A

A Brief Introduction to R

R is a language. Use it every day, and you will learn it quickly.

§, the precursor to R, is a quantitative programming environment developed at AT&T Bell Labs in the 1970s. S-Plus is a commercial, "value-added" version and was begun in the 1980s, and R was begun in the 1990s by Robert Gentleman and Ross Ihaka of the Statistics Department of the University of Auckland. Nearly 20 senior statisticians provide the core development group of the R language, including the primary developer of the original § language, John Chambers, of Bell Labs.

R is an official part of the Free Software Foundation's GNU project[1] (http://www.fsf.org/). It is free to all, and licensed to stay that way.

R is a language and environment for dynamical and statistical computing and graphics. R is similar to the S language, different implementation of §. Technically speaking, R is a "dialect" of §. R provides a very wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. R has become the lingua franca of academic statistics, and is very useful for a wide variety of computational fields, such as theoretical ecology.

A.1 Strengths of R/S

- Simple and compact syntax of the language. You can learn R quickly, and can accomplish a lot with very little code.
- Extensibility. Anyone can extend the functionality of R by writing code. This may be a simple function for personal use, or a whole new family of statistical procedures in a new package.
- A huge variety of statistical and computing procedures. This derives from the ease with which R/§ can be extended and shared by users around the world.
- Rapid updates.

[1] Pronounced "g-noo" — it is a recursive acronym standing for "GNU's Not Unix."

- Replicability and validation. All data analyses should be well documented, and this only happens reliably when the analyses are performed with scripts or programs, as in R or SAS. Point-and-click procedures cannot be validated by supervisors, reviewers, or auditors.
- Getting help from others is easy. Any scripted language can be quickly and easily shared with someone who can help you. I cannot help someone who says "first I clicked on this, and then I clicked on that ...."
- Repetitive tasks simplified. Writing code allows you to do anything you want a huge number of times. It also allows very simple updates with new data.
- High quality graphics. Well-designed publication-quality plots can be produced with ease, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.
- R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs out of the box on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux). It also compiles and runs on Windows 9x/NT/2000 and Mac OS.
- Accessibility. Go now to www.r-project.org. Type "R" into Google. The R Project page is typically the first hit.

There is a tremendous amount of free documentation for R. Rcomes with a selection of manuals under the "Help" menu — explore these first. At the main R project web site, see the "Documentation" on the left sidebar. The FAQ's are very helpful. The "Other" category includes a huge variety of items; search in particular for "Contributed documentation."[2] There you will find long (100+ pages) and short tutorials. You will also find two different "RReference Cards," which are useful lists of commonly used functions.[3]

A.2 The R Graphical User Interface (GUI)

R has a very simple but useful graphical user interface (GUI; Fig. A.1). A few points regarding the GUI:

- "You call this a graphical user interface?" Just kidding — the GUI is *not* designed for point-and-click modeling.
- The R GUI is designed for package management and updates.
- The R GUI is designed for use with scripts.

The R GUI does not provide a "statistics package." R is a language and programming environment. You can download an R package called `Rcmdr` that provides a point-and-click interface for introductory statistics, if you really, really want to. In my experience, students who plan to use statistics in their

[2] I find this when I google "r 'contributed documentation'."
[3] Try googling 'R Reference Card' in quotes.

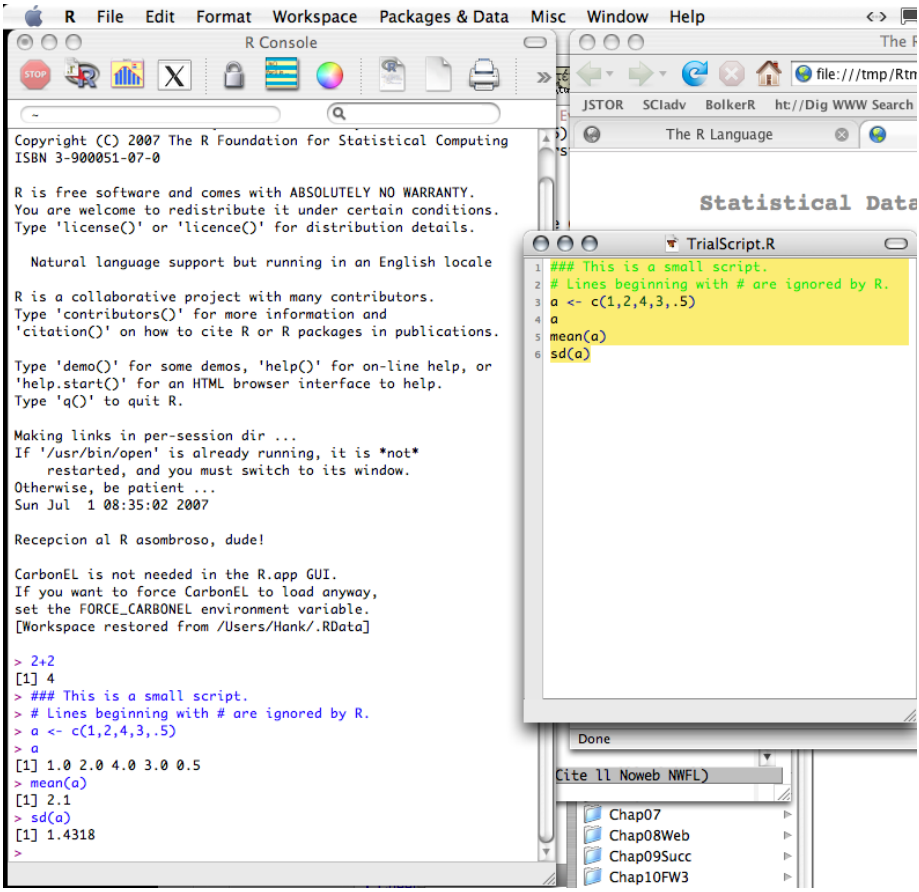


Fig. A.1: The Mac OS X R GUI. Color coded syntax not visible in this figure.

research find it more frustrating to learn this interface than to learn to take advantage of the language.

The R GUI *is* designed for maintenance. With the R GUI you can check for updates, and download any of the hundreds of small packages that extend R in hundreds of ways. (A package is not unlike a "PROC," for SAS users — first you tell call it, then you use it).

The R GUI *is* designed for using scripts. Scripts are text files that contain your analysis; that is, they contain both code to do stuff, and comments about what you are doing and why. These are opened within R and allow you to do work and save the code.

- Scripts are NOT Microsoft Word documents that require Microsoft Word to open them, but rather, simple text files, such as one could open in Notepad, or SimpleText.
- Scripts are a written record of everything you do along the way to achieving your results.

- Scripts are the core of data analysis, and provide many of the benefits of using a command-driven system, whether R, Matlab, or some other program or environment.
- Scripts are interactive. I find that because scripts allow me to do anything and record what I do, they are very interactive. They let me try a great variety of different things very quickly. This will be true for you too, once you begin to master the language.

The R GUI can be used for simple command line use. At the command line, you can add 2 + 2 to get 4. You could also do `ar(lake.phosphorus)` to perform autoregressive time series analysis on a variable called `lake.phosphorus`, but you would probably want to do that in a script that you can save and edit, to keep track of the analysis that you are doing.

A.3 Where is R?

As an Open Source project, R is distributed across the web. People all around the world continue to develop it, and much of it is stored on large computer servers ("mirrors") across the world. Therefore, when you download R, you download only a portion of it — the language and a few base packages that help everything run. Hundreds of "value-added" packages are available to make particular tasks and groups of tasks easier. We will download one or more of these.

It is useful to have a clear conception of where different parts of R reside (Fig. A.2). Computer servers around the world store identical copies of everything (hence the terms archive and "mirrors"). When you open R, you load into your computer's virtual, temporary RAM more than just the R language — you automatically load several useful packages including "base," "stat," and others. Many more packages exist (about a dozen come with the normal download) and hundreds are available at each mirror. These are easily downloaded through the R GUI.

A.4 Starting at the Very Beginning

To begin with, we will go through a series of steps that will get you up and running using a script file with which to interact with R, and using the proper working directory. Start here.

1. Create a new directory (i.e., a folder) in "Documents" (Mac) or "My Documents" (Windows). *and call it* "Rwork." For now, calling it the same thing as everyone else will just simplify your life. If you put "Rwork" elsewhere, adjust the relevant code as you go. For now, keep all of your script files and output files into that directory.
2. Open the R GUI in a manner that is appropriate for your operating system and setup (e.g., double-click the desktop icon).

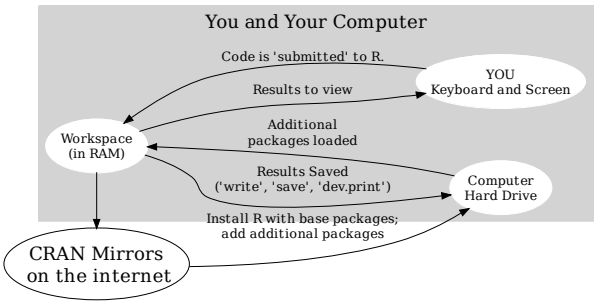


Fig. A.2: A conceptual representation of where R exists. "CRAN" stands for "Comprehensive R Archive Network." "RAM" (random access memory) is your computer's active brain; R keeps some stuff floating in this active memory and this "stuff" is the *workspace.*

3. Set the *working directory.* You can do this via Misc directory in Mac OS X or in the File menu in Windows using "Change dir...." Set the working directory to "Rwork." (If you have not already made an `Rwork` directory, do so now — put it in "Documents" or "My Documents.")
4. Open a new R script ("New Document") by using the File menu in the R GUI. On the first line, type `# My first script` with the pound sign. On the next line, type `setwd('~/Documents/Rwork')` if you are on a Mac, or `setwd('C:/Documents and Settings/Users/Jane/My Documents/Rwork')` on Windows, assuming you are named "Jane;" if not, use the appropriate pathname. Save this file in "Rwork;" save it as "RIntro.R." Windows may hide the fact that it is saving it as a ".txt" file. I will assume you can prevent this.

You should submit code to R directly from the script. *Use the script to store your ideas as comments (beginning with #) and your code, and submit code directly from the script file within R (see below for how to do that).* You do *not* need to cut-and-paste. There are slight differences between operating systems in how to submit code from the script file to the command line.

- In Microsoft Windows, place the cursor on a line in the script file *or* highlight a section of code, and then hit `Ctrl-R` to submit the code.
- In Apple Mac OS X, highlight a section of code and then hit `Command-return` to submit the code (see the Edit menu).

From this point on, enter all of the code (indicated in typewriter font, and beginning with ">") in your script file, save the file with Command-S (Mac) or Ctrl-S (Windows), and then submit the code as described above. Where a line begins with "+," ignore the plus sign, because this is just used by R to indicate continued lines of code. You may enter a single line of code on more than one line. R will simply continue as if you wrote it all on one line.

You can start the help interface with this command.

```
> help.start()
```

This starts the HTML interface to on-line help (using a web browser available at your machine). You can use help within the R GUI, or with this HTML help system.

Find out where you are using `getwd()` (*get* the *w*orking *d*irectory). Use a comment (beginning with #) to remind yourself what this does.

```
> # Here I Get my Working Directory; that is,
> # I find out which folder R is currently operating from.
> getwd()
```

The precise output will depend on your computer.

You can also *set* the *w*orking *d*irectory using `setwd()`; if you created a directory called `Rwork` as specified above, one of the following these should work, depending on your operating system. If these both fail, keep trying, or use the menu in the R GUI to set the working directory.

```
> setwd("~/Documents/Rwork")
```

or

```
> setwd("C:/Documents and Settings/Users/Jane/My Documents/Rwork")
```

On the Mac, a UNIX environment, the tilde-slash (~ /) represents your home directory.

I urge you to use `setwd` *at the beginning of each script file you write so that this script always sets you up to work in a particular, specified directory.* As you write your script file, remember,

- Text that begins with "#" will be ignored by R.
- Text that does not make sense to R will cause R to return an error message, but will not otherwise mess things up.

Remember that a strength of R is that you can provide, to yourself and others, comments that explain every step and every object. To add a comment, simply type one or more "#," followed by your comment.

Finally, have fun, be amused. Now you are ready for programming in R.

R is a language. Use it every day, and you will learn it quickly.

B

Programming in R

This material assumes you have completed Appendix A, the overview of R. Do the rest of this Appendix in a script. Make comments of your own throughout your script.

Open and save a new file (or *script*) in R. Think of your script as a pad of paper, on which you program and *also on which you write notes to yourself.* See Appendix A for instructions.

You will want to take copious notes about what you do. Make these notes in the script file, using the pound sign #. Here is an example:

```
> # This will calculate the mean of 10 random standard normal variables.
> mean( rnorm( 10 ) )

[1] 0.1053
```

You submit this (as described above) from the script directly to the Console with (select) Command-r on a Mac, or Ctrl-r on Windows.

B.1 Help

You cannot possibly use R without using its help facilities. R comes with a lot of documentation (see the relevant menu items), and people are writing more all the time (this document is an example!).

After working through this tutorial, you could go through the document "An Introduction to R" that comes with R. You can also browse "Keywords by Topic" which is found under "Search Engine & Keywords" in the Help menu.

To access help for a specific function, try

```
> `?`(mean)

Help for 'mean' is shown in browser /usr/bin/open ...
Use
        help("mean", htmlhelp = FALSE)
or
        options(htmlhelp = FALSE)
to revert.
```

or

```
> help(mean)

Help for 'mean' is shown in browser /usr/bin/open ...
Use
        help("mean", htmlhelp = FALSE)
or
        options(htmlhelp = FALSE)
to revert.
```

The help pages provide a very regular structure. There is a *name*, a brief *description*, its *usage*, its *arguments*, *details* about particular aspects of its use, the *value* (what you get when you use the function), *references*, *links* to other functions, and last, *examples*.

If you don't know the exact name of the R function you want help with, you can try

```
> help.search("mean")
> apropos("mean")
```

These will provide lists of places you can look for functions related to this keyword.

Last, a great deal of R resides in *packages* online (on duplicate servers around the world). If you are on line, help for functions that you have not yet downloaded can be retrieved with

```
> RSiteSearch("violin")
> RSiteSearch("violin", restrict = c("functions"))
```

To learn more about help functions, combine them!

```
> help(RSiteSearch)
```

B.2 Assignment

In general in R, we perform an action, and take the results of that action and *assign* the results to a new object, thereby creating a new object. Here I add two numbers and assign the result to an new object I call `a`.

```
> a <- 2 + 3
> a

[1] 5
```

Note that I use an arrow to make the assignment — I make the arrow with a less-than sign, <, and a dash. Note also that to reveal the contents of the object, I can type the name of the object.

I can then use the new object to perform another action, and assign

```
> b <- a + a
```

I can perform two actions on one line by separating them with a semicolon.

```
> a + a; a + b

[1] 10
[2] 15
```

Sometimes the semicolon is referred to as an "end of line" operator.

B.3 Data Structures

We refer to a single number as a scalar; a scalar is a single real number. Most objects in R are more complex. Here we describe some of these other objects: vectors, matrices, data frames, lists, and functions.

B.3.1 Vectors

Perhaps the fundamental unit of R is the *vector*, and most operations in R are performed on vectors. A vector may be just a column of scalars, for instance; this would be a column vector.

Here we create a vector called `Y`.

To enter data directly into R, we will use `c()` and create an R object, in particular a vector. A vector is, in this case, simply a group of numbers arranged in a row or column. Type into your script

```
> Y <- c(8.3, 8.6, 10.7, 10.8, 11, 11, 11.1, 11.2,
+     11.3, 11.4)
```

where the arrow is a less-than sign, `<`, and a dash, `-`. Similarly, you could use

```
> Y = c(8.3, 8.6, 10.7, 10.8, 11, 11, 11.1, 11.2, 11.3,
+     11.4)
```

These are equivalent.

R operates (does stuff) to *objects*. Those objects may be *vectors*, *matrices*, *lists*, or some other class of object.

Sequences

I frequently want to create ordered sequences of numbers. R has a shortcut for sequences of integers, and a slightly longer method that is completely flexible. First, integers:

```
> 1:4

[1] 1 2 3 4

> 4:1

[1] 4 3 2 1

> -1:3
```

```
[1] -1  0  1  2  3
```

```
> -(1:3)
```

```
[1] -1 -2 -3
```

Now more complex stuff, specifying either the units of the sequence, or the total length of the sequence.

```
> seq(from = 1, to = 3, by = 0.2)
```

```
 [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
```

```
> seq(1, 3, by = 0.2)
```

```
 [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
```

```
> seq(1, 3, length = 7)
```

```
[1] 1.000 1.333 1.667 2.000 2.333 2.667 3.000
```

I can also fill in with repetitive sequences. Compare carefully these examples.

```
> rep(1, 3)
```

```
[1] 1 1 1
```

```
> rep(1:3, 2)
```

```
[1] 1 2 3 1 2 3
```

```
> rep(1:3, each = 2)
```

```
[1] 1 1 2 2 3 3
```

B.3.2 Getting information about vectors

Here we can ask R to tell us about `Y`, getting the length (the number of elements), the mean, the maximum, and a six number summary.

```
> sum(Y)
```

```
[1] 105.4
```

```
> mean(Y)
```

```
[1] 10.54
```

```
> max(Y)
```

```
[1] 11.4
```

```
> length(Y)
```

```
[1] 10
```

```
> summary(Y)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    8.3    10.7    11.0    10.5    11.2    11.4
```

A vector could be character, or logical as well, for instance

```
> Names <- c("Sarah", "Yunluan")
> Names

[1] "Sarah"   "Yunluan"

> b <- c(TRUE, FALSE)
> b

[1]  TRUE FALSE
```

Vectors can also be dates, complex numbers, real numbers, integers, or factors. For factors, such as experimental treatments, see section B.3.5. We can also ask R what classes of data these belong to.

```
> class(Y)

[1] "numeric"

> class(b)

[1] "logical"
```

Here we test whether each element of a vector is greater than a particular value or greater than its mean. *When we test an object, we get a logical vector back that tells us, for each element, whether the condition was true or false.*

```
> Y > 10

 [1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE

> Y > mean(Y)

 [1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

We test using >, <, >=, <=, ==, ! = and other conditions. Here we test whether a vector is equal to a number.

```
> Y == 11

 [1] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

A test of "not equal to"

```
> Y != 11

 [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

This result turns out to be quite useful, including when we want to *extract* subsets of data.

Algebra with vectors

In R, we can add, subtract, multiply and divide vectors. When we do this, we are really *operating on the elements in the vectors*. Here we add vectors `a` and `b`.

```
> a <- 1:3
> b <- 4:6
> a + b

[1] 5 7 9
```

Similarly, when we multiply or divide, we also operate on each pair of elements in the pair of vectors.

```
> a * b

[1]  4 10 18

> a/b

[1] 0.25 0.40 0.50
```

We can also use scalars to operate on vectors.

```
> a + 1

[1] 2 3 4

> a * 2

[1] 2 4 6

> 1/a

[1] 1.0000 0.5000 0.3333
```

What R is doing is *recycling* the scalar (the 1 or 2) as many times as it needs to in order to match the length of the vector. Note that if we try to multiply vectors of unequal length, R performs the operation but may or may not give a warning. Above, we got no warningmessage. However, if we multiply a vector of length 3 by a vector of length 2, R returns a warning.

```
> a * 1:2

[1] 1 4 3
```

R *recycles the shorter vector* just enough to match the length of the longer vector. The above is the same as

```
> a * c(1, 2, 1)

[1] 1 4 3
```

On the other hand, if we multiply vectors of length 4 and 2, we get no error, because four is a multple of 2.

```
> 1:4 * 1:2

[1] 1 4 3 8
```

Recycling makes the above the same as the following.

```
> 1:4 * c(1, 2, 1, 2)

[1] 1 4 3 8
```

B.3.3 Extraction and missing values

We can extract or subset elements of the vector.

I extract subsets of data in two basic ways, by

- identifying which rows (or columns) I want (i.e. the first row), or
- providing a *logical* vector (of TRUE's and FALSE's) of the same length as the vector I am subsetting.

Here I use the first method, using a single integer, and a sequence of integers.

```
> Y[1]

[1] 8.3

> Y[1:3]

[1]  8.3  8.6 10.7
```

Now I want to extract all girths greater than the average girth. Although I don't have to, I remind myself what the logical vector looks like, and then I use it.

```
> Y > mean(Y)

 [1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE

> Y[Y > mean(Y)]

[1] 10.7 10.8 11.0 11.0 11.1 11.2 11.3 11.4
```

Note that I get back all the values of the vector where the condition was TRUE.

In R, *missing data* are of the type "NA." This means "not available," and R takes this appellation seriously. thus is you try to calculate the mean of a vector with missing data, R resists doing it, because if there are numbers missing from the set, how could it possibly calculate a mean? If you ask it to do something with missing data, the answer will be missing too.

Given that R treats missing data as missing data (and not something to be casually tossed aside), there are special methods to deal with such data. For instance, we can test which elements are missing with a special function, `is.na`.

```
> a <- c(5, 3, 6, NA)
> a

[1]  5  3  6 NA

> is.na(a)
```

```
[1] FALSE FALSE FALSE  TRUE
```

```
> !is.na(a)
```

```
[1]  TRUE  TRUE  TRUE FALSE
```

```
> a[!is.na(a)]
```

```
[1] 5 3 6
```

```
> na.exclude(a)
```

```
[1] 5 3 6
attr(,"na.action")
[1] 4
attr(,"class")
[1] "exclude"
```

Some functions allow you to remove missing elements on the fly. Here we let a function fail with missing data, and then provide three different ways to get the same thing.

```
> mean(a)
```

```
[1] NA
```

```
> mean(a, na.rm = TRUE)
```

```
[1] 4.667
```

```
> d <- na.exclude(a)
> mean(d)
```

```
[1] 4.667
```

Note that R takes missing data seriously. If the fourth element of the set really is missing, I cannot calculate a mean because I don't know what the vector is.

B.3.4 Matrices

A matrix is a two dimensional set of elements, for which *all elements are of the same type*. Here is a character matrix.

```
> matrix(letters[1:4], ncol = 2)
```

```
     [,1] [,2]
[1,] "a"  "c"
[2,] "b"  "d"
```

Here we make a numeric matrix.

```
> M <- matrix(1:4, nrow = 2)
> M
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Note that the matrix is filled in by columns, or *column major order*. We could also do it by rows.

```
> M2 <- matrix(1:4, nrow = 2, byrow = TRUE)
> M2
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

Here is a matrix with 1s on the diagonal.

```
> I <- diag(1, nrow = 2)
> I
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

The identity matrix plays a special role in matrix algebra; in many ways it is equivalent to the scalar 1. For instance, the inverse of a matrix, $\mathbf{M}$, is $\mathbf{M}^{-1}$, which is the matrix which satisfies the equality $\mathbf{M}\mathbf{M}^{-1} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. We solve for the inverse using a few different methods, including

```
> Minv <- solve(M)
> M %*% Minv
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

QR decomposition is available (e.g., `qr.solve()`).

Note that R recycles the "1" until the specified number of rows and columns are filled. If we do not specify the number of rows and columns, R fills in the matrix with what you give it (as it did above).

Extraction in matrices

I extract elements of matrices in the same fashion as vectors, but specify both rows and columns.

```
> M[1, 2]
[1] 3
> M[1, 1:2]
[1] 1 3
```

If I leave either rows or columns blank, R returns all rows (or columns).

```
> M[, 2]
[1] 3 4
> M[, ]
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Simple matrix algebra

Basic matrix algebra is similar to algebra with scalars, but with a few very important differences. Let us define another matrix.

```
> N <- matrix(0:3, nrow = 2)
> N
```

```
     [,1] [,2]
[1,]    0    2
[2,]    1    3
```

To perform scalar, or element-wsie operations, we have

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix};\ \mathbf{B} = \begin{pmatrix} m & o \\ n & p \end{pmatrix} \tag{B.1}$$

$$\mathbf{AB} = \begin{pmatrix} am & bo \\ cn & dp \end{pmatrix} \tag{B.2}$$

The element-wise operation on these two is the default in R,

```
> M * N
```

```
     [,1] [,2]
[1,]    0    6
[2,]    2   12
```

where the element in row 1, column 1 in `M` is multiplied by the element in the same position in `N`.

To perform *matrix mulitplication*, recall from Chap. 2 that,

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix};\ \mathbf{B} = \begin{pmatrix} m & o \\ n & p \end{pmatrix} \tag{B.3}$$

$$\mathbf{AB} = \begin{pmatrix} (am + bn) & (ao + bp) \\ (cm + dn) & (co + dp) \end{pmatrix} \tag{B.4}$$

To perform *matrix mulitplication* in R, we use **%*%**,

```
> M %*% N
```

```
     [,1] [,2]
[1,]    3   11
[2,]    4   16
```

Refer to Chapter 2 (or "matrix algebra" at Wikipedia) for why this is so.

Note that matrix multiplication is not commutative, that is, $\mathbf{NM} \neq \mathbf{MN}$. Compare the previous result to

```
> N %*% M
```

```
     [,1] [,2]
[1,]    4    8
[2,]    7   15
```

Note that a *vector* in R is not defined *a priori* as a column matrix or a row matrix. Rather, it is used as either depending on the circumstances. Thus, we can either left multiply or right multiply a vector of length 2 and `M`.

```
> 1:2 %*% M

     [,1] [,2]
[1,]    5   11

> M %*% 1:2

     [,1]
[1,]    7
[2,]   10
```

If you want to be very, very clear that your vector is really a matrix with one column (a column vector), you can make it thus.

```
> V <- matrix(1:2, ncol = 1)
```

Now when you multiply `M` by `V`, you will get the expected sucesses and failure, according to the rules of matrix algebra.

```
> M %*% V

     [,1]
[1,]    7
[2,]   10

> try(V %*% M)
```

R has formal rules about how it converts vectors to matrices on-the-fly, but it is good to be clear on your own.

Other matrix operations are available. Whenever we add or subtract matrices together, or add a matrix and a scalar, it is always element-wise.

```
> M + N

     [,1] [,2]
[1,]    1    5
[2,]    3    7

> M + 2

     [,1] [,2]
[1,]    3    5
[2,]    4    6
```

The transpose of a matrix is the matrix we get when we substitute rows for columns, and columns for rows. To transpose matrices, we use `t()`.

```
> t(M)

     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

More advanced matrix operations are available as well, for singular value decomposition (`svd`), eigenanalysis (`eigen`), finding determinants (`det`), QR decomposition (`qr`), Choleski factorization (`chol`), and related functions. The `Matrix` package was designed to handle with aplomb large sparse matrices.

B.3.5 Data frames

Data frames are two dimensional, a little like spreadsheets and matrices. All columns having exactly the same number of rows. Unlike matrices, each column can be a different data type (e.g., numeric, integer, charactor, complex, imaginary). For instance, the columns of a data frame could contain the names of species, the experimental treatment used, and the dimensions of species traits, as character, factor, and numeric variables, respectively.

```
> dat <- data.frame(species = c("S.altissima", "S.rugosa",
+     "E.graminifolia", "A. pilosus"), treatment = factor(c("Control",
+     "Water", "Control", "Water")), height = c(1.1,
+     0.8, 0.9, 1), width = c(1, 1.7, 0.6, 0.2))
> dat

         species treatment height width
1    S.altissima   Control    1.1   1.0
2       S.rugosa     Water    0.8   1.7
3 E.graminifolia   Control    0.9   0.6
4     A. pilosus     Water    1.0   0.2
```

We can extract data from data frames just the way we can with matrices.

```
> dat[2, ]

   species treatment height width
2 S.rugosa     Water    0.8   1.7

> dat[3, 4]

[1] 0.6
```

We can test elements in data frames, as here where I test whether each element column 2 is "Water." I then use that to extract rows of data that are associated with this criterion.

```
> dat[, 2] == "Water"

[1] FALSE  TRUE FALSE  TRUE

> dat[dat[, 2] == "Water", ]

     species treatment height width
2   S.rugosa     Water    0.8   1.7
4 A. pilosus     Water    1.0   0.2
```

I could also use the subset function

```
> subset(dat, treatment == "Water")
```

```
     species treatment height width
2   S.rugosa     Water    0.8   1.7
4 A. pilosus     Water    1.0   0.2
```

There are advantages to using data frames which will become apparent.

Factors

Factors are a class of data; as such they could belong above with our discussion of character and logical and numeric vectors. I tend, however, to use them in data frames almost exclusively, because I have a data set that includes a bunch of response variables, and *the factors imposed by my experiment.*

When defining a factor, R by default orders the factor levels in alphabetic order — we can reorder them as we like. Here I demonstrate each piece of code and then use the pieces to make a factor in one line of code.

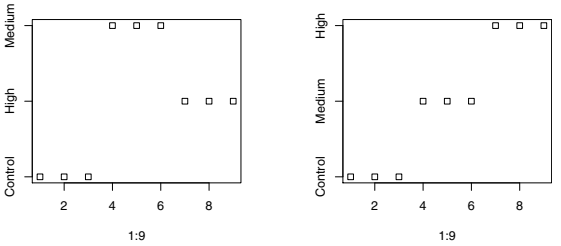
```
> c("Control", "Medium", "High")

[1] "Control" "Medium"  "High"

> rep(c("Control", "Medium", "High"), each = 3)

[1] "Control" "Control" "Control" "Medium"  "Medium"  "Medium"
[7] "High"    "High"    "High"

> Treatment <- factor(rep(c("Control", "Medium", "High"),
+     each = 3))
> Treatment

[1] Control Control Control Medium  Medium  Medium  High
[8] High    High
Levels: Control High Medium
```

Note that R orders the factor alphabetically. This may be relevant if we do something with the factor, such as when we plot it (Fig. B.1a).

```
> levels(Treatment)

[1] "Control" "High"    "Medium"

> stripchart(1:9 ~ Treatment)
```

Now we can re-specify the factor, telling R the order of the levels we want, taking care to remember that R can tell the difference between upper and lower case (Fig. B.1b). See also the function `relevel`.

```
> Treatment <- factor(rep(c("Control", "Medium", "High"),
+     each = 3), levels = c("Control", "Medium", "High"))
> levels(Treatment)

[1] "Control" "Medium"  "High"

> stripchart(1:9 ~ Treatment)
```



(a) Before (b) After

Fig. B.1: Graphics before and after the factor was releveled to place the factor levels in a logical order.

B.3.6 Lists

An amazing data structure that R boasts is the *list*. A *list* is simply a collection of other objects kept together in a hierarchical structure. Each component of the list can be a complete different class of object. Let's build one.

```
> my.list <- list(My.Y = Y, b = b, Names, Weed.data = dat,
+     My.matrix = M2, my.no = 4)
> my.list

$My.Y
 [1]  8.3  8.6 10.7 10.8 11.0 11.0 11.1 11.2 11.3 11.4

$b
[1] 4 5 6

[[3]]
[1] "Sarah"   "Yunluan"

$Weed.data
         species treatment height width
1    S.altissima   Control    1.1   1.0
2       S.rugosa     Water    0.8   1.7
3 E.graminifolia   Control    0.9   0.6
4     A. pilosus     Water    1.0   0.2

$My.matrix
     [,1] [,2]
[1,]    1    2
[2,]    3    4

$my.no
[1] 4
```

We see that this list is a set of objects: a numeric vector, a logical vector, a character vector, a data frame, a matrix, and a scalar (a number). Lists can be nested within other lists.

Note that if we do not specify a name for a component, we can still extract it using the number of the component.

I extract list components in several ways, including by name, and by number (see `?'['` for more information).

```
> my.list[["b"]]

[1] 4 5 6

> my.list[[2]]

[1] 4 5 6
```

If I use a name, there are a few ways, including

```
> my.list[["b"]]

[1] 4 5 6

> my.list$b

[1] 4 5 6
```

If by number, that are two ways, with one or two brackets. In addition to two brackets, as above, we can use one bracket. This allows for extraction of more than one component of the list.

```
> my.list[1:2]

$My.Y
 [1]  8.3  8.6 10.7 10.8 11.0 11.0 11.1 11.2 11.3 11.4

$b
[1] 4 5 6
```

Note that I can extract a subset of one component.

```
> my.list[["b"]][1]

[1] 4
```

If one way of extraction is working for you, experiment with others.

B.3.7 Data frames are also lists

You can also think of a data frame as a list of columns of identical length. I like to extract columns the same way — by name.

```
> mean(dat$height)

[1] 0.95
```

B.4 Functions

A function is a command that does something. You have already been using functions, throughout this document. Let's examine functions more closely.

Among other things, a function has a name, arguments, and values. For instance,

```
> help(mean)
```

This will open the help page (again), showing us the *arguments*. The first argument `x` is the object for which a mean will be calculated. The second argument is `trim=0`. If we read about this argument, we find that it will "trim" a specified fraction of the most extreme observations of `x`. *The fact that the argument `trim` is already set equal to zero means that is the default.* If you do not use `trim`, then the function will use `trim=0`. Thus, these two are equivalent.

```
> mean(1:4)

[1] 2.5

> mean(1:4, trim = 0)

[1] 2.5
```

R is an "object-oriented" language. A consequence of this is that the same function name will perform different actions, depending on the *class* of the object.[1]

```
> class(1:10)

[1] "integer"

> class(warpbreaks)

[1] "data.frame"

> summary(1:10)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.25    5.50    5.50    7.75   10.00

> summary(warpbreaks)

     breaks     wool   tension
 Min.   :10.0   A:27   L:18
 1st Qu.:18.2   B:27   M:18
 Median :26.0          H:18
 Mean   :28.1
 3rd Qu.:34.0
 Max.   :70.0
```

In the `warpbreaks` data frame, summary provides the six number summary for each numeric or integer column, but provides "tables" of the factors, that is, it counts the occurrences of each level of a factor and sorts the levels. When we use summary on a linear model, we get output of the regression,

[1] R has hundreds of built-in data sets for demonstrating things. We use one here called 'warpbreaks.' You can find some others by typing `data()`.

```
> summary(lm(breaks ~ wool, data = warpbreaks))

Call:
lm(formula = breaks ~ wool, data = warpbreaks)

Residuals:
   Min     1Q Median     3Q    Max
-21.04  -9.26  -3.65   4.71  38.96

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    31.04       2.50   12.41   <2e-16
woolB          -5.78       3.54   -1.63     0.11

Residual standard error: 13 on 52 degrees of freedom
Multiple R-squared: 0.0488,        Adjusted R-squared: 0.0305
F-statistic: 2.67 on 1 and 52 DF,  p-value: 0.108
```

B.4.1 Writing your own functions

One very cool thing in R is that you can write your own functions. Indeed it is the extensibility of R that makes it the home of cutting edge working, because edge cutters (i.e., leading scientists) can write code that we all can use. People actually write entire *packages*, which are integrated collections of functions, and R has been extended with hundreds of such packages available for download at all the R mirrors.

Let's make our own function to calculate a mean. Let's further pretend you work for an unethical boss who wants you to show that average sales are higher than they really are. Therefore your function should provide a mean plus 5%.

```
> MyBogusMean <- function(x, cheat = 0.05) {
+     SumOfX <- sum(x)
+     n <- length(x)
+     trueMean <- SumOfX/n
+     (1 + cheat) * trueMean
+ }
> RealSales <- c(100, 200, 300)
> MyBogusMean(RealSales)

[1] 210
```

Thus a function can take any input, do stuff, including produce graphics, or interact with the operating system, or manipulated numbers. You decide on the arguments of the function, in this case, `x` and `cheat`. Note that we supplied a number for `cheat`; this results in the `cheat` argument having a *default* value, and we do not have to supply it. If an argument does not have a default, we have to supply it. If there is a default value, we can change it. Now try these.

```
> MyBogusMean(RealSales, cheat = 0.1)

[1] 220
```

```
> MyBogusMean(RealSales, cheat = 0)

[1] 200
```

B.5 Sorting

We often like to sort our numbers and our data sets; a single vector is easy. To do something else is only a little more difficult.

```
> e <- c(5, 4, 2, 1, 3)
> e

[1] 5 4 2 1 3

> sort(e)

[1] 1 2 3 4 5

> sort(e, decreasing = TRUE)

[1] 5 4 3 2 1
```

If we want to sort all the rows of a data frame, keeping records (rows) intact, we can use `order`. This function is a little tricky, so we explore its use in a vector.

```
> e

[1] 5 4 2 1 3

> order(e)

[1] 4 3 5 2 1

> e[order(e)]

[1] 1 2 3 4 5
```

Here `order` generates an *index* to properly *order* something. Above, this index is used to tell R to select the 4th element of `e` first — `order` puts the number '4' into the first spot, indicating that R should put the 4th element of `e` first. Next, it places '3' in the second spot because the 3rd element of `e` belongs in the 2nd spot of an ordered vector, and so on.

We can use `order` to sort the rows of a data frame. Here I order the rows of the data frame according to increasing order of plant heights.

```
> dat

         species treatment height width
1    S.altissima   Control    1.1   1.0
2       S.rugosa     Water    0.8   1.7
3 E.graminifolia   Control    0.9   0.6
4     A. pilosus     Water    1.0   0.2

> order.nos <- order(dat$height)
> order.nos
```

```
[1] 2 3 4 1
```

This tells us that to order the rows, we have to use the 2nd row of the original data frame as the first row in the ordered data frame, the 3rd row as the new second row, etc. Now we use this index to select the rows of the original data frame in the correct order to sort the whole data frame.

```
> dat[order.nos, ]
```

```
         species treatment height width
2       S.rugosa     Water    0.8   1.7
3 E.graminifolia   Control    0.9   0.6
4     A. pilosus     Water    1.0   0.2
1    S.altissima   Control    1.1   1.0
```

We can reverse this too, of course.

```
> dat[rev(order.nos), ]
```

```
         species treatment height width
1    S.altissima   Control    1.1   1.0
4     A. pilosus     Water    1.0   0.2
3 E.graminifolia   Control    0.9   0.6
2       S.rugosa     Water    0.8   1.7
```

B.6 Iterated Actions: the apply Family and Loops

We often want to perform an action again and again and again..., perhaps thousands or millions of times. In some cases, each action is independent — we just want to do it a lot. In these cases, we have a choice of methods. Other times, each action depends on the previous action. In this case, I always use for-loops.[2] Here I discuss first methods that work only for independent actions.

B.6.1 Iterations of independent actions

Imagine that we have a matrix or data frame and we want to do the same thing to each column (or row). For this we use `apply`, to "apply" a function to each column (or row). We tell `apply` what data we want to use, we tell it the "margin" we want to focus on, and then we tell it the function. The *margin* is the *side* of the matrix. We describe matrices by their number of rows, then columns, as in "a 2 by 5 matrix," so rows constitute the first margin, and columns constitute the second margin. Here we create a 2×5 matrix, and take the mean of rows, for the first margin. Then we sum the columns for the second margin.

```
> m <- matrix(1:10, nrow = 2)
> m
```

[2] There are other methods we could use. These are discussed by others, under various topics, including "flow control." We use ODE solvers for continuous ordinary differential equations.

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

> apply(m, MARGIN = 1, mean)

[1] 5 6

> apply(m, MARGIN = 2, sum)

[1]  3  7 11 15 19
```

See `?rowMeans` for simple, and even faster, operations.

Similarly, `lapply` will "apply" a function to each element of a list, or each column of a data frame, and always returns a list. `sapply` does something similar, but will simplify the result, to a less complex data structure if possible.

Here we do an independent operation 10 times using `sapply`, defining a function on-the-fly to calculate the mean of a random draw of five observations from the standard normal distribution.

```
> sapply(1:10, function(i) mean(rnorm(5)))

 [1] -0.5612 -0.4815 -0.4646  0.7636  0.1416 -0.5003 -0.1171
 [8]  0.2647  0.6404 -0.1563
```

B.6.2 Dependent iterations

Often the repeated actions depend on previous outcomes, as with population growth. Here we provide a couple of examples where we accomplish this with *for loops*.

One thing to keep in mind for *for loops* in R: the computation of this is fastest if we first make a holder for the output. Here I simulate a random walk, where, for instance, we start with 25 individuals at time = 0, and increase or decrease by some amount that is drawn randomly from a normal distribution, with a mean of zero and a standard deviation 2. We will round the "amount" to the nearest integer (the zero-th decimal place). Your output will differ because it is a random process.

```
> gens <- 10
> output <- numeric(gens + 1)
> output[1] <- 25
> for (t in 1:gens) output[t + 1] <- output[t] + round(rnorm(n = 1,
+     mean = 0, sd = 2), 0)
> output

 [1] 25 29 25 26 28 29 30 32 33 29 30
```

B.7 Rearranging and Aggregating Data Frames

B.7.1 Rearranging or reshaping data

We often need to rearrange our data. A common example in ecology is to collect repeated measurements of an experimental unit and enter the data into multiple columns of a spreadsheet, creating a *wide* format. R prefers to analyze data in a single column, in a *long* format. Here we use `reshape` to rearrange this.

These data are carbon dioxide uptake in 12 individual plants. They are currently structured as longitudinal data; here we rearrange them in the wide format, as if we record uptake seven sequential observations on each plant in different columns. See `?reshape` for details. Here `v.names` refers to the column name of the response variable, `idvar` refers to the column name for the variable that identifies an individual on which we have repeated measurements, and `timevar` refers to the column name which identifies different observations of *the same individual plant.*

```
> summary(CO2)
```

```
    Plant             Type         Treatment       conc
 Qn1    : 7   Quebec     :42   nonchilled:42   Min.   :  95
 Qn2    : 7   Mississippi:42   chilled   :42   1st Qu.: 175
 Qn3    : 7                                   Median : 350
 Qc1    : 7                                   Mean   : 435
 Qc3    : 7                                   3rd Qu.: 675
 Qc2    : 7                                   Max.   :1000
 (Other):42
    uptake
 Min.   : 7.7
 1st Qu.:17.9
 Median :28.3
 Mean   :27.2
 3rd Qu.:37.1
 Max.   :45.5
```

```
> CO2.wide <- reshape(CO2, v.names = "uptake", idvar = "Plant",
+     timevar = "conc", direction = "wide")
> names(CO2.wide)
```

```
 [1] "Plant"       "Type"        "Treatment"   "uptake.95"
 [5] "uptake.175"  "uptake.250"  "uptake.350"  "uptake.500"
 [9] "uptake.675"  "uptake.1000"
```

This is often how we might record data, with an experimental unit (individual, or plot) occupying a single row. If we import the data in this format, we would typically like to reorganize it in the long format, because most analyses we want to do may require this. Here, `v.names` and `timevar` are the names we want to use for some new columns, for the response variable and the identifier of the repeated measurement (typically the latter may be a time interval, but here it is a CO_2 concentration). `times` supplies the identifier for each repeated observation.

```
> CO2.long <- reshape(CO2.wide, v.names = "Uptake",
+     varying = list(4:10), timevar = "Concentration",
+     times = c(95, 175, 250, 350, 500, 675, 1000))
> head(CO2.long)
```

```
     Plant   Type  Treatment Concentration Uptake id
1.95   Qn1 Quebec nonchilled            95   16.0  1
2.95   Qn2 Quebec nonchilled            95   13.6  2
3.95   Qn3 Quebec nonchilled            95   16.2  3
4.95   Qc1 Quebec    chilled            95   14.2  4
5.95   Qc2 Quebec    chilled            95    9.3  5
6.95   Qc3 Quebec    chilled            95   15.1  6
```

If we wanted to, we could use `order()` to re-sort the data frame, for instance to match the original.

```
> CO2.long2 <- with(CO2.long, CO2.long[order(Plant,
+     Concentration), ])
> head(CO2.long2)
```

```
      Plant   Type  Treatment Concentration Uptake id
1.95    Qn1 Quebec nonchilled            95   16.0  1
1.175   Qn1 Quebec nonchilled           175   30.4  1
1.250   Qn1 Quebec nonchilled           250   34.8  1
1.350   Qn1 Quebec nonchilled           350   37.2  1
1.500   Qn1 Quebec nonchilled           500   35.3  1
1.675   Qn1 Quebec nonchilled           675   39.2  1
```

See also the very simple functions `stack` and `unstack`.

B.7.2 Summarizing by groups

We often want to summarize a column of data *by groups* identified in another column. Here I summarize CO_2 uptake by the means of each experimental treatment, chilling. The code below provides the column to be summarized (`uptake`), a vector (or list of vectors) containing the group id's, and the function to use to summarize each subset (means). We calculate the mean CO_2 uptake for each group.

```
> tapply(CO2[["uptake"]], list(CO2[["Treatment"]]),
+     mean)
```

```
nonchilled    chilled
     30.64      23.78
```

We can get fancier, as well, with combinations of groups, for each combination of Type and Treatment.

```
> tapply(CO2[["uptake"]], list(CO2[["Treatment"]],
+     CO2[["Type"]]), sd)
```

```
           Quebec Mississippi
nonchilled  9.596       7.402
chilled     9.645       4.059
```

We can also define a function on-the-fly to calculate both mean and standard deviation of Type and Treatment combination. We will need, however, to define groups differently, by creating the interaction of the two factors.

```
> tapply(CO2[["uptake"]], list(CO2[["Treatment"]],
+     CO2[["Type"]]), function(x) c(mean(x), sd(x)))
```

```
           Quebec    Mississippi
nonchilled Numeric,2 Numeric,2
chilled    Numeric,2 Numeric,2
```

See also `by` that actually uses `tapply` to operate on data frames.

When we summarize data, as in `tapply`, we often want the result in a nice neat data frame. The function `aggregate` does this. Its use is a bit like `tapply` — you provide (i) the numeric columns of a data frame, or a matrix, (ii) a list of named factors by which to organize the responses, and then (iii) the function to summarize (or aggregate) the data. Here we summarize both concentration and uptake.

```
> aggregate(CO2[, 4:5], list(Plant = CO2[["Plant"]]),
+     mean)
```

```
   Plant conc uptake
1    Qn1  435  33.23
2    Qn2  435  35.16
3    Qn3  435  37.61
4    Qc1  435  29.97
5    Qc3  435  32.59
6    Qc2  435  32.70
7    Mn3  435  24.11
8    Mn2  435  27.34
9    Mn1  435  26.40
10   Mc2  435  12.14
11   Mc3  435  17.30
12   Mc1  435  18.00
```

A separate package entitled `reshape` supplies some very elegant and intuitive approaches to the sorting, reshaping and aggregating of data frames. I typically use the `reshape` *package* (with functions `melt` and `cast`), rather than the `reshape` function supplied in the `stat`. I do so merely because I find it a little more intuitive. R also has strong connections to relational database systems such as MySQL.

B.8 Getting Data out of and into the Workspace

We often want to get data into R, and we sometimes want to get it out, as well. Here we start with the latter (referred to as *writing* data), and finish with the former (referred to as *reading* data).

Here I create a data frame of numbers, and write it to a text file in two different formats. The first is a file where the observations in each row are separated by tabs, and the second separates them by commas.

```
> dat <- data.frame(Name = rep(c("Control", "Treatment"),
+     each = 5), First = runif(10), Second = rnorm(1))
> write.table(dat, file = "dat.txt")
> write.csv(dat, file = "dat.csv")
```

Open these in a spreadsheet such as Calc (in OpenOffice and NeoOffice). We can then read these into R using the `read.*` family of functions.

```
> dat.new <- read.csv("dat.csv")
> dat.new2 <- read.table("dat.txt", header = TRUE)
```

These objects will both be data frames.

Now let's get a statistical summary and export that.

```
> mod.out <- summary(aov(First ~ Name, data = dat))
> mod.out[[1]]
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
Name         1 0.1562  0.1562    4.44  0.068
Residuals    8 0.2814  0.0352
```

```
> write.csv(mod.out[[1]], "ModelANOVA.csv")
```

Open this in a spreadsheet, such as Calc, in OpenOffice, or in any other application.

See also the `xtable` package for making tables in LaTeX or HTML formats.

B.9 Probability Distributions and Randomization

R has a variety of probability distributions built-in. For the normal distribution, for instance, there are four functions:

`dnorm` The probability density function, that creates the widely observed bell-shaped curve.

`pnorm` The cumulative probability function that we usually use to describe the probability that a test statistic is greater than or equal to a critical value.

`qnorm` The quantile function that takes probabilities as input.

`rnorm` A random number generator which draws values (quantiles) from a distribution with a specified mean and standard deviation.

For each of these, default parameter values return the standard normal distribution ($\mu = 0$, $\sigma = 1$), but these parameters can be changed.

Here we have the 95% confidence intervals.

```
> qnorm(p = c(0.025, 0.975))
```

```
[1] -1.96  1.96
```

Next we create a histogram using 20 random draws from a normal distribution with a mean of 11 and a standard deviation of 6; we overlay this with the probability density function (Fig. B.2).

```
> myplot <- hist(rnorm(20, m = 11, sd = 6), probability = TRUE)
> myplot

$breaks
[1]  0  5 10 15 20 25

$counts
[1] 1 8 6 4 1

$intensities
[1] 0.01 0.08 0.06 0.04 0.01

$density
[1] 0.01 0.08 0.06 0.04 0.01

$mids
[1]  2.5  7.5 12.5 17.5 22.5

$xname
[1] "rnorm(20, m = 11, sd = 6)"

$equidist
[1] TRUE

attr(,"class")
[1] "histogram"

> lines(myplot$mids, dnorm(myplot$mids, m = 11, sd = 6))
```

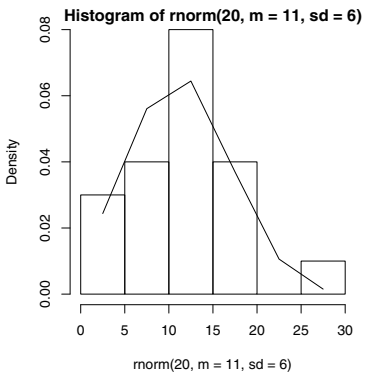


Fig. B.2: Histogram of random numbers drawn from a normal distribution with $\mu = 11$ and $\sigma = 6$. The normal probability density function is drawn as well.

B.10 Numerical integration of ordinary differential equations

In order to study continuous population dynamics, we often would like to integrate complex nonlinear functions of population dynamics. To do this, we need to use numerical techniques that turn the infinitely small steps of calculus, $\mathrm{d}x$, into very small, but finite steps, in order to approximate the change in y, given the change in x, or $\mathrm{d}y/\mathrm{d}x$. Mathematicians and computer scientists have devised very clever ways of doing this very accurately and precisely. In R, the best package for this is `deSolve`, which contains several *solvers* for differential equations that perform numerical integration. We will access these solvers (i.e. numerical integraters) using the `ode` function in the `deSolve` package. This function, `ode`, is a "wrapper" for the underlying suite of functions that do the work. That is, it provides a simple way to use any one of the small suite of functions.

When we have an ordinary differential equation (ODE) such as logistic growth,[3] we say that we "solve" the equation for a particular time interval given a set of parameters and initial conditions or initial population size. For instance, we say that we solve the logistic growth model for time at $t = 0, 1 \ldots 20$, with parameters $r = 1$, $\alpha = 0.001$, and $N_0 = 10$.

Let's do an example with `ode`, using logistic growth. We first have to define a function in a particular way. The arguments for the function must be time, a vector of populations, and a vector or list of model parameters.

```
> logGrowth <- function(t, y, p) {
+     N <- y[1]
+     with(as.list(p), {
+         dN.dt <- r * N * (1 - a * N)
+         return(list(dN.dt))
+     })
+ }
```

Note that I like to convert y into a readable or transparent state variable (N in this case). I also like to use `with` which allows me to use the names of my parameters [157]; this works only is `p` is a vector with named paramters (see below). Finally, we return the derivative as a list of one component.

The following is equivalent, but slightly less readable or transparent.

```
> logGrowth <- function(t, y, p) {
+     dN.dt <- p[1] * y[1] * (1 - p[2] * y[1])
+     return(list(dN.dt))
+ }
```

To solve the ODE, we will need to specify parameters, and initial conditions. Because we are using a vector of named parameters, we need to make sure we name them! We also need to supply the time steps we want.

```
> p <- c(r = 1, a = 0.001)
> y0 <- c(N = 10)
> t <- 1:20
```

[3] *e.g.* $dN/dt = rN(1 - \alpha N)$

Now you put it all into `ode`, with the correct arguments. The output is a matrix, with the first column being the time steps, and the remaining being your state variables. First we load the `deSolve` package.

```
> library(deSolve)
> out <- ode(y = y0, times = t, func = logGrowth, parms = p)
> out[1:5, ]
```

```
     time      N
[1,]    1  10.00
[2,]    2  26.72
[3,]    3  69.45
[4,]    4 168.66
[5,]    5 355.46
```

If you are going to model more than two species, y becomes a vector of length 2. Here we create a function for Lotka-Volterra competition, where

$$\frac{\mathrm{d}N_1}{\mathrm{d}t} = r_1 N_1 (1 - \alpha_{11} N_1 - \alpha_{12} N_2) \tag{B.5}$$

$$\frac{\mathrm{d}N_2}{\mathrm{d}t} = r_2 N_2 (1 - \alpha_{22} N_2 - \alpha_{21} N_1) \tag{B.6}$$

$$\tag{B.7}$$

```
> LVComp <- function(t, y, p) {
+     N <- y
+     with(as.list(p), {
+         dN1.dt <- r[1] * N[1] * (1 - a[1, 1] * N[1] -
+             a[1, 2] * N[2])
+         dN2.dt <- r[2] * N[2] * (1 - a[2, 1] * N[1] -
+             a[2, 2] * N[2])
+         return(list(c(dN1.dt, dN2.dt)))
+     })
+ }
```

Note that `LVComp` assumes that N and r are vectors, and the competition coefficients are in a matrix. For instance, the function extracts the the first element of `r` for the first species (`r[1]`); for the intraspecific competition coefficient for species 1, it uses the element of `a` that is in the first column and first row (`a[1,1]`). The vector of population sizes, N, contains one value for each population *at one time point*. Thus here, the vector contains only two elements (one for each of the two species); it holds only these values, but will do so repeatedly, at each time point. Only the output will contain all of the population sizxes through time.

To integrate these populations, we need to specify new initial conditions, and new parameters for the two-species model.

```
> a <- matrix(c(0.02, 0.01, 0.01, 0.03), nrow = 2)
> r <- c(1, 1)
> p2 <- list(r, a)
> N0 <- c(10, 10)
```

```
> t2 <- c(1, 5, 10, 20)
> out <- ode(y = N0, times = t2, func = LVComp, parms = p2)
> out[1:4, ]

     time     1     2
[1,]    1 10.00 10.00
[2,]    5 35.54 21.80
[3,]   10 39.61 20.36
[4,]   20 39.99 20.01
```

The `ode` function uses a superb ODE solver, `lsoda`, which is a very powerful, well tested tool, superior to many other such solvers. In addition, it has several bells and whistles that we will not need to take advantage of here, although I will mention one, `hmax`. This tells `lsoda` the largest step it can take. Once in a great while, with a very *stiff* ODE (a very wiggly complex dynamic), ODE assumes it can take a bigger step than it should. Setting `hmax` to a smallish number will limit the size of the step to ensure that the integration proceeds as it should.

One of the other solvers in the `deSolve`, `lsodar`, will also return roots (or equilibria), for a system of ODEs, if they exist. Here we find the roots (i.e. the solutions, or equilibria) for a two species enemy-victim model.

```
> EV <- function(t, y, p) {
+     with(as.list(p), {
+         dv.dt <- b * y[1] * (1 - 0.005 * y[1]) -
+             a * y[1] * y[2]
+         de.dt <- a * e * y[1] * y[2] - s * y[2]
+         return(list(c(dv.dt, de.dt)))
+     })
+ }
```

To use `lsodar` to find equilibria, we need to specify a root finding function whose inputs are are the sme of the ODE function, and which returns a scalar (a single number) that determines whether the rate of change ($\mathrm{d}y/\mathrm{d}x$) is sufficiently close to zero that we can say that the system has stopped changed, that is, has reached a steady state or equilibrium. Here we sum the absolute rates of change of each species, and then subtract 10^{-10}; if that difference is zero, we decide that, for all pratcial purposes, the system has stopped changing.

```
> rootfun <- function(t, y, p) {
+     dstate <- unlist(EV(t, y, p))
+     return(sum(abs(dstate)) - 1e-10)
+ }
```

Note that `unlist` changes the `list` returned by `EV` into a simple vector, which can then be summed.

Next we specify parameters, and time. Here all we want is the root, so we specify that we want the value of y after a really long time ($t = 10^{10}$). The `lsodar` function will stop sooner than that, and return the equilibrium it finds, and the time step at which it occurred.

```
> p <- c(b = 0.5, a = 0.02, e = 0.1, s = 0.2)
> t <- c(0, 1e+10)
```

Now we run the function.

```
> out <- ode(y = c(45, 200), t, EV, parms = p, rootfun = rootfun,
+     method = "lsodar")
> out[, ]
```

```
      time   1     2
[1,]   0.0  45 200.0
[2,] 500.8 100  12.5
```

Here we see that the steady state population sizes are $V = 100$ and $E = 12.5$, and that given our starting point, this steady state was achieved at $t = 500.8$. Other information is available; see `?lsodar` after loading the `deSolve package`.

B.11 Numerical Optimization

We frequently have a function or a model that we think can describe a pattern or process, but we need to "play around with" the numerical values of the constants in order to make the right shape with our function/model. That is, we need to find the value of the constant (or constants) that create the "best" representation of our data. This problem is known as *optimization*.

Optimization is an entire scientific discipline (or two). It boils down to quickly and efficiently finding parameters (i.e. constants) that meet our criteria. This is what we are doing when we "do" statistics. We fit models to data by telling the computer the structure of the model, and asking it to find values of the constants that minimize the residual error.

Once you have a model of the reality you want to describe, the basic steps toward optimization we consider are (i) create an *objective function*, (ii) use a routine to *minimize* (or *maximize*) the objective function through optimal choice of parameter values, and (iii) see if the "optimal" parameters values make sense, and perhaps refine and interpret them.

An *objective function* compares the data to the predicted values from the model, and returns a quantitative measure of their difference. One widely used objective function the *least-squares criterion*, that is, the objective function is the average or the sum of the squared deviations between the model values and the data — just like a simple ANOVA might. An optimization routine then tries to find model parameters that minimize this criterion.

Another widely used objective function is the likelihood function, or *maximum likelihood*. The likelihood function uses a probability distribution of our choice (often the normal distribution). The objective function then calculates the collective probability of observing those data, given the parameters and fit of the model. In other words, we pretend that the model and the predicted values are true, measure how far off each datum is from the predicted value, and then use a probability distribution to calculate the probability of seeing each datum. It then multiplies all those probabilities to get the *likelihood* of

observing those data, given the selected parameters. An optimization routine then tries to find model parameters that maximize this likelihood. In practice, it is more computationally stable to calculate the negative of the sum of the logarithms of the probabilities, and try to minimize that quantity, rather than maximize the likelihood — but in principle they are they same thing.

Once we have an objective function, an optimization routine makes educated guesses regarding good values for the parameters, until it finds the best values it can, those which minimize the objective function. There are a great variety of optimization routines, and they all have their strengths. One important tradeoff they exhibit is that the fastest and most accurate methods are sometimes the least able to handle difficult data [13]. Below, we rely on a combination to take advantage of the strengths of each type.

Here we introduce two of R's general purpose functions in the `base` package, `optimize` and `optim`, and another, in the `bbmle` package, `mle2` [13]. The function `optimize` should be used where we are in search of one parameter; use others when more than one parameter is being optimized. There are many other optimization routines in R, but we start here[4].

Here we start with one of R's general optimization functions, the one designed for finding a single parameter. Let us find the mean ($\bar{x}$) of some data through optimization. We will start with data, `y`, and let our conceptual model of that data be μ, the mean. We then create a *objective function* whose output will get smaller as the parameter of our model approaches the value we want. Poughly speaking, the mean is the value that minimizes the total difference between all the data and the itself. We will use the least-squares criterion, where the sum of all the squared deviations reaches a minimum when μ approaches the mean.

```
> y <- c(1, 0:10)
> f <- function(y, mu) {
+     sum((y - mu)^2)
+ }
```

Our function, `f`, subtracts μ from each value of y, squares each of these differences, and then sums these squared differences, to get the sum of squares. Our goal is to minimize this. If we guessed at it by hand, we would get this (Fig. B.3).

```
> guesses <- seq(4, 6, by = 0.05)
> LS.criterion <- sapply(guesses, function(mu) f(mu = mu,
+     y = y))
> plot(guesses, LS.criterion, type = "l")
```

Fig. B.3 shows us that the minimum of the objective function occurs when `mu` is a little over 4.5. Now let's let R minimize our least squared deviations. With `optimize`, we provide the function first, we then provide a range of possible values for the parameter of interest, and then give it the values of parameters or data used by the function, other than the parameter we want to fit.

[4] Indeed, all statistical models are fancy optimization routines.

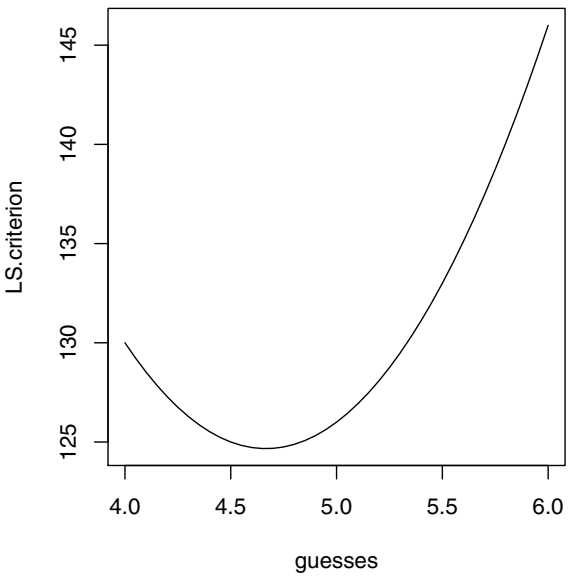


Fig. B.3: Illustration of the least squares criterion. Our objective function returns (i.e. generates) the squared deviations between the fitted model and the data. Optimization minimizes the criterion ("LS.criterion") and thereby finds the right guess (x axis).

```
> (results <- optimize(f, c(0, 10), y = y))

$minimum
[1] 4.667

$objective
[1] 124.7
```

We see that `optimize` returns two components in a list. The first is called `minimum`, which is the parameter value that causes our function `f` to be at a minimum. The second component, `objective` is the value of `f` when mu = 4.667.

Next we demonstrate `mle2`, a function for maximum likelihood estimation. Maximum likelihood relies on probability distributions to find the probability of observing a particular data set, assuming the model is correct. This class of optimization routines finds the parameters that maximize that probability.

Let us solve the same problem as above. For the same data, `y`, we create a maximum likelihood function to calculate the mean. In maximum likelihood, we actually minimize the negative logarithm of the likelihood because it is more computationally stable — the same parameters that minimize the negative log-likelihood also maximize the likelihood. We assume that the data are normally distributed, so it makes sense to assume that the probabilities derive from the normal probability density function.

```
> LL <- function(mu, SD) {
+     -sum(dnorm(y, mean = mu, sd = SD, log = TRUE))
+ }
```

This objective function calculates the negative logarithm of the probability density of each datum, given a particular mean and standard deviation, `mu`, `SD`. The optimization routine, `mle2`, then finds `mu` and `SD` that minimize the negative log-likelihood of those data.

```
> library(bbmle)
> (fit <- mle2(LL, start = list(mu = 5, SD = 1), control = list(maxit = 10^5)))

Call:
mle2(minuslogl = LL, start = list(mu = 5, SD = 1), control = list(maxit = 10^5))

Coefficients:
   mu    SD
4.667 3.223

Log-likelihood: -31.07
```

Another way to put this objective function into `mle2` is with a formula interface.

```
> mle2(y ~ dnorm(mu, sd = SD), start = list(mu = 1,
+     SD = 2))

Call:
mle2(minuslogl = y ~ dnorm(mu, sd = SD), start = list(mu = 1,
    SD = 2))

Coefficients:
   mu    SD
4.667 3.223

Log-likelihood: -31.07
```

We can examine this more closely, examing the probablities associated with the profile confidence intervals.

```
> summary(fit)

Maximum likelihood estimation

Call:
mle2(minuslogl = LL, start = list(mu = 5, SD = 1), control = list(maxit = 10^5))

Coefficients:
   Estimate Std. Error z value   Pr(z)
mu    4.667      0.930    5.02 5.3e-07
SD    3.223      0.658    4.90 9.6e-07

-2 log L: 62.14

> pr <- profile(fit)
```

```
> par(mar = c(5, 4, 3, 2))
> plot(pr)
```

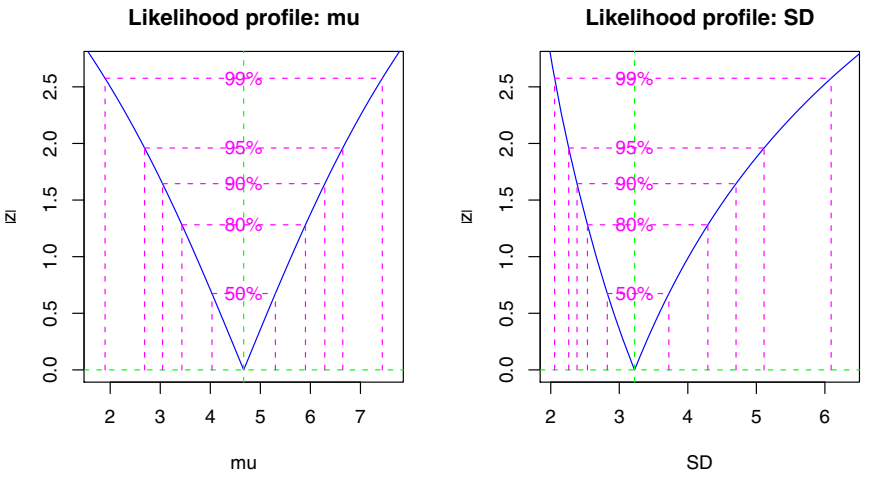


Fig. B.4: Profile confidence intervals for various limits, based on `mle2`.

Often we have reason to limit parameters to particular bounds. Most often, we may need to ensure that a parameter is greater than zero, or less than zero, or less often between zero and one. Sometimes we have a rationale based on physical or biological constraints that will limit a parameter within particular values.

To constrain parameters, we could use a routine that applies constraints directly (see particular optimization methods under `mle2`,`nlminb`, and `optim`). We could also transform the parameters, so that the optimizer uses the transformed version, while the ODE model uses the parameters in their original units. For instance, we could let the optimizer find the best value of a logarithm of our parameter that allows the original parameter to make model predictions that fit the data. Another consequence of using logarithms, rather than the original scale is that it facilitates computational procedures in estimating vary large and very small numbers. An example helps make this clear — see an extended example in Chap. 6, on disease models.

B.12 Derivatives

We can use `deriv` and `D` to have R provides derivatives. First we supply an expression, and then we get gradients.

```
> host1 <- expression(R * H * (1 + a * P)^-k)
> D(host1, "H")
```

```
R * (1 + a * P)^-k
```

B.13 Graphics

R is well known for its graphics capabilities, and entire books have been written of the subject(s). For beginners, however, R can be frustrating when compared to the point-and-click systems of most graphics "packages." This frustration derives from two issues. First, R's graphics have of a learning curve, and second, R requires us to type in, or code, our specifications. The upsides of these are that R has infinite flexibility, and total replicability, so that we get exactly the right figure, and the same figure, every time we run the same code.

B.13.1 `plot`

The most used graphics function is `plot`. Here I demonstrate several uses.

First let's just create the simplest scatterplot (Fig. B.5a).

```
> data(trees)
> attach(trees)
> plot(Girth, Height)
```

To this we can add a huge variety of variation, using arguments to `plot`.

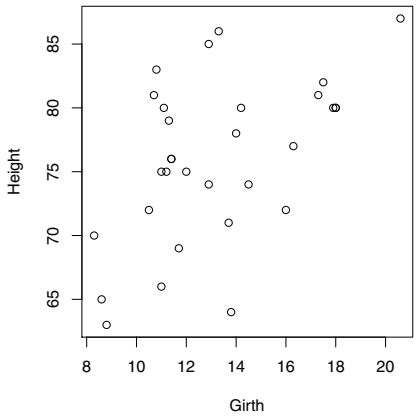
B.13.2 Adding points, lines and text to a plot

After we have started a plot, we may want to add more data or information. Here set up a new graph without plotting points, add text at each point, then more points, a line and some text.

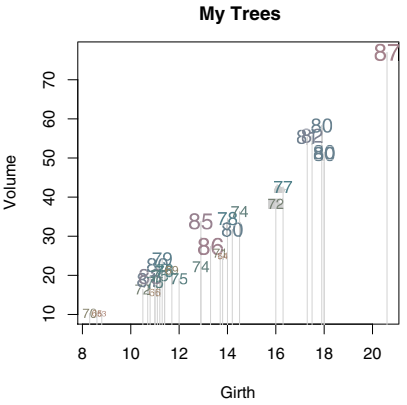
```
> par(mar = c(5, 4, 3, 2))
> plot(Girth, Volume, type = "n", main = "My Trees")
> points(Girth, Volume, type = "h", col = "lightgrey",
+     pch = 19)
```

Now we want to add points for these data, using the tree heights as the plotting symbol. We are going to use an alternate coloring system, designed with human perception in mind (`hcl`). We scale the colors so that the hue varies between 30 and 300, depending on the height of the tree; I allow the symbols to be transparent (90% opaque) overlapping. I also allow the size of the numbers to vary with height (`cex = 0.5 + hts`) Last, we add a legend (Fig. B.5b).

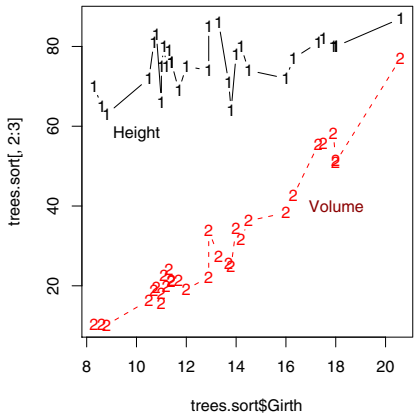
```
> hts <- (Height - min(Height))/max(Height - min(Height))
> my.colors <- hcl(h = 30 + 270 * hts, alpha = 0.9)
> text(Girth, Volume, Height, col = my.colors, cex = 0.5 +
+     hts)
```



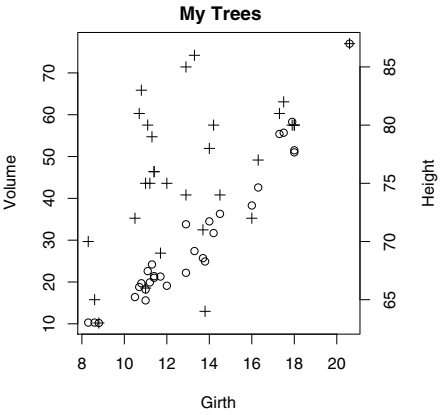
(a) Simple



(b) With Colors



(c) Two Y vars.



(d) Two Y axes

Fig. B.5: See code for graphics parameters used to generate these plots. Fig. (b) uses an alternate color scheme that provides human perception-adjusted hsv (hue, saturation, and value) specification.

B.13.3 More than one response variable

We often plot more than one response variable on a single axis. We could use `lines` or `points` to add each additional variable. We could also use `matplot` to plot a matrix of variables *vs.* one predictor (Fig. B.5c).

```
> trees.sort <- trees[order(trees$Girth, trees$Height),
+     ]
> matplot(trees.sort$Girth, trees.sort[, 2:3], type = "b")
> text(18, 40, "Volume", col = "darkred")
> text(10, 58, "Height")
```

Table B.1: Commonly used arguments to `plot`. See help pages at `?plot` and `?plot.default` for more information.

Argument	Meaning
`type`	Determines the type of X-Y plot, for example **p**, **l**, **s**, for points, lines, stair-step, and none, respectively. "None" is useful for setting up a plotting region upon which to elaborate (see example below). Defaults to **p**; see ?plot.default for other types.
`axes`	Indicates whether to plot the axes; defaults to TRUE. Useful if you want more control over each axis by using the `axis` function separately (see below).
`pch`	Point character (numeric value, 1–21). This can be a single value for an entire plot, or take on a unique value for each point, or anything in between. Defaults to 1. *To add text more than one character in length, for instance a species name, we can easily add text to a plot at each point (see the next section).*
`lty`	Line type, such as solid (1), dashed (2), etc. Defaults to 1.
`lwd`	Line width (numeric value usually 0.5–3; default is 1).
`col`	Color; can be specified by number (e.g., 2), or character (e.g. "red"). Defaults to 1 ("black"). R has tremendous options for color; see `?hcl`.
`main, ylab, xlab`	Text for main title, or axis labels.
`xlim, ylim`	Limits for x and y axes, e.g. `ylim=c(0, 1.5)` sets the limits for the y-axis at zero and 1.5. Defaults are calcualted from the data.
`log`	Indicates which axes should use a (natural) logarithm scale, e.g. `log = 'xy'` causes both axes to use logarithmic scales.

We frequently want to add a second y-axis to a graph that has a different scale (Fig. B.5d). The trick we use here is that we plot a graph, but then tell R we want to do the next command "*... as if it was on a new device*"[5] while it really is not. We overlay what we just did with new stuff, without clearing the previous stuff.

For our example, let's start with just X and our first Y. Note we also specify extra margin space room on the right hand side, preparing for the second Y axis.

```
> quartz(, 4, 4)
> par(mar = c(5, 4, 2, 4))
> plot(Girth, Volume, main = "My Trees")
```

Now we try our trick. We draw a new plot "as if" it were a new graph. We use the same X values, and the new Y data, and we also specify no labels. We also use a different line type, for clarity.

```
> par(new = TRUE)
> plot(Girth, Height, axes = FALSE, bty = "n", xlab = "",
+     ylab = "", pch = 3)
```

[5] From the `par` help page.

Now we put the new Y values on the fourth side, the right hand Y axis. We add a Y axis label using a function for *marginal text* (Fig. B.5d).

```
> axis(4)
> mtext("Height", side = 4, line = 3)
```

```
> par(mar = c(5, 4, 2, 4))
> plot(Girth, Volume, main = "My Trees")
> par(new = TRUE)
> plot(Girth, Height, axes = FALSE, bty = "n", xlab = "",
+     ylab = "", pch = 3)
> axis(4)
> mtext("Height", side = 4, line = 3)
```

B.13.4 Controlling Graphics Devices

When we make a graph with the `plot` function, or other function, it will typically open a graphics window on the computer screen automatically; if we desire more control, we can use several functions to be more deliberate. We create new graphics "devices" or graphs in several ways, including the functions `windows()` (Microsoft Windows OS), `quartz()` (Mac OS), `x11()` (X11 Window system). For instance, to open a "graphics device" on a Mac computer that is 5 inches wide and 3 inches tall, we write

```
> quartz(width = 5, height = 3)
```

To do the same thing on a computer running Windows, we type

```
> windows(width = 5, height = 3)
```

To control the *par*ameters of the graph, that is, what it looks like, aside from data, we use arguments to the `par` function. Many of these arguments refer to *sides* of the graph. These a numbered 1–4 for the bottom X axis, the left side Y axis, the top, and the right side Y axis. Arguments to `par` are many (see `?par`), and include the following.

`mar` controls the width of margins on each side; units are number of lines of text; defaults to c(5, 4, 4, 2) + 0.1, so the bottom has the most room, and the right hand side has the least room.

`mgp` controls the spacing of the axis title, labels and the actual line itself; units of number of lines of text, and default to c(3, 1, 0), so the axis title sits three lines away from the edge of the plotting region, the axis labels, one line away and the axis line sits at the edge of the plotting region.

`tcl` tick length, as a fraction of the height of a line of text; negative values put the tick marks outside, positive values put the tick marks inside. Defaults to -0.5.

We can build each side of the graph separately by initiating a graph but not plotting axes `plot(..., axes = FALSE)`, and then adding the axes separately. For instance, `axis(1)` adds the bottom axis.

Last, we can use `layout` to make graph with several smaller subgraphs (see also (`mfrow` and `mfcol` arguments to `par` and the function `split.screen`). The

function `layout` takes a matrix as its argument, the matrix contains a sequence of numbers that tells R how to fill the regions Graphs can fit in more than one of these regions if indicated by the same number.

Here we create a compound graphic organized on top of a 4×4 grid; it will have two rows, will be be filled in by rows. The first graph will be the upper left, the second the upper right, and the third will fill the third and fourth spots in the second. We will fill each with a slightly different plot of the same data (Fig. B.6).

```
> quartz(, 5, 5)
> layout(matrix(c(1, 2, 3, 3), nrow = 2, byrow = TRUE))
> plot(Girth, Height)
```

Now we add the second and third ones but with different settings.

```
> par(mar = c(3, 3, 1, 1), mgp = c(1.6, 0.2, 0), tcl = 0.2)
> plot(Girth, Height)
> par(mar = c(3, 3, 2, 1), mgp = c(1.6, 0.2, 0), tcl = 0.2)
> plot(Girth, Height, axes = FALSE, xlim = c(8, 22))
> axis(1, tcl = -0.3)
> axis(2, tick = F)
> rug(Height, side = 2, col = 2)
> title("A Third, Very Wide, Plot")
```

B.13.5 Creating a Graphics File

Now that you have made this beautiful thing, I suppose you would like to stick it into a manuscript. One way to get graphics out of R and into something else (presentation software, a manuscript), is to create a graphics device, and then save it with `dev.print` in a format that you like, such as PDF, postscript, PNG, or JPEG.

For instance, we might do this to save a graphics file in our working directory.

```
> getwd()
> quartz(, 4, 4)
> plot(Height, Volume, main = "Tree Data")
> dev.print(pdf, "MyTree.pdf")
```

This should have saved a small PDF figure in your current working directory, returned by `getwd`.

You will have to find your own way to make graphics files that suits your operating system, your preferred applications, and your personality.

B.14 Graphical displays that show distributions

Here we take a quick look at ways to reveal distributions of data. First, two views to see in the Console, a six number summary of quantiles and the mean, and the good ol' stem and leaf plot, a favorite of computational botanists everywhere.

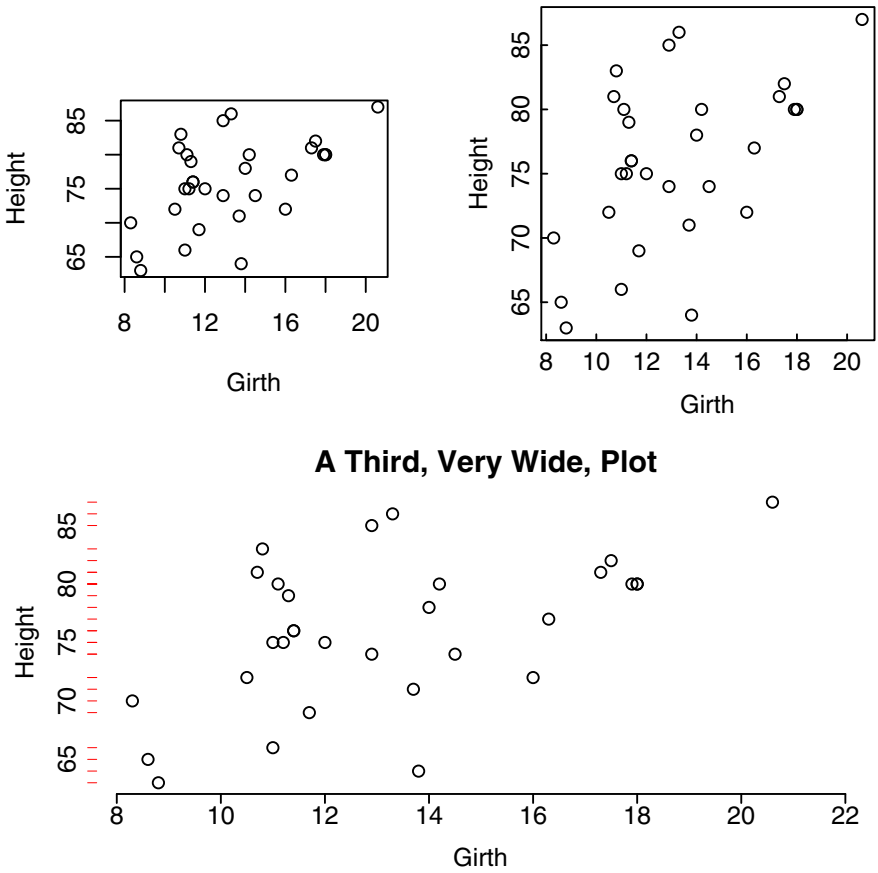


Fig. B.6: A variety of examples with different graphics *par*ameters.

```
> summary(Girth)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    8.3    11.0    12.9    13.2    15.2    20.6

> stem(Girth)

  The decimal point is at the |

   8 | 368
  10 | 57800123447
  12 | 099378
  14 | 025
  16 | 03359
  18 | 00
  20 | 6
```

Here we will create 4 various plots revealing different ways to look at your data, each with a couple bells and whistles. For kicks, we put them into a single compound figure, in a "layout" composed of a matrix of graphs.

```
> layout(matrix(c(1, 2, 2, 3, 4, 4), nrow = 2, byrow = TRUE))
> plot(1:length(Girth), Girth, xlab = "Order of Sample Collection?")
> hist(Girth, prob = TRUE)
> rug(Girth)
> lines(density(Girth))
> boxplot(Girth, main = "Boxplot of Girth")
> points(jitter(rep(1, length(Girth))), Girth)
> qqnorm(log(Girth))
> qqline(log(Girth))
> title(sub = "Log transformed data")
```

B.15 Eigenanalysis

Performing eigenanalysis in Ris easy. We use the `eigen` function which returns a list with two components. The first named component is a vector of eigenvalues and the second named component is a matrix of corresponding eigenvectors. These will be numeric if possible, or complex, if any of the elements are complex numbers.

Here we have a typical demographic stage matrix.

```
> A <- matrix(c(0, 0.1, 10, 0.5), nrow = 2)
> eig.A <- eigen(A)
> str(eig.A)

List of 2
 $ values : num [1:2] 1.28 -0.78
 $ vectors: num [1:2, 1:2] -0.9919 -0.127 -0.997 0.0778
```

Singular value decomposition (SVD) is a generalization of eigenanalysis and is used in Rfor some applications where eigenanalysis was used historically, but where SVD is more numerically accurate (`prcomp` for principle components analysis).

B.16 Eigenanalysis of demographic versus Jacobian matrices

Eigenanalyses of demographic and Jacobian matrices are worth comparing. In one sense, they have similar meanings — they both describe the asymptotic (long-term) properties of a system, either population size (demographic matrix) or a perturbation at an equilibrium. The quantitative interpretation of the eigenvalues will therefore differ.

In the case of the stage (or age) structured demographic model, the elements of the demographic matrix are discrete per capita increments of change over a

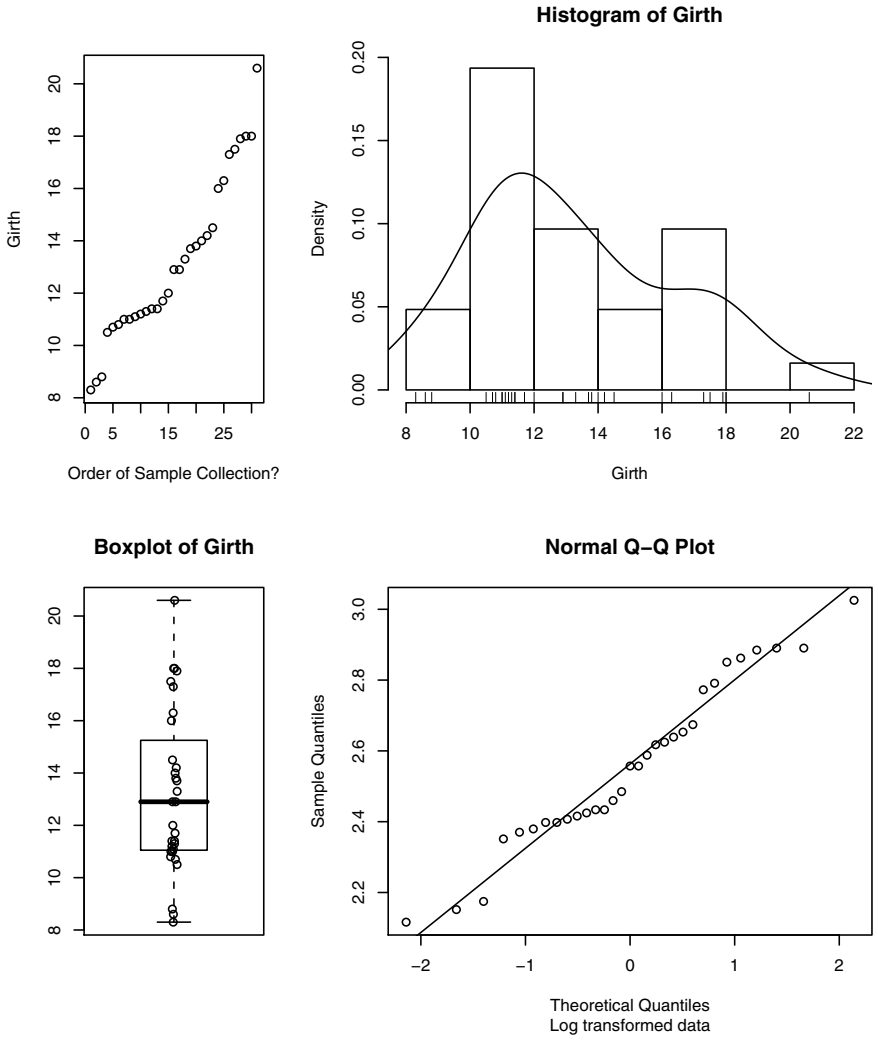


Fig. B.7: Examples of ways to look at the distribution of your data. See `?hist`, for example, for more information.

specified time interval. This is directly analogous to the finite rate of increase, λ, in discrete unstructured models. (Indeed, an unstructured discrete growth model is a stage-structured model with one stage). Therefore, the eigenvalues of a demographic matrix will have the same units — a per capita increment of change. That is why the dominant eigenvalue has to be greater than 1.0 for the population to increase, and less than 1 (not merely less than zero) for the population to decline.

In the case of the Jacobian matrix, comprised of continuous partial differential equations, the elements are per capita *instantaneous* rates of change. As differential equations, they describe the *instantanteous* rates of change, analogous to r. Therefore, values greater than zero indicate increases, and values less than zero indicate decreases. Because these rates are evaluated at an equilibrium, the equilibrium acts like a new zero — positive values indicate growth away from the equilibrium, and negative values indicate shrinkage back toward the equilibrium. When we evaluate these elements at the equilibrium, the numbers we get are in the same units as r, where values greater than zero indicate increase, and values less than zero indicate decrease. The change they describe is the instantaneous per capita rate of change of each population with respect to the others. The eigenvalues summarizing all of the elements Jacobian matrix thus must be less than zero for the disturbance to decline.

So, in summary, the elements of a demographic matrix are discrete increments over a real time interval. Therefore its eigenvalues represent relative per capita growth rates a discrete time interval, and we interpret the eigenvalues with respect to 1.0. On the other hand, the elements of the Jacobian matrix are instantaneous per captia rates of change evaluated at an equilibrium. Therefore its eigenvalues represent the per capita *instantaneous* rates of change of a tiny perturbation at the equilibrium. We interpret the eigenvalues with respect to 0 indicating whether the perturbation grows or shrinks.

B.17 Symbols used in this book

I am convinced that one of the biggest hurdles to learning theoretical ecology — and the one that is easiest to overcome — is to be able to "read" and hear them in your head. This requires being able to pronounce Greek symbols. Few of us learned how to pronounce "α" in primary school. Therefore, I provide here an incomplete simplistic American English pronunciation guide for (some of) the rest of us, for symbols in this book. Only a few are tricky, and different people will pronounce them differently.

Table B.2: Symbols and their pronunciation; occasional usage applies to lowercase, unless otherwise specified. A few symbols have common variants. Any symbol might be part of any equation; ecologists frequently ascribe other meanings which have to be defined each time they are used. See also http://en.wikipedia.org/wiki/Greek_letters

Symbol	Spelling	Pronunciation; occasional or conventional usage
A, α	alpha	al'-fa; point or local diversity (or a parameter in the logseries abundance distribution)
B, β	beta	bay'-ta; turnover diversity
Γ, γ	gamma	gam'-ma; regional diversity
Δ, δ, ∂	delta	del'-ta; change or difference
E, ϵ, ε	epsilon	ep'-si-lon; error
Θ, θ	theta	thay'-ta ("th" as in "thanks"); in neutral theory, biodiversity.
Λ, λ	lambda	lam'-da; eigenvalues, and finite rate of increase
M, μ	mu	meeoo, myou; mean
N, ν	nu	noo, nou
Π, π	pi	pie; uppercase for product (of the elements of a vector)
P, ρ	rho	row (as in "a boat"); correlation
Σ, σ, ς	sigma	sig'-ma; standard deviation (uppercase is used for summation)
T, τ	tau	(sounds like what you say when you stub your toe - "Ow!" but with a "t").
Φ, ϕ	phi	fie, figh
X, χ	chi	kie, kigh
Ψ, ψ	psi	sie, sigh
Ω, ω	omega	oh-may'-ga; degree of omnivory

References

1. D. Alonso, R. S. Etienne, and A. J. McKane. The merits of neutral theory. *Trends in Ecology & Evolution*, 21:451–457, 2006.
2. D. Alonso and A. J. McKane. Sampling hubbell's neutral theory of biodiversity. *Ecology Letters*, 7:901–910, 2004.
3. J. Antonovics and H.M. Alexander. Epidemiology of anther-smut infection of *Silene alba* (= *S. latifolia*) caused by *Ustilago violacea* – patterns of spore deposition in experimental populations. *Proceedings of the Royal Society B-Biological Sciences*, 250:157–163, 1992.
4. R. A. Armstrong. Fugitive species: Experiments with fungi and some theoretical considerations. *Ecology*, 57:953–963, 1976.
5. O. Arrhenius. Species and area. *Journal of Ecology*, 9:95–99, 1921.
6. J. P. Bakker and F. Berendse. Constraints in the restoration of ecological diversity in grassland and heathland communities. *Trends in Ecology & Evolution*, 14:63–68, 1999.
7. F. A. Bazzaz. *Plants in Changing Environments*. Cambridge University Press, Boston, 1996.
8. L. Becks, F. M. Hilker, H. Malchow, K. Jurgens, and H. Arndt. Experimental demonstration of chaos in a microbial food web. *Nature*, 435:1226–1229, 2005.
9. G. Bell. The distribution of abundance in neutral communities. *The American Naturalist*, 155:606–617, 2000.
10. G. Bell. Neutral macroecology. *Science*, 293(5539):2413–2418, 2001.
11. E. L. Berlow, A. M. Neutel, J. E. Cohen, P. C. de Ruiter, B. Ebenman, M. Emmerson, J. W. Fox, V. A. A. Jansen, J. I. Jones, G. D. Kokkoris, D. O. Logofet, A. J. McKane, J. M. Montoya, and O. Petchey. Interaction strengths in food webs: Issues and opportunities. *Journal of Animal Ecology*, 73:585–598, 2004.
12. E. J. Berry, D. L. Gorchov, B. A. Endress, and M. H. H. Stevens. Source-sink dynamics within a plant population: the impact of substrate and herbivory on palm demography. *Population Ecology*, 50:63–77, 2008.
13. B. M. Bolker. *Ecological Models and Data in R*. Princeton University Press, Princeton, NJ, 2008.
14. B. M. Bolker, S. W. Pacala, and C. Neuhauser. Spatial dynamics in model plant communities: What do we really know? *American Naturalist*, 162:135–148, 2003.
15. J.H. Brown. *Macroecology*. The University of Chicago Press, Chicago, 1995.
16. J.H. Brown and D.W. Davidson. Competition between seed-eating rodents and ants in desert ecosystems. *Science*, 196:880–882, 1977.

17. J.H. Brown and A. Kodric-Brown. Turnover rate in insular biogeography: effect of immigration on extinction. *Ecology*, 58:445–449, 1977.
18. K P. Burnham and D. R. Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer, New York, 2002.
19. J. E. Byers, K. Cuddington, C. G. Jones, T. S. Talley, A. Hastings, J. G. Lambrinos, J. A. Crooks, and W. G. Wilson. Using ecosystem engineers to restore ecological systems. *Trends in Ecology & Evolution*, 21:493–500, 2006.
20. C.E. Caceres. Temporal variation, dormancy, and coexistence: A field test of the storage effect. *Proceedings of the National Academy of Sciences, U.S.A.*, 94:9171–9175, 1997.
21. T. J. Case. *An Illustrated Guide to Theoretical Ecology*. Oxford University Press, Inc., New York, 2000.
22. H. Caswell. Community structure: A neutral model analysis. *Ecological Monographs*, 46:327–354, 1976.
23. H. Caswell. *Matrix Population Models: Construction, Analysis, and Interpretation*. Sinauer Associates, Inc., Sunderland, MA, USA, 2nd edition, 2001.
24. J. M. Chase and M. Leibold. *Ecological Niches: Linking Classical and Contemporary Approaches*. University of Chicago Press, Chicago, 2003.
25. X. Chen and J. E. Cohen. Global stability, local stability and permanence in model food webs. *Journal of Theoretical Biology*, 212:223–235, 2001.
26. P. L. Chesson. Multispecies competition in variable environments. *Theoretical Population Biology*, 45:227–276, 1994.
27. P. L. Chesson. General theory of competitive coexistence in spatially-varying environments. *Theoretical Population Biology*, 58:211–237, 2000.
28. P. L. Chesson. Mechanisms of maintenance of species diversity. *Annual Review of Ecology and Systematics*, 31:343–366, 2000.
29. P. L. Chesson. Quantifying and testing coexistence mechanisms arising from recruitment fluctuations. *Theoretical Population Biology*, 64:345–357, 2003.
30. P. L. Chesson and R. R. Warner. Environmental variability promotes coexistence in lottery competitive systems. *The American Naturalist*, 117:923–943, 1981.
31. J. S. Clark, S. LaDeau, and I. Ibanez. Fecundity of trees and the colonization-competition hypothesis. *Ecological Monographs*, 74:415–442, 2004.
32. J. S. Clark and J. S. McLachlan. Stability of forest biodiversity. *Nature*, 423:635–638, 2003.
33. J. E. Cohen. Human population: The next half century. *Science*, 302:1172–1175, 2003.
34. J.E. Cohen, F. Briand, and C.M. Newman. *Community Food Webs: Data and Theory*, volume 20 of *Biomathematics*. Springer-Verlag, Berlin, 1990.
35. S. L. Collins and S. M. Glenn. Importance of spatial and temporal dynamics in species regional abundance and distribution. *Ecology*, 72:654–664, 1991.
36. R. Condit, N. Pitman, E. G. Leigh, J. Chave, J. Terborgh, R. B. Foster, P. Nunez, S. Aguilar, R. Valencia, G. Villa, H. C. Muller-Landau, E. Losos, and S. P. Hubbell. Beta-diversity in tropical forest trees. *Science*, 295:666–669, 2002.
37. J.H. Connell. Diversity in tropical rain forests and coral reefs. *Science*, 199:1302–1310, 1978.
38. J.H. Connell and W.P. Sousa. On the evidence needed to judge ecological stability or persistence. *The American Naturalist*, 121:789–824, 1983.
39. R.F. Constantino, J.M. Cushing, B. Dennis, and R.A. Desharnais. Experimentally induced transitions in the dynamic behaviour of insect populations. *Nature*, 375:227–230, 1995.
40. J. M. Craine. Reconciling plant strategy theories of grime and tilman. *Journal of Ecology*, 93:1041–1052, 2005.

41. M. J. Crawley and J. E. Harral. Scale dependence in plant biodiversity. *Science*, 291:864–868, 2001.
42. T. O. Crist and J. A. Veech. Additive partitioning of rarefaction curves and species-area relationships: Unifying alpha-, beta- and gamma-diversity with sample size and habitat area. *Ecology Letters*, 9:923–932, 2006.
43. T. O. Crist, J. A. Veech, J. C. Gering, and K. S. Summerville. Partitioning species diversity across landscapes and regions: A hierarchical analysis of alpha, beta, and gamma diversity. *American Naturalist*, 162:734–743, 2003.
44. D. T. Crouse, L. B. Crowder, and H. Caswell. A stage-based population model for loggerhead sea turtles and implications for conservation. *Ecology*, 68:1412–1423, 1987.
45. J. F. Crow and M. Kimura. *An Introduction to Population Genetics Theory.* Harper & Row, New York, 1970.
46. A. C. Davison and D. V. Hinkley. *Bootstrap Methods and Their Applicaiton.* Cambridge Series on Statistical and Probabilistic Mathematics. Cambridge University Press, New York, 1997.
47. J. A. Dunne, R. J. Williams, and N. D. Martinez. Food-web structure and network theory: The role of connectance and size. *Proceedings of the National Academy of Sciences of the United States of America*, 99:12917–12922, 2002.
48. S. P. Ellner and J. Guckenheimer. *Dynamic Models in Biology.* Princeton University Press, Princeton, NJ, 2006.
49. S. P. Ellner and P. Turchin. When can noise induce chaos and why does it matter: A critique. *Oikos*, 111:620–631, 2005.
50. B. A. Endress, D. L. Gorchov, and R. B. Noble. Non-timber forest product extraction: Effects of harvest and browsing on an understory palm. *Ecological Applications*, 14:1139–1153, 2004.
51. R. S. Etienne. A new sampling formula for neutral biodiversity. *Ecology Letters*, 8:253–260, 2005.
52. R. S. Etienne. A neutral sampling formula for multiple samples and an 'exact' test of neutrality. *Ecology Letters*, 10:608–618, 2007.
53. W.J. Ewens. *Mathematical Population Genetics, I. Theoretical Introduction*, volume 27 of *Interdisciplinary Applied Mathematics.* Springer, New York, 2nd ed. edition, 2004.
54. J.M. Facelli, P. Chesson, and N. Barnes. Differences in seed biology of annual plants in arid lands: A key ingredient of the storage effect. *Ecology*, 86:2998–3006, 2005.
55. R. A. Fisher, A. S. Corbet, and C. B. Williams. The relation between the number of species and the number of individuals in a random sample of an animal population. *Jounral of Animal Ecology*, 12:42–58, 1943.
56. G. F. Fussmann, S. P. Ellner, N. G. Hairston, L. E. Jones, K. W. Shertzer, and T. Yoshida. Ecological and evolutionary dynamics of experimental plankton communities. *Advances in Ecological Research*, 37:221–243, 2005.
57. D. L. Gorchov and D. Christensen. Projecting the effect of harvest on the population biology of goldenseal, *Hydrasts canadensis* l., a medicianl herb in Ohio, USA forests. In New York Botanical Garden Society for Economic Botany, editor, *Symposium on Assessing Local Management of Wild Plant Resources: Approaches in Population Biology*, 2002.
58. N. J. Gotelli. *A Primer of Ecology.* Sinauer Associates, Inc., Sunderland, MA, 3rd ed. edition, 2001.
59. N. J. Gotelli and R. K. Colwell. Quantifying biodiversity: procedures and pitfalls in the measurement and comparison of species richness. *Ecology Letters*, 4:379–391, 2001.

60. N. J. Gotelli and G. R. Graves. *Null Models in Ecology*. Smithsonian Institution Press, Washington, DC, 1996.
61. N. J. Gotelli and B. J. McGill. Null versus neutral models: what's the difference? *Ecography*, 29:793–800, 2006.
62. N.G. Gotelli and W.G. Kelley. A general model of metapopulation dynamics. *Oikos*, 68:36–44, 1993.
63. N.J. Gotelli. Metapopulation models: the rescue effect, the propagule rain, and the core-satellite hypothesis. *The American Naturalist*, 138:768–776, 1991.
64. J. L. Green and J. B. Plotkin. A statistical theory for sampling species abundances. *Ecology Letters*, 10:1037–1045, 2007.
65. K. L. Gross and P. A. Werner. Colonizing abilities of "biennial" plant species in relation to ground cover: implications for their distriubtions in a successional sere. *Ecology*, 63:921–931, 1982.
66. N. G. Hairston, Sr. *Ecological Experiments: Purpose, Design, and Execution*. Cambridge Studies in Ecology. Cambridge University Press, Cambridge, UK, 1991.
67. J.M. Halley. Ecology, evolution and 1/f-noise. *Trends in Ecology & Evolution*, 11:33–37, 1996.
68. P. A. Hamback, K. S. Summerville, I. Steffan-Dewenter, J. Krauss, G. Englund, and T. O. Crist. Habitat specialization, body size, and family identity explain lepidopteran density-area relationships in a cross-continental comparison. *Proceeding of the National Academy of Sciences, USA*, 104:8368–8373, 2007.
69. Hankin, R.K.S. *untb: ecological drift under the UNTB*, 2007. R package version 1.3-3.
70. I. Hanski. Dynamics of regional distribution: The core and satellite species hypothesis. *Oikos*, 38:210–221, 1982.
71. J. Harte, T. Zillio, E. Conlisk, and A. B. Smith. Maximum entropy and the state-variable approach to macroecology. *Ecology*, 89:2700–2711, 2008.
72. M.P. Hassell. *The Dynamics of Arthropod Predator-Prey Systems*, volume 13 of *Monographs in Population Biology*. Princeton University Press, Princeton, 1978.
73. A. Hastings. Disturbance, coexistence, history and competition for space. *Theoretical Population Biology*, 18:363–373, 1980.
74. A. Hastings. Transients: the key to long-term ecological understanding? *Trends in Ecology and Evolution*, 19:39–45, 2004.
75. A. Helm, I. Hanski, and M. Partel. Slow response of plant species richness to habitat loss and fragmentation. *Ecology Letters*, 9:72–77, 2006.
76. M. O. Hill. Diversity and evenness: a unifying notion and its consequences. *Ecology*, 54:427–432, 1973.
77. C.S. Holling. Some characteristics of simple types of predation and parasitism. *Canadian Entomologist*, 91:385, 1959.
78. C.S. Holling. Resilience and stability of ecological systems. *Annual Review of Ecology and Systematics*, 4:1–23, 1973.
79. R.D. Holt and G.A Polis. A theoretical framework for intraguild predation. *The American Naturalist*, 149:745–764, 1997.
80. H. S. Horn and R. H. MacArthur. Competition among fugitive species in a harlequin environment. *Ecology*, 53:749–752, 1972.
81. S. P. Hubbell. *A Unified Theory of Biodiversity and Biogeography*. Monographs in Population Biology. Princeton University Press, Princeton, N.J., 2001.
82. S. H. Hurlbert. The nonconcept of species diversity: A critique and alternative parameters. *Ecology*, 52:577–586, 1971.

83. G. C. Hurtt and S. W. Pacala. The consequences of recruitment limitation: Reconciling chance, history, and competitive differences between plants. *Journal of Theoretical Biology*, 176:1–12, 1995.
84. G. E. Hutchinson. *An Introduction to Population Ecology*. Yale University Press, New Haven, CT, 1978.
85. G. R. Huxel and K. McCann. Food web stability: The influence of trophic flows across habitats. *American Naturalist*, 152:460–469, 1998.
86. F. Jabot and J. Chave. Inferring the parameters of the neutral theory of biodiversity using phylogenetic information, and implications for tropical forests. *Ecology Letters*, 12:239–248, 2009.
87. S. A. Juliano. Nonlinear curve fitting: predation and functional response curves. In S. M. Scheiner and J. Gurevitch, editors, *Design and Analysis of Ecological Experiments*. Oxford University Press, 2001.
88. P. Kareiva and U. Wennergren. Connecting landscape patterns to ecosystem and population processes. *Nature*, 373:299–302, 1995.
89. C. K. Kelly and M. G. Bowler. Coexistence and relative abundance in forest trees. *Nature*, 417:437–440, 2002.
90. B. E. Kendall, J. Prendergast, and O. N. Bjornstad. The macroecology of population dynamics: Taxonomic and biogeographic patterns in population cycles. *Ecology Letters*, 1:160–164, 1998.
91. W. O. Kermack and W. G. McCormick. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society, Series A*, 115:700–721, 1927.
92. C. J. Keylock. Simpson diversity and the shannon-wiener index as special cases of a generalized entropy. *Oikos*, 109:203–207, 2005.
93. S. E. Kingsland. *Modeling Nature*. University of Chicago Press, Chicago, 1985.
94. C.J. Krebs, S. Boutin, R. Boonstra, A.R.E. Sinclair, J.N.M. Smith, M.R.T. Dale, K. Martin, and R. Turkington. Impact of food and predation on the snowshoe hare cycle. *Science*, 269:1112–1115, 1995.
95. R. Lande. Extinction thresholds in demographic models of territorial populations. *American Naturalist*, 130:624–635, 1987.
96. R. Lande. Demographic models of the northern spotted owl (*Strix occidentalis caurina*). *Oecologia*, 75:601–607, 1988.
97. R. Lande. Statistics and partitioning of species diversity, and similarity among multiple communities. *Oikos*, 76:5–13, 1996.
98. R. Lande, P. J. DeVries, and T. R. Walla. When species accumulation curves intersect: Implications for ranking diversity using small samples. *Oikos*, 89:601–605, 2000.
99. A. M. Latimer, J. A. Silander, and R. M. Cowling. Neutral ecological theory reveals isolation and rapid speciation in a biodiversity hot spot. *Science*, 309:1722–1725, 2005.
100. L. P. Lefkovitch. The study of population growth in organisms grouped by stages. *Biometrics*, 21:1–18, 1965.
101. C.L. Lehman and D. Tilman. Competition in spatial habitats. In D. Tilman and P. Kareiva, editors, *Spatial Ecology: The Role of Space in Population Dynamics and Interspecific Interactions*, pages 185–203. Princeton University Press, Princeton, NJ, 1997.
102. M. A. Leibold, M. Holyoak, N. Mouquet, P. Amarasekare, J. M. Chase, M. F. Hoopes, R. D. Holt, J. B. Shurin, R. Law, D. Tilman, M. Loreau, and A. Gonzalez. The metacommunity concept: A framework for multi-scale community ecology. *Ecology Letters*, 7:601–613, 2004.

103. M. A. Leibold and M. A. McPeek. Coexistence of the niche and neutral perspectives in community ecology. *Ecology*, 87:1399–1410, 2006.
104. M.A. Leibold. A graphical model of keystone predators in food webs: trophic regulation of abundance, incidence, and diversity patterns in communities. *The American Naturalist*, 147:784–812, 1996.
105. Jr. Leigh, E. G. *Tropical Forest Ecology: A View from Barro Colorado Island*. Oxford University Press, Oxford, UK, 1999.
106. F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In Wolfgang Härdle and Bernd Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg, 2002.
107. P. H. Leslie. On the use of matrices in certain population mathematics. *Biometrika*, 35:183–212, 1945.
108. S. A. Levin, J. E. Cohen, and A. Hastings. Dispersal strategies in a patchy environment. *Theoretical Population Biology*, 26:165–191, 1984.
109. R. Levins. The strategy of model building in population biology. *American Scientist*, 54:421–431, 1966.
110. R. Levins. Some demographic and genetic consequences of environmental heterogeneity for biological control. *Bulletin of the Entomological Society of America*, 15:237–240, 1969.
111. R. Levins and D. Culver. Regional coexistence of species and competition between rare species. *Proceeding of the National Academy of Sciences, U.S.A.*, 68:1246–1248, 1971.
112. R.C. Lewontin. The meaning of stability. In *Diversity and Stability in Ecological Systems*, volume 22 of *Brookhaven Symposia in Biology*, pages 13–24, Upton, NY, 1969. Brookhaven National Laboratory.
113. R. Lincoln, G. Boxshall, and P. Clark. *A Dictionary of Ecology, Evolution and Systematics*. Cambridge University Press, Cambridge UK, 2nd edition, 1998.
114. R. Lindborg and O. Eriksson. Historical landscape connectivity affects prresent plant species diversity. *Ecology*, 85:1840–1845, 2004.
115. Z. T. Long and I. Karel. Resource specialization determines whether history influences community structure. *Oikos*, 96:62–69, 2002.
116. M. Loreau, N. Mouquet, and R. D. Holt. Meta-ecosystems: A theoretical framework for a spatial ecosystem ecology. *Ecology Letters*, 6:673–679, 2003.
117. A. J. Lotka. *Elements of Mathematical Biology*. Dover Publications, Inc., Mineola, NY, 1956.
118. R. H. MacArthur. On the relative abundance of bird species. *Proceedings of the National Academy of Sciences of the United States of America*, 43:293–295, 1957.
119. R. H. MacArthur. Some generalized theorems of natural selection. *Proceedings of the National Academy of Sciences, USA*, 48:1893–1897, 1962.
120. R. H. MacArthur. *Geographical Ecology: Patterns in the Distribution of Species*. Harper & Row, New York, 1972.
121. R. H. MacArthur and E. O. Wilson. An equilibrium theory of insular zoogeography. *Evolution*, 17:373–387, 1963.
122. R. H. MacArthur and E. O. Wilson. *The Theory of Island Biogeography*. Monographs in Population Biology. Princeton University Press, Princeton, 1967.
123. R.H. MacArthur and E.R. Pianka. On optimal use of a patchy environment. *The American Naturalist*, 100:603–609, 1966.
124. A. E. Magurran. *Measuring Biological Diversity*. Princeton University Press, 2004.

125. T. R. Malthus. *An Essay on the Principle of Population.* J. Johnson, London, 1798.
126. B.F.J. Manly. *Randomization, Bootstrap and Monte Carlo Methods in Biology.* Chapman and Hall, London, 1997.
127. R. M. May. *Stability and Complexity in Model Ecosystems*, volume 6 of *Monographs in Population Biology.* Princeton University Press, Princeton, NJ, 1973.
128. R. M. May. Biological populations with nonoverlapping generation: stable points, stable cycles, and chaos. *Science*, 186:645–647, 1974.
129. R. M. May. *Ecology and Evolution of Communities*, chapter Patterns of species abundance and diversity, pages 81–120. Harvard University Press, Cambridge, MA, 1975.
130. R. M. May. Thresholds and multiple breakpoints in ecosystems with a multiplicity of states. *Nature*, 269:471–477, 1977.
131. R. M. May. Host-parasitoid systems in patchy environments: A phenomenological model. *Journal of Animal Ecology*, 47:833–844, 1978.
132. R. M. May. *Stability and Complexity in Model Ecosystems.* Princeton Landmarks in Biology. Princeton University Press, 2001.
133. R. M. May. Network structure and the biology of populations. *Trends in Ecology & Evolution*, 21:394–399, 2006.
134. R.M. May. Will a large complex system be stable? *Nature*, 238:413–414, 1972.
135. R.M. May. *Theoretical Ecology: Principles and Applications.* Blackwell Scientific Publications, Oxford, 1976.
136. H. McCallum, N. Barlow, and J. Hone. How should pathogen transission be modeled? *Trends in Ecology and Evolution*, 16:295–300, 2001.
137. K. McCann. Density-dependent coexistence in fish communities. *Ecology*, 79:2957–2967, 1998.
138. K. McCann and A. Hastings. Re-evaluating the omnivory - stability relationship in food webs. *Proceedings of the Royal Society of London Series B-Biological Sciences*, 264:1249–1254, 1997.
139. A. McKane, D. Alonso, and R. V. Sole. Mean-field stochastic theory for species-rich assembled communities. *Physical Review E*, 62:8466–8484, 2000.
140. M. A. McPeek and S. Kalisz. Population sampling and bootstrapping in complex designs: Demographic analysis. In S.M. Scheiner and J. Gurevitch, editors, *Design and Analysis of Ecological Experiments*, pages 232–252. Chapman & Hall, New York, 1993.
141. F. Messier. Ungulate population models with predation - a case study with the north american moose. *Ecology*, 75:478–488, 1994.
142. P. J. Morin. *Community Ecology.* Blackwell Science, Inc., Malden, MA, 1999.
143. P. J. Morin. Productivity, intraguild predation, and population dynamics in experimental food webs. *Ecology*, 80:752–760, 1999.
144. H. Morlon, G. Chuyong, R. Condit, S.P. Hubbell, D. Kenfack, D. Thomas, R. Valencia, and J. L. Green. A general framework for the distance-decay of similarity in ecological communities. *Ecology Letters*, 11:904–917, 2008.
145. I. Motomura. A statistical treatment of associations [in Japanese]. *Japanese Journal of Zoology*, 44:379–383, 1932.
146. S. Nee and R.M. May. Dynamics of metapopulations: habitat destruction and competitive coexistence. *Journal of Animal Ecology*, 61:37–40, 1992.
147. M. Nei. *Molecular Evolutionary Genetics.* Columbia University Press, New York, 1987.
148. M.G. Neubert and H. Caswell. Alternatives to resilience for measuring the responses of ecological systems to perturbations. *Ecology*, 78:653–665, 1997.

149. A.J. Nicholson and V.A. Bailey. The balance of animal populations - Part I. *Proceedings of the Zoological Society of London*, pages 551–598, 1935.
150. I. Noy-Meir. Stability of grazing systems: An application of predator-prey graphs. *The Journal of Ecology*, 63:459–481, 1975.
151. J. Oksanen, R. Kindt, P. Legendre, R. O'Hara, G. L. Simpson, P. Solymos, M. H. H. Stevens, and H. Wagner. *vegan: Community Ecology Package*, 2008. R package version 1.15-1.
152. S. W. Pacala and M. Rees. Models suggesting field experiments to test two hypotheses explaining successional diversity. *The American Naturalist*, 152:729–737, 1998.
153. S.W. Pacala. Dynamics of plant communities. In M.J. Crawley, editor, *Plant Ecology*, pages 532–555. Blackwel Science, Ltd., Oxford, UK, 1997.
154. S.W. Pacala, M.P. Hassell, and R.M. May. Host-parasitoid associations in patchy environments. *Nature*, 344:150–153, 1990.
155. O.L. Petchey. Environmental colour affects the dynamics of single species populations. *Proceedings of the Royal Society of London B*, 267:747–754, 2001.
156. R. H. Peters. Some general problems in ecology illustrated by food web theory. *Ecology*, 69:1673–1676, 1988.
157. T. Petzoldt. R as a simulation platform in ecological modelling. *R News*, 3:8–16, 2003.
158. T. Petzoldt and K. Rinke. simecol: An object-oriented framework for ecological modeling in r. *Journal of Statistical Software*, 22:1–31, 2007.
159. S.L. Pimm and J.H. Lawton. Number of trophic levels in ecological communities. *Nature*, 268:329 331, 1977.
160. S.L. Pimm and J.H. Lawton. On feeding on more than one trophic level. *Nature*, 275:542–544, 1978.
161. J. Pinheiro and D. Bates. *Mixed-effects Models in S and S-PLUS*. Springer, New York, 2000.
162. W. Platt and I. Weis. Resource partitioning and competition within a guild of fugitive prairie plants. *The American Naturalist*, 111:479–513, 1977.
163. J. B. Plotkin, M. D. Potts, D. W. Yu, S. Bunyavejchewin, R. Condit, R. Foster, S. Hubbell, J. LaFrankie, N. Manokaran, L.H. Seng, R. Sukumar, M. A. Nowak, and P. S. Ashton. Predicting species diversity in tropical forests. *Proceedings of the National Academy of Sciences, U.S.A.*, 97(20):10850–10854, 2000.
164. J.B. Plotkin and H.C. Muller-Landau. Sampling the species composition of a landscape. *Ecology*, 83:3344–3356, 2002.
165. G. A. Polis, C. A. Myers, and R. D. Holt. The ecology and evolution of intraguild predation: potential competitors that eat each other. *Annual Review of Ecology and Systematics*, 20:297–330, 1989.
166. G.A. Polis. Complex trophic interactions in deserts: an empirical critique of food web ecology. *The American Naturalist*, 138:123–155, 1991.
167. D. M. Post. The long and short of food-chain length. *Trends in Ecology & Evolution*, 17:269–277, 2002.
168. F. W. Preston. The commonness, and rarity, of species. *Ecology*, 29:254–283, 1948.
169. F. W. Preston. Time and space and variation of variation. *Ecology*, 41:611–627, 1960.
170. F. W. Preston. The canonical distribution of commonness and rarity: part I. *Ecology*, 43:185–215., 1962.
171. S. Pueyo, F. He, and T. Zillio. The maximum entropy formalism and the idiosyncratic theory of biodiversity. *Ecology Letters*, 10:1017–1028, 2007.

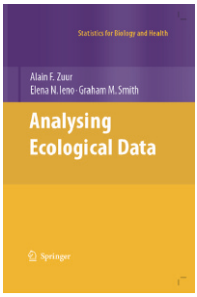
172. H. R. Pulliam. Sources, sinks, and population regulation. *The American Naturalist*, 132:652–661, 1988.
173. R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, version 2.8.1 edition, 2008.
174. D. Rabinowitz and J. K. Rapp. Dispersal abilities of seven sparse and common grasses from a Missouri prairie. *American Journal of Botany*, 68:616–624, 1981.
175. H. L. Reynolds and S. W. Pacala. An analytical treatment of root-to-shoot ratio and plant competition for soil nutrient and light. *American Naturalist*, 141:51–70, 1993.
176. J. F. Riebesell. Paradox of enrichment in competitive systems. *Ecology*, 55:183–187, 1974.
177. M. L. Rosenzweig. Why the prey curve has a hump. *American Naturalist*, 103:81–87, 1969.
178. M. L. Rosenzweig. *Species Diversity in Space and Time*. Cambridge University Press, Cambridge, 1995.
179. M.L. Rosenzweig. Paradox of enrichment: destabilization of exploitation ecosystems in ecological time. *Science*, 171:385–387, 1971.
180. M.L. Rosenzweig and R.H. MacArthur. Graphical representation and stability conditions of predator-prey interactions. *American Naturalist*, 97:209–223, 1963.
181. J. Roughgarden. *Primer of Ecological Theory*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.
182. J. Roughgarden and F. Smith. Why fisheries collapse and what to do about it. *Proceedings of the National Academy of Sciences, U.S.A.*, 93:5078–5083, May 1996.
183. P. F. Sale. The maintenance of high diversity in coral reef fish communities. *The American Naturalist*, 111:337–359, 1977.
184. M. Scheffer, S. Szabo, A. Gragnani, E. H. van Nes, S. Rinaldi, N. Kautsky, J. Norberg, R. M. M. Roijackers, and R. J. M. Franken. Floating plant dominance as a stable state. *Proceeding of the National Academy of Sciences, U.S.A.*, 100:4040–4045, 2003.
185. O. J. Schmitz. Perturbation and abrupt shift in trophic control of biodiversity and productivity. *Ecology Letters*, 7:403–409, 2004.
186. A. Schröder, L. Persson, and A. M. De Roos. Direct experimental evidence for alternative stable states: A review. *Oikos*, 110:3–19, 2005.
187. A. Shmida and S. P. Ellner. Coexistance of plant species with similar niches. *Vegetatio*, 58:29–55, 1984.
188. R. M. Sibly, D. Barker, M. C. Denham, J. Hone, and M. Pagel. On the regulation of populations of mammals, birds, fish, and insects. *Science*, 309:607–610, 2005.
189. J. G. Skellam. Random dispersal in theoretical populations. *Biometrika*, 38:196–218, 1951.
190. K. Soetaert, T. Petzoldt, and R. W. Setzer. *deSolve: General solvers for ordinary differential equations (ODE) and for differential algebraic equations (DAE)*. R package version 1.2-3.
191. N. C. Stenseth, W. Falck, O. N. Bjornstad, and C. J. Krebs. Population regulation in snowshoe hare and Canadian lynx: Asymmetric food web configurations between hare and lynx. *Proceedings of the National Academy of Sciences of the United States of America*, 94:5147–5152, 1997.
192. P. A. Stephens and W. J. Sutherland. Consequences of the allee effect for behaviour, ecology and conservation. *Trends in Ecology & Evolution*, 14:401–405, 1999.

193. R. W. Sterner and J. J. Elser. *Ecological Stoichiometry: The Biology of Elements From Molecules to the Biosphere*. Princeton University Press, Princeton, N.J., 2002.
194. M. H. H. Stevens, O. L. Petchey, and P. E. Smouse. Stochastic relations between species richness and the variability of species composition. *Oikos*, 103:479–488, 2003.
195. M. H. H. Stevens and C. E. Steiner. Effects of predation and nutrient enrichment on a food web with edible and inedible prey. *Freshwater Biology*, 51:666–671, 2006.
196. G. Sugihara. Minimal community structure: an explanation of species abundance patterns. *The American Naturalist*, 116:770–787, 1980.
197. K. S. Summerville and T. O. Crist. Determinants of lepidopteran community composition and species diversity in eastern deciduous forests: roles of season, eco- region and patch size. *Oikos*, 100:134–148, 2003.
198. K. S. Summerville and T. O. Crist. Contrasting effects of habitat quantity and quality on moth communities in fragmented landscapes. *Ecography*, 27:3–12, 2004.
199. R. M. Thompson, M. Hemberg, B. M. Starzomski, and J. B. Shurin. Trophic levels and trophic tangles: The prevalence of omnivory in real food webs. *Ecology*, 88:612–617, 2007.
200. D. Tilman. *Resource Competition and Community Structure*. Monographs in Population Biology. Princeton University Press, Princeton, NJ, 1982.
201. D. Tilman. *Plant Strategies and the dynamics and structure of plant communities*, volume 26 of *Monographs in Population Biology*. Princeton University Press, Princeton, NJ, 1988.
202. D. Tilman. Competition and biodiversity in spatially structured habitats. *Ecology*, 75:2–16, 1994.
203. D. Tilman, R. M. May, C. L. Lehman, and M. A. Nowak. Habitat destruction and the extinction debt. *Nature*, 371:65–66, 1994.
204. David Tilman. Resource competition and plant traits: A response to Craine et al. 2005. *Journal of Ecology*, 95:231–234, 2007.
205. M. Tokeshi. Niche apportionment or random assortment: species abundance patterns revisited. *Journal of Animal Ecology*, 59:1129–1146, 1990.
206. M. Tokeshi. *Species Coexistence: Ecological and Evolutionary Perspectives*. Blackwell Science, Ltd, Oxford, UK, 1999.
207. J. Vandermeer. Omnivory and the stability of food webs. *Journal of Theoretical Biology*, 238:497–504, 2006.
208. J. Vandermeer. Oscillating populations and biodiversity maintenance. *Bioscience*, 56:967–975, 2006.
209. J. Vandermeer, I. de la Cerda, I. G.and Perfecto, D. Boucher, J. Ruiz, and A. Kaufmann. Multiple basins of attraction in a tropical forest: Evidence for nonequilibrium community structure. *Ecology*, 85:575–579, 2004.
210. J. Vandermeer and P. Yodzis. Basin boundary collision as a model of discontinuous change in ecosystems. *Ecology*, 80:1817–1827, 1999.
211. M. J. Vanni, W. H. Renwick, J. L. Headworth, J. D. Auch, and M. H. Schaus. Dissolved and particulate nutrient flux from three adjacent agricultural watersheds: A five-year study. *Biogeochemistry*, 54:85–114, 2001.
212. J.A. Veech and T.O. Crist. Partition: software for hierarchical additive partitioning of species diversity, version 2.0. http:// www.users.muohio.edu/ cristto/ partition.htm, 2007.
213. M. Vellend and M. A. Geber. Connections between species diversity and genetic diversity. *Ecology Letters*, 8:767–781, 2005.

214. I. Volkov, J. R. Banavar, F. L. He, S. P. Hubbell, and A. Maritan. Density dependence explains tree species abundance and diversity in tropical forests. *Nature*, 438:658–661, 2005.
215. I. Volkov, J. R. Banavar, S. P. Hubbell, and A. Maritan. Neutral theory and relative species abundance in ecology. *Nature*, 424:1035–1037, 2003.
216. I. Volkov, J. R. Banavar, S. P. Hubbell, and A. Maritan. Patterns of relative species abundance in rainforests and coral reefs. *Nature*, 450:45–49, 2007.
217. R. R. Warner and P. L. Chesson. Coexistence mediated by recruitment fluctuations: a field guide to the storage effect. *The American Naturalist*, 125:769–787, 1985.
218. D. Wedin and D. Tilman. Competition among grasses along a nitrogen gradient: initial conditions and mechanisms of competition. *Ecological Monographs*, 63:199–229, 1993.
219. R. H. Whittaker. Vegetation of the siskiyou mountains, oregon and california. *Ecological Monographs*, 30:279–338, 1960.
220. J. A. Wiens. Spatial scaling in ecology. *Functional Ecology*, 3:385–397, 1989.
221. R. J. Williams and N. D. Martinez. Simple rules yield complex food webs. *Nature*, 404:180–182, 2000.
222. S. D. Wilson and D. Tilman. Competitive responses of 8 old-field plant-species in 4 environments. *Ecology*, 76:1169–1180, 1995.
223. J. T. Wootton. Field parameterization and experimental test of the neutral theory of biodiversity. *Nature*, 433:309–312, 2005.

Index

Analysing Ecological Data

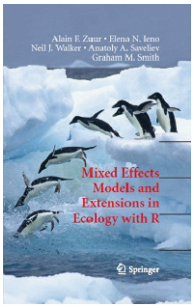


Alain F. Zuur

This book provides a practical introduction to analysing ecological data using real data sets collected as part of postgraduate ecological studies or research projects.
The first part of the book gives a largely non-mathematical introduction to data exploration, univariate methods (including GAM and mixed modelling techniques), multivariate analysis, time series analysis (e.g. common trends) and spatial statistics. The second part provides 17 case studies, mainly written together with biologists who attended courses given by the first authors.

2007. XXVI, 672 p. (Statistics for Biology and Health) Hardcover
ISBN 978-0-387-45967-7

Mixed Effects Models and Extensions in Ecology with R



Alain F. Zuur, Elena N. Ieno, Neil J. Walker
Anatoly A. Saveliev, and Graham M. Smith

The first part of the book is a largely non-mathematical introduction to linear mixed effects modelling, GLM and GAM, zero inflated models, GEE, GLMM and GAMM. The second part provides ten case studies that range from koalas to deep sea research. These chapters provide an invaluable insight into analysing complex ecological datasets, including comparisons of different approaches to the same problem. By matching ecological questions and data structure to a case study, these chapters provide an excellent starting point to analysing your own data.

2009. Approx 530 p., Hardcover
ISBN 978-0-387-87457-9

Introductory Time Series with R

Paul S.P. Cowpertwait, Andrew Metcalfe

This book gives you a step-by-step introduction to analysing time series using the open source software R. The book is written for undergraduate students of mathematics, economics, business and finance, geography, engineering and related disciplines, and postgraduate students who may need to analyse time series as part of their taught programme or their research.

2009. Approx. 280 p. (Use R) Softcover
ISBN: 978-0-387-88697-8