# Fehlerfortpflanzung

October 19, 2022

## 0.1 Fehlerfortpflanzung

*Notebook erstellt am 04.09.2022 von C. Rockstuhl, überarbeitet von Y. Augenstein*

### 0.1.1 Allgemeine Betrachtungen zur Berechnung der Standardabweichung

In diesem Notebook werden eine einfache Fehlerrechnung durchführen und werden vor allem sehen, dass zur korrekten Berechnung der Standardabweichung es wichtig ist, die Unsicherheit in der Streckenmesseung und der Zeitmessung explizt zu berücksichtigen. Als Daten verwenden wir Messungen von Ort und Zeit, die in einer Datei abgelegt sind.

Als erstes lesen wir die Daten ein. In dieser Datei `Geschwindigkeit.csv` gibt es zwei Spalten. Eine mit der entsprechenden Ortsmessung und eine mit der entsprechenden Zeitmessung. Die Geschwindigkeit ist dann einfach definiert als das Verhältniss von Ort und Zeit

$$v(x,t) = \frac{x}{t}.$$

Beachten Sie bitte, eine solche Messung bzw. die Zuordnung einer solchen Geschwindigkeit macht nur dann Sinn, wenn wir davon ausgehen, dass es sich um eine gleichförmige, also eine konstante, Geschwindigekit handelt.

[1]:
```
import pandas as pd                    # Das Einlesen der Daten erfolgt wieder␣
↪vergleichbar zu dem Vorgehen bei
                                       # der Betrachtung der Mittelwerte und␣
↪der Standardabweichung.
import matplotlib.pyplot as plt

df = pd.read_csv("Geschwindigkeit.csv", sep=",")
print(df.head())

zeit = df['Zeit']
ort = df['Ort']

geschwindigkeit = ort / zeit
```

```
      Zeit      Ort
0  1.00280  0.88743
```

```
1  1.08520  1.07940
2  1.03320  1.18190
3  0.92458  0.67745
4  0.86262  1.37430
```

Ohne diese Daten hier jetzt extra visualisieren zu wollen, rechnen wir zunächst den Mittelwert und die Standardabweichung jeder einzelnen Größe aus. Wir berechnen auch die Geschwindikgkeit aus jedem der einzelnen Wertepaare und können diesen Mittelwert und Standardabweichung in einem ersten (falschen) Ansatz berechnen.

[2]:
```
import numpy as np

sigma_x = np.std(ort, ddof = 1)
mean_x = np.mean(ort)
sigma_t = np.std(zeit, ddof = 1)
mean_t = np.mean(zeit)
print(f"Das abschliessende Ergebniss unserer Ortsmessung beträgt␣
 ↪({round(mean_x, 3)} \u00B1 {round(sigma_x, 3)}) m")
print(f"Das abschliessende Ergebniss unserer Zeitmessung beträgt␣
 ↪({round(mean_t, 3)} \u00B1 {round(sigma_t, 3)}) s")
```

```
Das abschliessende Ergebniss unserer Ortsmessung beträgt (0.999 Âś 0.202) m
Das abschliessende Ergebniss unserer Zeitmessung beträgt (1.0 Âś 0.099) s
```

Wir wollen jetzt die Standardabweichung korrekt ausrechnen. In der Vorlesung habe wir diese berechnet mittels Fehlerfortplflanzung als

$$\sigma_v = \sqrt{\left(\frac{\partial v(x,t)}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial v(x,t)}{\partial t}\right)^2 \sigma_t^2}$$

Im einfachen Fall können wir die partiellen Ableitungen natürlich direkt ausrechnen und wir erhalten

$$\sigma_v = \sqrt{\frac{1}{t^2}\sigma_x^2 + \frac{x^2}{t^4}\sigma_t^2}$$

.

Zur Berechnung der Standardabweichung können wir natürlich für diese einfache Gleichung die analytische Lösung der Ableitung benutzen und diese evaluieren.

[3]:
```
mean_v = np.mean(ort) / np.mean(zeit)
sigma_v = np.sqrt((1 / mean_t * sigma_x) ** 2 + (mean_x / mean_t**2 *␣
 ↪sigma_t) ** 2)
print(f"Unsere gemessene Geschwindigkeit beträgt ({mean_v:.3f} \u00B1␣
 ↪{sigma_v:.3f}) m/s")
```

```
Unsere gemessene Geschwindigkeit beträgt (0.999 Âś 0.225) m/s
```

Wenn wir die Standardabweichungen der Geschwindigkeit einfach aus den einzelnen Wertepaare berechnen, kommen wir nicht auf das richtige Ergebnis. Der Unterschied ist in unserem Fall nicht

ganz so dramatisch, aber Sie werden in Ihrem Studium sicherlich Fälle beobachten, in denen der Unterschied signifikant werden wird.

[4]:
```
sigma_v_falsch = np.std(geschwindigkeit, ddof = 1)
mean_v_falsch = np.mean(geschwindigkeit)
print(f"Unsere gemessene Geschwindigkeit mit der falsch berechneten␣
↪Standardabweichung beträgt ({mean_v_falsch:.3f} \u00B1 {sigma_v_falsch:.
↪3f}) m/s")
```

```
Unsere gemessene Geschwindigkeit mit der falsch berechneten Standardabweichung
beträgt (1.009 Âś 0.229) m/s
```

### 0.1.2 Symbolische Berechnung der Ableitungen

Technisch haben wir das Ziel dieses Notebooks erreicht. Praktisch möchten wir hier aber noch einen Schritt weitergehen und jegliche Rechnung den Computer überlassen. Insbesondere die Berechnung der pariellen Ableitungen kann unter Umständen so komplex sein, dass wir dies nicht mehr ohne größere Probleme analytisch auf dem Papier durchführen können. Stattdessen möchten Sie einen Computer verwenden, der diese Aufgabe für Sie übernimmt. Diese Rechnung soll in diesem Teil des Notebooks durchgeführt werden. Hierfür verwenden wir die Bibliothek zum Symbolischen Rechnen.

[5]:
```
import sympy
from sympy.abc import x, t
```

[6]:
```
v = x / t
Ableitung_t = sympy.diff(v, t)
Ableitung_x = sympy.diff(v, x)
print(f"Die partielle Ableitung der Geschwindigkeit nach der Zeit lautet␣
↪{Ableitung_t}")
print(f"Die partielle Ableitung der Geschwindigkeit nach dem Ort lautet␣
↪{Ableitung_x}")
```

```
Die partielle Ableitung der Geschwindigkeit nach der Zeit lautet -x/t**2
Die partielle Ableitung der Geschwindigkeit nach dem Ort lautet 1/t
```

Die Ableitungen lassen sich auch entsprechend evaluieren bei konkreten Funktionswerten. Zur Berechnung der Fehler sei gesagt, dass diese partiellen Ableitungen beim Mittelwert evaluiert werden.

[7]:
```
round(Ableitung_x.evalf(subs={x: mean_x, t: mean_t}), 3)
```

[7]:
1.0

Wir können dann die Standardabweichung mit der oben angegebenen Formel

$$\sigma_v = \sqrt{\left(\frac{\partial v(x,t)}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial v(x,t)}{\partial t}\right)^2 \sigma_t^2}$$

ganz automatisiert berechnen.

[8]:
```
sigma_v2 = sympy.sqrt(
    (Ableitung_x.evalf(subs={x: mean_x, t: mean_t}) * sigma_x) ** 2
    + (Ableitung_t.evalf(subs={x: mean_x, t: mean_t}) * sigma_t) ** 2
)
print(f"Unsere gemessene Geschwindigkeit beträgt ({mean_v:.3f} \u00B1␣
↪{sigma_v2:.3f}) m/s")
                                    # Beachten Sie bitte als kleine␣
↪Abweichung, dass wir hier den Wurzelbefehl in
                                    # SciPy verwenden müssen und nicht den␣
↪aus NumPy.
```

```
Unsere gemessene Geschwindigkeit beträgt (0.999 Âś 0.225) m/s
```

Als Letztes sei noch gesagt, dass wenn eine Funktion das Produkt oder der Quotient zweier physikalischer Größen ist, dann ist der relative Fehler dieser Größe gerade die Wurzel der Summe der relativen Fehler der beiden physikalischen Größen. In unserem Fall wäre das also

$$\frac{\sigma_v}{\bar{v}} = \sqrt{\left(\frac{\sigma_x}{\bar{x}}\right)^2 + \left(\frac{\sigma_t}{\bar{t}}\right)^2}$$

[9]:
```
relativer_fehler_1 = sigma_v / mean_v
relativer_fehler_2 = np.sqrt((sigma_x / mean_x) ** 2 + (sigma_t / mean_t) **␣
↪2)
print(f"Der direkt berechnete relative Fehler (einmal hier mit allen␣
↪Nachkommastellen) {relativer_fehler_1}")
print(f"Der indirelt berechnete relative Fehler (einmal hier mit allen␣
↪Nachkommastellen) {relativer_fehler_2}")
```

```
Der direkt berechnete relative Fehler (einmal hier mit allen Nachkommastellen)
0.22524160087399744
Der indirelt berechnete relative Fehler (einmal hier mit allen Nachkommastellen)
0.22524160087399742
```

Beachten Sie hier eine kleine Abweichung in der 16'ten Nachkommastelle. Dieser Fehler ist gerade remineszent zu unserer numerischen Diskretisierung der Funktionsevaluation, die in double-precission erfolgt. Hier werden gerade 16 Nachkommastellen mit berücksichtigt.