



2D - PLOTTEN/VISUALISIERUNG

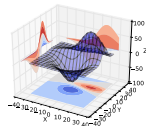
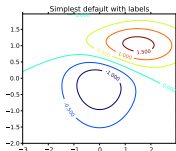
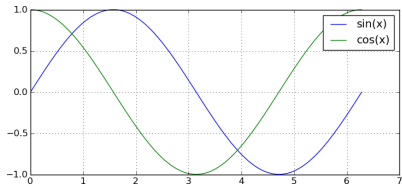
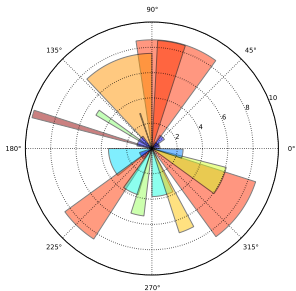
mit matplotlib

Sebastian Voigt, TU Dresden

Dresden, 06.05.2013

Worum geht es?

- Visualisierung von Messdaten, Simulationsergebnissen etc.
- Publikationsfertige Grafiken für Studien- und Diplomarbeiten, Dissertationen, wissenschaftliche Artikel und Beiträge





- Quasi Standard für 2D-Plotten mit Python
- Zunehmend auch 3D-Plotfunktionen
- Syntax an Matlab angelehnt → einfach für Umsteiger
- Sehr viele Plotfunktionen und -arten
- Qualitativ hochwertige Ergebnisse
- Interaktiv (zooming, panning)
- Einbinden von Formeln und Symbolen
- $\text{\LaTeX}$ -Schnittstelle für Text in Diagrammen
- Speichern von Plots in Bildformaten und als PDF
- Umfangreiche Dokumentation
- Nachteil: Performance

<http://matplotlib.sourceforge.net>

Zwei Möglichkeiten der Verwendung

1. interaktiv in der Konsole (pylab) → IPython
2. objektorientiert in Skripten und Anwendungen (pyplot)

```
import matplotlib
from matplotlib.pyplot import * # Matlab-like Umgebung

ion()                               # interaktiv on/off
ioff()
```

→ wird in Spyder-Interpreter bereits importiert (siehe „scientific“)

Einführungsbeispiel interaktiv

```
from matplotlib.pyplot import *
```

```
x = linspace(0,6.28,100)  
y = sin(x)
```

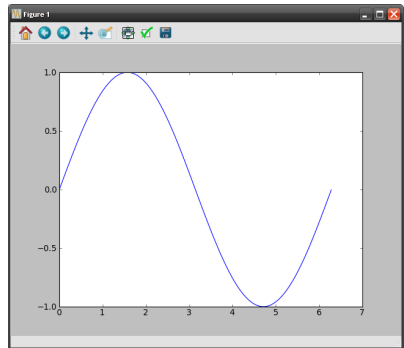
```
plot(x,y)
```

Plot individualisieren

```
title('Sinusfunktion')  
grid()  
xlabel('x')  
ylabel('y')
```

Schließen

```
close()
```



Plot mit Optionen (kurz)

`plot(x, y, 'ro:')` → Farbe, Marker und Linientyp

Plot mit Optionen (lang)

`plot(x, y, color='#FF0000', ls=':', lw=2, marker='o')`

Mehrere Kurven mit einem Befehl

`plot(x, sin(x), 'r--', x, cos(x), 'g:')`

Legende

`plot(x, sin(x), label='sin(x)')`

`plot(x, cos(x), label='cos(x)')`

`legend()`

Sonderzeichen / Formeln → $\text{\LaTeX}$ Support

```
plot(x, y/x, label=(r'$\frac{\sin(\phi)}{\phi}$'))  
leg = legend()
```

Schriftgröße Legende

```
setp(leg.texts, fontsize=20)
```

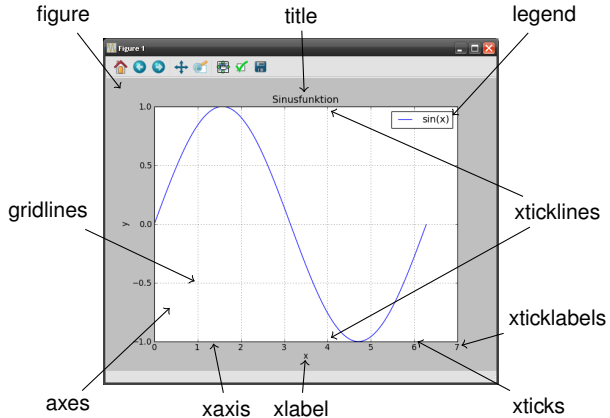
Schriftgröße Label

```
xlabel('Zeit[s]', fontsize=14)
```

Aussehen nachträglich ändern

```
p = plot(x,y)
```

```
setp(p, linewidth=3) → beliebige Anzahl von Optionen
```



Für komplexere Aufgaben wird i.d.R. Skript geschrieben:

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

x = np.linspace(0, 6.28, 100)
y = np.sin(x)

fig = plt.figure(figsize=(8, 6)) # Groesse der Figur in Zoll
ax = fig.add_subplot(1, 1, 1) # Subplot Layout

ax.set_title('Sinusfunktion')
ax.plot(x, y, label='sin(x)')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.legend()
ax.grid()

plt.savefig('test.pdf') # Format wie Dateierendung
plt.show() # ohne show() kein Bild
```

- Matplotlib ist sehr umfangreich und komplex → Doku unter <http://matplotlib.sourceforge.net/contents.html>
- Tipp 1: „gallery“ und von da abgucken
- Tipp 2: Dokumentationen der Axes-Klasse http://matplotlib.sourceforge.net/api/axes_api.html?matplotlib.axes.Axes
- Alle Plot- und Zeichenfunktionen sind über die Axes-Klasse zu erreichen!
 - `plot()`, `bar()`, `scatter()`, `arrow()`, ...
 - Besonders wichtig: Keyword-Argumente
- Siehe auch Cheatsheet im Anhang

- `ax.set_aspect('equal')` → Seitenverhältnis 1:1
- `ax.set_xlim(0,10)` → Wertebereich der X-Achse
- `ax.set_xticklabels(['a', 'b'])` → eigene Beschriftungen
- `ax.legend(loc=1)` → Position der Legende
- `ax.tick_params(kwargs**)` → Optik der Achsenbeschriftung

Grundsätzlich auch:

- `setp()` ermöglicht das setzen von Eigenschaften von Objekten
- `getp()` liefert die Objekte und deren `kwargs` (Introspection)

```
setp(getp(leg, 'texts'), fontsize=10)
```

- Matplotlib bringt einen (reduzierten) L^AT_EX Compiler mit
- Ausprobieren:

- `ax.plot(x,y, label='$sin(x)$')`
 - `ax.set_ylabel('$\mu_2$')`

- Optionen können global in rc gesetzt werden:

```
import matplotlib as mpl
```

```
mpl.rc('font', family = serif,  
      serif = ['Computer Modern Roman'],  
      size = 11)
```

```
mpl.rc('text', usetex=True)
```

→ Damit werden alle strings mit L^AT_EX compiliert

- Bibliothek für 2D-Visualisierung: `matplotlib`
- Sehr umfangreiche Optionen
- Bewährtes Vorgehen:
 - Beispiele in **Gallery** nachschauen und anpassen

Zwei Möglichkeiten: (1) kurz als Format-String

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

(2) lang als Keyword-Argument (Kurz- und Langform zulässig)

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

Kurzform	Langform	Beschreibung	Output
' ' oder ' '	None	ohne Linie	
'-'	'solid'	durchgehende Linie (Vorgabe)	_____
'--'	'dashed'	gestrichelte Linie	- - - - -
'-.'	'dashdot'	Strichpunktlinie	- . - . - .
':'	'dotted'	gepunktete Linie	

Wieder zwei Möglichkeiten: (1) kurz als Format-String

```
plot(x, y, 'o')
```

(2) lang als Keyword-Argument

```
plot(x, y, marker='h')
```

Größe mit `markersize=10` oder `ms=10` einstellbar

Auswahl:

Kurzform	Beschreibung	Output
' ' oder <code>None</code>	ohne Marker (Vorgabe)	
'.'	Punkt	●
'o'	Kreis	○
'D'	Diamant	◇
'H'	Hexagon	⬡
'+'	Plus	+
's'	Quadrat	■

(1) Kurzform im Format-String, oder als Keyword-Agrument

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

(2) Grau: Zahl (als Zeichenkette) zwischen 0 (schwarz) und 1 (weiß) erreichen:

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

(3) exakte RGB-Vorgabe:

- die hexadezimale Notation („HTML“) verwenden: `color='#e3e6f7'`
- ein Tupel für RGB vorgeben: `color=(0.3, 0.8, 0.1)`

Grundlegende Farben sind vordefiniert:

Kurzform	Langform	Farbe
'b'	'blue'	Blau
'g'	'green'	Grün
'r'	'red'	Rot
'c'	'cyan'	Cyan
'm'	'magenta'	Magenta
'y'	'yellow'	Gelb
'k'	'black'	Schwarz
'w'	'white'	Weiß

