МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО
ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

Лабораторная работа №2

по дисциплине
«Имитационное моделирование робототехнических систем»
Вариант 1

Студент:
Группа R4135c *Шумейко И.В.*

Преподаватель: *Ракшин Е.А.*

Санкт-Петербург 2025

Содержание

Дано

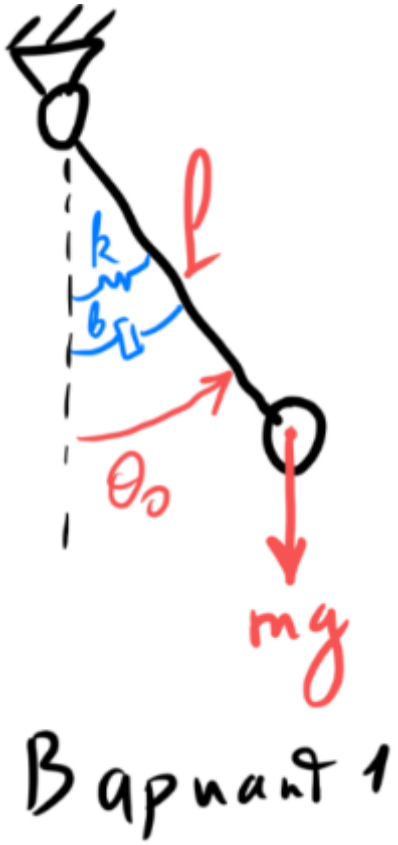

Рис. 1.1: Механическая система

Будем исходить из того, что пружина и демпфер крепятся непосредственно к телу под углом 90°. Изобразим систему согласно предположению, исходящему из заданых параметров:

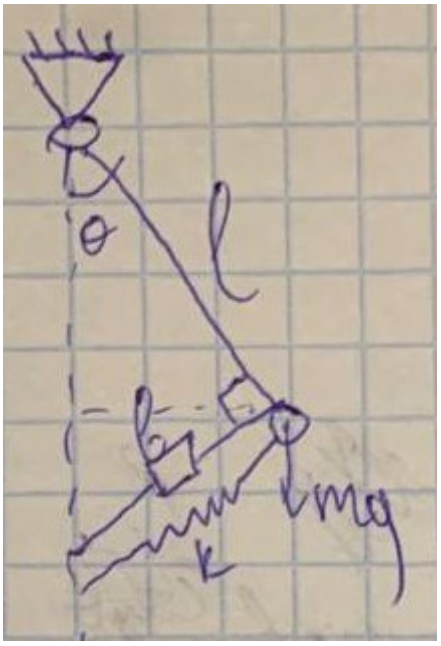

Рис. 1.2: Предполагаемая механическая система

Данные по условию задачи параметры:
$m = 0.3, g = 9.8$ – масса маятника и гравитационная постоянная Земли,
$\theta_0 = 1.508011310084 83$ – начальный угол отклонения маятника от вертикальной оси,
$l = 0.86, l_0 = l \tan \theta$ – длина плеча маятника, а также длина нерастянутой пружины,
$k = 10.6, b = 0.035$ – коэффициент жесткости пружины, коэффициент вязкости дэмпфера

Ход работы

1.1 Математический вывод

Найдем кинетическую и потенциальную энергию маятника:

$$K = \frac{1}{2} I \dot{\theta}^2 = \frac{1}{2} m l^2 \dot{\theta}^2 \tag{1.1}$$

$$P = mgl \cos\theta + \frac{1}{2} k (l \tan\theta - l \tan\theta_0)^2 \tag{1.2}$$

Тогда Лагранжиан системы:

$$\mathcal{L} = K - P = \frac{1}{2} m l^2 \dot{\theta}^2 - mgl \cos\theta - \frac{1}{2} k l^2 (\tan^2\theta - 2 \tan\theta_0 \tan\theta + \tan^2\theta_0) \tag{1.3}$$

Уравнение Эйлера-Лагранжа имеет вид:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right) - \frac{\partial \mathcal{L}}{\partial x} = Q \tag{1.4}$$

Производные Лагранжиана:

$$\frac{\partial \mathcal{L}}{\partial \theta} = mgl \sin\theta - k l^2 \frac{\tan\theta}{\cos^2\theta} + k l^2 \frac{\tan\theta_0}{\cos^2\theta} \tag{1.5}$$

$$\frac{\partial \mathcal{L}}{\partial \dot{\theta}} = m l^2 \dot{\theta} \tag{1.6}$$

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) = m l^2 \ddot{\theta} \tag{1.7}$$

Уравнение Эйлера-Лагранжа принимает вид:

$$m l^2 \ddot{\theta} - mgl \sin\theta + k l^2 \frac{\tan\theta - \tan\theta_0}{\cos^2\theta} = -bl\dot{\theta}, \tag{1.8}$$

или:

$$\ddot{\theta} = -\frac{b}{ml}\dot{\theta} + \frac{g}{l}\sin\theta - \frac{k}{m}\left(\frac{\tan\theta - \tan\theta_0}{\cos^2\theta}\right) \tag{1.9}$$

1.2 Моделирование MatLab

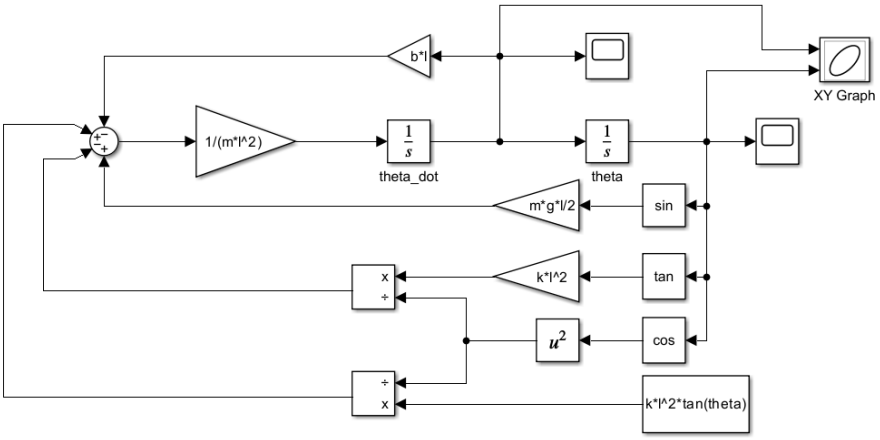

Рис. 1.3: Схема моделирования MatLab

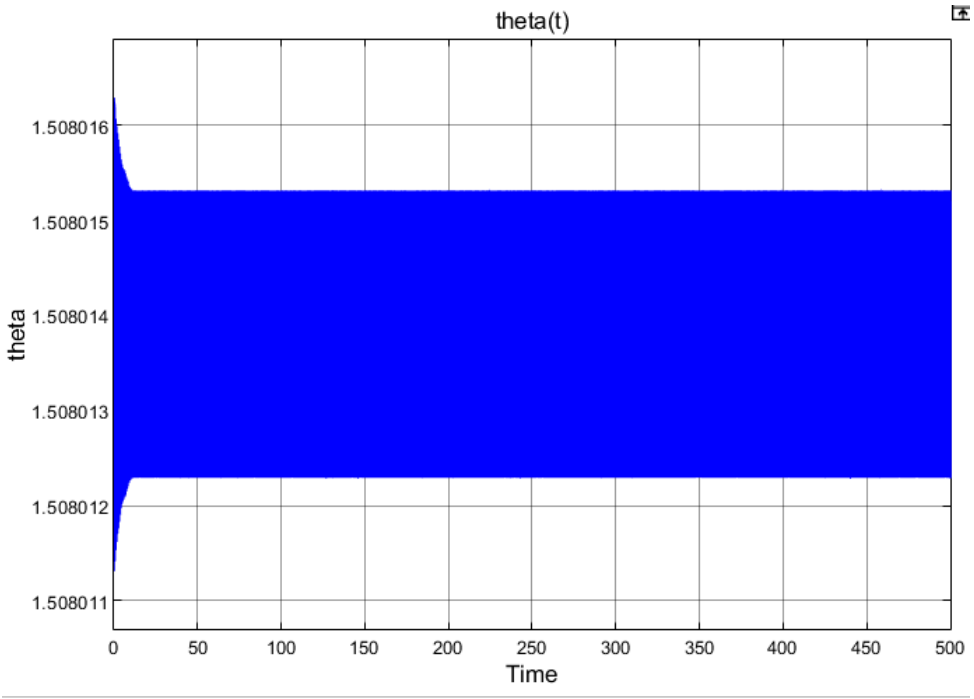

Рис. 1.4: График $\theta(t)$

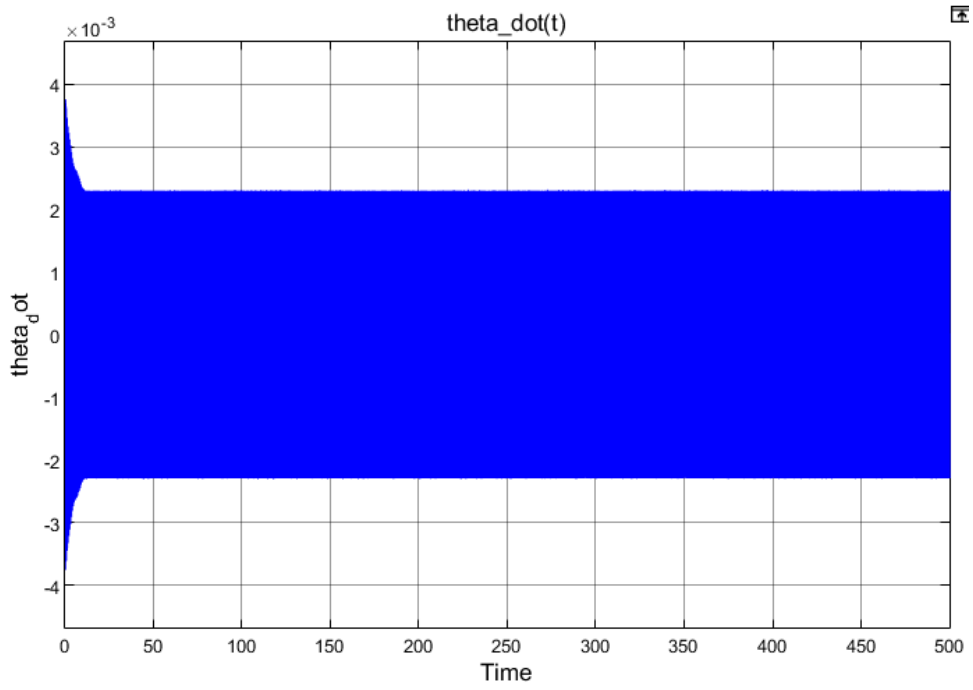

Рис. 1.5: График $\dot{\theta}(t)$

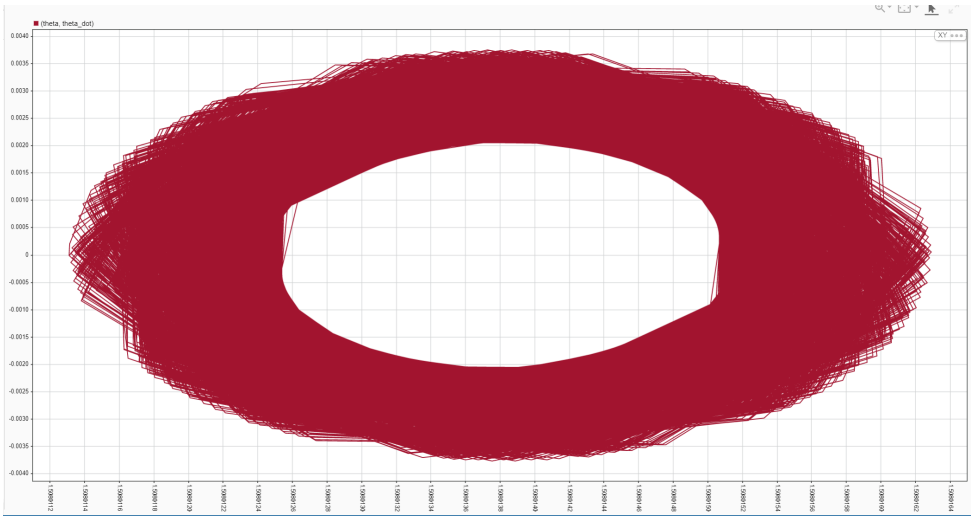

Рис. 1.6: Фазовый портрет системы

1.3 Моделирование численных методов Python

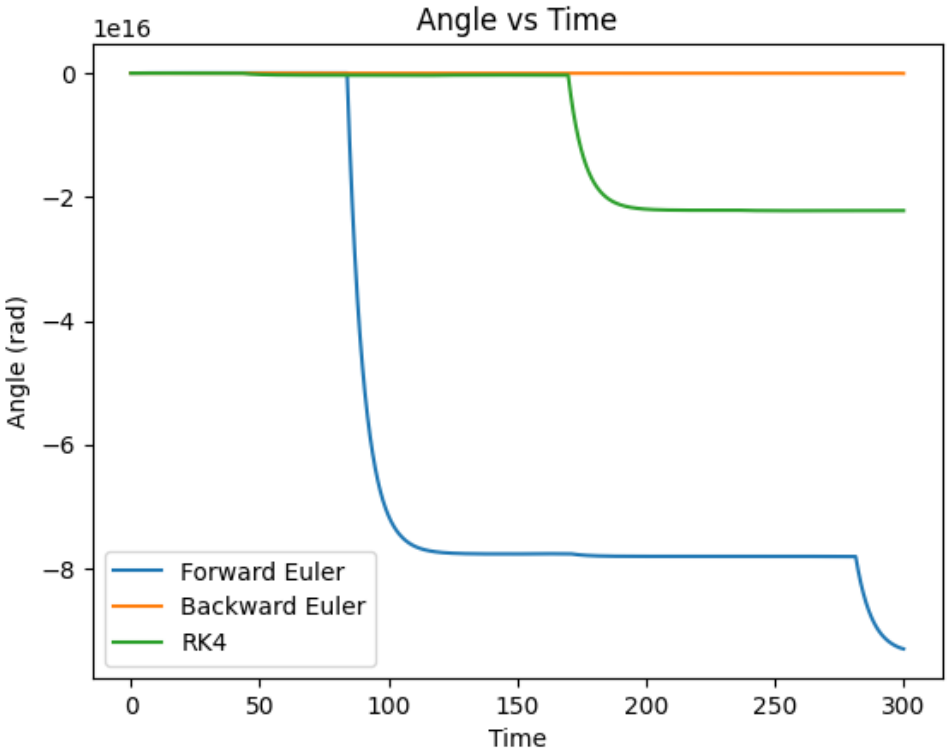

Рис. 1.7: График $\theta(t)$

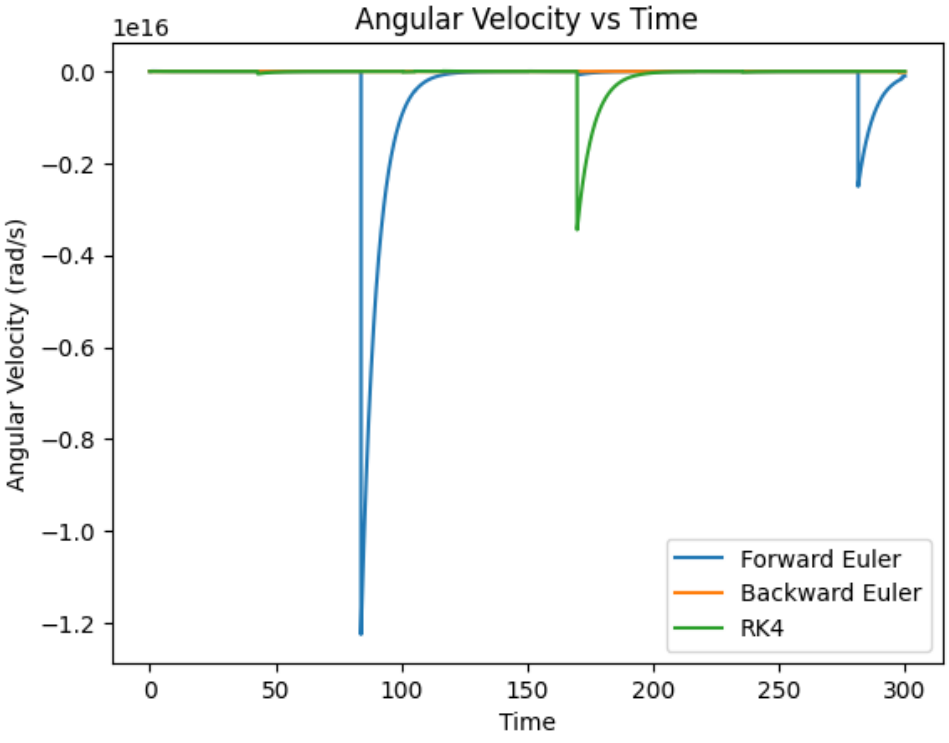

Рис. 1.8: График $\dot{\theta}(t)$

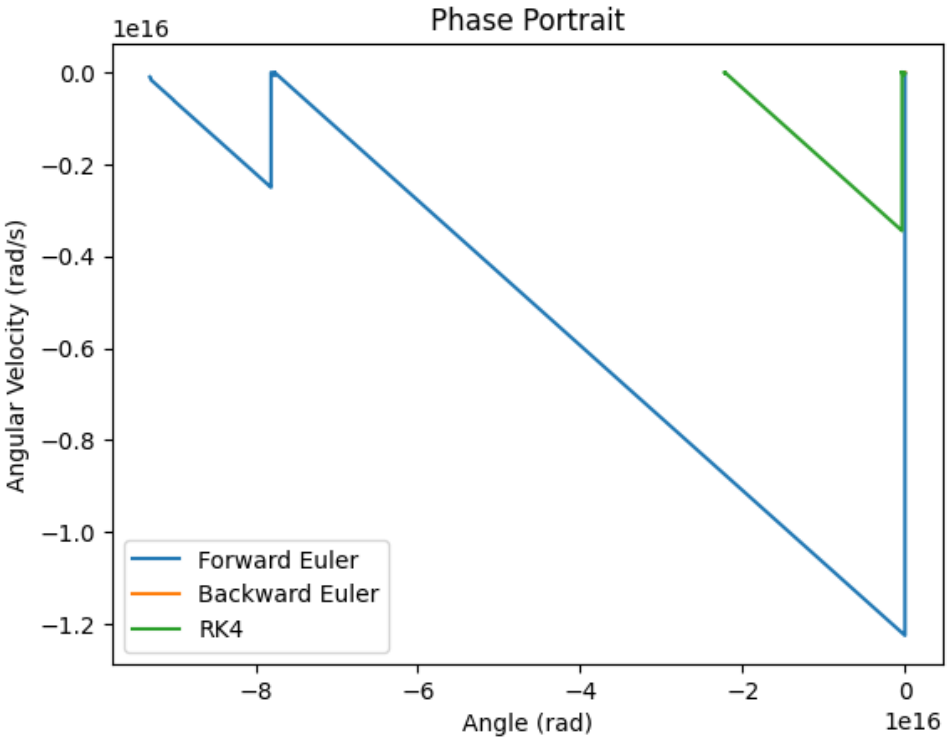

Рис. 1.9: Фазовый портрет системы

Выводы

Поскольку в уравнении Эйлера-Лагранжа θ находится в аргументах тригонометрических функций, а также имеется член с композицией таких функций, то данное ДУ аналитически не решается.

Проведя моделирование в MatLab, можно обнаружить, что система близка к состоянию равновесия с самого начала. Точность вычислений MatLab не позволяет получить $\dot{\theta} = 0$, поэтому маятник колеблется вокруг своего положения равновесия. Если уменьшать шаг дискретизации, то система все больше и больше будет сходится к точке по годографу и к нулю по угловой скорости.

Вычисления в Python не показали вразумительных результатов даже в том случае, если ставить маленькое время моделирование и очень маленький шаг. Это объясняется тем, что, треубемый для качественных графиков данной системы, шаг слишком мал, Python слишком медленен, а алгоритмы аппроксимации слишком простые. Из-за низкой точности методов ошибка копится крайне быстро, и алгоритм "теряет"связь с изначальной системой.

Приложение

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

mass = 0.3; g = 9.8;
theta = 1.50801131008483;

l = 0.86;  l_0 = l*m.tan(theta);
k = 10.6; b = 0.035;

def ode(x):
    """
    State vector x = [theta, theta_dot]
    """
    theta, theta_dot = x[0], x[1]

    theta_ddot = (-b*theta_dot + 1/2*mass*g*l*m.sin(theta) - k*l**2*

    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

    return x_hist, t

# Test all integrators
x0 = np.array([0.1, 0.0])  # Initial state: [angle, angular_velocity
Tf = 300.0
h = 0.001

# 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')

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

plt.show()
```

```
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.title('Angular Velocity vs Time')

plt.show()

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.title('Phase Portrait')

plt.show()
```