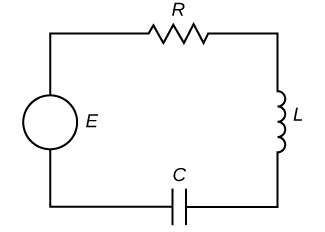
***Table Of Content***

* The RLC series circuit
* Values of constant and boundary conditions
* Analytical solution
* Numerical solutions
  1. Euler Method
  2. Runge Kutta 2
  3. Runge Kutta 4
  4. ODEINT()
* Comparison
  1. Value of function
  2. Error

***The RLC Series Circuit***

Consider an electrical circuit containing a resistor, an inductor, and a capacitor Such a circuit is called an ***RLC* series circuit**. *RLC* circuits are used in many electronic systems, most notably as tuners in AM/FM radios. Such circuits can be modeled by second-order, constant-coefficient differential equations.



Let I(t) denote the current in the *RLC* circuit and q(t) denote the charge on the capacitor. Furthermore, let L denote inductance in henrys (H), R denote resistance in ohms (Ω), and C denote capacitance in farads (F). Last, let E(t) denote electric potential in volts (V).

Kirchhoff’s voltage rule states that the sum of the voltage drops around any closed loop must be zero. So, We have

EL+ER+EC=E(t)

And we know that

Next, according to Ohm’s law,

ER =RI.

Last, the voltage drop across a capacitor is proportional to the charge, q, on the capacitor, with proportionality constant 1/C. Thus,

Adding these terms together, we get

Taking derivative and we know,I= (dq)/(dt), then

---(1)

***Boundary conditions and value of constant***

**Find the current in an RLC series circuit where L=10H, R=30Ω, C=0.02F, and E(t)=50sin(2t). Assume the initial charge on the capacitor is 0 C and the initial current is 0 A.**

***Analytical solution***

E’(t)=100cos(2t), Putting all the values in equation (1)

Now, solve this:

Roots are imaginary, so **complementary equation** is:

**The particular solution** is:

As

So, the particular equation will be

Hence

I(0)=0 so,

As I’(0)=0 so,

**So, the solution is**

***Numerical solutions***

First I write the true solution as function named ‘**func()’**

import math

def func(t):

  return math.exp(-3\*t/2)\*((-10/37)\*math.cos(math.sqrt(11)\*t/2) +(-270/(math.sqrt(11)\*37))\*math.sin(math.sqrt(11)\*t/2)) + (10/37)\*(math.cos(2\*t) +6\*math.sin(2\*t)) # represents the equation by analytical sol.

t = np.linspace(0, 10, 101) #t are the points

Tsol=[] #consist the true solution values on range t

for i in t:

    Tsol.append(func(i))

***Numerical method 1***

*I first solve using* ***Euler method (Rk1)***

import numpy as np

import matplotlib.pyplot as plt

def pend(I, t, b, c):

    return np.array([I[1], 10\*math.cos(2\*t) -b\*I[1] - c\*I[0]])

t = np.linspace(0, 10, 101) #t are the points

b = 3

c = 5

y0 = np.array([0,0])

def rungekutta1(f, y0, t, args=()):

    n = len(t)

    y = np.zeros((n, len(y0)))

    y[0] = y0

    for i in range(n - 1):

        y[i+1] = y[i] + (t[i+1] - t[i]) \* f(y[i], t[i], \*args)

    return y

solrk1 = rungekutta1(pend, y0, t, args=(b, c))

plt.plot(t, Tsol, 'r', label='TRUE(t)')

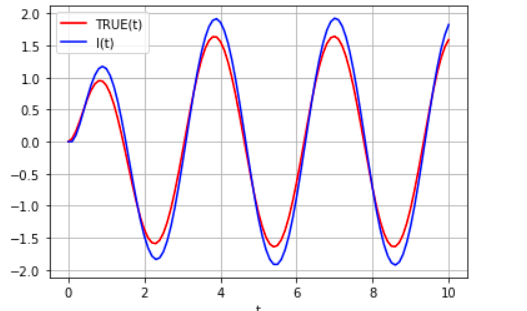
plt.plot(t, solrk1[:, 0], 'b', label='I(t)')

plt.legend(loc='best')

plt.xlabel('t')

plt.grid()

plt.show()



*Here exact solution and Euler solution are shown*

**Error:**

err=[]

for i in range(len(t)):

  err.append(Tsol[i]-solrk1[:,0][i])

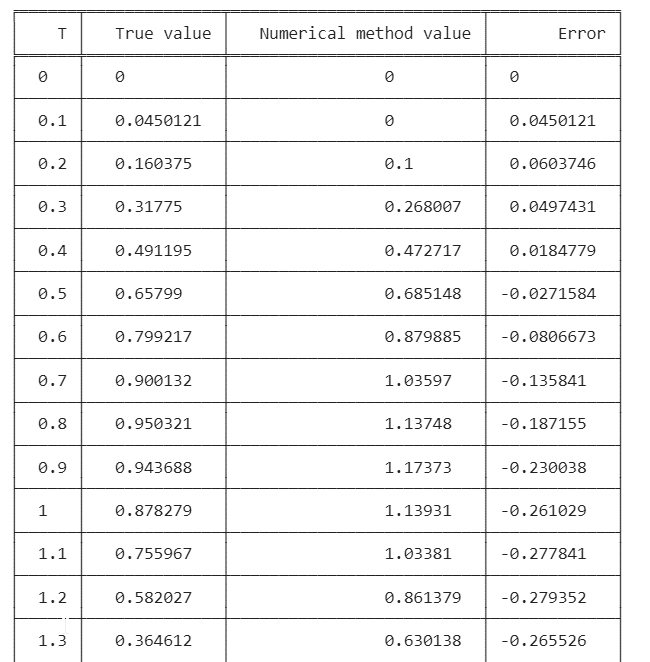
from tabulate import tabulate

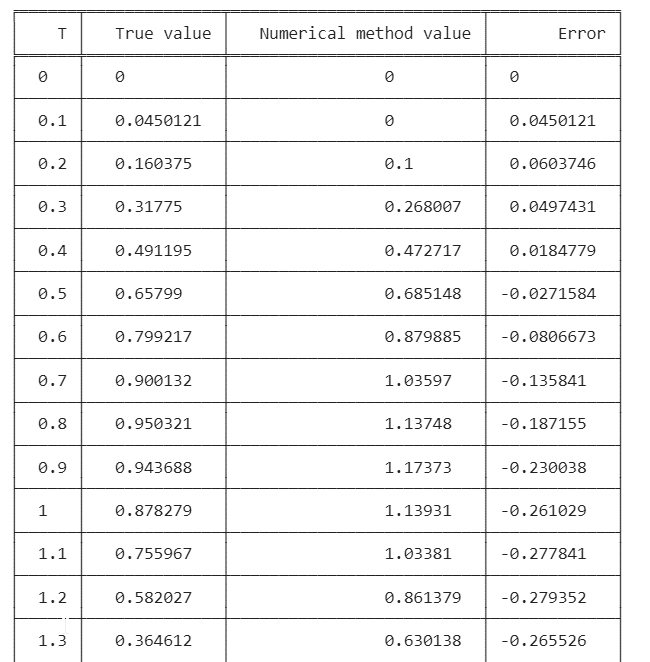
table=[['T','True value','Numerical method value','Error']]

for i in range(len(t)):

  table.append([t[i],Tsol[i],solrk1[:,0][i],err[i]])

print(tabulate(table,headers='firstrow',tablefmt='fancy\_grid'))



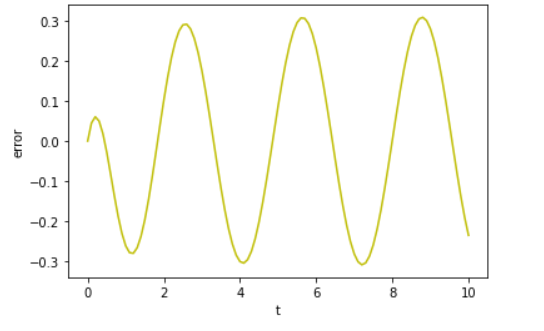


**…**

plt.plot(t, err, 'b', label='err')

plt.xlabel('t')

plt.ylabel('error')



*Error plot-it range from [-0.3,+0.3]*

SumOfSquare=0

absoluteSum=0

for i in range(len(t)):

  SumOfSquare=SumOfSquare=+ err[i]\*\*2

  absoluteSum=absoluteSum + abs(err[i])

#root mean square error

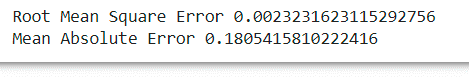
RMSE=math.sqrt(SumOfSquare)/len(t)

#Mean absolute Error

MAE=absoluteSum/len(t)

print("Root Mean Square Error",RMSE)

print("Mean Absolute Error",MAE)



***Numerical method 2***

*I have solved it with* ***runge kutta 2 method***

import numpy as np

import matplotlib.pyplot as plt

def pend(I, t, b, c):

    return np.array([I[1], 10\*math.cos(2\*t) -b\*I[1] - c\*I[0]])

t = np.linspace(0, 10, 101) #t are the points

b = 3

c = 5

y0 = np.array([0,0])

def rungekutta2(f, y0, t, args=()):

    n = len(t)

    y = np.zeros((n, len(y0)))

    y[0] = y0

    for i in range(n - 1):

        h = t[i+1] - t[i]

        y[i+1] = y[i] + h \* f(y[i] + f(y[i], t[i], \*args) \* h / 2., t[i] + h / 2., \*args)

    return y

solrk2 = rungekutta2(pend, y0, t, args=(b, c))

plt.plot(t, Tsol, 'r', label='TRUE(t)')

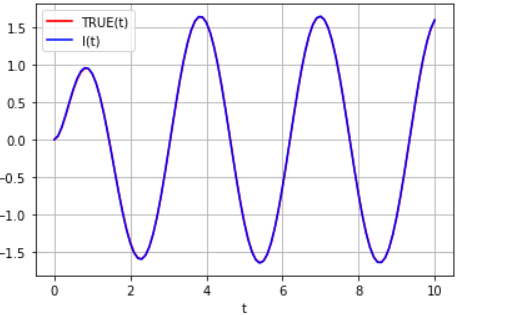
plt.plot(t, solrk2[:, 0], 'b', label='I(t)')

plt.legend(loc='best')

plt.xlabel('t')

plt.grid()

plt.show()



*Here true solution and Rk2 solution are shown-* ***good overlap***

**Error:**

err=[]

for i in range(len(t)):

  err.append(Tsol[i]-solrk2[:,0][i])

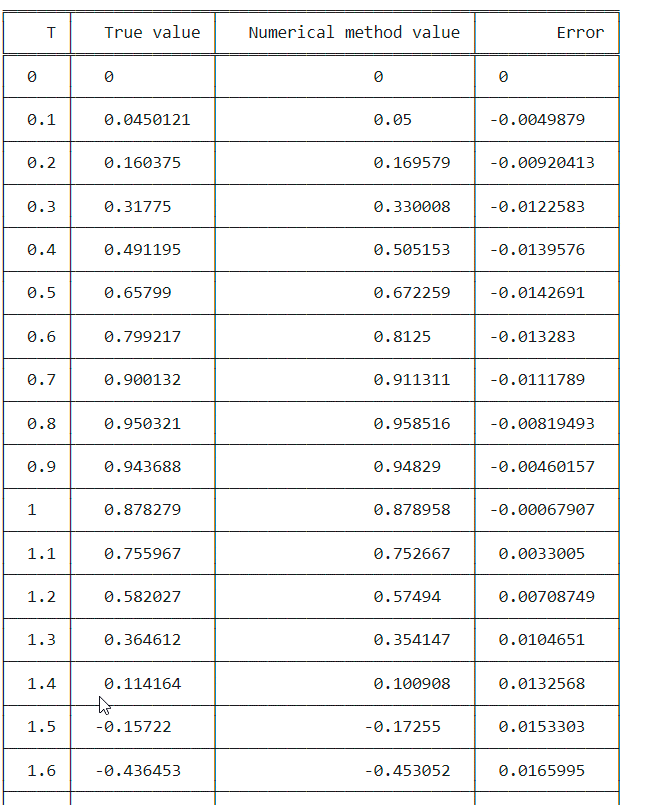
from tabulate import tabulate

table=[['T','True value','Numerical method value','Error']]

for i in range(len(t)):

  table.append([t[i],Tsol[i],solrk2[:,0][i],err[i]])

print(tabulate(table,headers='firstrow',tablefmt='fancy\_grid'))

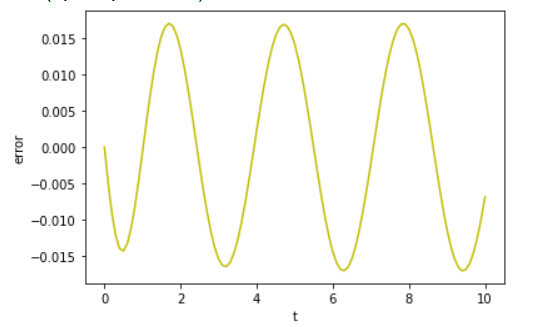


**…**

plt.plot(t, err, 'b', label='err')

plt.xlabel('t')

plt.ylabel('error')



*Error plot-it range from [-0.015,+0.015]*

SumOfSquare=0

absoluteSum=0

for i in range(len(t)):

  SumOfSquare=SumOfSquare=+ err[i]\*\*2

  absoluteSum=absoluteSum + abs(err[i])

#root mean square error

RMSE=math.sqrt(SumOfSquare)/len(t)

#Mean absolute Error

MAE=absoluteSum/len(t)

print("Root Mean Square Error = ",RMSE)

print("Mean Absolute Error = ",MAE)



***Numerical method 3***

*Here I have solve the problem using* ***runge kutta 4***

import numpy as np

import matplotlib.pyplot as plt

def pend(I, t, b, c):

    return np.array([I[1], 10\*math.cos(2\*t) -b\*I[1] - c\*I[0]])

t = np.linspace(0, 10, 101) #t are the points

b = 3

c = 5

y0 = np.array([0,0])

def rungekutta4(f, y0, t, args=()):

    n = len(t)

    y = np.zeros((n, len(y0)))

    y[0] = y0

    for i in range(n - 1):

        h = t[i+1] - t[i]

        k1 = f(y[i], t[i], \*args)

        k2 = f(y[i] + k1 \* h / 2., t[i] + h / 2., \*args)

        k3 = f(y[i] + k2 \* h / 2., t[i] + h / 2., \*args)

        k4 = f(y[i] + k3 \* h, t[i] + h, \*args)

        y[i+1] = y[i] + (h / 6.) \* (k1 + 2\*k2 + 2\*k3 + k4)

    return y

solrk4 = rungekutta4(pend, y0, t, args=(b, c))

plt.plot(t, Tsol, 'r', label='TRUE(t)')

plt.plot(t, solrk4[:, 0], 'b', label='Q(t)')

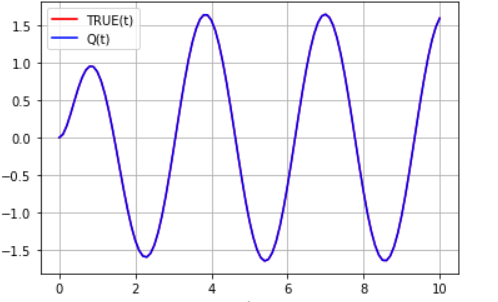
#plt.plot(t, solrk4[:, 1], 'g', label='I(t)')

plt.legend(loc='best')

plt.xlabel('t')

plt.grid()

plt.show()



*Here true solution and RK4 solution are shown- very good overlap*

**Error:**

err=[]

for i in range(len(t)):

  err.append(Tsol[i]-solrk4[:,0][i])

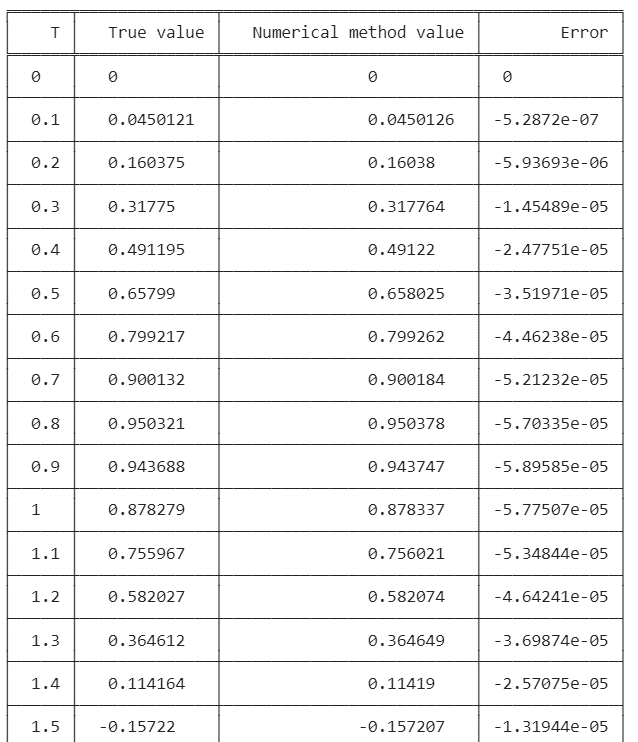
from tabulate import tabulate

table=[['T','True value','Numerical method value','Error']]

for i in range(len(t)):

  table.append([t[i],Tsol[i],solrk4[:,0][i],err[i]])

print(tabulate(table,headers='firstrow',tablefmt='fancy\_grid'))

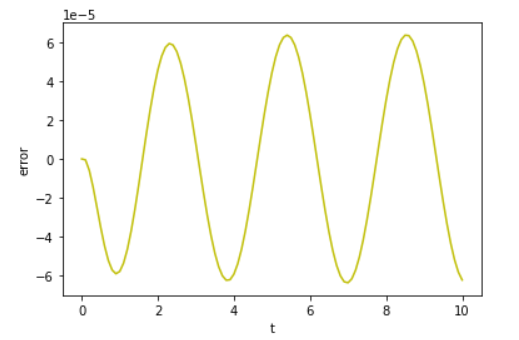


**…**

plt.plot(t, err, 'b', label='err')

plt.xlabel('t')

plt.ylabel('error')



*Error plot-it range from [-6\*e-5, 6 \*e-5]*

SumOfSquare=0

absoluteSum=0

for i in range(len(t)):

  SumOfSquare=SumOfSquare=+ err[i]\*\*2

  absoluteSum=absoluteSum + abs(err[i])

#root mean square error

RMSE=math.sqrt(SumOfSquare)/len(t)

#Mean absolute Error

MAE=absoluteSum/len(t)

print("Root Mean Square Error = ",RMSE)

print("Mean Absolute Error = ",MAE)



***Numerical method 4***

*Here i have used* ***ODEINT() method*** *provided by scipy.integrate*

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import odeint

def pend(I, t, b, c):

    return np.array([I[1], 10\*math.cos(2\*t) -b\*I[1] - c\*I[0]])

t = np.linspace(0, 10, 101) #t are the points

b = 3

c = 5

y0 = np.array([0,0])

sol = odeint(pend, y0, t, args=(b, c))

plt.plot(t, sol[:, 0], 'b', label='I(t)')

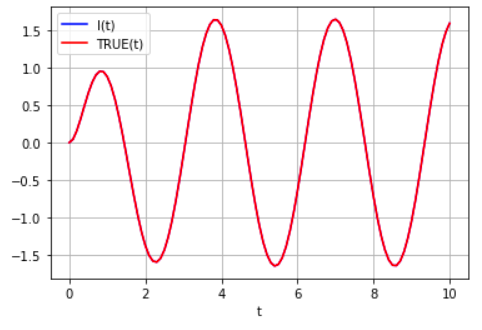
plt.plot(t, Tsol, 'r', label='TRUE(t)')

plt.legend(loc='best')

plt.xlabel('t')

plt.grid()

plt.show()



*Here true solution and odeint() solution are shown- perfect overlap*

**Error:**

err=[]

for i in range(len(t)):

  err.append(Tsol[i]-sol[:,0][i])

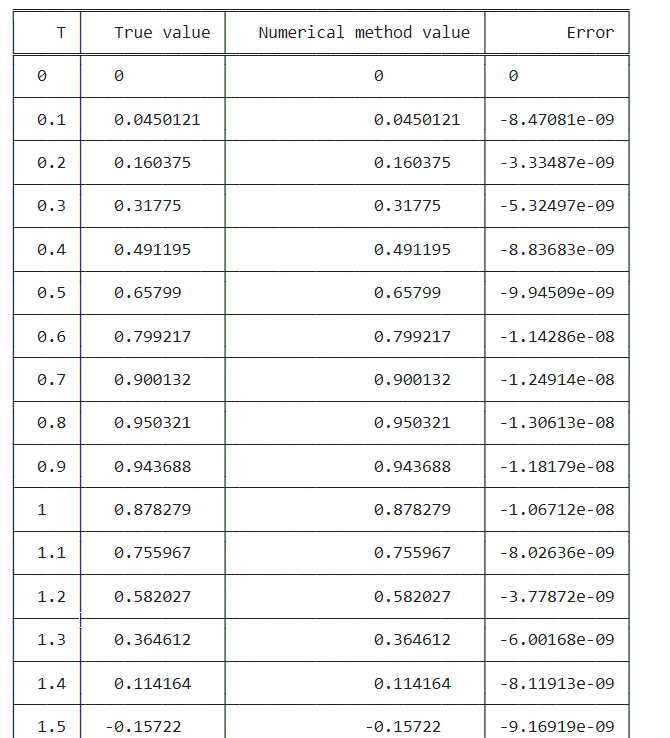
from tabulate import tabulate

table=[['T','True value','Numerical method value','Error']]

for i in range(len(t)):

  table.append([t[i],Tsol[i],sol[:,0][i],err[i]])

print(tabulate(table,headers='firstrow',tablefmt='fancy\_grid'))

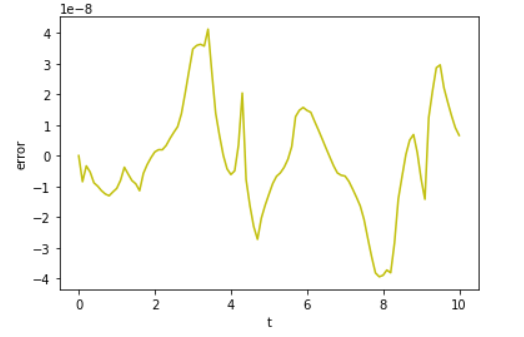


**…**

plt.plot(t, err, 'b', label='Q(t)')

plt.xlabel('t')

plt.ylabel('error')



*Error plot-it range from [-4\*e-8, 4 \*e-8]*

SumOfSquare=0

absoluteSum=0

for i in range(len(t)):

  SumOfSquare=SumOfSquare=+ err[i]\*\*2

  absoluteSum=absoluteSum + abs(err[i])

#root mean square error

RMSE=math.sqrt(SumOfSquare)/len(t)

#Mean absolute Error

MAE=absoluteSum/len(t)

print("Root Mean Square Error = ",RMSE)

print("Mean Absolute Error = ",MAE)



***Comparison***

*Value of function*

***Table***

from tabulate import tabulate

table=[['T','True value','RK1/Euler','RK2','RK4','ODEINT()']]

for i in range(0,101,10):

  table.append([t[i],Tsol[i],solrk1[:,0][i],solrk2[:,0][i],solrk4[:,0][i],sol[:,0][i]])

print(tabulate(table,headers='firstrow',tablefmt='fancy\_grid'))

╒═════╤══════════════╤═════════════╤═══════════╤═══════════╤════════════╕

│ T │ True value │ RK1/Euler │ RK2 │ RK4 │ ODEINT() │

╞═════╪══════════════╪═════════════╪═══════════╪═══════════╪════════════╡

│ 0 │ 0 │ 0 │ 0 │ 0 │ 0 │

├─────┼──────────────┼─────────────┼───────────┼───────────┼────────────┤

│ 1 │ 0.878279 │ 1.13931 │ 0.878958 │ 0.878337 │ 0.878279 │

├─────┼──────────────┼─────────────┼───────────┼───────────┼────────────┤

│ 2 │ -1.37158 │ -1.47345 │ -1.38505 │ -1.37163 │ -1.37158 │

├─────┼──────────────┼─────────────┼───────────┼───────────┼────────────┤

│ 3 │ -0.170775 │ -0.345919 │ -0.15539 │ -0.170783 │ -0.170775 │

├─────┼──────────────┼─────────────┼───────────┼───────────┼────────────┤

│ 4 │ 1.56254 │ 1.86254 │ 1.56028 │ 1.5626 │ 1.56254 │

├─────┼──────────────┼─────────────┼───────────┼───────────┼────────────┤

│ 5 │ -1.11001 │ -1.19661 │ -1.12417 │ -1.11006 │ -1.11001 │

├─────┼──────────────┼─────────────┼───────────┼───────────┼────────────┤

│ 6 │ -0.641885 │ -0.873565 │ -0.627487 │ -0.641908 │ -0.641885 │

├─────┼──────────────┼─────────────┼───────────┼───────────┼────────────┤

│ 7 │ 1.64339 │ 1.9244 │ 1.6456 │ 1.64346 │ 1.64339 │

├─────┼──────────────┼─────────────┼───────────┼───────────┼────────────┤

│ 8 │ -0.725707 │ -0.727825 │ -0.741973 │ -0.725737 │ -0.725707 │

├─────┼──────────────┼─────────────┼───────────┼───────────┼────────────┤

│ 9 │ -1.03936 │ -1.31872 │ -1.02803 │ -1.03939 │ -1.03936 │

├─────┼──────────────┼─────────────┼───────────┼───────────┼────────────┤

│ 10 │ 1.59074 │ 1.82538 │ 1.59759 │ 1.59081 │ 1.59074 │

╘═════╧══════════════╧═════════════╧═══════════╧═══════════╧════════════╛

***Graph***

plt.plot(t, Tsol, 'r', label='TRUE(t)')

plt.plot(t, solrk1[:, 0], 'b', label='RK1')

plt.plot(t, solrk2[:, 0], 'G', label='RK2')

plt.plot(t, solrk4[:, 0], 'Y', label='RK4')

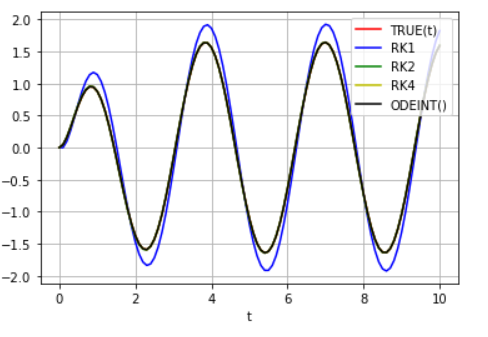
plt.plot(t, sol[:, 0], 'K', label='ODEINT()')

plt.legend(loc='best')

plt.xlabel('t')

plt.grid()

plt.show()



*True,RK2,RK4,ODEINT() are overlap and has same as black graph.*

*Error*

***Table***

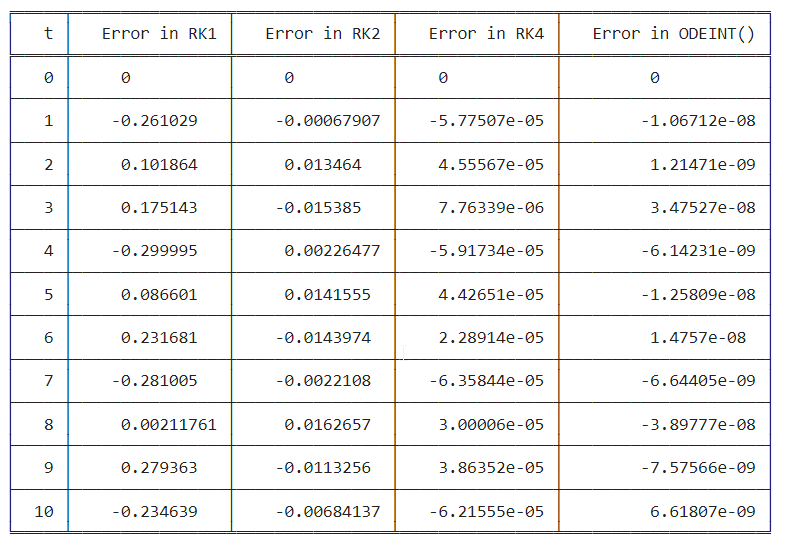
from tabulate import tabulate

table=[['t','Error in RK1','Error in RK2','Error in RK4','Error in ODEINT()']]

for i in range(0,101,10):

  table.append([t[i],e1[i],e2[i],e3[i],e4[i]])

print(tabulate(table,headers='firstrow',tablefmt='fancy\_grid'))



plt.plot(t, e1, 'b', label='RK1')

plt.plot(t, e2, 'G', label='RK2')

plt.plot(t, e3, 'Y', label='RK4')

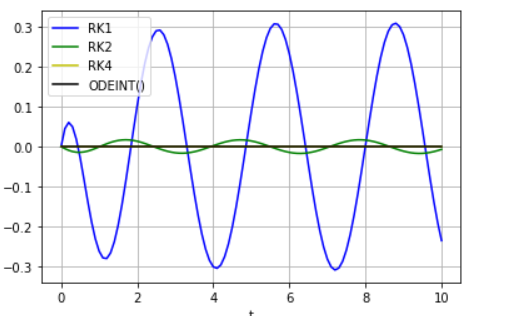
plt.plot(t, e4, 'K', label='ODEINT()')

plt.legend(loc='best')

plt.xlabel('t')

plt.grid()

plt.show()



*RK4,ODEINT() are overlapped and has same error as black graph.*

*\*\*\*The end\*\*\**