PHYS2600: Homework 1*

Due on Tuesday, Feb, 8th 2018

Yangrui Hu

*This Latex template is from http://www.latextemplates.com

Contents

Problem 1

Problem: The velocity of a freely falling object near Earth's surface is described by the equation

$$\frac{dv}{dt} = g, \tag{1}$$

where v is the velocity and $g = 9.8m/s^2$. Write a program that uses the Euler method to compute the solution to equation 1. For simplicity, assume that the object begins from rest and calculate the solution for times from $t = 0$ to $t = 10s$. Repeat the calculation for several different values of the time step, and compare the results with the exact solution to equation 1. Verify numerically that in this case the Euler method gives the exact result and then prove it analytically.

Solution: This problem is to ask me to solve the equation 1 using the Euler method with the initial condition $v(t = 0) = 0$. The original script is shown as Listing 1. The analytical solution of equation 1 is

$$v(t) = v(0) - gt \tag{2}$$

Repeat calculations for several different values of the time step: $dt = 0.1,\ 0.01,\ 0.05,\ 0.005$ and compare the numerical results and analytical results. Figure 1 shows the comparison of numerical and analytical results for this problem with different time step.

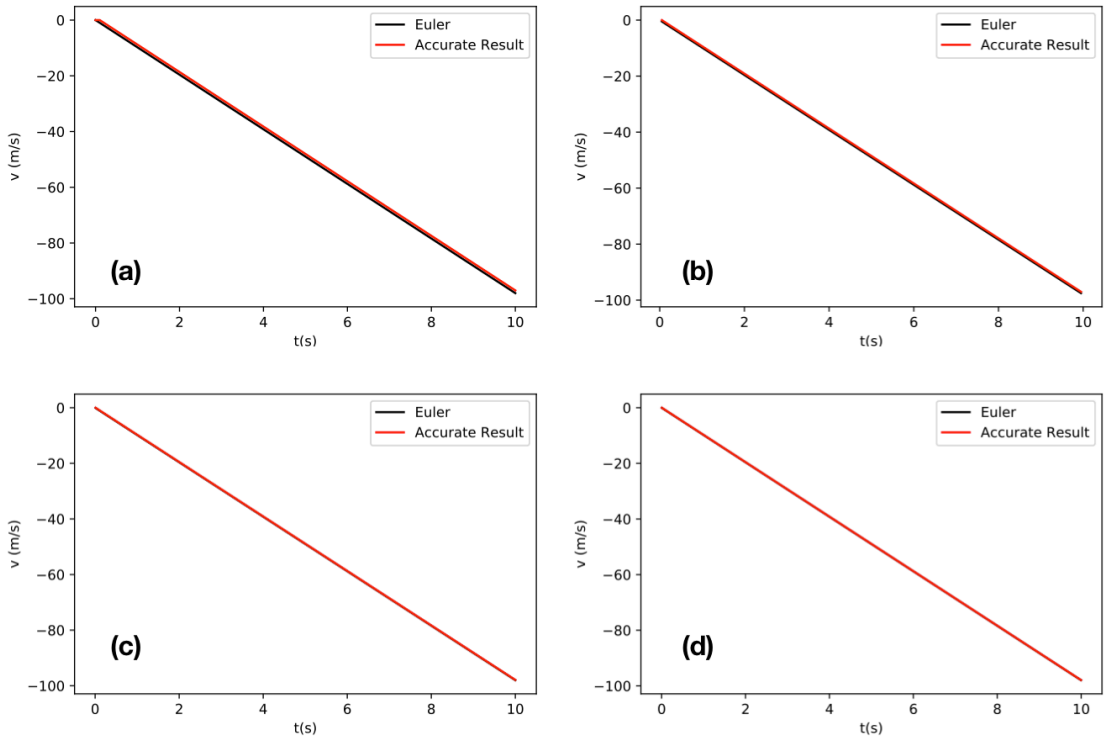

Figure 1: Numerical and analytical calculation results for problem 1: (a): $dt = 0.1$, (b): $dt = 0.05$, (c): $dt = 0.01$, (d): $dt = 0.005$

Discussion From Figure 1, we can see that as dt being smaller, the difference between accurate results and Euler method results becomes smaller. It shows that in this case, the Euler method gives the exact result. Analytical proof is shown as following: Suppose we have an general equation:

Problem 1 continued on next page...

$$\frac{dy}{dt} = f(t, y(t)) \tag{3}$$

The Euler method can be written as:

$$\frac{y(t_{n+1}) - y(t_n)}{dt} = f(t_n, y(t_n)) + \frac{dt}{2} y''(\xi_n) \tag{4}$$

$$y(t_{n+1}) = y(t_n) + f(t_n, y(t_n))dt + \frac{dt^2}{2} y''(\xi_n) \tag{5}$$

where ξ_n is an arbitrary value inside the range of (t_n, t_{n+1}). Then the error of the Euler method T_{n+1} is:

$$T_{n+1} = y(t_{n+1}) - y_{n+1} \tag{6}$$

$$= y(t_n) + f(t_n, y(t_n))dt + \frac{dt^2}{2} y''(\xi_n) - y_{n+1} \tag{7}$$

$$= y(t_n) + f(t_n, y(t_n))dt + \frac{dt^2}{2} y''(\xi_n) - y_n - dt f(t_n, y(t_n)) \tag{8}$$

$$= \frac{dt^2}{2} y''(\xi_n) \tag{9}$$

$|y''(\xi_n)|$ is finite, so when dt is small enough, the error of the Euler method is so small that we can regard the Euler method gives the exact result.

Listing 1: Original Script of Problem 1

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt


class Particle (object):

    def __init__(self, m = 1.0, v0=0.0,  tf = 10.0, dt = 0.01):
        self.m = m
        self.v = v0
        self.v1 = v0
        self.v0 = v0
        self.t = 0.0
        self.tf = tf
        self.dt = dt
        self.tarray = np.arange(0.0, tf+0.5*self.dt, self.dt)

    def F(self, y, v, t):
        # force on a free particle
        return 0.0

    def Euler_step(self): # increment position as before
        a = self.F(self.y, self.v, self.t) / self.m
        self.v += a * self.dt

        #accurate result of v for problem 1
        self.v1 = self.v0 + a * self.t
```

Problem 1 continued on next page...

```
        self.t += self.dt


class FallingParticle (Particle):

    g = 9.8

    def __init__(self, m = 1.0, v0=0.0, dt = 0.1, tf = 10.0):
        super().__init__(m, y0, v0, tf, dt)

    def F(self, x, v, t):
        return  (-1) * self.g * self.m



P2 = FallingParticle(v0 = 0.0, dt = 0.1)
t_list = []
v_list = []
v1_list = []

while ( P2.t < P2.tf ):
    v_list.append(P2.v)
    v1_list.append(P2.v1)
    t_list.append(P2.t)
    P2.Euler_step()


fig = plt.figure()
plt.plot(t_list,v_list,'k', label = "Euler")
plt.plot(t_list,v1_list,'r',label= "Accurate Result")
plt.ylabel('v (m/s)')
plt.xlabel('t(s)')
plt.legend()
plt.show()
fig.savefig('problem1-0.1.pdf')
```

Problem 2

Problem: It is often the case that the frictional or drag force on an object traveling through a medium will increase as an object moves faster. This is usually a reasonable approximation for an object under free fall in the earths atmosphere, where a frictional force due to air drag opposes the force of gravity resulting in a terminal velocity for free fall. Here, we consider a very simple example in which the frictional force depends linearly on the velocity. Assume that the velocity of an object in free fall obeys the equation

$$\frac{dv}{dt} = a - bv, \tag{10}$$

where a and b are constants. We assume that the frictional force opposes the motion and $b > 0$. Use the Euler method to solve equation 10 for v as a function of t. What is the behavior of $v(t)$ at long times?

Solution: This problem is to ask me to solve the first order differential equation 10 with $b > 0$. In order to simplify the situation, I just assume $a = b = 1.0$ and the initial velocity is zero in my calculation. Listing 2 shows my code. The analytical solution for equation 10 is

$$v(t) = \frac{bv(0) - a}{b} e^{-bt} + \frac{a}{b} \tag{11}$$

I also repeat calculations for different values of the time step: $dt = 0.1,\ 0.05,\ 0.01$ and compare the numerical results and analytical results. Figure 2 shows the comparison of numerical and analytical results for this problem with different time step.

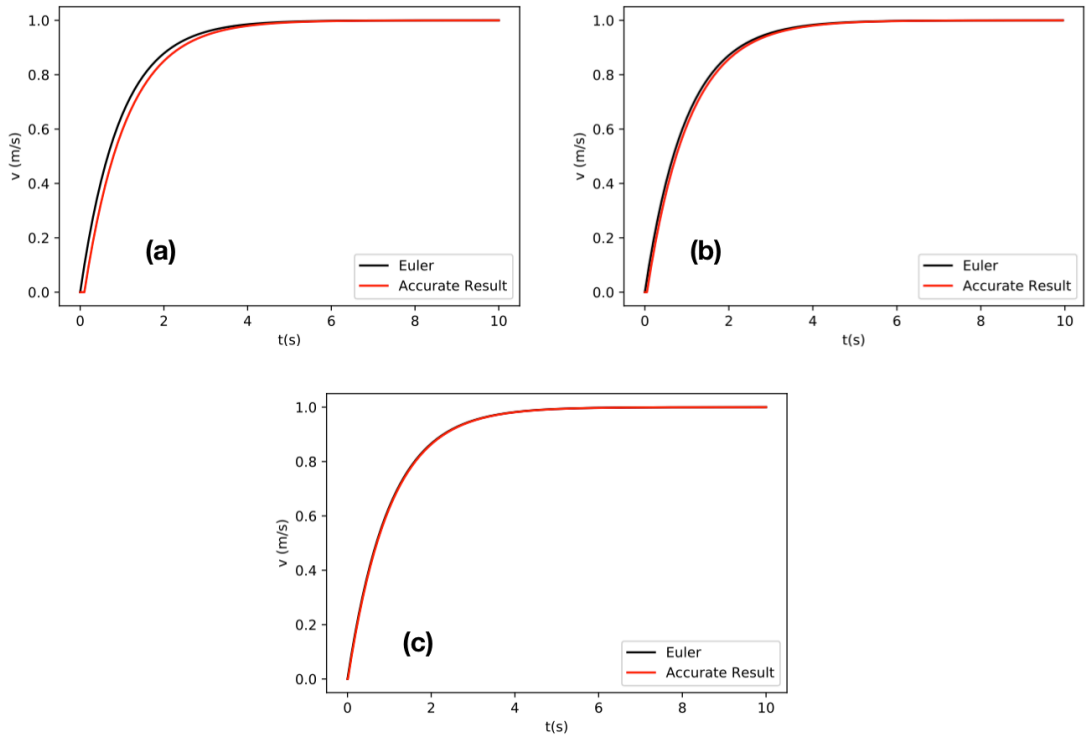

Figure 2: Numerical and analytical calculation results for problem 2: (a): $dt = 0.1$, (b): $dt = 0.05$, (c): $dt = 0.01$

Discussion From Figure 2, we can see that in this case, the Euler method also gives the exact result. After long times, $v(t)$ will approach to a fixed value : $v(+\infty) = a/b$. Here I set $a = b = 1.0$, so $v(+\infty) = 1.0$, which is consistent with the numerical result.

Listing 2: Original Script of Problem 2

```
# -*- coding: utf-8 -*-
import math
import numpy as np
import matplotlib.pyplot as plt

class FrictionalForce (object):

    a1 = 1.0
    b = 1.0

    def __init__(self, v0=0.0, dt = 0.1, tf = 10.0):
        self.v = v0
        self.v1 = v0
        self.v0 = v0
        self.t = 0.0
        self.tf = tf
        self.dt = dt
        self.tarray = np.arange(0.0, tf+0.5*self.dt, self.dt)

    def Euler_step(self):
     a = self.a1 - self.b * self.v
     self.v += a * self.dt
     self.v1 = (self.b * self.v0 - self.a1) *
        math.exp(-self.b * self.t) / self.b + self.a1 / self.b
        self.t += self.dt

P2 = FrictionalForce(v0 = 0.0, dt = 0.05)
t_list = []
v_list = []
v1_list = []

while ( P2.t < P2.tf ):
    v_list.append(P2.v)
    v1_list.append(P2.v1)
    t_list.append(P2.t)
    P2.Euler_step()

fig = plt.figure()
plt.plot(t_list,v_list,'k', label = "Euler")
plt.plot(t_list,v1_list,'r',label= "Accurate Result")
plt.ylabel('v (m/s)')
plt.xlabel('t(s)')
plt.legend()
plt.show()
fig.savefig('problem2-0.05.pdf')
```

Problem 3

Problem: Consider a radioactive decay involving two distinct types of nuclei, A and B, with populations $N_A(t)$ and $N_B(t)$. Suppose that the type A nuclei decay to form type B nuclei which decay in turn according to the differential equations

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

where τ_A and τ_B are the decay time constants for each type of nucleus. Use the Euler method to solve these coupled equations for N_A and N_B as functions of time. Compare the analytic solution of $N_A(t)$ and $N_B(t)$ with your numeric results. Explore the behavior for different values of the ratio τ_A/τ_B. Can you interpret the short and long time behaviors for different values of this ratio?

Solution: This problem is to solve two coupled equations 12 using Euler method. Listing 3 is my code for this problem. Because N_B is dependent on N_A while N_A is only dependent on itself, I update N_B first and then update N_A. The analytical solution for these two coupled equations are:

$$N_A(t) = N_A(0)e^{-t/\tau_A} \tag{13}$$

$$N_B(t) = (\frac{N_A(0)}{\frac{\tau_A}{\tau_B} - 1}(e^{t(\frac{1}{\tau_B} - \frac{1}{\tau_A})} - 1) + N_B(0))e^{-t/\tau_B} \text{ if } \tau_A \neq \tau_B \tag{14}$$

$$N_B(t) = (\frac{N_A(0)}{\tau_A}t + N_B(0))e^{-t/\tau_B} \text{ if } \tau_A = \tau_B \tag{15}$$

I set the time step $dt = 0,01$, $\tau_B = 1$ and a series of $\tau_A = \tau_A/\tau_B = 0.1,\ 0.3,\ 0.5,\ 1,\ 2,\ 3,\ 4$. In order to probe behaviors of $N_A(t)$ and $N_B(t)$ with different initial conditions, I set two kinds of initial conditions: (1): $N_A(0) = 100,\ N_B(0) = 0$; (2): $N_A(0) = 50,\ N_B(0) = 50$. From the analytical solution, it's clear that if $N_A(0) = 0,\ N_B(0) = 100$, $N_B(t) = N_B(0)e^{-t/\tau_B}$ which is the same situation as N_A with $N_A(0) = 100$ and $N_A(t) = 0$. So, here, I don't need to calculate this situation. Figure 3 shows the comparison of numerical and analytical results for this problem with the first initial condition and Figure 4 shows the second initial condition.

Discussion For the first initial condition, which is $N_A(0) = 100,\ N_B(0) = 0$, N_A will decrease as time going on and the speed of decreasing is proportional to $1/\tau_A$ since N_A is proportional to e^{-t/τ_A}. N_B will increase first and then decrease. The amplitude of rising is also proportional to $1/\tau_A$ because the effect of N_A on N_B decreases as the ratio increasing. If there is no effect of N_A on N_B, N_B should be zero all the time because $N_B(0)$ is zero in this case. After a long time, N_A and N_B for different ratios always approach to zero.
For the second initial condition, which is $N_A(0) = 50,\ N_B(0) = 50$, The behavior of N_A is the same as the previous case because N_A is independent on N_B. The behavior of N_B is also the addition of the behavior of N_B without any effect of N_A and the reaction term. So, as τ_A/τ_B increasing, the effect decreases and the amplitude of the pulse decreases.

Problem 3 continued on next page...

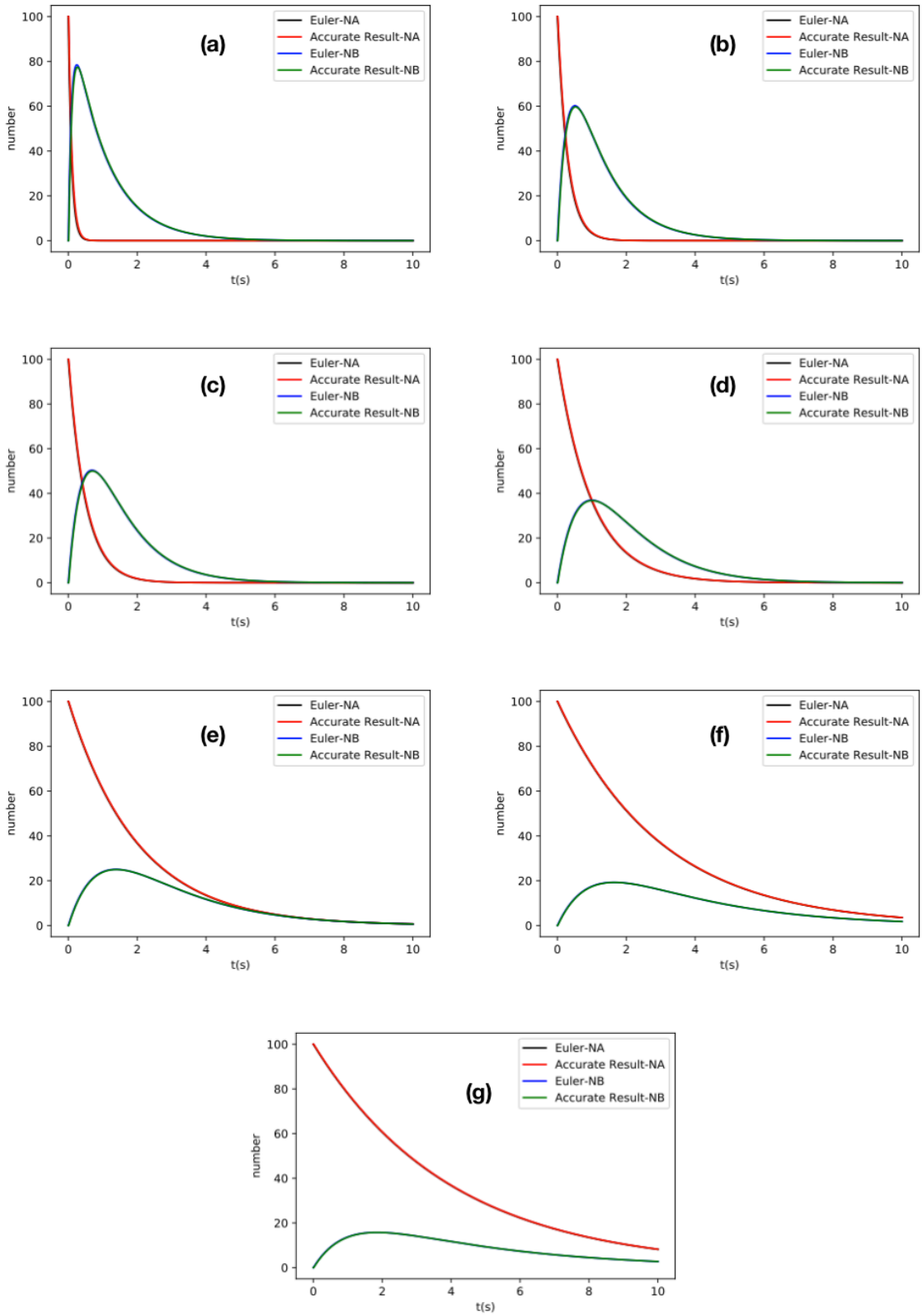

Figure 3: Numerical and analytical calculation results with $N_A(0) = 100$, $N_B(0) = 0$: (a): $\tau_A/\tau_B = 0.1$, (b): $\tau_A/\tau_B = 0.3$, (c): $\tau_A/\tau_B = 0.5$, (d): $\tau_A/\tau_B = 1$, (e): $\tau_A/\tau_B = 2$, (f): $\tau_A/\tau_B = 3$, (g): $\tau_A/\tau_B = 4$

Problem 3 continued on next page...

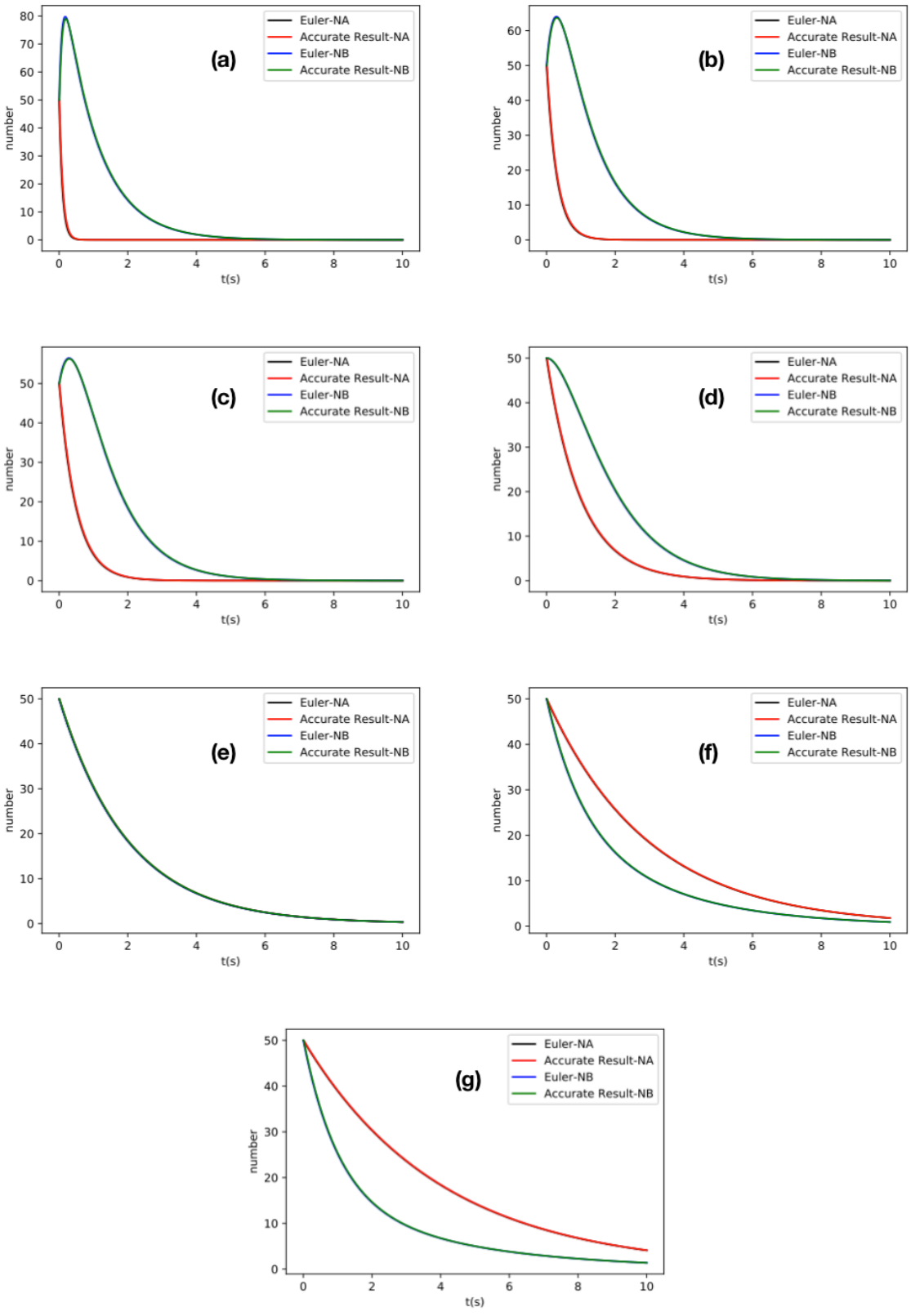

Figure 4: Numerical and analytical calculation results with $N_A(0) = 50$, $N_B(0) = 50$: (a): $\tau_A/\tau_B = 0.1$, (b): $\tau_A/\tau_B = 0.3$, (c): $\tau_A/\tau_B = 0.5$, (d): $\tau_A/\tau_B = 1$, (e): $\tau_A/\tau_B = 2$, (f): $\tau_A/\tau_B = 3$, (g): $\tau_A/\tau_B = 4$

Problem 3 continued on next page...

Listing 3: Original Script of Problem 3

```
# -*- coding: utf-8 -*-
import math
import numpy as np
import matplotlib.pyplot as plt

class Radioactive_decay (object):

    tau_A = 4.0
    tau_B = 1.0

    def __init__(self, Na0 = 0.0, Nb0 = 100.0, dt = 0.1, tf = 10.0):
        self.Na = Na0
        self.Na1 = Na0
        self.Na0 = Na0

        self.Nb = Nb0
        self.Nb1 = Nb0
        self.Nb0 = Nb0

        self.t = 0.0
        self.tf = tf
        self.dt = dt
        self.tarray = np.arange(0.0, tf+0.5*self.dt, self.dt)

    def Euler_step(self):
        self.Nb += (self.Na / self.tau_A - self.Nb / self.tau_B) * self.dt
        self.Na += ((-1) * self.Na / self.tau_A)* self.dt
        self.Na1 = self.Na0 * math.exp(-self.t/self.tau_A)

        #if tau_A = tau_B
        #self.Nb1 = (self.Na0 * self.t / self.tau_A + self.Nb0)
        #* math.exp(-self.t/self.tau_B)

        #if tau_A != tau_B
        self.Nb1 = (self.Nb0 + self.Na0 * (math.exp(self.t * (1/self.tau_B-1/self.tau_A))-1)
            /(self.tau_A/self.tau_B - 1)) * math.exp(-self.t/self.tau_B)
        self.t += self.dt


P = Radioactive_decay(dt=0.01)

t_list = []
Na_list = []
Na1_list = []
Nb_list = []
Nb1_list = []

while ( P.t < P.tf ):
    Na_list.append(P.Na)
    Na1_list.append(P.Na1)
    Nb_list.append(P.Nb)
    Nb1_list.append(P.Nb1)
```

```
    t_list.append(P.t)
    P.Euler_step()

fig = plt.figure()
plt.plot(t_list,Na_list,'k', label = "Euler-NA")
plt.plot(t_list,Na1_list,'r',label= "Accurate Result-NA")
plt.plot(t_list,Nb_list,'blue', label = "Euler-NB")
plt.plot(t_list,Nb1_list,'green',label= "Accurate Result-NB")
plt.ylabel('number')
plt.xlabel('t(s)')
plt.legend()
plt.show()
fig.savefig('problem3_step=0.01_ratio=4_ICA=0.pdf')
```

Problem 4

Problem: The dynamics of a population can often be described using first-order rate equations. For example, the total number of individuals in a population, N, may vary with time according to the equation

$$\frac{dN}{dt} = aN - bN^2, \tag{16}$$

where aN corresponds to the birth of new members and bN^2 corresponds to deaths. The death term is proportional to N^2 to account for the fact that food becomes more scarce when the population becomes large. First solve equation 16 with $b = 0$ using the Euler method and compare your numerical result with the exact solution. Then solve equation 16 with nonzero values of b. Interesting values of a and b depend on the initial population. For small $N(0)$, $a = 10$ and $b = 3$ is a good choice, while for $N(0) \sim 1000$, $a = 10$ and $b = 0.01$ is a good choice. Explore the behavior of $N(t)$ for different $N(0)$, a, and b values. Give an intuitive explanation of your results.

Solution: This problem is to solve the first-order differential equation 16 by Euler method. The original script is shown as Listing 4. The analytical solution for this equation is

$$N(t) = \frac{ae^{at+C}}{1 + be^{at+C}} \tag{17}$$

where C is a constant. When $t = 0$, $N(0) = \frac{ae^C}{1+be^C}$. So it's obvious that if $a - bN(0) \neq 0$, the solution is

$$N(t) = \frac{aN(0)e^{at}}{a + bN(0)(e^{at} - 1)} \tag{18}$$

If $a - bN(0) = 0$, $N(0)$ has to be zero. Listing 4 is my code for this problem. The initial condition and parameters is discussed as following:

- $b = 0$, $N(t) = N(0)e^{at}$. Figure 5 shows the comparison of numerical and analytical results of equation 16 with $b = 0$, $a = 1$, $N(0) = 50$, $dt = 0.001$.
- $b \neq 0$. Figure 6 shows the comparison of numerical and analytical results of equation 16 with different initial conditions and parameters: for small $N(0)$: $a = 10$, $b = 0.1,\ 0.3$, $N(0) = 50$ and for large $N(0)$: $a = 10$, $b = 0.005,\ 0.02$, $N(0) = 1000$.

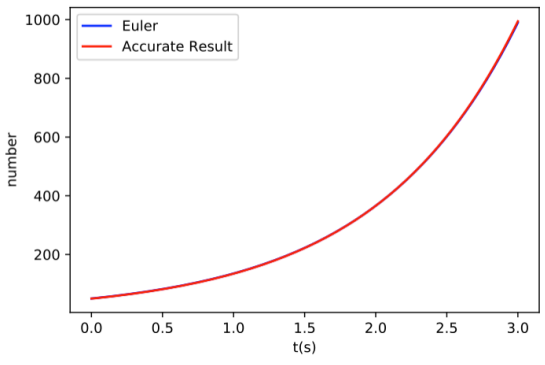

Figure 5: Numerical and analytical calculation results with $b = 0$

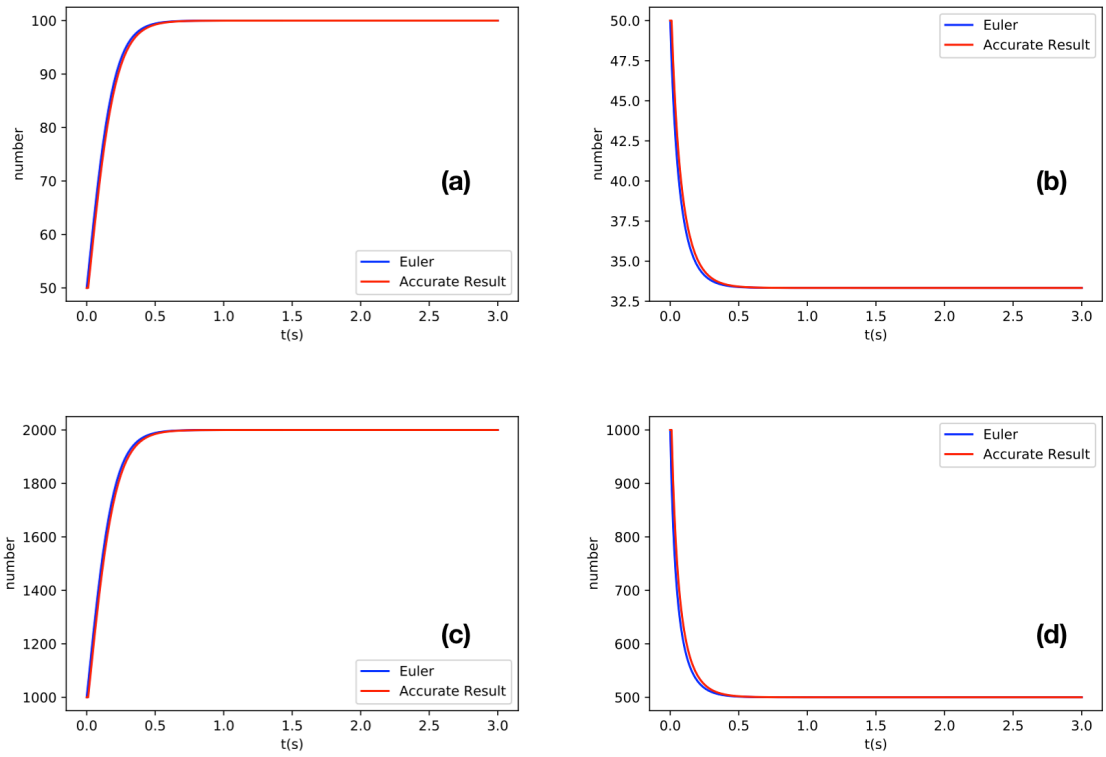

Figure 6: Numerical and analytical calculation results with $a = 10$: (a): $N(0) = 50,\ b = 0.1$, (b): $N(0) = 50,\ b = 0.3$, (c): $N(0) = 1000,\ b = 0.005$, (d): $N(0) = 1000,\ b = 0.02$

Discussion When I derive the equation 18, I get that $a - bN(0)$ can not be zero. So, we can discuss two different cases: $a - bN(0) > 0$ and $a - bN(0) < 0$. We can get $a/b > N(0)$ in the first case and $a/b < N(0)$. What is a/b? When time goes to infinity, we can see that $N(+\infty) = a/b$. In other words, when $a - bN(0) > 0$, $N(+\infty) = a/b > N(0)$, which means that the population will finally increase to a/b. Otherwise, when $a - bN(0) < 0$, $N(+\infty) = a/b < N(0)$, which means that the population will finally decrease to a/b.
For small $N(0)$, $N(0) = 50$, $a = 10$, the critical point for b is 0.2. So I choose $b = 0.1$ and $b = 0.3$. When $b = 0.1$, the population should finally increase to $a/b = 100$. When $b = 0.3$, the population should finally decrease to $a/b = 33.33$.
For large $N(0)$, $N(0) = 1000$, $a = 10$, the critical point for b is 0.01. So I choose $b = 0.005$ and $b = 0.02$. When $b = 0.005$, the population should finally increase to $a/b = 2000$. When $b = 0.02$, the population should finally decrease to $a/b = 500$.
From the Figure 6, we can clearly see that my calculation results are exactly consistent with theoretical predictions.

Listing 4: Original Script of Problem 4

```
# -*- coding: utf-8 -*-

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

class Population (object):

    a = 10.0
    b = 0.3

    def __init__(self, N0 = 50.0, dt = 0.001, tf = 3.0):

        self.N = N0
        self.N0 = N0
        self.N1 = N0

        self.t = 0.0
        self.tf = tf
        self.dt = dt
        self.tarray = np.arange(0.0, tf+0.5*self.dt, self.dt)

    def Euler_step(self):

        self.N += (self.a * self.N - self.b * self.N * self.N) * self.dt
        self.N1 = self.a * self.N0 * math.exp(self.a * self.t) /
        (self.a + self.b * self.N0 * (math.exp(self.a * self.t) -1))
        self.t += self.dt


P = Population(dt=0.01)

t_list = []
N_list = []
N1_list = []
```

```
while ( P.t < P.tf ):
    N_list.append(P.N)
    N1_list.append(P.N1)
    t_list.append(P.t)
    P.Euler_step()


fig = plt.figure()
plt.plot(t_list,N_list,'blue', label = "Euler")
plt.plot(t_list,N1_list,'r',label= "Accurate Result")
plt.ylabel('number')
plt.xlabel('t(s)')
plt.legend()
plt.show()
fig.savefig('problem4_step=0.001_a=10_b=0.3_N0=50.pdf')
```