Graphics with R

R Development Core Group

R-core@R-project.org

Graphical capabilities

One of the strengths of the **S** language is graphics.

- Simple, exploratory graphics are easy to produce.
- Publication quality graphics can be created.
- Several device drivers are available including:
  - On-screen graphics - either Windows, or X11, or Macintosh
  - postscript - PostScript graphics commands
  - pdf - Adobe Portable Document Format
  - png - PNG bitmap device (like .gif but free of software patents)
  - jpeg - JPEG bitmap
  - WMF - Windows meta-file (Windows only)

Declaring graphics devices

The on-screen devices are the most commonly used. For publication-quality graphics the postscript, pdf, or WMF devices are preferred because they produce scalable images. Use bitmap devices only when there is no alternative.

The preferred sequence is to specify a graphics device then call graphics functions. If you do not specify a device first, the on-screen device is started.

A contributed **R** package called **`lattice`** provides Trellis graphics functions. When using lattice it is important to declare the device using **`trellis.device`** before issuing graphics commands.

Types of graphics functions

High-level - functions such as **`plot`**, **`hist`**, **`boxplot`**, or **`pairs`** that produce an entire plot or initialize a plot.

low-level - functions that add to an existing plot created with a high-level plotting function. Examples are **`points`**, **`lines`**, **`text`**, **`axis`**, . . .

Trellis functions - functions such as **`xyplot`**, **`bwplot`**, or **`histogram`** that can produce an entire multipanel display in a single call.

After creating a new plot with a high-level plotting function, you can add to the plot by making calls to low-level plotting functions. You cannot, however, do this after a trellis function call.

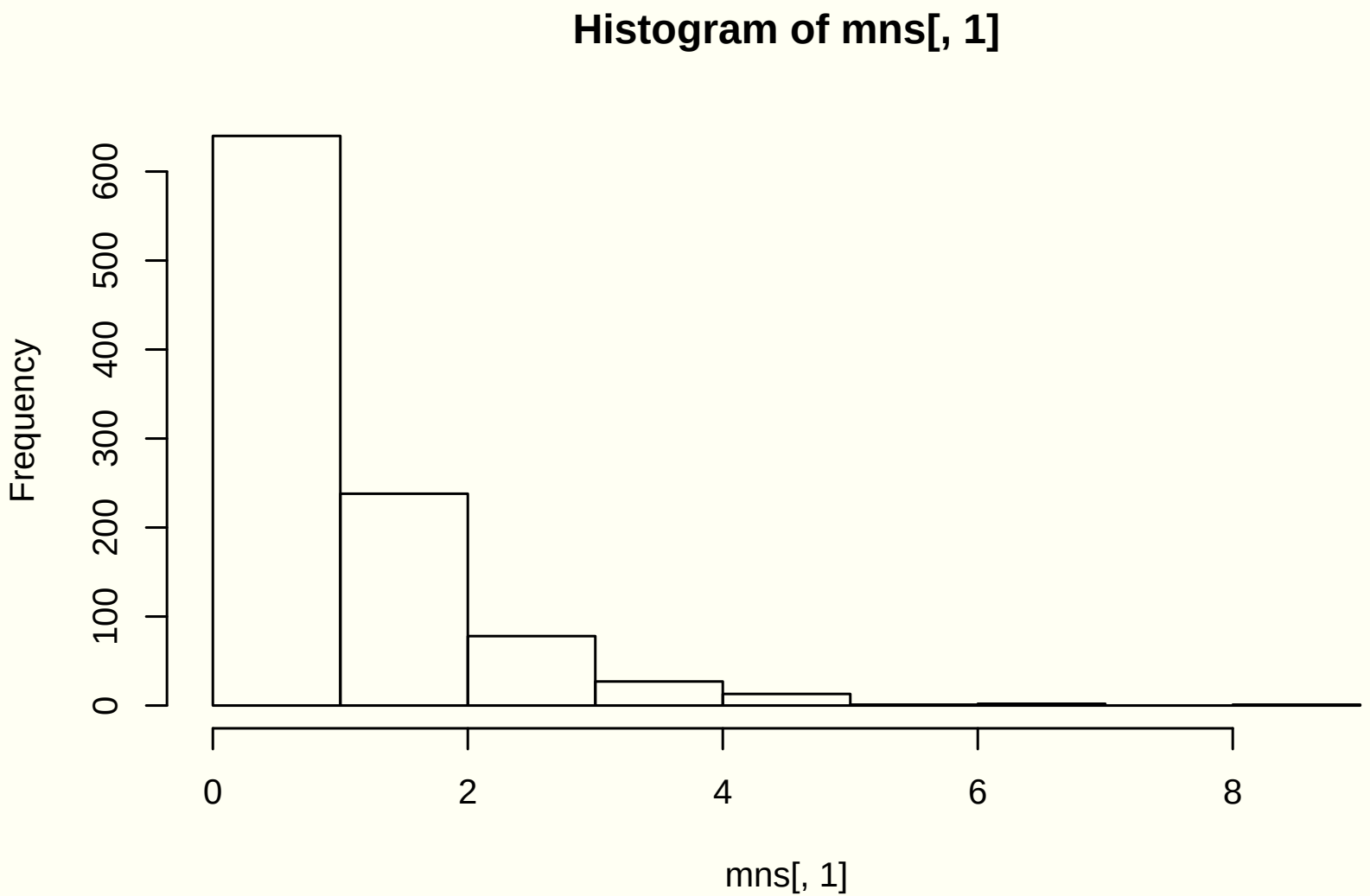
An example

It is common to illustrate the central-limit effect by computing means of samples from a nonsymmetric distribution and showing that the distribution of the mean tends to a normal distribution as the sample size increases. This is best illustrated with graphical displays.

```
> # generate the samples as a matrix
> rmt <- matrix(rexp(1000 * 16), nrow = 16)
> mns <-              # Apply the mean function to columns
+    cbind(rmt[ 1, ],                      # means of samples of 1
+          apply(rmt[ 1:4, ], 2, mean), # means of samples of 4
+          apply(rmt[ 1:16, ], 2, mean) # means of samples of 16
+          )
> meds <-             # Apply the median function to columns
+    cbind(rmt[ 1, ],                        # medians of samples of 1
+          apply(rmt[ 1:4, ], 2, median), # medians of samples of 4
+          apply(rmt[ 1:16, ], 2, median) # medians of samples of 16
+          )
```

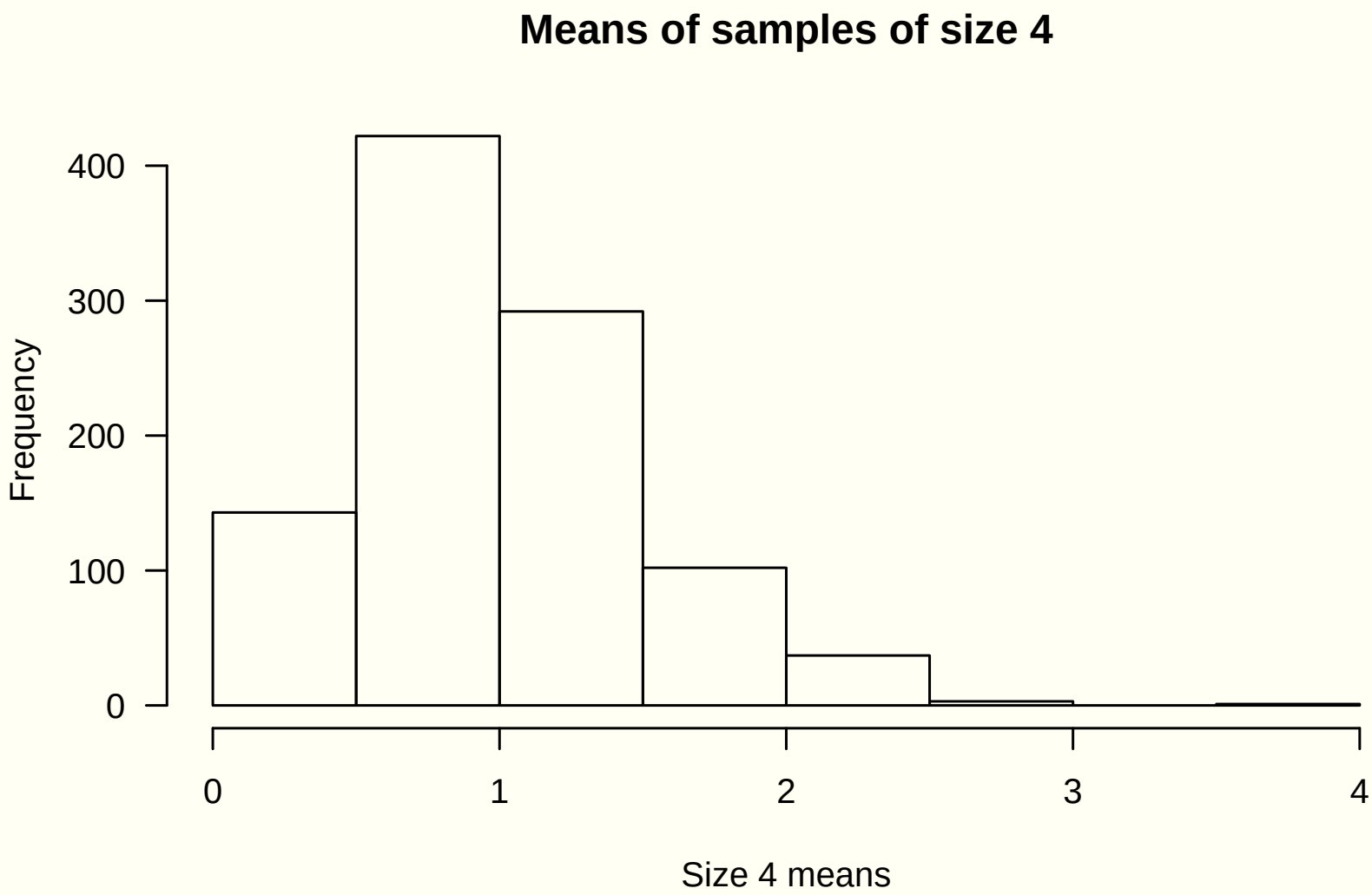
Using high-level plotting functions

```
> hist(mns[, 1])          # a histogram of the means of samples of 1
```



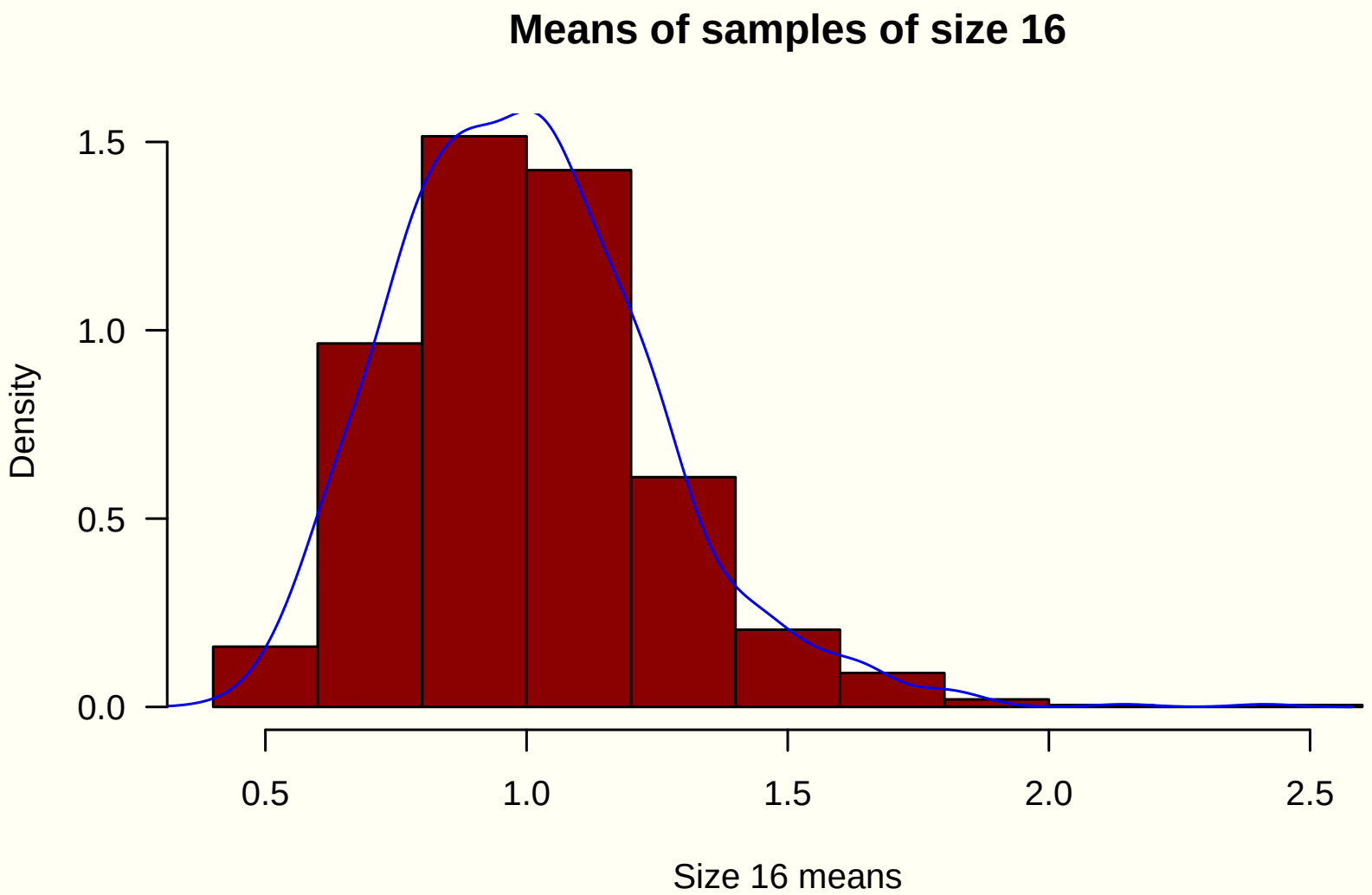
Enhancing high-level plots

```
> hist(mns[,2], main = "Means of samples of size 4",
+      xlab = "Size 4 means", las = 1)  # sets the axis label style
```



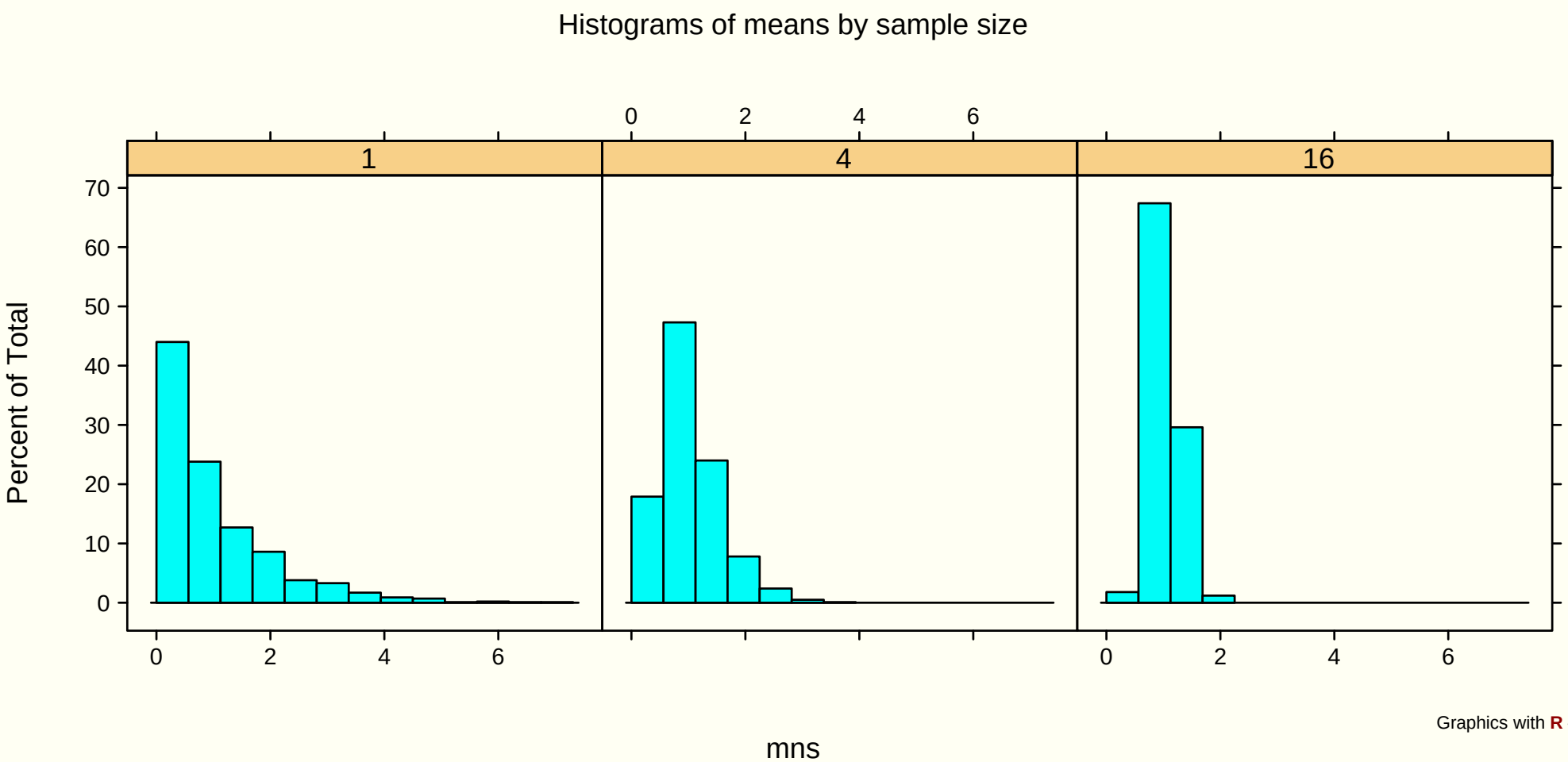
Using low-level graphics functions

```
> hist(mns[,3], main = "Means of samples of size 16",
+      xlab = "Size 16 means", las = 1, col = "darkred", prob = TRUE)
> lines(density(mns[,3]), col = "blue")
```



Using lattice graphics

```
> library(lattice)
> histogram(~ mns | ssz, data = data.frame(mns = c(mns),
+                           ssz = gl(3, 1000, labels = c("1", "4", "16"))),
+           layout = c(3, 1), main = "Histograms of means by sample si
```



Using the formula-data specification

Most of the high-level **R** graphics functions allow a formula-data specification for the plot. In trellis-style graphics from the lattice package the formula-data specification is the only way to specify a plot.

A *formula* in **S** is indicated by the $\sim$ character. Because formulas are used to specify statistical models, this character is often read as “is modelled as”. The second argument in a formula-data specification is usually a data frame with variables corresponding to the names in the formula.

To use the formula-data specification, first construct a data frame with the data to be plotted. The preferred form is to “stack” all the data into a single column with accompanying columns that indicate the groups of observations.

Stacking the simulation data

Recall the simulated data of means and medians of samples of size 1, 4, and 16 from an exponential distribution. We arrange this into a data frame with 6000 rows and three columns - the simulated data, the sample size being simulated, and an indicator of mean or median.

The *gl* function can be used to generate patterned data like the sample size and the type of simulation.

```
> alldat <- data.frame(sim = c(mns, meds),
            ssz = gl(3, 1000, len = 6000, labels = c("1", "4", "16")),
            type = gl(2, 3000, labels = c("Mean", "Median")))
> str(alldat)
'data.frame':   6000 obs. of  3 variables:
 $ sim : num  0.934 0.200 1.866 1.074 1.283 ...
 $ ssz : Factor w/ 3 levels "1","4","16": 1 1 1 1 1 1 1 1 1 1 ...
 $ type: Factor w/ 2 levels "Mean","Median": 1 1 1 1 1 1 1 1 1 1 ...
```

Formulas specifying plots

The general form of a formula specifying a plot is

```
y ~ x | g
```

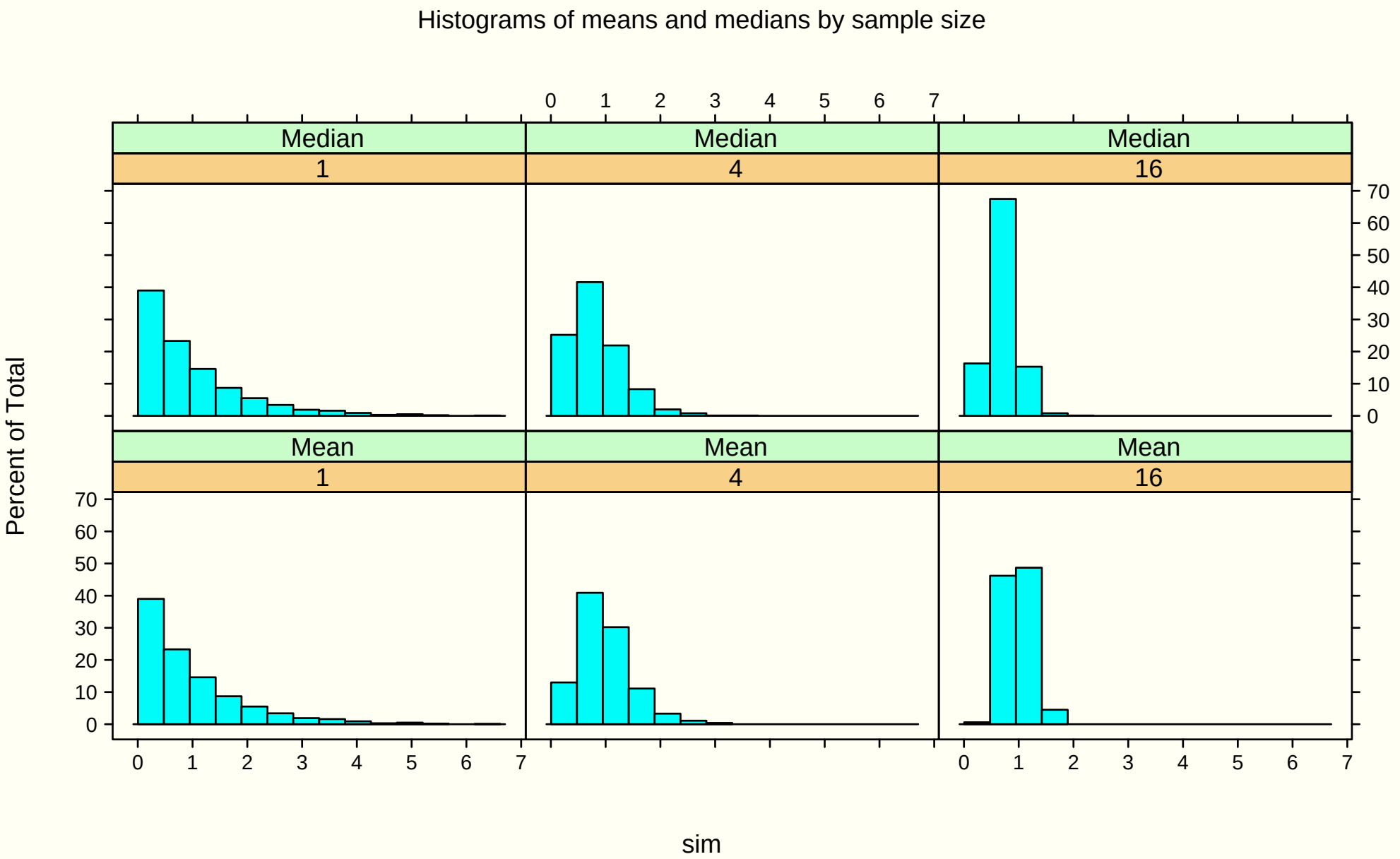
where **y** is assigned to the vertical axis, **x** is assigned to the horizontal axis, and **g** is a grouping factor or expression.

For special cases like the histogram, the vertical axis is pre-specified. In these cases we use a one-sided formula where **y** is omitted.

In trellis graphics functions the grouping expression can include multiple factors separated by an arithmetic operator - often the ***** because this indicates “crossing” the factors. For example,

```
histogram(~ sim | ssz * type, data = alldat,
    layout = c(3, 2),
    main = "Histograms of means and medians by sample size")
```

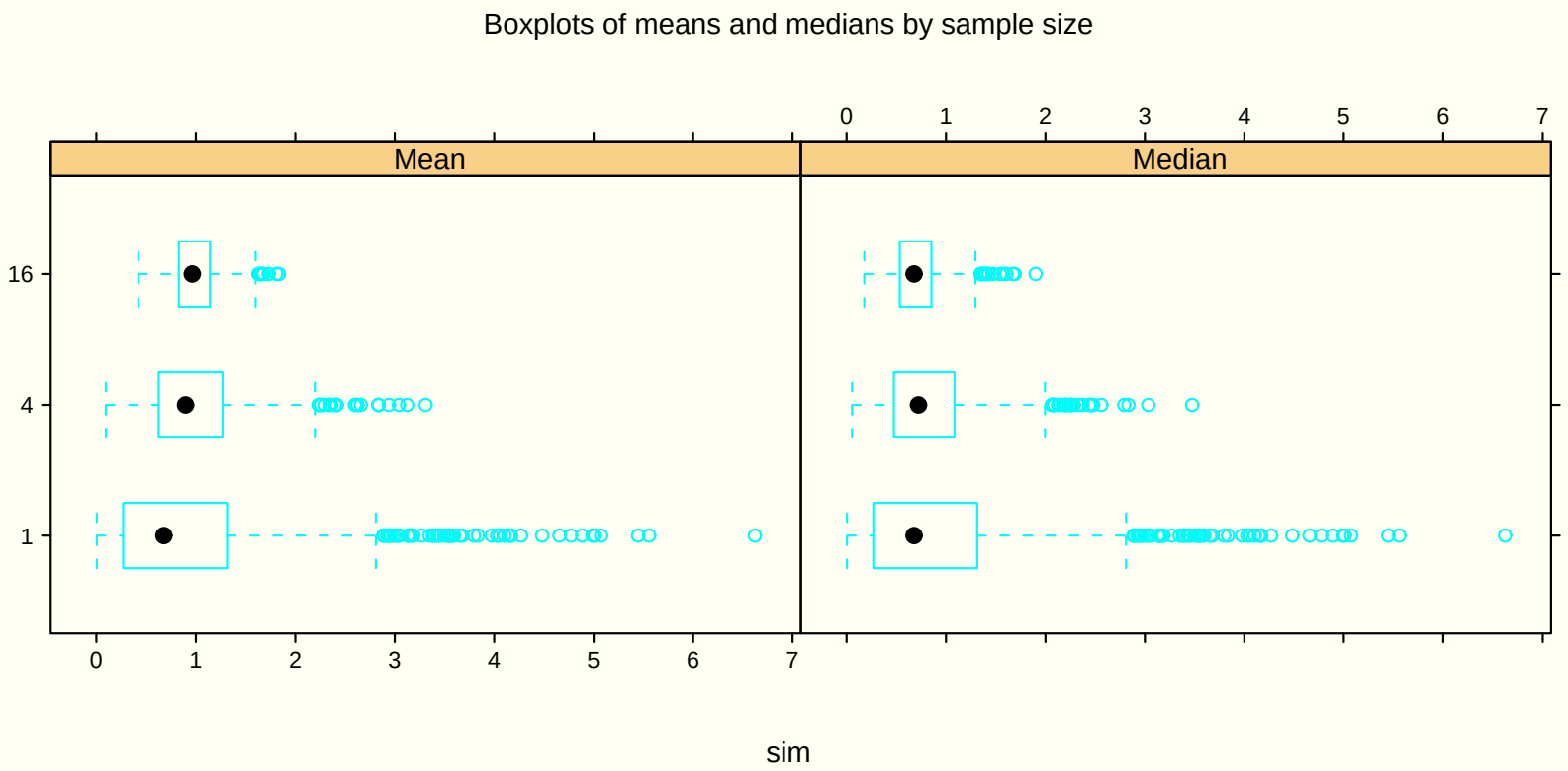
Example of multiple grouping factors



Boxplots

Another way of comparing distributions of sample data is with a **boxplot** (high-level graphics) or **bwplot** (lattice).

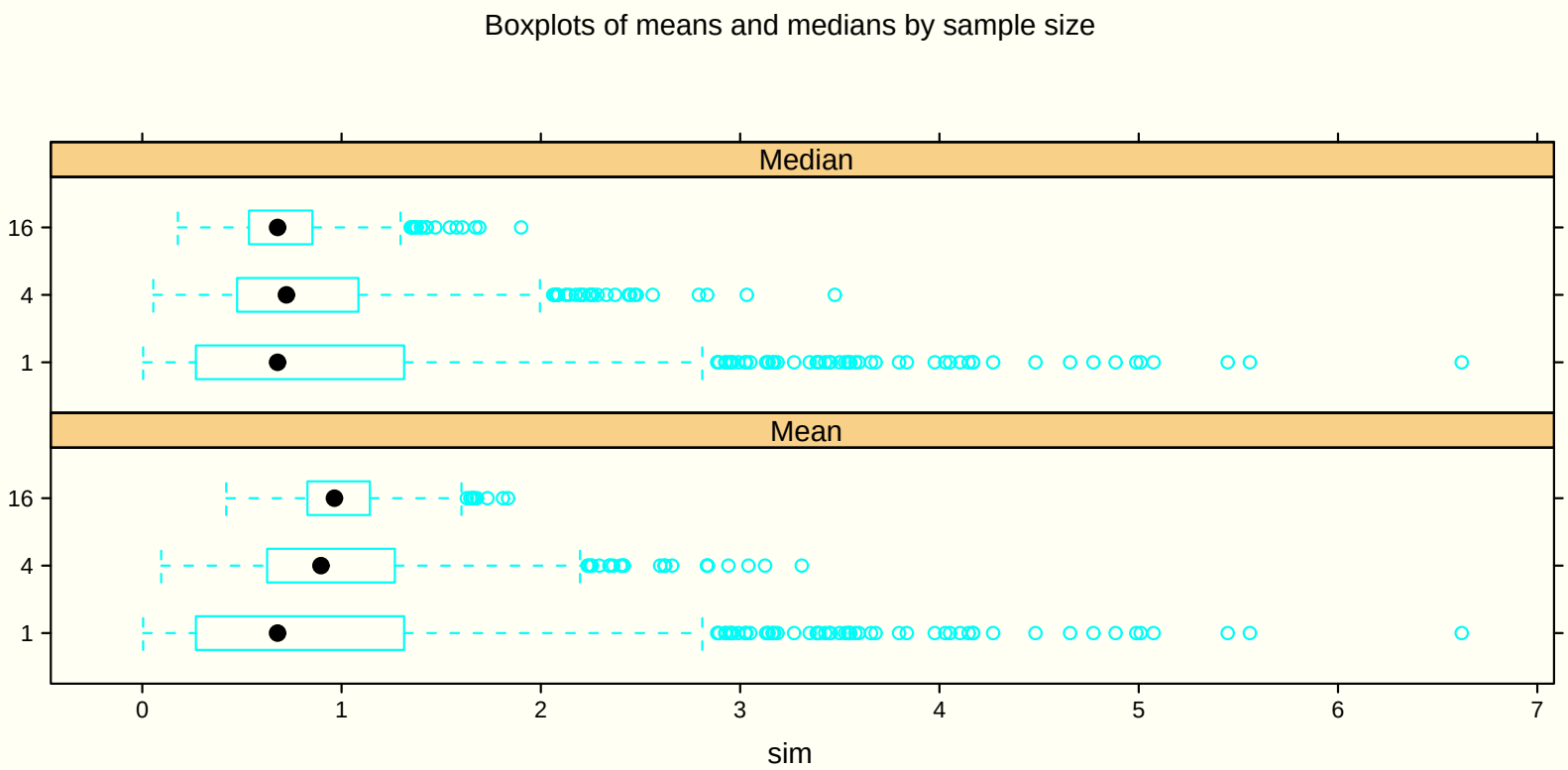
```
bwplot(ssz ~ sim | type, data = alldat,
       main = "Boxplots of means and medians by sample size")
```



Changing the lattice layout

The **`layout`** argument in lattice is used to rearrange the panels. It is a two-component or three-component integer vector in the order **`(columns, rows, pages)`** (not (rows, columns, pages)).

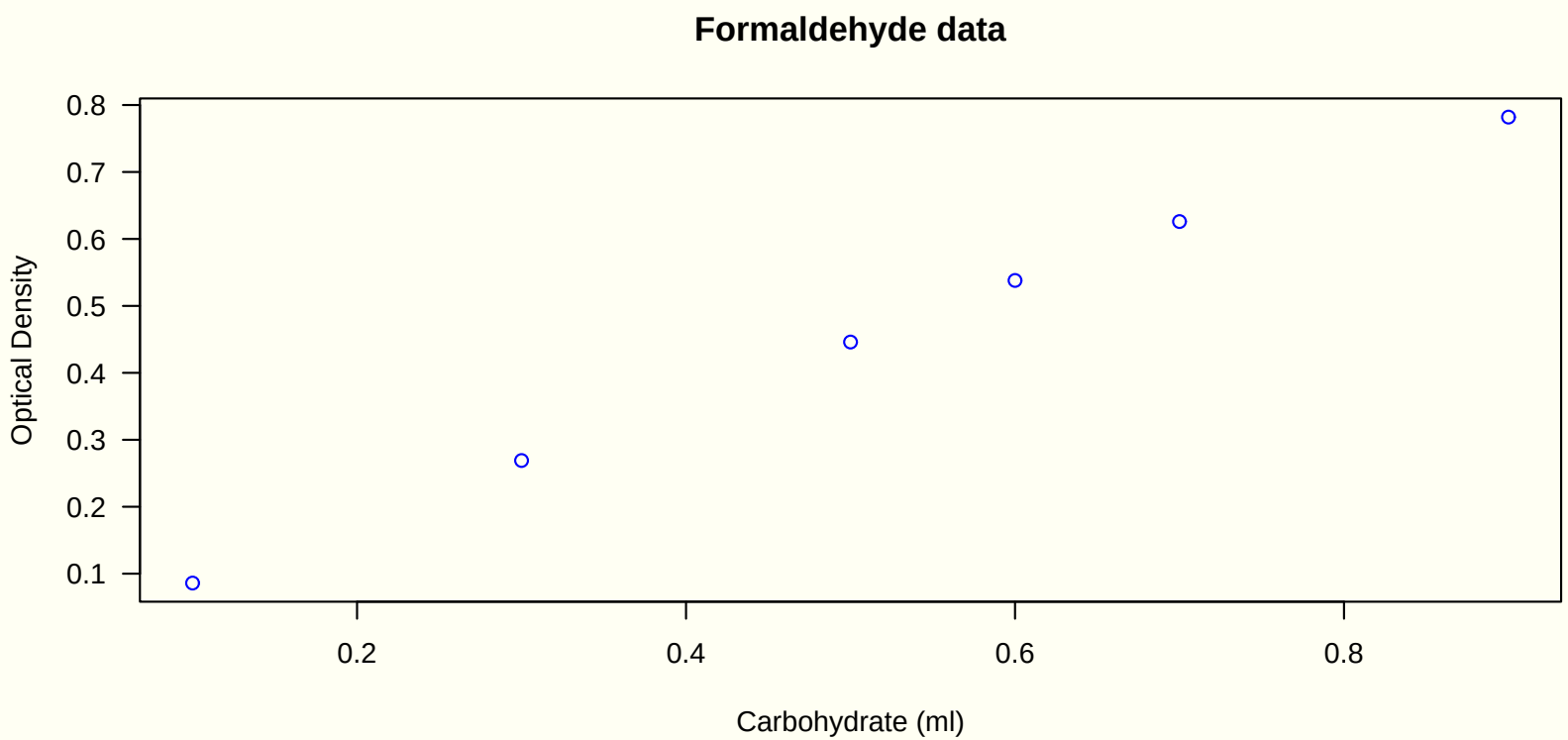
```
bwplot(ssz ~ sim | type, data = alldat, layout = c(1,2))
       main = "Boxplots of means and medians by sample size",
```



Scatter plots

A basic data plot is the scatter plot for x-y data. Recall

```
> data(Formaldehyde)
> plot(optden ~ carb, data = Formaldehyde, xlab = "Carbohydrate (ml)",
+    ylab = "Optical Density", main = "Formaldehyde data", col = 4,
+    las = 1)
```



Common additional graphics parameters

The call to **`plot`** on the previous slide included arguments

- **`xlab`** - x axis label
- **`ylab`** - y axis label
- **`main`** - main title on the plot
- **`col`** - color of the plotted points
- **`las`** - axis label style

Although not required it is common to use these arguments.

Usually it is easy to get a first cut at a data plot, which may be sufficient for exploratory graphics. Presentation graphics can require considerable experimentation with different settings, involving many cycles of editing and rerunning the code. Creating file of **R** code that can be edited and rerun using **`R BATCH`** is a good idea.

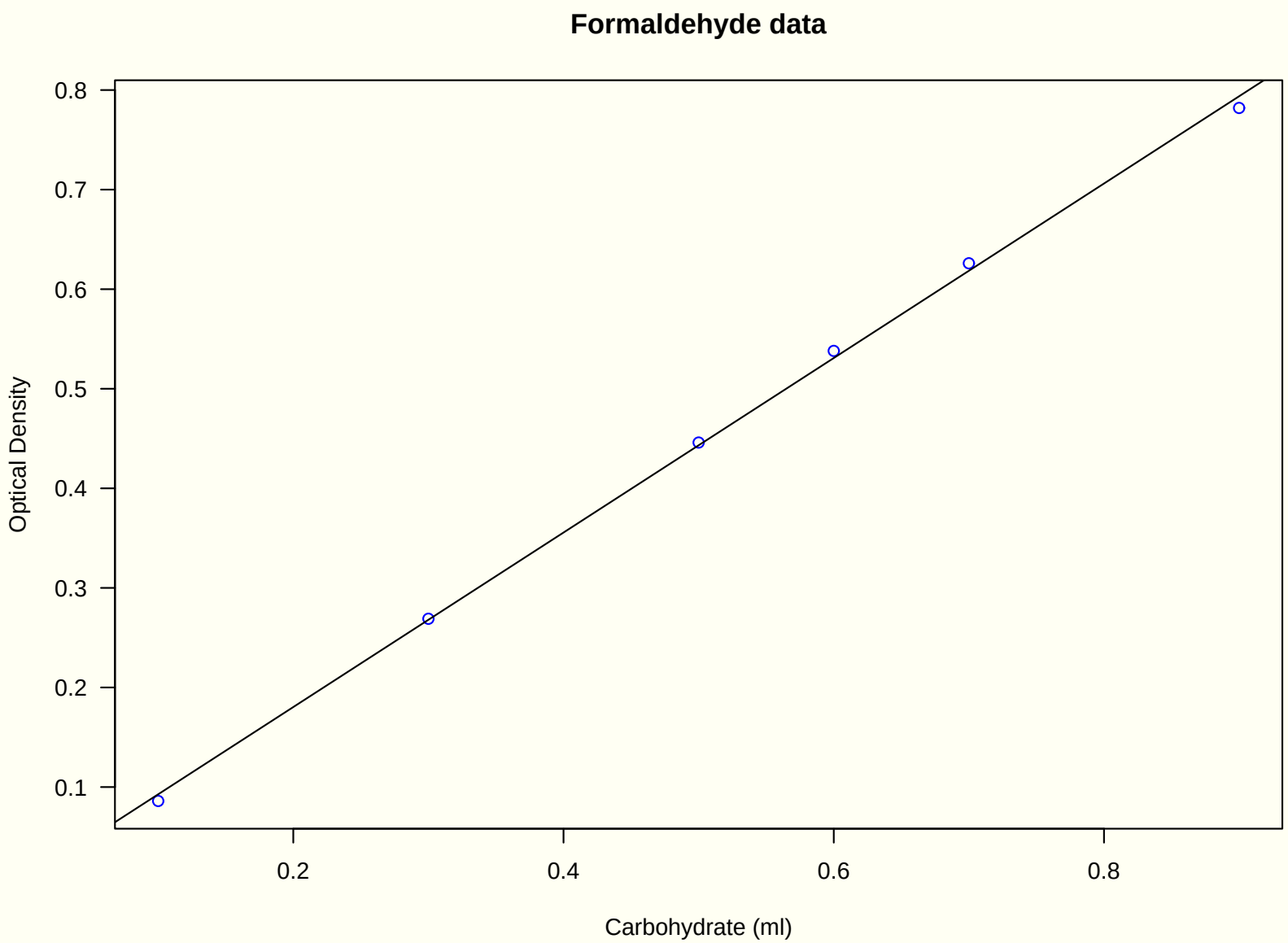
Adding lines to plots

We often use scatter plots to picture relationships between variables then procede to fit statistical models representing these relationships. The **lines** and **abline** functions can be used to add lines to a data plot depicting

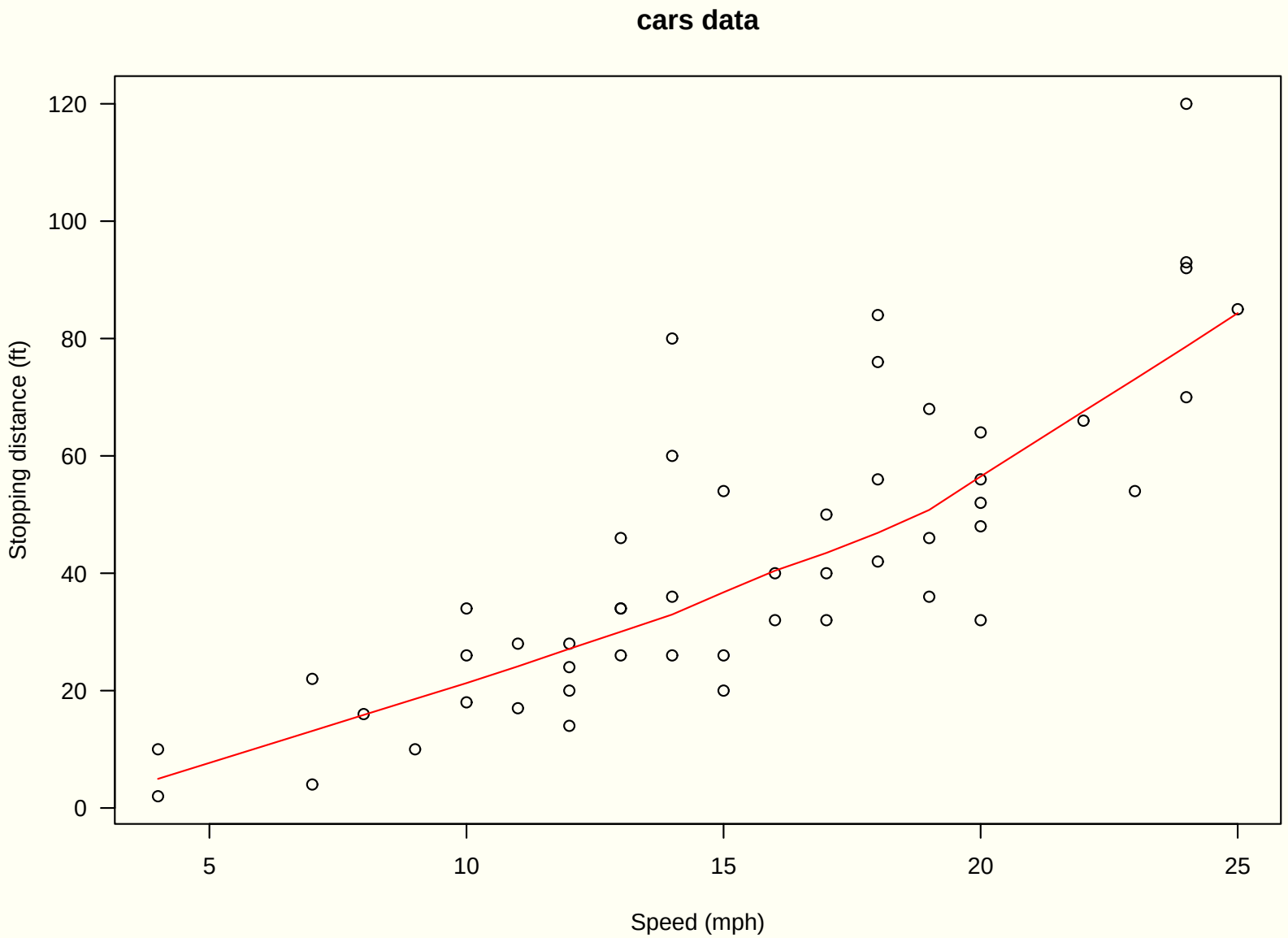
- **scatterplot smoothers** - a smooth curve generated from the **y** versus **x** data. This is added to enhance visualization of the **y ~ x** relationship.
- **predictions from fitted models** - a fitted simple linear regression model can be added directly to a plot with **abline**. For more complicated models, use **predict** to create **(x, y)** pairs to add to the plot with **lines**.

```
> abline(fm1 <- lm(optden ~ carb, data = Formaldehyde))
> data(cars); plot(cars, xlab = "Speed (mph)",
+   ylab = "Stopping distance (ft)", las = 1, main = "cars data")
> lines(lowess(cars$speed, cars$dist, f = 2/3, iter = 3), col = "red")
```

Adding a fitted model



Adding a smoothed curve



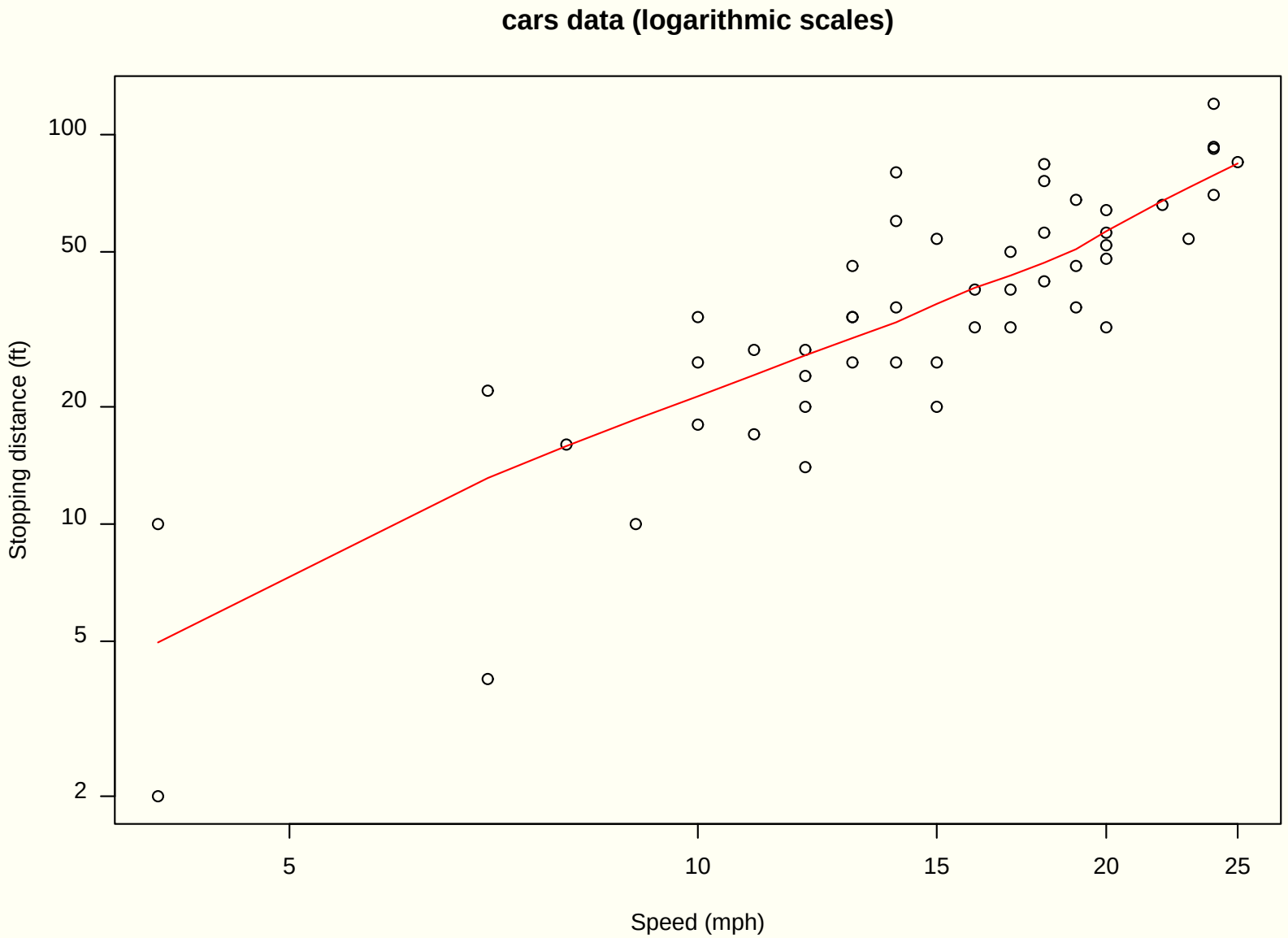
Logarithmic axes

The plot of the stopping distance versus speed for the **`cars`** data shows a curvilinear relationship. It also hints at increasing variance in the stopping distance as the speed (and the mean stopping distance) increase.

In cases like this a logarithmic transformation can stabilize the variance and perhaps produce a simpler relationship. The argument **`log`** is used to request logarithmic axes. It can take the values **`"x"`**, **`"y"`**, or **`"xy"`**.

```
plot(cars, xlab = "Speed (mph)", ylab = "Stopping distance (ft)",
     las = 1, log = "xy", main = "cars data (logarithmic scales)")
lines(lowess(cars$speed, cars$dist, f = 2/3, iter = 3), col = "red")
```

Cars data - logarithmic scales

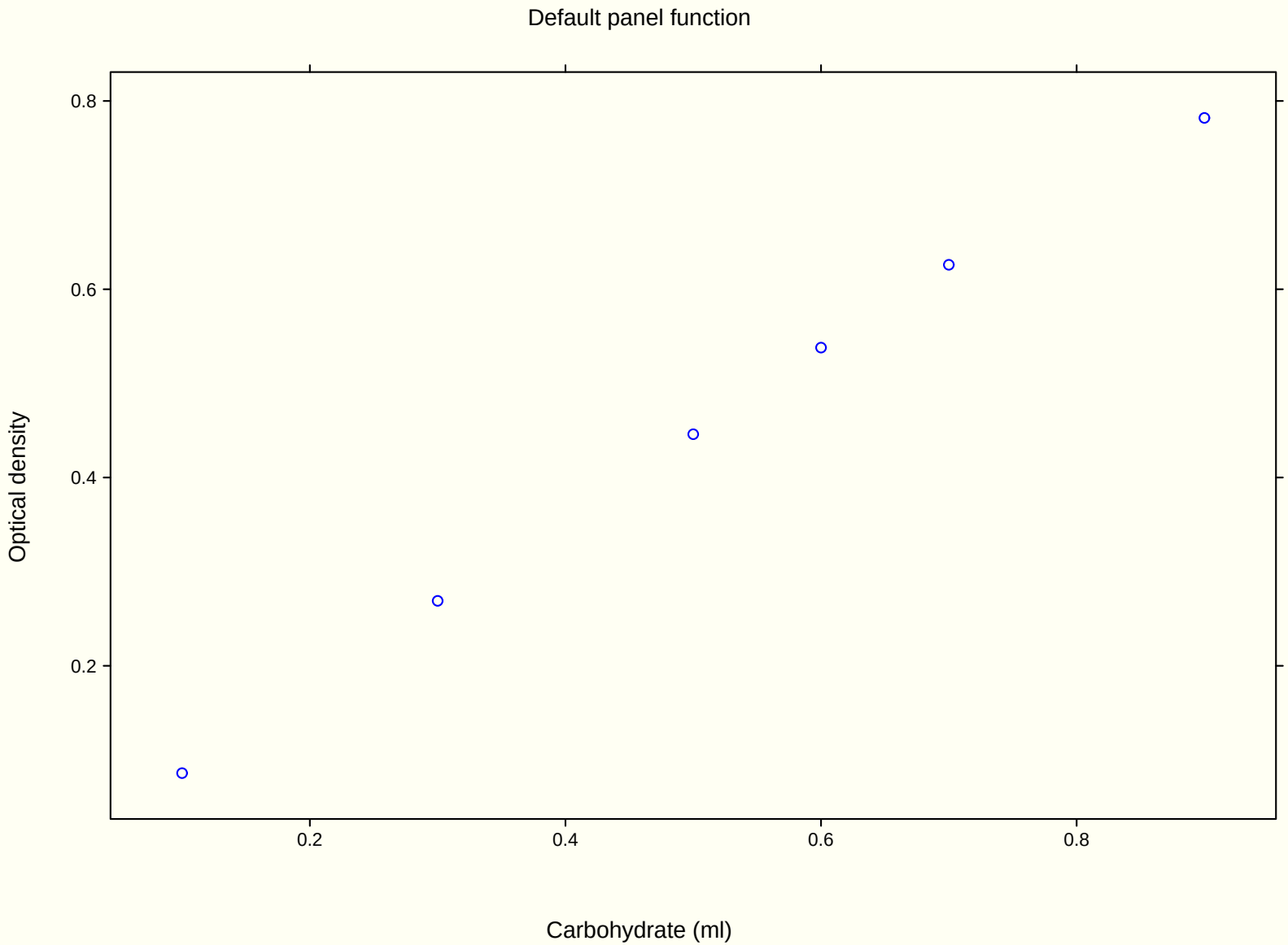


Scatterplots with lattice

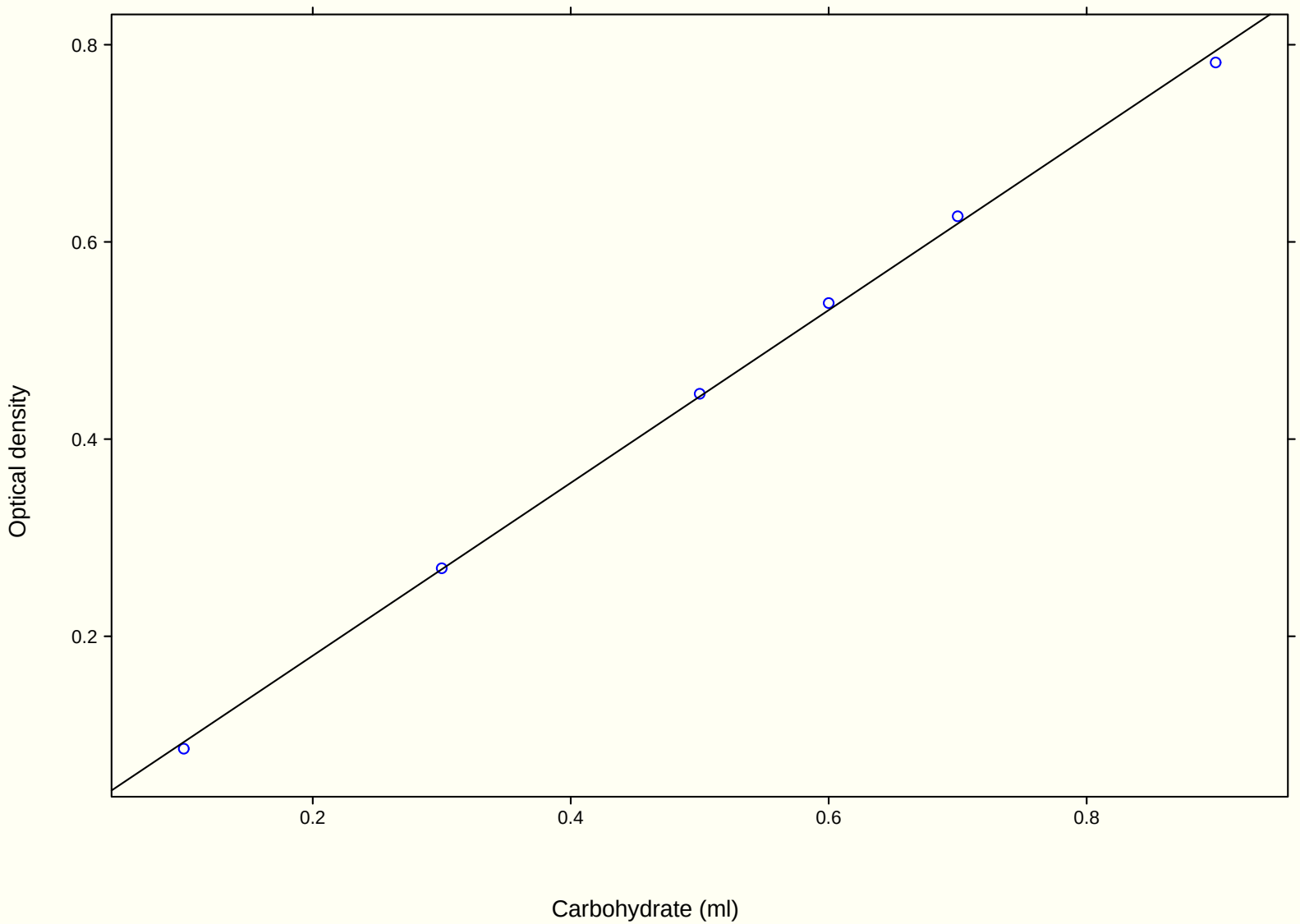
The lattice equivalent to **`plot`** is **`xyplot`**. You must use a formula-data specification with xyplot. Also, you must complete the plot in a single function call. Instead of using multiple calls to plot points then add scatterplot smoothers, etc., you describe all the actions for creating the plot in a *panel function* passed as the **`panel`** argument.

```
> xyplot(optden ~ carb, data = Formaldehyde, xlab = "Carbohydrate (ml)
+        ylab = "Optical density", main = "Default panel function")
> xyplot(optden ~ carb, data = Formaldehyde, xlab = "Carbohydrate (ml)
+        ylab = "Optical density",
+        panel = function(x, y) {panel.xyplot(x,y); panel.lmline(x,y)}
> xyplot(dist ~ speed, data = cars, xlab = "Speed (mph)",
+        ylab = "Stopping distance (ft)",
+        panel = function(x, y) {panel.xyplot(x,y); panel.loess(x,y)})
> xyplot(log(dist) ~ log(speed), data = cars, xlab = "log(Speed) (log(
+        ylab = "log(Stopping distance) (log(ft))",
+        panel = function(x, y) {panel.xyplot(x,y); panel.loess(x,y)})
```

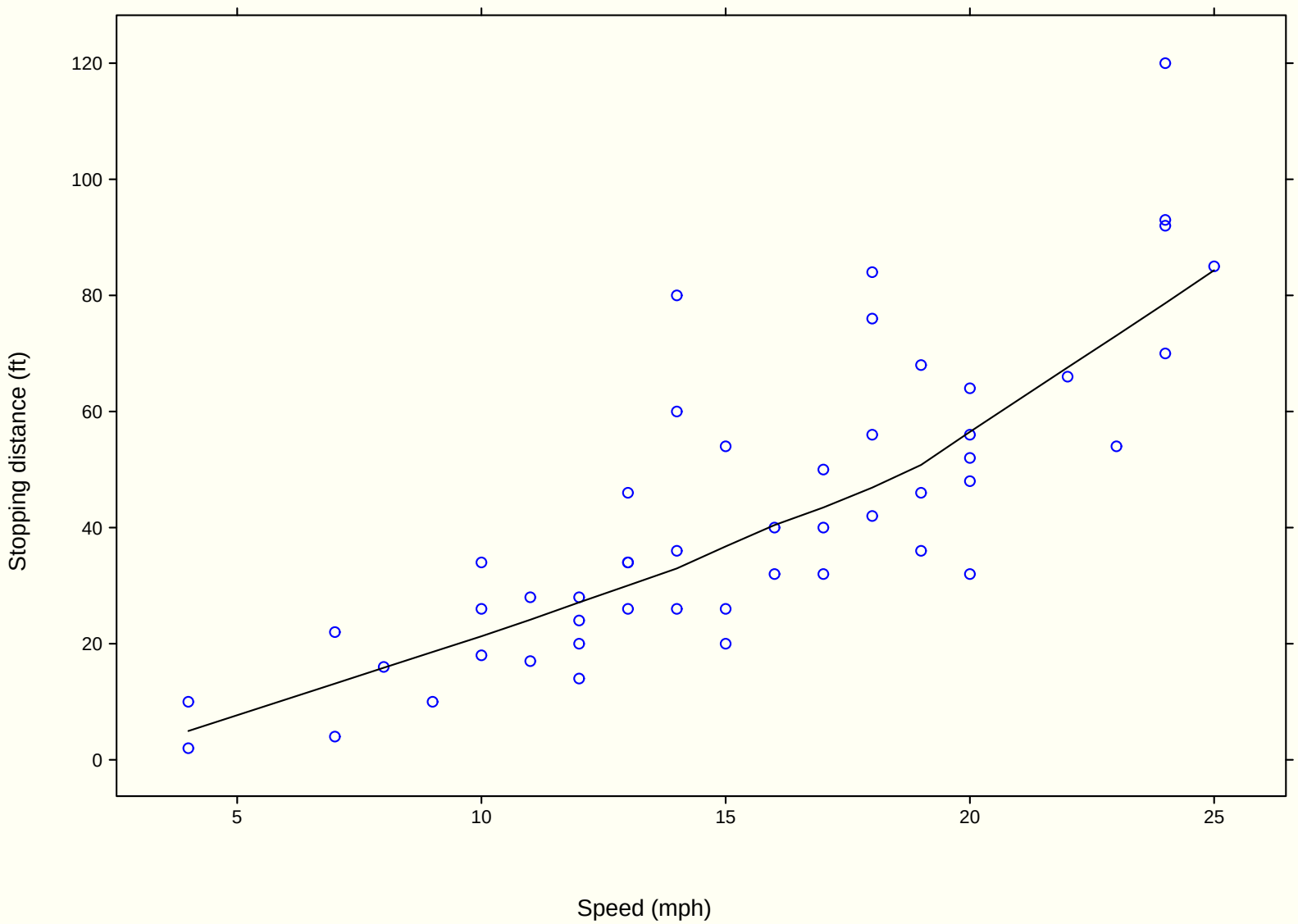
Formaldehyde xyplot with default panel



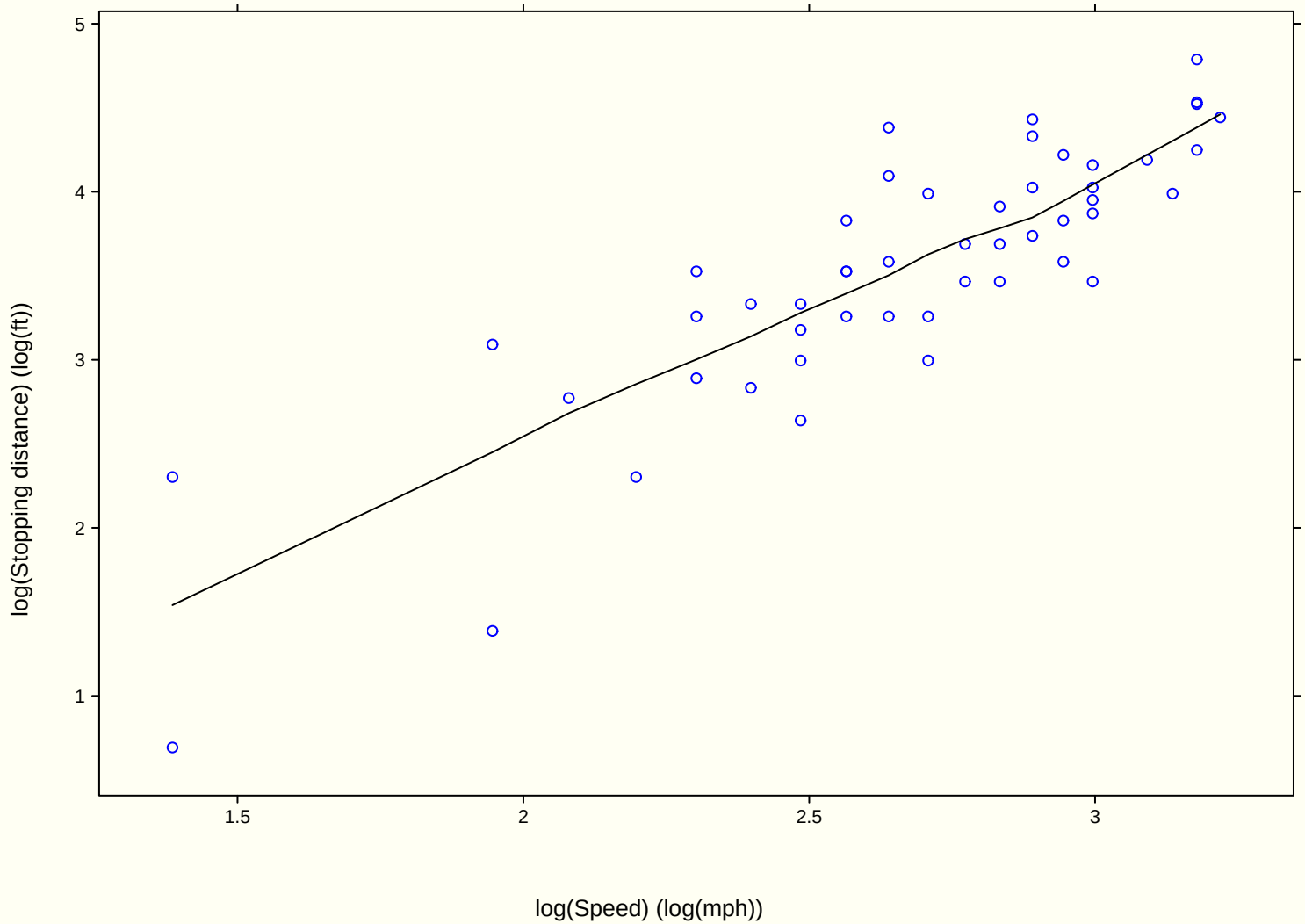
Formaldehyde xyplot with custom panel



Cars xyplot with custom panel



Cars xyplot on the logarithmic scale



There is a **`scales = list(log = TRUE)`** argument for **`xyplot`** but currently it has no effect.

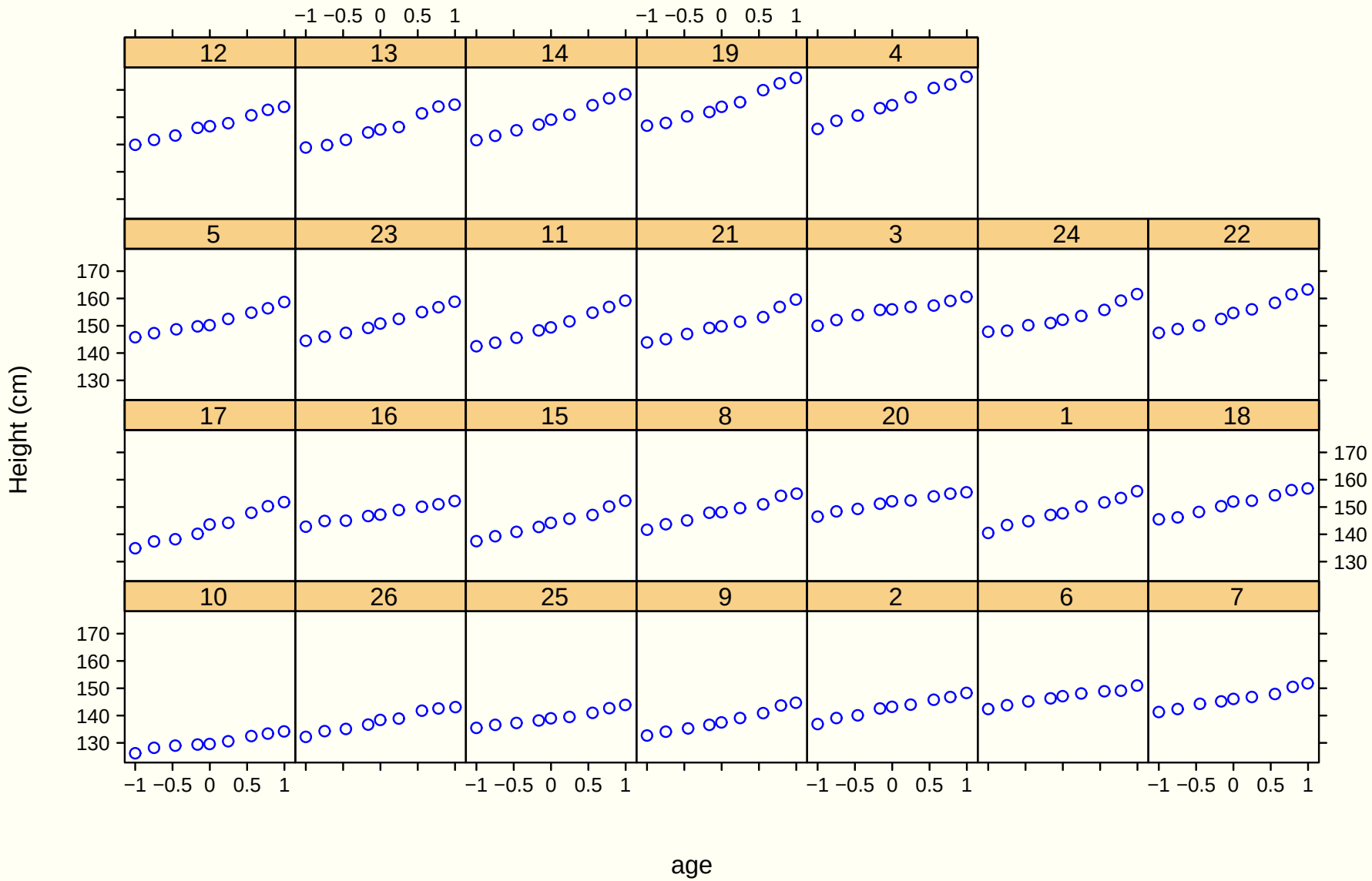
xyplots conditioned on a factor

One of the most powerful uses of trellis-style graphics is comparing patterns of responses across different groups. For example, the **Oxboys** data in the **nlme** package gives the height (cm) of some boys from Oxford, England. Each subject was measured several times throughout his adolescence. The time of measurement was converted to an arbitrary scale centered at the midpoint of the data.

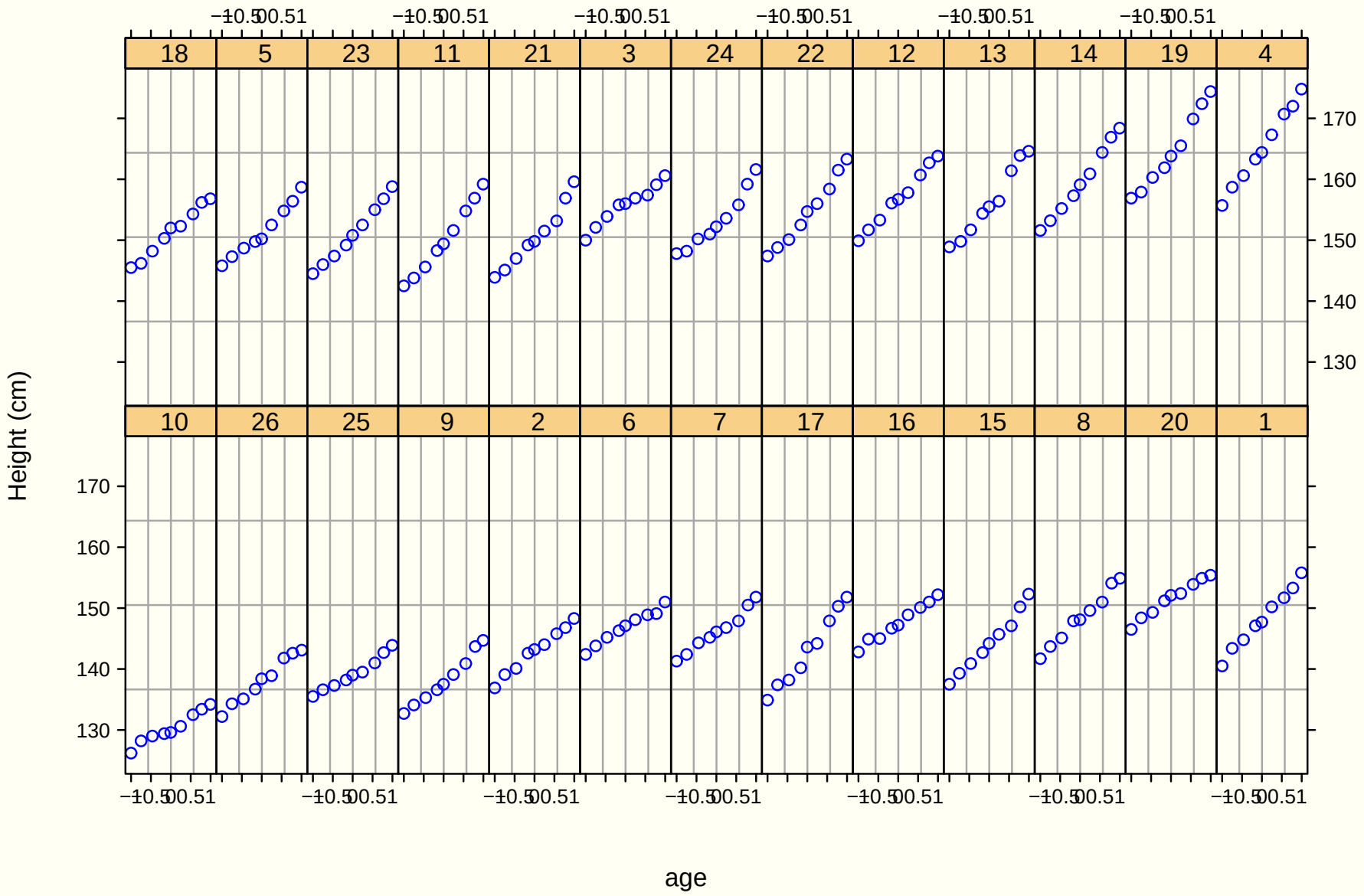
We wish to compare the growth patterns for these boys. To facilitate comparison, the data for each boy is plotted in a separate panel but scale of the axes is the same for each panel.

```
> data(Oxboys, package = nlme)
> xyplot(height ~ age | Subject, data = Oxboys, ylab = "Height (cm)")
> xyplot(height ~ age | Subject, data = Oxboys, ylab = "Height (cm)",
+        aspect = "xy",  # calculate an optimal aspect ratio
+        panel = function(x,y) {panel.grid(); panel.xyplot(x,y)})
```

Oxboys data, standard aspect ratio



Oxboys data, "xy" aspect ratio



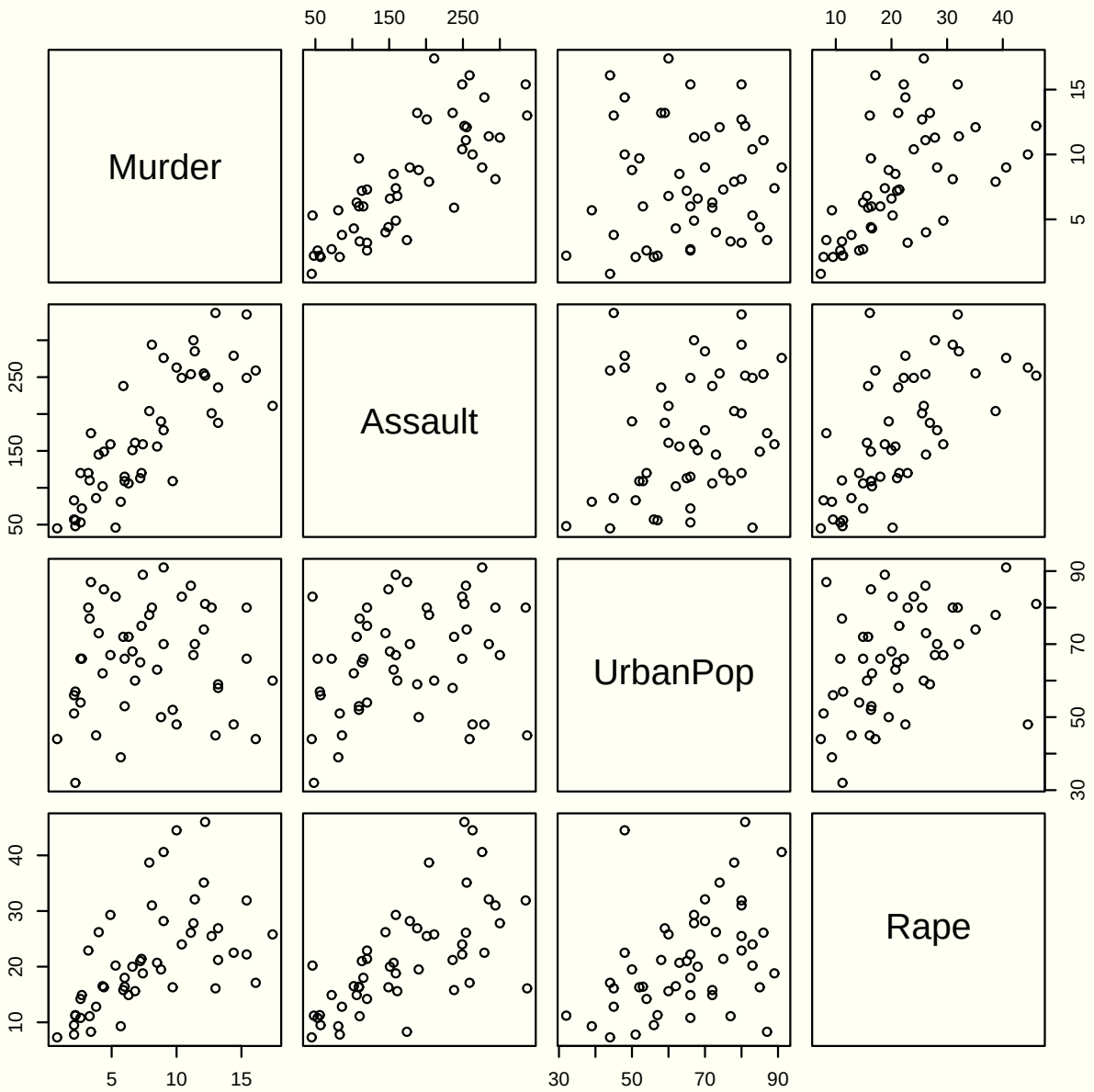
Scatterplot matrices

Scatterplot matrices are helpful in initial exploration of multivariate data. These provide scatterplots of all pairs of variables in the data and are produced by either the high-level graphics function **pairs** or the lattice function **splom**.

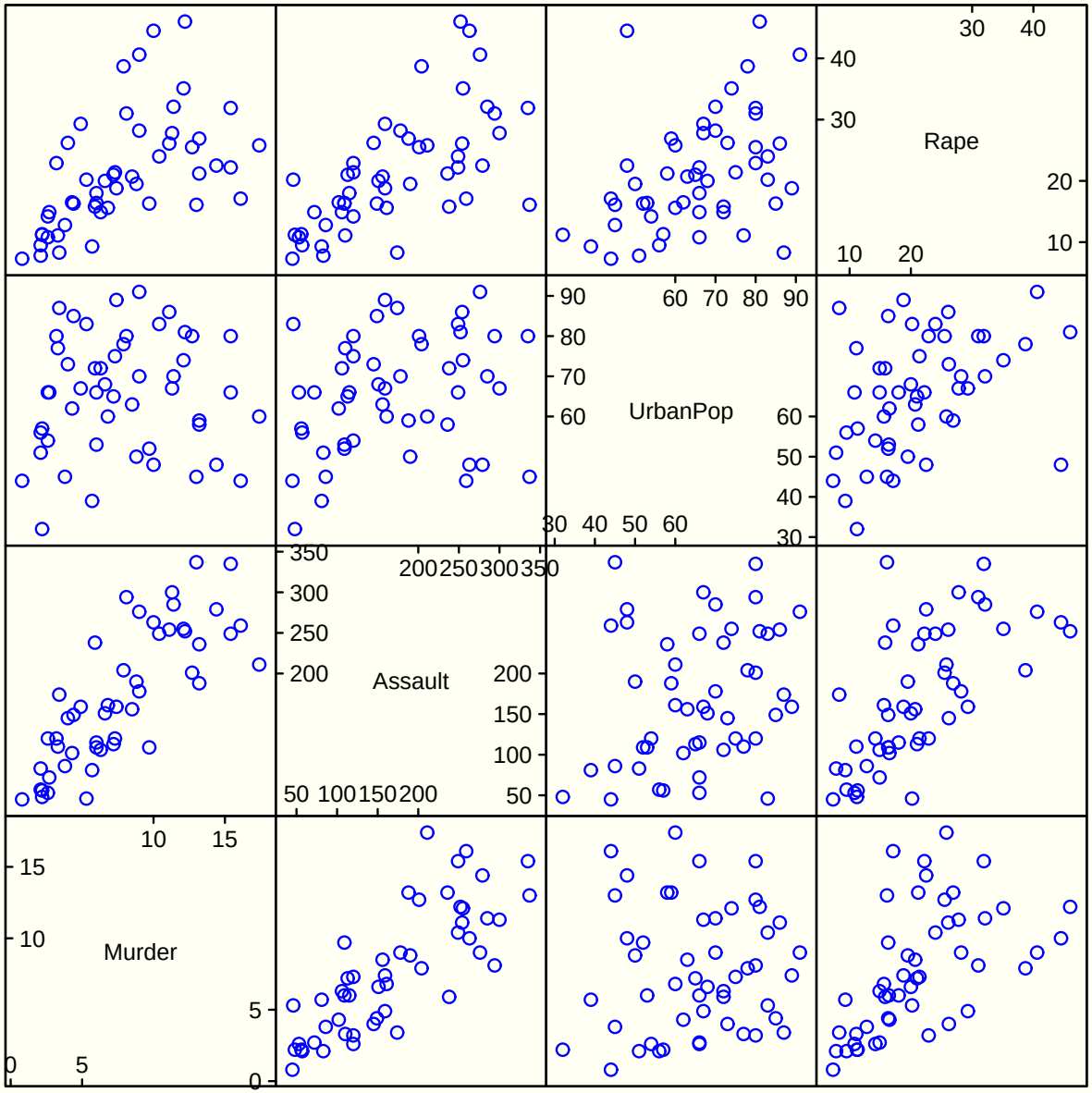
Both **pairs** and **splom** allow a panel function to be specified. Check the documentation for details.

```
> data(USArrests)
> pairs(USArrests)
> splom(~ USArrests)
> splom(~ USArrests,
+       panel = function(x,y) {panel.xyplot(x,y); panel.loess(x,y)})
```

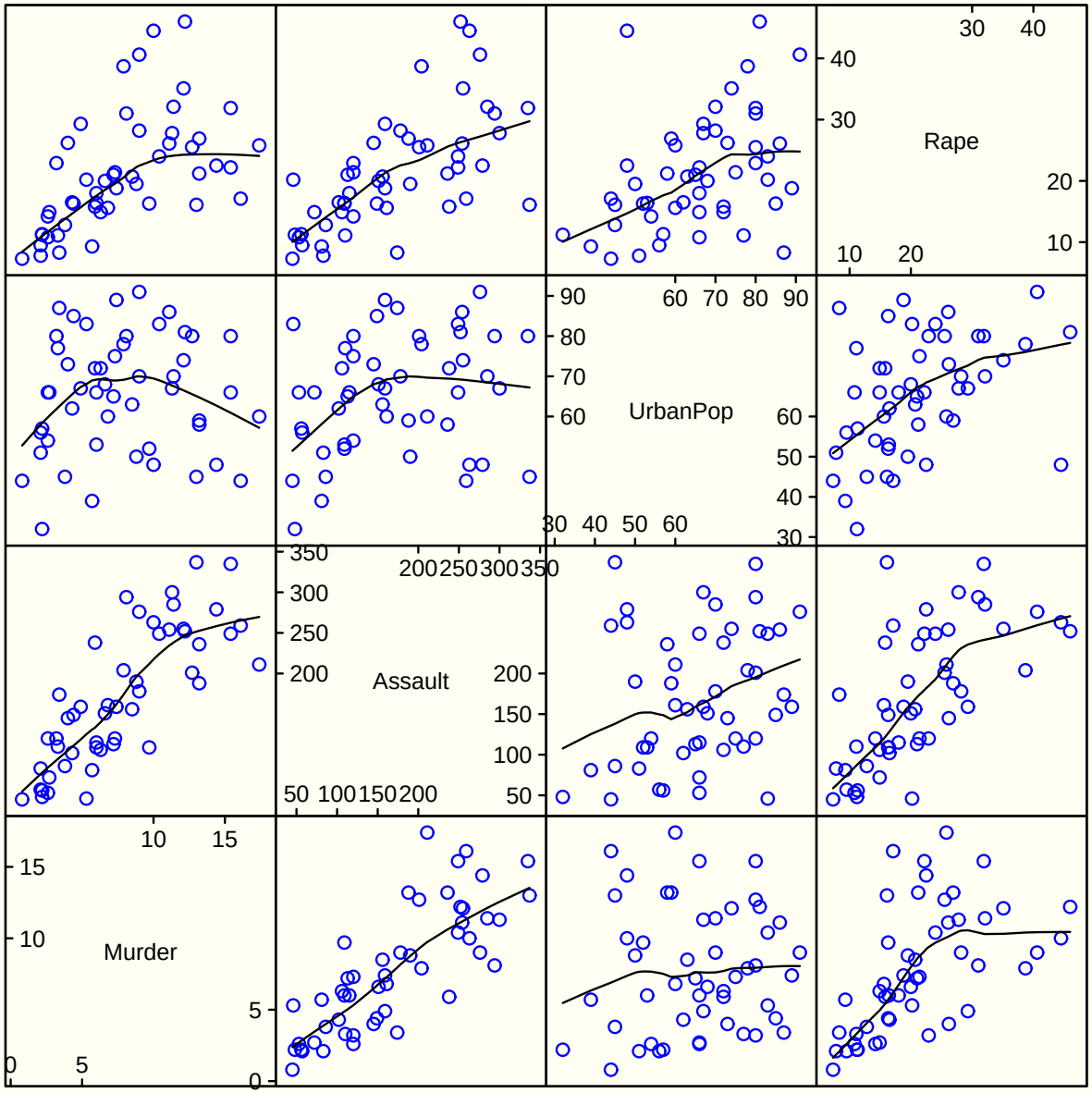
Pairs plot of USArrests data



Default splom plot of USArrests data



Enhanced splom plot of USArrests data



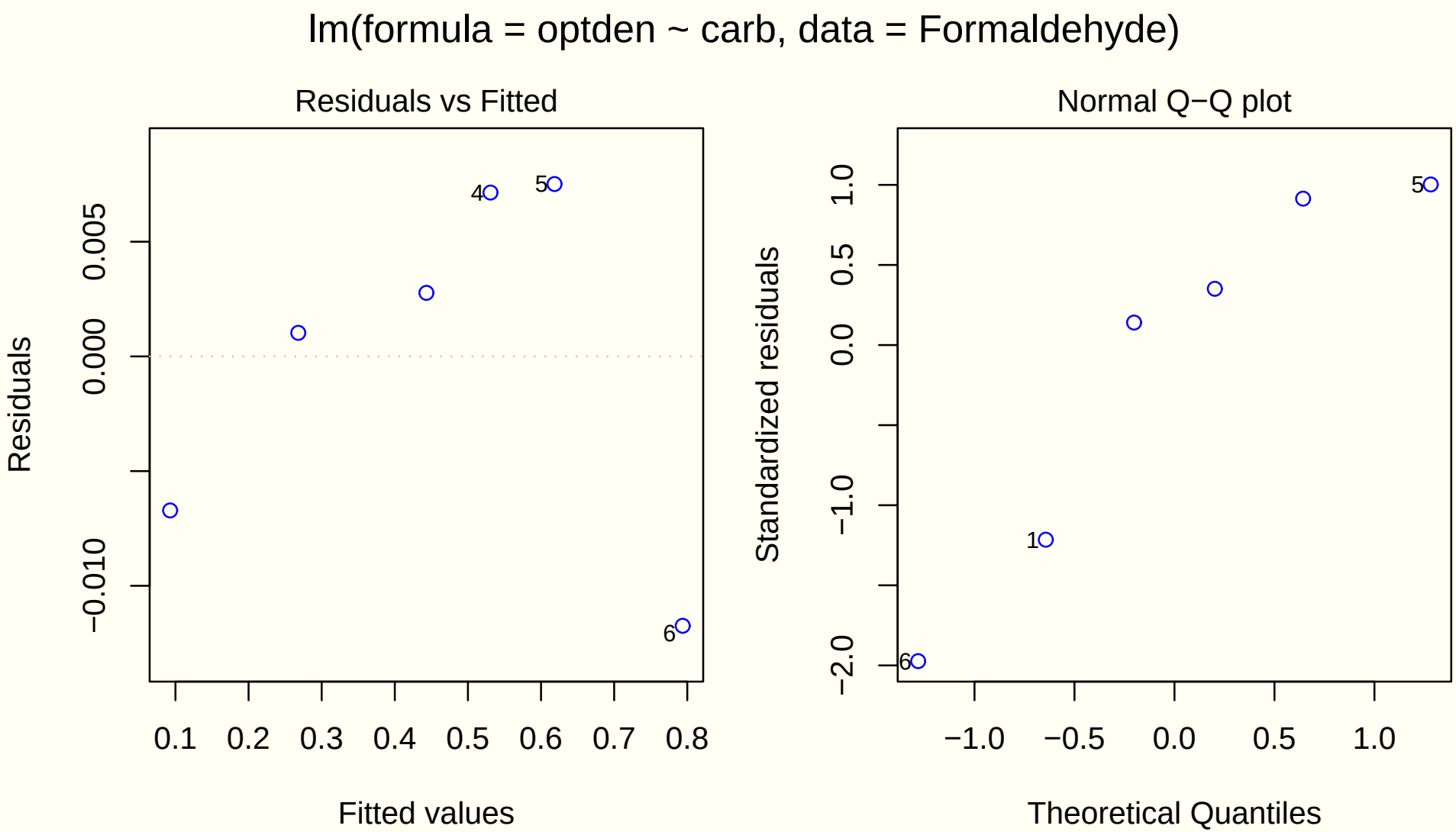
Multiple figures per page

We have seen the use of the lattice package to produce multipanel plots where all the panels are related in some way. Occasionally we wish to put multiple unrelated figures on a page. In the high-level graphics, the **par** function can be used to set the graphics parameters **mfrow** (multiple figures created row-wise) or **mfcol** (multiple figures created columnwise) to do this.

```
> data(Formaldehyde)
> fm1 <- lm(optden ~ carb, data = Formaldehyde)
> par(mfrow = c(1,2))  # the next plot command produces 4 figures
> plot(fm1)
```

Note - Creating an attractive multipanel plot in this way is difficult. The default setting for the axis spacing, etc., do not work well and usually require adjustment. If your multipanel plot can be created in the lattice style, use that instead. When browsing statistics journals one often sees figures created with **par(mfrow =)** that should instead have been done with trellis/lattice.

First two diagnostic plots



Second two diagnostic plots