Практическая работа №1.

Выполнил: Овинов Г.С. ИСУ-506420.

В данной работе было необходимо решить ОДУ аналитически, составить характеристическое уравнение. Использовав три метода интегрирования (явной метод Эйлера, неявный метод Эйлера, метод Рунге-Кутты) и сделать сравнительный анализ.

Цель: понять, какой численный метод лучше подходит для решения ОДУ второго порядка.

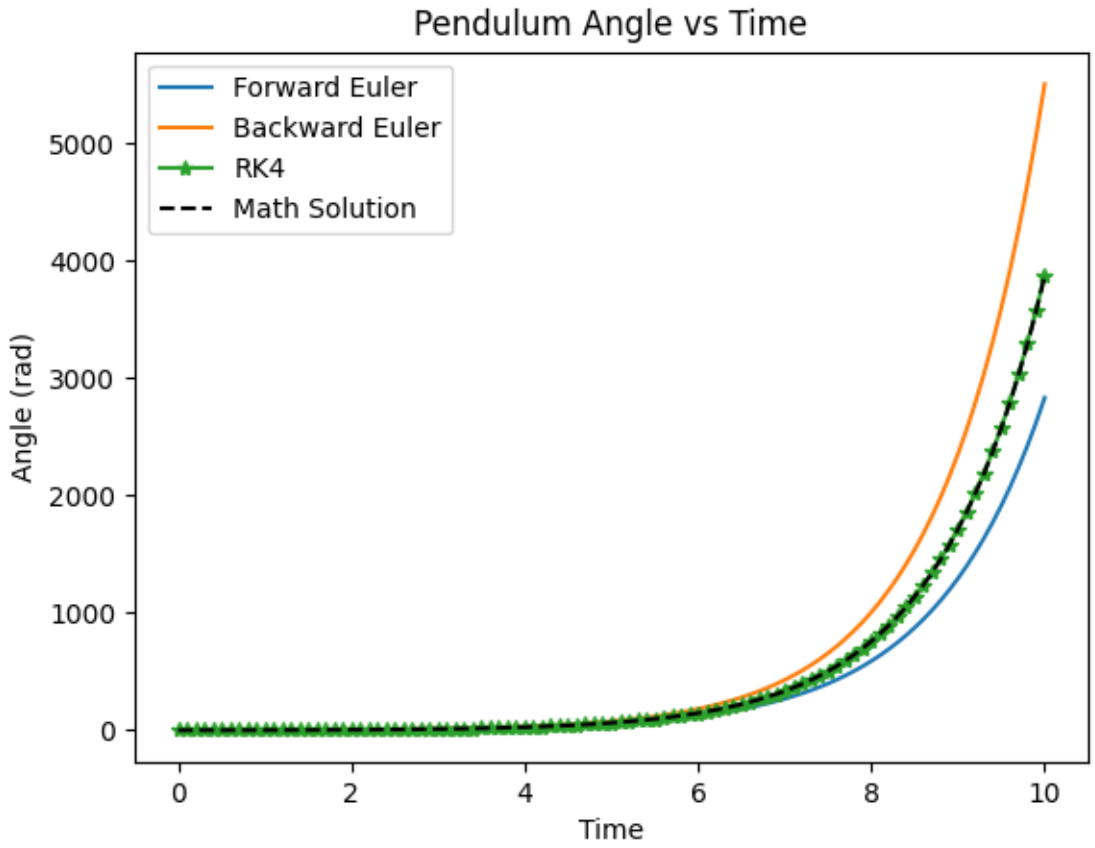
Решеие:

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

a, b, c, d = 7.81, -4.68, -1.36, 7.3
def ode(x):
    """
    State vector x = [x, x_dot]
    """
    x_ddot = (d - c*x[0] - b*x[1])/a

    return np.array([x[1], x_ddot])

def eq_solution(x0):
    t = np.arange(0, Tf + h, h)
    x = np.zeros((len(x0), len(t)))
    x[:, 0] = x0

    for k in range(1, len(t)):
        x[:, k] = [1.140 * m.exp(0.813 * t[k]) + 4.334 * m.exp(-0.214 * t[k]) - 5.374, 0.927 *
m.exp(0.813 * t[k]) - 0.927 * m.exp(-0.214 * t[k])]

    return x, t

def forward_euler(fun, x0, Tf, h):
    """
    Explicit Euler integration method
    """
    t = np.arange(0, Tf + h, h)
    x_hist = np.zeros((len(x0), len(t)))
    x_hist[:, 0] = x0

    for k in range(len(t) - 1):
        x_hist[:, k + 1] = x_hist[:, k] + h * fun(x_hist[:, k])

    return x_hist, t

def backward_euler(fun, x0, Tf, h, tol=1e-8, max_iter=100):
    """
    Implicit Euler integration method using fixed-point iteration
    """
```

```
    t = np.arange(0, Tf + h, h)
    x_hist = np.zeros((len(x0), len(t)))
    x_hist[:, 0] = x0

    for k in range(len(t) - 1):
        x_hist[:, k + 1] = x_hist[:, k]  # Initial guess

        for i in range(max_iter):
            x_next = x_hist[:, k] + h * fun(x_hist[:, k + 1])
            error = np.linalg.norm(x_next - x_hist[:, k + 1])
            x_hist[:, k + 1] = x_next

            if error < tol:
                break

    return x_hist, t

def runge_kutta4(fun, x0, Tf, h):
    """
    4th order Runge-Kutta integration method
    """
    t = np.arange(0, Tf + h, h)
    x_hist = np.zeros((len(x0), len(t)))
    x_hist[:, 0] = x0

    for k in range(len(t) - 1):
        k1 = fun(x_hist[:, k])
        k2 = fun(x_hist[:, k] + 0.5 * h * k1)
        k3 = fun(x_hist[:, k] + 0.5 * h * k2)
        k4 = fun(x_hist[:, k] + h * k3)

        x_hist[:, k + 1] = x_hist[:, k] + (h / 6.0) * (k1 + 2*k2 + 2*k3 + k4)

    return x_hist, t

# Test all integrators
x0 = np.array([0.1, 0.0])  # Initial state: [angle, angular_velocity]
Tf = 10.0
h = 0.1
x_sol, t_sol = eq_solution(x0)
# Forward Euler
x_fe, t_fe = forward_euler(ode, x0, Tf, h)
# Backward Euler
x_be, t_be = backward_euler(ode, x0, Tf, h)
# Runge-Kutta 4
x_rk4, t_rk4 = runge_kutta4(ode, x0, Tf, h)
# Plot results
plt.plot(t_fe, x_fe[0, :], label='Forward Euler')
plt.plot(t_be, x_be[0, :], label='Backward Euler')
plt.plot(t_rk4, x_rk4[0, :], label='RK4', marker='*')
plt.plot(t_sol, x_sol[0, :], label='Math Solution', color='black', linestyle='--')
plt.xlabel('Time')
```

```
plt.ylabel('Angle (rad)')
plt.legend()
plt.title('Pendulum Angle vs Time')

plt.show()
```

График сравнения методов решения ОДУ:



Вывод: Проведенное исследование позволило сравнить три численных метода интегрирования ОДУ второго порядка. Метод Рунге-Кутты 4-го порядка продемонстрировал наивысшую точность, что делает его предпочтительным выбором для задач, требующих высокой точности вычислений. Явный метод Эйлера, несмотря на простоту, показал значительную погрешность и неустойчивости при больших шагах интегрирования. Неявный метод Эйлера проявил лучшую устойчивость, однако его реализация требует дополнительных вычислительных затрат непрерывное решение нелинейных уравнений.