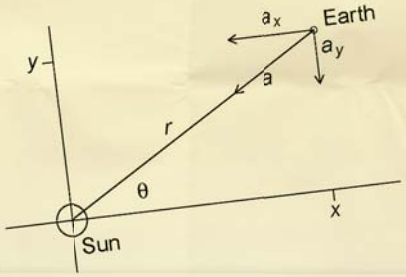
The R Student Companion

$$F = ma$$

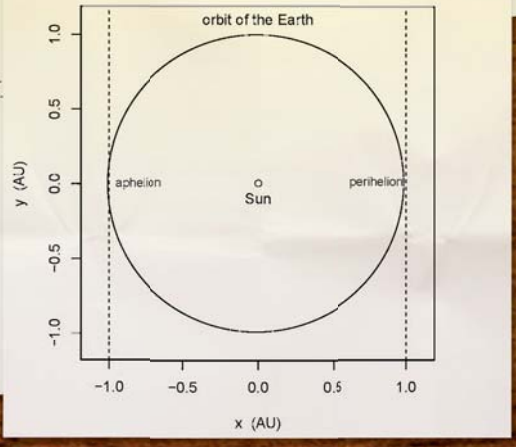
$$F_{Mm} = -G\frac{Mm}{r^2}$$

$$a_x = -\frac{GM_x}{(x^2+y^2)^{3/2}}$$

$$a_y = -\frac{GM_y}{(x^2+y^2)^{3/2}}$$




```
#-----------------------------------------------------------
# 6.  Insert the initial conditions into the vectors.
#-----------------------------------------------------------
x[1]=x0 # Initial value of x-position.
y[1]=y0 # Initial value of y-position.
v_x[1]=v_x0 # Initial value of x-direction velocity.
v_y[1]=v_y0 # Initial value of y-direction velocity.
t[1]=0 # Initial value of time.
c=GMsq # Pre-calculate a constant that appears repeatedly
       # in the equations.
#-----------------------------------------------------------
# 7. Loop to calculate trajectory.
#-----------------------------------------------------------
for (i in 1:(tot.int)) {
dvx=-c*x[i]*dt # Change in x.
dvy=-c*y[i]*dt # Change in y.
dx=v_x[i]*dt # New value of x.
dy=v_y[i]*dt # New value of y.
v_x[i+1]=v_x[i]+dvx # New value of x-velocity.
v_y[i+1]=v_y[i]+dvy # New value of y-velocity.
x[i+1]=x[i]+dx # New value of time.
y[i+1]=y[i]+dy
t[i+1]=t[i]+dt
}
#-----------------------------------------------------------
# 8. Plot the trajectory.
#-----------------------------------------------------------
par(pin=c(4,4))  # Equal screen size in x and y directions
plot(x,y,type="l",xlab="x (AU)",ylab="y (AU)",xlim=c(-1.1,1.1),
ylim=c(-1.1,1.1),
abline(v=0,lty=2),
abline(h=0,lty=2)
sx=0
sy=0
points(sx,sy,type="p")
text(0,-.1,"Sun")
text(0,1.1,"orbit of the Earth")
text(0.8,0,"perhelion",cex=.8)
text(-.8,0,"aphelion",cex=.8)
```



Brian Dennis

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

The R Student Companion

The R Student Companion

Brian Dennis

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20120803

International Standard Book Number-13: 978-1-4398-7541-4 (eBook - PDF)

For Chris, Ariel, Scott, and Ellen,

who make me so proud.

Contents

Preface

R is a computer package for scientific graphs and calculations. It is written and maintained by statisticians and scientists for scientists to use in their work. It is easy to use, yet is extraordinarily powerful. R is spreading rapidly throughout the science and technology world, and it is setting the standards for graphical data displays in science publications.

R is free. It is an open-source product that is easy to install on most computers. It is available for Windows, Mac, and Unix/Linux operating systems. One simply downloads and installs it from the R website (http://www.r-project.org/).

This book is for high school and college students, and anyone else who wants to learn to use R. With this book, you can put your computer to work in powerful fashion, in any subject that uses applied mathematics. In particular, physics, life sciences, chemistry, earth science, economics, engineering, and business involve much analysis, modeling, simulation, statistics, and graphing. These quantitative applications become remarkably straightforward and understandable when performed with R. Difficult concepts in mathematics and statistics become clear when illustrated with R.

The book starts from the beginning and assumes the reader has no computer programming background. The mathematical material in the book requires only a moderate amount of high school algebra.

R makes graphing calculators seem awkward and obsolete. Calculators are hard to learn, cumbersome to use for anything but tiny problems, and the graphs are small and have poor resolution. Calculating in R by comparison is intuitive, even fun. Fantastic, publication-quality graphs of data, equations, or both can be produced with little effort. High school and college courses in science and mathematics that currently rely on graphing calculators could benefit greatly from using R instead.

This book will introduce enough R to allow the reader to provide sophisticated solutions to quantitative problems in the sciences and social sciences. But R is huge, and this book is not meant as a comprehensive guide to R. Much of the myth that R has a "steep learning curve" arises from the dizzying complexity of introductory manuals. To become a proficient R user from scratch, one needs a place to begin!

This book will help the reader get started in R. The idea of the book is that one should start by building a basic set of R skills that are learned well. The skill set will handle the quantitative problems arising in most high school and college science courses as well as serve as a base for exploring more advanced material. Part of the fun and rewards of R is discovering for oneself the immense resources available in R.

It is the author's experience that students who have trouble learning R often are actually having more trouble with the underlying mathematical concepts behind the analysis. This book assumes only that the reader has had some high school algebra. Several of the chapters explore concepts from algebra that are highly useful in scientific applications, such as quadratic equations, systems of linear equations, trigonometric functions, and exponential functions. Each chapter provides an instructional review of the algebra concept, followed by a hands-on guide to performing calculations and graphing in R. The chapters describe real-world examples, often drawn from the original scientific publications, and the chapters then show how the scientific results can be reproduced with R. R puts contemporary, cutting-edge quantitative science within reach of high school and college students.

R has a well-deserved reputation as a leading software product for statistical analysis. However, R goes way beyond statistics. It is a comprehensive software package for scientific computations of all sorts, with many high-level mathematical, graphical, and simulation tools built in. Although the book covers some basic statistical methods, it focuses on the broader aspects of R as an all-round scientific calculation and graphing tool.

Another part of the mythical difficulty of learning R stems from the problem that many of the books and web sites currently available about R are also about statistics and data analysis. This book, however, is prestatistics and largely avoids the prepackaged statistical routines available in R. The concepts of statistical inference are challenging. The book introduces some computational probability and simulation, some summary statistics and data graphs, and some curve fitting, but the book does not dive into statistical inference concepts. In fact, students who have the R background contained in this book are positioned to get far more out of their initial exposure to statistical inference.

This book does *not* assume that the reader has had statistics or calculus. Relatively few students take calculus before college, and even fewer take statistics before their middle years in college. Instead, this book concentrates on the many uses of R in precalculus, prestatistics courses in sciences and mathematics. Anything in science, mathematics, and other quantitative courses for which a calculator is used is better performed in R. Moreover, R greatly expands the complexity of the scientific examples that can be tackled by students.

Students who use R in their science courses reap great benefits. With R, scientific calculations and graphs are fun and easy to produce. A student using R is freed to focus on the scientific and mathematical concepts without having to pore through a manual of daunting lists of calculator keystroke instructions. Those calculator instructions are never actually mastered and internalized, and they change with each new machine. R skills by contrast are mastered and grow with each use, and they follow the student on into more advanced courses. The students will be analyzing data and depicting equations just as scientists are doing in laboratories all over the world.

R invites collaboration. Like the scientists sharing their applications, students can work in groups to conduct projects in R, build R scripts, and improve each others' work, and they can collect, accumulate, and just plain show off exemplary graphical analyses. Results on a computer screen are much easier to view in groups than on a calculator, and R scripts are much easier to save and alter cooperatively than are the calculator keystroke lists. At home, students can message each other about their latest R scripts, working cooperatively online. Every new class can take what previous classes have done and build new accomplishments upon the old. *R builds on itself.*

R use has exploded in colleges and universities, and R has recently been adopted by forward-looking technology companies (see "Data Analysts Captivated by R's Power," by Ashlee Vance, *The New York Times*, January 6, 2009). A simple online search will reveal the extent to which R is shaping education in advanced university courses. Not only do R skills follow a student throughout their coursework, but knowledge of R is also a bona fide professional scientific credential of substantial value.

Online web resources for R use are vast. Scientists and professors have leaped at the opportunity to share their accumulating knowledge about using R in scientific applications. Free web-based tutorials, primers, and reference books are posted on scientists' and courses' web pages everywhere. Discussion forums about R exist, where people can get questions answered and share R "scripts" for reproducing the calculations and graphics from the latest scientific paper. These resources, however, tend to be targeted at more advanced and specialized college level courses. For the beginner, the sheer preponderance of material makes it hard to know where to start. Most R users have had to struggle initially with the need to cope with and filter the disorganized cloud of R instructional material.

This book is designed with the beginner in mind. With this book, the student can master an initial set of R techniques that will handle most computation and graphing projects in basic science and applied mathematics courses. The techniques are here in one place, easy to learn and easy to find. The student can then graduate to the more advanced techniques with confidence.

How to use this book: This book is meant to be *done*, not read! First, install R on your computer or locate an institutional computer on which R has been installed. Then, start at the beginning of the book and follow along and type the R commands as you go. Save your work in a folder on the computer hard drive or on portable storage media. At the end of each chapter, tackle one or more of the computational challenges. These are more than just exercises; they are miniprojects that will help you build R-thinking into your creativity and problem-solving. In a course setting, the computational challenges can be parceled out among students or groups of students, and their results can be shared with or reported to the class as a whole. The appendices at the end of the book provide additional advice about installation of R, information about getting help with R techniques, and a handy listing of the most commonly used R commands and options.

All the scripts presented in this book, all the scripts which produced each figure in the book, and all the data sets in this book are posted at http://webpages.uidaho.edu/~brian/rsc/RStudentCompanion.html.

Readin', Ritin', Rithmetic … and R!

Brian Dennis
Moscow, Idaho

Author

Brian Dennis is a professor with a joint appointment in the Department of Fish and Wildlife Sciences and the Department of Statistical Sciences at the University of Idaho. He received his master's degree in statistics and his PhD in ecology from the Pennsylvania State University. He has authored more than 70 scientific articles on applications of statistics and mathematical modeling in ecology and natural resource management. He has enthusiastically used R in his work and teaching R in his courses for a decade.

1

Introduction: Getting Started with R

R is a computer program for doing scientific graphs and calculations.

R was written by scientists, for scientists to use in their work.

R is incredibly powerful and amazingly easy to use.

R is free.

Did I mention that R is free?

Versions of R are available for Windows computers, Macs, and even Unix or Linux. If you have a computer at home, you can download and install R from the web site http://www.r-project.org/.

Installation is easier than installing a computer game (but if you need help, refer to Appendix A).

Installing R will put an icon, a blue "R," on the computer desktop or in the program list. Find it and click it to start the program. The R program window will appear, and the "R console," a window within the R program window, will pop up.

On the console, you will see the prompt ">" followed by a cursor. Now, R is waiting for your instructions! Just type the commands that appear at the prompt as you go through this tutorial, or follow along as your instructor demonstrates the commands on a projector. Afterward, work alone or in groups to answer the computational challenges at the end of this chapter. Be prepared to give a short presentation of your results to the class. Ready? Let's get started.

R Tutorial

The simplest way of using R is as a powerful calculator. At the prompt, type: `5+7` and hit the `Enter` key:

```
> 5+7
[1] 12
```

Part 1 of the answer is 12. We will later see that answers can often have many parts, and so R prints part numbers along with the answers.

Let us try subtraction. Each time, type the characters after the prompt and hit the `Enter` key; the answer you should see is then printed in this tutorial on the next line:

```
> 5-7
[1] -2
```

R knows about negative numbers. Let us try:

```
> 5+-2
[1] 3
```

Now, multiplication is represented by an asterisk "`*`":

```
> 5*7
[1] 35
```

Division is indicated by a forward slash "/", so 5 divided by 7 is

```
> 5/7
[1] 0.7142857
```

R will do calculations as decimal numbers, just like a calculator.

Raising to a power: Recall that "five to the seven power," written as 5^7, means $5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5$. R will calculate that for us. The caret symbol "`^`" denotes raising to a power. Thus, five to the seven power is

```
> 5^7
[1] 78125
```

You can put a string of calculations all in one command. The calculations with `*` and `/` are done first, then with + and –, and the calculations are done from left to right:

```
> 5+7*3-12/4-6
[1] 17
```

See if you can get the above answer by hand. Also, raising to a power is done first, even before multiplication and division:

```
> 1+4*3^2
[1] 37
```

You can alter the priority of operations by using parentheses:

```
> (5+7)*3-(12/4-6)
[1] 39
```

Parentheses inside of parentheses will be done first! Just ensure that every left parenthesis "`(`" has a right friend "`)`":

```
> (5+7)*3-(12/(4-6))
[1] 42
```

R will store *everything* for you. You just give names to your calculations:

```
> sally=5+7
> ralph=4-2
> sally-ralph
[1] 10
```

When giving names, remember that R notices lowercase and uppercase characters. In R, the name `sally` is different from `Sally`.

Are `ralph` and `sally` still there?

```
> sally
[1] 12
> ralph
[1] 2
```

They will disappear when you exit the R program without saving the "workspace."

If you use the name for something different, R will erase the old value:

```
> ralph=9
> ralph
[1] 9
```

Interestingly, the symbol "=" in R (and in many computer programming languages) does not mean "equals." Instead, it means

> calculate what is on the right, and store the result using the name on the left.

This was designed by computer scientists seemingly to irritate their former math teachers. For example, `ralph=ralph+1` is a statement that mathematicians do not like, because no number exists that, when you add one to it, you get the same number! However, the statement makes perfect sense to computer programmers. It means

> take the old value of ralph, add 1, and store the result as the new value of ralph.

A statement with an equals sign is called an "assignment statement," assigning the value resulting from the calculation on the right to the storage location on the left. Try it:

```
> ralph=ralph+1
> ralph
[1] 10
```

Actually, previous versions of R used the syntax `ralph<-ralph+1` for assignment statements (the word "syntax" means typographic rule). The symbol "<-" (a "less than" symbol followed by a hyphen) is supposed to look

like a little arrow pointing left. Many older web sites and books about R use this syntax, and in fact the syntax still works in the current versions of R. Let us try it:

```
> sally<-sally+ralph
> sally
[1] 22
```

The assignment statement calculated and stored 22 as the new value of `sally`. The scientists responsible for R finally gave up trying to be mathematical purists and instituted the equals sign for assignment statements in order to be consistent with most other computer languages.

Now, we are ready to unleash some of the power of R!

Vectors

R can work with whole "lists" of numbers. Try this:

```
> x=c(3,-2,4,7,5,-1,0)
> y=4
> x+y
[1]  7  2  8 11  9  3  4
```

The `c()` command in the first line above says "combine" the numbers 3, –2, 4, 7, 5, –1, and 0 into a list. We named the list `x`. R has a special term for a list of numbers: a **vector**. Here, `x` is a vector with seven **elements**. The value of `y` is 4. The expression `x+y` added 4 to every value in `x`! But what if `y` were a vector like `x`?

```
> y=c(1,2,3,4,5,6,7)
> z=x+y
> z
[1]  4  0  7 11 10  5  7
```

Each number in the vector `y` is added to the corresponding number in `x`!

Remember in fourth grade when your teacher gave you a bunch of big multiplications to do for homework:

75,634	2,339	103,458	48,761	628,003
× 567	× 138	× 974	× 876	× 402

Put the top numbers in a vector (let us name it "`top`"), and the bottom numbers in another vector (say, named "`bot`"). Then, multiply the vectors:

```
> top=c(75634,2339,103458,48761,628003)
> bot=c(567,138,974,856,402)
> top*bot
[1]  42884478   322782    100768092   41739416   252457206
```

There are a few things to note here. (1) When writing R statements, do not use commas within large numbers to group the digits in threes. Rather, commas are used in R for other things, such as to separate numbers in the `c()` (combine) command. (2) Enter the numbers in the two vectors in the same order. (3) Spaces in between the numbers are fine, as long as the commas are there to separate them. (4) Do not show this to your younger sibling in fourth grade.

All of the arithmetic operations, addition, subtraction, multiplication, division, and even power, can be done in R with vectors. We have seen that if you operate with a single number and a vector, then the single number operates on each element in the vector. If you operate with two vectors of the same length, then every element of the first vector operates on the corresponding element of the second vector.

The priority of operations is the same for vector arithmetic, and parentheses may be used in the usual way to indicate which calculations to perform first:

```
> ted=c(1,2,3)
> kat=c(-1,1,.5)
> 2*(ted+kat)
[1]  0  6  7
> 2*ted+kat
[1]  1  5  6.5
```

If you make a mistake while typing, just type the line again. R will calculate and store the newer version. Also, if a line is long, you can continue it on the next line by hitting the `Enter` key *at a place where the R command is obviously incomplete* (R is remarkably smart!). R will respond with a different prompt that looks like a plus sign; just continue the R command at the new prompt and hit the `Enter` key when the command is complete:

```
> kat=c(-1,1,
+ .5)
> kat
[1] -1.0  1.0  0.5
```

A special vector can be built with a colon ":" in the following way:

```
> j=0:10
> j
 [1]  0  1  2  3  4  5  6  7  8  9 10
```

Here, `j` was defined as a vector consisting of all the integers from 0 to 10. One can go backward if one wants:

```
> k=5:-5
> k
 [1]  5  4  3  2  1  0 -1 -2 -3 -4 -5
```

Do you want to see the powers of 2 from 2^0 to 2^{20}? Of course you do:

```
> j=0:20
> 2^j
[1]     1     2     4     8    16    32    64   128
[9]   256   512  1024  2048  4096  8192 16384 32768
[17] 65536 131072 262144 524288 1048576
```

You should note here that the text syntax in R for writing math expressions forms a completely unambiguous way of communicating about math homework problems via instant messaging or text messaging:

Ralph: `hey, what's up?`

Sally: `working on math homework. Yuck.`

Ralph: `oh darn, yeah, and I 4got to write down all that "solving a quadratic" stuff.`

Sally: `here it is…`

Sally: `a*x^2 + b*x + c = 0.`

Sally: `there are two solutions provided b^2 > 4*a*c.`

Sally: `(-b+sqrt(b^2-4*a*c))/(2*a).`

Sally: `(-b-sqrt(b^2-4*a*c))/(2*a).`

Ralph: `thx! R U using that R thing to do the homework?`

Sally: `of course! Using a calculator would take twice as long.`

Ralph: `want to meet at the coffee shop after ur done?`

Sally: `sure! I am almost finished, thx to R.`

Sally and Ralph are experienced R users and know that `sqrt()` takes the square root of whatever is inside the parentheses. We will look at that and other functions in Chapter 3.

Graphs

Are you ready for a graph? If you are not impressed yet by R, prepare to be so. Suppose you have accumulated $1000 and would like to save it for future use, perhaps for buying a house. Suppose you find a bank that offers a certificate of deposit (CD) paying interest of 5% per year, which will be reinvested in the CD. Such a CD would be a great opportunity to put your money to work for you.

Let us see why, by drawing a graph in R. The graph will show the amount of money in the CD after year 1, year 2, and so on, up to, say, year 10.

For interest of 5% compounded annually, we multiply the amount of money in the CD each year by $(1+0.05)$ in order to calculate how much money is in the CD at the end of the following year. So, we calculate the amount of money after year 1 by $1000(1+0.05)$. We calculate the year 2 amount by $1000(1+0.05)(1+0.05)$, the year 3 amount by $1000(1+0.05)(1+0.05)(1+0.05)$, and so on. See the pattern? We can represent the money after year t as an equation, based on the pattern. If n is the number of dollars in the CD after year t, then the equation is

$$n = 1000(1+0.05)^t.$$

Drawing a picture of the changing number of dollars through time according to this equation is a ready-made task for R. (1) We will set up a vector called "`t`" that will contain the years 0, 1, 2, 3, …, 10. (2) We will calculate a vector called "`n`" using the equation; `n` will contain all the dollar amounts in the CD corresponding to the years 0, 1, 2, 3, …, 10. (3) We will draw an x – y graph of the values of `n` (vertical axis) versus the values of `t` (horizontal axis), connecting the points with lines. Type in the following R commands (be sure to note that "`l`" in the third command is a lowercase "`L`," not the numeral "one"):

```
> t=0:10
> n=1000*(1+0.05)^t
> plot(t,n,type="l")
```

A separate graph window should pop up on your screen as shown in Figure 1.1. We used only three commands! Pretty cool, huh? The `plot()` command is a built-in routine in R for two-dimensional plots. There are many such graphical routines in R for many different kinds of graphical displays. Most of them can be customized to accommodate virtually anything that one might want to include in a display, such as different axis labels, tic marks, titles, symbols, and symbol legends.

In the `plot()` command, the horizontal axis variable is listed first, followed by the vertical axis variable and various optional statements setting the appearance of the graph. Here, the `type="l"` option (`l` stands for "line") specifies the type of plot to be a line plot with points connected by lines, but no symbols drawn for the points themselves. R automatically picks appropriate axis sizes, but they can be altered with additional options in the plot statement, along with axis labels, tic marks, title, line thicknesses, and so on. Appendix C lists the different types of plots available, and you will use many of them as you work through the chapters in this book. The entries or "arguments" in the `plot()` command are separated by commas, the standard syntax for the arguments and options entered in built-in routines in R.

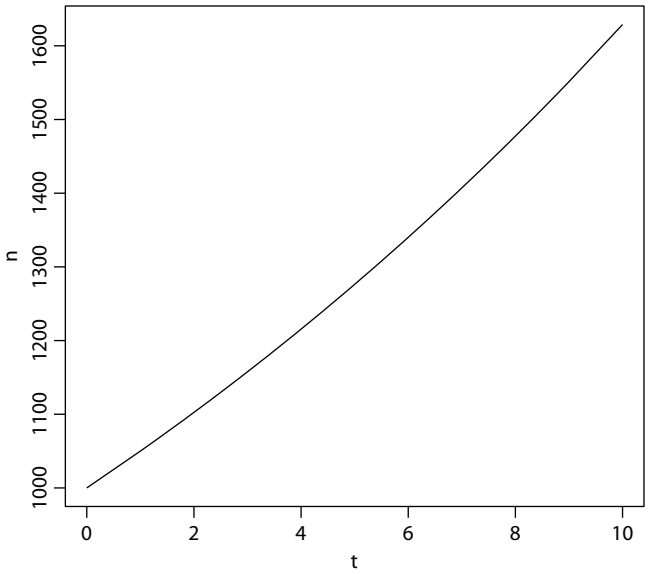


FIGURE 1.1
Amount of dollars n after year t in a certificate of deposit paying 5% interest compounded annually, with an initial investment of $1000.00.

The graph is a graphical object that can be saved as a file in various graphic formats. Click on the graph window to make it active, then look in "File" on the top menu bar, and select "Save As." Good graphical formats for such scientific plots are EPS or PDF. From the File menu, you can alternatively copy the graph to the clipboard and subsequently paste the graph into a word processor document.

The graphical object is actually "open" in R, waiting for you to add further points, curves, annotations, and so on. You will learn many such customizations in subsequent chapters.

When you are done with the graph and are ready to make another, close the graph window.

Real-World Example

Although it is hard to exceed the real-world importance of money, we chose the example of CD investment mentioned earlier mainly for its simplicity. Many chapters of this book will close by tackling more complex, real-world examples that will require putting R calculations and graphics together in a cumulative and comprehensive way.

Let us try a graph of some data from ecology. This is an example from real science, not a "toy" example, and so it requires a bit of explaining.

Ecology is a subfield of biology that deals with the study of the relationships between organisms and their environments. Predators and their prey is a topic that has excited ecologists for many decades, and the topic is important to society. For instance, questions about wolf predation on deer, elk, moose, and livestock have become politically controversial in some parts of the United States. The reintroductions of wolves in locales where they once were exterminated as pests have the potential to adversely affect hunting and livestock production.

How many moose do wolves eat in the wild? Well, we might expect that the average number of moose killed by an average wolf would depend on the supply of moose! Getting some idea about the form of the relationship between wolf feeding rate and moose supply would be helpful to wildlife managers who must make decisions about wolf culling and moose hunting over a wide range of moose and wolf abundances.

In the table below are some figures that an ecologist assembled together from studies across wolf ranges in North America (Messier 1994). In the regions studied, moose were the preferred food of the wolves and occurred in varying densities. Wolves live and hunt mostly in packs, and the moose kill rates are calculated as the average number of moose killed per wolf per 100 days:

moose density	kill rate
0.17	0.37
0.23	0.47
0.23	1.90
0.26	2.04
0.37	1.12
0.42	1.74
0.66	2.78
0.80	1.85
1.11	1.88
1.30	1.96
1.37	1.80
1.41	2.44
1.73	2.81
2.49	3.75

Here, "moose density" is the average number of moose per 1000 km^2. One thousand square kilometers is an area roughly equivalent to a square of 20 miles by 20 miles. A glance at the numbers suggests that moose—and wolves—use a lot of land: one moose or so per thousand square kilometers means that wolves must range far and wide for a square meal! But the numbers by themselves are just a stark table and do not convey much more to us beyond their range of values. Instead, a visual portrayal of the numbers will help reveal any relationships that might exist.

To explore these data, we will simply plot each pair of numbers (moose density and kill rate) as a point on an x–y graph. Scientists call that type of graph a scatterplot. We need two vectors containing the values to be plotted. Let us call them "`moose.density`" and "`kill.rate`." Scientists find it helpful to give descriptive names to the quantities and objects in R calculations, and when choosing names, they will often string together several words connected by periods. After setting up the vectors (and checking the numbers carefully), we then just add the `plot()` command, only this time using the `type="p"` option:

```
> moose.density=c(.17,.23,.23,.26,.37,.42,.66,.80,1.11,1.30,1.37,
+ 1.41,1.73,2.49)
> kill.rate=c(.37,.47,1.90,2.04,1.12,1.74,2.78,1.85,1.88,1.96,
+ 1.80,2.44,2.81,3.75)
> plot(moose.density,kill.rate,type="p")
```

You should now see a scatterplot of the points! The `type="p"` option in the `plot()` command produces the scatterplot (`p` stands for "points"), with symbols drawn for the points without connecting the points with lines. The first two commands for putting the data into the vectors used continuation lines ("+" prompt) in order to fit the commands compactly within the typesetting of this book, but most R consoles will accept very long commands in one line.

Let us add something to our graph! How about a mathematical curve summarizing ecologists' current understanding about what the relationship should look like? We will then have scientific hypothesis and real-world data compared on one graph.

Before adding such a curve, we will try to make some sense about what we see on the graph. The data are scattered, "noisy" as scientists say, but there is a sort of pattern. As moose density increases, so does the kill rate, but then the kill rate seems to flatten and does not continue increasing as fast.

Ecologists have seen this pattern in many predator–prey systems, and the current hypothesis for the cause of the pattern goes something like this. If moose are rare, a given wolf will likely consume very few moose in a fixed period of time, so the data point will be near zero for both axes. If the supply of moose increases, we can hypothesize that the number of moose consumed per wolf would also increase.

But what if moose were very abundant, as if nature staged an "all-you-can-eat" banquet for wolves? We would not expect the average wolf's rate of consumption of moose to increase without bound, because the physical capacity for killing, handling, and digesting moose is limited. Instead, we might expect that as the supply of moose increases from abundant to very abundant, the number of moose consumed by an average wolf in the given period of time would simply level off. There is an upper physical limit to the speed with which wolves can hunt and eat moose, just as there is an upper limit to the speed with which you can eat hamburgers (file that under biology lab exercises you would love to try).

To summarize, we conjecture that the relationship between the feeding rate of an average wolf and the supply of moose, if plotted on an x–y graph (with moose supply as the variable on the horizontal axis) would start near zero, rise steeply at first, but then gradually turn toward horizontal at some upper maximum feeding rate.

Ecologists have expressed this hypothesis in a mathematical model, an equation that captures the essential expected relationship under "ideal" circumstances, that is, only the moose supply is being varied, with any other environmental variables that could affect the situation being held fixed. The equation is as follows:

$$k = \frac{am}{b+m},$$

where k is the kill rate of an average predator, m is the supply of prey, and a and b are constants that have different values for each type of predator and each type of prey (lady beetles eating aphids have different values for a and b than do wolves eating moose). For the wolf–moose data, the scientist obtained the following values for a and b by using a sophisticated "curve-fitting" program (like the one you will learn in Chapter 15):

$$a = 3.37,$$

$$b = 0.47.$$

Let us add a plot of the equation to our graph! For this, we need two more vectors. One will contain a range of moose densities (let us call it `m`). The other will contain the resulting kill rates calculated using the equation (we will call it `k`). We also need to define the quantities `a` and `b` before the equation can be computed. To obtain a nice smooth curve, we should use many values of moose density (a hundred or so), ranging from 0 to, say, 2.5 (the upper end of the moose axis on our graph). The expression `(0:100)/100` will produce a vector with a low value of 0 and a high value of 1, with many values in between (check this in the console!). We just multiply that vector by 2.5 to get the vector with a range of values of moose supply from 0 to 2.5. Leave the scatterplot "open" (do not close the graph window), and click on the R console window to make it active. Carefully type the following:

```
> m=2.5*(0:100)/100
> a=3.37
> b=0.47
> k=b*m/(a+m)
```

These commands build the two vectors, `m` and `k`, to add to our graph. To put them on our graph in the form of a curve, just type one more statement:

```
> points(m,k,type="p")
```

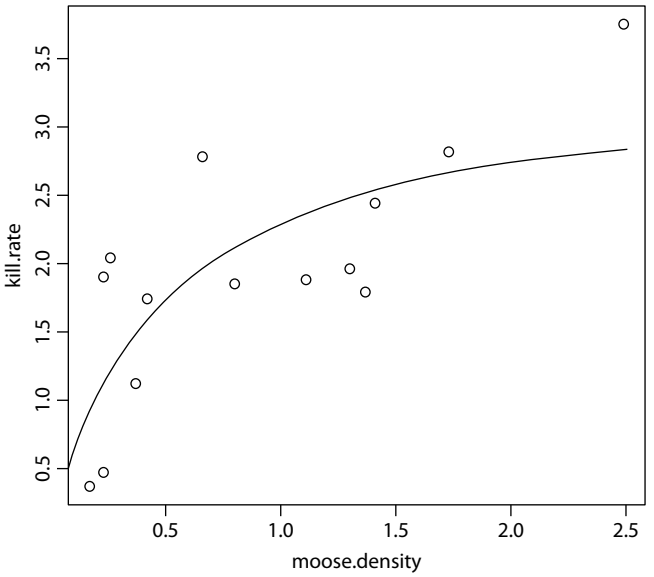


FIGURE 1.2
Data and model depicting the relationship between the wolf rate of feeding on moose and the density of moose. Circles: number of moose killed per wolf per 100 days (vertical axis) with the number of moose per 1000 km^2 (horizontal axis). Solid line: the "functional response" equation from predator–prey theory (Gotelli 2008) fitted to the data. (Data from Messier, F., *Ecology*, 75, 478–488, 1994.)

The `points()` command adds extra points onto an open graph. Its arguments and options work mostly like those in the `plot()` command.

The graph you should be seeing in your R window appears in Figure 1.2. You have essentially reproduced a figure in the original scientific journal article! Save the graph, and save the R commands (How might you do this? Try some ideas!) for future reference.

Final Remarks

It might seem that typing more than a few lines in the command console for a longer, more complex calculation or a complicated graph could become unwieldy. That certainly is true. In Chapter 2, you will find out how to enter, edit, and save a long list of commands into a special R file called a "script," and run the entire list all at once. It might also seem that typing a large data set into one or more vectors using the `c()` (combine) command is awkward and inconvenient. In Chapter 5, you will find out how to enter and store data in a separate data file and then how to bring the data file into R for plotting and analysis. Be assured that the writers of R know that most people, especially scientists, hate unnecessary work.

WHAT WE LEARNED

1. Priority of operations in arithmetic statements in R: raising to a power (^), multiplication and division (* and /), addition and subtraction (+ and -), operations performed from left to right, and priorities overridden by parentheses.

   Example:

```
> 3*(12-5)^2+4-6/2*2
[1] 145
```

2. Store the results in computer memory using assignment statements. Assignment statements use "=" or "<-" symbols, calculating what is on the right of the symbol and storing the result under the name on the left. Names are case sensitive. Type the name to retrieve the quantity.

   Example:

```
> Daily.run=2
> Cumulative.run=20
> Cumulative.run=Cumulative.run+Daily.run
> Cumulative.run
[1] 22
```

3. Vectors are ordered lists of numbers. The combine command `c()` is a handy way of creating small vectors. Arithmetic operations between a single number and a vector are performed using the single number and each element of the vector, using the same priority of operations as for ordinary arithmetic statements. Arithmetic operations for two vectors of the same size are performed on the corresponding elements of the two vectors, using the same priority of operations as for ordinary arithmetic statements. The colon command `:` provides a handy way of creating a vector containing a sequence of numbers.

   Example:

```
> time.long=c(0,1,2,3,4,5,6,7,8,9,10);
> time.long
[1]  0  1  2  3  4  5  6  7  8  9 10
> time.quick=0:10
> time.quick
[1]  0  1  2  3  4  5  6  7  8  9 10
> time.long+time.quick
[1]  0  2  4  6  8 10 12 14 16 18 20
```

```
> distance=16*time.long^2
> distance
[1]    0   16   64  144  256  400  576  784 1024 1296 1600
```

4. The `plot()` command produces *x*–*y* plots of two vectors. The horizontal axis variable is listed first in the command arguments. Command arguments are separated by commas. Different types of plot styles are obtained with the `type=` argument, with simple line plots (points connected by lines) designated by `type="l"` and scatterplot (points plotted with symbols but not connected) designated by `type="p"`. Plots can be saved as files under various graphical formats or copied to the clipboard. The `points()` command adds more points to an open plot and uses the same syntax as the original `plot()` command.

   Example:

```
> time=(0:10)/10
> distance=16*time^2
> plot(time,distance,type="l")
> lab.data=c(.3,1.4,1.5,1.9,4.8,5.3,7.9,9.8,13.7,15.7)
> lab.time=(1:10)/10
> points(lab.time,lab.data,type="p")
```

Computational Challenges

The computational problems that follow can be accomplished with the R techniques you have learned so far. Expect some snags in your early attempts. You will get the hang of R with practice, trial and error, and consultation with classmates. Remember, if you mistype a command, just type it again, and R will overwrite the previous value. Copy and save your successful commands and results into a word processor file for future reference.

1.1. Evaluate the following expressions:

$$\frac{93^2-164}{46^3+189} \qquad 376-\frac{23^2}{4} \qquad \frac{59+48^2}{-9+22^2}-\frac{-16+55^2}{13+29^2} \qquad 18^4-16^3+14^2-12$$

$$3^x \text{ for } x=1,2,....,20 \qquad 5^x \text{ for } x=1,2,....,10$$

1.2. Draw the investment equation (money in the CD) again, except use a much longer time horizon. Allow time to range as many years into the

future (50?) as, say, you have until you reach the retirement age of 65. Gaze at the graph with awe. Yes, it is really your choice: having those fancy designer jeans now or having many times the sale price later!

Add two or three more curves onto your graph showing the effects of two or three different interest rates.

1.3. The following are the population sizes of the United States through its entire history, according to the U.S. Census. Construct a line plot (`type="l"`) of the U.S. population (vertical axis) versus time (horizontal axis). By the way, rounding the population sizes to the nearest 100,000 or so will hardly affect the appearance of the graph.

1790	3,929,214
1800	5,308,483
1810	7,239,881
1820	9,638,453
1830	12,860,702
1840	17,063,353
1850	23,191,876
1860	31,443,321
1870	38,558,371
1880	50,189,209
1890	62,979,766
1900	76,212,168
1910	92,228,496
1920	106,021,537
1930	123,202,624
1940	142,164,569
1950	161,325,798
1960	189,323,175
1970	213,302,031
1980	236,542,199
1990	258,709,873
2000	291,421,906

Repeat the plot six more times (saving the graph each time), using `type="p"`, `type="b"`, `type="c"`, `type="o"`, `type="h"`, and `type="l"`. Compare the different graph types. What different aspects of the data are emphasized by the different graph types?

1.4. If you throw a baseball at an angle of 45°, at an initial velocity of 75 mph, while standing on a level field, the ball's horizontal distance x traveled after t seconds is described (neglecting air resistance) by the following equation from Newtonian physics:

$$x = 27.12t.$$

Furthermore, the height above the ground after t seconds, assuming the ball was initially released at a height of 5 ft, is described by

$$y = 1.524 + 19.71t - 4.905t^2.$$

The equations have been calibrated to give the distance x and height y in meters. The ball will hit the ground after about 4.09 seconds. Calculate a vector (say, `x`) of baseball distances for a range of values of t from 0 to 4.09. Calculate a vector of baseball heights (say, `y`) for the same collection of times. Make a plot of `x` (horizontal axis) and `y` (vertical axis). Read from the graph of the ball's trajectory how high and how far, approximately, the ball will travel.

NOTE: For different initial throwing velocities and angles, the above baseball equations will have different numerical coefficients in them. The equations can be written in more general form to accept different initial conditions, but to do that, we need a little bit of trigonometry (Chapter 9).

1.5. According to Newton's universal law of gravitation, the acceleration of an object in the direction of the sun due to the sun's gravity can be written in the form

$$a = \frac{1}{r^2},$$

where r is the distance of the object from the sun's center, in astronomical units (AU) of distance. One AU is the average distance of the Earth from the sun, about 150 million kilometers. The units of a are scaled for convenience in this version of Newton's equation so that one unit of acceleration is experienced at a distance of 1 AU. Use the equation to calculate the gravitational accelerations at each of the planets' average distances from the sun:

Planet	Mercury	Venus	Mars	Jupiter	Saturn	Uranus	Neptune	(Pluto)
Distance	0.39	0.723	1.524	5.203	9.539	19.18	30.06	39.53

Pluto is now considered to be a large comet-like object or dwarf planet that originated from the Kuiper Belt.

1.6. Using the equation from Question 1.5, draw a graph of the gravitational acceleration a (vertical axis) versus a range of values of r ranging from around 0.4 AU (distance of Mercury) to around 5.2 AU (distance of Jupiter). According to Newton's gravitational law, is there any distance at which the sun's gravity is escaped entirely?

1.7. Using the kill rate equation for the wolf–moose system that you studied in the tutorial, do some numerical exploration/experimentation to find out where the curve levels off (the maximum kill rate of the average wolf). How is it possible that some of the real data had kill rate values above that

maximum? Does the leveling off point resemble any numerical quantity you see in the equation itself?

1.8. To decrease the use of insecticides in agriculture, predator insects are often released to combat insect pests. Coccinellids (lady beetles), in particular, have a voracious appetite for aphids. In a recent study (Pervez and Omkar 2005), entomologists looked at the suitability of using coccinellids to control a particular aphid, *Myzus persicae* (common name is the "green peach aphid"), a serious pest of many fruit and vegetable crops. In the study, the entomologists experimentally ascertained aphid kill rates for three different species of coccinellids:

	COCCINELLID FEEDING RATE (# eaten per 24 hr)		
APHID DENSITY (#/jar)	*Cheilomenes sexmaculata*	*Coccinella transversalis*	*Propylea dissecta*
25	21	21	15
50	37	37	26
100	65	60	42
200	102	90	59
300	125	109	69
400	141	120	74
500	154	129	79
600	164	136	82
700	170	140	83
800	177	143	85

Enter the data columns above into vectors, giving them descriptive names. For each type of coccinellid, use R to construct a scatterplot (`type="p"`) of the feeding rate of the coccinellid versus aphid density. Then, add a kill rate curve to the coccinelid/aphid graph. Use the following constants in the kill rate equations:

$$C.\ sexmaculata:\ a = 234.5,\ b = 261.9,$$
$$C.\ transversalis:\ a = 178.9,\ b = 194.9,$$
$$P.\ dissecta:\ a = 100.4,\ b = 139.8.$$

Save and close each graph before starting the graph for the next coccinellid.

1.9. Plot the moose–wolf data again in a scatterplot, and save the graph under all the different graphical file formats (.JPG, .EPS, .PNG, etc.) available. Import each version into a word processor or presentation program so that you can compare the graphical formats side by side. Do some Internet research and find out the main advantages and disadvantages of each of the available formats. List these advantages and disadvantages in your document or presentation, in conjunction with the example you made of a scatterplot in each format. Share your discoveries!