Das Bootstrapverfahren unter kleinen Simulationsumfängen

Master-Arbeit

Thorsten Dickhaus

Mathematisches Institut
der
Heinrich-Heine-Universität Düsseldorf

Düsseldorf im März 2005
Betreuung: Prof. Dr. A. Janssen

Inhaltsverzeichnis

Verzeichnis der Abkürzungen und Symbole

ε_a	Einpunktmaß im Punkt a
F_X	Verteilungsfunktion der Zufallsvariablen X
$=^d$	Gleichhheit in Verteilung
$i.i.d.$	independent and identically distributed
$\mathbf{1}_M$	Indikatorfunktion der Menge M
$\lambda\!\!\lambda$	Lebesgue-Maß
$\mathcal{N}(\mu, \sigma^2)$	Normalverteilung mit den Parametern μ und σ^2
o. ä.	oder ähnliche
Φ	Verteilungsfunktion der $\mathcal{N}(0,1)$-Verteilung
$\mathcal{L}(X)$	Verteilung(sgesetz) der Zufallsvariablen X
vgl.	vergleiche

Kapitel 1

Einleitung und Problemstellung

1.1 Das Bootstrapverfahren in der Statistik

Ein Hauptproblem der statistischen Testtheorie ist das Testen des Erwartungswertes von Zufallsgrößen, die z.B. mit einer erhobenen Stichprobe im experimentativen Umfeld identifiziert werden. In dieser Situation liegen also n Zufallsvariablen $X_1, \ldots, X_n$ vor, wobei die X_i im einfachsten Fall i.i.d. verteilt sind. Das statistische Testproblem lautet nun häufig

$$H_0 : E(X_1) = 0 \text{ gegen } H_1 : E(X_1) > 0.$$

Dieses Testproblem ergibt sich zum Beispiel beim Testen der Wirksamkeit eines neuen Medikamentes im Vergleich mit einem bereits etablierten Produkt zum Zwecke der Zulassung des neuen Präparates.
Als Teststatistik für dieses Problem findet das arithmetische Mittel $T_n = \frac{1}{n} \sum_{i=1}^{n} X_i$ Verwendung; diese Teststatistik ist suffizient und vollständig für das zu Grunde liegende Testproblem.
Will man nun einen Niveau α-Test

$$\varphi_n = \begin{cases} 1 & > \\ T_n & c_n \\ 0 & \leq \end{cases}$$

konstruieren, stellt sich das Problem, den richtigen kritischen Wert c_n zu ermitteln. Lässt sich für die zur Modellierung herangezogenen Zufallsgrößen die Normalverteilungsannahme rechtfertigen, so ist dieses Problem bereits gelöst und das Ergebnis ist der sogenannte *t-Test*, bei welchem die kritischen Werte als die Quantile der *t-Verteilung* mit $(n-1)$ Freiheitsgraden gewählt werden. Ist die Normalverteilungsannahme jedoch nicht gerechtfertigt und ist insbesondere keine Information über die Verteilung von $X_1, \ldots, X_n$ verfügbar, so gibt es keine Theorie für die exakte

Bestimmung von c_n. Der *t-Test* ist in Fällen, in denen die X_i nicht normalverteilt sind nicht zu empfehlen, da er das Niveau α schlecht einhält.
Eine erste Möglichkeit, auch in diesem Fall einen Test anzugeben, stammt aus dem *Zentralen Grenzwertsatz.* Dieser besagt, dass

$$\mathcal{L}(\frac{\bar{X}_n - \mu}{\frac{\sigma}{\sqrt{n}}}) \to \mathcal{N}(0,1),\ n \to \infty,$$

wobei folgende Bezeichnungen verwendet werden:

$\bar{X}_n$	Arithmetisches Mittel der Größen $X_1, \ldots, X_n$
μ	Erwartungswert der Größen $X_1, \ldots, X_n$
σ^2	Varianz der Größen $X_1, \ldots, X_n$

Hieraus lässt sich ein asymptotischer Niveau α-Test für das obige Testproblem konstruieren:

$$\varphi_n^{as} = \begin{cases} 1 & > \\ \frac{\sqrt{n} \cdot \bar{X}_n}{V_n^{\frac{1}{2}}} & \Phi^{-1}(1-\alpha)\ . \\ 0 & \leq \end{cases}$$

Hierbei wird für die (zumeist unbekannte) Varianz σ^2 der Schätzer $V_n = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X}_n)^2$ eingesetzt. Allerdings ist bei diesem Vorgehen die Approximationsgüte für kleine Stichprobenumfänge n nicht hinreichend gut.

Eine Lösungsmöglichkeit der angedeuteten Problematik stellt der sogenannte **bootstrap**, eine Resamplingmethode, dar.
Seien im Einstichprobenproblem $X_1, \ldots, X_n : \Omega \to \Omega'$ $i.i.d. \sim \mathbb{P}^{X_1}$ Zufallsvariablen und

$$\begin{aligned} T : \{Q : Q \text{ Verteilung auf } \Omega'\} &\to \mathbb{R} \\ Q &\mapsto T(Q) \end{aligned}$$

ein interessierendes Funktional vom Bildraum Ω' in die reellen Zahlen. Ein Schätzer für das Wahrscheinlichkeitsmaß $\mathbb{P}^{X_1}$ ist dann das empirische Maß $\hat{P}_n = \frac{1}{n} \sum_{i=1}^n \varepsilon_{X_i}$ (Gleichverteilung auf den Daten). Daraus lässt sich ein Schätzer $T(\hat{P}_n)$ für das Funktional $T(\mathbb{P}^{X_1})$ gewinnen, der im Allgemeinen nicht erwartungstreu ist. Gesucht ist deshalb die Verteilung

$$P(T(\hat{P}_n) - T(\mathbb{P}^{X_1}) \leq t), \quad t \in \mathbb{R} \tag{1.1}$$

des Fehlers.
Die **bootstrap Idee** besteht nun darin, den ursprünglichen Wahrscheinlichkeitsraum $(\Omega'^n, \mathcal{A}'^n, P^{(X_1,\ldots,X_n)})$ durch eine empirische Version $(\Omega'^n, \mathcal{A}'^n, (\hat{P}_n)^n)$ zu ersetzen.
Dazu generiert man eine **bootstrap Stichprobe** $X_1^*, \ldots, X_n^* : (\Omega^*, \mathcal{A}^*, \mathbb{P}^*) \to (\Omega', \mathcal{A}')$, für die gilt:

$$\mathbb{P}^{*X_1^*|(X_1,\ldots,X_n)} = \hat{P}_n.$$

Auf Grund der Definition von $\hat{P}_n$ ist unmittelbar klar, dass das Ziehen der bootstrap Stichprobe dem Ziehen mit Zurücklegen von n Größen aus der Ausgangsstichprobe entspricht.
Man berechnet dann den Ausdruck (1.1) in dem bootstrap Modell, bestimmt also

$$P(T(\hat{P}_n^*) - T(\hat{P}_n) \leq t), \quad t \in \mathbb{R}. \tag{1.2}$$

Dieser Ausdruck ist der bootstrap Schätzer für(1.1); der Ausdruck (1.2) ist berechenbar.

Zentrales Hilfsmittel für den Nachweis der Konsistenz solcher bootstrap Verfahren ist der folgende Satz von Berry-Esséen:

Satz 1.1 (Satz von Berry-Esséen)
$(X_i)_{i\in\mathbb{N}}$ *unabhängig* , $0 < Var(X_i) < \infty \ \forall i$.
F_n *sei die Verteilungsfunktion der standardisierten Summe*

$$\frac{\sum_{i=1}^n (X_i - E(X_i))}{\sqrt{\sum_{i=1}^n Var(X_i)}}.$$

Dann gilt:

$$\sup_{x\in\mathbb{R}} |F_n(x) - \Phi(x)| \leq \frac{6}{s_n^3} \cdot \sum_{i=1}^n E(|X_i|^3),$$

wobei Φ *die Verteilungsfunktion der* $\mathcal{N}(0,1)$*-Verteilung bezeichnet und* $s_n^2 = \sum_{i=1}^n Var(X_i)$ *gilt.*

Liegen i.i.d. Variable X_i *vor, so ergibt sich damit die folgende Abschätzung:*

$$\sup_{x\in\mathbb{R}} |F_n(x) - \Phi(x)| \leq \frac{6}{\sqrt{n} \cdot Var(X_1)^{\frac{3}{2}}} \cdot E(|X_1|^3) \sim \frac{1}{\sqrt{n}}.$$

Beweis siehe Gänssler / Stute(1977). ■
Dieser Satz wird in den folgenden Kapiteln benutzt werden, um die Verteilungskonvergenz der Variablen aus der Originalstichprobe und denen aus der bootstrap Stichprobe nachzuweisen. Dazu ist die Kenntnis der (bedingten) Momente der bootstrap Größen erforderlich, die im folgenden Satz angegeben werden.

Satz 1.2 (Bedingte Momente von bootstrap Größen)
Es sei $\vec{X}_n = (X_1, \ldots, X_n)$ der Vektor der i.i.d. Originaldaten.
Dann gilt bedingt unter $\vec{X}_n$:

$$E(X_1^*|\vec{X}_n) = \frac{1}{n}\sum_{i=1}^{n} X_i =: \bar{X}_n \tag{1.3}$$

$$E(\frac{1}{m(n)}\sum_{i=1}^{m(n)} X_i^*|\vec{X}_n) = \frac{1}{n}\sum_{i=1}^{n} X_i = \bar{X}_n \tag{1.4}$$

$$E(X_1^{*2}|\vec{X}_n) = \frac{1}{n}\sum_{i=1}^{n} X_i{}^2 \tag{1.5}$$

$$Var(X_1^*|\vec{X}_n) = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X}_n)^2 \tag{1.6}$$

$$Var(\frac{1}{m(n)}\sum_{i=1}^{m(n)} X_i^*|\vec{X}_n) = \frac{1}{n \cdot m(n)}\sum_{i=1}^{n}(X_i - \bar{X}_n)^2 \tag{1.7}$$

$$E(X_1^{*3}|\vec{X}_n) = \frac{1}{n}\sum_{i=1}^{n} X_i{}^3 \tag{1.8}$$

Beweis
zu Gleichung (1.3):
Die Zufallsvariable X_1^* ist für feststehende Werte der Originaldaten diskret verteilt mit den möglichen Werten $X_1, \ldots, X_n$, welche von X_1^* jeweils mit der gleichen Wahrscheinlichkeit $\frac{1}{n}$ angenommen werden. Demnach gilt: $E(X_1^*|\vec{X}_n) = \sum_{i=1}^{n} X_i \cdot P(X_1^* = X_i) = \frac{1}{n}\sum_{i=1}^{n} X_i$.

zu den Gleichungen (1.5) und (1.8):
Diese Gleichungen folgen in Analogie zum voranstehenden Beweis.

zu Gleichung (1.4):
Wegen der Linearität der bedingten Erwartung und der i.i.d. Eigenschaft der Variablen X_i^* gilt: $E(\frac{1}{m(n)}\sum_{i=1}^{m(n)} X_i^*|\vec{X}_n) = \frac{1}{m(n)} \cdot m(n) \cdot E(X_1^*|\vec{X}_n) = E(X_1^*|\vec{X}_n) = \bar{X}_n$.

zu Gleichung (1.6):
Zunächst gilt auf Grund elementarer Rechenregeln für die bedinge Varianz: $Var(X_1^*|\vec{X}_n) = E(X_1^{*2}|\vec{X}_n) - (E(X_1^*|\vec{X}_n))^2 = \frac{1}{n}\sum_{i=1}^{n} X_i{}^2 - \frac{1}{n^2}(\sum_{i=1}^{n} X_i)^2$. Dies kann mit Hilfe einiger Um-

formungen auf die angegebene Form gebracht werden:

$$\begin{aligned}
Var(X_1^*|\vec{X}_n) &= \frac{1}{n}\sum_{i=1}^{n} {X_i}^2 - \frac{1}{n^2}(\sum_{i=1}^{n} X_i)^2 \\
&= \frac{1}{n}\left(\sum_{i=1}^{n} {X_i}^2 - \frac{1}{n}(\sum_{i=1}^{n} X_i)^2\right) \\
&= \frac{1}{n}\left(\sum_{i=1}^{n} {X_i}^2 - \frac{1}{n}\cdot n^2 \cdot \bar{X}_n^2\right) \\
&= \frac{1}{n}\left(\sum_{i=1}^{n} {X_i}^2 - n\cdot \bar{X}_n^2\right) \\
&= \frac{1}{n}\left(\sum_{i=1}^{n} {X_i}^2 - 2n\cdot \bar{X}_n^2 + n\cdot \bar{X}_n^2\right) \\
&= \frac{1}{n}\left(\sum_{i=1}^{n} {X_i}^2 - 2\bar{X}_n \sum_{i=1}^{n} X_i + n\cdot \bar{X}_n^2\right) \\
&= \frac{1}{n}\sum_{i=1}^{n} ({X_i}^2 - 2X_i\bar{X}_n + \bar{X}_n^2) \\
&= \frac{1}{n}\sum_{i=1}^{n} (X_i - \bar{X}_n)^2.
\end{aligned}$$

zu Gleichung (1.7):
Auf Grund der Unabhängigkeit der Variablen X_i^* ist die bedingte Varianz ihrer Summe gleich der Summe ihrer einzelnen bedingten Varianzen. Diese sind auf Grund der Tatsache, dass die X_i^* identisch verteilt sind, alle gleich $Var(X_1^*|\vec{X}_n)$. Diese Überlegungen liefern unmittelbar das Resultat. ■

1.2 Der $m(n)$ out of n bootstrap

Die Idee beim sogenannten $m(n)$ *out of* n *bootstrap* besteht darin, den bootstrap Stichprobenumfang $m(n)$ nicht zwingend gleich dem Umfang n der Originalstichprobe zu wählen. Von besonderem praktischen Interesse ist hierbei natürlich die Vorgehensweise der *low resampling intensity*, also der Fall, in dem $\frac{m(n)}{n} \to 0$ gilt.

Mit Hilfe des im vorherigen Abschnitt angegebenen Satzes von Berry-Esséen lässt sich in einigen Situationen ein bedingter Zentraler Grenzwertsatz für den $m(n)$ out of n bootstrap beweisen.

Genauer gilt:

Satz 1.3 (Asymptotische Normalität des bootstrap Mittels)
Die Bedingung

$$(\frac{n}{m(n)})^{\frac{1}{2}} \cdot \max_{i \leq n} \frac{|X_i - \bar{X}_n|}{(\sum_{j=1}^{n}(X_j - \bar{X}_n)^2)^{\frac{1}{2}}} \to 0 \ \ \mathbb{P} - \textit{stochastisch}$$

impliziert die bedingte Konvergenz unter $\vec{X}_n := (X_1, \ldots, X_n)$

$$\sup_{x \in \mathbb{R}} \left| \mathbb{P}^* \left(\frac{\sum_{i=1}^{m(n)} X_i^* - \frac{m(n)}{n} \cdot \sum_{i=1}^{n} X_i}{(\frac{m(n)}{n} \cdot \sum_{i=1}^{n}(X_i - \bar{X}_n)^2)^{\frac{1}{2}}} \leq x | \vec{X}_n \right) - \Phi(x) \right| \to 0 \ \ \mathbb{P} - \textit{stochastisch},$$

falls $\mathbb{P}(\{\sum_{i=1}^{n}(X_i - \bar{X}_n)^2 > 0\}) \to 1$.

Beweis:
Wir schätzen das angegebene Supremum mit Hilfe des Satzes von Berry-Esséen unter Verwendung der in Satz 1.2 berechneten bedingten Momente der bootstrap Größen ab. Es ergibt sich, da die Variablen X_i^* i.i.d. sind:

$$\sup_{x \in \mathbb{R}} \left| \mathbb{P}^* \left(\frac{\sum_{i=1}^{m(n)} X_i^* - \frac{m(n)}{n} \cdot \sum_{i=1}^{n} X_i}{(\frac{m(n)}{n} \cdot \sum_{i=1}^{n}(X_i - \bar{X}_n)^2)^{\frac{1}{2}}} \leq x | \vec{X}_n \right) - \Phi(x) \right| \leq \frac{6}{m(n)^{\frac{1}{2}}} \cdot \frac{n^{-1} \sum_{i=1}^{n} |X_i - \bar{X}_n|^3}{(n^{-1} \sum_{i=1}^{n}(X_i - \bar{X}_n)^2)^{\frac{3}{2}}}.$$

Die Summe $\sum_{i=1}^{n} |X_i - \bar{X}_n|^3$ im Zähler kann durch den Ausdruck
$\max_{j \leq n} |X_j - \bar{X}_n| \sum_{i=1}^{n}(X_i - \bar{X}_n)^2$ abgeschätzt werden. Damit ergibt sich nach Kürzen und Zusammenfassen:

$$\sup_{x \in \mathbb{R}} \left| \mathbb{P}^* \left(\frac{\sum_{i=1}^{m(n)} X_i^* - \frac{m(n)}{n} \cdot \sum_{i=1}^{n} X_i}{(\frac{m(n)}{n} \cdot \sum_{i=1}^{n}(X_i - \bar{X}_n)^2)^{\frac{1}{2}}} \leq x | \vec{X}_n \right) - \Phi(x) \right| \leq 6 \left(\frac{n}{m(n)} \right)^{\frac{1}{2}} \cdot \frac{\max_{j \leq n} |X_j - \bar{X}_n|}{(\sum_{i=1}^{n}(X_i - \bar{X}_n)^2)^{\frac{1}{2}}}$$

und unter der angegebenen Bedingung folgt damit die Behauptung. ■

Es stellt sich nun unmittelbar die Frage, inwiefern man dieses Ergebnis für den Anwendungsfall des $m(n)$ out of n bootstrap mit low resampling intensity verwenden kann, da hier ja wie oben ausgeführt $\frac{m(n)}{n} \to 0$ gilt und die Voraussetzung für Satz 1.3 damit unerfüllbar scheint. Unter gewissen Zusatzvoraussetzungen an die Variablen $X_1, \ldots, X_n$ lässt sich die Bedingung $(\frac{n}{m(n)})^{\frac{1}{2}} \cdot \max_{i \leq n} \frac{|X_i - \bar{X}_n|}{(\sum_{j=1}^{n}(X_j - \bar{X}_n)^2)^{\frac{1}{2}}} \to 0 \ \ \mathbb{P} -$ stochastisch jedoch auch hier erfüllen.

Dazu zwei Beispiele:

Beispiel 1.4 (Beschränkte Original-Zufallsvariablen)
Es seien $X_1, \ldots, X_n$ i.i.d. mit $|X_i| \leq K \ \ \forall i$ und $0 < \sigma^2 := Var(X_1) < \infty$. Dann besagt das Starke Gesetz der großen Zahlen, dass $(\frac{1}{n}\sum_{j=1}^{n}(X_j - \bar{X}_n)^2)^{\frac{1}{2}}$ gegen σ konvergent ist. Damit reicht unter diesen Voraussetzungen für die Gültigkeit des bedingten Zentralen Grenzwertsatzes für das $m(n)$ out of n bootstrap Mittels die Bedingung

$$m(n) \to \infty$$

aus, denn

$$(\frac{n}{m(n)})^{\frac{1}{2}} \cdot \max_{i \leq n} \frac{|X_i - \bar{X}_n|}{(\sum_{j=1}^{n}(X_j - \bar{X}_n)^2)^{\frac{1}{2}}} \leq m(n)^{-\frac{1}{2}} \cdot \frac{K}{(\frac{1}{n}\sum_{j=1}^{n}(X_j - \bar{X}_n)^2)^{\frac{1}{2}}}$$

und

$$\frac{K}{(\frac{1}{n}\sum_{j=1}^{n}(X_j - \bar{X}_n)^2)^{\frac{1}{2}}} \to \frac{K}{\sigma} < \infty.$$

Beispiel 1.5 (Existenz höherer Momente)
Sei $\gamma > 2$ und seien die Zufallsvariablen $X_1, \ldots, X_n$ i.i.d. mit $E(|X_1|^\gamma) < \infty$. Dann garantiert die Bedingung

$$\lim_{n \to \infty} \frac{n}{m(n)^{\frac{\gamma}{2}}} < \infty$$

die Gültigkeit des bedingten Zentralen Grenzwertsatzes für das $m(n)$-bootstrap Mittel.

Zum Nachweis dieser Aussage sei $(b_n)_{n \in \mathbb{N}}$ eine Folge mit $\lim_{n \to \infty} b_n = \infty$. Dann gilt einerseits

$$\int_{|X_1| \geq b_n} |X_1|^\gamma d\mathbb{P} \to 0$$

und zum anderen

$$\begin{aligned} \int_{|X_1| \geq b_n} |X_1|^\gamma d\mathbb{P} &\geq \int b_n^\gamma \mathbf{1}_{[b_n, \infty)}(|X_1|) d\mathbb{P} \\ &= b_n^\gamma \cdot \mathbb{P}(|X_1| \geq b_n). \end{aligned}$$

Also gilt in dieser Situation: $b_n^\gamma \cdot \mathbb{P}(|X_1| \geq b_n) \to 0$. Des Weiteren ist

$$\{\max_{i \leq n} |X_i| \geq b_n\} \subseteq \bigcup_{i=1}^{n} \{|X_i| \geq b_n\}$$

und daher folgt wegen der Monotonie und der σ-Subadditivität des Wahrscheinlichkeitsmaßes $\mathbb{P}$:

$$\begin{aligned}\mathbb{P}(\max_{i\leq n}|X_i|\geq b_n) &\leq \sum_{i=1}^{n}\mathbb{P}(|X_i|\geq b_n)\\ &= n\cdot\mathbb{P}(|X_1|\geq b_n)\\ &= \frac{n}{b_n^\gamma}\cdot b_n^\gamma\cdot\mathbb{P}(|X_1|\geq b_n).\end{aligned}$$

Nach diesen Vorüberlegungen ergibt sich bei der Überprüfung der Bedingung für die Gültigkeit des bedingten Zentralen Grenzwertsatzes:

$$\begin{aligned}&\mathbb{P}((\frac{n}{m(n)})^{\frac{1}{2}}\cdot\max_{i\leq n}\frac{|X_i-\bar{X_n}|}{(\sum_{j=1}^{n}(X_j-\bar{X}_n)^2)^{\frac{1}{2}}}\geq\varepsilon)\\ =\ &\mathbb{P}((\frac{n}{m(n)})^{\frac{1}{2}}\cdot\frac{n^{-\frac{1}{2}}\max_{i\leq n}|X_i-\bar{X_n}|}{(\frac{1}{n}\sum_{j=1}^{n}(X_j-\bar{X}_n)^2)^{\frac{1}{2}}}\geq\varepsilon)\\ =\ &\mathbb{P}(m(n)^{-\frac{1}{2}}\cdot\frac{\max_{i\leq n}|X_i-\bar{X_n}|}{(\frac{1}{n}\sum_{j=1}^{n}(X_j-\bar{X}_n)^2)^{\frac{1}{2}}}\geq\varepsilon).\end{aligned}$$

Betrachten wir nun die Folge $b_n := \sqrt{m(n)}\cdot\varepsilon$, so dass also $\varepsilon = \frac{b_n}{\sqrt{m(n)}}$, so erhalten wir mit den Überlegungen von oben:

$$\begin{aligned}&\mathbb{P}(m(n)^{-\frac{1}{2}}\cdot\frac{\max_{i\leq n}|X_i-\bar{X_n}|}{(\frac{1}{n}\sum_{j=1}^{n}(X_j-\bar{X}_n)^2)^{\frac{1}{2}}}\geq\frac{b_n}{\sqrt{m(n)}})\\ =\ &\mathbb{P}(\frac{\max_{i\leq n}|X_i-\bar{X_n}|}{(\frac{1}{n}\sum_{j=1}^{n}(X_j-\bar{X}_n)^2)^{\frac{1}{2}}}\geq b_n)\\ \leq\ &\frac{n}{b_n^\gamma}\cdot b_n^\gamma\cdot\mathbb{P}(\frac{|X_1-\bar{X_n}|}{(\frac{1}{n}\sum_{j=1}^{n}(X_j-\bar{X}_n)^2)^{\frac{1}{2}}}\geq b_n)\\ =\ &\frac{n}{m(n)^{\frac{\gamma}{2}}\cdot\varepsilon^\gamma}\cdot b_n^\gamma\cdot\mathbb{P}(\frac{|X_1-\bar{X_n}|}{(\frac{1}{n}\sum_{j=1}^{n}(X_j-\bar{X}_n)^2)^{\frac{1}{2}}}\geq b_n).\end{aligned}$$

Da $b_n^\gamma\cdot\mathbb{P}(\frac{|X_1-\bar{X_n}|}{(\frac{1}{n}\sum_{j=1}^{n}(X_j-\bar{X}_n)^2)^{\frac{1}{2}}}\geq b_n)$ nach den obigen Überlegungen gegen 0 konvergiert, reicht hier also für die Gültigkeit des bedingten Zentralen Grenzwertsatzes aus, dass der Quotient $n\cdot m(n)^{-\frac{\gamma}{2}}$ für $n\to\infty$ beschränkt bleibt.

1.3 Simulierte Fallstudien

Im praktischen Teil dieser Arbeit soll der vorgestellte $m(n)$ out of n bootstrap auf seine Tauglichkeit in konkreten Anwendungssituationen getestet werden. Zu diesem Zweck werden Daten

gemäß bekannter Verteilungen mit Hilfe von Computersimulationen generiert und dann Testprobleme an diesen Daten als bootstrap Tests durchgeführt. Es ist dabei von Interesse, wie „gut“ der $m(n)$ out of n bootstrap im Vergleich mit anderen Testmethoden abschneidet. Als Maßzahl dafür dient der beobachtete Fehler erster Art bei mehrfacher Durchführung der jeweiligen Testprozedur. Um eine Vergleichbarkeit mit anderen Verfahren zu gewährleisten, lehnen sich die durchgeführten Monte-Carlo Simulationen an das Simulationsschema von Janssen und Pauls (2005) an. Die erzeugten Daten besitzen die folgenden Verteilungsdichten:

Normalverteilung:	$f(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x^2)$
Doppel-Exponentialverteilung:	$f(x) = \frac{1}{2} \exp(-\lvert x \rvert)$
Exponentialverteilung:	$f(x) = \exp(-(x+1)),\ x \geq -1$
Gleichverteilung:	$f(x) = \frac{1}{2\sqrt{3}} \mathbf{1}_{[-\sqrt{3},\sqrt{3}]}(x)$
Cauchyverteilung:	$f(x) = \frac{1}{\pi} \frac{1}{1+x^2}$

Als konkrete Testszenarien werden das Einstichprobenproblem mit und ohne Studentisierung sowie das Zweistichprobenproblem bei unterschiedlichen Varianzen in den Stichproben, also das sogenannte *Behrens-Fisher-Problem*, behandelt.

1.3.1 Der Einstichprobenfall

Gegeben sei eine Stichprobe $X_1, \ldots, X_n$ vom Umfang n, wobei die X_i i.i.d. mit $\mu := E(X_1) = 0$ seien. Wir betrachten das einseitige Testproblem $H = \{\mu = 0\}$ gegen $K = \{\mu > 0\}$ zum Niveau α. Der $m(n)$ out of n bootstrap arbeitet hier derart, dass B-mal jeweils $m(n)$ der n Originalwerte zufällig gezogen und zu einer bootstrap Stichprobe zusammengefasst werden. Dann wird jeweils der bootstrap Schätzer für die Teststatistik für diese bootstrap Stichprobe bestimmt und nach Abschluss der B Durchführungen wird das empirische $(1-\alpha)$-Quantil c_α der bootstrap-Schätzungen ermittelt und als kritischer Wert für die Teststatistik an den Originaldaten verwendet. Der beobachtete Fehler 1. Art bei M Monte-Carlo Simulationen dieser Art ist dann $M^{-1} \cdot \#\{\text{Original-Teststatistik} > c_\alpha\}$.

Die genaue Form der Teststatistik sowie die Dokumentation der simulierten Tests finden sich in Kapitel 2.

1.3.2 Der Zweistichprobenfall

Hier liegen zwei unabhängige Stichproben $X_1, \ldots, X_{n_1}$ vom Umfang n_1 sowie $Y_1, \ldots, Y_{n_2}$ vom Umfang n_2 vor. Die X_i bzw. Y_j seien jeweils i.i.d. mit $\mu := E(X_1)$ und $\nu := E(Y_1)$, wobei $\mu = \nu$

gelte. Die Varianzen $\sigma_X^2 := Var(X_1)$ und $\sigma_Y^2 := Var(Y_1)$ werden als ungleich und unbekannt angenommen, denn im Falle gleicher Varianzen ist aus der stochastischen Theorie (vgl. u.a. Janssen (1998)) bekannt, dass der Pitman'sche Permutationstest das optimale Testverfahren darstellt, welches auch leicht implementiert werden kann.

Hier wird nun das Testproblem $H = \{\mu = \nu\}$ gegen $K = \{\mu > \nu\}$ zum Niveau α betrachtet. Wieder kommt der $m(n)$ out of n bootstrap zum Einsatz. Es werden eine bootstrap Stichprobe vom Umfang m_1 aus den n_1 Ausgangswerten für die X-Werte und eine bootstrap Stichprobe vom Umfang m_2 aus den n_2 Ausgangswerten für die Y-Werte gezogen und dann in Analogie zum erläuterten Verfahren beim Einstichprobenproblem bootstrap Schätzungen für die Teststatistik dazu verwendet, kritische Werte zu ermitteln und Fehler erster Art zu beobachten.

Eine detaillierte Darstellung der verwendeten Teststatistiken und der durchgeführten Simulationen für den Zweistichprobenfall liefert das Kapitel 3.

Kapitel 2

Das Einstichprobenproblem

2.1 Theorie und Teststatistik

Zur Schätzung der Verteilung des Mittels $\bar{X}_n$ unter der Voraussetzung $E(\bar{X}_n) = 0$ wird das $m(n)$-bootstrap Mittel

$$\frac{1}{m(n)} \sum_{i=1}^{m(n)} X_i^* - \bar{X}_n = \frac{1}{m(n)} \sum_{i=1}^{m(n)} (X_i^* - \bar{X}_n)$$

herangezogen.

Der Zentrale Grenzwertsatz und unser Satz 1.3 liefern unter der Bedingung von i.i.d. Zufallsvariablen $X_1, \ldots, X_n$ mit $E(X_1) = 0$:

$$\mathcal{L}(\frac{\sqrt{n} \cdot \bar{X}_n}{V_n^{\frac{1}{2}}}) \quad \underset{(n\to\infty)}{\to} \quad \mathcal{N}(0,1) \text{ sowie}$$

$$\mathcal{L}(\frac{\sqrt{m} \cdot (\bar{X}^*_{m(n)} - \bar{X}_n)}{\sqrt{\frac{n-1}{n} V_n}}) \quad \underset{(n\to\infty)}{\to} \quad \mathcal{N}(0,1),$$

wobei V_n die empirische Varianz von $X_1, \ldots, X_n$ bezeichnet, also

$$V_n = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X}_n)^2$$

gilt.
Hieraus lässt sich ein bootstrap Test für das Testproblem

$$H : \{E(X_1) = 0\} \quad \text{gegen} \quad K : \{E(X_1) > 0\} \tag{2.1}$$

der folgenden Form herleiten:

$$\varphi^*_{n,I} = \begin{cases} 1 & > \\ \sqrt{n} \cdot \bar{X}_n & \sqrt{\frac{n}{n-1}} \cdot \sqrt{m} \cdot c^*_n(\omega) \\ 0 & \leq \end{cases}$$

Dabei ist $c^*_n(\omega)$ das $(1-\alpha)$-Quantil der bedingten Verteilungsfunktion

$$x \mapsto \mathbb{P}(\bar{X}^*_{m(n)} - \bar{X}_n \leq x | \vec{X}_n).$$

Satz 2.1

$\varphi^*_{n,I}$ ist ein asymptotischer Niveau α-Test für das Testproblem (2.1).

Beweis

Wir schreiben den bootstrap Test um:

$$\varphi^*_{n,I} = \begin{cases} 1 & > \\ \frac{\sqrt{n} \cdot \bar{X}_n}{V_n^{\frac{1}{2}}} & \frac{\sqrt{\frac{n}{n-1}} \cdot \sqrt{m}}{V_n^{\frac{1}{2}}} \cdot c^*_n(\omega) =: \tilde{c}_n(\omega) \\ 0 & \leq \end{cases}$$

$\tilde{c}_n(\omega)$ ist damit das $(1-\alpha)$-Quantil von

$$x \quad \mapsto \quad \mathbb{P}\left(\frac{\sqrt{m} \cdot (\bar{X}^*_{m(n)} - \bar{X}_n)}{\sqrt{\frac{n-1}{n} V_n}} \leq x | \vec{X}_n\right). \tag{2.2}$$

Für (2.2) gilt der Zentrale Grenzwertsatz und daher folgt:

$$\tilde{c}_n(\omega) \to \Phi^{-1}(1-\alpha) \ \ \mathbb{P} - \text{stochastisch.}$$

Ebenso gilt der Zentrale Grenzwertsatz aber auch für die Statistik $\sqrt{n} \cdot \bar{X}_n \cdot V_n^{-\frac{1}{2}}$. ■

Für die Durchführung des Tests $\varphi^*_{n,I}$ lässt sich direkt das Resamplingschema 1.1 ableiten, das wir im folgenden Abschnitt angeben.

Von Peter Hall stammt der Verbesserungsvorschlag, stets studentisierte Versionen der Teststatistik in dem bootstrap Prozess zu verwenden, also den folgenden modifizierten Test zu verwenden:

$$\varphi^*_{n,II} = \begin{cases} 1 & > \\ \frac{\sqrt{n} \cdot \bar{X}_n}{V_n^{\frac{1}{2}}} & \bar{c}^*_n(\omega), \\ 0 & \leq \end{cases}$$

wobei $\bar{c}_n^*(\omega)$ gewählt wird als das $(1-\alpha)$-Quantil der bedingten bootstrap Verteilung

$$x \mapsto \mathbb{P}(\sqrt{m} \cdot \frac{\bar{X}^*_{m(n)} - \bar{X}_n}{(\frac{1}{m-1}\sum_{i=1}^m (X_i^* - \bar{X}^*_{m(n)})^2)^{\frac{1}{2}}} \leq x | \vec{X}_n).$$

Dies führt zu dem Resamplingschema 1.2 des folgenden Abschnitts.

2.2 Resamplingschemata

Für den Einstichprobenfall wird zur Approximation der Verteilung von $\sqrt{n} \cdot \bar{X}_n$ unter der Nullhypothese $H = \{E(X_1) = 0\}$ bei zu Grunde liegenden i.i.d. Zufallsvariablen $X_1, \ldots, X_n$ zunächst folgendes Resamplingschema verwendet, das dem Testverfahren $\varphi^*_{n,I}$ entspricht:

Resamplingschema 1.1:

(A) Erzeuge $X_1, \ldots, X_n$ gemäß der Testverteilung.

(B) Bilde die Teststatistik
$T_n = \sqrt{n} \cdot \bar{X}_n$.

(C) Generiere durch Ziehen mit Zurücklegen B bootstrap Datensätze.

(D) Berechne für $b = 1, \ldots, B$ die bootstrap Teststatistiken
$T_{m(n),b} = \sqrt{m(n)} \cdot \sqrt{\frac{n}{n-1}} \cdot (\bar{X}^*_{m(n),b} - \bar{X}_n)$.

(E) Ermittle den kritischen Wert für den Niveau α-Test als $(1-\alpha) \cdot (B+1)$-te Orderstatistik des Vektors $(T_n, T_{m(n),1}, \ldots, T_{m(n),B})$.

Dem Verbesserungsvorschlag Peter Halls folgend ergibt sich das

Resamplingschema 1.2:

(A) Erzeuge $X_1, \ldots, X_n$ gemäß der Testverteilung.

(B) Bilde die Teststatistik
$T_n = \sqrt{n} \cdot \bar{X}_n \cdot V_n^{-\frac{1}{2}}$.

(C) Generiere durch Ziehen mit Zurücklegen B bootstrap Datensätze.

(D) Berechne für $b = 1, \ldots, B$ die Studentisierungen
$V_{m(n),b} = \frac{1}{m-1} \sum_{i=1}^{m} (X^*_{i,b} - \bar{X}^*_{m(n),b})^2$.

(E) Berechne für $b = 1, \ldots, B$ die bootstrap Teststatistiken
$T_{m(n),b} = \sqrt{m(n)} \cdot \frac{(\bar{X}^*_{m(n),b} - \bar{X}_n)}{\sqrt{V_{m(n),b}}}$.

(F) Ermittle den kritischen Wert für den Niveau α-Test als $(1-\alpha) \cdot (B+1)$-te Orderstatistik des Vektors $(T_n, T_{m(n),1}, \ldots, T_{m(n),B})$.

Dieses Resamplingschema 1.2 implementiert den Test $\varphi^*_{n,II}$.

2.3 Simulation

Die durchgeführten Monte-Carlo Simulationen für den Einstichprobenfall lehnen sich an das Simulationsschema von Janssen und Pauls (2005) an. Betrachtet wird das Testproblem $H = \{\mu = 0\}$ gegen $K = \{\mu > 0\}$. Der Test wird als bootstrap Test gemäß den Resamplingschemata aus Abschnitt 2.2 durchgeführt. Hierbei werden unterschiedliche Werte für den Stichprobenumfang n, den Resampling-Stichprobenumfang $m(n)$, die Anzahl B an bootstrap-Replikationen sowie M der Monte Carlo-Wiederholungen benutzt. Die beobachteten Fehlerwahrscheinlichkeiten 1. Art sind in den folgenden Tabellen zusammengefasst, wobei die ersten beiden Tabellen sich aus das Resamplingschema 1.1 beziehen bzw. die Ergebnisse des simulierten Tests $\varphi^*_{n,I}$ widerspiegeln und der dritten Tabelle das Resamplingschema 1.2 bzw. der Test $\varphi^*_{n,II}$ zu Grunde liegt:

M	B	n	$m(n)$	normal	double-exp.	exponential	uniform	Cauchy
3000	999	32	24	0.0497	0.0517	0.0183	0.0573	0.0197
3000	999	32	16	0.0577	0.0497	0.0233	0.0640	0.0250
3000	999	32	8	0.0583	0.0560	0.0213	0.0683	0.0227
3000	999	24	18	0.0503	0.0510	0.0223	0.0670	0.0333
3000	999	24	12	0.0610	0.0497	0.0237	0.0620	0.0217
3000	999	24	6	0.0627	0.0440	0.0213	0.0637	0.0260
3000	999	16	12	0.0543	0.0577	0.0210	0.0717	0.0293
3000	999	16	8	0.0630	0.0490	0.0177	0.0633	0.0230
3000	999	16	4	0.0620	0.0570	0.0180	0.0643	0.0293
3000	999	8	6	0.0747	0.0630	0.0230	0.0817	0.0407
3000	999	8	4	0.0773	0.0667	0.0220	0.0807	0.0407

Tabelle 2.1: Beobachtete Fehlerwahrscheinlichkeiten 1. Art für $\varphi^*_{n,I}$, M = 3000

M	B	n	$m(n)$	normal	double-exp.	exponential	uniform	Cauchy
250000	999	32	24	0.0555	0.0509	0.0222	0.0566	0.0244
250000	999	32	16	0.0555	0.0513	0.0211	0.0575	0.0220
250000	999	32	8	0.0553	0.0495	0.0188	0.0581	0.0197
250000	999	24	18	0.0564	0.0524	0.0207	0.0588	0.0254
250000	999	24	12	0.0566	0.0525	0.0204	0.0608	0.0242
250000	999	24	6	0.0570	0.0505	0.0182	0.0613	0.0211
250000	999	16	12	0.0598	0.0559	0.0194	0.0641	0.0292
250000	999	16	8	0.0608	0.0535	0.0189	0.0651	0.0265
250000	999	16	4	0.0611	0.0535	0.0172	0.0664	0.0247
250000	999	8	6	0.0720	0.0658	0.0218	0.0783	0.0423
250000	999	8	4	0.0738	0.0629	0.0214	0.0803	0.0377

Tabelle 2.2: Beobachtete Fehlerwahrscheinlichkeiten 1. Art für $\varphi^*_{n,I}$, $M = 250000$

M	B	n	$m(n)$	normal	double-exp.	exponential	uniform	Cauchy
3000	999	32	24	0.0443	0.0603	0.0397	0.0393	0.1257
3000	999	32	16	0.0493	0.0640	0.0483	0.0350	0.1233
3000	999	32	8	0.0357	0.0643	0.0537	0.0230	0.2023
3000	999	24	18	0.0433	0.0620	0.0440	0.0393	0.1430
3000	999	24	12	0.0443	0.0587	0.0467	0.0267	0.1257
3000	999	24	6	0.0320	0.0533	0.0443	0.0127	0.1930
3000	999	16	12	0.0390	0.0673	0.0383	0.0327	0.1273
3000	999	16	8	0.0383	0.0567	0.0353	0.0170	0.1020
3000	999	16	4	0.0177	0.0517	0.0237	0.0060	0.1787
3000	999	8	6	0.0430	0.0600	0.0257	0.0230	0.1153
3000	999	8	4	0.0240	0.0440	0.0187	0.0163	0.0763

Tabelle 2.3: Beobachtete Fehlerwahrscheinlichkeiten 1. Art für $\varphi^*_{n,II}$

2.4 Diskussion der Ergebnisse

Zur Bewertung und Einordnung der obigen Ergebnisse lohnt zunächst ein Vergleich mit den Fehlerwahrscheinlichkeiten, die sich im Falle der Verwendung der Normalapproximation ergeben. Zu diesem Zwecke wurden die Simulationsrechnungen erneut durchgeführt, wobei nun das $(1-\alpha)$-Quantil der $\mathcal{N}(0,1)$-Verteilung als kritischer Wert verwendet wurde. Dies implementiert also die Durchführung des Tests

$$\varphi_n^{as} = \begin{cases} 1 & > \\ \frac{\sqrt{n}\cdot\bar{X}_n}{V_n^{\frac{1}{2}}} & \Phi^{-1}(1-\alpha) \\ 0 & \leq \end{cases}$$

aus Abschnitt 1.1.

Die beobachteten Fehlerwahrscheinlichkeiten 1. Art bei dieser Vorgehensweise finden sich in der folgenden Tabelle:

M	B	n	normal	double-exp.	exponential	uniform	Cauchy
3000	999	32	0.0507	0.0550	0.0287	0.0613	0.0373
3000	999	24	0.0530	0.0550	0.0253	0.0630	0.0437
3000	999	16	0.0623	0.0530	0.0263	0.0653	0.0377
3000	999	8	0.0767	0.0687	0.0220	0.0727	0.0517
250000	999	32	0.0557	0.0554	0.0268	0.0552	0.0357
250000	999	24	0.0574	0.0566	0.0256	0.0572	0.0373
250000	999	16	0.0601	0.0594	0.0238	0.0600	0.0392
250000	999	8	0.0725	0.0693	0.0243	0.0716	0.0489

Tabelle 2.4: Beobachtete Fehlerwahrscheinlichkeiten 1. Art für φ_n^{as}

Es lässt sich erkennen, dass bei wachsenden Stichprobenumfängen der Test φ_n^{as}, der sich einer Normalapproximation bedient, durchschnittlich geringere beobachtete Fehlerwahrscheinlichkeiten erster Art aufweist, während der $m(n)$ out of n bootstrap beim Vorliegen sehr kleiner Stichprobenumfänge bessere Ergebnisse liefert. Dies geht konform mit der Intention, für kleines n ein Alternativverfahren bereitzustellen, wenn die Normalapproximation noch nicht hinreichend gut greift.

Generell lässt sich bei allen durchgeführten Verfahren anmerken, dass sie für Stichproben, denen eine symmetrische Verteilung zu Grunde liegt (wie z.B. eine Normalverteilung oder die Doppelexponentialverteilung) wesentlich praktikabler sind als in Fällen, in denen die Stichprobe aus einer Grundgesamtheit stammt, deren Verteilung eine ausgeprägte Schiefe besitzt (hier stechen besonders die Werte im Falle der Exponentialverteilung hervor). Dieses Problem wurde auch schon theoretisch untersucht und so empfehlen zum Beispiel die Autoren in del Barrio, E. und Matrán, C.(2000) in solchen Situationen die Verwendung eines *gewichteten bootstraps*.

Bei den durchgeführten bootstrap Tests fällt zudem die Tendenz auf, dass sie mit abnehmender Resampling-Intensität, also mit fallenden Werten für den Quotienten $\frac{m(n)}{n}$, tendenziell ein liberaleres Verhalten zeigen, also das Niveau α immer schlechter einhalten. Es ist daher für die Praxis sicherlich ratsam, die Resampling-Intensität wenn möglich nicht zu gering zu wählen. Sollte jedoch ein klassischer bootstrap nicht durchführbar sein, so belegen sowohl die theoretischen als auch die praktischen Resultate dieses Kapitels, dass auch der $m(n)$ out of n bootstrap in den hier skizzierten Anwendungsfällen ein brauchbares Testverfahren darstellt.

Kapitel 3

Das Zweistichprobenproblem

3.1 Theorie und Teststatistik

Dem Behrens-Fisher-Problem liegt folgende Ausgangssituation zu Grunde: Gegeben sind zwei unabhängige Stichproben $X_1, \ldots, X_{n_1}$ sowie $Y_1, \ldots, Y_{n_2}$, wobei die X_i i.i.d. verteilt mit Erwartungswert $E(X_1)$ sowie Varianz σ_X^2 und die Y_j ebenfalls i.i.d. mit Erwartungswert $E(Y_1)$ sowie Varianz σ_Y^2 seien. Bei dieser Betrachtung sei die Ungleichheit der Varianzen, also der Fall $\sigma_X^2 \neq \sigma_Y^2$ ausdrücklich erlaubt und der Regelfall. Das nun betrachtete Testproblem lautet $H_0 : \{E(X_1) = E(Y_1)\}$ gegen $H_1 : \{E(X_1) > E(Y_1)\}$ und wurde von W.V. Behrens im Jahre 1929 sowie nachfolgend von R.A. Fisher im Jahre 1936 eingehend untersucht.

In Welch, B.L. (1937) wird die Verwendung der folgenden, auch als Welch-Teststatistik bezeichneten Statistik für dieses Testproblem empfohlen und ausgeführt, dass sie asymptotisch $\mathcal{N}(0,1)$-verteilt ist:

$$T_n = \frac{\bar{X}_{n_1} - \bar{Y}_{n_2}}{\sqrt{\frac{S_X^2}{n_1} + \frac{S_Y^2}{n_2}}},$$

wobei die Größen S_X^2 und S_Y^2 Varianzschätzer der folgenden Form sind:

$$\begin{aligned} S_X^2 &= \frac{1}{n_1 - 1} \sum_{i=1}^{n_1} (X_i - \bar{X}_{n_1})^2 \\ S_Y^2 &= \frac{1}{n_2 - 1} \sum_{i=1}^{n_2} (Y_i - \bar{Y}_{n_2})^2. \end{aligned}$$

Berechnet man eine analoge Teststatistik in der „$m(n)$ out of n bootstrap-Welt“, so hat diese die folgende Gestalt:

$$T_n^* = \frac{(\bar{X}_{m_1}^* - \bar{X}_{n_1}) - (\bar{Y}_{m_2}^* - \bar{Y}_{n_2})}{\sqrt{\frac{{S_X^*}^2}{m_1} + \frac{{S_Y^*}^2}{m_2}}},$$

wobei die auftretenden Größen $\bar{X}^*_{m_1}, \bar{Y}^*_{m_2}, {S^*_X}^2$ und ${S^*_Y}^2$ wie folgt definiert sind:

$$\begin{aligned}
\bar{X}^*_{m_1} &= \frac{1}{m_1}\sum_{i=1}^{m_1} X^*_i \\
\bar{Y}^*_{m_2} &= \frac{1}{m_2}\sum_{i=1}^{m_2} Y^*_i \\
{S^*_X}^2 &= \frac{1}{m_1-1}\sum_{i=1}^{m_1}(X^*_i - \bar{X}^*_{m_1})^2 \\
{S^*_Y}^2 &= \frac{1}{m_2-1}\sum_{i=1}^{m_2}(Y^*_i - \bar{Y}^*_{m_2})^2.
\end{aligned}$$

Die Rechtfertigung für die Verwendung dieser bootstrap Teststatistik T^*_n zur Approximation der Verteilung der Original-Teststatistik T_n in den durchgeführten Simulationsrechnungen liefert der folgende

Satz 3.1 (Asymptotische Normalität der Teststatistik $\mathbf{T^*_n}$)
Es sei $n := n_1 + n_2$ und $X_1, \ldots, X_{n_1}$ sowie $Y_1, \ldots, Y_{n_1}$ jeweils i.i.d. Zufallsvariablen. Ferner seien die folgenden Voraussetzungen erfüllt:

(i) $\lim_{n\to\infty} \frac{m_1}{m_1+m_2} =: \rho_1$ *mit* $0 < \rho_1 < \infty$.

(ii) $\mathbb{P}(\{\sum_{i=1}^{n_1}(X_i - \bar{X}_{n_1})^2 > 0\}) \to 1$

(iii) $\mathbb{P}(\{\sum_{i=1}^{n_2}(Y_i - \bar{Y}_{n_2})^2 > 0\}) \to 1$

(iv) $(\frac{n_1}{m_1})^{\frac{1}{2}} \cdot \max_{i\le n_1} \frac{|X_i - \bar{X}_{n_1}|}{(\sum_{j=1}^{n_1}(X_j - \bar{X}_{n_1})^2)^{\frac{1}{2}}} \to 0 \ \ \mathbb{P}-$ *stochastisch*

(v) $(\frac{n_2}{m_2})^{\frac{1}{2}} \cdot \max_{i\le n_2} \frac{|Y_i - \bar{Y}_{n_2}|}{(\sum_{j=1}^{n_2}(Y_j - \bar{Y}_{n_2})^2)^{\frac{1}{2}}} \to 0 \ \ \mathbb{P}-$ *stochastisch*

Dann gilt bedingt unter $\vec{X}_n = (X_1, \ldots, X_{n_1}, Y_1, \ldots, Y_{n_2})$ ein bedingter Zentraler Grenzwertsatz für die bootstrap Teststatistik T^*_n, also:

$$d(\mathcal{L}(T^*_n|\vec{X}_n), \mathcal{N}(0,1)) \to 0 \ \textit{für} \ n \to \infty$$

in einer geeigneten Norm d für Verteilungsfunktionen.

Beweis:
Zunächst erweitern wir T^*_n artifiziell mit dem Faktor $\sqrt{\frac{m_1 m_2}{m_1+m_2}}$ und bringen die bootstrap Teststatistik damit auf die Form

$$T^*_n = \frac{\sqrt{\frac{m_1 m_2}{m_1+m_2}}((\bar{X}^*_{m_1} - \bar{X}_{n_1}) - (\bar{Y}^*_{m_2} - \bar{Y}_{n_2}))}{\sqrt{\frac{m_1 m_2}{m_1+m_2}}\sqrt{\frac{{S^*_X}^2}{m_1} + \frac{{S^*_Y}^2}{m_2}}}.$$

Wir betrachten sodann Zähler und Nenner von T_n^* getrennt. Wegen der Voraussetzung (i) liefert unser Satz 1.3 mit $\rho_2 := 1 - \rho_1$ unter den genannten Voraussetzungen, dass

$$\mathcal{L}(\sqrt{\frac{m_1 m_2}{m_1 + m_2}}(\bar{X}^*_{m_1} - \bar{X}_{n_1})|\vec{X}_n) \to \mathcal{N}(0, \rho_2 \sigma_X^2)$$

und

$$\mathcal{L}(\sqrt{\frac{m_1 m_2}{m_1 + m_2}}(\bar{Y}^*_{m_2} - \bar{Y}_{n_2})|\vec{X}_n) \to \mathcal{N}(0, \rho_1 \sigma_Y^2)$$

mit $\sigma_X^2 = Var(X_1)$ und $\sigma_Y^2 = Var(Y_1)$.
Die Differenz zweier normalverteilter Zufallsvariablen ist wieder normalverteilt, wobei sich die Erwartungswerte subtrahieren und die Varianzen addieren. Daher ergibt sich als Grenzverteilung für den erweiterten Zähler von T_n^*:

$$\sqrt{\frac{m_1 m_2}{m_1 + m_2}}((\bar{X}^*_{m_1} - \bar{X}_{n_1}) - (\bar{Y}^*_{m_2} - \bar{Y}_{n_2})) \to \mathcal{N}(0, \rho_2 \sigma_X^2 + \rho_1 \sigma_Y^2).$$

Bei der Betrachtung des Nenners beschränken wir unsere Überlegungen auf eine Grenzwertbetrachtung für ${S_X^*}^2$, da die Rechnung für ${S_Y^*}^2$ analog ablaufen würde.
Definiere zunächst $\tilde{X}_i^* := X_i^* - \frac{1}{n_1}\sum_{i=1}^{n_1} X_i$ für $i = 1, \ldots, n_1$. Dann sind die ist $\tilde{X}_i^*$ i.i.d. mit $E(\tilde{X}_1^*|\vec{X}_{n_1}) = 0$ und $Var(\tilde{X}_1^*|\vec{X}_{n_1}) = Var(X_1^*|\vec{X}_{n_1}) = \frac{1}{n_1}\sum_{i=1}^{n_1}(X_i - \bar{X}_{n_1})^2$ sowie $Var(\frac{1}{m_1}\sum_{i=1}^{m_1}\tilde{X}_i^*|\vec{X}_{n_1}) = \frac{1}{m_1 n_1}\sum_{i=1}^{n_1}(X_i - \bar{X}_{n_1})^2$. Unser bedingter Zentraler Grenzwertsatz 1.3 für das $m(n)$ bootstrap Mittel besagt damit, dass

$$\frac{\frac{1}{m_1}\sum_{i=1}^{m_1}\tilde{X}_i^*}{\sqrt{\frac{1}{m_1 n_1}\sum_{i=1}^{n_1}(X_i - \bar{X}_{n_1})^2}} = \frac{\sum_{i=1}^{m_1}\tilde{X}_i^*}{\sqrt{m_1}\cdot\sqrt{\frac{1}{n_1}\sum_{i=1}^{n_1}(X_i - \bar{X}_{n_1})^2}} \to \mathcal{N}(0,1).$$

Die Konvergenz von $\frac{1}{n_1}\sum_{i=1}^{n_1}(X_i - \bar{X}_{n_1})^2$ gegen σ_X^2 impliziert damit, dass

$$\frac{\sum_{i=1}^{m_1}\tilde{X}_i^*}{\sqrt{m_1}} \to \mathcal{N}(0, \sigma_X^2).$$

Nun lässt sich der Satz von Raikov (siehe Janssen, A.(1998), Satz (A.6)) auf die Summe der Quadrate der $\tilde{X}_i^*$ anwenden und man erhält

$$\sum_{i=1}^{m_1}\left(\frac{\tilde{X}_i^*}{\sqrt{m_1}}\right)^2 = \frac{1}{m_1}\sum_{i=1}^{m_1}{\tilde{X}_i^*}^2 \to \sigma_X^2.$$

Damit lässt sich nun die Konvergenz von ${S_X^*}^2$ nachweisen:

$$\begin{aligned}
{S_X^*}^2 &= \frac{1}{m_1-1}\sum_{i=1}^{m_1}(X_i^* - \bar{X}_{m_1}^*)^2 \\
&= \frac{1}{m_1-1}\sum_{i=1}^{m_1}(\tilde{X}_i^* + \bar{X}_{n_1} - \bar{X}_{m_1}^*)^2 \\
&= \frac{1}{m_1-1}\sum_{i=1}^{m_1}(\tilde{X}_i^* - (\bar{X}_{m_1}^* - \bar{X}_{n_1}))^2 \\
&= \frac{1}{m_1-1}\left(\sum_{i=1}^{m_1}{\tilde{X}_i^*}^2 - m_1(\bar{X}_{m_1}^* - \bar{X}_{n_1})^2\right) \\
&= \frac{1}{m_1-1}\sum_{i=1}^{m_1}{\tilde{X}_i^*}^2 - \frac{m_1}{m_1-1}(\bar{X}_{m_1}^* - \bar{X}_{n_1})^2.
\end{aligned}$$

Aus dieser Darstellung erkennt man, dass ${S_X^*}^2 \to \sigma_X^2$, da der erste Summand wie oben gezeigt gegen σ_X^2 konvergiert und der zweite Summand asymptotisch verschwindet. Die analoge Rechnung für ${S_Y^*}^2$ ergibt ${S_Y^*}^2 \to \sigma_Y^2$.

Damit konvergiert der erweiterte Nenner von T_n^* also insgesamt (mit ρ_1 und ρ_2 wie oben) gegen $\sqrt{\rho_2\sigma_X^2 + \rho_1\sigma_Y^2}$ und wir können eine Grenzwertbetrachtung für den Quotienten T_n^* durchführen, die den Beweis vervollständigt. ■

Wir erhalten mit diesem Satz in Analogie zur Vorgehensweise im Einstichprobenfall folgenden bootstrap Test für das Behrens-Fisher-Problem, der ein asymptotischer Niveau α-Test für das angegebene Testproblem ist:

$$\varphi_{n,BFP}^* = \begin{cases} 1 & \\ & T_n = \frac{\bar{X}_{n_1} - \bar{Y}_{n_2}}{\sqrt{\frac{S_X^2}{n_1} + \frac{S_Y^2}{n_2}}} \quad > \quad c_n^*(\omega), \\ 0 & \end{cases}$$

mit dem kritischen Wert $c_n^*(\omega)$, der gewählt wird als das $(1-\alpha)$-Quantil der bedingten bootstrap Verteilung

$$x \mapsto \mathbb{P}(T_n^* \leq x | \vec{X}_n).$$

3.2 Resamplingschema

Nach den theoretischen Vorüberlegungen aus dem vorangegangenen Abschnitt lässt sich das Resamplingschema für das Behrens-Fisher-Problem sofort angeben:

Resamplingschema:

(A) Erzeuge $X_1, \ldots, X_{n_1}$ sowie $Y_1, \ldots, Y_{n_2}$ gemäß der Testverteilungen.

(B) Bilde die Varianzschätzungen
$S_X^2 = \frac{1}{n_1-1}\sum_{i=1}^{n_1}(X_i - \bar{X}_{n_1})^2$ und $S_Y^2 = \frac{1}{n_2-1}\sum_{i=1}^{n_2}(Y_i - \bar{Y}_{n_2})^2$.

(C) Bilde die Teststatistik $T_n = \frac{\bar{X}_{n_1} - \bar{Y}_{n_2}}{\sqrt{\frac{S_X^2}{n_1} + \frac{S_Y^2}{n_2}}}$.

(D) Generiere durch Ziehen mit Zurücklegen B bootstrap Datensätze für die $X-$ und die $Y - Variablen.$

(E) Berechne für $b = 1, \ldots, B$ die bootstrap Statistiken
${S^*_{X,b}}^2 = \frac{1}{m_1-1}\sum_{i=1}^{m_1}(X^*_{i,b} - \bar{X}^*_{m_1,b})^2$,
${S^*_{Y,b}}^2 = \frac{1}{m_2-1}\sum_{i=1}^{m_2}(Y^*_{i,b} - \bar{Y}^*_{m_2,b})^2$ und
$T^*_{n,b} = \frac{(\bar{X}^*_{m_1,b} - \bar{X}_{n_1}) - (\bar{Y}^*_{m_2,b} - \bar{Y}_{n_2})}{\sqrt{\frac{{S^*_{X,b}}^2}{m_1} + \frac{{S^*_{Y,b}}^2}{m_2}}}$

(F) Ermittle den kritischen Wert für den Niveau α-Test als $(1-\alpha)\cdot(B+1)$-te Orderstatistik des Vektors $(T_n, T^*_{n,1}, \ldots, T^*_{n,B})$.

3.3 Simulation

Die folgende Tabelle beinhaltet die Simulationsergebnisse für den durchgeführten $m(n)$ out of n bootstrap Test im Zweistichprobenmodell. Es wurden $M = 3000$ Monte Carlo-Schleifen mit jeweils $B = 999$ bootstrap Datensätzen gerechnet, wobei die simulierten Daten die gleichen Verteilungsdichten wie im Einstichprobenfall (vgl. Kapitel 2) besitzen. Die ungleichen Varianzen wurden dadurch simuliert, dass die Daten nach dem Modell

$$X_i = \mu_1 + \sigma_X \cdot W_i \text{ bzw. } Y_j = \mu_2 + \sigma_Y \cdot Z_j$$

generiert wurden. Dabei sind $\mu_1 := E(X_1)$ und σ_X^2 die Parameter für die erste und $\mu_2 := E(Y_1)$ und σ_Y^2 die Parameter für die zweite Stichprobe. Die Störvariablen W_i und Z_j wurden gemäß der angebenen Verteilungsdichten vom Computer erzeugt.

$m_i/n_i = 0.75,\ i = 1, 2$

$\sigma_1^2 : \sigma_2^2$	1.0:1.0				1.0:1.5				1.0:2.0			
$n_1 : n_2$	8:8	8:16	16:8	16:16	8:8	8:16	16:8	16:16	8:8	8:16	16:8	16:16
$m_1 : m_2$	6:6	6:12	12:6	12:12	6:6	6:12	12:6	12:12	6:6	6:12	12:6	12:12
normal	0.0417	0.0480	0.0407	0.0440	0.0420	0.0473	0.0437	0.0443	0.0443	0.0460	0.0440	0.0470
double-exp.	0.0620	0.0693	0.0617	0.0570	0.0633	0.0667	0.0620	0.0557	0.0637	0.0683	0.0630	0.0567
exponential	0.0617	0.0410	0.0850	0.0690	0.0767	0.0473	0.0923	0.0770	0.0863	0.0533	0.0957	0.0810
uniform	0.0323	0.0337	0.0310	0.0443	0.0313	0.0330	0.0303	0.0440	0.0323	0.0337	0.0307	0.0393
Cauchy	0.1190	0.1297	0.1167	0.1297	0.1173	0.1280	0.1150	0.1283	0.1150	0.1290	0.1160	0.1297

$m_i/n_i = 0.5,\ i = 1, 2$

$n_1 : n_2$	8:8	8:16	16:8	16:16	8:8	8:16	16:8	16:16	8:8	8:16	16:8	16:16
$m_1 : m_2$	4:4	4:8	8:4	8:8	4:4	4:8	8:4	8:8	4:4	4:8	8:4	8:8
normal	0.0303	0.0383	0.0440	0.0403	0.0317	0.0400	0.0407	0.0417	0.0317	0.0400	0.0410	0.0420
double-exp.	0.0543	0.0430	0.0577	0.0593	0.0563	0.0450	0.0577	0.0583	0.0580	0.0467	0.0560	0.0573
exponential	0.0507	0.0293	0.0880	0.0677	0.0663	0.0353	0.0950	0.0743	0.0767	0.0470	0.0970	0.0773
uniform	0.0190	0.0253	0.0257	0.0383	0.0200	0.0260	0.0263	0.0357	0.0217	0.0260	0.0260	0.0377
Cauchy	0.0867	0.1010	0.0960	0.1107	0.0863	0.1010	0.0930	0.1137	0.0863	0.1017	0.0940	0.1143

Tabelle 3.1: Beobachtete Fehlerwahrscheinlichkeiten 1. Art für $\varphi^*_{n,BFP}$

3.4 Diskussion der Ergebnisse

Wir geben zunächst die Ergebnisse der Vergleichsrechnungen an, die den Welch-Test

$$\varphi_n^{Welch} = \begin{cases} 1 & \\ & T_n = \frac{\bar{X}_{n_1} - \bar{Y}_{n_2}}{\sqrt{\frac{S_X^2}{n_1} + \frac{S_Y^2}{n_2}}} \\ 0 & \end{cases} \begin{matrix} > \\ \\ \leq \end{matrix} \Phi^{-1}(1-\alpha)$$

implementieren:

$\sigma_1^2 : \sigma_2^2$	1.0:1.0			
$n_1 : n_2$	8:8	8:16	16:8	16:16
normal	0.0613	0.0603	0.0587	0.0587
double-exp.	0.0593	0.0660	0.0663	0.0573
exponential	0.0590	0.0363	0.0863	0.0590
uniform	0.0580	0.0583	0.0630	0.0600
Cauchy	0.0367	0.0370	0.0337	0.0367
$\sigma_1^2 : \sigma_2^2$	1.0:1.5			
$n_1 : n_2$	8:8	8:16	16:8	16:16
normal	0.0637	0.0600	0.0620	0.0570
double-exp.	0.0597	0.0653	0.0683	0.0570
exponential	0.0723	0.0410	0.1027	0.0697
uniform	0.0590	0.0567	0.0667	0.0593
Cauchy	0.0393	0.0360	0.0327	0.0377
$\sigma_1^2 : \sigma_2^2$	1.0:2.0			
$n_1 : n_2$	8:8	8:16	16:8	16:16
normal	0.0640	0.0570	0.0643	0.0583
double-exp.	0.0603	0.0673	0.0700	0.0613
exponential	0.0807	0.0497	0.1140	0.0783
uniform	0.0567	0.0537	0.0667	0.0577
Cauchy	0.0383	0.0340	0.0333	0.0383

Tabelle 3.2: Beobachtete Fehlerwahrscheinlichkeiten 1. Art für φ_n^{Welch}

Wie im Einstichprobenfall ergibt sich auch hier die Tendenz, dass der Welch-Test, der auf einer Normalapproximation beruht, eher für Fälle mit größeren Stichprobenumfängen geeignet ist, während der $m(n)$ out of n bootstrap sich besser für kleine Stichprobenumfänge eignet. Zudem fällt auch hier im Zweistichprobenfall wieder auf, dass sysmmetrische Verteilungen mit den angegebenen Testverfahren wesentlich besser bearbeitet werden können als solche mit einer starken Schiefe.

Ein Vergleich der hier erzielten Simulationsergebnisse mit denen aus Janssen, A. und Pauls, Th. (2005) zeigt allerdings auf, dass der klassische bootstrap mit $m(n) \equiv n$ bessere Resultate liefert. Deswegen ist auch hier die Empfehlung auszusprechen, wenn möglich eine nicht zu geringe Resampling-Intensitätsrate zu wählen, sofern dies praktikabel sein sollte.

Allerdings ist hier im Zweistichprobenfall zusätzlich zu berücksichtigen, dass es mit dem Pitman'schen Permutationstest ein leicht zu implementierendes Verfahren gibt, das aus der Theorie (vgl. u.a. Janssen, A.(1998)) als bestes Testverfahren für das betrachtete Problem bekannt ist, zumindest wenn die Varianzen in den beiden Stichproben als annähernd gleich angenommen werden können. Deswegen sollte die Wahl des Testverfahrens von den Gegebenheiten der Daten abhängig gemacht werden und es ist nicht möglich, eine universelle Empfehlung zu einem speziellen Test auszusprechen.

Kapitel 4

Zusammenfassung und Ausblick

Wie die voranstehenden Rechnungen und Simulationsergebnisse zeigen, ist der $m(n)$ out of n bootstrap mit *low resampling intensity* in vielen Situationen eine praktikable Art, asymptotische Niveau α-Tests in solchen Fällen zu konstruieren, in denen aufgrund fehlender Verteilungsinformation kein exakter Test angegeben werden kann.

Dabei ist das hier gewählte Vorgehen, den bootstrap Stichprobenumfang m als einen festen Anteil des Stichprobenumfangs n der Original-Stichprobe zu wählen, zunächst einmal nicht die Modellsituation der *low resampling intensity*. Wie jedoch unter anderem in del Barrio, E; Cuesta-Albertos, J.A.; Matrán, C.(2002) erwähnt, ist dies die durchaus gängige Vorgehensweise in der Praxis, da die Umfänge der Original- sowie der resampling Stichprobe oft von vorne herein fest fixiert sind. Die Autoren geben zu bedenken, dass in dieser Situation die Intensitätsrate $c := \lim_{n\to\infty} \frac{m(n)}{n}$ noch nicht einmal eindeutig bestimmt werden kann. Liegen (wie zum Teil hier betrachtet) etwa $n = 16$ Originalvariablen zu Grunde und wird der resampling Stichprobenumfang $m = 4$ gewählt, so ist nicht unterscheidbar, ob $m(n) = \frac{n}{4}$ und somit $c = \frac{1}{4}$ oder aber $m(n) = \sqrt{n}$ und damit $c = 0$ als das hinter dem Experiment stehende Modell anzusehen ist. Aufgrund dessen setzen sich die Autoren in dem erwähnten Artikel das Ziel, Stetigkeitseigenschaften der asymptotischen Verteilung Γ_c^* der bootstrap Größen in Abhängigkeit der Intensitätsrate c herauszuarbeiten, was ihnen unter gewissen Voraussetzungen an die zu Grunde liegenden Original-Zufallsvariablen auch gelingt. Es ergibt sich, dass die Stetigkeit in allen Punkten $c \neq 0$ unter sehr moderaten Anforderungen an die Originalvariablen zu zeigen ist, während der Fall $c = 0$ weitergehende Restriktionen erfordert.

Es stellt sich damit natürlich unmittelbar die Frage, weshalb gerade bei der steigenden Leistungsfähigkeit der Computer der Anwendungsfall der *low resampling intensity* immer noch eine große praktische und theoretische Relevanz besitzt (Die erwähnte spanische Autorengruppe setzt sich

auch in einigen anderen Arbeiten mit der Auswirkung der Wahl der Intensitätsrate auf das asymptotische Verhalten der bootstrap Verteilung auseinander, vgl. dazu auch die Arbeiten von Athreya).
Eine Antwort darauf gibt die Gruppe belgischer Mathematiker um Paul Janssen, Jan Swanepoel und Noël Veraverbeke, die das Bootstrapverfahren im Kontext der *Survival Analysis* untersucht haben. So wird zum Beispiel in Janssen, P.; Swanepoel, J.; Veraverbeke, N.(2001) ausgeführt, dass es Probleme (etwa die konsistente Schätzung der Verteilungsfunktion des Maximums von n Zufallsgrößen) gibt, die mit dem herkömmlichen bootstrap Verfahren, bei welchem die Umfänge n der Original- sowie der resampling Stichprobe zusammenfallen, nicht zu lösen sind und somit der (nicht modifizierte) bootstrap nicht universell eingesetzt werden kann und darf. Allerdings arbeitete Jan Swanepoel im Jahre 1986 heraus, dass es einen Stichprobenplan mit *low resampling intensity*, also einen $m(n)$ out of n bootstrap mit $\frac{m(n)}{n} \to 0$ gibt, der die erwähnte Problemstellung löst. Ausgehend von dieser Erkenntnis bewegen sich die Forschungen dieser belgischen Gruppe dahin, bei gegebenem Problem optimale Intensitätsraten $\frac{m(n)}{n}$ für ein geeignetes Bootstrapverfahren abzuleiten. Siehe dazu unter anderem auch van Keilegom, I.; Veraverbeke, N.(1997) sowie Janssen, P.; Swanepoel, J.; Veraverbeke, N.(2002).

Ein weiterer, oft zitierter Übersichtsartikel zu der Vorgehensweise der *low resampling intensity* ist Bickel, P.J.; Götze, F.; van Zwet, W.R.(1997). Auch dort werden konkrete Szenarien untersucht, in denen der $m(n)$ out of n bootstrap eingesetzt werden sollte, weil der herkömmliche bootstrap fehlschlägt oder unpraktikabel ist.
Im ersten Teil ihres Artikels geben die Autoren Bedingungen an, unter denen das klassische Bootstrapverfahren (Resampling bezüglich der empirischen Verteilungsfunktion) fehlschlägt bzw. inkonsistente Schätzer liefert. Hierzu werden folgende Notationen eingeführt:

$X_1, \ldots, X_n$	i.i.d. Zufallsvariablen mit Werten in $\mathbb{R}^p$
F	Verteilungsfunktion von X_1
$\mathcal{F}_0$	Familie von Verteilungsfunktionen, aus der F stammt (z.B. Hypothesenmenge bei Testproblemen)
$\mathcal{F}$	größere Familie von Verteilungsfunktionen, die $\mathcal{F}_0$ umfasst
$T_n(X_1, \ldots, X_n, F) \equiv T_n(\hat{F}_n, F)$	interessierendes, symmetrisches Funktional
$\mathcal{L}_n(F)$	Verteilung von $T_n(\hat{F}_n, F)$
$\theta_n(F)$	zu schätzender Parameter von $\mathcal{L}_n(F)$
$\theta(F)$	Grenzwert von $\theta_n(F)$ für $n \to \infty$ auf $\mathcal{F}_0$

Ein Fehlschlagen des klassischen Bootstrapverfahrens kann unter anderem dann geschehen, wenn der zu schätzende Parameter $\theta_n(F)$ nicht stetig auf der Menge $\mathcal{F}_0$ ist. Die in der Praxis häufigste Fehlerquelle besteht nach Ansicht der Autoren jedoch darin, dass $\theta_n(F)$ zwar wohldefiniert auf ganz $\mathcal{F}$, die Grenzgröße $\theta(F)$ jedoch nur auf $\mathcal{F}_0$ defniert ist oder "wilde Unstetigkeiten" aufweist, wenn man sie als Funktion auf $\mathcal{F}$ betrachtet.
Anhand von Beispielen wird aufgezeigt, dass es konkrete Situationen gibt, in denen die empirische Verteilungsfunktion $\hat{F}_n$ zwar zu $\mathcal{F}$, nicht jedoch zu $\mathcal{F}_0$ gehört und das durch bootstrapping bezüglich $\hat{F}_n$ gewonnene $\theta_n(F)$ außerhalb von $\mathcal{F}_0$ nicht konvergiert. Dies hat dann ein Scheitern des klassischen Bootstrapverfahrens in solchen Anwendungsfällen zur Folge. Abhilfe kann in solchen Fällen die Verwendung einer anderen, konsistenten Schätzung $\tilde{F}_n$ für F schaffen, die in der interessierenden Menge $\mathcal{F}_0$ liegt.

Der zweite Teil des Artikels von Bickel, Götze und van Zwet behandelt dann verschiedene Formen des $m(n)$ out of n bootstraps, die als Alternativansätze zum oben angegebenen Lösungsvorschlag präsentiert werden. Dazu sei h eine beschränkte, reellwertige Funktion, die auf dem Wertebereich von T_n definiert ist, beispielsweise $t \to \mathbf{1}_{\{t \le t_0\}}$. Ziel ist die Schätzung von $\theta_n(F) \equiv E_F(h(T_n(\hat{F}_n, F)))$. Dazu werden vier Verfahren eingeführt:

(i) Der n/n bootstrap (Nichtparametrischer bootstrap):
Sei

$$B_n(F) = E^*(h(T_n(\hat{F}_n^*, F))) = n^{-n} \sum_{(i_1,\ldots,i_n)} h(T_n(X_{i_1}, \ldots, X_{i_n}, F)).$$

Dann heißt $B_n \equiv B_n(\hat{F}_n) = \theta_n(\hat{F}_n)$ der *n/n-bootstrap.*

(ii) Der m/n bootstrap:
Sei

$$B_{m,n}(F) = n^{-m} \sum_{(i_1,\ldots,i_m)} h(T_m(X_{i_1}, \ldots, X_{i_m}, F)).$$

Dann heißt $B_{m,n} \equiv B_{m,n}(\hat{F}_n) = \theta_m(\hat{F}_n)$ der *m/n-bootstrap.*
Der m/n-bootstrap bezieht seine Rechtfertigung aus der Generalvoraussetzung, dass sowohl $\mathcal{L}_n(F)$ als auch $\theta_n(F)$ für $n \to \infty$ gegen Grenzgrößen $\mathcal{L}(F)$ bzw. $\theta(F)$ konvergieren.

(iii) Der $\binom{n}{m}$ bootstrap:
Sei

$$J_{m,n}(F) = \binom{n}{m}^{-1} \sum_{(i_1<\ldots<i_m)} h(T_m(X_{i_1}, \ldots, X_{i_m}, F)).$$

Dann heißt $J_{m,n} \equiv J_{m,n}(\hat{F}_n)$ der $\binom{n}{m}$ *bootstrap.*
Der $\binom{n}{m}$ bootstrap entspricht einer Approximation des m-fachen Produktmaßes $F \times \ldots \times F$ durch Ziehen ohne Zurücklegen.

(iv) Der Stichprobenteilungs-Bootstrap:
Man nehme $n = m \cdot k$ an und definiere

$$N_{m,n}(F) \equiv k^{-1} \sum_{j=0}^{k-1} h(T_m(X_{j \cdot m+1}, \ldots, X_{(j+1) \cdot m}, F)).$$

Dann heißt $N_{m,n} \equiv N_{m,n}(\hat{F}_n)$ der *Stichprobenteilungs-Bootstrap.*
$N_{m,n}$ ist eine k-fache Cross-Validierung. Nachteile hierbei sind offensichtlich die willkührliche Aufteilung der Stichprobe und die Forderung, dass n groß sein muss, um m und k vernünftig wählen zu können.

Die Indizes der gezogenen Variablen werden in allen vier Verfahren durch eine zufällige Permutation gewonnen. Die Autoren zeigen, dass $B_{m,n}$, $J_{m,n}$ und $N_{m,n}$ allesamt konsistent sind für $m(n) \to \infty$ und $\frac{m}{n} \to 0$ für diejenigen Fälle, die sie im ersten Abschnitt untersucht haben, also insbesondere auch in manchen Situationen, in denen B_n ein inkonsistentes Schätzverfahren darstellt.

Der dritte Teil des Artikels beschäftigt sich dann mit der Fragestellung, ob Nachteile durch die Verwendung von $B_{m,n}$, $J_{m,n}$ bzw. $N_{m,n}$ entstehen, wenn B_n selbst schon konsistent ist. Die Autoren geben unter Zuhilfenahme der Entwicklung

$$\theta_n(F) = \theta(F) + \theta'(F) \cdot n^{-\frac{1}{2}} + \mathcal{O}(n^{-1}) \text{ auf } \mathcal{F}_0$$

Bedingungen an, unter denen $B_n = \theta_n(\hat{F}_n)$ ein effizienter Schätzer von $\theta_n(F)$ ist und führen aus, dass die m out of n-Bootstrapverfahren mit $\frac{m}{n} \to 0$ dann typischerweise die relative Effizienz 0 haben. Allerdings kann auch in diesen Fällen die Verwendung von $B_{m,n}$ weiterhin empfohlen werden, da es möglich ist, durch die Durchführung von $B_{n_0,n}$ sowie $B_{n_1,n}$ mit $n_0 < n_1 \ll n$ und anschließender Extrapolation einen Schätzer für θ_n zu gewinnen, der sich wie B_n verhält.

Wertvolle technische Resultate werden zusammenfassend in Csorgo, S.; Rosalsky, A.(2003) dargestellt.

Die voranstehenden Untersuchungen und Ergebnisse belegen, dass die hier behandelte Thematik ein aktueller Gegenstand der mathematischen Forschung mit hoher Praxisrelevanz ist. Es gibt deswegen noch einige interessante Fragestellungen in diesem Bereich, auf welche hier nicht eingegangen worden ist. So lohnt etwa auch die Betrachtung der Resamplingverfahren unter Alternativen, also die Untersuchung von *Gütefunktionen* der herausgestellten bootstrap Tests.

Anhang A

Source-Code ANSI-C

A.1 Headerdatei Zufallszahlengeneratoren

```
#ifndef _GENERATOREN_H_INCLUDED_
  #define _GENERATOREN_H_INCLUDED_

  /*********************************************************************/
  /* Header-Datei Zufallsgeneratoren fuer Bootstrap-Simulationen       */
  /* Voraussetzung: sizeof(unsigned long int) == 32 BIT                */
  /* Autor: Thorsten Dickhaus, Matrikelnummer: 1641523                 */
  /* Datum der letzten Aenderung: 21.02.2005                           */
  /*********************************************************************/

  #include <stdio.h>
  #include <stdlib.h>
  #include <math.h>
  #ifndef M_PI
   #define M_PI 3.14159
  #endif

  /* Startwerte fuer die Generatoren auslesen                          */
  extern int getseed(unsigned long int *a, unsigned long int *b);

  /* Startwerte fuer die Generatoren setzen                            */
  extern int setseed(unsigned long int a, unsigned long int b);

  /* Gleichverteilung, diskret auf [0; MAX32BIT]                       */
  extern unsigned long int uni_diskret(void);

  /* Gleichverteilung, kontinuierlich auf [0; 1]                       */
  extern double uni_cont(void);
```

```
  /* Gleichverteilung, kontinuierlich auf [-a; a]                    */
  extern double uni_offset(double a);

  /* Bernoulli-Verteilung mit Trefferwahrscheinlichkeit p            */
  extern int bernoullidf(double p);

  /* Binomialverteilung mit Parametern n und p                       */
  extern int binomialdf(int n, double p);

  /* Poisson-Verteilung mit Parameter lambda                         */
  extern int poissondf(double lambda);

  /* Cauchy-Verteilung                                               */
  extern double cauchydf(void);

  /* Normalverteilung mit Parametern mue und sigma^2                 */
  extern void normaldf(double mue, double sigma_square,
                       double *zufallszahl1, double *zufallszahl2);

  /* Weibull-Verteilung mit Parametern alpha und beta                */
  extern double weibulldf(double alpha, double beta);

  /* Exponentialverteilung mit Parameter lambda                      */
  extern double exponentialdf(double lambda);

  /* Geshiftete Exponentialverteilung mit Parameter a                */
  extern double shift_exponentialdf(double a);

  /* Doppelexponentialverteilung                                     */
  extern double double_exponentialdf(void);

#endif /* _GENERATOREN_H_INCLUDED_ */
```

A.2 Quellcodedatei Zufallszahlengeneratoren

```
/*****************************************************************/
/* Implementation Zufallsgeneratoren fuer Gleichverteilung       */
/* Voraussetzung: sizeof(unsigned long int) == 32 BIT            */
/* Autor: Thorsten Dickhaus, Matrikelnummer: 1641523             */
/* Datum der letzten Aenderung: 21.02.2005                       */
/*****************************************************************/



#include "generatoren.h"



/* Starting value Kongruenz 32 BIT */
static unsigned long con = 1803752341;


/* Starting value Tausworthe 32 BIT */
static unsigned long taus = 3697165728;



/*****************************************************************/
/* Startwerte fuer die Generatoren auslesen                      */
/*****************************************************************/
int getseed(unsigned long int *a, unsigned long int *b)
{
 *a = con;
 *b = taus;

 return(1);
}



/*****************************************************************/
/* Startwerte fuer die Generatoren setzen                        */
/*****************************************************************/
int setseed(unsigned long int a, unsigned long int b)
{
 if(a%2)
   {
```

```
    con = a;
    taus = b;
    return(1);
   }

 return(0);
}



/*********************************************************************/
/* Gleichverteilung, diskret auf [0; MAX32BIT]                       */
/*********************************************************************/
unsigned long int uni_diskret(void)
{
 unsigned long int n, lambda=69069;
 con = con * lambda;
 taus ^= taus >> 15;
 taus ^= taus << 17;
 n = taus ^ con;

 return((n>>1) & 017777777777);
}



/*********************************************************************/
/* Gleichverteilung, kontinuierlich auf [0; 1]                       */
/*********************************************************************/
double uni_cont(void)
{
 const double largest = 2147483648.0;

 return(uni_diskret()/largest);
}



/*********************************************************************/
/* Gleichverteilung, kontinuierlich auf [-a; a]                      */
/*********************************************************************/
double uni_offset(double a)
{
 return(2.0*a*(uni_cont()-0.5));
```

```
}


/*****************************************************************/
/* Bernoulli-Verteilung mit Trefferwahrscheinlichkeit p          */
/*****************************************************************/
int bernoullidf(double p)
{
 double zufallszahl = uni_cont();

 if(zufallszahl <= p)
   return(1);
 else
   return(0);
}


/*****************************************************************/
/* Binomialverteilung mit Parametern n und p                     */
/*****************************************************************/
int binomialdf(int n, double p)
{
 int zaehler = 0;
 int i;

 for(i=1; i<=n; ++i)
    zaehler += bernoullidf(p);

 return(zaehler);
}


/*****************************************************************/
/* Poisson-Verteilung mit Parameter lambda                       */
/*****************************************************************/
int poissondf(double lambda)
{
 double summe = 0.0;
 int zaehler=0;

 while(summe < 1.0)
```

```
      {
       summe += exponentialdf(lambda);
       ++zaehler;
      }

 return(zaehler);
}

/*********************************************************************/
/* Cauchy-Verteilung, Quantiltransformation                          */
/*********************************************************************/
double cauchydf(void)
{
 return(tan(M_PI*uni_cont()-0.5*M_PI));
}



/*********************************************************************/
/* Normalverteilung mit Parametern mue und sigma^2                   */
/* Box-Muller-Algorithmus                                            */
/*********************************************************************/
void normaldf(double mue, double sigma_square,
              double *zufallszahl1, double *zufallszahl2)
{
 double u = uni_cont();
 double v = uni_cont();
 double vorfaktor = sqrt(-2.0*log(u));
 double argument = 2*M_PI*v;
 double sigma = sqrt(sigma_square);

 *zufallszahl1 = vorfaktor * cos(argument);
 *zufallszahl1 = sigma * (*zufallszahl1) + mue;
 *zufallszahl2 = vorfaktor * sin(argument);
 *zufallszahl2 = sigma * (*zufallszahl2) + mue;

 return;
}



/*********************************************************************/
/* Weibull-Verteilung mit Parametern alpha und beta                  */
/* Quantiltransformation                                             */
```

```
/*********************************************************************/
double weibulldf(double alpha, double beta)
{
 return(pow(log(1.0/(1-uni_cont())), (1.0/beta)) / alpha);
}



/*********************************************************************/
/* Exponentialverteilung mit Parameter lambda                        */
/* Quantiltransformation                                             */
/*********************************************************************/
double exponentialdf(double lambda)
{
 return(-log(1.0-uni_cont()) / lambda);
}



/*********************************************************************/
/* Geshiftete Exponentialverteilung um Konstante a nach links        */
/* Quantiltransformation                                             */
/*********************************************************************/
double shift_exponentialdf(double a)
{
 return(-log(1.0-uni_cont()) - a);
}



/*********************************************************************/
/* Doppelexponentialverteilung                                       */
/* Quantiltransformation                                             */
/*********************************************************************/
double double_exponentialdf(void)
{
 return(exponentialdf(1.0)-exponentialdf(1.0));
}
```

A.3 Quellcodedatei für Test $\varphi^*_{n,I}$

```
/*****************************************************************/
/* Implementation von Test phi_1 fuer das Einstichprobenproblem  */
/* Autor: Thorsten Dickhaus, Matrikelnummer: 1641523             */
/* Datum der letzten Aenderung: 21.02.2005                       */
/*****************************************************************/



#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "../Bootstrap-Generatoren/generatoren.h"

#define N 32
#define FAKTOR 0.25
#define ALPHA 0.95
#define MONTES 250000
#define BOOTS 999

#define REAL double



/****************************************/
/* Stichprobenmittel berechnen          */
/****************************************/
REAL mittelwert(REAL daten[N], int anzahl)
{
 int i;
 REAL summe;

 summe = 0.0;
 for(i=0; i<anzahl; ++i)
    summe += daten[i];

 return(summe / anzahl);
}



/****************************************/
```

```
/* Test durchfuehren                          */
/*********************************************/
int main(int argc, char *argv[])
{
 REAL xdaten[N];
 REAL bootstrap_x[N];

 int m = (FAKTOR * N);
 REAL zahl1, zahl2;
 REAL mean_x, mean_iter;
 REAL teststatistik_orig, teststatistik_iter;
 REAL normierung_orig = (sqrt(N));
 REAL normierung_iter = (sqrt(m) * sqrt(N) / sqrt(N-1));
 unsigned long int monte, sample, boot_zaehler, monte_zaehler;
 int obs;
 int quantilstelle, i;
 REAL wurzel3 = sqrt(3);

 /* Monte-Carlo-Schleife */
 for(monte=0; monte<MONTES; ++monte)
    {
     /* Daten erzeugen, Cauchyverteilung             */
     /* for(obs=0; obs<N; ++obs)                     */
     /*    xdaten[obs] = cauchydf();                 */

     /* Daten erzeugen, Doppelexponentialverteilung  */
     /* for(obs=0; obs<N; ++obs)                     */
     /*    xdaten[obs] = double_exponentialdf();     */

     /* Daten erzeugen, Exponentialverteilung        */
     /* for(obs=0; obs<N; ++obs)                     */
     /*    xdaten[obs] = shift_exponentialdf(1.0);   */

     /* Daten erzeugen, Gleichverteilung             */
     /* for(obs=0; obs<N; ++obs)                     */
     /*    xdaten[obs] = uni_offset(wurzel3);        */

     /* Daten erzeugen, Normalverteilung             */
     for(obs=0; obs<N; obs+=2)
        {
         normaldf(0.0, 1.0, &zahl1, &zahl2);
         xdaten[obs] = zahl1;
         xdaten[obs+1] = zahl2;
        }
```

```
    /* Original-Teststatistik berechnen */
    mean_x =  mittelwert(xdaten, N);
    teststatistik_orig = normierung_orig * mean_x;

    /* Bootstrap-Schleife */
    boot_zaehler=0;
    for(sample=1; sample<=BOOTS; ++sample)
       {
        /* Bootstrap-Sample erzeugen */
        for(obs=0; obs<m; ++obs)
           bootstrap_x[obs] = xdaten[uni_diskret()%N];

        /* Bootstrap-Teststatistik auswerten */
        mean_iter = mittelwert(bootstrap_x, m);
        teststatistik_iter = normierung_iter * (mean_iter - mean_x);

        /* Vergleich mit Original-Teststatistik */
        if(teststatistik_orig > teststatistik_iter)
          ++boot_zaehler;
       }

    /* Rang der Original-Teststatistik ueberpruefen */
    quantilstelle = ALPHA*(BOOTS+1);
    if(boot_zaehler >= quantilstelle)
      ++monte_zaehler;
   }

 printf("Verteilung: Normal, M = %6d, B = %3d, n = %2d, m(n) = %2d, p = %6.4lf\n",
        MONTES, BOOTS, N, m, ( ((double) monte_zaehler) / MONTES));

 exit(EXIT_SUCCESS);
}
```

A.4 Quellcodedatei für Test $\varphi^*_{n,II}$

```
/*****************************************************************/
/* Implementation von Test phi_2 fuer das Einstichprobenproblem  */
/* Autor: Thorsten Dickhaus, Matrikelnummer: 1641523             */
/* Datum der letzten Aenderung: 21.02.2005                       */
/*****************************************************************/



#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "../Bootstrap-Generatoren/generatoren.h"

#define N 32
#define FAKTOR 0.25
#define ALPHA 0.95
#define MONTES 3000
#define BOOTS 999

#define REAL double



/****************************************/
/* Stichprobenmittel berechnen          */
/****************************************/
REAL mittelwert(REAL daten[N], int anzahl)
{
 int i;
 REAL summe;

 summe = 0.0;
 for(i=0; i<anzahl; ++i)
    summe += daten[i];

 return(summe / anzahl);
}



/****************************************/
```

```
/* Stichprobenvarianz berechnen            */
/*******************************************/
REAL stichprobenvarianz(REAL *daten, REAL mean, int n)
{
 int i;
 REAL summe;

 summe = 0.0;
 for(i=0; i<n; ++i)
    summe += ((daten[i] - mean) *  (daten[i] - mean));

 return(summe / (n-1));
}



/*******************************************/
/* Test durchfuehren                       */
/*******************************************/
int main(int argc, char *argv[])
{
 REAL xdaten[N];
 REAL bootstrap_x[N];

 int m = (FAKTOR * N);
 REAL zahl1, zahl2;
 REAL mean_x, mean_iter;
 REAL std_x, std_iter;
 REAL teststatistik_orig, teststatistik_iter;
 REAL normierung_orig = (sqrt(N));
 REAL normierung_iter = (sqrt(m));
 unsigned long int monte, sample, boot_zaehler, monte_zaehler;
 int obs;
 int quantilstelle, i;
 REAL wurzel3 = sqrt(3);

 /* Monte-Carlo-Schleife */
 for(monte=0; monte<MONTES; ++monte)
    {
     /* Daten erzeugen, Cauchyverteilung           */
     /* for(obs=0; obs<N; ++obs)                   */
     /*    xdaten[obs] = cauchydf();               */

     /* Daten erzeugen, Doppelexponentialverteilung */
```

```
/* for(obs=0; obs<N; ++obs)                    */
/*    xdaten[obs] = double_exponentialdf();    */

/* Daten erzeugen, Exponentialverteilung       */
/* for(obs=0; obs<N; ++obs)                    */
/*    xdaten[obs] = shift_exponentialdf(1.0);  */

/* Daten erzeugen, Normalverteilung */
for(obs=0; obs<N; obs+=2)
   {
    normaldf(0.0, 1.0, &zahl1, &zahl2);
    xdaten[obs] = zahl1;
    xdaten[obs+1] = zahl2;
   }

/* Daten erzeugen, Gleichverteilung            */
/* for(obs=0; obs<N; ++obs)                    */
/*    xdaten[obs] = uni_offset(wurzel3);       */

/* Original-Teststatistik berechnen */
mean_x =  mittelwert(xdaten, N);
std_x = sqrt(stichprobenvarianz(xdaten, mean_x, N));
teststatistik_orig = normierung_orig * mean_x / std_x;

/* Bootstrap-Schleife */
boot_zaehler=0;
for(sample=1; sample<=BOOTS; ++sample)
   {
    /* Bootstrap-Sample erzeugen */
    for(obs=0; obs<m; ++obs)
       bootstrap_x[obs] = xdaten[uni_diskret()%N];

    /* Bootstrap-Teststatistik auswerten */
    mean_iter = mittelwert(bootstrap_x, m);
    std_iter = sqrt(stichprobenvarianz(bootstrap_x, mean_iter, m));
    teststatistik_iter = normierung_iter * (mean_iter - mean_x) / std_iter;

    /* Vergleich mit Original-Teststatistik */
    if(teststatistik_orig > teststatistik_iter)
      ++boot_zaehler;
   }

/* Rang der Original-Teststatistik ueberpruefen */
quantilstelle = ALPHA*(BOOTS+1);
```

```
     if(boot_zaehler >= quantilstelle)
       ++monte_zaehler;
    }

 printf("Verteilung: Normal, M = %6d, B = %3d, n = %2d, m(n) = %2d, p = %6.4lf\n",
        MONTES, BOOTS, N, m, ( ((double) monte_zaehler) / MONTES));

 exit(EXIT_SUCCESS);
}
```

A.5 Quellcodedatei für Test φ_n^{as}

```
/*****************************************************************/
/* Implementation Normalapproximations-Test Einstichprobenproblem   */
/* Autor: Thorsten Dickhaus, Matrikelnummer: 1641523                */
/* Datum der letzten Aenderung: 21.02.2005                          */
/*****************************************************************/



#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "../Bootstrap-Generatoren/generatoren.h"

#define N 8
#define ALPHA 0.95
#define MONTES 250000
#define NORMAL_QUANTIL 1.645

#define REAL double



/****************************************/
/* Stichprobenmittel berechnen             */
/****************************************/
REAL mittelwert(REAL daten[N], int anzahl)
{
 int i;
 REAL summe;

 summe = 0.0;
 for(i=0; i<anzahl; ++i)
    summe += daten[i];

 return(summe / anzahl);
}



/****************************************/
/* Stichprobenvarianz berechnen            */
```

```
/*****************************************/
REAL stichprobenvarianz(REAL *daten, REAL mean, int n)
{
 int i;
 REAL summe;

 summe = 0.0;
 for(i=0; i<n; ++i)
    summe += ((daten[i] - mean) *  (daten[i] - mean));

 return(summe / (n-1));
}



/*****************************************/
/* Test durchfuehren                     */
/*****************************************/
int main(int argc, char *argv[])
{
 REAL xdaten[N];

 REAL zahl1, zahl2;
 REAL mean_x, std_x;
 REAL teststatistik_orig;
 REAL normierung_orig = (sqrt(N));
 unsigned long int monte, monte_zaehler=0;
 int obs;
 REAL wurzel3 = sqrt(3);

 /* Monte-Carlo-Schleife */
 for(monte=0; monte<MONTES; ++monte)
    {
     /* Daten erzeugen, Cauchyverteilung            */
     /* for(obs=0; obs<N; ++obs)                    */
     /*    xdaten[obs] = cauchydf();                */

     /* Daten erzeugen, Doppelexponentialverteilung */
     /* for(obs=0; obs<N; ++obs)                    */
     /*    xdaten[obs] = double_exponentialdf();    */

     /* Daten erzeugen, Exponentialverteilung       */
     /* for(obs=0; obs<N; ++obs)                    */
     /*    xdaten[obs] = shift_exponentialdf(1.0);  */
```

```
    /* Daten erzeugen, Normalverteilung */
    for(obs=0; obs<N; obs+=2)
       {
        normaldf(0.0, 1.0, &zahl1, &zahl2);
        xdaten[obs] = zahl1;
        xdaten[obs+1] = zahl2;
       }

    /* Daten erzeugen, Gleichverteilung            */
    /* for(obs=0; obs<N; ++obs)                    */
    /*    xdaten[obs] = uni_offset(wurzel3);       */

    /* Original-Teststatistik berechnen */
    mean_x =  mittelwert(xdaten, N);
    std_x = sqrt(stichprobenvarianz(xdaten, mean_x, N));
    teststatistik_orig = normierung_orig * mean_x / std_x;

    /* Rang der Original-Teststatistik ueberpruefen */
    if(teststatistik_orig > NORMAL_QUANTIL)
      ++monte_zaehler;
   }

 printf("Verteilung: Normal, M = %6d, n = %2d, p = %6.4lf\n",
        MONTES, N, ( ((double) monte_zaehler) / MONTES));

 exit(EXIT_SUCCESS);
}
```

A.6 Quellcodedatei für Test $\varphi^*_{n,BFP}$

```
/*********************************************************************/
/* Implementation des Bootstraptests fuer das Zweistichprobenproblem */
/* Autor: Thorsten Dickhaus, Matrikelnummer: 1641523                 */
/* Datum der letzten Aenderung: 21.02.2005                           */
/*********************************************************************/



#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "../Bootstrap-Generatoren/generatoren.h"

#define N_1 16
#define N_2 16
#define FAKTOR_1 0.5
#define FAKTOR_2 0.5
#define VARIANZ_1 1.0
#define VARIANZ_2 2.0
#define ALPHA 0.95
#define MONTES 3000
#define BOOTS 999

#define REAL double



/*****************************************/
/* Stichprobenmittel berechnen           */
/*****************************************/
REAL mittelwert(REAL *daten, int anzahl)
{
 int i;
 REAL summe;

 summe = 0.0;
 for(i=0; i<anzahl; ++i)
    summe += daten[i];

 return(summe / anzahl);
}
```

```
/*****************************************/
/* Stichprobenvarianz berechnen          */
/*****************************************/
REAL stichprobenvarianz(REAL *daten, REAL mean, int n)
{
 int i;
 REAL summe;

 summe = 0.0;
 for(i=0; i<n; ++i)
    summe += ((daten[i] - mean) *  (daten[i] - mean));

 return(summe / (n-1));
}



/*****************************************/
/* Test durchfuehren                     */
/*****************************************/
int main(int argc, char *argv[])
{
 REAL xdaten[N_1];
 REAL ydaten[N_2];
 REAL bootstrap_x[N_1];
 REAL bootstrap_y[N_2];
 REAL teststatistiken[BOOTS+1];

 int m1 = (FAKTOR_1 * N_1);
 int m2 = (FAKTOR_2 * N_2);
 REAL zahl1, zahl2;
 REAL sigma_1 = sqrt(VARIANZ_1);
 REAL sigma_2 = sqrt(VARIANZ_2);
 REAL mean_x, mean_iter_x;
 REAL mean_y, mean_iter_y;
 REAL var_x, var_iter_x;
 REAL var_y, var_iter_y;
 REAL teststatistik_orig, teststatistik_iter;
 unsigned long int monte, sample, boot_zaehler, monte_zaehler;
 int obs;
 int quantilstelle, i;
```

```
REAL wurzel3 = sqrt(3);

/* Monte-Carlo-Schleife */
for(monte=0; monte<MONTES; ++monte)
   {
    /* Daten erzeugen, Cauchyverteilung                    */
    /* for(obs=0; obs<N_1; ++obs)                          */
    /*    {                                                */
    /*     xdaten[obs] = sigma_1 * cauchydf();             */
    /*    }                                                */
    /* for(obs=0; obs<N_2; ++obs)                          */
    /*    {                                                */
    /*     ydaten[obs] = sigma_2 * cauchydf();             */
    /*    }                                                */

    /* Daten erzeugen, Doppelexponentialverteilung         */
    /* for(obs=0; obs<N_1; ++obs)                          */
    /*    {                                                */
    /*     xdaten[obs] = sigma_1 * double_exponentialdf(); */
    /*    }                                                */
    /* for(obs=0; obs<N_2; ++obs)                          */
    /*    {                                                */
    /*     ydaten[obs] = sigma_2 * double_exponentialdf(); */
    /*    }                                                */

    /* Daten erzeugen, Exponentialverteilung               */
    /* for(obs=0; obs<N_1; ++obs)                          */
    /*    {                                                */
    /*     xdaten[obs] = sigma_1 * shift_exponentialdf(1.0); */
    /*    }                                                */
    /* for(obs=0; obs<N_2; ++obs)                          */
    /*    {                                                */
    /*     ydaten[obs] = sigma_2 * shift_exponentialdf(1.0); */
    /*    }                                                */

    /* Daten erzeugen, Gleichverteilung                    */
    /* for(obs=0; obs<N_1; ++obs)                          */
    /*    {                                                */
    /*     xdaten[obs] = sigma_1 * uni_offset(wurzel3);    */
    /*    }                                                */
    /* for(obs=0; obs<N_2; ++obs)                          */
    /*    {                                                */
    /*     ydaten[obs] = sigma_2 * uni_offset(wurzel3);    */
    /*    }                                                */
```

```
/* Daten erzeugen, Normalverteilung */
for(obs=0; obs<N_1; obs+=2)
   {
    normaldf(0.0, 1.0, &zahl1, &zahl2);
    xdaten[obs] = sigma_1 * zahl1;
    xdaten[obs+1] = sigma_1 * zahl2;
   }
for(obs=0; obs<N_2; obs+=2)
   {
    normaldf(0.0, 1.0, &zahl1, &zahl2);
    ydaten[obs] = sigma_2 * zahl1;
    ydaten[obs+1] = sigma_2 * zahl2;
   }

/* Original-Teststatistik berechnen */
mean_x =  mittelwert(xdaten, N_1);
var_x = stichprobenvarianz(xdaten, mean_x, N_1);
mean_y =  mittelwert(ydaten, N_2);
var_y = stichprobenvarianz(ydaten, mean_y, N_2);
teststatistik_orig = (mean_x - mean_y) / sqrt((var_x/N_1) + (var_y/N_2));
/* printf("Original-Teststatistik: %lf\n", teststatistik_orig); */
teststatistiken[0] = teststatistik_orig;

/* Bootstrap-Schleife */
boot_zaehler=0;
for(sample=1; sample<=BOOTS; ++sample)
   {
    /* Bootstrap-Samples erzeugen */
    for(obs=0; obs<m1; ++obs)
       bootstrap_x[obs] = xdaten[uni_diskret()%N_1];
    for(obs=0; obs<m2; ++obs)
       bootstrap_y[obs] = ydaten[uni_diskret()%N_2];

    /* Bootstrap-Teststatistik auswerten */
    mean_iter_x = mittelwert(bootstrap_x, m1);
    var_iter_x = stichprobenvarianz(bootstrap_x, mean_iter_x, m1);
    mean_iter_y = mittelwert(bootstrap_y, m2);
    var_iter_y = stichprobenvarianz(bootstrap_y, mean_iter_y, m2);
    teststatistik_iter = ((mean_iter_x - mean_x) - (mean_iter_y - mean_y)) /
                          sqrt((var_iter_x/m1) + (var_iter_y/m2));

    /* Vergleich mit Original-Teststatistik */
    if(teststatistik_orig > teststatistik_iter)
```

```
          ++boot_zaehler;
        }

    /* Rang der Original-Teststatistik ueberpruefen */
    quantilstelle = ALPHA*(BOOTS+1);
    if(boot_zaehler >= quantilstelle)
      ++monte_zaehler;
   }

 printf("Normal, M = %4d, B = %3d, n1 = %2d, m1 = %2d, var1= %4.2lf,
                                   n2 = %2d, m2 = %2d, var2= %4.2lf,
                p = %6.4lf\n",
        MONTES, BOOTS, N_1, m1, VARIANZ_1,
                       N_2, m2, VARIANZ_2,
        ( ((double) monte_zaehler) / MONTES));

 exit(EXIT_SUCCESS);
}
```

A.7 Quellcodedatei für Test φ_n^{Welch}

```
/*****************************************************************/
/* Implementation des Welch-Tests fuer das Zweistichprobenproblem    */
/* Autor: Thorsten Dickhaus, Matrikelnummer: 1641523                 */
/* Datum der letzten Aenderung: 21.02.2005                           */
/*****************************************************************/



#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "../Bootstrap-Generatoren/generatoren.h"

#define N_1 16
#define N_2 16
#define VARIANZ_1 1.0
#define VARIANZ_2 2.0
#define ALPHA 0.95
#define MONTES 3000
#define NORMAL_QUANTIL 1.645

#define REAL double



/****************************************/
/* Stichprobenmittel berechnen          */
/****************************************/
REAL mittelwert(REAL *daten, int anzahl)
{
 int i;
 REAL summe;

 summe = 0.0;
 for(i=0; i<anzahl; ++i)
    summe += daten[i];

 return(summe / anzahl);
}
```

```
/*****************************************/
/* Stichprobenvarianz berechnen          */
/*****************************************/
REAL stichprobenvarianz(REAL *daten, REAL mean, int n)
{
 int i;
 REAL summe;

 summe = 0.0;
 for(i=0; i<n; ++i)
    summe += ((daten[i] - mean) *  (daten[i] - mean));

 return(summe / (n-1));
}



/*****************************************/
/* Test durchfuehren                     */
/*****************************************/
int main(int argc, char *argv[])
{
 REAL xdaten[N_1];
 REAL ydaten[N_2];

 REAL zahl1, zahl2;
 REAL sigma_1 = sqrt(VARIANZ_1);
 REAL sigma_2 = sqrt(VARIANZ_2);
 REAL mean_x, mean_y;
 REAL var_x, var_y;
 REAL teststatistik_orig;
 unsigned long int monte, monte_zaehler=0;
 int obs;
 REAL wurzel3 = sqrt(3);

 /* Monte-Carlo-Schleife */
 for(monte=0; monte<MONTES; ++monte)
    {
     /* Daten erzeugen, Cauchyverteilung                    */
     /* for(obs=0; obs<N_1; ++obs)                          */
     /*    {                                                */
     /*     xdaten[obs] = sigma_1 * cauchydf();             */
     /*    }                                                */
```

```
/* for(obs=0; obs<N_2; ++obs)                                */
/*    {                                                      */
/*     ydaten[obs] = sigma_2 * cauchydf();                   */
/*    }                                                      */

/* Daten erzeugen, Doppelexponentialverteilung             */
/* for(obs=0; obs<N_1; ++obs)                                */
/*    {                                                      */
/*     xdaten[obs] = sigma_1 * double_exponentialdf();       */
/*    }                                                      */
/* for(obs=0; obs<N_2; ++obs)                                */
/*    {                                                      */
/*     ydaten[obs] = sigma_2 * double_exponentialdf();       */
/*    }                                                      */

/* Daten erzeugen, Exponentialverteilung                   */
/* for(obs=0; obs<N_1; ++obs)                                */
/*    {                                                      */
/*     xdaten[obs] = sigma_1 * shift_exponentialdf(1.0); */
/*    }                                                      */
/* for(obs=0; obs<N_2; ++obs)                                */
/*    {                                                      */
/*     ydaten[obs] = sigma_2 * shift_exponentialdf(1.0); */
/*    }                                                      */

/* Daten erzeugen, Gleichverteilung                        */
/* for(obs=0; obs<N_1; ++obs)                                */
/*    {                                                      */
/*     xdaten[obs] = sigma_1 * uni_offset(wurzel3);          */
/*    }                                                      */
/* for(obs=0; obs<N_2; ++obs)                                */
/*    {                                                      */
/*     ydaten[obs] = sigma_2 * uni_offset(wurzel3);          */
/*    }                                                      */

/* Daten erzeugen, Normalverteilung */
for(obs=0; obs<N_1; obs+=2)
   {
    normaldf(0.0, 1.0, &zahl1, &zahl2);
    xdaten[obs] = sigma_1 * zahl1;
    xdaten[obs+1] = sigma_1 * zahl2;
   }
for(obs=0; obs<N_2; obs+=2)
   {
```

```
         normaldf(0.0, 1.0, &zahl1, &zahl2);
         ydaten[obs] = sigma_2 * zahl1;
         ydaten[obs+1] = sigma_2 * zahl2;
        }

     /* Original-Teststatistik berechnen */
     mean_x =  mittelwert(xdaten, N_1);
     var_x = stichprobenvarianz(xdaten, mean_x, N_1);
     mean_y =  mittelwert(ydaten, N_2);
     var_y = stichprobenvarianz(ydaten, mean_y, N_2);
     teststatistik_orig = (mean_x - mean_y) / sqrt((var_x/N_1) + (var_y/N_2));

     /* Rang der Original-Teststatistik ueberpruefen */
     if(teststatistik_orig > NORMAL_QUANTIL)
       ++monte_zaehler;
    }

 printf("Normal, M = %4d, n1 = %2d, var1= %4.2lf,
                          n2 = %2d, var2= %4.2lf,
                 p = %6.4lf\n",
        MONTES, N_1, VARIANZ_1, N_2, VARIANZ_2,
        ( ((double) monte_zaehler) / MONTES));

 exit(EXIT_SUCCESS);
}
```

Tabellenverzeichnis

Literaturverzeichnis

[1] Bickel, P.J.; Götze, F.; van Zwet, W.R. (1997). *Resampling fewer than n observations: Gains, losses and remedies for losses*. Statistica Sinica 7, 1-31.

[2] Csorgo, S.; Rosalsky, A. (2003). *A survay of limit laws for bootstrapped sums*. International journal of mathematics and mathematica, Vol. 43, 2835-2861.

[3] del Barrio, E. und Matrán, C. (2000). *The Weighted Bootstrap Mean for Heavy-Tailed Distributions*. Journal of Theoretical Probability, Vol. 13, No.2, 547-569.

[4] del Barrio, E.; Cuesta-Albertos, J.A.; Matrán, C. (2002). *Asymptotic stability of the bootstrap sample mean*. Stochastic Processes and their Applications 97, 289-306.

[5] Dikta, G. (2002). *Bootstrap Methods in Statistics*. Paderborn.

[6] Gänssler, P. und Stute, W. (1977). *Wahrscheinlichkeitstheorie*. Springer-Verlag, Berlin.

[7] Janssen, A. (1998). *Zur Asymptotik nichtparametrischer Tests*. Skripten zur Mathematischen Statistik Nr. 29, Münster.

[8] Janssen, A. (2000). *Resampling Student t-type statistics*. Heinrich-Heine-Universität, Düsseldorf.

[9] Janssen, A. und Pauls, Th. (2003). *How do bootstrap and permutation tests work?*. The Annals of Statistics 2003, Vol. 31, No. 3, 768-806.

[10] Janssen, A. und Pauls, Th. (2005). *A Monte Carlo comparison of studentized bootstrap and permutation tests for heteroscedastic two-sample problems*. Heinrich-Heine-Universität, Düsseldorf.

[11] Janssen, P.; Swanepoel, J.; Veraverbeke, N. (2001). *Modified bootstrap consistency rates for U-quantiles*. Statistics & Probability Letters 54, 261-268.

[12] Janssen, P.; Swanepoel, J.; Veraverbeke, N. (2002). *The modified bootstrap error process for Kaplan-Meier quantiles*. Statistics & Probability Letters 58, 31-39.

[13] Pauls, Th. (2003). *Resampling-Verfahren und ihre Anwendung in der nichtparametrischen Testtheorie*. Inaugural-Dissertation, Düsseldorf.

[14] van Keilegom, I. und Veraverbeke, N. (1997). *Estimation and bootstrap with cencored data in fixed design nonparametric regression*. Ann. Inst. Statist. Math., Vol. 49, No. 3, 467-491.

[15] Welch, B.L. (1937). *The significance of the difference between two means when the population variances are unequal*. Biometrika, Vol. 29, 350-362.

[16] Witting, H. (1985). *Mathematische Statistik I. Parametrische Verfahren bei festem Stichprobenumfang.* B.G. Teubner, Stuttgart.

[17] Witting, H. und Müller-Funk, U. (1995). *Mathematische Statistik II*. B.G. Teubner, Stuttgart.

Hiermit versichere ich, die Arbeit selbständig erstellt und keine anderen als die angegebenen Hilfsmittel benutzt zu haben.

Thorsten Dickhaus Düsseldorf, den 14. März 2005