this.tau;  
 }  
  
 this.u = new double[*N* + 1];  
 for (int i = 0; i <= *N*; i++) {  
 this.u[i] = U.*getValue*(this.t[i]);  
 }  
 }  
  
 public void solve() {  
 this.y3 = IncreasingAccuracyMethod.*getY*(*N*, this.tau, this.t, *U0*);  
 this.res3 = this.getResidual(this.y3);   
 }  
  
 public void outResult() {  
 Formatter fmt = new Formatter();  
  
 fmt.format("Точное решение уравнения:\n");  
 for (int i = 0; i <= *N*; i++) {  
 fmt.format("%.7f\n", this.u[i]);  
 }  
  
 fmt.format("\nЗадание 3. Метод последовательного повышения порядка точности.\n");  
 this.outYAndRes(fmt, this.y3, this.res3);   
  
 System.*out*.println(fmt);  
 }  
  
 private void outYAndRes(Formatter fmt, double[] y, double[] res) {  
 fmt.format("Полученное решение:\n");  
 for (int i = 0; i <= *N*; i++) {  
 fmt.format("%.7f\n", y[i]);  
 }  
 fmt.format("Вектор невязок:\n");  
 for (int i = 0; i <= *N*; i++) {  
 fmt.format("%E\n", res[i]);  
 }  
 }  
  
 private double[] getResidual(double[] y) {  
 double[] res = new double[*N* + 1];  
 for (int i = 0; i <= *N*; i++) {  
 res[i] = Math.*abs*(this.u[i] - y[i]);  
 }  
  
 return res;  
 }  
}  
  
public class Main {  
  
 public static void main(String[] args) {  
 CauchyProblem cp = new CauchyProblem();  
 cp.solve();  
 cp.outResult();  
 }  
}

Результаты



Выводы

Реальная погрешность соответствует виду главного члена локальной погрешности (для первого значения она составила примерно ). Для вычисления использовался метод последовательного повышения порядка точности 2-го порядка, вид локальной погрешности – . Погрешность метода получилась чуть больше (примерно в раз) погрешности прошлого метода, метода Рунге-Кутта, но они все равно похожи, т.к. оба являются методами 2-го порядка. Разница объясняется разницей в значении константы в главном члене локальной погрешности (а также есть добавка ). Данный метод всё ещё точнее, чем неявный метод Эйлера, порядок которого на 1 меньше.

Экстраполяционный метод Адамса

Краткие теоретические сведения

Экстраполяционный (явный) метод Адамса 3-го порядка записывается в следующем виде:

Для реализации метода требуется задать значения , которые называются началом таблицы. Для этого используем метод последовательного повышения порядка точности того же порядка, что и метод Адамса, т.е. 3-го порядка. Используем формулу, базирующуюся на квадратурной формуле Симпсона:

*.*

Локальная погрешность экстраполяционного метода Адамса 3-го порядка записывается в виде:

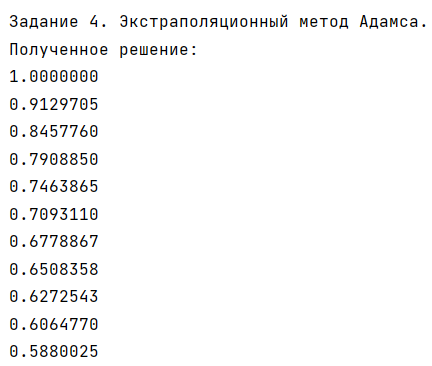
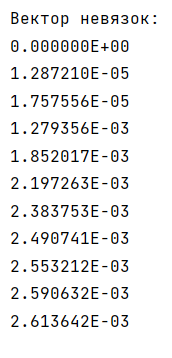
Используемый метод последовательного повышения порядка также является методом 3-го порядка, и его локальная погрешность записывается аналогично:

Листинг программы

import java.util.\*;  
  
class F {  
 public static double getValue(double t, double u) {  
 return (Math.*pow*(u, 2.) \* Math.*log*(t) - u) / t;  
 }  
  
 public static double getDuValue(double t, double u) {  
 return (2. \* u \* Math.*log*(t) - 1) / t;  
 }  
}  
  
class U {  
 public static double getValue(double t) {  
 return 1. / (Math.*log*(t) + 1);  
 }  
}  
  
class MultistepMethod {  
 public static double[] getY(int n, double tau, double[] t, double u0) {  
 double[] res = new double[n + 1];  
  
 res[0] = u0;  
 res[1] = MultistepMethod.*getYIncreasingMethod*(tau, t[0], res[0]);  
 res[2] = MultistepMethod.*getYIncreasingMethod*(tau, t[1], res[1]);  
  
 for (int i = 3; i <= n; i++) {  
 res[i] = res[i - 1] + (tau / 12.) \*  
 (23. \* F.*getValue*(t[i - 1], res[i - 1]) -  
 16. \* F.*getValue*(t[i - 2], res[i - 2]) +  
 5. \* F.*getValue*(t[i - 3], res[i - 3]));  
 }  
  
 return res;  
 }  
  
 private static double getYIncreasingMethod(double tau, double t, double y) {  
 double y12;  
 double y14;  
 double y11;  
 double res;  
  
 y14 = y + (tau / 4.) \* F.*getValue*(t, y);  
 y12 = y + (tau / 2.) \* F.*getValue*(t + (tau / 4.), y14);  
 y11 = y + tau \* F.*getValue*(t + (tau / 2.), y12);  
 res = y + (tau / 6.) \* (F.*getValue*(t, y) + 4. \* F.*getValue*(t + (tau / 2.), y12) + F.*getValue*(t + tau, y11));  
  
 return res;  
 }  
}  
  
class CauchyProblem {  
 private final static int *N* = 10;  
 private final static double *A* = 1.;  
 private final static double *B* = 2.;  
 private final static double *U0* = 1.;  
 private final double tau;  
 private final double[] t;  
 private final double[] u;   
 private double[] y4;   
 private double[] res4;  
  
  
 public CauchyProblem() {  
 this.tau = (*B* - *A*) / *N*;  
  
 this.t = new double[*N* + 1];  
 this.t[0] = *A*;  
 for (int i = 1; i <= *N*; i++) {  
 this.t[i] = this.t[i - 1] + this.tau;  
 }  
  
 this.u = new double[*N* + 1];  
 for (int i = 0; i <= *N*; i++) {  
 this.u[i] = U.*getValue*(this.t[i]);  
 }  
 }  
  
 public void solve() {  
 this.y4 = MultistepMethod.*getY*(*N*, this.tau, this.t, *U0*);  
 this.res4 = this.getResidual(this.y4);  
 }  
  
 public void outResult() {  
 Formatter fmt = new Formatter();  
  
 fmt.format("Точное решение уравнения:\n");  
 for (int i = 0; i <= *N*; i++) {  
 fmt.format("%.7f\n", this.u[i]);  
 }

fmt.format("\nЗадание 4. Экстраполяционный метод Адамса.\n");  
 this.outYAndRes(fmt, this.y4, this.res4);  
  
 System.*out*.println(fmt);  
 }  
  
 private void outYAndRes(Formatter fmt, double[] y, double[] res) {  
 fmt.format("Полученное решение:\n");  
 for (int i = 0; i <= *N*; i++) {  
 fmt.format("%.7f\n", y[i]);  
 }  
 fmt.format("Вектор невязок:\n");  
 for (int i = 0; i <= *N*; i++) {  
 fmt.format("%E\n", res[i]);  
 }  
 }  
  
 private double[] getResidual(double[] y) {  
 double[] res = new double[*N* + 1];  
 for (int i = 0; i <= *N*; i++) {  
 res[i] = Math.*abs*(this.u[i] - y[i]);  
 }  
  
 return res;  
 }  
}  
  
public class Main {  
  
 public static void main(String[] args) {  
 CauchyProblem cp = new CauchyProblem();  
 cp.solve();  
 cp.outResult();  
 }  
}

Результаты

Выводы

Из результатов видно, что реальная погрешность начала таблицы соответствует виду главного члена локальной погрешности (для первого значения она составила примерно ). Для вычисления начала таблицы использовался метод последовательного повышения порядка точности 3-го порядка, вид локальной погрешности – . Т.е. начало таблицы было вычислено чуть точнее, за счёт константы в главном члене локальной погрешности, она уменьшила погрешность на один порядок.

Что касается самого экстраполяционного метода Адамса, то здесь погрешность получилась чуть больше ожидаемого. Метод является методом 3-го порядка, его локальная погрешность – . Для третьего значения фактическая погрешность вышла примерно , т.е. на порядок больше, чем .

В итоге метод Рунге-Кутта, который является методом 2-го порядка, на практике для нашей задачи вышел самым точным, в том числе точнее метода Адамса 3-го порядка. Ниже приведены графики производных, которые фигурируют в локальных погрешностях этих методов. Из графиков видно, что производная для экстраполяционного метода Адамса на отрезке по модулю больше производной в погрешности метода Рунге-Кутта, в точке в раз. Этим и можно объяснить такую разницу в методах.

Заметим, что метод последовательного повышения порядка точности является довольно тяжелым в вычислениях в общем случае (вычислительная трудность заключается в том, сколько раз вычисляется значение функции правой части задачи в некоторой точке), именно поэтому есть смысл использовать экстраполяционный метод Адамса, он менее тяжелый в вычислениях.

