ME EN 2450 Assignment HW 5

Name: Christopher Wall

I declare that the assignment here submitted is original except for source material explicitly acknowledged.

I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the University website.

Chris Wall	10/30/2024
Name	Date
CHRISTOPHER WALL	u1467634
Signature	Student ID

Score

Total: /25

Exercise 1 (10 pts)

Solve the following system of first-order ODEs:

$$\frac{dy}{dt} = -2y + 4e^{-t}$$
$$\frac{dz}{dt} = -\frac{yz^2}{3}$$

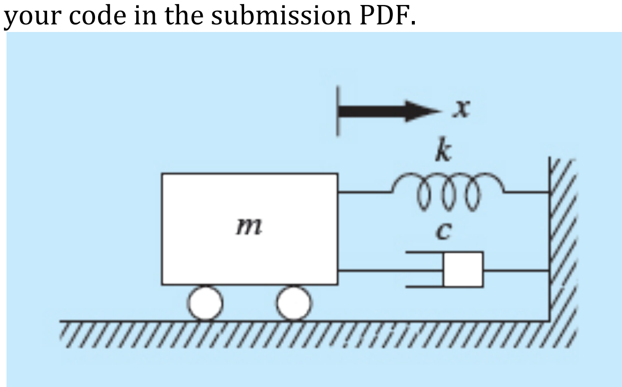
over a time from 0 to 1, $h = 0.1$, with $y(0) = 2$ and $z(0) = 4$. Use (a) Euler's and (b) fourth-order Runge Kutta. Submit a plot (in the pdf) of $y(t)$ and a separate plot for $z(t)$. On each plot, include the results from both methods. Include ALL your code in the submission PDF.

Exercise 2 (15 pts)

The motion of a damped spring-mass system is described by the following ODE:

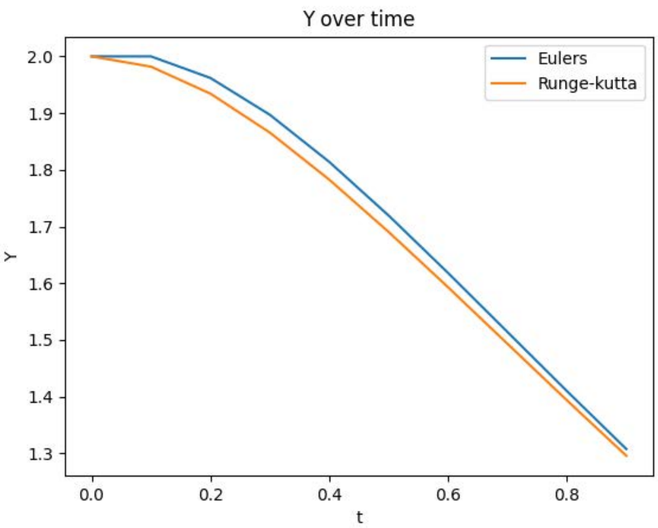
$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0$$

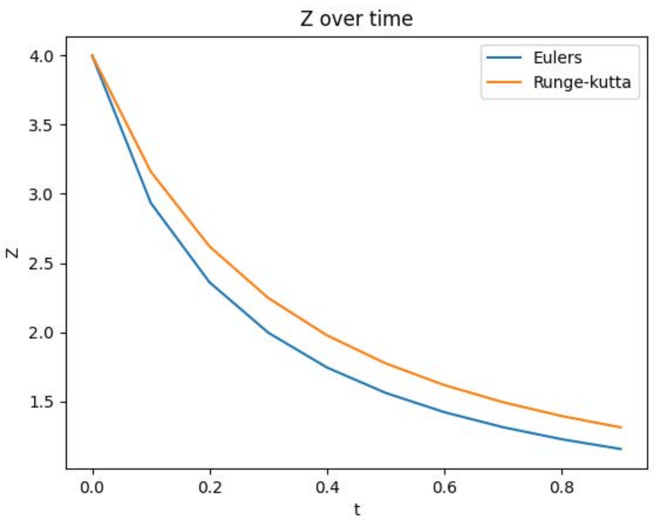
where x = displacement from equilibrium position [m], t = time [s], $m = 20$ [kg] mass, and c is the damping coefficient [N s/m]. The damping coefficient, c, takes on three values: 5 (underdamped), 40 (critically damped), and 200 (overdamped). The spring constant $k = 20$ N/m. The initial velocity is zero, and the initial displacement $x = 1$ [m]. Solve this IVP using a numerical method over the time period $0 \leq t \leq 15$ [s] **(Use the step size h = 0.1 [s])** Plot the displacement versus time for each of the three values of the damping coefficient on the same plot. Submit your plot in the pdf along with a statement about the characteristic differences in system behavior among the three damping coefficients. Include ALL your code in the submission PDF.



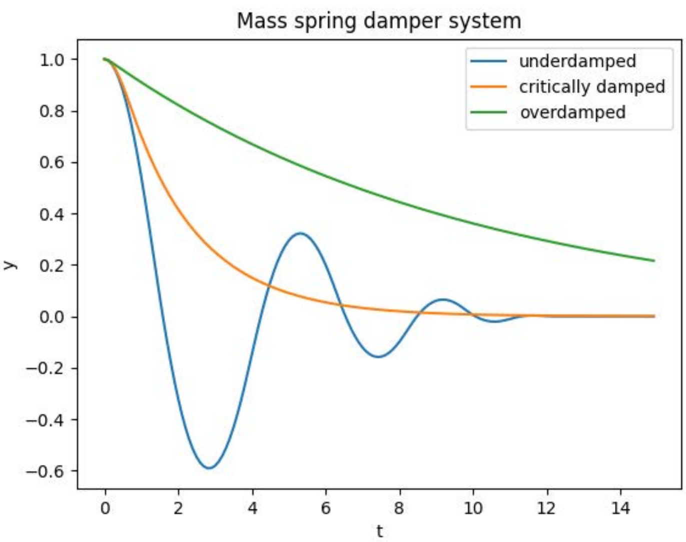
Hint: Before you start any coding, you need to first convert the ODE into a system of 1st-order ODEs and then put all equations into the *Standard Form.*

```
import numpy as np
import matplotlib.pyplot as plt
from eulers import solveSystemEulers
from RK4 import RK4

#derivative functions -----------------
def dydt(t, y, z):
    return -2*y + 4*np.e**(-t)

def dzdt(t, y, z):
    return -y*z**2/3
#-------------------------------------

#Finding solutions using eulers
y, z, tRange = solveSystemEulers( [dydt, dzdt] , 2, 4, 0, 1, .1)

#Finding solution using runge kutta
f = [dydt, dzdt]
y0 = [2, 4]
tRange, yR = RK4(0, y0, f, .1, 1 )

# Plotting the results
plt.figure()
plt.plot( tRange, y, label='Eulers')
plt.plot( tRange, yR[:,0], label='Runge-kutta')
plt.title('Y over time')
plt.ylabel('Y')
plt.xlabel('t')
plt.legend()
plt.figure()
plt.plot( tRange, z, label='Eulers')
plt.plot( tRange, yR[:,1], label='Runge-kutta')
plt.title('Z over time')
plt.ylabel('Z')
plt.xlabel('t')
plt.legend()
plt.show()
```





```
from RK4 import RK4
import numpy as np
import matplotlib.pyplot as plt

#Forcing funciton, for this case there are no forces over time
def F(t, y1, y2):
    return 0

#dz/dx
def f1(t, y1, y2):
    return y2

#standard form dy/dt for different damping coefficients--------------------
def f2(t, y1, y2):
    c = 5
    m = 20
    k = 20
    sign = np.sign((-c*y2 - k*y1 + F(t, y1, y2))/m)
    return np.sqrt(np.abs((-c*y2 - k*y1 + F(t, y1, y2))/m))*sign

def f3(t, y1, y2):
    c = 40
    m = 20
    k = 20
    sign = np.sign((-c*y2 - k*y1 + F(t, y1, y2))/m)
    return np.sqrt(np.abs((-c*y2 - k*y1 + F(t, y1, y2))/m))*sign

def f4(t, y1, y2):
    c = 200
    m = 20
    k = 20
    sign = np.sign((-c*y2 - k*y1 + F(t, y1, y2))/m)
    return np.sqrt(np.abs((-c*y2 - k*y1 + F(t, y1, y2))/m))*sign

#---------------------------------------------------------------------------

fu = [f1, f2]   #underdamped functions
fc = [f1, f3]   #critically damped funcs
fo = [f1, f4]   #overdamped funcs
t0 = 0          #intial time
y0 = [1, 0]     #intial conditions
#Solutions
tRange, y1 = RK4(0, y0, fu, 0.1, 15)
tRange, y2 = RK4(0, y0, fc, 0.1, 15)
tRange, y3 = RK4(0, y0, fo, 0.1, 15)

#Plotting
plt.figure()
plt.plot(tRange, y1[:, 0], label='underdamped')
plt.plot(tRange, y2[:, 0], label='critically damped')
plt.plot(tRange, y3[:, 0], label='overdamped')
plt.title('Mass spring damper system')
plt.xlabel('t')
plt.ylabel('y')
plt.legend()
plt.show()

#As the damping coefficient increases the motion of the spring goes from
# a wave like motion to an exponential decay motion. The critically damped scenario
# is the point at which the function is between the two cases of a wave and exponential.
#As c increases, it takes longer and longer for the block to reach its equilibrium state.
```



```
import numpy as np

#solves a system of equations using eulers methods
def solveSystemEulers( functions, y0, z0, t0, tf, h ):
    tRange = np.arange(t0, tf, h )
    y = np.zeros_like(tRange)
    z = np.zeros_like(tRange)

    y[0] = y0
    z[0] = z0

    # yi = yi-1 + f(...)*h
    for i in range( 1, len(tRange) ):
        y[i] = y[i-1] + functions[0]( tRange[i-1], y[i-1], z[i-1]) * h
        z[i] = z[i-1] + functions[1]( tRange[i-1], y[i-1], z[i-1]) * h

    return y, z, tRange
```

```
import numpy as np

def RK4(t0, y0, f, h, tf):
    tRange = np.arange(t0, tf, h)
    y = np.zeros((len(tRange), len(y0)))  # Store results for all time steps
    y[0] = y0
    k = np.zeros((len(y0), 4))  # Store the four k values for each equation

    for i, t in enumerate(tRange[:-1]):  # Iterate through the time steps
        #Find k constants for each dependent variable
        k[:, 0] = h * np.array([f_i(t, *y[i]) for f_i in f])
        k[:, 1] = h * np.array([f_i(t + h / 2, *(y[i] + k[:, 0] / 2)) for f_i in f])
        k[:, 2] = h * np.array([f_i(t + h / 2, *(y[i] + k[:, 1] / 2)) for f_i in f])
        k[:, 3] = h * np.array([f_i(t + h, *(y[i] + k[:, 2])) for f_i in f])
        y[i + 1] = y[i] + (k[:, 0] + 2 * k[:, 1] + 2 * k[:, 2] + k[:, 3]) / 6  # Update y

    return tRange, y
```