Basic graphics in R

Ulrich Halekoh
Biometry Research Unit
Danish Institute of Agricultural Sciences

January 9, 2007

Contents

1 Introduction

R has two primary methods for producing graphics.

1. The standard method of R is to build a plot sequentially. You use a high-level function as

   ```
   plot
   ```

   to set up the basic plot. You can control many elements in the `plot` function, e.g. the range of the axes, the type of the plotting symbols

or the title.
After creation of the basic plot, you add additional elements, such as `lines` or `text`.

2. The 'trellis'-graphics. These graphics were developed to enhance the clarity of information of statistical plots. These are especially strong in presenting the relationship of two variables conditioning on the levels of a third. The 'trellis' system is available in R in the `lattice` package. Its main function is `xyplot`. The graphics are not generated sequentially but by one call to the function.

Using the 'trellis'-method is somewhat more complex - but often more elegant - than the basic system. Especially the fine tuning of plots requires a good knowledge of RẆe will focus on the basic system but sometimes hint to solutions in 'trellis'.

2 The example data

For the illustration we use the data `Puromycin` which is a R data set. The data give the reaction velocity versus substrate concentration in an enzymatic reaction involving untreated cells or cells treated with Puromycin.

Loading and summarizing the data:

```
data(Puromycin)
summary(Puromycin)
```

We generate to subset data sets, `PuroA` contains only the data with `state=='treated'` and `PuroB` with `state=='untreated'`:

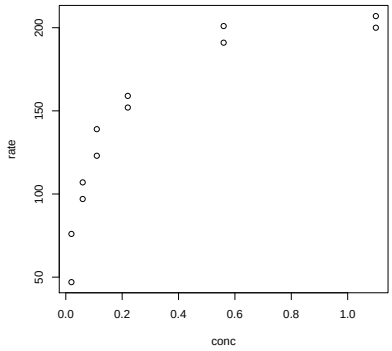
```
PuroA <- subset(Puromycin, state == "treated")
PuroB <- subset(Puromycin, state == "untreated")
```

3 Single plot

3.1 Simple scatterplot

A simple scatterplot of the continous variable `rate` against the concentration `conc` is obtained by

```
plot(rate ~ conc, data = PuroA)
```



We used the formula based method to plot the data. The instruction 'plot rate against conc' is given by the 'formula' `rate~conc`.

Another form is

```
with(PuroA, plot(conc, rate))
```

In this version the x-axis variable `conc` is given first. Here the `plot` function does not accept a `data` argument, and one has to use the `with` function to tell the function where to find the data for plotting.

3.2 Labeling axes and setting their range

Various aspects of the plot, like labels of the axis, their size and the range of the axis can be changed by extra arguments. These can be found by looking at

```
help(plot.default)
```

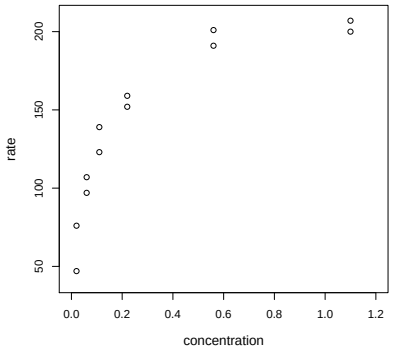
or more generally at

```
help(par)
```

R uses the names of the variables for the axes labels and computes a range for the axes. One can fine-tune these elements by

- axes labels: `xlim`, `ylim`
- size of labels: `cex.lab`
- axes range: `xlim`, `ylim`

```
plot(rate ~ conc, data = PuroA, xlim = c(0, 1.2), ylim = c(40,
    210), xlab = "concentration", ylab = "rate", cex.lab = 1.2)
```



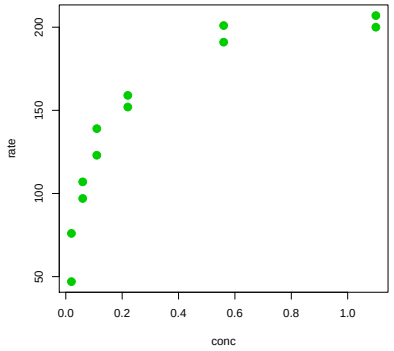
3.3 Choosing symbols, colors and sizes for the points

We select now

- symbol for the points: `pch=16`,

- a color: `col=3` or `col='green'`
- a size: `cex=1.1`, which is a factor multiplying the basic size.

```
plot(rate ~ conc, data = PuroA, pch = 16, col = 3, cex = 1.5)
```



A simple overview of the available integer colors and the plotting symbols is given with function of the `Hmisc` package:

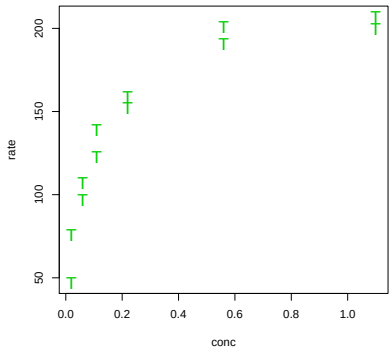
```
library(Hmisc)
```

```
Hmisc library by Frank E Harrell Jr

Type library(help='Hmisc'), ?Overview, or ?Hmisc.Overview')
to see overall documentation.

NOTE:Hmisc no longer redefines [.factor to drop unused levels when
subsetting.  To get the old behavior of Hmisc type dropUnusedLevels().
```

```
show.col()
show.pch()
```

Instead of using a symbol for the points it is sometimes more informative to use a single character as `pch="T"` or `pch="1"`.

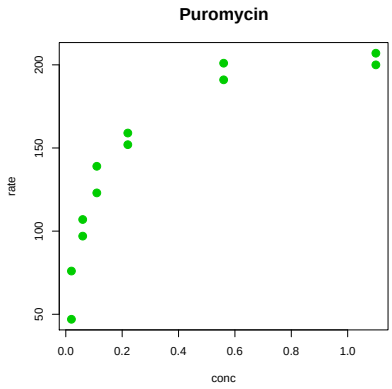
```
plot(rate ~ conc, data = PuroA, pch = "T", col = 3,
    cex = 1.5)
```



3.4 Title

A title for the plot is added using the `title`-function.

```
plot(rate ~ conc, data = PuroA, pch = 16, col = 3, cex = 1.5)
title(main = "Puromycin", cex.main = 1.5)
```



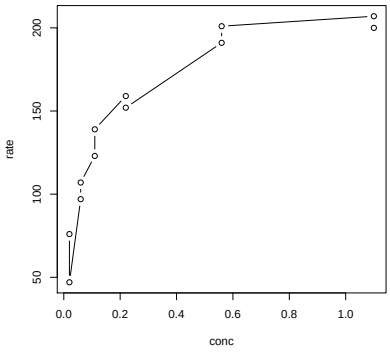
3.5 Adding lines to a plot

Adding different types of lines can be useful when visualizing trends in a scatterplot.

3.5.1 Connecting points

Points are connected using the `type`-argument in the `plot`-function. (`type='p'` or `'l','b'` yields only points, only lines or both.)

```
plot(rate ~ conc, data = PuroA, type = "b")
```



The `conc` values are already ordered in ascending order. If this were not the case, the above plot would look like spaghetti. To order the rows of a dataframe according to one or several variables use `orderBy` of the package `doBy`

```
library(doBy)
PuroA <- orderBy(~conc, PuroA)
```

Other plotting types are available. See the help page for `plot` by typing `?plot` for details.

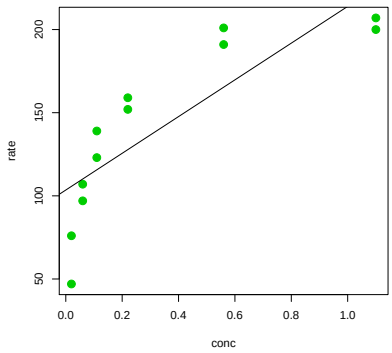
3.5.2 Add a straight line

A straight regression line can be added to a plot using the `abline`-function. First one has to compute the regression line by the `lm` function. Then one adds the line using `abline`. This is an instance of a low-level plotting function, which adds elements to the plot, already generated by `plot`. Generating the plot

```
plot(rate ~ conc, data = PuroA, pch = 16, col = 3, cex = 1.5)
```

Fitting the straight line by `lm` and adding it to the plot:

```
plot(rate ~ conc, data = PuroA, pch = 16, col = 3, cex = 1.5)
m1 <- lm(rate ~ conc, data = PuroA)
abline(m1)
```

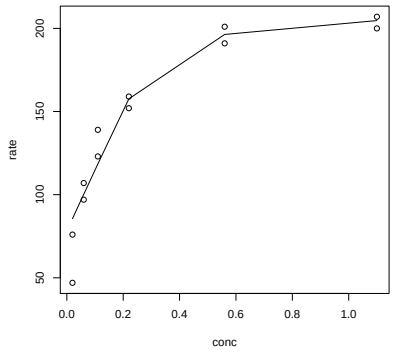


3.5.3 Add a smooth line

R provides different method functions to add a smooth line to a plot. Here it is illustrated how some of these can be used.

- **`lowess` function**
  First the `lowess` function is introduced. It takes a smoothing parameter `f` between 0 and 1 which determines how smooth the fit will be. The `with` tells the `lowess` function where to find the data.

  ```
  plot(rate ~ conc, data = PuroA)
  smooth <- with(PuroA, lowess(rate ~ conc, f = 0.9))
  lines(smooth)
  ```



The addition of the line could have been written more compact as

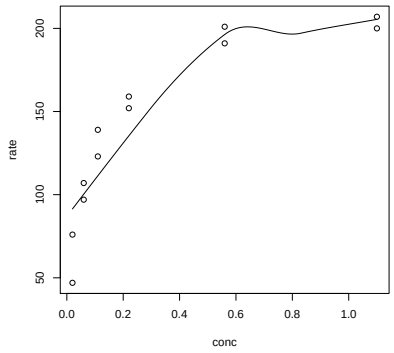
```
lines(with(PuroA, lowess(rate ~ conc, f = 0.9)))
```

- `locfit` **function**
  Another function that produces approximating smooth lines is `locfit`. This function is provided with the CRAN package `locfit`.

```
library(locfit)
```

```
locfit 1.5-3          2006-04-06
```

```
plot(rate ~ conc, data = PuroA)
smooth <- locfit(rate ~ lp(conc, nn = 0.99, deg = 1),
    data = PuroA)
lines(smooth)
```

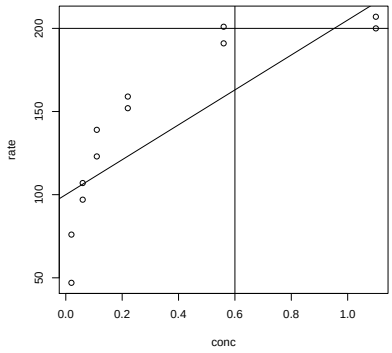


Here the `nn` is a smoothing parameter used to specify the 'smoothness' of the curve, and `deg` specifies the degree 1 or 2 of the polynomial that in a small region approximates the points. Choosing a low degree will produce a more rigid line fit.

3.5.4 `abline` Add a reference line

We have seen how the function `abline` allows to add a regression line. The function can also be used to produce a line $y = a+b\cdot x$ by typing `abline(a,b)`. Vertical lines at $x = a$ can be added with `abline(v=a)` and horizontal lines at $y = a$ are added by `abline(h=a)`.

```
plot(rate ~ conc, data = PuroA)
abline(100, 105)
abline(h = 200)
abline(v = 0.6)
```



3.6 Add text to a plot

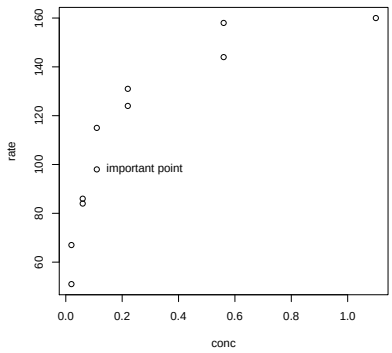
The `text` function allows to place text in the plotting region.

```
plot(rate ~ conc, data = PuroB)
```

Assume we want to annotate the observation with the rate=100 value. We can place a text in the vicinity to the point by

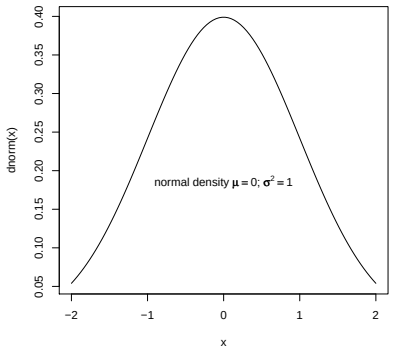
```
text(0.12, 98, label = "important point", pos = 4)
```

The `pos=4` argument asks to place the text right from the coordinates.



An important feature allows to use mathematical notation and to embed values calculated in R into the text. For example a plot of the normal density could be accompanied by the values for the mean μ and variance σ^2 used:

```
x <- seq(-2, 2, l = 100)
plot(x, dnorm(x), type = "l")
mu <- 0
variance <- 1
text(0, 0.2, pos = 1, label = bquote(paste("normal density ",
    mu == .(mu), "; ", sigma^2 == .(variance))))
```



The `bquote` function allows to combine mathematical notation and normal text. Notice that the construction `.(mu)` prints the value of `mu`. For further information on how to produce mathematical notation see `plotmath`. You can use this facility also in the title and the axis-labels.

4 Overlaying several scatterplots

We discuss now how to display two scatterplots on the same plot. In this case it is necessary to be able to distinguish between the different plots by using different symbols or colors.

4.1 Several scatterplots in one plot

First, we want to plot the observations of both levels of `state` in one plot. We will use the symbol `16` for the treated and `2` for the untreated. We generate a vector that contains for each observation the corresponding `pch` value. For the colors we chose 1 and 2:

```
pch.vec <- c(16, 2)[Puromycin$state]
col.vec <- c(1, 2)[Puromycin$state]
```

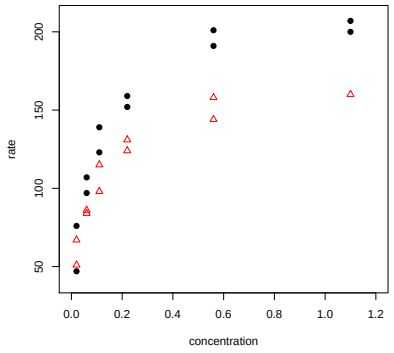
The entries in `c(16,2)` must be in the corresponding order of the levels of the factor `state`

```
levels(Puromycin$state)
```

```
[1] "treated"   "untreated"
```

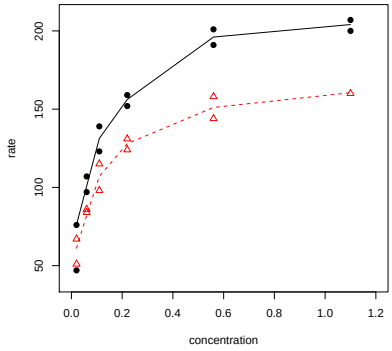
All the points in the dataset `Puromycin` can be plotted with the command

```
plot(rate ~ conc, data = Puromycin, col = c(1, 2)[state],
    pch = c(16, 2)[state], xlim = c(0, 1.2), ylim = c(40,
        210), xlab = "concentration")
```



Then smoothing lines for each method can now be added to the plot. Notice, that we chose different line types with the `lty` argument and different colors.

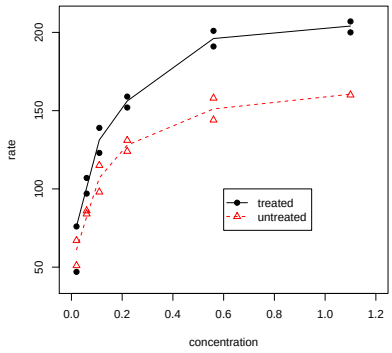
```
smooth <- with(PuroA, lowess(rate ~ conc))
lines(smooth, lty = 1, col = 1)
smooth <- with(PuroB, lowess(rate ~ conc))
lines(smooth, lty = 2, col = 2)
```



4.2 Legend

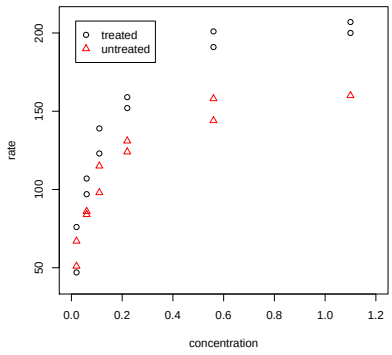
A legend can be added via the `legend` function. You have to provide the upper left coordinates of the box of the legend. In the `legend` arguments you provide explanatory test, other parameters are those characterizing the points and lines for the different subplots

```
legend(x = 0.6, y = 100, legend = c("treated", "untreated"),
    col = 1:2, pch = c(16, 2), lty = c(1, 2))
```



The function `smartlegend` of the package `gplots` allows a simpler placement of the legend. To place the legend in the left top corner use

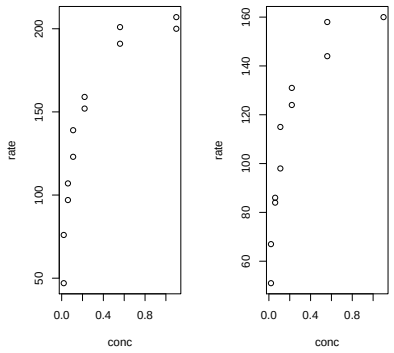
```
library(gplots)
plot(rate ~ conc, data = Puromycin, col = c(1, 2)[state],
    pch = c(1, 2)[state], xlim = c(0, 1.2), ylim = c(40,
        210), xlab = "concentration")
smartlegend("left", "top", legend = c("treated", "untreated"),
    col = 1:2, pch = 1:2)
```



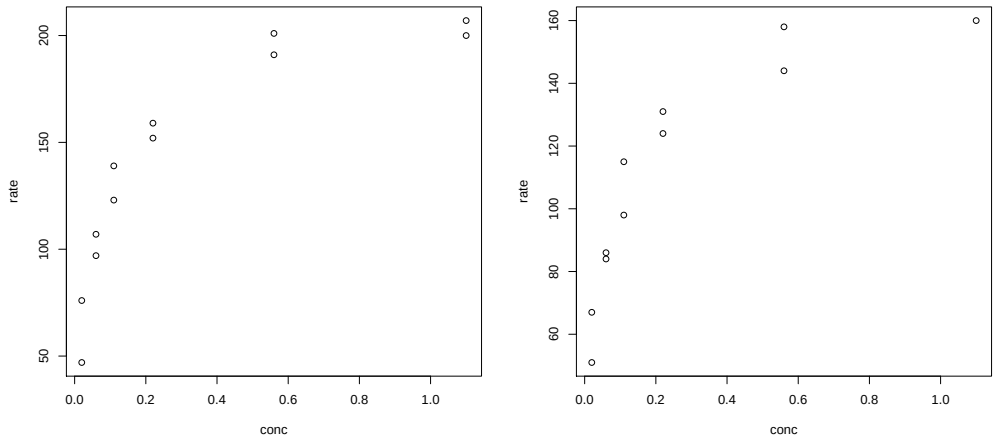
5 Plots side by side

Using the `par`-function it is possible to place several plots on one page. To place two plots side by side (1 row and 2 columns) use

```
par(mfrow = c(1, 2))
plot(rate ~ conc, data = PuroA)
plot(rate ~ conc, data = PuroB)
```



The plots look a bit squeezed in the x-direction. Change the size of the plotting window with the mouse until you get a more acceptable ratio of the y- to the x-axis.

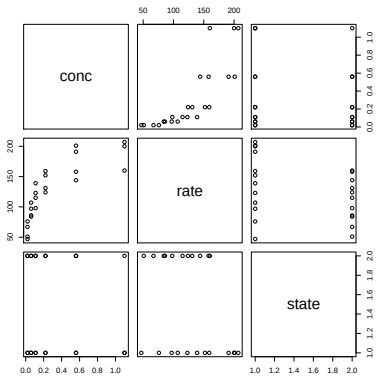


6 Some special plots

6.1 Scatterplotmatrix

A dataset contains many variables and one wants to get a quick overview over the pairwise relations between the variables. This may be achieved by generating all pairwise scatterplot of the variables.

```
plot(Puromycin)
```

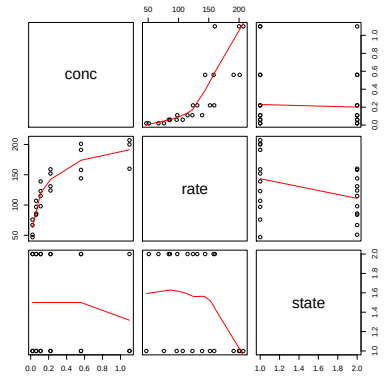


or alternatively

```
pairs(Puromycin)
```

These plots arranged in a matrix form a scatterplotmatrix. The `panel=panel.smooth` argument asks to add a smooth line to each plot.

```
pairs(Puromycin, panel = panel.smooth)
```

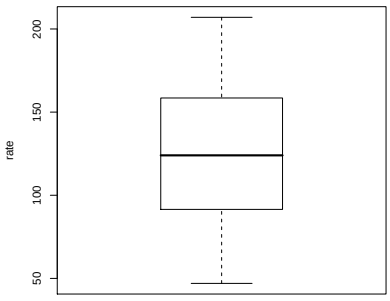


6.2 Boxplot

6.2.1 A single boxplot

Box and whisker plots showing certain quantiles of a variable can be created using the `boxplot`-function.

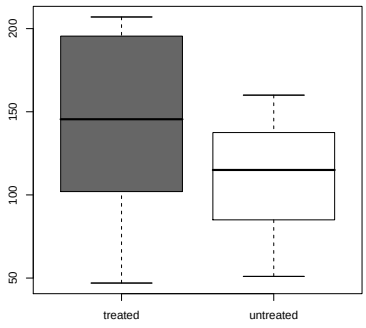
```
with(Puromycin, boxplot(rate, data = PuroB, ylab = "rate"))
```



6.2.2 Several boxplots in one plot

It is possible to get more than one boxplot in one figure. To get a boxplot for each level of `state` one uses the formula-based version of `boxplot`

```
boxplot(rate ~ state, data = Puromycin, col = grey(c(0.4,
    1)))
```



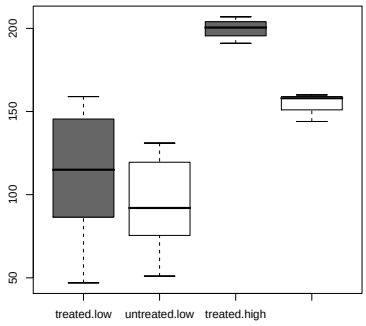
We used here grey-levels to distinguish the two levels of `state`.

The formula approach allows also to use several classifying or grouping variables. Assume that one has categorized the variable `conc` in two levels: 'low' of $conc < 0.3$ and 'high' otherwise. This task can be performed with the `cut` function:

```
Puromycin$conc.cat <- with(Puromycin, cut(conc, c(0,
    0.3, 4), label = c("low", "high")))
```

Then four boxplot, where the levels of `state` run fastest is obtained by

```
boxplot(rate ~ state + conc.cat, data = Puromycin, col = grey(c(0.4,
    1)))
```



Observe, how the vector of colors are recycled to distinguish between the levels of treated and untreated.

6.3 Dotplot

If there are only few observations in each group, the boxplot is not reasonable. One should plot each point.

```
with(Puromycin, stripchart(rate ~ state, vertical = T,
    pch = 16))
```

To avoid overplotting of points use the `jitter`-method.

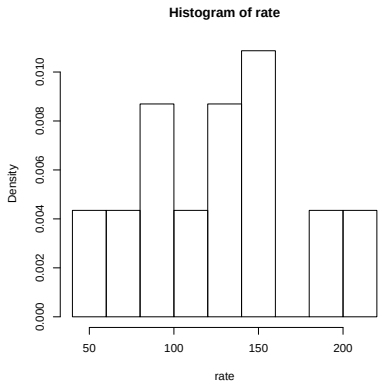
```
with(Puromycin, stripchart(rate ~ state, method = "jitter",
    jitter = 0.1, ylim = c(-1, 3), vertical = T))
```

The value in the argument `jitter` determines how far the points are spread apart.

6.4 Histogram and density

Graphical representation of the frequency distribution of data.

```
with(Puromycin, hist(rate, probability = TRUE))
```



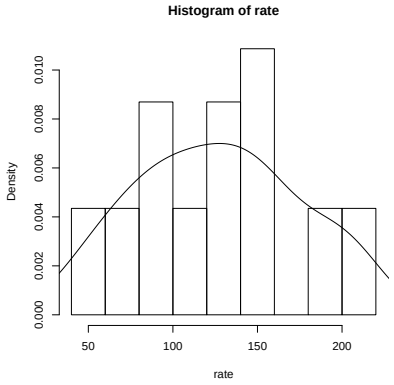
Using `probability=TRUE` the y-axis will denote density instead of number of counts. Using the `breaks`-argument it is possible to control the intervals used for counting observations.

```
with(Puromycin, hist(rate, breaks = c(0, 50, 150, 200,
    250), probability = TRUE))
```

Here the histogram is based on the counts in the intervals $(0, 50), (50, 150)$, $(150, 200), (200, 250)$.

You can add to the histogram a smooth density estimate with the `density` function. The density-based estimate is not dependent on the choice of the breaks as the histogram

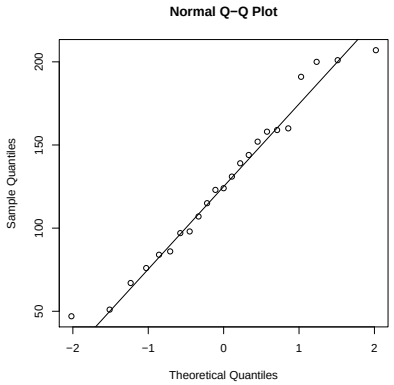
```
with(Puromycin, hist(rate, probability = TRUE))
with(Puromycin, lines(density(rate)))
```



6.5 Q-Q plot

A Q-Q-plot compares the quantiles of the data to those of a standard normal distribution. The Q-Q-plot and a reference line are produced by `qqnorm`-function and `qqline` function.

```
with(Puromycin, qqnorm(rate))
with(Puromycin, qqline(rate))
```



7 Trellis graphics

We will present two plots of the Puromycin data with trellis-graphics of the R package `lattice`.

These plots are especially effective in revealing the relation between two variables on different values of a third. The plots we present can be used to get a quick idea about such relations. The fine tuning of the plots requires a level of complication we do not want to follow here.

7.1 Scatterplots side by side

Separate scatterplots for each level of a factor are produced by

```
library(lattice)
trellis.device(theme = "col.whitebg")
xyplot(rate ~ conc, groups = state, data = Puromycin)
```

The `trellis.device('windows',theme='col.whitebg')` ensures a white plotting background. The `groups` argument is the conditioning factor. For each level of this factor a new plot between `rate` and `conc` is drawn.

Adding smooth lines is achieved by

```
library(lattice)
xyplot(rate ~ conc | state, type = c("p", "smooth"),
    span = 2, data = Puromycin, pch = 17, col = 2, main = "Puromycin")
```

7.2 Overlaid scatterplots

The most basic way to overlay different scatterplots in one figure is

```
xyplot(rate ~ conc, groups = state, data = Puromycin)
```

Adding smooth lines and a bit of customizing the plot is obtained by

```
xyplot(rate ~ conc, groups = state, data = Puromycin,
    type = c("p", "smooth"), span = 2, pch = c(16, 17),
    col = c(1, 3), lty = c(1, 2), cex = 1.3, main = "Title: Puromycin")
```

The addition of an informative legend (here called `key`) is a bit more complicated. Note that in the `key` one repeats the corresponding parameter-setting for the points and the lines.

```
my.key <- list(space = "top", border = T, title = "state",
    cex.title = 0.5, points = list(pch = c(16, 17)),
    lines = list(lty = c(1, 2)), text = list(c("treated",
        "untreated")), col = c(2, 3))
xyplot(rate ~ conc, groups = state, data = Puromycin,
    type = c("p", "smooth"), span = 2, pch = c(16, 17),
    col = c(2, 3), lty = c(1, 2), cex = 1.3, main = "Title: Puromycin",
    key = my.key)
```

8 How to place my plots in a document?

We present two ways to copy graphic output from R into a Word document:

1. The easiest way to get a graph into Word is as follows: When the graph window in R is active, click File -> Copy to clipboard and then choose one of the possible formats. Generally the metafile format (`wmf` or `emf`) gives the best results. Then go to your Word document and paste the graph into the document using Ctrl-V.

2. Alternatively you can go to File -> Save as. You must then choose a format for the file to be saved in. Word can read files of the type Metafile, Jpeg, Png and BMP. It is not so important which format you choose, except that we suggest not to use BMP format because these files can be quite large. After saving the file, you can go to Word and include the file as a picture.

9 More to learn about graphics

In R -News Marc Schwartz has an overview article[1] about basic R graphics. Have also a look at the Wiki[2] and the R Graph Gallery[3].

[1] http://cran.r-project.org/doc/Rnews/Rnews_2003-2.pdf
[2] http://wiki.r-project.org/rwiki/doku.php
[3] http://addictedtor.free.fr/graphiques/