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

по курсу «Численные методы»

Студент группы ИУ9-62Б Нащёкин Н.Д.

Преподаватель: Домрачева А.Б.

Москва, 2025

1 Задача

1. Найти численно с погрешностью $\varepsilon = 0.001$ решение задачи Коши дифференциального уравнения второго порядка

$$y'' + py' + qy = f(x), y(0) = y_0, y'(0) = y_0'$$

на отрезке [0, 1], приведя его к СОДУ первого порядка. Использовать классический метод Рунге-Кутта.
2. Найти точное решение дифференциального уравнения.
3. Сравнить приближённое и точное решения на каждом шаге вычислений.

2 Основная теория

Метод Рунге-Кутта

Метод Рунге-Кутта применяют для решения задачи Коши для системы обыкновенных дифференциальных уравнений (СОДУ) первого порядка:

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2, \ldots, y_n), \\ y_2' &= f_2(x, y_1, y_2, \ldots, y_n), \\ &\vdots \\ y_n' &= f_n(x, y_1, y_2, \ldots, y_n). \end{aligned}$$

на отрезке $[x_0, x_{\text{end}}]$ с начальными условиями

$$y_1(x_0) = y_{01}, \quad \ldots, \quad y_n(x_0) = y_{0n}.$$

Требуется приближённо найти решения системы

$$y_1(x_{\text{end}}), \quad \ldots, \quad y_n(x_{\text{end}})$$

в конечной точке отрезка. Запишем систему в векторной форме:

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \quad \mathbf{y}(x_0) = \mathbf{y}_0.$$

Здесь $\mathbf{y} = (y_1(x), \ldots, y_n(x))$ и $\mathbf{y}'$ — векторы неизвестных функций и их производных; $\mathbf{f} = (f_1(x, \mathbf{y}), \ldots, f_n(x, \mathbf{y}))$ — вектор правых частей; а $\mathbf{y}_0 = (y_{01}, \ldots, y_{0n})$ — вектор начальных условий.

Метод Рунге-Кутта позволяет последовательно, зная решение системы в некоторой точке x отрезка $[x_0, x_{\text{end}}]$, продвигаться на шаг h, то есть приближённо искать решение в точке $x + h$, затем в точке $x + 2h$ и так далее, пока не доберёмся до x_{end}.

Сам метод заключается в нахождении вектор-коэффициентов k_1, k_2, k_3 и k_4 по следующим формулам:

$$
\begin{aligned}
k_1 &= \mathbf{f}(x, \mathbf{y}),\\
k_2 &= \mathbf{f}(x + \frac{h}{2}, \mathbf{y} + \frac{hk_1}{2}),\\
k_3 &= \mathbf{f}(x + \frac{h}{2}, \mathbf{y} + \frac{hk_2}{2}),\\
k_4 &= \mathbf{f}(x + h, \mathbf{y} + hk_3).
\end{aligned}
$$

и построении очередного приближения к решению СОДУ в точке $x + h$ по формуле:

$$\mathbf{y}(x+h) \approx \mathbf{y}_h = \mathbf{y} + \frac{h}{6}(k_1 + 2(k_2 + k_3) + k_4).$$

Этот метод имеет четвёртый порядок точности, на каждом частичном отрезке:

$$||\mathbf{y}(x+h) - \mathbf{y}_h|| = \max_{1 \le i \le n} \left| y_i(x+h) - y_{h,i} \right| \le Ch^5$$

Поэтому при суммировании, так как число частичных отрезков равно $\frac{b-a}{h}$, получим $C(b-a)h^4$ в правой части неравенства.

Кроме того, существует алгоритм автоматического управления длиной шага h, обеспечивающий погрешность на каждом шаге не более ε:

1. Если начальная длина шага была выбрана равной h, то проводится вычисление векторов решений для шагов длины h и длины $\frac{h}{2}$.

2. Вычисляется погрешность по правилу Рунге практической оценки погрешности:

$$\mathrm{err} = \frac{\left\| \mathbf{y}_h - \mathbf{y}_{h/2} \right\|}{2^p - 1},$$

где p — порядок точности используемого метода (равен 4 в случае с методом Рунге-Кутта). Здесь для вектора $\mathbf{y}_{h/2}$ при вычислении нормы берётся каждая вторая координата.

3. Величина err сравнивается с ε, что позволяет вычислить оптимальную длину шага:

$$h_{\mathrm{opt}} = h \left(\frac{\varepsilon}{\mathrm{err}} \right)^{\frac{1}{p+1}}.$$

4. Если $\mathrm{err} \le \varepsilon$, то два вычисления $\mathbf{y}_h$ и $\mathbf{y}_{h/2}$ считаются принятыми и приближённым решением считается вектор $\mathbf{y}_{h/2}$. В противном случае оба шага отбрасываются и вычисления повторяются с длиной шага $h_{\mathrm{new}} = 0.9 h_{\mathrm{opt}}$.

3 Практическая реализация

Листинг 1 — реализация программы

```python
import math
from copy import deepcopy

import numpy as np

# y'' + p y' + q y = f(x)
# Вариант 22: p = -1, q = 0, f(x) = 3, y0 = 0, y0' = 2
# y'' - y' = 3
#
# Система: u' = v
#          v' = v + 3
# u(0) = 0, v(0) = 2
#
# Точное решение: u(1) = 5.59141 , v(1) = 10.59141
u_1_true = 5.59141
v_1_true = 10.59141

eps = 0.001


def u(t):
    return 5 * math.exp(t) - 3 * t - 5


def v(t):
    return 5 * math.exp(t) - 3


def f(x, y):
    u, v = y
    f1 = v        # u' = v
    f2 = v + 3    # v' = v + 3
    return np.array([f1, f2])


def runge_kutta_step(f, x, y, h):
    k1 = f(x, y)
    k2 = f(x + h / 2, y + h * k1 / 2)
    k3 = f(x + h / 2, y + h * k2 / 2)
    k4 = f(x + h, y + h * k3)
    return y + (h / 6) * (k1 + 2 * (k2 + k3) + k4)


def norm_one(a, b):
    max_val = 0
    for i in range(len(a)):
```

```
        cur_val = abs(a[i] - b[i])
        if cur_val > max_val:
            max_val = cur_val
    return max_val


def norm_full(a, b):
    max_val = 0
    for i in range(len(a)):
        try:
            cur_val = norm_one(a[i], b[2 * i])
        except IndexError:
            cur_val = 0
        if cur_val > max_val:
            max_val = cur_val
    return max_val


def runge_rule(y_h, y_2h, p):
    return norm_full(y_h, y_2h) / (2 ** p - 1)


def get_new_step(h, err, p):
    try:
        return 0.9 * h * (eps / err) ** (1 / (p + 1))
    except ZeroDivisionError:
        return 0.9 * h


def runge_solve(f, y0, x0, xend, h):
    x = x0
    y = y0
    y_vals = []
    x_vals = []
    y_real = []
    while x < xend:
        #print(h, x)
        if x + h - xend >= 0:
            y_vals.append(y)
            x_vals.append(x)
            y_real.append([u(x), v(x)])
            h = xend - x  # чтобы не перепрыгнуть через xend
            x = x + h
            y = runge_kutta_step(f, x, y, h)
            y_vals.append(y)
            x_vals.append(x)
            y_real.append([u(x), v(x)])
            break
        y_vals.append(y)
        x_vals.append(x)
```

```
        y_real.append([u(x), v(x)])
        x = x + h
        y = runge_kutta_step(f, x, y, h)
    return np.array(y_vals), np.array(y_real), np.array(x_vals)


def runge_kutta_with_autostep(f, y0, x0, xend, h):
    # Порядок метода
    p = 4

    y_h, _, _ = runge_solve(f, y0, x0, xend, h)
    y_h_2, y_r, xs = runge_solve(f, y0, x0, xend, h / 2)
    err = runge_rule(y_h, y_h_2, p)

    while err > eps:
        h = get_new_step(h , err, p)
        # print(h, err)
        y_h, _, _ = runge_solve(f, y0, x0, xend, h)
        y_h_2, y_r, xs = runge_solve(f, y0, x0, xend, h / 2)
        err = runge_rule(y_h, y_h_2, p)

    return np.array(y_h_2), np.array(y_r), np.array(xs)


def main():
    x0 = 0
    xend = 1

    y0 = np.array([0.0, 2.0]) # Н.у. u(0)=0, v(0)=2
    h = (xend - x0) / 2.0

    y, y_real, x = runge_kutta_with_autostep(f, y0, x0, xend, h)

    for i in range(len(y)):
        print(f'x = {x[i]:.8f}, [u(x), v(x)] приближённо = [{y[i][0]:.8f}, {y[i][1]:.8f}], '
            f'точное значение [u(x), v(x)] = [{y_real[i][0]:.8f}, {y_real[i][1]:.8f}], '
            f'абсолютная погрешность: {norm_one(y_real[i], y[i]):.8f}')
    print()

    print(f'eps = {eps}')
    print(f"Точные значения: u(1) = {u(xend):.8f}, v(1) = {v(xend):.8f}")
    print(f"Вычислено методом Рунге-Кутта: u(1) = {y[-1][0]:.8f}, v(1) = {y[-1][1]:.8f}")


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

Мой вариант – 22. По условию необходимо найти приближённое решение дифференциального уравнения $y'' - y' = 3$ с начальными условиями $y_0 = 0$,

$y'_0 = 2$. Для начала необходимо привести уравнение второго порядка к системе уравнений первого порядка. Вводим замены: $u = y, v = y'$. Исходя из замен, получим систему:

$$u' = v$$

$$v' = v + 3$$

и начальные условия: $u(0) = 0, v(0) = 2$. Эта система приближённо решается с использованием описанного метода Рунге-Кутта.

Изначально был реализован метод Рунге-Кутта без автоматического управления шагом: на каждой итерации длина шага просто делилась пополам. Затем было добавлено управление шагом для этого метода. На рисунке 1 представлен вывод результата работы программы.

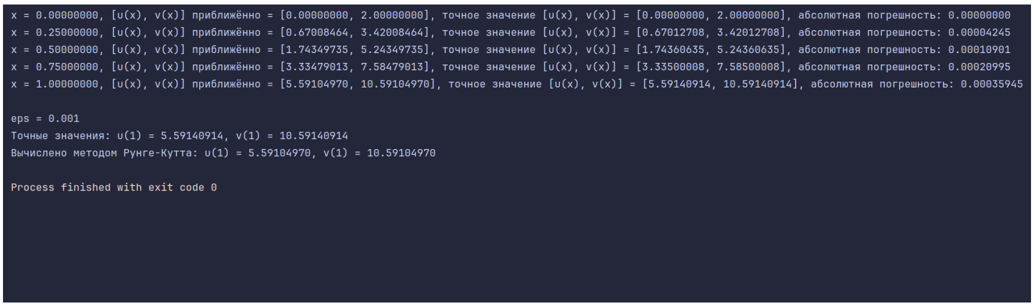

Рис. 1 — Результат выполнения программы

4 Вывод

В данной лабораторной работе был реализован метод Рунге-Кутта, позволяющий приближённо находить решение задачи Коши для системы обыкновенных дифференциальных уравнений. Кроме того, были выведены результаты работы реализованного метода и точного решения на каждом шаге. Также было проведено сравнение работы метода без управления шагом и с использованием алгоритма автоматического управления шагом. При сравнении погрешность была установлена как $\varepsilon = 0.00001$. Был получен следующий результат: в первом случае итоговое количество точек разбиения отрезка оказалось равно 17, тогда как во втором случае – 12, то есть алгоритм управления шагом уменьшил количество вычислений, сохранив точность.