

THE MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE
RUSSIAN FEDERATION

ITMO University

Report for the task 1
for the subject
SIMULATION OF ROBOTIC SYSTEMS

Name: Afif Hazem

ID_Number: 503259

Saint Petersburg 6/11/2025

Solving the ODE analytically

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

Dividing through a (assuming $a \neq 0$):

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

Solving the homogeneous equation:

$$\ddot{x} + \frac{b}{a}\dot{x} + \frac{c}{a}x = 0$$

The characteristic equation is:

$$r^2 + \frac{b}{a}r + \frac{c}{a} = 0$$
$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Depending on the discriminant, we have three cases:

Case 1: Two real distinct roots (discriminant > 0)

$$r_1, r_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$x_{homo}(t) = C_1 e^{r_1 t} + C_2 e^{r_2 t}$$

Case 2: Repeated real root (discriminant = 0)

$$r = \frac{-b}{2a}$$

$$x_{homo}(t) = (C_1 + C_2 t) e^{rt}$$

Case 3: Two complex conjugate roots (discriminant < 0)

$$r_1, r_2 = \frac{-b}{2a} \pm i \frac{\sqrt{4ac - b^2}}{2a} = \alpha \pm i\beta$$

$$x_{homo}(t) = e^{\alpha t} (C_1 \cos \beta t + C_2 \sin \beta t)$$

The particular solution:

Since the right-hand side is constant $\frac{d}{a}$, then substituting $x = x_p$:

$$0 + 0 + \frac{c}{a}x_p = \frac{d}{a} \Rightarrow x_p = \frac{d}{c}$$

Hence, the general solution is:

$$x(t) = x_{homo}(t) + x_p$$

For my values in the table:

Let $a = -6.91$, $b = -9.21$, $c = 1.48$, $d = -4.31$ then:

$$\text{discriminant} = \sqrt{b^2 - 4ac} = \sqrt{(-9.21)^2 - 4(-6.91)(1.48)} = 11.213 > 0$$

Then there are two distinct real roots for the characteristic equation:

$$r_1 = -1.478, r_2 = 0.145$$

$$x_{\text{homo}}(t) = C_1 e^{-1.478 t} + C_2 e^{0.145 t}$$

The particular solution:

$$x_p = \frac{d}{c} = \frac{-4.31}{1.48} = -2.912$$

Hence, the general solution is:

$$x(t) = C_1 e^{-1.478 t} + C_2 e^{0.145 t} - 2.912$$

By substituting the initial conditions, we can determine the values of C_1, C_2 accordingly.

Solving the ODE numerically

We will write a code in Python that solves the ODE in three numerical methods:

- Runge-Kutta 4 method
- Implicit Euler method
- Explicit Euler method

Then we will calculate the general solution of the ODE analytically and then plot the function $x(t)$ in the 4 different methods and compare the results.

Code

```
import numpy as np
import matplotlib.pyplot as plt
a = -6.91
b = -9.21
c = 1.48
d = -4.31
def ODE_solution(x):
    x1 = x[0]
    x2 = x[1]
    dx1 = x2
    dx2 = (-b * x2 - c * x1 + d) / a
    return np.array([dx1, dx2])
```

```

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: [x, dx]
Tf = 10.0
h = 0.2

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

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

# Runge-Kutta 4
x_rk4, t_rk4 = runge_kutta4(ODE_solution, x0, Tf, h)

# Analytical Solution
discriminant = b**2 - 4*a*c
r1 = (-b + np.sqrt(discriminant)) / (2*a)
r2 = (-b - np.sqrt(discriminant)) / (2*a)
xp = d / c # Particular (constant) solution
X0 = 0.1 # Initial conditions
dX0 = 0.0
A = np.array([[1.0, 1.0], [r1, r2]]) # Solve for C1, C2
B = np.array([X0 - xp, dX0])
C1, C2 = np.linalg.solve(A, B)
T = np.linspace(0, 10, 600) # Time vector
X = C1 * np.exp(r1 * T) + C2 * np.exp(r2 * T) + xp # Compute x(t) and x'(t)
dX = C1 * r1 * np.exp(r1 * T) + C2 * r2 * np.exp(r2 * T)

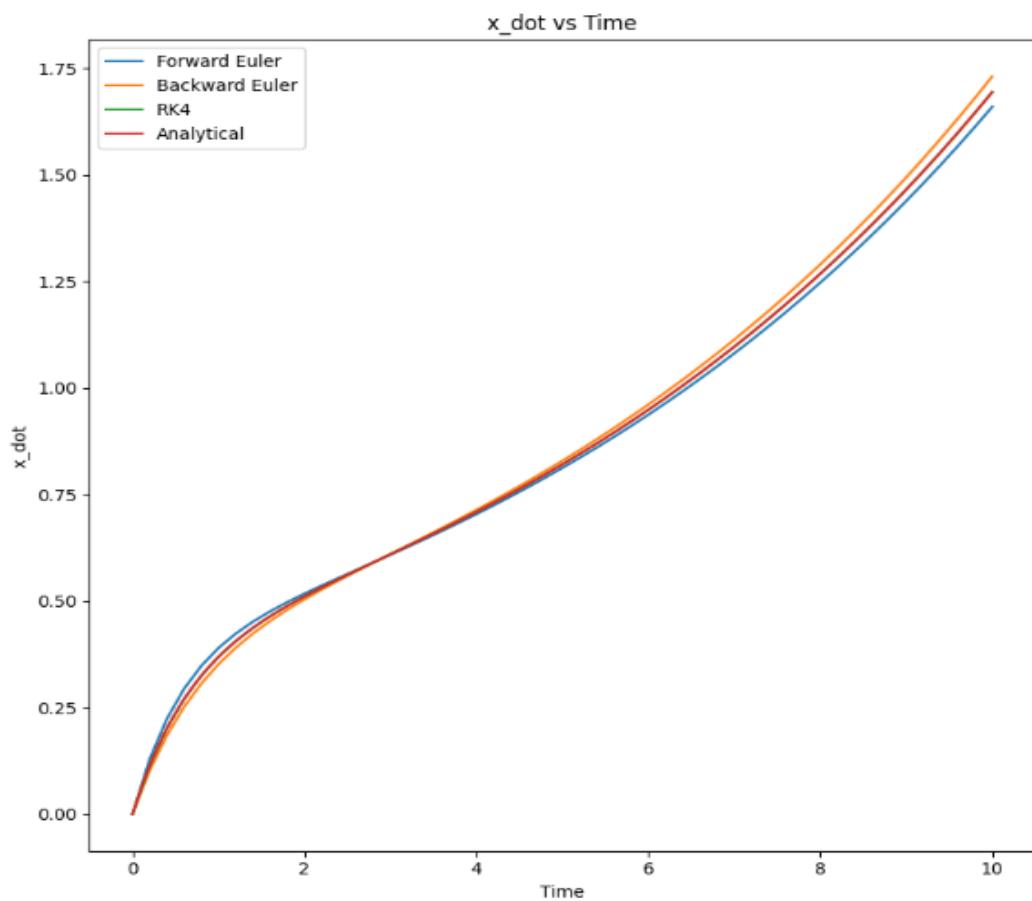
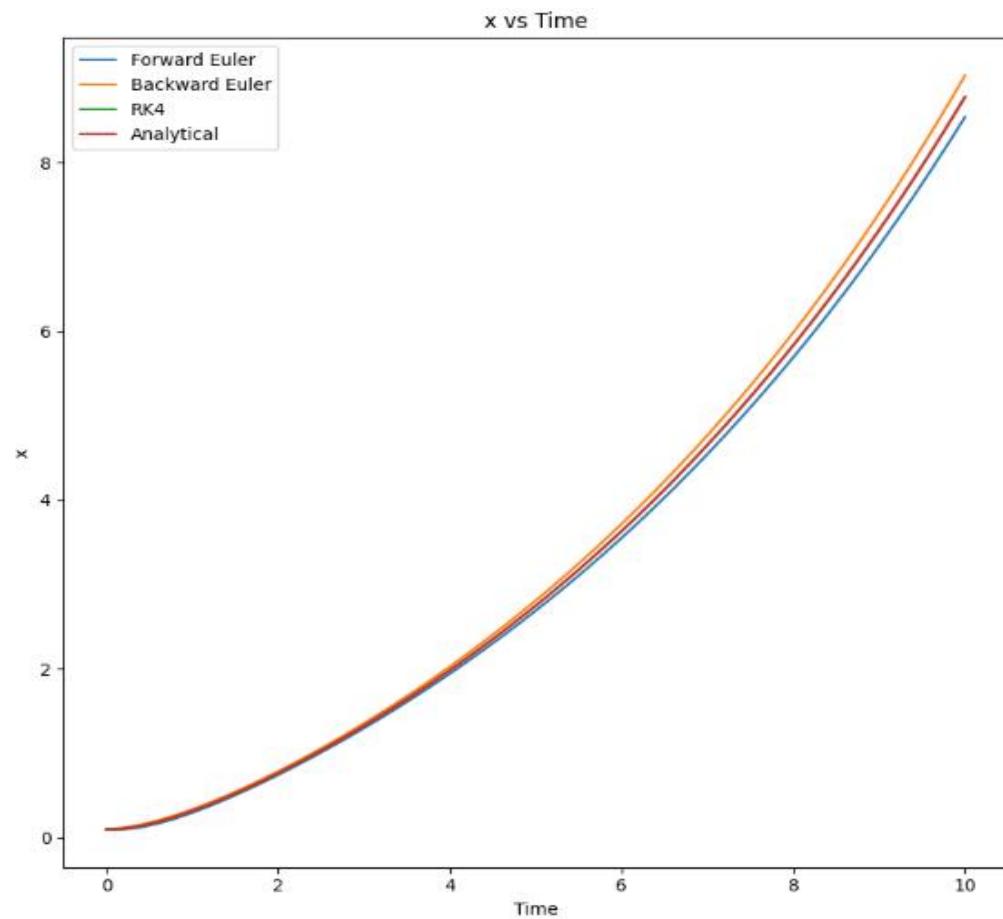
# Plot results
plt.figure(figsize=(24, 8))
plt.subplot(1, 3, 1)
plt.plot(t_fe, x_fe[0, :], label='Forward Euler')
plt.plot(t_be, x_be[0, :], label='Backward Euler')
plt.plot(t_rk4, x_rk4[0, :], label='RK4')
plt.plot(T, X[:,], label='Analytical')

```

```
plt.xlabel('Time')
plt.ylabel('x')
plt.legend()
plt.title('x vs Time')
plt.subplot(1, 3, 2)
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.plot(T, dX[:,], label='Analytical')
plt.xlabel('Time')
plt.ylabel('x_dot')
plt.legend()
plt.title('x_dot vs Time')
plt.subplot(1, 3, 3)
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.plot(X[:,], dX[:,], label='Analytical')
plt.xlabel('x')
plt.ylabel('x_dot')
plt.legend()
plt.title('Phase Portrait')
plt.tight_layout()
plt.show()
```

Results

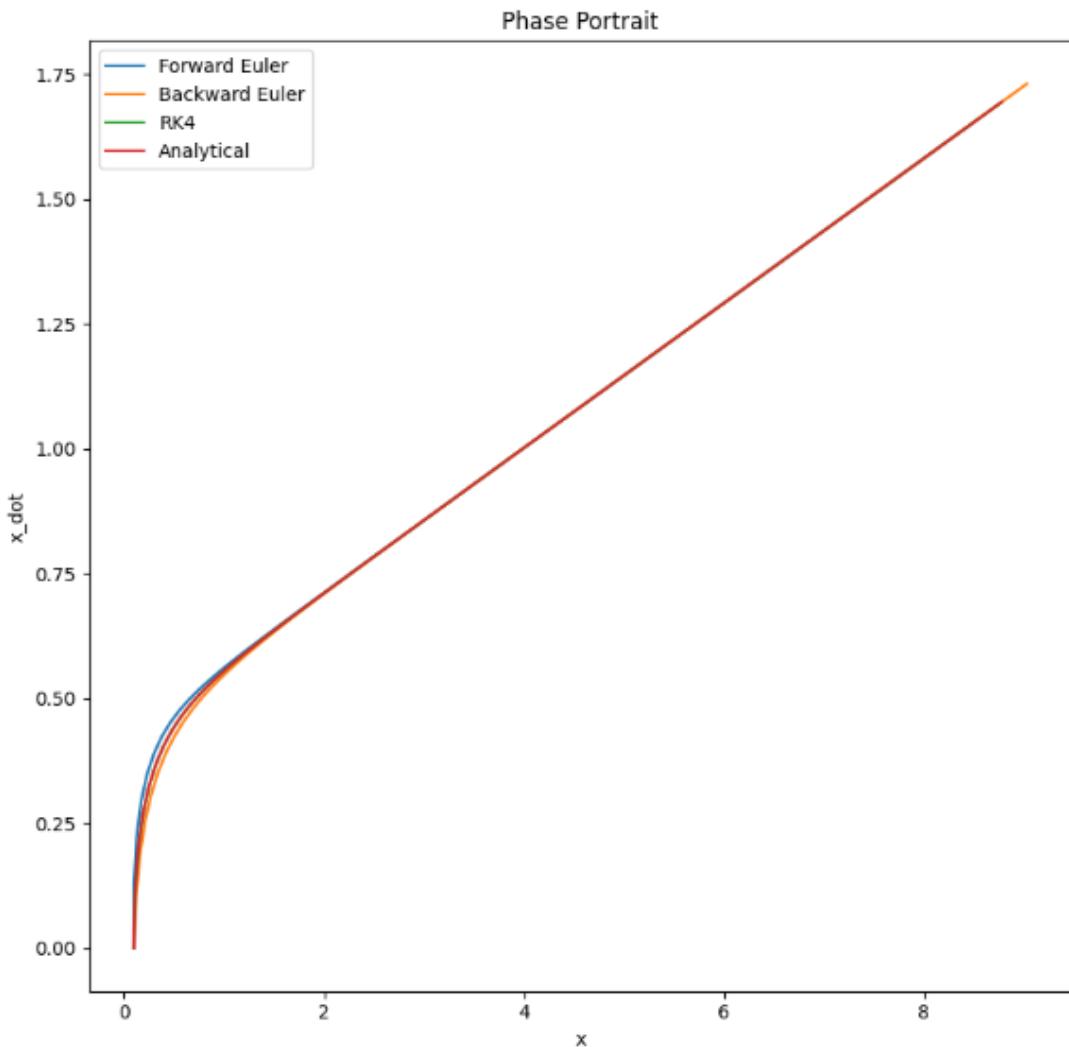
Plotting the function $x(t)$ using the 3 numerical methods comparing to the analytical solution:



We can notice from the plots that the RK4 method (green line) almost perfectly overlaps with the analytical solution (red line) which confirms RK4's high accuracy and low numerical error.

The Forward Euler (blue line) and Backward Euler (orange line) solutions deviate slightly from the analytical curve, especially for larger time values. We know that both are first-order methods, meaning their error grows linearly with the time step size h .

As a result, as time increases, the error between numerical and analytical solutions becomes more noticeable for the Euler methods, while RK4 remains very close.



For Phase Portrait plot, all methods follow a similar trajectory in the phase space, showing consistent dynamic behavior of the system.

The RK4 (green line) and Analytical (red line) curves almost perfectly overlap, indicating very high accuracy of RK4 as we said before.

The Forward Euler (blue line) and Backward Euler (orange line) curves deviate slightly near the beginning (small x), but converge closely with the analytical solution as x increases.