

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО
ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

Лабораторная работа №1

по дисциплине

«Имитационное моделирование робототехнических систем»

Вариант 71

Студент:

Группа R4135с

Шумейко И.В.

Преподаватель:

Ракшин Е.А.

Санкт-Петербург 2025

Содержание

Дано	2
Ход работы	3
1.1 Аналитическое решение	3
1.2 Численные методы	4
Выводы	7
Приложение	8

Дано

Однородное дифференциальное уравнение вида:

$$a\ddot{x} + b\dot{x} + cx = d, \quad (1.1)$$

где $a = -0.86$, $b = -6.17$, $c = 1.76$, $d = 6.91$

Ход работы

1.1 Аналитическое решение

Однородное дифференциальное уравнение:

$$-0.86\ddot{x} - 6.17\dot{x} + 1.76x = 6.91 \quad (1.2)$$

Характеристическое уравнение для приведенного дифференциального уравнения:

$$\lambda^2 + 7.174\lambda - 2.047 = 0 \quad (1.3)$$

Корни характеристического уравнения λ_1 и λ_2 :

$$\lambda_1 \approx 0.275 \quad (1.4)$$

$$\lambda_2 \approx -7.449 \quad (1.5)$$

Итак, однородное дифференциальное уравнение в общем виде:

$$x(t) = C_1 e^{0.275t} + C_2 e^{-7.449t}, \quad (1.6)$$

где C_1 и C_2 – константы, определяемые начальными условиями.

Частное решение:

$$\begin{aligned} x_r &= A, \\ 1.76A &= 6.91, \\ A &= 3.926 \end{aligned}$$

Итоговое общее решение однородного дифференциального уравнения:

$$x(t) = C_1 e^{0.275t} + C_2 e^{-7.449t} + 3.926 \quad (1.7)$$

Чтобы определить константы C_1 и C_2 задаются начальные условия:

$$x(0) = 0.1, \quad (1.8)$$

$$\dot{x}(0) = 0. \quad (1.9)$$

Для получения решения с начальными условиями требуется решить систему уравнений:

$$\begin{cases} C_1 + C_2 = -3.826, \\ 0.275C_1 - 7.449C_2 = 0 \end{cases} \quad \begin{cases} C_1 \approx -3.6898 \\ C_2 \approx -0.1362 \end{cases} \quad (1.10)$$

Общее решение с начальными условиями $x(0)$ и $\dot{x}(0)$:

$$x(t) = -3.6898e^{0.275t} + -0.1362e^{-7.449t} + 3.926 \quad (1.11)$$

1.2 Численные методы

В данной работе используются 3 численных метода: явный Эйлер, неявный Эйлер, Рунге-Кутты.

Для сравнения аналитического решения и численных методов были промоделированы все 4 метода и выведены на графики:

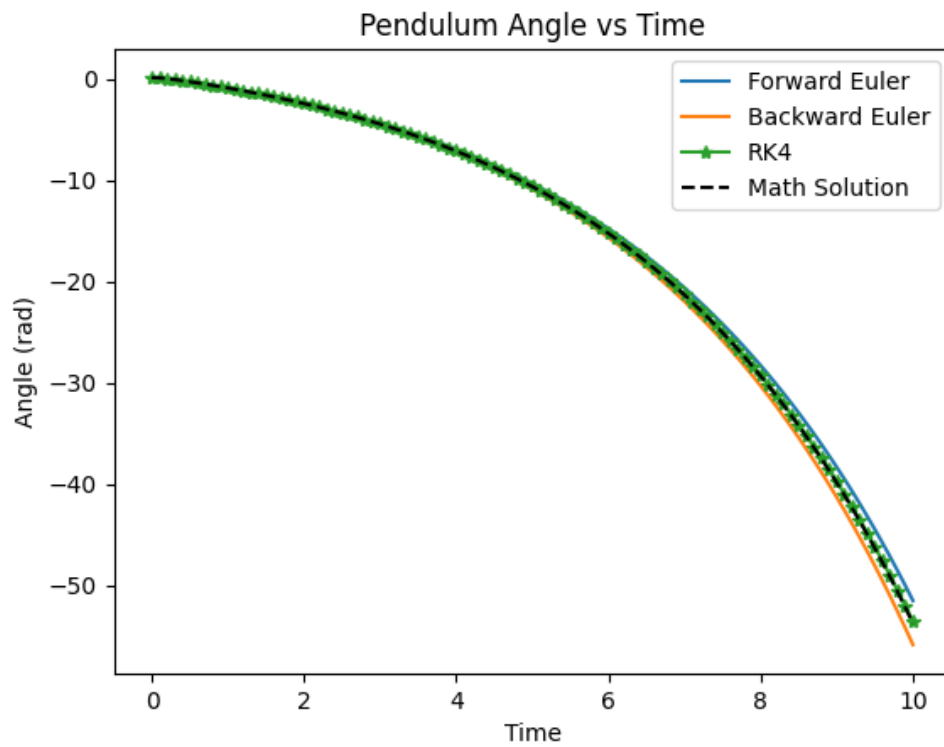


Рис. 1.1: Графики зависимостей положения от времени

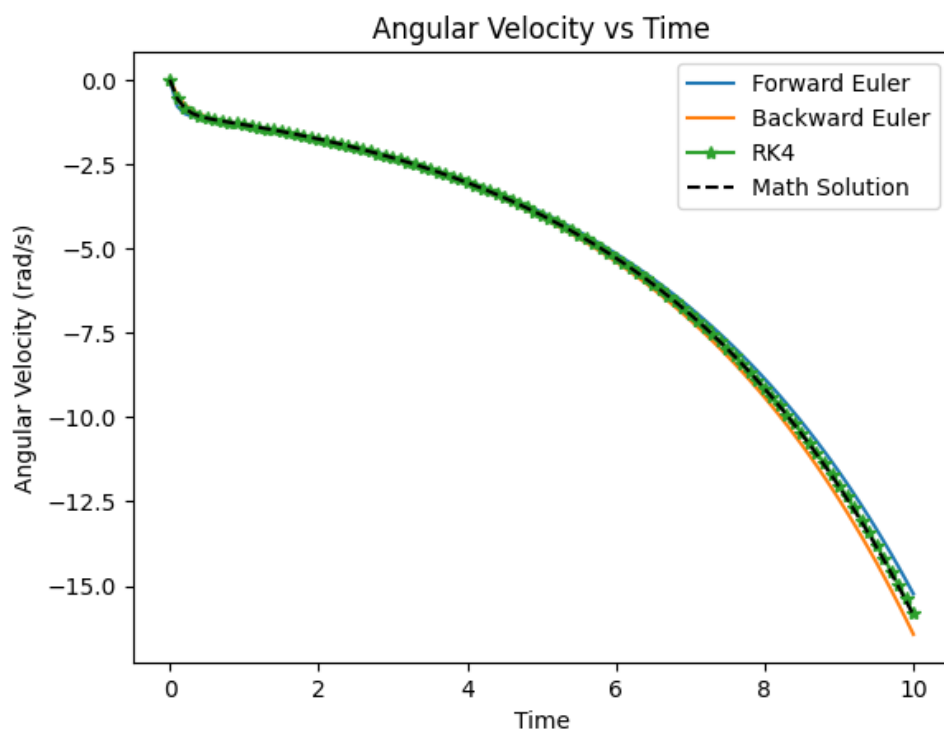


Рис. 1.2: Графики зависимостей скоростей от времени

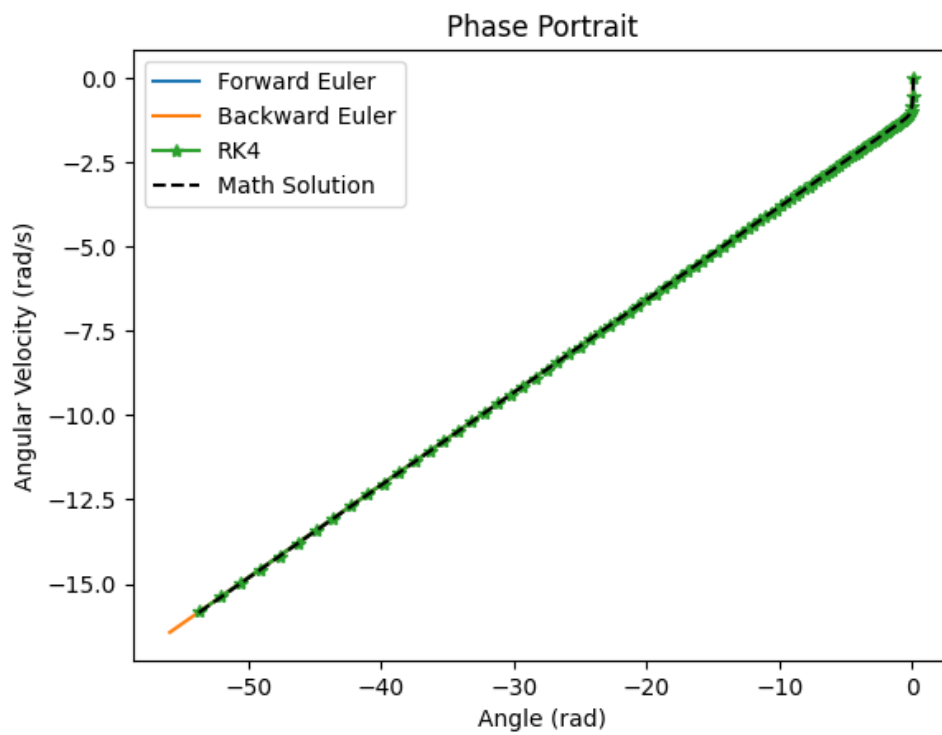


Рис. 1.3: Графики фазовых портретов

Выводы

В ходе работы было произведено сравнение решений однорого дифференциального уравнения аналитическим и численными методами. Сравнения показали, что метод Рунге-Кутты лучше всего совпал с аналитическим решением, что объясняется нелинейной аппроксимацией, используемой в данном решении (а именно, приближением значения интеграла полиномом 3-й степени).

Для методов явного и неявного Эйлера характерен рост ошибки, так как данные методы делают линейную аппроксимацию, из-за чего копится ошибка, а ее модуль на каждом шаге может расти, если система меняется достаточно быстро. Понижение шага h , уменьшает ошибку, однако при увеличении интервала времени, расхождение графиков снова становится заметным.

Несмотря на то, что все три метода имеют ошибку, их фазовые портреты совпадают с фазовым портретом системы, так как сохраняют отношения между x и x' . Ошибки в вычислениях $x(t)$ и $x'(t)$ согласованы. При неправильном вычислении одной, другая подстраивается в соответствии с фазовым портретом системы.

Приложение

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

a, b, c, d = -0.86, -6.17, 1.76, 6.91

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] = [-3.69*m.exp(0.275*t[k]) - 0.136*m.exp(-7.449*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

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', l

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

```
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', marker='*')
plt.plot(t_sol, x_sol[1, :], label='Math Solution', color='black', l
```

```
plt.xlabel('Time')
plt.ylabel('Angular Velocity (rad/s)')
plt.legend()
plt.title('Angular Velocity vs Time')
```

```
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', marker='*')
plt.plot(x_sol[0, :], x_sol[1, :], label='Math Solution', color='bla
```

```
plt.xlabel('Angle (rad)')
plt.ylabel('Angular Velocity (rad/s)')
plt.legend()
plt.title('Phase Portrait')
```

```
plt.show()
```