WElcome! Let's get started!!!

```
#importing important libraries
import numpy as np
import matplotlib.pyplot as plt
```

```
#solution using the euler method
```

```
#system of odes function
def func(y, t):
    return np.array([y[1], np.sin(y[0])])
```

```
#setting the initial conditions
y0 = np.array([np.pi /4, 0.0])
```

```
#setting the value for n (we are to input various values to test the output)
P=int(input('Enter the value of power(p) e.g 2**1 => P=1 P: '))
n=2**P
```

```
Enter the value of power(p) e.g 2**1 => P=1 P: 17
```

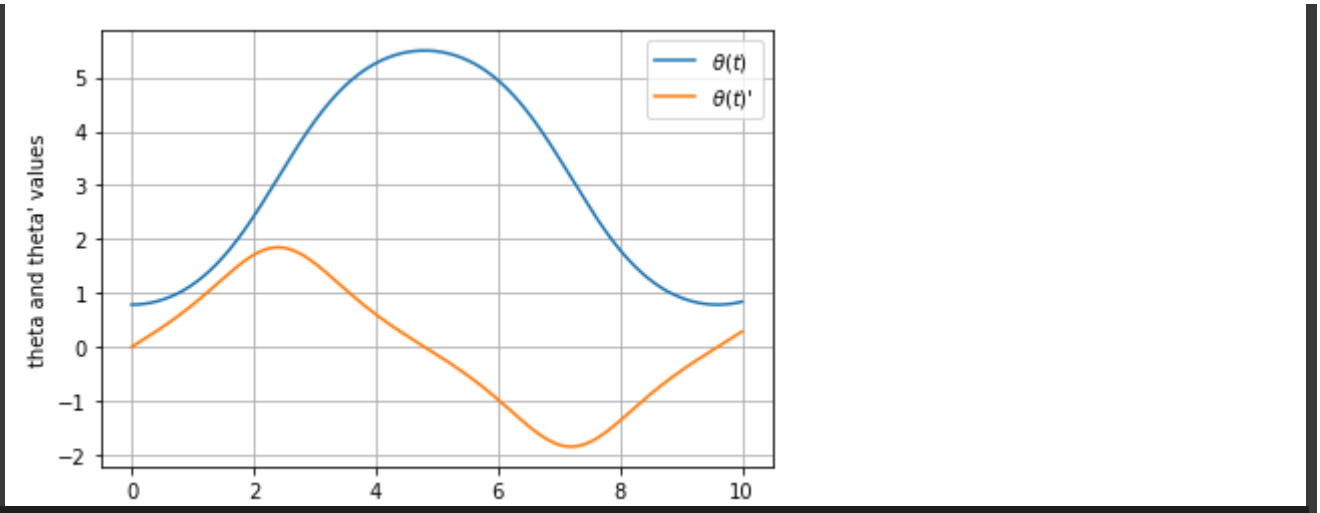
```
#setting the range of t values
t = np.linspace(0, 10, n)
```

```
#function for euler method
def euler_method(f, y0, t):
    n = len(t)
    y = np.zeros((n, len(y0)))
    y[0] = y0
    for i in range(n - 1):
        y[i+1] =(t[i+1] - t[i]) * f(y[i], t[i])+ y[i]
    return y
```

```
euler_solution = euler_method(func, y0, t)
```

```
#plotting the results
```

```
plt.plot(t, euler_solution[:, 0],label=r"$\theta(t)$")
plt.plot(t, euler_solution[:, 1],label=r"$\theta(t)$'")
plt.ylabel("theta and theta' values" )
plt.xlabel('t vals')
plt.legend()
plt.grid()
```




```
#finding the solution at a=30
np.interp(30, t,euler_solution[:, 0])
```

```
0.8412440520647136
```

```
#solution using the runger kutta method
```

```
##the system of odes function
def func(y, t):
    return np.array([y[1], np.sin(y[0])])
```

```
#initial conditions
y0 = np.array([np.pi /4, 0.0])
```

```
#n values param
P=int(input('Enter the value of power(p) e.g 2**1 => P=1 P: '))
n=2**P
```

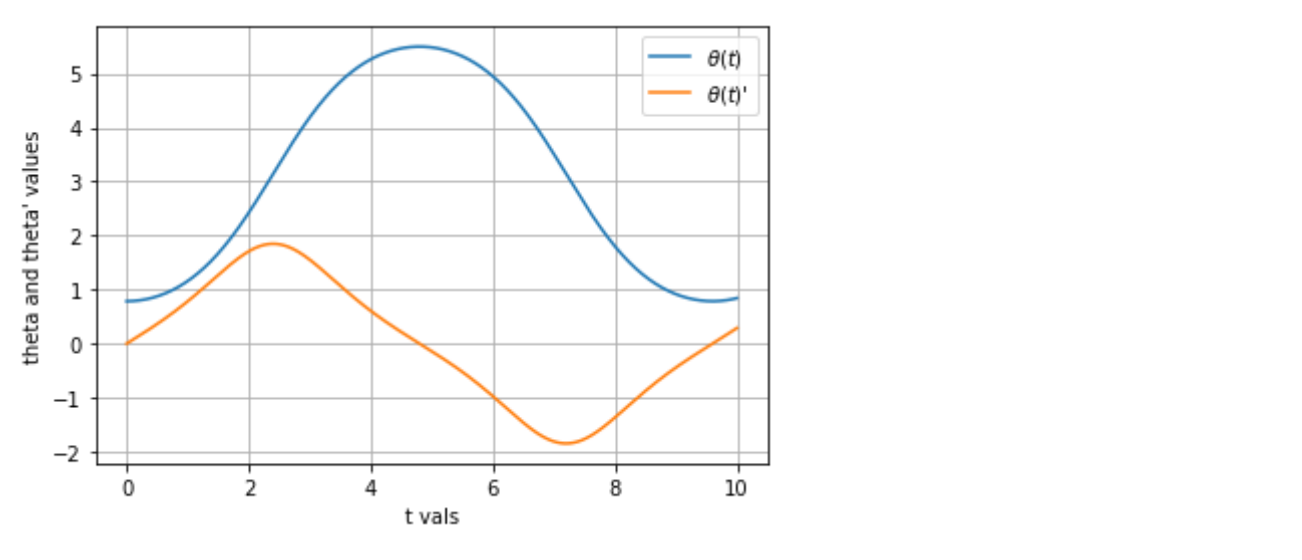
```
Enter the value of power(p) e.g 2**1 => P=1 P: 17
```

```
#set range val
t = np.linspace(0, 10, n)
```

```
#runge kutta function
def rungekutta4(f, y0, t):
    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])
        k2 = f(y[i] + k1 * h / 2, t[i] + h / 2)
        k3 = f(y[i] + k2 * h / 2, t[i] + h / 2)
        k4 = f(y[i] + k3 * h, t[i] + h)
        y[i+1] = (k1 + 2*k2 + 2*k3 + k4)*(h / 6) +y[i]
    return y
```

```
runge_kutta_solution= rungekutta4(func, y0, t)
```

```
plt.plot(t, runge_kutta_solution[:, 0], label=r'$\theta(t)$')
plt.plot(t, runge_kutta_solution[:, 1], label=r"$\theta(t)$'")
plt.ylabel("theta and theta' values" )
plt.xlabel('t vals')
plt.legend()
plt.grid()
```




```
#finding the solution at a=30
np.interp(30, t,runge_kutta_solution[:, 0]) #runge_kutta_solution
```

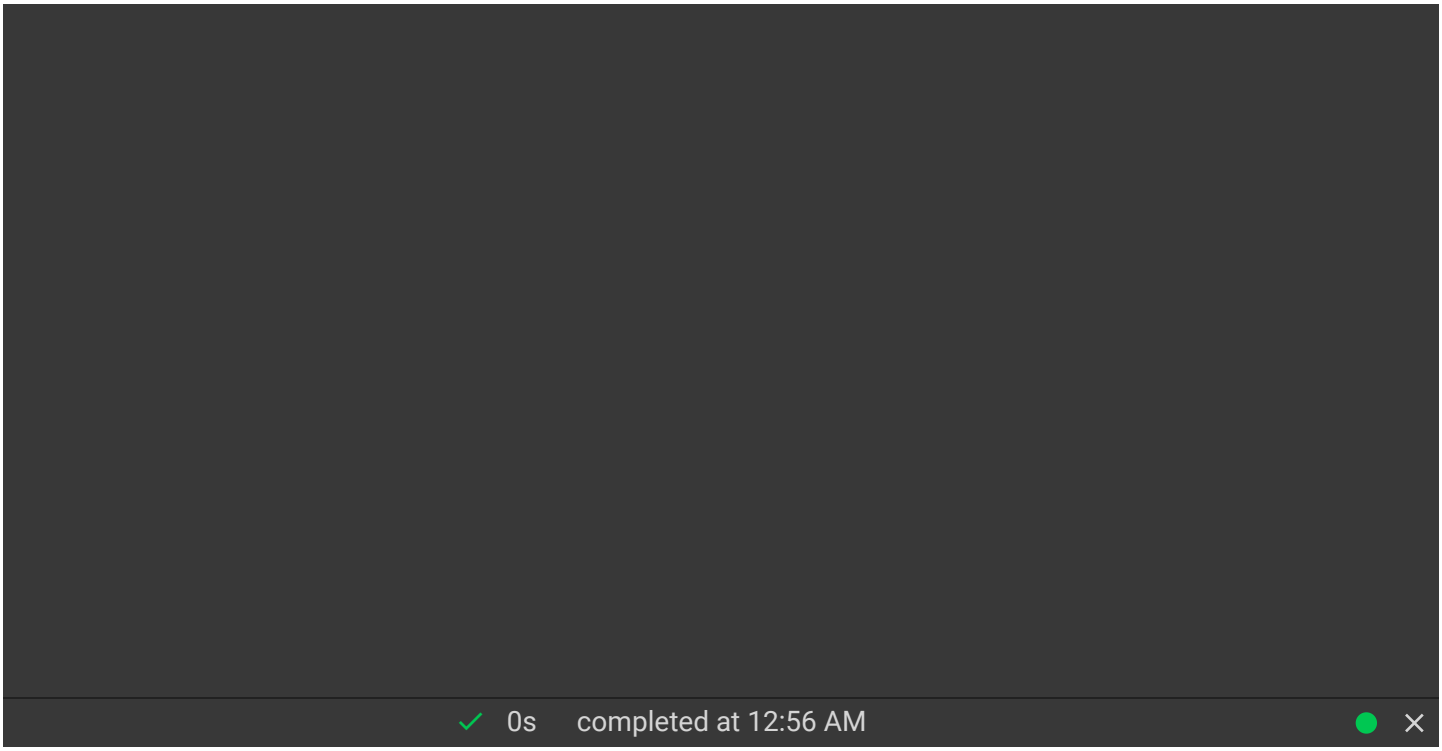
```
0.8423870795918288
```

```
#approximations

#at n=2
# ==> euler approximation=0.7853981633974483
# ==>runge kutta approximation=18.073324066944718

#at n=4
# ==> euler approximation=24.35562420294903
# ==>runge kutta approximation=0.8052655530761057

#at n=2**3
# ==> euler approximation=16.054823723316375
# ==>runge kutta approximation=1.1606909441616713

#at n=2**4
# ==> euler approximation=19.820698268921404
# ==>runge kutta approximation=0.8676538756766228

#at n=2**5
# ==> euler approximation=16.075593415171607
# ==>runge kutta approximation=0.8431265571542572

#at n=2**6
# ==> euler approximation=12.718831843250436
```

```
# ==>runge kutta approximation=0.8424053570533903

#at n=2**7
# ==> euler approximation=1.0645903661879352
# ==>runge kutta approximation=0.8423873972247754

#at n=2**8
# ==> euler approximation=0.47770105361226567
# ==>runge kutta approximation=0.842387074801793

#at n=2**9
# ==> euler approximation=0.5984810341734176
# ==>runge kutta approximation=0.8423870785425398

#at n=2**10
# ==> euler approximation=0.7074986452365337
# ==>runge kutta approximation=0.8423870795032059

#at n=2**11
# ==> euler approximation=0.7719783125579635
# ==>runge kutta approximation=0.842387079585641

#at n=2**12
# ==> euler approximation=0.8064747915345457
# ==>runge kutta approximation=0.8423870795915009

#at n=2**13
# ==> euler approximation=0.8242581226090757
# ==>runge kutta approximation=0.8423870795918872

#at n=2**14
# ==> euler approximation=0.8332799196878087
# ==>runge kutta approximation=0.8423870795918822

#at n=2**15
# ==> euler approximation=0.8378228946275429
# ==>runge kutta approximation=0.8423870795920042

#at n=2**16
# ==> euler approximation=0.8401023440318409
# ==>runge kutta approximation=0.8423870795919647

#at n=2**17
# ==> euler approximation=0.8412440520647136
# ==>runge kutta approximation=0.8423870795918288
```

```
#it's very clear that runge kutta method produces an accurate approximation quicker with a
```

THANK YOU!!!


0s completed at 12:56 AM