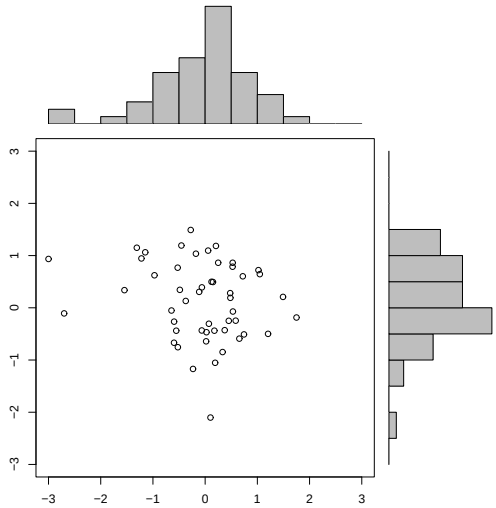
13 Seiten zur Funktion `layout()`

File: layout-function.rev
in: /home/wiwi/pwolf/lehre/rkurs/Datenanalyse/layout

16. November 2009



Inhalt

1 Die Funktion `layout()`

Die Funktion `layout()` gestattet es, ein graphisches Device schachbrettartig in Felder zu zerlegen und in jedem der Felder eine Graphik zu erzeugen. Die Größen der Felder können variieren, wobei jedoch die Grundstruktur aus Zeilen und Spalten erhalten bleiben muss. Weiterhin lässt sich steuern, in welcher Reihenfolge die Felder gefüllt werden.

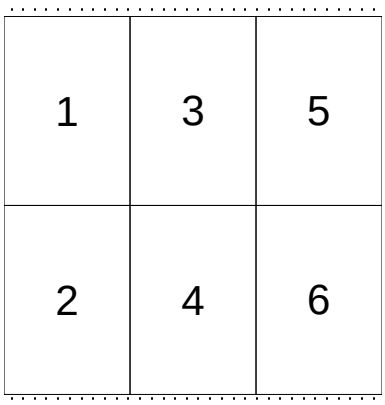
Neben der Funktion `layout()` wird in diesem Papier wesentlich nur noch die Funktion `layout.show()` verwendet, mit der man sich das gewählte Design anschauen kann. Zunächst werden wir die Definition des Schachbrettes, dann die Reihenfolge der Füllung der Schachbrettfelder mit Graphiken und zum Schluss die Manipulation von Größenverhältnissen diskutieren. Als Literaturhinweis sei hier besonders auf das Buch von Paul Murrell verwiesen, aus dem einige der Beispiele stammen.[1] Es sei darauf hingewiesen, dass der Mechanismus hinter `layout` sich nicht unbedingt mit Aufteilungen mittels `mfrow, mfcol, mfg` oder `split.screen` verträgt, so dass ggf. ein neues graphisches Device geöffnet werden sollte – siehe dazu auch die R-Hilfe.

2 Die Definition der Schachbrettgröße

Die Größe des Schachbretts wird durch die Dimensionen einer Matrix beschrieben. Beispielsweise legt eine (3×4)-Matrix eine Zerlegung des Device in 3 Zeilen und 4 Spalten fest. Die Zahlen der Matrix bestimmen die Reihenfolge der Felderfüllung.

1 ⟨* 1⟩ ≡

```
d.mat<-matrix(1:6,2,3)
layout(d.mat)
par(cex=4); layout.show(6); d.mat
```



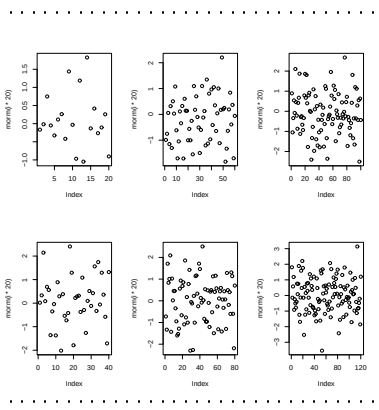

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

[1] P. Murrel (2006): R Graphics, Chapmann & Hall

Das schematische Bild ist mit der Funktion `layout.show` erstellt worden und zeigt uns das gewählte Aufteilungsschema. Interessiert uns zu erfahren, wie die Aufteilung nach der Erstellung von `6` Graphiken aussieht, übergeben wir der Funktion als Argument die Zahl `6`: `layout(6)`. Die eingezeichneten Zahlen zeigen die Felder für die Graphiken Nummer 1, 2, 3, 4, 5 und 6 an. An einem Beispiel wollen wir unser Layout überprüfen:

2 ⟨ * 1⟩+ ≡

```
d.mat<-matrix(1:6,2,3)
layout(d.mat)
for(i in 1:6) plot(rnorm(i*20))
```



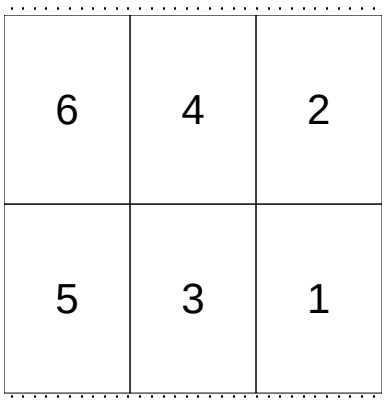
Diese einfache Verwendung der Layout-Funktion bildet genau das ab, was auch mit `mfrow` erzielbar ist. Insofern wird es erst im Folgenden spannend.

3 Füllungsreihenfolge

Die von der Funktion `layout.show` in die Felder eingetragen Zahlen geben die Füllungsreihenfolge an. Diese wird durch die Einträge in der Layout-Matrix (im Beispiel: `d.mat`) festgelegt. Siehe dazu ein Beispiel, bei dem wir die Reihenfolge umgedreht haben:

3 ⟨ * 1⟩+ ≡

```
d.mat<-matrix(6:1,2,3)
layout(d.mat)
par(cex=4); layout.show(6); d.mat
```



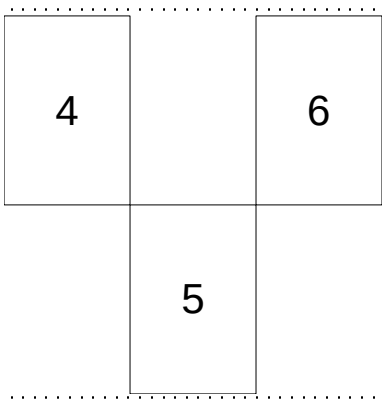

```
     [,1] [,2] [,3]
[1,]    6    4    2
[2,]    5    3    1
```

4 Leere Felder

Durch den Eintrag von 0-en bleiben die so gekennzeichneten Felder leer.

4 ⟨* 1⟩+ ≡

```
d.mat<-matrix(c(1,0,0,2,3,0),2,3)
layout(d.mat)
par(cex=4); layout.show(6)
d.mat
```




```
     [,1] [,2] [,3]
[1,]    1    0    3
[2,]    0    2    0
```

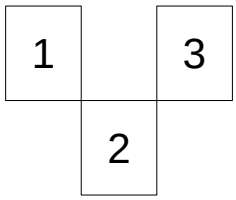
Da wir nur die Zahlen 1, 2 und 3 verwendet haben, stellt `layout` nur 3 der 6 Felder für Graphiken bereit. Die Funktion `layout.show` haben wir jedoch mit dem Input 6 bedient und wir erhalten das Schema für die Situation *6 High-Level-Funktionsaufrufe*: Nach dem dritten Plot wird die Tafel gewischt, und dann generiert R die restlichen drei Graphiken mit den Nummern 4, 5 und 6.

Mit Hilfe von 0-Zeilen und 0-Spalten lassen sich zusätzliche horizontale und vertikale Freiräume schaffen.

5 ⟨* 1⟩+ ≡

```
d.mat<-matrix(c(1,0,0,2,3,0),2,3)
d.mat<-rbind(0,d.mat,0); d.mat<-cbind(0,d.mat,0)
layout(d.mat)
par(cex=4); layout.show(3)
d.mat
```

. . . . . .



. . . . . .

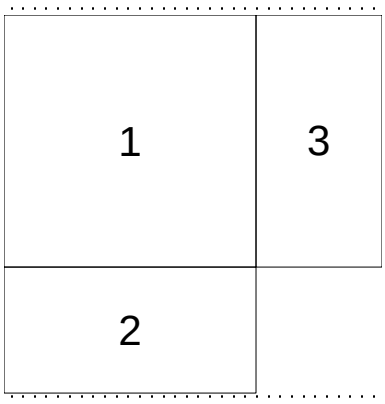
```
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    1    0    3    0
[3,]    0    0    2    0    0
[4,]    0    0    0    0    0
```

5 Feldschmelzungen

Eine mehrfache Verwendung von Zahlen (benachbarter Zellen) führt zur Verschmelzung von Feldern. Dieses ist eine grobe Methode, um Größen von Graphiken zu beeinflussen.

6 ⟨* 1⟩+ ≡

```
d.mat<-matrix(c(1,1,2,1,1,2,3,3,0),3,3)
layout(d.mat)
par(cex=4); layout.show(3)
d.mat
```

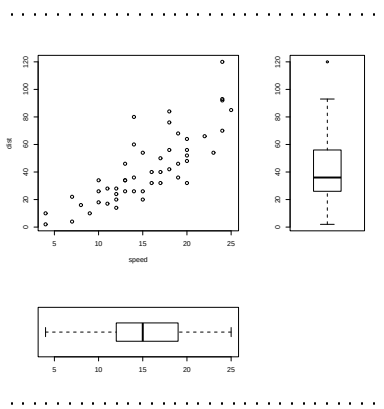



```
     [,1] [,2] [,3]
[1,]    1    1    3
[2,]    1    1    3
[3,]    2    2    0
```

Hierfür lässt sich eine einfache Anwendung finden, in der wir im ersten Bild einen Scatterplot und im zweiten wie dritten jeweils einen Boxplot einzeichnen.

7 ⟨* 1⟩+ ≡

```
d.mat<-matrix(c(1,1,2,1,1,2,3,3,0),3,3)
layout(d.mat)
plot(cars); boxplot(cars[,1],horizontal=TRUE); boxplot(cars[,2])
d.mat
```

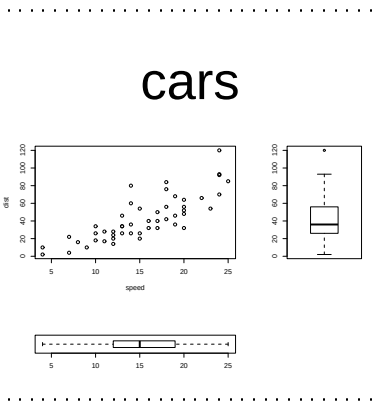


Falls wir oben noch etwas Platz benötigen, können wir diesen beispielsweise mit einer Zeile leerer Felder erzeugen. In diesen Freiraum lässt sich mit der Randbeschriftungsfunktiion `mtext` eine Überschrift einbringen.

8 ⟨* 1⟩+ ≡

```
d.mat<-matrix(c(1,1,2,1,1,2,3,3,0),3,3)
d.mat<-rbind(0,d.mat)
layout(d.mat)
plot(cars); boxplot(cars[,1],horizontal=TRUE); boxplot(cars[,2])
# par(cex=2); layout.show(3)
mtext("cars",3,outer=TRUE,line=-10,cex=5)
```

```
d.mat
```

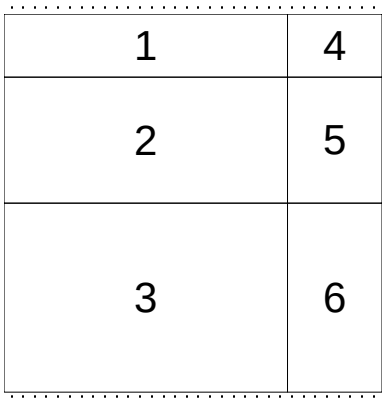


6 Steuerung der Größenverhältnisse

Oft reicht die Vereinigung von Rasterfeldern für die angestrebten Größenverhältnisse nicht aus. Deshalb gibt es die Möglichkeit, diese mit Hilfe der Argumente `widths` und `heights` zu setzen. Betrachten wir hierfür ein Beispiel:

9 ⟨* 1⟩+ ≡

```
d.mat<-matrix(1:6,3,2)
layout(d.mat,heights=c(1,2,3),widths=c(3,1))
par(cex=4); layout.show(6)
d.mat
```



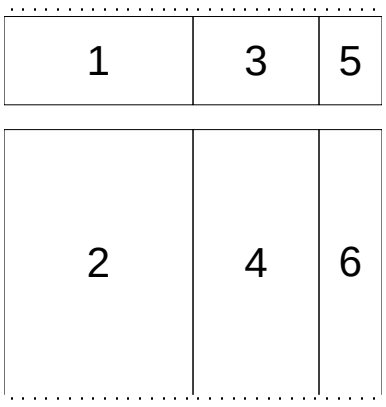

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Wir sehen, dass nun die zweite Zeile doppelt so groß ist wie die erste und die dritte sogar dreimal so groß. Weiter ist die erste Spalte dreimal so breit wie

die zweite. Die Argumente `widths` und `heights` legen also Größenverhältnisse innerhalb der Zeilen bzw. Spalten fest. Folglich ändert sich das Erscheinungsbild nicht, wenn beispielsweise alle Einträge von `heights` verdoppelt werden.

Manchmal möchte man jedoch Höhen und Breiten in absoluten Einheiten festlegen. Für solche Fälle können wir auch cm-Angaben eintragen. Als Beispiel fügen wir nun in ein (2×3)-Schachbrett eine Zwischenzeile der Größe `"1 cm"` ein. Dazu übergeben wir dem `heights`-Argument an passender Stelle den String `"1 cm"`, also inklusive eines Leerzeichens vor `cm`. Andere Einheiten wie `mm` oder `inch` funktionieren übrigens nicht auf diese Weise.

10 ⟨* 1⟩+ ≡
```
d.mat<-rbind( c(1,3,5), 0, c(2,4,6) )
layout(d.mat,heights=c(1,"1 cm",3),widths=c(3,2,1))
par(cex=4); layout.show(6)
d.mat
```




```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    0    0    0
[3,]    2    4    6
```

Als Schreiberleichterung hilft uns die Funktion `lcm`, die Zahlen in *cm* übersetzt:

11 ⟨* 1⟩+ ≡
```
lcm(3.5)
```
... liefert den Output:

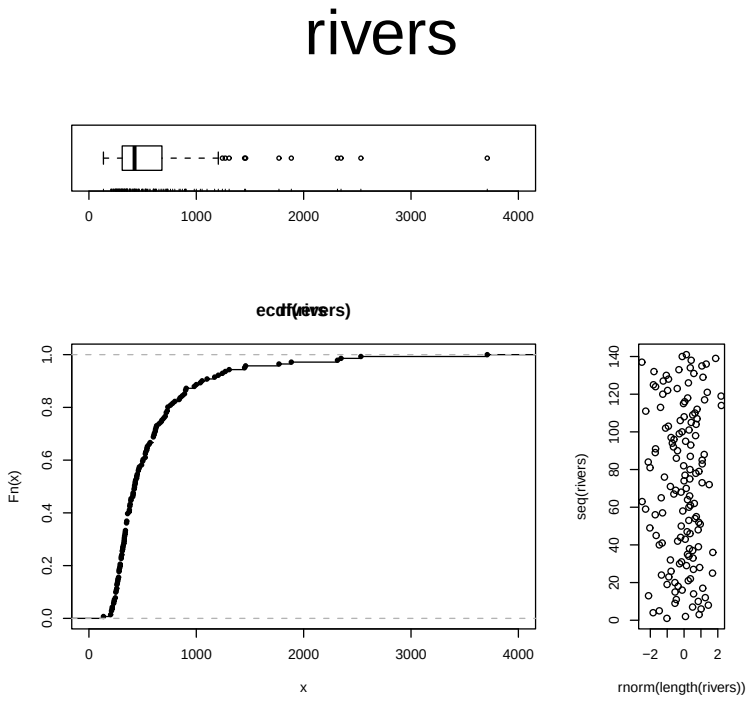
```
[1] "3.5 cm"
```

Wohlgemerkt: die kleine Hilfs-Funktion beginnt mit `l` wie Ludwig. Mit dieser Technik können wir uns beispielsweise oben einen Rand fester Größe erzeugen.

12 ⟨* 1⟩+ ≡
```
d.mat<-rbind(0, c(2,0), c(1,3) )
layout(d.mat,heights=c(lcm(2),1,2),widths=c(3,1))
# par(cex=4); layout.show(3)
xlim<-range(pretty(rivers))
plot(ecdf(rivers),xlim=xlim); title("rivers")
boxplot(rivers,horizontal=TRUE, ylim=xlim); rug(rivers)
plot(rnorm(length(rivers)),seq(rivers))
```

```
mtext("rivers",3,outer=TRUE,line=-6,cex=3)
d.mat
```

```
     [,1] [,2]
[1,]    0    0
[2,]    2    0
[3,]    1    3
```



7 Größenverhältnisse mit Bezugfeld

Zunächst möge sich der Leser überlegen, welche Wirkung eine einzeilige Design-Matrix mit den Werten 1 und 2 sowie den `widths`-Angaben `c(3,1)` haben würde. ... Die Antwort sollte etwa so lauten: Es werden zwei Plots nebeneinander entstehen, wobei der zweite doppelt so breit wie der erste ist. In vertikaler Richtung würde der gesamte Device-Bereich ausgeschöpft. Nun setzen wir das Argument `respect` auf `TRUE` und betrachten das Ergebnis.

13 ⟨ * 1⟩+ ≡

```
d.mat<-rbind( c(1,2) )
layout(d.mat,widths=c(3,1),respect=TRUE)
par(cex=4); layout.show(2)
d.mat
```

```
     [,1] [,2]
[1,]    1    2
```

. . . . . .

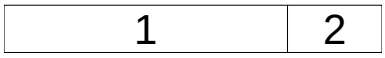
. . . . . .

Was sehen wir? In der Tat ist das erste Feld dreimal so groß wie das zweiterste. Doch wird nun die gesamte Device-Fläche nicht mehr ausgenutzt. Wir können weiter feststellen, dass das zweite Feld ziemlich quadratisch ausgefallen ist. Die Regel lautet, dass die Einträge von `heights` und `widths` in derselben Einheit angegeben sind: Die Verhältnisse von Längen und Breiten beachten oder respektieren sich gegenseitig. Da durch die Nicht-Angabe von `heights` alle Höhenangaben auf 1 gesetzt werden, muss für das erste Feld das Verhältnis aus Höhe und Breite `1/3` und für das zweite `1/1` betragen. Wir machen die Kontrolle, indem wir den `widths`-Vektor verdoppeln und müssten dann die Verhältnisse `1/6` und `1/2` für Höhe/Breite beobachten können.

14 ⟨* 1⟩+ ≡

```
d.mat<-rbind( c(1,2) )
layout(d.mat,widths=c(6,2),respect=TRUE)
par(cex=4); layout.show(2)
d.mat
```

. . . . . .



. . . . . .

Das Ergebnis entspricht unseren Erwartungen.

Betrachten wir ein zweites Beispiel:

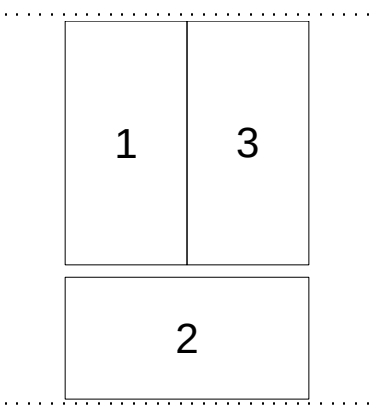
1. Wir wollen drei Plots erstellen, wobei Plot 1 und 3 oben nebeneinander platziert sind und Plot 2 mit etwas Abstand quer darunter liegt. Also wählen wir ein (3×2)-Layout, wobei die mittlere Zeile nur für den Platz sorgen soll, und setzen: `d.mat<-rbind( c(1,3), 0, c(2,2) )`.

2. Die oberen Plots sollen doppelt so hoch sein wie der untere. Deshalb wählen wir als Höhenvektor: `c(2,lcm(0.5),1)`.

3. Letztlich sollen Höhe und Breite mit gleicher Einheit gemessen werden. Folglich übergeben wir: `respect=TRUE`. Hinweis: Die Größe des Zwischenraums in dem abgedruckten Schema beträgt wegen verarbeitungstechnischen Verkleinerungsoperationen weniger als 0.5cm.

15 ⟨* 1⟩+ ≡
```
d.mat<-rbind( c(1,3), 0, c(2,2) )
layout(d.mat,heights=c(2,lcm(0.5),1),respect=TRUE)
par(cex=4); layout.show(3)
d.mat
```

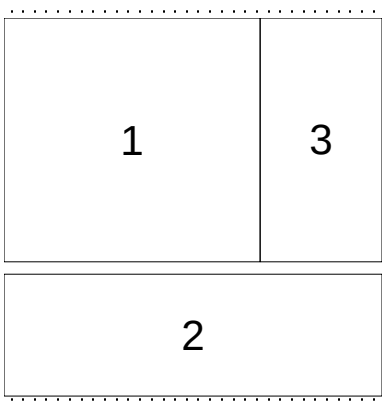
```
     [,1] [,2]
[1,]    1    3
[2,]    0    0
[3,]    2    2
```



Einschränkung von `respect`. Manchmal kommt der Wunsch auf, dass die Gleichsetzung der Einheiten für Höhe und Breite nicht auf allen Feldern gelten soll, sondern beispielsweise nur für das Feld unten links.

4. Auch diesen Wunsch erfüllt die Funktion `layout`. Hierzu müssen wir eine zu der Layout-Matrix gleich große, logische Matrix aus 0-en und 1-en aufbauen. Die 1-Einträge kennzeichnen Felder, in denen Höhe und Breite das Verhältnis aufweisen müssen, wie es durch die `heights`- und `widths`-Zahlenangaben festgelegt ist.

16 ⟨* 1⟩+ ≡
```
d.mat<-rbind( c(1,3), c(0,0), c(2,2) )
layout(d.mat,heights=c(2,lcm(0.5),1),respect=TRUE)
resp.mat<-rbind( c(0,0),0, c(0,1))
layout(d.mat,heights=c(2,lcm(0.5),1),respect=resp.mat)
par(cex=4); layout.show(3)
cat("resp.mat"); resp.mat
```
Es sollen also nur unten rechts die Höhe und die Breite im Verhältnis `1/1` stehen. Deshalb erhalten wir:




```
resp.mat
     [,1] [,2]
[1,]    0    0
[2,]    0    0
[3,]    0    1
```

Falls beiden unteren Felder des gedachten Schachbretts zu beachten sind ...

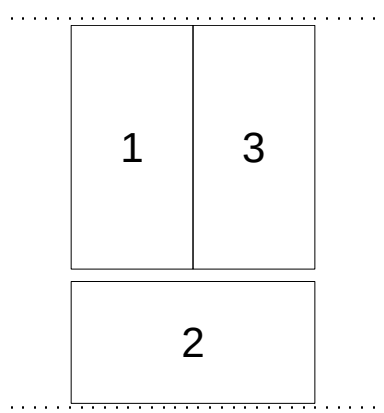
17 ⟨ * 1⟩+ ≡

```
d.mat<-rbind( c(1,3), c(0,0), c(2,2) )
layout(d.mat,heights=c(2,lcm(0.5),1),respect=TRUE)
resp.mat<-rbind( c(0,0),0, c(1,1))
layout(d.mat,heights=c(2,lcm(0.5),1),respect=resp.mat)
par(cex=4); layout.show(3)
cat("resp.mat"); resp.mat
```

... und zwei 1-en in der letzten Zeile stehen, ...

```
resp.mat
     [,1] [,2]
[1,]    0    0
[2,]    0    0
[3,]    1    1
```

... erhalten wir ein schlankeres Gesamtbild.



Dies ist ein Ergebnis, was uns schon bekannt vorkommt.

8 Hilfeseitenbeispiel und Abschlussbemerkung

Der auf Seite 1 abgebildete Plot ist mit Anweisungen der Hilfeseite zu `layout` erstellt. Diese werden nun nachgeliefert und müssten dem Leser schnell einleuchten.

18 ⟨*aus der R-Hilfe* 18⟩ ≡

```
##-- Create a scatterplot with marginal histograms -----
x <- pmin(3, pmax(-3, stats::rnorm(50)))
y <- pmin(3, pmax(-3, stats::rnorm(50)))
xhist <- hist(x, breaks=seq(-3,3,0.5), plot=FALSE)
yhist <- hist(y, breaks=seq(-3,3,0.5), plot=FALSE)
top <- max(c(xhist$counts, yhist$counts))
xrange <- c(-3,3); yrange <- c(-3,3)
nf <- layout(matrix(c(2,0,1,3),2,2,byrow=TRUE), c(3,1), c(1,3), TRUE)
layout.show(nf); par(mar=c(3,3,1,1))
plot(x, y, xlim=xrange, ylim=yrange, xlab="", ylab="")
par(mar=c(0,3,1,1))
barplot(xhist$counts, axes=FALSE, ylim=c(0, top), space=0)
par(mar=c(3,0,1,1))
barplot(yhist$counts, axes=FALSE, xlim=c(0, top), space=0, horiz=TRUE)
```

Abschlussbemerkung Es lohnt sich die Beispiele selbst auszuprobieren. Hierdurch entwickelt man schnell ein Gefühl für verschiedene Layouts und hat dann auch keine Hemmungen mehr, eigene Konstruktionen zu wagen. Bei der Wahl sehr kleiner Felder werden sich bisweilen Fehlermeldungen einstellen. Dieses liegt oft daran, dass nicht genügend viel Platz für das Anbringen von Labels oder Achsenbeschriftungen verbleibt. Dann hilft es nur, die Felder wieder zu vergrößern oder auf entsprechende Randeinträge zu verzichten.