Journal of the Korean Statistical Society (2007), **36***: 1, pp 31–55*

R: AN OVERVIEW AND SOME CURRENT DIRECTIONS[†]

LUKE TIERNEY[1]

ABSTRACT

R is an open source language for statistical computing and graphics based on the ACM software award-winning S language. R is widely used for data analysis and has become a major vehicle for making available new statistical methodology. This paper presents an overview of the design philosophy and the development model for R, reviews the basic capabilities of the system, and outlines some current projects that will influence future developments of R.

AMS 2000 subject classifications. Primary 68N15; Secondary 62-07.
Keywords. Computer language, graphics, isosurfaces, parallel computation, R, statistical computing.

1. INTRODUCTION

R is a language for statistical computing and graphics. R, along with S-plus, is a member of the S language family originally developed by John Chambers and colleagues at Bell Laboratories (Becker and Chambers, 1984; Becker *et al.*, 1988; Chambers, 1998). In 1998 the Association for Computing Machinery presented its prestigious Software System Award to John Chambers for his development of the S system. In recent years the S language family, and R in particular, have become the de facto standard for computing in statistical research. Many books describing R and S are now available; two examples are Dalgaard (2004) and Venables and Ripley (2002).

R can viewed as a different implementation or as a dialect of S the language. While there are some important differences between R and S-plus, some of which are discussed in Section 6.1, much code written for S-plus runs unaltered under R. In recent years R has become a major vehicle for making available new statistical methodology. Articles proposing new methods that also make available

Received October 2006; accepted November 2006.

[†]This research supported in part by National Science Foundation Grant DMS-0604593.

[1]Department of Statistics and Actuarial Science, University of Iowa, IA 52242, U.S.A. (e-mail: luke@stat.uiowa.edu)

software implementing these methods do so using R more than any other software framework.

R is an Open Source project. R was originally developed by Robert Gentleman and Ross Ihaka in the early 1990's for a Macintosh computer lab at the University of Auckland in New Zealand (Ihaka and Gentleman, 1996). In 1995 Ross and Robert decided to release R as open source software and to invite other researchers to join in R's development. In 1997 a core group of around 10 developers was formed, and John Chambers joined this group in 2000. The current members of the R Core team are listed in Table 1.1.

TABLE 1.1 *Current R Core members*

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

The basic design of R is motivated by the philosophy that good statistical analysis involves exploring the data, allowing the data to guide the choice of analysis tools, and adapting tools as needed for the appropriate analysis rather than adapting the analysis to easily available tools. R is an interactive system, in contrast to batch-oriented systems (*e.g.* SAS). R is a high level programming language, in contrast to pure GUI systems (*e.g.* JMP). This allows analyses to be documented and repeated; it also allows new methods to be programmed.

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 through the Comprehensive R Archive Network (CRAN) repository. A recent count gave about 840 separate packages available through CRAN. Other repositories include the Bioconductor repository.

2. BASIC USAGE

R uses a command line interface, a *read-evaluate-print* loop: the user types 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
```

A variable x containing a random sample of four numbers drawn from a standard uniform distribution is created by

```
> x <- runif(4)
> x

[1] 0.1137034 0.6222994 0.6092747 0.6233794
```

Arithmetic operations in R are *vectorized.* 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
```

A range of numerical summaries are available, including 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
```

and sorting and ranking:

```
> sort(x)

[1] 0.1137034 0.6092747 0.6222994 0.6233794

> rank(x)

[1] 1 3 2 4
```

3. Graphics

Graphics are an important component of modern data analysis. R encourages the use of graphics by making it easy to construct simple graphs and by providing a rich collection of different graphics types. Higher level graphics provided by R are built on a low level abstraction layer that is common across all graphics devices, and many contributed packages have used this foundation to further enrich the selection of possible graphical displays. A recent book by Paul Murrell (2005) provides a detailed introduction to graphics in R.

3.1. Simple graphics

Simple graphics include histograms and scatter plots. As an example, the data set `geyser` in package `MASS` is a *data frame* containing two variables representing measurements on successive eruptions of the Old Faithful geyser. The variables are the waiting time between eruptions, `waiting`, and the duration of the subsequent eruption, `duration`. A histogram of the durations with a superimposed kernel density estimate is constructed by

```
> library(MASS)
> hist(geyser$duration, prob = TRUE)
> lines(density(geyser$duration), col = "red")
```

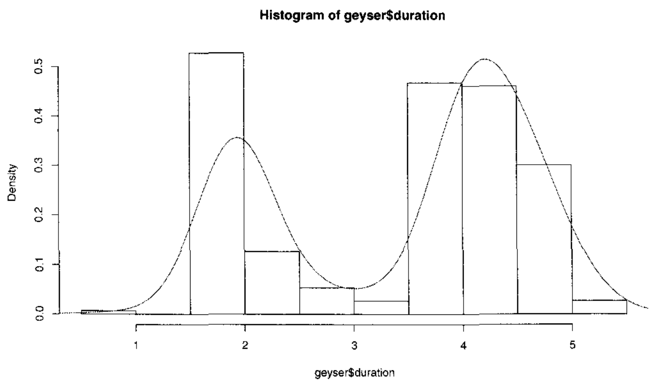


FIGURE 3.1 *Histogram and density estimate for the eruption durations of Old Faithful.*

The result is shown in Figure 3.1. The distribution has a clear bimodal structure.

The waiting times also have a bimodal distribution. An interesting question for park management is whether the duration of the most recent eruption can be used to predict the waiting time until the next eruption. We can compute and plot the previous duration against waiting times with

```
> geyser2 <- geyser[-1, ]
> geyser2$pduration <- geyser$duration[-299]
> plot(waiting ~ pduration, data = geyser2)
```

The negative indices delete the corresponding entries. The plot is shown in Figure 3.2.

3.2. More complex graphics

The display of relationships among three or more variables presents unique challenges. One of the standard displays available in R is the pairwise scatter plot or scatter plot matrix. Figure 3.3 shows such a plot for Edgar Anderson's iris data consisting of four measurements on 50 flowers from each of three iris species:

```
> pairs(iris[1:4], main = "Edgar Anderson's Iris Data", pch = 21,
+       bg = c("red", "green3", "blue")[unclass(iris$Species)])
```

Color is used to identify the different iris species.

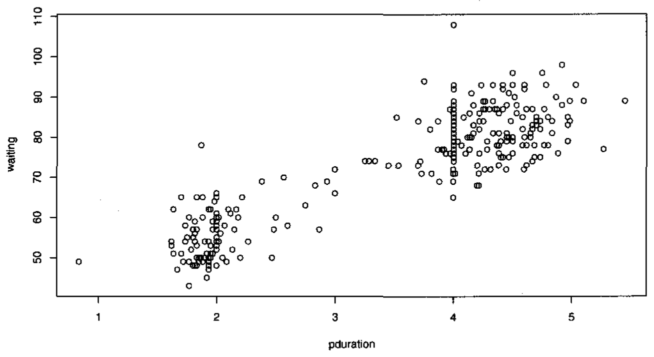


FIGURE 3.2 *Scatter plot of waiting time against previous eruption duration for the Old Faithful data.*

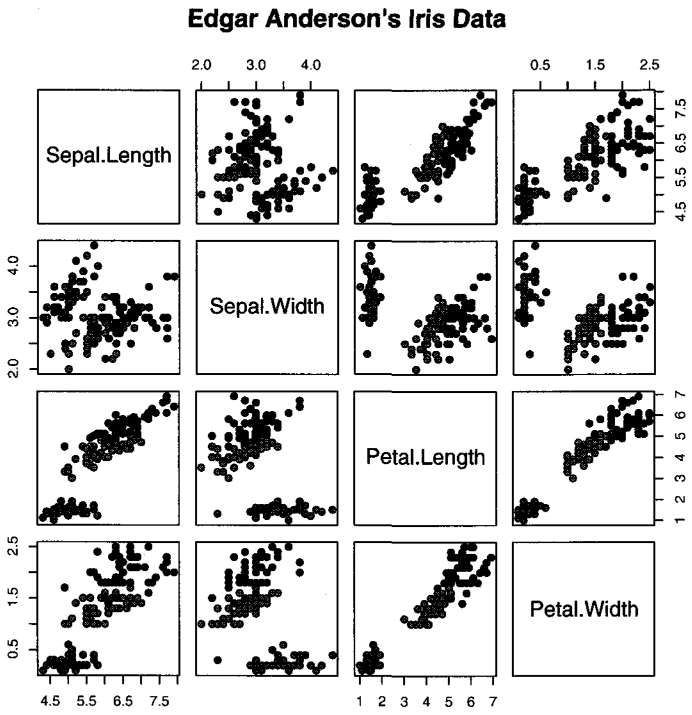


FIGURE 3.3 *Scatter plot matrix of Anderson's iris data.*

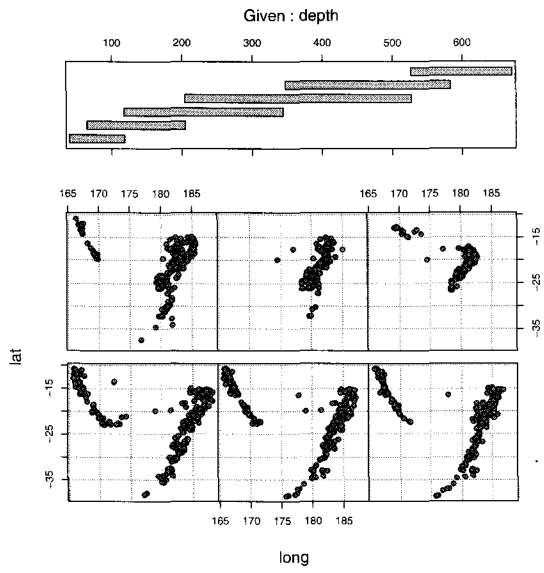


FIGURE 3.4 *Conditioning plot for locations of 1000 seismic events near Fiji.*

Three dimensional data can be represented using a conditioning plot, a collection of related two dimensional plots each corresponding to a restricted range of values for the third variable. Figure 3.4 shows a conditioning plot, or `coplot`, for the locations of 1000 seismic events in an area near Fiji. Longitude and latitude are plotted for different values of depth.

```
> coplot(lat ~ long | depth, data = quakes, pch = 21,
+    bg = "green3")
```

Each scatter plot panel corresponds to one of the depth ranges shown in the top strip; the lower left panel corresponds to the shallowest depth level and the upper right panel to the deepest. The coplot is an example of a *trellis display*; these are discussed further in Section 3.3.

Displays that can be useful for visualizing functions of two variables include perspective plots, image plots, and contour plots. Figure 3.5 shows a perspective plot and an image plot overlayed with a contour plot of the surface of a volcano in Auckland, New Zealand. The plots are constructed with

```
> z <- 2 * volcano
> x <- 10 * (1:nrow(z))
> y <- 10 * (1:ncol(z))
```

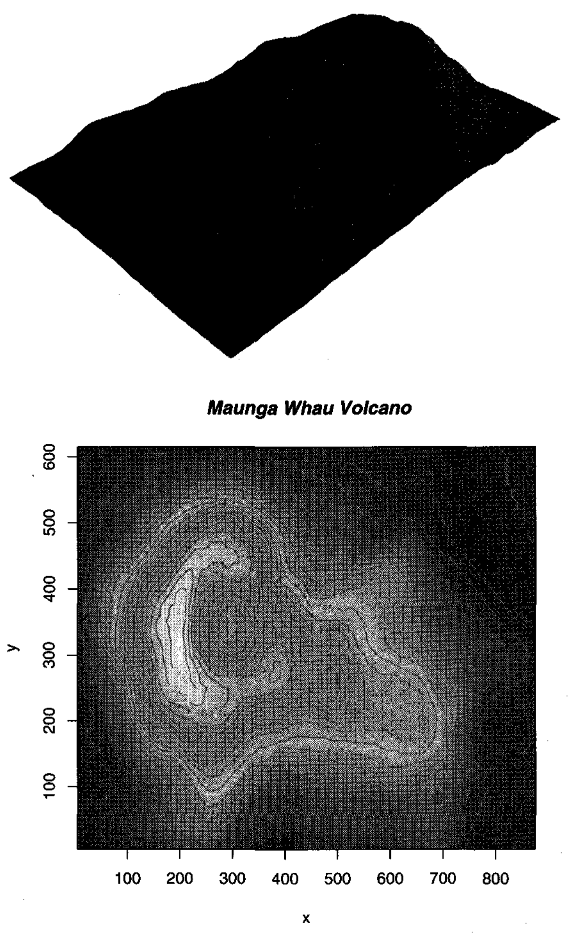


FIGURE 3.5 *Perspective, image and contour plots of the Maunga Whau volcano in Auckland, New Zealand.*

```
> persp(x, y, z, theta = 135, phi = 30, col = "green3",
+     scale = FALSE, ltheta = -120, shade = 0.75, border = NA,
+     box = FALSE)
```

and

```
> image(x, y, volcano, col = terrain.colors(100), axes = FALSE)
> contour(x, y, volcano, levels = seq(90, 200, by = 5),
+     add = TRUE, col = "peru")
> axis(1, at = seq(100, 800, by = 100))
> axis(2, at = seq(100, 600, by = 100))
> box()
> title(main = "Maunga Whau Volcano", font.main = 4)
```

The R Graph Gallery and `demo(graphics)` give more examples.

3.3. Lattice graphics

The `lattice` package by Deepayan Sarkar implements trellis displays (Becker *et al.*, 1996). Trellis displays are collections of related plots designed to visualize the relations among several variables. The `xyplot` function implements trellis scatter plots, which can consist of a single plot or a collection of plots conditioned on values of additional variables.

Trellis displays are particularly useful in hierarchical modeling. An example is provided by growth measurements recorded for a sample of 26 boys from Oxford, England. The measurements consist of height in centimeters and a standardized age value. The data are arranged in a data frame with a third column, `Subject`, recording the identity of the individual measured at a particular occasion. Figure 3.6 shows a basic conditioning plot constructed by

```
> library(nlme)
> library(lattice)
> xyplot(height ~ age | Subject, data = Oxboys)
```

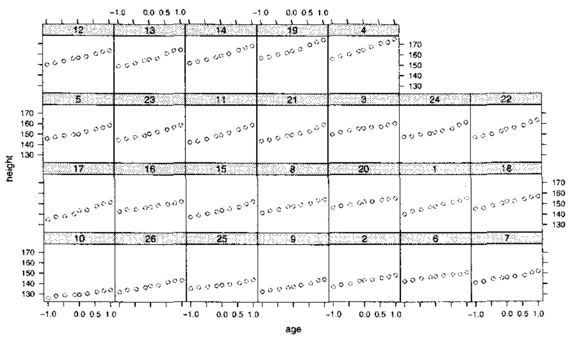


FIGURE 3.6 *Lattice plot of height against standardized age within subject for Oxford boys growth data.*

The plot shows near linear behavior within subject, but both slope and intercept vary between subjects. The covariation of slope and intercept is hard to judge from from Figure 3.6. A simple change to the plotting command changes the aspect ratio of the plots and adds regression lines within each plot:

```
> xyplot(height ~ age | Subject, data = Oxboys, aspect = "xy",
+      type = c("p", "r"), col.line = "black")
```

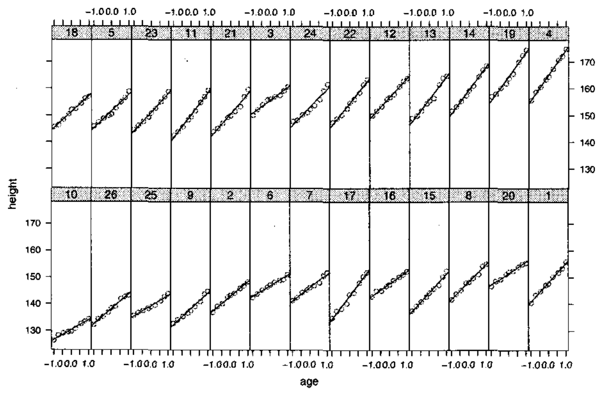


FIGURE 3.7 *Lattice plot with a different aspect ratio and superimposed regression lines.*

The "xy" aspect specification requests a plot in which the slopes are closer to ±45 degrees since research has shown that this helps to compare slopes accurately (Cleveland, 1993). The plot suggests that higher intercepts are associated with higher slopes. This is illustrated further in Section 4.2.

4. Statistical Models in R

The standard R distribution supports fitting a wide range on statistical models to data. Many more are supported through contributed packages. Some of the models that are supported in the standard distribution are linear, nonlinear, and linear mixed effects models with Gaussian errors, generalized linear and generalized additive models, and survival models. Most modeling functions use a *model formula* for specifying the model to be fit. Examples of model formulas are

- Linear model: `height ~ weight + age`
- Nonlinear mean function: `Weight ~ b0 + b1 * 2 ^ (-Days/th)`
- Survival model: `Surv(time, status) ~ dose + diet`

Similar formulas are also used in specifying graphical displays, in particular for lattice graphics.

4.1. Linear models

Linear models are fit using the function *lm*. For the Old Faithful geyser data introduced in Section 3.1 a linear model relating waiting time to the duration of the previous eruption is fit using the expression

```
> geyser.fit <- lm(waiting ~ pduration, data = geyser2)
```

The function *summary* produces a standard summary of the fit:

```
> summary(geyser.fit)

Call:
lm(formula = waiting ~ pduration, data = geyser2)

Residuals:
     Min       1Q   Median       3Q      Max
-14.6940  -4.4954  -0.0966   3.9544  29.9544

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  34.9452     1.1807   29.60   <2e-16 ***
pduration    10.7751     0.3235   33.31   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 6.392 on 296 degrees of freedom
Multiple R-Squared: 0.7894, Adjusted R-squared: 0.7887
F-statistic:  1110 on 1 and 296 DF,  p-value: < 2.2e-16
```

The plot function has a method for *lm* objects that can produce various diagnostic plots; a plot of residuals against fitted values, shown in Figure 4.1, is produced by

```
> plot(geyser.fit, which = 1)
```

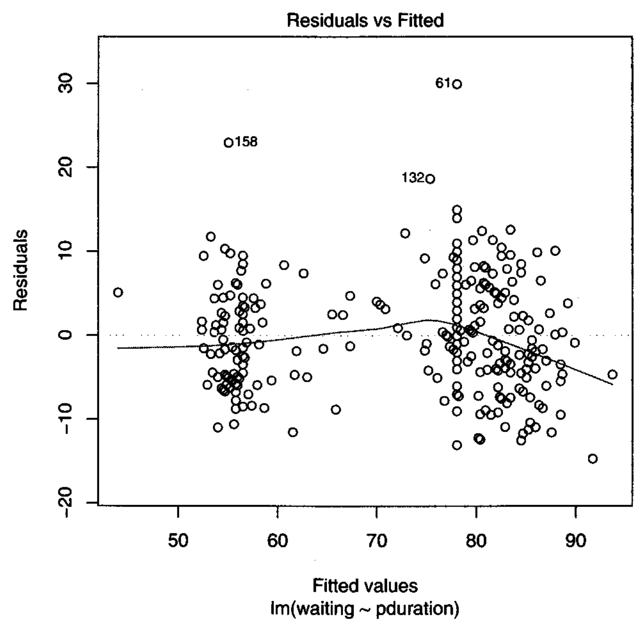


FIGURE 4.1 *Diagnostic plot of residuals against fitted values for predicting waiting time from previous eruption duration for the Old Faithful data.*

4.2. Linear mixed models

Linear mixed effects models can be fit using the function `lme` from package `nlme`. A model with random slopes and intercepts can be fit to the Oxford boys growth data of Section 3.3 using

```
> lme(height ~ age, data = Oxboys, random = ~age | Subject)

Linear mixed-effects model fit by REML
  Data: Oxboys
  Log-restricted-likelihood: -362.0455
  Fixed: height ~ age
(Intercept)         age
 149.371753    6.525469


Random effects:
 Formula: ~age | Subject
 Structure: General positive-definite, Log-Cholesky parametriza-
```

```
            tion
            StdDev   Corr
(Intercept) 8.081077 (Intr)
age         1.680717 0.641
Residual    0.659889

Number of Observations: 234
Number of Groups: 26
```

As suggested by the graphical analyses the fit estimates a strong positive correlation between the slopes and the intercepts.

5. SIMULATION

Simulations are useful for exploring and understanding a wide range of problems and for approximating otherwise intractable integrals and probabilities. R provides facilities for simulating from many univariate distributions and some multivariate distributions, and for sampling with and without replacement from a specified population. R includes a range of uniform generators and allows the generator and its seed to be specified. This is useful for ensuring reproducible results and for checking that a result is not unduly affected by the choice of a underlying random number generator. It is also possible to install a user-provided generator; this is useful in parallel computing contexts described in Section 8.1.

A simple example often used in the classroom is to use simulation to illustrate the central limit theorem. The following code generates 1000 samples of 32 exponential random variables and computes the sample mean for the first 4 and for all 32 observations in each sample. Histograms and density estimates are then plotted and shown in Figure 5.1.

```
> rmat <- matrix(rexp(1000 * 32), nrow = 32)
> mns <- cbind(colMeans(rmat[1:4, ]), colMeans(rmat))
> hist(mns[, 1], prob = TRUE, main = "Samples of Size 4")
> lines(density(mns[, 1]), col = "red")
> hist(mns[, 2], prob = TRUE, main = "Samples of Size 32")
> lines(density(mns[, 2]), col = "red")
```

The plot for samples of size 32 is clearly closer to normal in shape than the plot for samples of size 4.

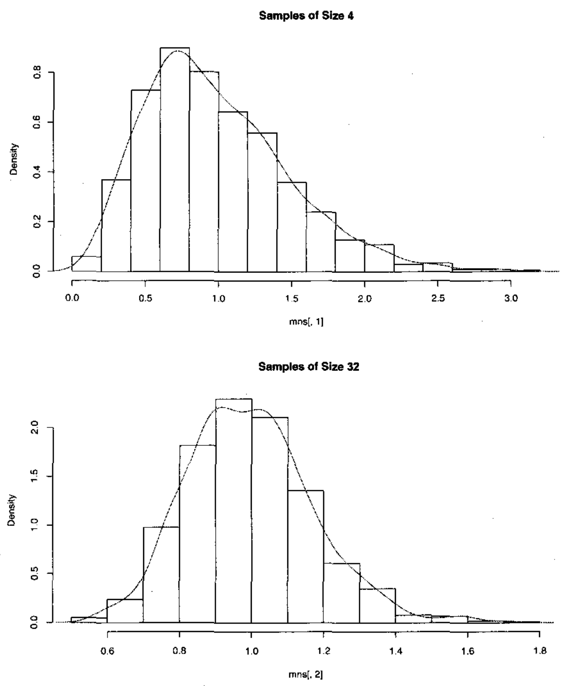


FIGURE 5.1 *Histograms and density estimates for sample means for samples of size 4 and 32 from an exponential distribution.*

R is often used for simulation-based inference, including simulation-based exact tests, bootstrapping, and Bayesian inference via Markov chain Monte Carlo. The **boot** package is the most used framework for bootstrapping. Several packages supporting generic MCMC are available, and a number of packages use MCMC internally for specific computations. There are also several interfaces to BUGS (Thomas *et al.*, 2006) available. MCMC samplers for more complex models are often programmed directly in R. The **coda** and **boa** packages are useful for the analysis of MCMC output.

6. Some Aspects of Software Development in R

R has become a major framework for implementing new statistical methodology. A number of features of R support the development of statistical software. R is designed as a high level language, but also provides means of interfacing to code written in low level languages such as C and FORTRAN. This is useful for allowing existing code written in these languages to be interfaced to R and to allow code to be written in these languages to improve performance. R also provides a very effective package mechanism for managing and distributing a collection of code and related documentation. A major goal of the package mechanism is to provide tools to help package authors test and consistently document their code. The `Sweave` system (Leisch, 2002, 2003) provides an excellent means for integrating the results of R computation into documentation and is used as the basis for more extensive usage documents called *vignettes* that can be included in packages.

R also provides a profiling mechanism to help identify execution hot spots that can be studied more carefully to improve performance. Thomas Lumley is currently working on extensions to the profiling mechanism to support profiling of memory use.

Recently R has been used as a statistical engine embedded in other applications such as data bases or web servers. Support to make this easier to do and work more robustly continues to be added to R.

R has a number of high level language features that contribute to its effectiveness for developing statistical software. Lazy evaluation of arguments is a feature of the S language family that is somewhat unusual but can be useful; in particular it means that new control constructs can be introduced without the need for a macro system. Lexical scope, which separates R from other members of the S family, is a standard feature of high level functional and nearly functional languages and can be very useful for succinctly expressing a number of computations; a brief example is given in Section 6.1. Name space management, discussed in Section 6.2, is valuable for ensuring that separately developed packages can be used together without interference. R also provides a sophisticated mechanism for handling error conditions and support for object-oriented programming.

6.1. Lexical scope

Function bodies contain two kinds of variables: variables that are *bound* to the function parameters and *free* variables. How free variables are interpreted is determined by a language's *scoping rule.* R uses lexical scope (Gentleman and Ihaka, 2000). This means that free variables in a function are resolved by looking in the environment in which the function is defined. For a pair of nested function definitions this means that free variables in the inner function are looked up in the containing function definition and then in the global environment. This is a major difference, perhaps *the* major difference, between R and S-plus.

Lexical scope is a powerful idea that enables higher level functional programming. It allows code such as optimization routines or generic MCMC samplers that naturally take functions as arguments to be written more simply since they do not need to provide mechanisms for passing additional arguments into functions—any additional data needed can be captured using lexical scope.

As a simple example of the use of lexical scope, a function that creates Bernoulli log likelihood functions can be defined as

```
> mkBernoulliLogLik <- function(x) {
+     n <- length(x)
+     k <- sum(x)
+     function(p) k * log(p) + (n - k) * log(1 - p)
+ }
```

This function computes the sufficient statistics, the number of trials n and the number of successes k, and returns the log likelihood function, a function of one argument p:

```
> mkBernoulliLogLik(rbinom(20, 1, 0.3))

function (p)
k * log(p) + (n - k) * log(1 - p)
<environment: 0x243ad98>
```

The result returned by `mkBernoulliLogLik` is a *function closure* consisting of a function definition and the environment in which it is defined. The function definition contains free variables n and k that are bound to the values in the enclosing environment from the call to `mkBernoulliLogLik` that created the log likelihood function closure. This function closure can now be used as a univariate function for plotting or optimization.

6.2. Name spaces

Name spaces (Tierney, 2003) provide a mechanism to ensure that only explicitly exported functions and variables are globally visible. Private functions and variables are seen by code defined in a package but not by code in other packages. In addition to its own private functions and variables code in a package with a name space will only see functions and variables it explicitly imports. This ensures that two packages developed independently will not interfere with each other by inadvertently using the same variable or function name for different purposes or by masking functionality needed from other packages. With the very large number of packages now available for R name spaces have become an essential mechanism to help ensure reliability.

Name spaces are implemented using the ideas of lexical scope. Functions in a name space are defined in the scope of the name space, which is contained in a scope consisting of the name space imports. For mostly historical reasons the base name space is implemented differently from other name spaces and, in particular, cannot have private variables and functions. This limitation will hopefully be overcome in a future release.

6.3. Some open issues

There are a number of areas in which R can use further improvement and development. One weakness is that there are currently too many systems supporting object oriented programming (two: S3 (Chambers and Hastie, 1991) and S4 (Chambers, 1998)), and too many basic graphics systems (again two: the traditional, or standard, system used for most standard graphics and the *grid* system used by lattice and several other packages). Development would be simplified if these duplications could be eliminated, but as there is a large code base using both, and as there remain some unresolved issues in the design of the S4 object system, this will take time.

Support for integrating R with an event loop of a GUI application is also less that satisfactory. A core issue is that R's graphics devices require a minimal event loop and thus supporting an external event loop requires integration of at least two event loops. This is a major challenge and requires further development.

At least partly related to the difficulties of event loop integration is the lack of support in R for dynamic graphics. Some interesting efforts are available, such as the Java-based ***iplots*** and integration with ***ggobi***, but no support for programmable dynamic graphics along the line of Lisp-Stat (Tierney, 1990) is available at this point.

R's approach of working with data in memory allows great flexibility in programming but can limit the size of data sets that can be analyzed, especially on 32-bit hardware. Some efforts to allow models to be fit to data not held in memory are available, as is support for working on subsets of data stored in data bases. Memory demands could be reduced somewhat by storing the workspace on disk rather than in memory. Because of relationships created by lexical scope this is more challenging than in S-plus but may be feasible.

R's documentation system helps to encourage providing high quality documentation but further improvements could be useful. More automation in the process of creating documentation and more flexibility in organization by keywords are two area under consideration.

The package mechanism provides a framework for automatically running examples from the documentation as well as additional tests provided by the package author. But run time tests cannot cover all possible execution paths. Additional tools for analyzing code for possible errors would be a helpful addition and are currently under development.

Performance is an important factor in the effectiveness of R for computationally intensive tasks. While performance of many facets of R is very good, continued efforts are needed to identify and improve performance weaknesses.

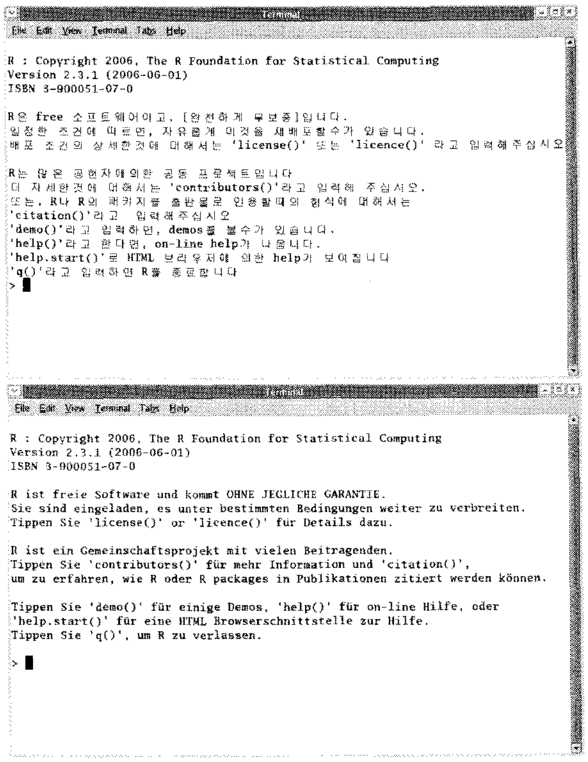
7. Recent Developments

R is an evolving system with major releases occurring twice a year. Many of the changes are incremental improvements and bug fixes, but others are larger and more noticeable. The most recent release at the time of writing included major enhancements to the S4 classes and methods system, performance enhancements of some key functions such as *rep*, support for one package to make C code available for use by C code in other packages, and experimental support for memory use profiling. Other recent releases have included enhancements to support for embedded use of R, improved support for use of FORTRAN 90/95 in packages, and lazy loading of packages to reduce memory use and improve start-up speed (Ripley, 2005).

A major recent addition is support for internationalization and localization that allows major aspects of the R interface to be presented in a language appropriate for the user. The infrastructure to support this was developed by Brian Ripley (2005), with translations provided by a network of volunteers. Figure 7.1 shows R consoles on Fedora Core 5 in Korean and German locales.

8. Work in Progress

New developments in R are typically initiated by one or two members of the R core group. New ideas are accepted by consensus, which has so far not been a limiting factor and has kept disagreements to a minimum. Once a new idea



```
R : Copyright 2006, The R Foundation for Statistical Computing
Version 2.3.1 (2006-06-01)
ISBN 3-900051-07-0

R은 free 소프트웨어이고, [완전하게 무보증]입니다.
일정한 조건에 따르면, 자유롭게 이것을 재배포할수가 있습니다.
배포 조건의 상세한것에 대해서는 'license()' 또는 'licence()' 라고 입력해주십시오

R는 많은 공헌자에의한 공동 프로젝트입니다
더 자세한것에 대해서는 'contributors()'라고 입력해 주십시오.
또는, R나 R의 패키지를 출판물로 인용할때의 형식에 대해서는
'citation()'라고  입력해주십시오
'demo()'라고 입력하면, demos를 볼수가 있습니다.
'help()'라고 한다면, on-line help가 나옵니다.
'help.start()'로 HTML 브라우저에 의한 help가 보여집니다
'q()'라고 입력하면 R를 종료합니다
>
```

```
R : Copyright 2006, The R Foundation for Statistical Computing
Version 2.3.1 (2006-06-01)
ISBN 3-900051-07-0

R ist freie Software und kommt OHNE JEGLICHE GARANTIE.
Sie sind eingeladen, es unter bestimmten Bedingungen weiter zu verbreiten.
Tippen Sie 'license()' or 'licence()' für Details dazu.

R ist ein Gemeinschaftsprojekt mit vielen Beitragenden.
Tippen Sie 'contributors()' für mehr Information und 'citation()',
um zu erfahren, wie R oder R packages in Publikationen zitiert werden können.

Tippen Sie 'demo()' für einige Demos, 'help()' für on-line Hilfe, oder
'help.start()' für eine HTML Browserschnittstelle zur Hilfe.
Tippen Sie 'q()', um R zu verlassen.

>
```

FIGURE 7.1 *R consoles on Fedora Core 5 in Korean and German locales.*

has been incorporated its maintenance becomes a group responsibility, though the original developer usually takes the lead. Individual members of R core have their own priorities on areas to develop. In this section I describe a few of my own areas of current interest.

Code analysis is the process of analyzing R source code to detect possible errors and possible areas for performance improvement. Code analysis can be a precursor to compilation to some form of code that can be executed more efficiently. Byte code is a natural and portable approach for a high level language like R. Developing a byte code compiler in an ongoing project (Tierney, 2001). A preliminary version is available from my web site, though it may need some updating to work with recent releases of R. A separate set of code analysis tools has evolved from the byte code compilation project. These are currently being used in analyzing CRAN submissions and will soon be incorporated into the R distribution.

Two other areas of current work that are described in more detail below are support for parallel computing in R and tools for visualizing functions of three variables and volume data, in particular isosurfaces or three-dimensional contour surfaces.

8.1. Parallel computing in R

With the power of modern processors many computations are essentially instantaneous. There are however computations that can take several minutes, hours, days, or months. Many statisticians have access to tens of workstations that could in principle be harnessed to speed up these computations. In addition, dual-core processors are becoming common, and workstations with dual dual-core processors are no longer unusual.

There are two possible approaches to bringing parallel computation into R: explicit parallelism in which the programmer needs to write explicitly parallel code, and implicit parallelism in which R arranges internally to carry out some computations in parallel. The *snow* package is one approach to supporting explicit parallelism. Implicit parallelism is currently available within linear algebra computations if a suitable version of the BLAS is used. Recent changes have made it easier to choose different BLAS versions at run time. Whether implicit parallelism can be further integrated into R's vectorized arithmetic system will be explored in the near future as discussed briefly below.

The *snow* (Simple Network of Workstations) package (Rossini *et al.*, 2007) is designed to make it easy to write explicitly parallel code for "embarrassingly parallel" problems. It is based on a message passing model in a distributed memory environment, as one has with a collection of workstations or a Beowulf cluster, and can use *rpvm* (Li and Rossini, 2001), *Rmpi* (Yu, 2002), or basic sockets as its communication infrastructure. In the *snow* model a master R process starts up a cluster of worker R processes or nodes, the task is divided up among the workers, and the results are then collected by the master. Bootstrapping is ideally suited to this scatter-compute-gather approach, but there is a need to be careful about random number generation. Fortunately R's random number generation infrastructure allows the use of generators that are designed for use in parallel computations. Several packages providing such generators are available for R; the default used by *snow* is currently the package `rlecuyer` (L'Ecuyer *et al.*, 2002).

A simple example of a bootstrap computation split among 10 R nodes would look like this:

```
> library(snow)
> cl <- makeCluster(10)
> clusterSetupRNG(cl)
> clusterEvalQ(cl, library(boot))
> b <- clusterCall(cl, boot, nuke.data, nuke.fun, R = 100, m = 1,
+     fit.pred = new.fit, x.pred = new.data)
> stopCluster(cl)
```

The first three expressions load the *snow* package on the master, start a cluster of 10 nodes, and set up the parallel random number generators. The fourth expression loads the `boot` package on all the nodes, and the final expression shuts the cluster down. The main work is done by the fifth expression using the function `clusterCall`, which calls a function with identical arguments on all worker nodes and returns a list of the results. Other functions available include `clusterApply`, a parallel version of `lapply`, and `clusterApplyLB`, a load-balancing version of `clusterApply`.

Many computations can be expressed as one or more sequences of scatter-compute-gather operations, perhaps with an initial larger data distribution and some intermediate data exchange. The BSP model for parallel computing is built on this idea (Bisseling, 2004). Current work is exploring how such sessions of related operations can be supported more effectively, while retaining the current simplicity that was a key design goal for the *snow* package. Other areas of current work include exploring how to handle R level errors and user interrupts in *snow* computations and how to handle node or network failures; these are likely to become more important if *snow* is to be extended to run on a computational grid (Foster and Kesselman, 2003). Some early results based on fault tolerance support in PVM are promising.

Because of the data transfer overheads involved the message passing approach used in *snow* is most suitable to coarse-grained parallel tasks. Shared memory approaches that use multiple threads within a single process may be more suitable to finer grained parallelism needed in automatic parallelization. Several BLAS implementations can use multiple threads, usually only for level 3 BLAS operations. I am currently investigating whether similar ideas can be incorporated into R's internal vectorized arithmetic system. The basic idea is that if `x` and `y` are vectors and `f` is a vectorized primitive operation, then one can use several threads, one for each processor, to compute `f(x,y)` in parallel. A key problem is that for short vectors synchronization overhead can make this slower than sequential computation. One solution is to carefully tune the mechanism so that parallel evaluation is only attempted when the vectors are long enough to overcome this overhead. Another interesting approach is so use compilation to fuse several operations so that the threads can carry out a sequence of operations independently and only need to synchronize at the end of the larger fused operation. An important consideration in developing a design for such a parallel vectorized arithmetic system is to also develop an interface to allow packages to contribute their own operations to the pool of operations that can take advantage of this approach. It is hoped that a prototype of this system will be available by September 2007.

8.2. Isosurfaces in R

Another area of current work is the development of tools for visualizing functions of three variables or volume data, such as medical imaging data, collected on a three dimensional grid. Isosurfaces or three dimensional contours are a useful tool. As part of this Ph.D. research Dai Feng has implemented a version of the *marching cubes* algorithm in R. The result is a collection of triangles approximating a contour surface. These triangles can then be rendered using the `rgl` package, which provides an interactive 3D image, or using standard graphics. Rendering in standard graphics requires that colors be adjusted using a lighting algorithm. The package `misc3d` contains the current implementation.

A challenging issue is how to represent nested sets of contours. One option is to make outer contours transparent. This is possible with the `rgl` rendering engine and is illustrated in the right hand plot in Figure 8.1. For standard graphics devices that do not support transparency an alternative is to cut away some of the plot as shown on the left in Figure 8.1. Some R graphics devices now support transparency, but there are some issues with the handling of boundaries of polygons that need to be resolved for this to be effective for transparent contour rendering.

9. CONCLUSIONS

Many resources are available for those interested in learning more about R. There are now several book devoted to R and S, including Dalgaard (2004) and Venables and Ripley (2002). Several good introductions are available on the web, many linked from the R home page. The R graph gallery provides a window into the range of graphics available in R and its contributed packages. Help is

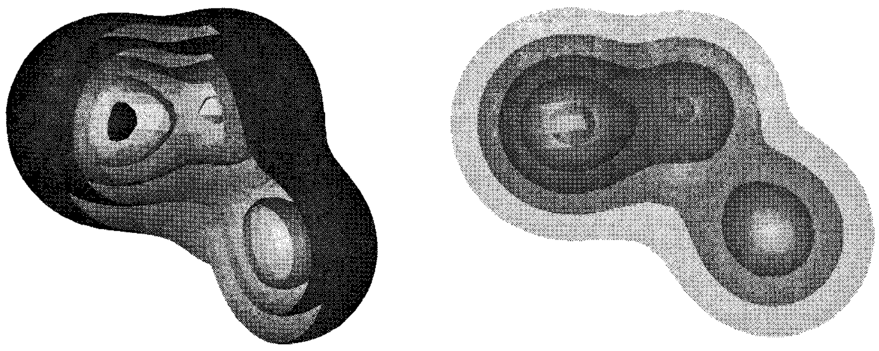


FIGURE 8.1 *Nested contours of a mixture of tri-variate normal densities. The figure on the left is rendered using standard graphics; the figure on the right is rendered with* `rgl`.

available through an active mailing list community; mailing lists are archived and are accessible from within R using the `RSiteSearch` command. A recent addition is the R Wiki.

R has been very successful on a number of dimensions: it provides a valuable tool for data analysis; it provides a framework for disseminating new methodology and enabling the development of other large projects (*e.g.* Bioconductor (Gentleman *et al.*, 2004)), and it serves as a framework for statistical computing research. R also illustrates the value of the open source development model for statistical software.

R's success has not come without costs. Changes now affect a large number of users; this can limit innovation. The desire to support new, less experienced users can be in conflict with the need to support advanced use as well. Nevertheless, R can be expected to evolve and improve for a number of years to come.

ACKNOWLEDGEMENTS

Parts of this presentation were drawn from material for R short courses prepared by members of the R core team. Code examples were integrated using the `Sweave` system.

REFERENCES

ADLER, D. AND MURDOCH, D. (2006). Package `rgl`, CRAN.

BECKER, R. A. AND CHAMBERS, J. M. (1984). *S: An Interactive Environment for Data Analysis and Graphics*, Chapman & Hall/CRC.

BECKER, R. A., CHAMBERS, J. M. AND WILKS, A. R. (1988). *The New S Language: A Programming Environment for Data Analysis and Graphics*, Chapman & Hall(Formerly Monterey: Wadsworth and Brooks/Cole.), New York.

BECKER, R. A., CLEVELAND, W. S. AND SHYU, M.-J. (1996). "The visual design and control of trellis display", *Journal of Computational and Graphical Statistics*, **5**, 123–155.

BIOCONDUCTOR: SOFTWARE FOR BIOINFORMATICS. `http://www.bioconductor.org`.

BISSELING, R. H. (2004). *Parallel Scientific Computation: A Structured Approach using BSP and MPI*, Oxford University Press, New York.

CANTY, A. AND RIPLEY, B. (2006). Package `boot`, CRAN.

CHAMBERS, J. M. AND HASTIE, T. J. (1991). *Statistical Models in S*, Chapman & Hall/CRC.

CHAMBERS, J. M. (1998). *Programming with Data: A Guide to the S Language*, Springer-Verlag, New York.

CLEVELAND, W. S. (1993). *Visualizing Data*, Hobart Press, Summit, New Jersey.

THE COMPREHENSIVE R ARCHIVE NETWORK. `http://cran.r-project.org`.

DALGAARD, P. (2004). *Introductory Statistics with R*, Springer-Verlag, New York.

FENG, D. AND TIERNEY, L. (2006). Package `misc3d`, CRAN.

FOSTER, I. AND KESSELMAN, C. (2003). *The Grid 2: Blueprint for a New Computing Infrastructure*, 2nd ed., Morgan Kaufmann, Amsterdam.

GENTLEMAN, R. AND IHAKA, R. (2000). "Lexical scope and statistical computing", *Journal of Computational and Graphical Statistics*, **9**, 491–508.

GENTLEMAN, R. C., CAREY, V. J., BATES, D. J., BOLSTAD, B. M., DETTLING, M., DUDOIT, S., ELLIS, B., GAUTIER, L., GE, Y., GENTRY, J., HORNIK, K., HOTHORN, T., HUBER, W., IACUS, S., IRIZARRY, R., LEISCH, F., LI, C., MAECHLER, M., ROSSINI, A. J., SAWITZKI, G., SMITH, C., SMYTH, G. K., TIERNEY, L., YANG, Y. H. AND ZHANG, J. (2004). "Bioconductor: Open software development for computational biology and bioinformatics", *Genome Biology*, **5**, R80.

GGOBI DATA VISUALIZATION SYSTEM. `http://www.ggobi.org`.

IHAKA, R. AND GENTLEMAN, R. (1996). "R: A language for data analysis and graphics", *Journal of Computational and Graphical Statistics*, **5**, 299–314.

L'ECUYER, P., SIMARD, R., CHEN, E. J. AND KELTON, W. D. (2002). "An object-oriented random-number package with many long streams and substreams", *Operations Research*, **50**, 1073–1075.

LEISCH, F. (2002). "Sweave, Part I: Mixing R and LaTeX", *R News*, **2**, 28–31.

LEISCH, F. (2003). "Sweave, Part II: Package Vignettes", *R News*, **3**, 21–24.

LI, M. N. AND ROSSINI, A. J. (2001). "RPVM: Cluster statistical computing in R", *R News*, **1**, 4–7.

LUMLEY, T. (2005). Package `biglm`, CRAN.

MURRELL, P. (2005). *R Graphics*, Chapman & Hall/CRC.

PLUMMER, M., BEST, N. AND COWLES, K. (2005). Package `coda`, CRAN.

R DEVELOPMENT CORE TEAM. (2006). "R: A language and environment for statistical computing", *R Foundation for Statistical Computing*, Vienna, Austria.

R GRAPH GALLERY. `http://addictedtor.free.fr/graphiques`.

THE R PROJECT FOR STATISTICAL COMPUTING. `http://cran.r-project.org`.

R WIKI. `http://wiki.r-project.org`.

RIPLEY, B. D. (2004). "Lazy loading and packages in R 2.0.0", *R News*, **4**, 2–4.

RIPLEY, B. D. (2005). "Internationalization features of R 2.1.0", *R News*, **5**, 2–7.

ROSSINI, A. J., TIERNEY, L. AND LI, N. (2007). "Simple parallel statistical computing in R", *Journal of Computational and Graphical Statistics*, **16**, to appear.

SEVCIKOVA, H. AND ROSSINI, A. J. (2005). Package `snowFT`, CRAN.

SEVCIKOVA, H. AND ROSSINI, T. (2004). Package `rlecuyer`, CRAN.

SMITH, B. J. (2005). Package `boa`, CRAN.

THOMAS, A., OHARA, B., LIGGES, U. AND STURTZ, S. (2006). "Making BUGS open", *R News*, **6**, 12–17.

TIERNEY, L. (1990). *LISP-STAT: An Object-Oriented Environment for Statistical Computing and Dynamic Graphics*, Wiley, New York.

TIERNEY, L. (2001). "Compiling R: A preliminary report", *Proceedings of the 2nd International Workshop on Distributed Statistical Computing*, Vienna, Austria.

TIERNEY, L. (2003). "Name space management for R", *R News*, **3**, 2–5.

TIERNEY, L., ROSSINI, A. J., LI, N. AND SEVCIKOVA, H. (2006). Package `snow`, CRAN.

URBANEK, S. AND WICHTREY, T. (2006). Package `iplots`, CRAN.

VENABLES, W. N. AND RIPLEY, B. D. (2002). *Modern Applied Statistics with S*, 4th ed., Springer, New York.

YU, H. (2002). "Rmpi: Parallel statistical computing in R", *R News*, **2**, 10–14.