|  |
| --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  Учреждение образования «Полоцкий государственный университет»    Факультет информационных технологий  Кафедра технологий программирования  ОТЧЕТ по  ЛАБОРАТОРНОЙ РАБОТЕ №4  По дисциплине  **«Методы численного анализа»**  На тему: «Метод редукции, метод стрельбы для решения краевой задачи ОДУ»  ВЫПОЛНИЛ студент группы 15-КБ  Осипенко В.В.  ПРОВЕРИЛА ассистент кафедры ТП  Рудькова Т.С.  Полоцк, 2017 г. |

**Название**: «Метод редукции, метод стрельбы для решения краевой задачи ОДУ».

**Цель работы**: Изучить основные методы для нахождения решен0ия краевой задачи обыкновенных дифференциальных уравнений

**Теоретическая часть**:

**Метод редукции.**

Пусть на отрезке [a, b] задано линейное ОДУ n-го порядка с непрерывными коэффициентами i = 0,1,2,...,n и функцией правой части f(x):

При этом для x∈ a b.

Кроме этого на отрезке x∈[a,b] заданы граничные условия вида:

i=1,2,…,n

Требуется найти решение поставленной краевой задачи (2)-(3) на отрезке [a,b] методом редукции. Суть метода состоит в том, чтобы поиск решения свести к решению задач Коши. Известно, что искомое решение задачи (2)- (3) можно представить в виде линейной комбинации:

где Y0 – является частным решением неоднородного ДУ (2), а Yi ,i=1,2,…,n– линейно независимые решения однородного ДУ

c - произвольные константы.

Метод редукции к задачам Коши состоит из следующих этапов:

1. На отрезке поиска решения [a,b] любым известным методом находим численные решения Yi ,i=0,1,2,…,n.

2. Используя общий вид решения и краевые условия, определяем .

3. Вычисляем по формуле искомое решение задачи.

**Метод стрельбы**

Рассмотрим метод стрельбы для двухточечной краевой задачи на основе ОДУ-2.

где p(x),q(x),f(x) – известные функции, определенные на отрезке поиска решения, а параметры , , , , имеют конкретное числовое значение, причем выполняется условие

Суть этого метода заключается в сведении решения краевой задачи к многократному решению задач Коши. Будем предполагать, что отрезок поиска решения задачи [0,1]. Известно, что любой отрезок [a,b] можно заменить отрезком [0,1], путем ввода замены переменной вида:

Граничные условия на концах рассматриваемого отрезка также примем в простейшем виде, т.е. u(0) ~y(0)=Y0, u(1) ~y(1)=Y1

Имеем задачу (1)-(3). Доказывается, что в случае, когда для двух линейно независимых решений Y0 и Y1 этой задачи одно граничное условие выполняется, например

то общее решение задачи (1)-(3) будет теперь зависеть от одного произвольного параметра: u(x)~Y(x) =Y0(x) + CY1(x)

Параметр С находим из второго граничного условия u(b) + u ′(b)= . Получим: C=

Для реализации такого алгоритма выбираем на левом конце одно из начальных условий, вообще говоря, произвольно. Например, если ≠ 0, то принимаем y ′(a) = ϕ1 тогда из граничного условия находим и решаем задачу Коши.

Затем выбираем другое значение y ′(a) =ϕ2 так, чтобы ϕ2 ≠ ϕ1 (т.е. решения при этих ϕ1,ϕ2 должны быть линейно независимы). Решаем полученную задачу Коши:

**Вариант №9**

Найти решения граничных задач с шагом h=0.1 на отрезке [a,b]:

1. используя метод редукции;

2. используя метод стрельбы.

**Ход работы:**

Составим график, на котором изображена функция значения которой будем брать за точные (Рисунок 1).

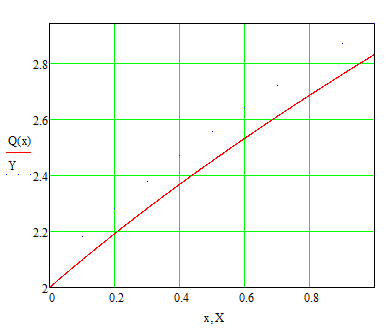


Рисунок 1 – График функции.

Напишем программу для первого метода и оценим её точность.

Для решения поставленной задачи нам необходимо составить три системы вида (Рисунок 2).

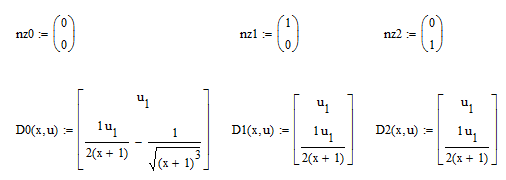


Рисунок 2 – Системы уравнений

Листинг 1 – Программа метода Редукции написанная на JAVA.

**public static double** asdf(**double** x){  
 **return** 2\*Math.*sqrt*(1+x);  
}  
**public static double** qwer(**double** x1, **double** x2){  
 **return** Math.*abs*(x1-x2)/Math.*abs*(x2);  
}  
**public static double** f(**double** x, **double** y, **double** z){  
 **return** z;  
}  
**public static double** k1(**double** x, **double** y, **double** z, **double** h){  
 **return** *f*(x,y,z);  
}  
**public static double** k2(**double** x, **double** y, **double** z, **double** h,**double** v){  
 **return** *f*(x+(h/2), y+((h/2)\**k1*(x,y,z,h)),z + h/2\**l1*(x,y,z,h,v));  
}  
**public static double** k3(**double** x, **double** y, **double** z, **double** h,**double** v){  
 **return** *f*(x+h/2, y + (h/2)\**k2*(x,y,z,h,v),z+(h/2)\**l2*(x,y,z,h,v));  
}  
**public static double** k4(**double** x, **double** y, **double** z, **double** h,**double** v){  
 **return** *f*(x+h,y+h\**k3*(x,y,z,h,v),z+h\**l3*(x,y,z,h,v));  
}  
**public static double** yn(**double** x, **double** y, **double** z, **double** h, **double** v){  
 **return** y + (h/6)\*(*k1*(x,y,z,h)+2\**k2*(x,y,z,h,v)+2\**k3*(x,y,z,h,v)+*k4*(x,y,z,h,v));  
}  
**public static double** z(**double** x, **double** y, **double** z, **double** v){  
 **double** res = 0;  
 **if**(v == 0) {  
 res = (1/(2\*(x+1)))\*z-1/(Math.*pow*(x+1,3/2));  
 }  
 **if**(v == 1){  
 res = (1/(2\*(x+1)))\*z;  
 }  
 **return** res;  
}  
**public static double** l1(**double** x, **double** y, **double** z, **double** h,**double** v){  
 **return** *z*(x,y,z,v);  
}  
**public static double** l2(**double** x, **double** y, **double** z, **double** h, **double** v){  
 **return** *z*(x+(h/2), y+((h/2)\**k1*(x,y,z,h)),z + h/2\**l1*(x,y,z,h,v), v);  
}  
**public static double** l3(**double** x, **double** y, **double** z, **double** h, **double** v){  
 **return** *z*(x+h/2, y + (h/2)\**k2*(x,y,z,h,v),z+(h/2)\**l2*(x,y,z,h,v),v);  
}  
**public static double** l4(**double** x, **double** y, **double** z, **double** h, **double** v){  
 **return** *z*(x+h,y+h\**k3*(x,y,z,h,v),z+h\**l3*(x,y,z,h,v), v);  
}  
**public static double** zn(**double** x, **double** y, **double** z, **double** h, **double** v){  
 **return** z + (h/6)\*(*l1*(x,y,z,h,v)+2\**l2*(x,y,z,h,v)+2\**l3*(x,y,z,h,v)+*l4*(x,y,z,h,v));  
}  
**public static void** main(String[] args){  
 **double** h = 0.01;  
 **double** y0 = 0;  
 **double** z0 = 0;  
 **double** y1 = 1;  
 **double** z1 = 0;  
 **double** y2 = 0;  
 **double** z2 = 1;  
 **double** xi[]= **new double**[12];  
 **double** Y0[] = **new double**[11];  
 **double** Y1[] = **new double**[11];  
 **double** Y2[] = **new double**[11];  
 **double** S0[] = **new double**[11];  
 **double** S1[] = **new double**[11];  
 **double** S2[] = **new double**[11];  
 **double** Yi[] = **new double**[11];  
 **int** i = 0;  
 **int** k = 0;  
 xi[0]=0;  
 Y0[0]=0;  
 S0[0]=0;  
 Y1[0]=1;  
 S1[0]=0;  
 Y2[0]=0;  
 S2[0]=1;  
 **for**(**double** x = 0; x<=1.01; x+=h){  
 **double** v = 0;  
 **double** ty;  
 **double** tz;  
 ty = *yn*(x,y0,z0,h,v);  
 tz = *zn*(x,y0,z0,h,v);  
 y0=ty;  
 z0=tz;  
 **if**(i == 10 || i == 20 || i == 30 || i == 40 || i == 50||i==60||i==70||i==80||i==90||i==100){  
 xi[k+1]=x;  
 Y0[k+1] = ty;  
 S0[k+1] = tz;  
 k++;  
 }  
 i++;  
 }  
 i = 0;  
 k = 0;  
 **for**(**double** x = 0; x<=1.01; x+=h){  
 **double** v = 1;  
 **double** ty;  
 **double** tz;  
 ty = *yn*(x,y1,z1,h,v);  
 tz = *zn*(x,y1,z1,h,v);  
 y1=ty;  
 z1=tz;  
 **if**(i == 10 || i == 20 || i == 30 || i == 40 || i == 50||i==60||i==70||i==80||i==90||i==100){  
 Y1[k+1] = ty;  
 S1[k+1] = tz;  
 k++;  
 }  
 i++;  
 }1  
 i = 0;  
 k = 0;  
 **for**(**double** x = 0; x<=1.01; x+=h){  
 **double** v = 1;  
 **double** ty;  
 **double** tz;  
 ty = *yn*(x,y2,z2,h,v);  
 tz = *zn*(x,y2,z2,h,v);  
 y2=ty;  
 z2=tz;  
 **if**(i == 10 || i == 20 || i == 30 || i == 40 || i == 50||i==60||i==70||i==80||i==90||i==100){  
 Y2[k+1] = ty;  
 S2[k+1] = tz;  
 k++;  
 }  
 i++;  
 }  
  
 **double** c2 = 1.068;  
 **double** c1 = 2.068;  
  
 **for**(**int** o = 0; o<11; o++){  
 Yi[o] = Y0[o] + c1\*Y1[o] + c2\*Y2[o];  
 }  
 System.***out***.println(**" X Y0 S0 Y1 S1 Y2 S2 Y Погрешность"**);  
 **for** (**int** j=0; j<11; j++) {  
 System.***out***.print(j);  
 System.***out***.print(String.*format*(**" %.1f"**, xi[j]));  
 System.***out***.print(String.*format*(**". %.4f"**,Y0[j]));  
 System.***out***.print(String.*format*(**". %.4f"**,S0[j]));  
 System.***out***.print(String.*format*(**". %.4f"**,Y1[j]));  
 System.***out***.print(String.*format*(**". %.4f"**,S1[j]));  
 System.***out***.print(String.*format*(**". %.4f"**,Y2[j]));  
 System.***out***.print(String.*format*(**". %.4f"**,S2[j]));  
 System.***out***.print(String.*format*(**". %.4f"**,Yi[j]));  
 System.***out***.print(String.*format*(**". %.4f"**,*asdf*(xi[j])));  
 System.***out***.print(String.*format*(**". %.4f"**,*qwer*(*asdf*(xi[j]),Yi[j])));  
 System.***out***.println(**"%"**);  
  
 }  
 System.***out***.print(**"c1 ="** + (**float**)c1 + **", c2 = "** + c2);  
}

Поскольку погрешность метода при шаге равном 0.1 слишком большая – мы уменьшили шаг до 0.01. Коэффициенты c1 и c2 находятся вручную, воспользуемся средствами MathCad и найдем их (Рисунок 3).

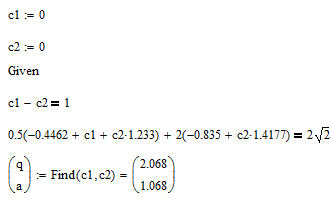


Рисунок 3 – Подсчет коэффициентов с помощью MathCad.

Результатом работы программы будет:

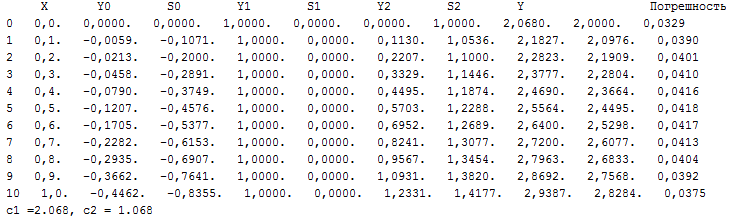


Рисунок 4 – Результат программы.

Как мы видим погрешность не высока, максимальная составила 0.418, построим график оценим погрешность графически.

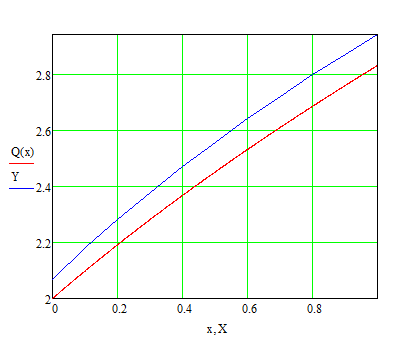


Рисунок 5 – График решений.

Как мы видим, погрешность данного метода не высока однако, она не стремится к минимуму и увеличивается со временем.

Напишем программу для второго метода – метода Стрельбы.

Листинг 2 – Программа - метод Стрельбы написанная на JAVA.

**public static double** f(**double** x, **double** y, **double** z) {  
 **return** z;  
}  
  
**public static double** k1(**double** x, **double** y, **double** z, **double** h) {  
 **return** *f*(x, y, z);  
}  
  
**public static double** k2(**double** x, **double** y, **double** z, **double** h) {  
 **return** *f*(x + (h / 2), y + ((h / 2) \* *k1*(x, y, z, h)), z + h / 2 \* *l1*(x, y, z, h));  
}  
  
**public static double** k3(**double** x, **double** y, **double** z, **double** h) {  
 **return** *f*(x + h / 2, y + (h / 2) \* *k2*(x, y, z, h), z + (h / 2) \* *l2*(x, y, z, h));  
}  
  
**public static double** k4(**double** x, **double** y, **double** z, **double** h) {  
 **return** *f*(x + h, y + h \* *k3*(x, y, z, h), z + h \* *l3*(x, y, z, h));  
}  
  
**public static double** yn(**double** x, **double** y, **double** z, **double** h) {  
 **return** y + (h / 6) \* (*k1*(x, y, z, h) + 2 \* *k2*(x, y, z, h) + 2 \* *k3*(x, y, z, h) + *k4*(x, y, z, h));  
}  
  
**public static double** z(**double** x, **double** y, **double** z) {  
 **return** (1/(2\*(x+1)))\*z-1/(Math.*pow*(x+1,3/2));  
}  
  
**public static double** l1(**double** x, **double** y, **double** z, **double** h) {  
 **return** *z*(x, y, z);  
}  
  
**public static double** l2(**double** x, **double** y, **double** z, **double** h) {  
 **return** *z*(x + (h / 2), y + ((h / 2) \* *k1*(x, y, z, h)), z + h / 2 \* *l1*(x, y, z, h));  
}  
  
**public static double** l3(**double** x, **double** y, **double** z, **double** h) {  
 **return** *z*(x + h / 2, y + (h / 2) \* *k2*(x, y, z, h), z + (h / 2) \* *l2*(x, y, z, h));  
}  
  
**public static double** l4(**double** x, **double** y, **double** z, **double** h) {  
 **return** *z*(x + h, y + h \* *k3*(x, y, z, h), z + h \* *l3*(x, y, z, h));  
}  
  
**public static double** zn(**double** x, **double** y, **double** z, **double** h) {  
 **return** z + (h / 6) \* (*l1*(x, y, z, h) + 2 \* *l2*(x, y, z, h) + 2 \* *l3*(x, y, z, h) + *l4*(x, y, z, h));  
}  
  
**public static void** main(String[] args) {  
 **double** h = 0.01;  
 **double** y0 = 2;  
 **double** z0 = 1;  
 **double** y1 = 1;  
 **double** z1 = 2;  
 **double** xi[] = **new double**[12];  
 **double** Y0[] = **new double**[11];  
 **double** Y1[] = **new double**[11];  
 **double** S0[] = **new double**[11];  
 **double** S1[] = **new double**[11];  
 **double** Yi[] = **new double**[11];  
 **int** i = 0;  
 **int** k = 0;  
 xi[0] = 0;  
 Y0[0] = y0;  
 S0[0] = z0;  
 Y1[0] = y1;  
 S1[0] = z1;  
 **for** (**double** x = 0; x <= 1.01; x += h) {  
 **double** ty;  
 **double** tz;  
 ty = *yn*(x, y0, z0, h);  
 tz = *zn*(x, y0, z0, h);  
 y0 = ty;  
 z0 = tz;  
 **if** (i == 10 || i == 20 || i == 30 || i == 40 || i == 50||i==60||i==70||i==80||i==90||i==100) {  
 xi[k + 1] = x;  
 Y0[k + 1] = ty;  
 S0[k + 1] = tz;  
 k++;  
 }  
 i++;  
 }  
 i = 0;  
 k = 0;  
 **for** (**double** x = 0; x <= 1.01; x += h) {  
 **double** ty;  
 **double** tz;  
 ty = *yn*(x, y1, z1, h);  
 tz = *zn*(x, y1, z1, h);  
 y1 = ty;  
 z1 = tz;  
 **if** (i == 10 || i == 20 || i == 30 || i == 40 || i == 50||i==60||i==70||i==80||i==90||i==100) {  
 Y1[k + 1] = ty;  
 S1[k + 1] = tz;  
 k++;  
 }  
 i++;  
 }  
 **double** C = (2\*Math.*sqrt*(2) - 0.5\*Y0[10]-2\*S0[10]) / (0.5\*Y1[10]+2\*S1[10]);  
  
  
 **for** (**int** o = 0; o < 11; o++) {  
 Yi[o] = Y0[o] + C \* Y1[o];  
 }  
 System.***out***.println(**" X Y0 S0 Y1 S1 Y"**);  
 **for** (**int** j = 0; j < 11; j++) {  
 System.***out***.print(j);  
 System.***out***.print(String.*format*(**" %.1f"**, xi[j]));  
 System.***out***.print(String.*format*(**". %.4f"**, Y0[j]));  
 System.***out***.print(String.*format*(**". %.4f"**, S0[j]));  
 System.***out***.print(String.*format*(**". %.4f"**, Y1[j]));  
 System.***out***.print(String.*format*(**". %.4f"**, S1[j]));  
 System.***out***.println(String.*format*(**". %.4f"**, Yi[j]));  
 }  
 System.***out***.print(**"c1 ="** + (**float**) C);  
}

Поскольку точность шага 0.1 была недостаточной мы взяли шаг 0.01.В качестве коэффициентов мы взяли 1, 2 и 2, 1.

Результатом программы будет:

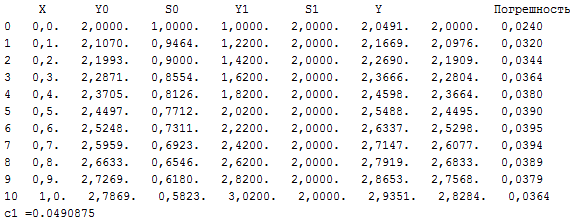


Рисунок 6 – Решение программы.

Как мы видим решение является достаточно точным, максимальная погрешность составила 0.395.

Построим график и проведем сравнение решений.

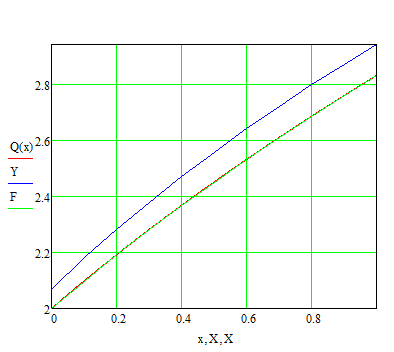


Рисунок 7 – График решений

Как мы видим точность второго метода высока и заметить погрешность графически трудно.

**Выводы о проделанной работе.**

В ходе данной лабораторной работы мы изучили два метода решения краевой задачи ОДУ, оценили их точность. Метод редукции оказался менее точным чем метод Стрельбы, однако его точность сильно зависит от подобранных коэффициентов и может варьироваться в зависимости от вида функции.