

# Laborbuch

October 28, 2022

# Contents

<b>1 Laborbuch</b>	<b>3</b>
1.1 Anleitung	3
1.1.1 Textzellen	3
1.1.2 Codezellen	3
1.1.3 Export als PDF	5
1.2 Anwendungsbeispiele	6
1.2.1 Definition des Mittelwertes	6
1.2.2 Definition Instrumenteller Gewichtungsfaktor	6
1.2.3 Definition des gewichteten Mittelwertes	6
1.2.4 Definition Unsicherheit des gewichteten Mittelwertes	6
1.2.5 Beispiel 1	7
1.2.6 Definition Pythagoreische Addition	7
1.2.7 Definition Empirische Standardabweichung	7
1.2.8 Definition Vertrauensbereich ( $n \geq 6$ )	7
1.2.9 Definition Größtfehlerabschätzung ( $n < 6$ )	8
1.2.10 Kochrezept Messwertberechnung	8
1.2.11 Beispiel 2	8
1.2.12 Definition Unkorrelierte Gaußsche Fehlerfortpflanzung	9
1.2.13 Definition: Zusammenhang Korrelation, Kovarianz und Unsicherheit	9
1.2.14 Definition Korrelierte Fortpflanzung (wtf)	10
1.3 Versuch 1	10

# 1 Laborbuch

---

## 1.1 Anleitung

Das hier ist ein sogenanntes Notebook, welches auf meinem Google-Drive-Account gespeichert ist. Zum Einlesen von Dateien ist es notwendig, dass eine Kopie dieses Notebooks auf deinem Google-Drive Account gespeichert wird. Du hast über einen Link volle Zugriffsrechte bekommen. Hier kann Text und Code geschrieben werden, Daten können eingelesen und Graphiken erstellt werden. Das ganze ist vom Handling her ein wenig wie Word und R-Markdown, aber ganz anders.

Notebooks bestehen aus Zellen die aneinandergereiht werden können. In diesen wird der Inhalt platziert. Dieser Text steht zum Beispiel in einer Zelle. Diese können durch Doppelklick auf die Zelle geöffnet und bearbeitet werden. Neue Zellen können oben links mit + Code bzw. + Text eingefügt werden. Textzellen sind für Text, Codezellen für Programme.

### 1.1.1 Textzellen

Es ist möglich mit neuen Textzellen Text zu schreiben. Dieser lässt sich quasi beliebig formatieren. Zum Beispiel lassen sich mit der Raute (#) Überschriften hinzufügen. Die Schriftart kann **verändert** werden.

Die Formelschreibweise ist die Latex-Schreibweise. Das sind Kommandos, wie zum Beispiel der zum Schreiben eines Bruches:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Oder Integralzeichen:

$$\int_1^3$$

Es sind alle mathematischen Symbole verfügbar, die entsprechenden Kommandos sind bei Wikipedia in der Liste mathematischer Symbole zu finden, siehe [Wikipedia: Liste mathematischer Symbole](#).

### 1.1.2 Codezellen

Neben den Textzellen existieren Code-Zellen. Diese können mit einem Klick auf den Play-Button (oben links in der Zelle) abgespielt werden. Das Ausführen dauert beim erstenmal ein paar Sekunden, weil das Programm nicht auf dem Computer direkt läuft, sondern bei Google.

In Python läuft viel über Module (das was in R *libraries* sind). Pandas ist so ein Modul, speziell für die Dataframes. Es wird zu Beginn eines Programms importiert:

```
[11]: import pandas as pd # Der erste Python-Code :)
```

Werden weitere Module benötigt können diese auf die gleiche Weise importiert werden. Dann kann mit der Datenverarbeitung begonnen werden. Dateien sind in einem GitHubrepository gespeichert. Sollen Dateien hier eingefügt werden, empfiehlt sich die Speicherung in dieses Repository (anderes Thema). Um auf diese hier zugreifen zu können wird der folgende Code benötigt. Dieser legt eine Kopie des Github-Verzeichnisses im Google-Drive Konto des Nutzers (Du und dein Google-Account, nicht wundern...) ab. Das GitHub-Repository ist: [Link](#).

```
[ ]: # Importieren zweier Module
from os.path import join
from google.colab import drive
# Def Root-Verzeichnis
ROOT = "/content/drive/"
# Def Drive-intern Pfad
```

```

MY_GOOGLE_DRIVE_PATH = 'MyDrive/Laborbuch/'
# Def git-token, git-username, git-Repository
GIT_TOKEN = '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'
GIT_USERNAME = 'JustusWeyers'
GIT_REPOSITORY = 'Laborbuch-im-Physikpraktikum'
# Def Projekt Pfad
PROJECT_PATH = join(ROOT, MY_GOOGLE_DRIVE_PATH)
# Def Pfad des geklonten Repository's
REPO_PATH = join(PROJECT_PATH, GIT_REPOSITORY)
# Def git-Adresse
GIT_PATH = f"https://{GIT_TOKEN}@github.com/{GIT_USERNAME}/{GIT_REPOSITORY}.git"

# Einbinden des Nutzer-Drives
drive.mount(ROOT)

# Anlegen des Projektverzeichnis auf dem Drive
!mkdir "{PROJECT_PATH}"
# Klonen des GitHub-Repository's
%cd '{PROJECT_PATH}'
!git clone "{GIT_PATH}"

# Definition einer Aktualisierungsfunktion
def refreshRepository(path):
    %cd path
    !rm -r Laborbuch-im-Physikpraktikum
    !git clone "{GIT_PATH}"
refreshRepository(REPO_PATH)

```

Das Einlesen aus dem mit dem vorigen Code angelegten Verzeichnis funktioniert dann in etwa so:

```

[12]: # Einlesen der Datei
filename = 'Bodenkundedaten.txt'
BoKu = pd.read_csv(f'{REPO_PATH}/INPUT/{filename}', sep=';')
print(BoKu.iloc[14:17,[0, 11, 12]]) # Ausgabe. Mit .iloc[Zeilen , Spalten]

```

	Probe	Volumen[cm <sup>3</sup> ]	Ksat[cm/s]
14	G3P5	247.63	29.83
15	G4P1	250.49	11.65
16	G4P2	256.00	6.80

Auch R-Code ist hier bis zu einem gewissen Grad möglich. Zunächst muss der folgende Befehl ausgeführt werden, der R verfügbar macht:

```

[27]: %load_ext rpy2.ipynon

```

The rpy2.ipynon extension is already loaded. To reload it, use:

```

%reload_ext rpy2.ipynon

```

Dann können nach einem %%R mehrere Zeilen R-Code stehen.

```

[28]: %%R
wd <- '/content/drive/MyDrive/Laborbuch/Laborbuch-im-Physikpraktikum/INPUT/'
filename <- paste(wd, 'Bodenkundedaten.txt', sep='')
Boku <- read.table(filename, sep=';', dec='.', header=TRUE)

```

```
#print(colnames(Boku))
boxplot(Boku$Ksat.cm.s.[Boku$Standort=='Kiefernwald'], horizontal=TRUE)
```

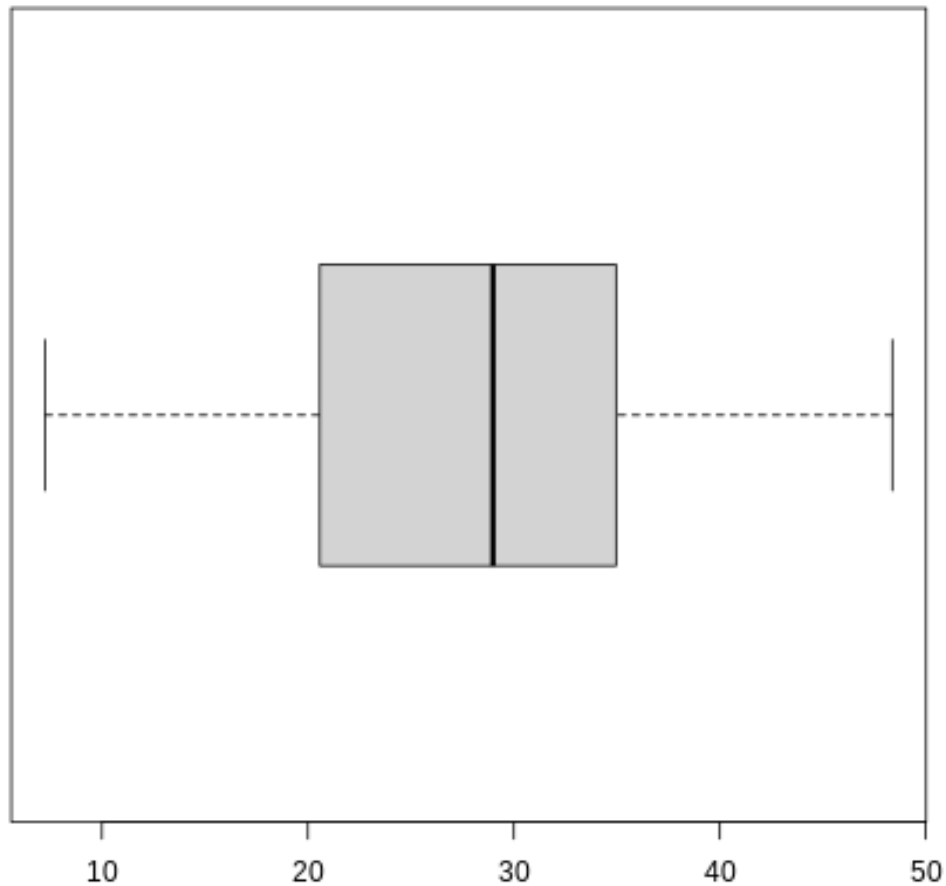


Figure 1: Caption

### 1.1.3 Export als PDF

Mit dem Befehl: `jupyter nbconvert /path/to/mynotebook.ipynb --to latex` kann ein lokal gespeichertes Jupyter-Notebook in ein tex-Dokument konvertiert werden.

## 1.2 Anwendungsbeispiele

Im Folgenden ein paar Rechenbeispiele. Es werden wieder drei Module importiert: \* math ist das Python-Mathemodul, \* numpy ist für das Rechnen mit Matrizen, \* statistics ist ein Modul für statistische Funktionen.

Diese werden aus Bequemlichkeit mit dem Befehl *as* mit Kürzeln importiert. Aus dem Paket *sympy* werden zudem alle funktionen direkt importiert, damit können alle Arten von Ableitungen durchgeführt werden.

Quelle: HU Berlin

```
[17]: import math as m
import numpy as np
import statistics as stat

from sympy import *
```

### 1.2.1 Definition des Mittelwertes

$$\bar{x} = \frac{1}{n} \sum_{i=0}^n x_i \quad (1.1)$$

\*  $\bar{x}$ : Arithmetischer Mittelwert \*  $n$ : Anzahl aller (Mess)Werte \*  $x_i$ :  $i$ -ter (Mess)Wert

```
[5]: ### Minibeispiel ###
# Berechnung Ar. Mittelwert (MW)
messwerte = [1.0456, 0.9774, 1.0023, 0.9904, 1.3995]
# funktion 'mean' des Moduls 'statistics'
stat.mean(messwerte)
```

[5]: 1.08304

### 1.2.2 Definition Instrumenteller Gewichtungsfaktor

$$p_i = \frac{1}{u_i^2} \quad (1.2)$$

- $p_i$ : Gewichtungsfaktor des Wertes ( $x_i \pm u_i$ )
- $u_i$ : Unsicherheit des Wertes ( $x_i \pm u_i$ )

### 1.2.3 Definition des gewichteten Mittelwertes

$$\bar{x} = \frac{\sum_i^n p_i * x_i}{\sum_{i=0}^n p_i} \quad (1.3)$$

\*  $\bar{x}$ : Gewichtungsfaktor des Wertes ( $x_i \pm u_i$ ) \*  $n$ : Anzahl aller Werte \*  $x_i$ :  $i$ -ter Wert \*  $i$ -ter Gewichtungsfaktor

### 1.2.4 Definition Unsicherheit des gewichteten Mittelwertes

$$\bar{u} = \pm \sqrt{\frac{C}{\sum_i^n p_i}} \Rightarrow \bar{u} = \pm \left( \sum_i^n p_i \right)^{-\frac{1}{2}} \quad (1.4)$$

- $\bar{u}$ : Unsicherheit des gewichteten Mittelwertes
- $n$ : Anzahl aller Werte
- $C$ : frei wählbarer Proportionalitätsfaktor
- $p_i$ :  $i$ -ter Gewichtungsfaktor

### 1.2.5 Beispiel 1

```
[8]: # Werte für die gemessene Erdbeschleunigung
g = np.array([9.81, 9.79, 9.80, 9.60])
# Bestimmte Unsicherheit
u = np.array([0.03, 0.11, 0.04, 0.70])
# Berechnung des Gewichtungsfaktors p_i (1.2)
p = 1.0 / u**2
# Liste [gewichteter MW (1.3), Unsicherheit des gewichteten MW (1.4)]
[sum(g * p) / sum(p), 1.0 / np.sqrt(sum(p))]
```

```
[8]: [9.805424275180432, 0.023435233683447708]
```

### 1.2.6 Definition Pythagoreische Addition

$$u = \sqrt{\sum_i^n u_i^2} \quad (1.5)$$

- $u$ : Pythagoreisch addierter Wert
- $n$ : Anzahl aller Werte
- $u_i$ :  $i$ -ter Wert

### 1.2.7 Definition Empirische Standardabweichung

$$\sigma = \sqrt{\frac{1}{n-1} \sum_i^n (x_i - \bar{x})^2} \quad (1.6)$$

- $\sigma$ : Standardabweichung
- $n$ : Anzahl aller Messwerte
- $x_i$ :  $i$ -ter Messwert
- $\bar{x}$ : Arithmetischer Mittelwert der Messwerte

```
[15]: ### Minibeispiel ###
# berechnung Standardabweichung
g = [9.81279, 9.83616, 9.76557, 9.78496, 9.84230,
9.75096, 9.72767, 9.84866, 9.74724, 9.76847]
stat.stdev(g)
```

```
[15]: 0.04362286092813679
```

### 1.2.8 Definition Vertrauensbereich ( $n \geq 6$ )

$$\bar{s} = \pm t * \frac{\sigma}{\sqrt{n}} \quad (1.7)$$

- $\bar{s}$ : Vertrauensbereich
- $\sigma$ : Standardabweichung
- $n$ : Anzahl aller Messwerte
- $t$ : Student T-Wert

```
[25]: # Python t-Wert
from scipy import stats
stats.t.ppf(1-0.025, 14)
```

```
[25]: 2.1447866879169273
```

### 1.2.9 Definition Größtfehlerabschätzung ( $n < 6$ )

$$\bar{s} \approx \pm \max_i |x_i - \bar{x}| \quad (1.8)$$

- $\bar{s}$ : Abgeschätzter Vertrauensbereich
- $x_i$ :  $i$ -ter Messwert
- $\bar{x}$ : Mittelwert

```
[29]: ### Minibeispiel ###
# Vertrauensbereich (Berechnung ohne t-Wert)
g = [9.81279, 9.83616, 9.76557, 9.78496, 9.84230,
     9.75096, 9.72767, 9.84866, 9.74724, 9.76847]
[np.std(g, ddof=1) / np.sqrt(len(g)), max(abs(g - np.mean(g)))]
```

```
[29]: [0.013794759858567901, 0.060808000000000153]
```

### 1.2.10 Kochrezept Messwertberechnung

1. Mittelwert berechnen (1.1)

$$\bar{x} = \frac{1}{n} \sum_{i=0}^n x_i \quad (1)$$

2. Bekannte systematische Fehler korrigieren (sehr selten)

$$\bar{x}_k = \bar{x} - u_{sys} \quad (1.9)$$

3. Vertrauensbereich berechnen (1.6) (1.7) (1.8)

$$\bar{s} = \pm \begin{cases} t * \frac{\sigma}{\sqrt{n}} = t * \sqrt{\frac{1}{n(n-1)} \sum_i^n (x_i - \bar{x})^2} & n \geq 6 \\ \bar{s} \approx \pm \max_i |x_i - \bar{x}| & n < 6 \end{cases} \quad (2)$$

4. Messunsicherheit durch pythagoreisches Addieren aller Unsicherheiten berechnen (1.5)

$$u = \sqrt{\bar{s}^2 + u_{\text{gert}}^2 + u_{\text{ablesen}}^2 + \dots} \quad (3)$$

5. Ergebnis auf signifikante Stellen runden, mit Einheit (und Zehnerpotenz oder SI- Präfix) angeben.

### 1.2.11 Beispiel 2

Im Pantheon in Paris liegen neben berühmten Franzosen auch einige berühmte Physiker begraben. Ebenfalls hängt dort ein 67 m langes Foucaultsches Pendel in der Hauptkuppel herunter. Es wurden 17 Periodendauern mit einem Handy gemessen und zu einem Messwert berechnet. Dabei wurden angenommen: zweimal 0.15 s Unsicherheit in der Reaktionszeit, 10 ms Ableseungenauigkeit und  $10^{-4}$  t systematische Unsicherheit der Handyuhr.

```
[31]: # 0. Messreihe
tMess = [16.38, 16.66, 16.54, 16.38, 16.31, 16.23, 16.56, 16.52, 16.32,
         16.48, 16.41, 16.32, 16.38, 16.34, 16.27, 16.35, 16.44]

# 1. Mittelwertbildung
t = np.mean(tMess)

# 2. Bias
pass
```



```
# 3. Vertrauensbereich
statU = np.std(tMess, ddof=1) / np.sqrt(len(tMess))

# 4. Messunsicherheit durch pythagoreisches Addieren
reaktU = 0.15 #Reaktionszeit
ableseU = 0.01 #Ableseungenauigkeit
systU = 0.0001 * t #syst. Unsicherheit Uhr
ut = np.sqrt(statU**2 + 2 * reaktU**2 + ableseU**2 + systU**2)

# 5. Runden (Muss noch gemacht werden)
print([t, ut])
```

```
[16.40529411764706, 0.21417381936353636]
```

### 1.2.12 Definition Unkorrelierte Gaußsche Fehlerfortpflanzung

$$u_f = \sqrt{\sum_{i=0}^n \left( \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_i} * u_i \right)^2} \quad (1.10)$$

- $u_f$ : Fortgepflanzte Unsicherheit der berechneten Funktion  $f(x_1, x_2, \dots, x_n)$
- $n$ : Anzahl aller Werte
- $x_i$ :  $i$ -te Variable/Wert
- $u_i$ :  $i$ -te Unsicherheit

In Python, können die Ableitungen mit dem Package SymPy automatisch berechnet werden und die Fehlerfortpflanzung manuell eingetippt werden.

```
[32]: u, i, uu, ui = symbols("u,i,uu,ui")
r = u / i
ur = sqrt((diff(r, u) * uu)**2 + (diff(r, i) * ui)**2)
ur.subs([(u, 238.46), (uu, 7.34), (i, 0.9239), (ui, 0.0081)])
```

```
[32]: 8.26055469654989
```

### 1.2.13 Definition: Zusammenhang Korrelation, Kovarianz und Unsicherheit

$$C_{i,j} = \frac{u_{i,j}}{u_i + u_j} \quad (1.11)$$

- $C_{i,j}$ : Korrelationskoeffizient  $C_{i,j} \in [-1, +1]$  der Werte  $x_i$  und  $x_j$
- $u_{i,j}$ : Kovarianz der Werte  $x_i$  und  $x_j$
- $u_i$ :  $i$ -te Unsicherheit des  $x_i$  Werts
- $u_j$ :  $j$ -te Unsicherheit des  $x_j$  Werts Komplizierte händische Berechnung. Python:

```
[37]: x_simple = np.array([-3.49, 2.33, 0.63, 2.8, -4.72, -1.84, 1.81, 0.36, -1.99, -0.65])
y_simple = np.array([0.78, -0.53, -0.28, -0.39, 0.75, 0.60, -0.45, 0.05, 0.42, 0.12])
my_rho = np.corrcoef(x_simple, y_simple)

print(my_rho)
```

```
[[ 1.          -0.9616651]
 [-0.9616651  1.          ]]
```

### 1.2.14 Definition Korrelierte Fortpflanzung (wtf)

$$u_f = \sqrt{\sum_i^n \left( \frac{\partial f(x_1, \dots)}{\partial x_i} * u_i \right)^2 + \sum_i^{n-1} \sum_{j=i+1}^n \frac{\partial f(x_1, \dots)}{\partial x_i} * \frac{\partial f(x_1, \dots)}{\partial x_j} * u_{i,j}} \quad (1.12)$$

- $u_f$ : Fortgepflanzte Unsicherheit der berechneten Funktion  $f(x_1, x_2, \dots, x_n)$
- $n$ : Anzahl aller Werte
- $x_i$ :  $i$ -te Variable/Wert
- $u_i$ :  $i$ -te Unsicherheit
- $u_{i,j}$ : Kovarianz der Werte  $x_i$  und  $x_j$

Definition über Kovarianzmatrix  $(\underline{U})_{i,j} = u_{i,j}$ :

$$u_f = \sqrt{\nabla^T f * \underline{U} * \nabla f} \quad (1.12)$$

```
[38]: u, i, uu, ui, uui = symbols("u,i,uu,ui,uui")
      r = u / i
      ur = sqrt((diff(r, u) * uu)**2 + (diff(r, i) * ui)**2 +
      2 * diff(r, u) * diff(r, i) * uui)
      ur.subs([(u, 238.46), (uu, 7.34), (i, 0.9239),
      (ui, 0.0081), (uui, -0.0545)])
```

```
[38]: 10.0595845339953
```

---

## 1.3 Versuch 1

*Hier könnte Ihr Versuch stehen*

```
[ ]:
```