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 №6

*Студент:*

Карандашева Анастасия Денисовна

Группа Р3268

*Преподаватель:*

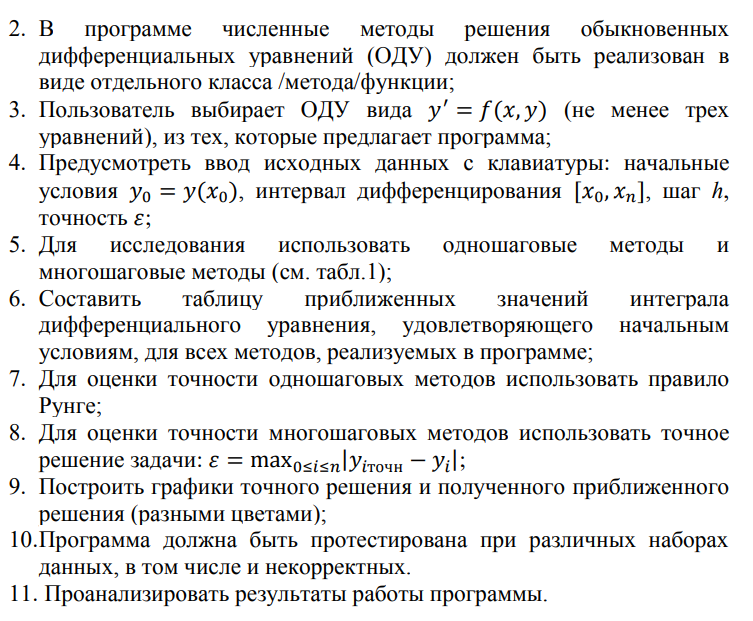
Машина Екатерина Александровна

Санкт-Петербург, 2024

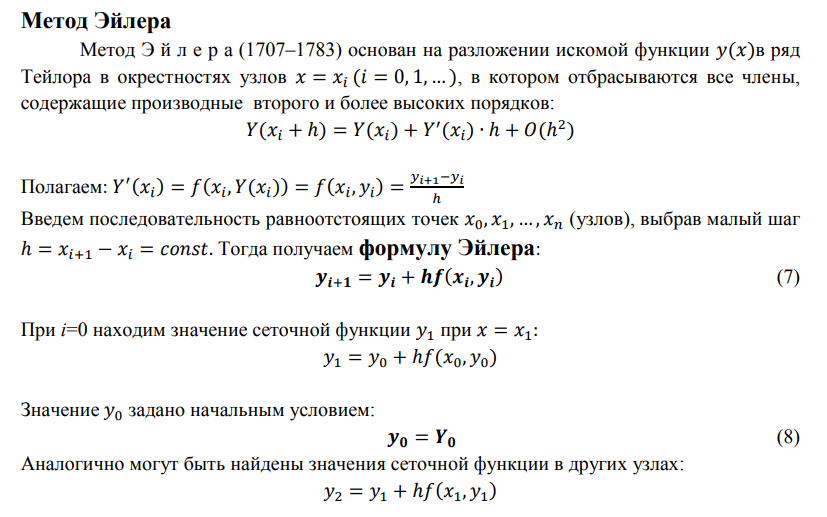
1. **Цель работы**

Решить задачу Коши для обыкновенных дифференциальных уравнений численными методами.

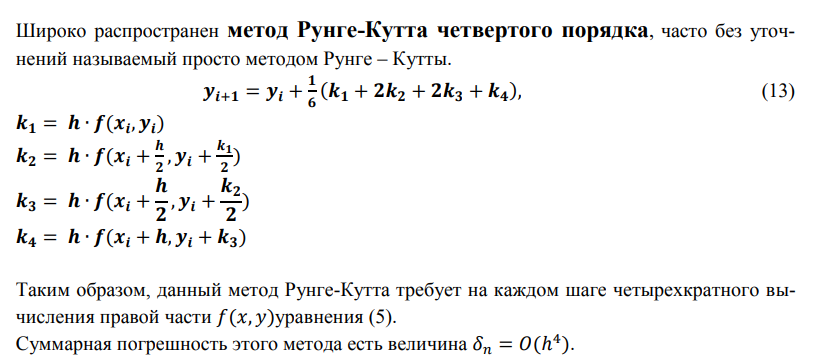
1. **Порядок выполнения работы**

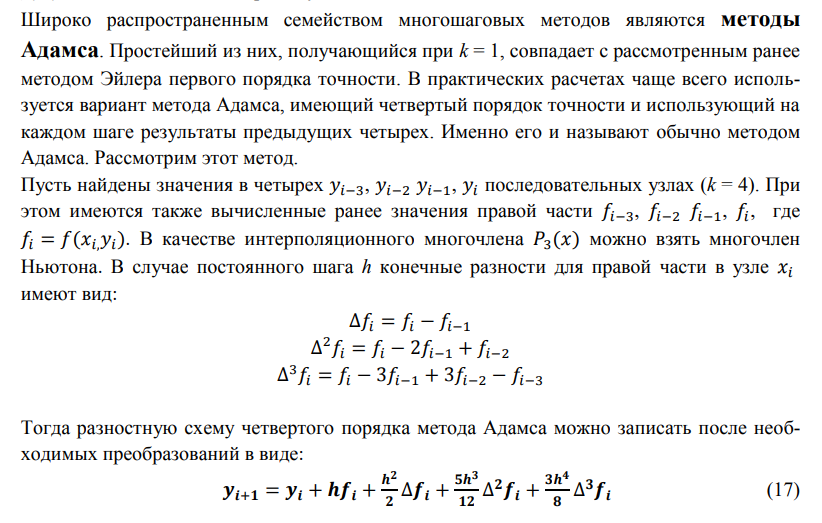
****

1. **Рабочие формулы**

****

****

****

****

1. **Листинг программы**

import numpy as np

import math

import matplotlib.pyplot as plt

from tabulate import tabulate

from scipy.integrate import odeint

def adams(f, x0, y0, interval, n):

    h = (interval[1] - interval[0])/n

    x\_values = np.arange(interval[0], interval[1] + 3\*h, h)

    y\_values = [y0]

    for i in range(1, 4):  # первые шаги выполним методом Рунге-Кутта 4-го порядка

        k1 = h \* f(x\_values[i-1], y\_values[i-1])

        k2 = h \* f(x\_values[i-1] + h/2, y\_values[i-1] + k1/2)

        k3 = h \* f(x\_values[i-1] + h/2, y\_values[i-1] + k2/2)

        k4 = h \* f(x\_values[i-1] + h, y\_values[i-1] + k3)

        y\_next = y\_values[i-1] + (k1 + 2\*k2 + 2\*k3 + k4) / 6

        y\_values.append(y\_next)

    for i in range(3, len(x\_values)-1):

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

        y\_values.append(y\_next)

    return x\_values, y\_values

def euler(f, x0, y0, interval, n):

    h = (interval[1] - interval[0])/n

    x\_values = np.arange(interval[0], interval[1] + h, h)

    y\_values = [y0]

    for i in range(1, len(x\_values)):

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

        y\_values.append(y\_next)

    return x\_values, y\_values

def runge\_kutta(f, x0, y0, interval, n):

    h = (interval[1] - interval[0])/n

    x\_values = np.arange(interval[0], interval[1] + h, h)

    y\_values = [y0]

    for i in range(1, len(x\_values)):

        k1 = h \* f(x\_values[i-1], y\_values[-1])

        k2 = h \* f(x\_values[i-1] + h/2, y\_values[-1] + k1/2)

        k3 = h \* f(x\_values[i-1] + h/2, y\_values[-1] + k2/2)

        k4 = h \* f(x\_values[i-1] + h, y\_values[-1] + k3)

        y\_next = y\_values[-1] + (k1 + 2\*k2 + 2\*k3 + k4) / 6

        y\_values.append(y\_next)

    return x\_values, y\_values

def f1(x, y):

    return x\*\*2 + y

def f2(x, y):

    return y + (1+x)\*y\*\*2

def f3(x, y):

    return x\*\*2 + y

def exact\_solution1(x):

    return np.exp((x\*\*2)/2)

def exact\_solution2(x):

    return (-np.exp(x))/(x\*(np.exp(x)))

def exact\_solution3(x):

    return np.exp((x\*\*2)/2)

# Правило рунге для проверки точности

def runge\_rule(method, f, x0, y0, accuracy, k, interval, h):

    n = abs((interval[-1] - interval[0]))/h

    prev\_x, prev\_y = method(f, x0, y0, interval, n)

    n \*= 2

    x, y = method(f, x0, y0, interval, n)

    while(abs(y[-1] - prev\_y[-1])/(2^k-1)) >= accuracy:

        prev\_y = y

        n \*= 2

        x, y = method(f, x0, y0, interval, n)

    return x, y

# Проверка точности решения

def check\_accuracy(method, f, df, x0, y0, accuracy, interval, h):

    n = abs((interval[-1] - interval[0]))/h

    x, y = method(f, x0, y0, interval, n)

    y\_exact = df(x[-1])

    while abs(y[-1] - y\_exact) < accuracy:

        n \*= 2

        x, y = method(f, x0, y0, interval, n)

    return x, y

#Функции и методы

functions = {1: f1, 2: f2, 3: f3}

methods = {

    1: euler,

    2: runge\_kutta,

    3: adams

}

dfunctions = {1: exact\_solution1, 2: exact\_solution2, 3: exact\_solution3}

def main():

    print("Выберите дифференциальное уравнение для решения задачи Коши: ")

    print("1. x\*\*2 + y")

    print("2. y + (1+x)\*y\*\*2")

    print("3. e^x")

    choice = int(input())

    if choice not in functions:

        raise ValueError("Введён неверный номер функции")

    f = functions[choice]

    df = dfunctions[choice]

    print("")

    print("Введите начальные условия ")

    x0 = float(input("x0:"))

    y0 = float(input("y0:"))

    print("")

    a = float(input("Введите левую границу интервала: "))

    b = float(input("Введите правую границу интервала: "))

    interval = [a, b]

    print("")

    h = float(input("Задайте шаг: "))

    print("")

    accuracy = float(input("Задай точность вычисления: "))

    print("")

    print("Выбери метод решения:")

    print("1. Метод Эйлера")

    print("2. Метод Рунге-Кутта 4-го порядка")

    print("3. Метод Адамса")

    method\_choice = int(input())

    if method\_choice == 1:

      k = 1

      x, y = runge\_rule(methods[method\_choice], f, x0, y0, accuracy, k, interval, h)

    elif (method\_choice == 2):

      k = 4

      x, y = runge\_rule(methods[method\_choice], f, x0, y0, accuracy, k, interval, h)

    elif (method\_choice == 3):

      x, y = check\_accuracy(methods[method\_choice], f, df, x0, y0, accuracy, interval, h)

    else:

        raise ValueError("Введён неверный номер метода")

    col\_headers = ["x", "y"]

    merged\_array = np.array([x, y]).T

    table = tabulate(merged\_array , col\_headers, tablefmt="fancy\_grid", floatfmt = ".2f")

    print(table)

    a = []

    plt.plot(x, y, label='Numerical Method', color='blue',

         marker='o', markerfacecolor='blue', markersize=5)

    plt.legend()

    plt.xlabel('x')

    plt.ylabel('y')

    x\_values = np.linspace(0, 10, 100)

    plt.plot(x, df(x), label='Exact', color='r', linestyle='-')

    plt.show()

try:

    main()

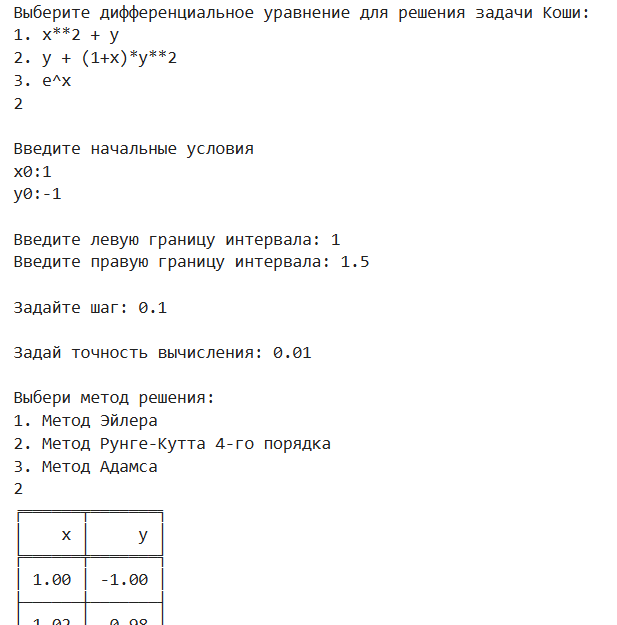
# except ValueError as e:

#     print("Ошибка: ", e)

except KeyboardInterrupt as e:

    print(e)

1. **Результаты выполнения программы**



1. **Выводы**

В ходе выполнения лабораторной работы были изучены численные методы решения задачи Коши для обыкновенных дифференциальных уравнений первого порядка. Реализованы методы Эйлера, Рунге-Рутта 4 порядка и Адамса на языке python для поиска решения с заданной точностью, начальными условиями, шагом и интервалом.