Laborbuch_30102022_09

October 30, 2022

Contents

List of Figures

1 Laborbuch

1.1 Anleitung

Das hier ist ein sogenanntes Notebook. Du hast über einen Link volle Zugriffsrechte bekommen. Hier kann Text und Code geschrieben werden, Daten können eingelesen und Grafiken 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 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 zu schreiben. Dieser lässt sich beliebig formatieren. Zum Beispiel lassen sich mit # Überschriften hinzufügen. Die Schriftart kann **verän***d**ert*** werden.

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

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

Es ist eine quasi unerschöpfliche Menge an Zeichen verfügbar, die entsprechenden Kommandos sind bei Wikipedia in der ßListe mathematischer Symbole zu finden. Dort findet sich eine Spalte, in der der Latex-Befehl für jedes Zeichen aufgelistet ist.

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 ersten Mal ein paar Sekunden, weil das Programm nicht auf dem Computer läuft, sondern Google diesen Part übernimmt. Grüße an der Stelle.

Der Programmierspaß kann also sofort beginnen, sobald eine Code-Zelle eingefügt wurde!

In Python läuft viel über Module (das was in R *libraries* sind) und der erste Code ist, solch ein Modul einzubinden. Pandas ist ein Modul, speziell für die Arbeit mit Dataframes. Es wird zu Beginn eines Programms importiert:

[2]:
```
import pandas as pd
```

Jetzt kann das Modul verwendet werden. Werden weitere Module benötigt, können diese auf die gleiche Weise importiert werden.

Sollen Daten verarbeitet werden, müssen die ebenfalls hier eingebunden werden. Als Sammlung für die erstellten Messreihen steht ein öffentliches GitHub-Repository zur Verfügung ß zu GitHub .

Das Einlesen der Dateien aus diesem Repository gestaltet sich erschreckend einfach:

[11]:
```
# Einlesen der Datei über GitHub-Addresse
url = 'https://raw.githubusercontent.com/JustusWeyers\
/Laborbuch-im-Physikpraktikum/main/INPUT/Bodenkundedaten.txt'
# Einlesen der url mit dem pandas-Modul (pd)
BoKuBeispieldaten = pd.read_csv(url, sep=';')
# Ausschnitt des Dataframes
print(BoKuBeispieldaten.iloc[14:17,[0, 11, 12]])
```

```
   Probe  Ksat[cm/s]  Wasserrgehalt[%]
14  G3P5       29.83              3.60
15  G4P1       11.65              7.99
16  G4P2        6.80              3.55
```

In diesem Notebook lassen sich Dateien temporär speichern, neben der Möglichkeit das im Explorer (links) zu tun, besteht die Möglichkeit das aus dem Python-Code heraus abzuwickeln. Dazu muss unter Umständen erst ein passendes, neues Verzeichnis angelegt werden. Als Speicherort ist das content-Verzeichnis vorgesehen.

[4]:
```
# Neues Verzeichnis
%mkdir /content/DATA
# Speichern eines Dataframes als csv-Datei in das Notebook
BoKuBeispieldaten.to_csv('/content/DATA/BokuBeispieldaten.csv', sep=';')
```

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

[5]:
```
%load_ext rpy2.ipython
```

Dann können nach einem %%R mehrere Zeilen R-Code stehen. In diesem Beispiel wird die eben in das Notebook gespeicherte csv-Datei mit den Bodenkundewerten durch R eingelesen und K_{sat} für den Standort *Kiefernforst* geplottet.

[8]:
```
%%R
# Einlesen der Datei BokuBeispieldaten.csv aus dem Notebook-Ordner /content/DATA
dateiname <- '/content/DATA/BokuBeispieldaten.csv'
Boku <- read.table(dateiname, sep=';', dec='.', header=TRUE)
# print(colnames(Boku))
boxplot(Boku$Ksat.cm.s.[Boku$Standort=='Kiefernwald'],
        main='Gesättigte Hydraulische Leitfähigkeit',
        xlab='K_sat in cm/s',
        col='firebrick3',
        horizontal=TRUE)
```

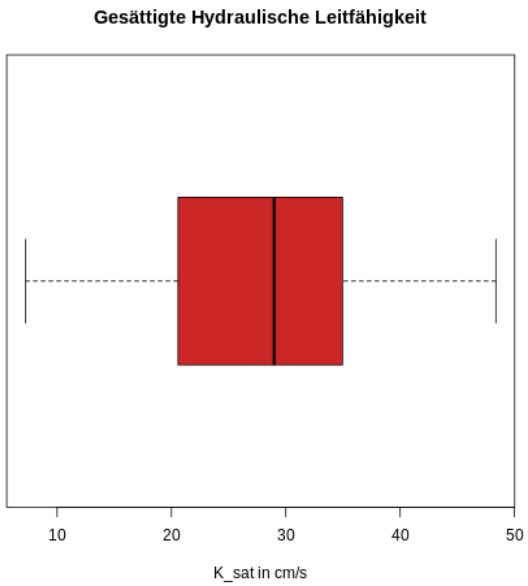

Figure 1: K_{sat} in $\frac{cm}{s}$

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 bearbeitbares tex-Dokument konvertiert werden.

Das Ergebnis eines alten Arbeitsstandes sieht dann so aus: ß PDF.

1.2 Speichern von Änderungen

Das Speichern von änderungen geschieht über: Datei ß Kopie in GitHub speichern $\rightarrow$ *OK*.

Dazu ist es erforderlich, dass du mit deinem GitHub-Account angemeldet bist. Über diesen Account hast du SChreibrechte für das GitHub-Repository erhalten. Die Änderungen überschreiben dort die Datei ßLaborbuch.ipynb. Ältere Arbeitsstände sind sowohl lokal, als auch in dem Verzeichnis ß Sicherheitskopien hinterlegt.

1.3 Anwendungsbeispiele

Im Folgenden ein paar Rechenbeispiele. Es werden zunächst wieder 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 Ableitungen durchgeführt werden.

Quelle: → HU Berlin

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

from sympy import *
```

1.3.1 Definition des Mittelwertes

$$\bar{x} = \frac{1}{n} \sum_{i=0}^{n} x_i \tag{1}$$

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

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

[ ]:
```
1.08304
```

1.3.2 Definition Instrumenteller Gewichtungsfaktor

$$p_i = \frac{1}{u_i^2} \tag{2}$$

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

1.3.3 Definition des gewichteten Mittelwertes

$$\bar{x} = \frac{\sum_{i}^{n} p_i * x_i}{\sum_{i=0}^{n} p_i} \tag{3}$$

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

1.3.4 Definition Unsicherheit des gewichteten Mittelwertes

$$\bar{u} = \pm \sqrt{\frac{C}{\sum_{i}^{n} p_i}} \Rightarrow \bar{u} = \pm (\sum_{i}^{n} p_i)^{-\frac{1}{2}} \tag{4}$$

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

1.3.5 Beispiel 1

[ ]:
```
# 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))]
```

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

1.3.6 Definition Pythagoreische Addition

$$u = \sqrt{\sum_{i}^{n} u_i^2} \tag{5}$$

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

1.3.7 Definition Empirische Standardabweichung

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i}^{n} (x_i - \bar{x})^2)} \tag{6}$$

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

[ ]:
```
### 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)
```

[ ]: 0.04362286092813679

1.3.8 Definition Vertrauensbereich ($n \geq 6$)

$$\bar{s} = \pm t * \frac{\sigma}{\sqrt{n}} \tag{7}$$

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

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

[ ]: 2.1447866879169273

1.3.9 Definition Größtfehlerabschätzung ($n < 6$)

$$\bar{s} \approx \pm \max_i |x_i - \bar{x}| \tag{8}$$

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

[ ]:
```
### 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)))]
```

[ ]: [0.013794759858567901, 0.06080800000000153]

1.3.10 Kochrezept Messwertberechnung

1. Mittelwert berechnen

$$\bar{x} = \frac{1}{n} \sum_{i=0}^{n} x_i \tag{9}$$

2. Bekannte systematische Fehler korrigieren(sehr selten)

$$\bar{x}_k = \bar{x} - u_{sys} \tag{10}$$

3. Vertrauensbereich berechnen

$$\bar{s} = \pm \begin{cases} t * \frac{\sigma}{\sqrt{n}} = t * \sqrt{\frac{1}{n(n-1)} \sum_i^n (x_i - \bar{x})^2} & n \geq 6 \\ \bar{s} \approx \pm \max_i |x_i - \bar{x}| & n < 6 \end{cases} \tag{11}$$

4. Messunsicherheit durch pythagoreisches Addieren aller Unsicherheiten berechnen

$$u = \sqrt{\bar{s}^2 + u_{gert}^2 + u_{ablesen}^2 + ...} \tag{12}$$

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

1.3.11 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 Re- aktionszeit, 10 ms Ableseungenauigkeit (1dgt.) und 10-4 · t systematische Unsicherheit der Handyuhr.

[ ]:
```
# 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. Messunsuicherheit 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]
```

1.3.12 Definition Unkorrelierte Gaußsche Fehlerfortpflanzung

$$u_f = \sqrt{\sum_{i=0}^{n} (\frac{\partial f(x_1, x_2, ..., x_n)}{\partial x_i} * u_i)^2} \tag{13}$$

- u_f: Fortgepflanzte Unsicherheit der berechneten Funktion $f(x_1, x_2, ..., x_n)$
- 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.

[ ]:
```
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)])
```

[ ]:
8.26055469654989

1.3.13 Definition: Zusammenhang Korrelation, Kovarianz und Unsicherheit

$$C_{i,j} = \frac{u_{i,j}}{u_i + u_j} \tag{14}$$

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

[ ]:
```
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.       ]]
```

1.3.14 Definition Korrelierte Fortpflanzung (wtf)

$$u_f = \sqrt{\sum_i^n (\frac{\partial f(x_1, ...}{\partial x_i} * u_i)^2 + \sum_i^{n-1} \sum_{j=i+1}^{n} \frac{\partial f(x_1, ...)}{\partial x_i} * \frac{\partial f(x_1, ...)}{\partial x_j} * u_{i,j}} \tag{15}$$

- u_f: Fortgepflanzte Unsicherheit der berechneten Funktion $f(x_1, x_2, ..., x_n)$
- 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 $(\underline{U})_{i,j} = u_{i,j}$:

$$u_f = \sqrt{\nabla^T f * \underline{U} * \nabla f} \tag{16}$$

[ ]:
```
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)])
```

[ ]: 10.0595845339953

1.4 Versuch 1

Hier könnte Ihr Versuch stehen

Auch Quefrverweise auf in Colab erstellte Formeln sind hier möglich. Siehe Definition des Mittelwertes in (1.3.1).