Министерство образования и науки РФ

Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

факультет программной инженерии и компьютерной техники

**ЛАБОРАТОРНАЯ РАБОТА №6**

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №25

*Студент:*

Хоанг Ван Куан

Группа Р3266

*Преподаватель:*

Машина Екатерина Александровна

Санкт-Петербург, 2024

1. **Цель работы**

Цель лабораторной работы: решить задачу Коши для обыкновенных дифференциальных уравнений численными методами.

1. **Порядок выполнения работы**

Программная реализация

* В программе численные методы решения обыкновенных дифференциальных уравнений (ОДУ) должен быть реализован в виде отдельного класса /метода/функции
* Пользователь выбирает ОДУ вида (не менее трех уравнений), из тех, которые предлагает программа
* Предусмотреть ввод исходных данных с клавиатуры: начальные условия , интервал дифференцирования , шаг *h*, точность
* Для исследования использовать одношаговые методы и многошаговые методы (см. табл.1)
* Составить таблицу приближенных значений интеграла дифференциального уравнения, удовлетворяющего начальным условиям, для всех методов, реализуемых в программе
* Для оценки точности одношаговых методов использовать правило Рунге
* Для оценки точности многошаговых методов использовать точное решение задачи: | |
* Построить графики точного решения и полученного приближенного решения (разными цветами)
* Программа должна быть протестирована при различных наборах данных, в том числе и некорректных.
* Проанализировать результаты работы программы.

1. **Листинг программы**

import math

# порядок точности: 1

def EulerMethod(f, a, b, y0, h, epsilon):

    check = True

    while(check):

        res1 = Euler(f, a, b, y0, h)

        res2 = Euler(f, a, b, y0, h/2)

        for i in range(len(res1) - 1, -1, -1):

            if(abs(res1[i][1] - res2[i \* 2][1]) >= epsilon):

                h /= 4

                break

            else: check = False

    return Euler(f, a, b, y0, h)

def Euler(f, a, b, y0, h):

    res = [(a, y0)]

    n = int((b - a) / h)

    for i in range(1, n + 1):

        res.append((res[i - 1][0] + h, res[i - 1][1] + h \* f(res[i - 1][0], res[i - 1][1])))

    return res

# порядок точности: 4

def RungeKuttaMethod(f, a, b, y0, h, epsilon):

    check = True

    while(check):

        res1 = RungeKutta(f, a, b, y0, h)

        res2 = RungeKutta(f, a, b, y0, h/2)

        for i in range(len(res1) - 1, -1, -1):

            if(abs(res1[i][1] - res2[i \* 2][1])/15 > epsilon):

                h /= 4

                break

            else: check = False

    return RungeKutta(f, a, b, y0, h)

def RungeKutta(f, a, b, y0, h):

    res = [(a, y0)]

    n = int((b - a) / h)

    for i in range(1, n + 1):

        k1 = h \* f(res[i - 1][0], res[i-1][1])

        k2 = h \* f(res[i - 1][0] + h/2, res[i-1][1] + k1/2)

        k3 = h \* f(res[i - 1][0] + h/2, res[i-1][1] + k2/2)

        k4 = h \* f(res[i - 1][0] + h, res[i-1][1] + k3)

        res.append((res[i - 1][0] + h, res[i - 1][1] + (k1 + 2\*k2 + 2\*k3 + k4)/6))

    return res

# порядок точности: 4

def MilnaMethod(f, a, b, y0, h, epsilon):

    n = int((b - a) / h)

    b0 = min(b, a + 3.1\*h) # 0.1 для погрешности в Python

    res = RungeKuttaMethod(f, a, b0, y0, h, epsilon)

    for i in range(4, n + 1):

        xi = res[i - 1][0] + h

        \_Yprog = res[i - 4][1] + 4\*h\*(2\*f(res[i - 3][0], res[i - 3][1]) - f(res[i - 2][0], res[i - 2][1]) + 2\*f(res[i - 1][0], res[i - 1][1])) / 3

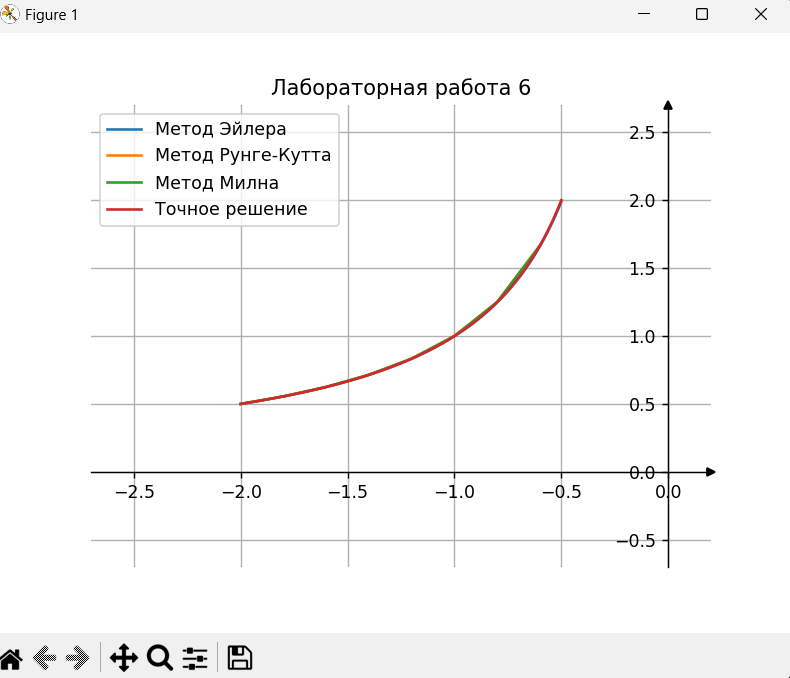
        \_Fprog = f(xi, \_Yprog)

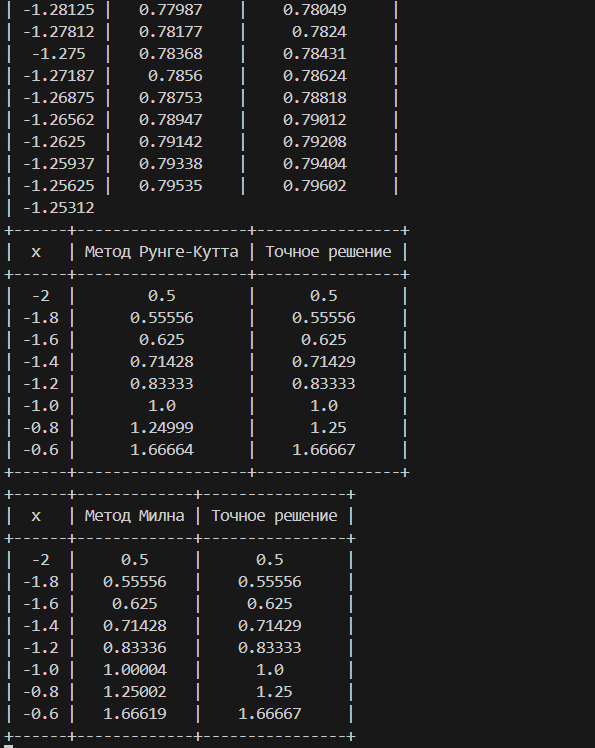
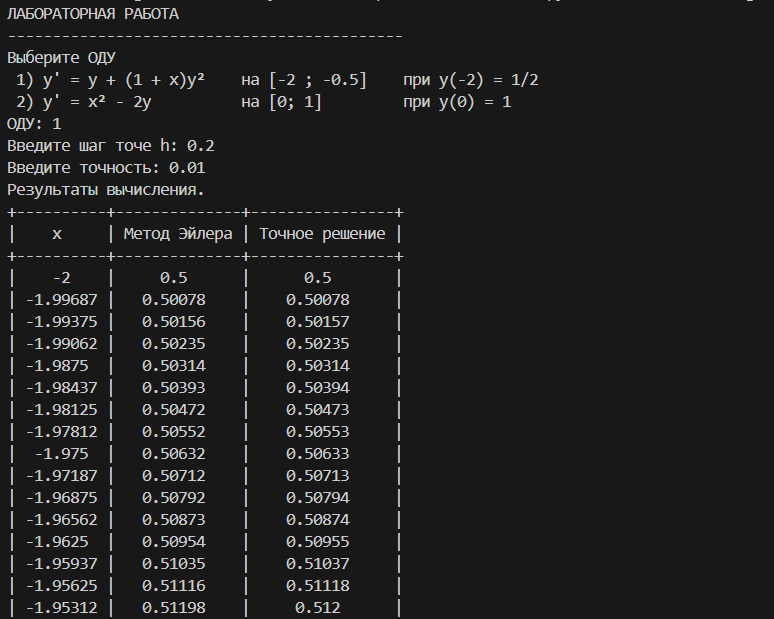
        \_Ykorr = res[i - 2][1] + h\*(f(res[i - 2][0], res[i - 2][1]) + 4\*f(res[i - 1][0], res[i - 1][1]) + \_Fprog) / 3

        res.append((xi, \_Ykorr))

    return res

1. **Результаты выполнения программы**

****

****