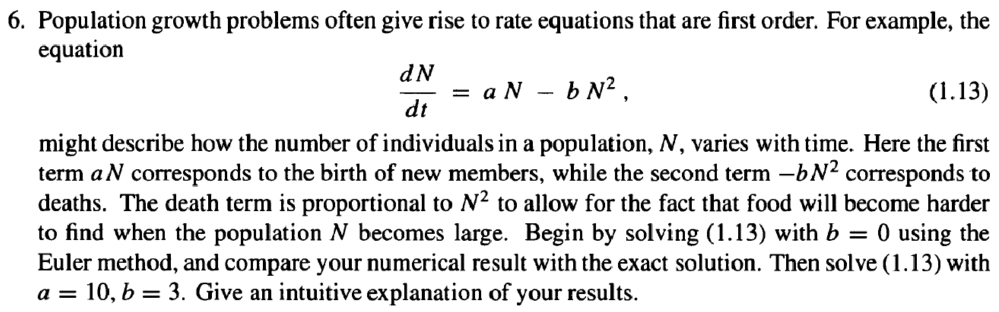
Computational Physics Assignment-1 Euler's Approximantion Method

140310220048-Muhammad Arief Mulyana
Lecturer: Dr. Budi Adiperdana

March 2024

1 Giordano problem 13-14 number 6

1.1 Defining the problem


6. Population growth problems often give rise to rate equations that are first order. For example, the equation

$$\frac{dN}{dt} = a\,N - b\,N^2\,, \tag{1.13}$$

might describe how the number of individuals in a population, N, varies with time. Here the first term aN corresponds to the birth of new members, while the second term $-bN^2$ corresponds to deaths. The death term is proportional to N^2 to allow for the fact that food will become harder to find when the population N becomes large. Begin by solving (1.13) with $b = 0$ using the Euler method, and compare your numerical result with the exact solution. Then solve (1.13) with $a = 10$, $b = 3$. Give an intuitive explanation of your results.

Figure 1: Source: Giordano p. 13-14 number 6.

From the problem in Figure 1 we can know that population growth can be represented by mathematical equations, especially give rise to rate equations that are first order. For example, the equation on above is

$$\frac{dN}{dt} = a\,N - b\,N^2. \tag{1}$$

From equation (1), have been stated on the problem that $a\,N$ corresponds to birth of new members. So, a is the constant that will represent the birth value on population growth. The opposite, from the equation (1), we can know that have been stated too on the problem that $-b\,N^2$ corresponds to deaths. So, b is the constant that will represent the death value on population growth. That can be happened due to the N^2 is proportional to death term, allowing them for the fact that food will become harder to find when the population become large.

1.2 Changing the first order equation to numerical approximation using Euler's method

From the problem above there's 2-conditions, which are

1. Solving the problem by using $b = 0$ so the equation will be

$$\frac{dN}{dt} = a\,N - b\,N = a\,N - (0)\,N$$

$$\frac{dN}{dt} = a\,N. \tag{2}$$

2. After that, solving the equation (1) with $a = 10$ & $b = 3$.

From the conditions above, we can change the equation to numerically form. The equation that will fit the first condition numerically is the equation (3). Furthermore, the equation that will fit the second condition is the equation (4).

$$dN \approx aN(t) \cdot dt, \tag{3}$$

$$dN \approx (aN(t) - bN(t)^2) \cdot dt. \tag{4}$$

After that we can solve the first order differentiation equation with the equation (5).

$$N(t + dt) = N(t) + dN \tag{5}$$

1.3 Flowchart

Algorithm 1: Numerical and Analytical Solution for the given problem

```
Data: Parameters $a$, $b$, $N_0$, $t_0$, $t_{\text{final}}$, $dt$
Result: $t$ values, $N$ values
euler_method($N_0$, $t_0$, $t_{final}$, $dt$) Initialize $N$ values list with $N_0$ ;
Initialize $t$ values list with $t_0$ ;
$N \leftarrow N_0$ ;
$t \leftarrow t_0$ ;
while $t < t_{final}$ do
    Calculate $\frac{dN}{dt} = a \cdot N - b \cdot N^2$ ;
    Update $N$ ($N \leftarrow N + \frac{dN}{dt} \cdot dt$) ;
    Update $t$ ($t \leftarrow t + dt$) ;
    Append $N$ to $N$ values list ;
    Append $t$ to $t$ values list ;
end
return $t$ values, $N$ values ;
analytical_solution($t$ values) Calculate $N_{\text{analytical}}$ using analytical
solution formula;
return $N_{analytical}$ ;
$t$ values, $N_{\text{numerical}} \leftarrow$ euler_method($N_0$, $t_0$, $t_{final}$, $dt$) ;
$N_{\text{analytical}} \leftarrow$ analytical_solution($t$ values) ;
```

Algorithm 2: Main Algorithm

```
Data: Parameters $a$, $b$, $N_0$, $t_0$, $t_{\text{final}}$, $dt$
Result: Plot of the results
MainAlgorithm$a$, $b$, $N_0$, $t_0$, $t_{\text{final}}$, $dt$ $t$ values, $N_{\text{numerical}} \leftarrow$
euler_method($N_0$, $t_0$, $t_{final}$, $dt$) ;
$N_{\text{analytical}} \leftarrow$ analytical_solution($t$ values) ;
Plot $t$ values vs $N_{\text{numerical}}$ (Numerical Solution) ;
Plot $t$ values vs $N_{\text{analytical}}$ (Analytical Solution) ;
```

1.4 Python code for solving the problem numerically and analytically

1.4.1 Python code for the first condition

```
# Define the function for the differential equation
def f(N, a):
  return a * N  # Simplified equation

# Set initial conditions
N0 = 1  # Initial population
a = 2  # Growth rate
t_end = 10  # End time
dt = 0.001  # Time step

# Initialize time and population arrays
t = np.arange(0, t_end + dt, dt)
N = np.zeros(len(t))
N[0] = N0  # Initial population

# Perform Euler approximation
for i in range(1, len(t)):
  N[i] = N[i-1] + f(N[i-1], a) * dt

# Plot the results
plt.plot(t, N, 'o', label="Euler␣Approximation")

# Analytical solution (for comparison)
def analytical_solution(N0, a, t):
  return N0 * np.exp(a * t)  # Exponential growth

plt.plot(t, analytical_solution(N0, a, t), label="
  Analytical␣Solution", linewidth='3')

plt.xlabel("Time␣(t)")
plt.ylabel("Population␣(N)")
plt.legend()
plt.grid(True)
plt.title("Graph␣of␣Birth␣of␣New␣Member␣Visualization")
plt.show()
```

1.4.2 Python code for the second condition

```
# Define parameters
a = 10  # Growth rate
b = 3  # Decay rate

# Initial condition
N0 = 0.1  # Initial population
t0 = 0.0  # Initial time

# Time parameters
t_final = 1  # Final time
dt = 0.01  # Time step size

# Analytical solution
def analytical_solution(t):
     return -a * N0 * np.exp(a * t) / (b * N0 * (1 - np.
       exp(a * t)) - 10)

# Euler method for numerical solution
def euler_method(N0, t0, t_final, dt):
    N_values = [N0]
    t_values = [t0]
    N = N0
    t = t0
    while t < t_final:
        dNdt = a * N - b * N**2
        N += dNdt * dt
        t += dt
        N_values.append(N)
        t_values.append(t)
    return np.array(t_values), np.array(N_values)

# Compute numerical solution
t_values, N_numerical = euler_method(N0, t0, t_final, dt
  )

# Compute analytical solution
N_analytical = analytical_solution(t_values)
```

```
# Plot results
plt.plot(t_values, N_numerical,'o', label='Numerical␣
  Solution␣(Euler␣Method)')
plt.plot(t_values, N_analytical, label='Analytical␣
  Solution', linewidth='3')
plt.xlabel('Time')
plt.ylabel('Population')
plt.title('Population␣Visualization␣Respected␣to␣Birth␣
  and␣Death')
plt.legend()
plt.grid()
plt.show()
```

1.4.3 Analysis

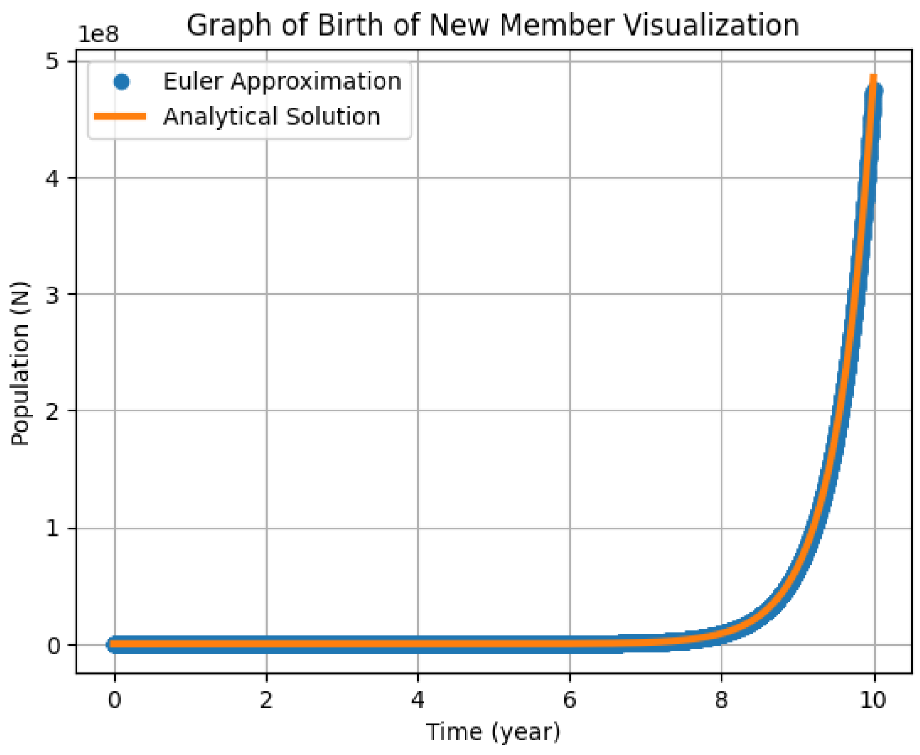


Figure 2: Solution of Condition-1 from problem in Figure 1

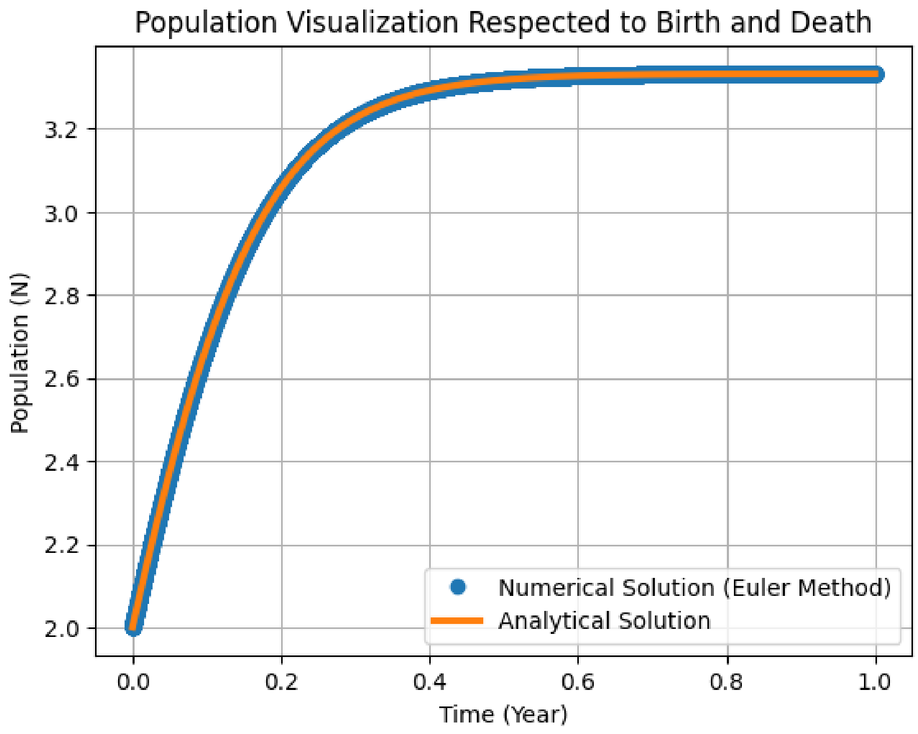


Figure 3: Solution of Condition-1 from problem in Figure 1

The analytical solution of the problem in Figure 1 can be written mathematically as equation (6) and (7).

$$N(t) = N_0 \cdot e^{at} \tag{6}$$

$$N(t) = \frac{-a\, N_0\, e^{at}}{b\, N_0\, (1 - e^{at}) - 10}. \tag{7}$$

Now, from these analytical form we can compute it in python and plot the graphic of the solution, in order to compare it to numerical solution. Therefore, we can get the visualization of the problem in Figure 1 and it can be seen in Figure 2 and Figure 3.

The output of numerical and analytical solution of condition-1 fit perfectly as can be seen in Figure 2. Hence, the numerical solution in condition-1 is representative to its analytical. Moreover, we can know that the more each people give giving birth, the more population will be increased as can be describe from the graph in Figure 2.

However, it will be a different story if we try to compute the equation (1) with b values are not zero. As can be seen from equation (7), it's analytical solution of condition-2, more complex solution from its analytical. Therefore, the numerically expression is more complex too rather than condition-1 as can be seen from equation (4). Even though, the numerical solution still good as it still fits the analytical that can be seen in Figure 3. Furthermore, we can know from the Figure 3 that if the population continues to increase, the death factor will also increase. This can be happened because of the more population in an area, the more difficult to them to survive, because of they have to compete in terms of economy and they needs such us foods.

2 Giordano problem 13-14 number 3

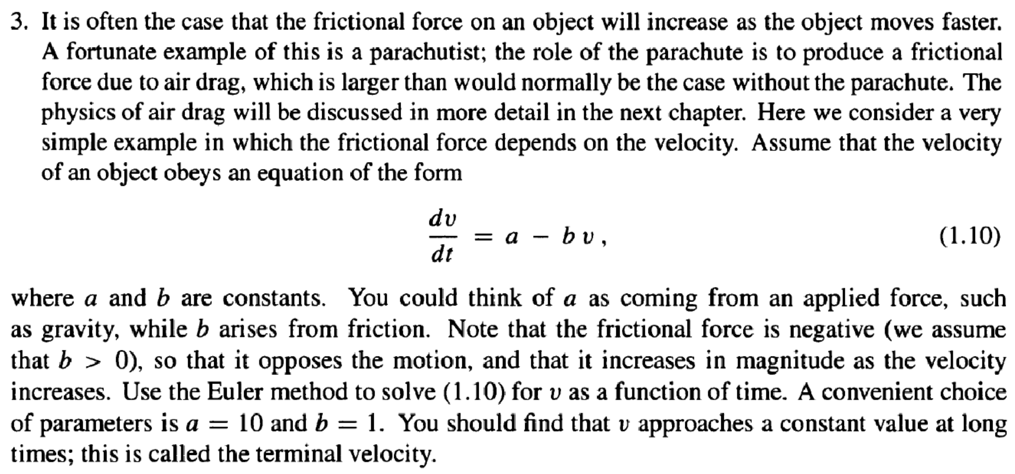
2.1 Defining the problem

The problem that focused in Figure 2, is one of the physics basics problem, it's the phenomena of the first derivative of velocity, which is the acceleration. In real physics condition, the faster an object that moves, it will increase the frictional force that occur in the object.

The simple example of this phenomena is a parachutist. The role of the parachute is to produce a frictional force due to air drag, which is larger than would normally the case without the parachute. This air drag, mathematically can be described by the simple equation (8).

$$\frac{dv}{dt} = a - b\,v. \tag{8}$$

From the equation (8), a & b are constants. We can consider the a is coming from the applied force, such as gravity, while b arises from friction that occur in a phenomenon of this physics. Because the b constants are arise from the frictional force, the vector is negative, so that in equation (8) there's a negative v that represent the factor of reduction as an object moving faster.



3. It is often the case that the frictional force on an object will increase as the object moves faster. A fortunate example of this is a parachutist; the role of the parachute is to produce a frictional force due to air drag, which is larger than would normally be the case without the parachute. The physics of air drag will be discussed in more detail in the next chapter. Here we consider a very simple example in which the frictional force depends on the velocity. Assume that the velocity of an object obeys an equation of the form

$$\frac{dv}{dt} = a - b\,v\,, \tag{1.10}$$

where a and b are constants. You could think of a as coming from an applied force, such as gravity, while b arises from friction. Note that the frictional force is negative (we assume that $b > 0$), so that it opposes the motion, and that it increases in magnitude as the velocity increases. Use the Euler method to solve (1.10) for v as a function of time. A convenient choice of parameters is $a = 10$ and $b = 1$. You should find that v approaches a constant value at long times; this is called the terminal velocity.

Figure 4: Source: Giordano p. 13-14 number 3.

2.2 Changing the first order equation to numerical approximation using Euler's method

The equation (8) can be compute numerically by changing the form of equation. The equation (9) bellow is the numerically form of equation (8).

$$dv \approx (a - b\,v(t)) \cdot dt, \tag{9}$$

the numerical solution of equation (9) can be determined by substitute the value of dv to equation (10)

$$v(t + dt) = v(t) + dv \tag{10}$$

2.3 Flowchart

Algorithm 3: Numerical and Analytical Solution for the given problem

```
Data: Parameters $a$, $b$, $v_0$, $t_0$, $t_{\text{final}}$, $dt$
Result: $t$ values, $v$ values

euler_method($a$, $b$, $v_0$, $t_0$, $t_{final}$, $dt$) Initialize $v$ values list with $v_0$ ;
Initialize $t$ values list with $t_0$ ;
$v \leftarrow v_0$ ;
$t \leftarrow t_0$ ;
while $t < t_{final}$ do
    Calculate $\frac{dv}{dt} = a - b \cdot v$ ;
    Update $v$ ($v \leftarrow v + \frac{dv}{dt} \cdot dt$) ;
    Update $t$ ($t \leftarrow t + dt$) ;
    Append $v$ to $v$ values list ;
    Append $t$ to $t$ values list ;
end
return $t$ values, $v$ values ;

analytical_solution($a$, $b$, $v_0$, $t$ values) Calculate
$v_{\text{analytical}} = \frac{a}{b} \cdot (1 - e^{-b \cdot t}) + v_0 \cdot e^{-b \cdot t}$ ;
return $v_{analytical}$ ;

$t$ values, $v$ values $\leftarrow$ euler_method($a$, $b$, $v_0$, $t_0$, $t_{final}$, $dt$) ;
$v_{\text{analytical}} \leftarrow$ analytical_solution($a$, $b$, $v_0$, $t$ values) ;
```

Algorithm 4: Main Algorithm

```
Data: Parameters $a$, $b$, $v_0$, $t_0$, $t_{\text{final}}$, $dt$
Result: Plot of the results

MainAlgorithm$a$, $b$, $v_0$, $t_0$, $t_{\text{final}}$, $dt$ $t$ values, $v$ values $\leftarrow$
euler_method($a$, $b$, $v_0$, $t_0$, $t_{final}$, $dt$) ;
$v_{\text{analytical}} \leftarrow$ analytical_solution($a$, $b$, $v_0$, $t$ values) ;
Plot $t$ values vs $v$ values (Numerical Solution) ;
Plot $t$ values vs $v_{\text{analytical}}$ (Analytical Solution) ;
```

2.4 Python code for solving the problem numerically and analytically

```
# Define parameters
a = 10  # Constant coefficient
b = 1  # Constant coefficient

# Initial condition
v0 = 1.0  # Initial velocity
t0 = 0.0  # Initial time

# Time parameters
t_final = 10.0  # Final time
dt = 0.01  # Time step size

# Euler method for numerical solution
def euler_method(a, b, v0, t0, t_final, dt):
    v_values = [v0]
    t_values = [t0]
    v = v0
    t = t0
    while t < t_final:
        dvdt = a - b * v
        v += dvdt * dt
        t += dt
        v_values.append(v)
        t_values.append(t)
    return np.array(t_values), np.array(v_values)

# Analytical solution
def analytical_solution(a, b, v0, t_values):
    v_analytical = (a / b) * (1 - np.exp(-b * t_values))
      + v0 * np.exp(-b * t_values)
    return v_analytical

# Compute numerical solution
t_values, v_values = euler_method(a, b, v0, t0, t_final,
   dt)

# Compute analytical solution
```

```
v_analytical = analytical_solution(a, b, v0, t_values)

# Plot the result
plt.plot(t_values, v_values, 'o', label='Numerical␣
  Solution␣(Euler␣Method)')
plt.plot(t_values, v_analytical, label='Analytical␣
  Solution', linewidth='3')
plt.xlabel('Time')
plt.ylabel('Velocity␣(v)')
plt.title('Visualization␣of␣Acceleration␣with␣Friction␣
  Factor')
plt.legend()
plt.grid(True)
plt.show()
```

2.5 Analysis

From the problem in Figure 4, when an object accelerates, its speed increases, and therefore, the force of air friction also increases. This happens because as the object moves faster, it encounters more air molecules per unit time, resulting in a greater force of air friction. Additionally, the surface area and shape of the object may influence air friction; for example, objects with larger surface areas or less aerodynamic shapes tend to experience greater air friction.

Therefore, as acceleration increases, the force of air friction also increases, eventually reaching a point where the force of air friction balances the force causing acceleration (such as the force applied by an engine or gravity). At this point, known as terminal velocity, the object stops accelerating and continues to move at a constant speed. This can be visualized from the Figure 5, which is the solution of the problem in Figure 4. Furthermore from the Figure 5 we can see that the numerical and analytical solution fit perfectly, so we can say that the numerical form in this problem represented that its analytical.

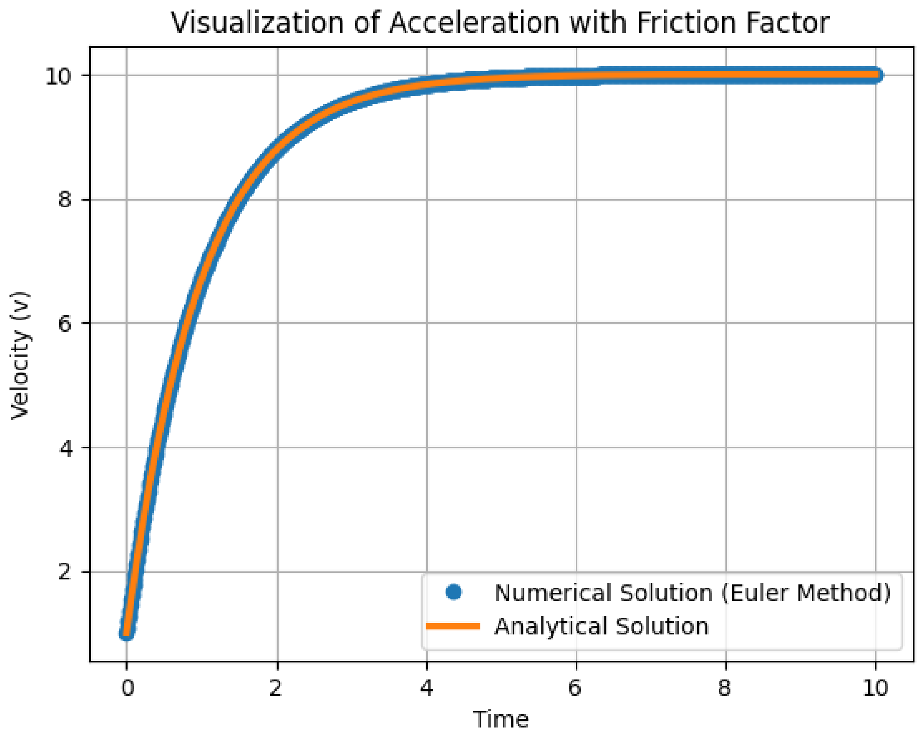


Figure 5: Solution of problem in Figure 4

3 3-types of radioactive nuclear decay

3.1 Defining the problem

This problem was made by my lecturer of this subject, Dr. Budi Adiperdana. In this problem we must calculate the solution of 3-types of radioactive nuclear decay numerically. This problem is a problem that 3-types of nuclei will decay, but are coupled to each other. The equation of this decay represented by equation (11), (12), & (13).

$$\frac{dN_A}{dt} = -\frac{N_A}{\tau_A} \tag{11}$$

$$\frac{dN_B}{dt} = \frac{N_A}{\tau_A} - \frac{N_B}{\tau_B} \tag{12}$$

$$\frac{dN_C}{dt} = \frac{N_A}{\tau_A} + \frac{N_B}{\tau_B} - \frac{N_C}{\tau_C} \tag{13}$$

From these equation above we can know that the decay of N_A is not been affected by any other factors. However, the decay of N_B is coupled by the decay of N_A and the decay of N_C are coupled by the decay of N_A & N_B. The Conditions of equation (11), (12), & (13) will be set at these conditions:

1. $\tau_A = 0.(X + 1)$,
2. $\tau_B = 0.(Y + 1)$,
3. when $\tau_C = \tau_A + 0.2$,
4. when $\tau_C = \tau_A + 0.2$, but if the value of $\tau_C < 0$, set the $\tau_A = 0.(X + 2)$

The X of Y values will be obtained by using two last digits of our NPM. For example my NPM is 140310220048. So, my NPM can also be written as 1403102200XY, with X = 4 and Y = 8.

3.2 Changing the first order equation to numerical approximation using Euler's method

The numerical expression of equation (11) is

$$dN_A \approx \frac{N_A(t)}{\tau_A} \cdot dt, \tag{14}$$

the numerical solution of equation (11) is

$$N_A(t + dt) = N_A(t) + dN_A. \tag{15}$$

Now, the numerical expression of equation (12) is

$$dN_B \approx (\frac{N_A}{\tau_A} - \frac{N_B}{\tau_B}) \cdot dt, \tag{16}$$

and the numerical solution of equation (12) is

$$N_B(t+dt) = N_B(t) + dN_B. \tag{17}$$

Last, the numerical expression of equation (13) is

$$dN_C \approx (\frac{N_A}{\tau_A} + \frac{N_B}{\tau_B} - \frac{N_C}{\tau_C}) \cdot dt \tag{18}$$

and its numerical solution is

$$N_C(t+dt) = N_C(t) + dN_C. \tag{19}$$

3.3 Flowchart

Algorithm 5: Euler Method

```
Function euler_method($N_{A0}$, $N_{B0}$, $N_{C0}$, ..., $dt$):
    Data: Decay constants $\tau_A$, $\tau_B$, $\tau_C$; Initial populations $N_{A0}$, $N_{B0}$,
          $N_{C0}$; Initial time $t0$; Final time $t_{\text{final}}$; Time step size $dt$
    Result: $t$ values, $N_A$ values, $N_B$ values, $N_C$ values
    Calculate num_steps;
    Initialize $t$ values, $N_A$ values, $N_B$ values, $N_C$ values;
    Initialize $N_A$, $N_B$, $N_C$ with initial values;
    while $i$ ¡ num_steps + 1 do
        if $i$ is odd then
            Store current populations;
            $N_{A\_values}[i] = N_A$;
            $N_{B\_values}[i] = N_B$;
            $N_{C\_values}[i] = N_C$;
        else
            Calculate rate of change ($dN_{A\_dt}$, $dN_{B\_dt}$, $dN_{C\_dt}$);
            Update populations ($N_A$, $N_B$, $N_C$);
        end
    end
    return $t$ values, $N_A$ values, $N_B$ values, $N_C$ values;
```

Algorithm 6: Main Process

```
Main Process:
    Data: Decay constants $\tau_A$, $\tau_B$, $\tau_C$; Initial populations $N_{A0}$, $N_{B0}$,
          $N_{C0}$; Initial time $t0$; Final time $t_{\text{final}}$; Time step size $dt$
    Result: $t$ values, $N_A$ values, $N_B$ values, $N_C$ values
    Define $\tau_A$, $\tau_B$, $\tau_C$, $N_{A0}$, $N_{B0}$, $N_{C0}$, $t0$, $t_{\text{final}}$, $dt$;
    Call euler_method($N_{A0}$, $N_{B0}$, $N_{C0}$, ..., $dt$);
    Obtain $t$ values, $N_A$ values, $N_B$ values, $N_C$ values;
end
```

3.4 Python code for solving the problem numerically and analytically

```
tau_A = 0.5 #0.(X+1)
tau_B = 0.9 #0.(Y+1)
tau_C = tau_A-0.2 #or tau_C = tau_A+0.2

N_A0 = 1.0
N_B0 = 0.0
N_C0 = 0.0

t0 = 0.0
t_final = 10.0
dt = 0.01

# Implement Euler method for numerical solution
def euler_method(N_A0, N_B0, N_C0, tau_A, tau_B, tau_C,
  t0, t_final, dt):
    num_steps = int((t_final - t0) / dt)
    t_values = np.linspace(t0, t_final, num_steps + 1)
    N_A_values = np.zeros(num_steps + 1)
    N_B_values = np.zeros(num_steps + 1)
    N_C_values = np.zeros(num_steps + 1)

    N_A = N_A0
    N_B = N_B0
    N_C = N_C0
    for i in range(num_steps + 1):
        N_A_values[i] = N_A
        N_B_values[i] = N_B
        N_C_values[i] = N_C

        dN_A_dt = -N_A / tau_A
        dN_B_dt = N_A / tau_A - N_B / tau_B
        dN_C_dt = N_A / tau_A + N_B / tau_B - N_C /
          tau_C

        N_A += dN_A_dt * dt
        N_B += dN_B_dt * dt
        N_C += dN_C_dt * dt

    return t_values, N_A_values, N_B_values, N_C_values

# Compute numerical solution using Euler method
t_values, N_A_values, N_B_values, N_C_values =
  euler_method(N_A0, N_B0, N_C0, tau_A, tau_B, tau_C, t0
  , t_final, dt)

#Analytical Solution

N_A_analytical = N_A0 * np.exp(-t/tau_A)
N_B_analytical = N_B0 * np.exp(-t/tau_B) + (tau_B/(tau_A
   - tau_B)) * N_A0 * (np.exp(-t/tau_A) - np.exp(-t/
  tau_B))
N_C_analytical = N_C0 * np.exp(-t/tau_C) + (tau_C/(tau_A
   - tau_C)) * N_A0 * (np.exp(-t/tau_A) - np.exp(-t/
  tau_C)) + (tau_C/(tau_B - tau_C)) * N_B0 * (np.exp(-t/
  tau_B) - np.exp(-t/tau_C))

# Plot the results
plt.plot(t_values, N_A_values, 'o', label='$N_A$')
plt.plot(t_values, N_B_values, 'o', label='$N_B$')
plt.plot(t_values, N_C_values, 'o', label='$N_C$')
plt.plot(t_values, N_A_analytical, label='$N_A$␣analytic
  ', linewidth='3')
plt.plot(t_values, N_B_analytical, label='$N_B$␣analytic
  ', linewidth='3')
```

```
plt.plot(t_values, N_C_analytical, label='$N_C$␣analytic
  ', linewidth = '3')
plt.xlabel('Time')
plt.ylabel('$N$')
plt.title('3-Types␣of␣Radioactive␣Nuclear␣Decay␣Chain')
plt.legend()
plt.grid(True)
plt.show()
```

3.5 Analysis

The solutions of the problem in this section are visualized the 3-types of radioactive nuclear decay chain that are coupled to each other. As can be seen in Figure 6 and Figure 7. In the figure we can know that the N_A decays straightly because the N_A is not is unaffected by any other factor. The opposite the N_B and N_C is affected by other factor, the N_B is coupled to N_A and the N_C is coupled to N_A and N_B. Consequently, the N_B and N_C increasing first before they decays as can be seen in Figure 6 and Figure 7.

We can also see that the numerical solutions were near to its analytical, especially for the problem when $\tau_C = \tau_A - 0.2$. However, when it comes to $\tau_C = \tau_A + 0.2$, the numerical solution, especially the solution for N_C still far to represent its analytical solution. So, in conclusion the Euler's method for this problem start to breakdown when $\tau_C = \tau_A + 0.2$.

Despite, the explanation above, this problem require an analysis further. This can be happen because the problem was made purely by my dear lecturer Dr. Budi Adiperdana. Nevertheless, the 3-types of radioactive nuclear decay chain in real world case have occurred. So, this problem can be realistically happens, but require more specific definition of N_A, N_B, and N_C to fit the case realistically.

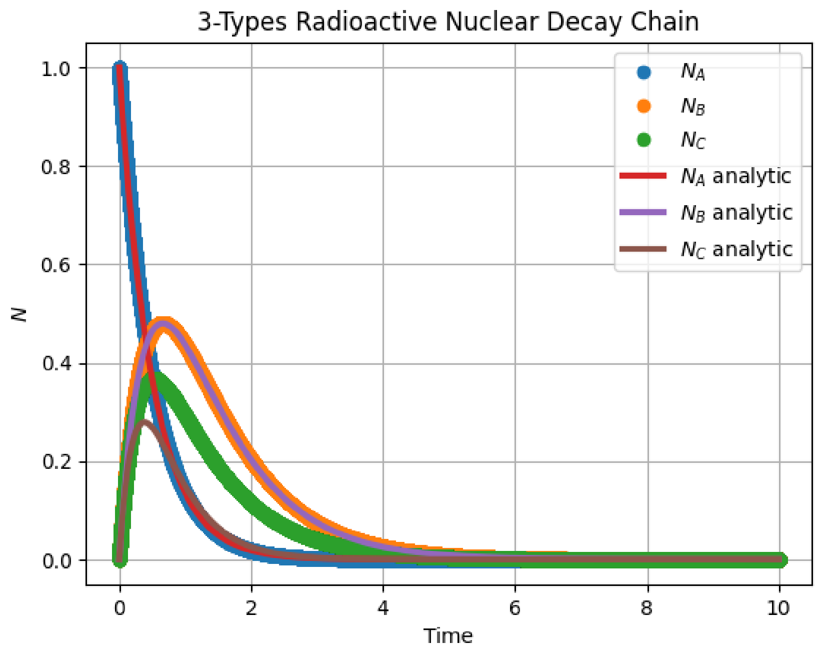


Figure 6: 3-types of radioactive nuclear decay chain when $\tau_C = \tau_A - 0.2$

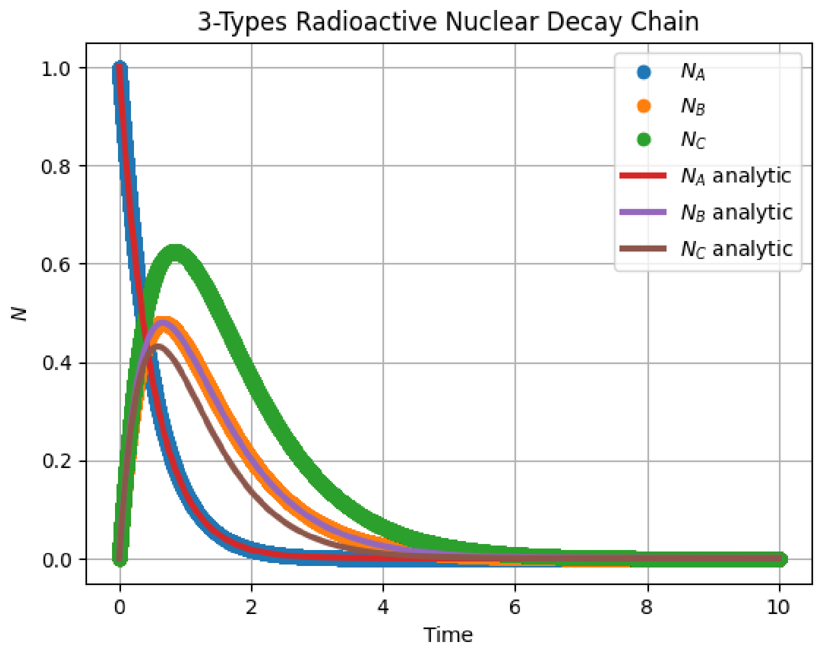


Figure 7: 3-types of radioactive nuclear decay chain when $\tau_C = \tau_A + 0.2$