Statistics and Computing

Series Editors:
J. Chambers
D. Hand
W. Härdle

Statistics and Computing

Brusco/Stahl: Branch and Bound Applications in Combinatorial Data Analysis
Chambers: Software for Data Analysis: Programming with R
Dalgaard: Introductory Statistics with R
Gentle: Elements of Computational Statistics
Gentle: Numerical Linear Algebra for Applications in Statistics
Gentle: Random Number Generation and Monte Carlo Methods, 2nd ed.
Härdle/Klinke/Turlach: XploRe: An Interactive Statistical Computing Environment
Hörmann/Leydold/Derflinger: Automatic Nonuniform Random Variate Generation
Krause/Olson: The Basics of S-PLUS, 4th ed.
Lange: Numerical Analysis for Statisticians
Lemmon/Schafer: Developing Statistical Software in Fortran 95
Loader: Local Regression and Likelihood
Ó Ruanaidh/Fitzgerald: Numerical Bayesian Methods Applied to Signal Processing
Pannatier: VARIOWIN: Software for Spatial Data Analysis in 2D
Pinheiro/Bates: Mixed-Effects Models in S and S-PLUS
Unwin/Theus/Hofmann: Graphics of Large Datasets: Visualizing a Million
Venables/Ripley: Modern Applied Statistics with S, 4th ed.
Venables/Ripley: S Programming
Wilkinson: The Grammar of Graphics, 2nd ed.

John M. Chambers

Software for Data Analysis

Programming with R

Springer

John Chambers
Department of Statistics–Sequoia Hall
390 Serra Mall
Stanford University
Stanford, CA 94305-4065
USA
jmc@r-project.org

ISBN: 978-0-387-75935-7 e-ISBN: 978-0-387-75936-4
DOI: 10.1007/978-0-387-75936-4

Library of Congress Control Number: 2008922937

Printed on acid-free paper.

9 8 7 6 5 4 3

springer.com

Preface

This is a book about *Software for Data Analysis*: using computer software to extract information from some source of data by organizing, visualizing, modeling, or performing any other relevant computation on the data. We all seem to be swimming in oceans of data in the modern world, and tasks ranging from scientific research to managing a business require us to extract meaningful information from the data using computer software.

This book is aimed at those who need to select, modify, and create software to explore data. In a word, programming. Our programming will center on the R system. R is an open-source software project widely used for computing with data and giving users a huge base of techniques. Hence, *Programming with R*.

R provides a general language for interactive computations, supported by techniques for data organization, graphics, numerical computations, model-fitting, simulation, and many other tasks. The core system itself is greatly supplemented and enriched by a huge and rapidly growing collection of software packages built on R and, like R, largely implemented as open-source software. Furthermore, R is designed to encourage learning and developing, with easy starting mechanisms for programming and also techniques to help you move on to more serious applications. The complete picture—the R system, the language, the available packages, and the programming environment—constitutes an unmatched resource for computing with data.

At the same time, the "with" word in *Programming with R* is important. No software system is sufficient for exploring data, and we emphasize interfaces between systems to take advantage of their respective strengths.

Is it worth taking time to develop or extend your skills in such programming? Yes, because the investment can pay off both in the ability to ask questions and in the trust you can have in the answers. Exploring data with the right questions and providing trustworthy answers to them are the key to analyzing data, and the twin principles that will guide us.

What's in the book?

A sequence of chapters in the book takes the reader on successive steps from user to programmer to contributor, in the gradual progress that R encourages. Specifically: using R; simple programming; packages; classes and methods; inter-system interfaces (Chapters 2; 3; 4; 9 and 10; 11 and 12). The order reflects a natural progression, but the chapters are largely independent, with many cross references to encourage browsing.

Other chapters explore computational techniques needed at all stages: basic computations; graphics; computing with text (Chapters 6; 7; 8). Lastly, a chapter (13) discusses how R works and the appendix covers some topics in the history of the language.

Woven throughout are a number of reasonably serious examples, ranging from a few paragraphs to several pages, some of them continued elsewhere as they illustrate different techniques. See "Examples" in the index. I encourage you to explore these as leisurely as time permits, thinking about how the computations evolve, and how you would approach these or similar examples.

The book has a companion R package, `SoDA`, obtainable from the main CRAN repository, as described in Chapter 4. A number of the functions and classes developed in the book are included in the package. The package also contains code for most of the examples; see the documentation for `"Examples"` in the package.

Even at five hundred pages, the book can only cover a fraction of the relevant topics, and some of those receive a pretty condensed treatment. Spending time alternately on reading, thinking, and interactive computation will help clarify much of the discussion, I hope. Also, the final word is with the online documentation and especially with the software; a substantial benefit of open-source software is the ability to drill down and see what's really happening.

Who should read this book?

I've written this book with three overlapping groups of readers generally in mind.

First, "data analysts"; that is, anyone with an interest in exploring data, especially in serious scientific studies. This includes statisticians, certainly, but increasingly others in a wide range of disciplines where data-rich studies now require such exploration. Helping to enable exploration is our mission

here. I hope and expect that you will find that working with R and related software enhances your ability to learn from the data relevant to your interests.

If you have not used R or S-Plus® before, you should precede this book (or at least supplement it) with a more basic presentation. There are a number of books and an even larger number of Web sites. Try searching with a combination of "introduction" or "introductory" along with "R". Books by W. John Braun and Duncan J. Murdoch [2], Michael Crawley [11], Peter Dalgaard [12], and John Verzani [24], among others, are general introductions (both to R and to statistics). Other books and Web sites are beginning to appear that introduce R or S-Plus with a particular area of application in mind; again, some Web searching with suitable terms may find a presentation attuned to your interests.

A second group of intended readers are people involved in research or teaching related to statistical techniques and theory. R and other modern software systems have become essential in the research itself and in communicating its results to the community at large. Most graduate-level programs in statistics now provide some introduction to R. This book is intended to guide you on the followup, in which your software becomes more important to your research, and often a way to share results and techniques with the community. I encourage you to push forward and organize your software to be reusable and extendible, including the prospect of creating an R package to communicate your work to others. Many of the R packages now available derive from such efforts..

The third target group are those more directly interested in software and programming, particularly software for data analysis. The efforts of the R community have made it an excellent medium for "packaging" software and providing it to a large community of users. R is maintained on all the widely used operating systems for computing with data and is easy for users to install. Its package mechanism is similarly well maintained, both in the central CRAN repository and in other repositories. Chapter 4 covers both using packages and creating your own. R can also incorporate work done in other systems, through a wide range of inter-system interfaces (discussed in Chapters 11 and 12).

Many potential readers in the first and second groups will have some experience with R or other software for statistics, but will view their involvement as doing only what's absolutely necessary to "get the answers". This book will encourage moving on to think of the interaction with the software as an important and valuable part of your activity. You may feel inhibited by not having done much programming before. Don't be. Programming with

R can be approached gradually, moving from easy and informal to more ambitious projects. As you use R, one of its strengths is its flexibility. By making simple changes to the commands you are using, you can customize interactive graphics or analysis to suit your needs. This is the takeoff point for programming: As Chapters 3 and 4 show, you can move from this first personalizing of your computations through increasingly ambitious steps to create your own software. The end result may well be your own contribution to the world of R-based software.

How should you read this book?

Any way that you find helpful or enjoyable, of course. But an author often imagines a conversation with a reader, and it may be useful to share my version of that. In many of the discussions, I imagine a reader pausing to decide how to proceed, whether with a specific technical point or to choose a direction for a new stage in a growing involvement with software for data analysis. Various chapters chart such stages in a voyage that many R users have taken from initial, casual computing to a full role as a contributor to the community. Most topics will also be clearer if you can combine reading with hands-on interaction with R and other software, in particular using the `Examples` in the `SoDA` package.

This pausing for reflection and computing admittedly takes a little time. Often, you will just want a "recipe" for a specific task—what is often called the "cookbook" approach. By "cookbook" in software we usually imply that one looks a topic up in the index and finds a corresponding explicit recipe. That should work sometimes with this book, but we concentrate more on general techniques and extended examples, with the hope that these will equip readers to deal with a wider range of tasks. For the reader in a hurry, I try to insert pointers to online documentation and other resources.

As an enthusiastic cook, though, I would point out that the great cookbooks offer a range of approaches, similar to the distinction here. Some, such as the essential *Joy of Cooking* do indeed emphasize brief, explicit recipes. The best of these books are among the cook's most valuable resources. Other books, such as Jacques Pépin's masterful *La Technique*, teach you just that: techniques to be applied. Still others, such as the classic *Mastering the Art of French Cooking* by Julia Child and friends, are about learning and about underlying concepts as much as about specific techniques. It's the latter two approaches that most resemble the goals of the present book. The book presents a number of explicit recipes, but the deeper emphasis is in on concepts and techniques. And behind those in turn, there will be two general principles of good software for data analyis.

Acknowledgments

The ideas discussed in the book, as well as the software itself, are the results of projects involving many people and stretching back more than thirty years (see the appendix for a little history).

Such a scope of participants and time makes identifying all the individuals a hopeless task, so I will take refuge in identifying groups, for the most part. The most recent group, and the largest, consists of the "contributors to R", not easy to delimit but certainly comprising hundreds of people at the least. Centrally, my colleagues in R-core, responsible for the survival, dissemination, and evolution of R itself. These are supplemented by other volunteers providing additional essential support for package management and distribution, both generally and specifically for repositories such as CRAN, BioConductor, omegahat, RForge and others, as well as the maintainers of essential information resources—archives of mailing lists, search engines, and many tutorial documents. Then the authors of the thousands of packages and other software forming an unprecedented base of techniques; finally, the interested users who question and prod through the mailing lists and other communication channels, seeking improvements. This community as a whole is responsible for realizing something we could only hazily articulate thirty-plus years ago, and in a form and at a scale far beyond our imaginings.

More narrowly from the viewpoint of this book, discussions within R-core have been invaluable in teaching me about R, and about the many techniques and facilities described throughout the book. I am only too aware of the many remaining gaps in my knowledge, and of course am responsible for all inaccuracies in the descriptions herein.

Looking back to the earlier evolution of the S language and software, time has brought an increasing appreciation of the contribution of colleagues and management in Bell Labs research in that era, providing a nourishing environment for our efforts, perhaps indeed a unique environment. Rick Becker, Allan Wilks, Trevor Hastie, Daryl Pregibon, Diane Lambert, and W. S. Cleveland, along with many others, made essential contributions.

Since retiring from Bell Labs in 2005, I have had the opportunity to interact with a number of groups, including students and faculty at several universities. Teaching and discussions at Stanford over the last two academic years have been very helpful, as were previous interactions at UCLA and at Auckland University. My thanks to all involved, with special thanks to Trevor Hastie, Mark Hansen, Ross Ihaka and Chris Wild.

A number of the ideas and opinions in the book benefited from collab-

orations and discussions with Duncan Temple Lang, and from discussions with Robert Gentleman, Luke Tierney, and other experts on R, not that any of them should be considered at all responsible for defects therein.

The late Gene Roddenberry provided us all with some handy terms, and much else to be enjoyed and learned from.

Each of our books since the beginning of S has had the benefit of the editorial guidance of John Kimmel; it has been a true and valuable collaboration, long may it continue.

John Chambers
Palo Alto, California
January, 2008

PS: The Web page `stat.stanford.edu/~jmc4/errata` contains corrections and notes on developments since the initial publication of the book.

Contents

Chapter 1

Introduction: Principles and Concepts

This chapter presents some of the concepts and principles that recur throughout the book. We begin with the two guiding principles: the mission to explore and the responsibility to be trustworthy (Sections 1.1 and 1.2). With these as guidelines, we then introduce some concepts for programming with R (Section 1.3, page 4) and add some justification for our emphasis on that system (Section 1.4, page 9).

1.1 Exploration: The Mission

The first principle I propose is that our *Mission*, as users and creators of software for data analysis, is to enable the best and most thorough exploration of data possible. That means that users of the software must be able to ask the meaningful questions about their applications, quickly and flexibly.

Notice that speed here is human speed, measured in clock time. It's the time that the actual computations take, but usually more importantly, it's also the time required to formulate the question and to organize the data in a way to answer it. This is the exploration, and software for data analysis makes it possible. A wide range of techniques is needed to access and transform data, to make predictions or summaries, to communicate results to others, and to deal with ongoing processes.

Whenever we consider techniques for these and other requirements in the chapters that follow, the first principle we will try to apply is the *Mission*:

How can these techniques help people to carry out this specific kind of exploration?

Ensuring that software for data analysis exists for such purposes is an important, exciting, and challenging activity. Later chapters examine how we can select and develop software using R and other systems.

The importance, excitement, and challenge all come from the central role that data and computing have come to play in modern society. Science, business and many other areas of society continually rely on understanding data, and that understanding frequently involves large and complicated data processes.

A few examples current as the book is written can suggest the flavor:

- Many ambitious projects are underway or proposed to deploy *sensor networks*, that is, coordinated networks of devices to record a variety of measurements in an ongoing program. The data resulting is essential to understand environmental quality, the mechanisms of weather and climate, and the future of biodiversity in the earth's ecosystems. In both scale and diversity, the challenge is unprecedented, and will require merging techniques from many disciplines.

- Astronomy and cosmology are undergoing profound changes as a result of large-scale digital mappings enabled by both satellite and ground recording of huge quantities of data. The scale of data collected allows questions to be addressed in an overall sense that before could only be examined in a few, local regions.

- Much business activity is now carried out largely through distributed, computerized processes that both generate large and complex streams of data and also offer through such data an unprecedented opportunity to understand one's business quantitatively. Telecommunications in North America, for example, generates databases with conceptually billions of records. To explore and understand such data has great attraction for the business (and for society), but is enormously challenging.

These and many other possible examples illustrate the importance of what John Tukey long ago characterized as "the peaceful collision of computing and data analysis". Progress on any of these examples will require the ability to explore the data, flexibly and in a reasonable time frame.

1.2 Trustworthy Software: The Prime Directive

Exploration is our mission; we and those who use our software want to find new paths to understand the data and the underlying processes. The mission is, indeed, to boldly go where no one has gone before. But, we need boldness to be balanced by our responsibility. We have a responsibility for the results of data analysis that provides a key compensating principle.

The complexity of the data processes and of the computations applied to them mean that those who receive the results of modern data analysis have limited opportunity to verify the results by direct observation. Users of the analysis have no option but to trust the analysis, and by extension the software that produced it. Both the data analyst and the software provider therefore have a strong responsibility to produce a result that is trustworthy, and, if possible, one that can be *shown* to be trustworthy.

This is the second principle: the computations and the software for data analysis should be trustworthy: they should do what they claim, and be seen to do so. Neither those who view the results of data analysis nor, in many cases, the statisticians performing the analysis can directly validate extensive computations on large and complicated data processes. Ironically, the steadily increasing computer power applied to data analysis often distances the results further from direct checking by the recipient. The many computational steps between original data source and displayed results must all be truthful, or the effect of the analysis may be worthless, if not pernicious. This places an obligation on all creators of software to program in such a way that the computations can be understood and trusted. This obligation I label the *Prime Directive*.

Note that the directive in no sense discourages exploratory or approximate methods. As John Tukey often remarked, better an approximate answer to the right question than an exact answer to the wrong question. We should seek answers boldly, but always explaining the nature of the method applied, in an open and understandable format, supported by as much evidence of its quality as can be produced. As we will see, a number of more technically specific choices can help us satisfy this obligation.

Readers who have seen the *Star Trek*® television series[1] may recognize the term "prime directive". Captains Kirk, Picard, and Janeway and their crews were bound by a directive which (slightly paraphrased) was: Do nothing to interfere with the natural course of a new civilization. Do not distort

[1] Actually, at least five series, from "The Original" in 1966 through "Enterprise", not counting the animated version, plus many films. See `startrek.com` and the many reruns if this is a gap in your cultural background.

the development. Our directive is not to distort the message of the data, and to provide computations whose content can be trusted and understood.

The prime directive of the space explorers, notice, was not their *mission* but rather an important safeguard to apply in pursuing that mission. Their mission was to explore, to "boldly go where no one has gone before", and all that. That's really our mission too: to explore how software can add new abilities for data analysis. And our own prime directive, likewise, is an important caution and guiding principle as we create the software to support our mission.

Here, then, are two motivating principles: the mission, which is bold exploration; and the prime directive, trustworthy software. We will examine in the rest of the book how to select and program software for data analysis, with these principles as guides. A few aspects of R will prove to be especially relevant; let's examine those next.

1.3 Concepts for Programming with R

The software and the programming techniques to be discussed in later chapters tend to share some concepts that make them helpful for data analysis. Exploiting these concepts will often benefit both the effectiveness of programming and the quality of the results. Each of the concepts arises naturally in later chapters, but it's worth outlining them together here for an overall picture of our strategy in programming for data analysis.

Functional Programming

Software in R is written in a *functional style* that helps both to understand the intent and to ensure that the implementation corresponds to that intent. Computations are organized around functions, which can encapsulate specific, meaningful computational results, with implementations that can be examined for their correctness. The style derives from a more formal theory of *functional programming* that restricts the computations to obtain well-defined or even formally verifiable results. Clearly, programming in a fully functional manner would contribute to trustworthy software. The S language does not enforce a strict functional programming approach, but does carry over some of the flavor, particularly when you make some effort to emphasize simple functional definitions with minimal use of non-functional computations.

As the scope of the software expands, much of the benefit from functional style can be retained by using *functional methods* to deal with varied types

of data, within the general goal defined by the generic function.

Classes and Methods

The natural complement to functional style in programming is the definition of classes of objects. Where functions should clearly encapsulate the actions in our analysis, classes should encapsulate the nature of the objects used and returned by calls to functions. The duality between function calls and objects is a recurrent theme of programming with R. In the design of new classes, we seek to capture an underlying concept of what the objects mean. The relevant techniques combine directly specifying the contents (the slots), relating the new class to existing classes (the inheritance), and expressing how objects should be created and validated (methods for initializing and validating).

Method definitions knit together functions and classes. Well-designed methods extend the generic definition of what a function does to provide a specific computational method when the argument or arguments come from specified classes, or inherit from those classes. In contrast to methods that are solely class-based, as in common object-oriented programming languages such as C++ or Java, methods in R are part of a rich but complex network of functional and object-based computation.

The ability to define classes and methods in fact is itself a major advantage in adhering to the *Prime Directive*. It gives us a way to isolate and define formally what information certain objects should contain and how those objects should behave when functions are applied to them.

Data Frames

Trustworthy data analysis depends first on trust in the data being analyzed. Not so much that the data must be perfect, which is impossible in nearly any application and in any case beyond our control, but rather that trust in the analysis depends on trust in the relation between the data as we use it and the data as it has entered the process and then has been recorded, organized and transformed.

In serious modern applications, the data usually comes from a process external to the analysis, whether generated by scientific observations, commercial transactions or any of many other human activities. To access the data for analysis by well-defined and trustworthy computations, we will benefit from having a description, or model, for the data that corresponds to its natural home (often in DBMS or spreadsheet software), but can also be

a meaningful basis for data as used in the analysis. Transformations and restructuring will often be needed, but these should be understandable and defensible.

The model we will emphasize is the *data frame*, essentially a formulation of the traditional view of observations and variables. The data frame has a long history in the S language but modern techniques for classes and methods allow us to extend the use of the concept. Particularly useful techniques arise from using the data frame concept both within R, for model-fitting, data visualization, and other computations, and also for effective communication with other systems. Spreadsheets and relational database software both relate naturally to this model; by using it along with unambiguous mechanisms for interfacing with such software, the meaning and structure of the data can be preserved. Not all applications suit this approach by any means, but the general data frame model provides a valuable basis for trustworthy organization and treatment of many sources of data.

Open Source Software

Turning to the general characteristics of the languages and systems available, note that many of those discussed in this book are *open-source* software systems; for example, R, Perl, Python, many of the database systems, and the Linux operating system. These systems all provide access to source code sufficient to generate a working version of the software. The arrangement is not equivalent to "public-domain" software, by which people usually mean essentially unrestricted use and copying. Instead, most open-source systems come with a copyright, usually held by a related group or foundation, and with a license restricting the use and modification of the software. There are several versions of license, the best known being the Gnu General Public License and its variants (see `gnu.org/copyleft/gpl.html`), the famous GPL. R is distributed under a version of this license (see the `"COPYING"` file in the home directory of R). A variety of other licenses exists; those accepted by the *Open Source Initiative* are described at `opensource.org/licenses`.

Distinctions among open-source licenses generate a good deal of heat in some discussions, often centered on what effect the license has on the usability of the software for commercial purposes. For our focus, particularly for the concern with trustworthy software for data analysis, these issues are not directly relevant. The popularity of open-source systems certainly owes a lot to their being thought of as "free", but for our goal of trustworthy software, this is also not the essential property. Two other characteristics contribute more. First, the simple openness itself allows any sufficiently

competent observer to enquire fully about what is actually being computed. There are no intrinsic limitations to the validation of the software, in the sense that it is all there. Admittedly, only a minority of users are likely to delve very far into the details of the software, but some do. The ability to examine and critique every part of the software makes for an open-ended scope for verifying the results.

Second, open-source systems demonstrably generate a spirit of community among contributors and active users. User groups, e-mail lists, chat rooms and other socializing mechanisms abound, with vigorous discussion and controversy, but also with a great deal of effort devoted to testing and extension of the systems. The active and demanding community is a key to trustworthy software, as well as to making useful tools readily available.

Algorithms and Interfaces

R is explicitly seen as built on a set of routines accessed by an interface, in particular by making use of computations in C or Fortran. User-written extensions can make use of such interfaces, but the core of R is itself built on them as well. Aside from routines that implement R-dependent techniques, there are many basic computations for numerical results, data manipulation, simulation, and other specific computational tasks. These implementations we can term *algorithms*. Many of the core computations on which the R software depends are now implemented by collections of such software that are widely used and tested. The algorithm collections have a long history, often predating the larger-scale open-source systems. It's an important concept in programming with R to seek out such algorithms and make them part of a new computation. You should be able to import the trust built up in the non-R implementation to make your own software more trustworthy.

Major collections on a large scale and many smaller, specialized algorithms have been written, generally in the form of subroutines in Fortran, C, and a few other general programming languages. Thirty-plus years ago, when I was writing *Computational Methods for Data Analysis*, those who wanted to do innovative data analysis often had to work directly from such routines for numerical computations or simulation, among other topics. That book expected readers to search out the routines and install them in the readers' own computing environment, with many details left unspecified.

An important and perhaps under-appreciated contribution of R and other systems has been to embed high-quality algorithms for many computations in the system itself, automatically available to users. For example, key parts of the LAPACK collection of computations for numerical linear algebra

are included in R, providing a basis for fitting linear models and for other matrix computations. Other routines in the collection may not be included, perhaps because they apply to special datatypes or computations not often encountered. These routines can still be used with R in nearly all cases, by writing an interface to the routine (see Chapter 11).

Similarly, the internal code for pseudo-random number generation includes most of the well-regarded and thoroughly tested algorithms for this purpose. Other tasks, such as sorting and searching, also use quality algorithms. Open-source systems provide an advantage when incorporating such algorithms, because alert users can examine in detail the support for computations. In the case of R, users do indeed question and debate the behavior of the system, sometimes at great length, but overall to the benefit of our trust in programming with R.

The best of the algorithm collections offer another important boost for trustworthy software in that the software may have been used in a wide variety of applications, including some where quality of results is critically important. Collections such as LAPACK are among the best-tested substantial software projects in existence, and not only by users of higher-level systems. Their adaptability to a wide range of situations is also a frequent benefit.

The process of incorporating quality algorithms in a user-oriented system such as R is ongoing. Users can and should seek out the best computations for their needs, and endeavor to make these available for their own use and, through packages, for others as well.

Incorporating algorithms in the sense of subroutines in C or Fortran is a special case of what we call *inter-system interfaces* in this book. The general concept is similar to that for algorithms. Many excellent software systems exist for a variety of purposes, including text-manipulation, spreadsheets, database management, and many others. Our approach to software for data analysis emphasizes R as the central system, for reasons outlined in the next section. In any case, most users will prefer to have a single home system for their data analysis.

That does not mean that we should or can absorb all computations directly into R. This book emphasizes the value of expressing computations in a natural way while making use of high-quality implementations in whatever system is suitable. A variety of techniques, explored in Chapter 12, allows us to retain a consistent approach in programming with R at the same time.

1.4 The R System and the S Language

This book includes computations in a variety of languages and systems, for tasks ranging from database management to text processing. Not all systems receive equal treatment, however. The central activity is data analysis, and the discussion is from the perspective that our data analysis is mainly expressed in R; when we examine computations, the results are seen from an interactive session with R. This view does not preclude computations done partly or entirely in other systems, and these computations may be complete in themselves. The data analysis that the software serves, however, is nearly always considered to be in R.

Chapter 2 covers the use of R broadly but briefly (if you have no experience with it, you might want to consult one of the introductory books or other sources mentioned on page vii in the preface). The present section give a brief summary of the system and relates it to the philosophy of the book.

R is an open-source software system, supported by a group of volunteers from many countries. The central control is in the hands of a group called R-core, with the active collaboration of a much larger group of contributors. The base system provides an interactive language for numerical computations, data management, graphics and a variety of related calculations. It can be installed on Windows, Mac OS X, and Linux operating systems, with a variety of graphical user interfaces. Most importantly, the base system is supported by well over a thousand packages on the central repository `cran.r-project.org` and in other collections.

R began as a research project of Ross Ihaka and Robert Gentleman in the 1990s, described in a paper in 1996 [17]. It has since expanded into software used to implement and communicate most new statistical techniques. The software in R implements a version of the S language, which was designed much earlier by a group of us at Bell Laboratories, described in a series of books ([1], [6], and [5] in the bibliography).

The S-Plus system also implements the S language. Many of the computations discussed in the book work in S-Plus as well, although there are important differences in the evaluation model, noted in later chapters. For more on the history of S, see Appendix A, page 475.

The majority of the software in R is itself written in the same language used for interacting with the system, a dialect of the S language. The language evolved in essentially its present form during the 1980s, with a generally functional style, in the sense used on page 4: The basic unit of programming is a function. Function calls usually compute an object that is a

function of the objects passed in as arguments, without side effects to those arguments. Subsequent evolution of the language introduced formal classes and methods, again in the sense discussed in the previous section. Methods are specializations of functions according to the class of one or more of the arguments. Classes define the content of objects, both directly and through inheritance. R has added a number of features to the language, while remaining largely compatible with S. All these topics are discussed in the present book, particularly in Chapters 3 for functions and basic programming, 9 for classes, and 10 for methods.

So why concentrate on R? Clearly, and not at all coincidentally, R reflects the same philosophy that evolved through the S language and the approach to data analysis at Bell Labs, and which largely led me to the concepts I'm proposing in this book. It is relevant that S began as a medium for statistics researchers to express their own computations, in support of research into data analysis and its applications. A direct connection leads from there to the large community that now uses R similarly to implement new ideas in statistics, resulting in the huge resource of R packages.

Added to the characteristics of the language is R's open-source nature, exposing the system to continual scrutiny by users. It includes some algorithms for numerical computations and simulation that likewise reflect modern, open-source computational standards in these fields. The LAPACK software for numerical linear algebra is an example, providing trustworthy computations to support statistical methods that depend on linear algebra.

Although there is plenty of room for improvement and for new ideas, I believe R currently represents the best medium for quality software in support of data analysis, and for the implementation of the principles espoused in the present book. From the perspective of our first development of S some thirty-plus years ago, it's a cause for much gratitude and not a little amazement.

Chapter 2

Using R

This chapter covers the essentials for using R to explore data interactively. Section 2.1 covers basic access to an R session. Users interact with R through a single language for both data analysis and programming (Section 2.3, page 19). The key concepts are function calls in the language and the objects created and used by those calls (2.4, 24), two concepts that recur throughout the book. The huge body of available software is organized around packages that can be attached to the session, once they are installed (2.5, 25). The system itself can be downloaded and installed from repositories on the Web (2.6, 29); there are also a number of resources on the Web for information about R (2.7, 31).

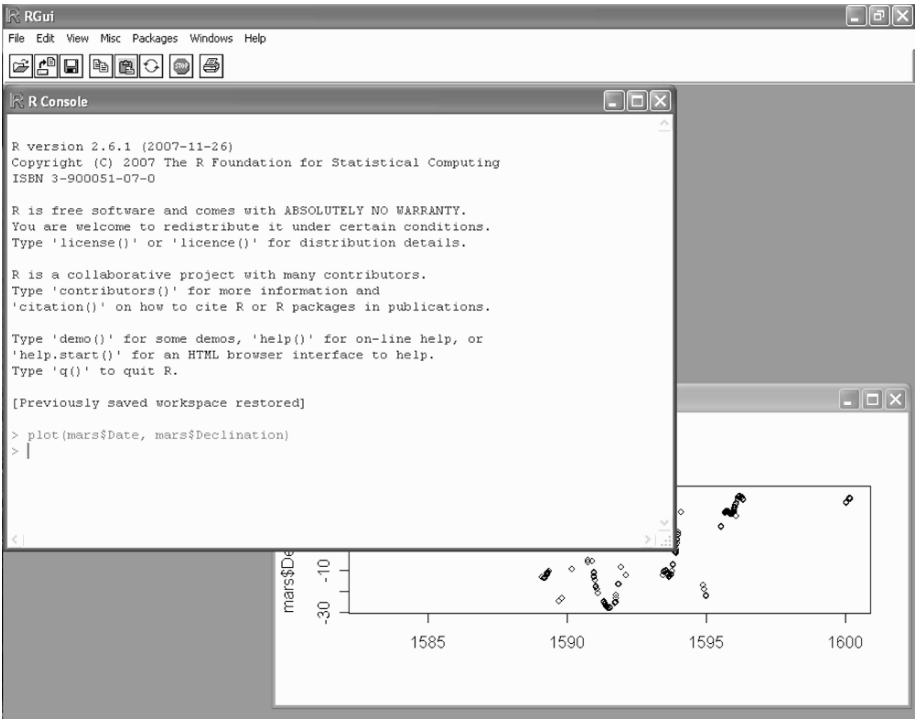
Lastly, we examine aspects of R that may raise difficulties for some new users (2.8, 34).

2.1 Starting R

R runs on the commonly used platforms for personal computing: Windows®, Mac OS X®, Linux, and some versions of UNIX®. In the usual desktop environments for these platforms, users will typically start R as they would most applications, by clicking on the R icon or on the R file in a folder of applications.

An application will then appear looking much like other applications on the platform: for example, a window and associated toolbar. In the

standard version, at least on most platforms, the application is called the `"R Console"`. In Windows recently it looked like this:



The application has a number of drop-down menus; some are typical of most applications (`"File"`, `"Edit"`, and `"Help"`). Others such as `"Packages"` are special to R. The real action in running R, however, is not with the menus but in the console window itself. Here the user is expected to type input to R in the form of expressions; the program underlying the application responds by doing some computation and if appropriate by displaying a version of the results for the user to look at (printed results normally in the same console window, graphics typically in another window).

This interaction between user and system continues, and constitutes an R session. The session is the fundamental user interface to R. The following section describes the logic behind it. A session has a simple model for user interaction, but one that is fundamentally different from users' most common experience with personal computers (in applications such as word processors, Web browsers, or audio/video systems). First-time users may feel abandoned, left to flounder on their own with little guidance about what to do and even less help when they do something wrong. More guidance is available than may be obvious, but such users are not entirely wrong in their

reaction. After intervening sections present the essential concepts involved in using R, Section 2.8, page 34 revisits this question.

2.2 An Interactive Session

Everything that you do interactively with R happens in a *session*. A session starts when you start up R, typically as described above. A session can also be started from other special interfaces or from a command shell (the original design), without changing the fundamental concept and with the basic appearance remaining as shown in this section and in the rest of the book. Some other interfaces arise in customizing the session, on page 17.

During an R session, you (the user) provide expressions for evaluation by R, for the purpose of doing any sort of computation, displaying results, and creating objects for further use. The session ends when you decide to quit from R.

All the expressions evaluated in the session are just that: general *expressions* in R's version of the S language. Documentation may mention "commands" in R, but the term just refers to a complete expression that you type interactively or otherwise hand to R for evaluation. There's only one language, used for either interactive data analysis or for programming, and described in section 2.3. Later sections in the book come back to examine it in more detail, especially in Chapter 3.

The R evaluator displays a prompt, and the user responds by typing a line of text. Printed output from the evaluation and other messages appear following the input line.

Examples in the book will be displayed in this form, with the default prompts preceding the user's input:

```
> quantile(Declination)
    0%    25%    50%    75%   100%
-27.98 -11.25   8.56  17.46  27.30
```

The `"> "` at the beginning of the example is the (default) prompt string. In this example the user responded with

```
quantile(Declination)
```

The evaluator will keep prompting until the input can be interpreted as a complete expression; if the user had left off the closing `")"`, the evaluator would have prompted for more input. Since the input here is a complete expression, the system evaluated it. To be pedantic, it parsed the input text

and evaluated the resulting object. The evaluation in this case amounts to calling a function named `quantile`.

The printed output may suggest a table, and that's intentional. But in fact nothing special happened; the standard action by the evaluator is to print the object that is the value of the expression. All evaluated expressions are objects; the printed output corresponds to the object; specifically, the form of printed output is determined by the kind of object, by its *class* (technically, through a method selected for that class). The call to `quantile()` returned a numeric vector, that is, an object of class `"numeric"`. A method was selected based on this class, and the method was called to print the result shown. The `quantile()` function expects a vector of numbers as its argument; with just this one argument it returns a numeric vector containing the minimum, maximum, median and quartiles.

The method for printing numeric vectors prints the values in the vector, five of them in this case. Numeric objects can optionally have a `names` attribute; if they do, the method prints the names as labels above the numbers. So the `"0%"` and so on are part of the object. The designer of the `quantile()` function helpfully chose a names attribute for the result that makes it easier to interpret when printed.

All these details are unimportant if you're just calling `quantile()` to summarize some data, but the important general concept is this: Objects are the center of computations in R, along with the function calls that create and use those objects. The duality of objects and function calls will recur in many of our discussions.

Computing with existing software hinges largely on using and creating objects, via the large number of available functions. Programming, that is, creating *new* software, starts with the simple creation of function objects. More ambitious projects often use a paradigm of creating new classes of objects, along with new or modified functions and methods that link the functions and classes. In all the details of programming, the fundamental duality of objects and functions remains an underlying concept.

Essentially all expressions are evaluated as function calls, but the language includes some forms that don't look like function calls. Included are the usual operators, such as arithmetic, discussed on page 21. Another useful operator is `` `?` ``, which looks up R help for the topic that follows the question mark. To learn about the function `quantile()`:

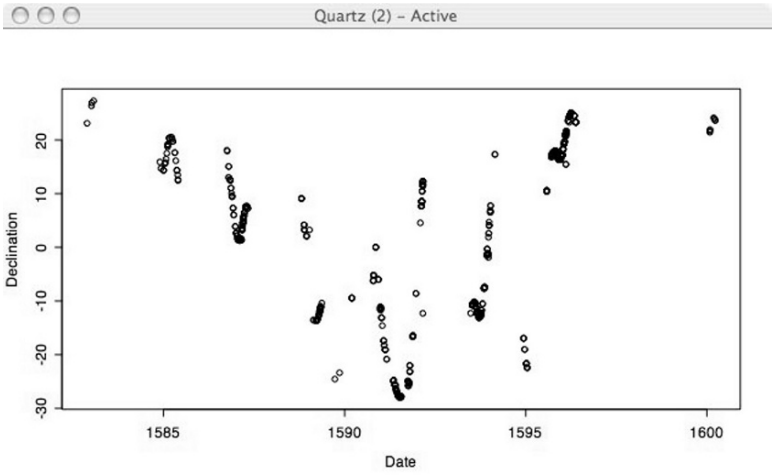
```
> ?quantile
```

In standard GUI interfaces, the documentation will appear in a separate window, and can be generated from a pull-down menu as well as from the

`?` operator.

Graphical displays provide some of the most powerful techniques in data analysis, and functions for data visualization and other graphics are an essential part of R:

```
> plot(Date, Declination)
```



Here the user typed another expression, `plot(Date, Declination)`; in this case producing a scatter plot as a side effect, but no printed output. The graphics during an interactive session typically appear in one or more separate windows created by the GUI, in this example a window using the native `quartz()` graphics device for Mac OS X. Graphic output can also be produced in a form suitable for inclusion in a document, such as output in a general file format (PDF or postscript, for example). Computations for graphics are discussed in more detail in Chapter 7.

The sequence of expression and evaluation shown in the examples is essentially all there is to an interactive session. The user supplies expressions and the system evaluates them, one after another. Expressions that produce simple summaries or plots are usually done to see something, either graphics or printed output. Aside from such immediate gratification, most expressions are there in order to assign objects, which can then be used in later computations:

```
> fitK <- gam(Kyphosis ~ s(Age, 4) + Number, family = binomial)
```

Evaluating this expression calls the function `gam()` and assigns the value of the call, associating that object with the name `fitK`. For the rest of the

session, unless some other assignment to this name is carried out, `fitK` can be used in any expression to refer to that object; for example, `coef(fitK)` would call a function to extract some coefficients from `fitK` (which is in this example a fitted model).

Assignments are a powerful and interesting part of the language. The basic idea is all we need for now, and is in any case the key concept: Assignment associates an object with a name. The term "associates" has a specific meaning here. Whenever any expression is evaluated, the context of the evaluation includes a local *environment*, and it is into this environment that the object is assigned, under the corresponding name. The object and name are associated in the environment, by the assignment operation. From then on, the name can be used as a *reference* to the object in the environment. When the assignment takes place at the "top level" (in an input expression in the session), the environment involved is the *global* environment. The global environment is part of the current session, and all objects assigned there remain available for further computations in the session.

Environments are an important part of programming with R. They are also tricky to deal with, because they behave differently from other objects. Discussion of environments continues in Section 2.4, page 24.

A session ends when the user quits from R, either by evaluating the expression `q()` or by some other mechanism provided by the user interface. Before ending the session, the system offers the user a chance to save all the objects in the global environment at the end of the session:

```
> q()
Save workspace image? [y/n/c]: y
```

If the user answers yes, then when a new session is started in the same working directory, the global environment will be restored. Technically, the environment is restored, not the session. Some actions you took in the session, such as attaching packages or using `options()`, may not be restored, if they don't correspond to objects in the global environment.

Unfortunately, your session may end involuntarily: the evaluator may be forced to terminate the session or some outside event may kill the process. R tries to save the workspace even when fatal errors occur in low-level C or Fortran computations, and such disasters should be rare in the core R computations and in well-tested packages. But to be truly safe, you should explicitly back up important results to a file if they will be difficult to recreate. See documentation for functions `save()` and `dump()` for suitable techniques.