Article Image

Plotting with matplotlib

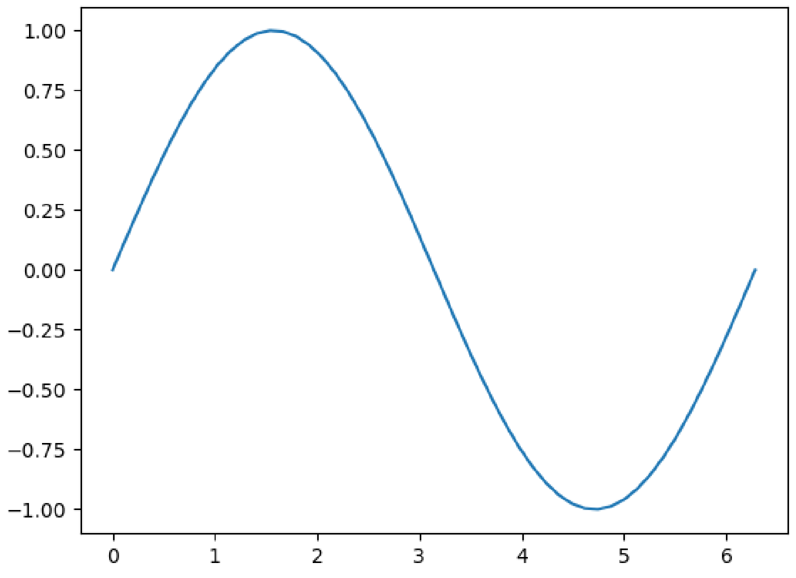
Table of contents

Plotting

Visualization in Python can be performed by a famous library named `matplotlib`, in particular its sub-package `matplotlib.pyplot`. Documentation can be found at matplotlib.org.

Plotting in matplotlib is very easy. Given two N-dimensional vectors $x = (x_1, \dots, x_N)$ and $y = (y_1, \dots, y_N)$, containing the N datapoints we want to represent, the function `plot(x, y)` will plot on the plate each couple (x_i, y_i) for $i = 1, \dots, N$, and will connect (by default) them with a line. Such a plot can be visualized by calling the function `show()`.

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

# Creating two vectors
a = 0
b = 2*np.pi
N = 50

x = np.linspace(a, b, N)
y = np.sin(x)

# Visualize
plt.plot(x, y)
plt.show()
```



As you can see, the code above will plot the sine function. We now want to see how we can change the aesthetic of this plot, by adding title, axis grid, axis label, …

Customize the plot

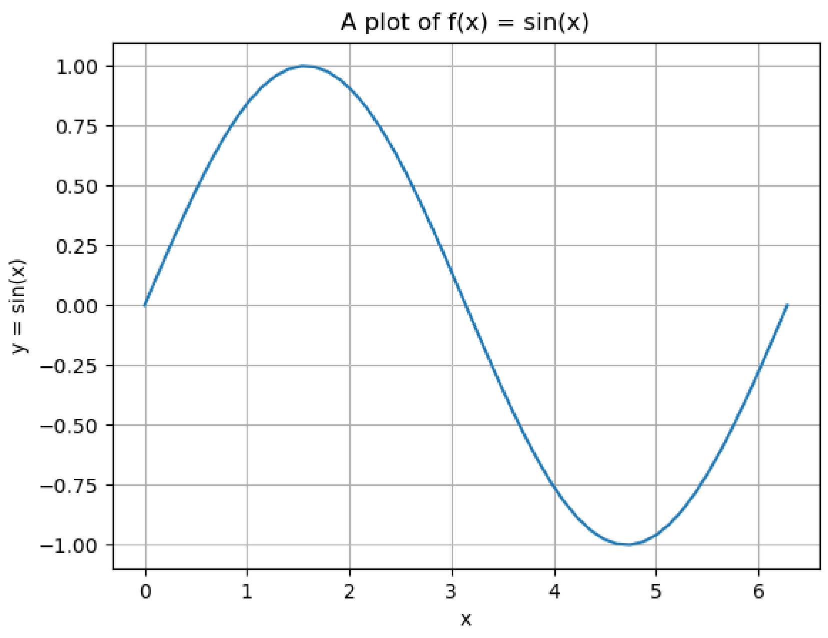
In matplotlib, most of the customization we want to add to the plot must be inserted in between the line `plt.plot(x, y)` and the line `plt.show()`. The most common customization functions are:

- `plt.title(str)`: Add a title to the plot;
- `plt.xlabel(str)`: Add a label to the x-axis;
- `plt.ylabel(str)`: Add a label to the y-axis;

- `plt.grid()`: Add an axis grid on the background of the plot;
- `plt.xlim([a, b])`: Force the horizontal limit of the axis to be `a` and `b`;
- `plt.ylim([a, b])`: Force the vertical limit of the axis to be `a` and `b`;

For example, we can customize the plot above to obtain something like that

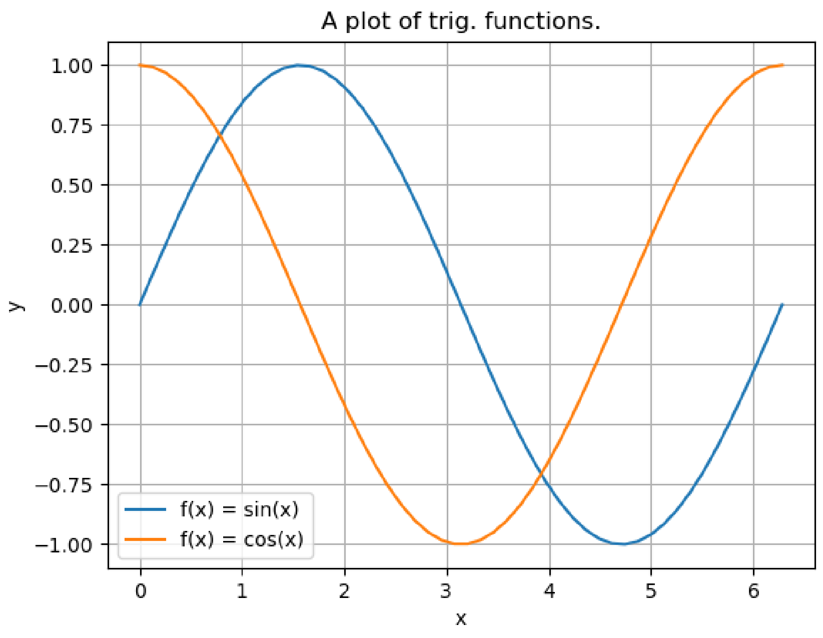
```
import numpy as np
import matplotlib.pyplot as plt

# Creating two vectors
a = 0
b = 2*np.pi
N = 50

x = np.linspace(a, b, N)
y = np.sin(x)

# Visualize
plt.plot(x, y)
plt.title('A plot of f(x) = sin(x)')
plt.xlabel('x')
plt.ylabel('y = sin(x)')
plt.grid()
plt.show()
```



Multiplot and Line customization

Clearly, it is also possible to plot more than one line at the same time. Simply define others $x', y' \in \mathbb{R}^N$ containg the new data we want to plot and add another `plt.plot(x', y')` in between `plt.plot(x, y)` and `plt.show()`.

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

# Creating two vectors
a = 0
b = 2*np.pi
N = 50

x = np.linspace(a, b, N)
y1 = np.sin(x)
y2 = np.cos(x)


# Visualize
plt.plot(x, y1)
plt.plot(x, y2)
plt.title('A plot of trig. functions.')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(['f(x) = sin(x)', 'f(x) = cos(x)'])
plt.grid()
plt.show()
```



As you can see, in the bottom-left of the plot, we also printed out a `legend`. Following the code above, it is easy to understand that a legend can be simply introduced by listing the name of the lines, ordered with respect to the ordering of the `plt.plot()` functions. Matplotlib will visualize the correct color of the line accordingly.

Clearly, we can also modify the line specifications such as the color, the thickness and the style. To to that, we have to insert the following specifications inside of the corresponding `plt.plot()` line.

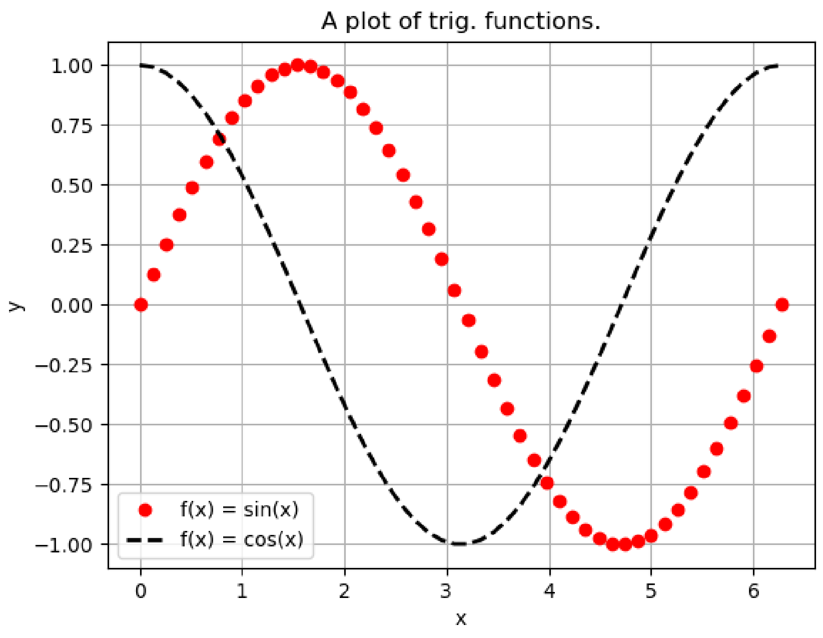
- `color='str'`: Change the color of the line. A list of all the available colors can be found here;
- `linewidth=int`: Change the thickness of the line.

Moreover, the style of the line can be modified by adding some specifications just after the `y` input. For example,

- `"o"`: Changes the linestyle to rounded markers;
- `"--"`: Changes the linestyle to be dotted lines;
- `"o-"`: Changes the linestyle to be a continuous line with markers on the points defined by `(x, y)`.

A complete list of all the possible linestyles can be found here.

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

# Creating two vectors
a = 0
b = 2*np.pi
N = 50

x = np.linspace(a, b, N)
y1 = np.sin(x)
y2 = np.cos(x)


# Visualize
plt.plot(x, y1, 'o', color='red')
plt.plot(x, y2, '--', color='k', linewidth=2)
plt.title('A plot of trig. functions.')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(['f(x) = sin(x)', 'f(x) = cos(x)'])
plt.grid()
plt.show()
```



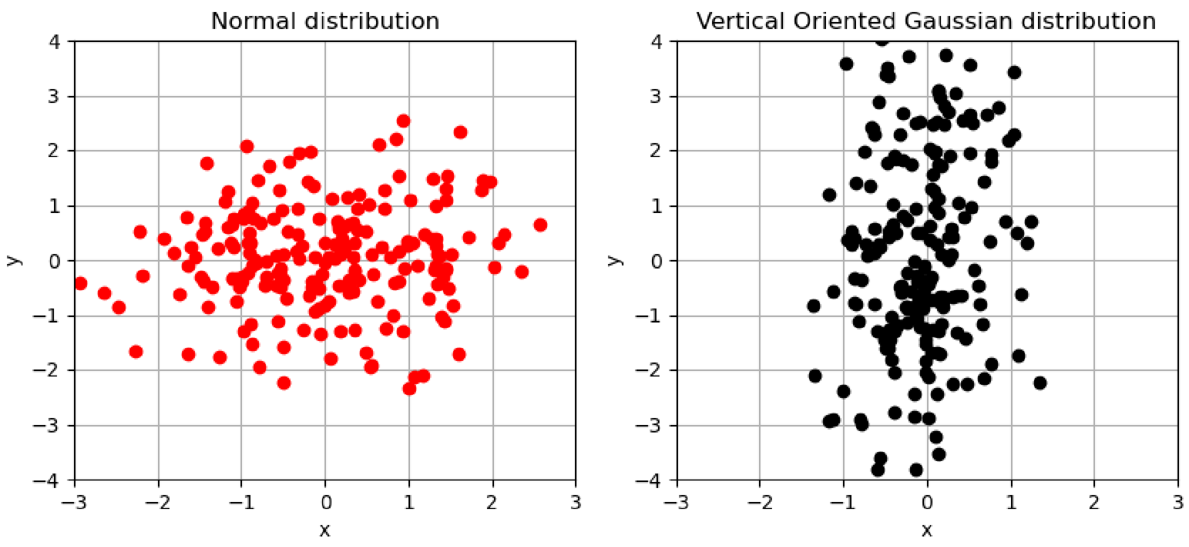
Subplots

Subplots are required to create a matrix of plots inside of the same figure, which can be very useful for various visualizations.

A subplot is created by first defining a `figure`. This can be done by the line `plt.figure(figsize=(w, h))` where the `figsize` argument is required to change the proportion of the resulting plot. After that, it is possible to open a subplot with the command `plt.subplot(nrow, ncol, idx)`, where `nrow` and `ncol` represents the number of images per rows and the number of images per columns in our matrix of plots, while `idx` is an incremental value, starting from 1, that indicate where the plot we are going to do should be placed inside of the matrix. `idx=1` represents the upper-left corner and, while increasing, it moves the image from left to right and from up to down into the matrix.

Each time we want to open a different plot in our subplot, we have to specify the command `plt.subplot(nrow, ncol, idx)` again, with the same `nrow` and `ncol` argument, but different `idx`.

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

# Creating data
N = 200

x1 = np.random.normal(0, 1, (N, ))
y1 = np.random.normal(0, 1, (N, ))

x2 = np.random.normal(0, 0.5, (N, ))
y2 = np.random.normal(0, 2, (N, ))


# Visualize
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(x1, y1, 'o', color='red')
plt.title('Normal distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim([-3, 3])
plt.ylim([-4, 4])
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(x2, y2, 'o', color='k')
plt.title('Vertical Oriented Gaussian distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim([-3, 3])
plt.ylim([-4, 4])
plt.grid()

plt.show()
```



Exercise: Plotting data

Going back to the example in the introductory post on Numpy where we introduced the library pandas, useful to read data into Python, we can now use matplotlib to visualize it.

First of all, download (if required) the data from Virtuale or equivalently by Kaggle at the following link: www.kaggle.com/mysarahmadbhat/us-births-2000-to-2014, and place the *.csv* file into the same folder of your `.py` file. Then, with the help of what you studied in the introductory post,

- Import the data into Python;
- Explore the data by visualizing the first rows and the columns of it (the function `data.head()` from pandas can be useful), or alternatively, use the data documentation on www.kaggle.com/mysarahmadbhat/us-births-2000-to-2014;
- Create a new column, `total_date`, representing each date into an increasing number, the days since the beginning of the data collection;
- Plot the number of birth with respect to `total_date` to visualize the incremental number of birth during the years;
- *Optional:* Plot an barplot (matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html) of the number of birth with respect to the day of the week and investigate if there are asymetries in the birth number in some days of the week.

Going on

The next topic will be linear systems.

Written by

Davide Evangelista

Published 16 Sep 2022