Project 3

ME-328-102: ME Analysis

Abstract

Using Explicit-Euler’s method, midpoint method, and the 4th order Runge-Kutta method to model a cylinder oscillating in water.

# Justin Dyer

University of South Alabama student, Author

# Landon Freeman

University of South Alabama student, Program Discussion

Index

3.1 Problem Description……………………………………………………………..……pg2

3.2 Flowchart………………………………………………………………………………..….pg3

3.3 Python Source Codes………………………………………………………...……pg4-9

3.4 Results and Discussion…………………………………………………….….pg10-25

3.5 Conclusions……………………………………………………………………………..pg26

# 3-1 Problem Description:

A cylinder floating in water can be modeled by the following second order ODE:

(Eqn 3.1)

where y is the distance from the water level to the neutral line. The mass, damping constant, and buoyancy constant are given as: m=1.0kg, c=0.1kg/s, and k=0.01kg2/s2. The cylinder was dropped from y=0.2m. Therefore, the initial conditions are y(0)=0.2m and y’(0)=0m/s.

The exact solution for this problem can be expressed as:

For our project, we are going to use our knowledge of Explicit-Euler’s method, midpoint method, and the 4th order Runge-Kutta method to model this problem using different time steps. The time steps we will use are 0.1(sec), 1.0(sec), 2.5 (sec), 5.0(sec), 10.0(sec).

After plotting these methods, we will alter the value “c” to study the critical damping and overdamping. For critical damping, c = 0.2kg/s, for overdamping, c = 1.5kg/s.

# 3-2 Flowchart:

To begin this problem, we need to derive a flowchart to guide us while writing the program. We found that the following flowchart was best suited to guide us:

(Figure 3-1: Flowchart)

Using this flowchart, we have a layout of the steps needed to take to complete what is asked. First, input values for c and dt. Second, take the values and define all the variables. Third, plug the values into the functions until t\_final. Fourth, save the plots and show them. Finally, end the program.

Following this layout should guide us and let us successfully create the program needed.

# 3-3 Python Source Codes:

For this project, we used the following Python source codes:

Functions:

import math

def y\_exact\_solution (t) :

t2=0.0866025\*t

y\_exact\_solution = math.exp(-0.05\*t) \* (0.2\*math.cos(t2) + 0.11547\*math.sin(t2));

return y\_exact\_solution

def derives (y,z,t,m,c,k) :

dy\_over\_dt = z

dz\_over\_dt = -c/m\*z-k/m\*y

return dy\_over\_dt, dz\_over\_dt

def explicit\_euler(y,z,t,h,m,c,k) :

dy\_over\_dt,dz\_over\_dt = derives(y,z,t,m,c,k)

y\_iplus1 = y+dy\_over\_dt\*h

z\_iplus1 = z+dz\_over\_dt\*h

return y\_iplus1, z\_iplus1

def midpoint(y,z,t,h,m,c,k) :

dy\_over\_dt,dz\_over\_dt = derives(y,z,t,m,c,k)

y\_mid=y+dy\_over\_dt\*0.5\*h

z\_mid=z+dz\_over\_dt\*0.5\*h

dy\_over\_dt,dz\_over\_dt = derives(y\_mid,z\_mid,t,m,c,k)

y\_iplus1=y+dy\_over\_dt\*h

z\_iplus1=z+dz\_over\_dt\*h

return y\_iplus1, z\_iplus1

def RK4(y,z,t,h,m,c,k) :

y\_iplus1 = 0

z\_iplus1 = 0

dy\_over\_dt=0

dz\_over\_dt=0

dy\_over\_dt, dz\_over\_dt = derives(y,z,t,m,c,k)

k1y=dy\_over\_dt

k1z=dz\_over\_dt

dy\_over\_dt, dz\_over\_dt = derives(y+k1y\*h/2,z+k1z\*h/2,t+h/2,m,c,k)

k2y=dy\_over\_dt

k2z=dz\_over\_dt

dy\_over\_dt, dz\_over\_dt = derives(y+k2y\*h/2,z+k2z\*h/2,t+h/2,m,c,k)

k3y=dy\_over\_dt

k3z=dz\_over\_dt

dy\_over\_dt, dz\_over\_dt = derives(y+k3y\*h, z+k3z\*h,t+h,m,c,k)

k4y=dy\_over\_dt

k4z=dz\_over\_dt

y\_iplus1 = y+(k1y+2\*k2y+2\*k3y+k4y)\*h/6;

z\_iplus1 = z+(k1z+2\*k2z+2\*k3z+k4z)\*h/6;

return y\_iplus1, z\_iplus1

Solutions:

import math

import numpy as np

import matplotlib.pyplot as plt

import project\_3\_functions as f

print('Project 3. ODE of modeling a cylinder oscillating in still water.\n')

# input parameters from keyboard

c = float(input('Damping Coefficient c : '))# c=0.1

dt = float(input('Time Step Size (h) : '))

#dt = 0.1, 0.1(sec), 0.5 (sec) , 1(sec), 5(sec), 10(sec)

m = 1.0

k = 0.01

#Set t information

ti = 0.0 #Initial t

tf = 200.0 #Final t

# Initial Conditions @ t=ti

yi = 0.2

zi = 0.0

#Assign h = t\_(i+1) - t\_(i) = dt

h = dt

#Create vectors to store data for t(i) & y(i)

numberOfDataSets = math.floor((tf-ti)/dt) + 1

t = np.zeros(numberOfDataSets)

y\_exact = np.zeros(numberOfDataSets)

y\_ex = np.zeros(numberOfDataSets) #Explicit Euler results for y

z\_ex = np.zeros(numberOfDataSets) #Explicit Euler results for z = dy/dt

y\_mid = np.zeros(numberOfDataSets)

z\_mid = np.zeros(numberOfDataSets)

y\_rk4 = np.zeros(numberOfDataSets)

z\_rk4 = np.zeros(numberOfDataSets)

#assign the initial values for all y(0) and z(0)

i = 0

t[i]=ti

y\_exact[i]=yi

y\_ex[i]=yi

z\_ex[i]=zi

y\_mid[i]=yi

z\_mid[i]=zi

y\_rk4[i]=yi

z\_rk4[i]=zi

flag = 1 #declare flag to stop while loop to get y values from ti until tf with step h

while (flag):

#If t[i]+dt exceeds tf, h=tf-t

if t[i]+dt > tf:

h=tf-t

t[i+1]=t[i]+h;

#calculate the exact solution

y\_exact[i+1] = f.y\_exact\_solution(t[i+1])

#Calculate the Explicit Euler Solution

y\_ex[i+1], z\_ex[i+1] = f.explicit\_euler(y\_ex[i], z\_ex[i], t[i], h, m, c, k)

#Calculate the midpoint solution

y\_mid[i+1], z\_mid[i+1] = f.midpoint(y\_mid[i], z\_mid[i], t[i], h, m, c, k);

#Calculate the RK4 solution

y\_rk4[i+1], z\_rk4[i+1] = f.RK4(y\_rk4[i], z\_rk4[i], t[i], h, m, c, k);

i = i + 1

print('@ x= {:10.7} : y\_Exact Solution= {:10.7f} , y\_Explicit Euler= {:10.7f} , y\_MidPoint= {:10.7f}, y\_RK4= {:10.7f} \n' .format(t[i], y\_exact[i], y\_ex[i], y\_mid[i], y\_rk4[i]));

#If t eaches tf, stop the while loop

if i >= numberOfDataSets -1:

flag = 0

#end of while loop

#plot y against t

fig1 = plt.figure(1)

plt.plot(t, y\_exact, 'o-', t, y\_ex, '-x', t, y\_mid, '+g', t,y\_rk4, '-.\*k') #plot 4 curves pf y vs t

plt.legend(('y\_Exact Solution for c=0.1 only', 'y\_Explicit Euler Method', 'y\_Mid Point', 'y\_RK4'), loc = 0)

plottitle = 'Figure 1. y vs. t for ODE : c={:8.5f}kg/s& h={:8.5f}s' .format(c, h)

plt.title(plottitle)

plt.xlabel('t (sec)')

plt.ylabel('y (m)')

plotFileName = 'y-t c' + str(c) + 'h' + str(h) + '.jpg'

plt.savefig(plotFileName, format = 'jpg') #save plot in a jpg format file

plt.show() #show the plot on screen

# plot z against t

fig2 = plt.figure(2)

plt.plot(t, z\_ex, '-x', t, z\_mid, '+g', t,z\_rk4, '-.\*k') #plot 3 curves for z vs t

plt.legend(('z\_Explicit Euler Method', 'z\_Mid Point', 'z\_RK4'), loc = 0)

plottitle = 'Figure 2. z vs. t for ODE with : c={:8.5f}kg/s & h={:8.5f}s' .format(c, h)

plt.title(plottitle)

plt.xlabel('t (sec)')

plt.ylabel('z (m/s)')

plotFileName = 'z-t c' + str(c) + 'h' + str(h) + '.jpg'

plt.savefig(plotFileName, format = 'jpg') # save plot in a jpg format file

plt.show() #show the plot on screen

del c, dt, h, m, k

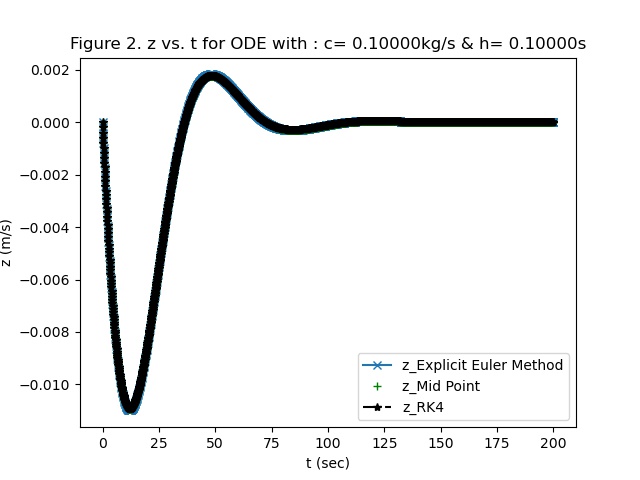
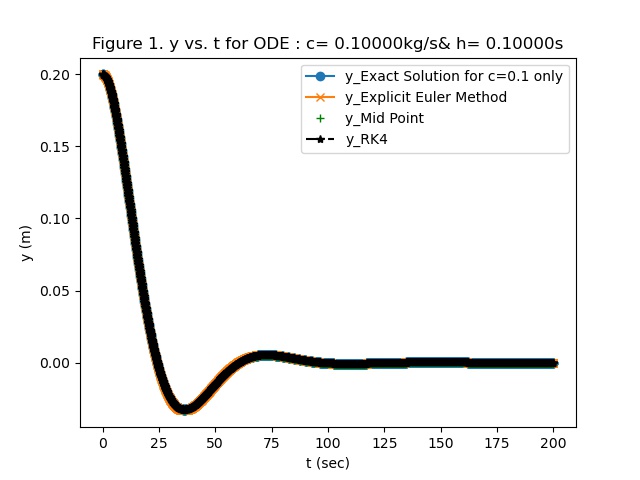
del t, ti, tf, numberOfDataSets

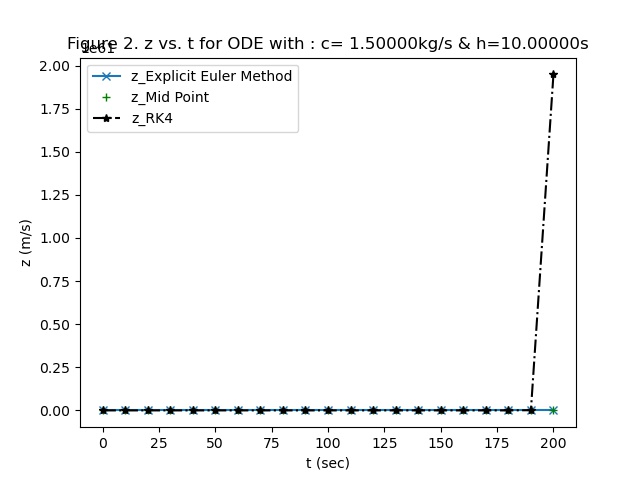
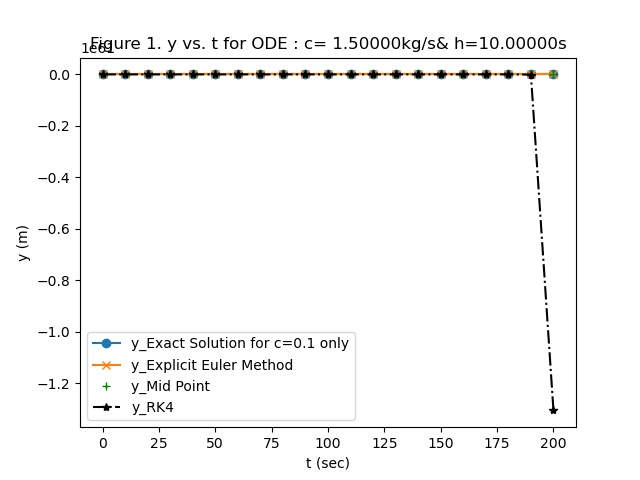
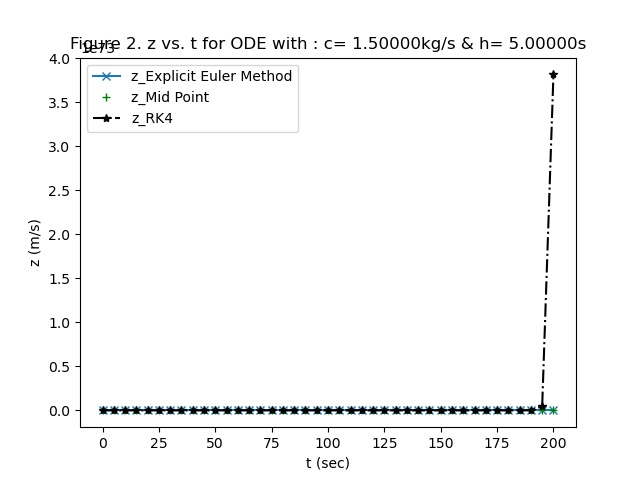
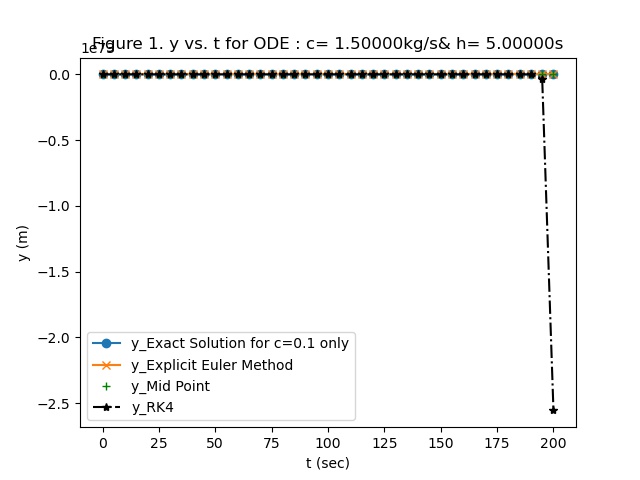
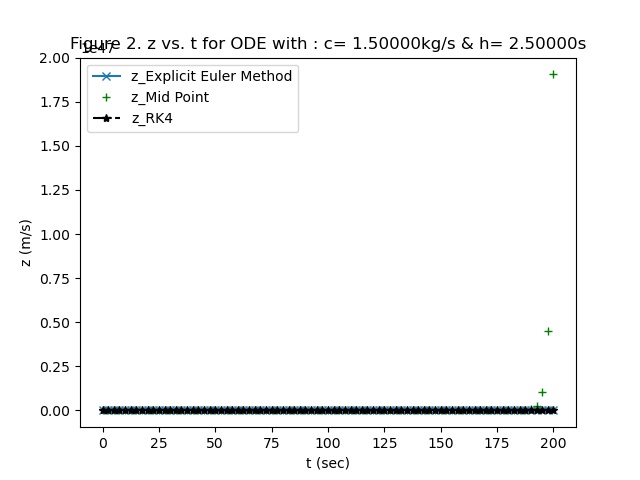
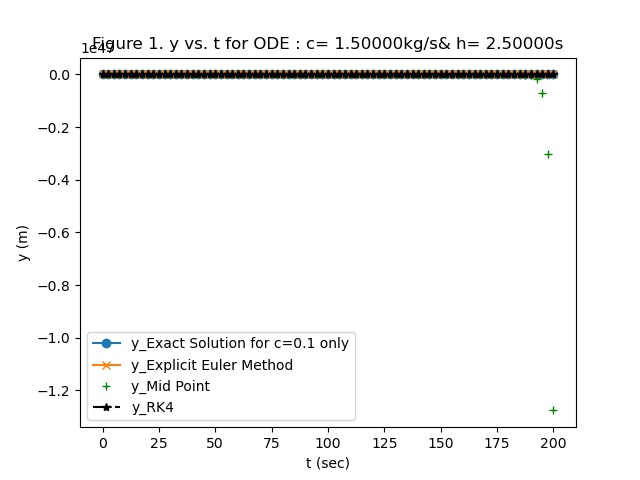
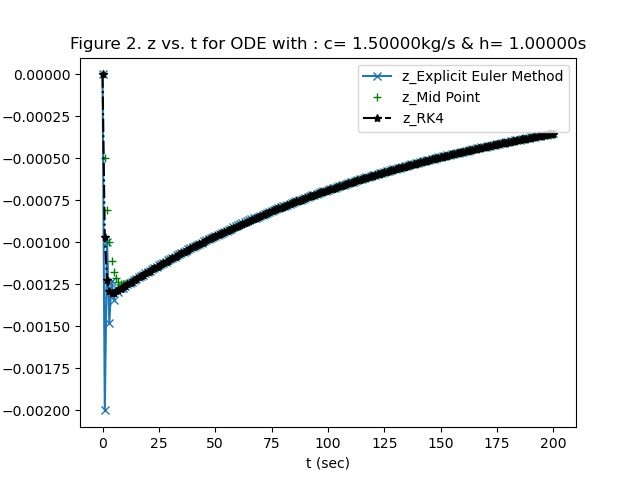
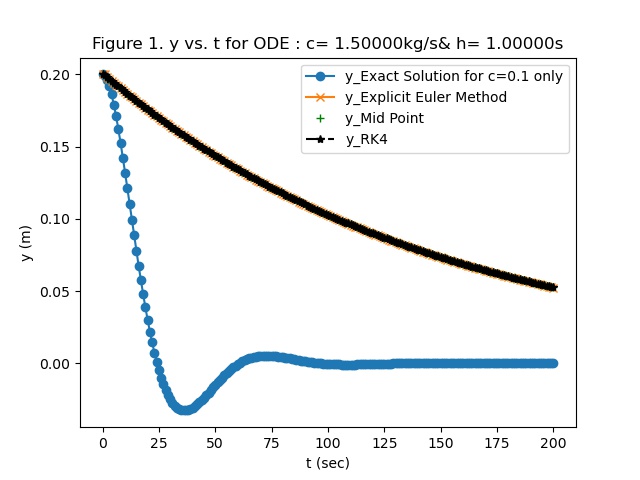
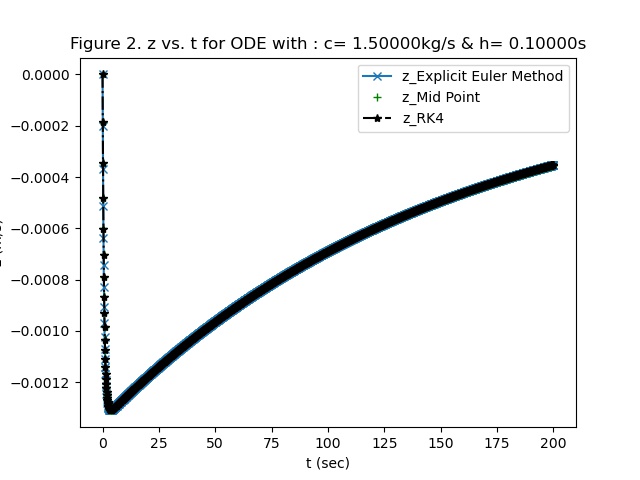
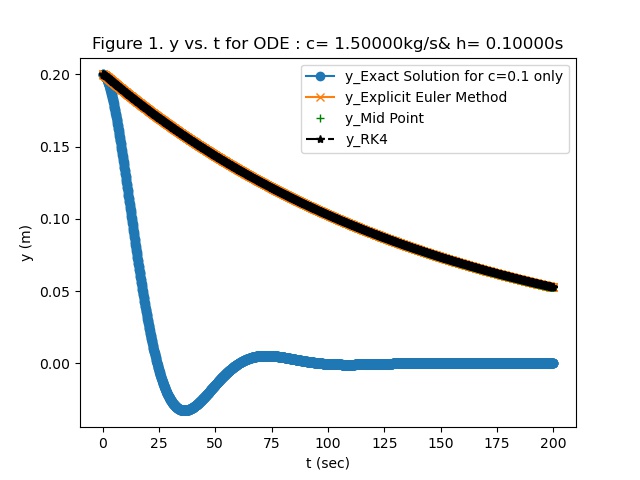
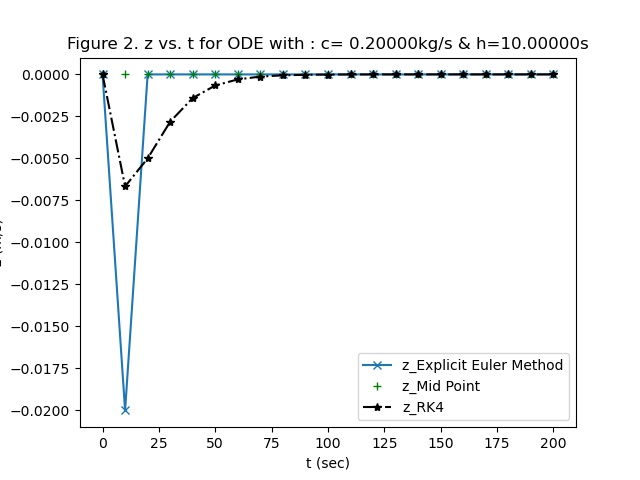
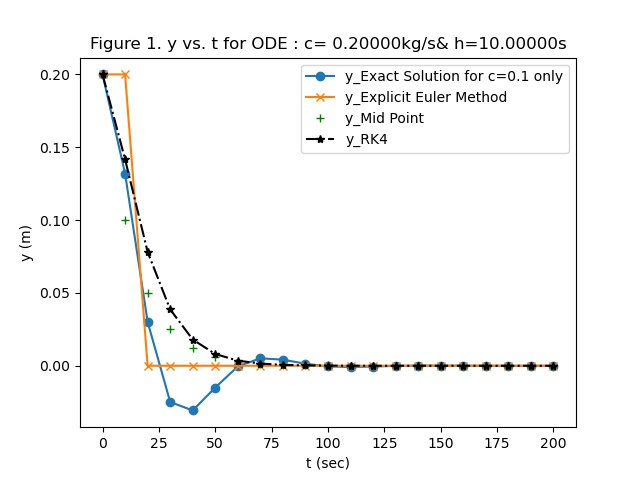
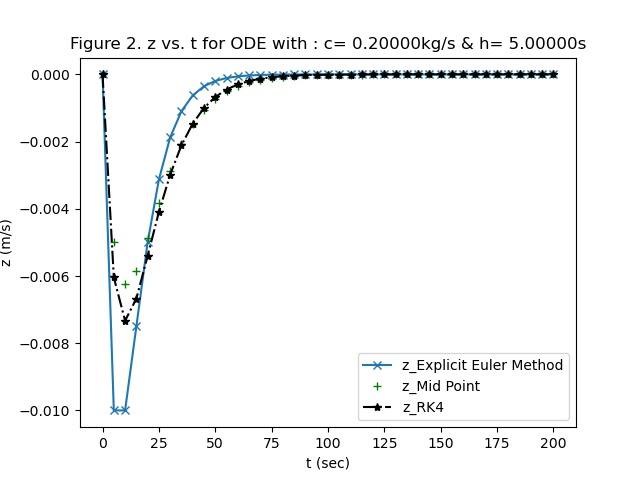
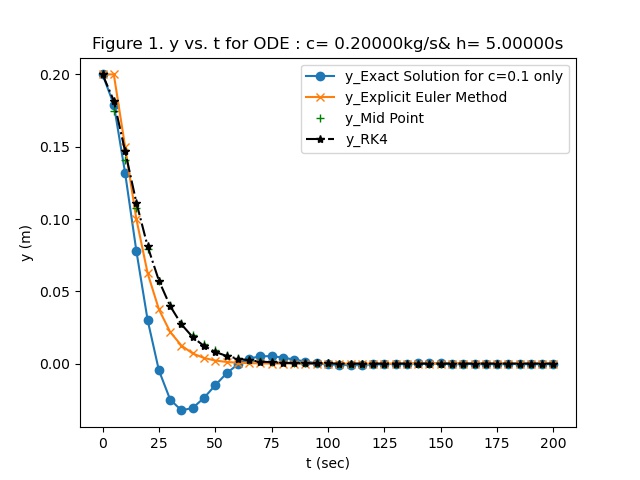
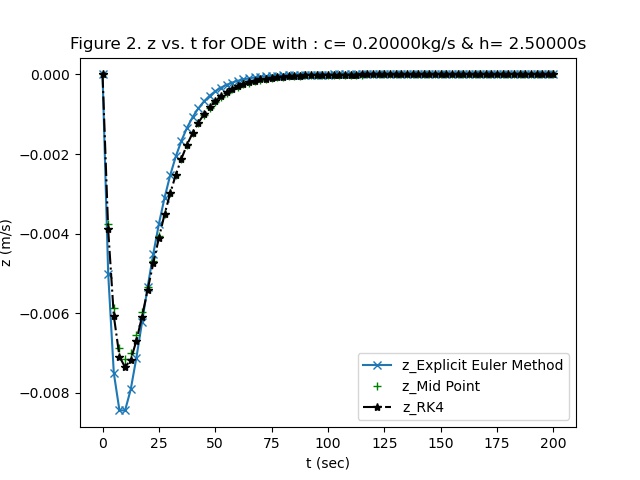
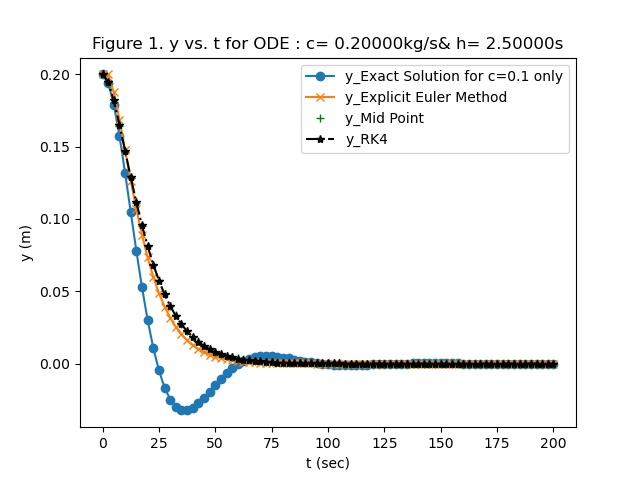
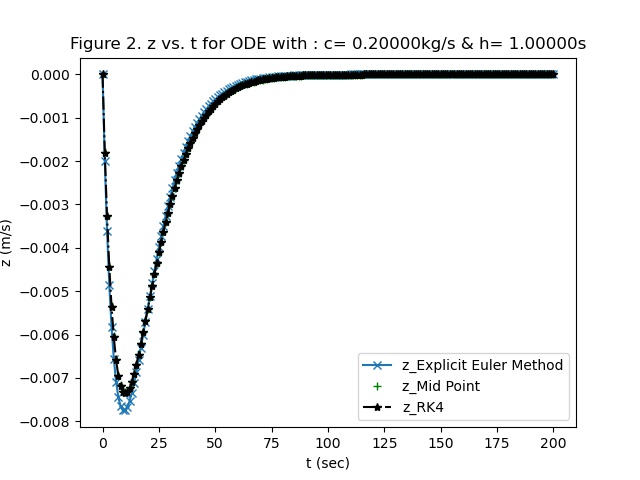
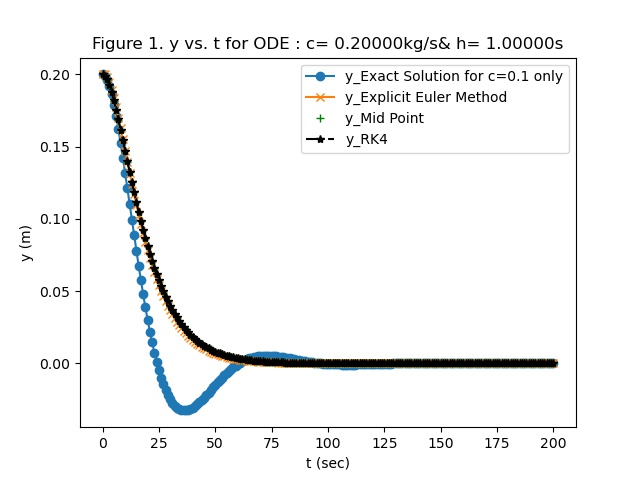
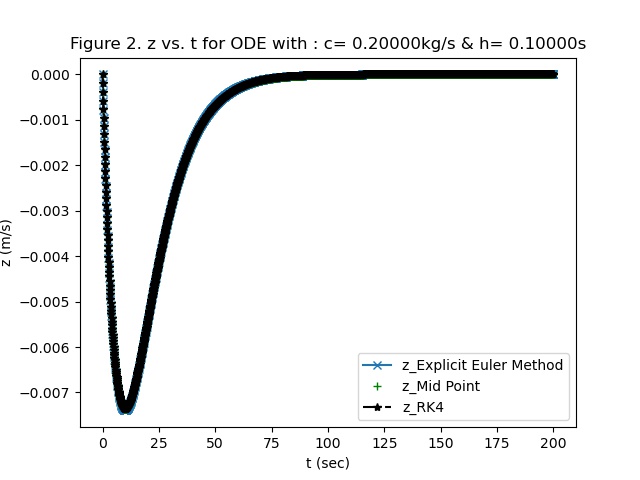
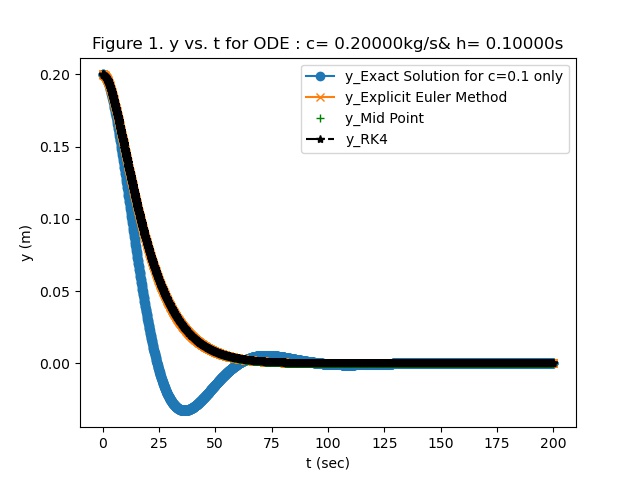
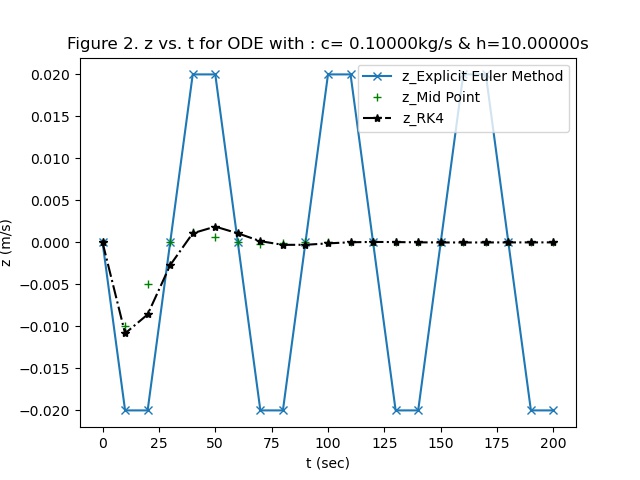
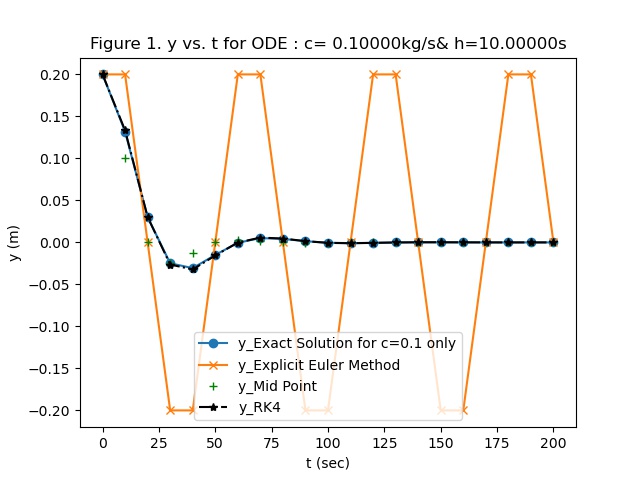
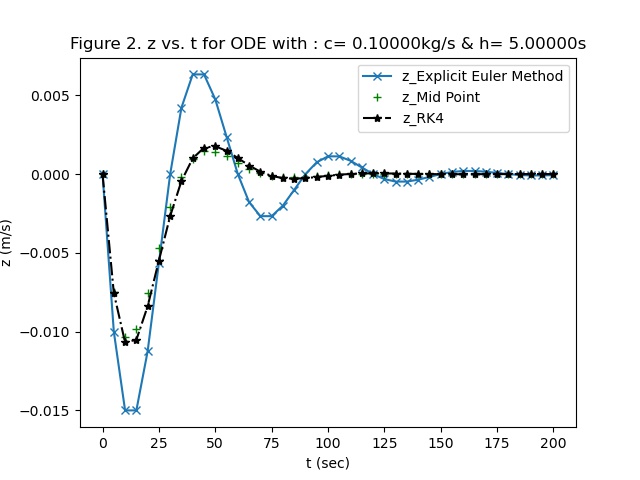
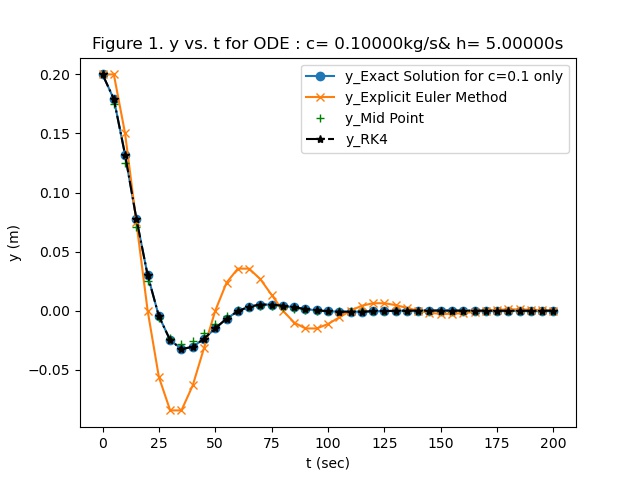
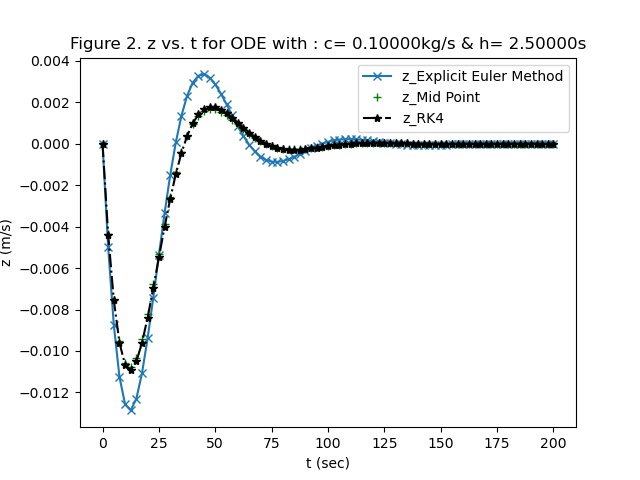
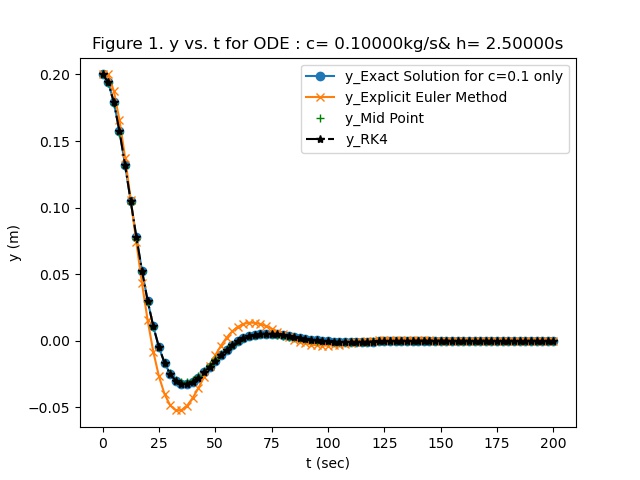
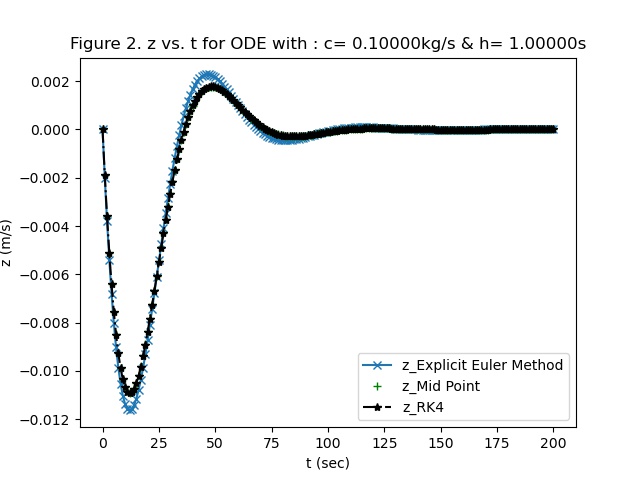
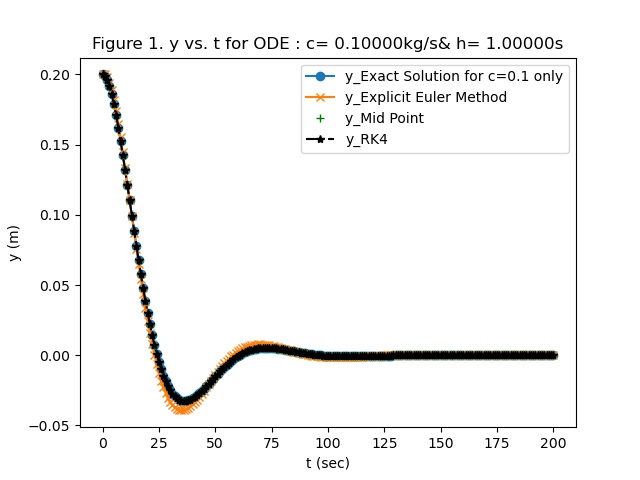
del y\_exact, y\_ex, z\_ex, y\_mid, z\_mid, y\_rk4, z\_rk4

del plottitle, plotFileName

3-4 Results and Discussion

The program outputs two plots. For our project, we ran the code multiple times adjusting the time steps and damping coefficients. The following 30 plots were received:





# 3-5 Conclusions:

We have learned to use Euler’s method, midpoint method, and Runge-Kutta method in previous classed. For each value given, with our knowledge of Python, we wrote a code plot the position of a cylinder relative to a body of water versus time.

With 15 inputs, our python program successfully took those values, and plotted the z vs dt and y vs dt. To improve this project, we could have more data input points and more accurate inputs; however, to our knowledge, the outputs we received are exact. We also could decrease the time step.