

Carsten Knoll

Chair of Fundamentals of Electrical Engineering

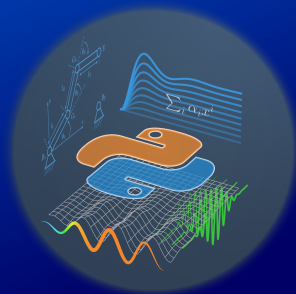
Python for Engineers

Pythonkurs für Ingenieur:innen

2d Visualization with `matplotlib`
2d Visualisierung mit `matplotlib`

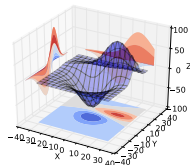
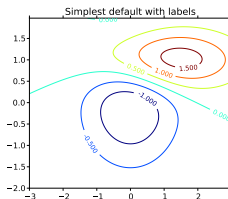
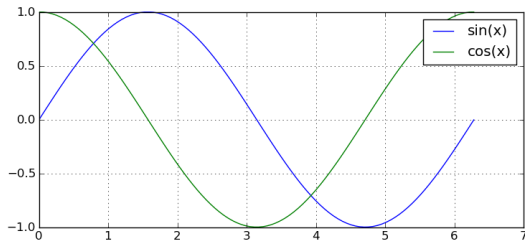
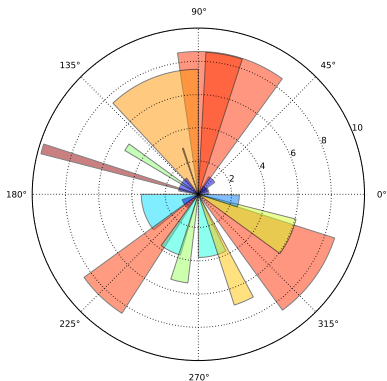
Dresden (Online), 2024-11-08

<https://tu-dresden.de/pythonkurs>
<https://python-fuer-ingenieure.de>



Preview

- Visualization of measurement data, simulation results → better understanding
- Publication-ready graphics for theses, dissertations, scientific articles etc.



Package Overview



- Quasi-standard for 2d plotting with Python
- Also support for some 3d plotting
- Syntax based on Matlab (easy)
- Many plot functions and plot types
- High quality results
- Interactive (zooming, panning)
- Embedding of formulas and symbols
- \LaTeX interface for text and formulas in diagrams
- Saving of plots in as raster graphics (e.g. `png`) or vector graphics (e.g. `pdf`)
- Extensive documentation
- Performance: acceptable, but not ideal for real-time visualization
- Huge example gallery: <https://matplotlib.org/stable/gallery/>

Simple Example

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

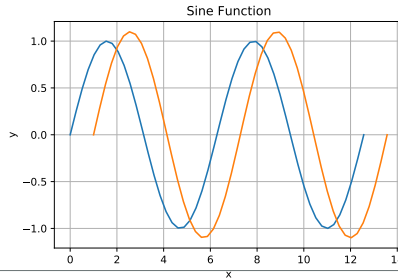
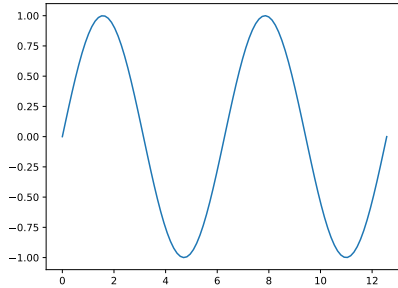
x = np.linspace(0, 4*np.pi, 50)
y = np.sin(x)

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

Customize plot:

```
# 1. create the plot:
plt.plot(x, y)
plt.plot(x+1, y*1.1)

# 2. specify properties:
plt.title('Sine Function')
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



Plot Function

`plot(...)` takes at least one positional argument:

`plot(y)` → plots sequence `y` over its indices

Common pattern: `plot(x, y, 'ro:')` → x-values, y-values, format string
(here: `:` → red, `o` → circle markers, `:` → dotted line),

For more better readability use keyword arguments (kwargs):

```
plt.plot(x, y, color='red', linestyle=':', marker='o', linewidth=2)
```

with short kwargs:

```
plt.plot(x, y, c='red', ls=':', marker='o', lw=2)
```

Automatic color cycling and legend

```
plt.plot(x, np.sin(x), label='sin(x)') # light blue
```

```
plt.plot(x, np.cos(x), label='cos(x)') # orange
```

```
plt.legend() # automatically placed (if not specified by kwarg)
```

See also: docstring of plot function e.g. `plt.plot?` in Jupyter Notebook

Text and Styling

Formulas and symbols via \rightarrow \LaTeX -support

Tip: use raw strings like `r"abc"` to unbind the backslash from special meanings
(like `"\n"` \rightarrow new line character)

```
plt.plot(x[1:], y[1:]/x[1:], label=(r"$\frac{\sin(\phi)}{\phi}$"))  
plt.legend()
```

Fontsize of legend

```
plt.legend(fontsize=20)  # only this time  
# ...  
plt.rcParams['legend.fontsize'] = 18  # from now on
```

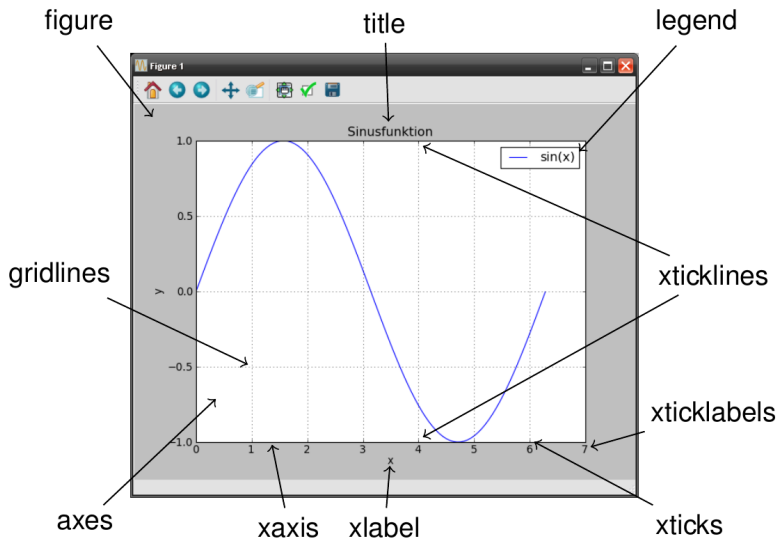
Fontsize of label

```
plt.xlabel('time [s]', fontsize=14)  # only this time  
# ...  
plt.rcParams['axes.labelsize'] = 10  # from now on
```

Change parameter of object after creation

```
line1, = plt.plot(x, y)  
plt.setp(line1, linewidth=3, color="purple")
```

Matplotlib Terminology



Example Script 1

Listing: example-code/matplotlib1.py

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 alpha = np.linspace(0, 6.28, 100)
5 y = np.sin(alpha)
6
7 mm = 1./25.4 # scaling factor millimeter -> inch
8 fig = plt.figure(figsize=(250*mm, 180*mm)) # specify size in mm
9
10 plt.plot(alpha, y, label=r'$\sin(\alpha)$')
11 plt.xlabel(r'$\alpha$ in rad')
12 plt.ylabel('$y$')
13 plt.title('Sine Function')
14 plt.legend() # add legend
15 plt.grid() # toggle grid
16
17 plt.savefig('test.pdf') # file format is inferred from ending (.pdf)
18 plt.show() # display the figure
```

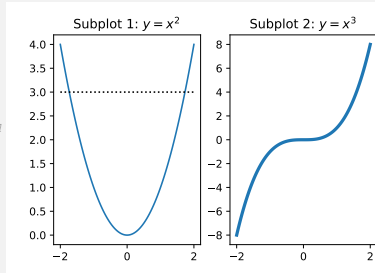
Recommendations:

- Visualization should be implemented separately from slow calculations (e.g. simulation)
- Visualization should be completely automated (no manual interactions)
- Reason: reproducibility; visualization will be created many times before it is ready

Example Script 2

Listing: example-code/matplotlib2.py

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 xx = np.linspace(-2, 2, 100)
5
6 mm = 1./25.4 # scaling factor millimeter -> inch
7 scale = 0.5 # for convenient proportional scaling
8 fs = np.array([250*mm, 180*mm])*scale # specify scaled
9
10 # small right margin
11 plt.rcParams['figure.subplot.right'] = .98
12
13 # subplots: 1 row, 2 columns
14 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=fs);
15
16 ax1.plot(xx, xx**2)
17 ax1.set_title("Subplot 1:  $y=x^2$ ")
18
19 ax2.plot(xx, xx**3, lw=3)
20 ax2.set_title("Subplot 2:  $y=x^3$ ")
21
22 ax1.plot(xx, xx*0+3, ":k") # dotted black horizontal line at bei  $y=3$ 
23
24 plt.savefig('subplots.pdf') # file format is inferred from ending (.pdf)
25 plt.show() # display the figure
```



Where do I get help?

- Matplotlib is very big and complex → see docs:
<https://matplotlib.org/stable/users/index.html>
- Tip 1: Gallery (<https://matplotlib.org/stable/gallery/>)
Many examples (images and corresponding code)
- Tip 2: Axes class documentation
https://matplotlib.org/stable/api/axes_api.html
- All plot and draw functions are accessible via the `axes` class!
 - `ax.plot()` , `ax.bar()` , `ax.scatter()` , `ax.arrow()` , ...
 - Especially important: keyword arguments
- docsrings of objects and functions, e.g. `plt.plot?` in Jupyter
- cheatsheets in appendix or at <https://matplotlib.org/cheatsheets/>

Frequently Used Properties

```
ax = plt.gca() # get current axes object
```

- `ax.set_aspect('equal')` → aspect ratio 1:1
- `ax.set_xlim(0, 10)` → range of X axis values
- `ax.set_xticklabels(['a', 'b'])` → custom labels
- `ax.legend(loc=1)` → position of legend
- `ax.tick_params(**kwargs)` → optics of axis ticks

Discover configurable properties:

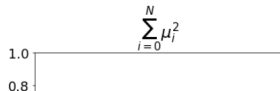
```
import ipydex  
ipydex.dirsearch("size", plt.rcParams)  
# or  
ipydex.dirsearch("tick", plt.rcParams)
```

More on L^AT_EX

Matplotlib comes with its own (reduced) L^AT_EX compiler

See examples above (`example-code/matplotlib2.py`)

or try `plt.title(r"$\sum_{i=0}^N \mu_i^2$")`



More on L^AT_EX

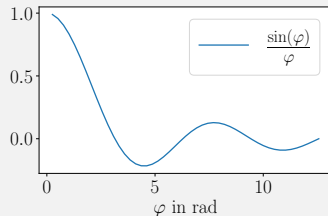
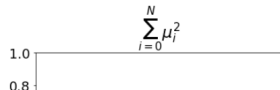
Matplotlib comes with its own (reduced) L^AT_EX compiler

See examples above (`example-code/matplotlib2.py`)

or try `plt.title(r"$\sum_{i=0}^N \mu_i^2$")`

Alternatively, use local L^AT_EX-installation (if present)

```
plt.rcParams["text.use_tex"] = True # activate LaTeX rendering
# optional but looks better:
plt.rcParams["font.family"] = "serif"
# the following function call is spread over multiple lines:
plt.plot(
    x[1:], y[1:]/x[1:],
    label=(r"$\frac{\sin(\varphi)}{\varphi}$")
)
plt.xlabel(r"$\varphi$ in rad")
plt.legend(); plt.show()
```



→ all strings will be rendered with L^AT_EX

Matplotlib - interactive usage

- assumption until now: usage from a script (.py file)
- `plt.show()` opens a new window (or more)
- program is paused until window is closed
- figure can be manipulated interactively (zoom, pan, ...)

Matplotlib - interactive usage

- assumption until now: usage from a script (.py file)
- `plt.show()` opens a new window (or more)
- program is paused until window is closed
- figure can be manipulated interactively (zoom, pan, ...)
- ∃ different behavior: `plt.ion()` # activate interactive mode
- program continues after figure is created
- figure can be manipulated by code after `plt.show()`

Matplotlib - interactive usage

- assumption until now: usage from a script (.py file)
- `plt.show()` opens a new window (or more)
- program is paused until window is closed
- figure can be manipulated interactively (zoom, pan, ...)
- \exists different behavior: `plt.ion()` # activate interactive mode
- program continues after figure is created
- figure can be manipulated by code after `plt.show()`
- Jupyter Notebook: ≥ 2 types of display modes (“backends”)
- activated via “magic commands” (beginning with `%`)

```
%matplotlib inline    # static image; good for exporting
%matplotlib notebook  # interactive widget inside the notebook
```


Summary

- library for 2d-visualization: `matplotlib`
- `plt.plot`, `plt.show`, `plt.savefig`, `plt.rcParams`, ...
- quite complex (many options)
- best practice:
 - customize examples from <https://matplotlib.org/stable/gallery/>

Cheatsheet: Line Styles

∃ two possibilities: short: as format string: `plot(x, y, '-.')`

long: as keyword argument (with long or short key):

`plot(x, y, linestyle='dashed')` or `plot(x, y, ls='--')`

short form	long form	description	output
' ' or ' '	None	without line	
'-'	'solid'	solid line (default) (<u>Vorgabe</u>)	————
'--'	'dashed'	dashed line	-----
'-.'	'dashdot'	line with dashes and dots	-.-.-.-
':'	'dotted'	dotted line

Cheatsheet: Marker

∃ two possibilities; short as format string: `plot(x, y, 'o')`

long as keyword argument: `plot(x, y, marker='h')`

Size with `markersize=10` or `ms=10` adjustable

Examples (there are more):

short form		description	output
' '	oder <code>None</code>	without marker (<i>default</i>)	
'.'		dot	●
'o'		circle	○
'D'		diamond	◇
'H'		hexagon	⬡
'+'		plus	+
's'		square	■

Cheatsheet: Colors

In format string, or as keyword argument

```
plot(x, y, 'g') or plot(x, y, color='green')
```

Gray: numeric value between 0 ($\hat{=}$ black) and 1 ($\hat{=}$ white) as `str` object:

```
plot(x, y, color='0.3')
```

more precise color specification:

- Hexadecimal notation (as in HTML/CSS): `color='#e3e6f7'`
- 3-tuple ($\hat{=}$ Red, Green, Blue): `color=(0.3, 0.8, 0.1)`
- 4-tuple ($\hat{=}$ RGB + Alpha (opacity)): `color=(0.3, 0.8, 0.1, 0.7) # 30% transparency`

Predefined Colors:

short form

long form

'b'

'blue'

'g'

'green'

'r'

'red'

'c'

'cyan'

'm'

'magenta'

'y'

'yellow'

'k'

'black'

'w'

'white'

Extensive cheatsheets:

<https://matplotlib.org/cheatsheets/>