Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа №3

«Классический метод Рунге-Кутта»

по курсу «Численные методы»

Студент: Шемякин В.А.

Группа: ИУ9-62Б

Преподаватель: Домрачева А.Б.

Москва 2025

1 Цель

Целью данной работы является реализация метода Рунге-Кутта для решения задачи Коши для дифференциального уравнения второго порядка.

2 Постановка задачи

Дано: уравнение

$$y'' = e^{4x} + 8y' + 8y$$

Задание:

1. Найти численно с погрешностью
2. Найти с точностью 0.001 все корни уравнения $f(x) = 0$ методом деления отрезка пополам и методом Ньютона; определить число приближений в каждом случае
3. Сравнить полученные результаты

Индивидуальный вариант(№ 5): $y(0) = 0, y'(0) = 1$

3 Основные теоретические сведения

Классический метод Рунге-Кутта 4-го порядка

Классический метод Рунге-Кутта 4-го порядка заключается в последовательном вычислении коэффициентов K_1, K_2, K_3, K_4 по формулам:

$$\begin{aligned}
K_1 &= f(x, y) \\
K_2 &= f\left(x + \frac{h}{2}, y + \frac{h}{2}K_1\right) \\
K_3 &= f\left(x + \frac{h}{2}, y + \frac{h}{2}K_2\right) \\
K_4 &= f(x + h, y + hK_3)
\end{aligned}$$

и построении приближения к решению в точке $x + h$:

$$y(x+h) \approx y_h = y + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

Автоматический выбор шага

Для автоматического управления длиной шага используется правило Рунге:

1. Выполняют вычисления с шагом h (два шага) и с шагом $2h$ (один шаг)
2. Оценивают погрешность:

$$\epsilon^* = \frac{1}{2^p - 1}\|\mathbf{Y}_h - \mathbf{Y}_{2h}\|$$

   где $p = 4$ - порядок метода

3. Если $\epsilon^* < \epsilon$ (заданная точность), шаг принимается, иначе уменьшается

4 Реализация

Listing 1: main.py

```python
import math
from typing import Callable, List, Tuple


def differential_equation(x: float, Y: List[float]) -> List[float]:
    return [Y[1], math.exp(4 * x) + 8 * Y[1] + 8 * Y[0]]


def runge_kutta_step(
    func: Callable[[float, List[float]], List[float]],
    x: float,
    Y: List[float],
    step_size: float
) -> List[float]:
    k1 = [step_size * val for val in func(x, Y)]
    Y1 = [Y[i] + 0.5 * k1[i] for i in range(len(Y))]

    k2 = [step_size * val for val in func(x + 0.5 * step_size, Y1)]
    Y2 = [Y[i] + 0.5 * k2[i] for i in range(len(Y))]

```

```
    k3 = [step_size * val for val in func(x + 0.5 * step_size, Y2)]
    Y3 = [Y[i] + k3[i] for i in range(len(Y))]

    k4 = [step_size * val for val in func(x + step_size, Y3)]

    Y_next = [
        Y[i] + (k1[i] + 2 * k2[i] + 2 * k3[i] + k4[i]) / 6
        for i in range(len(Y))
    ]
    return Y_next


def exact_solution(x: float) -> Tuple[float, float]:
    sqrt6 = math.sqrt(6)
    y = ((2*sqrt6 + 1)/48 * math.exp((4 + 2*sqrt6)*x) +
         (1 - 2*sqrt6)/48 * math.exp((4 - 2*sqrt6)*x) +
         (-1/24 * math.exp(4*x)))

    coef1 = (5*sqrt6 + 14)/24
    coef2 = (14 - 5*sqrt6)/24
    dy = (coef1 * math.exp((4 + 2*sqrt6)*x) +
          coef2 * math.exp((4 - 2*sqrt6)*x) +
          -1/6 * math.exp(4*x))
    return y, dy


def runge_kutta_solver(
    func: Callable[[float, List[float]], List[float]],
    x0: float,
    Y0: List[float],
    x_end: float,
    step_size: float
) -> List[Tuple[float, float, float, float, float, float]]:
    results = []
    x = x0
    Y = Y0.copy()

    exact_y, exact_y_prime = exact_solution(x)
    results.append((x, Y[0], Y[1], exact_y, exact_y_prime, 0.0))

    while x < x_end:
        if x + 2*step_size > x_end:
            return results

        Y_h_prev = runge_kutta_step(func, x, Y, step_size)
        Y_h = runge_kutta_step(func, x + step_size, Y_h_prev, step_size)
```

```
        Y_2h = runge_kutta_step(func, x, Y, 2*step_size)

        error = abs(Y_h[0] - Y_2h[0]) / (2 ** 4 - 1)
        x += 2*step_size
        Y = Y_h

        exact_y, exact_y_prime = exact_solution(x)
        results.append((x, Y[0], Y[1], exact_y, exact_y_prime, error))

    return results


def print_results(results: List[Tuple[float, float, float, float, float, float]]) -> None:
    headers = ("x", "Approx y", "Exact y", "Approx y'", "Exact y'", "Error")
    print("{:>8} {:>14} {:>14} {:>14} {:>14} {:>14}".format(*headers))

    for x_val, approx_y, approx_yprime, exact_y, exact_dy, err in results:
        print("{:8.4f} {:14.6f} {:14.6f} {:14.6f} {:14.6f} {:14.6f}".format(
            x_val, approx_y, exact_y, approx_yprime, exact_dy, err))


def main():
    initial_x = 0.0
    final_x = 1.0
    initial_conditions = [0.0, 1.0]
    step_size = 0.02

    solution = runge_kutta_solver(
        differential_equation,
        initial_x,
        initial_conditions,
        final_x,
        step_size
    )

    print_results(solution)


if __name__ == "__main__":
    main()
```

5 Результаты

x	Approx y	Exact y	Approx y'	Exact y'	Error
0.0000	0.000000	0.000000	1.000000	1.000000	0.000000
0.0400	0.048183	0.048183	1.436085	1.436089	0.000000
0.0800	0.117475	0.117476	2.067156	2.067168	0.000001
0.1200	0.217266	0.217269	2.977812	2.977836	0.000001
0.1600	0.361016	0.361021	4.289043	4.289088	0.000001
0.2000	0.568002	0.568011	6.173828	6.173909	0.000002
0.2400	0.865816	0.865832	8.879411	8.879551	0.000002
0.2800	1.293938	1.293964	12.759102	12.759334	0.000003
0.3200	1.908827	1.908870	18.317685	18.318065	0.000005
0.3600	2.791206	2.791275	26.276260	26.276871	0.000007
0.4000	4.056452	4.056561	37.664806	37.665776	0.000009
0.4400	5.869439	5.869610	53.954345	53.955870	0.000013
0.4800	8.465730	8.465997	77.245625	77.248002	0.000019
0.5200	12.181835	12.182249	110.538494	110.542173	0.000027
0.5600	17.498410	17.499046	158.116466	158.122127	0.000039
0.6000	25.101931	25.102904	226.095743	226.104408	0.000056
0.6400	35.972753	35.974236	323.208989	323.222193	0.000080
0.6800	51.510817	51.513069	461.924254	461.944293	0.000114
0.7200	73.715124	73.718530	660.042323	660.072629	0.000163
0.7600	105.439944	105.445078	942.977070	943.022760	0.000233
0.8000	150.760593	150.768312	1347.010820	1347.079510	0.000332
0.8400	215.495611	215.507186	1923.941605	1924.044611	0.000475
0.8800	307.952210	307.969527	2747.717420	2747.871529	0.000678
0.9200	439.990471	440.016326	3923.907022	3924.137107	0.000968
0.9600	628.542557	628.581084	5603.220058	5603.562912	0.001381

6 Вывод

В ходе выполнения лабораторной работы был изучен и реализован метод Рунге-Кутта четвертого порядка для решения задачи Коши для дифференциального уравнения второго порядка