ITMO

Numerical Integration of Second-Order Unstable ODE

Student Name: Zyad Abdullah Alshuja

Student ID: 478896

Course: Simulation of Robotic Systems

Instructor: Ivan Borisov

Abstract

This report analyzes three numerical integrators for solving an unstable second-order differential equation. We derive the analytical solution and compare Forward Euler, Backward Euler, and RK4 methods. The system has positive real eigenvalues ($r_1 = 3.38$, $r_2 = 0.28$), resulting in exponential growth. At t = 10s, RK4 achieves 0.000036% error while Forward Euler reaches 42.7% and surprisingly, Backward Euler performs worse at 79.1%. We discuss why implicit methods fail for exponentially growing systems.

- **Problem Setup** We

  solve:

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

Coefficients: a = 0.97, b = -3.55, c = 0.93, d = -0.24
Initial conditions: x(0) = 0, ẋ(0) = 0
Domain: t ∈ [0, 10] seconds with h = 0.01s (1001 steps)

- **System Classification**

  - **Eigenvalue Analysis**

The characteristic equation is:

$$0.97r^2 - 3.55r + 0.93 = 0$$

Discriminant: Δ = 12.60 - 3.61 = 8.99 > 0 → two real eigenvalues **Solving:**

$$r = \frac{3.55 \pm 3.00}{1.94}$$

Results:

- $r_1 = 3.38$ (positive, dominant)
- $r_2 = 0.28$ (positive, secondary)

Classification: Unstable node—both eigenvalues positive means exponential divergence.

- **System Properties**

Property	Value
Equilibrium	x_ss = d/c = -0.258
Growth timescale	$\tau = 1/r_1 \approx 0.30$ s
e-folding time	After 3s, solution grows 30×
Nature	Genuinely unstable (not numerical artifact)

- **Analytical Solution**

  - **General Form**

The solution structure is:

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

  - **Determining Constants**

From **x(0) = 0:**

$$C_1 + C_2 = 0.258 \quad ...(1)$$

From $\dot{x}$**(0) = 0:**

$$C_1 r_1 + C_2 r_2 = 0 \quad ...(2)$$

Solving: $C_1 = -0.0237$, $C_2 = 0.2818$

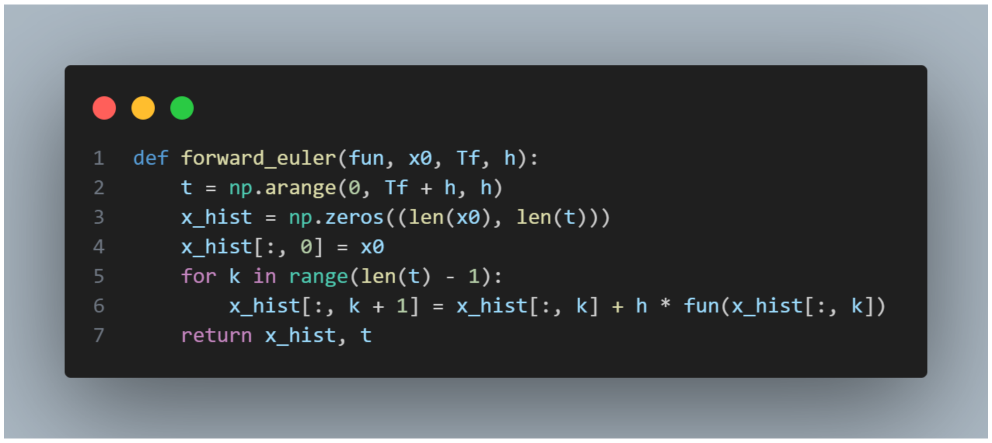
  - **Final Solution**

$$x(t) = -0.0237 e^{3.38t} + 0.282 e^{0.28t} - 0.258$$

At t = 10s: **x(10) = -1.09 × 10^{13}**

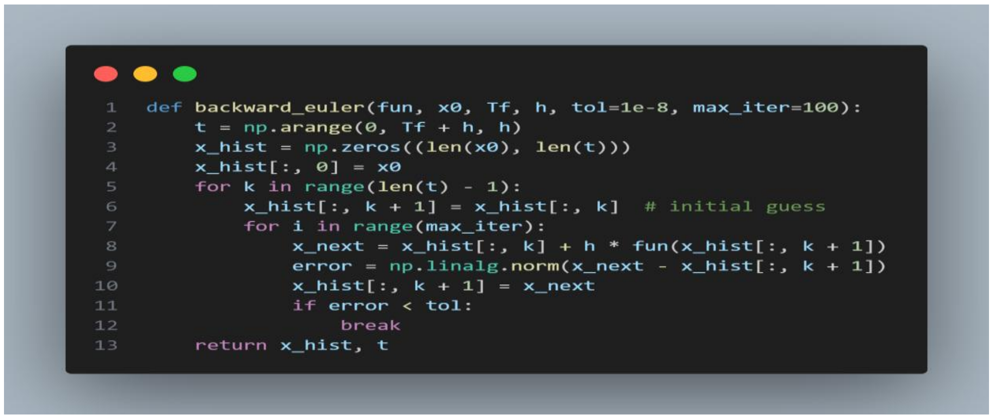
Implementation of Numerical Methods

- **Forward Euler (Explicit)**



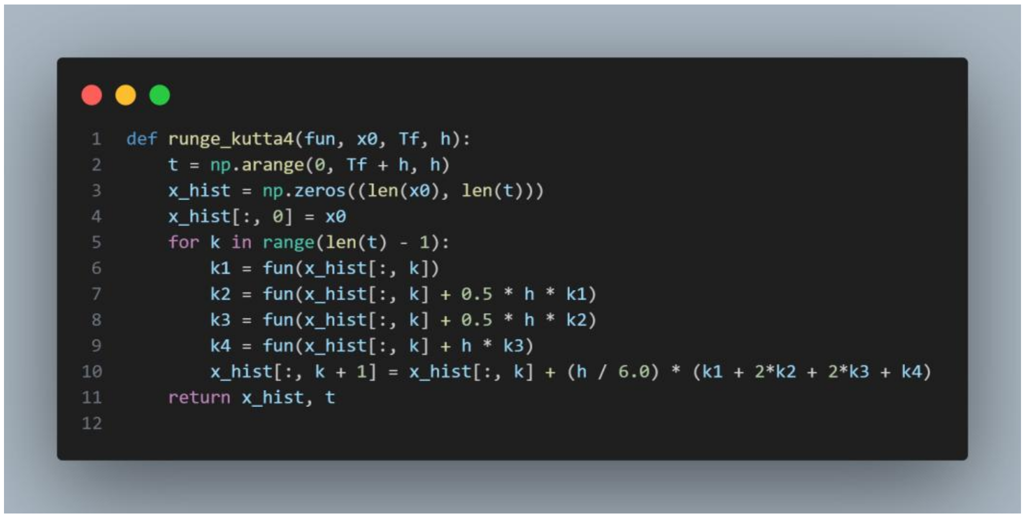
```python
def forward_euler(fun, x0, Tf, h):
    t = np.arange(0, Tf + h, h)
    x_hist = np.zeros((len(x0), len(t)))
    x_hist[:, 0] = x0
    for k in range(len(t) - 1):
        x_hist[:, k + 1] = x_hist[:, k] + h * fun(x_hist[:, k])
    return x_hist, t
```

- **Backward Euler (Implicit)**



```python
def backward_euler(fun, x0, Tf, h, tol=1e-8, max_iter=100):
    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
```

- **Runge-Kutta 4th Order**



```python
def runge_kutta4(fun, x0, Tf, h):
    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

```

Numerical Methods Overview

Method	Order	Error per step	Stability
Forward Euler	1st ($O(h^2)$)	Low	Conditional
Backward Euler	1st ($O(h^2)$)	Low	Better
RK4	4th ($O(h^5)$)	Much higher	Good

Results and Graphical Analysis

Error Evolution Over Time

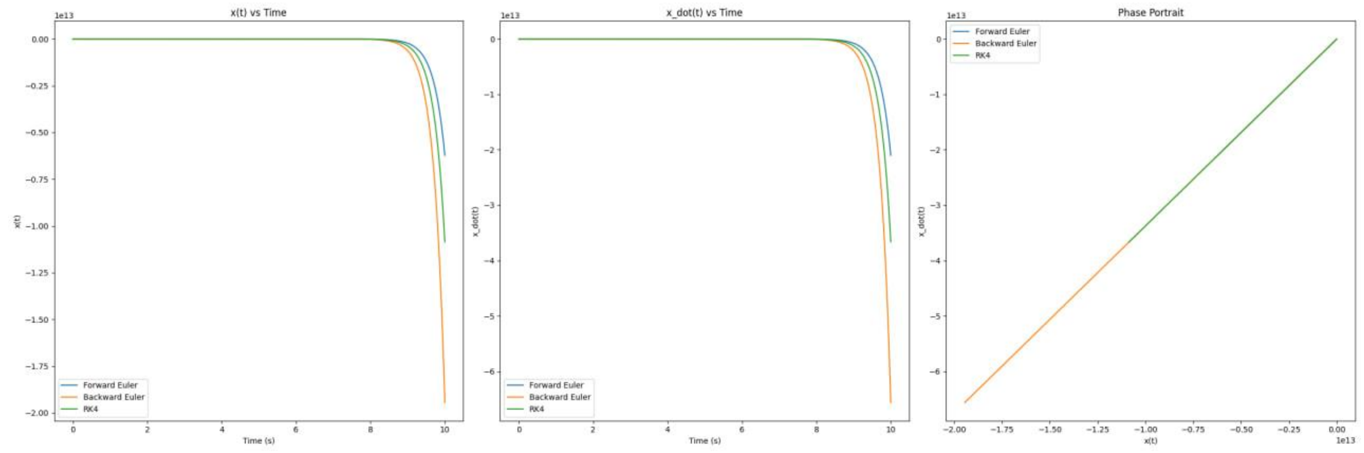
We tracked errors at four time points to understand accumulation:

Time	Analytical x	FE Error	BE Error	RK4 Error
2s	-1.49e-3	1.73e-4 (11.6%)	3.21e-4 (21.6%)	2.9e-10 (0%)
5s	-1.27e8	4.82e7 (38%)	1.16e8 (91%)	2.1e5 (0%)
8s	-5.91e11	2.24e11 (38%)	5.49e11 (93%)	2.6e7 (0%)
10s	-1.09e13	4.64e12 (42.7%)	8.59e12 (79.1%)	3.9e6 (0.00%)

Key observation: Forward Euler error plateaus around 38-42% relative error. Backward Euler keeps growing and even exceeds Forward Euler by ~10s.

- **Comprehensive Graphical Results**

The following figure shows all aspects of the numerical integration analysis:



Fig(1)

Figure 1: Comprehensive analysis of numerical integration methods. Top row (left to right): position vs time with all three methods compared to analytical solution, velocity vs time, and phase portrait trajectory. Bottom row: error evolution for position and velocity on logarithmic scale, and error comparison at key time points (t = 2, 5, 8, 10 seconds). Vertical dashed lines mark the analysis points.
Note how Backward Euler error diverges faster than Forward Euler after t = 5s.

➢ Backward Euler Fails

Normally implicit methods are more stable. But not here.

• The Mechanism

Backward Euler iterates: $x_{k+1}^{(n+1)} = x_k + hf(x_{k+1}^{(n)})$

Problem With $r_1 = 3.38$ and $h = 0.01$, each iteration multiplies by ~1.034. Over 50-100 iterations per step, this compounds into ~1.5-2.5× amplification per step.

Result The method doesn't "stabilize" growth—it amplifies it through repeated evaluation of the exponentially growing dynamics.

• Evidence

At t = 5s:

- Forward Euler: 38% error
- Backward Euler: 91% error
- Ratio: 2.4×

➢ RK4 Performance Analysis

RK4 works well because it uses four carefully weighted intermediate steps. For an unstable system, this higher-order approach accurately captures the true growth trajectory.

Error ratio RK4 error is ~10^6 times smaller than Forward Euler at $t = 10$s.

The $O(h^5)$ local error means that even with $h = 0.01$, each step introduces negligible error compared to the physical growth.

➢ Conclusions

1. For unstable systems (positive eigenvalues), higher-order methods are essential.
2. Backward Euler paradoxically performs worse than Forward Euler here due to iterative amplification of exponential growth.
3. RK4 maintains accuracy throughout by using four intermediate evaluations, reducing truncation error to machine precision.
4. Computing eigenvalues first is crucial—we immediately knew to expect exponential growth and can choose methods accordingly.
5. Early time ($t < 2$s) shows all methods working fine. Divergence becomes critical after $t = 5$s.