Klausurenbenotungsunterstützung

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

Version: 22.10.2010

Inhalt

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 α 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 α
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;;;;;;;
```

Dann wird die textuelle Ergebnisdatei folgende Form aufweisen:

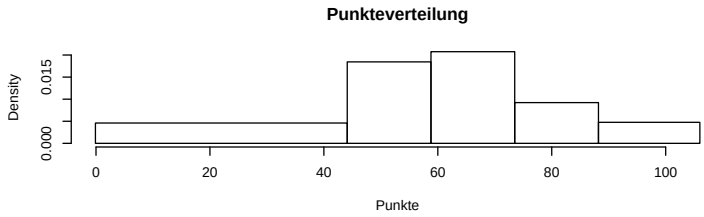
```
UNIVERSITAET BIELEFELD
Fakultaet fuer Wirtschaftswissenschaften
- Pruefungsamt -
                                                                        Notenliste
Titel der Veranstaltung:               Algorithmen und Datenstrukturen
Beleg-Nr.:                             314104
Semester:                              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      4
-------------------------------------------------------------------------
2        Bartlett     Norbert       1707754  34      3
-------------------------------------------------------------------------
3        Boller       Daniel        1843833  50      2
  ...
-------------------------------------------------------------------------

Datum:                           Unterschrift der Prueferin / des Pruefers:

_________________

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

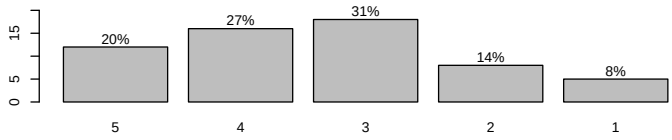
Und so sieht die neue csv-Datei aus:

```
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;;4;
2;Bartlett;Nobert;1707754;34;;3;
3;Boller;Daniel;1843833;50;;2;
 ; ; ; ; ; ; ;
Datum:; ; ;Unterschrift der Prueferin / des Pruefers:; ; ; ;
 ; ; ; ; ; ; ;
_________________; ; ; ; ; ; ;
 ; ; ; ; ; ; ;
Note;Anzahl  /  %; ;Klausureinsicht; ; ; ;
1; ; ; ; ; ; ;
2; ; ;Datum:; ; ; ;
3; ; ;Uhrzeit:; ; ; ;
```

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 Beispeil:



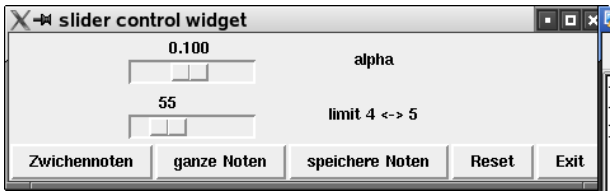


Notenspiegel, alpha=0.075

Note:	5	4	3	2	1
bis Pkt:	44.1	58.8	73.5	88.2	106
Anzahl:	12	16	18	8	5
Prozent:	20	27	31	14	8

Statistik I u. II Modulprüfung BA Modul 3 (Wolf)
Termin: SS 2006 (Wiederholungstermin)

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∗`limit.1.2`	4
4/6∗`limit.1.2`	3
5/6∗`limit.1.2`	2
6/6∗`limit.1.2`	1

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

1 ⟨*berechne grobe Grenzen* 1⟩ ≡ ⊂ 12, 14, 22
```
limit.4.5<-0.5*limit.1.2
limits.grob<-limit.1.2*(3:6)/6
```

Die feinere Einteilung ist wesentlich komplizierter. Denn es gibt mehrere Lösungen, für die sich eine Begründung finden lässt. Wir gehen nun davon aus, dass die Punktedifferenz zwischen zwei benachbarten Noten gerade `step` beträgt.

1. Die Schrittweiten zwischen zwei benachbarten Noten bei feinerer Einteilung sind unterschiedlich und betragen entweder 0.3 oder 0.4. Stellen wir uns eine Klausur vor, die die gerade notwendigen Punkte für eine 3.0 hat (schlechteste 3.0). Dann muss man für eine 2.7 mindesten `0.3 x step` mehr haben. Um sogar eine 2.3 zu bekommen, muss man weitere `0.4 x step` mehr an Punkten erarbeitet haben. Diese Argumentation wird dazu führen, dass die Punkte-Intervalllängen der Zensuren 3.0 und 2.7 sich wie 0.3 zu 0.4 verhalten. Zu alle .7-er Noten gehören so breitere Intervalle und es werden cum grano salis mehr .7-er Noten vorkommen als .3-er oder .0-er Noten.

2. Der Gedanke kann aber auch in umgekehrt formuliert werden. Betrachten wir dazu die beste 3.0-Punktezahl, dann ergibt sich die Punktezahl der besten 2.7-er Note durch Subtraktion von `0.3 x step`. Für die beste 2.3 finden wir die Punktezahl durch Subtraktion von weiteren `0.4 x step` Punkte. Nach dieser Argumentation wird es häufiger 0.3-er Noten als 0.7-er oder .0-er Noten geben.

3. Sehen wir die Note 3.0 als Mitte zwischen den Noten 3.3 und 2.7 an, dann sollten wir eine Symmetrie der Intervalllängen zu den Noten 3.3 und 2.7 fordern. Das können wir z.B. umsetzen, indem wir uns an den Grenzen der ganzen Note 3 orientieren und unterstellen, dass diese von der fiktiven Note 2.5 bis zur ebenfalls fiktiven Note 3.5 reicht. Interpretieren wir nun den Abbildungsprozesse auf die zulässigen Noten (3.3, 3.0 und 2.7) als Rundung zur nächsten zulässigen Note, dann ergeben sich die Noten-Intervalle, wie folgt: [3.5, 3.15] → 3.3, [3.15, 2.85] → 3.0 und [2.85, 2.5] → 2.7. Auf diese Weise kommen wir zu den Intervalllängen 0.35, 0.3, 0.35 und es ergeben sich glatte Noten seltener als die .3-er oder .4-er Noten.

4. Die Asymmetrie der ersten beiden Argumentationsketten gefällt nicht, auch die geringeren Häufigkeiten für ganze Noten ist nicht richtig schön. Außerdem erscheinen die Ansätze unnötig kompliziert zu sein. Deshalb soll noch eine vierte Idee vorgestellt werden, die diese Schwierigkeiten nicht hat: Nehmen wir an, es sind eigentlich Noten der Form 3 1/3, 3.0, 2 2/3 usw. erwünscht, aus Gründen der Einfachheit werden diese auf die erste Nachkommastelle gerundet und nur Noten wie 3.3, 3.0 oder 2.7 ausgewiesen. Dann betragen die Schritte zwischen benachbarten

Punktegrenzen für die intendierten Noten `1/3 x step`. Die Punkte-Grenzen sind nach dieser Idee leicht zu berechnen und zu vermitteln.

In der hier umgesetzten Lösung ist der vierte Ansatz gewählt worden. Es sei noch darauf hingewiesen, dass in der Regel nur ganze Punkte vergeben werden, so dass die Intervall-Grenzen in ihrer rechnerischen Genauigkeit nicht zum tragen kommen und sich auf ganze Punkte bezogen unterschiedliche Klassenbreiten einstellen werden. Weitergehende Forderung bzgl. ganzer Punkte würde wieder die Kompliziertheit erhöhen. Auch sei erwähnt, dass in einer früheren Version des Werkzeugs unterschiedliche Klassenbreiten umgesetzt wurden.

Ebenfalls weisen wir darauf hin, dass man auf http://www-sec.uni-regensburg.de/pnz/index.html.de eine nicht-äquivalente Lösung beschrieben findet. Dort wird übrigens auch ein Javascript-Mechanismus zur Grenzen-Ermittlung bereitgestellt.

Von der Note 1.0 bis 4.0 sind es drei ganze Noten. Zur Bestehensgrenze kommt man durch Halbierung der Grenzpunktezahl zur 1. Der Schritt zwischen zwei ganzen Noten folgt durch Drittelung des Abstands von der 1 zur 4 oder der Bestehenspunktezahl. Der Schritt zwischen zwei benachbarten feinen Noten ergibt sich durch eine weitere Drittelung.

Für die Berechnung der Grenzen zwischen den Noten müssen wir beachten, dass es eine 4.7 (gute 5) und eine 4.3 (schlechte 4) nicht gibt. Somit repräsentiert die feine Note 4.0 die feine Note 4.0 sowie die fiktive feine Note 4.3, so dass wir für 4.0 zwei feine Schritte ansetzen müssen. Wir setzen die Faktoren um, und berücksichtigen dabei, dass es die Note 4.3 laut PO nicht gibt.

2 ⟨ *berechne feine Grenzen* 2⟩ ≡ ⊂ 13, 14, 22

```
#limit.1.2<-120
limit.4.5 <- limit.1.2/2 # Punkte(Grenze zur 5) == Punkte(Grenze zur 1)/2
delta.4.1 <- limit.4.5   # Sprung zwischen 4 und 1 == Punkte(Grenze zur 5)
delta.grob <- (1/3) * delta.4.1  # Sprung zwischen zwei ganzen Noten
delta.fein <- (1/3) * delta.grob # Sprung zwischen zwei feinen Noten
limits.fein<-limit.4.5 +
             delta.fein * cumsum(c(0,2,   # Mindest-Pkt  4.0,3.7
                                   1,1,1, # Mindest-Pkt  3.3,3.0,2.7
                                   1,1,1, # Mindest-Pkt  2.3,2.0,1.7
                                   1,1))  # Mindest-Pkt  1.3,1.0
```

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⟩ ≡ ⊂ 7

```
x1<-max(x[x<=x0],-Inf);   x2<-min(x[x>x0],Inf)
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:

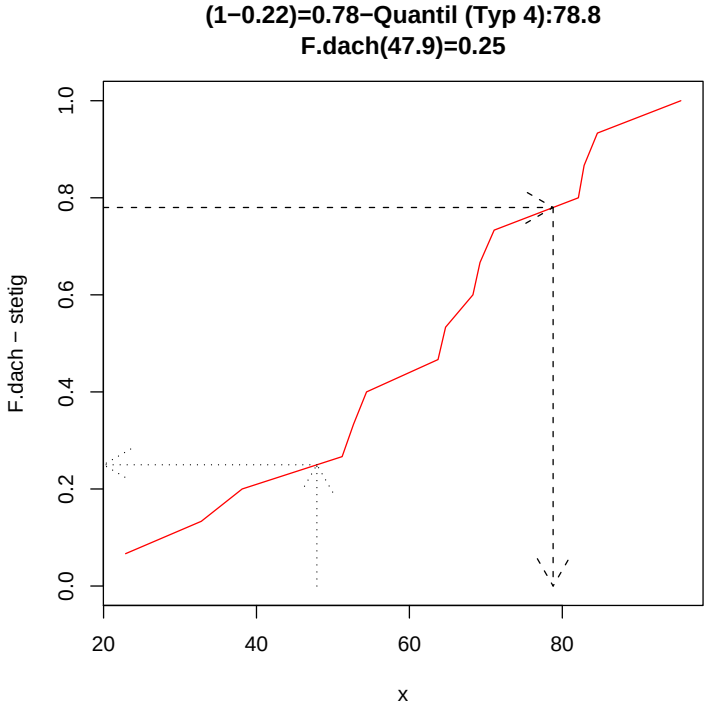
4 ⟨*berechne* `pkt.alpha` *zu vorgegebenem* `alpha` 4⟩ ≡ ⊂ 6

```
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* $\hat{F}$ 5⟩ ≡

```
set.seed(13);punkte<-sort(rnorm(15,60,20))
⟨definiere quantile_stetig 34⟩
## quantile_stetig(punkte)
```



Damit haben wir die wesenlichen 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⟩ ≡ ⊂ 9

```
effect.alpha<-function(...){
  alpha<-slider(no=1)
  ⟨berechne pkt.alpha zu vorgegebenem alpha 4⟩
  limit.1.2<-pkt.alpha
  ⟨berechne Notengrenzen 22⟩
  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⟩ ≡ ⊂ 9

```
effect.limit<-function(...){
  limit.1.2<-2*(limit.4.5<-slider(no=2))
  ⟨berechne Notengrenzen 22⟩
  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⟩ ≡ ⊂ 6, 7

```
par(mfrow=c(3,1))
⟨zeichne Histogramm der Punkte zu Notengrenzen 23⟩
⟨zeichne Noten-Verteilung 24⟩
⟨drucke Notenspiegel in die Graphik 25⟩
"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. 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⟩ ≡ ⊂ 11, 37

```
select.limits<-function(DEBUG=FALSE){
  ⟨Filename ermitteln und auf fname ablegen 15⟩
  ⟨Trennzeichen in Datei fname ermitteln und auf sep ablegen 16⟩
  ⟨Kopfinfos, Tabellenbereich und Fußbereich trennen 17⟩
  ⟨Punkte extrahieren 18⟩
  ⟨Kopfzeile finden oder erstellen 19⟩
  ⟨Anzahl der Leistungspunkte erfragen 20⟩
  ⟨speichere wichtige Infos 21⟩
# Definition des Operationsfensters
  x<-sort(x); n<-length(x)
  ⟨definiere effect.alpha 6⟩
  ⟨definiere effect.limit 7⟩
  ⟨definiere speichere.noten 26⟩
  ⟨definiere noten.fein und noten.grob 33⟩
  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(.25*quantile(x,.5))),  # MIN
####     c(0.20,round(.5*max(x))),
         c(0.30,round(.8*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 38⟩
```

4.5 Ein kleiner Test

Hier folgt ein Beispiel-Einsatz.

11 ⟨*ein fiktiver Einsatz von* `select.limits` *als Test* 11⟩ ≡

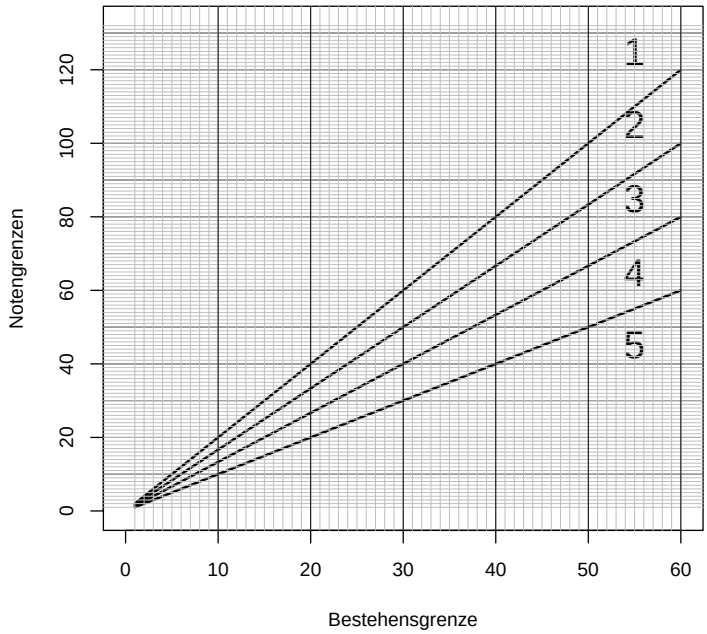
```
set.seed(17); punkte<-pmax(0,round(rnorm(230,70,30)))
matnr<-sample(2000:9000,230)
⟨definiere select.limits 9⟩
# select.limits(punkte)
# select.limits()
```

5 Punkte-Notengrenzen-Diagramme

Zur Übersicht mag es hilfreich sein, Übersichten für Punktegrenzen auf Basis der Bestehensgrenze oder aber der Einser-Grenze zu erstellen.

12 ⟨*erstelle Notengrenzen-Diagramm Noten grob* 12⟩ ≡

```
max.pkt.5.4<-60
plot(NULL,xlim=c(0,max.pkt.5.4),ylim=c(0,2.2*max.pkt.5.4),
     xlab="Bestehensgrenze",ylab="Notengrenzen")
names.noten.grob<-as.character(5:1)
res<-NULL
for(i in c(1:max.pkt.5.4)){
  ok.limit<-i
  limit.1.2<-2*(limit.4.5<-ok.limit) ###slider(no=2))
  ⟨berechne grobe Grenzen 1⟩
  limits<-limits.grob
  res<-cbind(res,limits)
}
anz<-dim(res)[2]
segments(1,res[,1],max.pkt.5.4,res[,anz],pch=16,cex=.5,lwd=2)
text(max.pkt.5.4-5,res[,anz]-15,names.noten.grob[-5],cex=2)
text(max.pkt.5.4-5,res[4,anz]+5,"1",cex=2)
abline(h=10*(1:floor(2.2*max.pkt.5.4/10)),lwd=0.3)
abline(h=(1:floor(2.2*max.pkt.5.4/1)),lwd=.3,col="gray")
abline(v=1:max.pkt.5.4,lwd=.3,col="gray")
abline(v=10*(1:floor(max.pkt.5.4/10)),lwd=.3)
res<-t(res)
colnames(res)<-paste(names.noten.grob[-5],"-",names.noten.grob[-1],sep="")
res
```



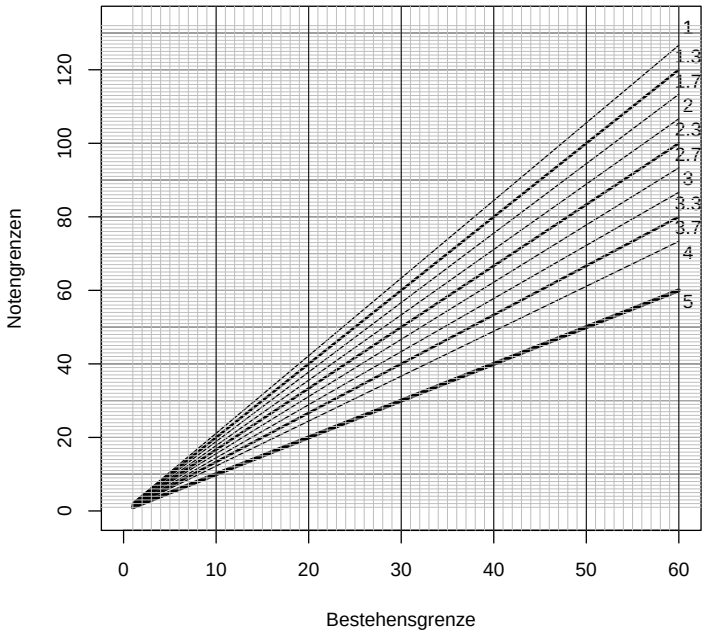
	5-4	4-3	3-2	2-1
limits	1	1.333333	1.666667	2
limits	2	2.666667	3.333333	4
limits	3	4.000000	5.000000	6
limits	4	5.333333	6.666667	8
limits	5	6.666667	8.333333	10
limits	6	8.000000	10.000000	12
limits	7	9.333333	11.666667	14
limits	8	10.666667	13.333333	16
limits	9	12.000000	15.000000	18
limits	10	13.333333	16.666667	20
limits	11	14.666667	18.333333	22
limits	12	16.000000	20.000000	24
limits	13	17.333333	21.666667	26
limits	14	18.666667	23.333333	28
limits	15	20.000000	25.000000	30
limits	16	21.333333	26.666667	32
limits	17	22.666667	28.333333	34
limits	18	24.000000	30.000000	36
limits	19	25.333333	31.666667	38
limits	20	26.666667	33.333333	40
limits	21	28.000000	35.000000	42
limits	22	29.333333	36.666667	44
limits	23	30.666667	38.333333	46
limits	24	32.000000	40.000000	48
limits	25	33.333333	41.666667	50
limits	26	34.666667	43.333333	52
limits	27	36.000000	45.000000	54
limits	28	37.333333	46.666667	56
limits	29	38.666667	48.333333	58
limits	30	40.000000	50.000000	60
limits	31	41.333333	51.666667	62
limits	32	42.666667	53.333333	64
limits	33	44.000000	55.000000	66
limits	34	45.333333	56.666667	68
limits	35	46.666667	58.333333	70
limits	36	48.000000	60.000000	72
limits	37	49.333333	61.666667	74
limits	38	50.666667	63.333333	76
limits	39	52.000000	65.000000	78
limits	40	53.333333	66.666667	80
limits	41	54.666667	68.333333	82
limits	42	56.000000	70.000000	84
limits	43	57.333333	71.666667	86
limits	44	58.666667	73.333333	88
limits	45	60.000000	75.000000	90
limits	46	61.333333	76.666667	92
limits	47	62.666667	78.333333	94
limits	48	64.000000	80.000000	96
limits	49	65.333333	81.666667	98
limits	50	66.666667	83.333333	100
limits	51	68.000000	85.000000	102
limits	52	69.333333	86.666667	104
limits	53	70.666667	88.333333	106
limits	54	72.000000	90.000000	108
limits	55	73.333333	91.666667	110
limits	56	74.666667	93.333333	112
limits	57	76.000000	95.000000	114

```
limits  58 77.333333  96.666667 116
limits  59 78.666667  98.333333 118
limits  60 80.000000 100.000000 120
```

Für die feinere Noteneinteilung können wir ein ähnliches Bild erzeugen.

13 ⟨*erstelle Notengrenzen-Diagramm Noten fein* 13⟩ ≡

```
x11()
max.pkt.5.4<-60
plot(NULL,xlim=c(0,max.pkt.5.4),ylim=c(0,2.2*max.pkt.5.4),
     xlab="Bestehensgrenze",ylab="Notengrenzen")
names.noten.fein<-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))
res<-NULL
for(i in c(1:max.pkt.5.4)){
  ok.limit<-i
  limit.1.2<-2*(limit.4.5<-ok.limit)
  ⟨berechne feine Grenzen 2⟩
  limits<-limits.fein
  res<-cbind(res,limits)
}
anz<-dim(res)[2]
segments(1,res[,1],max.pkt.5.4,res[,anz],pch=16,cex=.5,lwd=c(3,1,2,1,1,2,1,1,2,1))
text(max.pkt.5.4+1,res[,anz]-3,names.noten.fein[-11],cex=1)
text(max.pkt.5.4+1,res[10,anz]+5,"1",cex=1)
abline(h=10*(1:floor(2.2*max.pkt.5.4/10)),lwd=0.3)
abline(h=(1:floor(2.2*max.pkt.5.4/1)),lwd=.3,col="gray")
abline(v=1:max.pkt.5.4,lwd=.3,col="gray")
abline(v=10*(1:floor(max.pkt.5.4/10)),lwd=.3)
res<-t(res)
colnames(res)<-paste(names.noten.fein[-11],"-",names.noten.fein[-1],sep="")
rownames(res)<-NULL
noquote(format(res,digits=3))
```



	5-4	4-3.7	3.7-3.3	3.3-3	3-2.7	2.7-2.3	2.3-2	2-1.7	1.7-1.3	1.3-1
[1,]	1.00	1.22	1.33	1.44	1.56	1.67	1.78	1.89	2.00	2.11
[2,]	2.00	2.44	2.67	2.89	3.11	3.33	3.56	3.78	4.00	4.22
[3,]	3.00	3.67	4.00	4.33	4.67	5.00	5.33	5.67	6.00	6.33
[4,]	4.00	4.89	5.33	5.78	6.22	6.67	7.11	7.56	8.00	8.44
[5,]	5.00	6.11	6.67	7.22	7.78	8.33	8.89	9.44	10.00	10.56
[6,]	6.00	7.33	8.00	8.67	9.33	10.00	10.67	11.33	12.00	12.67
[7,]	7.00	8.56	9.33	10.11	10.89	11.67	12.44	13.22	14.00	14.78
[8,]	8.00	9.78	10.67	11.56	12.44	13.33	14.22	15.11	16.00	16.89
[9,]	9.00	11.00	12.00	13.00	14.00	15.00	16.00	17.00	18.00	19.00
[10,]	10.00	12.22	13.33	14.44	15.56	16.67	17.78	18.89	20.00	21.11
[11,]	11.00	13.44	14.67	15.89	17.11	18.33	19.56	20.78	22.00	23.22
[12,]	12.00	14.67	16.00	17.33	18.67	20.00	21.33	22.67	24.00	25.33
[13,]	13.00	15.89	17.33	18.78	20.22	21.67	23.11	24.56	26.00	27.44
[14,]	14.00	17.11	18.67	20.22	21.78	23.33	24.89	26.44	28.00	29.56
[15,]	15.00	18.33	20.00	21.67	23.33	25.00	26.67	28.33	30.00	31.67
[16,]	16.00	19.56	21.33	23.11	24.89	26.67	28.44	30.22	32.00	33.78
[17,]	17.00	20.78	22.67	24.56	26.44	28.33	30.22	32.11	34.00	35.89
[18,]	18.00	22.00	24.00	26.00	28.00	30.00	32.00	34.00	36.00	38.00
[19,]	19.00	23.22	25.33	27.44	29.56	31.67	33.78	35.89	38.00	40.11
[20,]	20.00	24.44	26.67	28.89	31.11	33.33	35.56	37.78	40.00	42.22
[21,]	21.00	25.67	28.00	30.33	32.67	35.00	37.33	39.67	42.00	44.33
[22,]	22.00	26.89	29.33	31.78	34.22	36.67	39.11	41.56	44.00	46.44
[23,]	23.00	28.11	30.67	33.22	35.78	38.33	40.89	43.44	46.00	48.56
[24,]	24.00	29.33	32.00	34.67	37.33	40.00	42.67	45.33	48.00	50.67
[25,]	25.00	30.56	33.33	36.11	38.89	41.67	44.44	47.22	50.00	52.78
[26,]	26.00	31.78	34.67	37.56	40.44	43.33	46.22	49.11	52.00	54.89
[27,]	27.00	33.00	36.00	39.00	42.00	45.00	48.00	51.00	54.00	57.00
[28,]	28.00	34.22	37.33	40.44	43.56	46.67	49.78	52.89	56.00	59.11
[29,]	29.00	35.44	38.67	41.89	45.11	48.33	51.56	54.78	58.00	61.22
[30,]	30.00	36.67	40.00	43.33	46.67	50.00	53.33	56.67	60.00	63.33
[31,]	31.00	37.89	41.33	44.78	48.22	51.67	55.11	58.56	62.00	65.44
[32,]	32.00	39.11	42.67	46.22	49.78	53.33	56.89	60.44	64.00	67.56
[33,]	33.00	40.33	44.00	47.67	51.33	55.00	58.67	62.33	66.00	69.67
[34,]	34.00	41.56	45.33	49.11	52.89	56.67	60.44	64.22	68.00	71.78
[35,]	35.00	42.78	46.67	50.56	54.44	58.33	62.22	66.11	70.00	73.89
[36,]	36.00	44.00	48.00	52.00	56.00	60.00	64.00	68.00	72.00	76.00
[37,]	37.00	45.22	49.33	53.44	57.56	61.67	65.78	69.89	74.00	78.11
[38,]	38.00	46.44	50.67	54.89	59.11	63.33	67.56	71.78	76.00	80.22
[39,]	39.00	47.67	52.00	56.33	60.67	65.00	69.33	73.67	78.00	82.33
[40,]	40.00	48.89	53.33	57.78	62.22	66.67	71.11	75.56	80.00	84.44
[41,]	41.00	50.11	54.67	59.22	63.78	68.33	72.89	77.44	82.00	86.56
[42,]	42.00	51.33	56.00	60.67	65.33	70.00	74.67	79.33	84.00	88.67
[43,]	43.00	52.56	57.33	62.11	66.89	71.67	76.44	81.22	86.00	90.78
[44,]	44.00	53.78	58.67	63.56	68.44	73.33	78.22	83.11	88.00	92.89
[45,]	45.00	55.00	60.00	65.00	70.00	75.00	80.00	85.00	90.00	95.00
[46,]	46.00	56.22	61.33	66.44	71.56	76.67	81.78	86.89	92.00	97.11
[47,]	47.00	57.44	62.67	67.89	73.11	78.33	83.56	88.78	94.00	99.22
[48,]	48.00	58.67	64.00	69.33	74.67	80.00	85.33	90.67	96.00	101.33
[49,]	49.00	59.89	65.33	70.78	76.22	81.67	87.11	92.56	98.00	103.44
[50,]	50.00	61.11	66.67	72.22	77.78	83.33	88.89	94.44	100.00	105.56
[51,]	51.00	62.33	68.00	73.67	79.33	85.00	90.67	96.33	102.00	107.67
[52,]	52.00	63.56	69.33	75.11	80.89	86.67	92.44	98.22	104.00	109.78
[53,]	53.00	64.78	70.67	76.56	82.44	88.33	94.22	100.11	106.00	111.89
[54,]	54.00	66.00	72.00	78.00	84.00	90.00	96.00	102.00	108.00	114.00
[55,]	55.00	67.22	73.33	79.44	85.56	91.67	97.78	103.89	110.00	116.11
[56,]	56.00	68.44	74.67	80.89	87.11	93.33	99.56	105.78	112.00	118.22
[57,]	57.00	69.67	76.00	82.33	88.67	95.00	101.33	107.67	114.00	120.33

```
[58,]  58.00  70.89  77.33   83.78  90.22  96.67  103.11 109.56 116.00  122.44
[59,]  59.00  72.11  78.67   85.22  91.78  98.33  104.89 111.44 118.00  124.56
[60,]  60.00  73.33  80.00   86.67  93.33 100.00  106.67 113.33 120.00  126.67
```

In manchen Situation ist es ausreichend, aufgrund der Grenze zwischen eins und fünf die Notengrenzen zu ermitteln. Dazu wird noch die Funktion `compute.limits()` entworfen.

14 ⟨*start* 14⟩ ≡ ⊂ 36

```
compute.limits<-function(limit.1.2=90){
  limit.4.5<-limit.1.2/2
  ⟨berechne grobe Grenzen 1⟩
  limits.grob<-rev(limits.grob)
  h<-as.character(5:1)
  names(limits.grob)<-rev(paste(h[-1],"-",h[-5],"-Limit",sep=""))
  ⟨berechne feine Grenzen 2⟩
  limits.fein<-rev(limits.fein)
  h<-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))
  names(limits.fein)<-rev(paste(h[-1],"-",h[-11],"-Limit",sep=""))
  return(list("grobe Grenzen"=limits.grob,"feine Grenzen"=limits.fein))
}
cat("Funktion compute.limits() definiert.\n",
    "compute.limits() ermittelt Noten-Limits basierend auf Grenze von 1 nach 2;")
cat("probiere\n  > compute.limits(limit.1.2=160)\n")
```

6 Restarbeiten

6.1 Der Einleseprozess

Es gilt für den Einleseprozess verschiedene verwendete Chunks zu definieren.

15 ⟨*Filename ermitteln und auf* `fname` *ablegen* 15⟩ ≡ ⊂ 9

```
fname<-tclvalue(tkgetOpenFile(title="Datei mit den Klausurpunkten?"))
#if(!file.exists(fname)){cat("Datei nicht gefunden\n"); return()}
```

16 ⟨*Trennzeichen in Datei* `fname` *ermitteln und auf* `sep` *ablegen* 16⟩ ≡ ⊂ 9

```
encoding.of.csv <- "unknown" # # 130724
a<-c(scan(fname,what="",sep="",encoding=encoding.of.csv),",",";") # encoding
encoding.check <- try({ nchar(a)})
if(class(encoding.check)=="try-error"){
  print("FEHLER: encoding ist unbekannt! Bitte ein encoding angeben!")
  print(" Bsp: latin1, utf8, UTF-8 ")
  print("Eingabe:")
  encoding.of.csv <- readline()
  a<-c(scan(fname,what="",sep="",encoding=encoding.of.csv),",",";") # encoding
  cat(fname,"mit Encoding",encoding.of.csv,"eingelesen")
  encoding.check <- try({ nchar(a)})
  if(class(encoding.check)=="try-error"){
    print("Sorry, hat nicht geklappt!")
    return()
  } else { print("Einlesen erfolgreich!") }
}
if(encoding.of.csv != "unknown") a <- iconv(a,encoding.of.csv,"")  # encoding
ax<-a[which.max(nchar(a))]; axn<-nchar(ax)
```

```
if(sum(is.na(match(substring(ax,1:axn,1:axn),c(LETTERS,letters,0:9,",",";"))))>(axn/2)){
  cat("ACHTUNG:\n","Input-Datei wahrscheinlich keine csv-Datei mit ';'",
      "oder ',' als Trennzeichen!\n",
      "Check Input-Datei",fname,"!!\n")
  return()
}
sep<-c(",",";")[1+(length(grep(",",a))<length(grep(";",a)))]
```

17 ⟨*Kopfinfos, Tabellenbereich und Fußbereich trennen* 17⟩ ≡ ⊂ 9

```
a<-readLines(fname, encoding=encoding.of.csv); # encoding
if(encoding.of.csv != "unknown") a <- iconv(a,encoding.of.csv,"") # encoding
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 <- matrix("
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 <- matri
```

18 ⟨*Punkte extrahieren* 18⟩ ≡ ⊂ 9

```
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!

19 ⟨*Kopfzeile finden oder erstellen* 19⟩ ≡ ⊂ 9

```
# 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("eingelesene Punkte:\n"); print(x)
}
```

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

20 ⟨*Anzahl der Leistungspunkte erfragen* 20⟩ ≡ ⊂ 9

```
LPs<-readline("Wie viele Leistungspunkte kann die Veranstaltung erbringen? ")
LPS<-gsub("[^1-9]","",LPs)
if(nchar(LPs)==0 || nchar(LPs)>2) return(paste("FEHLER: ->",LPs,
                  "Leistungspunkte unbekannt!\n Bitte noch einmal starten!"))
```

21 ⟨*speichere wichtige Infos* 21⟩ ≡ ⊂ 9

```
# 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)
```

6.2 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.

22 ⟨*berechne Notengrenzen* 22⟩ ≡ ⊂ 6, 7

```
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)
```

6.3 Die Elemente der Ausgabegraphik

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

23 ⟨*zeichne Histogramm der Punkte zu Notengrenzen* 23⟩ ≡ ⊂ 8

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

24 ⟨*zeichne Noten-Verteilung* 24⟩ ≡ ⊂ 8

```
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. Unten werden in der Graphik einige technische Daten zu der Veranstaltung eingetragen. Neuerung Mai 2007: Angabe von α mit 3 Nachkommastellen.

25 ⟨*drucke Notenspiegel in die Graphik* 25⟩ ≡ ⊂ 8

```
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)
h<-round(limits[-1],1); h[length(h)]<-"oo"
text(2:n,rep(3,n-1),h,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))),drop=FALSE]
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)
}
```

6.4 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.

26 ⟨*definiere* `speichere.noten` 26⟩ ≡ ⊂ 9

```
speichere.noten<-function(...){
  cat("Speicherung beginnt\n")
  ⟨trage Noten in rumpf.matein 27⟩
  ⟨setze Liste out.csv zusammen 28⟩
  ⟨erstelle txt-Liste mit LPs 30⟩
  ⟨erstelle txt-Liste mit Punkten 31⟩
  ⟨speichere Listen und Bild 29⟩
  "ok"
}
```

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

27 ⟨*trage Noten in* `rumpf.mat`*ein* 27⟩ ≡ ⊂ 26

```
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.

28 ⟨*setze Liste* `out.csv` *zusammen* 28⟩ ≡ ⊂ 26

```
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.

29 ⟨*speichere Listen und Bild* 29⟩ ≡ ⊂ 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.

30 ⟨*erstelle txt-Liste mit LPs* 30⟩ ≡ ⊂ 26

```
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.

31 ⟨*erstelle txt-Liste mit Punkten* 31⟩ ≡ ⊂ 26

```
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* 32⟩
```
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.

32 ⟨*erstelle Spiegel* 32⟩ ≡ ⊂ 31
```
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")))
```

6.5 `noten.fein` und `noten.grob`

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

33 ⟨*definiere* `noten.fein` *und* `noten.grob` 33⟩ ≡ ⊂ 9
```
noten.fein<-function(...){
    slider(obj.name="method",obj.value="fein")
    effect.alpha()
}
noten.grob<-function(...){
    slider(obj.name="method",obj.value="grob")
    effect.alpha()
}
```

7 Anhang: `quantile_stetig`

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

34 ⟨*definiere* `quantile_stetig` 34⟩ ≡ ⊂ 5
```
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],-Inf);   x2<-min(x[x>x0],Inf)
  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,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 ",
             qtype,"):",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,qtype,.5*min(x),max(x))); "ok"
}
## quantile_stetig(rnorm(10))
```

7.1 Erstellung einer Punkte-Demo-Datei

35 ⟨*konstruiere Demo-Datei mit Punkten* 35⟩ ≡

```
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,"\n",sep="")
cat(file="klpkte.csv",sep="",out)
```

7.2 Die Definition der Sliderfunktion

36 ⟨* 10⟩+ ≡
⟨*start* 14⟩

Für alle Fälle wird die Funktion `slider` bereitgestellt.

37 ⟨*start* 14⟩+ ≡ ⊂ 36

```
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, "600x150+0+50") # 130724
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
        sep = "")))
    tkpack(fr <- tkframe(nt))
    lab <- tklabel(fr, text = sl.names[i], width = "25")
    sc <- tkscale(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));tkdestro
    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)
}
```
⟨*definiere* `select.limits` 9⟩

7.3 Verarbeitung zum .R-File.

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

38 ⟨*tangle* `select.limits` 38⟩ ≡ ⊂ 10
```
tangleR("kpneu","noten.R","") # definiere [[select.limits]]")
file.copy("noten.R","noten-ekvv.R",overwrite=TRUE)
```

8 Index

Object Index

Code Chunk Index