Resampling mit R

ALAN T. ARNHOLT, BOONE, NC – ÜBERSETZUNG: MANFRED BOROVCNIK, KLAGENFURT

Zusammenfassung: *Dieser Aufsatz zeigt, wie man R benutzen kann, um Resampling mit und ohne Zurücklegen einfach durchzuführen .*

Einleitung

Mehrere Artikel in *Teaching Statistics* haben die Leser in Randomisierungstechniken eingeführt. Insbesondere erläutert Johnson (2001) Bootstrapping, eine Art von Resampling *mit* Zurücklegen mit Minitab; Taffe und Garnham (1996) wiederum stellen eine Bootstrap-Lösung für den Zwei-Gruppen-Vergleich mit Minitab vor. Kürzlich führte Christie (2004) die Leserschaft in Resampling mit Excel ein. Obwohl Minitab und Excel weit verbreitet sind, haben nicht alle Studierenden und Lehrer Zugang zu kommerzieller Software.

Der Autor stellt die Behauptung in den Raum, dass Studierende eher eine Software verwenden, die leicht auf ihrem PC installiert werden kann, als dass sie eine Software nutzen, zu der sie nur im Computerraum der Schule oder eben über einen respektablen Aufpreis Zugang haben.

Eine solche statistische Programmierumgebung ist R; das Paket ist eine freie Software, erhältlich von http://www.r-project.org Vorkompilierte Versionen gibt es für Windows, Macintosh und Linux (http://cran.at.r-project.org/). Zum Zeitpunkt des Verfassens des vorliegenden Artikels war die neueste Version R-2.0.1; um dies unter Windows zu installieren, muss man das File rw2001.exe [inzwischen ist dies R-2.5.0-win32.exe] herunterladen. Man benützt dazu die zum eigenen Zugang nächst gelegene Spiegel-Seite.

Dieser Artikel soll sowohl Studierende als Lehrer zum Gebrauch von R ermutigen, indem R-Code für die Aufgaben aus Christie (2004) bereitgestellt wird. Nach einem raschen Überblick über elementare R-Befehle und die Syntax, befasst sich der Rest des Artikels mit R, um dieselben Aufgaben wie in Christie (2004) zu lösen.

Überblick über R und ausgewählte Befehle

R ist eine Programmiersprache mit recht einfacher Syntax; beim Codieren muss man zwischen Groß- und Kleinschreibung unterscheiden.

R arbeitet mit benannten Datenstrukturen, der numerische Vektor ist darunter die einfachste. Um die Werte 3, 4, 5, 9 und 14 einem Vektor mit dem Namen `Frogs` zuzuordnen, verwendet man den R-Befehl

```
> Frogs <- c(3,4,5,9,14)
```

Dabei verwendet man eigentlich eine Zuordnung über die Funktion `c( )`, welche Werte aneinander reiht (‚Konkatenation'). Es gibt Tausende solcher vordefinierter Funktionen wie `mean( )`, `sd( )`, `var( )`, `IQR( )`, `quantile( )`, `hist( )`, `boxplot( )`, welche den Mittelwert, die Standardabweichung, die Varianz, den Interquartilabstand, ein Histogramm, einen Boxplot berechnen/erzeugen. Um Hilfe für irgendeine benannte R-Funktion zu bekommen, gibt man `?function` oder `help(function)` beim R-Cursor (>) ein. Z. B. erhält man Mittelwert und Standardabweichung der Werte in `Frogs` so:

```
> mean(Frogs)
[1]  7
> sd(Frogs)
[1]  4.52769310[1]
```

Die `[1]` am Beginn der Antwort ist der Index der ersten Zahl in dieser Zeile. In diesem Beispiel ist das nicht sonderlich informativ, es kann aber bei einem längeren Antwort-Vektor viel weiterhelfen. Wenn wir etwa zufällig neun Werte aus einer Normalverteilung mit Mittel 0 und Standardabweichung 1 erzeugen, so verwenden wir den Befehl

```
> rnorm(9,mean=0,sd=1)
```

und erhalten:

[1] In R steht ein Dezimalpunkt anstelle des in deutschsprachigen Ländern üblichen Kommas.

Original 'Resampling with R' in *Teaching Statistics* 29(2007)1, 21–26

```
[1]-0.63340789, 0.53253128, 0.77111686
[4]-0.67026081,-0.92304534,-0.36136974
[7] 0.59722085, 0.07256734,-0.50778981
```

Die `[7]` in diesem Beispiel zeigt an, dass `0.59722085` der siebente Wert im Vektor ist. Werte in einer benannten Datenstruktur, sagen wir der zweite bis vierte Wert von `Frogs`, können mit ihrem Index angesprochen werden; man schreibt

```
> Frogs[2:4]
[1]  4  5  9
```

Eine der angenehmsten Eigenschaften von R ist, dass es den Benützern erlaubt, sich ihre eigenen Funktionen zu schreiben und zu nutzen. Die Grundstruktur eines solchen Befehls lautet:

```
name <- function(arg1, arg2, … )
{
expression
}
```

Dabei sind `arg1` und `arg2` Argumente / Parameter, die an die Funktion übergeben werden. Leerzeichen nach Kommata sind optional, werden aber üblicherweise zur besseren Leserlichkeit geschrieben. Zur Illustration sei eine Benutzer-definierte Funktion `Mean` vorgestellt, die das Stichprobenmittel berechnet. Natürlich gibt es keinen Vorteil daraus, eine Funktion nocheinmal zu schreiben, die es in R ohnehin schon gibt. Man beachte, dass `x` ein beliebiges numerisches Objekt ist:

```
> Mean <- function(x)
+  {
+  sum(x)/length(x)
+  }

> Mean(Frogs)
[1]  7
```

Im letzten Beispiel hat R durch seine Voreinstellung automatisch ein + am Beginn jeder Zeile erzeugt, weil der Befehl zur Erzeugung der Funktion `Mean` noch nicht durch das Zeichen `}` abgeschlossen war.

Andere R-Befehle, die im Folgenden verwendet werden, sind `apply` und `sample`. Die Funktion `apply` wird dazu benützt, verschiedenste Kennziffern für die Zeilen einer $m \times n$-Matrix zu berechnen, wobei jede Zeile als eine Stichprobe des Umfangs n betrachtet wird. Eine 4×4-Matrix mit Zufallszahlen aus einer Normalverteilung mit Mittel 5 und Standardabweichung 1 soll zeigen, wie man die Funktion `apply` anwendet. Der Befehl `set.seed(14)` wird verwendet, damit bei jeder Anwendung dieselben ‚Zufallszahlen' erzeugt werden. Die `14` hat keine besondere Bedeutung, jeder Wert könnte dazu verwendet werden.

```
> set.seed(14)
>  values <- rnorm(16,5,1)
>  SmallMat<-matrix(values,nrow=4)
>  SmallMat <- round(SmallMat, 2)
>  SmallMat

      [ ,1]  [ ,2]  [ ,3]  [ ,4]
[1, ]  4.34   4.96   4.62   5.67
[2, ]  6.72   6.23   6.04   4.71
[3, ]  7.12   4.94   4.62   5.49
[4, ]  6.50   6.07   5.30   5.88
```

Eine Möglichkeit, das Maximum in jeder Zeile von `SmallMat` zu berechnen ist folgende:

```
>  apply(SmallMat, 1, max)
[1]  5.67  6.72  7.12  6.50
```

Das erste Argument von `apply` sind die Daten, während das zweite Argument `1` oder `2` ist, wenn man mit einer Matrix arbeitet. Die `1` sagt R, dass es den Befehl auf die Zeilen anwenden soll, während die `2` sagt, dass R auf Spalten operieren soll. Das letzte Argument ist jene Funktion, die man auf Zeilen oder Spalten anwenden soll. Um das Minimum in jeder Spalte von `SmallMat` zu berechnen, gibt man folgenden Befehl ein:

```
>  apply(SmallMat, 2, min)
[1]  4.34  4.94  4.62 4.71
```

Die Funktion `sample` dient dazu, eine Stichprobe von gegebenem Umfang aus einem definierten Objekt zu entnehmen. In den zwei ersten Aufgaben weiter unten entnimmt man die Stichprobe *ohne* Zurücklegen. Um eine Stichprobe vom Umfang 3 *mit* Zurücklegen aus den Werten von `Frog` zu bekommen, gibt man ein:

```
>  Frogs <- c(3,4,5,9,14)
>  sample(Frogs,size=3, replace=T)
[1]  9  4  4
```

Um eine andere Anordnung der Werte in `Frogs` zu erhalten, oder anders gesprochen, um eine Stichprobe vom Umfang 5 *ohne* Zurückzulegen zu entnehmen, gibt man ein:

```
>  sample(Frogs,size=5, replace=F)
[1] 3 9 14 5 4
```

Der Leser kann eine detaillierte Beschreibung jedes Befehls in der online-Hilfe über die Eingabe

`help(command)` oder `?command`

beim R-Cursor bekommen. Davon sollte man extensiven Gebrauch machen. Alle Befehle und Funktionen in diesem Artikel sind (mit Kommentaren) in einem Textfile unter
`http://www1.appstate.edu/~arnholta/TeachingStatistics/`
abrufbar; jeder Auszug daraus kann kopiert und beim R-Cursor eingefügt werden.

Beim Schreiben von R-Befehlen ist es allgemein angeraten, den Code statt in R direkt auf der Befehlszeile in Notepad oder Wordpad einzugeben Der gesamte Text einer Funktion kann dann mit Kopieren und Einfügen nach R übertragen werden. Windows-Benutzer, die LaTeX verwenden, können mit Vorteil auch R-WinEdt verwenden:
(`http://cran.us.r-project.org/contrib/extra/winedt`).

Zusatzpakete installieren

Sobald R installiert ist, kann man zusätzliche Pakete laden, um die Flexibilität von R zu vergrößern. Um die Analysen in diesem Artikel nachzuvollziehen, benötigt man noch das Paket `exactRankTests`. Vorausgesetzt, der Benutzer ist mit dem Internet verbunden, kann man dieses Paket mit folgendem Befehl nach dem R-Cursor installieren:

```
install.packages(„exactRankTests“,
  .libPaths( ) [1])
```

Die Windows-Version von R bietet eine bequeme Menüführung, um Zusatzpakete zu installieren. Um diese zu nutzen, wähle man `Packages` und dann `Install package(s) from CRAN`.

Es erscheint eine Rollbalkenliste, die man herunterrollt, bis `exactRankTests` erscheint; das wählt man aus. R wird dann fragen, ob die heruntergeladenen Files gelöscht werden sollen. Die Eingabe `y` wird dies nach Installation löschen. Die HTML-Files werden in der Folge immer aktualisiert, egal, ob man die heruntergeladenen Files gespeichert oder gelöscht hat.

Aufgabe1: Schätzen eines Mittelwerts

Christie's erstes Beispiel hat mit Stichprobenziehen *mit* Zurücklegen zu tun; man soll eine Schätzung angeben für die mittlere Zeit zwischen Autos, die auf der M1 nordwärts fahren. Den Datensatz kann man in Hand u. a. (1994, 3) finden. Ein Histogramm der Zwischenankunftszeiten ist in Abb. 1 wiedergegeben.

Mit Hilfe von Excel berechnet Christie eine „Resampling“-Standardabweichung und ein 95%-Vertrauensintervall, indem er die 2,5%- und 97,5%-Quantile der durch Resampling gewonnenen Verteilung von Stichproben-Mittelwerten berechnet.

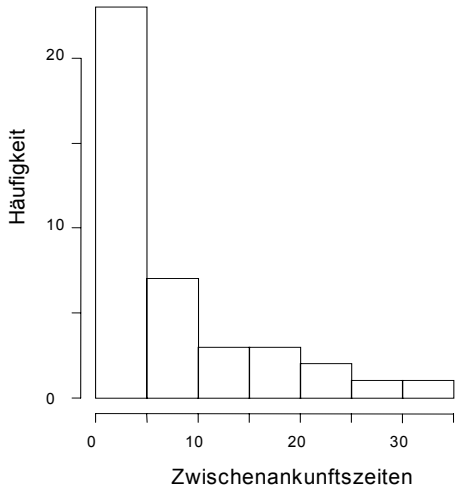

Abb. 1: Histogramm der Zwischenankunftszeiten, erzeugt durch den R-Befehl `hist(Times)`

Die Funktion `resamp`, mehr dazu später im Artikel, kann ganz einfach auf den ursprünglichen Datensatz der Zwischenankunftszeiten angewendet werden. Die Funktion `resamp` verlangt Daten (`x`) in Form eines Vektors, die Zahl der Stichproben, die man ziehen will (`m`, auf 10000 voreingestellt), die geschätzte Statistik (`theta`) und das Konfidenzniveau (`conf.level`, auf 0.95 voreingestellt).

Die Funktion `resamp` erzeugt eine $m \times n$-Matrix von Stichprobenwerten *mit* Zurücklegen aus den ursprünglichen Daten, wobei jede Zeile eine Stichprobe vom Umfang n darstellt.

Die statistische Kennziffer `theta` wird dann quer über die Zeilen (=Stichproben) angewendet und in einem Objekt mit dem Namen `thetastar` gespeichert. Mittelwert, Standardabweichung und Werte, die mit dem vom Benützer festgelegten Konfidenzniveau zusammen hängen, werden in der Folge aus den `thetastar`-Werten berechnet und als Liste wiedergegeben. Weil der Benützer ziemlich wahrscheinlich gar nicht alle „resampelten“ Werte aus `thetastar` sehen will, wird man typischerweise die Liste aus `resamp` in ein anderes Objekt speichern und die Information aus

der Liste ohne die Werte aus thetastar herauslesen.

Die ... am Ende der Liste der Argumente erlaubt es dem Benutzer, zusätzliche Argumente an die Funktion resamp zu übergeben. Die ... müssen hinzugefügt werden zur Funktion, wenn sie benützt wird, um die Korrelation in Aufgabe 2 in diesem Artikel zu berechnen. Wenn die Werte für die Zahl der Stichproben (m) oder das Konfidenzniveau nicht spezifiziert werden, nimmt resamp die voreingestellten Werte von 10000 bzw. 0.95.

```
resamp <- function(x, m=10000,
theta, conf.level= 0.95, …)
{
n <- length(x)
Data <- matrix(sample(x, size=n*m,
replace=T), nrow=m)
thetastar <- apply(Data,1,theta,…)
M <- mean(thetastar)
S <- sd(thetastar)
alpha <- 1-conf.level
CI <- quantile(thetastar,
c(alpha/2, 1-alpha/2))
return(list(ThetaStar=thetastar,
Mean.ThetaStar=M, S.E.ThetaStar=S,
Percentile.CI=CI))
}
```

In den folgenden Programmzeilen werden die Zwischenankunftszeiten im Vektor Times gespeichert. Es werden jedoch nicht alle Zwischenankunftszeiten im Code gezeigt, um Platz zu sparen. Die Funktion resamp wird auf Times angewendet, um 50000 Stichproben der Größe 41 zu erzeugen, das ist die Länge des ursprünglichen Datenvektors. In diesem Beispiel wird die zu schätzende statistische Kennziffer theta mit der R-Funktion mean gleichgesetzt. Man beachte, dass die Liste, die resamp als Ergebnis zurückgibt, dem Objekt answers zugewiesen wird und dass alle Komponenten der Liste angezeigt werden, ausgenommen jene von thetastar, indem man den Befehl answers[-1] eingibt.

```
> Times <-
c(12,2,6,2,19,5,34,4,1,4,...,16,2)
> answers <- resamp(Times,m=50000,
+ theta=mean)
> answers[-1]

$Mean.ThetaStar
[1]  7.776043
$S.E.ThetaStar
[1]  1.227912
$Percentile.CI
  2.5%  97.5%
  5.525  10.325
```

Aus dem Ausdruck kann man ersehen, dass die Resampling-Schätzung des Mittelwerts 7.78 bei einem Standardfehler von 1.23 beträgt; ein 95%-Konfidenzintervall auf der Basis von durch Resampling erzeugten Mittelwerten ist 5.525 bis 10.325. Die Zahlen unterscheiden sich erheblich von jenen, die Christie (2004) angibt, weil in diesem Beispiel *alle* Werte aus Hand u. a. (1994, 3) verwendet wurden und nicht nur die ersten 10. Wenn wir Resampling auf die ersten 10 Werte von Times anwenden, so erhalten wir Werte, die denen von Christie ziemlich ähnlich sind; genauer: 8.85 gegen 8.91 für den Mittelwert, 3.00 gegen 3.11 für den Standardfehler und 3.8 bis 15.9 gegen 3.7 bis 15.7 für ein 95%-Konfidenzintervall. Weil Resampling ein zufälliger Prozess ist, schwanken die Ergebnisse und gleichlautende Schätzungen sind nicht zu erwarten.

```
>  TimesChristie <- Times[1:10]
>  answers <-resamp(TimesChristie,
+  m=50000, theta=mean)
>  answers[-1]

$Mean.ThetaStar
[1]  8.906592
$S.E.ThetaStar
[1]  3.113669
$Percentile.CI
  2.5%  97.5%
  3.7  15.7
```

Aufgabe 2: Schätzen einer Korrelation

Die Daten aus Christie's zweiten Beispiel kommen auch aus Hand u. a. (1994, 5). Sie betreffen die jährliche Sterblichkeitsrate pro 100 000 Männern, gemittelt über die Jahre 1958 – 1964, und den Calciumgehalt (in ppm, das sind parts per million) in der öffentlichen Trinkwasserversorgung für 61 große Städte in England und Wales; Abb. 2 zeigt ein Streudiagramm der Daten.

Jetzt ist die Funktion theta der Produkt-Moment-Korrelationskoeffizient (nach Pearson). Durch read.table werden die Mortalitäts- und Calciumwerte aus einem externen ASCII-File in das Objekt TOT eingelesen. Um Platz zu sparen, werden nur die ersten fünf Zeilen, die in der Matrix TOTm gespeichert sind, angezeigt.

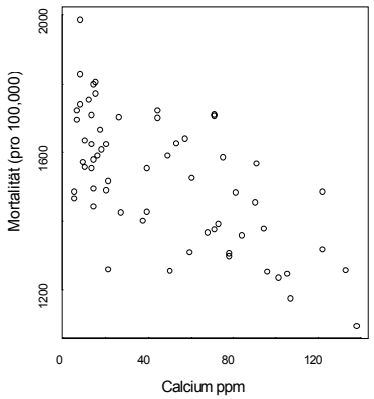

Abb. 2: Streudiagramm der Mortalität gegen den Calcium; erzeugt mit `plot(Calcium, Mortality)`

```
>  TOT<-read.table(„c:/calci.txt“,
+  header=T)
>  attach(TOT)
>  TOTm <- as.matrix(TOT)
>  dim(TOTm)
[1]  61  2
>  TOTm[1:5,]
    Mortality  Calcium
1        1247      105
2        1668       17
3        1466        5
4        1800       14
5        1609       18

>  n <- nrow(TOTm)
>  theta <- function(x, TOTm)
+  {
+  cor(TOTm[x,1], TOTm[x,2])
+  }
>  results <-resamp(x=1:n,m=10000,
+  theta=theta, TOTm=TOTm)
>  results[-1]

$Mean.ThetaStar
[1]  -0.6510727
$S.E.ThetaStar
[1]  0.07441022
$Percentile.CI
  2.5%  97.5%
  -0.7783729  -0.4860116
```

Vom Ausdruck liest man ab, dass die Resampling-Schätzung des Korrelationskoeffizienten –0.651 beträgt; dies bei einem Standardfehler von 0.0744. Ein 95%-Konfidenzintervall auf Basis der durch Resampling gewonnenen Korrelationskoeffizienten ist dann –0.778 bis –0.486. Wiederum unterscheiden sich diese Zahlen von jenen aus Christie, weil er nur eine *Teilmenge* der Daten und nicht den ganzen Datensatz verwendet. Man sollte anmerken, dass der Standardfehler für *r* in der Resampling-Analyse von Christie 0.432 beträgt, während der Standardfehler hier 0,102 ist (bei 10000 Stichproben, realisiert mit dem Programm weiter unten). Wenn man nur die ersten 13 Daten von `TOTm` (jene, die Christie benutzt) in Excel mit Resampling analysiert, so kommen Ergebnisse heraus, die mit jenen in R bis auf zwei Dezimalen übereinstimmen. Sehr wahrscheinlich ist der größere Standardfehler bei Christie darauf zurückzuführen, dass er relativ wenige Stichproben im Resampling durchführt.

```
>  TOTmC <- TOTm[1:13, ]
>  TOTmC
    Mortality  Calcium
[1, ]    1247      105
[2, ]    1668       17
[3, ]    1466        5
[4, ]    1800       14
[5, ]    1609       18
[6, ]    1558       10
[7, ]    1807       15
[8, ]    1299       78
[9, ]    1637       10
[10, ]   1359       84
[11, ]   1392       73
[12, ]   1755       12
[13, ]   1307       78
>  n <- nrow(TOTmC)
>  theta <- function(x, TOTmC)
+  {
+  cor(TOTmC[x,1], TOTmC[x,2])
+  }
>  results <- resamp(x=1:n,
+  m=10000, theta=theta,
+  TOTmC=TOTmC)
>  results[-1]

$Mean.ThetaStar
[1]  -0.8473666
$S.E.ThetaStar
[1]  0.1022861
$Percentile.CI
  2.5%  97.5%
  -0.9676258  -0.6268724
```

Aufgabe 3: Testen einer Hypothese

Das dritte Beispiel aus Christie ist ein Permutationstest, bei dem die Stichprobe aus den Daten *ohne* Zurücklegen gezogen wird. Der Datensatz stammt ursprünglich aus Ott und Mendenhall (1985, Aufgabe 8.17). Die Fragestellung lautet: Ist eine pharmazeutische Substanz in der Lage, den Blutdruck in Ratten zu verringern oder nicht? Aus zwölf Ratten wurden sechs per Zufall ausgewählt und zur Behandlungsgruppe zusammengefasst.

Die anderen sechs Ratten bildeten die Kontrollgruppe und erhielten nur ein Placebo. Die Senkung des Blutdrucks [mmHg] in beiden Gruppen betrug:

Kontrollgruppe	9	12	36	77.5	-7.5	32.5
Versuchsgruppe	69	24	63	87.5	77.5	40.0

Unter der Nullhypothese kommen die Daten aus *einer* Grundgesamtheit. Die entscheidende Frage lautet: Wenn die Nullhypothese zutrifft, wie wahrscheinlich ist es, eine Differenz zwischen Versuchs- und Kontrollgruppe (die Behandelten bzw. jene, die Placebo erhalten) zu beobachten, die mindestens so extrem ist wie die vorliegende.

```
>  Cont<-c(9,12,36,77.5,-7.5,32.5)
>  Treat<-c(69,24,63,87.5,77.5,40)
>  Blood <- c(Cont, Treat)
>  n <- length(Blood)
>  m <- 10000
>  TE <- array(0,c(m,n))
>  for (i in 1:m)
+  {
+  TE[i, ] <-
+  sample(Blood, n, replace=F)
+  }
>  Mean.T<-apply(TE[,7:12],1,mean)
>  Mean.C<-apply(TE[,1:6],1,mean)
>  Theta <- Mean.T - Mean.C
>  Obs.Mean.T <- mean(Blood[7:12])
>  Obs.Mean.C <- mean(Blood[1:6])
>  ThetaObs<-Obs.Mean.T-Obs.Mean.C
>  M <- mean(Theta)
>  S <- sd(Theta)
>  pval <- sum(Theta>=ThetaObs)/m
>  CI<-quantile(Theta,c(.025,.975))
>  Values<-list(ThetaObs=ThetaObs,
+  MeanTheta=M, SEtheta=S,
+  p.value=pval, CI=CI)
>  Values

$ThetaObs
[1]  33.5833
$MeanTheta

[1]  0.3076833
$SEtheta
[1]  18.01933
$p.value
[1]  0.0321
$CI
  2.5%  97.5%
-35.08333  35.08333
```

Natürlich kann man auch alle $\binom{12}{6} = 924$ verschiedenen Anordnungen der Daten bestimmen: Daraus ermittelt man dann die Zahl jener Anordnungen, welche eine größere Differenz in den Mittelwerten haben als in den ursprünglichen Daten beobachtet wurde (33.58). Während die Bestimmung aller möglichen Anordnungen per Hand zu lange braucht, ist das mit dem Computer rasch erledigt. Darüber hinaus kann man den exakten *p*-Wert mit `exactRankTests` bestimmen:

```
> library(exactRankTests)
> Cont<-c(9,12,36,77.5,-7.5,32.5)
> Treat<-c(69,24,63,87.5,77.5,40)
> perm.test(Treat,Cont,exact=T,a="g")

2-sample Permutation Test (scores
mapped into 1:(m+n) using rounded
scores)
data: Treat and Cont
T = 52, p-value = 0.03355
Alternative hypothesis: true mu is
greater than 0
```

Man beachte, dass der durch Resampling bestimmte *p*-Wert von 0.0321 sich gegen den exakten *p*-Wert von 0.03355 ganz beachtlich schlägt.

Literatur

Christie, D. (2004): Resampling with Excel. *Teaching Statistics* 26(1), 9–14.

Hand, D., Daly, F., Lunn, A., McConway, K. und Ostrowski, E. (Hsg.) (1994): *A Handbook of Small Data Sets*. London: Chapman and Hall.

Hothorn, T. und Hornik, K. (2005): ExactRankTests: Exact Distributions for Rank and Permutation Tests. *R package version 0.8-10*.

Johnson, R. W. (2001): An Introduction to the Bootstrap. *Teaching Statistics* 23(2), 49–54.

Ott, L. und Mendenhall, W. (1985): *Understanding Statistics*. Boston Duxbury Press.

R Development Core Team (2004): R: *A Language and Environment for Statistical Computing*. Wien: R Foundation for Statistical Computing. http://www.r-project.org

Taffe, J. u. Garnham, N. (1996): Resampling, the Bootstrap and Minitab. *Teach. Stat.* 18(1), 24–5.

Anschrift des Verfassers

Alan T. Arnholt
Department of Mathematical Sciences
Appalachian State University
121 Bodenheimer Dr, Walker Hall
Boone, NC 28608-2092
arnholt@math.appstate.edu