**Московский Авиационный Институт**

(Национальный Исследовательский Университет)

Институт №8 «Компьютерные науки и прикладная математика»

**Лабораторная работа №4**

**«Численные методы»**

**Вариант 9**

**Выполнила студентка группы**: М8О-305Б-21

Бондарева Елена Евгеньевна

# Преподаватель: Филиппов Глеб Сергеевич

Оценка: !

Дата: !

Москва, 2024

**Задание 4.1**

Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки . С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

|  |  |
| --- | --- |
| , |  |

**Теоретические сведения**

Рассматривается задача Коши для одного дифференциального уравнения первого порядка разрешенного относительно производной



Требуется найти решение на отрезке .

Введем разностную сетку на отрезке .

Точки называются узлами разностной сетки, расстояния между узлами – шагом разностной сетки, а совокупность значений какой-либо величины, заданных в узлах сетки называется сеточной функцией.

Приближенное решение задачи Коши будем искать численно в виде сеточной функции.

**Метод Эйлера:**

Метод Эйлера играет важную роль в теории численных методов решения ОДУ, хотя и не часто используется в практических расчетах из-за невысокой точности. Вывод расчетных соотношений для этого метода может быть произведен несколькими способами: с помощью геометрической интерпретации, с использованием разложения в ряд Тейлора, конечно-разностным методом (с помощью разностной аппроксимации производной), квадратурным способом (использованием эквивалентного интегрального уравнения).

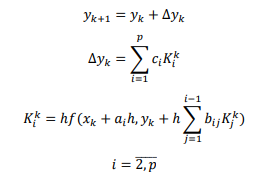
Рассмотрим вывод соотношений метода Эйлера геометрическим способом. Решение в узле известно из начальных условий. Рассмотрим процедуру получения решения в узле .

График функции , которая является решением задачи Коши, представляет собой гладкую кривую, проходящую через точку , согласно условию , и имеет в этой точке касательную. Тангенс угла наклона касательной к оси *Ох* равен значению производной от решения в точке и равен значению правой части дифференциального уравнения в точке согласно выражению . В случае небольшого шага разностной сетки *h* график функции и график касательной не успевают сильно разойтись друг от друга и можно в качестве значения решения в узле принять значение касательной , вместо значения неизвестного точного решения . При этом допускается погрешность геометрически представленная отрезком CD. Из прямоугольного треугольника ABC находим или . Учитывая, что и заменяя производную на правую часть дифференциального уравнения, получаем соотношение . Считая теперь точку начальной и повторяя все предыдущие рассуждения, получим значение в узле .

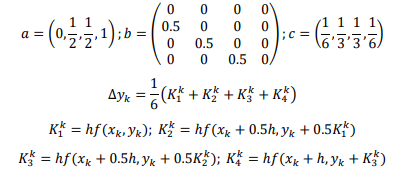
Переход к произвольным индексам дает формулу метода Эйлера:

**Метод Рунге-Кутты:**

Семейство явных методов Рунге-Кутты р-го порядка записывается в виде совокупности формул:



Метод Рунге-Кутты четвертого порядка:



Контроль шага интегрирования:

На каждом шаге h рассчитывается параметр



Если величина порядка нескольких сотых единицы, то расчет продолжается с тем же шагом, если больше одной десятой, то шаг следует уменьшить, если же меньше одной сотой, то шаг можно увеличить.

Таким образом с помощью определения величины можно организовать алгоритм выбора шага *h* для явного метода Рунге-Кутты.

**Метод Адамса:**

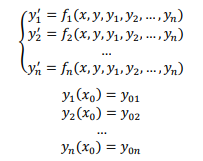
При использовании интерполяционного многочлена 3-ей степени построенного по значениям подынтегральной функции в последних четырех узлах получим метод Адамса четвертого порядка точности:



Метод Адамса, как и все многошаговые методы не является самостартующим, то есть для того, чтобы использовать метод Адамса необходимо иметь решения в первых четырех узлах. В узле решение известно из начальных условий, а в других трех узлах решения можно получить с помощью подходящего одношагового метода, например: метода Рунге-Кутты четвертого порядка.

***Решение задачи Коши для системы обыкновенных дифференциальных уравнений:***

Рассматривается задача Коши для системы дифференциальных уравнений первого порядка разрешенных относительно производной



К системе дифференциальных уравнений можно применить все методы, рассмотренные выше. Уравнения решаются по порядку

Для задачи Коши 2-го порядка можно применить следующее разложение в систему (используя замену :

**Код программы:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import solve\_ivp, solve\_bvp, odeint

from scipy.optimize import fsolve, newton, root\_scalar,root, minimize

from scipy.interpolate import interp1d

def euler\_method(f, x0, y0, z0, xn, h):

    n = int((xn - x0) / h) + 1

    x = np.linspace(x0, xn, n)

    y = np.zeros(n)

    z = np.zeros(n)

    y[0] = y0

    z[0] = z0

    for i in range(1, n):

        dy, dz = f(x[i-1], y[i-1], z[i-1])

        y[i] = y[i-1] + h \* dy

        z[i] = z[i-1] + h \* dz

    return x, y, z

def runge\_kutta\_method(f, x0, y0, z0, xn, h):

    n = int((xn - x0) / h) + 1

    x = np.linspace(x0, xn, n)

    y = np.zeros(n)

    z = np.zeros(n)

    y[0] = y0

    z[0] = z0

    for i in range(1, n):

        k1y, k1z = f(x[i-1], y[i-1], z[i-1])

        k2y, k2z = f(x[i-1] + h/2, y[i-1] + h/2 \* k1y, z[i-1] + h/2 \* k1z)

        k3y, k3z = f(x[i-1] + h/2, y[i-1] + h/2 \* k2y, z[i-1] + h/2 \* k2z)

        k4y, k4z = f(x[i-1] + h, y[i-1] + h \* k3y, z[i-1] + h \* k3z)

        y[i] = y[i-1] + h/6 \* (k1y + 2\*k2y + 2\*k3y + k4y)

        z[i] = z[i-1] + h/6 \* (k1z + 2\*k2z + 2\*k3z + k4z)

    return x, y, z

def adams\_bashforth\_method(f, x0, y0, z0, xn, h):

    n = int((xn - x0) / h) + 1

    x = np.linspace(x0, xn, n)

    y = np.zeros(n)

    z = np.zeros(n)

    y[0] = y0

    z[0] = z0

    \_, y\_rk4, z\_rk4 = runge\_kutta\_method(f, x0, y0, z0, x0 + 3 \* h, h)

    y[1:4] = y\_rk4[1:4]

    z[1:4] = z\_rk4[1:4]

    for i in range(3, n - 1):

        y[i+1] = y[i] + h \* (55 \* f(x[i], y[i], z[i])[0] - 59 \* f(x[i-1], y[i-1], z[i-1])[0] + 37 \* f(x[i-2], y[i-2], z[i-2])[0] - 9 \* f(x[i-3], y[i-3], z[i-3])[0]) / 24

        z[i+1] = z[i] + h \* (55 \* f(x[i], y[i], z[i])[1] - 59 \* f(x[i-1], y[i-1], z[i-1])[1] + 37 \* f(x[i-2], y[i-2], z[i-2])[1] - 9 \* f(x[i-3], y[i-3], z[i-3])[1]) / 24

    return x, y, z

def runge\_romberg(h1, y1, z1, h2, y2, z2, p):

    return y1 + (y1 - y2) / ((h2 / h1) \*\* p - 1), z1 + (z1 - z2) / ((h2 / h1) \*\* p - 1)

#Отрезок\_нач.условия

x0=0

xn=1

y0=1

z0=0

h=0.1

y\_tr = lambda x: x\*np.sin(x) + np.cos(x)

def ode\_sys(x, y, z):

    dy = z

    dz = 2\*np.cos(x) - y

    return dy, dz

# Решение с помощью метода Эйлера

x\_euler, y\_euler, z\_euler = euler\_method(ode\_sys, x0, y0, z0, xn, h)

# Решение с помощью метода Рунге-Кутты

x\_rk4, y\_rk4, z\_rk4 = runge\_kutta\_method(ode\_sys, x0, y0, z0, xn, h)

# Решение с помощью метода Адамса 4-го порядка

x\_adams, y\_adams, z\_adams = adams\_bashforth\_method(ode\_sys, x0, y0, z0, xn, h)

print("Решение методом Эйлера:", y\_euler[-1], z\_euler[-1])

print("Решение методом Рунге-Кутты", y\_rk4[-1], z\_rk4[-1])

print("Решение методом Адамса", y\_adams[-1], z\_adams[-1])

def ode\_sys2(x, y):

    dydx = [y[1], 2 \* np.cos(x) - y[0]]

    return dydx

y\_0 = [0, 1]

# Решение системы ОДУ методом Рунге-Кутты 4-го порядка

sol\_rk4 = solve\_ivp(ode\_sys2, [x0, xn], y\_0, method='RK45', t\_eval=np.arange(x0, xn+h, h))

# Решение системы ОДУ методом Адамса-Бэшфорта

sol\_adams = solve\_ivp(ode\_sys2, [x0, xn], y\_0, method='BDF', t\_eval=np.arange(x0, xn + h, h))

x\_rk4\_2 = sol\_rk4.t

y\_rk4\_2 = sol\_rk4.y[0]

z\_rk4\_2 = sol\_rk4.y[1]

x\_adams\_2 = sol\_adams.t

y\_adams\_2 = sol\_adams.y[0]

z\_adams\_2 = sol\_adams.y[1]

print("Решение методом Рунге-Кутты 4-го порядка:", y\_rk4\_2[-1], z\_rk4\_2[-1])

print("Решение методом Адамса-Бэшфорта:", y\_adams\_2[-1], z\_adams\_2[-1])

x\_tr = np.linspace(x0, xn, 1000)

plt.title("Решение ОДУ")

plt.xlabel("x")

plt.ylabel("y")

plt.grid(True)

plt.plot(x\_tr, y\_tr(x\_tr), label="Точное решение")

plt.plot(x\_euler, y\_euler, label="Метод Эйлера")

plt.plot(x\_rk4, y\_rk4, label="Метод Рунге-Кутты 4 порядка")

plt.plot(x\_adams, y\_adams, label="Метод Адамса")

plt.legend()

plt.show()

h2 = h/2

#Метод Эйлера

\_, y\_euler2, z\_euler2 = euler\_method(ode\_sys, x0, y0, z0, xn, h2)

y\_euler\_romberg, z\_euler\_romberg = runge\_romberg(h, y\_euler[-1], z\_euler[-1], h2, y\_euler2[-1], z\_euler2[-1], 1)

y\_etalon\_euler = y\_tr(xn)

error\_euler\_romberg = np.abs(y\_euler\_romberg - y\_euler[-1])

error\_euler\_etalon = np.abs(y\_etalon\_euler - y\_euler[-1])

print("Решение методом Эйлера:", y\_euler[-1])

print("Погрешность по Рунге-Ромбергу::", error\_euler\_romberg)

print("Погрешность с точным решением:", error\_euler\_etalon)

#Метод Рунге-Кутты

\_, y\_rk42, z\_rk42 = runge\_kutta\_method(ode\_sys, x0, y0, z0, xn, h2)

y\_rk4\_romberg, z\_rk4\_romberg = runge\_romberg(h, y\_rk4[-1], z\_rk4[-1], h2, y\_rk42[-1], z\_rk42[-1], 4)

y\_etalon\_rk4 = y\_tr(xn)

error\_rk4\_romberg = np.abs(y\_rk4\_romberg - y\_rk4[-1])

error\_rk4\_etalon = np.abs(y\_etalon\_rk4 - y\_rk4[-1])

print("\nРешение методом Рунге-Кутты:", y\_rk4[-1])

print("Погрешность по Рунге-Ромбергу:", "{:.14f}".format(error\_rk4\_romberg))

print("Погрешность с точным решением:", error\_rk4\_etalon)

#Метод Адамса

\_, y\_adams2, z\_adams2 = adams\_bashforth\_method(ode\_sys, x0, y0, z0, xn, h2)

y\_adams\_romberg, z\_adams\_romberg = runge\_romberg(h, y\_adams[-1], z\_adams[-1], h2, y\_adams2[-1], z\_adams2[-1], 4)

y\_etalon\_adams = y\_tr(xn)

error\_adams\_romberg = np.abs(y\_adams\_romberg - y\_adams[-1])

error\_adams\_etalon = np.abs(y\_etalon\_adams - y\_adams[-1])

print("\nРешение методом Адамса:", y\_adams[-1])

print("Погрешность по Рунге-Ромбергу:", error\_adams\_romberg)

print("Погрешность с точным решением:", error\_adams\_etalon)

**Вывод программы:**

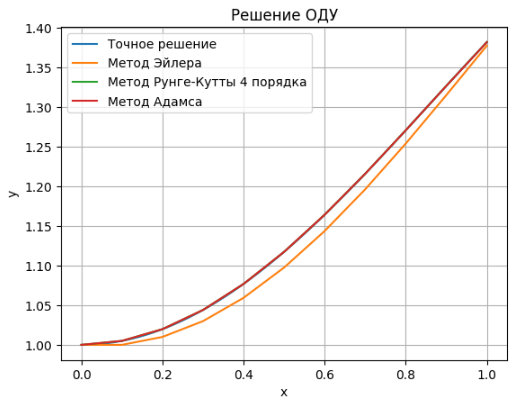
Решение методом Эйлера: 1.3772833758693874 0.6169954066440165

Решение методом Рунге-Кутты 1.3817721293338137 0.5403025917134935

Решение методом Адамса 1.3816848544271656 0.5402273372942328

Решение методом Рунге-Кутты 4-го порядка: 1.6830031893695985 1.9220480242238187

Решение методом Адамса-Бэшфорта: 1.6835192946626056 1.9225428714352115



Решение методом Эйлера: 1.3772833758693874

Погрешность по Рунге-Ромбергу:: 0.007528920079502122

Погрешность с точным решением: 0.004489914806648843

Решение методом Рунге-Кутты: 1.3817721293338137

Погрешность по Рунге-Ромбергу: 0.00000116030865

Погрешность с точным решением: 1.1613422226108838e-06

Решение методом Адамса: 1.3816848544271656

Погрешность по Рунге-Ромбергу: 8.67173611718286e-05

Погрешность с точным решением: 8.843624887067136e-05

**Вывод:**

В процессе лабораторной работы мной были изучены методы решения задачи Коши для ОДУ второго порядка на указанном отрезке, такие как метод Эйлера, метод Рунге-Кутты и метод Адамса 4-го порядка. Было проведено сравнение этих методов на примере нескольких тестовых задач. В результате сравнения было выявлено, что методы Рунге-Кутты и Адамса 4-го порядка более точны и устойчивы при решении задачи Коши для ОДУ второго порядка, чем метод Эйлера. Однако метод Адамса 4-го порядка требует вычисления нескольких начальных значений, что может быть затруднительно в некоторых случаях.

**Задание 4.2**

Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

|  |  |
| --- | --- |
| xy″-(2x+1)y′ +(x+1)y=0,  y′(0)=1,  y′ (1)-2y(1)=0 |  |

**Теоретические сведения**

Примером краевой задачи является двухточечная краевая задача для обыкновенного дифференциального уравнения второго порядка.

с граничными условиями, заданными на концах отрезка *[𝑎, 𝑏].*

Следует найти такое решение на этом отрезке, которое принимает на концах отрезка значения .

Кроме граничных условий первого рода, используются еще условия на производные от решения на концах - граничные условия второго рода:

или линейная комбинация решений и производных – граничные условия третьего рода:

Возможно на разных концах отрезка использовать условия различных типов.

**Метод стрельбы:**

Суть метода заключена в многократном решении задачи Коши для приближенного нахождения решения краевой задачи.

Пусть надо решить краевую задачу краевыми условиями 1-го рода на отрезке . Вместо исходной задачи формулируется задача Коши с уравнением и с начальными условиями

Положим сначала некоторое начальное значение параметру , после чего решим каким-либо методом задачу Коши. Пусть решение этой задачи на интервале , тогда сравнивая значение функции со значением в правом конце отрезка можно получить информацию для корректировки угла наклона касательной к решению в левом конце отрезка. Задачу можно сформулировать таким образом: требуется найти такое значение переменной , чтобы решение в правом конце отрезка совпало со значением . Другими словами, решение исходной задачи эквивалентно нахождению корня уравнения

Следующее значение искомого корня определяется по соотношению

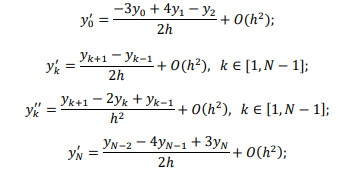


**Конечно-разностный метод:**

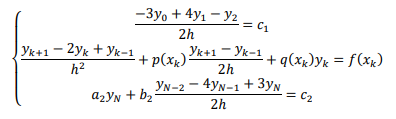
Рассмотрим двухточечную краевую задачу для линейного дифференциального уравнения второго порядка на отрезке



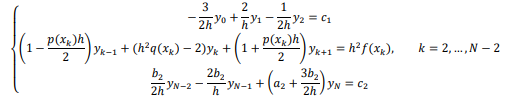
Введем разностную аппроксимацию производных следующим образом:



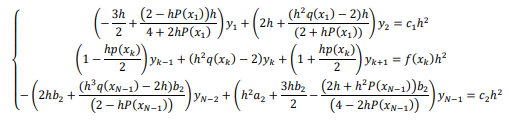
Подставляя аппроксимации производных, получим систему уравнений для нахождения :



Приводя подобные, получаем следующую систему

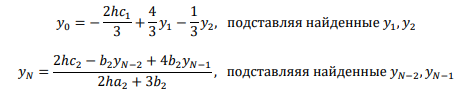


Данная система не является трехдиагональной, поэтому преобразуем систему, убрав . Получим новую систему:



Теперь система может быть решена методом прогонки. Получаем значения

Теперь найдем и из формул:



Оценить погрешность решения можно, сравнивая с точным решением, если оно есть, или с помощью метода Рунге-Ромберга. Он позволяет получать более высокий порядок точности вычисления. Если имеются результаты вычисления определенного интеграла на сетке с шагом ℎ − и на сетке с

шагом 𝑘ℎ − , то

**Код программы:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import solve\_ivp, solve\_bvp, odeint

from scipy.optimize import fsolve, newton, root\_scalar,root, minimize

from scipy.interpolate import interp1d

def shooting\_method(a,b,z0,eps=1e-6):

    x\_span = [a, b]

    x\_eval = np.linspace(x\_span[0], x\_span[1], 100)

    def solve\_and\_evaluate(y1\_guess):

        y\_init\_guess = [y1\_guess, z0]

        sol = solve\_ivp(ode, x\_span, y\_init\_guess, method='RK45', t\_eval=x\_eval,atol=eps)

        return boundary\_conditions(sol.y[:, 0], sol.y[:, -1])[1]

    y1\_guess = 0.0

    y1\_guess\_next = 1.0

    tolerance = eps

    max\_iterations = 1000

    for \_ in range(max\_iterations):

        f\_y1 = solve\_and\_evaluate(y1\_guess)

        f\_y1\_next = solve\_and\_evaluate(y1\_guess\_next)

        if abs(f\_y1\_next) < tolerance:

            break

        y1\_guess, y1\_guess\_next = y1\_guess\_next, y1\_guess\_next - f\_y1\_next \* (y1\_guess\_next - y1\_guess) / (f\_y1\_next - f\_y1)

    y\_init\_guess = [y1\_guess\_next, z0]

    solution = solve\_ivp(ode, x\_span, y\_init\_guess, method='RK45', t\_eval=x\_eval, atol=eps)

    return solution, y1\_guess\_next

def finite\_difference\_method(a, b,y0,y1, N):

    h = (b - a) / (N)

    x = np.linspace(a, b, N+1)

    y = np.zeros(N+1)

    y\_prime = np.zeros(N+1)

    y[0] = y0

    y\_prime[0] = y1

    for i in range(1, N+1):

        y\_prime[i] = y\_prime[i-1] + h \* f(x[i-1], y[i-1], y\_prime[i-1])

        y[i] = y[i-1] + h \* y\_prime[i-1]

    return x, y, y\_prime

def runge\_romberg\_error(h1, y1, h2, y2, p):

    return (y1 - y2) / ((h2 / h1)\*\*p - 1)

def ode(x, y):

    return [y[1], (-(x + 1) \* y[0] + (2 \* x + 1) \* y[1]) / x]

def boundary\_conditions(ya, yb):

    return [ya[1]-1, -2\* yb[0] + yb[1]]

def f(x, y, y\_p):

    return (-(x+1)\*y + (2\*x + 1)\*y\_p) / x

a,b=0.01,1

z0=1

point=0.8

solution, y = shooting\_method(a,b,z0)

N=100

y\_tr = lambda x: np.exp(x)\*(x\*\*2+1)

#Решение методом стрельбы

res\_sh=np.interp(point, solution.t, solution.y[0])

print(f"Численное решение в точке x={point}: {res\_sh}")

#Оценка погрешности

h1=0.01

h2=h1/2

solution2,\_ = shooting\_method(a,b,z0,eps=1e-3)

res\_sh2=np.interp(point, solution2.t, solution2.y[0])

print(f"Погрешность вычислений методом Рунге-Ромберга в точке: ",'{0:.{1}f}'.format(runge\_romberg\_error(h1,res\_sh,h2,res\_sh2,p=4),18))

print(f"Погрешность вычислений путем сравнения с точным решением: ",'{0:.{1}f}'.format(np.abs(res\_sh-y\_tr(point)),18))

#Решение конечно-разностным методом

y0 = y

x\_fd,y\_fd,\_=finite\_difference\_method(a,b,y0,z0,N)

res\_fd=np.interp(point, x\_fd, y\_fd)

print(f"Численное решение в точке x={point}: {res\_fd}")

#Оценка погрешности

N2=50

x\_fd2,y\_fd2,\_ = finite\_difference\_method(a,b,y0,z0,N2)

res\_fd2=np.interp(point, x\_fd2, y\_fd2)

print(f"Погрешность вычислений методом Рунге-Ромберга в точке x={point}:",'{0:.{1}f}'.format(runge\_romberg\_error(h1,res\_fd,h2,res\_fd2,p=4),18))

print(f"Погрешность вычислений путем сравнения с точным решением x={point}:",'{0:.{1}f}'.format(np.abs(res\_fd-y\_tr(point)),18))

t = np.linspace(a,b, 100)

plt.grid(True)

plt.xlabel("x")

plt.ylabel("y")

plt.plot(t, y\_tr(t), c="r", label="Эталонное решение")

plt.plot(solution.t, solution.y[0], label="Метод стрельбы")

plt.plot(x\_fd, y\_fd, label="Метод конечных разностей")

plt.legend()

plt.show()

x = np.linspace(a,b, N)

y\_b = np.zeros((2, x.size))

res = solve\_bvp(ode, boundary\_conditions, x, y\_b, tol=1e-6)

print(f"Проверка:");

print(f"Численное решение в точке x={point}:",res.sol(point)[0])

print(f"Погрешность вычислений путем сравнения с точным решением x={point}:",'{0:.{1}f}'.format(np.abs(res.sol(point)[0]-y\_tr(point)),18))

**Вывод программы:**

Численное решение в точке x=0.8: 3.5422446581491727

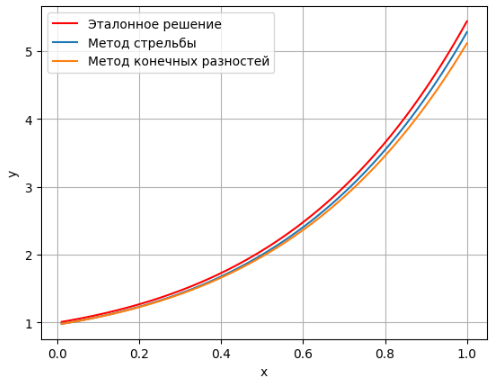
Погрешность вычислений методом Рунге-Ромберга в точке: 0.000019972372454428

Погрешность вычислений путем сравнения с точным решением: 0.107642464578475039

Численное решение в точке x=0.8: 3.4557645787464497

Погрешность вычислений методом Рунге-Ромберга в точке x=0.8: -0.081338757619113744

Погрешность вычислений путем сравнения с точным решением x=0.8: 0.194122543981197992

****

Проверка:

Численное решение в точке x=0.8: 3.5423685303812458

Погрешность вычислений путем сравнения с точным решением x=0.8: 0.107518592346401931

**Вывод:**

В ходе выполнения лабораторной работы были рассмотрены два численных метода решения краевой задачи для обыкновенных дифференциальных уравнений: метод стрельбы и конечно-разностный метод. Метод стрельбы заключается в поиске значения начальной производной, при котором решение уравнения удовлетворяет граничным условиям. Конечно-разностный метод основывается на аппроксимации производных разностными операторами и последующем решении системы линейных алгебраических уравнений, составленной на основе уравнения и граничных условий. Оба метода имеют свои преимущества и недостатки и выбор метода зависит от поставленной задачи и требуемой точности. В ходе работы были решены несколько примеров краевых задач, используя оба метода, что позволило сравнить их эффективность.