МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Университет ИТМО

дисциплина

«Имитационное моделирование робототехнических систем»

Отчет по лабораторной работе № 2

Выполнил:
Смирнов И. Д. R4136c

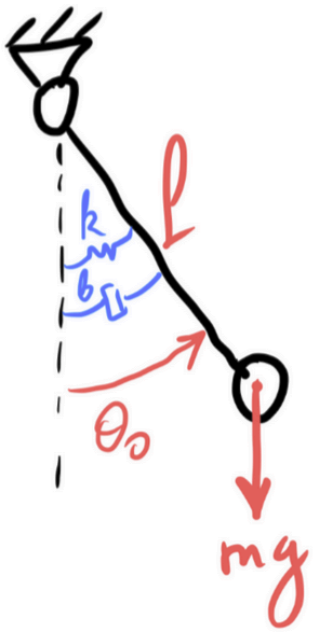
Проверил:
Ракшин Е. А.

Санкт-Петербург

2025

Задание

1) Составить уравнение движения (ОДУ) для системы



m, kg	k, N/m, Nm/rad	b, N*s/m, Nm*s/rad	l, m	θ_0, rad	x_0, m
0.1	2.2	0.035	0.87	-0.9739239711	0.87

2) Решить уравнение аналитически. Если аналитическое решение невозможно, описать причину и объяснить почему.

3) Сравнить результаты численных методов с аналитическим решением (если оно существует) и сделать выводы.

Ход работы

1.Уравнение движения

Лагранжиан:

$L = K - P$

$K = \frac{1}{2}m\omega^2 = \frac{1}{2}ml^2\theta^2$ $\quad$ $P = m \cdot g \cdot l \cdot cos(\theta) + \frac{1}{2}k\theta^2$

Метод Эйлера-Лагранжа:

$\frac{d}{dx}(\frac{\partial L}{\partial \dot{x}}) - \frac{\partial L}{\partial x} = Q$ где $Q = - b\dot{\theta}$

Выполним подстановку:

$ml^2\ddot{\theta} + m \cdot g \cdot l \cdot sin(\theta) + k\theta + b\dot{\theta}$

Тогда итоговое уравнение будет иметь вид

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

---

2. Аналитическое решение

Решить уравнение аналитически невозможно. Основной причиной является нелинейный член $sin(\theta)$. Однако данная модель может быть линеаризованная, если мы примем, что при малых углах $sin(\theta) = \theta$. Однако, начальное условие x_0= −0.9739рад (что примерно −55.8) не является малым углом, поэтому, аналитического решения быть не может.

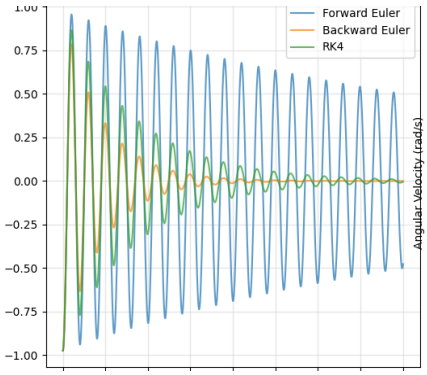
---

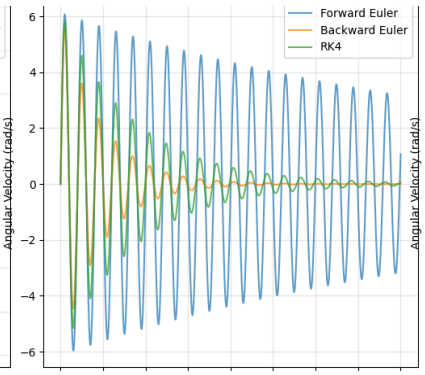
3. Эксперимент:

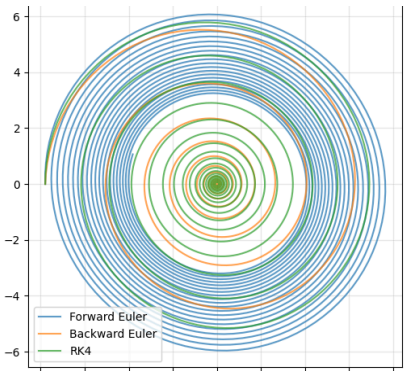
Параметры эксперимента:

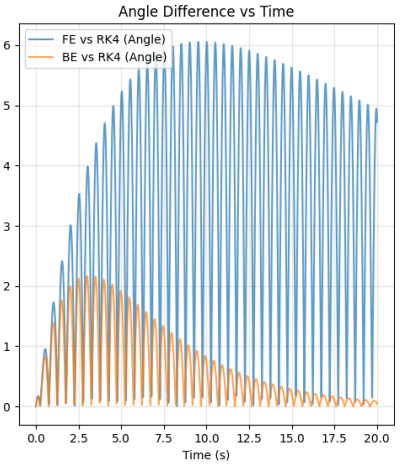
```
# параметры моделирования
Tf = 20.0 # время моделирования
h = 0.1 # шаг интегрирования
```

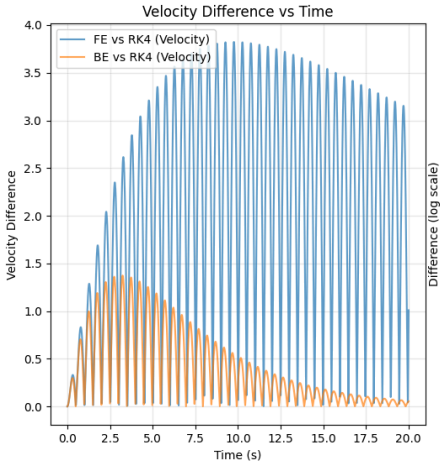
Результат:

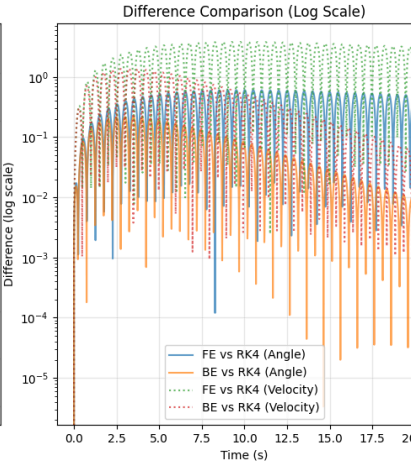












Статистика ошибок:

Difference Statistics (FE vs RK4):
Angle - Max diff: 0.605484, Mean diff: 0.322302
Velocity - Max diff: 3.823217, Mean diff: 2.015516

Difference Statistics (BE vs RK4):
Angle - Max diff: 0.217020, Mean diff: 0.060292
Velocity - Max diff: 1.372602, Mean diff: 0.380782

Вывод

Для нелинейного уравнения движения системы "масса-пружина-демпфер" аналитическое решение невозможно из-за нелинейного члена $\sin(\theta)$. При линеаризации

для малых углов аналитическое решение возможно, но в данной работе оно не применимо, так как начальное условие $\theta_0 = -0.9739$ рад не является малым.

Численное моделирование с использованием методов явного Эйлера, неявного Эйлера и Рунге-Кутты 4-го порядка показало качественно правильное поведение системы (затухающие колебания). Сравнение методов между собой показало, что неявный метод Эйлера продемонстрировал лучшую точность и меньшее накопление ошибки по сравнению с явным методом Эйлера, что видно из статистики разниц с методом РК4. Метод Рунге-Кутты 4-го порядка показал наивысшую стабильность и точность.

Для решения нелинейных дифференциальных уравнений, возникающих в задачах моделирования робототехнических систем, предпочтительно использовать методы более высокого порядка, такие как RK4, или неявные методы, которые обеспечивают лучшую устойчивость. Полученные результаты подтверждают важность выбора численного метода в зависимости от требований к точности и вычислительным ресурсам

Листинги программ:

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

def oscillator_dynamics(x):
    """
    Dynamics for the mass-spring-damper system (Variant 1)
    State vector x = [theta, theta_dot]
    Equation: theta_ddot = -(b/m) * theta_dot - (g/l) * sin(theta) - (k/m) * sin(theta) * cos(theta)
    """
    m = 0.1
    k = 2.2
    b = 0.035
    l = 0.87
    g = 9.81

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

    theta_ddot = -(b/(m*l**2)) * theta_dot - (g/l) * np.sin(theta) - (k/(m*l**2)) * 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]

        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


m = 0.1
k = 2.2
b = 0.035
l = 0.87
g = 9.81


theta0 = -0.9739239711
theta_dot0 = 0.0
x0 = np.array([theta0, theta_dot0])

# Time parameters
Tf = 20.0
h = 0.01

x_fe, t_fe = forward_euler(oscillator_dynamics, x0, Tf, h)
x_be, t_be = backward_euler(oscillator_dynamics, x0, Tf, h)
x_rk4, t_rk4 = runge_kutta4(oscillator_dynamics, x0, Tf, h)
```

```python
# Plotting
plt.figure(figsize=(24, 8))

# Plot 1: Position vs Time
plt.subplot(1, 3, 1)
plt.plot(t_fe, x_fe[0, :], label='Forward Euler', alpha=0.7)
plt.plot(t_be, x_be[0, :], label='Backward Euler', alpha=0.7)
plt.plot(t_rk4, x_rk4[0, :], label='RK4', alpha=0.7)
plt.xlabel('Time (s)')
plt.ylabel('Angle (rad)')
plt.legend()
plt.title('Angle vs Time')
plt.grid(True, alpha=0.3)

# Plot 2: Velocity vs Time
plt.subplot(1, 3, 2)
plt.plot(t_fe, x_fe[1, :], label='Forward Euler', alpha=0.7)
plt.plot(t_be, x_be[1, :], label='Backward Euler', alpha=0.7)
plt.plot(t_rk4, x_rk4[1, :], label='RK4', alpha=0.7)
plt.xlabel('Time (s)')
plt.ylabel('Angular Velocity (rad/s)')
plt.legend()
plt.title('Angular Velocity vs Time')
plt.grid(True, alpha=0.3)

# Plot 3: Phase Portrait
plt.subplot(1, 3, 3)
plt.plot(x_fe[0, :], x_fe[1, :], label='Forward Euler', alpha=0.7)
plt.plot(x_be[0, :], x_be[1, :], label='Backward Euler', alpha=0.7)
plt.plot(x_rk4[0, :], x_rk4[1, :], label='RK4', alpha=0.7)
plt.xlabel('Angle (rad)')
plt.ylabel('Angular Velocity (rad/s)')
plt.legend()
plt.title('Phase Portrait')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()


def interpolate_solution(t_num, x_num, t_target):
    """Interpolate numerical solution to match target time grid"""
    x_interp = np.interp(t_target, t_num, x_num)
    return x_interp

t_rk4_interp = t_rk4
x_fe_angle_interp = interpolate_solution(t_fe, x_fe[0, :], t_rk4_interp)
x_be_angle_interp = interpolate_solution(t_be, x_be[0, :], t_rk4_interp)

x_fe_vel_interp = interpolate_solution(t_fe, x_fe[1, :], t_rk4_interp)
x_be_vel_interp = interpolate_solution(t_be, x_be[1, :], t_rk4_interp)


diff_angle_fe_rk4 = np.abs(x_fe_angle_interp - x_rk4[0, :])
diff_angle_be_rk4 = np.abs(x_be_angle_interp - x_rk4[0, :])

diff_vel_fe_rk4 = np.abs(x_fe_vel_interp - x_rk4[1, :])
diff_vel_be_rk4 = np.abs(x_be_vel_interp - x_rk4[1, :])

print("Difference Statistics (FE vs RK4):")
```

```
print(f"Angle - Max diff: {np.max(diff_angle_fe_rk4):.6f}, Mean diff:
{np.mean(diff_angle_fe_rk4):.6f}")
print(f"Velocity - Max diff: {np.max(diff_vel_fe_rk4):.6f}, Mean diff:
{np.mean(diff_vel_fe_rk4):.6f}")

print("\nDifference Statistics (BE vs RK4):")
print(f"Angle - Max diff: {np.max(diff_angle_be_rk4):.6f}, Mean diff:
{np.mean(diff_angle_be_rk4):.6f}")
print(f"Velocity - Max diff: {np.max(diff_vel_be_rk4):.6f}, Mean diff:
{np.mean(diff_vel_be_rk4):.6f}")

# Plot differences
plt.figure(figsize=(18, 6))

plt.subplot(1, 3, 1)
plt.plot(t_rk4_interp, diff_angle_fe_rk4, label='FE vs RK4 (Angle)', alpha=0.7)
plt.plot(t_rk4_interp, diff_angle_be_rk4, label='BE vs RK4 (Angle)', alpha=0.7)
plt.xlabel('Time (s)')
plt.ylabel('Angle Difference')
plt.legend()
plt.title('Angle Difference vs Time')
plt.grid(True, alpha=0.3)

plt.subplot(1, 3, 2)
plt.plot(t_rk4_interp, diff_vel_fe_rk4, label='FE vs RK4 (Velocity)', alpha=0.7)
plt.plot(t_rk4_interp, diff_vel_be_rk4, label='BE vs RK4 (Velocity)', alpha=0.7)
plt.xlabel('Time (s)')
plt.ylabel('Velocity Difference')
plt.legend()
plt.title('Velocity Difference vs Time')
plt.grid(True, alpha=0.3)

plt.subplot(1, 3, 3)
plt.semilogy(t_rk4_interp, diff_angle_fe_rk4, label='FE vs RK4 (Angle)', alpha=0.7)
plt.semilogy(t_rk4_interp, diff_angle_be_rk4, label='BE vs RK4 (Angle)', alpha=0.7)
plt.semilogy(t_rk4_interp, diff_vel_fe_rk4, label='FE vs RK4 (Velocity)', alpha=0.7, linestyle=':')
plt.semilogy(t_rk4_interp, diff_vel_be_rk4, label='BE vs RK4 (Velocity)', alpha=0.7, linestyle=':')
plt.xlabel('Time (s)')
plt.ylabel('Difference (log scale)')
plt.legend()
plt.title('Difference Comparison (Log Scale)')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```