ИТМО

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

ОТЧЕТ

ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2

Работу выполнил:

Сосенский Е.К.

Группа:

R4136c

Преподаватель:

Ракшин Е. А.

Санкт-Петербург

2025

Цель

Целью данной работы является составление ОДУ по схеме (Рисунок 1) и его решение с помощью явного метода Эйлера, неявного метода Эйлера и метода Рунге-Кутты. Также, в ходе решения необходимо построить графики, а затем провести аналитическое решение и сравнить результаты.

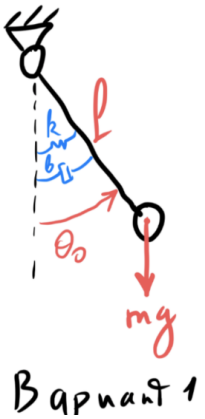

Рисунок 1 - Схема для первого варианта.

Решение

Для начала запишем вводные данные, взятые из таблицы с вариантами.

Таблица 1. Вводные данные.

$m,\ kg$	$k,\ N/m,$ Nm/rad	$b,\ N \cdot s/m,$ $Nm \cdot s/rad$	$l,\ m$	$theta_0,\ rad$	$x_0,\ m$
0.9	6.4	0.015	0.47	-1.445177505	0.47

Запишем ОДУ как Лагранжиан:

$$L = K - P$$

$$K = \frac{1}{2}m\omega^2 = \frac{1}{2}ml^2\dot{\theta}^2$$

$$P = -m \cdot g \cdot l \cdot \cos\theta + \frac{1}{2}k\theta^2$$

Далее используем метод Эйлера-Лагранжа:

$$\frac{d}{dx}\left(\frac{dL}{d\dot{x}}\right) - \frac{dL}{dx} = Q$$

$$Q = -b\dot{\theta}$$

В итоге получаем выражение:

$$\ddot{x} = -\frac{b\dot{\theta}}{ml^2} - \frac{g \cdot \sin\theta}{l} - \frac{k\theta}{ml^2}$$

Решение

Для начала прогоним получившееся выражение через код и получим следующие графики:

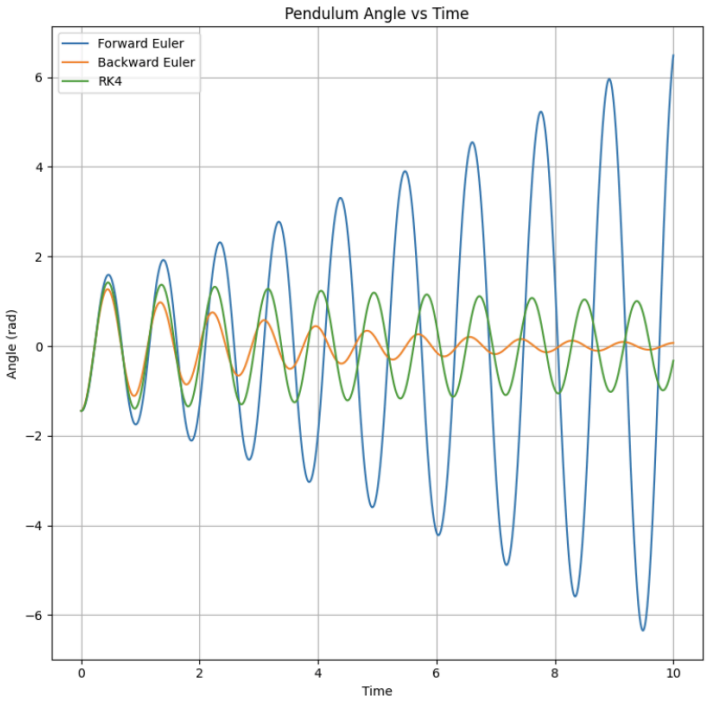

Рисунок 2 - График положения

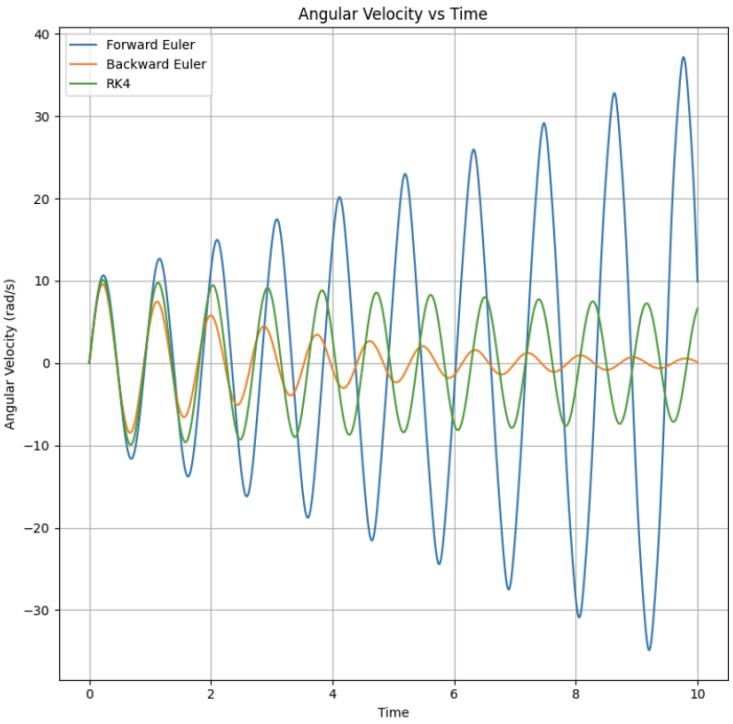

Рисунок 3 - График скорости

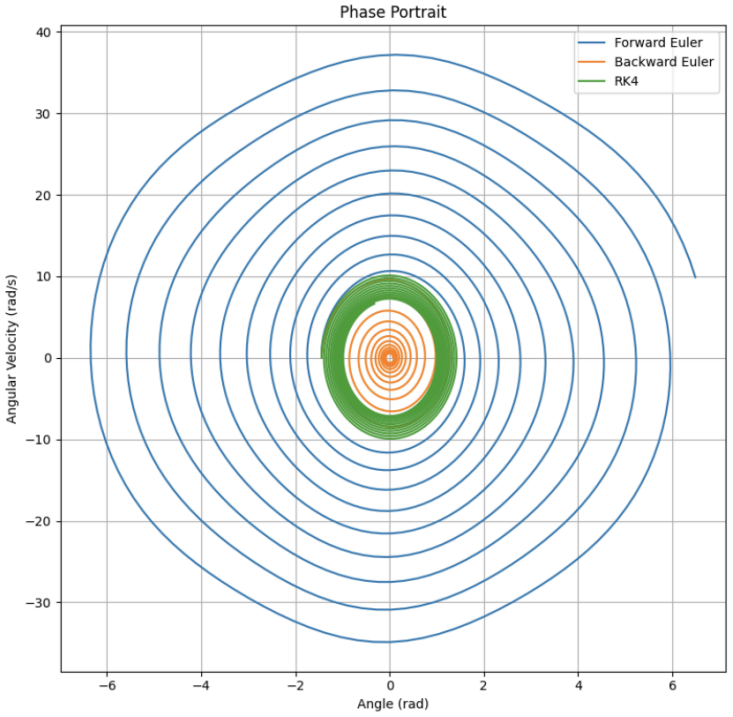

Рисунок 4 - График фазы

После получения результатов, их необходимо проверить. Аналитическое решение выполняется в среде Simulink. Для начала построим схему, которая

будет решать ОДУ и содержать значения, соответствующие варианту:

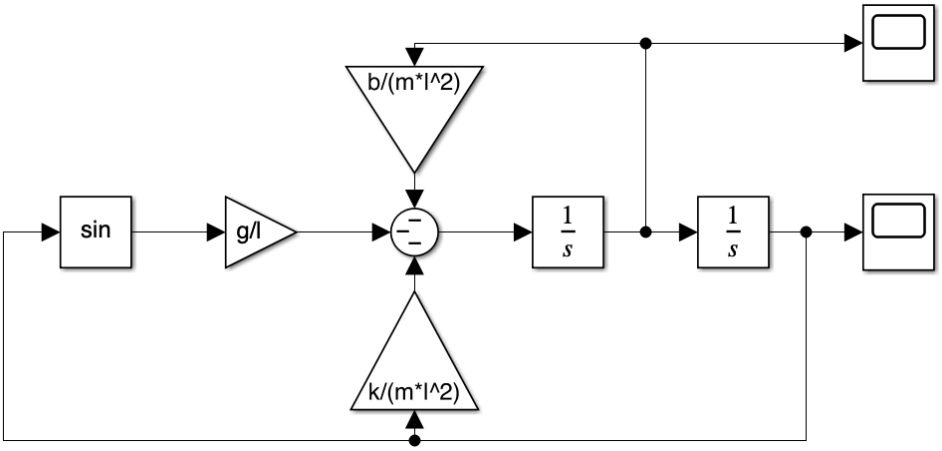

Рисунок 4 - Схема Simulink для решения ОДУ

После построения схемы запустим ее и получим следующие графики:

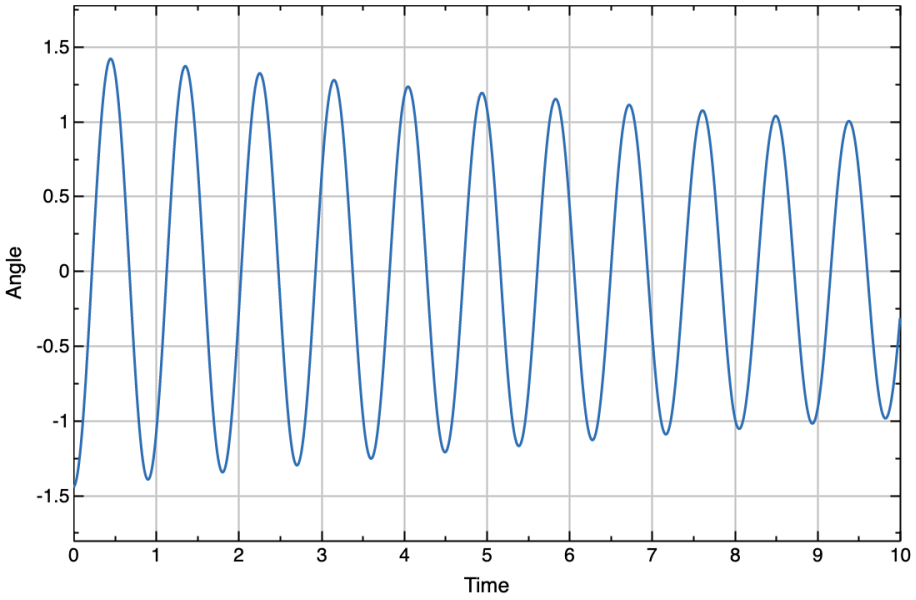

Рисунок 6 - График положения в Simulink

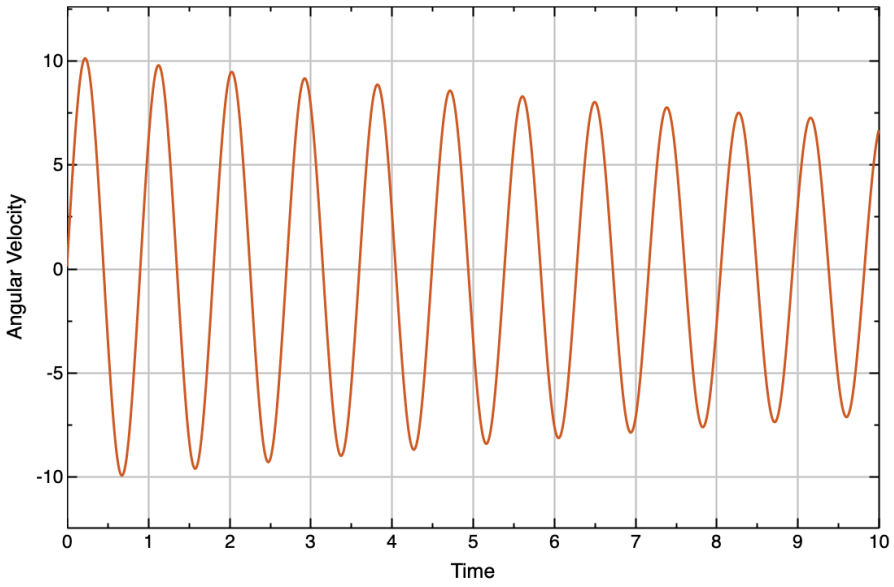

Рисунок 7 - График скорости в Simulink

Вывод

В ходе решения лабораторной работы были рассмотрены три метода решения ОДУ для системы с маятником. Сравнение показало, что методы Эйлера являются менее точными по сравнению с методом Рунге-Кутты, так как явный метод Эйлера имеет возрастающий график из-за зависимости от шага моделирования, а неявный метод Эйлера имеет большую ошибку. По графикам, построенным с помощью среды Simulink, можно проверить достоверность аналитического решения, так как получившиеся графики совпадают с первоначальными, методом Рунге-Кутты, который имеет минимальную ошибку. График фазы в Simulink не построен, так как он показывает зависимость скорости от положения, а значит при совпадении первых двух, он также будет совпадать с решением на Python.

Приложение А

Код Python для решения:

```
import numpy as np

import matplotlib.pyplot as plt
```

```
    def pendulum_dynamics(x):
        """
        Pendulum dynamics: d²θ/dt² = -(g/l) * sin(θ)
        State vector x = [θ, dθ/dt]
        """
        m = 0.9;
        k = 6.4;
        b = 0.015;
        l = 0.47;
        theta_0 = -1.445177505;
        g = 9.81;

        theta = x[0]
        theta_dot = x[1]

         theta_ddot = -(b * theta_dot)/(m * l * l) - g/l * np.sin(theta) -
k/(m * l * l) * theta

        return np.array([theta_dot, theta_ddot])

    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([-1.445177505, 0.0])    # Initial state: [angle,
angular_velocity]
    Tf = 10.0
    h = 0.01

    # Forward Euler
    x_fe, t_fe = forward_euler(pendulum_dynamics, x0, Tf, h)

    # Backward Euler
    x_be, t_be = backward_euler(pendulum_dynamics, x0, Tf, h)

    # Runge-Kutta 4
    x_rk4, t_rk4 = runge_kutta4(pendulum_dynamics, x0, Tf, h)
```

```
    # Plot results
    plt.figure(figsize=(24, 8))


    plt.subplot(1, 3, 1)
    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')
    plt.xlabel('Time')
    plt.ylabel('Angle (rad)')
    plt.legend()
    plt.grid(True)
    plt.title('Pendulum Angle vs Time')


    plt.subplot(1, 3, 2)
    plt.plot(t_fe, x_fe[1, :], label='Forward Euler')
    plt.plot(t_be, x_be[1, :], label='Backward Euler')
    plt.plot(t_rk4, x_rk4[1, :], label='RK4')
    plt.xlabel('Time')
    plt.ylabel('Angular Velocity (rad/s)')
    plt.legend()
    plt.grid(True)
    plt.title('Angular Velocity vs Time')


    plt.subplot(1, 3, 3)
    plt.plot(x_fe[0, :], x_fe[1, :], label='Forward Euler')
    plt.plot(x_be[0, :], x_be[1, :], label='Backward Euler')
    plt.plot(x_rk4[0, :], x_rk4[1, :], label='RK4')
    plt.xlabel('Angle (rad)')
    plt.ylabel('Angular Velocity (rad/s)')
```

```
    plt.legend()

    plt.grid(True)

    plt.title('Phase Portrait')



    plt.tight_layout()

    plt.show()
```