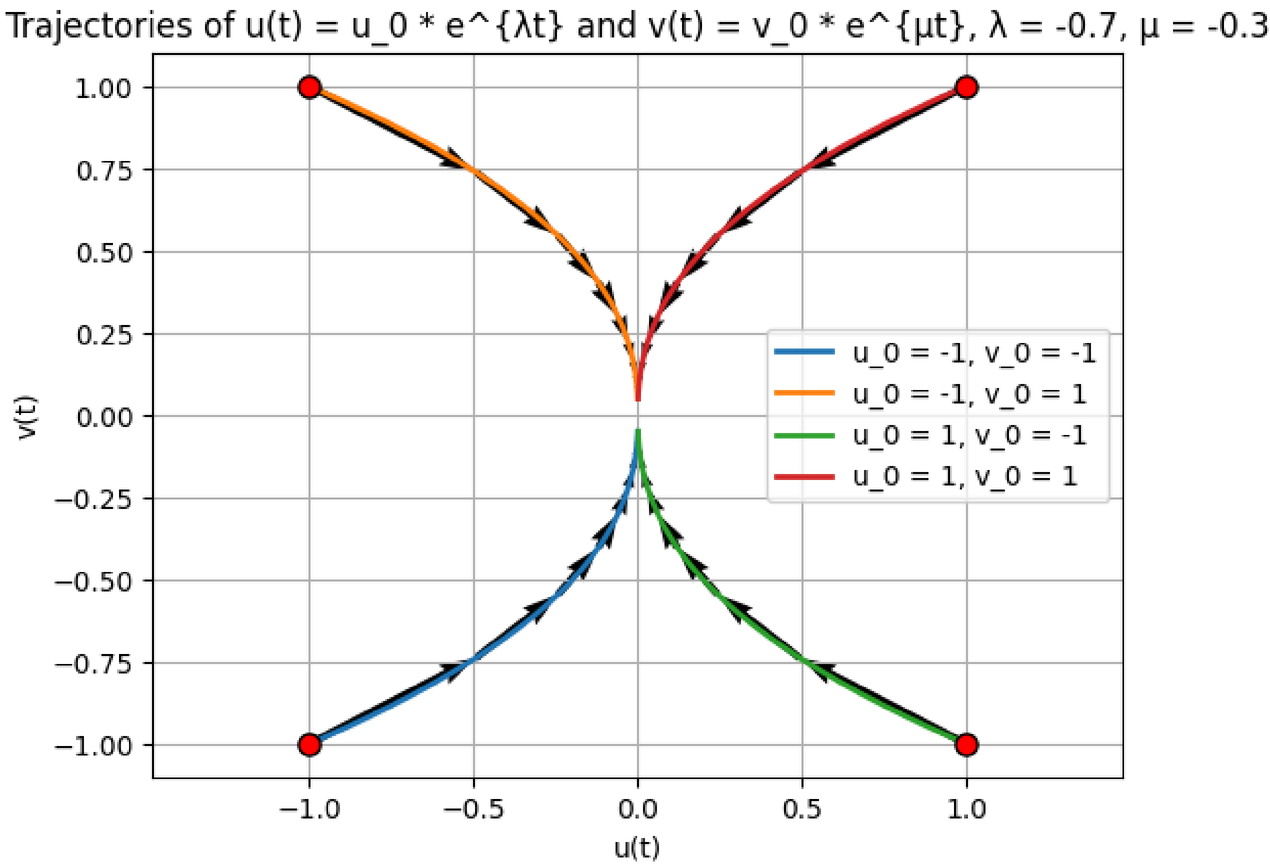
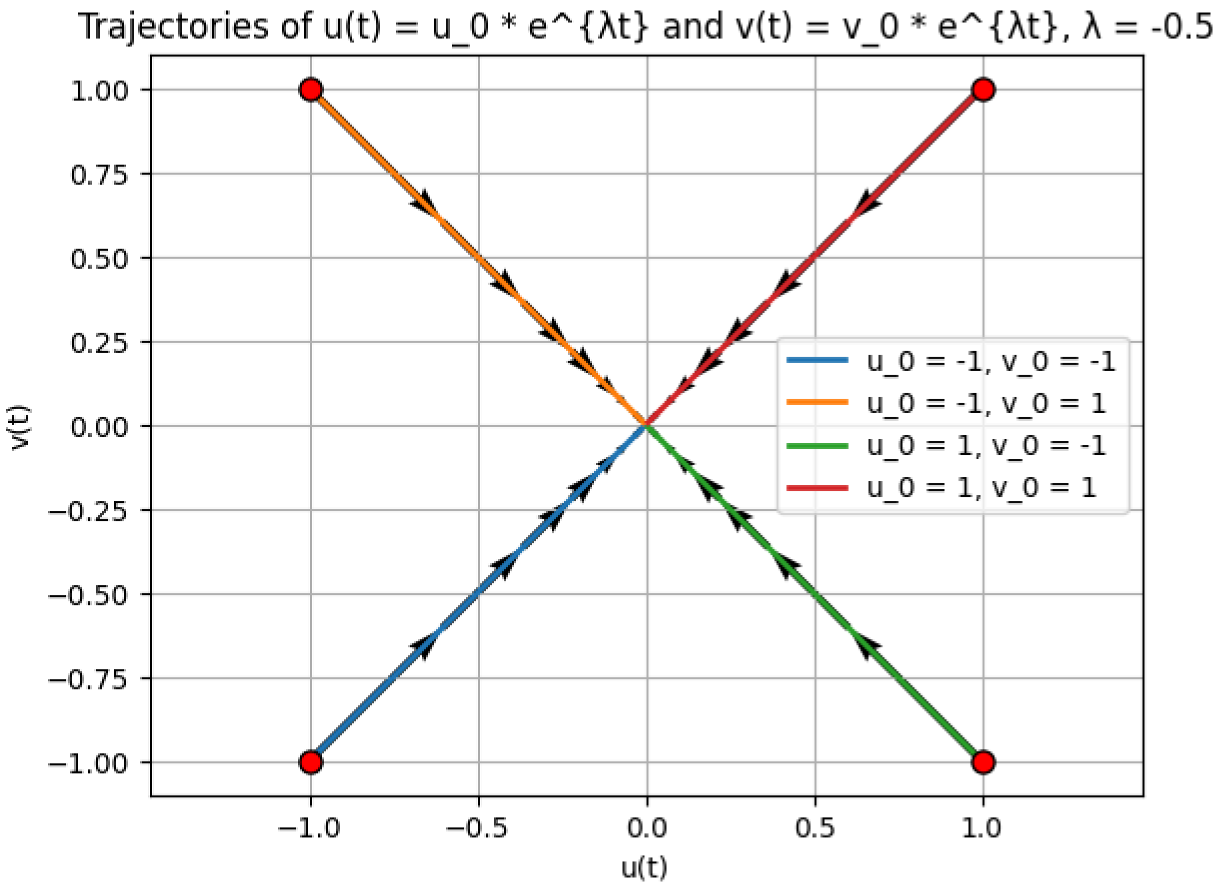
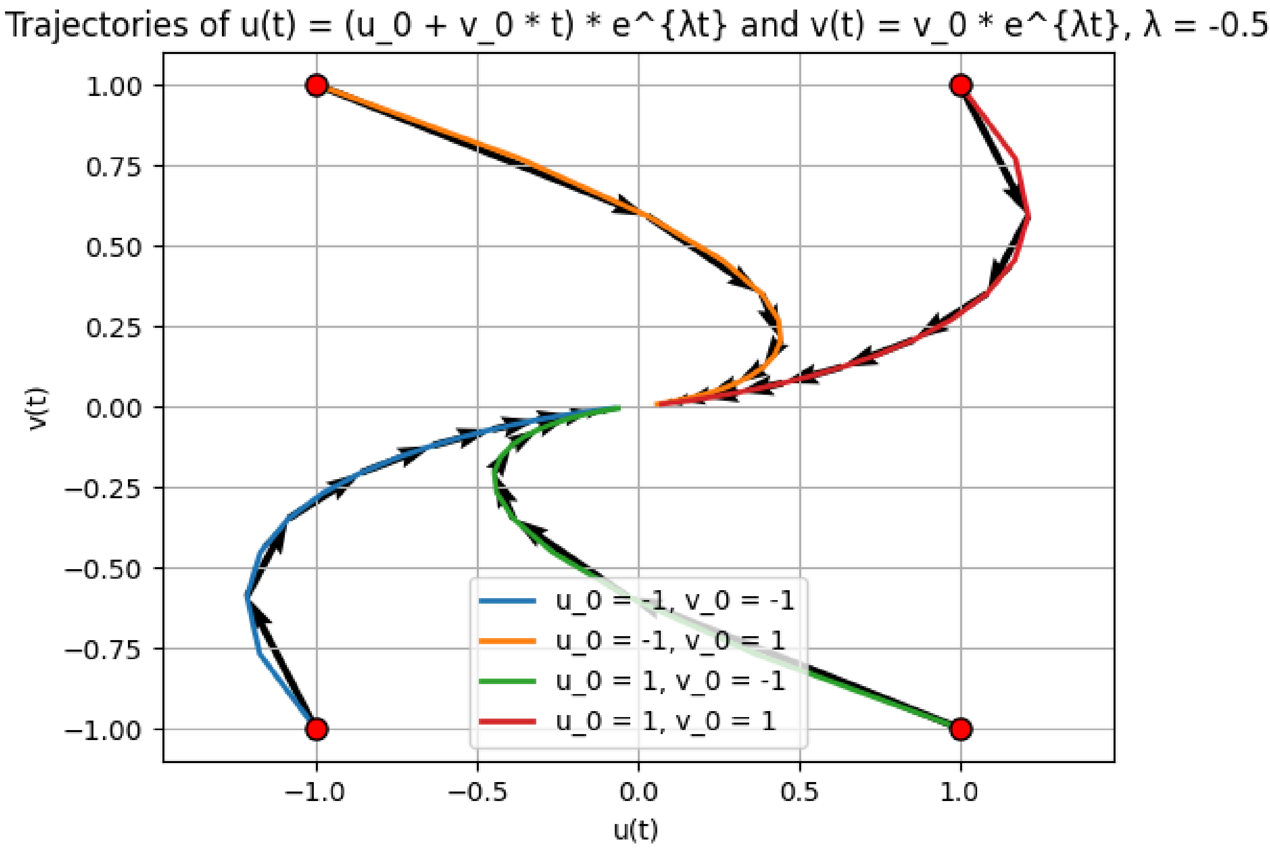
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
lambda_val = -0.7  # Lambda (must be negative)
mu = -0.3  # Mu (must be less negative than lambda, i.e., -0.3 < -0.7)

# Uncomment the following lines for the other case:
# Asymptotically stable (if λ < μ < 0)
# lambda_val = 0.7
# mu = 0.3

# t represents the time values
t = np.linspace(0, 10, 100)  # Reduced number of data points for clarity

# Initial conditions for the trajectory
u0_vals = [-1, -1, 1, 1]  # Initial values of u in different quadrants
v0_vals = [-1, 1, -1, 1]  # Initial values of v in different quadrants

plt.figure()
plt.grid(True)
plt.axis('equal')  # Ensure the scale is the same in all directions

# Loop over each initial condition
for i in range(len(u0_vals)):
    u0 = u0_vals[i]
    v0 = v0_vals[i]

    # Functions for the case: u = u0 * exp(lambda * t), v = v0 * exp(mu * t)
    u = u0 * np.exp(lambda_val * t)
    v = v0 * np.exp(mu * t)

    # Plot trajectory
    plt.plot(u, v, linewidth=2, label=f'u_0 = {u0}, v_0 = {v0}')

    # Plot initial point as a dot
    plt.plot(u[0], v[0], 'ko', markersize=8, markerfacecolor='r')  # Black dot wi

    # Add arrows to indicate direction
    # Arrow placement and length scaling
    arrow_interval = round(len(t) / 10)  # Interval for arrow placement
    for j in range(0, len(t) - arrow_interval, arrow_interval):
        plt.quiver(u[j], v[j], u[j + arrow_interval] - u[j], v[j + arrow_interval
                   angles='xy', scale_units='xy', scale=1, color='k', linewidth=2

# Axis labels and title
plt.xlabel('u(t)')
plt.ylabel('v(t)')
plt.title(f'Trajectories of u(t) = u_0 * e^{{λt}} and v(t) = v_0 * e^{{μt}}, λ =
plt.legend()
plt.show()
```


Trajectories of u(t) = u_0 * e^{λt} and v(t) = v_0 * e^{μt}, λ = -0.7, μ = -0.3
u_0 = -1, v_0 = -1
u_0 = -1, v_0 = 1
u_0 = 1, v_0 = -1
u_0 = 1, v_0 = 1
v(t)
u(t)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0

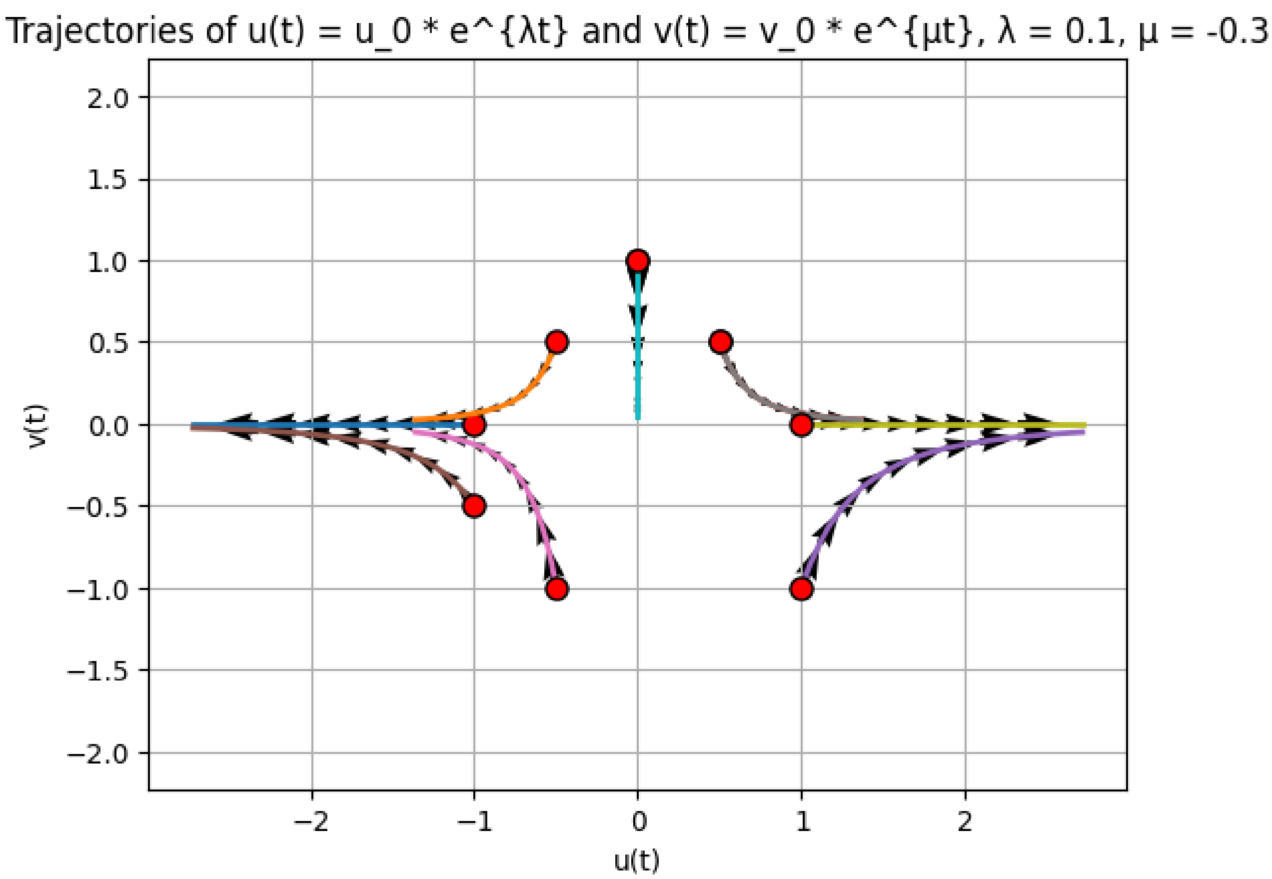
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
lambda_val = -0.5  # Lambda value
# λ < 0: Stable, orbits approach the origin along straight lines.
# λ > 0: Unstable, orbits diverge from the origin along straight lines.

# Time array
t = np.linspace(0, 10, 100)

# Initial conditions for the trajectory
u0_vals = [-1, -1, 1, 1]  # Initial values of u in different quadrants
v0_vals = [-1, 1, -1, 1]  # Initial values of v in different quadrants

plt.figure()
plt.grid(True)
plt.axis('equal')  # Ensure the scale is the same in all directions

# Loop over each initial condition
for i in range(len(u0_vals)):
    u0 = u0_vals[i]
    v0 = v0_vals[i]

    # Functions for the case: u = u0 * exp(lambda * t), v = v0 * exp(lambda * t)
    u = u0 * np.exp(lambda_val * t)
    v = v0 * np.exp(lambda_val * t)

    # Plot trajectory
    plt.plot(u, v, linewidth=2, label=f'u_0 = {u0}, v_0 = {v0}')

    # Plot initial point as a dot
    plt.plot(u[0], v[0], 'ko', markersize=8, markerfacecolor='r')  # Black dot wi

    # Add arrows to indicate direction
    # Arrow placement and length scaling
    arrow_interval = round(len(t) / 10)  # Interval for arrow placement
    for j in range(0, len(t) - arrow_interval, arrow_interval):
        plt.quiver(u[j], v[j], u[j + arrow_interval] - u[j], v[j + arrow_interval
                   angles='xy', scale_units='xy', scale=1, color='k', linewidth=2

# Axis labels and title
plt.xlabel('u(t)')
plt.ylabel('v(t)')
plt.title(f'Trajectories of u(t) = u_0 * e^{{λt}} and v(t) = v_0 * e^{{λt}}, λ =
plt.legend()
plt.show()
```


Trajectories of u(t) = u_0 * e^{λt} and v(t) = v_0 * e^{λt}, λ = -0.5
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
v(t)
−1.0
−0.5
0.0
0.5
1.0
u(t)
u_0 = -1, v_0 = -1
u_0 = -1, v_0 = 1
u_0 = 1, v_0 = -1
u_0 = 1, v_0 = 1

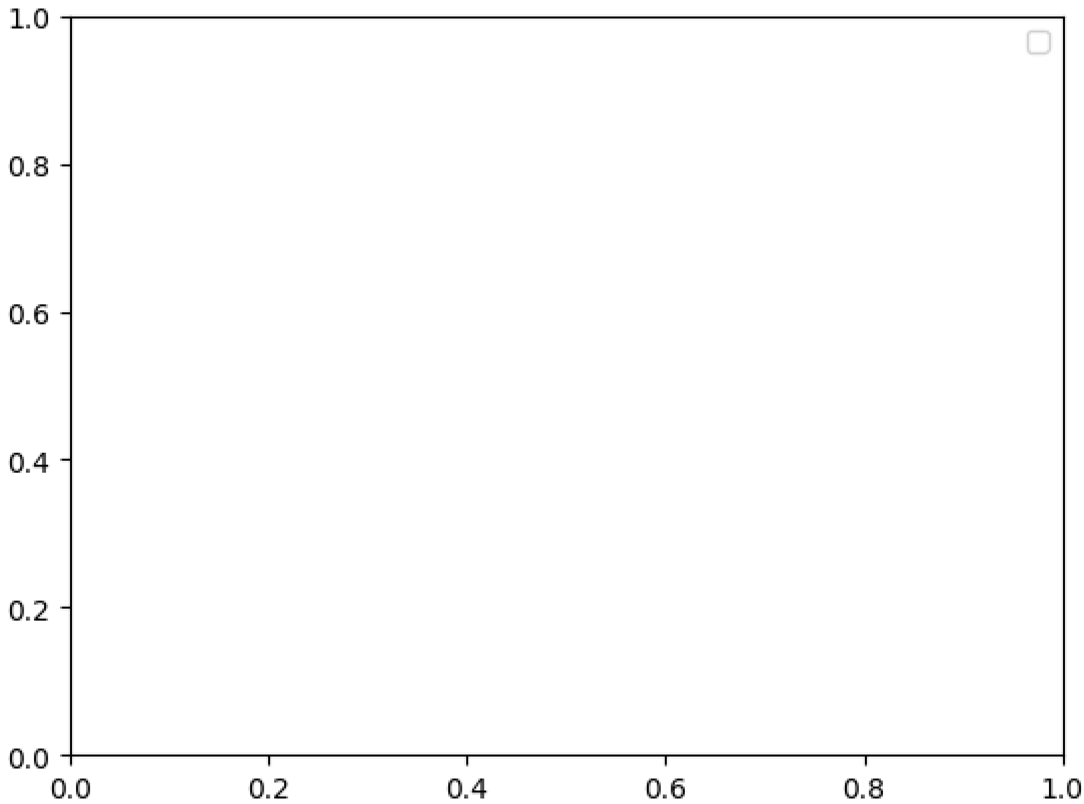
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
lambda_val = -0.5  # Lambda
t = np.linspace(0, 10, 20)  # Time vector

# λ < 0: Stable, spirals inward, orbits have a maximum or minimum u-value before
# λ > 0: Unstable, spirals outward from the origin.
# Type: Asymptotically stable (if λ < 0) or unstable (if λ > 0).

# Initial conditions for the trajectory
u0_vals = [-1, -1, 1, 1]  # Initial values of u in different quadrants
v0_vals = [-1, 1, -1, 1]  # Initial values of v in different quadrants

plt.figure()
plt.grid(True)
plt.axis('equal')  # Ensure the scale is the same in all directions

# Loop over each initial condition
for i in range(len(u0_vals)):
    u0 = u0_vals[i]
    v0 = v0_vals[i]

    # Compute v(t)
    v = v0 * np.exp(lambda_val * t)

    # Compute u(t)
    u = (u0 + v0 * t) * np.exp(lambda_val * t)

    # Plot trajectory
    plt.plot(u, v, linewidth=2, label=f'u_0 = {u0}, v_0 = {v0}')

    # Plot initial point as a dot
    plt.plot(u[0], v[0], 'ko', markersize=8, markerfacecolor='r')  # Black dot wi

    # Add arrows to indicate direction
    # Arrow placement and length scaling
    arrow_interval = round(len(t) / 10)  # Interval for arrow placement
    for j in range(0, len(t) - arrow_interval, arrow_interval):
        plt.quiver(u[j], v[j], u[j + arrow_interval] - u[j], v[j + arrow_interval
                   angles='xy', scale_units='xy', scale=1, color='k', linewidth=2

# Axis labels and title
plt.xlabel('u(t)')
plt.ylabel('v(t)')
plt.title(f'Trajectories of u(t) = (u_0 + v_0 * t) * e^{{λt}} and v(t) = v_0 * e^
plt.legend()
plt.show()
```


Trajectories of u(t) = (u_0 + v_0 * t) * e^{λt} and v(t) = v_0 * e^{λt}, λ = -0.5
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
v(t)
−1.0
−0.5
0.0
0.5
1.0
u(t)
u_0 = -1, v_0 = -1
u_0 = -1, v_0 = 1
u_0 = 1, v_0 = -1
u_0 = 1, v_0 = 1

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

# Parameters
lambda_val = 0.1  # Lambda (positive)
mu = -0.3         # Mu (negative)
t = np.linspace(0, 10, 20)  # Time vector

# λ > 0 > μ: Saddle point, u is unbounded while v tends to zero, orbits diverge a
# Type: Saddle point (always unstable).

# Initial conditions for the trajectory near the axes of four quadrants
u0_vals = [-1, -0.5, 0, 0.5, 1, -1, -0.5, 0.5, 1, 0]  # Initial values of u
v0_vals = [0, 0.5, 1, 0.5, -1, -0.5, -1, 0.5, 0, 1]   # Initial values of v

plt.figure()
plt.grid(True)
plt.axis('equal')  # Ensure the scale is the same in all directions

# Loop over each initial condition
for i in range(len(u0_vals)):
    u0 = u0_vals[i]
    v0 = v0_vals[i]

    # Compute u(t) and v(t)
    u = u0 * np.exp(lambda_val * t)
    v = v0 * np.exp(mu * t)

    # Plot trajectory
    plt.plot(u, v, linewidth=2, label=f'u_0 = {u0}, v_0 = {v0}')

    # Plot initial point as a dot
    plt.plot(u[0], v[0], 'ko', markersize=8, markerfacecolor='r')  # Black dot wi

    # Add arrows to indicate direction
    # Arrow placement and length scaling
    arrow_interval = round(len(t) / 10)  # Interval for arrow placement
    for j in range(0, len(t) - arrow_interval, arrow_interval):
        plt.quiver(u[j], v[j], u[j + arrow_interval] - u[j], v[j + arrow_interval
                   angles='xy', scale_units='xy', scale=1, color='k', linewidth=2

# Axis labels and title
plt.xlabel('u(t)')
plt.ylabel('v(t)')
plt.title(f'Trajectories of u(t) = u_0 * e^{{λt}} and v(t) = v_0 * e^{{μt}}, λ =
plt.show()

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




```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Not
```



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

# Parameters
beta = -1  # Beta value (can be positive (clockwise) or negative (counterclockwis
t = np.linspace(0, 10, 150)  # Time vector

# Initial conditions for the trajectory
initial_conditions = np.array([
    [1, 0],
    [-1, 1],
    [-0.5, -0.5]
])

# Preallocate for magnitudes
magnitudes = np.zeros(len(initial_conditions))

# Compute magnitudes
for i in range(len(initial_conditions)):
    A = initial_conditions[i, 0]
    B = initial_conditions[i, 1]

    # Compute u(t) and v(t) for a specific initial condition
    u = A * np.cos(beta * t) + B * np.sin(beta * t)
    v = -A * np.sin(beta * t) + B * np.cos(beta * t)

    # Compute magnitude
    magnitudes[i] = np.mean(u**2 + v**2)  # Use mean to represent the magnitude

# Find unique magnitudes and their indices
unique_magnitudes, idx = np.unique(magnitudes, return_index=True)

# Only keep unique initial conditions
unique_conditions = initial_conditions[idx, :]

plt.figure()
plt.grid(True)
plt.axis('equal')  # Ensure the scale is the same in all directions

# Plot only unique magnitudes
for i in range(len(unique_conditions)):
    A = unique_conditions[i, 0]
    B = unique_conditions[i, 1]

    # Compute u(t) and v(t) for unique conditions
    u = A * np.cos(beta * t) + B * np.sin(beta * t)
    v = -A * np.sin(beta * t) + B * np.cos(beta * t)

    # Plot trajectory
    plt.plot(u, v, linewidth=2, label=f'A = {A}, B = {B}')

    # Plot initial point as a dot
    plt.plot(u[0], v[0], 'ko', markersize=8, markerfacecolor='r')  # Black dot wi

    # Add arrows to indicate direction with invisible arrow rods
```

```
        arrow_interval = round(len(t) / 10)  # Interval for arrow placement
        for j in range(0, len(t) - arrow_interval, arrow_interval):
            # Calculate the direction vector
            dx = u[j + arrow_interval] - u[j]
            dy = v[j + arrow_interval] - v[j]

            # Add invisible arrows
            plt.quiver(u[j], v[j], dx, dy, angles='xy', scale_units='xy', scale=1,
                       color=[0, 0, 0, 0], linewidth=0.5, headwidth=3, headlength=5)

    # Axis labels and title
    plt.xlabel('u(t)')
    plt.ylabel('v(t)')
    plt.title(f'Trajectories with Unique Magnitudes of u^2 + v^2, β = {beta}')

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