Stem-and-Leaf-Displays — selbstgemacht

H. P. Wolf

August 31, 2007, file: ms.rev

1 Einleitung

In diesem Papier wird eine eigene Umsetzung eines Stem-and-Leaf-Displays gewagt.[1] In der Tat enthielt der Weg der Programmierung einige Hürden, die inzwischen hoffentlich zum größten Teil übersprungen sind. Besondere Herausforderung sollte dabei in einem verständlichen Code sowie einer Auflistung von Tests zur Sicherstellung der gewünschten Funktionalität liegen.

2 Die Funktionsdefinition

2.1 Überblick

Der vorgestellte Vorschlag lehnt sich eng an *UREDA* (Hoaglin, Mosteller, Tukey, 1983: Understanding Robust and Exploratory Data Analysis) an. Haupteinsatzzweck wird in der Verwendung ohne weitere Parameter gesehen, jedoch sollten bei Unzufriedenheiten oder Sonderwünschen durch gezielte Setzungen Varianten erstellt werden können. Hierzu stehen folgende Argumente bereit:

1 ⟨*definiere Kurzkommentar* 1⟩≡

```
######################################################################
#Description:                                                        #
#   stem.leaf  produces a stem-and-leaf-display of a data set        #
#                                                                    #
#Usage:                                                              #
#   stem.leaf(data)                                                  #
#   stem.leaf(data,unit=100,m=5,Min=50,Max=1000,rule.line="Dixon"#
#                                                                    #
#Arguments:                                                          #
#   data:      vector of input data                                  #
#   unit:      unit of leafs in:  { ...,100,10,1,.1,.01,... }        #
#   m:         1, 2 or 5 -- 10/m=number of possible leaf digits      #
#   Min:       minimum of stem                                       #
#   Max:       maximum of stem                                       #
#   rule.line:   = "Dixon"    => number of lines <- 10*log(n,10)     #
#                = "Velleman" => number of lines <- 2*sqrt(n)        #
#                = "Sturges"  => number of lines <- 1 + log(n,2)     #
#   style:       = "Tukey"    => Tukey-like stem ( m = 2, 5 )       #
#   trim.outliers=TRUE        => outliers are printed absent        #
#   depths       =TRUE        => depths info is printed             #
#   reverse.negative.leaves=TRUE => neg.leaves are rev. sorted      #
#Author:                                                             #
#   Peter Wolf 05/2003 (modified slightly by J. Fox, 20 July 03)     #
#   rounding operation for comparing added 29 March 06               #
######################################################################
```

Das schwierigste Problem ist die Erstellung einer geeeigneten Skala. Ist die Skala gefunden, können die Daten als Blätter bzw. Extremwerte identifiziert und im Plot angebracht werden. Zum Schluß ist das Ergebnis geeignet auszugeben. Am 29.3.2006 wurde ein Rundungsproblem behoben.

[1] Hinweis von DT: *...aus Velleman/Hoaglin: ABC of EDA, Seite 15: It is easy to construct a Stem-and-Leaf-Display by hand... It is not nearly as easy to write a general computer program to produce Stem-and-Leaf-Displays.*

2 ⟨*start* 2⟩≡
⟨*definiere ms* 3⟩

3 ⟨*definiere ms* 3⟩≡
```
## ms <-
stem.leaf<-function(data, unit, m, Min, Max, rule.line=c("Dixon", "Velleman", "Sturges"),
     style=c("Tukey", "bare"), trim.outliers=TRUE, depths=TRUE, reverse.negative.leaves=TRUE,
     na.rm=FALSE){
#Author:  Peter Wolf 05/2003, (modified slightly by J. Fox, 20 July 03)
#  03/2005 additional rounding to prevent misclasification
```
⟨*checke Input* 5⟩
⟨*setze ggf.* `verb` *gemäß Debugging-Wunsch* 31⟩
⟨*definiere Kurzkommentar* 1⟩
⟨*generiere die Skala für den Plot* 4⟩
⟨*erstelle Stem-and-Leaf-Display* 14⟩
⟨*stelle Ergebnis zusammen* 28⟩
```
}
```

2.2 Skala

Für die Skala wird zunächst gemäß der festgelegten Regel eine grobe Zeilenzahl für den Plot bestimmt. Dann wird der Bereich, den die Skala abdecken muß, grob mittels `boxplot` festgestellt, sofern keine Skalengrenzen beim Funktionsaufruf angegeben worden sind. Mit Hilfe des Skalenbereiches und der Zeilenzahl läßt sich die anzustrebende Größe des Bereiches ermitteln, den es mit einer Zeile abzudecken gilt. Diese Größe gilt es mittels passendem Stamm und passender Maserung umzusetzen. Da im Folgenden die Position des Dezimalpunktes für das Stem-and-Leaf-Display keine Rolle mehr spielen, können alle relevanten Variablen transformiert / normiert werden. Mit den groben Berechnungen und den verarbeiteten Sonderwünschen kann dann die endgültige Skala erstellt werden.

4 ⟨*generiere die Skala für den Plot* 4⟩≡
⟨*stelle gemäß* `rule.line` *maximale Zeilenanzahl fest* 6⟩
⟨*ermittle mittels* `boxplot` *groben Skalenbereich* 7⟩
⟨*bestimme Intervalllänge und ggf. Faktor* `factor` 9⟩
⟨*berechne aus* `zeilen.intervall.laenge` *und* `factor` *Tickabstand* 10⟩
⟨*bestimme ggf. Maserung* `m` 11⟩
⟨*transformiere Daten* 12⟩
⟨*bestimme Skalenbereich* 13⟩

Zunächst gilt es den Input zu checken.

5 ⟨*checke Input* 5⟩≡
```
rule.line <- match.arg(rule.line)
style <- match.arg(style)
if(any(is.na(data))){
  print("Warning: there are NAs in the data!!")
  if(na.rm){ data<-data[!is.na(data)]
  }else{
    data[is.na(data)]<-mean(data,na.rm=TRUE)
  }
}
```

Zeilenanzahl Nach UREDA sind drei Regeln für die Anzahl der Zeilen einsetzbar, die auch zur Definition der Klassenanzahl von Histogrammen herangezogen werden. Die erste, die auf Dixon zurückgeht, gilt als bewährt, die zweite (von Velleman) empfiehlt sich besonders bei kleineren Stichprobenumfängen, die dritte (Sturges) findet weniger Unterstützung.
Zunächst berechnen wir nach der gewählten Regel die Zeilenanzahl des Plots. Dazu wird der Stichprobenumfang auf `n` abgelegt und zusätzlich werden die Daten sortiert.

6 ⟨*stelle gemäß* `rule.line` *maximale Zeilenanzahl fest* 6⟩≡
```
n<-length(data<-sort(data))
row.max <- floor(  c(Dixon   =10*log(n,10),
                     Velleman=2*sqrt(n),
                     Sturges =1+log(n,2)        ))[rule.line]
```

Skalenbereich In der Regel werden beim Aufruf keine Grenzen für den Bereich der Skala angegeben werden. Das Maximum und das Minimum können untauglich sein, da eventuelle Ausreißer zu üblen Effekten führen können. Deshalb wird, falls `Min` oder `Max` nicht festgelegt sind, diese mittels `boxplot` ermittelt. Die Spannweite der nicht-Ausreißer wird auf `spannweite.red` abgelegt.

7 ⟨*ermittle mittels* `boxplot` *groben Skalenbereich* 7⟩≡
```
stats<-boxplot(data,plot=FALSE)
if(missing(Min)) Min <- if (trim.outliers) stats$stats[1,1] else min(data, na.rm=TRUE)
if(missing(Max)) Max <- if (trim.outliers) stats$stats[5,1] else max(data, na.rm=TRUE)
spannweite.red<-Max - Min
```

Normierungsfaktor Zur Darstellung muß eine geeignete Normierung der Daten erfolgen. Hierzu wird intern ein Skalierungsfaktor `factor` ermittelt. Der Faktor zeigt an, mit welcher 10-er Potenz der Stamm multipliziert werden muß, damit er den Bereich der Input-Daten abdeckt. Das Maximum der Daten reicht nicht zu seiner Bestimmung aus, da Inputs aus [1,989] zu einem anderen Stamm als aus [980,989] führen. Besser ist die Spannweite als Ausgangspunkt. Diese erbringt im ersten Fall 998 und im zweiten 9. Im ersten Fall könnte sich ein Faktor von 100 ergeben und die Zeilenstruktur `0 | xyz` bis `10 | xyz`, im zweiten ein Faktor von 1 bei Zeilen der Form: `980 | xyz` bis `990 | xyz`. Weiter betrachten wir Daten aus einem Intervall [980,982]: Wenn wenige Daten vorliegen, werden sich die Stämme `980, 981, 982` ergeben. Steigt die Anzahl Daten an, steigt durch eine feinere Maserung die Zeilenanzahl. Bei 1000 Werten werden nach der ersten Regel ca. 30 Klassen benötigt, was zu einer Faktorveränderung führen muß: `9800, 9801, ..., 9802` mit Faktor 1/10. Nach Regel 2 benötigen wir dann 63 Klassen, nach der dritten 10. Im Fall von 5 Werten liefern die Regeln 6, 4 und 3. Hier ist eine Übersicht:

8 ⟨*zeige Beziehung Werteanzahl Zeilenanzahl gemäß Regel* 8⟩≡
```
anz<-rbind(dixon=floor(10*log(n,10)),
           velleman=floor(2*sqrt(n)),
           sturges=floor(1+log(n,2)))
colnames(anz)<-paste("n=",n,sep="")
print(anz)
```

```
         n=2 n=4 n=8 n=16 n=32 n=64 n=128 n=256 n=512 n=1024 n=2048
dixon      3   6   9   12   15   18    21    24    27     30     33
velleman   2   4   5    8   11   16    22    32    45     64     90
sturges    2   3   4    5    6    7     8     9    10     11     12
```

Wir erkennen, daß gemäß der ersten und der dritten Regel der Unterschied der Zeilenanzahlen eine Zehnerpotenz umfaßt, nach der zweiten differiert die Klassenanzahl um 2 Zehnerpotenzen.
Wir wollen ausgehend von der Regel die Länge des Intervalls bestimmen, das zu einer Zeile gehört. Dann versuchen wir dieser Länge durch Kombination von Faktor und Maserung möglichst nahe zu kommen. Ausreißer dürfen dabei natürlich nicht berücksichtigt werden.

Eine grobe Länge für das Zeilenintervall erhalten wir durch Division der gesammten Länge durch die anzustrebende Zeilenanzahl. Wenn eine Einheit angegeben worden ist, ergibt sich der Normierungsfaktor mittels `unit*10` sowie zur Erzielung einer 10-er Potenz durch einen Rundungsprozeß. Ist keine Einheit angegeben, ergibt sich diese aus der zur Zeilenintervalllänge nächst größeren Zehnerpotenz.

9 ⟨*bestimme Intervalllänge und ggf. Faktor* `factor` 9⟩≡
```
zeilen.intervall.laenge<-spannweite.red / row.max
if(missing(unit)){
       factor <- 10^ceiling(log(zeilen.intervall.laenge,10))
} else factor <- 10^round(log(unit*10,10))
debug.show("factor")
```

Zeilenintervalllänge Nun werden aufgrund der ermittelten Intervalllänge (im Zweifelsfall eher etwas groessere) Intervalle (und dadurch weniger Klassen) definiert: `delta.tick`. `z` zeigt schon eine normierte Länge an, die mit Länge der Größe `0,.1,.2,.5` verglichen werden. Der Vergleich mit 0 dient nur der Absicherung gegenüber pathologische Fällen. Als Resultat wird eine normierte Zeilenintervalllänge aus `.2,.5,1` ausgewählt.

10 ⟨*berechne aus* `zeilen.intervall.laenge` *und* `factor` *Tickabstand* 10⟩≡
```
z<-zeilen.intervall.laenge/factor  # z in (0.1 ,1]
delta.tick<-c(.2,.2,.5,1)[sum(z>c(0,.1,.2,.5))]
```

Maserung Nach der hier implementierten Auffassung gibt es nur Maserungen aus der Menge: $\{1, 2, 5\}$. Die Maserung `m` ist der Kehrwert des normierten Tickabstands, so daß Tickabstand .2 zur Maserung 5 führt, in einer Zeile können dann 2 verschiedene Ziffern auftauchen. Wird jedoch die Maserung über das Input-Argument `m` festgelegt, muß `delta.tick` angepaßt werden. Hierdurch lassen sich übrigens auch Maserungen wie `m=10` erzwingen.

11 ⟨*bestimme ggf. Maserung* `m` 11⟩≡
```
if(missing(m)) m<-round(1/delta.tick) else delta.tick<-1/m
debug.show("delta.tick"); debug.show("m")
```

Datennormierung Im weiteren Verlauf wollen wir mit normierten Werten weiterarbeiten. Deshalb transformieren wir Werte wie auch die Extremwerte der Skalen.

12 ⟨*transformiere Daten* 12⟩≡
```
data.tr<-data/factor
Min.tr <- Min/factor
Max.tr <- Max/factor
```

Skalenkonstruktion Die Skala ist wie folgt zu interpretieren: im positiven Bereich bezeichnet eine Eintragung x im Stem-and-Leaf-Display das Intervall $[x, x+1)$, im negativen $(x-1, x]$. In der folgenden Tabelle lassen sich einige Beispiele ablesen:

Min-Eintrag	Max-Eintrag	Wertebereich	Spannweite
2	6	[2.000, 6.999]	4.999
-2	2	[-2.999, 2.999]	5.998
-6	-2	[-6.999,-2.000]	4.999

Zur Ermittlung des Skalenbereiches runden wir zunächst die transformierten Extremwerte ab bzw. auf: Der erste Skaleneintrag ist wie der letzte eine ganze Zahl. Die Produktion einer Skala ist mit `seq` kein Problem. Jedoch müssen wir für die gewünschte Interpretation eventuell noch zwei Modifikationen vornehmen. Denn im positiven bezeichnet ein Stamm-Skalenwert die Untergrenze der Werte, die in der Zeile eingetragen werden sollen. Im negativen wechselt die Skala die Bezeichnung: `-2, -1, 0, 1, 2, ...` wird zu `-1, -0, 0, 1, 2, ....` Um bei einem gewünschten `Min==-2` dieses noch unterzubringen, müssen wir eine entsprechende Zeile ergänzen, die später bei `m=1` Werte von `-2.9999` bis `-2.0` aufnehmen kann. Entsprechend kann es vorkommen, daß als Maximum `-2` geplant ist. Dann wird ohne Korrektur, wie am kleinen Beispiel zu sehen ist, aus der Obergrenze `sk.max` von `seq` der Eintrag `-1` werden, der jedoch überflüssig ist.

13 ⟨*bestimme Skalenbereich* 13⟩≡

```
spannweite.red<-Max.tr - Min.tr
sk.min<-  floor(Min.tr)
sk.max<-ceiling(Max.tr)
skala <- seq(sk.min,sk.max,by=delta.tick)
if(sk.min<0) skala<-c(sk.min-delta.tick,skala)
if(sk.max<0) skala<-skala[-length(skala)]
debug.show("skala")
```

2.3 Displayerstellung

Jetzt sind die Vorarbeiten abgeschlossen: `unit`, `m` und `skala` sind definiert, es liegen transformierte Werte vor und der Erstellungsprozeß kann beginnen.
Für die Erstellung werden zunächst Ausreißer erkannt und entfernt. Die verbleibenden Daten werden im zentralen Plot eingetragen und zum Schluß für die Legende einige Infos zusammengefaßt.

14 ⟨*erstelle Stem-and-Leaf-Display* 14⟩≡

⟨*merke Ausreißer* 15⟩
⟨*konstruiere zentralen Teil des Plots* 16⟩
⟨*erstelle Interpretationshilfen* 27⟩

Ausreißer Ein Wert, der außerhalb des Bereiches der Skala liegt, ist ein Ausreißer. Ist der erste Skalenwert positiv, so sind das alle Werte, die kleiner als der Skalenwert sind. Ist `skala[1]` negativ, dann wird schon ein Wert genau von der Größe `skala[1]` nicht aufgenommen und gilt als `LO`. Für positive Maxima sind Werte Ausreißer, die größer gleich `skala[n.sk]+delta.tick` sind. Falls das Maximum unter Null ist, wird ein Wert der Größe `skala[n.sk]+delta.tick` gerade noch eingetragen.
Für die Tiefenberechnung ist es günstig, die Anzahl der Ausreißer zu vermerken. Die Ausreißer selbst werden auf `lower.line` bzw. `upper.line` abgelegt. Zum Schluß werden die Daten ohne Ausreißer auf `data.tr.red` abgelegt.

15 ⟨*merke Ausreißer* 15⟩≡
```
lo.limit <- if (trim.outliers) skala[1] else -Inf
lo.log   <- if(skala[1   ] <  0) data.tr <= lo.limit else data.tr <  lo.limit
n.sk <- length(skala)
hi.limit <- if (trim.outliers) skala[n.sk] + delta.tick else Inf
hi.log   <- if(skala[n.sk] >= 0) data.tr >= hi.limit else data.tr >  hi.limit

n.lower.extr.values <- sum(lo.log); n.upper.extr.values <- sum(hi.log)
if(0<n.lower.extr.values){
  lower.line<- paste("LO:", paste(data[lo.log],collapse=" "))
}
if(0<n.upper.extr.values){
  upper.line<- paste("HI:", paste(data[hi.log],collapse=" "))
}
data.tr.red <-data.tr[(!lo.log)&(!hi.log)]
```

2.3.1 Zentraler Stem-and-Leaf-Display

Für den zentralen Plot müssen zu den verbleibenden Daten Stämme und Blätter gefunden werden. Dann werden die Blätter auf die Zeilen verteilt. Die Blätter müssen zu Ästen zusammengefaßt und aus `skala` ein Baumstamm erstellt werden. Zum Schluß ist die Tiefeninformation zu ermitteln und anzubringen.

16 ⟨*konstruiere zentralen Teil des Plots* 16⟩≡
```
⟨zerlege Zahlen in Stamm und Blatt 17⟩
⟨verteile Blätter auf passende Klassen 18⟩
⟨ermittle Äste mit Blättern 19⟩
⟨konstruiere Skala und füge sie an den zentralen Plot an 20⟩
⟨ermittle Tiefen und füge sie an zentralen Plot an 26⟩
```

Zerlegung der Werte Stämme werden durch Abschneiden gebildet. Für negative Werte geschieht das durch Aufrunden, für positive durch Abrunden. Die Blätter ergeben sich über Differenzbildung von um eine Stelle nach links geshifteten Daten und Stämmen. Die Differenzen negativer Werte sind dann aufzurunden, die anderen abzurunden. Übrigens führte `ceiling((data.tr.red-stem)*10)` zu Fehlern.

17 ⟨*zerlege Zahlen in Stamm und Blatt* 17⟩≡
```
stem <- ifelse(data.tr.red<0, ceiling(data.tr.red), floor(data.tr.red) )
leaf <- floor(abs(data.tr.red*10-stem*10))
debug.show("leaf"); debug.show("stem")
```

Blätterzuordnung Die Blätter werden gemäß der Größe der Daten auf Klassen aufgeteilt. Die Klassen für nicht-negative Werte werden durch Zählen der Skalenwerte, die kleiner gleich sind, gefunden. Hier ist es für die Vorstellung praktisch, daß die Werte sortiert sind. Negative Werte werden nach der selben Logik zugeordnet, jedoch wird dazu vom Maximum aus operiert.
Damit leere Klassen keine Probleme bereiten, wird in jede Klasse zwischenzeitlich ein Dummyelement plaziert. Anhand von `class.of.data.tr` werden die Blätter gesplittet und die Dummyelemente wieder entfernt.
2.8 ist nicht gleich 2.8. Deshalb wurde am 29.3.2006 Rundungen mit `signif` in den Vergleichsprozess eingebaut.

18 ⟨*verteile Blätter auf passende Klassen* 18⟩≡
```
class.of.data.tr<-unlist(c(
   sapply(signif(data.tr.red[data.tr.red< 0],10),
     function(x,sk)length(sk)-sum(-sk<=-x),signif(skala,10))
  ,sapply(signif(data.tr.red[data.tr.red>=0],10),
     function(x,sk)sum( sk<= x),signif(skala,10))
))
debug.show("class.of.data.tr")
class.of.data.tr  <- c(1:length(skala),class.of.data.tr)
leaf.grouped      <- split(c(rep(-1,length(skala)),leaf),class.of.data.tr)
leaf.grouped      <- lapply(leaf.grouped, function(x){ sort(x[-1]) })
# debug.show("leaf.grouped")
```

`paste` regelt die Astbildung problemlos.

19 ⟨*ermittle Äste mit Blättern* 19⟩≡
```
leaf.grouped.ch <- paste("|",unlist(lapply(leaf.grouped,paste,collapse="")))
# debug.show("leaf.grouped")
```

Display-Skala Die Konstruktion der Bezeichnung für die Skalen verläuft in drei Schritten.

20 ⟨*konstruiere Skala und füge sie an den zentralen Plot an* 20⟩≡
⟨*merke negative Klassen und Klasse, die bei* −1 *beginnt* 21⟩
⟨*spiegele ggf. Blätter im negativen Bereich* 22⟩
⟨*ermittle Zeilennamen für den Stamm* 23⟩
⟨*modifiziere Zeilennamen gemäß Maserung* 24⟩

Für die Bezeichnung der Zeilen werden negative und `-0`-Klassen gemerkt.

21 ⟨*merke negative Klassen und Klasse, die bei* −1 *beginnt* 21⟩≡
```
class.negative <- skala < 0
class.neg.zero <- floor(skala) == -1
```

22 ⟨*spiegele ggf. Blätter im negativen Bereich* 22⟩≡
```
if (reverse.negative.leaves){
        for (i in seq(class.negative))
            if (class.negative[i]) leaf.grouped[[i]] <- rev(leaf.grouped[[i]])
}
```

Die Zeilennamen ergeben sich aus der Skala, indem negative Werte um 1 verschoben werden, die Klassen `class.neg.zero` bekommt den korrekten Namen `-0`.

23 ⟨*ermittle Zeilennamen für den Stamm* 23⟩≡
```
line.names <- skala
line.names[class.negative] <- line.names[class.negative]+1
line.names <- as.character(floor(line.names))
line.names[class.neg.zero] <- "-0"
```

Tukey-Stil Bei `style="Tukey"` werden spezielle Symbole zur Stammverschönerung angebracht. Wieder führen negative Werte zu Fallunterscheidungen.

24 ⟨*modifiziere Zeilennamen gemäß Maserung* 24⟩≡
```
if(style=="Tukey"){
  switch(as.character(m),
  "1"={},
  "2"={
        h<-round(2*(skala%%1)) #; line.names[h!=0] <- ""
        line.names<-paste(line.names,
                ifelse(skala<0,c(".","*")[1+h],c("*",".")[1+h]),sep="")
      },
  "5"={
        h<-round(5*(skala%%1)); line.names[h>0 & h<4] <- ""
        line.names<-paste(line.names, ifelse(skala<0,
                        c(".","s","f","t","*")[1+h],
                        c("*","t","f","s",".")[1+h]), sep="")
      }
  )
}
⟨definiere Funktion ragged.left 25⟩
line.names <- ragged.left(line.names)
```

Damit hinterher die |-Trennstriche untereinander stehen, ist eine Auffüllung mit Leerzeichen erforderlich. Dieses leistet die Funktion `ragged.left`.

25 ⟨*definiere Funktion* `ragged.left` 25⟩≡
```
ragged.left<-function(ch.lines){
  max.n <-max(n.lines<-nchar(ch.lines))
  h     <-paste(rep(" ",max.n),collapse="")
  ch.lines <-paste( substring(h,1,1+max.n-n.lines), ch.lines)
  ch.lines
}
```

Tiefenermittlung Die Tiefenermittlung geschieht über zwei Zählprozesse. Dabei müssen ggf. die Anzahlen der Ausreißer (`n.lower.extr.values` und `n.upper.extr.values`) beachtet werden.
Die Stelle des Medians liegt dort, wo die Tiefenvektoren, entstanden durch Kumulation von `n.class`, sich – graphisch gesprochen – schneiden. Dort kommen zwei Zeilen infrage. Die mit der kleineren Differenz zwischen den Zählvektoren ist die gesuchte.
Der jeweils kleinste Wert der Tiefenvektoren ist festzuhalten und das entstandene Objekt mit passend vielen Leerzeichen zu füllen. Weiter sind Tiefeneinträge in Zeilen ohne Blätter zu löschen. Nebenbei werden die Positionen leerer Zeilen vermerkt `select==F`.

26 ⟨*ermittle Tiefen und füge sie an zentralen Plot an* 26⟩≡
```
n.class<-unlist(lapply(leaf.grouped,length))
select <- (cumsum(n.class) > 0) & rev((cumsum(rev(n.class)) > 0))
depth    <-    cumsum(n.class)          + n.lower.extr.values
depth.rev<-rev(cumsum(rev(n.class))     + n.upper.extr.values)
debug.show("depth")

uplow<-depth>=depth.rev
pos.median<-which(uplow)[1] + (-1:0)
h <- abs(depth[pos.median]-depth.rev[pos.median])
pos.median<-pos.median[1]+(h[1]>h[2])
debug.show("pos.median")

depth[uplow]<-depth.rev[uplow]
depth<-paste(depth,"")
depth[pos.median]<-paste("(",n.class[pos.median],")",sep="")
depth[n.class==0]<-" "
depth <- if (depths) ragged.left(depth) else ""
```

Zur Information werden die wesentlichen Infos in der Variablen `info` zusammengefaßt.

27 ⟨*erstelle Interpretationshilfen* 27⟩≡

```
info<-     c(  paste("1 | 2: represents",1.2*factor),
           #  paste("    m:",m     ),
              paste(" leaf unit:",factor/10),
              paste("            n:",n     ))
```

2.4 Ausgabe

Zum Schluß werden die Ergebnisse in einem Objekt zusammengebunden bzw. ausgegeben.

28 ⟨*stelle Ergebnis zusammen* 28⟩≡

```
stem <- paste(depth, line.names, leaf.grouped.ch)
stem <- if((m!=5)||sum(select)>4) stem[select] else stem
result<-list( stem=stem)
if(exists("lower.line")) result<-c(lower=lower.line,result)
if(exists("upper.line")) result<-c(result,upper=upper.line)
result<-c(list( info=info), result)
for(i in seq(result)) cat(result[[i]],sep="\n")
invisible(result)
```

3 Demos

Für Demonstrationen bietet sich Chambers, Cleveland, Kleiner, Tukey (1983): *Graphical Methods for Data Analysis*, S.27, an. Dort wird ein Teil eines im Buch abgedruckten Ozon-Datensatzes mit verschiedenen `m`-Werten dargestellt:

29 ⟨*a* 29⟩≡

```
# Chambers, Cleveland, Kleiner, Tukey (1983), p27
oz<-c( 60+c(0,1,1,4,4,4,4,6,6,8,8,8,9),
       70+c(1,1,1,1,1,1,1,2,2,3,5,5),
       80+c(0,0,0,0,0,0,2,2,3,5,6,6,7,7,7,9) )
data(co2)
"bd384" <- c(2.968, 2.097, 1.611, 3.038, 7.921, 5.476, 9.858,
             1.397, 0.155, 1.301, 9.054, 1.958, 4.058, 3.918, 2.019, 3.689,
             3.081, 4.229, 4.669, 2.274, 1.971, 10.379, 3.391, 2.093,
             6.053, 4.196, 2.788, 4.511, 7.3, 5.856, 0.86, 2.093, 0.703,
             1.182, 4.114, 2.075, 2.834, 3.698, 6.48, 2.36, 5.249, 5.1,
             4.131, 0.02, 1.071, 4.455, 3.676, 2.666, 5.457, 1.046, 1.908,
             3.064, 5.392, 8.393, 0.916, 9.665, 5.564, 3.599, 2.723, 2.87,
             1.582, 5.453, 4.091, 3.716, 6.156, 2.039)
repeat{
  cat("Wahl des Tests:\n")
  h<-menu(c( "Ozon  - m=1", "Ozon  - m=2", "Ozon  - m=5", "co2   - m=1",
             "co2   - m=2", "co2   - m=5", "bd384 - m=1", "bd384 - m=2",
             "bd384 - m=5"))
  switch(h, stem.leaf(oz,m=1), stem.leaf(oz,m=2), stem.leaf(oz,m=5),
            stem.leaf(co2,m=1), stem.leaf(co2,m=2), stem.leaf(co2,m=5),
            stem.leaf(bd384,m=1), stem.leaf(bd384,m=2), stem.leaf(bd384,m=5)  )
  if(h==0) break
}
```

4 RD-File

John Fox wrote the following RD-File (some small changes are done by Peter Wolf).

30 ⟨*definiere Hilfe zu* `stem.leaf` 30⟩≡

```
\name{stem.leaf}
\alias{stem.leaf}

\title{Stem-and-Leaf Display}
\description{
  Creates a classical ("Tukey-style") stem-and-leaf display.
}

\usage{
stem.leaf(data, unit, m, Min, Max, rule.line = c("Dixon", "Velleman", "Sturges"),
    style = c("Tukey", "bare"), trim.outliers = TRUE, depths = TRUE,
    reverse.negative.leaves = TRUE, na.rm = FALSE)
}

\arguments{
  \item{data}{a numeric vector.}
  \item{unit}{leaf unit, as a power of 10 (e.g., \code{100}, \code{.01});
    if \code{unit} is missing \code{unit} is choosen by \code{stem.leaf}.}
  \item{m}{number of parts (1, 2, or 5) into which each stem will be separated;
  if \code{m} is missing the number of parts/stem
  (\code{m}) is choosen by \code{stem.leaf}.}
  \item{Min}{smallest non-outlying value; omit for automatic choice.}
  \item{Max}{largest non-outlying value; omit for automatic choice.}
  \item{rule.line}{the rule to use for choosing the desired number of lines
    in the display; \code{"Dixon"} = 10*log10(n); \code{"Velleman"} = 2*sqrt(n);
    \code{"Sturges"} = 1 + log2(n); the default is \code{"Dixon"}.}
  \item{style}{\code{"Tukey"} (the default) for "Tukey-style" divided stems;
    \code{"bare"} for divided stems that simply repeat the stem digits.}
  \item{trim.outliers}{if \code{TRUE} (the default), outliers are placed on \code{LO} and
    \code{HI} stems.}
  \item{depths}{if \code{TRUE} (the default), print a column of "depths" to the left of the
    stems; the depth of the stem containing the median is the stem-count enclosed in
    parentheses.}
  \item{reverse.negative.leaves}{if \code{TRUE} (the default), reverse direction the leaves on negative
    stems (so, e.g., the leaf 9 comes before the leaf 8, etc.).}
  \item{na.rm}{ if TRUE 'NA' values are removed otherwise exchanged by mean}
}

\details{
  Unlike the \code{stem} function in the \code{base} package, this function produces
  classic stem-and-leaf displays, as described in Tukey's \emph{Exploratory Data Analysis}.
}

\value{
  The computed stem and leaf display is printed out.
  Invisibly \code{stem.leaf} returns the stem and leaf
  display as a list containing the elements
  \code{info} (legend), \code{stem} (display as character vecter), \code{lower} (very small values) ,
  and \code{upper} (very large values).
}
\references{
    Tukey, J.
    \emph{Exploratory Data Analysis.}
    Addison-Wesley, 1977.
    }

\author{Peter Wolf, the code has been slightly modified by John Fox \email{jfox@mcmaster.ca}
    with the original author's permission, help page written by John Fox, the help page has been slightly mod

\seealso{\code{\link[base]{stem}}}
```

```
\examples{
stem.leaf(co2)
}

\keyword{misc}
```

5 Test

Testen ist eine schwierige Sache. Systematische Aufrufe werden sich hier besser als Zufallsaufrufe zu eignen. Zunächst empfiehlt es sich schon während des Entwicklungsprozesses, an bestimmten Punkten Öffnungen einzubauen, die bei Bedarf Auskunft über die Innereien während der Bearbeitung, also des Prozesses, geben. Dieses ist im Code umgesetzt durch `debug.show("xyz")`-Konstruktionen. Jetzt gilt es die Funktion `debug.show` geeignet zu definieren.

31 ⟨*setze ggf.* `verb` *gemäß Debugging-Wunsch* 31⟩≡

```
debug.show<-function(name){
  if(!exists("debug.cond")) return()
  if(debug.cond=="all"|| (name %in% debug.cond) ){
    cat(name,":\n"); obj<-eval(parse(text=name))
    if(is.vector(obj)){ print(obj) }
    return()
  }
}
```

Mit Hilfe dieser Testunterstützungsfunktion werden im Folgenden einige wichtige Tests absolviert. Zur Erinnerung hier noch einmal die Argumente. `unit,m,Min,Max,rule.line="Dixon",style="Tukey"`

5.1 Code-Erzeugung

32 ⟨* 32⟩≡

```
tangleR("ms.rev",expand.roots = "", expand.root.start = TRUE))
```

5.2 Diverse Tests

5.2.1 Fehlersituation von DT

Dietrich Trenkler hat einen Fehler gefunden, der auf Rundungsprobleme zurückgeführt werden konnte. In *verteile Blätter auf passende Klassen* wurde 2.8 mit 2.8 verglichen mit dem Ergebnis, dass 2.8 größer als 2.8 ist. Deshalb wurde am 29.3.2006 Rundungen mit `signif` in den Vergleichsprozess eingebaut. 10 Stellen sollten reichen.

33 ⟨* 32⟩+≡

```
debug.cond<<-""

"a" <- structure(c(12, 29, 49, 280, 78, 41, 49, 308, 70, 57,
 41, 37, 275, 33, 267, 37, 33, 57, 37, 41, 25, 41, 53, 74,
57, 53, 37, 49, 66, 70, 134, 33, 57, 45, 62, 250, 37, 271,
37, 41, 12, 70, 25), .Names = c("Acerola", "Ananas", "Apfel",                          "Apfel, getrocknet", "A
"Granatapfel", "Grapefruit", "Heidelbeeren", "Himbeeren",                             "Holunderbeeren", "Honi
"Johannisbeeren, schwarz", "Kaki", "Kirsche, s\"u{\ss}", "Kiwi", "Mandarine", "Mango", "Mirabellen", "Nektari
"Passionsfrucht", "Pfirsich", "Pflaumen",
"Pflaumen,getrocknet", "Preiselbeeren","Rosinen", "Satsuma",
"Stachelbeeren", "Wassermelone", "Weintrauben","Zitrone"))
names(a)<-NULL; aa<-c(rev(sort(a))[1:5],sort(a)[1:5])
print(stem.leaf(a,Min=0,Max=300) )
```

5.2.2 Erfolgreiche Tests

Als Datensätze wollen wir `oz` wie auch `co2` verwenden. Für den Test bietet sich eine kleine Unterstützungsfunktion an:

34 ⟨*definiere* `test` 34⟩≡

```
oz<-c( 60+c(0,1,1,4,4,4,4,6,6,8,8,8,9), 70+c(rep(1,7),2,2,3,5,5),
       80+c(rep(0,6),2,2,3,5,6,6,7,7,7,9) )
if(exists("data")) data(co2)
test<-function(what) {
  cat(what,"\n"); eval(parse(text=what)); return()
}
```

Mit `test` lassen sich bequem einige Tests erledigen. Von hinten beginnend testen wir, ob `style` für `m=2` und `m=5` wirksam wird, sofern es auf `"Tukey"` gesetzt ist. Damit ist auch gleich ein erster Test für `m` beschrieben.

35 ⟨*Test von* `style` 35⟩≡

```
cat("style-Test-start\n")
test('stem.leaf(oz,m=1,style="Tukey")')
test('stem.leaf(oz,m=2,style="Tukey")')
test('stem.leaf(oz,m=5,style="Tukey")')
test('stem.leaf(oz,m=1,style="")')
test('stem.leaf(oz,m=2,style="")')
test('stem.leaf(oz,m=5,style="")')
cat("style-Test-end\n")
```

```
style-Test-start
stem.leaf(oz,m=1,style="Tukey")
1 | 2: represents 12
    m: 1
 unit: 1
    n: 41
   13    6 | 0114444668889
  (12)   7 | 111111122355
   16    8 | 0000002235667779
         9 |
stem.leaf(oz,m=2,style="Tukey")
1 | 2: represents 12
    m: 2
 unit: 1
    n: 41
    7    6* | 0114444
   13    6. | 668889
  (10)   7* | 1111111223
   18    7. | 55
   16    8* | 000000223
    7    8. | 5667779
         9* |
stem.leaf(oz,m=5,style="Tukey")
1 | 2: represents 12
    m: 5
 unit: 1
    n: 41
   3    6* | 011
         t |
   7     f | 4444
   9     s | 66
  13    6. | 8889
  20    7* | 1111111
  (3)    t | 223
  18     f | 55
         s |
        7. |
  16    8* | 000000
  10     t | 223
   7     f | 5
   6     s | 66777
   1    8. | 9
        9* |
stem.leaf(oz,m=1,style="")
1 | 2: represents 12
    m: 1
 unit: 1
    n: 41
   13    6 | 0114444668889
  (12)   7 | 111111122355
   16    8 | 0000002235667779
         9 |
stem.leaf(oz,m=2,style="")
1 | 2: represents 12
    m: 2
 unit: 1
    n: 41
    7    6 | 0114444
   13    6 | 668889
  (10)   7 | 1111111223
   18    7 | 55
   16    8 | 000000223
    7    8 | 5667779
         9 |
stem.leaf(oz,m=5,style="")
1 | 2: represents 12
    m: 5
 unit: 1
    n: 41
   3    6 | 011
         6 |
   7    6 | 4444
   9    6 | 66
  13    6 | 8889
  20    7 | 1111111
  (3)   7 | 223
  18    7 | 55
         7 |
         7 |
  16    8 | 000000
  10    8 | 223
   7    8 | 5
   6    8 | 66777
   1    8 | 9
         9 |
style-Test-end
Thu May 22 13:47:44 2003
```

Test der verschiedenen Regeln. Wir probieren sowohl Datensatz `oz` wie auch `co2`

36 ⟨*Test von* `rule.line` 36⟩≡

```
cat("rule-Test-start\n")
test('stem.leaf(oz,rule.line="Dixon")')
test('stem.leaf(oz,rule.line="Velleman")')
test('stem.leaf(oz,rule.line="Sturges")')
test('stem.leaf(co2,rule.line="Dixon")')
test('stem.leaf(co2,rule.line="Velleman")')
test('stem.leaf(co2,rule.line="Sturges")')
cat("rule-Test-end\n")
```

```
rule-Test-start
stem.leaf(oz,rule.line="Dixon")
1 | 2: represents 12
    m: 5
 unit: 1
    n: 41
   3    6* | 011
         t |
   7     f | 4444
   9     s | 66
  13    6. | 8889
  20    7* | 1111111
  (3)    t | 223
  18     f | 55
         s |
        7. |
  16    8* | 000000
  10     t | 223
   7     f | 5
   6     s | 66777
   1    8. | 9
        9* |
stem.leaf(oz,rule.line="Velleman")
1 | 2: represents 12
    m: 2
 unit: 1
    n: 41
    7    6* | 0114444
   13    6. | 668889
  (10)   7* | 1111111223
   18    7. | 55
   16    8* | 000000223
    7    8. | 5667779
         9* |
stem.leaf(oz,rule.line="Sturges")
1 | 2: represents 12
    m: 2
 unit: 1
    n: 41
    7    6* | 0114444
   13    6. | 668889
  (10)   7* | 1111111223
   18    7. | 55
   16    8* | 000000223
    7    8. | 5667779
         9* |
stem.leaf(co2,rule.line="Dixon")
1 | 2: represents 12
    m: 2
 unit: 1
    n: 468
    8    31* | 33344444
   70    31. | 55555556666666666666667777777777788888888888888999999999999999
  135    32* | 000000000000000011111111111222222222222222333333333334444444444
  187    32. | 555555556666666667777777777777778888888889999999999
  233    33* | 000000001111111111222222222233333333344444444
  (40)   33. | 55555566666666677777777788888999999999
  195    34* | 00000001111111222222222233333334444444
  156    34. | 555555556666666677777778888888999999
  119    35* | 000001111111222222222333333333444444444
   78    35. | 5555555556666666677777777888888999999999999999
   33    36* | 000000111111222233333344444
    5    36. | 55566
         37* |
stem.leaf(co2,rule.line="Velleman")
1 | 2: represents 12
    m: 5
 unit: 1
    n: 468
         31* |
    3      t | 333
   15      f | 444445555555
   41      s | 66666666666666677777777777
   70    31. | 88888888888888899999999999999
   98    32* | 00000000000000011111111111
  126      t | 22222222222222233333333333
  143      f | 44444444555555555
  168      s | 66666666677777777777777
  187    32. | 888888889999999999
  205    33* | 000000001111111111
  225      t | 22222222223333333333
  (14)     f | 44444444555555
  229      s | 66666666667777777
  210    33. | 888889999999999
  195    34* | 00000001111111
  181      t | 222222222223333333
  163      f | 4444444555555555
  148      s | 6666666677777777
  133    34. | 88888889999999
  119    35* | 000001111111
  107      t | 22222222233333333333
   87      f | 444444444555555555
   69      s | 66666666777777777
   53    35. | 88888899999999999999
   33    36* | 000000111111
   21      t | 2222333333
   11      f | 444444555
    2      s | 66
         36. |
         37* |
stem.leaf(co2,rule.line="Sturges")
1 | 2: represents 12
    m: 1
 unit: 1
    n: 468
   70    31 | 333444445555555666666666666667777777777788888888888888999999999999999
  187    32 | 000000000000000011111111111222222222222222333333333334444444445555555566666666677777777777777778888888889999999999
  (86)   33 | 00000000111111111122222222223333333334444444455555566666666677777777788888999999999
  195    34 | 0000000011111112222222222333333334444444455555555666666667777777788888889999999
  119    35 | 000001111111222222222333333333444444444555555555666666667777777788888899999999999999
   33    36 | 00000011111122223333334444455566
         37 |
rule-Test-end
Thu May 22 13:52:49 2003
```

Test von unit

37 ⟨*Test von* unit 37⟩≡

```
cat("unit-Test-start\n")
test('stem.leaf(oz,unit=10)')
test('stem.leaf(oz,unit=1)')
test('stem.leaf(oz,unit=.1)')
cat("unit-Test-end\n")
```

```
unit-Test-start
stem.leaf(oz,unit=10)
1 | 2: represents 120
    m: 5
 unit: 10
    n: 41
         0* |
          t |
    1     f | 6
  (24)    s | 666666666666777777777777
   16    0. | 8888888888888888
         1* |
stem.leaf(oz,unit=1)
1 | 2: represents 12
    m: 5
 unit: 1
    n: 41
   3    6* | 011
          t |
   7     f | 4444
   9     s | 66
  13    6. | 8889
  20    7* | 1111111
  (3)    t | 223
  18     f | 55
          s |
        7. |
  16    8* | 000000
  10     t | 223
   7     f | 5
   6     s | 66777
   1    8. | 9
        9* |
stem.leaf(oz,unit=.1)
1 | 2: represents 1.2
    m: 1
 unit: 0.1
    n: 41
   1    60 | 0
   3    61 | 00
        62 |
        63 |
   7    64 | 0000
        65 |
   9    66 | 00
        67 |
  12    68 | 000
  13    69 | 0
        70 |
  20    71 | 0000000
  (2)   72 | 00
  19    73 | 0
        74 |
  18    75 | 00
        76 |
        77 |
        78 |
        79 |
  16    80 | 000000
        81 |
  10    82 | 00
   8    83 | 0
        84 |
   7    85 | 0
   6    86 | 00
   4    87 | 000
        88 |
   1    89 | 0
unit-Test-end
Thu May 22 13:56:30 2003
```

Test der Extremwertsetzungen.

38 ⟨*Test von* Min/Max 38⟩≡

```
cat("Max-Min-Test-start\n")
test('stem.leaf(oz,Min=65,Max=83,unit=.1,m=1)')
test('stem.leaf(oz,Min=65,Max=83,unit=1,m=1)')
test('stem.leaf(-oz,Min=-83,Max=-65,unit=.1,m=1)')
test('stem.leaf(-oz,Min=-83,Max=-65,unit=1,m=1)')
test('stem.leaf(1:12,Min=5,Max=8,unit=.1,m=1)')
test('stem.leaf(.5+(-7:6),Min=-3,Max=3,unit=.1,m=1)')
cat("Max-Min-Test-end\n")
```

```
Max-Min-Test-start
stem.leaf(oz,Min=65,Max=83,unit=.1,m=1)
1 | 2: represents 1.2
    m: 1
 unit: 0.1
    n: 41
        65 |
   9    66 | 00
        67 |
  12    68 | 000
  13    69 | 0
        70 |
  20    71 | 0000000
  (2)   72 | 00
  19    73 | 0
        74 |
  18    75 | 00
        76 |
        77 |
        78 |
        79 |
  16    80 | 000000
        81 |
  10    82 | 00
   8    83 | 0
LO: 60 61 61 64 64 64 64
HI: 85 86 86 87 87 87 89
stem.leaf(oz,Min=65,Max=83,unit=1,m=1)
1 | 2: represents 12
    m: 1
 unit: 1
    n: 41
   13     6 | 0114444668889
  (12)    7 | 111111122355
   16     8 | 0000002235667779
          9 |
stem.leaf(-oz,Min=-83,Max=-65,unit=.1,m=1)
1 | 2: represents 1.2
    m: 1
 unit: 0.1
    n: 41
   8    -83 | 0
  10    -82 | 00
        -81 |
  16    -80 | 000000
        -79 |
        -78 |
        -77 |
        -76 |
  18    -75 | 00
        -74 |
  19    -73 | 0
  (2)   -72 | 00
  20    -71 | 0000000
        -70 |
  13    -69 | 0
  12    -68 | 000
        -67 |
   9    -66 | 00
        -65 |
LO: -89 -87 -87 -87 -86 -86 -85
HI: -64 -64 -64 -64 -61 -61 -60
stem.leaf(-oz,Min=-83,Max=-65,unit=1,m=1)
1 | 2: represents 12
    m: 1
 unit: 1
    n: 41
         -9 |
   16    -8 | 0000002235667779
  (12)   -7 | 111111122355
   13    -6 | 0114444668889
stem.leaf(1:12,Min=5,Max=8,unit=.1,m=1)
1 | 2: represents 1.2
    m: 1
 unit: 0.1
    n: 12
   5     5 | 0
  (1)    6 | 0
   6     7 | 0
   5     8 | 0
LO: 1 2 3 4
HI: 9 10 11 12
stem.leaf(.5+(-7:6),Min=-3,Max=3,unit=.1,m=1)
1 | 2: represents 1.2
    m: 1
 unit: 0.1
    n: 14
   4    -3 | 5
   5    -2 | 5
   6    -1 | 5
  (1)   -0 | 5
   7     0 | 5
   6     1 | 5
   5     2 | 5
   4     3 | 5
LO: -6.5 -5.5 -4.5
HI: 4.5 5.5 6.5
Max-Min-Test-end
Thu May 22 14:11:37 2003
```

Klassenzuordnungstest:

39 ⟨*Klassenzuordnungstest* 39⟩≡

```
debug.cond<-"skala"
cat("Klassen-Test-start\n")
test('stem.leaf(c(.7+(1:12),4.999,5.0,5.001,7,7.001,7.999,
      8,8.001,8.999,9,9.001,9.999),Min=5,Max=8,unit=.1,m=1)')
test('stem.leaf(-c(.7+(1:12),4.999,5.0,5.001,7,7.001,7.999,
      8,8.001,8.999,9,9.001,9.999),Min=-8,Max=-5,unit=.1,m=1)')
test('stem.leaf(c(.7+(-5:5),-4.001,-4,-3.999, -3, 0, 3, 3.999,
      4, 4.001),Min=-3,Max=3,unit=.1,m=1)')
cat("Klassen-Test-end\n")
```

```
Klassen-Test-start
stem.leaf(c(.7+(1:12),4.999,5.0,5.001,7,7.001,7.999,
        8,8.001,8.999,9,9.001,9.999),Min=5,Max=8,unit=.1,m=1)
1 | 2: represents 1.2
    m: 1
 unit: 0.1
    n: 24
   8    5 | 007
   9    6 | 7
  (4)   7 | 0079
  11    8 | 0079
LO: 1.7 2.7 3.7 4.7 4.999
HI: 9 9.001 9.7 9.999 10.7 11.7 12.7
stem.leaf(-c(.7+(1:12),4.999,5.0,5.001,7,7.001,7.999,
        8,8.001,8.999,9,9.001,9.999),Min=-8,Max=-5,unit=.1,m=1)
1 | 2: represents 1.2
    m: 1
 unit: 0.1
    n: 24
  11    -8 | 0079
  (4)   -7 | 0079
   9    -6 | 7
   8    -5 | 007
LO: -12.7 -11.7 -10.7 -9.999 -9.7 -9.001 -9
HI: -4.999 -4.7 -3.7 -2.7 -1.7
stem.leaf(c(.7+(-5:5),-4.001,-4,-3.999, -3, 0, 3, 3.999, 4, 4.001),
         Min=-3,Max=3,unit=.1,m=1)
1 | 2: represents 1.2
    m: 1
 unit: 0.1
    n: 20
   6    -3 | 039
   7    -2 | 3
   8    -1 | 3
   9    -0 | 3
  (2)    0 | 07
   9     1 | 7
   8     2 | 7
   7     3 | 079
LO: -4.3 -4.001 -4
HI: 4 4.001 4.7 5.7
Klassen-Test-end
Thu May 22 14:19:01 2003
```

Hier noch einmal die Testaufrufe zusammengefaßt:

40 ⟨*Testaufrufe* 40⟩≡

- ⟨*Test von* `style` 35⟩
- ⟨*Test von* `rule.line` 36⟩
- ⟨*Test von* `unit` 37⟩
- ⟨*Test von* `Min/Max` 38⟩
- ⟨*Klassenzuordnungstest* 39⟩