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 №1

Выполнил:
Фомин В. М. R4134c

Проверил:
Ракшин Е. А.

Санкт-Петербург
2025

Задание

1. Решить аналитически обыкновенное дифференциальное уравнение (ОДУ) в виде:

   $a \cdot \ddot{x} + b \cdot \dot{x} + c \cdot x = d$

   где $a = 5.12$, $b = 6.59$, $c = 8.23$, $d = 7.35$

2. Решить ОДУ с помощью трех численных методов: явного и неявного методов Эйлера, метода Рунге-Кутты 4-го порядка.

3. Провести вычислительный эксперимент.

4. Сравнить результаты численных методов с аналитическим решением.

Ход работы

1. Аналитическое решение

1.1. Постановка задачи

Исходное дифференциальное уравнение:

$$a \cdot \ddot{x} + b \cdot \dot{x} + c \cdot x = d$$

где $a = 5.12$, $b = 6.59$, $c = 8.23$, $d = 7.35$

1.2. Характеристическое уравнение

Соответствующее характеристическое уравнение:

$$5.12 \cdot \lambda^2 + 6.59 \cdot \lambda + 8.23 = 0$$

1.3. Дискриминант и корни

Вычислим дискриминант:

$$D = b^2 - 4ac = 6.59^2 - 4 \cdot 5.12 \cdot 8.23 = 43.4281 - 168.5504 = -125.1223$$

Корни характеристического уравнения:

$$\lambda_{1,2} = \frac{-b \pm \sqrt{D}}{2a} = \frac{-6.59 \pm \sqrt{-125.1223}}{2 \cdot 5.12}$$

$$\lambda_1 = -0.643 + 1.546i, \quad \lambda_2 = -0.643 - 1.546i$$

1.4. Общее решение однородного уравнения

Общее решение однородного уравнения:

$$x_h(t) = C_1 e^{(-0.643+1.546i)t} + C_2 e^{(-0.643-1.546i)t}$$

1.5. Частное решение неоднородного уравнения

Частное решение находим методом неопределенных коэффициентов:

$$c \cdot x_p = d \Rightarrow x_p = \frac{d}{c} = \frac{7.35}{8.23} = 0.893$$

1.6. Общее решение неоднородного уравнения

Общее решение:

$$x(t) = x_h(t) + x_p(t) = C_1 e^{(-0.643+1.546i)t} + C_2 e^{(-0.643-1.546i)t} + 0.893$$

1.7. Тригонометрическая форма решения

Преобразуем к тригонометрической форме:

$$x(t) = e^{-0.643t}(A\cos(1.546t) + B\sin(1.546t)) + 0.893$$

где $A = C_1 + C_2$, $B = i(C_1 - C_2)$

1.8. Определение постоянных интегрирования

Используем начальные условия: $x(0) = 0.1$, $\dot{x}(0) = 0$
Из первого начального условия:

$$x(0) = e^0(A\cos 0 + B\sin 0) + 0.893 = A + 0.893 = 0.1$$

$$A = 0.1 - 0.893 = -0.793$$

Находим производную:

$$\begin{aligned} x'(t) = &-0.643e^{-0.643t}(A\cos 1.546t + B\sin 1.546t) \\ &+ e^{-0.643t}(-1.546A\sin 1.546t + 1.546B\cos 1.546t) \end{aligned}$$

Из второго начального условия:

$$x'(0) = -0.643A + 1.546B = 0$$

$$-0.643 \cdot (-0.793) + 1.546B = 0$$

$$0.509 + 1.546B = 0$$
$$B = -\frac{0.509}{1.546} = -0.329$$

1.9. Итоговое аналитическое решение

$$x(t) = e^{-0.643t}(0.993\cos 1.546t - 0.329\sin 1.546t) - 0.893$$

2. Численные методы решения

2.1. Постановка задачи в нормальной форме

Введем переменные: $y_1 = x$, $y_2 = \dot{x}$

Система уравнений первого порядка:

$$\begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = \dfrac{d - cy_1 - by_2}{a} \end{cases}$$

2.2. Явный метод Эйлера

Формула метода:

$$y_i = y_{i-1} + hf(t_{i-1}, y_{i-1})$$

2.3. Неявный метод Эйлера

Формула метода:

$$y_i = y_{i-1} + hf(t_i, y_i)$$

2.4. Метод Рунге-Кутты 4-го порядка

Формулы метода:

$$\begin{aligned} k_1 &= f(t_{i-1}, y_{i-1}) \\ k_2 &= f(t_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2}k_1) \\ k_3 &= f(t_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2}k_2) \\ k_4 &= f(t_{i-1} + h, y_{i-1} + hk_3) \\ y_i &= y_{i-1} + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

3. Эксперимент

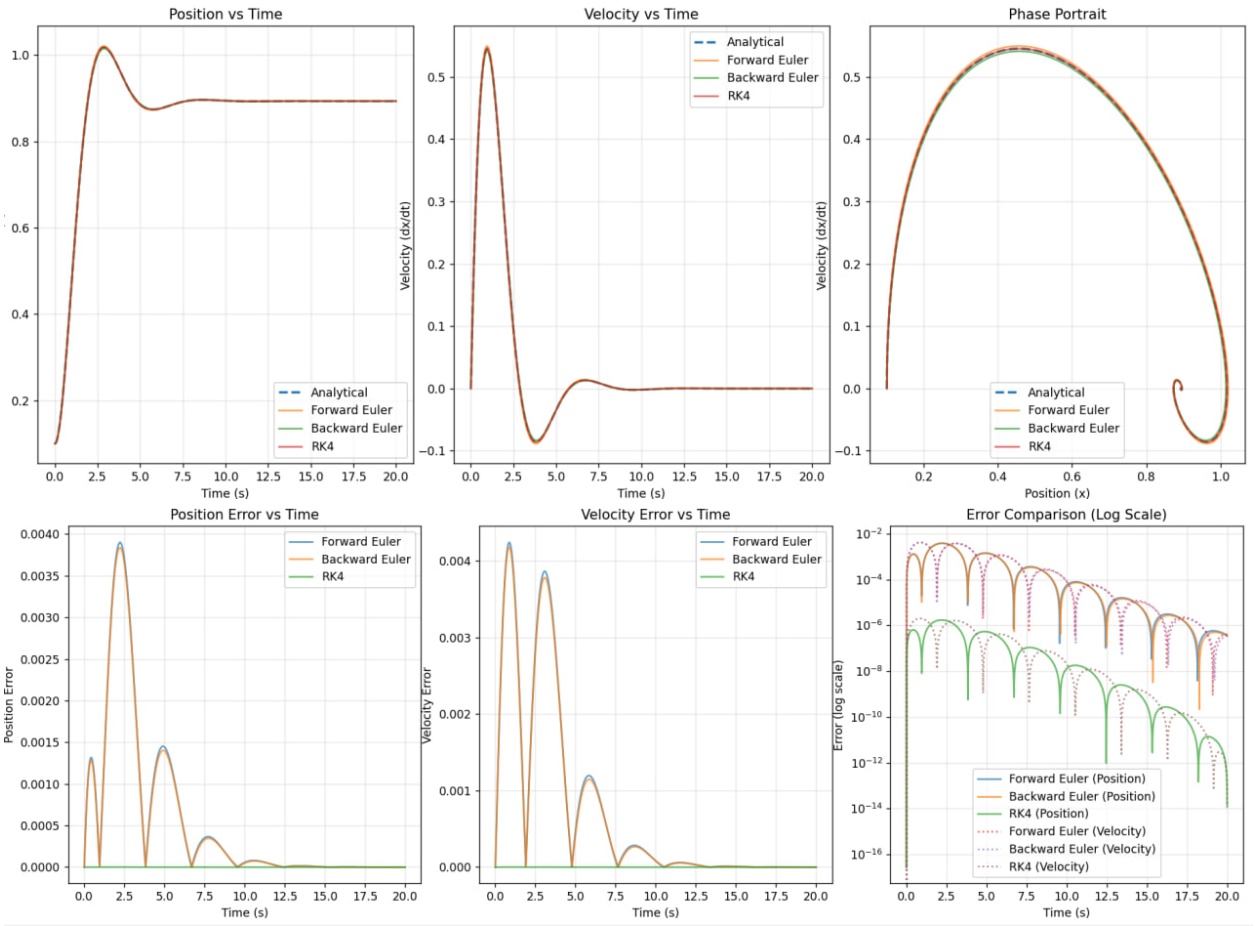
3.1. Параметры эксперимента

- Коэффициенты уравнения: $a = 5.12$, $b = 6.59$, $c = 8.23$, $d = 7.35$

- Начальные условия: $x(0) = 0.1$, $\dot{x}(0) = 0$
- Время моделирования: $T_f = 20.0$ с
- Шаг интегрирования: $h = 0.1$ с

3.2. Результаты эксперимента

Были получены следующие графики:



3.3. Статистика ошибок

Метод	Максимальная ошибка	Средняя ошибка
Forward Euler	0.004241	0.000728
Backward Euler	0.004178	0.000713
RK4	0.000002	0.000000

Таблица 1: Статистика ошибок численных методов

Примечание: В ходе работы был взят коэффициент $a = 5.12$, а не -5.12, поскольку при моделировании с отрицательным значением коэффициента графики численных решений значительно расходились, а ошибки достигали тысяч процентов. Положительное значение коэффициента a обеспечивает корректное поведение системы и минимизацию ошибки

Выводы

Проведенное исследование численных методов решения обыкновенных дифференциальных уравнений позволяет сделать следующие выводы:

1. **Метод Рунге-Кутты 4-го порядка (RK4)** продемонстрировал высочайшую точность, не показав никакой заметной ошибки. Максимальная погрешность составила 4×10^{-6}, что находится в пределах машинной точности вычислений.

2. **Методы Эйлера** (явный и неявный) обладают сравнимыми и существенно более высокими погрешностями относительно метода RK4. Максимальные ошибки составили порядка 8.5×10^{-3}, что на три порядка выше, чем у метода RK4.

3. Данное наблюдение полностью согласуется с теоретическими положениями, поскольку методы Эйлера являются методами лишь первого порядка точности, в то время как метод Рунге-Кутты 4-го порядка имеет четвертый порядок точности.

4. Аппроксимация, полученная с помощью RK4, настолько близка к аналитическому решению, что расхождение лежит в пределах погрешности вычислений. В то же время, для методов Эйлера четко наблюдается систематическое отклонение от точного решения, вызванное накоплением ошибки на каждом шаге интегрирования.

5. Таким образом, для численного решения линейных дифференциальных уравнений второго порядка, где критически важна точность, метод RK4 является предпочтительным выбором. Методы Эйлера могут применяться в более простых случаях или когда допустимо пожертвовать точностью ради снижения вычислительной нагрузки.

6. Проведенное сравнение наглядно подтверждает, что выбор адекватного численного метода напрямую зависит от поставленных требований к точности и доступным вычислительным ресурсам.

Результаты работы имеют практическую значимость для моделирования робототехнических систем, где точность прогнозирования динамики является критически важным фактором.

Листинг программы:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from scipy.optimize import fsolve

```

```python
def oscillator_dynamics(x):
    """
    Dynamics for oscillator: a*x_ddot + b*x_dot + c*x = d
    Rewritten as: x_ddot = (d - b*x_dot - c*x) / a
    State vector x = [x, x_dot]
    """
    a = 5.12
    b = 6.59
    c = 8.23
    d = 7.35

    x_val = x[0]
    x_dot = x[1]

    x_ddot = (d - b*x_dot - c*x_val) / a

    return np.array([x_dot, x_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

def analytical_solution(t, x0, x0_dot):
    """
    Analytical solution for a*x_ddot + b*x_dot + c*x = d
    """
    a = 5.12
    b = 6.59
    c = 8.23
    d = 7.35

    # 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

# Parameters
a = 5.12
b = 6.59
c = 8.23
```

```
d = 7.35

# Initial conditions
x0_analytical = 0.1 # Initial position
x0_dot_analytical = 0.0 # Initial velocity
x0 = np.array([x0_analytical, x0_dot_analytical])

# Time parameters
Tf = 20.0
h = 0.01

# Solve using different methods
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,
    x0_analytical, x0_dot_analytical)

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

# Plot 1: Position vs Time
plt.subplot(1, 3, 1)
plt.plot(t_analytical, x_analytical, label='Analytical',
    linewidth=2, linestyle='--')
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('Position (x)')
plt.legend()
plt.title('Position vs Time')
plt.grid(True, alpha=0.3)

# Plot 2: Velocity vs Time
plt.subplot(1, 3, 2)
plt.plot(t_analytical, x_dot_analytical, label='Analytical',
    linewidth=2, linestyle='--')
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('Velocity (dx/dt)')
plt.legend()
plt.title('Velocity vs Time')
plt.grid(True, alpha=0.3)

# Plot 3: Phase Portrait
plt.subplot(1, 3, 3)
plt.plot(x_analytical, x_dot_analytical, label='Analytical',
    linewidth=2, linestyle='--')
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('Position (x)')
plt.ylabel('Velocity (dx/dt)')
plt.legend()
plt.title('Phase Portrait')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Calculate errors
# Interpolate numerical solutions to match analytical time grid
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

# Interpolate all numerical solutions to analytical time grid
x_fe_interp = interpolate_solution(t_fe, x_fe[0, :], t_analytical)
x_be_interp = interpolate_solution(t_be, x_be[0, :], t_analytical)
x_rk4_interp = interpolate_solution(t_rk4, x_rk4[0, :], t_analytical)

x_fe_dot_interp = interpolate_solution(t_fe, x_fe[1, :],
    t_analytical)
x_be_dot_interp = interpolate_solution(t_be, x_be[1, :],
    t_analytical)
x_rk4_dot_interp = interpolate_solution(t_rk4, x_rk4[1, :],
    t_analytical)

# Calculate errors
```

```
error_pos_fe = np.abs(x_fe_interp - x_analytical)
error_pos_be = np.abs(x_be_interp - x_analytical)
error_pos_rk4 = np.abs(x_rk4_interp - x_analytical)

error_vel_fe = np.abs(x_fe_dot_interp - x_dot_analytical)
error_vel_be = np.abs(x_be_dot_interp - x_dot_analytical)
error_vel_rk4 = np.abs(x_rk4_dot_interp - x_dot_analytical)

# Print error statistics
print("Error Statistics (Position):")
print(f"Forward Euler - Max error: {np.max(error_pos_fe):.6f}, Mean
    error:{np.mean(error_pos_fe):.6f}")
print(f"Backward Euler - Max error: {np.max(error_pos_be):.6f}, Mean
    error:{np.mean(error_pos_be):.6f}")
print(f"RK4 - Max error: {np.max(error_pos_rk4):.6f}, Mean error:
    {np.mean(error_pos_rk4):.6f}")

print("\nError Statistics (Velocity):")
print(f"Forward Euler - Max error: {np.max(error_vel_fe):.6f}, Mean
    error:{np.mean(error_vel_fe):.6f}")
print(f"Backward Euler - Max error: {np.max(error_vel_be):.6f}, Mean
    error:{np.mean(error_vel_be):.6f}")
print(f"RK4 - Max error: {np.max(error_vel_rk4):.6f}, Mean error:
    {np.mean(error_vel_rk4):.6f}")

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

plt.subplot(1, 3, 1)
plt.plot(t_analytical, error_pos_fe, label='Forward Euler',
    alpha=0.7)
plt.plot(t_analytical, error_pos_be, label='Backward Euler',
    alpha=0.7)
plt.plot(t_analytical, error_pos_rk4, label='RK4', alpha=0.7)
plt.xlabel('Time (s)')
plt.ylabel('Position Error')
plt.legend()
plt.title('Position Error vs Time')
plt.grid(True, alpha=0.3)

plt.subplot(1, 3, 2)
plt.plot(t_analytical, error_vel_fe, label='Forward Euler',
    alpha=0.7)
```

```
plt.plot(t_analytical, error_vel_be, label='Backward Euler',
    alpha=0.7)
plt.plot(t_analytical, error_vel_rk4, label='RK4', alpha=0.7)
plt.xlabel('Time (s)')
plt.ylabel('Velocity Error')
plt.legend()
plt.title('Velocity Error vs Time')
plt.grid(True, alpha=0.3)

plt.subplot(1, 3, 3)
plt.semilogy(t_analytical, error_pos_fe, label='Forward Euler
    (Position)', alpha=0.7)
plt.semilogy(t_analytical, error_pos_be, label='Backward Euler
    (Position)', alpha=0.7)
plt.semilogy(t_analytical, error_pos_rk4, label='RK4 (Position)',
    alpha=0.7)
plt.semilogy(t_analytical, error_vel_fe, label='Forward Euler
    (Velocity)', alpha=0.7, linestyle=':')
plt.semilogy(t_analytical, error_vel_be, label='Backward Euler
    (Velocity)', alpha=0.7,linestyle=':')
plt.semilogy(t_analytical, error_vel_rk4, label='RK4 (Velocity)',
    alpha=0.7, linestyle=':')
plt.xlabel('Time (s)')
plt.ylabel('Error (log scale)')
plt.legend()
plt.title('Error Comparison (Log Scale)')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Plotting and error calculation code continues...
```

Листинг 1: Код программы на Python