Python als Werkzeug in den Ingenieurwissenschaften

Thorbjörn Siaenen, Daryl Tayack

Version v. 29. Januar 2026

Inhaltsverzeichnis

Einleitung

Diese Sammlung zum Thema Python soll dazu helfen, ingenieurwissenschaftliche Aufgabenstellungen schnell nachrechnen zu können. Dazu gehört auch das Erstellen von Grafiken. Dieses Dokument wurde mit Sorgfalt erstellt. Dennoch können Fehler oder Ungenauigkeiten nicht ausgeschlossen werden. Sollten Sie einen Verbesserungsvorschlag haben, senden Sie ihn bitte an:

t.siaenen@ostfalia.de

Teil I
Python als Hilfsmittel in der Ingenieurmathematik

1 Syntax

1.1 Zeilenumbruch im Code

Der Zeilenumbruch wird mit dem \ Backslash gemacht. Allgemein sollte eine Code-Zeile nicht breiter als 79 Zeichen sein. Quelle: PEP 8 Style Guide `https://www.python.org/dev/peps/pep-0008/`

```
z = Q3/((X-xp3)**2+ Y**2)+ \
Q4/((X-xp4)**2+ (Y-yp4)**2)**0.5
```

Alternativ kann der Backslash auch entfallen, wenn Code in runden Klammern steht:

```
z = 15/((20-13)**2
        + 7**2)
```

1.2 Kommentare

Kommentare werden mit einem Lattenkreuz gekennzeichnet

```
# Dies ist ein Kommentar
```

1.3 Mehrzeilenkommentare

Kommentare über mehrere Zeilen werden mit drei Kochkommas gekennzeichnet:

```
'''
Hier␣steht␣ein␣Kommentar
über␣mehrere␣Zeilen
'''
```

1.4 Variablentypen ermitteln

Variablen gehören einer Klasse an, die ermittelt werden kann

```
print(type(12)) # <class 'int'>
print(type(1.2)) # <class 'float'>
print(type('foobar')) # <class 'str'>
```

1.5 Einzelne Variablen löschen

Einzelne Variablen können gelöscht werden

```
x = 42.3
y = 32.1
del x, y # Loescht die Variablen
```

2 Jupyter Lab

2.1 Start von .ipynb-Dateien per Doppelklick (Windows)

Jupyter Lab kann mit einem Doppelklick auf eine .ipynb-Datei gestartet werden, indem eine Batch-Datei zum Starten verwendet wird. Die Batch-Datei `jupyterlaboopenwith.bat` hat folgenden Inhalt (`UN` steht für den Benutzernamen unter dem die Anaconda-Distribution zu finden ist):

```
start C:\Users\UN\anaconda3\pythonw.exe ^
C:\Users\UN\anaconda3\cwp.py ^
C:\Users\UN\anaconda3 "C:\Users\UN\anaconda3/pythonw.exe" ^
"C:\Users\UN\anaconda3/Scripts/jupyter-lab-script.py"  %1
timeout 3
```

Die Batch-Datei wird in einem leicht zu merkenden Verzeichnis gespeichert. Mit dem Kontext-Menü „Öffnen mit...” einer .ipynp-Datei im Windows-Explorer wird die Batch-Datei ausgewählt. Wenn anschließend mit einem Doppelklick auf eine .ipynb-Datei geklickt wird, öffnet sich automatisch Jupyter Lab mit dieser Datei.

2.2 Zugriff auf andere Laufwerke (Windows)

Wenn JupyterLab gestartet wurde, hat es ein bestimmtes Startverzeichnis erhalten. Notebook-Dateien können dabei nicht oberhalb oder außerhalb dieses Verzeichnisses gespeichert und geöffnet werden, nur in Unterverzeichnissen dieses Startverzeichnisses. Der Pfad des Startverzeichnisses wird nicht angezeigt, nur der Inhalt links in der Verzeichnisansicht.

Es kann ein alternatives Startverzeichnis und Startlaufwerk verwendet werden. Dazu wird im Anaconda-Browser eine Kommndozeile geöffnet (Anaconda Navigator → CMD.exe Prompt), dann in das gewünschte Laufwerk (beispielsweise `h:`) und das gewünschte Verzeichnis gewechselt (beispielsweise `cd MeineProjekte\Masterarbeit`). Dann wird mit `jupyter lab` die Entwicklungsumgebung gestartet.

2.3 Hotkeys

- Enter: Wechsle in den Zellenmodus (Zelle wird editiert)
- Esc: Beende den Zellenmodus (Zelle wird nicht mehr editiert)
- Strg + Enter: Führe aktuelle Zelle aus
- Eingabe griechischer Buchstaben: \alpha tab für α

2.4 Sinnvolle Extensions

- jupyter-matplotlib. Dies ermöglicht ein `%matplotlib notebook` erfordert das Paket ipympl. Danach in der Anaconda-Shell:
  - jupyter labextension install @jupyter-widgets/jupyterlab-manager
  - jupyter lab build

2.5 interact

Interact ermöglicht interaktive Berechnungen mit Auswahlfeldern und Zeigern:

```
from __future__ import print_function
from ipywidgets import interact,\
interactive, fixed, interact_manual
import ipywidgets as widgets
def f(x):
        return x*2
interact(f, x=(0.0,10.0))
```

Funktionen mit mehreren Parametern:

```
L = ['eins', 'zwei']
def f(A, B):
        print(A)
        print(B)
        pass
# Laesst den Benutzer fuer A einen
# Wert aus der Liste L auswaehlen:
interact(f, A=L, B=(0,2.0,0.1))
# Uebergibt die gesamte Liste L an
# den Parameter A:
interact(f,A=fixed(L), B=(0,2.0,0.1))
```

2.6 Flickern interaktiver Grafiken verhindern

Am Ende der Funktion, die die Grafik erzeugt muss `clear_output(wait=true)` stehen:

```
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import clear_output
import math
def f(x=0):
        fig, ax = plt.subplots()
        fig.set_size_inches(120/25.4, 80/25.4)
        wanderpunkt = plt.Circle((np.cos(x*2*math.pi), 0),0.1)
        ax.add_artist(wanderpunkt)
        ax.set_xlim([-1.2,1.2])
        ax.set_ylim([-0.8,0.8])
        clear_output(wait=True)
interact(f, x=(0.0,1.0,0.01))
```

2.7 Anzeigegröße von Grafiken ändern

Wenn Grafiken beispielsweise mit Matplotlib erzeugt werden, haben sie eine Standardgröße, die recht klein ist. Sie kann wie folgt mit dem Parameter `dpi` verändert werden (150 = groß):

```
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
%matplotlib inline
import matplotlib.pyplot as mpl
from IPython.display import clear_output
mpl.rcParams['figure.dpi'] = 150
```

2.8 Anzeige von Markdown-Code aus einer Python-Zelle

Mit folgendem Code kann in Python Markdown-Code erzeugt werden, der dann in Jupyter Lab als gesetzte Elemente angezeigt werden:

```
from IPython.display import display, Markdown, Latex
display(Markdown('*hervorgehobener␣Text*␣\n\n'+
                 'Formel:␣$\sin(2\,\pi\,f\,t)$␣\n'+
                 '␣#␣Ueberschrift␣\n'+
                 'Dies␣ist␣die␣erste␣Zeile\n'+
                 '|␣spalte␣1␣|␣spalte␣2␣|\n'+
                 '|␣--------␣|␣--------␣|\n'+
                 '|␣3,54␣␣␣␣␣|␣3,24␣␣␣␣␣|\n'+
                 '|␣3,54␣␣␣␣␣|␣3,24␣␣␣␣␣|\n'+
                 '|␣--------␣|␣--------␣|\n'+
                 '*␣Aufzaehlung␣Punkt␣1\n'+
                '*␣Aufzaehlung␣Punkt␣2'))
```

Dies erzeugt folgende Ausgabe:

hervorgehobener Text

Formel: $\sin(2\,\pi\,f\,t)$

Ueberschrift

Dies ist die erste Zeile

spalte 1	spalte 2
3,54	3,24
3,54	3,24
--------	--------

- Aufzaehlung Punkt 1
- Aufzaehlung Punkt 2

2.9 Automatisiert erstellte Tabelle in Markdown

```
from IPython.display import display, Markdown, Latex
import numpy as np
```

```
mdstring  = '| spalte 1 | spalte 2 |\n| :-------- | -------- |\n'
for x in np.linspace(0,3.14, 4):
        mdstring = mdstring + "{:1.4} | {:1.4}\n".format(x,x**2)
display(Markdown(mdstring))
print(mdstring)
```

Dies ergibt:

spalte 1	spalte 2
0.0	0.0
1.047	1.096
2.093	4.382
3.14	9.86

```
| spalte 1 | spalte 2 |
| :-------- | -------- |
0.0 | 0.0
1.047 | 1.096
2.093 | 4.382
3.14 | 9.86
```

2.10 Eingabe griechischer Buchstaben

In Code-Zellen können mit LaTeX-Befehlen einhzelne griechische Buchstaben gesetzt werden, indem zuerst der Rückschrägsttrich (\), dann der Name des griechischen Buchstabens (beipielsweise pi) eingegeben wird und dann die Tabulatortaste gedrückt wird. Es erscheint ein Auswahldialog mit einem Eintrag und der kann mit der Eingabetaste geschlossen werden. Beispielsweise erzeugt `\pi tab` den Buchstaben π.

2.11 Markdown

2.11.1 Zwischenzeilenformeln

Formeln innerhalb eines textes können mit LaTeX beschrieben werden und werden in Dollar-Zeichen eingesetzt.

Beispielcode:

```
Satz des Pythagoras: $x^2 + y^2 = z^2$
```

#

Ausgabe:

Satz des Pythagoras: $x^2 + y^2 = z^2$

2.11.2 Abgesetzte Formeln

Abgesetzte Formeln stehen in einer separaten Zeile. Abgesetzte Formeln werden mit zwei Dollarzeichen eingefasst. Beispielcode:

```
Satz des Pythagoras: $$x^2 + y^2 = z^2$$
```

Ausgabe:

Satz des Pythagoras:

$$x^2 + y^2 = z^2$$

3 Zahlen-Operationen

3.1 Komplexe Zahlen

Eine komplexe Zahl wird mit einem angehangenen `j` gekennzeichnet. Beispiel: $a = 3 + \mathrm{j}\,4$

```
a = 3+4j
```

Euler-Form:

```
import cmath
from math import pi
z = 3*cmath.exp(1j*2*pi/6)
print(z) #1.5+2.59j
```

3.2 Real- und Imaginärteil extrahieren

```
a = 3+4j
a.real # = 3
b.imag # = 4
```

3.3 Quadrieren

3^2

```
3**2
```

3.4 Ganzzahlige Division

Division ohne Rest. Beispiel: $14 \div 4 = 3\,\text{Rest}\,2$

```
14//4  # ergibt 3
```

3.5 Modulo-Operator (Divisionsrest)

Rest einer Division. Beispiel: $14 \div 5 = 2 \text{ Rest } 4$

```
rest = 14%5 # ergibt 4
```

Alternativ

```
import numpy as np
rest = np.mod(4.32, 2) # ergibt 0.32
```

3.6 Aufrunden

Aufrunden auf die nächste Ganzzahl

```
import math
math.ceil(3.14) #ergibt 4
```

3.7 Abrunden

Aufrunden auf die nächste Ganzzahl

```
import math
math.floor(3.14) #ergibt 3
```

3.8 Vergleiche Float und Integer

Bei Zahlenvergleichen wird gerundet:

```
1.0000000000000001==1 # True
1.000000000000001==1 # False
```

4 Mathematische Funktionen

4.1 Fakultät (x!)

Die Faktultät einer ganzen Zahl ist das Produkt aller Zahlen von 1 bis zu dieser Zahl:

$$n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (n-2) \cdot (n-1) \cdot n$$

Beispiel:

$$7! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 = 5040$$

```
import math
print(math.factorial(5)) # 5! = 120
```

4.2 arctan2

Der Winkel eines Zeigers auf eine Koordinate mit x- und y-Anteil, kann mit der Funktion `arctan2` berechnet werden. Diese Funtkioniert in jedem Quadranten. Das folgende Code-Beispiel zeigt die Berechnung des Winkels des Zeigers auf die Koordinate (-3;-3). Das Ergebnis liegt im Bogenmaß vor.

```
import numpy as np
phi = arctan2(-3,-3)
```

4.3 Exponentialfunktion

Die Funktion 10^x kann über 2 Varianten berechnet werden:

```
import numpy as np
x = 3
a = np.power(10, x)
import math
b = math.pow(10, x)
```

5 Komplexe Zahlen

Komplexe Zahlen können mit dem Basis-Python verwendet werden:

```
z = 4 + 3j
```

Leider wird bei der Multiplikation mit einer komplexen Zahl mit `j` das Ergebnis in eine Float-Variable geändert. Beispiel:

```
from sympy import Rational
print(4+Rational(1,3)*1j)
# 4 + 0.3333333333333333*I
```

Wenn exakte Zahlen, also Brüche, gefordert sind, kann dies mit der Bibliothek sympy berechnet werden. Dazu wird die komplexen Einheit `I` aus der sympy-Bibliothek importiert und verwendet:

```
from sympy import Rational I
print(4+Rational(1,3)*I)
# ergibt: 4 + I/3
print(4+Rational(1,3)*I**2)
# ergibt: 11/3
```

6 Numpy/Numerik

6.1 Numpy Array erzeugen

Numpy Array erzeugen aus einem normalen Array

```
import numpy as np
x = np.array([1, 2, 3])
```

6.2 Numpy Arrays verbinden

Mehrere Numpy-Arrays konnen wie folgt zu einem gemeinsamen Numpy-Array verbunden werden:

```
import numpy as np
x = np.concatenate([np.array([1]), np.array([2, 3, 4]), np.array([5])])
# ergibt array([1, 2, 3, 4, 5])
```

6.3 Numpy Matrix aus Vektoren, vertikal

Mehrere Numpy-2D-Arrays konnen wie folgt zu einer gemeinsamen Numpy-Matrix verbunden werden:

```
import numpy as np
hv1 = np.array([11, 12, 13])
hv2 = np.array([21, 22, 23])
Mh = np.vstack((hv1, hv2))
# ergibt: [[11 12 13]
           [21 22 23]]
```

6.4 Numpy Matrix aus Vektoren, horizontal

Mehrere Numpy-2D-Arrays konnen wie folgt zu einer gemeinsamen Numpy-Matrix nebeneinander verbunden werden:

```
import numpy as np
v1 = [[11], [21], [31]]
v2 = [[12], [22], [32]]
# Argument als Tupel mit ( ... )
# Die Argumente werden automatisch
# zu np.arrays gemacht
Mh = np.hstack((v1, v2))
# ergibt: [[11 12]
           [21 22]
           [31 32]]
```

6.5 Kreuzprodukt

$$\vec{a} \times \vec{b} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

```
import numpy as np
x = [1, 0, 0]
y = [0, 1, 0]
np.cross(x, y) # ergibt: array([0, 0, 1])
```

6.6 Länge (Norm) eines Vektors

$$|\vec{a}| = \left|\begin{pmatrix} 3 \\ 4 \\ 12 \end{pmatrix}\right| = \sqrt{3^2 + 4^2 + 12^2} = 13$$

```
import numpy as np
v  = np.array([3,4,12])
print(np.linalg.norm(v)) # ergibt: 13.0
```

6.7 Skalarprodukt

$$\vec{a} \cdot \vec{b} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} = 1 \cdot 2 + 0 \cdot 1 + 0 \cdot 0 = 2$$

```
import numpy as np
x = [1, 0, 0]
y = [2, 1, 0]
np.dot(x, y) # ergibt: 2
```

6.8 nparray 1 D adressieren

Ein eindimensionales Numpy-Array (Liste von Zahlenwerten) hat im Gegensatz zum normalen Array mehr Anwendungsmöglichkeiten.

```
import numpy as np
x = np.arange(0,7)
print(x)
# ergibt [0 1 2 3 4 5 6 7]
print(x[1:4]) #1 bis 3, 4 fehlt
print(x[1:]) # 1 bis ende
print(x[:-2]) # ohne letzte 2
```

6.9 nparray 1 D Differenzen

Differenzen zwischen Werten in einem Numpy-Array berechnnen:

```
import numpy as np
x = np.array([0, 1, 3, 4, 5])
print(np.diff(x))
# ergibt [1 2 1 1]
```

Das Ergebnis hat ein Element weniger als das Argument der Funktion `diff`.

6.10 Numpy arrays

Verschiedene Methoden zur Arrayerzeugung:

```
import numpy as np
a = np.array([0, 1, 3, 4, 5])
# Start, Stop, Schrittweite:
b = np.arange(-3,4,0.5)
# Start, Stop, Anzahl Zahlen
c = np.linspace(-3,4,100)
```

6.11 nparray 2 D

Ein zweidimensionales Numpy-Array hat im Gegensatz zum normalen Array mehr Anwendungsmöglichkeiten (Transponieren, Matrizenmulitplikation) und wird mit folgendem Code erzeugt:

```
import numpy as np
A = np.array([[11, 12, 13],\
  [21, 22, 23]])
```

6.12 nparray Größe

Gesamtzahl der Elemente in einem Array und Anzahl Zeilen und Spalten

```
import numpy as np
x = np.array([[11, 12], [21, 22], [31, 32]])
print(x.shape) # ergibt (3, 2)(Zeilen, Spalten)
print(x.size) # ergibt 6
```

6.13 nparray transponieren

Ein Numpy-Array wird folgendermaßen transponiert::

```
import numpy as np
A = np.array([[11, 12, 13],\
  [21, 22, 23]])
B = A.transpose()
```

6.14 Inverse Matrix (Matrix invertieren)

Eine quadratische Matrix wird wie folgt invertiert:

```
import numpy as np
X = np.array([[2, 3], [3, 4]])
X_inv = np.linalg.inv(X)
# ergibt [[-4, 3], [3, -2]]
```

6.15 Pseudoinverse einer Matrix

Eine quadratische Matrix wird wie folgt invertiert:

```
import numpy as np
X = np.array([[2, 3], [3, 4], [4, 6]])
np.linalg.pinv(X)
# ergibt [[-0.8, 3, -1.6], [0.6, -2, 1.2]]
```

6.16 nparray umdrehen

Die Reihenfolge aller Element ein einem numpy-array wird umgedreht:

```
import numpy as np
A = np.flip(np.array([1, 2, 3])) # [3, 2, 1]
```

6.17 Lineares Gleichungssystem lösen

Ein lineares Gleichungssystem in Matrizenschreibweise

$$\begin{pmatrix} 1 & 3 & -5 \\ -4 & 0 & -4 \\ -3 & 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ -3 \\ -3 \end{pmatrix}$$

wird wie folgt gelöst:

```
import numpy as np
M = np.array([[1, 3, -5],
    [-4, 0, -4],[-3, 0, -1]])
b = np.array([[2],[-3],[-3]])
x = np.linalg.solve(M,b)
print(x)
```

Dies ist auch mit komplexen Werten möglich. Das Gleichungssystem

$$\begin{pmatrix} 1+\mathrm{j}\,5 & 3-\mathrm{j}\,2 & -5 \\ 3+\mathrm{j}\,2 & 0 & -4 \\ -5 & -4 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ -3 \\ -3 \end{pmatrix}$$

kann wie folgt gelöst werden:

```
import numpy as np
M = np.array([
    [ 1+5j,  3-2j, -5],
    [ 3+2j,  0   , -4],
    [-5   , -4   , -1]])
b = np.array([[2],[-3],[-3]])
x = np.linalg.solve(M,b)
print(x)
```

6.18 Polynomdivision

Eine Polynomdivision wird wie folgt mit der Bibliothek numpy durchgeführt. Beispiel:

$$\frac{2\,x^5 - 11\,x^4 + 18\,x^3 - 20\,x^2 + 56\,x - 65}{x^3 - 7\,x^2 + 16\,x - 12}$$

```
import numpy as np
# Zaehlerkoeffizienten:
Z = np.array([2, -11, 18, -20, 56, -65])
# Nennerkoeffizienten:
N = np.array([1, -7, 16, -12])
E = np.polydiv(Z, N)
```

```
print(E) # ergibt (array([2., 3., 7.]), array([  5., -20.,  19.]))
def pl(kp):
    if kp == 0:
        return("")
    else:
        return("␣+␣")
A = "".join([pl(k)+"("+str(E[0][k])+")*x^"+str(len(E[0])-k-1) for k in range(len(E[0]))])
B = "".join([pl(k)+"("+str(E[1][k])+")*x^"+str(len(E[1])-k-1) for k in range(len(E[1]))])
C = "".join([pl(k)+"("+str(N[k])+")*x^"+str(len(N)-k-1) for k in range(len(N))])
print( A +"␣+␣("+B+")/("+C+")" )
```

Ergebnis:

$$2\,x^2 + 3\,x + 7 + \frac{5\,x^2 - 20\,x + 19}{x^3 - 7\,x^2 + 16\,x - 12}$$

6.19 Integrale (Trapezregel)

Wenn eine Kurve in x-y-Koordinaten vorliegt, kann mit der Trapezregel die Fläche berechnet werden. Beispiel Halbkreis:

```
import numpy as np
import scipy.integrate
t = np.linspace(0,3.14159265,10000)
x = np.cos(t)
y = np.sin(t)
# Achtung: zuerst y, dann x:
A = scipy.integrate.trapezoid(y,x)
print(A*2) # = -pi
# -pi, weil die x-werte absteigend
```

6.20 Potenz einer Matrix

Einzelne Elemnte einer Matrix oder ganze Zeilen oder Spalten können folgendermaßen manipuliert werden. Beispiel

$$\begin{pmatrix}1&0&0\\0&2&0\\0&0&3\end{pmatrix}\cdot\begin{pmatrix}1&0&0\\0&2&0\\0&0&3\end{pmatrix}\cdot\begin{pmatrix}1&0&0\\0&2&0\\0&0&3\end{pmatrix}=\begin{pmatrix}1&0&0\\0&2&0\\0&0&3\end{pmatrix}^3$$

```
import numpy as np
e=np.array([[1,0,0],[0,2,0],[0,0,3]])
ep=np.linalg.matrix_power(e,3)
```

6.21 Integral einer Funktion

Wenn eine Funktion gegeben ist, kann mit numerisch das Integral berechnet werden: Beispiel $\int_{x=0}^{4} x^2\,\mathrm{d}x$

```
from scipy import integrate
y = lambda x:x**2
integrate.quad(y,0,4)[0]
```

Erhöhte Genauigkeit wird wie folgt erreicht (parameter epsabs und epsrel, keine Zahl = große Genauigkeit):

```
from scipy import integrate
y = lambda x:x**2
integrate.quad(y,0,4,epsabs=1.49e-13, epsrel=1.49e-13)[0]
```

6.22 Zweidimensionales Integral

Mehrfachintegral am Beispiel:

$$R = \int_{a=2}^{5} \int_{b=\sin(a)}^{a^2} a \cdot b \, \mathrm{d}b \, \mathrm{d}a$$

```
import numpy as np
from scipy import integrate

def bunten(apar):
    return(np.sin(apar))

def boben(apar):
    return(apar**2)

# der erste Parameter ist die Integrationsvariable des inneren Integrals
def ing(bpar, apar):
    return(apar*bpar)

R = integrate.dblquad(ing, 2, 5, bunten,
  boben, epsabs=1.5e-8, epsrel=1.5e-8)[0]
print(R) # ergibt 1293.96
```

6.23 Integral einer komplexwertigen Funktion

Das Integral einer komplexwertigen Funktion wird gebildet, indem Realteil und der Imaginäteil getrennt voneinander integriert werden: $\int_{t=0}^{2} \mathrm{e}^{\mathrm{j}\pi t} \, \mathrm{d}t = \int_{t=0}^{2} \mathrm{Re}\left(\mathrm{e}^{\mathrm{j}\pi t}\right) \mathrm{d}t + \mathrm{j} \int_{t=0}^{2} \mathrm{Im}\left(\mathrm{e}^{\mathrm{j}\pi t}\right) \mathrm{d}t$

```
import numpy as np
import math
from scipy import integrate
y_re=lambda x: np.real(\
  np.exp(1j*math.pi*x))
y_im=lambda x: np.imag(\
  np.exp(1j*math.pi*x))
z = integrate.quad(y_re,0,2)[0]+\
  1j*integrate.quad(y_im,0,2)[0]
print(z)
```

6.24 Dreifachintegral

Das Integral: $I = \int_{z=0}^{9} \int_{\varphi=\frac{-\pi}{6}}^{\frac{\pi}{6}} \int_{r=\cos(\varphi)}^{7} f(r,\varphi,z) \, \mathrm{d}r \, \mathrm{d}\varphi \, \mathrm{d}z$ mit $f(r,\varphi,z) = r^3 7$ wird wie folgt berechnet:

```
from scipy import integrate
from math import pi, cos
import numpy as np
def integrant(r, phi, z):
    return(r**3)*7
E,F=integrate.tplquad(integrant,0,9,\
  lambda z:-pi/6, lambda z: pi/6,\
  lambda z, phi: cos(phi),\
  lambda z, phi: 7)
print(E) # Ergebnis
print(F) # Fehler
```

Wichtig: Die Integrationsvariable des innersten Integrals ist das erste Argument der Funktion. Die Grenzen des innersten Integrals sind die letzten beiden Argumente der Funktion tplquad. Merke: Erstes Argument des Integranden ↔ Letze beiden Argumente der Funktion tplquad

6.25 DGL 1. Ordnung numerisch lösen

Lösung der DGL $y' = 2\,t\,y + 5\,t$

```
from scipy.integrate import solve_ivp
import numpy as np
#Reihenfolge wichtig: erst t dann y!
def dydt(t,y):
    retval =  2*t*y + 5*t
    return(retval)
tend=0.017
t = np.linspace(0,tend,100)
s=solve_ivp(dydt, [0, tend], [3], t_eval=t, rtol=1e-9)
import matplotlib.pyplot as mpl
fig, ax = mpl.subplots()
ax.plot(s.t, s.y[0])
ax.grid()
ax.set_ylim([0,50])
yana = 11/2*np.exp(t**2)-5/2
ax.plot(t, yana, 'rx')
```

6.25.1 Faktorisierung eines Polynoms

Faktorisierung eines Polynoms am Beispiel $f(x) = x^4 + 4\,x^2$:

```
from sympy import *
x = symbols('x')
y = factor(x**4+4*x**2)
print(y) # ergibt x**2*(x**2 + 4)
```

6.25.2 Koeffizienten eines Polynoms

Berechnet werden die Koeffizienten des Polynoms, welches faktorisiert $(y-2)\cdot(y-3)$ lautet. Ausmultipliziert ist dies $y^2 - 5\,y + 6$. Die Koeffzienten sind also 1, -5 und 6.

```
import sympy as sp
y = sp.symbols('y')
exp = (y-2)*(y-3) # y^2 -5*y + 6
Ply = sp.Poly(exp,y)
print(Ply.all_coeffs()) # ergibt [1, -5, 6]
```

6.25.3 Einheitsmatrix

Einheitsmatrix

```
import numpy as np
# Datentyp float (1.0):
e = np.eye(4)
# Datentyp int (1):
eint = np.eye(4, dtype=int)
```

6.25.4 Determinante

Berechnung der Determinante einer Matrix

```
import numpy as np
M = [[ 4,  2],\
     [ 3, -2]]
print(np.linalg.det(M))
```

6.25.5 Elementmanipulation

Einzelne Elemnte einer Matrix oder ganze Zeilen oder Spalten können folgendermaßen manipuliert werden

```
import numpy as np
e = eye(3) # Matrix erzeugen
e[:,3]=7 # dritte Spalte alle 7
e[3,:]=7 # dritte Zeile alle 7
e[0:2,:]=7 # erste zwei Zeilen 7
# Die Auswahl betrifft alle
# bis zur 2. Zeile
```

6.25.6 Matrizenmultiplikation

Zwei Matrizen können mit den folgenden drei Methoden multipliziert werden (Es gibt noch mehr.):

```
import numpy as np
a = np.array([[1, 2],[3, 4]])
amala1 = np.matmul(a,a)
amala2 = a@a
amala3 = a.dot(a)
```

6.25.7 Vektoren und Matrizen aneinanderhängen

Die Matrix $\begin{pmatrix} 11 & 12 \\ 21 & 22 \\ 31 & 32 \end{pmatrix}$ und der Vektor $\begin{pmatrix} 13 \\ 23 \\ 33 \end{pmatrix}$ werden folgendermaßen aneinandergehangen:

```
import numpy as np
a = np.array([[11, 12],\
   [21, 22], [31, 32]])
b = np.array([[13], [23], [33]])
print(np.c_[a,b])
```

Ergebnis:

```
[[11 12 13]
 [21 22 23]
 [31 32 33]]
```

Die Matrix $\begin{pmatrix} 11 & 12 \\ 21 & 22 \end{pmatrix}$ und der Vektor $\begin{pmatrix} 31 & 32 \end{pmatrix}$ werden untereinander folgendermaßen aneinandergehangen:

```
import numpy as np
a = np.array([[11, 12], [21, 22]])
b = np.array([[31, 32]])
print(np.r_[a,b])
```

Ergebnis:

```
[[11 12]
 [21 22]
 [31 32]]
```

6.25.8 XY-Matrizen

Im Berereich Datenvisualisierung benötigt man gelegentlich zwei Matrizen gleicher Größe, die jeweils x- und y-Koordinaten angeben. Diese können mit dem Befehl `meshgrid` erzeugt werden. Der Vektor `yv` muss dabei in fallender Richtung (erste Grenze größer als die zweite) und der Vektor `xv` in steigender Richtung (erste Grenze kleiner als die zweite) erzeugt werden.

```
import numpy as np
xv=np.linspace(2,4,3)
yv=np.linspace(2,-2,5)
Mx,My=np.meshgrid(xv,yv)
```

Ergebnisse:

$$\vec{x} = \begin{pmatrix} 2 & 3 & 4 \end{pmatrix} \quad \vec{y} = \begin{pmatrix} 2 & 1 & 0 & -1 & -2 \end{pmatrix}$$

$$\vec{\vec{X}} = \begin{pmatrix} 2 & 3 & 4 \\ 2 & 3 & 4 \\ 2 & 3 & 4 \\ 2 & 3 & 4 \\ 2 & 3 & 4 \end{pmatrix} \quad \vec{\vec{Y}} = \begin{pmatrix} 2 & 2 & 2 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \\ -2 & -2 & -2 \end{pmatrix}$$

6.25.9 Zufallszahlen

Integer Zufallszahl zwischen 1 und 90

```
import random
z = random.randint(1,90)
```

Startwert-seed für den Zufallsgenerator setzten. Die danach erzeugten Zufallszahlen sehen immer gleich aus.

```
import random
random.seed(1)
z = random.randint(1,90)
```

Reelle Zufallszahl zwischen 0,0 und 1,0

```
import random
random.seed(1)
z = random.rand()
```

7 Zufallszahlen mit Normalverteilung mit dem Mittelwert 2 und der Streuung 0,1 erzeugen:

```
import numpy as np
np.random.seed(0)
z = np.random.normal(2, 0.1, 7)
```

Mit `np.random.seed(0)` wird der Zufallsgenerator auf den Startwert 0 gesetzt. Nachfolgende Aufrufe von `np.random` erzeugen eine immer gleiche Abfolge von Zufallszahlen.

6.25.10 lineare Differenzialgleichung n-ter Ordnung lösen

Eine lineare inhomogene DGL mit konstanten Koeffizienten kann in folgende explizite Form gebracht werden:

$$y^{(n)}(t) = \overbrace{a_0\, y(t) + a_1\, y'(t) + \ldots + a_{n-1}\, y^{(n-1)}(t)}^{\text{steigende Ableitungen}} + g(t)$$

Darin ist $y(t)$ die gesuchte Funktion und $g(t)$ das Störglied.
Diese DGL wird zunächst in Systemdarstellung überführt:

$$\begin{pmatrix} y'(t) \\ y''(t) \\ y'''(t) \\ \vdots \\ y^{(n)}(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ \vdots & 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \ldots & 0 & 1 \\ a_0 & a_1 & a_2 & \ldots & a_{n-1} \end{pmatrix} \cdot \begin{pmatrix} y(t) \\ y'(t) \\ y''(t) \\ \vdots \\ y^{(n-1)} \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ g(t) \end{pmatrix}$$

Darin ist der grün eingezeichnete Bereich eine Einheitsmatrix und die blauen Bereiche sind mit Nullen gefüllt.

Die rechte Seite wird für die numerische Lösung als Funktion definiert. Dies wird Anhand des Beispiels

$$\underbrace{\begin{pmatrix} 0 & 1 \\ -3 & -4 \end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix} y(t) \\ y'(t) \end{pmatrix}}_{\boldsymbol{y}} + \underbrace{\begin{pmatrix} 0 \\ 200\, \sin(6\,\pi\, t) \end{pmatrix}}_{\boldsymbol{b}}$$

gezeigt:

```
from scipy.integrate import solve_ivp
import numpy as np
import matplotlib.pyplot as mpl
import os

def dydt(tp,yT):
    A = np.array([[0, 1],[-3, -4]])
    b = np.array([[0],[200*np.sin(6*pi*tp)]])
    retval = ((np.array([yT@(A.transpose())]).transpose())+b).transpose()[0]
    return(retval)
```

Die ungewöhnliche Darstellung mit den Transponierten wurde gewählt, weil die Zustandsvariablen `yT` als Zeilenvektoren übergeben werden und in der mathematischen Systemdarstellung eigentlich Spaltenvektoren verwendet werden.

Es gilt $(A\,y)^{\mathrm{T}} = y^{\mathrm{T}}\,A^{\mathrm{T}} \Rightarrow A\,y = \left(y^{\mathrm{T}}\,A^{\mathrm{T}}\right)^{\mathrm{T}}$. Darin ist y^T der Zeilenvektor `yT` in Python. Bei der Berechnung des Rückgabewertes `retval` wird am Ende das erste Element aus einer Matrix mit `[0]` extrahiert. Das ist notwendig, weil als Rückgabewert ein Zeilenvektor erwartet wird. Ohne die Extraktion wäre es eine Matrix mit einer Zeile und zwei Spalten.

Mit dem nachfolgenden Code wird das Anfangswertproblem (die DGL) mit den Anfangswerten `[3, 0]` gelöst. Dies steht für $y(0) = 3$ und $y'(0) = 0$. In Matrizenschreibweise ist dies

$$\begin{pmatrix} y(0) \\ y'(0) \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \end{pmatrix}.$$

Die numerischen und die analytischen Werte werden zu den Zeitpunkten ermittelt, die in `t` angegeben sind. Ein graphischer Vergleich der analytischen und der numerischen Lösung wird als PDF-Grafik gespeichert. Ebenfalls wird die maximale Abweichung zwischen den ermittelten Funktionwerten ausgegeben. Der Parameter `rtol` ist standardmäßig `1e-3` und beschreibt die relative Genauigkeit des Lösers.

```
tend=4
t = np.linspace(0,tend,175)
s=solve_ivp(dydt, [0, tend], [3, 0], t_eval=t, rtol=1e-9)
yana = np.sin(6*pi*t) #ana. Lsng
fig, ax = mpl.subplots()
ax.grid()
g1, = ax.plot(t, yana, 'g-', label='analytisch')
g2, = ax.plot(t, s.y[0], 'rx', label='numerisch')
ax.legend(handles=[g1,g2])
print("max.␣Diff.␣ana.-num.␣Lsng:␣{:f}".format(max(s.y[0]-yana)))
import os
mpl.savefig(os.path.basename(__file__).replace('.py','') + '_gen.pdf')
# Mit Jupyter: mpl.savefig(os.path.basename(sys.argv[0]).replace('.py','') + '_gen.pdf')
mpl.close("all")
```

6.25.11 nichtlineare Differenzialgleichung n-ter Ordnung

Gegeben sei die nichtlineare DGL $y''(t) = -1/\left(t + y(t)^2\right)$. Dies wird zunächst in eine vektorielle Schreibweise überführt:

$$\begin{pmatrix} y(t) & y'(t) \end{pmatrix}' = \begin{pmatrix} y'(t) & \frac{-1}{t+y(t)^2} \end{pmatrix}$$

Die Lösung kann mit `solve_ivp` berechnet werden. Die Funktion `nonlinearfunktion` beschreibt die DGL in der vektoriellen Schreibweise. Der Funktionsverlauf wird an denjenigen Zeitpunkten ausgegeben, die mit `t_eval` definiert sind.

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as mpl
def nonlinearfunktion(t,Yvec):return np.array([Yvec[1], -1/(t+Yvec[0]**2)])
t0 = 0 # Startzeit
tend = 20.5 # Endzeit
Yvecnull = [3, 0] # Anfangswerte [y(0), y'(0)]
R =solve_ivp(nonlinearfunktion,[t0, tend], Yvecnull, t_eval=np.linspace(t0,tend,25))
```

```
# Ergebnis plotten:
fig, ax = mpl.subplots()
ax.plot (R.t, R.y[0], 'rx')
ax.grid()
```

6.26 Werteliste nahe Null zu Null runden

Werte in der Nähe zu Null zu Null runden

```
import numpy as np
def tszero(v):
  return(np.array([int(not(x)) for x in np.isclose(v,0.0*v)])*v)
print(tszero(np.array([0.234]))) # [0.234]
print(tszero(np.array([0.234e-14]))) # [0.]
```

6.27 Abschnittsweise definierte Funktionen

Abschnittsweise definierte Funktionen können mit if/then/else-Konstruktionen erzeugt werden oder mit np.piecewise.

```
import numpy as np
import matplotlib.pyplot as plt
def fvonx(x):
    fx = np.piecewise(x, [x<3,           (3<=x) & (x<3.5),   (3.5<=x) ],
         [lambda x: 0.0, lambda x: (x-3)**2, lambda x: -(x-4)**2+1/2])
    return(fx)
fig, ax = plt.subplots()
x = np.linspace(0,7,150)
y = fvonx(x)
ax.plot(x,y,'r-')
ax.grid()
plt.show()
```

6.28 Heaviside-Funktion (Sprungfunktion)

Die Heaviside- oder Sprungfunktion ist wie folgt defindiert:

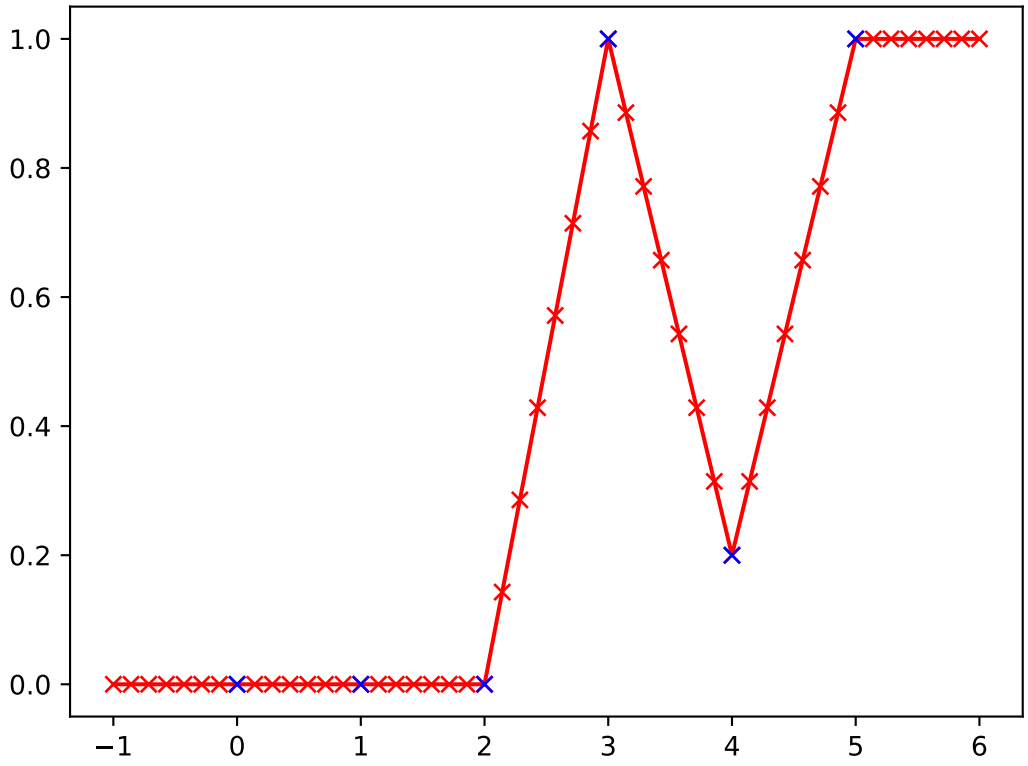
$$f(t) = \begin{cases} 1 & 0 < t \\ x & t = 0 \\ 0 & t < 0 \end{cases}$$

```
import numpy as np
x = 2
t = 3.14
print(np.heaviside(t, x)) # ergibt 1
print(np.heaviside(0, x)) # ergibt 2
print(np.heaviside(-10, x)) # ergibt 0
```

6.29 Lineare Interpolation

Eine lineare Interpolation berechnet y-Werte zwischen gegebenen xy-Stützwerten. Das nachfolgende Programm berechnet 50 Werte aus 6 Stützwerten. Die Stützwerte sind in der Grafik mit blauen Kreuzen gekennzeichnet und die interpolierten Werte mit roten Kreuzen.

```
import numpy as np
import matplotlib.pyplot as plt
x_stuetz = [0,1,2,3,4,5]
y_stuetz = [0,0,0,1,0.2,1]
x = np.linspace(-1,6,50)
y = np.interp(x, x_stuetz, y_stuetz)
fig, ax = plt.subplots()
ax.plot(x,y, 'rx-')
ax.plot(x_stuetz, y_stuetz, 'bx')
fig.savefig('interpolation.pdf')
```



7 Symbolische Mathematik

7.1 Brüche

Brüche mit sympy. Beispiel: $\frac{1}{3}$

```
from sympy import Rational
b = sympy.Rational(1,3)
```

Leider kann ein Rational-Ausdruck nicht als Argument an einen Rational-Ausdruck übergeben werden. Beispiel: $z = \frac{\sqrt{3}}{2}$

```
#Fehler:
z = sympy.Rational(\
  3**sympy.Rational(1,2),2)
```

Abhilfe: $z = \frac{\sqrt{3}}{2} = \frac{\sqrt{3}}{\sqrt{2^2}} = \sqrt{\frac{3}{2^2}}$

```
#funktionier:
z = sympy.Rational(\
  3,2**2)**sympy.Rational(1,2)
```

7.2 Terme rational machen

Ausdrücke in einen einzelnen Bruch umformen

```
import sympy
x = sympy.symbols('x')
f = (-2*x+3)/((x-1)**2+2**2)+(5)/(x-3)
print(sympy.ratsimp(f))
# ergibt:
# (3*x**2-x+16)/(x**3-5*x**2+11*x-15)
```

7.3 Faktoren eines Teilausdruckes ermitteln

Mit dem Befehl `.coeffs()` können Faktoren eines Teilausdrucks extrahiert werden. So kann beispielsweise aus dem Ausdruck $2\,a_0 - 10\,a_1$ der Faktor vor a_0 (also 2) wie folgt extrahiert werden:

```
import sympy as sp
a0, a1 = sp.symbols('a0␣a1')
print((2*a0 - 10*a1).coeff(a0))
# ergibt 2
```

7.4 Ableitungen

Ableitung von Funktionen berechnen

```
from sympy import *
t = symbols('t')
expr = Rational(1,2)*log((1+t)/(1-t))
pprint(diff(expr, t, 1))
# sympy.sqrt() statt math.sqrt()
pprint(diff(sqrt(2*t), t, 1))
```

7.5 Integrieren

Symbolisch ein bestimmtes Integral bilden

```
from sympy import *
init_printing(use_unicode=False, wrap_line=False)
del t
t = Symbol('t')
ak = integrate(2*pi*t*k+4,(t,0,1))
print(ak) # ergibt 4 + 15*pi
```

Symbolisch ein unbestimmtes Integral bilden:

```
from sympy import *
init_printing(use_unicode=False, wrap_line=False)
del t
t = Symbol('t')
ak = integrate(2*pi*t*k+4,t)
print(ak)  # ergibt 15*pi*t**2 + 4*t
```

Sonderfälle und Fallunterscheidungen im Ergebnis unterdrücken, indem Konstanten als Positive relle Zahlen definiert werden:

```
from sympy import *
w, tau, t, tp = symbols('w tau t tp', positive=True, real = True)
integrate(exp(-tau*tp)*cos(w*tp),(tp,0,t))
```

7.6 Ausmultiplizieren

Ausmultiplizieren (engl. expand) von Ausdrücken

```
from sympy import *
x = symbols('x')
f = ((x+2)*(x+2)*(x-2)*(x-3))
pprint(expand(f))
```

7.7 Ersetzungen

Ersetzen von Teilausdrücken. In diesem Beispiel wird in $-6\,x^3+16\,x^2-84\,x+64$ der Ausdruck x^2 ersetzt durch $-2\,x-17$:

```
from sympy import *
x = symbols('x')
g = -6*x**3 + 16*x**2 - 84*x+64
print(g.subs(x**2,-2*x-17))
```

Eine Schwierigkeit ist, in x^3 den Ausdruck x^2 durch beispielsweise w zu ersetzen zu $x^2\,w$. Abhilfe:

```
x = symbols('x')
x = symbols('w')
g = -6*x**3 + 16*x**2 - 84*x+64
g = g.replace(x**3,UnevaluatedExpr(x**2)*x)
print(r2.subs(x**2,w).doit())
```

7.8 Numerische Auswertung

Ein symbolischer Ausdruck kann mit `evalf` in einen Zahlenwert umgewandelt werden:

```
import sympy
y = sympy.sqrt(2)
print(y) # sqrt(2)
print(y.evalf()) # 1.4142
```

7.9 Evaluation verhindern

Gelegentlich will man eine Auswertung eines Ausdrucks verhindern:

```
from sympy import *
x = symbols('x')
print(x+x) # ergibt 2*x
```

Abhilfe:

```
from sympy import *
x = symbols('x')
print(UnevaluatedExpr(x)+x) # x+x
```

Der Parameter `evaluate=False` verhindert die Auswertung bis zum nächsten Aufruf

```
from sympy import *
w = symbols('w')
z = Add(w,w,w, evaluate=False)
z = Add(z,w, evaluate = False)
print(z) # w+w+w+w
z = z
print(z+1) # z wird evaluiert: 4*w+1
```

Permanente Auswertung wird mit `UnevaluatedExpr()` verhindert und mit `doit()` aufgehoben:

```
z = UnevaluatedExpr(w)+w
print(z)
print(z+1) # z wird nicht evaluiert
print((z+1).doit()) # z wird evaluiert
```

7.10 Vereinfachungen

Vereinfachungen können mit simplify erfolgen:

```
from sympy import *
x = symbols('x')
print(simplify(x*(x**3*x**2)))
# ergibt: x**6
```

7.11 Automatische Vereinfachung

Summen werden automatisch gebildet und Ausdrücke vereinfacht. In diesem Beispiel wird in $4\,x + 5\,x^2 - 2\,x^2 + 2\,x^2 - 4$ alle gleiche Potenzen von x summiert und ein Polynom gebildet:

```
from sympy import *
x = symbols('x')
g = 4*x+5*x**2-2*x**2+2*x**2-4
print(g)
#ergibt: 5*x**2 + 4*x - 4
```

7.12 Ausklammern

Ausdrücke können wie folgt ausgeklammert werden:

```
from sympy import *
t = symbols('t')
expr = sin(t)*t+cos(t)*t+\
  exp(2*t)*t**2+exp(3*t+5)*t**2
pprint(collect(expr, [t, t**2]))
```

Ergibt: $t^2 \left(e^{2t} + e^{3t+5}\right) + t \cdot (\sin(t) + \cos(t))$

7.13 Vereinfachung von Rationalen Funktionen (Polynombrüche)

Rationale Funktionen sind Brüche mit Polynomen im Zähler und Polynomen im Nenner. Diese können mit ratsimp vereinfacht werden.

```
from sympy import *
t = symbols('t')
#expr=Rational(1,2)*log((1+t)/(1-t))
#pprint(diff(expr, t, 1))
expr = Poly(1,t)/Poly(1-t**2,t)
pprint(ratsimp(diff(expr, t, 7)))
```

7.14 Partialbruchzerlegung

Funktion zur Partialbruchzerlegung:

```
from sympy import *
s = symbols('s')
f = s/((s**2+7**2)*\
   ((s+4)**2+3**2)*(s+5))
pprint(apart(f))
```

7.15 Ausmultiplizieren von Partialbrüchen

Ausmulitplizieren der linken Seite der folgenden Gleichung ergibt die rechte:

$$\frac{5}{3}\,\frac{1}{s+1} - \frac{2}{s+2} + \frac{1}{3}\,\frac{1}{s-2} = \frac{3\,s-2}{s^3+s^2-4\,s-4}$$

```
from sympy import *
s = symbols('s')
expr = (Rational(5,3)*Poly(1,s)/Poly(s+1,s)
  -2*Poly(1,s)/Poly(s+2,s)
  + Rational(1,3)*Poly(1,s)/Poly(s-2,s))
pprint(ratsimp(expr))
```

7.16 Ausgabe in Formel-Schreibweise

Mit pprint (pretty-print) werden Formeln in Ascii–Art dargestellt:

```
from sympy import *
t = symbols('t')
expr = Poly(1,t)/Poly(1-t**2,t)
pprint(ratsimp(diff(expr, t, 1)))
```

Ergibt:

```
      2*t
-------------
 4       2
t  - 2*t  + 1
```

7.17 Lineare Gleichungssysteme exakt lösen

Häufig sind lineare Gleichungssysteme gegeben, bei denen als Koeffizienten oder Konstantenvektor als Brüche angegeben sind. Beispiel:

$$\begin{pmatrix} 1/3 & -1/2 & -1/5 \\ 1/4 & -1 & -1/4 \\ 1/2 & -1/3 & -1/6 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} 5/30 \\ 1/16 \\ 9/36 \end{pmatrix}$$

Die exakte Lösung für die Unbekannten sind i.d.R. Brüche und kann mit der Bibliothek sympy und fractions berechnet werden:

```
from sympy import *
from fractions import Fraction
def F(a,b):
    return(Fraction(a,b))
A, B, C = symbols('A␣B␣C')
system = Matrix(\
 ((F(1,3),F(-1,2),F(-1,5),F(5,30)),\
 (F(1,4),F(-1,1),F(-1,4),F(1,16)),\
 (F(1,2),F(-1,3),F(-1,6),F(9,36))))
erg=solve_linear_system(system,A,B,C)
print(erg)
```

Ergibt:

```
{A: 17/40, B: 1/5, C: -5/8}
```

7.18 Matrix symbolisch invertieren

Eine Matrix soll invertiert werden. Beispiel:

$$M = \begin{pmatrix} L_1 & 0 & L_3 \\ L_1 & L_2 & 0 \\ 0 & -L_2 & 0 \end{pmatrix}$$

Code:

```
from math import pi
from sympy import *
L1, L2, L3 = symbols('L1,␣L2,␣L3')
Mat = Matrix([[L1, 0,  L3],
              [L1, L2, 0],
```

```
            [0, -L2, 0]
            ])
Mat.inv()
```

Ergibt:

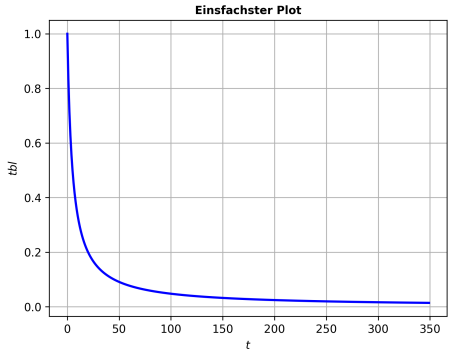
```
Matrix([
[   0,  1/L1,  1/L1],
[   0,     0, -1/L2],
[1/L3, -1/L3, -1/L3]])
```

8 Matplotlib

8.1 Einfachster Plot

```
import matplotlib.pyplot as plt # Erstellung von Grafiken
import numpy as np # Rechnen und Datenverarbeitung
import os

t = [k for k in range(0, round(70/0.2))]
tbl = [1/(1+k*0.2) for k in t]

plt.plot(t, tbl, color="blue", linewidth=2)
plt.title("Einsfachster Plot", fontsize=10, fontweight="bold")
plt.xlabel('$t$')
plt.ylabel('$tbl$')
plt.grid()
plt.savefig("EinfachsterPlot.png", dpi=300, bbox_inches='tight')
# bbox_inches='tight' Entfernt unnötigen Leerraum um das Diagramm
# dpi: Auflösung des Bildes in Punkten pro Zoll (300 für hohe Qualität)
```

Ergibt:



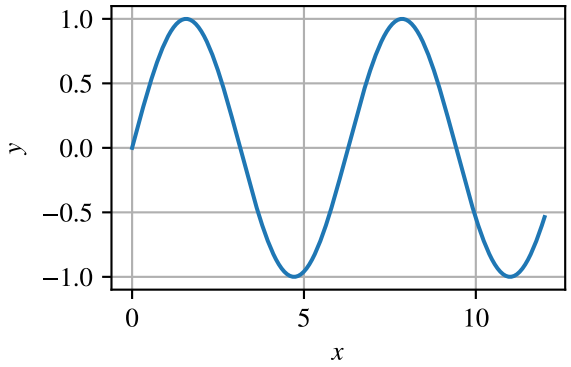
8.2 Schriftart Stix

In dem folgenden Code-Beispiel ist erklärt, wie in einer Matplotlib-Grafik die Schrifart Stix (https://www.stixfonts.org/) aktiviert ist. In der Windows-Zeichentabelle muss dazu die Schriftart „STIX Two Text" erscheinen.

```
import matplotlib.pyplot as plt
import numpy as np
plt.rcParams['font.family']=\
```

```
    'STIXGeneral'
plt.rcParams['mathtext.fontset']=\
    'stix'
x = np.linspace(0, 12, 150)
fig, ax = plt.subplots()
fig.set_size_inches(80/25.4,50/25.4)
g1, = ax.plot(x, np.sin(x))
plt.grid()
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.subplots_adjust(left=0.22,\
right=0.97, top=0.97, bottom=0.22)
plt.savefig('schriftartstix.pdf')
```

Ergibt:



8.3 Ausgabe verfügbarer Schriftarten

In dem folgenden Code-Beispiel werden die installieren Schriftarten angezeigt:

```
import matplotlib.font_manager
fpaths = matplotlib.font_manager.findSystemFonts()

for i in fpaths:
    f = matplotlib.font_manager.get_font(i)
    print(f.family_name)
```

Ergibt beispielsweise:

```
...
Lucida Bright
Calisto MT
Rockwell
Castellar
...
```

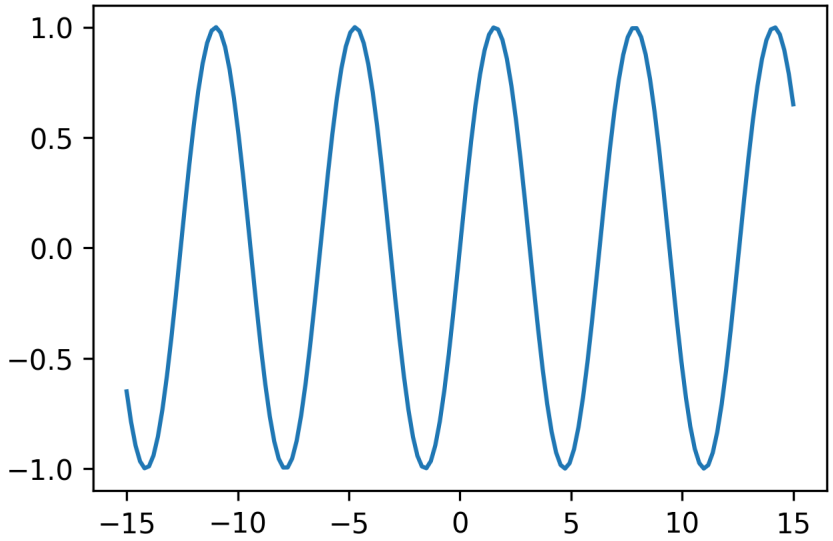
8.4 Auflösung von Rastergrafiken

Rastergrafiken können mit einer explizit angegebenen Anzahl pro Bildpunkten pro Zoll (dpi = dots per inch) exportiert werden. Beispiel für 300 Bildpunkte pro Zoll (Standard sind 100 dpi):

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-15, 15, 150)
fig, ax = plt.subplots()
```

```
fig.set_size_inches(120/25.4,80/25.4)
g1, = ax.plot(x,np.sin(x))
plt.savefig('bilddatei.png', dpi=300)
```

Ergibt:

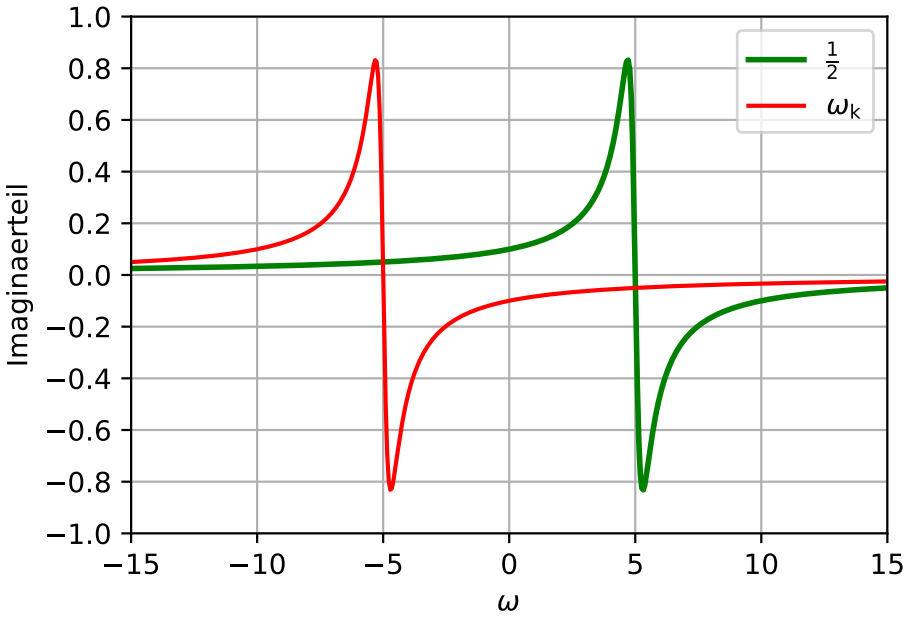


8.5 Universal-Plot

Das folgende Codebeispiel zeigt ein universell einsetzbares Diagramm.

```
import matplotlib.pyplot as plt
import numpy as np
a = 0.3
wc = 5
wk = np.linspace(-15, 15, 500)
A, B, C, D = -1/2, wc/2, -1/2, -wc/2
impt3 = (A*wk+B)/((wk-wc)**2 + a**2)
impt4 = (C*wk+D)/((wk+wc)**2 + a**2)
fig, ax = plt.subplots()
fig.set_size_inches(120/25.4,80/25.4)
g1, = ax.plot(wk, impt3, 'g-',\
  lw=2 , label=r'$\frac{1}{2}$')
g2, = ax.plot(wk, impt4, 'r-',\
  label=r'$\omega_{\mathrm{k}}$')
# fuer Legende: Komma nach g1 und g2
# nicht vergessen beim Plot-Befehl!
plt.legend(handles=[g1, g2])
plt.xticks(np.arange(-15,15+5,step=5))
plt.yticks(np.arange(-1,1+0.2,step=0.2))
ax.set_xlim([-15,15])
ax.set_ylim([-1,1])
plt.grid()
plt.xlabel('$\\omega$')
plt.ylabel('Imaginaerteil')
plt.subplots_adjust(left=0.17,\
  right=0.97, top=0.97, bottom=0.15)
plt.savefig('universalgrafik.pdf')
```

Ergibt:

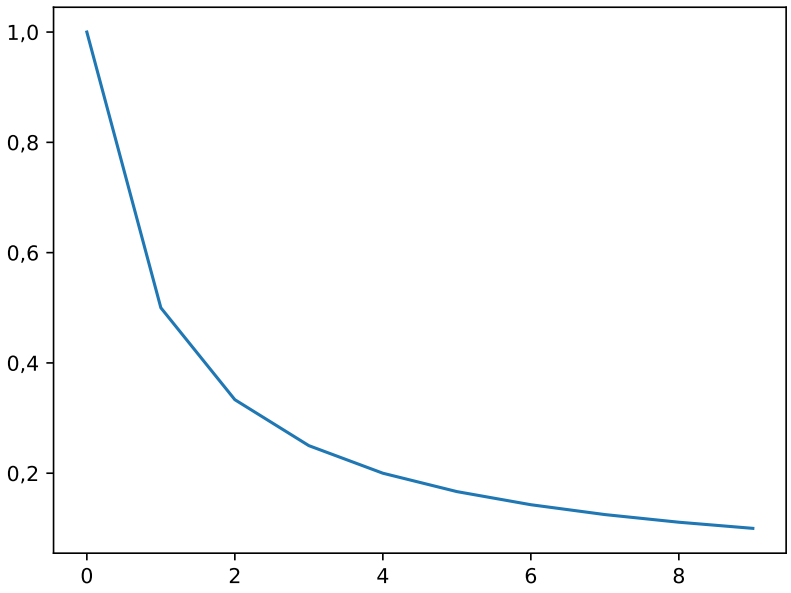


8.6 Kommas statt Punkte

In diesem Minimalbeispiel wird gezeigt, wie als Dezimaltrennzeichen das Komma verwendet werden kann

```
import matplotlib.pyplot as plt
import matplotlib.ticker as tkr

def func(x, pos): # formatter
  s1 = '{:1.1f}'.format(x)
  s2 = s1.replace('.',',')
  return s2
y_frmt = tkr.FuncFormatter(func)

t = [k for k in range(10)]
y = [1/(k+1) for k in t]
fig, ax = plt.subplots()
p1 = ax.plot(t, y)
ax.yaxis.set_major_formatter(y_frmt)
plt.savefig('kommastattpunkt.pdf')
```

Ergibt:

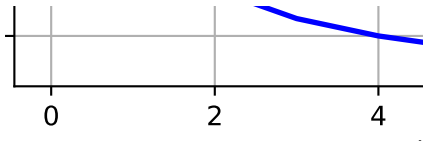


8.7 Achsenzahlen verschieben

Die Achsenzahlen (’ticks’) können von der Achse weg oder zur Achse und darüber hinaus mit dem Befehl `ax.xaxis.set_tick_params(pad=...)` verschoben werden.

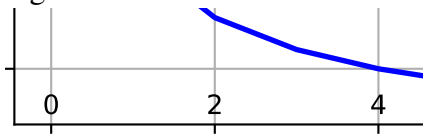
```
import matplotlib.pyplot as plt
import matplotlib.transforms

t = [k for k in range(10)]
y = [1/(k+1) for k in t]

fig, ax1 = plt.subplots()

# Erstes Bild ohne Achsenzahlen Verschiebeung
p1 = ax1.plot(t, y, color="blue", linewidth=2)
ax1.set_xlabel('$t$')
ax1.set_ylabel('$y$')
ax1.set_title("Achsenzahlen␣nicht␣verschoben", fontsize=10, fontweight="bold")
ax1.grid(True)


fig.set_size_inches(128/25.4 , 69/25.4) # Umwandlung von millimeter in Zoll
plt.tight_layout() # sorgt fuer schoene Abstaende
Breite_inches=2.2
Hoehe_inches=0.7
xmin_inches=0.6
ymin_inches=0.3
plt.savefig('AchsenzahlenVerschieben_unten.pdf',
            bbox_inches=matplotlib.transforms.Bbox([[xmin_inches,
    ymin_inches],[xmin_inches+Breite_inches,Hoehe_inches+ymin_inches]]))
#plt.show()
```

Ergibt:




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

t = [k for k in range(10)]
y = [1/(k+1) for k in t]

fig, ax1 = plt.subplots()

# Erstes Bild ohne Achsenzahlen Verschiebeung
p1 = ax1.plot(t, y, color="blue", linewidth=2)
ax1.set_xlabel('$t$')
ax1.set_ylabel('$y$')
ax1.xaxis.set_tick_params(pad=-15)
ax1.set_title("Achsenzahlen nicht verschoben", fontsize=10, fontweight="bold")
ax1.grid(True)


fig.set_size_inches(128/25.4 , 69/25.4) # Umwandlung von millimeter in Zoll
plt.tight_layout() # sorgt fuer schoene Abstaende
Breite_inches=2.2
Hoehe_inches=0.7
xmin_inches=0.6
ymin_inches=0.3
plt.savefig('Achsenzahlenverschieben_verschoben.pdf',
            bbox_inches=matplotlib.transforms.Bbox([[xmin_inches, ymin_inches],
            [xmin_inches+Breite_inches,Hoehe_inches+ymin_inches]]))
#plt.show()
```

Ergibt:



8.8 Positionen Hilfsgitterlinien

In diesem Minimalbeispiel wird gezeigt, wie die Positionen der Hilfsgitterlinien eingestellt werden können

```
import matplotlib.pyplot as plt
import numpy as np
t = np.linspace(0,12,100)
y = 4*t**2-3*t+3
fig, ax = plt.subplots()
p1 = ax.plot(t, y)
plt.xticks(np.arange(0,12+1,step=1))
plt.yticks(np.arange(0,601,step=50))
ax.grid()
plt.show()
```

8.9 Beschriftungspfeile

In der Grafik kann über verschiedene Wege eine Pfeil eingefügt werden. Eine gute Lösung ist:

```
plt.annotate("", xy=(-1,2.3),
 xytext=(0,2.3),\
 arrowprops=dict(arrowstyle="|-|",\
 linewidth=1.3, color='k',shrinkA=0,\
shrinkB=0, capstyle='round'))
```

Darin sind die ersten Koordinaten -1;2,3 und die zweiten Koordinaten 0;2.3. Für die Pfeilart gibt es, unter anderen, die Möglichkeiten „|-|", „-|>" und „<|-". Die Koordinaten werden in Daten-Koordinaten angegeben.

8.10 Beschriftungstext

In der Grafik kann Text als Beschriftung eingefügt werden.

```
plt.text(0.5,2.4,"$T_0$", ha='center')
```

Darin ist 0,5 die x-Koordinate, 2,4 die y-Koordinate. Der Text ist als Latex-Formel gesetzt. Die horizontale Ausrichtung ist zentriert.

8.11 Größe der Abbildung

Die Anzeigegröße von matplotlib-plots in Rastergrafiken kann wie folgt vergrößert werden:

```
fig, ax = plt.subplots()
...
fig.set_dpi(150)
```

je größer die dpi-zahl ist, desto größer wird die Abbildung angezeigt. Die dpi-Zahl gibt an, wieviele Pixel pro Zoll ausgegeben werden.

8.12 Grenzen der Achsen

Die angezeigten Zahlenwerte der Achsen können folgendermaßen geändert werden:

```
ax.set_xlim([xmin, xmax])
ax.set_ylim([ymin, ymax])
```

8.13 Plotstile

Linien können unterschiedlich gestaltet sein:

```
import matplotlib.pyplot as mpl
import numpy as np
fig, ax = plt.subplots()
x = np.linspace(0,10,100)
ax.plot(x,np.sin(x),'g-')
# 'g-' ist die Linien-Stildefinitino
```

Bei den Farben gibt es : `'b'`=blau, `'g'`=grün, `'r'`=rot, `'c'`=cyan, `'m'`=magenta, `'y'`=gelb, `'k'`=schwarz, `'w'`=weiß, `'tab:orange'`, `'tab:purple'`, `'tab:brown'`, `'tab:gray'`, `'tab:pink'`,

Bei den Linienarten gibt es: `'-'`=durchgängige Linie, `'--'`=Gestrichelte Linie, `'-.'`=Punkt-Strich-Linie, `':'`=Gepunktete Linie

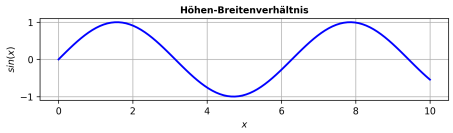
Weitere Farben: https://matplotlib.org/3.1.1/gallery/color/named_colors.html#sphx-glr-gallery-color-named-colors-py

8.14 Höhen-Breitenverhältnis

Gelegentlich müssen Achsen gleich skaliert sein. Dies ist beispielsweise bei Ortskurven wichtig. Eine Einheit in der horizontalen Richtung muss einer Einheit in vertikaler Richtung entsprechen. Eine Gerade mit der Steigung 1 wird dann einen Winkel von 45 Grad zur horizontalen Achse einnehmen.

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

x = np.linspace(0, 10, 100)
fig, ax = plt.subplots()
ax.plot(x, np.sin(x), color="blue", linewidth=2)
ax.set_aspect('equal') # Achsen gleich skaliert
fig.set_size_inches(200/25.4 , 150/25.4)

plt.xlabel('$x$')
plt.ylabel('$sin(x)$')
plt.title("Höhen-Breitenverhältnis", fontsize=10, fontweight="bold")
plt.grid()
plt.savefig("Hoehen-Breitenverhaeltnis.png", dpi=300, bbox_inches='tight')
plt.show()
```

Ergibt:



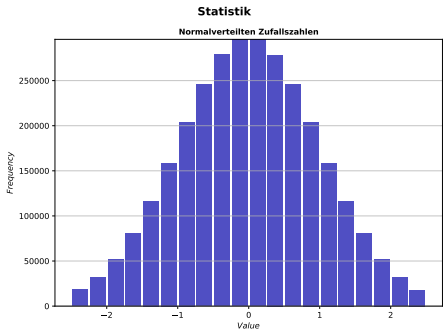
8.15 Statistik

Einfaches Statistik-Beispiel:

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

# Gemeinsame Zufallsdaten
Nx = 3000000
x = np.random.randn(Nx)

# Figur mit zwei Diagrammen (nebeneinander)
fig, ax = plt.subplots()
fig.set_size_inches(200/25.4 , 150/25.4) # Umwandlung von millimeter in Zoll

# 22.16 Statistik
n, bins, patches = ax.hist(
    x=x, bins=np.arange(-2.5, 2.75, 0.25),
    color='#0504aa', alpha=0.7, rwidth=0.85
)
ax.grid(axis='y', alpha=0.75)
ax.set_xlabel('$Value$')
ax.set_ylabel('$Frequency$')
ax.set_title('Normalverteilten␣Zufallszahlen')
```

```
ax.title.set_fontsize(10)
ax.title.set_fontweight('bold')

maxfreq = n.max()
ax.set_ylim(ymax=np.ceil(maxfreq / 10)*10 if maxfreq % 10 else maxfreq + 10)

fig.suptitle("Statistik", fontsize=14, fontweight='bold')
plt.tight_layout()
plt.savefig("Statistik1.pdf")
plt.show()
```

Ergibt:

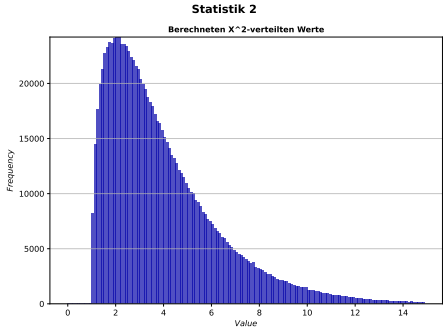


8.16 Statistik 2

Beispiel zum Thema Statistik.

```

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

Nx = 3000000
x = np.random.randn(Nx)

fig, ax2 = plt.subplots()
fig.set_size_inches(200/25.4 , 150/25.4) # Umwandlung von millimeter in Zoll
grosse = 3
chiv = np.ones(Nx//grosse)
for k in range(Nx//grosse):
    for m in range(grosse):
        chiv[k] = chiv[k] + x[k*grosse + m]**2

n2, bins2, patches2 = ax2.hist(
    x=chiv, bins=np.arange(0, 15, 0.1),
    color='#0504aa', alpha=0.7, rwidth=1.0
)
ax2.grid(axis='y', alpha=0.75)
ax2.set_xlabel('$Value$')
ax2.set_ylabel('$Frequency$')
ax2.set_title('Berechneten␣X^2-verteilten␣Werte')
ax2.title.set_fontsize(10)
ax2.title.set_fontweight('bold')
maxf = n2.max()
ax2.set_ylim(ymax=np.ceil(maxf / 10)*10 if maxf % 10 else maxf + 10)
fig.suptitle("Statistik␣2", fontsize=14, fontweight='bold')
plt.tight_layout()
plt.savefig("Statistik_b.pdf")
plt.show()
```

Ergibt:

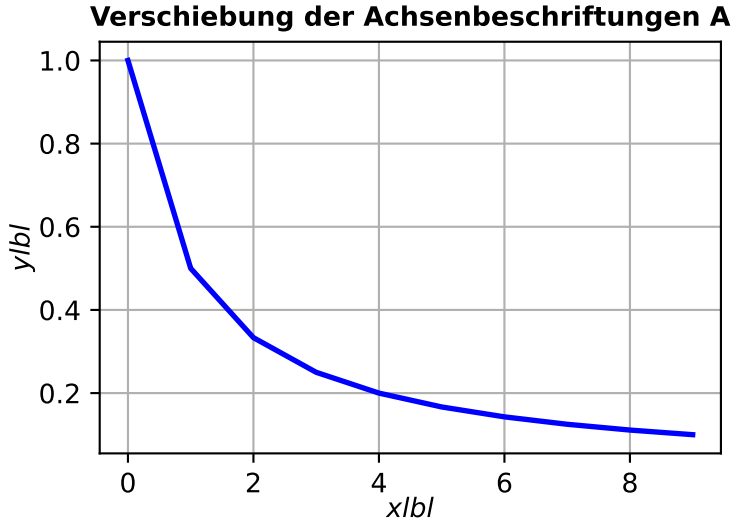


8.17 Verschiebung der Achsenbeschriftungen A

Die Achsenbeschriftungen können mit dem Parameter labelpad von den Achsen weg verschoben werden. Negative Werte für labelpad führen zu einer Verschiebung zu den Achsen hin.

```
import matplotlib.pyplot as plt
import os

t = [k for k in range(10)]
y = [1/(k+1) for k in t]
fig, ax = plt.subplots() # 1 Reihe, Zwei Spalten

# Verschiebung der Achsenbeschriftungen A mit labelpad
p1 = ax.plot(t, y, color="blue", linewidth=2)
ax.set_xlabel('$xlbl$', labelpad=0)
ax.set_ylabel('$ylbl$', labelpad=0)
ax.set_title("Verschiebung der Achsenbeschriftungen A", fontsize=10, fontweight="bold")
ax.grid(True)
fig.set_size_inches(102.4/25.4 , 76.8/25.4) # Umwandlung von millimeter in Zoll
plt.tight_layout() # sorgt fuer schoene Abstaende
plt.savefig("Verschiebung_der_Achsenbeschriftung_A.pdf")
```

Ergibt:

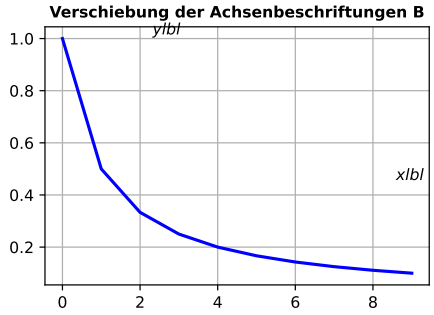


8.18 Verschiebung der Achsenbeschriftungen B

Die Achsenbeschriftungen können auf bestimmte Positionen verschoben werden. Die Koordinatenangaben erfolgen in relativen Koordinaten zur Zeichenfläche (rechts = 1,0, links = 0, unten = 0, oben = 1,0). Weiterhin kann die Drehung und Ausrichtung eingestellt werden.

```
import matplotlib.pyplot as plt
import os

t = [k for k in range(10)]
y = [1/(k+1) for k in t]
fig, ax = plt.subplots() # 1 Reihe, Zwei Spalten

t2 = [k for k in range(10)]
y2 = [1/(k+1) for k in t]
p2 = ax.plot(t2, y2, color="blue", linewidth=2)
ax.set_xlabel('$xlbl$')
  # rotation = drehung in Grad, ha = horizontal aligment (Ausrichtung)
ax.set_ylabel('$ylbl$', rotation=0, ha='left')
ax.xaxis.set_label_coords(0.95, 0.45)
ax.yaxis.set_label_coords(0.28, 0.95)
ax.set_title("Verschiebung der Achsenbeschriftungen B", fontsize=10, fontweight="bold")
ax.grid(True)

fig.set_size_inches(102.4/25.4 , 76.8/25.4) # Umwandlung von millimeter in Zoll
plt.tight_layout() # sorgt fuer schoene Abstaende
plt.savefig("Verschiebung_der_Achsenbeschriftungen_B.pdf")
```

Ergibt:



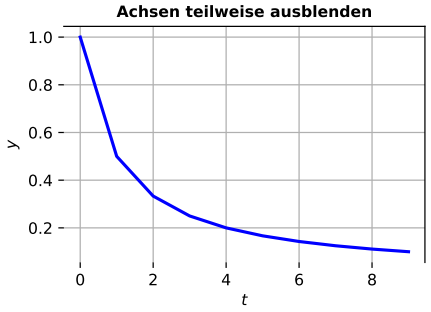
8.19 Achsen teilweise oder vollständig ausblenden

Die Anzeige der Achsen kann teilweise oder vollständig ausgeschaltet werden:

```
import matplotlib.pyplot as plt
import os
t = [k for k in range(10)]
y = [1/(k+1) for k in t]
fig1, ax1 = plt.subplots()
p1 = ax1.plot(t, y, color="blue", linewidth=2)
ax1.set_xlabel('$t$')
ax1.set_ylabel('$y$')
ax1.set_title("Achsen teilweise ausblenden", fontsize=10, fontweight="bold")

ax1.spines['bottom'].set_color('none')
ax1.spines['left'].set_color('none')
ax1.grid(True)
fig1.set_size_inches(102.4/25.4 , 76.8/25.4)
fig1.tight_layout()
```

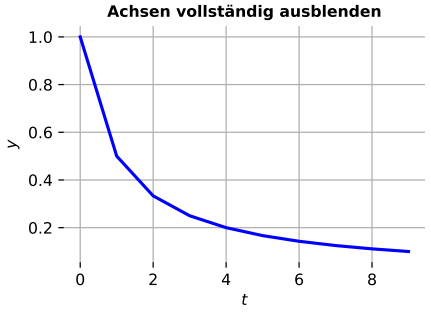
```
fig1.savefig("AchsenTeilweiseOderVollstaendigAusblenden_A.pdf")
```

Ergibt:




```
import matplotlib.pyplot as plt
import os
t = [k for k in range(10)]
y = [1/(k+1) for k in t]
fig1, ax1 = plt.subplots()
fig2, ax2 = plt.subplots()
p2 = ax2.plot(t, y, color="blue", linewidth=2)
ax2.set_xlabel('$t$')
ax2.set_ylabel('$y$')
ax2.set_title("Achsen vollständig ausblenden", fontsize=10, fontweight="bold")

ax2.spines['top'].set_color('none')
ax2.spines['right'].set_color('none')
ax2.spines['bottom'].set_color('none')
ax2.spines['left'].set_color('none')
ax2.grid(True)
fig2.set_size_inches(102.4/25.4 , 76.8/25.4)
fig2.tight_layout()
fig2.savefig("AchsenTeilweiseOderVollstaendigAusblenden_B.pdf")
```

Ergibt:



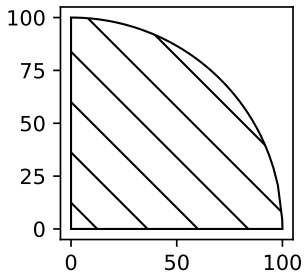
8.20 Schraffierte Flächen

Flächen können Schraffiert dargestellt werden. Als Optionen für hatch gibt es: \\, \, /, |, -, +, x, o, O, ., *. Die Schraffurfläche kann mit mehrfachen „\\” verdichtet werden. Beispiel: `hatch='\\\\\\'`

```
import matplotlib.pyplot as plt
import numpy as np
R = 100
```

```
xo = np.linspace(0, 0.999*R, 50)
x = np.append([R,0], xo)
y = np.append([0,0], np.sqrt(R**2-xo**2))
fig, ax = plt.subplots()
fig.set_size_inches(64/25.4, 48/25.4)
ax.fill(x,y, fill=False, hatch='\\')
ax.set_aspect('equal')
plt.subplots_adjust(left=0.12,\
  right=0.99, top=0.97, bottom=0.15)
plt.title("Schraffierte␣Flächen", fontsize=10, fontweight="bold")
plt.savefig("SchraffierteFlaechen.pdf")
plt.show()
```

Ergibt:

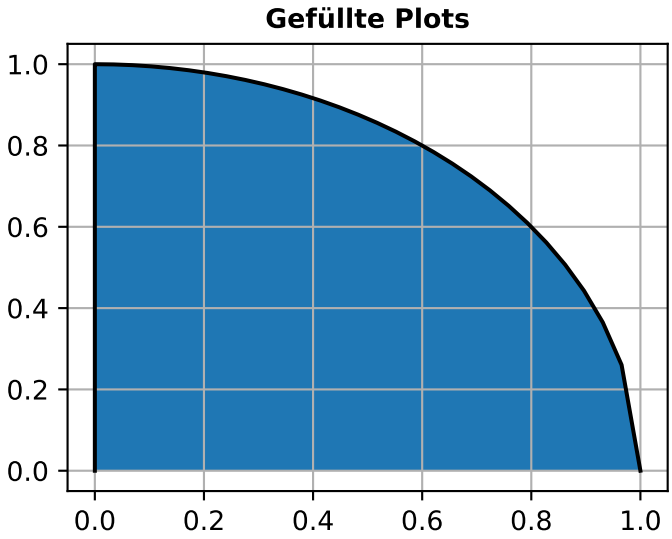


8.21 Gefüllte Plots

Mit dem Befehl fill statt plot können Flächen unter Kurven ausgefüllt werden:

```
import matplotlib.pyplot as plt
import numpy as np
fig, ax = plt.subplots()
xorig = np.linspace(0, 1 ,30)
x = np.append([0], xorig)
y = np.append([0], np.sqrt(1-xorig**2))
ax.fill(x, y)
ax.plot(x, y, 'k-')
plt.grid()
plt.title("Gefüllte␣Plots", fontsize=10, fontweight="bold")
fig.set_size_inches(102.4/25.4 , 76.8/25.4)
plt.savefig('GefuelltePlots.pdf')
plt.show()
```

Ergibt:

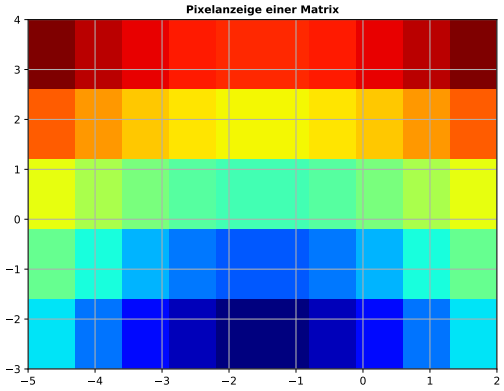


8.22 Pixelanzeige einer Matrix

Werte einer Matrix können als gerasterte Fläche dargestellt werden. Die Angabe `extent=[xmin, xmax, ymin, ymax]` gibt die Koordinatengrenzen der gerasterten Fläche an. Der Wert der einzelenen Matrixelemente wird in eine Farbe umgerechnet. Die Umrechnungsformel wird mit dem Parameter `cmap` angegeben. Einige mögliche sind: `'jet'`, `'gist_earth'` oder `'gnuplot'`

```
import matplotlib.pyplot as plt
import numpy as np
xv = np.linspace(-2, 2, 10)
yv = np.linspace(4, 1, 5)
Mx, My = np.meshgrid(xv, yv)
Mz = np.sqrt(Mx**2+My**2)
fig, ax = plt.subplots()
fig.set_size_inches(200/25.4, 150/25.4)
ax.imshow(Mz, extent = [-5, 2, -3, 4], aspect = 'auto',
          cmap = 'jet', interpolation = 'none')
ax.grid(lw = 1)
ax.set_title('Pixelanzeige␣einer␣Matrix')
ax.title.set_fontsize(10)
ax.title.set_fontweight("bold")
plt.savefig('Pixelanzeige_einer_Matrix.pdf')
plt.show()
```

Ergibt:

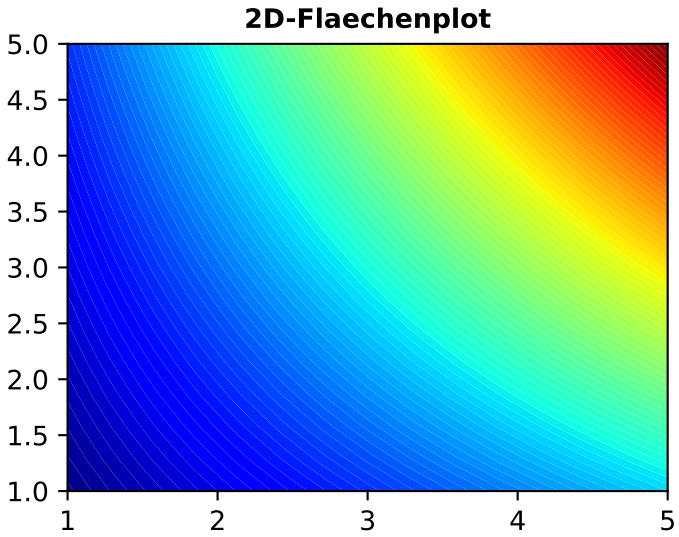


8.23 2D-Flächenplot

2D-Werte über eine Fläche Plotten, wobei den Koordinaten Funktionswerte zugeordnet werden und diese Funktionswerte in eine Farbe übersetzt wird.

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

phi00 = 1
phi01 = 2
phi10 = 3
phi11 = 7
x0 = 1
x1 = 5
y0 = 1
y1 = 5

def phixy(xp, yp):
    retval = 1/((y1-y0)*(x1-x0)) * (((x1-xp)*phi00 + (xp-x0)*phi10)
    * (y1-yp) + ((x1-xp)*phi01 + (xp-x0)*phi11) * (yp-y0))
    return(retval)

xv = np.linspace(x0, x1 ,15)
yv = np.linspace(y0, y1, 15)
Mx, My = np.meshgrid(xv, yv)
phi = phixy(Mx, My)
fig, ax = plt.subplots()
fig.set_size_inches(102.4/25.4, 76.8/25.4)
ax.set_title('2D-Flaechenplot')
ax.title.set_fontsize(10)
ax.title.set_fontweight("bold")
surf = ax.contourf(Mx, My, phi, 100, cmap=cm.jet)
plt.savefig('2D-Flaechenplot.pdf')
plt.show()
```

Ergibt:



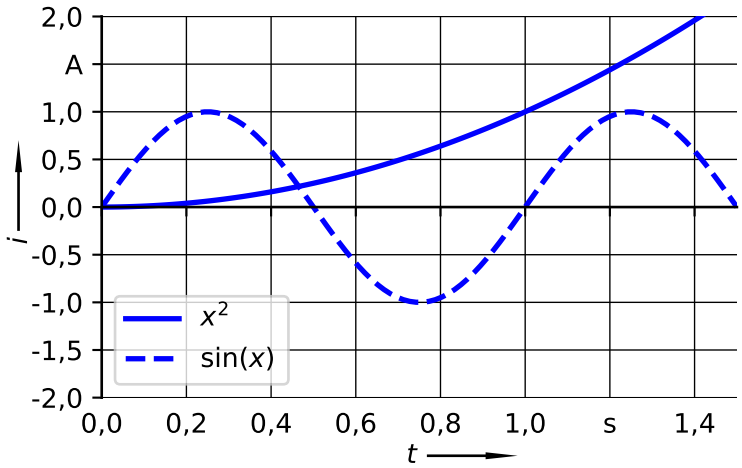
8.24 Plot nach DIN

Das nachfolgende Programm erzeugt einen Funktionsplot nach DIN-Norm:

```
import numpy as np
import matplotlib.pyplot as plt
import math
from matplotlib import rc
rc('text', usetex=False)
rc('legend',**{'fontsize':10})
t0 = 0.0
t1 = 1.5
steps = 100
xvals = np.linspace(t0,t1,steps)
yvals1 = xvals**2
yvals2 = np.sin(xvals*2*math.pi)
fig = plt.figure()
fig.set_size_inches([126/25.4, 80/25.4])
#Hoehe, Breite und Position des Plots
rect = 0.14, 0.17, 0.84, 0.80
ax1 = fig.add_axes(rect, frameon=True)
ax1.set_ylim([-2,2])
ax1.set_xlim([t0,t1])
stromplot = ax1.plot(xvals, yvals1, label="$x^2$",c='b',lw=0.7/0.3527)
sinusplot = ax1.plot(xvals, yvals2, label="$\\sin(x)$", color = 'b',
                     ls='--', lw=0.7/0.3527)
ax1.set_xlabel('$t$')
ax1.xaxis.set_label_coords(0.55, 0.07, transform=fig.transFigure)
ax1.set_ylabel('$i$')
ax1.yaxis.set_label_coords(0.05, 0.5, transform=fig.transFigure)
legend=ax1.legend( loc='lower left',shadow=False)
xls = 0.2
xachseneinheit_str = 's'
x=[ax1.get_xlim()[0]+k*xls for k in \
 range(math.floor((ax1.get_xlim()[1]-ax1.get_xlim()[0])*(1.0+1e-12)/xls)+1)]
lbls=[("%.1f"%d).replace('.',',') for d in x]
lbls[-2]=xachseneinheit_str
plt.xticks(x,lbls)
# Y-Achse:
yachseneinheit_str = 'A'
yls = 0.5
```

```
y=[ax1.get_ylim()[0]+k*yls for k in\
 range(math.floor((ax1.get_ylim()[1]-ax1.get_ylim()[0])*(1.0+1e-12)/yls)+1)]
lblsy=[("%.1f"%d).replace('.',',') for d in y]
lblsy[-2]=yachseneinheit_str
w35=0.35/0.3527
w18=0.18/0.3527
plt.yticks(y,lblsy)
ax1.spines['top'].set_color('none')
ax1.spines['right'].set_color('none')
ax1.spines['bottom'].set_position('zero')
ax1.spines['left'].set_linewidth(w35)
ax1.spines['bottom'].set_linewidth(w35)
ax1.xaxis.set_ticks_position('bottom')
ax1.grid(True,which='both',axis='x',c='k',ls='-',lw=w18)
ax1.grid(True,which='both',axis='y',c='k',ls='-',lw=w18)
fig.set_size_inches(100/25.4, 63/25.4)

dstaxtxpt=fig.get_size_inches()[1]*25.4*rect[3]/0.3527*(0-\
  ax1.get_ylim()[0])/(ax1.get_ylim()[1]-ax1.get_ylim()[0])+2
ax1.tick_params(axis='x',which='major', pad=dstaxtxpt)
# Achsenpfeil X:
b_mm=math.tan(7.5*math.pi/180)*20*0.35
arrowlen_mm = 12.0
deltax_frac = arrowlen_mm/(fig.get_size_inches()[0]*25.4)
arrowfrac = 10*0.35/arrowlen_mm
ybf = 0.048
xbf = 0.57
ax1.annotate('',xy=(xbf+deltax_frac,ybf),xycoords='figure␣fraction',\
  xytext=(xbf,ybf), textcoords='figure␣fraction',\
  arrowprops=dict(width=w35, facecolor='black', edgecolor= 'none', \
  headwidth=b_mm/0.3527))
# Achsenpfeil Y:
deltay_frac = arrowlen_mm / (fig.get_size_inches()[1]*25.4)
ybyf = 0.52
xbyf = 0.03
ax1.annotate('', xy=(xbyf,ybyf+deltay_frac), \
  xycoords='figure␣fraction', xytext=(xbyf,ybyf),\
  textcoords='figure␣fraction', arrowprops=dict(width=w35,\
  facecolor='black', edgecolor= 'none',  headwidth=b_mm/0.3527))

plt.title("Plot␣nach␣DIN", fontsize=10, fontweight="bold")
plt.savefig('Plot_nach_DIN.pdf')
plt.show()
```

Ergibt:



8.25 Gantt-Chart

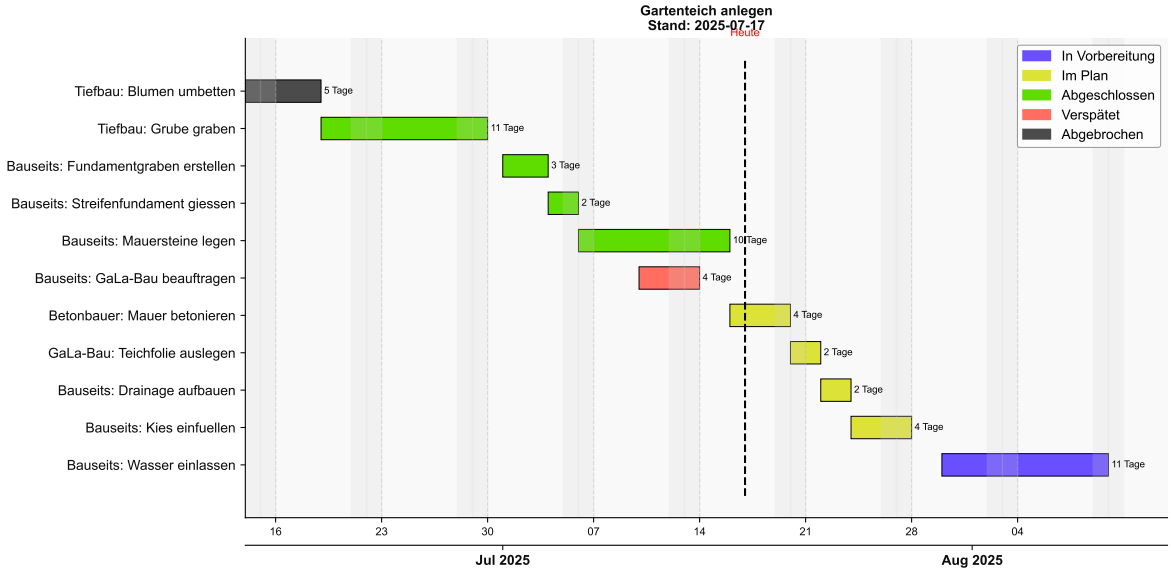
Ein Gantt-Chart ist ein Projektablaufplan. Beispiel:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib.patches as mpatches
import datetime
from matplotlib import rcParams

DATE_FORMAT = '%Y-%m-%d'
TODAY = '2025-07-17'

TITLE = "Gartenteich anlegen"
TITLE_SIZE = 10
TITLE_FONT_WEIGHT = "bold"

FONT_COLOR = "#0C0C0C"

X_LABEL = ""
Y_LABEL = ""
LABEL_SIZE = 8

DAY_FONT_SIZE = 8
MONTH_FONT_SIZE = 10
MONTH_FONT_WEIGHT = "bold"

BAR_COLOR = "#30C7DC"
TASKTYPE_BAR_COLORS = {
    "In Vorbereitung": "#694fff",
    "Im Plan": "#dce238",
    "Abgeschlossen": "#5fdc00",
    "Verspätet": "#ff6e61",
    "Abgebrochen": "#4b4b4b"
}

FONT_FAMILY = "sans-serif"
FONT_SANS_SERIF = ["Arial", "Roboto", "DejaVu Sans"]

rcParams['font.family'] = FONT_FAMILY
rcParams['font.sans-serif'] = FONT_SANS_SERIF
rcParams['axes.titlesize'] = TITLE_SIZE
rcParams['axes.labelsize'] = LABEL_SIZE


def build_week_ticks(start_date, end_date):
    mondays = pd.date_range(start=start_date, end=end_date, freq='W-MON')
    return mondays, [d.strftime('%d') for d in mondays]


def plot_gantt(tasks, output_path=None):
    if tasks.empty:
        print("No tasks to plot.")
        return

    start_date = tasks['start_date'].min()
    end_date = tasks['end_date'].max()

    tasks = tasks.sort_values(by=['start_date', 'task_group'], ascending=False)

    fig, ax = plt.subplots(figsize=(12, 6))

    # Sanfter Hintergrund für bessere Lesbarkeit
    ax.set_facecolor('#f9f9f9')
```

```
63      fig.patch.set_facecolor('#ffffff')
64
65
66      for _, task in tasks.iterrows():
67          duration = (task['end_date'] - task['start_date']).days
68          ax.barh(
69              task['task_group'] + ':␣' + task['task_description'],
70              width=duration,
71              height=0.6,
72              left=task['start_date'],
73              color=TASKTYPE_BAR_COLORS.get(task['task_status'], BAR_COLOR),
74              edgecolor='black', linewidth=0.5  # feine Balkenränder
75          )
76
77          # Dauer am Ende jedes Balkens anzeigen
78          ax.text(task['end_date'] + pd.Timedelta(days=0.2),
79                  task['task_group'] + ':␣' + task['task_description'],
80                  f"{duration}␣Tage",
81                  va='center', fontsize=7, color='black')
82
83      # vertikale Linie für das heutige Datum
84      today_dt = datetime.datetime.fromisoformat(TODAY)
85      ax.plot(mdates.date2num(today_dt) * np.array([1, 1]),
86               ax.get_ylim(), 'k--', label="Heute")
87
88      # Textlabel für die heutige Linie
89      ax.text(today_dt, ax.get_ylim()[1], 'Heute',
90              color='red', fontsize=8, ha='center', va='bottom')
91
92      # Wochen-Labels
93      week_positions, week_labels = build_week_ticks(start_date, end_date)
94
95      # Titel mit Untertitel
96      ax.set_title(f"{TITLE}\nStand:␣{TODAY}",
97                   fontsize=TITLE_SIZE,
98                   fontweight=TITLE_FONT_WEIGHT,
99                   color=FONT_COLOR)
100
101     ax.set_xlabel(X_LABEL, fontsize=LABEL_SIZE, color=FONT_COLOR)
102     ax.set_ylabel(Y_LABEL, fontsize=LABEL_SIZE, color=FONT_COLOR)
103     ax.tick_params(axis='both', colors=FONT_COLOR)
104     ax.set_xticks(week_positions)
105     ax.set_xticklabels(week_labels, fontsize=DAY_FONT_SIZE, color=FONT_COLOR)
106     ax.grid(axis='x', linestyle='--', alpha=0.4)
107
108     # Wochenenden farblich hinterlegen
109     for d in pd.date_range(start_date, end_date):
110         if d.weekday() >= 5:  # Samstag oder Sonntag
111             ax.axvspan(d, d + pd.Timedelta(days=1), color='lightgray', alpha=0.2)
112
113     # Untere Achse mit Monatsnamen
114     sec_ax = ax.secondary_xaxis('bottom')
115     sec_ax.xaxis.set_major_formatter(mdates.DateFormatter('%b␣%Y'))
116     sec_ax.xaxis.set_major_locator(mdates.MonthLocator())
117     sec_ax.tick_params(axis='x', labelsize=MONTH_FONT_SIZE, colors=FONT_COLOR)
118     sec_ax.spines['bottom'].set_position(('outward', 20))
119
120     for label in sec_ax.get_xticklabels():
121         label.set_fontsize(MONTH_FONT_SIZE)
122         label.set_weight(MONTH_FONT_WEIGHT)
123         label.set_color(FONT_COLOR)
124
125     for spine in ['top', 'right']:
126         ax.spines[spine].set_visible(False)
127         sec_ax.spines[spine].set_visible(False)
128
```

```
    # Legende + schöner Rahmen
    handles = []
    for task_statuss, color in TASKTYPE_BAR_COLORS.items():
        patch = mpatches.Patch(color=color, label=task_statuss)
        handles.append(patch)
    legend = ax.legend(handles=handles, loc='best', framealpha=0.8)
    legend.get_frame().set_linewidth(0.5)
    legend.get_frame().set_edgecolor('gray')

    plt.tight_layout()

tl = pd.DataFrame(columns=('task_group', 'task_description', 'task_status',
                           'start_date', 'end_date'))
tl.loc[0] = ["Tiefbau", "Blumen umbetten", "Abgebrochen","2025-06-14", "2025-06-19"]
tl.loc[len(tl)] = ["Tiefbau","Grube graben","Abgeschlossen","2025-06-19","2025-06-30"]
tl.loc[len(tl)] = ["Bauseits", "Fundamentgraben erstellen", "Abgeschlossen",
                   "2025-07-01", "2025-07-04"]
tl.loc[len(tl)] = ["Bauseits", "Streifenfundament giessen", "Abgeschlossen",
                   "2025-07-04", "2025-07-06"]
tl.loc[len(tl)] = ["Bauseits", "GaLa-Bau beauftragen", "Verspätet",
                   "2025-07-10", "2025-07-14"]
tl.loc[len(tl)] = ["Bauseits", "Mauersteine legen", "Abgeschlossen",
                   "2025-07-06", "2025-07-16"]
tl.loc[len(tl)] = ["Betonbauer", "Mauer betonieren", "Im Plan",
                   "2025-07-16", "2025-07-20"]
tl.loc[len(tl)] = ["GaLa-Bau", "Teichfolie auslegen", "Im Plan",
                   "2025-07-20", "2025-07-22"]
tl.loc[len(tl)] = ["Bauseits", "Drainage aufbauen", "Im Plan",
                   "2025-07-22", "2025-07-24"]
tl.loc[len(tl)] = ["Bauseits", "Kies einfuellen", "Im Plan",
                   "2025-07-24", "2025-07-28"]
tl.loc[len(tl)] = ["Bauseits", "Wasser einlassen", "In Vorbereitung",
                   "2025-07-30", "2025-08-10"]

tl['start_date'] = pd.to_datetime(tl['start_date'], format=DATE_FORMAT)
tl['end_date'] = pd.to_datetime(tl['end_date'], format=DATE_FORMAT)
tl.set_index(pd.DatetimeIndex(tl['start_date'].values), inplace=True)

plot_gantt(tl)
plt.savefig('Gantt-Chart.pdf')
plt.show()
```

Ergibt:



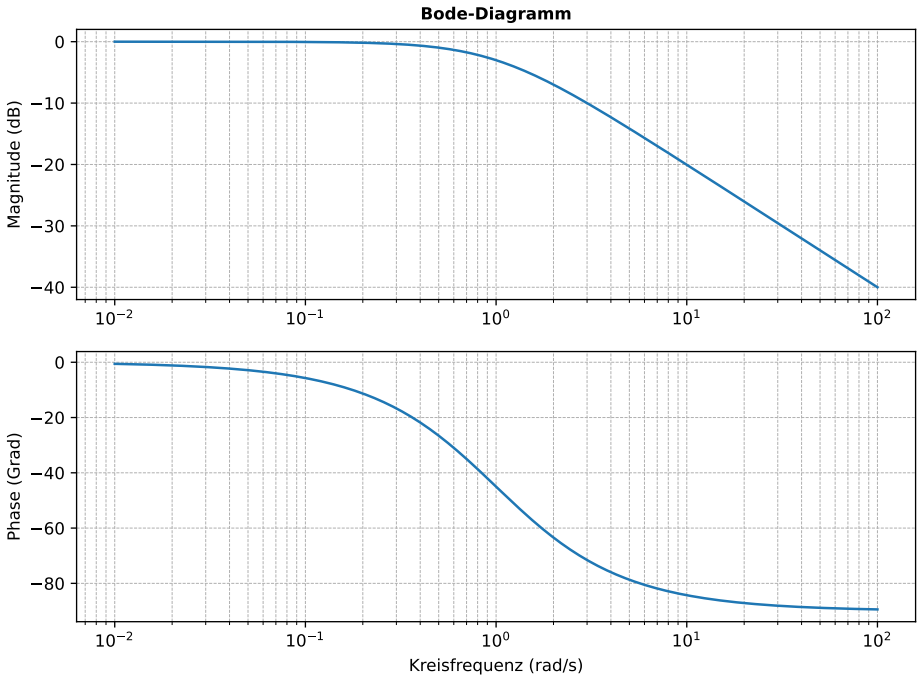
8.26 Bode - Diagramm

Das Bode-Diagramm kommt in der Regelungstechnik Vielfalt zum Einsatz und wird für die Analyse des Systemverhaltens, Stabilitätsuntersuchungen und Auslegung von Regelkreise verwendet. In dem folgenden Code-Beispiel wird angezeigt wie man das Bode-Diagramm einer Tiefpass 1er Ordnung mit der Bibliothek Spicy darstellt:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal

# Uebertragungsfunktion F(p) = 1/T1p+1 (Tiefpass 1.Ordnung)
num = [1]
den = [1, 1]
system = signal.TransferFunction(num, den)

# Frequenzbereich Logarithmisch waehlen
w = np.logspace(-2, 2, 500) # von 10^-2 bis 10^2 rad/s

# Frequenzgang berechnen
w, mag, phase = signal.bode(system, w=w)
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8,6))

# Betrag in dB
ax1.semilogx(w, mag)
ax1.set_title("Bode-Diagramm", fontsize=10, fontweight="bold")
ax1.set_ylabel('Magnitude␣(dB)')
ax1.grid(True, which="both", linestyle="--", linewidth=0.5)

# Phase in Grad
ax2.semilogx(w, phase)
ax2.set_xlabel('Kreisfrequenz␣(rad/s)')
ax2.set_ylabel('Phase␣(Grad)')
ax2.grid(True, which="both", linestyle="--", linewidth=0.5)

fig.set_size_inches(200/25.4, 150/25.4) # Umwandlung von millimeter in Zoll
plt.tight_layout() # sorgt fuer schoene Abstaende
```

```
plt.savefig('bodediagramm.pdf')
```

Ergibt:



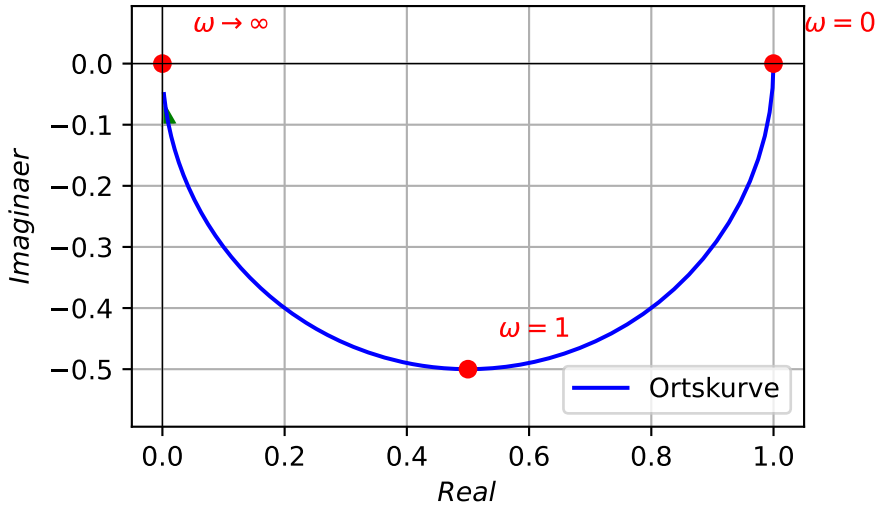
8.27 Ortskurve

Eine Ortskurve stellt dar, wie sich die komplexe Übertragungsfunktion eines Systems abhängig von der Frequenz in der komplexen Ebene verhält. In dem folgenden Code-Beispiel wird wie oben die Ortskurve für einen Tiefpass erste Ordnung dargestellt:

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

# Uebertragungsfunktion F(jw) = 1 / (1 + jwT) Tiefpass 1er Ordnung mit T = 1
def F(w):
    return 1 / (1 + 1j*w)
# Frequenzbereich w waehlen
w = np.linspace(0, 20, 500)
F_vals = F(w) # F_vals enthaelt alle komplexen Punkte der Ortskurve
plt.figure(figsize=(5,3))
plt.plot(F_vals.real, F_vals.imag, 'b-', label="Ortskurve")
# F_vals.real, H_vals.imag Real- und Imaginaerteil
# fuer w = 0; w = w0 und w = undendlich markieren. Mit w0 = 1
points = {
    r"$\omega=0$": F(0),
    r"$\omega=1$": F(1),
    r"$\omega\to\infty$": 0
}

for label, val in points.items():
    x, y = (val.real, val.imag) if val != 0 else (0, 0)
    plt.plot(x, y, 'ro')
    plt.text(x+0.05, y+0.05, label, color='red')

# Zeiger hinzufuegen, um die Richtung von w zu zeigen
```

```
mid = len(F_vals)//2
plt.arrow(F_vals.real[mid], F_vals.imag[mid],
          F_vals.real[mid+1]-F_vals.real[mid],
          F_vals.imag[mid+1]-F_vals.imag[mid],
          shape='full', head_width=0.02, color='green')

# Formatierung
plt.axhline(0, color='black', lw=0.5)
plt.axvline(0, color='black', lw=0.5)
plt.xlabel('$Real$')
plt.ylabel('$Imaginaer$')
plt.grid()
plt.subplots_adjust(left=0.2 ,right=0.9 ,top=0.93 ,bottom =0.2)
plt.axis("equal") # bei Ortskurven obligatorisch
plt.legend()
plt.savefig('Ortskurve.pdf')
plt.show()
```

Ergibt:



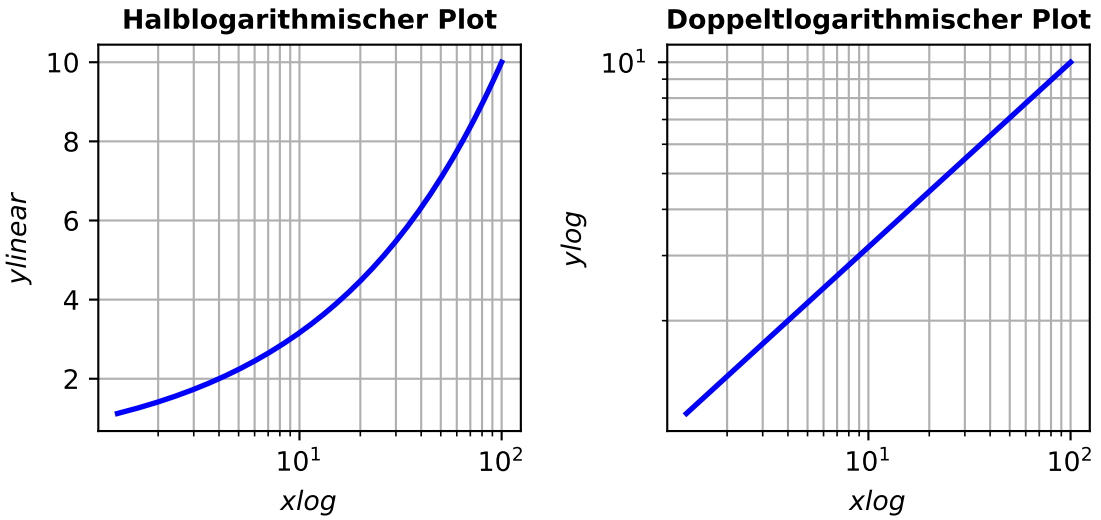
8.28 Halb und doppeltlogarithmische Plots

Beim halblogarithmische Plots ist nur eine Achse logarithmisch skaliert. Doppeltlogarithmische Plots hingegen weisen die beiden Achsen logarithmisch auf. Die Daten werden nach Auftragen auf einer logarithmischen Skala als gerade erscheinen. Beispiel-Code von halb und doppeltlogarithmische Plots

```
import matplotlib.pyplot as plt
import numpy as np
x = np.logspace(0.1, 2, 400) # x von 10^0.1 bis 10^2
y = np.sqrt(x)
fig, (ax1, ax2) = plt.subplots(1, 2) # eine Reihe, Zwei Spalten
# Halblogarithmische Plots(Semi-Log-Diagramm)
ax1.semilogx(x, y, color="blue", linewidth=2) # x-Achse logarithmisch
ax1.set_xlabel('$xlog$')
ax1.set_ylabel('$ylinear$')
ax1.set_title("Halblogarithmischer␣Plot", fontsize=10, fontweight="bold")
ax1.grid(True, which="both") # grid on both axis
# Doppeltlogarithmische Plots
```

```
ax2.loglog(x, y, color="blue", linewidth=2) # beide Achsen logarithmisch
ax2.set_xlabel('$xlog$')
ax2.set_ylabel('$ylog$')
ax2.set_title("Doppeltlogarithmischer Plot", fontsize=10, fontweight="bold")
ax2.grid(True, which="both")
fig.set_size_inches(152.4/25.4 , 76.8/25.4)
plt.tight_layout()
plt.savefig('HalbUndDoppeltLogarithmischePlots.pdf')
plt.show()
```

Ergibt:



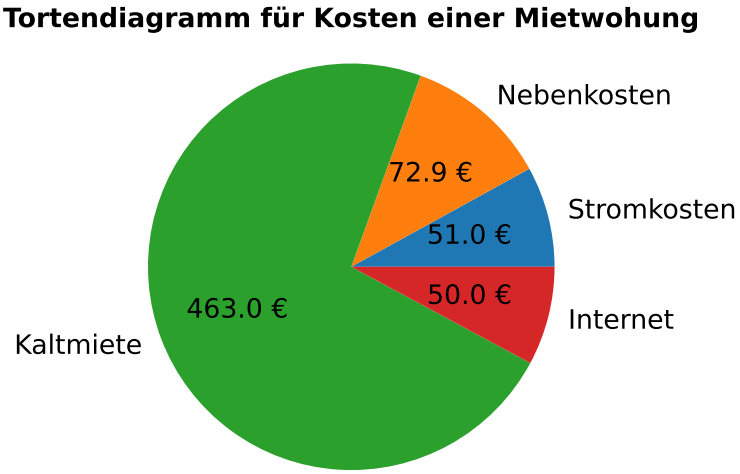
8.29 Tortendiagramm

Mit Tortendiagramm werden Daten als Sektoren eines Kreises dargestellt. Jede Sektorgröße repräsentiert den Anteil an der Gesamtmenge. Bei der Erstellung von Tortendiagrammen darf in wissenschaftlichen Arbeiten keine perspektivische Darstellung der Daten gezeigt werden. Die perspektivische Darstellung führt zu einer Verzerrung der Flächenverhältnisse. Ein Teilstück des Kreises mit dem Nenn-Anteil x hat bei einer verzerrten Darstellung nicht den den Flächenanteil x. Die Verwendung von perspektivischen Tortendiagrammen ist eine bewusste oder unbewusste Fälschung von Daten.

```
import matplotlib.pyplot as plt

labels = 'Stromkosten', 'Nebenkosten', 'Kaltmiete', 'Internet'
sizes = [51, 72.90, 462.95, 49.99]
# Das zweites Stueckt ist nicht/leicht herausgezogen
explode = (0, 0.1, 0, 0)
explode = (0, 0.0, 0, 0) # nicht herausgezogen
fig, ax = plt.subplots()
fig.set_size_inches(102.4/25.4 , 76.8/25.4)
ax.set_title("Tortendiagramm für Kosten einer Mietwohung")
ax.title.set_fontsize(10)
ax.title.set_fontweight("bold")
ax.pie(sizes, explode=explode, labels=labels,
       autopct= lambda p: f'{p*sum(sizes) / 100:.1f} €',
        shadow=False, startangle=0)
```

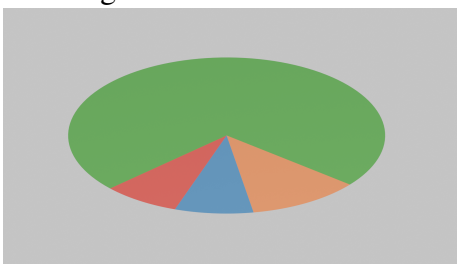
```
16 ax.axis('equal') # Der Kuchen wird als Kreis gezeichnet
17 plt.subplots_adjust(right=0.85)
18 plt.savefig('Tortendiagramm.pdf')
19 plt.savefig('Tortendiagramm.svg')
```

Ergibt:

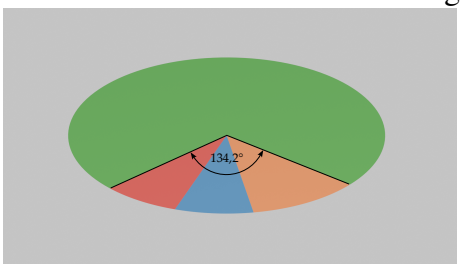


Die nachfolgenden Bilder zeigen jeweils die Daten der oberen Grafik.

Das erste Bild ist eine isometrische Verzerrung des Tortendiagramms. Das Höhen-Seitenverhältnis ist nicht mehr 1:1. Die Flächenanteilesverhältnisse, wie beispielsweise der rote Teil an der Gesamtfläche, bleibt in dieser Darstellung unverändert.

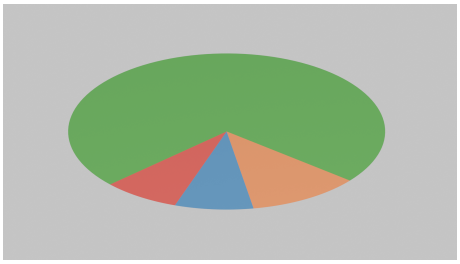


Allerdings werden die Winkel der Sektoren verfälscht. Die zeigt die folgende Abbildung:



Der Winkel für die Anteile “Nebenkosten”, “Strom” und “Internet” beträgt in der Kreisdarstellung 98,28° und in der isometrischen Verzerrung 134,2°. Dies könnte der Leserin oder dem Leser suggerieren, dass ein Anteil mehr Bedeutung hat, als in der Realität.

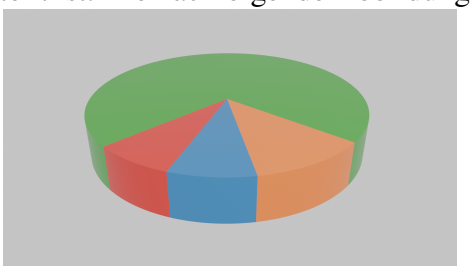
Die nächste Abbildung zeigt eine perspektivische Verzerrung des Bildes.



Die perspektivische Verzerrung ergibt sich, wenn das Tortendiagramm ausgedruckt werden und von einem schrägen Winkel fotografiert werden würde. Es ist optisch kaum von einer isometrischen Verzerrung zu unterscheiden. Auch in dieser Darstellung werden die Winkel verfälscht. Bei der perspektivischen Verzerrung werden zusätzlich Flächenanteile verfälscht. Einen Vergleich zwischen tatsächlichen Anteilen und den Farbflächen-Anteilen zeigt die folgende Tabelle:

	Kaltmiete (grün)	Nebenkosten (gelb)	Stromkosten (blau)	Internet (rot)
Real:	72,7 %	11,45 %	8,01 %	7,85 %
Flächenanteil:	59,03 %	16,81 %	12,81 %	11,35 %

Noch drastischer ist die Verzerrung, wenn das Tortendiagramm dreidimensional aufgepolstert und perspektivisch dargestellt ist. Die nachfolgende Abbildung zeigt ein Beispiel.



Die zu den ohnehin verfälschten Flächenanteilen kommen nun durch die dreidimensionale Darstellung Flächen hinzu, welche insbesondere die vorderen Sektoren größer aussehen lassen. Die Nachfolgende Tabelle zeigt einen Vergleich zwischen den realen Anteilen und den Flächenanteilen der einzelnen Farben.

	Kaltmiete (grün)	Nebenkosten (gelb)	Stromkosten (blau)	Internet (rot)
Real:	72,7 %	11,45 %	8,01 %	7,85 %
Flächenanteil in der Abbildung:	43,42 %	22,36 %	19,5 %	14,72 %

Die visuelle Wahrnehmung der Daten ist verzerrt. So ist beispielsweise bei den Stromkosten der blaue Bereich doppelt so groß wie der reale Anteil.

8.30 Box-Plot

Ein Boxplot-Diagramm stellt wichtige Kenngröße einer Stichprobe graphisch dar. Dies ist der Median als mittlerer Strich der Kiste. Das obere Ende wird durch das obere Quartil (25% der sortierten Messwerte oberhalb des Medians) definiert. Das untere Ende wird durch das untere Quartil (beinhaltet 25% der sortierten Messwerte unterhalb des Medians) definiert. Die Antennen (engl. “whiskers”) haben maximal die Länge, die 1,5 mal dem Interquartilsabstand ent-

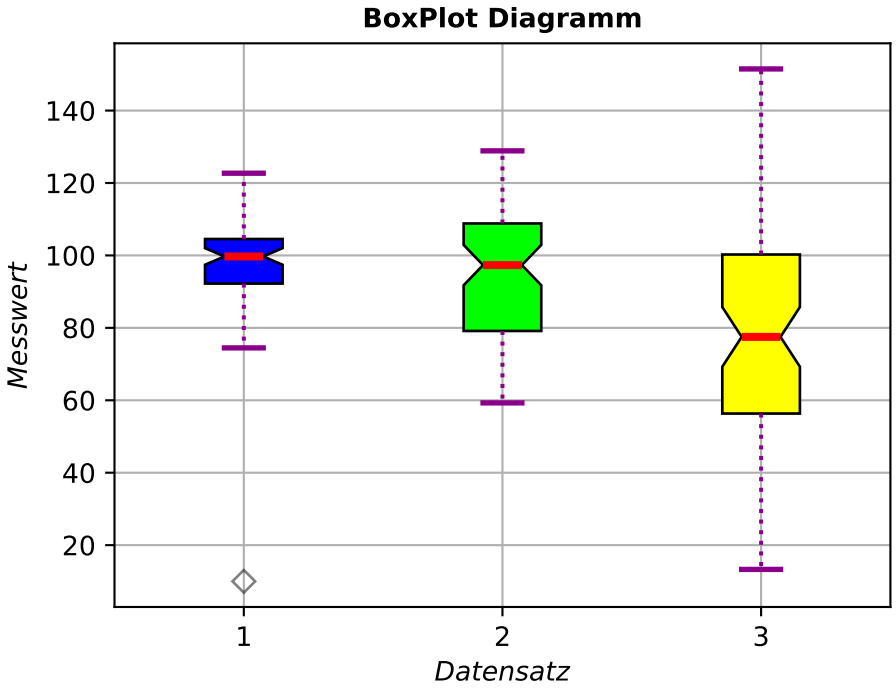
spricht. Das äußere Ende der Antennen liegt immer auf dem letzten Messwert in dem Bereich der Maximallänge. Die beiden Antennen sind also in der Regel ungleich lang. Messwerte, die außerhalb des Bereiches der Antennen liegen, werden als Ausreißer bezeichnet und als Datenpunkt markiert. Mit dem Befehl `boxplot` aus der Bibliothek matplotlib werden alle Ausreißer mit demselben Datenpunkt-Symbol gekennzeichnet. In der Literatur werden Varianten eines Boxplot-Diagramms beschrieben, bei dem Ausreißer mit einem anderen Datenpunkt-Symbol gekennzeichnet werden, die außerhalb des dreifachen Interquartilsabstand ober- und unterhalb der Box liegen.

Das folgende Code-Beispiel stellt ein Beispiel für ein BoxPlot Diagramm dar:

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

# Zufallsdaten erstellen
np.random.seed(0)
N = 70
data_1 = np.random.normal(100, 10, N)
data_2 = np.random.normal(90, 20, N)
data_3 = np.random.normal(80, 30, N)
data = [data_1, data_2, data_3]

fig, ax = plt.subplots()
fig.set_size_inches(132.4/25.4, 96.8/25.4)

# Boxplot erstellen
bp = ax.boxplot(data, patch_artist=True, notch=True, vert=0)

# Farben anpassen
colors = ['#0000FF', '#00FF00', '#FFFF00', '#FF00FF']
for patch, color in zip(bp['boxes'], colors):
    patch.set_facecolor(color)

# Linien anpassen
for whisker in bp['whiskers']:
    whisker.set(color='#8B008B', linewidth=1.5, linestyle=":")

for cap in bp['caps']:
    cap.set(color='#8B008B', linewidth=2)

for median in bp['medians']:
    median.set(color='red', linewidth=3)

for flier in bp['fliers']:
    flier.set(marker='D', color='#e7298a', alpha=0.5)

print("data_1 : ", data_1)
print(f"\n data_2 :  {data_2}")
print(f"\n data_3 :  {data_3}")

# Statistische Werte berechnen und anzeigen
for i, d in enumerate(data, start=1):
    Q1 = np.percentile(d, 25)
    Q3 = np.percentile(d, 75)
    IQR = Q3 - Q1

    lower_whisker = Q1 - 1.5 * IQR
    upper_whisker = Q3 + 1.5 * IQR

    outliers = d[(d < lower_whisker) | (d > upper_whisker)]

    print(f"\n Datensatz {i}: data_{i}")
    print(f"Q1 (25%): {Q1:.2f}")
```

```
    print(f"Q3 (75%): {Q3:.2f}")
    print(f"IQR: {IQR:.2f}")
    print(f"Unterer Whisker (Grenze): {lower_whisker:.2f}")
    print(f"Oberer Whisker (Grenze): {upper_whisker:.2f}")
    print(f"Ausreißer: {outliers if len(outliers) > 0 else 'Keine'}")

plt.xlabel('$Datensatz$')
plt.ylabel('$Messwert$')
plt.title("BoxPlot Diagramm", fontsize=10, fontweight="bold")
plt.grid()
plt.savefig('Box-Plot.pdf')
plt.show()
```

Ergibt:



Beachten Sie, dass Boxplots nicht einheitlich definiert sind. Insbesondere die unten und oberen Enden der Antennen, sind uneinheitlich definiert. Geben Sie immer an, was die einzelnen Diagrammelemente zeigen.

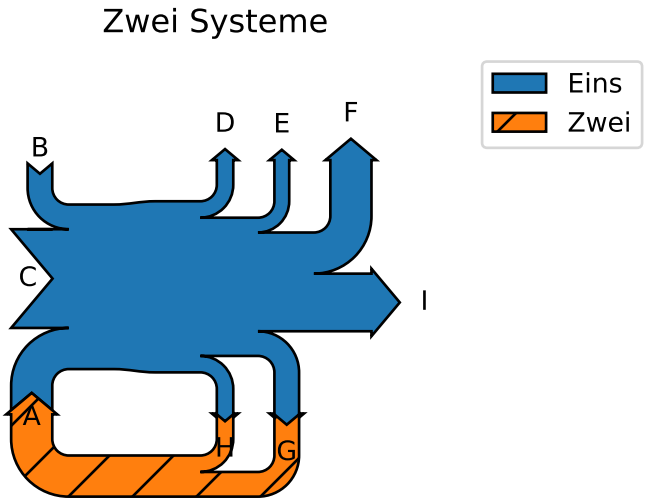
8.31 Sankey Diagramm

Ein Sankey Diagramm ist eine spezielle Form von Flussdiagramm, bei denen die Breite der Pfeile proportional zur dargestellten Flussmenge ist. Ein Sankey Diagramm wird typischerweise verwendet, um Energie-, Material- oder Kostenströme zwischen verschiedenen Prozessen anschaulich darzustellen. Für die Erstellung von Sankey Diagramm verwenden kann das Modul `matplotlib.sankey` verwendet werden.. In dem folgenden Beispiel-Code wird das Sankey Diagramm einer Halogenlampe, wobei ihre zugeführte Energie beziehungsweise Nutz- und Wärmeenergie gezeigt wird, dargestellt.

```
import matplotlib.pyplot as plt
from matplotlib.sankey import Sankey
```

```
3
4
5  fig = plt.figure()
6  ax = fig.add_subplot(1, 1, 1, xticks=[], yticks=[], title="Zwei␣Systeme")
7  sankey = Sankey(ax=ax, unit=None)
8  flows = [0.25, 0.15, 0.60, -0.10, -0.09, -0.25, -0.15, -0.10, -0.35]
9  # flows: Positiv: hinaus, Negativ, hinein
10 sankey.add(flows=flows, label='Eins',
11            orientations=[-1, 1, 0, 1, 1, 1, -1, -1, 0],
12            labels=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I'])
13 # orientation: -1: nach oben, 0:rechts, 1: nach unten
14 sankey.add(flows=[-0.25, 0.15, 0.1], label='Zwei', orientations=[-1, -1, -1], prior=0, connect=(0, 0))
15 diagrams = sankey.finish()
16 diagrams[-1].patch.set_hatch('/')
17 plt.legend()
18 plt.axis('off') # Rahmen entfernen
19 fig.set_size_inches(120/25.4,80/25.4)
20 plt.subplots_adjust(left=0.0,\
21   right=0.97, top=0.85, bottom=0.0)
22 fig.savefig('sankeydiag.pdf')
```

Ergibt:

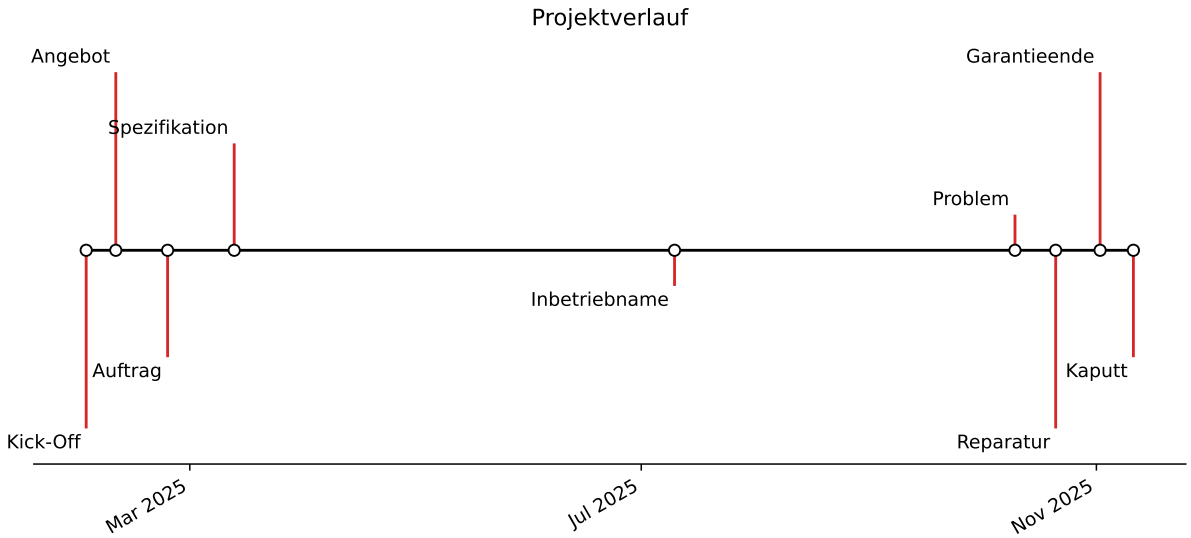


8.32 Timeline diagramm, Zeitleistendiagramm

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  import matplotlib.dates as mdates
4  from datetime import datetime
5
6
7  namedates = np.array([['2025-02-01','Kick-Off'],
8                        ['2025-02-09','Angebot'],
9                        ['2025-02-23','Auftrag'],
10                       ['2025-03-13','Spezifikation'],
11                       ['2025-07-10','Inbetriebname'],
12                       ['2025-10-10','Problem'],
13                       ['2025-10-21','Reparatur'],
14                       ['2025-11-02','Garantieende'],
```

```
                      ['2025-11-11','Kaputt']
                      ])
names = namedates[:,1]
dates = namedates[:,0]
dates = [datetime.strptime(d, "%Y-%m-%d") for d in dates]

# Choose some nice levels
levels = np.tile([-5, 5, -3, 3, -1, 1],
                 int(np.ceil(len(dates)/6)))[:len(dates)]

# Create figure and plot a stem plot with the date
fig, ax = plt.subplots(figsize=(8.8, 4), constrained_layout=True)
ax.set(title="Projektverlauf")

markerline, stemline, baseline = ax.stem(dates, levels,
                                         linefmt="C3-", basefmt="k-")

plt.setp(markerline, mec="k", mfc="w", zorder=3)

# Shift the markers to the baseline by replacing the y-data by zeros.
markerline.set_ydata(np.zeros(len(dates)))

# annotate lines
vert = np.array(['top', 'bottom'])[(levels > 0).astype(int)]
for d, l, r, va in zip(dates, levels, names, vert):
    ax.annotate(r, xy=(d, l), xytext=(-3, np.sign(l)*3),
                textcoords="offset␣points", va=va, ha="right")

# format xaxis with 4 month intervals
ax.get_xaxis().set_major_locator(mdates.MonthLocator(interval=4))
ax.get_xaxis().set_major_formatter(mdates.DateFormatter("%b␣%Y"))
plt.setp(ax.get_xticklabels(), rotation=30, ha="right")

# remove y axis and spines
ax.get_yaxis().set_visible(False)
for spine in ["left", "top", "right"]:
    ax.spines[spine].set_visible(False)

ax.margins(y=0.1)
fig.savefig('Zeitleiste.pdf')
```

Ergibt:

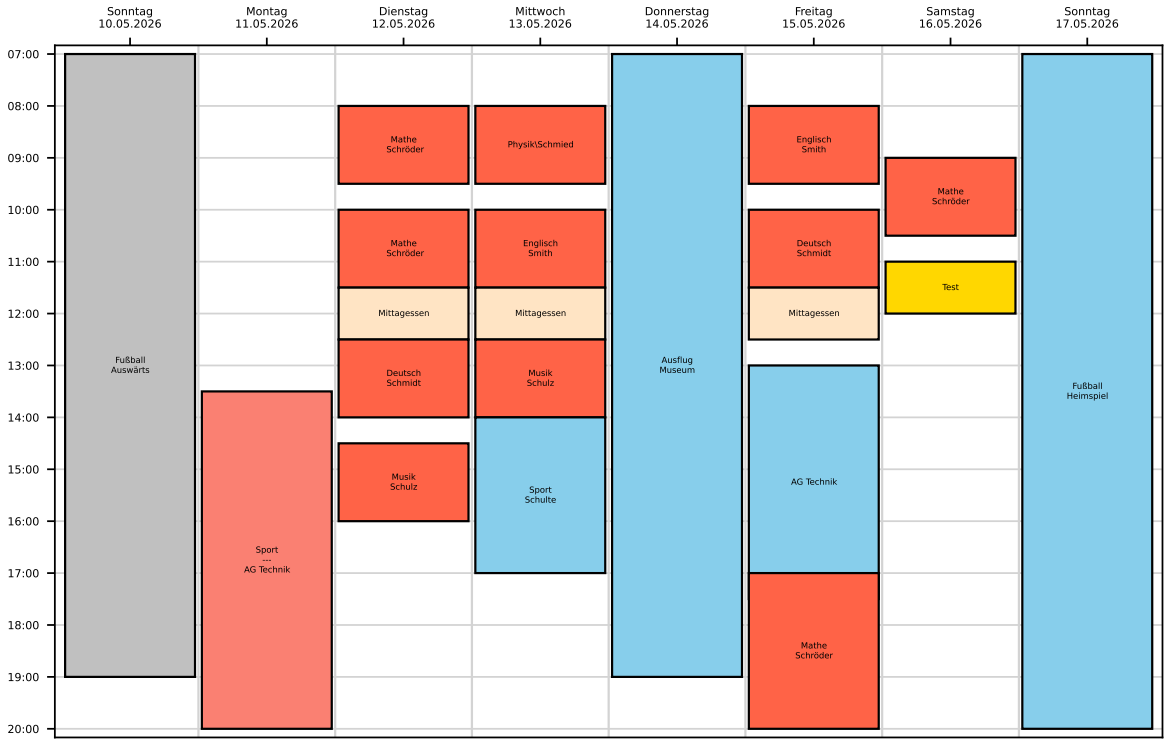


8.33 Stundenplan

```
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime, timedelta, date, time
from babel.dates import format_date, format_datetime, format_time
Sprache = ['de_DE', 'en_US'][0]  # oder [1] für Englisch
Farben = {'Reise':'silver', 'Soziales':'salmon', 'Vorlesung':'tomato', 'Essen':'bisque',
          'Exkursion':'skyblue', 'Exam':'gold'}
Stundenplan = np.array([
    [datetime(2026, 5, 10, 7, 0, 0), 12*60, "Fußball\nAuswärts", 'Reise'],
    [datetime(2026, 5, 11, 13, 30, 0), 6.5*60, "Sport\n---\nAG␣Technik", 'Soziales'],
    [datetime(2026, 5, 12, 8, 0, 0), 90, "Mathe\n␣Schröder", 'Vorlesung'],
    [datetime(2026, 5, 12, 10, 0, 0), 90, "Mathe\n␣Schröder", 'Vorlesung'],
    [datetime(2026, 5, 12, 11, 30, 0), 60, "Mittagessen", 'Essen'],
    [datetime(2026, 5, 12, 12, 30, 0), 90, "Deutsch\nSchmidt", 'Vorlesung'],
    [datetime(2026, 5, 12, 14, 30, 0), 90, "Musik\nSchulz", 'Vorlesung'],
    [datetime(2026, 5, 13,  8,  0, 0), 90, "Physik\Schmied", 'Vorlesung'],
    [datetime(2026, 5, 13, 10,  0, 0), 90, "Englisch\nSmith", 'Vorlesung'],
    [datetime(2026, 5, 13, 11,  30, 0), 60, "Mittagessen", 'Essen'],
    [datetime(2026, 5, 13, 12,  30, 0), 90, "Musik\nSchulz", 'Vorlesung'],
    [datetime(2026, 5, 13, 14,  0, 0), 180, "Sport\nSchulte", 'Exkursion'],
    [datetime(2026, 5, 14, 7,  0, 0), 12*60, "Ausflug\nMuseum", 'Exkursion'],
    [datetime(2026, 5, 15, 8,  0, 0), 90, "Englisch\nSmith", 'Vorlesung'],
    [datetime(2026, 5, 15, 10,  0, 0), 90, "Deutsch\nSchmidt", 'Vorlesung'],
    [datetime(2026, 5, 15, 11,  30, 0), 60, "Mittagessen", 'Essen'],
    [datetime(2026, 5, 15, 13,  0, 0), 45*60/10, "AG␣Technik", 'Exkursion'],
    [datetime(2026, 5, 15, 17,  0, 0), 180, "Mathe\nSchröder", 'Vorlesung'],
    [datetime(2026, 5, 16, 9,  0, 0), 90, "Mathe\nSchröder", 'Vorlesung'],
    [datetime(2026, 5, 16, 11,  0, 0), 60, "Test", 'Exam'],
    [datetime(2026, 5, 17, 7,  0, 0), 13*60, "Fußball\nHeimspiel", 'Exkursion']
                        ])
fig, ax = plt.subplots()
globalfontsize=5
barfontsize=3.8
StartStunde = 7
EndStunde = 20
DatumErsterTag =datetime(2026, 5, 10)
DatumLetzterTag =datetime(2026, 5, 17)

StartUhrzeit = DatumErsterTag + timedelta(hours=StartStunde)
Uhrzeit_Text = [format_time(StartUhrzeit+timedelta(minutes=30*k), format='short',
                            locale=Sprache) for k in range(27)]
Uhrzeit_Minuten = [-StartStunde*60-30*k for k in range(len(Uhrzeit_Text))]
ax.set_yticks(Uhrzeit_Minuten, Uhrzeit_Text, rotation=0, fontsize=globalfontsize)
ax.set_ylim([-EndStunde*60-10, -StartStunde*60+10])
ax.set_xlim([-0.55, (DatumLetzterTag - DatumErsterTag).days+0.55])
for VNr in range(len(Stundenplan)):
    Startuhrzeit_Minuten = Stundenplan[VNr,0].hour*60+Stundenplan[VNr,0].minute
    DatumVeranstaltung = Stundenplan[VNr,0]
    Dauer_Minuten = Stundenplan[VNr,1]
    Text = Stundenplan[VNr,2]
    ax.bar([(DatumVeranstaltung - DatumErsterTag).days], [-Dauer_Minuten],
           bottom=-Startuhrzeit_Minuten, width=0.95,
           facecolor=Farben[Stundenplan[VNr,3]], edgecolor='black', zorder = 10 )
    ax.text((DatumVeranstaltung - DatumErsterTag).days,
            -Startuhrzeit_Minuten-0.5*Dauer_Minuten, Text , zorder = 20,
            horizontalalignment='center', verticalalignment='center',
            fontsize=barfontsize )
ax.grid(axis='y', c='lightgray')
TagIndex = [t for t in range((DatumVeranstaltung - DatumErsterTag).days+1)]
TagNamen = [format_date((DatumErsterTag+timedelta(days=0+tidx)), format='EEEE',
                        locale=Sprache)  for tidx in TagIndex]
TagNamen = [TagNamen[tidx] + '\n'+format_date((DatumErsterTag+timedelta(days=0+tidx)),
                            format='medium', locale=Sprache)  for tidx in TagIndex]
for vgl in range((DatumVeranstaltung - DatumErsterTag).days):
```

```
    ax.axvline(x=0.5+vgl, c='lightgray', lw=1)

ax.set_xticks(TagIndex, TagNamen, fontsize=globalfontsize)
ax.set_yticks(np.arange(-StartStunde*60,-(EndStunde+1)*60,step=-60))
ax.tick_params(top=True, labeltop=True, bottom=False, labelbottom=False)
fig.set_size_inches([190/25.4, 120/25.4])
plt.subplots_adjust(left=0.06, right=0.99, top=0.93, bottom=0.01)
plt.savefig('Stundenplan.pdf')
plt.savefig('Stundenplan.png')
```

Ergibt:



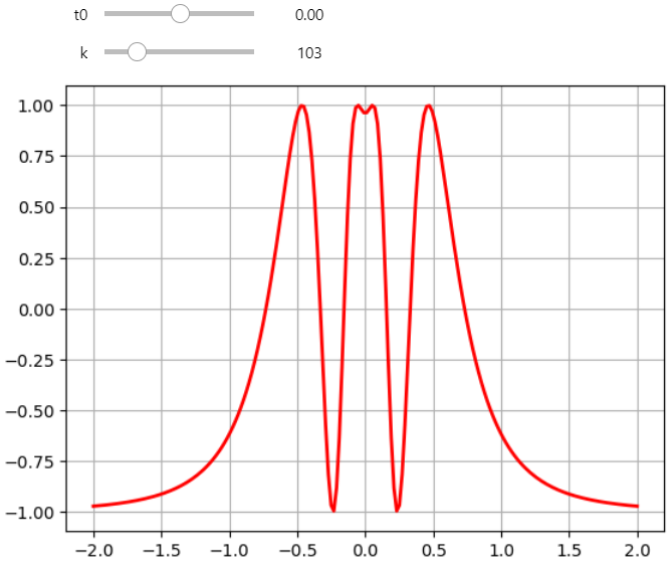
8.34 Interaktive Plots

Plots können mit Schiebereglern zu interaktiven Grafiken gemacht werden. Die Schieberegler bestimmen einen Variablenwert und daraufhin wird die Grafik sofort aktualisiert. Dies Funktionalität ist in JuPyter vorhanden, wird hier aber für iPython gezeigt. Der Vorteil von iPython ist, dass das Dateiformat eine Textdatei ist.

```
#%%
# Quelle: https://ipywidgets.readthedocs.io/
# en/stable/examples/Using%20Interact.html
import matplotlib.pyplot as plt
import numpy as np
from ipywidgets import interact
import ipywidgets as widget

def f(t0, k=1): #k=1 ist der startwert für den Schieberegler
    t = np.linspace(-2, 2, 200)
    y = -np.cos(1/(k/1000+(t-t0)**2))
    fig, ax = plt.subplots()
    g1, = ax.plot(t, y, 'r-',  lw=2)
```

```
    plt.grid()

# Format:(Startwert, Endwert, Schrittweite)
# Der Startwerte ist entweder die Mitte zwischen
# Startwert und Endwert oder wird in der Funktionsdefinition
# angegeben.
interact(f, t0=(-2,2,0.1), k=(1,500,1))
```

Ausgabe:



Teil II
Python als Programmiersprache

9 Kontrollstrukturen

9.1 Funktionen

Funktionen haben einen Funktionsnamen, einen oder mehrere Parameter und eventuell einen Rückgabewert.

```
def plusneun(zahl):
        erg = zahl + 9
        return(erg)
print(plusneun(1))
# ergibt 10
```

9.2 Fallunterscheidung mit if

Fallunterscheidungen:

```
a = 42
if 10 < a:
        print('10 < a')
elif 10 == a:
        print('10 = a')
else:
        print('a < 10')
```

9.3 Fallunterscheidung mit if (ternary operator)

Fallunterscheidungen nach dem Muster (Rückgabewehrt bei False, Rückgabewert für True)[Testabfrage]

```
a = 42
print(('false','frue')[a<10]) #ergibt: 'false'
```

9.4 for Schleife

For schleife (k durchläuft alle Zahlen zwischen 0 und 6. Die Schleife ende bei $k = 3$)

```
for k in range(7):
        if k == 3:
        break
```

9.5 Boolsche Ausdrücke

Boolsche Ausdrücke: `(a <= b) and (c != d) or (e == f)`

Boolsche Vergleiche von numpy-Arrays mit Wahrheitswerten (und = &, oder = |):

```
import numpy as np
x = np.array(range(1,10))
y = (3<=x) & (x<7)
z = (x<=3) | (7<x)
print(y*x)
print(z*x)
```

ergibt:

```
[0 0 3 4 5 6 0 0 0]
[1 2 3 0 0 0 0 8 9]
```

10 Datenstrukturen

10.1 Range

Die Funktion `range(k)` liefert alle Zahlen zwischen 0 und k-1

```
for k in range(3):
        print(k) # 0, 1, 2
```

10.2 Dictionarys

Dictionarys sind eine Name-Wert-Datenstruktur. Die Werte können über den Namen ausgelesen werden:

```
mydict = {'alter':41, 'groesse_cm':190, 'Name':'Markus'}
print(mydict['Name']) # 'Markus'
```

Dictionarys können auch in Listen zusammengefasst sein:

```
mydict = []
mydict.append({'alter':41, 'groesse_cm':190, 'Name':'Markus'})
mydict.append({'alter':42, 'groesse_cm':175, 'Name':'Pascal'})
print(len(mydict)) # 2
print(mydict[0]['Name']) # 'Markus'
```

Dictionary erweitern:

```
my_dict = {"username": "XYZ"}
my_dict['name']='Nick'
```

10.3 Tupel

Tupel sind aneinandergehängte Elemente. Die Liste an Elementen kann im nachhinein nicht geändert werden.

```
L = (1, 2, 3)
print(L[0]) # 1
L[0] = 7 # geht nicht
```

10.4 Set

Sets sind im mathematischen Sinne Mengen. Ein Element kann nur einmal vorhanden sein. Die Reihenfolge ist nicht definiert

```
L = {1, 2, 3}
print(len(L)) # Laenge, ergibt 3
L.remove(2) # Fehler, wenn nicht vorhanden
L.discard(3) # Kein Fehler, wenn nicht vorhanden
print(L) #ergibt {3}
L.union({4, 5}) #ergibt {1, 4, 5}
```

10.5 Arrays gleicher Länge zu Tupeln

Zwei Arrays gleicher Länge können zu einer Liste von Tupeln umgewandelt werden

```
A = [1, 2, 3]
B = [10, 9, 8]
C = zip(A, B)
print(list(C))
# [(1, 10), (2, 9), (3, 8)]
```

10.6 Arrays verbinden

Aneinanderhängen von Listen zu einer Gesamt-Liste

```
y = [1] + [2, 3, 4] + [5]
# ergibt: [1, 2, 3, 4, 5]
```

10.7 Array um ein Element verlängern

```
y = [1, 3, 4]
y.append(42) # Rückgabewert ist none
print(y) # ergibt [1, 3, 4, 42]
```

10.8 Länge eines Arrays

Länge eines Arrays

```
y = [2, 3, 4]
len(y)# ergibt: 3
```

10.9 deep und shallow copy

Das erzeugen einer neuen Variablen erzeugt nicht zwangsläufig ein neues undabhängiges Objekt. Mit copy und deepcopy können flache und tiefe Kopien erstellt werden. Bei der flachen Kopie wird ein neues Objekt auf höchster Ebene erzeugt. Bei der tiefen Kopie werden alle verzweiten Objekte dupliziert. Beispiel:

```
import copy
z = 666
A = [z, 2, 3]
B = A # zusaetzliche Variable fuer gleiches Objekt
B[1]=42
print(A) # ergibt [42, 2, 3]
print(B) # ergibt [42, 2, 3]
print((id(A), id(B), id(A[0]), id(B[0])))

A = [z, 2, 3]
B = copy.copy(A) # shallow copy
B[1]=42
print(A) # ergibt [42, 2, 3]
print(B) # ergibt [42, 2, 3]
print((id(A), id(B), id(A[0]), id(B[0])))

A = [z, 2, 3]
B = copy.deepcopy(A)
B[0]=42
print(A) # ergibt [1, 2, 3]
print(B) # ergibt [42, 2, 3]
print((id(A), id(B), id(A[0]), id(B[0])))
```

11 Strings (Zeichenketten)

11.1 Strings aneinanderhängen

Eine list von Strings zu einem einzigen String aneinanderhängen:

```
neustr = "".join(["a", "b", "c"])
```

Alternativ:

```
neustr = "a" + "b" + "c"
```

11.2 Strings mit Trennstring aneinanderhängen

Eine list von Strings zu einem einzigen String mit Trennstring aneinanderhängen:

```
s = "␣kennt␣".join(["Markus", "Maria", "Andrea"])
print(s) # Markus kennt Maria kennt Andrea
```

11.3 Strings formatieren

Variablen und Textbausteine zu einem String zusammenbauen:

```
x = 5
y = 7
neustr = f"x␣=␣{x},␣y␣={y}"
# ergibt "x = 5, y =7"
```

11.4 Substrings extrahieren

Unter-Zeichenketten extrahieren. Mit den eckigen Klammern kann eine Startposition, Endposition und eine Schrittweite angegeben werden. Wird eine Zahl als Grenze ausgelassen, bedeutet das soviel wie „alle".

```
zk = 'hallo␣welt'
print(zk[0:5]) # hallo
print(zk[0:5:2]) # hlo
print(zk[4::-1]) # ollah
print(zk[:-1]) # hallo wel
```

11.5 Substring finden

Suchstring in einem anderen String finden. Das Ergebnis ist die Startposition des ersten gefundenen Substrings:

```
pos = "Hallo␣Welt".find("l")
```

11.6 String in Int umformen

```
zahl = int("123")
```

11.7 Int in String umformen

```
zahls = str(123)
```

11.8 float in String umformen

```
x=13.1415
e =  'pi␣=␣{:f}'.format(x)
s = 'pi␣=␣{0:1.2f}'.format(x)
print(e) # 'pi = 13.141500'
print(s) # 'pi = 13.14'
# Integer-Wert
print('Wert:␣{0:d}'.format(27828))
# fuehrende Nullen:
print('{:03d}'.format(7)) # 007
# Scientific (nicht Eng.) Format:
print(f'{0.000002342:.3E}') # 2.342E-6
print(f'{0.000002342:E}') # 2.342000E-6
```

12 VisualStudio Code (Jupyter Lab)

12.1 Installation

- Im Anaconda -> cmd.exe Prompt -> „conda install ipykernel”

Informationen über den Kernel einholen:

```
import sys
print(sys.executable)
print(sys.version)
print(sys.version_info)
```

Besser: Statt Anaconda Miniconda installieren. In Miniconda sind nur wenige packages vorhanden. Diese können wie folgt installiert werden.

```
PS C:\Users\USERNAME\appdata\Local\miniconda3\Scripts>  .\conda.exe install matplotlib
```

12.2 Tastaturkürzel

- Markierte Zelle -> y: Code-Zelle
- Markierte Zelle -> m: markdown-Zelle
- Alt + Enter: Zelle ausführen und neue Zelle hinzufügen

- Strg + Shift + Enter: Zelle ausführen
- F5: Starte Debugger
- F10: Nächster Schritt

12.3 Neues Notebook anlegen

- Strg + Shift + p (Eine Kommandozeile wird angezeigt)
- „Create: New Jupyter Notebook" `RET`

12.4 Mehrzeilige Formleln einfach editieren

- In LyX eine abgesetzte Formel erstellen und dort ein Array mit zwei Spalten anlegen
- Dieses markieren, kopieren und in die Markdown-Zelle einfügen
- In der Markdown-Zelle „Strg + RET" drücken, sodass die Zelle in eine Formel gewandelt wird.

12.5 Sinnvolle Einstellungen

Unter File -> Preferences -> Settings wird nach „Scroll" gesucht. Im Bereich „Jupyter" gibt es die Einstellung „Jupyter: Always Scroll On New Cell". Dies sollte aktiviert sein. Damit zeigt das Ausgabefenster immer die letzten Ausgaben. Ein maunelles Scrollen zu den Ausgaben entfällt.

Weiterhin sollte eine Abfrage erscheinen, ob eine Datei gespeichert werden soll, wenn VS-Code geschlossen wird und noch nicht gespeicherte Code-Dateien offen sind: File → Preferences → Setting → Text Editor → Hot Exit: Off.

12.6 Standardkernel einstellen

View -> Extensions -> Python -> RMB Extension Settings -> Python: Default Interpreter Path editieren. (Beispiel `C:\Users\UNAME\anaconda3\python.exe`)

(Oder auch `C:\Users\USERNAME\AppData\Local\Continuum\anaconda3`)

- Etwas ungewöhnlich ist, dass der Standard-Kernel für die Jupyter Lab-extension in der Python extension eingestellt wird.

13 Dateien

13.1 Textdateien zeilenweise lesen

Textdateien können ohne zusätzliche importierte Module gelesen werden ('r' für 'read'):

```
fd = open("Datei.lyx", 'r')
print(fd.readline()) # zeige Zeile 1
fd.close()
```

13.2 Textdateien zeilenweise schreiben

Textdateien können ohne zusätzliche importierte Module gelesen werden ('w' für 'write'):

```
fd = open("Datei.txt", 'w') # 'a' fuer anhaengen/append
fd.write('Hallo Welt\n')
fd.close()
```

13.3 Textdateien ganz lesen

Textdateien können ohne zusätzliche importierte Module gelesen werden ('r' für 'read') in einen String eingelesen werden:

```
fd = open("Testdatei.text", 'r')
inhalt = fd.read() # Lese Datei
fd.close()
```

13.4 Über alle Zeilen iterieren

Textdateien können ohne zusätzliche importierte Module gelesen werden:

```
fd = open("Datei.lyx", 'r')
for line in fd:
    print(line) # zeige Zeile
fd.close()
```

14 VisualStudio Code (Python, iPython)

14.1 iPython Zellen

iPython-Zellen werden erzeugt, indem als Kommentar # %% zugefügt wird. Alles was danach kommt gehört zu einer Zelle. Eine Zelle wird ausgeführt, indem Strg+Ret gedrückt wird.

Die Zelle wird ausgeführt und eine neue Zelle erzeugt mit Shift + Ret.

Mit Alt+Ret wird die aktuelle Zelle nicht ausgeführt und eine neue Zelle erzeugt.

14.2 iPython Variablen-Browser

Der Variablenbrowser wird angezeigt, indem der Cursor in das Interaktive Ausgabefenster (häufig „Interactive-1") auf  geklickt wird. Alternativ: View -> open view -> „view variables"

14.3 Debugger

Wenn ein Haltepunkt gesetzt wurde und dann RMB → Edit Breakpoint... → Log Message gewählt wurde, wird mit `Wert = {Variablenname}` an der Stelle des Breakpoints der Wert der Variablen am Haltepunkt in der Debug-Console ausgegeben. Beispiel: `Wert = 42`.

14.4 Cursorposition Anzeigen

Die Cursorposition wird in der Status-Bar angezeigt. Sollte die Ausgeblendet sein, wird sie unter View->Apperance->Show Status Bar wieder angezeigt.

15 Systemzugriff

15.1 Dateiname des Programms

Der Dateiname des Python-Programms (Funktioniert nicht mit iPython & Jupyter) wird wie folgt ermittelt:

```
import os
print(__file__) # ganzer Pfad
# Nur Dateiname (programm.py):
print(os.path.basename(__file__))
# Pfad des Programms:
print(os.path.dirname(__file__))
```

Mit Jupyter wird folgender Code zur Ermittlung des Programmnamens verwendet:

```
print(os.path.basename(sys.argv[0]))
```

15.2 Sound/Wave/Audio ausgebem

Ausgabe eines Audiosignales, welches auf einer Funktion beruht (Funktioniert mit iPython & Jupyter):

```
#%%
import numpy as np
from IPython.display import Audio
from math import pi
fsample = 44000 # Abtastfrequenz
f1 = 440
f2 = 441
tmax = 5 #Maximaldauer
t = np.linspace(0,tmax,tmax*fsample)
wave_audio = 0.2*(np.cos(2*pi*f1*t)+np.cos(2*pi*f2*t))
```

```
Audio(wave_audio, rate=fsample)
```

15.3 Datum und Zeit

Ausgabe eines ISO-Zeitstempels#%%

```
import datetime print(datetime.datetime.now(
   datetime.timezone.utc).strftime("%Y-%m-%d␣%H:%M␣Uhr"))
# erzeugt 2024-10-30 20:15 Uhr
print(datetime.datetime.now().isoformat())
# erzeugt 2024-10-30T17:48:16.600919
```

16 SQLite-Datenbank

16.1 Daten abfragen

Daten aus einer SQLite-Datenbank abfragen:

```
import sqlite3
con = sqlite3.connect("Moduldaten.db")
cur = con.cursor()
res = cur.execute("SELECT␣Spalte1,␣Spalte2␣from␣Tabelle;")
print(res.fetchone()) #gibt das erste aus
print(res.fetchall()) #gibt den Rest aus
# [('a','b'), ('c','d'),('e','f')...]
con.close()
```

17 Objektorientierung

17.1 Klassen

Klassenvariablen sind in allen Objekten der Klasse gleich. Wird die Klassenvariable in einer Instanz (aka Objekt) geändert, ist sie auch in einer anderen Instanz geändert.

```
class Kumpel:
        alter = 18 # Klassenvariable
        # Methoden:
        def gruss(self):
                print('Glueck␣Auf!')
        pass
```

Mit der Instanzierung wird eine Instanz (ein Objekt) der Klasse gebildet:

```
Maria = Kumpel()
```

Der Konstruktor (Es gibt in Python kein Überladen, also mehrere Funktionen gleichen Namens und unterschiedlichen Parametern) heißt `__init__` und ermöglicht das Setzen von Instanzvariablen (Variablen die zu dem Objekt gehören):

```
class Kumpel:
        # Konstruktor:
        def __init__(self, N, A=2):
```

```
4                 self.Nachname = N
5                 self.Arme = A
6         pass
7 Maria = Kumpel('Schulz')
8 Jo = Kumpel('Jo', 3)
9 print(Maria.Nachname) # 'Schulz'
```

Optionale Parameter können mit Default-Werten angegeben werden. Wenn diese Parameter nicht angegeben werden, werden die Default-Werte verwendet.

Ein Objekt kann in einen String über die Funktion `__str__` gewandelt werden:

```
1 class Kumpel:
2   def __init__(self, N):
3     self.Nachname = N
4   def __str__(self):
5     return('Nachname␣=␣'\
6     + self.Nachname)
7   pass
8
9 Maria = Kumpel('Schulz')
10 print(Maria) # 'Schulz'
```

18 Anaconda-Spezialitäten

18.1 graphviz

Hinzufügen und die Pakete

- graphviz
- python-graphviz
- pydot

installieren.

18.2 Anaconda aufräumen

In Anaconda → Home → cmd.exe Prompt → „conda clean -a”

18.3 Anaconda Updates installieren

In Anaconda → Home → cmd.exe Prompt → „conda update --all”
oder
In Anaconda → Home → cmd.exe Prompt → „conda update jupyterlab”

19 Regular Expressions

19.1 Extrakte

Wenn aus dem String „`magicwoche 12`" die Zahl 12 extrahiert werden soll, dann kann das mit folgendem Befehl erfolgen

```
import re #regular expressions
s1 = "magicwoche␣12"
p = re.compile('magicwoche␣([0-9]+)')
print(p.match(s1).group(1))
```

Die runden Klammern um `[0-9]+` bilden eine Gruppe. Diese Gruppe wird mit `.match(s).group(1)` angesprochen. Mit Match werden nur Treffer zu Beginn eines Strings gefunden. Für eine Volltextsuche sollte man `search` verwenden.

19.2 Muster

Muster	Wirkung
`\s`	Whitespace
`\S`	alles außer Whitespace
`[(]`	eine geöffnete runde Klammer
`[a-z]`	ein kleiner Buchstabe
`[0-9]`	eine Ziffer

Muster	Wirkung
`[0-9]{4}`	vier Ziffern
`[0-9]{4,}`	vier oder mehr Ziffern
`[0-9]{2,5}`	zwischen 2 und 5 Ziffern
`[0-9]?`	Keine oder eine Ziffer
`[0-9]*`	keine, eine oder mehrere Ziffer
`[0-9]+`	eine oder mehrere Ziffern

19.3 Substrings ersetzen

Mit regular expressions können substrings gefunden werden, die dann durch andere Strings ersetzt werden:

```
import re
erg = re.sub('y($|[^\^])', # Muster
'y(t)', # Ersatzstring
r'9*y^{\prime}(t)␣+␣20*y') # Suchtext
print(erg) #ergibt 9*y^{\prime}(t) + 20*y(t)
```

In diesem Besipiel bedeutet `($|[^\^])`, dass y am Ende stehen muss ($) oder nicht von einem ^ gefolgt wird. Was in runden Klammern steht ist eine Gruppe.

19.4 Test auf Funde

Mit dem folgenden Code wird überprüft, ob ein String zu einer regular expression passt. `match` gibt None zurück, wenn nichts gefunden wurde.

```
import re
p = re.compile('hal')
if p.match("hallo")!=None:
    print(p.match("hallo").group(1))
```

20 Anwendungen

20.1 Lineare Regression mit Pandas

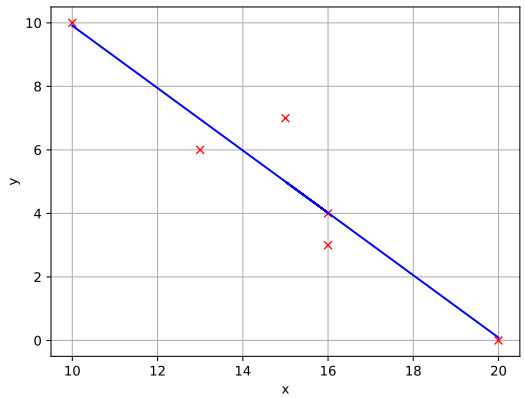
Die Bibliothek Pandas ermöglich die Arbeit mit Tabellen wie mit einem Tabellenkalkulationsprogramm. Anhand eines Beispieles zur linearen Regression wird die Bedienung gezeigt:

```
import numpy as np
import pandas as pd

xy = [\
      (20, 0),\
      (16, 3),\
      (15, 7),\
      (16, 4),\
      (13, 6),\
      (10, 10),\
      ]
df = pd.DataFrame(xy, columns=['x', 'y'])
print(df)
# Mittelwert = Summe durch Anzahl
x_avg = df['x'].sum()/df['x'].count()
y_avg = df['y'].sum()/df['y'].count()
y_avg2 = df['y'].mean() #geht auch
# Erzeugung neuer Spalten:
df['Dx'] = df['x']-x_avg
df['Dy'] = df['y']-y_avg
df['Dx*Dy'] = df['Dx']*df['Dy']
df['Dx^2'] = df['Dx']**2
m = df['Dx*Dy'].sum()/df['Dx^2'].sum() # Steigung
b = y_avg-m*x_avg # Offset
print(df)

# Ergebnis plotten:
import matplotlib.pyplot as mpl
fix, ax = mpl.subplots()
ax.plot(df['x'],df['y'], 'rx')
ax.plot(df['x'],m*df['x']+b, 'b-')
mpl.grid()
mpl.xlabel('x')
mpl.ylabel('y')
mpl.savefig('LinRegPandas.pdf')
```

Erzeugt:

```
    x   y   Dx   Dy  Dx*Dy  Dx^2
0  20   0  5.0 -5.0  -25.0  25.0
1  16   3  1.0 -2.0   -2.0   1.0
2  15   7  0.0  2.0    0.0   0.0
3  16   4  1.0 -1.0   -1.0   1.0
```

```
4   13    6 -2.0   1.0    -2.0    4.0
5   10   10 -5.0   5.0   -25.0   25.0
```

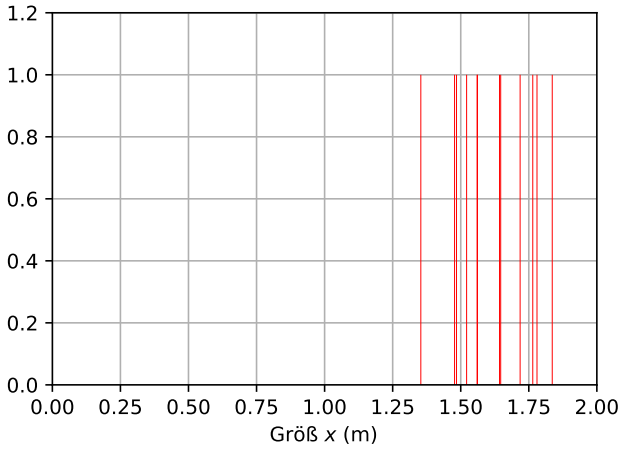


20.2 Impulsplot

Ausgabe von Impulsplots:

```
import matplotlib.pyplot as plt
import numpy as np
xavg = 1.5
xsigma = 0.15
fig, ax = plt.subplots()
fig.set_size_inches(120/25.4,80/25.4)
np.random.seed(0)
x = np.random.normal(xavg, xsigma, 12)
y = 0.0*x+1
ml, sl, bl = ax.stem(x,y,linefmt='r-',markerfmt='␣', basefmt='␣')
plt.setp(sl, linewidth=0.5)
plt.setp(bl, color="none")
ax.grid()
ax.set_xlim([0,2])
ax.set_ylim([0,1.2])
plt.xlabel('Groesse␣$x$␣(m)')
plt.subplots_adjust(left=0.17,\
  right=0.97, top=0.97, bottom=0.15)
plt.savefig('bilddatei.pdf')
plt.show()
```

Ausgabe:

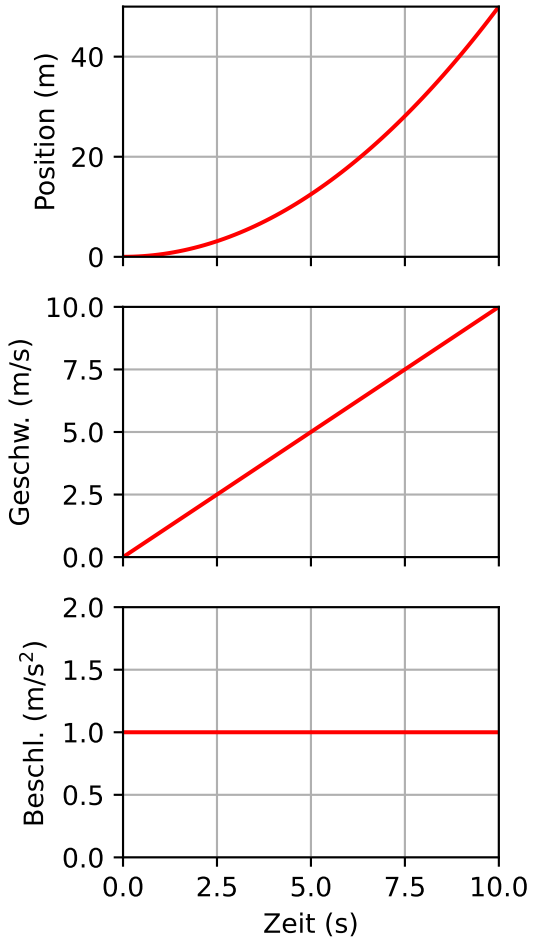


20.3 Mehrere Plots übereinander

Mehrere Plots übereinander:

```
import numpy as np
import matplotlib.pyplot as mpl
t = np.linspace(0,10,75)
fig, axs = mpl.subplots(3)
fig.set_size_inches(70/25.4,125/25.4)
axs[0].set_ylabel('Position␣(m)')
gxphase1, = axs[0].plot(t, 1/2*t**2, 'r-', label='besch')
axs[1].set_ylabel('Geschw.␣(m/s)')
gvphase1, = axs[1].plot(t, t, 'r-')
axs[2].set_ylabel('Beschl.␣$\mathrm{(m/s^2)}$')
gaphase1, = axs[2].plot(t, 1+0*t, 'r-')

axs[0].grid()
axs[1].grid()
axs[2].grid()

axs[0].set_xlim([0,10])
axs[0].set_xticklabels([])
axs[1].set_xlim([0,10])
axs[1].set_xticklabels([])
axs[2].set_xlim([0,10])
axs[0].set_ylim([0,50])
axs[1].set_ylim([0,10])
axs[2].set_ylim([0,2])

axs[2].set_xlabel('Zeit␣(s)')
mpl.subplots_adjust(left=0.23, right=0.94, top=0.99, bottom=0.09, wspace=0.1, hspace=0.2)
fig.show()
fig.savefig('x-v-a-diagramm.pdf')
```

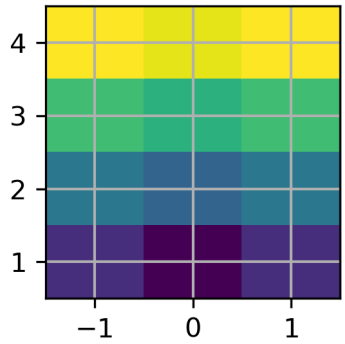
Ausgabe:



20.4 Pixel-Farbnetz

Ein Pixel-Farbnetz wird aus drei Matrizen gleicher Größe erzeugt. Die erste gibt für jeden Pixel die x-Koordinate an. Die zweite Matrix die y-Koordinate und die dritte Matrix den Farbwert des Pixels. Mit der Option `shading='auto'` wird angegeben, dass die Mitte der Pixel den x- und y-Koordinaten entsprechen. Entfällt diese Option, sind die äußeren Ecken der Randpixel an den Maximal-Koordinaten.

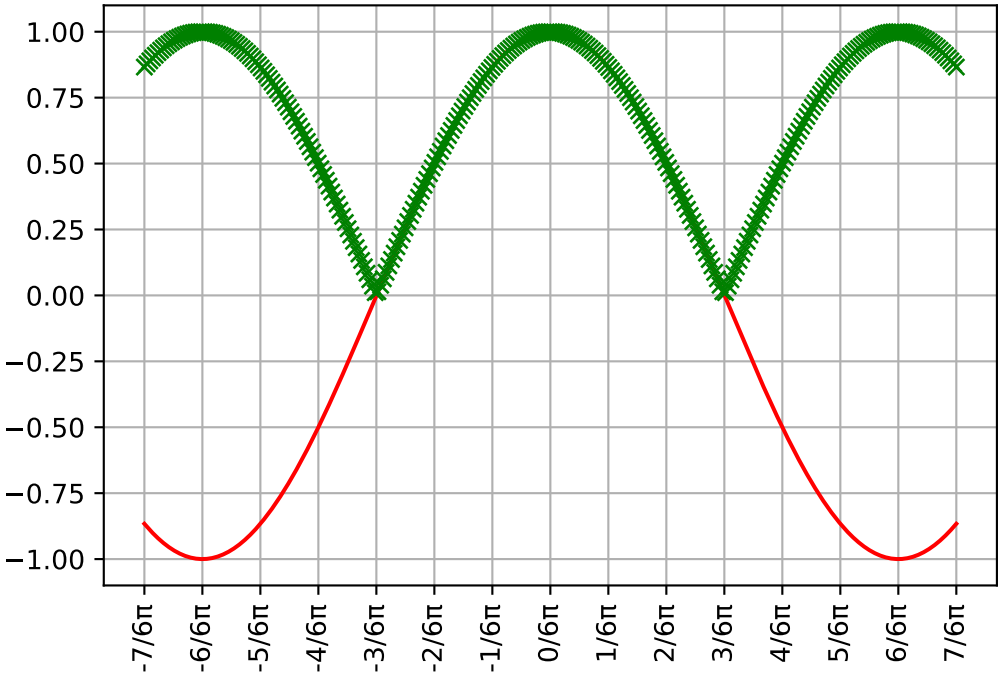
```
import numpy as np
import matplotlib.pyplot as plt
xv = np.linspace(-1,1,3)
yv =  np.linspace(4,1,4)
Mx, My = np.meshgrid(xv, yv)
Mz = np.sqrt(Mx**2+My**2)
fig, ax= plt.subplots()
fig.set_size_inches(50/25.4, 50/25.4)
ax.pcolormesh(Mx, My, Mz,\
  shading='auto')
ax.grid(lw=1)
```



20.5 Benutzerspezifische Achsenbeschriftung

Die Beschrifung einer Achse lässt sich mit Koordinaten und Strings benuzerspezifisch anpassen. In diesem Beispiel ist die horizontale Achsenbeschriftung ein vielfaches von pi/6

```
#%%
import matplotlib.pyplot as plt
import numpy as np
from numpy import *
from math import pi
import os

fig, ax = plt.subplots()
p = np.array([p for p in range(-7,7+1)])
t = p/6*pi
lbl = [str(k)+'/6pi' for k in p] # Hier pi durch
# das Unicode-Pi ersetzen.
# ax.plot(t, i1_von_t, 'g-')
thighres = np.linspace(-7/6*pi,7/6*pi,300)
ax.plot(thighres, cos(thighres), 'r-')
ax.plot(thighres, sqrt(1-sin(thighres)**2), 'gx')
plt.xticks(t, lbl, rotation=90)
ax.grid()
plt.savefig(os.path.basename(__file__).replace('.py','') + '_gen.pdf')
```



21 PDF-Dateierzeugung

21.1 PDF-Dateiausgabe

Mit dem folgenden Code wird eine PDF-Datei mit der Breite 80 mm und der Höhe 50 mm erzeugt. In Anaconda muss gegebenenfalls das Paket „reportlab" installiert werden.

```
from reportlab.pdfgen.canvas import Canvas
# pagesize units are in 1/72 inch
canvas = Canvas("ausgabe.pdf", pagesize=(80/25.4*72, 50/25.4*72))
FoSi = 10 # Fontsize
canvas.setFont("Times-Roman", FoSi)
textobject = canvas.beginText(2,50/25.4*72-FoSi)
txt = "hallo"
txt = txt + "\n welt"
txt = txt + "\n dritte Zeile"
txt = txt + "\n vierte zeile"
textobject.textLines(txt)
canvas.drawText(textobject)
canvas.save()
```

Erzeugt:

hallo
welt
dritte Zeile
vierte zeile

22 Pandas

22.1 Tabellenbreite in Pandas

Bei der Ausgabe von Tabellen mit Pandas wird ab einer bestimmten Breite ein Zeilenumbruch eingefügt. Der nachfolgende Code verhindert das (juPyter)

```
import pandas as pd
pd.set_option('display.expand_frame_repr', False)
```

22.2 Mehrzeilenstring als Datenquelle

Die Dateneingabe als mehzeiliger String kann folgendermaßen erfolgen:

```
#%%
import sys
import pandas as pd
from io import StringIO
txtdata = StringIO("""
Author;␣␣␣␣␣␣Titel;␣␣ISBN
Muster,␣Max;␣Test;␣␣␣␣12345
""")
df = pd.read_csv(txtdata, sep=";")
print(df)
```

22.3 Spalten kombinieren

Inhalt zweier Spalten zusammenführen, in ein Set zusammenfassen und ein Element entfernen

```
import pandas as pd
dummy_data3 = {
        'id': ['1', '2', '3'],
        'F1': [11, 12, 13],
        'F2': [21, 22, 23],}
df3 = pd.DataFrame(dummy_data3)
F1L = df3['F1'].tolist()
F2L = df3['F2'].tolist()
elemente =  set(F1L).union(set(F2L))
elemente.remove(22)
print(elemente) # {21, 23, 11, 12, 13}
```

23 Konsolenanwendungen

23.1 Benutzereingaben in der Konsole

Bei einfachen Anwendungen kann der Benutzer auch in der Konsole Eingaben machen:

```
text = input("Eingabe:␣")
print(text)
```

Dies kann auch in einer Schleife genutzt werde, um den Programmablauf anzuhalten:

```
for i in range(10):
    print(i)
    input()
```

24 GUIs

24.1 Messagebox

Einfache Messagebox:

```
from tkinter import *
from tkinter import messagebox
root = Tk()
root.withdraw()
messagebox.showinfo('Dialogtitel', 'Nachricht')
```

Wenn mehrere Messageboxen nacheinander angezeigt werden, sollen sie im Vordergrund angezeigt werden

```
root = Tk()
root.withdraw()
texto = Toplevel(root)
messagebox.showinfo('Warn', 'Messagetext', parent=texto)
root.destroy()
```

25 XML-Dateien

25.1 Elemente finden

Elemente eines bestimmten Typs (in diesem Beispiel „element") rekursiv finden:

```
import xml.etree.ElementTree as ET
tree = ET.parse('schaltplan.qet')
root = tree.getroot()
for e in root.iter('element'):
    print(e)
```

25.2 Subelemente ermitteln

Ob und wieviele Subelemente (child elements) ein Element hat, lässt sich mit der list-Funktion ermitteln:

```
1 import xml.etree.ElementTree as ET
2 tree = ET.parse('schaltplan.qet')
3 root = tree.getroot()
4 print(list(root))
```