# Animation using Matplotlib

To create and display animations in Google Colab using Matplotlib, you need to ensure that the necessary backend is configured properly. The FuncAnimation in Matplotlib may require some additional setup for rendering animations directly in Jupyter Lab. Here's how you can modify the lab exercise to work correctly in Jupyter Lab.

## Lab Exercise: Animations in Matplotlib (using Google Colab)

Objective

Learn how to create and customize animations using Matplotlib

**Requirements**

- Python 3.x
- Matplotlib library
- Numpy library
- Google Colab

**Steps**

Install Required Libraries (if not already installed):

```
pip install matplotlib numpy ipympl
```

**Load the Required Libraries and Enable the Interactive Backend:**

Use the %matplotlib widget magic to enable interactive plotting.

```
import numpy as np

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

from google.colab import output

output.enable_custom_widget_manager()

%matplotlib widget
```

## Generate Sample Data:

Generate some sample data to be used in the animation.

```
# Generate sample data
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)
```

## Create a Basic Animation:

Create a simple animation that shows a sine wave.

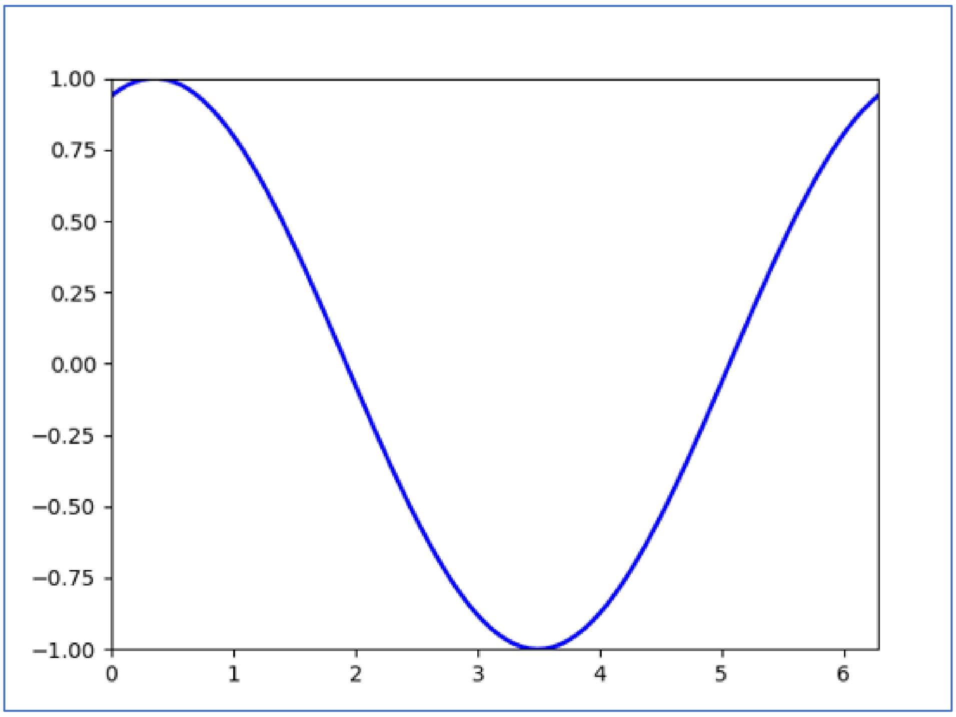
```
fig, ax = plt.subplots()
line, = ax.plot([], [], 'b-', lw=2)
ax.set_xlim(0, 2 * np.pi)
ax.set_ylim(-1, 1)

# Initialization function
def init():
    line.set_data([], [])
    return line,

# Animation function
def animate(i):
    x = np.linspace(0, 2 * np.pi, 100)
    y = np.sin(x + 0.1 * i)
    line.set_data(x, y)
    return line,

# Create animation
anim = FuncAnimation(fig, animate, init_func=init, frames=100, interval=20,
blit=True)

# Display the animation
plt.show()
```



### Task: Customizing the Animation

Explanation: Customize the animation to include additional features like titles, labels, and a second animated element.

**Task 1: Adding Titles and Labels:**

Add a title and labels to the plot.

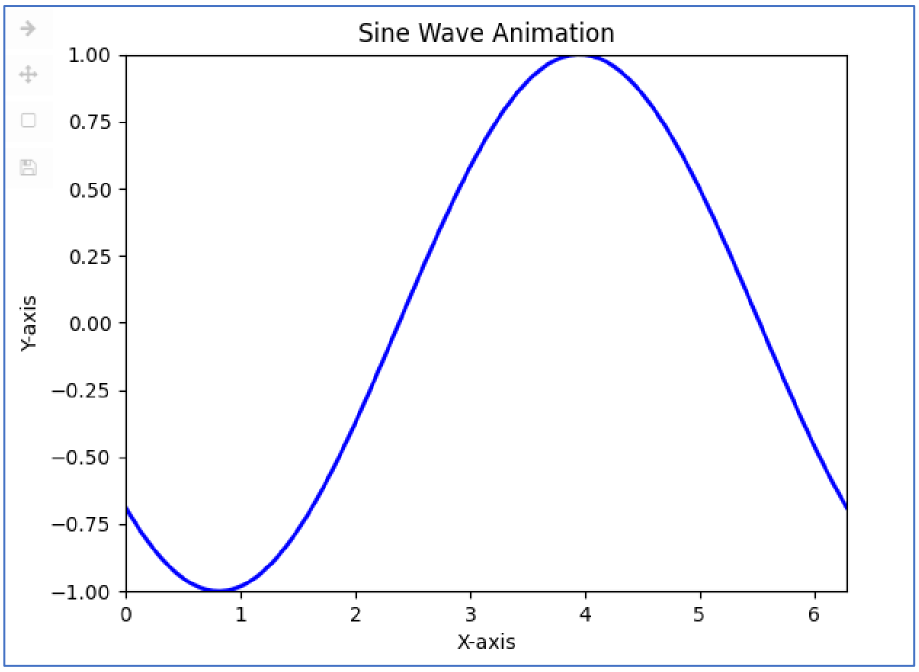
```
fig, ax = plt.subplots()
line, = ax.plot([], [], 'b-', lw=2)
ax.set_xlim(0, 2 * np.pi)
ax.set_ylim(-1, 1)
ax.set_title('Sine Wave Animation')
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')

# Initialization function
def init():
   line.set_data([], [])
   return line,

# Animation function
def animate(i):
   x = np.linspace(0, 2 * np.pi, 100)
   y = np.sin(x + 0.1 * i)
   line.set_data(x, y)
   return line,

# Create animation
anim = FuncAnimation(fig, animate, init_func=init, frames=100, interval=20,
blit=True)

# Display the animation
plt.show()
```


Sine Wave Animation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Y-axis
0
1
2
3
4
5
6
X-axis

## Task 2: Adding a Second Animated Element:

Add a cosine wave to the animation.

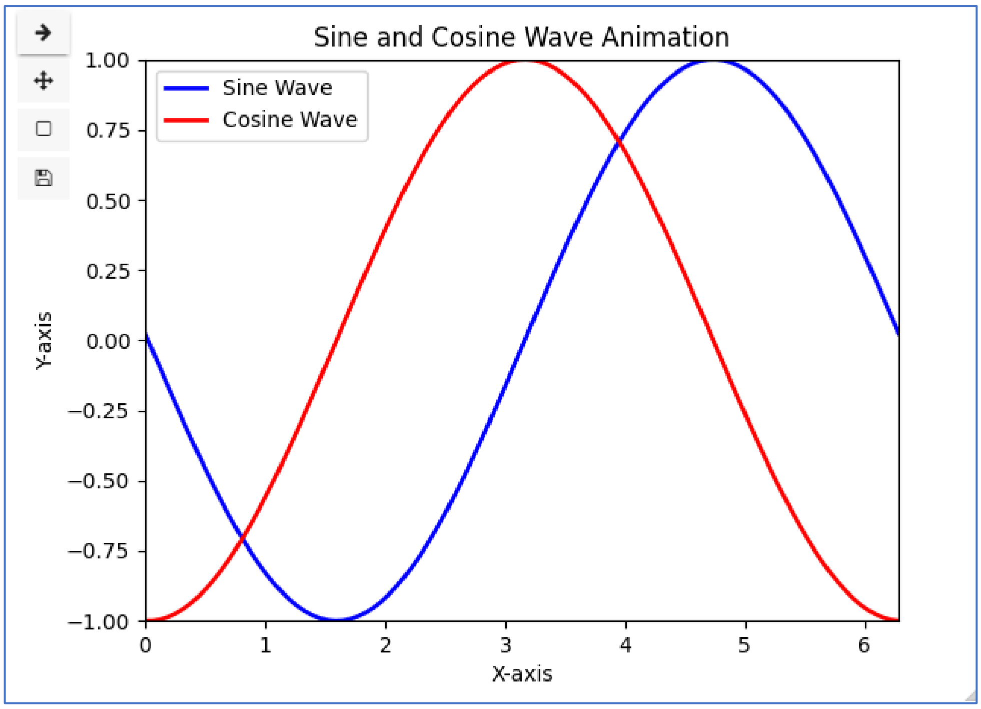
```
fig, ax = plt.subplots()
line1, = ax.plot([], [], 'b-', lw=2, label='Sine Wave')
line2, = ax.plot([], [], 'r-', lw=2, label='Cosine Wave')
ax.set_xlim(0, 2 * np.pi)
ax.set_ylim(-1, 1)
ax.set_title('Sine and Cosine Wave Animation')
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.legend()

# Initialization function
def init():
   line1.set_data([], [])
   line2.set_data([], [])
   return line1, line2

# Animation function
def animate(i):
   x = np.linspace(0, 2 * np.pi, 100)
   y1 = np.sin(x + 0.1 * i)
   y2 = np.cos(x + 0.1 * i)
   line1.set_data(x, y1)
   line2.set_data(x, y2)
   return line1, line2

# Create animation
anim = FuncAnimation(fig, animate, init_func=init, frames=100, interval=20,
blit=True)

# Display the animation
plt.show()
```


Sine and Cosine Wave Animation
Sine Wave
Cosine Wave
Y-axis
X-axis
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6

## Task 3: Save the Animation:

Save the animation as an MP4 file.

```
fig, ax = plt.subplots()
line1, = ax.plot([], [], 'b-', lw=2, label='Sine Wave')
line2, = ax.plot([], [], 'r-', lw=2, label='Cosine Wave')
ax.set_xlim(0, 2 * np.pi)
ax.set_ylim(-1, 1)
ax.set_title('Sine and Cosine Wave Animation')
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.legend()

# Initialization function
def init():
    line1.set_data([], [])
    line2.set_data([], [])
    return line1, line2

# Animation function
def animate(i):
    x = np.linspace(0, 2 * np.pi, 100)
    y1 = np.sin(x + 0.1 * i)
    y2 = np.cos(x + 0.1 * i)
    line1.set_data(x, y1)
    line2.set_data(x, y2)
    return line1, line2

# Create animation
anim = FuncAnimation(fig, animate, init_func=init, frames=100, interval=20,
blit=True)

# Save the animation
anim.save('sine_cosine_wave.mp4', writer='ffmpeg')

# Display the animation
plt.show()
```

## Explanation

**Generating Sample Data:**

- numpy.linspace creates evenly spaced values over a specified range.
- numpy.sin and numpy.cos compute the trigonometric sine and cosine of an array element-wise.

## Creating a Basic Animation:

- %matplotlib widget enables interactive plotting in Jupyter Lab.
- FuncAnimation is used to create the animation. It takes the figure, the animation function, the initialization function, the number of frames, the interval between frames, and whether to use blitting for better performance.
- init function initializes the plot.
- animate function updates the plot for each frame.

**Customizing the Animation:**

- Titles and labels are added using set_title, set_xlabel, and set_ylabel.
- A second animated element (cosine wave) is added by creating another line and updating it in the animate function.
- The animation is saved as an MP4 file using anim.save.

By completing these tasks, you will learn how to create and customize animations using Matplotlib, making your data visualizations more dynamic and engaging, especially in a Jupyter Lab environment..