

Ein Unterstützungstool zur Klausurenbenotung

File: klausurpunktegrenzen.rev
in: /home/wiwi/pwolf/tmp

August 9, 2006

Inhalt

1	Einleitung	2
2	Struktur der Lösung	2
2.1	Vorgehen	2
2.2	Input und Output	2
2.3	Fazit	4
3	Festsetzung der Notengrenzen	4
4	Umsetzung	6
4.1	Die Auswirkung der Veränderung von α	6
4.2	Die Auswirkung der Veränderung der Bestehensgrenze	7
4.3	Die Erstellung der Ausgabegraphik	7
4.4	Die Hauptfunktion: <code>select.limits</code>	7
4.5	Ein kleiner Test	8
5	Restarbeiten	8
5.1	Die methodenabhängige Berechnung der Notengrenzen	8
5.2	Die Elemente der Ausgabegraphik	9
5.3	Die Ergebnisspeicherung	9
5.4	<code>noten.fein</code> und <code>noten.grob</code>	10
6	Anhang: <code>quantile_stetig</code>	11
6.1	Erstellung einer Punkte-Demo-Datei	11
6.2	Die Definition der Sliderfunktion	11

1 Einleitung

Ärger bereitet aus Sicht der Studierenden, wenn die Notenvergabe nicht nachvollzogen werden kann. Gleiches gilt auch für die Prüfer, die für die Zuordnung von Zensuren zu Punktebereichen nicht leicht die Übersicht behalten. Hier könnte eine kleine Unterstützung angeraten sein.

In diesem Papier wird ein Werkzeug beschrieben, mit dem sich datengetriebenen Grenzen von Notenbereichen finden lassen. Der Ansatz setzt folgende Regeln um:

1. Eine Klausur ist bestanden, wenn sie mindestens 50% der Punkte besitzt, die zu der Note *sehr gut* gereicht hätte.
2. Die Zwischengrenzen werden linear aufgrund der beiden Grenzen (zur *Fünf* bzw. zur *Eins*) ermittelt.

Im Prinzip wird die Zuordnung *Punkte* \rightarrow *Noten* durch einen Parameter beschrieben. Denn ist die Grenze zur Eins gewählt, ergibt sich die zur Fünf und umgekehrt. Als dritte Möglichkeit der Festsetzung kann der Anteil α der Klausuren mit der Note *Eins* angesehen werden, der eine Vergleichbarkeit verschiedener Klausuren ermöglicht.

2 Struktur der Lösung

2.1 Vorgehen

Das hier präsentierte Werkzeug arbeitet in drei Schritten:

1. Einlesen der realisierten Punkteanzahlen
2. Interaktive Festsetzung der Steuerungsgröße α
3. Erstellung einer Ergebnisliste und eines mit Graphiken versehenen Notenspiegels

2.2 Input und Output

Wir gehen davon aus, dass die Punkte der Klausuren zeilenweise in einer Textdatei (.csv) abgelegt sind. Vor der Punkteanzahl können als Felder Namen und Matrikelnummern stehen:

```
Mustermann, Martin; 1234567; 87
Musterfrau, Monika; 1325476; 117
Flink, Pink; 1334456; 37
...
```

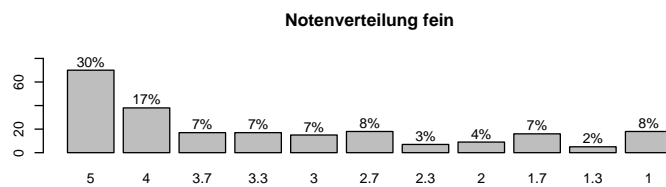
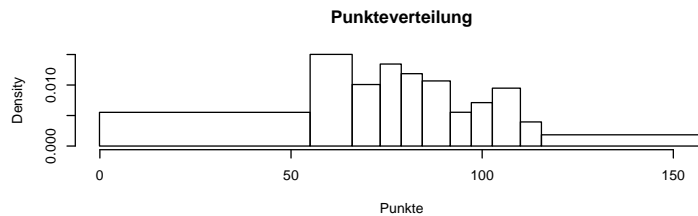
Dann wird die textuelle Ergebnisdatei folgende Form aufweisen:

```

...
-----
Mustermann, Martin    1234567    87    3
-----
Musterfrau, Monika    1325476    117   1
-----
Flink, Pink           1334456    37    5
-----
...

```

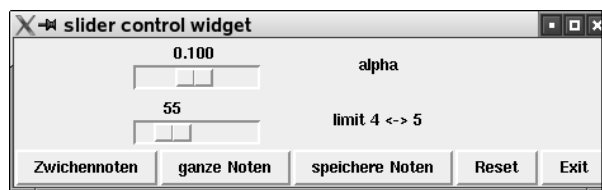
Für die Entscheidung können verschiedene Setzungen für α probiert und sofort die Auswirkung auf die Verteilung der Notenhäufigkeiten betrachtet werden. Die graphische Darstellung zu dem jeweiligen α -Wert wird ebenfalls als Datei gespeichert. Hier ein Beispiel:



Notenspiegel, alpha=0.1

Note:	5	4	3.7	3.3	3	2.7	2.3	2	1.7	1.3	1
bis Pkt:	55	66	73.3	78.8	84.3	91.7	97.2	102.7	110	115.5	158
Anzahl:	70	38	17	17	15	18	7	9	16	5	18
Prozent:	30	17	7	7	7	8	3	4	7	2	8

Die Werte für α bzw. die Bestehensgrenze können über folgendes Schieberfenster gesetzt werden:



2.3 Fazit

Mit dem Werkzeug liegt ein Vorschlag zur Orientierung vor, das den Einteilungsprozess und die Erstellung des Ergebnisberichtes (an das Prüfungsamt) unterstützt. Als Voraussetzung ist eine Digitalisierung der vergebenen Punkte erforderlich.

3 Festsetzung der Notengrenzen

Falls keine Zwischennoten gefordert sind, ergeben sich die Notengrenzen auf Basis der Punktegrenze `limit.1.2` zwischen den Zensuren Eins und Zwei wie folgt:

Mindestpunkte	für Note
0	5
$3/6 * \text{limit.1.2}$	4
$4/6 * \text{limit.1.2}$	3
$5/6 * \text{limit.1.2}$	2
$6/6 * \text{limit.1.2}$	1

`limit.4.5` zeigt die Punktzahl, mit der es gerade so eine 4 gibt.

```
1 <berechne grobe Grenzen 1> ≡  
  limit.4.5 <- 0.5 * limit.1.2  
  limits.grob <- limit.1.2 * (3:6) / 6
```

Die feinere Einteilung ist etwas komplizierter. Dazu überlegen wir: eine auf 1 gerundete Note hat, wer mindestens die reale Note 1.3 bekommen hat. Hätten wir die feinere Aufteilung bis in den 5-er Bereich bekäme man eine gerundete 4, falls man mit 4.3 oder besser benotet wird. Hieraus ergibt sich wie bei gerundeten Noten eine Differenz von $3 = 4.3 - 1.3$.

Zwischen Punktezahlen und Noten soll eine lineare Beziehung herrschen. Wir müssen also die Größen der Notenklassen berücksichtigen, also *Schrittweiten* zwischen benachbarten Noten. Diese haben wir in folgender Tabelle jeweils neben der besseren notiert. Die Schrittweiten drücken damit die Verbesserung gegenüber der nächst schlechteren Note aus.

Noten	Schrittweiten	Schrittweiten kumuliert	Faktor
5.0			
4.3	—	0.0	0/30
4.0	0.3	0.3	3/30
3.7	0.3	0.6	6/30
3.3	0.4	1.0	10/30
3.0	0.3	1.3	13/30
2.7	0.3	1.6	16/30
2.3	0.4	2.0	20/30
2.0	0.3	2.3	23/30
1.7	0.3	2.6	26/30
1.3	0.4	3.0	30/30
1.0	0.3	3.3	33/30

Wir setzen die Faktoren um, und berücksichtigen dabei, dass es die Note 4.3 laut PO nicht gibt. Der Sprung bis zur Eins beträgt 3, so dass wir die zur Verfügung stehenden Punkte ($0.5 * \text{limit}.1.2$) gemäß der Klassenbreiten durch Multiplikation mit den Faktoren aufteilen.

```
2 <berechne feine Grenzen 2> ≡
  limit.4.5<-delta.4.1<-0.5*limit.1.2
  limits.fein<-limit.4.5 +
    delta.4.1 * c(0,6,10,13,16,20,23,26,30,33)/30
```

Wir wollen nun den Zusammenhang zwischen den Notengrenzen und dem Parameter α modellieren. Ist α vorgegeben, ergibt sich die zugehörige Punkteanzahl als $(1 - \alpha)$ -Quantil. Um Eindeutigkeit für beliebige α -Werte aus $[0,1]$ herzustellen, muss eine der möglichen Quantil-Definitionen ausgewählt werden. Die maximale Punkteanzahl soll das 100%-Quantil sein, dann ist $\alpha = 0$. Die minimale Punkteanzahl wollen wir als $(1/n)$ -Quantil berechnen und ist $\alpha = (n - 1)/n$ zugeordnet. Weiter soll der Verlauf zwischen realisierten Punkten als linear angenommen werden. Damit ergibt sich folgende Beziehung:

$$1 - \alpha = \hat{F}(x_0) = \frac{\text{Anzahl Werte} \leq x_0}{\text{Werteanzahl}} + \frac{x_0 - \text{linker Nachbar}}{\text{Abstand der Nachbarpunkte}} \cdot \frac{1}{n}$$

sowie

$$\alpha = 1 - \frac{\text{Anzahl Werte} \leq x_0}{\text{Werteanzahl}} - \frac{x_0 - \text{linker Nachbar}}{\text{Abstand der Nachbarpunkte}} \cdot \frac{1}{n}$$

Diese Formel lässt sich schnell umsetzen:

```
3 <berechne alpha zu vorgegebener Punkteanzahl x0 3> ≡
  x1<-max(x[x<=x0]); x2<-min(x[x>x0])
  F.dach<-sum(x<=x0)/n+(x0-x1)/(x2-x1)/n
  alpha.x0<-1-F.dach
```

Die Umkehrbeziehung erhalten wir durch

$$x_{1-\alpha} = \hat{F}^{-1}(1 - \alpha)$$

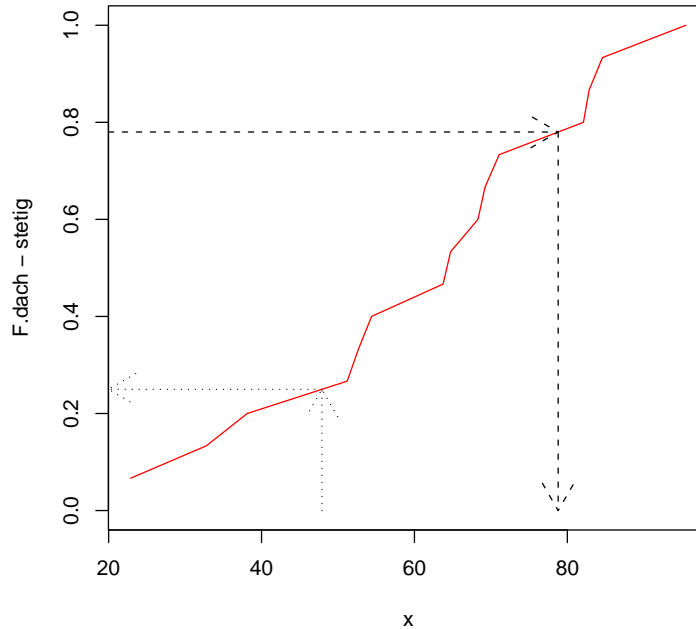
Diese ist implementiert in `quantile(1-alpha, x, type=4)`, und wir legen fest:

```
4 <berechne pkt.alpha zu vorgegebenem alpha 4> ≡
  if(DEBUG) {print(x); print(alpha)}
  pkt.alpha<-quantile(x, 1-alpha, type=4)
```

Zur Veranschaulichung sei ein Beispiel eingefügt.

```
5 <demonstriere Zugriff auf F-hat 5> ≡
  set.seed(13);punkte<-sort(rnorm(15,60,20))
  quantile_stetig(punkte)
```

(1-0.22)=0.78-Quantil (Typ 4):78.8
 F.dach(47.9)=0.25



Damit haben wir die wesentlichen Dinge festgelegt und können uns an eine Umsetzung wagen.

4 Umsetzung

4.1 Die Auswirkung der Veränderung von α

Zur Umsetzung der beschriebenen Ideen definieren wir eine Funktion, die den Effekt eines neu gesetzten α -Wertes umsetzt. Als Folge ist die zum neuen α zugehörige Punkteanzahl `limit.1.2` zu ermitteln, und die Notengrenzen sind wie beschrieben zu berechnen. Dann kann das Ergebnis dargestellt werden. Die neue Bestehensgrenze muss natürlich auch für die Steuerung angepasst werden.

```
6 <definiere effect.alpha 6> ≡
  effect.alpha<-function(...) {
    alpha<-slider(no=1)
    <berechne pkt.alpha zu vorgegebenem alpha 4>
    limit.1.2<-pkt.alpha
    <berechne Notengrenzen 11>
    slider(set.no.value=c(2,limit.4.5))
    <erstelle Graphik 8>
  }
```

4.2 Die Auswirkung der Veränderung der Bestehensgrenze

Die Entscheidung kann sich aber auch an der Bestehensgrenze orientieren. Nach Änderung dieser Grenze müssen die Grenze zur 1 sowie auch alle anderen Notengrenzen und das α neu berechnet werden.

```
7 <definiere effect.limit 7> ≡
  effect.limit<-function(...) {
    limit.1.2<-2*(limit.4.5<-slider(no=2))
    <berechne Notengrenzen 11>
    x0<-limit.1.2
    <berechne alpha zu vorgegebener Punkteanzahl x0 3>
    alpha<-alpha.x0; slider(set.no.value=c(1,alpha))
    <erstelle Graphik 8>
  }
```

4.3 Die Erstellung der Ausgabegraphik

Wir wollen ein Histogramm zur Darstellung der Punkteverteilung, ein Balkendiagramm für die Notenverteilung und eine Tabelle mit den wesentlichen Infos konstruieren.

```
8 <erstelle Graphik 8> ≡
  par(mfrow=c(3,1))
  <zeichne Histogramm der Punkte zu Notengrenzen 12>
  <zeichne Noten-Verteilung 13>
  <drucke Notenspiegel in die Graphik 14>
  par(mfrow=c(1,1)); "ok"
```

4.4 Die Hauptfunktion: select.limits

Die zentrale Funktion wollen wir `select.limits` nennen. Mit ihr können interaktiv ein α oder die Bestehensgrenze gewählt und die Auswirkungen studiert werden. Es ist übrigens das Einlesen aus einer Datei noch zu ergänzen.

```
9 <definiere select.limits 9> ≡
  select.limits<-function(sep=";",DEBUG=FALSE) {
  # if(missing(x)) {
    fname<-tclvalue(tkgetOpenFile(title="Datei mit den Klausurpunkten?"))
    if(!file.exists(fname)){cat("Datei nicht gefunden\n"); return()}
    studs.pkte<-readLines(fname)
    slider(obj.name="fname",obj.value=fname)
    x.orig<-x<-unlist(lapply(strsplit(studs.pkte,sep),function(x) rev(x)[1] ))
    x.orig<-x<-as.numeric(x)
    cat("x.orig:",x.orig)
  # }
  x<-sort(x); n<-length(x)
  <definiere effect.alpha 6>
  <definiere effect.limit 7>
  <definiere speichere.noten 15>
  <definiere noten.fein und noten.grob 16>
  slider(obj.name="method",obj.value="grob")
```

```

slider(c(effect.alpha, effect.limit),
      c("alpha", "limit 4 <-> 5"),
      c(0.01, round(.5*quantile(x, .8))),
      c(0.20, round(.5*max(x))),
      c(0.005, 1), c(0.1, round(.3*max(x))),
      c(noten.fein, noten.grob, speichere.noten),
      c("Zwischennoten", "ganze Noten", "speichere Noten")
); "ok"
}

```

4.5 Ein kleiner Test

Hier folgt ein Beispiel-Einsatz.

```

10 <ein fiktiver Einsatz von select.limits als Test 10> ≡
  set.seed(17); punkte<-pmax(0, round(rnorm(230, 70, 30)))
  matnr<-sample(2000:9000, 230)
  <definiere select.limits 9>
  # select.limits(punkte)
  select.limits()

```

5 Restarbeiten

5.1 Die methodenabhängige Berechnung der Notengrenzen

Die Berechnung der Notengrenzen hängt von der gewählten Methode (Zwischennoten ja/nein) ab. Diese wird festgestellt und dann wird die Berechnung der Grenzen vorgenommen. Weiter werden die Noten auf `names.noten` gespeichert.

```

11 <berechne Notengrenzen 11> ≡
  method<-slider(obj.name="method")
  if(method=="grob"){ # grob
    <berechne grobe Grenzen 1>
    limits<-limits.grob
    names.noten<-as.character(5:1)
  }else{ # fein
    <berechne feine Grenzen 2>
    limits<-limits.fein
    names.noten<-as.character(
      c(5.0, 4.0, 3.7, 3.3, 3.0, 2.7, 2.3, 2.0, 1.7, 1.3, 1.0))
  }
  limits<-c(-.1, limits, max(x)+1)
  slider(obj.name="limits", obj.value=limits)
  if(DEBUG) cat("method:", method, "limits:", limits, "\n")

```

5.2 Die Elemente der Ausgabegraphik

Für die graphische Ausgabe müssen noch die relevanten Anweisungen formuliert werden. Hier sind sie.

```
12 <zeichne Histogramm der Punkte zu Notengrenzen 12> ≡
    counts<-hist(x,breaks=limits,prob=TRUE,xlab="Punkte",
                main="Punkteverteilung")$counts
```

```
13 <zeichne Noten-Verteilung 13> ≡
    mp<-barplot(counts,names.arg=names.noten,
                ylim=c(0,1.2*max(counts)))
    text(mp,counts+.1*max(counts),
         paste(round(counts*100/n),"%",sep=""))
    title(paste("Notenverteilung",method))
```

Kommen wir zu den relevanten Infos, die wir in eine kleine Tabelle eintragen wollen.

```
14 <drucke Notenspiegel in die Graphik 14> ≡
    n<-length(limits); pos<-2
    plot(c(-1,n+0.5),c(-2,7),type="n",axes=F,xlab="",ylab="")
    title(paste("Notenspiegel",alpha=",round(alpha,2),sep=""))
    text(2:n,rep(5,n-1),names.noten,cex=1,pos=pos)
    text(2:n,rep(3,n-1),round(limits[-1],1),cex=1,pos=pos)
    text(2:n,rep(1,n-1),counts,cex=1,pos=pos)
    text(2:n,rep(-1,n-1),round(100*counts/sum(counts)),
         cex=1,pos=pos)
    text(1.1,5,"Note:",cex=1.2,pos=pos)
    text(1.1,3,"bis Pkt:",cex=1.2,pos=pos)
    text(1.1,1,"Anzahl:",cex=1.2,pos=pos)
    text(1.1,-1,"Prozent:",cex=1.2,pos=pos)
```

5.3 Die Ergebnisspeicherung

Die Ergebnisabspeicherung ist nur angedeutet. Sie muss nach den genauen Anforderungen noch angepasst werden.

```
15 <definiere speichere.noten 15> ≡
    speichere.noten<-function(...){
      cat("Speicherung beginnt\n")
      fname<-slider(obj.name="fname")
      # Originaldatei mit Punkten noch einmal lesen
      studs.pkts<-readLines(fname)
      # Punkte extrahieren
      x<-unlist(lapply(strsplit(studs.pkts,sep),function(x) rev(x)[1] ))
      x.orig<-x<-as.numeric(x)
      # noten herausfinden
      method<-slider(obj.name="method")
      limits<-slider(obj.name="limits")
      noten<-if(method=="grob") 5:1 else
```

```

        c(5.0,4.0,3.7,3.3,3.0,2.7,2.3,2.0,1.7,1.3,1.0)
# noten anhaengen
for(i in seq(x.orig)){
  n<-noten[sum(x.orig[i]>limits)]
  studs.pkte[i]<-paste(studs.pkte[i],sep=sep,n)
}
# liste ggf. sortieren
yesno<-tclvalue(tkmessageBox(message=
  "Soll die Notenliste nach erstem Feld sortiert werden?",
  icon="question",type="yesnocancel",default="yes"))
if(yesno=="yes"){
  studs.pkte<-sort(studs.pkte)
}
# Eintraege in kleine listen zerlegen
liste<-strsplit(studs.pkte,sep)
# lfd. Nummern vorhaengen
out<-substring(1000+(seq(x.orig)),2)
# geeignet Leerzeichen einfuegen
leerzeile<-paste(rep(" ",60),collapse="")
items<-length(liste[[1]])
for(i in 1:items){
  newitem<-unlist(lapply(liste,function(x) x[i] ))
  out<-paste(out,newitem)
  maxch<-max(nchar(out)); minch<-nchar(out)
  luft<-substring(leerzeile,1,maxch-minch)
  out<-paste(out,luft)
}
# Dateinamen abfragen
outfile<-tclvalue(tkgetSaveFile(initialfile="KlausurListe"))
if(" "==outfile){ cat("nichts gespeichert!\n"); return() }
if(0==length(grep(".txt$",outfile))) outfile<-paste(outfile,".txt",sep="")
# Speicherung
cat(out,file=outfile,sep="\n")
cat("Ergebnisdatei",outfile,"erstellt!\n")
## try({dev.copy(postscript,"KlausurStatistik.ps",horizontal=FALSE);
##      dev.off()})
outfile<-sub(".txt$",".pdf",outfile)
try({dev.copy(pdf,outfile); dev.off()})
cat("ErgebnisGraphik:",outfile,"erstellt!\n")
}

```

5.4 noten.fein **und** noten.grob

Für die Wahl der Methode legen wir eine Information auf der Slider-Variablen method ab.

```

16 <definiere noten.fein und noten.grob 16> ≡
  noten.fein<-function(...){
    slider(obj.name="method",obj.value="fein")
    effect.alpha()
  }
  noten.grob<-function(...){

```

```

    slider(obj.name="method",obj.value="grob")
    effect.alpha()
}

```

6 Anhang: quantile_stetig

Eine kleine Funktion zur Illustration des Zugriffs auf \hat{F} rundet dieses Papier ab.

```

17 (*17) ≡
quantile_stetig<-function(x,digits=2,qttype=4){
  x<-sort(x);  n<-length(x)
  redo<-function(...){
    alpha<-slider(no=1)
    qttype<-slider(no=2)
    x0<-slider(no=3)
    x1<-max(x[x<=x0]);  x2<-min(x[x>x0])
    delta.y<-(x0-x1)/(x2-x1)/n
    F.dach<-(1:n)/n
    gamma0<-sum(x<=x0)/n+delta.y
    plot(x,F.dach,type="l",ylim=0:1,ylab="F.dach - stetig",
         col="red")
    xmin<-par()$usr[1]
    y.0<-quantile(x,1-alpha,type=qttype)
    arrows(xmin,1-alpha,y.0,1-alpha,lty=2)
    arrows(y.0,1-alpha,y.0,0,lty=2)
    arrows(x0,gamma0,xmin,gamma0,lty=3)
    arrows(x0,0,x0,gamma0,lty=3)
    tit<-paste("1-",alpha,"=",1-alpha,"-Quantil (Typ ",
              qttype,"):",round(y.0,digits),"n",
              "F.dach(",x0,")=",round(gamma0,2),sep="")
    title(tit)
  }
  slider(redo,c("alpha","quantil-type","x0"),
        c(0,1,min(x)),c(1,9,max(x)),
        c(.01,1,.01*(max(x)-min(x))),
        c(.1,qttype,.5*min(x),max(x))); "ok"
}
quantile_stetig(rnorm(10))

```

6.1 Erstellung einer Punkte-Demo-Datei

```

18 <konstruiere Demo-Datei mit Punkten 18> ≡
set.seed(17);
punkte<-pmax(0,round(rnorm(230,70,30)))
matnr<-sample(200:900,230)
matnr<-1000000+13*(matnr)
Namen<-seq(punkte)
ll<-paste(letters[1:13],collapse="")
Namen<-substring(ll,1,sample(3:13,length(matnr),replace=T))
out<-paste(Namen,";",matnr,";",punkte,"\n",sep="")
cat(file="klpkte.csv",sep=" ",out)

```

6.2 Die Definition der Sliderfunktion

Für alle Fälle wird die Funktion `slider` bereitgestellt.

```

19 <start 19> ≡
library(tcltk)

```

```

slider<-
function (sl.functions, sl.names, sl.mins, sl.maxs, sl.deltas,
        sl.defaults, but.functions, but.names, no, set.no.value,
        obj.name, obj.value, reset.function, title)
{
    if (!missing(no))
        return(as.numeric(tclvalue(get(paste("slider", no, sep = ""),
            env = slider.env))))
    if (!missing(set.no.value)) {
        try(eval(parse(text = paste("tclvalue(slider", set.no.value[1],
            "<-", set.no.value[2], sep = "))), env = slider.env))
        return(set.no.value[2])
    }
    if (!exists("slider.env"))
        slider.env <- new.env()
    if (!missing(obj.name)) {
        if (!missing(obj.value))
            assign(obj.name, obj.value, env = slider.env)
        else obj.value <- get(obj.name, env = slider.env)
        return(obj.value)
    }
    if (missing(title))
        title <- "slider control widget"
    require(tcltk)
    nt <- tktoplevel()
    tkwm.title(nt, title)
    tkwm.geometry(nt, "+0+0")
    if (missing(sl.names))
        sl.names <- NULL
    if (missing(sl.functions))
        sl.functions <- function(...) {
        }
    for (i in seq(sl.names)) {
        eval(parse(text = paste("assign('slider", i, ', tclVar(sl.defaults[i]), env=slider.env)",
            sep = ")))
        tkpack(fr <- tkframe(nt))
        lab <- tklabel(fr, text = sl.names[i], width = "25")
        sc <- tkyscale(fr, from = sl.mins[i], to = sl.maxs[i],
            showvalue = T, resolution = sl.deltas[i], orient = "horiz")
        tkpack(lab, sc, side = "right")
        assign("sc", sc, env = slider.env)
        eval(parse(text = paste("tkconfigure(sc, variable=slider",
            i, ")", sep = "))), env = slider.env)
        sl.fun <- if (length(sl.functions) > 1)
            sl.functions[[i]]
        else sl.functions
        if (!is.function(sl.fun))
            sl.fun <- eval(parse(text = paste("function(...){" ,
            sl.fun, "}")
            )))
        tkconfigure(sc, command = sl.fun)
    }
    assign("slider.values.old", sl.defaults, env = slider.env)
    tkpack(f.but <- tkframe(nt), fill = "x")
    tkpack(tkbutton(f.but, text = "Exit", command = function() tkdestroy(nt)),
        side = "right")
    if (missing(reset.function))
        reset.function <- function(...) print("relax")
    if (!is.function(reset.function))
        reset.function <- eval(parse(text = paste("function(...){" ,
            reset.function, "}")
            )))
    tkpack(tkbutton(f.but, text = "Reset", command = function() {
        for (i in seq(sl.names)) eval(parse(text = paste("tclvalue(slider",
            i, "<-", sl.defaults[i], sep = "))), env = slider.env)
        reset.function()
    }), side = "right")
    if (missing(but.names))
        but.names <- NULL

```

```

for (i in seq(but.names)) {
  but.fun <- if (length(but.functions) > 1)
    but.functions[[i]]
  else but.functions
  if (!is.function(but.fun))
    but.fun <- eval(parse(text = paste("function(...){" ,
    but.fun, " }")))
  tkpack(tkbutton(f.but, text = but.names[i], command = but.fun),
    side = "left")
}
invisible(nt)
}

```

(ein fiktiver Einsatz von select.limits als Test 10)