Annotations to Plots

File: annotations8d.rev
in: /home/pwolf/R/aplpack/annotations

January 13, 2014

Inhalt

Code Chunk Index

1 Problemstellung

Wiederholt tauchen Situationen auf, in denen man die graphischen Fähigkeiten von R und die Formatierungskünste von LaTeX kombinieren möchte. Man möchte zum Beispiel nachträglich in ein Bild eine schön formatierte mathematische Gleichung eintragen oder man möchte eine simple Überschrift wie *Dichte der χ^2-Verteilung* der Graphik hinzufügen. Für solche Anliegen werden in diesem Papier Lösungen beschrieben.

Für die Ergänzung von LaTeX-Texten lassen sich verschiedene Situationen unterscheiden:

Daten Analyse **Situation** Ein R-Anwender befindet sich mitten in einer Analyse und hat einen wunderschönen Ergebnisplot geschaffen. Um einem fremden Betrachter auf eine Besonderheit hinzuweisen, möchte er in dem Plot erklärende LaTeX-Texte und einen Hinweispfeil zur Lenkung des Blicks ergänzen.

Lösungsansatz: Eine Picture-Umgebung zur Positionierung Aus technischer Sicht liegt ein geöffnetes graphisches Device vor, das für einen Papier-Ausdruck kopiert bzw. in einem passenden Format rekonstruiert werden muss. Die entstehende Datei mit dem Bild kann dann in ein LaTeX-Dokument eingebunden werden. Für die gewünschten Kommentare sind LaTeX-Schnipsel erforderlich, die platziert werden müssen.

Für das zielgenaue Anheften von Anmerkungen benötigt man ein Koordinatensystem. Da die Formatierung mittels TeX stattfindet, muss dieses Koordinatensystem in der TeX-Welt organisiert werden. Wünschenswert wäre es zudem, wenn der Anwender später auch noch kleine Korrekturen am TeX-Code vornehmen könnte. Als einfache Lösung für normale LaTeX-Anwender bietet sich eine `picture`-Umgebung an, in der sich Texte an Koordinatenpunkten fixieren lassen.

Das Koordinatensystem aus Anwendersicht Während der Datenanalyse, also bei der Erstellung der Graphik, sind Beschreibungen in Koordinatensystem-Einheiten einer Picture-Umgebung nicht bekannt und können deshalb innerhalb der R-Umgebung nicht genutzt werden. Auch ist ein Weg zu umständlich, bei dem der Anwender zunächst das Bild "ausdruckt", dann die Orte mit einem Lineal vermisst, um in einem weiteren Schritt Zusatztexte mit Positionsangaben in das LaTeX-Quelldokument einzubringen. Außerdem könnte im Falle der Veränderung der Graphik eine Wiederholung aller Schritte notwendig werden.

Wenig sinnig ist auch eine Vermessung von mm-Positionen auf dem Bildschirm. Deshalb sollen innerhalb der R-Welt Orte mit Hilfe des Weltkoordinatensystems der Darstellung festgelegt werden. Dieser Ansatz führt natürlich ebenfalls zu Problemen. Denn logarithmische Skalierungen wie auch die Erstellung mehrere Einzelgraphiken in einem Device sind dann nicht mehr gut zu hantieren.

Input: Positionen in Weltkoordinaten Damit gehen wir im Folgenden bei der Definition von Positionen in der R-Welt von den Weltkoordinaten des Anwenders aus. Diese muss der Anwender für seine Ergänzungstexte angeben. Für Randbereiche, um in diesen typische Inhalten wie Skalen- und Achsenbeschriftungen sowie Titel einzutragen, sollen dagegen automatische Platzierungen vorgenommen werden. Zur Erleichterung der Platzierung von Annotationen im Inneren der Graphik könnte ein interaktives Tool helfen, das auf die Funktion `locator()` zurückgreift.

Output: Wohin mit dem LaTeX-Code und welche Form soll er haben? Nachdem alles festgelegt ist, kann LaTeX-Code generiert werden. Fraglich ist, wo die erarbeiteten LaTeX-Anweisungen bleiben sollen. Hierfür sind verschiedene Szenarien vorstellbar.

- Der LaTeX-Code kann auf der Konsole in einer Form ausgeben ausgegeben werden, dass er sich per Cut-and-Paste weiterverarbeiten lässt.
- Alternativ könnte er auch in einer TeX-Datei landen.
- Der Code könnte auf Wunsch mit einer Präambel und einem Ende (`\end{document}`) versehen werden.
- Weiterhin können wir uns vorstellen, dass mehrere Bilder in einem Dokument gesammelt werden sollen. Dann dürfte nur einmal ein Kopf und ein Dokument-Ende erstellt werden.

Naheliegend ist es, solche Alternativen anzubieten und durch Parameter zu steuern.

`relax` Im Prinzip ändert sich wenig, wenn sich die interaktive Arbeit mit Hilfe des `relax`-Editor zuträgt. Eine typische Situation lässt sich wohl so beschreiben: In einem Papier ist eine spezielle Analyse beschrieben. Nun sollen neue Daten verarbeitet werden. Dazu laden wir das passende Papier und die neuen Daten in den Editor und aktivieren die Chunks, die die neuen graphischen Ergebnisse erzeugen. Wird in einem Chunk eine Graphik mit Annotationen beschrieben, dann werden die gewünschten Annotationen in LaTeX-Befehle umgesetzt. Auch hier muss ein neues Bild als Datei erzeugt werden, und die Annotationen müssen in LaTeX-Anweisungen übersetzt werden. Für einzelne Bilder würde man sich die Generation einer formatierbaren LaTeX-Datei wünschen. Soll das formatierte rev-Dokument das Bild enthalten, müssen die LaTeX-Anweisungen ins Arbeitsfeld eingetragen werden. Die Fixierung der Orte könnte auch in diesem Szenario interaktiv unterstützt werden. Doch gilt es zunächst, die Basisarbeiten zu erledigen.

Als Outputs für die LaTeX-Anweisungen kommen also in Betracht: Output-Fenster, TeX-Datei für `\input{...}`-Kommandos oder vollständige Datei, die sich formatieren lässt.

`Sweave` Eine automatische Integration von Bild und Annotationen in ein Dokument ist auch im Rahmen der `Sweave`-Verarbeitung wünschenswert. In dieser Situation müssen alle Anweisungen zur Erzeugung des fertigen Dokumentes in der Quelldatei stehen. Der `Sweave`- und Formatierprozess realisiert dann alle R-Ausdrücke und generiert alle Bilder. Dabei sind auch alle beschriebenen Annotationen umzusetzen. Folglich müssen die zugehörigen Annotationsorte und -texte im Quellfile enthalten sein.

Da `Sweave` einen Mechanismus zur Einbindung von Graphiken besitzt, ist eine Idee, diesen zu nutzen. Jedoch führt eine solche Bilderstellung zu speziellen Strukturen, bei denen der `\includegraphics`-Befehl in Sweave-Klammen verpackt wird. Diese erlauben es nicht ohne Weiteres, an anderer Stelle eine Picture-Umgebung zu öffnen, dann den von Sweave generierten Include-Befehl zu nutzen, um anschließend weitere Picture-Elemente anzuschließen und die Umgebung zu schließen. Deshalb bleibt auch bei der Sweave-Nutzung als praktizierbarer Weg, den Erzeugungsprozess der Graphik selbst in die Hand zu nehmen.

2 Wunsch, Beispiel und Test

2.1 Bilder mit verschiedenen Annotationen

Ein Ein-Bild-Test: Stellen wir uns zunächst eine graphische Darstellung der Zahlen von 1 bis 6 in einem Scatterplot, vor. In diesen Plot wollen wir:

- an der Stelle (2,5) den Text $\boxed{\text{LaTeX-Text}}$ platzieren,
- als Überschrift *LaTeX-Titel* setzen,
- als Untertitel das Datum eintragen,
- als x-Label und als y-Label LaTeX-X-lab bzw. LaTeX-Y-labschreiben,
- die x-Ticks mit den Bezeichnungen X_1 bis X_6 versehen,
- die y-Ticks zu den y-Werten 2, 4 und 6 mit Y_a, Y_b, Y_c bezeichnen.

Wir müssen also einen Plot erstellen und diesen mit weiteren Elementen anreichern. Dieses muss die Funktion `anno()` umsetzen, die eine R-Graphik konstruiert, sie in eine Datei kopiert und einen geeigneten LaTeX-File erzeugt.

Es ist aus Anwendersicht einleuchtend, die R-Plot-Anweisungen zusammen zu fassen und dann nacheinander die verschiedenartigen Dinge zu ergänzen. Zum Schluss sind noch ein paar technische Dinge wie Größenangaben oder Dateiname festzulegen. Je nach intendierter Weiterverarbeitung muss ein passender Treiber (`jpg` für pdfLaTeX oder `postscript` für LaTeX) gewählt werden.

1 ⟨*ein Bild mit* `anno()` *erzeugt und angereichert* 1⟩ ≡

```
⟨define all in one function do.anno 34⟩
driver <- "jpg" ; driver <- "postscript" # !!!!!!!!!

do.anno(
   # Plotanweisungen
        {
         plot(1:6, axes=FALSE, ylab="", xlab="")
         title("<-                                ->", sub="I")
         axis(1, labels=FALSE); axis(2, labels=FALSE)
         text(5, 2, "Ein-R-Text")
         abline(h=mean(par()$usr[3:4]), v=mean(par()$usr[1:2]))
        },
   # Text ins Bild
        x=2, y=5, txt="\\fbox{\\LaTeX-Text}", txt.chsize="\\large",
   # Titel
        main="\\emph{\\LaTeX-Titel}", sub=date(),
   # Achsenbezeichnungen
        xlab="\\LaTeX-X-lab",         ylab="\\LaTeX-Y-lab",
   # Tickbezeichnungen
        xticklabels = paste("$X_", 1:6, "$", sep=""), xticklabels.pos="",
        yticklabels = paste("$Y_", c("a", "b", "c"), "$", sep=""),
        yticklabels.pos = 2*(1:3), yticklabels.rot = 90,
   # Zielobjekt
        file="t1.tex", head=TRUE, driver=driver,
   # Bildeigenschaften
        width=100, height=80, designsize=70
)  # end of do.anno call

# Schritte der Verarbeitung
⟨finde os-Typ 2⟩
if(os=="windows"){ shell( "echo q | latex t1"); shell("yap t1.dvi", wait=FALSE) }
if(os=="linux")    system("echo q | latex t1; dvipdf t1; o t1.pdf")            # linux
if(os=="mac")      system("echo q | /usr/texbin/pdflatex t1.tex; open t1.pdf") # mac
```

Für die Test-Umsetzung haben wir nach Betriebssystem unterschieden, dieses muss aber erst noch ermittelt werden.

2 ⟨*finde os-Typ* 2⟩ ≡ ⊂ 1, 4, 6, 8, 11, 13, 14, 36, 39, 41, 43

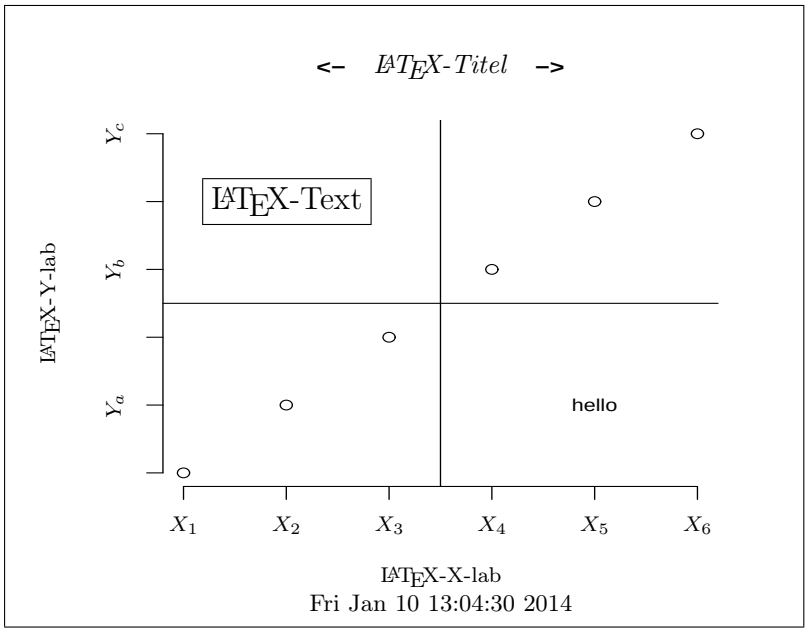
```
os <- c("linux", "mac", "windows")[1+(0<length(grep("^dar", version$os)))+
                                   2*(0<length(grep("^min", version$os)))]
```

Ergebnis des Ein-Bild-Tests. Bei diesem Test kommt eine Datei heraus, die wir etwas abstrippen ...

3 ⟨* 3⟩ ≡

```
idx <- grep("document}", scan("t1.tex","",sep="\n"))
system(paste(sep="","head -",idx[2]-1," t1.tex | ",
             "tail -",idx[2]-idx[1]-1," > test-outputs/test1.tex"))
```

und mit `\input` einbinden können:



Ein Zwei-Bilder-Beispiel: Es muss natürlich auch möglich sein, mehrere Bilder zu definieren. Das Vorgehen wird mit folgendem Chunk demonstriert. In diesem Beispiel finden wir zwei `anno()`-Aufrufe. Beim ersten werden `head=TRUE` und `tail=FALSE` gesetzt, beim zweiten erfolgt verständlicherweise eine umgekehrte Belegung.

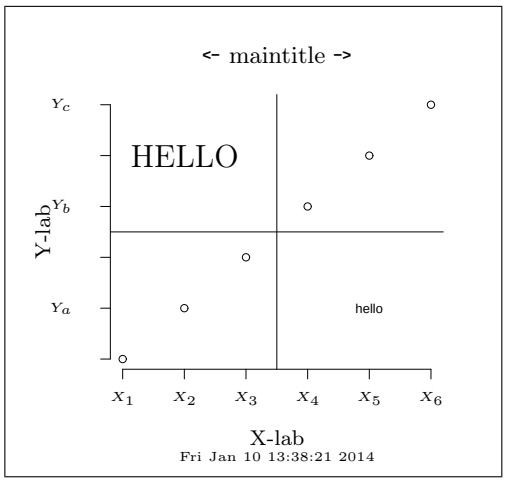
4 ⟨*test 2 of all in one* 4⟩ ≡

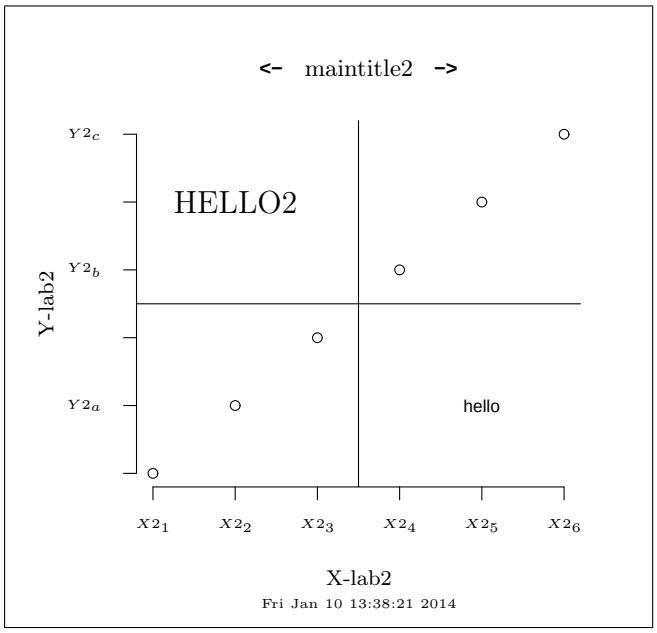
```
driver <- "jpg" ; driver <- "postscript" # !!!!!!!
⟨define all in one function do.anno 34⟩

do.anno(
  # Plotanweisungen
      {
        plot(1:6, axes=FALSE, ylab="", xlab="")
        title(paste("<-", paste(rep(" ", 25), collapse=""), "->"))
        axis(1, labels=FALSE); axis(2, labels=FALSE)
        text(5, 2, "hello")
        abline(h=mean(par()$usr[3:4]),v=mean(par()$usr[1:2]))
      },
  # Text ins Bild
      x=2, y=5, txt="HELLO", txt.chsize="\\large",
  # Titel
      main="maintitle", sub=date(), xlab="X-lab", ylab="Y-lab",
      main.chsize="\\small", sub.chsize="\\tiny", lab.chsize="\\footnotesize",
  # Tickbeschriftungen
      xticklabels = paste("$X_", 1:6, "$", sep=""),
      xticklabels.pos = "",
      yticklabels = paste("$Y", c("a$", "b$", "c$"), sep="_"),
      yticklabels.pos = 2*(1:3), yticklabels.rot = "",
      xyticklabels.chsize="\\tiny",
  # Zielobjekt
      file="t1.tex", head=TRUE, tail=FALSE, driver=driver,
  # Bildeigenschaften
      width=60, height=60, designsize=70
)
do.anno(
  # Plotanweisungen
      {
        plot(1:6, axes=FALSE, ylab="", xlab="")
        title(paste("<-", paste(rep(" ",25), collapse=""), "->"))
        axis(1, labels=FALSE); axis(2, labels=FALSE)
        text(5, 2, "hello")
        abline(h=mean(par()$usr[3:4]), v=mean(par()$usr[1:2]))
      },
  # Text ins Bild
      x=2, y=5, txt="HELLO2", txt.chsize="\\large",
  # Titel
      main="maintitle2", sub=date(), xlab="X-lab2", ylab="Y-lab2",
      main.chsize="\\small", sub.chsize="\\tiny", lab.chsize="\\footnotesize",
  # Tickbeschriftungen
      xticklabels = paste("$X2_",1:6,"$",sep=""),
      xticklabels.pos = "",
      yticklabels = paste("$Y2", c("a$", "b$", "c$"), sep="_"),
      yticklabels.pos = 2*(1:3), yticklabels.rot = "",
      xyticklabels.chsize="\\tiny",
  # Zielobjekt
      file="t1.tex", head=FALSE, tail=TRUE, driver=driver,
  # Bildeigenschaften
      width=80, height=80, designsize=70
)
# Schritte der Verarbeitung

⟨finde os-Typ 2⟩
if(os=="windows"){ shell("echo q  | latex t1"); shell("yap t1.dvi", wait=FALSE)}
if(os=="linux")    system("echo q | latex t1; dvipdf t1; o t1.pdf") # linux
if(os=="mac")      system("echo q | /usr/texbin/pdflatex t1.tex; open t1.pdf") # mac os
```

Ergebnis des Zwei-Bilder-Tests. Wir wollen uns das Ergebnis des zweiten Tests ansehen:

⟨ * 3⟩+ ≡

```
idx <- grep("document}", scan("t1.tex","",sep="\n"))
system(paste(sep="","head -",idx[2]-1," t1.tex | ",
             "tail -",idx[2]-idx[1]-1," > test-outputs/test2.tex"))
```





Damit lassen sich nun einzelne oder auch mehrere Bilder anreichern. `width` und `height` beeinflussen die Bildbreite und -höhe. Die Designgröße legt fest, für welche Größenordnung – unabhängig von der konkreten Verwendung das Bild – geplant wird. Große Designgrößen führen bei fester Bildgröße im Text zu relativ kleineren Rändern als kleine Designgrößen. Als Voreinstellung und Orientierungspunkt wurde 100 bzw. 100mm gewählt.

Wenn man zwischen die Bilder Text einfügen möchte, kann man das durch eine entsprechende Setzung des `tail`-Argumentes beim ersten `do.anno`-Aufrufs erreichen. Eine Umsetzung von `head` im zweiten geht dagegen nicht.

2.2 DT: Dichte der χ^2-Verteilung

Für den ersten Testfall von DT erstellen wir ein Histogramm von χ^2-Zufallszahlen und zeichnen ein Dichtekurve durch das Balkengebirge. Die Überschrift soll nun LaTeX-Symbole enthalten. Zum Check der Platzierung des Untertitels erzeugen wir diesen auf zwei Wegen: per R und bei der Formatierung.

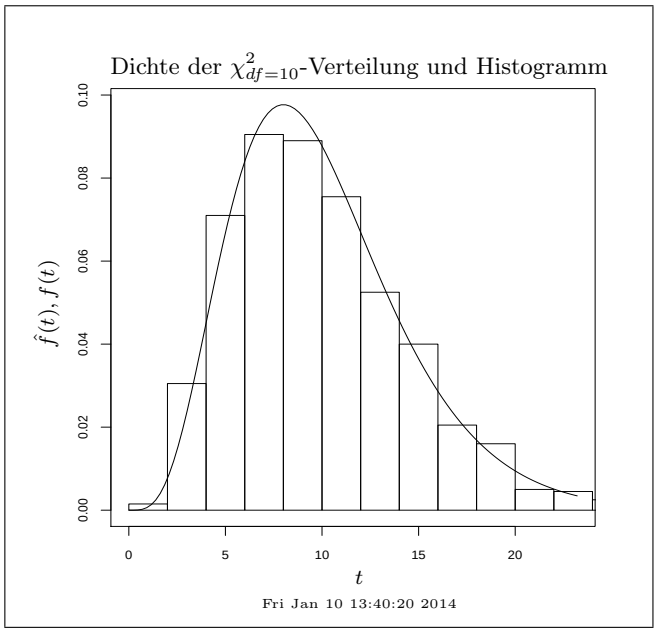
6 ⟨*test* 6⟩ ≡

```
driver <- "jpg" ; driver <- "postscript"  # !!!!!!!
⟨define all in one function do.anno 34⟩

do.anno(
  # Plotanweisungen
     {
       set.seed(7); x <- rchisq(1000,10)
       curve(dchisq(x,10),xlim=c(0,qchisq(0.99,10)),xlab="",ylab="")
       hist(x, freq=FALSE, add=TRUE, breaks=20); title(sub=date())
     },
  # Titel
      main="Dichte der $\\chi^2_{df=10}$-Verteilung und Histogramm",
      xlab="$t$", ylab="$\\hat f(t), f(t)$", sub=date(),
      main.chsize="\\small", sub.chsize="\\tiny", lab.chsize="\\footnotesize",
  # Zielobjekt
      file="dtchi.tex", head=TRUE, tail=TRUE, driver=driver,
  # Bildeigenschaften
      width=80, height=80, designsize=70
)

# Schritte der Verarbeitung
⟨finde os-Typ 2⟩
if(os=="windows"){ shell( "echo q | latex dtchi"); shell("yap dtchi.dvi", wait=FALSE)}
if(os=="linux")    system("echo q | pdflatex dtchi.tex; o dtchi.pdf")
if(os=="mac")      system("echo q | /usr/texbin/pdflatex dtchi.tex; open dtchi.pdf") # mac os
```

Ergebnis Test3 Wieder erhalten wir eine Datei, die wir sogleich einbinden.

7 ⟨* 3⟩+ ≡

```
idx <- grep("document}", scan("dtchi.tex","",sep="\n"))
system(paste(sep="","head -",idx[2]-1," dtchi.tex | tail -",idx[2]-idx[1]-1," > test-outputs/test3.tex"))
```



2.3 DT: Integralapproximation

Das zweite Beispiel von DT zeigt die Approximation eines Integrals durch eine Obersumme. Es sei darauf hingewiesen, dass die Pfeile mittels LaTeX gesetzt werden.

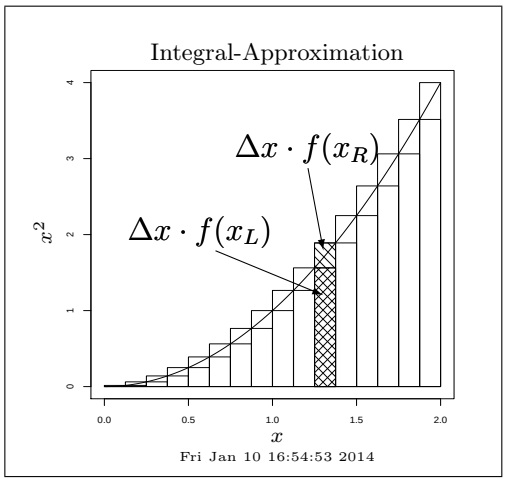
8 ⟨* 3⟩+ ≡

```
driver <- "jpg" ;# driver <- "postscript" # !!!!!!!!
⟨define all in one function do.anno 34⟩

do.anno(
  # Plotanweisungen
     {
       Delta <- 1/8
       curve(x^2,xlim=c(0,2),xlab="",ylab="")
       for(x in seq(0,2-Delta,by=Delta)) rect(x,0,x+Delta,(x+Delta)^2)
       for(x in seq(0,2-Delta,by=Delta)) rect(x,0,x+Delta,x^2)
       rect(1.25,0,1.375,1.25^2,density=10)
       rect(1.25,0,1.375,1.375^2,density=10,angle=-45)
     },
  # Text ins Bild
      x=c(.5,1.2), y=c(2,3.3), txt.chsize="\\large",
      txt=c("$\\Delta x \\cdot f(x_L)$","$\\Delta x \\cdot f(x_R)$"),
  # Titel
      main="Integral-Approximation", xlab="$x$", ylab="$x^2$",  sub=date(),
      main.chsize="\\small", sub.chsize="\\tiny", lab.chsize="\\footnotesize",
  # Pfeile
      vec.x1 = c(.6,1.2), vec.y1 = c(1.7,3), vec.x2 = c(1.3,1.3), vec.y2 =  c(1.0,1.7),
  # Zielobjekt
      file="dtxq.tex", head=TRUE, tail=TRUE, driver=driver,
  # Bildeigenschaften
      width=60, height=60, designsize=70
)

# Schritte der Verarbeitung
⟨finde os-Typ 2⟩
if(os=="windows"){ shell( "echo q | latex dtxq"); shell("yap dtxq.dvi", wait=FALSE)}
# if(os=="linux")  system("echo q | pdflatex dtxq.tex; o dtxq.pdf") # driver<-"jpg"!!
if(os=="linux")    system("echo q | latex dtxq.tex; dvipdf dtxq; o dtxq.pdf")
if(os=="mac")      system("echo q | /usr/texbin/pdflatex dtxq.tex; open dtxq.pdf") # mac os
```

Ergebnis Test4 Das Ergebnisbild haben wir etwas kleiner gesetzt, doch wird es hoffentlich noch zu erkennen sein.

9 ⟨* 3⟩+ ≡

```
idx <- grep("document}", scan("dtxq.tex","",sep="\n"))
system(paste(sep="","head -",idx[2]-1," dtxq.tex | ",
             "tail -",idx[2]-idx[1]-1," > test-outputs/test4.tex"))
```



3 Bausteine für Annotationsvorhaben

In dieser Sektion betrachten wir eine Menge von Low-Level-Funktionen, mit denen wir die in der letzten Sektion verwendeten High-Level-Funktion `do.anno()` definieren können. Diese Baustein-Funktionen erlauben es zudem, weitere High-Level-Funktionen zu entwerfen.

Grundsätzlich werden für ein Annotationsvorhaben Initialisierungen benötigt. Diese sollen von der Funktion `anno.open.annotations` durchgeführt werden. Ebenso muss jede einzelne Graphik eingeleitet werden. Diesen Job wird die Funktion `anno.begin.picture` erledigen. Die Funktionen `anno.end.picture` und `anno.close.annotations` dienen als Pendants zum schließen eines Bildes bzw. eines Annotationsvorhaben. Für die Ergänzung graphischer Elemente mittels LaTeX finden die Funktionen `anno.text`, `anno.vector`, `anno.ticklabels`, `anno.title` Verwendung. Damit eine Grundlage für Annotationen existiert, muss zunächst eine Graphik geschaffen worden sein. Hierzu erstellt `anno.copy.R.plot` quasi eine Kopie des aktuellen graphischen Device erstellt wird. Ohne Kopiervorgänge – also ohne einen geöffneten graphischen Bildschirmdevice vorauszusetzen – erzeugt das Funktionspaar `anno.begin.R.plot` und `anno.end.R.plot` ebenfalls eine Datei mit einer Graphik.

3.1 Übersicht über die elementaren Funktionen

Eine kurze Zusammenfassung der angesprochenen Funktionen zeigt folgende Tabelle:

`anno.open.annotations`	initialisiert wesentliche (LaTeX-) Größen.
`anno.begin.picture`	definiert Bildbeginn per `\begin{picture}` u.a.
`anno.copy.R.plot`	kopiert den Graphik-Device.
`anno.text`	platziert Texte in den Darstellungsbereich.
`anno.vector`	erstellt mit Hilfe von LaTeX-Befehlen einen Vektor.
`anno.ticklabels`	setzt Namen an Stellen der x- oder y-Achsen.
`anno.title`	platziert xlab, ylab, main und sub.
`anno.end.picture`	schließt ein Bild und die Picture-Umgebung.
`anno.close.annotations`	schließt Datei ab.

Mit diesen Funktionen lassen sich Annotationswünsche während Datenanalysen umsetzen. Für den Gebrauch von Sweave sind dagegen folgende beiden Funktionen entworfen worden:

`anno.begin.R.plot`	öffnet ein graphisches Device und integriert eine `includegraphics`-Anweisung an.
`anno.end.R.plot`	definiert Transformationsfunktionen, schließt das graphische Device und damit die Datei mit dem Bild.

Für Bereingigungsprozesse wird letztlich noch die Funktion

`anno.clean.pic.info`	bereinigt ggf. den Speicher der Bildinfos.

3.2 Definition der Low-Level-Funktionen

Die Definitionen der Funktionen fassen wir in einem Chunk zusammen, so dass wir bei Veränderungen schnell eine aktuelle Fassung aller dieser erzeugen können. Auch legen wir die Menge der nun definierten elementaren Funktionen in der Datei `anno.R` ab.

10 ⟨*define Annotationsfunktionen* 10⟩ ≡ ⊂ 11, 13, 14, 34, 36

```
h <- ls(pattern="^anno"); if(0 < length(h)) rm(list=h) # cleaning
⟨define anno.clean.pic.info 33⟩
⟨define anno.open.annotations 18⟩
⟨define anno.begin.picture 21⟩
⟨define anno.begin.R.plot 29⟩
⟨define anno.copy.R.plot 28⟩
⟨define anno.end.R.plot 30⟩
⟨define anno.end.picture 22⟩
⟨define anno.close.annotations 20⟩
⟨define anno.transformtion.fns 31⟩
⟨define anno.text 24⟩
⟨define anno.ticklabels 26⟩
⟨define anno.title 27⟩
⟨define anno.vector 25⟩
dump(ls(pattern="^anno"), file="anno.R")
"anno functions (re)defined"
```

3.3 Tests der Low-Level-Funktionen

Die Tests aus diesem Abschnitt basieren (wie auch die Funktion `do.anno()`) auf Funktionen, die bisher noch nicht entworfen sind. Deshalb könnten diese Tests auch gut nach der Diskussion der einzelnen Funktionen platziert werden. Jedoch werden sich durch ihre frühzeitige Demonstration grobe Vorstellungen über den Einsatzes der einzelnen Funktionen und über strukturelle Zusammenhänge einstellen. Auch kann so eventuell die kritische Würdigung des Entwurfs erleichtert werden.

Ein Plot der Zahlen von 1 bis 6. Für einen einfachen Test erstellen wir wieder einen Plot, der dann mit Annotationen unter Verwendung der elementaren Funktionenen angereichert wird.

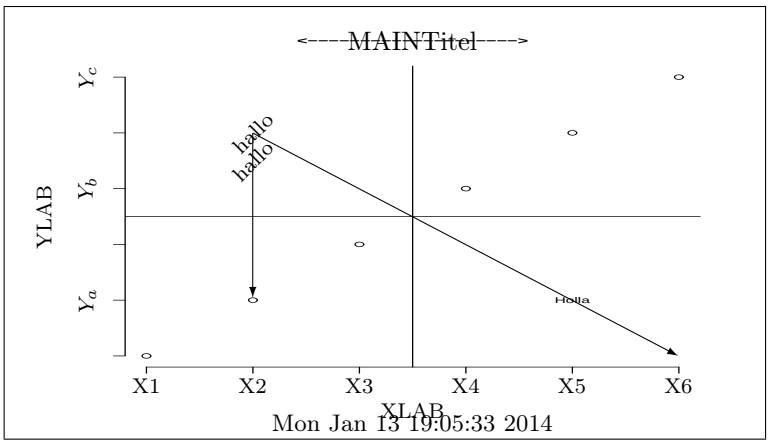
11 ⟨*test* 6⟩+ ≡

```
# set some parameters and define low level functions
driver <- "jpg" ; driver <- "postscript" # !!!!!!!!!!!
chsize <- c("\\normalsize","\\footnotesize","\\tiny")[2]
file   <- "t2.tex"
⟨define Annotationsfunktionen 10⟩

# construct plot # if(dev.cur()>2) dev.off(); dev.cur() # close open graphics devices
plot(1:6, axes=FALSE, ylab="", xlab="", main="<-------------------->")
axis(1, labels=FALSE); axis(2, labels=FALSE)
text(5, 2, "Holla")
abline(h=mean(par()$usr[3:4]), v=mean(par()$usr[1:2]))
```

```
# describe annotations
anno.clean.pic.info()
  anno.open.annotations(head=TRUE, file=file, width=95, height=55)
    anno.begin.picture()
      anno.copy.R.plot(driver=driver)
      anno.text(2,   5, "hallo", txt.adj="c", txt.rot=45, txt.chsize=chsize)
      anno.text(2, 4.5, "hallo", txt.adj="c", txt.rot=45, txt.chsize=chsize)
      anno.title(main="MAINTitel", sub=date(), xlab="XLAB", ylab="YLAB")
      anno.ticklabels(paste("X", 1:6, sep=""), xyticklabels.chsize=chsize,
                      paste("$Y_", c("a$", "b$", "c$")), yticklabels.pos=2*(1:3))
      anno.vector(2, 5, 6, 1); anno.vector(2, 5, 2, 2)
    anno.end.picture()
  anno.close.annotations()

# define some processing commands
⟨finde os-Typ 2⟩
if(os=="windows"){ shell("echo q | latex t2"); shell("yap t2.dvi", wait=FALSE)} #postscript!!
if(os=="linux"&&driver=="jpg")        system("echo q | pdflatex t2.tex; o t2.pdf") #linux->jpg+pdflatex
if(os=="linux"&&driver=="postscript") system("echo q|latex t2.tex;dvipdf t2;o t2.pdf ") #linux->latex
if(os=="mac")     system("echo q | /usr/texbin/pdflatex t2.tex; open t2.pdf") # mac os
```

Ergebnis Test5 Wir erhalten ein Bild, das unserem ersten Testbild sehr ähnlich sieht.

12 ⟨*erstelle aus* `t2.tex` *eine TEX-Inputdatei* 12⟩ ≡
```
idx <- grep("document}", scan("t2.tex","",sep="\n"))
system(paste(sep="","head -",idx[2]-1," t2.tex | tail -",idx[2]-idx[1]-1," > test-outputs/test5.tex"))
```



Das χ^2-Beispiel von DT will auch nachgestellt sein.

13 ⟨*DT-chi-quadrat* 13⟩ ≡
```
# set some parameters and define low level functions
driver <- "jpg" ; driver <- "postscript"
chsize <- c("\\normalsize","\\footnotesize","\\tiny")[2]
file   <- "dtchi.tex"
⟨define Annotationsfunktionen 10⟩

# construct plot
if(dev.cur()>2) dev.off(); dev.cur()
x <- rchisq(1000,10)
curve(dchisq(x, 10), xlim=c(0, qchisq(0.99, 10)), xlab="", ylab="")
hist(x,freq=FALSE,add=TRUE,breaks=20)

# describe annotations
anno.clean.pic.info()
  anno.open.annotations(head=TRUE, file=file)
    anno.begin.picture(width=80, height=80)
      anno.copy.R.plot(driver=driver)
```

```
      anno.title(main="Dichte der $\\chi^2_{df=10}$-Verteilung und Histogramm",
                 sub=date(), xlab="t", ylab="$\\hat f(t), f(t)$")
    anno.end.picture()
  anno.close.annotations()

# some processing commands
⟨finde os-Typ 2⟩
if(os=="windows"){ shell( "echo q | latex dtchi"); shell("yap dtchi.dvi", wait=FALSE)}# windows
if(os=="linux")    system("echo q | latex dtchi.tex; dvipdf dtchi; o dtchi.pdf")      # linux->latex
if(os=="mac")      system("echo q | /usr/texbin/pdflatex dtchi.tex; open dtchi.pdf")  # mac os
```

Das Integral-Approximations-Beispiel von DT ist mehr eine Fleißarbeit.

14 ⟨*DTxq* 14⟩ ≡

```
# set some parameters and define low level functions
driver <- "jpg" ; driver <- "postscript"
chsize <- c("\\normalsize","\\footnotesize","\\tiny")[2]
file   <- "dtxq.tex"
⟨define Annotationsfunktionen 10⟩

# construct plot # if(dev.cur()>2) dev.off(); dev.cur()
Delta  <- 1/8
curve(x^2, xlim=c(0,2), xlab="", ylab="")
for(x in seq(0, 2-Delta, by=Delta)) rect(x, 0, x+Delta, (x+Delta)^2)
for(x in seq(0, 2-Delta, by=Delta)) rect(x, 0, x+Delta, x^2)
rect(1.25, 0, 1.375, 1.25^2,  density=10)
rect(1.25, 0, 1.375, 1.375^2, density=10, angle=-45)
#arrows(0.695976,1.63574, 1.29813, 0.838917,length=0.07, angle=20, col=1)
#arrows(1.14233, 3.05426, 1.30234, 1.75833, length=0.07, angle=20, col=1)

# describe annotations
anno.clean.pic.info()
  anno.open.annotations(head=TRUE, file=file, width=80, height=80)
    anno.begin.picture()
      anno.copy.R.plot(driver=driver)
      anno.title(xlab="x", ylab="$x^2$", sub=date())
      anno.text( .5, 2,   "$\\Delta x \\cdot f(x_L)$")
      anno.text(1.2, 3.3, "$\\Delta x \\cdot f(x_R)$")
      anno.vector( .6, 1.7, 1.3, 1.0)
      anno.vector(1.2, 3,   1.3, 1.7)
    anno.end.picture()
  anno.close.annotations()
# some processing commands
⟨finde os-Typ 2⟩
if(os=="windows"){ shell( "echo q | latex t1"); shell("yap t1.dvi", wait=FALSE)}
if(os=="linux")    system("echo q | latex dtxq.tex; dvipdf dtxq; o dtxq.pdf") #linux->latex
if(os=="mac")      system("echo q | /usr/texbin/pdflatex dtxq.tex; open dtxq.pdf") # mac os
```

Wir sehen, dass es zunächst gilt, die Umgebung – so sie da ist – zu säubern. Der Annotations-Öffnungsfunktion geben wir neben den gewünschten Größenverhältnissen mit, ob wir einen Kopf haben wollen oder nicht. Ebenfalls könnten wir dieses für Tail machen. Das Bild wird mit Hilfe von `anno.begin.picture()` vorbereitet. Dieser Funktion können wir ebenfalls die gewünschten Bildgrößenparameter mitgeben. Auch lässt sich mittels `fbox=FALSE` der standardmäßig gezeichnete Kasten um die Graphik weglassen. Der Funktion `anno.copy.R.plot()` können wir einen Bildnamen, die Designgröße, den Treiber und die Punktgröße für die Zeichen mitgeben. Per `anno.title()` lassen sich Über-, Unterschriften, Achsenbeschriftungen und Schriftgrößen für diese Qualitäten fixieren. `anno.text()` platziert Texte, wir können wieder wird Schriftgrößen setzen und die Texte adjustieren und rotieren. Mit `anno.vector()` werden Pfeile, die mit Weltkoordinaten beschrieben werden, generiert. Die Schließungsfunktionen benötigt keine Parameter.

4 Umsetzung

Wenn ein großer Job auf kleine Bearbeiter aufgeteilt wird, muss zunächst einmal der Datenaustausch zwischen den Bearbeitern überlegt werden. Hierzu wird in der ersten Untersektion eine eigene Umgebung als Dateispeicher eingeführt. Zweitens gilt es den Rahmen zu definieren, was in der zweiten Untersektion geschieht. Auch die einzelnen Bilder benötigen einen Rahmen, dieser wird in Untersektion 3 besprochen. Das Ablegen erarbeiteter LaTeX-Stücke beschreibt Untersektion 4. Die Unterabschnitte 5 bis 7 stellen die produktiven Funktionen vor, die ergänzende Bildelemente schaffen. In der 8. Untersektion wird das Kopieren einer R-Graphik in eine Dateien umgesetzt. Untersektion 9 beschreibt die direkte Erzeugung von Graphiken in einen Datei-Device. In der anschließenden Untersektion werden die Transformationsfunktionen zur Überführung von Weltkoordinaten in Picture-Koordinaten beschrieben. Die letzte Untersektion stellt eine Funktion zur Bereinigung der Umgebung für die Picture-Infos bereit.

4.1 `pic.env` als Datenspeicher

Alle wichtigen Daten werden während des Arbeitsprozesses auf dem Objekt `pic.info` in der Umgebung `pic.env` abgelegt. Damit ist ein Informationsaustausch gewährleistet. Ggf. wird die Umgebung neu erzeugt und ein Link auf sie mit einer starken Zuweisung `<<-` auf `pic.env` abgelegt.

15 ⟨*define storage for pic.info* 15⟩ ≡ ⊂ 18
```
if(!exists("pic.env")) pic.env <<- new.env()
```

Die Speicherung von Bildinfos erfordert eine explizite Zuweisung.

16 ⟨*store* `pic.info` *in* `pic.env` 16⟩ ≡ ⊂ 18, 20, 21, 22, 24, 25, 26, 27, 28, 29, 32, 33, 34
```
assign("pic.info", pic.info, envir=pic.env)
```

Ein Zugriff auf die Bildinfos ergänzt das Info-Management.

17 ⟨*get* `pic.info` *from environment* 17⟩ ≡ ⊂ 20, 21, 22, 24, 25, 26, 27, 28, 29, 32, 33, 34
```
pic.info <- get("pic.info", envir=pic.env)
```

4.2 Die Verpackung: `anno.open.annotations` und `anno.close.annotations`

Die Funktion `anno.open.annotations` erstellt einen TeX-Code, der auf Wunsch eine Präamble `head!=FALSE` enthält. Es ist schwer zu sagen, was ein Kopf alles so leisten muss. Deshalb wird an dieser Stelle ein sehr einfacher Vorschlag gemacht, bei dem als Pakete `graphicx, pict2e, rotating, fontenc` geladen werden. Für den Fall, dass ein Anwender einen eigenen Kopf zur Hand hat, kann er diesen nutzen. Dann muss dem Argument `head` der Kopf übergeben werden. Werden der Funktion `width` und `height` (in mm) übergeben, werden diese per `\setkeys` als Bildbreite bzw. Bildhöhe umgesetzt. Andernfalls werden Weite und Höhe auf `.8\textwidth` gesetzt, aber nur dann, wenn in dem TeX-Dokument diese Parameter noch nicht gesetzt worden sind.

Für Skalierungsoperationen von y-Werten wird die TeX-Länge `\yvalue` eingeführt. Als Operationen wird später der Quotient von Längen verwendet, und es werden `\DivideLengths, \myratio, \scaley, \gety` definiert. Falls mit LaTeX Vektoren gezeichnet werden sollen, sind weiterhin `\yvaluea` als Längenspeicher sowie die Makros `\cle@n@, \xsplitstringa, \scaleya, \getya` erforderlich. Zur Funktion der verwendeten Makro- Konstruktionen sei auf Internet-Quellen verwiesen, die man über die Namen der Makros findet.

Weiter lassen sich das Fontencoding `fontencoding`, ein Filename `file` und ein Pfad `path` festlegen. Wird das Argument `vector` auf `TRUE` gesetzt, werden Vorbereitungen getroffen, damit der Anwender auch LaTeX-Pfeile erzeugen kann. Als letztes Argument teilt der Anwender über das Argument `Sweavepic` mit, dass er eine Sweave-Verarbeitung im Sinn hat. Hat dieses

Argument den Wert `TRUE`, werden die Ausgaben Sweave-kompatibel erzeugt. Wird dem Argument ein Dateiname übergeben, wird dieser für die Bilddateien verwendet.

18 ⟨*define* anno.open.annotations 18⟩ ≡ ⊂ 10

```
anno.open.annotations <- function(head=TRUE, tail=TRUE, width=100, height=100,
        fontencoding="T1", Sweavepic=FALSE, file="console", path="", vector=TRUE){
  ⟨define storage for pic.info 15⟩
  latex.code <- NULL
  if(head[1]!=FALSE){                  # if preamble has to be construted
    if(head[1]!=TRUE){ latex.code <- c(latex.code,head) }else{
      latex.code <- c(latex.code,
        "\\documentclass[a4paper]{article}\n",
        "\\usepackage{graphicx}\\usepackage{pict2e}\\usepackage{rotating}\n",
        paste("\\usepackage[",fontencoding,"]{fontenc}\n",sep=""),
        "\\begin{document}\n",
        "")
    }
  }
  latex.code <- c(latex.code,                      # width
      if(width!="") paste(sep="","\\setkeys{Gin}{width=",width,"mm}\n"),
      "\\makeatletter\\if!\\Gin@ewidth \\setkeys{Gin}{width=.8\\textwidth}",
      "\\else \\relax\\fi\\makeatother\n",
      "\\ifdefined\\ginwidth \\relax \\else \\newlength{\\ginwidth}\\fi",
      "\\makeatletter\\ginwidth=\\Gin@ewidth\\makeatother\n")
  latex.code <- c(latex.code,                      # height
      if(height!="") paste(sep="","\\setkeys{Gin}{height=",height,"mm}\n"),
      "\\makeatletter\\if!\\Gin@eheight \\setkeys{Gin}{height=.8\\textwidth}",
      "\\else \\relax\\fi\\makeatother\n",
      "\\ifdefined\\ginheight \\relax \\else \\newlength{\\ginheight}\\fi",
      "\\makeatletter\\ginheight=\\Gin@eheight\\makeatother\n")
  latex.code <- c(latex.code,                      # yvalue
      "\\makeatletter\\def\\getlength#1{\\strip@pt#1}\\makeatother\n",
      "\\ifcsname yvalue\\endcsname \\relax\\else\\newlength{\\yvalue}\\fi\n",
      "\\makeatletter\\def\\DivideLengths#1#2{\n",
      paste(sep="","\\strip@pt\\dimexpr\\number\\numexpr\\number\\dimexpr#1",
            "\\relax*65536/\\number\\dimexpr#2\\relax\\relax sp\\relax\n"),
      "}\\makeatother\n",
      "\\def\\myratio{\\number\\DivideLengths{\\ginheight}{\\ginwidth}}\n",
      paste(sep="","\\def\\scaley#1{\\setlength{\\yvalue}{#1 pt}\\global",
            "\\setlength{\\yvalue}{\\myratio\\yvalue}}\n"),
      "\\def\\gety{\\getlength{\\yvalue}}")
  if(vector){
    latex.code <- c(latex.code,
      "\\makeatletter\n",
      "\\def\\cle@n@#1>#2{#2}\n", "\\def\\xsplitstringa#1{%\n",
      "\\ifnum\\pdfmatch subcount 5 {((.+)[.])(.*)}{#1}=1 %\n", # %First:
      "\\expandafter\\cle@n@\\pdflastmatch2 %\n",
      #2nd:%\expandafter\cle@n@\pdflastmatch3
      "\\else\\fi\n}\n", "\\makeatother\n",
      "\\ifcsname yvaluea\\endcsname \\relax\\else\\newlength{\\yvaluea}\\fi\n",
      paste(sep="","\\def\\scaleya#1{\\setlength{\\yvaluea}{#1 pt}\\global",
            "\\setlength{\\yvaluea}{\\myratio\\yvaluea}}\n"),
      "\\def\\getya {\\xsplitstringa{\\getlength{\\yvaluea}}} \n")
  }
  pic.info <- list(latex.code=latex.code, head=head, tail=tail, fontencoding=fontencoding,
                   file=file, path=path, vector=vector, width=width,
                   height=height, Sweavepic=Sweavepic, designsize=100)
  if(pic.info$Sweavepic == FALSE && pic.info$file != ""){
    cat(paste("%",pic.info$file,"generated by annotation tool\n"), file=pic.info$file)
  }
  pic.info$pic.counter <- 0
  pic.info$latex.code <- latex.code
  ⟨generate Sweave-output 35⟩
  ⟨output pic.info$latex.code to file 23⟩
  ⟨store pic.info in pic.env 16⟩
  ⟨erzeuge unsichtbare Ausgabe 19⟩
} # end of anno.open.annotations()
```

Man sieht: Je nach Parameter-Konstellation wird ein entsprechender LaTeX-Code erzeugt. Ein Bildzähler wird auf 0 gesetzt und dann wird der Code im Sweave-Fall oder sonstwie ausgegeben. Zum Schluss erfolgt eine Sicherung wichtiger Informationen in der Umgebung.

Sowohl bei der Verarbeitung mit Sweave als auch in den anderen Fällen wird der Output per Schreibanweisung mit `cat()` ausgegeben. Deshalb returnen die verschiedenen Funktionen keine Ergebnisse bzw. nur unsichtbare Leere per `return()` aus.

19 ⟨*erzeuge unsichtbare Ausgabe* 19⟩ ≡ ⊂ 18, 20, 21, 22, 24, 25, 26, 27, 28, 31
```
return(invisible())
```

`anno.close.annotations.` Da es in einem Dokument mehrere Bilder geben kann, erfordert der letzte Abschluss eine weitere Funktion. In dieser werden alle noch nicht verarbeiteten LaTeX-Befehle ausgegeben.

20 ⟨*define* `anno.close.annotations` 20⟩ ≡ ⊂ 10
```
anno.close.annotations <- function(){ # end of picture
  ⟨get pic.info from environment 17⟩
  latex.code <- pic.info$latex.code
  if(pic.info$tail==TRUE){
    latex.code <- c(latex.code, "\\end{document}\n")
  } else {
    if(pic.info$tail!=FALSE) latex.code <- c(latex.code, "\n", pic.info$tail, "\n")
  }
  pic.info$latex.code <- latex.code
  ⟨generate Sweave-output 35⟩
  ⟨output pic.info$latex.code to file 23⟩
  ⟨store pic.info in pic.env 16⟩
  ⟨erzeuge unsichtbare Ausgabe 19⟩
}
```

Infos holen, Code konstruieren, Ausgaben umsetzen und Informationen sichern, bildet den Rahmen der gerade eingeführten Funktion. Dieses Grundgerüst werden wir im Folgenden wiederholt antreffen.

4.3 Der Bilderrahmen: `anno.begin.picture` und `anno.end.picture`

`anno.begin.picture` Die Funktion `anno.begin.picture` leitet den Beginn einer LaTeX-Graphik ein. Zuerst wird die eingestellte `unitlength` gesichert. Hierzu wird als neue Länge `saveunitlength` – sofern noch unbekannt – eingeführt, auf der dann die Einheit abgelegt wird. Für den Fall, dass der Funktion `anno.begin.picture` eine Bildhöhe oder Bildbreite übergeben werden, werden diese mit Hilfe von `\setkeys` bei der Formatierung umgesetzt. Diese Setzung ist dann solange wirksam, bis eine neue Setzung erfolgt.

Damit die verwendeten Koordinaten übersichtlich bleiben, wird die aktuell eingestellte Bildbreite in 100 Einheiten zerlegt. Falls eine `fbox` um die Abbildung verlangt wird, wird deren Anfangsklammer definiert.

Für die Beginn-Anweisung einer Picture-Umgebung sind deren Breite und Höhe in den eingestellten Einheiten anzugeben. Die Breite ist auf 100 Einheiten fixiert. Die passende Anzahl an Einheiten für die Höhe ergibt sich für beliebige Maßeinheiten aus der Formel 100*Höhe/Breite. Diese Operation wird erledigt, indem der Länge `yvalue` 100 zugewiesen wird, dann wird diese Größe mittels `\scaley` passend skaliert, mit `\gety` abgefragt und in die Headerzeile der Picture-Umgebung eingebaut.

21 ⟨*define* `anno.begin.picture` 21⟩ ≡ ⊂ 10
```
anno.begin.picture <- function(fbox = TRUE, width="", height=""){
  ⟨get pic.info from environment 17⟩
  latex.code <- pic.info$latex.code
  # save unitlength
  latex.code <- c(latex.code,
```

```
      "\\ifcsname saveunitlength\\endcsname",
      "  \\setlength{\\saveunitlength}{\\unitlength}\n",
      "\\else\\newlength{\\saveunitlength}",
      "  \\setlength{\\saveunitlength}{\\unitlength}",
      "\\fi\n")
  # reset width and height
  if(width!=""){
    pic.info$width <- width
    latex.code <- c(latex.code,
                    paste("\\setkeys{Gin}{width=",width,"mm}\n",
                          "\\makeatletter\\ginwidth=\\Gin@ewidth\\makeatother\n",
                          sep=""))
  }
  if(height!=""){
    pic.info$height <- height
    latex.code <- c(latex.code,
                    paste("\\setkeys{Gin}{height=",height,"mm}\n",
                          "\\makeatletter\\ginheight=\\Gin@eheight\\makeatother\n",
                          sep=""))
  }
  # set unitlength
  latex.code <- c(latex.code,"\\unitlength=0.01\\ginwidth\n")
  # define frame box
  latex.code <- c(latex.code, if(fbox) "\\fbox{\n" else "{\n") # }}
  # define begin of picture
  latex.code <- c(latex.code,  " \\scaley{100}",
                  paste(sep="","  \\begin{picture}(100,\\gety)\n"))
  pic.info$pic.counter <- pic.info$pic.counter + 1
  pic.info$latex.code <- latex.code
  ⟨generate Sweave-output 35⟩
  ⟨output pic.info$latex.code to file 23⟩
  ⟨store pic.info in pic.env 16⟩
  ⟨erzeuge unsichtbare Ausgabe 19⟩
} # anno.open.annotations(); anno.begin.picture()
```

Die Klammer der Picture-Umgebung gilt es wieder zu schließen. Das leistet die Funktion `anno.end.picture`. Sie restauriert auch die gespeicherte `unitlength`.

22 ⟨*define* `anno.end.picture` 22⟩ ≡ ⊂ 10

```
anno.end.picture <- function(){ # end of picture
  ⟨get pic.info from environment 17⟩
  latex.code <- c(pic.info$latex.code," \\end{picture}\n")
  latex.code <- c(latex.code,"}\n")   # frame box end
  latex.code <- c(latex.code,"\\setlength{\\unitlength}{\\saveunitlength}","\\par{} \n")
  pic.info$latex.code <- latex.code
  ⟨generate Sweave-output 35⟩
  ⟨output pic.info$latex.code to file 23⟩
  ⟨store pic.info in pic.env 16⟩
  ⟨erzeuge unsichtbare Ausgabe 19⟩
}
```

4.4 Ausgabe von LaTeX-Anweisungen

An dieser Stelle bietet es sich an, die Ausgabe der generierten LaTeX-Kommandos auszuformulieren.

23 ⟨*output* `pic.info$latex.code` *to file* 23⟩ ≡ ⊂ 18, 20, 21, 22, 28

```
if(pic.info$Sweavepic == FALSE){
  latex.code <- pic.info$latex.code
  latex.code <- latex.code[latex.code != ""]
  if(pic.info$file!=""){
    base::cat(latex.code, file=pic.info$file, append=TRUE)
  } else { cat(latex.code) }
  pic.info$latex.code <- latex.code <- NULL
}
```

4.5 Texte ins Bild setzen

Für die Einbringung von Texten sind als wesentliche Parameter die Weltkoordinaten (`x` und `y`) und die Texte `txt` selbst festzulegen. Weiterhin könnte die Zeichengröße `txt.chsize`, Rotation `txt.rot` und eine Adjustierung `txt.adj` interessant sein. Diese Größen können bzw. müssen als Parameter an die Funktion zur Textintegration übergeben werden. Notwendig für die Umrechnung der Koordinaten sind die Funktionen `pic.info$transform.x` und `pic.info$transform.y`, die dem Informationsobjekt entnommen werden. Diese Transformationen müssen die konkrete Realisation des Bildes berücksichtigen und erfordern eine Sicherung der relevanten graphischen Parameter während des Generierungsprozesses. Näheres entnehme man den Erklärungen zu den Funktionen `anno.copy.R.plot` bzw. `anno.transformation.fns`.

24 ⟨*define* `anno.text` 24⟩ ≡ ⊂ 10

```
anno.text <- function(x, y, txt=NULL, txt.chsize = "\\footnotesize{}",
                      txt.adj="", txt.rot=""){
  # this fn includes annotations defined by positions in user coordinates
  if(0 == length(txt)) return()
  ⟨get pic.info from environment 17⟩
  if(txt.adj=="") txt.adj <- rep("c",length(txt))
  if(txt.rot=="") txt.rot <- rep(0,length(txt))
  xt <- pic.info$transform.x(x);   yt <- pic.info$transform.y(y);
  latex.code <- pic.info$latex.code
  for(i in seq(along=xt))
    latex.code <- c(latex.code,
      paste(sep="","  \\scaley{",round(yt[i],digits=2),"}",
            "\\put(",round(xt[i],digits=2),",\\gety)",
              "{\\makebox(0,0)[",txt.adj[i],"]{",txt.chsize,
                "\\rotatebox{",txt.rot,"}{",txt[i],"}}}\n"
           # or: "\\begin{rotate}{",txt.rot,"}{",txt[i],"}\\end{rotate}"
      ))
  pic.info$latex.code <- latex.code
  pic.info <- c(pic.info, list(txt.chsize=txt.chsize, txt.adj=txt.adj, txt.rot=txt.rot))
  ⟨store pic.info in pic.env 16⟩
  ⟨erzeuge unsichtbare Ausgabe 19⟩
} # anno.text(pic.info2, 15, 10, "hallo")
```

4.6 Pfeile ins Bild schießen

Natürlich lassen sich mit R Pfeile platzieren. Jedoch könnte es wünschenswert sein, diese Aufgabe an LaTeX zu übertragen. Diese einfache Idee erfordert zusätzlichen Aufwand, der bei der Einbindung des LaTeX-Paketes `pict2e` beginnt. Auch wird ein weiteres Längenregister mit zugehörigen Makros erforderlich. Letztlich muss es eine Funktion geben, mit der solche Pfeile definiert werden können. Der Funktion `anno.vector` sind die Anfangs- und Endpunkte als Vektor mit 4 Elementen oder aber als 4-spaltige Matrix zu übergeben. Dann werden die relevanten Bildparameter beschafft und mit diesen die Koordinaten transformiert. Für TeX ist neben dem Anfangspunkt die Länge wie auch die Steigung in einer delta-x-delta-y-Form anzugeben. Aus Sicht der Weltkoordinaten lassen sich die Deltas einfach gewinnen. Jedoch müssen sie ebenfalls durch einen Transformationsprozess umgerechnet werden. Mit `pict2e` können für die Richtungsangaben Werte zwischen -999 und 999 eingesetzt werden. Hier werden die Werte auf einen Bereich von -420 bis 420 skaliert, damit später durch Bildvergrößerung maximal noch ein Faktor von etwas größer als 2 umgesetzt werden kann. Mittels `\scaley` und `\scaleya` werden die Transformationen bei der Formatierung realisiert.

25 ⟨*define* anno.vector 25⟩ ≡ ⊂ 10

```
anno.vector <- function(x1, y1, x2, y2){
  ⟨get pic.info from environment 17⟩
  din <- pic.info$din; mai <- pic.info$mai; mar <- pic.info$mar; usr <- pic.info$usr
  transform.x <- pic.info$transform.x; transform.y <- pic.info$transform.y
  if(is.matrix(x1)) arrowmat <- x1 else arrowmat <- cbind(x1, y1, x2, y2)
  # 4 columns: x1, y1, x2, y2
  arrowmat <- rbind(arrowmat)
  # transform to 0-100-coordinates
  arrowmat[,c(1,3)] <- transform.x(arrowmat[,c(1,3)])
  arrowmat[,c(2,4)] <- transform.y(arrowmat[,c(2,4)])
  # find length in direction x
  arrowmat <- cbind(arrowmat, abs(arrowmat[,3]-arrowmat[,1]))
  arrowmat[,5] <- ifelse(arrowmat[,5] == 0, # vertical vector ??
                         abs(arrowmat[,4]-arrowmat[,2]), arrowmat[,5])
  arrowmat[,c(3,4)] <- arrowmat[,c(3,4)]-arrowmat[,c(1,2)] # find deltas
  # scaling to 420 is done to be able to rescale by factor 2.0 + eps
  arrowmat[,c(3,4)] <- floor(420*arrowmat[,c(3,4)]/max(abs(arrowmat[,c(3,4)])))
  # arrowmat contains 5 columns: x1, y1, direction x, direction y, length in x
  latex.code <- pic.info$latex.code
  for(i in seq(along=arrowmat[,1])){
    if(0!=arrowmat[i,3]){ put.cmd <- c(
                  paste("  \\scaley{", round(arrowmat[i,2],digits=2),"}",
                        "\\scaleya{",0.001+round(arrowmat[i,4],digits=2),"}",
                        "\\put(",round(arrowmat[i,1],digits=2),",\\gety){",
                        "\\vector(",arrowmat[i,3],",\\getya){",
                        round(arrowmat[i,5],digits=2),"}}\n",sep="") )
    } else { put.cmd <- c(
                  paste("  \\scaley{", round(arrowmat[i,2],digits=2),"}",
                        "\\scaleya{",0.001+abs(round(arrowmat[i,5],digits=2)),"}",
                         "\\put(",round(arrowmat[i,1],digits=2),",\\gety){",
                        "\\vector(",arrowmat[i,3],",",arrowmat[i,4],")",
                              "{\\getya}}\n",sep="") )
    }
    latex.code <- c(latex.code, put.cmd)
  }
  pic.info$latex.code <- latex.code
  ⟨store pic.info in pic.env 16⟩
  ⟨erzeuge unsichtbare Ausgabe 19⟩
}
```

4.7 Titel und Skalen LaTeXen

Eine wichtige Aufgabe besteht in der Anbringung von Namen, Titeln und Achsenbeschriftungen. Hierzu soll die Funktion `anno.ticklabels` die passenden Rotationen (`yticklabels.rot`) und Positionierungen (`xticklabels.pos` bzw. `yticklabels.pos`) für Tick-Beschriftungen (`xticklabels` bzw. `yticklabels`) erledigen. Beispielsweise können so verschiedene Boxplots am Rand mit einem Namen versehen werden. Die Orte werden durch die ganzzahligen Stellen der Achsen definiert, sofern keine Positionen angegeben wurden. Die orthogonale Entfernung von einer Achse wird durch Bildparameter wie `mai` oder `mar` beeinflusst. Für Bezeichnungen an der x-Achsen erfordert die Positionierung in x-Richtung die Transformationsfunktion `transform.x` und in der anderen Richtung müssen Infos über den Rand ausgewertet werden. `height0..100` hält die Höhe des 0-100-normierten Gerätes, also 100.

26 ⟨*define* anno.ticklabels 26⟩ ≡ ⊂ 10

```
anno.ticklabels <- function(xticklabels="", yticklabels="", xticklabels.pos="", yticklabels.pos="",
                            yticklabels.rot=90, xyticklabels.chsize="\\footnotesize"){
  ⟨get pic.info from environment 17⟩
  latex.code <- pic.info$latex.code
  if(xticklabels[1]!=""){
    x <- if(xticklabels.pos[1]=="") 1:length(xticklabels) else xticklabels.pos
    x <- pic.info$transform.x(x)
```

```
  height0..100 <- 100 # 0-100-normalized coordinates
  y <- 3.6*pic.info$mai[1]/pic.info$mar[1]*height0..100/pic.info$din[2]
  latex.code <- c(latex.code,
                  paste("  \\scaley{",round(y,digits=2),"}",
                        "\\put(",round(x,digits=2),",\\gety){",
                        "\\makebox(0,0){",xyticklabels.chsize," ",xticklabels,"}}\n",sep=""))
}
if(yticklabels[1]!=""){
  y <- if(yticklabels.pos[1]=="") 1:length(yticklabels) else yticklabels.pos
  y <- pic.info$transform.y(y)
  x <- 2.6*pic.info$mai[2]/pic.info$mar[2]*pic.info$width/pic.info$din[1]
  latex.code <- c(latex.code,
                  paste("  \\scaley{",round(y,digits=2),"}",
                        "\\put(",round(x,digits=2),",\\gety){",
                        "\\makebox(0,0)[c]{\\rotatebox{",yticklabels.rot,"}",
                        "{",xyticklabels.chsize," ",yticklabels,"}}}\n",sep=""))
}
pic.info$latex.code <- latex.code
⟨store pic.info in pic.env 16⟩
⟨erzeuge unsichtbare Ausgabe 19⟩
}
# anno.ticklabels(pic.info2, paste("X-Werte",1:3))
```

Für die Achsenbeschriftungen und Titel gibt es voreingestellte Orte. Diese sollen auch für LaTeX-Beschriftungen gelten. Für die Anbringung des Titels wird der Mittelwert der x-Bereichsgrenzen ermittelt und transformiert, der y-Wert folgt ausgehend vom oberen Rand.

27 ⟨define anno.title 27⟩ ≡ ⊂ 10

```
anno.title <- function(main="", sub="", xlab="", ylab="",
                       main.chsize="", sub.chsize="", lab.chsize=""){
  if(main.chsize=="") main.chsize <- "\\normalsize"
  if(sub.chsize=="")  sub.chsize <- "\\small"
  if(lab.chsize=="")  lab.chsize <- "\\footnotesize"
  ⟨get pic.info from environment 17⟩
  din <- pic.info$din; mai <- pic.info$mai; mar <- pic.info$mar; usr <- pic.info$usr
  transform.x <- pic.info$transform.x; transform.y <- pic.info$transform.y
  height0..100 <- 100 # height of R graphics in 0-100-normalized coordinates
  latex.code <- pic.info$latex.code
  if(main!=""){ # mai/mar for all directions identical
    x <- transform.x(0.5*(usr[1]+usr[2]))
    y <- height0..100 - 2.05*mai[3]/mar[3]*height0..100/din[2]
    latex.code <- c(latex.code,
          paste("  \\scaley{",round(y,digits=2),"}",
                "\\put(",round(x,digits=2),",\\gety){",
                "\\makebox(0,0){",main.chsize," ",main,"}}\n",sep=""))
  }
  if(sub!=""){
    x <- transform.x(0.5*(usr[1]+usr[2]))
    y <- 0.6*mai[1]/mar[1]*height0..100/din[2] # 2.6
    latex.code <- c(latex.code,
          paste("  \\scaley{",round(y,digits=2),"}",
                "\\put(",round(x,digits=2),",\\gety){",
                "\\makebox(0,0){",sub.chsize," ",sub,"}}\n",sep=""))
  }
  if(xlab!=""){
    x <- transform.x(0.5*(usr[1]+usr[2]))
    y <- 1.6*mai[1]/mar[1]*height0..100/din[2]
    latex.code <- c(latex.code,
          paste("  \\scaley{",round(y,digits=2),"}",
                "\\put(",round(x,digits=2),",\\gety){",
                "\\makebox(0,0){",lab.chsize," ",xlab,"}}\n",sep=""))
  }
  if(ylab!=""){
    y <- transform.y(0.5*(usr[3]+usr[4]))
    x <- 0.6*mai[2]/mar[2]*height0..100/din[1]
    latex.code <- c(latex.code,
          paste("  \\scaley{",round(y,digits=2),"}",
```

```
                "\\put(",round(x,digits=2),",\\gety){",
                "\\makebox(0,0)[c]{",lab.chsize,"\\rotatebox{90}",
                "{",ylab,"}}}\n",sep=""))
  }  # height of rot. box: *.5, y <- y - dy
  pic.info$latex.code <- latex.code
  ⟨store pic.info in pic.env 16⟩
  ⟨erzeuge unsichtbare Ausgabe 19⟩
}
```

4.8 Bilderfälscherei `anno.copy.R.plot`

Während der interaktiven Arbeit mit R bietet es sich an, das Bild als Datei per `dev.copy` zu erzeugen bzw. zu regenerieren. Dabei können verschiedene Parameter gewählt werden, um das Bild passend in Form zu bringen. Im Moment der Bildgenerierung müssen die relevanten Parameter gesichert werden, damit mit diesen eine geeignete Transformationsfunktion für Positionen erstellt werden kann.

Neben der Öffnung eines graphischen Device wird auch ein `\includegraphics`-Befehl in den LaTeX-Code integriert. Dabei wird davon ausgegangen, dass die linke untere Ecke des Bild auf dem Punkt $(0, 0)$ des Picture-Koordinatensystems zu liegen kommt.

Bei einer Sweave-Verarbeitung kommt diese Funktion nicht zum Einsatz.

Als Argumente werden übergeben: `picname` zur Spezifizierung eines Bildnamens, `designsize` zur Modifikation der Entwurfsgröße, `driver` für die Auswahl eines Graphik-Treibers, `pointsize` zur Beeinflussung der Zeichengröße.

28 ⟨*define* `anno.copy.R.plot` 28⟩ ≡ ⊂ 10

```
anno.copy.R.plot <- function(picname = "", designsize = "", driver = "",
                             pointsize = 12, DEBUG = TRUE){
  ⟨get pic.info from environment 17⟩
  # define name of graphics file ------------------------------------------------------
  if(!is.null(pic.info$picname)) picname <- sub("[-][0-9]{3}","",pic.info$picname)
  if(picname == "") picname <- "pict"
  picno <- pic.info$pic.counter                                  # get picture number
  # picname <- sub("[.][a-zA-Z]{2,3}$","",picname)               # remove extension
  picname <- paste(picname, sep="-", substring(1000 + picno,2))  # compose pic name
  pic.info$picname <- picname
  picname <- paste(sub(".tex$","",pic.info$file),picname,sep="-")  # add file name
  if(""!=pic.info$path) picname <- file.path(pic.info$path,picname)# add path name
  # define driver if not set ----------------------------------------------------------
  if(driver==""){ # set default driver according OS
    if(substring(version$os,1,6)=="darwin") driver <- "jpg"
    if(substring(version$os,1,5)=="linux")  driver <- "jpg"
    if((.Platform$OS.type=="windows"))      driver <- "postscript"
  }
  if(exists("DEBUG")) cat("% driver:",driver,"picname",picname)
  # check designsize ------------------------------------------------------------------
  if(designsize=="") designsize <- pic.info$designsize
  pic.info$designsize <- designsize
  # generate graphics file by driver --------------------------------------------------
  if(driver == "jpg"){
    picname <- paste(picname, ".jpg", sep="")
    # linux default: width = 480, height = 480 // 100=100mm->480=default mac // linux
    designsize <- designsize*4.8  # designsize <- designsize*150/25.4 ## ?? linux
    dev.copy(jpeg, picname, quality=100, pointsize=pointsize, units = "px",
             width=designsize, height=designsize)
  }
  if(driver == "postscript"){ # postscript()
    picname <- paste(picname, ".ps", sep="")
    # psdesignwidth<-psdesignheight <- designsize/15 ; pointsize <- 12/12*pointsize #== 12
    psdesignwidth <- psdesignheight <- designsize/12 ; pointsize <- 15/12*pointsize #== 15
    dev.copy(postscript, picname, horizontal=FALSE,
```

```
            width=psdesignwidth, height=psdesignheight, pointsize=pointsize)
  }
  if(driver == "pdf"){ # under construction
    picname <- paste(picname,".pdf",sep="")
    dev.copy(pdf, picname)
  }
  # construct transformation functions ---------------------------------------------
  ⟨construct and save transformation functions 32⟩
  # close dev.copy device
  dev.off()
  # store LaTeX include command ----------------------------------------------------
  latex.code <- c(pic.info$latex.code,
                  paste("  \\put(0,0){\\makebox{\\includegraphics{",
                  sub("[.][a-z]{2,3}$", "", picname), "}}}\n", sep=""))
  pic.info$latex.code <- latex.code # save information -------------------
  # pic.info$picname <- picname
  pic.info$driver <- driver
  ⟨generate Sweave-output 35⟩
  ⟨output pic.info$latex.code to file 23⟩
  ⟨store pic.info in pic.env 16⟩
  ⟨erzeuge unsichtbare Ausgabe 19⟩
}
# anno.copy.R.plot(pic.info)
```

4.9 Bildmalerei mit `anno.begin.R.plot` und `anno.end.R.plot`

Die Funktion `anno.begin.R.plot` öffnet einen Device und legt auch das Include-Kommando für LaTeX an. Diese Funktion ist insbesondere für Sweave erforderlich.

29 ⟨*define* `anno.begin.R.plot` 29⟩ ≡ ⊂ 10

```
anno.begin.R.plot <- function(picname = "", designsize = "", driver = "jpg",
                              pointsize = 12, DEBUG = FALSE){
  ⟨get pic.info from environment 17⟩
  # define name of graphics file ---------------------------------------------------
  # if(is.numeric(picname)) picname <- paste(sep="","-pict-",substring(1000+picname,2))
  if(!is.null(pic.info$picname)) picname <- sub("[-][0-9]{3}", "", pic.info$picname)
  if(picname == "") picname <- "pict"
  picno <- pic.info$pic.counter                                  # get picture number
  # picname <- sub("[.][a-zA-Z]{2,3}$","",picname)               # remove extension
  picname <- paste(picname, sep="-", substring(1000 + picno,2))  # compose pic name
  pic.info$picname <- picname
  file <- if(pic.info$Sweavepic==TRUE) "Sweave" else pic.info$file # get file name
  picname <- paste(sub(".tex$","",file), sep="-", picname)       # add file name
  if(""!=pic.info$path) picname <- file.path(pic.info$path,picname)# add path name
  # include graphics command -------------------------------------------------------
  latex.code <- c(pic.info$latex.code,
                  paste("  \\put(0,0){\\makebox{\\includegraphics{",
                        picname,"}}}\n",sep=""))
  pic.info$latex.code <- latex.code
  # check designsize ---------------------------------------------------------------
  if(designsize=="") designsize <- pic.info$designsize
  pic.info$designsize <- designsize
  # store pic.info -----------------------------------------------------------------
  ⟨store pic.info in pic.env 16⟩
  # construct graphics -------------------------------------------------------------
  if(driver == "jpg"){
    picname <- paste(picname, ".jpg", sep="")
      # linux default: width = 480, height = 480//100=100mm->480=default mac//linux
      designsize <- designsize*4.8
      jpeg(picname, quality=100, pointsize=pointsize, units = "px",
           width=designsize, height=designsize)
  }
  if(driver == "postscript"){ # postscript()
```

```
    picname <- paste(picname, ".ps", sep="")
    psdesignwidth <- psdesignheight <- designsize/12; pointsize <- 15/12*pointsize #== 15
    postscript(picname, horizontal=FALSE,
               width=psdesignwidth, height=psdesignheight, pointsize=pointsize)
  }
  return(invisible())
}
# anno.begin.R.plot()
```

Nach Bildfertigstellung gilt es, den Device wieder zu schließen.

30 ⟨*define* `anno.end.R.plot` 30⟩ ≡ ⊂ 10

```
anno.end.R.plot <- function(){
  ⟨construct and save transformation functions 32⟩
  dev.off()
  return(invisible())
}
```

4.10 Transformationsfunktionen

Bei der Bilderstellung, aber noch vor dem Schließen des Device, müssen die Transformationsfunktionen definiert werden.

Koordinaten zur Zustellung von Mitteilungen

Die Erstellung von Transformationsfunktionen kann in unterschiedlichen Zusammenhängen nötig sein. Deshalb wird ergänzend eine eigene Funktion geschaffen.

31 ⟨*define* `anno.transformtion.fns` 31⟩ ≡ ⊂ 10

```
anno.transformtion.fns <- function(){
  # usr- to 0-100-normalised coordinates
  ⟨construct and save transformation functions 32⟩
  ⟨erzeuge unsichtbare Ausgabe 19⟩
}
```

Notwendig für den Übergang von User- zu `picture`-Umgebungskoordinaten sind Transformationsprozesse. Diese werden hier definiert. Zur Zeit werden keine Setzung von `mfg, mfrow, mfcol` verarbeitet. Auch logarithmische Skalen sind noch ein ungelöstes Problem. Es werden wichtige graphische Parameter abgefragt und mit deren Hilfe Transformationsfunktionen geschaffen.

32 ⟨*construct and save transformation functions* 32⟩ ≡ ⊂ 28, 30, 31

```
  ⟨get pic.info from environment 17⟩
  par <- par(); usr <- par$usr; pin <- par$pin; mai <- par$mai
  omi <- par$omi; din <- par$din; mar <- par$mar
  transform.x <- function(x){
    xi <- (x-usr[1])/(usr[2]-usr[1]) # in 0-1-Viewport-Koordinaten
    xi <- xi*pin[1] # 0-1-Viewport in inch
    xi <- xi+mai[2] # add inner margin
    xi <- xi+omi[2] # add outer margin
    width0..100 <- 100 # max of 0-100-normalized coordinates
    xi <- width0..100*xi/din[1] # coordinates to 0-100-normalized coordinates
    xi
  }
  transform.y <- function(y){
    yi <- (y-usr[3])/(usr[4]-usr[3]); yi <- yi*pin[2]; yi <- yi+mai[1]
    height0..100 <- 100; yi <- yi+omi[1]; yi <- height0..100*yi/din[2]
  }
  pic.info$transform.x <- transform.x; pic.info$transform.y <- transform.y
  pic.info$din <- din; pic.info$mai <- mai
  pic.info$mar <- mar; pic.info$usr <- usr
  ⟨store pic.info in pic.env 16⟩
```

4.11 Platte putzen: `anno.clean.pic.info`

Gerade in der Entwicklungsphase des Papiers wurde es ab und zu erforderlich, bereinigende Löschoperationen umzusetzen. Hierfür sei die Funktion `anno.clean.pic.info()` vorgeschlagen.

33 ⟨*define* `anno.clean.pic.info` 33⟩ ≡ ⊂ 10

```
anno.clean.pic.info <- function(){
  if(!exists("pic.env")) return()
  ⟨get pic.info from environment 17⟩
  pic.info <- NULL
  ⟨store pic.info in pic.env 16⟩
  cat("% picinfo cleaned\n")
}
# anno.clean.pic.info()
```

5 Definition der Devise: All inclusive!

Jetzt haben wir eine Menge schöner Bausteine, mit denen wir leicht die oben schon demonstrierte Funktion `do.anno()` definieren können. Alle wichtigen Infos sind dieser Funktion über Parameter mitzugeben. Da schon die einzelnen Funktionen eine Vielzahl von Parametern besitzen, ist die Menge der Parameter der zusammenfassenden Funktion entsprechend groß.

Wie sich zeigt, ist die Struktur von `do.anno()` nicht weiter kompliziert. Das liegt auch daran, dass die schwierigen Fragen alle schon in den Bausteinen gelöst worden sind. In einem Rutsch legen wir diese umfassende Funktion unter dem Dateinamen `anno-aio.R` ab.

34 ⟨*define all in one function* `do.anno` 34⟩ ≡ ⊂ 1, 4, 6, 8, 39, 41, 43

```
do.anno <- function(plot.cmds, file="", path="", head=TRUE, tail=TRUE,
           driver="postscript", width="", height="", designsize="",
           x, y, txt = "", txt.chsize, txt.adj = "", txt.rot = "",
           main="", sub="", xlab="", ylab="",
           main.chsize = "", sub.chsize = "", lab.chsize = "",
           xticklabels = "", xticklabels.pos = "",
           yticklabels = "", yticklabels.pos = "", yticklabels.rot = 90,
           xyticklabels.chsize = "\\footnotesize",
           vec.x1 = "", vec.y1 = "", vec.x2 = "", vec.y2 = "",
           Sweavepic=FALSE, clean.pic.info=TRUE
  ){
  ⟨define Annotationsfunktionen 10⟩
  if(head != FALSE || Sweavepic != FALSE) {
    if(clean.pic.info){
      anno.clean.pic.info()
      anno.open.annotations(head=head, tail=tail, file=file, path=path, Sweavepic=Sweavepic)
    }
  }
  ⟨get pic.info from environment 17⟩
  if(tail != FALSE){
    pic.info$tail <- tail; pic.info$head <- head;
    ⟨store pic.info in pic.env 16⟩
  }
  if(width == "") width <- pic.info$width; if(height == "") height <- pic.info$height
  if(designsize == "") designsize <- pic.info$designsize
  anno.begin.picture(width=width, height=height)
  if(Sweavepic != FALSE) anno.begin.R.plot(driver=driver, designsize=designsize)
  eval( plot.cmds )
  if(Sweavepic == FALSE) anno.copy.R.plot(driver=driver, designsize=designsize)
  if(Sweavepic != FALSE) anno.end.R.plot()
  if(txt[1] != "")
    anno.text(x, y, txt, txt.chsize=txt.chsize,
              txt.adj=txt.adj, txt.rot=txt.rot)
  if(any(c(main, sub, xlab, ylab) != ""))
    anno.title(main=main, sub=sub, xlab=xlab, ylab=ylab,
               main.chsize = main.chsize,
               sub.chsize = sub.chsize,
               lab.chsize = lab.chsize)
  if(any(c(xticklabels[1],yticklabels[1])!= ""))
    anno.ticklabels(xticklabels = xticklabels,
                    xticklabels.pos = xticklabels.pos,
                    yticklabels = yticklabels,
                    yticklabels.pos = yticklabels.pos,
                    yticklabels.rot = yticklabels.rot,
                    xyticklabels.chsize = xyticklabels.chsize)
  if(vec.x1[1] != "")
    anno.vector(vec.x1, vec.y1, vec.x2, vec.y2)
  anno.end.picture()
  if(tail!=FALSE) anno.close.annotations( )
}
dump("do.anno",file="anno-aio.R")
```

Nach dem Studium dieser Definition möge der Leser wieder zu der Sektion zurückblättern, in der Tests von `do.anno()` beschrieben und durchgeführt worden sind.

6 Sweave – geht das schief?

Bei der Erstellung von Bildern mit Sweave müssen die Parameter während der Abarbeitung eines Code-Chunks abgefragt werden. Sweave arbeitet bei `fig=true` einen Code-Chunk nach folgendem Schema ab: Zuerst wird ein graphischer Device geöffnet, dann werden die Anweisungen des Chunk umgesetzt und zum Schluss erfolgt ein `dev.off()`-Befehl. Damit könnte folgende Konstruktion funktionieren:

```
@
<init chunk, results=tex>=
 anno.open.annotations(head=FALSE, file="") # some definitions
   anno.begin.picture()                # begin picture and so on
@
<code chunk header, fig=true>=
     plot-cmds
     anno.define.transform.fns()         # find out the transformations
     anno.text(2, 5, "hallo")            # annotations
     anno.title(main="MAIN-Titel")       # further annotations
   anno.end.picture()
@
```

Leider werden die Chunks eingeklammert, so dass das Splitten einer Picture-Umgebung nicht ohne weitere Handstände funktioniert. Als praktizierbarer Ausweg werden wir die Graphik unabhängig von dem Sweave-Automatismus erzeugen und erhalten ungefähr folgendes Schema:

```
@
<init chunk, results=tex>=
 anno.open.annotations(head=FALSE, file="") # some definitions
@
<code chunk header, fig=true>=
   anno.begin.picture()              # begin picture and so on
     anno.begin.R.plot               # selection of device parameter and file name
       plot(1:6)
     anno.end.R.plot()               # storing the graphics parameter and dev.off
     anno.text(2, 5, "hallo")        # annotations
     anno.title(main="MAIN-Titel")   # further annotations
   anno.end.picture()
@
```

Hier ist ein Beispiel-File, der mit Sweave verarbeitet wurde bzw. werden kann.

```
% generated by annotation tool
\documentclass{article}
\usepackage{pict2e}
\begin{document}
@
Document-Beginn\par
Schritt 1: Vorbereitungen\par
<init chunk, results=tex, echo=false>=
# some definitions
source("anno.R")
anno.open.annotations(head=FALSE, file="", width=80, height=120, Sweavepic=TRUE)
@
Vorbereitungen abgeschlossen\par
Schritt 2: Chunk mit Bild\par
<code chunk header, fig=false, results=tex, echo=false>=
  chsize <- c("\\normalsize", "\\footnotesize", "\\tiny")[2]
  anno.begin.picture(width=80, height=60)
    anno.begin.R.plot()
      plot(1:6, xes=FALSE, ylab="", xlab="", main="<-------------------->")
      axis(1, labels=FALSE); axis(2, labels=FALSE)
      p <- par(); abline(h=mean(p$usr[3:4]), v=mean(p$usr[1:2]))
    anno.end.R.plot()
```

```
    anno.text(2, 5,   "hallo", txt.adj="c", txt.rot=45, txt.chsize=chsize)
    anno.text(2, 4.5, "hallo", txt.adj="c", txt.rot=45, txt.chsize=chsize)
    anno.title(main="MAINTitel", sub=date(), xlab="XLAB", ylab="YLAB")
    anno.ticklabels(paste("X", 1:6, sep=""),
                    paste("Y", c("a","b","c")), yticklabels.pos=2*(1:3),
                    xyticklabels.chsize=chsize)
    anno.vector(2, 5, 6, 1); anno.vector(2, 5, 2, 2)
  anno.end.picture()
anno.close.annotations()
@
\par  Ende der Darbietung \par
@
Verarbeitung wurde erledigt durch: Sweave("tSweave.Snw")
\end{document}
```

Merke: Mit dem hier verfolgten Ansatz werden Bilddateien also nicht mehr von Sweave verwaltet. Größenparameter lassen sich aber dennoch im Text über `\setkey` setzen.

Sweave-Output-Generierung

Für Sweave gilt es, die schon beschriebenen Output-Anweisungen zu definieren: Wir geben die erarbeiteten LaTeX-Anweisungen einfach aus. Sweave wird es schon richten.

35 ⟨*generate Sweave-output* 35⟩ ≡ ⊂ 18, 20, 21, 22, 28

```
if(pic.info$Sweavepic != FALSE){
    # latex.code <- latex.code[latex.code != ""]
    latex.code[latex.code == ""] <- "\\par"
    cat(latex.code); latex.code <- NULL # using relax: a copy -> output field
    pic.info$latex.code <- latex.code <- NULL
}
```

Sweave-Testsektion

Sweave-Test: ein Bild mit den Bausteinen erzeugt Nun können wir den ersten Sweave-Test wagen.

36 ⟨* 3⟩+ ≡

```
⟨define Snw-File 37⟩
⟨define Annotationsfunktionen 10⟩
Sweave("tSweave.Snw")
⟨finde os-Typ 2⟩
if(os=="windows"){ shell( "echo q | pdflatex tSweave.tex") }                          # windows
if(os=="linux")    system("echo q | pdflatex tSweave.tex; o tSweave.pdf")              # linux
if(os=="mac")      system("echo q | /usr/texbin/pdflatex tSweave.tex; open tSweave.pdf") # mac
```

Der Snw-File für diesen Test wird mit folgendem Chunk erzeugt.

37 ⟨*define Snw-File* 37⟩ ≡ ⊂ 36

```
snwfile <- c(' % generated by annotation tool
 \\documentclass{article}
 \\usepackage{pict2e}
 \\begin{document}
 @
 Document-Beginn\\par
 Schritt 1: Vorbereitungen\\par'
,paste(sep="", ' <', '<init chunk, results=tex, echo=false>', '>=')
,' # some definitions
 source("anno.R")
 anno.open.annotations(head=FALSE, file="", width=80, height=120, Sweavepic=TRUE)
 \n
 @
 Vorbereitungen abgeschlossen\\par
 Schritt 2: Chunk mit Bild\\par'
```

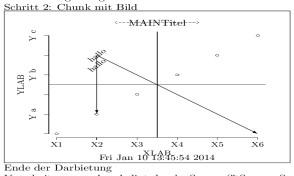
```
 ,paste(sep="", ' <', '<code chunk header, fig=false, results=tex, echo=false>', '>=')
,' chsize <- c("\\\\normalsize","\\\\footnotesize","\\\\tiny")[2]
   \n
   anno.begin.picture(width=80, height=60)
     anno.begin.R.plot()
       plot(1:6, axes=FALSE, ylab="", xlab="", main="<-------------------->")
       axis(1, labels=FALSE); axis(2, labels=FALSE) # abline(-1800,900)
       p <- par(); abline(h=mean(p$usr[3:4]), v=mean(p$usr[1:2]))
     anno.end.R.plot()
     \n \n
     anno.text(2, 5, "hallo", txt.adj="c", txt.rot=45, txt.chsize=chsize)
     anno.text(2, 4.5, "hallo", txt.adj="c", txt.rot=45, txt.chsize=chsize)
     anno.title(main="MAINTitel", sub=date(), xlab="XLAB", ylab="YLAB")
     anno.ticklabels(paste("X",1:6,sep=""),
                     paste("Y", c("a","b","c")), yticklabels.pos=2*(1:3),
                     xyticklabels.chsize=chsize)
     anno.vector(2, 5, 6, 1); anno.vector(2, 5, 2, 2)

   anno.end.picture()
 anno.close.annotations()
 \n
 @
 \\par  Ende der Darbietung \\par
 \n
 @
 Verarbeitung wurde erledigt durch: Sweave("tSweave.Snw")
 \\end{document}'
)
snwfile <- unlist(strsplit(snwfile, "\n")); snwfile <- substring(snwfile, 2)
cat(snwfile, file="tSweave.Snw", sep="\n")
```

Ergebnis Test6 Nach erfolgreicher Erledigung des Tests ist eine pdf-Datei entstanden, die wir hier einblenden:

38 ⟨ * 3⟩+ ≡

```
system("convert -density 200x200 tSweave.pdf test-outputs/test6.ps")
```

Document-Beginn
Schritt 1: Vorbereitungen
Vorbereitungen abgeschlossen
Schritt 2: Chunk mit Bild



Ende der Darbietung
Verarbeitung wurde erledigt durch: Sweave("tSweave.Snw")

1

Sweave-Testsektion mit All-in-One-Funktion Als zweites wollen wir versuchen, mit der All-in-One-Funktion ebenfalls zum Ziel zu kommen.

39 ⟨* 3⟩+ ≡
```
⟨define Snw-File-aio 40⟩
⟨define all in one function do.anno 34⟩
Sweave("tSweave-aio.Snw")
# Schritte der Verarbeitung
⟨finde os-Typ 2⟩
if(os=="windows"){ shell("echo q  | pdflatex tSweave-aio.tex") }                          # windows
if(os=="linux")    system("echo q | pdflatex tSweave-aio.tex; o tSweave-aio.pdf")        # linux
if(os=="mac")      system("echo q | /usr/texbin/pdflatex tSweave-aio.tex; open tSweave-aio.pdf") # mac
```

Der Snw-File für diesen Test ist so definiert.

40 ⟨define Snw-File-aio 40⟩ ≡ ⊂ 39
```
snwfile <- c(' % generated by annotation tool
 \\documentclass{article} \\usepackage{pict2e} \\usepackage{rotating}
 \\begin{document}
 @
 BEGINN\n
 erster Chunk\n'
 ,paste(sep="",' <','<init chunk, results=tex, echo=false>','>=')
 ,' # some definitions
 source("anno-aio.R")
 do.anno(
 # Plotanweisungen
      {
        plot(1:6, axes=FALSE, ylab="", xlab="")
        title(paste("<-", paste(rep(" ", 25), collapse=""), "->"))
        axis(1,labels=FALSE); axis(2, labels=FALSE)
        text(5, 2, "hello")
        abline(h=mean(par()$usr[3:4]), v=mean(par()$usr[1:2]))
      },
 # Text ins Bild
      x=2, y=5, txt="HELLO2", txt.chsize="\\\\large",
 # Titel
      main="maintitle2", sub=date(), xlab="X-lab2", ylab="Y-lab2",
      main.chsize="\\\\small", sub.chsize="\\\\tiny", lab.chsize="\\\\footnotesize",
 # Tickbeschriftungen
      xticklabels = paste("$X2_", 1:6, "$", sep=""),
      xticklabels.pos = "",
      yticklabels = paste("$Y2", c("a$", "b$", "c$"), sep="_"),
      yticklabels.pos = 2*(1:3), yticklabels.rot = "90",
      xyticklabels.chsize="\\\\tiny",
 # Zielobjekt
      file="",
      head=FALSE, tail=FALSE, driver="jpg", Sweavepic=TRUE,
 # Bildeigenschaften
      width=80, height=80, designsize=70
 # anno.open.annotations(head=FALSE, file="", width=80, height=120, Sweavepic=TRUE)
 )
 \n
 @
 AUS \n ENDE
 @
 Verarbeitung durch: Sweave("testSweave.Snw") \n \\end{document}'
)
snwfile <- unlist(strsplit(snwfile, "\n"))
snwfile <- substring(snwfile, 2)
cat(snwfile, file="tSweave-aio.Snw", sep="\n")
```

Zwei Bilder mit Sweave Als dritter Test sollen mittels der all-in-one Funktion nun noch zwei Bilder verarbeitet werden. Dazu werden wir den Dateinamen des letzten Tests verwenden.

41 ⟨* 3⟩+ ≡
```
⟨define Snw-File-aio Test 2 42⟩
⟨define all in one function do.anno 34⟩
Sweave("tSweave-aio.Snw")
# Schritte der Verarbeitung
⟨finde os-Typ 2⟩
if(os=="windows"){ shell("echo q  | pdflatex tSweave-aio.tex") }                          # windows
if(os=="linux")    system("echo q | pdflatex tSweave-aio.tex; o tSweave-aio.pdf")        # linux
if(os=="mac")      system("echo q | /usr/texbin/pdflatex tSweave-aio.tex; open tSweave-aio.pdf") # mac
```

Der Snw-File für diesen Test folgt, wie der Leser schon erwartet haben wird.

42 ⟨define Snw-File-aio Test 2 42⟩ ≡ ⊂ 41
```
snwfile <- c(' % generated by annotation tool
 \\documentclass{article}
 \\usepackage{pict2e}
 \\usepackage{rotating}
 \\begin{document}
 @
 BEGINN Bild 1\n
 erster Chunk\n \\par'
 ,paste(sep="", ' <', '<init chunk, results=tex, echo=false>', '>=')
 ,' # some definitions
 source("anno-aio.R")
 do.anno(
 # Plotanweisungen
      {
        plot(1:6, ylab="", xlab="", axes=FALSE)
        text(5, 2, "test2-bild1")
        axis(1); axis(2, labels=FALSE)
        abline(h=mean(par()$usr[3:4]), v=mean(par()$usr[1:2]))
      },
 # Text ins Bild
      x=2, y=5, txt="Sweave-aio-2", txt.chsize="\\\\large",
 # Titel
      main="2bilder-pic1", sub=date(), xlab="X-lab2", ylab="Y-lab2",
      main.chsize="\\\\small", sub.chsize="\\\\tiny", lab.chsize="\\\\footnotesize",
 # Tickbeschriftungen
      yticklabels = paste("$Y2", c("a$","b$","c$"), sep="_"),
      yticklabels.pos = 2*(1:3), yticklabels.rot = "90",
      xyticklabels.chsize="\\\\tiny",
 # Zielobjekt
      file="", path ="pic",
      head=FALSE, tail=FALSE, driver="jpg", Sweavepic=TRUE,
 # Bildeigenschaften
      width=70, height=70, designsize=70
 )
 \n
 @
 etwas Text
 @
 BEGINN Bild 2\n
 zweiter Chunk\n \\par'
 ,paste(sep="",' <','<init chunk, results=tex, echo=false>','>=')
 ,' # some definitions
 source("anno-aio.R")
 do.anno(
 # Plotanweisungen
      {
        plot(1:6, ylab="", xlab="", axes=FALSE)
        text(5, 2, "test2-bild2")
        axis(1, labels=FALSE); axis(2)
        abline(h=mean(par()$usr[3:4]), v=mean(par()$usr[1:2]))
      },
 # Text ins Bild
```

```
      x=2, y=5, txt="Sweave-aio-2", txt.chsize="\\\\large",
 # Titel
      main="zwei Bilder PIC 2", sub=date(), xlab="X-lab2", ylab="Y-lab2",
      main.chsize="\\\\small", sub.chsize="\\\\tiny", lab.chsize="\\\\footnotesize",
 # Tickbeschriftungen
      xticklabels = paste("$X_", 1:6, "$", sep=""),
      xyticklabels.chsize="\\\\tiny",
 # Zielobjekt
      file="",
      head=FALSE, tail=FALSE, driver="jpg", Sweavepic=TRUE, clean.pic.info=FALSE,
 # Bildeigenschaften
      # width=100, height=60, designsize=70
 )
 \n
 @
 \\rule{70mm}{1mm} \\par AUS ENDE SCHLUSS \n
 @
 Verarbeitung durch: Sweave("testSweave.Snw") \\end{document}'
)
snwfile <- unlist(strsplit(snwfile,"\n"))
snwfile <- substring(snwfile,2)
cat(snwfile,file="tSweave-aio.Snw",sep="\n")
"ok"
```

Die beiden Bilder von DT in einem Sweave-File

Zwei Bilder mit Sweave Als letzter Test werden die beiden DT-Beispiel-Bilder mittels der all-in-one Funktion und Sweave erzeugt.

43 ⟨* 3⟩+ ≡
```
⟨define Snw-File-aio Test mit 2 DT-Beispielen 44⟩
⟨define all in one function do.anno 34⟩
Sweave("tSweave-aio-2-DT.Snw")
# Schritte der Verarbeitung
⟨finde os-Typ 2⟩
if(os=="windows"){ shell("echo q  | pdflatex tSweave-aio-2-DT.tex") }                          # windows
if(os=="linux")    system("echo q | pdflatex tSweave-aio-2-DT.tex; o tSweave-aio-2-DT.pdf")   # linux
if(os=="mac")      system("echo q | /usr/texbin/pdflatex tSweave-aio-2-DT.tex; open tSweave-aio-2-DT.pdf") # mac
```

Der Snw-File für diesen Test wird in folgendem Code-Chunk definiert.

44 ⟨define Snw-File-aio Test mit 2 DT-Beispielen 44⟩ ≡ ⊂ 43
```
snwfile <- c(' % generated by annotation tool
 \\documentclass{article}
 \\usepackage{pict2e}
 \\usepackage{rotating}
 \\begin{document}
 @
 Eine Stichprobe aus einer $\\chi^2$-Verteilung per Histogramm dargestellt:\n
 \n \\par'
 ,paste(sep="",' <','<init chunk, results=tex, echo=false>','>=')
 ,' # some definitions
 source("anno-aio.R")
 do.anno(
 # Plotanweisungen
     {
       set.seed(7); x <- rchisq(1000,10)
       curve(dchisq(x, 10), xlim=c(0, qchisq(0.99, 10)), xlab="", ylab="")
       hist(x, freq=FALSE, add=TRUE, breaks=20); title(sub=date())
     },
  # Titel
      main="Dichte der $\\\\chi^2_{df=10}$-Verteilung und Histogramm",
      xlab="$t$", ylab="$\\\\hat f(t), f(t)$", sub=date(),
      main.chsize="\\\\small", sub.chsize="\\\\tiny", lab.chsize="\\\\footnotesize",
  # Zielobjekt
```

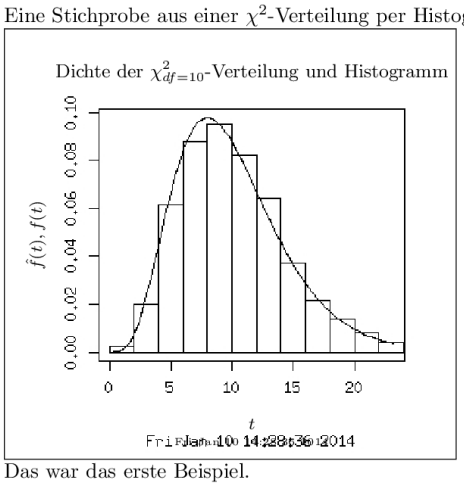
```
      file="", path="pic",
      head=FALSE, tail=FALSE, driver="jpg", Sweavepic=TRUE,
  # Bildeigenschaften
      width=70, height=70, designsize=70
)
\n
@
Das war das erste Beispiel. \\par
@
Beispiel 2: Integral-Approximation\n
\\par'
,paste(sep="",' <','<init chunk, results=tex, echo=false>','>=')
,' # some definitions
do.anno(
# Plotanweisungen
    {
      Delta <- 1/8
      curve(x^2, xlim=c(0,2), xlab="", ylab="")
      for(x in seq(0, 2-Delta, by=Delta)) rect(x, 0, x+Delta, (x+Delta)^2)
      for(x in seq(0, 2-Delta, by=Delta)) rect(x, 0, x+Delta, x^2)
      rect(1.25, 0,1.375, 1.25^2,  density=10)
      rect(1.25, 0,1.375, 1.375^2, density=10, angle=-45)
      # arrows(0.695976, 1.63574, 1.29813, 0.838917, length=0.07, angle=20,col=1)
      # arrows(1.14233,  3.05426, 1.30234, 1.75833,  length=0.07, angle=20,col=1)
    },
# Text ins Bild
     x=c(.5,1.2), y=c(2,3.3),
     txt=c("$\\\\Delta x \\\\cdot f(x_L)$", "$\\\\Delta x \\\\cdot f(x_R)$"),
     txt.chsize="\\\\large",
# Titel
     main="Integral-Approximation",
     xlab="$x$", ylab="$x^2$",  sub=date(),
     main.chsize="\\\\small", sub.chsize="\\\\tiny", lab.chsize="\\\\footnotesize",
# Pfeile
     vec.x1 = c(.6,1.2), vec.y1 = c(1.7,3), vec.x2 = c(1.3,1.3), vec.y2 =  c(1.0,1.7),
# Zielobjekt
     file="", head=FALSE, tail=FALSE, driver="jpg", Sweavepic=TRUE, clean.pic.info=FALSE,
# Bildeigenschaften
     width=80, height=80, designsize=70
)
\n
@
\\rule{70mm}{1mm} \\par Das war das zweite Bild. \n
@
Verarbeitung durch: Sweave("tSweave-aio-2-DT.Snw") \\end{document}'
)
snwfile <- unlist(strsplit(snwfile,"\n"))
snwfile <- substring(snwfile,2)
cat(snwfile, file="tSweave-aio-2-DT.Snw", sep="\n")
"ok"
```

Ergebnis Test7 Bei diesem Test kommt eine Datei heraus, die wir etwas abstrippen und dann einbinden können.

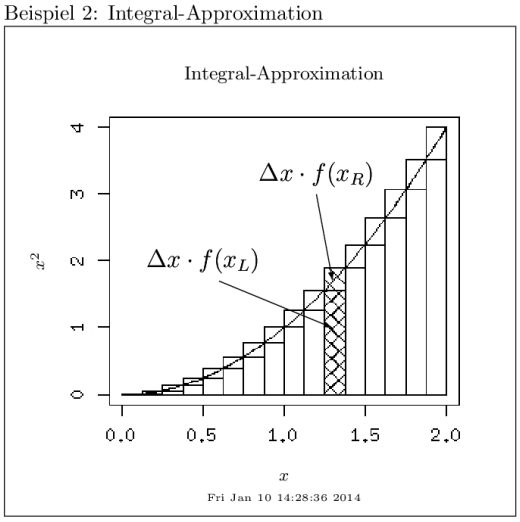
45 ⟨* 3⟩+ ≡

```
system("convert -density 200x200 tSweave-aio-2-DT.pdf test-outputs/test7.ps")
```

Eine Stichprobe aus einer χ^2-Verteilung per Histogramm dargestellt:



Das war das erste Beispiel.
Beispiel 2: Integral-Approximation



Das war das zweite Bild.
Verarbeitung durch: Sweave("tSweave-aio-2-DT.Snw")

1

... was wollen wir mehr?

7 Funktionsargumente, Object Index, Links

Die Argumente der Funktionen

46 ⟨* 3⟩+ ≡

```
h <- "                          "
for(f.name in ls(pattern="anno")) {
  x <- deparse(args(get(f.name))); x <- sub(".*function", "", x); x <- sub("NULL","",x)
  cat("--------------------------------\nargs(", f.name,")::\n") ; print(noquote(x))
}
```

```
--------------------------------
args( anno.begin.picture )::
[1]  (fbox = TRUE, width = "", height = "")
[2]
--------------------------------
args( anno.begin.R.plot )::
[1]  (picname = "", designsize = "", driver = "jpg", pointsize = 12,
[2]     DEBUG = FALSE)
[3]
--------------------------------
args( anno.clean.pic.info )::
[1]  ()
--------------------------------
args( anno.close.annotations )::
[1]  ()
--------------------------------
args( anno.copy.R.plot )::
[1]  (picname = "", designsize = "", driver = "", pointsize = 12,
[2]     DEBUG = TRUE)
[3]
--------------------------------
args( anno.end.picture )::
[1]  ()
--------------------------------
args( anno.end.R.plot )::
[1]  ()
--------------------------------
args( anno.open.annotations )::
[1]  (head = TRUE, tail = TRUE, width = 100, height = 100,
[2]     fontencoding = "T1", Sweavepic = FALSE, file = "console",
[3]     path = "", vector = TRUE)
[4]
--------------------------------
args( anno.text )::
[1]  (x, y, txt = , txt.chsize = "\\\\footnotesize{}",
[2]     txt.adj = "", txt.rot = "")
[3]
--------------------------------
args( anno.ticklabels )::
[1]  (xticklabels = "", yticklabels = "", xticklabels.pos = "",
[2]     yticklabels.pos = "", yticklabels.rot = 90, xyticklabels.chsize = "\\\\footnotesize")
[3]
--------------------------------
args( anno.title )::
[1]  (main = "", sub = "", xlab = "", ylab = "", main.chsize = "",
[2]     sub.chsize = "", lab.chsize = "")
[3]
--------------------------------
args( anno.transformtion.fns )::
[1]  ()
--------------------------------
args( anno.vector )::
[1]  (x1, y1, x2, y2)
```

Object Index

Links

http://tex.stackexchange.com/questions/558/is-there-a-show-for-lengths
http://tex.stackexchange.com/questions/15001/getting-length-as-number?rq=1
http://tex.stackexchange.com/questions/15001/getting-length-as-number