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:

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

$$m\frac{d^2y}{dt^2} + c\frac{dy}{dt} + ky = 0 \qquad \text{(Eqn 3.1)}$$

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=0.01kg^2/s^2. 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:

$$y(t) = \exp(-0.05t)\,[0.2\cos(\omega t) + 0.11547\sin(\omega t)] \;\; \omega = 0.0866025 rad/s$$

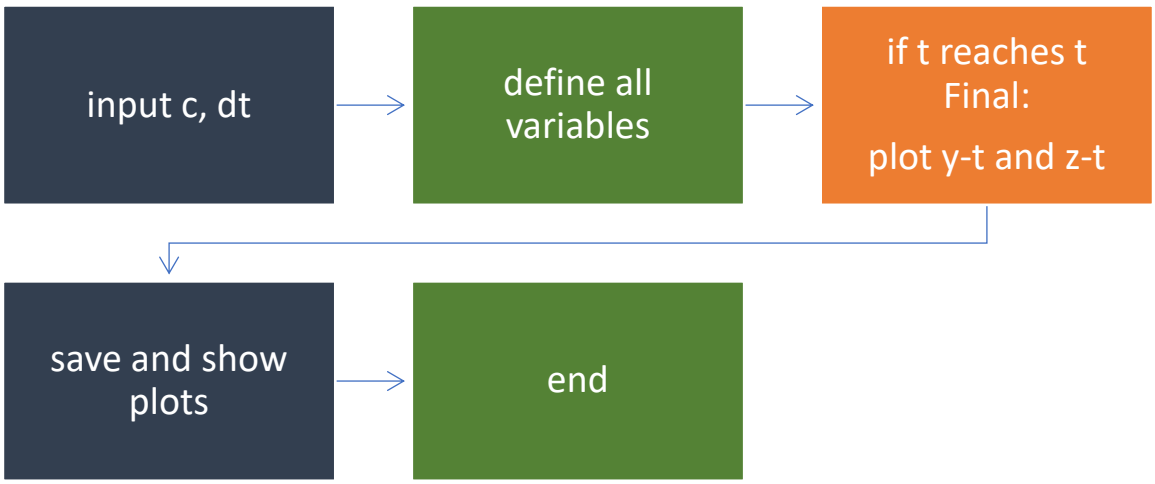
For our project, we are going to use our knowledge of Explicit-Euler's method, midpoint method, and the 4th order Runge-Kutta method to model this problem using different time steps. The time steps we will use are 0.1(sec), 1.0(sec), 2.5 (sec), 5.0(sec), 10.0(sec).

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)
```

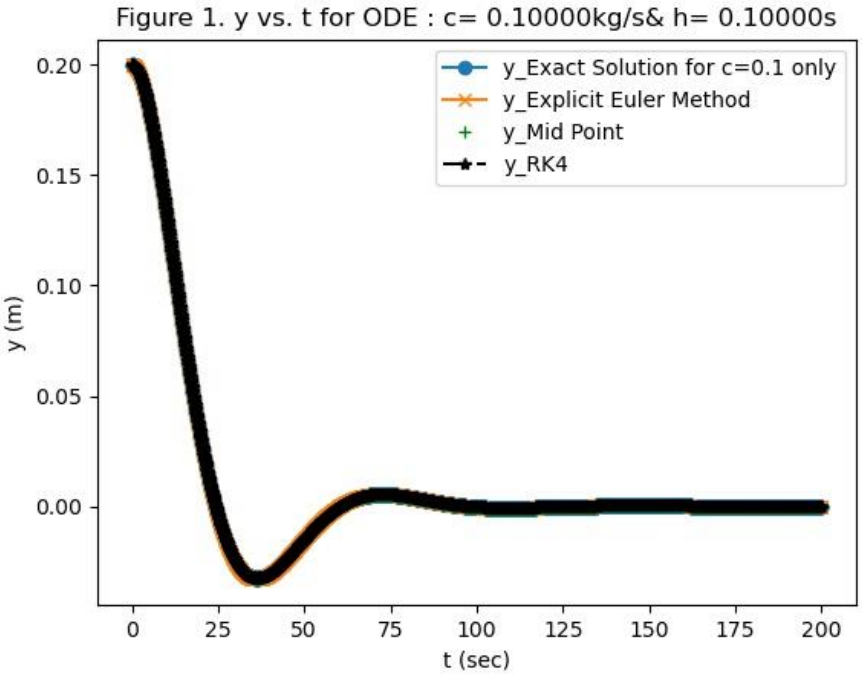
```
i = 0
t[i]=ti
y_exact[i]=yi
y_ex[i]=yi
z_ex[i]=zi
y_mid[i]=yi
z_mid[i]=zi
y_rk4[i]=yi
z_rk4[i]=zi

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]));
```

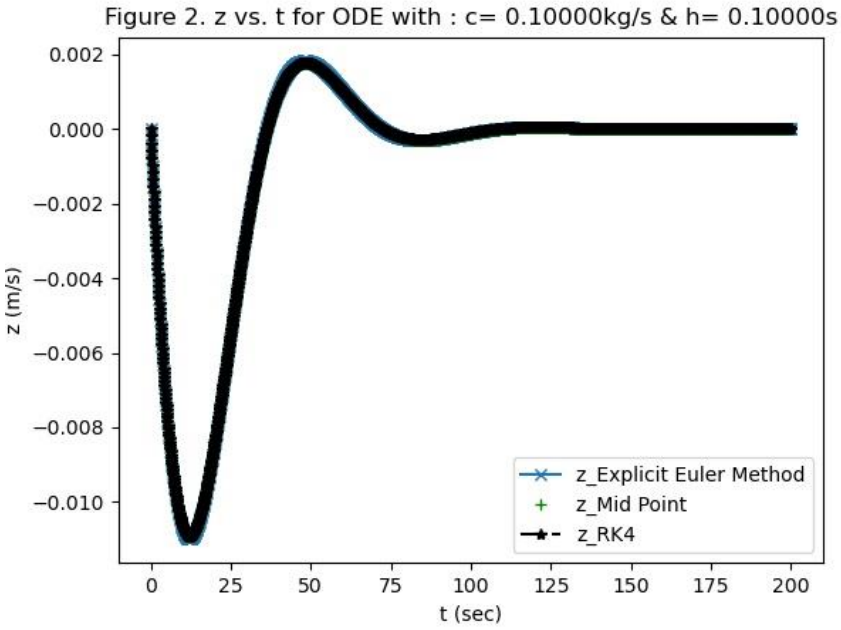
```
    #If t eaches tf, stop the while loop
    if i >= numberOfDataSets -1:
        flag = 0

    #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:





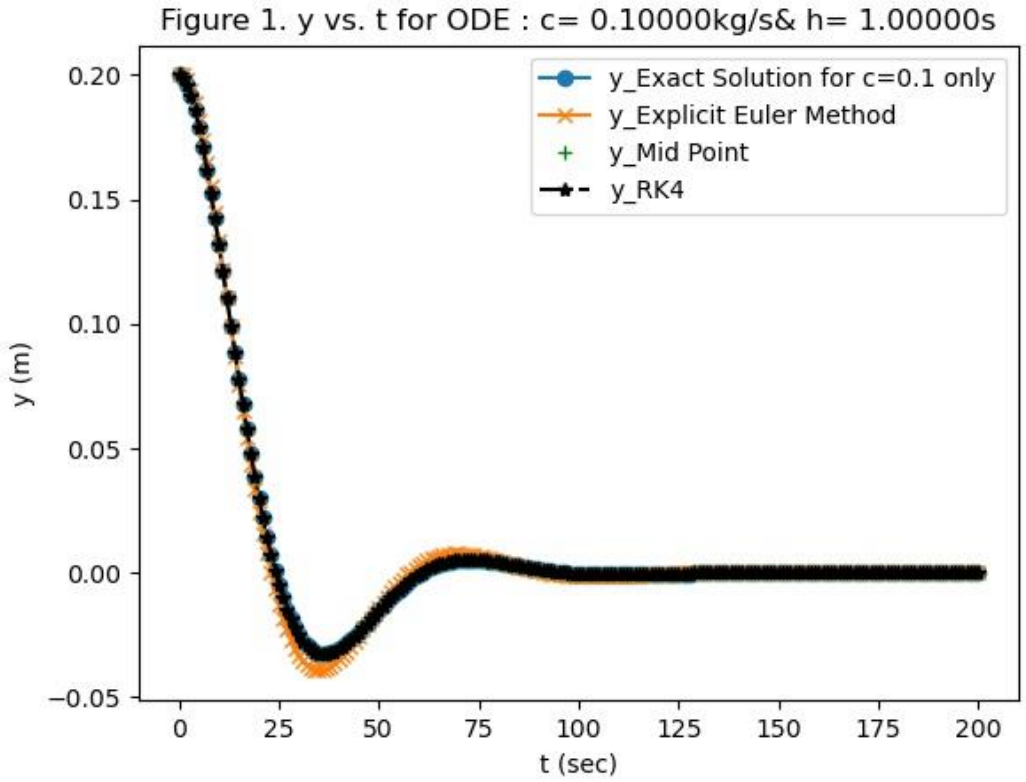


Figure 1. y vs. t for ODE : c= 0.10000kg/s& h= 1.00000s
y_Exact Solution for c=0.1 only
y_Explicit Euler Method
y_Mid Point
y_RK4
y (m)
t (sec)
0.20
0.15
0.10
0.05
0.00
−0.05
0
25
50
75
100
125
150
175
200

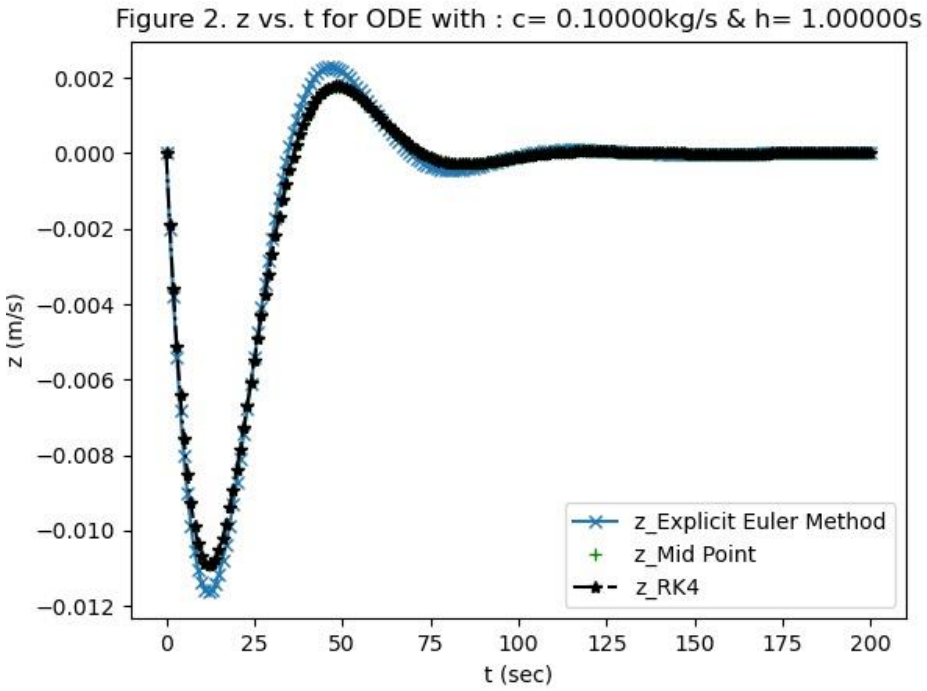


Figure 2. z vs. t for ODE with : c= 0.10000kg/s & h= 1.00000s
z_Explicit Euler Method
z_Mid Point
z_RK4
z (m/s)
t (sec)
0.002
0.000
−0.002
−0.004
−0.006
−0.008
−0.010
−0.012
0
25
50
75
100
125
150
175
200

Figure 1. y vs. t for ODE : c= 0.10000kg/s& h= 2.50000s

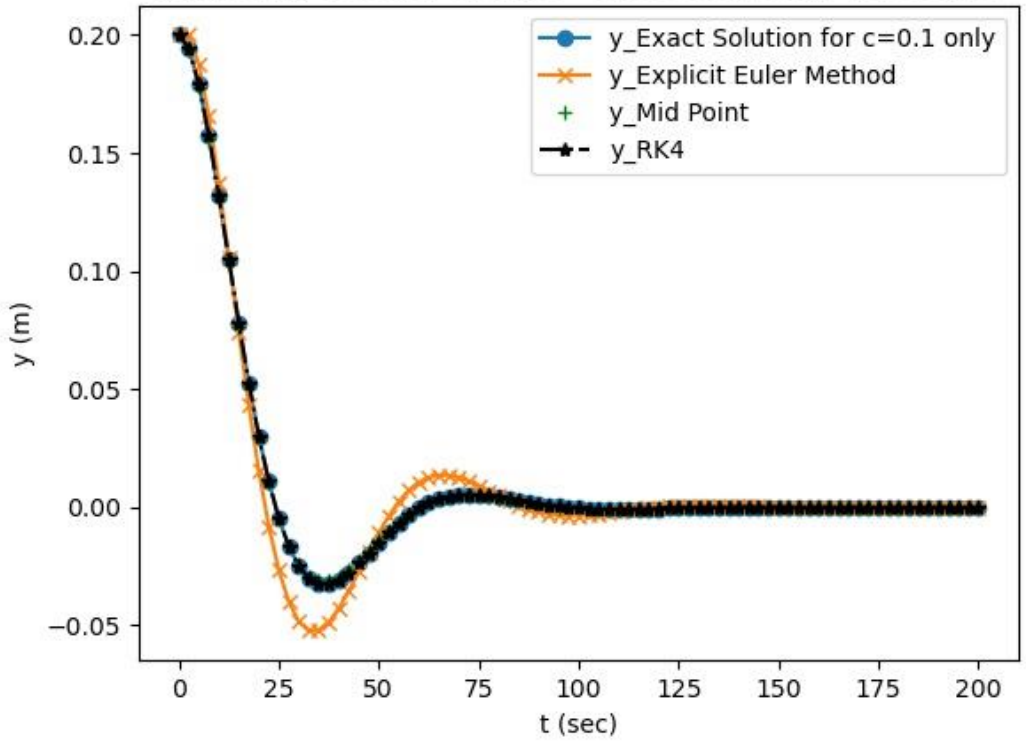


Figure 2. z vs. t for ODE with : c= 0.10000kg/s & h= 2.50000s

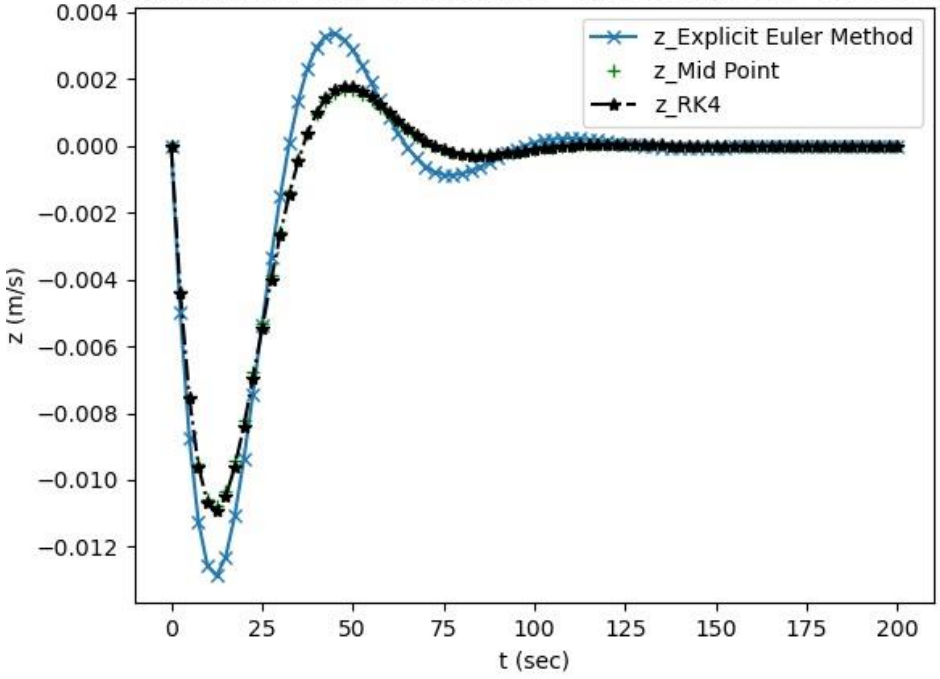


Figure 1. y vs. t for ODE : c= 0.10000kg/s& h= 5.00000s

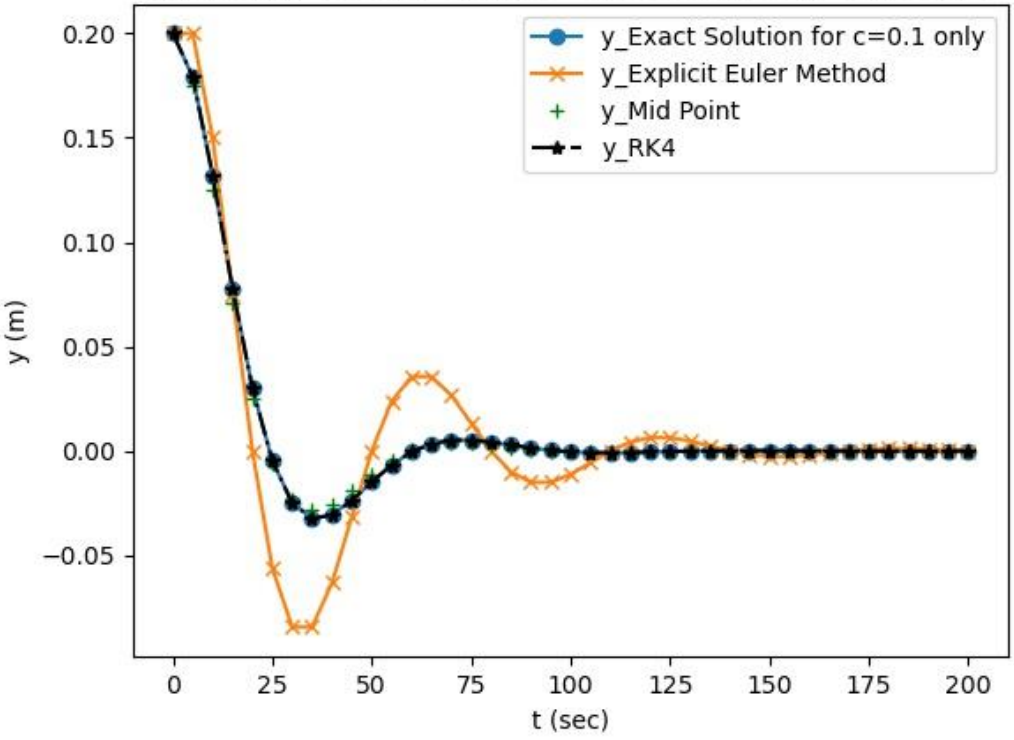


Figure 2. z vs. t for ODE with : c= 0.10000kg/s & h= 5.00000s

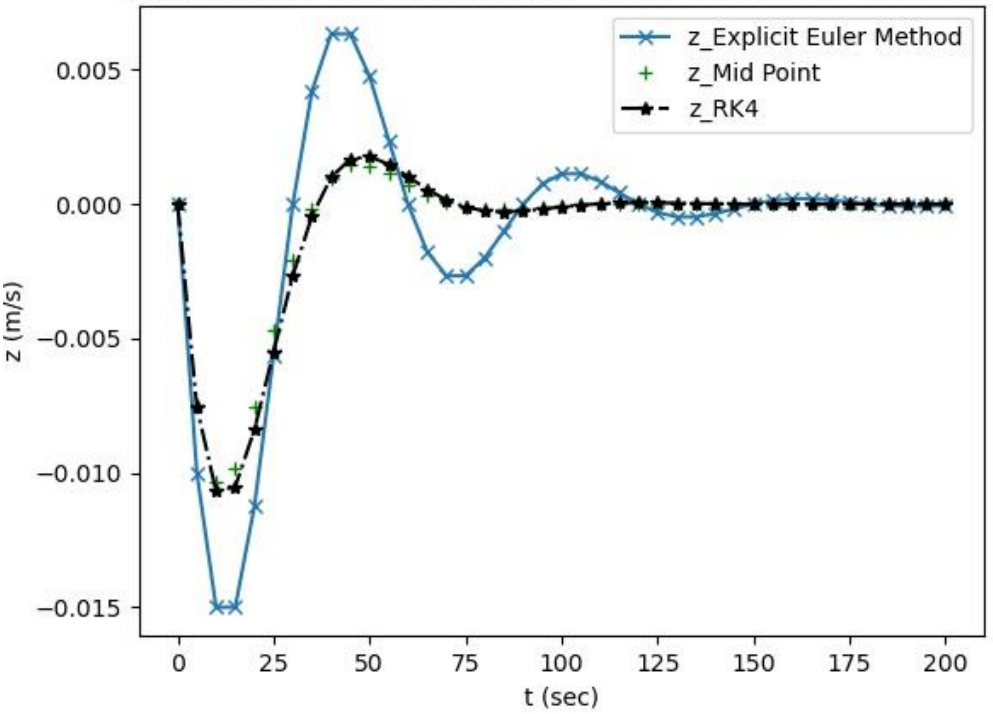


Figure 1. y vs. t for ODE : c= 0.10000kg/s& h=10.00000s

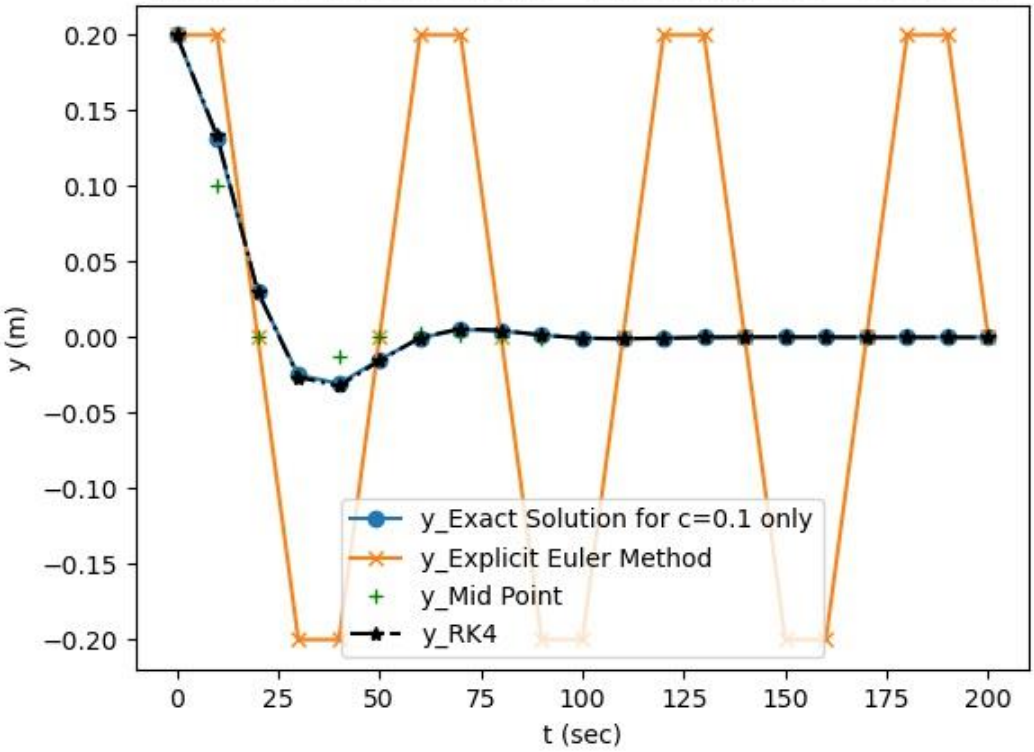


Figure 2. z vs. t for ODE with : c= 0.10000kg/s & h=10.00000s

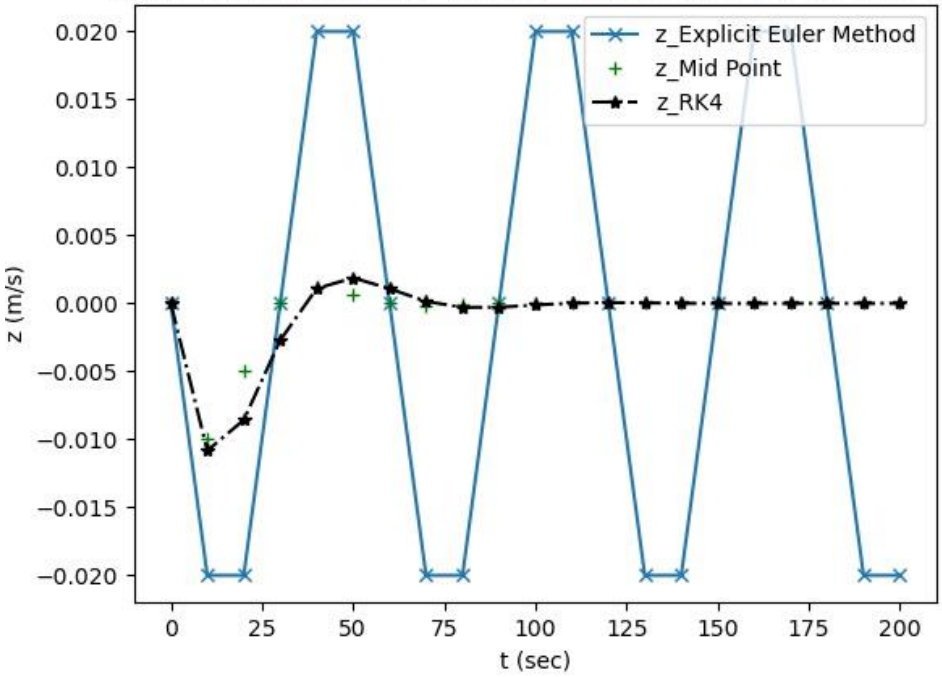


Figure 1. y vs. t for ODE : c= 0.20000kg/s& h= 0.10000s

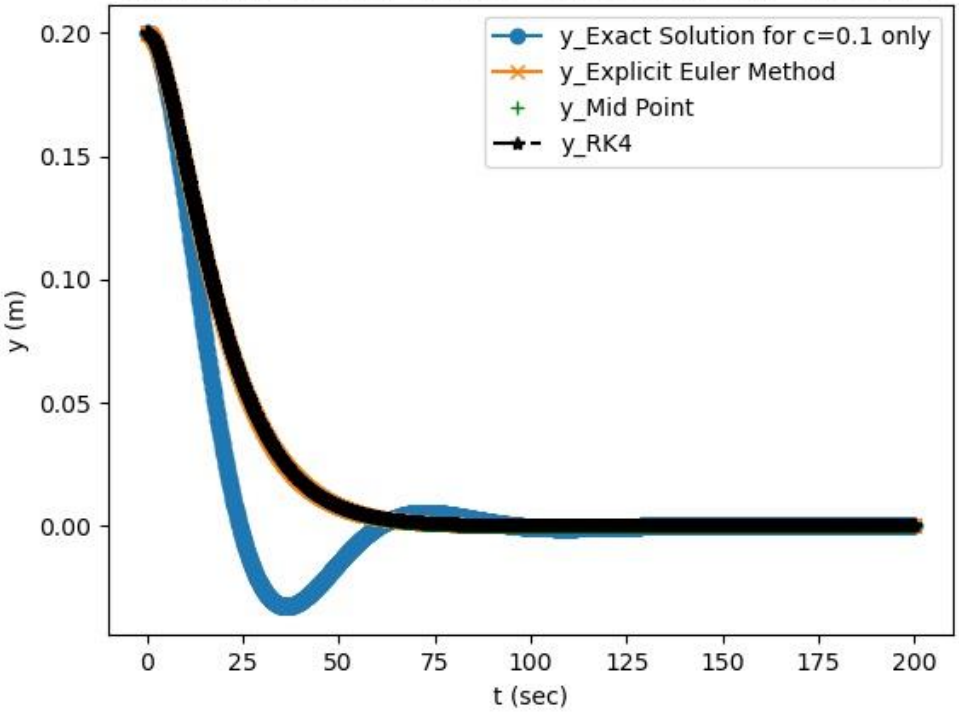


Figure 2. z vs. t for ODE with : c= 0.20000kg/s & h= 0.10000s

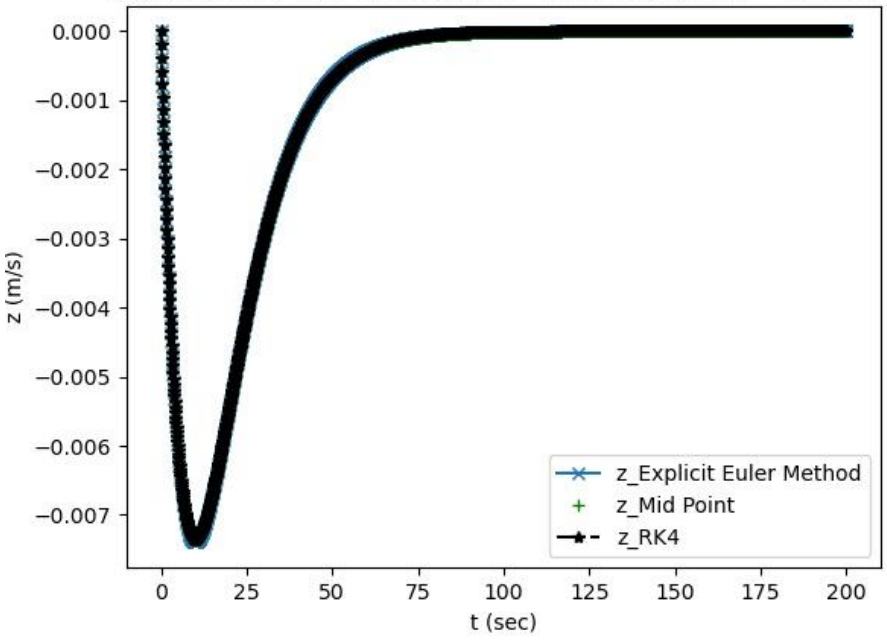


Figure 1. y vs. t for ODE : c= 0.20000kg/s& h= 1.00000s

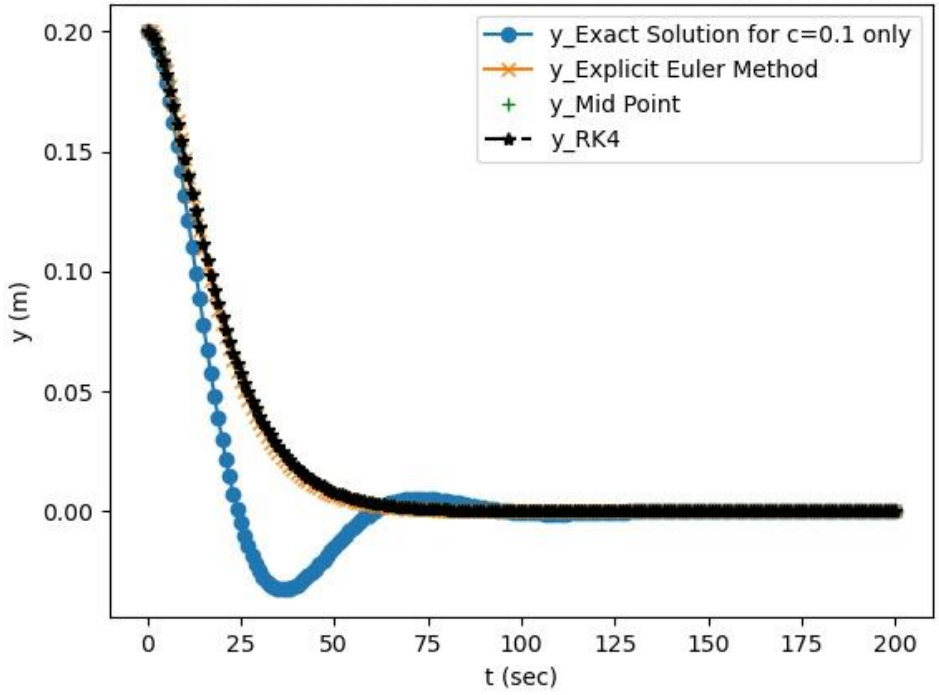


Figure 2. z vs. t for ODE with : c= 0.20000kg/s & h= 1.00000s

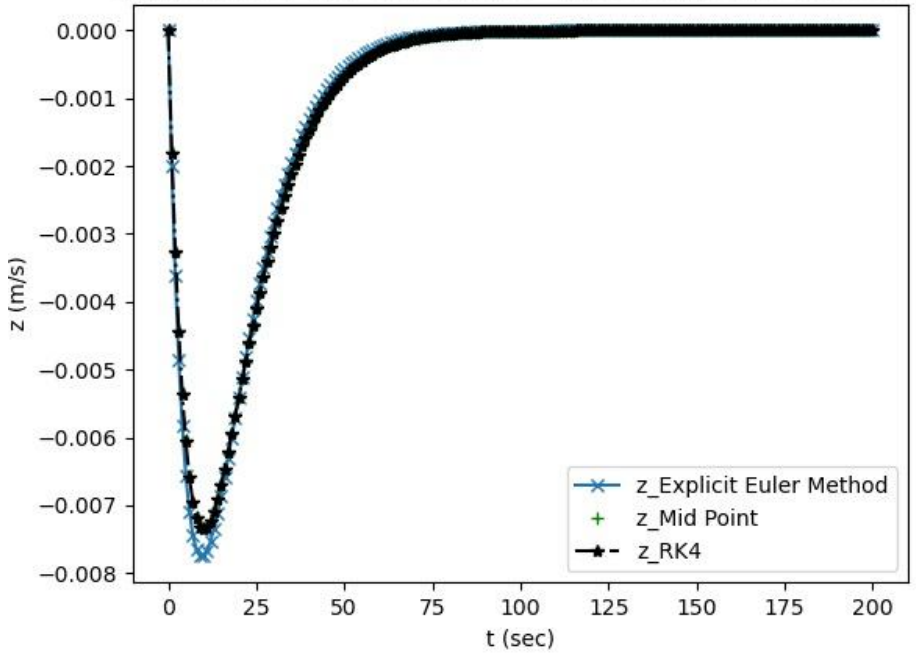


Figure 1. y vs. t for ODE : c= 0.20000kg/s& h= 2.50000s

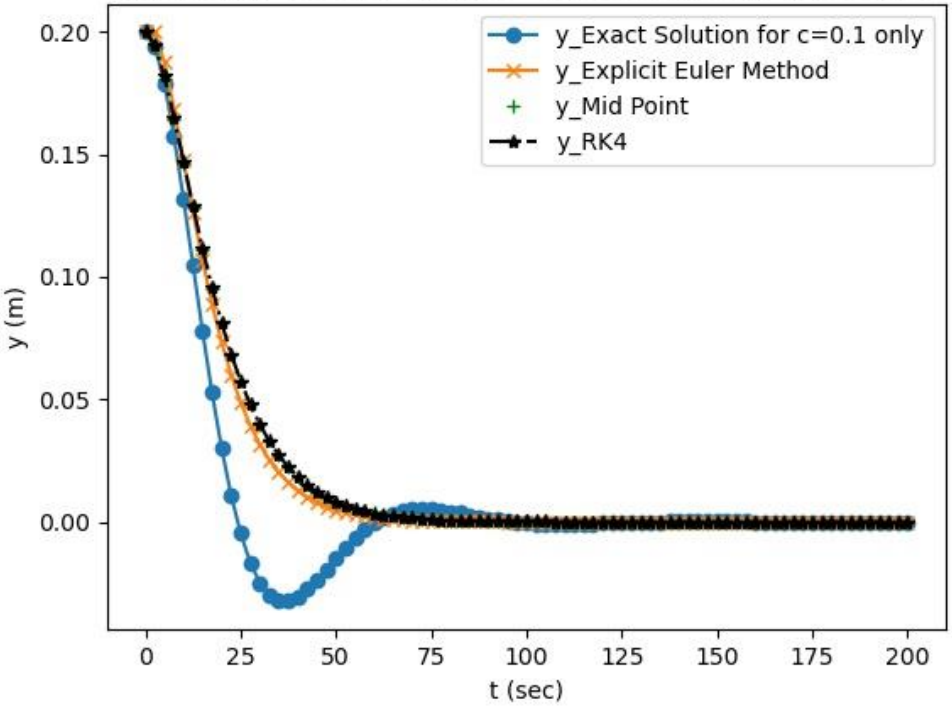


Figure 2. z vs. t for ODE with : c= 0.20000kg/s & h= 2.50000s

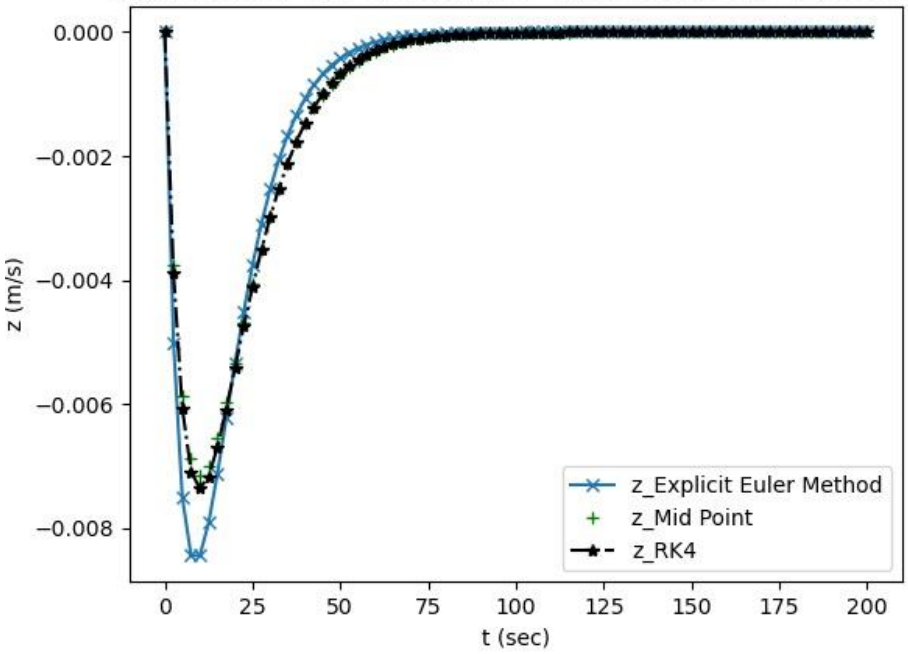


Figure 1. y vs. t for ODE : c= 0.20000kg/s& h= 5.00000s

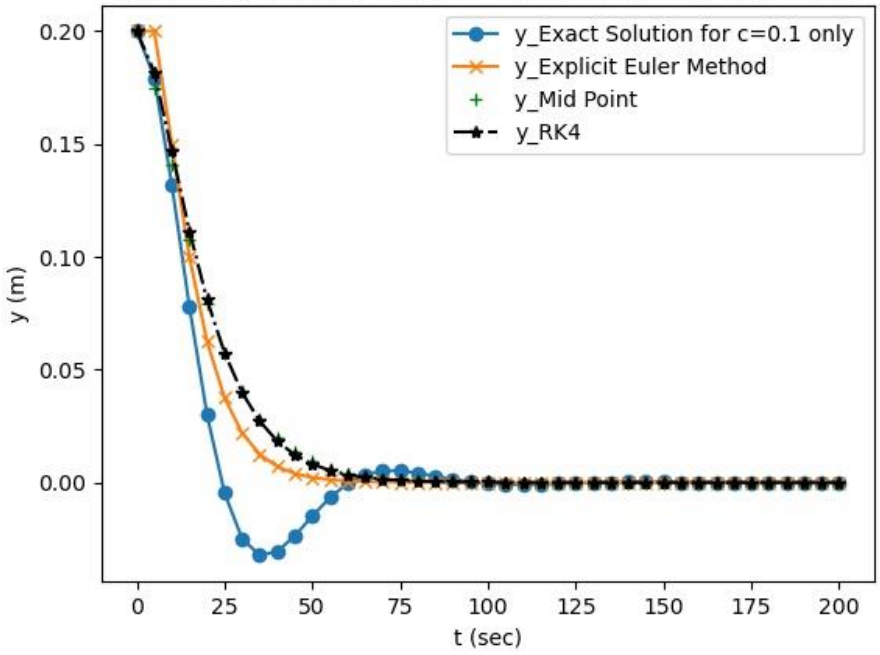


Figure 2. z vs. t for ODE with : c= 0.20000kg/s & h= 5.00000s

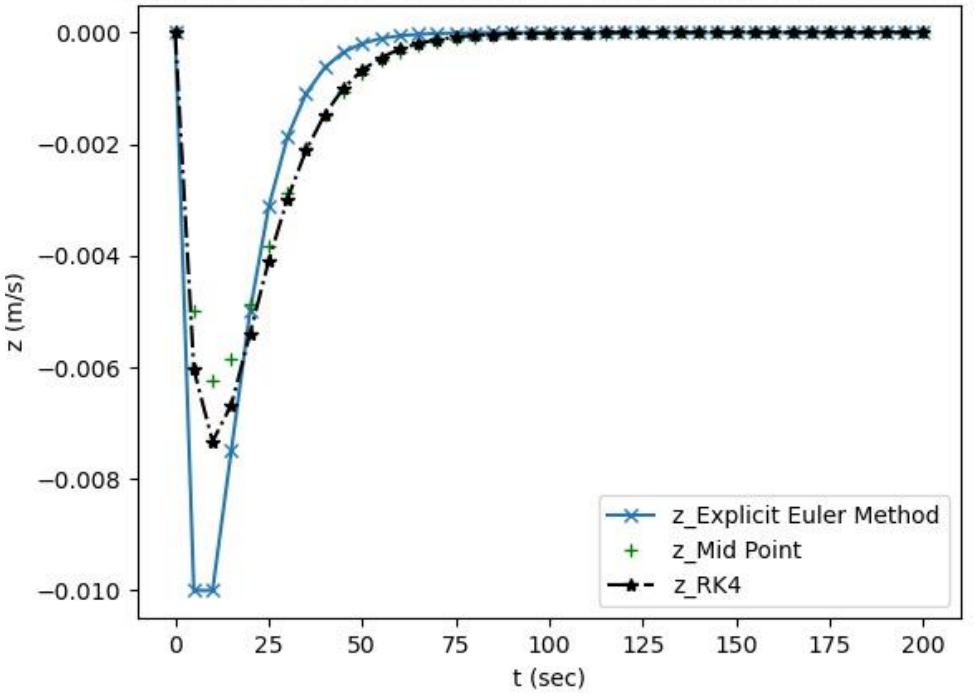


Figure 1. y vs. t for ODE : c= 0.20000kg/s& h=10.00000s

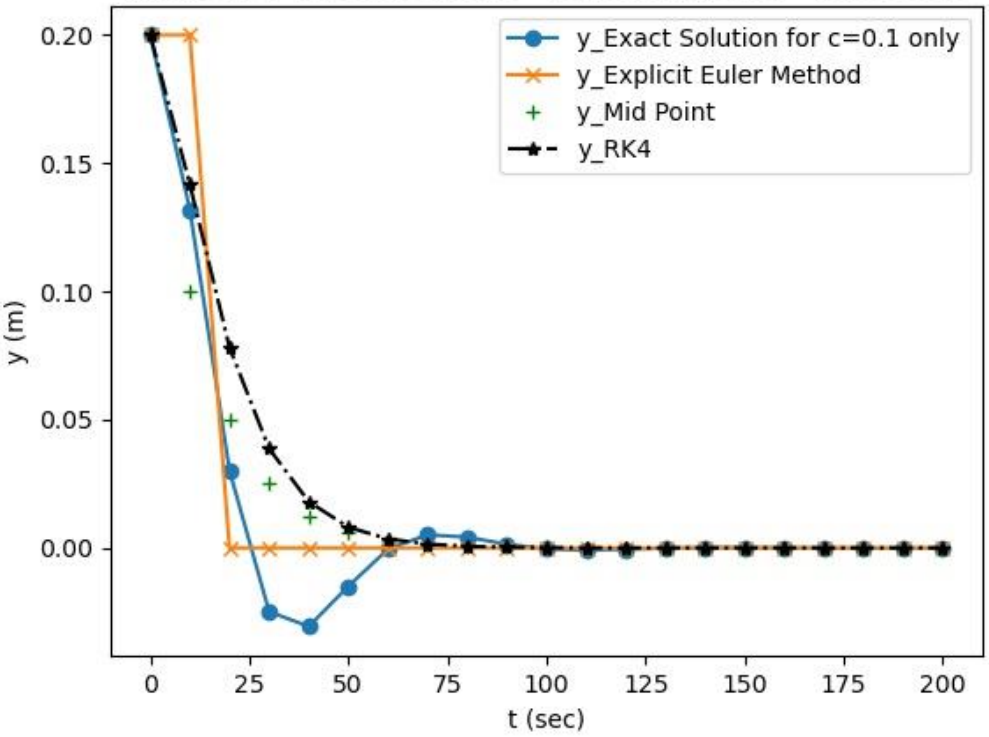


Figure 2. z vs. t for ODE with : c= 0.20000kg/s & h=10.00000s

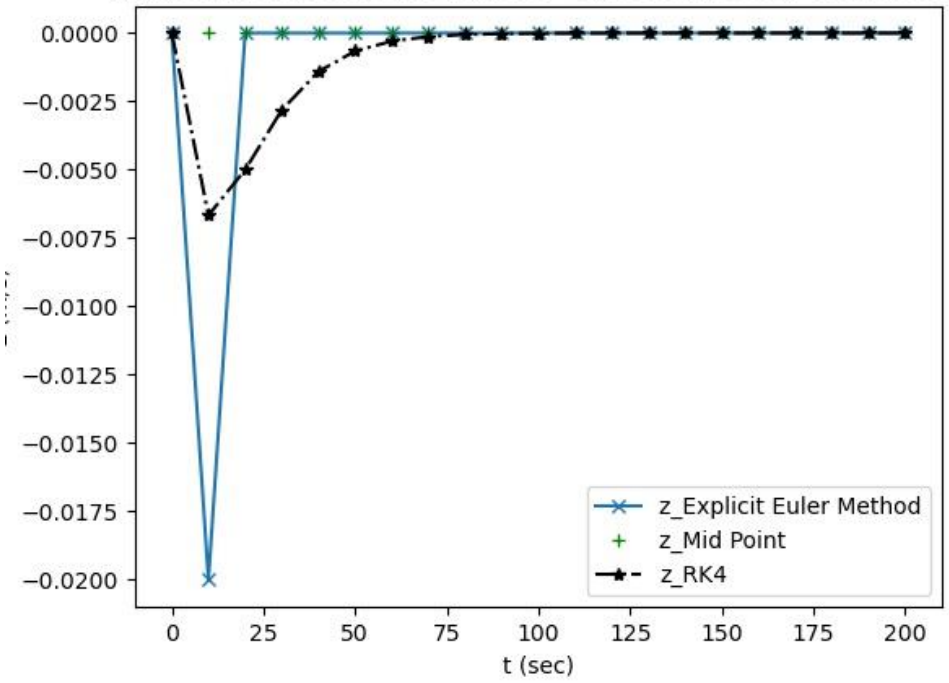


Figure 1. y vs. t for ODE : c= 1.50000kg/s& h= 0.10000s

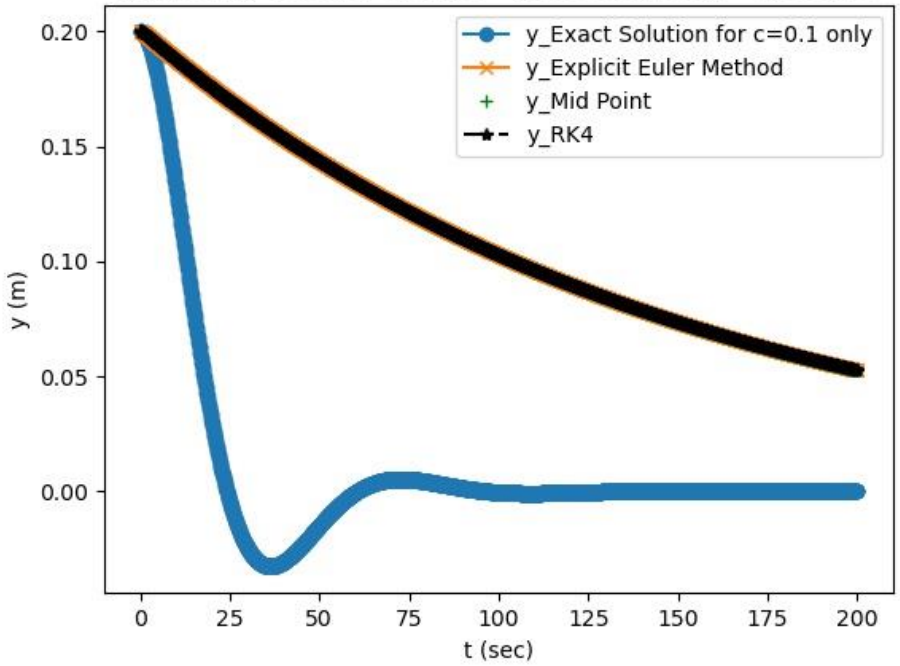


Figure 2. z vs. t for ODE with : c= 1.50000kg/s & h= 0.10000s

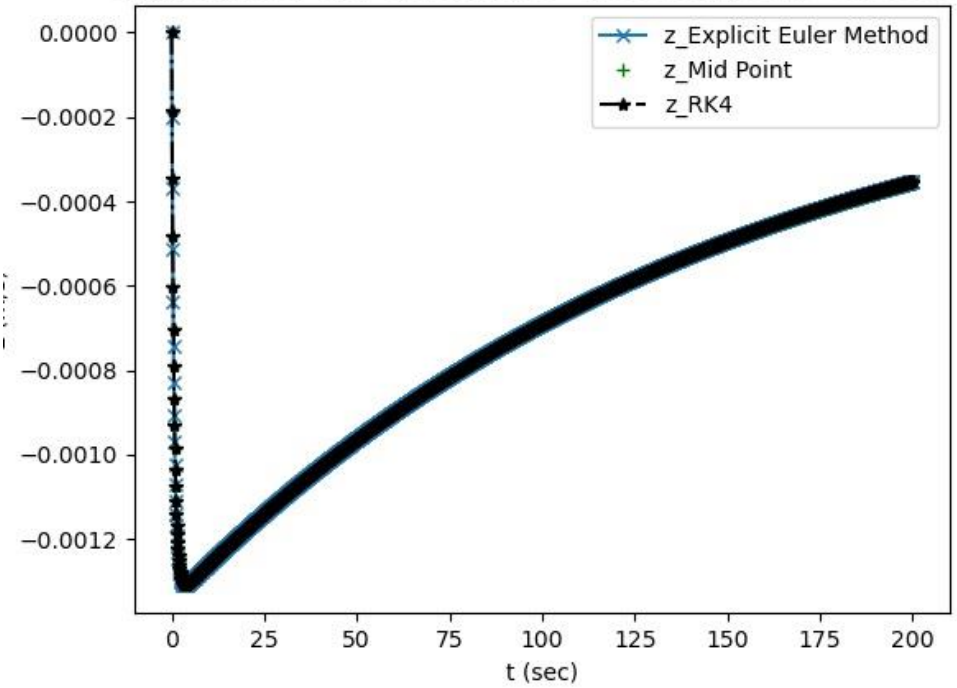


Figure 1. y vs. t for ODE : c= 1.50000kg/s& h= 1.00000s

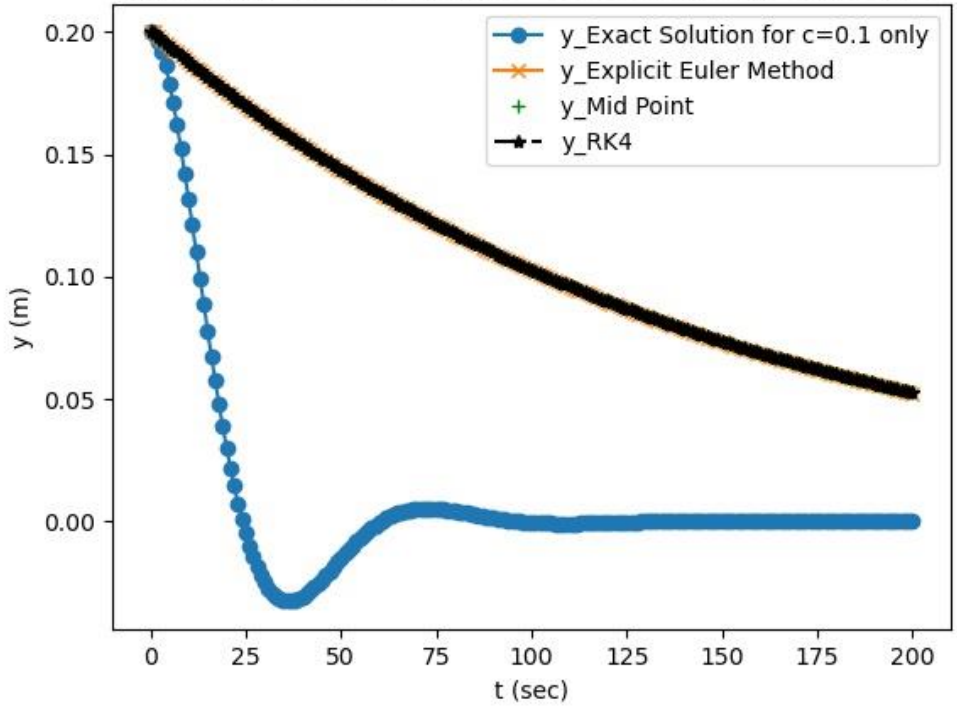


Figure 2. z vs. t for ODE with : c= 1.50000kg/s & h= 1.00000s

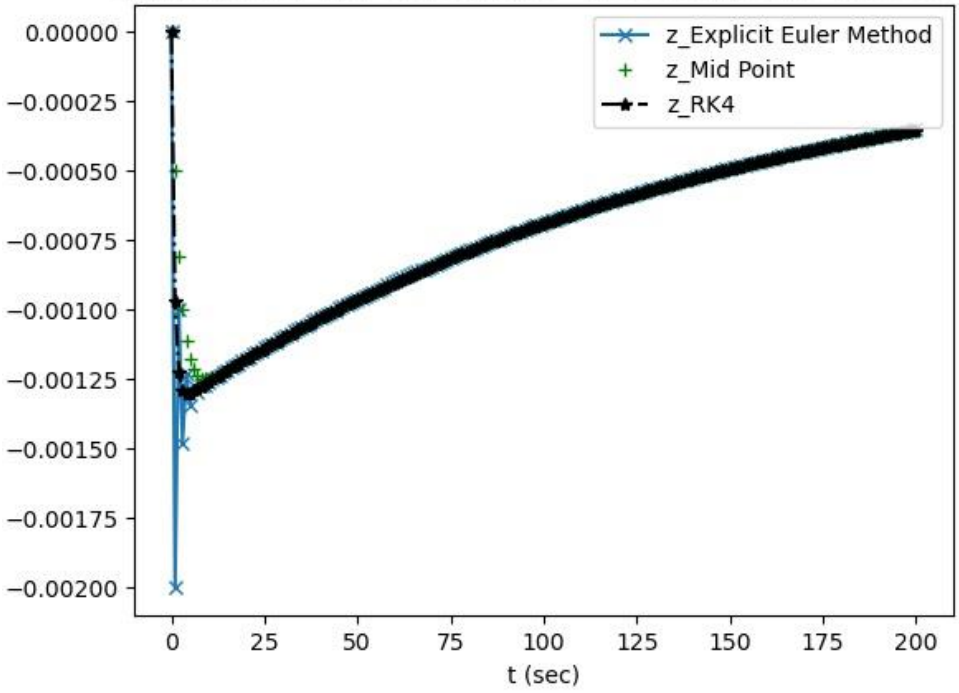


Figure 1. y vs. t for ODE : c= 1.50000kg/s& h= 2.50000s

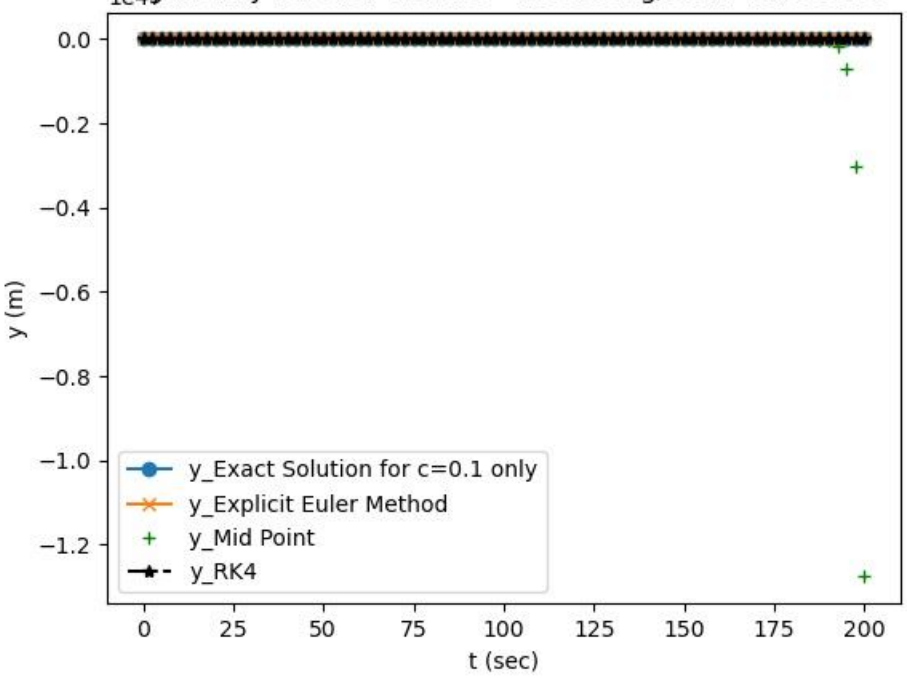


Figure 2. z vs. t for ODE with : c= 1.50000kg/s & h= 2.50000s

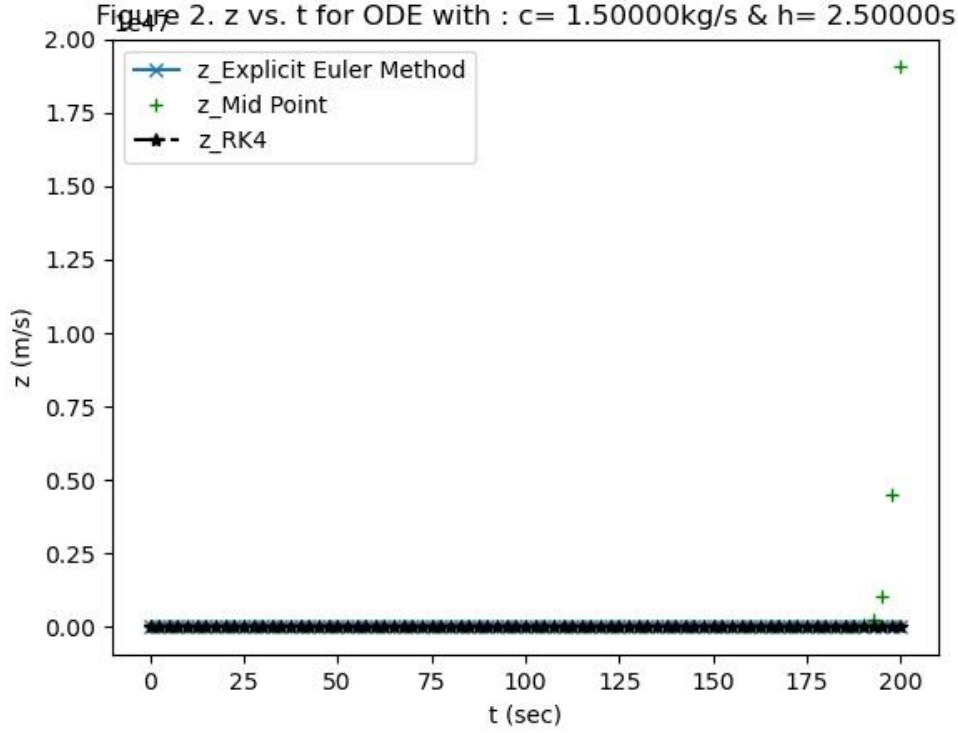


Figure 1. y vs. t for ODE : c= 1.50000kg/s& h= 5.00000s

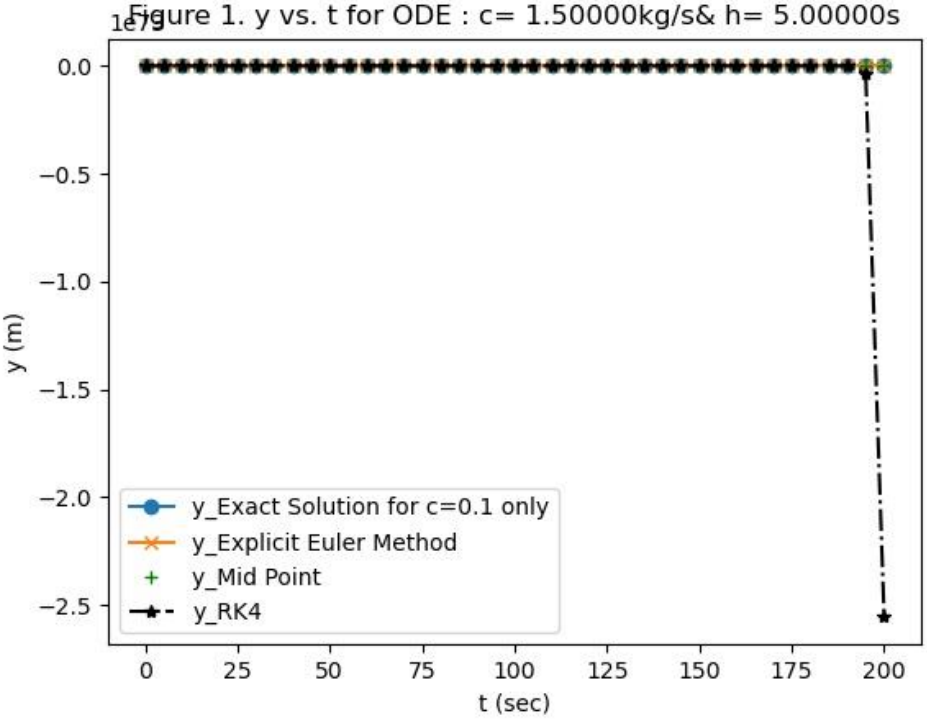


Figure 2. z vs. t for ODE with : c= 1.50000kg/s & h= 5.00000s

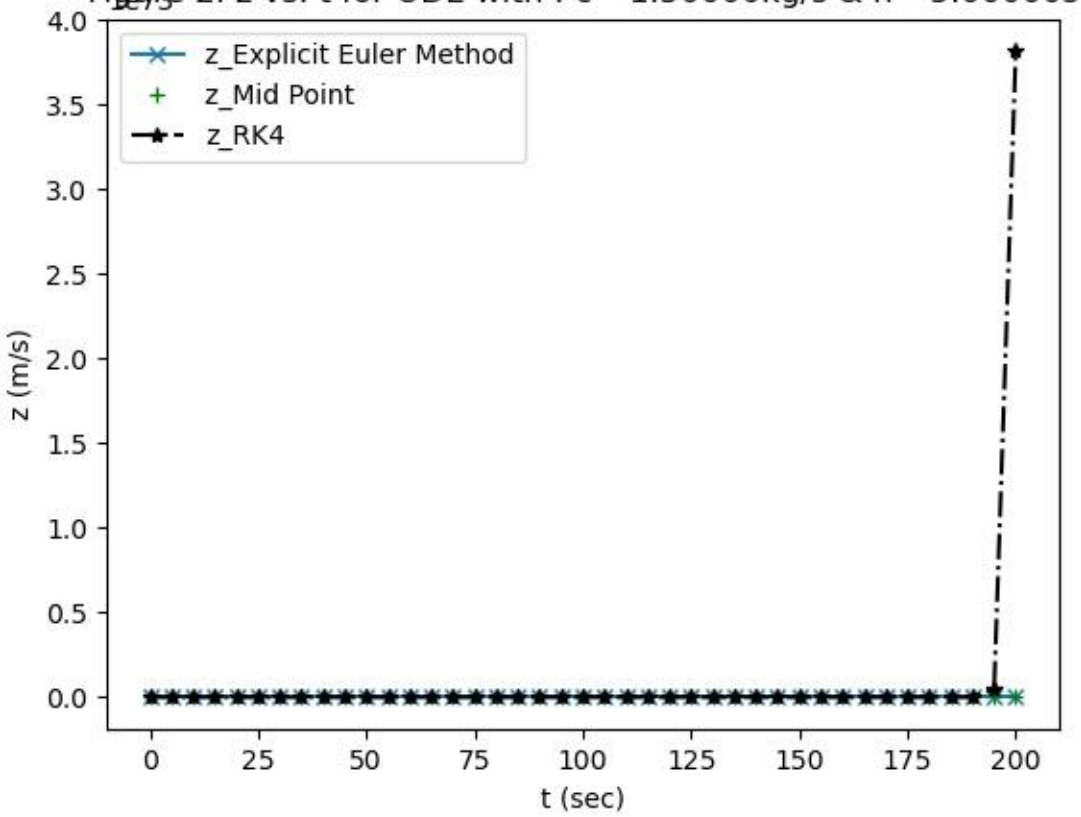


Figure 1. y vs. t for ODE : c= 1.50000kg/s& h=10.00000s

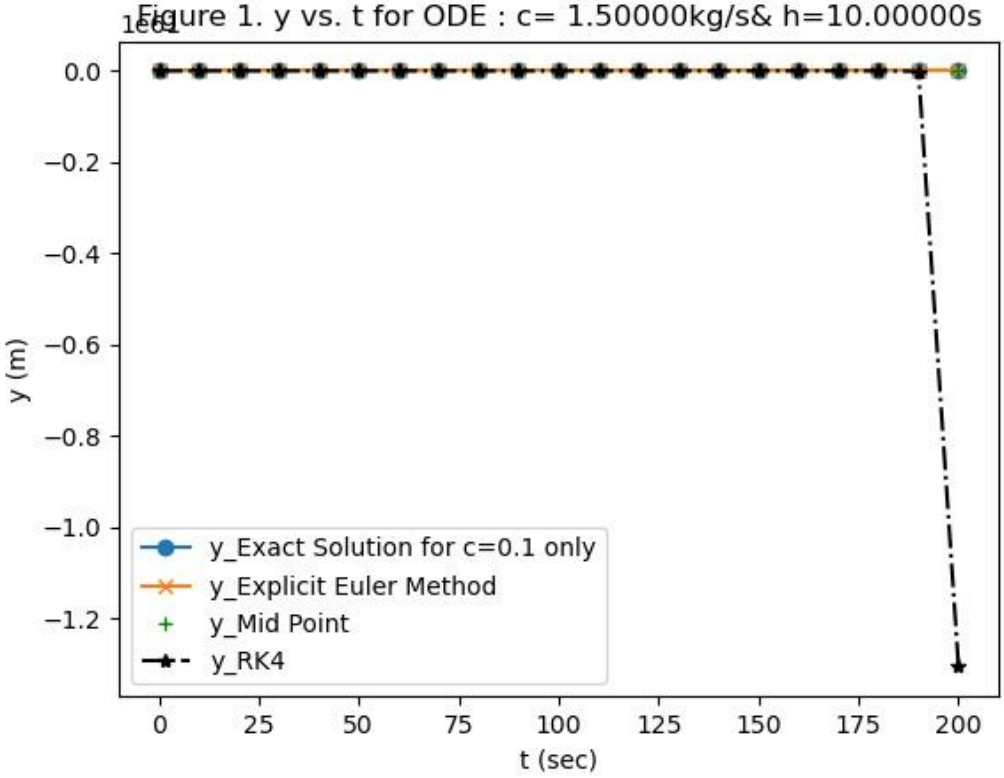
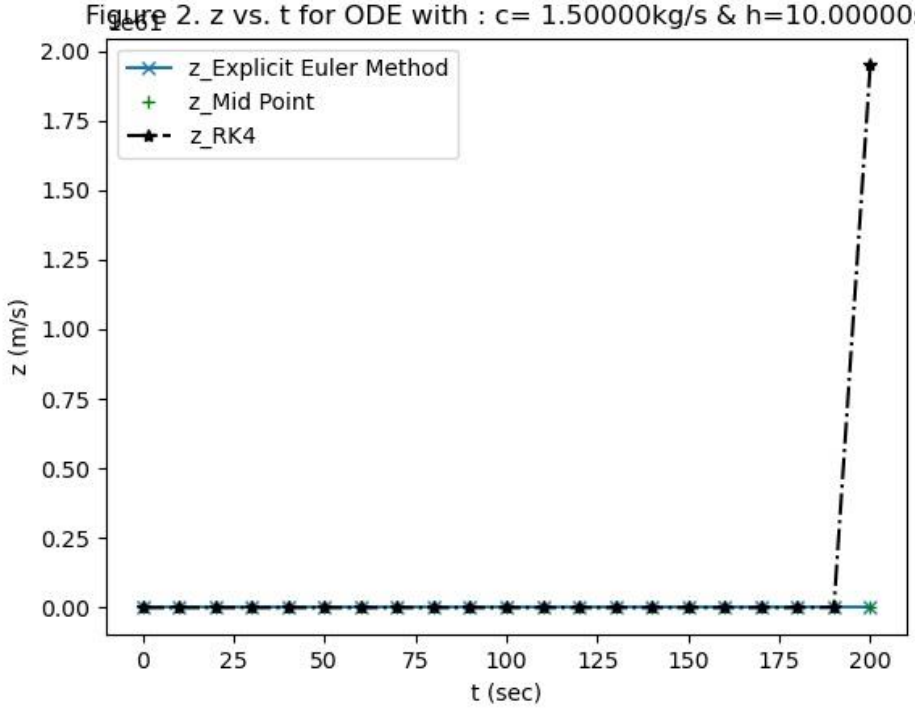


Figure 2. z vs. t for ODE with : c= 1.50000kg/s & h=10.00000s



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.