Kapitel 11

Resampling-Verfahren

Resampling-Verfahren kommen für eine Vielzahl von Tests in Frage, können hier aber nur in Grundzügen vorgestellt werden. Ausgangspunkt ist die gesuchte Verteilung einer Teststatistik $\hat{\theta}$ – etwa eines Schätzers $\hat{\theta}$ für einen theoretischen Parameter θ. Diese Verteilung kann aus verschiedenen Gründen unbekannt sein: So sind etwa die in parametrischen Tests gemachten Annahmen, unter denen ihre Teststatistik eine bekannte Verteilung aufweist, nicht immer zu rechtfertigen. In vielen klassischen nonparametrischen Verfahren ist die Verteilung der Teststatistik zwar im Prinzip exakt zu ermitteln, praktisch aber der Rechenaufwand dafür zu hoch.

Grundidee von Bootstrap-Verfahren und Permutationstests ist es, aus den gegebenen Daten einer festen Basisstichprobe viele neue Zufallsstichproben (*resamples*) zu generieren und die Teststatistik für jedes resample zu ermitteln – die dabei berechneten Werte werden als $\hat{\theta}^\star$ bezeichnet. Die empirische Verteilung der $\hat{\theta}^\star$ dient der Approximation der theoretischen Verteilung von $\hat{\theta}$. Die Beziehung zwischen resample und Basisstichprobe wird also auf die Beziehung zwischen Basisstichprobe und Population übertragen. Im Vergleich zu klassischen nonparametrischen Tests (vgl. Kap. 10) versprechen Resampling-Methoden oft eine höhere Power.

11.1 Nonparametrisches Bootstrapping

Bei Bootstrap-Verfahren (Chernik & LaBudde, 2011; Chihara & Hesterberg, 2011; Davison & Hinkley, 1997) werden die resamples als *Replikationen* bezeichnet. Beim nonparametrischen bootstrap sind dies mit Zurücklegen gezogene Zufallsstichproben aus der Basisstichprobe mit derem Stichprobenumfang. $\hat{\theta}$ ist der auf Grundlage der Basisstichprobe berechnete *plug-in*-Schätzer von θ, wird also auf empirischer Ebene rechnerisch genauso gebildet wie θ auf theoretischer Ebene.[1] Über den – oft weniger interessanten – Bootstrap-Punktschätzer für θ hinaus lassen sich aus der empirischen Verteilung von $\hat{\theta}^\star$ vor allem Konfidenzintervalle für θ bestimmen. Dafür approximiert die empirische Verteilung von $\hat{\theta}^\star - \hat{\theta}$ in vielen Fällen jene der Pivot-Statistik $\hat{\theta} - \theta$, deren Verteilung unabhängig vom konkreten Wert für θ ist.

Ein vollständiger nonparametrischer bootstrap umfasst alle möglichen Replikationen einer Basisstichprobe vom Umfang n, wofür der Rechenaufwand jedoch meist zu groß ist: Es gibt bereits $2^n - 1$ relevante Teilmengen von Beobachtungsobjekten (die Mächtigkeit der Potenzmenge

[1] θ ist ein *Funktional* der theoretischen Verteilungsfunktion F der ursprünglichen Zufallsvariable, bildet also F auf eine Zahl ab. Analog ist $\hat{\theta}$ dasselbe Funktional der empirischen kumulativen Häufigkeitsverteilung $\hat{F}_n$ der Basisstichprobe vom Umfang n und $\hat{\theta}^\star$ dasselbe Funktional der empirischen kumulativen Häufigkeitsverteilung $\hat{F}_n^\star$ in einer Replikation.

ohne die leere Menge), wobei die Elemente jeder denkbaren Zusammensetzung noch mit unterschiedlichen Häufigkeiten berücksichtigt werden können. Deswegen wird bootstrapping als *Monte-Carlo-Verfahren* durchgeführt und nur eine zufällige Auswahl von Replikationen berücksichtigt.

11.1.1 Replikationen erstellen

Das Paket `boot` stellt mit `boot()` eine Funktion bereit, die Bootstrap-Replikationen durchführt.[2]

```
> boot(data=⟨Basisstichprobe⟩, statistic=⟨Funktion⟩, R=⟨# Replikationen⟩,
+      strata=⟨Faktor⟩)
```

Für `data` ist der Vektor mit den Werten der Basisstichprobe zu übergeben. Basiert $\hat{\theta}$ auf Daten mehrerer Variablen, muss `data` ein Datensatz mit diesen Variablen sein. Das Argument `statistic` erwartet den Namen einer Funktion mit ihrerseits zwei Argumenten zur Berechnung von $\hat{\theta}^\star$: Ihr erstes Argument ist ebenfalls der Vektor bzw. Datensatz der Basisstichprobe. Das zweite Argument von `statistic` ist ein Indexvektor, dessen Elemente sich auf die beobachteten Fälle beziehen. `boot()` ruft für jede der `R` vielen Replikationen `statistic` auf und übergibt die Basis-Daten samt eines zufällig gewählten Indexvektors als Anweisung, wie eine konkrete Replikation aus Fällen der Basisstichprobe zusammengesetzt sein soll. Das Ergebnis von `statistic` muss $\hat{\theta}^\star$ sein – sind gleichzeitig mehrere Parameter θ_j zu schätzen, analog ein Vektor mit den $\hat{\theta}^\star_j$. Um beim bootstrapping eine vorgegebene Stratifizierung der Stichprobe beizubehalten, kann ein Faktor an `strata` übergeben werden, der eine Gruppeneinteilung definiert. In den Replikationen sind die Gruppengrößen dann gleich jenen der Basisstichprobe.

Als Beispiel diene die die Situation eines t-Tests für eine Stichprobe auf einen festen Erwartungswert μ_0 (vgl. Abschn. 7.2.1). Ziel im folgenden Abschnitt ist die Konstruktion eines Vertrauensintervalls für μ $(= \theta_1)$. Die Stichprobengröße sei n, der Mittelwert M $(= \hat{\theta}_1)$ und die unkorrigierte Varianz des Mittelwertes S^2_M $(= \hat{\sigma}^2_{\hat{\theta}_1} = \hat{\theta}_2)$ als plug-in-Schätzer der theoretischen Varianz σ^2_M $(= \theta_2)$. Dazu ist aus jeder Bootstrap-Replikation der Mittelwert $M^\star$ $(= \hat{\theta}^\star_1)$ und die unkorrigierte Varianz des Mittelwertes $S^{2\star}_M$ $(= \hat{\sigma}^{2\star}_{\hat{\theta}_1} = \hat{\theta}^\star_2)$ zu berechnen. Für das Erstellen eigener Funktionen vgl. Abschn. 16.2.

```
> muH0 <- 100                                  # Erwartungswert unter H0
> sdH0 <- 40                                   # Streuung unter H0
> N    <- 200                                  # Größe Basisstichprobe
> DV   <- rnorm(N, muH0, sdH0)                 # Basisstichprobe

# Mittelwert M* und Varianz des Mittelwertes S(M)^2* aus BS-Replikation,
# deren zufällige Zusammensetzung durch Indexvektor idx bestimmt wird
> getM <- function(orgDV, idx) {
+     n     <- length(orgDV[idx])
+     bsM   <- mean(orgDV[idx])                  # M*
+     bsS2M <- (((n-1)/n) * var(orgDV[idx])) / n    # S(M)^2*
```

[2] Flexible bootstrap-basierte Tests für eine oder zwei Stichproben ermöglicht auch das Paket `resample` (Hesterberg, 2014).

```
+     c(bsM, bsS2M)
}

> library(boot)                                # für boot(), boot.ci()
> nR     <- 999                                # Anzahl BS-Replikationen
> (bsRes <- boot(DV, statistic=getM, R=nR))         # BS-Replikationen
ORDINARY NONPARAMETRIC BOOTSTRAP
Call:
boot(data = DV, statistic = getM, R = nR)

Bootstrap Statistics :
       original          bias   std. error
t1*  102.655867   -0.09535249     3.057508
t2*    9.162478   -0.02025724     0.881103
```

Die Ausgabe von `boot()` nennt in den Zeilen `t1*` und `t2*` Eigenschaften der von `statistic` zurückgegebenen Bootstrap-Schätzer: In der Spalte `original` stehen die für die Basisstichprobe berechneten Werte. Hier sind dies der Mittelwert und seine unkorrigierte Varianz. In der Spalte `bias` folgt die Verzerrung jedes Bootstrap-Schätzers als Mittelwert der Abweichungen $\hat{\theta}^\star - \hat{\theta}$. Für die Punktschätzung von θ kann sie zu einer einfachen Bias-Korrektur eingesetzt werden, indem sie von $\hat{\theta}$ abgezogen wird. Schließlich folgt in der Spalte `std. error` die korrigierte Streuung für jeden Kennwert der pro Replikation von `statistic` berechneten Kennwerte. Diese sind in der von `boot()` zurückgegebenen Liste spaltenweise als Matrix in der Komponente `t` gespeichert.

```
> (M <- mean(DV))                            # Mittelwert Basisstichprobe
[1] 102.6559

> (S2M <- (((N-1)/N) * var(DV)) / N)         # unkorrig. Varianz von M
[1] 9.162478

> Mstar   <- bsRes$t[ , 1]                   # M* jeder Replikation
> S2Mstar <- bsRes$t[ , 2]                   # S(M)^2* jeder Replikation
> (biasM  <- mean(Mstar) - M)                # Bias-Schätzung für M
[1] -0.0953525

> mean(S2Mstar) - S2M                        # Bias-Schätzung für S(M)^2
[1] -0.02025724

> c(sd(Mstar), sd(S2Mstar))                  # Streuungen der BS-Kennwerte
[1] 3.057508 0.881103
```

Die Verteilung von $\hat{\theta}^\star - \hat{\theta}$ sollte unimodal und symmetrisch sein, nicht unähnlich einer Normalverteilung (Abb. 11.1). Dies lässt sich etwa durch ein Histogramm mit eingezeichneter Dichtefunktion einer passenden Normalverteilung zusammen mit einem nonparametrischen Kerndichteschätzer oder einem Q-Q-Plot prüfen.

```
> hist(Mstar-M, freq=FALSE, breaks="FD")  # Histogramm von M* - M
```

```
> rug(jitter(Mstar-M))                    # Einzelwerte der M* - M

# Dichtefunktion einer Normalverteilung
> curve(dnorm(x, mean(Mstar-M), sd(Mstar-M)), lwd=2, col="blue", add=TRUE)
> lines(density(Mstar-M), lwd=2, col="red", lty=2)    # Kerndichteschätzer
> qqnorm(Mstar-M, pch=16)                 # Q-Q-Plot der M* - M
> qqline(Mstar-M, lwd=2, col="blue")      # Normalverteilung-Referenz
```

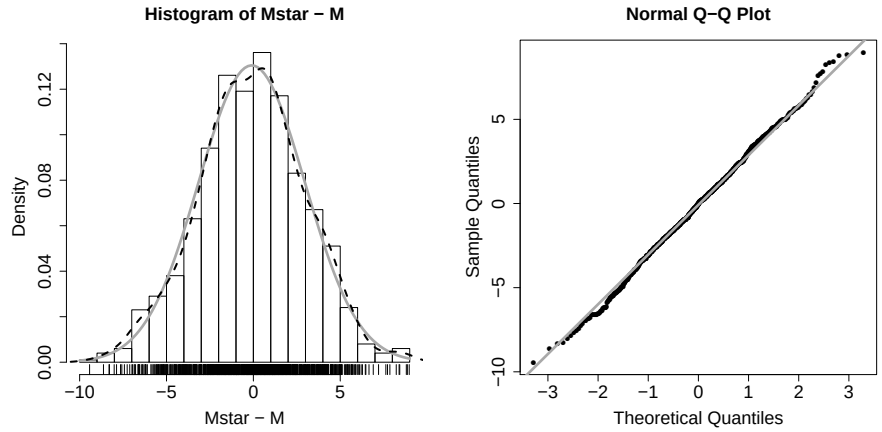

Abbildung 11.1: Histogramm von $M^\star - M$ aus Bootstrap-Replikationen mit passender Normalverteilung und nonparametrischem Kerndichteschätzer. Q-Q-Plot von $M^\star - M$ mit Vergleich zur Standardnormalverteilung.

Detaillierte Informationen über die Zusammensetzung aller von `boot()` erstellten Replikationen liefert `boot.array(`⟨`boot-Objekt`⟩`, indices=TRUE)`. Bei einer Basisstichprobe vom Umfang n und R Replikationen ist das Ergebnis mit dem Argument `indices=TRUE` eine $(R \times n)$-Matrix mit einer Zeile für jede Replikation und einer Spalte pro Beobachtung. Die Zelle (r, i) enthält den Index des ausgewählten Elements der Basisstichprobe. Zusammen mit dem Vektor der Basisstichprobe lassen sich damit alle Replikation rekonstruieren.

```
> bootIdx <- boot.array(bsRes, indices=TRUE)
# Replikationen 1-3: je ausgewählte erste 10 Indizes Basisstichprobe
> bootIdx[1:3, 1:10]
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]   78  148   83   72   86   78   33   59   87   141
[2,]   41  116  197   87  183    4  102  129   84   175
[3,]  123  107   60  181   35  133  196  197   15    71

> repl1Idx <- bootIdx[1, ]        # Indizes der ersten Replikation
> repl1DV  <- DV[repl1Idx]        # Werte der ersten Replikation
> head(repl1DV, n=5)
[1]  86.12395  72.41568 135.87073 149.96011  99.03936
```

11.1.2 Bootstrap-Vertrauensintervalle für μ

Ein von `boot()` erzeugtes Objekt ist für das Argument `boot.out` der Funktion `boot.ci()` anzugeben, die das zweiseitige Konfidenzintervall für θ bestimmt.

```
> boot.ci(boot.out=⟨boot-Objekt⟩, conf=⟨Breite VI⟩, index=⟨Nummer⟩,
+         type="⟨Methode⟩")
```

Gibt die für `boot(..., statistic)` genannte Funktion bei jedem Aufruf J geschätzte Parameter zurück, ist `index` der Reihe nach auf $1, \ldots, J$ zu setzen, um das zugehörige Konfidenzintervall zu erhalten. Das Intervall wird mit der Breite `conf` nach einer über `type` festzulegenden Methode gebildet:

- `"basic"`: Das klassische Bootstrap-Intervall um $\hat{\theta}$, dessen Breite durch die $\frac{\alpha}{2}$- und $1 - \frac{\alpha}{2}$-Quantile der Werte von $\hat{\theta} - \hat{\theta}^\star$ definiert ist.
- `"perc"`: Das Perzentil-Intervall, dessen Grenzen die $\frac{\alpha}{2}$- und $1 - \frac{\alpha}{2}$-Quantile der Werte von $\hat{\theta}^\star$ sind.
- `"norm"`: Das Normalverteilungs-Intervall mit dem Zentrum in $\hat{\theta} - M(\hat{\theta}^\star - \hat{\theta})$ (Bias-Korrektur), dessen Breite definiert ist durch die $\frac{\alpha}{2}$- und $1 - \frac{\alpha}{2}$-Quantile der Standardnormalverteilung multipliziert mit der Streuung von $\hat{\theta}^\star$.
- `"stud"`: Das t-Vertrauensintervall um $\hat{\theta}$, dessen Breite definiert ist durch die $\frac{\alpha}{2}$- und $1 - \frac{\alpha}{2}$-Quantile der Werte von $t^\star = \frac{\hat{\theta}^\star - \hat{\theta}}{S^\star_{\hat{\theta}}}$ multipliziert mit der Streuung von $\hat{\theta}^\star$. Voraussetzung ist, das die Funktion `statistic` als zweites Element den plug-in Schätzer $S^{2\star}_{\hat{\theta}}$ der theoretischen Varianz $\sigma^2_{\hat{\theta}}$ zurückliefert. Ist deren geschlossene Form unbekannt oder existiert nicht, lässt sich $S^{2\star}_{\hat{\theta}}$ innerhalb jedes Aufrufs von `statistic` im ursprünglichen bootstrapping durch eine eigene (*nested*) Bootstrap-Schätzung ermitteln.
- `"bca"`: Das BC_a-Intervall (*bias-corrected and accelerated*). Für symmetrische Verteilungen und Schätzer mit geringem bias ähnelt es dem Perzentil- und t-Intervall meist stark. Bei schiefen Verteilungen und größerem bias wird das BC_a-Intervall den anderen oft vorgezogen.

Bei den Intervallen gehen zur Erhöhung der Genauigkeit nicht die $\frac{\alpha}{2}$- und $1 - \frac{\alpha}{2}$-Quantile selbst von $t^\star$ bzw. von $\hat{\theta}^\star$ ein, die mit `quantile(..., probs=c(⟨alpha⟩/2, 1 - ⟨alpha⟩/2))` zu ermitteln wären: Bei n_R vielen Replikationen sind dies stattdessen die Elemente mit den Indizes $(n_R + 1) \cdot \frac{\alpha}{2}$ und $(n_R + 1) \cdot (1 - \frac{\alpha}{2})$ der sortierten Werte von $t^\star$ bzw. von $\hat{\theta}^\star$. Ergibt sich kein ganzzahliges Ergebnis für die Indizes, interpoliert `boot.ci()` jeweils zwischen den angrenzenden Elementen.

```
> alpha <- 0.05                                    # alpha-Niveau
> boot.ci(bsRes, conf=1-alpha,
+          type=c("basic", "perc", "norm", "stud", "bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 999 bootstrap replicates
CALL :
boot.ci(boot.out=bsRes, conf=1 - alpha,
```

```
        type = c("basic", "perc", "norm", "stud", "bca"))

Intervals :
Level      Normal             Basic         Studentized
95%   ( 96.8, 108.7 )   ( 97.0, 109.1 )   ( 96.7, 109.0 )

Level     Percentile            BCa
95%   ( 96.2, 108.3 )   ( 96.2, 108.3 )
Calculations and Intervals on Original Scale

# Kontrolle: Indizes (n_R + 1) * alpha/2 und (n_R + 1) * (1-alpha/2)
> (idx <- trunc((nR + 1) * c(alpha/2, 1 - alpha/2)))
[1] 25 975

> tStar <- (Mstar-M) / sqrt(S2Mstar)        # t*
> tCrit <- sort(tStar)[idx]                 # 2.5%, 97.5%-Quantil von t*
> zCrit <- qnorm(c(alpha/2, 1 - alpha/2))   # 2.5%, 97.5%-Quantil N(0, 1)
> (ciBasic <- 2*M - sort(Mstar)[idx])       # klassisches VI
[1] 109.11192  96.97443

> (ciPerc <- sort(Mstar)[idx])              # Perzentil-VI
[1] 96.19981 108.33731

> (ciNorm <- M-biasM - zCrit*sd(Mstar))     # N(0, 1)-VI
[1] 108.74383  96.75861

> (ciT <- M - tCrit*sqrt(S2M))              # t-VI
[1] 108.99497  96.70034
```

Bei der manuellen Umsetzung der Bootstrap-Schätzungen sollen als Maß für deren Güte die kumulierten relativen Häufigkeiten von $t^\star = \frac{M^\star - M}{S^\star_M}$ mit der Verteilungsfunktion von $t = \frac{M-\mu_0}{s/\sqrt{n}}$ verglichen werden (mit s als korrigierter Streuung). Im Fall n unabhängiger Realisierungen einer normalverteilten Variable ist dies die t_{n-1} Verteilung (Abb. 11.2).

```
# M*, S(M)* und t* aus Bootstrap-Replikationen
> res    <- replicate(nR, getM(DV, sample(seq(along=DV), replace=TRUE)))
> Mstar  <- res[1, ]                        # M*
> SMstar <- sqrt(res[2, ])                  # S(M)*
> tStar  <- (Mstar-mean(DV)) / SMstar       # t*

# kumulierte relative Häufigkeiten von t*
> plot(tStar, ecdf(tStar)(tStar), col="gray60", pch=1,
+      xlab="t* bzw. t", ylab="P(T <= t)",
+      main="t*: Kumulierte rel. Häufigkeiten und Verteilungsfunktion")

# theoretische Verteilungsfunktion von t und Legende
> curve(pt(x, N-1), lwd=2, add=TRUE)
```

```
> legend(x="topleft", lty=c(NA, 1), pch=c(1, NA),
+        lwd=c(2, 2), col=c("gray60", "black"), legend=c("t*", "t"))
```

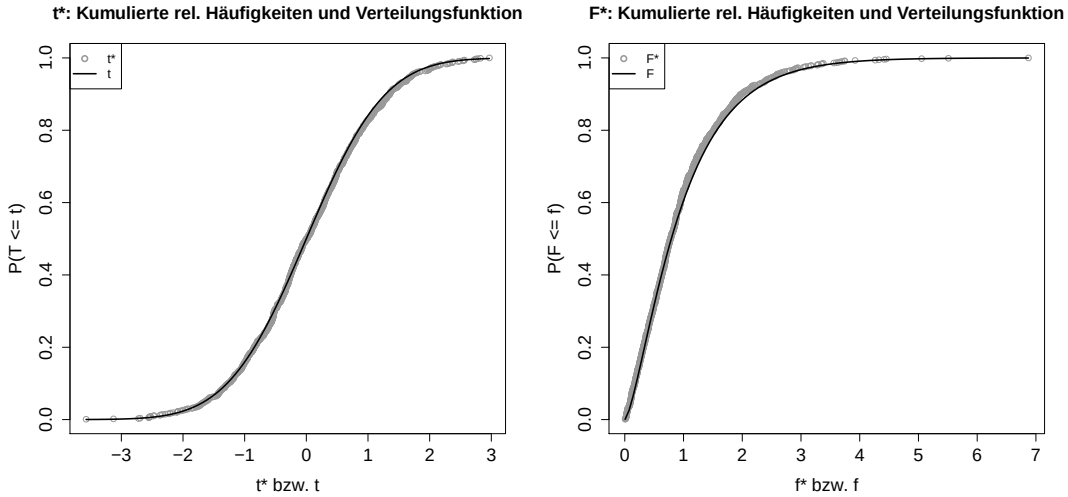

Abbildung 11.2: Kumulierte relative Häufigkeiten von $t^\star$ aus Bootstrap-Replikationen mit theoretischer Verteilungsfunktion t_{n-1}. Kumulierte relative Häufigkeiten von $F^\star$ aus Bootstrap-Replikationen mit theoretischer Verteilungsfunktion $F_{p-1,\,n-p}$.

11.1.3 Bootstrap-Vertrauensintervalle für $\mu_2 - \mu_1$

In der Situation eines t-Tests für zwei unabhängige Stichproben (vgl. Abschn. 7.2.2) kann bootstrapping eine nonparametrische Schätzung des Konfidenzintervalls für die Differenz der Erwartungswerte $\theta = \mu_2 - \mu_1$ liefern. In der Basisstichprobe ist die Differenz der Gruppenmittelwerte $\hat{\theta} = M_2 - M_1$ ein Schätzer für θ, in jeder Replikation analog $\hat{\theta}^\star = M_2^\star - M_1^\star$. Für die Replikationen ist zu beachten, dass jeweils innerhalb jeder Gruppe mit Zurücklegen aus der Basisstichprobe gezogen wird, damit Gruppenzugehörigkeit und Gruppengrößen erhalten bleiben. Dies lässt sich mit dem Argument `strata` von `boot()` erreichen. Als Beispiel diene jenes aus Abschn. 7.2.2 mit einer bei Frauen und Männern erhobenen Variable.

```
# Datensatz aus Variable bei Männern, bei Frauen und Gruppierungsfaktor
> n1  <- 18                                    # Stichprobenumfang 1
> n2  <- 21                                    # Stichprobenumfang 2
> DVm <- rnorm(n1, 180, 10)                    # Stichprobe Männer
> DVf <- rnorm(n2, 175, 6)                     # Stichprobe Frauen
> tDf <- data.frame(DV=c(DVm, DVf),   # Variable bei Männern und Frauen
+                   IV=factor(rep(c("m", "f"), c(n1, n2))))

# Funktion, um Differenz der Mittelwerte von f und m zu berechnen
> getDM <- function(dat, idx) {
+     # Gruppenmittelwerte für durch idx definierte Beobachtungen
+     Mfm <- aggregate(DV ~ IV, data=dat, subset=idx, FUN=mean)
+     -diff(Mfm$DV)      # M-Differenz, Reihenfolge f-m wie in t.test()
+ }
```

```
> library(boot)                              # für boot(), boot.ci()
> bsTind <- boot(tDf, statistic=getDM, strata=tDf$IV, R=999)
> boot.ci(bsTind, conf=0.95, type=c("basic", "bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 999 bootstrap replicates
CALL :
boot.ci(boot.out = bsTind, conf = 0.95, type = c("basic", "bca"))

Intervals :
Level      Basic                 BCa
95%   (-8.157,  0.878 )   (-8.529,  0.532 )
Calculations and Intervals on Original Scale
```

Als Vergleich diene das parametrische Konfidenzintervall aus dem zugehörigen t-Test.

```
> tt <- t.test(DV ~ IV, alternative="two.sided", var.equal=TRUE, data=tDf)
> tt$conf.int
[1] -8.3796152  0.8963473
```

Die analoge Situation mit zwei abhängigen Stichproben lässt sich auf den Fall einer Stichprobe zurückführen, indem eine Differenzvariable aus der jeweils pro Person berechneten Differenz beider Beobachtungen gebildet wird (vgl. Abschn. 7.2.3).

11.1.4 Lineare Modelle: case resampling

Die in Abschn. 6.2.2 und 6.3 vorgestellten Tests der Parameter einer linearen Regression setzen die Gültigkeit verschiedener Annahmen voraus, deren Plausibilität mit Methoden untersucht werden kann, wie sie Abschn. 6.5 erläutert. Sind diese Annahmen verletzt, sind die berechneten Standardfehler der Parameterschätzer womöglich verzerrt und führen zu falschen p-Werten. Oft eignen sich in diesem Fall Bootstrap-Verfahren, um angemessenere Vertrauensintervalle für die Regressionsgewichte zu erhalten.

Als Beispiel sei auf die Daten der multiplen linearen Regression in Abschn. 6.3.1 zurückgegriffen, die das Körpergewicht anhand der Prädiktoren Körpergröße, Alter und der für Sport aufgewendeten Zeit vorhersagen soll. Die Daten wurden unter gültigen Modellannahmen simuliert, was es hier erlaubt, die korrekten Standardfehler und Konfidenzintervalle der parametrischen Regressionsanalyse zur Validierung der Bootstrap-Ergebnisse zu verwenden.

```
> sqrt(diag(vcov(fitHAS)))            # parametrische Standardfehler
(Intercept)      height         age       sport
 8.18162620  0.04590320  0.04033012  0.01201414

> confint(fitHAS)                     # parametrische Konfidenzintervalle
                 2.5 %      97.5 %
(Intercept)  -8.1391666   24.3416327
height        0.4237764    0.6060106
age          -0.3667003   -0.2065910
```

```
sport        -0.4398003   -0.3921046
```

Die erste Methode, bootstrapping auf die Situation eines linearen Modells wie das der Regression anzuwenden, besteht im *case resampling*, auch *Vektor-Sampling* genannt: Hierfür werden die beobachteten Werte aller Variablen als zufällig betrachtet, insbesondere auch jene der Prädiktoren. Für jede Replikation wird nun aus der Menge der beobachteten Personen (*cases*) mit Zurücklegen eine Stichprobe vom ursprünglichen Umfang gezogen. Die Werte der jeweils gezogenen Personen für Prädiktoren und Kriterium liegen der Anpassung des Regressionsmodells für eine Replikation zugrunde. So liefert jede Replikation eine Schätzung für jeden Regressionsparameter, woraus sich deren Bootstrap-Verteilungen – und damit Standardfehler und Konfidenzintervalle bestimmen lassen.

Die Methode gilt als robust, da sich mit jeder Replikation die Design-Matrix des Modells ändert und daher Ausreißer oder übermäßig einflussreiche Beobachtungen die Parameterschätzungen nicht immer verzerren. Case resampling ist auch für verallgemeinerte lineare Modelle geeignet (vgl. Kap. 8).

Die im Aufruf von `boot()` für das Argument `statistic` genannte Funktion muss unter `data` einen Datensatz mit den ursprünglichen Werten von Prädiktoren und Kriterium akzeptieren sowie die Parameterschätzungen für die über den Indexvektor `idx` definierte Replikation zurückgeben.

```
# berechne für Daten dat und Replikation idx die Regressionsgewichte
> getRegr <- function(dat, idx) {
+     bsFit <- lm(weight ~ height + age + sport, subset=idx, data=dat)
+     coef(bsFit)                      # Regressionsgewichte Replikation
+ }

> library(boot)                        # für boot(), boot.ci()
> nR <- 999                            # Anzahl BS-Replikationen
> (bsRegr <- boot(regrDf, statistic=getRegr, R=nR))
ORDINARY NONPARAMETRIC BOOTSTRAP
Call:
boot(data = regrDf, statistic = getRegr, R = nR)

Bootstrap Statistics :
       original           bias  std. error
t1*   8.1012331  -0.1025429121  7.86996801
t2*   0.5148935   0.0001543189  0.04557696
t3*  -0.2866457   0.0017596402  0.04037238
t4*  -0.4159525   0.0004067643  0.01125020
```

Die Zeilen der Ausgabe nennen die Bootstrap-Kennwerte für jeweils einen Koeffizienten in der Reihenfolge des Rückgabewertes der im Aufruf von `boot()` für `statistic` verwendeten Funktion. Für die gewählten Daten stimmen die Bootstrap-Standardfehler weitgehend mit jenen der parametrischen Regressionsanalyse überein. Das Vertrauensintervall für jeden Parameter liefert wieder `boot.ci()`, wobei mit dem Argument `index=`⟨`Nummer`⟩ auszuwählen ist, für welchen Parameter θ_j das Vertrauensintervall benötigt wird.

```
# BCa-Konfidenzintervalle für die Regressionsparameter
> boot.ci(bsRegr, conf=0.95, type="bca", index=1)$bca       # b0
[1,] 0.95 22.45 972.23 -6.975915 24.44877

> boot.ci(bsRegr, conf=0.95, type="bca", index=2)$bca       # height
[1,] 0.95 27.62 977.4 0.4238508 0.5991131

> boot.ci(bsRegr, conf=0.95, type="bca", index=3)$bca       # age
[1,] 0.95 26.18 976.23 -0.3637959 -0.2059037

> boot.ci(bsRegr, conf=0.95, type="bca", index=4)$bca       # sport
[1,] 0.95 22.45 972.22 -0.4382711 -0.3939868
```

Das erste Element des Vektors in der Komponente `bca` der von `boot.ci()` zurückgegebenen Liste nennt die Breite des Intervalls, die beiden folgenden Elemente die zu den $\frac{\alpha}{2}$- und $1 - \frac{\alpha}{2}$-Quantilen gehörenden Indizes für den Vektor der sortierten Werte,[3] die letzten beiden Elemente sind schließlich die gesuchten Intervallgrenzen. Auch die Konfidenzintervalle gleichen hier jenen aus der parametrischen Regressionsanalyse.

11.1.5 Lineare Modelle: model-based resampling

Varianzanalysen lassen sich wie eine Regression als lineares Modell formulieren, wobei die Gruppenzugehörigkeiten (Werte der UV) die Rolle der Prädiktoren einnehmen und eine Zielvariable (AV) das Kriterium bildet (vgl. Abschn. 12.9). Die Gruppenzugehörigkeiten sind im Gegensatz zu den Prädiktorwerten der Regression experimentell festgelegt und sollten deshalb für alle resamples konstant sein. Eine Methode, um dies zu gewährleisten, besteht im *model-based resampling*. Hier werden nur die Werte der AV als mit zufälligen Fehlern behaftet betrachtet. Dieser Logik folgend berechnet man zunächst für die Daten der Basisstichprobe die Modellvorhersage $\hat{Y}$ sowie die zugehörigen Residuen $E = Y - \hat{Y}$.

Die Residuen werden daraufhin zu $\frac{E}{\sqrt{1-h}}$ reskaliert, damit sie überall die bedingte theoretische Streuung σ besitzen (vgl. Abschn. 12.9.4). Dabei ist h die Variable Hebelwert (vgl. Abschn. 6.5.1, 6.5.2). Um zu gewährleisten, dass E im Mittel 0 ist, sollte das Modell einen absoluten Term β_0 einschließen oder zentrierte Variablen verwenden.

Für jede Replikation wird aus $\frac{E}{\sqrt{1-h}}$ mit Zurücklegen eine Stichprobe $E^\star$ vom ursprünglichen Umfang gezogen. Diese Resample-Residuen werden zu $\hat{Y}$ addiert, um die Resample-AV $Y^\star = \hat{Y} + E^\star$ zu erhalten. Für jede Bootstrap-Schätzung der Parameter wird dann die Varianzanalyse mit $Y^\star$ und der ursprünglichen UV berechnet. Model-based resampling gilt als effizienter als case resampling, allerdings auch als anfälliger für eine falsche Modell-Spezifizierung.

Residuen E und Modellvorhersage $\hat{Y}$ für die Basisstichprobe lassen sich sowohl für das H_0- wie für das H_1-Modell bilden: Unter der H_0 der einfaktoriellen Varianzanalyse unterscheiden sich die Erwartungswerte in den Gruppen nicht, als Vorhersage $\hat{Y}$ ergibt sich damit für alle Personen

[3] Die Indizes sind hier trotz der 999 Replikationen nicht ganzzahlig (25 und 975), da die dem BC_a-Intervall zugrundeliegende Korrektur über die Verschiebung der Intervallgrenzen funktioniert. Vergleiche etwa das Perzentil-Intervall für θ_1 aus `boot.ci(bsRegr, conf=0.95, type="perc", index=1)$percent`.

der Gesamtmittelwert M. Wird dieses Modell mit der Formel ⟨`AV`⟩ `~ 1` für die Basisstichprobe angepasst, erzeugt das bootstrapping eine Approximation der Verteilung von $\hat{\theta}$ (etwa des F-Bruchs) unter H_0. Dies erlaubt es, p-Werte direkt als Anteil der resamples zu berechnen, bei denen $\hat{\theta}^\star$ i. S. der H_1 mindestens so extrem wie $\hat{\theta}$ ist.[4]

Als Beispiel sei auf die Daten der einfaktoriellen Varianzanalyse in Abschn. 7.3.1 zurückgegriffen. Die Daten wurden unter gültigen Modellannahmen simuliert, was es hier erlaubt, die Verteilung der Bootstrap-Schätzer $F^\star$ mit der F-Verteilung zu vergleichen und den Bootstrap-p-Wert mit dem p-Wert der F-Verteilung zu validieren.

```
> anBase <- anova(lm(DV ~ IV))              # ANOVA Basisstichprobe
> Fbase  <- anBase["IV", "F value"]         # F-Wert Basisstichprobe
> (pBase <- anBase["IV", "Pr(>F)"])         # p-Wert Basisstichprobe
[1] 0.002921932

# H0-Modell in Basisstichprobe anpassen
> fit0 <- lm(DV ~ 1)                        # Modellanpassung
> E    <- residuals(fit0)                   # ursprüngliche Residuen
> Er   <- E / sqrt(1-hatvalues(fit0))       # reskalierte Residuen
> Yhat <- fitted(fit0)                      # ursprüngliche Vorhersage

# ANOVA-Parameter für ursprüngliche Gruppenzugehörigkeiten und Y*
> getAnova <- function(dat, idx) {
+     Ystar <- Yhat + Er[idx]               # Resample-AV Y* = Y^ + E*
+     anBS  <- anova(lm(Ystar ~ IV, data=dat))      # ANOVA Resample-AV
+     anBS["IV", "F value"]                 # F*-Wert Replikation
+ }

> library(boot)                             # für boot(), boot.ci()
> nR       <- 999                           # Anzahl Replikationen
> (bsAnova <- boot(dfCRp, statistic=getAnova, R=nR))
ORDINARY NONPARAMETRIC BOOTSTRAP
Call:
boot(data = dfCRp, statistic = getAnova, R = nR)

Bootstrap Statistics :
     original        bias    std. error
t1*  4.861867   -3.887228     0.8136521
```

Der für den p-Wert notwendige Vergleich $F^\star \geq F$ wird hier in zwei Vergleiche aufgeteilt, um robuster gegenüber Problemen der numerischen Repräsentation von Gleitkommazahlen zu sein (vgl. Abschn. 1.3.6). Dafür wird nur auf ungefähre, nicht auf exakte Gleichheit von $F^\star$ und F geprüft.

[4] Der p-Wert kann bei Monte-Carlo-Approximationen zur höheren Genauigkeit nach Hinzufügen eines zusätzlichen extremeren Falles gebildet werden: Ist n_R die Anzahl der generierten resamples und $n^\star$ die Anzahl der Fälle, bei denen $\hat{\theta}^\star$ mindestens so extrem wie $\hat{\theta}$ ist, setzt man $p = \frac{n^\star+1}{n_R+1}$. Auf diese Weise wird vermieden, dass der p-Wert exakt 0 werden kann.

```
# F* aus Bootstrap-Replikationen
> Fstar <- bsAnova$t

# numerische Toleranz wg. Gleitkomma-Arithmetik
> tol <- .Machine$double.eps^0.5

# Fstar größer Basis-F ODER Fstar ungefähr gleich Basis-F?
> FsIsGEQ <- (Fstar > Fbase) | (abs(Fstar - Fbase) < tol)

# p-Wert: Anteil der mindestens so extremen F* wie Basis F
> (pValBS <- (sum(FsIsGEQ) + 1) / (length(Fstar) + 1))
[1] 0.004

# Grafik: kumulierte relative Häufigkeiten von F*
> plot(Fstar, ecdf(Fstar)(Fstar), col="gray60", pch=1,
+      xlab="f* bzw. f", ylab="P(F <= f)",
+      main="F*: Kumulierte rel. Häufigkeiten und Verteilungsfunktion")

# theoretische Verteilungsfunktion von t und Legende
> curve(pf(x, P-1, sum(Nj) - P), lwd=2, add=TRUE)
> legend(x="topleft", lty=c(NA, 1), pch=c(1, NA), lwd=c(2, 2),
+        col=c("gray60", "black"), legend=c("F*", "F"))
```

Die Verteilung der $F^\star$ ist hier der theoretischen F-Verteilung sehr ähnlich (Abb. 11.2), was auch für die Größenordnung des Bootstrap-p-Wertes verglichen mit dem p-Wert der ursprünglichen Varianzanalyse gilt.

11.1.6 Lineare Modelle: wild bootstrap

Eine Variante des model-based resampling ist der *wild bootstrap* für Situationen, in denen Heteroskedastizität vorliegt. Hier wird $E^\star$ aus dem Produkt $\frac{E}{\sqrt{1-h}} \cdot U$ gebildet. Dabei ist U eine unabhängige Zufallsvariable mit $E(U) = 0$ und $E(U^2) = 1$, deren Werte für jede Replikation simuliert werden müssen.

- Eine Wahl für U sind dichotome Variablen mit F_1-Verteilung, die den Wert $\frac{-(\sqrt{5}-1)}{2}$ mit Wahrscheinlichkeit $p = \frac{\sqrt{5}+1}{2\sqrt{5}}$ annehmen und den Wert $\frac{\sqrt{5}+1}{2}$ mit Wahrscheinlichkeit $1 - p = \frac{\sqrt{5}-1}{2\sqrt{5}}$.
- Eine alternative Wahl für U sind ebenfalls dichotome Variablen mit F_2- bzw. Rademacher-Verteilung: Sie nehmen die Werte -1 und 1 jeweils mit Wahrscheinlichkeit $\frac{1}{2}$ an, drehen also das ursprüngliche Vorzeichen jedes Residuums zufällig um.

```
> getAnovaWild <- function(dat, idx) {
+     n  <- length(idx)                    # Größe der Replikation
+     # Ur: Rademacher Variablen, Uf: zweite Variante
+     Ur <- sample(c(-1, 1), size=n, replace=TRUE, prob=c(0.5, 0.5))
+     Uf <- sample(c(-(sqrt(5) - 1)/2, (sqrt(5) + 1)/2), size=n,
```

```
+                    replace=TRUE, prob=c((sqrt(5) + 1)/(2*sqrt(5)),
+                                         (sqrt(5) - 1)/(2*sqrt(5))))
+
+     Ystar <- Yhat + (Er*Ur)[idx]        # E* mit Rademacher-Variable
+     # Ystar <- Yhat + (Er*Uf)[idx]      # zweite Variante
+     anBS  <- anova(lm(Ystar ~ IV, data=dat))    # ANOVA Resample-AV
+     anBS["IV", "F value"]               # F*-Wert Replikation
}

# fortfahren wie oben
> bsAnovaW <- boot(dfCRp, statistic=getAnovaWild, R=nR)
> FstarW   <- bsAnova$t                 # F*-Werte
> tol      <- .Machine$double.eps^0.5   # numerische Toleranz

# Fstar größer Basis-F ODER Fstar ungefähr gleich Basis-F?
> FsIsGEQ  <- (FstarW > Fbase) | (abs(FstarW - Fbase) < tol)
> (pValBSw <- (sum(FsIsGEQ) + 1) / (length(FstarW) + 1))    # ...
```

Werden Vorhersage und Residuen für die Basisstichprobe für das H_1 Modell mit der Formel ⟨AV⟩ ~ ⟨UV⟩ berechnet, erzeugt das modellbasierte bootstrapping eine Verteilung von $\hat{\theta}^\star$, aus der sich Vertrauensintervalle für θ bestimmen lassen (vgl. Abschn. 11.1.2, 11.1.4). Für die Bootstrap-Verteilung von $F^\star$ sollte dabei auf das BC_a-Intervall zurückgegriffen werden, da diese Verteilung typischerweise schief ist und der Schätzer einen deutlichen bias aufweist.

11.2 Parametrisches Bootstrapping

Parametrisches bootstrapping setzt ein statistisches Modell dafür voraus, wie Beobachtungen einer Variable Y aus der Kombination theoretischer Parameter θ_j und zufälliger Fehler zustande kommen sollen. Jede Replikation ergibt sich dann als neue modellgerechte Simulation eines Datensatzes: Dafür werden die plug-in Schätzer der Basisstichprobe $\hat{\theta}_j$ im Modell anstelle der θ_j eingesetzt und die Fehler zufällig entsprechend der für sie angenommenen Verteilung simuliert. Das weitere Vorgehen ist identisch zum nonparametrischen bootstrap: Für jede Replikation schätzt man die Parameter $\hat{\theta}_j^\star$, aus deren empirischer Verteilung über alle Replikationen hinweg sich Konfidenzintervalle für die θ_j bestimmen lassen.

Auch für den parametrischen bootstrap eignet sich die Funktion `boot()` aus dem gleichnamigen Paket (vgl. Abschn. 11.1.1), wenn das Argument `sim="parametric"` gesetzt wird.

```
> boot(data=⟨Basisstichprobe⟩, statistic=⟨Funktion⟩, R=⟨# Replikationen⟩,
+      sim="parametric", ran.gen=⟨Funktion⟩, mle=⟨Schätzer Basisstichprobe⟩)
```

Für `data` sind die Daten der Basisstichprobe als Vektor oder Datensatz zu übergeben. Für `statistic` ist eine Funktion mit ihrerseits einem Argument zu nennen – den Daten der Replikation, auf deren Basis sich die Schätzungen $\hat{\theta}^\star$ berechnen lassen. `boot()` ruft für jede der `R` vielen Replikationen `statistic` auf und übergibt einen neu simulierten Datensatz mit derselben Form wie jener der Basisstichprobe. Das Ergebnis von `statistic` muss ein Vektor der $\hat{\theta}_j^\star$ sein.

Die Simulation eines neuen Datensatzes erfolgt in einer eigenen Funktion, die an `ran.gen` zu übergeben ist. Diese muss ihrerseits zwei Argumente besitzen: Das erste für die Daten der Basisstichprobe und das zweite für deren Schätzer $\hat{\theta}_j$. Diese Schätzer sind zunächst für das Argument `mle` (*maximum likelihood estimate*) von `boot()` zu nennen. Allgemein kann `mle` auch ein komplexeres Objekt sein, aus dem sich innerhalb von `ran.gen` die $\hat{\theta}_j$ ableiten lassen, etwa eine von `lm()` oder `glm()` für die Basisstichprobe angepasste Regression. Das Ergebnis von `ran.gen` ist ein simulierter Datensatz mit derselben Form wie jener der Basisstichprobe, also mit denselben Variablennamen desselben Typs. Er wird pro Replikation an die Funktion `statistic` übergeben.

11.2.1 Bootstrap-Vertrauensintervalle für $\mu_2 - \mu_1$

Als Beispiel sollen in der Situation eines t-Tests für zwei unabhängige Stichproben (vgl. Abschn. 7.2.2, 11.1.3) Konfidenzintervalle für die Differenz der Erwartungswerte $\theta = \mu_2 - \mu_1$ konstruiert werden, wenn nicht von Varianzhomogenität auszugehen ist.

```
# Datensatz aus Variable bei Männern, bei Frauen und Gruppierungsfaktor
> n1  <- 18                                   # Stichprobenumfang 1
> n2  <- 21                                   # Stichprobenumfang 2
> DVm <- rnorm(n1, 180, 10)                   # Stichprobe Männer
> DVf <- rnorm(n2, 175, 6)                    # Stichprobe Frauen
> tDf <- data.frame(DV=c(DVm, DVf),   # Variable bei Männern und Frauen
+                   IV=factor(rep(c("m", "f"), c(n1, n2))))
```

Für die Simulation neuer Daten sei angenommen, dass sich in jeder Gruppe j die Messwerte y_{ij} als Summe $\mu_j + \epsilon_{ij}$ des Erwartungswerts μ_j und eines normalverteilten Fehlers ϵ_{ij} mit Erwartungswert 0 und gruppenspezifischer Varianz σ_j^2 ergeben. Die Simulations-Funktion nimmt dafür einerseits den Basis-Datensatz entgegen, andererseits eine Liste mit zwei Komponenten: In der ersten Komponente wurden alle ursprünglichen Werte y_{ij} durch ihren Gruppenmittelwert M_j ersetzt – dem Schätzer für μ_j in der Basisstichprobe. Analog wurden in der zweiten Komponente alle y_{ij} durch ihre unkorrigierte Gruppenstreuung S_j ersetzt, da die unkorrigierte Gruppenvarianz S_j^2 der plug-in und gleichzeitig maximum likelihood Schätzer in der Basisstichprobe für die gruppenspezifische Varianz σ_j^2 ist.

```
# Funktion für unkorrigierte Varianz S^2 eines Vektors x
> getSDML <- function(x) {
+     sqrt(cov.wt(as.matrix(x), method="ML")$cov[1, 1])
+ }

# ersetze Daten der Basisstichprobe jeweils durch ihren
# Gruppenmittelwert bzw. durch ihre unkorrigierte Gruppenstreuung
> MSD <- list( M=ave(tDf$DV, tDf$IV, FUN=mean),
+              SD=ave(tDf$DV, tDf$IV, FUN=getSDML))

# Simulation: Argument dat für Daten der Basisstichprobe
# Argument MSD für Liste mit Gruppen-M und -SD der Basisstichprobe
> rGenMD <- function(dat, MSD) {
```

```
+     out <- dat
+     # simuliere modellgerecht neue Messwerte
+     out$DV <- MSD$M + rnorm(length(MSD$M), mean=0, sd=MSD$SD)
+     return(out)
+ }
```

Die Verteilung der Differenz der Gruppenmittelwerte $M_2^\star - M_1^\star$ aus jeder Replikation dient schließlich als Grundlage der Konfidenzintervalle für die Differenz der Erwartungswerte $\theta = \mu_2 - \mu_1$.

```
# Funktion für Differenz der Gruppenmittelwerte in einer Replikation
> getMD <- function(dat) {
+     -diff(tapply(dat$DV, dat$IV, FUN=mean))
+ }

# parametrischer bootstrap
> library(boot)                              # für boot(), boot.ci()
> nR   <- 999                                # Anzahl Replikationen
> bsMD <- boot(dat, statistic=getMD, R=nR,
+              sim="parametric", mle=MSD, ran.gen=rGenMD)
```

Das von `boot()` erzeugte Objekt ist das erste Argument für die Funktion `boot.ci()`, die das zweiseitige Konfidenzintervall für θ bestimmt (vgl. Abschn. 11.1.2).

```
> boot.ci(bsMD, conf=0.95, type="basic")$basic
     conf
[1,] 0.95 975 25 -8.326444 1.179377
```

Als Vergleich diene das parametrische Konfidenzintervall aus dem zugehörigen t-Test ohne Annahme von Varianzhomogenität.

```
> tt <- t.test(DV ~ IV, alternative="two.sided", var.equal=FALSE, data=tDf)
> tt$conf.int
[1] -8.648956  1.165688
```

11.2.2 Verallgemeinerte lineare Modelle

Für die Anwendung des parametrischen bootstrap bei einem verallgemeinerten linearen Modell (vgl. Kap. 8) soll die Poisson-Regression mit den Prädiktoren X_1 und X_2 aus Abschn. 8.4.1 als Beispiel dienen. Für (verallgemeinerte) lineare Modelle ist es besonders einfach, auf Basis einer bereits mit `lm()` oder `glm()` angepassten Regression modellgerecht neue Werte der vorhergesagten Variable zu simulieren, da R hierfür die Funktion `simulate()` bereitstellt. Sie akzeptiert ein von `lm()` oder `glm()` erzeugtes Objekt und liefert in der ersten Komponente der zurückgegebenen Liste die neu simulierten Werte der vorhergesagten Variable. Dafür leitet `simulate()` das statistische Modell mit Schätzern $\hat{\theta}_j$ der Basisstichprobe sowie die angenommene Verteilung der Fehler aus dem übergebenen Objekt selbst ab.

```
# Simulation: Argument dat erhält Daten der Basisstichprobe
# Argument mle erhält Ergebnis von glm() für Basisstichprobe
> rGen <- function(dat, mle) {
+     out   <- dat
+     out$Y <- simulate(mle)[[1]]
+     return(out)
+ }
```

In jeder Replikation ist eine Poisson-Regression für die übergebenen simulierten Daten anzupassen, aus der sich dann die Regressionskoeffizienten $b_j^\star$ als Schätzer $\hat{\theta}_j^\star$ ergeben.

```
> getPois <- function(dat) {
+     glmFit <- glm(Y ~ X1 + X2, family=poisson(link="log"), data=dat)
+     coef(glmFit)
+ }

glmFitP
```

```
# parametrischer bootstrap
> library(boot)                              # für boot(), boot.ci()
> nR <- 999                                  # Anzahl Replikationen
> bsPois <- boot(dfCount, statistic=getPois, R=nR,
+                sim="parametric", mle=glmFitP, ran.gen=rGen)
```

Das von `boot()` erzeugte Objekt ist als erstes Argument für die Funktion `boot.ci()` anzugeben, die das zweiseitige Konfidenzintervall für die θ_j bestimmt (vgl. Abschn. 11.1.2).

```
> boot.ci(bsPois, conf=0.95, type="basic", index=1)$basic
     conf
[1,] 0.95 975 25 -0.04471856 0.3480377

> boot.ci(bsPois, conf=0.95, type="basic", index=2)$basic
     conf
[1,] 0.95 975 25 -0.3028855 -0.220306

> boot.ci(bsPois, conf=0.95, type="basic", index=3)$basic
     conf
[1,] 0.95 975 25 -0.0004523571 0.04146108
```

Als Vergleich dienen die parametrischen Konfidenzintervalle mit der profile likelihood Methode für die in der Basisstichprobe angepasste Poisson-Regression:

```
> confint(glmFitP)
                    2.5 %       97.5 %
(Intercept)  -0.0534587597   0.33602778
X1           -0.3036802852  -0.22116851
X2           -0.0007271649   0.04155718
```

11.3 Permutationstests

Permutationstests (Chihara & Hesterberg, 2011; Good, 2004) verwenden analog zu Bootstrap-Verfahren Permutationen einer festen Basisstichprobe als resamples, um ausgehend von der empirischen Verteilung der für diese resamples berechneten Schätzer $\hat{\theta}^\star$ Aussagen über die theoretische Verteilung einer Teststatistik $\hat{\theta}$ abzuleiten. Jede Permutation wird dabei so gebildet, dass sie dieselben Werte der Basisstichprobe umfasst, die Reihenfolge der Beobachtungen i. S. der Zusammensetzung der Untersuchungsbedingungen jedoch im Einklang mit der H_0 des Tests im gegebenen Untersuchungsdesign variiert.

Stimmen etwa unter H_0 die Verteilungen einer Zielvariable (AV) in zwei Bedingungen überein, ist die Zugehörigkeit der Beobachtungen zu einer Bedingung für die Ausprägung der AV unwesentlich und kann deshalb permutiert werden – jede Beobachtung hätte genauso gut aus jeder Bedingung stammen können.[5] Im Fall von abhängigen Stichproben ist dies separat innerhalb jedes Beobachtungsobjekts zu tun, bei unabhängigen Stichproben entsprechend über Beobachtungsobjekte hinweg. Sind zwei Variablen unter H_0 unabhängig, ist es für die Ausprägung der zweiten AV irrelevant, welchen Wert die erste AV besitzt – die Zuordnung von Werten der zweiten AV zu jenen der ersten kann also permutiert werden.

Die empirische Verteilung von $\hat{\theta}^\star$ schätzt die Verteilung von $\hat{\theta}$ unter H_0, was es erlaubt, p-Werte direkt zu berechnen: Dies ist der Anteil der Permutationen, bei denen $\hat{\theta}^\star$ i. S. der H_1 mindestens so extrem wie $\hat{\theta}$ ist. Da die Anzahl möglicher Permutationen (und damit der Rechenaufwand) sehr schnell mit der Stichprobengröße wächst, ist es nur in Spezialfällen oder bei kleinen Stichproben möglich, die empirische Verteilung von $\hat{\theta}^\star$ exakt zu bestimmen. Ist die praktische Berechenbarkeit aller $\hat{\theta}^\star$ nicht gegeben, lassen sich Permutationstests als Monte-Carlo-Verfahren durchführen, die $\hat{\theta}^\star$ nur für eine zufällige Auswahl von Permutationen berechnen (vgl. Abschn. 11.1.5, Fußnote 4).

Das bereits in Abschn. 10.5.8, 10.5.10 und 10.5.11 verwendete Paket `coin` stellt für viele Hypothesen exakte, oder aber durch Monte-Carlo-Verfahren approximierte Permutationstests bereit, für deren konventionelle Prüfung sonst auf parametrische Tests oder nonparametrische Verfahren mit einer asymptotisch gültigen Verteilung der Teststatistik zurückgegriffen werden muss – für eine Übersicht vgl. `vignette("coin")`.[6]

11.3.1 Test auf gleiche Lageparameter in unabhängigen Stichproben

Im Beispiel soll linksseitig getestet werden, ob die Verteilung einer quantitativen AV in (hier zwei) unabhängigen Stichproben übereinstimmt. Bei Verteilungen gleicher Form ist dies der Fall, wenn ihre Lageparameter identisch sind. Anders als etwa beim Wilcoxon-Rangsummen-Test (vgl. Abschn. 10.5.4) sollen hier die ursprünglichen AV-Werte ohne Rangtransformation Verwendung finden, wofür sich `oneway_test()` aus dem Paket `coin` eignet.

```
> oneway_test(formula=⟨Modellformel⟩, data=⟨Datensatz⟩, subset=⟨Indexvektor⟩,
+             alternative=c("two.sided", "less", "greater"),
+             distribution=⟨Verteilungstyp⟩)
```

[5] Formal muss das Kriterium der *Austauschbarkeit* erfüllt sein (Good, 2004).

[6] Auch die Pakete `resample` und `vegan` (Oksanen et al., 2014) bieten flexible Möglichkeiten, um Permutationstests für verschiedenen Untersuchungs-Designs umzusetzen.

Unter `formula` sind Daten und Gruppierungsvariable als Modellformel ⟨`AV`⟩ `~` ⟨`UV`⟩ einzugeben, wobei ⟨`UV`⟩ ein Faktor derselben Länge wie ⟨`AV`⟩ ist und für jede Beobachtung die zugehörige UV-Stufe angibt. Geschieht dies mit Variablen aus einem Datensatz, muss dieser unter `data` eingetragen werden. Das Argument `subset` erlaubt es, nur eine Teilmenge der Fälle einfließen zu lassen, es erwartet einen entsprechenden Indexvektor, der sich auf die Zeilen des Datensatzes bezieht. Mit `alternative` wird festgelegt, ob die H_1 gerichtet (`"less"` bzw. `"greater"`) oder ungerichtet (`"two.sided"`) ist. Die Anzahl der zufälligen Permutationen, auf deren Basis die Verteilung der Teststatistik approximiert wird, lässt sich über `approximate(B=`⟨`Anzahl`⟩`)` für das Argument `distribution` übergeben. Mit der in bestimmten Situationen wählbaren Option `"exact"` für dieses Argument basiert der p-Wert auf der exakten Verteilung aller möglichen $\hat{\theta}^{\star}$.

```
> Nj    <- c(7, 8)                              # Gruppengrößen
> DVa   <- round(rnorm(Nj[1], 100, 20))         # Daten Gruppe A
> DVb   <- round(rnorm(Nj[2], 110, 20))         # Daten Gruppe B
> DVab  <- c(DVa, DVb)                          # Gesamt-Daten
> IVbtw <- factor(rep(c("A", "B"), Nj))         # Gruppenzugehörigkeit
> library(coin)                                 # für oneway_test()
> (ot <- oneway_test(DVab ~ IVbtw, alternative="less",
+                    distribution="exact"))
Exact 2-Sample Permutation Test
data: DVab by IVbtw (A, B)
Z = -2.0981, p-value = 0.01756
alternative hypothesis: true mu is less than 0
```

Das Ergebnis nennt den Wert der intern verwendeten Teststatistik unter `Z`,[7] gefolgt vom p-Wert unter `p-value`.

Die Hilfe-Seite von `support()` erläutert die Verwendung von `dperm()`, `qperm()` und `rperm()`, um Dichteverteilung und Quantile der Permutations-Teststatistik von `oneway_test()` zu ermitteln sowie aus ihr Zufallszahlen zu erzeugen. Die Permutationsverteilung sollte unimodal und symmetrisch sein, nicht unähnlich einer Normalverteilung (Abb. 11.3).

```
# Dichteverteilung der Teststatistik von oneway_test()
> supp <- support(ot)
> dens <- sapply(supp, dperm, object=ot)            # Dichte
> plot(supp, dens, xlab="Support", ylab=NA, pch=20,
+      main="Dichte Permutationsverteilung")

# Q-Q-Plot der Teststatistik von oneway_test() gegen Standard-NV
> qEmp <- sapply(ppoints(supp), qperm, object=ot)   # Quantile Teststat.
> qqnorm(qEmp, xlab="Quantile Normalverteilung",
+        ylab="Permutations-Quantile",
+        main="Permutations- vs. theoretische NV-Quantile")

> abline(a=0, b=1, lwd=2, col="blue")
```

[7] Für deren Wahl vgl. `vignette("coin_implementation")`.

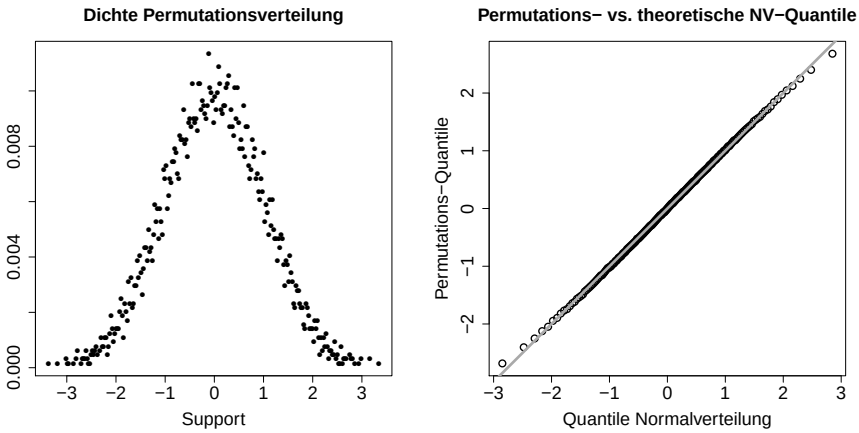

Abbildung 11.3: Dichteverteilung der Teststatistik des Permutationstests. Q-Q-Plot der Teststatistik des Permutationstests im Vergleich zur Standardnormalverteilung.

Zum Vergleich mit dem Permutationstest soll zunächst der p-Wert des analogen parametrischen t-Tests ermittelt werden (vgl. Abschn. 7.2.2). Bei der folgenden manuellen Kontrolle dient die Mittelwertsdifferenz zwischen beiden Gruppen als Teststatistik. Dabei ist zu beachten, dass $\hat{\theta}^\star$ hier nicht für alle $N!$ möglichen Permutationen der Gesamtstichprobe vom Umfang N bestimmt werden muss. Dies ist nur für jene Permutationen notwendig, die auch zu unterschiedlichen Gruppenzusammensetzungen führen, also nicht lediglich innerhalb jeder Gruppe die Reihenfolge vertauschen. Es gibt $\binom{N}{n_1}$ viele Möglichkeiten (Kombinationen, vgl. Abschn. 2.3.3), zwei Gruppen der Größe n_1 und n_2 zu bilden, dabei ist jede Kombination gleich wahrscheinlich.

```
# Vergleich: p-Wert aus analogem parametrischen t-Test
> t.test(DVab ~ IVbtw, alternative="less", var.equal=TRUE)$p.value
[1] 0.01483469

# Indizes aller unterschiedlichen Zusammensetzungen Gruppe A*
> idx  <- seq(along=DVab)               # Indizes Gesamt-Daten
> idxA <- combn(idx, Nj[1])             # alle n1-Kombinationen

# Mittelwertsdifferenz für gegebene Indizes x der Gruppe A*
> getDM  <- function(x) { mean(DVab[x]) - mean(DVab[!(idx %in% x)]) }
> DMstar <- apply(idxA, 2, getDM)       # M-Differenz aller Kombinationen
> DMbase <- mean(DVa) - mean(DVb)       # beobachtete M-Differenz
```

Der für den p-Wert notwendige Vergleich $\hat{\theta}^\star \leq \hat{\theta}$ wird hier in zwei Vergleiche aufgeteilt, um robuster gegenüber Problemen der numerischen Repräsentation von Gleitkommazahlen zu sein (vgl. Abschn. 1.3.6). Dafür wird nur auf ungefähre, nicht auf exakte Gleichheit von $\hat{\theta}^\star$ und $\hat{\theta}$ geprüft.

```
> tol <- .Machine$double.eps^0.5        # numerische Toleranz

# DMstar kleiner Basis-DM ODER DMstar ungefähr gleich Basis-DM?
> DMsIsLEQ <- (DMstar < DMbase) | (abs(DMstar - DMbase) < tol)
```

```
# p-Wert: Anteil der mind. so extremen Mittelwertsdifferenzen
> (pVal <- sum(DMsIsLEQ) / length(DMstar))
[1] 0.01756022
```

11.3.2 Test auf gleiche Lageparameter in abhängigen Stichproben

Beim Test auf Übereinstimmung von Verteilungen aus (hier zwei) abhängigen Stichproben hat die für `oneway_test()` anzugebende Modellformel die Form ⟨`AV`⟩ ~ ⟨`UV`⟩ | ⟨`BlockId`⟩. Dabei codiert ⟨`UV`⟩ als Faktor derselben Länge wie ⟨`AV`⟩ den Messzeitpunkt. ⟨`BlockId`⟩ ist ebenfalls ein solcher Faktor und gibt im Fall von Messwiederholung die Zugehörigkeit jedes Wertes zu einem Beobachtungsobjekt (bei gematchten Personen: zu einem Block) an.

```
> N      <- 12                              # Anzahl Personen
> id     <- factor(rep(1:N, times=2))       # Faktor Personen-ID
> DVpre  <- round(rnorm(N, 100, 20))        # Daten Zeitpunkt prä
> DVpost <- round(rnorm(N, 110, 20))        # Daten Zeitpunkt post
> DVpp   <- c(DVpre, DVpost)                # Gesamt-Daten

# Faktor Messzeitpunkt
> IV <- factor(rep(0:1, each=N), labels=c("pre", "post"))
> library(coin)                             # für oneway_test()
> oneway_test(DVpp ~ IV | id, alternative="less",
+             distribution=approximate(B=9999))
Approximative 2-Sample Permutation Test
data: DVpp by IV (pre, post) stratified by id
Z = -0.6056, p-value = 0.2748
alternative hypothesis: true mu is less than 0
```

Das Ergebnis soll zunächst mit dem p-Wert des analogen parametrischen t-Tests verglichen werden (vgl. Abschn. 7.2.3). Bei der anschließenden manuellen Kontrolle dient die mittlere paarweise Differenz zwischen den Werten der zwei abhängigen Bedingungen als Teststatistik. Dafür ist separat für jede Person die Zuordnung ihrer beiden Werte zu einem Testzeitpunkt zu permutieren. Dies ist äquivalent zur Permutation des Vorzeichens der personenweisen Messwertdifferenz. Bei n Personen führt dies zu 2^n verschiedenen Gesamt-Permutationen.

```
> t.test(DVpp ~ IV, alternative="less", paired=TRUE)$p.value
[1] 0.2839126

# alle 2^N Möglichk., pro Person Vorzeichen der Differenz zu permutieren
> DVd    <- DVpre - DVpost        # personenweise Messwertdifferenzen
> ordLst <- lapply(numeric(N), function(x) { c(-1, 1) } )
> ordMat <- data.matrix(expand.grid(ordLst))    # alle 2^N Perm.

# für Gesamt-Permutation x der Vorzeichen: mittlere personenweise Diff.
> getMD  <- function(x) { mean(abs(DVd) * x) }
> MDstar <- apply(ordMat, 1, getMD)     # mittlere Differenzen alle Perm.
> MDbase <- mean(DVd)                   # mittl. Differenz Basis-Stichprobe
```

Der für den p-Wert notwendige Vergleich $\hat{\theta}^\star \leq \hat{\theta}$ wird hier in zwei Vergleiche aufgeteilt, um robuster gegenüber Problemen der numerischen Repräsentation von Gleitkommazahlen zu sein (vgl. Abschn. 1.3.6). Dafür wird nur auf ungefähre, nicht auf exakte Gleichheit von $\hat{\theta}^\star$ und $\hat{\theta}$ geprüft.

```
> tol <- .Machine$double.eps^0.5        # numerische Toleranz

# MDstar kleiner Basis-MD ODER MDstar ungefähr gleich Basis-MD?
> MDsIsLEQ <- (MDstar < MDbase) | (abs(MDstar - MDbase) < tol)

# p-Wert: Anteil der mind. so extremen personenweisen Differenzen
> (pVal <- sum(MDsIsLEQ) / length(MDstar))
[1] 0.2800293
```

11.3.3 Test auf Unabhängigkeit von zwei Variablen

Für den Test auf Unabhängigkeit zweier an denselben Personen erhobener Variablen sollen hier dichotome Daten dienen, um das Ergebnis der manuellen Umsetzung mit jenem von Fishers exaktem Test vergleichen zu können – dem passenden Permutationstest (vgl. Abschn. 10.2.4).

```
> Nf  <- 8                                      # Anzahl Personen
> DV1 <- rbinom(Nf, size=1, prob=0.5)           # Daten AV 1
> DV2 <- rbinom(Nf, size=1, prob=0.5)           # Daten AV 2

# p-Wert rechtsseitiger Test
> fisher.test(DV1, DV2, alternative="greater")$p.value
[1] 0.7857143
```

Die manuelle Kontrolle verwendet `Permn(⟨Vektor⟩)` aus dem Paket `DescTools`, um die Zuordnung von Werten der zweiten AV zu jenen der ersten zu permutieren. Die Funktion erzeugt eine Matrix mit allen $n!$ vielen Permutationen des übergebenen Vektors in den Zeilen. Mit den Indizes für den Vektor der zweiten AV liefern diese Permutationen die gewünschten Zuordnungen und führen so zu allen möglichen Kontingenztafeln der gemeinsamen Häufigkeiten mit denselben Randhäufigkeiten wie in der Basisstichprobe. Teststatistik ist die Anzahl der in der Diagonale einer Kontingenztafel stehenden Fälle, also der Übereinstimmungen beider Variablen.

```
# generiere alle Nf! vielen Zuordnungen von AV 1 zu AV 2
> library(DescTools)                          # für Permn()

# Matrix aller Permutationen der Indizes 1:Nf
> permIdx <- Permn(seq(length.out=Nf))

# Anzahl der Übereinstimmungen für gegebene Permutation x der AV 2
> getAgree <- function(x) { sum(diag(table(DV1, DV2[x]))) }

# Anzahl der Übereinstimmungen für jede der Nf! Permutationen
> resAgree <- apply(permIdx, 1, getAgree)
```

```
> agree12  <- sum(diag(table(DV1, DV2)))      # beobachtete Übereinst.

# p-Wert: Anteil der Perm. mit mind. so großer Übereinstimmung
> (pVal <- sum(resAgree >= agree12) / length(resAgree))
[1] 0.7857143
```