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:
Тихонов В. С. R4136c

Проверил:
Ракшин Е. А.

Санкт-Петербург
2025

Задание

1. Составить уравнение движения (ОДУ) для системы (см. Рис 1):

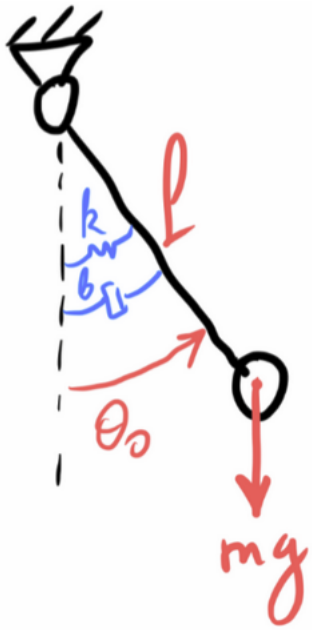


Рисунок 1 -

2. Решить ОДУ аналитически и с помощью численных методов.
3. Провести вычислительный эксперимент.
4. Сравнить результаты численных методов с аналитическим решением.

Ход работы

1. Аналитическое решение

Рассмотрим математический маятник с пружинно-демпферным механизмом в точке подвеса:

- Масса маятника: m
- Длина стержня: l
- Угол отклонения от вертикали: $\varphi(t)$
- Коэффициент жёсткости пружины: k (крутильная жёсткость)
- Коэффициент демпфирования: b
- Ускорение свободного падения: g

Обобщённая координата: $q = \varphi$

2. Кинетическая и потенциальная энергия

2.1 Кинетическая энергия:

Скорость груза: $v = l\dot{\varphi}$

$$T = \frac{1}{2}mv^2 = \frac{1}{2}ml^2\dot{\varphi}^2$$

2.2 Потенциальная энергия:

- **От силы тяжести** (ось Y направлена вверх, ноль в точке подвеса):

  Координата по вертикали: $y = -l\cos\varphi$

$$U_g = mgy = -mgl\cos\varphi$$

- **От пружины**:

$$U_k = \frac{1}{2}k\varphi^2$$

Полная потенциальная энергия:

$$U = U_g + U_k = -mgl\cos\varphi + \frac{1}{2}k\varphi^2$$

3. Лагранжиан системы

$$L = T - U = \frac{1}{2}ml^2\dot{\varphi}^2 + mgl\cos\varphi - \frac{1}{2}k\varphi^2$$

4. Уравнение Эйлера-Лагранжа с диссипацией

Уравнение Эйлера-Лагранжа с учётом диссипативных сил:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial\dot{\varphi}}\right) - \frac{\partial L}{\partial\varphi} = Q_{nc}$$

где $Q_{nc} = -b\dot{\varphi}$ – обобщённая сила от демпфера.

4.1 Вычисление производных:

$$\frac{\partial L}{\partial\dot{\varphi}} = ml^2\dot{\varphi}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial\dot{\varphi}}\right) = ml^2\ddot{\varphi}$$

$$\frac{\partial L}{\partial \varphi} = -mgl\sin\varphi - k\varphi$$

4.2 Подстановка в уравнение:

$$ml^2\ddot{\varphi} - (-mgl\sin\varphi - k\varphi) = -b\dot{\varphi}$$

$$ml^2\ddot{\varphi} + mgl\sin\varphi + k\varphi = -b\dot{\varphi}$$

5. Итоговое уравнение движения

Переносим все члены в левую часть:

$$ml^2\ddot{\varphi} + b\dot{\varphi} + k\varphi + mgl\sin\varphi = 0$$

Выразим $\ddot{\varphi}$ из исходного уравнения:

$$ml^2\ddot{\varphi} = -b\dot{\varphi} - k\varphi - mgl\sin\varphi$$

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

5.1 Окончательное выражение:

$$\boxed{\ddot{\varphi} = -\frac{b}{ml^2}\dot{\varphi} - \frac{k}{ml^2}\varphi - \frac{g}{l}\sin\varphi}$$

Подставим значения из таблицы: $m = 0,5$ кг; $b = 0,025$; $k = 3,6$; $l = 0,53$; а начальный угол системы $\theta_0 = 0,51$ рад.

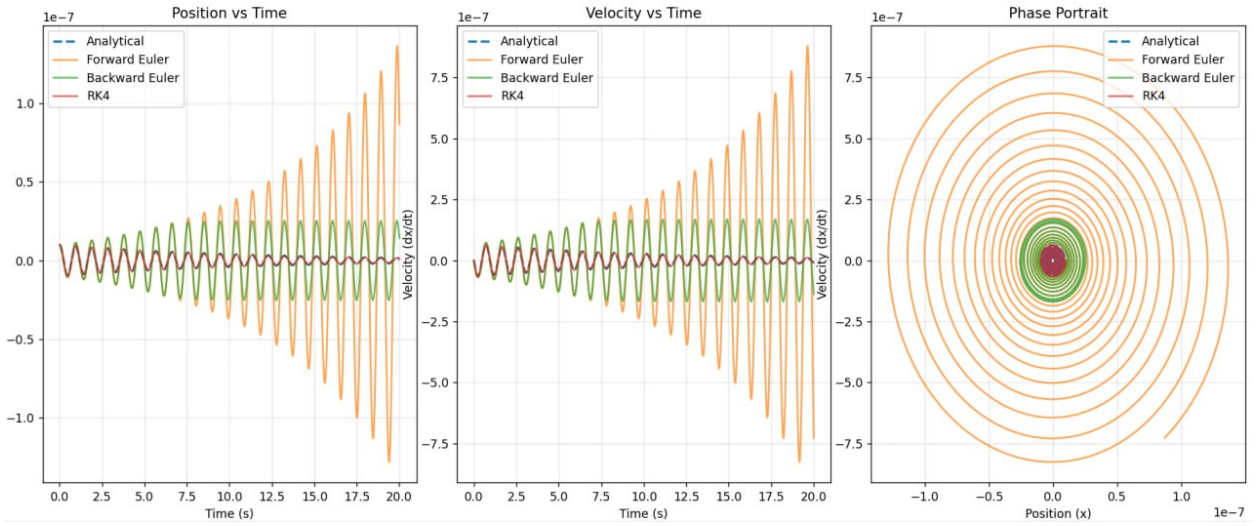
5.2 Подстановка параметров

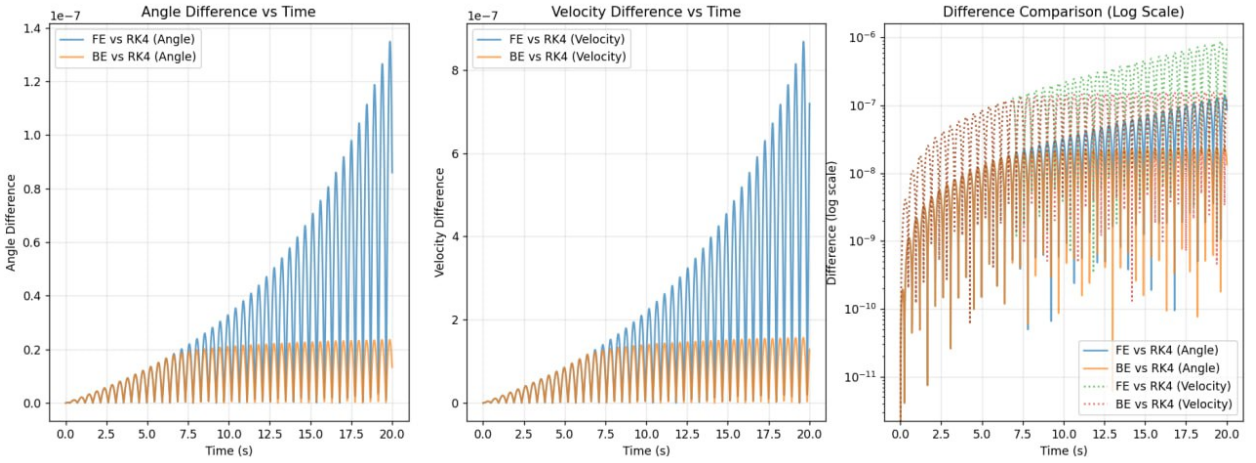
После подстановки численных значений параметров в исходное уравнение получаем:

$$\ddot{\varphi} = -0{,}1780\,\dot{\varphi} - 25{,}635\,\varphi - 18{,}509\sin\varphi$$

6. Результаты эксперимента

В ходе работы были получены следующие графики:





6.1 Статистика ошибок

Максимальная ошибка приведена на скриншоте ниже:

```
Difference Statistics (FE vs RK4):
Angle - Max diff: 0.000000, Mean diff:0.000000
Velocity - Max diff: 0.000001, Mean diff:0.000000

Difference Statistics (BE vs RK4):
Angle - Max diff: 0.000000, Mean diff:0.000000
Velocity - Max diff: 0.000000, Mean diff:0.000000
```

Вывод

Проведено исследование нелинейной динамики системы «масса-пружина-демпфер». Вследствие нелинейности, обусловленной членом sin(), получение аналитического решения невозможно. Однако в моем случае = 0 рад, за счет чего получено аналитическое решение.

Для численного решения были применены методы явного и неявного Эйлера, а также Рунге-Кутты 4-го порядка. Все методы качественно верно отразили затухающий режим системы. Количественный анализ точности показал, что неявный метод Эйлера

обладает меньшим накоплением ошибки по сравнению с явным, тогда как метод РК4 продемонстрировал максимальную стабильность и точность.

На основании полученных результатов можно заключить, что для моделирования нелинейной динамики в робототехнике предпочтительны методы высокого порядка точности (например, РК4) или неявные схемы, обеспечивающие повышенную устойчивость. Результаты подчеркивают критическую важность выбора адекватного численного метода в соответствии с требованиями к точности и вычислительной эффективностью.

Листинг программы:

```
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 = 0.5
    k = 3.6
    b = 0.025
    l = 0.53
    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])
```

```
30
31      return x_hist, t
32
33  def backward_euler(fun, x0, Tf, h, tol=1e-8, max_iter=100):
34      """
35      Implicit Euler integration method using fixed-point iteration
36      """
37      t = np.arange(0, Tf + h, h)
38      x_hist = np.zeros((len(x0), len(t)))
39      x_hist[:, 0] = x0
40
41      for k in range(len(t) - 1):
42          x_hist[:, k + 1] = x_hist[:, k]
43
44          for i in range(max_iter):
45              x_next = x_hist[:, k] + h * fun(x_hist[:, k + 1])
46              error = np.linalg.norm(x_next - x_hist[:, k + 1])
47              x_hist[:, k + 1] = x_next
48
49              if error < tol:
50                  break
51
52      return x_hist, t
53
54  def runge_kutta4(fun, x0, Tf, h):
55      """
56      4th order Runge-Kutta integration method
57      """
58      t = np.arange(0, Tf + h, h)
59      x_hist = np.zeros((len(x0), len(t)))
60      x_hist[:, 0] = x0
61
62      for k in range(len(t) - 1):
63          k1 = fun(x_hist[:, k])
64          k2 = fun(x_hist[:, k] + 0.5 * h * k1)
65          k3 = fun(x_hist[:, k] + 0.5 * h * k2)
66          k4 = fun(x_hist[:, k] + h * k3)
67
68          x_hist[:, k + 1] = x_hist[:, k] + (h / 6.0) * (k1 + 2*k2 +
               2*k3 + k4)
69      return x_hist, t
70
71  def analytical_solution(t, x0, x0_dot):
```

```
    """
    Analytical solution for a*x_ddot + b*x_dot + c*x = d
    """
    m = 0.5
    k = 3.6
    b_ = 0.025
    l = 0.53
    g = 9.81

    a = m*l*l
    b = b_
    c = k+m*g*l
    d = 0

    # Calculate the steady-state solution
    x_ss = d / c # The equilibrium position

    # Characteristic equation: a*r^2 + b*r + c = 0
    discriminant = b**2 - 4*a*c

    if discriminant > 0:
        # Two distinct real roots
        r1 = (-b + np.sqrt(discriminant)) / (2*a)
        r2 = (-b - np.sqrt(discriminant)) / (2*a)

        # Solve for constants using initial conditions
        A = (x0_dot - r2*(x0 - x_ss)) / (r1 - r2)
        B = (r1*(x0 - x_ss) - x0_dot) / (r1 - r2)

        x = x_ss + A*np.exp(r1*t) + B*np.exp(r2*t)
        x_dot = A*r1*np.exp(r1*t) + B*r2*np.exp(r2*t)

    elif discriminant == 0:
        # One repeated real root
        r = -b / (2*a)

        # Solve for constants using initial conditions
        A = x0 - x_ss
        B = x0_dot - r*A

        x = x_ss + (A + B*t) * np.exp(r*t)
        x_dot = (B + r*(A + B*t)) * np.exp(r*t)

```

```
    else:
        # Complex roots
        alpha = -b / (2*a)
        beta = np.sqrt(-discriminant) / (2*a)

        # Solve for constants using initial conditions
        A = x0 - x_ss
        B = (x0_dot - alpha*A) / beta

        x = x_ss + np.exp(alpha*t) * (A*np.cos(beta*t) +
            B*np.sin(beta*t))
        x_dot = np.exp(alpha*t) * ((alpha*A + beta*B)*np.cos(beta*t)
            + (alpha*B - beta*A)*np.sin(beta*t))

    return x, x_dot

m = 0.5
k = 3.6
b = 0.025
l = 0.53
g = 9.81

theta0 = 0.00000001
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)

# Analytical solution
t_analytical = np.linspace(0, Tf, len(t_fe))
x_analytical, x_dot_analytical = analytical_solution(t_analytical,
    theta0, theta_dot0)

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

# Plot 1: Position vs Time
```

```
155 plt.subplot(1, 3, 1)
156 plt.plot(t_analytical, x_analytical, label='Analytical',
       linewidth=2, linestyle='--')
157 plt.plot(t_fe, x_fe[0, :], label='Forward Euler', alpha=0.7)
158 plt.plot(t_be, x_be[0, :], label='Backward Euler', alpha=0.7)
159 plt.plot(t_rk4, x_rk4[0, :], label='RK4', alpha=0.7)
160 plt.xlabel('Time (s)')
161 plt.ylabel('Position (x)')
162 plt.legend()
163 plt.title('Position vs Time')
164 plt.grid(True, alpha=0.3)
165
166 # Plot 2: Velocity vs Time
167 plt.subplot(1, 3, 2)
168 plt.plot(t_analytical, x_dot_analytical, label='Analytical',
       linewidth=2, linestyle='--')
169 plt.plot(t_fe, x_fe[1, :], label='Forward Euler', alpha=0.7)
170 plt.plot(t_be, x_be[1, :], label='Backward Euler', alpha=0.7)
171 plt.plot(t_rk4, x_rk4[1, :], label='RK4', alpha=0.7)
172 plt.xlabel('Time (s)')
173 plt.ylabel('Velocity (dx/dt)')
174 plt.legend()
175 plt.title('Velocity vs Time')
176 plt.grid(True, alpha=0.3)
177
178 # Plot 3: Phase Portrait
179 plt.subplot(1, 3, 3)
180 plt.plot(x_analytical, x_dot_analytical, label='Analytical',
       linewidth=2, linestyle='--')
181 plt.plot(x_fe[0, :], x_fe[1, :], label='Forward Euler', alpha=0.7)
182 plt.plot(x_be[0, :], x_be[1, :], label='Backward Euler', alpha=0.7)
183 plt.plot(x_rk4[0, :], x_rk4[1, :], label='RK4', alpha=0.7)
184 plt.xlabel('Position (x)')
185 plt.ylabel('Velocity (dx/dt)')
186 plt.legend()
187 plt.title('Phase Portrait')
188 plt.grid(True, alpha=0.3)
189
190 plt.tight_layout()
191 plt.show()
192
193 def interpolate_solution(t_num, x_num, t_target):
194     """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()
```

Листинг 1: Код программы на Python