Introduction to General and Generalized Linear Models

Introduction to R

Anders Nielsen, Henrik Madsen, Jan Kloppenborg

Informatics and Mathematical Modelling
Technical University of Denmark
DK-2800 Kgs. Lyngby

January 2012

This lecture

Introduction to the R software we will be using in this course

What is R

- Environment to carry out statistical analysis
- Based on "**S**" which was developed by John M. Chambers (Bell Lab)
- Received the prestigious ACM (Association for Computing Machinery) award in 1998
- Quote from the award: "For The S system, which has forever altered how people analyze, visualize, and manipulate data"
- Open source and free
- Simple calculator

```
> 2 + 2
[1] 4
```

What is R

- Vector, matrix, and common linear algebra stuff

```
> x <- c(1, 2)
> A <- matrix(c(1, 2, 3, 4), nrow = 2)
> solve(A, x)

[1] 1 0
```

- Simulating random numbers

```
> x <- runif(20, 0, 10)
> y <- 2 * x - 3 + rnorm(20, sd = 1.5)
```

- Statistical modeling

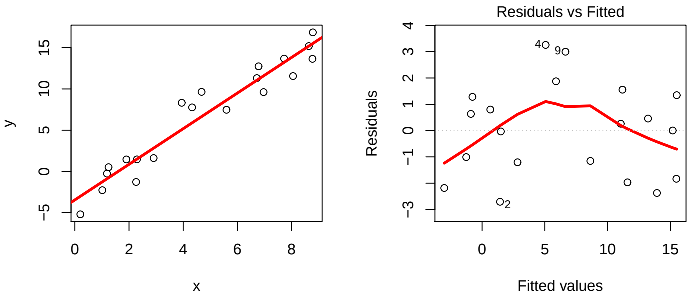
```
> coef(lm(y ~ x))

(Intercept)           x
  -3.454192    2.159108
```

What is R

- Graphics is excellent in R:

```
> par(mfrow = c(1, 2))
> plot(x, y)
> abline(lm(y ~ x), col = "red", lwd = 3)
> plot(lm(y ~ x), which = 1, lwd = 3)
```



What is R

- R is a fairly complete programming language
- A good editor is useful for writing longer programs
- Emacs has an "Emacs Speak Statistics" mode
- On windows the editor "TINN-R" is preferred by many
- These editors allow you to paste code directly into R and have syntax highlighting
- Another acceptable way is to use "your favorite editor" and copy and paste into R
- Run all commands saved in a text file with:

```
> source("myfile.R")
```

Extracting sub-elements in R

- Consider the vector:

```
> x <- c(1:5, -5:-1, 10, 20)
> x
```

```
 [1]  1  2  3  4  5 -5 -4 -3 -2 -1 10 20
```

- Get elements number 5, 6, 7, and 8

```
> x[5:8]
```

```
[1]  5 -5 -4 -3
```

- Get elements below 0

```
> x[x < 0]
```

```
[1] -5 -4 -3 -2 -1
```

- Get all except number 5 and 9

```
> x[-c(5, 9)]
```

```
 [1]  1  2  3  4 -5 -4 -3 -1 10 20
```

Indexing also works for matrices

- The matrix:

```
> A <- matrix((-4):5, nrow = 2, ncol = 5)
> A
     [,1] [,2] [,3] [,4] [,5]
[1,]   -4   -2    0    2    4
[2,]   -3   -1    1    3    5
```

- Get elements below 0

```
> A[A < 0]
[1] -4 -3 -2 -1
```

- Or assign them something else

```
> A[A < 0] <- 0
> A
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    2    4
[2,]    0    0    1    3    5
```

Indexing also works for matrices

- The matrix is still:

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

- pick a row

```
> A[2, ]
[1] 0 0 1 3 5
```

- Or two columns

```
> A[, c(2, 4)]
     [,1] [,2]
[1,]    0    2
[2,]    0    3
```

Writing own functions

- Let's write a simple function

```
> weighted.ave <- function(x, w = rep(1, length(x))) {
+       s1 <- sum(x * w)
+       s2 <- sum(w)
+       return(s1/s2)
+ }
> weighted.ave(c(0, 1, 2, 3, 4), c(7, 3, 10, 17, 21))
[1] 2.724138
```

- Notice we can supply default values

The working space

- To get info about the current working space

```
> getwd()
```

```
[1] "/home/an/02424/slides"
```

- To set the working space to something else

```
> setwd("c:/my/path/to/somewhere")
```

- To save an image of all the current defined variables and functions use

```
> save.image(file = "myStuff.RData")
```

- To load a saved image use

```
> load("myStuff.RData")
```

Reading data from a file

- Often data is organized as below (columns separated with white space, and a line of headings).

```
sex  x  y
M 0.3 0.01
M 1.0 0.11
M 2.1 0.04
F 2.2 0.02
F 0.1 0.10
F 0.2 0.06
```

- Such data can be read in with:

```
> myData <- read.table("datafile.tab", header = TRUE)
```

- The resulting object `myData` is a so-called dataframe.
- A dataframe behaves mostly like a matrix, but not quite.
- Can contain columns of different types.
- Columns can be extracted via the $-operator e.g. `myData$x`

Factors

- Factors are used to describe categories, and a factor in R knows how many categories is has.

```
> x <- rep(1:5, each = 3)
> x
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
> f <- factor(x)
> f
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
Levels: 1 2 3 4 5
> is.factor(x)
[1] FALSE
> is.factor(f)
[1] TRUE
```

Fitting linear models

- General linear models `lm()`
- Generalized linear models `glm()`
- Mixed effects linear models `lme()` (in package nlme).

Model formulas

- A specified model can e.g. look like

  ```
  > fit <- lm(y ~ x + f + g:h + k:z)
  ```

  which would correspond to:

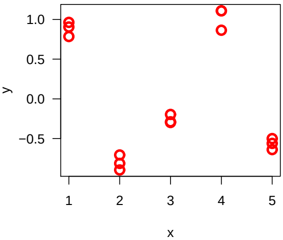
  $$y_i = \mu + \alpha x_i + \beta(f_i) + \delta(g_i, h_i) + \gamma(k_i) z_i + \varepsilon_i$$

- Which are factors?
- Interactions between two factors is different from interaction between factor and covariate.
- What does interactions between two covariates mean?
- Interactions are specified with e.g. `f:g`
- A shorthand for specifying both main effects and interaction effects is `f*g` (same as `f+g+f:g`)
- Adding a `-1` to the formula will get rid of the common intercept μ

Quick example

- What are we doing here?

```
> x <- rep(1:5, each = 3)
> y <- sin(2 * x) + rnorm(15, sd = 0.1)
> f <- factor(x)
> coef(lm(y ~ x))
(Intercept)           x
  0.3767725  -0.1069740
> coef(lm(y ~ f - 1))
        f1         f2         f3         f4         f5
 0.8841206 -0.8044325 -0.2608534  1.0267678 -0.5663493
```



- Are any of these models useful?

Build in distributions

- R has quite a few build in distributions. The naming convention is:
  `d<name>(x)` Density function
  `p<name>(x)` Cumulated density function (probability $\leq x$)
  `q<name>(p)` Quantile (the point x where the probability $\leq x$ is p)
  `r<name>(n)` Simulate random numbers from the distribution
- A basic R installation has: `beta`, `binom`, `cauchy`, `chisq`, `exp`, `f`, `gamma`, `geom`, `hyper`, `logis`, `multinom`, `nbinom`, `norm`, `pois`, `signrank`, `t`, `tukey`, `unif`, `weibull`, `wilcox`
- Is anything missing?
- The multivariate normal, but that is in an extension package

```
> install.packages("mvtnorm")
> library(mvtnorm)
```

- Now we also have: `dmvnorm`, `pmvnorm`, `qmvnorm`, `rmvnorm`,

Basic control-flow

- The matrix

```
> A <- matrix((-4):5, nrow = 2, ncol = 5)
```

- Sum all positive elements (in a sub-optimal way)

```
> S <- 0
> for (i in 1:nrow(A)) {
+     for (j in 1:ncol(A)) {
+         if (A[i, j] > 0) {
+             S <- S + A[i, j]
+         }
+     }
+ }
> S
[1] 15
```

- Would be better to use

```
> sum(A[A > 0])
[1] 15
```

Loop avoidance

- Generally speaking loops are slow, and can mostly be avoided
- Use build in functions (`sum()`, `min()`, `max()`, `which()`, `which.min()`, `rowMeans()`, `colMeans()`, `rowSums()`, `colSums()`,...)
- Use linear algebra
- Use the build in apply functions. A few examples are:
  - `apply()` Use a function over one index (or more) of a matrix (array)
  - `tapply()` Use a function within a number of groups
  - `lapply()` Use a function for each element in a list

```
> apply(A, 2, sd)
[1] 0.7071068 0.7071068 0.7071068 0.7071068 0.7071068
```

- If none of the above does the trick, then it is possible to implement the computer intensive part in a small piece of C code and call that from within R (see e.g. help for `.C()`).

Getting help

- From within R simply type "?" followed by the function name e.g:

  ```
  > ?dnorm
  > ?"for"
  ```

- Many manuals and reference cards at http://www.r-project.org
- http://stackoverflow.com/questions/tagged/r
- http://www.google.com
- Very helpful community on mailing list.