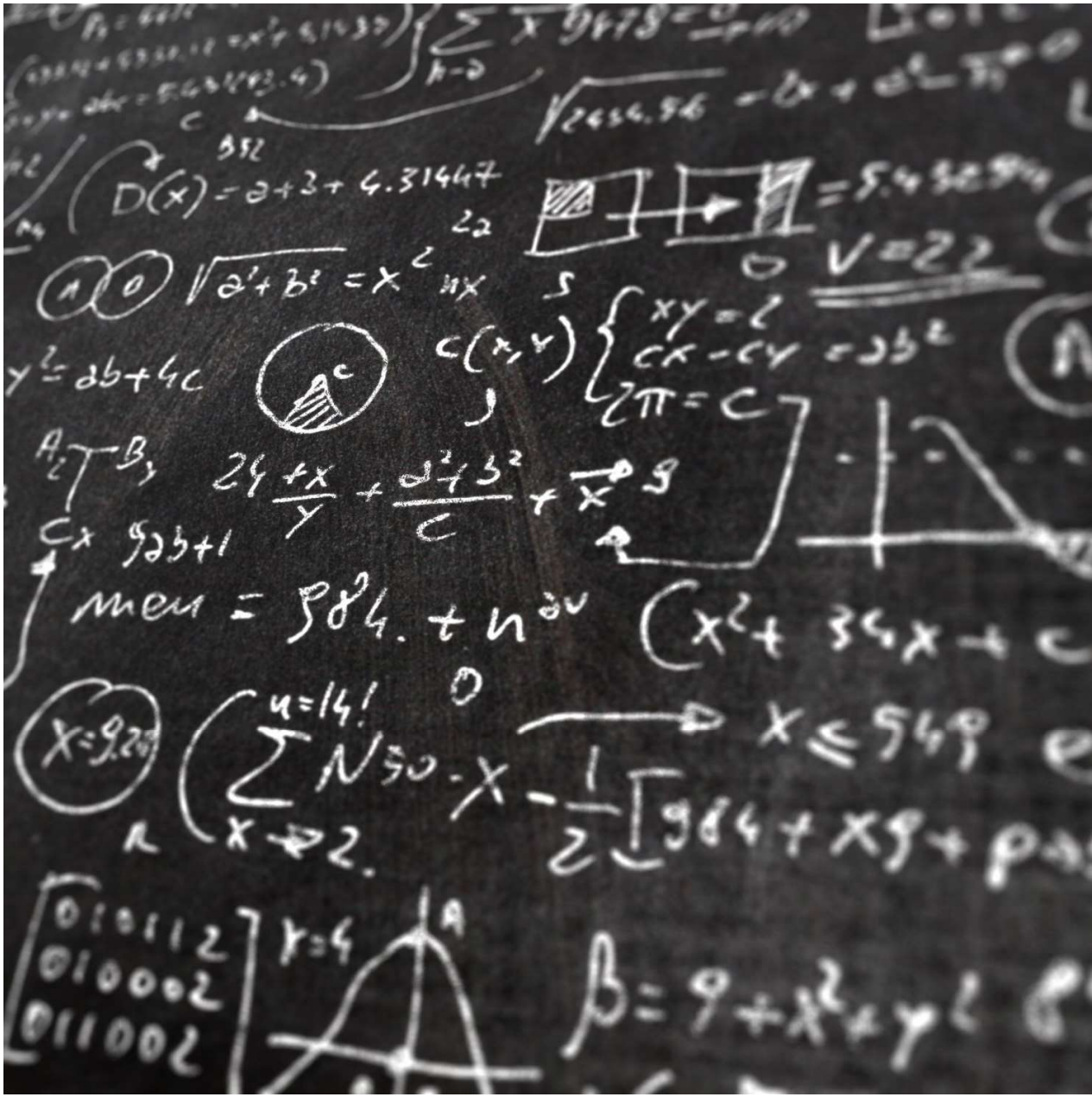
Mathematical Plotting with Matplotlib



Requirements

pip install matplotlib

Try if your installation is fine:

import matplotlib.pyplot as plt

Before plotting we need values

+ To plot a 2D graph we need values for X and Y axis.

+ Lets plot our function:

  primary * (1+r/100)n**

Functions in Python

```
def f(n, r, primary):
        return primary * (1+r/100)**n
```

Plotting in Python

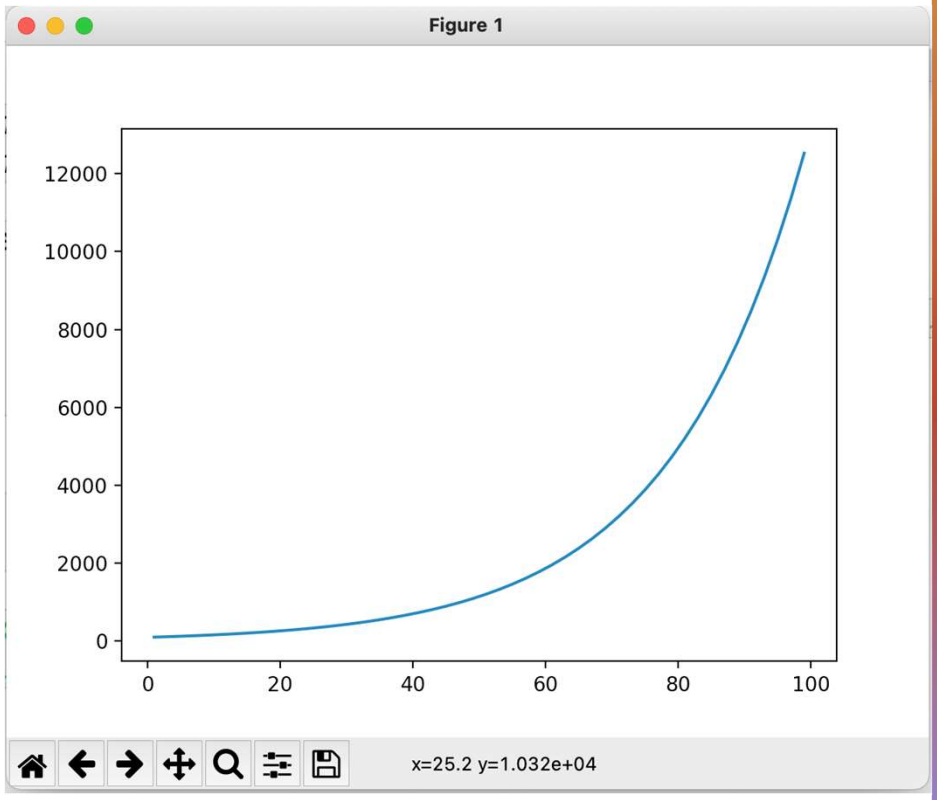
+ We have defined the function to calculate our Y values.
+ Now we need to generate the values for X axis.
+ Lets plot the evolution of the balance from 1 to 99 years

```
x = np.linspace(1, 99) # 50 values
```

numpy.linspace

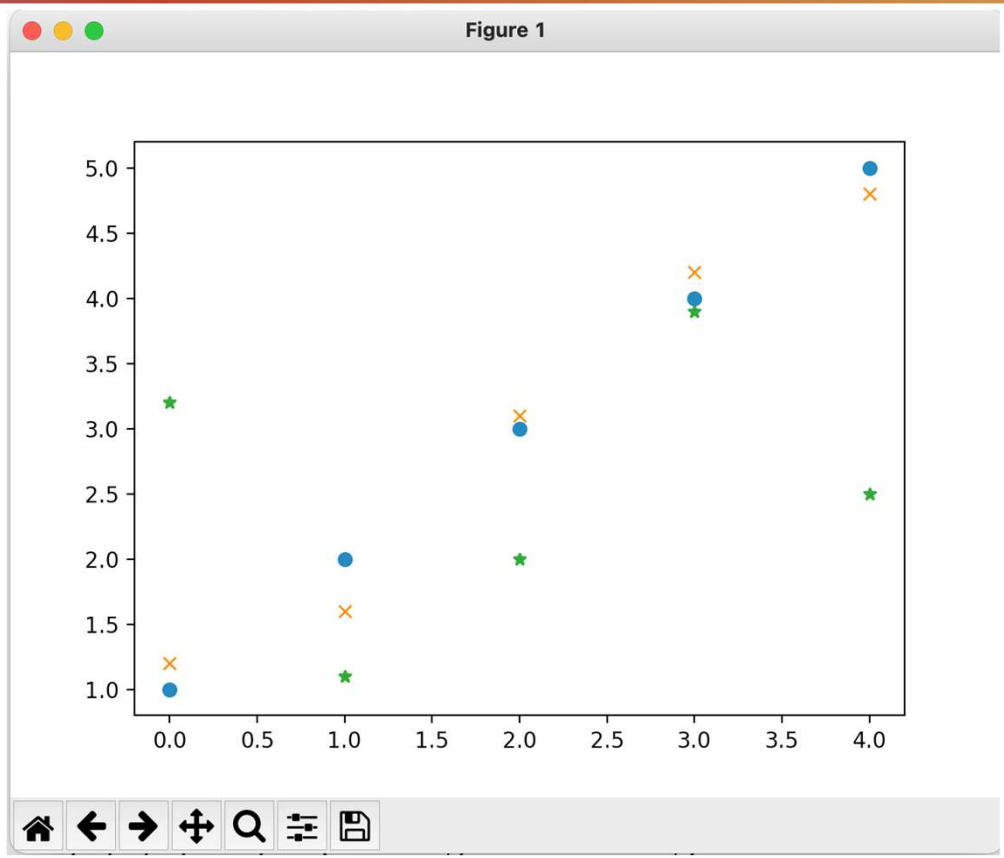
- numpy.**linspace**(*start*, *stop*, *num=50*, *endpoint=True*, *retstep=False*, *dtype=None*, *axis=0*)
- Return evenly spaced numbers over a specified interval.
- Returns *num* evenly spaced samples, calculated over the interval [*start*, *stop*].
- The endpoint of the interval can optionally be excluded.

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

def f(n, r, primary):
        return primary * (1+r/100)**n
r = 5
primary = 100
x = np.linspace(1, 99)
y = f(x, r, primary)
ax = plt.plot(x, y)
plt.show()
```



Plotting style

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

y1 = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y2 = np.array([1.2, 1.6, 3.1, 4.2, 4.8])
y3 = np.array([3.2, 1.1, 2.0, 3.9, 2.5])

fig, ax = plt.subplots()
lines = ax.plot(y1, 'o', y2, 'x', y3, '*')
plt.show()
```



Note: Plotting according their position in the array

Adding labels and legends

Add title

ax.set_title("Plot of the data y1, y2, and y3")

Add axis label

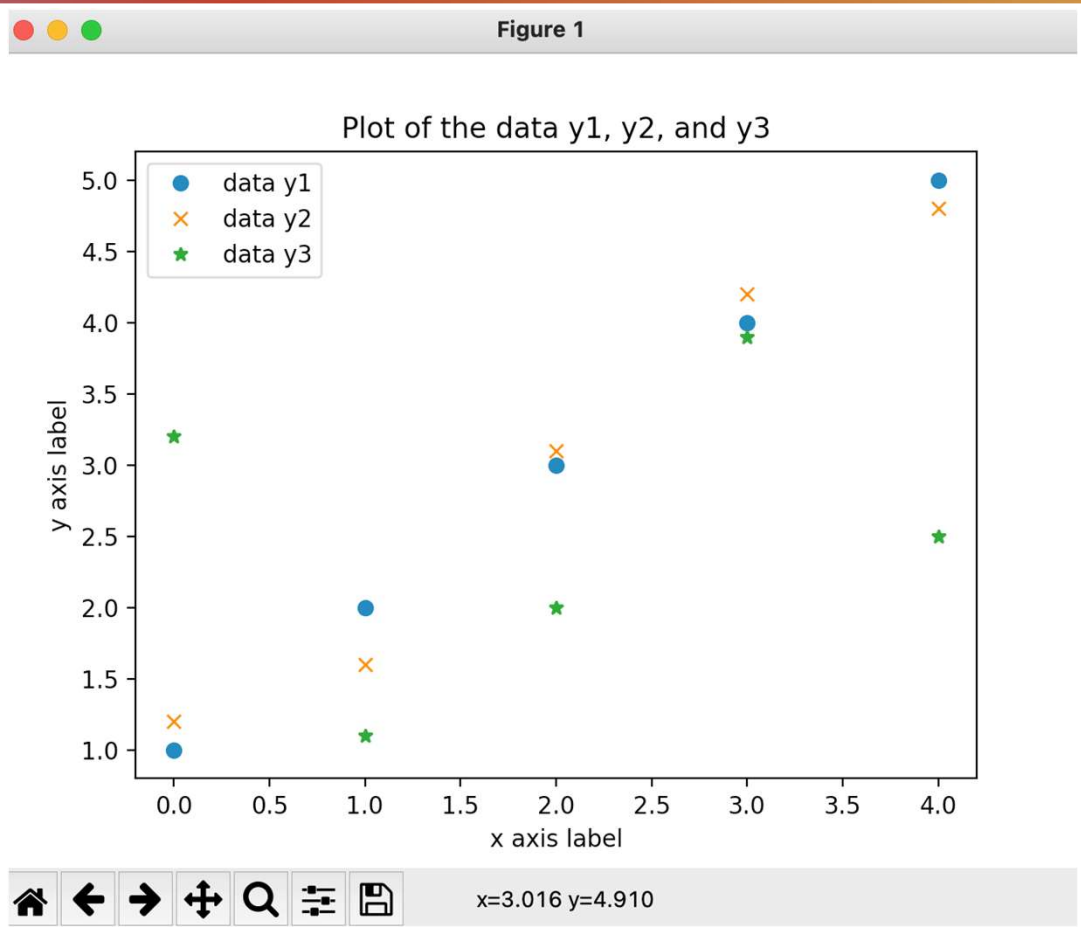
ax.set_xlabel("x axis label")

ax.set_ylabel("y axis label")

Add legend

ax.legend(("data y1", "data y2", "data y3"))

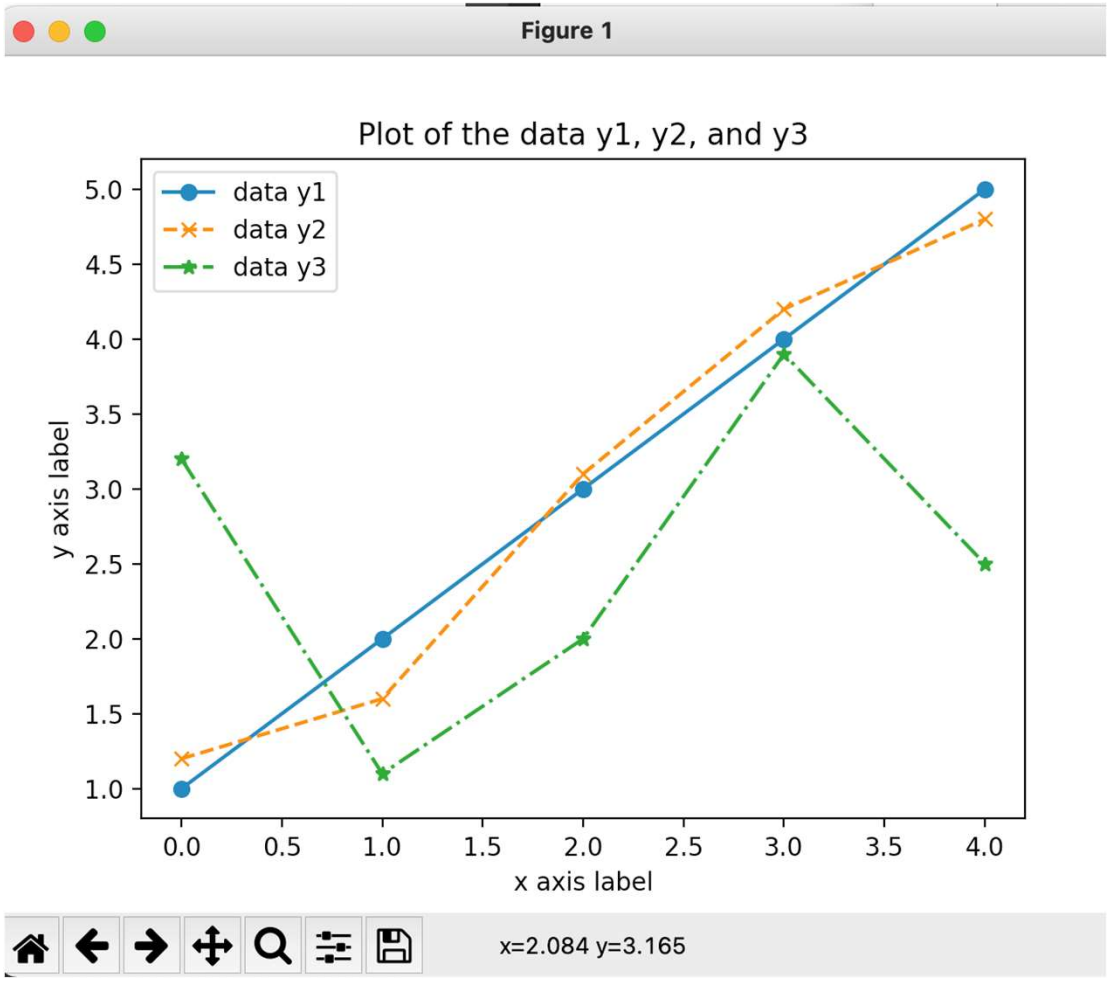
```
import numpy as np
import matplotlib.pyplot as plt

y1 = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y2 = np.array([1.2, 1.6, 3.1, 4.2, 4.8])
y3 = np.array([3.2, 1.1, 2.0, 3.9, 2.5])

fig, ax = plt.subplots()
lines = ax.plot(y1, 'o', y2, 'x', y3, '*')
ax.set_title("Plot of the data y1, y2, and y3")
ax.set_xlabel("x axis label")
ax.set_ylabel("y axis label")
ax.legend(("data y1", "data y2", "data y3"))

plt.show()
```



Plotting lines

+ We can also customize the lines of the plotting by changing to the following:

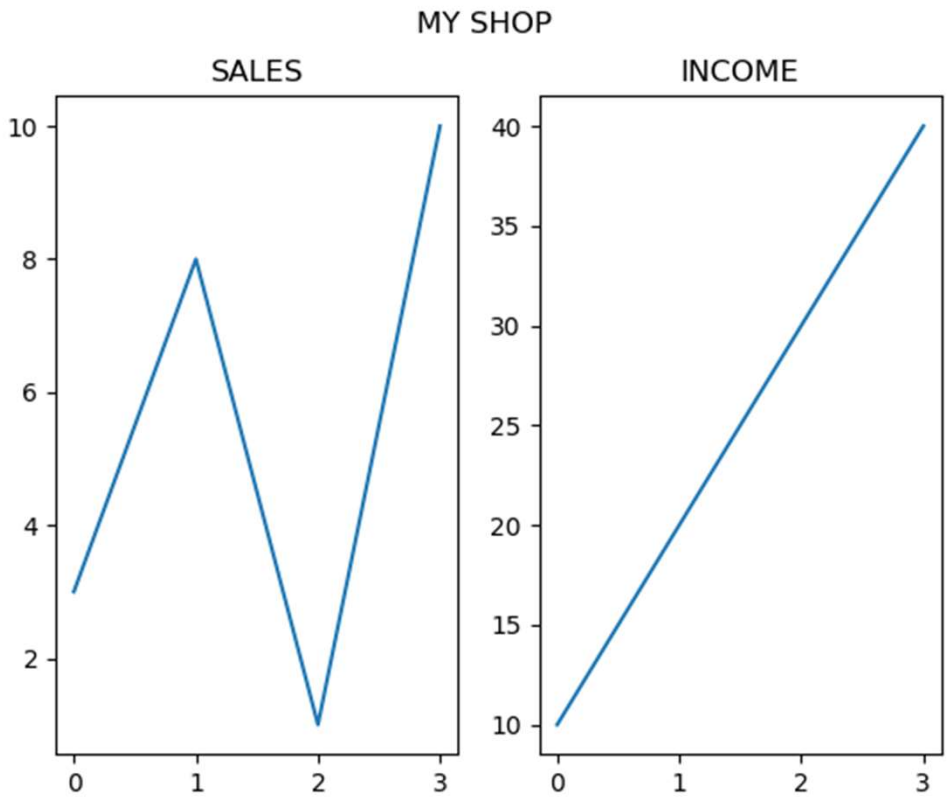
lines = ax.plot(y1, 'o-', y2, 'x--', y3, '*-.')



Subplots

- It is also possible to draw multiple graphs in the same figure.
- plt.subplots(int, int, *index*), default: (1, 1, 1)
- Three integers (*nrows*, *ncols*, *index*).
- The subplot will take the *index* position on a grid with *nrows* rows and *ncols* columns.
- *index* starts at 1 in the upper left corner and increases to the right.
- *index* can also be a two-tuple specifying the (*first*, *last*) indices (1-based, and including *last*) of the subplot,
- e.g., fig.add_subplot(3, 1, (1, 2)) makes a subplot that spans the upper 2/3 of the figure.

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

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.suptitle("MY SHOP")
plt.show()
```



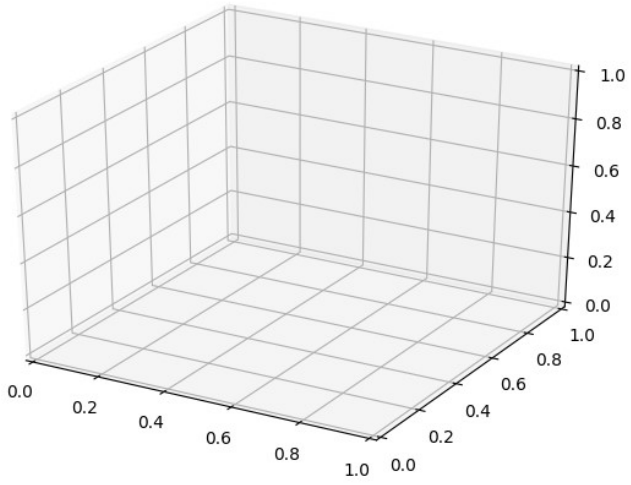
Saving Matplotlib figures

- The figures can be saved in several formats such as:
  1. PNG
  2. SVG
  3. PDF
  4. PS

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

y1 = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y2 = np.array([1.2, 1.6, 3.1, 4.2, 4.8])
y3 = np.array([3.2, 1.1, 2.0, 3.9, 2.5])

fig, ax = plt.subplots()
lines = ax.plot(y1, 'o', y2, 'x', y3, '*')
ax.set_title("Plot of the data y1, y2, and y3")
ax.set_xlabel("x axis label")
ax.set_ylabel("y axis label")
ax.legend(("data y1", "data y2", "data y3"))

plt.savefig('figure1.png')
```

3D plotting with Matplotlib

- 3D plotting in Matplotlib starts by enabling the utility toolkit.
- Importing the mplot3d library



projection="3d"

- Once this sub-module is imported, 3D plots can be created by passing the keyword projection="3d" to any of the regular axes creation functions in Matplotlib.

Try this!!!

```
from mpl_toolkits import mplot3d

import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax = plt.axes(projection="3d")

plt.show()
```

Empty

- We just created our axes.
- Now, we can start plotting in 3D.
- The 3D plotting functions are quite intuitive: instead of just scatter we call scatter3D , and instead of passing only *x* and *y* data, we pass over *x*, *y*, and *z*.
- All of the other function settings such as colour and line type remain the same as with the 2D plotting functions.

Draw something in 3D

```
fig = plt.figure()
ax = plt.axes(projection="3d")

z_line = np.linspace(0, 15, 1000)
x_line = np.cos(z_line)
y_line = np.sin(z_line)
ax.plot3D(x_line, y_line, z_line, 'gray')

z_points = 15 * np.random.random(100)
x_points = np.cos(z_points) + 0.1 * np.random.randn(100)
y_points = np.sin(z_points) + 0.1 * np.random.randn(100)
ax.scatter3D(x_points, y_points, z_points, c=z_points, cmap='hsv');

plt.show()
```

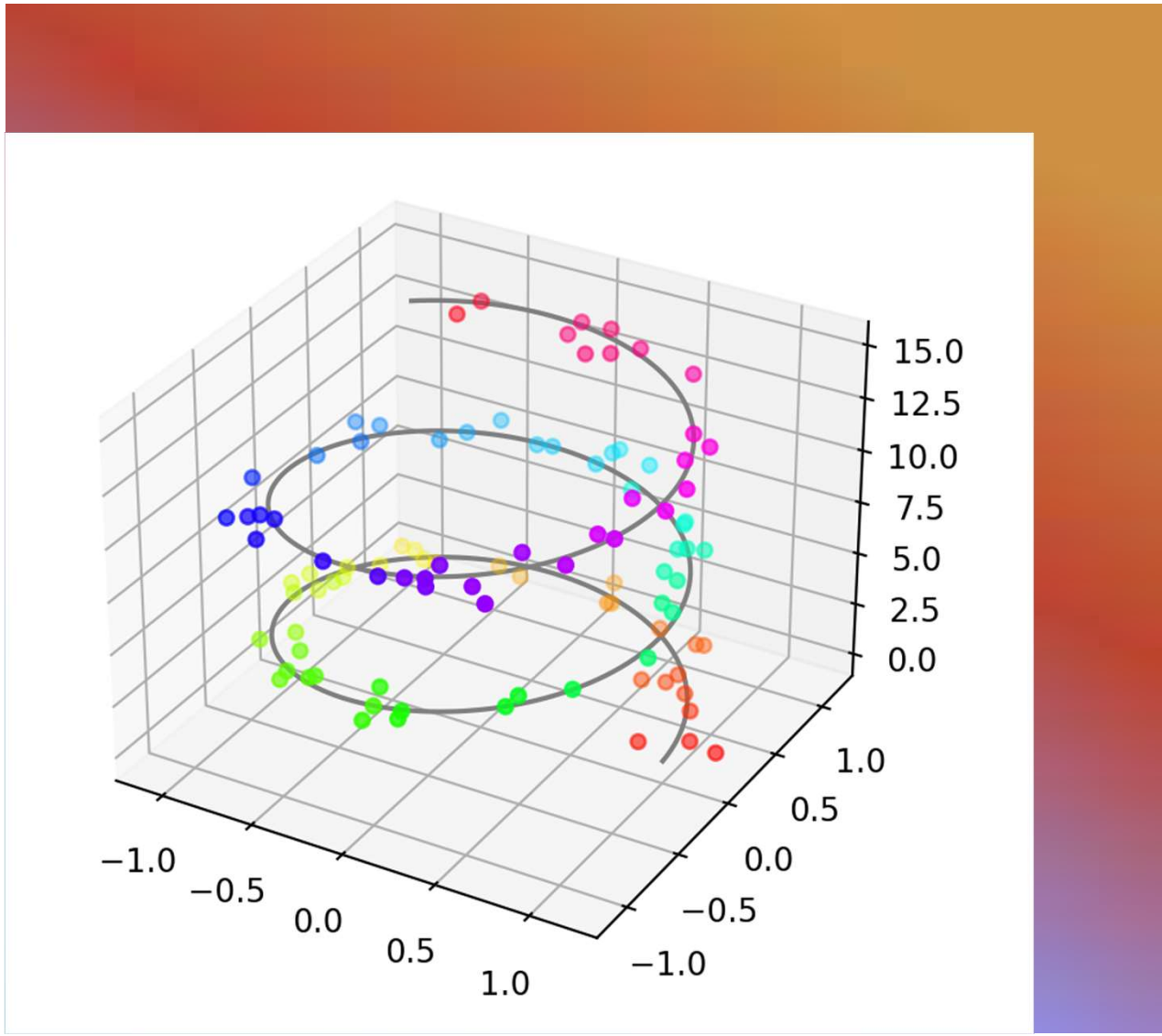
Did you find an issue?

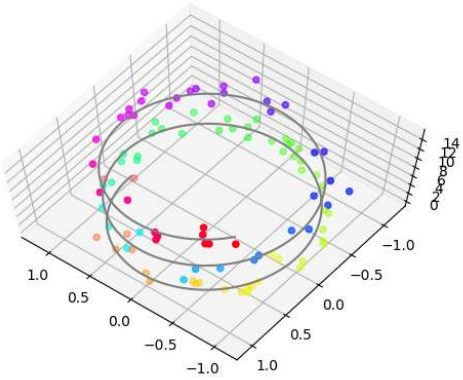
+ You need to import the necessary libs

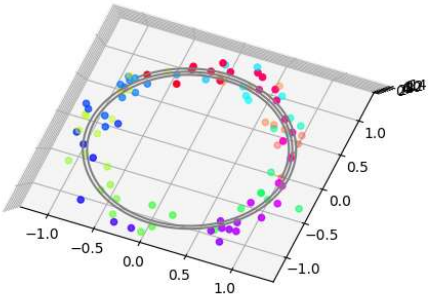
from mpl_toolkits *import* mplot3d
import numpy *as* np
import matplotlib.pyplot *as* plt

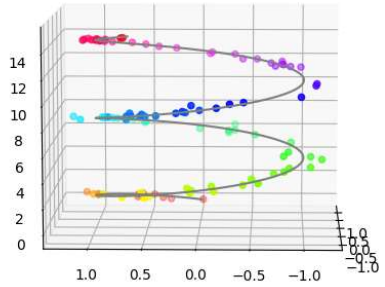
Result

+ Did you get this output?









Interactivity

+ The interactivity of plots becomes extremely useful for exploring your visualised data once you've plotted in 3D.

Surface Plots

- Surface plots are **diagrams of three-dimensional data**. Rather than showing the individual data points, surface plots show a functional relationship between a designated dependent variable (Y), and two independent variables (X and Z).
- The plot is a companion plot to the contour plot.

Surface Plots in Python with matplotlib

+ Constructing a surface plot in Matplotlib is a 3-step process.

(1) First, we need to generate the actual points that will make up the surface plot.

(2) The second step is to plot a wire-frame — this is our estimate of the surface.

(3) Finally, we'll project our surface onto our wire-frame estimate and extrapolate all the points.

First

```
fig = plt.figure()
ax = plt.axes(projection="3d")
def z_function(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))

x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)

X, Y = np.meshgrid(x, y)
Z = z_function(X, Y)
```

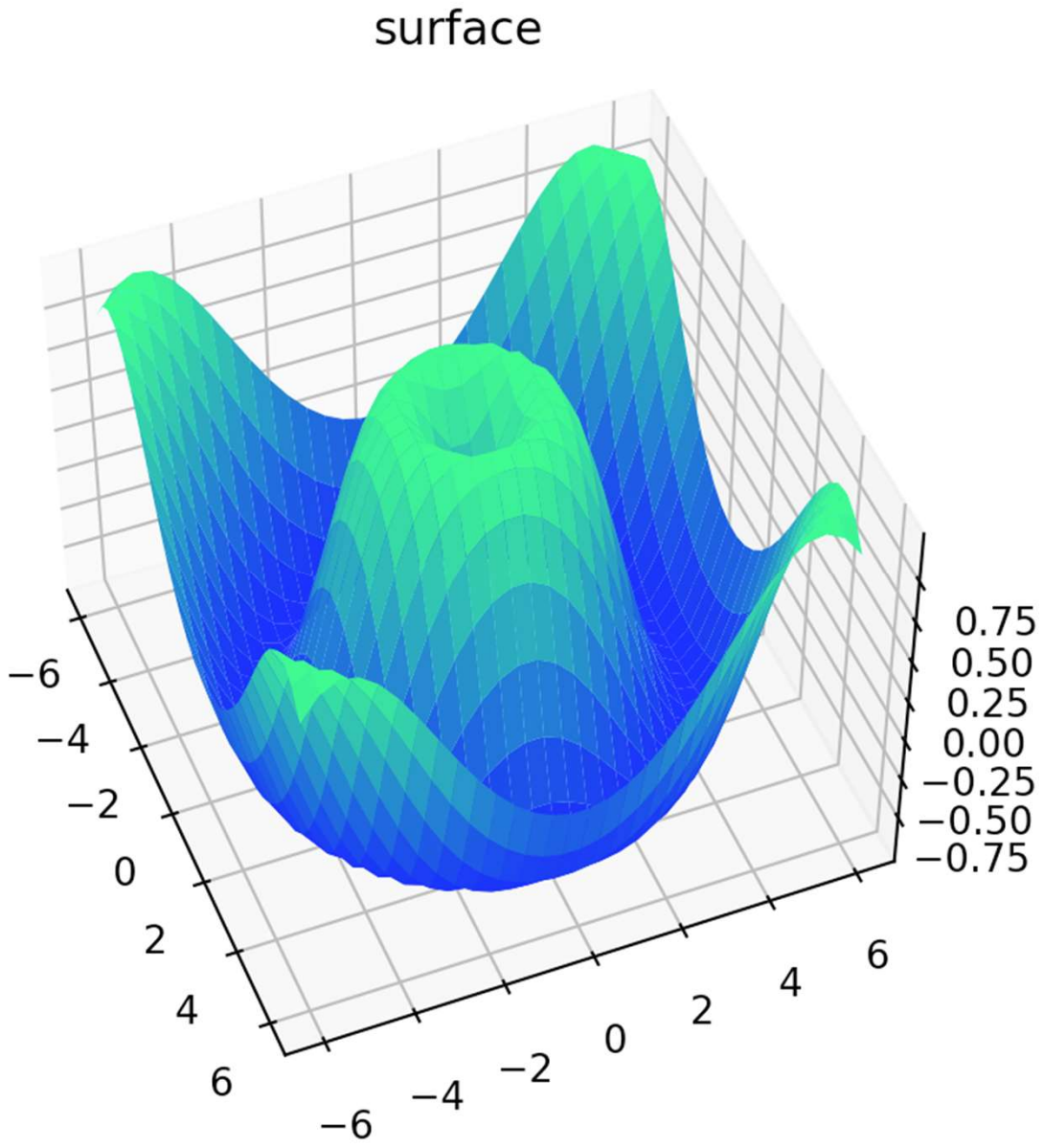
Second

+ ax.plot_wireframe(X, Y, Z, color='green')
  ax.set_xlabel('x')
  ax.set_ylabel('y')
  ax.set_zlabel('z')

Third

```
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
cmap='winter', edgecolor='none')
ax.set_title('surface’);

plt.show()
```

Result



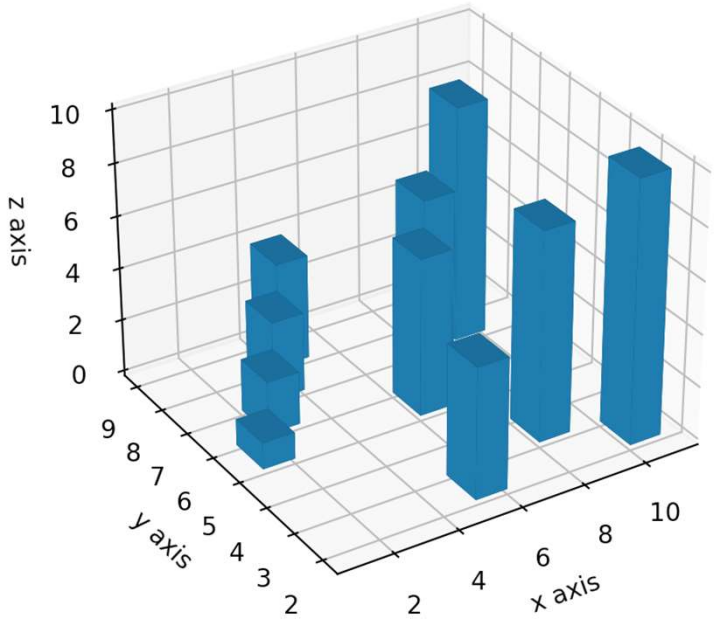
3D Bar Plots

- Bar plots are used quite frequently in data visualisation projects since they're able to convey information, usually some type of comparison, in a simple and intuitive way.
- The beauty of 3D bar plots is that they maintain the simplicity of 2D bar plots while extending their capacity to represent comparative information.

Bar plot requirements

- Each bar in a bar plot always needs 2 things: a position and a size.
- With 3D bar plots three variables such as *x, y, z* are needed.
- We'll select the *z* axis to encode the height of each bar; therefore, each bar will start at $z = 0$ and have a size that is proportional to the value we are trying to visualise. The *x* and *y* positions will represent the coordinates of the bar across the 2D plane of $z = 0$.
- We'll set the *x* and *y* size of each bar to a value of 1 so that all the bars have the same shape.

Bar plot




```
fig = plt.figure()
ax1 = fig.add_subplot(111, projection='3d')
x3 = [1,2,3,4,5,6,7,8,9,10]
y3 = [5,6,7,8,2,5,6,3,7,2]
z3 = np.zeros(10)
dx = np.ones(10)
dy = np.ones(10)
dz = [1,2,3,4,5,6,7,8,9,10]
ax1.bar3d(x3, y3, z3, dx, dy, dz)
ax1.set_xlabel('x axis')
ax1.set_ylabel('y axis')
ax1.set_zlabel('z axis')
plt.show()
```