RAMAKRISHNA MISSION VIVEKANANDA CENTENARY COLLEGE

KOLKATA - 700118



**MATHEMATICAL PHYSICS PRACTICAL NOTEBOOK**

[DSE – 1]

SUBMITTED BY –

ANIK MANDAL

(B.Sc. 5th semester student of Department of Physics)

COLLEGE ROLL NO.: 157

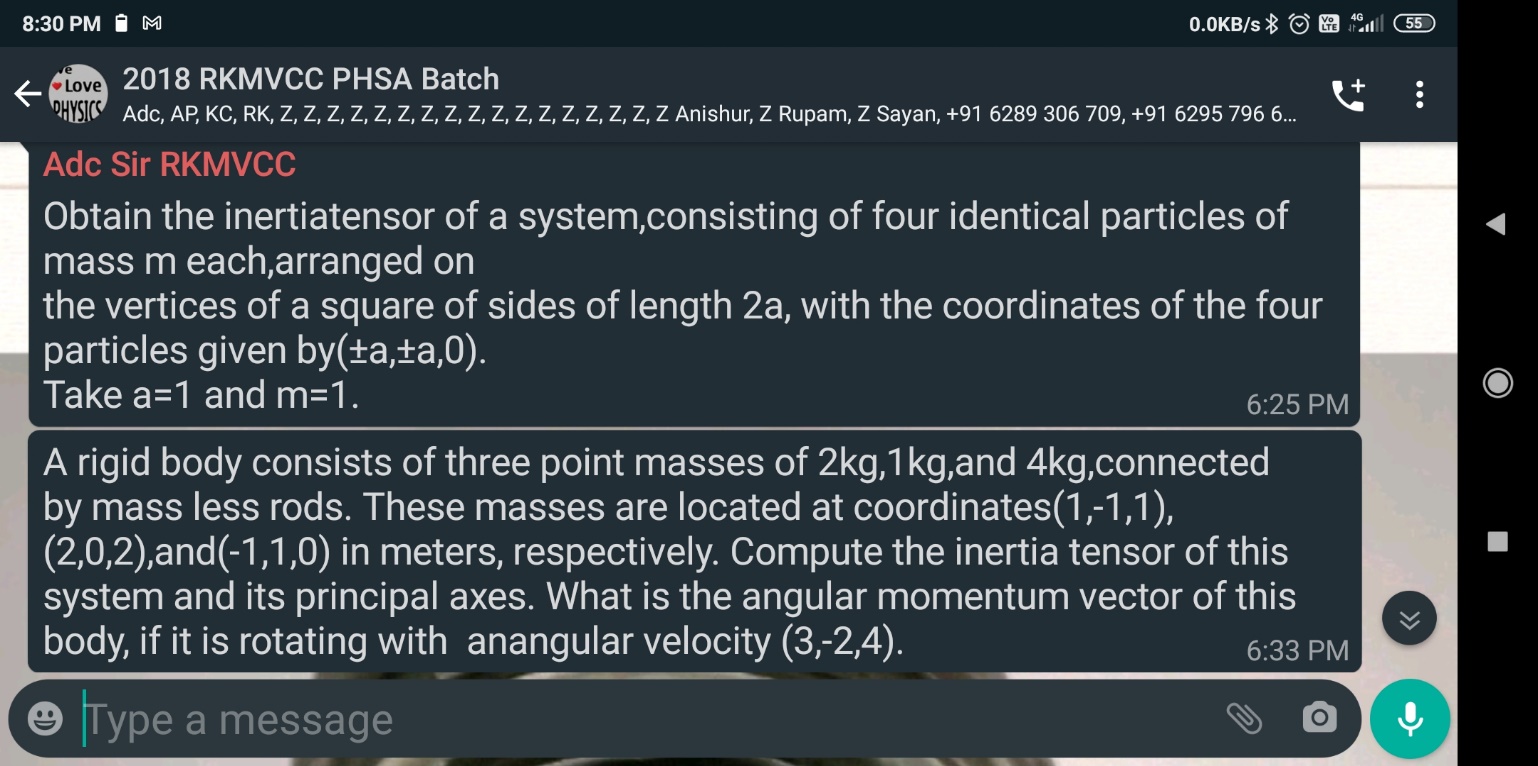
REGESTRATION NO.: AO1-1152-111-042-2018

ADMIT NO.: 2020151031

YEAR: 2021

**CONTENT**

|  |  |  |
| --- | --- | --- |
| Serial No. | Title | Page No. |
| 1 | Determination of Inertia tensor – example 1 |  |
| 2 | Determination of Inertia tensor – example 2 with Inertia tensor module |  |
| 3 | Determination of Principle axis of Inertia Tensor and angular momentum vector |  |
| 4 | IVP: Euler’s method for solving 2nd order ODE |  |
| 5 | Comparison: Over damping, critical damping, small damping |  |
| 6 | Comparison: Euler’s method of approx. and RK-2 method of approx. |  |
| 7 | IVP: using RK2 method – example 1 |  |
| 8 | IVP: using RK2 method – example 2 |  |
| 9 | IVP: using RK2 method – example 3 |  |
| 10 | IVP: using RK4 method |  |
| 11 | BVP: using Linearisation method |  |
| 12 |  |  |
| 13 |  |  |
| 14 |  |  |
| 15 |  |  |

**1.**

Program:

# Determination of Inertia Tensor:

import numpy as np

M = [1, 1, 1, 1]

pos = [[1, 1, 0], [1, -1, 0], [-1, 1, 0], [-1, -1, 0]]

I = np.zeros((3,3))

for m in range(3):

for n in range(3):

if m == n:

for k in range(len(M)):

s = 0

for i in range(3):

if i != m:

s = s + M[k]\*pos[k][i]\*\*2

I[m][n] = I[m][n] + s

else:

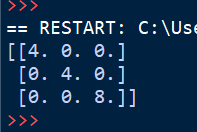
for k in range(len(M)):

s = - M[k]\*pos[k][m]\*pos[k][n]

I[m][n] = I[m][n] + s

print(I)

Output:



1. **Determine inertial tensor for 1(1,0,0), 0.5(0,1,2), 2(0,2,1) masses.**

Program:

# Inertia Tensor Module:

import numpy as np

def Inertia\_T(M, pos):

I = np.zeros((3,3))

for m in range(3):

for n in range(3):

if m == n:

for k in range(len(M)):

s = 0

for i in range(3):

if i != m:

s = s + M[k]\*pos[k][i]\*\*2

I[m][n] = I[m][n] + s

else:

for k in range(len(M)):

s = - M[k]\*pos[k][m]\*pos[k][n]

I[m][n] = I[m][n] + s

return I

# Determination of inertia tensor(Ex2 with Module):

import numpy as np

from LocalModule.Inertia\_T import \*

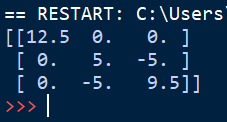
M = [1, 0.5, 2]

pos = [[1,0,0], [0,1,2], [0,2,1]]

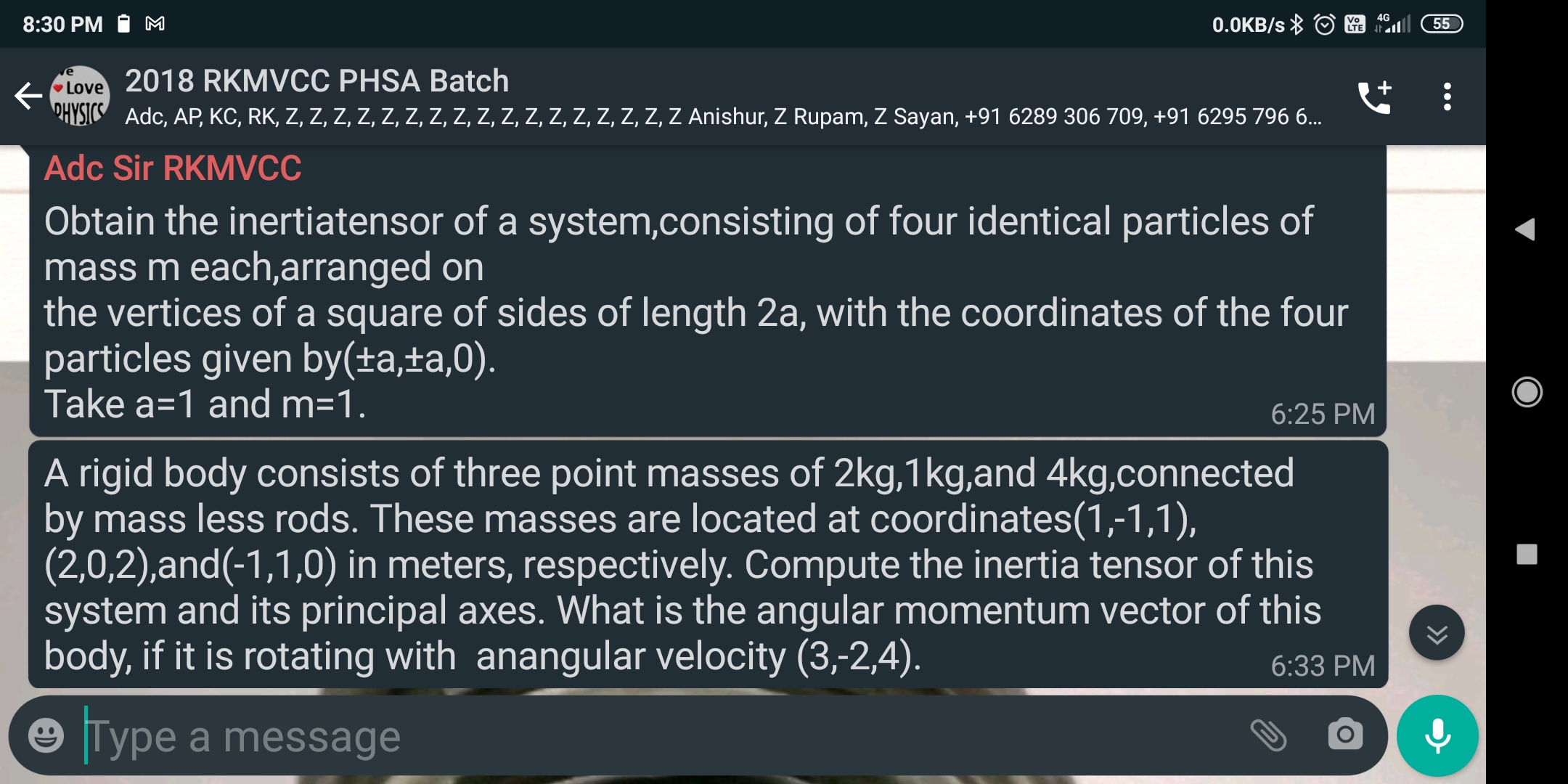
I = Inertia\_T(M,pos)

print(I)

Output:







Program:

//Inertia tensor module is given (2).

# Matrix Order:

def M\_Order(xx): # Matrix is defined as as [[]]

r = len(xx)

c = len(xx[0])

o = [r, c]

return o

# Matrix Multiplication:

def M\_Multiplication(xx,yy):

Ox = M\_Order(xx)

Oy = M\_Order(yy)

zz = np.zeros((Ox[0],Oy[1]))

if Ox[1] == Oy[0]:

for i in range(0,Ox[0]):

for j in range(0,Oy[1]):

for k in range(0,Ox[1]):

zz[i][j] = zz[i][j] + xx[i][k]\*yy[k][j]

return zz

# Determination of Principle axis of an inertia tensor and Angular momentum:

import scipy.linalg as al

from LocalModule.MatrixOperation import \*

from LocalModule.Inertia\_T import \*

M = [2, 1, 4]

Pos = [[1, -1, 1], [2, 0, 2], [-1, 1, 0]]

I = Inertia\_T(M, Pos)

evall, evec = al.eig(I)

eveci = al.inv(evec)

print("Principle axis vectors:\n", evec,)

print("And corrasponding eigen values:\n",evall)

Id = M\_Multiplication(M\_Multiplication(eveci,I),evec)

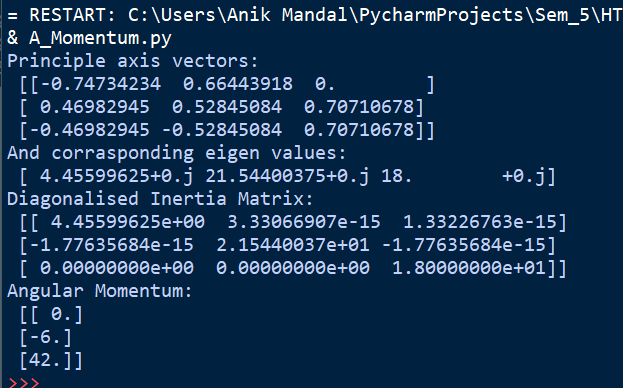
print("Diagonalised Inertia Matrix:\n",Id)

omg = [[3], [-2], [4]]

L = M\_Multiplication(I, omg)

print('Angular Momentum:\n', L)

Output:



1. **Initial value problem: solve 2nd order ODE using Euler’s method. Given function and boudary conditions: y’’+2by’+w2y=0, b=1, w2=0.5, y(0)=0, y’(0)=2**

Program:

# IVP: Euler's method for solving 2nd order ODE:

import numpy as np

import matplotlib.pyplot as plt

# y'' + 2b \*y' + w\*\*2 \* y = f(x) let, b = 1, w = 0.5, f(x) = 0

# conditions:

t0 = 0

y0 = 0

yp0 = 2

b = 1

f = 0.5 # w\*\*2

ti = t0

tf = 2

n = 17

h = (tf-ti)/(n-1)

yi = y0

ypi = yp0

y2pi = -2\*b\*ypi-f\*yi

tt = [t0]

yy = [y0]

for i in range(1, n+1):

ti = ti + h

yi = yi + h\*ypi

ypi = ypi + h\*y2pi

y2pi = -2 \* b \* ypi - f \* yi

tt.append(ti)

yy.append(yi)

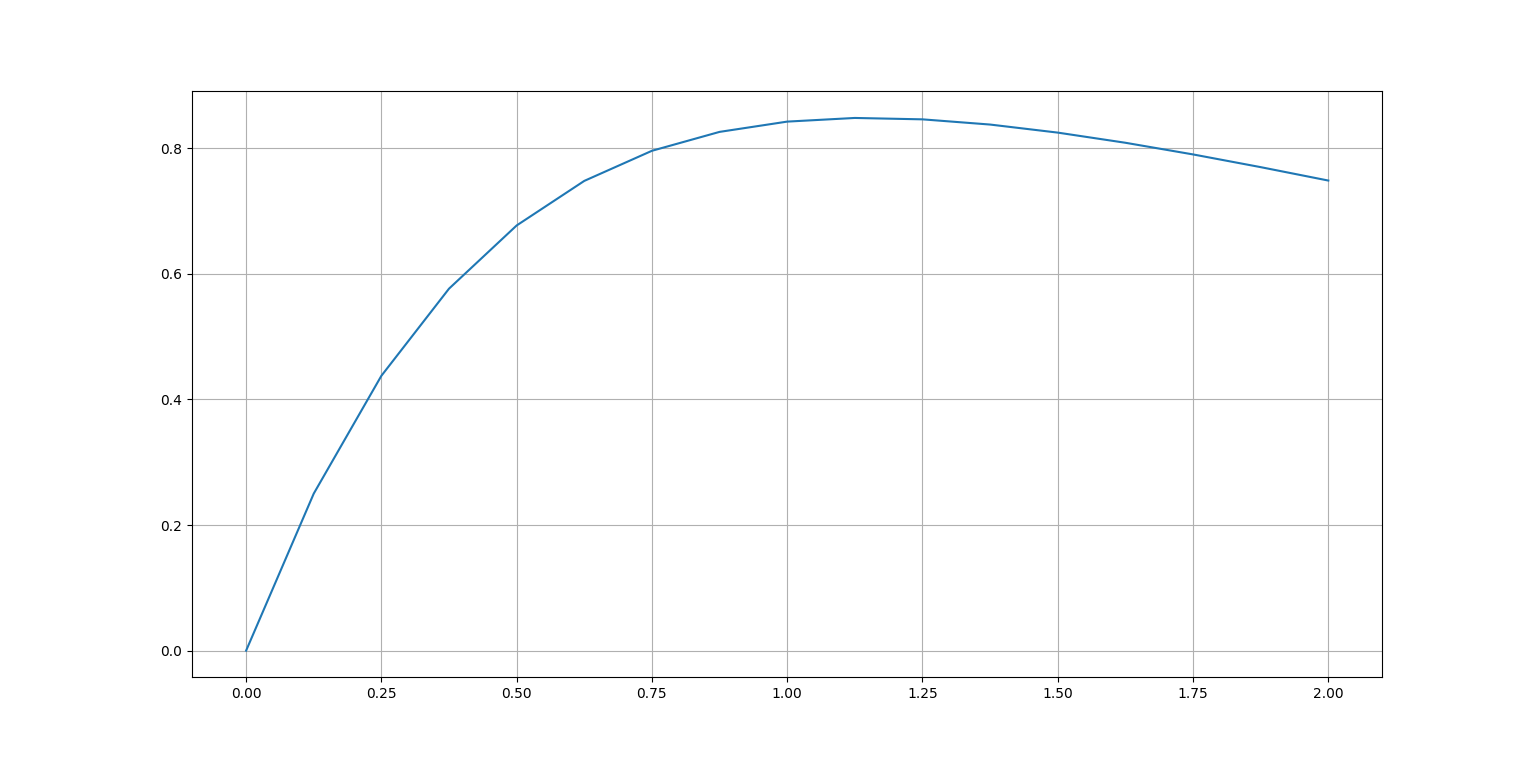
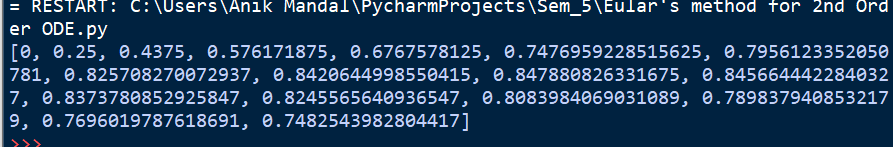
print(yy)

plt.plot(tt, yy)

plt.grid()

plt.show()

Output:



1. **Comparison: Over damping(b=20),critical damping(b=10), small damping(b=5), w = 100.**

Program:

# Damping(over, crtical, small):

import numpy as np

import matplotlib.pyplot as plt

# y’’ + 2b \*y’ + w\*\*2 \* y = f(x) , f(x) = 0

B = [20, 10, 5]

for I in range(len(B)):

# conditions:

t0 = 0

y0 = 0

yp0 = 2

f = 100 # w\*\*2

b=B[i]

ti = t0

tf = 2

n = 129

h = (tf-ti)/(n-1)

yi = y0

ypi = yp0

y2pi = -2\*b\*ypi-f\*yi

tt = [t0]

yy = [y0]

for j in range(1, n):

ti = ti + h

yi = yi + h\*ypi

ypi = ypi + h\*y2pi

y2pi = -2 \* b \* ypi – f \* yi

tt.append(ti)

yy.append(yi)

plt.plot(tt, yy)

tt.clear()

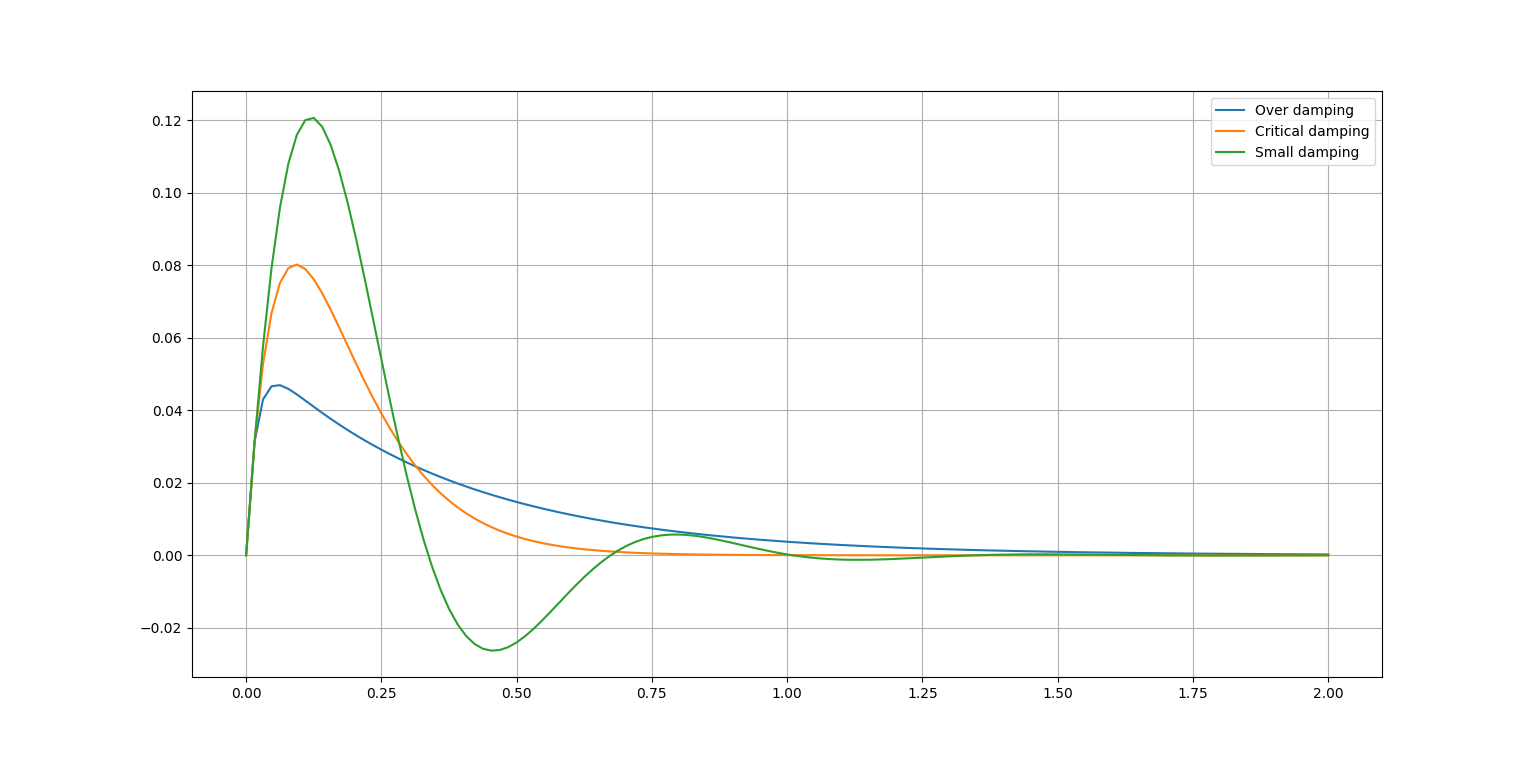
yy.clear()

plt.legend([“Over damping”,”Critical damping”,”Small damping”])

plt.grid()

plt.show()

Output:



1. **Comparison: Euler’s method of approximation , RK-2 method of approximation with the actual curve.**

Program:

import numpy as np

import matplotlib.pyplot as plt

x0 = 0

y0 = 0

h = 0.1

n = 40

x1 = x0

y1 = y0

xx = [x0]

yy = [y0]

zz = [0]

xr = [x0]

yr = [y0]

def m(x, y):

s = (2\*x-2\*x\*\*3)\*np.exp(-x\*\*2)

return s

for I in range(n):

y1 = y1 + h\*m(x1, y1)

x1 = x1 + h

xx.append(x1)

yy.append(y1)

z = (x1\*\*2)\*np.exp(-x1\*\*2)

zz.append(z)

x1 = x0

y1 = y0

yd = y0

for I in range(n):

m1 = m(x1, yd)

yd = yd + h\*m1

x1 = x1 + h

m2 = m(x1, yd)

y1 = y1 + h\*(m1 + m2)/2

xr.append(x1)

yr.append(y1)

plt.plot(xx, zz, ‘-+r’)

plt.plot(xx, yy, ‘c’)

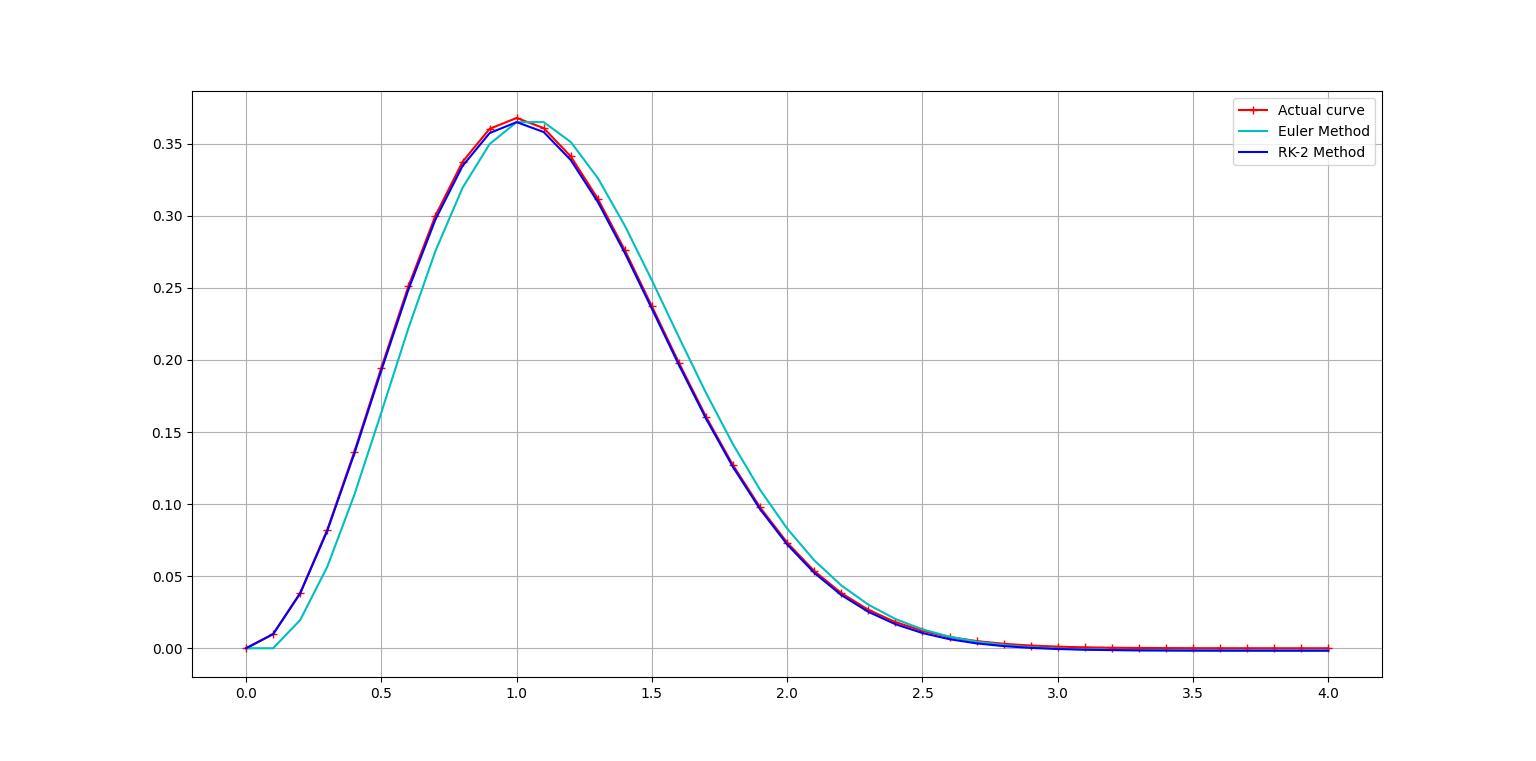
plt.plot(xr, yr, ‘b’)

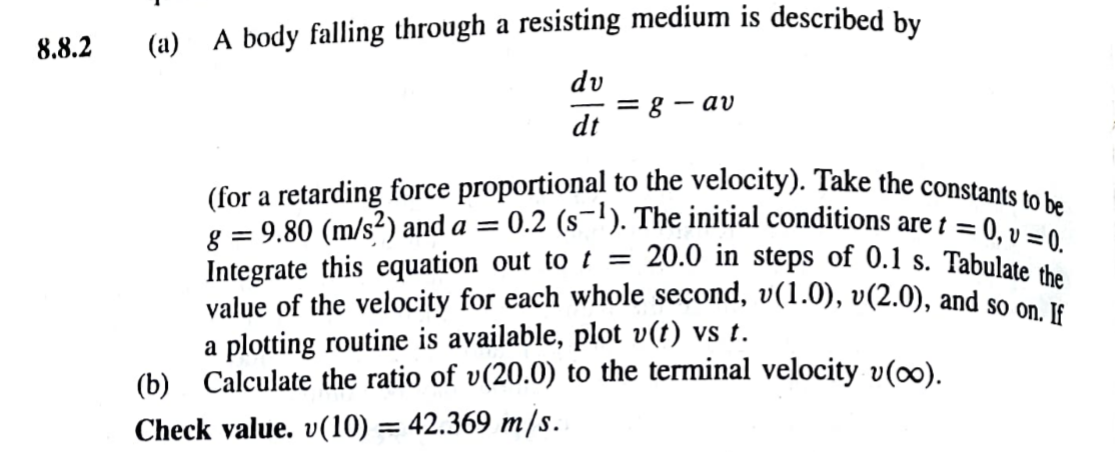
plt.legend([‘Actual curve’, ‘Euler Method’, ‘RK-2 Method’])

plt.grid()

plt.show()

Output:





Program:

# Integration Module:

"""

1. Simpson 1/3rd method with step

"""

import numpy as np

def Integrate(yy, i, f, step):

h = abs(f-i)/(step-1)

s = 0

for i in range(1, step-1):

if i%2 == 1:

s = s + 4\*yy[i]

else:

s = s + 2\*yy[i]

s = (yy[0] + s + yy[step-1])\*h/3

return s

# 8.8.2 IVP: solution using RK-2 Method

import numpy as np

import matplotlib.pyplot as plt

from LocalModule.Integration import \*

import pandas as pd

import sys

t0 = 0

v0 = 0

g = 9.80

a = 0.2

h = 0.1

n = 2001

t1 = t0

v1 = v0

vd = v0

tt = [t0]

vv = [v0]

zz = [0]

def m(t, v):

s = g-a\*v

return s

i = 0

while i < n-1:

m1 = m(t1, vd)

vd = vd + h\*m1

t1 = t1 + h

m2 = m(t1, vd)

v1 = v1 + h\*(m1 + m2)/2

tt.append(t1)

vv.append(v1)

z = (g/a)\*(1-np.exp(-a\*t1))

zz.append(z)

i = i+1

ig = Integrate(vv, t0, t0+200\*h, 201)

print('Value of Integration(0s to 20s)', ig)

print('v(', t0+100\*h, '):', vv[100])

t\_data = []

v\_data = []

for i in range(201):

if round(tt[i],1)%1 == 0:

t\_data.append(tt[i])

v\_data.append(vv[i])

df = pd.DataFrame({'Time(s)': t\_data, 'Velocity(m/s)': v\_data})

df.to\_excel(r'C:\Users\Anik Mandal\Desktop\val.xlsx')

print(df)

### let assume, v(inf)==v(200)

print('v(', t0+200\*h, ')/v(inf):', vv[200]/vv[2000])

plt.plot(tt, vv, 'r', tt, zz, '--c')

plt.xlim(-0.5, 20.5)

plt.legend(['Curve through RK2 method', 'Actual Curve'])

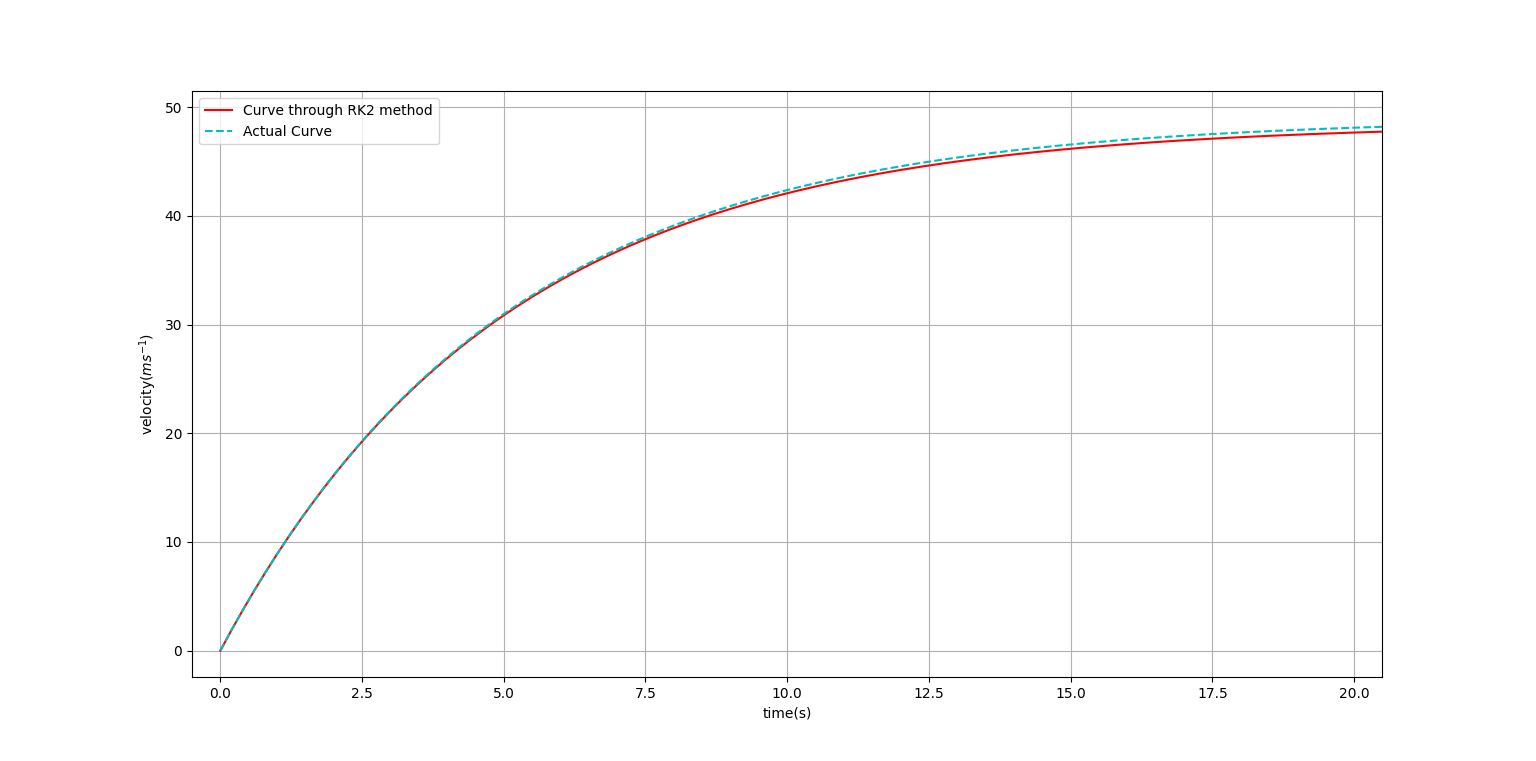
plt.xlabel("time(s)")

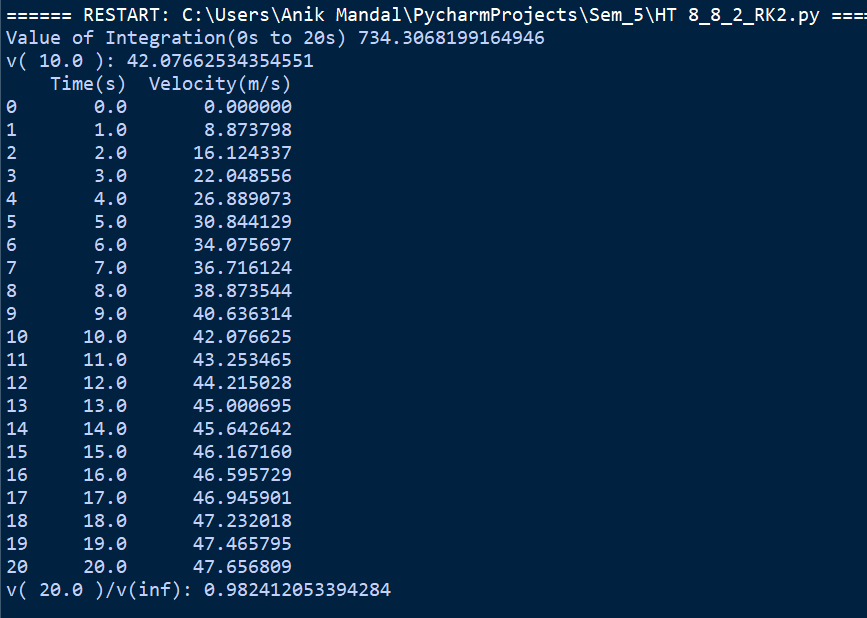
plt.ylabel(r"velocity($ms^{-1}$)")

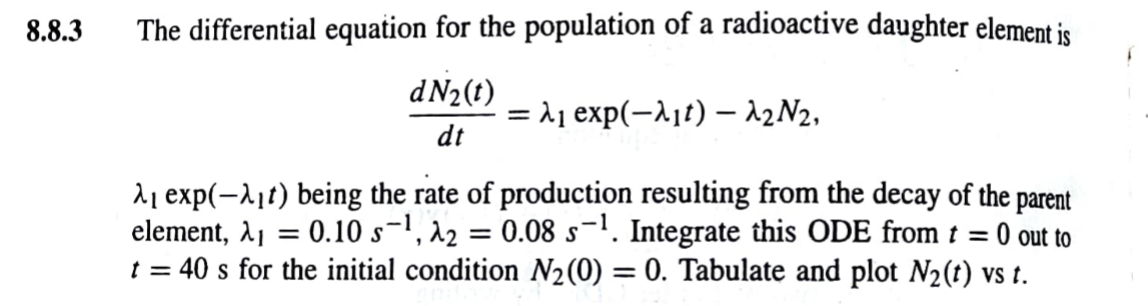
plt.grid()

plt.show()

Output:





Program:

import numpy as np

import matplotlib.pyplot as plt

from LocalModule.Integration import \*

import pandas as pd

t0 = 0

N = 0

l1 = 0.10

l2 = 0.08

h = 0.1

n = 401

t1 = t0

N1 = N

Nd = N

tt = [t0]

NN = [N]

zz = [0]

def m(t, N\_2):

s = l1\*np.exp(-l1\*t)-l2\*N\_2

return s

i = 0

while i < n-1:

m1 = m(t1, Nd)

Nd = Nd + h\*m1

t1 = t1 + h

m2 = m(t1, Nd)

N1 = N1 + h\*(m1 + m2)/2

tt.append(t1)

NN.append(N1)

z = (l1/(l1-l2))\*(np.exp(-l2\*t1)-np.exp(-l1\*t1))

zz.append(z)

i = i+1

ig = Integrate(NN, t0, t0+n\*h, n)

print('Value of Integration(0s to 40s)', ig)

t\_data=[]

N\_data=[]

for i in range (len(tt)):

if round(tt[i],1)%1==0:

t\_data.append(tt[i])

N\_data.append(NN[i])

df = pd.DataFrame({'Time(s)': t\_data, 'N\_2': N\_data})

df.to\_excel(r'C:\Users\Anik Mandal\Desktop\val\_8\_8\_3.xlsx')

print(df)

plt.plot(tt, NN, 'r')

plt.plot(tt, zz, '--c')

plt.xlim(-0.5, 40.5)

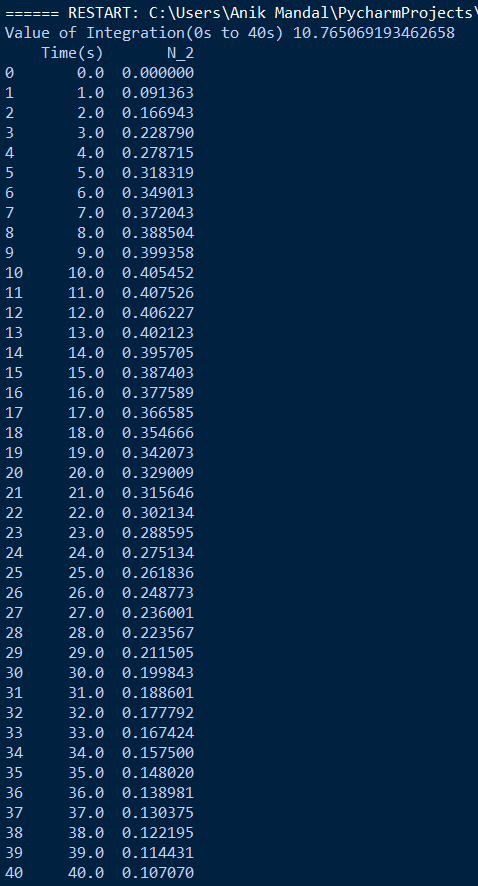
plt.xlabel('Time(s)')

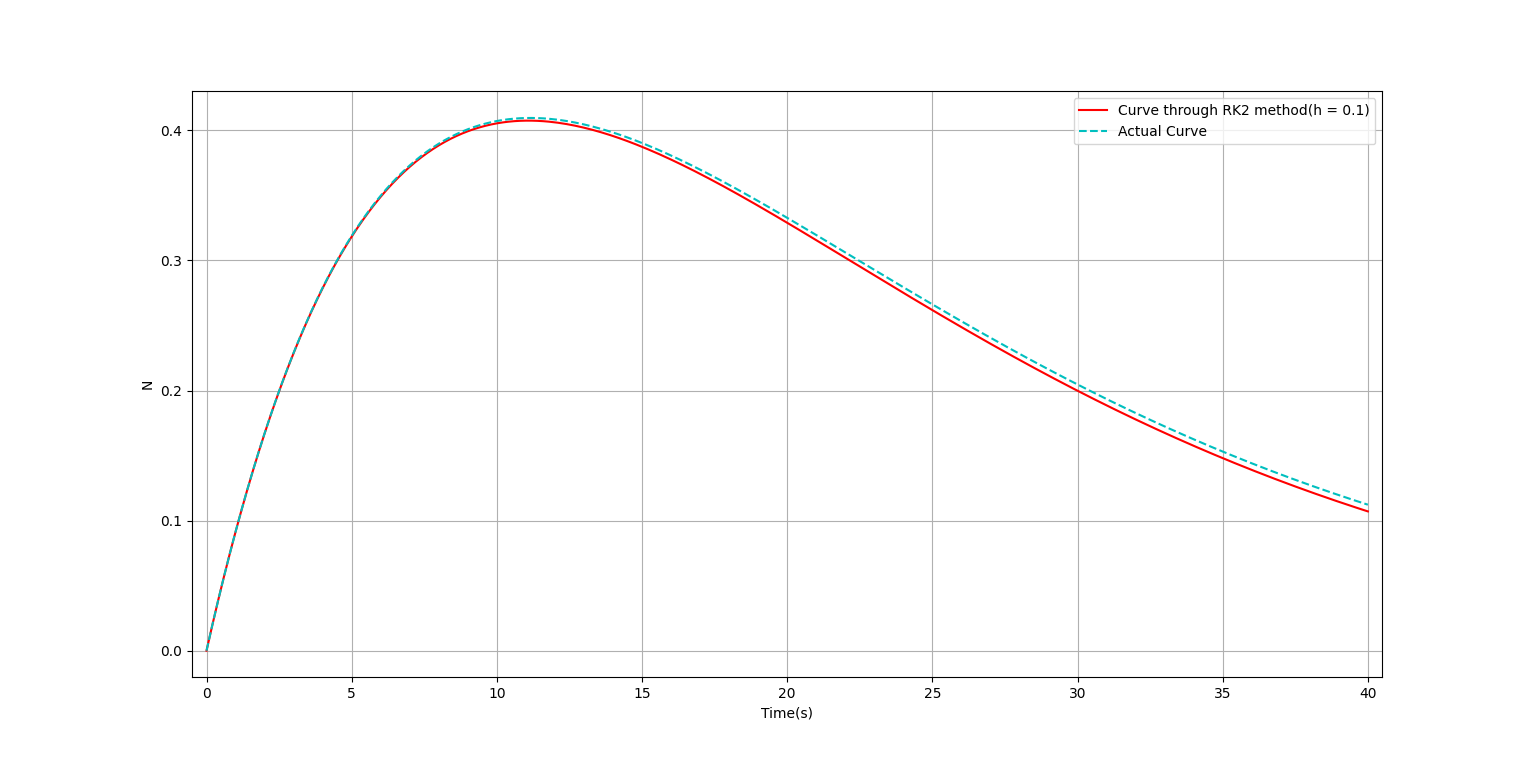
plt.ylabel('N')

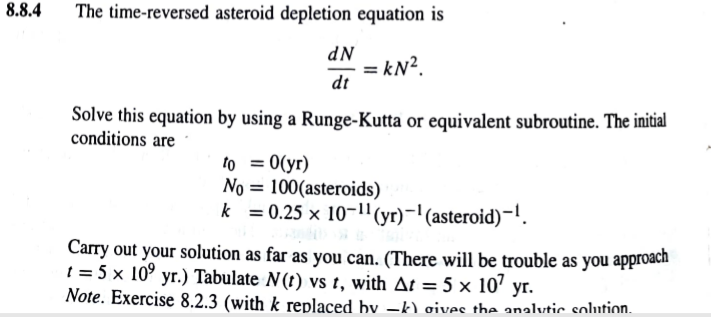
plt.legend(['Curve through RK2 method(h = 0.1)', 'Actual Curve'])

plt.grid()

plt.show()

Output:





Program:

import numpy as np

import matplotlib.pyplot as plt

from LocalModule.Integration import \*

import pandas as pd

t0 = 0

N0 = 100

k = 0.25\*10\*\*-11

h = 5\*10\*\*7 # △T

n = (3.5\*10\*\*9/h)+1

t1 = t0

N1 = N0

Nd = N0

tt = [t0]

NN = [N0]

zz = [N0]

def m(t, N\_y):

s = k\*N\_y\*\*2

return s

i = 0

while i < n-1:

m1 = m(t1, Nd)

Nd = Nd + h\*m1

t1 = t1 + h

m2 = m(t1, Nd)

N1 = N1 + h\*(m1 + m2)/2

tt.append(t1)

NN.append(N1)

z = 1/(0.01-k\*t1)

zz.append(z)

i = i+1

df = pd.DataFrame({'Time(s)': tt, 'N\_2': NN})

df.to\_excel(r'C:\Users\Anik Mandal\Desktop\val\_8\_8\_4.xlsx')

print(df)

plt.plot(tt, NN, 'r')

plt.plot(tt, zz, '--c')

plt.xlabel('Time(s)')

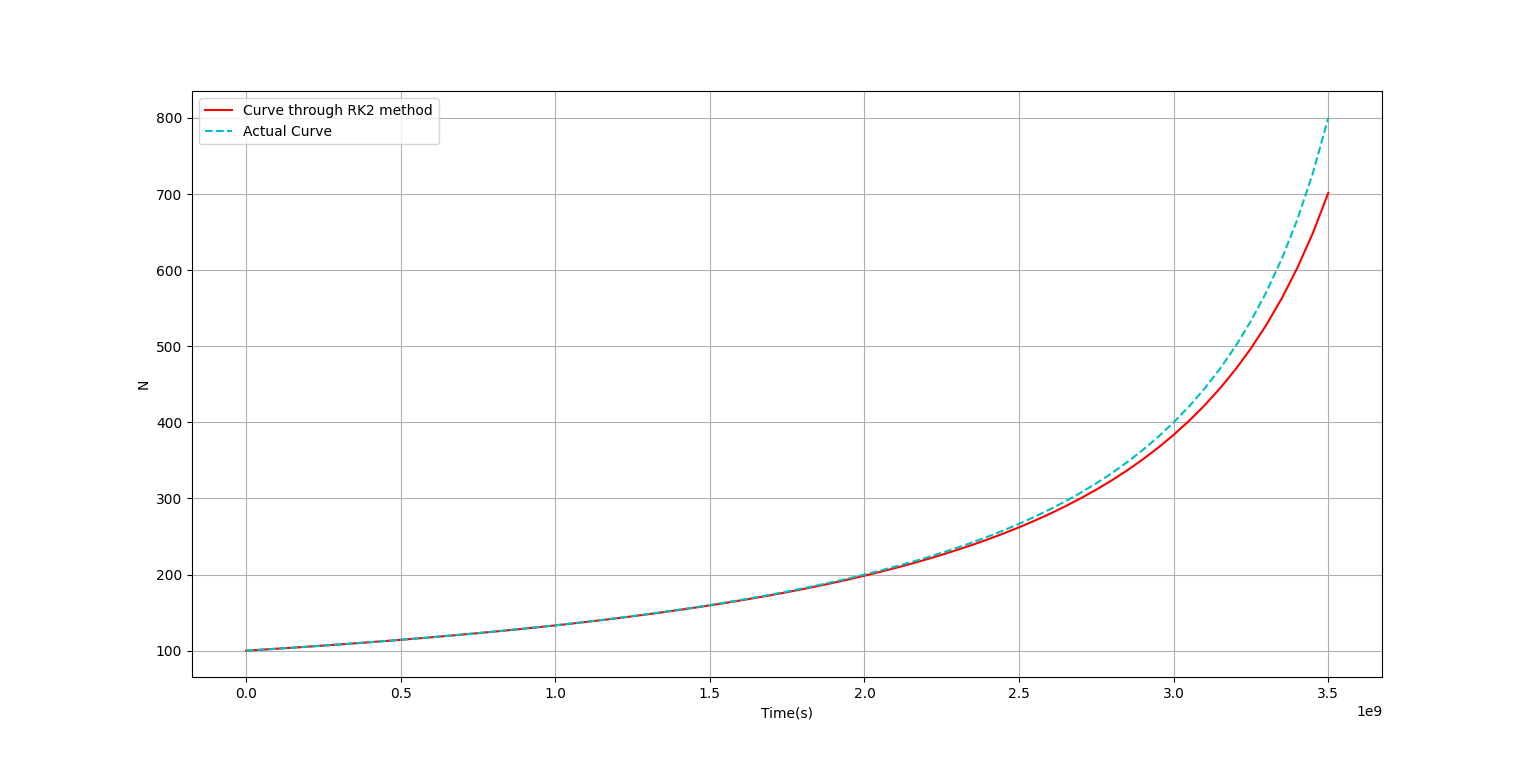
plt.ylabel('N')

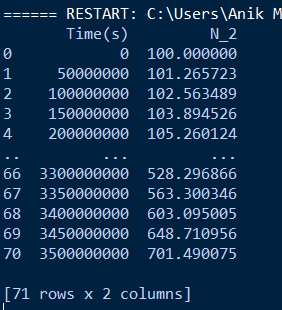
plt.legend(['Curve through RK2 method', 'Actual Curve'])

plt.grid()

plt.show()

Output:





1. Initial Value problem: solve 2nd order ODE using RK-4 method.

Program: