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

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №6
по дисциплине «Вычислительная математика»

Вариант 16

Преподаватель:

Машина Е.А.

Выполнила:

Шайхутдинова Н.В.

P3208

Санкт-Петербург

2024 г.

Цель лабораторной работы

Решить задачу Коши для обыкновенных дифференциальных уравнений численными методами.

Программная реализация задачи

```
def eulerMethod(f, y0, a, b, h):
    t = np.arange(a, b + h, h)
    y = np.zeros(len(t))
    y[0] = y0
    for i in range(1, len(t)):
        y[i] = y[i - 1] + h * f(t[i - 1], y[i - 1])
    return t, y


def adaptiveImprovedEulerMethod(f, y0, a, b, h, epsilon):
    t = [a]
    y = [y0]
    p = 2

    while t[-1] < b:
        t_current = t[-1]
        y_current = y[-1]

        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)

        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)

        error = np.abs(y_h_half_2 - y_h) / (2**p - 1)

        if error <= epsilon:
            t.append(t_current + h)
            y.append(y_h_half_2)
            if error < epsilon / 2:
                h *= 2
        else:
            h /= 2

        if t[-1] + h > b:
            h = b - t[-1]
```

```
    return np.array(t), np.array(y)


def rungeKutta4(f, y0, a, b, h):
    t = np.arange(a, b + h, h)
    y = np.zeros(len(t))
    y[0] = y0
    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
    return t, y


def milneSimpson(f, y0, a, b, h):
    t = np.arange(a, b + h, h)
    y = np.zeros(len(t))
    y[0] = y0

    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

    for i in range(3, len(t) - 1):
        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])
        )
        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
```

Результат программы

Выберете функцию "1" -> y' + 2y = x^2, "2" -> y' = 2x, "3" -> y' = y + (1 + x) * x^2:

Введите номер уравнения: 1

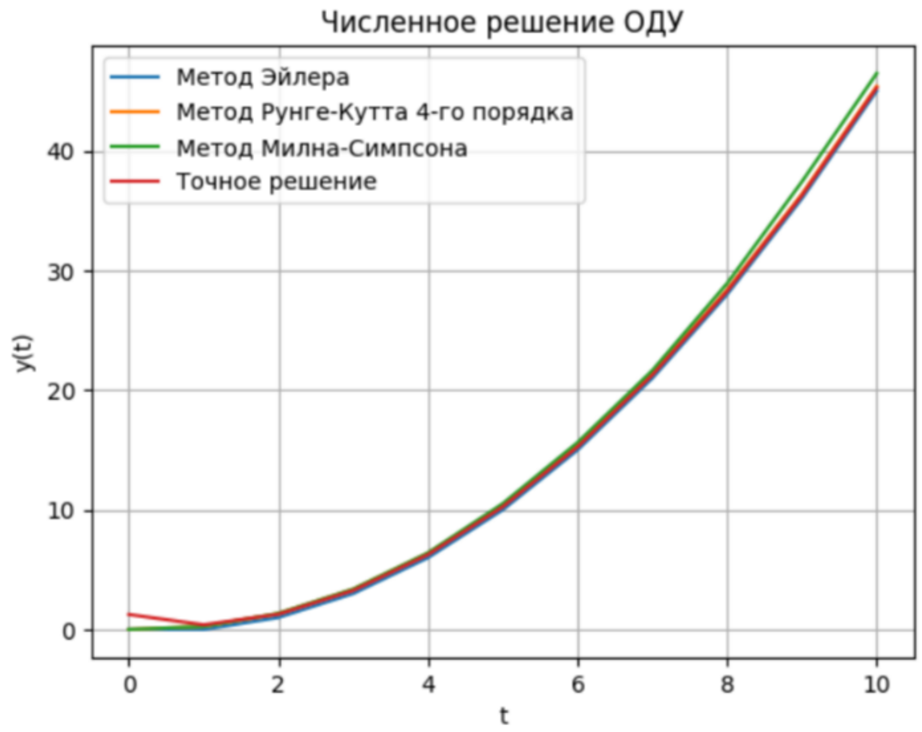
Введите начальное условие y0: 0

Введите левую границу отрезка: 0

Введите правую границу отрезка: 10

Введите начальный шаг h: 1

Введите желаемую точность: 0.01



i	x_i	Метод Эйлера	Рунге-Кутта	Милн-Симпсон	Точные значения
0	0	0	0	0	1.25
1	1	0	0.25	0.25	0.385335
2	2	1	1.33333	1.33333	1.26832
3	3	3	3.36111	3.36111	3.25248
4	4	6	6.37037	6.39506	6.25034
5	5	10	10.3735	10.5147	10.25
6	6	15	15.3745	15.6153	15.25
7	7	21	21.3748	21.634	21.25

8	8	28	28.3749	28.9083	28.25
9	9	36	36.375	37.4028	36.25
10	10	45	45.375	46.4457	45.25

Максимальная ошибка метода Милна-Симпсона: 1.25

Оценка точности метода Эйлера по правилу Рунге: 0.25446428507822105

Оценка точности метода Рунге-Кутта по правилу Рунге: 0.008035290950923486