Университет ИТМО

Факультет Программной Инженерии и Компьютерных Техники

Лабораторная работа №6

Вариант № 15

Выполнила:
Студент группы Р3213
Юсупова Алиса Ильясовна
Преподаватель:

Преподаватель практики

Санкт-Петербург

2025

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

№ варианта задания лабораторной работы определяется как номер в списке группы согласно ИСУ.

1. Порядок выполнения работы

2. В программе численные методы решения обыкновенных дифференциальных уравнений (ОДУ) должен быть реализован в виде отдельного класса /метода/функции;
3. Пользователь выбирает ОДУ вида $y' = f(x, y)$ (не менее трех уравнений), из тех, которые предлагает программа;
4. Предусмотреть ввод исходных данных с клавиатуры: начальные условия $y_0 = y(x_0)$, интервал дифференцирования $[x_0, x_n]$, шаг h, точность ε;
5. Для исследования использовать одношаговые методы и многошаговые методы (см. табл.1);
6. Составить таблицу приближенных значений интеграла дифференциального уравнения, удовлетворяющего начальным условиям, для всех методов, реализуемых в программе;
7. Для оценки точности одношаговых методов использовать правило Рунге;
8. Для оценки точности многошаговых методов использовать точное решение задачи: $\varepsilon = \max_{0 \le i \le n} |y_{i\text{точн}} - y_i|$;
9. Построить графики точного решения и полученного приближенного решения (разными цветами);
10. Программа должна быть протестирована при различных наборах данных, в том числе и некорректных.
11. Проанализировать результаты работы программы.

Программная реализация:

Main:

```
from methods import EulerMethod, RungeKutta4Method, AdamsMethod, Runge_rule
from input_output_handler import InputOutputHandler
from plot import Plotter

def main():
    io_handler = InputOutputHandler()
    plotter = Plotter()

    # Получение входных данных от пользователя
    equation_choice, x0, y0, xn, h, epsilon, exact_solution =
io_handler.get_input()

    # Инициализация методов
    euler = EulerMethod()
    rk4 = RungeKutta4Method()
    adams = AdamsMethod()

    # Решение уравнения выбранными методами
    methods = {
```

```
        '1': ('Метод Эйлера', euler.solve, 1),
        '2': ('Метод Рунге-Кутта 4 порядка', rk4.solve, 4),
        '3': ('Метод Адамса', adams.solve, None)
    }

    results = {}
    for method_num, (method_name, solver, order) in methods.items():
        x_values, y_values = solver(equation_choice, x0, y0, xn, h)
        results[method_name] = (x_values, y_values)

        # Оценка точности
        if order is not None:  # Для одношаговых методов используем правило
Рунге
            _, y_values_half = solver(equation_choice, x0, y0, xn, h / 2)
            error = Runge_rule(y_values[-1], y_values_half[-1], order)
        else:  # Для многошаговых методов сравниваем с точным решением
            if exact_solution:
                exact_values = [exact_solution(x) for x in x_values]
                error = max(abs(y - exact) for y, exact in zip(y_values,
exact_values))
            else:
                error = float('nan')

        print(f"\n{method_name}:")
        print(f"Погрешность: {error:.6f}")

    # Вывод результатов в таблицу
    exact_values = [exact_solution(x) for x in results['Метод Эйлера'][0]] if
exact_solution else None
    io_handler.print_results_table(
        results['Метод Эйлера'],
        results['Метод Рунге-Кутта 4 порядка'],
        results['Метод Адамса'],
        exact_values
    )

    # Построение графиков
    plotter.plot_results(
        results['Метод Эйлера'],
        results['Метод Рунге-Кутта 4 порядка'],
        results['Метод Адамса'],
        exact_solution
    )


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

methods:

```
# methods.py
class EulerMethod:
    def solve(self, equation_num, x0, y0, xn, h):
        f = self._get_equation(equation_num)
        x_values = []
        y_values = []

        x = x0
        y = y0
        x_values.append(x)
        y_values.append(y)

        while x < xn:
            y += h * f(x, y)
            x += h
```

```python
            x_values.append(x)
            y_values.append(y)

        return x_values, y_values

    def _get_equation(self, num):
        equations = {
            '1': lambda x, y: x + y,
            '2': lambda x, y: x**2 + y**2,
            '3': lambda x, y: 2 * x - y
        }
        return equations.get(num, lambda x, y: x + y)


class RungeKutta4Method:
    def solve(self, equation_num, x0, y0, xn, h):
        f = self._get_equation(equation_num)
        x_values = []
        y_values = []

        x = x0
        y = y0
        x_values.append(x)
        y_values.append(y)

        while x < xn:
            k1 = h * f(x, y)
            k2 = h * f(x + h/2, y + k1/2)
            k3 = h * f(x + h/2, y + k2/2)
            k4 = h * f(x + h, y + k3)

            y += (k1 + 2*k2 + 2*k3 + k4) / 6
            x += h
            x_values.append(x)
            y_values.append(y)

        return x_values, y_values

    def _get_equation(self, num):
        equations = {
            '1': lambda x, y: x + y,
            '2': lambda x, y: x**2 + y**2,
            '3': lambda x, y: 2*x - y
        }
        return equations.get(num, lambda x, y: x + y)


class AdamsMethod:
    def solve(self, equation_num, x0, y0, xn, h):
        f = self._get_equation(equation_num)
        x_values = []
        y_values = []

        # Используем Рунге-Кутта для получения первых 4 точек
        rk4 = RungeKutta4Method()
        start_x, start_y = rk4.solve(equation_num, x0, y0, x0 + 3*h, h)

        x_values.extend(start_x)
        y_values.extend(start_y)

        x = x0 + 3*h
        while x < xn:
            # Предыдущие значения функции
            f0 = f(x_values[-4], y_values[-4])
```

```
            f1 = f(x_values[-3], y_values[-3])
            f2 = f(x_values[-2], y_values[-2])
            f3 = f(x_values[-1], y_values[-1])

            # Формула Адамса (предиктор)
            y_next = y_values[-1] + h*(55*f3 - 59*f2 + 37*f1 - 9*f0)/24
            x_next = x + h

            # Корректор (одна итерация)
            f_next = f(x_next, y_next)
            y_next = y_values[-1] + h*(9*f_next + 19*f3 - 5*f2 + f1)/24

            x_values.append(x_next)
            y_values.append(y_next)
            x = x_next

        return x_values, y_values

    def _get_equation(self, num):
        """Возвращает выбранное уравнение"""
        equations = {
            '1': lambda x, y: x + y,
            '2': lambda x, y: x**2 + y**2,
            '3': lambda x, y: 2*x - y
        }
        return equations.get(num, lambda x, y: x + y)


def Runge_rule(y_h, y_h2, p):
    return abs(y_h - y_h2) / (2 ** p - 1)
```

Input_output_nahdlers:

```
# input_output_handler.py
import math


class InputOutputHandler:
    def __init__(self):
        self.equations = {
            '1': {
                'text': "y' = x + y",
                'solution': lambda x0, y0, x: -x - 1 + (y0 + x0 + 1) *
math.exp(x - x0)
            },
            '2': {
                'text': "y' = x^2 + y^2",
                'solution': None
            },
            '3': {
                'text': "y' = 2x - y",
                'solution': lambda x0, y0, x: 2 * x - 2 + (y0 - 2 * x0 + 2) *
math.exp(-(x - x0))
            }
        }

    def get_input(self):
        """Получает входные данные от пользователя"""
        print("Доступные дифференциальные уравнения:")
        for num, eq in self.equations.items():
            print(f"{num}. {eq['text']}")

        while True:
            choice = input("Выберите уравнение (1-3): ")
            if choice in self.equations:
```

```
                break
            print("Неверный ввод. Попробуйте снова.")

        while True:
            try:
                x0 = float(input("Введите начальное значение x0: "))
                y0 = float(input(f"Введите начальное значение y0 = y({x0}):
"))
                xn = float(input("Введите конечное значение xn: "))
                h = float(input("Введите шаг h: "))
                epsilon = float(input("Введите точность ε: "))

                if xn <= x0:
                    print("xn должно быть больше x0. Попробуйте снова.")
                    continue
                if h <= 0:
                    print("Шаг h должен быть положительным. Попробуйте
снова.")
                    continue
                if epsilon <= 0:
                    print("Точность ε должна быть положительной. Попробуйте
снова.")
                    continue

                break
            except ValueError:
                print("Неверный формат числа. Попробуйте снова.")

        # Формируем точное решение, если оно доступно
        exact_solution = None
        if self.equations[choice]['solution'] is not None:
            exact_solution = lambda x: self.equations[choice]['solution'](x0,
y0, x)

        return choice, x0, y0, xn, h, epsilon, exact_solution

    def print_results_table(self, euler_results, rk4_results, adams_results,
exact_results=None):
        """Выводит таблицу результатов"""
        x_euler, y_euler = euler_results
        x_rk4, y_rk4 = rk4_results
        x_adams, y_adams = adams_results

        print("\nРезультаты численного решения:")
        header = "| {:^10} | {:^15} | {:^15} | {:^15} |".format(
            "x", "Метод Эйлера", "Рунге-Кутта 4", "Метод Адамса")
        if exact_results:
            header = header[:-1] + " {:^15} |".format("Точное решение")
        print(header)
        print("-" * len(header))

        max_len = max(len(x_euler), len(x_rk4), len(x_adams))
        for i in range(max_len):
            x = x_euler[i] if i < len(x_euler) else "-"
            euler_val = y_euler[i] if i < len(y_euler) else "-"
            rk4_val = y_rk4[i] if i < len(y_rk4) else "-"
            adams_val = y_adams[i] if i < len(y_adams) else "-"
            exact_val = exact_results[i] if exact_results and i <
len(exact_results) else "-"

            row = "| {:^10.4f} | {:^15.6f} | {:^15.6f} | {:^15.6f} |".format(
                x if x != "-" else "-",
                euler_val if euler_val != "-" else "-",
                rk4_val if rk4_val != "-" else "-",
```

```
                adams_val if adams_val != "-" else "-")

            if exact_results:
                row = row[:-1] + " {:^15.6f} |".format(
                    exact_val if exact_val != "-" else "-")

            print(row)
```

Plot:

```
import matplotlib.pyplot as plt


class Plotter:
    def plot_results(self, euler_results, rk4_results, adams_results,
exact_solution=None):
        plt.figure(figsize=(12, 8))

        x_euler, y_euler = euler_results
        x_rk4, y_rk4 = rk4_results
        x_adams, y_adams = adams_results

        plt.plot(x_euler, y_euler, 'b-', label='Метод Эйлера', linewidth=2)
        plt.plot(x_rk4, y_rk4, 'g-', label='Рунге-Кутта 4 порядка',
linewidth=2)
        plt.plot(x_adams, y_adams, 'r-', label='Метод Адамса', linewidth=2)

        # Если есть точное решение, строим его
        if exact_solution is not None:
            x_exact = x_rk4  # Используем точки метода Рунге-Кутта как
наиболее точные
            y_exact = [exact_solution(x) for x in x_exact]
            plt.plot(x_exact, y_exact, 'k--', label='Точное решение',
linewidth=2)

        plt.title('Сравнение численных методов решения ОДУ')
        plt.xlabel('x')
        plt.ylabel('y(x)')
        plt.legend()
        plt.grid(True)

        plt.show()
```

Вывод: Я реализовала программу на языке Python для решения задачи Коши для обыкновенных дифференциальных уравнений численными методами.