

# Klausurenbenotungsunterstützung

— Verarbeitung einer csv-Datei —,  
File: kpneu.rev  
in: /home/wiwi/pwolf/projekte/noten

Version: 4.3.2008

## Inhalt

<b>1</b>	<b>Einleitung</b>	<b>2</b>
<b>2</b>	<b>Struktur der Lösung</b>	<b>2</b>
2.1	Vorgehen . . . . .	2
2.2	Input und Output . . . . .	2
2.3	Fazit . . . . .	5
<b>3</b>	<b>Festsetzung der Notengrenzen</b>	<b>5</b>
<b>4</b>	<b>Umsetzung</b>	<b>8</b>
4.1	Die Auswirkung der Veränderung von $\alpha$ . . . . .	8
4.2	Die Auswirkung der Veränderung der Bestehensgrenze . . . . .	9
4.3	Die Erstellung der Ausgabegraphik . . . . .	9
4.4	Die Hauptfunktion: <code>select.limits</code> . . . . .	9
4.5	Ein kleiner Test . . . . .	10
<b>5</b>	<b>Restarbeiten</b>	<b>10</b>
5.1	Der Einleseprozess . . . . .	10
5.2	Die methodenabhängige Berechnung der Notengrenzen . . . . .	12
5.3	Die Elemente der Ausgabegraphik . . . . .	13
5.4	Die Ergebnisspeicherung . . . . .	14
5.5	<code>noten.fein</code> und <code>noten.grob</code> . . . . .	16
<b>6</b>	<b>Anhang: <code>quantile_stetig</code></b>	<b>17</b>
6.1	Erstellung einer Punkte-Demo-Datei . . . . .	17

6.2 Die Definition der Sliderfunktion . . . . .	17
6.3 Verarbeitung zum .R-File. . . . .	18

## 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 datengetrieben Grenzen von Notenbereichen finden lassen. Der Ansatz setzt folgende Regeln um:

1. Eine Klausur ist bestanden, wenn sie mindestens 50% der Punkte besitzt, die gerade noch 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  $\alpha$  der Klausuren mit der Note *Eins* angesehen werden, der eine Vergleichbarkeit verschiedener Klausuren ermöglicht.

## 2 Struktur der Lösung

### 2.1 Vorgehen

Der hier präsentierte Vorschlag arbeitet in vier Schritten:

1. Beschaffung der leeren Notenliste über Internet
2. Ergänzung der realisierten Punkteanzahlen
3. Interaktive Festsetzung der Steuerungsgröße  $\alpha$
4. Erstellung einer Ergebnisliste und eines mit Graphiken versehenen Notenspiegels

### 2.2 Input und Output

Für diese Werkzeug ist zur Zeit notwendig, dass eine csv-Datei als Input vorliegt, die zum Beispiel aus einer xls-Datei durch Speicherung "als csv" entstanden ist. In dieser müssen zeilenweise die Daten der Prüfungsteilnehmer aufgeführt sein. Teilnehmereinträge ohne 7-stellige Matrikel-Nummer werden nicht erkannt! Diese csv-Datei muss für den hier beschriebenen Vorschlag gegenüber der automatisch generierten Vorlage um eine Spalte erweitert werden, in die die erreichten Punkte stehen. Mit Punkten wird die Liste dann etwa so aussehen:

INPUT:

UNIVERSITAET BIELEFELD  
Fakultaet fuer Wirtschaftswissenschaften  
- Pruefungsamt -  
Notenliste  
(Notenerfassung)

Titel der Veranstaltung: Algorithmen und Datenstrukturen  
Beleg-Nr.: 314104  
Semester: WS 07/08  
Pruefer: Wolf  
Datum der Pruefung: 14.02.08  
Termin: 1. Termin

lfd.-Nr.	Nachname	Vorname	Matr.Nr.	Pkte	LP	Note	Bemerkungen
1	Becker	Franz	1666666	32			
2	Bartlett	Nobert	1707754	34			
3	Boller	Daniel	1843833	50			

...

Datum: Unterschrift der Prueferin / des Pruefers:

---

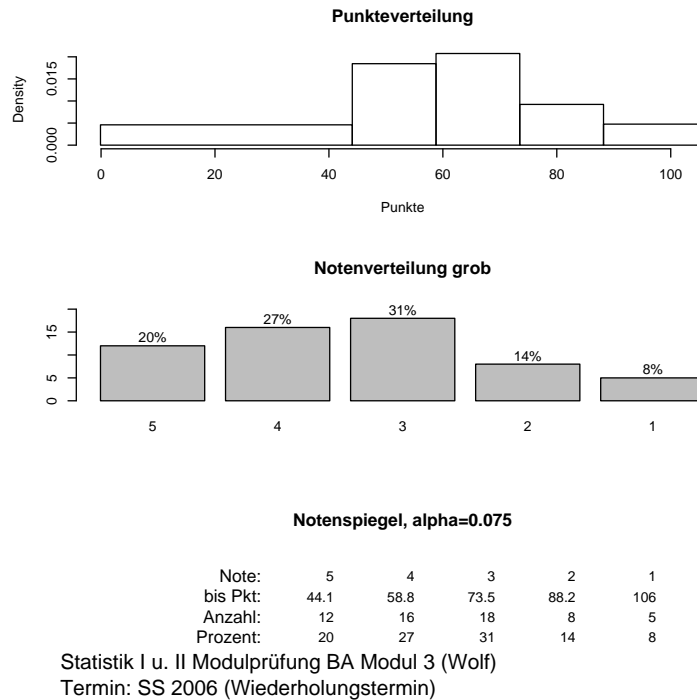
Note	Anzahl	/	%	Klausureinsicht	
1					
2				Datum:	
3					Uhrzeit:
4				Ort:	
5					

Die neue Liste ist nun als csv-Datei abzuspeichern! Das Innere dieser csv-Datei muss dann folgende Gestalt haben:

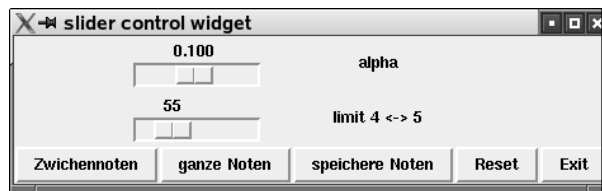
```
UNIVERSITAET BIELEFELD;;;;;
Fakultaet fuer Wirtschaftswissenschaften;;;;;
- Pruefungsamt -;;;;;
;;;Notenliste;;;
;;;(Notenerfassung);;;
;;;;;
Titel der Veranstaltung;;;Algorithmen und Datenstrukturen;;;
Beleg-Nr.;;;314104;;;;;
Semester;;;WS 07/08;;;;;
Pruefer;;;Wolf;;;;;
Datum der Pruefung;;;14.02.2008;;;;;
Termin;;;1. Termin;;;;;
;;;;;
lfd.-Nr.;Nachname;Vorname;Matr.Nr.;Pkte;LP;Note;Bemerkungen
1;Becker;Franz;1666666;32;;;
2;Bartlett;Nobert;1707754;34;;;
3;Boller;Daniel;1843833;50;;;
...
;;;;;
Datum;;;Unterschrift der Prueferin / des Pruefers;;;;;
;;;;;
_____;;;;;
;;;;;
Note;Anzahl / %;Klausureinsicht;;;
1;;;;;
2;;;Datum:;;;
3;;;Uhrzeit:;;;
4;;;Ort:;;;
5;;;;;
```



graphische Darstellung zu dem jeweiligen  $\alpha$ -Wert wird ebenfalls als Datei gespeichert. Hier ein Beispiel:



Die Werte für  $\alpha$  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 Punkteanzahlen 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	—	0.0	0/30
4.3	—	0.3	3/30
4.0	0.3	0.6	6/30
3.7	0.3	1.0	10/30
3.3	0.4	1.3	13/30
3.0	0.3	1.6	16/30
2.7	0.3	2.0	20/30
2.3	0.4	2.3	23/30
2.0	0.3	2.6	26/30
1.7	0.3	3.0	30/30
1.3	0.4	3.3	33/30
1.0	0.3		

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  $\alpha$  modellieren. Ist  $\alpha$  vorgegeben, ergibt sich die zugehörige Punkteanzahl als  $(1 - \alpha)$ -Quantil. Um Eindeutigkeit für beliebige  $\alpha$ -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

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

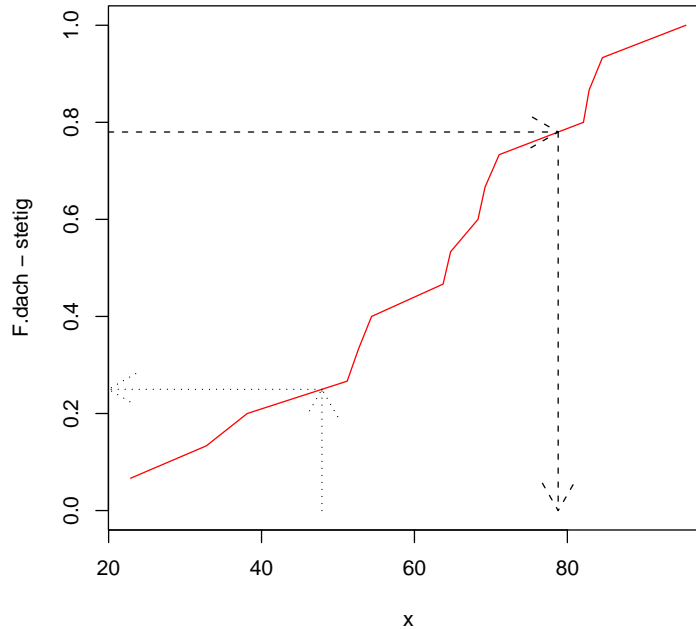
Diese ist implementiert in `quantile(1-alpha, x, type=4)`, und wir legen fest:

```
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. Wer dieses beleben möchte, entferne bitte das Kommentarzeichen in folgendem Chunk:

```
5 <demonstriere Zugriff auf F-hat 5> ≡
  set.seed(13);punkte<-sort(rnorm(15,60,20))
  <definiere quantile_stetig 31>
  ## 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 $\alpha$

Zur Umsetzung der beschriebenen Ideen definieren wir eine Funktion, die den Effekt eines neu gesetzten  $\alpha$ -Wertes umsetzt. Als Folge ist die zum neuen  $\alpha$  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 19>
    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  $\alpha$  neu berechnet werden.

```
7 <definiere effect.limit 7> ≡
  effect.limit<-function(...){
    limit.1.2<-2*(limit.4.5<-slider(no=2))
    <berechne Notengrenzen 19>
    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 20>
  <zeichne Noten-Verteilung 21>
  <drucke Notenspiegel in die Graphik 22>
  "ok"
```

## 4.4 Die Hauptfunktion: select.limits

Die zentrale Funktion wollen wir `select.limits` nennen. Mit ihr können interaktiv ein  $\alpha$  oder die Bestehensgrenze gewählt und die Auswirkungen studiert werden. Es ist übrigens das Einlesen aus einer Datei noch zu ergänzen. Neuerungen Mai 2007: vergrößerte Obergrenze für 4-5-Grenze, Schrittweite für Punkte: 0.5 statt 1, Auch fehlerhafte Mat-Nr aus 6 Ziffern werden verarbeitet, falls keine Prüfungsamt-Muster eingelesen wird, wird ein leerer Kopf ergänzt.

```
9 <definiere select.limits 9> ≡
  select.limits<-function(DEBUG=FALSE) {
    <Filename ermitteln und auf fname ablegen 12>
    <Trennzeichen in Datei fname ermitteln und auf sep ablegen 13>
    <Kopfinfos, Tabellenbereich und Fußbereich trennen 14>
    <Punkte extrahieren 15>
    <Kopfzeile finden oder erstellen 16>
    <Anzahl der Leistungspunkte erfragen 17>
    <speichere wichtige Infos 18>
    # Definition des Operationsfensters
    x<-sort(x); n<-length(x)
```

```

    <definiere effect.alpha 6>
    <definiere effect.limit 7>
    <definiere speichere.noten 23>
    <definiere noten.fein und noten.grob 30>
    slider(obj.name="slider1",obj.value=0)
    slider(obj.name="slider2",obj.value=0)
    slider(c(effect.alpha,effect.limit),
           c("alpha","limit 4 <-> 5"),
           c(0.001,round(.5*quantile(x,.6))), # MIN
#####   c(0.20,round(.5*max(x))),
           c(0.25,round(.7*max(x))), # MAX
#####   c(0.005,1 ),c(0.1,round(.3*max(x))),
           c(0.005,.5),c(0.1,round(.3*max(x))), # STEP, DEFAULT,
           c(noten.fein,noten.grob,speichere.noten),
           c("Zwischennoten","ganze Noten","speichere Noten")
    ); "ok"
  }
  select.limits() # zum Benutzen
  # select.limits(TRUE) # zum Debuggen

```

Zur Erstellung der "BATCH"-Version kann folgender Chunk gestartet werden:  
Für das PA die Version ist weiter unten zu finden.

```

10 <* 10> ≡
    <tangle select.limits 34>

```

## 4.5 Ein kleiner Test

Hier folgt ein Beispiel-Einsatz.

```

11 <ein fiktiver Einsatz von select.limits als Test 11> ≡
    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 Der Einleseprozess

Es gilt für den Einleseprozess verschiedene verwendete Chunks zu definieren.

```

12 <Filename ermitteln und auf fname ablegen 12> ≡
    fname<-tclvalue(tkgetOpenFile(title="Datei mit den Klausurpunkten?"))
    #if(!file.exists(fname)){cat("Datei nicht gefunden\n"); return()}

13 <Trennzeichen in Datei fname ermitteln und auf sep ablegen 13> ≡
    a<-c(scan(fname,what=" ",sep=" "),",",",",";")

```

```
sep<-c(",",";")[1+(length(grep(",",a))<length(grep(";",a)))]
```

- 14 *⟨Kopffinfos, Tabellenbereich und Fußbereich trennen 14⟩ ≡*
- ```
a<-readLines(fname); a<-gsub("\\\\",",",a)
n<-length(a)
# infos als Matrix darstellen
al<-strsplit(a,sep)
max.spa<-max(unlist(lapply(al,length)))
al<-lapply(al,function(x){
  if(length(x)<max.spa) x<-c(x,rep(" ",max.spa-length(x)))
  x[x==""]<-" "; x
})
roh.mat<-matrix(unlist(al),n,max.spa,byrow=TRUE)
# Matrikel-Nummer-Spalte finden
matnr.cand<-(lapply(1:max.spa,function(i) (grep("[0-9]{7}",roh.mat[,i]))))
# print(matnr.cand)
matnr.col<-which.max(unlist(lapply(matnr.cand,length)))
matnr.cand<-matnr.cand[[matnr.col]]
# Kopf finden
if(matnr.cand[1]>1) kopf<-roh.mat[1:(matnr.cand[1]-1),,drop=FALSE] else kopf <-
if(dim(kopf)[1]==1) kopf<-rbind(" ",kopf)
# Rumpf finden
rumpf.mat<-roh.mat[matnr.cand,]
# Fuss finden
if(max(matnr.cand)<n) fuss<-roh.mat[(max(matnr.cand)+1):n,,drop=FALSE] else fuss
```
- 15 *⟨Punkte extrahieren 15⟩ ≡*
- ```
n.studs<-dim(rumpf.mat)[1]
pkt.col<-lapply((1+matnr.col):max.spa,function(i) (grep("[0-9]",rumpf.mat[,i])))
pkt.col<-matnr.col+which.max(unlist(lapply(pkt.col,length)))

x<-rumpf.mat[,pkt.col]
pkte.orig<-x<-as.numeric(sub(",",".",gsub("\\\\",",",x)))
Wenn eine Notenspalte nicht da ist, wird sie angehängt!
```
- 16 *⟨Kopfzeile finden oder erstellen 16⟩ ≡*
- ```
# Kopfzeile finden
kopfzeile<-kopf[dim(kopf)[1],]
if("Note" %in% kopfzeile){
  noten.col<-which("Note"==kopfzeile)
}else{
  rumpf.mat<-cbind(rumpf.mat," "); kopf<-cbind(kopf," "); fuss<-cbind(fuss," ")
  noten.col<-max.spa<-(max.spa+1)
  kopfzeile<-rep(" ",max.spa); kopfzeile[matnr.col]<-"Matr.Nr.";
  kopfzeile[pkt.col]<-"Pkte"; kopfzeile[noten.col]<-"Note"
  kopf<-rbind(kopf,kopfzeile)
}
if(exists("DEBUG")&&DEBUG){
  cat("Kopf:\n"); print(kopf)
  cat("Kopfzeile:\n"); print(kopfzeile)
  cat("Rumpf:\n"); print(rumpf.mat)
  cat("Fuss:\n"); print(fuss)
  cat("eingelezene Punkte:\n"); print(x)
```

```
}
```

Das Prüfungsamt möchte eine Liste, in der Leistungspunkte angegeben sind.  
Dazu müssen die Leistungspunkte bekannt sein. Hier werden sie erfragt.

```
17 <Anzahl der Leistungspunkte erfragen 17> ≡
  LPS<-readline("Wie viele Leistungspunkte kann die Veranstaltung erbringen? ")
  LPS<-gsub("[^1-9]", "", LPS)
  if(nchar(LPS)!=1) return(paste("FEHLER: ->", LPS,
                                "Leistungspunkte unbekannt!\n Bitte noch einmal starten!"))
```

```
18 <speichere wichtige Infos 18> ≡
  # Speicherung zentraler Infos
  slider(obj.name="fname", obj.value=fname)
  slider(obj.name="kopf", obj.value=kopf)
  slider(obj.name="kopfzeile", obj.value=kopfzeile)
  slider(obj.name="rumpf.mat", obj.value=rumpf.mat)
  slider(obj.name="fuss", obj.value=fuss)
  slider(obj.name="pkte.orig", obj.value=pkte.orig)
  slider(obj.name="noten.col", obj.value=noten.col)
  slider(obj.name="pkt.col", obj.value=pkt.col)
  slider(obj.name="matnr.col", obj.value=matnr.col)
  slider(obj.name="roh.mat", obj.value=roh.mat)
  slider(obj.name="method", obj.value="grob") # default Methode
  slider(obj.name="sep", obj.value=sep)
  slider(obj.name="LPS", obj.value=LPS)
```

## 5.2 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. Übrigens werden die ermittelten Grenzen aus Praktikabilitätsgründen auf eine Nachkommastelle gerundet. Neuerung Mai 2007: Als maximale Obergrenze wird das Maximum aus den Grenzen und der maximalen Punkteanzahlen +1 verwendet, damit nicht die 1.3-1-Grenze größer als die 1.0-Grenze sein kann.

```
19 <berechne Notengrenzen 19> ≡
  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(max(x),max(limits))+1)
  limits<-round(limits,1)
```

```

slider(obj.name="limits",obj.value=limits)
if(DEBUG) cat("method:",method,"limits:",limits,"\n")
slider(obj.name="limits",obj.value=limits)
slider(obj.name="names.noten",obj.value=names.noten)

```

### 5.3 Die Elemente der Ausgabegraphik

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

20 *<zeichne Histogramm der Punkte zu Notengrenzen 20> ≡*

```

counts<-hist(x,breaks=limits,prob=TRUE,xlab="Punkte",
             main="Punkteverteilung")$counts
slider(obj.name="counts",obj.value=counts)

```

21 *<zeichne Noten-Verteilung 21> ≡*

```

mp<-barplot(counts,names.arg=names.noten,
            ylim=c(0,1.2*max(counts)))
text(mp,count+.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. Unten werden in der Graphik einige technische Daten zu der Veranstaltung eingetragen. Neuerung Mai 2007: Angabe von  $\alpha$  mit 3 Nachkommastellen.

22 *<drucke Notenspiegel in die Graphik 22> ≡*

```

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,3),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)
## print(limits) ####
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)

kopf<-slider(obj.name="kopf")
info<-rbind("-",kopf[1,])
key<-c("Titel","Beleg-Nr","Semester","Datum:")
for(was in key){
  if(DEBUG){cat("was in key:\n");print(was)}
  if(0<length(h<-grep(was,kopf)))info<-rbind(info,kopf[h,])
}
if(DEBUG){cat("INFO:");print(info)}
info<-info[,unlist(lapply(1:dim(kopf)[2],function(j) any(nchar(info[,j])>1))),dr

```

```

if(2<=dim(info)[2]){
  info<-paste(paste(info[,1],info[,2]),collapse="\n")
  text(-1.6,-6,info,cex=1.0,pos=4,xpd=TRUE)
}

```

## 5.4 Die Ergebnisspeicherung

Die Ergebnisabspeicherung ist in grober Form ist einfach und muss nach den genauen Anforderungen umgesetzt werden.

Die Funktion `speichere.noten` sucht sich die eingestellten Infos zusammen und erstellt den ersehnten Output.

```

23 <definiere speichere.noten 23> ≡
speichere.noten<-function(...){
  cat("Speicherung beginnt\n")
  <trage Noten in rumpf.matein 24>
  <setze Liste out.csv zusammen 25>
  <erstelle txt-Liste mit LPs 27>
  <erstelle txt-Liste mit Punkten 28>
  <speichere Listen und Bild 26>
  "ok"
}

```

Nun werden die Rohlisten mit Punkten und Noten für den Ausdruck erstellt und in der Matrix `rumpf.mat` abgelegt.

```

24 <trage Noten in rumpf.matein 24> ≡
fname<-slider(obj.name="fname")
# Liste und Punkte holen
kopf<- slider(obj.name="kopf")
kopfzeile<-slider(obj.name="kopfzeile")
rumpf.mat<-slider(obj.name="rumpf.mat")
fuss<- slider(obj.name="fuss")
pkte.orig<-slider(obj.name="pkte.orig")
noten.col<-slider(obj.name="noten.col")
pkt.col<- slider(obj.name="pkt.col")
matnr.col<-slider(obj.name="matnr.col")
roh.mat<- slider(obj.name="roh.mat")
method<- slider(obj.name="method")
sep<- slider(obj.name="sep")
limits<- slider(obj.name="limits")
LPs<- slider(obj.name="LPs")
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
LPs.vec<-noten.vec<-rep(NA,length(pkte.orig))
for(i in seq(pkte.orig)){
  if(!is.na(pkte.orig[i])){
    n<-noten[sum(pkte.orig[i]>limits)]
    noten.vec[i]<-n
  }
}

```

```

    LPs.vec[i]<-c(0,LPs)[1+ (n<4.9)]
  }else{
    LPs.vec[i]<-noten.vec[i]<-"--"
  }
}
rumpf.mat[,noten.col]<-noten.vec
rumpf.matLP<-rumpf.mat
rumpf.matLP[,pkt.col]<-LPs.vec

```

In diesem Chunk wird ein Kopf an die Rohlisten für die zu erstellende csv-Datei angehängt, dem die technischen Details zu der Veranstaltung entnommen werden können. Das Ergebnis wird auf `out.csv` abgelegt, denn es wird keine neue Sortierung der Liste vorgenommen.

```

25 <setze Liste out.csv zusammen 25> ≡
kopfzeile<-slider(obj.name="kopfzeile")
kopf<-kopfLP<-slider(obj.name="kopf")
if(DEBUG) {cat("Kopf:\n"); print(kopfLP)}
zz<-dim(kopfLP)[1]
kopfLP[zz,]<-sub("LP", " ",kopfLP[zz,])
kopfLP[zz,]<-sub("Pkte", "LP",kopfLP[zz,])
kopfLP[zz,]<-sub("Punkte", "LP",kopfLP[zz,])
n.sp<-dim(rumpf.mat)[2]
if(dim(fuss)[2]!=n.sp){
  fuss<-cbind(fuss,matrix(" ",dim(fuss)[1],n.sp))[1:n.sp]
}
if(dim(kopfLP)[2]!=n.sp){
  kopfLP<-cbind(kopfLP,matrix(" ",zz,n.sp))[1:n.sp]
  kopf <-cbind(kopf ,matrix(" ",zz,n.sp))[1:n.sp]
}
if(DEBUG) {
  cat("dim(kopfLP)",dim(kopfLP))
  cat("dim(rumpf.mat)",dim(rumpf.mat))
  cat("dim(rumpf.matLP)",dim(rumpf.matLP))
  cat("dim(fuss)",dim(fuss))
}
out.csv<-rbind(kopfLP,rumpf.matLP,fuss)
out.csv<-as.data.frame(out.csv)

```

Es werden vier Dateien erstellt. Diese sollten wohl genügen.

```

26 <speichere Listen und Bild 26> ≡
# Dateinamen abfragen
fname<-paste(sub("\\.csv$", "", fname), "-noten.csv", sep=" ")
outfile<-tclvalue(tkgetSaveFile(initialfile=fname))
if(" "==outfile){ cat("nichts gespeichert!\n"); return() }
if(0==length(grep("noten.csv$", outfile)))
  outfile<-paste(outfile, "noten.csv", sep=" ")

# Speicherung der csv-Datei
write.table(out.csv,file=outfile,sep=sep,
            row.names=FALSE,col.names=FALSE,quote=FALSE)
cat("Ergebnisdatei",outfile,"erstellt!\n")

# Speicherung der Graphik

```

```

outfile<-sub("noten.csv$", "graphik.pdf", outfile)
try({dev.copy(pdf, outfile); dev.off()})
cat("ErgebnisGraphik:", outfile, "erstellt!\n")

# Erstellung der PA-Notenliste
outfile<-sub("graphik.pdf$", "noten.txt", outfile)
cat(out.txtLP, file=outfile, sep="\n")
cat("Ergebnisdatei", outfile, "erstellt!\n")

# Erstellung der Notenliste mit Punkten
outfile<-sub("noten.txt$", "pkt-noten.txt", outfile)
cat(out.txt, file=outfile, sep="\n")
cat("Ergebnisdatei - mit Punkten:", outfile, "erstellt!\n")

```

Für die ausgedruckte Liste muss der Kopf ein wenig aufgearbeitet werden. Das Design lehnt sich an alte Notenübersichten an. Das Prüfungsamt ist nicht an Punkten interessiert. Deshalb wird eine Sparversion mit LPs statt mit Klausurpunkten erstellt.

```

27 <erstelle txt-Liste mit LPs 27> ≡
add.space<-function(mat, sep=" ", size){
  J<-ncol(mat)
  I<-nrow(mat)
  mat<-gsub("\\ +$", "", mat)
  max.char<-unlist(lapply(1:J, function(j) max(nchar(mat[, j]))))
  leer<-paste(rep(" ", max(max.char)), collapse="")
  for(j in 1:J){
    mat[, j]<-paste(mat[, j], substring(leer, 1, max.char[j]-nchar(mat[, j])))
  }
  mat
}
txt.kopf<-add.space(kopfLP[-dim(kopfLP)[1], , drop=FALSE])
txt.kopf<-paste(t(cbind(txt.kopf, "\n")), collapse="")
txt.kopf<-unlist(strsplit(txt.kopf, "\n"))
txt.kopf<-sub(" ", "", txt.kopf)

txt.fuss<-add.space(fuss)
txt.fuss<-paste(t(cbind(txt.fuss, "\n")), collapse="")
txt.fuss<-unlist(strsplit(txt.fuss, "\n"))

kopfzeileLP<-sub("LP", " ", kopfzeile)
kopfzeileLP<-sub("Pkte", " LP", kopfzeileLP)
kopfzeileLP<-sub("Punkte", " LP", kopfzeileLP)

txt.rumpf<-rbind(kopfzeileLP, rumpf.matLP)
txt.rumpf<-add.space(txt.rumpf)
txt.rumpf<-paste(t(cbind(txt.rumpf, "\n")), collapse="")
txt.rumpf<-unlist(strsplit(txt.rumpf, "\n"))
txt.rumpf<-cbind(txt.rumpf, paste(rep("-", nchar(txt.rumpf[1])), collapse=""))
out.txtLP<-c(txt.kopf, as.vector(t(txt.rumpf)), txt.fuss)

```

Für die eigenen Unterlagen mag jedoch eine Notenliste mit Klausurpunkten interessanter sein. Deshalb wird auch diese generiert. Einige der Vorarbeiten aus dem letzten Chunk müssen nicht mehr wiederholt werden. Andere sind im

Prinzip kopiert.

```
28 <erstelle txt-Liste mit Punkten 28> ≡
txt.kopf<-add.space(kopf[-dim(kopf)[1],,drop=FALSE])
txt.kopf<-paste(t(cbind(txt.kopf,"\n")),collapse="")
txt.kopf<-unlist(strsplit(txt.kopf,"\n"))
txt.kopf<-sub("          ", "", txt.kopf)

txt.rumpf<-rbind(kopfzeile,rumpf.mat)
txt.rumpf<-add.space(txt.rumpf)
txt.rumpf<-paste(t(cbind(txt.rumpf,"\n")),collapse="")
txt.rumpf<-unlist(strsplit(txt.rumpf,"\n"))
txt.rumpf<-cbind(txt.rumpf,paste(rep("-",nchar(txt.rumpf[1])),collapse=""))
<erstelle Spiegel 29>
out.txt<-c(txt.kopf,as.vector(t(txt.rumpf)),spiegel,txt.fuss)
```

Für den eigenen Überblick ist ein Notenspiegel erwünscht. Den wollen wir für den Text-Ausdruck mit den Punkten auch noch eintragen.

```
29 <erstelle Spiegel 29> ≡
limits<-slider(obj.name="limits")
names.noten<-slider(obj.name="names.noten")
counts<-slider(obj.name="counts")
#spiegel<-c("Notenspiegel:",
spiegel<-rbind(c("Note:", "bis Punkte:", "Anzahl", "Prozent"),
               cbind(names.noten,round(limits[-1],1),counts,
                       round(100*counts/sum(counts))))
spiegel<-add.space(spiegel)
spiegel<-paste(t(cbind(spiegel,"\n")),collapse="")
spiegel<-c("\nNotenspiegel:\n",unlist(strsplit(spiegel,"\n")))
```

## 5.5 noten.fein **und** noten.grob

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

```
30 <definiere noten.fein und noten.grob 30> ≡
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.

```

31 (definiere quantile_stetig31) ≡
quantile_stetig<-function(x,digits=2,qtype=4){
  x<-sort(x); n<-length(x)
  redo<-function(...){
    alpha<-slider(no=1)
    qtype<-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=qtype)
    arrows(xmin,1-alpha,y.0,1-alpha,pty=2)
    arrows(y.0,1-alpha,y.0,0,pty=2)
    arrows(x0,gamma0,xmin,gamma0,pty=3)
    arrows(x0,0,x0,gamma0,pty=3)
    tit<-paste("1-",alpha,"=","1-alpha","-Quantil (Typ ",
              qtype,"):" ,round(y.0,digits),"\",
              "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,qtype,.5*min(x),max(x))); "ok"
}
## quantile_stetig(rnorm(10))

```

## 6.1 Erstellung einer Punkte-Demo-Datei

```

32 (konstruiere Demo-Datei mit Punkten 32) ≡
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,sep,matnr,sep,punkte,"\",sep="")
cat(file="klpkte.csv",sep="",out)

```

## 6.2 Die Definition der Sliderfunktion

Für alle Fälle wird die Funktion slider bereitgestellt.

```

33 (start 33) ≡
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 <- tkoplevel()
  tkwm.title(nt, title)
  tkwm.geometry(nt, "+0+50")
  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(){par(mfrow=c(1,1));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(sl原因",
    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)
}
<i>(definiere select.limits 9)</i>

```

## 6.3 Verarbeitung zum .R-File.

Vor der tangle-Verarbeitung muss diese Datei gespeichert werden!!

```

34 <i>(tangle select.limits 34) ≡
##tangleR("klausurpunktegrenzen", expand.roots=""")
##file.copy("klausurpunktegrenzen.R", "noten-pa.R", overwrite=TRUE)
tangleR("kpneu", "noten.R", "definiere [[select.limits]]")
file.copy("noten.R", "noten-ekvv.R", overwrite=TRUE)

```