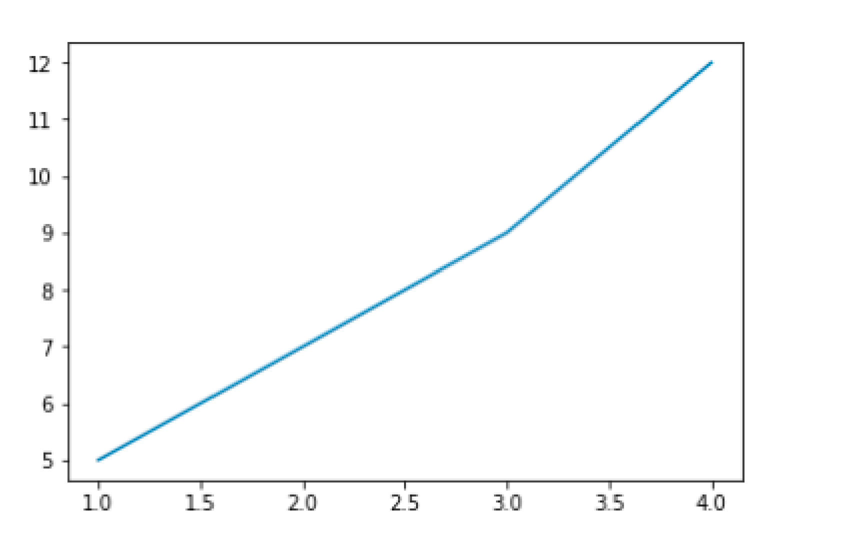
Python for Physics
Lab -5

Matplotlib

```
In [706]: import matplotlib.pyplot as plt

          x = np.array([1,2,3,4])
          y = np.array([5,7,9,12])

          plt.plot(x,y)
          plt.show()
```



Waves and Oscillations

Simple Harmonic Motion

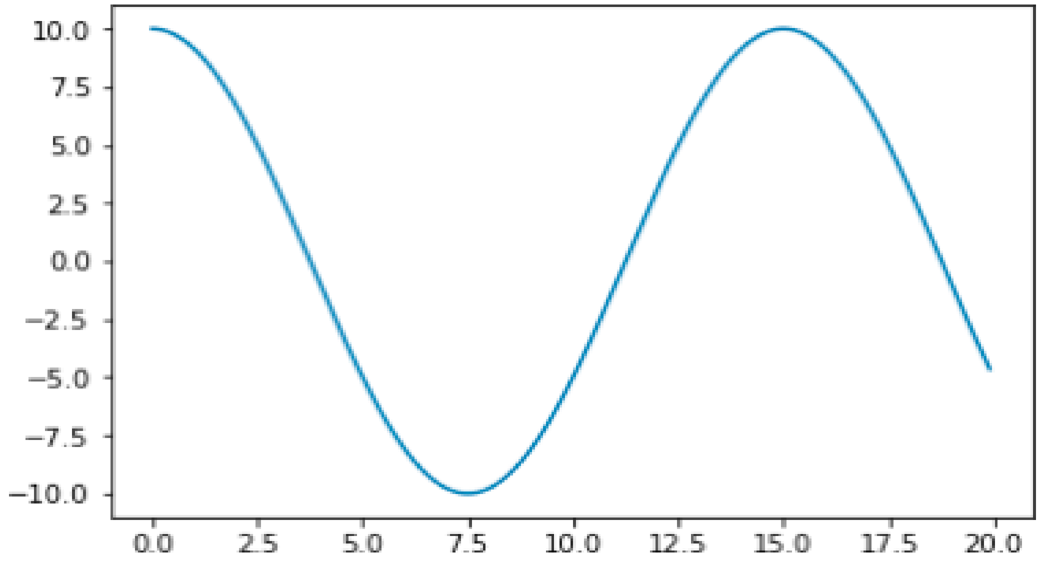
Displacement :

$$x = x_m Cos(\omega t + \theta)$$

- x_m is the amplitude (maximum displacement of the system)
- t is the time
- w is the angular frequency, and
- θ is the phase constant or phase

In [577]:
```
xm = 10
w = 24
phase = 0
t= np.arange(0,20, 0.1)
x = xm*np.cos(np.radians(w*t- phase))
plt.plot(t,x)
```

Out[577]:
```
[<matplotlib.lines.Line2D at 0x253ccaa3fd0>]
```



```
xm = 10
```

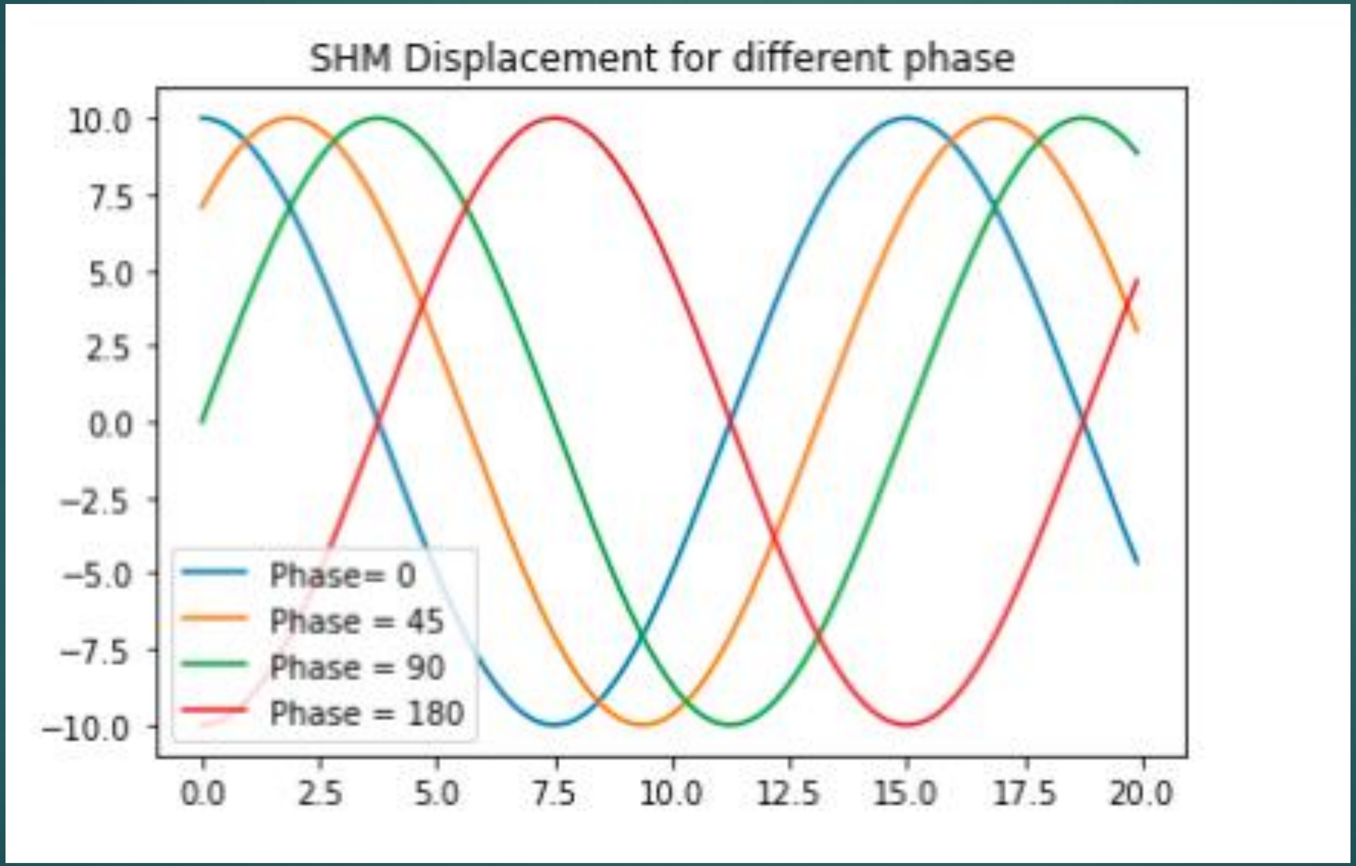
```
w = 24
phase = 0
t= np.arange(0,20, 0.1)
x = xm*np.cos(np.radians(w*t- phase))

phase1 = 45
x1 = xm*np.cos(np.radians(w*t- phase1))

phase2 = 90
x2 = xm*np.cos(np.radians(w*t- phase2))

phase3 = 180
x3 = xm*np.cos(np.radians(w*t- phase3))

ax = plt.subplot(111)
ax.plot(t,x, label='Phase= 0')
ax.plot(t,x1, label='Phase = 45')
ax.plot(t,x2, label='Phase = 90')
ax.plot(t,x3, label='Phase = 180')
plt.title('SHM Displacement for different phase ')
ax.legend()
plt.show()
```


SHM Displacement for different phase
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Phase= 0
Phase = 45
Phase = 90
Phase = 180

Velocity :

$$V = -x_m \omega Sin(\omega t + \theta)$$

Acceleration :

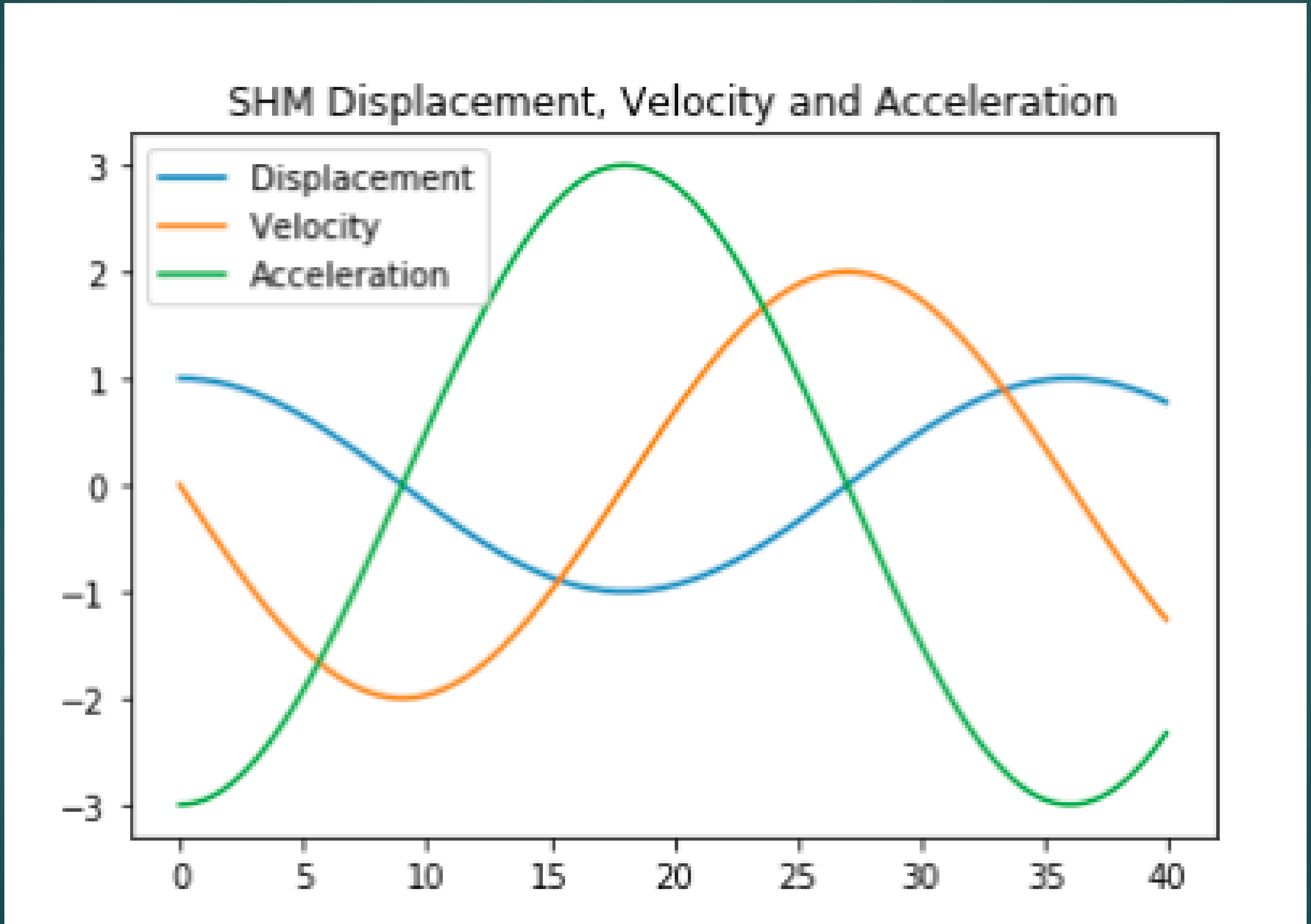
$$a = -x_m \omega^2 Sin(\omega t + \theta)$$

In [579]:

```
xm = 1
w = 10
vm= 2
am =3
phase = 0
t= np.arange(0,40, 0.1)
x = xm*np.cos(np.radians(w*t- phase))
v =- vm* np.sin(np.radians(w*t- phase))
a =- am* np.cos(np.radians(w*t- phase))
#v =- xm*w* np.sin(np.radians(w*t- phase))
#a =- xm*w**2* np.cos(np.radians(w*t- phase))

ax = plt.subplot(111)
ax.plot(t,x, label='Displacement')
ax.plot(t,v, label='Velocity ')
ax.plot(t,a, label='Acceleration')

plt.title('SHM Displacement, Velocity and Acceleration')
ax.legend()
plt.show()
```



SHM Displacement, Velocity and Acceleration
Displacement
Velocity
Acceleration
3
2
1
0
−1
−2
−3
0
5
10
15
20
25
30
35
40

SHM as Circular Motion

Simple Harmonic Motion as Circular Motion

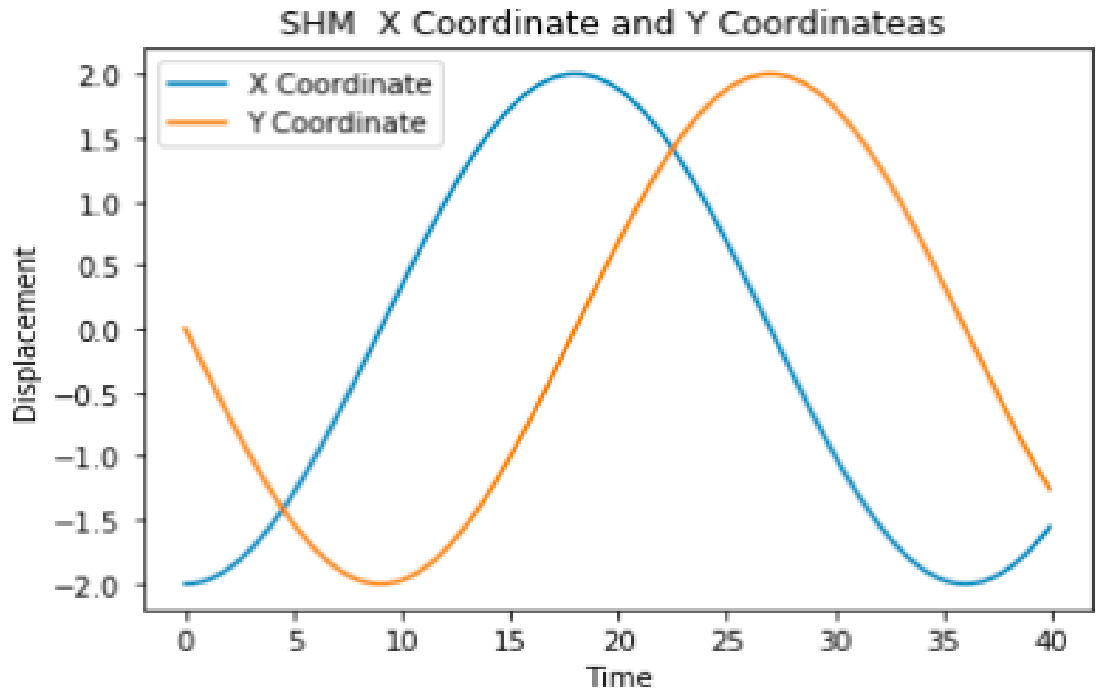
$$x = x_m \omega Cos(\omega t + \theta)$$

$$y = y_m \omega Sin(\omega t + \theta)$$

In [584]:
```
w = 10
xm= 2
ym =2
phase = 180
```

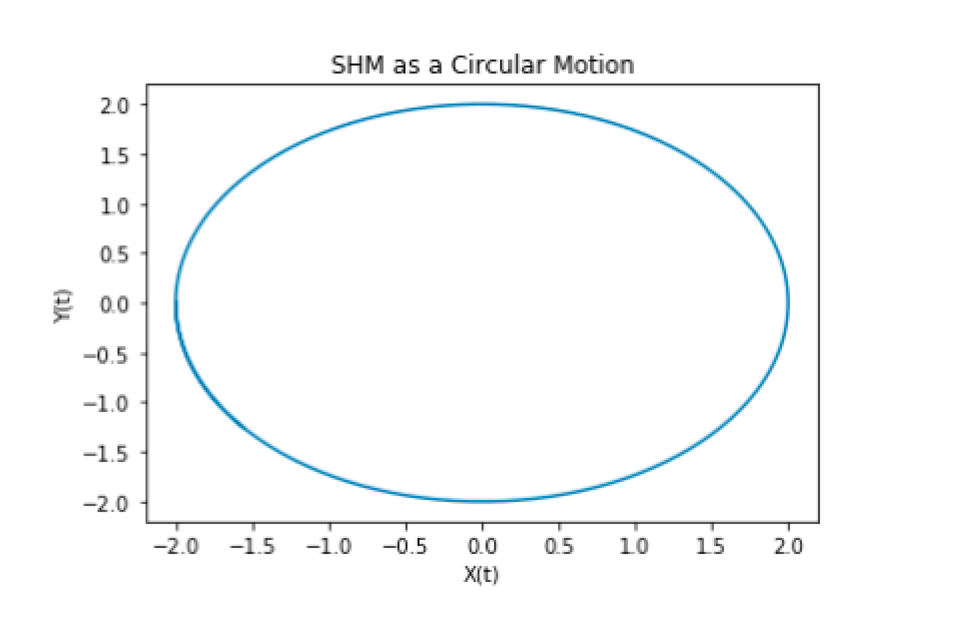
```
t= np.arange(0,40, 0.1)
x = xm*np.cos(np.radians(w*t- phase))
y = ym*np.sin(np.radians(w*t- phase))

ax = plt.subplot(111)
ax.plot(t,x, label='X Coordinate')
ax.plot(t,y, label='Y Coordinate')
plt.title('SHM  X Coordinate and Y Coordinateas ')
plt.xlabel('Time')
plt.ylabel('Displacement')
ax.legend()
plt.show()
```



```
In [585]: plt.plot(x,y)
          plt.xlabel('X(t)')
          plt.ylabel('Y(t)')
          plt.title('SHM as a Circular Motion')

Out[585]: Text(0.5,1,'SHM as a Circular Motion')
```



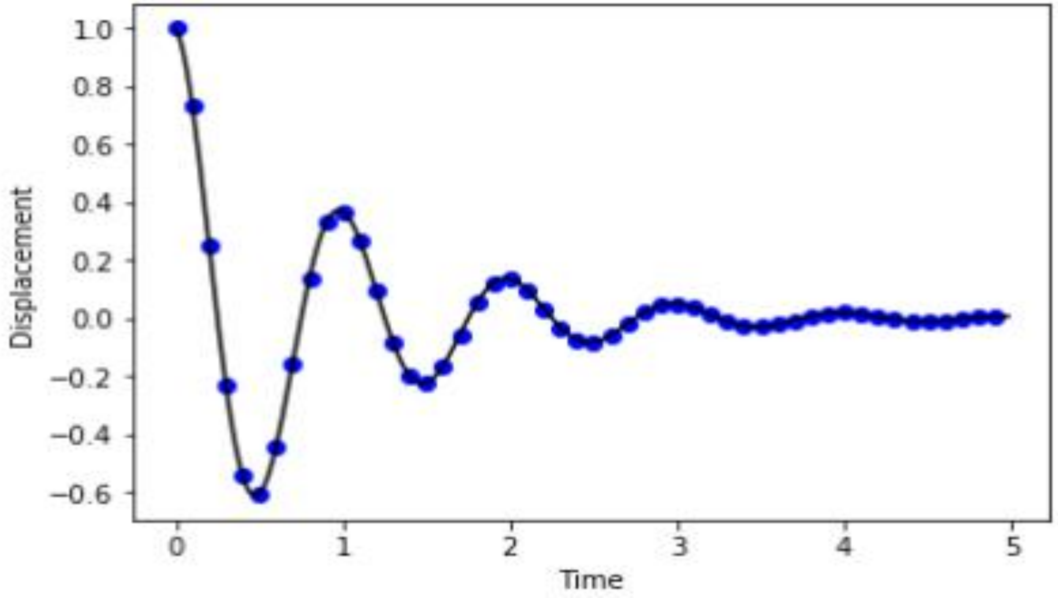
Damped Oscillation

```
In [586]: def f(t):
              return np.exp(-t) * np.cos(2*np.pi*t)

          t1 = np.arange(0.0, 5.0, 0.1)
          t2 = np.arange(0.0, 5.0, 0.02)

          plt.figure()
          plt.subplot(111)
          plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
          plt.xlabel('Time')
          plt.ylabel('Displacement')

Out[586]: Text(0,0.5,'Displacement')
```




```
In [587]: def f(t):
              b = 4
              m = 10
              w = 2*np.pi
              p=0
              return np.exp(-2*b*t/m) * np.cos(w*t - p)     # w = srtq(k/m)

          t1 = np.arange(0.0, 5.0, 0.1)
          t2 = np.arange(0.0, 5.0, 0.02)

          plt.figure()
          plt.subplot(111)
          plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
          plt.xlabel('Time')
          plt.ylabel('Displacement')

Out[587]: Text(0,0.5,'Displacement')
```

Conditions for different types of Damping

```
b = 6
m = 5
w = np.pi*2


if int(b) == int(2*m*w):
    print (" This Critical Under Damped condition:",int(b),'=', int(2*m
*w ))

elif int(b) <= int(2*m*w):
    print (" This is Under Damped condition:", int(b),'<',int(2*m*w) )

elif int(b) >= int(2*m*w):
    print (" This is Over Dapmed condition: ", int(b),'>', int(2*m*w) )
```

```
 This is Under Damped condition: 6 < 62
```

Underdamped, critical damped and over damped

```
def f(t,b,m,w,p):

    if int(b) == int(2*m*w):
        print (" This Critical Under Damped condition:",int(b),'=', int
(2*m*w ))

    elif int(b) <= int(2*m*w):
        print (" This is Under Damped condition:", int(b),'<',int(2*m*w
) )

    elif int(b) >= int(2*m*w):
        print (" This is Over Dapmed condition: ", int(b),'>', int(2*m*
w) )

    x = np.exp(-2*b*t/m) * (np.cos(w*t - p) )

    return x    # w = srtq(k/m)

t1 = np.arange(0.0, 5.0, 0.01)
t2 = np.arange(0.0, 5.0, 0.02)


plt.figure()
plt.subplot(111)
plt.plot(t1, f(t1,89,5,2*np.pi,90), 'k')
plt.xlabel('Time')
plt.ylabel('Displacement')
```

```
This is Over Dapmed condition:  89 > 62

Out[740]: Text(0,0.5,'Displacement')
```