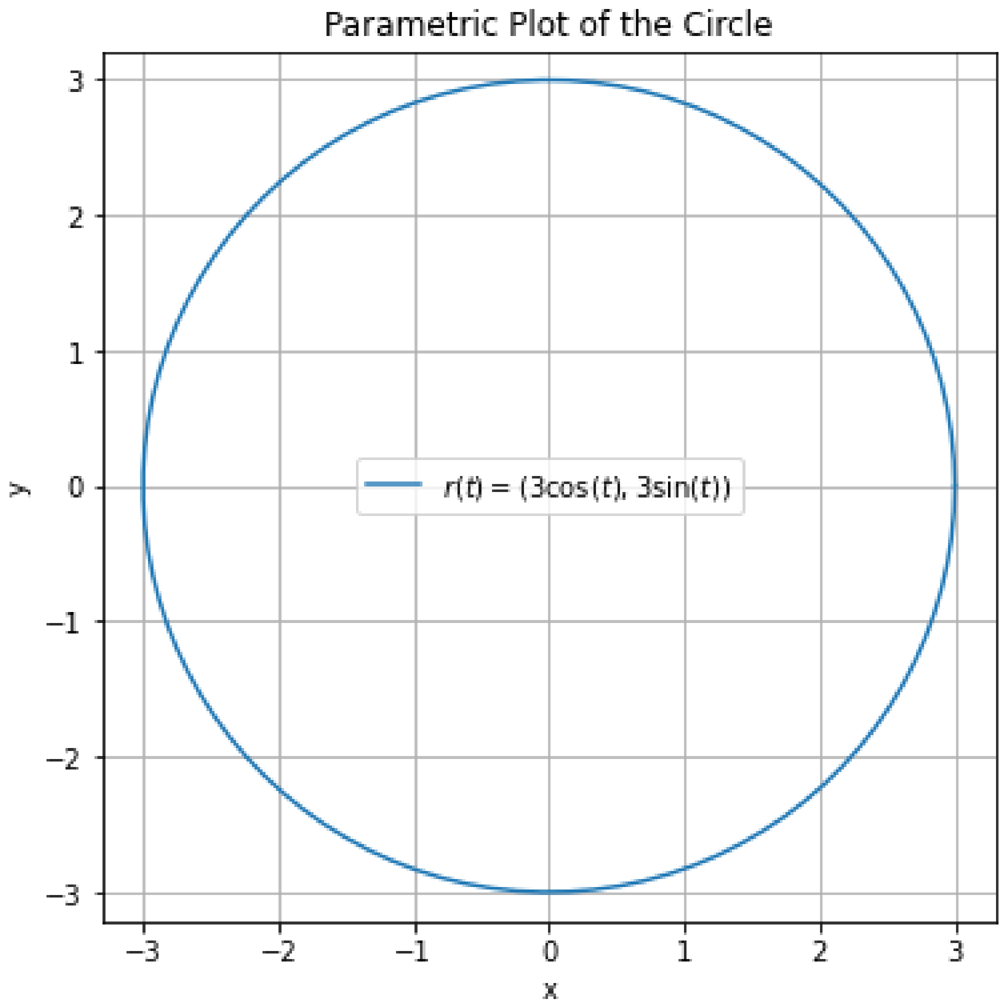
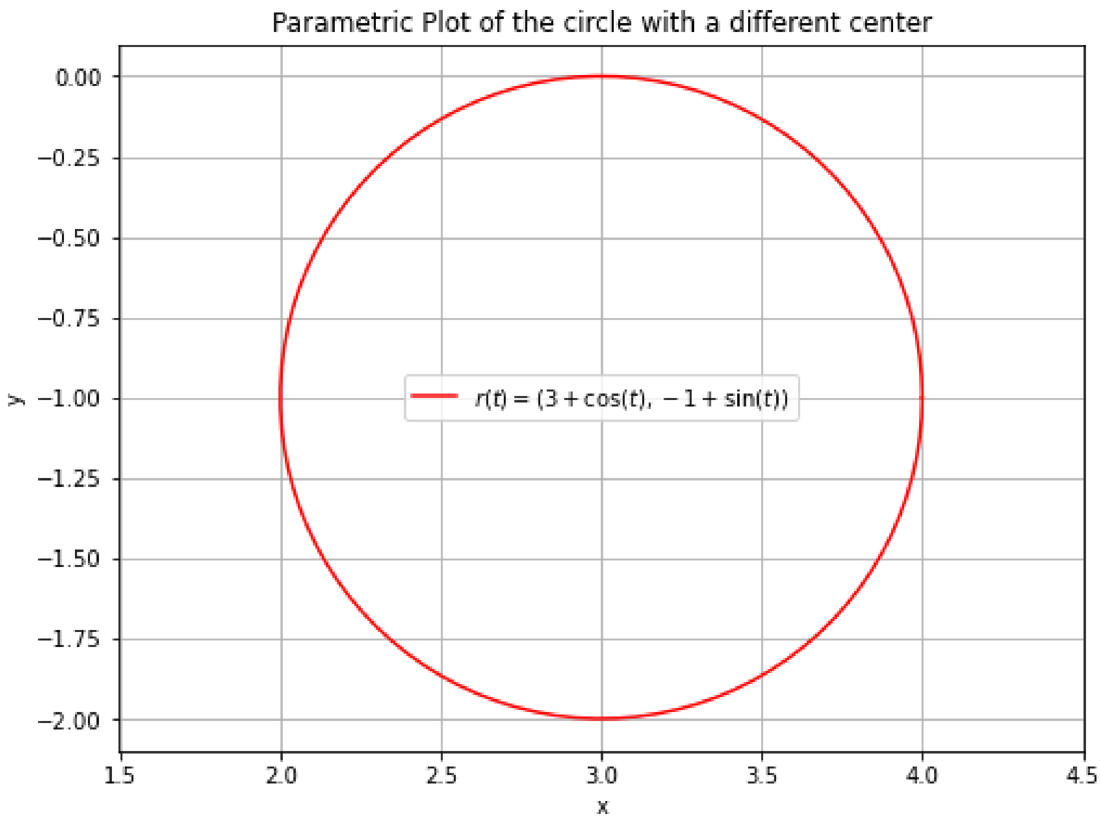
CG_Curs7

April 23, 2024

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the parameter t from 0 to 2*pi
t = np.linspace(0, 2*np.pi, 100)

# Parametric equations
x = 3 * np.cos(t)
y = 3 * np.sin(t)

# Create the plot
plt.figure(figsize=(6,6))
plt.plot(x, y, label=r'$r(t) = (3\cos(t), 3\sin(t))$')
plt.title('Parametric Plot of the Circle')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.axis('equal')  # Ensure the aspect ratio makes the circle look like a circle
plt.legend()
plt.show()
```

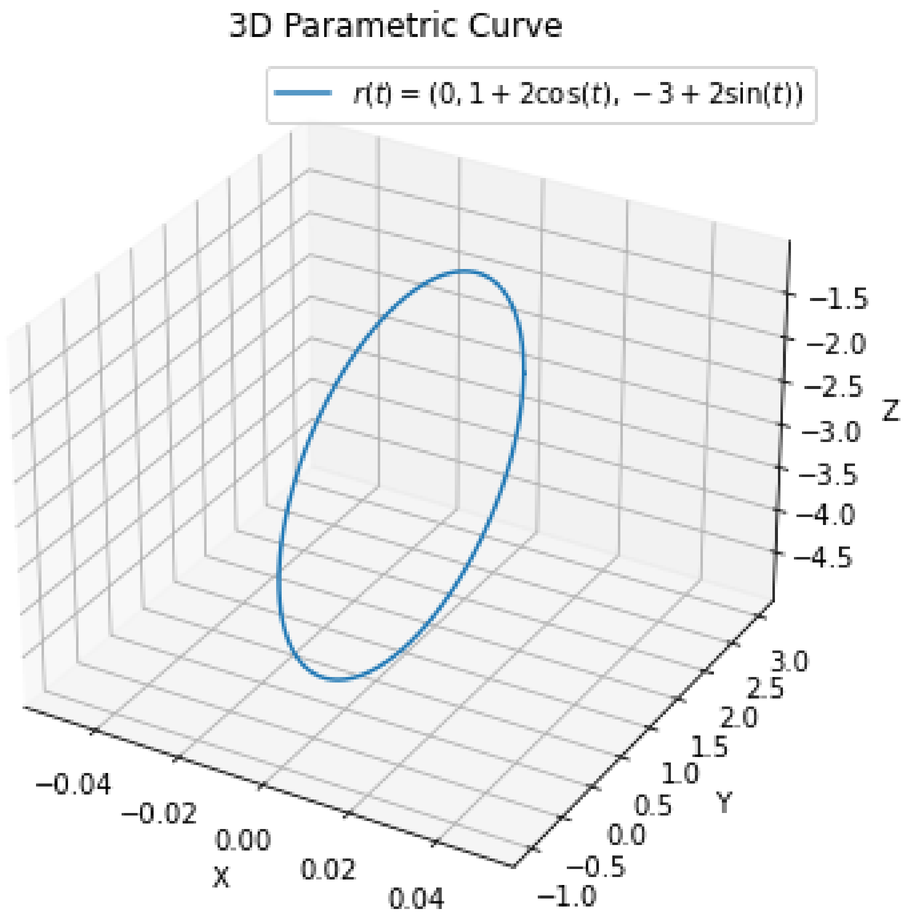


[2]:
```
# With different center
x = 3 + np.cos(t)
y = -1 + np.sin(t)

# Create the plot for the circle
plt.figure(figsize=(8,6))
plt.plot(x, y, label=r'$r(t) = (3+\cos(t), -1+\sin(t))$', color='red')
plt.title('Parametric Plot of the circle with a different center')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.axis('equal')  # Ensure the aspect ratio is correct
plt.legend()
plt.show()
```



[3]:
```python
from mpl_toolkits.mplot3d import Axes3D  # Import the 3D plotting tool

# Define the parameter t from 0 to 2*pi
t = np.linspace(0, 2*np.pi, 100)

# Parametric equations for the 3D curve
x = 0 * t  # x is 0 for all t
y = 1 + 2 * np.cos(t)
z = -3 + 2 * np.sin(t)

# Create the 3D plot
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')
ax.plot(x, y, z, label=r'$r(t) = (0, 1+2\cos(t), -3+2\sin(t))$')
ax.set_title('3D Parametric Curve')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.legend()
# Set view angle
```

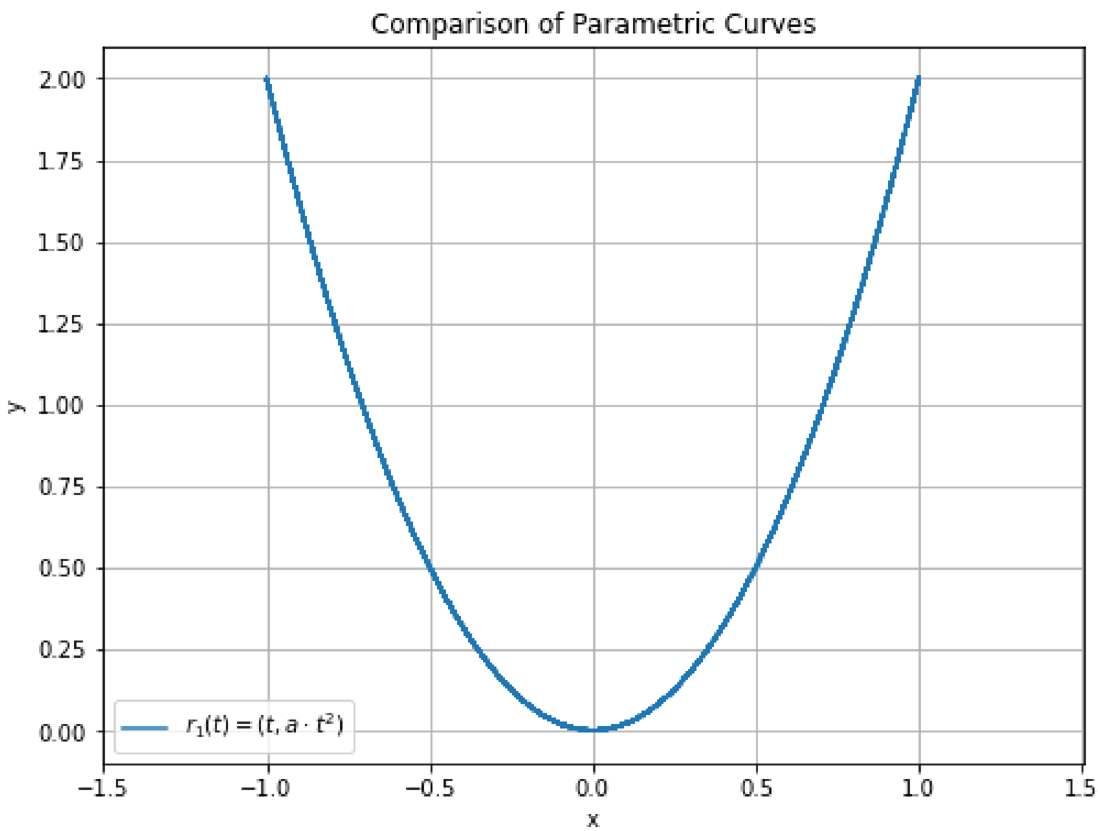
```
#ax.view_init(elev=20, azim=60)
plt.show()
```



[4]:
```
# start with the parameterization of a circle and obtain the parameterization␣
↪of an ellipse
```

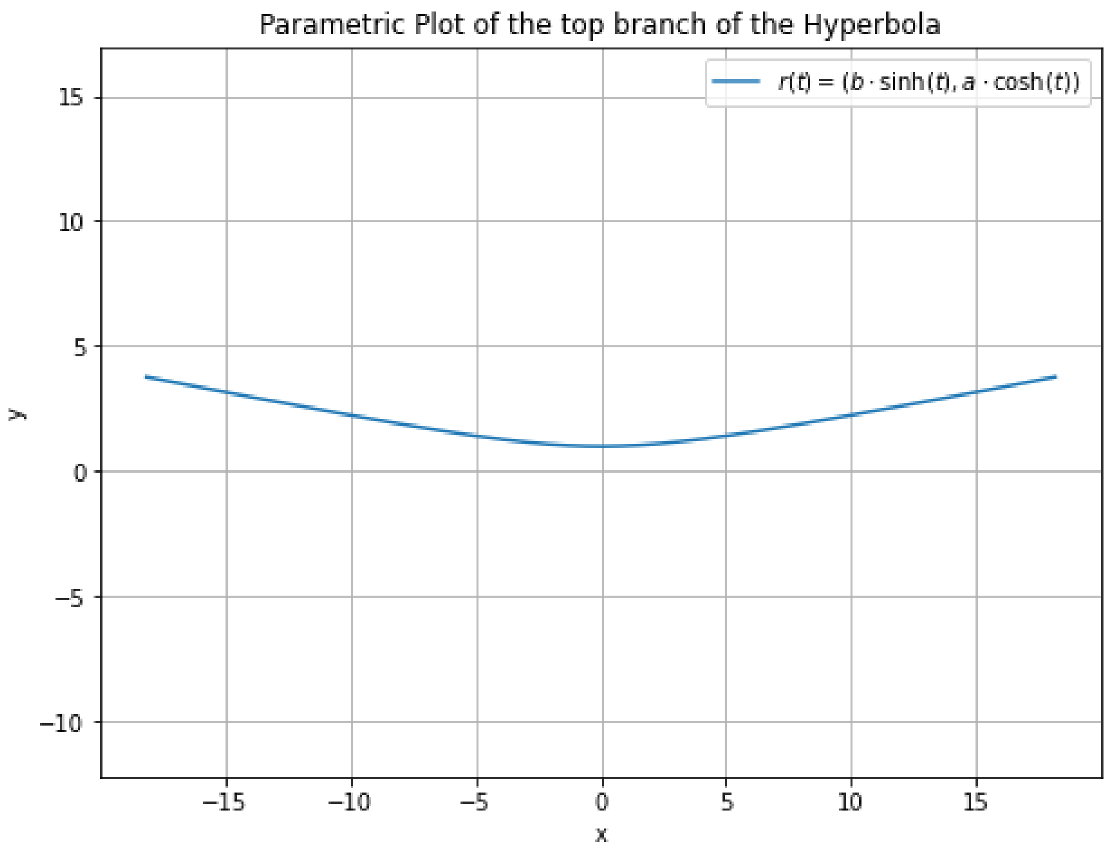
[5]:
```
# Define a fixed real number a
a = 2  # Change this value as needed

# Define the parameter t within a reasonable range
t = np.linspace(-100, 100, 4000)

# Parametric equations for the curves
x1 = np.sin(t)
y1 = a * (np.sin(t))**2

x2 = t**3
```

```
y2 = a * t**6

# Create the plot
plt.figure(figsize=(8, 6))
plt.plot(x1, y1, label=r'$r_1(t) = (t, a \cdot t^2)$')
#plt.plot(x2, y2, label=r'$r_2(t) = (t^3, a \cdot t^6)$', linestyle='--')
plt.title('Comparison of Parametric Curves')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.axis('equal')
plt.legend()
plt.show()
```



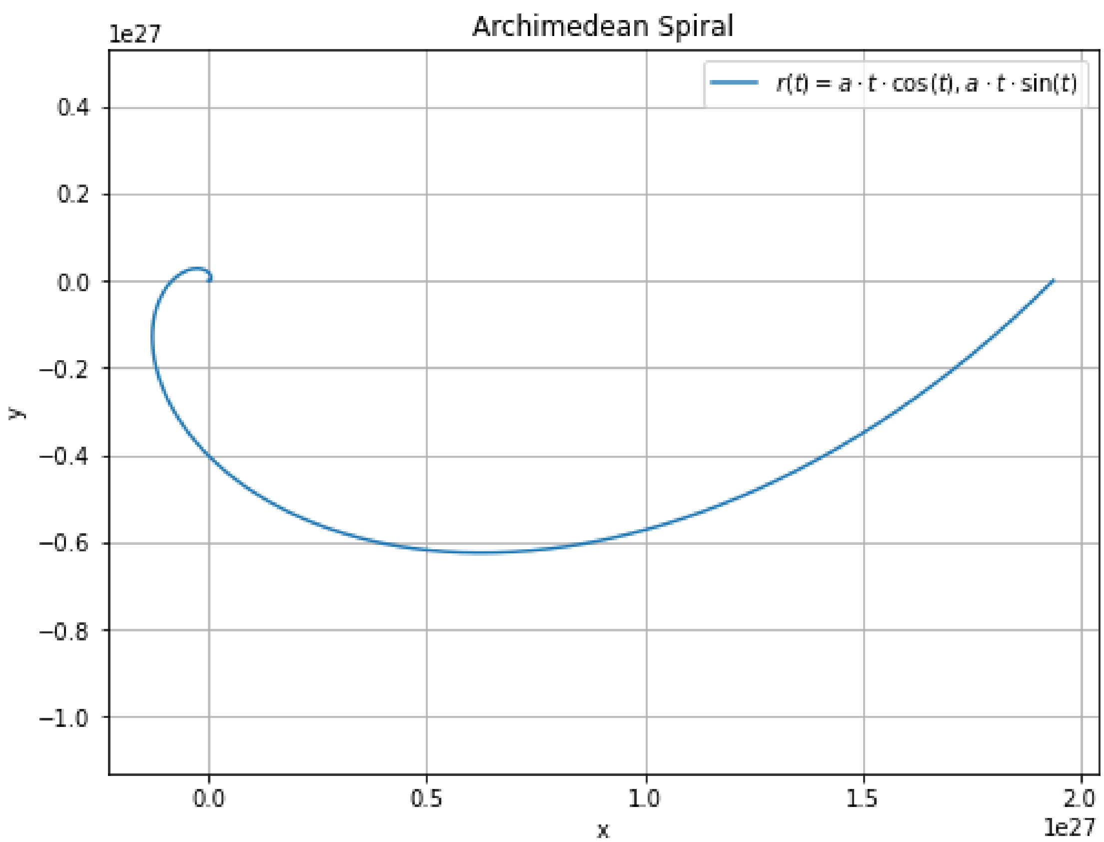
[6]:
```
# Define the positive real numbers a and b
a = 1  # Change this value as needed
b = 5  # Change this value as needed

# Define the parameter t within a reasonable range
t = np.linspace(-2, 2, 400)
```

```
# Parametric equations for the curve
x = b * np.sinh(t)
y = a * np.cosh(t)

# Create the plot
plt.figure(figsize=(8, 6))
plt.plot(x, y, label=r'$r(t) = (b \cdot \sinh(t), a \cdot \cosh(t))$')
plt.title('Parametric Plot of the top branch of the Hyperbola')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.axis('equal')
plt.legend()
plt.show()
```



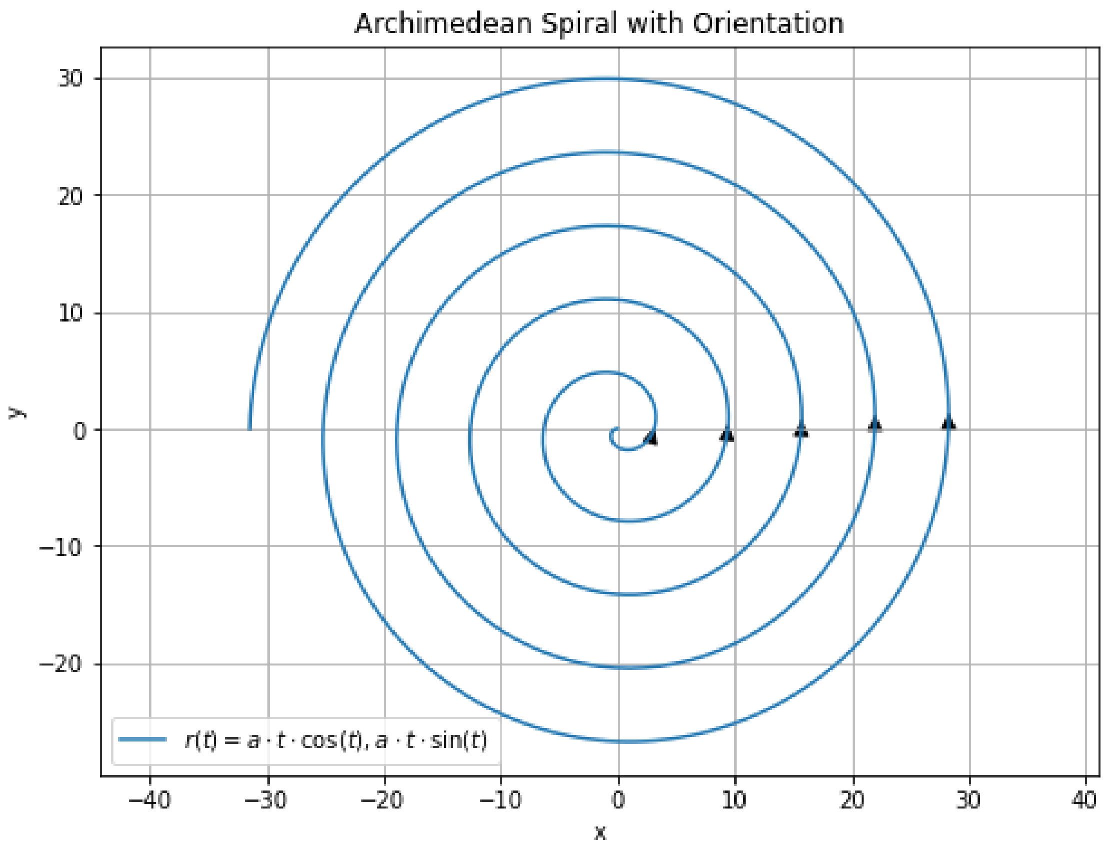
[7]:
```
#Now the Archimedean Spiral
```

[8]:
```python
# Define the positive real number a
a = 1  # Change this value as needed

# Define the parameter t within a reasonable range
t = np.linspace(0, 20*np.pi, 10000)  # Covers several turns

# Parametric equations for the Archimedean Spiral
x = np.exp(a * t) * np.cos(t)
y = np.exp(a * t) * np.sin(t)

# Create the plot
plt.figure(figsize=(8, 6))
plt.plot(x, y, label=r'$r(t) = a \cdot t \cdot \cos(t), a \cdot t \cdot␣
↪\sin(t)$')
plt.title('Archimedean Spiral')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.axis('equal')
plt.legend()
plt.show()
```



[9]:
```
#seeing the orientation
```

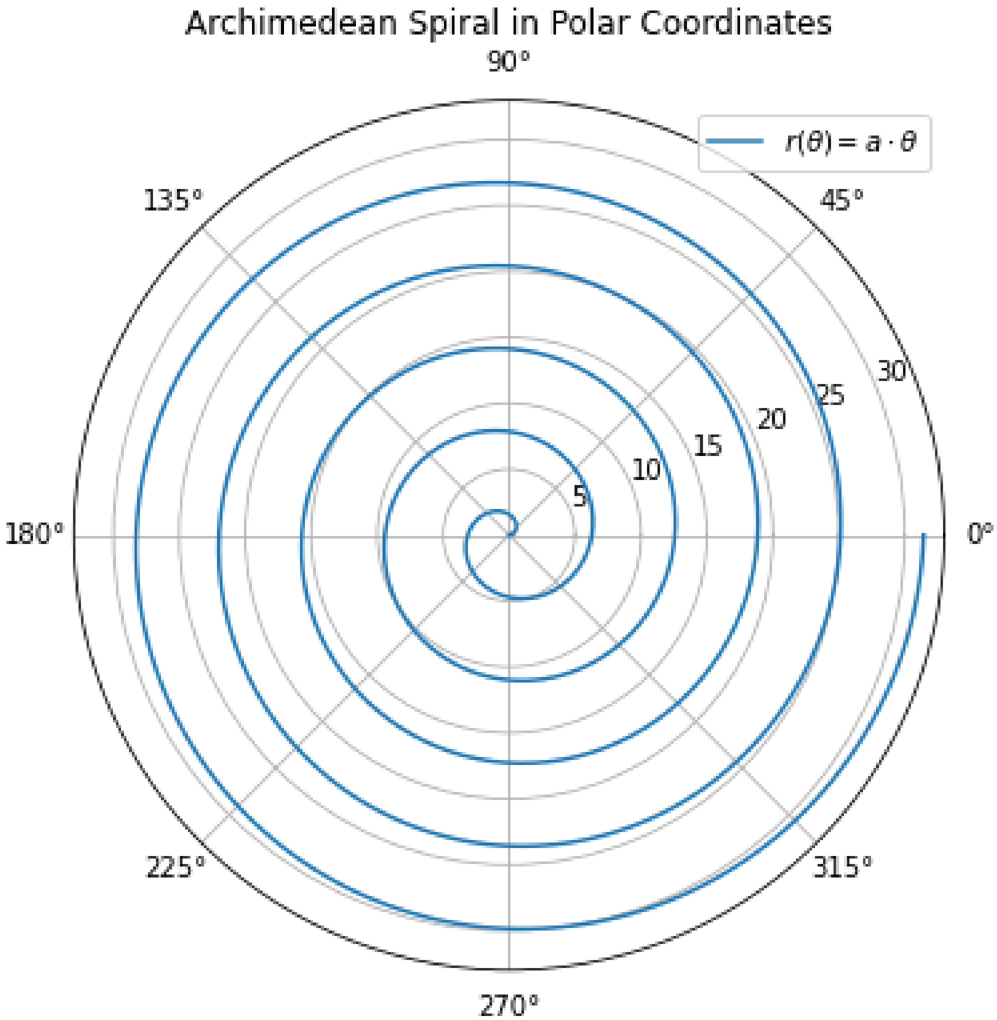
[10]:
```
from matplotlib.patches import FancyArrowPatch

# Define the positive real number a
a = -1  # Change this value as needed

# Define the parameter t within a reasonable range
t = np.linspace(0, 10*np.pi, 1000)  # Covers several turns

# Parametric equations for the Archimedean Spiral
x = a * t * np.cos(t)
y = a * t * np.sin(t)

# Create the plot
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(x, y, label=r'$r(t) = a \cdot t \cdot \cos(t), a \cdot t \cdot␣
↪\sin(t)$')
ax.set_title('Archimedean Spiral with Orientation')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.grid(True)
ax.axis('equal')

# Function to add an arrow at a specified index
def add_arrow(line, position=None, direction='right', size=15, color='k'):
    if position is None:
        position = line.get_xydata().shape[0] // 2
    x, y = line.get_xydata()[position]
    dx, dy = line.get_xydata()[position + 1] - line.get_xydata()[position]
    arrow = FancyArrowPatch((x, y), (x + dx, y + dy), arrowstyle='-|>',␣
↪color=color, mutation_scale=size)
    ax.add_patch(arrow)

# Adding arrows to the spiral
line = ax.lines[0]
arrow_positions = [100, 300, 500, 700, 900]  # positions along the curve for␣
↪arrows
for pos in arrow_positions:
    add_arrow(line, position=pos)

ax.legend()
plt.show()
```



[11]:
```
#Now let us do it in polar coordinates
```

[12]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the positive real number a
a = 1  # Change this value as needed

# Define the parameter theta within a reasonable range
theta = np.linspace(0, 10*np.pi, 1000)  # Covers several turns

# Radial coordinate for Archimedean Spiral
r = a * theta

# Create the polar plot
plt.figure(figsize=(8, 6))
ax = plt.subplot(111, projection='polar')
ax.plot(theta, r, label=r'$r(\theta) = a \cdot \theta$')
ax.set_title('Archimedean Spiral in Polar Coordinates')
plt.legend()
```

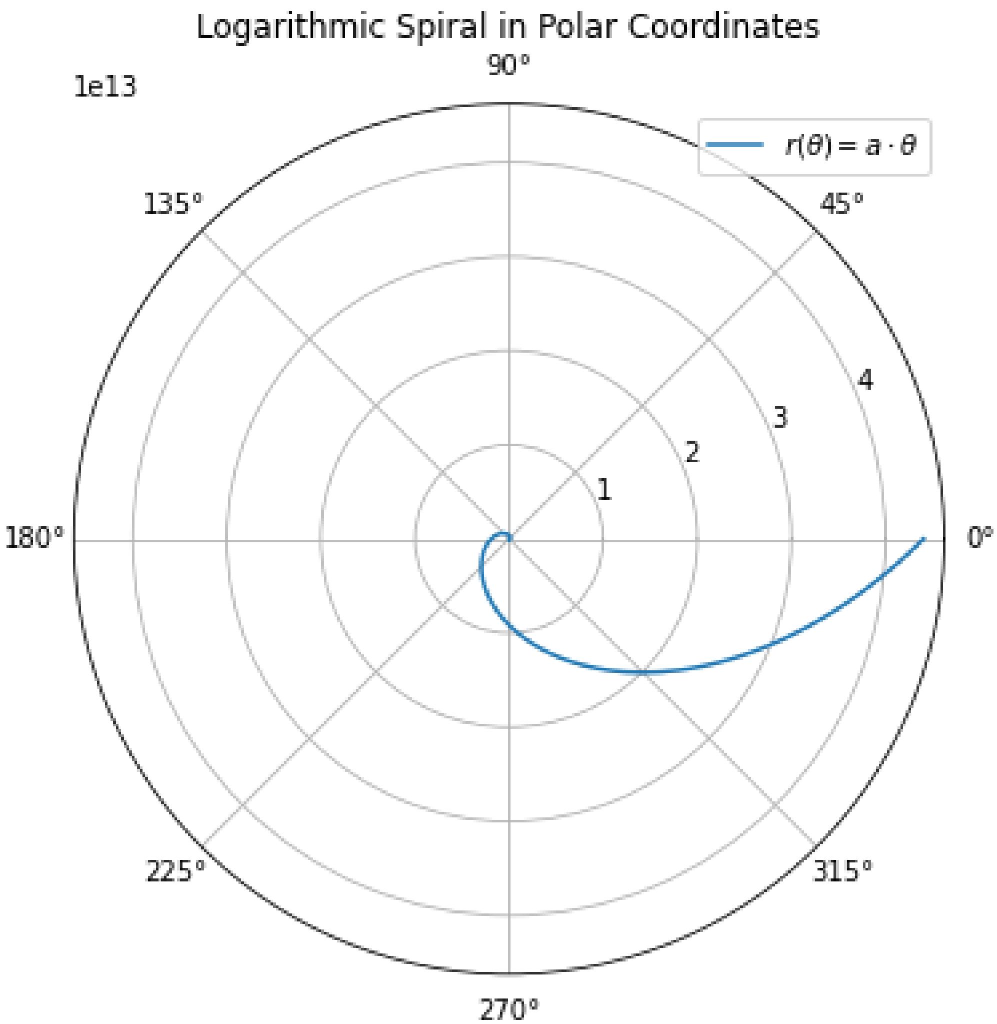
```
plt.show()
```



[13]:
```
#Logarithmic spiral
```

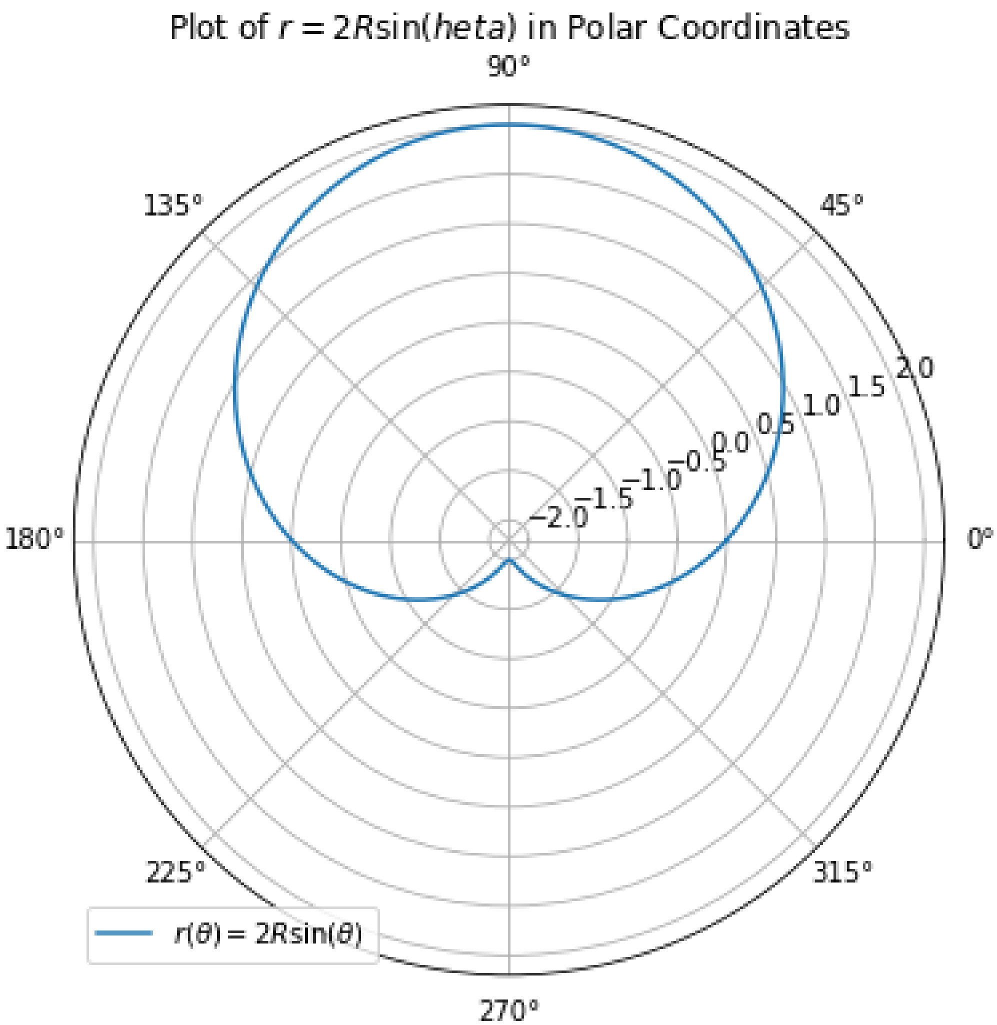
[14]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the positive real number a
a = 1  # Change this value as needed

# Define the parameter theta within a reasonable range
theta = np.linspace(0, 10*np.pi, 1000)  # Covers several turns

# Radial coordinate for Logaritmic Spiral
r = np.exp(a * theta)
```

```
# Create the polar plot
plt.figure(figsize=(8, 6))
ax = plt.subplot(111, projection='polar')
ax.plot(theta, r, label=r'$r(\theta) = a \cdot \theta$')
ax.set_title('Logarithmic Spiral in Polar Coordinates')
plt.legend()
plt.show()
```



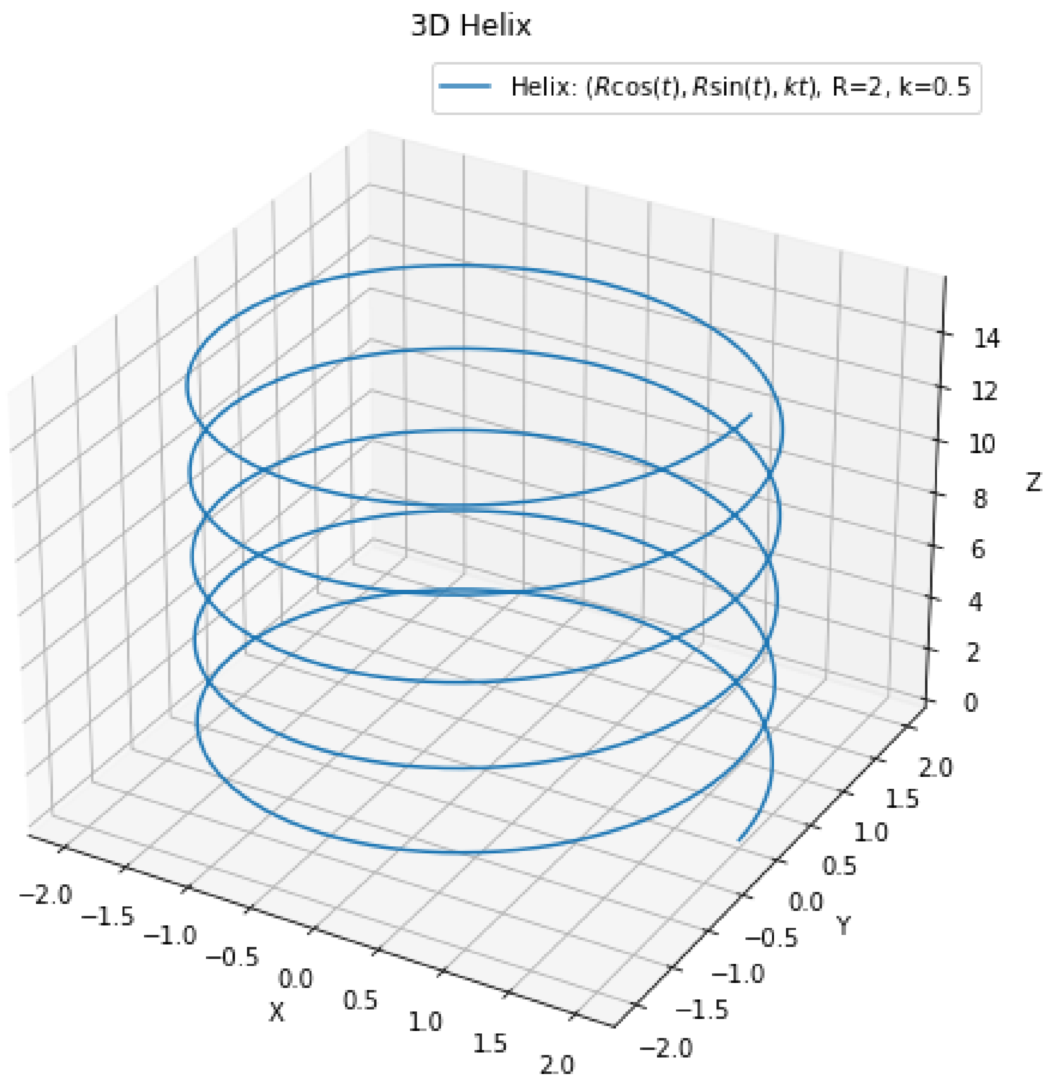
[15]:
```
# Define the positive real number R
R = 1  # Change this value as needed

# Define the parameter theta within a reasonable range
theta = np.linspace(0, 2*np.pi, 1000)  # Complete circle
```

```python
# Radial coordinate for the given curve
r = 2 * R * np.sin(theta)

# Create the polar plot
plt.figure(figsize=(8, 6))
ax = plt.subplot(111, projection='polar')
ax.plot(theta, r, label=r'$r(\theta) = 2R \sin(\theta)$')
ax.set_title('Plot of $r = 2R \sin(\theta)$ in Polar Coordinates')
plt.legend()
plt.show()
```



[16]:
```python
#the helix
```

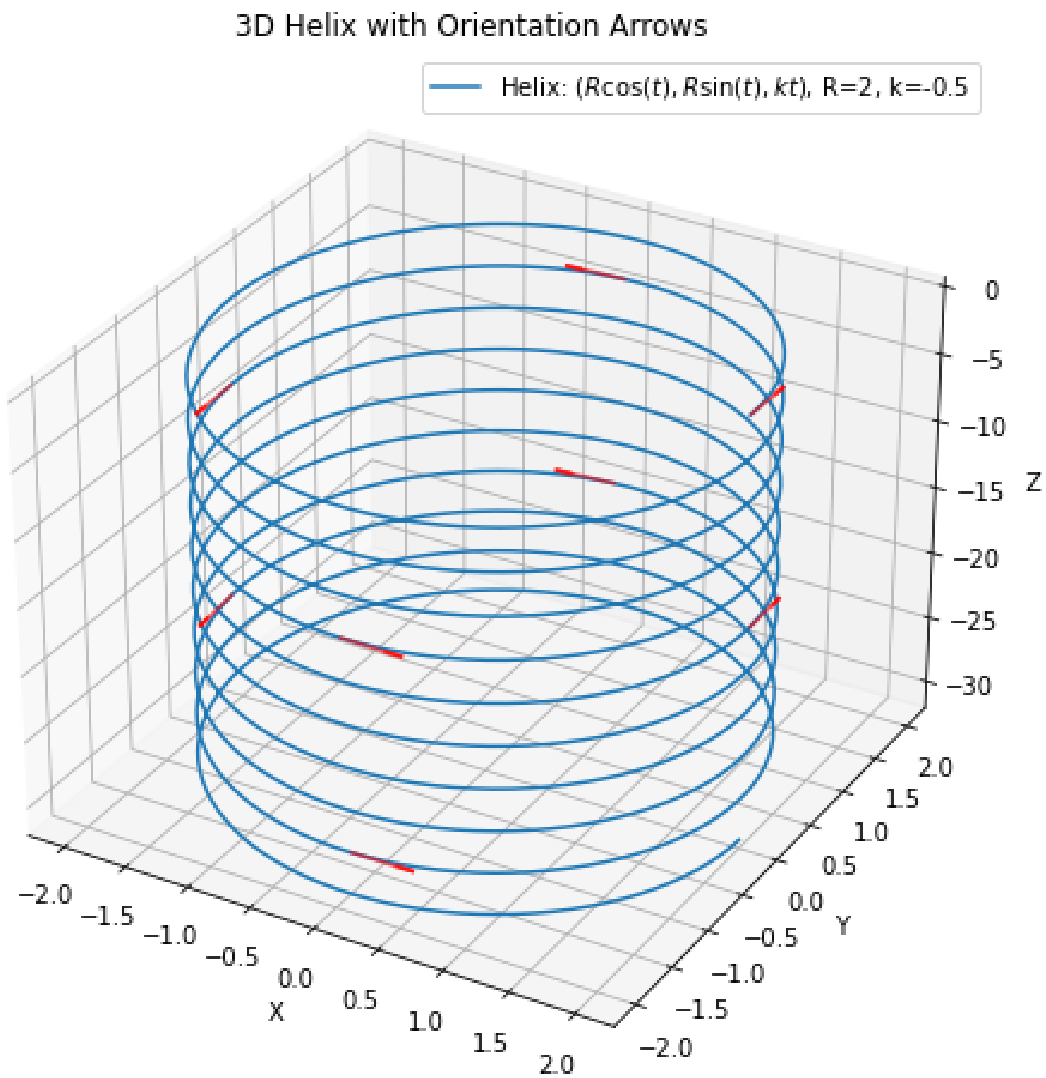
[17]:
```
# Define the parameters R and k
R = 2  # Radius of the helix, change as needed
k = 0.5  # Linear progression rate, change as needed

# Define the parameter t within a reasonable range
t = np.linspace(0, 10*np.pi, 1000)  # Adjust the range as needed for more or
 ↪fewer turns

# Parametric equations for the helix
x = R * np.cos(t)
y = R * np.sin(t)
z = k * t

# Create the 3D plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot(x, y, z, label=f'Helix: $(R\cos(t), R\sin(t), kt)$, R={R}, k={k}')
ax.set_title('3D Helix')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.legend()
plt.show()
```



[18]:
```
#if we want, we can add some arrows to emphasize the orientation
```

[19]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the parameters R and k
R = 2  # Radius of the helix, change as needed
k = -0.5  # Linear progression rate, change as needed

# Define the parameter t within a reasonable range
t = np.linspace(0, 20*np.pi, 1000)  # Adjust the range as needed for more or␣
↪fewer turns
```

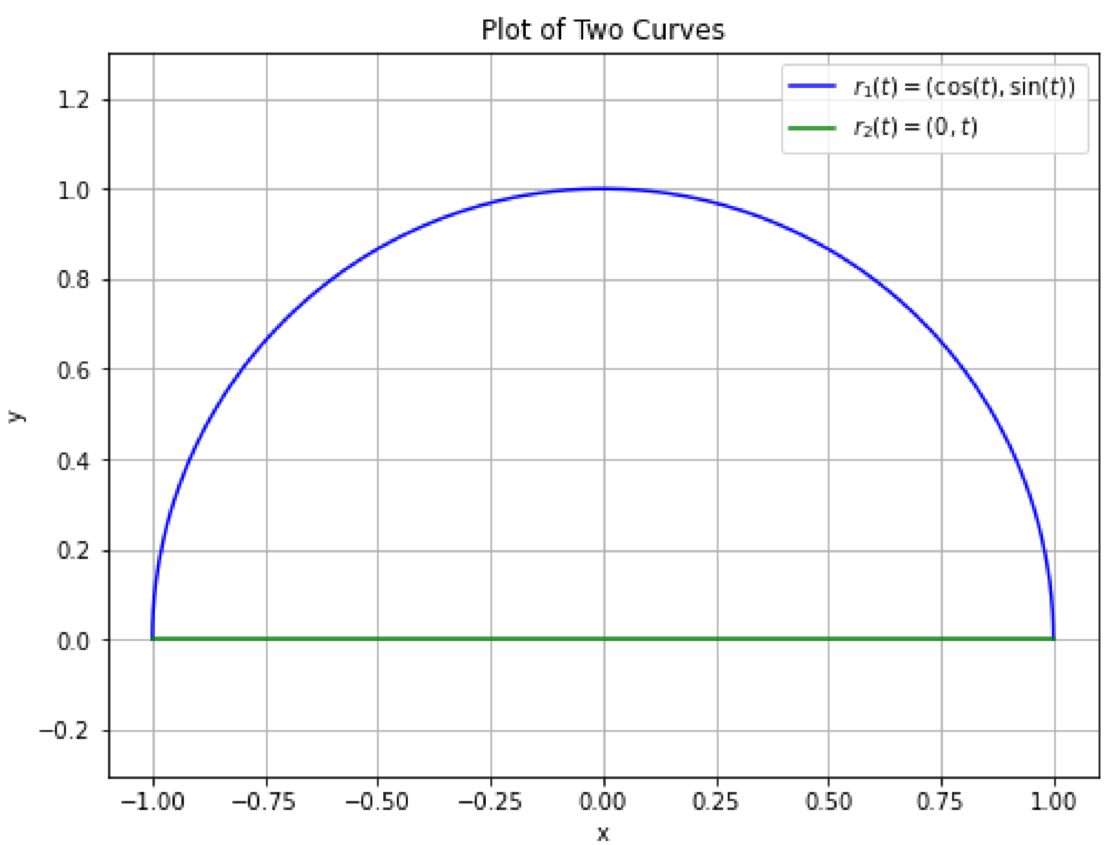
```python
# Parametric equations for the helix
x = R * np.cos(t)
y = R * np.sin(t)
z = k * t

# Create the 3D plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot(x, y, z, label=f'Helix: $(R\cos(t), R\sin(t), kt)$, R={R}, k={k}')

# Adding arrows using quiver
# Select points to place arrows
step = 125  # Interval for placing arrows
t_arrows = t[::step]
x_arrows = R * np.cos(t_arrows)
y_arrows = R * np.sin(t_arrows)
z_arrows = k * t_arrows

# Vectors for the direction of the arrows
u = -R * np.sin(t_arrows)  # Derivative of R*cos(t)
v = R * np.cos(t_arrows)   # Derivative of R*sin(t)
w = k * np.ones_like(t_arrows)  # Derivative of kt

ax.quiver(x_arrows, y_arrows, z_arrows, u, v, w, length=0.5, normalize=True,
 ↪color='r', alpha=0.6)

ax.set_title('3D Helix with Orientation Arrows')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.legend()
plt.show()
```




```
[20]: #piecewise functions - one from [0,pi], the other from [-1,1]
```

```
[21]: import numpy as np
import matplotlib.pyplot as plt

# Define the parameter t for the half-circle
t1 = np.linspace(0, np.pi, 100)

# Define the parameter t for the line segment
t2 = np.linspace(-1, 1, 100)

# Parametric equations for the half-circle
x1 = np.cos(t1)
```

```python
y1 = np.sin(t1)

# Parametric equations for the vertical line
x2 = t2
y2 = np.zeros_like(t2) # x = 0 for all t in [-1, 1]

# Create the plot
plt.figure(figsize=(8, 6))
plt.plot(x1, y1, label=r'$r_1(t) = (\cos(t), \sin(t))$', color='blue')
plt.plot(x2, y2, label=r'$r_2(t) = (0, t)$', color='green')
plt.title('Plot of Two Curves')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.axis('equal')
plt.legend()
plt.show()
```

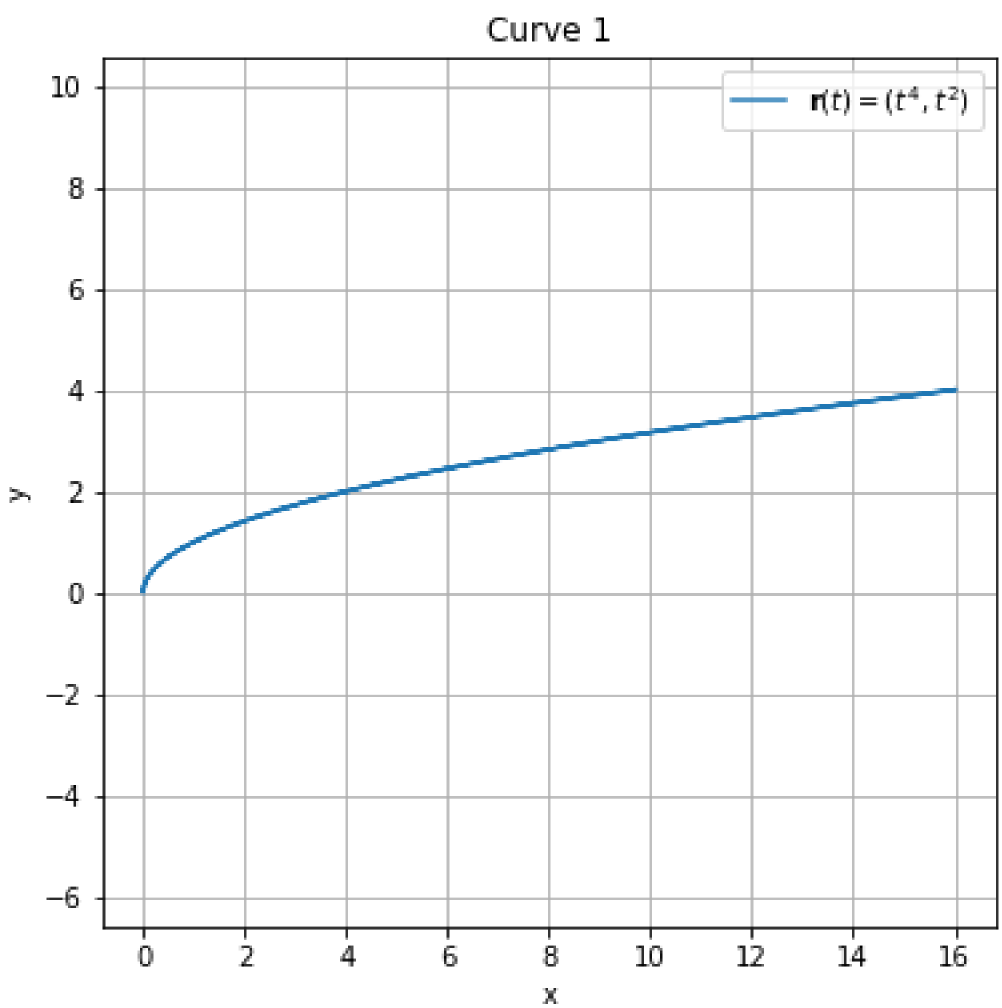


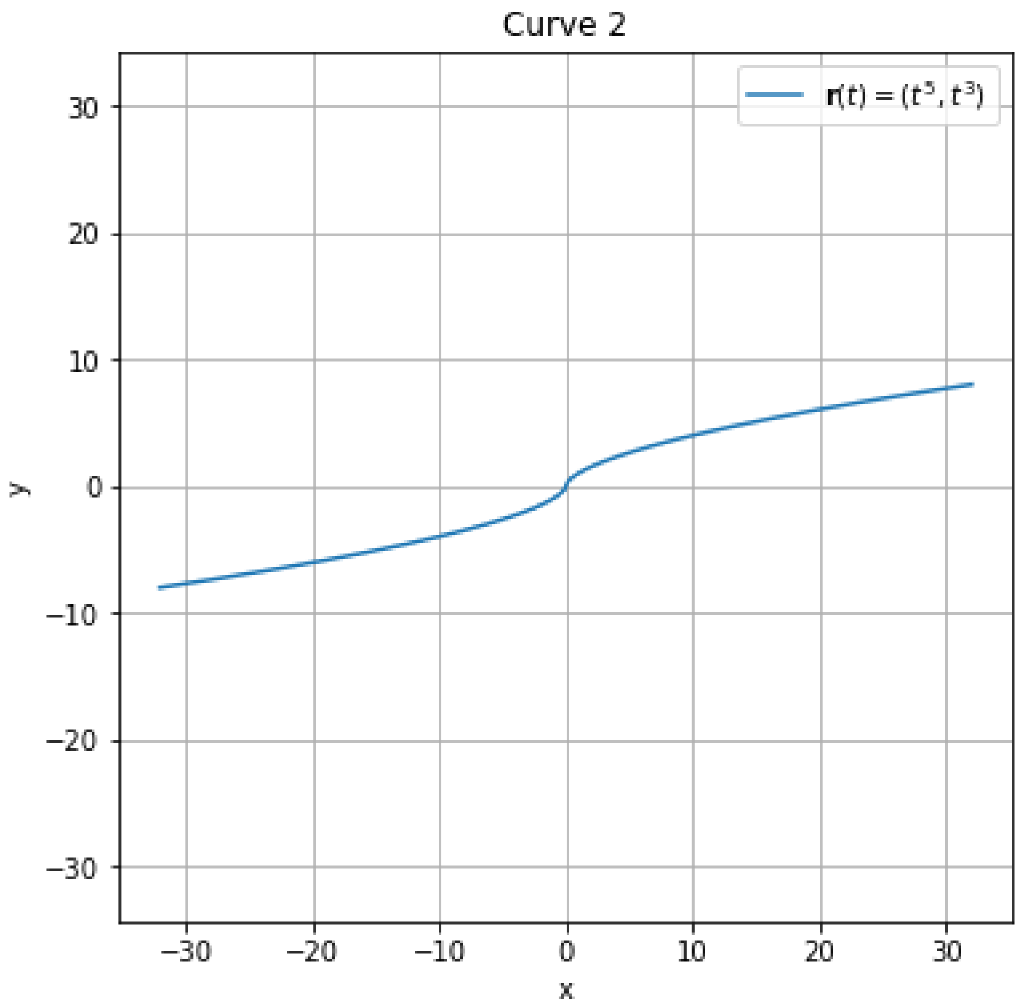
[22]:
```python
# Plots for the first problems in the seminar. You should be able to sketch␣
↪these by hand
```

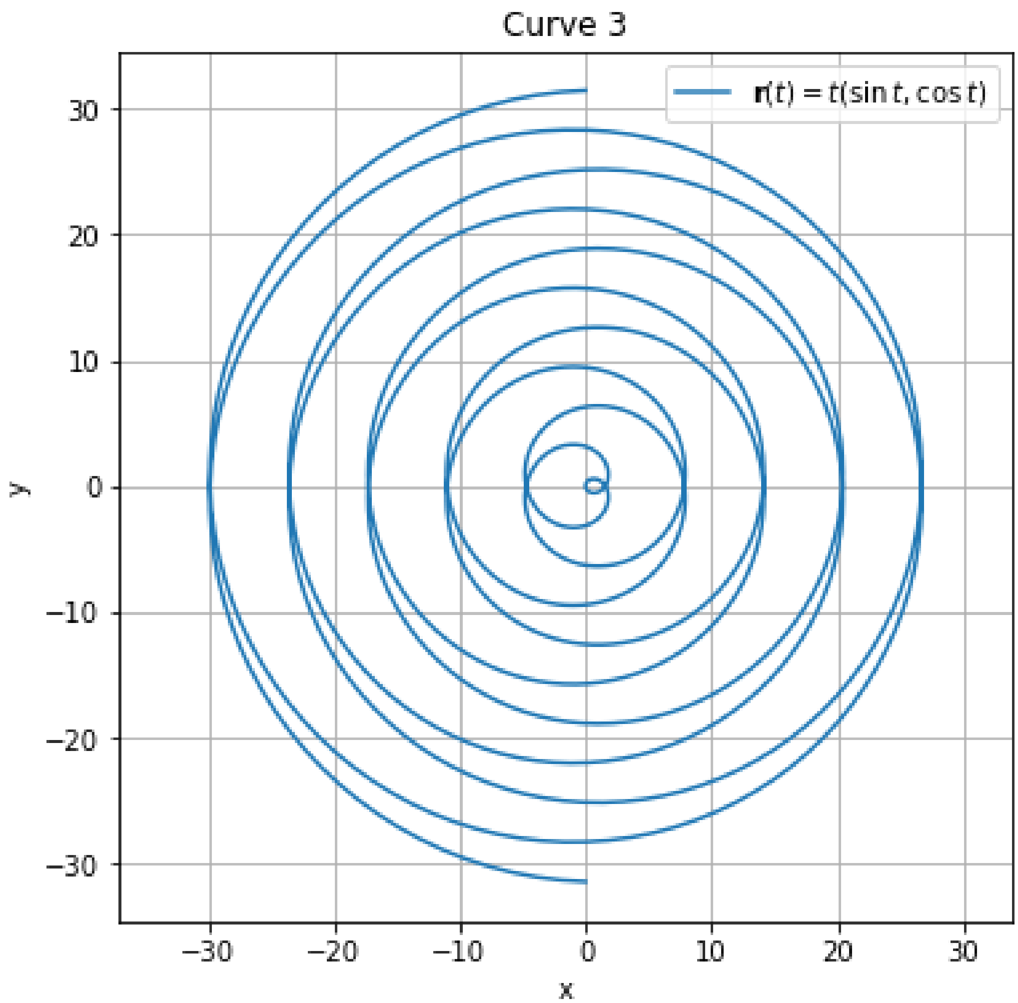
[23]:

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the parameter t for each curve
t1 = np.linspace(-2, 2, 400)
t2 = np.linspace(-2, 2, 400)
t3 = np.linspace(-10*np.pi, 10*np.pi, 1000)
t4 = np.linspace(0, 2*np.pi, 400)

# Curve 1: (t^4, t^2)
x1 = t1**4
y1 = t1**2
plt.figure(figsize=(6, 6))
plt.plot(x1, y1, label=r'$\mathbf{r}(t) = (t^4, t^2)$')
plt.title('Curve 1')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.legend()
plt.axis('equal')
plt.show()

# Curve 2: (t^5, t^3)
x2 = t2**5
y2 = t2**3
plt.figure(figsize=(6, 6))
plt.plot(x2, y2, label=r'$\mathbf{r}(t) = (t^5, t^3)$')
plt.title('Curve 2')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.legend()
plt.axis('equal')
plt.show()

# Curve 3: t(sin t, cos t)
x3 = t3 * np.sin(t3)
y3 = t3 * np.cos(t3)
plt.figure(figsize=(6, 6))
plt.plot(x3, y3, label=r'$\mathbf{r}(t) = t(\sin t, \cos{t})$')
plt.title('Curve 3')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.legend()
plt.axis('equal')
```

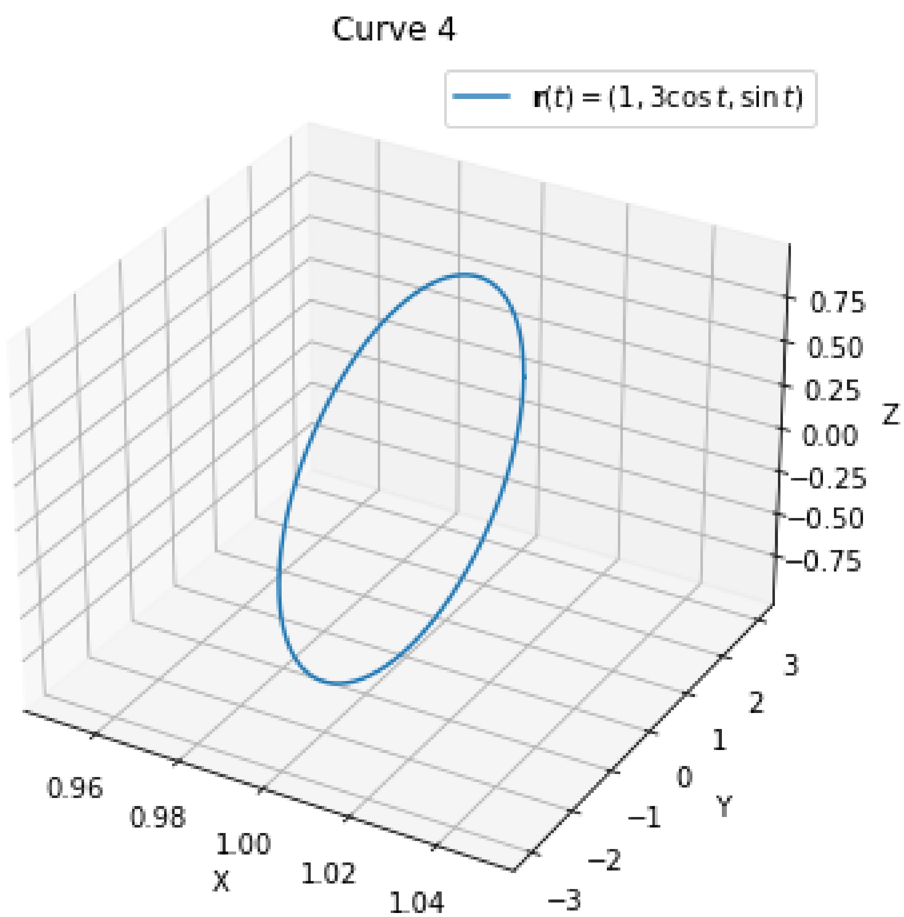
```
plt.show()

# Curve 4: (1, 3 cos t, sin t) - 3D plot
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
x4 = np.ones_like(t4)
y4 = 3 * np.cos(t4)
z4 = np.sin(t4)
ax.plot(x4, y4, z4, label=r'$\mathbf{r}(t) = (1, 3\cos{t}, \sin{t})$')
ax.set_title('Curve 4')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.legend()
plt.show()
```




Curve 2
$\mathbf{r}(t) = (t^5, t^3)$
30
20
10
0
−10
−20
−30
y
−30
−20
−10
0
10
20
30
x

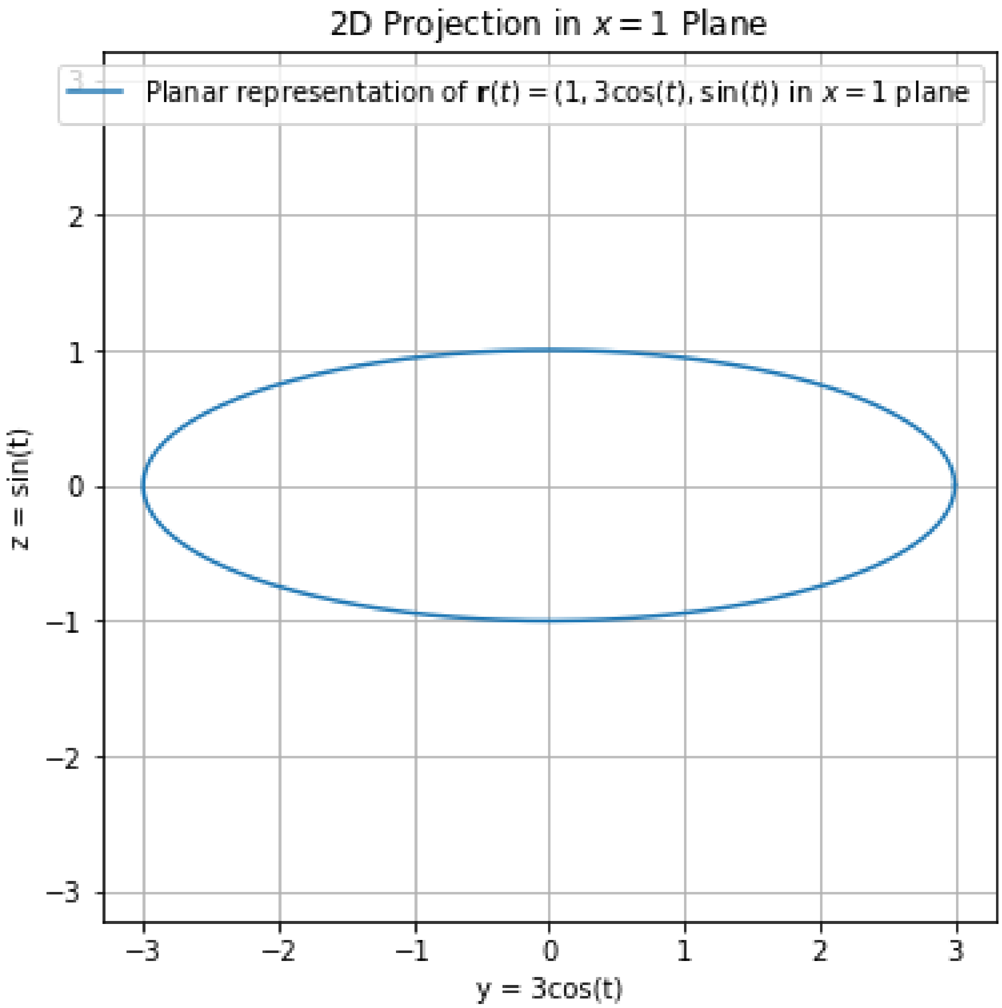


Curve 3
$\mathbf{r}(t) = t(\sin t, \cos t)$
30
20
10
0
−10
−20
−30
y
−30
−20
−10
0
10
20
30
x



[24]:
```
# Define the parameter t for the 4th curve
t4 = np.linspace(0, 2*np.pi, 400)

# Since x = 1 always, we consider only the y and z coordinates
y4 = 3 * np.cos(t4)
z4 = np.sin(t4)

# Create the 2D plot for the plane x = 1
plt.figure(figsize=(6, 6))
plt.plot(y4, z4, label=r'Planar representation of $\mathbf{r}(t) = (1,␣
 ↪3\cos(t), \sin(t))$ in $x=1$ plane')
plt.title('2D Projection in $x=1$ Plane')
plt.xlabel('y = 3cos(t)')
plt.ylabel('z = sin(t)')
plt.grid(True)
plt.axis('equal')
plt.legend()
```

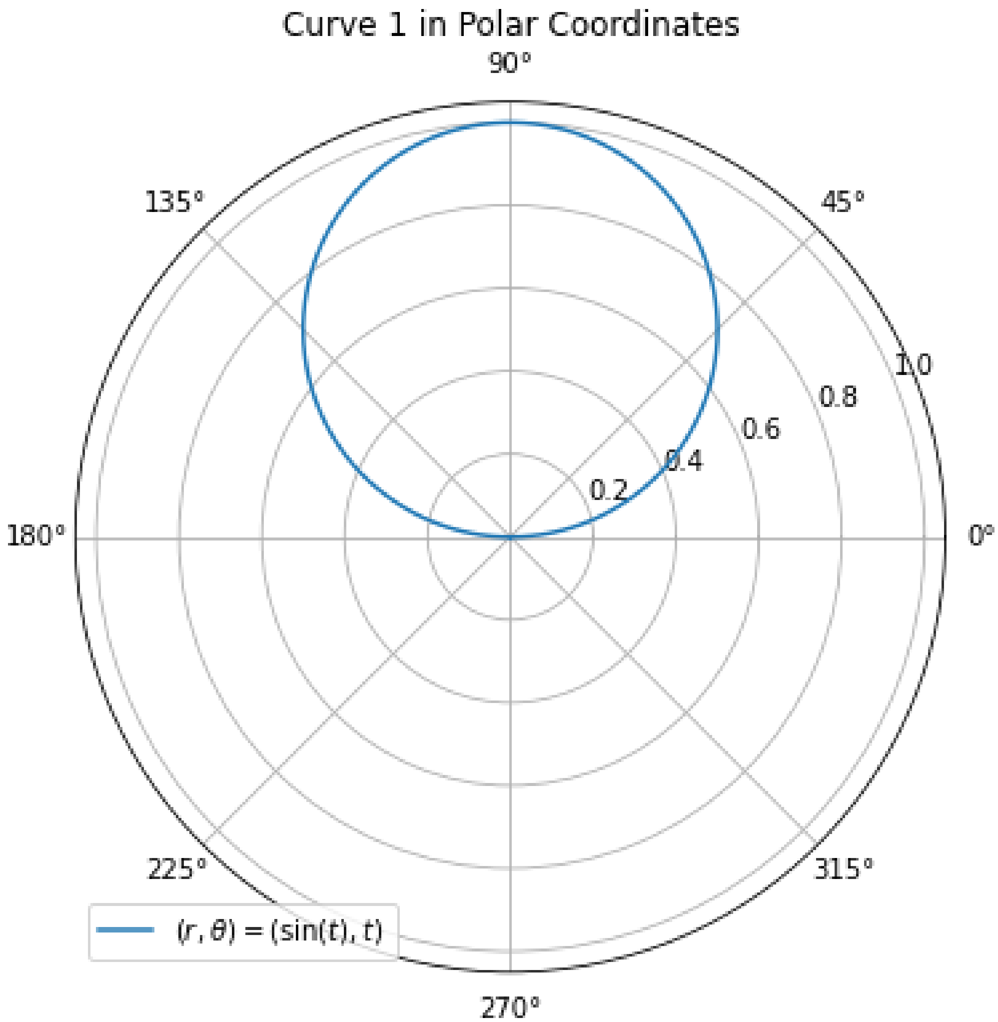
```
plt.show()
```

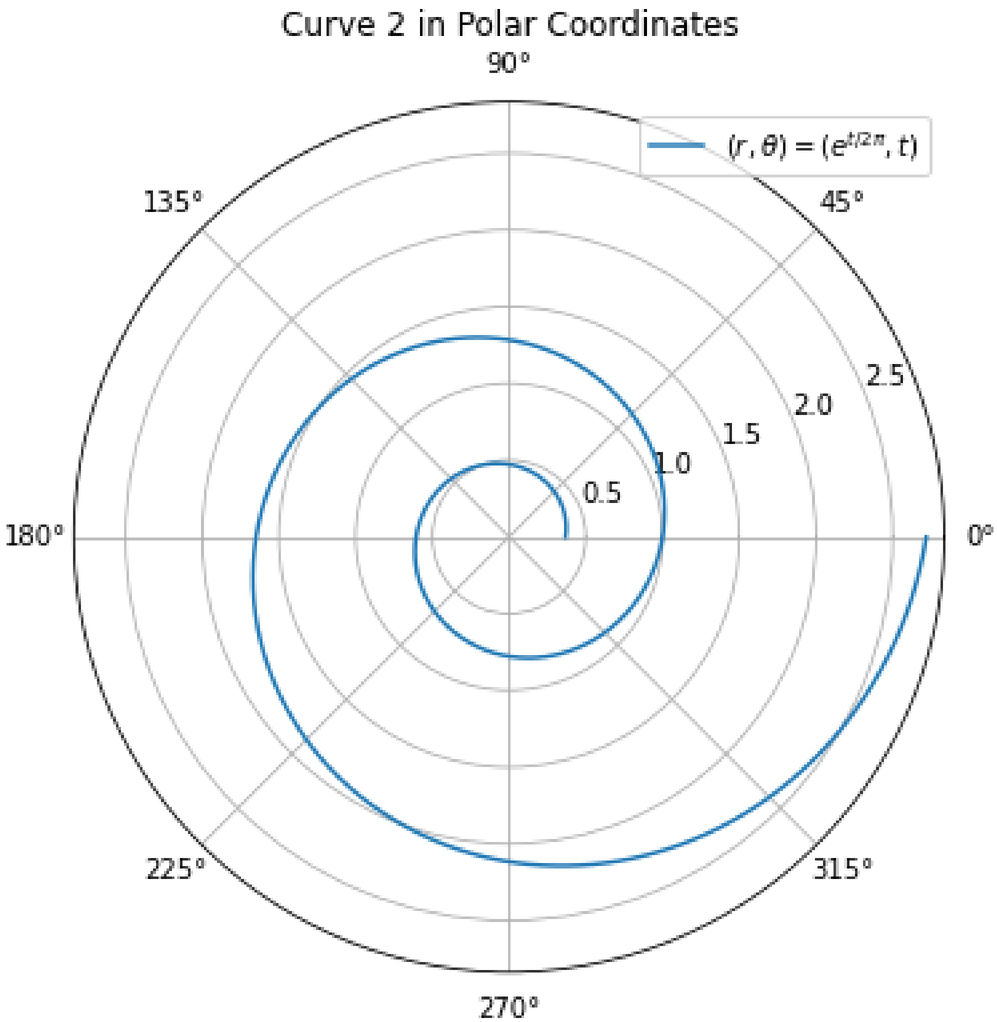


[25]:
```
#Problem 2
```

[26]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the parameter t for each curve
t1 = np.linspace(0, np.pi, 500)
t2 = np.linspace(-2*np.pi, 2*np.pi, 1000)

# Curve 1: (sin(t), t)
r1 = np.sin(t1)
theta1 = t1
```

```python
# Curve 2: (exp(t/2pi), t)
r2 = np.exp(t2 / (2 * np.pi))
theta2 = t2

# Plotting Curve 1
plt.figure(figsize=(6, 6))
ax1 = plt.subplot(111, projection='polar')
ax1.plot(theta1, r1, label=r'$(r, \theta) = (\sin(t), t)$')
ax1.set_title('Curve 1 in Polar Coordinates')
ax1.legend()

# Plotting Curve 2
plt.figure(figsize=(6, 6))
ax2 = plt.subplot(111, projection='polar')
ax2.plot(theta2, r2, label=r'$(r, \theta) = \left(e^{t / 2 \pi}, t\right)$')
ax2.set_title('Curve 2 in Polar Coordinates')
ax2.legend()

plt.show()
```

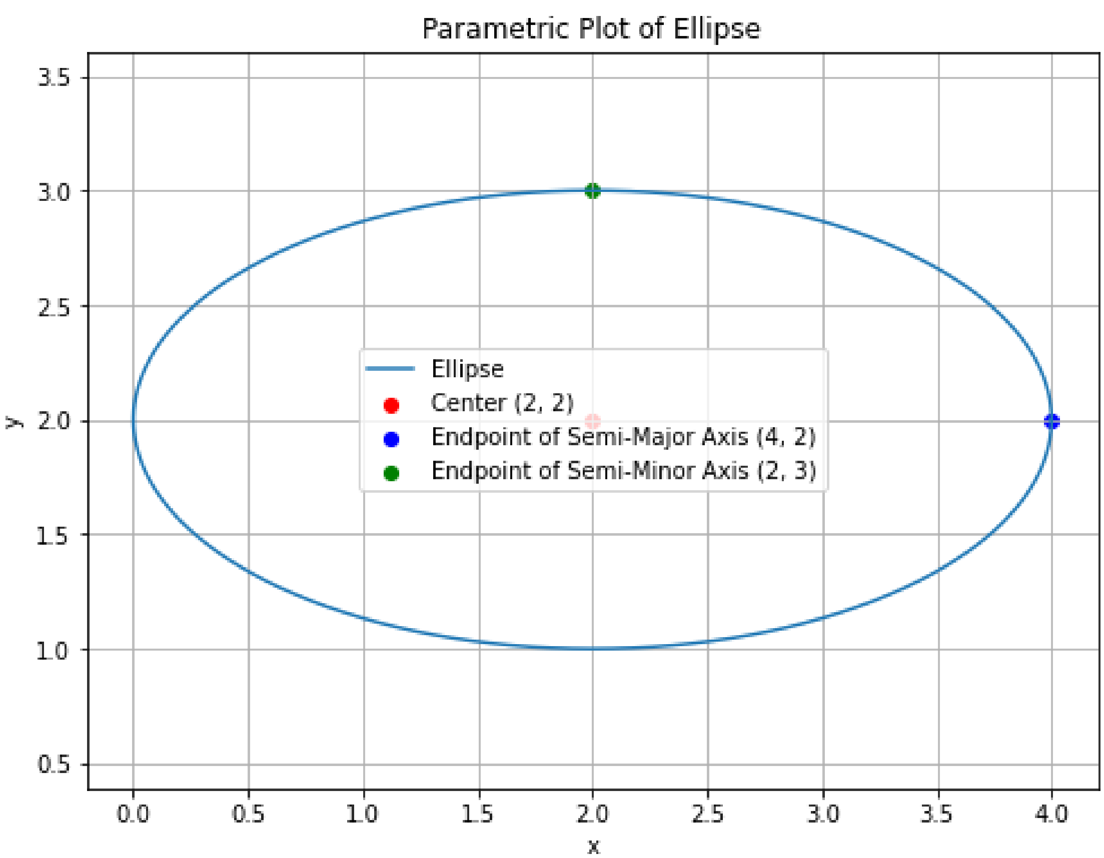

Curve 1 in Polar Coordinates
90°
45°
135°
180°
0°
225°
315°
270°
0.2
0.4
0.6
0.8
1.0
$(r, \theta) = (\sin(t), t)$



[27]:
```
#Plot of the ellipse in problem 4
```

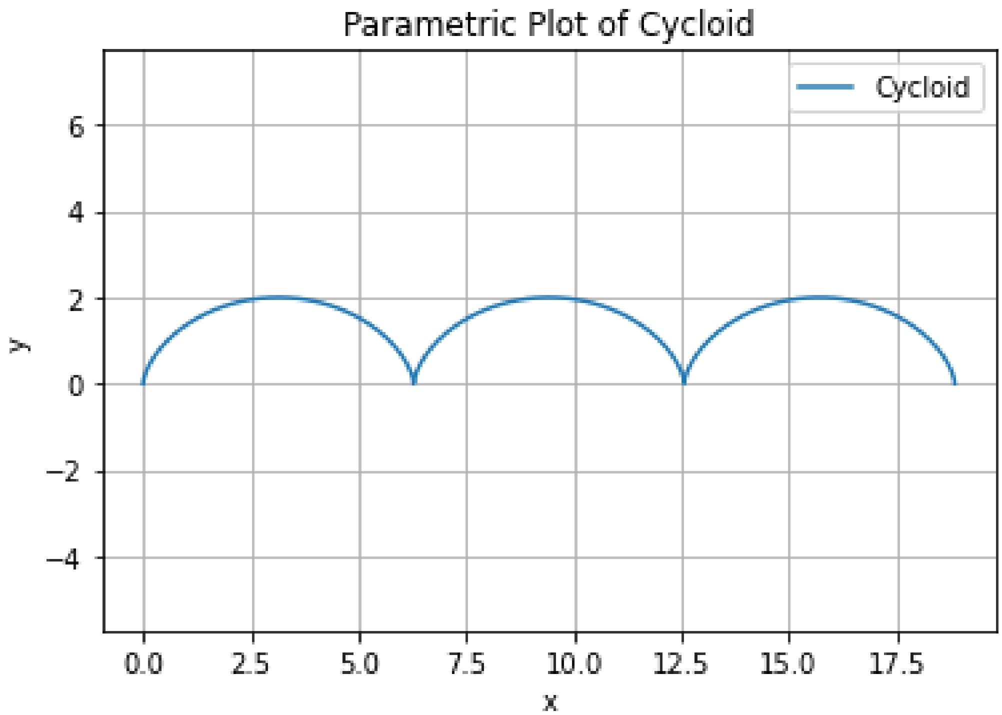
[28]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the parameter t
t = np.linspace(0, 2*np.pi, 1000)

# Define the center and semi-axes
h = 2
k = 2
a = 2
b = 1

# Parametric equations of the ellipse
```

```
x = h + a * np.cos(t)
y = k + b * np.sin(t)

# Plot the ellipse
plt.figure(figsize=(8, 6))
plt.plot(x, y, label='Ellipse')
plt.scatter([h], [k], color='red', label='Center (2, 2)')
plt.scatter([h + a], [k], color='blue', label='Endpoint of Semi-Major Axis (4,␣
↪2)')
plt.scatter([h], [k + b], color='green', label='Endpoint of Semi-Minor Axis (2,␣
↪3)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Parametric Plot of Ellipse')
plt.axis('equal')
plt.legend()
plt.grid(True)
plt.show()
```



[29]:
```
#cycloid
```

[30]:
```
import numpy as np
import matplotlib.pyplot as plt

def cycloid_parametrization(a, t_values):
    x_values = a * (t_values - np.sin(t_values))
    y_values = a * (1 - np.cos(t_values))
    return x_values, y_values

# Define the range of t values
t_values = np.linspace(0, 6*np.pi, 1000)  # adjust the range as needed

# Set the value of the parameter a
a = 1  # Replace 1 with your desired value

# Calculate x and y values using the parametric equations
x_values, y_values = cycloid_parametrization(a, t_values)

# Plot the cycloid
plt.plot(x_values, y_values, label='Cycloid')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Parametric Plot of Cycloid')
plt.grid(True)
plt.axis('equal')
plt.legend()
plt.show()
```


Parametric Plot of Cycloid
Cycloid
6
4
2
0
−2
−4
y
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
x

[ ]: