HACETTEPE UNIVERSITY

Department of Nuclear Engineering

NEM 394 ENGINEERING PROJECT II

Assignment 4

Numerical and Analytical Solution of Point Kinetics Equations with One Group of Delayed Neutrons

Student No. : **2230386062**

Student Name : **Emre Sakarya**

Delivery Date: 16.01.2026

Due Date: 16.01.2026

1 ABSTRACT

In this project, only the single-group delayed neutron approach was used, and accordingly, the point reactor kinetic equations were investigated. The time-dependent behavior of neutron density and precursor nucleus concentration was analyzed under the name of piecewise constant reactivity cases. The project essentially consists of four stages.

First, the system was expressed in matrix form, and an analytical solution was obtained using eigenvalues and eigenvectors. This obtained solution was then used as a reference for comparison with analytical solutions.

Next, numerical solutions were calculated using the Euler predictor-corrector (Heun) method and the fourth-order Runge-Kutta method, respectively. The accuracy of the numerical methods was compared with the analytical solution. Using different time steps, the effect of the chosen step size on the cutoff error and error propagation was clearly demonstrated with data.

Finally, the mathematical reason for the different shapes exhibited by the neutron density and precursor nucleus concentration graphs was investigated, and it was shown that this difference stems from the different time scales of the system. The results show that smaller time steps clearly improve numerical accuracy and that precursor nuclei create a delayed and smoothing effect on reactor dynamics.

2 INTRODUCTION

Understanding and predicting the time-dependent behavior of nuclear reactors is crucial for reactor safety, control, and even operational sustainability. Changes in reactor power result from the interaction of neutron flux, number, and delayed neutron precursors. These are a complex set of interactions that are difficult to resolve. These interactions are mathematically expressed by reactor kinetic equations and are widely used to study the reactor's response in transient states.

The point reactor kinetics approach offers a simplified model that considers only the time-dependent behavior of the neutron flux, neglecting the spatial effects of the reactor. This approach provides a suitable basis for comparing analytical and numerical methods, especially in transient state analyses. The presence of delayed neutrons slows down the reactor's response to abrupt changes in reactivity and contributes to more stable system behavior. Therefore, the presence of delayed neutrons is extremely critical for nuclear reactors.

In this project, reactor dynamics were investigated under a specific reactivity scenario using a single-group delayed neutron precursor approach. In the scenario considered, a positive reactivity step was applied to the system while the reactor was in a critical state, followed by a negative reactivity step, and the system's response and the effects it created were observed.

The main objective of this project is to solve this physical problem, which has an analytical solution, first analytically and then using different numerical methods. The accuracy, stability, and error propagation characteristics of these methods will be examined, and the method that yields more accurate results will be compared. After obtaining the analytical solution, numerical solutions were calculated using the Euler estimator-corrector method and the fourth-order Runge-Kutta method. Furthermore, different step sizes were chosen when using these methods, and the effect of step size on the accuracy of the method was investigated.

In the final part of the project, the reasons why the neutron density and precursor nucleus concentration graphs exhibit different shapes are explained mathematically. It is shown that the reason for this difference stems from the fast and slow dynamic modes of the system and their corresponding different time scales. This study highlights the critical impact of selecting the appropriate numerical method and time step on the accuracy of results in the simulation of nuclear systems.

3 METHODS AND CALCULATIONS

Point kinetic equations involving a single set of delayed neutrons were used to model the dynamic behavior of the nuclear reactor. The analyses were performed using three different approaches: analytical solutions and, in addition, numerical solutions using the Heun method and the Runge-Kutta (RK4) method.

3.1 Point Reactor Kinetics Equations (Single Group Delayed Neutron)

The system of differential equations describing the system is given below:

$$\frac{\boldsymbol{dn(t)}}{\boldsymbol{dt}} = \left(\frac{\boldsymbol{\rho(t) - \beta}}{\boldsymbol{\Lambda}}\right)\boldsymbol{n(t)} + \boldsymbol{\lambda\, C(t)}$$

$$\frac{\boldsymbol{dC(t)}}{\boldsymbol{dt}} = \left(\frac{\boldsymbol{\beta}}{\boldsymbol{\Lambda}}\right)\boldsymbol{n(t)} - \boldsymbol{\lambda\, C(t)}$$

Here, n(t) represents the neutron density and C(t) represents the precursor concentration. The constant parameters used in the simulations are as follows:

- Delayed neutron fraction (β): 0.007
- Prior decay constant (λ): 0,08 s^{-1}
- Neutron production time (λ): 10^{-3} s
- Initial neutron density (n): 10.0

3.1.1 Initial Condition (Equilibrium State)

$$\boldsymbol{n(0) = n_0}$$

$$\boldsymbol{C(0) = \beta\, n_0 / (\Lambda\, \lambda)}$$

3.2 Piecewise Constant Reactivity Function

$$\boldsymbol{\rho(t) = \begin{cases} 0.05\,\beta, & 0 \le t < 10 \\ -0.05\,\beta, & 10 \le t < 20 \\ 0, & t \ge 20 \end{cases}}$$

3.3 Analytical Solution – Matrix Form

$$\boldsymbol{A = \begin{bmatrix} \frac{\rho - \beta}{\Lambda} & \lambda \\ \frac{\beta}{\Lambda} & -\lambda \end{bmatrix}}$$

$$\boldsymbol{y(t) = c_1 v_1 e^{\omega_1 t} + c_2 v_2 e^{\omega_2 t}}$$

3.4 Heun Method

Predictor (Euler step):

$$\tilde{y}_{k+1} = \boldsymbol{y_k + h\, f(t_k, y_k)}$$

Corrector (Trapezoidal rule):

$$y_{k+1} = y_k + \left(\frac{h}{2}\right)[f(t_k, y_k) + f(t_{k+1}, \tilde{y}_{k+1})]$$

3.5 Runge-Kutta Method (RK4)

$$k_1 = f(t_k, y_k)$$

$$k_2 = f\left(t_k + \frac{h}{2}, y_k + \left(\frac{h}{2}\right)k^1\right)$$

$$k_3 = f\left(t_k + \frac{h}{2}, y_k + \left(\frac{h}{2}\right)k^2\right)$$

$$k_4 = f(t_k + h, y_k + h\,k^3)$$

$$y_{k+1} = y_k + \left(\frac{h}{6}\right)(k^1 + 2k^2 + 2k^3 + k^4)$$

3.6 Error analysis

$$E_{abs(t)} = \left| n_{num(t)} - n_{exact(t)} \right|$$

$$E_{rel(t)} = \frac{\left| n_{num(t)} - n_{exact(t)} \right|}{\left| n_{exact(t)} \right|}$$

3.7 Eigenvalue Equation

$$\det(A - \omega I) = 0$$

Eigenvalues:

$$\omega_1, \omega_2$$

Speed ratio:

$$|\omega_1| \,/\, |\omega_2|$$

4 RESULTS

This section investigates the dynamic behavior of point kinetic equations under the single-group delayed neutron approach. The system's response to changes in reactivity is analyzed using both analytical methods and various numerical integration techniques (Heun and RK4). The analyses calculate the steps the system undergoes and the changes in reactivity at specific time intervals, starting from the steady state at t=0. Finally, the difference between neutron density and delayed neutron precursor concentration is analyzed mathematically. The results are discussed using graphs and error analyses. This discussion includes the accuracy of the methods, error propagation characteristics, and which method is superior to the other.

Part A: Analytical Solution

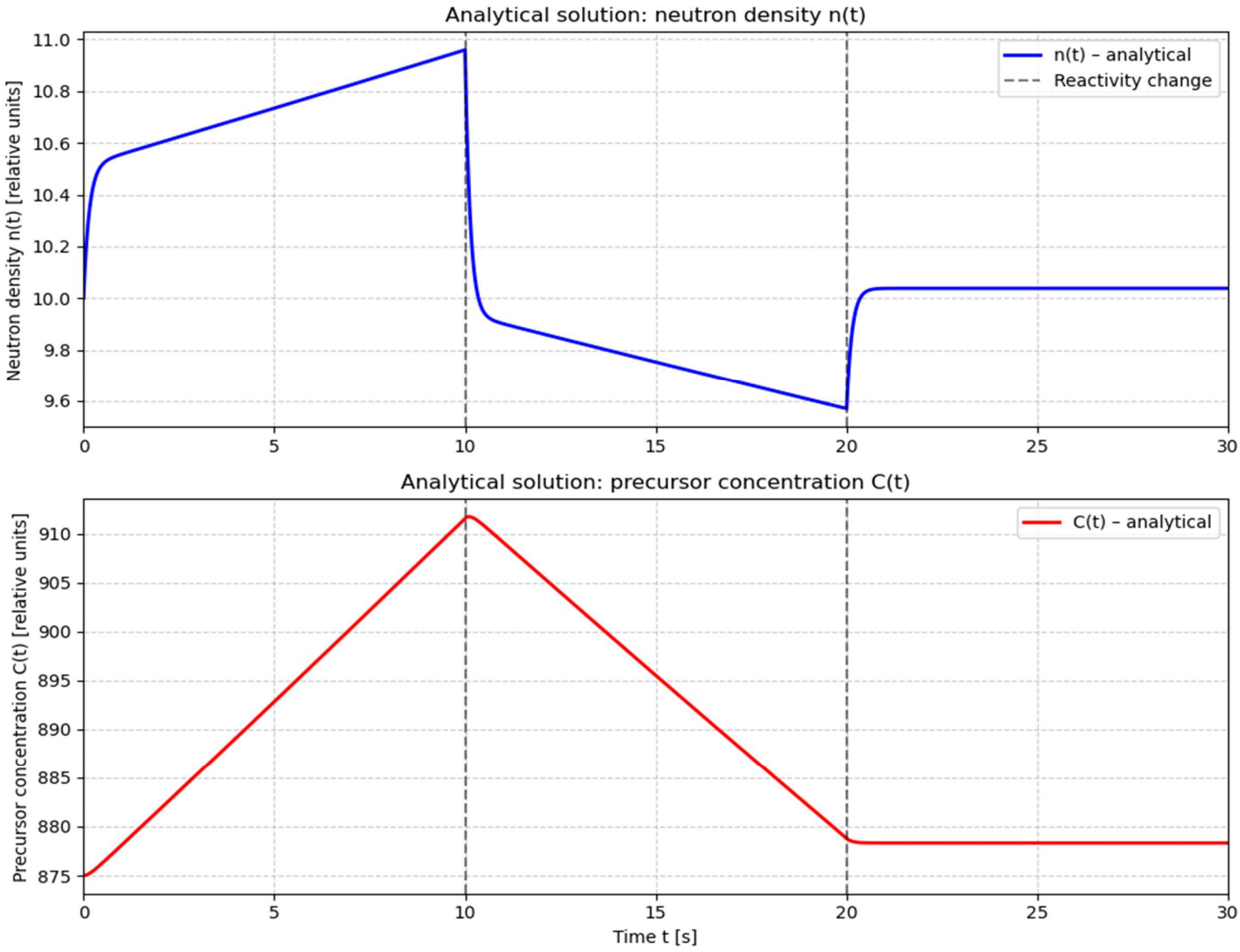

Figure 1: Analytical solution of neutron density $n(t)$and delayed neutron precursor concentration $C(t)$as functions of time under piecewise constant reactivity.

In the first phase of this project, the point kinetic equations for the single-group delayed neutron model were solved analytically. Reactivity was set to 0.05 β between 0-10 seconds, -0.05 β between 10-20 seconds, and zero after 20 seconds.

The graph at the top of Figure 1 shows the time-dependent change in neutron density. With positive reactivity, neutron density increases rapidly in the first 10 seconds. Then, as reactivity decreases, neutron density begins to decrease, and a new equilibrium state is reached when the change in reactivity completely stops.

The other graph in Figure 1 shows the concentration of delayed neutron precursors. Although the change in reactivity is exactly the same, the change in concentration over time is much smoother, indicating a more delayed response. Unlike abrupt changes in neutron concentration, it shows a slower and more regular change. This clearly demonstrates that delayed neutrons have a stabilizing effect on reactor kinetics.

In addition to the graphs, the neutron densities and neutron precursor concentrations obtained at the time intervals specified in the problem are given in the table below. These critical time points numerically represent the change in fluxes during the reactivity change of the system and as it approaches equilibrium.

Table 1: Analytical solution values of neutron density $n(t)$and delayed neutron precursor concentration $C(t)$at selected time points.

Time (s)	**Neutron density n(t)**	**Precursor concentration C(t)**
0	10.000000	875.000000
10	10.959593	911.585789
20	9.570000	878.792756
30	10.037997	878.324760

Part B: Heun Method

In this section, the point kinetic equations were solved using the Heun Method (Euler Predictive-Correction), a second-order numerical integration technique. To verify the accuracy of the method and observe how the step size affects the results, the results obtained in this section were compared with the exact analytical solution obtained in part (A), and error analysis was examined with different step intervals.

The simulation was initially performed with a time step of h = 0.01 seconds; this provides an optimal balance between computational cost and accuracy. This point is ideal for both accuracy and avoiding unnecessary steps. Figure 2 below shows a comparison of the obtained Heun numerical solution for neutron density and precursor nucleus concentration with the analytical solution.

The results are quite similar. Analysis of the graph reveals that the numerical solution fits the analytical solution curve perfectly. In particular, the sudden jumps in neutron density at t=10 and t=20 and the slower increase of delayed neutrons were successfully captured using the Heun method.

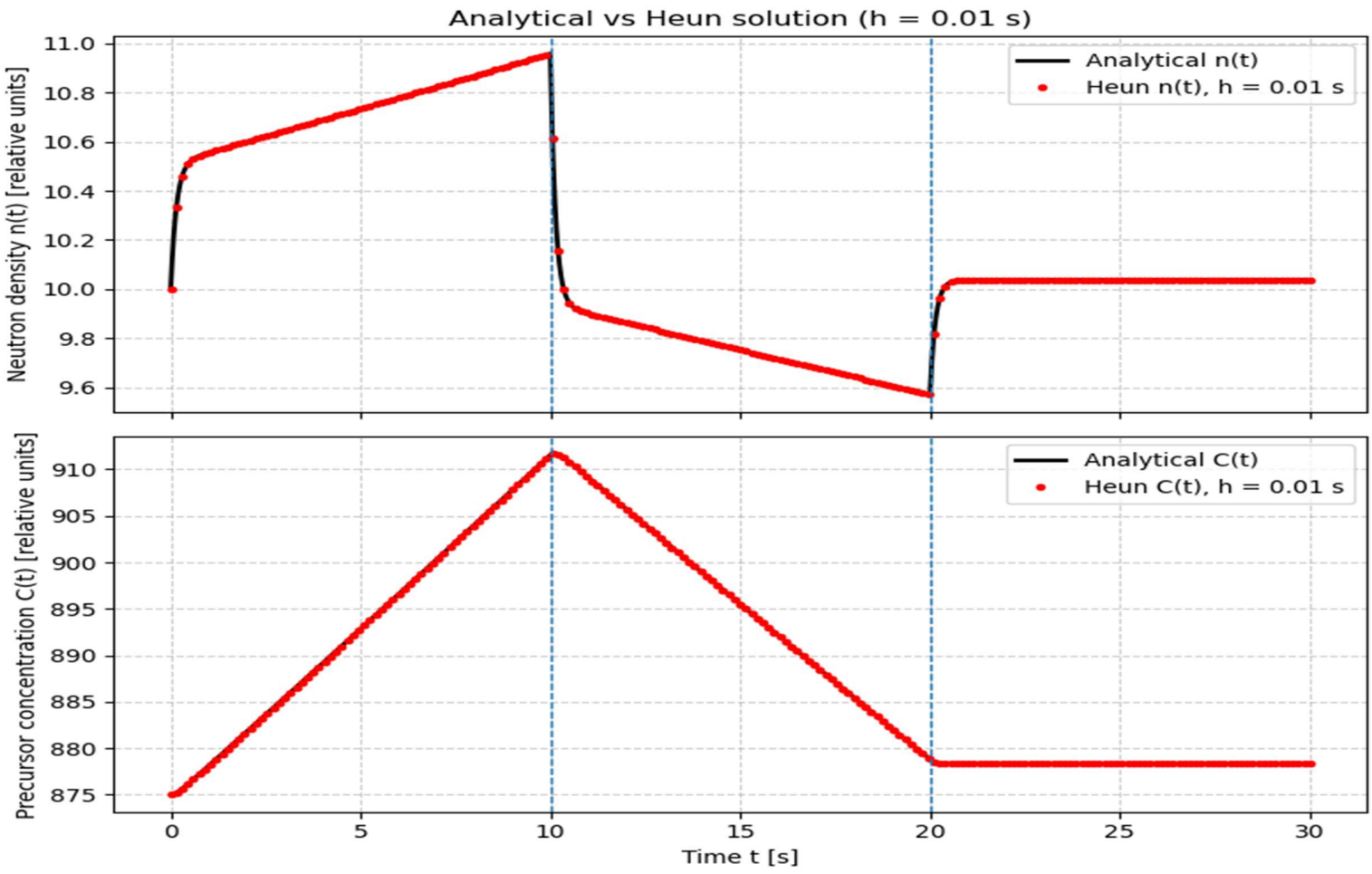

Figure 2: Analytical and Heun solutions of neutron density n(t) and precursor concentration C(t) for h = 0.01 s.

To evaluate the accuracy of the numerical method, this simulation was repeated with four different time steps (h = 0.1, 0.05, 0.01, 0.001 s). For each case, the maximum absolute error and the maximum relative error were calculated. These error rates are given in Table 2.

Table 2: Maximum absolute and relative errors in neutron density n(t) for different time step sizes using the Heun method.

h = dt [s]	Max abs. error	Max rel. error [%]
0.1000	3.835857e-01	3.500000
0.0500	1.917929e-01	1.750000
0.0100	3.835858e-02	0.350000
0.0010	3.835858e-03	0.035000

As can be seen from the table, as the time step decreases, the error values decrease significantly, as expected. Since the accuracy of the Heun method is on the order of $O(h^2)$, reducing the step size causes the error to decrease by a factor of the square of the square of the error. In the results obtained for h=0.001 s, the error has decreased to negligible, acceptable levels.

To examine how the error changes and accumulates over time, an additional graph showing the absolute error values on a logarithmic scale using the time steps in the table has been added and is shown in Figure 3.

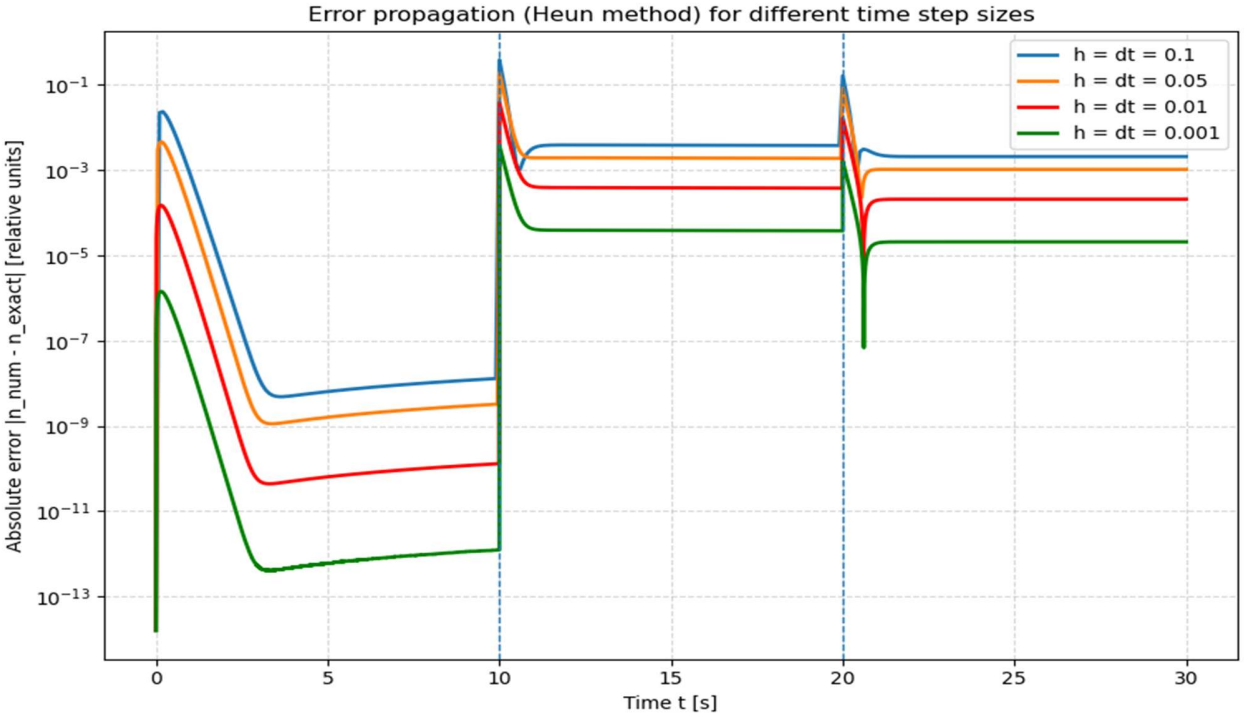

Figure 3: Error propagation of the Heun method for different time steps.

From this graph, we can easily draw two conclusions:

Firstly, as expected, as the time step h decreases, the error curves shift downwards (towards a lower error region). This proves that the method is consistent and convergent.

Another important point to note is:

Sudden increases are observed in the error graphs at t=10 s and t=20 s. The main reason for this is the change in reactivity, i.e., the reactivity function shows a step change at these points. This discontinuity in the derivative causes the numerical method to produce a slightly higher error at that step only momentarily. However, since the numerical method is stable and accurate, the error does not increase after this rise; on the contrary, it tends to decrease over time and reach equilibrium.

Part C: Runge-Kutta Method

In this part of the project, the equations were solved numerically using the fourth-order Runge–Kutta (RK4) method, and the results were compared with the analytical solution. The aim is to investigate the accuracy of the RK4 method on this problem and the effect of time step selection on error behavior.

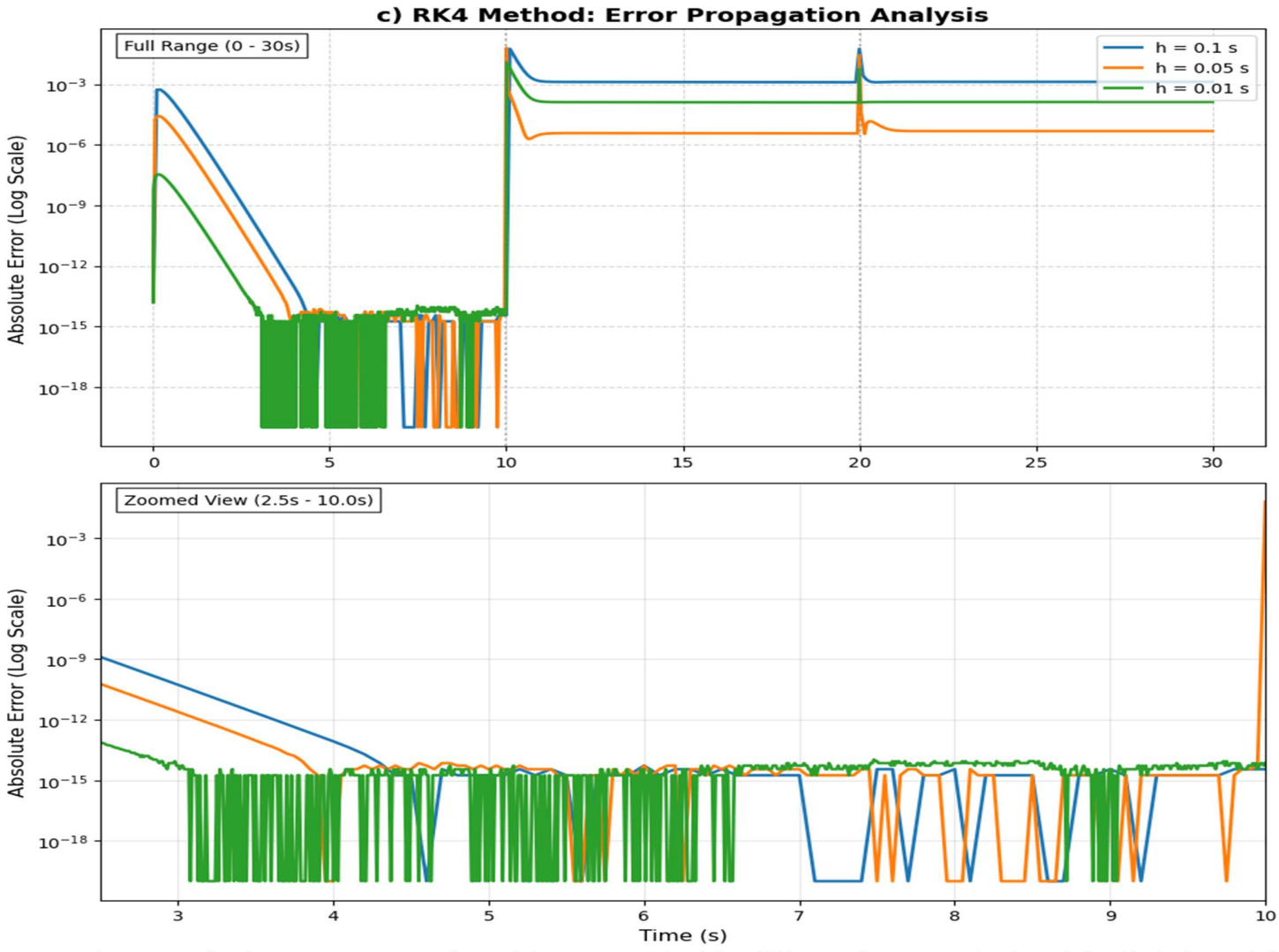

Figure 4: Absolute error propagation of the RK4 method for different time steps (top) and detailed view of the error behavior (bottom).

Figure 4 shows the variation of the absolute error of the neutron density solutions obtained by the RK4 method relative to the analytical solution at different time intervals in the 0-30 second time range. Examining the results obtained for different time steps, it is clearly seen that the error level decreases systematically as the time step decreases. This is due to the RK4 method having fourth-order accuracy. The graph shows that this method converges to the analytical solution very quickly.

To examine the time-dependent behavior of the error in more detail, Figure 4 shows an enlarged error graph for the 2.5–10 s interval. This time interval includes the critical region where positive reactivity is applied and the neutron population exhibits the fastest change. The enlarged graph shows that the RK4 method produces an extremely stable and smooth solution even in this fast transition region. This method obtained the smallest error in this interval for every time interval.

Finally, the numerical error results for this method are given in the table below.

Table 3: Maximum absolute and relative error results of the Runge-Kutta method.

Time Step (h)	**Max Abs Error (n)**	**Max Rel Error (%)**
0.1000	5.752995E-02	0.596624%
0.0500	6.393096E-02	0.583333%
0.0100	1.188003E-02	0.109130%

Part D: The Reason for the Difference Between the n(t) and C(t) Graphs

This part of the project analyzes and explains, mathematically and numerically, why the graph n(t) shows sudden jumps, while the graph C(t) exhibits a much slower and smoother change.

When we examine the structure of the point kinetic equations, we see an important difference. There is a very large difference between the time constants that control the neutron and precursor delayed nucleus populations.

- The neutron production time (Λ) is on the order of 10^-3 seconds.
- The average lifetime ($1/\lambda$) of the precursor nucleus is approximately on the order of 12.5 seconds.

The difference between these two time scales can be shown by comparing the eigenvalues of the system matrix; this value is 1620. This result indicates that the neutron population responds to changes in reactivity approximately 1620 times faster than the precursor nuclei. This also mathematically demonstrates that the system of differential equations is stiff.

If we compare the instantaneous rates of change, or their derivatives, the following graph emerges.

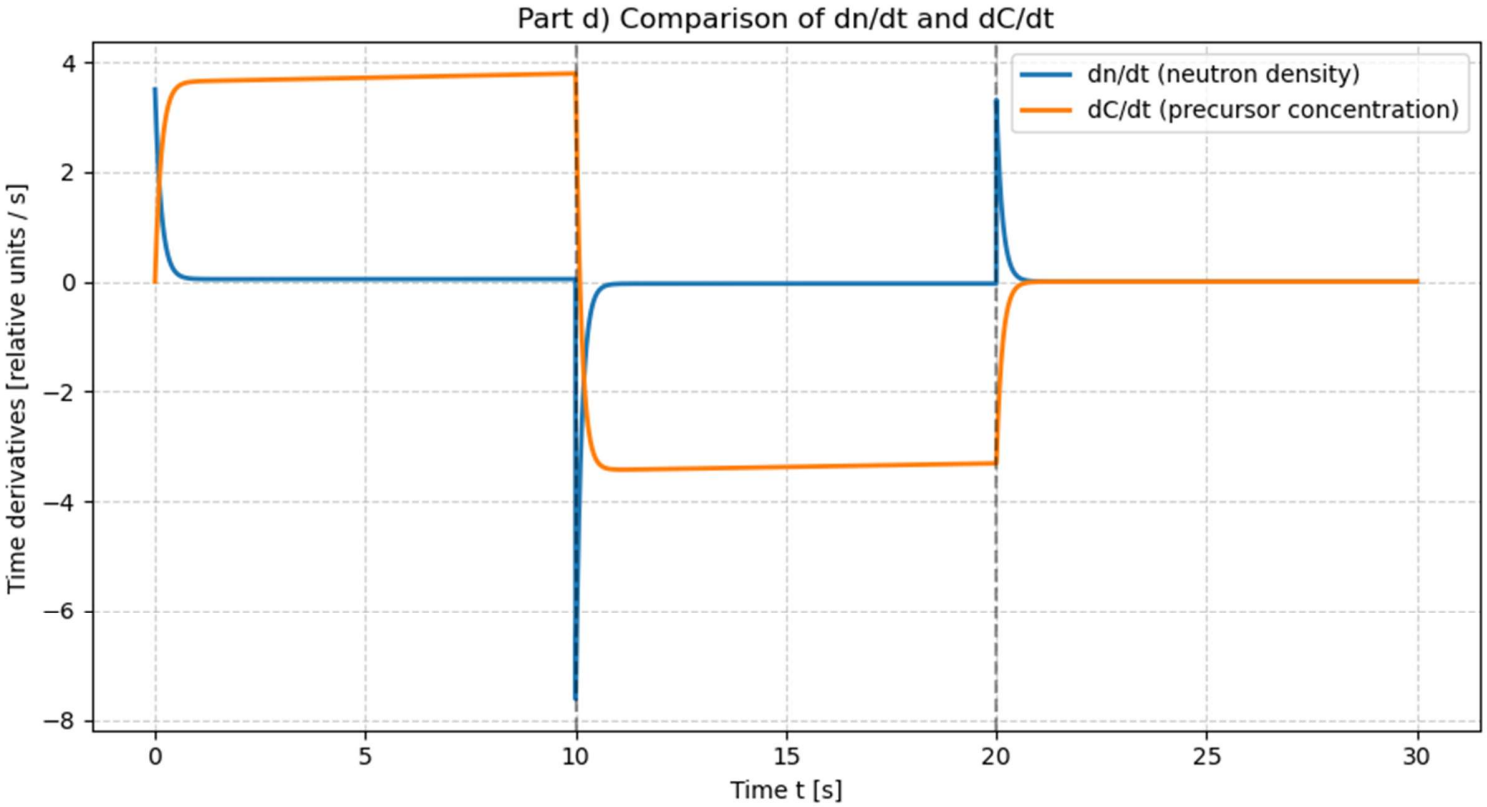

Figure 5: Comparison of the time-dependent rates of change in neutron density and precursor nucleus concentration.

As shown in the graph, when reactivity is first increased, i.e., at t = 0, the neutron exchange rate dn/dt suddenly rises to a very high value, while the previous exchange rate dC/dt only increases slightly and remains close to zero. Numerical analysis performed approximately immediately after the reactivity step (t ≈ 0.0010 s) yielded the following results:

$$\left|\frac{dn}{dt}\right| \approx 3.50\,\frac{units}{s} \qquad \left|\frac{dC}{dt}\right| \approx 0.025\,\frac{units}{\mathrm{s}}$$

The ratio of these two velocities is approximately 143 times. This result proves why the graph n(t) rises steeply, while numerically demonstrating why the graph C(t) increases slowly due to the "inertia" effect. Therefore, the fundamental mathematical reason for the shape difference between the graphs n(t) and C(t) is this significant time scale decomposition arising from the eigenvalues of the system and the large difference between the derivative magnitudes.

5 CONCLUSION

In this study, the dynamic behavior of a nuclear reactor is comprehensively investigated using point kinetic equations and analytical and numerical methods. Under a single-group delayed neutron model, both positive and negative constant reactivities are applied to the system; the system's response is studied, and consequently, the time-dependent variation of neutron density and precursor nucleus concentration is analyzed.

In the first stage, an analytical solution was obtained to reveal the fundamental dynamics of the system, and a reference value was taken for the numerical results. It was shown that neutron density exhibits abrupt responses to changes in reactivity, while the concentration of precursor nuclei shows a slower and more uniform behavior. These results are in perfect agreement with the physical structure of point kinetic equations.

In the second stage, numerical solutions were generated using the Euler Predictive-Correction (Heun) method and compared with the analytical solution. As the time step decreased, the numerical results rapidly approached the analytical solution, achieving the expected convergence. It was clearly observed that the error propagation decreased with second-order accuracy $O(h^2)$. This confirms that the Heun method offers a reliable approach for such systems depending on the number of steps.

In the third stage, the Runge-Kutta fourth-order (RK4) method was applied, and the results showed significantly lower error levels compared to the Heun method. For example, when the time interval was taken as h=0.01 seconds, the absolute error of the Heun method was on the order of 10^-3, while in the RK4 method this error was reduced to the order of 10^-8. This demonstrates the reason and necessity of preferring fourth-order methods for obtaining more accurate results in more complex nuclear simulations requiring high precision.

Analytical and numerical solutions have revealed that neutron density responds rapidly to changes in reactivity, while precursor nuclei show a much slower change. The reason for this difference has been proven by derivative analysis, examined in Section D. It has been calculated that the rate of change of neutrons dn/dt at the beginning of the reactivity step is approximately 143 times greater than the rate of change of precursor nuclei dC/dt. This difference provides extremely important information for nuclear reactor kinetics.

Additionally, the stiffness ratio was calculated, and the analysis revealed a difference of approximately 1600 times. This indicated that this differential equation is a "stiff" differential equation.

In conclusion, this project has demonstrated the importance of both accurate numerical methods and correct time steps in modelling nuclear reactor dynamics. The RK4 method has emerged as the most suitable method for solving this hard differential problem, providing both stability and high accuracy. In addition, this project has tested the accuracy of numerical methods in general and presented a clear and precise physical analysis of point kinetic equations.

6 REFERENCES

- (IAEA), I. A. (2011). *International Atomic Energy Agency (IAEA).* Vienna: Reactor Kinetics and Control.
- Chapra, S. C., & Canale, R. P. (2015). *Chapra, S. C.; Canale, R. P.* New York: Numerical Methods for Engineers.
- Duderstadt, J. J. (1976). *Duderstadt, J. J., & Hamilton, L. J.* New York: John Wiley & Sons.
- Hetrick, D. L. (1971). *Dynamics of Nuclear Reactors.* Chicago: University of Chicago Press.
- Lamarsh, J. R., & Baratta, A. J. (2001). *Introduction to Nuclear Engineering* (3rd ed.). Prentice Hall.
- Stacey, W. M. (2007). *Nuclear Reactor Physics.* Wiley-VCH: Weeinheim.

7 APPENDIX

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import eig, solve

# Parametreler
beta = 0.007
```

```
lam = 0.08
Lambda = 1e-3
n0 = 10.0

# Denge
C0 = (beta * n0) / (Lambda * lam)

def solve_interval(duration, X_start, rho_val, t_offset):
    # Katsayılar matrisi A
    a11 = (rho_val - beta) / Lambda
    a12 = lam
    a21 = beta / Lambda
    a22 = -lam
    A = np.array([[a11, a12], [a21, a22]])

    # Özdeğer ve özvektör hesabı
    w, v = eig(A)
    c = solve(v, X_start)

    # Zaman dizisi
    t = np.linspace(0, duration, 1000)

    # Çözümün oluşturulması
    term1 = c[0] * np.outer(v[:, 0], np.exp(w[0] * t))
    term2 = c[1] * np.outer(v[:, 1], np.exp(w[1] * t))

    X_sol = term1 + term2

    return t + t_offset, X_sol[0].real, X_sol[1].real

# --- 1. Aralık: 0-10 sn (rho = 0.05 * beta) ---
t1, n1, c1 = solve_interval(10.0, [n0, C0], 0.05 * beta, 0.0)

# --- 2. Aralık: 10-20 sn (rho = -0.05 * beta) ---
X_start_2 = [n1[-1], c1[-1]]
t2, n2, c2 = solve_interval(10.0, X_start_2, -0.05 * beta, 10.0)

# --- 3. Aralık: 20-30 sn (rho = 0) ---
X_start_3 = [n2[-1], c2[-1]]
t3, n3, c3 = solve_interval(10.0, X_start_3, 0.0, 20.0)
```

```
t_all = np.concatenate((t1, t2, t3))
n_all = np.concatenate((n1, n2, n3))
c_all = np.concatenate((c1, c2, c3))

plt.figure(figsize=(10, 8))

# n(t) grafiği
plt.subplot(2, 1, 1)
plt.plot(t_all, n_all, 'b-', label='n(t)')
plt.axvline(10, color='k', linestyle='--', alpha=0.5)
plt.axvline(20, color='k', linestyle='--', alpha=0.5)
plt.ylabel('Nötron Yoğunluğu n(t)')
plt.title('Analitik Çözüm Sonuçları')
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()

# C(t)
plt.subplot(2, 1, 2)
plt.plot(t_all, c_all, 'r-', label='C(t)')
plt.axvline(10, color='k', linestyle='--', alpha=0.5)
plt.axvline(20, color='k', linestyle='--', alpha=0.5)
plt.xlabel('Zaman (s)')
plt.ylabel('Öncül Yoğunluğu C(t)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()

plt.tight_layout()
plt.show()

print(f"{'time':<10} {'n(t)':<15} {'C(t)':<15}")
print("-" * 40)
print(f"{0:<10} {n_all[0]:<15.4f} {c_all[0]:<15.4f}")
print(f"{10:<10} {n1[-1]:<15.4f} {c1[-1]:<15.4f}")
print(f"{20:<10} {n2[-1]:<15.4f} {c2[-1]:<15.4f}")
print(f"{30:<10} {n3[-1]:<15.4f} {c3[-1]:<15.4f}")
```

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import eig, solve
```

```
beta = 0.007
lam = 0.08
Lambda = 1e-3
n0 = 10.0
C0 = (beta * n0) / (Lambda * lam)

def get_A(r):
    return np.array([[(r - beta)/Lambda, lam], [beta/Lambda, -lam]])

def rho(t):
    if t < 10.0: return 0.05 * beta
    elif t < 20.0: return -0.05 * beta
    else: return 0.0

def f(t, y):
    n, C = y
    r = rho(t)
    dn = ((r - beta)/Lambda)*n + lam*C
    dC = (beta/Lambda)*n - lam*C
    return np.array([dn, dC])

def heun(dt, t_end):
    N = int(round(t_end/dt))
    t = np.linspace(0, N*dt, N+1)
    y = np.zeros((N+1, 2))
    y[0] = [n0, C0]

    for k in range(N):
        dt_k = t[k+1] - t[k]
        k1 = f(t[k], y[k])
        y_p = y[k] + dt_k * k1
        k2 = f(t[k+1], y_p)
        y[k+1] = y[k] + (dt_k/2)*(k1 + k2)
    return t, y

# Analitik Cozum Hazirligi
# Bolge 1 (0-10s)
A1 = get_A(0.05 * beta)
w1, v1 = eig(A1)
```

```
c1 = solve(v1, [n0, C0])

def calc_reg1(t):
    res = c1[0]*v1[:,0]*np.exp(w1[0]*t) + c1[1]*v1[:,1]*np.exp(w1[1]*t)
    return res.real

X10 = calc_reg1(10.0)

# Bolge 2 (10-20s)
A2 = get_A(-0.05 * beta)
w2, v2 = eig(A2)
c2 = solve(v2, X10)

def calc_reg2(t):
    tau = t - 10.0
    res = c2[0]*v2[:,0]*np.exp(w2[0]*tau) + c2[1]*v2[:,1]*np.exp(w2[1]*tau)
    return res.real

X20 = calc_reg2(20.0)

# Bolge 3 (20-30s)
A3 = get_A(0.0)
w3, v3 = eig(A3)
c3 = solve(v3, X20)

def calc_reg3(t):
    tau = t - 20.0
    res = c3[0]*v3[:,0]*np.exp(w3[0]*tau) + c3[1]*v3[:,1]*np.exp(w3[1]*tau)
    return res.real

def get_exact_n(t_arr):
    n_res = np.zeros_like(t_arr)
    for i, t in enumerate(t_arr):
        if t <= 10.0:
            val = calc_reg1(t)
        elif t <= 20.0:
            val = calc_reg2(t)
        else:
            val = calc_reg3(t)
        n_res[i] = val[0]
```

```
    return n_res

# Hata
dt_list = [0.1, 0.05, 0.01, 0.001]
t_end = 30.0

print(f"{'dt':<10} | {'Max Abs Error':<20} | {'Max Rel Error (%)':<20}")
print("-" * 60)

store_data = {}

for dt in dt_list:
    t_num, y_num = heun(dt, t_end)
    n_num = y_num[:, 0]

    n_ex = get_exact_n(t_num)

    abs_err = np.abs(n_num - n_ex)
    max_abs = np.max(abs_err)

    mask = n_ex > 1e-12
    max_rel = np.max(np.abs((n_num[mask] - n_ex[mask])/n_ex[mask])) * 100

    print(f"{dt:<10.4f} | {max_abs:<20.6e} | {max_rel:.6f}%")
    store_data[dt] = (t_num, abs_err)

# Grafik
dt_plot = 0.01
t_p, y_p = heun(dt_plot, t_end)
n_p = y_p[:, 0]
n_ex_p = get_exact_n(t_p)

plt.figure(figsize=(10, 8))

plt.subplot(2, 1, 1)
plt.plot(t_p, n_ex_p, 'k-', label='Analytical')
plt.plot(t_p[::5], n_p[::5], 'ro', markersize=3, label=f'Heun (dt={dt_plot})')
plt.title(f'Heun vs Analytical (dt={dt_plot})')
plt.ylabel('n(t)')
plt.legend()
```

```
plt.grid(True, linestyle='--')

plt.subplot(2, 1, 2)
colors = ['r', 'orange', 'b', 'g']
for i, dt in enumerate(dt_list):
    t_d, err = store_data[dt]
    plt.semilogy(t_d, err, color=colors[i], label=f'dt={dt}')

plt.title('Error Propagation')
plt.xlabel('Time (s)')
plt.ylabel('Abs Error (Log)')
plt.legend()
plt.grid(True, linestyle='--')

plt.tight_layout()
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt

#Paremeters
beta = 0.007
lam = 0.08
Lambda = 1e-3
n0 = 10.0
C0 = (beta * n0) / (Lambda * lam)

t_end = 30.0

print("=" * 60)
print("PART C - RK4 METHOD ANALYSIS")
print("=" * 60)



def rho(t):
    if t <= 10.0: return 0.05 * beta
    elif t <= 20.0: return -0.05 * beta
    else: return 0.0
```

```
def f_system(t, y):
    n, C = y
    r = rho(t)
    dn = ((r - beta) / Lambda) * n + lam * C
    dC = (beta / Lambda) * n - lam * C
    return np.array([dn, dC], dtype=float)


def rk4_solver(dt, t_final):
    N = int(np.round(t_final / dt))
    t = np.linspace(0.0, N * dt, N + 1)
    y = np.zeros((N + 1, 2))
    y[0] = [n0, C0]

    for i in range(N):
        h = dt
        k1 = f_system(t[i], y[i])
        k2 = f_system(t[i] + 0.5*h, y[i] + 0.5*h*k1)
        k3 = f_system(t[i] + 0.5*h, y[i] + 0.5*h*k2)
        k4 = f_system(t[i] + h, y[i] + h*k3)
        y[i+1] = y[i] + (h/6.0)*(k1 + 2*k2 + 2*k3 + k4)

    return t, y

def get_analytical_solution(t_array):
    t_array = np.atleast_1d(t_array)
    n_res = np.zeros_like(t_array, dtype=float)

    def get_A(r):
        return np.array([[(r-beta)/Lambda, lam], [beta/Lambda, -lam]])

    configs = []
    for r in [0.05*beta, -0.05*beta, 0.0]:
        val, vec = np.linalg.eig(get_A(r))
        configs.append((val, vec, np.linalg.inv(vec)))

    X0 = np.array([n0, C0])

    # Transition points
    w, V, Vi = configs[0]
```

```
    X10 = np.real(V @ np.diag(np.exp(w * 10.0)) @ Vi @ X0)

    w, V, Vi = configs[1]
    X20 = np.real(V @ np.diag(np.exp(w * 10.0)) @ Vi @ X10)

    for i, t in enumerate(t_array):
        if t <= 10.0:
            tau, start, cfg = t, X0, configs[0]
        elif t <= 20.0:
            tau, start, cfg = t - 10.0, X10, configs[1]
        else:
            tau, start, cfg = t - 20.0, X20, configs[2]

        w, v, vi = cfg
        res = np.real(v @ np.diag(np.exp(w * tau)) @ vi @ start)
        n_res[i] = res[0]

    return n_res

dt_list = [0.1, 0.05, 0.01]
error_data = {}

print(f"\n{'dt (s)':<10} | {'Max Abs Error':<20} | {'Max Rel Error (%)':<20}")
print("-" * 60)

for dt in dt_list:
    t_rk, y_rk = rk4_solver(dt, t_end)
    n_rk = y_rk[:, 0]
    n_ref = get_analytical_solution(t_rk)

    abs_err = np.abs(n_rk - n_ref)
    max_abs = np.max(abs_err)

    mask = n_ref > 1e-12
    max_rel = np.max(np.abs((n_rk[mask] - n_ref[mask]) / n_ref[mask])) * 100.0

    abs_err[abs_err == 0] = 1e-20 # Avoid log(0)
    error_data[dt] = (t_rk, abs_err)

    print(f"{dt:<10.4f} | {max_abs:<20.6e} | {max_rel:.6f}%")
```

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10))

# General view
for dt in dt_list:
    t_d, e_d = error_data[dt]
    ax1.semilogy(t_d, e_d, linewidth=2, label=f'h = {dt} s')

ax1.set_title('c) RK4 Method: Error Propagation Analysis')
ax1.set_ylabel('Absolute Error (Log Scale)')
ax1.grid(True, which="both", linestyle='--', alpha=0.5)
ax1.legend()
ax1.axvline(10, color='gray', linestyle=':')
ax1.axvline(20, color='gray', linestyle=':')

# Zoomed view
x_start, x_end = 2.5, 10.0
for dt in dt_list:
    t_d, e_d = error_data[dt]
    mask = (t_d >= x_start) & (t_d <= x_end)
    ax2.semilogy(t_d[mask], e_d[mask], linewidth=2, label=f'h = {dt} s')

ax2.set_xlim(x_start, x_end)
ax2.set_ylabel('Absolute Error (Log Scale)')
ax2.set_xlabel('Time (s)')
ax2.grid(True, which="both", linestyle='-', alpha=0.3)
ax2.text(0.02,       0.95,       f'Zoomed       View       ({x_start}-{x_end}s)',
transform=ax2.transAxes,
            bbox=dict(facecolor='white', alpha=0.8))

plt.tight_layout()
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import eig

# Parametreler
beta = 0.007
```

```
lam = 0.08
Lambda = 1e-3
n0 = 10.0
C0 = (beta * n0) / (Lambda * lam)

t_end = 30.0

# Reaktivite ve Diferansiyel Denklem
def rho(t):
    if t < 10.0: return 0.05 * beta
    elif t < 20.0: return -0.05 * beta
    else: return 0.0

def f(t, y):
    n, C = y
    r = rho(t)
    dn = ((r - beta)/Lambda)*n + lam*C
    dC = (beta/Lambda)*n - lam*C
    return np.array([dn, dC])

# RK4 Cozum ve Turev Hesabi
dt = 0.001
N = int(round(t_end/dt))
t = np.linspace(0, N*dt, N+1)
y = np.zeros((N+1, 2))
y[0] = [n0, C0]

dn_dt = np.zeros_like(t)
dC_dt = np.zeros_like(t)

for k in range(N):
    # Turevleri kaydet
    derivs = f(t[k], y[k])
    dn_dt[k] = derivs[0]
    dC_dt[k] = derivs[1]

    # RK4 Adimi
    h = dt
    k1 = f(t[k], y[k])
    k2 = f(t[k] + 0.5*h, y[k] + 0.5*h*k1)
```

```
    k3 = f(t[k] + 0.5*h, y[k] + 0.5*h*k2)
    k4 = f(t[k] + h, y[k] + h*k3)

    y[k+1] = y[k] + (h/6.0)*(k1 + 2*k2 + 2*k3 + k4)

# Son adim turevi
derivs = f(t[-1], y[-1])
dn_dt[-1] = derivs[0]
dC_dt[-1] = derivs[1]

# Matematiksel Analiz (Ozdegerler ve Oranlar)
r_val = 0.05 * beta
A = np.array([[(r_val - beta)/Lambda, lam], [beta/Lambda, -lam]])
w = np.sort(np.abs(eig(A)[0]))[::-1]
ratio_stiff = w[0] / w[1]

# Reaktivite adimindan hemen sonraki hiz orani
idx = 1
ratio_rate = np.abs(dn_dt[idx] / dC_dt[idx])

print(f"Stiffness Orani: {ratio_stiff:.1f}")
print(f"Hiz Orani (|dn/dt| / |dC/dt|): {ratio_rate:.1f}")

# Grafik
plt.figure(figsize=(10, 6))
plt.plot(t, dn_dt, label='dn/dt (Notron Hiz)')
plt.plot(t, dC_dt, label='dC/dt (Oncul Hiz)')
plt.axvline(10, color='k', linestyle='--', alpha=0.5)
plt.axvline(20, color='k', linestyle='--', alpha=0.5)
plt.xlabel('Zaman (s)')
plt.ylabel('Degisim Hizi (Turev)')
plt.title('Part D: dn/dt ve dC/dt Karsilastirmasi')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```