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

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №6

«Численное решение обыкновенных дифференциальных уравнений»

по дисциплине «Вычислительная математика»

Вариант: **9**

Преподаватель:
Машина Екатерина Алексеевна

Выполнил:
Пронкин Алексей Дмитриевич
Группа: P3208

Санкт-Петербург, 2025 г.

Цель работы

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

Описание алгоритма решения задачи

1. Пользователь выбирает одно из трёх ОДУ вида $y' = f(x, y)$, для которых известно точное решение.
2. Вводятся начальные условия x_0, y_0, конечная точка x_n, шаг интегрирования h и критерий точности ε.
3. С помощью трёх численных методов (Эйлера, усовершенствованного Эйлера, Милна–Мортона) строится приближённое решение на сетке $x_0, x_1, \dots, x_N$.
4. Для одношаговых методов (Эйлер и улучшённый Эйлер) оценка погрешности производится по правилу Рунге, сравнивая решения при шагах h и $h/2$.
5. Для многошагового метода Милна–Мортона погрешность оценивается по максимальному отклонению от точного решения:

$$\varepsilon_{\text{max}} = \max_{0 \le i \le N} \left| y_{\text{точн}}(x_i) - y_i \right|.$$

6. Результаты (таблицы значений и оценки погрешностей) выводятся в консоль, а для выбранного метода строится график точного и численного решения.

Рабочие формулы

Метод Эйлера

$$y_{i+1} = y_i + h\, f(x_i, y_i).$$

Улучшённый метод Эйлера (метод средних наклонов)

$$\begin{aligned}
k_1 &= f(x_i, y_i),\\
k_2 &= f(x_i + h,\ y_i + h\, k_1),\\
y_{i+1} &= y_i + \tfrac{h}{2}(k_1 + k_2).
\end{aligned}$$

Метод Милна

$$y_{i+1}^{(p)} = y_{i-3} + \frac{4h}{3}\big[2f(x_{i-2}, y_{i-2}) - f(x_{i-1}, y_{i-1}) + 2f(x_i, y_i)\big],$$

$$y_{i+1} = y_{i-1} + \frac{h}{3}\big[f(x_{i-1}, y_{i-1}) + 4f(x_i, y_i) + f(x_{i+1}, y_{i+1}^{(p)})\big].$$

Листинг программы

```python
import numpy as np
import math
import matplotlib.pyplot as plt


# Define ODEs and their exact solutions
# 1) y' = y - x^2 + 1, exact: y = (x+1)^2 - 0.5*e^x
# 2) y' = -2*x*y^2,    exact: y = 1/(x^2 + C)
# 3) y' = y + x,       exact: y = Ce^x - x - 1

# Right-hand sides
def f1(x, y):
    return y - x ** 2 + 1


def f2(x, y):
    return -2 * x * y ** 2


def f3(x, y):
    return y + x


# Exact solutions with initial condition y(x0)=y0
def y1_exact(x, y0, x0):
    A = (y0 - ((x0 + 1) ** 2 - 0.5 * math.exp(x0))) / math.exp(x0)
    return (x + 1) ** 2 - 0.5 * math.exp(x) + A * math.exp(x)


def y2_exact(x, y0, x0):
    C = 1 / y0 - x0 ** 2
    return 1 / (x ** 2 + C)


def y3_exact(x, y0, x0):
    C = (y0 + x0 + 1) / math.exp(x0)
    return C * math.exp(x) - x - 1


class ODESolver:
    def __init__(self, f, y_exact=None):
        self.f = f
        self.y_exact = y_exact

    def euler(self, x0, y0, xn, h):
        xs = np.arange(x0, xn + h, h)
        ys = np.zeros_like(xs)
        ys[0] = y0
        for i in range(1, len(xs)):
            ys[i] = ys[i - 1] + h * self.f(xs[i - 1], ys[i - 1])
        return xs, ys

    def improved_euler(self, x0, y0, xn, h):
        xs = np.arange(x0, xn + h, h)
        ys = np.zeros_like(xs)
        ys[0] = y0
        for i in range(1, len(xs)):
```

```
            k1 = self.f(xs[i - 1], ys[i - 1])
            y_pred = ys[i - 1] + h * k1
            k2 = self.f(xs[i], y_pred)
            ys[i] = ys[i - 1] + h * (k1 + k2) / 2
        return xs, ys

    def milne(self, x0, y0, xn, h):
        xs = np.arange(x0, xn + h, h)
        ys = np.zeros_like(xs)
        ys[0] = y0
        # Initialize first 3 points by improved Euler
        for i in range(1, 4):
            k1 = self.f(xs[i - 1], ys[i - 1])
            y_pred = ys[i - 1] + h * k1
            k2 = self.f(xs[i], y_pred)
            ys[i] = ys[i - 1] + h * (k1 + k2) / 2
        # Milne predictor-corrector
        for i in range(4, len(xs)):
            y_pred = ys[i - 4] + 4 * h / 3 * (
                        2 * self.f(xs[i - 3], ys[i - 3]) - self.f(xs[i - 2], ys[i - 2]) +

            y_corr = ys[i - 2] + h / 3 * (
                        self.f(xs[i - 2], ys[i - 2]) + 4 * self.f(xs[i - 1], ys[i - 1]) +
            ys[i] = y_corr
        return xs, ys


# Error estimation: Runge rule for one-step methods
def runge_error(solver, method, p, x0, y0, xn, h):
    xs1, ys1 = method(x0, y0, xn, h)
    _, ys2 = method(x0, y0, xn, h / 2)
    ys2_at_h = ys2[::2]
    return np.max(np.abs((ys2_at_h - ys1) / (2 ** p - 1)))


# Multi-step max error vs exact
def max_error_exact(xs, ys, y_exact, y0, x0):
    exact_vals = np.array([y_exact(x, y0, x0) for x in xs])
    return np.max(np.abs(exact_vals - ys))


if __name__ == '__main__':
    # Menu of ODEs
    funcs = [
        (f1, y1_exact, "y' = y - x^2 + 1"),
        (f2, y2_exact, "y' = -2*x*y^2"),
        (f3, y3_exact, "y' = y + x")
    ]
    print('Select ODE:')
    for i, (_, _, desc) in enumerate(funcs, 1):
        print(f"{i}. {desc}")
    choice = int(input('> ')) - 1
    f, y_exact, desc = funcs[choice]
    x0 = float(input('x0 = '))
    y0 = float(input('y0 = '))
    xn = float(input('xn = '))
    h = float(input('step h = '))
    eps = float(input('epsilon = '))
```

```python
    solver = ODESolver(f, y_exact)
    methods = [
        ('Euler', solver.euler, 1),
        ('Improved Euler', solver.improved_euler, 2),
        ('Milne', solver.milne, None)
    ]

    # Compute and display table
    print('\nResults:')
    for name, method, p in methods:
        xs, ys = method(x0, y0, xn, h)
        if p is not None:
            err = runge_error(solver, method, p, x0, y0, xn, h)
        else:
            err = max_error_exact(xs, ys, y_exact, y0, x0)
        print(f"\n{name}: max error = {err:.5e}")
        print('  x    \t y_approx\t y_exact')
        for x_val, y_val in zip(xs, ys):
            print(f"  {x_val:.3f}\t {y_val:.6f}\t {y_exact(x_val, y0, x0):.6f}")

    # Plot all solutions vs exact
    xs = np.arange(x0, xn + h, h)
    ys_exact = [y_exact(x, y0, x0) for x in xs]

    plt.figure()
    plt.plot(xs, ys_exact, label='Exact', linewidth=2)

    # Euler
    _, ys_euler = solver.euler(x0, y0, xn, h)
    plt.plot(xs, ys_euler, '--', label='Euler')

    # Improved Euler
    _, ys_imp = solver.improved_euler(x0, y0, xn, h)
    plt.plot(xs, ys_imp, '-.', label='Improved Euler')

    # Milne
    _, ys_mil = solver.milne(x0, y0, xn, h)
    plt.plot(xs, ys_mil, ':', label='Milne')

    plt.legend()
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Exact vs Numerical Solutions')
    plt.grid(True)
    plt.show()
```

Выводы

- Одношаговые методы показывают закономерное уменьшение ошибки при повышении порядка: улучшённый метод Эйлера (2-й порядок) даёт точность на порядок лучше, чем прямой метод Эйлера (1-й порядок) при тех же параметрах.
- Метод Милна (4-й порядок) продемонстрировал наилучшую точность для гладких не жёстких ОДУ, однако потребовал предварительного «разгона» методом улучшённого Эйлера и хранения нескольких предыдущих значений.
- При выборе шага h и критерия ε следует учитывать баланс между точностью и вычисли-

тельными затратами: слишком маленький шаг снижает эффективность, слишком большой — увеличивает погрешность.

- Визуализация решений на графиках подтверждает численные оценки и наглядно показывает зоны расхождения.