Plotting Multiple Curves

In this lesson, we will learn some different ways of plotting multiple graphs and how to label them.

WE'LL COVER THE FOLLOWING

In this course, we will be using the object-oriented approach when working with `matplotlib`.The main idea with object-oriented programming is to have objects that one can apply functions and actions to, and no object or program states should be global. The real advantage of the object-oriented approach in plotting becomes apparent when more than one figure is created, or when a figure contains more than one plot.

Multiple curves on the same graph

We can draw multiple curves on the same graph by repeatedly using the `plot` command.

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

x = np.linspace(0, 2 * np.pi , 100)
y1 = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y1)     # curve 1
plt.plot(x, y2)     # curve 2
plt.plot(x, y1 + y2)       # curve 3
```

> 💡 If you don't specify a color for a plotting statement, `matplotlib` will use its default colors. The first three default colors are special shades of blue, orange and green.

You can find more colors using the command `help("matplotlib.pyplot.plot") below:

```
import matplotlib.pyplot as plt

help("matplotlib.pyplot.plot")
```

Adding legends

When plotting multiple curves on the same graph, it is imperative to add a legend to make the graph more readable. There are two ways to add legends:

1. Use the `legend()` method of the axis object and pass a list of legend texts for the curves plotted previously.

```
plt.plot(x1, y1)
plt.plot(x2, y2)
plt.legend(["curve 1", "curve 2"])
```

2. Use the `label = "curve 1"` keyword argument when plots are added to the figure, and then use the `legend()` method without arguments to add the legend to the figure.

```
plt.plot(x1, y1, label="curve 1")
plt.plot(x2, y2, label="curve 2")
legend()
```

While method 1 makes the code look cleaner, method 2 removes the hassle of separately changing the legend when making changes to the graph. It is entirely up to the user which method they choose.

The legend function takes an optional keyword argument `loc` that can be used to specify where in the figure the legend is to be drawn.

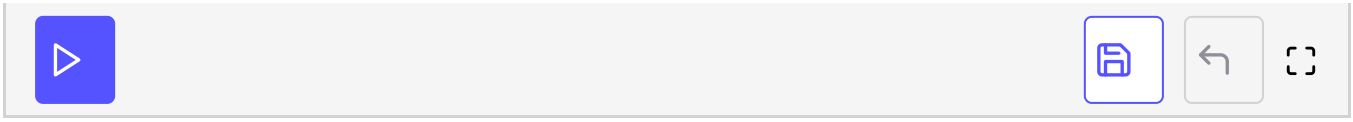
```
plt.legend(loc=0) # let matplotlib decide the optimal location
plt.legend(loc=1) # upper right corner
plt.legend(loc=2) # upper left corner
plt.legend(loc=3) # lower left corner
plt.legend(loc=4) # lower right corner
```

The example below uses method 1 for implementing legends.

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

x = np.linspace(0, 2 * np.pi , 100)
y1 = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y1)
plt.plot(x, y2)
plt.plot(x, y1 + y2)
plt.legend(["sin(x)", "cos(x)", "sin(x) + cos(x)"], loc=2)  # legend at upper left corner
```



Making a new figure

Whenever there is a `plot` statement, a figure with a default size is automatically created, and all subsequent plotting statements in the code are added to the same figure. If you want the figure to be a different size, you can create a figure of the desired size first using the `plt.figure(figsize=(width, height))` syntax.

Any subsequent plotting statement in the code is then added to the figure. You can even create a second figure or third or fourth and so on.

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

plt.figure(figsize=(10, 3))
plt.plot([1, 2, 3], [2, 4, 3], linewidth = 6)
plt.title('very wide figure')
plt.savefig('output/fig1.png') # do not change, saving to output folder

plt.figure()  # new figure of default size
plt.plot([1, 2, 3], [1, 3, 1], 'r')
plt.title('second figure');
```

```
plt.savefig('output/fig2.png')   # do not change, saving to output folder
```

Subplots

We can use many axis layout managers in `matplotlib` if we do not care about the positioning of the axis in the figure window. The most commonly used is `subplot`:

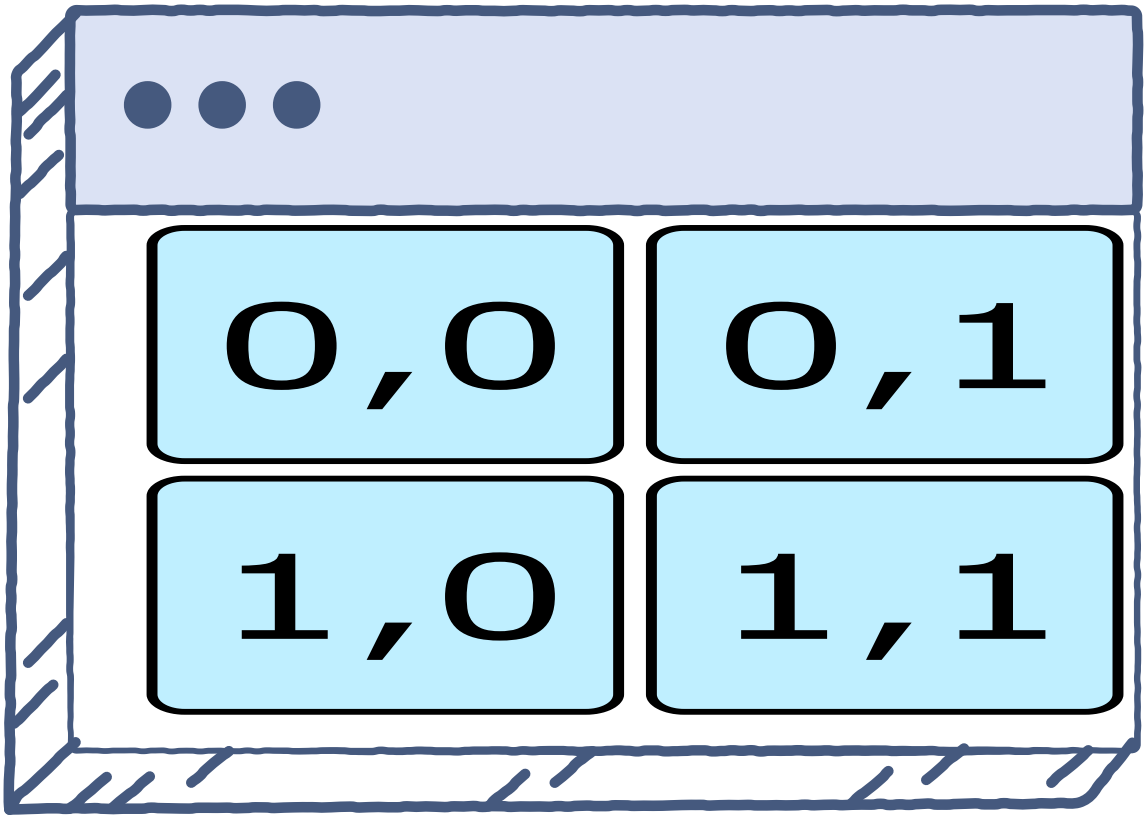
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 *np.pi , 100)
y1 = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y1)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('y = sin(x)')
```

Using the subplots, we can plot more than one graph on the same figure by dividing the figure into a grid. We need to specify the number of rows and columns in the `subplots()` function.

```
plt.subplots(rows, columns)
```



Plot numbers for a 2x2 grid

This `plt.subplots(rows, columns)` returns an `ndarray` with `subplot` objects in it. Each subplot can be accessed using its corresponding index. We can also define the value for the `figsize` property as one of the arguments of the `subplot()` command. Let's look at an example of this below:

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

x = np.linspace(0, 2 * np.pi , 100)
y1 = np.sin(x)
y2 = np.cos(x)

fig, ax = plt.subplots(2, 2, figsize=(9, 5)) # creates a 2x2 grid

ax[0, 0].plot(x, y1) # plots on [0, 0] block on the grid
ax[0, 0].set_title("sin(x)")

ax[0, 1].plot(x, y2) # plots on the [0, 1] block on the grid
ax[0, 1].set_title("cos(x)")

ax[1, 0].plot(x, y1 + y2) # plots on [1, 0] block on the grid
ax[1, 0].set_title("sin(x) + cos(x)")

ax[1, 1].plot(x, y1 - y2) # plots on the [1, 1] block on the grid
ax[1, 1].set_title("sin(x) - cos(x)")
```

```
fig.tight_layout()
```

You might notice the `fig.tight_layout()` command in line 22.

> 💡 `fig.tight_layout()` automatically adjusts the position of the axes on the figure so that there is no overlapping content.

Remove this line and see what happens.

Alternate method

We can also use the `add_subplot()` function to add subplots to a figure. This alternate method gives us the flexibility to specify the type of axes for each subplot. We will see examples of this in the next couple of lessons. For now, let's understand the syntax:

```
fig.add_subplot(nrows, ncols, index)
```

- `nrows` is the number of rows
- `ncols` is the number of columns
- `index` is the index of the subplot in the grid.

```
fig.add_subplot(1, 2, 1)
```

can also be written as the following for simplicity purposes:

```
fig.add_subplot(121)
```

Let's see an implementation of this below:

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

x = np.linspace(0, 2 * np.pi , 100)
y1 = np.sin(x)
y2 = np.cos(x)
```

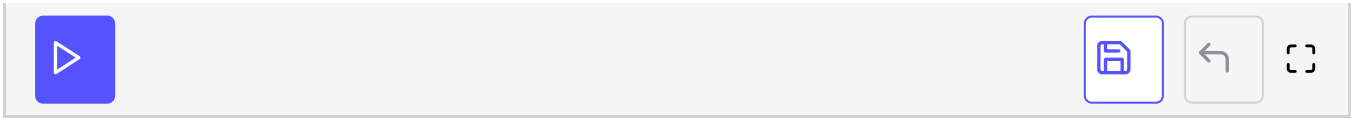
```
fig = plt.figure(figsize=(9, 5)) # creates a figure
# we are using a 2x2 grid
ax1 = fig.add_subplot(221)      # creates axis at index 1
ax1.plot(x, y1)
ax1.set_title("sinx(x)")

ax2 = fig.add_subplot(222)      # creates axis at index 2
ax2.plot(x, y2)
ax2.set_title("cos(x)")

ax3 = fig.add_subplot(223)      # creates axis at index 3
ax3.plot(x, y1 + y2)
ax3.set_title("sin(x) + cos(x)")

ax4 = fig.add_subplot(224)       # creates axis at index 4
ax4.plot(x, y1 - y2)
ax4.set_title("sin(x) - cos(x)")

fig.tight_layout()
```



In the next lesson, we will learn about setting up the axes.