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

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №6

Вариант 3

Группа: P3269

Выполнили:

Грибкова В.Е

Долганова О.А

Проверил:

Машина Е. А.

Г. Санкт-Петербург

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

2. Листинг программы.

```
from scipy.interpolate import interp1d
import numpy as np
import matplotlib.pyplot as plt

# Функция для уравнения y' + 2y = x^2
def f1(x, y):
    return x ** 2 - 2 * y  # начальные условия: y(0) = 1; интервал [0, 1], шаг h = 0.1

# Функция для уравнения y' = 2x
def f2(x, y):
    return 2 * x  # начальные условия: y(0) = 0, интервал [0, 7], шаг h = 1

# Функция для уравнения y' = y + (1 + x) * x^2
def f3(x, y):
    return y + (1 + x) * x**2

# Функция для выбора уравнения
def selectEquation():
    while True:
        # Вывод доступных функций для выбора
        print("Выберете функцию \"1\" -> y' + 2y = x^2, \"2\" -> y' = 2x, \"3\" -> y' = y + (1 + x) * x^2:")
        try:
            # Получение выбора пользователя
            choice = int(input("Введите номер уравнения: "))
            # Словарь для сопоставления выбора с функциями
            equations = {1: f1, 2: f2, 3: f3}
            # Возвращение выбранной функции
            return equations[choice]
        except KeyError:
            # Обработка ошибки неверного ввода ключа
            print("Ошибка: Неверный ввод. Пожалуйста, введите номер уравнения из списка.")
        except ValueError:
            # Обработка ошибки ввода, если введено не целое число
            print("Ошибка: Введите целое число.")

# Функция для вычисления точного решения уравнения
def exactSolution(x, equation):
```

```
    if equation == f1:
        return (1 / np.exp(2 * x)) + (x**2 / 2) - (x / 2) + (1 / 4)
# Точное решение для f1
    elif equation == f2:
        return x**2  # Точное решение для f2
    elif equation == f3:
        return np.exp(x)-x**3-4*x**2-8*x-8  # Точное решение для f3

# Метод Эйлера для численного решения ОДУ
def eulerMethod(f, y0, a, b, h):
    t = np.arange(a, b + h, h)  # создание массива значений времени
    y = np.zeros(len(t))  # создание массива значений решения
    y[0] = y0  # начальное условие
    for i in range(1, len(t)):
        y[i] = y[i-1] + h * f(t[i-1], y[i-1])  # расчет значения y
на следующем шаге
    return t, y  # возвращение массивов времени и решения

# Адаптивный улучшенный метод Эйлера
def adaptiveImprovedEulerMethod(f, y0, a, b, h, epsilon):
    t = [a]  # начальный список значений времени
    y = [y0]  # начальный список значений решения
    p = 2  # порядок метода

    while t[-1] < b:
        t_current = t[-1]  # текущее значение времени
        y_current = y[-1]  # текущее значение решения

        # Один шаг метода Эйлера с полным шагом h
        k1_h = h * f(t_current, y_current)
        k2_h = h * f(t_current + h, y_current + k1_h)
        y_h = y_current + 0.5 * (k1_h + k2_h)

        # Два шага метода Эйлера с половинным шагом h/2
        h_half = h / 2
        k1_h_half_1 = h_half * f(t_current, y_current)
        k2_h_half_1 = h_half * f(t_current + h_half, y_current +
k1_h_half_1)
        y_h_half_1 = y_current + 0.5 * (k1_h_half_1 + k2_h_half_1)

        k1_h_half_2 = h_half * f(t_current + h_half, y_h_half_1)
        k2_h_half_2 = h_half * f(t_current + h, y_h_half_1 +
k1_h_half_2)
        y_h_half_2 = y_h_half_1 + 0.5 * (k1_h_half_2 + k2_h_half_2)

        # Оценка ошибки
        error = np.abs(y_h_half_2 - y_h) / (2**p - 1)
```

```
        if error <= epsilon:
            t.append(t_current + h)  # обновление времени
            y.append(y_h_half_2)  # обновление решения
            if error < epsilon / 2:
                h *= 2  # увеличение шага, если ошибка меньше
половины допустимой
        else:
            h /= 2  # уменьшение шага, если ошибка больше
допустимой

        # Обеспечение, чтобы шаг не выходил за пределы интервала
        if t[-1] + h > b:
            h = b - t[-1]

    return np.array(t), np.array(y)  # возвращение массивов времени
и решения

# Метод Рунге-Кутта 4-го порядка для численного решения ОДУ
def rungeKutta4(f, y0, a, b, h):
    t = np.arange(a, b + h, h)  # создание массива значений времени
    y = np.zeros(len(t))  # создание массива значений решения
    y[0] = y0  # начальное условие
    for i in range(1, len(t)):
        k1 = h * f(t[i-1], y[i-1])
        k2 = h * f(t[i-1] + 0.5 * h, y[i-1] + 0.5 * k1)
        k3 = h * f(t[i-1] + 0.5 * h, y[i-1] + 0.5 * k2)
        k4 = h * f(t[i], y[i-1] + k3)
        y[i] = y[i-1] + (k1 + 2 * k2 + 2 * k3 + k4) / 6  # расчет
значения y на следующем шаге
    return t, y  # возвращение массивов времени и решения

# Метод Милна-Симпсона для численного решения ОДУ
def milneSimpson(f, y0, a, b, h):
    t = np.arange(a, b + h, h)  # создание массива значений времени
    y = np.zeros(len(t))  # создание массива значений решения
    y[0] = y0  # начальное условие

    # Начальные значения вычисляются с помощью метода Рунге-Кутта
4-го порядка
    for i in range(1, 4):
        k1 = h * f(t[i-1], y[i-1])
        k2 = h * f(t[i-1] + 0.5 * h, y[i-1] + 0.5 * k1)
        k3 = h * f(t[i-1] + 0.5 * h, y[i-1] + 0.5 * k2)
        k4 = h * f(t[i], y[i-1] + k3)
        y[i] = y[i-1] + (k1 + 2 * k2 + 2 * k3 + k4) / 6

    # Основной цикл метода Милна-Симпсона
    for i in range(3, len(t)-1):
```

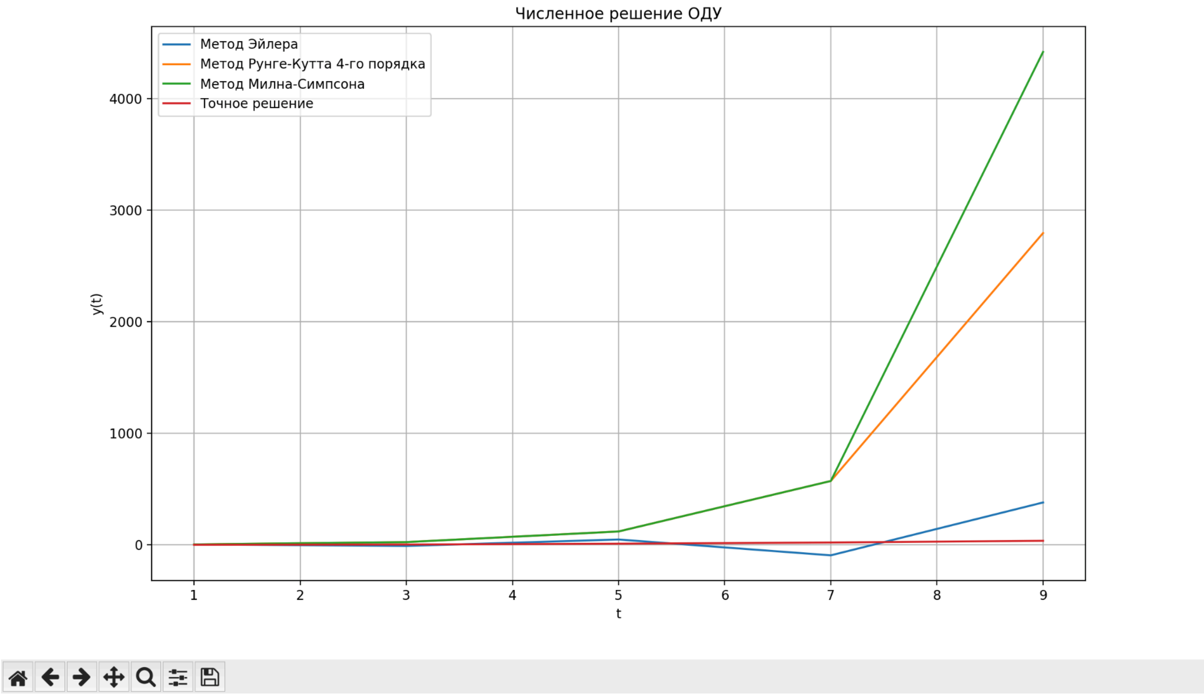
```
        # Предсказание значения
        y_pred = y[i-3] + 4 * h / 3 * (2 * f(t[i-2], y[i-2]) -
f(t[i-1], y[i-1]) + 2 * f(t[i], y[i]))
        # Коррекция значения
        y_corr = y[i-1] + h / 3 * (f(t[i-1], y[i-1]) + 4 * f(t[i],
y[i]) + f(t[i+1], y_pred))
        y[i+1] = y_corr
    return t, y  # возвращение массивов времени и решения

# Функция для построения графиков решений
def plotResults(t_euler, t_adaptive_euler, y_euler,
y_adaptive_euler, y_rk4, y_milne, exact_y):
    plt.plot(t_euler, y_euler, label='Метод Эйлера')
    plt.plot(t_euler, y_rk4, label='Метод Рунге-Кутта 4-го
порядка')
    plt.plot(t_euler, y_milne, label='Метод Милна-Симпсона')
    plt.plot(t_euler, exact_y, label='Точное решение')

    plt.legend()  # Добавление легенды
    plt.xlabel('t')  # Подпись оси X
    plt.ylabel('y(t)')  # Подпись оси Y
    plt.title('Численное решение ОДУ')  # Заголовок графика
    plt.grid(True)  # Включение сетки на графике
    plt.show()  # Отображение графика

# Функция для оценки точности численных методов
def evaluateAccuracy(t_euler, y_euler, t_adaptive_euler,
y_adaptive_euler, t_rk4, y_rk4, t_milne, y_milne, exact_y):
    # Максимальная ошибка метода Милна-Симпсона
    epsilon_milne = np.max(np.abs(exact_y - y_milne))
    print(f"Максимальная ошибка метода Милна-Симпсона:
{epsilon_milne}")

    # Интерполяция значений для адаптивного улучшенного метода
Эйлера
    interp_adaptive_euler = interp1d(t_adaptive_euler,
y_adaptive_euler, fill_value="extrapolate")
    y_adaptive_euler_interpolated_rk4 =
interp_adaptive_euler(t_rk4)
    y_adaptive_euler_interpolated_euler =
interp_adaptive_euler(t_euler)

    # Оценка точности методом Рунге для метода Эйлера и метода
Рунге-Кутта 4-го порядка
    runge_error_euler = np.abs(y_euler[-1] -
y_adaptive_euler_interpolated_euler[-1]) / (2**1 - 1)
    runge_error_rk4 = np.abs(y_rk4[-1] -
y_adaptive_euler_interpolated_rk4[-1]) / (2**4 - 1)
```

```
    print(f"Оценка точности метода Эйлера по правилу Рунге:
{runge_error_euler}")
    print(f"Оценка точности метода Рунге-Кутта по правилу Рунге:
{runge_error_rk4}")

# Основная функция программы
def main():
    while True:
        try:
            equation = selectEquation()  # выбор уравнения
пользователем
            y0 = float(input("Введите начальное условие y0: "))  #
ввод начального условия
            a = float(input("Введите левую границу отрезка: "))  #
ввод левой границы интервала
            b = float(input("Введите правую границу отрезка: "))  #
ввод правой границы интервала
            h = float(input("Введите начальный шаг h: "))  # ввод
начального шага
            epsilon = float(input("Введите желаемую точность: "))
# ввод желаемой точности

            # Решение ОДУ с помощью различных численных методов
            t_euler, y_euler = eulerMethod(equation, y0, a, b, h)
            t_adaptive_euler, y_adaptive_euler =
adaptiveImprovedEulerMethod(equation, y0, a, b, h, epsilon)
            t_rk4, y_rk4 = rungeKutta4(equation, y0, a, b, h)
            t_milne, y_milne = milneSimpson(equation, y0, a, b, h)

            # Вычисление точного решения
            exact_y = exactSolution(t_euler, equation)

            # Построение графиков результатов
            plotResults(t_euler, t_adaptive_euler, y_euler,
y_adaptive_euler, y_rk4, y_milne, exact_y)

            # Вывод численных решений и точного решения
            print("Метод Эйлера = ", y_euler)
            print("Рунге-Кутта = ", y_rk4)
            print("Милн Симпсон = ", y_milne)
            print("Точные значения = ", exact_y)

            # Оценка точности методов
            evaluateAccuracy(t_euler, y_euler, t_adaptive_euler,
y_adaptive_euler, t_rk4, y_rk4, t_milne, y_milne, exact_y)

            break  # Завершение цикла при успешном выполнении
```

```
        except ValueError:
            print("Ошибка: Неверный ввод. Пожалуйста, введите
числовое значение.")  # Обработка ошибки ввода

# Запуск основной функции
if __name__ == "__main__":
   main()
```

3. Результаты выполнения программы.

```
Выберете функцию "1" -> y' + 2y = x^2, "2" -> y' = 2x, "3" -> y' =
y + (1 + x) * x^2:
Введите номер уравнения: 1
Введите начальное условие y0: 4
Введите левую границу отрезка: 1
Введите правую границу отрезка: 9
Введите начальный шаг h: 2
Введите желаемую точность: 3
Метод Эйлера =  [  4. -10.  48. -94. 380.]
Рунге-Кутта =  [   4.           24.66666667  120.
572.66666667 2796.        ]
Милн Симпсон =  [4.00000000e+00 2.46666667e+01 1.20000000e+02
5.72666667e+02
4.42029630e+03]
Точные значения =  [ 0.38533528  3.25247875 10.2500454
21.25000083 36.25000002]
Максимальная ошибка метода Милна-Симпсона: 4384.046296281065
Оценка точности метода Эйлера по правилу Рунге: 343.53271484375
Оценка точности метода Рунге-Кутта по правилу Рунге:
183.96884765625
```


Численное решение ОДУ
Метод Эйлера
Метод Рунге-Кутта 4-го порядка
Метод Милна-Симпсона
Точное решение
4000
3000
2000
1000
0
y(t)
1
2
3
4
5
6
7
8
9
t

4. Выводы.