AKSHAY J

21105012

FINITE ELEMENT METHODS FOR FLUID

ASSIGNMENT-1

A> Consider a flow in which velocity field is given as:-

$$V_\theta = 3/r \quad , V_r = 0.$$

This corresponds to a flow due to free vortex and results in circular streamlines. Trace, numerically, the trajectory of a fluid particle released at $r=1$ and $\theta=0$ by following:-

1> Write equations for $\dot{\theta}$ and $\dot{r}$. find the analytical sol^n.

Ans: $\frac{d}{dt}(r) = \dot{r} = V_r \Rightarrow \underline{\underline{\dot{r} = 0}}$

$\frac{d}{dt}(r\theta) = V_\theta = 3/r$

$r\theta = 3t/r + c_1$

$\theta = 3t/r^2 + c_1/r$

$\frac{d\theta}{dt} = \underline{\underline{\dot{\theta} = 3/r^2}}$

2> Employing the basic Euler method solve the differential equation in cylinderical /polar coordinates. Utilise a computer program and plot your results. Carry out the computations for several revolutions of the particle. Compare the result for various values of Δt. (0.1, 0.01, 0.001, 0.0001)

Ans. Euler method

$\frac{dr}{dt} = f(r,\theta) = V_r = 0$

$\frac{r^{n+1} - r^n}{\Delta t} = 0 \Rightarrow \underline{\underline{r^{n+1} = r^n}}$

$\frac{d\theta}{dt} = 3/r^2 = g(r,\theta)$

$\frac{\theta^{n+1} - \theta^n}{\Delta t} = 3/r_n^2.$

$\underline{\underline{\theta^{n+1} = \theta^n + \frac{3\Delta t}{r_n^2}}}$

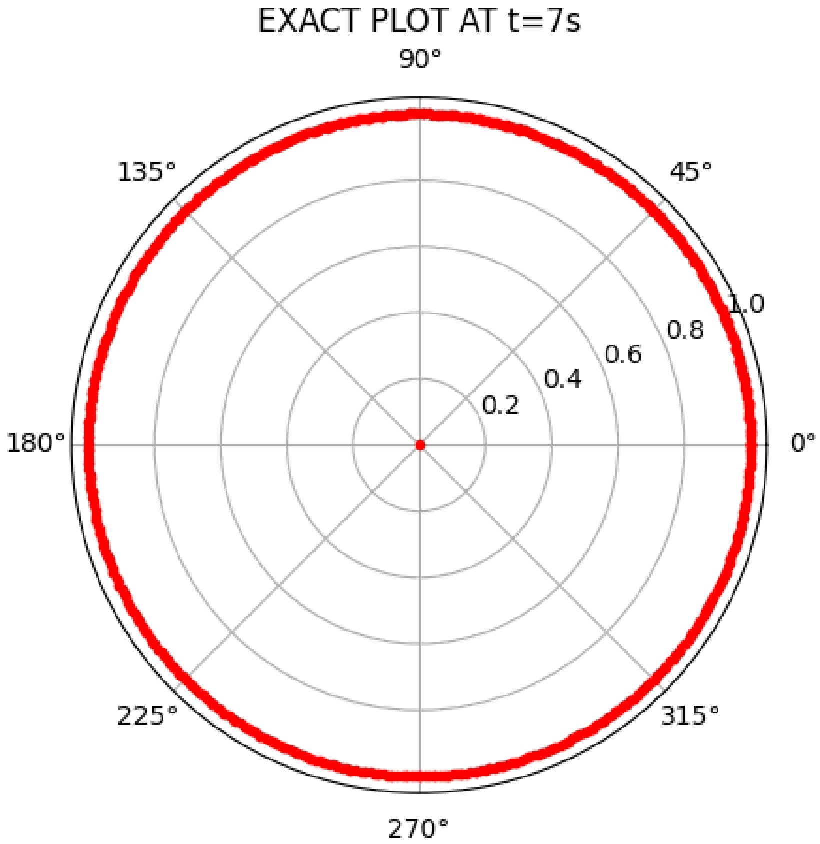
ω_0 and r values for different dt values are plotted and accuracy of plotting increases with decrease in dt values.

$\theta_{exact} = 3t/r^2$ $\qquad$ $r_{exact} = 1$

CODE FOR EXACT SOLUTION:-

```
import numpy as np
from  matplotlib import pyplot as plt
from matplotlib import animation

##SETTING THE CELL

r=1
t=np.linspace(0,7,500)
theta_exact=np.zeros(500)
r_exact=np.zeros(500)

##CALCULATION

for n in range(0,len(t)-1):
    theta_exact[n]=(3*t[n])/r**2
    r_exact[n]=r

##ANIMATING THE PLOT

fig =plt.figure()
ax=plt.subplot(projection="polar")
def animate(i):
    ax.plot(theta_exact[i],r_exact[i],'r.')
anim=animation.FuncAnimation(fig,animate,frames
```

```
=len(t),interval=100,blit=False)
plt.title('EXACT PLOT AT t=7s')
plt.show()
```



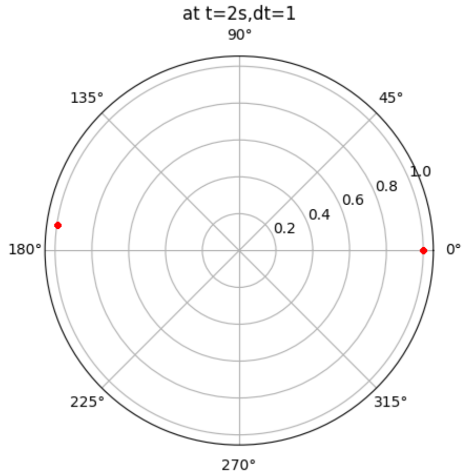
CODE FOR EULER METHOD IN POLAR:-

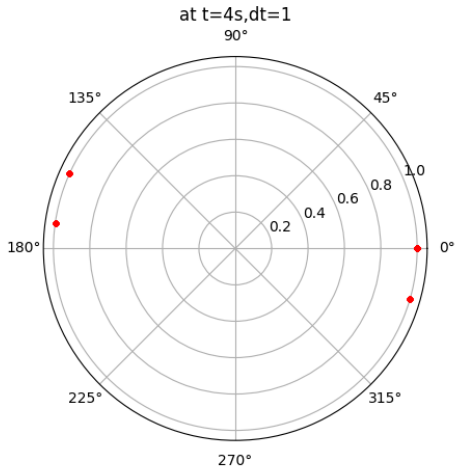
```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation

#Setting the cell

t=500
dt1=1
dt2=0.1
dt3=0.01
```

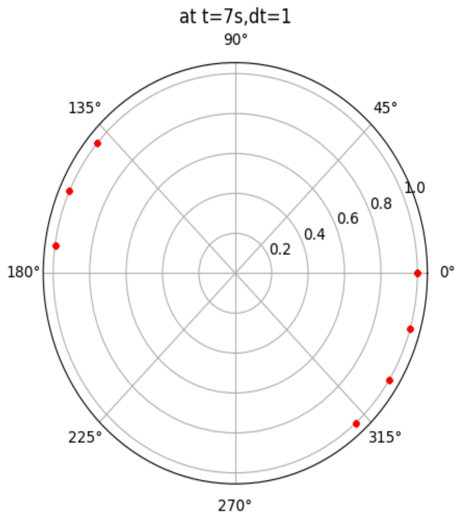
```python
dt4=0.001
theta_1=np.zeros(t)
theta_2=np.zeros(t)
theta_3=np.zeros(t)
theta_4=np.zeros(t)
theta_exact=np.zeros(t)
r_1=np.zeros(t)
r_2=np.zeros(t)
r_3=np.zeros(t)
r_4=np.zeros(t)


##INITIAL_VALUES

r_1[0]=1
r_2[0]=1
r_3[0]=1
r_4[0]=1
theta_1[0]=0
theta_2[0]=0
theta_3[0]=0
theta_4[0]=0

##CALCULATION

for n in range(0,t-1):
    theta_1[n + 1] = theta_1[n] + (3 / (r_1[n]
** 2)) * dt1
    theta_2[n + 1] = theta_2[n] + (3 / (r_2[n]
** 2)) * dt2
    theta_3[n + 1] = theta_3[n] + (3 / (r_3[n]
** 2)) * dt3
    theta_4[n + 1] = theta_4[n] + (3 / (r_4[n]
** 2)) * dt4
    r_1[n + 1] = r_1[n]
    r_2[n + 1] = r_2[n]
    r_3[n + 1] = r_3[n]
    r_4[n + 1] = r_4[n]
```

```
##ANIMATING THE PLOT

fig =plt.figure(figsize=(12,9))
ax1=plt.subplot(221,projection='polar')
ax2=plt.subplot(222,projection='polar')
ax3=plt.subplot(223,projection='polar')
ax4=plt.subplot(224,projection='polar')
ax1.set_title('dt=1',color='r',loc='left')
ax2.set_title('dt=0.1',color='g',loc='left')
ax3.set_title('dt=0.01',color='b',loc='left')
ax4.set_title('dt=0.001',color='y',loc='left')
def animate(i):
    ax1.plot(theta_1[i],
r_1[i],'r.',label='dt=1')
    ax2.plot(theta_2[i], r_2[i],
'g.',label='dt=0.1')
    ax3.plot(theta_3[i], r_3[i],
'b.',label='dt=0.01')
    ax4.plot(theta_4[i],
r_4[i],'y.',label='dt=0.001')

anim=animation.FuncAnimation(fig,animate,frames
=t,interval=100,blit=False)
plt.suptitle('Comparison plot for different dt
values-EULER(POLAR)',color='r')
plt.show()
```

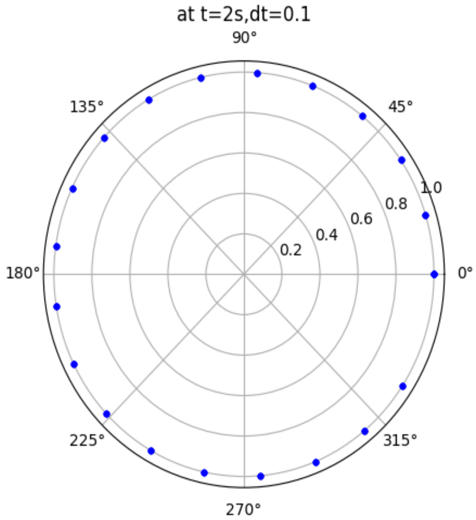
dt=1:-

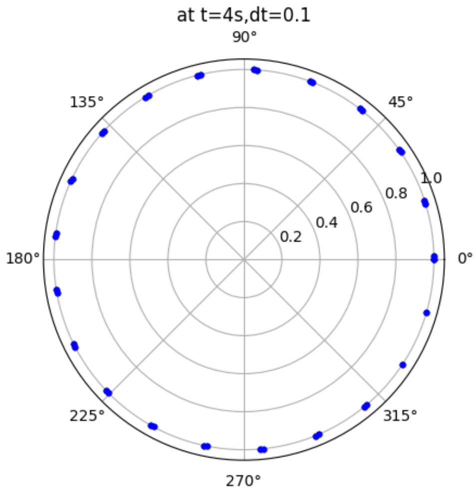

at t=2s,dt=1
90°
45°
135°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0

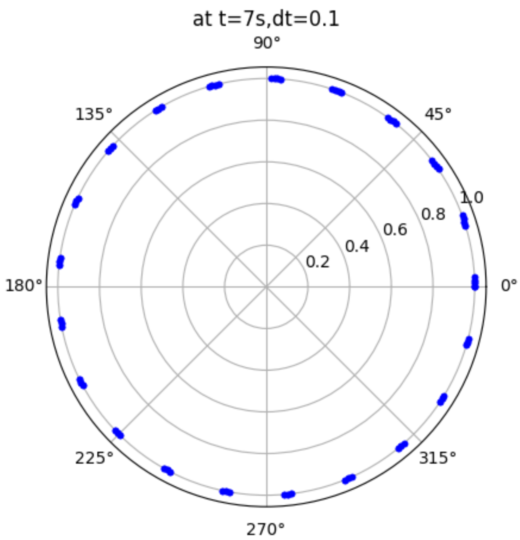

at t=4s,dt=1
90°
45°
135°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0


at t=7s,dt=1
90°
45°
135°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0

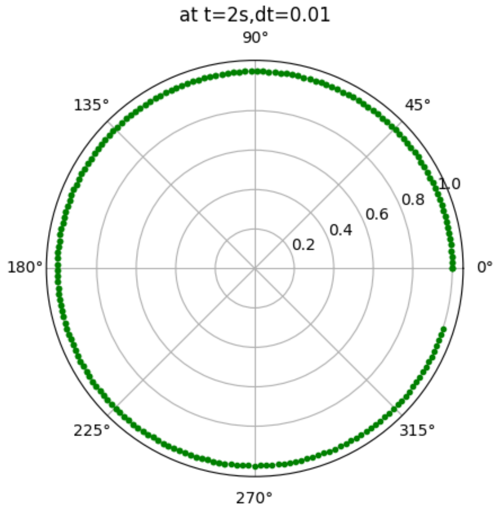
dt=0.1:-

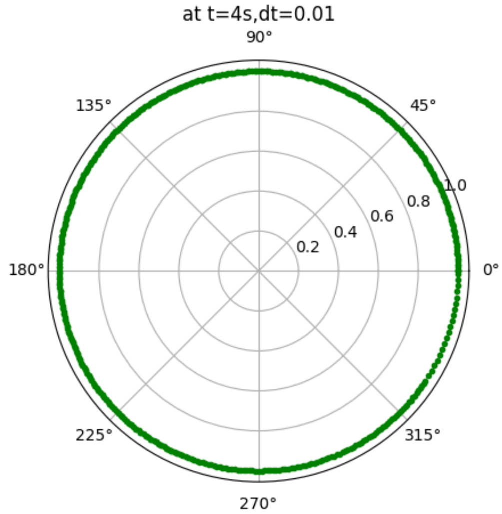

at t=2s,dt=0.1
90°
45°
135°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0

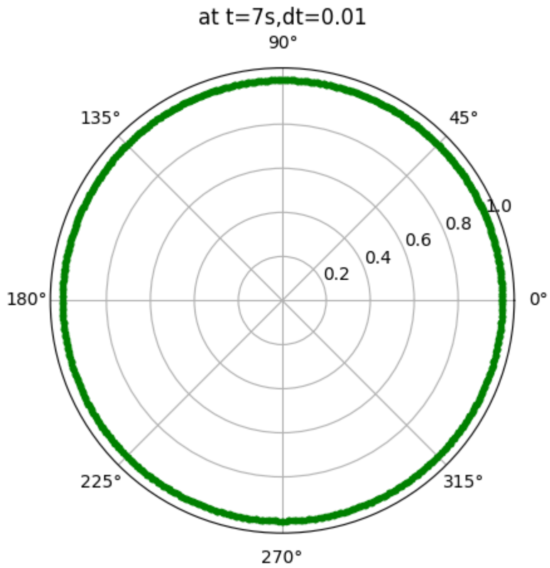

at t=4s,dt=0.1
90°
45°
135°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0


at t=7s,dt=0.1
90°
45°
135°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0

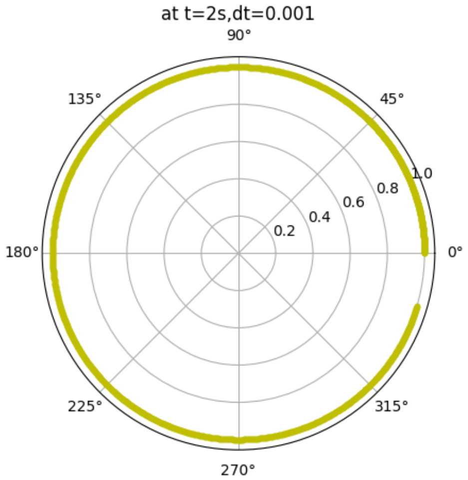
dt=0.01:-

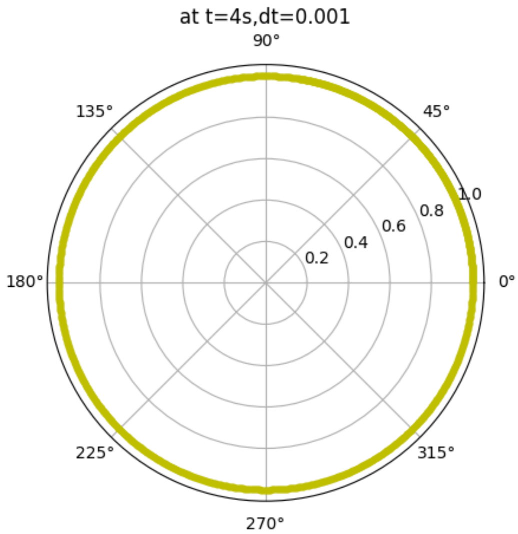

at t=2s,dt=0.01
90°
135°
45°
1.0
0.8
0.6
0.4
0.2
180°
0°
225°
315°
270°

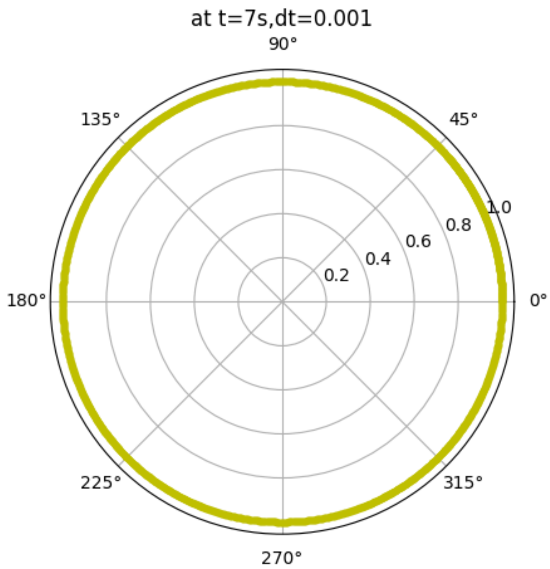

at t=4s,dt=0.01
90°
135°
45°
1.0
0.8
0.6
0.4
0.2
180°
0°
225°
315°
270°


at t=7s,dt=0.01
90°
135°
45°
1.0
0.8
0.6
0.4
0.2
180°
0°
225°
315°
270°

dt=0.001:-


at t=2s,dt=0.001
90°
45°
135°
0.2
0.4
0.6
0.8
1.0
180°
0°
225°
315°
270°


at t=4s,dt=0.001
90°
45°
135°
0.2
0.4
0.6
0.8
1.0
180°
0°
225°
315°
270°


at t=7s,dt=0.001
90°
45°
135°
0.2
0.4
0.6
0.8
1.0
180°
0°
225°
315°
270°

3> Repeat the above step for improved Euler.

Improved Euler

Predictor step :- $r_1^{*\,n+1} = r^{n} + \Delta t\, f(r^n,\theta^n) = r^n + \Delta t \times 0$

Corrector step :- $r^{n+1} = r^n + \frac{\Delta t}{2}\left[f(r^n,\theta^n) + f(r^{*n+1},\theta^{*n+1})\right]$

$$r^{n+1} = r^n + \frac{\Delta t}{2}\left[r^{*n} + r^{*n+1}\right] = r^n + \frac{\Delta t}{2}(0)$$

Predictor step :- $\theta^{*\,n+1} = \theta^n + \Delta t\, f(r^n,\theta^n) = \theta^n + \Delta t \times 3/r_n^2$

Corrector step :- $\theta^{n+1} = \theta^n + \frac{\Delta t}{2}\left[f(r^n,\theta^n) + f(r^{*n+1},\theta^{*n+1})\right]$

$$\theta^{n+1} = \theta^n + \frac{\Delta t}{2}\left[3/(r^n)^2 + 3/(r^{*n+1})^2\right]$$

CODE FOR IMPROVED EULER IN POLAR:-

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation

#Setting the cell

t=500
dt1=1
dt2=0.1
dt3=0.01
dt4=0.001
theta_1=np.zeros(t)
theta_2=np.zeros(t)
theta_3=np.zeros(t)
theta_4=np.zeros(t)
theta_1_a=np.zeros(t)
theta_2_a=np.zeros(t)
theta_3_a=np.zeros(t)
theta_4_a=np.zeros(t)
```

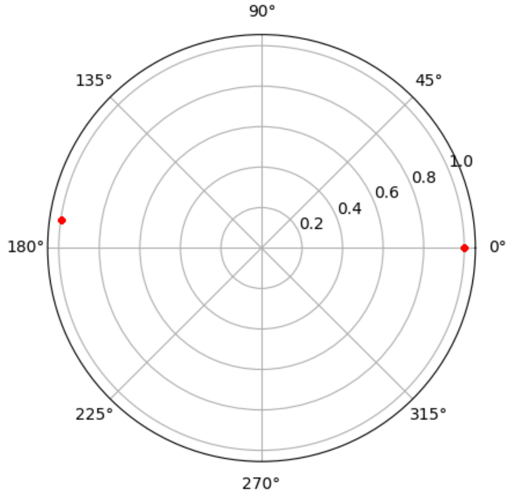
```
r_1=np.zeros(t)
r_2=np.zeros(t)
r_3=np.zeros(t)
r_4=np.zeros(t)
r_1_a=np.zeros(t)
r_2_a=np.zeros(t)
r_3_a=np.zeros(t)
r_4_a=np.zeros(t)

##INITIAL_VALUES

r_1[0]=1
r_2[0]=1
r_3[0]=1
r_4[0]=1
r_1[0]=1
r_2[0]=1
r_3[0]=1
r_4[0]=1
theta_1[0]=0
theta_2[0]=0
theta_3[0]=0
theta_4[0]=0
theta_1_a[0]=0
theta_2_a[0]=0
theta_3_a[0]=0
theta_4_a[0]=0

##CALCULATION

for n in range(0,t-1):

 #PREDICTOR STEP
    r_1_a[n + 1] = r_1[n]+dt1*0
    r_2_a[n + 1] = r_2[n]+dt2*0
    r_3_a[n + 1] = r_3[n]+dt3*0
    r_4_a[n + 1] = r_4[n]+dt4*0
    theta_1_a[n + 1] = theta_1[n] + ((3 * dt1)
```

```
/ r_1[n] ** 2)
    theta_2_a[n + 1] = theta_2[n] + ((3 * dt2)
/ r_2[n] ** 2)
    theta_3_a[n + 1] = theta_3[n] + ((3 * dt3)
/ r_3[n] ** 2)
    theta_4_a[n + 1] = theta_4[n] + ((3 * dt4)
/ r_4[n] ** 2)

 #CORRECTOR STEP

    r_1[n + 1] = r_1[n] + (dt1 * 0.5) * 0
    r_2[n + 1] = r_2[n] + (dt2 * 0.5) * 0
    r_3[n + 1] = r_3[n] + (dt3 * 0.5) * 0
    r_4[n + 1] = r_4[n] + (dt4 * 0.5) * 0
    theta_1[n + 1] = theta_1[n] + (3 * 0.5 *
dt1) * ((1 / r_1[n] ** 2) + (1 / r_1_a[n + 1]
** 2))
    theta_2[n + 1] = theta_2[n] + (3 * 0.5 *
dt2) * ((1 / r_2[n] ** 2) + (1 / r_2_a[n + 1]
** 2))
    theta_3[n + 1] = theta_3[n] + (3 * 0.5 *
dt3) * ((1 / r_3[n] ** 2) + (1 / r_3_a[n + 1]
** 2))
    theta_4[n + 1] = theta_4[n] + (3 * 0.5 *
dt4) * ((1 / r_4[n] ** 2) + (1 / r_4_a[n + 1]
** 2))

#ANIMATING THE PLOT

fig =plt.figure(figsize=(12,9))
ax1=plt.subplot(221,projection='polar')
ax2=plt.subplot(222,projection='polar')
ax3=plt.subplot(223,projection='polar')
ax4=plt.subplot(224,projection='polar')
ax1.set_title('dt=1',color='r',loc='left')
ax2.set_title('dt=0.1',color='g',loc='left')
ax3.set_title('dt=0.01',color='b',loc='left')
ax4.set_title('dt=0.001',color='y',loc='left')
```

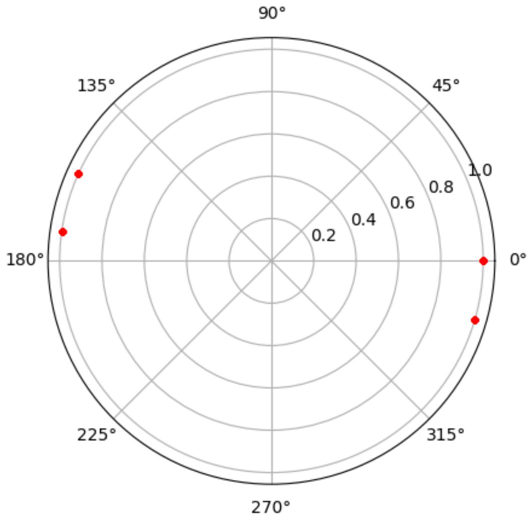
```
def animate(i):
    ax1.plot(theta_1[i], r_1[i],
'r.',label='dt=1')
    ax2.plot(theta_2[i], r_2[i],
'g.',label='dt=0.1')
    ax3.plot(theta_3[i], r_3[i],
'b.',label='dt=0.01')
    ax4.plot(theta_4[i], r_4[i],
'y.',label='dt=0.001')

anim=animation.FuncAnimation(fig,animate,frames
=t,interval=100,blit=False)
plt.suptitle('Comparison plot for different dt
values-IMPROVED EULER(POLAR)',color='r')
plt.show()
```

dt=1:-

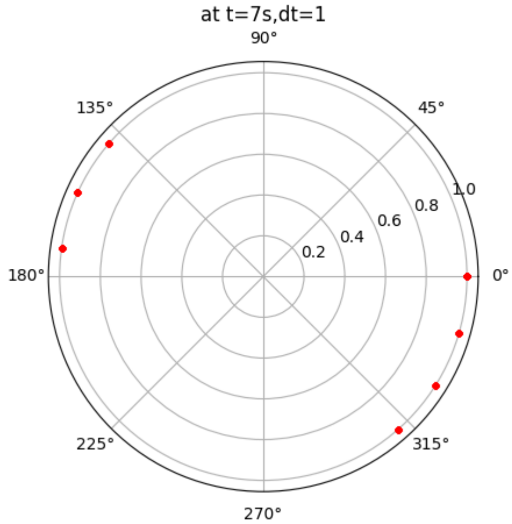
at t=2s,dt=1



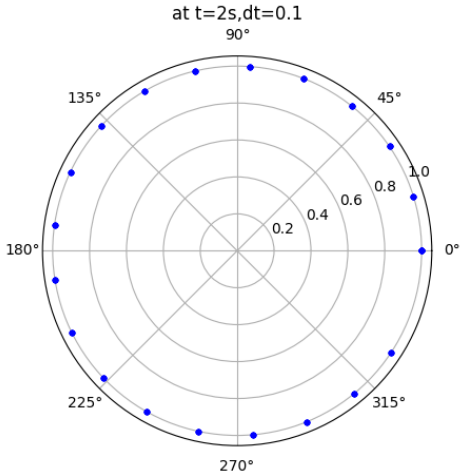
at t=4s,dt=1

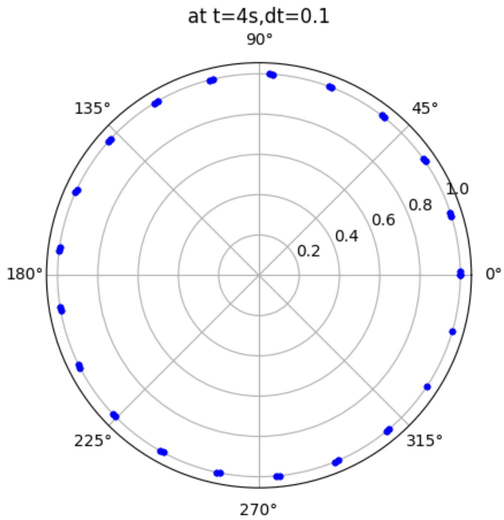


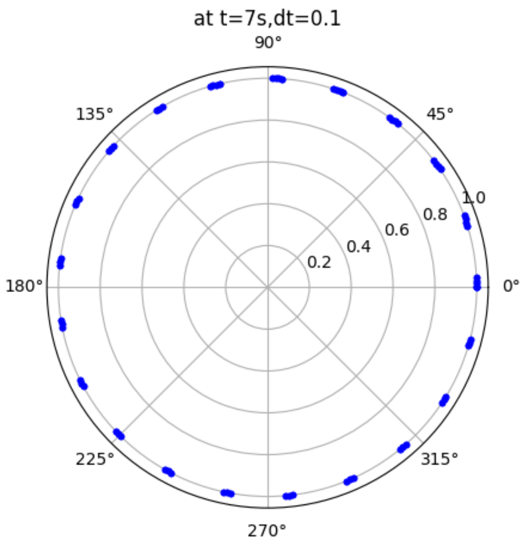
at t=7s,dt=1



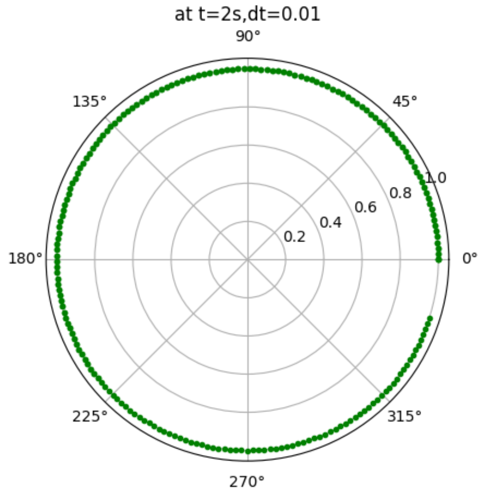
dt=0.1:-

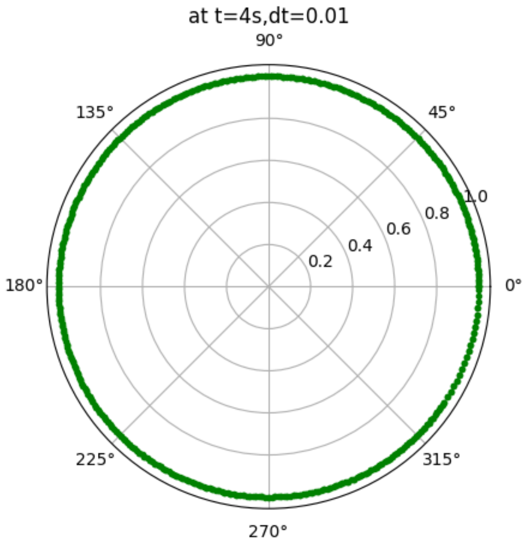

at t=2s,dt=0.1
90°
45°
135°
0.2
0.4
0.6
0.8
1.0
180°
0°
225°
315°
270°

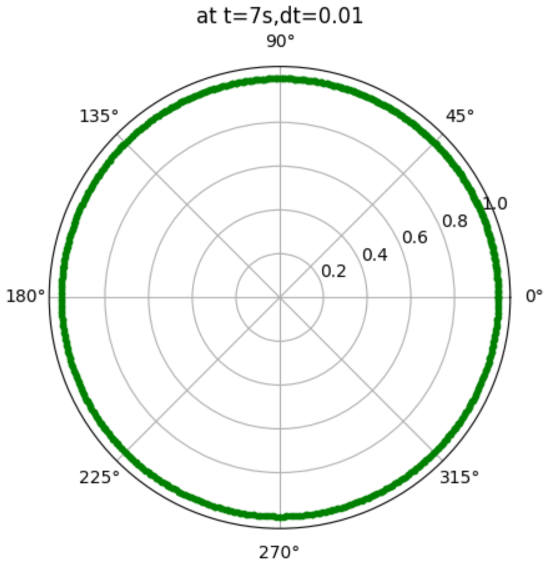

at t=4s,dt=0.1
90°
45°
135°
0.2
0.4
0.6
0.8
1.0
180°
0°
225°
315°
270°


at t=7s,dt=0.1
90°
45°
135°
0.2
0.4
0.6
0.8
1.0
180°
0°
225°
315°
270°

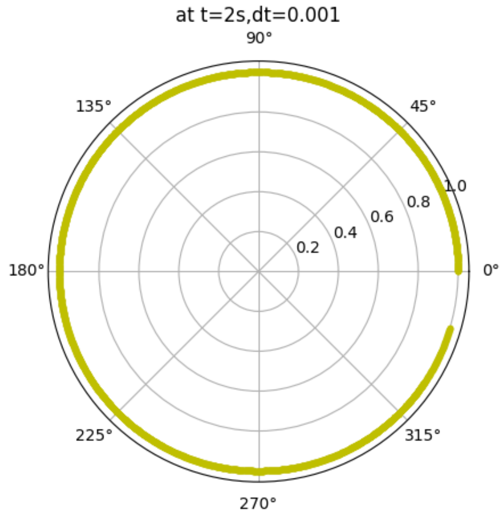
dt=0.01:-


at t=2s,dt=0.01
90°
45°
135°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0

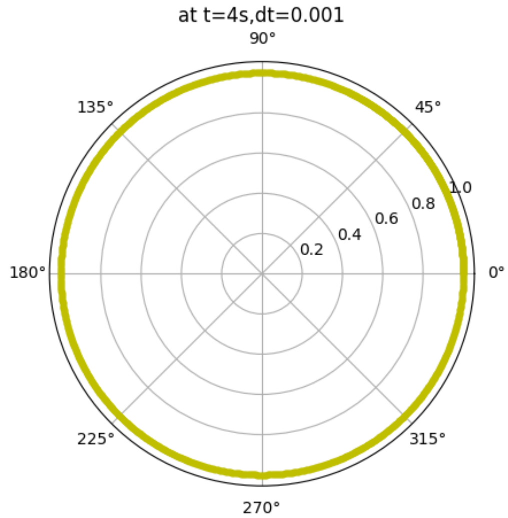

at t=4s,dt=0.01
90°
45°
135°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0


at t=7s,dt=0.01
90°
45°
135°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0

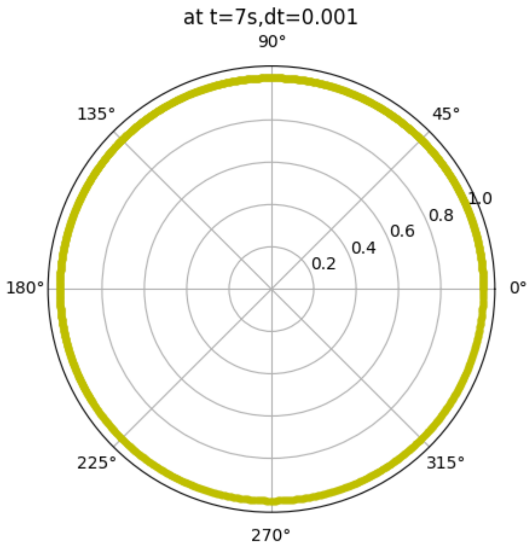
dt=0.001:-



at t=2s,dt=0.001
90°
135°
45°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0



at t=4s,dt=0.001
90°
135°
45°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0

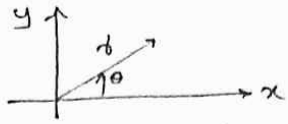


at t=7s,dt=0.001
90°
135°
45°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0

4> Comment on relative accuracy of two methods and effect of time step. $\left(\frac{\partial\theta}{\partial t} = f(r,\theta)\right)$

Ans. It can be seen that since $f(r,\theta)$ is found to be a constant both euler as well as improved euler trace out the values exactly. With decrease in ∂t values the traced out plot becomes more exact to the real solution. It can also been seen that with decrease in ∂t the amount of computational resources needed increased.

B> Repeat the above computations in cartesian coordinate.



$$r = \sqrt{x^2+y^2}.$$

$$\frac{dr}{dt} = \frac{1}{\cancel{2}\sqrt{x^2+y^2}}(\cancel{2}x\dot{x} + 2y\dot{y}) = V_r = 0$$

$$\Rightarrow \quad x\dot{x} + y\dot{y} = 0 \quad — (1)$$

$$\theta = \tan^{-1}(y/x)$$

$$\frac{d\theta}{dt} = \frac{1}{1+y^2/x^2} \times \frac{x\dot{y} - y\dot{x}}{x^2} = \frac{\cancel{x^2}}{x^2+y^2}\,\frac{x\dot{y} - y\dot{x}}{\cancel{x^2}} = 3/r^2 = 3/\cancel{x^2+y^2}$$

$$\Rightarrow \quad x\dot{y} - y\dot{x} = 3 \quad — (2)$$

$$\begin{aligned} x\dot{x} + y\dot{y} &= 0 \\ x\dot{y} - y\dot{x} &= 3 \end{aligned} \quad \Rightarrow \quad \begin{aligned} xy\dot{x} + y^2\dot{y} &= 0 \\ -xy\dot{x} + x^2\dot{y} &= 3x \\ \hline (x^2+y^2)\,\dot{y} &= x \end{aligned}$$

$$\dot{y} = \frac{3x}{x^2+y^2} \qquad \begin{aligned} x\dot{x} + y\dot{y} &= 0 \\ x\dot{x} + \frac{3xy}{x^2+y^2} &= 0. \end{aligned} \quad \Rightarrow \quad \dot{x} = \frac{-3y}{x^2+y^2}.$$

$$\dot{x} = \frac{dx}{dt} = \frac{-3y}{x^2+y^2}. \qquad \dot{y} = \frac{dy}{dt} = \frac{3x}{x^2+y^2}$$

$$\frac{x^{n+1} - x^n}{\Delta t} = \frac{-3y^n}{x^{n2}+y^{n2}} \qquad \frac{y^{n+1} - y^n}{\Delta t} = \frac{3x^n}{x^{n2}+y^{n2}}. \qquad \text{(Euler)}$$

$$x^{n+1} = x^n - \frac{3\Delta t\, y^n}{x^{n2}+y^{n2}}. \qquad y^{n+1} = y^n + \frac{3\Delta t\, x^n}{(x^n)^2+(y^n)^2}.$$

Predictor step :-

$$x^{*n+1} = x^n - \frac{3\Delta t\, y^n}{(x^n)^2+(y^n)^2}, \qquad y^{*n+1} = y^n + \frac{3\Delta t\, x^n}{(x^n)^2+(y^n)^2}.$$

corrector step :-

$$x^{n+1} = x^n + \frac{\Delta t}{2}\left[\frac{-3y^n}{(x^n)^2+(y^n)^2} + \frac{-3y^{*n+1}}{x^{*2}_{n+1}+y^{*2}_{n+1}}\right], \qquad y^{n+1} = y^n + \frac{\Delta t}{2}\left[\frac{3x^n}{(x^n)^2+(y^n)^2} + \frac{3x^{*n+1}}{x^{*2}_{n+1}+y^{*2}_{n+1}}\right]$$

(Improved Euler)

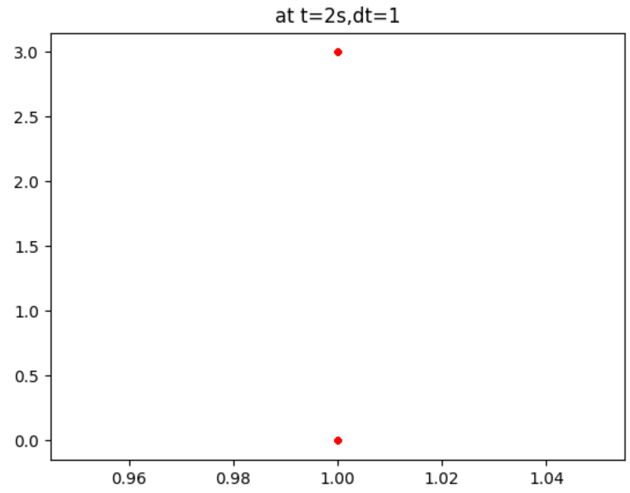
CODE FOR EULER METHOD IN CARTESIAN:-

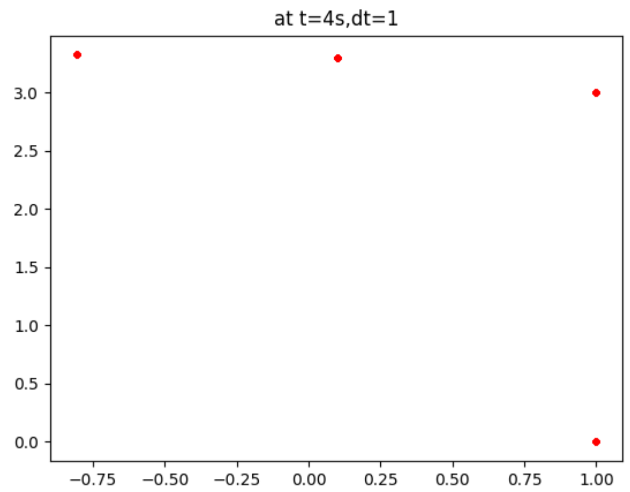
```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation

#Setting the cell

t=500
dt1=1
dt2=0.1
dt3=0.01
dt4=0.001
x_1=np.zeros(t)
x_2=np.zeros(t)
x_3=np.zeros(t)
x_4=np.zeros(t)
y_1=np.zeros(t)
y_2=np.zeros(t)
y_3=np.zeros(t)
y_4=np.zeros(t)

##INITIAL_VALUES

x_1[0]=1
x_2[0]=1
x_3[0]=1
x_4[0]=1
y_1[0]=0
y_2[0]=0
y_3[0]=0
y_4[0]=0

##CALCULATION

for n in range(0,t-1):
    x_1[n + 1] = x_1[n] - ((3 * dt1 * y_1[n]) /
```

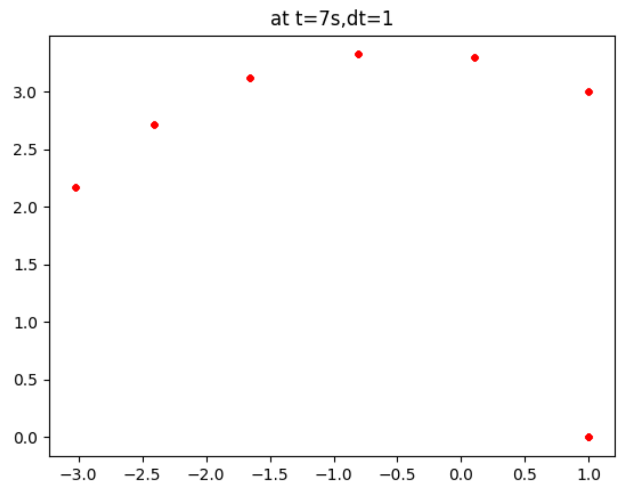
```
(x_1[n] ** 2 + y_1[n] ** 2))
    x_2[n + 1] = x_2[n] - ((3 * dt2 * y_2[n]) /
(x_2[n] ** 2 + y_2[n] ** 2))
    x_3[n + 1] = x_3[n] - ((3 * dt3 * y_3[n]) /
(x_3[n] ** 2 + y_3[n] ** 2))
    x_4[n + 1] = x_4[n] - ((3 * dt4 * y_4[n]) /
(x_4[n] ** 2 + y_4[n] ** 2))
    y_1[n + 1] = y_1[n] + ((3 * x_1[n] * dt1) /
(x_1[n] ** 2 + y_1[n] ** 2))
    y_2[n + 1] = y_2[n] + ((3 * x_2[n] * dt2) /
(x_2[n] ** 2 + y_2[n] ** 2))
    y_3[n + 1] = y_3[n] + ((3 * x_3[n] * dt3) /
(x_3[n] ** 2 + y_3[n] ** 2))
    y_4[n + 1] = y_4[n] + ((3 * x_4[n] * dt4) /
(x_4[n] ** 2 + y_4[n] ** 2))

##ANIMATING THE PLOT

fig =plt.figure(figsize=(12,9))
ax1=plt.subplot(221)
ax2=plt.subplot(222)
ax3=plt.subplot(223)
ax4=plt.subplot(224)
ax1.set_title('dt=1',color='r',loc='left')
ax2.set_title('dt=0.1',color='g',loc='left')
ax3.set_title('dt=0.01',color='b',loc='left')
ax4.set_title('dt=0.001',color='y',loc='left')
def animate(i):
    ax1.plot(x_1[i],y_1[i],'r.',label='dt=1')
    ax2.plot(x_2[i], y_2[i],
'g.',label='dt=0.1')
    ax3.plot(x_3[i], y_3[i],
'b.',label='dt=0.01')
    ax4.plot(x_4[i],
y_4[i],'y.',label='dt=0.001')
anim=animation.FuncAnimation(fig,animate,frames
=t,interval=100,blit=False)
plt.suptitle('Comparison plot for different dt
```

```
values- EULER(CARTESIAN)')
plt.show()
```

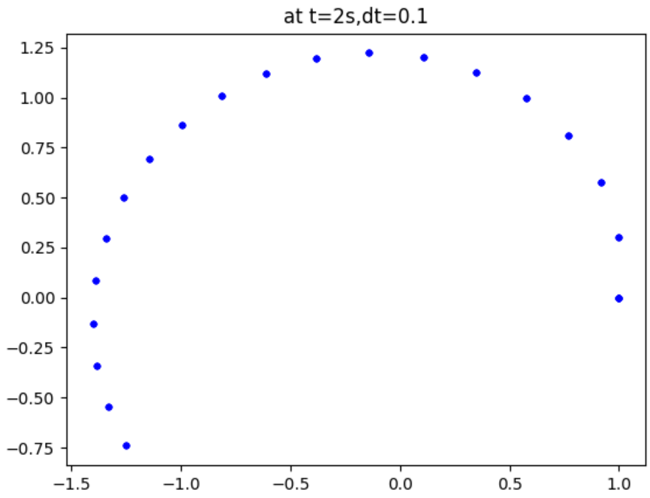
dt=1:-

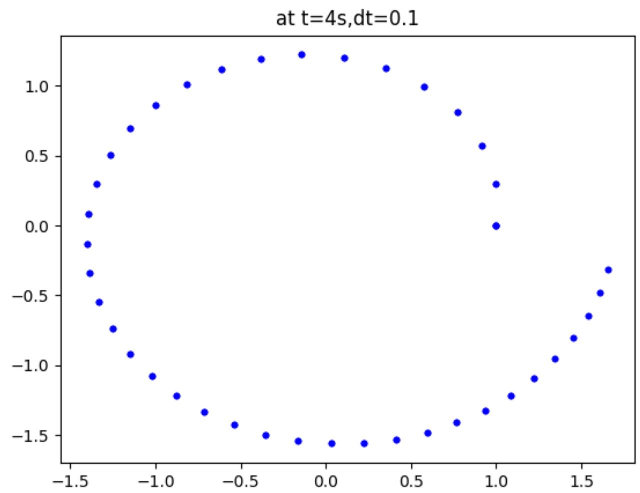

at t=2s,dt=1
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.96
0.98
1.00
1.02
1.04

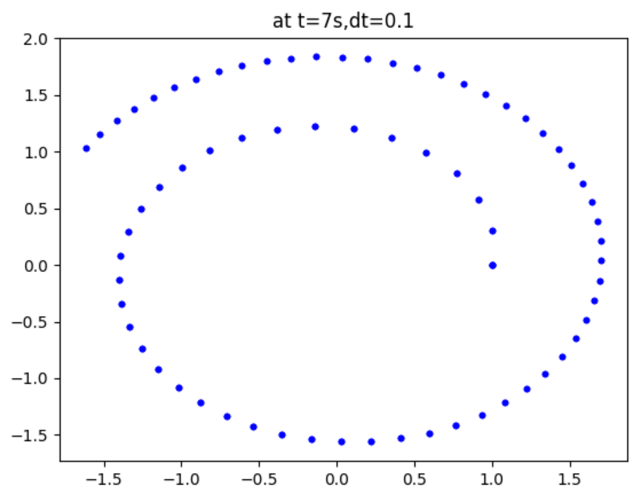

at t=4s,dt=1
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


at t=7s,dt=1
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0

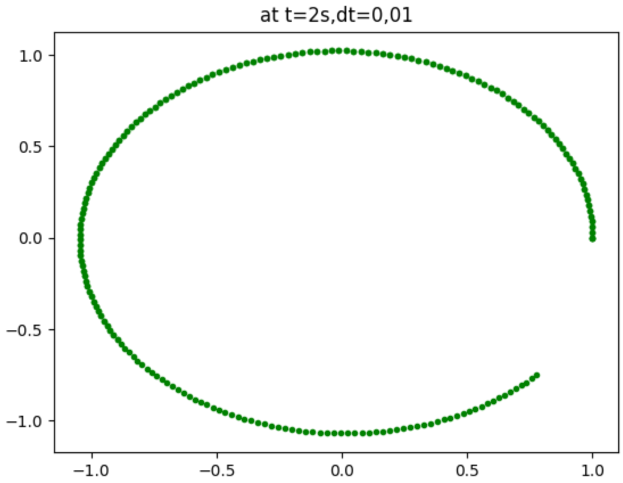
dt=0.1:-

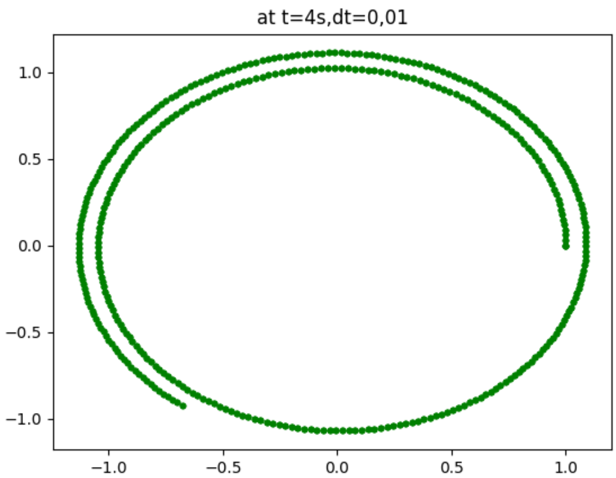

at t=2s,dt=0.1
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.5
−1.0
−0.5
0.0
0.5
1.0

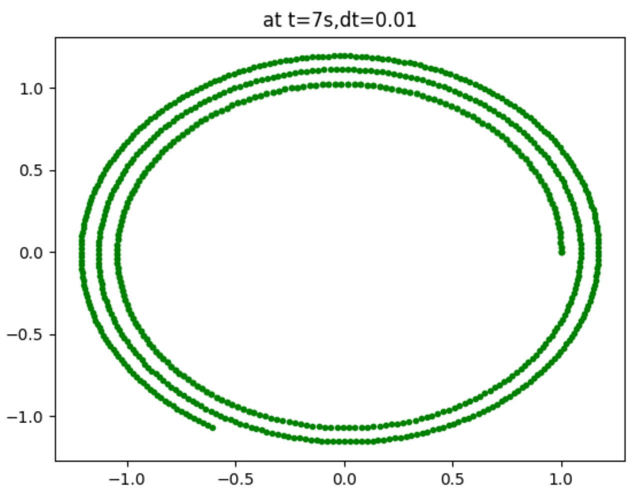

at t=4s,dt=0.1
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


at t=7s,dt=0.1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

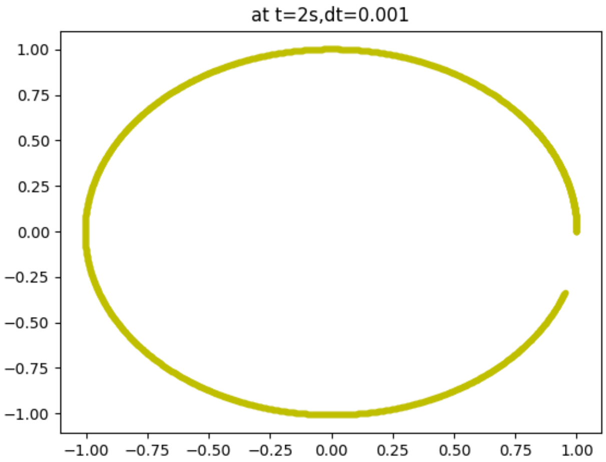
dt=0.01:-

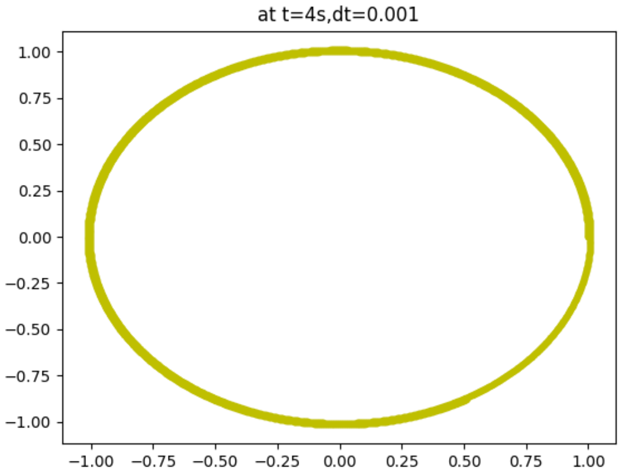

at t=2s,dt=0,01
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

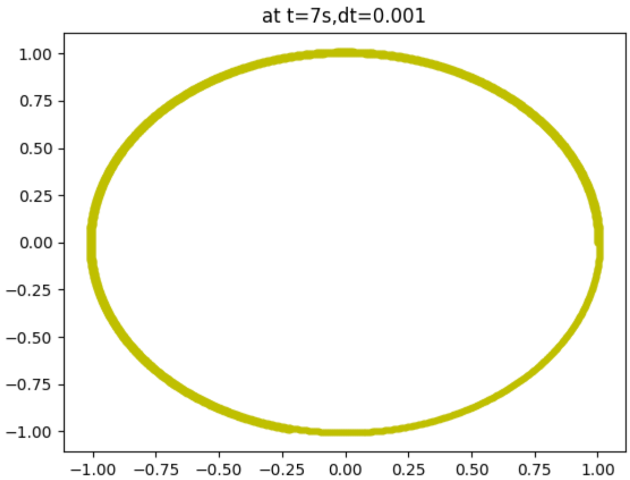

at t=4s,dt=0,01
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


at t=7s,dt=0.01
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

dt=0.001:-


at t=2s,dt=0.001
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


at t=4s,dt=0.001
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


at t=7s,dt=0.001
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

CODE FOR IMPROVED EULER IN CARTESIAN:-

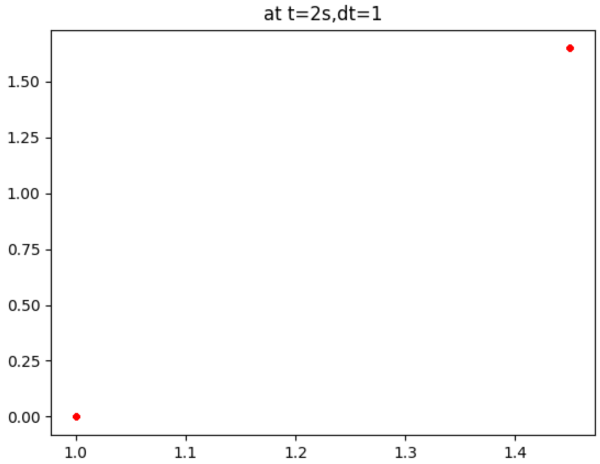
```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation

#Setting the cell

t=500
dt1=1
dt2=0.1
dt3=0.01
dt4=0.001
x_1=np.zeros(t)
x_2=np.zeros(t)
x_3=np.zeros(t)
x_4=np.zeros(t)
x_1_a=np.zeros(t)
x_2_a=np.zeros(t)
x_3_a=np.zeros(t)
x_4_a=np.zeros(t)
y_1_a=np.zeros(t)
y_2_a=np.zeros(t)
y_3_a=np.zeros(t)
y_4_a=np.zeros(t)
y_1=np.zeros(t)
y_2=np.zeros(t)
y_3=np.zeros(t)
y_4=np.zeros(t)

##INITIAL_VALUES

x_1[0]=1
x_2[0]=1
x_3[0]=1
x_4[0]=1
y_1[0]=0
```

```
y_2[0]=0
y_3[0]=0
y_4[0]=0

##CALCULATION

for n in range(0,t-1):

    #PREDICTOR STEP

  x_1_a[n + 1] = x_1[n] - ((3 * dt1 * y_1[n]) /
(x_1[n] ** 2 + y_1[n] ** 2))
  x_2_a[n + 1] = x_2[n] - ((3 * dt2 * y_2[n]) /
(x_2[n] ** 2 + y_2[n] ** 2))
  x_3_a[n + 1] = x_3[n] - ((3 * dt3 * y_3[n]) /
(x_3[n] ** 2 + y_3[n] ** 2))
  x_4_a[n + 1] = x_4[n] - ((3 * dt4 * y_4[n]) /
(x_4[n] ** 2 + y_4[n] ** 2))
  y_1_a[n + 1] = y_1[n] - ((3 * dt1 * x_1[n]) /
(x_1[n] ** 2 + y_1[n] ** 2))
  y_2_a[n + 1] = y_2[n] - ((3 * dt2 * x_2[n]) /
(x_2[n] ** 2 + y_2[n] ** 2))
  y_3_a[n + 1] = y_3[n] - ((3 * dt3 * x_3[n]) /
(x_3[n] ** 2 + y_3[n] ** 2))
  y_4_a[n + 1] = y_4[n] - ((3 * dt4 * x_4[n]) /
(x_4[n] ** 2 + y_4[n] ** 2))


    #CORRECTOR STEP

  x_1[n + 1] = x_1[n] - (3 * 0.5 * dt1) *
((y_1[n] / (x_1[n] ** 2 + y_1[n] ** 2)) +
(y_1_a[n + 1] / (x_1_a[n + 1] ** 2 + y_1_a[n +
1] ** 2)))
  x_2[n + 1] = x_2[n] - (3 * 0.5 * dt2) *
((y_2[n] / (x_2[n] ** 2 + y_2[n] ** 2)) +
(y_2_a[n + 1] / (x_2_a[n + 1] ** 2 + y_2_a[n +
1] ** 2)))
```

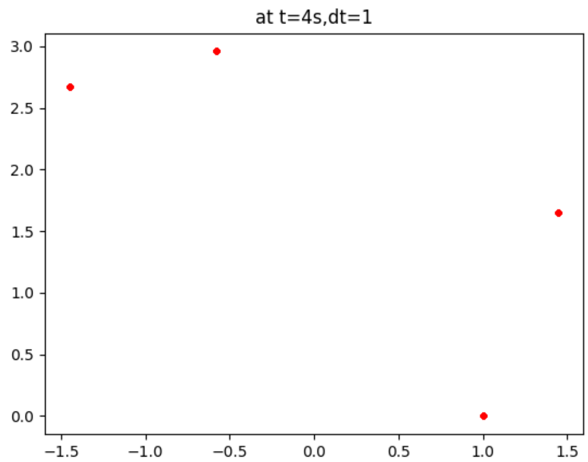
```
  x_3[n + 1] = x_3[n] - (3 * 0.5 * dt3) *
((y_3[n] / (x_3[n] ** 2 + y_3[n] ** 2)) +
(y_3_a[n + 1] / (x_3_a[n + 1] ** 2 + y_3_a[n +
1] ** 2)))
  x_4[n + 1] = x_4[n] - (3 * 0.5 * dt4) *
((y_4[n] / (x_4[n] ** 2 + y_4[n] ** 2)) +
(y_4_a[n + 1] / (x_4_a[n + 1] ** 2 + y_4_a[n +
1] ** 2)))
  y_1[n + 1] = y_1[n] + (3 * 0.5 * dt1) *
((x_1[n] / (x_1[n] ** 2 + y_1[n] ** 2)) +
(x_1_a[n + 1] / (x_1_a[n + 1] ** 2 + y_1_a[n +
1] ** 2)))
  y_2[n + 1] = y_2[n] + (3 * 0.5 * dt2) *
((x_2[n] / (x_2[n] ** 2 + y_2[n] ** 2)) +
(x_2_a[n + 1] / (x_2_a[n + 1] ** 2 + y_2_a[n +
1] ** 2)))
  y_3[n + 1] = y_3[n] + (3 * 0.5 * dt3) *
((x_3[n] / (x_3[n] ** 2 + y_3[n] ** 2)) +
(x_3_a[n + 1] / (x_3_a[n + 1] ** 2 + y_3_a[n +
1] ** 2)))
  y_4[n + 1] = y_4[n] + (3 * 0.5 * dt4) *
((x_4[n] / (x_4[n] ** 2 + y_4[n] ** 2)) +
(x_4_a[n + 1] / (x_4_a[n + 1] ** 2 + y_4_a[n +
1] ** 2)))



##ANIMATING THE PLOT

fig =plt.figure(figsize=(12,9))
ax1=plt.subplot(221)
ax2=plt.subplot(222)
ax3=plt.subplot(223)
ax4=plt.subplot(224)
ax1.set_title('dt=1',color='r',loc='left')
ax2.set_title('dt=0.1',color='g',loc='left')
ax3.set_title('dt=0.01',color='b',loc='left')
ax4.set_title('dt=0.001',color='y',loc='left')
```

```
def animate(i):
     ax1.plot(x_1[i], y_1[i], 'r.',label='dt=1')
     ax2.plot(x_2[i], y_2[i],
'g.',label='dt=0.1')
     ax3.plot(x_3[i], y_3[i],
'b.',label='dt=0.01')
     ax4.plot(x_4[i], y_4[i],
'y.',label='dt=0.001')
anim=animation.FuncAnimation(fig,animate,frames
=t,interval=100,blit=False)
plt.suptitle('Comparison plot for different dt
values-IMPROVED EULER(CARTESIAN)')
plt.show()
```

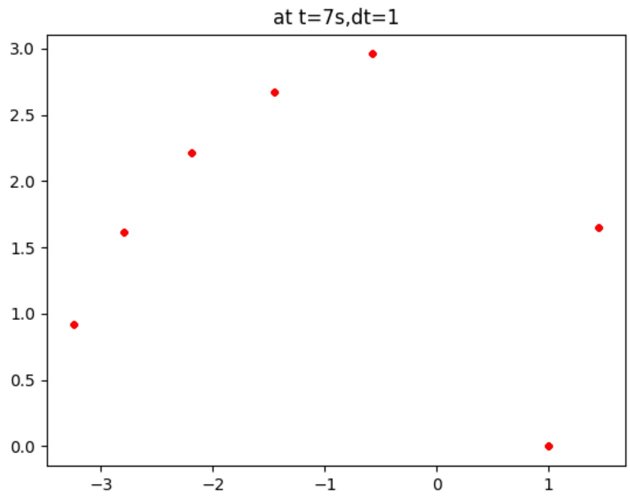
dt=1:-



at t=2s,dt=1
1.50
1.25
1.00
0.75
0.50
0.25
0.00
1.0
1.1
1.2
1.3
1.4

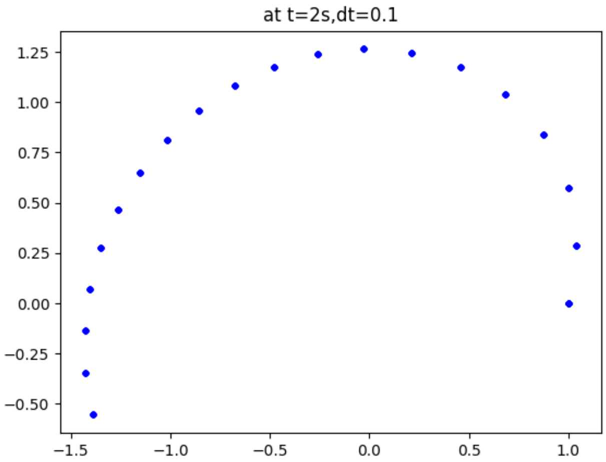


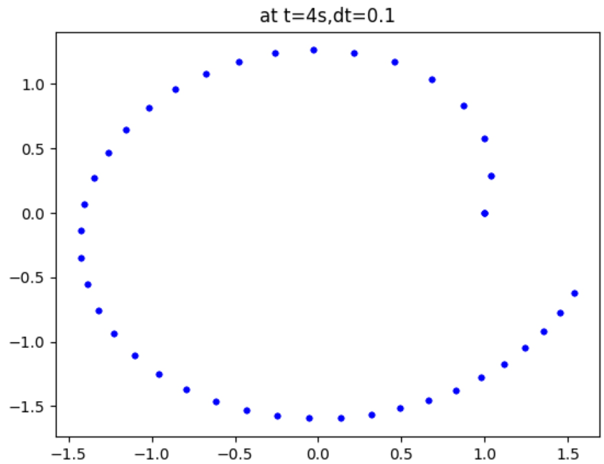
at t=4s,dt=1
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

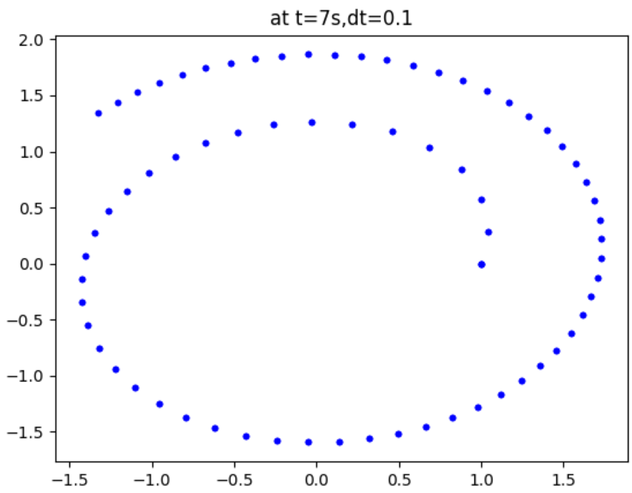


at t=7s,dt=1
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−3
−2
−1
0
1

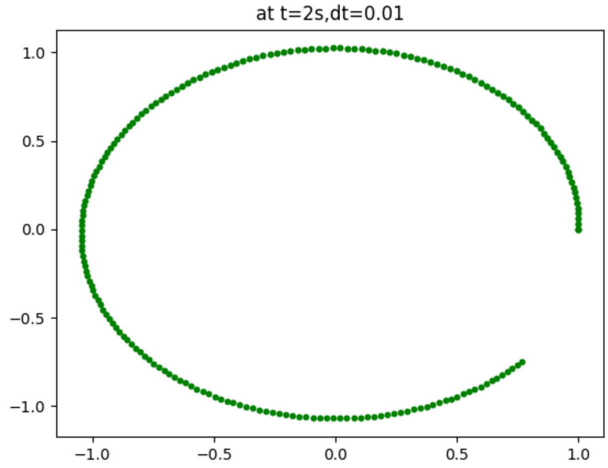
dt=0.1:-



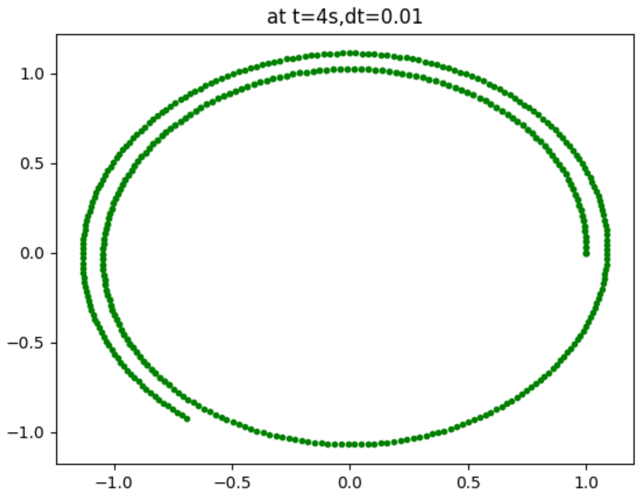




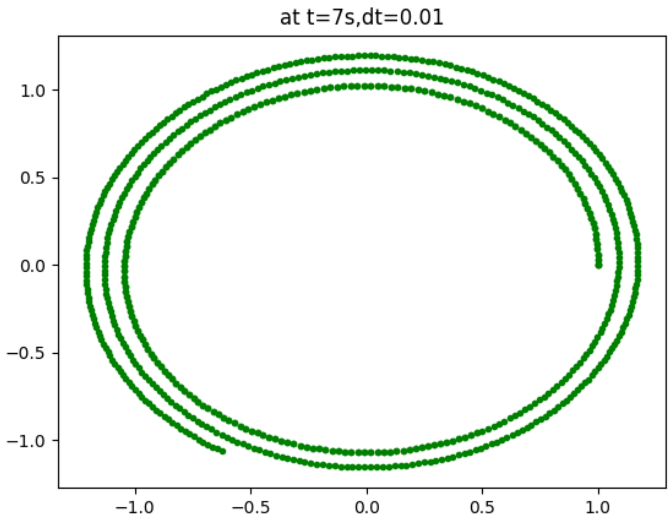
dt=0.01:-



at t=2s,dt=0.01
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

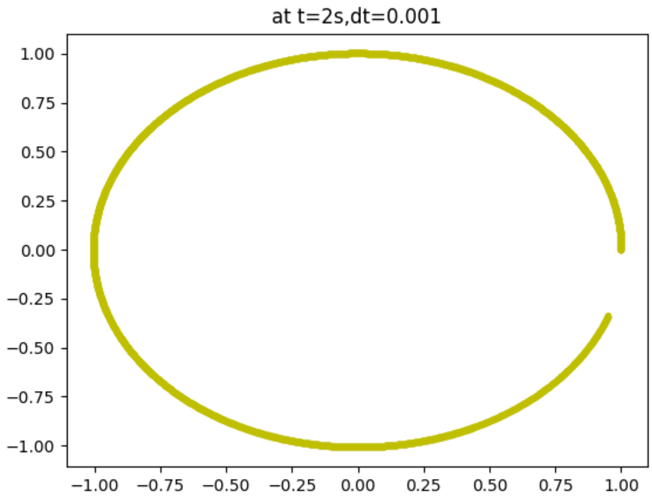


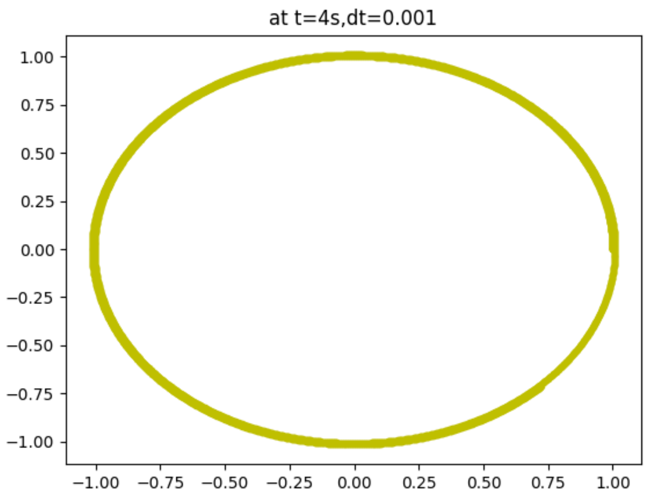
at t=4s,dt=0.01
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

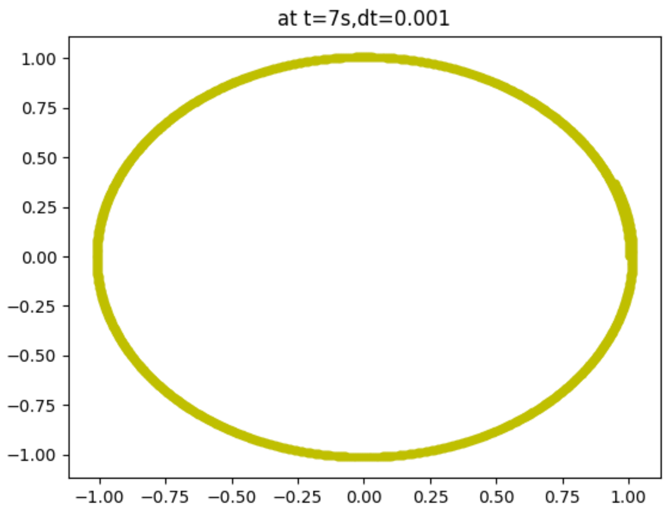


at t=7s,dt=0.01
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

dt=0.001:-


at t=2s,dt=0.001
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


at t=4s,dt=0.001
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


at t=7s,dt=0.001
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

for cartesian coordinates it can be seen that for each rotation of particle r is getting increased slightly as opposed to constant 'r' value in exact solution. This is due to approximation error in calculating '$r = \sqrt{x^2+y^2}$' in each point. It can be seen that with smaller dt values r value almost remains constant.

As for the comparison of Euler and Improved Euler the accuracy of plotting can be seen higher for Improved Euler as compared to Euler since for improved euler r values remains almost constant for same dt values.

c> Write a subroutine to solve a linear equation system : $Ax=b$. Check it out by solving for a test problem and submit the test only.

Ans For the test case used example is :-

$$x + y + z + \omega = 13$$
$$2x + 3y + 0z + \omega = -1$$
$$-3x + 4y + z + 2 = 10$$
$$x + 2y - z + 1 = 1.$$

Solution is obtained with the help of numpy library.

obtained solution is :- $x = 2$
$y = 0$
$z = 6$
$\omega = 5$

CODE FOR TEST CASE:-

```
A=[[1,1,1,1],[2,3,0,-1],[-3,4,1,2],[1,2,-1,1]]
B=[[13],[-1],[10],[1]]

from Solving_For_X import linear_eqn_solver

x=linear_eqn_solver(3,A,B)
     ##using numpy for solving the system of
equations

print('SOLVED X VALUES FOR TEST CASE:')

print(x)
```

OUTPUT OF TEST CASE:-

```
##AKSHAY J
##21105012
#TEST CASE
A=[[1,1,1,1],[2,3,0,-1],[-3,4,1,2],[1,2,-1,1]]
B=[[13],[-1],[10],[1]]
from Solving_For_X import linear_eqn_solver
x=linear_eqn_solver(3,A,B)      ##using numpy for solving the system of equations
print('SOLVED X VALUES FOR TEST CASE:')
print(x)
```

test2 × Test_Qn ×

```
"D:\Python Scripts\pythonProject10\venv\Scripts\python.exe" "D:/Python Scripts/pythonProject10/Test_Qn.py"
SOLVED X VALUES FOR TEST CASE:
[[2.]
 [0.]
 [6.]
 [5.]]

Process finished with exit code 0
```