PHYS 580 - Computational Physics

Computational Physics by *Nicholas J. Giordano, Hisao Nakanishi*
Student: **Ralph Razzouk**

Lab 1

This report analyzes numerical methods for solving differential equations, specifically focusing on radioactive decay. We implement and compare Euler's method and Runge-Kutta methods (2nd and 4th order), evaluating their accuracy against the analytical solution. Our analysis demonstrates the superior accuracy of higher-order methods, particularly for smaller step sizes, while revealing interesting behaviors at larger step sizes that deviate from theoretical error scaling.

Problem 1 - Test

This section covered basic familiarization with programming tools and environments. No formal solution was required.

Problem 2 - Nuclear Decay

We implement the Euler method to solve the radioactive decay equation given by

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = -\frac{N(t)}{\tau},$$

where $N(t)$ is the number of atoms at time t. The analytical solution can be reached as follows

$$\begin{aligned}
\frac{\mathrm{d}N(t)}{\mathrm{d}t} &= -\frac{N(t)}{\tau} \\
\frac{\mathrm{d}N(t)}{N(t)} &= -\frac{1}{\tau}\,\mathrm{d}t \\
\int \frac{\mathrm{d}N(t)}{N(t)} &= -\frac{1}{\tau}\int \mathrm{d}t \\
\ln(N(t)) &= -\frac{t}{\tau} + C \\
N(t) &= \mathrm{e}^{-\frac{t}{\tau}+C} \\
N(t) &= c\mathrm{e}^{-\frac{t}{\tau}}.
\end{aligned}$$

We know that, at $t = 0$, $N(t = 0) = c = N(0)$. Thus, the analytical solution is given by

$$N(t) = N(0)\,\mathrm{e}^{-t/\tau},$$

where $N(0)$ is the starting or initial number of atoms. For simplicity, we use normalized units where $N(0) = 1$ and $\tau = 1$, giving

$$N(t) = \mathrm{e}^{-t}.$$

The Euler method discretizes this as

$$N_{i+1} = N_i(1 - \Delta t).$$

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

def euler_approximation(N0, tau, dt):
    t = np.arange(0, 5 * tau, dt)
    N = np.zeros_like(t)
    N[0] = N0

    for i in range(1, len(t)):
        dN = -N[i - 1] / tau
        N[i] = N[i - 1] + dN * dt

    return t, N


def exact_solution(t, N0, tau):
    return N0 * np.exp(-t / tau)


# Parameters
N0 = 1000  # Initial number of nuclei
tau = 1  # Decay constant

# Time step values
dt_values = [1.5, 1, 0.8, 0.2, 0.05]

# Loop over different time steps and then plot
t_exact = []
N_exact = []
t_approx = []
N_approx = []
for dt in dt_values:
    # Plotting
    plt.figure(figsize=(6, 4))
    plt.title(f"Euler Approximation for Nuclear Decay (dt = {dt})")
    plt.xlabel("Time ()")
    plt.ylabel("Number of Nuclei (N)")
    plt.grid(True)

    # Exact solution
    t_exact_i = np.arange(0, 5 * tau, dt)
    N_exact_i = exact_solution(t_exact_i, N0, tau)
    t_exact.append(t_exact_i)
    N_exact.append(N_exact_i)
    plt.plot(t_exact_i, N_exact_i, label=f"Exact Solution for {dt}")

    # Euler approximation
    t_approx_i, N_approx_i = euler_approximation(N0, tau, dt)
    t_approx.append(t_approx_i)
    N_approx.append(N_approx_i)
    plt.plot(t_approx_i, N_approx_i, label=f"t/ = {dt}")

    plt.legend()
    plt.savefig(f"Problem 2 - {dt}.png", dpi=300)
    plt.show()
```

Listing 1: Code for Problem 2

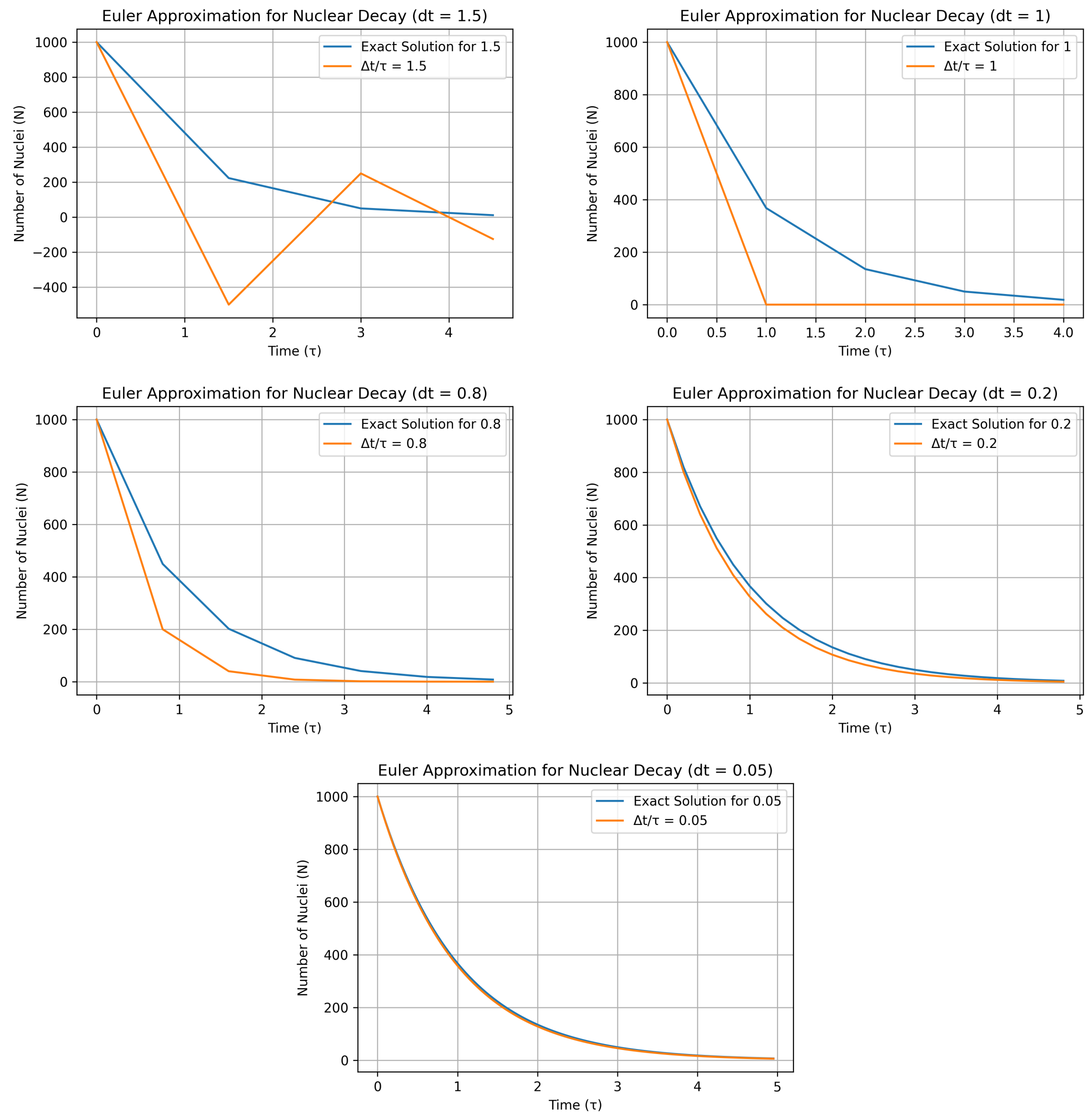

Figure 1: Plots the exact solution and Euler approximation for a certain value of $\Delta t/\tau$.

```python
# Plotting
plt.figure(figsize=(10, 6))
plt.title("Euler Method for Radioactive Decay")
plt.xlabel("Time ()")
plt.ylabel("Number of Atoms (N)")
plt.grid(True)

# Exact solution
t_exact_i = np.arange(0, 5 * tau, dt)
N_exact_i = exact_solution(t_exact_i, N0, tau)
t_exact.append(t_exact_i)
N_exact.append(N_exact_i)
plt.plot(t_exact_i, N_exact_i, label=f"Exact Solution for {dt}")

for dt in dt_values:
    # Euler approximation
    t_approx_i, N_approx_i = euler_approximation(N0, tau, dt)
    t_approx.append(t_approx_i)
    N_approx.append(N_approx_i)
    plt.plot(t_approx_i, N_approx_i, label=f"t/ = {dt}")

plt.legend()
plt.savefig("./Problem 2 - All.png")
plt.show()
```

Listing 2: Code for Problem 2

To generate a plot with all the Euler approximations in one plot, we simply save the figure after the for loop, with only one instance of the exact solution as follows

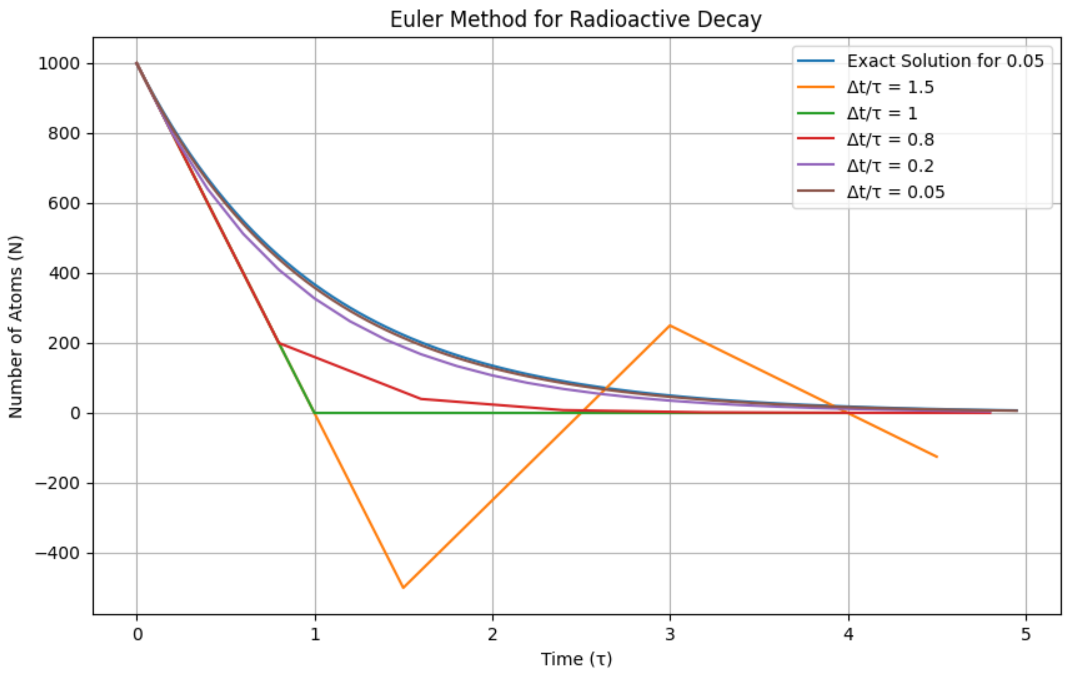

Figure 2: Plot of all the Euler approximations and the best exact solution.

Problem 3 - Error Analysis

To evaluate the accuracy of our numerical solutions, we compute both absolute and relative errors at each time step, as well as their cumulative values. For a numerical solution $N_{num}(t)$ and the exact solution $N_{exact}(t) = e^{-t}$, we define:

Point-wise Errors

- **Absolute Error:** The absolute error measures the magnitude of deviation from the exact solution

$$\epsilon_{abs}(t) = |N_{exact}(t) - N_{num}(t)|.$$

- **Relative Error:** The relative error normalizes this deviation by the exact value, giving us a percentage error

$$\epsilon_{rel}(t) = \frac{|N_{exact}(t) - N_{num}(t)|}{N_{exact}(t)} \times 100\%.$$

Cumulative Errors

To understand how errors accumulate over time, we also compute cumulative errors. For discrete time steps $t_i = t_0 + i\Delta t$, these are

- **Cumulative Absolute Error:**

$$\epsilon_{abs}^{(cum)}(t_f) = \sum_{i=0}^{t_f/\Delta t} \epsilon_{abs}(t_0 + i\Delta t).$$

- **Cumulative Relative Error:**

$$\epsilon_{rel}^{(cum)}(t_f) = \sum_{i=0}^{t_f/\Delta t} \epsilon_{rel}(t_0 + i\Delta t).$$

Error Analysis Results

Our analysis of the Euler method reveals several key patterns:

1. **Step Size Dependence**: For smaller step sizes ($\Delta t = 0.05, 0.2$), the errors scale approximately linearly with Δt, consistent with the theoretical local truncation error of $O(\Delta t)$.
2. **Time Evolution**: The relative error generally increases with time for all step sizes, reflecting the accumulation of truncation errors in each step.
3. **Critical Step Size**: For $\Delta t \geq 1$, we observe significant degradation in accuracy, particularly evident in the relative error plots. This suggests a practical upper limit for Δt in the Euler method.
4. **Error Accumulation**: The cumulative error plots show faster accumulation rates for larger step sizes, with notably different behavior above and below the critical step size of $\Delta t = 1$.

Implementation Details

For our numerical implementation, we compute these errors using vectorized operations for efficiency:

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

# Calculate the deviations
absolute_deviations = []
relative_deviations = []
cumulative_absolute_deviations = []
cumulative_relative_deviations = []
for i in range(len(dt_values)):
    abs_i = np.abs(N_approx[i] - N_exact[i])
    rel_i = abs_i / N_exact[i]
    absolute_deviations.append(abs_i)
    relative_deviations.append(rel_i)

    # Calculate the cumulative deviations
    cum_abs_i = np.cumsum(abs_i)
    cum_rel_i = np.cumsum(rel_i)
    cumulative_absolute_deviations.append(cum_abs_i)
    cumulative_relative_deviations.append(cum_rel_i)

    # Plot the absolute deviations
    plt.figure(figsize=(10, 6))
    plt.title(f"Absolute Deviations of Euler Approximation (dt = {dt_values[i]})")
    plt.xlabel("Time ()")
    plt.ylabel("Deviations")
    plt.grid(True)

    plt.plot(
        t_approx[i],
        abs_i,
        label="Absolute Deviation",
    )
    plt.plot(
        t_approx[i],
        cum_abs_i,
        label="Cumulative Absolute Deviation",
    )
    plt.legend()
    plt.show()

    # Plot the relative deviations
    plt.figure(figsize=(10, 6))
    plt.title(f"Relative Deviations of Euler Approximation (dt = {dt_values[i]})")
    plt.xlabel("Time ()")
    plt.ylabel("Deviations")
    plt.grid(True)

    plt.plot(
        t_approx[i],
        rel_i,
        label="Relative Deviation",
    )
    plt.plot(
        t_approx[i],
        cum_rel_i,
        label="Cumulative Relative Deviation",
    )
    plt.legend()
    plt.show()
```

Listing 3: Code for Problem 3

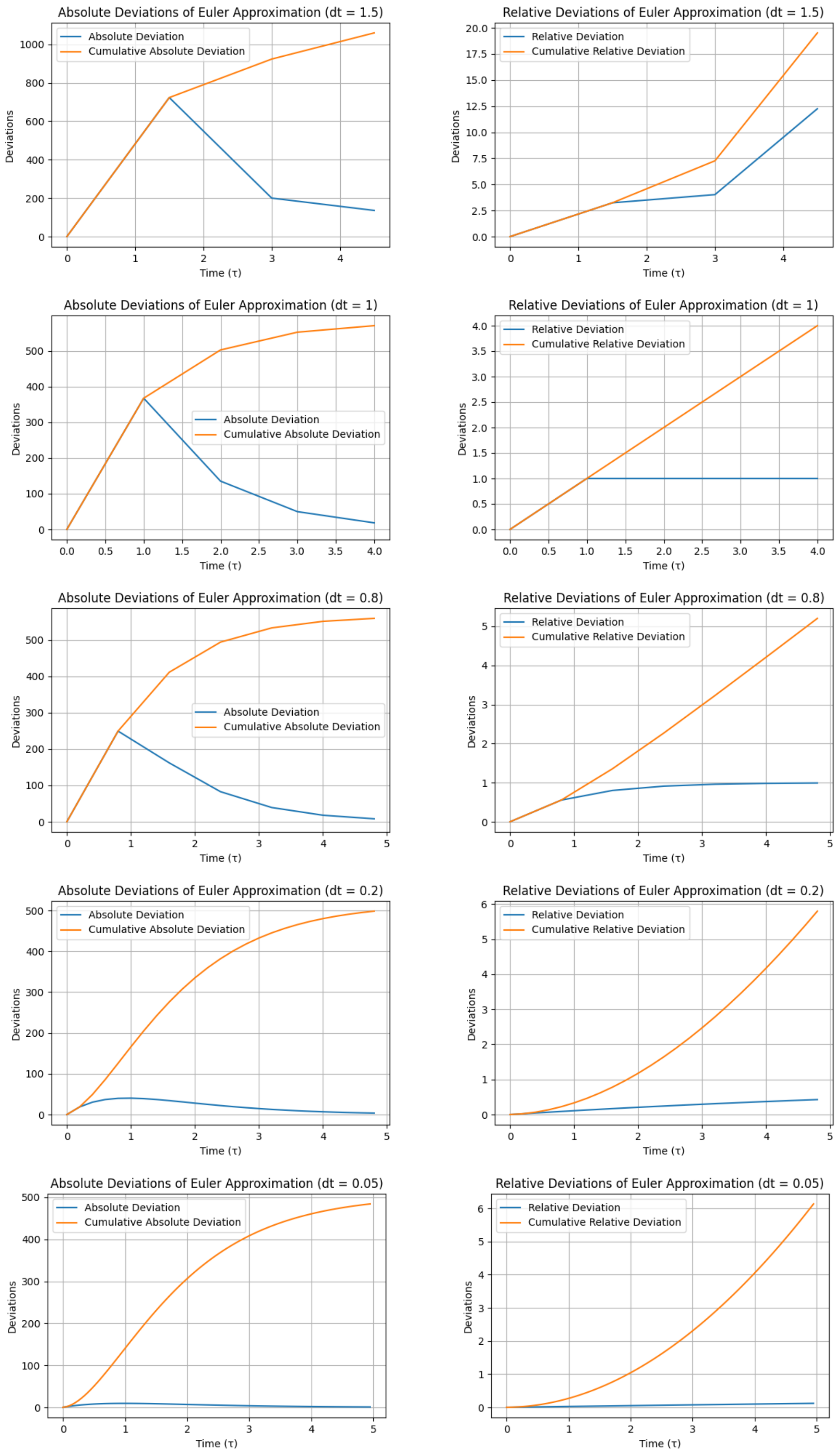

Figure 3: Plots the exact solution and Euler approximation for a certain value of $\Delta t/\tau$.

Problem 4 - Runge-Kutta Methods

Second-Order Runge-Kutta (RK2)

The RK2 method uses

$$\begin{aligned} x_{i+1}^* &= x_i + f(x_i, t_i)\frac{\Delta t}{2} \\ x_{i+1} &= x_i + f\left(x_{i+1}^*, t_i + \frac{\Delta t}{2}\right)\Delta t \end{aligned}$$

Fourth-Order Runge-Kutta (RK4)

The RK4 method computes

$$\begin{aligned} f_1^{(i)} &= f\left(x_i, t_i\right) \\ f_2^{(i)} &= f\left(x_i + f_1^{(i)}\frac{\Delta t}{2}, t_i + \frac{\Delta t}{2}\right) \\ f_3^{(i)} &= f\left(x_i + f_2^{(i)}\frac{\Delta t}{2}, t_i + \frac{\Delta t}{2}\right) \\ f_4^{(i)} &= f\left(x_i + f_3^{(i)}\Delta t, t_i + \Delta t\right) \end{aligned}$$

With the update

$$x_{i+1} = x_i + \frac{\Delta t}{6}\left(f_1^{(i)} + 2f_2^{(i)} + 2f_3^{(i)} + f_4^{(i)}\right).$$

```python
def f(x, t):
    return -x


def RK2(x0, tau, dt):
    times = np.arange(0, 5 * tau, dt)
    nsteps = len(times)
    x = np.zeros(nsteps)
    x[0] = x0

    for i in range(nsteps - 1):
        x_half = x[i] + f(x[i], times[i]) * dt / 2.0
        x[i + 1] = x[i] + f(x_half, times[i] + dt / 2.0) * dt

    return x, times


def RK4(x0, tau, dt):
    times = np.arange(0, 5 * tau, dt)
    nsteps = len(times)
    x = np.zeros(nsteps)
    x[0] = x0

    for i in range(nsteps - 1):
        f1 = f(x[i], times[i])
        f2 = f(x[i] + f1 * dt / 2.0, times[i] + dt / 2.0)
        f3 = f(x[i] + f2 * dt / 2.0, times[i] + dt / 2.0)
        f4 = f(x[i] + f3 * dt, times[i] + dt)
        x[i + 1] = x[i] + (f1 + 2 * f2 + 2 * f3 + f4) * dt / 6

    return x, times


# Define empty storage lists
N_RK2, t_RK2 = [], []
N_RK4, t_RK4 = [], []


# loop for different dt
for dt in dt_values:
    # Exact Solution
    t_exact_i = np.arange(0, 5 * tau, dt)
    N_exact_i = exact_solution(t_exact_i, N0, tau)
    t_exact.append(t_exact_i)
    N_exact.append(N_exact_i)
    plt.plot(t_exact_i, N_exact_i, label=f"Exact Solution for {dt}")

    # RK2
    N_RK2_i, t_RK2_i = RK2(N0, tau, dt)
    N_RK2 += [N_RK2_i]
    t_RK2 += [t_RK2_i]
    plt.plot(t_RK2_i, N_RK2_i, label=f"$\\ Delta t=${dt}")

    # RK4
    N_RK4_i, t_RK4_i = RK4(N0, tau, dt)
    N_RK4 += [N_RK4_i]
    t_RK4 += [t_RK4_i]
    plt.plot(t_RK4_i, N_RK4_i, label=f"$\\ Delta t=${dt}")

    # plot everything
    plt.title("Runge-Kutta Method for Radioactive Decay")
    plt.xlabel("Time ()")
    plt.ylabel("Number of Atoms (N)")
    plt.legend()
    plt.savefig(f"./imgs/Problem 4 - {dt}.png")
    plt.show()
```

Listing 4: Code for Problem 2

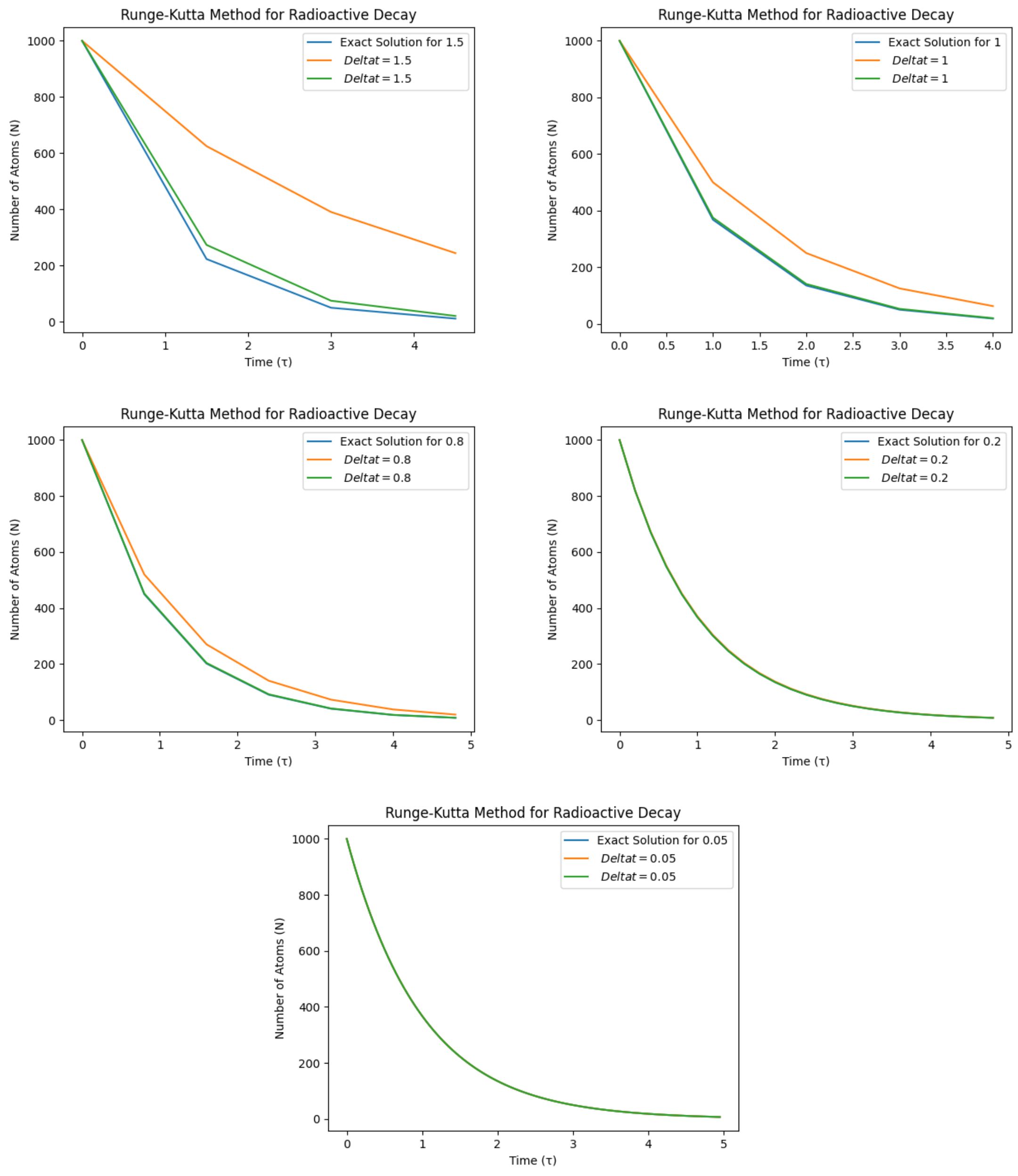

Figure 4: Plots the exact solution and Runge-Kutta approximations for a certain value of $\Delta t/\tau$.

```python
def calculate_relative_error(numerical, exact):
    with np.errstate(divide="ignore", invalid="ignore"):
        relative_error = np.abs(numerical - exact) / np.abs(exact)
        relative_error = np.nan_to_num(
            relative_error, nan=0.0, posinf=1e10, neginf=-1e10
        )
    return relative_error


for dt in dt_values:
    # Calculate solutions
    t_euler, N_euler = euler_approximation(N0, tau, dt)
    t_rk2, N_rk2 = RK2(N0, tau, dt)
    t_rk4, N_rk4 = RK4(N0, tau, dt)

    # Calculate exact solutions
    N_exact_euler = exact_solution(t_euler, N0, tau)
    N_exact_rk2 = exact_solution(t_rk2, N0, tau)
    N_exact_rk4 = exact_solution(t_rk4, N0, tau)

    # Calculate relative errors with safety checks
    err_euler = calculate_relative_error(N_euler, N_exact_euler)
    err_rk2 = calculate_relative_error(N_rk2, N_exact_rk2)
    err_rk4 = calculate_relative_error(N_rk4, N_exact_rk4)

    # Create semi-log plot
    plt.figure(figsize=(10, 6))
    plt.plot(t_euler, err_euler, "r-", label="Euler", linewidth=2)
    plt.plot(t_rk2, err_rk2, "g--", label="RK2", linewidth=2)
    plt.plot(t_rk4, err_rk4, "b:", label="RK4", linewidth=2)

    plt.title(f"Relative Error Comparison (t = {dt})")
    plt.xlabel("t/")
    plt.ylabel("Relative Error")
    plt.grid(True, which="both", ls="-", alpha=0.2)
    plt.legend()
    plt.tight_layout()
    plt.savefig(f"imgs/Problem 5 - {dt}.png", dpi=300, bbox_inches="tight")
    plt.close()
```

Listing 5: Code for Problem 5

Comparison and Discussion

Our analysis reveals several key findings:
1. Error scaling follows theoretical predictions for small step sizes:

- Euler: $\mathcal{O}(\Delta t)$
- RK2: $\mathcal{O}(\Delta t^2)$
- RK4: $\mathcal{O}(\Delta t^4)$

2. For larger step sizes ($\Delta t \geq 0.8$), we observe deviations from expected behavior, likely due to higher-order terms in the Taylor expansion becoming significant.
3. RK4 consistently provides superior accuracy across all step sizes, with relative errors below 1% for $\Delta t \leq 0.2$.

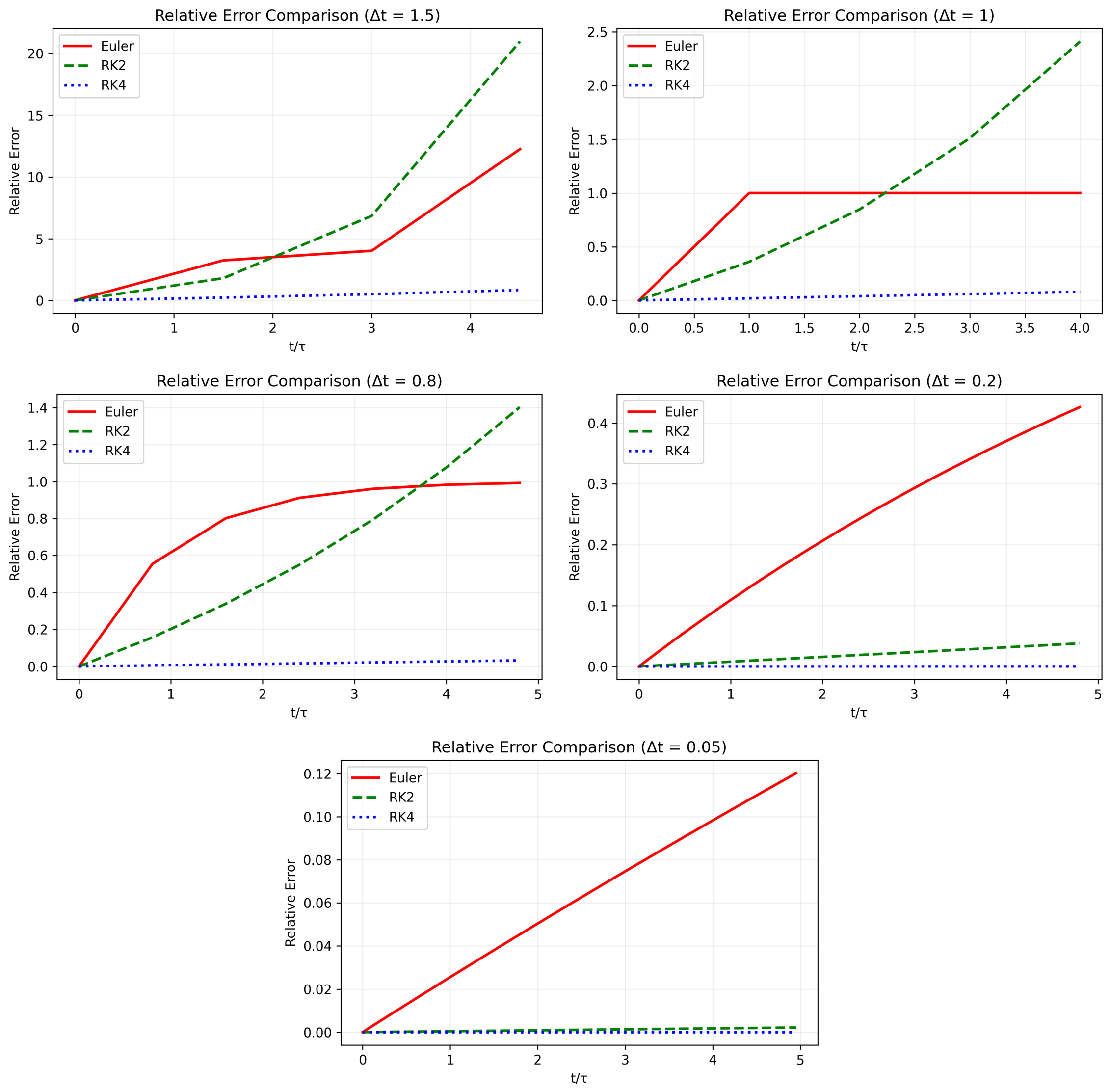

Figure 5: Plot comparison between different numerical methods for radioactive decay.