Part 5: Graphics and Statistics

140.776 Statistical Computing

Ingo Ruczinski

Thanks to Thomas Lumley and Robert Gentleman of the R-core group (http://www.r-project.org/) for providing some tex files that appear in part in the following slides.

Graphical capabilities

One of the strengths of the S language is graphics:

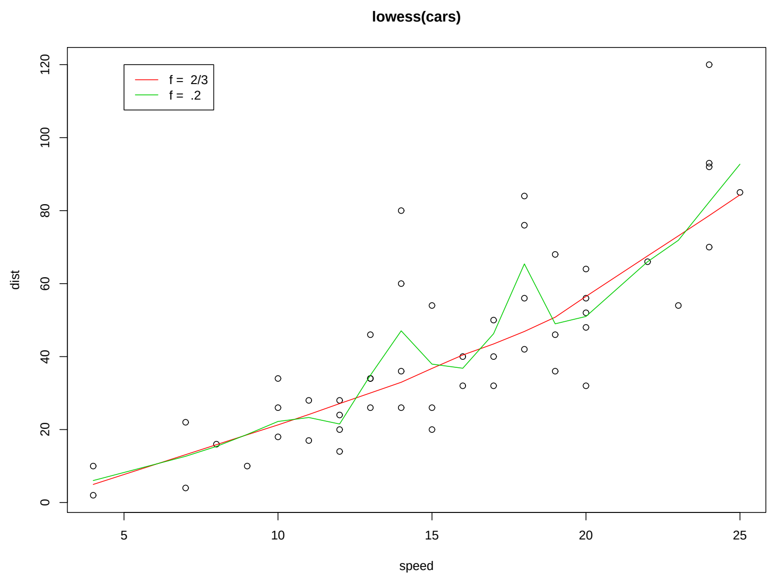
- Simple, exploratory graphics are easy to produce.
- Publication quality graphics can be created.
- Several device drivers are available, including on-screen graphics (such as `X11`), `postscript`, `pdf`, `png`, `jpeg`, `WMF`.

Declaring graphics devices

- The on-screen devices are the devices 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, and then to call a 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.

Declaring graphics devices

```
> postscript("lowess.ps")
> example(lowess)
> dev.off()
```



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 `text`, `axis`, `points`, `lines`.

- Trellis functions:
  Functions such as `histogram`, and `xyplot`, `bwplot` 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.

`par`

```
par                      package:base                    R Documentation

Set or Query Graphical Parameters

Description:
     'par' can be used to set or query graphical parameters. Parameters
     can be set by specifying them as arguments to 'par' in 'tag =
     value' form, or by passing them as a list of tagged values.

Usage:
     par(..., no.readonly = FALSE)
     <highlevel plot> (..., <tag> = <value>)

Arguments:
     ...: arguments in 'tag = value' form, or a list of tagged values.
          The tags must come from the graphical parameters described
          below.
no.readonly: logical; if 'TRUE' and there are no other arguments, only
          parameters are returned which can be set by a subsequent
          'par()' call.
```

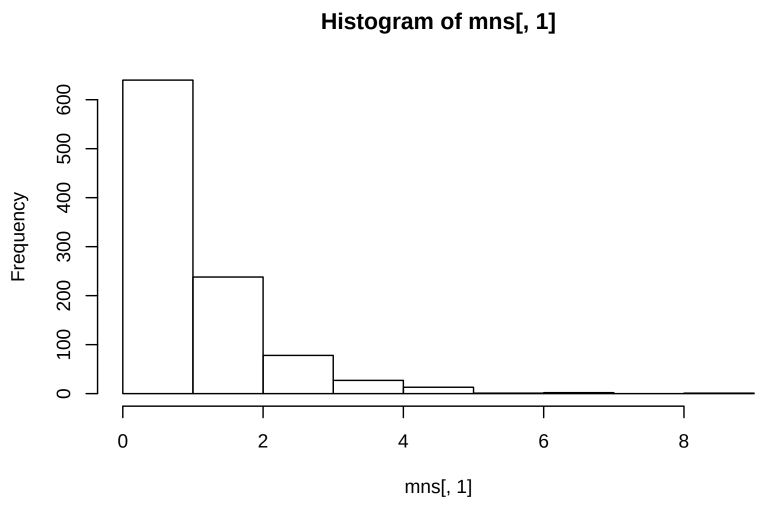
An example

One can 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
+          )
```

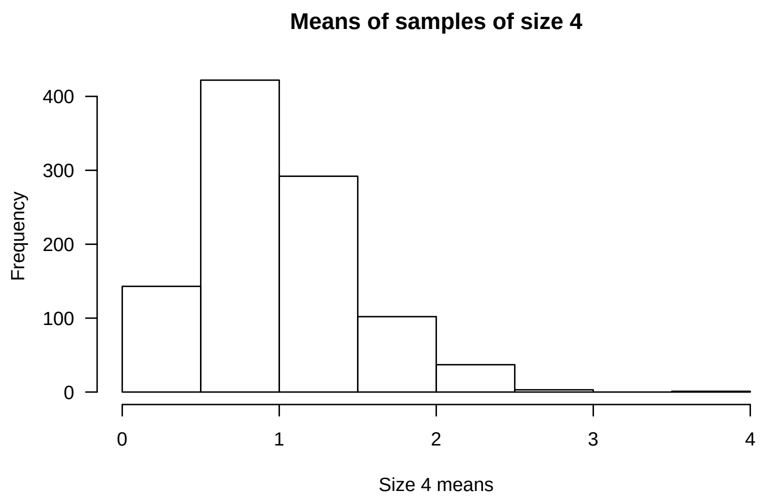
Using high-level plotting functions

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



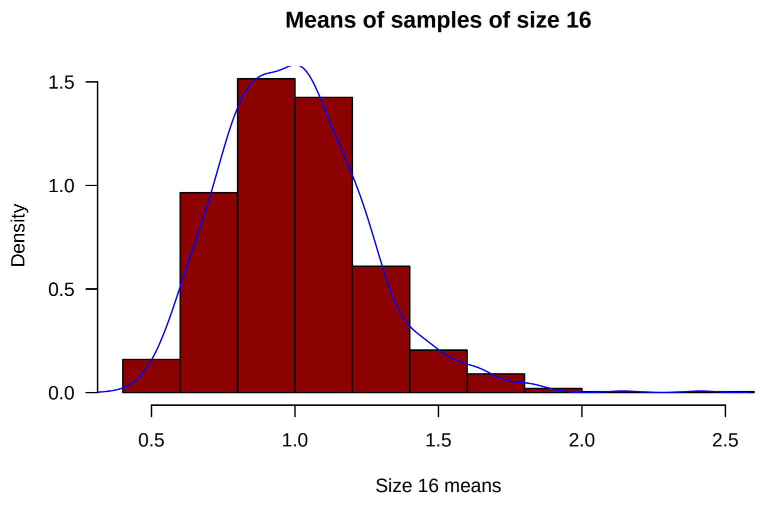
Enhancing high-level plots

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



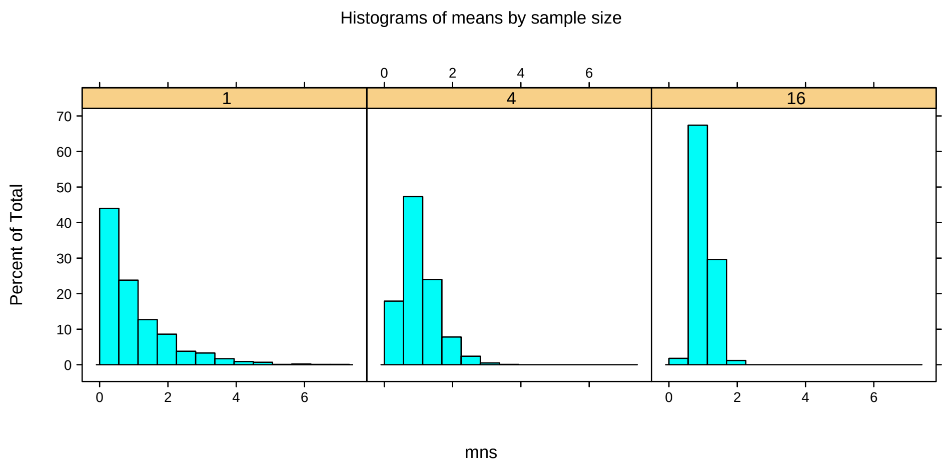
Using low-level graphics functions

```
> 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 size")
```



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, it is a good idea to first construct a data frame with the data to be plotted. One possibility is to stack all the data into a single column with accompanying columns that indicate the groups of observations.

Stacking the simulation data

We arrange the simulated data of means and medians 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 ...
 $ type: Factor w/ 2 levels "Mean","Median": 1 1 1 1 1 1 1 1 ...
```

Formulas specifying plots

The general form of a formula specifying a plot is

```
y ~ x | g
```

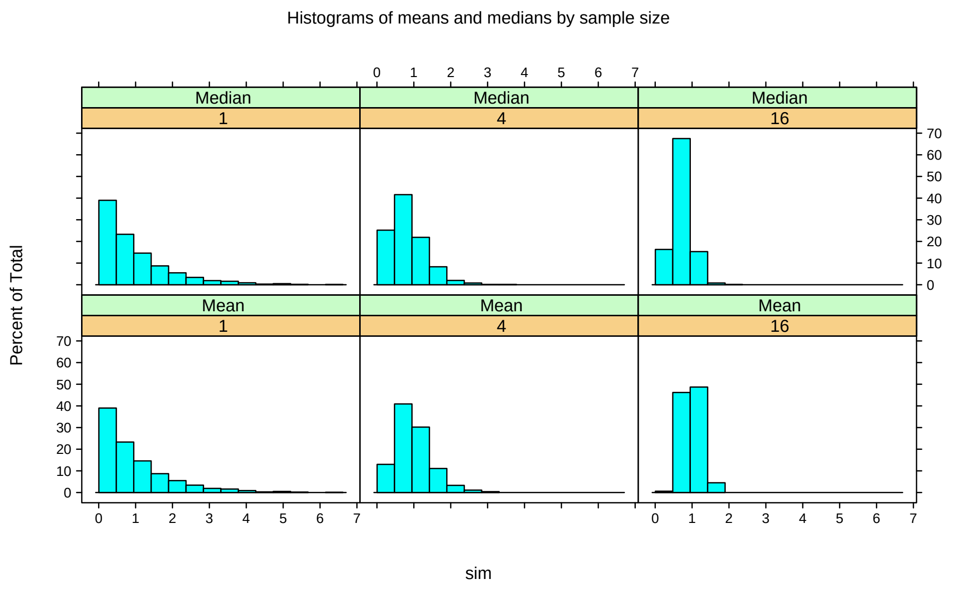
where y is assigned to the vertical axis, x is assigned to the horizontal axis, and g is a grouping factor or expression.

- 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.

Example of multiple grouping factors

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

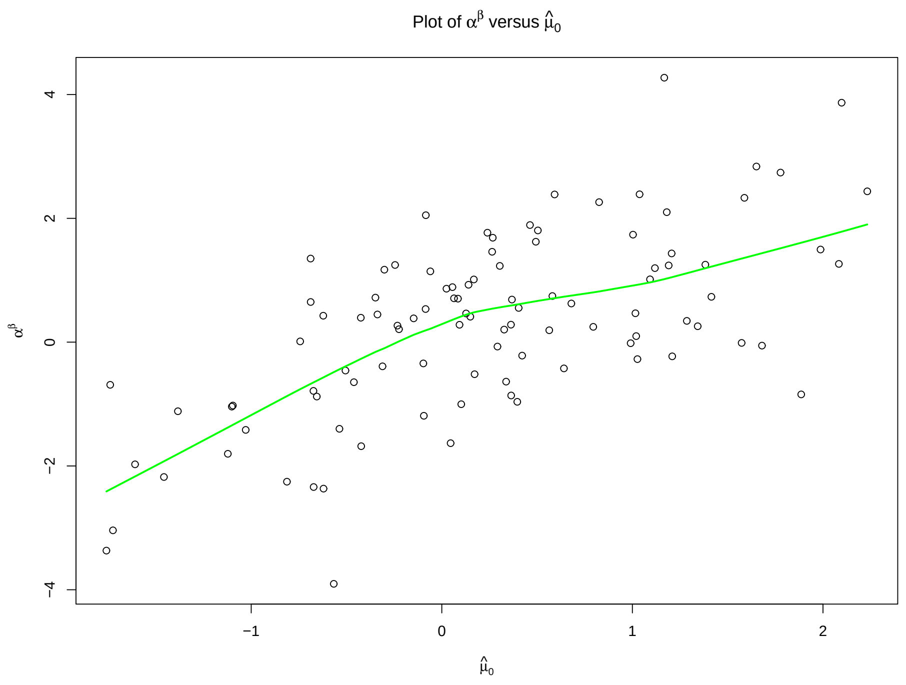


Color and LaTeX symbols

One of the nice additions to R (relative to Splus) is the easy inclusion of mathematical expressions in plots using the function `expression`. Take a look at `help(plotmath)` to see a big list of what you can do; also look at the examples in the help file for the function `legend`.

```
x <- rnorm(100)
y <- x+rnorm(100)
plot(x,y,xlab=expression(hat(mu)[0]),ylab=expression(alpha^beta),
     main=expression(
     paste("Plot of ",alpha^beta," versus ",hat(mu)[0])))
lines(lowess(x,y),col="green",lwd=2)
```

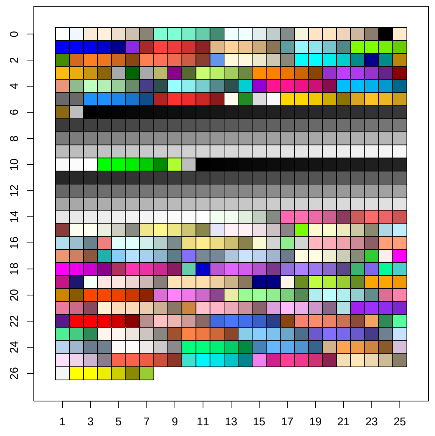
Color and LaTeX symbols



Color and LaTeX symbols

Another advantage of R compared to Splus is the much simpler control of colors. There are plenty of pre-defined colors:

```
> str(colors())
 chr [1:657] "white" "aliceblue" "antiquewhite" "antiquewhite1"
             "antiquewhite2" "antiquewhite3" "antiquewhite4"
             "aquamarine" "aquamarine1" "aquamarine2" ...
```



Color and LaTeX symbols

You can define pretty much any color you like using the function `rgb`. Check out the help file.

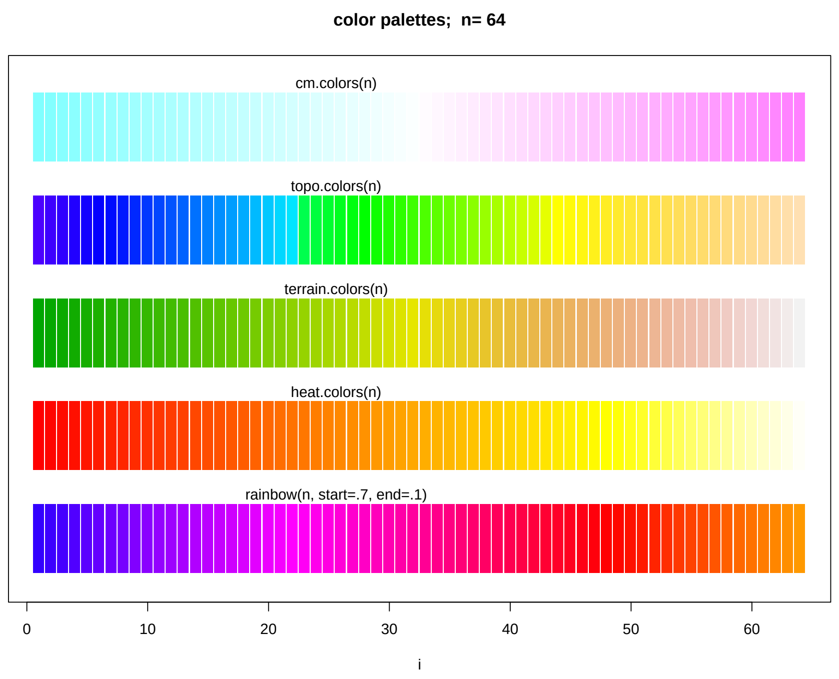
```
> rgb(0.5,0.2,0.9)
[1] "#8033E6"

> col2rgb("aquamarine")
      [,1]
red    127
green  255
blue   212
```

Some “colors” are functions!

```
> grey(0.7)
[1] "#B2B2B2"
```

Color and LaTeX symbols

```
> example(rainbow)
```



Statistical Modelling in S

The systematic part of a model is specified a model formula with basic structure

```
outcome ~ exposure*modifier + confounder
```

- The left-hand side is the outcome (response) variable, the right-hand side describes the predictors.
- The `*` specifies an interaction and the corresponding main effects (`a:b` specifies just the interaction term).
- Factors (e. g. race, subtype of disease) are coded by default with indicator variables for all except the first category.

Statistical Modelling in S

- `depress~rural*agegp+partner+parity+income`

  Does the risk of postnatal depression vary between urban and rural areas, separately for each age group, adjusted for having a domestic partner, previous number of pregnancies, income?
- `asthma~pm25+temp+I(temp^2)+month`

  How does the number of hospital admissions for asthma vary with the fi ne particulate air pollution, adjusted for temperature and month of the year?
- `log(pm25)~temp+stag+month+lag(temp,1)`

  Predict log-transformed fi ne particulate air pollution from temperature, air stagnation, month, and yesterday’s temperature.
- `Surv(ttoMI,MI)~LDL+age+sex+hibp+diabetes`

  How does LDL cholesterol predict (time to) myocardial infarction after adjusting for age, sex, hypertension, and diabetes?

Example: generalized linear models

Generalized linear models (linear regression, logistic regression, poisson regression) are handled by the `glm()` function. This requires:

- A model formula.
- A dataframe containing the variables [optional].
- A model family such as given by `binomial()`, `gaussian()`, and `poisson()`.

Example:

```
glm(asthma~pm25+temp+I(temp^2)+month,
            data=pmdat,family=poisson())
```

Model objects

Typical statistics packages fit a model and output the results. In S a model object is created that stores all the information about the fitted model. Coefficients, diagnostics, and other model summaries are produced by methods for this object.

- `coef(model)` returns the coefficients.
- `summary(model)` gives a table with coefficients, standard errors, and other statistics.
- `resid(model)` returns (various flavours of) residuals.
- `anova(model)` gives an ANOVA table showing likelihood ratio tests for adding each term sequentially. Also, the function `anova(model1,model2)` compares the two models directly.
- `plot(model)` may give some useful diagnostic plots.

Classes of model

R has most of the commomly used regression models:

- Linear regression `lm()`,
- Generalized linear models `glm()`,
- Cox proportional hazards model `coxph()`,
- Parametric survival models `survreg()`,
- Conditional logistic regression `clogit()`,
- Generalized estimating equations `gee()`,
- Linear mixed models `lme()`.

The functions `lm()` and `glm()` are in the R `base` package, the others are in the `survival` package,`gee` package, and `nlme` package respectively.