Fehler 1. und 2. Art Vertrauensintervalle

Peter Büchel

HSLU I

Stat: Block 07

Fehler Hypothesentest

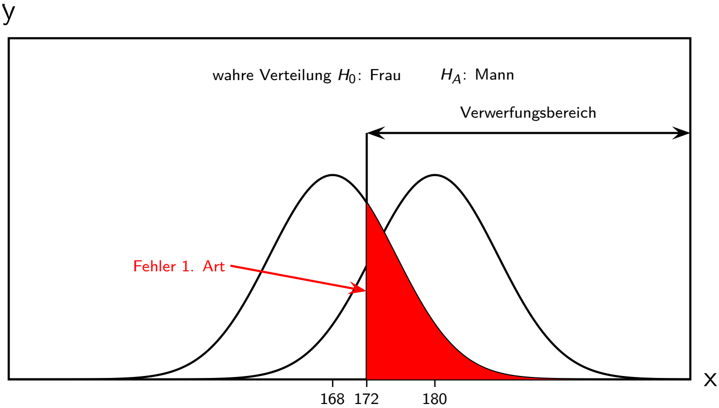
- Schema:

Wahrheit \ Entscheidung	H_0	H_A
H_0	✓	Fehler 1. Art
H_A	Fehler 2. Art	✓

- Entscheidung für H_0, aber H_A wäre richtig $\longrightarrow$ Fehler 2. Art
- Entscheidung für H_A, aber H_0 wäre richtig $\longrightarrow$ Fehler 1. Art

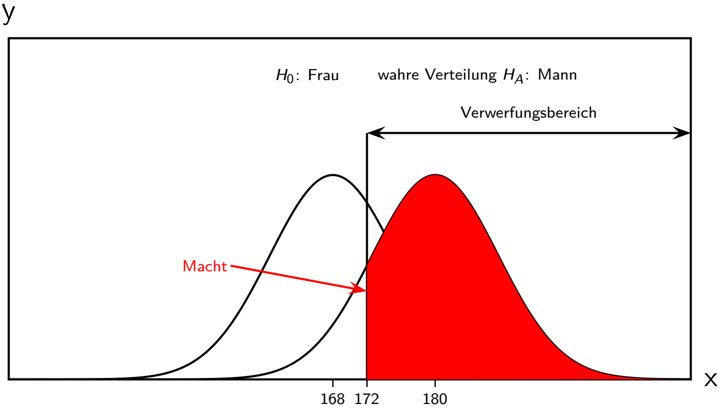
Fehler 1. Art

- Entscheidung für H_A, aber H_0 wäre richtig $\longrightarrow$ Fehler 1. Art
- Entspricht gerade Signifikanzniveau
- Skizze:



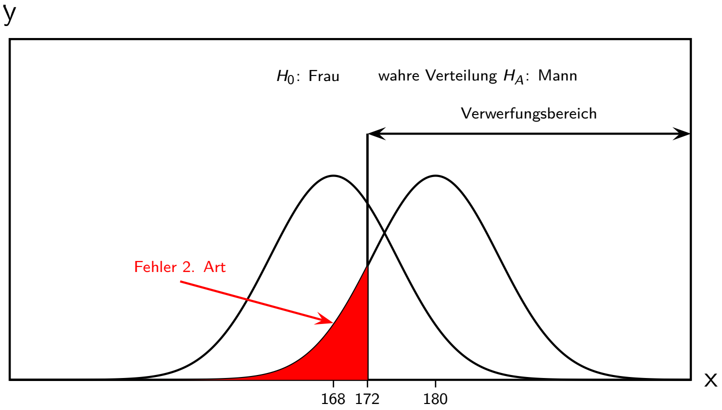
Macht

- H_A wird angenommen und H_A richtig $\quad\rightarrow\quad$ das was wir wollen
- Der wahre Parameter für H_A muss bekannt sein $\quad\rightarrow\quad$ hier $\mu_A = 180$
- Skizze:



Fehler 2. Art

- Entscheidung für H_0, aber H_A wäre richtig $\longrightarrow$ Fehler 2. Art
- Der wahre Parameter für H_A muss bekannt sein $\quad\rightarrow\quad$ hier $\mu_A = 180$
- Fehler 2. Art $= 1 -$ Macht
- Skizze:



Welche Fehlerart ist wichtiger?

- Fehler 1. Art hat traditionell mehr Gewicht als Fehler 2. Art
- Wissenschaftler arbeiten genau und haben Angst, einen Humbug zu publizieren, der sich dann als falsch herausstellt
- Denn wenn Wissenschaftler einen Effekt (signifikante Abweichung von Nullhypothese) beobachten, möchten sie sicher sein, dass es sich nicht bloss um Zufall handelt
- Fehler 1. Art soll vermieden werden
- Nimmt in Kauf, dass man manchmal wichtigen Effekt verpasst
- Fehler 2. Art ist also zweitrangig

- Fehler 1. Art wird direkt kontrolliert durch Konstruktion eines Tests, indem Signifikanzniveau α möglichst klein gehalten wird

- Über die W'keit eines Fehlers 2. Art keine solche Kontrolle

- Die beiden Fehlerarten konkurrenzieren sich gegenseitig:

  $$P(\text{Fehler 2. Art}) \text{ wird grösser falls } \alpha \text{ kleiner gewählt wird}$$

- Wahl von α steuert Kompromiss zwischen Fehler 1. und 2. Art

- Weil man aber primär einen Fehler 1. Art vermeiden will, wählt man α klein, z.B. $\alpha = 0.05$

- Je kleiner α, desto kleiner der Verwerfungsbereich

- Vertikale Linie wandert nach rechts $\quad \rightarrow \quad$ Fehler 2. Art wird umso grösser

Vertrauensintervalle

- Zwei Methoden, um Vertrauensintervalle zu bestimmen:
  - Bootstrap: Macht keine Annahme über die Verteilung Messdaten
  - Normalverteilung der Messdaten
- Zuerst jeweils Konstruktion dann Interpretation

Bootstrap

- Bootstrap: Wichtige statistische Methode
- Methode anhand der Bildung eines Vertrauensintervalles kennenlernen
- Zuerst Vertrauensintervall konstruieren und dann entsprechende Interpretation
- Implementierung des Bootstrap sehr einfach, aber ohne Computereinsatz nicht realisierbar
- Idee: Aus einer Testreihe durch Resampling (Stichproben aus dieser Testreihe) Informationen über die Testreihe gewonnen werden

Beispiel

- Testreihe:

$$30, \quad 37, \quad 36, \quad 43, \quad 42, \quad 43, \quad 43, \quad 46, \quad 41, \quad 42$$

- Testreihe folgt einer unbekannten Verteilung und hat einen unbekannten Erwartungswert μ
- Annäherung (Punktschätzung) von

$$\mu \approx \overline{x} = 40.3$$

- Aber wie gut ist diese Schätzung?
- Dazu bestimmen wir das Vertrauensintervall mit dem Bootstrap

Empirischer Bootstrap

- n Datenpunkte, die einer (unbekannten) Verteilung F folgen

$$x_1,\ x_2,\ \ldots,\ x_n$$

- Eine empirischer Bootstrap ist ein Resample mit derselben Länge n
- Da der Standardfehler von der Länge der Testreihen abhängt, wird ein Resample mit derselben Länge gewählt
- Dies werden benutzen um das Vertrauensintervall zu konstruieren

Beispiel

- Testreihe

  30, 37, 36, 43, 42, 43, 43, 46, 41, 42

- Klein genug, um jeden Schritt explizit durchzuführen
- Resample: Testreihe derselben Länge, die aus Daten der Testreihe oben besteht
- Beschriften Kugeln mit den Zahlen oben
- Zahl 43 kommt dann dreimal vor, Zahl 30 aber nur einmal
- Legen Kugeln in eine Schüssel und ziehen die Kugeln blind
- Notieren die Zahl und legen die Kugel wieder zurück
- 10 mal wiederholen, da ursprüngliche Testreihe Länge 10 hat
- In der Praxis: Resampling mit `Python`

- Python-Code

```
import numpy as np
np.random.seed(1)

x = np.array([30, 37, 36, 43, 42, 43, 43, 46, 41, 42])
n = x.size

nboot = 1

tmpdata = np.random.choice(x, n*nboot, replace=True)

tmpdata

## [43 41 42 43 30 30 37 46 43 42]
```

- Hier kommt z.B. die Zahl 36 nicht vor, dafür die 30 zweimal
- Mittelwert dieser simulierten Testreihe ist 39.7

```
tmpdata.mean()

## 39.7
```

- Idee des Bootstrap: Resampling sehr oft durchgeführen wird, um

$$\mu \approx \overline{x} = 40.3$$

abzuschätzen

- Zur Abschätzung von μ gibt man ein 95 %-Bootstrap-Vertrauensintervall

- Hier: Vereinfachte Variante, Prinzip besser nachvollziehbar
Dies geht konkret wie folgt.

- Mit `Python`: Erzeugen 20 Bootstrap-Proben alle mit der Länge 10

- Jede der 20 Spalten im folgenden Array ist eine Bootstrap-Stichprobe

43	41	42	43	30	30	37	46	43	42	36	42	43	36	42	36	42
37	46	30	43	42	42	46	43	42	37	30	37	41	41	43	42	41
43	37	42	43	42	41	37	42	30	43	42	36	30	42	42	36	46
43	42	43	46	46	42	43	42	43	43	41	30	36	46	46	42	46
46	46	37	37	43	30	41	43	42	43	43	36	43	46	41	42	42
42	30	36	30	46	37	46	42	41	42	30	37	42	41	36	43	37
43	30	42	36	43	43	36	46	46	30	43	43	37	42	43	30	43
37	43	42	30	46	41	42	43	46	30	42	43	42	37	42	42	43
36	46	43	43	42	43	41	43	41	37	37	41	46	30	43	42	36
37	36	42	43	30	43	30	46	36	41	43	30	41	42	36	42	30

- Code:

```
import numpy as np

x = np.array([30, 37, 36, 43, 42, 43, 43, 46, 41, 42])
n = x.size

np.random.seed(1)

xbar = x.mean()

nboot = 20
tmpdata = np.random.choice(x, n*nboot, replace=True)

bootstrapsample = np.reshape(tmpdata, (n, nboot))

bootstrapsample

## [[43 41 42 43 30 30 37 46 43 42 36 42 43 36 42 36 42 46 46 42]
##  [37 46 30 43 42 42 46 43 42 37 30 37 41 41 43 42 41 46 43 43]
##  [43 37 42 43 42 41 37 42 30 43 42 36 30 42 42 36 46 46 42 41]
##  [43 42 43 46 46 42 43 42 43 43 41 30 36 46 46 42 46 43 30 41]
##  [46 46 37 37 43 30 41 43 42 43 43 36 43 46 41 42 42 46 46 42]
##  [42 30 36 30 46 37 46 42 41 42 30 37 42 41 36 43 37 36 46 36]
##  [43 30 42 36 43 43 36 46 46 30 43 43 37 42 43 30 43 43 37 36]
##  [37 43 42 30 46 41 42 43 46 30 42 43 42 37 42 42 43 41 41 42]
##  [36 46 43 43 42 43 41 43 41 37 37 41 46 30 43 42 36 30 43 43]
##  [37 36 42 43 30 43 30 46 36 41 43 30 41 42 36 42 30 43 41 37]]
```

- Berechnen Mittelwerte in allen Spalten und ordnen sie der Reihen nach:

```
xbarstar = bootstrapsample.mean(axis=0)

np.sort(xbarstar)
```

```
## [37.5 38.7 38.8 39.2 39.4 39.7 39.7 39.9 39.9 40.1 40.3 40.3
##  41.  41.  41.4 41.5 42.  43.6]
```

- 95 %-Bootstrap-Vertrauensintervall: Wählen die „mittleren" 95 % dieser Mittelwert
- Lassen ist Liste oben auf beiden Seiten 2.5 % der Mittelwert weg
- Diese werden durch die 2.5 %- und 97.5 %-Quantile begrenzt

```
d = np.percentile(xbarstar, q=[2.5, 97.5])
print('Vertrauensintervall: ',d)
```

```
## Vertrauensintervall:  [38.07 42.84]
```

- 95 %-Bootstrap-Vertrauensintervall im Vergleich zu $\overline{x} = 40.3$:

$$[40.3 - 2.23,\ 40.3 + 2.54]$$

- Beim „richtigen" Bootstrap werden diese Unterschiede zum Mittelwert abgeschätzt
- Diese lauten dann allerdings leicht anders
- Aber wie immer: bei grossen Datensätzen und bei einer grossen Anzahl von Resamplings spielt dies alles keine Rolle.
- Hier der Übersichtlichkeit halber nur 20 Bootstrap-Stichproben erzeugt
- Mit `Python`: Auch 10 000 Bootstrap-Stichproben erzeugen
- Genauere Abschätzung für 95 %-Bootstrap-Vertrauensintervall

```
## Vertrauensintervall:  [37.4 42.8]
```

Interpretation des Vertrauensintervalls

- Wie lässt sich Vertrauensintervall interpretieren?
- Simulieren Daten, deren wahres μ bekannt ist
- Wählen 100 Zufallszahlen, die Verteilung $\mathcal{N}(40, 5^2)$ folgen
- Das wahre μ ist also 40
- Man kann sich fragen, ob diese μ nun im entsprechenden 95 % Bootstrap-Vertrauensintervall liegt oder nicht

- Code: (mit „richtigem" Bootstrap)

```
import numpy as np
np.random.seed(4)

x = np.random.normal(loc=40, scale=5, size=100)
n = x.size

xbar = x.mean()

nboot = 20

tmpdata = np.random.choice(x, n*nboot, replace=True)

bootstrapsample = np.reshape(tmpdata, (n, nboot))

xbarstar = bootstrapsample.mean(axis=0)

deltastar = xbarstar - xbar

d = np.percentile(deltastar, q=[2.5, 97.5])

ci = xbar - [d[1], d[0]]
print("Vertrauensintervall: ",ci)

## Vertrauensintervall:  [39.47300068 40.93571204]
```

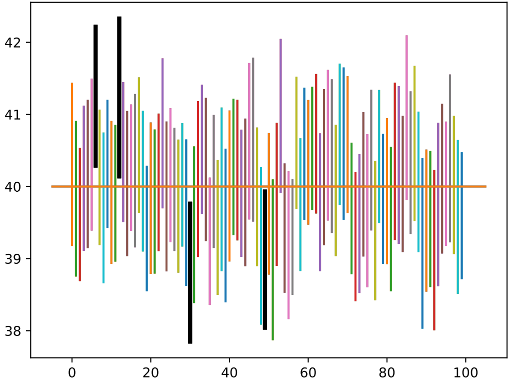
- Das wahre μ liegt in diesem Intervall

- Ist dies aber immer der Fall?

- Wählen nun 1000 Testreihen mit je 100 Werten, bestimmen von jedem das Vertrauensintervall und schauen, ob das wahre μ darin liegt.

```
import numpy as np
np.random.seed(8)
x = np.random.normal(loc=40, scale=5, size=100000)

sample = np.reshape(x,(1000,100))

nboot = 1000
n = 100
k=0
for i in range(0,1000):
    y = sample[i]
    xbar = y.mean()
    tmpdata = np.random.choice(y, n*nboot, replace=True)
    bootstrapsample = np.reshape(tmpdata, (n, nboot))
    xbarstar = bootstrapsample.mean(axis=0)
    deltastar = xbarstar - xbar
    d = np.percentile(deltastar, q=[2.5, 97.5])
    if xbar-d[1]<= 40 <= xbar-d[0]:
        k=k+1

print(k)

## 943
```

- In 943 Fällen liegt das wahre μ im Vertrauensintervall der entsprechenden Messreihe
- D.h.: In 94.5 % dies der Fall
- Graphisch Darstellung für 100 Datensätzen



- In Abbildung 100 Simulationen von 95 % Bootstrap-Vertrauensintervallen
- Zudem ist das wahre Mittel $\mu = 40$ eingezeichnet
- Vier Vertrauensintervalle (schwarz eingezeichnet) schneiden die horizontale Linie 40 nicht
- D.h.: Diese Vertrauensintervalle enthalten das wahre Mittel *nicht*
- Wahres Mittel in 96 % aller 95 %-Vertrauenintervalle enthalten
- In beiden Beispielen vorher liegt der wahre Wert in *etwa* 95 % aller 95 % Bootstrap-Vertrauensintervalle

- Lassen sehr Anzahl Testreihen sehr gross werden

- Nach Gesetz der grossen Zahlen liegt das wahre Mittel in 95 % aller 95 % Bootstrap-Vertrauensintervalle.

- Allgemein:

> **Vertrauensintervalle**
>
> Im $\alpha \cdot 100\,\%$-Vertrauensintervall liegt zu $\alpha \cdot 100\,\%$ Sicherheit der wahre Wert μ.

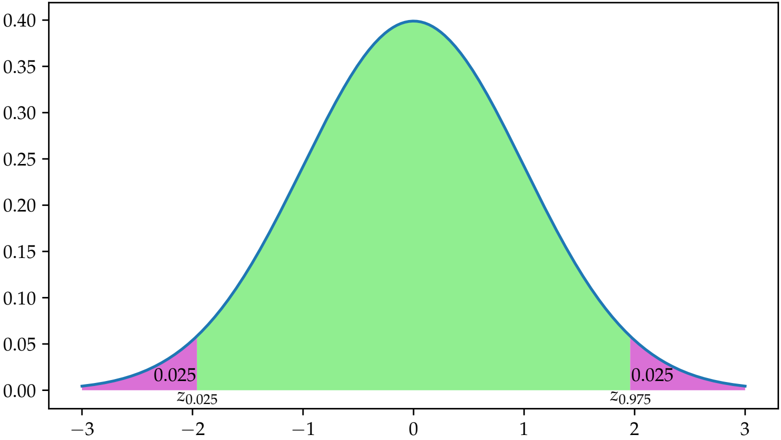
- Ist das Vertrauensintervall sehr breit, so haben wir wenig Gewissheit, wo der wahre Wert von μ liegt

- Anderer Seite gibt schmales Vertrauensintervall eine gute Schätzung für den wahren Wert von μ

Vertrauensintervalle für Normalverteilungen: Einleitung

- Betrachten nochmals Verwerfungsbereich einer normalverteilten Zufallsvariable X mit bekannten σ_X
- Jetzt: Standardnormalverteilung
- Beschränkung vorläufig auf Signifikanzniveau 5 % und zweiseitigem Verwerfungsbereich

- Skizze:



0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.025
0.025
$z_{0.025}$
$z_{0.975}$
−3
−2
−1
0
1
2
3

- Bestimmen den zweiseitigen Verwerfungsbereich einer standardnormalverteilten Zufallsvariable Z.

- Abbildung: Verwerfungsbereich durch $z_{0.025}$ und $z_{0.975}$ begrenzt

- Diese haben die Werte -1.96 und 1.96.

```
from scipy.stats import norm

norm.ppf(q=[0.025, 0.975])

## [-1.95996398  1.95996398]
```

- Interpretation: W'keit, dass ein Z-verteilter Messwert zwischen -1.96 und 1.96 liegt ist 95 %

- Intervall wird wie folgt geschrieben

$$-1.96 \leq Z \leq 1.96$$

- Schon gesehen: Jede normalverteilte Zufallsvariable kann man standardisieren

- Beispiel: $X \sim \mathcal{N}(6, 2^2)$

- Standardisierte Variable:

$$Z = \frac{X - 6}{2}$$

- Setzen dies in Gleichung

$$-1.96 \leq Z \leq 1.96$$

  ein:

$$-1.96 \leq \frac{X - 6}{2} \leq 1.96$$

- Multiplizieren Ungleichung mit 2 und addieren mit 6:

$$6 - 1.96 \cdot 2 \leq X \leq 6 + 1.96 \cdot 2$$

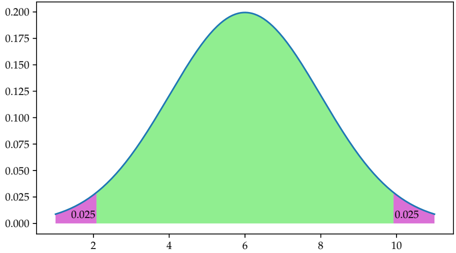
- Oder:

$$2.08 \leq X \leq 9.92$$

- Beiden Zahlen geben gerade die Grenzen des Verwerfungsbereiches an

```
norm.ppf(q=[0.025, 0.975], loc=6, scale=2)

## [2.08007203 9.91992797]
```

- Skizze:



- Können nun den Bereich

$$2.08 \leq X \leq 9.92$$

auch als das Intervall auffassen, indem 95 % aller Messwerte liegen

- Letztes Beispiel verallgemeinern
- Ist $X \sim \mathcal{N}(\mu, \sigma^2)$, so ist lautet die standardisierte Variable:

$$Z = \frac{X - \mu}{\sigma}$$

- Auf 5 %-Signifikanzniveau gilt dann:

$$-1.96 \leq \frac{X - \mu}{\sigma} \leq 1.96$$

- Multiplizieren mit σ und addieren μ:

$$\mu - 1.96 \cdot \sigma \leq X \leq \mu + 1.96 \cdot \sigma$$

- D.h.: Messwert, der Normalverteilung X folgt, liegt mit W'keit 95 % im Intervall

$$[\mu - 1.96 \cdot \sigma,\ \mu + 1.96 \cdot \sigma]$$

- Dieses Intervall macht Aussage über Messwerte, wenn μ bekannt

- Entscheidende Überlegung für das Vertrauensintervall: Gleichung

$$-1.96 \leq \frac{X-\mu}{\sigma} \leq 1.96$$

  auch in der Mitte nach μ auflösen

- Dies gibt:

$$X - 1.96 \cdot \sigma \leq \mu \leq X + 1.96 \cdot \sigma$$

- Diese Ungleichung macht eine Aussage über dass wahre μ, wenn ein Messwert für X gegeben ist

- Das heisst, das wahre μ liegt zu 95 % im Intervall

$$[X - 1.96 \cdot \sigma, \ X + 1.96 \cdot \sigma]$$

- Wie ist dies nun zu verstehen?

- Betrachten $X \sim \mathcal{N}(6, 2^2)$ und simulieren einen Wert

```python
from scipy.stats import norm
import numpy as np
np.random.seed(1)

norm.rvs(size=1, loc=6, scale=2)

## [4.77648717]
```

- Intervall

$$I = [4.78 - 1.96 \cdot 2, 4.78 + 1.96 \cdot 2]$$

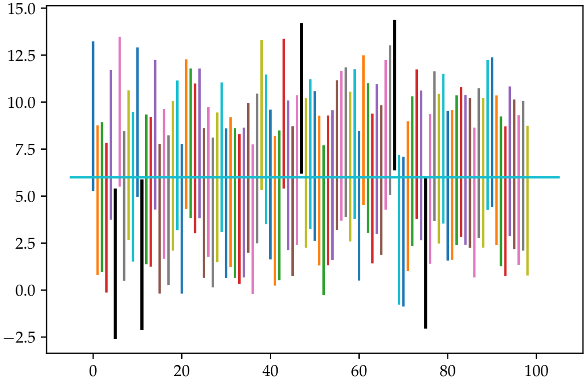
- Oder:

$$I = [0.86, 8.7]$$

- Der wahre Wert 6 liegt nun in diesem Intervall

- Ist dies aber immer der Fall?

- Simulieren nun 100 Messwerte und bestimmen für alle das Intervall oben
- Diese Intervalle sind in Abbildung unten eingezeichnet
- Die horizontale blaue Linie entspricht dem Wert 6



- Schwarz eingezeichneten Intervalle enthalten wahren Wert $\mu = 6$ *nicht*
- Dies sind hier genau 5 Intervalle
- Demnach enthalten 95 % der Intervalle den wahren Wert $\mu = 6$
- Nun die Interpretation von *einem* Vertrauensintervall wie

$$I = [0.86, 8.7]$$

- Man kann mit 95 % Sicherheit sagen, dass das wahre (aber unbekannte) μ in diesem Intervall liegt

- Bis jetzt: Signifikanzniveau von 5 % ausgegangen

- Dies ist zwar oft der Fall, aber nicht notwendig

- Für ein allgemeines Signifikanzniveau α wird in Herleitung oben einfach -1.96 durch $z_{\frac{\alpha}{2}}$ und 1.96 durch $z_{1-\frac{\alpha}{2}}$ ersetzt

- Das allgemeine Vertrauensintervall zum Messwert x lautet dann

$$I = [x - z_{1-\frac{\alpha}{2}} \cdot \sigma, x - z_{\frac{\alpha}{2}} \cdot \sigma]$$

- Für $\alpha = 0.01$ gilt dann

$$I = [x - 2.58 \cdot \sigma, x + 2.58 \cdot \sigma]$$

```
from scipy.stats import norm

norm.ppf(q=[0.005, 0.995])

## [-2.5758293  2.5758293]
```

Vertrauensintervall allgemein

- Das sogenannte *Vertrauensintervall* bei Messdaten besteht aus denjenigen Werten μ, bei denen der entsprechende Test nicht verwirft
- Das sind also alle Parameterwerte des Zufallsmodells, bei denen die Daten recht wahrscheinlich oder plausibel sind
- Dieses Intervall enthält dann das wahre, aber meist unbekannte μ mit einer gegebenen Wahrscheinlichkeit
- Z.B. ist das 95 %-Vertrauensintervall für μ das Intervall, das μ mit einer Wahrscheinlichkeit von 0.95 enthält
- D.h. wenn wir sehr viele Testreihen machen und jeweils das Vertrauensintervall bestimmen, so wird μ in 95 % dieser Intervalle enthalten sein
- Ist das Signifikanzniveau α, so ist nennen wir das Intervall $(1-\alpha)\cdot 100\,\%$-Vertrauensintervall.

Vertrauensintervalle für μ einer Messreihe

- Gehen nun wieder von *Messreihen* aus
- Annahme: Daten Realisierungen von

$$X_1, \ldots, X_n \text{ i.i.d.} \sim \mathcal{N}(\mu, \sigma_X^2)$$

- Müssen wieder unterscheiden, ob σ_X bekannt oder unbekannt ist

Vertrauensintervalle, falls σ_X bekannt

- Diesen Fall schon in Einleitung betrachtet
- Der Mittelwert $\overline{X}_n$ folgt der Verteilung

$$\overline{X}_n \sim \mathcal{N}\left(\mu, \sigma^2_{\overline{X}_n}\right) = \mathcal{N}\left(\mu, \frac{\sigma^2_X}{n}\right)$$

- Formel der Einleitung

$$[x - z_{1-\frac{\alpha}{2}} \cdot \sigma, X - z_{\frac{\alpha}{2}} \cdot \sigma]$$

- Ersetzen Messwert x durch den Mittelwert $\overline{x}_n$ der Messreihe und σ durch $\sigma_{\overline{X}_n} = \frac{\sigma_X}{\sqrt{n}}$: $\left[\overline{x}_n - z_{1-\frac{\alpha}{2}} \cdot \frac{\sigma_X}{\sqrt{n}},\ \overline{x}_n - z_{\frac{\alpha}{2}} \cdot \frac{\sigma_X}{\sqrt{n}}\right]$

Beispiel

- Schmelzwärme von früher: Normalverteilt mit $\mu = 80$ und $\sigma_X = 0.02$
- Standardabweichung wird hier also als bekannt angenommen
- Mittelwert: $\overline{x}_{13} = 80.02$
- Es gilt
$$z_{0.975} = 1.96$$
- Zweiseitige Konfidenzintervall für Methode A:
$$I = 80.02 \pm 1.96 \cdot 0.02/\sqrt{13} = [80.009, 80.031]$$
- Insbesondere liegt 80.00 nicht im Intervall I
- Wert $\mu = 80.00$ ist folglich nicht mit den Daten kompatibel

- Berechnung von Hand ist ein bisschen mühsam, aber mit **Python** ist es recht einfach

```
from scipy.stats import norm, t
import numpy as np

norm.interval(alpha=0.95, loc=80.02, scale=0.02/np.sqrt(13))

## (80.00912807593181, 80.03087192406818)
```

Vertrauensintervalle, falls σ_X bekannt: Allgemein

- Allgemein gilt für bekanntes σ_X:

Zweiseitiges Vertrauensintervall

Dies führt dann auf die folgenden *zweiseitigen Vertrauensintervalle* (die dazugehörigen Tests sind zweiseitig mit Alternative $H_A: \mu \neq \mu_0$) zum Niveau $1-\alpha$:

$$\left[\overline{x}_n - z_{1-\alpha/2} \cdot \frac{\sigma_X}{\sqrt{n}},\ \overline{x}_n + z_{1-\alpha/2} \cdot \frac{\sigma_X}{\sqrt{n}}\right]$$

Analog kann man auch *einseitige Vertrauensintervalle* konstruieren. Sie enthalten alle Parameter, bei denen ein einseitiger Test nicht verwerfen würde. Beim t-Test sehen die einseitigen $(1-\alpha)$-Vertrauensintervalle so aus:

$$\text{Falls } H_A\colon \mu < \mu_0 : \left(-\infty; \overline{x}_n + z_{1-\alpha} \cdot \frac{\sigma_X}{\sqrt{n}}\right]$$

$$\text{Falls } H_A\colon \mu > \mu_0 : \left[\overline{x}_n - z_{1-\alpha} \cdot \frac{\sigma_X}{\sqrt{n}};\ \infty\right)$$

Vertrauensintervalle, falls σ_X unbekannt

- Ist σ_X unbekannt, so verwenden wir t-Verteilungen und die geschätzte Standardabweichung $\widehat{\sigma}_X$
- Ersetzen also in der grünen Box oben $z_{1-\alpha/2}$ bzw. z_α durch $t_{n-1;1-\frac{\alpha}{2}}$ bzw. $t_{n-1;\alpha}$ und σ_X durch $\widehat{\sigma}_X$.

- Vertrauensintervalle, falls σ_X unbekannt

Zweiseitiges Vertrauensintervall

Dies führt dann auf die folgenden *zweiseitigen Vertrauensintervalle* (die dazugehörigen Tests sind zweiseitig mit Alternative $H_A\colon \mu \neq \mu_0$) zum Niveau $1-\alpha$:

$$\left[\overline{x}_n - t_{n-1,1-\alpha/2} \cdot \frac{\widehat{\sigma}_X}{\sqrt{n}},\ \overline{x}_n + t_{n-1,1-\alpha/2} \cdot \frac{\widehat{\sigma}_X}{\sqrt{n}}\right]$$

Analog kann man auch *einseitige Vertrauensintervalle* konstruieren. Sie enthalten alle Parameter, bei denen ein einseitiger Test nicht verwerfen würde. Beim t-Test sehen die einseitigen $(1-\alpha)$-Vertrauensintervalle so aus:

$$\text{Falls } H_A\colon \mu < \mu_0 : \left(-\infty;\ \overline{x}_n + t_{n-1,1-\alpha} \cdot \frac{\widehat{\sigma}_X}{\sqrt{n}}\right]$$

$$\text{Falls } H_A\colon \mu > \mu_0 : \left[\overline{x}_n - t_{n-1,1-\alpha} \cdot \frac{\widehat{\sigma}_X}{\sqrt{n}};\ \infty\right)$$

Beispiel: Schmelzwärme Methode A

- Freiheitsgrade

$$n - 1 = 13 - 1 = 12$$

- Es gilt:

$$t_{12,0.975} = 2.18$$

```
from scipy.stats import t

t.ppf(q=0.975, df=12)

## 2.1788128296634177
```

- Die mittlere mit Methode A gemessene Schmelzwärme ist

$$\overline{x}_n = 80.02$$

- Die Standardabweichung lautet

$$\widehat{\sigma}_X = 0.024$$

- Zweiseitige Konfidenzintervall für die mit Methode A gemessene Schmelzwärme:

$$I = 80.02 \pm 2.18 \cdot 0.024/\sqrt{13} = [80.01, 80.03]$$

- Insbesondere liegt 80.00 nicht im Intervall I

- Der Wert $\mu = 80.00$ ist folglich nicht mit den Daten kompatibel, was wir bereits mit Hilfe des t-Tests ermittelt hatten.

- Python-Befehl:

```
import scipy.stats as st
from scipy.stats import norm, t
import numpy as np

t.interval(alpha=0.95, df=12, loc=80.02, scale=0.024/np.sqrt(13))

## (80.00549694515017, 80.03450305484982)
```