**Data Visualization**

**Aim:** Customize Matplotlib plots with interactive widgets (e.g., sliders, buttons) using libraries like ipywidgets.

We can customize Matplotlib plots with interactive widgets such as sliders and buttons by using libraries like ipywidgets in Jupyter notebooks or Google Colab. This allows you to dynamically update the plots based on user input, which is particularly useful for exploratory data analysis and interactive data visualization.

**Step-by-step Process to integrate ipywidgets with Matplotlib:**

**1. Install the necessary libraries**

pip install ipywidgets matplotlib

**2. Import the required libraries**

import matplotlib.pyplot as plt

import numpy as np

from ipywidgets import interact, FloatSlider

import ipywidgets as widgets

### 3. Create a simple plot with interactive widgets

let’s create a sine wave and add a slider to control the frequency of the wave:

def plot\_sine\_wave(frequency):

x = np.linspace(0, 2 \* np.pi, 1000)

y = np.sin(frequency \* x)

plt.figure(figsize=(8, 4))

plt.plot(x, y)

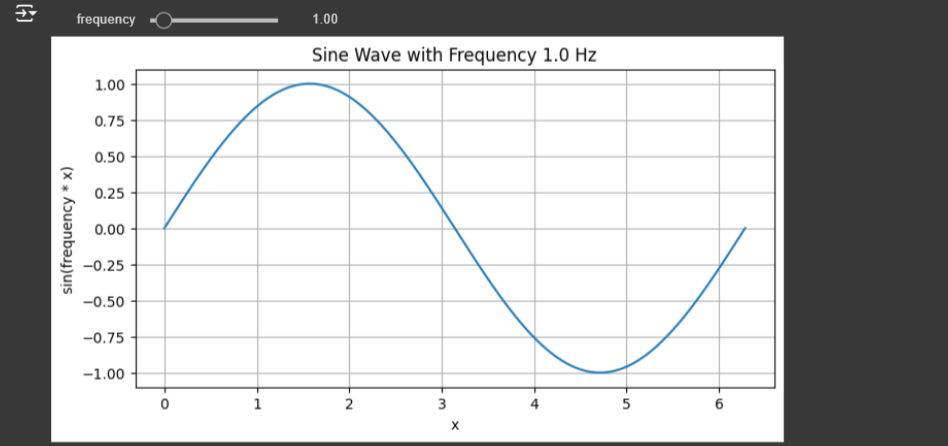
plt.title(f'Sine Wave with Frequency {frequency} Hz')

plt.xlabel('x')

plt.ylabel('sin(frequency \* x)')

plt.grid(True)

plt.show()

interact(plot\_sine\_wave, frequency=FloatSlider(min=0.1, max=10, step=0.1, value=1));

### 4. Adding more interactivity with buttons

We can also use buttons to trigger specific events or update plots. Here's an example where a button is used to regenerate a plot

def plot\_noisy\_sine\_wave(noise\_amplitude):

x = np.linspace(0, 2 \* np.pi, 1000)

y = np.sin(x) + noise\_amplitude \* np.random.randn(1000)

plt.figure(figsize=(8, 4))

plt.plot(x, y)

plt.title(f'Sine Wave with Noise Amplitude {noise\_amplitude}')

plt.xlabel('x')

plt.ylabel('sin(x) + noise')

plt.grid(True)

plt.show()

noise\_slider = FloatSlider(min=0, max=1, step=0.05, value=0.1, description="Noise")

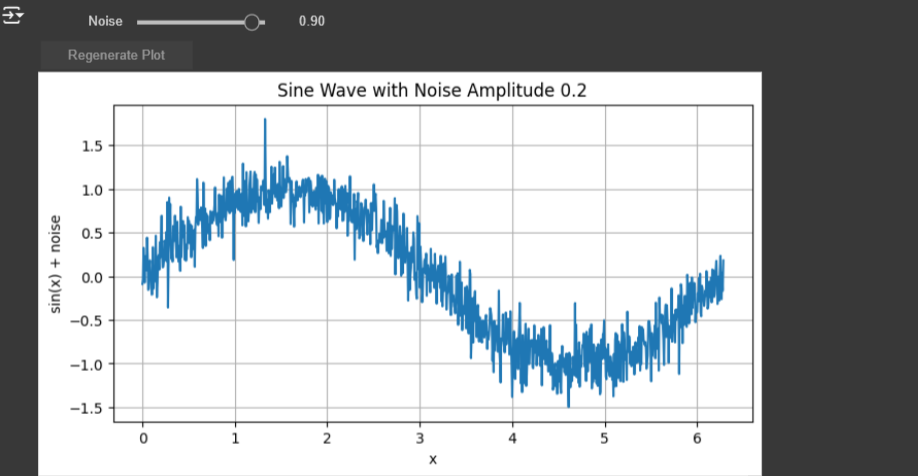
button = widgets.Button(description="Regenerate Plot")

def on\_button\_click(b):

plot\_noisy\_sine\_wave(noise\_slider.value)

display(noise\_slider, button)

button.on\_click(on\_button\_click)



### 5. Combine multiple widgets

You can combine sliders, buttons, and other widgets to create complex interactive dashboards. For instance, controlling both the amplitude and frequency of a wave:

def plot\_custom\_sine\_wave(amplitude, frequency):

x = np.linspace(0, 2 \* np.pi, 1000)

y = amplitude \* np.sin(frequency \* x)

plt.figure(figsize=(8, 4))

plt.plot(x, y)

plt.title(f'Sine Wave with Amplitude {amplitude} and

Frequency {frequency} Hz')

plt.xlabel('x')

plt.ylabel(f'{amplitude} \* sin({frequency} \* x)')

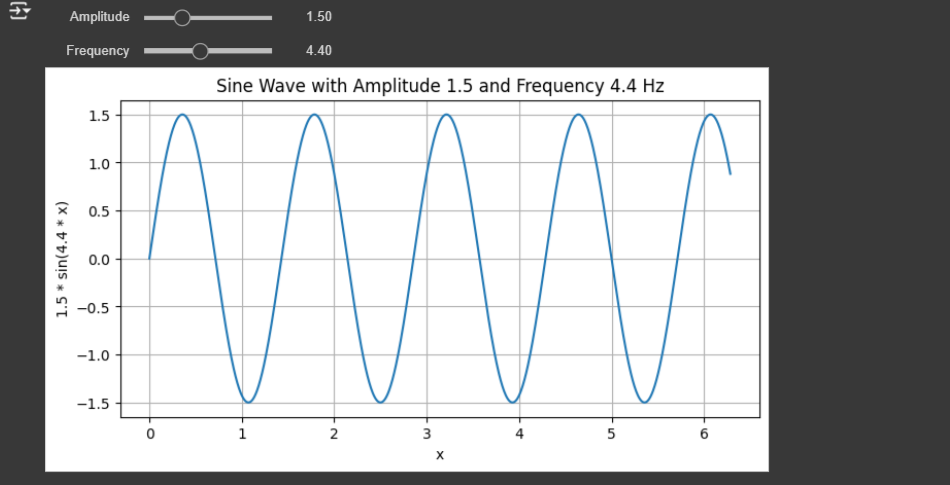
plt.grid(True)

plt.show()

amplitude\_slider = FloatSlider(min=0.1, max=5, step=0.1, value=1, description="Amplitude")

frequency\_slider = FloatSlider(min=0.1, max=10, step=0.1, value=1, description="Frequency")

interact(plot\_custom\_sine\_wave, amplitude=amplitude\_slider, frequency=frequency\_slider);



### 6. More complex interactivity

You can extend the interactivity by using ipywidgets's more complex components such as dropdown menus, checkboxes, and text inputs. This gives you the flexibility to create rich and intuitive data exploration tools.

Let's enhance the previous interactive sine wave plot by adding a button that resets the sliders to their default values (amplitude = 1, frequency = 1). The button will trigger the reset action when clicked.

Here's the complete updated code:

import matplotlib.pyplot as plt

import numpy as np

from ipywidgets import interact, FloatSlider, Button, VBox

import ipywidgets as widgets

def plot\_custom\_sine\_wave(amplitude, frequency):

x = np.linspace(0, 2 \* np.pi, 1000)

y = amplitude \* np.sin(frequency \* x)

plt.figure(figsize=(8, 4))

plt.plot(x, y)

plt.title(f'Sine Wave with Amplitude {amplitude} and

Frequency {frequency} Hz')

plt.xlabel('x')

plt.ylabel(f'{amplitude} \* sin({frequency} \* x)')

plt.grid(True)

plt.show()

amplitude\_slider = FloatSlider(min=0.1, max=5, step=0.1, value=1,

description="Amplitude")

frequency\_slider = FloatSlider(min=0.1, max=10, step=0.1, value=1,

description="Frequency")

def reset\_sliders(button):

amplitude\_slider.value = 1

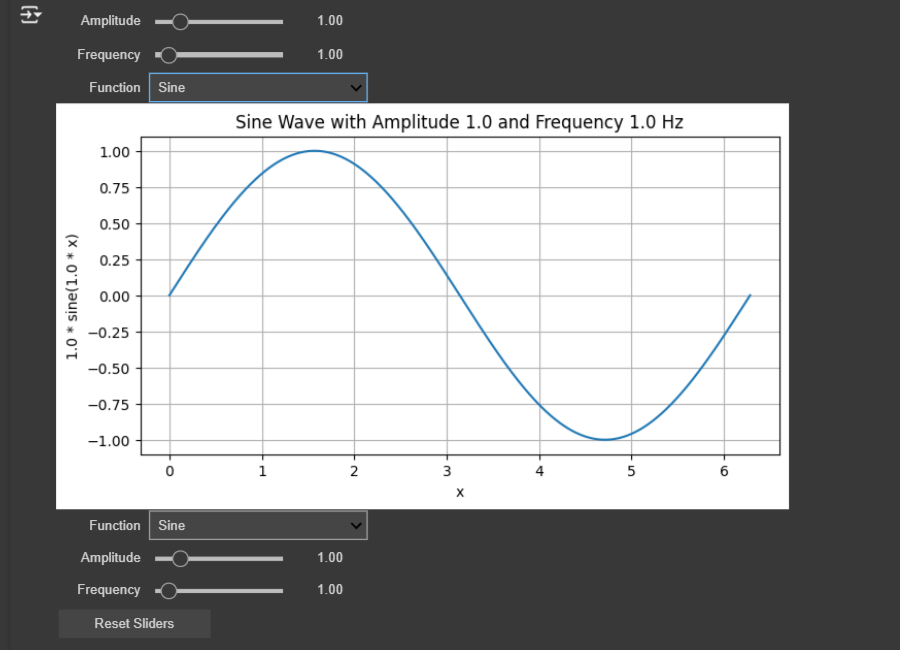
frequency\_slider.value = 1

reset\_button = Button(description="Reset Sliders")

reset\_button.on\_click(reset\_sliders)

interactive\_plot = interact(plot\_custom\_sine\_wave, amplitude=amplitude\_slider, frequency=frequency\_slider)

display(VBox([amplitude\_slider, frequency\_slider, reset\_button]))



### Working:

* The plot\_custom\_sine\_wave function dynamically updates the plot based on the values of the amplitude and frequency sliders.
* The reset button (reset\_button) is programmed to reset both sliders to their default values (1 for amplitude and frequency) when clicked.
* The VBox layout from ipywidgets is used to stack the sliders and the button vertically for a clean layout.

Now, when the user clicks the "Reset Sliders" button, the sliders will return to their default values, and the plot will update accordingly.

Now we’ll add a dropdown menu to select different mathematical functions (e.g., sine, cosine, and tangent) in addition to the existing sliders and reset button. The plot will update according to the chosen function from the dropdown.

import matplotlib.pyplot as plt

import numpy as np

from ipywidgets import interact, FloatSlider, Button, Dropdown, VBox

import ipywidgets as widgets

**Function to plot the selected mathematical wave (sine, cosine, tangent)**:

def plot\_math\_wave(amplitude, frequency, function\_type):

x = np.linspace(0, 2 \* np.pi, 1000)

# Choose the correct function based on the dropdown selection

if function\_type == 'Sine':

y = amplitude \* np.sin(frequency \* x)

elif function\_type == 'Cosine':

y = amplitude \* np.cos(frequency \* x)

elif function\_type == 'Tangent':

y = amplitude \* np.tan(frequency \* x)

y = np.clip(y, -10, 10) # Clip tangent values to avoid huge spikes in the plot

# Plot the selected function

plt.figure(figsize=(8, 4))

plt.plot(x, y)

plt.title(f'{function\_type} Wave with Amplitude {amplitude} and Frequency {frequency} Hz')

plt.xlabel('x')

plt.ylabel(f'{amplitude} \* {function\_type.lower()}({frequency} \* x)')

plt.grid(True)

plt.show()

**sliders for amplitude and frequency:**

amplitude\_slider = FloatSlider(min=0.1, max=5, step=0.1, value=1, description="Amplitude")

frequency\_slider = FloatSlider(min=0.1, max=10, step=0.1, value=1, description="Frequency")

**Dropdown menu for selecting the function type (sine, cosine, tangent):**

function\_dropdown = Dropdown(

options=['Sine', 'Cosine', 'Tangent'],

value='Sine', # Default value

description='Function',

)

**Reset function to set sliders and dropdown to default values:**

def reset\_sliders(button):

amplitude\_slider.value = 1

frequency\_slider.value = 1

function\_dropdown.value = 'Sine'

**Create a button to reset sliders and dropdown:**

reset\_button = Button(description="Reset Sliders")

**Attach the reset function to the button's click event:**

reset\_button.on\_click(reset\_sliders)

**The interactive plot:**

interactive\_plot = interact(plot\_math\_wave, amplitude=amplitude\_slider, frequency=frequency\_slider, function\_type=function\_dropdown)

**Arranging the sliders, dropdown, and button vertically :**

display(VBox([function\_dropdown, amplitude\_slider, frequency\_slider, reset\_button]))

### Key additions:

1. **Dropdown menu** (function\_dropdown):
   * Allows the user to select from 'Sine', 'Cosine', or 'Tangent'.
   * Updates the plot to reflect the selected function.
2. plot\_math\_wave **function**:
   * Updated to plot different mathematical functions based on the dropdown selection.
   * For the tangent function, I added a clipping (np.clip) to avoid extremely large values that can cause the plot to go out of bounds.
3. **Reset function**:
   * The reset button now resets both the sliders and the dropdown to their default values (amplitude = 1, frequency = 1, function = 'Sine').

Now, the user can:

* Adjust amplitude and frequency using the sliders.
* Select which mathematical function (sine, cosine, tangent) to plot using the dropdown.
* Reset all controls to default values using the reset button.