MP2 LAB
ASSIGNMENT - 6

Name:- Abdul Qadir

Course:- B.Sc Physics(H)

Roll No:- 245673982

Q1.

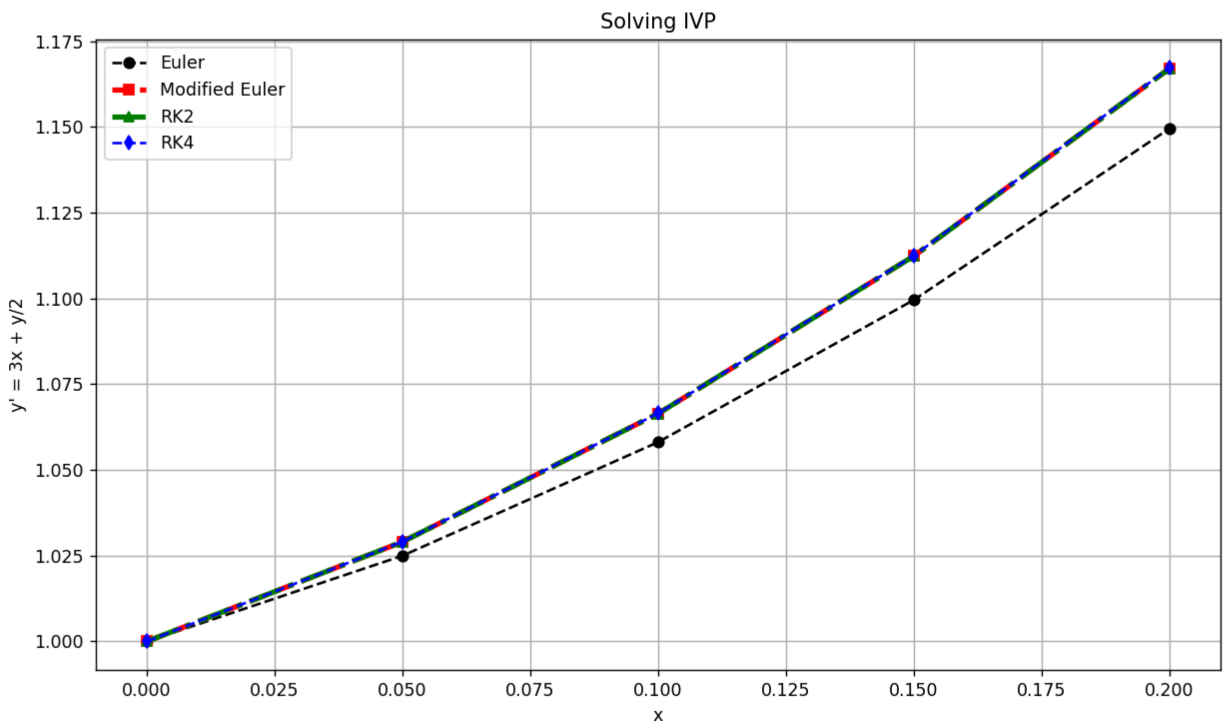
Que 1: Solve the IVP: $y'=3x + y/2$ with the initial condition $y\,(0) = 1$ in step sizes of 0.05. Evaluate y(0.2) using the Eular, Modified Eular, Second & Fourth Order Runge Kutta Methods.

compare the computed results with the analytical solution [Show all values for each iteration and for each (the Eular, Modified Eular, Second & Fourth Order Runge Kutta) method]. Comment on the accuracy of the two methods in terms of the step size.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

def f(x, y):
    return 3*x + y/2

def euler(x, y):
    return y + h*f(x,
    y)

def mod_euler(x, y, h):
    return y + (h/2)*(f(x, y) + f(x+h, y + h*f(x,y)))

def rk2(x, y, h):
    k1 = h*f(x, y)
    k2 = h*f(x+h, y+k1)
    return y + (k1+k2)/2

def rk4(x, y, h):
    k1  =  h*f(x,
    y)
    k2      =      h*f(x+h/2,
    y+k1/2)       k3       =
    h*f(x+h/2,  y+k2/2)  k4
    = h*f(x+h, y+k3)
    return y + (k1 + 2*k2 + 2*k3 + k4)/6

h = 0.05
X = [0]
eu = [1]
mod_eu = [1]
r2 = [1]
r4 = [1]

while X[-1]!=0.2:
    eu.append(euler(X[-1],
    eu[-1]))
    mod_eu.append(mod_euler(X[-1], mod_eu[-1], h))
    r2.append(rk2(X[-1], r2[-1], h))
    r4.append(rk4(X[-1], r4[-1], h))
    X.append(round(X[-1]+h, 2))
df = pd.DataFrame({'x':X, 'Euler':eu, 'Modified Euler':mod_eu, 'RK2':r2, 'RK4':r4})
```

```
print(df)

plt.figure(figsize=(10, 6))
plt.plot(X, eu, color='k', linestyle='--', marker='o', label='Euler')
plt.plot(X, mod_eu, color='r', linestyle='--', marker='s', label='Modified Euler', linewidth=3)
plt.plot(X, r2, color='g', linestyle='-.', marker='^', label='RK2', linewidth=3)
plt.plot(X, r4, color='b', linestyle='--', marker='d', label='RK4')
plt.title('Solving IVP')
plt.xlabel('x')
plt.ylabel('y\' = 3x +
y/2') plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Output:-



	x	Euler	Modified Euler	RK2	RK4
0	0.00	1.000000	1.000000	1.000000	1.000000
1	0.05	1.025000	1.029062	1.029062	1.029097
2	0.10	1.058125	1.066454	1.066454	1.066524
3	0.15	1.099578	1.112387	1.112387	1.112494
4	0.20	1.149568	1.167075	1.167075	1.167222

Q2.

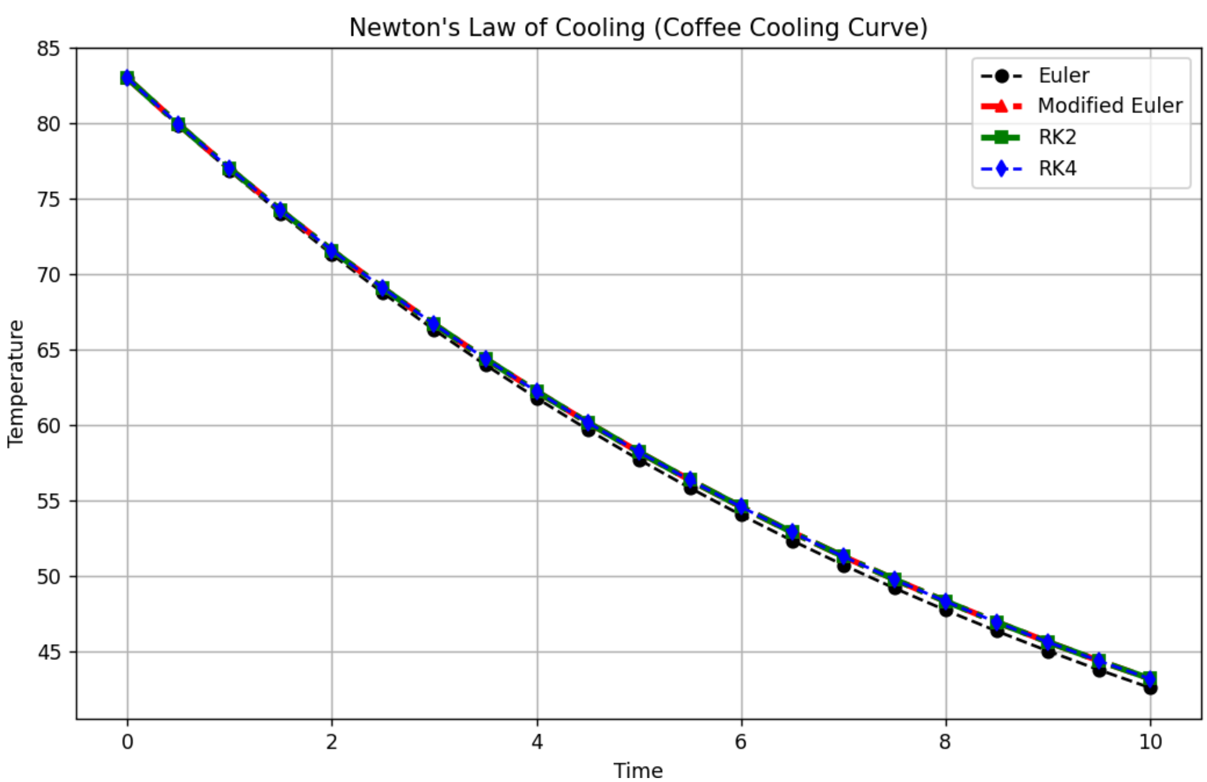
Que 2: Estimate the cooling temperature of coffee in a ceramic cup for 10s in step size of 0.5s using the Eular, Modified Eular, Second & Fourth Order Runge Kutta Methods. The initial temperature of coffee is 83.0°C. The air temperature is 20 °C. The cooling rate coefficient is $0.1s^{-1}$.

Plot the Cooling curve of Coffee and compare the computed results with the analytical solution (Show all values for each iteration and for each (the Eular, Modified Eular, Second & Fourth Order Runge Kutta) method].

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def f(T, t):
    return -k * (T - T_air)

def euler(fun, T, t, h):
    return T + h * fun(T, t)

def mod_euler(fun, T, t, h):
    k1 = fun(T, t)
    k2 = fun(T + h * k1, t + h)
    return T + (h/2) * (k1 + k2)

def rk2(fun, T, t, h):
    k1 = fun(T, t)
    k2 = fun(T + (h/2) * k1, t + h/2)
    return T + h * k2

def rk4(fun, T, t, h):
    k1 = fun(T, t)
    k2 = fun(T + (h/2) * k1, t + h/2)
    k3 = fun(T + (h/2) * k2, t + h/2)
    k4 = fun(T + h * k3, t + h)
    return T + (h/6) * (k1 + 2*k2 + 2*k3 + k4)

k = 0.1
T_air = 20
h = 0.5
t_end = 10
t = np.arange(0, t_end + h, h)

eu = [83]
mod_eu = [83]
r2 = [83]
r4 = [83]
```

```
for i in t[:-1]:
    eu.append(euler(f, eu[-1], i, h))
    mod_eu.append(mod_euler(f, mod_eu[-1], i, h))
    r2.append(rk2(f, r2[-1], i, h))
    r4.append(rk4(f, r4[-1], i, h))

df = pd.DataFrame({'Time (s)':t, 'Euler':eu, 'Modified Euler':mod_eu, 'RK2':r2, 'RK4':r4})
print(df)

plt.figure(figsize=(10,6))
plt.plot(t, eu, color='k', linestyle='--', marker='o', label='Euler')
plt.plot(t, mod_eu, color='r', linestyle='--', marker='^', label='Modified Euler', linewidth=3)
plt.plot(t, r2, color='g', linestyle='-.', marker='s', label='RK2', linewidth=3)
plt.plot(t, r4, color='b', linestyle='--', marker='d', label='RK4')
plt.xlabel('Time')
plt.ylabel('Temperature')
plt.title("Newton's Law of Cooling (Coffee Cooling Curve)")
plt.legend()
plt.grid()
plt.show()
```

Output:-



	Time (s)	Euler	Modified Euler	RK2	RK4
0	0.0	83.000000	83.000000	83.000000	83.000000
1	0.5	79.850000	79.928750	79.928750	79.927454
2	1.0	76.857500	77.007223	77.007223	77.004758
3	1.5	74.014625	74.228121	74.228121	74.224603
4	2.0	71.313894	71.584500	71.584500	71.580038
5	2.5	68.748199	69.069756	69.069756	69.064450
6	3.0	66.310789	66.677605	66.677605	66.671549
7	3.5	63.995250	64.402072	64.402072	64.395350
8	4.0	61.795487	62.237471	62.237471	62.230164
9	4.5	59.705713	60.178394	60.178394	60.170575
10	5.0	57.720427	58.219698	58.219698	58.211433
11	5.5	55.834406	56.356487	56.356487	56.347839
12	6.0	54.042686	54.584109	54.584109	54.575134
13	6.5	52.340551	52.898133	52.898133	52.888885
14	7.0	50.723524	51.294349	51.294349	51.284875
15	7.5	49.187348	49.768750	49.768750	49.759094
16	8.0	47.727980	48.317523	48.317523	48.307726
17	8.5	46.341581	46.937044	46.937044	46.927142
18	9.0	45.024502	45.623863	45.623863	45.613890
19	9.5	43.773277	44.374700	44.374700	44.364686
20	10.0	42.584613	43.186433	43.186433	43.176406

Q3.

Que 3: 1 mg of a radioactive material with half-life of 1600 years is kept for 2000years.

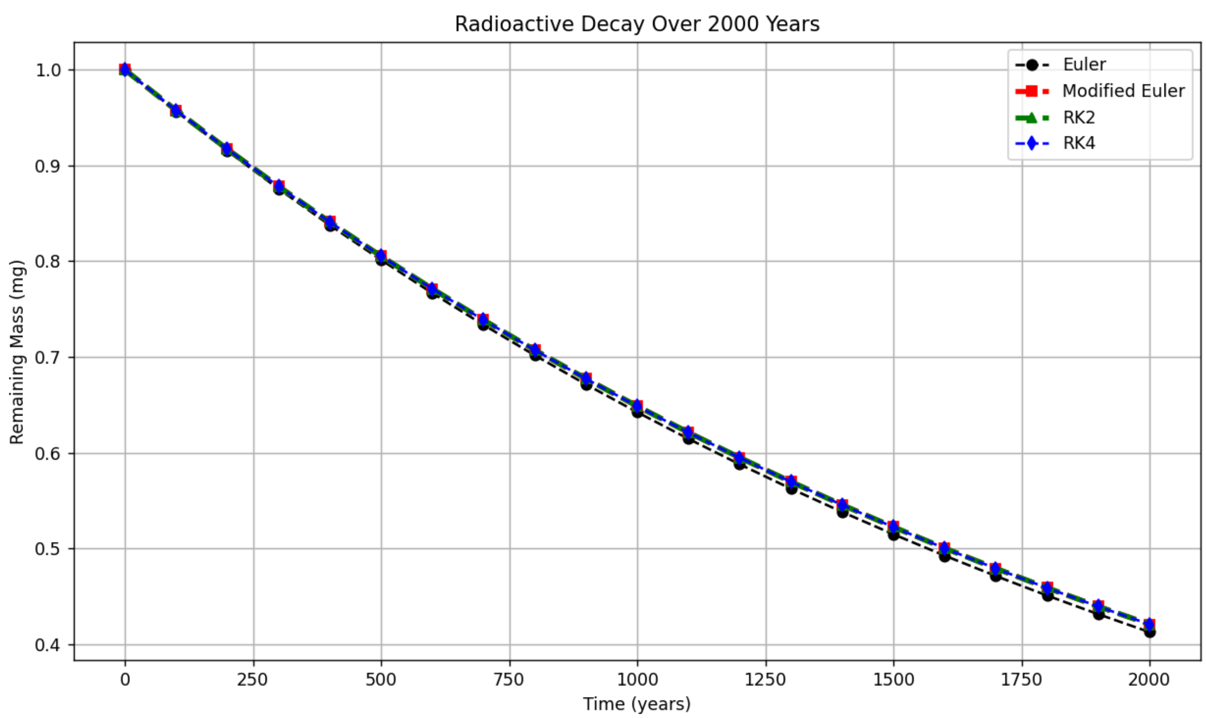
(a) Compute the mass which would have decayed by this time using **Euler Method**, **Modified Euler Method, Runge Kutta (both order) method.**

(b) Plot the decay Curve and compare the computed results with the analytical solution [Show all values for each iteration and for each (the Eular, Modified Eular, Second & Fourth Order Runge Kutta) method].

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

def f(t, y):
    return -k * y

def euler(x, y):
    return y + h*f(x,
    y)

def mod_euler(x, y, h):
    return y + (h/2)*(f(x, y) + f(x+h, y + h*f(x,y)))

def rk2(x, y, h):
    k1 = h*f(x, y)
    k2 = h*f(x+h, y+k1)
    return y + (k1+k2)/2

def rk4(x, y, h):
    k1   =   h*f(x,
    y)
    k2       =       h*f(x+h/2,
    y+k1/2)        k3        =
    h*f(x+h/2,   y+k2/2)  k4
    = h*f(x+h, y+k3)
    return y + (k1 + 2*k2 + 2*k3 + k4)/6

T_half = 1600
k = np.log(2) / T_half
h = 100

T = [0]
eu = [1]
mod_eu = [1]
r2 = [1]
r4 = [1]

while T[-1] < 2000:
    eu.append(euler(T[-1],
    eu[-1]))
```

```
    mod_eu.append(mod_euler(T[-1], mod_eu[-1],
    h)) r2.append(rk2(T[-1], r2[-1], h))
    r4.append(rk4(T[-1], r4[-1],
    h)) T.append(T[-1] + h)

df = pd.DataFrame({'Year': T, 'Euler': eu, 'Modified Euler': mod_eu, 'RK2': r2, 'RK4': r4})
print(df)

plt.figure(figsize=(10, 6))
plt.plot(T, eu, color='k', linestyle='--', marker='o', label='Euler')
plt.plot(T, mod_eu, color='r', linestyle='--', marker='s', label='Modified Euler', linewidth=3)
plt.plot(T, r2, color='g', linestyle='--', marker='^', label='RK2', linewidth=3)
plt.plot(T, r4, color='b', linestyle='--', marker='d', label='RK4')
plt.title('Radioactive Decay Over 2000 Years')
plt.xlabel('Time (years)')
plt.ylabel('Remaining Mass (mg)')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Output:-



	Year	Euler	Modified Euler	RK2	RK4
0	0	1.000000	1.000000	1.000000	1.000000
1	100	0.956678	0.957617	0.957617	0.957603
2	200	0.915233	0.917030	0.917030	0.917004
3	300	0.875584	0.878163	0.878163	0.878126
4	400	0.837652	0.840944	0.840944	0.840896
5	500	0.801364	0.805302	0.805302	0.805245
6	600	0.766647	0.771170	0.771170	0.771105
7	700	0.733435	0.738485	0.738485	0.738413
8	800	0.701661	0.707186	0.707186	0.707107
9	900	0.671264	0.677213	0.677213	0.677128
10	1000	0.642184	0.648511	0.648511	0.648420
11	1100	0.614363	0.621025	0.621025	0.620929
12	1200	0.587748	0.594703	0.594703	0.594604
13	1300	0.562286	0.569498	0.569498	0.569394
14	1400	0.537926	0.545361	0.545361	0.545254
15	1500	0.514623	0.522247	0.522247	0.522137
16	1600	0.492328	0.500112	0.500112	0.500000
17	1700	0.471000	0.478916	0.478916	0.478802
18	1800	0.450595	0.458618	0.458618	0.458502
19	1900	0.431075	0.439180	0.439180	0.439063
20	2000	0.412400	0.420566	0.420566	0.420448

Q4.

Que 4: Set up the differential equation for discharging of a capacitor in a **RC** circuit. Given $R = 2\ \Omega, C = 1\mu F$, initial charge$= 4C$ and step size$= 0.2s$.

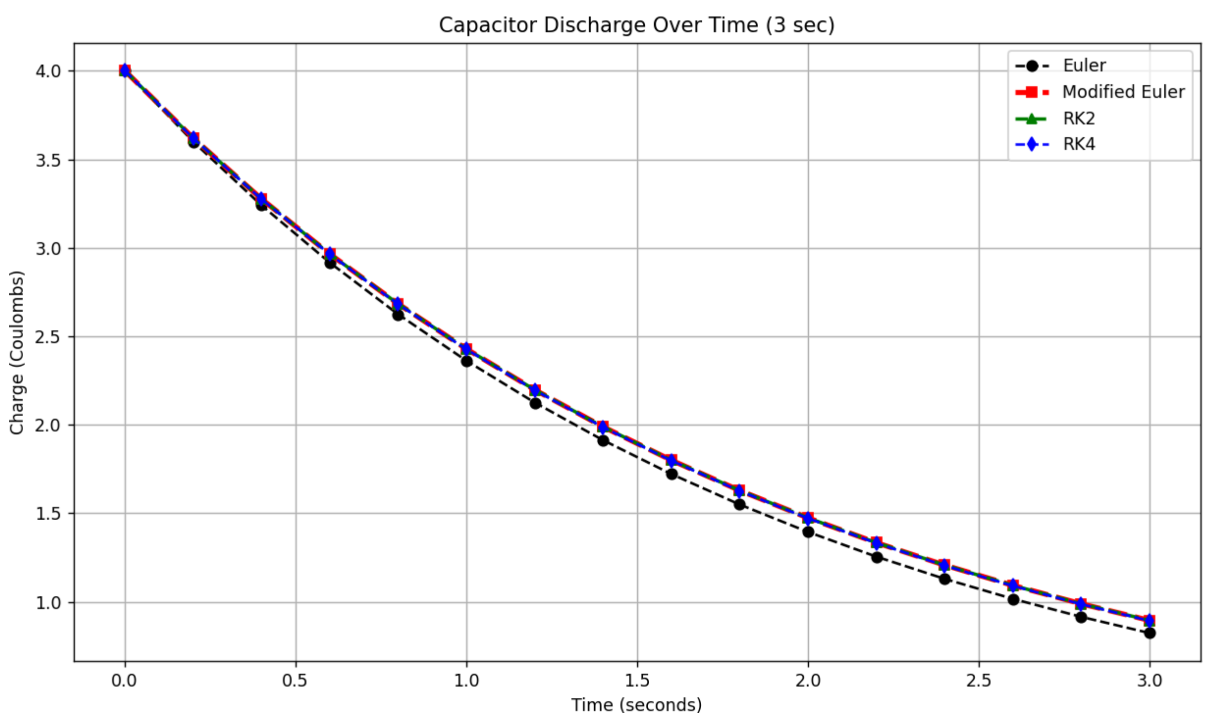
(a) Use the Eular, Modified Eular, Second & Fourth Order Runge Kutta to compute the charge on the capacitor after $1.2s$.

(b) Plot the decay of charge in the circuit as a function of time for $3s$. How much time does it take for the capacitor to get completely discharged? Compare the computed results with the analytical solution [Show all values for each iteration and for each (the Eular, Modified Eular, Second & Fourth Order Runge Kutta) method].

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

R = 2
C = 1
h = 0.2
t_end = 3

def f(t, Q):
    return -1/(R*C) * Q

def euler(x, y):
    return y + h*f(x,
    y)

def mod_euler(x, y, h):
    return y + (h/2)*(f(x, y) + f(x+h, y + h*f(x,y)))

def rk2(x, y, h):
    k1 = h*f(x, y)
    k2 = h*f(x+h, y+k1)
    return y + (k1+k2)/2

def rk4(x, y, h):
    k1  =  h*f(x,
    y)
    k2      =      h*f(x+h/2,
    y+k1/2)       k3       =
    h*f(x+h/2,  y+k2/2)  k4
    = h*f(x+h, y+k3)
    return y + (k1 + 2*k2 + 2*k3 + k4)/6

T = [0]
eu = [4]
mod_eu = [4]
r2 = [4]
r4 = [4]
```

```
while T[-1] < t_end:
    eu.append(euler(T[-1],
    eu[-1]))
    mod_eu.append(mod_euler(T[-1], mod_eu[-1],
    h)) r2.append(rk2(T[-1], r2[-1], h))
    r4.append(rk4(T[-1], r4[-1],
    h)) T.append(T[-1] + h)

df = pd.DataFrame({ 'Time (s)': T, 'Euler': eu, 'Modified Euler': mod_eu, 'RK2': r2, 'RK4': r4})
print(df)

plt.figure(figsize=(10, 6))
plt.plot(T, eu, color='k', marker='o', linestyle='--', label='Euler')
plt.plot(T, mod_eu, color='r', marker='s', linestyle='--', label='Modified Euler', linewidth=3)
plt.plot(T, r2, color='g', marker='^', linestyle='--', label='RK2', linewidth=2)
plt.plot(T, r4, color='b', marker='d', linestyle='--', label='RK4')
plt.title('Capacitor Discharge Over Time (3 sec)')
plt.xlabel('Time (seconds)')
plt.ylabel('Charge (Coulombs)')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Output:-



	Time (s)	Euler	Modified Euler	RK2	RK4
0	0.0	4.000000	4.000000	4.000000	4.000000
1	0.2	3.600000	3.620000	3.620000	3.619350
2	0.4	3.240000	3.276100	3.276100	3.274924
3	0.6	2.916000	2.964870	2.964870	2.963274
4	0.8	2.624400	2.683208	2.683208	2.681281
5	1.0	2.361960	2.428303	2.428303	2.426124
6	1.2	2.125764	2.197614	2.197614	2.195248
7	1.4	1.913188	1.988841	1.988841	1.986342
8	1.6	1.721869	1.799901	1.799901	1.797317
9	1.8	1.549682	1.628910	1.628910	1.626280
10	2.0	1.394714	1.474164	1.474164	1.471519
11	2.2	1.255242	1.334118	1.334118	1.331486
12	2.4	1.129718	1.207377	1.207377	1.204778
13	2.6	1.016746	1.092676	1.092676	1.090128
14	2.8	0.915072	0.988872	0.988872	0.986389
15	3.0	0.823565	0.894929	0.894929	0.892522

Q5.

Que 5: Set up the differential equation for growth of current across the inductor in an **R-L** circuit by applying a voltage V across the inductor and resistor R in series.

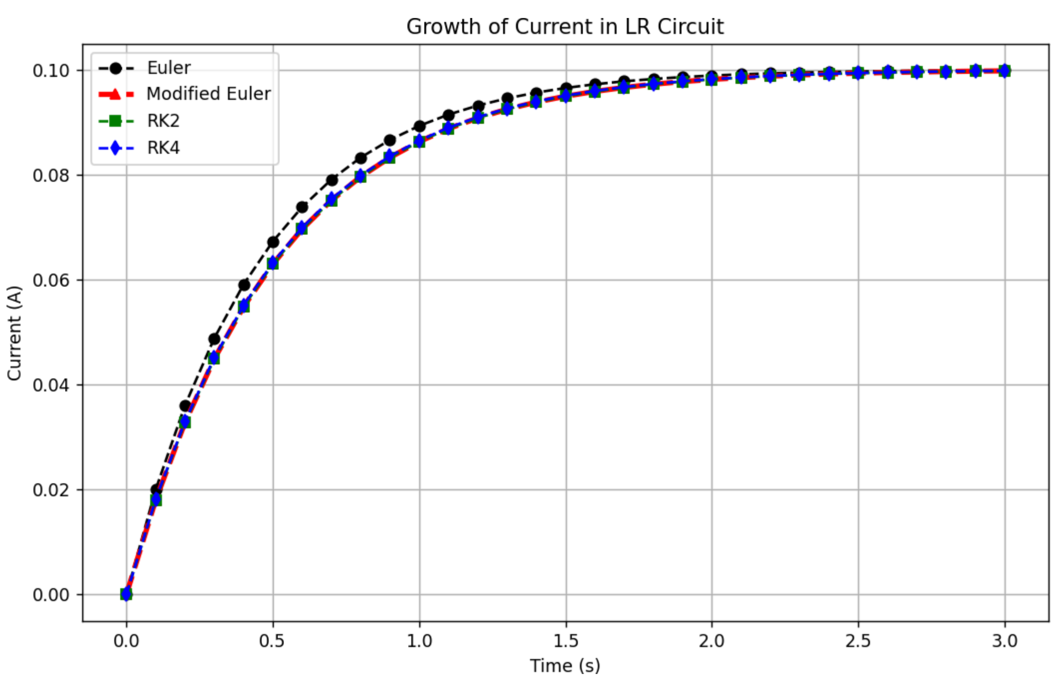
(a) Determine the current in steps of 0.1s using the Eular, Modified Eular, Second & Fourth Order Runge Kutta Methods. If R=10Ω, L=5H, V=1V and the initial current in the circuit is zero.

(b) Also plot the growth of current as a function of time. What will be the value of current across the indicator after 2.5s ? Plot the growth curve and compare the computed results with the analytical solution [Show all values for each iteration and for each (the Eular, Modified Eular, Second & Fourth Order Runge Kutta) method].

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def f(i, t):
    return (V - R * i) / L

def euler(fun, i, t, h):
    return i + h * fun(i, t)

def mod_euler(fun, i, t, h):
    k1 = fun(i, t)
    k2 = fun(i + h * k1, t + h)
    return i + (h/2) * (k1 + k2)

def rk2(fun, i, t, h):
    k1 = fun(i, t)
    k2 = fun(i + (h/2) * k1, t + h/2)
    return i + h * k2

def   rk4(fun,   i,   t,
    h): k1 = fun(i, t)
    k2 = fun(i + (h/2) * k1, t + h/2)
    k3 = fun(i + (h/2) * k2, t + h/2)
    k4 = fun(i + h * k3, t + h)
    return i + (h/6) * (k1 + 2*k2 + 2*k3 + k4)

R = 10
L = 5
V = 1
h = 0.1
t_end = 3
t = np.arange(0, t_end + h, h)
```

```
eu = [0]
mod_eu = [0]
r2 = [0]
r4 = [0]

for i in t[:-1]:
    eu.append(euler(f, eu[-1], i, h))
    mod_eu.append(mod_euler(f, mod_eu[-1], i, h))
    r2.append(rk2(f, r2[-1], i, h))
    r4.append(rk4(f, r4[-1], i, h))

df = pd.DataFrame({'Time (s)': t, 'Euler': eu, 'Modified Euler': mod_eu, 'RK2': r2, 'RK4': r4})
print(df)

plt.figure(figsize=(10,6))
plt.plot(t, eu, color='k', linestyle='--', marker='o', label='Euler')
plt.plot(t, mod_eu, color='r', linestyle='--', marker='^', label='Modified Euler', linewidth=3)
plt.plot(t, r2, color='g', linestyle='--', marker='s', label='RK2')
plt.plot(t, r4, color='b', linestyle='--', marker='d', label='RK4')
plt.xlabel('Time (s)')
plt.ylabel('Current (A)')
plt.title("Growth of Current in LR Circuit")
plt.legend()
plt.grid()
plt.show()
```

Output:-



	Time (s)	Euler	Modified Euler	RK2	RK4
0	0.0	0.000000	0.000000	0.000000	0.000000
1	0.1	0.020000	0.018000	0.018000	0.018127
2	0.2	0.036000	0.032760	0.032760	0.032968
3	0.3	0.048800	0.044863	0.044863	0.045118
4	0.4	0.059040	0.054788	0.054788	0.055067
5	0.5	0.067232	0.062926	0.062926	0.063211
6	0.6	0.073786	0.069599	0.069599	0.069880
7	0.7	0.079028	0.075071	0.075071	0.075340
8	0.8	0.083223	0.079559	0.079559	0.079810
9	0.9	0.086578	0.083238	0.083238	0.083470
10	1.0	0.089263	0.086255	0.086255	0.086466
11	1.1	0.091410	0.088729	0.088729	0.088919
12	1.2	0.093128	0.090758	0.090758	0.090928
13	1.3	0.094502	0.092422	0.092422	0.092572
14	1.4	0.095602	0.093786	0.093786	0.093919
15	1.5	0.096482	0.094904	0.094904	0.095021
16	1.6	0.097185	0.095821	0.095821	0.095924
17	1.7	0.097748	0.096574	0.096574	0.096662
18	1.8	0.098199	0.097190	0.097190	0.097267
19	1.9	0.098559	0.097696	0.097696	0.097763
20	2.0	0.098847	0.098111	0.098111	0.098168
21	2.1	0.099078	0.098451	0.098451	0.098500

20	2.0	0.098847	0.098111	0.098111	0.098168
21	2.1	0.099078	0.098451	0.098451	0.098500
22	2.2	0.099262	0.098730	0.098730	0.098772
23	2.3	0.099410	0.098958	0.098958	0.098995
24	2.4	0.099528	0.099146	0.099146	0.099177
25	2.5	0.099622	0.099300	0.099300	0.099326
26	2.6	0.099698	0.099426	0.099426	0.099448
27	2.7	0.099758	0.099529	0.099529	0.099548
28	2.8	0.099807	0.099614	0.099614	0.099630
29	2.9	0.099845	0.099683	0.099683	0.099697
30	3.0	0.099876	0.099740	0.099740	0.099752