

THE MINISTRY OF SCIENCE AND HIGHER EDUCATION
OF THE RUSSIAN FEDERATION

ITMO University
(ITMO)

Faculty of Control Systems and Robotics

SYNOPSIS
for the subject
“Simulation of Robotic Systems”

on the topic:
practice 1. Integrators

Student:
R 4131c, 506185
Arthur

Movsesyan

Tutor:
Assistant, MSc *Rakshin Egor*

Saint Petersburg 2025

Task

1. Solve analytically the ODE in the form of:

$$a * x'' + b * x' + c * x = d$$

2. From the list take coefficients of your ODE and solve them with three integrators: Explicit/Implicit Euler, Rung-Kutta methods.

3. Compare results of these methods with analytical solution, discuss and conclude your thoughts in report.

Analytical Solution of the ODE

$$0.62 x''(t) + 9.26 x'(t) + 7.35 x(t) = 0.88$$

1. Normalized equation

$$x''(t) + 14.9355 x'(t) + 11.8548 x(t) = 1.4194$$

2. Homogeneous solution

Characteristic equation:

$$\lambda^2 + 14.9355 \lambda + 11.8548 = 0$$

Discriminant:

$$D = (14.9355)^2 - 4 \cdot 1 \cdot 11.8548 \approx 175.649, \sqrt{D} \approx 13.2555$$

Roots:

$$r_1 = (-14.9355 + 13.2555)/2 \approx -0.841$$

$$r_2 = (-14.9355 - 13.2555)/2 \approx -14.095$$

Homogeneous solution:

$$x_h(t) = C_1 e^{-0.841 t} + C_2 e^{-14.095 t}$$

3. Particular solution

Assume $x_p(t) = A$ (constant).

Substitute into original equation:

$$7.35 A = 0.88 \Rightarrow A = 0.88 / 7.35 = 88/735 \approx 0.11973$$

So: $x_p(t) = 0.11973$

4. General solution

$$x(t) = C_1 e^{-0.841t} + C_2 e^{-14.095t} + 0.11973$$

$$x'(t) = -0.83995 * C_1 e^{-0.841t} - 0.83995t - 14.0956 C_2 e^{-14.0956t}$$

Constants C_1 and C_2 are determined with initial values of $x(0) = 0.1$ and $x'(0) = 0$.

$$C_1 = -0.0209$$

$$C_2 = 0.0013$$

Integration Methods

Explicit Euler:

$$x_{n+1} = x_n + h * f(x_n)$$

Implicit Euler:

$$x_{n+1} = x_n + h * f(x_{n+1})$$

4th order Runge-Kutta integration method:

$$\begin{aligned} x_{n+1} &= x_n + \frac{h}{6} * (k_1 + 2k_2 + 2k_3 + k_4), \\ k_1 &= f(x_n), \\ k_2 &= f(x_n) + \frac{1}{2}h * k_1, \\ k_3 &= f(x_n) + \frac{1}{2}h * k_2, \\ k_4 &= f(x_n) + \frac{1}{2}h * k_3, \end{aligned}$$

Python solution of laboratory work:

```

import numpy as np
import matplotlib.pyplot as plt

def system_dynamics(x):

    a = 0.62
    b = 9.26
    c = 7.35
    d = 0.88

    x1 = x[0]
    x2 = x[1]

    x2_ddot = (d - b * x2 - c * x1) / a

    return np.array([x2, x2_ddot])

def analytical_x(Tf, h):
    t = np.arange(0, Tf + h, h)
    r1 = -0.83995
    r2 = -14.0956
    c1 = -0.0209
    c2 = 0.0013
    xp = 0.1197
    return [(c1 * np.exp(r1 * t)) + (c2 * np.exp(r2 * t)) + xp, t]

def analytical_xdot(Tf, h):
    t = np.arange(0, Tf + h, h)
    r1 = -0.83995
    r2 = -14.0956
    c1 = -0.0209
    c2 = 0.0013
    return [(r1 * c1 * np.exp(r1 * t)) + (r2 * c2 * np.exp(r2 * t)), 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*k3 + k4)

    return x_hist, t

# Test all integrators
x0 = np.array([0.1, 0.0]) # Initial state: [angle, angular_velocity]
Tf = 15.0
h = 0.05

# Forward Euler
x_fe, t_fe = forward_euler(system_dynamics, x0, Tf, h)

# Backward Euler
x_be, t_be = backward_euler(system_dynamics, x0, Tf, h)

# Runge-Kutta 4

```

```

x_rk4, t_rk4 = runge_kutta4(system_dynamics, x0, Tf, h)

# Analytical Solution
x_v, t_an = analytical_x(Tf, h)
x_vdot, t_andot = analytical_xdot(Tf, h)

# Plot results
plt.figure(figsize=(24, 8))

plt.subplot(1, 3, 1)
plt.plot(t_an, x_v, 'b-', linewidth=2, label= 'Analytical Solution')
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('Position')
plt.legend()
plt.title('Position vs Time')

plt.subplot(1, 3, 2)
plt.plot(t_andot, x_vdot, 'b-', linewidth=2, label= 'Analytical Solution')
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('Velocity')
plt.legend()
plt.title('Velocity vs Time')

plt.subplot(1, 3, 3)
plt.plot(x_v, x_vdot, 'b-', linewidth=2, label= 'Analytical Solution')
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('Position x')
plt.ylabel('Velocity')
plt.legend()
plt.title('Phase Portrait')

plt.tight_layout()
plt.show()

```

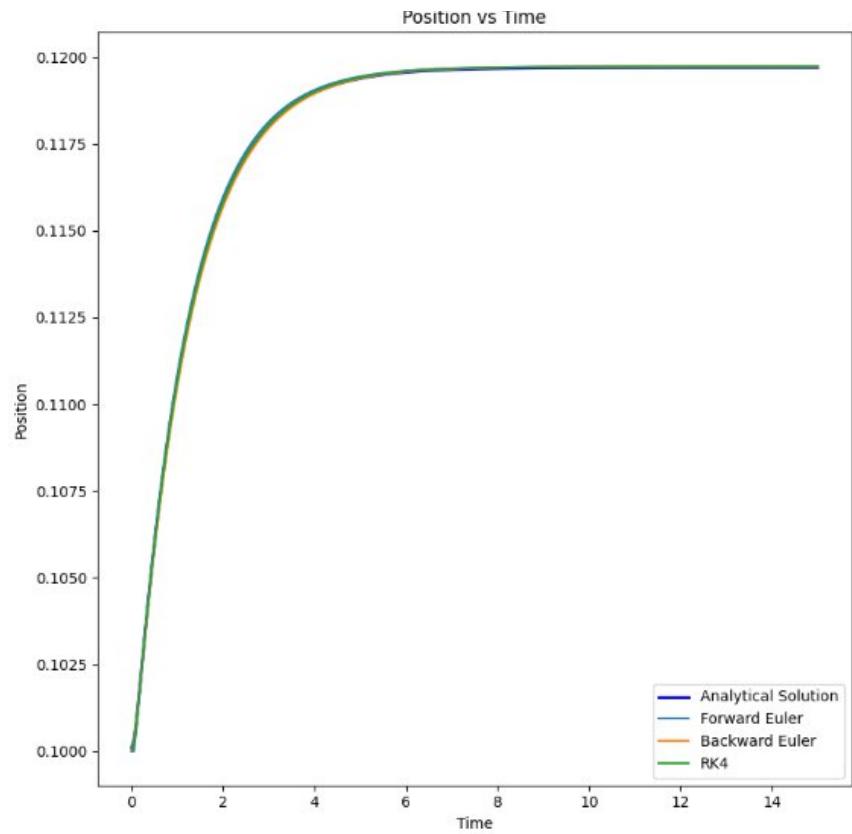


Fig 1. Position as a function of time

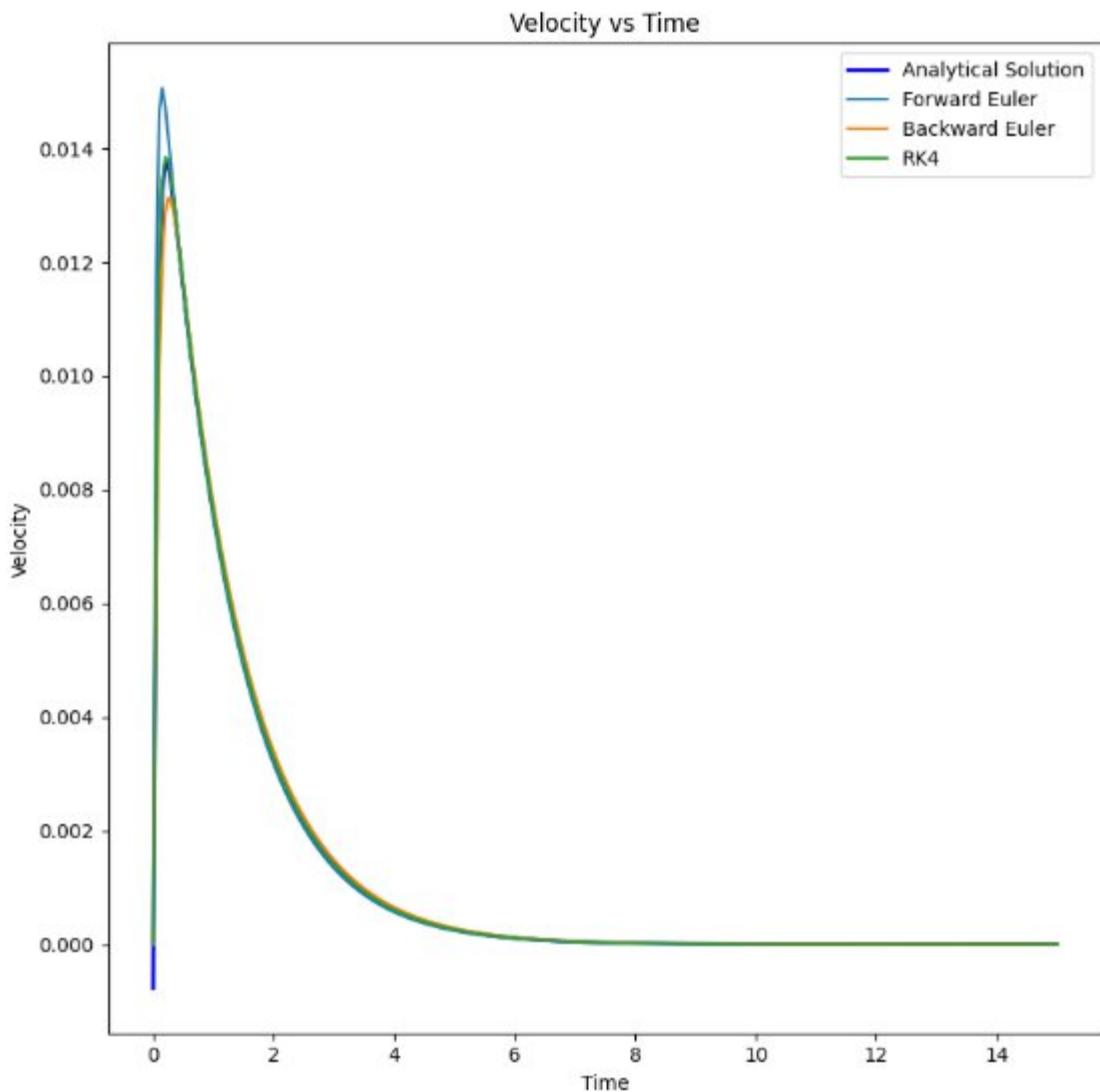


Fig. 2. Velocity as a function of time

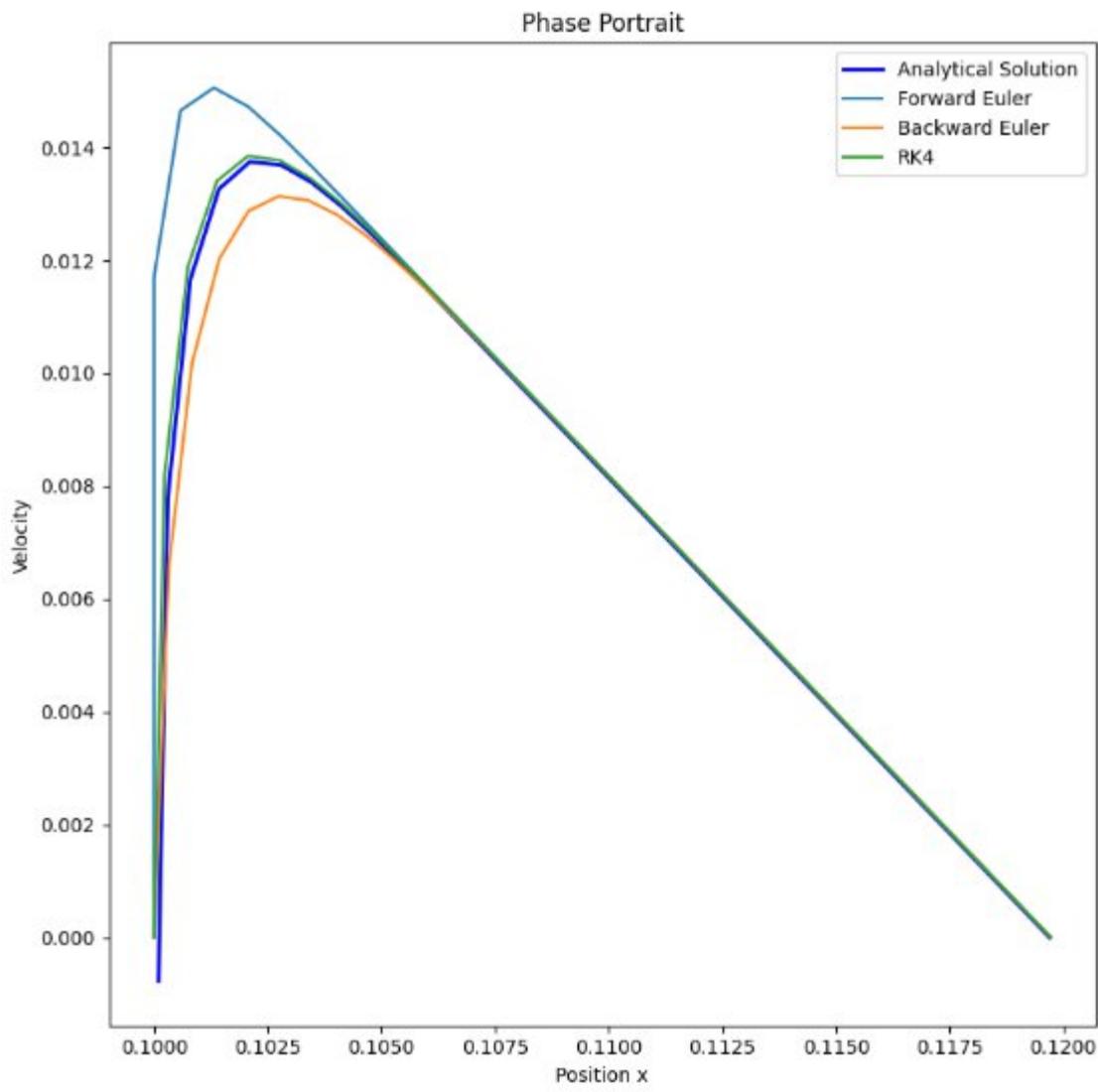


Fig. 3. Phase portrait

Conclusion

Among the numerical integration methods evaluated, the Runge–Kutta method yields the most accurate results, exhibiting excellent agreement with the analytically derived solution.