

Konzeptsammlung Python

Thorbjörn Siaenen

Version v. 8. Mai 2025

Dieses Werk ist lizenziert unter einer Creative Commons “Namensnennung 4.0 International” Lizenz.



Inhaltsverzeichnis

1	Syntax	1
1.1	Zeilenumbruch im Code	1
1.2	Kommentare	1
1.3	Mehrzeilenkommentare	2
1.4	Variablentypen ermitteln	2
1.5	Einzelne Variablen löschen	2
2	Zahlen-Operationen	2
2.1	Komplexe Zahlen	2
2.2	Real- und Imaginärteil extrahieren	2
2.3	Quadrieren	2
2.4	Ganzzahlige Division	3
2.5	Modulo-Operator (Divisionsrest)	3
2.6	Aufrunden	3
2.7	Abrunden	3
2.8	Vergleiche Float und Integer	3
3	Kontrollstrukturen	3
3.1	Funktionen	3
3.2	Fallunterscheidung mit if	4
3.3	Fallunterscheidung mit if (ternary operator)	4
3.4	for Schleife	4
3.5	Boolsche Ausdrücke	4
4	Objektorientierung	5
4.1	Klassen	5

5	Listen/Arrays/Datenstrukturen	5
5.1	Range	5
6	Datenstrukturen	6
6.1	Dictionarys	6
6.2	Tupel	6
6.3	Set	6
6.4	Arrays gleicher Länge zu Tupeln	7
6.5	Arrays verbinden	7
6.6	Länge eines Arrays	7
6.7	deep und shallow copy	7
7	Strings (Zeichenketten)	8
7.1	Strings aneinanderhängen	8
7.2	Strings mit Trennstring aneinanderhängen	8
7.3	Strings formatieren	8
7.4	Substrings extrahieren	8
7.5	Substring finden	8
7.6	String in Int umformen	9
7.7	Int in String umformen	9
7.8	float in String umformen	9
8	Jupyter Lab	9
8.1	Start von .ipynb-Dateien per Doppelklick (Windows)	9
8.2	Zugriff auf andere Laufwerke (Windows)	9
8.3	Hotkeys	10
8.4	Sinnvolle Extensions	10
8.5	interact	10
8.6	Flickern interaktiver Grafiken verhindern	11
8.7	Anzeigegröße von Grafiken ändern	11
8.8	Anzeige von Markdown-Code aus einer Python-Zelle	11
8.9	Automatisiert erstellte Tabelle in Markdown	12
9	VisualStudio Code (Jupyter Lab)	12
9.1	Installation	12
9.2	Tastaturkürzel	13
9.3	Neues Notebook anlegen	13
9.4	Mehrzeilige Formeln einfach editieren	13
9.5	Sinnvolle Einstellungen	13
9.6	Standardkernel einstellen	14
10	VisualStudio Code (Python, iPython)	14
10.1	iPython Zellen	14
10.2	iPython Variablen-Browser	14

10.3 Debugger	14
10.4 Cursorposition Anzeigen	14
11 Dateien	14
11.1 Textdateien zeilenweise lesen	14
11.2 Textdateien zeilenweise schreiben	15
11.3 Textdateien ganz lesen	15
11.4 Über alle Zeilen iterieren	15
12 SQLite-Datenbank	15
12.1 Daten abfragen	15
13 Systemzugriff	16
13.1 Dateiname des Programms	16
13.2 Sound/Wave/Audio ausgeben	16
13.3 Datum und Zeit	16
14 Mathematische Funktionen	16
14.1 Fakultät (x!)	16
14.2 arctan2	17
14.3 Exponentialfunktion	17
15 Numpy/Numerik	17
15.1 Numpy Array erzeugen	17
15.2 Numpy Arrays verbinden	17
15.3 Numpy Matrix aus Vektoren, vertikal	18
15.4 Numpy Matrix aus Vektoren, horizontal	18
15.5 nparray 1 D adressieren	18
15.6 nparray 1 D Differenzen	18
15.7 Numpy arrays	19
15.8 nparray 2 D	19
15.9 nparray Größe	19
15.10 nparray transponieren	19
15.11 Inverse Matrix (Matrix invertieren)	19
15.12 Pseudoinverse einer Matrix	20
15.13 nparray umdrehen	20
15.14 Lineares Gleichungssystem lösen	20
15.15 Polynomdivision	21
15.16 Integrale (Trapezregel)	21
15.17 Potenz einer Matrix	21
15.18 Integral einer Funktion	22
15.19 Zweidimensionales Integral	22
15.20 Integral einer komplexwertigen Funktion	22
15.21 Dreifachintegral	23

15.22DGL 1. Ordnung numerisch lösen	23
15.22.1 Faktorisierung eines Polynoms	23
15.22.2 Einheitsmatrix	23
15.22.3 Determinante	24
15.22.4 Elementmanipulation	24
15.22.5 Matrizenmultiplikation	24
15.22.6 Vektoren und Matrizen aneinanderhängen	24
15.22.7 XY-Matrizen	25
15.22.8 Zufallszahlen	25
15.22.9 lineare Differenzialgleichung n-ter Ordnung lösen	26
15.22.10 Nichtlineare Differenzialgleichung n-ter Ordnung	27
15.23 Werteliste nahe Null zu Null runden	28
15.24 Abschnittsweise definierte Funktionen	28
15.25 Heaviside-Funktion (Sprungfunktion)	28
16 Symbolische Mathematik	29
16.1 Brüche	29
16.2 Terme rational machen	29
17 Komplexe Zahlen	29
17.1 Ableitungen	30
17.2 Integrieren	30
17.3 Ausmultiplizieren	30
17.4 Ersetzungen	30
17.5 Numerische Auswertung	31
17.6 Evaluation verhindern	31
17.7 Vereinfachungen	31
17.8 Automatische Vereinfachung	32
17.9 Ausklammern	32
17.10 Vereinfachung von Rationalen Funktionen (Polynombrüche)	32
17.11 Partialbruchzerlegung	32
17.12 Ausmultiplizieren von Partialbrüchen	32
17.13 Ausgabe in Formel-Schreibweise	33
17.14 Exakte Lösungen linearer Gleichungssysteme	33
17.15 Matrix symbolisch invertieren	33
18 Anaconda-Spezialitäten	34
18.1 graphviz	34
18.2 Anaconda aufräumen	34
18.3 Anaconda Updates installieren	34
19 Regular Expressions	35
19.1 Extrakte	35
19.2 Muster	35

19.3	Substrings ersetzen	35
19.4	Test auf Funde	36
20	Matplotlib	36
20.1	Einfachster Plot	36
20.2	Schriftart Stix	36
20.3	Ausgabe verfügbarer Schriftarten	36
20.4	Auflösung von Rastergrafiken	37
20.5	Universal-Plot	37
20.6	Kommas statt Punkte	37
20.7	Achsenzahlen verschieben	38
20.8	Positionen Hilfsgitterlinien	38
20.9	Beschriftungspfeile	38
20.10	Beschriftungstext	39
20.11	Größe der Abbildung	39
20.12	Grenzen der Achsen	39
20.13	Plotstile	39
20.14	Höhen-Breitenverhältnis	39
20.15	Höhen-Breitenverhältnis	40
20.16	Statistik	40
20.17	Statistik 2	41
20.18	Verschiebung der Achsenbeschriftungen A	41
20.19	Verschiebung der Achsenbeschriftungen B	41
20.20	Achsen teilweise oder vollständig ausblenden	42
20.21	Schraffierte Flächen	42
20.22	Gefüllte Plots	43
20.23	Pixelanzeige einer Matrix	43
20.24	2D-Flächenplot	43
20.25	Plot nach DIN	44
21	Anwendungen	46
21.1	Lineare Regression mit Pandas	46
21.2	Impulsplot	47
21.3	Mehrere Plots übereinander	48
21.4	Pixel-Farbnetz	49
21.5	Benutzerspezifische Achsenbeschriftung	50
22	PDF-Dateierzeugung	51
22.1	PDF-Dateiausgabe	51
23	Pandas	52
23.1	Tabellenbreite in Pandas	52
23.2	Mehrzeilenstring als Datenquelle	52
23.3	Spalten kombinieren	52

24 Konsolenanwendungen	53
24.1 Benutzereingaben in der Konsole	53
25 GUIs	53
25.1 MessageBox	53
26 XML-Dateien	53
26.1 Elemente finden	53
26.2 Subelemente ermitteln	54
27 Interaktive Plots	54
27.1 Interaktive Plots mit iPython	54

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

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/>

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

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

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

1.2 Kommentare

Kommentare werden mit einem Lattenkreuz gekennzeichnet

```
1 # Dies ist ein Kommentar
```

1.3 Mehrzeilenkommentare

Kommentare über mehrere Zeilen werden mit drei Kochkommas gekennzeichnet:

1.4 Variablentypen ermitteln

Variablen gehören einer Klasse an, die ermittelt werden kann

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

1.5 Einzelne Variablen löschen

Einzelne Variablen können gelöscht werden

```
1 x = 42.3
2 y = 32.1
3 del x, y # Löscht die Variablen
```

2 Zahlen-Operationen

2.1 Komplexe Zahlen

Eine komplexe Zahl wird mit einem angehangenen j gekennzeichnet. Beispiel: $a = 3 + j4$

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

Euler-Form:

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

2.2 Real- und Imaginärteil extrahieren

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

2.3 Quadrieren

3^2

```
1 3**2
```

2.4 Ganzzahlige Division

Division ohne Rest. Beispiel: $14 \div 4 = 3$ Rest 2

```
1 14//4
```

2.5 Modulo-Operator (Divisionsrest)

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

```
1 rest = 14%5
```

Alternativ

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

2.6 Aufrunden

Aufrunden auf die nächste Ganzzahl

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

2.7 Abrunden

Aufrunden auf die nächste Ganzzahl

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

2.8 Vergleiche Float und Integer

Bei Zahlenvergleichen wird gerundet:

```
1 1.0000000000000001==1 # True
2 1.0000000000000001==1 # False
```

3 Kontrollstrukturen

3.1 Funktionen

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

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


3.2 Fallunterscheidung mit if

Fallunterscheidungen:

```

1 a = 42
2 if 10 < a:
3     print('10 < a')
4 elif 10 == a:
5     print('10 == a')
6 else:
7     print('a < 10')
```

3.3 Fallunterscheidung mit if (ternary operator)

Fallunterscheidungen:

```

1 a = 42
2 if 10 < a:
3     print('10 < a')
4 elif 10 == a:
5     print('10 == a')
6 else:
7     print('a < 10')
```

3.4 for Schleife

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

```

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

3.5 Boolsche Ausdrücke

Boolsche Ausdrücke: $(a \leq b)$ and $(c \neq d)$ or $(e == f)$

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

```

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

ergibt:

```

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

4 Objektorientierung

4.1 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.

```

1 class Kumpel:
2     alter = 18 # Klassenvariable
3     # Methoden:
4     def gruss(self):
5         print('Glueck_Auf!')
6     pass

```

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

```

1 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):

```

1 class Kumpel:
2     # Konstruktor:
3     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_=_\n'
6             + self.Nachname)
7     pass
8
9 Maria = Kumpel('Schulz')
10 print(Maria) # 'Schulz'

```

5 Listen/Arrays/Datenstrukturen

Siehe auch Listen und Arrays bei Numpy im Abschnitt xxxx

5.1 Range

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

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

6 Datenstrukturen

6.1 Dictionarys

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

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

Dictionarys können auch in Listen zusammengefasst sein:

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

Dictionary erweitern:

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

6.2 Tupel

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

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

6.3 Set

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

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

6.4 Arrays gleicher Länge zu Tupeln

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

```
1 A = [1, 2, 3]
2 B = [10, 9, 8]
3 C = zip(A, B)
4 print(list(C))
5 # [(1, 10), (2, 9), (3, 8)]
```

6.5 Arrays verbinden

Aneinanderhängen von Listen zu einer Gesamt-Liste

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

6.6 Länge eines Arrays

Länge eines Arrays

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

6.7 deep und shallow copy

Das Erzeugen einer neuen Variablen erzeugt nicht zwangsläufig ein neues unabhängiges 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 verzweigten Objekte dupliziert. Beispiel:

```
1 import copy
2 z = 666
3 A = [z, 2, 3]
4 B = A # zusätzliche Variable fuer gleiches Objekt
5 B[1]=42
6 print(A) # ergibt [42, 2, 3]
7 print(B) # ergibt [42, 2, 3]
8 print((id(A), id(B), id(A[0]), id(B[0])))
9
10 A = [z, 2, 3]
11 B = copy.copy(A) # shallow copy
12 B[1]=42
13 print(A) # ergibt [42, 2, 3]
14 print(B) # ergibt [42, 2, 3]
15 print((id(A), id(B), id(A[0]), id(B[0])))
16
17 A = [z, 2, 3]
18 B = copy.deepcopy(A)
19 B[0]=42
20 print(A) # ergibt [1, 2, 3]
21 print(B) # ergibt [42, 2, 3]
22 print((id(A), id(B), id(A[0]), id(B[0])))
```

7 Strings (Zeichenketten)

7.1 Strings aneinanderhängen

Eine list von Strings zu einem einzigen String aneinanderhängen:

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

Alternativ:

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

7.2 Strings mit Trennstring aneinanderhängen

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

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

7.3 Strings formatieren

Variablen und Textbausteine zu einem String zusammenbauen:

```
1 x = 5
2 y = 7
3 neustr = f"x={x}, y={y}"
4 # ergibt "x = 5, y = 7"
```

7.4 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“.

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

7.5 Substring finden

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

```
1 pos = "Hallo_Welt".find("l")
```

7.6 String in Int umformen

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

7.7 Int in String umformen

```
1 zahl = str(123)
```

7.8 float in String umformen

```
1 x=13.1415
2 e = 'pi={:f}'.format(x)
3 s = 'pi={0:1.2f}'.format(x)
4 print(e) # 'pi = 13.141500'
5 print(s) # 'pi = 13.14'
6 # Integer-Wert
7 print('Wert: {0:d}'.format(27828))
8 # führende Nullen:
9 print('{:03d}'.format(7)) # 007
10 # Scientific (nicht Eng.) Format:
11 print(f'{0.000002342:.3E}') # 2.342E-6
12 print(f'{0.000002342:E}') # 2.342000E-6
```

8 Jupyter Lab

8.1 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 `jupyterlabopenwith.bat` hat folgenden Inhalt (UN steht für den Benutzernamen unter dem die Anaconda-Distribution zu finden ist):

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

Die Batch-Datei wird in einem leicht zu merkenden Verzeichnis gespeichert. Mit dem Kontext-Menü „Öffnen mit...” einer .ipynb-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.

8.2 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 Kommandozeile 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.

8.3 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

8.4 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`

8.5 interact

Interact ermöglicht interaktive Berechnungen mit Auswahlfeldern und Zeigern:

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

Funktionen mit mehreren Parametern:

```
1 L = ['eins', 'zwei']
2 def f(A, B):
3     print(A)
4     print(B)
5     pass
6 # Laesst den Benutzer fuer A einen
7 # Wert aus der Liste L auswaehlen:
8 interact(f, A=L, B=(0,2.0,0.1))
9 # Uebergibt die gesamte Liste L an
10 # den Parameter A:
11 interact(f,A=fixed(L), B=(0,2.0,0.1))
```

8.6 Flickern interaktiver Grafiken verhindern

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

```
1 from ipywidgets import interact, interactive, fixed, interact_manual
2 import ipywidgets as widgets
3 %matplotlib inline
4 import matplotlib.pyplot as plt
5 import numpy as np
6 from IPython.display import clear_output
7 import math
8 def f(x=0):
9     fig, ax = plt.subplots()
10    fig.set_size_inches(120/25.4, 80/25.4)
11    wanderpunkt = plt.Circle((np.cos(x*2*math.pi), 0), 0.1)
12    ax.add_artist(wanderpunkt)
13    ax.set_xlim([-1.2, 1.2])
14    ax.set_ylim([-0.8, 0.8])
15    clear_output(wait=True)
16 interact(f, x=(0.0, 1.0, 0.01))
```

8.7 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):

```
1 from ipywidgets import interact, interactive, fixed, interact_manual
2 import ipywidgets as widgets
3 %matplotlib inline
4 import matplotlib.pyplot as mpl
5 from IPython.display import clear_output
6 mpl.rcParams['figure.dpi'] = 150
```

8.8 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:

```
1 from IPython.display import display, Markdown, Latex
2 display(Markdown('*hervorgehobener Text*\n\n'+
3               'Formel: $\sin(2\pi f t)$\n'+
4               '#Ueberschrift'\n'+
5               'Dies ist die erste Zeile\n'+
6               '|spalte1|spalte2|\n'+
7               '|-----|\n'+
8               '|3,54|3,24|\n'+
9               '|3,54|3,24|\n'+
10              '|-----|\n'+
11              '*AufzaehlungPunkt1*\n'+
12              '*AufzaehlungPunkt2*'))
```

Dies erzeugt folgende Ausgabe:

hervorgehobener Text

Formel: $\sin(2\pi ft)$

Ueberschrift

Dies ist die erste Zeile

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

- Aufzaehlung Punkt 1
- Aufzaehlung Punkt 2

8.9 Automatisiert erstellte Tabelle in Markdown

```
1 from IPython.display import display, Markdown, Latex
2 import numpy as np
3 mdstring = '|_spalte1_|_spalte2_|_\\n|_:-----|_:-----|_\\n'
4 for x in np.linspace(0,3.14, 4):
5     mdstring = mdstring + "{:1.4}|_{:1.4}\\n".format(x,x**2)
6 display(Markdown(mdstring))
7 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

9 VisualStudio Code (Jupyter Lab)

9.1 Installation

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

Informationen über den Kernel einholen:

```
1 import sys
2 print(sys.executable)
3 print(sys.version)
4 print(sys.version_info)
```

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

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

9.2 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

9.3 Neues Notebook anlegen

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

9.4 Mehrzeilige Formeln 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.

9.5 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 manuelles 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.

9.6 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.

10 VisualStudio Code (Python, iPython)

10.1 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.

10.2 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“

10.3 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.

10.4 Cursorposition Anzeigen

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

11 Dateien

11.1 Textdateien zeilenweise lesen

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

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

11.2 Textdateien zeilenweise schreiben

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

```
1 fd = open("Datei.txt", 'w') # 'a' fuer anhaengen/append
2 fd.write('Hallo_Welt\n')
3 fd.close()
```

11.3 Textdateien ganz lesen

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

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

11.4 Über alle Zeilen iterieren

Textdateien können ohne zusätzliche importierte Module gelesen werden:

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

12 SQLite-Datenbank

12.1 Daten abfragen

Daten aus einer SQLite-Datenbank abfragen:

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

13 Systemzugriff

13.1 Dateiname des Programms

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

```
1 import os
2 print(__file__) # ganzer Pfad
3 # Nur Dateiname (programm.py):
4 print(os.path.basename(__file__))
5 # Pfad des Programms:
6 print(os.path.dirname(__file__))
```

Mit Jupyter wird folgender Code zur Ermittlung des Programmnamens verwendet:

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

13.2 Sound/Wave/Audio ausgeben

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

```
1 """
2 import numpy as np
3 from IPython.display import Audio
4 from math import pi
5 fsample = 44000 # Abtastfrequenz
6 f1 = 440
7 f2 = 441
8 tmax = 5 #Maximaldauer
9 t = np.linspace(0,tmax,tmax*fsample)
10 wave_audio = 0.2*(np.cos(2*pi*f1*t)+np.cos(2*pi*f2*t))
11 Audio(wave_audio, rate=fsample)
```

13.3 Datum und Zeit

Ausgabe eines ISO-Zeitstempels"""

```
1 import datetime print(datetime.datetime.now(
2     datetime.timezone.utc).strftime("%Y-%m-%d_%H:%M_Uhr"))
3 # erzeugt 2024-10-30 20:15 Uhr
4 print(datetime.datetime.now().isoformat())
5 # erzeugt 2024-10-30T17:48:16.600919
```

14 Mathematische Funktionen

14.1 Fakultät (x!)

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

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

Beispiel:

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

```
1 import math
2 print(math.factorial(5)) # 5! = 120
3 # oder
4 import numpy as np
5 print(np.math.factorial(5))
```

14.2 arctan2

Der Winkel eines Zeigers auf eine Koordinate mit x- und y-Anteil, kann mit der Funktion `arctan2` berechnet werden. Diese funktioniert 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.

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

14.3 Exponentialfunktion

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

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

15 Numpy/Numerik

15.1 Numpy Array erzeugen

Numpy Array erzeugen aus einem normalen Array

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

15.2 Numpy Arrays verbinden

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

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

15.3 Numpy Matrix aus Vektoren, vertikal

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

```
1 import numpy as np
2 hv1 = np.array([11, 12, 13])
3 hv2 = np.array([21, 22, 23])
4 Mh = np.vstack((hv1, hv2))
5 # ergibt: [[11 12 13]
6           [21 22 23]]
```

15.4 Numpy Matrix aus Vektoren, horizontal

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

```
1 import numpy as np
2 v1 = [[11], [21], [31]]
3 v2 = [[12], [22], [32]]
4 # Argument als Tupel mit ( ... )
5 # Die Argumente werden automatisch
6 # zu np.arrays gemacht
7 Mh = np.hstack((v1, v2))
8 # ergibt: [[11 12]
9           [21 22]
10          [31 32]]
```

15.5 ndarray 1 D adressieren

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

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

15.6 ndarray 1 D Differenzen

Differenzen zwischen Werten in einem Numpy-Array berechnen:

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

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

15.7 Numpy arrays

Verschiedene Methoden zur Arrayerzeugung:

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

15.8 ndarray 2 D

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

```
1 import numpy as np
2 A = np.array([[11, 12, 13],\
3             [21, 22, 23]])
```

15.9 ndarray Größe

Gesamtzahl der Elemente in einem Array und Anzahl Zeilen und Spalten

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

15.10 ndarray transponieren

Ein Numpy-Array wird folgendermaßen transponiert::

```
1 import numpy as np
2 A = np.array([[11, 12, 13],\
3             [21, 22, 23]])
4 B = A.transpose()
```

15.11 Inverse Matrix (Matrix invertieren)

Eine quadratische Matrix wird wie folgt invertiert:

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


15.12 Pseudoinverse einer Matrix

Eine quadratische Matrix wird wie folgt invertiert:

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

15.13 ndarray umdrehen

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

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

15.14 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:

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

Dies ist auch mit komplexen Werten möglich. Das Gleichungssystem

$$\begin{pmatrix} 1+j5 & 3-j2 & -5 \\ 3+j2 & 0 & -4 \\ -5 & -4 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ -3 \\ -3 \end{pmatrix}$$

kann wie folgt gelöst werden:

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

15.15 Polynomdivision

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

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

```

1 import numpy as np
2 # Zaehlerkoeffizienten:
3 Z = np.array([2, -11, 18, -20, 56, -65])
4 # Nennerkoeffizienten:
5 N = np.array([1, -7, 16, -12])
6 E = np.polydiv(Z, N)
7 print(E) # ergibt (array([2., 3., 7.]), array([ 5., -20., 19.]))
8 def pl(kp):
9     if kp == 0:
10         return("")
11     else:
12         return("␣+␣")
13 A = "".join([pl(k)+"("+str(E[0][k])+"*x^"+str(len(E[0])-k-1) for k in range(len(E[0]))])
14 B = "".join([pl(k)+"("+str(E[1][k])+"*x^"+str(len(E[1])-k-1) for k in range(len(E[1]))])
15 C = "".join([pl(k)+"("+str(N[k])+"*x^"+str(len(N)-k-1) for k in range(len(N))])
16 print( A + "␣+␣(" + B + ") / (" + C + ")" )

```

Ergebnis:

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

15.16 Integrale (Trapezregel)

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

```

1 import numpy as np
2 import scipy.integrate
3 t = np.linspace(0,3.14159265,10000)
4 x = np.cos(t)
5 y = np.sin(t)
6 # Achtung: zuerst y, dann x:
7 A = scipy.integrate.trapz(y,x)
8 print(A*2) # = -pi
9 # -pi, weil die x-werte absteigend

```

15.17 Potenz einer Matrix

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

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

```

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

```

15.18 Integral einer Funktion

Wenn eine Funktion gegeben ist, kann mit numerisch das Integral berechnet werden: Beispiel

$$\int_{x=0}^4 x^2 dx$$

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

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

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

15.19 Zweidimensionales Integral

Mehrfachintegral am Beispiel:

$$R = \int_{a=2}^5 \int_{b=\sin(a)}^{a^2} a \cdot b db da$$

```
1 import numpy as np
2 from scipy import integrate
3
4 def b unten(apar):
5     return(np.sin(apar))
6
7 def oben(apar):
8     return(apar**2)
9
10 # der erste Parameter ist die Integrationsvariable des inneren Integrals
11 def ing(bpar, apar):
12     return(apar*bpar)
13
14 R = integrate.dblquad(ing, 2, 5, unten,
15     oben, epsabs=1.5e-8, epsrel=1.5e-8)[0]
16 print(R) # ergibt 1293.96
```

15.20 Integral einer komplexwertigen Funktion

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

```
1 import numpy as np
2 import math
3 from scipy import integrate
4 y_re=lambda x: np.real(\
5     np.exp(1j*math.pi*x))
6 y_im=lambda x: np.imag(\
7     np.exp(1j*math.pi*x))
8 z = integrate.quad(y_re,0,2)[0]+\
9     1j*integrate.quad(y_im,0,2)[0]
10 print(z)
```

15.21 Dreifachintegral

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

```

1 from scipy import integrate
2 from math import pi, cos
3 import numpy as np
4 def integrant(r, phi, z):
5     return(r**3)*7
6 E,F=integrate.tplquad(integrant,0,9,\
7     lambda z:-pi/6, lambda z: pi/6,\
8     lambda z, phi: cos(phi),\
9     lambda z, phi: 7)
10 print(E) # Ergebnis
11 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 \leftrightarrow Letzte beiden Argumente der Funktion `tplquad`

15.22 DGL 1. Ordnung numerisch lösen

Lösung der DGL $y' = 2ty + 5t$

```

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

```

15.22.1 Faktorisierung eines Polynoms

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

```

1 from sympy import *
2 x = symbols('x')
3 y = factor(x**4+4*x**2)
4 print(y)

```

15.22.2 Einheitsmatrix

Einheitsmatrix

```

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

```

15.22.3 Determinante

Berechnung der Determinante einer Matrix

```

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

```

15.22.4 Elementmanipulation

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

```

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

```

15.22.5 Matrizenmultiplikation

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

```

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

```

15.22.6 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 aneinandergehängt:

```

1 import numpy as np
2 a = np.array([[11, 12],\
3              [21, 22], [31, 32]])
4 b = np.array([[13], [23], [33]])
5 print(np.c_[a,b])

```

Ergebnis:

```

1 [[11 12 13]
2  [21 22 23]
3  [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 aneinandergelagert:

```

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

```

Ergebnis:

```

1 [[11 12]
2  [21 22]
3  [31 32]]

```

15.22.7 XY-Matrizen

Im Bereich 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.

```

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

```

Ergebnisse:

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

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

15.22.8 Zufallszahlen

Integer Zufallszahl zwischen 1 und 90

```

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

```

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

```

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

```

Reelle Zufallszahl zwischen 0,0 und 1,0

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

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

```
1 import numpy as np
2 np.random.seed(0)
3 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.

15.22.9 lineare Differenzialgleichung n-ter Ordnung lösen

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

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

Darin ist $y(t)$ die gesuchte Funktion und $g(t)$ das Störglied.

Diese DGL wird zunächst in Systemdarstellung überführt:

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

Darin ist der grün eingzeichnete 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

$$\underbrace{\begin{pmatrix} 0 & 1 \\ -3 & -4 \end{pmatrix}}_A \underbrace{\begin{pmatrix} y(t) \\ y'(t) \end{pmatrix}}_y + \underbrace{\begin{pmatrix} 0 \\ 200 \sin(6\pi t) \end{pmatrix}}_b$$

gezeigt:

```
1 from scipy.integrate import solve_ivp
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import os
5
6 def dydt(tp, yT):
7     A = np.array([[0, 1], [-3, -4]])
8     b = np.array([[0], [200*np.sin(6*pi*tp)]])
9     retval = ((np.array([yT@A.transpose()])).transpose()+b).transpose()[0]
10    return(retval)
```

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

Es gilt $(A y)^T = y^T A^T \Rightarrow A y = (y^T A^T)^T$. Darin ist y^T der Zeilenvektor y^T 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 Funktionswerten ausgegeben. Der Parameter `rtol` ist standardmäßig $1e-3$ und beschreibt die relative Genauigkeit des Löser.

```

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

```

15.22.10 nichtlineare Differenzialgleichung n-ter Ordnung

Gegeben sei die nichtlineare DGL $y''(t) = -1/(t + y(t)^2)$. 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.

```

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

```



```

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

```

15.23 Werteliste nahe Null zu Null runden

Werte in der Nähe zu Null zu Null runden

```

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

```

15.24 Abschnittsweise definierte Funktionen

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

```

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

```

15.25 Heaviside-Funktion (Sprungfunktion)

Die Heaviside- oder Sprungfunktion ist wie folgt definiert:

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

```

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

```

16 Symbolische Mathematik

16.1 Brüche

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

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

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

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

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

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

16.2 Terme rational machen

Ausdrücke in einen einzelnen Bruch umformen

```
1 import sympy
2 x = sympy.symbols('x')
3 f = (-2*x+3)/((x-1)**2+2**2)+(5)/(x-3)
4 print(sympy.ratsimp(f))
5 # ergibt:
6 # (3*x**2-x+16)/(x**3-5*x**2+11*x-15)
```

17 Komplexe Zahlen

Komplexe Zahlen können mit dem Basis-Python verwendet werden:

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

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

```
1 from sympy import Rational
2 print(4+Rational(1,3)*1j)
3 # 4 + 0.333333333333333*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:

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

17.1 Ableitungen

Ableitung von Funktionen berechnen

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

17.2 Integrieren

Symbolisch ein bestimmtes Integral bilden

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

Symbolisch ein unbestimmtes Integral bilden:

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

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

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

17.3 Ausmultiplizieren

Ausmultiplizieren (engl. expand) von Ausdrücken

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

17.4 Ersetzungen

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

```
1 from sympy import *
2 x = symbols('x')
3 g = -6*x**3 + 16*x**2 - 84*x+64
4 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:

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

17.5 Numerische Auswertung

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

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

17.6 Evaluation verhindern

Gelegentlich will man eine Auswertung eines Ausdrucks verhindern:

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

Abhilfe:

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

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

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

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

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

17.7 Vereinfachungen

Vereinfachungen können mit `simplify` erfolgen:

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

17.8 Automatische Vereinfachung

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

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

17.9 Ausklammern

Ausdrücke können wie folgt ausgeklammert werden:

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

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

17.10 Vereinfachung von Rationalen Funktionen (Polynombrüche)

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

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

17.11 Partialbruchzerlegung

Funktion zur Partialbruchzerlegung:

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

17.12 Ausmultiplizieren von Partialbrüchen

Ausmultiplizieren der linken Seite der folgenden Gleichung ergibt die rechte:

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

```

1 from sympy import *
2 s = symbols('s')
3 expr = (Rational(5,3)*Poly(1,s)/Poly(s+1,s)
4        -2*Poly(1,s)/Poly(s+2,s)
5        + Rational(1,3)*Poly(1,s)/Poly(s-2,s))
6 pprint(ratsimp(expr))

```

17.13 Ausgabe in Formel-Schreibweise

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

```

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

```

Ergibt:

```

1      2*t
2  -----
3  4      2
4  t  - 2*t  + 1

```

17.14 Exakte Lösungen linearer Gleichungssysteme

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:

```

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

```

Ergibt:

```

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

```

17.15 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:

```
1 from math import pi
2 from sympy import *
3 L1, L2, L3 = symbols('L1, L2, L3')
4 Mat = Matrix([[L1, 0, L3],
5               [L1, L2, 0],
6               [0, -L2, 0]
7               ])
8 Mat.inv()
```

Ergibt:

```
1 Matrix([
2 [ 0, 1/L1, 1/L1],
3 [ 0, 0, -1/L2],
4 [1/L3, -1/L3, -1/L3]])
```

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

```
1 import re #regular expressions
2 s1 = "magicwoche_12"
3 p = re.compile('magicwoche_([0-9]+)')
4 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
<code>\s</code>	Whitespace
<code>\S</code>	alles außer Whitespace
<code>[(]</code>	eine geöffnete runde Klammer
<code>[a-z]</code>	ein kleiner Buchstabe
<code>[0-9]</code>	eine Ziffer

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

19.3 Substrings ersetzen

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

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

In diesem Beispiel 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.

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

20 Matplotlib

20.1 Einfachster Plot

```
1 t=[k for k in range(0,round(70/0.2))]
2 tbl=[1/(1+k*0.2) for k in t]
3 import matplotlib
4 matplotlib.pyplot.plot(t,tbl)
5 matplotlib.pyplot.grid()
```

20.2 Schriftart Stix

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

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 plt.rcParams['font.family']=\
4     'STIXGeneral'
5 plt.rcParams['mathtext.fontset']=\
6     'stix'
7 x = np.linspace(0, 12, 150)
8 fig, ax = plt.subplots()
9 fig.set_size_inches(80/25.4,50/25.4)
10 g1, = ax.plot(x, np.sin(x))
11 plt.grid()
12 plt.xlabel('$x$')
13 plt.ylabel('$y$')
14 plt.subplots_adjust(left=0.22,\
15 right=0.97, top=0.97, bottom=0.22)
16 plt.savefig('bilddatei.pdf')
```

20.3 Ausgabe verfügbarer Schriftarten

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

```
1 import matplotlib.font_manager
2 fpaths = matplotlib.font_manager.findSystemFonts()
3
4 for i in fpaths:
5     f = matplotlib.font_manager.get_font(i)
6     print(f.family_name)
```

20.4 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):

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 x = np.linspace(-15, 15, 150)
4 fig, ax = plt.subplots()
5 fig.set_size_inches(120/25.4, 80/25.4)
6 g1, = ax.plot(x, np.sin(x))
7 plt.savefig('bilddatei.png', dpi=300)
```

20.5 Universal-Plot

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 a = 0.3
4 wc = 5
5 wk = np.linspace(-15, 15, 500)
6 A, B, C, D = -1/2, wc/2, -1/2, -wc/2
7 impt3 = (A*wk+B)/((wk-wc)**2 + a**2)
8 impt4 = (C*wk+D)/((wk+wc)**2 + a**2)
9 fig, ax = plt.subplots()
10 fig.set_size_inches(120/25.4, 80/25.4)
11 g1, = ax.plot(wk, impt3, 'g-', \
12             lw=5, label=r'$\frac{1}{2}$')
13 g2, = ax.plot(wk, impt4, 'r-', \
14             label=r'$\omega_{\mathrm{k}}$')
15 # fuer Legende: Komma nach g1 und g2
16 # nicht vergessen beim Plot-Befehl!
17 plt.legend(handles=[g1, g2])
18 plt.xticks(np.arange(-2, 13, step=1))
19 plt.yticks(np.arange(0, 6, step=1))
20 ax.set_xlim([-15, 15])
21 ax.set_ylim([-1, 1])
22 plt.grid()
23 plt.xlabel('$\omega$')
24 plt.ylabel('Imaginaerteil')
25 plt.subplots_adjust(left=0.17, \
26                   right=0.97, top=0.97, bottom=0.15)
27 plt.savefig('bilddatei.pdf')
```

20.6 Kommas statt Punkte

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

```
1 import matplotlib.pyplot as plt
2 import matplotlib.ticker as tkr
3
4 def func(x, pos): # formatter
5     s1 = '{:1.1f}'.format(x)
6     s2 = s1.replace('.', ',')
7     return s2
8 y_frmt = tkr.FuncFormatter(func)
9
10 t = [k for k in range(10)]
```

```

11 y = [1/(k+1) for k in t]
12 fig, ax = plt.subplots()
13 p1 = ax.plot(t, y)
14 ax.yaxis.set_major_formatter(y_frmt)
15 plt.show()

```

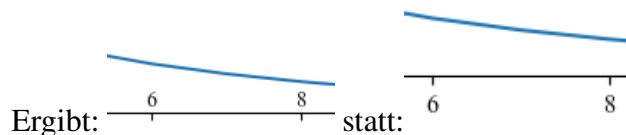
20.7 Achsenzahlen verschieben

Die Achsenzahlen ('ticks') können von der Achse weg oder zur Achse und darüber hinaus verschoben werden.

```

1 import matplotlib.pyplot as plt
2 t = [k for k in range(10)]
3 y = [1/(k+1) for k in t]
4 fig, ax = plt.subplots()
5 p1 = ax.plot(t, y)
6 ax.xaxis.set_tick_params(pad=-15)
7 plt.show()

```



20.8 Positionen Hilfsgitterlinien

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

```

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

```

20.9 Beschriftungspfeile

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

```

1 plt.annotate("", xy=(-1,2.3),
2 xytext=(0,2.3),\
3 arrowprops=dict(arrowstyle="|-|",\
4 linewidth=1.3, color='k',shrinkA=0,\
5 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.

20.10 Beschriftungstext

In der Grafik kann Text als Beschriftung eingefügt werden.

```
1 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.

20.11 Größe der Abbildung

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

```
1 fig, ax = plt.subplots()
2 ...
3 fig.set_dpi(150)
```

je größer die dpi-zahl ist, desto größer wird die Abbildung angezeigt.

20.12 Grenzen der Achsen

Die angezeigten Zahlenwerte der Achsen können folgendermaßen geändert werden:

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

20.13 Plotstile

Linien können unterschiedlich gestaltet sein:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 fig, ax = plt.subplots()
4 x = np.linspace(0,10,100)
5 ax.plot(x,np.sin(x),'g-')
6 # 'g-' ist die Linien-Stildefinition
```

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

20.14 Höhen-Breitenverhältnis

Gelegentlich müssen Achsen gleichskaliert sein. 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.

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 fig, ax = plt.subplots()
4 x = np.linspace(0,10,100)
5 ax.plot(x,np.sin(x),'g-')
6 ax.set_aspect('equal')

```

20.15 Höhen-Breitenverhältnis

Gelegentlich müssen Achsen gleichskaliert sein. 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.

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 fig, ax = plt.subplots()
4 x = np.linspace(0,10,100)
5 ax.plot(x,np.sin(x),'g-')
6 ax.set_aspect('equal')

```

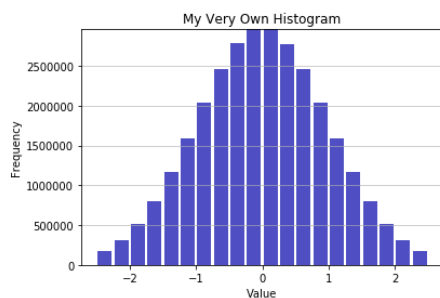
20.16 Statistik

Einfaches Statistik-Beispiel:

```

1 import numpy as np
2 import random
3 import matplotlib.pyplot as plt
4 Nx = 30000000;
5 x = np.random.randn(Nx)
6 hist, bin_edges = np.histogram(x)
7 n, bins, patches = plt.hist(x=x,\
8     bins=np.arange(-2.5,2.75,0.25),\
9     color='#0504aa', alpha=0.7, rwidth=0.85)
10 plt.grid(axis='y', alpha=0.75)
11 plt.xlabel('Value')
12 plt.ylabel('Frequency')
13 plt.title('My_Very_Own_Histogram')
14 maxfreq = n.max()
15 # Set a clean upper y-axis limit.
16 plt.ylim(ymax=np.ceil(maxfreq / 10)\
17     * 10 if maxfreq % 10 \
18     else maxfreq + 10)

```



20.17 Statistik 2

Beispiel zum Thema Statistik.

```

1 grsse = 3
2 chiv = np.ones(Nx//grsse)
3 for k in range(Nx//grsse):
4     for m in range(grsse):
5         chiv[k] = chiv[k]+\
6             x[k*grsse+m]**2
7 hist, bin_edges = np.histogram(chiv)
8 n, bins, patches = plt.hist(x=chiv,\
9     bins=np.arange(0,15,0.1),\
10    color='#0504aa', alpha=0.7, rwidth=1.0)
11 plt.grid(axis='y', alpha=0.75)
12 plt.xlabel('Value')
13 plt.ylabel('Frequency')
14 plt.title('My_Very_Own_Histogram')
15 maxf = n.max()
16 # Set a clean upper y-axis limit.
17 plt.ylim(ymax=np.ceil(maxf / 10)\
18 * 10 if maxf % 10 else maxf + 10)

```



20.18 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.

```

1 import matplotlib.pyplot as plt
2 t = [k for k in range(10)]
3 y = [1/(k+1) for k in t]
4 fig, ax = plt.subplots()
5 p1 = ax.plot(t, y)
6 plt.xlabel('xlbl', labelpad=0)
7 plt.ylabel('ybl', labelpad=0)
8 plt.show()

```

20.19 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

```

1 import matplotlib.pyplot as plt
2 t = [k for k in range(10)]
3 y = [1/(k+1) for k in t]
4 fig, ax = plt.subplots()
5 p1 = ax.plot(t, y)
6 ax.set_xlabel('xlbl')
7 ax.xaxis.set_label_coords(0.95, 0.45)
8 ax.set_ylabel('ybl', rotation=0, ha='left')
9 # Rotation = drehung in Grad
10 # ha = horizontal alignment (Ausrichtung)
11 ax.yaxis.set_label_coords(0.28, 0.95)
12 plt.show()

```

20.20 Achsen teilweise oder vollständig ausblenden

Die Anzeige der Achsen kann teilweise oder vollständig ausgeschaltet werden:

```

1 import matplotlib.pyplot as mpl
2 ...
3 ax1.spines['top'].set_color('none')
4 ax1.spines['right'].set_color('none')
5 ax1.spines['bottom'].set_color('none')
6 ax1.spines['left'].set_color('none')
7 #Alle Achsenteile ausblenden:
8 mpl.axis('off')

```

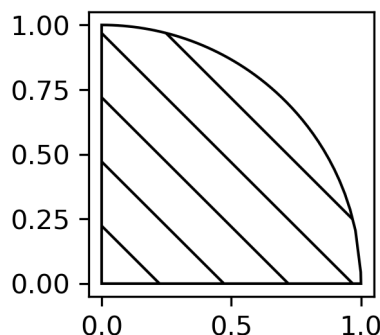
20.21 Schraffierte Flächen

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

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 fig, ax= plt.subplots()
4 fig.set_size_inches(50/25.4, 50/25.4)
5 xo=np.linspace(0, 0.999, 50)
6 x=np.append([1,0],xo)
7 y=np.append([0,0],np.sqrt(1-xo**2))
8 ax.fill(x,y, fill=False, hatch='\\\\')
9 plt.subplots_adjust(left=0.22,\
10     right=0.97,top=0.97, bottom=0.22)
11 ax.set_aspect('equal')
12 fig.show()

```



20.22 Gefüllte Plots

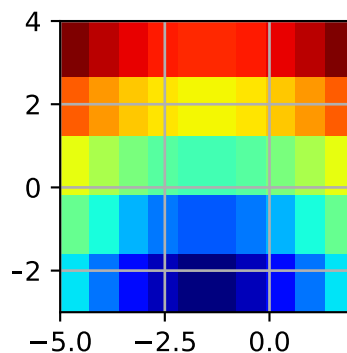
Mit dem Befehl `fill` statt `plot` können Flächen unter Kurven ausgefüllt werden:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 fig, ax= plt.subplots()
4 xorig=np.linspace(0,1,30)
5 x=np.append([0], xorig)
6 y=np.append([0], np.sqrt(1-xorig**2))
7 ax.fill(x,y)
8 ax.plot(x,y,'k-')
9 fig.show()
```

20.23 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 einzelnen Matrixelemente wird in eine Farbe umgerechnet. Die Umrechnungsformel wird mit dem Parameter `cmap` angegeben. Einige mögliche sind: `'jet'`, `'gist_earth'` oder `'gnuplot'`

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 #linspace in y Richtung abfallend
4 xv=np.linspace(-2,2,10)
5 yv=np.linspace(4,1,5)
6 Mx, My = np.meshgrid(xv, yv)
7 Mz = np.sqrt(Mx**2+My**2)
8 fig, ax= plt.subplots()
9 fig.set_size_inches(50/25.4, 50/25.4)
10 ax.imshow(Mz, extent=[-5, 2, -3, 4],\
11     aspect='auto', cmap='jet',\
12     interpolation = 'none')
13 ax.grid(lw=1)
```



20.24 2D-Flächenplot

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

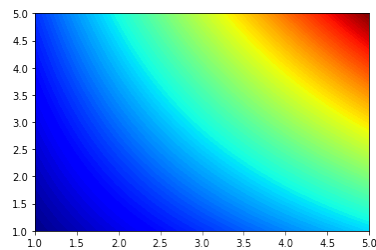
```
1 phi00 = 1
2 phi01 = 2
3 phi10 = 3
```



```

4 phi11 = 7
5 x0 = 1
6 x1 = 5
7 y0 = 1
8 y1 = 5
9
10 def phixy(xp, yp):
11     retval = 1/((y1-y0)*(x1-x0)) * (((x1-xp)*phi00 + (xp-x0)*phi10)
12     * (y1-yp) + ((x1-xp)*phi01 + (xp-x0)*phi11) * (yp-y0))
13     return(retval)
14
15 import numpy as np
16 import matplotlib.pyplot as plt
17 from matplotlib import cm
18
19 xv=np.linspace(x0,x1,15)
20 yv=np.linspace(y0,y1,15)
21 Mx,My=np.meshgrid(xv,yv)
22 phi = phixy(Mx,My)
23 plt, ax = plt.subplots()
24 surf = ax.contourf(Mx, My, phi, 100, cmap=cm.jet)
25 plt.show()

```



20.25 Plot nach DIN

Das nachfolgende Programm erzeugt einen Funktionsplot nach DIN-Norm:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import math
4 from matplotlib import rc
5 rc('text', usetex=True)
6 rc('legend',**{'fontsize':10})
7 t0 = 0.0
8 t1 = 1.5
9 steps = 100
10 xvals = np.linspace(t0,t1,steps)
11 yvals1 = xvals**2
12 yvals2 = np.sin(xvals*2*math.pi)
13 fig = plt.figure()
14 #Hoehe, Breite und Position des Plot
15 rect = 0.14, 0.17, 0.84, 0.80
16 ax1 = fig.add_axes(rect, frameon=True)
17 ax1.set_ylim([-2,2])
18 ax1.set_xlim([t0,t1])
19 stromplot = ax1.plot(xvals, yvals1,\
20     label="$x^2$", c='b', lw=0.7/0.3527)
21 sinusplot = ax1.plot(xvals, yvals2,\
22     label="$\\sin(x)$", color = 'b',\
23     ls='--', lw=0.7/0.3527)
24 ax1.set_xlabel('$t$')
25 ax1.xaxis.set_label_coords(0.55,\
26     0.07, transform=fig.transFigure)

```

```

27 ax1.set_ylabel('$i$')
28 ax1.yaxis.set_label_coords(0.05,\
29     0.5, transform=fig.transFigure)
30 legend=ax1.legend(\
31     loc='lower_left', shadow=False)
32 xls = 0.2
33 xachseneinheit_str = 's'
34 x=[ax1.get_xlim()[0]+k*xls for k in \
35     range(math.floor((ax1.get_xlim()[1]\
36     -ax1.get_xlim()[0])*\
37         (1.0+1e-12)/xls)+1)]
38 lbls=[("%.1f"%d).replace('.',',')\
39     for d in x]
40 lbls[-2]=xachseneinheit_str
41 plt.xticks(x, lbls)
42 # Y-Achse:
43 yachseneinheit_str = 'A'
44 yls = 0.5
45 y=[ax1.get_ylim()[0]+k*yls for k in \
46     range(math.floor((ax1.get_ylim()[1]\
47     -ax1.get_ylim()[0])*\
48         (1.0+1e-12)/yls)+1)]
49 lblsy=[("%.1f"%d).replace('.',',')\
50     for d in y]
51 lblsy[-2]=yachseneinheit_str
52 w35=0.35/0.3527
53 w18=0.18/0.3527
54 plt.yticks(y, lblsy)
55 ax1.spines['top'].set_color('none')
56 ax1.spines['right'].set_color('none')
57 ax1.spines['bottom']. \
58     set_position('zero')
59 ax1.spines['left'].set_linewidth(w35)
60 ax1.spines['bottom']. \
61     set_linewidth(w35)
62 print(ax1.xaxis.get_ticks_position())
63 ax1.xaxis.set_ticks_position('bottom')
64 ax1.grid(b=True, which='both', \
65     axis='x', c='k', ls='-', lw=w18)
66 ax1.grid(b=True, which='both', \
67     axis='y', c='k', ls='-', lw=w18)
68 fig.set_size_inches(100/25.4, 63/25.4)

```

```

1 dstaxtxpt=fig.get_size_inches()[1]*\
2     25.4*rect[3]/0.3527*(0-\
3     ax1.get_ylim()[0])/ \
4     (ax1.get_ylim()[1]-\
5     ax1.get_ylim()[0])+2
6 ax1.tick_params(axis='x', \
7     which='major', pad=dstaxtxpt)
8 # Achsenpfeil X:
9 b_mm=math.tan(7.5*math.pi/180)*20*0.35
10 arrowlen_mm = 12.0
11 deltax_frac = arrowlen_mm /\
12     (fig.get_size_inches()[0]*25.4)
13 arrowfrac = 10*0.35/arrowlen_mm
14 ybf = 0.048
15 xbf = 0.57
16 ax1.annotate('', \
17     xy=(xbf+deltax_frac, ybf), \
18     xycoords='figure_fraction', \
19     xytext=(xbf, ybf), \
20     textcoords='figure_fraction', \
21     arrowprops=dict(width=w35, \
22     frac=arrowfrac, facecolor='black', \
23     edgecolor='none', \

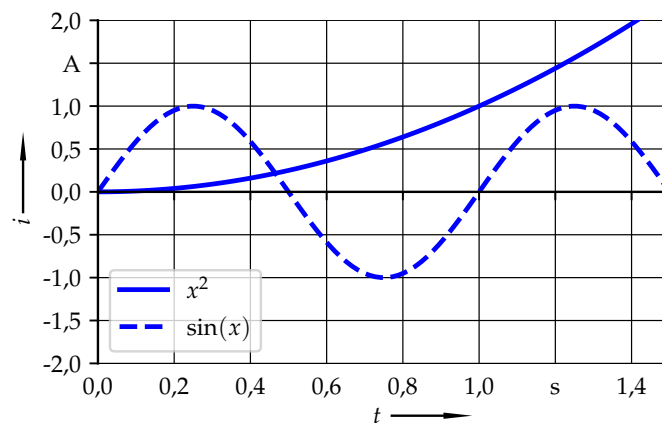
```

```

24     headwidth=b_mm/0.3527))
25 # Achsenpfeil Y:
26 deltax_frac = arrowlen_mm /\
27     (fig.get_size_inches()[1]*25.4)
28 ybyf = 0.52
29 xbyf = 0.03
30 ax1.annotate('',\
31     xy=(xbyf,ybyf+deltax_frac), \
32     xycoords='figure_fraction',\
33     xytext=(xbyf,ybyf),\
34     textcoords='figure_fraction',\
35     arrowprops=dict(width=w35,\
36         frac=arrowfrac,facecolor='black',\
37         edgecolor='none', \
38         headwidth=b_mm/0.3527))
39 plt.savefig('plot_din.pdf',\
40     format='pdf')
41 plt.savefig('plot_din.png',\
42     format='png',dpi=600)
43 plt.close("all")

```

Ausgabe:



21 Anwendungen

21.1 Lineare Regression mit Pandas

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

```

1 import numpy as np
2 import pandas as pd
3
4 xy = [\
5     (20, 0),\
6     (16, 3),\
7     (15, 7),\
8     (16, 4),\
9     (13, 6),\
10    (10, 10),\
11    ]
12 df = pd.DataFrame(xy, columns=['x', 'y'])
13 print(df)
14 # Mittelwert = Summe durch Anzahl

```

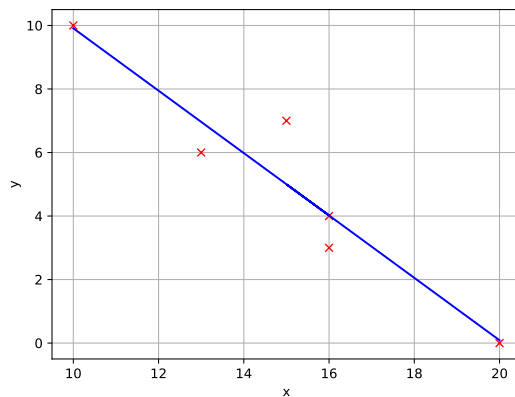
```

15 x_avg = df['x'].sum()/df['x'].count()
16 y_avg = df['y'].sum()/df['y'].count()
17 y_avg2 = df['y'].mean() #geht auch
18 # Erzeugung neuer Spalten:
19 df['Dx'] = df['x']-x_avg
20 df['Dy'] = df['y']-y_avg
21 df['Dx*Dy'] = df['Dx']*df['Dy']
22 df['Dx^2'] = df['Dx']**2
23 m = df['Dx*Dy'].sum()/df['Dx^2'].sum() # Steigung
24 b = y_avg-m*x_avg # Offset
25 print(df)
26
27 # Ergebnis plotten:
28 import matplotlib.pyplot as plt
29 fig, ax = plt.subplots()
30 ax.plot(df['x'],df['y'], 'rx')
31 ax.plot(df['x'],m*df['x']+b, 'b-')
32 plt.grid()
33 plt.xlabel('x')
34 plt.ylabel('y')
35 plt.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



21.2 Impulsplot

Ausgabe von Impulsplots:

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 xavg = 1.5
4 xsigma = 0.15
5 fig, ax = plt.subplots()
6 fig.set_size_inches(120/25.4,80/25.4)
7 np.random.seed(0)
8 x = np.random.normal(xavg, xsigma, 12)
9 y = 0.0*x+1

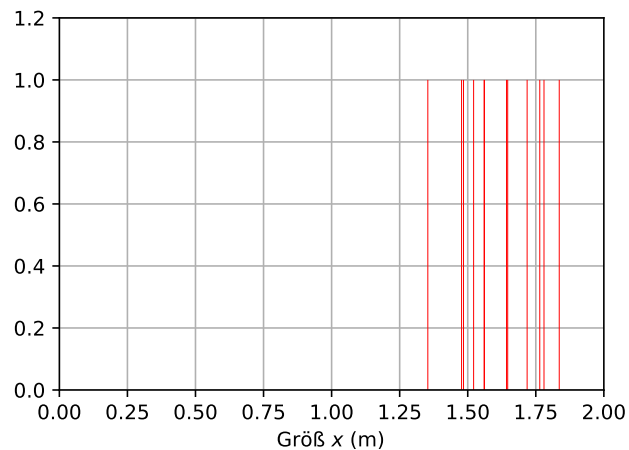
```

```

10 ml, sl, bl = ax.stem(x,y,linefmt='r-',markerfmt='_ ', basefmt='_ ')
11 plt.setp(sl, linewidth=0.5)
12 plt.setp(bl, color="none")
13 ax.grid()
14 ax.set_xlim([0,2])
15 ax.set_ylim([0,1.2])
16 plt.xlabel('Groesse_ $x$ (m)')
17 plt.subplots_adjust(left=0.17,\
18     right=0.97, top=0.97, bottom=0.15)
19 plt.savefig('bilddatei.pdf')
20 plt.show()

```

Ausgabe:



21.3 Mehrere Plots übereinander

Mehrere Plots übereinander:

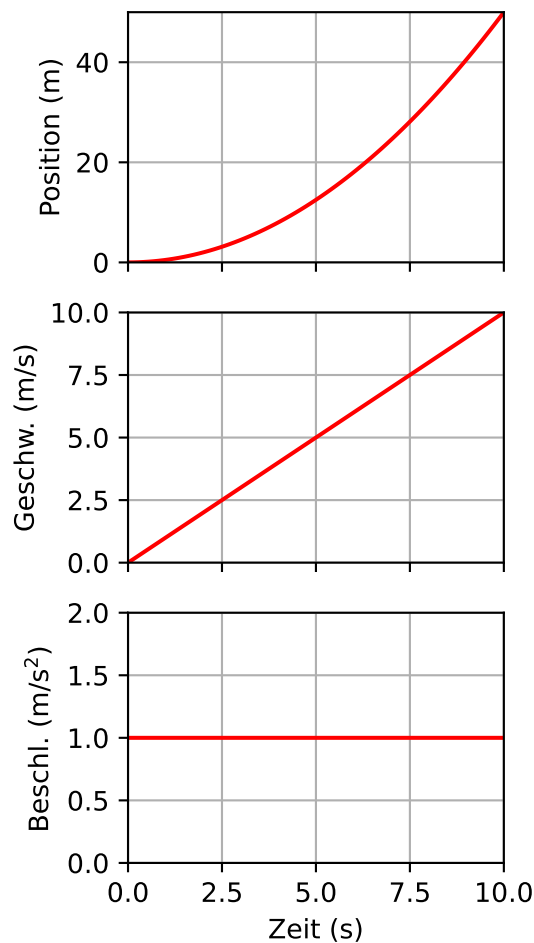
```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 t = np.linspace(0,10,75)
4 fig, axs = plt.subplots(3)
5 fig.set_size_inches(70/25.4,125/25.4)
6 axs[0].set_ylabel('Position_ (m)')
7 gxphase1, = axs[0].plot(t, 1/2*t**2, 'r-', label='besch')
8 axs[1].set_ylabel('Geschw_ (m/s)')
9 gvphase1, = axs[1].plot(t, t, 'r-')
10 axs[2].set_ylabel('Beschl_ $\mathrm{(m/s^2)}$')
11 gaphase1, = axs[2].plot(t, 1+0*t, 'r-')
12
13 axs[0].grid()
14 axs[1].grid()
15 axs[2].grid()
16
17 axs[0].set_xlim([0,10])
18 axs[0].set_xticklabels([])
19 axs[1].set_xlim([0,10])
20 axs[1].set_xticklabels([])
21 axs[2].set_xlim([0,10])
22 axs[0].set_ylim([0,50])
23 axs[1].set_ylim([0,10])
24 axs[2].set_ylim([0,2])
25
26 axs[2].set_xlabel('Zeit_ (s)')
27 plt.subplots_adjust(left=0.23, right=0.94, top=0.99, bottom=0.09, wspace=0.1, hspace=0.2)
28 fig.show()

```

```
29 fig.savefig('x-v-a-diagramm.pdf')
```

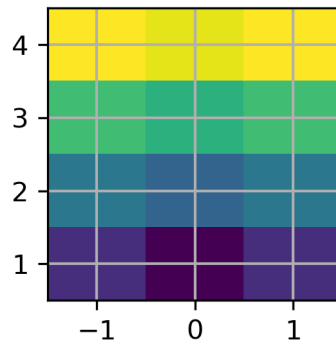
Ausgabe:



21.4 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.

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



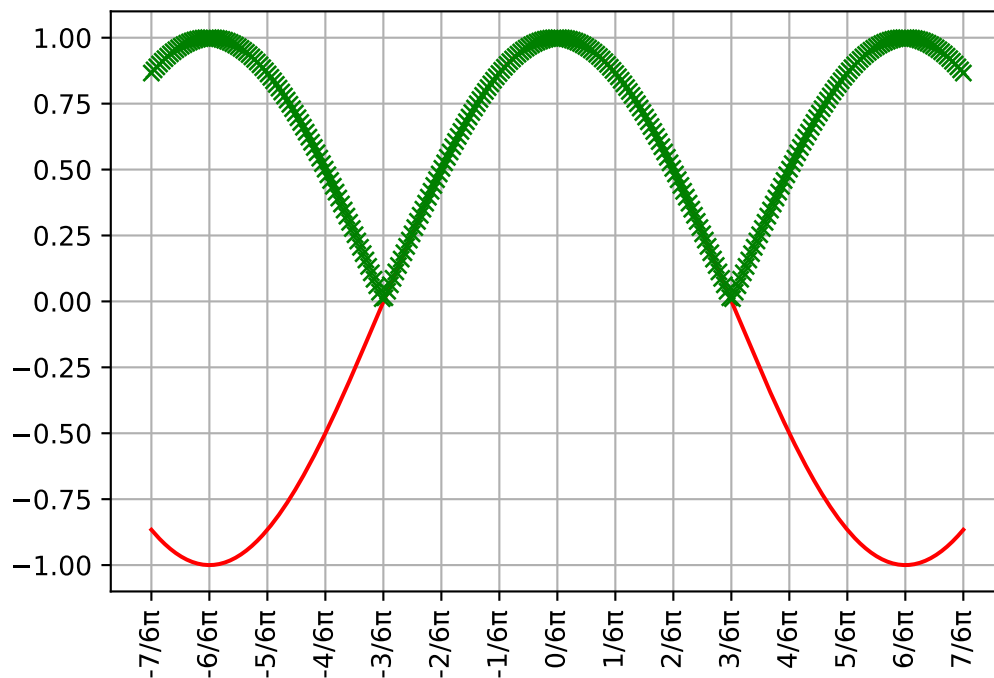
21.5 Benutzerspezifische Achsenbeschriftung

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

```

1  """
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from numpy import *
5  from math import pi
6  import os
7
8  fig, ax = plt.subplots()
9  p = np.array([p for p in range(-7,7+1)])
10 t = p/6*pi
11 lbl = [str(k)+''/6pi' for k in p] # Hier pi durch
12 # das Unicode-Pi ersetzen.
13 # ax.plot(t, il_von_t, 'g-')
14 thighres = np.linspace(-7/6*pi,7/6*pi,300)
15 ax.plot(thighres, cos(thighres), 'r-')
16 ax.plot(thighres, sqrt(1-sin(thighres)**2), 'gx')
17 plt.xticks(t, lbl, rotation=90)
18 ax.grid()
19 plt.savefig(os.path.basename(__file__).replace('.py','') + '_gen.pdf')

```



22 PDF-Dateierzeugung

22.1 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.

```

1 from reportlab.pdfgen.canvas import Canvas
2 # pagesize units are in 1/72 inch
3 canvas = Canvas("ausgabe.pdf", pagesize=(80/25.4*72, 50/25.4*72))
4 FoSi = 10 # Fontsize
5 canvas.setFont("Times-Roman", FoSi)
6 textobject = canvas.beginPath(2, 50/25.4*72 - FoSi)
7 txt = "hallo"
8 txt = txt + "\n_welt"
9 txt = txt + "\n_dritte_Zeile"
10 txt = txt + "\n_vierte_Zeile"
11 textobject.textLines(txt)
12 canvas.drawText(textobject)
13 canvas.save()

```

Erzeugt:


```
hallo
welt
dritte Zeile
vierte zeile
```

23 Pandas

23.1 Tabellenbreite in Pandas

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

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

23.2 Mehrzeilenstring als Datenquelle

Die Dateneingabe als mehzeiliger String kann folgendermaßen erfolgen:

```
1 ###
2 import sys
3 import pandas as pd
4 from io import StringIO
5 txtdata = StringIO("""
6 Author;uuuuuuTitel;uuISBN
7 Muster,uMax;uTest;uuu12345
8 """)
9 df = pd.read_csv(txtdata, sep=";")
10 print(df)
```

23.3 Spalten kombinieren

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

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

24 Konsolenanwendungen

24.1 Benutzereingaben in der Konsole

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

```
1 text = input("Eingabe: ")
2 print(text)
```

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

```
1 for i in range(10):
2     print(i)
3     input()
```

25 GUIs

25.1 MessageBox

Einfache MessageBox:

```
1 from tkinter import *
2 from tkinter import messagebox
3 root = Tk()
4 root.withdraw()
5 messagebox.showinfo('Dialogtitel', 'Nachricht')
```

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

```
1 root = Tk()
2 root.withdraw()
3 texto = Toplevel(root)
4 messagebox.showinfo('Warn', 'Messagetext', parent=texto)
5 root.destroy()
```

26 XML-Dateien

26.1 Elemente finden

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

```
1 import xml.etree.ElementTree as ET
2 tree = ET.parse('schaltplan.get')
3 root = tree.getroot()
4 for e in root.iter('element'):
5     print(e)
```

26.2 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.get')
3 root = tree.getroot()
4 print(list(root))
```

27 Interaktive Plots

27.1 Interaktive Plots mit iPython

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

```
1 """
2 # Quelle: https://ipywidgets.readthedocs.io/
3 # en/stable/examples/Using%20Interact.html
4 import matplotlib.pyplot as plt
5 import numpy as np
6 from ipywidgets import interact
7 import ipywidgets as widget
8
9 def f(t0, k):
10     t = np.linspace(-2, 2, 200)
11     y = -np.cos(1/(k/1000+(t-t0)**2))
12     fig, ax = plt.subplots()
13     g1, = ax.plot(t, y, 'r-', lw=2)
14     plt.grid()
15
16 interact(f, t0=(-2,2,0.1), k=(1,500,1))
```

Ausgabe:

