Introduction to R

John Fox

McMaster University

ICPSR 2010

Outline

- Getting Started with R
- Statistical Models in R
- Data in R
- R Programming
- R Graphics
- Building R packages (or another topic)
A statistical programming language and computing environment, implementing the S language.

Two implementations of S:

- S-PLUS: commercial, for Windows and (some) Unix/Linux, eclipsed by R.
- R: free, open-source, for Windows, Macintoshes, and (most) Unix/Linux.

How does a statistical programming environment differ from a statistical package (such as SPSS)?

- A package is oriented toward combining instructions and rectangular datasets to produce (voluminous) printouts and graphs. Routine, standard data analysis is easy; innovation or nonstandard analysis is hard or impossible.
- A programming environment is oriented toward transforming one data structure into another. Programming environments such as R are extensible. Standard data analysis is easy, but so are innovation and nonstandard analysis.
Among statisticians, R has become the de-facto standard language for creating statistical software. Consequently, new statistical methods are often first implemented in R.

There is a great deal of built-in statistical functionality in R, and many (literally thousands of) add-on packages available that extend the basic functionality.

R creates fine statistical graphs with relatively little effort.

The R language is very well designed and finely tuned for writing statistical applications.

(Much) R software is of very high quality.

R is easy to use (for a programming language).

R is free (in both of senses: costless and distributed under the Free Software Foundation’s GPL).

The purpose of this lecture series/workshop is to get participants started using R.

The statistical content is largely assumed known.


Additional materials and links are available on the web site for the first edition of the book:

http://socserv.socsci.mcmaster.ca/jfox/Books/Companion/index.html

The book is associated with an R package (called **car**) that implements a variety of methods helpful for analyzing data with linear and generalized linear models.
Other references are given on the workshop web site.
- Workshop web site: 
  http://socserv.socsci.mcmaster.ca/jfox/Courses/R-course/index.html

Statistical Models in R

Topics

- Multiple linear regression
- Factors and dummy regression models
- Overview of the `lm` function
- The structure of generalized linear models (GLMs) in R; the `glm` function
- GLMs for binary/binomial data
- GLMs for count data
- Traditional ANOVA and MANOVA for repeated-measures designs (time permitting)
### Arguments of the `lm` function

- **formula**: expression that specifies the model. The terms in the model can include:  
  - `A + B` to include both `A` and `B`  
  - `A - B` to exclude `B` from `A`  
  - `A:B` for all interactions of `A` and `B`  
  - `A*B` for `A + B + A:B`  
  - `B %in% A` for `B` nested within `A`  
  - `A/B` for `A + B %in% A`  
  - `A^k` for effects crossed to order `k`

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interpretation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>A + B</td>
<td>include both A and B</td>
<td>income + education</td>
</tr>
<tr>
<td>A - B</td>
<td>exclude B from A</td>
<td>a<em>b</em>d - a:b:d</td>
</tr>
<tr>
<td>A:B</td>
<td>all interactions of A and B</td>
<td>type:education</td>
</tr>
<tr>
<td>A*B</td>
<td>A + B + A:B</td>
<td>type*education</td>
</tr>
<tr>
<td>B %in% A</td>
<td>B nested within A</td>
<td>education %in% type</td>
</tr>
<tr>
<td>A/B</td>
<td>A + B %in% A</td>
<td>type/education</td>
</tr>
<tr>
<td>A^k</td>
<td>effects crossed to order k</td>
<td>(a + b + d)^2</td>
</tr>
</tbody>
</table>

- **data**: A data frame containing the data for the model.
- **subset**:  
  - a logical vector: subset = sex == "F"  
  - a numeric vector of observation indices: subset = 1:100  
  - a negative numeric vector with observations to be omitted: subset = -c(6, 16)
- **weights**: for weighted-least-squares regression
- **na.action**: name of a function to handle missing data; default given by the `na.action` option, initially "na.omit"
- **method, model, x, y, qr, singular.ok**: technical arguments
- **contrasts**: specify list of contrasts for factors; e.g.,  
  `contrasts=list(partner.status=contr.sum, fcategory=contr.poly)`
- **offset**: term added to the right-hand-side of the model with a fixed coefficient of 1.
A generalized linear model consists of three components:

1. A **random component**, specifying the conditional distribution of the response variable, \( y_i \), given the predictors. Traditionally, the random component is an exponential family — the normal (Gaussian), binomial, Poisson, gamma, or inverse-Gaussian.

2. A linear function of the regressors, called the **linear predictor**,

\[ \eta_i = \alpha + \beta_1 x_{i1} + \cdots + \beta_k x_{ik} \]

on which the expected value \( \mu_i \) of \( y_i \) depends.

3. A **link function** \( g(\mu_i) = \eta_i \), which transforms the expectation of the response to the linear predictor. The inverse of the link function is called the **mean function**: \( g^{-1}(\eta_i) = \mu_i \).

In the following table, the logit, probit and complementary log-log links are for binomial or binary data:

<table>
<thead>
<tr>
<th>Link</th>
<th>( \eta_i = g(\mu_i) )</th>
<th>( \mu_i = g^{-1}(\eta_i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity</td>
<td>( \mu_i )</td>
<td>( \eta_i )</td>
</tr>
<tr>
<td>log</td>
<td>( \log_e \mu_i )</td>
<td>( e^{\eta_i} )</td>
</tr>
<tr>
<td>inverse</td>
<td>( \mu_i^{-1} )</td>
<td>( \eta_i^{-1} )</td>
</tr>
<tr>
<td>inverse-square</td>
<td>( \mu_i^{-2} )</td>
<td>( \eta_i^{-1/2} )</td>
</tr>
<tr>
<td>square-root</td>
<td>( \sqrt{\mu_i} )</td>
<td>( \eta_i )</td>
</tr>
<tr>
<td>logit</td>
<td>( \log_e \left( \frac{1 - \mu_i}{1 + \mu_i} \right) ) ( \Phi(\mu_i) )</td>
<td>( \frac{1}{1 + e^{-\eta_i}} ) ( \Phi^{-1}(\eta_i) )</td>
</tr>
<tr>
<td>probit</td>
<td>( \Phi(\mu_i) )</td>
<td>( \Phi^{-1}(\eta_i) )</td>
</tr>
<tr>
<td>complementary log-log</td>
<td>( \log_e [-\log_e(1 - \mu_i)] )</td>
<td>( 1 - \exp[-\exp(\eta_i)] )</td>
</tr>
</tbody>
</table>
Generalized linear models are fit with the `glm` function. Most of the arguments of `glm` are similar to those of `lm`:

- The response variable and regressors are given in a model formula.
- `data`, `subset`, and `na.action` arguments determine the data on which the model is fit.
- The additional `family` argument is used to specify a family-generator function, which may take other arguments, such as a link function.

The following table gives family generators and default links:

| Family            | Default Link | Range of $y_i$ | $V(y_i|\eta_i)$ |
|-------------------|--------------|----------------|-----------------|
| gaussian          | identity     | $(-\infty, +\infty)$ | $\phi$          |
| binomial          | logit        | $0, 1, \ldots, n_i$ | $\mu_i(1 - \mu_i)$ |
| poisson           | log          | $0, 1, 2, \ldots$ | $\mu_i$         |
| Gamma             | inverse      | $(0, \infty)$ | $\phi \mu_i^2$ |
| inverse.gaussian  | $1/\mu^2$    | $(0, \infty)$ | $\phi \mu_i^3$ |

For distributions in the exponential families, the variance is a function of the mean and a dispersion parameter $\phi$ (fixed to 1 for the binomial and Poisson distributions).
The following table shows the links available for each family in R, with the default links as □:

<table>
<thead>
<tr>
<th>family</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>identity</td>
</tr>
<tr>
<td>gaussian</td>
<td>□</td>
</tr>
<tr>
<td>binomial</td>
<td></td>
</tr>
<tr>
<td>poisson</td>
<td>□</td>
</tr>
<tr>
<td>Gamma</td>
<td>□</td>
</tr>
<tr>
<td>inverse.gaussian</td>
<td>□</td>
</tr>
<tr>
<td>quasi</td>
<td>□</td>
</tr>
<tr>
<td>quasibinomial</td>
<td>□</td>
</tr>
<tr>
<td>quasipoisson</td>
<td>□</td>
</tr>
</tbody>
</table>

- The quasi, quasibinomial, and quasipoisson family generators do not correspond to exponential families.
The response for a binomial GLM may be specified in several forms:

- For binary data, the response may be
  - a variable or an S expression that evaluates to 0’s (‘failure’) and 1’s (‘success’).
  - a logical variable or expression (with TRUE representing success, and FALSE failure).
  - a factor (in which case the first category is taken to represent failure and the others success).

- For binomial data, the response may be
  - a two-column matrix, with the first column giving the count of successes and the second the count of failures for each binomial observation.
  - a vector giving the proportion of successes, while the binomial denominators (total counts or numbers of trials) are given by the weights argument to glm.

Poisson generalized linear models are commonly used when the response variable is a count (Poisson regression) and for modeling associations in contingency tables (loglinear models).

- The two applications are formally equivalent. Poisson GLMs are fit in S using the poisson family generator with glm.

- Overdispersed binomial and Poisson models may be fit via the quasibinomial and quasipoisson families.
**Statistical Models in R**

**ANOVA and MANOVA for Repeated-Measures**

- Begin by fitting a multivariate linear model using `lm`, with a matrix of responses on the LHS of the model formula, producing an `mlm` object; e.g.,
  
  ```r
  mod <- lm(cbind(time1, time2, time3) ~ condition, data=Data)
  ```

  for a design with one between-subject factor, `condition`, and one within-subject factor, `time`.

- Create a data frame for the factor(s) in the within-subject design:
  ```r
  idata <- data.frame(time=factor(1:3)).
  ```

- Specify the within-subject design to `Anova` via the `idata` and `idesign` arguments:
  ```r
  Anova(mod, idata=idata, idesign= ~time).
  ```

- For more detail, and to get univariate tests, summarize the object returned by the `Anova` function:
  ```r
  summary(Anova(mod, idata=idata, idesign= ~time)).
  ```

**Data in R**

- Data input

  - From the keyboard
  - From an ascii (plain text) file
  - From the clipboard
  - Importing data (e.g., from SPSS or Excel)
  - From an R package

- The R search path

- Missing data

- Numeric variables, character variables, and factors

- Manipulating matrices, arrays, and lists

- Indexing vectors, matrices, lists and data frames
Programming Basics

Topics

- Function definition
- Control structures:
  - Conditionals: if, ifelse, switch
  - Iteration: for, while, repeat
- Recursion

---

Review of MLE of the Binary Logit Model: Estimation by Newton-Raphson

1. Choose initial estimates of the regression coefficients, such as $b_0 = 0$.
2. At each iteration $t$, update the coefficients:

   $$b_t = b_{t-1} + (X'V_{t-1}X)^{-1}X'(y - p_{t-1})$$

   where
   - $X$ is the model matrix, with $x_i^t$ as its $i$th row;
   - $y$ is the response vector (containing 0's and 1's);
   - $p_{t-1}$ is the vector of fitted response probabilities from the previous iteration, the $i$th entry of which is
     $$p_{i,t-1} = \frac{1}{1 + \exp(-x_i^t b_{t-1})}$$
   - $V_{t-1}$ is a diagonal matrix, with diagonal entries $p_{i,t-1}(1 - p_{i,t-1})$.
3. Step 2 is repeated until $b_t$ is close enough to $b_{t-1}$. The estimated asymptotic covariance matrix of the coefficients is given by
   $$(X'VX)^{-1}.$$
Another approach is to let a general-purpose optimizer do the work of maximizing the log-likelihood,

\[ \log_e L = \sum y_i \log_e p_i + (1 - y_i) \log_e (1 - p_i) \]

Optimizers work by evaluating the gradient (vector of partial derivatives) of the ‘objective function’ (the log-likelihood) at the current estimates of the parameters, iteratively improving the parameter estimates using the information in the gradient; iteration ceases when the gradient is sufficiently close to zero.

For the logistic-regression model, the gradient of the log-likelihood is

\[ \frac{\partial \log_e L}{\partial b} = \sum (y_i - p_i)x_i \]

The covariance matrix of the coefficients is the inverse of the matrix of second derivatives. The matrix of second derivatives, called the Hessian, is

\[ \frac{\partial \log_e L}{\partial b \partial b'} = X'VX \]

The optim function in R, however, calculates the Hessian numerically (rather than using an analytic formula).
Debugging and Profiling R Code

- Locating an error: `traceback()`
- Setting a breakpoint and examining the local environment of an executing function: `browser()`
- A simple interactive debugger: `debug()`
- Some other facilities: `debug` package, `debugger()` post-mortem debugger
- Measuring time and memory usage with `system.time` and `Rprof`

Object-Oriented Programming

The S3 Object System

- S3 versus S4 objects
- How the S3 object system works
- Method dispatch, for `object` of class "class": `generic(object)`
  \[\Rightarrow\] `generic.class(object)` \[\Rightarrow\] `generic.default(object)`
  - For example, summarizing an object `mod` of class "lm": `summary(mod)`
    \[\Rightarrow\] `summary.lm(mod)`
- Objects can have more than one class, in which case the first applicable method is used.
  - For example, objects produced by `glm()` are of class `c("glm", "lm")` and therefore can `inherit` methods from class "lm".
- Generic functions: `generic <- function(object, other-arguments, ...) UseMethod("generic")`
  - For example, `summary <- function(object, ...) UseMethod("summary")`
R Graphics

Topics

- Traditional S graphics
  - points
  - lines
  - text
  - axes
  - frames
  - arrows
  - polygons
  - legends
  - curves
  - use of colour

- Trellis graphics (via the lattice package)

Example: Diagram of the Standard-Normal Density Function

The Standard Normal Density Function $\phi(z)$

\[
\phi(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}
\]

\[
\int_{-\infty}^{1.96} \phi(z) dz = 0.025
\]
Example: Explanation of Nearest-Neighbor Kernel Regression

(a) Observations Within the Window |
(b) Tricube Weights

(c) Weighted Average (Kernel Estimates) |
(d) Complete Kernel Estimate

---

Example: Trellis Display

Catholic Schools

---

John Fox  (McMaster University)  Introduction to R  ICPSR 2010  30 / 34
Building R Packages

- Primary reference for creating R Packages: *Writing R Extensions* (distributed with R).
- Create a *source package* containing “meta-information” describing the package; R code; data; documentation; and possibly other components.
- The source package is then checked and built into a universal .tar.gz format that can be distributed and installed under any operating system on which R runs, or built into a *binary package* for a particular system, such as Windows or Mac OS X.
- Under Windows, the ability to build packages requires installing software in addition to R: a working LaTeX, Perl, Unix-like command-line tools, and a C++ compiler, all of which are readily available. See the “Windows toolset” appendix in the *R Installation and Administration* manual and [http://www.murdoch-sutherland.com/Rtools/](http://www.murdoch-sutherland.com/Rtools/).

To check a package, assuming that the R bin directory is on the Windows path, and that the package subdirectory is in the current directory:

R CMD check *package-name*

To build the package, producing a tar.gz file:

R CMD build --force *package-name*

To build a Windows binary package, producing a .zip file:

R CMD build --binary *package-name*

To install directly:

R CMD INSTALL *package-name*
Building R Packages

Package Structure

- An R source package consists of a directory containing:
  - A DESCRIPTION file with meta information such as the package name, version, and author, and other packages on which the package depends.
  - An optional NAMESPACE file (for a package with a namespace), enumerating, e.g., the objects that are “exported” from the package.
  - An R subdirectory containing .R files with R code for creating objects such as functions.
  - A man (“manual”) subdirectory that includes .Rd documentation files (using LaTeX-like markup). All public objects in the package should be documented.
  - A data subdirectory containing data objects, such as text files that can be read as R data frames. Thus, a file named Duncan.txt would produce a data frame named Duncan.
  - Possibly an inst (“install”) subdirectory containing arbitrary files and subdirectories to be installed in the package.
  - Possibly other subdirectories (e.g., for compiled C code).

The function package.skeleton creates the “skeleton” of a source package for objects that are currently in the R workspace.

Example: The matrixDemos package.