IN DIESEM KAPITEL

- Was Python ist und was Sie damit machen können
- Ersetzen Sie Ihren Taschenrechner durch einen, der Variablen und komplexe Zahlen versteht
- Werten Sie mit *NumPy* einen Datensatz aus
- Zeichnen Sie Ihr erstes druckreifes Diagramm

Kapitel 1
Erstkontakt

Computer sind aus dem Ingenieursalltag – wie auch aus dem Rest des Lebens – nicht mehr wegzudenken. Egal ob Sie Maschinen auslegen, Daten analysieren oder visualisieren wollen oder eine Vermutung durch Simulation überprüfen, fast jeder Schritt wird durch Software unterstützt. Viele sind mit etwas Tabellenkalkulation und dem gelegentlichen Spezialprogramm für eine besondere Anwendung bereits zufrieden. Doch sobald Sie sich abseits ausgelaufener Pfade bewegen wollen, kann ein bisschen maßgeschneiderter Code Ihre Arbeit enorm erleichtern.

Was ist Python und wozu ist es gut?

Fortgeschrittene Nutzer, zu denen Sie mit hoher Wahrscheinlichkeit zählen, bauen sich diese Lösungen einfach selbst. Dazu greifen in den letzten Jahren immer mehr und mehr Menschen auf Python zurück. Aktuelle Statistiken wie die stackoverflow.com-Entwicklerumfrage 2020 sprechen Bände: Python ist die am vierthäufigsten verwendete Programmiersprache und rangiert damit vor Java, C#, C++ und vielleicht allen anderen alteingesessenen Sprachen, die Ihnen bisher begegnet sind. Und Leute nutzen Python nicht unter Zwang, auf der Beliebtheitsskala ist Python auf Platz 3. Unter den Sprachen, die Menschen gerne mal ausprobieren möchten, nimmt Python sogar den ersten Platz ein.

Woher kommt diese Begeisterung? Hier ein kurzer Überblick über die wichtigsten Schlagwörter: Python …

- ✔ ermutigt zum Schreiben von gut lesbarem Code, unter anderem durch minimale Syntax, wenige Sonderzeichen, verständliche Schlüsselwörter und verpflichtende Einrückungen.

- ✔ wird von einem Interpreter ausgeführt. Damit ist es plattformübergreifend und es gibt keine Kompilationszeit.
- ✔ kann prozedural, objektorientiert oder funktional geschrieben werden, je nach Anwendungsfall und Geschmack.
- ✔ bringt eine umfangreiche Standardbibliothek und nützliche Datentypen eingebaut mit (`list`, `tuple`, `dict`).
- ✔ kann externen Code aus kompilierten Sprachen wie C, C++ oder Fortran einbinden.

Und das Beste: Python ist freie Software, also komplett kostenlos und quelloffen. Damit steht es im starken Kontrast zum Großteil der sehr teuren Softwarepakete für Ingenieursanwendungen.

Wem ist das alles zu verdanken? Die Version 1.0 wurde 1994 vom Niederländer Guido van Rossum veröffentlicht. Der Name ist abgeleitet von der britischen Komikertruppe »Monty Python«, die Assoziation mit der Schlange folgte erst später. Die Weiterentwicklung der Sprache wird jetzt koordiniert von der »Python Software Foundation«, mit van Rossum (bis 2018) in der Position des »wohlwollenden Diktators auf Lebenszeit«. Der Antrieb kommt jedoch aus der globalen Community, bestehend aus professionellen Programmierern in verschiedenen Firmen und Freiwilligen, die Fehler beheben, eigene Pakete veröffentlichen und Änderungsvorschläge an der Sprache selbst in Form sogenannter »Python Enhancement Proposals« (PEP) einreichen.

Dieser Community haben wir auch die starke Prägung in Richtung Wissenschafts- und Ingenieursanwendungen zu verdanken. Dank vieler kostenloser und hochqualitativer Pakete können Sie mit Python unter anderem

- ✔ sowohl numerisch als auch symbolisch rechnen,
- ✔ Daten visualisieren,
- ✔ Benutzeroberflächen (GUIs) entwickeln,
- ✔ mit externen Laborgeräten kommunizieren.

Jupyter-Notebook im Web

Sicher brennt es Ihnen jetzt unter den Fingern, direkt loszulegen. Damit Ihnen kein langer Installationsprozess den Wind aus den Segeln nimmt, empfehlen wir, die nächsten Beispiele im Browser auszuprobieren.

Eine Umgebung mit allen benötigten Paketen haben wir für Sie bereits vorbereitet. Diese können Sie direkt im Browser benutzen, besuchen Sie dazu `https://www.wiley-vch.de/ISBN9783527717675` oder `https://python-fuer-ingenieure.de`.

Natürlich können Sie das alles auch lokal durchführen. Dazu benötigen Sie einen installierten Python-Interpreter, die Pakete *NumPy* und *Matplotlib* sowie den *Jupyter*-Server. Wie Sie all das einrichten, können Sie in Kapitel 2 »Installation und Inbetriebnahme« nachschlagen.

Nach einer kurzen Ladezeit von etwa einer Minute befinden Sie sich in der *Jupyter*-Oberfläche. Öffnen Sie jetzt das *Notebook* mit dem Namen `Leeres Notebook.ipynb`. Es gibt viele Arten, Python auszuführen. Neben dem klassischen Code in Textdateien bietet sich gerade für die Erkundungsphase ein interaktiver Interpreter an, in dem man sich jederzeit Zwischenergebnisse anzeigen lassen kann. Das bietet unter anderem die Notebook-Oberfläche, in der Sie sich gerade befinden. Code ist hier in sogenannte *Zellen* unterteilt, die einzeln ausgeführt werden können. Wie das aussieht, sehen Sie in Abbildung 1.1.

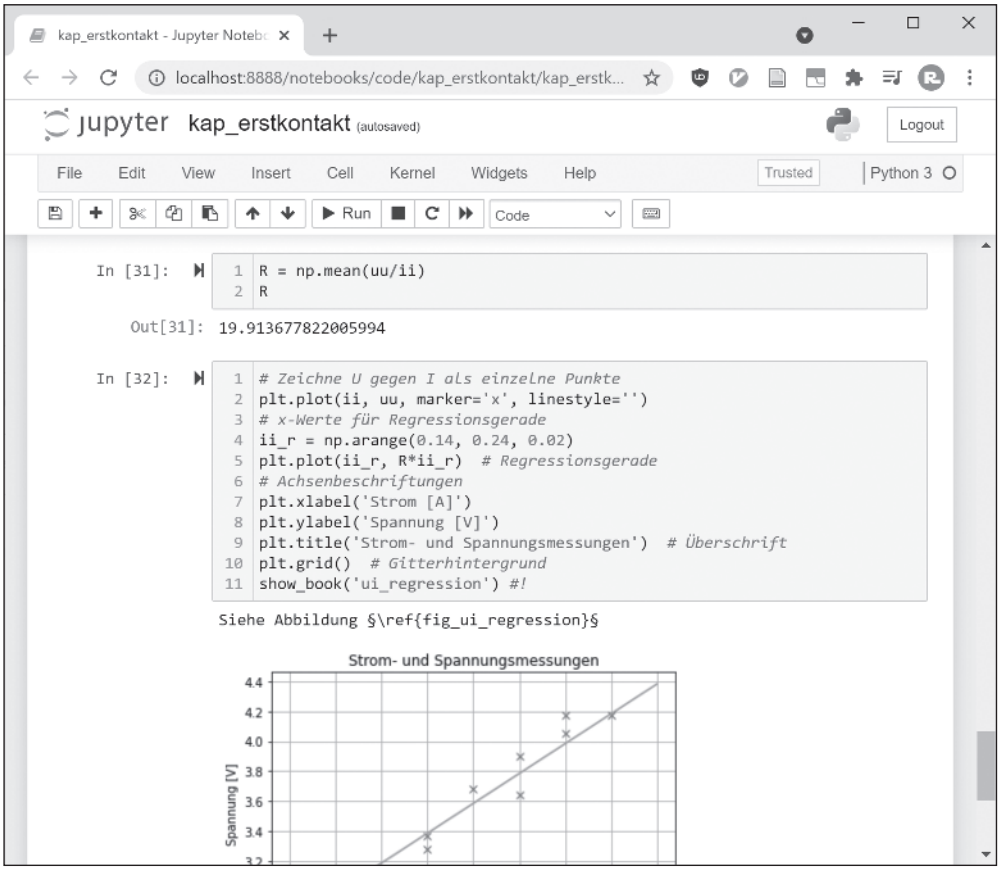

Abbildung 1.1: Oberfläche eines *Jupyter*-Notebooks. Code ist in einzeln ausführbare Zellen unterteilt, Grafiken werden direkt eingebettet.

Probieren geht über studieren – klicken Sie in die leere Zelle und geben Sie ein:

```
>> 1 + 1
```

Klicken Sie dann auf den Button Run oder nutzen Sie die Tastenkombination ⇧ + ↵, um die Zelle auszuführen. Das Ergebnis sollte jetzt mit der Beschriftung `Out[1]:` aufgetaucht sein. Wenn die letzte Zeile einen Wert enthält – und zwar ohne = davor – wird dieser immer automatisch ausgegeben.

Im Notebook gibt es zwei unterschiedliche Modi:

- ✔ Ist Ihr Cursor innerhalb einer Zelle, befinden Sie sich im *Eingabemodus*. Von hier aus können Sie aus mehreren Tasten bestehende Kombinationen ausführen (wie eben ⇧ + ↵).
- ✔ Ist kein Textcursor zu sehen, sondern eine ganze Zelle ausgewählt, befinden Sie sich im *Befehlsmodus*. Jetzt können Sie auch Kürzel aus einzelnen Buchstaben nutzen.

In den Eingabemodus wechseln Sie mit ↵, mit Esc kehren Sie zum Befehlsmodus zurück.

Hier noch eine Handvoll weitere nützliche Tastenkürzel:

⇧ + ↵	führt Zelle aus und bewegt Cursor in nächste Zelle
Strg + ↵	führt Zelle aus und belässt Cursor auf aktueller Zelle
A	fügt neue Zelle über aktueller Zelle ein (»above«)
B	fügt neue Zelle unter aktueller Zelle ein (»below«)
H	zeigt Übersicht mit allen Tastenkürzeln an

Über die Schaltfläche Help | User Interface Tour gelangen Sie außerdem zu einem interaktiven Überblick über die gesamte Benutzeroberfläche.

Besser als Ihr »Casio«

Die erste nützliche Verwendung von Python ist als Taschenrechner, und zwar mit einer solchen Fülle von Funktionen, dass Ihr wissenschaftlicher Taschenrechner aus Schulzeiten einpacken kann. Neben den Grundrechenarten und Elementarfunktionen können Sie auch große Listen von Daten in einem Rutsch auswerten, mit Vektoren und Matrizen rechnen sowie sich Diagramme jeglicher Art anzeigen lassen. Aber fangen wir erst einmal einfach an.

```
>> 27+39  # Grundrechenarten wie erwartet
66
>> 3*7.5  # Punkt ist Dezimaltrennung
22.5
>> 10/3  # Division liefert Gleitkommazahl
3.3333333333333335
>> 2**10  # Potenzen mit ** Operator
1024
```

```
>> n = 42  # Variablen mit = gleichzeitig anlegen und zuweisen
>> n + 1/n
42.023809523809526
```

Wenn es über die Grundrechenarten hinausgehen soll, müssen Sie das Modul `math` importieren. Das ist in der Standardbibliothek enthalten, damit bei jeder Python-Installation bereits dabei und muss nur importiert werden.

```
>> import math  # Modul mit mehr Funktionen und Konstanten
>> # Vor allen importierten Werten steht dann "math."
>> 2*math.pi
6.283185307179586
>> math.sin(math.pi/2)
1.0
>> (1+5j)*1j  # Komplexe Zahlen mit imag. Einheit j
(-5+1j)
>> math.e**(1j*math.pi)  # (naja, fast)
(-1+1.2246467991473532e-16j)
```

In der letzten Zeile können Sie erkennen, dass wir immer noch mit Gleitkommazahlen rechnen und gelegentlich auf deren numerische Beschränkungen stoßen. Symbolische Rechnung geht natürlich auch, sie hätte hier das exakte Ergebnis von −1 geliefert. Dazu jedoch erst in Kapitel 6 »Brunftzeit für Termhirsche« mehr.

Wenn Sie Datenreihen vorliegen haben, ist es nur logisch, die einzelnen Werte auch geschlossen in einer Datenstruktur abzuspeichern. Dafür bietet Python ein vielseitiges Werkzeug: die Liste. Eingeschlossen in eckige Klammern `[]` wird Ihnen dieser Kollege regelmäßig dort begegnen, wo es um Sammlungen von Objekten jeglicher Art geht.

```
>> a = [5, 6, 4, 8, 9, 10]  # Listen in [] schreiben
>> a[2]  # mit [] Elemente auslesen, Zählung beginnt bei 0
4
>> a[2] = 7  # Elemente schreiben
>> a
[5, 6, 7, 8, 9, 10]
>> # Teilliste (Slice) erzeugen, oberer Index nicht inklusive
>> a[0:2]
[5, 6]
>> a[:2]  # implizite untere Grenze
[5, 6]
>> a[0:6:2]  # Slice mit Schrittweite 2
[5, 7, 9]
>> a[::-1]  # Negative Schrittweite dreht Liste um
[10, 9, 8, 7, 6, 5]
>> [1, 2, 3, 4] + a  # + fügt Listen zusammen
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>> a * 2  # Liste "wiederholen"
[5, 6, 7, 8, 9, 10, 5, 6, 7, 8, 9, 10]
```

Bei den letzten zwei Zeilen hätte Ihr Mathelehrer sicher etwas skeptisch geguckt. Für gewöhnliche Listen werden in Python einige Rechenoperatoren – eben wie + und * – für andere Operationen zweckentfremdet. Wenn Sie tatsächlich alle Listenelemente verdoppeln wollen, sieht das mit ganz altmodischem Python-Code so aus:

```
>> doubled_a = []  # Lege neue, leere Liste an
>> for x in a:  # Für jedes Element x in Liste a...
>>     doubled_a.append(x * 2)  # ... füge der neuen Liste x*2 an
>> doubled_a
[10, 12, 14, 16, 18, 20]
```

In dieser Form mit Datenreihen zu rechnen, wäre umständlicher, als Sie sich sicher erhofft hätten. Glücklicherweise sah das die Python-Community genau so und entwickelte eine praktikablere Alternative, die Sie im nächsten Abschnitt kennenlernen.

Erstes Date mit NumPy-Arrays

Wie Sie gerade gesehen haben, sind Listen zum Rechnen mit vielen Zahlen auf einmal nicht gerade das Mittel der Wahl. Hier können Sie das erste Mal auf die starke Community zurückfallen. Das Paket *NumPy* bietet einen regelrechten Baumarkt an Werkzeugen für numerische Berechnungen in Python, dessen Kern das sogenannte *NumPy*-Array bildet. Ein Array sieht auf den ersten Blick aus wie eine Liste, sämtliche Rechenoperationen werden aber standardmäßig elementweise ausgeführt. Außerdem sind häufige Operationen intern mit C beschleunigt, sodass Sie problemlos und sehr performant (also schnell) auch große Datenmengen verarbeiten können. In den oben verlinkten Online-Notebooks ist das Paket bereits installiert, sodass Sie es genau wie jedes eingebaute Modul verwenden können.

Bei einer lokalen Installation muss das Paket erst vom *Python Package Index (PyPI)* heruntergeladen werden. Wenn Sie Glück haben, funktioniert das in der Kommandozeile mit dem Befehl `pip install numpy`. Falls das nicht auf Anhieb klappt, konsultieren Sie Kapitel 2 »Installation und Inbetriebnahme«.

Auf dem Cover dieses Buches steht ja schließlich »Python für Ingenieure«, also wird es auch langsam Zeit für ein kleines Ingenieursbeispiel. An einem elektrischen Widerstand haben Sie Strom und Spannung gemessen, 10 Messwerte in einem Zeitraum von einer Sekunde. Mit diesem Datensatz wollen wir nachfolgend etwas experimentieren.

```
>> import numpy as np  # Abkürzung des Importnamens zu np
>> # np.array konvertiert normale Liste zu Numpy-Array
>> # Spannungsmessungen
>> uu = np.array([2.97, 3.37, 3.68, 3.90, 4.17,
>>                4.17, 4.05, 3.64, 3.28, 2.87])
array([2.97, 3.37, 3.68, 3.9 , 4.17, 4.17, 4.05, 3.64, 3.28, 2.87])
```

```
>> # Strommessungen in mA
>> ii_mA = np.array([150, 170, 180, 190, 200,
>>                   210, 200, 190, 170, 150])
>> # Operationen mit Skalaren wirken elementweise
>> ii = ii_mA / 1000  # mA in A umrechnen
array([0.15, 0.17, 0.18, 0.19, 0.2 , 0.21, 0.2 , 0.19, 0.17, 0.15])
>> # Momentanleistung berechnen
>> pp = uu * ii  # Operationen mit Arrays wirken auch elementweise
array([0.4455, 0.5729, 0.6624, 0.741 , 0.834 , 0.8757, 0.81  ,
       0.6916, 0.5576, 0.4305])
>> # Durchschnittliche Leistung
>> (uu @ ii) / len(uu)  # Skalarprodukt mit @ Operator
0.66212
>> np.mean(pp)  # berechnet auch arithmetisches Mittel
0.66212
```

An dieser Stelle versteckt sich eine Falle: als Grundregel sollten Sie sich merken, dass die normalen Mathefunktionen in Python (aus dem Modul `math`) nichts mit *NumPy* -Arrays anfangen können. Stattdessen müssen Sie die passenden Funktionen aus *NumPy* verwenden.

```
>> math.log(uu)  # Funktionen aus math gehen nicht mit Arrays
TypeError: only size-1 arrays can be converted to Python
           scalars
>> np.log(uu)  # Dafür gibt es NumPy-Alternativen
array([1.08856195, 1.21491274, ..., 1.05431203])
```

Bunte Bilder mit Matplotlib

Bis hier konnte Ihr Taschenrechner vielleicht noch mithalten, aber am Ende jeder Auswertung steht meistens ein überzeugendes Diagramm. Kein Grund, das Millimeterpapier zu zücken und den Bleistift zu spitzen! Mit dem Paket *Matplotlib* können Sie in nur wenigen Zeilen Code druckreife Abbildungen jeglicher Art erstellen. Die Zeitverläufe von Strom und Spannung sind als erste dran:

```
>> import matplotlib.pyplot as plt
>> # Array der Messzeiten (von, bis, Schrittweite)
>> tt = np.arange(0, 1.0, 0.1)
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
>> plt.plot(tt, uu, label='U [V]')  # Zeichne Spannungsverlauf
>> plt.plot(tt, ii, label='I [A]')  # Zeichne Stromverlauf
>> plt.legend()  # Legende anzeigen
```

Die entstandenen Verläufe sind in Abbildung 1.2 zu sehen.

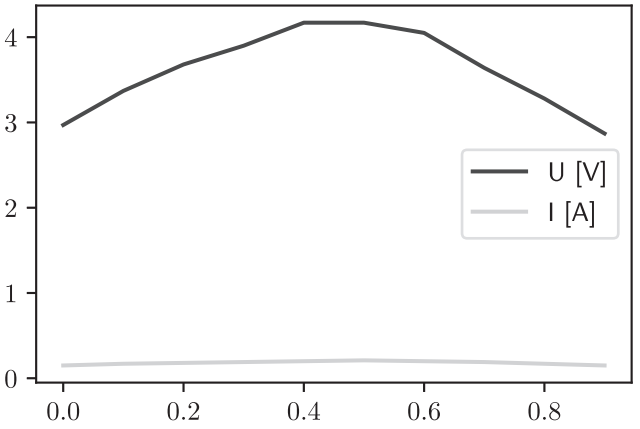

Abbildung 1.2: Zeitverlauf von Strom und Spannung.

An die Funktion `plot(xx, yy, ...)` werden zwei Arrays übergeben: das erste enthält die x-Koordinaten der darzustellenden Punkte, das zweite die y-Koordinaten. Für den Zeitverlauf müssen wir also zuerst mit `numpy.arange(start, stop, step)` ein Array von Abtastzeiten anlegen. Dieses Vorgehen wird Ihnen sehr häufig begegnen, da dieses Beispiel sicher nicht der letzte Zeitverlauf in diesem Buch oder Ihrer Python-Karriere war.

Ziel ist es jetzt herauszufinden, welchen Widerstandswert das vermessene Bauelement besitzt. Dazu böte sich grundsätzlich eine kleine Regressionsanalyse an. Für dieses Einführungsbeispiel reicht aber auch eine Mittelung über die Einzelwiderstandswerte. Mit dem so erhaltenen Wert lässt sich der lineare Zusammenhang $U = I \cdot R$ als schöne Näherungsgerade darstellen:

```
>> R = np.mean(uu/ii)
19.913677822005994
>> # Zeichne I über U als einzelne Punkte
>> plt.plot(ii, uu, marker='x', linestyle='')
>> # x-Werte für Gerade
>> ii_r = np.arange(0.14, 0.24, 0.02)
>> plt.plot(ii_r, R*ii_r)  # Näherungsgerade
>> # Achsenbeschriftungen
>> plt.xlabel('Strom [A]')
>> plt.ylabel('Spannung [V]')
>> plt.title('Strom- und Spannungsmessungen')  # Überschrift
>> plt.grid()  # Gitterhintergrund
```

In Abbildung 1.3 ist das Ergebnis dargestellt.

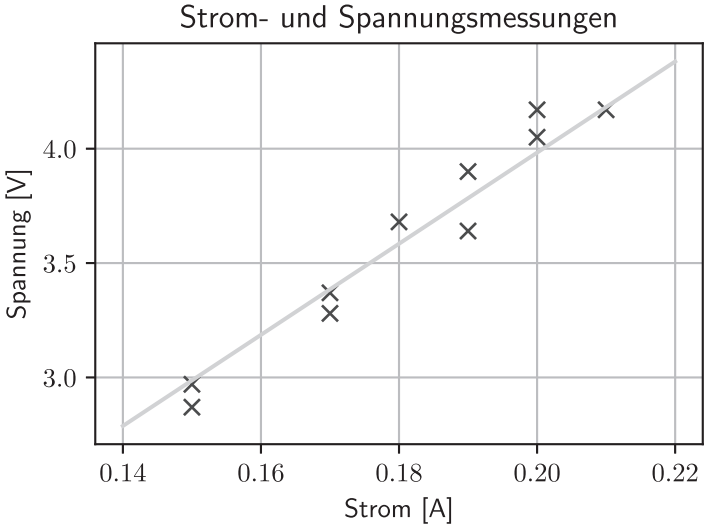

Abbildung 1.3: Messdaten und Näherungsgerade im U-von-I-Diagramm.

Was Sie hier gerade ausgerechnet haben, ist keine »echte« Regression. Dazu müsste man die Methode der kleinsten Fehlerquadrate verwenden, aber das hat Zeit bis Kapitel 5 »Ein bisschen Mathe – So viel Spaß muss sein«. Im Abschnitt »Fit for Fun: Funktionsapproximation« in Kapitel 9 »Optimierung – Besser geht's nicht« wird ein sehr ähnliches Beispiel dann nochmal aufgegriffen und der Unterschied diskutiert.

Nichtsdestotrotz: für die hier betrachtete Situation liefert dieser hemdsärmelige Ansatz ohne viel Trara ein ziemlich gutes Ergebnis.

Und damit ist es bereits erledigt! Vor einigen Minuten haben Sie vielleicht noch nie Python gesehen, jetzt berechnen und visualisieren Sie schon eine lineare Approximation eines verrauschten Datensatzes – und das alles in nur wenigen Zeilen gut lesbarem Code. Das fasst die Hauptargumente aus der Einleitung gut zusammen: die leicht lesbare Syntax, kombiniert mit einer hilfreichen Community, die viele nützliche Pakete kostenlos bereitstellt, macht Python zu einem überzeugenden Werkzeug, von dem fast alle profitieren können, die gelegentlich zum Taschenrechner greifen.

Zusammenfassung

- ✔ Python wurde entworfen, um einfach lesbar und benutzbar zu sein, was sich in der Syntax und typischen Code-Mustern widerspiegelt.
- ✔ Im interaktiven Modus eignet sich der Interpreter als leistungsstarker »Taschenrechner«.

- ✔ Für umfangreichere Experimente sind *Jupyter*-Notebooks ein perfekter Spielplatz.
- ✔ Python besitzt dank seiner Community eine besonders umfangreiche Sammlung nützlicher Pakete für wissenschaftliche Anwendungen.
- ✔ Sobald es Richtung Numerik oder Visualisierung geht, werden die Pakete *NumPy* und *Matplotlib* nahezu Pflicht.