БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №3

**Методы решения задачи Коши**

Вариант 8

**Выполнила:**

Зуйкевич Лидия

3 курс 7 группа

**Преподаватель:**

Будник А.М.

Минск, 2022

Оглавление

Постановка задачи 3

[1. Неявный метод Эйлера 4](#_Toc122396302)

[Алгоритм решения 4](#_Toc122396303)

[Листинг программы 5](#_Toc122396303)

[Вывод программы 7](#_Toc122396303)

[2. Метод Рунге-Кутта 7](#_Toc122396304)

[Алгоритм решения 7](#_Toc122396303)

[Листинг программы 8](#_Toc122396303)

[Вывод программы 10](#_Toc122396303)

[3. Метод последовательного повышения порядка точности 10](#_Toc122396306)

[Алгоритм решения 10](#_Toc122396303)

[Листинг программы 12](#_Toc122396303)

[Вывод программы 13](#_Toc122396303)

[4. Экстраполяционный метод Адамса 13](#_Toc122396308)

[Алгоритм решения 13](#_Toc122396303)

[Листинг программы 14](#_Toc122396303)

[Вывод программы 15](#_Toc122396303)

[Выводы 15](#_Toc122396303)

Постановка задачи

Найти приближенное 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.

|  |  |
| --- | --- |
| 0  1 | 0 0  1 0 |
|  |  |

1. Метод Рунге-Кутта, построенный по таблице Бутчера вида:
2. Метод последовательного повышения порядка точности 2-го порядка при .
3. Экстраполяционный метод Адамса 3-го порядка с началом таблицы, построенным по соответствующему методу последовательного повышения порядка точности.

Для проведения анализа необходимо:

1. Используя таблицу приближенных результатов, получить погрешности методов 1 – 3, считая «точным решением» метод Адамса.
2. Исходя из вида главного члена локальной погрешности методов 1 – 3, объяснить разницу результатов.
3. На основе полученных численных и теоретических результатов сделать вывод о точности каждого метода 1 – 4.

# Неявный метод Эйлера

Алгоритм решения

Зададим сетку узлов на :

Неявный метод Эйлера имеет вид:

Где - приближенное значение искомой функции в точке , а - начальное условие из задачи Коши.

Т.к. метод неявный, то для нахождения каждого мы используем . Для того, чтобы найти , по заданию нужно использовать метод Ньютона, формула которого:

Для уравнения .

В нашем случае:

Тогда производная:, где – производная функции по .

И метод Ньютона для нахождения будет иметь вид:

В качестве начального приближения каждый раз будем брать: .

Вычисления продолжаем, пока не выполняется условие остановки внутреннего итерационного процесса . Т.к. неявный метод Эйлера имеет 1 порядок точности, то возьмем .

Для нашей задачи:

Тогда в итоге получим следующий метод для нахождения решения Задачи Коши в узлах сетки:

Где находится следующим образом:

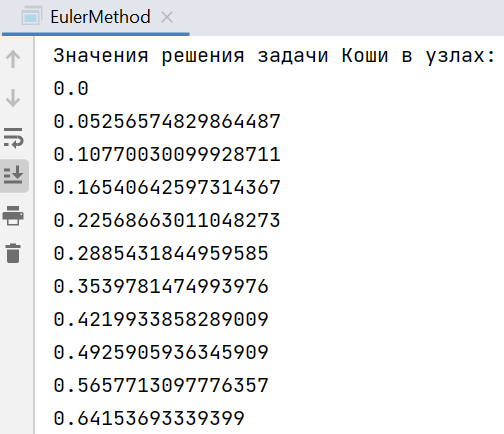
Пока .

# Листинг программы

public class EulerMethod {  
 private final int N;  
 private final double a;  
 private final double b;  
 private final double h;  
 private final double[] t;  
 private double[] y;  
 private final double u0;  
 private final double EPS;  
  
 public EulerMethod(int n, double a, double b, double u0) {  
 N = n;  
 this.a = a;  
 this.b = b;  
 this.h = (b - a) / N;  
 this.t = new double[N + 1];  
 for (int i = 0; i <= N; i++){  
 t[i] = a + h \* i;  
 }  
 this.y = new double[N + 1];  
 this.u0 = u0;  
 this.EPS = Math.pow(h, 2);  
 }  
  
 public double f(double y, double t){  
 return (y + Math.sqrt(Math.pow(t, 2) + Math.pow(y, 2))) / t;  
 }  
  
 public double f\_(double y, double t){  
 return 1 / t + (2 \* y) / (2 \* t \* Math.sqrt(Math.pow(t, 2) + Math.pow(y, 2)));  
 }  
  
 public double function (double yk, int i){  
 return yk - (yk - y[i] - h \* f(yk, t[i + 1])) / (1 - h \* f\_(yk, t[i + 1]));  
 }  
  
 public double newtonMethod(int i){  
 double y1 = y[i];  
 double y2 = function(y1, i);  
  
 while (Math.abs(y2 - y1) > EPS){  
 y1 = y2;  
 y2 = function(y1, i);  
 }  
  
 return y2;  
 }  
  
 public void solve(){  
 y[0] = u0;  
 double yk;  
  
 for (int i = 0; i < N; i++){  
 yk = newtonMethod(i);  
 y[i + 1] = y[i] + h \* f(yk, t[i + 1]);  
  
 }

System.*out*.println("Решение задачи Коши в узлах: ");  
 for(double elem: y){  
 System.out.println(elem);  
 }  
 }  
  
 public static void main(String[] args){  
 EulerMethod obj = new EulerMethod(10, 1., 1.5, 0.);  
 obj.solve();  
 }  
}

Вывод программы



# Метод Рунге-Кутта

Алгоритм решения

Способ Рунге-Кутта построения методов численного решения задачи Коши основан на следующем:

У нас есть сетка узлов, на которой мы хотим найти решение, и решение в точке , нужно перейти к точке . От исходного дифференциального уравнения переходим к некоторому интегральному соотношению, например, интегрируем уравнение на промежутке :

Делаем замену в интеграле так, чтобы промежуток интегрирования был . Для приближенного вычисления интеграла заменяем его квадратурной формулой на той же сетке узлов, на которой хотим найти решение, а затем рассматриваем решение в каждой точке сетки и, исходя из требуемого порядка точности, определяем неизвестные параметры.

Формула метода Рунге-Кутта имеет вид:

|  |  |
| --- | --- |
| c | A |
|  |  |

Где ; b, c - s-мерные вектора, A - матрица размерности .

Таблица Бутчера для записи метода имеет вид:

|  |  |
| --- | --- |
| 0  1 | 0 0  1 0 |
|  |  |

В нашем случае:

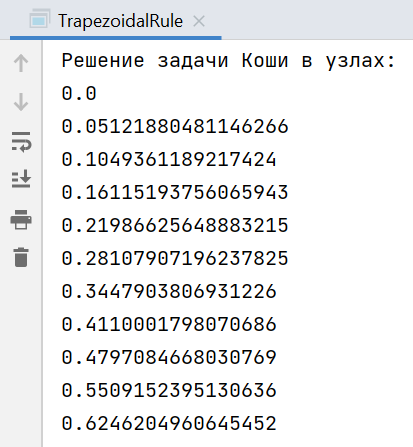
Тогда метод (формула метода – аналоги формулы трапеций):

В нашем случае:

# Листинг программы

public class TrapezoidalRule {  
 private final int N;  
 private final double a;  
 private final double b;  
 private final double h;  
 private final double[] t;  
 private double[] y;  
 private final double u0;  
  
 public TrapezoidalRule(int n, double a, double b, double u0) {  
 N = n;  
 this.a = a;  
 this.b = b;  
 this.h = (b - a) / N;  
 this.t = new double[N + 1];  
 for (int i = 0; i <= N; i++){  
 t[i] = a + h \* i;  
 }  
 this.y = new double[N + 1];  
 this.u0 = u0;  
 }  
  
 public double[] getY() {  
 return y;  
 }  
  
 public double f(double y, double t){  
 return (y + Math.*sqrt*(Math.*pow*(t, 2) + Math.*pow*(y, 2))) / t;  
 }  
  
  
 public void solve(){  
 y[0] = u0;  
  
 for (int i = 0; i < N; i++){  
 y[i + 1] = y[i] + (h / 2) \* (f(y[i], t[i]) + f(y[i] + h \* f(y[i], t[i]), t[i + 1]));  
 }  
  
 System.*out*.println("Решение задачи Коши в узлах: ");  
 for(double elem: y){  
 System.*out*.println(elem);  
 }  
 }  
  
 public static void main(String[] args){  
 TrapezoidalRule obj = new TrapezoidalRule(10, 1., 1.5, 0.);  
 obj.solve();  
 }  
}

Вывод программы



# 3. Метод последовательного повышения порядка точности

Алгоритм решения

Метод последовательного повышения порядка точности основан на той же идее, что и метод Рунге-Кутта:

У нас есть сетка узлов, на которой мы хотим найти решение, и решение в точке , нужно перейти к точке . От исходного дифференциального уравнения переходим к некоторому интегральному соотношению, например, интегрируем уравнение на промежутке :

Делаем замену в интеграле так, чтобы промежуток интегрирования был . Для приближенного вычисления интеграла заменяем его квадратурной формулой на той же сетке узлов, на которой хотим найти решение, а затем рассматриваем решение в каждой точке сетки и, исходя из требуемого порядка точности, определяем неизвестные параметры. Однако при получении в формуле значения в точке, которого у нас нет (например, неявный метод или в точке ), мы добавляем к методу формулу, по которой будем находить значение в этой точке. Эта формула должна иметь порядок на 1 меньше порядка нашего метода, чтобы не ухудшить порядок нашего метода. Если эта формула та же использует значение в некоторой точке, которого у нас нет, поступаем тем же образом: добавляем еще одну формулу для вычисления этого значения, имеющую порядок на 1 меньше предыдущей формулы. При этом участвующие в формулах значения решения в точках, которые у нас есть, берем найденные с порядком точности самой первой формулы.

Формула метода имеет вид:

Система для определения коэффициентов:

Где - требуемый порядок точности метода.

В нашем случае:

Получаем значение коэффициентов:

И тогда метод:

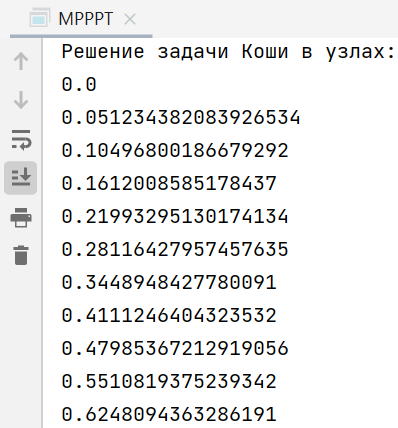
Нужно найти значение с локальной погрешностью 2 порядка. Для этого воспользуемся формулой явного метода Эйлера с шагом , получим метод:

(В квадратных скобках указан порядок локальной погрешности).

# Листинг программы

public class MPPPT {  
 private final int N;  
 private final double a;  
 private final double b;  
 private final double h;  
 private final double[] t;  
 private double[] y;  
 private final double u0;  
  
 public MPPPT(int n, double a, double b, double u0) {  
 N = n;  
 this.a = a;  
 this.b = b;  
 this.h = (b - a) / N;  
 this.t = new double[N + 1];  
 for (int i = 0; i <= N; i++){  
 t[i] = a + h \* i;  
 }  
 this.y = new double[N + 1];  
 this.u0 = u0;  
 }  
  
 public double[] getY() {  
 return y;  
 }  
  
 public double f(double y, double t){  
 return (y + Math.*sqrt*(Math.*pow*(t, 2) + Math.*pow*(y, 2))) / t;  
 }  
  
 public double euler(int i){  
 return y[i] + (h / 2) \* f(y[i], t[i]);  
 }  
  
 public void solve(){  
 y[0] = u0;  
  
 for (int i = 0; i < N; i++){  
 y[i + 1] = y[i] + h \* f(euler(i), t[i] + h / 2);  
 }  
  
 System.*out*.println("Решение задачи Коши в узлах: ");  
 for(double elem: y){  
 System.*out*.println(elem);  
 }  
 }  
  
 public static void main(String[] args){  
 MPPPT obj = new MPPPT(10, 1., 1.5, 0.);  
 obj.solve();  
 }  
}

Вывод программы



# 4. Экстраполяционный метод Адамса

Алгоритм решения

Формула экстраполяционного метода Адамса 3 порядка имеет вид:

Где .

В формуле при используются значения приближенного решения в точках . Значение в точке известно – это начальные условия задачи Коши. Значения в точках (начало таблицы), согласно заданию, необходимо найти по соответствующему методу последовательного повышения порядка точности. Для этого воспользуемся следующей формулой метода, имеющего локальную погрешность порядка :

Где . В нашем случае:

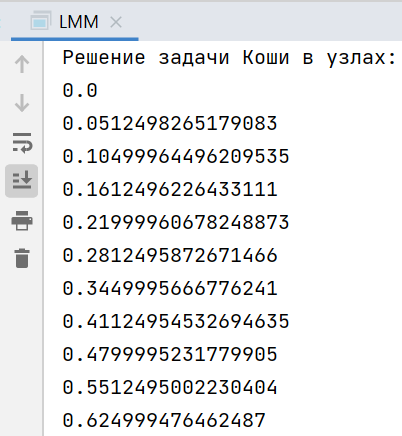
И тогда получим метод:

Где

# Листинг программы

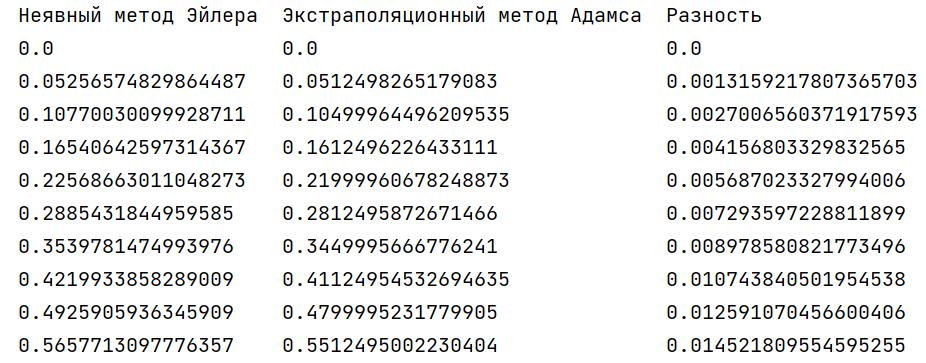
public class LMM {  
 private final int N;  
 private final double a;  
 private final double b;  
 private final double h;  
 private final double[] t;  
 private double[] y;  
 private final double u0;  
  
 public LMM(int n, double a, double b, double u0) {  
 N = n;  
 this.a = a;  
 this.b = b;  
 this.h = (b - a) / N;  
 this.t = new double[N + 1];  
 for (int i = 0; i <= N; i++){  
 t[i] = a + h \* i;  
 }  
 this.y = new double[N + 1];  
 this.u0 = u0;  
 }  
  
 public double[] getY() {  
 return y;  
 }  
  
 public double f(double y, double t){  
 return (y + Math.*sqrt*(Math.*pow*(t, 2) + Math.*pow*(y, 2))) / t;  
 }  
  
 public void mpppt(int i){  
 double y2 = y[i - 1] + (h / 3) \* f(y[i - 1], t[i - 1]);  
 double y3 = y[i - 1] + (2 \* h / 3) \* f(y2, t[i - 1] + h / 3);  
 y[i] = y[i - 1] + (h / 4) \* (f(y[i - 1], t[i - 1]) + 3 \* f(y3, t[i - 1] + 2 \* h /3));  
 }  
  
 public void solve(){  
 y[0] = u0;  
 mpppt(1);  
 mpppt(2);  
  
 for (int i = 2; i < N; i++){  
 y[i + 1] = y[i] + (h / 12) \* (23 \* f(y[i], t[i]) - 16 \* f(y[i - 1], t[i - 1]) + 5 \* f(y[i - 2], t[i - 2]));  
 }  
  
 System.*out*.println("Решение задачи Коши в узлах: ");  
 for(double elem: y){  
 System.*out*.println(elem);  
 }  
 }  
  
 public static void main(String[] args){  
 LMM obj = new LMM(10, 1., 1.5, 0.);  
 obj.solve();  
 }  
}

Вывод программы

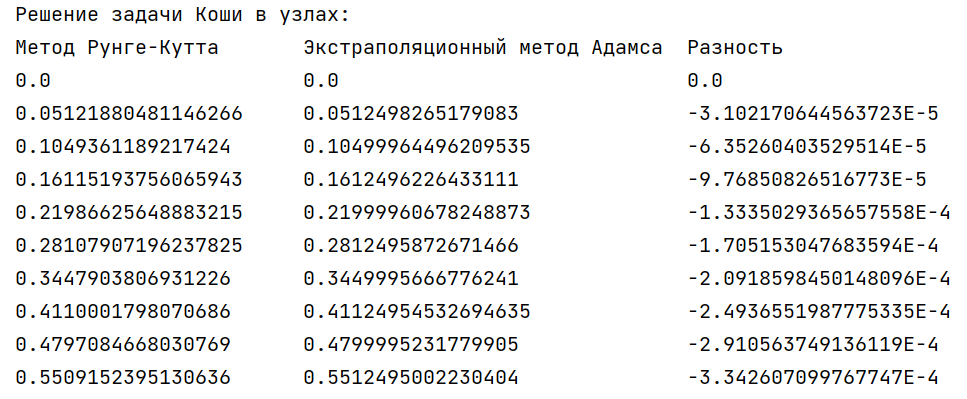


Выводы

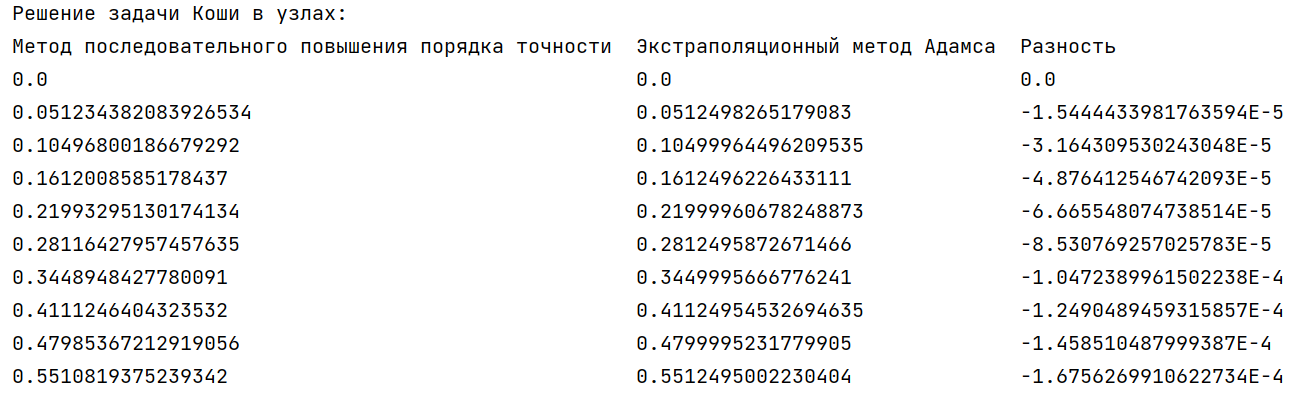
Сравним полученные результаты, считая “точным” решением решение, полученное методом Адамса.



Неявный метод Эйлера является методом 1 порядка точности, т.е. имеет локальную погрешность порядка . В нашем случае , . Найденное методом Эйлера решение совпадает с решением, найденным методом Адамса до 2 знаков после запятой, что подтверждает локальную погрешность метода Эйлера порядка . Стоит отметить, что для реализации неявного метода Эйлера был использован метод Ньютона, что значительно увеличило количество вычислений. Неявный же метод Эйлера не требует дополнительных вычислений, но тоже является методом 1 порядка точности, поэтому полученные с помощью него результаты должны быть схожими, однако количество вычислений значительно меньше.



Метод Рунге-Кутта, построенный в пункте 2, имеет 2 порядок точности и локальную погрешность порядка . В нашем случае , . Разница между значениями, полученными методом Рунге-Кутта и методом Адамса, имеет тот же порядок. Отметим, что реализовать этот метод было проще всего, т.к. он не требовал никаких дополнительных вычислений.



Метод последовательного повышения порядка точности из п.3 имеет 2 порядок точности и локальную погрешность порядка . В нашем случае , . Разница между значениями, полученными методом последовательного повышения порядка точности и методом Адамса, имеет тот же порядок. Для нахождения значения в каждой точке потребовалось дополнительно найти значение в точке , для этого была использована формула метода Эйлера с шагом .

В п.4 для нахождения решения был использован метод Адамса 3 порядка, имеющий локальную погрешность порядка . Полученные методом Адамса результаты были использованы в качестве «точного» решения для определения погрешности результатов, полученных другими методами, и их сравнения. Т.к. метод экстраполяционный, для его реализации было построено начало таблицы, а именно значения приближенного решения в точках и , для этого был использован одношаговый метод последовательного повышения порядка точности 3 порядка. Для вычисления каждого из 2 значений использовалось 3 формулы.