Mittelwert Standardabweichung

September 26, 2022

0.1 Einfache Beispiele zur Fehlerrrechnung

Notebook erstellt am 03.09.2022 von C. Rockstuhl, überarbeitet von Y. Augenstein

In diesem Notebook werden wir einfache Beispiele zur Berechnung des Mittelwertes und der Standardabweichung kennenlernen. Dieses allererste Notebook wird uns vor allem auch helfen, einfache Problemstellungen zu bearbeiten und die Umgebung besser kennenzulernen.

0.1.1 Ausgangspunkt: Unsere Daten

Ausgangspunkt aller unserer weiterer Betrachtungen sind zunächst einmal unsere Messgrößen. Hier gehen wir davon aus, dass wir eine gegebene Messgröße eine endliche Anzahl von Malen unter nominell gleichen experimentellen Bedingungen gemessen haben. Beispiele für eine solche Messgröße können eine Zeitspanne t oder eine Ortskoordinate x sein. Wir beschränken uns hier der Einfachheit halber auf skalare Größen, wie wir sie in der Vorlesung bisher diskutiert haben.

Diese Messgrößen sind also gegeben als eine endliche Anzahl von N Messwerten, also $t_1, t_2, ..., t_N$. Zur besseren Handhabung werden wir alle diese Messwerte in einer Liste zusammenfassen.

[1]:
```
meine_messwerte = [3, 4, 5, 6, 10]  # Sie können hier einfach dynamisch neue␣
↪Listen erstellen, die
                                    # verschiedene Werte beinhalten. Später␣
↪können Sie Elemente
                                    # dieser Liste individuell adressieren.
                                    # Tendenziell können diese Listen␣
↪verschiedene Variabeltypen beinhalten.
```

0.1.2 Mittelwert

In der Vorlesung wurde der Mittelwert einer Messgröße t berechnet als

$$\langle t \rangle = \frac{\sum_{i=1}^{N} t_i}{N}$$

Die Schritte zur Berechnung des Mittelwerts sehen also wie folgt aus: 1. Berechnen Sie als Erstes die Summe aller gemessenen Werte der Funktion t. 2. Teilen Sie diese Summe durch die Anzahl der Messungen, um zum Mittelwert zu gelangen.

Wir definieren uns als Erstes eine Funktion, die die einzelnen Werte aufaddiert und dann diese Summe durch die Anzahl der Einträge dividert.

[2]:
```
def mean(data):                    # Hier definieren wir den Funktionskopf. Er beinhaltet den Funktionsnamen
                                   # und das Argument, mit dem dann die Funktion aufgerufen werden muss.

                                   # Baechten Sie bitte hier und im Folgenden einige Python spezifische typische
                                   # Elemente der Formatierung. Dazu zählen vier Lehrzeichnen beim Einzug und ein
                                   # Leerzeichen zwischen zwei Operatoren.

    summe = 0                      # Zu Beginn unserer Summation weisen wir dieser Variablen den Wert 0 zu.

    for element in data:           # Mit dieser Schleife iterieren wir jetzt über alle Elemente unserer Summe.
                                   # Beachten Sie bitte, dass die Notation der Summe hier etwas speziell ist und
                                   # anders im Vergleich zu anderen Programmiersprachen. Sie addressieren hier exlizit
                                   # die Elemente der Liste. Versuchen Sie sich an die Notation zu gewöhnen, und verstehen
                                   # es bitte als einen Konstrukt der Sprache.

        summe = summe + element    # Das Format dieser Einrückung ist wichtig um zu erkennen, über was hier
                                   # alles iteriert wird. Hier wird praktisch jedes einzelne Element der Liste aufsummiert.

    mean = summe / len(data)       # Abschliessend müssen wir zur Mittelwertbildung diese Summer noch durch die Anzahl der
                                   # Elemente dividieren.
                                   # Mit dem Befehl bestimmen wir die Länge der Liste, die Anzahl der Werte also,
                                   # die zur Mittelwertbildung benötigt wird.

    return mean                    # Wir beenden die Funktion, in dem wir angeben, welches Ergebniss unsere Funktion
                                   # zurückgeben soll.
```

Die for-Schleife durchläuft hier also eine vorgegebene Liste und wir können direkt die Einzelwerte der Liste verwenden.

[3]:
```
mean(meine_messwerte)       # Hier rufen Sie einfach die Funktion auf und␣
↪erhalten das Ergebnis. Sie können es sich
                            # hier anzeigen lassen oder einer neuen Variabel␣
↪zuordnen.
```

[3]:
```
5.6
```

Das ist eine etwas sehr ausführliche Berechnung und wir können hier auch einfacher schreiben und auf eingebaute Routinen zurückgreifen.

[4]:
```
def mean_2(data):
    mean = sum(data) / len(data)  # Alles, was wir in dieser Funktion␣
↪vereinfachen, ist das direkte Summieren über
                                  # alle Elemente in der Liste mit der␣
↪vordefinierten Funktion sum(). Das vereinfacht die
                                  # Rechnung ungemein. Allgemein gilt, dass␣
↪wenn Sie in einer Schleife eine Operation
                                  # durchführen, die alle Elemente einer␣
↪Liste betreffen, Sie üblicherweise auch einfache
                                  # Funktionen schreiben oder auf diese␣
↪zurückgreifen können, in der diese Operation
                                  # direkt ausgeführt wird.
    return mean
```

[5]:
```
mean_2(meine_messwerte)
```

[5]:
```
5.6
```

Es ist natürlich nicht immer notwendig, alle Funktionen selbst zu programmieren. Der Vorteil von Python ist gerade, dass sehr viele Bibliotheken open source zur Verfügung gestellt werden, und auf diese können wir hier zurückgreifen. Eine Bibliothek mit Funktionen rund um die Statistik lässt sich z.B. so einbinden:

[6]:
```
import numpy as np          # Mit diesem import Befehl importieren Sie eine␣
↪bestimmte Bibliothek.
                            # Diese enthält (meistens thematisch) eine ganze␣
↪Reihe von Funktionen, auf welche Sie
                            # dann im Folgenden zurückgreifen können.
                            # Welche Funktionen zur Verfügung gestellt␣
↪werden,können Sie in den meistens
                            # sehr umfangreichen Online-Dokumentationen␣
↪nachlesen.
```

```
                        # Wir versuchen hier überwiegend zwei Bibliotheken␣
↪zu verwenden. NumPy und SciPy.
                        # NumPy wird für alle Arten numerischer Rechnungen␣
↪benötigt bei denen es vor allem darum
                        # geht, Vektoren, Matrizen oder groß e␣
↪mehrdimensionale Arrays zu verarbeiten.
                        # Die Dokumentation für diese NumPy-Bibliothek␣
↪finden Sie unter:
                        # https://numpy.org

                        # SciPy wird für viele wissenschaftliche Arbeiten␣
↪verwendet, welche auf NumPy aufbauen.
                        # Die Dokumentation für diese SciPy-Bibliothek␣
↪finden Sie unter:
                        # https://scipy.org

                        # Es gibt eine sehr umfangreiche␣
↪Standardbibliothek (dokumentiert unter
                        # https://docs.python.org/3/library/) und viele␣
↪Spezialbibliotheken.
                        # Diese werden wir im Laufe des Kurses punktuell␣
↪kennenlernrn.


                        # Beachten Sie bitte, dass einer üblichen␣
↪Konvention folgend, NumPy as np importiert wird,
                        # was die spätere Arbeit etwas erleichtern wird.
```

Eine erneute Berechnung des Mittelwertes erfolgt nun mit dem Befehl:

[7]:
```
np.mean(meine_messwerte)    # Eine der zur Verfügung gestellten Funktionen␣
↪ist gerade die Mittelwertbildung.
```

[7]:
```
5.6
```

Sie können auch einen ganzen Satz ausgeben in dem Sie den folgenden Befehl verwenden

[8]:
```
print(f"Der Mittelwert unserer Messergebnisse beträgt {np.
↪mean(meine_messwerte)}")
                        # Die Logik des Printbefehls ist hoffentlich␣
↪eindeutig. Sie können ihn zur Ausgabe von Text verwenden.
```

```
                        # Wir verwenden hier sogenannte f-strings (f"" - das␣
↪f ist wichtig!), in die wir einfach die Variable,
                        # die wir ausgeben wollen, mit geschweiften Klammern␣
↪einsetzen können.
                        # Hier soll dann also der Mittelwert ausgegeben␣
↪werden.
```

```
Der Mittelwert unserer Messergebnisse beträgt 5.6
```

0.1.3 Standardabweichung

In der Vorlesung wurde die Standardabweichung einer Messgröße t berechnet als

$$\sigma_t = \sqrt{\frac{\sum_{i=1}^{N} (t_i - \langle t \rangle)^2}{N-1}}$$

Die Schritte zur Berechnung der Standardabweichung sehen also wie folgt aus: 1. Berechnen Sie den Mittelwert wie oben beschrieben. 2. Berechnen Sie anschliessend die Varianz für jeden Eintrag, indem Sie den Mittelwert vom Wert des Eintrags subtrahieren. 3. Quadriere dann jeden dieser resultierenden Werte und summieren Sie die Ergebnisse. 4. Teilen Sie dann das Ergebnis durch die Anzahl der Datenpunkte minus eins. 5. Die Quadratwurzel der Varianz (oben berechnet) ist die Standardabweichung.

Wir können wieder alle diese Schritte transparent in einer eigens hierfür definierten Funktion durchführen. Beachten Sie bitte, dass wir hier eine neue Bibliothek importieren müssen zur Berechnung der Wurzel.

[9]:
```
def stdev(data):
    N = len(data)
    mean = sum(data) / N
    deviations = [(element - mean) ** 2 for element in data]
                                    # Hier sehen Sie zwei neue Elemente. Zum␣
↪einen beachten Sie bitte, wie das Quadrat einer
                                    # Zahl berechnet wird.
                                    # Zum anderen sehen Sie hier eine noch␣
↪kompaktere Version, eine for-Schleife ablaufen
                                    # zu lassen (siehe https://docs.python.
↪org/3/tutorial/datastructures.html#list-comprehensions).
                                    # Im Ergebniss erzeugen Sie sich hier␣
↪eine Liste, deren Elemente dann gerade
                                    # den Elementen entspricht, über die Sie␣
↪dann noch summieren müssen.
```

```
    var = sum(deviations) / (N-1)  # Hier berechnen Sie nun das Argument der␣
↪Wurzel.
    std_dev = np.sqrt(var)
    return std_dev
```

In dem obigen Beispielcode haben wir eine weitere Art kennengelernt, wie man eine for-Schleife in Python programmieren kann. In diesem Falle ist es eine ein-zeilige for-Schleife, die den Code schon ziemlich kompakt erscheinen lässt. Wenn Sie sich damit unwohl fühlen, schreiben Sie einfach die for-Schleife wie oben angegeben aus.

[10]:
```
stdev(meine_messwerte)
```

[10]:
```
2.701851217221259
```

Beachten Sie, auch hier hätten wir nicht zwingend die entsprechende Funktion selbst implementieren müssen, sondern wir hätten auch hier wieder auf die entsprechenden Funktionen zurückgreifen können.

[11]:
```
np.std(meine_messwerte, ddof = 1)   # Baechten Sie hier bitte eine␣
↪Besonderheit in der Implementierung in
                                    # in NumPy. Hier wird in der Berechnung␣
↪der Standardabweichung dividiert
                                    # duch N und nicht durch N-1. Daher␣
↪verwenden wir als ein zusätzliches Argument
                                    # ddof, was für Delta Degrees of Freedom␣
↪steht. In der Division wird dann
                                    # N-ddof berücksichtig.
```

[11]:
```
2.701851217221259
```

Damit wir wieder einen ganzen Satz als Ergebnis bekommen, können wir schreiben:

[12]:
```
print(f"Die Standardabweichung unserer Messergebnisse beträgt {np.
↪std(meine_messwerte)}")
```

```
Die Standardabweichung unserer Messergebnisse beträgt 2.4166091947189146
```

Da es etwas unnatürlich aussieht, dass wir so viele Stellen für das Ergebnis angeben, wollen wir es besser runden auf die Anzahl der Stellen, in denen unsere Messwerte auch vorliegen.

[13]:
```
print(f"Die Standardabweichung unserer Messergebnisse beträgt {round(np.
↪std(meine_messwerte, ddof = 1), 2)}")
```

```
                    # Hier benutzen wir die Funktion round zum Runden einer␣
↪Zahl. Das zweite Argument gibt die Anzahl
                    # der Nachkommastellen an, mit denen das Ergebnis␣
↪angezeigt wird. In unserem Falle sehen Sie
                    # keine zwei Nachkommastellen, da die zweite␣
↪Nachkommastelle 0 ist.
```

```
Die Standardabweichung unserer Messergebnisse beträgt 2.7
```

0.1.4 Beispiel: Magnetfeld

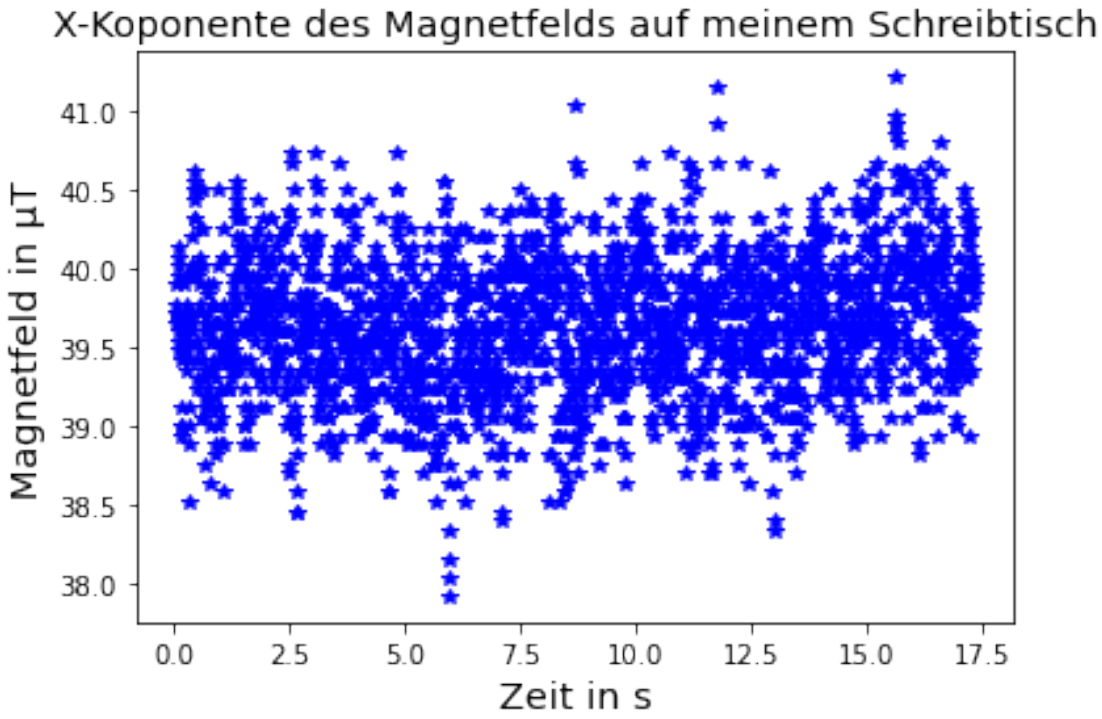
Für die folgende einfache Datenanalyse habe ich mit dem Programm phyphox den Magnetfeldsensor meines Funktelefons für einige Zeit ausgelesen, um passende Daten zu generieren. Ich verwende im Folgenden nur die x-Komponente des Magnetfelds und gehe davon aus, dass diese keine Funktion der Zeit ist. Ich verwende hierfür im Folgenden die Bibliotheken `pandas` (Python Data Analysis Library) zum Einlesen der Daten sowie `matplotlib` zur Visualisierung. Für die Darstellung der Messdaten habe ich einige Kommandos verwendet, um die Abbildung etwas zu verbessern. Die konkreten Befehle hier sind nicht ganz so wichtig. Im Laufe der Zeit werden Sie sicherlich viele verschiedene kennenlernen.

[14]:
```
import pandas as pd             # Wir benötigen hier die Bibliothek pandas,␣
↪die oben im Text beschrieben ist.
                                # Weitere Informationen zu den zur Verfügung␣
↪gestellten Funktionen finden Sie hier:
                                # https://pandas.pydata.org
import matplotlib.pyplot as plt
                                # Hier importieren wir eine spezielle␣
↪Bibliothek zur Visualisierung unserer Daten.
                                # Mehr Informationen zu dieser Bibliothek␣
↪finden Sie hier:
                                # https://matplotlib.org/stable/tutorials/
↪introductory/pyplot.html
                                # Beachten Sie hier als Besonderheit, dass␣
↪ich die Bibliothek unter einem kurzen Namen
                                # importiere, der die spätere Notation␣
↪vereinfacht.

df = pd.read_csv("Magnetfeld.csv", sep=",")
                                # Mit diesem Befehl lesen wir den Inhalt der␣
↪Datei "Magnetfeld.csv". Schauen Sie
                                # sich die Datei einmal in einem Texteditor␣
↪an um den Aufbau zu studieren.
                                # Sie hat also ein header mit Meta-Daten und␣
↪dann die eigentlichen Messdaten.
```

```
                                # Diese Messdaten sind mit einem Komma␣
 ↪getrennt. Da cvs-Dateien unterschiedliche
                                # Zeichen zur Separierung verwenden, müssen␣
 ↪Sie das hier mitteilen.

print(df.head())                # Das was als Header erkannt wurde, können␣
 ↪Sie hier anzeigen lassen.

time = df['Time (s)']           # Und dann weisen Sie die Werte in der␣
 ↪Spalte "Time (s)" der Zeitvariabel zu

Magnetic_field = df['Magnetic Field x (ÂţT)']
                                # Und mit der Spalte "Magnetic Field x␣
 ↪(ÂţT)" verfahren Sie genauso.

                                # Im Folgenden haben wir einige Befehle, die␣
 ↪eine Abbildung erstellen.
                                # Wir fügen der Abbildung einen Titel hinzu␣
 ↪und definieren, was an der x-Achse
                                # und was an der y-Achse stehen soll.

plt.title("X-Koponente des Magnetfelds auf meinem Schreibtisch",␣
 ↪size="x-large")
plt.ylabel("Magnetfeld in ÂţT", size="x-large")
plt.xlabel("Zeit in s", size="x-large")

plt.plot(time, Magnetic_field, "*", markersize=6, color='b')
                                # Der vorhergehende Befehl sagt dem Program,␣
 ↪was genau dargestellt werden soll.
                                # Wir plotten hier also die Werte des␣
 ↪Magnetfeldes als Funktion der Zeit.
                                # Da wir diskrete Messewerte haben werden␣
 ↪die einzelnen Wertepaare als Sternchen
                                # dargestellt mit einer Größ e von 6 und das␣
 ↪ganze in Blau.
plt.show()
```

```
   Time (s)  Magnetic Field x (ÂţT)  Magnetic Field y (ÂţT)  \
0  0.031599              39.719997                 -14.04
1  0.039473              39.719997                 -14.52
2  0.047468              39.899998                 -14.70
3  0.055372              39.779999                 -15.24
4  0.063307              39.660000                 -15.78

   Magnetic Field z (ÂţT)  Absolute field (ÂţT)
0               31.439999             52.566846
```

1	31.859999	52.948731
2	31.980000	53.205453
3	32.219997	53.411929
4	32.639999	53.733542



[15]:
```
print(f"Der Mittelwert der x-Komponente des Magnetfeldes beträgt {round(np.
↪mean(Magnetic_field),4)}")
```

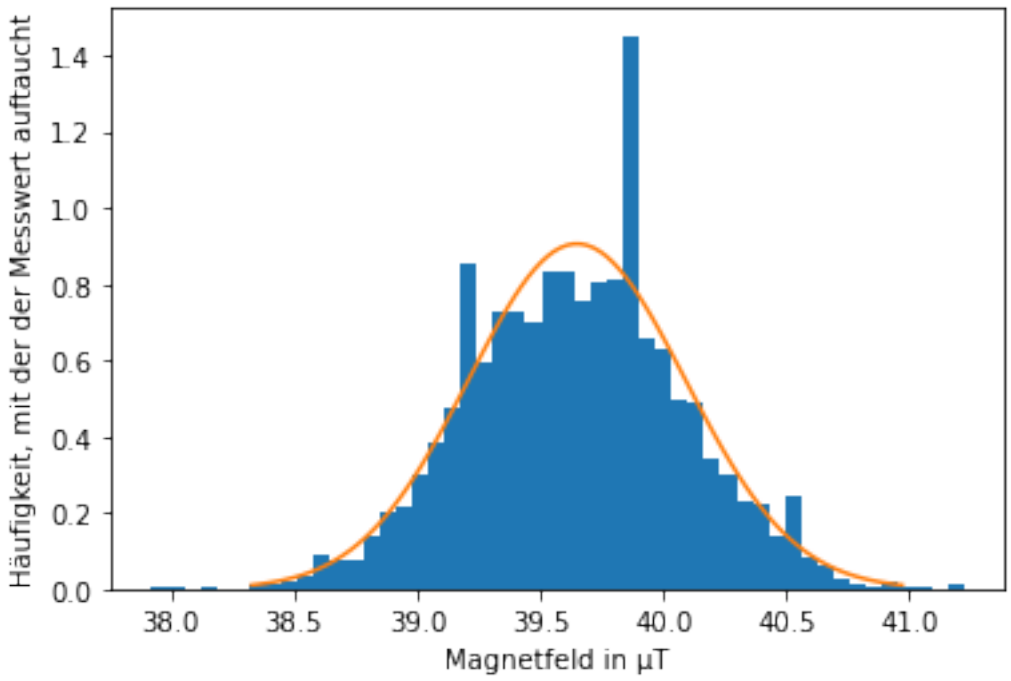
Der Mittelwert der x-Komponente des Magnetfeldes beträgt 39.6484

[16]:
```
print(f"Die Standardabweichung der x-Komponente des Magnetfeldes beträgt␣
↪{round(np.std(Magnetic_field),4)}")
```

Die Standardabweichung der x-Komponente des Magnetfeldes beträgt 0.4404

[17]:
```
import scipy.stats as stats      # Hier importieren wir jetzt lediglich das␣
↪Statistikmodul der Bibliothek SciPy.

mu = np.mean(Magnetic_field)
sigma = np.std(Magnetic_field)
x = np.linspace(mu - 3*sigma, mu + 3*sigma, 100)
```

```
                                # Dieser Befehl erzeugt einen Vektor, der␣
 ↪diskrete Werte annimmt zwischen einem
                                # Anfangswert und einem Endwert. Wir␣
 ↪erzeugen uns hier einen Vektor mit insgesamt
                                # 100 Punkten.

plt.hist(Magnetic_field, density=True, bins=50)
                                # Hier plotten wir ein Histogramm der␣
 ↪Häufigkeitswahrscheinlichkeit.
                                # Beachten Sie bitte, dass wir diese␣
 ↪Häufigkeit normieren, so dass die Summe der
                                # Wahrscheinlichkeit multipliziert mit dem␣
 ↪entsprechenden Wert des Bins (
                                # also dem Wert auf der x-Achse) gerade eins␣
 ↪ergibt.
                                # Würden wir density=False setzen, würde wir␣
 ↪einfach die Anzahl der Messwerte in
                                # einem bestimmten Interval bekommeb

plt.plot(x, stats.norm.pdf(x, mu, sigma))
                                # Zum besseren Vergleich können wir uns hier␣
 ↪eine Normalverteilung erzeugen. Die
                                # drei Paramater sagen gerade an welchen␣
 ↪Stützstellen die Funktion evaluiert
                                # werden soll und geben den Mittelwert und␣
 ↪die Standardabweichung der Werte an.
plt.ylabel("Häufigkeit, mit der der Messwert auftaucht")
plt.xlabel("Magnetfeld in ÂţT")
plt.show()
```


Häufigkeit, mit der der Messwert auftaucht
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
38.0
38.5
39.0
39.5
40.0
40.5
41.0
Magnetfeld in µT

[ ]: