

# visualization\_with\_Matplotlib\_L1

March 17, 2022

## 1 Visualization with matplotlib I



### 1.1 Content:

- Section 1.2
- Section 1.3
- Section ??
- Section ??
- Section ??
- Section 1.7
- Section ??
- Section 1.10
- Section 1.11
- Section 1.12
- Section 2
- Section ??

### 1.2 Introduction

One of the most important modules in Python for 2-dimensional visualization is Matplotlib. It is a very comprehensive package with many display options for 2-dimensional data, whether xy plots, bar charts, pie charts, box plots, contours, vectors, scatters, and so on.

Matplotlib home page <https://matplotlib.org/>

Matplotlib provides many example scripts with a short description, see <https://matplotlib.org/3.3.3/gallery/index.html> For the beginner it is sometimes difficult to get started, but through the gallery with the many examples you can quickly find something that suits your needs.

Matplotlib's collection of style functions **pyplot** work very similar to MATLAB. For a detailed description see [https://matplotlib.org/users/pyplot\\_tutorial.html](https://matplotlib.org/users/pyplot_tutorial.html).

### 1.3 Preliminary work

Jupyter notebook offers the possibility to display the plots within the notebook. Therefore we have to turn on the matplotlib mode.

```
%matplotlib inline
```

```
[ ]: %matplotlib inline
```

### 1.4 Import plot library

Import the function collection pyplot from matplotlib and use the shortcut plt for it.

```
import matplotlib.pyplot as plt
```

```
[ ]: import matplotlib.pyplot as plt
```

### 1.5 A simple plot

With the plot function we can create an empty plot.

```
plt.plot()
```

```
[ ]: plt.plot()
```

### 1.6 Add grid lines

If you want to add some grid lines to the plot use the grid function.

```
plt.grid()
```

```
[ ]: plt.grid()
```

### 1.7 Basic concepts

Matplotlib allows us to create a basic plot in an comfortable way. Only two lists or arrays are needed.

We define two short lists containing some integer values for x and y with the same size.

```
x = [1,3,6,7,9]
```

```
y = [1,3,2,1,3]
```

```
[ ]: x = [1,3,6,7,9]
      y = [1,3,2,1,3]
```

Now we use the two lists to generate a line plot with Matplotlib's plot function. Running the following command line will display the line plot inside this notebook (if you have set %matplotlib inline).

```
plt.plot(x, y)
```

```
[ ]: plt.plot(x, y)
```

This plot is more than minimalistic and we will show you how to change the line style now.

You can try some of the following line styles. The code string as well as the short code string can be used.

Line style	code	short code
-----		
Solid line	'solid'	'-'
Dashed line	'dashed'	'--'
Dotted line	'dotted'	':'
Dash dot line	'dashdot'	'-.'

Note: There are many more line styles available. See:  
[https://matplotlib.org/3.1.0/gallery/lines\\_bars\\_and\\_markers/linestyles.html](https://matplotlib.org/3.1.0/gallery/lines_bars_and_markers/linestyles.html)

```
plt.plot(x, y, linestyle=':')
```

```
[ ]: plt.plot(x, y, linestyle=':')
```

Let's switch from line to marker style.

Circle	'o'
Triangle up	'^'
Triangle down	'v'
Square	's'
Star	'*'
Plus	'+'
X	'x'
Diamond	'D'
Diamond thin	'd'

Note: There are many more markers available. See:  
[https://matplotlib.org/3.3.3/api/markers\\_api.html](https://matplotlib.org/3.3.3/api/markers_api.html)

```
plt.plot(x, y, 'D')
```

```
[ ]: plt.grid()

plt.plot(x, y, 'D')
```

That looks quite nice but we want to use another color, maybe red.

The following color abbreviations are available:

Color	code
-----	
black	'k'
white	'w'
red	'r'
green	'g'

```
blue      'b'
cyan      'c'
magenta   'm'
yellow    'y'
```

Note: There are many more named colors available. See:  
[https://matplotlib.org/3.1.0/gallery/color/named\\_colors.html](https://matplotlib.org/3.1.0/gallery/color/named_colors.html)

Matplotlib accepts combinations of multiple parameter abbreviations, e.g.

'Dr' is the same as `linestyle='none', marker='D', color='r'`

==>

```
plt.plot(x, y, 'Dr')
```

is the same as

```
plt.plot(x, y, linestyle='none', marker='D', color='r')
```

```
[ ]: plt.plot(x, y, 'Dr')
      #plt.plot(x, y, linestyle='none', marker='D', color='r')
```

Let's create a dashed line plot with circle markers in magenta.

```
plt.plot(x, y, '--om')
```

```
#plt.plot(x, y, linestyle='--', marker='o', color='m')
```

```
[ ]: plt.plot(x, y, '--om')
      #plt.plot(x, y, linestyle='--', marker='o', color='m')
```

## 1.8 Customize markers and lines

You can customize the plot further by changing \* marker size \* marker facecolor \* marker edgecolor \* linewidth

In the following plot, we generate larger red markers with black edges. For the line, we use the default linestyle and linecolor, but we increase the linewidth to 2.

```
[ ]: plt.plot(x, y, linewidth=2, marker="v", markersize=14, markerfacecolor="red",
             →markeredgecolor="black")
```

Sometimes you want to overlay different components of your plots using transparency. In matplotlib this is achieved by setting the transparency parameter alpha. Here, alpha=1 corresponds to full opacity, and alpha=0 to full transparency. In the following plot, we draw a transparent line on top of a solid line:

```
[ ]: plt.grid()
      plt.plot(x, y, 'o-', linewidth=2, alpha=1.0)
      plt.plot(x, y, 's:', linewidth=4, alpha=0.5)
```

Exercise

1. Import numpy `import numpy as np`

2. Create data

```
x = np.arange(1., 100., 4)
y = np.sin(3.14159*(x+.5)/50.)
```

3. Create a line plot with the following attributes

- set line style to dotted
- set line color to red
- set line width to 1
- add plus markers in black with marker size 8 to the plot

4. Create another line plot with the following attributes:

- set line style to solid
- set line color to red
- set line width to 2
- add triangle markers in green with marker size 12 and marker edgecolor black
- make the line plot transparent with 75% transparency

Hint: [https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.pyplot.plot.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html)

```
[ ]: # 1.
```

```
[ ]: # 2.
```

```
[ ]: # 3.
```

```
[ ]: # 4.
```

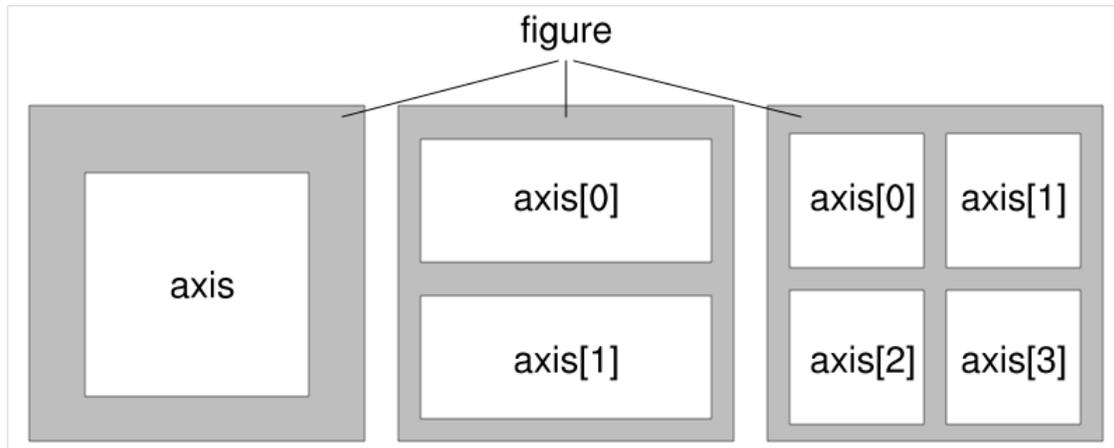
But of course you can still adjust the various settings manually, such as size, color, output resolution, line width, text settings, etc..

## 1.9 Figures and axes

To make the best use of Matplotlib, you should understand the base plot concept. Matplotlib provides two objects to control the plots

- **figure** object
- **axis** object

Figure can be considered as the workspace and axis as the actual plot. You can create a figure that contains one or more plots.



The left part of the image shows in gray a figure containing one plot (axis). In the middle we see in gray one figure containing two plots (axis with 2 elements) and on the right side we see in gray a figure containing four plots (axis with 4 elements). In the next section, the use of figure and axis will be shown in more detail.

It is very convenient to create figure and axis objects at once using the Matplotlib `pyplot.subplots` function. Furthermore, further setting options for the axis are introduced, which are mostly self-explanatory.

The next example demonstrates the use of `fig` and `ax` for a plot where we also add a title and the x- and y-axis titles.

```
fig, ax = plt.subplots(figsize=(6,4))

ax.set_title('Example A', fontsize=24)
ax.set_ylabel('y axis label', fontsize=16)
ax.set_xlabel('x axis label', fontsize=16)
ax.tick_params(labelsize=16)

ax.plot(x, y, linestyle="solid", linewidth=2, color="blue",
        marker="o", markersize="10", markerfacecolor="yellow")
```

```
[ ]: fig, ax = plt.subplots(figsize=(6,4))

ax.set_title('Example A', fontsize=24)
ax.set_ylabel('y axis label', fontsize=16)
ax.set_xlabel('x axis label', fontsize=16)
ax.tick_params(labelsize=16)

ax.plot(x, y, linestyle="solid", linewidth=2, color="blue",
        marker="o", markersize="10", markerfacecolor="yellow")
```

## 1.10 Multiple plots

In the next section we explain how to use `pyplot.subplots` functionality to create two plots on one workspace.

1. Create two plots in one row.
2. Use same x- and y-axis ranges for both plots.
3. Draw a common title.

```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(8,4), sharex=True, sharey=True)
fig.suptitle('Panel plot', fontsize=24)
```

```
x2 = [0, 20, 30, 60, 90]
y2 = [ 2, 2, 2.5, -1.5, -2]
```

```
axs[0].plot(x, y)
axs[1].plot(x2, y2, "r")
```

Exercise

- define a frame that contains two plots, one plot per row
- write a title on top of the frame
- create left plot with magenta line color
- create right plot with green line color

```
[ ]: import numpy as np

x = np.arange(1.,100.,4)
y = np.sin(3.14159*(x+.5)/50.)

fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(8,8), sharex=True,
→sharey=True)
fig.suptitle('Panel plot', fontsize=24)

x2 = [0, 20, 30, 60, 90]
y2 = [ 2, 2, 2.5, -1.5, -2]

axs[0].plot(x, y, "m")
axs[1].plot(x2, y2, "g")
```

## 1.11 Plot types

Line plots are a little bit boring that's why we change the plot type and see what we get.

You can find the list of different pyplot functions at [https://matplotlib.org/api/pyplot\\_summary.html](https://matplotlib.org/api/pyplot_summary.html)

Next, we use arrays as input, which we create using the Numpy module (plots just looks nice ;)). Here are just a few examples of the different plot functions.

```
import numpy as np
```

```

x = np.linspace(0.1, 2 * np.pi, 41)
y = np.exp(np.sin(x))

fig, axs = plt.subplots(nrows=1, ncols=4, figsize=(16,4))

axs[0].plot(x, y)
axs[1].scatter(x, y)
axs[2].bar(x, y)
axs[3].stem(x, y)

```

```

[ ]: import numpy as np

x = np.linspace(0.1, 2 * np.pi, 41)
y = np.exp(np.sin(x))

fig, axs = plt.subplots(nrows=1, ncols=4, figsize=(16,4))

axs[0].plot(x, y)
axs[1].scatter(x, y)
axs[2].bar(x, y)
axs[3].stem(x, y)

```

## 1.12 Text annotations

You can customize the titles and axis labels with the `ax.set()` method in one line.

The next example adds

1. a multiline title above the plot
2. multiline labels at x- and y-axis

```

fig, ax = plt.subplots(figsize=(6,4))

ax.plot(x, y)

ax.set(title='This is a title string\nyou can add more information',
       xlabel='x-axis\n[units]',
       ylabel='y-axis\n[units]')

```

```

[ ]: fig, ax = plt.subplots(figsize=(6,4))

ax.plot(x, y)

ax.set(title='This is a title string\nyou can add more information',
       xlabel='x-axis\n[units]',
       ylabel='y-axis\n[units]')

```

You can also write additional text in a plot and of course change the font, e.g. Greek, and size.

```

fig, ax = plt.subplots(figsize=(6,4))

```

```
ax.plot(x, y)
```

```
ax.text(3.5, 2.0, 'Some text',fontsize=24)
```

```
ax.text(0.2, 0.8, r'$\omega\ \mu\ \sigma\ \pi\ \nu\ \phi$', fontsize=24)
```

```
[ ]: fig, ax = plt.subplots(figsize=(6,4))

ax.plot(x, y)

ax.text(3.5, 2.0, 'Some text',fontsize=24)
ax.text(0.2, 0.8, r'$\omega\ \mu\ \sigma\ \pi\ \nu\ \phi$', fontsize=24)
```

We can also add text to the workspace outside the plot, e.g. a copyright text. The figure size will be extended when the text string won't fit and the plot size will be unchanged.

```
fig, ax = plt.subplots(figsize=(6,4))
```

```
ax.plot(x, y)
```

```
plt.text(0.7, 0.01, '@ your copyright text', fontsize=8, transform=plt.gcf().transFigure)
```

```
[ ]: fig, ax = plt.subplots(figsize=(6,4))

ax.plot(x, y)

plt.text(0.7, 0.01, '@ your copyright text', fontsize=8, transform=plt.gcf().
→transFigure)
```

Exercise

Move the copyright text to the right side in vertical mode.

Hint: [https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.pyplot.text.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.text.html)

```
[ ]: # Solution
```

## 2 Plot settings

Next, as an example, we want to plot annual average temperature data from Germany (from DWD), coloring the data points according to their values, add a title string, and set the axis titles as well. To display the data as individual points we use the scatter function as shown above.

Used parameters of the scatter plot command:

```
c=temp          use variable temp for coloring the marker
```

```
cmap='afmhot_r' use colormap afmhot but reversed
```

```
s=80           increase the size of the markers
```

Example:

```
years = np.arange(1951,2021)
```

```
temp = np.array([3.02, 7.91, 5.08, 2.53, 0.93, 0.58, 6.85, 1.58, 5.63, 1.30, 3.97, 2.34, 4.50, 9.97, 2.64, 3.95, 2.38, 6.63, 2.01, 7.08, 4.70, 5.78, 2.75, 6.63, 10.20, 1.25, 1.89, 2.29, 1.45, 2.19, 6.54, 9.93, 3.30, 2.63, 4.31, 1.64, 2.10, 4.67, 5.91, 5.27, 9.64, 1.89, 16.27, 10.54, 3.00, 5.16, 7.08, 5.16, 5.67, 7.36, 6.23, 19.01, 4.70, 7.57, 14.34, 4.82, 7.24, 4.58, 10.63, 4.27, 7.62, 10.47, 6.14, 17.60, 9.21, 6.80, 20.37, 16.97, 11.39])
```

```
fig, ax = plt.subplots(figsize=(6,4))
```

```
ax.set_title('Annual average temperature', fontsize=24)
```

```
ax.set_ylabel('Temperature\n[deg C]', fontsize=16)
```

```
ax.set_xlabel('year', fontsize=16)
```

```
ax.tick_params(labelsize=16)
```

```
ax.plot(years, temp, color='lightgray')
```

```
ax.scatter(years, temp, c=temp, cmap='afmhot_r', s=80)
```

```
[ ]: years = np.arange(1951,2021)

temp = np.array([3.02, 7.91, 5.08, 2.53, 0.93, 0.58, 6.85, 1.58, 5.63, 1.30, 3.
→97, 2.34, 4.50, 9.93, 1.18, \
                2.64, 3.95, 2.38, 6.63, 2.01, 7.08, 4.70, 5.78, 2.75, 6.63, 10.
→20, 1.25, 1.89, 2.29, 1.45, \
                2.19, 6.54, 9.93, 3.30, 2.63, 4.31, 1.64, 2.10, 4.67, 5.91, 5.
→27, 9.64, 1.89, 16.27, 10.54, \
                3.00, 5.16, 7.08, 5.16, 5.67, 7.36, 6.23, 19.01, 4.70, 7.57, 14.
→34, 4.82, 7.24, 4.58, 10.63, \
                4.27, 7.62, 10.47, 6.14, 17.60, 9.21, 6.80, 20.37, 16.97, 11.39])

fig, ax = plt.subplots(figsize=(6,4))

ax.set_title('Annual average temperature', fontsize=24)
ax.set_ylabel('Temperature\n[deg C]', fontsize=16)
ax.set_xlabel('year', fontsize=16)
ax.tick_params(labelsize=16)

ax.plot(years, temp, color='lightgray')
ax.scatter(years, temp, c=temp, cmap='afmhot_r', s=80)
```

**Note:** The draw order is first marker and then line, which doesn't look quite nice. To change the draw order we can use the **zorder** parameter.

Example:

```
fig, ax = plt.subplots(figsize=(6,4))
```

```

ax.set_title('Annual average temperature', fontsize=24)
ax.set_ylabel('Temperature\n[deg C]', fontsize=16)
ax.set_xlabel('year', fontsize=16)
ax.tick_params(labelsize=16)

ax.plot(years, temp, color='lightgray', zorder=1)
ax.scatter(years, temp, c=temp, cmap='afmhot_r', s=80, zorder=2)

```

```

[ ]: fig, ax = plt.subplots(figsize=(6,4))

ax.set_title('Annual average temperature', fontsize=24)
ax.set_ylabel('Temperature\n[deg C]', fontsize=16)
ax.set_xlabel('year', fontsize=16)
ax.tick_params(labelsize=16)

ax.plot(years, temp, color='lightgray', zorder=1)
ax.scatter(years, temp, c=temp, cmap='afmhot_r', s=80, zorder=2)

```

Matplotlib offers a wide selection of colormaps. You can select colormaps for sequential and for categorical data. For hints which colormap to choose, see <https://matplotlib.org/stable/tutorials/colors/colormaps.html>.

## 2.1 Save the plot to PNG

Since we do not only want to have the plots as an inline image, but also save them as a PNG file, we show here how to do that. Therefore, we add a line with `savefig` function at the end of the script snippet.

```

[ ]: plt.savefig("Germany_average_annual_temperature_1951-2020.png")

```

## 2.2 Beautiful plots with Seaborn

Seaborn is a high-level interface to matplotlib, specifically aimed at statistical plots. It is very useful for generating beautiful plots that are easy to understand. Documentation: <https://seaborn.pydata.org/>

With the `set_style` function we can select one of five predefined styles (`whitegrid`, `white`, `darkgrid`, `dark`, `ticks`). Seaborn automatically adjusts the grid and figure background to the preferences and chooses reasonable values for tick and label sizes. We now make a lineplot with Seaborn, using dashed red lines and square markers:

```

import seaborn as sns
sns.set_style('whitegrid')
sns.lineplot(x=x, y=y, color='red', marker='s', linestyle=':')
plt.title('Seaborn plot')
plt.xlabel('x axis')
plt.ylabel('y axis')

```

```
[ ]: import seaborn as sns

sns.set_style('whitegrid')

sns.lineplot(x=x, y=y, color='red', marker='s', linestyle=':')
plt.title('Seaborn plot')
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.show()
```

Scatter plot with Seaborn, this time using the 'ticks' style:

```
sns.set_style('ticks')
sns.scatterplot(x=x, y=y, color='blue', marker='s')
plt.title('Seaborn plot')
plt.xlabel('x axis')
plt.ylabel('y axis')
```

```
[ ]: sns.set_style('ticks')
sns.scatterplot(x=x, y=y, color='blue', marker='s')
plt.title('Seaborn scatter plot')
plt.xlabel('x axis')
plt.ylabel('y axis')
```

Depending where you want to use your figure, the settings for the plot elements need to be different. For example, in a paper you might need to use small font sizes, while for a poster presentation you need to produce plots that are easy to read from a distance. Seaborn offers the function `set_context` that can take any value of notebook (default), paper, talk, and poster. We repeat the line plot, this time in the poster context and using round red markers.

```
sns.set_context('poster')
sns.set_style('ticks')
sns.lineplot(x=x, y=y, color='red', marker='o')
plt.title('Seaborn plot')
plt.xlabel('x axis')
plt.ylabel('y axis')
```

```
[ ]: sns.set_context('poster') # default context: sns.set_context("notebook")
sns.set_style('ticks')
sns.lineplot(x=x, y=y, color='red', marker='o')
plt.title('Seaborn plot')
plt.xlabel('x axis')
plt.ylabel('y axis')
```

Exercise

Now, it is on you to play with matplotlib's and seaborn's capabilities. :)

[ ]: