The R statistical computing environment

Luke Tierney

Department of Statistics & Actuarial Science
University of Iowa

June 17, 2011

Introduction

- R is a language for data analysis and graphics.
- Originally developed by Ross Ihaka and Robert Gentleman at University of Auckland, New Zealand.
- R is based on the S language developed by John Chambers and others at Bell Labs.
  - Strong emphasis on flexibility and ability to handle non-standard problems.
- R is widely used in the field of statistics and beyond, especially in university environments.
- R has become the primary framework for developing and making available new statistical methodology.
- Many extension packages are available through CRAN or similar repositories.

History and Development Model

- R is an Open Source project.
- Originally developed by Robert Gentleman and Ross Ihaka in the early 1990's for a Macintosh computer lab at U. of Auckland, NZ.
- Developed by the R-core group since mid 1997,

Douglas Bates	John Chambers	Peter Dalgaard
Robert Gentleman	Seth Falcon	Kurt Hornik
Stefano Iacus	Ross Ihaka	Friedrich Leisch
Thomas Lumley	Martin Maechler	Duncan Murdoch
Paul Murrell	Martyn Plummer	Brian Ripley
Deepayan Sarkar	Duncan Temple Lang	Luke Tierney
Simon Urbanek		

- Strong community support through
  - contributed extension packages
  - mailing lists and blogs
  - contributed documentation and task views

Basic Design of R

- Basic philosophy: Good statistical analysis involves
  - exploring the data
  - allowing the data to guide the choice of analysis tools
  - adapting tools as needed for the appropriate analysis
- R is an interactive system
  - contrasts to batch-oriented systems (e.g. SAS)
- R is a high level language
  - contrasts to pure GUI systems (e.g. JMP)
  - allows analysis to be documented, repeated
  - allows new methods to be programmed

Extending the R System

- Writing simple R functions is a natural part of working in R.
- Collections of functions that implement a particular analysis are often best organized into a *package*.
- The R package system provides a framework for developing, documenting, and testing extension code.
- Packages can include R code as well as foreign code (C, FORTRAN).
- Many R packages are made available though the CRAN repository `http://cran.r-project.org` (recently reached 3,000 mark)

Some Basics

- R uses a command line interface: *read-evaluate-print* loop
  - you type an expression
  - R reads the expression, evaluates it, and prints the result
- Some simple examples:

```
> 2 + 3
[1] 5
> exp(-2)
[1] 0.1353353
> log(100, base = 10)
[1] 2
```

Variables and Vectors

- A variable `x` containing some uniform random numbers:

```
> x <- runif(4)
> x
[1] 0.1137034 0.6222994 0.6092747 0.6233794
```

- Some vectorized operations:

```
> x + 1
[1] 1.113703 1.622299 1.609275 1.623379
> log(x)
[1] -2.1741619 -0.4743339 -0.4954860 -0.4725999
```

Numerical Summaries

- mean and standard deviation:

```
> mean(x)
[1] 0.4921642
> sd(x)
[1] 0.2523886
```

- median and inter-quartile range:

```
> median(x)
[1] 0.6157871
> IQR(x)
[1] 0.1371875
```

- sorting and ranking:

```
> sort(x)
[1] 0.1137034 0.6092747 0.6222994 0.6233794
> rank(x)
[1] 1 3 2 4
```

Probability Distributions

- Some standard distributions:

Distribution	Density	CDF	Quantile	Generate
Uniform	dunif	punif	qunif	runif
Normal	dnorm	pnorm	qnorm	rnorm
t	dt	pt	qt	rt
F	df	pf	qf	rf
Gamma	dgamma	pgamma	qgamma	rgamma
Poisson	dpois	ppois	qpois	rpois
...				

- Others included in base R: Exponential, Beta, Cauchy, Binomial, ...
- More available in extension packages.

Defining Functions

- The Pareto distribution is not covered by base R (though it is by several contributed packages)
- The CDF of the Pareto distribution is

$$F(x) = \begin{cases} 1 - x\left(\frac{x_m}{x}\right)^{\alpha} & \text{for } x \geq x_m \\ 0 & \text{otherwise} \end{cases}$$

  for parameters $x_m, \alpha > 0$.
- A vectorized R function ppareto to compute this CDF is

```
ppareto <- function(x, xmin, alpha = 1) {
    stopifnot(all(xmin > 0) && all(alpha > 0))
    1 - pmin(xmin / x, 1) ^ alpha
}
```

Linear Algebra

- Decompositions and linear equations
  - QR factorization: qr
  - Cholesky factorization: chol
  - Singular value decomposition: svd
  - General linear systems: solve
  - Triangular systems: forwardsolve, backsolve
- Eigen values: eigen
- Implementation uses LAPACK, LINPACK
- Underlying BLAS can easily be replaced by optimized version
- The `Matrix` package provides richer facilities, including extensive sparse matrix support

Graphics

- R contains a rich set of graphical facilities
- Several graphics frameworks are available
- Two frameworks provides in the basic R distribution are
  - Base graphics
  - Lattice graphics
- Two widely used frameworks provides as add-on packages are
  - ggplot2 graphics
  - Interactive RGL graphics
- A number of others are available or in development

Base Graphics

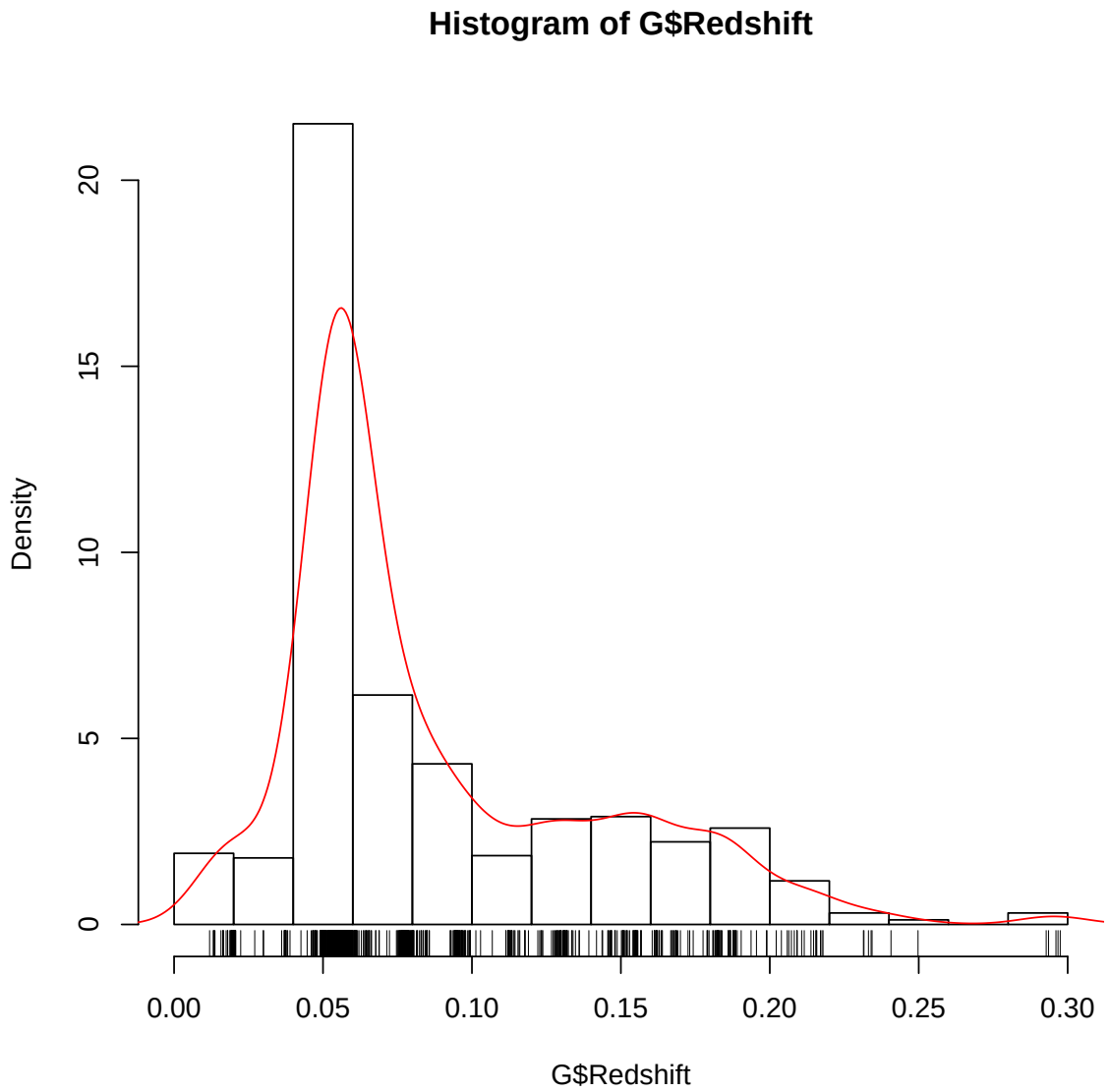
- Base graphics provides a number of standard graphs, such as
  - dot plots
  - box plots
  - histograms
  - scatter plots
  - scatter plot matrices
  - perspective plots
- Base graphics is easy to use for simple tasks
- Base graphics supports composing and augmenting plots

Base Graphics Example

- Adapted from Alastair Sanderson's tutorial pages at `http://www.sr.bham.ac.uk/~ajrs/R/r-tutorials.html`
- Data on galaxy cluster Abell 85 from NASA Extragalactic Database
- Create a plot of the distribution of redshift values for galaxy objects most 0.3:

```
# read in the data
A <- read.table("a85_extended_NEDsearch.txt", sep="|",
                skip=20, header=TRUE)
# simplify some variable names
colnames(A)[c(2, 3, 4, 5)] <- c("name", "ra", "dec", "type")
# create the data subset
G <- subset(A, type=="G" & !is.na(Redshift) & Redshift < 0.3)
# plot a histogram and kernel density estimate
hist(G$Redshift, prob = TRUE)
lines(density(G$Redshift), col = "red")
# add raw data values as a "rug plot"
rug(G$Redshift)
```

Base Graphics Example



Lattice Graphics

- Lattice graphics are used for creating structured sets of related graphs for understanding multivariate data.
- Lattice was developed by Deepayan Sarkar based on the Cleveland's Trellis system for S
- Some advantages of lattice graphics:
  - Better default choices for many graphical parameters and layout
  - Simpler mechanisms for adding annotations
  - Easier to show multiple data sets in single plots
- Lattice plots usually display one or two variables given values of additional variables using
  - grouping, using separate colors or symbols
  - conditioning, using separate plots on identical scales

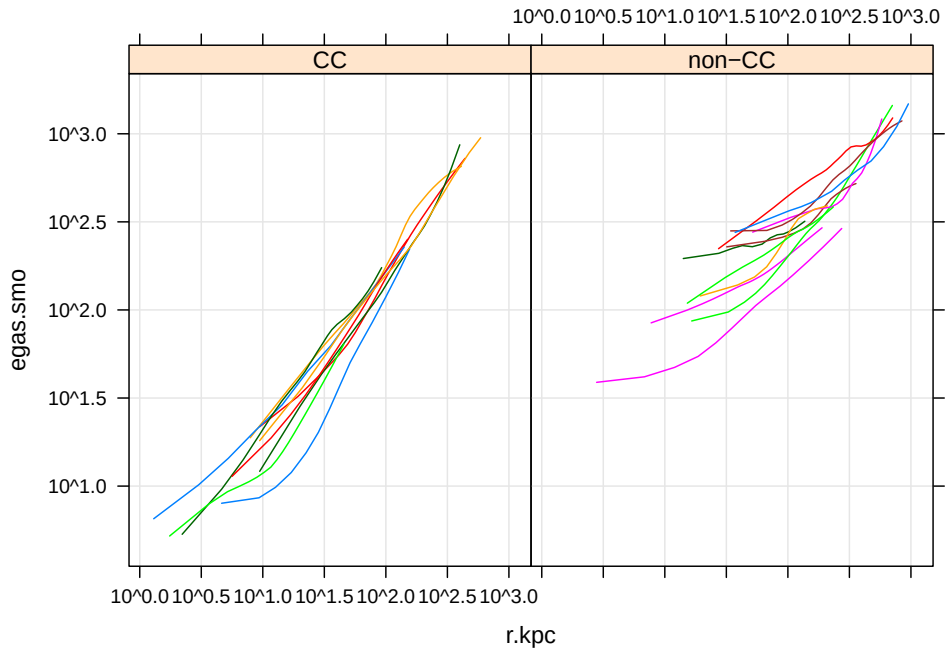
Lattice Graphics Example

- Adapted from Alastair Sanderson's tutorial pages at `http://www.sr.bham.ac.uk/~ajrs/R/r-tutorials.html`
- Data are smoothed gas entropy measurements at a series of radii of 20 galaxies
- Variables in the data set are
  - egas.smo: smoothed gas entropy
  - r.kpc: radius, in kiloparsecs
  - cname: cluster name
  - cctype: cool core or non-cool core

Lattice Graphics Example

Comparing cool core and non-cool core profiles:

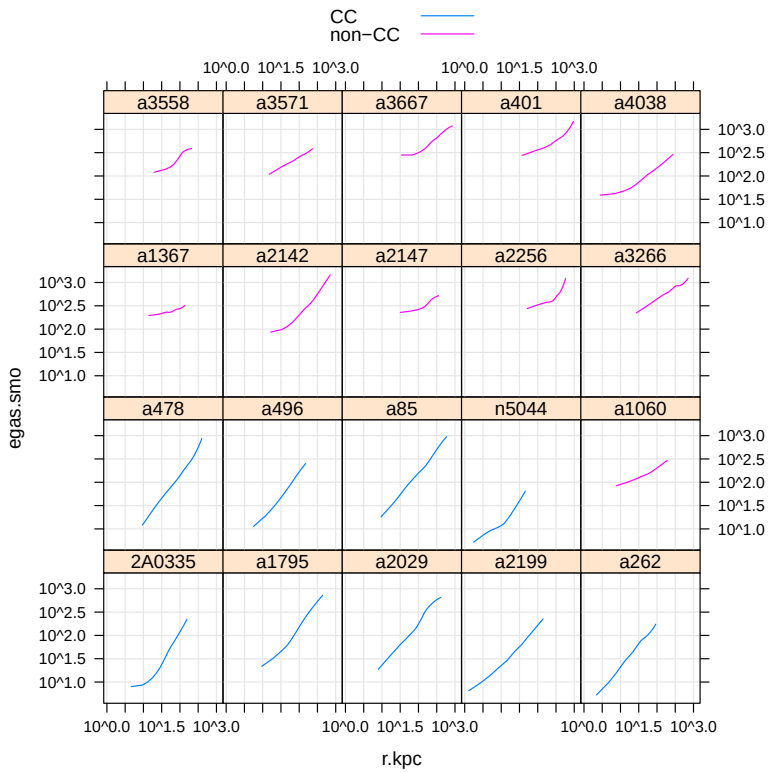
```
xyplot(egas.smo ~ r.kpc | cctype,  groups=cname,
       scales=list(log=TRUE),
       type=c("g", "l"), aspect = "xy", data=entropy)
```



Lattice Graphics Example

Examining individual profiles:

```
xyplot(egas.smo ~ r.kpc | reorder(cname, as.numeric(cctype)),
       groups=cctype, type=c("g", "l"), data=entropy,
       scales=list(log=TRUE),
       auto.key=list(points=FALSE, lines=TRUE))
```



Statistical Models

- The basic R distribution supports fitting a wide range of statistical models, including
  - linear regression models
  - nonlinear regression models
  - generalized linear models
  - mixed models
  - survival models
  - time series
  - spatial models
- Tools for general optimization and maximum likelihood fitting are also provides.
- Contributed packages add support for many more models and methods.

Common Modeling Function Features

- Most modeling functions support a *formula language* for specifying a model

```
y ~ a + b
```

- Data is usually taken from a data frame specified as a data argument.
- Functions usually return a model object that can be used to
  - extract coefficients and standard error estimates
  - compute residuals or fitted values
  - predict responses at new explanatory variable values
  - obtain summary information for the fit
- The most basic modeling function is lm for fitting linear models.

Linear Model Example

A simple (not very sensible) model can be fit to the gas entropy data:

```
> summary(lm(log(egas.smo) ~ cctype + cctype * log(r.kpc), data = entropy))

Call:
lm(formula = log(egas.smo) ~ cctype + cctype * log(r.kpc), data = entropy)

Residuals:
     Min       1Q   Median       3Q      Max
-0.87240 -0.15384  0.02201  0.16733  1.05008

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)              0.57021    0.06988    8.16 5.12e-15 ***
cctypenon-CC             2.01888    0.11629   17.36  < 2e-16 ***
log(r.kpc)               0.97672    0.01700   57.47  < 2e-16 ***
cctypenon-CC:log(r.kpc) -0.32162    0.02520  -12.76  < 2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.2767 on 375 degrees of freedom
Multiple R-squared: 0.9487,        Adjusted R-squared: 0.9482
F-statistic:  2310 on 3 and 375 DF,  p-value: < 2.2e-16
```

Data Types

- R supports a number of different vector data types:
  - logical vectors
  - integer vectors
  - real (double precision) vectors
  - character vectors
  - generic vectors
- There are no scalars, only vectors of length 1
- Arrays are vector with a dim attribute
- Generic vectors are building blocks for more general data types
  - Of these, *data frames* are the most important.
- Non-vector data types include
  - functions
  - environments

Some Unusual Language Features

- Vectors are immutable.
  - Conceptually,

    `x[i] <- y`

    assigns a modified copy of `x` to `x`.
  - R works to avoid making copies that are not necessary.
- All atomic (i.e. non-generic) vector types support missing values.
- Function arguments are evaluated only if needed (lazy evaluation)
  - This means even control flow constructs can be viewed as ordinary functions.
- Call argument expressions are available to functions
  - Useful for creating default labels for plots
  - Combined with lazy evaluation, this means functions can implement their own evaluation rules (non-standard evaluation).

Future Directions: Performance

- Many computations make use to the BLAS routines
  - It is already possible to substitute high performance BLAS libraries.
  - More work may be done on supporting alternate BLAS libraries.
- OpenMP can be used to parallelize many R operations on multi-core computers, including
  - row or column-wise operations on matrices
  - vectorized math functions
- Should GPU computing be integrated directly into R?
  - Several packages use GPU resources via CUDA
  - Need to explore whether direct integration into core R is useful
- A byte code compiler has recently been added to R.
  - Currently this helps most with scalar-intensive computations
  - Future work will look into
    - helping with parallelizing vector operations
    - possible native code generation
    - improved function call performance.

Future Directions: Large Data Sets

- Most R functions work on data held in computer memory
- A current limitation of R is that integers are represented as 32-bit quantities
- This limits the number of elements in an array R can address to $2^{31} - 1 = 2,147,483,647$ elements
- Work over the next year or so will investigate how to remove this limit
- With storage capacity likely to remain larger than available memory more methods for handling data sets larger than available memory will continue to be needed.
- Various approaches can be programmed in R, and several packages are available for operating on very large data stored on disk or other storage media

Getting Started with R

- R home page: `http://www.r-project.com`
- Obtaining R:
  - R is available from `http://cran.r-project.com`
    - source code
    - binary installations for Windows, Mac OS X
  - Many Linux distributions make R available through their package management systems.
  - There are also several commercial options.
- Getting help:
  - The help command accesses the manual included in R.
  - Additional manuals are available at `http://cran.at.r-project.org`
  - Links to user-contributed documentation, tutorials, and books about R are also available at the CRAN web site.
  - There are several active mailing lists and blogs available.
  - *Task Views* available at CRAN provide a useful way of learning about relevant contributed packages (there is no Astronomy task view yet ...)