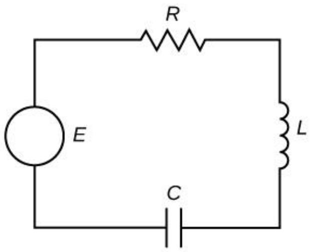
Table Of Content

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

$$E_L+E_R+E_C=E(t)$$

And we know that

$$E_L = \mathrm{L}\,\frac{dI}{dt}$$

Next, according to Ohm's law,

$$E_R = RI.$$

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

$$E_c = \frac{1}{C}q$$

Adding these terms together, we get

$$\mathrm{L}\,\frac{dI}{dt} + RI + \frac{1}{C}q = E(t)$$

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

$$L\,\frac{d^2I}{dt^2} + R\frac{dI}{dt} + \frac{1}{C}I = E'(t) \qquad \text{---(1)}$$

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)

$$10\frac{d^2I}{dt^2} + 30\frac{dI}{dt} + \frac{1}{0.02}I = 100\cos(2t)$$

$$10\frac{d^2I}{dt^2} + 30\frac{dI}{dt} + 50I = 100\cos(2t)$$

Now, solve this:

$$10r^2 + 30r + 50 = 0$$

$$r^2 + 3r + 5 = 0$$

$$(r^2 + 3r + \frac{9}{4}) - \frac{9}{4} + 5 = 0$$

$$(r^2 + \frac{3}{2})^2 + \frac{11}{4} = 0$$

$$r_1 = -\frac{3}{2} + \frac{\sqrt{11}}{2}\,, r_2 = -\frac{3}{2} - \frac{\sqrt{11}}{2}$$

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

$$I_c = e^{at}(c_1\cos(bt) + \ c_2\sin(bt))$$

$$I_c = e^{-\frac{3}{2}t}(c_1\cos\left(\frac{\sqrt{11}}{2}t\right) + \ c_2\sin\left(\frac{\sqrt{11}}{2}t\right))$$

The particular solution is:

$$I_p = A\cos(2t) + \ B\sin(2t)$$

As $E(t)$ =50cos$(2t)$, *then we have*

$$\omega = 2 \ and \ E_0 = 50$$

$$S = \omega L - \frac{1}{\omega C} = 2(10) - \frac{1}{2(0.02)} = -5$$

$$A = -\frac{E_0 S}{R^2 + S^2} = -\frac{50 * (-5)}{30^2 + (-5)^2} = \frac{10}{37}$$

$$A = \frac{E_0 R}{R^2 + S^2} = \frac{50 * (30)}{30^2 + (-5)^2} = \frac{60}{37}$$

So, the particular equation will be

$$I_p = \frac{10}{37}\cos(2t) + \frac{60}{37}\sin(2t)$$

Hence

$$q(t) = q_c + q_p$$

$$I(t) = e^{-\frac{3}{2}t}(c_1 \cos\left(\frac{\sqrt{11}}{2}t\right) + \ c_2 \sin\left(\frac{\sqrt{11}}{2}t\right)) + \frac{10}{37}\cos(2t) + \frac{60}{37}\sin(2t)$$

I(0)=0 so,

$$q(0) = e^0(c_1(1) + \ 0) + \frac{10}{37}(1) + 0$$

$$0 = c_1 + \frac{10}{37} => \ c_1 = -\frac{10}{37}$$

As I'(0)=0 so,

$$q'^{(0)} = -\frac{3}{2}c_1 + \ \frac{\sqrt{11}}{2}c_2 + \frac{120}{37}$$

$$c_2 = -\frac{270}{37\sqrt{11}}$$

So, the solution is

$$\boldsymbol{q(t) = e^{-\frac{3}{2}t}(-\frac{10}{37}\cos\left(\frac{\sqrt{11}}{2}t\right) \ - \frac{270}{37\sqrt{11}}\sin\left(\frac{\sqrt{11}}{2}t\right)) + \frac{10}{37}\cos(2t) + \frac{60}{37}\sin(2t)}$$

Numerical solutions

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

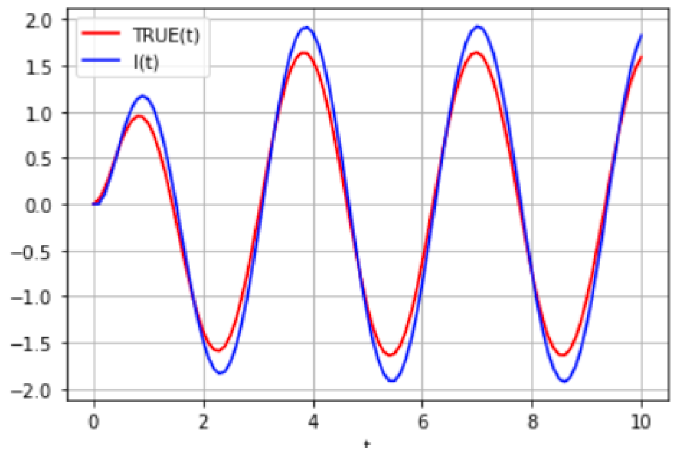
```
import math

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:

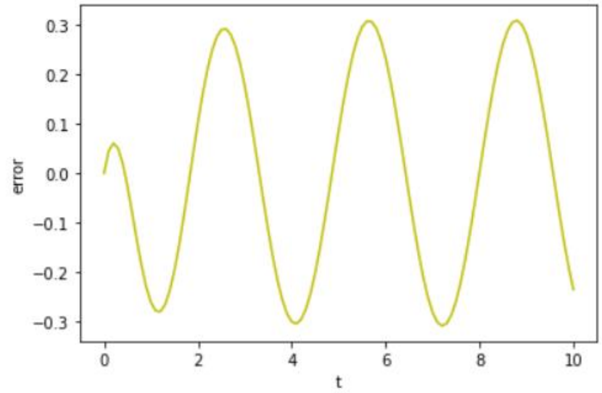
```
err=[]
for i in range(len(t)):
  err.append(Tsol[i]-solrk1[:,0][i])

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'))
```

T	True value	Numerical method value	Error
0	0	0	0
0.1	0.0450121	0	0.0450121
0.2	0.160375	0.1	0.0603746
0.3	0.31775	0.268007	0.0497431
0.4	0.491195	0.472717	0.0184779
0.5	0.65799	0.685148	-0.0271584
0.6	0.799217	0.879885	-0.0806673

0.7	0.900132	1.03597	-0.135841
0.8	0.950321	1.13748	-0.187155
0.9	0.943688	1.17373	-0.230038
1	0.878279	1.13931	-0.261029
1.1	0.755967	1.03381	-0.277841
1.2	0.582027	0.861379	-0.279352
1.3	0.364612	0.630138	-0.265526

...

```
plt.plot(t, err, 'b', label='err')
plt.xlabel('t')
plt.ylabel('error')
```



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

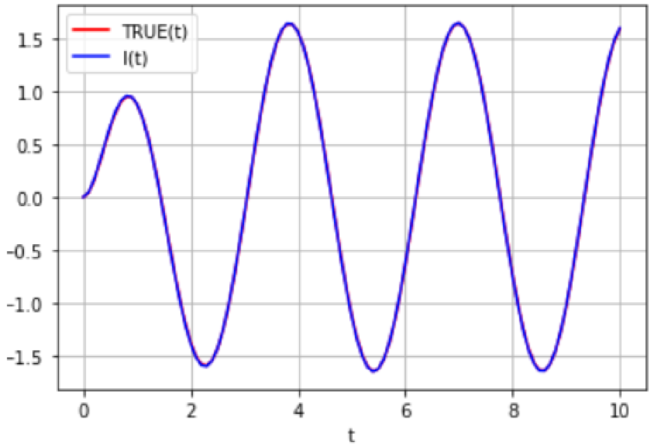
```
SumOfSquare=0
absoluteSum=0

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)
```

```
Root Mean Square Error 0.0023231623115292756
Mean Absolute Error 0.1805415810222416
```

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***

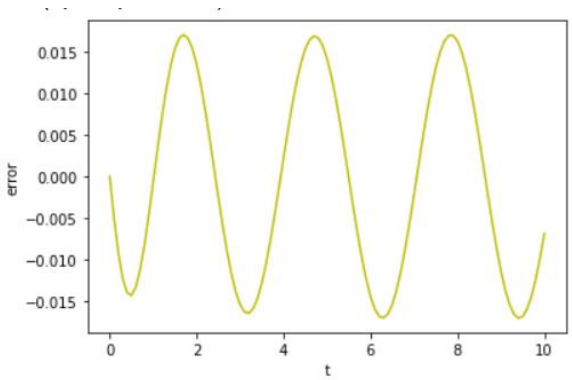
Error:

```
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'))
```

T	True value	Numerical method value	Error
0	0	0	0
0.1	0.0450121	0.05	-0.0049879
0.2	0.160375	0.169579	-0.00920413
0.3	0.31775	0.330008	-0.0122583
0.4	0.491195	0.505153	-0.0139576
0.5	0.65799	0.672259	-0.0142691
0.6	0.799217	0.8125	-0.013283
0.7	0.900132	0.911311	-0.0111789
0.8	0.950321	0.958516	-0.00819493
0.9	0.943688	0.94829	-0.00460157
1	0.878279	0.878958	-0.00067907
1.1	0.755967	0.752667	0.0033005
1.2	0.582027	0.57494	0.00708749
1.3	0.364612	0.354147	0.0104651

…

```
plt.plot(t, err, 'b', label='err')
plt.xlabel('t')
plt.ylabel('error')
```



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

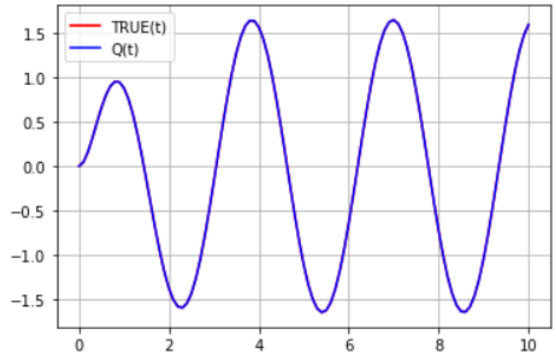
```python
SumOfSquare=0
absoluteSum=0
for i in range(len(t)):
  SumOfSquare=SumOfSquare=+ err[i]**2
  absoluteSum=absoluteSum + abs(err[i])


#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)
```

```
Root Mean Square Error =  6.773633528449245e-05
Mean Absolute Error =  0.010669684938809777
```

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:

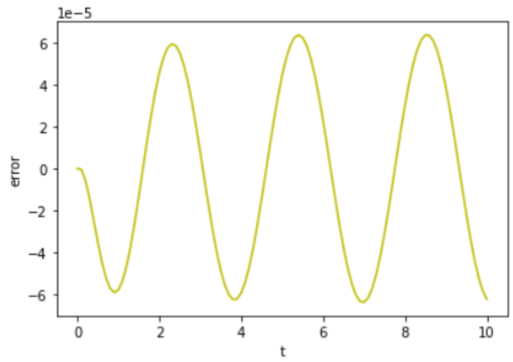
```
err=[]
for i in range(len(t)):
  err.append(Tsol[i]-solrk4[:,0][i])

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'))
```

T	True value	Numerical method value	Error
0	0	0	0
0.1	0.0450121	0.0450126	-5.2872e-07
0.2	0.160375	0.16038	-5.93693e-06
0.3	0.31775	0.317764	-1.45489e-05
0.4	0.491195	0.49122	-2.47751e-05
0.5	0.65799	0.658025	-3.51971e-05
0.6	0.799217	0.799262	-4.46238e-05
0.7	0.900132	0.900184	-5.21232e-05
0.8	0.950321	0.950378	-5.70335e-05
0.9	0.943688	0.943747	-5.89585e-05
1	0.878279	0.878337	-5.77507e-05
1.1	0.755967	0.756021	-5.34844e-05
1.2	0.582027	0.582074	-4.64241e-05
1.3	0.364612	0.364649	-3.69874e-05
1.4	0.114164	0.11419	-2.57075e-05
1.5	-0.15722	-0.157207	-1.31944e-05

…

```
plt.plot(t, err, 'b', label='err')
plt.xlabel('t')
```

```
plt.ylabel('error')
```



Error plot-it range from $[-6*e^{-5}, 6 *e^{-5}]$

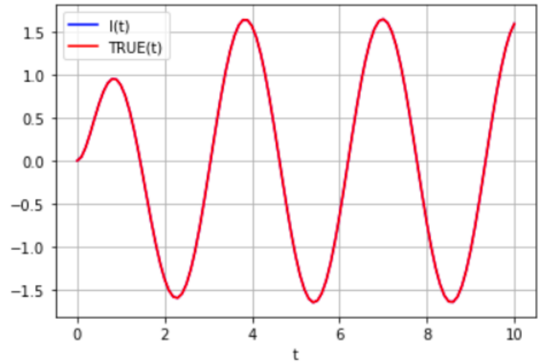
```
SumOfSquare=0
absoluteSum=0
for i in range(len(t)):
  SumOfSquare=SumOfSquare=+ err[i]**2
  absoluteSum=absoluteSum + abs(err[i])


#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)
```

```
Root Mean Square Error =  6.15400560774608e-07
Mean Absolute Error =  3.8478353957005165e-05
```

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:

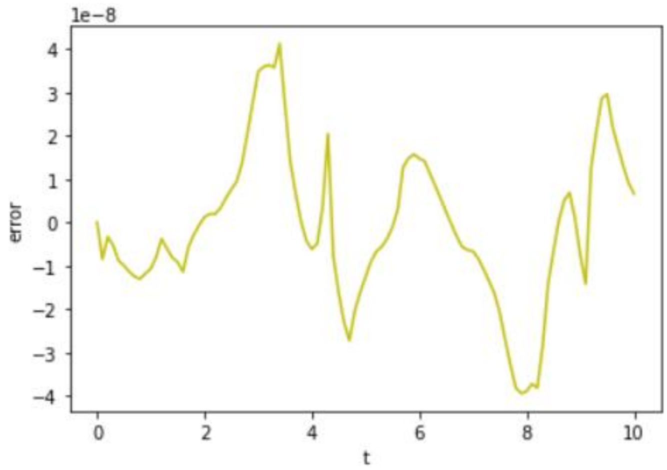
```
err=[]
for i in range(len(t)):
  err.append(Tsol[i]-sol[:,0][i])
```

```python
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'))
```

T	True value	Numerical method value	Error
0	0	0	0
0.1	0.0450121	0.0450121	-8.47081e-09
0.2	0.160375	0.160375	-3.33487e-09
0.3	0.31775	0.31775	-5.32497e-09
0.4	0.491195	0.491195	-8.83683e-09
0.5	0.65799	0.65799	-9.94509e-09
0.6	0.799217	0.799217	-1.14286e-08
0.7	0.900132	0.900132	-1.24914e-08
0.8	0.950321	0.950321	-1.30613e-08
0.9	0.943688	0.943688	-1.18179e-08
1	0.878279	0.878279	-1.06712e-08
1.1	0.755967	0.755967	-8.02636e-09
1.2	0.582027	0.582027	-3.77872e-09
1.3	0.364612	0.364612	-6.00168e-09
1.4	0.114164	0.114164	-8.11913e-09
1.5	-0.15722	-0.15722	-9.16919e-09

...

```python
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)
```

```
Root Mean Square Error =  6.552547503490991e-11
Mean Absolute Error =  1.3431390947033232e-08
```

Comparison

Value of function

Table

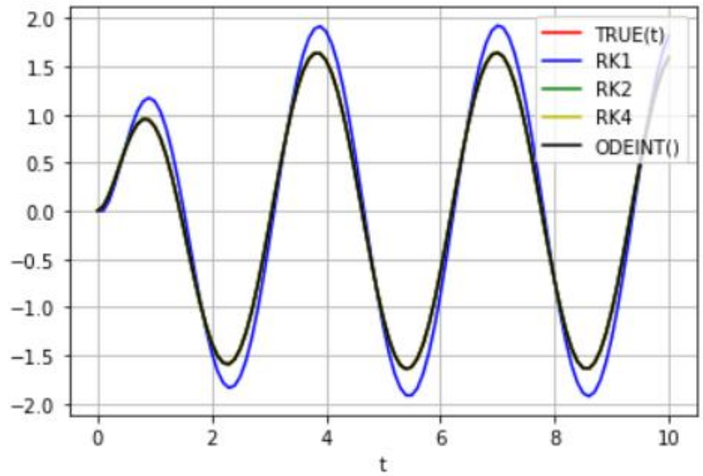
```
from tabulate import tabulate
table=[['T','True value','RK1/Euler','RK2','RK4','ODEINT()']]
for i in range(0,101,10):
  table.append([t[i],Tsol[i],solrk1[:,0][i],solrk2[:,0][i],solrk4[:,0][i],
sol[:,0][i]])
print(tabulate(table,headers='firstrow',tablefmt='fancy_grid'))
```

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.

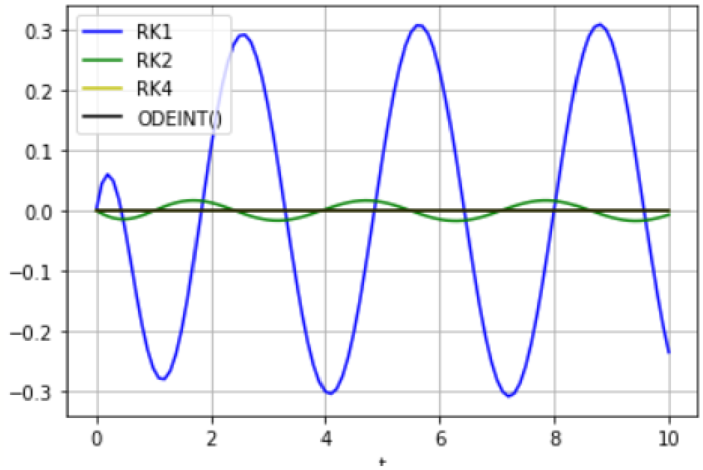
Error

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'))
```

t	Error in RK1	Error in RK2	Error in RK4	Error in ODEINT()
0	0	0	0	0
1	-0.261029	-0.00067907	-5.77507e-05	-1.06712e-08
2	0.101864	0.013464	4.55567e-05	1.21471e-09
3	0.175143	-0.015385	7.76339e-06	3.47527e-08
4	-0.299995	0.00226477	-5.91734e-05	-6.14231e-09
5	0.086601	0.0141555	4.42651e-05	-1.25809e-08
6	0.231681	-0.0143974	2.28914e-05	1.4757e-08
7	-0.281005	-0.0022108	-6.35844e-05	-6.64405e-09
8	0.00211761	0.0162657	3.00006e-05	-3.89777e-08
9	0.279363	-0.0113256	3.86352e-05	-7.57566e-09
10	-0.234639	-0.00684137	-6.21555e-05	6.61807e-09

```
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****